Trusted Firmware-A

Release 2.15.0

Trusted Firmware-A contributors

Jun 22, 2026

CONTENTS

1 About 1

2 Getting Started 64

3 Processes & Policies 112

4 Components 153

5 System Design 347

6 Porting Guide 494

7 Platform Ports 566

8 Performance & Testing 732

9 Security Advisories 750

10 Design Documents 781

11 Threat Model 847

12 Architecture Features 921

13 Tools 930

14 Change Log & Release Notes 943

15 Glossary 1263

16 License 1269

17 Getting Started 1273

CHAPTER

ONE

ABOUT

1.1 Feature Overview

This page provides an overview of the current *TF-A* feature set. For a full description of these features and their implementation details, please see the documents that are part of the *Components* and *System Design* chapters.

The *Change Log & Release Notes* provides details of changes made since the last release.

1.1.1 Current features

- Initialization of the secure world, for example exception vectors, control registers and interrupts for the platform.
- Library support for CPU specific reset and power down sequences. This includes support for errata workarounds and the latest Arm DynamIQ CPUs.
- Drivers to enable standard initialization of Arm System IP, for example Generic Interrupt Controller (GIC), Cache Coherent Interconnect (CCI), Cache Coherent Network (CCN), Network Interconnect (NIC) and TrustZone Controller (TZC).
- Secure Monitor library code such as world switching, EL2/EL1 context management and interrupt routing.
- SMC (Secure Monitor Call) handling, conforming to the SMC Calling Convention using an EL3 runtime services framework.
- *PSCI* library support for CPU, cluster and system power management use-cases. This library is pre-integrated with the AArch64 EL3 Runtime Software, and is also suitable for integration with other AArch32 EL3 Runtime Software, for example an AArch32 Secure OS.
- A generic *SCMI* driver to interface with conforming power controllers, for example the Arm System Control Processor (SCP).
- A minimal AArch32 Secure Payload (*SP_MIN*) to demonstrate *PSCI* library integration with AArch32 EL3 Runtime Software.
- Secure partition manager dispatcher(SPMD) with following two configurations:
  - S-EL2 SPMC implementation, widely compliant with FF-A v1.1 EAC0 and initial support of FF-A v1.2.

  - EL3 SPMC implementation, compliant with a subset of FF-A v1.1 EAC0.
- Support for Arm CCA based on FEAT_RME which supports authenticated boot and execution of RMM with the necessary routing of RMI commands as specified in RMM Beta 0 Specification.
- A Test SP and SPD to demonstrate AArch64 Secure Monitor functionality and SP interaction with PSCI.
- SPDs for the OP-TEE Secure OS, NVIDIA Trusted Little Kernel, Trusty Secure OS and ProvenCore Secure OS.
- A Trusted Board Boot implementation, conforming to all mandatory TBBR requirements. This includes image authentication, Firmware recovery, Firmware encryption and packaging of the various firmware images into a Firmware Image Package (FIP).
- Measured boot support with PoC to showcase its interaction with firmware TPM (fTPM) service implemneted on top of OP-TEE.
- Support for Dynamic Root of Trust for Measurement (DRTM).
- Following firmware update mechanisms available:
  - PSA Firmware Update (PSA FWU)
  - TBBR Firmware Update (TBBR FWU)
- Reliability, Availability, and Serviceability (RAS) functionality, including
  - A Secure Partition Manager (SPM) to manage Secure Partitions in Secure-EL0, which can be used to implement simple management and security services.
  - An *SDEI* dispatcher to route interrupt-based *SDEI* events.
  - An Exception Handling Framework (EHF) that allows dispatching of EL3 interrupts to their registered handlers, to facilitate firmware-first error handling.
- A dynamic configuration framework that enables each of the firmware images to be configured at runtime if required by the platform. It also enables loading of a hardware configuration (for example, a kernel device tree) as part of the FIP, to be passed through the firmware stages. This feature is now incorporated inside the firmware configuration framework (fconf).
- Support for alternative boot flows, for example to support platforms where the EL3 Runtime Software is loaded using other firmware or a separate secure system processor, or where a non-TF-A ROM expects BL2 to be loaded at EL3.
- Support for Errata management firmware interface.
- Support for the GCC, LLVM and Arm Compiler 6 toolchains.
- Support for combining several libraries into a “romlib” image that may be shared across images to reduce memory footprint. The romlib image is stored in ROM but is accessed through a jump-table that may be stored in read-write memory, allowing for the library code to be patched.
- Position-Independent Executable (PIE) support.

1.1.2 Experimental features

A feature is considered experimental when still in development or isn't known to the TF-A team as widely deployed or proven on end products. It is generally advised such options aren't pulled into real deployments, or done with the appropriate level of supplementary integration testing.

A feature is no longer considered experimental when it is generally agreed the said feature has reached a level of maturity and quality comparable to other features that have been integrated into products.

Experimental build options are found in following section *Experimental build options*. Their use through the build emits a warning message.

Additionally the following libraries are marked experimental when included in a platform:

- RSE comms driver `drivers/arm/rse`
- GICv5 driver `drivers/arm/gicv5` via `USE_GIC_DRIVER=5`
- FEAT_IDTE3 `lib/extensions/idte/` via `ENABLE_FEAT_IDTE3`

1.1.3 Still to come

- Support for additional platforms.
- Documentation enhancements.
- Ongoing support for new architectural features, CPUs and System IP.
- Ongoing support for new Arm system architecture specifications.
- Ongoing security hardening, optimization and quality improvements.

1.2 Release Processes

1.2.1 Project Release Cadence

The project currently aims to do a release once every 6 months which will be tagged on the master branch. There will be a code freeze (stop merging non-essential changes) up to 4 weeks prior to the target release date. The release candidates will start appearing after this and only bug fixes or updates required for the release will be merged. The maintainers are free to use their judgement on what changes are essential for the release. A release branch may be created after code freeze if there are significant changes that need merging onto the integration branch during the merge window.

The release testing will be performed on release candidates and depending on issues found, additional release candidates may be created to fix the issues.

```
                        |<----------6 months---------->|
      |<---4 weeks--->|                |<---4 weeks--->|
+-----------------------------------------------------------> time
      |                 |                |                 |
  code freeze        ver w.x       code freeze       ver y.z
```

Version numbering

TF-A version is given in Makefile, through several macros:

- VERSION_MAJOR
- VERSION_MINOR
- VERSION_PATCH

For example, TF-A v2.10 has VERSION_MAJOR=2, VERSION_MINOR=10 and VERSION_PATCH=0.

This VERSION_PATCH macro is only increased for *LTS - Long-Term Support* releases.

Upcoming Releases

These are the estimated dates for the upcoming release. These may change depending on project requirement and partner feedback.

Release Version	Target Date	Expected Code Freeze
v2.0	1st week of Oct '18	1st week of Sep '18
v2.1	5th week of Mar '19	1st week of Mar '19
v2.2	4th week of Oct '19	1st week of Oct '19
v2.3	4th week of Apr '20	1st week of Apr '20
v2.4	2nd week of Nov '20	4th week of Oct '20
v2.5	3rd week of May '21	5th week of Apr '21
v2.6	4th week of Nov '21	2nd week of Nov '21
v2.7	5th week of May '22	3rd week of May '22
v2.8	5th week of Nov '22	3rd week of Nov '22
v2.9	4th week of May '23	2nd week of May '23
v2.10	4th week of Nov '23	2nd week of Nov '23
v2.11	4th week of May '24	2nd week of May '24
v2.12	4th week of Nov '24	2nd week of Nov '24
v2.13	4th week of May '25	2nd week of May '25
v2.14	4th week of Nov '25	2nd week of Nov '25
v2.15	4th week of May '26	2nd week of May '26

1.2.2 Removal of Deprecated Interfaces

As mentioned in the *Platform Ports Policy*, this is a live document cataloging all the deprecated interfaces in TF-A project and the Release version after which it will be removed.

Interface	Deprecation Date	Removed after Release	Comments

1.2.3 Removal of Deprecated Drivers

As mentioned in the *Platform Ports Policy*, this is a live document cataloging all the deprecated drivers in TF-A project and the Release version after which it will be removed.

Driver	Deprecation Date	Removed after Release	Comments
None at this time.			

1.2.4 Build Options deprecated/removed

Populated table provides details about build options that were removed or deprecated.

Build Option	Deprecated from TF-A Version	Removed after Release
NS_TIMER_SWITCH	v2.14.0	v2.16.0
SPM_MM	v2.14.0	v2.16.0
ENABLE_RME	v2.15.0	v2.17.0

1.3 LTS - Long-Term Support

Table 1: Table 1: Document History

Date	Author	Description
2022-07-20	Okash Khawaja, Varun Wadekar	Initial draft.
2022-07-21	Varun Wadekar	Refine the Maintainership guidelines and planning sections. Introduce a new section documenting a day in the life of a LTS branch maintainer
2022-08-05	Okash Khawaja, Varun Wadekar	Merge two drafts (draft 1 and 2), address comments made by both authors, cosmetic changes to the content all over the document
2022-08-05	Okash Khawaja	Add note about testing support available from TF.org
2022-08-05	Varun Wadekar	Changed the "Future plans" section to "FAQ" and answered some of the questions with feedback from the community.
2025-01-07	Govindraj Raja	Convert from pdf to rst.
2025-01-07	Govindraj Raja	Updates based on learnings and suggestions.
2025-03-27	Chris Palmer	Playbook for making a new release.

This document proposes a plan for long-term support (LTS) of the *TF-A* project.

1.3.1 Why is LTS required?

LTS is needed for commercial reasons. More specifically, on the device side, when a product is released, the companies have to support that in-market product such that the amount of changes to the firmware are kept to a minimum to avoid the risk of regression. At the same time the companies don't want to exclude critical patches such as those for security advisories. Similarly on the server side, companies want to minimize the churn when deploying fixes during incident response, e.g. due to critical security bugs.

Also, the European Cyber Resilience Act (CRA) is a new EU legislation that mandates cybersecurity standards for products containing digital elements, aiming to protect consumers and businesses by ensuring manufacturers build security into their hardware and software throughout their lifecycle, including automatic updates and incident reporting; essentially requiring all digital products sold in the EU to meet specific cybersecurity requirements.

This means that companies have to maintain and backport critical updates to old branches internally. As this effort is duplicated across different companies using TF-A, it makes sense to factor out this effort into a community-wide LTS.

1.3.2 What does LTS mean for TF-A?

In this section we will define exactly what constitutes LTS for TF-A. Specifically, we will define the following characteristics:

- criteria for selecting patches which will be backported to LTS branches
- lifetime and frequency of LTS branches

Criteria

We must have an objective criterion for selecting patches to be backported to LTS branches. This will make maintenance easy because:

a. there will be less – ideally no – discussion when selecting patches to backport

b. large parts of the process can be automated

Below is the criteria

1. No features will be backported.
2. Security advisories: Any patch that makes it into *Security Advisories* is automatically selected for back porting. This includes patches to external components too, e.g. libfdt.
3. Workarounds for CPU and other ARM IP errata
4. Workarounds for non-ARM IP errata, e.g. TI UART
5. Fixes for platform bugs. These patches must not modify any code outside of the specific platform that the fix applies to.
6. Patches can only be backported from the master branch. In other words, the master branch will be a superset of all the changes in any LTS branch.

Lifetime and frequency

This section approaches three questions: for how long should an LTS release be supported, how frequently should LTS releases be made and at which time(s) of the year should the releases be made.

1. For how long should an LTS release be supported?

The Linux kernel maintainers supports an LTS branch for 2 years. Since firmware tends to have less churn and longer lifetime than a HLOS, TF-A is trying to support at-least 7 years for its LTS. Initially it was intended to support 5 years but there has been no objections to extend LTS support to 7 years. There are many challenges that may influence the 7 year support from CI infrastructure to availability of maintainers.

2. How frequently should LTS releases be made?

Given that many products that have a release cycle, have a yearly release cycle, it would make sense to have yearly TF-A releases.

3. Which time(s) of the year should the releases be made?

TF-A releases are cut twice a year: May and November. Basing LTS release on the November TF-A release has a few benefits. First, it aligns with Linux LTS releases which happen towards the end of each year. Second, it aligns with Android releases which tend to fall in Q3 each year. Since product releases are timed with Android release, this gives enough time to harden the TF-A LTS release during development so that it's ready for launch in Q3 following year. On the other hand, if the May release of TF-A is chosen as the basis for LTS then developers will have little time – about a month, taking into account the test-and-debug phase before LTS is cut (see below) – before Android release.

To summarize, there will be one LTS release per year. It will be supported for 5 years and we can discuss extending it to 7 years later on. The LTS release will be based on the November release of TF-A.

Testing Criteria

Every patch merged to the LTS branch will complete the following tests before getting approved.

1. TFTF tests currently running in the testing farm
2. CI/CD static analysis scans
3. Coverity scans
4. Platform tests

Platforms that are not maintained upstream will undergo testing downstream in a pre-defined window. The platform maintainer will complete the testing and provide a verified score on the patch once testing is completed.

** A note about test coverage from TF.org **

Currently TF.org maintains a CI system to run TF-A automated tests on a selection of HW boards donated by TF.org members (a benefit reserved to project members, see the project charter for more details). This automated test coverage will be extended to cover testing for LTS as well for boards that are part of the CI system.

TFTF Branching

A note about testing here. After a patch is backported to an LTS branch, that branch will need to be regression tested. Since TFTF moves forward with latest TF-A changes, newer TFTF tests may not apply to old LTS branches. Therefore TFTF will also need to be branched, in-sync with TF-A LTS branches. In other words, there will be one TFTF LTS branch corresponding to each TF-A LTS branch. The TFTF LTS branch will be used to regression test the corresponding TF-A LTS branch.

As we work with the LTS branch of TFTF, we might also need fixes for TFTF itself to be ported to LTS. However, decision-making about those patches need not be as stringent as for TF-A.

CI Scripts

CI Scripts moves forward with TF-A changes, since we need to checkout the corresponding release version of CI scripts for LTS.

Though we are unlikely to update CI scripts, but time to time migrating a newer FVP version or deprecating certain tests due to unavailability of platforms may influence updates to CI Scripts.

Hafnium / OP-TEE

Both Hafnium and OP-TEE move forward with TF-A changes so we need to freeze their corresponding version from TF-A release for a LTS.

MbedTLS

Updates to the version of MbedTLS used with LTS will happen time to time based on maintainers call to update them or not.

1.3.3 Release details

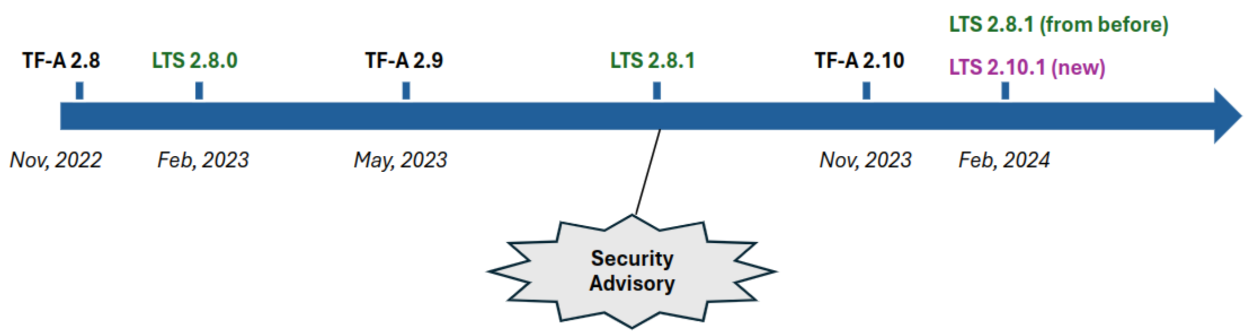
This section goes into details of what the LTS release process will look like.

Test-and-debug period

Since the LTS branch will be used in product releases, it is expected that more testing and debugging will be done on the November release of TF-A. Therefore it would make sense to leave at least a month after the November release and then cut the LTS branch. We recommend two months, given that one of the months is December which tends to be slower due to holidays. So, an end-of-November TF-A release would result in a beginning-of-February LTS release. Note that the LTS branch will be created at the same time as the TF-A November release, but it will be officially released at the end of January or early February. Going forward we should strive to make the period smaller and smaller until ideally it coincides with TF-A November release which means that our test and CI/CD infra is good enough to allow that to happen.

Example timeline

Below is an example timeline starting from the November 2022 release of TF-A.



- Nov 2022: TF-A 2.8 is released towards the end of Nov, 2022. Not shown in the diagram, at the same time LTS release candidate branch is made which is based on TF-A 2.8. This means new features going in 2.8 won't go in the LTS branch. We can call it *LTS 2.8-rc*.
- Feb 2023: After testing and debugging LTS 2.8-rc for a couple of months, LTS 2.8.0 is officially released in early Feb 2023.
- May 2023: TF-A 2.9 is released but since this is not an LTS branch it doesn't affect LTS.
- Somewhere between May and Nov of 2023: A security advisory comes up and the related patches go into TF-A master branch. Since these patches fall under LTS criteria, they are backported to LTS 2.8.0 which results in LTS 2.8.1 being released. Note that here we don't allow the extra testing and debugging time that we had between Nov 2022 and early Feb 2023. This is because there isn't as much to test and debug as an annual LTS release has. Also companies might want to deploy critical patches soon.
- Nov 2023: TF-A 2.10 is released. Not shown in the diagram, at the same time LTS 2.10-rc is made. It's tested by partners for a couple of months.
- Feb 2024: LTS 2.10.1 is released in early Feb. Now there are two LTS branches: 2.8.1 and 2.10.1.

Note that TFTF will follow similar branching model as TF-A LTS, i.e. there will be TFTF LTS 2.8.0 in Feb 2023, 2.8.1 (if new TFTF tests need to be added for the security advisory) when there is TF-A LTS 2.8.1 and so on.

1.3.4 Maintainership

Guidelines & Responsibilities

1. Maintainers shall be impartial and strive to work for the benefit of the community
2. Objective and well-defined merge criteria to avoid confusion and discussions at random points in time when there is a "candidate" patch
3. The maintainers shall explain the lifecycle of a patch to the community, with a detailed description of the maximum time spent in each step
4. Automate, automate, automate
5. Reviewers should not focus too much on "what" and instead focus on "how"
6. Constantly refine the merge criteria to include more partner use cases
7. Ensure that all candidate patches flow from the main branch to all LTS branches
8. Maintainers collaborate in the following discord channel - https://discord.com/channels/1106321706588577904/1162029539761852436
9. Maintainers discuss and provide updates about upcoming LTS releases in the above mentioned discord channel.

Options

These are some options in the order of preference.

1. Current set of *LTS Maintainers* from tf.org(or hired contractor) take care of the LTS
2. From the community, create a set of maintainers focused solely on the LTS branches

A day in the life of a maintainer

This section documents the daily tasks that a maintainer might perform to support the LTS program. It is expected that a maintainer follows clearly laid down steps and does not have to make policy level decisions for merge, testing, or candidate patch selection.

1. Monitor the main branch to identify candidate patches for the LTS branches
2. Monitor emails from LTS triage report to choose patches that should be cherry-picked for LTS branches.
3. Cherry-pick agreed patches to LTS branches co-ordinate review process and Monitor CI results.
4. Monitor the mailing list for any LTS related issues
5. Propose or solicit patches to the main branch and tag them as candidates for LTS
6. Monitor Github dependabot pull requests to identify patches that could be taken for a given LTS branch: https://github.com/TrustedFirmware-A/trusted-firmware-a/pulls

1.3.5 Playbook for new releases

To make a new minor release (e.g. 2.x.y → 2.x.y+1), follow these steps.

1. Every Friday, LTS maintainers receive a triage report email (subject: “TF-A LTS Triage report”) that contains attached CSV files, 1 per currently-supported LTS major release branch (e.g. lts-2.8, lts-2.10, lts-2.12, etc.). It contains a list of patches to be cherry-picked into a new minor release of each supported LTS branch.
2. Run `git fetch origin`.
3. Run `git checkout -b lts-v2.x.y+1 --track origin/lts-v2.x`.
4. Run `git log` and verify that the most recent commit is the changelog for the v2.x.y release, and that it has the origin/lts-v2.x tag.
5. For the version 2.x for which you want to create a new release, open its CSV file. For each patch listed, **from the bottom to the top**, run `git cherry-pick -x sha1-hash`.
6. Some of the patches of this list may not be taken, mainly due to false positive. If in doubt, that can be discussed either in the “tf-a-lts” channel on Discord or during the LTS weekly meeting.
7. Some dependency patches, not listed in the CSV file, may have to be taken, to ease the application of the LTS patches. This can also be discussed with the other LTS maintainers.
8. Run `git remote add dependabot https://github.com/TrustedFirmware-A/trusted-firmware-a.git` (This has to be done only once).
9. Run `git fetch dependabot`.
10. Cherry-pick the dependabot patches dedicated to the given LTS. Those patches should be amended to add a gerrit Change ID.
11. Push the stack of changes: `git push origin HEAD:refs/for/lts-v2.x%topic=for-lts-v2.x.y+1`. You might need the `--no-verify` option: `git push origin --no-verify HEAD:refs/for/lts-v2.x%topic=for-lts-v2.x.y+1`.
12. The AllowCI+2 job runs automatically on each LTS branch once a new cherry-picked patch/patch-stack is pushed to the corresponding branch. If this CI run passes, it automatically applies the Verified+1 (V+1) label to the patch/all patches in the stack. The other LTS maintainers will provide MR+1 and COR+1 votes. If the CI is OK and votes V+1, and if the Maintainer-Review+1 (MR+1), Code-Owner-Review+1 (COR+1), and V+1 votes are present, Gerrit will automatically merge the patch.
13. Once the patch stack is merged, LTS maintainers will then manually trigger the Jenkins `tf-a-lts-create-release` job for the target LTS branch/release. This job generates the changelog and triggers the follow-up jobs `tf-a-lts-release-tagged`, `tf-a-lts-release-docs` and `tf-a-lts-release-mail`. These jobs tag the release, tag the documentation and send the release announcement email respectively.
14. Some features may also require updates in other repositories (tf-a-ci-scripts, tf-a-job-configs or tf-a-tests…). For tf-a-job-configs, there are no LTS branches, but dedicated scripts for each LTS version which have to be updated manually. This is the case for e.g. MbedTLS updates. For tf-a-ci-scripts and tf-a-tests, there are LTS branches and patches will be cherry-picked from master branch to the LTS branch the same way it is done for TF-A. There is no automation for those repositories. So the

patches will have to be merged manually, and for tf-a-ci-scripts and tf-a-tests, tags will also have to be set manually.

Execution Plan

This section lists the steps needed to put the LTS system in place. However, to kick start LTS in Nov '22, only a few steps are needed. The rest can follow in the background.

Initial release steps

The following steps are necessary to kickstart the project and potentially create the first LTS from the Nov'22 release.

1. Create a TF-A LTS release-candidate branch and a TFTF LTS branch immediately after the Nov'22 release
2. Request all platform-owners to test and debug the RC branch
3. Gather feedback from the test and debug cycle
4. Mark the TF-A LTS branch ready by the end of January
5. Announce the official LTS release availability on the mailing lists

Long term release plan

Above will buy us time to then work on the rest of the execution plan which is given below.

1. The review criteria for LTS patches must be the same as TF-A patches
2. The maintainers shall publish the well-defined merge criteria to allow the community to choose candidate patches
3. The maintainers shall publish a well-defined test specification for any patch entering the LTS branch
   a. Tests required to pass in the CI/CD flow
   b. Static analysis scans
   c. Coverity scans
4. The maintainers shall publish a mechanism to choose candidate patches for the LTS branch
5. The maintainers shall publish a mechanism to report bugs *[1]* seen with an LTS branch
6. The maintainers shall publish a versioning mechanism for the LTS branch
   a. Bump minor version for any "logical" *[2]* fix(es) that gets merged
7. The CI/CD infrastructure shall provide test support for all "live" LTS branches at any given point in time
8. The CI/CD infrastructure shall provide means to
   a. notify all maintainers that a patch is ready for review

b. automatically cherry-pick a patch to a given LTS branch

c. get it through the CI/CD testing flow

d. gentle ping in LTS discord channel asking for reviews to ensure cherry-picks are merged.

FAQ

In our discussions, in addition to the above points we also considered some questions. They have been discussed on the mailing list too.

Q. What happens when a bug fix applies just to a LTS branch and not to the master branch?
A. This will be treated as a special case and the bug, and the fix will be discussed

Q. When testing a backported patch, what if one of the partners needs more time while the patch fix is time-critical and, hence slowing other partners?
A. The maintainers will add more detail to the review and merge process to handle this scenario.

Q. How do we handle the increasing version numbers for errata fixes?
A. Too many CPU errata workarounds resulting in too many LTS releases. We propose bumping the version number for each logical fix as described in the section "Long term release plan" above because that will help accurately track what changes have been deployed in-field.

Q. What if LTS support duration needs to be extended to longer than 5 years?
A. Still under discussion.

These are uncharted waters, and we will face some unseen problems. When they become real problems, then we will have concrete data and be better able to address them. This means that our LTS definition as presented in this document is not the final one. We will constantly be discussing it and deciding how to adapt it as we see practical problems.

[1] The plan is to create a system where reviewers can tag a patch on mainline which gets automatically rebased on LTS and pushed to Gerrit. On seeing this patch, the CI/CD starts tests and provides a score. In parallel, the system also sends an email to the maintainers announcing the arrival of a candidate patch for the LTS branch.

[2] Logical will be a patch or patches implementing a certain fix. For example, if a security mitigation is fixed with the help of three patches, then all of them are considered as one "logical" fix. The version is incremented only after all these patches are merged. with the maintainers. If agreed unanimously, the bug fix will be merged to the affected LTS branches after completing the review process.

1.4 Project Maintenance

Trusted Firmware-A (TF-A) is an open governance community project. All contributions are reviewed and merged by the community members listed below.

For more details on the roles of *maintainers*, *code owners* and general information about code reviews in TF-A project, please refer to the *Code Review Guidelines*.

1.4.1 Maintainers

Note: If you wish to become a maintainer for TF-A project, please refer to the *Project Maintenance Processes*.

Mail
Dan Handley <dan.handley@arm.com>

GitHub ID
danh-arm

Mail
Soby Mathew <soby.mathew@arm.com>

GitHub ID
soby-mathew

Mail
Sandrine Bailleux <sandrine.bailleux@arm.com>

GitHub ID
sandrine-bailleux-arm

Mail
Alexei Fedorov <Alexei.Fedorov@arm.com>

GitHub ID
AlexeiFedorov

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Mark Dykes <mark.dykes@arm.com>

GitHub ID
mardyk01

Mail
Olivier Deprez <olivier.deprez@arm.com>

GitHub ID
odeprez

Mail
Bipin Ravi <bipin.ravi@arm.com>

GitHub ID
bipinravi-arm

Mail
Joanna Farley <joanna.farley@arm.com>

GitHub ID
joannafarley-arm

Mail
Julius Werner <jwerner@chromium.org>

GitHub ID
jwerner-chromium

Mail
Varun Wadekar <vwadekar@nvidia.com>

GitHub ID
vwadekar

Mail
Andre Przywara <andre.przywara@arm.com>

GitHub ID
Andre-ARM

Mail
Lauren Wehrmeister <Lauren.Wehrmeister@arm.com>

GitHub ID
laurenw-arm

Mail
Madhukar Pappireddy <Madhukar.Pappireddy@arm.com>

GitHub ID
madhukar-Arm

Mail
Raghu Krishnamurthy <raghuoss@raghushome.com>

GitHub ID
raghuncstate

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Mail
Yann Gautier <yann.gautier@st.com>

GitHub ID
Yann-lms

Mail
Govindraj Raja <govindraj.raja@arm.com>

GitHub ID
govindraj-arm

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Mail
Harrison Mutai <harrison.mutai@arm.com>

GitHub ID
harrisonmutai-arm

1.4.2 LTS Maintainers

Mail
Bipin Ravi <bipin.ravi@arm.com>

GitHub ID
bipinravi-arm

Mail
Olivier Deprez <olivier.deprez@arm.com>

GitHub ID
odeprez

Mail
Jamie Fox <jamiefox@google.com>

GitHub ID
jf549

Mail
Varun Wadekar <vwadekar@nvidia.com>

GitHub ID
vwadekar

Mail
Yann Gautier <yann.gautier@st.com>

GitHub ID
Yann-lms

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

1.4.3 Code owners

Common Code

Armv7-A architecture port

Mail
Etienne Carriere <etienne.carriere@linaro.org>

GitHub ID
etienne-lms

Build Definitions for CMake Build System

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Files
/

Software Delegated Exception Interface (SDEI)

Mail
Jayanth Dodderi Chidanand <jayanthdodderi.chidanand@arm.com>

GitHub ID
jayanthchidanand-arm

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Files
services/std_svc/sdei/

Trusted Boot

Mail
Sandrine Bailleux <sandrine.bailleux@arm.com>

GitHub ID
sandrine-bailleux-arm

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Mail
Lauren Wehrmeister <Lauren.Wehrmeister@arm.com>

GitHub ID
laurenw-arm

Files
drivers/auth/

Secure Partition Manager Core (EL3 FF-A SPMC)

Mail
Marc Bonnici <marc.bonnici@arm.com>

GitHub ID
marcbonnici

Files
services/std_svc/spm/el3_spmc/*

Files
include/services/el3_spmc_*

Files
include/services/spmc_svc.h

Secure Partition Manager Dispatcher (SPMD)

Mail
Olivier Deprez <olivier.deprez@arm.com>

GitHub ID
odeprez

Mail
Madhukar Pappireddy <Madhukar.Pappireddy@arm.com>

GitHub ID
madhukar-Arm

Files
services/std_svc/spmd/*

Files
plat/common/plat_spmd_manifest.c

Files
include/services/ffa_svc.h

Files
include/services/el3_spmd_logical_sp.h

Files
include/services/spmd_svc.h

Exception Handling Framework (EHF)

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
bl31/ehf.c

Runtime Exceptions and Interrupt Management

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
bl31/aarch64/

Files
bl31/interrupt_mgmt.c

Files
include/bl31/interrupt_mgmt.h

Realm Management Monitor Dispatcher (RMMD)

Mail
Javier Almansa Sobrino <javier.almansasobrino@arm.com>

GitHub ID
javieralso-arm

Mail
Alexei Fedorov <Alexei.Fedorov@arm.com>

GitHub ID
AlexeiFedorov

Files
services/std_svc/rmmd/*

Files
include/services/rmmd_svc.h

Files
include/services/rmm_core_manifest.h

Realm Management Extension (RME)

Mail
Javier Almansa Sobrino <javier.almansasobrino@arm.com>

GitHub ID
javieralso-arm

Mail
Alexei Fedorov <Alexei.Fedorov@arm.com>

GitHub ID
AlexeiFedorov

Drivers, Libraries and Framework Code

Activity Monitors Unit (AMU) extensions

Mail
Alexei Fedorov <Alexei.Fedorov@arm.com>

GitHub ID
AlexeiFedorov

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
lib/extensions/amu/

Arm CPU libraries

Mail
Bipin Ravi <bipin.ravi@arm.com>

GitHub ID
bipinravi-arm

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
: loosper-arm

Files
: lib/cpus/

Files
: include/lib/cpus/

Arm® Ethos™-N NPU driver

Mail
: Joshua Slater <joshua.slater@arm.com>

GitHub ID
: jslater8

Mail
: Ştefana Simion <stefana.simion@arm.com>

GitHub ID
: stefanasimion

Files
: drivers/arm/ethosn/

Files
: include/drivers/arm/ethosn.h

Files
: include/drivers/arm/ethosn_cert.h

Files
: include/drivers/arm/ethosn_fip.h

Files
: include/drivers/arm/ethosn_oid.h

Files
: plat/arm/board/juno/juno_ethosn_tzmp1_def.h

Files
: plat/arm/common/fconf/fconf_ethosn_getter.c

Files
: include/plat/arm/common/fconf_ethosn_getter.h

Files
: fdts/juno-ethosn.dtsi

Clock Framework

Mail
Gabriel Fernandez <gabriel.fernandez@st.com>

GitHub ID
Gabriel-Fernandz

Files
drivers/clk

Files
include/drivers/clk.h

Console API framework

Mail
Julius Werner <jwerner@chromium.org>

GitHub ID
jwerner-chromium

Files
drivers/console/

Files
include/drivers/console.h

Files
plat/common/aarch64/crash_console_helpers.S

Context Management

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
bl1/aarch32/bl1_context_mgmt.c

Files
bl1/aarch64/bl1_context_mgmt.c

Files
bl31/bl31_context_mgmt.c

Files
lib/el3_runtime/

Files
include/lib/el3_runtime/

coreboot support libraries

Mail
Julius Werner <jwerner@chromium.org>

GitHub ID
jwerner-chromium

Files
drivers/coreboot/

Files
include/drivers/coreboot/

Files
include/lib/coreboot.h

Files
lib/coreboot/

DebugFS

Mail
Olivier Deprez <olivier.deprez@arm.com>

GitHub ID
odeprez

Files
lib/debugfs/

Files
include/lib/debugfs.h

DRTM

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Files
services/std_svc/drtm

Files
include/plat/common/plat_drtm.h

Files
include/services/drtm_svc.h

eMMC/UFS drivers

Mail
Haojian Zhuang <haojian.zhuang@linaro.org>

GitHub ID
hzhuang1

Files
drivers/partition/

Files
drivers/synopsys/emmc/

Files
drivers/synopsys/ufs/

Files
drivers/ufs/

Files
include/drivers/dw_ufs.h

Files
include/drivers/ufs.h

Files
include/drivers/synopsys/dw_mmc.h

Errata Management

Mail
Bipin Ravi <bipin.ravi@arm.com>

GitHub ID
bipinravi-arm

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Firmware Configuration Framework (FCONF)

Mail
Madhukar Pappireddy <Madhukar.Pappireddy@arm.com>

GitHub ID
madhukar-Arm

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Mail
Lauren Wehrmeister <Lauren.Wehrmeister@arm.com>

GitHub ID
laurenw-arm

Files
lib/fconf/

Files
plat/arm/common/fconf/

Files
include/lib/fconf/

Files
include/plat/arm/common/arm_fconf_*

Files
include/plat/arm/common/fconf_*

Firmware Encryption Framework

Mail
Sumit Garg <sumit.garg@kernel.org>

GitHub ID
b49020

Files
drivers/io/io_encrypted.c

Files
include/drivers/io/io_encrypted.h

Files
include/tools_share/firmware_encrypted.h

Firmware Handoff Library (Transfer List)

Mail
Raymond Mao <raymond.mao@linaro.org>

GitHub ID
raymo200915

Mail
Harrison Mutai <harrison.mutai@arm.com>

GitHub ID
harrisonmutai-arm

Files
lib/transfer_list

Files
include/lib/transfer_list.h

GIC driver

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Madhukar Pappireddy <Madhukar.Pappireddy@arm.com>

GitHub ID
madhukar-Arm

Mail
Olivier Deprez <olivier.deprez@arm.com>

GitHub ID
odeprez

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
drivers/arm/gic/

GICv5 driver

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
drivers/arm/gicv5

Files
plat/common/plat_gicv5.c

Files
plat/arm/board/fvp/gicv5_config.yaml

Granule Protection Tables Library (GPT-RME)

Mail
Soby Mathew <soby.mathew@arm.com>

GitHub ID
soby-mathew

Mail
Javier Almansa Sobrino <javier.almansasobrino@arm.com>

GitHub ID
javieralso-arm

Files
lib/gpt_rme

Files
include/lib/gpt_rme

IO abstraction layer

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Olivier Deprez <olivier.deprez@arm.com>

GitHub ID
odeprez

Files
drivers/io/

JTAG DCC console driver

Mail
Michal Simek <michal.simek@amd.com>

GitHub ID
michalsimek

Mail
Akshay Belsare <akshay.belsare@amd.com>

GitHub ID
Akshay-Belsare

Mail
Maheedhar Sai Bollapalli <maheedharsai.bollapalli@amd.com>

GitHub ID
maheedhar-bollapalli

Files
drivers/arm/dcc/

Files
include/drivers/arm/dcc.h

Libfdt wrappers

Mail
Madhukar Pappireddy <Madhukar.Pappireddy@arm.com>

GitHub ID
madhukar-Arm

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Files
common/fdt_wrappers.c

Library At ROM (ROMlib)

Mail
Madhukar Pappireddy <Madhukar.Pappireddy@arm.com>

GitHub ID
madhukar-Arm

Files
lib/romlib/

Live Firmware Activation Service

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Files
services/std_svc/lfa

Files
include/plat/common/plat_lfa.h

Files
include/services/lfa_svc.h

Max Power Mitigation Mechanism (MPMM)

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Files
include/lib/cpus/aarch64/cpu_macros.S

Measured Boot

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Mail
Harrison Mutai <harrison.mutai@arm.com>

GitHub ID
harrisonmutai-arm

Files
drivers/measured_boot

Files
include/drivers/measured_boot

Files
docs/components/measured_boot

Files
plat/arm/board/fvp/fvp*_measured_boot.c

Memory Partitioning And Monitoring (MPAM) extensions

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
lib/extensions/mpam/

Message Handling Unit (MHU) driver

Mail
David Vincze <david.vincze@arm.com>

GitHub ID
davidvincze

Files
include/drivers/arm/mhu.h

Files
drivers/arm/mhu

Per-CPU Framework

Mail
Rohit Mathew <rohit.mathew@arm.com>

GitHub ID
rohit-arm

Mail
Sammit Joshi <sammit.joshi@arm.com>

GitHub ID
sammit-arm

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Files
include/lib/per_cpu

Files
lib/per_cpu

Files
docs/components/numa-per-cpu.rst

Performance Measurement Framework (PMF)

Mail
Harrison Mutai <harrison.mutai@arm.com>

GitHub ID
harrisonmutai-arm

Files
lib/pmf/

Platform Security Architecture (PSA) APIs

Mail
Sandrine Bailleux <sandrine.bailleux@arm.com>

GitHub ID
sandrine-bailleux-arm

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Files
include/lib/psa

Files
lib/psa

Pointer Authentication (PAuth) and Branch Target Identification (BTI) extensions

Mail
Alexei Fedorov <Alexei.Fedorov@arm.com>

GitHub ID
AlexeiFedorov

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
lib/extensions/pauth/

Power State Coordination Interface (PSCI)

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
lib/psci/

Files
include/lib/psci/

PSA Firmware Update

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Mail
Sandrine Bailleux <sandrine.bailleux@arm.com>

GitHub ID
sandrine-bailleux-arm

Files
drivers/fwu

Files
include/drivers/fwu

Reliability Availability Serviceabilty (RAS) framework

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Olivier Deprez <olivier.deprez@arm.com>

GitHub ID
: odeprez

Mail
: Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
: loosper-arm

Files
: lib/extensions/ras/

Runtime Security Engine (RSE) comms driver

Mail
: David Vincze <david.vincze@arm.com>

GitHub ID
: davidvincze

Files
: include/drivers/arm/rse_comms.h

Files
: drivers/arm/rse

Runtime Services

Mail
: Manish Pandey <manish.pandey2@arm.com>

GitHub ID
: manish-pandey-arm

Mail
: Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
: loosper-arm

Files
: services/std_svc/std_svc_setup.c

Files
: common/runtime_svc.c

Files
: include/common/runtime_svc.h

Files
: include/services/arm_arch_svc.h

Files
include/services/std_svc.h

Standard C library

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Mail
Madhukar Pappireddy <Madhukar.Pappireddy@arm.com>

GitHub ID
madhukar-Arm

Files
lib/libc/

Statistical Profiling Extension (SPE)

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Boyan Karatotev <boyan.karatotev@arm.com>

GitHub ID
loosper-arm

Files
lib/extensions/spe/

System Control and Management Interface (SCMI) Server

Mail
Etienne Carriere <etienne.carriere@st.com>

GitHub ID
etienne-lms

Mail
Peng Fan <peng.fan@nxp.com>

GitHub ID
MrVan

Files
drivers/scmi-msg

Files
include/drivers/scmi*

Translation tables (`xlat_tables`) library

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Mail
Harrison Mutai <harrison.mutai@arm.com>

GitHub ID
harrisonmutai-arm

Files
lib/xlat_tables_*/

Platform Ports

Allwinner ARMv8 platform port

Mail
Andre Przywara <andre.przywara@arm.com>

GitHub ID
Andre-ARM

Mail
Samuel Holland <samuel@sholland.org>

GitHub ID
smaeul

Files
docs/plat/allwinner.rst

Files
plat/allwinner/

Files
drivers/allwinner/

Amlogic Meson S905 (GXBB) platform port

Mail
Andre Przywara <andre.przywara@arm.com>

GitHub ID
Andre-ARM

Files
docs/plat/meson-gxbb.rst

Files
drivers/amlogic/

Files
plat/amlogic/gxbb/

Amlogic Meson S905x (GXL) platform port

Mail
Remi Pommarel <repk@triplefau.lt>

GitHub ID
remi-triplefault

Files
docs/plat/meson-gxl.rst

Files
plat/amlogic/gxl/

Amlogic Meson S905X2 (G12A) platform port

Mail
Carlo Caione <ccaione@baylibre.com>

GitHub ID
carlocaione

Files
docs/plat/meson-g12a.rst

Files
plat/amlogic/g12a/

Amlogic Meson A113D (AXG) platform port

Mail
Carlo Caione <ccaione@baylibre.com>

GitHub ID
carlocaione

Files
docs/plat/meson-axg.rst

Files
plat/amlogic/axg/

Arm FPGA platform port

Mail
Andre Przywara <andre.przywara@arm.com>

GitHub ID
Andre-ARM

Mail
Javier Almansa Sobrino <Javier.AlmansaSobrino@arm.com>

GitHub ID
javieralso-arm

Files
plat/arm/board/arm_fpga

Arm FVP Platform port

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Harrison Mutai <harrison.mutai@arm.com>

GitHub ID
harrisonmutai-arm

Files
plat/arm/board/fvp

Arm Juno Platform port

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Files
plat/arm/board/juno

Arm Morello and N1SDP Platform ports

Mail
Anurag Koul <anurag.koul@arm.com>

GitHub ID
anukou

Mail
Chandni Cherukuri <chandni.cherukuri@arm.com>

GitHub ID
chandnich

Files
plat/arm/board/morello

Files
plat/arm/board/n1sdp

Arm Rich IoT Platform ports

Mail
Abdellatif El Khlifi <abdellatif.elkhlifi@arm.com>

GitHub ID
abdellatif-elkhlifi

Mail
Hugues Kamba Mpiana <hugues.kambampiana@arm.com>

GitHub ID
hugues-kambampiana-arm

Files
plat/arm/board/corstone700

Files
plat/arm/board/corstone1000

Arm Reference Design platform ports

Mail
Thomas Abraham <thomas.abraham@arm.com>

GitHub ID
thomas-arm

Mail
Vijayenthiran Subramaniam <vijayenthiran.subramaniam@arm.com>

GitHub ID
vijayenthiran-arm

Mail
Rohit Mathew <Rohit.Mathew@arm.com>

GitHub ID
rohit-arm

Files
plat/arm/board/neoverse_rd/common

Files
plat/arm/board/neoverse_rd/platform/rdn2/

Files
plat/arm/board/neoverse_rd/platform/rdv3/

Arm Total Compute platform port

Mail
Artem Kopotev <artem.kopotev@arm.com>

GitHub ID
artkopotev

Mail
Oleksandr Tarhunakov <Oleksandr.Tarhunakov@arm.com>

GitHub ID
o-tarhunakov

Files
plat/arm/board/tc

Arm Automotive RD platform port

Mail
Diego Sueiro <diego.sueiro@arm.com>

GitHub ID
diego-sueiro

Mail
Peter Hoyes <peter.hoyes@arm.com>

GitHub ID
hoyes

Mail
Divin Raj <divin.raj@arm.com>

GitHub ID
divin-raj

Files
plat/arm/board/automotive_rd

Mail
Ahmed Azeem <ahmed.azeem@arm.com>

GitHub ID
Ahmed-Azeem

Files
plat/arm/board/automotive_rd/rdaspen

Aspeed platform port

Mail
Chia-Wei Wang <chiawei_wang@aspeedtech.com>

GitHub ID
ChiaweiW

Mail
Neal Liu <neal_liu@aspeedtech.com>

GitHub ID
Neal-liu

Files
docs/plat/ast2700.rst

Files
plat/aspeed/

HiSilicon HiKey and HiKey960 platform ports

Mail
Haojian Zhuang <haojian.zhuang@linaro.org>

GitHub ID
hzhuang1

Files
docs/plat/hikey.rst

Files
docs/plat/hikey960.rst

Files
plat/hisilicon/hikey/

Files
plat/hisilicon/hikey960/

HiSilicon Poplar platform port

Mail
Shawn Guo <shawn.gsc@gmail.com>

GitHub ID
shawnguo2

Files
docs/plat/poplar.rst

Files
plat/hisilicon/poplar/

Intel SocFPGA platform ports

Mail
Sieu Mun Tang <sieu.mun.tang@intel.com>

GitHub ID
sieumunt

Mail
Benjamin Jit Loon Lim <jit.loon.lim@intel.com>

GitHub ID
BenjaminLimJL

Files
plat/intel/soc/

Files
drivers/intel/soc/

Marvell platform ports and SoC drivers

Mail
Jaiprakash Singh <jaiprakashs@marvell.com>

GitHub ID
sjaypee208

Mail
George Cherian <george.cherian@marvell.com>

GitHub ID
gcherianv

Files
docs/plat/marvell/

Files
plat/marvell/

Files
drivers/marvell/

Files
tools/marvell/

Marvell Armada a80x0_nbx platform port

Mail
Vincent Jardin <vjardin@free.fr>

GitHub ID
vjardin

Files
docs/plat/marvell/armada/a80x0_nbx.rst

Files
plat/marvell/armada/a8k/a80x0_nbx/

MediaTek platform ports

Mail
Leon Chen <leon.chen@mediatek.com>

GitHub ID
leon-chen-mtk

Mail
Jason-CH Chen <jason-ch.chen@mediatek.com>

GitHub ID
jason-ch-chen

Mail
Yidi Lin <yidilin@chromium.org>

GitHub ID
linyidi

Files
docs/plat/mt*.rst

Files
plat/mediatek/

Nuvoton npcm845x platform port

Mail
Hila Miranda-Kuzi <hila.miranda.kuzi1@gmail.com>

GitHub ID
hilamirandakuzi1

Mail
Margarita Glushkin <rutigl@gmail.com>

GitHub ID
rutigl

Mail
Avi Fishman <avi.fishman@nuvoton.com>

GitHub ID
avifishman

Files
docs/plat/npcm845x.rst

Files
include/drivers/nuvoton/

Files
include/plat/nuvoton/

Files
plat/nuvoton/

NVidia platform ports

Mail
Varun Wadekar <vwadekar@nvidia.com>

GitHub ID
vwadekar

Files
docs/plat/nvidia-tegra.rst

Files
include/lib/cpus/aarch64/denver.h

Files
lib/cpus/aarch64/denver.S

Files
plat/nvidia/

NXP i.MX 7 WaRP7 platform port and SoC drivers

Mail
Bryan O'Donoghue <bryan.odonoghue@linaro.org>

GitHub ID
bryanodonoghue

Mail
Jun Nie <jun.nie@linaro.org>

GitHub ID
niej

Files
docs/plat/warp7.rst

Files
plat/imx/common/

Files
plat/imx/imx7/

Files
drivers/imx/timer/

Files
drivers/imx/uart/

Files
drivers/imx/usdhc/

NXP i.MX 8 platform port

Mail
Peng Fan <peng.fan@nxp.com>

GitHub ID
MrVan

Files
docs/plat/imx8.rst

Files
plat/imx/

NXP i.MX8M platform port

Mail
Jacky Bai <ping.bai@nxp.com>

GitHub ID
JackyBai

Files
docs/plat/imx8m.rst

Files
plat/imx/imx8m/

NXP i.MX8ULP platform port

Mail
Jacky Bai <ping.bai@nxp.com>

GitHub ID
JackyBai

Files
docs/plat/imx8ulp.rst

Files
plat/imx/imx8ulp/

NXP i.MX9 platform port

Mail
Jacky Bai <ping.bai@nxp.com>

GitHub ID
JackyBai

Files
docs/plat/imx9.rst

Files
plat/imx/imx93/

Files
plat/imx/imx9/imx94/

Files
plat/imx/imx9/imx95/

NXP QorIQ Layerscape common code for platform ports

Mail
Pankaj Gupta <pankaj.gupta@nxp.com>

GitHub ID
pangupta

Mail
Jiafei Pan <jiafei.pan@nxp.com>

GitHub ID
JiafeiPan

Files
docs/plat/nxp/

Files
plat/nxp/

Files
drivers/nxp/

Files
tools/nxp/

NXP SoC Part LX2160A and its platform port

Mail
Pankaj Gupta <pankaj.gupta@nxp.com>

GitHub ID
pangupta

Files
plat/nxp/soc-lx2160a

Files
plat/nxp/soc-lx2160a/lx2162aqds

Files
plat/nxp/soc-lx2160a/lx2160aqds

Files
plat/nxp/soc-lx2160a/lx2160ardb

NXP SoC Part LS1028A and its platform port

Mail
Jiafei Pan <jiafei.pan@nxp.com>

GitHub ID
JiafeiPan

Files
plat/nxp/soc-ls1028a

Files
plat/nxp/soc-ls1028a/ls1028ardb

NXP SoC Part LS1043A and its platform port

Mail
Jiafei Pan <jiafei.pan@nxp.com>

GitHub ID
JiafeiPan

Files
plat/nxp/soc-ls1043a

Files
plat/nxp/soc-ls1043a/ls1043ardb

NXP SoC Part LS1046A and its platform port

Mail
Jiafei Pan <jiafei.pan@nxp.com>

GitHub ID
JiafeiPan

Files
plat/nxp/soc-ls1046a

Files
plat/nxp/soc-ls1046a/ls1046ardb

Files
plat/nxp/soc-ls1046a/ls1046afrwy

Files
plat/nxp/soc-ls1046a/ls1046aqds

NXP SoC Part LS1088A and its platform port

Mail
Jiafei Pan <jiafei.pan@nxp.com>

GitHub ID
JiafeiPan

Files
plat/nxp/soc-ls1088a

Files
plat/nxp/soc-ls1088a/ls1088ardb

Files
plat/nxp/soc-ls1088a/ls1088aqds

NXP SoC Part S32G274A and its platform port

Mail
Ghennadi Procopciuc <ghennadi.procopciuc@nxp.com>

GitHub ID
gprocopciucnxp

Files
docs/plat/s32g274a.rst

Files
drivers/nxp/clk/s32cc

Files
drivers/nxp/console/linflex_console.S

Files
include/drivers/nxp/console/linflex.h

Files
plat/nxp/s32

QEMU platform port

Mail
Jens Wiklander <jens.wiklander@linaro.org>

GitHub ID
jenswi-linaro

Files
docs/plat/qemu.rst

Files
plat/qemu/

QTI platform port and SoC drivers

Mail
Sumit Garg <sumit.garg@oss.qualcomm.com>

GitHub ID
b49020

Mail
Jorge Ramirez-Ortiz <jorge.ramirez@oss.qualcomm.com>

GitHub ID
ldts

Mail
Sreevyshanavi Kare <skare@qti.qualcomm.com>

GitHub ID
skare-git

Files
docs/plat/qti/

Files
plat/qti/

Files
drivers/qti/

Files
include/drivers/qti/

Files
tools/qti/

QTI MSM8916 platform port

Mail
Stephan Gerhold <stephan@gerhold.net>

GitHub ID
stephan-gh

Mail
Nikita Travkin <nikita@trvn.ru>

GitHub ID
TravMurav

Files
docs/plat/qti/msm8916.rst

Files
plat/qti/bear/

Raspberry Pi 3 platform port

Mail
Ying-Chun Liu (PaulLiu) <paul.liu@linaro.org>

GitHub ID
grandpaul

Files
docs/plat/rpi3.rst

Files
plat/rpi/rpi3/

Files
plat/rpi/common/

Files
drivers/rpi3/

Files
include/drivers/rpi3/

Raspberry Pi 4 platform port

Mail
Andre Przywara <andre.przywara@arm.com>

GitHub ID
Andre-ARM

Files
docs/plat/rpi4.rst

Files
plat/rpi/rpi4/

Files
plat/rpi/common/

Files
drivers/rpi3/

Files
include/drivers/rpi3/

Renesas rcar-gen3 platform port

Mail
Marek Vasut <marek.vasut@gmail.com>

GitHub ID
marex

Files
docs/plat/rcar-gen3.rst

Files
plat/renesas/common

Files
plat/renesas/rcar

Files
drivers/renesas/common

Files
drivers/renesas/rcar

Files
tools/renesas/rcar_layout_create

Renesas RZ/A platform port

Mail
Nhut Nguyen <nhut.nguyen.kc@renesas.com>

GitHub ID
nhutnguyenkc

Files
docs/plat/rz-a3.rst

Files
plat/renesas/rza

Files
drivers/renesas/rza

Files
tools/renesas/rza

Renesas RZ/G2 platform port

Mail
Biju Das <biju.das.jz@bp.renesas.com>

GitHub ID
bijucdas

Mail
Marek Vasut <marek.vasut@gmail.com>

GitHub ID
marex

Mail
Lad Prabhakar <prabhakar.mahadev-lad.rj@bp.renesas.com>

GitHub ID
prabhakarlad

Files
docs/plat/rz-g2.rst

Files
plat/renesas/common

Files
plat/renesas/rzg

Files
drivers/renesas/common

Files
drivers/renesas/rzg

Files
tools/renesas/rzg_layout_create

RockChip platform port

Mail
Tony Xie <tony.xie@rock-chips.com>

GitHub ID
TonyXie06

GitHub ID
rockchip-linux

Mail
Heiko Stuebner <heiko@sntech.de>

GitHub ID
mmind

Files
plat/rockchip/

STMicroelectronics platform ports

Mail
Yann Gautier <yann.gautier@st.com>

GitHub ID
Yann-lms

Mail
Maxime Méré <maxime.mere@foss.st.com>

GitHub ID
meremST

Files
docs/plat/st/*

Files
docs/plat/stm32mp1.rst

Files
drivers/st/

Files
fdts/stm32*

Files
include/drivers/st/

Files
include/dt-bindings/*/stm32*

Files
plat/st/

Files
tools/fiptool/plat_fiptool/st/

Files
tools/stm32image/

Synquacer platform port

Mail
Sumit Garg <sumit.garg@kernel.org>

GitHub ID
b49020

Mail
Masahisa Kojima <kojima.masahisa@socionext.com>

GitHub ID
masahisak

Files
docs/plat/synquacer.rst

Files
plat/socionext/synquacer/

Texas Instruments platform port

Mail
Nishanth Menon <nm@ti.com>

GitHub ID
nmenon

Files
docs/plat/ti-k3.rst

Files
plat/ti/

UniPhier platform port

Mail
Orphan

Files
docs/plat/socionext-uniphier.rst

Files
plat/socionext/uniphier/

AMD-Xilinx platform port

Mail
Michal Simek <michal.simek@amd.com>

GitHub ID
michalsimek

Mail
Akshay Belsare <akshay.belsare@amd.com>

GitHub ID
Akshay-Belsare

Mail
Maheedhar Sai Bollapalli <maheedharsai.bollapalli@amd.com>

GitHub ID
maheedhar-bollapalli

Files
docs/plat/amd*

Files
docs/plat/xilinx*

Files
plat/amd/

Files
plat/xilinx/

Secure Payloads and Dispatchers

OP-TEE dispatcher

Mail
Jens Wiklander <jens.wiklander@linaro.org>

GitHub ID
jenswi-linaro

Files
docs/components/spd/optee-dispatcher.rst

Files
services/spd/opteed/

TLK

Mail
Varun Wadekar <vwadekar@nvidia.com>

GitHub ID
vwadekar

Files
docs/components/spd/tlk-dispatcher.rst

Files
include/bl32/payloads/tlk.h

Files
services/spd/tlkd/

Trusty secure payloads

Mail
Arve Hjønnevåg <arve@android.com>

GitHub ID
arve-android

Mail
Marco Nelissen <marcone@google.com>

GitHub ID
marcone

Mail
Varun Wadekar <vwadekar@nvidia.com>

GitHub ID
: vwadekar

Files
: docs/components/spd/trusty-dispatcher.rst

Files
: services/spd/trusty/

Test Secure Payload (TSP)

Mail
: Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
: ManishVB-Arm

Files
: bl32/tsp/

Files
: services/spd/tspd/

ProvenCore Secure Payload Dispatcher

Mail
: Jérémie Corbier <jeremie.corbier@provenrun.com>

GitHub ID
: jcorbier

Files
: docs/components/spd/pnc-dispatcher.rst

Files
: services/spd/pncd/

Tools

Fiptool

Mail
: Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
: ManishVB-Arm

Mail
: Harrison Mutai <harrison.mutai@arm.com>

GitHub ID
harrisonmutai-arm

Files
tools/fiptool/

Cert_create tool

Mail
Sandrine Bailleux <sandrine.bailleux@arm.com>

GitHub ID
sandrine-bailleux-arm

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Files
tools/cert_create/

Encrypt_fw tool

Mail
Sumit Garg <sumit.garg@kernel.org>

GitHub ID
b49020

Files
tools/encrypt_fw/

Sptool

Mail
Manish Badarkhe <manish.badarkhe@arm.com>

GitHub ID
ManishVB-Arm

Mail
Manish Pandey <manish.pandey2@arm.com>

GitHub ID
manish-pandey-arm

Files
tools/sptool/

Build system

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Mail
Harrison Mutai <harrison.mutai@arm.com>

GitHub ID
harrisonmutai-arm

Files
Makefile

Files
make_helpers/

Threat Model

Mail
Sandrine Bailleux <sandrine.bailleux@arm.com>

GitHub ID
sandrine-bailleux-arm

Mail
Joanna Farley <joanna.farley@arm.com>

GitHub ID
joannafarley-arm

Mail
Raghu Krishnamurthy <raghuoss@raghushome.com>

GitHub ID
raghuncstate

Mail
Varun Wadekar <vwadekar@nvidia.com>

GitHub ID
vwadekar

Files
docs/threat_model/

Conventional Changelog Extensions

Mail
Chris Kay <chris.kay@arm.com>

GitHub ID
CJKay

Files
tools/conventional-changelog-tf-a

1.5 Support & Contact

We welcome any feedback on *TF-A* and there are several methods for providing it or for obtaining support.

> **Warning:** If you think you have found a security vulnerability, please report this using the process defined in the *Security Handling* document.

1.5.1 Mailing Lists

Public mailing lists for TF-A and the wider Trusted Firmware project are hosted on TrustedFirmware.org. The mailing lists can be used for general enquiries, enhancement requests and issue reports, or to follow and participate in technical or organizational discussions around the project. These discussions include design proposals, advance notice of changes and upcoming events.

The relevant lists for the TF-A project are:

- TF-A development
- TF-A-Tests development

You can see a summary of all the lists on the TrustedFirmware.org website.

1.5.2 Open Tech Forum Call

Every other week, we organize a call with all interested TF-A contributors. Anyone is welcome to join. This is an opportunity to discuss any technical topic within the community. More details can be found here.

1.5.3 Issue Tracker

Bug reports may be filed on the issue tracker on Github. Using this tracker gives everyone visibility of the known issues in TF-A.

1.5.4 Arm Licensees

Arm licensees have an additional support conduit - they may contact Arm directly via their partner managers.

1.6 Contributor Acknowledgements

Note: This file is only relevant for legacy contributions, to acknowledge the specific contributors referred to in "Arm Limited and Contributors" copyright notices. As contributors are now encouraged to put their name or company name directly into the copyright notices, this file is not relevant for new contributions. See the *License* document for the correct template to use for new contributions.

- Linaro Limited
- Marvell International Ltd.
- NVIDIA Corporation
- NXP Semiconductors
- Socionext Inc.
- STMicroelectronics
- Xilinx, Inc.

CHAPTER

TWO

GETTING STARTED

2.1 Prerequisites

This document describes the software requirements for building *TF-A* for AArch32 and AArch64 target platforms.

It may possible to build *TF-A* with combinations of software packages that are different from those listed below, however only the software described in this document can be officially supported.

2.1.1 Getting the TF-A Source

Source code for *TF-A* is maintained in a Git repository hosted on TrustedFirmware.org. To clone this repository from the server, run the following in your shell:

```
git clone "https://review.trustedfirmware.org/TF-A/trusted-firmware-a"
```

2.1.2 Requirements

Program	Min supported version
Arm Compiler	6.23
Arm GNU Compiler	15.2
Clang/LLVM	18.1.8
CMake	3.22
Device Tree Compiler	1.6.1
GNU Make	4.3
Node.js[2]	20.11.1
OpenSSL	1.0.0
Poetry	1.3.2
QCBOR[2]	1.2
Sphinx[1]	5.3.0

[2] Required only when enabling DICE Protection Environment support.

[1] Required only for building TF-A documentation.

Toolchain

TF-A can be compiled using any cross-compiler toolchain specified in the preceding table that target Armv7-A or Armv8-A. For AArch32 and AArch64 builds, the respective targets required are `arm-none-eabi` and `aarch64-none-elf`.

Testing has been performed with the version of the Arm GNU compiler listed in the table above. This can be installed from the Arm Developer website.

In addition, a native compiler is required to build supporting tools.

Note: Versions greater than the ones specified are likely but not guaranteed to work. This is predominantly because TF-A carries its own copy of compiler-rt, which may be older than the version expected by the compiler. Fixes and bug reports are always welcome.

Note: For instructions on how to select the cross compiler refer to *Performing an Initial Build*.

OpenSSL

OpenSSL is required to build the cert_create, encrypt_fw, and fiptool tools.

If using OpenSSL 3, older Linux versions may require it to be built from source code, as it may not be available in the default package repositories. Please refer to the OpenSSL project documentation for more information.

Warning: Versions 1.0.x and from v3.0.0 up to v3.0.6 are strongly advised against due to concerns regarding security vulnerabilities!

Device Tree Compiler (DTC)

Needed if you want to rebuild the provided Flattened Device Tree (FDT) source files (`.dts` files). DTC is available for Linux through the package repositories of most distributions.

Arm Development Studio (Arm-DS)

The standard software package used for debugging software on Arm development platforms and *FVP* models.

Node.js

Highly recommended, and necessary in order to install and use the packaged Git hooks and helper tools. Without these tools you will need to rely on the CI for feedback on commit message conformance.

Poetry

Required for managing Python dependencies, this will allow you to reliably reproduce a Python environment to build documentation and run some of the integrated Python tools. Most importantly, it ensures your system environment will not be affected by dependencies in the Python scripts.

For installation instructions, see the official Poetry documentation.

2.1.3 Package Installation (Linux)

TF-A can be compiled on both Linux and Windows-based machines. However, we strongly recommend using a UNIX-compatible build environment.

Testing is performed using Ubuntu 22.04 LTS (64-bit), but other distributions should also work, provided the necessary tools and libraries are installed.

The following are steps to install the required packages:

```
sudo apt install build-essential
```

The optional packages can be installed using:

```
sudo apt install device-tree-compiler
```

Additionally, to install a version of Node.js compatible with TF-A's repository scripts, you can use the Node Version Manager. To install both NVM and an appropriate version of Node.js, run the following **from the root directory of the repository**:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh |␣
↪bash
exec "$SHELL" -ic "nvm install; exec $SHELL"
```

2.1.4 Supporting Files

TF-A has been tested with pre-built binaries and file systems from Linaro Release 20.01. Alternatively, you can build the binaries from source using instructions in *Performing an Initial Build*.

Additional Steps for Contributors

If you are planning on contributing back to TF-A, there are some things you'll want to know.

TF-A is hosted by a Gerrit Code Review server. Gerrit requires that all commits include a `Change-Id` footer, and this footer is typically automatically generated by a Git hook installed by you, the developer.

If you have Node.js installed already, you can automatically install this hook, along with any additional hooks and Javascript-based tooling that we use, by running from within your newly-cloned repository:

```
npm install --no-save
```

If you have opted **not** to install Node.js, you can install the Gerrit hook manually by running:

```
curl -Lo $(git rev-parse --git-dir)/hooks/commit-msg https://review.
↪trustedfirmware.org/tools/hooks/commit-msg
chmod +x $(git rev-parse --git-dir)/hooks/commit-msg
```

You can read more about Git hooks in the *githooks* page of the Git documentation, available here.

Cloning Additional Git Submodules

Some dependencies in TF-A, such as Transfer List Library `libtl`, are managed using Git submodules. Submodules allow external repositories to be included within the main project while maintaining their own commit history.

Initial Clone with Submodules

If you're cloning the repository for the first time, run the following commands to initialize and fetch all submodules:

```
git clone --recurse-submodules "https://git.trustedfirmware.org/TF-A/trusted-
↪firmware-a"
```

This ensures all submodules (including `libtl`) are correctly checked out.

Updating Submodules

If the project updates the reference to a submodule (e.g., points to a new commit of `libtl`), you can update your local copy by running:

```
git pull
git submodule update --init --recursive
```

To fetch the latest commits from all submodules, you can use:

```
git submodule update --remote
```

2.2 Building Documentation

To create a rendered copy of this documentation locally you can use the Sphinx tool to build and package the plain-text documents into HTML-formatted pages.

If you are building the documentation for the first time then you will need to check that you have the required software packages, as described in the *Prerequisites* section that follows.

Note: An online copy of the documentation is available at https://www.trustedfirmware.org/docs/tf-a, if you want to view a rendered copy without doing a local build.

2.2.1 Prerequisites

For building a local copy of the *TF-A* documentation you will need:

- Python 3 (3.8 or later)
- PlantUML (1.2017.15 or later)
- Poetry (Python dependency manager)
- Optionally, the Dia application can be installed if you need to edit existing `.dia` diagram files, or create new ones.

Below is an example set of instructions to get a working environment (tested on Ubuntu):

```
sudo apt install python3 python3-pip plantuml [dia]
curl -sSL https://install.python-poetry.org | python3 -
```

2.2.2 Building rendered documentation

The documentation can be compiled into HTML-formatted pages from the project root directory by running:

```
make doc
```

Output from the build process will be placed in: `docs/build/html`.

Other Output Formats

We also support building documentation in other formats. From the `docs` directory of the project, run the following command to see the supported formats.

```
make -C docs help
```

To build the documentation in PDF format, additionally ensure that the following packages are installed:

- FreeSerif font
- latexmk
- librsvg2-bin
- xelatex
- xindy

Below is an example set of instructions to install the required packages (tested on Ubuntu):

```
sudo apt install fonts-freefont-otf latexmk librsvg2-bin texlive-xetex xindy
```

Once all the dependencies are installed, run the command `poetry run make -C docs latexpdf` to build the documentation. Output from the build process (`trustedfirmware-a.pdf`) can be found in `docs/build/latex`.

Building rendered documentation from Poetry's virtual environment

If Poetry is installed, the `doc` target wraps its build steps with `poetry run`, which runs the specified command within the project's virtual environment. The easiest way to activate this environment manually is by using the `poetry shell` command.

Running `poetry shell` from the directory containing this project, activates the same virtual environment. This creates a sub-shell through which you can build the documentation directly with `make`.

```
poetry shell
make -C docs html
```

Type `exit` to deactivate the virtual environment and exit this new shell. For other use cases, please see the official Poetry documentation.

2.2.3 Building rendered documentation from a container

There may be cases where you can not either install or upgrade required dependencies to generate the documents, so in this case, one way to create the documentation is through a docker container. The first step is to check if docker is installed in your host, otherwise check main docker page for installation instructions. Once installed, run the following script from project root directory

```
docker run --rm -v $PWD:/tf-a sphinxdoc/sphinx \
     bash -c 'cd /tf-a &&
         apt-get update && apt-get install -y curl plantuml &&
         curl -sSL https://install.python-poetry.org | python3 - &&
         ~/.local/bin/poetry run make doc'
```

The above command fetches the `sphinxdoc/sphinx` container from docker hub, launches the container, installs documentation requirements and finally creates the documentation. Once done, exit the container and output from the build process will be placed in: `docs/build/html`.

2.3 Performing an Initial Build

- Before building TF-A, the environment variable `CROSS_COMPILE` must point to your cross compiler.

  For AArch64:

  ```
  export CROSS_COMPILE=<path-to-aarch64-gcc>/bin/aarch64-none-elf-
  ```

  For AArch32:

  ```
  export CROSS_COMPILE=<path-to-aarch32-gcc>/bin/arm-none-eabi-
  ```

  It is possible to build TF-A using Clang or Arm Compiler 6. To do so `CC` needs to point to the clang or armclang binary, which will also select the clang or armclang assembler. Arm Compiler 6 will be selected when the base name of the path assigned to `CC` matches the string 'armclang'. GNU binutils are required since the TF-A build system doesn't currently support Arm Scatter files. Meaning the GNU linker is used by default for Arm Compiler 6. Because of this dependency, `CROSS_COMPILE` should be set as described above.

  For AArch64 using Arm Compiler 6:

  ```
  export CROSS_COMPILE=<path-to-aarch64-gcc>/bin/aarch64-none-elf-
  make CC=<path-to-armclang>/bin/armclang PLAT=<platform> all
  ```

  On the other hand, Clang uses LLVM linker (LLD) and other LLVM binutils by default instead of GNU utilities (LLVM linker (LLD) 14.0.0 is known to work with TF-A). `CROSS_COMPILE` need not be set for Clang. Please note, that the default linker may be manually overridden using the `LD` variable.

  Clang will be selected when the base name of the path assigned to `CC` contains the string 'clang'. This is to allow both clang and clang-X.Y to work.

  For AArch64 using clang:

  ```
  make CC=<path-to-clang>/bin/clang PLAT=<platform> all
  ```

- Change to the root directory of the TF-A source tree and build.

  For AArch64:

```
make PLAT=<platform> all
```

For AArch32:

```
make PLAT=<platform> ARCH=aarch32 AARCH32_SP=sp_min all
```

Notes:

- If `PLAT` is not specified, `fvp` is assumed by default. See the *Build Options* document for more information on available build options.
- (AArch32 only) Currently only `PLAT=fvp` is supported.
- (AArch32 only) `AARCH32_SP` is the AArch32 EL3 Runtime Software and it corresponds to the BL32 image. A minimal `AARCH32_SP`, sp_min, is provided by TF-A to demonstrate how PSCI Library can be integrated with an AArch32 EL3 Runtime Software. Some AArch32 EL3 Runtime Software may include other runtime services, for example Trusted OS services. A guide to integrate PSCI library with AArch32 EL3 Runtime Software can be found at *Porting Guide*.
- (AArch64 only) The TSP (Test Secure Payload), corresponding to the BL32 image, is not compiled in by default. Refer to the *Test Secure Payload (TSP) and Dispatcher (TSPD)* document for details on building the TSP.
- By default this produces a release version of the build. To produce a debug version instead, refer to the "Debugging options" section below.
- The build process creates products in a `build` directory tree, building the objects and binaries for each boot loader stage in separate sub-directories. The following boot loader binary files are created from the corresponding ELF files:
  - `build/<platform>/<build-type>/bl1.bin`
  - `build/<platform>/<build-type>/bl2.bin`
  - `build/<platform>/<build-type>/bl31.bin` (AArch64 only)
  - `build/<platform>/<build-type>/bl32.bin` (mandatory for AArch32)

  where `<platform>` is the name of the chosen platform and `<build-type>` is either `debug` or `release`. The actual number of images might differ depending on the platform.

- Build products for a specific build variant can be removed using:

```
make DEBUG=<D> PLAT=<platform> clean
```

… where `<D>` is `0` or `1`, as specified when building.

The build tree can be removed completely using:

```
make realclean
```

2.4 Building Supporting Tools

Note: OpenSSL 3.0 is needed in order to build the tools. A custom installation can be used if not updating the OpenSSL version on the OS. In order to do this, use the `OPENSSL_DIR` variable after the `make` command to indicate the location of the custom OpenSSL build. Then, to run the tools, use the `LD_LIBRARY_PATH` to indicate the location of the built libraries. More info about `OPENSSL_DIR` can be found at *Build Options*.

2.4.1 Building and using the FIP tool

The following snippets build a *FIP* for the FVP platform. While it is not an intrinsic part of the FIP format, a BL33 image is required for these examples. For the purposes of experimentation, Trusted Firmware-A Tests (*tftf.bin*`) may be used. Refer to to the TFTF documentation for instructions on building a TFTF binary.

The TF-A build system provides the make target `fip` to create a FIP file for the specified platform using the FIP creation tool included in the TF-A project. Examples below show how to build a FIP file for FVP, packaging TF-A and BL33 images.

For AArch64:

```
make PLAT=fvp BL33=<path-to>/bl33.bin fip
```

For AArch32:

```
make PLAT=fvp ARCH=aarch32 AARCH32_SP=sp_min BL33=<path-to>/bl33.bin fip
```

The resulting FIP may be found in:

```
build/fvp/<build-type>/fip.bin
```

For advanced operations on FIP files, it is also possible to independently build the tool and create or modify FIPs using this tool. To do this, follow these steps:

It is recommended to remove old artifacts before building the tool:

```
make -C tools/fiptool clean
```

Build the tool:

```
make [DEBUG=1] [V=1] fiptool
```

The tool binary can be located in:

```
./tools/fiptool/fiptool
```

Invoking the tool with `help` will print a help message with all available options.

Example 1: create a new Firmware package `fip.bin` that contains BL2 and BL31:

```
./tools/fiptool/fiptool create \
    --tb-fw build/<platform>/<build-type>/bl2.bin \
    --soc-fw build/<platform>/<build-type>/bl31.bin \
    fip.bin
```

Example 2: view the contents of an existing Firmware package:

```
./tools/fiptool/fiptool info <path-to>/fip.bin
```

Example 3: update the entries of an existing Firmware package:

```
# Change the BL2 from Debug to Release version
./tools/fiptool/fiptool update \
    --tb-fw build/<platform>/release/bl2.bin \
    build/<platform>/debug/fip.bin
```

Example 4: unpack all entries from an existing Firmware package:

```
# Images will be unpacked to the working directory
./tools/fiptool/fiptool unpack <path-to>/fip.bin
```

Example 5: remove an entry from an existing Firmware package:

```
./tools/fiptool/fiptool remove \
    --tb-fw build/<platform>/debug/fip.bin
```

Note that if the destination FIP file exists, the create, update and remove operations will automatically overwrite it.

The unpack operation will fail if the images already exist at the destination. In that case, use -f or –force to continue.

More information about FIP can be found in the *Firmware Design* document.

2.4.2 Building the Certificate Generation Tool

The `cert_create` tool is built as part of the TF-A build process when the `fip` make target is specified and TBB is enabled (as described in the previous section), but it can also be built separately with the following command:

```
make PLAT=<platform> [DEBUG=1] [V=1] certtool
```

For platforms that require their own IDs in certificate files, the generic 'cert_create' tool can be built with the following command. Note that the target platform must define its IDs within a `platform_oid.h` header file for the build to succeed.

```
make PLAT=<platform> USE_TBBR_DEFS=0 [DEBUG=1] [V=1] certtool
```

`DEBUG=1` builds the tool in debug mode. `V=1` makes the build process more verbose. The following command should be used to obtain help about the tool:

```
./tools/cert_create/cert_create -h
```

Building the Firmware Encryption Tool

The `encrypt_fw` tool is built as part of the TF-A build process when the `fip` make target is specified, DECRYPTION_SUPPORT and TBB are enabled, but it can also be built separately with the following command:

```
make PLAT=<platform> [DEBUG=1] [V=1] enctool
```

`DEBUG=1` builds the tool in debug mode. `V=1` makes the build process more verbose. The following command should be used to obtain help about the tool:

```
./tools/encrypt_fw/encrypt_fw -h
```

Note that the enctool in its current implementation only supports encryption key to be provided in plain format. A typical implementation can very well extend this tool to support custom techniques to protect encryption key.

Also, a user may choose to provide encryption key or nonce as an input file via using `cat <filename>` instead of a hex string.

2.5 Build Options

The TF-A build system supports the following build options. Unless mentioned otherwise, these options are expected to be specified at the build command line and are not to be modified in any component makefiles. Note that the build system doesn't track dependency for build options. Therefore, if any of the build options are changed from a previous build, a clean build must be performed.

2.5.1 Common build options

- `AARCH32_INSTRUCTION_SET`: Choose the AArch32 instruction set that the compiler should use. Valid values are T32 and A32. It defaults to T32 due to code having a smaller resulting size.
- `AARCH32_SP` : Choose the AArch32 Secure Payload component to be built as as the BL32 image when `ARCH=aarch32`. The value should be the path to the directory containing the SP source, relative to the `bl32/`; the directory is expected to contain a makefile called `<aarch32_sp-value>.mk`.
- `ARM_BL2_SP_LIST_DTS`: Path to DTS file snippet to override the hardcoded SP nodes in tb_fw_config.
- `ARM_SPMC_MANIFEST_DTS` : path to an alternate manifest file used as the SPMC Core manifest. Valid when `SPD=spmd` is selected.
- `BL2`: This is an optional build option which specifies the path to BL2 image for the `fip` target. In this case, the BL2 in the TF-A will not be built.

- BL2U: This is an optional build option which specifies the path to BL2U image. In this case, the BL2U in TF-A will not be built.
- RESET_TO_BL2: Boolean option to enable BL2 entrypoint as the CPU reset vector instead of the BL1 entrypoint. It can take the value 0 (CPU reset to BL1 entrypoint) or 1 (CPU reset to BL2 entrypoint). The default value is 0.
- BL2_RUNS_AT_EL3: This is an implicit flag to denote that BL2 runs at EL3. While it is explicitly set to 1 when RESET_TO_BL2 is set to 1 it can also be true in a 4-world system where RESET_TO_BL2 is 0.
- BL2_INV_DCACHE: This is an optional build option which control dcache invalidation upon BL2 entry. Some platform cannot handle cache operations during entry as the coherency unit is not yet initialized. This may cause crashing. Leaving this option to '1' (default) will allow the operation. This option is only relevant when BL2 is set to run at EL3.
- BL2_ENABLE_SP_LOAD: Boolean option to enable loading SP packages from the FIP. Automatically enabled if SP_LAYOUT_FILE is provided.
- BL2_IN_XIP_MEM: In some use-cases BL2 will be stored in eXecute In Place (XIP) memory, like BL1. In these use-cases, it is necessary to initialize the RW sections in RAM, while leaving the RO sections in place. This option enable this use-case. For now, this option is only supported when RESET_TO_BL2 is set to '1'.
- BL31: This is an optional build option which specifies the path to BL31 image for the fip target. In this case, the BL31 in TF-A will not be built.
- BL31_KEY: This option is used when GENERATE_COT=1. It specifies a file that contains the BL31 private key in PEM format or a PKCS11 URI. If SAVE_KEYS=1, only a file is accepted and it will be used to save the key.
- BL32: This is an optional build option which specifies the path to BL32 image for the fip target. In this case, the BL32 in TF-A will not be built.
- BL32_EXTRA1: This is an optional build option which specifies the path to Trusted OS Extra1 image for the fip target.
- BL32_EXTRA2: This is an optional build option which specifies the path to Trusted OS Extra2 image for the fip target.
- BL32_KEY: This option is used when GENERATE_COT=1. It specifies a file that contains the BL32 private key in PEM format or a PKCS11 URI. If SAVE_KEYS=1, only a file is accepted and it will be used to save the key.
- RMM: This is an optional build option used when ENABLE_RMM is set. It specifies the path to RMM binary for the fip target. If the RMM option is not specified, TF-A builds the TRP to load and run at R-EL2.
- BL33: Path to BL33 image in the host file system. This is mandatory for fip target in case TF-A BL2 is used.
- BL33_KEY: This option is used when GENERATE_COT=1. It specifies a file that contains the BL33 private key in PEM format or a PKCS11 URI. If SAVE_KEYS=1, only a file is accepted and it will be used to save the key.

- BUILD_MESSAGE_TIMESTAMP: String used to identify the time and date of the compilation of each build. It must be set to a C string (including quotes where applicable). Defaults to a string that contains the time and date of the compilation.
- BUILD_STRING: Input string for VERSION_STRING, which allows the TF-A build to be uniquely identified. Defaults to the current git commit id.
- BUILD_BASE: Output directory for the build. Defaults to ./build
- CFLAGS: Extra user options appended on the compiler's command line in addition to the options set by the build system.
- COLD_BOOT_SINGLE_CPU: This option indicates whether the platform may release several CPUs out of reset. It can take either 0 (several CPUs may be brought up) or 1 (only one CPU will ever be brought up during cold reset). Default is 0. If the platform always brings up a single CPU, there is no need to distinguish between primary and secondary CPUs and the boot path can be optimised. The plat_is_my_cpu_primary() and plat_secondary_cold_boot_setup() platform porting interfaces do not need to be implemented in this case.
- COT: When Trusted Boot is enabled, selects the desired chain of trust. Defaults to tbbr.
- CRASH_REPORTING: A non-zero value enables a console dump of processor register state when an unexpected exception occurs during execution of BL31. This option defaults to the value of DEBUG - i.e. by default this is only enabled for a debug build of the firmware.
- CREATE_KEYS: This option is used when GENERATE_COT=1. It tells the certificate generation tool to create new keys in case no valid keys are present or specified. Allowed options are '0' or '1'. Default is '1'.
- DEBUG: Chooses between a debug and release build. It can take either 0 (release) or 1 (debug) as values. 0 is the default.
- DECRYPTION_SUPPORT: This build flag enables the user to select the authenticated decryption algorithm to be used to decrypt firmware/s during boot. It accepts 2 values: aes_gcm and none. The default value of this flag is none to disable firmware decryption which is an optional feature as per TBBR.
- DISABLE_BIN_GENERATION: Boolean option to disable the generation of the binary image. If set to 1, then only the ELF image is built. 0 is the default.
- DYN_DISABLE_AUTH: Provides the capability to dynamically disable Trusted Board Boot authentication at runtime. This option is meant to be enabled only for development platforms. TRUSTED_BOARD_BOOT flag must be set if this flag has to be enabled. 0 is the default.
- E: Boolean option to make warnings into errors. Default is 1.

  When specifying higher warnings levels (W=1 and higher), this option defaults to 0. This is done to encourage contributors to use them, as they are expected to produce warnings that would otherwise fail the build. New contributions are still expected to build with W=0 and E=1 (the default).
- EARLY_CONSOLE: This option is used to enable early traces before default console is properly setup. It introduces EARLY_* traces macros, that will use the non-EARLY traces macros if the flag is enabled, or do nothing otherwise. To use this feature, platforms will have to create the function plat_setup_early_console(). Default is 0 (disabled)

- EL3_PAYLOAD_BASE: This option enables booting an EL3 payload instead of the normal boot flow. It must specify the entry point address of the EL3 payload. Please refer to the "Booting an EL3 payload" section for more details.
- ENABLE_AMU_AUXILIARY_COUNTERS: Enables support for AMU auxiliary counters (also known as group 1 counters). These are implementation-defined counters, and as such require additional platform configuration. Default is 0.
- ENABLE_ASSERTIONS: This option controls whether or not calls to assert() are compiled out. For debug builds, this option defaults to 1, and calls to assert() are left in place. For release builds, this option defaults to 0 and calls to assert() function are compiled out. This option can be set independently of DEBUG. It can also be used to hide any auxiliary code that is only required for the assertion and does not fit in the assertion itself.
- ENABLE_BACKTRACE: This option controls whether to enable backtrace dumps or not. It is supported in both AArch64 and AArch32. However, in AArch32 the format of the frame records are not defined in the AAPCS and they are defined by the implementation. This implementation of backtrace only supports the format used by GCC when T32 interworking is disabled. For this reason enabling this option in AArch32 will force the compiler to only generate A32 code. This option is enabled by default only in AArch64 debug builds, but this behaviour can be overridden in each platform's Makefile or in the build command line.
- ENABLE_LTO: Boolean option to enable Link Time Optimization (LTO) support. This option is currently only supported for AArch64. On GCC it only applies to TF-A proper, and not its libraries. If LTO on libraries (except the libc) is desired a platform can pass *-flto -ffat-lto-objects* as long as GCC >= 14 is in use. ENABLE_LTO is enabled by default on release builds. Default is 0.
- ENABLE_MPMM: Boolean option to enable support for the Maximum Power Mitigation Mechanism supported by certain Arm cores, which allows the SoC firmware to detect and limit high activity events to assist in SoC processor power domain dynamic power budgeting and limit the triggering of whole-rail (i.e. clock chopping) responses to overcurrent conditions. Defaults to 0.
- ENABLE_PIE: Boolean option to enable Position Independent Executable(PIE) support within generic code in TF-A. This option is currently only supported in BL2, BL31, and BL32 (TSP) for AARCH64 binaries, and in BL32 (SP_min) for AARCH32. Default is 0.
- ENABLE_PMF: Boolean option to enable support for optional Performance Measurement Framework(PMF). Default is 0.
- ENABLE_PSCI_STAT: Boolean option to enable support for optional PSCI functions PSCI_STAT_RESIDENCY and PSCI_STAT_COUNT. Default is 0. In the absence of an alternate stat collection backend, ENABLE_PMF must be enabled. If ENABLE_PMF is set, the residency statistics are tracked in software.
- ENABLE_RUNTIME_INSTRUMENTATION: Boolean option to enable runtime instrumentation which injects timestamp collection points into TF-A to allow runtime performance to be measured. Currently, only PSCI is instrumented. Enabling this option enables the ENABLE_PMF build option as well. Default is 0.
- ENABLE_STACK_PROTECTOR: String option to enable the stack protection checks in GCC. Allowed values are "all", "strong", "default" and "none". The default value is set to "none". "strong" is the recommended stack protection level if this feature is desired. "none" disables the stack protection. For all values other than "none", the plat_get_stack_protector_canary() plat-

form hook needs to be implemented. The value is passed as the last component of the option `-fstack-protector-$ENABLE_STACK_PROTECTOR`.

- **`ENABLE_ERRATA_ALL`: This option is used only for testing purposes, Boolean**
  option to enable the workarounds for all errata that TF-A implements. Normally they should be explicitly enabled depending on each platform's needs. Not recommended for release builds. This option is default set to 0.
- `ENCRYPT_BL31`: Binary flag to enable encryption of BL31 firmware. This flag depends on `DECRYPTION_SUPPORT` build flag.
- `ENCRYPT_BL32`: Binary flag to enable encryption of Secure BL32 payload. This flag depends on `DECRYPTION_SUPPORT` build flag.
- `ENC_KEY`: A 32-byte (256-bit) symmetric key in hex string format. It could either be SSK or BSSK depending on `FW_ENC_STATUS` flag. This value depends on `DECRYPTION_SUPPORT` build flag.
- `ENC_NONCE`: A 12-byte (96-bit) encryption nonce or Initialization Vector (IV) in hex string format. This value depends on `DECRYPTION_SUPPORT` build flag.
- `ERROR_DEPRECATED`: This option decides whether to treat the usage of deprecated platform APIs, helper functions or drivers within Trusted Firmware as error. It can take the value 1 (flag the use of deprecated APIs as error) or 0. The default is 0.
- `ETHOSN_NPU_DRIVER`: boolean option to enable a SiP service that can configure an Arm® Ethos™-N NPU. To use this service the target platform's `HW_CONFIG` must include the device tree nodes for the NPU. Currently, only the Arm Juno platform has this included in its `HW_CONFIG` and the platform only loads the `HW_CONFIG` in AArch64 builds. Default is 0.
- `ETHOSN_NPU_TZMP1`: boolean option to enable TZMP1 support for the Arm® Ethos™-N NPU. Requires `ETHOSN_NPU_DRIVER` and `TRUSTED_BOARD_BOOT` to be enabled.
- `ETHOSN_NPU_FW`: location of the NPU firmware binary (`` `ethosn.bin` ``). This firmware image will be included in the FIP and loaded at runtime.
- `EL3_EXCEPTION_HANDLING`: When set to `1`, enable handling of exceptions targeted at EL3. When set `0` (default), no exceptions are expected or handled at EL3, and a panic will result. The exception to this rule is when `SPMD_SPM_AT_SEL2` is set to `1`, in which case, only exceptions occuring during normal world execution, are trapped to EL3. Any exception trapped during secure world execution are trapped to the SPMC. This is supported only for AArch64 builds.
- `FAULT_INJECTION_SUPPORT`: Boolean option to enable FEAT_RASv1p1 fault injection. When unset, all accesses will be trapped to EL3 and emulated as RAZ/WI. Default value is `0`.

  This feature is intended for testing purposes only, and is advisable to keep disabled for production images.
- `FIP_NAME`: This is an optional build option which specifies the FIP filename for the `fip` target. Default is `fip.bin`.
- `FWU_FIP_NAME`: This is an optional build option which specifies the FWU FIP filename for the `fwu_fip` target. Default is `fwu_fip.bin`.
- `FW_ENC_STATUS`: Top level firmware's encryption numeric flag, values:

```
0: Encryption is done with Secret Symmetric Key (SSK) which is common
   for a class of devices.
1: Encryption is done with Binding Secret Symmetric Key (BSSK) which is
   unique per device.
```

This flag depends on DECRYPTION_SUPPORT build flag.

- GENERATE_COT: Boolean flag used to build and execute the cert_create tool to create certificates as per the Chain of Trust described in *Trusted Board Boot*. The build system then calls fiptool to include the certificates in the FIP and FWU_FIP. Default value is '0'.

  Specify both TRUSTED_BOARD_BOOT=1 and GENERATE_COT=1 to include support for the Trusted Board Boot feature in the BL1 and BL2 images, to generate the corresponding certificates, and to include those certificates in the FIP and FWU_FIP.

  Note that if TRUSTED_BOARD_BOOT=0 and GENERATE_COT=1, the BL1 and BL2 images will not include support for Trusted Board Boot. The FIP will still include the corresponding certificates. This FIP can be used to verify the Chain of Trust on the host machine through other mechanisms.

  Note that if TRUSTED_BOARD_BOOT=1 and GENERATE_COT=0, the BL1 and BL2 images will include support for Trusted Board Boot, but the FIP and FWU_FIP will not include the corresponding certificates, causing a boot failure.

- GICV2_G0_FOR_EL3: Unlike GICv3, the GICv2 architecture doesn't have inherent support for specific EL3 type interrupts. Setting this build option to 1 assumes GICv2 *Group 0* interrupts are expected to target EL3, both by *platform abstraction layer* and *Interrupt Management Framework*. This allows GICv2 platforms to enable features requiring EL3 interrupt type. This also means that all GICv2 Group 0 interrupts are delivered to EL3, and the Secure Payload interrupts needs to be synchronously handed over to Secure EL1 for handling. The default value of this option is 0, which means the Group 0 interrupts are assumed to be handled by Secure EL1.

- HANDLE_EA_EL3_FIRST_NS: When set to 1, External Aborts and SError Interrupts, resulting from errors in NS world, will be always trapped in EL3 i.e. in BL31 at runtime. When set to 0 (default), these exceptions will be trapped in the current exception level (or in EL1 if the current exception level is EL0).

- HW_ASSISTED_COHERENCY: On most Arm systems to-date, platform-specific software operations are required for CPUs to enter and exit coherency. However, newer systems exist where CPUs' entry to and exit from coherency is managed in hardware. Such systems require software to only initiate these operations, and the rest is managed in hardware, minimizing active software management. In such systems, this boolean option enables TF-A to carry out build and run-time optimizations during boot and power management operations. This option defaults to 0 and if it is enabled, then it implies WARMBOOT_ENABLE_DCACHE_EARLY is also enabled.

  If this flag is disabled while the platform which TF-A is compiled for includes cores that manage coherency in hardware, then a compilation error is generated. This is based on the fact that a system cannot have, at the same time, cores that manage coherency in hardware and cores that don't. In other words, a platform cannot have, at the same time, cores that require HW_ASSISTED_COHERENCY=1 and cores that require HW_ASSISTED_COHERENCY=0.

  Note that, when HW_ASSISTED_COHERENCY is enabled, version 2 of translation library (xlat tables v2) must be used; version 1 of translation library is not supported.

- `IMPDEF_SYSREG_TRAP`: Numeric value to enable the handling traps for implementation defined system register accesses from lower ELs. Default value is `0`.
- `INVERTED_MEMMAP`: memmap tool print by default lower addresses at the bottom, higher addresses at the top. This build flag can be set to '1' to invert this behavior. Lower addresses will be printed at the top and higher addresses at the bottom.
- `INIT_UNUSED_NS_EL2`: This build flag guards code that disables EL2 safely in scenario where NS-EL2 is present but unused. This flag is set to 0 by default. Platforms without NS-EL2 in use must enable this flag.
- `KEY_ALG`: This build flag enables the user to select the algorithm to be used for generating the PKCS keys and subsequent signing of the certificate. It accepts 5 values: `rsa`, `rsa_1_5`, `ecdsa`, `ecdsa-brainpool-regular` and `ecdsa-brainpool-twisted`. The option `rsa_1_5` is the legacy PKCS#1 RSA 1.5 algorithm which is not TBBR compliant and is retained only for compatibility. The default value of this flag is `rsa` which is the TBBR compliant PKCS#1 RSA 2.1 scheme.
- `KEY_SIZE`: This build flag enables the user to select the key size for the algorithm specified by `KEY_ALG`. The valid values for `KEY_SIZE` depend on the chosen algorithm and the cryptographic module.

KEY_ALG	Possible key sizes
rsa	1024 , 2048 (default), 3072, 4096
ecdsa	256 (default), 384
ecdsa-brainpool-regular	256 (default)
ecdsa-brainpool-twisted	256 (default)

- `HASH_ALG`: This build flag enables the user to select the secure hash algorithm. It accepts 3 values: `sha256`, `sha384` and `sha512`. The default value of this flag is `sha256`.
- **`HW_CONFIG_BASE`: This option specifies the location in memory where the DTB**
  should either be loaded by BL2 or can be found by later stages.
- `LDFLAGS`: Extra user options appended to the linkers' command line in addition to the one set by the build system.
- `LOG_LEVEL`: Chooses the log level, which controls the amount of console log output compiled into the build. This should be one of the following:

```
0  (LOG_LEVEL_NONE)
10 (LOG_LEVEL_ERROR)
20 (LOG_LEVEL_NOTICE)
30 (LOG_LEVEL_WARNING)
40 (LOG_LEVEL_INFO)
50 (LOG_LEVEL_VERBOSE)
```

  All log output up to and including the selected log level is compiled into the build. The default value is 40 in debug builds and 20 in release builds.

  `LOG_DEBUG`: Boolean option to enable support for module level internal logs. There can be situation where a module has detail internal debugging logs, these debugging logs may not be required to print even when log level is VERBOSE. Such logs can be put under this flag. This is a file level build flag. By default this should be disabled (`0`) in each file.

- `MEASURED_BOOT`: Boolean flag to include support for the Measured Boot feature. This flag can be enabled with `TRUSTED_BOARD_BOOT` in order to provide trust that the code taking the measurements and recording them has not been tampered with.

  This option defaults to 0.

- `DISCRETE_TPM`: Boolean flag to include support for a Discrete TPM.

  This option defaults to 0.

- `TPM_INTERFACE`: When `DISCRETE_TPM=1`, this is a required flag to select the TPM interface. Currently only one interface is supported:

```
FIFO_SPI
```

- `MBOOT_TPM_HASH_ALG`: Build flag to select the TPM hash algorithm used during Measured Boot. Currently only accepts `sha256` as a valid algorithm.
- `MARCH_DIRECTIVE`: used to pass a -march option from the platform build options to the compiler. An example usage:

```
MARCH_DIRECTIVE := -march=armv8.5-a
```

- `HARDEN_SLS`: used to pass -mharden-sls=all from the TF-A build options to the compiler currently supporting only of the options. GCC documentation: https://gcc.gnu.org/onlinedocs/gcc/AArch64-Options.html#index-mharden-sls

  An example usage:

```
HARDEN_SLS := 1
```

  This option defaults to 0.

- `NON_TRUSTED_WORLD_KEY`: This option is used when `GENERATE_COT=1`. It specifies a file that contains the Non-Trusted World private key in PEM format or a PKCS11 URI. If `SAVE_KEYS=1`, only a file is accepted and it will be used to save the key.
- `NS_BL2U`: Path to NS_BL2U image in the host file system. This image is optional. It is only needed if the platform makefile specifies that it is required in order to build the `fwu_fip` target.
- `NS_TIMER_SWITCH`: (deprecated) Enable save and restore for non-secure timer register contents upon world switch. It can take either 0 (don't save and restore) or 1 (do save and restore). 0 is the default. An SPD may set this to 1 if it wants the timer registers to be saved and restored. This option has been deprecated since it breaks Linux preemption model.
- `OVERRIDE_LIBC`: This option allows platforms to override the default libc for the BL image. It can be either 0 (include) or 1 (remove). The default value is 0.
- `PL011_GENERIC_UART`: Boolean option to indicate the PL011 driver that the underlying hardware is not a full PL011 UART but a minimally compliant generic UART, which is a subset of the PL011. The driver will not access any register that is not part of the SBSA generic UART specification. Default value is 0 (a full PL011 compliant UART is present).

- PLAT: Choose a platform to build TF-A for. The chosen platform name must be subdirectory of any depth under plat/, and must contain a platform makefile named platform.mk. For example, to build TF-A for the Arm Juno board, select PLAT=juno.
- PLATFORM_REPORT_CTX_MEM_USE: Reports the context memory allocated for each core as well as the global context. The data includes the memory used by each world and each privileged exception level. This build option is applicable only for ARCH=aarch64 builds. The default value is 0.
- **PLAT_EXTRA_LD_SCRIPT: Allows the platform to include a custom LD script**
  snippet for any custom sections that cannot be expressed otherwise. Defaults to 0.
- PRELOADED_BL33_BASE: This option enables booting a preloaded BL33 image instead of the normal boot flow. When defined, it must specify the entry point address for the preloaded BL33 image. This option is incompatible with EL3_PAYLOAD_BASE. If both are defined, EL3_PAYLOAD_BASE has priority over PRELOADED_BL33_BASE.
- PRESERVE_DSU_PMU_REGS: This options when enabled allows the platform to save/restore the DynamIQ Shared Unit's(DSU) Performance Monitoring Unit(PMU) registers when the cluster goes through a power cycle. This is disabled by default and platforms that require this feature have to enable them.
- PROGRAMMABLE_RESET_ADDRESS: This option indicates whether the reset vector address can be programmed or is fixed on the platform. It can take either 0 (fixed) or 1 (programmable). Default is 0. If the platform has a programmable reset address, it is expected that a CPU will start executing code directly at the right address, both on a cold and warm reset. In this case, there is no need to identify the entrypoint on boot and the boot path can be optimised. The plat_get_my_entrypoint() platform porting interface does not need to be implemented in this case.
- PSCI_EXTENDED_STATE_ID: As per PSCI1.0 Specification, there are 2 formats possible for the PSCI power-state parameter: original and extended State-ID formats. This flag if set to 1, configures the generic PSCI layer to use the extended format. The default value of this flag is 0, which means by default the original power-state format is used by the PSCI implementation. This flag should be specified by the platform makefile and it governs the return value of PSCI_FEATURES API for CPU_SUSPEND smc function id. When this option is enabled on Arm platforms, the option ARM_RECOM_STATE_ID_ENC needs to be set to 1 as well.
- PSCI_OS_INIT_MODE: Boolean flag to enable support for optional PSCI OS-initiated mode. This option defaults to 0.
- ARCH_FEATURE_AVAILABILITY: Boolean flag to enable support for the optional SMCCC_ARCH_FEATURE_AVAILABILITY call. This option implicitly interacts with IMPDEF_SYSREG_TRAP and software emulation. This option defaults to 0.
- RESET_TO_BL31: Enable BL31 entrypoint as the CPU reset vector instead of the BL1 entrypoint. It can take the value 0 (CPU reset to BL1 entrypoint) or 1 (CPU reset to BL31 entrypoint). The default value is 0.
- RESET_TO_SP_MIN: SP_MIN is the minimal AArch32 Secure Payload provided in TF-A. This flag configures SP_MIN entrypoint as the CPU reset vector instead of the BL1 entrypoint. It can take the value 0 (CPU reset to BL1 entrypoint) or 1 (CPU reset to SP_MIN entrypoint). The default value is 0.
- RME_GPT_BITLOCK_BLOCK: This defines the block size (in number of 512MB blocks) covered by a single bit of the bitlock structure during RME GPT operations. The lower the block size, the better opportunity for parallelising GPT operations but at the cost of more bits being needed for the bitlock

structure. This numeric parameter can take the values from 0 to 512 and must be a power of 2. The value of 0 is special and and it chooses a single spinlock for all GPT L1 table entries. Default value is 1 which corresponds to block size of 512MB per bit of bitlock structure.

- `RME_GPT_MAX_BLOCK`: Numeric value in MB to define the maximum size of supported contiguous blocks in GPT Library. This parameter can take the values 0, 2, 32 and 512. Setting this value to 0 disables use of Contigious descriptors. Default value is 512.
- `ROT_KEY`: This option is used when `GENERATE_COT=1`. It specifies a file that contains the ROT private key in PEM format or a PKCS11 URI and enforces public key hash generation. If `SAVE_KEYS=1`, only a file is accepted and it will be used to save the key.
- `SAVE_KEYS`: This option is used when `GENERATE_COT=1`. It tells the certificate generation tool to save the keys used to establish the Chain of Trust. Allowed options are '0' or '1'. Default is '0' (do not save).
- `SCP_BL2`: Path to SCP_BL2 image in the host file system. This image is optional. If a SCP_BL2 image is present then this option must be passed for the `fip` target.
- `SCP_BL2_KEY`: This option is used when `GENERATE_COT=1`. It specifies a file that contains the SCP_BL2 private key in PEM format or a PKCS11 URI. If `SAVE_KEYS=1`, only a file is accepted and it will be used to save the key.
- `SCP_BL2U`: Path to SCP_BL2U image in the host file system. This image is optional. It is only needed if the platform makefile specifies that it is required in order to build the `fwu_fip` target.
- `SDEI_SUPPORT`: Setting this to `1` enables support for Software Delegated Exception Interface to BL31 image. This defaults to `0`.

  When set to `1`, the build option `EL3_EXCEPTION_HANDLING` must also be set to `1`.
- `SEPARATE_CODE_AND_RODATA`: Whether code and read-only data should be isolated on separate memory pages. This is a trade-off between security and memory usage. See "Isolating code and read-only data on separate memory pages" section in *Firmware Design*. This flag is disabled by default and affects all BL images.
- `SEPARATE_NOBITS_REGION`: Setting this option to `1` allows the NOBITS sections of BL31 (.bss, stacks, page tables, and coherent memory) to be allocated in RAM discontiguous from the loaded firmware image. When set, the platform is expected to provide definitions for `BL31_NOBITS_BASE` and `BL31_NOBITS_LIMIT`. When the option is `0` (the default), NOBITS sections are placed in RAM immediately following the loaded firmware image.
- `SEPARATE_BL2_NOLOAD_REGION`: Setting this option to `1` allows the NOLOAD sections of BL2 (.bss, stacks, page tables) to be allocated in RAM discontiguous from loaded firmware images. When set, the platform need to provide definitions of `BL2_NOLOAD_START` and `BL2_NOLOAD_LIMIT`. This flag is disabled by default and NOLOAD sections are placed in RAM immediately following the loaded firmware image.
- `SEPARATE_BL2_FIP`: This option enables the separation of the BL2 FIP image from the main FIP image. When this option is enabled, the BL2 FIP image is built as a separate FIP image. The default value is 0.
- **`SEPARATE_SIMD_SECTION`: Setting this option to `1` allows the SIMD context**

  data structures to be put in a dedicated memory region as decided by platform integrator. Default value is 0 which means the SIMD context is put in BSS section of EL3 firmware.

- SMC_PCI_SUPPORT: This option allows platforms to handle PCI configuration access requests via a standard SMCCC defined in DEN0115. When combined with UEFI+ACPI this can provide a certain amount of OS forward compatibility with newer platforms that aren't ECAM compliant.
- SPD: Choose a Secure Payload Dispatcher component to be built into TF-A. This build option is only valid if ARCH=aarch64. The value should be the path to the directory containing the SPD source, relative to services/spd/; the directory is expected to contain a makefile called <spd-value>.mk. The SPM Dispatcher standard service is located in services/std_svc/spmd and enabled by SPD=spmd. The SPM Dispatcher cannot be enabled when the SPM_MM option is enabled.
- SPIN_ON_BL1_EXIT: This option introduces an infinite loop in BL1. It can take either 0 (no loop) or 1 (add a loop). 0 is the default. This loop stops execution in BL1 just before handing over to BL31. At this point, all firmware images have been loaded in memory, and the MMU and caches are turned off. Refer to the "Debugging options" section for more details.
- SPMC_AT_EL3 : This boolean option is used jointly with the SPM Dispatcher option (SPD=spmd). When enabled (1) it indicates the SPMC component runs at the EL3 exception level. The default value is 0 (disabled). This configuration supports pre-Armv8.4 platforms (aka not implementing the FEAT_SEL2 extension).
- SPMC_AT_EL3_SEL0_SP : Boolean option to enable SEL0 SP load support when SPMC_AT_EL3 is enabled. The default value if 0 (disabled). This option cannot be enabled (1) when (SPMC_AT_EL3) is disabled.
- SPMC_OPTEE : This boolean option is used jointly with the SPM Dispatcher option (SPD=spmd) and with SPMD_SPM_AT_SEL2=0 to indicate that the SPMC at S-EL1 is OP-TEE and an OP-TEE specific loading mechanism should be used.
- SPMD_SPM_AT_SEL2 : This boolean option is used jointly with the SPM Dispatcher option (SPD=spmd). When enabled (1) it indicates the SPMC component runs at the S-EL2 exception level provided by the FEAT_SEL2 extension. This is the default when enabling the SPM Dispatcher. When disabled (0) it indicates the SPMC component runs at the S-EL1 execution state or at EL3 if SPMC_AT_EL3 is enabled. The latter configurations support pre-Armv8.4 platforms (aka not implementing the FEAT_SEL2 extension).
- SPM_MM : Boolean option to enable the Management Mode (MM)-based Secure Partition Manager (SPM) implementation. The default value is 0 (disabled). This option cannot be enabled (1) when SPM Dispatcher is enabled (SPD=spmd).
- SP_LAYOUT_FILE: Platform provided path to JSON file containing the description of secure partitions. The build system will parse this file and package all secure partition blobs into the FIP. This file is not necessarily part of TF-A tree. Only available when SPD=spmd.
- SP_MIN_WITH_SECURE_FIQ: Boolean flag to indicate the SP_MIN handles secure interrupts (caught through the FIQ line). Platforms can enable this directive if they need to handle such interruption. When enabled, the FIQ are handled in monitor mode and non secure world is not allowed to mask these events. Platforms that enable FIQ handling in SP_MIN shall implement the api sp_min_plat_fiq_handler(). The default value is 0.
- SVE_VECTOR_LEN: SVE vector length to configure in ZCR_EL3. Platforms can configure this if they need to lower the hardware limit, for example due to asymmetric configuration or limitations of software

run at lower ELs. The default is the architectural maximum of 2048 which should be suitable for most configurations, the hardware will limit the effective VL to the maximum physically supported VL.

- `TRNG_SUPPORT`: Setting this to `1` enables support for True Random Number Generator Interface to BL31 image. This defaults to `0`.
- `TRUSTED_BOARD_BOOT`: Boolean flag to include support for the Trusted Board Boot feature. When set to '1', BL1 and BL2 images include support to load and verify the certificates and images in a FIP, and BL1 includes support for the Firmware Update. The default value is '0'. Generation and inclusion of certificates in the FIP and FWU_FIP depends upon the value of the `GENERATE_COT` option.

> **Warning:** This option depends on `CREATE_KEYS` to be enabled. If the keys already exist in disk, they will be overwritten without further notice.

- `TRUSTED_WORLD_KEY`: This option is used when `GENERATE_COT=1`. It specifies a file that contains the Trusted World private key in PEM format or a PKCS11 URI. If `SAVE_KEYS=1`, only a file is accepted and it will be used to save the key.
- `TSP_INIT_ASYNC`: Choose BL32 initialization method as asynchronous or synchronous, (see "Initializing a BL32 Image" section in *Firmware Design*). It can take the value 0 (BL32 is initialized using synchronous method) or 1 (BL32 is initialized using asynchronous method). Default is 0.
- `TSP_NS_INTR_ASYNC_PREEMPT`: A non zero value enables the interrupt routing model which routes non-secure interrupts asynchronously from TSP to EL3 causing immediate preemption of TSP. The EL3 is responsible for saving and restoring the TSP context in this routing model. The default routing model (when the value is 0) is to route non-secure interrupts to TSP allowing it to save its context and hand over synchronously to EL3 via an SMC.

> **Note:** When `EL3_EXCEPTION_HANDLING` is `1`, `TSP_NS_INTR_ASYNC_PREEMPT` must also be set to `1`.

- `TS_SP_FW_CONFIG`: DTC build flag to include Trusted Services (Crypto and internal-trusted-storage) as SP in tb_fw_config device tree.
- `TWED_DELAY`: Numeric value to be set in order to delay the trapping of WFE instruction. `ENABLE_FEAT_TWED` build option must be enabled to set this delay. It can take values in the range (0-15). Default value is `0` and based on this value, 2^(TWED_DELAY + 8) cycles will be delayed. Platforms need to explicitly update this value based on their requirements.
- `USE_ARM_LINK`: This flag determines whether to enable support for ARM linker. When the `LINKER` build variable points to the armlink linker, this flag is enabled automatically. To enable support for armlink, platforms will have to provide a scatter file for the BL image. Currently, Tegra platforms use the armlink support to compile BL3-1 images.
- `USE_COHERENT_MEM`: This flag determines whether to include the coherent memory region in the BL memory map or not (see "Use of Coherent memory in TF-A" section in *Firmware Design*). It can take the value 1 (Coherent memory region is included) or 0 (Coherent memory region is excluded). Default is 1.

- `USE_KERNEL_DT_CONVENTION`: When this option is enabled, the hardware device tree is passed to BL33 using register x0, aligning with the expectations of the Linux kernel on Arm platforms. If this option is disabled, a different register, typically x1, may be used instead. This build option is not necessary when firmware handoff is active (that is, when TRANSFER_LIST=1 is set), and it will be removed once all platforms have transitioned to that convention.
- `USE_DSU_DRIVER`: This flag enables DSU (DynamIQ Shared Unit) driver. The DSU driver allows save/restore of DSU PMU registers through `PRESERVE_DSU_PMU_REGS` build option, provides access to PMU registers at EL1 and allows platforms to configure powerdown and power settings of DSU.
- `ARM_IO_IN_DTB`: This flag determines whether to use IO based on the firmware configuration framework. This will move the io_policies into a configuration device tree, instead of static structure in the code base.
- `COT_DESC_IN_DTB`: This flag determines whether to create COT descriptors at runtime using fconf. If this flag is enabled, COT descriptors are statically captured in tb_fw_config file in the form of device tree nodes and properties. Currently, COT descriptors used by BL2 are moved to the device tree and COT descriptors used by BL1 are retained in the code base statically.
- `SDEI_IN_FCONF`: This flag determines whether to configure SDEI setup in runtime using firmware configuration framework. The platform specific SDEI shared and private events configuration is retrieved from device tree rather than static C structures at compile time. This is only supported if SDEI_SUPPORT build flag is enabled.
- `SEC_INT_DESC_IN_FCONF`: This flag determines whether to configure Group 0 and Group1 secure interrupts using the firmware configuration framework. The platform specific secure interrupt property descriptor is retrieved from device tree in runtime rather than depending on static C structure at compile time.
- `USE_ROMLIB`: This flag determines whether library at ROM will be used. This feature creates a library of functions to be placed in ROM and thus reduces SRAM usage. Refer to *Library at ROM* for further details. Default is 0.
- `V`: Verbose build. If assigned anything other than 0, the build commands are printed. Default is 0.
- `VERSION_STRING`: String used in the log output for each TF-A image. Defaults to a string formed by concatenating the version number, build type and build string.
- `W`: Warning level. Some compiler warning options of interest have been regrouped and put in the root Makefile. This flag can take the values 0 to 3, each level enabling more warning options. Default is 0.

  This option is closely related to the `E` option, which enables `-Werror`.

  - `W=0` (default)

    Enables a wide assortment of warnings, most notably `-Wall` and `-Wextra`, as well as various bad practices and things that are likely to result in errors. Includes some compiler specific flags. No warnings are expected at this level for any build.

  - `W=1`

    Enables warnings we want the generic build to include but are too time consuming to fix at the moment. It re-enables warnings taken out for `W=0` builds (a few of the `-Wextra` additions). This level is expected to eventually be merged into `W=0`. Some warnings are expected on some builds, but new contributions should not introduce new ones.

  - W=2 (recommended)

    Enables warnings we want the generic build to include but cannot be enabled due to external libraries. This level is expected to eventually be merged into W=0. Lots of warnings are expected, primarily from external libraries like zlib and compiler-rt, but new controbutions should not introduce new ones.

  - W=3

    Enables warnings that are informative but not necessary and generally too verbose and frequently ignored. A very large number of warnings are expected.

  The exact set of warning flags depends on the compiler and TF-A warning level, however they are all succinctly set in the top-level Makefile. Please refer to the GCC or Clang documentation for more information on the individual flags.

- WARMBOOT_ENABLE_DCACHE_EARLY : Boolean option to enable D-cache early on the CPU after warm boot. This is applicable for platforms which do not require interconnect programming to enable cache coherency (eg: single cluster platforms). If this option is enabled, then warm boot path enables D-caches immediately after enabling MMU. This option defaults to 0.

- ERRATA_SPECULATIVE_AT: This flag determines whether to enable AT speculative errata workaround or not. It accepts 2 values: 1 and 0. The default value of this flag is 0.

  AT speculative errata workaround disables stage1 page table walk for lower ELs (EL1 and EL0) in EL3 so that AT speculative fetch at any point produces either the correct result or failure without TLB allocation.

  This boolean option enables errata for all below CPUs.

Errata	CPU	Workaround Define
1165522	Cortex-A76	ERRATA_A76_1165522
1319367	Cortex-A72	ERRATA_A72_1319367
1541130	Cortex-A65	ERRATA_A65_1541130
1638571	Cortex-A65AE	ERRATA_A65AE_1638571
1319537	Cortex-A57	ERRATA_A57_1319537
1530923	Cortex-A55	ERRATA_A55_1530923
1530924	Cortex-A53	ERRATA_A53_1530924

  Note: This option is enabled by build only if platform sets any of above defines mentioned in 'Workaround Define' column in the table. If this option is enabled for the EL3 software then EL2 software also must implement this workaround due to the behaviour of the errata mentioned in new SDEN document which will get published soon.

- RAS_TRAP_NS_ERR_REC_ACCESS: Boolean option to disable FEAT_RAS access to ERR and ERX registers from lower ELs. When set, all accesses will be trapped to EL3 and emulated as RAZ/WI. Default value is 0.

- OPENSSL_DIR: This option is used to provide the path to a directory on the host machine where a custom installation of OpenSSL is located, which is used to build the certificate generation, firmware encryption and FIP tools. If this option is not set, the default OS installation will be used.

- `USE_SP804_TIMER`: Use the SP804 timer instead of the Generic Timer for functions that wait for an arbitrary time length (udelay and mdelay). The default value is 0.
- `CONDITIONAL_CMO`: Boolean option to enable call to platform-defined routine `plat_can_cmo` which will return zero if cache management operations should be skipped and non-zero otherwise. By default, this option is disabled which means platform hook won't be checked and CMOs will always be performed when related functions are called.
- `ERRATA_ABI_SUPPORT`: Boolean option to enable support for Errata management firmware interface for the BL31 image. By default its disabled (`0`).
- `ERRATA_NON_ARM_INTERCONNECT`: Boolean option to enable support for the errata mitigation for platforms with a non-arm interconnect using the errata ABI. By default its disabled (`0`).
- `ENABLE_CONSOLE_GETC`: Boolean option to enable *getc()* feature in console driver(s). By default it is disabled (`0`) because it constitutes an attack vector into TF-A by potentially allowing an attacker to inject arbitrary data. This option should only be enabled on a need basis if there is a use case for reading characters from the console.

2.5.2 CPU Architecture feature options

Meta options

- `ARCH` : Choose the target build architecture for TF-A. It can take either `aarch64` or `aarch32` as values. By default, it is defined to `aarch64`.
- `ARM_ARCH_MAJOR`: The major version of Arm Architecture to target when compiling TF-A. Its value must be numeric, and defaults to 8 . See also, *Armv8 Architecture Extensions* and *Armv7 Architecture Extensions* in *Firmware Design*.
- `ARM_ARCH_MINOR`: The minor version of Arm Architecture to target when compiling TF-A. Its value must be a numeric, and defaults to 0. See also, *Armv8 Architecture Extensions* in *Firmware Design*.
- `ARM_ARCH_FEATURE`: Optional Arm Architecture build option which specifies one or more feature modifiers. This option has the form `[no]feature+...` and defaults to `none`. It translates into compiler option `-march=armvX[.Y]-a+[no]feature+...`. See compiler's documentation for the list of supported feature modifiers.

CPU feature enables

This section describes flags that control the enablement of Architectural features in the A class Armv8.0+ architecture. Unless stated otherwise, these flags support *The ENABLE_FEAT mechanism* and take numeric values from 0 to 2.

- `FEATURE_DETECTION`: Boolean option to enable the architectural features verification mechanism. This is a debug feature that compares the architectural features enabled through the feature specific build flags (ENABLE_FEAT_xxx) with the features actually available on the CPU running, and reports any discrepancies.

  It is expected that this feature is only used for flexible platforms like software emulators, or for hardware platforms at bringup time, to verify that the configured feature set matches the CPU. Default value is `0`.

- `CTX_INCLUDE_NEVE_REGS`: When set, causes the Armv8.4-NV registers to be saved/restored when entering/exiting an EL2 execution context. Default value is 0.
- `CTX_INCLUDE_PAUTH_REGS`: Enables Pointer Authentication for Secure world. This will cause the ARMv8.3-PAuth registers to be included when saving and restoring the CPU context as part of world switch. Automatically enabled when `BRANCH_PROTECTION` is enabled. Default value is 0.

  Note that Pointer Authentication is enabled for Non-secure world irrespective of the value of this flag if the CPU supports it. Alternatively, when `BRANCH_PROTECTION` is enabled, this flag is superseded.
- `CTX_INCLUDE_AARCH32_REGS` : Boolean option that, when set to 1, will cause the AArch32 system registers to be included when saving and restoring the CPU context. The option must be set to 0 for AArch64-only platforms (that is on hardware that does not implement AArch32, or at least not at EL1 and higher ELs). Default value is 1.
- `CTX_INCLUDE_FPREGS`: Boolean option that, when set to 1, will cause the FP registers to be included when saving and restoring the CPU context. Default is 0.
- `CTX_INCLUDE_MPAM_REGS`: Boolean option that, when set to 1, will cause the Memory System Resource Partitioning and Monitoring (MPAM) registers to be included when saving and restoring the CPU context. Default is '0'.
- `CTX_INCLUDE_SVE_REGS`: Boolean option that, when set to 1, will cause the SVE registers to be included when saving and restoring the CPU context. Note that this build option requires `ENABLE_SVE_FOR_SWD` to be enabled. In general, it is recommended to perform SVE context management in lower ELs and skip in EL3 due to the additional cost of maintaining large data structures to track the SVE state. Hence, the default value is 0.
- `DISABLE_MTPMU`: Disables `FEAT_MTPMU` (Multi Threaded PMU). `FEAT_MTPMU` is an optional feature available on Armv8.6 onwards. Default is `0`.
- `ENABLE_FEAT_AMU`: Enables Activity Monitor Unit extensions. This is an optional architectural feature available on v8.4 onwards. Some v8.2 implementations also implement an AMU and this option can be used to enable this feature on those systems as well. The default is 0.
- `AMU_RESTRICT_COUNTERS`: Register reads to the group 1 counters will return zero at all but the highest implemented exception level. External memory-mapped debug accesses are unaffected by this control. The default value is 1 for all platforms.
- `ENABLE_FEAT_AMUv1p1`: Enables the `FEAT_AMUv1p1` extension. `FEAT_AMUv1p1` is an optional feature available on Arm v8.6 onwards. Default value is `0`.
- **`ENABLE_FEAT_CLRBHB`: Enables the CLRBHB instruction.**

  Clear Branch History clears the branch history for the current context to the extent that branch history information created before the CLRBHB instruction cannot be used by code. This is an optional architectural feature available on v8.0 onwards and is a mandatory feature from v8.9 onwards. Default value is `0`.
- `ENABLE_FEAT_CPA2`: Enables the `FEAT_CPA2` extension. It enables checked pointer arithmetic in EL3, which will result in address faults in the event that a pointer arithmetic overflow error occurs. This is an optional feature starting from Arm v9.4. Default value is `0`.
- `ENABLE_FEAT_CSV2_2`: Enables the `FEAT_CSV2_2` extension. It allows access to the SCXTNUM_EL2 (Software Context Number) register during EL2 context save/restore operations.

FEAT_CSV2_2 is an optional feature available on Arm v8.0 onwards. Default value is 0.

- ENABLE_FEAT_CSV2_3: Enables support for FEAT_CSV2_3 extension. This feature is supported in AArch64 state only and is an optional feature available in Arm v8.0 implementations. FEAT_CSV2_3 implies the implementation of FEAT_CSV2_2. Default value is 0.
- ENABLE_FEAT_CRYPTO: Enables the FEAT_CRYPTO extension. It allows using the SIMD crypto extension AES, SHA1 and SHA2 instructions for mbedtls HASH256, which speeds up the authentication process of the subsequent images in BL1 and BL2. FEAT_CRYPTO is an optional feature available on Arm v8 onwards. Setting this flag to 2 has the same effect as 0 since there is no way to do runtime checks. Default value is 0.
- ENABLE_FEAT_CRYPTO_SHA3: Enables the FEAT_CRYPTO extension. It allows using the SIMD crypto extension SHA3 instructions for mbedtls HASH384 and HASH512, which speeds up the authentication process of the subsequent images in BL1 and BL2. FEAT_CRYPTO_SHA3 is an optional feature available on Arm v8.2 onwards. Setting this flag to 2 has the same effect as 0 since there is no way to do runtime checks. Default value is 0.
- ENABLE_FEAT_DEBUGV8P9: Enables FEAT_DEBUGV8P9 extension which allows the ability to implement more than 16 breakpoints and/or watchpoints. This feature is mandatory from v8.9 and is optional from v8.8. Default value is 0.
- ENABLE_FEAT_DIT: Enables FEAT_DIT (Data Independent Timing) extension. It allows setting the DIT bit of PSTATE in EL3. FEAT_DIT is a mandatory architectural feature and is enabled from v8.4 and upwards. Default value is 0.
- ENABLE_FEAT_ECV: Enables support for the Enhanced Counter Virtualization feature, allowing for access to the CNTPOFF_EL2 (Counter-timer Physical Offset register) during EL2 to EL3 context save/restore operations. Its a mandatory architectural feature and is enabled from v8.6 and upwards. Default value is 0.
- ENABLE_FEAT_FPMR: Enables support for Floating Point Mode Register feature, allowing access to the FPMR register. FPMR register controls the behaviors of FP8 instructions. It is an optional architectural feature from v9.2 and upwards. This flag can take value of 0 to 2, to align with the FEATURE_DETECTION mechanism. Default value is 0.
- ENABLE_FEAT_FGT: Enables support for FGT (Fine Grain Traps) feature allowing for access to the HDFGRTR_EL2 (Hypervisor Debug Fine-Grained Read Trap Register) during EL2 to EL3 context save/restore operations. Its a mandatory architectural feature and is enabled from v8.6 and upwards. Default value is 0.
- ENABLE_FEAT_FGT2: Enables support for FGT2 (Fine Grain Traps 2) feature allowing for access to Fine-grained trap 2 registers during EL2 to EL3 context save/restore operations. Its an optional architectural feature and is available from v8.8 and upwards. Default value is 0.
- ENABLE_FEAT_FGWTE3: Enables support for Fine Grained Write Trap EL3 (FEAT_FGWTE3), a feature that allows EL3 to restrict overwriting certain EL3 registers after boot. This lockdown is established by setting individual trap bits for system registers that are not expected to be overwritten after boot. This feature is an optional architectural feature and is available from Armv9.4 onwards. The default value is 0.

Note: This feature currently traps access to all EL3 registers in FGWTE3_EL3, except for MDCR_EL3, MPAM3_EL3, TPIDR_EL3`` (when ``CRASH_REPORTING=1), and SCTLR_EL3`` (when ``HW_ASSISTED_COHERENCY=0). If additional traps need to be disabled for specific platforms, please contact the Arm team on TF-A public mailing list.

- ENABLE_FEAT_HDBSS: Enables support for HDBSS (Hardware Dirty state tracking structure) by setting SCR_EL3.HDBSSEn for NS world. This is an optional architectural feature and is available from v9.4 and upwards. Default value is 0.
- ENABLE_FEAT_HACDBS: Enables support for HACDBS (Hardware accelerator for cleaning Dirty state) by setting SCR_EL3.HACDBSEn for NS world. This is an optional architectural feature and is available from v9.4 and upwards. Default value is 0.
- ENABLE_FEAT_HCX: Sets the bit SCR_EL3.HXEn in EL3 to allow access to HCRX_EL2 (extended hypervisor control register) from EL2 as well as adding HCRX_EL2 to the EL2 context save/restore operations. Its a mandatory architectural feature and is enabled from v8.7 and upwards. Default value is 0.
- ENABLE_FEAT_IDTE3: Sets SCR_EL3.TID3/TID5 bits, which enables trapping of ID register reads by lower ELs to EL3. This allows EL3 to control the feature visibility to lower ELs by returning a sanitized value based on current feature enablement status. Hypervisors are expected to cache ID register during their boot stage. Default value is 0. This feature is EXPERIMENTAL.

Note: This feature traps all lower EL accesses to Group 3 and Group 5 ID registers to EL3. This can incur a performance impact and platforms should enable them only if they have a specific need.

- ENABLE_FEAT_MOPS: Enables FEAT_MOPS (Standardization of memory operations) when INIT_UNUSED_NS_EL2=1. This feature is mandatory from v8.8 and enabling of FEAT_MOPS does not require any settings from EL3 as the controls are present in EL2 registers (HCRX_EL2.{MSCEn,MCE2} and SCTLR_EL2.MSCEn) and in most configurations we expect EL2 to be present. But in case of INIT_UNUSED_NS_EL2=1 , EL3 should configure the EL2 registers. Default value is 0.
- ENABLE_FEAT_MTE2: Enables Memory Tagging Extension2 if the platform wants to use this feature and MTE2 is enabled at ELX. Default value is 0.
- ENABLE_FEAT_PAN: Enables the FEAT_PAN (Privileged Access Never) extension. FEAT_PAN adds a bit to PSTATE, generating a permission fault for any privileged data access from EL1/EL2 to virtual memory address, accessible at EL0, provided (HCR_EL2.E2H=1). It is a mandatory architectural feature and is enabled from v8.1 and upwards. Default value is 0.
- BRANCH_PROTECTION: Enables ARMv8.3 Pointer Authentication and ARMv8.5 Branch Target Identification support for TF-A BL images themselves. If enabled, it is needed to use a compiler that supports the option -mbranch-protection. The value of the -march (via ARM_ARCH_MINOR and ARM_ARCH_MAJOR) option will control which instructions will be emitted (HINT space or not). Selects the branch protection features to use:
- 0: Default value turns off all types of branch protection (FEAT_STATE_DISABLED)
- 1: Enables all types of branch protection features
- 2: Return address signing to its standard level

- 3: Extend the signing to include leaf functions
- 4: Turn on branch target identification mechanism
- 5: Enables all types of branch protection features, only if present in hardware (FEAT_STATE_CHECK).

  The table below summarizes `BRANCH_PROTECTION` values, GCC compilation options and resulting PAuth/BTI features.

Value	GCC option	PAuth	BTI
0	none	N	N
1	standard	Y	Y
2	pac-ret	Y	N
3	pac-ret+leaf	Y	N
4	bti	N	Y
5	dynamic	Y	Y

  This option defaults to 0. Note that Pointer Authentication is enabled for Non-secure world irrespective of the value of this option if the CPU supports it.

- `ENABLE_FEAT_PAUTH_LR`: Enables the `FEAT_PAUTH_LR` extension. `FEAT_PAUTH_LR` is an optional feature available from Arm v9.4 onwards. This feature requires PAUTH to be enabled via the `BRANCH_PROTECTION` flag. Default value is `0`.
- `ENABLE_FEAT_RNG`: Enables the `FEAT_RNG` extension. `FEAT_RNG` is an optional feature available on Arm v8.5 onwards. Default value is `0`.
- `ENABLE_FEAT_RNG_TRAP`: Enables the `FEAT_RNG_TRAP` extension. This feature is only supported in AArch64 state. Default value is `0`. `FEAT_RNG_TRAP` is an optional feature from Armv8.5 onwards.
- `ENABLE_FEAT_SB`: Lets the TF-A code use the `FEAT_SB` (Speculation Barrier) instruction `FEAT_SB` is an optional feature and defaults to `0` for pre-Armv8.5 CPUs, but is mandatory for Armv8.5 or later CPUs. It is enabled from v8.5 and upwards and if needed can be overidden from platforms explicitly. Default value is `0`.
- `ENABLE_FEAT_SEL2`: Enables the `FEAT_SEL2` (Secure EL2) extension. `FEAT_SEL2` is a mandatory feature available on Arm v8.4. Default is `0`.
- `ENABLE_FEAT_TWED`: Enables the `FEAT_TWED` (Delayed trapping of WFE Instruction) extension. `FEAT_TWED` is a optional feature available on Arm v8.6. Default is `0`.

  > When `ENABLE_FEAT_TWED` is set to `1`, WFE instruction trapping gets delayed by the amount of value in `TWED_DELAY`.

- `ENABLE_FEAT_VHE`: Enables the `FEAT_VHE` (Virtualization Host Extensions) extension. It allows access to CONTEXTIDR_EL2 register during EL2 context save/restore operations.``FEAT_VHE`` is a mandatory architectural feature and is enabled from v8.1 and upwards. Default value is `0`.
- `ENABLE_FEAT_TCR2`: Sets the bit SCR_EL3.ENTCR2 in EL3 to allow access to TCR2_EL2 (extended translation control) from EL2 as well as adding TCR2_EL2 to the EL2 context save/restore operations. Its a mandatory architectural feature and is enabled from v8.9 and upwards. Default value is `0`.

- `ENABLE_FEAT_S2PIE`: Enables support for FEAT_S2PIE at EL2 and below, and context switch relevant registers. Default value is `0`.
- `ENABLE_FEAT_S1PIE`: Enables support for FEAT_S1PIE at EL2 and below, and context switch relevant registers. Default value is `0`.
- `ENABLE_FEAT_S2POE`: Enables support for FEAT_S2POE at EL2 and below, and context switch relevant registers. Default value is `0`.
- `ENABLE_FEAT_S1POE`: Enables support for FEAT_S1POE at EL2 and below, and context switch relevant registers. Default value is `0`.
- `ENABLE_FEAT_GCS`: Sets the bit SCR_EL3.GCSEn in EL3 to allow use of Guarded Control Stack from EL2 as well as adding the GCS registers to the EL2 context save/restore operations. Default value is `0`.
- `ENABLE_FEAT_GCIE`: Boolean value to enable support for the GICv5 CPU interface (see `USE_GIC_DRIVER` for the IRI). GICv5 and GICv3 are mutually exclusive so the `ENABLE_FEAT` mechanism is currently not supported. Default value is `0`.
- `ENABLE_FEAT_THE`: Enables support for FEAT_THE (Translation Hardening Extension) at EL2 and below, setting the bit SCR_EL3.RCWMASKEn in EL3 to allow access to RCWMASK_EL1 and RCWSMASK_EL1 registers and context switch them. Its an optional architectural feature and is available from v8.8 and upwards. Default value is `0`.
- `ENABLE_FEAT_SCTLR2`: Enables support for FEAT_SCTLR2 (Extension to SCTLR_ELx) at EL2 and below, setting the bit SCR_EL3.SCTLR2En in EL3 to allow access to SCTLR2_ELx registers and context switch them. This feature is OPTIONAL from Armv8.0 implementations and mandatory in Armv8.9 implementations. Default value is `0`.
- `ENABLE_FEAT_STEP2`: Enables support for FEAT_STEP2 by setting `MDCR_EL3.EnSTEPOP` so that lower ELs can access `MDSTEPOP_EL1`. This feature is optional from Armv9.4 implementations and is mandatory in Armv9.5 implementations. Defaults value is `0`.
- `ENABLE_FEAT_D128`: Enables support for FEAT_D128 at EL2 and below, setting the bit SCT_EL3.D128En in EL3 to allow access to 128 bit version of system registers like PAR_EL1, TTBR0_EL1, TTBR1_EL1, TTBR0_EL2, TTBR1_EL2, TTBR0_EL12, TTBR1_EL12 , VTTBR_EL2, RCWMASK_EL1, and RCWSMASK_EL1. Its an optional architectural feature and is available from 9.3 and upwards. Default value is `0`.
- `ENABLE_FEAT_UINJ`: Enables FEAT_UINJ support, which is hardware based injection of undefined instruction exceptions. The objective of this feature is to provide higher privilege software with a future proofed mechanism to inject an Undefined Instruction exception into lower privilege software. It is an optional architectural feature from v9.0 and mandatory from v9.6. This flag can take value of 0 to 2, to align with the `FEATURE_DETECTION` mechanism. Default value is `0`.
- `ENABLE_FEAT_MEC`: Enables support for the ARMv9.2 Memory Encryption Contexts (MEC). MEC supports multiple encryption contexts for Realm security state and only one encryption context for the rest of the security states. Default value is 0.
- `ENABLE_FEAT_MORELLO` : Enables the Morello capability aware firmware. This option is experimental and supported only with LLVM CLANG toolchain and not with GCC toolchain. Capability awareness is currently enabled only in BL31 firmware and not in other firmware types of trusted firmware. Enabling this on regular AARCH64 system might not work.

- `ENABLE_FEAT_RAS`: Enables Armv8.2 RAS features. RAS features are an optional extension for pre-Armv8.2 CPUs, but are mandatory for Armv8.2 or later CPUs. NOTE: This flag enables use of IESB capability to reduce entry latency into EL3 even when RAS error handling is not performed on the platform. Hence this flag is recommended to be turned on Armv8.2 and later CPUs. The default is 0.
- `ENABLE_FEAT_RME`: Enables support for the ARMv9 Realm Management Extension. Default value is 0. This flag solely controls the architectural bits of RME, to let TF-A run in the "root" physical address space and this will setup Granule Protection Tables (GPT). Also this will make BL2 run in EL3, so it has access to the new root address space. For deploying a Realm-EL2 payload (RMM), also set `ENABLE_RMM` and provide an RMM image file.
- `ENABLE_SPE_FOR_NS` : Enables Statistical Profiling extensions. This is an optional architectural feature for AArch64. The default is 2 but is automatically disabled when the target architecture is AArch32.
- `ENABLE_FEAT_SPEV1P5`: Numeric value to enable `FEAT_SPEv1p5` related Features (`FEAT_SPE_EXC` and `FEAT_SPE_nVM`) support and requires `ENABLE_SPE_FOR_NS` to be enabled. This is an optional architectural feature for AArch64. This flag can take the values 0 to 2, to align with the `ENABLE_FEAT` mechanism. The default is `0`.
- `ENABLE_SVE_FOR_NS`: Enables Scalable Vector Extension (SVE) for the Non-secure world only. SVE is an optional architectural feature for AArch64. At this time, this build option cannot be used on systems that have SPM_MM enabled. The default value is 2.

  Note that when SVE is enabled for the Non-secure world, access to SVE, SIMD and floating-point functionality from the Secure world is independently controlled by build option `ENABLE_SVE_FOR_SWD`. When enabling `CTX_INCLUDE_FPREGS` and `ENABLE_SVE_FOR_NS` together, it is mandatory to enable `CTX_INCLUDE_SVE_REGS`. This is to avoid corruption of the Non-secure world data in the Z-registers which are aliased by the SIMD and FP registers.
- `ENABLE_SVE_FOR_SWD`: Boolean option to enable SVE and FPU/SIMD functionality for the Secure world. SVE is an optional architectural feature for AArch64. The default is 0 and it is automatically disabled when the target architecture is AArch32.

  Note: This build flag requires `ENABLE_SVE_FOR_NS` to be enabled. When enabling `ENABLE_SVE_FOR_SWD`, a developer must carefully consider whether `CTX_INCLUDE_SVE_REGS` is also needed.

- `ENABLE_BRBE_FOR_NS`: Enables access to the branch record buffer registers from NS ELs when FEAT_BRBE is implemented. BRBE is an optional architectural feature for AArch64. The default is 0 and it is automatically disabled when the target architecture is AArch32.
- `ENABLE_TRBE_FOR_NS`: Enables access of trace buffer control registers from NS ELs, NS-EL2 or NS-EL1(when NS-EL2 is implemented but unused) when FEAT_TRBE is implemented. TRBE is an optional architectural feature for AArch64. The default is 0 and it is automatically disabled when the target architecture is AArch32.
- `USE_SPINLOCK_CAS`: Uses FEAT_LSE atomics instead of load/store exclusive instructions with spinlocks. FEAT_LSE is a mandatory feature from v8.1, however it is only architecturally guaranteed to work on "conventional memory" which may not apply to tightly coupled memory (eg. SRAM, TF-A's usual place). Platforms must check if TF-A's memory can be targetted by atomics before enabling this feature. Expected to increase performance on systems with many cores. The default is 0.

- ENABLE_SYS_REG_TRACE_FOR_NS: Enables trace system registers access from NS ELs, NS-EL2 or NS-EL1 (when NS-EL2 is implemented but unused). This feature is available if trace unit such as ETMv4.x, and ETE(extending ETM feature) is implemented. The default is 0.
- ENABLE_TRF_FOR_NS: Enables trace filter control registers access from NS ELs, NS-EL2 or NS-EL1 (when NS-EL2 is implemented but unused), if FEAT_TRF is implemented. This flag is disabled by default.
- ENABLE_SME_FOR_NS: Enables Scalable Matrix Extension (SME), SVE, and FPU/SIMD for the non-secure world only. These features share registers so are enabled together. Using this option without ENABLE_SME_FOR_SWD=1 will cause SME, SVE, and FPU/SIMD instructions in secure world to trap to EL3. Requires ENABLE_SVE_FOR_NS to be set as SME is a superset of SVE. SME is an optional architectural feature for AArch64. At this time, this build option cannot be used on systems that have SPD=spmd/SPM_MM and atempting to build with this option will fail. Default is 0.
- ENABLE_SME2_FOR_NS: Enables Scalable Matrix Extension version 2 (SME2) for the non-secure world only. SME2 is an optional architectural feature for AArch64. This should be set along with ENABLE_SME_FOR_NS=1, if not, the default SME accesses will still be trapped. Default is 0.
- ENABLE_FEAT_MPAM: Enables lower ELs to use MPAM feature. MPAM is an optional Armv8.4 extension that enables various memory system components and resources to define partitions; software running at various ELs can assign themselves to desired partition to control their performance aspects.

  When this option is set to 1 or 2, EL3 allows lower ELs to access their own MPAM registers without trapping into EL3. This option doesn't make use of partitioning in EL3, however. Platform initialisation code should configure and use partitions in EL3 as required. This option defaults to 2 since MPAM is enabled by default for NS world only. The flag is automatically disabled when the target architecture is AArch32.
- ENABLE_FEAT_MPAM_PE_BW_CTRL: This option enables Armv9.3 MPAM PE-side bandwidth controls and disables traps to EL3/EL2 (when INIT_UNUSED_NS_EL2 = 1). The default is 0.
- ENABLE_FEAT_LS64_ACCDATA: Enables access to, and save and restore the ACCDATA_EL1 system register, at EL2 and below. Default value is 0.
- ENABLE_FEAT_AIE: Enables access to the (A)MAIR2 system registers from non-secure world. Default value is 0.
- ENABLE_FEAT_PFAR: Enables access to the PFAR system registers from non-secure world. Default value is 0.

2.5.3 GIC driver options

The generic GIC driver can be included with the USE_GIC_DRIVER option. It is a numeric option that can take the following values:

- 0: generic GIC driver not enabled. Any support is entirely in platform code. Strongly discouraged for GIC based interrupt controllers.
- 1: enable the use of the generic GIC driver but do not include any files or function definitions. It is then the platform's responsibility to provide these. This is useful if the platform either

has a custom GIC implementation or an alternative interrupt controller design. Use of this option is strongly discouraged for standard GIC implementations.

- `2`: use the GICv2 driver
- `3`: use the GICv3 driver. See the next section on how to further configure it. Use this option for GICv4 implementations. Requires calling `gic_set_gicr_frames()`.
- `5`: use the EXPERIMENTAL GICv5 driver. Requires `ENABLE_FEAT_GCIE=1`.

For GIC driver versions other than `1`, deciding when to save and restore GIC context on a power domain state transition, as well as any GIC actions outside of the PSCI library's visibility are the platform's responsibility. The driver provides implementations of all necessary subroutines, they only need to be called as appropriate.

GICv3 driver options

`USE_GIC_DRIVER=3` is the preferred way of including GICv3 driver files. The old (deprecated) way of included them is using the directive: `include drivers/arm/gic/v3/gicv3.mk`

The driver can be configured with the following options set in the platform makefile:

- `GICV3_SUPPORT_GIC600`: Add support for the GIC-600 variants of GICv3. Enabling this option will add runtime detection support for the GIC-600, so is safe to select even for a GIC500 implementation. This option defaults to 0.
- **`GICV3_SUPPORT_GIC600AE_FMU`: Add support for the Fault Management Unit**
  for GIC-600 AE. Enabling this option will introduce support to initialize the FMU. Platforms should call the init function during boot to enable the FMU and its safety mechanisms. This option defaults to 0.
- `GICV3_IMPL_GIC600_MULTICHIP`: Selects GIC-600 variant with multichip functionality. This option defaults to 0
- `GICV3_OVERRIDE_DISTIF_PWR_OPS`: Allows override of default implementation of `arm_gicv3_distif_pre_save` and `arm_gicv3_distif_post_restore` functions. This is required for FVP platform which need to simulate GIC save and restore during SYSTEM_SUSPEND without powering down GIC. Default is 0.
- `GIC_ENABLE_V4_EXTN` : Enables GICv4 related changes in GICv3 driver. This option defaults to 0.
- `GIC_EXT_INTID`: When set to `1`, GICv3 driver will support extended PPI (1056-1119) and SPI (4096-5119) range. This option defaults to 0.

2.5.4 Debugging options

To compile a debug version and make the build more verbose use

```
make PLAT=<platform> DEBUG=1 V=1 all
```

AArch64 GCC 11 uses DWARF version 5 debugging symbols by default. Some tools (for example Arm-DS) might not support this and may need an older version of DWARF symbols to be emitted by GCC. This can be achieved by using the `-gdwarf-<version>` flag, with the version being set to 2, 3, 4 or 5. Setting the version to 4 is recommended for Arm-DS.

When debugging logic problems it might also be useful to disable all compiler optimizations by using `-O0`.

> **Warning:** Using `-O0` could cause output images to be larger and base addresses might need to be recalculated (see the **Memory layout on Arm development platforms** section in the *Firmware Design*).

Extra debug options can be passed to the build system by setting `CFLAGS` or `LDFLAGS`:

```
CFLAGS='-O0 -gdwarf-2'                                    \
make PLAT=<platform> DEBUG=1 V=1 all
```

Note that using `-Wl,` style compilation driver options in `CFLAGS` will be ignored as the linker is called directly.

It is also possible to introduce an infinite loop to help in debugging the post-BL2 phase of TF-A. This can be done by rebuilding BL1 with the `SPIN_ON_BL1_EXIT=1` build flag. Refer to the *Common build options* section. In this case, the developer may take control of the target using a debugger when indicated by the console output. When using Arm-DS, the following commands can be used:

```
# Stop target execution
interrupt

#
# Prepare your debugging environment, e.g. set breakpoints
#

# Jump over the debug loop
set var $AARCH64::$Core::$PC = $AARCH64::$Core::$PC + 4

# Resume execution
continue
```

2.5.5 Experimental build options

Common build options

- DICE_PROTECTION_ENVIRONMENT: Boolean flag to specify the measured boot backend when MEASURED_BOOT is enabled. The default value is 0. When set to 1 then measurements and additional metadata collected during the measured boot process are sent to the DICE Protection Environment for storage and processing. A certificate chain, which represents the boot state of the device, can be queried from the DPE.
- DRTM_SUPPORT: Boolean flag to enable support for Dynamic Root of Trust for Measurement (DRTM). This feature has trust dependency on BL31 for taking the measurements and recording them as per PSA DRTM specification. For platforms which use BL2 to load/authenticate BL31 TRUSTED_BOARD_BOOT can be used and for the platforms which use RESET_TO_BL31 platform owners should have mechanism to authenticate BL31. This option defaults to 0.
- ENABLE_RMM: Boolean flag to enable the Realm-EL2 payload (RMM). This will take care of loading and initialising an image in Realm-EL2, and will at runtime dispatch calls from non-secure world to the RMM, if applicable. Default value is 0. Setting this flag implies that RMM must be enabled therefore, it mandates ENABLE_FEAT_RME to 1.
- ENABLE_RME: This options will be deprecated. Please use ENABLE_FEAT_RME. Until deprecated, setting this option to 1, will also set ENABLE_FEAT_RME and ENABLE_RMM to 1.
- RMM_V1_COMPAT: Boolean flag to enable support for RMM v1.x compatibility mode. When set to 0, TF-A will use the RMM-EL3 interface version required for RMMv2.0. Default value is 0.
- FIRME_SUPPORT: This option enables the FIRME service in TF-A.
- RMMD_ENABLE_EL3_TOKEN_SIGN: Numeric value to enable support for singing realm attestation token signing requests in EL3. This flag can take the values 0 and 1. The default value is 0. When set to 1, this option enables additional RMMD SMCs to push and pop requests for signing to EL3 along with platform hooks that must be implemented to service those requests and responses.
- ENABLE_SME_FOR_SWD: Boolean option to enable the Scalable Matrix Extension for secure world. Used along with SVE and FPU/SIMD. ENABLE_SME_FOR_NS and ENABLE_SVE_FOR_SWD must also be set to use this. Default is 0.
- ENABLE_SPMD_LP : This boolean option is used jointly with the SPM Dispatcher option (SPD=spmd). When enabled (1) it indicates support for logical partitions in EL3, managed by the SPMD as defined in the FF-A v1.2 specification. This flag is disabled by default. This flag must not be used if SPMC_AT_EL3 is enabled.
- PSA_CRYPTO: Boolean option for enabling MbedTLS PSA crypto APIs support. The platform will use PSA compliant Crypto APIs during authentication and image measurement process by enabling this option. It uses APIs defined as per the PSA Crypto API specification. This feature is only supported if using MbedTLS 3.x version. It is disabled (0) by default.
- LFA_SUPPORT: Boolean flag to enable support for Live Firmware activation as per the specification. This option defaults to 0.
- TRANSFER_LIST: Setting this to 1 enables support for Firmware Handoff using Transfer List defined in Firmware Handoff specification. This defaults to 0. Current implementation follows the Firmware

Handoff specification v0.9.

- USE_DEBUGFS: When set to 1 this option exposes a virtual filesystem interface through BL31 as a SiP SMC function. Default is disabled (0).
- HOB_LIST: Setting this to 1 enables support for passing boot information using HOB defined in Platform Initialization specification. This defaults to 0.
- ENABLE_ACS_SMC: When set to 1, this enables support for ACS SMC handler code to handle SMC calls from the Architecture Compliance Suite. The handler is intentionally empty to reserve the SMC section and allow project-specific implementations in future ACS use cases.

Firmware update options

- PSA_FWU_SUPPORT: Enable the firmware update mechanism as per the PSA FW update specification. The default value is 0. PSA firmware update implementation has few limitations, such as:
  - BL2 is not part of the protocol-updatable images. If BL2 needs to be updated, then it should be done through another platform-defined mechanism.
  - It assumes the platform's hardware supports CRC32 instructions.
- NR_OF_FW_BANKS: Define the number of firmware banks. This flag is used in defining the firmware update metadata structure. This flag is by default set to '2'.
- NR_OF_IMAGES_IN_FW_BANK: Define the number of firmware images in each firmware bank. Each firmware bank must have the same number of images as per the PSA FW update specification. This flag is used in defining the firmware update metadata structure. This flag is by default set to '1'.
- **PSA_FWU_METADATA_FW_STORE_DESC: To be enabled when the FWU**

  metadata contains image description. The default value is 1.

  The version 2 of the FWU metadata allows for an opaque metadata structure where a platform can choose to not include the firmware store description in the metadata structure. This option indicates if the firmware store description, which provides information on the updatable images is part of the structure.

SP Live Activation build options

- SUPPORT_SP_LIVE_ACTIVATION: Boolean option to enable live activation of Secure Partition(s) by using common SPMD LSP helpers. Enforces all of the following dependencies are met:
  - LFA_SUPPORT=1 to enable the live activation service in BL31.
  - ENABLE_SPMD_LP=1 allows SPMD logical secure partition to be enabled.
  - SPMD_SPM_AT_SEL2=1 as the current implementation only supports working with an S-EL2 SPMC (for example, Hafnium) to live activate an SP; It is incompatible with EL3 SPMC.

  This flag is experimental and currently exercised on FVP. Default value is 0.

Negative test scenario options

- TEST_IO_SHORT_READ_FI: Boolean flag to enable a negative-test for the BL image loader. When set to 1, TF-A intentionally simulates a short read for the image selected by TEST_IO_SHORT_READ_FI_IMAGE_ID by reporting fewer bytes read than the image size. This exercises the "read too short" error handling path in load_image() and is intended for test/CI only. Default is 0 (disabled). Must not be used on production devices.
- TEST_IO_SHORT_READ_FI_IMAGE_ID: Numeric flag that selects the image_id for which the short-read fault is injected when TEST_IO_SHORT_READ_FI=1. The value must match the image identifiers used by the platform image loading flow (for example BL31_IMAGE_ID, BL33_IMAGE_ID, etc., depending on the build and platform). Default is 0.

2.6 Internal Build Options

TF-A internally uses certain options that are not exposed directly through *build-options* but enabled or disabled indirectly and depends on certain options to be enabled or disabled.

- CTX_INCLUDE_EL2_REGS: This boolean option provides context save/restore operations when entering/exiting an EL2 execution context. This is of primary interest when Armv8.4-SecEL2 or RME extension is implemented. Default is 0 (disabled). This option will be set to 1 (enabled) when SPD=spmd and SPMD_SPM_AT_SEL2 is set or when ENABLE_RMM is set to 1 (enabled).
- FFH_SUPPORT: This boolean option provides support to enable Firmware First handling (FFH) of External aborts and SError interrupts originating from lower ELs which gets trapped in EL3. This option will be set to 1 (enabled) if HANDLE_EA_EL3_FIRST_NS is set. Currently only NS world routes EA to EL3 but in future when Secure/Realm wants to use FFH then they can introduce new macros which will enable this option implicitly.
- OPTEE_SP_FW_CONFIG: DTC build flag to include OP-TEE as SP in tb_fw_config device tree. This flag is defined only when ARM_SPMC_MANIFEST_DTS manifest file name contains pattern optee_sp.
- TRUSTY_SP_FW_CONFIG: DTC build flag to include Trusty as SP in tb_fw_config device tree. This flag is defined only when ARM_SPMC_MANIFEST_DTS manifest file name contains pattern trusty_sp.

2.7 Image Terminology

This page contains the current name, abbreviated name and purpose of the various images referred to in the Trusted Firmware project.

2.7.1 Common Image Features

- Some of the names and abbreviated names have changed to accommodate new requirements. The changed names are as backward compatible as possible to minimize confusion. Where applicable, the previous names are indicated. Some code, documentation and build artefacts may still refer to the previous names; these will inevitably take time to catch up.
- The main name change is to prefix each image with the processor it corresponds to (for example `AP_`, `SCP_`, …). In situations where there is no ambiguity (for example, within AP specific code/documentation), it is permitted to omit the processor prefix (for example, just BL1 instead of `AP_BL1`).
- Previously, the format for 3rd level images had 2 forms; `BL3` was either suffixed with a dash ("-") followed by a number (for example, `BL3-1`) or a subscript number, depending on whether rich text formatting was available. This was confusing and often the dash gets omitted in practice. Therefore the new form is to just omit the dash and not use subscript formatting.
- The names no longer contain dash ("-") characters at all. In some places (for example, function names) it's not possible to use this character. All dashes are either removed or replaced by underscores ("_").
- The abbreviation BL stands for BootLoader. This is a historical anomaly. Clearly, many of these images are not BootLoaders, they are simply firmware images. However, the BL abbreviation is now widely used and is retained for backwards compatibility.
- The image names are not case sensitive. For example, `bl1` is interchangeable with `BL1`, although mixed case should be avoided.

2.7.2 Trusted Firmware Images

Firmware Image Package: `FIP`

This is a packaging format used by TF-A to package firmware images in a single binary. The number and type of images that should be packed in a FIP is platform-specific and may include TF-A images and other firmware images required by the platform. For example, most platforms require a BL33 image which corresponds to the normal world bootloader (e.g. UEFI or U-Boot).

AP Boot ROM: `AP_BL1`

Typically, this is the first code to execute on the AP and cannot be modified. Its primary purpose is to perform the minimum initialization necessary to load and authenticate an updateable AP firmware image into an executable RAM location, then hand-off control to that image.

AP RAM Firmware: `AP_BL2`

This is the 2nd stage AP firmware. It is currently also known as the "Trusted Boot Firmware". Its primary purpose is to perform any additional initialization required to load and authenticate all 3rd level firmware images into their executable RAM locations, then hand-off control to the EL3 Runtime Firmware.

EL3 Runtime Firmware: `AP_BL31`

Also known as "SoC AP firmware" or "EL3 monitor firmware". Its primary purpose is to handle transitions between the normal and secure world.

Secure-EL1 Payload (SP): `AP_BL32`

Typically this is a TEE or Trusted OS, providing runtime secure services to the normal world. However, it may refer to a more abstract Secure-EL1 Payload (SP). Note that this abbreviation should only be used in systems where there is a single or primary image executing at Secure-EL1. In systems where there are potentially multiple SPs and there is no concept of a primary SP, this abbreviation should be avoided; use the recommended **Other AP 3rd level images** abbreviation instead.

AP Normal World Firmware: `AP_BL33`

For example, UEFI or uboot. Its primary purpose is to boot a normal world OS.

Other AP 3rd level images: `AP_BL3_XXX`

The abbreviated names of the existing 3rd level images imply a load/execution ordering (for example, `AP_BL31 -> AP_BL32 -> AP_BL33`). Some systems may have additional images and/or a different load/execution ordering. The abbreviated names of the existing images are retained for backward compatibility but new 3rd level images should be suffixed with an underscore followed by text identifier, not a number.

In systems where 3rd level images are provided by different vendors, the abbreviated name should identify the vendor as well as the image function. For example, `AP_BL3_ARM_RAS`.

Realm Monitor Management Firmware: `RMM`

This is the Realm-EL2 firmware. It is required if *Realm Management Extension (RME)* feature is enabled. If a path to RMM image is not provided, TF-A builds Test Realm Payload (TRP) image by default and uses it as the RMM image.

SCP Boot ROM: SCP_BL1 (previously BL0)

Typically, this is the first code to execute on the SCP and cannot be modified. Its primary purpose is to perform the minimum initialization necessary to load and authenticate an updateable SCP firmware image into an executable RAM location, then hand-off control to that image. This may be performed in conjunction with other processor firmware (for example, AP_BL1 and AP_BL2).

This image was previously abbreviated as BL0 but in some systems, the SCP may directly load/authenticate its own firmware. In these systems, it doesn't make sense to interleave the image terminology for AP and SCP; both AP and SCP Boot ROMs are BL1 from their own point of view.

SCP RAM Firmware: SCP_BL2 (previously BL3-0)

This is the 2nd stage SCP firmware. It is currently also known as the "SCP runtime firmware" but it could potentially be an intermediate firmware if the SCP needs to load/authenticate multiple 3rd level images in future.

This image was previously abbreviated as BL3-0 but from the SCP's point of view, this has always been the 2nd stage firmware. The previous name is too AP-centric.

2.7.3 Firmware Update (FWU) Images

The terminology for these images has not been widely adopted yet but they have to be considered in a production Trusted Board Boot solution.

AP Firmware Update Boot ROM: AP_NS_BL1U

Typically, this is the first normal world code to execute on the AP during a firmware update operation, and cannot be modified. Its primary purpose is to load subsequent firmware update images from an external interface and communicate with AP_BL1 to authenticate those images.

During firmware update, there are (potentially) multiple transitions between the secure and normal world. The "level" of the BL image is relative to the world it's in so it makes sense to encode "NS" in the normal world images. The absence of "NS" implies a secure world image.

AP Firmware Update Config: AP_BL2U

This image does the minimum necessary AP secure world configuration required to complete the firmware update operation. It is potentially a subset of AP_BL2 functionality.

SCP Firmware Update Config: `SCP_BL2U` (previously `BL2-U0`)

This image does the minimum necessary SCP secure world configuration required to complete the firmware update operation. It is potentially a subset of `SCP_BL2` functionality.

AP Firmware Updater: `AP_NS_BL2U` (previously `BL3-U`)

This is the 2nd stage AP normal world firmware updater. Its primary purpose is to load a new set of firmware images from an external interface and write them into non-volatile storage.

2.7.4 Other Processor Firmware Images

Some systems may have additional processors to the AP and SCP. For example, a Management Control Processor (MCP). Images for these processors should follow the same terminology, with the processor abbreviation prefix, followed by underscore and the level of the firmware image.

For example,

MCP Boot ROM: `MCP_BL1`

MCP RAM Firmware: `MCP_BL2`

2.8 EL3 Runtime Service Writer's Guide

2.8.1 Introduction

This document describes how to add a runtime service to the EL3 Runtime Firmware component of Trusted Firmware-A (TF-A), BL31.

Software executing in the normal world and in the trusted world at exception levels lower than EL3 will request runtime services using the Secure Monitor Call (SMC) instruction. These requests will follow the convention described in the SMC Calling Convention PDD (SMCCC). The SMCCC assigns function identifiers to each SMC request and describes how arguments are passed and results are returned.

SMC Functions are grouped together based on the implementor of the service, for example a subset of the Function IDs are designated as "OEM Calls" (see SMCCC for full details). The EL3 runtime services framework in BL31 enables the independent implementation of services for each group, which are then compiled into the BL31 image. This simplifies the integration of common software from Arm to support PSCI, Secure Monitor for a Trusted OS and SoC specific software. The common runtime services framework ensures that SMC Functions are dispatched to their respective service implementation - the *Firmware Design* document provides details of how this is achieved.

The interface and operation of the runtime services depends heavily on the concepts and definitions described in the SMCCC, in particular SMC Function IDs, Owning Entity Numbers (OEN), Fast and Standard calls, and the SMC32 and SMC64 calling conventions. Please refer to that document for a full explanation of these terms.

2.8.2 Owning Entities, Call Types and Function IDs

The SMC Function Identifier includes a OEN field. These values and their meaning are described in SMCCC and summarized in table 1 below. Some entities are allocated a range of of OENs. The OEN must be interpreted in conjunction with the SMC call type, which is either *Fast* or *Yielding*. Fast calls are uninterruptible whereas Yielding calls can be pre-empted. The majority of Owning Entities only have allocated ranges for Fast calls: Yielding calls are reserved exclusively for Trusted OS providers or for interoperability with legacy 32-bit software that predates the SMCCC.

```
Type       OEN     Service
Fast        0      Arm Architecture calls
Fast        1      CPU Service calls
Fast        2      SiP Service calls
Fast        3      OEM Service calls
Fast        4      Standard Secure Service calls
Fast        5      Standard Hypervisor Service Calls
Fast        6      Vendor Specific Hypervisor Service Calls
Fast        7      Vendor Specific EL3 Monitor Calls
Fast       8-47    Reserved for future use
Fast      48-49    Trusted Application calls
Fast      50-63    Trusted OS calls

Yielding   0- 1    Reserved for existing Armv7-A calls
Yielding   2-63    Trusted OS Standard Calls
```

Table 1: Service types and their corresponding Owning Entity Numbers

Each individual entity can allocate the valid identifiers within the entity range as they need - it is not necessary to coordinate with other entities of the same type. For example, two SoC providers can use the same Function ID within the SiP Service calls OEN range to mean different things - as these calls should be specific to the SoC. The Standard Runtime Calls OEN is used for services defined by Arm standards, such as PSCI.

The SMC Function ID also indicates whether the call has followed the SMC32 calling convention, where all parameters are 32-bit, or the SMC64 calling convention, where the parameters are 64-bit. The framework identifies and rejects invalid calls that use the SMC64 calling convention but that originate from an AArch32 caller.

The EL3 runtime services framework uses the call type and OEN to identify a specific handler for each SMC call, but it is expected that an individual handler will be responsible for all SMC Functions within a given service type.

2.8.3 Getting started

TF-A has a `services` directory in the source tree under which each owning entity can place the implementation of its runtime service. The PSCI implementation is located here in the `lib/psci` directory.

Runtime service sources will need to include the `runtime_svc.h` header file.

2.8.4 Registering a runtime service

A runtime service is registered using the DECLARE_RT_SVC() macro, specifying the name of the service, the range of OENs covered, the type of service and initialization and call handler functions.

```
#define DECLARE_RT_SVC(_name, _start, _end, _type, _setup, _smch)
```

- _name is used to identify the data structure declared by this macro, and is also used for diagnostic purposes
- _start and _end values must be based on the OEN_* values defined in smccc.h
- _type must be one of SMC_TYPE_FAST or SMC_TYPE_YIELD
- _setup is the initialization function with the rt_svc_init signature:

  ```
  typedef int32_t (*rt_svc_init)(void);
  ```

- _smch is the SMC handler function with the rt_svc_handle signature:

  ```
  typedef uintptr_t (*rt_svc_handle_t)(uint32_t smc_fid,
                                        u_register_t x1, u_register_t x2,
                                        u_register_t x3, u_register_t x4,
                                        void *cookie,
                                        void *handle,
                                        u_register_t flags);
  ```

Details of the requirements and behavior of the two callbacks is provided in the following sections.

During initialization the services framework validates each declared service to ensure that the following conditions are met:

1. The _start OEN is not greater than the _end OEN
2. The _end OEN does not exceed the maximum OEN value (63)
3. The _type is one of SMC_TYPE_FAST or SMC_TYPE_YIELD
4. _setup and _smch routines have been specified

std_svc_setup.c provides an example of registering a runtime service:

```
/* Register Standard Service Calls as runtime service */
DECLARE_RT_SVC(
        std_svc,
        OEN_STD_START,
        OEN_STD_END,
        SMC_TYPE_FAST,
        std_svc_setup,
        std_svc_smc_handler
);
```

2.8.5 Initializing a runtime service

Runtime services are initialized once, during cold boot, by the primary CPU after platform and architectural initialization is complete. The framework performs basic validation of the declared service before calling the service initialization function (`_setup` in the declaration). This function must carry out any essential EL3 initialization prior to receiving a SMC Function call via the handler function.

On success, the initialization function must return `0`. Any other return value will cause the framework to issue a diagnostic:

```
Error initializing runtime service <name of the service>
```

and then ignore the service - the system will continue to boot but SMC calls will not be passed to the service handler and instead return the *Unknown SMC Function ID* result `0xFFFFFFFF`.

If the system must not be allowed to proceed without the service, the initialization function must itself cause the firmware boot to be halted.

If the service uses per-CPU data this must either be initialized for all CPUs during this call, or be done lazily when a CPU first issues an SMC call to that service.

2.8.6 Handling runtime service requests

SMC calls for a service are forwarded by the framework to the service's SMC handler function (`_smch` in the service declaration). This function must have the following signature:

```
typedef uintptr_t (*rt_svc_handle_t)(uint32_t smc_fid,
                                     u_register_t x1, u_register_t x2,
                                     u_register_t x3, u_register_t x4,
                                     void *cookie,
                                     void *handle,
                                     u_register_t flags);
```

The handler is responsible for:

1. Determining that `smc_fid` is a valid and supported SMC Function ID, otherwise completing the request with the *Unknown SMC Function ID*:

   ```
   SMC_RET1(handle, SMC_UNK);
   ```

2. Determining if the requested function is valid for the calling security state. SMC Calls can be made from Non-secure, Secure or Realm worlds and the framework will forward all calls to the service handler.

   The `flags` parameter to this function indicates the caller security state in bits 0 and 5. The `is_caller_secure(flags)`, `is_caller_non_secure(flags)` and `is_caller_realm(flags)` helper functions can be used to determine whether the caller's security state is Secure, Non-secure or Realm respectively.

   If invalid, the request should be completed with:

   ```
   SMC_RET1(handle, SMC_UNK);
   ```

3. Truncating parameters for calls made using the SMC32 calling convention. Such calls can be determined by checking the CC field in bit[30] of the `smc_fid` parameter, for example by using:

   ```
   if (GET_SMC_CC(smc_fid) == SMC_32) ...
   ```

   For such calls, the upper bits of the parameters x1-x4 and the saved parameters X5-X7 are UNDEFINED and must be explicitly ignored by the handler. This can be done by truncating the values to a suitable 32-bit integer type before use, for example by ensuring that functions defined to handle individual SMC Functions use appropriate 32-bit parameters.

4. Providing the service requested by the SMC Function, utilizing the immediate parameters x1-x4 and/or the additional saved parameters X5-X7. The latter can be retrieved using the `SMC_GET_GP(handle, ref)` function, supplying the appropriate `CTX_GPREG_Xn` reference, e.g.

   ```
   uint64_t x6 = SMC_GET_GP(handle, CTX_GPREG_X6);
   ```

5. Implementing the standard SMC32 Functions that provide information about the implementation of the service. These are the Call Count, Implementor UID and Revision Details for each service documented in section 6 of the SMCCC.

   TF-A expects owning entities to follow this recommendation.

6. Returning the result to the caller. Based on SMCCC spec, results are returned in W0-W7(X0-X7) registers for SMC32(SMC64) calls from AArch64 state. Results are returned in R0-R7 registers for SMC32 calls from AArch32 state. The framework provides a family of macros to set the multi-register return value and complete the handler:

   ```
   AArch64 state:

   SMC_RET1(handle, x0);
   SMC_RET2(handle, x0, x1);
   SMC_RET3(handle, x0, x1, x2);
   SMC_RET4(handle, x0, x1, x2, x3);
   SMC_RET5(handle, x0, x1, x2, x3, x4);
   SMC_RET6(handle, x0, x1, x2, x3, x4, x5);
   SMC_RET7(handle, x0, x1, x2, x3, x4, x5, x6);
   SMC_RET8(handle, x0, x1, x2, x3, x4, x5, x6, x7);

   AArch32 state:

   SMC_RET1(handle, r0);
   SMC_RET2(handle, r0, r1);
   SMC_RET3(handle, r0, r1, r2);
   SMC_RET4(handle, r0, r1, r2, r3);
   SMC_RET5(handle, r0, r1, r2, r3, r4);
   SMC_RET6(handle, r0, r1, r2, r3, r4, r5);
   SMC_RET7(handle, r0, r1, r2, r3, r4, r5, r6);
   SMC_RET8(handle, r0, r1, r2, r3, r4, r5, r6, r7);
   ```

The `cookie` parameter to the handler is reserved for future use and can be ignored. The `handle` is returned by the SMC handler - completion of the handler function must always be via one of the `SMC_RETn()` macros.

2.8.7 Security Considerations: SMC Argument Validation

When implementing a platform-specific SiP handler, the following architectural patterns MUST be followed to prevent Privilege-Blind Forwarding (PBF) and Double-Fetch (TOCTOU) vulnerabilities:

1. **Atomic Shadow-Copy** When an SMC passes a pointer or memory range referencing Non-Secure memory, the contents of that memory are volatile and can be modified by another CPU core after EL3 has validated the address.

   All arguments MUST be copied into EL3-local variables precisely once before any validation or use occurs. Subsequent logic should only reference these local copies.

2. **Memory Range Tuple Validation** When an SMC passes a memory range (base + size), validating them independently is insufficient as it fails to account for 64-bit integer overflow/wrap-around.

   The handler MUST verify that `(base + size)` does not wrap around and that the entire range resides within the expected security boundary (e.g. Non-Secure DRAM).

3. **Dynamic Platform Validation** Validation should not rely on static memory maps. Handlers should use platform-specific hooks (e.g. `plat_is_valid_ns_address_range()`) that query the Granule Protection Tables (GPT) or other dynamic memory controllers to ensure the requested range is currently owned by the caller's world.

Standard Validation Framework

TF-A provides a standardized validation framework in `include/common/smc_validation_framework.h` to simplify these checks and reduce boilerplate error-prone code. It is recommended to use the unmarshaling macros provided by this framework:

```
#include <common/smc_validation_framework.h>

uintptr_t sip_handler(uint32_t smc_fid, u_register_t x1, ..., void *handle)
    /* 1. Atomic Shadow-Copy & Validation */
    SMC_ARG_MEM_RANGE(buf, len, handle, 2, 3);

    /* 2. Parameters are now safe to use */
    return plat_perform_action(cmd, buf, len);
}
```

Note: The PSCI and Test Secure-EL1 Payload Dispatcher services do not follow all of the above requirements yet.

2.8.8 Security Considerations: SMC Function ID Validation

Every handler receives calls for all function IDs within its registered OEN range. Before acting on a call, the handler must validate the internal fields of `smc_fid` and enforce an explicit binding between the function ID and the permitted caller security world.

1. **Extract fields via macros, never compare raw FIDs**

   The `smc_fid` parameter contains defined bit fields and reserved bits that may carry undefined values. Use the `GET_SMC_*` macros from `lib/smccc.h` to decompose the identifier before any comparison:

   ```
   uint32_t oen  = GET_SMC_OEN(smc_fid);   /* bits[29:24] */
   uint32_t type = GET_SMC_TYPE(smc_fid);  /* bit[31]     */
   uint32_t cc   = GET_SMC_CC(smc_fid);    /* bit[30]     */
   uint32_t num  = GET_SMC_NUM(smc_fid);   /* bits[15:0]  */
   ```

   Comparing a raw `smc_fid` value directly against a constant risks false matches if reserved bits are set by the caller.

2. **Bind each function ID to its permitted caller world**

   Each supported function number should have an explicit allowlist of security worlds. Use the `flags` parameter with `is_caller_non_secure()`, `is_caller_secure()`, and `is_caller_realm()` from `lib/smccc.h` to enforce this at the point of dispatch:

   ```
   switch (GET_SMC_NUM(smc_fid)) {
   case PLAT_SIP_FUNC_A:          /* NS-only */
       if (!is_caller_non_secure(flags)) {
           SMC_RET1(handle, SMC_UNK);
       }
       return plat_sip_func_a(x1, x2);

   case PLAT_SIP_FUNC_B:          /* Secure-only */
       if (!is_caller_secure(flags)) {
           SMC_RET1(handle, SMC_UNK);
       }
       return plat_sip_func_b(x1);

   default:
       SMC_RET1(handle, SMC_UNK);
   }
   ```

   Absence of an explicit caller-world check is a **Privilege-Blind Forwarding (PBF)** vulnerability: a caller from the wrong security world can invoke a function it was never intended to reach.

3. **Re-check caller world at every forwarding point**

   When a handler forwards a call to a sub-dispatcher, the sub-dispatcher must perform its own caller-world check independently. A check at the outer handler does **not** protect the inner one. If the forwarding path does not re-validate security state, a caller can reach a privileged sub-operation in the wrong world.

   Each handler in the call chain is responsible for its own security-state validation.

2.8.9 Services that contain multiple sub-services

It is possible that a single owning entity implements multiple sub-services. For example, the Standard calls service handles `0x84000000-0x8400FFFF` and `0xC4000000-0xC400FFFF` functions. Within that range, the PSCI service handles the `0x84000000-0x8400001F` and `0xC4000000-0xC400001F` functions. In that respect, PSCI is a 'sub-service' of the Standard calls service. In future, there could be additional such sub-services in the Standard calls service which perform independent functions.

In this situation it may be valuable to introduce a second level framework to enable independent implementation of sub-services. Such a framework might look very similar to the current runtime services framework, but using a different part of the SMC Function ID to identify the sub-service. TF-A does not provide such a framework at present.

2.8.10 Secure-EL1 Payload Dispatcher service (SPD)

Services that handle SMC Functions targeting a Trusted OS, Trusted Application, or other Secure-EL1 Payload are special. These services need to manage the Secure-EL1 context, provide the *Secure Monitor* functionality of switching between the normal and secure worlds, deliver SMC Calls through to Secure-EL1 and generally manage the Secure-EL1 Payload through CPU power-state transitions.

TODO: Provide details of the additional work required to implement a SPD and the BL31 support for these services. Or a reference to the document that will provide this information….

2.8.11 Additional References:

1. *ARM SiP Services*
2. *Vendor Specific EL3 Monitor Service Calls*

CHAPTER

THREE

PROCESSES & POLICIES

3.1 Security Handling

3.1.1 Security Disclosures

We disclose all security vulnerabilities we find, or are advised about, that are relevant to Trusted Firmware-A. We encourage responsible disclosure of vulnerabilities and inform users as best we can about all possible issues.

We disclose TF-A vulnerabilities as Security Advisories, all of which are listed at the bottom of this page. Any new ones will, additionally, be announced on the TF-A project's mailing list.

3.1.2 Found a Security Issue?

Although we try to keep TF-A secure, we can only do so with the help of the community of developers and security researchers.

> **Warning:** If you think you have found a security vulnerability, please **do not** report it in the issue tracker or on the mailing list. Instead, please follow the TrustedFirmware.org security incident process.

One of the goals of this process is to ensure providers of products that use TF-A have a chance to consider the implications of the vulnerability and its remedy before it is made public. As such, please follow the disclosure plan outlined in the process. We do our best to respond and fix any issues quickly.

Afterwards, we encourage you to write-up your findings about the TF-A source code.

3.1.3 Attribution

We will name and thank you in the *Change Log & Release Notes* distributed with the source code and in any published security advisory.

3.1.4 Security Advisories

ID	Title
Advisory TFV-1 (CVE-2016-10319)	Malformed Firmware Update SMC can result in copy of unexpectedly large data into secure memory
Advisory TFV-2 (CVE-2017-7564)	Enabled secure self-hosted invasive debug interface can allow normal world to panic secure world
Advisory TFV-3 (CVE-2017-7563)	RO memory is always executable at AArch64 Secure EL1
Advisory TFV-4 (CVE-2017-9607)	Malformed Firmware Update SMC can result in copy or authentication of unexpected data in secure memory in AArch32 state
Advisory TFV-5 (CVE-2017-15031)	Not initializing or saving/restoring PMCR_EL0 can leak secure world timing information
Advisory TFV-6 (CVE-2017-5753, CVE-2017-5715, CVE-2017-5754)	Trusted Firmware-A exposure to speculative processor vulnerabilities using cache timing side-channels
Advisory TFV-7 (CVE-2018-3639)	Trusted Firmware-A exposure to cache speculation vulnerability Variant 4
Advisory TFV-8 (CVE-2018-19440)	Not saving x0 to x3 registers can leak information from one Normal World SMC client to another
Advisory TFV-9 (CVE-2022-23960)	Trusted Firmware-A exposure to speculative processor vulnerabilities with branch prediction target reuse
Advisory TFV-10 (CVE-2022-47630)	Incorrect validation of X.509 certificate extensions can result in an out-of-bounds read
Advisory TFV-11 (CVE-2023-49100)	A Malformed SDEI SMC can cause out of bound memory read

3.2 Platform Ports Policy

This document clarifies a couple of policy points around platform ports management.

3.2.1 Platform compatibility policy

Platform compatibility is mainly affected by changes to Platform APIs (as documented in the *Porting Guide*), driver APIs (like the GICv3 drivers) or library interfaces (like xlat_table library). The project will try to maintain compatibility for upstream platforms.

Due to evolving requirements and enhancements, there might be changes affecting platform compatibility, which means the previous interface needs to be deprecated and a new interface introduced to replace it. In case the migration to the new interface is trivial, the contributor of the change is expected to make good effort to migrate the upstream platforms to the new interface.

The project will generally not take into account downstream platforms. If those are affected by a deprecation / removal decision, we encourage their maintainers to upstream their platform code or copy the latest version of the code being deprecated into their downstream tree.

The deprecated interfaces are listed inside *Release Processes* as well as the release after which each one will be removed. When an interface is deprecated, the page must be updated to indicate the release after which the interface will be removed. This must be at least 1 full release cycle in future. For non-trivial interface changes, an email should be sent out to the TF-A public mailing list to notify platforms that they should migrate away from the deprecated interfaces. Platforms are expected to migrate before the removal of the deprecated interface.

3.2.2 Deprecation policy

If a platform, driver or library interface is no longer maintained, it is best to deprecate it to keep the projects' source tree clean and healthy. Deprecation can be a 1-stage or 2-stage process (up to the maintainers).

- *2-stage*: The source code can be kept in the repository for a cooling off period before deleting it (typically 2 release cycles). In this case, we keep track of the *Deprecated* version separately from the *Deleted* version.
- *1-stage*: The source code can be deleted straight away. In this case, both versions are the same.

The *Platform Ports* page provides a list of all deprecated/deleted platform ports (or soon to be) to this day.

3.3 Commit Style

When writing commit messages, please think carefully about the purpose and scope of the change you are making: describe briefly what the change does, and describe in detail why it does it. This helps to ensure that changes to the code-base are transparent and approachable to reviewers, and it allows us to keep a more accurate changelog. You may use Markdown in commit messages.

A good commit message provides all the background information needed for reviewers to understand the intent and rationale of the patch. This information is also useful for future reference.

For example:

- What does the patch do?
- What motivated it?
- What impact does it have?
- How was it tested?
- Have alternatives been considered? Why did you choose this approach over another one?
- If it fixes an issue, include a reference.

TF-A follows the Conventional Commits specification. All commits to the main repository are expected to adhere to these guidelines, so it is **strongly** recommended that you read at least the quick summary of the specification.

To briefly summarize, commit messages are expected to be of the form:

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

Note that the type, the scope and the first letter of the description (also called subject by the commitlint checker) must be lower case.

The following example commit message demonstrates the use of the `refactor` type and the `amu` scope:

```
refactor(amu): factor out register accesses

This change introduces a small set of register getters and setters to
avoid having to repeatedly mask and shift in complex code.

Change-Id: Ia372f60c5efb924cd6eeceb75112e635ad13d942
Signed-off-by: Chris Kay <chris.kay@arm.com>
```

The following *types* are permissible and are strictly enforced:

Scope	Description
`feat`	A new feature
`fix`	A bug fix
`build`	Changes that affect the build system or external dependencies
`ci`	Changes to our CI configuration files and scripts
`docs`	Documentation-only changes
`perf`	A code change that improves performance
`refactor`	A code change that neither fixes a bug nor adds a feature
`revert`	Changes that revert a previous change
`style`	Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc.)
`test`	Adding missing tests or correcting existing tests
`chore`	Any other change

The permissible *scopes* are more flexible, and we maintain a list of them in our `changelog configuration file`. Scopes in this file are organized by their changelog section, where each changelog section has a single scope that is considered to be blessed, and possibly several deprecated scopes. Please avoid using deprecated scopes.

While we don't enforce scopes strictly, we do ask that commits use these if they can, or add their own if no appropriate one exists (see *Adding Scopes*).

It's highly recommended that you use the tooling installed by the optional steps in the *prerequisites* guide to validate commit messages locally, as commitlint reports a live list of the acceptable scopes.

3.3.1 Adding Scopes

Scopes that are not present in the changelog configuration file are considered to be deprecated, and should be avoided. If you are adding a new component that does not yet have a designated scope, please add one.

For example, if you are adding or making modifications to *Foo*'s latest and greatest new platform *Bar* then you would add it to the *Platforms* changelog sub-section, and the hierarchy should look something like this:

```yaml
- title: Platforms

  subsections:
    - title: Foo
      scope: foo

      subsections:
        - title: Bar
          scope: bar
```

When creating new scopes, try to keep them short and succinct, and use kebab case (`this-is-kebab-case`). Components with a product name (i.e. most platforms and some drivers) should use that name (e.g. `gic600ae`, `flexspi`, `stpmic1`), otherwise use a name that uniquely represents the component (e.g. `marvell-comphy-3700`, `rcar3-drivers`, `a3720-uart`).

3.3.2 Mandated Trailers

Commits are expected to be signed off with the `Signed-off-by:` trailer using your real name and email address. You can do this automatically by committing with Git's `-s` flag. By adding this line the contributor certifies the contribution is made under the terms of the `Developer Certificate of Origin`.

There may be multiple `Signed-off-by:` lines depending on the history of the patch, but one **must** be the committer. More details may be found in the Gerrit Signed-off-by Lines guidelines.

Ensure that each commit also has a unique `Change-Id:` line. If you have followed optional steps in the prerequisites to either install the Node.js tools or clone the repository using the "*Clone with commit-msg hook*" clone method, then this should be done automatically for you.

More details may be found in the Gerrit Change-Ids documentation.

3.3.3 Optional Automation Trailers

Projects and workflows built around *TF-A* may use additional machine-readable trailers to simplify triage and backporting. For LTS triage, developers may add `LTS-candidate: yes` in the footer block at the end of the commit message to indicate to the LTS maintainers that the patch should be evaluated as a candidate for a future LTS release. LTS maintainers retain the final say in triage and merge decisions for the LTS branches.

These trailers are optional and should be added in the footer block of the commit message. They do not change the standard TF-A commit subject format, which should remain a normal Conventional Commits subject such as `fix(psci): ....`

3.4 Coding Style

The following sections outline the *TF-A* coding style for *C* code. The style is based on the Linux kernel coding style, with a few modifications.

The style should not be considered *set in stone*. Feel free to provide feedback and suggestions.

Note: You will almost certainly find code in the *TF-A* repository that does not follow the style. The intent is for all code to do so eventually.

3.4.1 File Encoding

The source code must use the **UTF-8** character encoding. Comments and documentation may use non-ASCII characters when required (e.g. Greek letters used for units) but code itself is still limited to ASCII characters.

Newlines must be in **Unix** style, which means that only the Line Feed (`LF`) character is used to break a line and reset to the first column.

3.4.2 Language

The primary language for comments and naming must be International English. In cases where there is a conflict between the American English and British English spellings of a word, the American English spelling is used.

Exceptions are made when referring directly to something that does not use international style, such as the name of a company. In these cases the existing name should be used as-is.

3.4.3 C Language Standard

The C language mode used for TF-A is *GNU99*. This is the "GNU dialect of ISO C99", which implies the *ISO C99* standard with GNU extensions.

Both GCC and Clang compiler toolchains have support for *GNU99* mode, though Clang does lack support for a small number of GNU extensions. These missing extensions are rarely used, however, and should not pose a problem.

3.4.4 MISRA Compliance

TF-A attempts to comply with the MISRA C:2012 Guidelines. *ECLAIR* static analysis is used to regularly generate a report of current MISRA defects and to prevent the addition of new ones.

It is not possible for the project to follow all MISRA guidelines. Table 1 below lists all rules and directives and whether we aim to comply with them or not. A rationale is given for each deviation.

Note: Enforcing a rule does not mean that the codebase is free of defects of that rule, only that they would ideally be removed.

Note: Third-party libraries are not considered in our MISRA analysis and we do not intend to modify them to make them MISRA compliant.

Table

Seq	Dir / Rule	Number	Source	Category	Checker Enabled	Enforced	Comments
1	D	1.1	MISRA C 2012	Required	N/A	Yes	
2	D	2.1	MISRA C 2012	Required	N/A	Yes	
3	D	3.1	MISRA C 2012	Required	N/A	No	It can't be done retro
4	D	4.1	MISRA C 2012	Required	N/A	Yes	
5	D	4.2	MISRA C 2012	Advisory	N/A	Yes	
6	D	4.3	MISRA C 2012	Required	Yes	Yes	
7	D	4.4	MISRA C 2012	Advisory	Yes	Yes	
8	D	4.5	MISRA C 2012	Advisory	Yes	Yes	
9	D	4.6	MISRA C 2012	Advisory	No	No	We use a mix of bot
10	D	4.7	MISRA C 2012	Required	Yes	Yes	
11	D	4.8	MISRA C 2012	Advisory	No	No	Fixing all instances v
12	D	4.9	MISRA C 2012	Advisory	No	No	We mustn't introduc
13	D	4.10	MISRA C 2012	Required	Yes	Yes	
14	D	4.11	MISRA C 2012	Required	Yes	Yes	
15	D	4.12	MISRA C 2012	Required	Yes	Yes	
16	D	4.13	MISRA C 2012	Advisory	Yes	Yes	
17	D	4.14	MISRA C 2012 AMD-1	Required	Yes	Yes	
18	R	1.1	MISRA C 2012	Required	Yes	Yes	
19	R	1.2	MISRA C 2012	Advisory	Yes	Optional	It bans __attribute__
20	R	1.3	MISRA C 2012	Required	N/A	Yes	
21	R	2.1	MISRA C 2012	Required	Yes	Yes	
22	R	2.2	MISRA C 2012	Required	Yes	Yes	
23	R	2.3	MISRA C 2012	Advisory	Yes	Optional	It prevents the usage
24	R	2.4	MISRA C 2012	Advisory	No	No	Header files may use
25	R	2.5	MISRA C 2012	Advisory	No	No	We define many hea
26	R	2.6	MISRA C 2012	Advisory	Yes	Yes	
27	R	2.7	MISRA C 2012	Advisory	No	No	Doesn't allow for sin
28	R	3.1	MISRA C 2012	Required	Yes	Yes	
29	R	3.2	MISRA C 2012	Required	Yes	Yes	
30	R	4.1	MISRA C 2012	Required	Yes	Yes	
31	R	4.2	MISRA C 2012	Advisory	Yes	Yes	
32	R	5.1	MISRA C 2012	Required	No	No	We use weak symbo
33	R	5.2	MISRA C 2012	Required	Yes	Yes	
34	R	5.3	MISRA C 2012	Required	Yes	Yes	

35	R	5.4	MISRA C 2012	Required	Yes	Yes	
36	R	5.5	MISRA C 2012	Required	Yes	Yes	
37	R	5.6	MISRA C 2012	Required	Yes	Yes	
38	R	5.7	MISRA C 2012	Required	Yes	Optional	Fixing all existing de
39	R	5.8	MISRA C 2012	Required	No	No	We use weak symbo
40	R	5.9	MISRA C 2012	Advisory	Yes	Yes	
41	R	6.1	MISRA C 2012	Required	Yes	Yes	
42	R	6.2	MISRA C 2012	Required	Yes	Yes	
43	R	7.1	MISRA C 2012	Required	Yes	Yes	
44	R	7.2	MISRA C 2012	Required	Yes	Yes	
45	R	7.3	MISRA C 2012	Required	Yes	Yes	
46	R	7.4	MISRA C 2012	Required	Yes	Yes	
47	R	8.1	MISRA C 2012	Required	Yes	Yes	
48	R	8.2	MISRA C 2012	Required	Yes	Yes	
49	R	8.3	MISRA C 2012	Required	Yes	Yes	
50	R	8.4	MISRA C 2012	Required	Yes	Yes	
51	R	8.5	MISRA C 2012	Required	Yes	Yes	
52	R	8.6	MISRA C 2012	Required	No	No	We use weak symbo
53	R	8.7	MISRA C 2012	Advisory	No	No	Bans pattern of decl
54	R	8.8	MISRA C 2012	Required	Yes	Yes	
55	R	8.9	MISRA C 2012	Advisory	Yes	Yes	
56	R	8.10	MISRA C 2012	Required	Yes	Yes	
57	R	8.11	MISRA C 2012	Advisory	Yes	Optional	This may not be pos
58	R	8.12	MISRA C 2012	Required	Yes	Yes	
59	R	8.13	MISRA C 2012	Advisory	Yes	Optional	The benefits of fixin
60	R	8.14	MISRA C 2012	Required	Yes	Yes	
61	R	9.1	MISRA C 2012	Mandatory	Yes	Yes	
62	R	9.2	MISRA C 2012	Required	Yes	Yes	
63	R	9.3	MISRA C 2012	Required	Yes	Yes	
64	R	9.4	MISRA C 2012	Required	Yes	Yes	
65	R	9.5	MISRA C 2012	Required	Yes	Yes	
66	R	10.1	MISRA C 2012	Required	Yes	Optional	Fixing existing code
67	R	10.2	MISRA C 2012	Required	Yes	Yes	
68	R	10.3	MISRA C 2012	Required	Yes	Optional	Fixing existing code
69	R	10.4	MISRA C 2012	Required	Yes	Optional	Fixing existing code
70	R	10.5	MISRA C 2012	Advisory	Yes	Yes	
71	R	10.6	MISRA C 2012	Required	Yes	Yes	
72	R	10.7	MISRA C 2012	Required	Yes	Yes	
73	R	10.8	MISRA C 2012	Required	Yes	Yes	
74	R	11.1	MISRA C 2012	Required	Yes	Yes	
75	R	11.2	MISRA C 2012	Required	Yes	Yes	
76	R	11.3	MISRA C 2012	Required	Yes	Yes	
77	R	11.4	MISRA C 2012	Advisory	No	No	This would be invasi
78	R	11.5	MISRA C 2012	Advisory	No	No	This seems to preclu

79	R	11.6	MISRA C 2012	Required	Yes	Optional	This is needed in sev
80	R	11.7	MISRA C 2012	Required	Yes	Yes	
81	R	11.8	MISRA C 2012	Required	Yes	Yes	
82	R	11.9	MISRA C 2012	Required	Yes	Yes	
83	R	12.1	MISRA C 2012	Advisory	Yes	Yes	
84	R	12.2	MISRA C 2012	Required	Yes	Yes	This rule is fine, but
85	R	12.3	MISRA C 2012	Advisory	Yes	Yes	
86	R	12.4	MISRA C 2012	Advisory	Yes	Yes	
87	R	12.5	MISRA C 2012 AMD-1	Mandatory	Yes	Yes	
88	R	13.1	MISRA C 2012	Required	Yes	Yes	
89	R	13.2	MISRA C 2012	Required	Yes	Yes	
90	R	13.3	MISRA C 2012	Advisory	Yes	Yes	
91	R	13.4	MISRA C 2012	Advisory	Yes	Yes	
92	R	13.5	MISRA C 2012	Required	Yes	Yes	
93	R	13.6	MISRA C 2012	Mandatory	Yes	Yes	
94	R	14.1	MISRA C 2012	Required	Yes	Yes	
95	R	14.2	MISRA C 2012	Required	Yes	Yes	
96	R	14.3	MISRA C 2012	Required	Yes	Yes	
97	R	14.4	MISRA C 2012	Required	Yes	Yes	
98	R	15.1	MISRA C 2012	Advisory	No	No	In some cases goto r
99	R	15.2	MISRA C 2012	Required	Yes	Yes	
100	R	15.3	MISRA C 2012	Required	Yes	Yes	
101	R	15.4	MISRA C 2012	Advisory	Yes	Yes	
102	R	15.5	MISRA C 2012	Advisory	No	No	This has no real valu
103	R	15.6	MISRA C 2012	Required	No	No	This directly contrac
104	R	15.7	MISRA C 2012	Required	Yes	Yes	
105	R	16.1	MISRA C 2012	Required	No	No	Cannot comply with
106	R	16.2	MISRA C 2012	Required	Yes	Yes	
107	R	16.3	MISRA C 2012	Required	No	No	Returns within switc
108	R	16.4	MISRA C 2012	Required	Yes	Yes	
109	R	16.5	MISRA C 2012	Required	Yes	Yes	
110	R	16.6	MISRA C 2012	Required	Yes	Yes	
111	R	16.7	MISRA C 2012	Required	Yes	Yes	
112	R	17.1	MISRA C 2012	Required	No	No	This is needed for pr
113	R	17.2	MISRA C 2012	Required	Yes	Yes	Bans recursion. We
114	R	17.3	MISRA C 2012	Mandatory	Yes	Yes	
115	R	17.4	MISRA C 2012	Mandatory	Yes	Yes	
116	R	17.5	MISRA C 2012	Advisory	Yes	Yes	
117	R	17.6	MISRA C 2012	Mandatory	Yes	Yes	
118	R	17.7	MISRA C 2012	Required	Yes	Optional	In some cases it does
119	R	17.8	MISRA C 2012	Advisory	Yes	Optional	It would make some
120	R	18.1	MISRA C 2012	Required	Yes	Yes	
121	R	18.2	MISRA C 2012	Required	Yes	Yes	
122	R	18.3	MISRA C 2012	Required	Yes	Yes	

123	R	18.4	MISRA C 2012	Advisory	Yes	Yes	
124	R	18.5	MISRA C 2012	Advisory	Yes	Yes	
125	R	18.6	MISRA C 2012	Required	Yes	Yes	
126	R	18.7	MISRA C 2012	Required	Yes	Yes	
127	R	18.8	MISRA C 2012	Required	Yes	Yes	
128	R	19.1	MISRA C 2012	Mandatory	Yes	Yes	
129	R	19.2	MISRA C 2012	Advisory	Yes	Optional	Unions can be usefu
130	R	20.1	MISRA C 2012	Advisory	Yes	Optional	In some files we hav
131	R	20.2	MISRA C 2012	Required	Yes	Yes	
132	R	20.3	MISRA C 2012	Required	Yes	Yes	
133	R	20.4	MISRA C 2012	Required	Yes	Yes	
134	R	20.5	MISRA C 2012	Advisory	Yes	Yes	
135	R	20.6	MISRA C 2012	Required	Yes	Yes	
136	R	20.7	MISRA C 2012	Required	Yes	Yes	
137	R	20.8	MISRA C 2012	Required	Yes	Optional	We need a new conf
138	R	20.9	MISRA C 2012	Required	Yes	Optional	We use a mix of #if
139	R	20.10	MISRA C 2012	Advisory	Yes	Optional	It's good to avoid the
140	R	20.11	MISRA C 2012	Required	Yes	Yes	
141	R	20.12	MISRA C 2012	Required	Yes	Yes	
142	R	20.13	MISRA C 2012	Required	Yes	Yes	
143	R	20.14	MISRA C 2012	Required	Yes	Yes	
144	R	21.1	MISRA C 2012	Required	Yes	Yes	
145	R	21.2	MISRA C 2012	Required	Yes	Yes	
146	R	21.3	MISRA C 2012	Required	Yes	Yes	
147	R	21.4	MISRA C 2012	Required	Yes	Yes	
148	R	21.5	MISRA C 2012	Required	Yes	Yes	
149	R	21.6	MISRA C 2012	Required	No	No	This bans printf.
150	R	21.7	MISRA C 2012	Required	Yes	Yes	
151	R	21.8	MISRA C 2012	Required	Yes	Yes	
152	R	21.9	MISRA C 2012	Required	Yes	Yes	
153	R	21.10	MISRA C 2012	Required	Yes	Yes	
154	R	21.11	MISRA C 2012	Required	Yes	Yes	
155	R	21.12	MISRA C 2012	Advisory	Yes	Yes	
156	R	21.13	MISRA C 2012 AMD-1	Mandatory	Yes	Yes	
157	R	21.14	MISRA C 2012 AMD-1	Required	Yes	Yes	
158	R	21.15	MISRA C 2012 AMD-1	Required	Yes	Yes	
159	R	21.16	MISRA C 2012 AMD-1	Required	Yes	Yes	
160	R	21.17	MISRA C 2012 AMD-1	Mandatory	Yes	Yes	
161	R	21.18	MISRA C 2012 AMD-1	Mandatory	Yes	Yes	
162	R	21.19	MISRA C 2012 AMD-1	Mandatory	Yes	Yes	
163	R	21.20	MISRA C 2012 AMD-1	Mandatory	Yes	Yes	
164	R	22.1	MISRA C 2012	Required	Yes	Yes	
165	R	22.2	MISRA C 2012	Mandatory	Yes	Yes	
166	R	22.3	MISRA C 2012	Required	Yes	Yes	

167	R	22.4	MISRA C 2012	Mandatory	Yes	Yes	
168	R	22.5	MISRA C 2012	Mandatory	Yes	Yes	
169	R	22.6	MISRA C 2012	Mandatory	Yes	Yes	
170	R	22.7	MISRA C 2012 AMD-1	Required	Yes	Yes	
171	R	22.8	MISRA C 2012 AMD-1	Required	Yes	Yes	
172	R	22.9	MISRA C 2012 AMD-1	Required	Yes	Yes	
173	R	22.10	MISRA C 2012 AMD-1	Required	Yes	Yes	

3.4.5 Indentation

Use **tabs** for indentation. The use of spaces for indentation is forbidden except in the case where a term is being indented to a boundary that cannot be achieved using tabs alone.

Tab spacing should be set to **8 characters**.

Trailing whitespace is not allowed and must be trimmed.

3.4.6 Spacing

Single spacing should be used around most operators, including:

- Arithmetic operators (`+`, `-`, `/`, `*`)
- Assignment operators (=, +=, etc)
- Boolean operators (`&&`, `||`)
- Comparison operators (<, >, ==, etc)

A space should also be used to separate parentheses and braces when they are not already separated by a newline, such as for the `if` statement in the following example:

```
int function_foo(bool bar)
{
    if (bar) {
        function_baz();
    }
}
```

Note that there is no space between the name of a function and the following parentheses.

Control statements (`if`, `for`, `switch`, `while`, etc) must be separated from the following open parenthesis by a single space. The previous example illustrates this for an `if` statement.

3.4.7 Line Length

Line length *should* be at most **80 characters**. This limit does not include non-printing characters such as the line feed.

This rule is a *should*, not a must, and it is acceptable to exceed the limit **slightly** where the readability of the code would otherwise be significantly reduced. Use your judgement in these cases.

3.4.8 Blank Lines

Functions are usually separated by a single blank line. In certain cases it is acceptable to use additional blank lines for clarity, if required.

The file must end with a single newline character. Many editors have the option to insert this automatically and to trim multiple blank lines at the end of the file.

3.4.9 Braces

Opening Brace Placement

Braces follow the **Kernighan and Ritchie (K&R)** style, where the opening brace is **not** placed on a new line.

Example for a `while` loop:

```
while (condition) {
    foo();
    bar();
}
```

This style applies to all blocks except for functions which, following the Linux style, **do** place the opening brace on a new line.

Example for a function:

```
int my_function(void)
{
    int a;

    a = 1;
    return a;
}
```

Conditional Statement Bodies

Where conditional statements (such as `if`, `for`, `while` and `do`) are used, braces must be placed around the statements that form the body of the conditional. This is the case regardless of the number of statements in the body.

Note: This is a notable departure from the Linux coding style that has been adopted to follow MISRA guidelines more closely and to help prevent errors.

For example, use the following style:

```
if (condition) {
    foo++;
}
```

instead of omitting the optional braces around a single statement:

```
/* This is violating MISRA C 2012: Rule 15.6 */
if (condition)
    foo++;
```

The reason for this is to prevent accidental changes to control flow when modifying the body of the conditional. For example, at a quick glance it is easy to think that the value of `bar` is only incremented if `condition` evaluates to `true` but this is not the case - `bar` will always be incremented regardless of the condition evaluation. If the developer forgets to add braces around the conditional body when adding the `bar++;` statement then the program execution will not proceed as intended.

```
/* This is violating MISRA C 2012: Rule 15.6 */
if (condition)
    foo++;
    bar++;
```

3.4.10 Naming

Functions

Use lowercase for function names, separating multiple words with an underscore character (_). This is sometimes referred to as *Snake Case*. An example is given below:

```
void bl2_arch_setup(void)
{
    ...
}
```

Local Variables and Parameters

Local variables and function parameters use the same format as function names: lowercase with underscore separation between multiple words. An example is given below:

```
static void set_scr_el3_from_rm(uint32_t type,
                                uint32_t interrupt_type_flags,
                                uint32_t security_state)
{
    uint32_t flag, bit_pos;

    ...

}
```

Preprocessor Macros

Identifiers that are defined using preprocessor macros are written in all uppercase text.

```
#define BUFFER_SIZE_BYTES 64
```

3.4.11 Function Attributes

Place any function attributes after the function type and before the function name.

```
void __init plat_arm_interconnect_init(void);
```

3.4.12 Alignment

Alignment should be performed primarily with tabs, adding spaces if required to achieve a granularity that is smaller than the tab size. For example, with a tab size of eight columns it would be necessary to use one tab character and two spaces to indent text by ten columns.

Switch Statement Alignment

When using `switch` statements, align each `case` statement with the `switch` so that they are in the same column.

```
switch (condition) {
case A:
    foo();
case B:
    bar();
default:
    baz();
}
```

Pointer Alignment

The reference and dereference operators (ampersand and *pointer star*) must be aligned with the name of the object on which they are operating, as opposed to the type of the object.

```
uint8_t *foo;

foo = &bar;
```

3.4.13 Comments

The general rule for comments is that the double-slash style of comment (//) is not allowed. Examples of the allowed comment formats are shown below:

```
/*
 * This example illustrates the first allowed style for multi-line comments.
 *
 * Blank lines within multi-lines are allowed when they add clarity or when
 * they separate multiple contexts.
 *
 */
```

```
/**************************************************************************
 * This is the second allowed style for multi-line comments.
 *
 * In this style, the first and last lines use asterisks that run the full
 * width of the comment at its widest point.
 *
 * This style can be used for additional emphasis.
 *
 *************************************************************************/
```

```
/* Single line comments can use this format */
```

```
/***************************************************************************
 * This alternative single-line comment style can also be used for emphasis.
 **************************************************************************/
```

3.4.14 Headers and inclusion

Header guards

For a header file called "some_driver.h" the style used by *TF-A* is:

```
#ifndef SOME_DRIVER_H
#define SOME_DRIVER_H

```

(continues on next page)

(continued from previous page)

```
<header content>

#endif /* SOME_DRIVER_H */
```

Include statement ordering

All header files that are included by a source file must use the following, grouped ordering. This is to improve readability (by making it easier to quickly read through the list of headers) and maintainability.

1. *System* includes: Header files from the standard *C* library, such as `stddef.h` and `string.h`.
2. *Project* includes: Header files under the `include/` directory within *TF-A* are *project* includes.
3. *Platform* includes: Header files relating to a single, specific platform, and which are located under the `plat/<platform_name>` directory within *TF-A*, are *platform* includes.

Within each group, `#include` statements must be in alphabetical order, taking both the file and directory names into account.

Groups must be separated by a single blank line for clarity.

The example below illustrates the ordering rules using some contrived header file names; this type of name reuse should be otherwise avoided.

```
#include <string.h>

#include <a_dir/example/a_header.h>
#include <a_dir/example/b_header.h>
#include <a_dir/test/a_header.h>
#include <b_dir/example/a_header.h>

#include "a_header.h"
```

The preferred approach for third-party headers is to include them immediately following system header files like in the example below, where the `version.h` header from the Mbed TLS library immediately follows the `stddef.h` system header.

```
/* system header files */
#include <stddef.h>

/* Mbed TLS header files */
#include <mbedtls/version.h>

/* project header files */
#include <drivers/auth/auth_mod.h>
#include <drivers/auth/tbbr_cot_common.h>

/* platform header files */
#include <platform_def.h>
```

Include statement variants

Two variants of the `#include` directive are acceptable in the *TF-A* codebase. Correct use of the two styles improves readability by suggesting the location of the included header and reducing ambiguity in cases where generic and platform-specific headers share a name.

For header files that are in the same directory as the source file that is including them, use the `"..."` variant.

For header files that are **not** in the same directory as the source file that is including them, use the `<...>` variant.

Example (bl1_fwu.c):

```
#include <assert.h>
#include <errno.h>
#include <string.h>

#include "bl1_private.h"
```

3.4.15 Typedefs

Avoid anonymous typedefs of structs/enums in headers

For example, the following definition:

```
typedef struct {
        int arg1;
        int arg2;
} my_struct_t;
```

is better written as:

```
struct my_struct {
        int arg1;
        int arg2;
};
```

This allows function declarations in other header files that depend on the struct/enum to forward declare the struct/enum instead of including the entire header:

```
struct my_struct;
void my_func(struct my_struct *arg);
```

instead of:

```
#include <my_struct.h>
void my_func(my_struct_t *arg);
```

Some TF definitions use both a struct/enum name **and** a typedef name. This is discouraged for new definitions as it makes it difficult for TF to comply with MISRA rule 8.3, which states that “All declarations of an object or function shall use the same names and type qualifiers”.

The Linux coding standards also discourage new typedefs and checkpatch emits a warning for this.

Existing typedefs will be retained for compatibility.

3.5 Coding Guidelines

This document provides some additional guidelines to consider when writing *TF-A* code. These are not intended to be strictly-enforced rules like the contents of the *Coding Style*.

3.5.1 Automatic Editor Configuration

Many of the rules given below (such as indentation size, use of tabs, and newlines) can be set automatically using the EditorConfig configuration file in the root of the repository: `.editorconfig`. With a supported editor, the rules set out in this file can be automatically applied when you are editing files in the *TF-A* repository.

Several editors include built-in support for EditorConfig files, and many others support its functionality through plugins.

Use of the EditorConfig file is suggested but is not required.

3.5.2 Automatic Compliance Checking

To assist with coding style compliance, the project Makefile contains two targets which both utilise the *checkpatch.pl* script that ships with the Linux source tree. The project also defines certain *checkpatch* options in the `.checkpatch.conf` file in the top-level directory.

Note: Checkpatch errors will gate upstream merging of pull requests. Checkpatch warnings will not gate merging but should be reviewed and fixed if possible.

To check the entire source tree, you must first download copies of `checkpatch.pl`, `spelling.txt` and `const_structs.checkpatch` available in the Linux master tree *scripts* directory, then set the `CHECKPATCH` environment variable to point to `checkpatch.pl` (with the other 2 files in the same directory) and build the *checkcodebase* target:

```
make CHECKPATCH=<path-to-linux>/linux/scripts/checkpatch.pl checkcodebase
```

To just check the style on the files that differ between your local branch and the remote master, use:

```
make CHECKPATCH=<path-to-linux>/linux/scripts/checkpatch.pl checkpatch
```

If you wish to check your patch against something other than the remote master, set the `BASE_COMMIT` variable to your desired branch. By default, `BASE_COMMIT` is set to `origin/master`.

Ignored Checkpatch Warnings

Some checkpatch warnings in the TF codebase are deliberately ignored. These include:

- `**WARNING: Use of volatile is usually wrong`: see Why the "volatile" type class should not be used . Although this document contains some very useful information, there are several legimate uses of the volatile keyword within the TF codebase.

Those warnings are already handled with `.checkpatch.conf` file in the top-level directory.

3.5.3 Performance considerations

Avoid printf and use logging macros

`debug.h` provides logging macros (for example, `WARN` and `ERROR`) which wrap `tf_log` and which allow the logging call to be compiled-out depending on the `make` command. Use these macros to avoid print statements being compiled unconditionally into the binary.

Each logging macro has a numerical log level:

```
#define LOG_LEVEL_NONE    0
#define LOG_LEVEL_ERROR   10
#define LOG_LEVEL_NOTICE  20
#define LOG_LEVEL_WARNING 30
#define LOG_LEVEL_INFO    40
#define LOG_LEVEL_VERBOSE 50
```

By default, all logging statements with a log level `<= LOG_LEVEL_INFO` will be compiled into debug builds and all statements with a log level `<= LOG_LEVEL_NOTICE` will be compiled into release builds. This can be overridden from the command line or by the platform makefile (although it may be necessary to clean the build directory first).

For example, to enable `VERBOSE` logging on FVP:

```
make PLAT=fvp LOG_LEVEL=50 all
```

Use const data where possible

For example, the following code:

```
struct my_struct {
        int arg1;
        int arg2;
};

void init(struct my_struct *ptr);

void main(void)
{
```

(continues on next page)

(continued from previous page)

```
        struct my_struct x;
        x.arg1 = 1;
        x.arg2 = 2;
        init(&x);
}
```

is better written as:

```
struct my_struct {
        int arg1;
        int arg2;
};

void init(const struct my_struct *ptr);

void main(void)
{
        const struct my_struct x = { 1, 2 };
        init(&x);
}
```

This allows the linker to put the data in a read-only data section instead of a writeable data section, which may result in a smaller and faster binary. Note that this may require dependent functions (`init()` in the above example) to have `const` arguments, assuming they don't need to modify the data.

3.5.4 Libc functions that are banned or to be used with caution

Below is a list of functions that present security risks and either must not be used (Banned) or are discouraged from use and must be used with care (Caution).

libc function	Status	Comments
`strcpy, wcscpy, strncpy`	Banned	use strlcpy instead
`strcat, wcscat, strncat`	Banned	use strlcat instead
`sprintf, vsprintf`	Banned	use snprintf, vsnprintf instead
`snprintf`	Caution	ensure result fits in buffer i.e : snprintf(buf,size...) < size
`vsnprintf`	Caution	inspect va_list match types specified in format string
`strtok`	Banned	use strtok_r or strsep instead
`strtok_r, strsep`	Caution	inspect for terminated input buffer
`ato*`	Banned	use equivalent strto* functions
`*toa`	Banned	Use snprintf instead

The *libc* component in the codebase will not add support for the banned APIs.

3.5.5 Error handling and robustness

Using CASSERT to check for compile time data errors

Where possible, use the `CASSERT` macro to check the validity of data known at compile time instead of checking validity at runtime, to avoid unnecessary runtime code.

For example, this can be used to check that the assembler's and compiler's views of the size of an array is the same.

```
#include <cassert.h>

define MY_STRUCT_SIZE 8 /* Used by assembler source files */

struct my_struct {
    uint32_t arg1;
    uint32_t arg2;
};

CASSERT(MY_STRUCT_SIZE == sizeof(struct my_struct), assert_my_struct_size_
↪mismatch);
```

If `MY_STRUCT_SIZE` in the above example were wrong then the compiler would emit an error like this:

```
my_struct.h:10:1: error: size of array 'assert_my_struct_size_mismatch' is␣
↪negative
```

Using assert() to check for programming errors

In general, each secure world TF image (BL1, BL2, BL31 and BL32) should be treated as a tightly integrated package; the image builder should be aware of and responsible for all functionality within the image, even if code within that image is provided by multiple entities. This allows us to be more aggressive in interpreting invalid state or bad function arguments as programming errors using `assert()`, including arguments passed across platform porting interfaces. This is in contrast to code in a Linux environment, which is less tightly integrated and may attempt to be more defensive by passing the error back up the call stack.

Where possible, badly written TF code should fail early using `assert()`. This helps reduce the amount of untested conditional code. By default these statements are not compiled into release builds, although this can be overridden using the `ENABLE_ASSERTIONS` build flag.

Examples:

- Bad argument supplied to library function
- Bad argument provided by platform porting function
- Internal secure world image state is inconsistent

Handling integration errors

Each secure world image may be provided by a different entity (for example, a Trusted Boot vendor may provide the BL2 image, a TEE vendor may provide the BL32 image and the OEM/SoC vendor may provide the other images).

An image may contain bugs that are only visible when the images are integrated. The system integrator may not even have access to the debug variants of all the images in order to check if asserts are firing. For example, the release variant of BL1 may have already been burnt into the SoC. Therefore, TF code that detects an integration error should _not_ consider this a programming error, and should always take action, even in release builds.

If an integration error is considered non-critical it should be treated as a recoverable error. If the error is considered critical it should be treated as an unexpected unrecoverable error.

Handling recoverable errors

The secure world **must not** crash when supplied with bad data from an external source. For example, data from the normal world or a hardware device. Similarly, the secure world **must not** crash if it detects a non-critical problem within itself or the system. It must make every effort to recover from the problem by emitting a `WARN` message, performing any necessary error handling and continuing.

Examples:

- Secure world receives SMC from normal world with bad arguments.
- Secure world receives SMC from normal world at an unexpected time.
- BL31 receives SMC from BL32 with bad arguments.
- BL31 receives SMC from BL32 at unexpected time.
- Secure world receives recoverable error from hardware device. Retrying the operation may help here.
- Non-critical secure world service is not functioning correctly.
- BL31 SPD discovers minor configuration problem with corresponding SP.

Handling unrecoverable errors

In some cases it may not be possible for the secure world to recover from an error. This situation should be handled in one of the following ways:

1. If the unrecoverable error is unexpected then emit an `ERROR` message and call `panic()`. This will end up calling the platform-specific function `plat_panic_handler()`.
2. If the unrecoverable error is expected to occur in certain circumstances, then emit an `ERROR` message and call the platform-specific function `plat_error_handler()`.

Cases 1 and 2 are subtly different. A platform may implement `plat_panic_handler` and `plat_error_handler` in the same way (for example, by waiting for a secure watchdog to time-out or by invoking an interface on the platform's power controller to reset the platform). However, `plat_error_handler` may take additional action for some errors (for example, it may set a flag so the platform resets into a different mode). Also, `plat_panic_handler()` may implement additional debug functionality (for example, invoking a hardware breakpoint).

Examples of unexpected unrecoverable errors:

- BL32 receives an unexpected SMC response from BL31 that it is unable to recover from.
- BL31 Trusted OS SPD code discovers that BL2 has not loaded the corresponding Trusted OS, which is critical for platform operation.
- Secure world discovers that a critical hardware device is an unexpected and unrecoverable state.
- Secure world receives an unexpected and unrecoverable error from a critical hardware device.
- Secure world discovers that it is running on unsupported hardware.

Examples of expected unrecoverable errors:

- BL1/BL2 fails to load the next image due to missing/corrupt firmware on disk.
- BL1/BL2 fails to authenticate the next image due to an invalid certificate.
- Secure world continuously receives recoverable errors from a hardware device but is unable to proceed without a valid response.

Handling critical unresponsiveness

If the secure world is waiting for a response from an external source (for example, the normal world or a hardware device) which is critical for continued operation, it must not wait indefinitely. It must have a mechanism (for example, a secure watchdog) for resetting itself and/or the external source to prevent the system from executing in this state indefinitely.

Examples:

- BL1 is waiting for the normal world to raise an SMC to proceed to the next stage of the secure firmware update process.
- A Trusted OS is waiting for a response from a proxy in the normal world that is critical for continued operation.
- Secure world is waiting for a hardware response that is critical for continued operation.

3.5.6 Use of built-in *C* and *libc* data types

The *TF-A* codebase should be kept as portable as possible, especially since both 64-bit and 32-bit platforms are supported. To help with this, the following data type usage guidelines should be followed:

- Where possible, use the built-in *C* data types for variable storage (for example, `char`, `int`, `long long`, etc) instead of the standard *C99* types. Most code is typically only concerned with the minimum size of the data stored, which the built-in *C* types guarantee.
- Avoid using the exact-size standard *C99* types in general (for example, `uint16_t`, `uint32_t`, `uint64_t`, etc) since they can prevent the compiler from making optimizations. There are legitimate uses for them, for example to represent data of a known structure. When using them in struct definitions,

consider how padding in the struct will work across architectures. For example, extra padding may be introduced in *AArch32* systems if a struct member crosses a 32-bit boundary.

- Use `int` as the default integer type - it's likely to be the fastest on all systems. Also this can be assumed to be 32-bit as a consequence of the Procedure Call Standard for the Arm Architecture and the Procedure Call Standard for the Arm 64-bit Architecture .
- Avoid use of `short` as this may end up being slower than `int` in some systems. If a variable must be exactly 16-bit, use `int16_t` or `uint16_t`.
- Avoid use of `long`. This is guaranteed to be at least 32-bit but, given that *int* is 32-bit on Arm platforms, there is no use for it. For integers of at least 64-bit, use `long long`.
- Use `char` for storing text. Use `uint8_t` for storing other 8-bit data.
- Use `unsigned` for integers that can never be negative (counts, indices, sizes, etc). TF intends to comply with MISRA "essential type" coding rules (10.X), where signed and unsigned types are considered different essential types. Choosing the correct type will aid this. MISRA static analysers will pick up any implicit signed/unsigned conversions that may lead to unexpected behaviour.
- For pointer types:
  - If an argument in a function declaration is pointing to a known type then simply use a pointer to that type (for example: `struct my_struct *`).
  - If a variable (including an argument in a function declaration) is pointing to a general, memory-mapped address, an array of pointers or another structure that is likely to require pointer arithmetic then use `uintptr_t`. This will reduce the amount of casting required in the code. Avoid using `unsigned long` or `unsigned long long` for this purpose; it may work but is less portable.
  - For other pointer arguments in a function declaration, use `void *`. This includes pointers to types that are abstracted away from the known API and pointers to arbitrary data. This allows the calling function to pass a pointer argument to the function without any explicit casting (the cast to `void *` is implicit). The function implementation can then do the appropriate casting to a specific type.
  - Avoid pointer arithmetic generally (as this violates MISRA C 2012 rule 18.4) and especially on void pointers (as this is only supported via language extensions and is considered non-standard). In TF-A, setting the `W` build flag to `W=3` enables the *-Wpointer-arith* compiler flag and this will emit warnings where pointer arithmetic is used.
  - Use `ptrdiff_t` to compare the difference between 2 pointers.
- Use `size_t` when storing the `sizeof()` something.
- Use `ssize_t` when returning the `sizeof()` something from a function that can also return an error code; the signed type allows for a negative return code in case of error. This practice should be used sparingly.
- Use `u_register_t` when it's important to store the contents of a register in its native size (32-bit in *AArch32* and 64-bit in *AArch64*). This is not a standard *C99* type but is widely available in libc implementations, including the FreeBSD version included with the TF codebase. Where possible, cast the variable to a more appropriate type before interpreting the data. For example, the following struct in `ep_info.h` could use this type to minimize the storage required for the set of registers:

```
typedef struct aapcs64_params {
        u_register_t arg0;
        u_register_t arg1;
        u_register_t arg2;
        u_register_t arg3;
        u_register_t arg4;
        u_register_t arg5;
        u_register_t arg6;
        u_register_t arg7;
} aapcs64_params_t;
```

If some code wants to operate on `arg0` and knows that it represents a 32-bit unsigned integer on all systems, cast it to `unsigned int`.

These guidelines should be updated if additional types are needed.

3.5.7 Favor C language over assembly language

Generally, prefer code written in C over assembly. Assembly code is less portable, harder to understand, maintain and audit security wise. Also, static analysis tools generally don't analyze assembly code.

If specific system-level instructions must be used (like cache maintenance operations), please consider using inline assembly. The `arch_helpers.h` files already define inline functions for a lot of these.

There are, however, legitimate uses of assembly language. These are usually early boot (eg. cpu reset sequences) and exception handling code before the C runtime environment is set up.

When writing assembly please note that a wide variety of common instruction sequences have helper macros in `asm_macros.S` which are preferred over writing them directly. This is especially important for debugging purposes as debug symbols must manually be included. Please use the `func_prologue` and `func_epilogue` macros if you need to use the stack. Also, obeying the Procedure Call Standard (PCS) is generally recommended.

3.5.8 Do not use weak functions

Note: The following guideline applies more strongly to common, platform-independent code. For plaform code (under `plat/` directory), it is up to each platform maintainer to decide whether this should be striclty enforced or not.

The use of weak functions is highly discouraged in the TF-A codebase. Newly introduced platform interfaces should be strongly defined, wherever possible. In the rare cases where this is not possible or where weak functions appear as the best tool to solve the problem at hand, this should be discussed with the project's maintainers and justified in the code.

For the purpose of providing a default implementation of a platform interface, an alternative to weak functions is to provide a strongly-defined implementation under the `plat/common/` directory. Then platforms have two options to pull in this implementation:

- They can include the source file through the platform's makefile. Note that this method is suitable only if the platform wants *all* default implementations defined in this file, else either the file should be refactored or the next approach should be used.
- They access the platform interface through a **constant** function pointer.

In both cases, what matters is that platforms include the default implementation as a conscious decision.

Rationale

Weak functions may sound useful to simplify the initial porting effort to a new platform, such that one can quickly get the firmware to build and link, without implementing all platform interfaces from the beginning. For this reason, the TF-A project used to make heavy use of weak functions and there are still many outstanding usages of them across the code base today. We intend to convert them to strongly-defined functions over time.

However, weak functions also have major drawbacks, which we consider outweighing their benefits. They can make it hard to identify which implementation gets built into the firmware, especially when using multiple levels of "weakness". This has resulted in bugs in the past.

Weak functions are also forbidden by MISRA coding guidelines, which TF-A aims to comply with.

3.6 Contributor's Guide

3.6.1 Getting Started

- Make sure you have a Github account and you are logged on to review.trustedfirmware.org.

  Also make sure that you have registered your full name and email address in your review.trustedfirmware.org profile. Otherwise, the Gerrit server might reject patches you attempt to post for review.

- If you plan to contribute a major piece of work, it is usually a good idea to start a discussion around it on the TF-A mailing list. This gives everyone visibility of what is coming up, you might learn that somebody else is already working on something similar or the community might be able to provide some early input to help shaping the design of the feature.

  If you intend to include Third Party IP in your contribution, please mention it explicitly in the email thread and ensure that the changes that include Third Party IP are made in a separate patch (or patch series).

- Clone the Trusted Firmware-A source code on your own machine as described in *Additional Steps for Contributors*.
- Create a local topic branch based on the Trusted Firmware-A `master` branch.

3.6.2 Making Changes

- Ensure commits adhere to the project's *Commit Style*.
- Make commits of logical units. See these general Git guidelines for contributing to a project.
- Keep the commits on topic. If you need to fix another bug or make another enhancement, please address it on a separate topic branch.
- Split the patch in manageable units. Small patches are usually easier to review so this will speed up the review process.
- Avoid long commit series. If you do have a long series, consider whether some commits should be squashed together or addressed in a separate topic.
- Follow the *Coding Style* and *Coding Guidelines*.
  - Use the checkpatch.pl script provided with the Linux source tree. A Makefile target is provided for convenience, see *this section* for more details.
- If a patch should be considered for LTS backporting, keep the normal *TF-A* commit subject format and add an `LTS-candidate: yes` trailer in the footer block at the end of the commit message, alongside the other commit trailers. This tag acts as an indication to the LTS maintainers that the patch should be evaluated as a candidate for the next LTS release. It also helps the LTS weekly triage tooling to automatically pull the patch in so it can be reviewed among the maintainers. For example:

  ```
  fix(psci): get the cpu_ops before exiting coherency

  Dereference the cpu_ops pointer before disabling coherency so the
  warm path does not observe stale data from cache.

  Signed-off-by: Developer Name <developer@example.com>
  Change-Id: I0123456789abcdef0123456789abcdef01234567
  LTS-candidate: yes
  ```

  Reviewers and maintainers should use this trailer as the preferred machine-readable signal for LTS triage automation, but LTS maintainers retain the final say on whether a patch is taken into an LTS branch. They should also look for LTS-eligible patches that were posted without it and ask authors to add it in follow-up revisions where practical.
- Where appropriate, please update the documentation.
  - Consider whether the *Porting Guide*, *Firmware Design* document or other in-source documentation needs updating.
  - If you are submitting new files that you intend to be the code owner for (for example, a new platform port), then also update the *Code owners* file.
  - For topics with multiple commits, you should make all documentation changes (and nothing else) in the last commit of the series. Otherwise, include the documentation changes within the single commit.
- Ensure that each changed file has the correct copyright and license information. Files that entirely consist of contributions to this project should have a copyright notice and BSD-3-Clause SPDX license identifier

of the form as shown in *License*. Files that contain changes to imported Third Party IP files should retain their original copyright and license notices.

For significant contributions you may add your own copyright notice in the following format:

```
Portions copyright (c) [XXXX-]YYYY, <OWNER>. All rights reserved.
```

where XXXX is the year of first contribution (if different to YYYY) and YYYY is the year of most recent contribution. <OWNER> is your name or your company name.

- Ensure that each patch in the patch series compiles in all supported configurations. Patches which do not compile will not be merged.
- Please test your changes. As a minimum, ensure that Linux boots on the Foundation FVP. See *Arm Fixed Virtual Platforms (FVP)* for more information. For more extensive testing, consider running the TF-A Tests against your patches.
- Ensure that all CI automated tests pass. Failures should be fixed. They might block a patch, depending on how critical they are.

3.6.3 Submitting Changes

Note: Please follow the How to Contribute Code section of the OpenCI documentation for general instructions on setting up Gerrit and posting patches there. The rest of this section provides details about patch submission rules specifically for the TF-A project.

- Submit your changes for review using the `git review` command.

  This will automatically rebase them onto the upstream `integration` branch, as required by TF-A's patch submission process.
- From the Gerrit web UI, add reviewers for your patch:
  - At least one code owner for each module modified by the patch. See the list of modules and their *Code owners*.
  - At least one maintainer. See the list of *Maintainers*.
  - If some module has no code owner, try to identify a suitable (non-code owner) reviewer. Running `git blame` on the module's source code can help, as it shows who has been working the most recently on this area of the code.

    Alternatively, if it is impractical to identify such a reviewer, you might send an email to the TF-A mailing list to broadcast your review request to the community.

  Note that self-reviewing a patch is prohibited, even if the patch author is the only code owner of a module modified by the patch. Getting a second pair of eyes on the code is essential to keep up with the quality standards the project aspires to.
- The changes will then undergo further review by the designated people. Any review comments will be made directly on your patch. This may require you to do some rework. For controversial changes, the discussion might be moved to the TF-A mailing list to involve more of the community.

Refer to the Gerrit Uploading Changes documentation for more details.

- The patch submission rules are the following. For a patch to be approved and merged in the tree, it must get:
  - One `Code-Owner-Review+1` for each of the modules modified by the patch.
  - A `Maintainer-Review+1`.

  In the case where a code owner could not be found for a given module, `Code-Owner-Review+1` is substituted by `Code-Review+1`.

  In addition to these various code review labels, the patch must also get a `Verified+1`. This is usually set by the Continuous Integration (CI) bot when all automated tests passed on the patch. Sometimes, some of these automated tests may fail for reasons unrelated to the patch. In this case, the maintainers might (after analysis of the failures) override the CI bot score to certify that the patch has been correctly tested.

  In the event where the CI system lacks proper tests for a patch, the patch author or a reviewer might agree to perform additional manual tests in their review and the reviewer incorporates the review of the additional testing in the `Code-Review+1` or `Code-Owner-Review+1` as applicable to attest that the patch works as expected. Where possible additional tests should be added to the CI system as a follow up task. For example, for a platform-dependent patch where the said platform is not available in the CI system's board farm.

- When the changes are accepted, the *Maintainers* will integrate them.
  - Typically, the *Maintainers* will merge the changes into the `integration` branch.
  - If the changes are not based on a sufficiently-recent commit, or if they cannot be automatically rebased, then the *Maintainers* may rebase it on the `integration` branch or ask you to do so.
  - After final integration testing, the changes will make their way into the `master` branch. If a problem is found during integration, the *Maintainers* will request your help to solve the issue. They may revert your patches and ask you to resubmit a reworked version of them or they may ask you to provide a fix-up patch.

3.6.4 Add CI Configurations

TF-A uses Jenkins for Continuous Integration and testing activities. Various CI jobs are deployed to run tests on every patch before being merged. Each of your patches go through a series of checks before they get merged on to the master branch. Kindly ensure that every time you add new files under your platform, they are covered by the following two sections.

Coverity Scan

The TF-A project makes use of *Coverity Scan* for static analysis, a service offered by Synopsys for open-source projects. This tool is able to find defects and vulnerabilities in a code base, such as dereferences of NULL pointers, use of uninitialized data, control flow issues and many other things.

The TF-A source code is submitted daily to this service for analysis. Results of the latest and previous scans, as well as the complete list of defects it detected, are accessible online from https://scan.coverity.com/projects/arm-software-arm-trusted-firmware.

The tf-a-ci-scripts repository contains scripts to run the Coverity Scan tools on the integration branch of the TF-A code base and make them available on https://scan.coverity.com. These scripts get executed daily by the tf-a-coverity Jenkins job.

In order to maintain a high level of coverage, including on newly introduced code, it is important to maintain the appropriate TF-A CI scripts. Details of when to update these scripts and how to do so follow.

We maintain a build script - `tf-cov-make` - which contains the build configurations of various platforms in order to cover the entire source code being analysed by Coverity.

When you submit your patches for review, and if they contain new source files, TF-A CI static checks job might report that these files are not covered. In this case, the job's console output will show the following error message:

```
****** Newly added files detection check for Coverity Scan analysis on␣
↪patch(es) ******

Result : FAILURE

New source files have been identified in your patch..
some/dir/file.c

please ensure to include them for the ``Coverity Scan analysis`` by adding
the respective build configurations in the ``tf-cov-make`` build script.
```

In this section you find the details on how to append your new build configurations for Coverity scan analysis illustrated with examples:

1. We maintain a separate repository named tf-a-ci-scripts repository for placing all the test scripts which will be executed by the CI Jobs.
2. In this repository, `tf-cov-make` script is located at `tf-a-ci-scripts/script/tf-coverity/tf-cov-make`
3. Edit the tf-cov-make script by appending all the possible build configurations with the specific build flags relevant to your platform, so that newly added source files get built and analysed by Coverity.
4. For better understanding follow the below specified examples listed in the `tf-cov-make` script.

```
Example 1:
#Intel
make PLAT=stratix10 $(common_flags) all
make PLAT=agilex $(common_flags) all
```

- In the above example there are two different SoCs `stratix` and `agilex` under the Intel platform and the build configurations has been added suitably to include most of their source files.

```
Example 2:
#Hikey
make PLAT=hikey $(common_flags) ${TBB_OPTIONS} ENABLE_PMF=1 all
make PLAT=hikey960 $(common_flags) ${TBB_OPTIONS} all
make PLAT=poplar $(common_flags) all
```

- In this case for `Hikey` boards additional build flags have been included along with the `common_flags` to cover most of the files relevant to it.
- Similar to this you can still find many other different build configurations of various other platforms listed in the `tf-cov-make` script. Kindly refer them and append your build configurations respectively.

Test Build Configurations

We have CI jobs which run a set of test configurations on every TF-A patch before they get merged upstream.

At the bare minimum, TF-A code should build without any errors for every supported platform - and every feature of this platform. To make sure this is the case, we maintain a set of build tests. `tf-l1-build-plat` is the test group which holds all build tests for all platforms. So be kind enough to verify that your newly added files are covered by such a build test.

If this is not the case, please follow the instructions below to add the appropriate files. We will illustrate this with an example for the `Hikey` platform.

- In the tf-a-ci-scripts repository we need to add a build configuration file `hikey-default` under `tf_config/` folder. `tf_config/hikey-default` must list all the build parameters relevant to it.

```
# Hikey Build Parameters
CROSS_COMPILE=aarch64-none-elf-
PLAT=hikey
```

- Further another file, `hikey-default:nil`, needs to be added under `group/tf-l1-build-plat/` folder to allow the platform to be built as part of this test group. `group/tf-l1-build-plat/hikey-default:nil` file just needs to exist but does not contain anything meaningful, apart from a mandatory copyright notice:

```
#
# Copyright (c) 2019-2022 Arm Limited. All rights reserved.
#
# SPDX-License-Identifier: BSD-3-Clause
#
```

- As illustrated above, you need to add similar files supporting your platform.

For a more elaborate explanation of the TF-A CI scripts internals, including how to add more complex tests beyond a simple build test, please refer to the TF-A CI scripts overview section of the OpenCI documentation.

3.6.5 Binary Components

- Platforms may depend on binary components submitted to the Trusted Firmware binary repository if they require code that the contributor is unable or unwilling to open-source. This should be used as a rare exception.
- All binary components must follow the contribution guidelines (in particular licensing rules) outlined in the readme.rst file of the binary repository.
- Binary components must be restricted to only the specific functionality that cannot be open-sourced and must be linked into a larger open-source platform port. The majority of the platform port must still be implemented in open source. Platform ports that are merely a thin wrapper around a binary component that contains all the actual code will not be accepted.
- Only platform port code (i.e. in the `plat/<vendor>` directory) may rely on binary components. Generic code must always be fully open-source.

3.7 Code Review Guidelines

3.7.1 Why do we do code reviews?

The main goal of code reviews is to improve the code quality. By reviewing each other's code, we can help catch issues that were missed by the author before they are integrated in the source tree. Different people bring different perspectives, depending on their past work, experiences and their current use cases of TF-A in their products.

Code reviews also play a key role in sharing knowledge within the community. People with more expertise in one area of the code base can help those that are less familiar with it.

Code reviews are meant to benefit everyone through team work. It is not about unfairly criticizing or belittling the work of any contributor.

3.7.2 Overview of the code review process

All contributions to Trusted Firmware-A project are reviewed by the community to ensure they meet the project's expectations before they get merged, according to the Project Maintenance Process defined for all *Trusted Firmware* projects.

Technical ownership of most parts of the codebase falls on the *Code owners*. All patches are ultimately merged by the *Maintainers*.

Approval of a patch is tracked using Gerrit *labels*. For a patch to be merged, it must get all of the following votes:

- At least one `Code-Owner-Review+1` up-vote, and no `Code-Owner-Review-1` down-vote.
- At least one `Maintainer-Review+1` up-vote, and no `Maintainer-Review-1` down-vote.

- `Verified+1` vote applied by the automated Continuous Integration (CI) system.

Note that, in some instances, the maintainers might give a waiver for some of the CI failures and manually override the `Verified+1` score.

3.7.3 Good practices for all reviewers

To ensure the code review gives the greatest possible benefit, participants in the project should:

- Be considerate of other people and their needs. Participants may be working to different timescales, and have different priorities. Keep this in mind - be gracious while waiting for action from others, and timely in your actions when others are waiting for you.
- Review other people's patches where possible. The more active reviewers there are, the more quickly new patches can be reviewed and merged. Contributing to code review helps everyone in the long run, as it creates a culture of participation which serves everyone's interests.

3.7.4 Guidelines for patch contributors

In addition to the rules outlined in the *Contributor's Guide*, as a patch contributor you are expected to:

- Answer all comments from people who took the time to review your patches.
- Be patient and resilient. It is quite common for patches to go through several rounds of reviews and rework before they get approved, especially for larger features.

  In the event that a code review takes longer than you would hope for, you may try the following actions to speed it up:
- Ping the reviewers on Gerrit or on the mailing list. If it is urgent, explain why. Please remain courteous and do not abuse this.
- If one code owner has become unresponsive, ask the other code owners for help progressing the patch.
- If there is only one code owner and they have become unresponsive, ask one of the project maintainers for help.
- Do the right thing for the project, not the fastest thing to get code merged.

  For example, if some existing piece of code - say a driver - does not quite meet your exact needs, go the extra mile and extend the code with the missing functionality you require - as opposed to copying the code into some other directory to have the freedom to change it in any way. This way, your changes benefit everyone and will be maintained over time.
- It is the patch-author's responsibility to respond to review comments within 21 days. In the event that the patch-author does not respond within this timeframe, the maintainer is entitled to abandon the patch(es). Patch author(s) may be busy with other priorities, causing a delay in responding to active review comments after posting patch(es). In such a situation, if the author's patch(es) is/are abandoned, they can restore their work for review by resolving comments, merge-conflicts, and revising their original submissions.

3.7.5 Guidelines for all reviewers

There are no good or bad review comments. If you have any doubt about a patch or need some clarifications, it's better to ask rather than letting a potential issue slip. Examples of review comments could be:

- Questions ("Why do you need to do this?", "What if X happens?")
- Bugs ("I think you need a logical || rather than a bitwise |.")
- Design issues ("This won't scale well when we introduce feature X.")
- Improvements ("Would it be better if we did Y instead?")

3.7.6 Guidelines for code owners

Code owners are listed on the *Project Maintenance* page, along with the module(s) they look after.

When reviewing a patch, code owners are expected to check the following:

- The patch looks good from a technical point of view. For example:
- The structure of the code is clear.
- It complies with the relevant standards or technical documentation (where applicable).
- It leverages existing interfaces rather than introducing new ones unnecessarily.
- It fits well in the design of the module.
- It adheres to the security model of the project. In particular, it does not increase the attack surface (e.g. new SMCs) without justification.
- The patch adheres to the TF-A *Coding Style*. The CI system should help catch coding style violations.
- (Only applicable to generic code) The code is MISRA-compliant (see *MISRA Compliance*). The CI system should help catch violations.
- Documentation is provided/updated (where applicable).
- The patch has had an appropriate level of testing. Testing details are expected to be provided by the patch author. If they are not, do not hesitate to request this information.
- All CI automated tests pass.

If a code owner is happy with a patch, they should give their approval through the `Code-Owner-Review+1` label in Gerrit. If instead, they have concerns, questions, or any other type of blocking comment, they should set `Code-Owner-Review-1`.

Code owners are expected to behave professionally and responsibly. Here are some guidelines for them:

- Once you are engaged in a review, make sure you stay involved until the patch is merged. Rejecting a patch and going away is not very helpful. You are expected to monitor the patch author's answers to your review comments, answer back if needed and review new revisions of their patch.
- Provide constructive feedback. Just saying, "This is wrong, you should do X instead." is usually not very helpful. The patch author is unlikely to understand why you are requesting this change and might feel personally attacked.

- Be mindful when reviewing a patch. As a code owner, you are viewed as the expert for the relevant module. By approving a patch, you are partially responsible for its quality and the effects it has for all TF-A users. Make sure you fully understand what the implications of a patch might be.

3.7.7 Guidelines for maintainers

Maintainers are listed on the *Project Maintenance* page.

When reviewing a patch, maintainers are expected to check the following:

- The general structure of the patch looks good. This covers things like:
  - Code organization.
  - Files and directories, names and locations.

    For example, platform code should be added under the `plat/` directory.
  - Naming conventions.

    For example, platform identifiers should be properly namespaced to avoid name clashes with generic code.
  - API design.
- Interaction of the patch with other modules in the code base.
- The patch aims at complying with any standard or technical documentation that applies.
- New files must have the correct license and copyright headers. See *this paragraph* for more information. The CI system should help catch files with incorrect or no copyright/license headers.
- There is no third party code or binary blobs with potential IP concerns. Maintainers should look for copyright or license notices in code, and use their best judgement. If they are unsure about a patch, they should ask other maintainers for help.
- Generally speaking, new driver code should be placed in the generic layer. There are cases where a driver has to stay into the platform layer but this should be the exception, rather than the rule.
- Existing common drivers (in particular for Arm IPs like the GIC driver) should not be copied into the platform layer to cater for platform quirks. This type of code duplication hurts the maintainability of the project. The duplicate driver is less likely to benefit from bug fixes and future enhancements. In most cases, it is possible to rework a generic driver to make it more flexible and fit slightly different use cases. That way, these enhancements benefit everyone.
- When a platform specific driver really is required, the burden lies with the patch author to prove the need for it. A detailed justification should be posted via the commit message or on the mailing list.
- Before merging a patch, verify that all review comments have been addressed. If this is not the case, encourage the patch author and the relevant reviewers to resolve these together.

If a maintainer is happy with a patch, they should give their approval through the `Maintainer-Review+1` label in Gerrit. If instead, they have concerns, questions, or any other type of blocking comment, they should set `Maintainer-Review-1`.

3.8 Frequently-Asked Questions (FAQ)

3.8.1 How do I update my changes?

Often it is necessary to update your patch set before it is merged. Refer to the Gerrit Upload Patch Set documentation on how to do so.

If you need to modify an existing patch set with multiple commits, refer to the Gerrit Replace Changes documentation.

3.8.2 How long will my changes take to merge into `integration`?

This can vary a lot, depending on:

- How important the patch set is considered by the TF maintainers. Where possible, you should indicate the required timescales for merging the patch set and the impact of any delay. Feel free to add a comment to your patch set to get an estimate of when it will be merged.
- The quality of the patch set. Patches are likely to be merged more quickly if they follow the coding guidelines, have already had some code review, and have been appropriately tested.
- The impact of the patch set. For example, a patch that changes a key generic API is likely to receive much greater scrutiny than a local change to a specific platform port.
- How much opportunity for external review is required. For example, the TF maintainers may not wait for external review comments to merge trivial bug-fixes but may wait up to a week to merge major changes, or ones requiring feedback from specific parties.
- How many other patch sets are waiting to be integrated and the risk of conflict between the topics.
- If there is a code freeze in place in preparation for the release. Please refer the *Release Processes* document for more details.
- The workload of the TF maintainers.

3.8.3 How long will it take for my changes to go from `integration` to `master`?

This depends on how many concurrent patches are being processed at the same time. In simple cases where all potential regressions have already been tested, the delay will be less than 1 day. If the TF maintainers are trying to merge several things over the course of a few days, it might take up to a week. Typically, it will be 1-2 days.

The worst case is if the TF maintainers are trying to make a release while also receiving patches that will not be merged into the release. In this case, the patches will be merged onto `integration`, which will temporarily diverge from the release branch. The `integration` branch will be rebased onto `master` after the release, and then `master` will be fast-forwarded to `integration` 1-2 days later. This whole process could take up

4 weeks. Please refer to the *Release Processes* document for code freeze dates. The TF maintainers will inform the patch owner if this is going to happen.

It is OK to create a patch based on commits that are only available in `integration` or another patch set, rather than `master`. There is a risk that the dependency commits will change (for example due to patch set rework or integration problems). If this happens, the dependent patch will need reworking.

3.8.4 What are these strange comments in my changes?

All the comments from `TrustedFirmware Code Review` user (email: `ci@trustedfirmware.org`) are associated with Continuous Integration (CI) infrastructure. The links published on the comments redirect to the CI web interface at http://ci.trustedfirmware.org, where details of the tests failures, if any, can be examined.

3.9 Project Maintenance Processes

Trusted Firmware-A (TF-A) project follows the generic trustedfirmware.org Project Maintenance Process. The present document complements it by defining TF-A project-specific decisions.

3.9.1 How to become a maintainer?

Qualifying Criteria

To be elligible to become a maintainer for TF-A project, all criteria outlined here must be fullfilled. These are:

- Being an active member of the project for at least a couple of years.
- Having contributed a substantial number of non-trivial and high-quality patches.
- Having reviewed a substantial number of non-trivial patches, preferably in the generic layer, with high-quality constructive feedback.
- Behaving in a professional and polite way, with the best interests of the project at heart.
- Showing a strong will to improve the project and to do the right thing, rather than going for the quick and easy path.
- Participating in design discussions on the development mailing list and during TF-A tech forums calls.
- Having appropriate bandwidth (minimum 2 hours per week) to deal with the workload.

Election Process

To put an individual's name up for election,

1. Send an email to all existing TF-A maintainers, asking whether they have any objections to this individual becoming a TF-A maintainer.
2. Give existing maintainers one calendar week to participate in the discussion.
3. If there are objections, the existing maintainers should try to resolve them amongst themselves. If they cannot, this should be escalated to the trustedfirmware.org Technical Steering Commitee (TSC).
4. If there are no (more) objections, announce the news on the TF-A mailing list and update the list of maintainers on the *Project Maintenance* page.

3.10 Secure Development Guidelines

This page contains guidance on what to check for additional security measures, including build options that can be modified to improve security or catch issues early in development.

3.10.1 Security considerations

Part of the security of a platform is handling errors correctly, as described in the previous section. There are several other security considerations covered in this section.

Do not leak secrets to the normal world

The secure world **must not** leak secrets to the normal world, for example in response to an SMC.

Handling Denial of Service attacks

The secure world **should never** crash or become unusable due to receiving too many normal world requests (a *Denial of Service* or *DoS* attack). It should have a mechanism for throttling or ignoring normal world requests.

Preventing Secure-world timing information leakage via PMU counters

The Secure world needs to implement some defenses to prevent the Non-secure world from making it leak timing information. In general, higher privilege levels must defend from those below when the PMU is treated as an attack vector.

Refer to the *Performance Monitoring Unit* guide for detailed information on the PMU registers.

Timing leakage attacks from the Non-secure world

Since the Non-secure world has access to the `PMCR` register, it can configure the PMU to increment counters at any exception level and in both Secure and Non-secure state. Thus, it attempts to leak timing information from the Secure world.

Shown below is an example of such a configuration:

- `PMEVTYPER0_EL0` and `PMCCFILTR_EL0`:
  - Set `P` to `0`.
  - Set `NSK` to `1`.
  - Set `M` to `0`.
  - Set `NSH` to `0`.
  - Set `SH` to `1`.
- `PMCNTENSET_EL0`:
  - Set `P[0]` to `1`.
  - Set `C` to `1`.
- `PMCR_EL0`:
  - Set `DP` to `0`.
  - Set `E` to `1`.

This configuration instructs `PMEVCNTR0_EL0` and `PMCCNTR_EL0` to increment at Secure EL1, Secure EL2 (if implemented) and EL3.

Since the Non-secure world has fine-grained control over where (at which exception levels) it instructs counters to increment, obtaining event counts would allow it to carry out side-channel timing attacks against the Secure world. Examples include Spectre, Meltdown, as well as extracting secrets from cryptographic algorithms with data-dependent variations in their execution time.

Secure world mitigation strategies

The `MDCR_EL3` register allows EL3 to configure the PMU (among other things). The Arm ARM details all of the bit fields in this register, but for the PMU there are two bits which determine the permissions of the counters:

- `SPME` for the programmable counters.
- `SCCD` for the cycle counter.

Depending on the implemented features, the Secure world can prohibit counting in AArch64 state via the following:

- ARMv8.2-Debug not implemented:

- Prohibit general event counters and the cycle counter: `MDCR_EL3.SPME == 0 && PMCR_EL0.DP == 1 && !ExternalSecureNoninvasiveDebugEnabled()`.
  - `MDCR_EL3.SPME` resets to `0`, so by default general events should not be counted in the Secure world.
  - The `PMCR_EL0.DP` bit therefore needs to be set to `1` when EL3 is entered and `PMCR_EL0` needs to be saved and restored in EL3.
  - `ExternalSecureNoninvasiveDebugEnabled()` is an authentication interface which is implementation-defined unless ARMv8.4-Debug is implemented. The Arm ARM has detailed information on this topic.
- The only other way is to disable the `PMCR_EL0.E` bit upon entering EL3, which disables counting altogether.

- ARMv8.2-Debug implemented:
  - Prohibit general event counters: `MDCR_EL3.SPME == 0`.
  - Prohibit cycle counter: `MDCR_EL3.SPME == 0 && PMCR_EL0.DP == 1`. `PMCR_EL0` therefore needs to be saved and restored in EL3.
- ARMv8.5-PMU implemented:
  - Prohibit general event counters: as in ARMv8.2-Debug.
  - Prohibit cycle counter: `MDCR_EL3.SCCD == 1`

In Aarch32 execution state the `MDCR_EL3` alias is the `SDCR` register, which has some of the bit fields of `MDCR_EL3`, most importantly the `SPME` and `SCCD` bits.

3.10.2 Build options

Several build options can be used to check for security issues. Refer to the *Build Options* for detailed information on these.

- The `BRANCH_PROTECTION` build flag can be used to enable Pointer Authentication and Branch Target Identification.
- The `ENABLE_STACK_PROTECTOR` build flag can be used to identify buffer overflows.
- The `W` build flag can be used to enable a number of compiler warning options to detect potentially incorrect code. TF-A is tested with `W=0` but it is recommended to develop against `W=2` (which will eventually become the default).

Additional guidelines are provided below for some security-related build options:

- The `ENABLE_CONSOLE_GETC` build flag should be set to 0 to disable the *getc()* feature, which allows the firmware to read characters from the console. Keeping this feature enabled is considered dangerous from a security point of view because it potentially allows an attacker to inject arbitrary data into the firmware. It should only be enabled on a need basis if there is a use case for it, for example in a testing or factory environment.

References

- Arm ARM

CHAPTER

FOUR

COMPONENTS

4.1 Secure Payload Dispatcher (SPD)

4.1.1 OP-TEE Dispatcher

OP-TEE OS is a Trusted OS running as Secure EL1.

To build and execute OP-TEE follow the instructions at OP-TEE build.git

There are two different modes for loading the OP-TEE OS. The default mode will load it as the BL32 payload during boot, and is the recommended technique for platforms to use. There is also another technique that will load OP-TEE OS after boot via an SMC call by enabling the option for OPTEE_ALLOW_SMC_LOAD that was specifically added for ChromeOS. Loading OP-TEE via an SMC call may be insecure depending upon the platform configuration. If using that option, be sure to understand the risks involved with allowing the Trusted OS to be loaded this way. ChromeOS uses a boot flow where it verifies the signature of the firmware before executing it, and then only if the signature is valid will the 'secrets' used by the TEE become accessible. The firmware then verifies the signature of the kernel using depthcharge, and the kernel verifies the rootfs using dm-verity. The SMC call to load OP-TEE is then invoked immediately after the kernel finishes loading and before any attack vectors can be opened up by mounting writable filesystems or opening network/device connections. this ensures the platform is 'closed' and running signed code through the point where OP-TEE is loaded.

4.1.2 Trusted Little Kernel (TLK) Dispatcher

TLK dispatcher (TLK-D) adds support for NVIDIA's Trusted Little Kernel (TLK) to work with Trusted Firmware-A (TF-A). TLK-D can be compiled by including it in the platform's makefile. TLK is primarily meant to work with Tegra SoCs, so while TF-A only supports TLK on Tegra, the dispatcher code can only be compiled for other platforms.

In order to compile TLK-D, we need a BL32 image to be present. Since, TLKD just needs to compile, any BL32 image would do. To use TLK as the BL32, please refer to the "Build TLK" section.

Once a BL32 is ready, TLKD can be included in the image by adding "SPD=tlkd" to the build command.

Trusted Little Kernel (TLK)

TLK is a Trusted OS running as Secure EL1. It is a Free Open Source Software (FOSS) release of the NVIDIA® Trusted Little Kernel (TLK) technology, which extends technology made available with the development of the Little Kernel (LK). You can download the LK modular embedded preemptive kernel for use on Arm, x86, and AVR32 systems from https://github.com/travisg/lk

NVIDIA implemented its Trusted Little Kernel (TLK) technology, designed as a free and open-source trusted execution environment (OTE).

TLK features include:

- Small, pre-emptive kernel
- Supports multi-threading, IPCs, and thread scheduling
- Added TrustZone features
- Added Secure Storage
- Under MIT/FreeBSD license

NVIDIA extensions to Little Kernel (LK) include:

- User mode
- Address-space separation for TAs
- TLK Client Application (CA) library
- TLK TA library
- Crypto library (encrypt/decrypt, key handling) via OpenSSL
- Linux kernel driver
- Cortex A9/A15 support
- Power Management
- TrustZone memory carve-out (reconfigurable)
- Page table management
- Debugging support over UART (USB planned)

TLK is hosted by NVIDIA on http://nv-tegra.nvidia.com under the 3rdparty/ote_partner/tlk.git repository. Detailed information about TLK and OTE can be found in the Tegra_BSP_for_Android_TLK_FOSS_Reference.pdf manual located under the "documentation" directory_.

Build TLK

To build and execute TLK, follow the instructions from "Building a TLK Device" section from Tegra_BSP_for_Android_TLK_FOSS_Reference.pdf manual.

Input parameters to TLK

TLK expects the TZDRAM size and a structure containing the boot arguments. BL2 passes this information to the EL3 software as members of the bl32_ep_info struct, where bl32_ep_info is part of bl31_params_t (passed by BL2 in X0)

Example

```
bl32_ep_info->args.arg0 = TZDRAM size available for BL32
bl32_ep_info->args.arg1 = unused (used only on Armv7-A)
bl32_ep_info->args.arg2 = pointer to boot args
```

4.1.3 Trusty Dispatcher

Trusty is a a set of software components, supporting a Trusted Execution Environment (TEE) on mobile devices, published and maintained by Google.

Detailed information and build instructions can be found on the Android Open Source Project (AOSP) webpage for Trusty hosted at https://source.android.com/security/trusty

Boot parameters

Custom boot parameters can be passed to Trusty by providing a platform specific function:

```
void plat_trusty_set_boot_args(aapcs64_params_t *args)
```

If this function is provided `args->arg0` must be set to the memory size allocated to trusty. If the platform does not provide this function, but defines `TSP_SEC_MEM_SIZE`, a default implementation will pass the memory size from `TSP_SEC_MEM_SIZE`. `args->arg1` can be set to a platform specific parameter block, and `args->arg2` should then be set to the size of that block.

Supported platforms

Out of all the platforms supported by Trusted Firmware-A, Trusty is only verified and supported by NVIDIA's Tegra SoCs.

4.1.4 ProvenCore Dispatcher

ProvenCore dispatcher (PnC-D) adds support for ProvenRun's ProvenCore micro-kernel to work with Trusted Firmware-A (TF-A).

ProvenCore is a secure OS developed by ProvenRun S.A.S. using deductive formal methods.

Once a BL32 is ready, PnC-D can be included in the image by adding "SPD=pncd" to the build command.

4.2 Activity Monitors

FEAT_AMUv1 of the Armv8-A architecture introduces the Activity Monitors extension. This extension describes the architecture for the Activity Monitor Unit (*AMU*), an optional non-invasive component for monitoring core events through a set of 64-bit counters.

When the `ENABLE_FEAT_AMU` build option is enabled, Trusted Firmware-A sets up the *AMU* prior to its exit from EL3, and will save and restore architected *AMU* counters upon suspend and resume. It will also save and restore counters 2 and 3 (INST_RETIRED and STALL_BACKEND_MEM) when switching worlds to prevent leaking potentially sensitive information. Counters 0 and 1 (CPU_CYCLES and CNT_CYCLES) always count to enable accurate performance modelling of the OS.

4.2.1 Auxiliary counters

FEAT_AMUv1 describes a set of implementation-defined auxiliary counters (also known as group 1 counters), controlled by the `ENABLE_AMU_AUXILIARY_COUNTERS` build option.

As a security precaution, Trusted Firmware-A does not enable these by default. Instead, platforms must configure their auxiliary counters through the `plat_amu_aux_enables` platform hook. This is a per-core array indexed with `plat_my_core_pos()`. A core's value will be written verbatim into `AMCNTENSET1_EL0`.

4.3 Arm SiP Services

This document enumerates and describes the Arm SiP (Silicon Provider) services.

SiP services are non-standard, platform-specific services offered by the silicon implementer or platform provider. They are accessed via `SMC` ("SMC calls") instruction executed from Exception Levels below EL3. SMC calls for SiP services:

- Follow SMC Calling Convention;
- Use SMC function IDs that fall in the SiP range, which are `0xc2000000` - `0xc200ffff` for 64-bit calls, and `0x82000000` - `0x8200ffff` for 32-bit calls.

The Arm SiP implementation offers the following services:

- Execution State Switching service

Source definitions for Arm SiP service are located in the `arm_sip_svc.h` header file.

ARM_SIP_SVC_VERSION_MAJOR	ARM_SIP_SVC_VERSION_MINOR	Changes
1	0	Move DebugFS and PMF to the new vendor specific FID range. The old FID range for these services are deprecated

Table 1: Showing different versions of arm-sip-service and changes done with each version

4.3.1 Execution State Switching service

Execution State Switching service provides a mechanism for a non-secure lower Exception Level (either EL2, or NS EL1 if EL2 isn't implemented) to request to switch its execution state (a.k.a. Register Width), either from AArch64 to AArch32, or from AArch32 to AArch64, for the calling CPU. This service is only available when Trusted Firmware-A (TF-A) is built for AArch64 (i.e. when build option `ARCH` is set to `aarch64`).

ARM_SIP_SVC_EXE_STATE_SWITCH

```
Arguments:
    uint32_t Function ID
    uint32_t PC hi
    uint32_t PC lo
    uint32_t Cookie hi
    uint32_t Cookie lo

Return:
    uint32_t
```

The function ID parameter must be `0x82000020`. It uniquely identifies the Execution State Switching service being requested.

The parameters *PC hi* and *PC lo* defines upper and lower words, respectively, of the entry point (physical address) at which execution should start, after Execution State has been switched. When calling from AArch64, *PC hi* must be 0.

When execution starts at the supplied entry point after Execution State has been switched, the parameters *Cookie hi* and *Cookie lo* are passed in CPU registers 0 and 1, respectively. When calling from AArch64, *Cookie hi* must be 0.

This call can only be made on the primary CPU, before any secondaries were brought up with `CPU_ON` PSCI call. Otherwise, the call will always fail.

The effect of switching execution state is as if the Exception Level were entered for the first time, following power on. This means CPU registers that have a defined reset value by the Architecture will assume that value. Other registers should not be expected to hold their values before the call was made. CPU endianness, however, is preserved from the previous execution state. Note that this switches the execution state of the calling CPU only. This is not a substitute for PSCI `SYSTEM_RESET`.

The service may return the following error codes:

- `STATE_SW_E_PARAM`: If any of the parameters were deemed invalid for a specific request.
- `STATE_SW_E_DENIED`: If the call is not successful, or when TF-A is built for AArch32.

If the call is successful, the caller wouldn't observe the SMC returning. Instead, execution starts at the supplied entry point, with the CPU registers 0 and 1 populated with the supplied *Cookie hi* and *Cookie lo* values, respectively.

4.4 Debug FS

Contents

- *Debug FS*
  - *Overview*
  - *Virtual filesystem*
    - *Namespace*
    - *9p interface*
  - *SMC interface*
  - *Security considerations*
  - *Limitations*
  - *Applications*

4.4.1 Overview

The *DebugFS* feature is primarily aimed at exposing firmware debug data to higher SW layers such as a non-secure component. Such component can be the TFTF test payload or a Linux kernel module.

4.4.2 Virtual filesystem

The core functionality lies in a virtual file system based on a 9p file server interface (Notes on the Plan 9 Kernel Source and Linux 9p remote filesystem protocol). The implementation permits exposing virtual files, firmware drivers, and file blobs.

Namespace

Two namespaces are exposed:

- # is used as root for drivers (e.g. #t0 is the first uart)
- / is used as root for virtual "files" (e.g. /fip, or /dev/uart)

9p interface

The associated primitives are:

- Unix-like:
  - open(): create a file descriptor that acts as a handle to the file passed as an argument.
  - close(): close the file descriptor created by open().
  - read(): read from a file to a buffer.
  - write(): write from a buffer to a file.
  - seek(): set the file position indicator of a file descriptor either to a relative or an absolute offset.
  - stat(): get information about a file (type, mode, size, …).

```
int open(const char *name, int flags);
int close(int fd);
int read(int fd, void *buf, int n);
int write(int fd, void *buf, int n);
int seek(int fd, long off, int whence);
int stat(char *path, dir_t *dir);
```

- Specific primitives :
  - mount(): create a link between a driver and spec.
  - create(): create a file in a specific location.
  - bind(): expose the content of a directory to another directory.

```
int mount(char *srv, char *mnt, char *spec);
int create(const char *name, int flags);
int bind(char *path, char *where);
```

This interface is embedded into the BL31 run-time payload when selected by build options. The interface multiplexes drivers or emulated "files":

- Debug data can be partitioned into different virtual files e.g. expose PMF measurements through a file, and internal firmware state counters through another file.
- This permits direct access to a firmware driver, mainly for test purposes (e.g. a hardware device that may not be accessible to non-privileged/ non-secure layers, or for which no support exists in the NS side).

4.4.3 SMC interface

The communication with the 9p layer in BL31 is made through an SMC conduit (SMC Calling Convention), using a specific SiP Function Id. An NS shared buffer is used to pass path string parameters, or e.g. to exchange data on a read operation. Refer to *ARM SiP Services* for a description of the SMC interface.

4.4.4 Security considerations

- Due to the nature of the exposed data, the feature is considered experimental and importantly **shall only be used in debug builds**.
- Several primitive imply string manipulations and usage of string formats.
- Special care is taken with the shared buffer to avoid TOCTOU attacks.

4.4.5 Limitations

- In order to setup the shared buffer, the component consuming the interface needs to allocate a physical page frame and transmit its address.
- In order to map the shared buffer, BL31 requires enabling the dynamic xlat table option.
- Data exchange is limited by the shared buffer length. A large read operation might be split into multiple read operations of smaller chunks.
- On concurrent access, a spinlock is implemented in the BL31 service to protect the internal work buffer, and re-entrancy into the filesystem layers.
- Notice, a physical device driver if exposed by the firmware may conflict with the higher level OS if the latter implements its own driver for the same physical device.

4.4.6 Applications

The SMC interface is accessible from an NS environment, that is:

- a test payload, bootloader or hypervisor running at NS-EL2
- a Linux kernel driver running at NS-EL1
- a Linux userspace application through the kernel driver

4.5 Exception Handling Framework

This document describes various aspects of handling exceptions by Runtime Firmware (BL31) that are targeted at EL3, other than SMCs. The *EHF* takes care of the following exceptions when targeted at EL3:

- Interrupts
- Synchronous External Aborts
- Asynchronous External Aborts

TF-A's handling of synchronous `SMC` exceptions raised from lower ELs is described in the *Firmware Design document*. However, the *EHF* changes the semantics of *Interrupt handling* and *synchronous exceptions* other than SMCs.

The *EHF* is selected by setting the build option `EL3_EXCEPTION_HANDLING` to `1`, and is only available for AArch64 systems.

4.5.1 Introduction

Through various control bits in the `SCR_EL3` register, the Arm architecture allows for asynchronous exceptions to be routed to EL3. As described in the *Interrupt Management Framework* document, depending on the chosen interrupt routing model, TF-A appropriately sets the `FIQ` and `IRQ` bits of `SCR_EL3` register to effect this routing. For most use cases, other than for the purpose of facilitating context switch between Normal and Secure worlds, FIQs and IRQs routed to EL3 are not required to be handled in EL3.

However, the evolving system and standards landscape demands that various exceptions are targeted at and handled in EL3. For instance:

- Starting with ARMv8.2 architecture extension, many RAS features have been introduced to the Arm architecture. With RAS features implemented, various components of the system may use one of the asynchronous exceptions to signal error conditions to PEs. These error conditions are of critical nature, and it's imperative that corrective or remedial actions are taken at the earliest opportunity. Therefore, a *Firmware-first Handling* approach is generally followed in response to RAS events in the system.
- The Arm SDEI specification defines interfaces through which Normal world interacts with the Runtime Firmware in order to request notification of system events. The *SDEI* specification requires that these events are notified even when the Normal world executes with the exceptions masked. This too implies

that firmware-first handling is required, where the events are first received by the EL3 firmware, and then dispatched to Normal world through purely software mechanism.

For *TF-A*, firmware-first handling means that asynchronous exceptions are suitably routed to EL3, and the Runtime Firmware (BL31) is extended to include software components that are capable of handling those exceptions that target EL3. These components—referred to as *dispatchers*[1] in general—may choose to:

- Receive and handle exceptions entirely in EL3, meaning the exceptions handling terminates in EL3.
- Receive exceptions, but handle part of the exception in EL3, and delegate the rest of the handling to a dedicated software stack running at lower Secure ELs. In this scheme, the handling spans various secure ELs.
- Receive exceptions, but handle part of the exception in EL3, and delegate processing of the error to dedicated software stack running at lower secure ELs (as above); additionally, the Normal world may also be required to participate in the handling, or be notified of such events (for example, as an *SDEI* event). In this scheme, exception handling potentially and maximally spans all ELs in both Secure and Normal worlds.

On any given system, all of the above handling models may be employed independently depending on platform choice and the nature of the exception received.

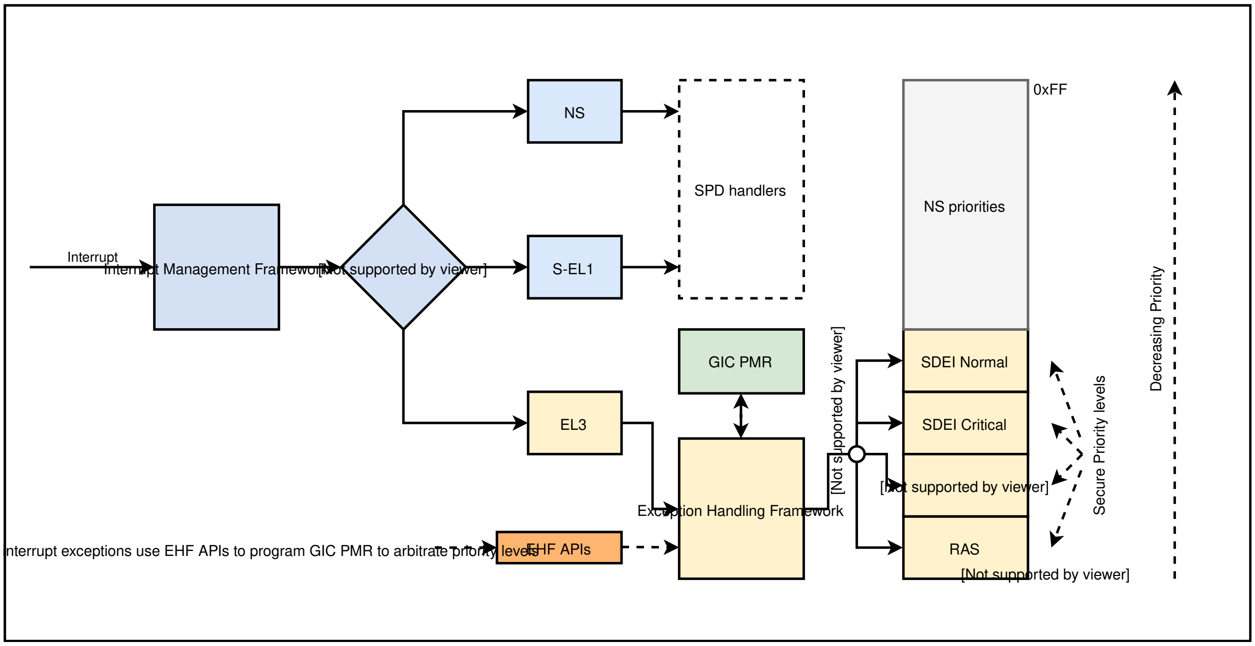
4.5.2 The role of Exception Handling Framework

Corollary to the use cases cited above, the primary role of the *EHF* is to facilitate firmware-first handling of exceptions on Arm systems. The *EHF* thus enables multiple exception dispatchers in runtime firmware to co-exist, register for, and handle exceptions targeted at EL3. This section outlines the basics, and the rest of this document expands the various aspects of the *EHF*.

In order to arbitrate exception handling among dispatchers, the *EHF* operation is based on a priority scheme. This priority scheme is closely tied to how the Arm GIC architecture defines it, although it's applied to non-interrupt exceptions too (SErrors, for example).

The platform is required to *partition* the Secure priority space into priority levels as applicable for the Secure software stack. It then assigns the dispatchers to one or more priority levels. The dispatchers then register handlers for the priority levels at runtime. A dispatcher can register handlers for more than one priority level.

[1] Not to be confused with *Secure Payload Dispatcher*, which is an EL3 component that operates in EL3 on behalf of Secure OS.



A priority level is *active* when a handler at that priority level is currently executing in EL3, or has delegated the execution to a lower EL. For interrupts, this is implicit when an interrupt is targeted and acknowledged at EL3, and the priority of the acknowledged interrupt is used to match its registered handler. The priority level is likewise implicitly deactivated when the interrupt handling concludes by EOIing the interrupt.

Non-interrupt exceptions (SErrors, for example) don't have a notion of priority. In order for the priority arbitration to work, the *EHF* provides APIs in order for these non-interrupt exceptions to assume a priority, and to interwork with interrupts. Dispatchers handling such exceptions must therefore explicitly activate and deactivate the respective priority level as and when they're handled or delegated.

Because priority activation and deactivation for interrupt handling is implicit and involves GIC priority masking, it's impossible for a lower priority interrupt to preempt a higher priority one. By extension, this means that a lower priority dispatcher cannot preempt a higher-priority one. Priority activation and deactivation for non-interrupt exceptions, however, has to be explicit. The *EHF* therefore disallows for lower priority level to be activated whilst a higher priority level is active, and would result in a panic. Likewise, a panic would result if it's attempted to deactivate a lower priority level when a higher priority level is active.

In essence, priority level activation and deactivation conceptually works like a stack—priority levels stack up in strictly increasing fashion, and need to be unstacked in strictly the reverse order. For interrupts, the GIC ensures this is the case; for non-interrupts, the *EHF* monitors and asserts this. See *Transition of priority levels*.

4.5.3 Interrupt handling

The *EHF* is a client of *Interrupt Management Framework*, and registers the top-level handler for interrupts that target EL3, as described in the *Interrupt Management Framework* document. This has the following implications:

- On GICv3 systems, when executing in S-EL1, pending Non-secure interrupts of sufficient priority are signalled as FIQs, and therefore will be routed to EL3. As a result, S-EL1 software cannot expect to handle Non-secure interrupts at S-EL1. Essentially, this deprecates the routing mode described as

CSS=0, TEL3=0.

In order for S-EL1 software to handle Non-secure interrupts while having *EHF* enabled, the dispatcher must adopt a model where Non-secure interrupts are received at EL3, but are then *synchronously* handled over to S-EL1.

- On GICv2 systems, it's required that the build option `GICV2_G0_FOR_EL3` is set to `1` so that *Group 0* interrupts target EL3.
- While executing in Secure world, *EHF* sets GIC Priority Mask Register to the lowest Secure priority. This means that no Non-secure interrupts can preempt Secure execution. See *Effect on SMC calls* for more details.

As mentioned above, with *EHF*, the platform is required to partition *Group 0* interrupts into distinct priority levels. A dispatcher that chooses to receive interrupts can then *own* one or more priority levels, and register interrupt handlers for them. A given priority level can be assigned to only one handler. A dispatcher may register more than one priority level.

Dispatchers are assigned interrupt priority levels in two steps:

Partitioning priority levels

Interrupts are associated to dispatchers by way of grouping and assigning interrupts to a priority level. In other words, all interrupts that are to target a particular dispatcher should fall in a particular priority level. For priority assignment:

- Of the 8 bits of priority that Arm GIC architecture permits, bit 7 must be 0 (secure space).
- Depending on the number of dispatchers to support, the platform must choose to use the top *n* of the 7 remaining bits to identify and assign interrupts to individual dispatchers. Choosing *n* bits supports up to 2^n distinct dispatchers. For example, by choosing 2 additional bits (i.e., bits 6 and 5), the platform can partition into 4 secure priority ranges: `0x0`, `0x20`, `0x40`, and `0x60`. See *Interrupt handling example*.

Note: The Arm GIC architecture requires that a GIC implementation that supports two security states must implement at least 32 priority levels; i.e., at least 5 upper bits of the 8 bits are writeable. In the scheme described above, when choosing *n* bits for priority range assignment, the platform must ensure that at least `n+1` top bits of GIC priority are writeable.

The priority thus assigned to an interrupt is also used to determine the priority of delegated execution in lower ELs. Delegated execution in lower EL is associated with a priority level chosen with `ehf_activate_priority()` API (described *later*). The chosen priority level also determines the interrupts masked while executing in a lower EL, therefore controls preemption of delegated execution.

The platform expresses the chosen priority levels by declaring an array of priority level descriptors. Each entry in the array is of type `ehf_pri_desc_t`, and declares a priority level, and shall be populated by the `EHF_PRI_DESC()` macro.

Warning: The macro `EHF_PRI_DESC()` installs the descriptors in the array at a computed index, and not necessarily where the macro is placed in the array. The size of the array might therefore be larger than what it appears to be. The `ARRAY_SIZE()` macro therefore should be used to determine the size of array.

Finally, this array of descriptors is exposed to *EHF* via the `EHF_REGISTER_PRIORITIES()` macro.

Refer to the *Interrupt handling example* for usage. See also: *Interrupt Prioritisation Considerations*.

Programming priority

The text in *Partitioning priority levels* only describes how the platform expresses the required levels of priority. It however doesn't choose interrupts nor program the required priority in GIC.

The *Firmware Design guide* explains methods for configuring secure interrupts. *EHF* requires the platform to enumerate interrupt properties (as opposed to just numbers) of Secure interrupts. The priority of secure interrupts must match that as determined in the *Partitioning priority levels* section above.

See *Limitations*, and also refer to *Interrupt handling example* for illustration.

4.5.4 Registering handler

Dispatchers register handlers for their priority levels through the following API:

```
int ehf_register_priority_handler(int pri, ehf_handler_t handler)
```

The API takes two arguments:

- The priority level for which the handler is being registered;
- The handler to be registered. The handler must be aligned to 4 bytes.

If a dispatcher owns more than one priority levels, it has to call the API for each of them.

The API will succeed, and return `0`, only if:

- There exists a descriptor with the priority level requested.
- There are no handlers already registered by a previous call to the API.

Otherwise, the API returns `-1`.

The interrupt handler should have the following signature:

```
typedef int (*ehf_handler_t)(uint32_t intr_raw, uint32_t flags, void *handle,
                void *cookie);
```

The parameters are as obtained from the top-level *EL3 interrupt handler*.

The *SDEI dispatcher*, for example, expects the platform to allocate two different priority levels—`PLAT_SDEI_CRITICAL_PRI`, and `PLAT_SDEI_NORMAL_PRI` —and registers the same handler to handle both levels.

4.5.5 Interrupt handling example

The following annotated snippet demonstrates how a platform might choose to assign interrupts to fictitious dispatchers:

```
#include <common/interrupt_props.h>
#include <drivers/arm/gic_common.h>
#include <exception_mgmt.h>

...

/*
 * This platform uses 2 bits for interrupt association. In total, 3 upper
 * bits are in use.
 *
 *  7 6 5   3      0
 * .-.-.-.----------.
 * |0|b|b|  ..0..   |
 * '-'-'-'----------'
 */
#define PLAT_PRI_BITS        2

/* Priorities for individual dispatchers */
#define DISP0_PRIO           0x00 /* Not used */
#define DISP1_PRIO           0x20
#define DISP2_PRIO           0x40
#define DISP3_PRIO           0x60

/* Install priority level descriptors for each dispatcher */
ehf_pri_desc_t plat_exceptions[] = {
     EHF_PRI_DESC(PLAT_PRI_BITS, DISP1_PRIO),
     EHF_PRI_DESC(PLAT_PRI_BITS, DISP2_PRIO),
     EHF_PRI_DESC(PLAT_PRI_BITS, DISP3_PRIO),
};

/* Expose priority descriptors to Exception Handling Framework */
EHF_REGISTER_PRIORITIES(plat_exceptions, ARRAY_SIZE(plat_exceptions),
     PLAT_PRI_BITS);

...

/* List interrupt properties for GIC driver. All interrupts target EL3 */
const interrupt_prop_t plat_interrupts[] = {
     /* Dispatcher 1 owns interrupts d1_0 and d1_1, so assigns priority DISP1_
↪PRIO */
     INTR_PROP_DESC(d1_0, DISP1_PRIO, INTR_TYPE_EL3, GIC_INTR_CFG_LEVEL),
     INTR_PROP_DESC(d1_1, DISP1_PRIO, INTR_TYPE_EL3, GIC_INTR_CFG_LEVEL),

     /* Dispatcher 2 owns interrupts d2_0 and d2_1, so assigns priority DISP2_
↪PRIO */
     INTR_PROP_DESC(d2_0, DISP2_PRIO, INTR_TYPE_EL3, GIC_INTR_CFG_LEVEL),
     INTR_PROP_DESC(d2_1, DISP2_PRIO, INTR_TYPE_EL3, GIC_INTR_CFG_LEVEL),
```

(continues on next page)

(continued from previous page)

```
    /* Dispatcher 3 owns interrupts d3_0 and d3_1, so assigns priority DISP3_
↪PRIO */
    INTR_PROP_DESC(d3_0, DISP3_PRIO, INTR_TYPE_EL3, GIC_INTR_CFG_LEVEL),
    INTR_PROP_DESC(d3_1, DISP3_PRIO, INTR_TYPE_EL3, GIC_INTR_CFG_LEVEL),
};

...

/* Dispatcher 1 registers its handler */
ehf_register_priority_handler(DISP1_PRIO, disp1_handler);

/* Dispatcher 2 registers its handler */
ehf_register_priority_handler(DISP2_PRIO, disp2_handler);

/* Dispatcher 3 registers its handler */
ehf_register_priority_handler(DISP3_PRIO, disp3_handler);

...
```

See also the *Build-time flow* and the *Run-time flow*.

4.5.6 Activating and Deactivating priorities

A priority level is said to be *active* when an exception of that priority is being handled: for interrupts, this is implied when the interrupt is acknowledged; for non-interrupt exceptions, such as SErrors or *SDEI explicit dispatches*, this has to be done via calling `ehf_activate_priority()`. See *Run-time flow*.

Conversely, when the dispatcher has reached a logical resolution for the cause of the exception, the corresponding priority level ought to be deactivated. As above, for interrupts, this is implied when the interrupt is EOId in the GIC; for other exceptions, this has to be done via calling `ehf_deactivate_priority()`.

Thanks to *different provisions* for exception delegation, there are potentially more than one work flow for deactivation:

- The dispatcher has addressed the cause of the exception, and decided to take no further action. In this case, the dispatcher's handler deactivates the priority level before returning to the *EHF*. Runtime firmware, upon exit through an `ERET`, resumes execution before the interrupt occurred.
- The dispatcher has to delegate the execution to lower ELs, and the cause of the exception can be considered resolved only when the lower EL returns signals complete (via an `SMC`) at a future point in time. The following sequence ensues:
  1. The dispatcher calls `setjmp()` to setup a jump point, and arranges to enter a lower EL upon the next `ERET`.
  2. Through the ensuing `ERET` from runtime firmware, execution is delegated to a lower EL.
  3. The lower EL completes its execution, and signals completion via an `SMC`.
  4. The `SMC` is handled by the same dispatcher that handled the exception previously. Noticing the conclusion of exception handling, the dispatcher does `longjmp()` to resume beyond the previous jump point.

As mentioned above, the *EHF* provides the following APIs for activating and deactivating interrupt:

- `ehf_activate_priority()` activates the supplied priority level, but only if the current active priority is higher than the given one; otherwise panics. Also, to prevent interruption by physical interrupts of lower priority, the *EHF* programs the *Priority Mask Register* corresponding to the PE to the priority being activated. Dispatchers typically only need to call this when handling exceptions other than interrupts, and it needs to delegate execution to a lower EL at a desired priority level.
- `ehf_deactivate_priority()` deactivates a given priority, but only if the current active priority is equal to the given one; otherwise panics. *EHF* also restores the *Priority Mask Register* corresponding to the PE to the priority before the call to `ehf_activate_priority()`. Dispatchers typically only need to call this after handling exceptions other than interrupts.

The calling of APIs are subject to allowed *transitions*. See also the *Run-time flow*.

4.5.7 Transition of priority levels

The *EHF* APIs `ehf_activate_priority()` and `ehf_deactivate_priority()` can be called to transition the current priority level on a PE. A given sequence of calls to these APIs are subject to the following conditions:

- For activation, the *EHF* only allows for the priority to increase (i.e. numeric value decreases);
- For deactivation, the *EHF* only allows for the priority to decrease (i.e. numeric value increases). Additionally, the priority being deactivated is required to be the current priority.

If these are violated, a panic will result.

4.5.8 Effect on SMC calls

In general, Secure execution is regarded as more important than Non-secure execution. As discussed elsewhere in this document, EL3 execution, and any delegated execution thereafter, has the effect of raising GIC's priority mask—either implicitly by acknowledging Secure interrupts, or when dispatchers call `ehf_activate_priority()`. As a result, Non-secure interrupts cannot preempt any Secure execution.

SMCs from Non-secure world are synchronous exceptions, and are mechanisms for Non-secure world to request Secure services. They're broadly classified as *Fast* or *Yielding* (see SMCCC).

- *Fast* SMCs are atomic from the caller's point of view. I.e., they return to the caller only when the Secure world has finished serving the request. Any Non-secure interrupts that become pending meanwhile cannot preempt Secure execution.
- *Yielding* SMCs carry the semantics of a preemptible, lower-priority request. A pending Non-secure interrupt can preempt Secure execution handling a Yielding SMC. I.e., the caller might observe a Yielding SMC returning when either:
  1. Secure world completes the request, and the caller would find `SMC_OK` as the return code.
  2. A Non-secure interrupt preempts Secure execution. Non-secure interrupt is handled, and Non-secure execution resumes after `SMC` instruction.

The dispatcher handling a Yielding SMC must provide a different return code to the Non-secure caller to distinguish the latter case. This return code, however, is not standardised (unlike `SMC_UNKNOWN` or `SMC_OK`, for example), so will vary across dispatchers that handle the request.

For the latter case above, dispatchers before *EHF* expect Non-secure interrupts to be taken to S-EL1[2], so would get a chance to populate the designated preempted error code before yielding to Non-secure world.

The introduction of *EHF* changes the behaviour as described in *Interrupt handling*.

When *EHF* is enabled, in order to allow Non-secure interrupts to preempt Yielding SMC handling, the dispatcher must call `ehf_allow_ns_preemption()` API. The API takes one argument, the error code to be returned to the Non-secure world upon getting preempted.

4.5.9 Build-time flow

Please refer to the *figure* above.

The build-time flow involves the following steps:

1. Platform assigns priorities by installing priority level descriptors for individual dispatchers, as described in *Partitioning priority levels*.
2. Platform provides interrupt properties to GIC driver, as described in *Programming priority*.
3. Dispatcher calling `ehf_register_priority_handler()` to register an interrupt handler.

Also refer to the *Interrupt handling example*.

4.5.10 Run-time flow

The following is an example flow for interrupts:

1. The GIC driver, during initialization, iterates through the platform-supplied interrupt properties (see *Programming priority*), and configures the interrupts. This programs the appropriate priority and group (Group 0) on interrupts belonging to different dispatchers.
2. The *EHF*, during its initialisation, registers a top-level interrupt handler with the *Interrupt Management Framework* for EL3 interrupts. This also results in setting the routing bits in `SCR_EL3`.
3. When an interrupt belonging to a dispatcher fires, GIC raises an EL3/Group 0 interrupt, and is taken to EL3.
4. The top-level EL3 interrupt handler executes. The handler acknowledges the interrupt, reads its *Running Priority*, and from that, determines the dispatcher handler.
5. The *EHF* programs the *Priority Mask Register* of the PE to the priority of the interrupt received.
6. The *EHF* marks that priority level *active*, and jumps to the dispatcher handler.
7. Once the dispatcher handler finishes its job, it has to immediately *deactivate* the priority level before returning to the *EHF*. See *deactivation workflows*.

The following is an example flow for exceptions that targets EL3 other than interrupt:

[2] In case of GICv2, Non-secure interrupts while in S-EL1 were signalled as IRQs, and in case of GICv3, FIQs.

1. The platform provides handlers for the specific kind of exception.
2. The exception arrives, and the corresponding handler is executed.
3. The handler calls `ehf_activate_priority()` to activate the required priority level. This also has the effect of raising GIC priority mask, thus preventing interrupts of lower priority from preempting the handling. The handler may choose to do the handling entirely in EL3 or delegate to a lower EL.
4. Once exception handling concludes, the handler calls `ehf_deactivate_priority()` to deactivate the priority level activated earlier. This also has the effect of lowering GIC priority mask to what it was before.

4.5.11 Interrupt Prioritisation Considerations

The GIC priority scheme, by design, prioritises Secure interrupts over Normal world ones. The platform further assigns relative priorities amongst Secure dispatchers through *EHF*.

As mentioned in *Partitioning priority levels*, interrupts targeting distinct dispatchers fall in distinct priority levels. Because they're routed via the GIC, interrupt delivery to the PE is subject to GIC prioritisation rules. In particular, when an interrupt is being handled by the PE (i.e., the interrupt is in *Active* state), only interrupts of higher priority are signalled to the PE, even if interrupts of same or lower priority are pending. This has the side effect of one dispatcher being starved of interrupts by virtue of another dispatcher handling its (higher priority) interrupts.

The *EHF* doesn't enforce a particular prioritisation policy, but the platform should carefully consider the assignment of priorities to dispatchers integrated into runtime firmware. The platform should sensibly delineate priority to various dispatchers according to their nature. In particular, dispatchers of critical nature (RAS, for example) should be assigned higher priority than others (*SDEI*, for example); and within *SDEI*, Critical priority *SDEI* should be assigned higher priority than Normal ones.

4.5.12 Limitations

The *EHF* has the following limitations:

- Although there could be up to 128 Secure dispatchers supported by the GIC priority scheme, the size of descriptor array exposed with `EHF_REGISTER_PRIORITIES()` macro is currently limited to 32. This serves most expected use cases. This may be expanded in the future, should use cases demand so.
- The platform must ensure that the priority assigned to the dispatcher in the exception descriptor and the programmed priority of interrupts handled by the dispatcher match. The *EHF* cannot verify that this has been followed.

4.6 Firmware Configuration Framework

This document provides an overview of the *FCONF* framework.

4.6.1 Introduction

The Firmware CONfiguration Framework (*FCONF*) is an abstraction layer for platform specific data, allowing a "property" to be queried and a value retrieved without the requesting entity knowing what backing store is being used to hold the data.

It is used to bridge new and old ways of providing platform-specific data. Today, information like the Chain of Trust is held within several, nested platform-defined tables. In the future, it may be provided as part of a device blob, along with the rest of the information about images to load. Introducing this abstraction layer will make migration easier and will preserve functionality for platforms that cannot / don't want to use device tree.

4.6.2 Accessing properties

Properties defined in the *FCONF* are grouped around namespaces and sub-namespaces: a.b.property. Examples namespace can be:

- (*TBBR*) Chain of Trust data: tbbr.cot.trusted_boot_fw_cert
- (*TBBR*) dynamic configuration info: tbbr.dyn_config.disable_auth
- Arm io policies: arm.io_policies.bl2_image
- GICv3 properties: hw_config.gicv3_config.gicr_base

Properties can be accessed with the `FCONF_GET_PROPERTY(a,b,property)` macro.

4.6.3 Defining properties

Properties composing the *FCONF* have to be stored in C structures. If properties originate from a different backend source such as a device tree, then the platform has to provide a `populate()` function which essentially captures the property and stores them into a corresponding *FCONF* based C structure.

Such a `populate()` function is usually platform specific and is associated with a specific backend source. For example, a populator function which captures the hardware topology of the platform from the HW_CONFIG device tree. Hence each `populate()` function must be registered with a specific `config_type` identifier. It broadly represents a logical grouping of configuration properties which is usually a device tree file.

Example:

- FW_CONFIG: properties related to base address, maximum size and image id of other DTBs etc.
- TB_FW: properties related to trusted firmware such as IO policies, mbedtls heap info etc.
- HW_CONFIG: properties related to hardware configuration of the SoC such as topology, GIC controller, PSCI hooks, CPU ID etc.

Hence the `populate()` callback must be registered to the (*FCONF*) framework with the `FCONF_REGISTER_POPULATOR()` macro. This ensures that the function would be called inside the generic `fconf_populate()` function during initialization.

```
int fconf_populate_topology(uintptr_t config)
{
    /* read hw config dtb and fill soc_topology struct */
}

FCONF_REGISTER_POPULATOR(HW_CONFIG, topology, fconf_populate_topology);
```

Then, a wrapper has to be provided to match the `FCONF_GET_PROPERTY()` macro:

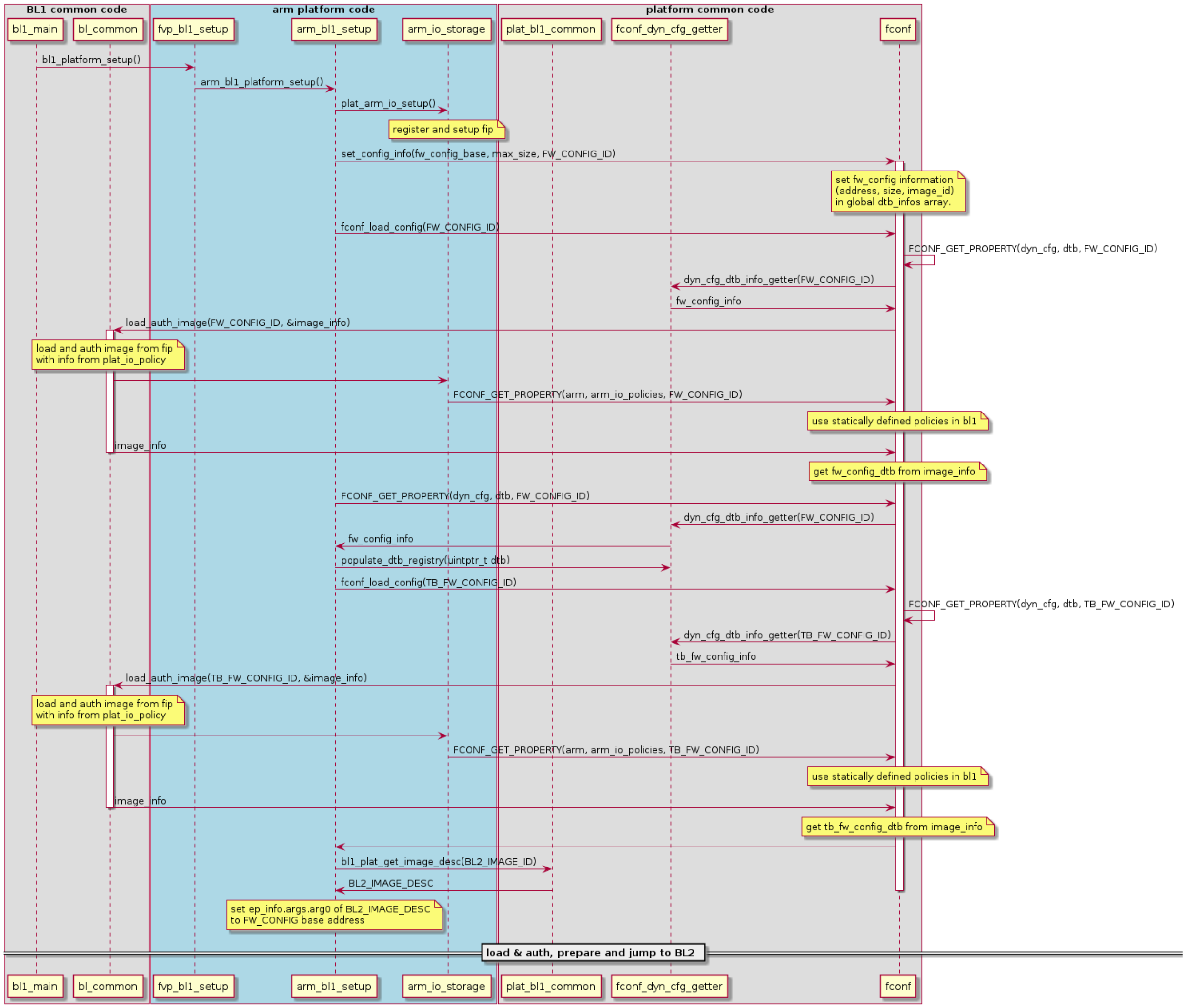
```
/* generic getter */
#define FCONF_GET_PROPERTY(a,b,property)    a##__##b##_getter(property)

/* my specific getter */
#define hw_config__topology_getter(prop) soc_topology.prop
```

This second level wrapper can be used to remap the `FCONF_GET_PROPERTY()` to anything appropriate: structure, array, function, etc..

To ensure a good interpretation of the properties, this documentation must explain how the properties are described for a specific backend. Refer to the *Properties binding information* section for more information and example.

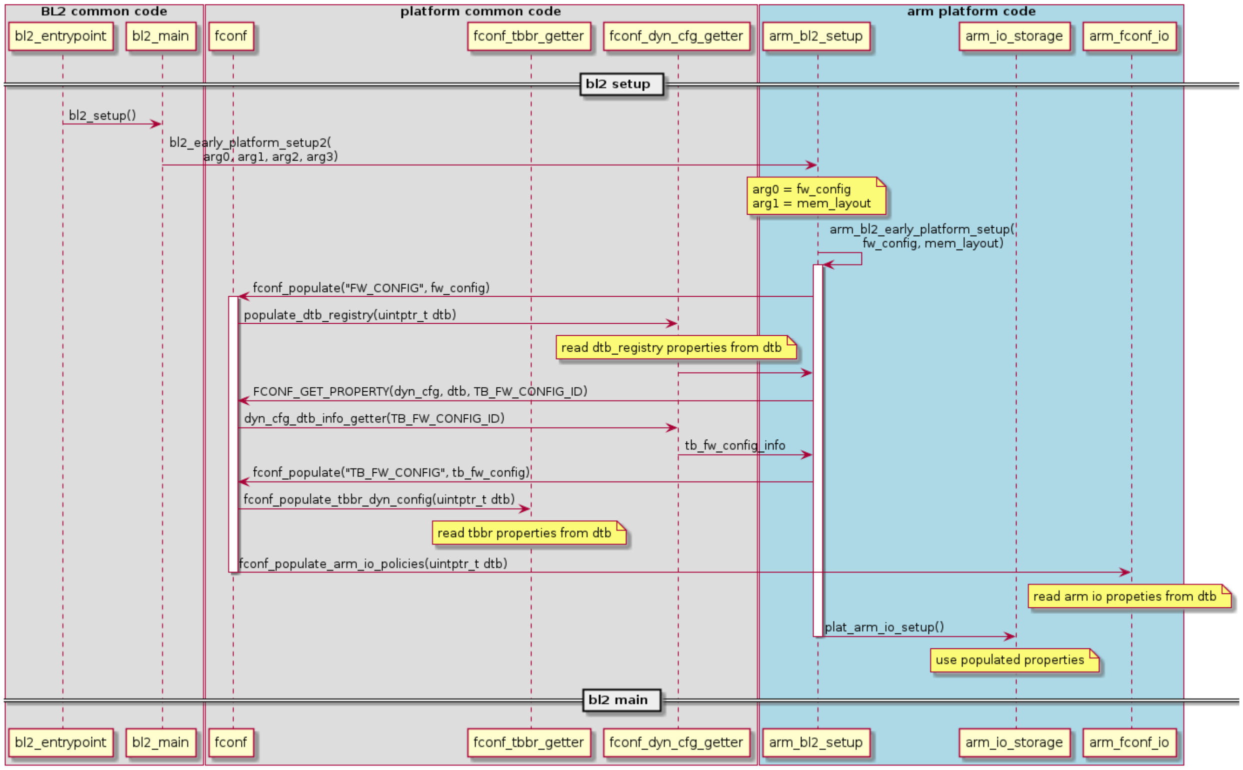
4.6.4 Loading the property device tree

The `fconf_load_config(image_id)` must be called to load fw_config and tb_fw_config devices tree containing the properties' values. This must be done after the io layer is initialized, as the *DTB* is stored on an external device (FIP).



4.6.5 Populating the properties

Once a valid device tree is available, the `fconf_populate(config)` function can be used to fill the C data structure with the data from the config *DTB*. This function will call all the `populate()` callbacks which have been registered with `FCONF_REGISTER_POPULATOR()` as described above.



4.6.6 Namespace guidance

As mentioned above, properties are logically grouped around namespaces and sub-namespaces. The following concepts should be considered when adding new properties/namespaces. The framework differentiates two types of properties:

- Properties used inside common code.
- Properties used inside platform specific code.

The first category applies to properties being part of the firmware and shared across multiple platforms. They should be globally accessible and defined inside the `lib/fconf` directory. The namespace must be chosen to reflect the feature/data abstracted.

Example:

- *TBBR* related properties: tbbr.cot.bl2_id
- Dynamic configuration information: dyn_cfg.dtb_info.hw_config_id

The second category should represent the majority of the properties defined within the framework: Platform specific properties. They must be accessed only within the platform API and are defined only inside the platform scope. The namespace must contain the platform name under which the properties defined belong.

Example:

- Arm io framework: arm.io_policies.bl31_id

4.6.7 Properties binding information

DTB binding for FCONF properties

This document describes the device tree format of *FCONF* properties. These properties are not related to a specific platform and can be queried from common code.

Dynamic configuration

The *FCONF* framework expects a *dtb-registry* node with the following field:

- **compatible [mandatory]**
  - value type: <string>
  - Must be the string "fconf,dyn_cfg-dtb_registry".

Then a list of subnodes representing a configuration *DTB*, which can be used by *FCONF*. Each subnode should be named according to the information it contains, and must be formed with the following fields:

- **load-address [mandatory]**
  - value type: <u64>
  - Physical loading base address of the configuration. If secondary-load-address is also provided (see below), then this is the primary load address.
- **max-size [mandatory]**
  - value type: <u32>
  - Maximum size of the configuration.
- **id [mandatory]**
  - value type: <u32>
  - Image ID of the configuration.
- **secondary-load-address [optional]**
  - value type: <u64>
  - A platform uses this physical address to copy the configuration to another location during the boot-flow.

Trusted Boot Firmware Configuration bindings

This document defines the nodes and properties used to define the Trusted-Boot firmware configuration. Platform owners are advised to define shared bindings here. If a binding does not generalize, they should be documented alongside platform documentation. There is no guarantee of backward compatibility with the nodes and properties outlined in this context.

IO FIP Handles

- **compatible [mandatory]**
  - value type: <string>
  - Should be the string `"<plat>,io-fip-handle"`, where `<plat>` is the name of the platform (i.e. `"arm,io-fip-handle"`).
- **scp_bl2_uuid [mandatory]**
  - value type: <string>
  - SCP Firmware SCP_BL2 UUID
- **bl31_uuid [mandatory]**
  - value type: <string>
  - EL3 Runtime Firmware BL31 UUID
- **bl32_uuid [mandatory]**
  - value type: <string>
  - Secure Payload BL32 (Trusted OS) UUID
- **bl32_extra1_uuid [mandatory]**
  - value type: <string>
  - Secure Payload BL32_EXTRA1 (Trusted OS Extra1) UUID
- **bl32_extra2_uuid [mandatory]**
  - value type: <string>
  - Secure Payload BL32_EXTRA2 (Trusted OS Extra2) UUID
- **bl33_uuid [mandatory]**
  - value type: <string>
  - Non-Trusted Firmware BL33 UUID
- **hw_cfg_uuid [mandatory]**
  - value type: <string>
  - HW_CONFIG (e.g. Kernel DT) UUID
- **soc_fw_cfg_uuid [mandatory]**

  - value type: <string>
  - SOC Firmware Configuration SOC_FW_CONFIG UUID
- **tos_fw_cfg_uuid [mandatory]**
  - value type: <string>
  - Trusted OS Firmware Configuration TOS_FW_CONFIG UUID
- **nt_fw_cfg_uuid [mandatory]**
  - value type: <string>
  - Non-Trusted Firmware Configuration NT_FW_CONFIG UUID
- **cca_cert_uuid [optional]**
  - value type: <string>
  - CCA Content Certificate UUID
- **core_swd_cert_uuid [optional]**
  - value type: <string>
  - Core SWD Key Certificate UUID
- **plat_cert_uuid [optional]**
  - value type: <string>
  - Core SWD Key Certificate UUID
- **t_key_cert_uuid [optional]**
  - value type: <string>
  - Trusted Key Certificate UUID
- **scp_fw_key_uuid [optional]**
  - value type: <string>
  - SCP Firmware Key UUID
- **soc_fw_key_uuid [optional]**
  - value type: <string>
  - SOC Firmware Key UUID
- **tos_fw_key_cert_uuid [optional]**
  - value type: <string>
  - TOS Firmware Key UUID
- **nt_fw_key_cert_uuid [optional]**
  - value type: <string>
  - Non-Trusted Firmware Key UUID

- **scp_fw_content_cert_uuid [optional]**
  - value type: <string>
  - SCP Firmware Content Certificate UUID
- **soc_fw_content_cert_uuid [optional]**
  - value type: <string>
  - SOC Firmware Content Certificate UUID
- **tos_fw_content_cert_uuid [optional]**
  - value type: <string>
  - TOS Firmware Content Certificate UUID
- **nt_fw_content_cert_uuid [optional]**
  - value type: <string>
  - Non-Trusted Firmware Content Certificate UUID
- **plat_sp_content_cert_uuid [optional]**
  - value type: <string>
  - Platform Secure Partition Content Certificate UUID

Secure Partitions

- **compatible [mandatory]**
  - value type: <string>
  - Should be the string `"<plat>,sp"`, where `<plat>` is the name of the platform (i.e. `"arm,sp"`).
- **uuid [mandatory]**
  - value type: <string>
  - A string identifying the UUID of the service implemented by this partition. The UUID format is described in RFC 4122.
- **load-address [mandatory]**
  - value type: <u32>
  - Physical base address of the partition in memory. Absence of this field indicates that the partition is position independent and can be loaded at any address chosen at boot time.
- **owner [optional]**
  - value type: <string>
  - A string property representing the name of the owner of the secure partition, which may be the silicon or platform provider.

Chain of Trust Descriptors

If a structure includes a Chain of Trust (CoT) for secure authentication and verification, it must conform to the bindings described in the *Chain of trust bindings* document. Specifically, the CoT should be represented using a `cot` node containing `manifests` and `images` sub-nodes, with certificates, images, and non-volatile counters defined as per the specifications therein. This ensures compatibility with the authentication framework and supports features such as certificate hierarchies, rollback protection, and root-of-trust key integration. For full details on required properties and node structure, refer to the *Chain of trust bindings* document.

4.7 Firmware Update (FWU)

This document describes the design of the various Firmware Update (FWU) mechanisms available in TF-A.

1. PSA Firmware Update (PSA FWU)
2. TBBR Firmware Update (TBBR FWU)

PSA Firmware Update implements the specification of the same name (Arm document IHI 0093), which defines a standard firmware interface for installing firmware updates. On the other hand, TBBR Firmware Update only covers firmware recovery. Arguably, its name is somewhat misleading but the TBBR specification and terminology predates PSA FWU. Both mechanisms are complementary in the sense that PSA FWU assumes that the device has a backup or recovery capability in the event of a failed update, which can be fulfilled with TBBR FWU implementation.

4.7.1 PSA Firmware Update (PSA FWU)

Introduction

The PSA FW update specification defines the concepts of `Firmware Update Client` and `Firmware Update Agent`. The new firmware images are provided by the `Client` to the `Update Agent` to flash them in non-volatile storage.

A common system design will place the `Update Agent` in the Secure-world while the `Client` executes in the Normal-world. The PSA FW update specification provides ABIs meant for a Normal-world entity aka `Client` to transmit the firmware images to the `Update Agent`.

Scope

The design of the `Client` and `Update Agent` is out of scope of this document. This document mainly covers `Platform Boot` details i.e. the role of the second stage Bootloader after FWU has been done by `Client` and `Update Agent`.

Overview

There are active and update banks in the non-volatile storage identified by the `active_index` and the `update_index` respectively. An active bank stores running firmware, whereas an update bank contains firmware updates.

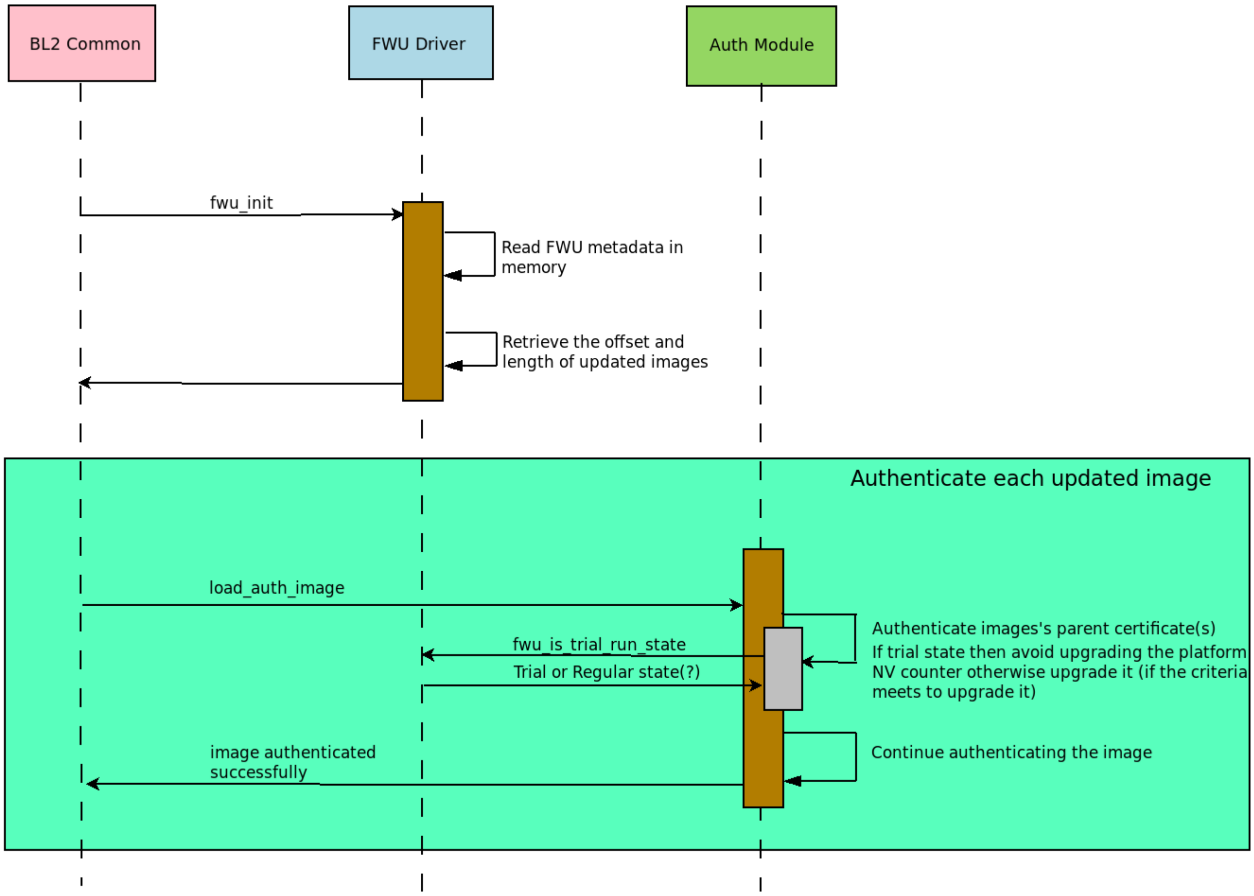
Once Firmwares are updated in the update bank of the non-volatile storage, then `Update Agent` marks the update bank as the active bank, and write updated FWU metadata in non-volatile storage. On subsequent reboot, the second stage Bootloader (BL2) performs the following actions:

- Read FWU metadata in memory
- Retrieve the image specification (offset and length) of updated images present in non-volatile storage with the help of FWU metadata
- Set these image specification in the corresponding I/O policies of the updated images using the FWU platform functions `plat_fwu_set_images_source()` and `plat_fwu_set_metadata_image_source()`, please refer *Porting Guide*
- Use these I/O policies to read the images from this address into the memory

By default, the platform uses the active bank of non-volatile storage to boot the images in `trial state`. If images pass through the authentication check and also if the system successfully booted the Normal-world image then `Update Agent` marks this update as accepted after further sanitisation checking at Normal-world.

The second stage Bootloader (BL2) avoids upgrading the platform NV-counter until it's been confirmed that given update is accepted.

The following sequence diagram shows platform-boot flow:



If the platform fails to boot from active bank due to any reasons such as authentication failure or non-fuctionality of Normal-world software then the watchdog will reset to give a chance to the platform to fix the issue. This boot failure & reset sequence might be repeated up to `trial state` times. After that, the platform can decide to boot from the `previous_active_index` bank.

If the images still does not boot successfully from the `previous_active_index` bank (e.g. due to ageing effect of non-volatile storage) then the platform can choose firmware recovery mechanism *TBBR Firmware Update (TBBR FWU)* to bring system back to life.

4.7.2 TBBR Firmware Update (TBBR FWU)

Introduction

This technique enables authenticated firmware to update firmware images from external interfaces such as USB, UART, SD-eMMC, NAND, NOR or Ethernet to SoC Non-Volatile memories such as NAND Flash, LPDDR2-NVM or any memory determined by the platform. This feature functions even when the current firmware in the system is corrupt or missing; it therefore may be used as a recovery mode. It may also be complemented by other, higher level firmware update software.

FWU implements a specific part of the Trusted Board Boot Requirements (TBBR) specification, Arm DEN0006C-1. It should be used in conjunction with the *Trusted Board Boot* design document, which describes the image authentication parts of the Trusted Firmware-A (TF-A) TBBR implementation.

It can be used as a last resort when all firmware updates that are carried out as part of the *PSA Firmware Update (PSA FWU)* procedure have failed to function.

Scope

This document describes the secure world FWU design. It is beyond its scope to describe how normal world FWU images should operate. To implement normal world FWU images, please refer to the "Non-Trusted Firmware Updater" requirements in the TBBR.

Overview

The FWU boot flow is primarily mediated by BL1. Since BL1 executes in ROM, and it is usually desirable to minimize the amount of ROM code, the design allows some parts of FWU to be implemented in other secure and normal world images. Platform code may choose which parts are implemented in which images but the general expectation is:

- BL1 handles:
  - Detection and initiation of the FWU boot flow.
  - Copying images from non-secure to secure memory
  - FWU image authentication
  - Context switching between the normal and secure world during the FWU process.
- Other secure world FWU images handle platform initialization required by the FWU process.
- Normal world FWU images handle loading of firmware images from external interfaces to non-secure memory.

The primary requirements of the FWU feature are:

1. Export a BL1 SMC interface to interoperate with other FWU images executing at other Exception Levels.
2. Export a platform interface to provide FWU common code with the information it needs, and to enable platform specific FWU functionality. See the *Porting Guide* for details of this interface.

TF-A uses abbreviated image terminology for FWU images like for other TF-A images. See the *Image Terminology* document for an explanation of these terms.

The following diagram shows the FWU boot flow for Arm development platforms. Arm CSS platforms like Juno have a System Control Processor (SCP), and these use all defined FWU images. Other platforms may use a subset of these.

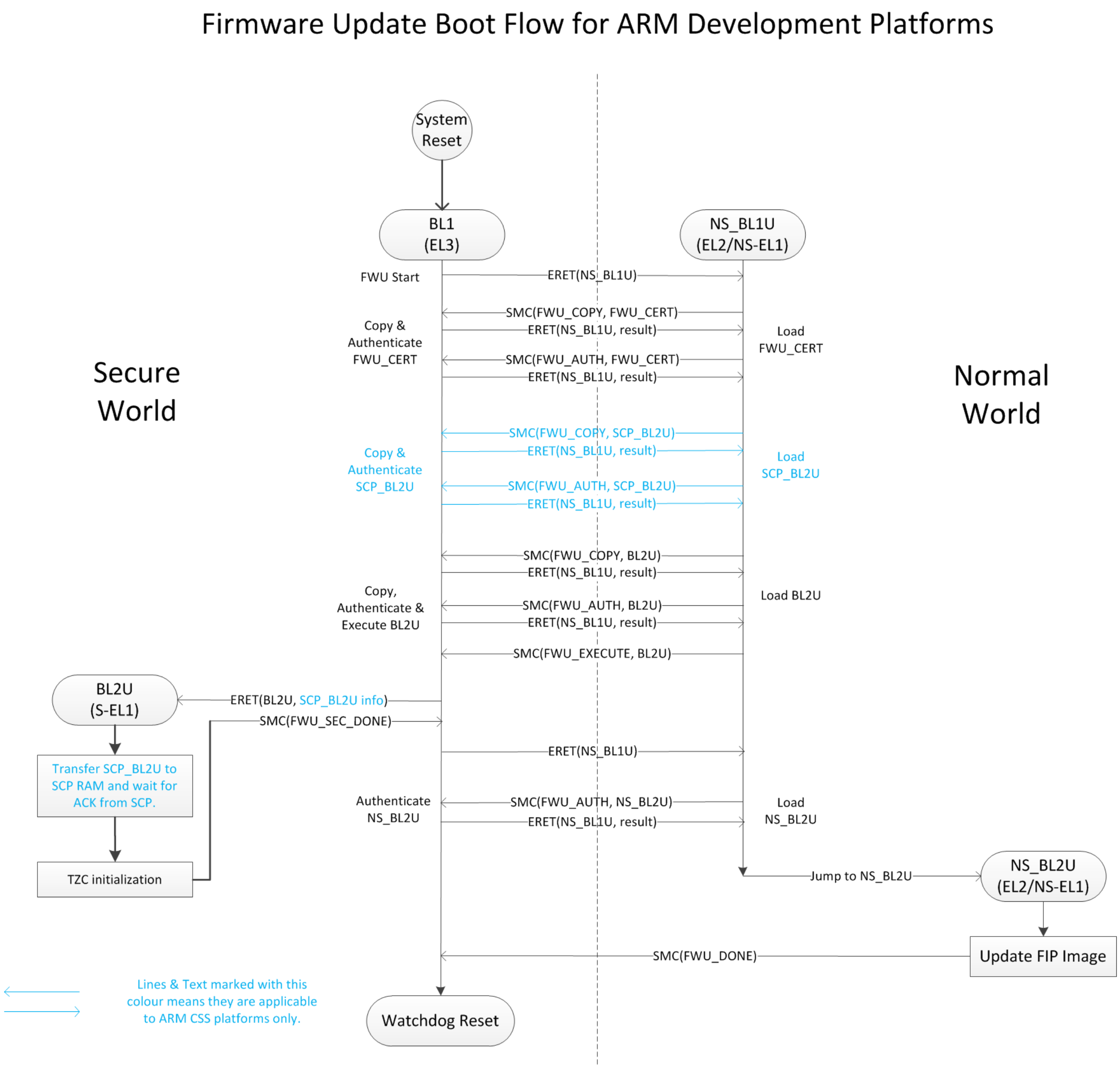


Image Identification

Each FWU image and certificate is identified by a unique ID, defined by the platform, which BL1 uses to fetch an image descriptor (`image_desc_t`) via a call to `bl1_plat_get_image_desc()`. The same ID is also used to prepare the Chain of Trust (Refer to the *Authentication Framework & Chain of Trust* document for more information).

The image descriptor includes the following information:

- Executable or non-executable image. This indicates whether the normal world is permitted to request execution of a secure world FWU image (after authentication). Secure world certificates and non-AP images are examples of non-executable images.
- Secure or non-secure image. This indicates whether the image is authenticated/executed in secure or

non-secure memory.

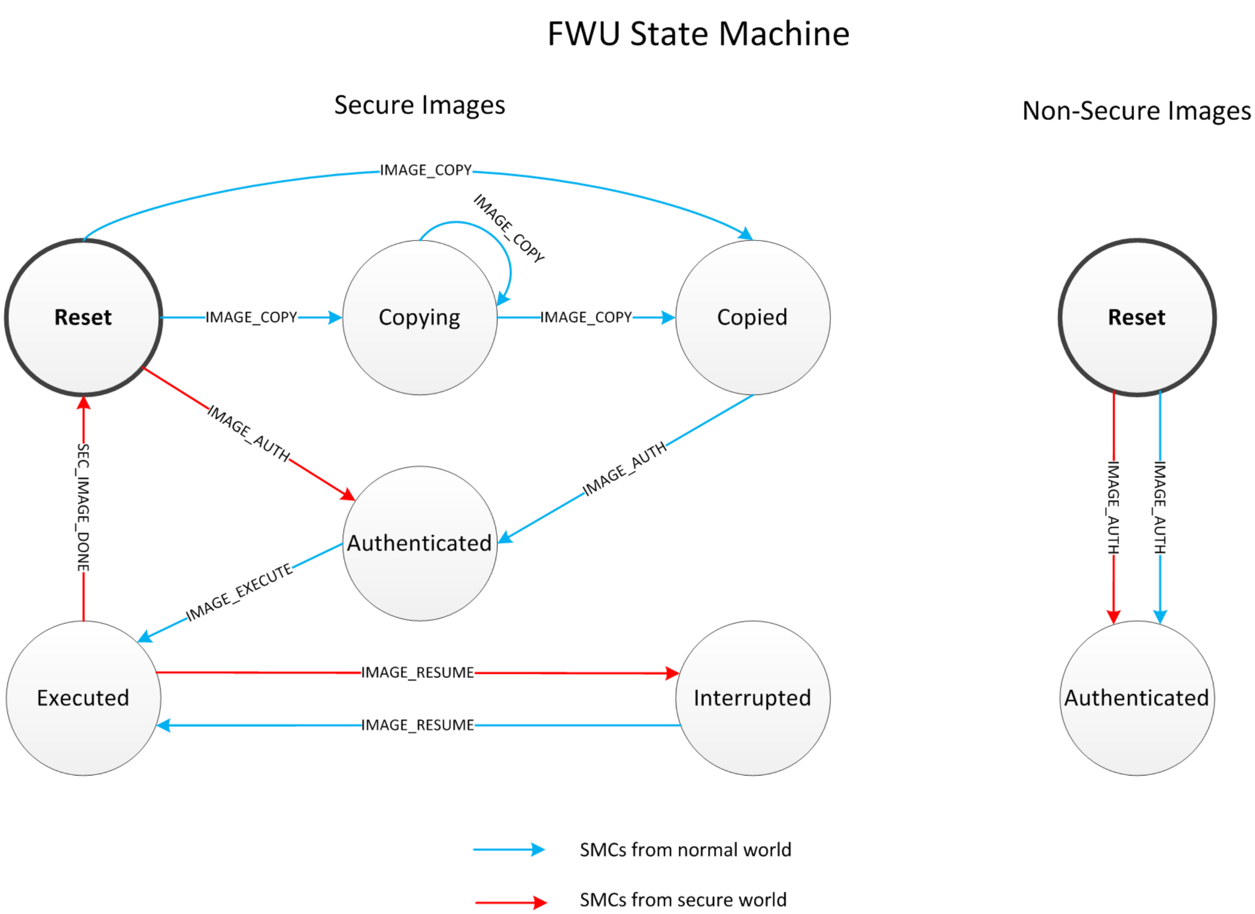
- Image base address and size.
- Image entry point configuration (an `entry_point_info_t`).
- FWU image state.

BL1 uses the FWU image descriptors to:

- Validate the arguments of FWU SMCs
- Manage the state of the FWU process
- Initialize the execution state of the next FWU image.

FWU State Machine

BL1 maintains state for each FWU image during FWU execution. FWU images at lower Exception Levels raise SMCs to invoke FWU functionality in BL1, which causes BL1 to update its FWU image state. The BL1 image states and valid state transitions are shown in the diagram below. Note that secure images have a more complex state machine than non-secure images.



The following is a brief description of the supported states:

- RESET: This is the initial state of every image at the start of FWU. Authentication failure also leads to this state. A secure image may yield to this state if it has completed execution. It can also be reached by using `FWU_SMC_IMAGE_RESET`.
- COPYING: This is the state of a secure image while BL1 is copying it in blocks from non-secure to secure memory.
- COPIED: This is the state of a secure image when BL1 has completed copying it to secure memory.
- AUTHENTICATED: This is the state of an image when BL1 has successfully authenticated it.
- EXECUTED: This is the state of a secure, executable image when BL1 has passed execution control to it.
- INTERRUPTED: This is the state of a secure, executable image after it has requested BL1 to resume normal world execution.

BL1 SMC Interface

BL1_SMC_CALL_COUNT

```
Arguments:
    uint32_t function ID : 0x0

Return:
    uint32_t
```

This SMC returns the number of SMCs supported by BL1.

BL1_SMC_UID

```
Arguments:
    uint32_t function ID : 0x1

Return:
    UUID : 32 bits in each of w0-w3 (or r0-r3 for AArch32 callers)
```

This SMC returns the 128-bit Universally Unique Identifier for the BL1 SMC service.

BL1_SMC_VERSION

```
Argument:
    uint32_t function ID : 0x3

Return:
    uint32_t : Bits [31:16] Major Version
               Bits [15:0] Minor Version
```

This SMC returns the current version of the BL1 SMC service.

BL1_SMC_RUN_IMAGE

```
Arguments:
    uint32_t           function ID : 0x4
    entry_point_info_t *ep_info

Return:
    void

Pre-conditions:
    if (normal world caller) synchronous exception
    if (ep_info not EL3) synchronous exception
```

This SMC passes execution control to an EL3 image described by the provided `entry_point_info_t` structure. In the normal TF-A boot flow, BL2 invokes this SMC for BL1 to pass execution control to BL31.

FWU_SMC_IMAGE_COPY

```
Arguments:
    uint32_t     function ID : 0x10
    unsigned int image_id
    uintptr_t    image_addr
    unsigned int block_size
    unsigned int image_size

Return:
    int : 0 (Success)
        : -ENOMEM
        : -EPERM

Pre-conditions:
    if (image_id is invalid) return -EPERM
    if (image_id is non-secure image) return -EPERM
    if (image_id state is not (RESET or COPYING)) return -EPERM
    if (secure world caller) return -EPERM
    if (image_addr + block_size overflows) return -ENOMEM
    if (image destination address + image_size overflows) return -ENOMEM
    if (source block is in secure memory) return -ENOMEM
    if (source block is not mapped into BL1) return -ENOMEM
    if (image_size > free secure memory) return -ENOMEM
    if (image overlaps another image) return -EPERM
```

This SMC copies the secure image indicated by `image_id` from non-secure memory to secure memory for later authentication. The image may be copied in a single block or multiple blocks. In either case, the total size of the image must be provided in `image_size` when invoking this SMC for the first time for each image; it is ignored in subsequent calls (if any) for the same image.

The `image_addr` and `block_size` specify the source memory block to copy from. The destination address is provided by the platform code.

If `block_size` is greater than the amount of remaining bytes to copy for this image then the former is truncated to the latter. The copy operation is then considered as complete and the FWU state machine transitions to the "COPIED" state. If there is still more to copy, the FWU state machine stays in or transitions to the COPYING state (depending on the previous state).

When using multiple blocks, the source blocks do not necessarily need to be in contiguous memory.

Once the SMC is handled, BL1 returns from exception to the normal world caller.

FWU_SMC_IMAGE_AUTH

```
Arguments:
    uint32_t     function ID : 0x11
    unsigned int image_id
    uintptr_t    image_addr
    unsigned int image_size

Return:
    int : 0 (Success)
        : -ENOMEM
        : -EPERM
        : -EAUTH

Pre-conditions:
    if (image_id is invalid) return -EPERM
    if (secure world caller)
        if (image_id state is not RESET) return -EPERM
        if (image_addr/image_size is not mapped into BL1) return -ENOMEM
    else // normal world caller
        if (image_id is secure image)
            if (image_id state is not COPIED) return -EPERM
        else // image_id is non-secure image
            if (image_id state is not RESET) return -EPERM
            if (image_addr/image_size is in secure memory) return -ENOMEM
            if (image_addr/image_size not mapped into BL1) return -ENOMEM
```

This SMC authenticates the image specified by `image_id`. If the image is in the RESET state, BL1 authenticates the image in place using the provided `image_addr` and `image_size`. If the image is a secure image in the COPIED state, BL1 authenticates the image from the secure memory that BL1 previously copied the image into.

BL1 returns from exception to the caller. If authentication succeeds then BL1 sets the image state to AUTHENTICATED. If authentication fails then BL1 returns the -EAUTH error and sets the image state back to RESET.

FWU_SMC_IMAGE_EXECUTE

```
Arguments:
    uint32_t     function ID : 0x12
    unsigned int image_id

Return:
    int : 0 (Success)
        : -EPERM

Pre-conditions:
    if (image_id is invalid) return -EPERM
    if (secure world caller) return -EPERM
    if (image_id is non-secure image) return -EPERM
    if (image_id is non-executable image) return -EPERM
    if (image_id state is not AUTHENTICATED) return -EPERM
```

This SMC initiates execution of a previously authenticated image specified by `image_id`, in the other security world to the caller. The current implementation only supports normal world callers initiating execution of a secure world image.

BL1 saves the normal world caller's context, sets the secure image state to EXECUTED, and returns from exception to the secure image.

FWU_SMC_IMAGE_RESUME

```
Arguments:
    uint32_t   function ID : 0x13
    register_t image_param

Return:
    register_t : image_param (Success)
               : -EPERM

Pre-conditions:
    if (normal world caller and no INTERRUPTED secure image) return -EPERM
```

This SMC resumes execution in the other security world while there is a secure image in the EXECUTED/INTERRUPTED state.

For normal world callers, BL1 sets the previously interrupted secure image state to EXECUTED. For secure world callers, BL1 sets the previously executing secure image state to INTERRUPTED. In either case, BL1 saves the calling world's context, restores the resuming world's context and returns from exception into the resuming world. If the call is successful then the caller provided `image_param` is returned to the resumed world, otherwise an error code is returned to the caller.

FWU_SMC_SEC_IMAGE_DONE

```
Arguments:
    uint32_t function ID : 0x14

Return:
    int : 0 (Success)
        : -EPERM

Pre-conditions:
    if (normal world caller) return -EPERM
```

This SMC indicates completion of a previously executing secure image.

BL1 sets the previously executing secure image state to the RESET state, restores the normal world context and returns from exception into the normal world.

FWU_SMC_UPDATE_DONE

```
Arguments:
    uint32_t   function ID : 0x15
    register_t client_cookie

Return:
    N/A
```

This SMC completes the firmware update process. BL1 calls the platform specific function `bl1_plat_fwu_done`, passing the optional argument `client_cookie` as a `void *`. The SMC does not return.

FWU_SMC_IMAGE_RESET

```
Arguments:
    uint32_t     function ID : 0x16
    unsigned int image_id

Return:
    int : 0 (Success)
        : -EPERM

Pre-conditions:
    if (secure world caller) return -EPERM
    if (image in EXECUTED) return -EPERM
```

This SMC sets the state of an image to RESET and zeroes the memory used by it.

This is only allowed if the image is not being executed.

4.8 FIRME

This page summarizes the current status of the FIRME implementation in TF-A.

4.8.1 Background

FIRME defines standard EL3 firmware interfaces used by software above EL3 to manage capabilities associated with Arm Confidential Compute Architecture (Arm CCA) and, in particular, the Realm Management Extension (FEAT_RME). These interfaces cover the EL3 dependencies needed for Realm-related flows such as resource assignment and isolation, Realm construction, and attestation.

The specification groups these interfaces into FIRME services. The service areas defined by the specification are:

- granule protection table management
- IDE key management for PCIe and CXL root ports
- memory encryption context management
- attestation token management
- integrated device management

In addition to those service-specific interfaces, FIRME defines a base service that provides common concepts plus version and feature discovery for the other services.

FIRME sits on the EL3 boundary and is intended to be callable from the Non-secure, Realm, and Secure security states. A FIRME implementation is also permitted on systems that do not implement FEAT_RME, for example when parts of the interface are still useful independently of Realm support.

TF-A currently implements the Arm FIRME specification version `1.0 alp2` as defined by DEN0149.

FIRME is exposed as a 64-bit fast SMC standard service. The currently allocated TF-A function ID range is `0xC4000400` to `0xC4000412`.

4.8.2 Build and integration

FIRME support in TF-A is experimental:

- `FIRME_SUPPORT` enables support in builds.
- When build with `FIRME_SUPPORT`, RME supported system will make use of the FIRME granule management services.
- When FIRME support is enabled in a RMM enabled build, TRP uses `FIRME_GM_GPI_SET` for granule transitions instead of the legacy delegate/undelegate interface.

4.8.3 Current status

- Only a single FIRME version is supported per service.
- The FIRME base service implements version and feature discovery through `FIRME_SERVICE_VERSION` and `FIRME_SERVICE_FEATURES`.
- `FIRME_GM_GPI_SET` is currently the only implemented granule management operation.
- The IDE key management, MECID management, attestation, and integrated device management services are not implemented yet.
- `FIRME_GM_GPI_OP_CONTINUE` is mandatory in the `1.0 alp2` specification, but it is not implemented yet.

As a result, the current implementation tracks FIRME `1.0 alp2` for versioning purposes, but it does not yet provide all mandatory functionality defined by that specification revision.

4.8.4 Implementation details

- TF-A currently advertises FIRME support for Secure, Non-secure, and Realm callers.
- The base service is the entry point used to discover per-service versions and feature registers.
- The granule management service publishes two feature registers:
  - register 0 advertises `FIRME_GM_GPI_SET` support only
  - register 1 is populated at runtime from `GPCCR_EL3` and reports platform granule protection geometry such as page size, `L0GPTSZ`, and `PPS`
- `FIRME_GM_GPI_SET` is implemented on top of TF-A's generic GPT transition helper and returns the number of successfully transitioned granules.

4.9 Measured Boot Driver (MBD)

4.9.1 Properties binding information

DTB binding for Event Log properties

This document describes the device tree format of Event Log properties. These properties are not related to a specific platform and can be queried from common code.

Dynamic configuration for Event Log

Measured Boot driver expects a *tpm_event_log* node with the following field in 'tb_fw_config', 'nt_fw_config' and 'tsp_fw_config' DTS files:

- **compatible [mandatory]**
  - value type: <string>
  - Must be the string "arm,tpm_event_log".

Then a list of properties representing Event Log configuration, which can be used by Measured Boot driver. Each property is named according to the information it contains:

- **tpm_event_log_sm_addr [fvp_nt_fw_config.dts with OP-TEE]**
  - value type: <u64>
  - Event Log base address in secure memory.

Note. Currently OP-TEE does not support reading DTBs from Secure memory and this property should be removed when this feature is supported.

- **tpm_event_log_addr [mandatory]**
  - value type: <u64>
  - Event Log base address in non-secure memory.
- **tpm_event_log_size [mandatory]**
  - value type: <u32>
  - Event Log size.
- **tpm_event_log_max_size [mandatory]**
  - value type: <u32>
  - Event Log maximum size.

4.10 Maximum Power Mitigation Mechanism (MPMM)

MPMM is an optional microarchitectural power management mechanism supported by some Arm Armv9-A cores, beginning with the Cortex-X2, Cortex-A710 and Cortex-A510 cores. This mechanism detects and limits high-activity events to assist in *SoC* processor power domain dynamic power budgeting and limit the triggering of whole-rail (i.e. clock chopping) responses to overcurrent conditions.

MPMM is enabled on a per-core basis by the EL3 runtime firmware.

Warning: *MPMM* exposes gear metrics through the auxiliary *AMU* counters. An external power controller can use these metrics to budget SoC power by limiting the number of cores that can execute higher-activity workloads or switching to a different DVFS operating point. When this is the case, the *AMU* counters that make up the *MPMM* gears must be enabled by the EL3 runtime firmware - please see *Auxiliary counters* for documentation on enabling auxiliary *AMU* counters.

4.11 Platform Interrupt Controller API

This document lists the optional platform interrupt controller API that abstracts the runtime configuration and control of interrupt controller from the generic code. The mandatory APIs are described in the *Porting Guide*.

4.11.1 Function: unsigned int plat_ic_get_running_priority(void); [optional]

```
Argument : void
Return   : unsigned int
```

This API should return the priority of the interrupt the PE is currently servicing. This must be be called only after an interrupt has already been acknowledged via `plat_ic_acknowledge_interrupt`.

In the case of Arm standard platforms using GIC, the *Running Priority Register* is read to determine the priority of the interrupt.

4.11.2 Function: int plat_ic_is_spi(unsigned int id); [optional]

```
Argument : unsigned int
Return   : int
```

The API should return whether the interrupt ID (first parameter) is categorized as a Shared Peripheral Interrupt. Shared Peripheral Interrupts are typically associated to system-wide peripherals, and these interrupts can target any PE in the system.

4.11.3 Function: int plat_ic_is_ppi(unsigned int id); [optional]

```
Argument : unsigned int
Return   : int
```

The API should return whether the interrupt ID (first parameter) is categorized as a Private Peripheral Interrupt. Private Peripheral Interrupts are typically associated with peripherals that are private to each PE. Interrupts from private peripherals target to that PE only.

4.11.4 Function: int plat_ic_is_sgi(unsigned int id); [optional]

```
Argument : unsigned int
Return   : int
```

The API should return whether the interrupt ID (first parameter) is categorized as a Software Generated Interrupt. Software Generated Interrupts are raised by explicit programming by software, and are typically used in inter-PE communication. Secure SGIs are reserved for use by Secure world software.

4.11.5 Function: unsigned int plat_ic_get_interrupt_active(unsigned int id); [optional]

```
Argument : unsigned int
Return   : int
```

This API should return the *active* status of the interrupt ID specified by the first parameter, `id`.

In case of Arm standard platforms using GIC, the implementation of the API reads the GIC *Set Active Register* to read and return the active status of the interrupt.

4.11.6 Function: void plat_ic_enable_interrupt(unsigned int id); [optional]

```
Argument : unsigned int
Return   : void
```

This API should enable the interrupt ID specified by the first parameter, `id`. PEs in the system are expected to receive only enabled interrupts.

In case of Arm standard platforms using GIC, the implementation of the API inserts barrier to make memory updates visible before enabling interrupt, and then writes to GIC *Set Enable Register* to enable the interrupt.

4.11.7 Function: void plat_ic_disable_interrupt(unsigned int id); [optional]

```
Argument : unsigned int
Return   : void
```

This API should disable the interrupt ID specified by the first parameter, `id`. PEs in the system are not expected to receive disabled interrupts.

In case of Arm standard platforms using GIC, the implementation of the API writes to GIC *Clear Enable Register* to disable the interrupt, and inserts barrier to make memory updates visible afterwards.

4.11.8 Function: void plat_ic_set_interrupt_priority(unsigned int id, unsigned int priority); [optional]

```
Argument : unsigned int
Argument : unsigned int
Return   : void
```

This API should set the priority of the interrupt specified by first parameter `id` to the value set by the second parameter `priority`.

In case of Arm standard platforms using GIC, the implementation of the API writes to GIC *Priority Register* set interrupt priority.

4.11.9 Function: bool plat_ic_has_interrupt_type(unsigned int type); [optional]

```
Argument : unsigned int
Return   : bool
```

This API should return whether the platform supports a given interrupt type. The parameter `type` shall be one of `INTR_TYPE_EL3`, `INTR_TYPE_S_EL1`, or `INTR_TYPE_NS`.

In case of Arm standard platforms using GICv3, the implementation of the API returns *true* for all interrupt types.

In case of Arm standard platforms using GICv2, the API always return *true* for `INTR_TYPE_NS`. Return value for other types depends on the value of build option `GICV2_G0_FOR_EL3`:

- For interrupt type `INTR_TYPE_EL3`:
  - When `GICV2_G0_FOR_EL3` is `0`, it returns *false*, indicating no support for EL3 interrupts.
  - When `GICV2_G0_FOR_EL3` is `1`, it returns *true*, indicating support for EL3 interrupts.
- For interrupt type `INTR_TYPE_S_EL1`:
  - When `GICV2_G0_FOR_EL3` is `0`, it returns *true*, indicating support for Secure EL1 interrupts.
  - When `GICV2_G0_FOR_EL3` is `1`, it returns *false*, indicating no support for Secure EL1 interrupts.

4.11.10 Function: void plat_ic_set_interrupt_type(unsigned int id, unsigned int type); [optional]

```
Argument : unsigned int
Argument : unsigned int
Return   : void
```

This API should set the interrupt specified by first parameter `id` to the type specified by second parameter `type`. The `type` parameter can be one of:

- `INTR_TYPE_NS`: interrupt is meant to be consumed by the Non-secure world.

- `INTR_TYPE_S_EL1`: interrupt is meant to be consumed by Secure EL1.
- `INTR_TYPE_EL3`: interrupt is meant to be consumed by EL3.

In case of Arm standard platforms using GIC, the implementation of the API writes to the GIC *Group Register* and *Group Modifier Register* (only GICv3) to assign the interrupt to the right group.

For GICv3:

- `INTR_TYPE_NS` maps to Group 1 interrupt.
- `INTR_TYPE_S_EL1` maps to Secure Group 1 interrupt.
- `INTR_TYPE_EL3` maps to Secure Group 0 interrupt.

For GICv2:

- `INTR_TYPE_NS` maps to Group 1 interrupt.
- When the build option `GICV2_G0_FOR_EL3` is set to `0` (the default), `INTR_TYPE_S_EL1` maps to Group 0. Otherwise, `INTR_TYPE_EL3` maps to Group 0 interrupt.

4.11.11 Function: void plat_ic_raise_el3_sgi(int sgi_num, u_register_t target); [optional]

```
Argument : int
Argument : u_register_t
Return   : void
```

This API should raise an EL3 SGI. The first parameter, `sgi_num`, specifies the ID of the SGI. The second parameter, `target`, must be the MPIDR of the target PE.

In case of Arm standard platforms using GIC, the implementation of the API inserts barrier to make memory updates visible before raising SGI, then writes to appropriate *SGI Register* in order to raise the EL3 SGI.

4.11.12 Function: void plat_ic_set_spi_routing(unsigned int id, unsigned int routing_mode, u_register_t mpidr); [optional]

```
Argument : unsigned int
Argument : unsigned int
Argument : u_register_t
Return   : void
```

This API should set the routing mode of Share Peripheral Interrupt (SPI) specified by first parameter `id` to that specified by the second parameter `routing_mode`.

The `routing_mode` parameter can be one of:

- `INTR_ROUTING_MODE_ANY` means the interrupt can be routed to any PE in the system. The `mpidr` parameter is ignored in this case.
- `INTR_ROUTING_MODE_PE` means the interrupt is routed to the PE whose MPIDR value is specified by the parameter `mpidr`.

In case of Arm standard platforms using GIC, the implementation of the API writes to the GIC *Target Register* (GICv2) or *Route Register* (GICv3) to set the routing.

4.11.13 Function: void plat_ic_set_interrupt_pending(unsigned int id); [optional]

```
Argument : unsigned int
Return   : void
```

This API should set the interrupt specified by first parameter `id` to *Pending*.

In case of Arm standard platforms using GIC, the implementation of the API inserts barrier to make memory updates visible before setting interrupt pending, and writes to the GIC *Set Pending Register* to set the interrupt pending status.

4.11.14 Function: void plat_ic_clear_interrupt_pending(unsigned int id); [optional]

```
Argument : unsigned int
Return   : void
```

This API should clear the *Pending* status of the interrupt specified by first parameter `id`.

In case of Arm standard platforms using GIC, the implementation of the API writes to the GIC *Clear Pending Register* to clear the interrupt pending status, and inserts barrier to make memory updates visible afterwards.

4.11.15 Function: unsigned int plat_ic_set_priority_mask(unsigned int id); [optional]

```
Argument : unsigned int
Return   : int
```

This API should set the priority mask (first parameter) in the interrupt controller such that only interrupts of higher priority than the supplied one may be signalled to the PE. The API should return the current priority value that it's overwriting.

In case of Arm standard platforms using GIC, the implementation of the API inserts barriers to order memory updates before updating mask, then writes to the GIC *Priority Mask Register*, and make sure memory updates are visible before potential trigger due to mask update.

4.11.16 Function: unsigned int plat_ic_deactivate_priority(unsigned int id); [optional]

```
Argument : unsigned int
Return   : int
```

This API performs the operations of plat_ic_set_priority_mask along with calling the errata workaround gicv3_apply_errata_wa_2384374(). This is performed when priority mask is restored to it's older value. This API returns the current priority value that it's overwriting.

In case of Arm standard platforms using GIC, the implementation of the API inserts barriers to order memory updates before updating mask, then writes to the GIC *Priority Mask Register*, and make sure memory updates are visible before potential trigger due to mask update, and applies 2384374 GIC errata workaround to process pending interrupt packets.

4.11.17 Function: unsigned int plat_ic_get_interrupt_id(unsigned int raw); [optional]

```
Argument : unsigned int
Return   : unsigned int
```

This API should extract and return the interrupt number from the raw value obtained by the acknowledging the interrupt (read using `plat_ic_acknowledge_interrupt()`). If the interrupt ID is invalid, this API should return `INTR_ID_UNAVAILABLE`.

In case of Arm standard platforms using GIC, the implementation of the API masks out the interrupt ID field from the acknowledged value from GIC.

4.12 Reliability, Availability, and Serviceability (RAS) Extensions

This document describes *TF-A* support for Arm Reliability, Availability, and Serviceability (RAS) extensions. RAS is a mandatory extension for Armv8.2 and later CPUs, and also an optional extension to the base Armv8.0 architecture.

For the description of Arm RAS extensions, Standard Error Records, and the precise definition of RAS terminology, please refer to the Arm Architecture Reference Manual and RAS Supplement. The rest of this document assumes familiarity with architecture and terminology.

IMPORTANT NOTE: TF-A implementation assumes that if RAS extension is present then FEAT_IESB is also implemented.

There are two philosophies for handling RAS errors from Non-secure world point of view.

- *Firmware First Handling (FFH)*
- *Kernel First Handling (KFH)*

4.12.1 Firmware First Handling (FFH)

Introduction

EA's and Error interrupts corresponding to NS nodes are handled first in firmware

- Errors signaled back to NS world via suitable mechanism
- Kernel is prohibited from accessing the RAS error records directly
- Firmware creates CPER records for kernel to navigate and process
- Firmware signals error back to Kernel via SDEI

Overview

FFH works in conjunction with *Exception Handling Framework*. Exceptions resulting from errors in Non-secure world are routed to and handled in EL3. Said errors are Synchronous External Abort (SEA), Asynchronous External Abort (signalled as SErrors), Fault Handling and Error Recovery interrupts. RAS Framework in TF-A allows the platform to define an external abort handler and to register RAS nodes and interrupts. It also provides *helpers* for accessing Standard Error Records as introduced by the RAS extensions

4.12.2 Kernel First Handling (KFH)

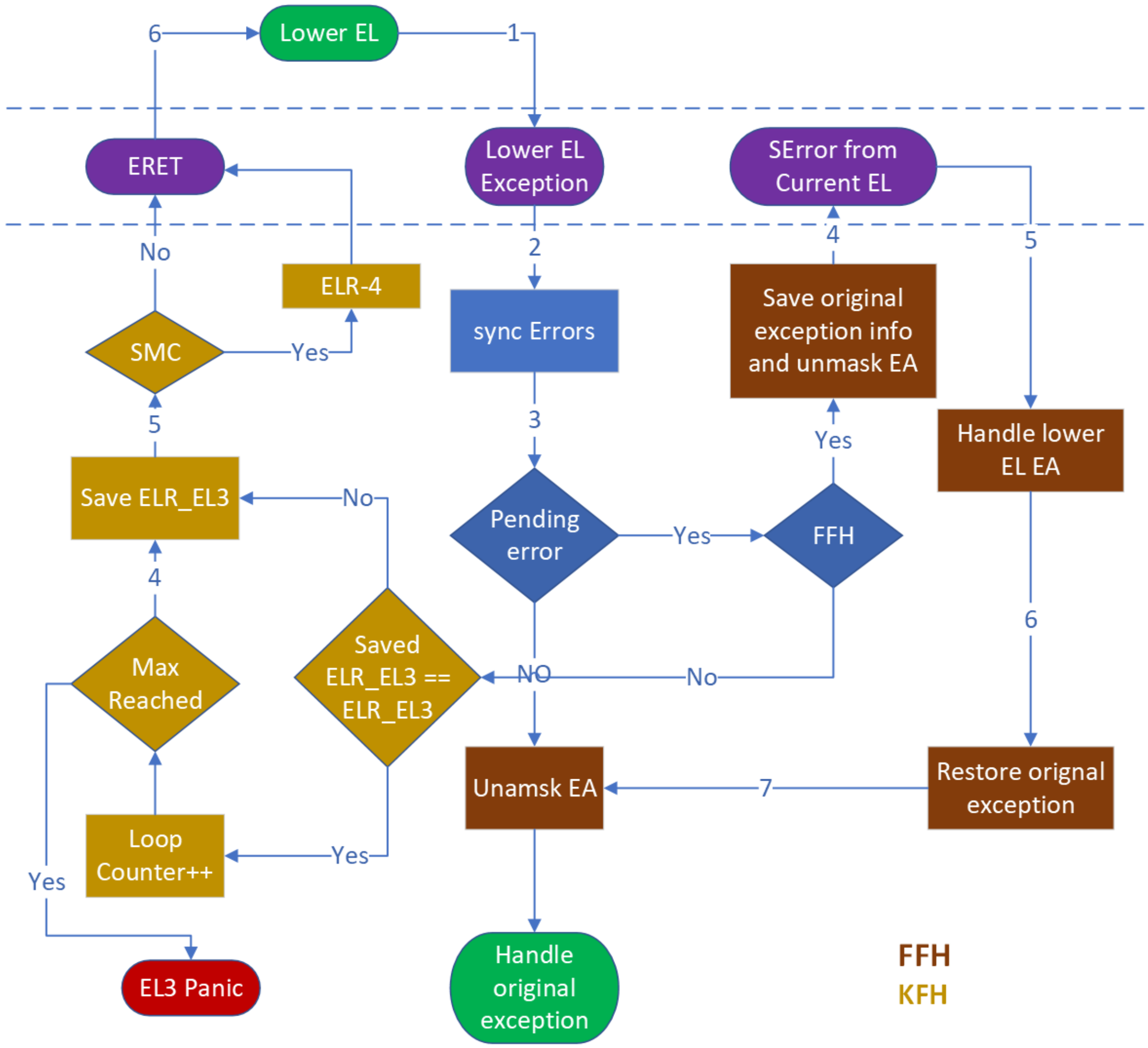
Introduction

EA's originating/attributed to NS world are handled first in NS and Kernel navigates the std error records directly.

- KFH is the default handling mode if platform does not explicitly enable FFH mode.
- KFH mode does not need any EL3 involvement except for the reflection of errors back to lower EL. This happens when there is an error (EA) in the system which is not yet signaled to PE while executing at lower EL. During entry into EL3 the errors (EA) are synchronized causing async EA to pend at EL3.

4.12.3 Error Syncronization at EL3 entry

During entry to EL3 from lower EL, if there is any pending async EAs they are either reflected back to lower EL (KFH) or handled in EL3 itself (FFH).



Limitation in KFH Mode

When handling asynchronous External Aborts (EAs) synchronized at EL3 entry in Kernel First Handling (KFH) mode, there is a limitation in the current implementation:

- The handler reflects pending async EAs back to the lower EL if the EA routing model is KFH
- However, if the asynchronous EA is masked at the target exception level, or if its priority relative to an EL3/secure interrupt is lower, repeated back-and-forth transitions between lower EL and EL3 can occur.

To prevent infinite cycling between EL3 and lower EL, a loop counter (`CTX_NESTED_EA_FLAG`) and the previously saved ELR (`CTX_SAVED_ELR_EL3`) are used to detect this condition. If a loop is detected, EL3 will trigger a panic (label `check_loop_ctr`) to indicate a problem.

Future Plan: Delegated SError Injection (FEAT_E3DSE)

In future revisions, this limitation can be mitigated by utilizing **FEAT_E3DSE** — the **Delegated SError exception injection** feature introduced for EL3.

FEAT_E3DSE provides a mechanism for EL3 to inject a virtual SError into lower exception levels. Once this capability is supported in TF-A, EL3 will be able to handle the original exception and then inject the delegated SError to the appropriate lower EL before returning, thereby eliminating the need for panic handling in this scenario.

This planned enhancement will improve robustness and correctness of asynchronous error handling in KFH mode.

4.12.4 TF-A build options

- **ENABLE_FEAT_RAS**: Enable RAS extension feature at EL3.
- **HANDLE_EA_EL3_FIRST_NS**: Required for FFH
- **RAS_TRAP_NS_ERR_REC_ACCESS**: Trap Non-secure access of RAS error record registers.
- **RAS_EXTENSION**: Deprecated macro, equivalent to ENABLE_FEAT_RAS and HANDLE_EA_EL3_FIRST_NS put together.

RAS internal macros

- **FFH_SUPPORT**: Gets enabled if **HANDLE_EA_EL3_FIRST_NS** is enabled.

RAS feature has dependency on some other TF-A build flags

- **EL3_EXCEPTION_HANDLING**: Required for FFH
- **FAULT_INJECTION_SUPPORT**: Required for testing RAS feature on fvp platform

4.12.5 TF-A Tests

RAS functionality is regularly tested in TF-A CI using RAS test group which has multiple configurations for testing lower EL External aborts.

All the tests are written in TF-A tests which runs as NS-EL2 payload.

- **FFH without RAS extension**

  fvp-ea-ffh,fvp-ea-ffh:fvp-tftf-fip.tftf-aemv8a-debug

  > Couple of tests, one each for sync EA and async EA from lower EL which gets handled in El3. Inject External aborts(sync/async) which traps in EL3, FVP has a handler which gracefully handles these errors and returns back to TF-A Tests
  >
  > Build Configs : **HANDLE_EA_EL3_FIRST_NS** , **PLATFORM_TEST_EA_FFH**

- **FFH with RAS extension**

  Three Tests :

– *fvp-ras-ffh,fvp-single-fault:fvp-tftf-fip.tftf-aemv8a.fi-debug*

  Inject an unrecoverable RAS error, which gets handled in EL3.

– *fvp-ras-ffh,fvp-uncontainable:fvp-tftf.fault-fip.tftf-aemv8a.fi-debug*

  Inject uncontainable RAS errors which causes platform to panic.

– *fvp-ras-ffh,fvp-ras-ffh-nested:fvp-tftf-fip.tftf-ras_ffh_nested-aemv8a.fi-debug*

  Test nested exception handling at El3 for synchronized async EAs. Inject an SError in lower EL which remain pending until we enter EL3 through SMC call. At EL3 entry on encountering a pending async EA it will handle the async EA first (nested exception) before handling the original SMC call.

- **KFH with RAS extension**

  Couple of tests in the group :

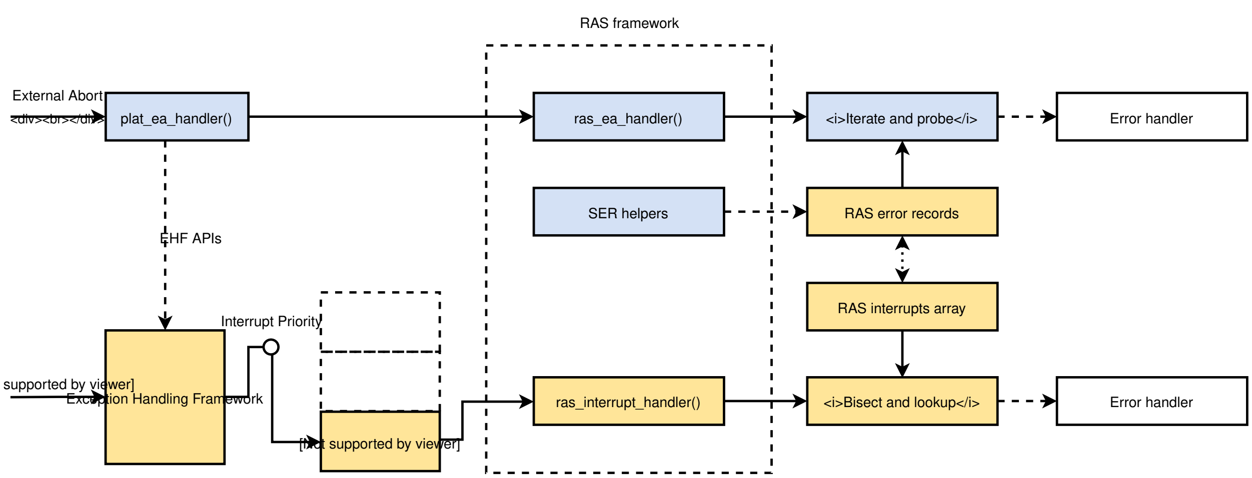
  - *fvp-ras-kfh,fvp-ras-kfh:fvp-tftf-fip.tftf-aemv8a.fi-debug*

    Inject and handle RAS errors in TF-A tests (no El3 involvement)

  - *fvp-ras-kfh,fvp-ras-kfh-reflect:fvp-tftf-fip.tftf-ras_kfh_reflection-aemv8a.fi-debug*

    Reflection of synchronized errors from EL3 to TF-A tests, two tests one each for reflecting in IRQ and SMC path.

4.12.6 RAS Framework



Platform APIs

The RAS framework allows the platform to define handlers for External Abort, Uncontainable Errors, Double Fault, and errors rising from EL3 execution. Please refer to *RAS Porting Guide*.

Registering RAS error records

RAS nodes are components in the system capable of signalling errors to PEs through one one of the notification mechanisms—SEAs, SErrors, or interrupts. RAS nodes contain one or more error records, which are registers through which the nodes advertise various properties of the signalled error. Arm recommends that error records are implemented in the Standard Error Record format. The RAS architecture allows for error records to be accessible via system or memory-mapped registers.

The platform should enumerate the error records providing for each of them:

- A handler to probe error records for errors;
- When the probing identifies an error, a handler to handle it;
- For memory-mapped error record, its base address and size in KB; for a system register-accessed record, the start index of the record and number of continuous records from that index;
- Any node-specific auxiliary data.

With this information supplied, when the run time firmware receives one of the notification mechanisms, the RAS framework can iterate through and probe error records for error, and invoke the appropriate handler to handle it.

The RAS framework provides the macros to populate error record information. The macros are versioned, and the latest version as of this writing is 1. These macros create a structure of type `struct err_record_info` from its arguments, which are later passed to probe and error handlers.

For memory-mapped error records:

```
ERR_RECORD_MEMMAP_V1(base_addr, size_num_k, probe, handler, aux)
```

And, for system register ones:

```
ERR_RECORD_SYSREG_V1(idx_start, num_idx, probe, handler, aux)
```

The probe handler must have the following prototype:

```
typedef int (*err_record_probe_t)(const struct err_record_info *info,
                int *probe_data);
```

The probe handler must return a non-zero value if an error was detected, or 0 otherwise. The `probe_data` output parameter can be used to pass any useful information resulting from probe to the error handler (see *below*). For example, it could return the index of the record.

The error handler must have the following prototype:

```
typedef int (*err_record_handler_t)(const struct err_record_info *info,
           int probe_data, const struct err_handler_data *const data);
```

The `data` constant parameter describes the various properties of the error, including the reason for the error, exception syndrome, and also `flags`, `cookie`, and `handle` parameters from the *top-level exception handler*.

The platform is expected populate an array using the macros above, and register the it with the RAS framework using the macro `REGISTER_ERR_RECORD_INFO()`, passing it the name of the array describing the records. Note that the macro must be used in the same file where the array is defined.

Standard Error Record helpers

The *TF-A* RAS framework provides probe handlers for Standard Error Records, for both memory-mapped and System Register accesses:

```
int ras_err_ser_probe_memmap(const struct err_record_info *info,
            int *probe_data);

int ras_err_ser_probe_sysreg(const struct err_record_info *info,
            int *probe_data);
```

When the platform enumerates error records, for those records in the Standard Error Record format, these helpers maybe used instead of rolling out their own. Both helpers above:

- Return non-zero value when an error is detected in a Standard Error Record;
- Set `probe_data` to the index of the error record upon detecting an error.

Registering RAS interrupts

RAS nodes can signal errors to the PE by raising Fault Handling and/or Error Recovery interrupts. For the firmware-first handling paradigm for interrupts to work, the platform must setup and register with *EHF*. See *Interaction with Exception Handling Framework*.

For each RAS interrupt, the platform has to provide structure of type `struct ras_interrupt`:

- Interrupt number;
- The associated error record information (pointer to the corresponding `struct err_record_info`);
- Optionally, a cookie.

The platform is expected to define an array of `struct ras_interrupt`, and register it with the RAS framework using the macro `REGISTER_RAS_INTERRUPTS()`, passing it the name of the array. Note that the macro must be used in the same file where the array is defined.

The array of `struct ras_interrupt` must be sorted in the increasing order of interrupt number. This allows for fast look of handlers in order to service RAS interrupts.

Double-fault handling

A Double Fault condition arises when an error is signalled to the PE while handling of a previously signalled error is still underway. When a Double Fault condition arises, the Arm RAS extensions only require for handler to perform orderly shutdown of the system, as recovery may be impossible.

The RAS extensions part of Armv8.4 introduced new architectural features to deal with Double Fault conditions, specifically, the introduction of `NMEA` and `EASE` bits to `SCR_EL3` register. These were introduced to assist EL3 software which runs part of its entry/exit routines with exceptions momentarily masked—meaning, in such systems, External Aborts/SErrors are not immediately handled when they occur, but only after the exceptions are unmasked again.

TF-A, for legacy reasons, executes entire EL3 with all exceptions unmasked. This means that all exceptions routed to EL3 are handled immediately. *TF-A* thus is able to detect a Double Fault conditions in software, without needing the intended advantages of Armv8.4 Double Fault architecture extensions.

Double faults are fatal, and terminate at the platform double fault handler, and doesn't return.

Engaging the RAS framework

Enabling RAS support is a platform choice

The RAS support in *TF-A* introduces a default implementation of `plat_ea_handler`, the External Abort handler in EL3. When `ENABLE_FEAT_RAS` is set to `1`, it'll first call `ras_ea_handler()` function, which is the top-level RAS exception handler. `ras_ea_handler` is responsible for iterating to through platform-supplied error records, probe them, and when an error is identified, look up and invoke the corresponding error handler. When `ENABLE_FEAT_RAS` is set to `2` the handler will also be built but it will not be called if `FEAT_RAS` is not present in hardware. The checks to do so will incur a performance penalty.

Note that, if the platform chooses to override the `plat_ea_handler` function and intend to use the RAS framework, it must explicitly call `ras_ea_handler()` from within.

Similarly, for RAS interrupts, the framework defines `ras_interrupt_handler()`. The RAS framework arranges for it to be invoked when a RAS interrupt taken at EL3. The function bisects the platform-supplied sorted array of interrupts to look up the error record information associated with the interrupt number. That error handler for that record is then invoked to handle the error.

Interaction with Exception Handling Framework

As mentioned in earlier sections, RAS framework interacts with the *EHF* to arbitrate handling of RAS exceptions with others that are routed to EL3. This means that the platform must partition a *priority level* for handling RAS exceptions. The platform must then define the macro `PLAT_RAS_PRI` to the priority level used for RAS exceptions. Platforms would typically want to allocate the highest secure priority for RAS handling.

Handling of both *interrupt* and *non-interrupt* exceptions follow the sequences outlined in the *EHF* documentation. I.e., for interrupts, the priority management is implicit; but for non-interrupt exceptions, they're explicit using *EHF APIs*.

4.13 Library at ROM

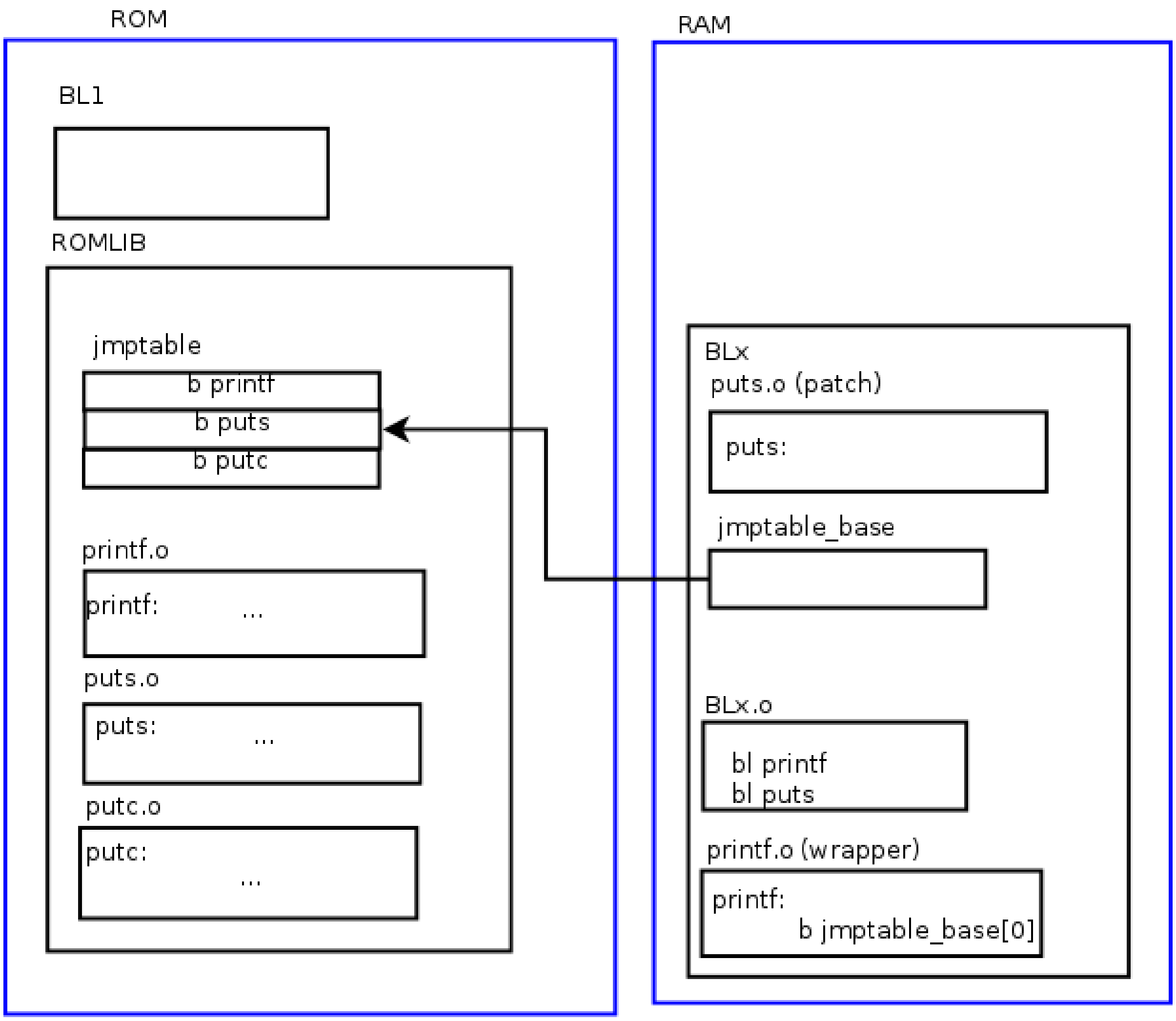
This document provides an overview of the "library at ROM" implementation in Trusted Firmware-A (TF-A).

4.13.1 Introduction

The "library at ROM" feature allows platforms to build a library of functions to be placed in ROM. This reduces SRAM usage by utilising the available space in ROM. The "library at ROM" contains a jump table with the list of functions that are placed in ROM. The capabilities of the "library at ROM" are:

1. Functions can be from one or several libraries.
2. Functions can be patched after they have been programmed into ROM.
3. Platform-specific libraries can be placed in ROM.
4. Functions can be accessed by one or more BL images.

4.13.2 Index file



Library at ROM is described by an index file with the list of functions to be placed in ROM. The index file is platform specific and its format is:

```
lib function    [patch]

lib      -- Name of the library the function belongs to
function -- Name of the function to be placed in library at ROM
[patch]  -- Option to patch the function
```

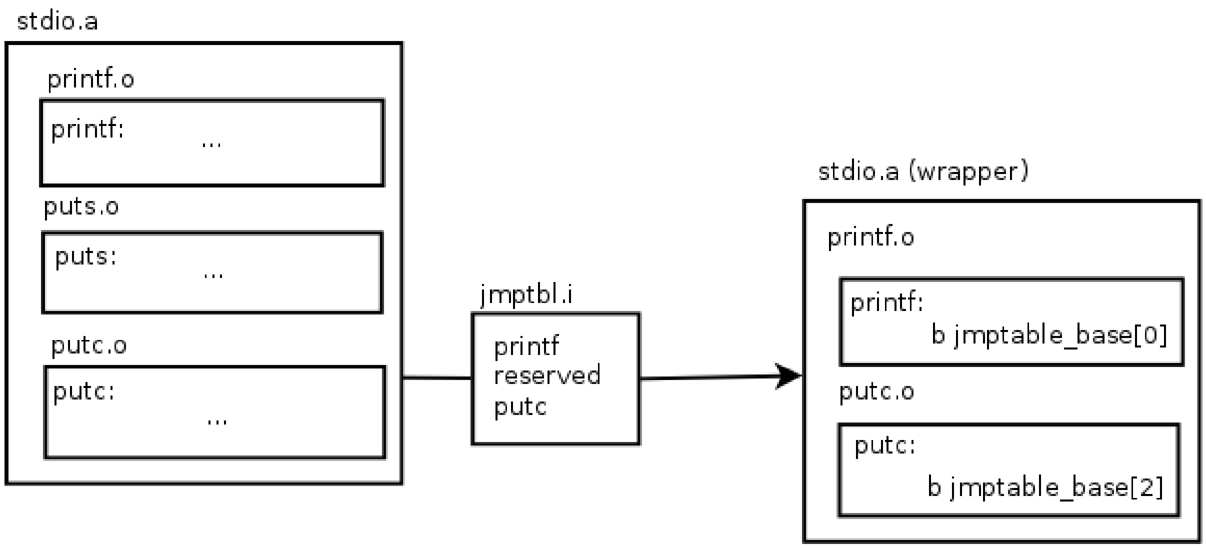
It is also possible to insert reserved spaces in the list by using the keyword "reserved" rather than the "lib" and "function" names as shown below:

```
reserved
```

The reserved spaces can be used to add more functions in the future without affecting the order and location of functions already existing in the jump table. Also, for additional flexibility and modularity, the index file can include other index files.

For an index file example, refer to `lib/romlib/jmptbl.i`.

4.13.3 Wrapper functions



When invoking a function of the "library at ROM", the calling sequence is as follows:

BL image –> wrapper function –> jump table entry –> library at ROM

The index file is used to create a jump table which is placed in ROM. Then, the wrappers refer to the jump table to call the "library at ROM" functions. The wrappers essentially contain a branch instruction to the jump table entry corresponding to the original function. Finally, the original function in the BL image(s) is replaced with the wrapper function.

The "library at ROM" contains a necessary init function that initialises the global variables defined by the functions inside "library at ROM".

Wrapper functions are specified at the link stage of compilation and cannot interpose uppon functions within the same translation unit. For example, if function `fn_a` calls `fn_b` within translation unit `functions.c` and the romlib jump table includes an entry for `fn_b`, `fn_a` will include a reference to `fn_b`'s original program text instead of the wrapper. Thus the jumptable author must take care to include public entry points into translation units to avoid paying the program text cost twice, once in the original executable and once in romlib.

4.13.4 Script

There is a `romlib_generator.py` Python script that generates the necessary files for the "library at ROM" to work. It implements multiple functions:

1. `romlib_generator.py gentbl [args]` - Generates the jump table by parsing the index file.
2. `romlib_generator.py genvar [args]` - Generates the jump table global variable (**not** the jump table itself) with the absolute address in ROM. This global variable is, basically, a pointer to the jump table.

3. `romlib_generator.py genwrappers [args]` - Generates a wrapper function for each entry in the index file except for the ones that contain the keyword `patch`. The generated wrapper file is called `wrappers.s`.
4. `romlib_generator.py pre [args]` - Preprocesses the index file which means it resolves all the include commands in the file recursively. It can also generate a dependency file of the included index files which can be directly used in makefiles.

Each `romlib_generator.py` function has its own manual which is accessible by runing `romlib_generator.py [function] --help`.

`romlib_generator.py` requires Python 3 environment.

4.13.5 Patching of functions in library at ROM

The `romlib_generator.py genwrappers` does not generate wrappers for the entries in the index file that contain the keyword `patch`. Thus, it allows calling the function from the actual library by breaking the link to the "library at ROM" version of this function.

The calling sequence for a patched function is as follows:

BL image –> function

4.13.6 Memory impact

Using library at ROM will modify the memory layout of the BL images:

- The ROM library needs a page aligned RAM section to hold the RW data. This section is defined by the ROMLIB_RW_BASE and ROMLIB_RW_END macros. On Arm platforms a section of 1 page (0x1000) is allocated at the top of SRAM. This will have for effect to shift down all the BL images by 1 page.
- Depending on the functions moved to the ROM library, the size of the BL images will be reduced. For example: moving MbedTLS function into the ROM library reduces BL1 and BL2, but not BL31.
- This change in BL images size can be taken into consideration to optimize the memory layout when defining the BLx_BASE macros.

4.13.7 Build library at ROM

The environment variable `CROSS_COMPILE` must be set appropriately. Refer to *Performing an Initial Build* for more information about setting this variable.

In the below example the usage of ROMLIB together with mbed TLS is demonstrated to showcase the benefits of library at ROM - it's not mandatory.

```
make PLAT=fvp                                                  \
TRUSTED_BOARD_BOOT=1 GENERATE_COT=1                            \
ARM_ROTPK_LOCATION=devel_rsa                                   \
ROT_KEY=plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem       \
```

(continues on next page)

(continued from previous page)

```
BL33=</path/to/bl33.bin>                                           \
USE_ROMLIB=1                                                       \
all fip
```


4.14 SDEI: Software Delegated Exception Interface

This document provides an overview of the SDEI dispatcher implementation in Trusted Firmware-A (TF-A).

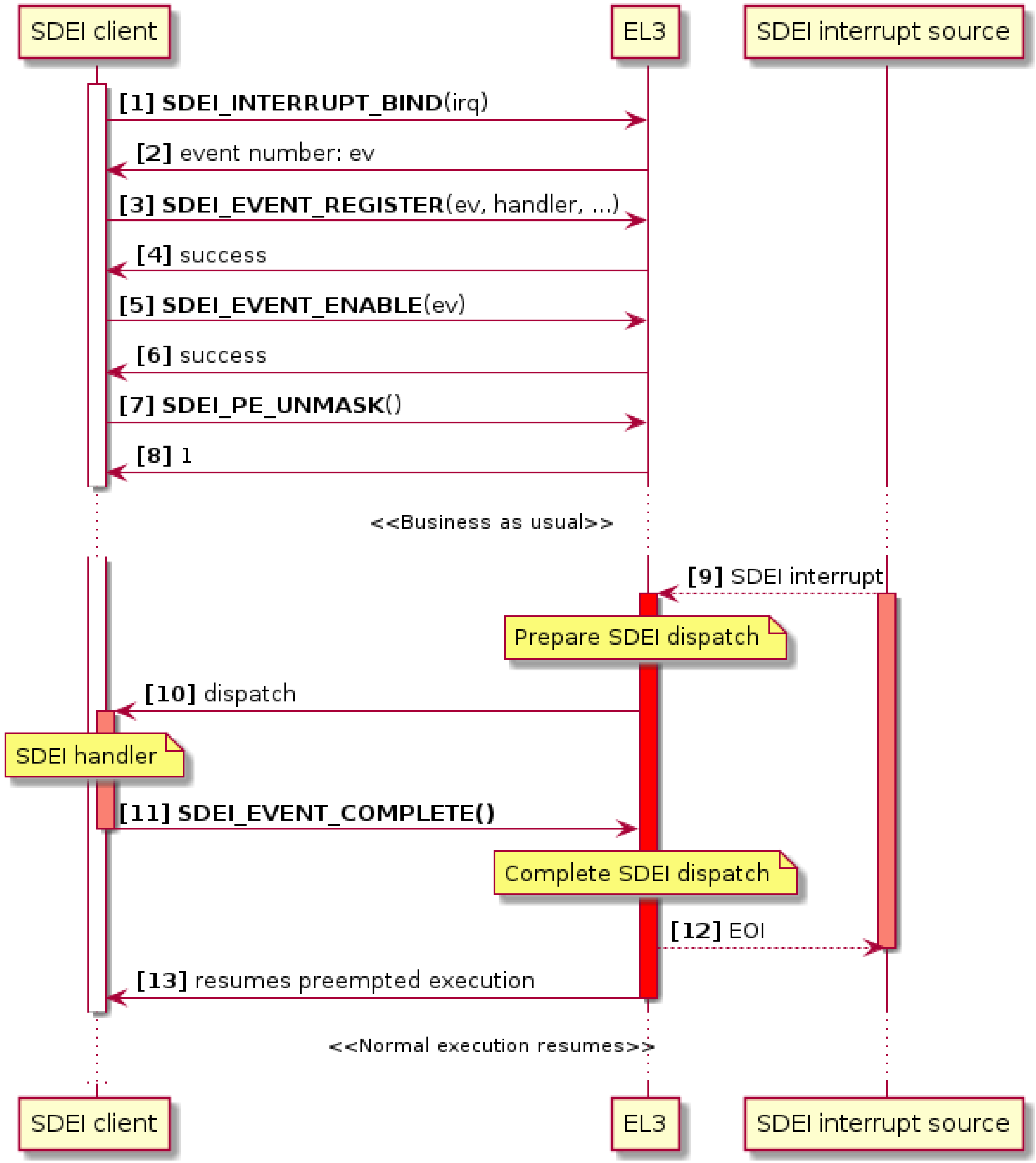
4.14.1 Introduction

Software Delegated Exception Interface (*SDEI*) is an Arm specification for Non-secure world to register handlers with firmware to receive notifications about system events. Firmware will first receive the system events by way of asynchronous exceptions and, in response, arranges for the registered handler to execute in the Non-secure EL.

Normal world software that interacts with the SDEI dispatcher (makes SDEI requests and receives notifications) is referred to as the *SDEI Client*. A client receives the event notification at the registered handler even when it was executing with exceptions masked. The list of SDEI events available to the client are specific to the platform[1]. See also *Determining client EL*.

The following figure depicts a general sequence involving SDEI client executing at EL2 and an event dispatch resulting from the triggering of a bound interrupt. A commentary is provided below:

[1] Except event 0, which is defined by the SDEI specification as a standard event.



As part of initialisation, the SDEI client binds a Non-secure interrupt [1], and the SDEI dispatcher returns a platform dynamic event number [2]. The client then registers a handler for that event [3], enables the event [5], and unmasks all events on the current PE [7]. This sequence is typical of an SDEI client, but it may involve additional SDEI calls.

At a later point in time, when the bound interrupt triggers [9], it's trapped to EL3. The interrupt is handed over to the SDEI dispatcher, which then arranges to execute the registered handler [10]. The client terminates its execution with `SDEI_EVENT_COMPLETE` [11], following which the dispatcher resumes the original EL2 execution [13]. Note that the SDEI interrupt remains active until the client handler completes, at which point

EL3 does EOI [12].

Other than events bound to interrupts, as depicted in the sequence above, SDEI events can be explicitly dispatched in response to other exceptions, for example, upon receiving an *SError* or *Synchronous External Abort*. See *Explicit dispatch of events*.

The remainder of this document only discusses the design and implementation of SDEI dispatcher in TF-A, and assumes that the reader is familiar with the SDEI specification, the interfaces, and their requirements.

4.14.2 Defining events

A platform choosing to include the SDEI dispatcher must also define the events available on the platform, along with their attributes.

The platform is expected to provide two arrays of event descriptors: one for private events, and another for shared events. The SDEI dispatcher provides `SDEI_PRIVATE_EVENT()` and `SDEI_SHARED_EVENT()` macros to populate the event descriptors. Both macros take 3 arguments:

- The event number: this must be a positive 32-bit integer.
- For an event that has a backing interrupt, the interrupt number the event is bound to:
  - If it's not applicable to an event, this shall be left as `0`.
  - If the event is dynamic, this should be specified as `SDEI_DYN_IRQ`.
- A bit map of *Event flags*.

To define event 0, the macro `SDEI_DEFINE_EVENT_0()` should be used. This macro takes only one parameter: an SGI number to signal other PEs.

To define an event that's meant to be explicitly dispatched (i.e., not as a result of receiving an SDEI interrupt), the macro `SDEI_EXPLICIT_EVENT()` should be used. It accepts two parameters:

- The event number (as above);
- Event priority: `SDEI_MAPF_CRITICAL` or `SDEI_MAPF_NORMAL`, as described below.

Once the event descriptor arrays are defined, they should be exported to the SDEI dispatcher using the `REGISTER_SDEI_MAP()` macro, passing it the pointers to the private and shared event descriptor arrays, respectively. Note that the `REGISTER_SDEI_MAP()` macro must be used in the same file where the arrays are defined.

Regarding event descriptors:

- For Event 0:
  - There must be exactly one descriptor in the private array, and none in the shared array.
  - The event should be defined using `SDEI_DEFINE_EVENT_0()`.
  - Must be bound to a Secure SGI on the platform.
- Explicit events should only be used in the private array.
- Statically bound shared and private interrupts must be bound to shared and private interrupts on the platform, respectively. See the section on *Configuration within Exception Handling Framework*.

- Both arrays should be one-dimensional. The `REGISTER_SDEI_MAP()` macro takes care of replicating private events for each PE on the platform.
- Both arrays must be sorted in the increasing order of event number.

The SDEI specification doesn't have provisions for discovery of available events on the platform. The list of events made available to the client, along with their semantics, have to be communicated out of band; for example, through Device Trees or firmware configuration tables.

See also *Event definition example*.

Event flags

Event flags describe the properties of the event. They are bit maps that can be `OR`ed to form parameters to macros that define events (see *Defining events*).

- `SDEI_MAPF_DYNAMIC`: Marks the event as dynamic. Dynamic events can be bound to (or released from) any Non-secure interrupt at runtime via the `SDEI_INTERRUPT_BIND` and `SDEI_INTERRUPT_RELEASE` calls.
- `SDEI_MAPF_BOUND`: Marks the event as statically bound to an interrupt. These events cannot be re-bound at runtime.
- `SDEI_MAPF_NORMAL`: Marks the event as having *Normal* priority. This is the default priority.
- `SDEI_MAPF_CRITICAL`: Marks the event as having *Critical* priority.

4.14.3 Event definition example

```
static sdei_ev_map_t plat_private_sdei[] = {
     /* Event 0 definition */
     SDEI_DEFINE_EVENT_0(8),

     /* PPI */
     SDEI_PRIVATE_EVENT(8, 23, SDEI_MAPF_BOUND),

     /* Dynamic private events */
     SDEI_PRIVATE_EVENT(100, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC),
     SDEI_PRIVATE_EVENT(101, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC)

     /* Events for explicit dispatch */
     SDEI_EXPLICIT_EVENT(2000, SDEI_MAPF_NORMAL);
     SDEI_EXPLICIT_EVENT(2000, SDEI_MAPF_CRITICAL);
};

/* Shared event mappings */
static sdei_ev_map_t plat_shared_sdei[] = {
     SDEI_SHARED_EVENT(804, 0, SDEI_MAPF_DYNAMIC),

     /* Dynamic shared events */
     SDEI_SHARED_EVENT(3000, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC),
```

(continues on next page)

(continued from previous page)

```
    SDEI_SHARED_EVENT(3001, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC)
};

/* Export SDEI events */
REGISTER_SDEI_MAP(plat_private_sdei, plat_shared_sdei);
```

4.14.4 Configuration within Exception Handling Framework

The SDEI dispatcher functions alongside the Exception Handling Framework. This means that the platform must assign priorities to both Normal and Critical SDEI interrupts for the platform:

- Install priority descriptors for Normal and Critical SDEI interrupts.
- For those interrupts that are statically bound (i.e. events defined as having the `SDEI_MAPF_BOUND` property), enumerate their properties for the GIC driver to configure interrupts accordingly.

  The interrupts must be configured to target EL3. This means that they should be configured as *Group 0*. Additionally, on GICv2 systems, the build option `GICV2_G0_FOR_EL3` must be set to `1`.

See also *SDEI porting requirements*.

4.14.5 Determining client EL

The SDEI specification requires that the *physical* SDEI client executes in the highest Non-secure EL implemented on the system. This means that the dispatcher will only allow SDEI calls to be made from:

- EL2, if EL2 is implemented. The Hypervisor is expected to implement a *virtual* SDEI dispatcher to support SDEI clients in Guest Operating Systems executing in Non-secure EL1.
- Non-secure EL1, if EL2 is not implemented or disabled.

See the function `sdei_client_el()` in `sdei_private.h`.

4.14.6 Explicit dispatch of events

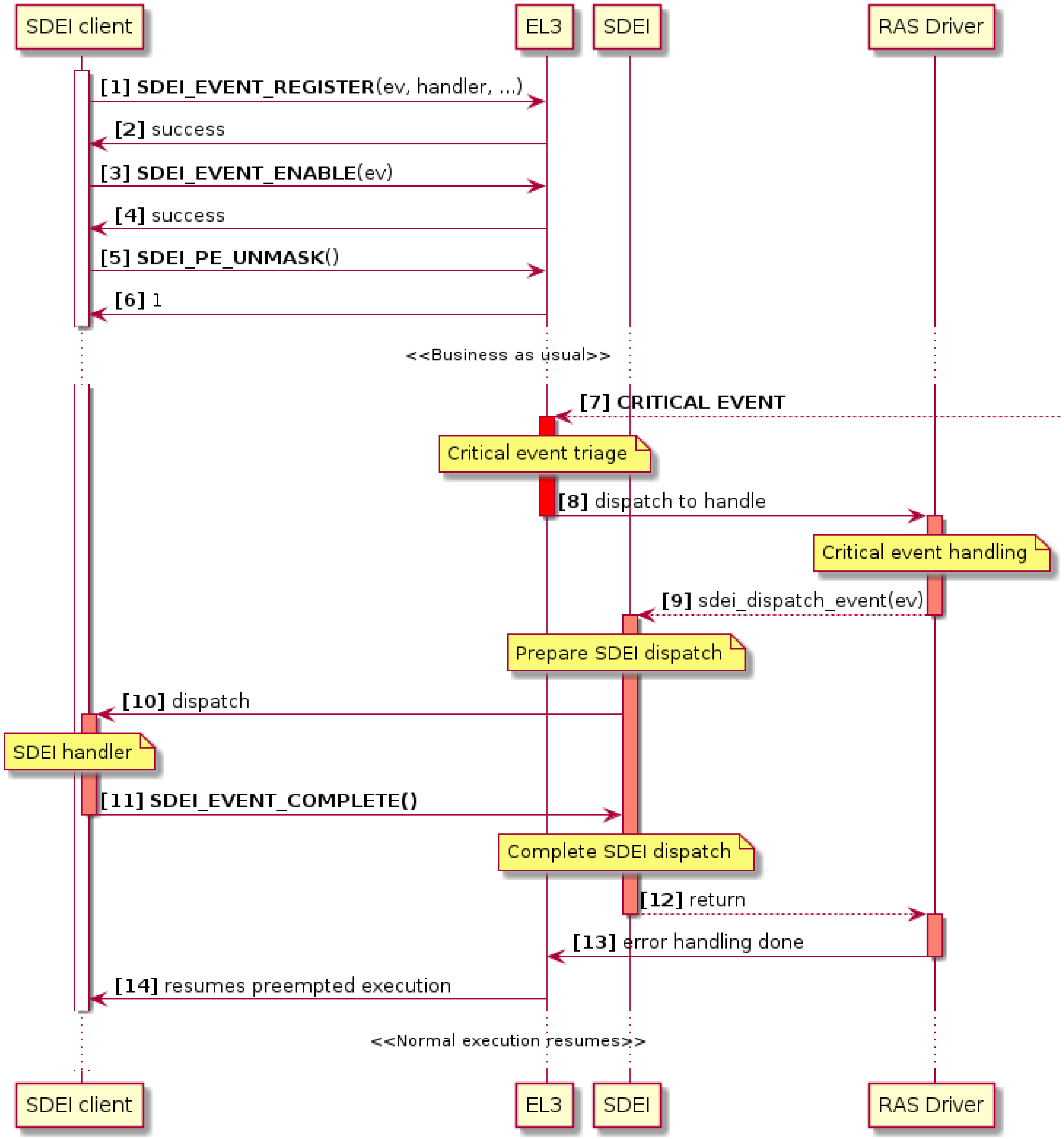
Typically, an SDEI event dispatch is caused by the PE receiving interrupts that are bound to an SDEI event. However, there are cases where the Secure world requires dispatch of an SDEI event as a direct or indirect result of a past activity, such as receiving a Secure interrupt or an exception.

The SDEI dispatcher implementation provides `sdei_dispatch_event()` API for this purpose. The API has the following signature:

```
int sdei_dispatch_event(int ev_num);
```

The parameter `ev_num` is the event number to dispatch. The API returns `0` on success, or `-1` on failure.

The following figure depicts a scenario involving explicit dispatch of SDEI event. A commentary is provided below:



As part of initialisation, the SDEI client registers a handler for a platform event [1], enables the event [3], and unmasks the current PE [5]. Note that, unlike in *general SDEI dispatch*, this doesn't involve interrupt binding, as bound or dynamic events can't be explicitly dispatched (see the section below).

At a later point in time, a critical event[2] is trapped into EL3 [7]. EL3 performs a first-level triage of the event, and a RAS component assumes further handling [8]. The dispatch completes, but intends to involve Non-secure world in further handling, and therefore decides to explicitly dispatch an event [10] (which the client had already registered for [1]). The rest of the sequence is similar to that in the *general SDEI dispatch*: the requested event is dispatched to the client (assuming all the conditions are met), and when the handler completes, the preempted

[2] Examples of critical events are *SError*, *Synchronous External Abort*, *Fault Handling interrupt* or *Error Recovery interrupt* from one of RAS nodes in the system.

execution resumes.

Conditions for event dispatch

All of the following requirements must be met for the API to return `0` and event to be dispatched:

- SDEI events must be unmasked on the PE. I.e. the client must have called `PE_UNMASK` beforehand.
- Event 0 can't be dispatched.
- The event must be declared using the `SDEI_EXPLICIT_EVENT()` macro described above.
- The event must be private to the PE.
- The event must have been registered for and enabled.
- A dispatch for the same event must not be outstanding. I.e. it hasn't already been dispatched and is yet to be completed.
- The priority of the event (either Critical or Normal, as configured by the platform at build-time) shouldn't cause priority inversion. This means:
  - If it's of Normal priority, neither Normal nor Critical priority dispatch must be outstanding on the PE.
  - If it's of a Critical priority, no Critical priority dispatch must be outstanding on the PE.

Further, the caller should be aware of the following assumptions made by the dispatcher:

- The caller of the API is a component running in EL3; for example, a RAS driver.
- The requested dispatch will be permitted by the Exception Handling Framework. I.e. the caller must make sure that the requested dispatch has sufficient priority so as not to cause priority level inversion within Exception Handling Framework.
- The caller must be prepared for the SDEI dispatcher to restore the Non-secure context, and mark that the active context.
- The call will block until the SDEI client completes the event (i.e. when the client calls either `SDEI_EVENT_COMPLETE` or `SDEI_COMPLETE_AND_RESUME`).
- The caller must be prepared for this API to return failure and handle accordingly.

4.14.7 Porting requirements

The porting requirements of the SDEI dispatcher are outlined in the *Porting Guide*.

4.14.8 Note on writing SDEI event handlers

This section pertains to SDEI event handlers in general, not just when using the TF-A SDEI dispatcher.

The SDEI specification requires that event handlers preserve the contents of all registers except `x0` to `x17`. This has significance if event handler is written in C: compilers typically adjust the stack frame at the beginning and end of C functions. For example, AArch64 GCC typically produces the following function prologue and epilogue:

```
c_event_handler:
        stp     x29, x30, [sp,#-32]!
        mov     x29, sp

        ...

        bl      ...

        ...

        ldp     x29, x30, [sp],#32
        ret
```

The register `x29` is used as frame pointer in the prologue. Because neither a valid `SDEI_EVENT_COMPLETE` nor `SDEI_EVENT_COMPLETE_AND_RESUME` calls return to the handler, the epilogue never gets executed, and registers `x29` and `x30` (in the case above) are inadvertently corrupted. This violates the SDEI specification, and the normal execution thereafter will result in unexpected behaviour.

To work this around, it's advised that the top-level event handlers are implemented in assembly, following a similar pattern as below:

```
asm_event_handler:
        /* Save link register whilst maintaining stack alignment */
        stp     xzr, x30, [sp, #-16]!
        bl      c_event_handler

        /* Restore link register */
        ldp     xzr, x30, [sp], #16

        /* Complete call */
        ldr     x0, =SDEI_EVENT_COMPLETE
        smc     #0
        b       .
```

4.14.9 Security Considerations

SDEI introduces concept of providing software based non-maskable interrupts to Hypervisor/OS. In doing so, it modifies the priority scheme defined by Interrupt controllers and relies on Non-Secure clients, Hypervisor or OS, to create/manage high priority events.

Considering a Non-secure client is involved in SDEI state management, there exists some security considerations which needs to be taken care of in both client and EL3 when using SDEI. Few of them are mentioned below.

Bound events

A bound event is an SDEI event that corresponds to a client interrupt. The binding of event is done using `SDEI_INTERRUPT_BIND` SMC call to associate an SDEI event with a client interrupt. There is a possibility that a rogue client can request an invalid interrupt to be bound. This may potentially cause out-of-bound memory read.

Even though TF-A implementation has checks to ensure that interrupt ID passed by client is architecturally valid, Non-secure client should also ensure the validity of interrupts.

Recurring events

For a given event source, if the events are generated continuously, then NS client may be unusable. To mitigate against this, the Non-secure client must have mechanism in place to remove such interrupt source from the system.

One of the examples is a memory region which continuously generates RAS errors. This may result in unusable Non-secure client.

Dispatched events

For a dispatched event, it is the client's responsibility to ensure that the handling finishes in finite time and notify the dispatcher through `SDEI_EVENT_COMPLETE` or `SDEI_EVENT_COMPLETE_AND_RESUME`. If the client fails to complete the event handling, it might result in `UNPREDICTABLE` behavior in the client and potentially end up in unusable PE.

4.15 Secure Partition Manager

Contents

- *Secure Partition Manager*
  - *Acronyms*

- *Foreword*
  - *Support for legacy platforms*
- *TF-A build options*
- *Boot process*
  - *SPMC boot*
- *Support for Secure Partition live activation*
  - *Live Firmware Activation LSP responsibilities*
  - *Reference implementation*
- *References*

4.15.1 FF-A manifests binding to device tree

This document defines the nodes and properties used to define a partition, according to the FF-A specification, and the SPMC manifest.

FF-A Partition Manifest Properties

The FF-A partition manifest is consumed by the SPMC to configure the state associated with the related Secure Partition.

- **compatible [mandatory]**
  - value type: <string>
  - Must be the string "arm,ffa-manifest-X.Y" which specifies the major and minor versions of the device tree binding for the FFA manifest represented by this node. The minor number is incremented if the binding changes in a backwards compatible manner.
    - X is an integer representing the major version number of this document.
    - Y is an integer representing the minor version number of this document.
- **ffa-version [mandatory]**
  - value type: <u32>
  - Must be two 16 bits values (X, Y), concatenated as 31:16 -> X, 15:0 -> Y, where:
    - X is the major version of FF-A expected by the partition at the FFA instance it will execute.
    - Y is the minor version of FF-A expected by the partition at the FFA instance it will execute.
- **id**
  - value type: <u32>

  - Pre-allocated partition ID.
- **auxiliary-id**
  - value type: <u32>
  - Pre-allocated ID that could be used in memory management transactions.
- **description**
  - value type: <string>
  - Name of the partition e.g. for debugging purposes.
- **execution-ctx-count [mandatory]**
  - value type: <u32>
  - Number of vCPUs that a VM or SP wants to instantiate.
    * In the absence of virtualization, this is the number of execution contexts that a partition implements.
    * If value of this field = 1 and number of PEs > 1 then the partition is treated as UP & migrate capable.
    * If the value of this field > 1 then the partition is treated as a MP capable partition irrespective of the number of PEs.
- **exception-level [mandatory]**
  - value type: <u32>
  - The target exception level for the partition:
    * 0x0: EL1
    * 0x1: S_EL0
    * 0x2: S_EL1
- **execution-state [mandatory]**
  - value type: <u32>
  - The target execution state of the partition:
    * 0: AArch64
    * 1: AArch32
- **load-address**
  - value type: <u64>
  - Physical base address of the partition in memory. Absence of this field indicates that the partition is position independent and can be loaded at any address chosen at boot time.
- **entrypoint-offset**
  - value type: <u64>

  - Offset from the base of the partition's binary image to the entry point of the partition. Absence of this field indicates that the entry point is at offset 0x0 from the base of the partition's binary.
- **xlat-granule**
  - value type: <u32>
  - Translation granule used with the partition:
    * 0x0: 4k
    * 0x1: 16k
    * 0x2: 64k
- **boot-order**
  - value type: <u32>
  - A unique number amongst all partitions that specifies if this partition must be booted before others. The partition with the smaller number will be booted first. Highest vlue allowed for this field is 0xFFFF.
- **rx-tx-buffer**
  - value type: "memory-regions" node
  - Specific "memory-regions" nodes that describe the RX/TX buffers expected by the partition. The "compatible" must be the string "arm,ffa-manifest-rx_tx-buffer".
- **managed-exit**
  - value type: <empty>
  - Specifies if managed exit is supported.
  - This field is deprecated in favor of ns-interrupts-action field in the FF-A v1.1 EAC0 spec.
- **managed-exit-virq**
  - value type: <empty>
  - Indicates if the partition needs managed exit, if supported, to be signaled through vIRQ signal.
- **ns-interrupts-action [mandatory]**
  - value type: <u32>
  - Specifies the action that the SPMC must take in response to a Non-secure physical interrupt.
    * 0x0: Non-secure interrupt is queued
    * 0x1: Non-secure interrupt is signaled after a managed exit
    * 0x2: Non-secure interrupt is signaled
  - This field supersedes the managed-exit field in the FF-A v1.0 spec.
- **other-s-interrupts-action**
  - value type: <u32>

   - Specifies the action that the SPMC must take in response to a Other-Secure physical interrupt.
     * 0x0: Other-Secure interrupt is queued
     * 0x1: Other-Secure interrupt is signaled

- sri-interrupts-policy
  - value type: <u32>
  - Specifies how secure interrupts are handled when the SP is in a waiting state and is targeted by a secure interrupt, or when the SP attempts to return to a waiting state with pending secure interrupts. The value is a bitfield.
    * 0x0: Proactively inject the VI and resume SP when handling a secure interrupt and SP in the waiting state.
    * 0x1: Only when a secure interrupt is fired while target SP in the waiting state, pend SRI to the NWd and rely on the scheduler to explicitly donate CPU cycles to the SP.
    * 0x2: Only when the SP attempts to go back to the waiting state while having pending secure interrupts, trigger the SRI to the NWd and rely on the scheduler to explicitly donate CPU cycles to the SP.
    * 0x3: Enable both actions for values 0x1 and 0x2.

- **has-primary-scheduler**
  - value type: <empty>
  - Presence of this field indicates that the partition implements the primary scheduler. If so, run-time EL must be EL1.

- **time-slice-mem**
  - value type: <empty>
  - Presence of this field indicates that the partition doesn't expect the partition manager to time slice long running memory management functions.

- **gp-register-num**
  - value type: <u32>
  - The field specifies the general purpose register number but not its width. The width is derived from the partition's execution state, as specified in the partition properties. For example, if the number value is 1 then the general-purpose register used will be x1 in AArch64 state and w1 in AArch32 state. Presence of this field indicates that the partition expects the address of the FF-A boot information blob to be passed in the specified general purpose register.

- **power-management-messages**
  - value type: <u32>
  - Specifies which power management messages a partition subscribes to. A set bit means the partition should be informed of the power event, clear bit - should not be informed of event:
    * Bit[0]: CPU_OFF

    * Bit[1]: CPU_SUSPEND
    * Bit[2]: CPU_SUSPEND_RESUME

- **vm-availability-messages**
  - value type: <u32>
  - Specifies which VM availability messages a partition subscribes to. A set bit means the partition should be informed of the event, clear bit - should not be informed of event:
    * Bit[0]: VM created
    * Bit[1]: VM destroyed
- **lifecycle-support**
  - value type: <empty>
  - Presence of this field indicates support for all partition lifecycle states defined in the FF-A v1.3 ALP2 spec.
- **abort-action**
  - value type: <u32>
  - Specifies the action that the SPMC takes when a partition encounters a fatal error.
    * 0x0: STOP
    * 0x1: DESTROY
    * 0x2: RESTART
    * 0x3: PROPAGATE
  - All other values are unsupported. If a partition does not specify this field in the manifest, the SPMC takes implementation defined action.
- **live-activation-support**
  - value type: <empty>
  - Presence of this field indicates support for partition framework messages necessary for live activation as defined in the FF-A v1.3 ALP2 spec.
- **live-activation-register**
  - value type: <u32>
  - Register used by the SPMC to report to the SP that it is undergoing live activation. Must be one of `x0`–`x3` or `x5`–`x7` so that it does not clash with the boot-info register or the reserved `x4` linear vCPU index.
- **image-uuid**
  - value type: <prop-encoded-array>
  - Exactly one tuple of 4 <u32> values identifying the partition image UUID. The UUID must not be all zeros and must not collide with any protocol UUID exposed by the partition.

- **live-state-buffer-info**
  - value type: <phandle>
  - Child node with `compatible = "arm,ffa-manifest,live-state-buffer"` whose `live-state-buffer` property points to the backing memory-region entry.

When `live-activation-support` is present the manifest must also meet the following requirements:

- `lifecycle-support` must be present and `execution-ctx-count` must be 1.
- `live-activation-register` and `image-uuid` must follow the constraints above; manifests advertising multiple image UUIDs or invalid registers are rejected.
- The live state buffer referenced via `live-state-buffer-info` must be defined under `memory-regions` and describe a page-aligned, read/write region that sits outside the working memory region (which is overwritten during activation) and is large enough to preserve the framework state.

Informative example:

```
lifecycle-support;
execution-ctx-count = <1>;
abort-action = <1>;
live-activation-support;
live-activation-register = <1>;
image-uuid = <0x962a7bf0 0x174d471d 0xa686c89e 0x5c3e254e>;

live-state-buffer-info {
    compatible = "arm,ffa-manifest,live-state-buffer";
    live-state-buffer = <&livestatebuffer>;
};

memory-regions {
    compatible = "arm,ffa-manifest-memory-regions";

    livestatebuffer: live-state-buffer {
        description = "live-state-buffer";
        base-address = <0x00000000 0x06780000>;
        pages-count = <1>;
        attributes = <0x3>; /* read-write */
    };
};
```

Services

- A service pairs a protocol UUID with supported messaging methods that clients use to reach the SP.
- For the compatible value at the root set to “arm,ffa-manifest-1.0”, the UUIDs and messaging methods are specified in two separate lists at the root node as below:
  - **uuid [mandatory]**
    - value type: <prop-encoded-array>

        * An array of comma separated tuples each consisting of 4 <u32> values, identifying the protocol UUIDs of the services implemented by this partition. The UUID format is described in RFC 4122.
        * These 4 <u32> values are packed similar to the UUID register mapping specified in section '5.3 Unique Identification format', SMC Calling Convention, DEN0028, v1.6 G BET0 (https://developer.arm.com/documentation/den0028/latest/).

    - **messaging-method [mandatory]**
        * value type: <prop-encoded-array>
        * An array of <u32> values which specify which messaging methods are supported for each service. If one value is specified all services support those messaging methods. If multiple values are specified, there must be the same number as UUIDs specified, and the messaging method at each index is paired with the UUID at the matching index of the uuid array. Set bit means the feature is supported, clear bit - not supported:
            · Bit[0]: partition can receive direct requests via FFA_MSG_SEND_DIRECT_REQ ABI if set
            · Bit[1]: partition can send direct requests via FFA_MSG_SEND_DIRECT_REQ ABI if set
            · Bit[2]: partition can send and receive indirect messages
            · Bit[9]: partition can receive direct requests via FFA_MSG_SEND_DIRECT_REQ2 ABI if set
            · Bit[10]: partition can send direct requests via FFA_MSG_SEND_DIRECT_REQ2 ABI if set
        * A request targeting a specific protocol UUID requires that service to have the corresponding bit set. Requests that do not target a specific protocol UUID (for example, a NULL UUID or FFA_MSG_SEND_DIRECT_REQ) are allowed if any service exposes the required messaging method.

- For compatible value "arm,ffa-manifest-X.Y" with X.Y > 1.0, specify services under a parent "services" node like so:
    - **services [mandatory]**
        * **compatible [mandatory]**
            · value type: <string>
            · Must be the string "arm,ffa-manifest-services".
        * **Child service nodes [mandatory]**
            · **uuid [mandatory]**
                · value type: <string>
                · A protocol UUID specified in the canonical string format as described in RFC 9562. (https://www.rfc-editor.org/rfc/rfc9562.html#name-uuid-format)
            · **messaging-method [mandatory]**

- value type: <u32>
- Specifies which messaging methods are supported by this service, set bit means the feature is supported, clear bit - not supported:
  - Bit[0]: partition can receive direct requests via FFA_MSG_SEND_DIRECT_REQ ABI if set
  - Bit[1]: partition can send direct requests via FFA_MSG_SEND_DIRECT_REQ ABI if set
  - Bit[2]: partition can send and receive indirect messages
  - Bit[9]: partition can receive direct requests via FFA_MSG_SEND_DIRECT_REQ2 ABI if set
  - Bit[10]: partition can send direct requests via FFA_MSG_SEND_DIRECT_REQ2 ABI if set
- Each child under "services" pairs a protocol UUID with a messaging method. At least one child must be specified.
- A request targeting a specific protocol UUID requires that service to have the corresponding bit set. Requests that do not target a specific protocol UUID (for example, a NULL UUID or FFA_MSG_SEND_DIRECT_REQ) are allowed if any service exposes the required messaging method.

Memory Regions

- **compatible [mandatory]**
  - value type: <string>
  - Must be the string "arm,ffa-manifest-memory-regions".
- **description**
  - value type: <string>
  - Name of the memory region e.g. for debugging purposes.
- **pages-count [mandatory]**
  - value type: <u32>
  - Count of pages of memory region as a multiple of the translation granule size
- **attributes [mandatory]**
  - value type: <u32>
  - Mapping modes: ORed to get required permission
    - 0x1: Read
    - 0x2: Write
    - 0x4: Execute

    * 0x8: Security state

- **base-address**
  - value type: <u64>
  - Base address of the region. The address must be aligned to the translation granule size. The address given may be a Physical Address (PA), Virtual Address (VA), or Intermediate Physical Address (IPA). Refer to the FF-A specification for more information on the restrictions around the address type. If the base address is omitted then the partition manager must map a memory region of the specified size into the partition's translation regime and then communicate the region properties (including the base address chosen by the partition manager) to the partition.
- **load-address-relative-offset**
  - value type: <u64>
  - Offset relative to the load address of the partition. When this is provided in the partition manifest, it should be added to the load address to get the base address of the region. The secure partition manifest can have either "base-address" or "load-address-relative-offset". It cannot have both.
- **stream-ids**
  - value type: <prop-encoded-array>
  - List of IDs belonging to a DMA capable peripheral device that has access to the memory region represented by current node.
  - Each ID must have been declared in exactly one device region node.
- **smmu-id**
  - value type: <u32>
  - Identifies the SMMU IP that enforces the access control for the DMA device that owns the above stream-ids.
- **stream-ids-access-permissions**
  - value type: <prop-encoded-array>
  - List of attributes representing the instruction and data access permissions used by the DMA device streams to access the memory region represented by current node.

Device Regions

- **compatible [mandatory]**
  - value type: <string>
  - Must be the string "arm,ffa-manifest-device-regions".
- **description**
  - value type: <string>

  - Name of the device region e.g. for debugging purposes.

- **pages-count [mandatory]**
  - value type: <u32>
  - Count of pages of memory region as a multiple of the translation granule size
- **attributes [mandatory]**
  - value type: <u32>
  - Mapping modes: ORed to get required permission
    - 0x1: Read
    - 0x2: Write
    - 0x4: Execute
    - 0x8: Security state
- **base-address [mandatory]**
  - value type: <u64>
  - Base address of the region. The address must be aligned to the translation granule size. The address given may be a Physical Address (PA), Virtual Address (VA), or Intermediate Physical Address (IPA). Refer to the FF-A specification for more information on the restrictions around the address type.
- **smmu-id**
  - value type: <u32>
  - On systems with multiple System Memory Management Units (SMMUs) this identifier is used to inform the partition manager which SMMU the device is upstream of. If the field is omitted then it is assumed that the device is not upstream of any SMMU.
- **stream-ids**
  - value type: <prop-encoded-array>
  - List of IDs where an ID is a unique <u32> value amongst all devices assigned to the partition.
- **interrupts**
  - value type: <prop-encoded-array>
  - A list of (id, attributes) pair describing the device interrupts, where:
    - id: The <u32> interrupt IDs.
    - attributes: A <u32> value, containing attributes for each interrupt ID:

Field	Bit(s)
Priority	7:0
Security state	8
Config(Edge/Level)	9
Type(SPI/PPI/SGI)	11:10

Security state:

- Secure: 1
- Non-secure: 0

Configuration:

- Edge triggered: 0
- Level triggered: 1

Type:

- SPI: 0b10
- PPI: 0b01
- SGI: 0b00

- **interrupts-target**
  - value type: <prop-encoded-array>
  - A list of (id, mpdir upper bits, mpidr lower bits) tuples describing which mpidr the interrupt is routed to, where:
    - **id: The <u32> interrupt ID. Must be one of those specified in the** "interrupts" field.
    - **mpidr upper bits: The <u32> describing the upper bits of the 64 bits** mpidr
    - **mpidr lower bits: The <u32> describing the lower bits of the 64 bits** mpidr
- **exclusive-access**
  - value type: <empty>
  - Presence of this field implies that this endpoint must be granted exclusive access and ownership of this device's MMIO region.

SPMC Manifest Properties

This manifest contains the SPMC *attribute* node consumed by the SPMD at boot time.

attribute

- **spmc_id**
  - value type: <u32>
  - Defines the endpoint ID value that SPMC can query through `FFA_ID_GET`.
- **maj_ver**
  - value type: <u32>
  - Major of the FF-A version implemented by the SPMC. SPMD checks against its own version.
- **min_ver**
  - value type>: <u32>
  - Minor of the FF-A version implemented by the SPMC. SPMD checks against its own version.
- **exec_state**
  - value type: <u32>
  - Defines the SPMC execution state (AArch64 or AArch32).
- **load_address**
  - value type: <u64>
  - Base physical address in which the SPMC binary is placed. Should be page aligned.
- **entrypoint:**
  - value type: <u64>
  - Defines the physical address for the cold boot primary core entrypoint used by the SPMD (currently matches `BL32_BASE`) to enter the SPMC.
- **binary_size**
  - value type: <u32>
  - Defines the maximum size of the SPMC binary. It is used with load_address to sanitize the specified entrypoint.

4.15.2 Acronyms

DTS	Device Tree Source
FF-A	Firmware Framework for Arm A-profile
NWd	Normal World
SP	Secure Partition
SPD	Secure Payload Dispatcher
SPM	Secure Partition Manager
SPMC	SPM Core
SPMD	SPM Dispatcher
SWd	Secure World

4.15.3 Foreword

Three implementations of a Secure Partition Manager co-exist in the TF-A codebase:

1. S-EL2 SPMC based on the FF-A specification *[1]*, enabling virtualization in the secure world, managing multiple S-EL1 or S-EL0 partitions *[5]*.
2. EL3 SPMC based on the FF-A specification, managing a single S-EL1 partition without virtualization in the secure world *[6]*.
3. EL3 SPM based on the MM specification, legacy implementation managing a single S-EL0 partition *[2]*.

These implementations differ in their respective SW architecture and only one can be selected at build time.

Support for legacy platforms

The SPM is split into a dispatcher and a core component (respectively SPMD and SPMC) residing at different exception levels. To permit the FF-A specification adoption and a smooth migration, the SPMD supports an SPMC residing either at S-EL1 or S-EL2:

- The SPMD is located at EL3 and mainly relays the FF-A protocol from NWd (Hypervisor or OS kernel) to the SPMC.
- The same SPMD component is used for both S-EL1 and S-EL2 SPMC configurations.
- The SPMC exception level is a build time choice.

TF-A supports both cases:

- S-EL1 SPMC for platforms not supporting the FEAT_SEL2 architecture extension. The SPMD relays the FF-A protocol from EL3 to S-EL1.
- S-EL2 SPMC for platforms implementing the FEAT_SEL2 architecture extension. The SPMD relays the FF-A protocol from EL3 to S-EL2.

4.15.4 TF-A build options

This section explains the TF-A build options involved in building with support for an FF-A based SPM where the SPMD is located at EL3 and the SPMC located at S-EL1, S-EL2 or EL3:

- **SPD=spmd**: this option selects the SPMD component to relay the FF-A protocol from NWd to SWd back and forth. It is not possible to enable another Secure Payload Dispatcher when this option is chosen.
- **SPMD_SPM_AT_SEL2**: this option adjusts the SPMC exception level to being at S-EL2. It defaults to enabled (value 1) when SPD=spmd is chosen.
- **SPMC_AT_EL3**: this option adjusts the SPMC exception level to being at EL3. If neither `SPMD_SPM_AT_SEL2` or `SPMC_AT_EL3` are enabled the SPMC exception level is set to S-EL1. `SPMD_SPM_AT_SEL2` is enabled. The context save/restore routine and exhaustive list of registers is visible at *[4]*.
- **SPMC_AT_EL3_SEL0_SP**: this option enables the support to load SEL0 SP when SPMC at EL3 support is enabled.
- **SP_LAYOUT_FILE**: this option specifies a text description file providing paths to SP binary images and manifests in DTS format (see *[3]*). It is required when `SPMD_SPM_AT_SEL2` is enabled hence when multiple secure partitions are to be loaded by BL2 on behalf of the SPMC.

	SPMD_SPM_AT_SEL2	SPMC_AT_EL3	CTX_INCLUDE_EL2_REGS(*)
SPMC at S-EL1	0	0	0
SPMC at S-EL2	**1 (default when** SPD=spmd)	0	1
SPMC at EL3	0	1	0

Other combinations of such build options either break the build or are not supported.

Notes:

- Only Arm's FVP platform is supported to use with the TF-A reference software stack.
- When `SPMD_SPM_AT_SEL2=1`, the reference software stack assumes enablement of FEAT_PAuth, FEAT_BTI and FEAT_MTE2 architecture extensions.
- `(*) CTX_INCLUDE_EL2_REGS`, this flag is *TF-A* internal and informational in this table. When set, it provides the generic support for saving/restoring EL2 registers required when S-EL2 firmware is present.
- BL32 option is re-purposed to specify the SPMC image. It can specify either the Hafnium binary path (built for the secure world) or the path to a TEE binary implementing FF-A interfaces.
- BL33 option can specify the TFTF binary or a normal world loader such as U-Boot or the UEFI framework payload.

Sample TF-A build command line when the SPMC is located at S-EL1 (e.g. when the FEAT_SEL2 architecture extension is not implemented):

```
make \
CROSS_COMPILE=aarch64-none-elf- \
SPD=spmd \
SPMD_SPM_AT_SEL2=0 \
BL32=<path-to-tee-binary> \
BL33=<path-to-bl33-binary> \
PLAT=fvp \
all fip
```

Sample TF-A build command line when FEAT_SEL2 architecture extension is implemented and the SPMC is located at S-EL2:

```
make \
CROSS_COMPILE=aarch64-none-elf- \
PLAT=fvp \
SPD=spmd \
ARM_ARCH_MINOR=5 \
BRANCH_PROTECTION=1 \
CTX_INCLUDE_PAUTH_REGS=1 \
ENABLE_FEAT_MTE2=1 \
BL32=<path-to-hafnium-binary> \
BL33=<path-to-bl33-binary> \
SP_LAYOUT_FILE=sp_layout.json \
all fip
```

Sample TF-A build command line when FEAT_SEL2 architecture extension is implemented, the SPMC is located at S-EL2, and enabling secure boot:

```
make \
CROSS_COMPILE=aarch64-none-elf- \
PLAT=fvp \
SPD=spmd \
ARM_ARCH_MINOR=5 \
BRANCH_PROTECTION=1 \
CTX_INCLUDE_PAUTH_REGS=1 \
ENABLE_FEAT_MTE2=1 \
BL32=<path-to-hafnium-binary> \
BL33=<path-to-bl33-binary> \
SP_LAYOUT_FILE=sp_layout.json \
TRUSTED_BOARD_BOOT=1 \
COT=dualroot \
ARM_ROTPK_LOCATION=devel_rsa \
ROT_KEY=plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem \
GENERATE_COT=1 \
all fip
```

Sample TF-A build command line when the SPMC is located at EL3:

```
make \
CROSS_COMPILE=aarch64-none-elf- \
SPD=spmd \
SPMD_SPM_AT_SEL2=0 \
```

(continues on next page)

(continued from previous page)

```
SPMC_AT_EL3=1 \
BL32=<path-to-tee-binary> \
BL33=<path-to-bl33-binary> \
PLAT=fvp \
all fip
```

Sample TF-A build command line when the SPMC is located at EL3 and SEL0 SP is enabled:

```
make \
CROSS_COMPILE=aarch64-none-elf- \
SPD=spmd \
SPMD_SPM_AT_SEL2=0 \
SPMC_AT_EL3=1 \
SPMC_AT_EL3_SEL0_SP=1 \
BL32=<path-to-tee-binary> \
BL33=<path-to-bl33-binary> \
PLAT=fvp \
all fip
```

4.15.5 Boot process

The boot process involving SPMC is highly dependent on the SPMC implementation. It is recommended to refer to corresponding SPMC documentation for further details. Some aspects of boot process are described here in the greater interest of the project.

SPMC boot

When SPMC resides at a lower EL i.e., S-EL1 or S-EL2, it is loaded by BL2 as the BL32 image. The SPMC manifest is loaded by BL2 as the `TOS_FW_CONFIG` image *[7]*.

BL2 passes the SPMC manifest address to BL31 through a register. At boot time, the SPMD in BL31 runs from the primary core, initializes the core contexts and launches the SPMC (BL32) passing the following information through registers:

- X0 holds the `TOS_FW_CONFIG` physical address (or SPMC manifest blob).
- X1 holds the `HW_CONFIG` physical address.
- X4 holds the currently running core linear id.

4.15.6 Support for Secure Partition live activation

Live firmware activation, defined in the LFA spec *[9]*, is implemented by the agent framework in TF-A BL31. The framework handles the LFA SMCs and expects each platform to describe every activatable firmware component (BL31, RMM, Secure Partitions, etc.) through a set of callbacks. Secure Partitions (SPs) require additional oversight because they are managed by the Secure Partition Manager Core (SPMC) through FF-A.

Consequently, TF-A shall implement Logical Secure Partition (LSP) managed by SPMD at EL3 that bridges the LFA calls to FF-A framework messages understood by the SPMC. Note that support for live activation is limited to v1.3 compliant Secure Partitions with single execution context and managed by S-EL2 SPMC.

A typical Secure Partition is highly platform specific given the way it is integrated by an OEM; every platform decides which SP packages are bundled in the FIP, how and where the new SP images are staged and so on. Therefore, the LSP in TF-A project needs to be independently implemented by the platform port based on the guidance in FF-A specification. TF-A code base supplies reusable helpers in `services/std_svc/spmd/spmd_logical_sp.c`.

Live Firmware Activation LSP responsibilities

An LSP must implement the `prime` and `activate` callbacks for the LFA framework to execute SP live activation. At a minimum, an LSP must:

- Implement the platform-specific steps needed to stage a new partition package for SP.
- Perform partition discovery and ensure the partition properties are suitable for live activation.
- Translate `LFA_ACTIVATE` into the FF-A framework messages defined by FF-A v1.3 ALP2: `FFA_FRAMEWORK_MSG_LIVE_ACTIVATION_START_REQ` followed by `FFA_FRAMEWORK_MSG_LIVE_ACTIVATION_FINISH_REQ`. The SPMC at S-EL2 then performs the in-place activation and responds with SUCCESS status code to the corresponding framework messages.

Reference implementation

The Arm FVP port implements all of the above in:

- `plat/arm/board/fvp/fvp_lfa.c` – enumerates components and ties them to LFA.
- `plat/arm/board/fvp/fvp_spmd_logical_sp.c` – provides the logical partition, prime/activate callbacks, and staging logic.
- `services/std_svc/spmd/spmd_logical_sp.c` – common helpers that build and parse the FF-A framework messages on behalf of the platform LSP.

Platforms that wish to add live activation for their own SPs can use the FVP code as a template, customize the staging strategy and component table, and ensure that the SP manifests include the live activation properties documented in the FF-A manifest binding (*docs/components/ffa-manifest-binding.rst*). Secure Partition live activation is guarded by few build options *[8]*.

4.15.7 References

[1] Arm Firmware Framework for Arm A-profile

[2] *Secure Partition Manager using MM interface*

[3] https://hafnium.readthedocs.io/en/latest/secure-partition-manager/secure-partition-manager.html#secure-partitions-layout-file

[4] https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/tree/lib/el3_runtime/aarch64/context.S#n45

[5] https://hafnium.readthedocs.io/en/latest/secure-partition-manager/index.html

[6] *EL3 Secure Partition Manager*

[7] https://trustedfirmware-a.readthedocs.io/en/latest/design/firmware-design.html#dynamic-configuration-during-cold-boot

[8] *SP Live Activation build options*

[9] https://developer.arm.com/documentation/den0147/1-0bet1/?lang=en

---

4.16 EL3 Secure Partition Manager

Contents

- *EL3 Secure Partition Manager*
  - *Foreword*
  - *Sample reference stack*
  - *TF-A build options*
  - *FVP model invocation*
  - *Platform Guide*
  - *Logical Secure Partition (LSP)*
  - *SPMC boot*
    - *Parsing SP partition manifests*
    - *Passing boot data to the SP*
  - *Supported interfaces*
    - *FFA_VERSION*
    - *FFA_FEATURES*

    * *FFA_RXTX_MAP*
    * *FFA_RXTX_UNMAP*
    * *FFA_PARTITION_INFO_GET*
    * *FFA_ID_GET*
    * *FFA_MSG_SEND_DIRECT_REQ*
    * *FFA_MSG_SEND_DIRECT_RESP*
    * *FFA_SPM_ID_GET*
    * *FFA_ID_GET*
    * *FFA_MEM_SHARE*
    * *FFA_MEM_LEND*
    * *FFA_MEM_RETRIEVE_REQ*
    * *FFA_MEM_RETRIEVE_RESP*
    * *FFA_MEM_FRAG_RX*
    * *FFA_MEM_FRAG_TX*
    * *FFA_SECONDARY_EP_REGISTER*
- *Power management*
- *Secure partitions scheduling*
- *Partition Runtime State and Model*
- *Platform topology*
- *Interrupt handling*
    * *Secure Interrupt handling*
    * *Non-Secure Interrupt handling*
- *Test Secure Payload (TSP)*
    * *TSP Tests in CI*
- *References*

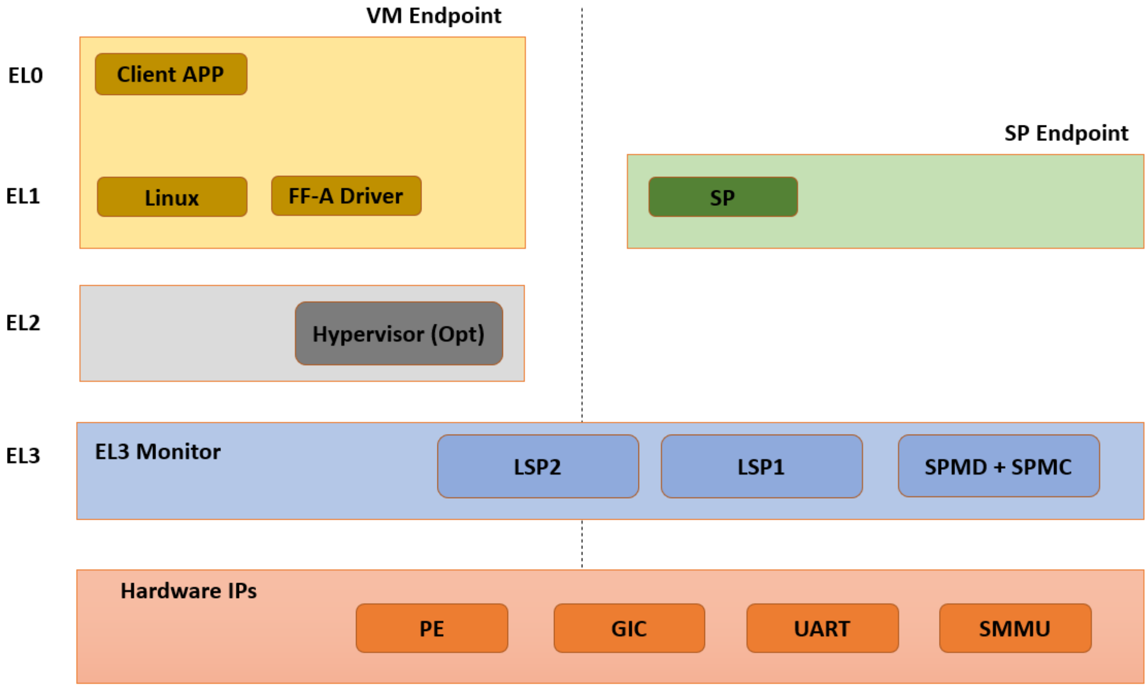
4.16.1 Foreword

This document describes the design of the EL3 SPMC based on the FF-A specification. EL3 SPMC provides reference FF-A compliant implementation without S-EL2 virtualization support, to help adopt and migrate to FF-A early. EL3 SPMC implementation in TF-A:

- Manages a single S-EL1 Secure Partition
- Provides a standard protocol for communication and memory sharing between FF-A endpoints.
- Provides support for EL3 Logical Partitions to support easy migration from EL3 to S-EL1.

4.16.2 Sample reference stack

The following diagram illustrates a possible configuration when the FEAT_SEL2 architecture extension is not implemented, showing the SPMD and SPMC at EL3, one S-EL1 secure partition, with an optional Hypervisor:



4.16.3 TF-A build options

This section explains the TF-A build options involved in building an FF-A based SPM where the SPMD and SPMC are located at EL3:

- **SPD=spmd**: this option selects the SPMD component to relay the FF-A protocol from NWd to SWd back and forth. It is not possible to enable another Secure Payload Dispatcher when this option is chosen.
- **SPMC_AT_EL3**: this option adjusts the SPMC exception level to being at EL3.

- **ARM_SPMC_MANIFEST_DTS**: this option specifies a manifest file providing SP description. It is required when `SPMC_AT_EL3` is enabled, the secure partitions are loaded by BL2 on behalf of the SPMC.

Notes:

- BL32 option is re-purposed to specify the S-EL1 TEE or SP image. BL32 option can be omitted if using TF-A Test Secure Payload as SP.
- BL33 option can specify the TFTF binary or a normal world loader such as U-Boot or the UEFI framework payload.

Sample TF-A build command line when the SPMC is located at EL3:

```
make \
CROSS_COMPILE=aarch64-none-elf- \
SPD=spmd \
SPMD_SPM_AT_SEL2=0 \
SPMC_AT_EL3=1 \
BL32=<path-to-tee-binary> (opt for TSP) \
BL33=<path-to-bl33-binary> \
PLAT=fvp \
all fip
```

4.16.4 FVP model invocation

Sample FVP command line invocation:

```
<path-to-fvp-model>/FVP_Base_RevC-2xAEMvA -C pctl.startup=0.0.0.0 \
-C cluster0.NUM_CORES=4 -C cluster1.NUM_CORES=4 -C bp.secure_memory=1 \
-C bp.secureflashloader.fname=trusted-firmware-a/build/fvp/debug/bl1.bin \
-C bp.flashloader0.fname=trusted-firmware-a/build/fvp/debug/fip.bin \
-C bp.pl011_uart0.out_file=fvp-uart0.log -C bp.pl011_uart1.out_file=fvp-uart1.
↪log \
-C bp.pl011_uart2.out_file=fvp-uart2.log -C bp.vis.disable_visualisation=1
```

4.16.5 Platform Guide

- Platform Hooks See - *[4]*
  - plat_spmc_shmem_begin
  - plat_spmc_shmem_reclaim

SPMC provides platform hooks related to memory management interfaces. These hooks can be used for platform specific implementations such as managing access control, programming TZ Controllers or MPUs, and validating ownership and security state of memory regions supplied through FFA_MEM_SHARE and FFA_MEM_LEND requests.

Platform implementations are expected to ensure that memory regions accepted for sharing or lending comply with the platform security model and enforce any ownership, access control, and security-state checks required by the platform.

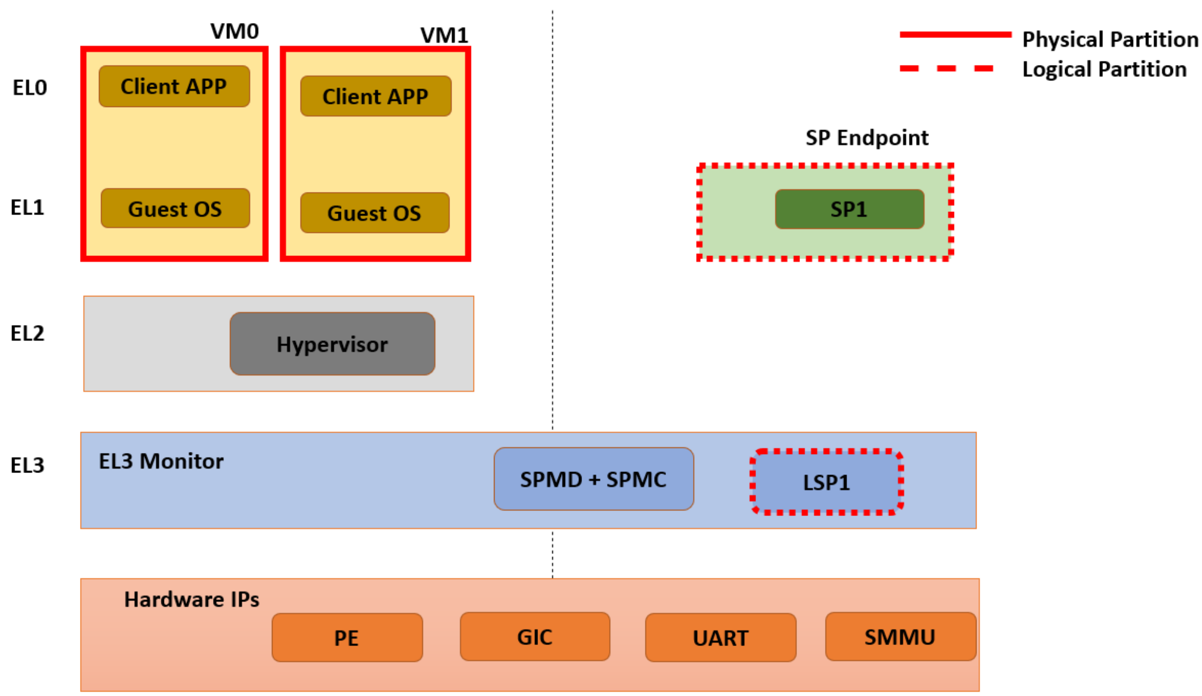
- Datastore
  - plat_spmc_shmem_datastore_get

    EL3 SPMC uses datastore for tracking memory transaction descriptors. On FVP platform datastore is allocated from TZC DRAM section. Other platforms need to allocate a similar secure memory region to be used as shared memory datastore.

    The accessor function is used during SPMC initialization to obtain address and size of the datastore. SPMC will also zero out the provided memory region.
- Platform Defines See - *[5]*
  - SECURE_PARTITION_COUNT Number of Secure Partitions supported: must be 1.
  - NS_PARTITION_COUNT Number of NWd Partitions supported.
  - MAX_EL3_LP_DESCS_COUNT Number of Logical Partitions supported.

4.16.6 Logical Secure Partition (LSP)

- The SPMC provides support for statically allocated EL3 Logical Secure Partitions as per FF-A v1.1 specification.
- The DECLARE_LOGICAL_PARTITION macro can be used to add a LSP.
- For reference implementation See - *[2]*



4.16.7 SPMC boot

The SPMD and SPMC are built into the BL31 image along with TF-A's runtime components. BL2 loads the BL31 image as a part of (secure) boot process.

The SPMC manifest is loaded by BL2 as the `TOS_FW_CONFIG` image *[9]*.

BL2 passes the SPMC manifest address to BL31 through a register.

At boot time, the SPMD in BL31 runs from the primary core, initializes the core contexts and launches the SPMC passing the following information through registers:

- X0 holds the SPMC manifest blob address.
- X4 holds the currently running core linear id.

Parsing SP partition manifests

SPMC consumes the SP manifest, as defined in *[7]*. SP manifest fields align with Hafnium SP manifest for easy porting.

```
compatible = "arm,ffa-manifest-1.0";

ffa-version = <0x00010001>; /* 31:16 - Major, 15:0 - Minor */
id = <0x8001>;
uuid = <0x6b43b460 0x74a24b78 0xade24502 0x40682886>;
messaging-method = <0x3>; /* Direct Messaging Only */
exception-level = <0x2>;  /* S-EL1 */
execution-state = <0>;
execution-ctx-count = <8>;
gp-register-num = <0>;
power-management-messages = <0x7>;
```

Passing boot data to the SP

In *[1]* , the section "Boot information protocol" defines a method for passing data to the SPs at boot time. It specifies the format for the boot information descriptor and boot information header structures, which describe the data to be exchanged between SPMC and SP. The specification also defines the types of data that can be passed. The aggregate of both the boot info structures and the data itself is designated the boot information blob, and is passed to a Partition as a contiguous memory region.

Currently, the SPM implementation supports the FDT type which is used to pass the partition's DTB manifest.

The region for the boot information blob is statically allocated (4K) by SPMC. BLOB contains Boot Info Header, followed by SP Manifest contents.

The configuration of the boot protocol is done in the SP manifest. As defined by the specification, the manifest field 'gp-register-num' configures the GP register which shall be used to pass the address to the partitions boot information blob when booting the partition.

4.16.8 Supported interfaces

The following interfaces are exposed to SPs only:

- FFA_MSG_WAIT
- FFA_MEM_RETRIEVE_REQ
- FFA_MEM_RETRIEVE_RESP
- FFA_MEM_RELINQUISH
- FFA_SECONDARY_EP_REGISTER

The following interfaces are exposed to both NS Client and SPs:

- FFA_VERSION
- FFA_FEATURES
- FFA_RX_RELEASE
- FFA_RXTX_MAP
- FFA_RXTX_UNMAP
- FFA_PARTITION_INFO_GET
- FFA_ID_GET
- FFA_MSG_SEND_DIRECT_REQ
- FFA_MSG_SEND_DIRECT_RESP
- FFA_MEM_FRAG_TX
- FFA_SPM_ID_GET

The following additional interfaces are forwarded from SPMD to support NS Client:

- FFA_RUN
- FFA_MEM_LEND
- FFA_MEM_SHARE
- FFA_MEM_FRAG_RX
- FFA_MEM_RECLAIM

FFA_VERSION

`FFA_VERSION` requires a *requested_version* parameter from the caller. SPMD forwards call to SPMC, the SPMC returns its own implemented version. SPMC asserts SP and SPMC are at same FF-A Version.

FFA_FEATURES

FF-A features supported by the SPMC may be discovered by secure partitions at boot (that is prior to NWd is booted) or run-time.

The SPMC calling FFA_FEATURES at secure physical FF-A instance always get FFA_SUCCESS from the SPMD.

The request made by an Hypervisor or OS kernel is forwarded to the SPMC and the response relayed back to the NWd.

FFA_RXTX_MAP

FFA_RXTX_UNMAP

When invoked from a secure partition FFA_RXTX_MAP maps the provided send and receive buffers described by their PAs to the EL3 translation regime as secure buffers in the MMU descriptors.

When invoked from the Hypervisor or OS kernel, the buffers are mapped into the SPMC EL3 translation regime and marked as NS buffers in the MMU descriptors.

The FFA_RXTX_UNMAP unmaps the RX/TX pair from the translation regime of the caller, either it being the Hypervisor or OS kernel, as well as a secure partition.

FFA_PARTITION_INFO_GET

Partition info get call can originate:

- from SP to SPMC
- from Hypervisor or OS kernel to SPMC. The request is relayed by the SPMD.

The format (v1.0 or v1.1) of the populated data structure returned is based upon the FFA version of the calling entity.

EL3 SPMC also supports returning only the count of partitions deployed.

All LSPs and SP are discoverable from FFA_PARTITION_INFO_GET call made by either SP or NWd entities.

FFA_ID_GET

The FF-A ID space is split into a non-secure space and secure space:

- FF-A ID with bit 15 clear relates to VMs.
- FF-A ID with bit 15 set related to SPs or LSPs.
- FF-A IDs 0, 0xffff, 0x8000 are assigned respectively to the Hypervisor (or OS Kernel if Hyp is absent), SPMD and SPMC.

This convention helps the SPM to determine the origin and destination worlds in an FF-A ABI invocation. In particular the SPM shall filter unauthorized transactions in its world switch routine. It must not be permitted for a VM to use a secure FF-A ID as origin world by spoofing:

- A VM-to-SP direct request/response shall set the origin world to be non-secure (FF-A ID bit 15 clear) and destination world to be secure (FF-A ID bit 15 set).
- Similarly, an SP-to-LSP direct request/response shall set the FF-A ID bit 15 for both origin and destination IDs.

An incoming direct message request arriving at SPMD from NWd is forwarded to SPMC without a specific check. The SPMC is resumed through eret and "knows" the message is coming from normal world in this specific code path. Thus the origin endpoint ID must be checked by SPMC for being a normal world ID.

An SP sending a direct message request must have bit 15 set in its origin endpoint ID and this can be checked by the SPMC when the SP invokes the ABI.

The SPMC shall reject the direct message if the claimed world in origin endpoint ID is not consistent:

- It is either forwarded by SPMD and thus origin endpoint ID must be a "normal world ID",
- or initiated by an SP and thus origin endpoint ID must be a "secure world ID".

FFA_MSG_SEND_DIRECT_REQ

FFA_MSG_SEND_DIRECT_RESP

This is a mandatory interface for secure partitions participating in direct request and responses with the following rules:

- An SP can send a direct request to LSP.
- An LSP can send a direct response to SP.
- An SP cannot send a direct request to an Hypervisor or OS kernel.
- An Hypervisor or OS kernel can send a direct request to an SP or LSP.
- An SP and LSP can send a direct response to an Hypervisor or OS kernel.
- SPMD can send direct request to SPMC.

FFA_SPM_ID_GET

Returns the FF-A ID allocated to an SPM component which can be one of SPMD or SPMC.

At initialization, the SPMC queries the SPMD for the SPMC ID, using the FFA_ID_GET interface, and records it. The SPMC can also query the SPMD ID using the FFA_SPM_ID_GET interface at the secure physical FF-A instance.

Secure partitions call this interface at the virtual FF-A instance, to which the SPMC returns the SPMC ID.

The Hypervisor or OS kernel can issue the FFA_SPM_ID_GET call handled by the SPMD, which returns the SPMC ID.

FFA_ID_GET

Returns the FF-A ID of the calling endpoint.

FFA_MEM_SHARE

FFA_MEM_LEND

- If SP is borrower in the memory transaction, these calls are forwarded to SPMC. SPMC performs Relayer responsibilities, caches the memory descriptors in the datastore, and allocates FF-A memory handle.
- If format of descriptor was v1.0, SPMC converts the descriptor to v1.1 before caching. In case of fragmented sharing, conversion of memory descriptors happens after last fragment has been received.
- Multiple borrowers (including NWd endpoint) and fragmented memory sharing are supported.

FFA_MEM_RETRIEVE_REQ

FFA_MEM_RETRIEVE_RESP

- Memory retrieve is supported only from SP.
- SPMC fetches the cached memory descriptor from the datastore,
- Performs Relayer responsiilities and sends FFA_MEM_RETRIEVE_RESP back to SP.
- If descriptor size is more than RX buffer size, SPMC will send the descriptor in fragments.
- SPMC will set NS Bit to 1 in memory descriptor response.

FFA_MEM_FRAG_RX

FFA_MEM_FRAG_TX

FFA_MEM_FRAG_RX is to be used by:

- SP if FFA_MEM_RETRIEVE_RESP returned descriptor with fragment length less than total length.
- or by SPMC if FFA_MEM_SHARE/FFA_MEM_LEND is called with fragment length less than total length.

SPMC validates handle and Endpoint ID and returns response with FFA_MEM_FRAG_TX.

FFA_SECONDARY_EP_REGISTER

When the SPMC boots, secure partition is initialized on its primary Execution Context.

The FFA_SECONDARY_EP_REGISTER interface is to be used by a secure partition from its first execution context, to provide the entry point address for secondary execution contexts.

A secondary EC is first resumed either upon invocation of PSCI_CPU_ON from the NWd or by invocation of FFA_RUN.

4.16.9 Power management

In platforms with or without secure virtualization:

- The NWd owns the platform PM policy.
- The Hypervisor or OS kernel is the component initiating PSCI service calls.
- The EL3 PSCI library is in charge of the PM coordination and control (eventually writing to platform registers).
- While coordinating PM events, the PSCI library calls backs into the Secure Payload Dispatcher for events the latter has statically registered to.

When using the SPMD as a Secure Payload Dispatcher:

- A power management event is relayed through the SPD hook to the SPMC.
- In the current implementation CPU_ON (svc_on_finish), CPU_OFF (svc_off), CPU_SUSPEND (svc_suspend) and CPU_SUSPEND_RESUME (svc_suspend_finish) hooks are registered.

4.16.10 Secure partitions scheduling

The FF-A specification *[1]* provides two ways to relinquinsh CPU time to secure partitions. For this a VM (Hypervisor or OS kernel), or SP invokes one of:

- the FFA_MSG_SEND_DIRECT_REQ interface.
- the FFA_RUN interface.

Additionally a secure interrupt can pre-empt the normal world execution and give CPU cycles by transitioning to EL3.

4.16.11 Partition Runtime State and Model

EL3 SPMC implements Partition runtime states are described in v1.1 FF-A specification *[1]*

An SP can be in one of the following state:

- RT_STATE_WAITING
- RT_STATE_RUNNING
- RT_STATE_PREEMPTED
- RT_STATE_BLOCKED

An SP will transition to one of the following runtime model when not in waiting state:

- RT_MODEL_DIR_REQ
- RT_MODEL_RUN
- RT_MODEL_INIT
- RT_MODEL_INTR

4.16.12 Platform topology

SPMC only supports a single Pinned MP S-EL1 SP. The *execution-ctx-count* SP manifest field should match the number of physical PE.

4.16.13 Interrupt handling

Secure Interrupt handling

- SPMC is capable of forwarding Secure interrupt to S-EL1 SP which has preempted the normal world.
- Interrupt is forwarded to SP using FFA_INTERRUPT interface.
- Interrupt Number is not passed, S-EL1 SP can access the GIC registers directly.
- Upon completion of Interrupt handling SP is expected to return to SPMC using FFA_MSG_WAIT interface.

- SPMC returns to normal world after interrupt handling is completed.

In the scenario when secure interrupt occurs while the secure partition is running, the SPMC is not involved and the handling is implementation defined in the TOS.

Non-Secure Interrupt handling

The 'managed exit' scenario is the responsibility of the TOS and the SPMC is not involved.

4.16.14 Test Secure Payload (TSP)

- TSP provides reference implementation of FF-A programming model.
- TSP has the following support:
  - SP initialization on all CPUs.
  - Consuming Power Messages including CPU_ON, CPU_OFF, CPU_SUSPEND, CPU_SUSPEND_RESUME.
  - Event Loop to receive Direct Requests.
  - Sending Direct Response.
  - Memory Sharing helper library.
  - Ability to handle secure interrupt (timer).

TSP Tests in CI

- TSP Tests are exercised in the TF-A CI using prebuilt FF-A Linux Test driver in NWd.
- Expected output:

```
#ioctl 255
Test: Echo Message to SP.
Status: Completed Test Case: 1
Test Executed Successfully

Test: Message Relay vis SP to EL3 LSP.
Status: Completed Test Case: 2
Test Executed Successfully

Test: Memory Send.
Verified 1 constituents successfully
Status: Completed Test Case: 3
Test Executed Successfully

Test: Memory Send in Fragments.
Verified 256 constituents successfully
Status: Completed Test Case: 4
```

(continues on next page)

(continued from previous page)

```
Test Executed Successfully

Test: Memory Lend.
Verified 1 constituents successfully
Status: Completed Test Case: 5
Test Executed Successfully

Test: Memory Lend in Fragments.
Verified 256 constituents successfully
Status: Completed Test Case: 6
Test Executed Successfully

Test: Memory Send with Multiple Endpoints.
random: fast init done
Verified 256 constituents successfully
Status: Completed Test Case: 7
Test Executed Successfully

Test: Memory Lend with Multiple Endpoints.
Verified 256 constituents successfully
Status: Completed Test Case: 8
Test Executed Successfully

Test: Ensure Duplicate Memory Send Requests are Rejected.
Status: Completed Test Case: 9
Test Executed Successfully

Test: Ensure Duplicate Memory Lend Requests are Rejected.
Status: Completed Test Case: 10
Test Executed Successfully

0 Tests Failed

Exiting Test Application -  Total Failures: 0
```

4.16.15 References

[1] Arm Firmware Framework for Arm A-profile

[2] https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/tree/plat/arm/board/fvp/fvp_el3_spmc_logical_sp.c

[3] Trusted Boot Board Requirements Client

[4] https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/tree/plat/arm/board/fvp/fvp_el3_spmc.c

[5] https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/tree/plat/arm/board/fvp/include/platform_def.h

[6] https://trustedfirmware-a.readthedocs.io/en/latest/components/ffa-manifest-binding.html

[7] https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/tree/plat/arm/board/fvp/fdts/fvp_tsp_sp_manifest.dts

[8] https://lists.trustedfirmware.org/archives/list/tf-a@lists.trustedfirmware.org/thread/CFQFGU6H2D5GZYMUYGTGUSXIU3OYZP6U/

[9] https://trustedfirmware-a.readthedocs.io/en/latest/design/firmware-design.html#dynamic-configuration-during-cold-boot

---

4.17 Secure Partition Manager (MM)

4.17.1 Foreword

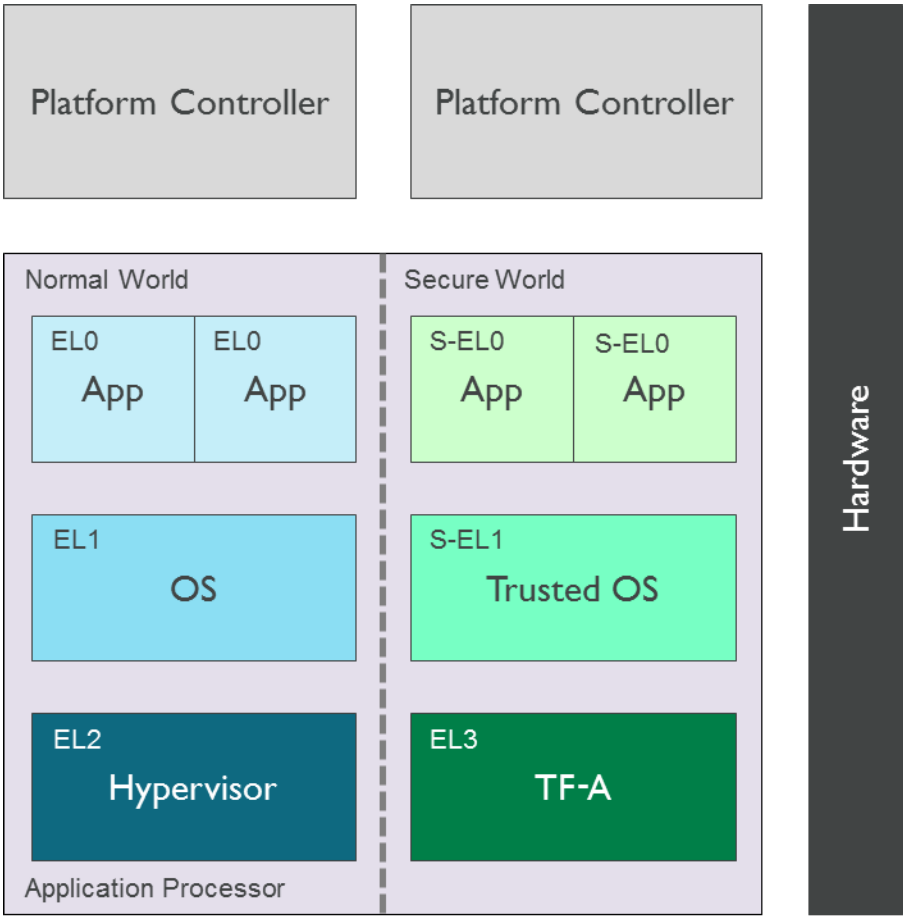
This document describes the implementation where the Secure Partition Manager resides at EL3 and management services run from isolated Secure Partitions at S-EL0. The communication protocol is established through the Management Mode (MM) interface.

4.17.2 Background

In some market segments that primarily deal with client-side devices like mobile phones, tablets, STBs and embedded devices, a Trusted OS instantiates trusted applications to provide security services like DRM, secure payment and authentication. The Global Platform TEE Client API specification defines the API used by Non-secure world applications to access these services. A Trusted OS fulfils the requirements of a security service as described above.

Management services are typically implemented at the highest level of privilege in the system, i.e. EL3 in Trusted Firmware-A (TF-A). The service requirements are fulfilled by the execution environment provided by TF-A.

The following diagram illustrates the corresponding software stack:



In other market segments that primarily deal with server-side devices (e.g. data centres and enterprise servers) the secure software stack typically does not include a Global Platform Trusted OS. Security functions are accessed through other interfaces (e.g. ACPI TCG TPM interface, UEFI runtime variable service).

Placement of management and security functions with diverse requirements in a privileged Exception Level (i.e. EL3 or S-EL1) makes security auditing of firmware more difficult and does not allow isolation of unrelated services from each other either.

4.17.3 Introduction

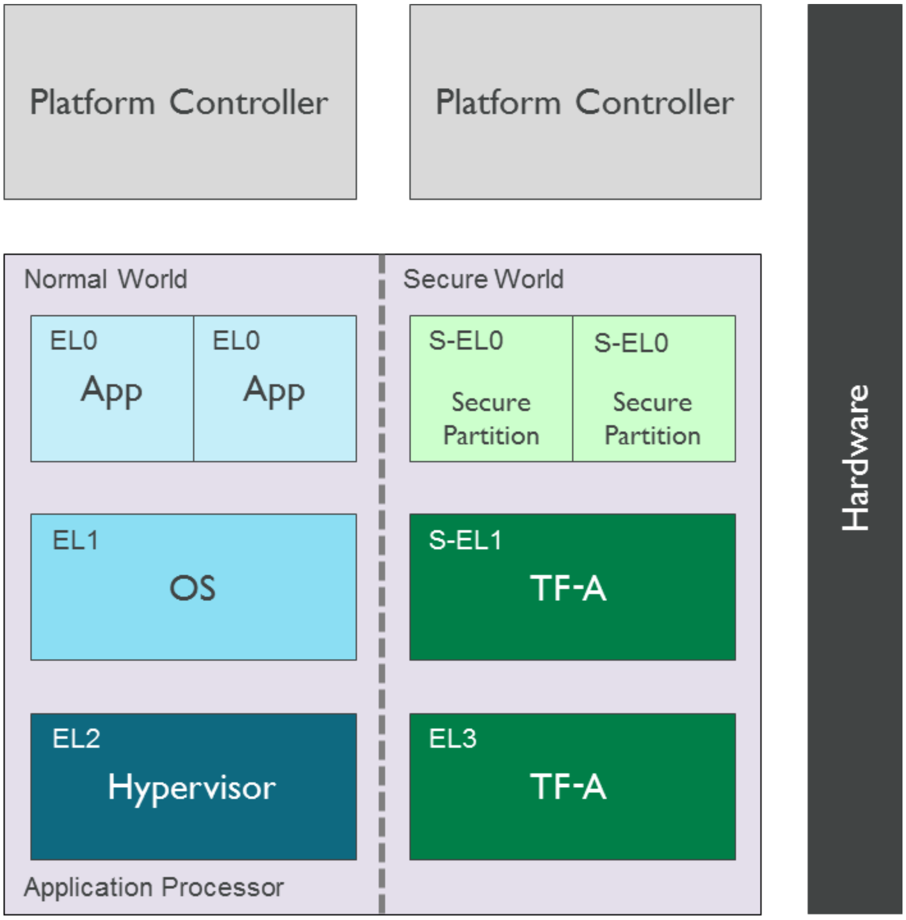
A **Secure Partition** is a software execution environment instantiated in S-EL0 that can be used to implement simple management and security services. Since S-EL0 is an unprivileged Exception Level, a Secure Partition relies on privileged firmware (i.e. TF-A) to be granted access to system and processor resources. Essentially, it is a software sandbox in the Secure world that runs under the control of privileged software, provides one or more services and accesses the following system resources:

- Memory and device regions in the system address map.
- PE system registers.
- A range of synchronous exceptions (e.g. SMC function identifiers).

Note that currently TF-A only supports handling one Secure Partition.

A Secure Partition enables TF-A to implement only the essential secure services in EL3 and instantiate the rest in a partition in S-EL0. Furthermore, multiple Secure Partitions can be used to isolate unrelated services from each other.

The following diagram illustrates the place of a Secure Partition in a typical Armv8-A software stack. A single or multiple Secure Partitions provide secure services to software components in the Non-secure world and other Secure Partitions.



The TF-A build system is responsible for including the Secure Partition image in the FIP. During boot, BL2 includes support to authenticate and load the Secure Partition image. A BL31 component called **Secure Partition Manager (SPM)** is responsible for managing the partition. This is semantically similar to a hypervisor managing a virtual machine.

The SPM is responsible for the following actions during boot:

- Allocate resources requested by the Secure Partition.
- Perform architectural and system setup required by the Secure Partition to fulfil a service request.
- Implement a standard interface that is used for initialising a Secure Partition.

The SPM is responsible for the following actions during runtime:

- Implement a standard interface that is used by a Secure Partition to fulfil service requests.
- Implement a standard interface that is used by the Non-secure world for accessing the services exported

by a Secure Partition. A service can be invoked through a SMC.

Alternatively, a partition can be viewed as a thread of execution running under the control of the SPM. Hence common programming concepts described below are applicable to a partition.

4.17.4 Description

The previous section introduced some general aspects of the software architecture of a Secure Partition. This section describes the specific choices made in the current implementation of this software architecture. Subsequent revisions of the implementation will include a richer set of features that enable a more flexible architecture.

Building TF-A with Secure Partition support

SPM is supported on the Arm FVP exclusively at the moment. The current implementation supports inclusion of only a single Secure Partition in which a service always runs to completion (e.g. the requested services cannot be preempted to give control back to the Normal world).

It is not currently possible for BL31 to integrate SPM support and a Secure Payload Dispatcher (SPD) at the same time; they are mutually exclusive. In the SPM bootflow, a Secure Partition image executing at S-EL0 replaces the Secure Payload image executing at S-EL1 (e.g. a Trusted OS). Both are referred to as BL32.

A working prototype of a SP has been implemented by re-purposing the EDK2 code and tools, leveraging the concept of the *Standalone Management Mode (MM)* in the UEFI specification (see the PI v1.6 Volume 4: Management Mode Core Interface). This will be referred to as the *Standalone MM Secure Partition* in the rest of this document.

To enable SPM support in TF-A, the source code must be compiled with the build flag `SPM_MM=1`, along with `EL3_EXCEPTION_HANDLING=1` and `ENABLE_SVE_FOR_NS=0`. On Arm platforms the build option `ARM_BL31_IN_DRAM` must be set to 1. Also, the location of the binary that contains the BL32 image (`BL32=path/to/image.bin`) must be specified.

First, build the Standalone MM Secure Partition. To build it, refer to the instructions in the EDK2 repository.

Then build TF-A with SPM support and include the Standalone MM Secure Partition image in the FIP:

```
BL32=path/to/standalone/mm/sp BL33=path/to/bl33.bin \
make PLAT=fvp SPM_MM=1 EL3_EXCEPTION_HANDLING=1 ENABLE_SVE_FOR_NS=0 ARM_BL31_
↪IN_DRAM=1 all fip
```

Describing Secure Partition resources

TF-A exports a porting interface that enables a platform to specify the system resources required by the Secure Partition. Some instructions are given below. However, this interface is under development and it may change as new features are implemented.

- A Secure Partition is considered a BL32 image, so the same defines that apply to BL32 images apply to a Secure Partition: `BL32_BASE` and `BL32_LIMIT`.

- The following defines are needed to allocate space for the translation tables used by the Secure Partition: `PLAT_SP_IMAGE_MMAP_REGIONS` and `PLAT_SP_IMAGE_MAX_XLAT_TABLES`.
- The functions `plat_get_secure_partition_mmap()` and `plat_get_secure_partition_boot_info()` have to be implemented. The file `plat/arm/board/fvp/fvp_common.c` can be used as an example. It uses the defines in `include/plat/arm/common/arm_spm_def.h`.
  - `plat_get_secure_partition_mmap()` returns an array of mmap regions that describe the memory regions that the SPM needs to allocate for a Secure Partition.
  - `plat_get_secure_partition_boot_info()` returns a `spm_mm_boot_info_t` struct that is populated by the platform with information about the memory map of the Secure Partition.

For an example of all the changes in context, you may refer to commit `e29efeb1b4`, in which the port for FVP was introduced.

Accessing Secure Partition services

The SMC Calling Convention (*Arm DEN 0028B*) describes SMCs as a conduit for accessing services implemented in the Secure world. The `MM_COMMUNICATE` interface defined in the Management Mode Interface Specification (*Arm DEN 0060A*) is used to invoke a Secure Partition service as a Fast Call.

The mechanism used to identify a service within the partition depends on the service implementation. It is assumed that the caller of the service will be able to discover this mechanism through standard platform discovery mechanisms like ACPI and Device Trees. For example, *Volume 4: Platform Initialisation Specification v1.6. Management Mode Core Interface* specifies that a GUID is used to identify a management mode service. A client populates the GUID in the `EFI_MM_COMMUNICATE_HEADER`. The header is populated in the communication buffer shared with the Secure Partition.

A Fast Call appears to be atomic from the perspective of the caller and returns when the requested operation has completed. A service invoked through the `MM_COMMUNICATE` SMC will run to completion in the partition on a given CPU. The SPM is responsible for guaranteeing this behaviour. This means that there can only be a single outstanding Fast Call in a partition on a given CPU.

Exchanging data with the Secure Partition

The exchange of data between the Non-secure world and the partition takes place through a shared memory region. The location of data in the shared memory area is passed as a parameter to the `MM_COMMUNICATE` SMC. The shared memory area is statically allocated by the SPM and is expected to be either implicitly known to the Non-secure world or discovered through a platform discovery mechanism e.g. ACPI table or device tree. It is possible for the Non-secure world to exchange data with a partition only if it has been populated in this shared memory area. The shared memory area is implemented as per the guidelines specified in Section 3.2.3 of the Management Mode Interface Specification (*Arm DEN 0060A*).

The format of data structures used to encapsulate data in the shared memory is agreed between the Non-secure world and the Secure Partition. For example, in the Management Mode Interface specification (*Arm DEN 0060A*), Section 4 describes that the communication buffer shared between the Non-secure world and the Management Mode (MM) in the Secure world must be of the type `EFI_MM_COMMUNICATE_HEADER`.

This data structure is defined in *Volume 4: Platform Initialisation Specification v1.6. Management Mode Core Interface*. Any caller of a MM service will have to use the `EFI_MM_COMMUNICATE_HEADER` data structure.

4.17.5 Runtime model of the Secure Partition

This section describes how the Secure Partition interfaces with the SPM.

Interface with SPM

In order to instantiate one or more secure services in the Secure Partition in S-EL0, the SPM should define the following types of interfaces:

- Interfaces that enable access to privileged operations from S-EL0. These operations typically require access to system resources that are either shared amongst multiple software components in the Secure world or cannot be directly accessed from an unprivileged Exception Level.
- Interfaces that establish the control path between the SPM and the Secure Partition.

This section describes the APIs currently exported by the SPM that enable a Secure Partition to initialise itself and export its services in S-EL0. These interfaces are not accessible from the Non-secure world.

Conduit

The SMC Calling Convention (*Arm DEN 0028B*) specification describes the SMC and HVC conduits for accessing firmware services and their availability depending on the implemented Exception levels. In S-EL0, the Supervisor Call exception (SVC) is the only architectural mechanism available for unprivileged software to make a request for an operation implemented in privileged software. Hence, the SVC conduit must be used by the Secure Partition to access interfaces implemented by the SPM.

A SVC causes an exception to be taken to S-EL1. TF-A assumes ownership of S-EL1 and installs a simple exception vector table in S-EL1 that relays a SVC request from a Secure Partition as a SMC request to the SPM in EL3. Upon servicing the SMC request, Trusted Firmware-A returns control directly to S-EL0 through an ERET instruction.

Calling conventions

The SMC Calling Convention (*Arm DEN 0028B*) specification describes the 32-bit and 64-bit calling conventions for the SMC and HVC conduits. The SVC conduit introduces the concept of SVC32 and SVC64 calling conventions. The SVC32 and SVC64 calling conventions are equivalent to the 32-bit (SMC32) and the 64-bit (SMC64) calling conventions respectively.

Communication initiated by SPM

A service request is initiated from the SPM through an exception return instruction (ERET) to S-EL0. Later, the Secure Partition issues an SVC instruction to signal completion of the request. Some example use cases are given below:

- A request to initialise the Secure Partition during system boot.
- A request to handle a runtime service request.

Communication initiated by Secure Partition

A request is initiated from the Secure Partition by executing a SVC instruction. An ERET instruction is used by TF-A to return to S-EL0 with the result of the request.

For instance, a request to perform privileged operations on behalf of a partition (e.g. management of memory attributes in the translation tables for the Secure EL1&0 translation regime).

Interfaces

The current implementation reserves function IDs for Fast Calls in the Standard Secure Service calls range (see SMC Calling Convention (*Arm DEN 0028B*) specification) for each API exported by the SPM. This section defines the function prototypes for each function ID. The function IDs specify whether one or both of the SVC32 and SVC64 calling conventions can be used to invoke the corresponding interface.

Secure Partition Event Management

The Secure Partition provides an Event Management interface that is used by the SPM to delegate service requests to the Secure Partition. The interface also allows the Secure Partition to:

- Register with the SPM a service that it provides.
- Indicate completion of a service request delegated by the SPM

Miscellaneous interfaces

`SPM_MM_VERSION_AARCH32`

- Description

  Returns the version of the interface exported by SPM.

- Parameters
  - **uint32** - Function ID
    - SVC32 Version: **0x84000060**

- Return parameters
  - **int32** - Status

    On success, the format of the value is as follows:

    * Bit [31]: Must be 0
    * Bits [30:16]: Major Version. Must be 0 for this revision of the SPM interface.
    * Bits [15:0]: Minor Version. Must be 1 for this revision of the SPM interface.

    On error, the format of the value is as follows:

    * `NOT_SUPPORTED`: SPM interface is not supported or not available for the client.

- Usage

  This function returns the version of the Secure Partition Manager implementation. The major version is 0 and the minor version is 1. The version number is a 31-bit unsigned integer, with the upper 15 bits denoting the major revision, and the lower 16 bits denoting the minor revision. The following rules apply to the version numbering:

  - Different major revision values indicate possibly incompatible functions.
  - For two revisions, A and B, for which the major revision values are identical, if the minor revision value of revision B is greater than the minor revision value of revision A, then every function in revision A must work in a compatible way with revision B. However, it is possible for revision B to have a higher function count than revision A.

- Implementation responsibilities

  If this function returns a valid version number, all the functions that are described subsequently must be implemented, unless it is explicitly stated that a function is optional.

See *Error Codes* for integer values that are associated with each return code.

Secure Partition Initialisation

The SPM is responsible for initialising the architectural execution context to enable initialisation of a service in S-EL0. The responsibilities of the SPM are listed below. At the end of initialisation, the partition issues a `MM_SP_EVENT_COMPLETE_AARCH64` call (described later) to signal readiness for handling requests for services implemented by the Secure Partition. The initialisation event is executed as a Fast Call.

Entry point invocation

The entry point for service requests that should be handled as Fast Calls is used as the target of the ERET instruction to start initialisation of the Secure Partition.

Architectural Setup

At cold boot, system registers accessible from S-EL0 will be in their reset state unless otherwise specified. The SPM will perform the following architectural setup to enable execution in S-EL0

MMU setup

The platform port of a Secure Partition specifies to the SPM a list of regions that it needs access to and their attributes. The SPM validates this resource description and initialises the Secure EL1&0 translation regime as follows.

1. Device regions are mapped with nGnRE attributes and Execute Never instruction access permissions.
2. Code memory regions are mapped with RO data and Executable instruction access permissions.
3. Read Only data memory regions are mapped with RO data and Execute Never instruction access permissions.
4. Read Write data memory regions are mapped with RW data and Execute Never instruction access permissions.
5. If the resource description does not explicitly describe the type of memory regions then all memory regions will be marked with Code memory region attributes.
6. The `UXN` and `PXN` bits are set for regions that are not executable by S-EL0 or S-EL1.

System Register Setup

System registers that influence software execution in S-EL0 are setup by the SPM as follows:

1. `SCTLR_EL1`
   - `UCI=1`
   - `EOE=0`
   - `WXN=1`
   - `nTWE=1`
   - `nTWI=1`
   - `UCT=1`
   - `DZE=1`
   - `I=1`
   - `UMA=0`
   - `SA0=1`
   - `C=1`
   - `A=1`

    - `M=1`

2. `CPACR_EL1`

    - `FPEN=b'11`

3. `PSTATE`

    - `D,A,I,F=1`
    - `CurrentEL=0` (EL0)
    - `SpSel=0` (Thread mode)
    - `NRW=0` (AArch64)

General Purpose Register Setup

SPM will invoke the entry point of a service by executing an ERET instruction. This transition into S-EL0 is special since it is not in response to a previous request through a SVC instruction. This is the first entry into S-EL0. The general purpose register usage at the time of entry will be as specified in the "Return State" column of Table 3-1 in Section 3.1 "Register use in AArch64 SMC calls" of the SMC Calling Convention (*Arm DEN 0028B*) specification. In addition, certain other restrictions will be applied as described below.

1. `SP_EL0`

    A non-zero value will indicate that the SPM has initialised the stack pointer for the current CPU.

    The value will be 0 otherwise.

2. `X4-X30`

    The values of these registers will be 0.

3. `X0-X3`

    Parameters passed by the SPM.

    - `X0`: Virtual address of a buffer shared between EL3 and S-EL0. The buffer will be mapped in the Secure EL1&0 translation regime with read-only memory attributes described earlier.
    - `X1`: Size of the buffer in bytes.
    - `X2`: Cookie value (*IMPLEMENTATION DEFINED*).
    - `X3`: Cookie value (*IMPLEMENTATION DEFINED*).

Runtime Event Delegation

The SPM receives requests for Secure Partition services through a synchronous invocation (i.e. a SMC from the Non-secure world). These requests are delegated to the partition by programming a return from the last `MM_SP_EVENT_COMPLETE_AARCH64` call received from the partition. The last call was made to signal either completion of Secure Partition initialisation or completion of a partition service request.

`MM_SP_EVENT_COMPLETE_AARCH64`

- Description

  Signal completion of the last SP service request.

- Parameters

  - **uint32** - Function ID

    - SVC64 Version: **0xC4000061**

  - **int32** - Event Status Code

    Zero or a positive value indicates that the event was handled successfully. The values depend upon the original event that was delegated to the Secure partition. They are described as follows.

    - `SUCCESS` : Used to indicate that the Secure Partition was initialised or a runtime request was handled successfully.

    - Any other value greater than 0 is used to pass a specific Event Status code in response to a runtime event.

    A negative value indicates an error. The values of Event Status code depend on the original event.

- Return parameters

  - **int32** - Event ID/Return Code

    Zero or a positive value specifies the unique ID of the event being delegated to the partition by the SPM.

    In the current implementation, this parameter contains the function ID of the `MM_COMMUNICATE` SMC. This value indicates to the partition that an event has been delegated to it in response to an `MM_COMMUNICATE` request from the Non-secure world.

    A negative value indicates an error. The format of the value is as follows:

    - `NOT_SUPPORTED`: Function was called from the Non-secure world.

    See *Error Codes* for integer values that are associated with each return code.

  - **uint32** - Event Context Address

    Address of a buffer shared between the SPM and Secure Partition to pass event specific information. The format of the data populated in the buffer is implementation defined.

    The buffer is mapped in the Secure EL1&0 translation regime with read-only memory attributes described earlier.

For the SVC64 version, this parameter is a 64-bit Virtual Address (VA).

For the SVC32 version, this parameter is a 32-bit Virtual Address (VA).

- **uint32** - Event context size

  Size of the memory starting at Event Address.

- **uint32/uint64** - Event Cookie

  This is an optional parameter. If unused its value is SBZ.

- Usage

  This function signals to the SPM that the handling of the last event delegated to a partition has completed. The partition is ready to handle its next event. A return from this function is in response to the next event that will be delegated to the partition. The return parameters describe the next event.

- Caller responsibilities

  A Secure Partition must only call `MM_SP_EVENT_COMPLETE_AARCH64` to signal completion of a request that was delegated to it by the SPM.

- Callee responsibilities

  When the SPM receives this call from a Secure Partition, the corresponding syndrome information can be used to return control through an ERET instruction, to the instruction immediately after the call in the Secure Partition context. This syndrome information comprises of general purpose and system register values when the call was made.

  The SPM must save this syndrome information and use it to delegate the next event to the Secure Partition. The return parameters of this interface must specify the properties of the event and be populated in `X0-X3/W0-W3` registers.

Secure Partition Memory Management

A Secure Partition executes at S-EL0, which is an unprivileged Exception Level. The SPM is responsible for enabling access to regions of memory in the system address map from a Secure Partition. This is done by mapping these regions in the Secure EL1&0 Translation regime with appropriate memory attributes. Attributes refer to memory type, permission, cacheability and shareability attributes used in the Translation tables. The definitions of these attributes and their usage can be found in the Armv8-A ARM (*Arm DDI 0487*).

All memory required by the Secure Partition is allocated upfront in the SPM, even before handing over to the Secure Partition for the first time. The initial access permissions of the memory regions are statically provided by the platform port and should allow the Secure Partition to run its initialisation code.

However, they might not suit the final needs of the Secure Partition because its final memory layout might not be known until the Secure Partition initialises itself. As the Secure Partition initialises its runtime environment it might, for example, load dynamically some modules. For instance, a Secure Partition could implement a loader for a standard executable file format (e.g. an PE-COFF loader for loading executable files at runtime). These executable files will be a part of the Secure Partition image. The location of various sections in an executable file and their permission attributes (e.g. read-write data, read-only data and code) will be known only when the file is loaded into memory.

In this case, the Secure Partition needs a way to change the access permissions of its memory regions. The SPM provides this feature through the `MM_SP_MEMORY_ATTRIBUTES_SET_AARCH64` SVC interface. This interface is available to the Secure Partition during a specific time window: from the first entry into the Secure Partition up to the first `SP_EVENT_COMPLETE` call that signals the Secure Partition has finished its initialisation. Once the initialisation is complete, the SPM does not allow changes to the memory attributes.

This section describes the standard SVC interface that is implemented by the SPM to determine and change permission attributes of memory regions that belong to a Secure Partition.

MM_SP_MEMORY_ATTRIBUTES_GET_AARCH64

- Description

  Request the permission attributes of a memory region from S-EL0.

- Parameters

  - **uint32** Function ID

    - SVC64 Version: **0xC4000064**

  - **uint64** Base Address

    This parameter is a 64-bit Virtual Address (VA).

    There are no alignment restrictions on the Base Address. The permission attributes of the translation granule it lies in are returned.

  - **uint32** Input Page Count

    This parameter is the number of translation granule size pages from *Base Address* whose permission should be returned. This is calculated as *Input Page count + 1*. (i.e. If Input Page Count is 0, then it is calculated as 1).

- Return parameters

  - **int32** - Memory Attributes/Return Code

    On success the format of the Return Code is as follows:

    - Bits[1:0] : Data access permission

      - b'00 : No access
      - b'01 : Read-Write access
      - b'10 : Reserved
      - b'11 : Read-only access

    - Bit[2]: Instruction access permission

      - b'0 : Executable
      - b'1 : Non-executable

    - Bit[30:3] : Reserved. SBZ.

        * Bit[31] : Must be 0

    On failure the following error codes are returned:

        * `INVALID_PARAMETERS`: The Secure Partition is not allowed to access the memory region the Base Address lies in.
        * `NOT_SUPPORTED` : The SPM does not support retrieval of attributes of any memory page that is accessible by the Secure Partition, or the function was called from the Non-secure world. Also returned if it is used after `MM_SP_EVENT_COMPLETE_AARCH64`.

    See *Error Codes* for integer values that are associated with each return code.

  - **uint32** - Output Page Count

    On success, the number of translation granule size pages from the *Base address* whose permissions match those returned in the *Memory Attributes* output parameter. This is calculated as *Output Page count + 1*. (i.e. If Output Page Count is 0, It is calculated as 1).

    On failure, It must be zero:

- Usage

  This function is used to request the permission attributes for S-EL0 on a memory region accessible from a Secure Partition. The size of the memory region is equal to the Translation Granule size used in the Secure EL1&0 translation regime. Requests to retrieve other memory region attributes are not currently supported.

- Caller responsibilities

  The caller must obtain the Translation Granule Size of the Secure EL1&0 translation regime from the SPM through an implementation defined method.

- Callee responsibilities

  The SPM must not return the memory access controls for a page of memory that is not accessible from a Secure Partition.

`MM_SP_MEMORY_ATTRIBUTES_SET_AARCH64`

- Description

  Set the permission attributes of a memory region from S-EL0.

- Parameters

  - **uint32** - Function ID

        * SVC64 Version: **0xC4000065**

  - **uint64** - Base Address

    This parameter is a 64-bit Virtual Address (VA).

    The alignment of the Base Address must be greater than or equal to the size of the Translation Granule Size used in the Secure EL1&0 translation regime.

- **uint32** - Page count

  Number of pages starting from the Base Address whose memory attributes should be changed. The page size is equal to the Translation Granule Size.

- **uint32** - Memory Access Controls

  * Bits[1:0] : Data access permission
    · b'00 : No access
    · b'01 : Read-Write access
    · b'10 : Reserved
    · b'11 : Read-only access
  * Bit[2] : Instruction access permission
    · b'0 : Executable
    · b'1 : Non-executable
  * Bits[31:3] : Reserved. SBZ.

  A combination of attributes that mark the region with RW and Executable permissions is prohibited. A request to mark a device memory region with Executable permissions is prohibited.

• Return parameters

- **int32** - Return Code

  * `SUCCESS`: The Memory Access Controls were changed successfully.
  * `DENIED`: The SPM is servicing a request to change the attributes of a memory region that overlaps with the region specified in this request.
  * `INVALID_PARAMETER`: An invalid combination of Memory Access Controls has been specified. The Base Address is not correctly aligned. The Secure Partition is not allowed to access part or all of the memory region specified in the call.
  * `NO_MEMORY`: The SPM does not have memory resources to change the attributes of the memory region in the translation tables.
  * `NOT_SUPPORTED`: The SPM does not permit change of attributes of any memory region that is accessible by the Secure Partition. Function was called from the Non-secure world. Also returned if it is used after `MM_SP_EVENT_COMPLETE_AARCH64`.

  See *Error Codes* for integer values that are associated with each return code.

• Usage

  This function is used to change the permission attributes for S-EL0 on a memory region accessible from a Secure Partition. The size of the memory region is equal to the Translation Granule size used in the Secure EL1&0 translation regime. Requests to change other memory region attributes are not currently supported.

This function is only available at boot time. This interface is revoked after the Secure Partition sends the first `MM_SP_EVENT_COMPLETE_AARCH64` to signal that it is initialised and ready to receive run-time requests.

- Caller responsibilities

  The caller must obtain the Translation Granule Size of the Secure EL1&0 translation regime from the SPM through an implementation defined method.

- Callee responsibilities

  The SPM must preserve the original memory access controls of the region of memory in case of an unsuccessful call. The SPM must preserve the consistency of the S-EL1 translation regime if this function is called on different PEs concurrently and the memory regions specified overlap.

Error Codes

Name	Value
`SUCCESS`	0
`NOT_SUPPORTED`	-1
`INVALID_PARAMETER`	-2
`DENIED`	-3
`NO_MEMORY`	-5
`NOT_PRESENT`	-7

4.18 Translation (XLAT) Tables Library

This document describes the design of the translation tables library (version 2) used by Trusted Firmware-A (TF-A). This library provides APIs to create page tables based on a description of the memory layout, as well as setting up system registers related to the Memory Management Unit (MMU) and performing the required Translation Lookaside Buffer (TLB) maintenance operations.

More specifically, some use cases that this library aims to support are:

1. Statically allocate translation tables and populate them (at run-time) based on a description of the memory layout. The memory layout is typically provided by the platform port as a list of memory regions;
2. Support for generating translation tables pertaining to a different translation regime than the exception level the library code is executing at;
3. Support for dynamic mapping and unmapping of regions, even while the MMU is on. This can be used to temporarily map some memory regions and unmap them later on when no longer needed;
4. Support for non-identity virtual to physical mappings to compress the virtual address space;
5. Support for changing memory attributes of memory regions at run-time.

4.18.1 About version 1 and version 2

This document focuses on version 2 of the library, whose sources are available in the `lib/xlat_tables_v2` directory. Version 1 of the library can still be found in `lib/xlat_tables` directory but it is less flexible and doesn't support dynamic mapping. Although potential bug fixes will be applied to both versions, future features enhancements will focus on version 2 and might not be back-ported to version 1. Therefore, it is recommended to use version 2, especially for new platform ports.

However, please note that version 2 is still in active development and is not considered stable yet. Hence, compatibility breaks might be introduced.

From this point onwards, this document will implicitly refer to version 2 of the library.

4.18.2 Design concepts and interfaces

This section presents some of the key concepts and data structures used in the translation tables library.

mmap regions

An `mmap_region` is an abstract, concise way to represent a memory region to map. It is one of the key interfaces to the library. It is identified by:

- its physical base address;
- its virtual base address;
- its size;
- its attributes;
- its mapping granularity (optional).

See the `struct mmap_region` type in `xlat_tables_v2.h`.

The user usually provides a list of such mmap regions to map and lets the library transpose that in a set of translation tables. As a result, the library might create new translation tables, update or split existing ones.

The region attributes specify the type of memory (for example device or cached normal memory) as well as the memory access permissions (read-only or read-write, executable or not, secure or non-secure, and so on). In the case of the EL1&0 translation regime, the attributes also specify whether the region is a User region (EL0) or Privileged region (EL1). See the `MT_xxx` definitions in `xlat_tables_v2.h`. Note that for the EL1&0 translation regime the Execute Never attribute is set simultaneously for both EL1 and EL0.

The granularity controls the translation table level to go down to when mapping the region. For example, assuming the MMU has been configured to use a 4KB granule size, the library might map a 2MB memory region using either of the two following options:

- using a single level-2 translation table entry;
- using a level-2 intermediate entry to a level-3 translation table (which contains 512 entries, each mapping 4KB).

The first solution potentially requires less translation tables, hence potentially less memory. However, if part of this 2MB region is later remapped with different memory attributes, the library might need to split the existing page tables to refine the mappings. If a single level-2 entry has been used here, a level-3 table will need to be allocated on the fly and the level-2 modified to point to this new level-3 table. This has a performance cost at run-time.

If the user knows upfront that such a remapping operation is likely to happen then they might enforce a 4KB mapping granularity for this 2MB region from the beginning; remapping some of these 4KB pages on the fly then becomes a lightweight operation.

The region's granularity is an optional field; if it is not specified the library will choose the mapping granularity for this region as it sees fit (more details can be found in *The memory mapping algorithm* section below).

Translation Context

The library can create or modify translation tables pertaining to a different translation regime than the exception level the library code is executing at. For example, the library might be used by EL3 software (for instance BL31) to create translation tables pertaining to the S-EL1&0 translation regime.

This flexibility comes from the use of *translation contexts*. A *translation context* constitutes the superset of information used by the library to track the status of a set of translation tables for a given translation regime.

The library internally allocates a default translation context, which pertains to the translation regime of the current exception level. Additional contexts may be explicitly allocated and initialized using the `REGISTER_XLAT_CONTEXT()` macro. Separate APIs are provided to act either on the default translation context or on an alternative one.

To register a translation context, the user must provide the library with the following information:

- A name.

  The resulting translation context variable will be called after this name, to which `_xlat_ctx` is appended. For example, if the macro name parameter is `foo`, the context variable name will be `foo_xlat_ctx`.

- The maximum number of *mmap* regions to map.

  Should account for both static and dynamic regions, if applicable.

- The number of sub-translation tables to allocate.

  Number of translation tables to statically allocate for this context, excluding the initial lookup level translation table, which is always allocated. For example, if the initial lookup level is 1, this parameter would specify the number of level-2 and level-3 translation tables to pre-allocate for this context.

- The size of the virtual address space.

  Size in bytes of the virtual address space to map using this context. This will incidentally determine the number of entries in the initial lookup level translation table : the library will allocate as many entries as is required to map the entire virtual address space.

- The size of the physical address space.

  Size in bytes of the physical address space to map using this context.

The default translation context is internally initialized using information coming (for the most part) from platform-specific defines:

- name: hard-coded to `tf` ; hence the name of the default context variable is `tf_xlat_ctx`;
- number of *mmap* regions: `MAX_MMAP_REGIONS`;
- number of sub-translation tables: `MAX_XLAT_TABLES`;
- size of the virtual address space: `PLAT_VIRT_ADDR_SPACE_SIZE`;
- size of the physical address space: `PLAT_PHY_ADDR_SPACE_SIZE`.

Please refer to the *Porting Guide* for more details about these macros.

Static and dynamic memory regions

The library optionally supports dynamic memory mapping. This feature may be enabled using the `PLAT_XLAT_TABLES_DYNAMIC` platform build flag.

When dynamic memory mapping is enabled, the library categorises mmap regions as *static* or *dynamic*.

- *Static regions* are fixed for the lifetime of the system. They can only be added early on, before the translation tables are created and populated. They cannot be removed afterwards.
- *Dynamic regions* can be added or removed any time.

When the dynamic memory mapping feature is disabled, only static regions exist.

The dynamic memory mapping feature may be used to map and unmap transient memory areas. This is useful when the user needs to access some memory for a fixed period of time, after which the memory may be discarded and reclaimed. For example, a memory region that is only required at boot time while the system is initializing, or to temporarily share a memory buffer between the normal world and trusted world. Note that it is up to the caller to ensure that these regions are not accessed concurrently while the regions are being added or removed.

Although this feature provides some level of dynamic memory allocation, this does not allow dynamically allocating an arbitrary amount of memory at an arbitrary memory location. The user is still required to declare at compile-time the limits of these allocations ; the library will deny any mapping request that does not fit within this pre-allocated pool of memory.

4.18.3 Library APIs

The external APIs exposed by this library are declared and documented in the `xlat_tables_v2.h` header file. This should be the reference point for getting information about the usage of the different APIs this library provides. This section just provides some extra details and clarifications.

Although the `mmap_region` structure is a publicly visible type, it is not recommended to populate these structures by hand. Instead, wherever APIs expect function arguments of type `mmap_region_t`, these should be constructed using the `MAP_REGION*()` family of helper macros. This is to limit the risk of compatibility breaks, should the `mmap_region` structure type evolve in the future.

The `MAP_REGION()` and `MAP_REGION_FLAT()` macros do not allow specifying a mapping granularity, which leaves the library implementation free to choose it. However, in cases where a specific granularity is required, the `MAP_REGION2()` macro might be used instead.

As explained earlier in this document, when the dynamic mapping feature is disabled, there is no notion of dynamic regions. Conceptually, there are only static regions. For this reason (and to retain backward compatibility with the version 1 of the library), the APIs that map static regions do not embed the word *static* in their functions names (for example `mmap_add_region()`), in contrast with the dynamic regions APIs (for example `mmap_add_dynamic_region()`).

Although the definition of static and dynamic regions is not based on the state of the MMU, the two are still related in some way. Static regions can only be added before `init_xlat_tables()` is called and `init_xlat_tables()` must be called while the MMU is still off. As a result, static regions cannot be added once the MMU has been enabled. Dynamic regions can be added with the MMU on or off. In practice, the usual call flow would look like this:

1. The MMU is initially off.
2. Add some static regions, add some dynamic regions.
3. Initialize translation tables based on the list of mmap regions (using one of the `init_xlat_tables*()` APIs).
4. At this point, it is no longer possible to add static regions. Dynamic regions can still be added or removed.
5. Enable the MMU.
6. Dynamic regions can continue to be added or removed.

Because static regions are added early on at boot time and are all in the control of the platform initialization code, the `mmap_add*()` family of APIs are not expected to fail. They do not return any error code.

Nonetheless, these APIs will check upfront whether the region can be successfully added before updating the translation context structure. If the library detects that there is insufficient memory to meet the request, or that the new region will overlap another one in an invalid way, or if any other unexpected error is encountered, they will print an error message on the UART. Additionally, when asserts are enabled (typically in debug builds), an assertion will be triggered. Otherwise, the function call will just return straight away, without adding the offending memory region.

4.18.4 Library limitations

Dynamic regions are not allowed to overlap each other. Static regions are allowed to overlap as long as one of them is fully contained inside the other one. This is allowed for backwards compatibility with the previous behaviour in the version 1 of the library.

4.18.5 Implementation details

Code structure

The library is divided into 4 modules:

- **Core module**

  Provides the main functionality of the library, such as the initialization of translation tables contexts and mapping/unmapping memory regions. This module provides functions such as `mmap_add_region_ctx` that let the caller specify the translation tables context affected by them.

  See `xlat_tables_core.c`.

- **Active context module**

  Instantiates the context that is used by the current BL image and provides helpers to manipulate it, abstracting it from the rest of the code. This module provides functions such as `mmap_add_region`, that directly affect the BL image using them.

  See `xlat_tables_context.c`.

- **Utilities module**

  Provides additional functionality like debug print of the current state of the translation tables and helpers to query memory attributes and to modify them.

  See `xlat_tables_utils.c`.

- **Architectural module**

  Provides functions that are dependent on the current execution state (AArch32/AArch64), such as the functions used for TLB invalidation, setup the MMU, or calculate the Physical Address Space size. They do not need a translation context to work on.

  See `aarch32/xlat_tables_arch.c` and `aarch64/xlat_tables_arch.c`.

From mmap regions to translation tables

A translation context contains a list of `mmap_region_t`, which holds the information of all the regions that are mapped at any given time. Whenever there is a request to map (resp. unmap) a memory region, it is added to (resp. removed from) the `mmap_region_t` list.

The mmap regions list is a conceptual way to represent the memory layout. At some point, the library has to convert this information into actual translation tables to program into the MMU.

Before the `init_xlat_tables()` API is called, the library only acts on the mmap regions list. Adding a static or dynamic region at this point through one of the `mmap_add*()` APIs does not affect the translation tables in any way, they only get registered in the internal mmap region list. It is only when the user calls the `init_xlat_tables()` that the translation tables are populated in memory based on the list of mmap regions registered so far. This is an optimization that allows creation of the initial set of translation tables in one go, rather than having to edit them every time while the MMU is disabled.

After the `init_xlat_tables()` API has been called, only dynamic regions can be added. Changes to the translation tables (as well as the mmap regions list) will take effect immediately.

The memory mapping algorithm

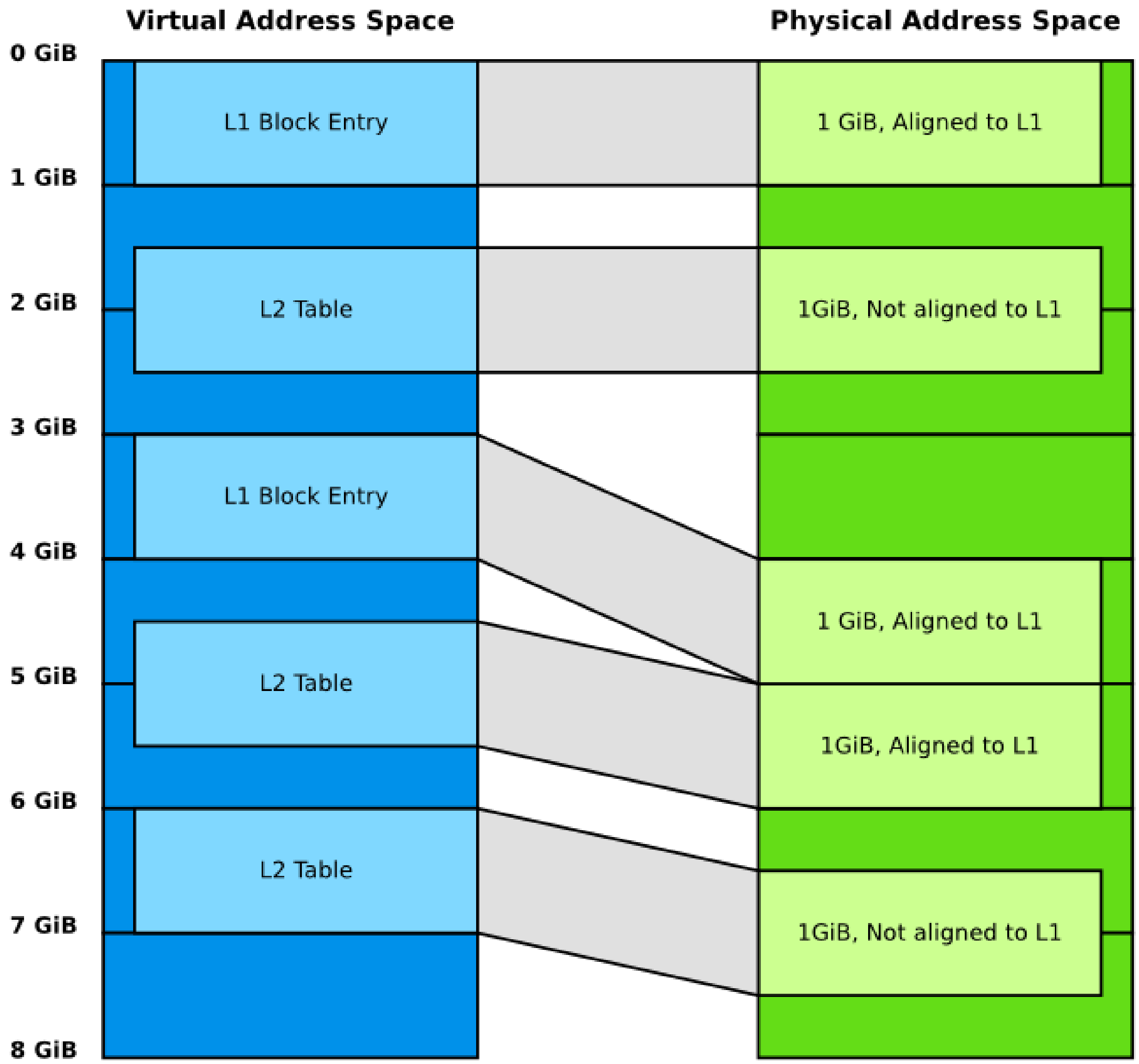
The mapping function is implemented as a recursive algorithm. It is however bound by the level of depth of the translation tables (the Armv8-A architecture allows up to 4 lookup levels).

By default[1], the algorithm will attempt to minimize the number of translation tables created to satisfy the user's request. It will favour mapping a region using the biggest possible blocks, only creating a sub-table if it is strictly necessary. This is to reduce the memory footprint of the firmware.

The most common reason for needing a sub-table is when a specific mapping requires a finer granularity. Misaligned regions also require a finer granularity than what the user may had originally expected, using a lot more memory than expected. The reason is that all levels of translation are restricted to address translations of the same granularity as the size of the blocks of that level. For example, for a 4 KiB page size, a level 2 block entry can only translate up to a granularity of 2 MiB. If the Physical Address is not aligned to 2 MiB then additional level 3 tables are also needed.

Note that not every translation level allows any type of descriptor. Depending on the page size, levels 0 and 1 of translation may only allow table descriptors. If a block entry could be able to describe a translation, but that level does not allow block descriptors, a table descriptor will have to be used instead, as well as additional tables at the next level.

[1] That is, when mmap regions do not enforce their mapping granularity.



The mmap regions are sorted in a way that simplifies the code that maps them. Even though this ordering is only strictly needed for overlapping static regions, it must also be applied for dynamic regions to maintain a consistent order of all regions at all times. As each new region is mapped, existing entries in the translation tables are checked to ensure consistency. Please refer to the comments in the source code of the core module for more details about the sorting algorithm in use.

TLB maintenance operations

The library takes care of performing TLB maintenance operations when required. For example, when the user requests removing a dynamic region, the library invalidates all TLB entries associated to that region to ensure that these changes are visible to subsequent execution, including speculative execution, that uses the changed translation table entries.

A counter-example is the initialization of translation tables. In this case, explicit TLB maintenance is not required. The Armv8-A architecture guarantees that all TLBs are disabled from reset and their contents have no effect on address translation at reset[2]. Therefore, the TLBs invalidation is deferred to the `enable_mmu*()` family of functions, just before the MMU is turned on.

TLB invalidation is not required when adding dynamic regions either. Dynamic regions are not allowed to overlap existing memory region. Therefore, if the dynamic mapping request is deemed legitimate, it automatically concerns memory that was not mapped in this translation regime and the library will have initialized its corresponding translation table entry to an invalid descriptor. Given that the TLBs are not architecturally permitted to hold any invalid translation table entry[3], this means that this mapping cannot be cached in the TLBs.

4.19 Chain of trust bindings

The device tree allows to describe the chain of trust with the help of 'cot' node which contain 'manifests' and 'images' as sub-nodes. 'manifests' and 'images' nodes contains number of sub-nodes (i.e. 'certificate' and 'image' nodes) mentioning properties of the certificate and image respectively.

Also, device tree describes 'non-volatile-counters' node which contains number of sub-nodes mentioning properties of all non-volatile-counters used in the chain of trust.

4.19.1 cot

This is root node which contains 'manifests' and 'images' as sub-nodes

4.19.2 Manifests and Certificate node bindings definition

- **Manifests node**

  Description: Container of certificate nodes.

  PROPERTIES

[2] See section D4.9 `Translation Lookaside Buffers (TLBs)`, subsection `TLB behavior at reset` in Armv8-A, rev C.a.

[3] See section D4.10.1 `General TLB maintenance requirements` in Armv8-A, rev C.a.

- **compatible:**

  Usage: required

  Value type: <string>

  Definition: must be "arm, cert-descs"

- **Certificate node**

  Description:

  Describes certificate properties which are used during the authentication process.

  PROPERTIES

  - **root-certificate**

    Usage:

    Required for the certificate with no parent. In other words, certificates which are validated using root of trust public key.

    Value type: <boolean>

  - **image-id**

    Usage: Required for every certificate with unique id.

    Value type: <u32>

  - **parent**

    Usage:

    It refers to their parent image, which typically contains information to authenticate the certificate. This property is required for all non-root certificates.

    This property is not required for root-certificates as root-certificates are validated using root of trust public key provided by platform.

    Value type: <phandle>

  - **signing-key**

    Usage:

    For non-root certificates, this property is used to refer public key node present in parent certificate node and it is required property for all non-root certificates which are authenticated using public-key present in parent certificate.

    This property is not required for all root-certificates. If omitted, the root certificate will be validated using the default platform ROTPK. If instead the root certificate needs validating using a different ROTPK, the signing-key property should provide a reference to the ROTPK node to use.

    Value type: <phandle>

  - **antirollback-counter**

    Usage:

    This property is used by all certificates which are protected against rollback attacks using a non-volatile counter and it is an optional property.

This property is used to refer one of the non-volatile counter sub-node present in 'non-volatile counters' node.

Value type: <phandle>

SUBNODES

- Description:

  Hash and public key information present in the certificate are shown by these nodes.

- **public key node**

  Description: Provide public key information in the certificate.

  PROPERTIES

  * **oid**

    Usage:

    This property provides the Object ID of public key provided in the certificate with the help of which public key information can be extracted.

    Value type: <string>

- **hash node**

  Description: Provide the hash information in the certificate.

  PROPERTIES

  * **oid**

    Usage:

    This property provides the Object ID of hash provided in the certificate with the help of which hash information can be extracted.

    Value type: <string>

Example:

```
cot {
   manifests {
      compatible = "arm, cert-descs”

      trusted-key-cert: trusted-key-cert {
         root-certificate;
         image-id = <TRUSTED_KEY_CERT_ID>;
         antirollback-counter = <&trusted_nv_ctr>;

         trusted-world-pk: trusted-world-pk {
            oid = TRUSTED_WORLD_PK_OID;
         };
         non-trusted-world-pk: non-trusted-world-pk {
            oid = NON_TRUSTED_WORLD_PK_OID;
         };
      };

      scp_fw_key_cert: scp_fw_key_cert {
```

(continues on next page)

(continued from previous page)

```
            image-id = <SCP_FW_KEY_CERT_ID>;
            parent = <&trusted-key-cert>;
            signing-key = <&trusted_world_pk>;
            antirollback-counter = <&trusted_nv_ctr>;

            scp_fw_content_pk: scp_fw_content_pk {
                oid = SCP_FW_CONTENT_CERT_PK_OID;
            };
        };
        .
        .
        .

        next-certificate {

        };
    };
};
```

4.19.3 Images and Image node bindings definition

- **Images node**

  Description: Container of image nodes

  PROPERTIES

  - **compatible:**

    Usage: required

    Value type: <string>

    Definition: must be "arm, img-descs"

- **Image node**

  Description:

  Describes image properties which will be used during authentication process.

  PROPERTIES

  - **image-id**

    Usage: Required for every image with unique id.

    Value type: <u32>

  - **parent**

    Usage:

    Required for every image to provide a reference to its parent image, which contains the necessary information to authenticate it.

    Value type: <phandle>

  - **hash**
    Usage:

    Required for all images which are validated using hash method. This property is used to refer hash node present in parent certificate node.

    Value type: <phandle>

    Note:

    Currently, all images are validated using 'hash' method. In future, there may be multiple methods can be used to validate the image.

Example:

```
cot {
   images {
      compatible = "arm, img-descs";

      scp_bl2_image {
         image-id = <SCP_BL2_IMAGE_ID>;
         parent = <&scp_fw_content_cert>;
         hash = <&scp_fw_hash>;
      };

      .
      .
      .

      next-img {

      };
   };
};
```

4.19.4 non-volatile counter node binding definition

- **non-volatile counters node**
  Description: Contains properties for non-volatile counters.

  PROPERTIES

  - **compatible:**
    Usage: required

    Value type: <string>

    Definition: must be "arm, non-volatile-counter"

  - **#address-cells**
    Usage: required

    Value type: <u32>

    Definition:

Must be set according to address size of non-volatile counter register

- **#size-cells**

  Usage: required

  Value type: <u32>

  Definition: must be set to 0

SUBNODE

- **counters node**

  Description: Contains various non-volatile counters present in the platform.

PROPERTIES

- **id**

  Usage: Required for every nv-counter with unique id.

  Value type: <u32>

- **reg**

  Usage:

  Register base address of non-volatile counter and it is required property.

  Value type: <u32>

- **oid**

  Usage:

  This property provides the Object ID of non-volatile counter provided in the certificate and it is required property.

  Value type: <string>

Example: Below is non-volatile counters example for ARM platform

```
non_volatile_counters: non_volatile_counters {
     compatible = "arm, non-volatile-counter";
     #address-cells = <1>;
     #size-cells = <0>;

     trusted_nv_ctr: trusted_nv_ctr {
        id  = <TRUSTED_NV_CTR_ID>;
        reg = <TFW_NVCTR_BASE>;
        oid = TRUSTED_FW_NVCOUNTER_OID;
     };

     non_trusted_nv_ctr: non_trusted_nv_ctr {
        id  = <NON_TRUSTED_NV_CTR_ID>;
        reg = <NTFW_CTR_BASE>;
        oid = NON_TRUSTED_FW_NVCOUNTER_OID;
     };
};
```

4.19.5 rot_keys node binding definition

- **rot_keys node**

  Description: Contains root-of-trust keys for the root certificates.

  SUBNODES

  - Description:

    Root of trust key information present in the root certificates are shown by these nodes.

  - **rot key node**

    Description: Provide ROT key information in the certificate.

    PROPERTIES

    * **oid**

      Usage:

      This property provides the Object ID of ROT key provided in the certificate.

      Value type: <string>

Example: Below is rot_keys example for CCA platform

```
rot_keys {
        swd_rot_pk: swd_rot_pk {
           oid = SWD_ROT_PK_OID;
        };

        prot_pk: prot_pk {
           oid = PROT_PK_OID;
        };
};
```

4.19.6 Future update to chain of trust binding

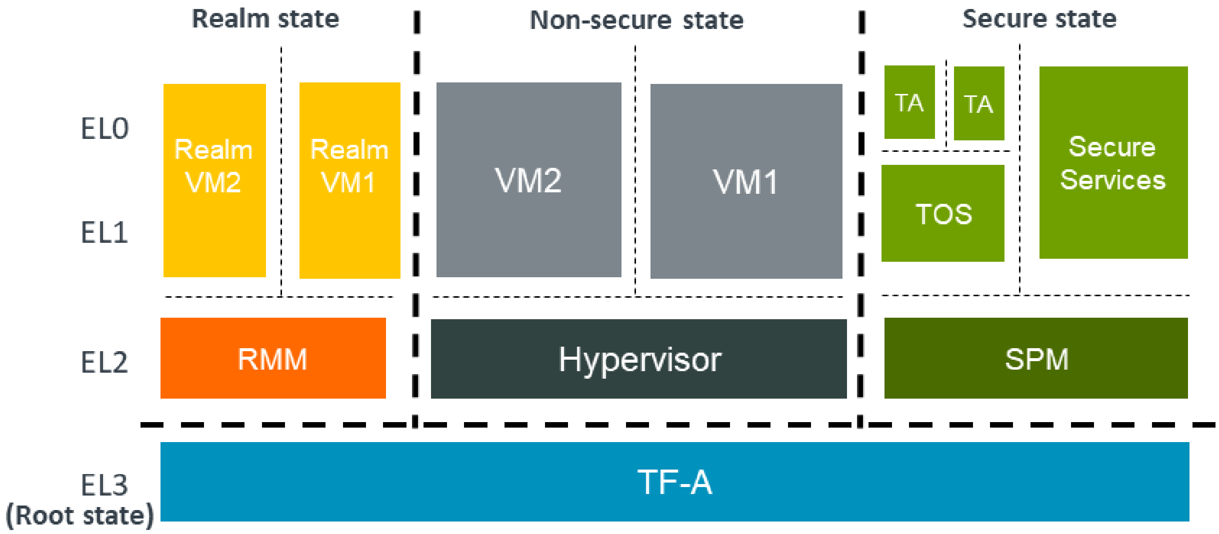
This binding document needs to be revisited to generalise some terminologies which are currently specific to X.509 certificates for e.g. Object IDs.

4.20 Realm Management Extension (RME)

FEAT_RME (or RME for short) is an Armv9-A extension and is one component of the Arm Confidential Compute Architecture (Arm CCA). TF-A supports RME starting from version 2.6. This chapter discusses the changes to TF-A to support RME and provides instructions on how to build and run TF-A with RME.

4.20.1 RME support in TF-A

The following diagram shows an Arm CCA software architecture with TF-A as the EL3 firmware. In the Arm CCA architecture there are two additional security states and address spaces: `Root` and `Realm`. TF-A firmware runs in the Root world. In the realm world, a Realm Management Monitor firmware (RMM) manages the execution of Realm VMs and their interaction with the hypervisor.



RME is the hardware extension to support Arm CCA. To support RME, various changes have been introduced to TF-A. We discuss those changes below.

Changes to translation tables library

RME adds Root and Realm Physical address spaces. To support this, two new memory type macros, `MT_ROOT` and `MT_REALM`, have been added to the *Translation (XLAT) Tables Library*. These macros are used to configure memory regions as Root or Realm respectively.

Note: Only version 2 of the translation tables library supports the new memory types.

Changes to context management

A new CPU context for the Realm world has been added. The existing can be used to manage Realm context.

Boot flow changes

In a typical TF-A boot flow, BL2 runs at Secure-EL1. However when RME is enabled, TF-A runs in the Root world at EL3. Therefore, the boot flow is modified to run BL2 at EL3 when RME is enabled. In addition to this, a Realm-world firmware (RMM) is loaded by BL2 in the Realm physical address space.

The boot flow when RME is enabled looks like the following:

1. BL1 loads and executes BL2 at EL3
2. BL2 loads images including RMM
3. BL2 transfers control to BL31
4. BL31 initializes SPM (if SPM is enabled)
5. BL31 initializes RMM
6. BL31 transfers control to Normal-world software

Granule Protection Tables (GPT) library

Isolation between the four physical address spaces is enforced by a process called Granule Protection Check (GPC) performed by the MMU downstream any address translation. GPC makes use of Granule Protection Table (GPT) in the Root world that describes the physical address space assignment of every page (granule). A GPT library that provides APIs to initialize GPTs and to transition granules between different physical address spaces has been added. More information about the GPT library can be found in the *Granule Protection Tables Library* chapter.

RMM Dispatcher (RMMD)

RMMD is a new standard runtime service that handles the switch to the Realm world. It initializes the RMM and handles Realm Management Interface (RMI) SMC calls from Non-secure.

There is a contract between RMM and RMMD that defines the arguments that the former needs to take in order to initialize and also the possible return values. This contract is defined in the RMM Boot Interface, which can be found at *RMM Boot Interface*.

There is also a specification of the runtime services provided by TF-A to RMM. This can be found at *RMM-EL3 Runtime Interface*.

Test Realm Payload (TRP)

TRP is a small test payload that runs at R-EL2 and implements a subset of the Realm Management Interface (RMI) commands to primarily test EL3 firmware and the interface between R-EL2 and EL3. When building TF-A with RME enabled, if the path to an RMM image is not provided, TF-A builds the TRP by default and uses it as the R-EL2 payload.

4.20.2 Building and running TF-A with RME

The recommended procedure for building and running an RME enabled reference platform is by the use of Shrinkwrap.

The tool offers examples of Three-worlds and Four-worlds configurations for running on the *FVP_Base_RevC-2xAEMvA* model.

- Three-world execution: this is the configuration to use if Secure world functionality is not needed.
- Four-world execution: this is the configuration to use if both Secure and Realm world functionality is needed.

TF-A is tested with the following software entities in each world as listed below:

- NS Host (RME capable Linux),
- EL3 Root (TF-A)
- R-EL2 (RMM)
- S-EL2 (SPM / Hafnium) in a 4-worlds configuration

Additionally, the TF-RMM project documentation has the specific Shrinkwrap guide.

TF-A CI pipeline includes coverage for building TF-A along with TF-RMM and SPM/Hafnium for running 3 worlds and 4 worlds configurations. In those cases, the normal world payload and test suite originates from TF-a-tests project.

4.21 RMM-EL3 Communication interface

This document defines the communication interface between RMM and EL3. There are two parts in this interface: the boot interface and the runtime interface.

The Boot Interface defines the ABI between EL3 and RMM when the CPU enters R-EL2 for the first time after boot. The cold boot interface defines the ABI for the cold boot path and the warm boot interface defines the same for the warm path.

The RMM-EL3 runtime interface defines the ABI for EL3 services which can be invoked by RMM as well as the register save-restore convention when handling an SMC call from NS.

The below sections discuss these interfaces more in detail.

4.21.1 RMM-EL3 Interface versioning

The RMM Boot and Runtime Interface uses a version number to check compatibility with the register arguments passed as part of Boot Interface and RMM-EL3 runtime interface.

The Boot Manifest, discussed later in section *Boot Manifest*, uses a separate version number but with the same scheme.

The version number is a 32-bit type with the following fields:

Bits	Value
[0:15]	`VERSION_MINOR`
[16:30]	`VERSION_MAJOR`
[31]	RES0

The version numbers are sequentially increased and the rules for updating them are explained below:

- `VERSION_MAJOR`: This value is increased when changes break compatibility with previous versions. If the changes on the ABI are compatible with the previous one, `VERSION_MAJOR` remains unchanged.
- `VERSION_MINOR`: This value is increased on any change that is backwards compatible with the previous version. When `VERSION_MAJOR` is increased, `VERSION_MINOR` must be set to 0.
- `RES0`: Bit 31 of the version number is reserved 0 as to maintain consistency with the versioning schemes used in other parts of RMM.

This document specifies the 2.0 version of Boot Interface ABI and RMM-EL3 services specification and the 0.5 version of the Boot Manifest.

4.21.2 RMM Boot Interface

This section deals with the Boot Interface part of the specification.

One of the goals of the Boot Interface is to allow EL3 firmware to pass down into RMM certain platform specific information dynamically. This allows RMM to be less platform dependent and be more generic across platform variations. It also allows RMM to be decoupled from the other boot loader images in the boot sequence and remain agnostic of any particular format used for configuration files.

The Boot Interface ABI defines a set of register conventions and also a memory based manifest file to pass information from EL3 to RMM. The Boot Manifest and the associated platform data in it can be dynamically created by EL3 and there is no restriction on how the data can be obtained (e.g by DTB, hoblist or other).

The register convention and the manifest are versioned separately to manage future enhancements and compatibility.

RMM completes the boot by issuing the `RMM_BOOT_COMPLETE` SMC (0xC40001CF) back to EL3. After the RMM has finished the boot process, it can only be entered from EL3 as part of RMI handling.

If RMM returns an error during boot (in any CPU), then RMM must not be entered from any CPU.

Cold Boot Interface

During cold boot RMM expects the following register values:

Register	Value
x0	Linear index of this PE. This index starts from 0 and must be less than the maximum number of CPUs to be supported at runtime (see x2).
x1	Version for this Boot Interface as defined in *RMM-EL3 Interface versioning*.
x2	Maximum number of CPUs to be supported at runtime. RMM should ensure that it can support this maximum number.
x3	Base address for the shared buffer used for communication between EL3 firmware and RMM. This buffer must be of 4KB size (1 page). The Boot Manifest must be present at the base of this shared buffer during cold boot.
x4	Activation token. Should be set to 0 on the initial boot of the system. For a subsequent warm boot or when using Live Firmware Activation, the activation token should be set to the value returned by RMM for this CPU during the initial boot (in x2).

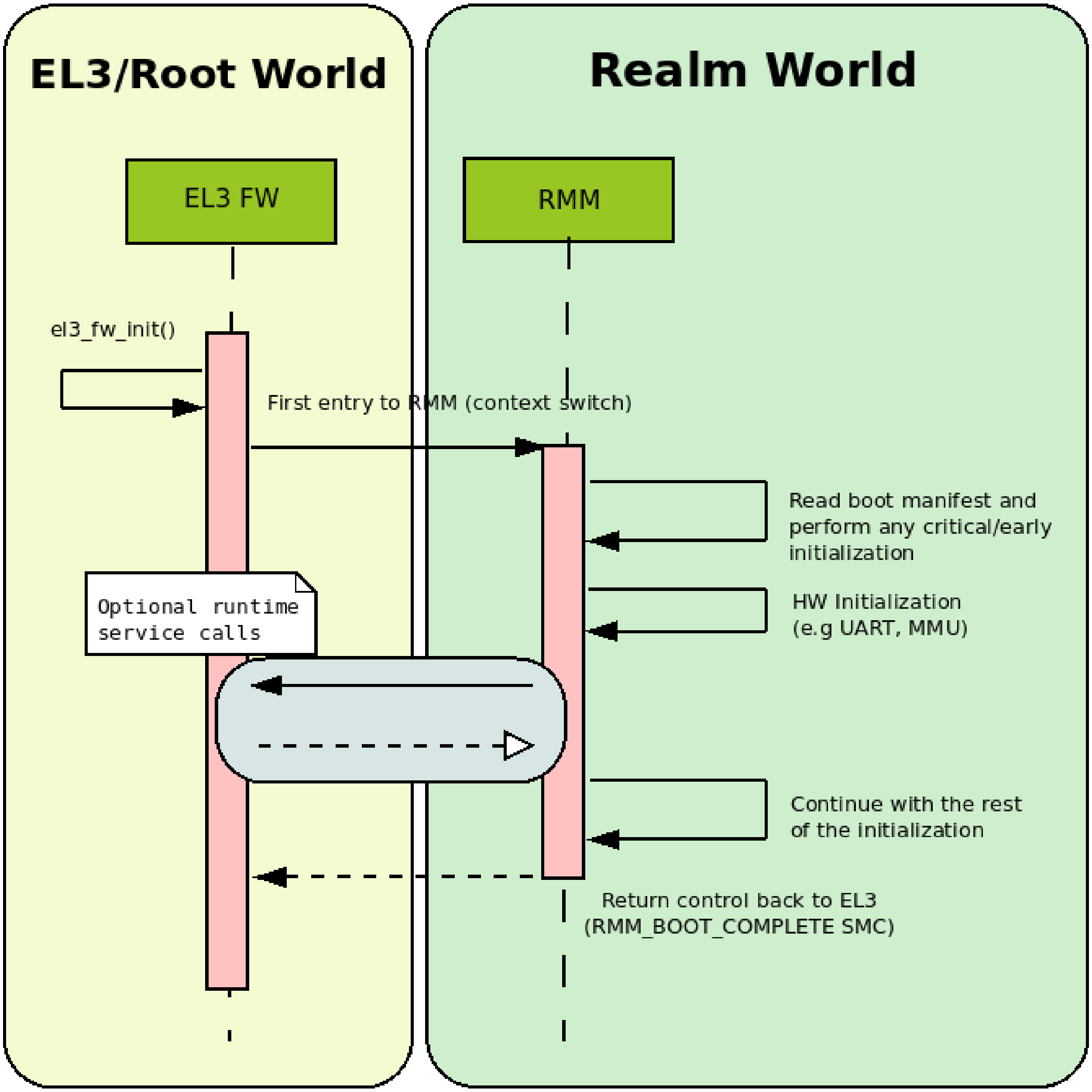
During cold boot, EL3 firmware needs to allocate a 4KB page that will be passed to RMM in x3. This memory will be used as shared buffer for communication between EL3 and RMM. It must be assigned to Realm world and must be mapped with Normal memory attributes (IWB-OWB-ISH) at EL3. At boot, this memory will be used to populate the Boot Manifest. Since the Boot Manifest can be accessed by RMM prior to enabling its MMU, EL3 must ensure that proper cache maintenance operations are performed after the Boot Manifest is populated.

EL3 should also ensure that this shared buffer is always available for use by RMM during the lifetime of the system and that it can be used for runtime communication between RMM and EL3. For example, when RMM invokes attestation service commands in EL3, this buffer can be used to exchange data between RMM and EL3. It is also allowed for RMM to invoke runtime services provided by EL3 utilizing this buffer during the boot phase, prior to return back to EL3 via RMM_BOOT_COMPLETE SMC.

RMM should map this memory page into its Stage 1 page-tables using Normal memory attributes.

During runtime, it is the RMM which initiates any communication with EL3. If that communication requires the use of the shared area, it is expected that RMM needs to do the necessary concurrency protection to prevent the use of the same buffer by other PEs.

The following sequence diagram shows how a generic EL3 Firmware would boot RMM.



Warm Boot Interface

At warm boot, RMM is already initialized and only some per-CPU initialization is still pending. The only argument that is required by RMM at this stage is the CPU Id, which will be passed through register x0 whilst x1 to x3 are RES0. This is summarized in the following table:

Register	Value
x0	Linear index of this PE. This index starts from 0 and must be less than the maximum number of CPUs to be supported at runtime (see x2).
x1	Activation token. Should be set to 0 on the initial boot of the system. For a subsequent warm boot or when using Live Firmware Activation, the activation token should be set to the value returned by RMM for this CPU during the initial boot (in x2).
x2 - x3	RES0

Boot error handling and return values

After boot up and initialization, RMM returns control back to EL3 through a `RMM_BOOT_COMPLETE` SMC call. The first argument of this SMC call will be returned in x1 and it will encode a signed integer with the error reason. x2 will contain the per-CPU activation token, which is an opaque value that should be passed back to RMM when doing Live Firmware Activations or on a subsequent warm boot. The following table describes possible values for the error code in x1:

Error code	Description	ID
`E_RMM_BOOT_SUCCESS`	Boot successful	0
`E_RMM_BOOT_ERR_UNKNOWN`	Unknown error	-1
`E_RMM_BOOT_VERSION_NOT_VALID`	Boot Interface version reported by EL3 is not supported by RMM	-2
`E_RMM_BOOT_CPUS_OUT_OF_RANGE`	Number of CPUs reported by EL3 larger than maximum supported by RMM	-3
`E_RMM_BOOT_CPU_ID_OUT_OF_RANGE`	Current CPU Id is higher or equal than the number of CPUs supported by RMM	-4
`E_RMM_BOOT_INVALID_SHARED_BUFFER`	Invalid pointer to shared memory area	-5
`E_RMM_BOOT_MANIFEST_VERSION_NOT_SUPPORTED`	Version reported by the Boot Manifest not supported by RMM	-6
`E_RMM_BOOT_MANIFEST_DATA_ERROR`	Error parsing core Boot Manifest	-7

For any error detected in RMM during cold or warm boot, RMM will return back to EL3 using `RMM_BOOT_COMPLETE` SMC with an appropriate error code. It is expected that EL3 will take necessary action to disable Realm world for further entry from NS Host on receiving an error. This will be done across all the PEs in the system so as to present a symmetric view to the NS Host. Any further warm boot by any PE should not enter RMM using the warm boot interface.

Boot Manifest

During cold boot, EL3 Firmware passes a memory Boot Manifest to RMM containing platform information.

This Boot Manifest is versioned independently of the Boot Interface, to help evolve the former independent of the latter. The current version for the Boot Manifest is `v0.4` and the rules explained in *RMM-EL3 Interface versioning* apply on this version as well.

The Boot Manifest v0.4 has the following fields:

- version : Version of the Manifest (v0.4)
- plat_data : Pointer to the platform specific data and not specified by this document. These data are optional and can be NULL.
- plat_dram : Structure encoding the NS DRAM information on the platform. This field is optional and platform can choose to zero out this structure if RMM does not need EL3 to send this information during the boot.
- plat_console : Structure encoding the list of consoles for RMM use on the platform. This field is optional and platform can choose to not populate the console list if this is not needed by the RMM for this platform.

For the current version of the Boot Manifest, the core manifest contains a pointer to the platform data. EL3 must ensure that the whole Boot Manifest, including the platform data, if available, fits inside the RMM EL3 shared buffer.

For the data structure specification of Boot Manifest, refer to *RMM-EL3 Boot Manifest structure*

4.21.3 RMM-EL3 Runtime Interface

This section defines the RMM-EL3 runtime interface which specifies the ABI for EL3 services expected by RMM at runtime as well as the register save and restore convention between EL3 and RMM as part of RMI call handling. It is important to note that RMM is allowed to invoke EL3-RMM runtime interface services during the boot phase as well. The EL3 runtime service handling must not result in a world switch to another world unless specified. Both the RMM and EL3 are allowed to make suitable optimizations based on this assumption.

If the interface requires the use of memory, then the memory references should be within the shared buffer communicated as part of the boot interface. See *Cold Boot Interface* for properties of this shared buffer which both EL3 and RMM must adhere to.

RMM-EL3 runtime service return codes

The return codes from EL3 to RMM is a 32 bit signed integer which encapsulates error condition as described in the following table:

Error code	Description	ID
E_RMM_OK	No errors detected	0
E_RMM_UNK	Unknown/Generic error	-1
E_RMM_BAD_ADDR	The value of an address used as argument was invalid	-2
E_RMM_BAD_PAS	Incorrect PAS	-3
E_RMM_NOMEM	Not enough memory to perform an operation	-4
E_RMM_INVAL	The value of an argument was invalid	-5
E_RMM_AGAIN	The resource is busy. Try again.	-6

If multiple failure conditions are detected in an RMM to EL3 command, then EL3 is allowed to return an error code corresponding to any of the failure conditions.

RMM-EL3 runtime services

The following table summarizes the RMM runtime services that need to be implemented by EL3 Firmware.

FID	Command
0xC400018F	RMM_RMI_REQ_COMPLETE
0xC40001B0	RMM_GTSI_DELEGATE
0xC40001B1	RMM_GTSI_UNDELEGATE
0xC40001B2	RMM_ATTEST_GET_REALM_KEY
0xC40001B3	RMM_ATTEST_GET_PLAT_TOKEN
0xC40001B4	RMM_EL3_FEATURES
0xC40001B5	RMM_EL3_TOKEN_SIGN
0xC40001B6	RMM_MECID_KEY_UPDATE
0xC40001B7	RMM_IDE_KEY_PROG
0xC40001B8	RMM_IDE_KEY_SET_GO
0xC40001B9	RMM_IDE_KEY_SET_STOP
0xC40001BA	RMM_IDE_KM_PULL_RESPONSE
0xC40001BB	RMM_RESERVE_MEMORY

RMM_RMI_REQ_COMPLETE command

Notifies the completion of an RMI call to the Non-Secure world.

This call is the only function currently in RMM-EL3 runtime interface which results in a world switch to NS. This call is the reply to the original RMI call and it is forwarded by EL3 to the NS world.

FID

`0xC400018F`

Input values

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
err_code	x1	[63:0]	RmiCommandReturnCode	Error code returned by the RMI service invoked by NS World. See Realm Management Monitor specification for more info

Output values

This call does not return.

Failure conditions

Since this call does not return to RMM, there is no failure condition which can be notified back to RMM.

RMM_GTSI_DELEGATE command

Delegate a memory granule by changing its PAS from Non-Secure to Realm.

FID

`0xC40001B0`

Input values

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
base_pa	x1	[63:0]	Address	PA of the start of the granule to be delegated

Output values

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

ID	Condition
`E_RMM_BAD_ADDR`	`PA` does not correspond to a valid granule address
`E_RMM_BAD_PAS`	The granule pointed by `PA` does not belong to Non-Secure PAS
`E_RMM_OK`	No errors detected

RMM_GTSI_UNDELEGATE command

Undelegate a memory granule by changing its PAS from Realm to Non-Secure.

FID

`0xC40001B1`

Input values

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
base_pa	x1	[63:0]	Address	PA of the start of the granule to be undelegated

Output values

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

ID	Condition
`E_RMM_BAD_ADDR`	`PA` does not correspond to a valid granule address
`E_RMM_BAD_PAS`	The granule pointed by `PA` does not belong to Realm PAS
`E_RMM_OK`	No errors detected

RMM_ATTEST_GET_REALM_KEY command

Retrieve the Realm Attestation Token Signing key from EL3.

FID

`0xC40001B2`

Input values

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
buf_pa	x1	[63:0]	Address	PA where the Realm Attestation Key must be stored by EL3. The PA must belong to the shared buffer
buf_size	x2	[63:0]	Size	Size in bytes of the Realm Attestation Key buffer. `bufPa + bufSize` must lie within the shared buffer
ecc_curve	x3	[63:0]	Enum	Type of the elliptic curve to which the requested attestation key belongs to. See *Supported ECC Curves*

Output values

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status
keySize	x1	[63:0]	Size	Size of the Realm Attestation Key

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

ID	Condition
`E_RMM_BAD_ADDR`	`PA` is outside the shared buffer
`E_RMM_INVAL`	`PA + BSize` is outside the shared buffer
`E_RMM_INVAL`	`Curve` is not one of the listed in *Supported ECC Curves*
`E_RMM_UNK`	An unknown error occurred whilst processing the command
`E_RMM_OK`	No errors detected

Supported ECC Curves

ID	Curve
0	ECC SECP384R1

RMM_ATTEST_GET_PLAT_TOKEN command

Retrieve the Platform Token from EL3. If the entire token does not fit in the buffer, EL3 returns a hunk of the token (via `tokenHunkSize` parameter) and indicates the remaining bytes that are pending retrieval (via `remainingSize` parameter). The challenge object for the platform token must be populated in the buffer for the first call of this command and the size of the object is indicated by `c_size` parameter. Subsequent calls to retrieve remaining hunks of the token must be made with `c_size` as 0.

If `c_size` is not 0, this command could cause regeneration of platform token and will return token hunk corresponding to beginning of the token.

It is valid for the calls of this command to return `E_RMM_AGAIN` error, which is an indication to the caller to retry this command again. Depending on the platform, this mechanism can be used to implement queuing to HES, if HES is involved in platform token generation.

FID

`0xC40001B3`

Input values

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
buf_pa	x1	[63:0]	Address	PA of the platform attestation token. The challenge object must be passed in this buffer for the first call of this command. Any subsequent calls, if required to retrieve the full token, should not have this object. The PA must belong to the shared buffer.
buf_size	x2	[63:0]	Size	Size in bytes of the platform attestation token buffer. `bufPa + bufSize` must lie within the shared buffer
c_size	x3	[63:0]	Size	Size in bytes of the challenge object. It corresponds to the size of one of the defined SHA algorithms. Any subsequent calls, if required to retrieve the full token, should set this size to 0.

Output values

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status
tokenHunkSize	x1	[63:0]	Size	Size of the platform token hunk retrieved
remainingSize	x2	[63:0]	Size	Remaining bytes of the token that are pending retrieval

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

ID	Condition
`E_RMM_AGAIN`	Resource for Platform token retrieval is busy. Try again.
`E_RMM_BAD_ADDR`	`PA` is outside the shared buffer
`E_RMM_INVAL`	`PA + BSize` is outside the shared buffer
`E_RMM_INVAL`	`CSize` does not represent the size of a supported SHA algorithm for the first call to this command
`E_RMM_INVAL`	`CSize` is not 0 for subsequent calls to retrieve remaining hunks of the token
`E_RMM_UNK`	An unknown error occurred whilst processing the command
`E_RMM_OK`	No errors detected

RMM_EL3_FEATURES command

This command provides a mechanism to discover features and ABIs supported by the RMM-EL3 interface, for a given version. This command is helpful when there are platform specific optional RMM-EL3 interfaces and features exposed by vendor specific EL3 firmware, and a generic RMM that can modify its behavior based on discovery of EL3 features.

The features can be discovered by specifying the feature register index that has fields defined to indicate presence or absence of features and other relevant information. The feature register index is specified in the `feat_reg_idx` parameter. Each feature register is a 64 bit register.

This command is available from v0.4 of the RMM-EL3 interface.

The following is the register definition for feature register index 0 for v0.4 of the interface:

RMM-EL3 Feature Resister 0

```
63      32      31      16       15       8        7        1        0
+-------+-------+-------+--------+--------+--------+--------+--------+
|       |       |       |        |        |        |        |        |
|       |       |       |        |        |        |        |        |
+-------+-------+-------+--------+--------+--------+--------+--------+
                                                            ^
                                                            |
                                                  RMMD_EL3_TOKEN_SIGN
```

Bit Fields:

- **Bit 0:** ***RMMD_EL3_TOKEN_SIGN***
  - When set to 1, the *RMMD_EL3_TOKEN_SIGN* feature is enabled.
  - When cleared (0), the feature is disabled.
- **Bits [1:63]**: Reserved (must be zero)

FID

`0xC40001B4`

Input values

Table 1: Input values for RMM_EL3_FEATURES

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
feat_reg_idx	x1	[63:0]	UInt64	Feature register index. For v0.4, a value of 0 is the only acceptable value

Output values

Table 2: Output values for RMM_EL3_FEATURES

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status
feat_reg	x1	[63:0]	Value	Value of the register as defined above

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

Table 3: Failure conditions for RMM_EL3_FEATURES

ID	Condition
`E_RMM_INVAL`	`feat_reg_idx` is out of valid range
`E_RMM_UNK`	if the SMC is not present, if interface version is <0.4
`E_RMM_OK`	No errors detected

RMM_EL3_TOKEN_SIGN command

This command is an optional command that can be discovered using the RMM_EL3_FEATURES command. This command is used to send requests related to realm attestation token signing requests to EL3. The command supports 3 opcodes:

- RMM_EL3_TOKEN_SIGN_PUSH_REQ_OP
- RMM_EL3_TOKEN_SIGN_PULL_RESP_OP
- RMM_EL3_TOKEN_SIGN_GET_RAK_PUB_OP

The above opcodes can be used to send realm attestation token signing requests to EL3 and get their response, so that the realm attestation token can be constructed.

This command is useful when the RMM may not have access to the private portion of the realm attestation key and needs signing services from EL3 or CCA HES, or other platform specific mechanisms to perform signing.

The RMM-EL3 interface for this command is modeled as two separate queues, one for signing requests and one for retrieving the signed responses. It is possible that the queue in EL3 is full or EL3 is busy and unable to service the RMM requests, in which case the RMM is expected to retry the push operation for requests and pop operation for responses.

FID

`0xC40001B5`

Input values

Table 4: Input values for RMM_EL3_TOKEN_SIGN

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
opcode	x1	[63:0]	UInt64	Opcode that is one of: • RMM_EL3_TOKEN_SIGN_PUSH_REQ_OP: 0x1 - Opcode to push a token signing request to EL3 using struct el3_token_sign_request as described above • RMM_EL3_TOKEN_SIGN_PULL_RESP_OP: 0x2 - Opcode to pull a token signing response from EL3 using struct el3_token_sign_response as described above • RMM_EL3_TOKEN_SIGN_GET_RAK_PUB_OP: 0x3 - Opcode to get the realm attestation public key
buf_pa	x2	[63:0]	Address	PA where the request structure is stored for the opcode RMM_EL3_TOKEN_SIGN_PUSH_REQ_OP, the response structure needs to be populated for the opcode RMM_EL3_TOKEN_SIGN_PULL_RESP_OP, or where the public key must be populated for the opcode RMM_EL3_TOKEN_SIGN_GET_RAK_PUB_OP. The PA must belong to the RMM-EL3 shared buffer
buf_size	x3	[63:0]	Size	Size in bytes of the input buffer in `buf_pa`. `buf_pa + buf_size` must lie within the shared buffer
ecc_curve	x4	[63:0]	Enum	Type of the elliptic curve to which the requested attestation key belongs to. See *Supported ECC Curves*. This parameter is valid on for the opcode RMM_EL3_TOKEN_SIGN_GET_RAK_PUB_OP

Output values

Table 5: Output values for RMM_EL3_TOKEN_SIGN

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status. Valid for all opcodes listed in input values
retval1	x1	[63:0]	Value	If opcode is RMM_EL3_TOKEN_SIGN_GET_RAK_PUB_OP, then returns length of public key returned. Otherwise, reserved

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

Table 6: Failure conditions for RMM_EL3_TOKEN_SIGN

ID	Condition
`E_RMM_INVAL`	if opcode is invalid or buffer address and length passed to the EL3 are not in valid range corresponding to the RMM-EL3 shared buffer, or if the curve used for opcode RMM_EL3_TOKEN_SIGN_GET_RAK_PUB_OP is not the ECC P384 curve
`E_RMM_UNK`	if the SMC is not present, if interface version is <0.4
`E_RMM_AGAIN`	For opcode RMM_EL3_TOKEN_SIGN_PUSH_REQ_OP, if the request is not queued since the EL3 queue is full, or if the response is not ready yet, for other opcodes
`E_RMM_OK`	No errors detected

RMM_MEC_REFRESH command

This command updates the tweak for the encryption key/programs a new encryption key associated with a given MECID. After the execution of this command, all memory accesses associated with the MECID are encrypted/decrypted using the new key. This command is available from v0.8 of the RMM-EL3 interface.

FID

`0xC40001B6`

Input values

Table 7: Input values for RMM_MEC_REFRESH

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
mecid	x1	[47:32]	UInt64	mecid is a 16-bit value between 0 and 65,535 that identifies the MECID for which the encryption key is to be updated. Value has to be a valid MECID as per field MECIDWidthm1 read from MECIDR_EL2. Bits [63:16] must be 0.
mecid	x1	[31:1]	UInt64	Reserved, MBZ
reason	x1	[0]	UInt64	reason is a single bit field used to indicate the reason for the MEC refresh. Values are: 0 (Realm creation), 1 (Realm destruction).

Output values

Table 8: Output values for RMM_MEC_REFRESH

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status. Valid for all opcodes listed in input values

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

Table 9: Failure conditions for RMM_MEC_REFRESH

ID	Condition
`E_RMM_INVAL`	If a field in the x1 register is incorrectly encoded or if MECID is invalid (larger than the common MECID width, determined by MECIDR_EL2.MECIDWidthm1 + 1 or by other system components, whichever is lower)
`E_RMM_UNK`	An unknown error occurred whilst processing the command, FEAT_MEC is not present in hardware or the SMC is not present if the version is < 0.8.
`E_RMM_OK`	No errors detected

RMM_IDE_KEY_PROG command

Set the key/IV info at Root port for an IDE stream as part of Device Assignment flow. This command is available from v0.6 of the RMM-EL3 interface.

Please refer to IDE-KM RFC for description of the IDE setup sequence and how this will be invoked by RMM.

The key is 256 bits and IV is 96 bits. The caller needs to call this SMC to program this key to the Rx, Tx ports and for each sub-stream corresponding to a single keyset.

FID

`0xC40001B7`

Input values

Table 10: Input values for RMM_IDE_KEY_PROG

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
ecam_address	x1	[63:0]	UInt64	Used to identify the root complex(RC)
rp_id	x2	[63:0]	UInt64	Used to identify the root port within the root complex(RC)
Keyset[12]: Dir[11]: Substream[10:8]: StreamID[7:0]	x3	[63:0]	UInt64	IDE selective stream informationKey set: can be 0 or 1unused bits MBZ.
KeqQW0	x4	[63:0]	UInt64	Quad word of key [63:0]
KeqQW1	x5	[63:0]	UInt64	Quad word of key [127:64]
KeqQW2	x6	[63:0]	UInt64	Quad word of key [191:128]
KeqQW3	x7	[63:0]	UInt64	Quad word of key [255:192]
IFVQW0	x8	[63:0]	UInt64	Quad word of IV [63:0]
IFVQW1	x9	[63:0]	UInt64	Quad word of IV [95:64]
request_id	x10	[63:0]	UInt64	Used only in non-blocking mode. Ignored in blocking mode.
cookie	x11	[63:0]	UInt64	Used only in non-blocking mode. Ignored in blocking mode.

Output values

Table 11: Output values for RMM_IDE_KEY_PROG

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

Table 12: Failure conditions for RMM_IDE_KEY_PROG

ID	Condition
`E_RMM_OK`	The Key programming is successful.
`E_RMM_FAULT`	The Key programming is not successful.
`E_RMM_INVAL`	The Key programming arguments are incorrect.
`E_RMM_UNK`	Unknown error or the SMC is not present if the version is < 0.6.
`E_RMM_AGAIN`	Returned only for non-blocking mode. IDE-KM interface is busy or request is full. Retry required.
`E_RMM_INPROGRESS`	Returned only for non-blocking mode. The caller must issue RMM_IDE_KM_PULL_RESPONSE SMC to pull the response.

RMM_IDE_KEY_SET_GO command

Activate the IDE stream at Root Port once the keys have been programmed as part of Device Assignment flow. This command is available from v0.6 of the RMM-EL3 interface.

Please refer to IDE-KM RFC for description of the IDE setup sequence and info on how this will be invoked by RMM.

The caller(RMM) needs to ensure the EL3_IDE_KEY_PROG() call had succeeded prior to this call.

FID

`0xC40001B8`

Input values

Table 13: Input values for RMM_IDE_KEY_SET_GO

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
ecam_address	x1	[63:0]	UInt64	Used to identify the root complex(RC)
rp_id	x2	[63:0]	UInt64	Used to identify the root port within the root complex(RC)
Keyset[12]: Dir[11]: Substream[10:8]: StreamID[7:0]	x3	[63:0]	UInt64	IDE selective stream information. Key set can be 0 or 1. Unused bits MBZ.
request_id	x4	[63:0]	UInt64	Used only in non-blocking mode. Ignored in blocking mode.
cookie	x5	[63:0]	UInt64	Used only in non-blocking mode. Ignored in blocking mode.

Output values

Table 14: Output values for RMM_IDE_KEY_SET_GO

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

Table 15: Failure conditions for RMM_IDE_KEY_SET_GO

ID	Condition
`E_RMM_OK`	The Key set go is successful.
`E_RMM_FAULT`	The Key set go is not successful.
`E_RMM_INVAL`	Incorrect arguments.
`E_RMM_UNK`	Unknown error or the SMC is not present if the version is < 0.6.
`E_RMM_AGAIN`	Returned only for non-blocking mode. IDE-KM interface is busy or request is full. Retry required.
`E_RMM_INPROGRESS`	Returned only for non-blocking mode. The caller must issue RMM_IDE_KM_PULL_RESPONSE SMC to pull the response.

RMM_IDE_KEY_SET_STOP command

Deactivate the IDE stream at Root Port as part of Device Assignment flow. This command is available from v0.6 of the RMM-EL3 interface.

Please refer to IDE-KM RFC for description of the IDE setup sequence and info on how this will be invoked by RMM.

This SMC is used to tear down an IDE Stream.

FID

`0xC40001B9`

Input values

Table 16: Input values for RMM_IDE_KEY_SET_STOP

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
ecam_address	x1	[63:0]	UInt64	Used to identify the root complex(RC)
rp_id	x2	[63:0]	UInt64	Used to identify the root port within the root complex(RC)
Keyset[12]: Dir[11]: Substream[10:8]: StreamID[7:0]	x3	[63:0]	UInt64	IDE selective stream information. Key set can be 0 or 1. Unused bits MBZ.
request_id	x4	[63:0]	UInt64	Used only in non-blocking mode. Ignored in blocking mode.
cookie	x5	[63:0]	UInt64	Used only in non-blocking mode. Ignored in blocking mode.

Output values

Table 17: Output values for RMM_IDE_KEY_SET_STOP

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

Table 18: Failure conditions for RMM_IDE_KEY_SET_STOP

ID	Condition
`E_RMM_OK`	The Key set stop is successful.
`E_RMM_FAULT`	The Key set stop is not successful.
`E_RMM_INVAL`	Incorrect arguments.
`E_RMM_UNK`	Unknown error or the SMC is not present if the version is < 0.6.
`E_RMM_AGAIN`	Returned only for non-blocking mode. IDE-KM interface is busy or request is full. Retry required.
`E_RMM_INPROGRESS`	Returned only for non-blocking mode. The caller must issue RMM_IDE_KM_PULL_RESPONSE SMC to pull the response.

RMM_IDE_KM_PULL_RESPONSE command

Retrieve the response from Root Port to a previous non-blocking IDE-KM SMC request as part of Device Assignment flow. This command is available from v0.6 of the RMM-EL3 interface.

Please refer to IDE-KM RFC for description of the IDE setup sequence and info on how this will be invoked by RMM.

The response from this call could correspond to any of the last pending requests and the RMM needs to identify the request and populate the response. For blocking calls, this SMC always returns E_RMM_UNK.

FID

`0xC40001BA`

Input values

Table 19: Input values for RMM_IDE_KM_PULL_RESPONSE

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
ecam_address	x1	[63:0]	UInt64	Used to identify the root complex(RC)
rp_id	x2	[63:0]	UInt64	Used to identify the root port within the root complex(RC)

Output values

Table 20: Output values for RMM_IDE_KM_PULL_RESPONSE

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status
Result	x1	[63:0]	Error Code	Retrieved response corresponding to previous IDE_KM requests.
Result	x2	[63:0]	value	passthrough from requested SMC
Result	x3	[63:0]	value	passthrough from requested SMC

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

Table 21: Failure conditions for RMM_IDE_KM_PULL_RESPONSE(x0)

ID	Condition
`E_RMM_OK`	Response is retrieved successfully.
`E_RMM_INVAL`	Arguments to pull response SMC is not correct.
`E_RMM_UNK`	Unknown error or the SMC is not present if the version is < 0.6.
`E_RMM_AGAIN`	IDE-KM response queue is empty and no response is available.

Table 22: Failure conditions for RMM_IDE_KM_PULL_RESPONSE(x1)

ID	Condition
`E_RMM_OK`	The previous request was successful.
`E_RMM_FAULT`	The previous request was not successful.
`E_RMM_INVAL`	Arguments to previous request were incorrect.
`E_RMM_UNK`	Previous request returned unknown error.

RMM_RESERVE_MEMORY command

This command is used to reserve memory for the RMM, during RMM boot time. This is not a fully featured dynamic memory allocator, since reservations cannot be freed again, and they must happen during the cold/warm boot phase of RMM. However it allows to size data structures in RMM based on runtime decisions, for instance depending on the number of cores or the amount of memory installed. This command is available from v0.7 of the RMM-EL3 interface.

FID

`0xC40001BB`

Input values

Table 23: Input values for RMM_RESERVE_MEMORY

Name	Register	Field	Type	Description
fid	x0	[63:0]	UInt64	Command FID
size	x1	[63:0]	Size	required size of the memory region, in bytes
args	x2	[63:56]	UInt64	alignment requirement, in bits. A value of 16 would return a 64 KB aligned base address.
args	x2	[55:32]	UInt64	reserved
args	x2	[31:1]	UInt64	flags (reserved)
args	x2	[0]	UInt64	flags: local CPU: Determines whether the reservation should be taken from a pool close to the calling CPU.

Output values

Table 24: Output values for RMM_RESERVE_MEMORY

Name	Register	Field	Type	Description
Result	x0	[63:0]	Error Code	Command return status.
address	x1	[63:0]	Address	Physical address of the reserved memory area.

Failure conditions

The table below shows all the possible error codes returned in `Result` upon a failure. The errors are ordered by condition check.

Table 25: Failure conditions for RMM_RESERVE_MEMORY

ID	Condition
`E_RMM_INVAL`	unrecognised flag bit
`E_RMM_UNK`	if the SMC is not present, if interface version is <0.7
`E_RMM_NOMEM`	size of region is larger than the available memory
`E_RMM_OK`	No errors detected

4.21.4 EL3–RMM Register Context Management Contract

As part of the Non-secure (NS) <-> Realm world switch, EL3 shall maintain a separate register context for each world. During a world transition, EL3 shall save and restore the required register state to ensure correct execution, isolation, and security.

Register Context Managed by EL3

EL3 shall maintain a distinct register context per world and shall save and restore the following registers during a world switch.

General-Purpose and Stack Registers

- General-purpose registers: `x0-x30`
- Stack pointers: * `SP_EL0` * `SP_EL2`

EL2 System Registers

EL3 shall save and restore all EL2 system registers corresponding to features enabled by EL3 (that is, registers with the `_EL2` architectural suffix), excluding the registers listed below (see *Registers Not Managed by EL3*).

Pointer Authentication Registers

EL3 shall save and restore the following Pointer Authentication (PAuth) key registers:

- `APIA`
- `APIB`
- `APDA`
- `APDB`
- `APGA`

Registers Not Managed by EL3

EL3 shall not save or restore the registers listed below. If the Realm world makes use of any of the registers below, the Realm Management Monitor (RMM) is responsible for ensuring their correct preservation.

EL2 Registers Excluded from EL3 Management

- EL2 physical timer registers
- EL2 virtual timer registers
- `ZCR_EL2`
- `ICC_SRE_EL2` if interface version is >= 1.0
- `ICH_HCR_EL2` if interface version is >= 1.0
- `ICH_VMCR_EL2` if interface version is >= 1.0

Additional Registers Outside EL3 Scope

- FP/SIMD registers
- SVE registers
- SME registers
- EL1 and EL0 system registers (except the PAuth key registers listed above)

SMCCC Compliance

During SMC forwarding between the Non-secure world and the Realm world:

- EL3 shall forward registers `x0-x7` from the Non-secure context to the Realm context as SMC input arguments.
- EL3 populates return values in registers `x0-x7` on return back to Non-Secure.

In accordance with SMCCC v1.2:

- Register `x4-x7` shall be preserved if it is not used to return a result.
- The responsibility for preserving `x4-x7`, when required, lies with RMM.

In accordance with SMCCC v1.3:

- The Non-secure world may specify whether SVE context is in use.
- If the SVE context is marked as not in use by the Non-secure world, RMM may choose not to preserve the incoming SVE state, even if SVE registers are used during Realm execution. Before returning to the Non-secure world, the SVE register state is cleared in this case
- The same requirements apply to SME registers.

Note:

Communication between NS world and RMM as defined by the RMM specification expects a maximum of registers x0–x7 for input arguments and returns results in x0–x7. If a future RMM specification requires extended input or output beyond this register set, this section of EL3–RMM contract will be updated.

Security Requirement

EL3 shall adhere to this contract to preserve the confidentiality and integrity of the Realm world during world transitions. Non-compliance may cause state corruption or cross-world information leakage.

4.21.5 Types

RMM-EL3 Boot Manifest structure

The RMM-EL3 Boot Manifest v0.5 structure contains platform boot information passed from EL3 to RMM. The size of the Boot Manifest is 160 bytes.

The members of the RMM-EL3 Boot Manifest structure are shown in the following table:

Name	Offset	Type	Description
version	0	uint32_t	Boot Manifest version
padding	4	uint32_t	Reserved, set to 0
plat_data	8	uint64_t	Pointer to Platform Data section
plat_dram	16	memory_info	NS DRAM Layout Info structure
plat_console	40	console_list	List of consoles available to RMM
plat_ncoh_region	64	memory_info	Device non-coherent ranges Info structure
plat_coh_region	88	memory_info	Device coherent ranges Info structure
plat_smmu	112	smmu_list	List of SMMUs available to RMM (from Boot Manifest v0.5)
plat_root_complex	136	root_complex_list	List of PCIe root complexes available to RMM (from Boot Manifest v0.5)

Memory Info structure

Memory Info structure contains information about platform memory layout. The members of this structure are shown in the table below:

Name	Offset	Type	Description
num_banks	0	uint64_t	Number of memory banks/device regions
banks	8	memory_bank *	Pointer to 'memory_bank'[] array
checksum	16	uint64_t	Checksum

Checksum is calculated as two's complement sum of 'num_banks', 'banks' pointer and memory banks data array pointed by it.

Memory Bank/Device region structure

Memory Bank structure contains information about each memory bank/device region:

Name	Offset	Type	Description
base	0	uint64_t	Base address
size	8	uint64_t	Size of memory bank/device region in bytes

Console List structure

Console List structure contains information about the available consoles for RMM. The members of this structure are shown in the table below:

Name	Offset	Type	Description
num_consoles	0	uint64_t	Number of consoles
consoles	8	console_info *	Pointer to ‘console_info’[] array
checksum	16	uint64_t	Checksum

Checksum is calculated as two’s complement sum of ‘num_consoles’, ‘consoles’ pointer and the consoles array pointed by it.

Console Info structure

Console Info structure contains information about each Console available to RMM.

Name	Offset	Type	Description
base	0	uint64_t	Console Base address
map_pages	8	uint64_t	Num of pages to map for console MMIO
name	16	char[8]	Name of console
clk_in_hz	24	uint64_t	UART clock (in Hz) for console
baud_rate	32	uint64_t	Baud rate
flags	40	uint64_t	Additional flags (RES0)

SMMU List structure

SMMU List structure contains information about SMMUs available for RMM. The members of this structure are shown in the table below:

Name	Offset	Type	Description
num_smmus	0	uint64_t	Number of SMMUs
smmus	8	smmu_info *	Pointer to ‘smmu_info’[] array
checksum	16	uint64_t	Checksum

SMMU Info structure

SMMU Info structure contains information about each SMMU available to RMM.

Name	Offset	Type	Description
smmu_base	0	uint64_t	SMMU Base address
smmu_r_base	8	uint64_t	SMMU Realm Pages base address

Root Complex List structure

Root Complex List structure contains information about PCIe root complexes available for RMM. The members of this structure are shown in the table below.

Name	Offset	Type	Description
num_root_complex	0	uint64_t	Number of root complexes
rc_info_version	8	uint32_t	Root Complex Info structure version
padding	12	uint32_t	Reserved, set to 0
root_complex	16	root_complex_info *	Pointer to 'root_complex'[] array
checksum	24	uint64_t	Checksum

The checksum calculation of Root Complex List structure includes all data structures referenced by 'root_complex_info' pointer.

Root Complex Info structure

Root Complex Info structure contains information about each PCIe root complex available to RMM. The table below describes the members of this structure as per v0.1.

Name	Offset	Type	Description
ecam_base	0	uint64_t	PCIe ECAM Base address
segment	8	uint8_t	PCIe segment identifier
padding[3]	9	uint8_t	Reserved, set to 0
num_root_ports	12	uint32_t	Number of root ports
root_ports	16	root_port_info *	Pointer to 'root_port_info'[] array

The Root Complex Info structure version uses the same numbering scheme as described in *RMM-EL3 Interface versioning*.

Root Port Info structure

Root Complex Info structure contains information about each root port in PCIe root complex.

Name	Offset	Type	Description
root_port_id	0	uint16_t	Root Port identifier
padding	2	uint16_t	Reserved, set to 0
num_bdf_mappings	4	uint32_t	Number of BDF mappings
bdf_mappings	8	bdf_mapping_info *	Pointer to 'bdf_mapping_info'[] array

BDF Mapping Info structure

BDF Mapping Info structure contains information about each Device-Bus-Function (BDF) mapping for PCIe root port.

Name	Offset	Type	Description
mapping_base	0	uint16_t	Base of BDF mapping (inclusive)
mapping_top	2	uint16_t	Top of BDF mapping (exclusive)
mapping_off	4	uint16_t	Mapping offset, as per Arm Base System Architecture: StreamID = RequesterID[N-1:0] + (1<<N)*Constant_B
smmu_idx	6	uint16_t	SMMU index in 'smmu_info'[] array

EL3 Token Sign Request structure

This structure represents a realm attestation token signing request.

Name	Offset	Type	Description
sig_alg_id	0	uint32_t	Algorithm idenfier for the sign request. - 0x0: ECC SECP384R1 (ECDSA) - Other values reserved
rec_granule	8	uint64_t	Identifier used by RMM to associate a signing request to a realm. Must not be interpreted or modified.
req_ticket	16	uint64_t	Value used by RMM to associate request and responses. Must not be interpreted or modified.
hash_alg_id	24	uint32_t	Hash algorithm for data in *hash_buf* - 0x1: SHA2-384 - All other values reserved.
hash_buf	32	uint8_t[]	TBS (to-be-signed) Hash of length defined by hash algorithm *hash_alg_id*

EL3 Token Sign Response structure

This structure represents a realm attestation token signing response.

Name	Offset	Type	Description
rec_granule	0	uint64_t	Identifier used by RMM to associate a signing request to a realm. Must not be interpreted or modified.
req_ticket	8	uint64_t	Value used by RMM to associate request and responses. Must not be interpreted or modified.
sig_len	16	uint16_t	Length of the *signature_buf* field
signature_buf	18	uint8_t[]	Signature

4.22 Granule Protection Tables Library

This document describes the design of the Granule Protection Tables (GPT) library used by Trusted Firmware-A (TF-A). This library provides the APIs needed to initialize the GPTs based on a data structure containing information about the systems memory layout, configure the system registers to enable granule protection checks based on these tables, and transition granules between different PAS (physical address spaces) at runtime.

Arm CCA adds two new security states for a total of four: root, realm, secure, and non-secure. In addition to new security states, corresponding physical address spaces have been added to control memory access for each state. The PAS access allowed to each security state can be seen in the table below.

Table 26: Security states and PAS access rights

	Root state	Realm state	Secure state	Non-secure state
Root PAS	yes	no	no	no
Realm PAS	yes	yes	no	no
Secure PAS	yes	no	yes	no
Non-secure PAS	yes	yes	yes	yes

The GPT can function as either a 1 level or 2 level lookup depending on how a PAS region is configured. The first step is the level 0 table, each entry in the level 0 table controls access to a relatively large region in memory (GPT Block descriptor), and the entire region can belong to a single PAS when a one step mapping is used. Level 0 entry can also link to a level 1 table (GPT Table descriptor) with a 2 step mapping. To change PAS of a region dynamically, the region must be mapped in Level 1 table.

The Level 1 tables entries with the same PAS can be combined to form a contiguous block entry using GPT Contiguous descriptor. More details about this is explained in the following section.

4.22.1 Design Concepts and Interfaces

This section covers some important concepts and data structures used in the GPT library.

There are three main parameters that determine how the tables are organized and function: the PPS (protected physical space) which is the total amount of protected physical address space in the system, PGS (physical granule size) which is how large each level 1 granule is, and L0GPTSZ (level 0 GPT size) which determines how much physical memory is governed by each level 0 entry. A granule is the smallest unit of memory that can be independently assigned to a PAS.

L0GPTSZ is determined by the hardware and is read from the GPCCR_EL3 register. PPS and PGS are passed into the APIs at runtime and can be determined in whatever way is best for a given platform, either through some algorithm or hard coded in the firmware.

GPT setup is split into two parts: table creation and runtime initialization. In the table creation step, a data structure containing information about the desired PAS regions is passed into the library which validates the mappings, creates the tables in memory, and enables granule protection checks. It also allocates memory for fine-grained locks adjacent to the L0 tables. In the runtime initialization step, the runtime firmware locates the existing tables in memory using the GPT register configuration and saves important data to a structure used by the granule transition service which will be covered more below.

In the reference implementation for FVP models, you can find an example of PAS region definitions in the file `plat/arm/board/fvp/include/fvp_pas_def.h`. Table creation API calls can be found in `plat/arm/common/arm_common.c` and runtime initialization API calls can be seen in `plat/arm/common/arm_bl31_setup.c`.

During the table creation time, the GPT lib opportunistically fuses contiguous GPT L1 entries having the same PAS. The maximum size of supported contiguous blocks is defined by `RME_GPT_MAX_BLOCK` build option.

Defining PAS regions

A `pas_region_t` structure is a way to represent a physical address space and its attributes that can be used by the GPT library to initialize the tables.

This structure is composed of the following:

1. The base physical address
2. The region size
3. The desired attributes of this memory region (mapping type, PAS type)

See the `pas_region_t` type in `include/lib/gpt_rme/gpt_rme.h`.

The programmer should provide the API with an array containing `pas_region_t` structures, then the library will check the desired memory access layout for validity and create tables to implement it.

`pas_region_t` is a public type, however it is recommended that the macros `GPT_MAP_REGION_BLOCK` and `GPT_MAP_REGION_GRANULE` be used to populate these structures instead of doing it manually to reduce the risk of future compatibility issues. These macros take the base physical address, region size, and PAS type as arguments to generate the pas_region_t structure. As the names imply, `GPT_MAP_REGION_BLOCK` creates a region using only L0 mapping while `GPT_MAP_REGION_GRANULE` creates a region using L0 and L1 mappings.

Level 0 and Level 1 Tables

The GPT initialization APIs require memory to be passed in for the tables to be constructed. The `gpt_init_l0_tables` API takes a memory address and size for building the level 0 tables.

The `gpt_init_pas_l1_tables` API takes an address and size for building the level 1 tables which are linked from level 0 descriptors. The tables should have PAS type `GPT_GPI_ROOT` and a typical system might place its level 0 table in SRAM and its level 1 table(s) in DRAM.

Granule Transition Service

The Granule Transition Service allows memory mapped with `GPT_MAP_REGION_GRANULE` ownership to be changed using SMC calls. Non-secure granules can be transitioned to either realm or secure space, and realm and secure granules can be transitioned back to non-secure. This library only allows Level 1 entries to be transitioned. The lib may either shatter contiguous blocks or fuse adjacent GPT entries to form a contiguous block opportunistically. Depending on the maximum block size, the fuse operation may propogate to higher block sizes as allowed by RME Architecture. Thus a higher maximum block size may have a higher runtime cost due to software operations that need to be performed for fuse to bigger block sizes. This cost may be offset by better TLB performance due to the higher block size and platforms need to make the trade-off decision based on their particular workload.

Locking Scheme

During Granule Transition access to L1 tables is controlled by a lock to ensure that no more than one CPU is allowed to make changes at any given time. The granularity of the lock is defined by `RME_GPT_BITLOCK_BLOCK` build option which defines the size of the memory block protected by one bit of `bitlock_t` structure. Setting this option to 0 chooses a single spinlock for all GPT L1 table entries.

4.22.2 Library APIs

The public APIs and types can be found in `include/lib/gpt_rme/gpt_rme.h` and this section is intended to provide additional details and clarifications.

To create the GPTs and enable granule protection checks the APIs need to be called in the correct order and at the correct time during the system boot process.

1. Firmware must enable the MMU.
2. Firmware must call `gpt_init_l0_tables` to initialize the level 0 tables to a default state, that is, initializing all of the L0 descriptors to allow all accesses to all memory. The PPS is provided to this function as an argument.
3. DDR discovery and initialization by the system, the discovered DDR region(s) are then added to the L1 PAS regions to be initialized in the next step and used by the GTSI at runtime.
4. Firmware must call `gpt_init_pas_l1_tables` with a pointer to an array of `pas_region_t` structures containing the desired memory access layout. The PGS is provided to this function as an argument.
5. Firmware must call `gpt_enable` to enable granule protection checks by setting the correct register values.
6. In systems that make use of the granule transition service, runtime firmware must call `gpt_runtime_init` to set up the data structures needed by the GTSI to find the tables and transition granules between PAS types. The base address of bitlocks array and its size are provided to this function as arguments. These parameters are not used in case of a single spinlock for all GPT L1 table entries(`RME_GPT_BITLOCK_BLOCK` is 0) and are passed as zero values.

API Constraints

The values allowed by the API for PPS and PGS are enumerated types defined in the file `include/lib/gpt_rme/gpt_rme.h`.

Allowable values for PPS along with their corresponding size.

- `GPCCR_PPS_4GB` (4GB protected space, 0x100000000 bytes)
- `GPCCR_PPS_64GB` (64GB protected space, 0x1000000000 bytes)
- `GPCCR_PPS_1TB` (1TB protected space, 0x10000000000 bytes)
- `GPCCR_PPS_4TB` (4TB protected space, 0x40000000000 bytes)
- `GPCCR_PPS_16TB` (16TB protected space, 0x100000000000 bytes)
- `GPCCR_PPS_256TB` (256TB protected space, 0x1000000000000 bytes)
- `GPCCR_PPS_4PB` (4PB protected space, 0x10000000000000 bytes)

Allowable values for PGS along with their corresponding size.

- `GPCCR_PGS_4K` (4KB granules, 0x1000 bytes)
- `GPCCR_PGS_16K` (16KB granules, 0x4000 bytes)
- `GPCCR_PGS_64K` (64KB granules, 0x10000 bytes)

Allowable values for L0GPTSZ along with the corresponding size.

- `GPCCR_L0GPTSZ_30BITS` (1GB regions, 0x40000000 bytes)
- `GPCCR_L0GPTSZ_34BITS` (16GB regions, 0x400000000 bytes)
- `GPCCR_L0GPTSZ_36BITS` (64GB regions, 0x1000000000 bytes)
- `GPCCR_L0GPTSZ_39BITS` (512GB regions, 0x8000000000 bytes)

Note that the value of the PPS, PGS, and L0GPTSZ definitions is an encoded value corresponding to the size, not the size itself. The decoded hex representations of the sizes have been provided for convenience.

The L0 table memory has some constraints that must be taken into account.

- The L0 table must be aligned to either the table size or 4096 bytes, whichever is greater. L0 table size is the total protected space (PPS) divided by the size of each L0 region (L0GPTSZ) multiplied by the size of each L0 descriptor (8 bytes). ((PPS / L0GPTSZ) * 8)
- The L0 memory must fall within a PAS of type GPT_GPI_ROOT.

The L1 memory also has some constraints.

- The L1 tables must be aligned to their size. The size of each L1 table is the size of each L0 region (L0GPTSZ) divided by the granule size (PGS) divided by the granules controlled in each byte (2). ((L0GPTSZ / PGS) / 2)
- There must be enough L1 memory supplied to build all requested L1 tables.
- The L1 memory must fall within a PAS of type GPT_GPI_ROOT.

- The platform allocates the bitlock array which contains fine-grained `bitlock_t` data structures. The RME GPT library will check that the array has at least the amount of memory defined by PPS and `RME_GPT_BITLOCK_BLOCK` value.

If an invalid combination of parameters is supplied, the APIs will print an error message and return a negative value. The return values of APIs should be checked to ensure successful configuration.

Sample Calculation for L0 memory size and alignment

Let PPS=4GB and L0GPTSZ=GPCCR_L0GPTSZ_30BITS

We can find the total L0 table size with ((PPS / L0GPTSZ) * 8)

Substitute values to get this: ((0x100000000 / 0x40000000) * 8)

And solve to get 32 bytes. In this case, 4096 is greater than 32, so the L0 tables must be aligned to 4096 bytes.

Sample calculation for bitlocks array size

Let PPS=256TB and RME_GPT_BITLOCK_BLOCK=1

The size of bitlocks array in bits is the total protected space (PPS) divided by the size of memory block per bit. The size of memory block is `RME_GPT_BITLOCK_BLOCK` (number of 512MB blocks per bit) times 512MB (0x20000000). This is then divided by the number of bits in `bitlock_t` structure (8) to get the size of array in bytes.

In other words, we can find the total size of `bitlock_t` array in bytes with PPS / (RME_GPT_BITLOCK_BLOCK * 0x20000000 * 8).

Substitute values to get this: 0x1000000000000 / (1 * 0x20000000 * 8)

And solve to get 0x10000 bytes.

Sample calculation for L1 table size and alignment

Let PGS=GPCCR_PGS_4K and L0GPTSZ=GPCCR_L0GPTSZ_30BITS

We can find the size of each L1 table with ((L0GPTSZ / PGS) / 2).

Substitute values: ((0x40000000 / 0x1000) / 2)

And solve to get 0x20000 bytes per L1 table.

4.23 Vendor Specific EL3 Monitor Service Calls

This document enumerates and describes the Vendor Specific EL3 Monitor Service Calls.

These are Service Calls defined by the vendor of the EL3 Monitor. They are accessed via `SMC` ("SMC calls") instruction executed from Exception Levels below EL3. SMC calls for Vendor Specific EL3 Monitor Services:

- Follow SMC Calling Convention;
- Use SMC function IDs that fall in the vendor-specific EL3 range, which are

SMC Function Identifier	Service Type
0x87000000 - 0x8700FFFF	SMC32: Vendor Specific EL3 Monitor Service Calls
0xC7000000 - 0xC700FFFF	SMC64: Vendor Specific EL3 Monitor Service Calls

Vendor-specific EL3 monitor services are as follows:

SMC Function Identifier	Service Type	FID's Usage
0x87000010 - 0x8700001F (SMC32)	DebugFS Interface	0 - 11 are in use. 12 - 15 are reserved for future expansion.
0xC7000010 - 0xC700001F (SMC64)		
0x87000020 - 0x8700002F (SMC32)	Performance Measurement Framework (PMF)	0,1 is in use. 2 - 15 are reserved for future expansion.
0xC7000020 - 0xC700002F (SMC64)		
0x87000030 - 0x8700003F (SMC32)	ACS (Architecture Compliance Suite) SMC handler	0 in use. 1 - 15 are reserved for future expansion.
0xC7000030 - 0xC700003F (SMC64)		
0x87000040 - 0x8700004F (SMC32)	TPM Start method	0 is in use. 1 - 15 are reserved for future expansion.
0xC7000040 - 0xC700004F (SMC64)		
0x87000050 - 0x8700FFFF (SMC32)	Reserved	reserved for future expansion
0xC7000050 - 0xC700FFFF (SMC64)		

Source definitions for vendor-specific EL3 Monitor Service Calls used by TF-A are located in the `ven_el3_svc.h` header file.

VEN_EL3_SVC_VERSION_MAJOR	VEN_EL3_SVC_VERSION_MINOR	Changes
1	0	Added Debugfs and PMF services.
1 | 1 | Added ACS SMC handler services.		
1 | 2 | Added TPM Start method.		

Table 1: Showing different versions of Vendor-specific service and changes done with each version

Each sub service will have its own version, one FID allocated for sub service version.

Some ground rules when one should update top level version.

- VEN_EL3_SVC_VERSION_MAJOR is incremented when any of the sub service version discovery FID changes or the FID that was allocated for discovery changes. So any breaking subfeature discovery changes will lead to major version update.
- VEN_EL3_SVC_VERSION_MINOR is incremented when we add a new FID or a new sub service. For example adding an new monitor service at 0x30, Debugfs starts at 0x10 and PMF starts at 0x20 next one will start at 0x30, this will need a update to minor version.

4.23.1 Performance Measurement Framework (PMF)

The *Performance Measurement Framework* allows callers to retrieve timestamps captured at various paths in TF-A execution.

4.23.2 DebugFS interface

The optional DebugFS interface is accessed through Vendor specific EL3 service. Refer to *DebugFS interface* documentation for further details and usage.

4.23.3 Architecture Compliance Suite (ACS) SMC handler

The Architecture Compliance Suite (ACS) SMC handler allows callers to branch to their ACS EL3 code based on their respective use-cases. For more details on System ACS, System ACS.

4.23.4 TPM Start method

TPM start method as mentioned in TCG ACPI specification section 3.3.1.

4.24 DebugFS interface

The optional DebugFS interface is accessed through a Vendor specific EL3 service. Refer to the component documentation for details.

String parameters are passed through a shared buffer using a specific union:

```
union debugfs_parms {
    struct {
        char fname[MAX_PATH_LEN];
    } open;

    struct mount {
        char srv[MAX_PATH_LEN];
        char where[MAX_PATH_LEN];
        char spec[MAX_PATH_LEN];
    } mount;

    struct {
        char path[MAX_PATH_LEN];
        dir_t dir;
    } stat;

    struct {
        char oldpath[MAX_PATH_LEN];
        char newpath[MAX_PATH_LEN];
    } bind;
};
```

Format of the dir_t structure as such:

```
typedef struct {
    char            name[NAMELEN];
    long            length;
    unsigned char   mode;
    unsigned char   index;
    unsigned char   dev;
    qid_t           qid;
} dir_t;
```

- Identifiers

SMC_OK	0
SMC_UNK	-1
DEBUGFS_E_INVALID_PARAMS	-2

MOUNT	0
CREATE	1
OPEN	2
CLOSE	3
READ	4
WRITE	5
SEEK	6
BIND	7
STAT	8
INIT	10
VERSION	11

4.24.1 MOUNT

Description

This operation mounts a blob of data pointed to by path stored in *src*, at filesystem location pointed to by path stored in *where*, using driver pointed to by path in *spec*.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	`MOUNT`

Return values

int32_t	w0 == SMC_OK on success w0 == DEBUGFS_E_INVALID_PARAMS if mount operation failed

4.24.2 OPEN

Description

This operation opens the file path pointed to by *fname*.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	OPEN
uint32_t	mode

mode can be one of:

```
enum mode {
    O_READ   = 1 << 0,
    O_WRITE  = 1 << 1,
    O_RDWR   = 1 << 2,
    O_BIND   = 1 << 3,
    O_DIR    = 1 << 4,
    O_STAT   = 1 << 5
};
```

Return values

int32_t	w0 == SMC_OK on success w0 == DEBUGFS_E_INVALID_PARAMS if open operation failed
uint32_t	w1: file descriptor id on success.

4.24.3 CLOSE

Description

This operation closes a file described by a file descriptor obtained by a previous call to OPEN.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	CLOSE
uint32_t	File descriptor id returned by OPEN

Return values

int32_t	w0 == SMC_OK on success w0 == DEBUGFS_E_INVALID_PARAMS if close operation failed

4.24.4 READ

Description

This operation reads a number of bytes from a file descriptor obtained by a previous call to OPEN.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	`READ`
uint32_t	File descriptor id returned by OPEN
uint32_t	Number of bytes to read

Return values

On success, the read data is retrieved from the shared buffer after the operation.

int32_t	w0 == SMC_OK on success w0 == DEBUGFS_E_INVALID_PARAMS if read operation failed
uint32_t	w1: number of bytes read on success.

4.24.5 SEEK

Description

Move file pointer for file described by given *file descriptor* of given *offset* related to *whence*.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	`SEEK`
uint32_t	File descriptor id returned by OPEN
sint32_t	offset in the file relative to whence
uint32_t	whence

whence can be one of:

KSEEK_SET	0
KSEEK_CUR	1
KSEEK_END	2

Return values

int32_t	w0 == SMC_OK on success w0 == DEBUGFS_E_INVALID_PARAMS if seek operation failed

4.24.6 BIND

Description

Create a link from *oldpath* to *newpath*.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	`BIND`

Return values

int32_t	w0 == SMC_OK on success w0 == DEBUGFS_E_INVALID_PARAMS if bind operation failed

4.24.7 STAT

Description

Perform a stat operation on provided file *name* and returns the directory entry statistics into *dir*.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	`STAT`

Return values

int32_t	w0 == SMC_OK on success w0 == DEBUGFS_E_INVALID_PARAMS if stat operation failed

4.24.8 INIT

Description

Initial call to setup the shared exchange buffer. Notice if successful once, subsequent calls fail after a first initialization. The caller maps the same page frame in its virtual space and uses this buffer to exchange string parameters with filesystem primitives.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	`INIT`
uint64_t	Physical address of the shared buffer.

Return values

int32_t	w0 == SMC_OK on success w0 == DEBUGFS_E_INVALID_PARAMS if already initialized, or internal error occurred.

4.24.9 VERSION

Description

Returns the debugfs interface version if implemented in TF-A.

Parameters

uint32_t	FunctionID (0x87000010 / 0xC7000010)
uint32_t	VERSION

Return values

int32_t	w0 == SMC_OK on success w0 == SMC_UNK if interface is not implemented
uint32_t	w1: On success, debugfs interface version, 32 bits value with major version number in upper 16 bits and minor version in lower 16 bits.

- CREATE(1) and WRITE (5) command identifiers are unimplemented and return *SMC_UNK*.

4.25 Context Management Library

This document provides an overview of the Context Management library implementation in Trusted Firmware-A (TF-A). It enumerates and describes the APIs implemented and their accessibility from other components at EL3.

4.25.1 Overview

Arm TrustZone architecture facilitates hardware-enforced isolation between software running in various security states (Secure/Non-Secure/Realm). The general-purpose registers, most of the system registers and vector registers are not banked per world. When moving between the security states it is the responsibility of the secure monitor software (BL31(AArch64) / BL32(Aarch32)) in TF-A, not the hardware, to save and restore register state. Refer to Trustzone for AArch64 for more details.

EL3 Runtime Firmware, also termed as secure monitor firmware, is integrated with a context management library to handle the context of the CPU, managing the saving and restoring of register states across the worlds.

4.25.2 TF-A Context

In TF-A, the context is represented as a data structure used by the EL3 firmware to preserve the state of the CPU at the next lower exception level (EL) in a given security state and save enough EL3 metadata to be able to return to that exception level and security state. The memory for the context data structures are allocated in BSS section of EL3 firmware.

In a trusted system at any instance, a given CPU could be executing in one of the security states (Non-Secure, Secure, Realm). Each world must have its configuration of system registers independent of other security states to access and execute any of the architectural features.

If the CPU switches across security states (for example: from Non-secure to Secure or vice versa), the register contents, especially the ones that are not banked (EL2/EL1, vector, general-purpose registers), will be overwritten, as the software running in either state has the privileges to access them. Additionally, some of the architectural features enabled in the former security state will be unconditionally accessible in the latter security state as well. This can be a major concern when dealing with security-specific bits, as they need to be explicitly enabled or disabled in each state to prevent data leakage across the worlds.

In general, an ideal trusted system should have Secure world-specific configurations that are not influenced by Normal World operations. Therefore, for each CPU, we need to maintain world-specific context to ensure that register entries from one world do not leak or impact the execution of the CPU in other worlds. This will help ensure the integrity and security of the system, preventing any unauthorized access or data corruption between the different security states.

4.25.3 Design

The Context Management library in TF-A is designed to cover all the requirements for maintaining world-specific context essential for a trusted system. This includes implementing CPU context initialization and management routines, as well as other helper APIs that are required by dispatcher components in EL3 firmware, which are collectively referred to as CPU Context Management. The APIs and their usecases are listed in detail under the *Library APIs* section.

Originally, the Context Management library in TF-A was designed to cater for a two-world system, comprising of Non-Secure and Secure Worlds. In this case, the EL3 Firmware is assumed to be running in Secure World. With introduction of Realm Management Extension (RME), from Armv9.2 a system can have four distinct worlds (Non-Secure, Secure, Realm, Root). RME isolates EL3 from all other Security states and moves it into its own security state called root. EL3 firmware now runs at Root World and thereby is trusted from software in Non-secure, Secure, and Realm states. Refer to Security States with RME for more details.

Key principles followed in designing the context management library :

1. **EL3 should only initialize immediate used lower EL**

Context Management library running at EL3 should only initialize and monitor the immediate used lower EL. This implies that, when S-EL2 is present in the system, EL3 should initialise and monitor S-EL2 registers only. S-EL1 registers should not be the concern of EL3 while S-EL2 is in place. In systems where S-EL2 is absent, S-EL1 registers should be initialised from EL3.

2. **Decentralized model for context management**

Each world (Non-Secure, Secure, and Realm) should have their separate component in EL3 responsible for their respective world context management. Both the Secure and Realm world have associated dispatcher components in EL3 firmware to allow management of the respective worlds. For the Non-Secure world, PSCI Library (BL31)/context management library provides routines to help initialize the Non-Secure world context.

3. **Flexibility for Dispatchers to select desired feature set to save and restore**

Each feature is supported with a helper function `is_feature_supported(void)`, to detect its presence at runtime. This helps dispatchers to select the desired feature set, and thereby save and restore the configuration associated with them.

4. **Dynamic discovery of Feature enablement by EL3**

TF-A supports four states for feature enablement at EL3, to make them available for lower exception levels.

```
#define FEAT_STATE_DISABLED             0
#define FEAT_STATE_ENABLED              1
#define FEAT_STATE_CHECK                2
```

A pattern is established for feature enablement behavior. Each feature must support the 3 possible values with rigid semantics.

- **FEAT_STATE_DISABLED** - all code relating to this feature is always skipped. Firmware is unaware of this feature.
- **FEAT_STATE_ALWAYS** - all code relating to this feature is always executed. Firmware expects this feature to be present in hardware.
- **FEAT_STATE_CHECK** - same as `FEAT_STATE_ALWAYS` except that the feature's existence will be checked at runtime. Default on dynamic platforms (example: FVP).

Note: In general, it is assumed that all cores will support the same set of architectural features (features will be symmetrical). However, there are cases where this is impractical to achieve. Only some features can be mismatched among cores and this is the exception rather than the rule. This is due to the fact that Operating systems are designed for SMP systems. There are no clear guidelines what kind of mismatch is allowed but following pointers can help in making a decision:

- All mandatory features must be symmetric.
- Any feature that impacts the generation of page tables must be symmetric.
- Any feature access which does not trap to EL3 should be symmetric.
- Features related with profiling, debug and trace could be asymmetric
- Migration of vCPU/tasks between CPUs should not cause an error

TF-A caters for mismatched features, however, this is not regularly tested for all features and may not work as expected, even without considering OS support.

Note: `FEAT_RAS` is an exception here, as it impacts the execution of EL3 and it is essential to know its presence at compile time. Refer to `ENABLE_FEAT` macro under *Build Options* section for more details.

4.25.4 Code Structure

lib/el3_runtime/(aarch32/aarch64) - Context library code directory.

Source Files

1. `context_mgmt.c` : consists of core functions that setup, save and restore context for different security states alongside high level feature enablement APIs for individual worlds.
2. `cpu_data_array.c` : contains per_cpu_data structure instantiation.
3. `context.S` : consists of functions that save and restore some of the context structure members in assembly code.
4. `cpu_data.S` : consists of helper functions to initialise per_cpu_data pointers.
5. `el3_common_macros.S` : consists of macros to facilitate actions to be performed during cold and warmboot and el3 registers initialisation in assembly code.

Header Files

1. `context_mgmt.h` : contains the public interface to Context Management Library.
2. `context.h` : contains the helper macros and definitions for context entries.
3. `cpu_data.h` : contains the public interface to Per CPU data structure.
4. `context_debug.h` : contains public interface to report context memory utilisation across the security states.
5. `context_el2.h` : internal header consisting of helper macros to access EL2 context entries. Used by `context.h`.

Apart from these files, we have some context related source files under `BL1` and `BL31` directory. `bl1_context_mgmt.c` `bl31_context_mgmt.c`

4.25.5 Bootloader Images utilizing Context Management Library

Bootloader	Context Management Library
BL1	Yes
BL2	No
BL31 (Aarch64- EL3runtime firmware)	Yes
BL32 (Aarch32- EL3runtime firmware)	Yes

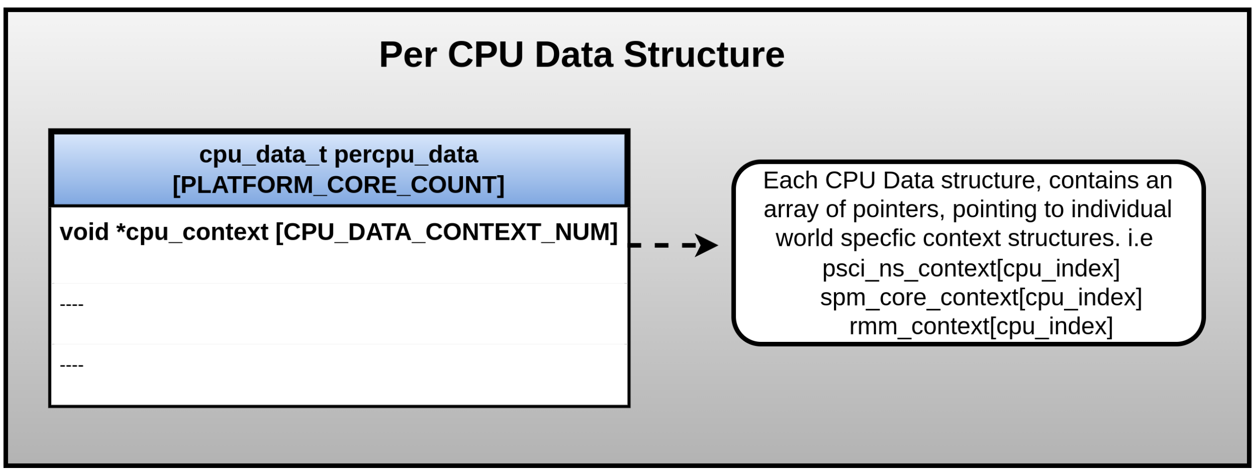
4.25.6 CPU Data Structure

For a given system, depending on the CPU count, the platform statically allocates memory for the CPU data structure.

```
/* The per_cpu_ptr_cache_t space allocation */
cpu_data_t percpu_data[PLATFORM_CORE_COUNT];
```

This CPU data structure has a member element with an array of pointers to hold the Non-Secure, Realm and Secure security state context structures as listed below.

```
typedef struct cpu_data {
#ifdef __aarch64__
void *cpu_context[CPU_DATA_CONTEXT_NUM];
#endif

....
....

#if (ENABLE_FEAT_IDTE3 && defined(__aarch64__))
percpu_idregs_t idregs[CPU_CONTEXT_NUM];
#endif
}cpu_data_t;
```



At runtime, `cpu_context[CPU_DATA_CONTEXT_NUM]` array will be intitialised with the Secure, Non-Secure and Realm context structure addresses to ensure proper handling of the register state. See *Library APIs* section for more details.

When FEAT_IDTE3 is enabled, the ID registers ID_AA64DFR0_EL1 and ID_AA64DFR1_EL1 are cached in the percpu_idregs_t idregs[CPU_CONTEXT_NUM] array within the CPU data structure. These cached copies are used to service trapped reads of the corresponding registers from lower exception levels. Because debug and trace features can vary across CPUs, these ID registers are cached per-CPU and per-world to accurately represent asymmetric configurations.

The per-cpu cached ID registers are initialized in `psci_arch_setup()` via `cm_init_percpu_once_regs()`.

4.25.7 CPU Context and Memory allocation

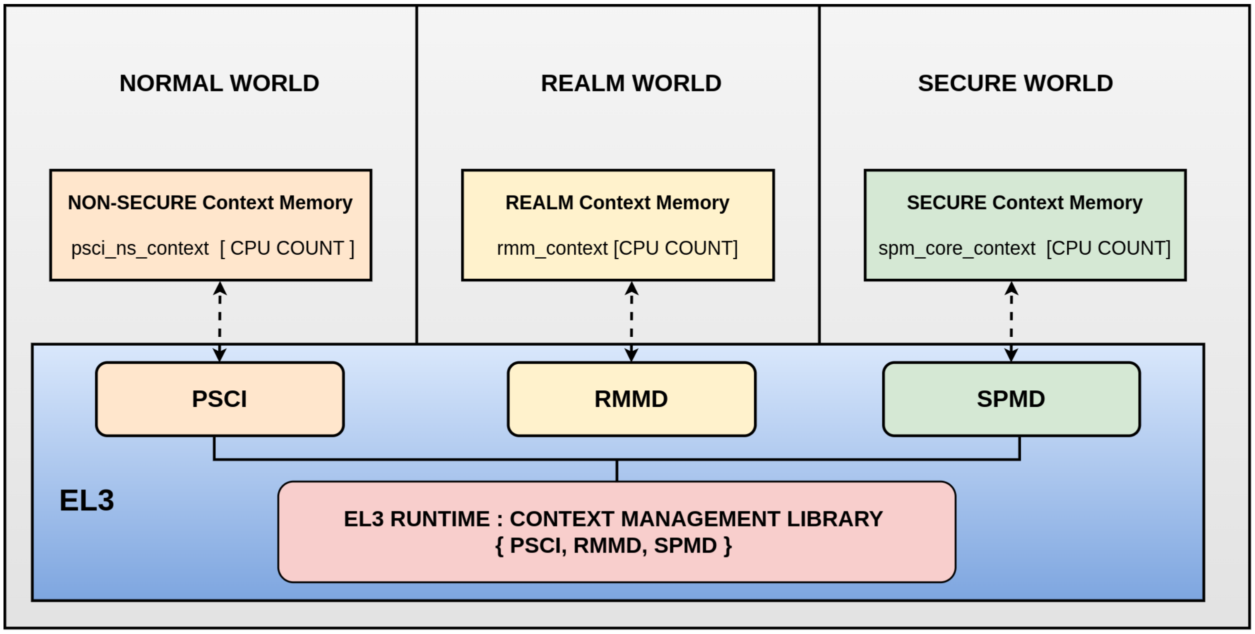
CPU Context

The members of the context structure used by the EL3 firmware to preserve the state of CPU across exception levels for a given security state are listed below.

```
typedef struct cpu_context {
gp_regs_t gpregs_ctx;
el3_state_t el3state_ctx;

cve_2018_3639_t cve_2018_3639_ctx;

#if ERRATA_SPECULATIVE_AT
errata_speculative_at_t errata_speculative_at_ctx;
#endif

#if CTX_INCLUDE_PAUTH_REGS
pauth_t pauth_ctx;
#endif

#if (CTX_INCLUDE_EL2_REGS && IMAGE_BL31)
el2_sysregs_t el2_sysregs_ctx;
#else
el1_sysregs_t el1_sysregs_ctx;
#endif
} cpu_context_t;
```

Context Memory Allocation

CPUs maintain their context per world. The individual context memory allocation for each CPU per world is allocated by the world-specific dispatcher components at compile time as shown below.



NS-Context Memory

It's important to note that the Normal world doesn't possess the dispatcher component found in the Secure and Realm worlds. Instead, the PSCI library at EL3 handles memory allocation for `Non-Secure` world context for all CPUs.

```
static PER_CPU_DEFINE(cpu_context_t, psci_ns_context);
```

Secure-Context Memory

Secure World dispatcher (such as SPMD) at EL3 allocates the memory for `Secure` world context of all CPUs.

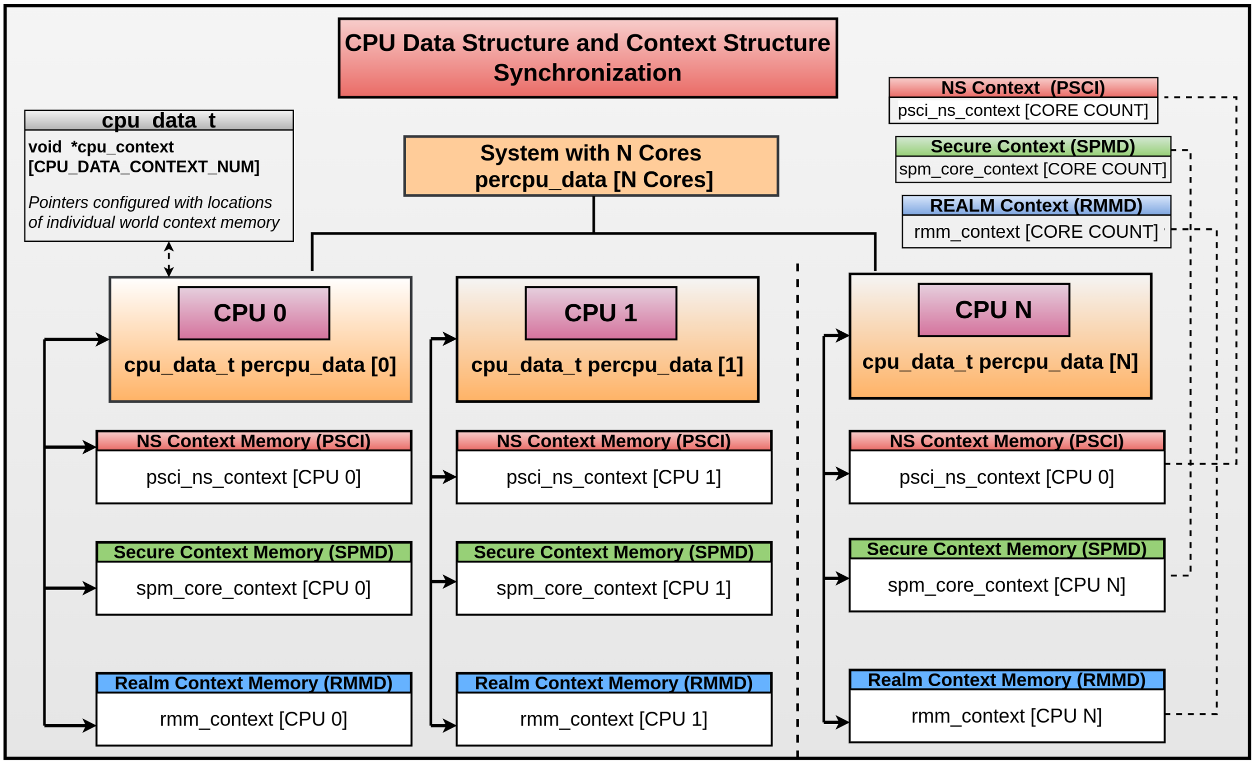
```
static PER_CPU_DEFINE(spmd_spm_core_context_t, spm_core_context);
```

Realm-Context Memory

Realm World dispatcher (RMMD) at EL3 allocates the memory for `Realm` world context of all CPUs.

```
PER_CPU_DEFINE(rmmd_rmm_context_t, rmm_context);
```

To summarize, the world-specific context structures are synchronized with per-CPU data structures, which means that each CPU will have an array of pointers to individual worlds. The figure below illustrates the same.



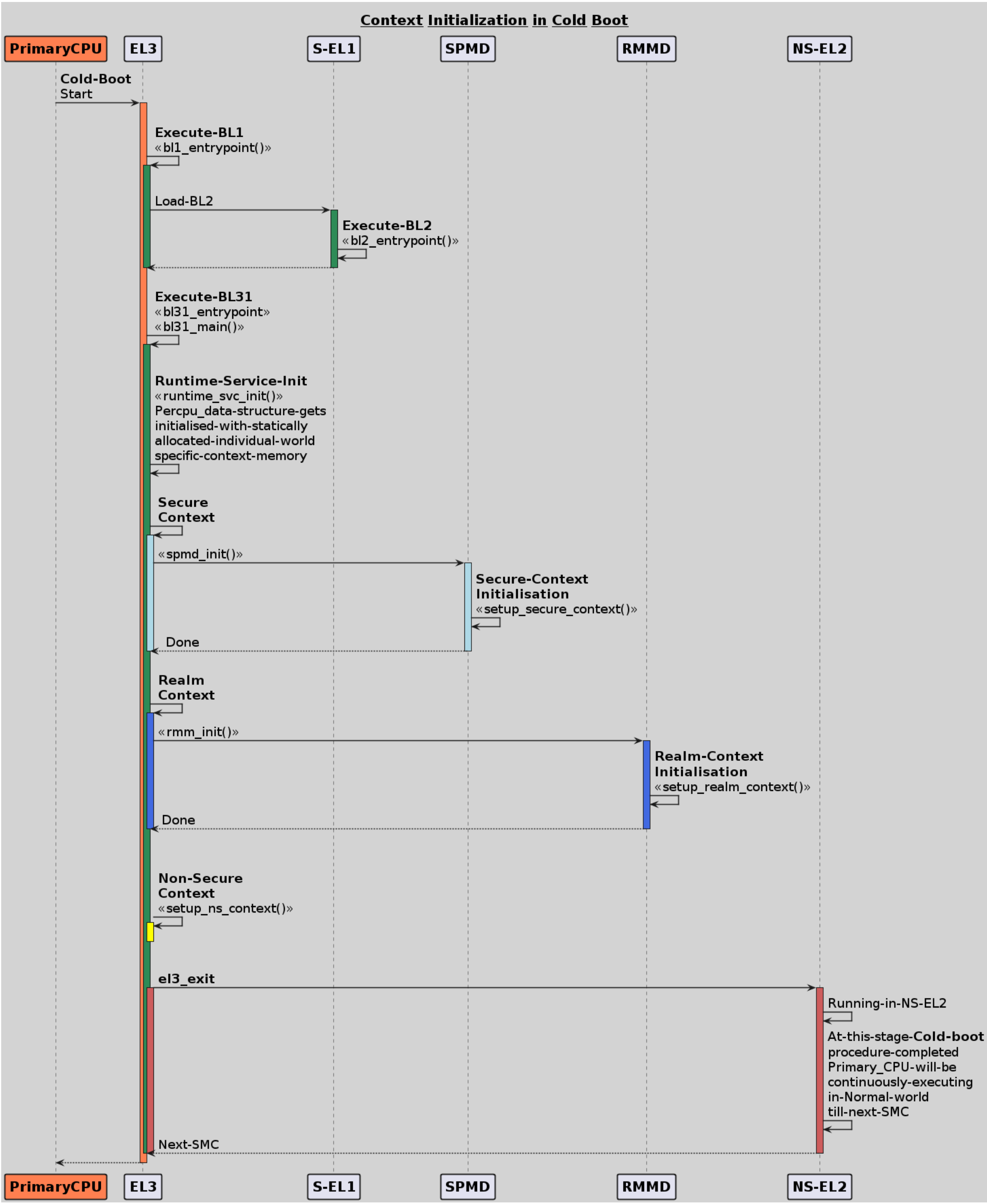
4.25.8 Context Setup/Initialization

The CPU has been assigned context structures for every security state, which include Non-Secure, Secure and Realm. It is crucial to initialize each of these structures during the bootup of every CPU before they enter any security state for the first time. This section explains the specifics of how the initialization of every CPU context takes place during both cold and warm boot paths.

Context Setup during Cold boot

The cold boot path is mainly executed by the primary CPU, other than essential CPU initialization executed by all CPUs. After executing BL1 and BL2, the Primary CPU jumps to the BL31 image for runtime services initialization. During this process, the per_cpu_data structure gets initialized with statically allocated world-specific context memory.

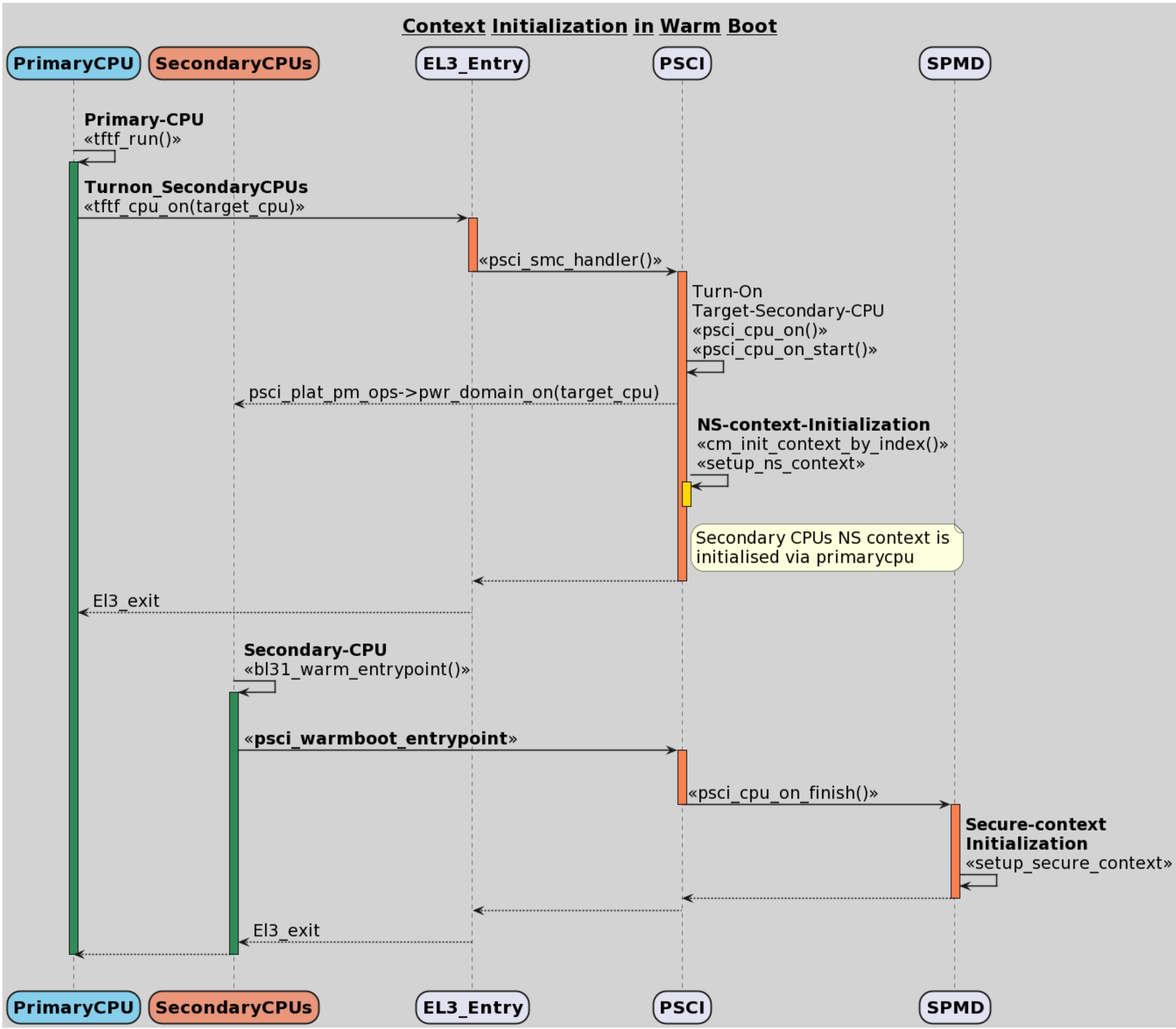
Later in the cold boot sequence, the BL31 image at EL3 checks for the presence of a Secure world image at S-EL2. If detected, it invokes the secure context initialization sequence under SPMD. Additionally, based on RME enablement, the Realm context gets initialized from the RMMD at EL3. Finally, before exiting to the normal world, the Non-Secure context gets initialized via the context management library. At this stage, all Primary CPU contexts are initialized and the CPU exits EL3 to enter the Normal world.



Note: The figure above illustrates a scenario on FVP for one of the build configurations with TFTF component at NS-EL2.

Context Setup during Warmboot

During a warm boot sequence, the primary CPU is responsible for powering on the secondary CPUs. Refer to *CPU Reset* and *Firmware Design* sections for more details on the warm boot.



The primary CPU writes the entrypoint for the secondary CPU. When the secondary wakes up it initialises its own context via `cm_init_my_context( ep )` using the provided entrypoint.

`psci_warmboot_entrypoint()` is the warm boot entrypoint procedure. During the warm bootup process, secondary CPUs have their secure context initialized through SPMD at EL3. Upon successful SP initialization, the SPD power management operations become shared with the PSCI library. During this process, the SPMD duly registers its handlers with the PSCI library.

```
file: psci_common.c
const spd_pm_ops_t *psci_spd_pm;

file: spmd_pm.c
const spd_pm_ops_t spmd_pm = {
```

(continues on next page)

(continued from previous page)

```
.svc_on_finish = spmd_cpu_on_finish_handler,
.svc_off = spmd_cpu_off_handler
}
```

Secondary CPUs during their bootup in the `psci_cpu_on_finish()` routine get their secure context initialised via the registered SPMD handler `spmd_cpu_on_finish_handler()` at EL3. The figure above illustrates the same with reference of Primary CPU running at NS-EL2.

4.25.9 Library APIs

The public APIs and types can be found in `include/lib/el3_runtime/context_management.h` and this section is intended to provide additional details and clarifications.

Context Initialization for Individual Worlds

The library implements high level APIs for the CPUs in setting up their individual context for each world (Non-Secure, Secure and Realm).

static void **setup_context_common** (cpu_context_t *ctx, const entry_point_info_t *ep) ;

This function is responsible for the general context initialization that applies to all worlds. It will be invoked first, before calling the individual world-specific context setup APIs.

static void **setup_ns_context** (cpu_context_t *ctx, const struct entry_point_info *ep) ;

static void **setup_realm_context** (cpu_context_t *ctx, const struct entry_point_info *ep) ;

static void **setup_secure_context** (cpu_context_t *ctx, const struct entry_point_info *ep) ;

Depending on the security state that the CPU needs to enter, the respective world-specific context setup handlers listed above will be invoked once per-CPU to set up the context for their execution.

void **cm_manage_extensions_el3** (void)

This function initializes all EL3 registers whose values do not change during the lifetime of EL3 runtime firmware. It is invoked from each CPU via the cold boot path `bl31_main()` and in the WarmBoot entry path `void psci_warmboot_entrypoint()`.

Runtime Save and Restore of Registers

EL1 Registers

void **cm_el1_sysregs_context_save** (uint32_t security_state) ;

void **cm_el1_sysregs_context_restore** (uint32_t security_state) ;

These functions are utilized by the world-specific dispatcher components running at EL3 to facilitate the saving and restoration of the EL1 system registers during a world switch.

EL2 Registers

void **cm_el2_sysregs_context_save** (uint32_t security_state) ;

void **cm_el2_sysregs_context_restore** (uint32_t security_state) ;

These functions are utilized by the world-specific dispatcher components running at EL3 to facilitate the saving and restoration of the EL2 system registers during a world switch.

void **cm_el2_sysreg_context_save_gic** (uint32_t security_state) ;

void **cm_el2_sysreg_context_restore_gic** (uint32_t security_state) ;

These functions are utilized by world dispatcher components at EL3 to save and restore GIC-related EL2 system register context during a world switch. They are required when worlds do not share GIC context.

Pauth Registers

Pointer Authentication feature is enabled by default for Non-Secure world and disabled for Secure and Realm worlds. In this case, we don't need to explicitly save and restore the Pauth registers during world switch. However, `CTX_INCLUDE_PAUTH_REGS` flag is explicitly used to enable Pauth for lower exception levels of Secure and Realm worlds. In this scenario, we save the general purpose and Pauth registers while we enter EL3 from lower ELs via `prepare_el3_entry` and restore them back while we exit EL3 to lower ELs via `el3_exit`.

```
.macro save_el3_runtime_regs
func restore_el3_runtime_regs
```

Feature Enablement for Individual Worlds

static void **manage_extensions_nonsecure** (cpu_context_t *ctx) ;

static void **manage_extensions_secure** (cpu_context_t *ctx) ;

static void **manage_extensions_realm** (cpu_context_t *ctx)

Functions that allow the enabling and disabling of architectural features for each security state. These functions are invoked from the top-level setup APIs during context initialization.

Further, a pattern is established for feature enablement code (AArch64). Each feature implements following APIs as applicable: Note: (`xxx` is the name of the feature in the APIs)

- `is_feat_xxx_supported()` and `is_feat_xxx_present()` - mandatory for all features.
- `xxx_enable(cpu_context * )` and `xxx_disable(cpu_context * )` - optional functions to enable the feature for the passed context only. To be called in the respective world's setup_context to select behaviour.

- `xxx_init_el3()` - optional function to enable the feature in-place in any EL3 registers that are never context switched. The values they write must never change, otherwise the functions mentioned in previous point should be used. Invoked from `cm_manage_extensions_el3()`.
- `xxx_init_el2_unused()` - optional function to enable the feature in-place in any EL2 registers that are necessary for execution in EL1 with no EL2 present.

The above mentioned rules, followed for `FEAT_SME` is shown below:

```
void sme_enable(cpu_context_t *context);
void sme_init_el3(void);
void sme_init_el2_unused(void);
void sme_disable(cpu_context_t *context);
```

4.25.10 Per-world Context

Apart from the CPU context structure, we have another structure to manage some of the EL3 system registers whose values are identical across all the CPUs referred to as `per_world_context_t`. The Per-world context structure is intended for managing EL3 system registers with identical values across all CPUs, requiring only a singular context entry for each individual world. This structure operates independently of the CPU context structure and is intended to manage specific EL3 registers.

```
typedef struct per_world_context {
        uint64_t ctx_cptr_el3;
        uint64_t ctx_mpam3_el3;
#if (ENABLE_FEAT_IDTE3 && IMAGE_BL31)
        perworld_idregs_t idregs;
#endif
} per_world_context_t;
```

When FEAT_IDTE3 (Trapping ID register accesses to EL3) is enabled, the `per_world_context_t` structure includes a `perworld_idregs_t` member that caches architectural ID registers common across all CPUs in a world. This cache allows EL3 firmware to provide consistent, virtualized ID register values to lower exception levels when traps occur. The cached values stored in `idregs` are returned in place of the actual hardware register values. During initialization, the per-world ID register cache is populated by `idte3_init_cached_idregs_per_world()`.

These functions facilitate the activation of architectural extensions that possess identical values across all cores for the individual Non-secure, Secure, and Realm worlds.

4.25.11 Root-Context (EL3-Execution-Context)

EL3/Root Context is the execution environment while the CPU is running at EL3.

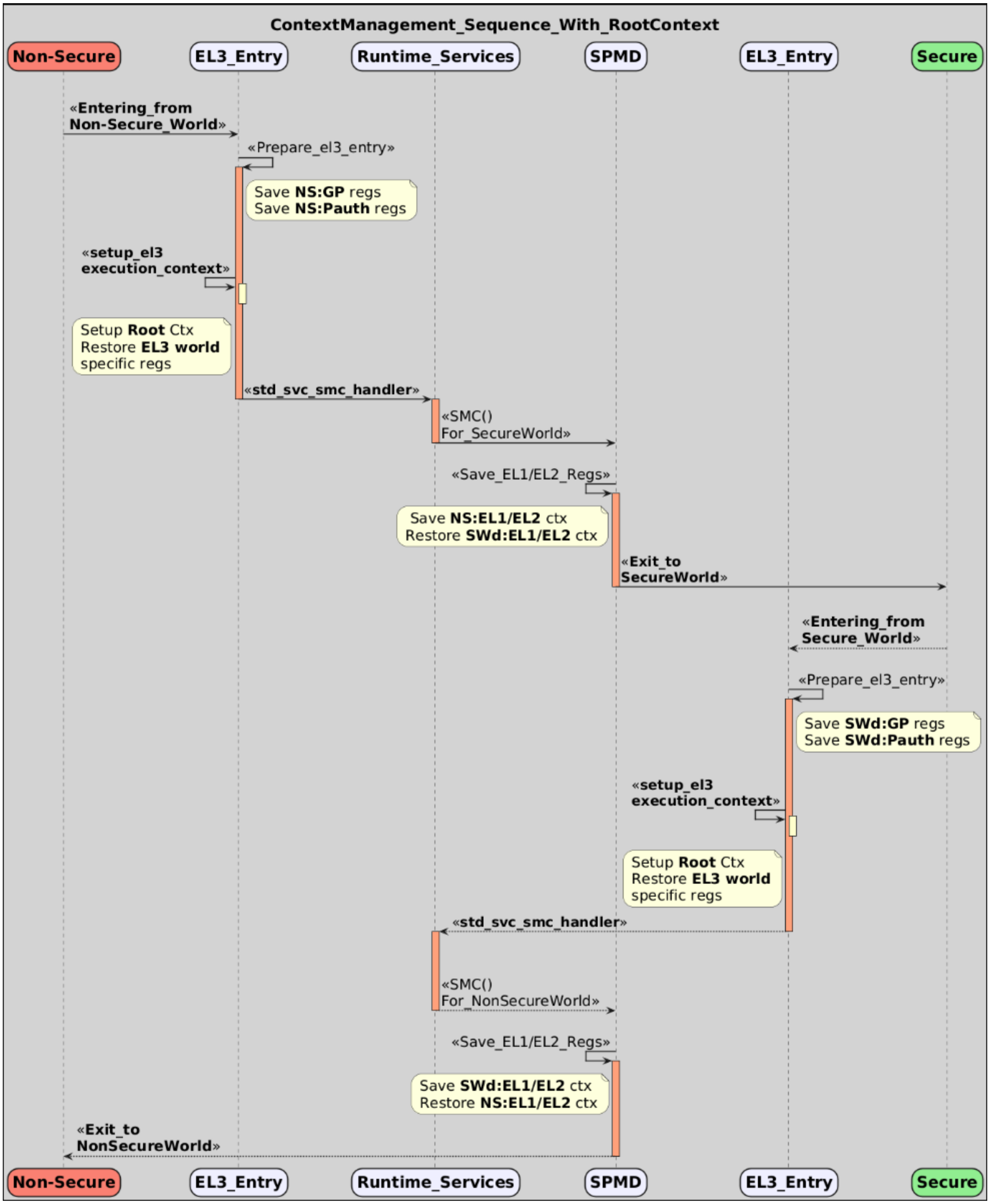
Previously, while the CPU is in execution at EL3, the system registers persist with the values of the incoming world. This implies that if the CPU is entering EL3 from NS world, the EL1 and EL2 system registers which might be modified in lower exception levels NS(EL2/EL1) will carry forward those values to EL3. Further the EL3 registers also hold on to the values configured for Non-secure world, written during the previous ERET from EL3 to NS(EL2/EL1). Same policy is followed with respect to other worlds (Secure/Realm) depending on the system configuration.

The firmware at EL3 has traditionally operated within the context of the incoming world (Secure/Non-Secure/Realm). This becomes problematic in scenarios where the EL3/Root world must explicitly use architectural features that depend on system registers configured for lower exception levels. A good example of this is the PAuth regs. The Root world would need to program its own PAuth Keys while executing in EL3 and this needs to be restored in entry to EL3 from any world. Therefore, Root world should maintain its own distinct settings to access features for its own execution at EL3.

Register values which are currently known to be of importance during EL3 execution, is referred to as the EL3/Root context. This includes (MDCR_EL3.SDD, SCR_EL3.{EA, SIF}, PMCR_EL0.DP, PSTATE.DIT) EL3 Context ensures, CPU executes under fixed EL3 system register settings which is not affected by settings of other worlds.

Root Context needs to be setup as early as possible before we try and access/modify architectural features at EL3. Its a simple restore operation `setup_el3_execution_context` that overwrites the selected bits listed above. EL3 never changes its mind about what those values should be, sets it as required for EL3. Henceforth, a Root context save operation is not required.

The figure below illustrates the same with NS-world as a reference while entering EL3.




```
# EL3/Root_Context routine
.macro setup_el3_execution_context
```

EL3 execution context needs to setup at both boot time (cold and warm boot) entrypaths and at all the possible exception handlers routing to EL3 at runtime.

4.26 NUMA-Aware Per-CPU Framework

- *Introduction*
- *Design*
  - `.per_cpu` *Section*
  - *Definer Interfaces*
  - *Accessor Interfaces*
- *Platform Responsibilities (NUMA-only)*
  - *1. Enable the Framework*
  - *2. Provide Per-CPU Section Base Address Data*
  - *3. Implement Required Platform Hooks*
- *References*

4.26.1 Introduction

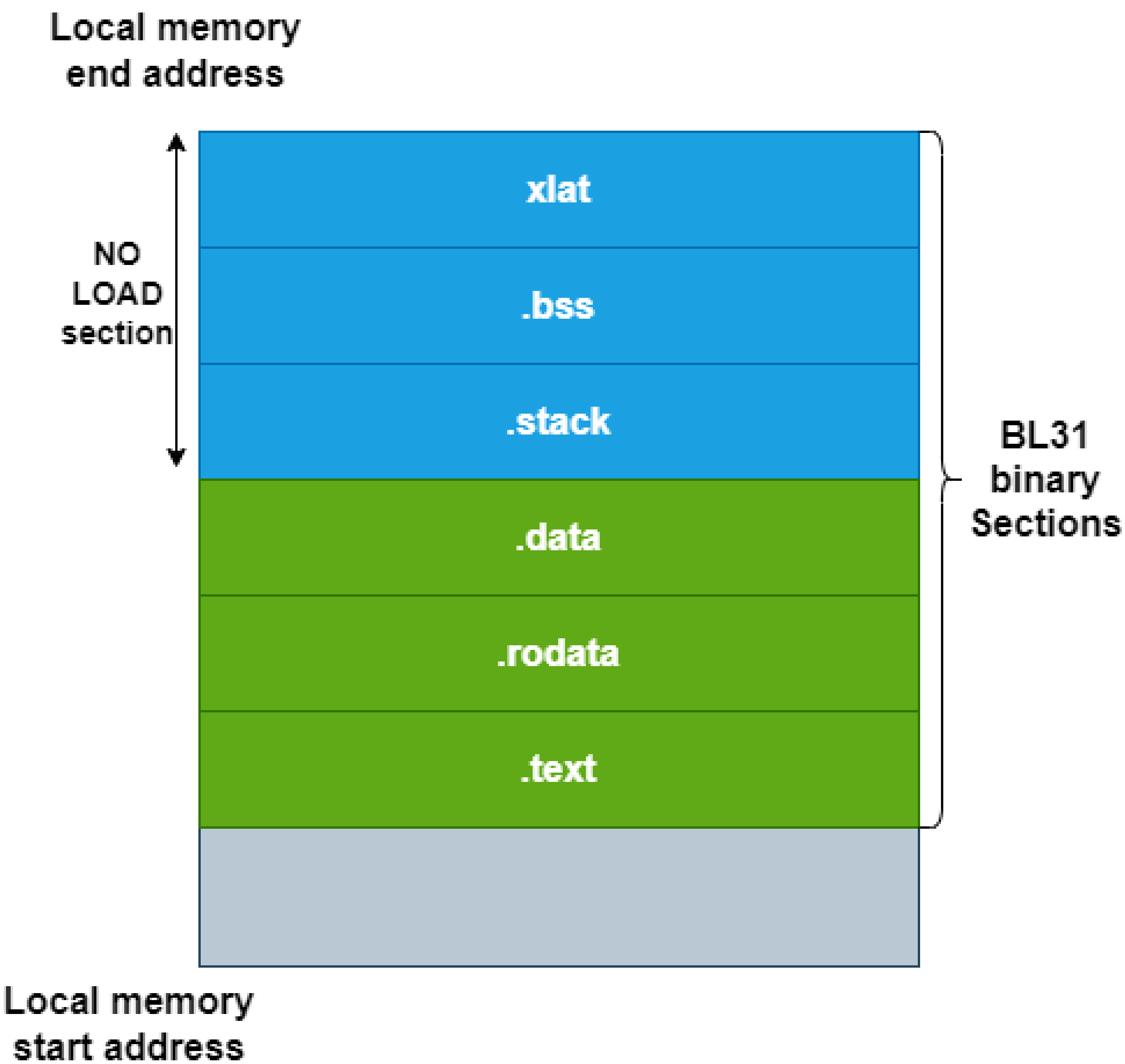
Modern system designs increasingly adopt multi-node architectures, where the system is divided into multiple topological units such as chiplets, sockets, or other isolated compute and memory units. Each node typically has its own local memory, and CPUs within a node can access this memory with lower latency than CPUs on remote nodes. In TF-A's current implementation, per-CPU data (for example, PSCI or SPM context) is stored in a global array or contiguous region, usually located in the memory of a single node. This approach introduces two key issues in multi-node systems:

- **Storage Constraints:** As systems scale to include more CPUs and nodes, this centralized allocation becomes a bottleneck. The memory capacity of a single node may be insufficient to hold per-CPU data for all CPUs. This constraint limits scalability in systems where each node has limited local memory.

  Figure: Typical BL31/BL32 binary storage in local memory

- **Non-Uniform Memory Access (NUMA) Latency:** In multi-node systems, memory access across nodes incurs additional latency because of interconnect traversal. When per-CPU data is centralized on a single node, CPUs on remote nodes must access that data via the interconnect, leading to increased latency for frequent operations such as context switching, exception handling, and crash reporting. This violates NUMA design principles, where data locality is critical to achieving performance and scalability.

To address these challenges, TF-A provides the NUMA-Aware Per-CPU Framework. The framework optimizes the allocation and access of per-CPU objects by letting platforms place them in the nodes with the lowest access latency.


Local memory
end address
NO
LOAD
section
xlat
.bss
.stack
.data
.rodata
.text
BL31
binary
Sections
Local memory
start address

4.26.2 Design

The framework provides standardized interfaces and mechanisms for **allocating**, **defining**, and **accessing** per-CPU data in a NUMA-aware environment. This ensures portability and maintainability across different platforms while optimizing for performance in multi-node systems.

`.per_cpu` Section

The framework dedicates a zero-initialized, cache-aligned `.per_cpu` section to **allocate** per-CPU global variables and ensure that these objects reside in the local memory of each NUMA node. The figure below illustrates how per-CPU objects are allocated in the local memory of their respective nodes.

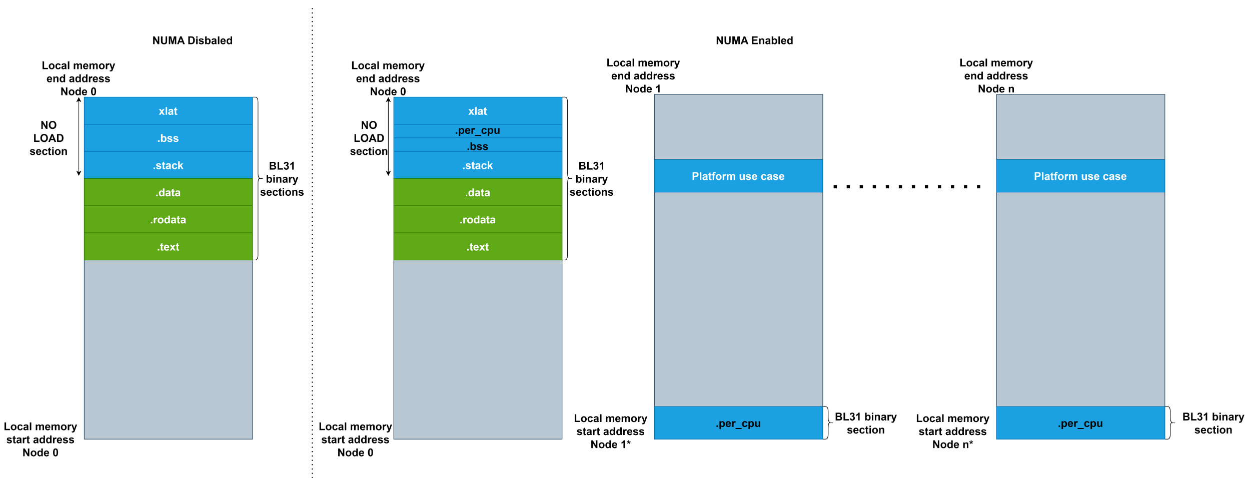


Figure: BL31/BL32 binary storage in local memory of per node when per-cpu NUMA framework is enabled

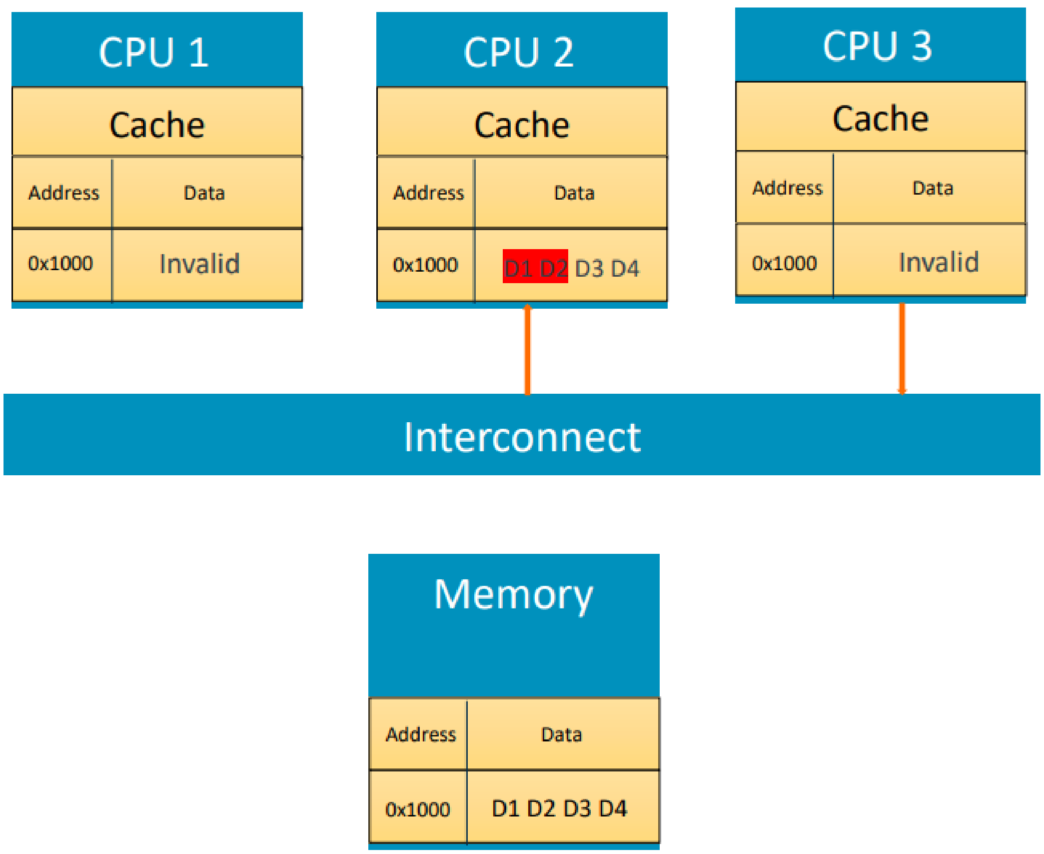
At link time, TF-A linker scripts carve out this section and publish section bounds and per-object stride via internal symbols so that they can be replicated and initialized across the non-primary nodes.

This linker section also addresses a common performance issue in modern multi-CPU systems known as **false sharing**. This issue arises when multiple CPUs access different addresses that lie on the same cache line. Although the accessed variables may be logically independent, their proximity in memory can result in repeated cache invalidations and reloads. Cache-coherency mechanisms operate at the granularity of cache lines (typically 64 bytes). If two CPUs write to different addresses within the same cache line, the line bounces between cores and incurs unnecessary overhead.

Figure: Two processors modifying different variables placed too closely in memory, leading to false sharing

To eliminate false sharing, this framework employs **linker-script-based alignment**, which:

- Places all per-CPU variables into a **dedicated, aligned** section (`.per_cpu`).
- Aligns that section using the cache granularity size (`CACHE_WRITEBACK_GRANULE`).


CPU 1
Cache
Address
Data
0x1000
Invalid
CPU 2
Cache
Address
Data
0x1000
D1 D2 D3 D4
CPU 3
Cache
Address
Data
0x1000
Invalid
Interconnect
Memory
Address
Data
0x1000
D1 D2 D3 D4

Definer Interfaces

The NUMA-Aware Per-CPU Framework provides a set of macros to define and declare per-CPU objects efficiently in multi-node systems.

- `PER_CPU_DECLARE(TYPE, NAME)`

  Declares an external per-CPU object so that other translation units can refer to it without allocating storage.

- `PER_CPU_DEFINE(TYPE, NAME)`

  Defines a per-CPU object and assigns it to `PER_CPU_SECTION_NAME` so the linker emits it into the `.per_cpu` section that the framework manages.

Accessor Interfaces

The NUMA-Aware Per-CPU Framework also provides macros to access per-CPU objects efficiently in multi-node systems.

- `PER_CPU_BY_INDEX(NAME, CPU)`

  Returns a pointer to the per-CPU object `NAME` for the specified CPU by combining the per-node base with the object's offset within `.per_cpu`.

- `PER_CPU_CUR(NAME)`

  Returns a pointer to the per-CPU object `NAME` for the current CPU.

For use in assembly routines, a corresponding macro version is provided:

In assembly routines, the `per_cpu_cur` helper macro performs the same calculation. It accepts the label of the per-CPU object and optional register arguments (destination and clobber) to materialize the per-CPU pointer without duplicating addressing logic in assembly files.

4.26.3 Platform Responsibilities (NUMA-only)

When NUMA is enabled, the platform is required to comply with some additional requirements in order for the runtime to correctly set up per-CPU sections on remote nodes:

1. Enable the Framework

Set `PLATFORM_NODE_COUNT` to a value greater than 1 (>=2) in the platform makefile to enable NUMA-aware per-CPU support:

```
PLATFORM_NODE_COUNT := 2  # >= 2 enables NUMA-aware per-CPU support
```

Platforms that are not multi-node do not need to modify this value because the default `PLATFORM_NODE_COUNT` is 1. The NUMA framework is not supported in 32-bit images such as BL32 SP_MIN.

2. Provide Per-CPU Section Base Address Data

Ensure that the platform can supply the base address of the `.per_cpu` section for each node and CPU when implementing `plat_per_cpu_node_base` and `plat_per_cpu_base`. The framework does not mandate how this information is obtained, only that each hook returns a valid base address. Platforms may:

- derive the base addresses from platform descriptors or firmware configuration data;
- read them from device tree nodes or other runtime discovery mechanisms; or
- encode them in platform-specific tables compiled into the image.

If a node described in platform data is not populated at runtime, the hooks may return `UINT64_MAX` to signal that no per-CPU section exists for that node.

The platform is free to maintain this mapping however it prefers, and may do so at either compile-time or through employing runtime discovery. The only requirement is that the `plat_per_cpu_node_base` and `plat_per_cpu_base` hooks translate a node or CPU identifier into the base address of the corresponding `.per_cpu` section.

Platform-defined regions that hold remote per-CPU sections must have page-aligned bases and sizes for page table mapping through the xlat library, which requires page alignment for mapped entries. The per-CPU section itself requires only cache writeback granule alignment for its base.

3. Implement Required Platform Hooks

Provide the following platform-specific functions:

- `uintptr_t plat_per_cpu_base(uint64_t cpu)`

  Returns the base address of the `.per_cpu` section for the specified CPU.

- `uintptr_t plat_per_cpu_node_base(uint64_t node)`

  Returns the base address of the `.per_cpu` section for the specified node.

- `uintptr_t plat_per_cpu_dcache_clean(void)`

  Cleans the entire per-CPU section from the data cache. This ensures that any modifications made to per-CPU data are written back to memory, making them visible to other CPUs or system components that may access this memory. This step is especially important on platforms that do not support hardware-managed coherency early in the boot process.

4.26.4 References

- Original presentation: https://www.trustedfirmware.org/docs/NUMA-aware-PER-CPU-framework-18Jul24.pdf

4.27 CPU Monitor Service Calls

This document enumerates and describes the CPU Monitor Service Calls.

These are Service Calls defined by the implementation of the CPU Monitor. They are accessed via `SMC` ("SMC calls") instruction executed from Exception Levels below EL3. SMC calls for CPU Monitor Services:

- Follow SMC Calling Convention;
- Use SMC function IDs that fall in the CPU range, which are

SMC Function Identifier	Service Type
0x81000000 - 0x8100FFFF	SMC32: CPU Monitor Service Calls
0xC1000000 - 0xC100FFFF	SMC64: CPU Monitor Service Calls

CPU monitor services are as follows:

SMC Function Identifier	Service Type	FID's Usage
0x81000010 - 0x8100001F (SMC32)	CVE-2026-0995 mitigation for C1-Pro CPU	0 is in use. 1 - 15 are Reserved.
0xC1000010 - 0xC100001F (SMC64)		
0x81000020 - 0x8100FFFF (SMC32)	Reserved	Reserved
0xC1000020 - 0xC100FFFF (SMC64)		

Source definitions for CPU Monitor Service Calls used by TF-A are located in the `cpu_svc.h` header file.

CPU_SVC_VERSION_MAJOR	CPU_SVC_VERSION_MINOR	Changes
1	0	Added for mitigating for CVE-2026-0995 erratum for C1-Pro CPU .

Table 1: Showing different versions of CPU service and changes done with each version

Each sub service will have its own version, one FID allocated for sub service version.

Some ground rules when one should update top level version.

- CPU_SVC_VERSION_MAJOR is incremented when any of the sub service version discovery FID changes or the FID that was allocated for discovery changes. So any breaking subfeature discovery changes will lead to major version update.
- CPU_SVC_VERSION_MINOR is incremented when we add a new FID or a new sub service. For example adding an new monitor service at 0x20.

CHAPTER FIVE

SYSTEM DESIGN

5.1 Alternative Boot Flows

5.1.1 EL3 payloads alternative boot flow

On a pre-production system, the ability to execute arbitrary, bare-metal code at the highest exception level is required. It allows full, direct access to the hardware, for example to run silicon soak tests.

Although it is possible to implement some baremetal secure firmware from scratch, this is a complex task on some platforms, depending on the level of configuration required to put the system in the expected state.

Rather than booting a baremetal application, a possible compromise is to boot `EL3 payloads` through TF-A instead. This is implemented as an alternative boot flow, where a modified BL2 boots an EL3 payload, instead of loading the other BL images and passing control to BL31. It reduces the complexity of developing EL3 baremetal code by:

- putting the system into a known architectural state;
- taking care of platform secure world initialization;
- loading the SCP_BL2 image if required by the platform.

When booting an EL3 payload on Arm standard platforms, the configuration of the TrustZone controller is simplified such that only region 0 is enabled and is configured to permit secure access only. This gives full access to the whole DRAM to the EL3 payload.

The system is left in the same state as when entering BL31 in the default boot flow. In particular:

- Running in EL3;
- Current state is AArch64;
- Little-endian data access;
- All exceptions disabled;
- MMU disabled;
- Caches disabled.

Booting an EL3 payload

The EL3 payload image is a standalone image and is not part of the FIP. It is not loaded by TF-A. Therefore, there are 2 possible scenarios:

- The EL3 payload may reside in non-volatile memory (NVM) and execute in place. In this case, booting it is just a matter of specifying the right address in NVM through `EL3_PAYLOAD_BASE` when building TF-A.
- The EL3 payload needs to be loaded in volatile memory (e.g. DRAM) at run-time.

To help in the latter scenario, the `SPIN_ON_BL1_EXIT=1` build option can be used. The infinite loop that it introduces in BL1 stops execution at the right moment for a debugger to take control of the target and load the payload (for example, over JTAG).

It is expected that this loading method will work in most cases, as a debugger connection is usually available in a pre-production system. The user is free to use any other platform-specific mechanism to load the EL3 payload, though.

5.1.2 Preloaded BL33 alternative boot flow

Some platforms have the ability to preload BL33 into memory instead of relying on TF-A to load it. This may simplify packaging of the normal world code and improve performance in a development environment. When secure world cold boot is complete, TF-A simply jumps to a BL33 base address provided at build time.

For this option to be used, the `PRELOADED_BL33_BASE` build option has to be used when compiling TF-A. For example, the following command will create a FIP without a BL33 and prepare to jump to a BL33 image loaded at address 0x80000000:

```
make PRELOADED_BL33_BASE=0x80000000 PLAT=fvp all fip
```

5.2 Authentication Framework & Chain of Trust

The aim of this document is to describe the authentication framework implemented in Trusted Firmware-A (TF-A). This framework fulfills the following requirements:

1. It should be possible for a platform port to specify the Chain of Trust in terms of certificate hierarchy and the mechanisms used to verify a particular image/certificate.
2. The framework should distinguish between:
   - The mechanism used to encode and transport information, e.g. DER encoded X.509v3 certificates to ferry Subject Public Keys, hashes and non-volatile counters.
   - The mechanism used to verify the transported information i.e. the cryptographic libraries.

The framework has been designed following a modular approach illustrated in the next diagram:

```
+---------------+---------------+------------+
| Trusted       | Trusted       | Trusted    |
| Firmware      | Firmware      | Firmware   |
| Generic       | IO Framework  | Platform   |
| Code i.e.     | (IO)          | Port       |
| BL1/BL2 (GEN) |               | (PP)       |
+---------------+---------------+------------+
       ^               ^               ^
       |               |               |
       v               v               v
 +-----------+   +-----------+   +-----------+
 |           |   |           |   | Image     |
 | Crypto    |   | Auth      |   | Parser    |
 | Module    |<->| Module    |<->| Module    |
 | (CM)      |   | (AM)      |   | (IPM)     |
 |           |   |           |   |           |
 +-----------+   +-----------+   +-----------+
       ^                               ^
       |                               |
       v                               v
+----------------+             +-----------------+
| Cryptographic  |             | Image Parser    |
| Libraries (CL) |             | Libraries (IPL) |
+----------------+             +-----------------+
              |                |
              |                |
              |                |
              v                v
             +-----------------+
             | Misc. Libs e.g. |
             | ASN.1 decoder   |
             |                 |
             +-----------------+

DIAGRAM 1.
```

This document describes the inner details of the authentication framework and the abstraction mechanisms available to specify a Chain of Trust.

5.2.1 Framework design

This section describes some aspects of the framework design and the rationale behind them. These aspects are key to verify a Chain of Trust.

Chain of Trust

A CoT is basically a sequence of authentication images which usually starts with a root of trust and culminates in a single data image. The following diagram illustrates how this maps to a CoT for the BL31 image described in the TBBR-Client specification.

```
+------------------+       +-------------------+
| ROTPK/ROTPK Hash |------>| Trusted Key       |
+------------------+       | Certificate       |
                           | (Auth Image)      |
                          /+-------------------+
                         /            |
                        /             |
                       /              |
                      /               |
                     L                v
+------------------+       +-------------------+
| Trusted World    |------>| BL31 Key          |
| Public Key       |       | Certificate       |
+------------------+       | (Auth Image)      |
                           +-------------------+
                          /           |
                         /            |
                        /             |
                       /              |
                      /               v
+------------------+ L     +-------------------+
| BL31 Content     |------>| BL31 Content      |
| Certificate PK   |       | Certificate       |
+------------------+       | (Auth Image)      |
                           +-------------------+
                          /           |
                         /            |
                        /             |
                       /              |
                      /               v
+------------------+ L     +-------------------+
| BL31 Hash        |------>| BL31 Image        |
|                  |       | (Data Image)      |
+------------------+       |                   |
                           +-------------------+

DIAGRAM 2.
```

The root of trust is usually a public key (ROTPK) that has been burnt in the platform and cannot be modified.

Image types

Images in a CoT are categorised as authentication and data images. An authentication image contains information to authenticate a data image or another authentication image. A data image is usually a boot loader binary, but it could be any other data that requires authentication.

Component responsibilities

For every image in a Chain of Trust, the following high level operations are performed to verify it:

1. Allocate memory for the image either statically or at runtime.
2. Identify the image and load it in the allocated memory.
3. Check the integrity of the image as per its type.
4. Authenticate the image as per the cryptographic algorithms used.
5. If the image is an authentication image, extract the information that will be used to authenticate the next image in the CoT.

In Diagram 1, each component is responsible for one or more of these operations. The responsibilities are briefly described below.

TF-A Generic code and IO framework (GEN/IO)

These components are responsible for initiating the authentication process for a particular image in BL1 or BL2. For each BL image that requires authentication, the Generic code asks recursively the Authentication module what is the parent image until either an authenticated image or the ROT is reached. Then the Generic code calls the IO framework to load the image and calls the Authentication module to authenticate it, following the CoT from ROT to Image.

TF-A Platform Port (PP)

The platform is responsible for:

1. Specifying the CoT for each image that needs to be authenticated. Details of how a CoT can be specified by the platform are explained later. The platform also specifies the authentication methods and the parsing method used for each image.
2. Statically allocating memory for each parameter in each image which is used for verifying the CoT, e.g. memory for public keys, hashes etc.
3. Providing the ROTPK or a hash of it.
4. Providing additional information to the IPM to enable it to identify and extract authentication parameters contained in an image, e.g. if the parameters are stored as X509v3 extensions, the corresponding OID must be provided.
5. Fulfill any other memory requirements of the IPM and the CM (not currently described in this document).

6. Export functions to verify an image which uses an authentication method that cannot be interpreted by the CM, e.g. if an image has to be verified using a NV counter, then the value of the counter to compare with can only be provided by the platform.
7. Export a custom IPM if a proprietary image format is being used (described later).

Authentication Module (AM)

It is responsible for:

1. Providing the necessary abstraction mechanisms to describe a CoT. Amongst other things, the authentication and image parsing methods must be specified by the PP in the CoT.
2. Verifying the CoT passed by GEN by utilising functionality exported by the PP, IPM and CM.
3. Tracking which images have been verified. In case an image is a part of multiple CoTs then it should be verified only once e.g. the Trusted World Key Certificate in the TBBR-Client spec. contains information to verify SCP_BL2, BL31, BL32 each of which have a separate CoT. (This responsibility has not been described in this document but should be trivial to implement).
4. Reusing memory meant for a data image to verify authentication images e.g. in the CoT described in Diagram 2, each certificate can be loaded and verified in the memory reserved by the platform for the BL31 image. By the time BL31 (the data image) is loaded, all information to authenticate it will have been extracted from the parent image i.e. BL31 content certificate. It is assumed that the size of an authentication image will never exceed the size of a data image. It should be possible to verify this at build time using asserts.

Cryptographic Module (CM)

The CM is responsible for providing an API to:

1. Verify a digital signature.
2. Verify a hash.

The CM does not include any cryptography related code, but it relies on an external library to perform the cryptographic operations. A Crypto-Library (CL) linking the CM and the external library must be implemented. The following functions must be provided by the CL:

```
void (*init)(void);
int (*verify_signature)(
                        /* Data to verify. */
                        void *data_ptr, unsigned int data_len,
                        /* Bit string of the signature in DER format. */
                        void *sig_ptr, unsigned int sig_len,
                        /* ASN1 SignatureAlgorithm struct. */
                        void *sig_alg, unsigned int sig_alg_len,
                        /* ASN1 SubjectPublicKeyInfo struct. */
                        void *pk_ptr, unsigned int pk_len);
int (*calc_hash)(
```

(continues on next page)

(continued from previous page)

```
                                    /* SHA256, SHA384 and SHA512 can be used. */
                                    enum crypto_md_algo alg
                                    /* Data to hash. */
                                    void *data_ptr, unsigned int data_len,
                                    /* Buffer to store the output. */
                                    unsigned char output[CRYPTO_MD_MAX_SIZE]);
int (*verify_hash)(
                                    /* Data to verify. */
                                    void *data_ptr, unsigned int data_len,
                                    /* ASN1 DigestInfo struct. */
                                    void *digest_info_ptr, unsigned int digest_info_len);
int (*auth_decrypt)(
                                    /* Currently AES-GCM is the only supported alg. */
                                    enum crypto_dec_algo dec_algo,
                                    /* Data to decrypt. */
                                    void *data_ptr, size_t len,
                                    /* Decryption key. */
                                    const void *key, unsigned int key_len,
                                    unsigned int key_flags,
                                    /* Initialization vector. */
                                    const void *iv, unsigned int iv_len,
                                    /* Authentication tag. */
                                    const void *tag, unsigned int tag_len);
```

The above functions return values from the enum `crypto_ret_value`. The functions are registered in the CM using the macro:

```
REGISTER_CRYPTO_LIB(_name,
                    _init,
                    _verify_signature,
                    _verify_hash,
                    _calc_hash,
                    _auth_decrypt,
                    _convert_pk);
```

`_name` must be a string containing the name of the CL. This name is used for debugging purposes.

The `_init` function is used to perform any initialization required for the specific CM and CL.

The `_verify_signature` function is used to verify certificates, and `_verify_hash` is used to verify raw images.

The `_calc_hash` function is mainly used in the `MEASURED_BOOT` and `DRTM_SUPPORT` features to calculate the hashes of various images/data.

The `_auth_decrypt` function uses an authentication tag to perform authenticated decryption, providing guarantees on the authenticity of encrypted data. This function is used when the optional encrypted firmware feature is enabled, that is when `ENCRYPT_BL31` or `ENCRYPT_BL32` are set to `1` and `DECRYPTION_SUPPORT` is set to `aes_gcm`.

Optionally, a platform function can be provided to convert public key (_convert_pk). It is only used if the platform saves a hash of the ROTPK. Most platforms save the hash of the ROTPK, but some may save slightly different information - e.g the hash of the ROTPK plus some related information. Defining this function allows to transform the ROTPK used to verify the signature to the buffer (a platform specific public key) which hash is saved in OTP.

```
int (*convert_pk)(void *full_pk_ptr, unsigned int full_pk_len,
                  void **hashed_pk_ptr, unsigned int *hashed_pk_len);
```

- `full_pk_ptr`: Pointer to Distinguished Encoding Rules (DER) ROTPK.
- `full_pk_len`: DER ROTPK size.
- `hashed_pk_ptr`: to return a pointer to a buffer, which hash should be the one saved in OTP.
- `hashed_pk_len`: previous buffer size

Image Parser Module (IPM)

The IPM is responsible for:

1. Checking the integrity of each image loaded by the IO framework.
2. Extracting parameters used for authenticating an image based upon a description provided by the platform in the CoT descriptor.

Images may have different formats (for example, authentication images could be x509v3 certificates, signed ELF files or any other platform specific format). The IPM allows to register an Image Parser Library (IPL) for every image format used in the CoT. This library must implement the specific methods to parse the image. The IPM obtains the image format from the CoT and calls the right IPL to check the image integrity and extract the authentication parameters.

See Section "Describing the image parsing methods" for more details about the mechanism the IPM provides to define and register IPLs.

Authentication methods

The AM supports the following authentication methods:

1. Hash
2. Digital signature

The platform may specify these methods in the CoT in case it decides to define a custom CoT instead of reusing a predefined one.

If a data image uses multiple methods, then all the methods must be a part of the same CoT. The number and type of parameters are method specific. These parameters should be obtained from the parent image using the IPM.

1. Hash

   Parameters:

   1. A pointer to data to hash
   2. Length of the data

3. A pointer to the hash
4. Length of the hash

The hash will be represented by the DER encoding of the following ASN.1 type:

```
DigestInfo ::= SEQUENCE {
    digestAlgorithm  DigestAlgorithmIdentifier,
    digest           Digest
}
```

This ASN.1 structure makes it possible to remove any assumption about the type of hash algorithm used as this information accompanies the hash. This should allow the Cryptography Library (CL) to support multiple hash algorithm implementations.

2. Digital Signature

   Parameters:

   1. A pointer to data to sign
   2. Length of the data
   3. Public Key Algorithm
   4. Public Key value
   5. Digital Signature Algorithm
   6. Digital Signature value

   The Public Key parameters will be represented by the DER encoding of the following ASN.1 type:

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
    algorithm         AlgorithmIdentifier{PUBLIC-KEY,
↪{PublicKeyAlgorithms}},
    subjectPublicKey  BIT STRING  }
```

   The Digital Signature Algorithm will be represented by the DER encoding of the following ASN.1 types.

```
AlgorithmIdentifier {ALGORITHM:IOSet } ::= SEQUENCE {
    algorithm         ALGORITHM.&id({IOSet}),
    parameters        ALGORITHM.&Type({IOSet}{@algorithm}) OPTIONAL
}
```

   The digital signature will be represented by:

```
signature  ::=  BIT STRING
```

The authentication framework will use the image descriptor to extract all the information related to authentication.

5.2.2 Specifying a Chain of Trust

A CoT can be described as a set of image descriptors linked together in a particular order. The order dictates the sequence in which they must be verified. Each image has a set of properties which allow the AM to verify it. These properties are described below.

The PP is responsible for defining a single or multiple CoTs for a data image. Unless otherwise specified, the data structures described in the following sections are populated by the PP statically.

Describing the image parsing methods

The parsing method refers to the format of a particular image. For example, an authentication image that represents a certificate could be in the X.509v3 format. A data image that represents a boot loader stage could be in raw binary or ELF format. The IPM supports three parsing methods. An image has to use one of the three methods described below. An IPL is responsible for interpreting a single parsing method. There has to be one IPL for every method used by the platform.

1. Raw format: This format is effectively a nop as an image using this method is treated as being in raw binary format e.g. boot loader images used by TF-A. This method should only be used by data images.
2. X509V3 method: This method uses industry standards like X.509 to represent PKI certificates (authentication images). It is expected that open source libraries will be available which can be used to parse an image represented by this method. Such libraries can be used to write the corresponding IPL e.g. the X.509 parsing library code in mbed TLS.
3. Platform defined method: This method caters for platform specific proprietary standards to represent authentication or data images. For example, The signature of a data image could be appended to the data image raw binary. A header could be prepended to the combined blob to specify the extents of each component. The platform will have to implement the corresponding IPL to interpret such a format.

The following enum can be used to define these three methods.

```
typedef enum img_type_enum {
    IMG_RAW,            /* Binary image */
    IMG_PLAT,           /* Platform specific format */
    IMG_CERT,           /* X509v3 certificate */
    IMG_MAX_TYPES,
} img_type_t;
```

An IPL must provide functions with the following prototypes:

```
void init(void);
int check_integrity(void *img, unsigned int img_len);
int get_auth_param(const auth_param_type_desc_t *type_desc,
                      void *img, unsigned int img_len,
                      void **param, unsigned int *param_len);
```

An IPL for each type must be registered using the following macro:

```
REGISTER_IMG_PARSER_LIB(_type, _name, _init, _check_int, _get_param)
```

- `_type`: one of the types described above.

- `_name`: a string containing the IPL name for debugging purposes.
- `_init`: initialization function pointer.
- `_check_int`: check image integrity function pointer.
- `_get_param`: extract authentication parameter function pointer.

The `init()` function will be used to initialize the IPL.

The `check_integrity()` function is passed a pointer to the memory where the image has been loaded by the IO framework and the image length. It should ensure that the image is in the format corresponding to the parsing method and has not been tampered with. For example, RFC-2459 describes a validation sequence for an X.509 certificate.

The `get_auth_param()` function is passed a parameter descriptor containing information about the parameter (`type_desc` and `cookie`) to identify and extract the data corresponding to that parameter from an image. This data will be used to verify either the current or the next image in the CoT sequence.

Each image in the CoT will specify the parsing method it uses. This information will be used by the IPM to find the right parser descriptor for the image.

Describing the authentication method(s)

As part of the CoT, each image has to specify one or more authentication methods which will be used to verify it. As described in the Section "Authentication methods", there are three methods supported by the AM.

```
typedef enum {
    AUTH_METHOD_NONE,
    AUTH_METHOD_HASH,
    AUTH_METHOD_SIG,
    AUTH_METHOD_NUM
} auth_method_type_t;
```

The AM defines the type of each parameter used by an authentication method. It uses this information to:

1. Specify to the `get_auth_param()` function exported by the IPM, which parameter should be extracted from an image.
2. Correctly marshall the parameters while calling the verification function exported by the CM and PP.
3. Extract authentication parameters from a parent image in order to verify a child image e.g. to verify the certificate image, the public key has to be obtained from the parent image.

```
typedef enum {
    AUTH_PARAM_NONE,
    AUTH_PARAM_RAW_DATA,     /* Raw image data */
    AUTH_PARAM_SIG,          /* The image signature */
    AUTH_PARAM_SIG_ALG,      /* The image signature algorithm */
    AUTH_PARAM_HASH,         /* A hash (including the algorithm) */
    AUTH_PARAM_PUB_KEY,      /* A public key */
    AUTH_PARAM_NV_CTR,       /* A non-volatile counter */
} auth_param_type_t;
```

The AM defines the following structure to identify an authentication parameter required to verify an image.

```
typedef struct auth_param_type_desc_s {
    auth_param_type_t type;
    void *cookie;
} auth_param_type_desc_t;
```

`cookie` is used by the platform to specify additional information to the IPM which enables it to uniquely identify the parameter that should be extracted from an image. For example, the hash of a BL3x image in its corresponding content certificate is stored in an X509v3 custom extension field. An extension field can only be identified using an OID. In this case, the `cookie` could contain the pointer to the OID defined by the platform for the hash extension field while the `type` field could be set to `AUTH_PARAM_HASH`. A value of 0 for the `cookie` field means that it is not used.

For each method, the AM defines a structure with the parameters required to verify the image.

```
/*
 * Parameters for authentication by hash matching
 */
typedef struct auth_method_param_hash_s {
    auth_param_type_desc_t *data;   /* Data to hash */
    auth_param_type_desc_t *hash;   /* Hash to match with */
} auth_method_param_hash_t;

/*
 * Parameters for authentication by signature
 */
typedef struct auth_method_param_sig_s {
    auth_param_type_desc_t *pk; /* Public key */
    auth_param_type_desc_t *sig;    /* Signature to check */
    auth_param_type_desc_t *alg;    /* Signature algorithm */
    auth_param_type_desc_t *tbs;    /* Data signed */
} auth_method_param_sig_t;
```

The AM defines the following structure to describe an authentication method for verifying an image

```
/*
 * Authentication method descriptor
 */
typedef struct auth_method_desc_s {
    auth_method_type_t type;
    union {
        auth_method_param_hash_t hash;
        auth_method_param_sig_t sig;
    } param;
} auth_method_desc_t;
```

Using the method type specified in the `type` field, the AM finds out what field needs to access within the `param` union.

Storing Authentication parameters

A parameter described by `auth_param_type_desc_t` to verify an image could be obtained from either the image itself or its parent image. The memory allocated for loading the parent image will be reused for loading the child image. Hence parameters which are obtained from the parent for verifying a child image need to have memory allocated for them separately where they can be stored. This memory must be statically allocated by the platform port.

The AM defines the following structure to store the data corresponding to an authentication parameter.

```
typedef struct auth_param_data_desc_s {
    void *auth_param_ptr;
    unsigned int auth_param_len;
} auth_param_data_desc_t;
```

The `auth_param_ptr` field is initialized by the platform. The `auth_param_len` field is used to specify the length of the data in the memory.

For parameters that can be obtained from the child image itself, the IPM is responsible for populating the `auth_param_ptr` and `auth_param_len` fields while executing the `img_get_auth_param()` function.

The AM defines the following structure to enable an image to describe the parameters that should be extracted from it and used to verify the next image (child) in a CoT.

```
typedef struct auth_param_desc_s {
    auth_param_type_desc_t type_desc;
    auth_param_data_desc_t data;
} auth_param_desc_t;
```

Describing an image in a CoT

An image in a CoT is a consolidation of the following aspects of a CoT described above.

1. A unique identifier specified by the platform which allows the IO framework to locate the image in a FIP and load it in the memory reserved for the data image in the CoT.
2. A parsing method which is used by the AM to find the appropriate IPM.
3. Authentication methods and their parameters as described in the previous section. These are used to verify the current image.
4. Parameters which are used to verify the next image in the current CoT. These parameters are specified only by authentication images and can be extracted from the current image once it has been verified.

The following data structure describes an image in a CoT.

```
typedef struct auth_img_desc_s {
    unsigned int img_id;
    const struct auth_img_desc_s *parent;
    img_type_t img_type;
    const auth_method_desc_t *const img_auth_methods;
```

(continues on next page)

(continued from previous page)

```
    const auth_param_desc_t *const authenticated_data;
} auth_img_desc_t;
```

A CoT is defined as an array of pointers to `auth_image_desc_t` structures linked together by the `parent` field. Those nodes with no parent must be authenticated using the ROTPK stored in the platform.

5.2.3 Implementation example

This section is a detailed guide explaining a trusted boot implementation using the authentication framework. This example corresponds to the Applicative Functional Mode (AFM) as specified in the TBBR-Client document. It is recommended to read this guide along with the source code.

The TBBR CoT

CoT specific to BL1 and BL2 can be found in `drivers/auth/tbbr/tbbr_cot_bl1.c` and `drivers/auth/tbbr/tbbr_cot_bl2.c` respectively. The common CoT used across BL1 and BL2 can be found in `drivers/auth/tbbr/tbbr_cot_common.c`. This CoT consists of an array of pointers to image descriptors and it is registered in the framework using the macro `REGISTER_COT(cot_desc)`, where `cot_desc` must be the name of the array (passing a pointer or any other type of indirection will cause the registration process to fail).

The number of images participating in the boot process depends on the CoT. There is, however, a minimum set of images that are mandatory in TF-A and thus all CoTs must present:

- `BL2`
- `SCP_BL2` (platform specific)
- `BL31`
- `BL32` (optional)
- `BL33`

The TBBR specifies the additional certificates that must accompany these images for a proper authentication. Details about the TBBR CoT may be found in the *Trusted Board Boot* document.

Following the *Porting Guide*, a platform must provide unique identifiers for all the images and certificates that will be loaded during the boot process. If a platform is using the TBBR as a reference for trusted boot, these identifiers can be obtained from `include/common/tbbr/tbbr_img_def.h`. Arm platforms include this file in `include/plat/arm/common/arm_def.h`. Other platforms may also include this file or provide their own identifiers.

Important: the authentication module uses these identifiers to index the CoT array, so the descriptors location in the array must match the identifiers.

Each image descriptor must specify:

- `img_id`: the corresponding image unique identifier defined by the platform.

- `img_type`: the image parser module uses the image type to call the proper parsing library to check the image integrity and extract the required authentication parameters. Three types of images are currently supported:
  - `IMG_RAW`: image is a raw binary. No parsing functions are available, other than reading the whole image.
  - `IMG_PLAT`: image format is platform specific. The platform may use this type for custom images not directly supported by the authentication framework.
  - `IMG_CERT`: image is an x509v3 certificate.
- `parent`: pointer to the parent image descriptor. The parent will contain the information required to authenticate the current image. If the parent is NULL, the authentication parameters will be obtained from the platform (i.e. the BL2 and Trusted Key certificates are signed with the ROT private key, whose public part is stored in the platform).
- `img_auth_methods`: this points to an array which defines the authentication methods that must be checked to consider an image authenticated. Each method consists of a type and a list of parameter descriptors. A parameter descriptor consists of a type and a cookie which will point to specific information required to extract that parameter from the image (i.e. if the parameter is stored in an x509v3 extension, the cookie will point to the extension OID). Depending on the method type, a different number of parameters must be specified. This pointer should not be NULL. Supported methods are:
  - `AUTH_METHOD_HASH`: the hash of the image must match the hash extracted from the parent image. The following parameter descriptors must be specified:
    - `data`: data to be hashed (obtained from current image)
    - `hash`: reference hash (obtained from parent image)
  - `AUTH_METHOD_SIG`: the image (usually a certificate) must be signed with the private key whose public part is extracted from the parent image (or the platform if the parent is NULL). The following parameter descriptors must be specified:
    - `pk`: the public key (obtained from parent image)
    - `sig`: the digital signature (obtained from current image)
    - `alg`: the signature algorithm used (obtained from current image)
    - `data`: the data to be signed (obtained from current image)
- `authenticated_data`: this array pointer indicates what authentication parameters must be extracted from an image once it has been authenticated. Each parameter consists of a parameter descriptor and the buffer address/size to store the parameter. The CoT is responsible for allocating the required memory to store the parameters. This pointer may be NULL.

In the `tbbr_cot*.c` file, a set of buffers are allocated to store the parameters extracted from the certificates. In the case of the TBBR CoT, these parameters are hashes and public keys. In DER format, an RSA-4096 public key requires 550 bytes, and a hash requires 51 bytes. Depending on the CoT and the authentication process, some of the buffers may be reused at different stages during the boot.

Next in that file, the parameter descriptors are defined. These descriptors will be used to extract the parameter data from the corresponding image.

Example: the BL31 Chain of Trust

Four image descriptors form the BL31 Chain of Trust:

```
static const auth_img_desc_t trusted_key_cert = {
        .img_id = TRUSTED_KEY_CERT_ID,
        .img_type = IMG_CERT,
        .parent = NULL,
        .img_auth_methods =  (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_SIG,
                        .param.sig = {
                                .pk = &subject_pk,
                                .sig = &sig,
                                .alg = &sig_alg,
                                .data = &raw_data
                        }
                },
                [1] = {
                        .type = AUTH_METHOD_NV_CTR,
                        .param.nv_ctr = {
                                .cert_nv_ctr = &trusted_nv_ctr,
                                .plat_nv_ctr = &trusted_nv_ctr
                        }
                }
        },
        .authenticated_data = (const auth_param_desc_t[COT_MAX_VERIFIED_
↪PARAMS]) {
                [0] = {
                        .type_desc = &trusted_world_pk,
                        .data = {
                                .ptr = (void *)trusted_world_pk_buf,
                                .len = (unsigned int)PK_DER_LEN
                        }
                },
                [1] = {
                        .type_desc = &non_trusted_world_pk,
                        .data = {
                                .ptr = (void *)non_trusted_world_pk_buf,
                                .len = (unsigned int)PK_DER_LEN
                        }
                }
        }
};
static const auth_img_desc_t soc_fw_key_cert = {
        .img_id = SOC_FW_KEY_CERT_ID,
        .img_type = IMG_CERT,
        .parent = &trusted_key_cert,
        .img_auth_methods =  (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_SIG,
                        .param.sig = {
```

(continues on next page)

(continued from previous page)

```
                                .pk = &trusted_world_pk,
                                .sig = &sig,
                                .alg = &sig_alg,
                                .data = &raw_data
                        }
                },
                [1] = {
                        .type = AUTH_METHOD_NV_CTR,
                        .param.nv_ctr = {
                                .cert_nv_ctr = &trusted_nv_ctr,
                                .plat_nv_ctr = &trusted_nv_ctr
                        }
                }
        },
        .authenticated_data = (const auth_param_desc_t[COT_MAX_VERIFIED_
↪PARAMS]) {
                [0] = {
                        .type_desc = &soc_fw_content_pk,
                        .data = {
                                .ptr = (void *)content_pk_buf,
                                .len = (unsigned int)PK_DER_LEN
                        }
                }
        }
};
static const auth_img_desc_t soc_fw_content_cert = {
        .img_id = SOC_FW_CONTENT_CERT_ID,
        .img_type = IMG_CERT,
        .parent = &soc_fw_key_cert,
        .img_auth_methods =  (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_SIG,
                        .param.sig = {
                                .pk = &soc_fw_content_pk,
                                .sig = &sig,
                                .alg = &sig_alg,
                                .data = &raw_data
                        }
                },
                [1] = {
                        .type = AUTH_METHOD_NV_CTR,
                        .param.nv_ctr = {
                                .cert_nv_ctr = &trusted_nv_ctr,
                                .plat_nv_ctr = &trusted_nv_ctr
                        }
                }
        },
        .authenticated_data = (const auth_param_desc_t[COT_MAX_VERIFIED_
↪PARAMS]) {
                [0] = {
                        .type_desc = &soc_fw_hash,
                        .data = {
```

(continues on next page)

(continued from previous page)

```
                                .ptr = (void *)soc_fw_hash_buf,
                                .len = (unsigned int)HASH_DER_LEN
                        }
                },
                [1] = {
                        .type_desc = &soc_fw_config_hash,
                        .data = {
                                .ptr = (void *)soc_fw_config_hash_buf,
                                .len = (unsigned int)HASH_DER_LEN
                        }
                }
        }
};
static const auth_img_desc_t bl31_image = {
        .img_id = BL31_IMAGE_ID,
        .img_type = IMG_RAW,
        .parent = &soc_fw_content_cert,
        .img_auth_methods =  (const auth_method_desc_t[AUTH_METHOD_NUM]) {
                [0] = {
                        .type = AUTH_METHOD_HASH,
                        .param.hash = {
                                .data = &raw_data,
                                .hash = &soc_fw_hash
                        }
                }
        }
};
```

The **Trusted Key certificate** is signed with the ROT private key and contains the Trusted World public key and the Non-Trusted World public key as x509v3 extensions. This must be specified in the image descriptor using the `img_auth_methods` and `authenticated_data` arrays, respectively.

The Trusted Key certificate is authenticated by checking its digital signature using the ROTPK. Four parameters are required to check a signature: the public key, the algorithm, the signature and the data that has been signed. Therefore, four parameter descriptors must be specified with the authentication method:

- `subject_pk`: parameter descriptor of type `AUTH_PARAM_PUB_KEY`. This type is used to extract a public key from the parent image. If the cookie is an OID, the key is extracted from the corresponding x509v3 extension. If the cookie is NULL, the subject public key is retrieved. In this case, because the parent image is NULL, the public key is obtained from the platform (this key will be the ROTPK).
- `sig`: parameter descriptor of type `AUTH_PARAM_SIG`. It is used to extract the signature from the certificate.
- `sig_alg`: parameter descriptor of type `AUTH_PARAM_SIG`. It is used to extract the signature algorithm from the certificate.
- `raw_data`: parameter descriptor of type `AUTH_PARAM_RAW_DATA`. It is used to extract the data to be signed from the certificate.

Once the signature has been checked and the certificate authenticated, the Trusted World public key needs to be extracted from the certificate. A new entry is created in the `authenticated_data` array for that purpose.

In that entry, the corresponding parameter descriptor must be specified along with the buffer address to store the parameter value. In this case, the `trusted_world_pk` descriptor is used to extract the public key from an x509v3 extension with OID `TRUSTED_WORLD_PK_OID`. The BL31 key certificate will use this descriptor as parameter in the signature authentication method. The key is stored in the `trusted_world_pk_buf` buffer.

The **BL31 Key certificate** is authenticated by checking its digital signature using the Trusted World public key obtained previously from the Trusted Key certificate. In the image descriptor, we specify a single authentication method by signature whose public key is the `trusted_world_pk`. Once this certificate has been authenticated, we have to extract the BL31 public key, stored in the extension specified by `soc_fw_content_pk`. This key will be copied to the `content_pk_buf` buffer.

The **BL31 certificate** is authenticated by checking its digital signature using the BL31 public key obtained previously from the BL31 Key certificate. We specify the authentication method using `soc_fw_content_pk` as public key. After authentication, we need to extract the BL31 hash, stored in the extension specified by `soc_fw_hash`. This hash will be copied to the `soc_fw_hash_buf` buffer.

The **BL31 image** is authenticated by calculating its hash and matching it with the hash obtained from the BL31 certificate. The image descriptor contains a single authentication method by hash. The parameters to the hash method are the reference hash, `soc_fw_hash`, and the data to be hashed. In this case, it is the whole image, so we specify `raw_data`.

The image parser library

The image parser module relies on libraries to check the image integrity and extract the authentication parameters. The number and type of parser libraries depend on the images used in the CoT. Raw images do not need a library, so only an x509v3 library is required for the TBBR CoT.

Arm platforms will use an x509v3 library based on mbed TLS. This library may be found in `drivers/auth/mbedtls/mbedtls_x509_parser.c`. It exports three functions:

```
void init(void);
int check_integrity(void *img, unsigned int img_len);
int get_auth_param(const auth_param_type_desc_t *type_desc,
                   void *img, unsigned int img_len,
                   void **param, unsigned int *param_len);
```

The library is registered in the framework using the macro `REGISTER_IMG_PARSER_LIB()`. Each time the image parser module needs to access an image of type `IMG_CERT`, it will call the corresponding function exported in this file.

The build system must be updated to include the corresponding library and mbed TLS sources. Arm platforms use the `arm_common.mk` file to pull the sources.

The cryptographic library

The cryptographic module relies on a library to perform essential operations such as verifying a hash or a digital signature. Arm platforms use a library based on mbedTLS located at `drivers/auth/mbedtls/mbedtls_crypto.c`. Additionally, an experimental alternative library based on PSA Crypto is available at `drivers/auth/mbedtls/mbedtls_psa_crypto.c`. In future, `mbedtls_psa_crypto.c` will replace `mbedtls_crypto.c` as the default Arm CM. Both libraries are registered in the authentication framework using the macro `REGISTER_CRYPTO_LIB()`. These libraries implement the following exported functions, their implementations are compared side-by-side below:

Table 1: Comparison of exported CM function implementations

CM function	`mbedtls_crypto.c`	`mbedtls_psa_crypto.c`
`init`	Initialize the heap for mbedTLS.	Initialize the heap for mbedTLS and call `psa_crypto_init`.
`verify_signature`	Use mbedTLS to parse the ASN1 inputs, and then use the mbedTLS pk module to verify the signature.	Use mbedTLS to parse the ASN1 inputs, use the mbedTLS pk module to parse the key, import it into the PSA key system and then use `psa_verify_message` to verify the signature.
`calc_hash`	Use the `mbedtls_md` API to calculate the hash of the given data.	Use `psa_hash_compute` to calculate the hash of the given data.
`verify_hash`	Use the `mbedtls_md` API to calculate the hash of the given data, and then compare it against the data which is to be verified.	Call `psa_hash_compare`, which both calculates the hash of the given data and compares this hash against the data to be verified.
`auth_decrypt`	Use the `mbedtls_gcm` API to decrypt the data, and then verify the returned tag by comparing it to the inputted tag.	Load the key into the PSA key store, and then use `psa_aead_verify` to decrypt and verify the tag.

The mbedTLS library algorithm support is configured by both the `TF_MBEDTLS_KEY_ALG` and `TF_MBEDTLS_KEY_SIZE` variables.

- `TF_MBEDTLS_KEY_ALG` can take in 3 values: *rsa*, *ecdsa* or *rsa+ecdsa*. This variable allows the Makefile to include the corresponding sources in the build for the various algorithms. Setting the variable to *rsa+ecdsa* enables support for both rsa and ecdsa algorithms in the mbedTLS library.
- `TF_MBEDTLS_KEY_SIZE` sets the supported RSA key size for TFA. Valid values include 1024, 2048, 3072 and 4096.
- `TF_MBEDTLS_USE_AES_GCM` enables the authenticated decryption support based on AES-GCM algorithm. Valid values are 0 and 1.

Note: If code size is a concern, the build option `MBEDTLS_SHA256_SMALLER` can be defined in the platform Makefile. It will make mbed TLS use an implementation of SHA-256 with smaller memory footprint (~1.5 KB less) but slower (~30%).

5.3 Arm CPU Specific Build Macros

This document describes the various build options present in the CPU specific operations framework to enable errata workarounds and to enable optimizations for a specific CPU on a platform.

5.3.1 Security Vulnerability Workarounds

TF-A exports a series of build flags which control which security vulnerability workarounds should be applied at runtime.

- `WORKAROUND_CVE_2017_5715`: Enables the security workaround for CVE-2017-5715. This flag can be set to 0 by the platform if none of the PEs in the system need the workaround. Setting this flag to 0 provides no performance benefit for non-affected platforms, it just helps to comply with the recommendation in the spec regarding workaround discovery. Defaults to 1.
- `WORKAROUND_CVE_2018_3639`: Enables the security workaround for CVE-2018-3639. Defaults to 1. The TF-A project recommends to keep the default value of 1 even on platforms that are unaffected by CVE-2018-3639, in order to comply with the recommendation in the spec regarding workaround discovery.
- `DYNAMIC_WORKAROUND_CVE_2018_3639`: Enables dynamic mitigation for CVE-2018-3639. This build option should be set to 1 if the target platform contains at least 1 CPU that requires dynamic mitigation. Defaults to 0.
- `WORKAROUND_CVE_2022_23960`: Enables mitigation for CVE-2022-23960. This build option should be set to 1 if the target platform contains at least 1 CPU that requires this mitigation. Defaults to 1.
- `WORKAROUND_CVE_2024_5660`: Enables mitigation for *CVE-2024-5660*. The fix is to disable hardware page aggregation by setting CPUECTLR_EL1[46] in EL3 FW. This build option should be set to 1 if the target platform contains at least 1 CPU that requires this mitigation. Defaults to 1.
- `WORKAROUND_CVE_2024_7881`: Enables mitigation for *CVE-2024-7881*. This build option should be set to 1 if the target platform contains at least 1 CPU that requires this mitigation. Defaults to 1.
- `WORKAROUND_CVE_2025_0647`: Enables mitigation for *CVE-2025-0647*. This build option should be set to 1 if the target platform contains at least 1 CPU that requires this mitigation. Defaults to 1.
- `WORKAROUND_CVE_2025_10263`: Enables mitigation for CVE-2025-10263. This build option should be set to 1 if the target platform contains at least 1 CPU that requires this mitigation. Defaults to 0, due to the nature of this workaround it cannot be checked at runtime so should not be enabled unless needed by the platform.

5.3.2 CPU Errata Workarounds

TF-A exports a series of build flags which control the errata workarounds that are applied to each CPU by the reset handler. The errata details can be found in the CPU specific errata documents published by Arm: For example: Cortex-A72 MPCore Software Developers Errata Notice

The errata workarounds are implemented for a particular revision or a set of processor revisions. This is checked by the reset handler at runtime. Each errata workaround is identified by its `ID` as specified in the processor's errata notice document. The format of the define used to enable/disable the errata workaround is `ERRATA_<Processor name>_<ID>`, where the `Processor name` is for example `A57` for the `Cortex_A57` CPU.

Refer to *CPU errata implementation* for information on how to write errata workaround functions.

All workarounds are disabled by default. The platform is responsible for enabling these workarounds according to its requirement by defining the errata workaround build flags in the platform specific makefile. In case these workarounds are enabled for the wrong CPU revision then the errata workaround is not applied. In the DEBUG build, this is indicated by printing a warning to the crash console.

In the current implementation, a platform which has more than 1 variant with different revisions of a processor has no runtime mechanism available for it to specify which errata workarounds should be enabled or not.

The value of the build flags is 0 by default, that is, disabled. A value of 1 will enable it.

For Cortex-A9, the following errata build flags are defined :

- `ERRATA_A9_794073`: This applies errata 794073 workaround to Cortex-A9 CPU. This needs to be enabled for all revisions of the CPU.

For Cortex-A15, the following errata build flags are defined :

- `ERRATA_A15_816470`: This applies errata 816470 workaround to Cortex-A15 CPU. This needs to be enabled only for revision >= r3p0 of the CPU.
- `ERRATA_A15_827671`: This applies errata 827671 workaround to Cortex-A15 CPU. This needs to be enabled only for revision >= r3p0 of the CPU.

For Cortex-A17, the following errata build flags are defined :

- `ERRATA_A17_852421`: This applies errata 852421 workaround to Cortex-A17 CPU. This needs to be enabled only for revision <= r1p2 of the CPU.
- `ERRATA_A17_852423`: This applies errata 852423 workaround to Cortex-A17 CPU. This needs to be enabled only for revision <= r1p2 of the CPU.

For Cortex-A35, the following errata build flags are defined :

- `ERRATA_A35_855472`: This applies errata 855472 workaround to Cortex-A35 CPUs. This needs to be enabled only for revision r0p0 of Cortex-A35.

For Cortex-A53, the following errata build flags are defined :

- `ERRATA_A53_819472`: This applies errata 819472 workaround to all CPUs. This needs to be enabled only for revision <= r0p1 of Cortex-A53.

- `ERRATA_A53_824069`: This applies errata 824069 workaround to all CPUs. This needs to be enabled only for revision <= r0p2 of Cortex-A53.
- `ERRATA_A53_826319`: This applies errata 826319 workaround to Cortex-A53 CPU. This needs to be enabled only for revision <= r0p2 of the CPU.
- `ERRATA_A53_827319`: This applies errata 827319 workaround to all CPUs. This needs to be enabled only for revision <= r0p2 of Cortex-A53.
- `ERRATA_A53_835769`: This applies erratum 835769 workaround at compile and link time to Cortex-A53 CPU. This needs to be enabled for some variants of revision <= r0p4. This workaround can lead the linker to create `*.stub` sections.
- `ERRATA_A53_836870`: This applies errata 836870 workaround to Cortex-A53 CPU. This needs to be enabled only for revision <= r0p3 of the CPU. From r0p4 and onwards, this errata is enabled by default in hardware. Identical to `A53_DISABLE_NON_TEMPORAL_HINT`.
- `ERRATA_A53_843419`: This applies erratum 843419 workaround at link time to Cortex-A53 CPU. This needs to be enabled for some variants of revision <= r0p4. This workaround can lead the linker to emit `*.stub` sections which are 4kB aligned.
- `ERRATA_A53_855873`: This applies errata 855873 workaround to Cortex-A53 CPUs. Though the erratum is present in every revision of the CPU, this workaround is only applied to CPUs from r0p3 onwards, which feature a chicken bit in CPUACTLR_EL1 to enable a hardware workaround. Earlier revisions of the CPU have other errata which require the same workaround in software, so they should be covered anyway.
- `ERRATA_A53_1530924`: This applies errata 1530924 workaround to all revisions of Cortex-A53 CPU.

For Cortex-A55, the following errata build flags are defined :

- `ERRATA_A55_768277`: This applies errata 768277 workaround to Cortex-A55 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- `ERRATA_A55_778703`: This applies errata 778703 workaround to Cortex-A55 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- `ERRATA_A55_798797`: This applies errata 798797 workaround to Cortex-A55 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- `ERRATA_A55_846532`: This applies errata 846532 workaround to Cortex-A55 CPU. This needs to be enabled only for revision <= r0p1 of the CPU.
- `ERRATA_A55_903758`: This applies errata 903758 workaround to Cortex-A55 CPU. This needs to be enabled only for revision <= r0p1 of the CPU.
- `ERRATA_A55_1221012`: This applies errata 1221012 workaround to Cortex-A55 CPU. This needs to be enabled only for revision <= r1p0 of the CPU.
- `ERRATA_A55_1530923`: This applies errata 1530923 workaround to all revisions of Cortex-A55 CPU.

For Cortex-A57, the following errata build flags are defined :

- ERRATA_A57_806969: This applies errata 806969 workaround to Cortex-A57 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- ERRATA_A57_813419: This applies errata 813419 workaround to Cortex-A57 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- ERRATA_A57_813420: This applies errata 813420 workaround to Cortex-A57 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- ERRATA_A57_814670: This applies errata 814670 workaround to Cortex-A57 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- ERRATA_A57_817169: This applies errata 817169 workaround to Cortex-A57 CPU. This needs to be enabled only for revision <= r0p1 of the CPU.
- **ERRATA_A57_817171: This applies erratum 817171 workaround to Cortex-A57**
  CPU. This needs to be enabled for revisions r0p0, r0p1 of the CPU. It is fixed in r1p0.
- ERRATA_A57_826974: This applies errata 826974 workaround to Cortex-A57 CPU. This needs to be enabled only for revision <= r1p1 of the CPU.
- ERRATA_A57_826977: This applies errata 826977 workaround to Cortex-A57 CPU. This needs to be enabled only for revision <= r1p1 of the CPU.
- ERRATA_A57_828024: This applies errata 828024 workaround to Cortex-A57 CPU. This needs to be enabled only for revision <= r1p1 of the CPU.
- ERRATA_A57_829520: This applies errata 829520 workaround to Cortex-A57 CPU. This needs to be enabled only for revision <= r1p2 of the CPU.
- ERRATA_A57_833471: This applies errata 833471 workaround to Cortex-A57 CPU. This needs to be enabled only for revision <= r1p2 of the CPU.
- **ERRATA_A57_836019: This applies erratum 836019 workaround to Cortex-A57**
  CPU. This needs to be enabled for revisions r0p0, r0p1, r1p0, r1p1, r1p2, r1p3 of the CPU. It is still open.
- ERRATA_A57_859972: This applies errata 859972 workaround to Cortex-A57 CPU. This needs to be enabled only for revision <= r1p3 of the CPU.
- ERRATA_A57_1319537: This applies errata 1319537 workaround to all revisions of Cortex-A57 CPU.

For Cortex-A65, the following errata build flags are defined :

- ERRATA_A65_1179935: This applies errata 1179935 workaround to Cortex-A65 CPU. This needs to be enabled only for revision r0p0 of the CPU, and is fixed in r1p0.
- ERRATA_A65_1227419: This applies errata 1227419 workaround to Cortex-A65 CPU. This needs to be enabled only for revision r0p0 and r1p0 of the CPU and is fixed in r1p1.
- ERRATA_A65_1541130: This applies errata 1541130 workaround to r0p0, r1p0, r1p1, r1p2 revisions of the CPU and is still open.

For Cortex-A65AE, the following errata build flags are defined :

- `ERRATA_A65AE_1638571`: This applies errata 1638571 workaround to Cortex-A65AE CPU. This needs to be enabled r0p0, r1p0, r1p1 revisions of the CPU and is still open.

For Cortex-A72, the following errata build flags are defined :

- `ERRATA_A72_859971`: This applies errata 859971 workaround to Cortex-A72 CPU. This needs to be enabled only for revision <= r0p3 of the CPU.
- `ERRATA_A72_1319367`: This applies errata 1319367 workaround to all revisions of Cortex-A72 CPU.

For Cortex-A73, the following errata build flags are defined :

- `ERRATA_A73_852427`: This applies errata 852427 workaround to Cortex-A73 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- `ERRATA_A73_855423`: This applies errata 855423 workaround to Cortex-A73 CPU. This needs to be enabled only for revision <= r0p1 of the CPU.

For Cortex-A75, the following errata build flags are defined :

- `ERRATA_A75_764081`: This applies errata 764081 workaround to Cortex-A75 CPU. This needs to be enabled only for revision r0p0 of the CPU.
- **`ERRATA_A75_790748`: This applies errata 790748 workaround to Cortex-A75**
  CPU. This needs to be enabled only for revision r0p0 of the CPU.

For Cortex-A76, the following errata build flags are defined :

- `ERRATA_A76_1073348`: This applies errata 1073348 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r1p0 of the CPU.
- `ERRATA_A76_1130799`: This applies errata 1130799 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r2p0 of the CPU.
- `ERRATA_A76_1165347`: This applies errata 1165347 workaround to Cortex-A76 CPU. This needs to be enabled for revisions r0p0, r1p0, and r2p0 of the CPU. It is fixed in r3p0.
- `ERRATA_A76_1165522`: This applies errata 1165522 workaround to all revisions of Cortex-A76 CPU. This errata is fixed in r3p0 but due to limitation of errata framework this errata is applied to all revisions of Cortex-A76 CPU.
- `ERRATA_A76_1207823`: This applies errata 1207823 workaround to Cortex-A76 CPU. This needs to be enabled for revisions r0p0, r1p0, and r2p0 of the CPU. It is fixed in r3p0.
- `ERRATA_A76_1220197`: This applies errata 1220197 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r2p0 of the CPU.
- `ERRATA_A76_1257314`: This applies errata 1257314 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.
- `ERRATA_A76_1262606`: This applies errata 1262606 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.
- `ERRATA_A76_1262888`: This applies errata 1262888 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.

- ERRATA_A76_1275112: This applies errata 1275112 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.
- ERRATA_A76_1791580: This applies errata 1791580 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r4p0 of the CPU.
- ERRATA_A76_1868343: This applies errata 1868343 workaround to Cortex-A76 CPU. This needs to be enabled only for revision <= r4p0 of the CPU.
- ERRATA_A76_1946160: This applies errata 1946160 workaround to Cortex-A76 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0, r3p0, r3p1, r4p0, and r4p1 of the CPU. It is still open.
- **ERRATA_A76_2356586: This applies erratum 2356586 workaround to Cortex-A76**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0, r3p0, r3p1, r4p0, r4p1 of the CPU. It is still open.
- ERRATA_A76_2743102: This applies errata 2743102 workaround to Cortex-A76 CPU. This needs to be enabled for all revisions <= r4p1 of the CPU and is still open.
- **ERRATA_A76_3888013: This applies erratum 3888013 workaround to Cortex-A76**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0, r3p0, r3p1, r4p0, r4p1 of the CPU. It is still open.

For Cortex-A76AE, the following errata build flags are defined :

- ERRATA_A76AE_1931427: This applies errata 1931427 workaround to Cortex-A76AE CPU. This needs to be enabled for revision r0p0 and r1p0 of the CPU and it is fixed in r1p1.
- ERRATA_A76AE_1931435: This applies errata 1931435 workaround to Cortex-A76AE CPU. This needs to be enabled for revision r0p0 and r1p0 of the CPU and it is fixed in r1p1.
- ERRATA_A76AE_1969401: This applies errata 1969401 workaround to Cortex-A76AE CPU. This needs to be enabled for revision r0p0 and r1p0 of the CPU and it is fixed in r1p1.
- ERRATA_A76AE_2371140: This applies errata 2371140 workaround to Cortex-A76AE CPU. This needs to be enabled for all revisions <= r1p1 of the CPU and is still open.
- ERRATA_A76AE_2753838: This applies errata 2753838 workaround to Cortex-A76AE CPU. This needs to be enabled for all revisions <= r1p1 of the CPU and is still open.
- **ERRATA_A76AE_3888014: This applies erratum 3888014 workaround to**
  Cortex-A76AE CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 of the CPU. It is still open.

For Cortex-A77, the following errata build flags are defined :

- **ERRATA_A77_1160841: This applies erratum 1160841 workaround to Cortex-A77**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A77_1204882: This applies erratum 1204882 workaround to Cortex-A77**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A77_1220737: This applies erratum 1220737 workaround to Cortex-A77**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A77_1253791: This applies erratum 1253791 workaround to Cortex-A77**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.

- **ERRATA_A77_1273521: This applies erratum 1273521 workaround to Cortex-A77**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- ERRATA_A77_1508412: This applies errata 1508412 workaround to Cortex-A77 CPU. This needs to be enabled only for revision <= r1p0 of the CPU.
- **ERRATA_A77_1515815: This applies erratum 1515815 workaround to Cortex-A77**
  CPU. This needs to be enabled for revisions r0p0, r1p0 of the CPU. It is fixed in r1p1.
- ERRATA_A77_1925769: This applies errata 1925769 workaround to Cortex-A77 CPU. This needs to be enabled only for revision <= r1p1 of the CPU.
- ERRATA_A77_1946167: This applies errata 1946167 workaround to Cortex-A77 CPU. This needs to be enabled only for revision <= r1p1 of the CPU.
- ERRATA_A77_1791578: This applies errata 1791578 workaround to Cortex-A77 CPU. This needs to be enabled for r0p0, r1p0, and r1p1, it is still open.
- ERRATA_A77_2356587: This applies errata 2356587 workaround to Cortex-A77 CPU. This needs to be enabled for r0p0, r1p0, and r1p1, it is still open.
- ERRATA_A77_1800714: This applies errata 1800714 workaround to Cortex-A77 CPU. This needs to be enabled for revisions <= r1p1 of the CPU.
- ERRATA_A77_2743100: This applies errata 2743100 workaround to Cortex-A77 CPU. This needs to be enabled for r0p0, r1p0, and r1p1, it is still open.
- **ERRATA_A77_3888015: This applies erratum 3888015 workaround to Cortex-A77**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 of the CPU. It is still open.

For Cortex-A78, the following errata build flags are defined :

- **ERRATA_A78_1467580: This applies erratum 1467580 workaround to Cortex-A78**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A78_1479939: This applies erratum 1479939 workaround to Cortex-A78**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A78_1492189: This applies erratum 1492189 workaround to Cortex-A78**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A78_1503072: This applies erratum 1503072 workaround to Cortex-A78**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A78_1515634: This applies erratum 1515634 workaround to Cortex-A78**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- ERRATA_A78_1688305: This applies errata 1688305 workaround to Cortex-A78 CPU. This needs to be enabled only for revision r0p0 - r1p0 of the CPU.
- **ERRATA_A78_1827429: This applies erratum 1827429 workaround to Cortex-A78**
  CPU. This needs to be enabled for revisions r0p0, r1p0 of the CPU. It is fixed in r1p1.
- ERRATA_A78_1941498: This applies errata 1941498 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r0p0, r1p0, and r1p1 of the CPU.

- ERRATA_A78_1951500: This applies errata 1951500 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r1p0 and r1p1, r0p0 has the same issue but there is no workaround for that revision.
- ERRATA_A78_1821534: This applies errata 1821534 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r0p0 and r1p0.
- ERRATA_A78_1952683: This applies errata 1952683 workaround to Cortex-A78 CPU. This needs to be enabled for revision r0p0, it is fixed in r1p0.
- ERRATA_A78_2242635: This applies errata 2242635 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r1p0, r1p1, and r1p2. The issue is present in r0p0 but there is no workaround. It is still open.
- ERRATA_A78_2376745: This applies errata 2376745 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, and r1p2, and it is still open.
- ERRATA_A78_2395406: This applies errata 2395406 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, and r1p2, and it is still open.
- **ERRATA_A78_2712571: This applies erratum 2712571 workaround to Cortex-A78**
  CPU, this erratum affects system configurations that do not use an ARM interconnect IP. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2 and it is still open.
- ERRATA_A78_2742426: This applies erratum 2742426 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2 and it is still open.
- ERRATA_A78_2772019: This applies errata 2772019 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, and r1p2, and it is still open.
- ERRATA_A78_2779479: This applies erratum 2779479 workaround to Cortex-A78 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2 and it is still open.
- **ERRATA_A78_3888017: This applies erratum 3888017 workaround to Cortex-A78**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.
- **ERRATA_A78_4302972: This applies erratum 4302972 workaround to Cortex-A78**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.

For Cortex-A78AE, the following errata build flags are defined :

- **ERRATA_A78_AE_1827431: This applies erratum 1827431 workaround to**
  Cortex-A78AE CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A78_AE_1827433: This applies erratum 1827433 workaround to**
  Cortex-A78AE CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A78_AE_1941500**
  [This applies errata 1941500 workaround to] Cortex-A78AE CPU. This needs to be enabled for revisions r0p0 and r0p1. This erratum is still open.
- ERRATA_A78_AE_1951502 : This applies errata 1951502 workaround to Cortex-A78AE CPU. This needs to be enabled for revisions r0p0 and r0p1. This erratum is still open.
- **ERRATA_A78_AE_2242639: This applies erratum 2242639 workaround to**
  Cortex-A78AE CPU. This needs to be enabled for revisions r0p0, r0p1 of the CPU. It is fixed in

r0p2. There is no workaround for revision r0p0.

- ERRATA_A78_AE_2376748 : This applies errata 2376748 workaround to Cortex-A78AE CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. This erratum is still open.
- ERRATA_A78_AE_2395408 : This applies errata 2395408 workaround to Cortex-A78AE CPU. This needs to be enabled for revisions r0p0 and r0p1. This erratum is still open.
- **ERRATA_A78_AE_2466780: This applies erratum 2466780 workaround to**
  Cortex-A78AE CPU. This needs to be enabled for revisions r0p0, r0p1 of the CPU. It is fixed in r0p2.
- ERRATA_A78_AE_2712574 : This applies erratum 2712574 workaround to Cortex-A78AE CPU. This erratum affects system configurations that do not use an ARM interconnect IP. This needs to be enabled for revisions r0p0, r0p1 and r0p2. This erratum is still open.
- **ERRATA_A78_AE_2743229: This applies erratum 2743229 workaround to**
  Cortex-A78AE CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU. It is fixed in r0p3.
- **ERRATA_A78_AE_2779481: This applies erratum 2779481 workaround to**
  Cortex-A78AE CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU. It is fixed in r0p3.
- **ERRATA_A78_AE_3888018: This applies erratum 3888018 workaround to**
  Cortex-A78AE CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 of the CPU. It is still open.
- **ERRATA_A78_AE_4302973: This applies erratum 4302973 workaround to**
  Cortex-A78AE CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 of the CPU. It is still open.

For Cortex-A78C, the following errata build flags are defined :

- ERRATA_A78C_1827430 : This applies errata 1827430 workaround to Cortex-A78C CPU. This needs to be enabled for revision r0p0. The erratum is fixed in r0p1.
- ERRATA_A78C_1827440 : This applies errata 1827440 workaround to Cortex-A78C CPU. This needs to be enabled for revision r0p0. The erratum is fixed in r0p1.
- **ERRATA_A78C_1941499: This applies erratum 1941499 workaround to**
  Cortex-A78C CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A78C_1951501: This applies erratum 1951501 workaround to**
  Cortex-A78C CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- ERRATA_A78C_2242638 : This applies errata 2242638 workaround to Cortex-A78C CPU. This needs to be enabled for revisions r0p1, r0p2 and it is still open.
- **ERRATA_A78C_2376746: This applies erratum 2376746 workaround to**
  Cortex-A78C CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU. It is still open.
- ERRATA_A78C_2376749 : This applies errata 2376749 workaround to Cortex-A78C CPU. This needs to be enabled for revisions r0p1 and r0p2. This erratum is still open.

- **ERRATA_A78C_2395407: This applies erratum 2395407 workaround to**
  Cortex-A78C CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU. It is still open.
- ERRATA_A78C_2395411 : This applies errata 2395411 workaround to Cortex-A78C CPU. This needs to be enabled for revisions r0p1 and r0p2. This erratum is still open.
- **ERRATA_A78C_2478780: This applies erratum 2478780 workaround to**
  Cortex-A78C CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU. It is still open.
- ERRATA_A78C_2683027 : This applies errata 2683027 workaround to Cortex-A78C CPU. This needs to be enabled for revisions r0p1 and r0p2. This erratum is still open.
- ERRATA_A78C_2712575 : This applies erratum 2712575 workaround to Cortex-A78C CPU, this erratum affects system configurations that do not use an ARM interconnect IP. This needs to be enabled for revisions r0p1 and r0p2 and is still open.
- ERRATA_A78C_2743232 : This applies erratum 2743232 workaround to Cortex-A78C CPU. This needs to be enabled for revisions r0p1 and r0p2. This erratum is still open.
- ERRATA_A78C_2772121 : This applies errata 2772121 workaround to Cortex-A78C CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. This erratum is still open.
- **ERRATA_A78C_2779483: This applies erratum 2779483 workaround to**
  Cortex-A78C CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU. It is still open.
- ERRATA_A78C_2779484 : This applies errata 2779484 workaround to Cortex-A78C CPU. This needs to be enabled for revisions r0p1 and r0p2. This erratum is still open.
- **ERRATA_A78C_3888019: This applies erratum 3888019 workaround to**
  Cortex-A78C CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU. It is still open.
- **ERRATA_A78C_4302974: This applies erratum 4302974 workaround to**
  Cortex-A78C CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU. It is still open.

For Cortex-X1 CPU, the following errata build flags are defined:

- **ERRATA_X1_1467580: This applies erratum 1467580 workaround to Cortex-X1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_X1_1492189: This applies erratum 1492189 workaround to Cortex-X1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_X1_1503072: This applies erratum 1503072 workaround to Cortex-X1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_X1_1515634: This applies erratum 1515634 workaround to Cortex-X1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_X1_1479939: This applies erratum 1479939 workaround to Cortex-X1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.

- **ERRATA_X1_1821534**
  [This applies errata 1821534 workaround to Cortex-X1] CPU. This needs to be enabled only for revision <= r1p0 of the CPU.
- **ERRATA_X1_1688305**
  [This applies errata 1688305 workaround to Cortex-X1] CPU. This needs to be enabled only for revision <= r1p0 of the CPU.
- **ERRATA_X1_1827429**
  [This applies errata 1827429 workaround to Cortex-X1] CPU. This needs to be enabled only for revision <= r1p0 of the CPU.
- **ERRATA_X1_1941498: This applies erratum 1941498 workaround to Cortex-X1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 of the CPU. It is fixed in r1p2.
- **ERRATA_X1_1951500: This applies erratum 1951500 workaround to Cortex-X1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 of the CPU. It is fixed in r1p2.
- **ERRATA_X1_1952683: This applies erratum 1952683 workaround to Cortex-X1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_X1_2376745: This applies erratum 2376745 workaround to Cortex-X1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.
- **ERRATA_X1_2395406: This applies erratum 2395406 workaround to Cortex-X1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.
- **ERRATA_X1_2742426: This applies erratum 2742426 workaround to Cortex-X1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.
- **ERRATA_X1_2779479: This applies erratum 2779479 workaround to Cortex-X1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.
- **ERRATA_X1_3888017: This applies erratum 3888017 workaround to Cortex-X1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.
- **ERRATA_X1_4302972: This applies erratum 4302972 workaround to Cortex-X1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.

For Neoverse N1, the following errata build flags are defined :

- **ERRATA_N1_925373: This applies erratum 925373 workaround to Neoverse N1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- `ERRATA_N1_1073348`: This applies errata 1073348 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision r0p0 and r1p0 of the CPU.
- `ERRATA_N1_1130799`: This applies errata 1130799 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r2p0 of the CPU.
- `ERRATA_N1_1165347`: This applies errata 1165347 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r2p0 of the CPU.
- `ERRATA_N1_1207823`: This applies errata 1207823 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r2p0 of the CPU.

- ERRATA_N1_1220197: This applies errata 1220197 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r2p0 of the CPU.
- ERRATA_N1_1257314: This applies errata 1257314 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.
- ERRATA_N1_1262606: This applies errata 1262606 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.
- ERRATA_N1_1262888: This applies errata 1262888 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.
- ERRATA_N1_1275112: This applies errata 1275112 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.
- ERRATA_N1_1315703: This applies errata 1315703 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r3p0 of the CPU.
- ERRATA_N1_1542419: This applies errata 1542419 workaround to Neoverse-N1 CPU. This needs to be enabled only for revisions r3p0 - r4p0 of the CPU.
- **ERRATA_N1_1791580: This applies erratum 1791580 workaround to Neoverse N1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0, r3p0, r3p1, r4p0 of the CPU. It is fixed in r4p1.
- ERRATA_N1_1868343: This applies errata 1868343 workaround to Neoverse-N1 CPU. This needs to be enabled only for revision <= r4p0 of the CPU.
- ERRATA_N1_1946160: This applies errata 1946160 workaround to Neoverse-N1 CPU. This needs to be enabled for revisions r3p0, r3p1, r4p0, and r4p1, for revisions r0p0, r1p0, and r2p0 there is no workaround.
- ERRATA_N1_2743102: This applies errata 2743102 workaround to Neoverse-N1 CPU. This needs to be enabled for all revisions <= r4p1 of the CPU and is still open.
- ERRATA_N1_3324349: This applies errata 3324349 workaround to Neoverse-N1 CPU. This needs to be enabled for all revisions <= r4p1 of the CPU and is still open.
- **ERRATA_N1_3888013: This applies erratum 3888013 workaround to Neoverse N1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0, r3p0, r3p1, r4p0, r4p1 of the CPU. It is still open.

For Neoverse V1, the following errata build flags are defined :

- **ERRATA_V1_1542436: This applies erratum 1542436 workaround to Neoverse V1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_V1_1618634: This applies erratum 1618634 workaround to Neoverse V1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- ERRATA_V1_1618635: This applies errata 1618635 workaround to Neoverse-V1 CPU. This needs to be enabled for revision r0p0 of the CPU, it is fixed in r1p0.
- **ERRATA_V1_1618636: This applies erratum 1618636 workaround to Neoverse V1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.

- **`ERRATA_V1_1619807`: This applies erratum 1619807 workaround to Neoverse V1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **`ERRATA_V1_1654562`: This applies erratum 1654562 workaround to Neoverse V1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **`ERRATA_V1_1674403`: This applies erratum 1674403 workaround to Neoverse V1**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- `ERRATA_V1_1774420`: This applies errata 1774420 workaround to Neoverse-V1 CPU. This needs to be enabled only for revisions r0p0 and r1p0, it is fixed in r1p1.
- `ERRATA_V1_1791573`: This applies errata 1791573 workaround to Neoverse-V1 CPU. This needs to be enabled only for revisions r0p0 and r1p0, it is fixed in r1p1.
- `ERRATA_V1_1852267`: This applies errata 1852267 workaround to Neoverse-V1 CPU. This needs to be enabled only for revisions r0p0 and r1p0, it is fixed in r1p1.
- `ERRATA_V1_1925756`: This applies errata 1925756 workaround to Neoverse-V1 CPU. This needs to be enabled for r0p0, r1p0, and r1p1, it is still open.
- `ERRATA_V1_1940577`: This applies errata 1940577 workaround to Neoverse-V1 CPU. This needs to be enabled only for revision r1p0 and r1p1 of the CPU.
- `ERRATA_V1_1966096`: This applies errata 1966096 workaround to Neoverse-V1 CPU. This needs to be enabled for revisions r1p0 and r1p1 of the CPU, the issue is present in r0p0 as well but there is no workaround for that revision. It is still open.
- `ERRATA_V1_2139242`: This applies errata 2139242 workaround to Neoverse-V1 CPU. This needs to be enabled for revisions r0p0, r1p0, and r1p1 of the CPU. It is still open.
- `ERRATA_V1_2216392`: This applies errata 2216392 workaround to Neoverse-V1 CPU. This needs to be enabled for revisions r1p0 and r1p1 of the CPU, the issue is present in r0p0 as well but there is no workaround for that revision. It is still open.
- `ERRATA_V1_2294912`: This applies errata 2294912 workaround to Neoverse-V1 CPU. This needs to be enabled for revisions r0p0, r1p0, and r1p1 and r1p2 of the CPU.
- `ERRATA_V1_2348377`: This applies errata 2348377 workaroud to Neoverse-V1 CPU. This needs to be enabled for revisions r0p0, r1p0 and r1p1 of the CPU. It has been fixed in r1p2.
- `ERRATA_V1_2372203`: This applies errata 2372203 workaround to Neoverse-V1 CPU. This needs to be enabled for revisions r0p0, r1p0 and r1p1 of the CPU. It is still open.
- **`ERRATA_V1_2701953`: This applies erratum 2701953 workaround to Neoverse-V1**
  CPU, this erratum affects system configurations that do not use an ARM interconnect IP. This needs to be enabled for revisions r0p0, r1p0 and r1p1. It has been fixed in r1p2.
- `ERRATA_V1_2743093`: This applies errata 2743093 workaround to Neoverse-V1 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2 of the CPU. It is still open.
- `ERRATA_V1_2743233`: This applies erratum 2743233 workaround to Neoverse-V1 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, and r1p2 of the CPU. It is still open.
- `ERRATA_V1_2779461`: This applies erratum 2779461 workaround to Neoverse-V1 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.

- **ERRATA_V1_3888016: This applies erratum 3888016 workaround to Neoverse V1**
  CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 of the CPU. It is still open.

For Neoverse V2, the following errata build flags are defined :

- **ERRATA_V2_2394277: This applies erratum 2394277 workaround to Neoverse V2**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- ERRATA_V2_2618597: This applies errata 2618597 workaround to Neoverse-V2 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_V2_2662553: This applies errata 2662553 workaround to Neoverse-V2 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_V2_2719103: This applies errata 2719103 workaround to Neoverse-V2 CPU, this affects system configurations that do not use and ARM interconnect IP. This needs to be enabled for revisions r0p0 and r0p1. It has been fixed in r0p2.
- ERRATA_V2_2719105: This applies errata 2719105 workaround to Neoverse-V2 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_V2_2743011: This applies errata 2743011 workaround to Neoverse-V2 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_V2_2779510: This applies errata 2779510 workaround to Neoverse-V2 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_V2_2801372: This applies errata 2801372 workaround to Neoverse-V2 CPU, this affects all configurations. This needs to be enabled for revisions r0p0 and r0p1. It has been fixed in r0p2.
- ERRATA_V2_3442699: This applies errata 3442699 workaround to Neoverse-V2 CPU. This needs to be enabled for revision r0p0 - r0p2 and is still open.
- ERRATA_V2_3701771: This applies errata 3701771 workaround to Neoverse-V2 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2 and is still open.
- ERRATA_V2_3841324: This applies errata 3841324 workaround to Neoverse-V2 CPU. This needs to be enabled only for revisions r0p0 and r0p1 of the CPU. It is fixed in r0p2.
- ERRATA_V2_3888126: This applies errata 3888126 workaround to Neoverse-V2 CPU. This needs to be enabled only for revisions r0p0, r0p1, r0p2 of the CPU. It is still open.
- ERRATA_V2_4302968: This applies errata 4302968 workaround to Neoverse-V2 CPU. This needs to be enabled only for revisions r0p0, r0p1, r0p2 of the CPU. It is still open.

For Neoverse V3, the following errata build flags are defined :

- ERRATA_V3_2970647: This applies errata 2970647 workaround to Neoverse-V3 CPU. This needs to be enabled for revision r0p0. It is fixed in r0p1.
- ERRATA_V3_3312417: This applies errata 3312417 workaround to Neoverse-V3 CPU. This needs to be enabled for revisions r0p0 and r0p1 of the CPU and is fixed in r0p2.
- ERRATA_V3_3696307: This applies errata 3696307 workaround to Neoverse-V3 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.

- `ERRATA_V3_3701767`: This applies errata 3701767 workaround to Neoverse-V3 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 of the CPU and is still open.
- `ERRATA_V3_3734562`: This applies errata 3734562 workaround to Neoverse-V3 CPU. This needs to be enabled for revisions r0p0 and r0p1 of the CPU and is fixed in r0p2.
- `ERRATA_V3_3782181`: This applies errata 3782181 workaround to Neoverse-V3 CPU. This needs to be enabled for revision r0p1 of the CPU and is fixed in r0p2.
- `ERRATA_V3_3864536`: This applies errata 3864536 workaround to Neoverse-V3 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2 of the CPU and is still open.
- `ERRATA_V3_3878291`: This applies errata 3878291 workaround to Neoverse-V3 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2 of the CPU and is still open.

For Cortex-A710, the following errata build flags are defined :

- **`ERRATA_A710_1785648`: This applies erratum 1785648 workaround to**
  Cortex-A710 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **`ERRATA_A710_1793423`: This applies erratum 1793423 workaround to**
  Cortex-A710 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **`ERRATA_A710_1847092`: This applies erratum 1847092 workaround to**
  Cortex-A710 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **`ERRATA_A710_1887102`: This applies erratum 1887102 workaround to**
  Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0 of the CPU. It is fixed in r2p0.
- `ERRATA_A710_1901946`: This applies errata 1901946 workaround to Cortex-A710 CPU. This needs to be enabled only for revision r1p0. It has been fixed in r2p0.
- `ERRATA_A710_1916945`: This applies errata 1916945 workaround to Cortex-A710 CPU. This needs to be enabled only for revisions r0p0 and r1p0. It has been fixed in r2p0.
- `ERRATA_A710_1917258`: This applies errata 1917258 workaround to Cortex-A710 CPU. This needs to be enabled only for revisions r0p0 and r1p0. It has been fixed in r2p0.
- `ERRATA_A710_1927200`: This applies errata 1927200 workaround to Cortex-A710 CPU. This needs to be enabled only for revisions r0p0 and r1p0. It has been fixed in r2p0.
- `ERRATA_A710_1987031`: This applies errata 1987031 workaround to Cortex-A710 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU. It is still open.
- `ERRATA_A710_2081180`: This applies errata 2081180 workaround to Cortex-A710 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU. It is still open.
- `ERRATA_A710_2055002`: This applies errata 2055002 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r1p0, r2p0 of the CPU and is still open.
- `ERRATA_A710_2017096`: This applies errata 2017096 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0 and r2p0 of the CPU and is still open.
- `ERRATA_A710_2083908`: This applies errata 2083908 workaround to Cortex-A710 CPU. This needs to be enabled for revision r2p0 of the CPU and is still open.

- ERRATA_A710_2267065: This applies errata 2267065 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0 and r2p0 of the CPU and is fixed in r2p1.
- ERRATA_A710_2136059: This applies errata 2136059 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0 and r2p0 of the CPU and is fixed in r2p1.
- ERRATA_A710_2147715: This applies errata 2147715 workaround to Cortex-A710 CPU. This needs to be enabled for revision r2p0 of the CPU and is fixed in r2p1.
- ERRATA_A710_2216384: This applies errata 2216384 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0 and r2p0 of the CPU and is fixed in r2p1.
- ERRATA_A710_2282622: This applies errata 2282622 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- **ERRATA_A710_2291219: This applies errata 2291219 workaround to**
  Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0 and r2p0 of the CPU and is fixed in r2p1.
- ERRATA_A710_2008768: This applies errata 2008768 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0 and r2p0 of the CPU and is fixed in r2p1.
- ERRATA_A710_2371105: This applies errata 2371105 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0 and r2p0 of the CPU and is fixed in r2p1.
- ERRATA_A710_2701952: This applies erratum 2701952 workaround to Cortex-A710 CPU, and applies to system configurations that do not use and ARM interconnect IP. This needs to be enabled for r0p0, r1p0, r2p0 and r2p1 and is still open.
- ERRATA_A710_2742423: This applies errata 2742423 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- ERRATA_A710_2768515: This applies errata 2768515 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- ERRATA_A710_2778471: This applies errata 2778471 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- ERRATA_A710_3324338: This applies errata 3324338 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- ERRATA_A710_3701772: This applies errata 3701772 workaround to Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0, r2p1 of the CPU and is still open.
- **ERRATA_A710_3888122: This applies erratum 3888122 workaround to**
  Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0, r2p1 of the CPU. It is still open.
- **ERRATA_A710_4302969: This applies erratum 4302969 workaround to**
  Cortex-A710 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0, r2p1 of the CPU. It is still open.

For Neoverse N2, the following errata build flags are defined :

- ERRATA_N2_2002655: This applies errata 2002655 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.

- ERRATA_N2_2009478: This applies errata 2009478 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2067956: This applies errata 2067956 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2025414: This applies errata 2025414 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2138953: This applies errata 2138953 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 and r0p3 of the Neoverse N2 cpu and is still open.
- ERRATA_N2_2138956: This applies errata 2138956 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2138958: This applies errata 2138958 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2189731: This applies errata 2189731 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2242400: This applies errata 2242400 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2242415: This applies errata 2242415 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2280757: This applies errata 2280757 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU and is fixed in r0p1.
- ERRATA_N2_2326639: This applies errata 2326639 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU, it is fixed in r0p1.
- ERRATA_N2_2340933: This applies errata 2340933 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU, it is fixed in r0p1.
- ERRATA_N2_2346952: This applies errata 2346952 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2 of the CPU, it is fixed in r0p3.
- ERRATA_N2_2376738: This applies errata 2376738 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0, r0p1, r0p2, r0p3 and is still open.
- ERRATA_N2_2388450: This applies errata 2388450 workaround to Neoverse-N2 CPU. This needs to be enabled for revision r0p0 of the CPU, it is fixed in r0p1.
- ERRATA_N2_2743014: This applies errata 2743014 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. It is fixed in r0p3.
- ERRATA_N2_2743089: This applies errata 2743089 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. It is fixed in r0p3.
- **ERRATA_N2_2728475: This applies erratum 2728475 workaround to Neoverse-N2**
  CPU, this erratum affects system configurations that do not use and ARM interconnect IP. This needs to be enabled for revisions r0p0, r0p1 and r0p2. It is fixed in r0p3.
- ERRATA_N2_2779511: This applies errata 2779511 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. It is fixed in r0p3.

- `ERRATA_N2_3324339`: This applies errata 3324339 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 and is still open.
- `ERRATA_N2_3701773`: This applies errata 3701773 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 and is still open.
- `ERRATA_N2_3888123`: This applies errata 3888123 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 and is still open.
- `ERRATA_N2_4302970`: This applies errata 4302970 workaround to Neoverse-N2 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 and is still open.

For Neoverse N3, the following errata build flags are defined :

- `ERRATA_N3_3456111`: This applies errata 3456111 workaround to Neoverse-N3 CPU. This needs to be enabled for revisions r0p0 and r0p1 and is still open.
- `ERRATA_N3_3699563`: This applies errata 3699563 workaround to Neoverse-N3 CPU. This needs to be enabled for revisions r0p0 and is still open.

For Cortex-X2, the following errata build flags are defined :

- **`ERRATA_X2_1785648`: This applies erratum 1785648 workaround to Cortex-X2**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **`ERRATA_X2_1793423`: This applies erratum 1793423 workaround to Cortex-X2**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **`ERRATA_X2_1863568`: This applies erratum 1863568 workaround to Cortex-X2**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **`ERRATA_X2_1887102`: This applies erratum 1887102 workaround to Cortex-X2**
  CPU. This needs to be enabled for revisions r0p0, r1p0 of the CPU. It is fixed in r2p0.
- **`ERRATA_X2_1887413`: This applies erratum 1887413 workaround to Cortex-X2**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- `ERRATA_X2_1901946`: This applies errata 1901946 workaround to Cortex-X2 CPU. This needs to be enabled only for r1p0, it is fixed in r2p0.
- `ERRATA_X2_1916945`: This applies errata 1916945 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0 and r1p0 of the CPU, it is fixed in r2p0.
- `ERRATA_X2_1917258`: This applies errata 1917258 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0 and r1p0 of the CPU, it is fixed in r2p0.
- `ERRATA_X2_1927200`: This applies errata 1927200 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0 and r1p0 of the CPU, it is fixed in r2p0.
- `ERRATA_X2_1934260`: This applies errata 1934260 workaround to Cortex-X2 CPU. This needs to be enabled only for revision r1p0 of the CPU, it is fixed in r2p0.
- `ERRATA_X2_2002765`: This applies errata 2002765 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU, it is fixed in r2p1.
- `ERRATA_X2_2017096`: This applies errata 2017096 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU, it is fixed in r2p1.

- `ERRATA_X2_2081180`: This applies errata 2081180 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU, it is fixed in r2p1.
- `ERRATA_X2_2083908`: This applies errata 2083908 workaround to Cortex-X2 CPU. This needs to be enabled only for revision r2p0 of the CPU, it is fixed in r2p1.
- `ERRATA_X2_2136059`: This applies errata 2136059 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU, it is fixed in r2p1.
- `ERRATA_X2_2147715`: This applies errata 2147715 workaround to Cortex-X2 CPU. This needs to be enabled only for revision r2p0 of the CPU, it is fixed in r2p1.
- `ERRATA_X2_2216384`: This applies errata 2216384 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU, it is fixed in r2p1.
- `ERRATA_X2_2267065`: This applies errata 2267065 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU, it is fixed in r2p1.
- `ERRATA_X2_2282622`: This applies errata 2282622 workaround to Cortex-X2 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- `ERRATA_X2_2291219`: This applies errata 2291219 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU, it is fixed in r2p1.
- `ERRATA_X2_2371105`: This applies errata 2371105 workaround to Cortex-X2 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r2p0 of the CPU, it is fixed in r2p1.
- **`ERRATA_X2_2701952`: This applies errata 2701952 workaround to Cortex-X2**
  CPU and affects system configurations that do not use an Arm interconnect IP. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 and is still open.
- `ERRATA_X2_2742423`: This applies errata 2742423 workaround to Cortex-X2 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- `ERRATA_X2_2768515`: This applies errata 2768515 workaround to Cortex-X2 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- `ERRATA_X2_2778471`: This applies errata 2778471 workaround to Cortex-X2 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- `ERRATA_X2_3324338`: This applies errata 3324338 workaround to Cortex-X2 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- `ERRATA_X2_3701772`: This applies errata 3701772 workaround to Cortex-X2 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- `ERRATA_X2_3888122`: This applies errata 3888122 workaround to Cortex-X2 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.
- `ERRATA_X2_4302969`: This applies errata 4302969 workaround to Cortex-X2 CPU. This needs to be enabled for revisions r0p0, r1p0, r2p0 and r2p1 of the CPU and is still open.

For Cortex-X3, the following errata build flags are defined :

- **`ERRATA_X3_2138930`: This applies erratum 2138930 workaround to Cortex-X3**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.

- **ERRATA_X3_2147714: This applies erratum 2147714 workaround to Cortex-X3**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_X3_2184829: This applies erratum 2184829 workaround to Cortex-X3**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_X3_2214778: This applies erratum 2214778 workaround to Cortex-X3**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- `ERRATA_X3_2266875`: This applies errata 2266875 workaround to the Cortex-X3 CPU. This needs to be enabled only for revisions r0p0 and r1p0 of the CPU, it is fixed in r1p1.
- `ERRATA_X3_2302506`: This applies errata 2302506 workaround to the Cortex-X3 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r1p1, it is fixed in r1p2.
- `ERRATA_X3_2313909`: This applies errata 2313909 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0 and r1p0 of the CPU, it is fixed in r1p1.
- `ERRATA_X3_2372204`: This applies errata 2372204 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0 and r1p0 of the CPU, it is fixed in r1p1.
- `ERRATA_X3_2615812`: This applies errata 2615812 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r1p1 of the CPU, it is fixed in r1p2.
- `ERRATA_X3_2641945`: This applies errata 2641945 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0 and r1p0 of the CPU. It is fixed in r1p1.
- `ERRATA_X3_2701951`: This applies erratum 2701951 workaround to Cortex-X3 CPU and affects system configurations that do not use an ARM interconnect IP. This needs to be applied to revisions r0p0, r1p0 and r1p1. It is fixed in r1p2.
- `ERRATA_X3_2742421`: This applies errata 2742421 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r1p1. It is fixed in r1p2.
- `ERRATA_X3_2743088`: This applies errata 2743088 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r1p1. It is fixed in r1p2.
- `ERRATA_X3_2779509`: This applies errata 2779509 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r1p1 of the CPU. It is fixed in r1p2.
- `ERRATA_X3_3213672`: This applies errata 3213672 workaround to Cortex-X3 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2 of the CPU. It is still open.
- `ERRATA_X3_3692984`: This applies errata 3692984 workaround to Cortex-X3 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2 of the CPU. It is still open.
- `ERRATA_X3_3701769`: This applies errata 3701769 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0, r1p0, r1p1 and r1p2 of the CPU and it is still open.
- `ERRATA_X3_3827463`: This applies errata 3827463 workaround to Cortex-X3 CPU. This needs to be enabled only for revisions r0p0, r1p0 and r1p1 of the CPU. It is fixed in r1p2.
- `ERRATA_X3_3888125`: This applies errata 3888125 workaround to Cortex-X3 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2 of the CPU. It is still open.
- `ERRATA_X3_4302966`: This applies errata 4302966 workaround to Cortex-X3 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2 of the CPU. It is still open.

For Cortex-X4, the following errata build flags are defined :

- **`ERRATA_X4_2302507`: This applies erratum 2302507 workaround to Cortex-X4**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **`ERRATA_X4_2620954`: This applies erratum 2620954 workaround to Cortex-X4**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **`ERRATA_X4_2631888`: This applies erratum 2631888 workaround to Cortex-X4**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **`ERRATA_X4_2646977`: This applies erratum 2646977 workaround to Cortex-X4**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- `ERRATA_X4_2701112`: This applies erratum 2701112 workaround to Cortex-X4 CPU and affects system configurations that do not use an Arm interconnect IP. This needs to be enabled for revisions r0p0 and is fixed in r0p1. The workaround for this erratum is not implemented in EL3, but the flag can be enabled/disabled at the platform level. The flag is used when the errata ABI feature is enabled and can assist the Kernel in the process of mitigation of the erratum.
- `ERRATA_X4_2726228`: This applies erratum 2726228 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- `ERRATA_X4_2740089`: This applies errata 2740089 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- `ERRATA_X4_2763018`: This applies errata 2763018 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- `ERRATA_X4_2816013`: This applies errata 2816013 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- `ERRATA_X4_2897503`: This applies errata 2897503 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- `ERRATA_X4_2923985`: This applies errata 2923985 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- `ERRATA_X4_2957258`: This applies errata 2957258 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- `ERRATA_X4_3076789`: This applies errata 3076789 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- `ERRATA_X4_3133195`: This applies errata 3133195 workaround to Cortex-X4 CPU. This needs to be enabled for revision r0p2. It is fixed in r0p3.
- `ERRATA_X4_3701758`: This applies errata 3701758 workaround to Cortex-X4 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2 and r0p3. It is still open.
- **`ERRATA_X4_3841338`: This applies erratum 3841338 workaround to Cortex-X4**
  CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- `ERRATA_X4_3887999`: This applies errata 3887999 workaround to Cortex-X4 CPU. This needs to be enabled for revision r0p0, r0p1, r0p2 and r0p3. It is still open.

For Cortex-X925, the following errata build flags are defined :

- ERRATA_X925_2921199: This applies errata 2921199 workaround to Cortex-X925 CPU. This needs to be enabled for revision r0p0. It is fixed in r0p1.
- ERRATA_X925_2922378: This applies errata 2922378 workaround to Cortex-X925 CPU. This needs to be enabled for revision r0p0. It is fixed in r0p1.
- ERRATA_X925_2933290: This applies errata 2933290 workaround to Cortex-X925 CPU. This needs to be enabled for revision r0p0. It is fixed in r0p1.
- ERRATA_X925_2963999: This applies errata 2963999 workaround to Cortex-X925 CPU. This needs to be enabled for revision r0p0. It is fixed in r0p1.
- ERRATA_X925_3324334: This applies errata 3324334 workaround to Cortex-X925 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_X925_3692980: This applies errata 3692980 workaround to Cortex-X925 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_X925_3730893: This applies errata 3730893 workaround to Cortex-X925 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_X925_3865185: This applies errata 3865185 workaround to Cortex-X925 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_X925_3701747: This applies errata 3701747 workaround to Cortex-X925 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is still open.

For Cortex-A510, the following errata build flags are defined :

- **ERRATA_A510_1910738: This applies erratum 1910738 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A510_1937669: This applies erratum 1937669 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A510_1942494: This applies erratum 1942494 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A510_1952872: This applies erratum 1952872 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A510_1966377: This applies erratum 1966377 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A510_1975068: This applies erratum 1975068 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A510_1976290: This applies erratum 1976290 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- **ERRATA_A510_2002389: This applies erratum 2002389 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revisions r0p0, r0p1 of the CPU. It is fixed in r0p2.
- ERRATA_A510_2008766: This applies errata 2008766 workaround to Cortex-A510 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3, r1p0, r1p1, r1p2 and r1p3. It is still open.

- **ERRATA_A510_2027318: This applies erratum 2027318 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revisions r0p0, r0p1 of the CPU. It is fixed in r0p2.
- **ERRATA_A510_2028010: This applies erratum 2028010 workaround to**
  Cortex-A510 CPU. This needs to be enabled for revisions r0p0, r0p1 of the CPU. It is fixed in r0p2.
- ERRATA_A510_2288014: This applies errata 2288014 workaround to Cortex-A510 CPU. This needs to be enabled only for revisions r0p0, r0p1, r0p2, r0p3 and r1p0, it is fixed in r1p1.
- ERRATA_A510_2042739: This applies errata 2042739 workaround to Cortex-A510 CPU. This needs to be enabled only for revisions r0p0, r0p1 and r0p2, it is fixed in r0p3.
- ERRATA_A510_2041909: This applies errata 2041909 workaround to Cortex-A510 CPU. This needs to be enabled only for revision r0p2 and is fixed in r0p3. The issue is also present in r0p0 and r0p1 but there is no workaround for those revisions.
- ERRATA_A510_2080326: This applies errata 2080326 workaround to Cortex-A510 CPU. This needs to be enabled only for revision r0p2 and is fixed in r0p3. This issue is also present in r0p0 and r0p1 but there is no workaround for those revisions.
- ERRATA_A510_2169012: This applies errata 2169012 workaround to Cortex-A510 CPU. This needs to be enabled only for revisions r0p0, r0p1, r0p2, r0p3 and r1p0, it is fixed in r1p1.
- ERRATA_A510_2218134: This applies errata 2218134 workaround to Cortex-A510 CPU. This needs to be enabled only for revision r1p0 and is fixed in r1p1.
- ERRATA_A510_2250311: This applies errata 2250311 workaround to Cortex-A510 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 and r1p0, it is fixed in r1p1. This workaround disables MPMM even if ENABLE_MPMM=1.
- ERRATA_A510_2218950: This applies errata 2218950 workaround to Cortex-A510 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 and r1p0, it is fixed in r1p1.
- ERRATA_A510_2172148: This applies errata 2172148 workaround to Cortex-A510 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3 and r1p0, it is fixed in r1p1.
- ERRATA_A510_2347730: This applies errata 2347730 workaround to Cortex-A510 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3, r1p0 and r1p1. It is fixed in r1p2.
- ERRATA_A510_2371937: This applies errata 2371937 workaround to Cortex-A510 CPU. This needs to applied for revisions r0p0, r0p1, r0p2, r0p3, r1p0, r1p1, and is fixed in r1p2.
- ERRATA_A510_2420992: This applies errata 2420992 workaround to Cortex-A510 CPU. This needs to applied for revisions r1p0 and r1p1, and is fixed in r1p2.
- ERRATA_A510_2666669: This applies errata 2666669 workaround to Cortex-A510 CPU. This needs to applied for revisions r0p0, r0p1, r0p2, r0p3, r1p0, r1p1. It is fixed in r1p2.
- ERRATA_A510_2684597: This applies erratum 2684597 workaround to Cortex-A510 CPU. This needs to be applied to revision r0p0, r0p1, r0p2, r0p3, r1p0, r1p1 and r1p2. It is fixed in r1p3.
- ERRATA_A510_2971420: This applies erratum 2971420 workaround to Cortex-A510 CPU. This needs to be applied to revisions r0p1, r0p2, r0p3, r1p0, r1p1, r1p2 and r1p3 and is still open.

- ERRATA_A510_3672349: This applies erratum 3672349 workaround to Cortex-A510 CPU. This needs to be applied to revisions r0p1, r0p2, r0p3, r1p0, r1p1, r1p2 and r1p3 and is still open.
- ERRATA_A510_3704847: This applies erratum 3704847 workaround to Cortex-A510 CPU. This needs to be applied to revisions r0p1, r0p2, r0p3, r1p0, r1p1, r1p2 and r1p3 and is still open.

For Cortex-A520, the following errata build flags are defined :

- ERRATA_A520_2630792: This applies errata 2630792 workaround to Cortex-A520 CPU. This needs to applied for revisions r0p0, r0p1 of the CPU and is still open.
- **ERRATA_A520_2677201: This applies erratum 2677201 workaround to**
  Cortex-A520 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r0p1.
- ERRATA_A520_2858100: This applies errata 2858100 workaround to Cortex-A520 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is still open.
- ERRATA_A520_2938996: This applies errata 2938996 workaround to Cortex-A520 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- **ERRATA_A520_3631357: This applies erratum 3631357 workaround to**
  Cortex-A520 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3, r0p4 of the CPU. It is still open.
- **ERRATA_A520_3685825: This applies erratum 3685825 workaround to**
  Cortex-A520 CPU. This needs to be enabled for revisions r0p0, r0p1, r0p2, r0p3, r0p4 of the CPU. It is still open.

For Cortex-A715, the following errata build flags are defined :

- **ERRATA_A715_2238661: This applies erratum 2238661 workaround to**
  Cortex-A715 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A715_2239006: This applies erratum 2239006 workaround to**
  Cortex-A715 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A715_2275754: This applies erratum 2275754 workaround to**
  Cortex-A715 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A715_2284544: This applies erratum 2284544 workaround to**
  Cortex-A715 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A715_2285473: This applies erratum 2285473 workaround to**
  Cortex-A715 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- **ERRATA_A715_2292761: This applies erratum 2292761 workaround to**
  Cortex-A715 CPU. This needs to be enabled for revision r0p0 of the CPU. It is fixed in r1p0.
- ERRATA_A715_2331818: This applies errata 2331818 workaround to Cortex-A715 CPU. This needs to be enabled for revisions r0p0 and r1p0. It is fixed in r1p1.
- **ERRATA_A715_2344187: This applies errata 2344187 workaround to**
  Cortex-A715 CPU. This needs to be enabled for revisions r0p0 and r1p0. It is fixed in r1p1.
- ERRATA_A715_2376701: This applies errata 2376701 workaround to Cortex-A715 CPU. This needs to be enabled for revisions r0p0 and r1p0. It is fixed in r1p1.

- ERRATA_A715_2409570: This applies errata 2409570 workaround to Cortex-A715 CPU. This needs to be enabled only for revision r1p0. It is fixed in r1p1. This errata also applies to r0p0 but that revision has a different workaround, and since r0p0 is not used in production hardware it is not implemented.
- ERRATA_A715_2413290: This applies errata 2413290 workaround to Cortex-A715 CPU. This needs to be enabled only for revision r1p0 and when SPE(Statistical profiling extension)=True. The errata is fixed in r1p1.
- ERRATA_A715_2420947: This applies errata 2420947 workaround to Cortex-A715 CPU. This needs to be enabled only for revision r1p0. It is fixed in r1p1.
- ERRATA_A715_2429384: This applies errata 2429384 workaround to Cortex-A715 CPU. This needs to be enabled for revision r1p0. There is no workaround for revision r0p0. It is fixed in r1p1.
- ERRATA_A715_2561034: This applies errata 2561034 workaround to Cortex-A715 CPU. This needs to be enabled only for revision r1p0. It is fixed in r1p1.
- ERRATA_A715_2728106: This applies errata 2728106 workaround to Cortex-A715 CPU. This needs to be enabled for revisions r0p0, r1p0 and r1p1. It is fixed in r1p2.
- ERRATA_A715_2804830: This applies errata 2804830 workaround to Cortex-A715 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1 and r1p2. It is fixed in r1p3.
- ERRATA_A715_3456084: This applies errata 3456084 workaround to Cortex-A715 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 and r1p3. It is still open.
- ERRATA_A715_3699560: This applies errata 3699560 workaround to Cortex-A715 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p2 and r1p3. It is still open.
- ERRATA_A715_3711916: This applies errata 3711916 workaround to Cortex-A715 CPU. This needs to be enabled for revisions r0p0, r1p0, r1p1, r1p2 and r1p3. It is still open.

For Cortex-A720, the following errata build flags are defined :

- ERRATA_A720_2729604: This applies errata 2729604 workaround to Cortex-A720 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_A720_2792132: This applies errata 2792132 workaround to Cortex-A720 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_A720_2844092: This applies errata 2844092 workaround to Cortex-A720 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_A720_2926083: This applies errata 2926083 workaround to Cortex-A720 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_A720_2940794: This applies errata 2940794 workaround to Cortex-A720 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_A720_3456091: This applies errata 3456091 workaround to Cortex-A720 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. It is still open.
- ERRATA_A720_3699561: This applies errata 3699561 workaround to Cortex-A720 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. It is still open.

- ERRATA_A720_3711910: This applies errata 3711910 workaround to Cortex-A720 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. It is still open.

For Cortex-A720_AE, the following errata build flags are defined :

- ERRATA_A720_AE_3456103: This applies errata 3456103 workaround to Cortex-A720_AE CPU. This needs to be enabled for revisions r0p0 and r0p1. It is still open.
- ERRATA_A720_AE_3699562: This applies errata 3699562 workaround to Cortex-A720_AE CPU. This needs to be enabled for revisions r0p0. It is still open.

For Cortex-A725, the following errata build flags are defined :

- ERRATA_A725_2874943: This applies errata 2874943 workaround to Cortex-A725 CPU. This needs to be enabled for revisions r0p0 when FEAT_SPE is enabled. It is fixed in r0p1.
- ERRATA_A725_2936490: This applies errata 2936490 workaround to Cortex-A725 CPU. This needs to be enabled for revisions r0p0. It is fixed in r0p1.
- ERRATA_A725_3456106: This applies errata 3456106 workaround to Cortex-A725 CPU. This needs to be enabled for revisions r0p0, r0p1 and r0p2. It is still open.
- ERRATA_A725_3699564: This applies errata 3699564 workaround to Cortex-A725 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.
- ERRATA_A725_3711914: This applies errata 3711914 workaround to Cortex-A725 CPU. This needs to be enabled for revisions r0p0 and r0p1. It is fixed in r0p2.

For C1-Ultra, the following errata build flags are defined :

- ERRATA_C1ULTRA_3324333: This applies erratum 3324333 workaround to C1-Ultra CPU. This needs to be enabled for revision r0p0 and is fixed in r1p0.
- ERRATA_C1ULTRA_3502731: This applies erratum 3502731 workaround to C1-Ultra CPU. This needs to be enabled for revision r0p0 and is fixed in r1p0.
- ERRATA_C1ULTRA_3658374: This applies erratum 3658374 workaround to C1-Ultra CPU. This needs to be enabled for revisions r0p0 and r1p0 and is still open.
- ERRATA_C1ULTRA_3684152: This applies erratum 3684152 workaround to C1-Ultra CPU. This needs to be enabled for revision r0p0 and is fixed in r1p0.
- ERRATA_C1ULTRA_3705939: This applies erratum 3705939 workaround to C1-Ultra CPU. This needs to be enabled for revisions r0p0 and r1p0 and is still open.
- ERRATA_C1ULTRA_3815514: This applies erratum 3815514 workaround to C1-Ultra CPU. This needs to be enabled for revisions r0p0 and r1p0 and is still open.
- ERRATA_C1ULTRA_3865171: This applies erratum 3865171 workaround to C1-Ultra CPU. This needs to be enabled for revisions r0p0 and r1p0 and is still open.
- ERRATA_C1ULTRA_3926381: This applies erratum 3926381 workaround to C1-Ultra CPU. This needs to be enabled for revision r1p0 and is still open.
- ERRATA_C1ULTRA_4102704: This applies erratum 4102704 workaround to C1-Ultra CPU. This needs to be enabled for revisions r0p0 and r1p0 and is still open.

For C1-Premium, the following errata build flags are defined :

- `ERRATA_C1PREMIUM_3324333`: This applies errata 3324333 workaround to C1-Premium CPU. This needs to be enabled for revision r0p0, and is fixed in r1p0.
- `ERRATA_C1PREMIUM_3502731`: This applies errata 3502731 workaround to C1-Premium CPU. This needs to be enabled for revision r0p0, and is fixed in r1p0.
- `ERRATA_C1PREMIUM_3684152`: This applies errata 3684152 workaround to C1-Premium CPU. This needs to be enabled for revision r0p0, and is fixed in r1p0.
- `ERRATA_C1PREMIUM_3705939`: This applies errata 3705939 workaround to C1-Premium CPU. This needs to be enabled for revisions r0p0, r1p0 and is still open.
- `ERRATA_C1PREMIUM_3815514`: This applies errata 3815514 workaround to C1-Premium CPU. This needs to be enabled for revisions r0p0, r1p0 and is still open.
- `ERRATA_C1PREMIUM_3865171`: This applies errata 3865171 workaround to C1-Premium CPU. This needs to be enabled for revisions r0p0, r1p0 and is still open.
- `ERRATA_C1PREMIUM_3926381`: This applies errata 3926381 workaround to C1-Premium CPU. This needs to be enabled for revision r1p0 and is still open.
- `ERRATA_C1PREMIUM_4102704`: This applies errata 4102704 workaround to C1-Premium CPU. This needs to be enabled for revisions r0p0, r1p0 and is still open.

For C1-Pro, the following errata build flags are defined :

- `ERRATA_C1PRO_3619847`: This applies errata 3619847 workaround to C1-Pro CPU. This needs to be enabled only for revision r0p0 and is fixed in r1p0.
- `ERRATA_C1PRO_3338470`: This applies errata 3338470 workaround to C1-Pro CPU. This needs to be enabled only for revision r0p0 and is fixed in r1p0.
- `ERRATA_C1PRO_3362007`: This applies errata 3362007 workaround to C1-Pro CPU. This needs to be enabled only for revision r0p0 and is fixed in r1p0.
- **`ERRATA_C1PRO_3684268`: This applies erratum 3684268 workaround to C1-Pro**
  CPU. This needs to be enabled for revisions r0p0, r1p0 of the CPU. It is fixed in r1p1.
- `ERRATA_C1PRO_3686597`: This applies errata 3686597 workaround to C1-Pro CPU. This needs to be enabled for revisions r0p0 and r1p0 of the CPU, it is fixed in r1p1.
- `ERRATA_C1PRO_3694158`: This applies errata 3694158 workaround to C1-Pro CPU. This needs to be enabled for revisions r0p0, r1p0 and r1p1 and is fixed in r1p2.
- `ERRATA_C1PRO_3706576`: This applies errata 3706576 workaround to C1-Pro CPU. This needs to be enabled for revisions r0p0 and r1p0 of the CPU, it is fixed in r1p1.
- `ERRATA_C1PRO_3300099`: This applies errata 3300099 workaround to C1-Pro CPU. This needs to be enabled for revisions r0p0 and r1p0 of the CPU, it is fixed in r1p1.

For C1-Nano, the following errata build flags are defined :

- `ERRATA_C1NANO_3392149`: This applies errata 3392149 workaround to C1-Nano CPU. This needs to be enabled for revision r0p0 and is fixed in r0p1.
- `ERRATA_C1NANO_3419531`: This applies errata 3419531 workaround to C1-Nano CPU. This needs to be enabled for revision r0p0 and is fixed in r0p1.

- ERRATA_C1NANO_3437202: This applies errata 3437202 workaround to C1-Nano CPU. This needs to be enabled for revision r0p0 and is fixed in r0p1.
- ERRATA_C1NANO_3516455: This applies errata 3516455 workaround to C1-Nano CPU. This needs to be enabled for revision r0p0 and is fixed in r0p1.
- ERRATA_C1NANO_3616450: This applies errata 3616450 workaround to C1-Nano CPU. This needs to be enabled for revision r0p0 and is fixed in r0p1.
- ERRATA_C1NANO_3630925: This applies errata 3630925 workaround to C1-Nano CPU. This needs to be enabled for revision r0p0 and is fixed in r0p1.
- ERRATA_C1NANO_3754876: This applies errata 3754876 workaround to C1-Nano CPU. This needs to be enabled for revisions r0p0 and r0p1, and is fixed in r0p2.

5.3.3 DSU Errata Workarounds

Similar to CPU errata, TF-A also implements workarounds for DSU (DynamIQ Shared Unit) errata. The DSU errata details can be found in the respective Arm documentation:

- Arm DSU Software Developers Errata Notice.

Each erratum is identified by an ID, as defined in the DSU errata notice document. Thus, the build flags which enable/disable the errata workarounds have the format ERRATA_DSU_<ID>. The implementation and application logic of DSU errata workarounds are similar to *CPU errata workarounds*.

For DSU errata, the following build flags are defined:

- ERRATA_DSU_798953: This applies errata 798953 workaround for the affected DSU configurations. This errata applies only for those DSUs that revision is r0p0 (on r0p1 it is fixed). However, please note that this workaround results in increased DSU power consumption on idle.
- ERRATA_DSU_936184: This applies errata 936184 workaround for the affected DSU configurations. This errata applies only for those DSUs that contain the ACP interface **and** the DSU revision is older than r2p0 (on r2p0 it is fixed). However, please note that this workaround results in increased DSU power consumption on idle.
- ERRATA_DSU_2313941: This applies errata 2313941 workaround for the affected DSU configurations. This errata applies for those DSUs with revisions r0p0, r1p0, r2p0, r2p1, r3p0, r3p1 and is still open. However, please note that this workaround results in increased DSU power consumption on idle.
- ERRATA_DSU_2900952: This applies errata 2900952 workaround for the affected DSU-120 configurations. This erratum applies to some r2p0 implementations and is fixed in r2p1. The affected r2p0 implementations are determined by reading the IMP_CLUSTERREVIDR_EL1[1] register bit and making sure it's clear.

5.3.4 CPU Specific optimizations

This section describes some of the optimizations allowed by the CPU micro architecture that can be enabled by the platform as desired.

- `SKIP_A57_L1_FLUSH_PWR_DWN`: This flag enables an optimization in the Cortex-A57 cluster power down sequence by not flushing the Level 1 data cache. The L1 data cache and the L2 unified cache are inclusive. A flush of the L2 by set/way flushes any dirty lines from the L1 as well. This is a known safe deviation from the Cortex-A57 TRM defined power down sequence. Each Cortex-A57 based platform must make its own decision on whether to use the optimization.
- `A53_DISABLE_NON_TEMPORAL_HINT`: This flag disables the cache non-temporal hint. The LDNP/STNP instructions as implemented on Cortex-A53 do not behave in a way most programmers expect, and will most probably result in a significant speed degradation to any code that employs them. The Armv8-A architecture (see Arm DDI 0487A.h, section D3.4.3) allows cores to ignore the non-temporal hint and treat LDNP/STNP as LDP/STP instead. Enabling this flag enforces this behaviour. This needs to be enabled only for revisions <= r0p3 of the CPU and is enabled by default.
- `A57_DISABLE_NON_TEMPORAL_HINT`: This flag has the same behaviour as `A53_DISABLE_NON_TEMPORAL_HINT` but for Cortex-A57. This needs to be enabled only for revisions <= r1p2 of the CPU and is enabled by default, as recommended in section "4.7 Non-Temporal Loads/Stores" of the Cortex-A57 Software Optimization Guide.
- **''A57_ENABLE_NON_CACHEABLE_LOAD_FWD": This flag enables non-cacheable**
  streaming enhancement feature for Cortex-A57 CPUs. Platforms can set this bit only if their memory system meets the requirement that cache line fill requests from the Cortex-A57 processor are atomic. Each Cortex-A57 based platform must make its own decision on whether to use the optimization. This flag is disabled by default.
- `NEOVERSE_Nx_EXTERNAL_LLC`: This flag indicates that an external last level cache(LLC) is present in the system, and that the DataSource field on the master CHI interface indicates when data is returned from the LLC. This is used to control how the LL_CACHE* PMU events count. Default value is 0 (Disabled).
- `NEOVERSE_Vx_EXTERNAL_LLC`: This flag has the same behaviour as `NEOVERSE_Nx_EXTERNAL_LLC` but for Neoverse-V2. This is disabled by default. Default value is 0 (Disabled).
- `NEOVERSE_N2_PREFETCHER_DISABLE`: This flag disables the region prefetcher on the Neoverse N2 core. This is used during performance analysis to get clean and repeatable measurements of the cache by preventing speculative data fetches from interfering with benchmark results. Default value is 0 (Disabled).

5.3.5 GIC Errata Workarounds

- `GIC600_ERRATA_WA_2384374`: This flag applies part 2 of errata 2384374 workaround for the affected GIC600 and GIC600-AE implementations. It applies to implementations of GIC600 and GIC600-AE with revisions less than or equal to r1p6 and r0p2 respectively. If the platform sets GICV3_SUPPORT_GIC600, then this flag is enabled; otherwise, it is 0 (Disabled).

5.4 Firmware Design

Trusted Firmware-A (TF-A) implements a subset of the Trusted Board Boot Requirements (TBBR) Platform Design Document (PDD) for Arm reference platforms.

The TBB sequence starts when the platform is powered on and runs up to the stage where it hands-off control to firmware running in the normal world in DRAM. This is the cold boot path.

TF-A also implements the PSCI as a runtime service. PSCI is the interface from normal world software to firmware implementing power management use-cases (for example, secondary CPU boot, hotplug and idle). Normal world software can access TF-A runtime services via the Arm SMC (Secure Monitor Call) instruction. The SMC instruction must be used as mandated by the SMC Calling Convention (SMCCC).

TF-A implements a framework for configuring and managing interrupts generated in either security state. The details of the interrupt management framework and its design can be found in *Interrupt Management Framework*.

TF-A also implements a library for setting up and managing the translation tables. The details of this library can be found in *Translation (XLAT) Tables Library*.

TF-A can be built to support either AArch64 or AArch32 execution state.

Note: The descriptions in this chapter are for the Arm TrustZone architecture. For changes to the firmware design for the Arm Confidential Compute Architecture (Arm CCA) please refer to the chapter *Realm Management Extension (RME)*.

5.4.1 Cold boot

The cold boot path starts when the platform is physically turned on. If `COLD_BOOT_SINGLE_CPU=0`, one of the CPUs released from reset is chosen as the primary CPU, and the remaining CPUs are considered secondary CPUs. The primary CPU is chosen through platform-specific means. The cold boot path is mainly executed by the primary CPU, other than essential CPU initialization executed by all CPUs. The secondary CPUs are kept in a safe platform-specific state until the primary CPU has performed enough initialization to boot them.

Refer to the *CPU Reset* for more information on the effect of the `COLD_BOOT_SINGLE_CPU` platform build option.

The cold boot path in this implementation of TF-A depends on the execution state. For AArch64, it is divided into five steps (in order of execution):

- Boot Loader stage 1 (BL1) *AP Trusted ROM*
- Boot Loader stage 2 (BL2) *Trusted Boot Firmware*
- Boot Loader stage 3-1 (BL31) *EL3 Runtime Software*
- Boot Loader stage 3-2 (BL32) *Secure-EL1 Payload* (optional)
- Boot Loader stage 3-3 (BL33) *Non-trusted Firmware*

For AArch32, it is divided into four steps (in order of execution):

- Boot Loader stage 1 (BL1) *AP Trusted ROM*
- Boot Loader stage 2 (BL2) *Trusted Boot Firmware*
- Boot Loader stage 3-2 (BL32) *EL3 Runtime Software*
- Boot Loader stage 3-3 (BL33) *Non-trusted Firmware*

Arm development platforms (Fixed Virtual Platforms (FVPs) and Juno) implement a combination of the following types of memory regions. Each bootloader stage uses one or more of these memory regions.

- Regions accessible from both non-secure and secure states. For example, non-trusted SRAM, ROM and DRAM.
- Regions accessible from only the secure state. For example, trusted SRAM and ROM. The FVPs also implement the trusted DRAM which is statically configured. Additionally, the Base FVPs and Juno development platform configure the TrustZone Controller (TZC) to create a region in the DRAM which is accessible only from the secure state.

The sections below provide the following details:

- dynamic configuration of Boot Loader stages
- initialization and execution of the first three stages during cold boot
- specification of the EL3 Runtime Software (BL31 for AArch64 and BL32 for AArch32) entrypoint requirements for use by alternative Trusted Boot Firmware in place of the provided BL1 and BL2

Dynamic Configuration during cold boot

Each of the Boot Loader stages may be dynamically configured if required by the platform. The Boot Loader stage may optionally specify a firmware configuration file and/or hardware configuration file as listed below:

- FW_CONFIG - The firmware configuration file. Holds properties shared across all BLx images. An example is the "dtb-registry" node, which contains the information about the other device tree configurations (load-address, size, image_id).
- HW_CONFIG - The hardware configuration file. Can be shared by all Boot Loader stages and also by the Normal World Rich OS.
- TB_FW_CONFIG - Trusted Boot Firmware configuration file. Shared between BL1 and BL2.

- SOC_FW_CONFIG - SoC Firmware configuration file. Used by BL31.
- TOS_FW_CONFIG - Trusted OS Firmware configuration file. Used by Trusted OS (BL32).
- NT_FW_CONFIG - Non Trusted Firmware configuration file. Used by Non-trusted firmware (BL33).

The Arm development platforms use the Flattened Device Tree format for the dynamic configuration files.

Each Boot Loader stage can pass up to 4 arguments via registers to the next stage. BL2 passes the list of the next images to execute to the *EL3 Runtime Software* (BL31 for AArch64 and BL32 for AArch32) via *arg0*. All the other arguments are platform defined. The Arm development platforms use the following convention:

- BL1 passes the address of a meminfo_t structure to BL2 via `arg1`. This structure contains the memory layout available to BL2.
- When dynamic configuration files are present, the firmware configuration for the next Boot Loader stage is populated in the first available argument and the generic hardware configuration is passed the next available argument. For example,
  - FW_CONFIG is loaded by BL1, then its address is passed in `arg0` to BL2.
  - TB_FW_CONFIG address is retrieved by BL2 from FW_CONFIG device tree.
  - If HW_CONFIG is loaded by BL1, then its address is passed in `arg2` to BL2. Note, `arg1` is already used for meminfo_t.
  - If SOC_FW_CONFIG is loaded by BL2, then its address is passed in `arg1` to BL31. Note, `arg0` is used to pass the list of executable images.
  - Similarly, if HW_CONFIG is loaded by BL1 or BL2, then its address is passed in `arg2` to BL31.
  - For other BL3x images, if the firmware configuration file is loaded by BL2, then its address is passed in `arg0` and if HW_CONFIG is loaded then its address is passed in `arg1`.
  - In case SPMC_AT_EL3 is enabled, populate the BL32 image base, size and max limit in the entry point information, since there is no platform function to retrieve these in generic code. We choose `arg2`, `arg3` and `arg4` since the generic code uses `arg1` for stashing the SP manifest size. The SPMC setup uses these arguments to update SP manifest with actual SP's base address and it size.
  - In case of the Arm FVP platform, FW_CONFIG address passed in `arg1` to BL31/SP_MIN, and the SOC_FW_CONFIG and HW_CONFIG details are retrieved from FW_CONFIG device tree.

BL1

This stage begins execution from the platform's reset vector at EL3. The reset address is platform dependent but it is usually located in a Trusted ROM area. The BL1 data section is copied to trusted SRAM at runtime.

On the Arm development platforms, BL1 code starts execution from the reset vector defined by the constant `BL1_RO_BASE`. The BL1 data section is copied to the top of trusted SRAM as defined by the constant `BL1_RW_BASE`.

The functionality implemented by this stage is as follows.

Determination of boot path

Whenever a CPU is released from reset, BL1 needs to distinguish between a warm boot and a cold boot. This is done using platform-specific mechanisms (see the `plat_get_my_entrypoint()` function in the *Porting Guide*). In the case of a warm boot, a CPU is expected to continue execution from a separate entrypoint. In the case of a cold boot, the secondary CPUs are placed in a safe platform-specific state (see the `plat_secondary_cold_boot_setup()` function in the *Porting Guide*) while the primary CPU executes the remaining cold boot path as described in the following sections.

This step only applies when `PROGRAMMABLE_RESET_ADDRESS=0`. Refer to the *CPU Reset* for more information on the effect of the `PROGRAMMABLE_RESET_ADDRESS` platform build option.

Architectural initialization

BL1 performs minimal architectural initialization as follows.

- Exception vectors

  BL1 sets up simple exception vectors for both synchronous and asynchronous exceptions. The default behavior upon receiving an exception is to populate a status code in the general purpose register `X0/R0` and call the `plat_report_exception()` function (see the *Porting Guide*). The status code is one of:

  For AArch64:

```
0x0 : Synchronous exception from Current EL with SP_EL0
0x1 : IRQ exception from Current EL with SP_EL0
0x2 : FIQ exception from Current EL with SP_EL0
0x3 : System Error exception from Current EL with SP_EL0
0x4 : Synchronous exception from Current EL with SP_ELx
0x5 : IRQ exception from Current EL with SP_ELx
0x6 : FIQ exception from Current EL with SP_ELx
0x7 : System Error exception from Current EL with SP_ELx
0x8 : Synchronous exception from Lower EL using aarch64
0x9 : IRQ exception from Lower EL using aarch64
0xa : FIQ exception from Lower EL using aarch64
0xb : System Error exception from Lower EL using aarch64
0xc : Synchronous exception from Lower EL using aarch32
0xd : IRQ exception from Lower EL using aarch32
0xe : FIQ exception from Lower EL using aarch32
0xf : System Error exception from Lower EL using aarch32
```

  For AArch32:

```
0x10 : User mode
0x11 : FIQ mode
0x12 : IRQ mode
0x13 : SVC mode
0x16 : Monitor mode
0x17 : Abort mode
0x1a : Hypervisor mode
```

(continues on next page)

(continued from previous page)

```
0x1b : Undefined mode
0x1f : System mode
```

The `plat_report_exception()` implementation on the Arm FVP port programs the Versatile Express System LED register in the following format to indicate the occurrence of an unexpected exception:

```
SYS_LED[0]   - Security state (Secure=0/Non-Secure=1)
SYS_LED[2:1] - Exception Level (EL3=0x3, EL2=0x2, EL1=0x1, EL0=0x0)
               For AArch32 it is always 0x0
SYS_LED[7:3] - Exception Class (Sync/Async & origin). This is the value
               of the status code
```

A write to the LED register reflects in the System LEDs (S6LED0..7) in the CLCD window of the FVP.

BL1 does not expect to receive any exceptions other than the SMC exception. For the latter, BL1 installs a simple stub. The stub expects to receive a limited set of SMC types (determined by their function IDs in the general purpose register `X0/R0`):

– `BL1_SMC_RUN_IMAGE`: This SMC is raised by BL2 to make BL1 pass control to EL3 Runtime Software.

– All SMCs listed in section "BL1 SMC Interface" in the *Firmware Update (FWU)* Design Guide are supported for AArch64 only. These SMCs are currently not supported when BL1 is built for AArch32.

Any other SMC leads to an assertion failure.

- CPU initialization

  BL1 calls the `reset_handler` macro/function which in turn calls the CPU specific reset handler function (see the section: "CPU specific operations framework").

Platform initialization

On Arm platforms, BL1 performs the following platform initializations:

- Enable the Trusted Watchdog.
- Initialize the console.
- Configure the Interconnect to enable hardware coherency.
- Enable the MMU and map the memory it needs to access.
- Configure any required platform storage to load the next bootloader image (BL2).
- If the BL1 dynamic configuration file, `TB_FW_CONFIG`, is available, then load it to the platform defined address and make it available to BL2 via `arg0`.
- Configure the system timer and program the *CNTFRQ_EL0* for use by NS-BL1U and NS-BL2U firmware update images.

Firmware Update detection and execution

After performing platform setup, BL1 common code calls `bl1_plat_get_next_image_id()` to determine if *Firmware Update (FWU)* is required or to proceed with the normal boot process. If the platform code returns `BL2_IMAGE_ID` then the normal boot sequence is executed as described in the next section, else BL1 assumes that *Firmware Update (FWU)* is required and execution passes to the first image in the *Firmware Update (FWU)* process. In either case, BL1 retrieves a descriptor of the next image by calling `bl1_plat_get_image_desc()`. The image descriptor contains an `entry_point_info_t` structure, which BL1 uses to initialize the execution state of the next image.

BL2 image load and execution

In the normal boot flow, BL1 execution continues as follows:

1. BL1 prints the following string from the primary CPU to indicate successful execution of the BL1 stage:

   ```
   "Booting Trusted Firmware"
   ```

2. BL1 loads a BL2 raw binary image from platform storage, at a platform-specific base address. Prior to the load, BL1 invokes `bl1_plat_handle_pre_image_load()` which allows the platform to update or use the image information. If the BL2 image file is not present or if there is not enough free trusted SRAM the following error message is printed:

   ```
   "Failed to load BL2 firmware."
   ```

3. BL1 invokes `bl1_plat_handle_post_image_load()` which again is intended for platforms to take further action after image load. This function must populate the necessary arguments for BL2, which may also include the memory layout. Further description of the memory layout can be found later in this document.

4. BL1 passes control to the BL2 image at Secure EL1 (for AArch64) or at Secure SVC mode (for AArch32), starting from its load address.

BL2

BL1 loads and passes control to BL2 at Secure-EL1 (for AArch64) or at Secure SVC mode (for AArch32) . BL2 is linked against and loaded at a platform-specific base address (more information can be found later in this document). The functionality implemented by BL2 is as follows.

Architectural initialization

For AArch64, BL2 performs the minimal architectural initialization required for subsequent stages of TF-A and normal world software. EL1 and EL0 are given access to Floating Point and Advanced SIMD registers by setting the `CPACR.FPEN` bits.

For AArch32, the minimal architectural initialization required for subsequent stages of TF-A and normal world software is taken care of in BL1 as both BL1 and BL2 execute at PL1.

Platform initialization

On Arm platforms, BL2 performs the following platform initializations:

- Initialize the console.
- Configure any required platform storage to allow loading further bootloader images.
- Enable the MMU and map the memory it needs to access.
- Perform platform security setup to allow access to controlled components.
- Reserve some memory for passing information to the next bootloader image EL3 Runtime Software and populate it.
- Define the extents of memory available for loading each subsequent bootloader image.
- If BL1 has passed TB_FW_CONFIG dynamic configuration file in `arg0`, then parse it.

Image loading in BL2

BL2 generic code loads the images based on the list of loadable images provided by the platform. BL2 passes the list of executable images provided by the platform to the next handover BL image.

The list of loadable images provided by the platform may also contain dynamic configuration files. The files are loaded and can be parsed as needed in the `bl2_plat_handle_post_image_load()` function. These configuration files can be passed to next Boot Loader stages as arguments by updating the corresponding entrypoint information in this function.

SCP_BL2 (System Control Processor Firmware) image load

Some systems have a separate System Control Processor (SCP) for power, clock, reset and system control. BL2 loads the optional SCP_BL2 image from platform storage into a platform-specific region of secure memory. The subsequent handling of SCP_BL2 is platform specific. For example, on the Juno Arm development platform port the image is transferred into SCP's internal memory using the Boot Over MHU (BOM) protocol after being loaded in the trusted SRAM memory. The SCP executes SCP_BL2 and signals to the Application Processor (AP) for BL2 execution to continue.

EL3 Runtime Software image load

BL2 loads the EL3 Runtime Software image from platform storage into a platform- specific address in trusted SRAM. If there is not enough memory to load the image or image is missing it leads to an assertion failure.

AArch64 BL32 (Secure-EL1 Payload) image load

BL2 loads the optional BL32 image from platform storage into a platform- specific region of secure memory. The image executes in the secure world. BL2 relies on BL31 to pass control to the BL32 image, if present. Hence, BL2 populates a platform-specific area of memory with the entrypoint/load-address of the BL32 image. The value of the Saved Processor Status Register (`SPSR`) for entry into BL32 is not determined by BL2, it is initialized by the Secure-EL1 Payload Dispatcher (see later) within BL31, which is responsible for managing interaction with BL32. This information is passed to BL31.

BL33 (Non-trusted Firmware) image load

BL2 loads the BL33 image (e.g. UEFI or other test or boot software) from platform storage into non-secure memory as defined by the platform.

BL2 relies on EL3 Runtime Software to pass control to BL33 once secure state initialization is complete. Hence, BL2 populates a platform-specific area of memory with the entrypoint and Saved Program Status Register (`SPSR`) of the normal world software image. The entrypoint is the load address of the BL33 image. The `SPSR` is determined as specified in Section 5.13 of the PSCI. This information is passed to the EL3 Runtime Software.

AArch64 BL31 (EL3 Runtime Software) execution

BL2 execution continues as follows:

1. BL2 passes control back to BL1 by raising an SMC, providing BL1 with the BL31 entrypoint. The exception is handled by the SMC exception handler installed by BL1.
2. BL1 turns off the MMU and flushes the caches. It clears the `SCTLR_EL3.M/I/C` bits, flushes the data cache to the point of coherency and invalidates the TLBs.
3. BL1 passes control to BL31 at the specified entrypoint at EL3.

Running BL2 at EL3 execution level

Some platforms have a non-TF-A Boot ROM that expects the next boot stage to execute at EL3. On these platforms, TF-A BL1 is a waste of memory as its only purpose is to ensure TF-A BL2 is entered at S-EL1. To avoid this waste, a special mode enables BL2 to execute at EL3, which allows a non-TF-A Boot ROM to load and jump directly to BL2. This mode is selected when the build flag RESET_TO_BL2 is enabled. The main differences in this mode are:

1. BL2 includes the reset code and the mailbox mechanism to differentiate cold boot and warm boot. It runs at EL3 doing the arch initialization required for EL3.
2. BL2 does not receive the meminfo information from BL1 anymore. This information can be passed by the Boot ROM or be internal to the BL2 image.
3. Since BL2 executes at EL3, BL2 jumps directly to the next image, instead of invoking the RUN_IMAGE SMC call.

We assume 3 different types of BootROM support on the platform:

1. The Boot ROM always jumps to the same address, for both cold and warm boot. In this case, we will need to keep a resident part of BL2 whose memory cannot be reclaimed by any other image. The linker script defines the symbols __TEXT_RESIDENT_START__ and __TEXT_RESIDENT_END__ that allows the platform to configure correctly the memory map.
2. The platform has some mechanism to indicate the jump address to the Boot ROM. Platform code can then program the jump address with psci_warmboot_entrypoint during cold boot.
3. The platform has some mechanism to program the reset address using the PROGRAMMABLE_RESET_ADDRESS feature. Platform code can then program the reset address with psci_warmboot_entrypoint during cold boot, bypassing the boot ROM for warm boot.

In the last 2 cases, no part of BL2 needs to remain resident at runtime. In the first 2 cases, we expect the Boot ROM to be able to differentiate between warm and cold boot, to avoid loading BL2 again during warm boot.

This functionality can be tested with FVP loading the image directly in memory and changing the address where the system jumps at reset. For example:

> -C cluster0.cpu0.RVBAR=0x4022000 –data cluster0.cpu0=bl2.bin@0x4022000

With this configuration, FVP is like a platform of the first case, where the Boot ROM jumps always to the same address. For simplification, BL32 is loaded in DRAM in this case, to avoid other images reclaiming BL2 memory.

AArch64 BL31

The image for this stage is loaded by BL2 and BL1 passes control to BL31 at EL3. BL31 executes solely in trusted SRAM. BL31 is linked against and loaded at a platform-specific base address (more information can be found later in this document). The functionality implemented by BL31 is as follows.

Architectural initialization

Currently, BL31 performs a similar architectural initialization to BL1 as far as system register settings are concerned. Since BL1 code resides in ROM, architectural initialization in BL31 allows override of any previous initialization done by BL1.

BL31 initializes the per-CPU data framework, which provides a cache of frequently accessed per-CPU data optimised for fast, concurrent manipulation on different CPUs. This buffer includes pointers to per-CPU contexts, crash buffer, CPU reset and power down operations, PSCI data, platform data and so on.

It then replaces the exception vectors populated by BL1 with its own. BL31 exception vectors implement more elaborate support for handling SMCs since this is the only mechanism to access the runtime services implemented by BL31 (PSCI for example). BL31 checks each SMC for validity as specified by the SMC Calling Convention before passing control to the required SMC handler routine.

BL31 programs the `CNTFRQ_EL0` register with the clock frequency of the system counter, which is provided by the platform.

Platform initialization

BL31 performs detailed platform initialization, which enables normal world software to function correctly.

On Arm platforms, this consists of the following:

- Initialize the console.
- Configure the Interconnect to enable hardware coherency.
- Enable the MMU and map the memory it needs to access.
- Initialize the generic interrupt controller.
- Initialize the power controller device.
- Detect the system topology.

Runtime services initialization

BL31 is responsible for initializing the runtime services. One of them is PSCI.

As part of the PSCI initializations, BL31 detects the system topology. It also initializes the data structures that implement the state machine used to track the state of power domain nodes. The state can be one of `OFF`, `RUN` or `RETENTION`. All secondary CPUs are initially in the `OFF` state. The cluster that the primary CPU belongs to is `ON`; any other cluster is `OFF`. It also initializes the locks that protect them. BL31 accesses the state of a CPU or cluster immediately after reset and before the data cache is enabled in the warm boot path. It is not currently possible to use 'exclusive' based spinlocks, therefore BL31 uses locks based on Lamport's Bakery algorithm instead.

The runtime service framework and its initialization is described in more detail in the "EL3 runtime services framework" section below.

Details about the status of the PSCI implementation are provided in the "Power State Coordination Interface" section below.

AArch64 BL32 (Secure-EL1 Payload) image initialization

If a BL32 image is present then there must be a matching Secure-EL1 Payload Dispatcher (SPD) service (see later for details). During initialization that service must register a function to carry out initialization of BL32 once the runtime services are fully initialized. BL31 invokes such a registered function to initialize BL32 before running BL33. This initialization is not necessary for AArch32 SPs.

Details on BL32 initialization and the SPD's role are described in the *Secure-EL1 Payloads and Dispatchers* section below.

BL33 (Non-trusted Firmware) execution

EL3 Runtime Software initializes the EL2 or EL1 processor context for normal- world cold boot, ensuring that no secure state information finds its way into the non-secure execution state. EL3 Runtime Software uses the entrypoint information provided by BL2 to jump to the Non-trusted firmware image (BL33) at the highest available Exception Level (EL2 if available, otherwise EL1).

Using alternative Trusted Boot Firmware in place of BL1 & BL2 (AArch64 only)

Some platforms have existing implementations of Trusted Boot Firmware that would like to use TF-A BL31 for the EL3 Runtime Software. To enable this firmware architecture it is important to provide a fully documented and stable interface between the Trusted Boot Firmware and BL31.

Future changes to the BL31 interface will be done in a backwards compatible way, and this enables these firmware components to be independently enhanced/ updated to develop and exploit new functionality.

Required CPU state when calling `bl31_entrypoint()` during cold boot

This function must only be called by the primary CPU.

On entry to this function the calling primary CPU must be executing in AArch64 EL3, little-endian data access, and all interrupt sources masked:

```
PSTATE.EL = 3
PSTATE.RW = 1
PSTATE.DAIF = 0xf
SCTLR_EL3.EE = 0
```

X0 and X1 can be used to pass information from the Trusted Boot Firmware to the platform code in BL31:

```
X0 : Reserved for common TF-A information
X1 : Platform specific information
```

BL31 zero-init sections (e.g. `.bss`) should not contain valid data on entry, these will be zero filled prior to invoking platform setup code.

Use of the X0 and X1 parameters

The parameters are platform specific and passed from `bl31_entrypoint()` to `bl31_early_platform_setup()`. The value of these parameters is never directly used by the common BL31 code.

Legacy platforms may still pass a `bl31_params` structure in `X0`, while TL adopted platforms may use Transfer List; both are platform-specific.

MMU, Data caches & Coherency

BL31 does not depend on the enabled state of the MMU, data caches or interconnect coherency on entry to `bl31_entrypoint()`. If these are disabled on entry, these should be enabled during `bl31_plat_arch_setup()`.

Data structures used in the BL31 cold boot interface

In the cold boot flow, `entry_point_info` is used to represent the execution state of an image; that is, the state of general purpose registers, PC, and SPSR.

There are two variants of this structure, for AArch64:

```
typedef struct entry_point_info {
     param_header_t h;
     uintptr_t pc;
     uint32_t spsr;

     aapcs64_params_t args;
}
```

and, AArch32:

```
typedef struct entry_point_info {
   param_header_t h;
   uintptr_t pc;
   uint32_t spsr;

   uintptr_t lr_svc;
   aapcs32_params_t args;
} entry_point_info_t;
```

These structures are designed to support compatibility and independent evolution of the structures and the firmware images. For example, a version of BL31 that can interpret the BL3x image information from different versions of BL2, a platform that uses an extended entry_point_info structure to convey additional register information to BL31, or a ELF image loader that can convey more details about the firmware images.

To support these scenarios the structures are versioned and sized, which enables BL31 to detect which information is present and respond appropriately. The `param_header` is defined to capture this information:

```
typedef struct param_header {
    uint8_t type;       /* type of the structure */
    uint8_t version;    /* version of this structure */
    uint16_t size;      /* size of this structure in bytes */
    uint32_t attr;      /* attributes */
} param_header_t;
```

In *entry_point_info*, Bits 0 and 5 of `attr` field are used to encode the security state; in other words, whether the image is to be executed in Secure, Non-Secure, or Realm mode.

Other structures using this format are `image_info` and `bl31_params`. The code that allocates and populates these structures must set the header fields appropriately, the `SET_PARAM_HEAD()` macro is defined to simplify this action.

Required CPU state for BL31 Warm boot initialization

When requesting a CPU power-on, or suspending a running CPU, TF-A provides the platform power management code with a Warm boot initialization entry-point, to be invoked by the CPU immediately after the reset handler. On entry to the Warm boot initialization function the calling CPU must be in AArch64 EL3, little-endian data access and all interrupt sources masked:

```
PSTATE.EL = 3
PSTATE.RW = 1
PSTATE.DAIF = 0xf
SCTLR_EL3.EE = 0
```

The PSCI implementation will initialize the processor state and ensure that the platform power management code is then invoked as required to initialize all necessary system, cluster and CPU resources.

AArch32 EL3 Runtime Software entrypoint interface

To enable this firmware architecture it is important to provide a fully documented and stable interface between the Trusted Boot Firmware and the AArch32 EL3 Runtime Software.

Future changes to the entrypoint interface will be done in a backwards compatible way, and this enables these firmware components to be independently enhanced/updated to develop and exploit new functionality.

Required CPU state when entering during cold boot

This function must only be called by the primary CPU.

On entry to this function the calling primary CPU must be executing in AArch32 EL3, little-endian data access, and all interrupt sources masked:

```
PSTATE.AIF = 0x7
SCTLR.EE = 0
```

R0 and R1 are used to pass information from the Trusted Boot Firmware to the platform code in AArch32 EL3 Runtime Software:

```
R0 : Reserved for common TF-A information
R1 : Platform specific information
```

Use of the R0 and R1 parameters

The parameters are platform specific and the convention is that `R0` conveys information regarding the BL3x images from the Trusted Boot firmware and `R1` can be used for other platform specific purpose. This convention allows platforms which use TF-A's BL1 and BL2 images to transfer additional platform specific information from Secure Boot without conflicting with future evolution of TF-A using `R0` to pass a `bl_params` structure.

The AArch32 EL3 Runtime Software is responsible for entry into BL33. This information can be obtained in a platform defined manner, e.g. compiled into the AArch32 EL3 Runtime Software, or provided in a platform defined memory location by the Trusted Boot firmware, or passed from the Trusted Boot Firmware via the Cold boot Initialization parameters. This data may need to be cleaned out of the CPU caches if it is provided by an earlier boot stage and then accessed by AArch32 EL3 Runtime Software before the caches are enabled.

When using AArch32 EL3 Runtime Software, the Arm development platforms pass a `bl_params` structure in `R0` from BL2 to be interpreted by AArch32 EL3 Runtime Software platform code.

MMU, Data caches & Coherency

AArch32 EL3 Runtime Software must not depend on the enabled state of the MMU, data caches or interconnect coherency in its entrypoint. They must be explicitly enabled if required.

Data structures used in cold boot interface

The AArch32 EL3 Runtime Software cold boot interface uses `bl_params` instead of `bl31_params`. The `bl_params` structure is based on the convention described in AArch64 BL31 cold boot interface section.

Required CPU state for warm boot initialization

When requesting a CPU power-on, or suspending a running CPU, AArch32 EL3 Runtime Software must ensure execution of a warm boot initialization entrypoint. If TF-A BL1 is used and the PROGRAMMABLE_RESET_ADDRESS build flag is false, then AArch32 EL3 Runtime Software must ensure that BL1 branches to the warm boot entrypoint by arranging for the BL1 platform function, plat_get_my_entrypoint(), to return a non-zero value.

In this case, the warm boot entrypoint must be in AArch32 EL3, little-endian data access and all interrupt sources masked:

```
PSTATE.AIF = 0x7
SCTLR.EE = 0
```

The warm boot entrypoint may be implemented by using TF-A `psci_warmboot_entrypoint()` function. In that case, the platform must fulfil the pre-requisites mentioned in the *Porting Guide*.

5.4.2 EL3 runtime services framework

Software executing in the non-secure state and in the secure state at exception levels lower than EL3 will request runtime services using the Secure Monitor Call (SMC) instruction. These requests will follow the convention described in the SMC Calling Convention PDD (SMCCC). The SMCCC assigns function identifiers to each SMC request and describes how arguments are passed and returned.

The EL3 runtime services framework enables the development of services by different providers that can be easily integrated into final product firmware. The following sections describe the framework which facilitates the registration, initialization and use of runtime services in EL3 Runtime Software (BL31).

The design of the runtime services depends heavily on the concepts and definitions described in the SMCCC, in particular SMC Function IDs, Owning Entity Numbers (OEN), Fast and Yielding calls, and the SMC32 and SMC64 calling conventions. Please refer to that document for more detailed explanation of these terms.

The following runtime services are expected to be implemented first. They have not all been instantiated in the current implementation.

1. Standard service calls

   This service is for management of the entire system. The Power State Coordination Interface (PSCI) is the first set of standard service calls defined by Arm (see PSCI section later).

2. Secure-EL1 Payload Dispatcher service

   If a system runs a Trusted OS or other Secure-EL1 Payload (SP) then it also requires a *Secure Monitor* at EL3 to switch the EL1 processor context between the normal world (EL1/EL2) and trusted world (Secure-EL1). The Secure Monitor will make these world switches in response to SMCs. The SMCCC provides for such SMCs with the Trusted OS Call and Trusted Application Call OEN ranges.

   The interface between the EL3 Runtime Software and the Secure-EL1 Payload is not defined by the SMCCC or any other standard. As a result, each Secure-EL1 Payload requires a specific Secure Monitor that runs as a runtime service - within TF-A this service is referred to as the Secure-EL1 Payload Dispatcher (SPD).

   TF-A provides a Test Secure-EL1 Payload (TSP) and its associated Dispatcher (TSPD). Details of SPD design and TSP/TSPD operation are described in the *Secure-EL1 Payloads and Dispatchers* section below.

3. CPU implementation service

   This service will provide an interface to CPU implementation specific services for a given platform e.g. access to processor errata workarounds. This service is currently unimplemented.

Additional services for Arm Architecture, SiP and OEM calls can be implemented. Each implemented service handles a range of SMC function identifiers as described in the SMCCC.

Registration

A runtime service is registered using the `DECLARE_RT_SVC()` macro, specifying the name of the service, the range of OENs covered, the type of service and initialization and call handler functions. This macro instantiates a `const struct rt_svc_desc` for the service with these details (see `runtime_svc.h`). This structure is allocated in a special ELF section `.rt_svc_descs`, enabling the framework to find all service descriptors included into BL31.

The specific service for a SMC Function is selected based on the OEN and call type of the Function ID, and the framework uses that information in the service descriptor to identify the handler for the SMC Call.

The service descriptors do not include information to identify the precise set of SMC function identifiers supported by this service implementation, the security state from which such calls are valid nor the capability to support 64-bit and/or 32-bit callers (using SMC32 or SMC64). Responding appropriately to these aspects of a SMC call is the responsibility of the service implementation, the framework is focused on integration of services from different providers and minimizing the time taken by the framework before the service handler is invoked.

Details of the parameters, requirements and behavior of the initialization and call handling functions are provided in the following sections.

Initialization

`runtime_svc_init()` in `runtime_svc.c` initializes the runtime services framework running on the primary CPU during cold boot as part of the BL31 initialization. This happens prior to initializing a Trusted OS and running Normal world boot firmware that might in turn use these services. Initialization involves validating each of the declared runtime service descriptors, calling the service initialization function and populating the index used for runtime lookup of the service.

The BL31 linker script collects all of the declared service descriptors into a single array and defines symbols that allow the framework to locate and traverse the array, and determine its size.

The framework does basic validation of each descriptor to halt firmware initialization if service declaration errors are detected. The framework does not check descriptors for the following error conditions, and may behave in an unpredictable manner under such scenarios:

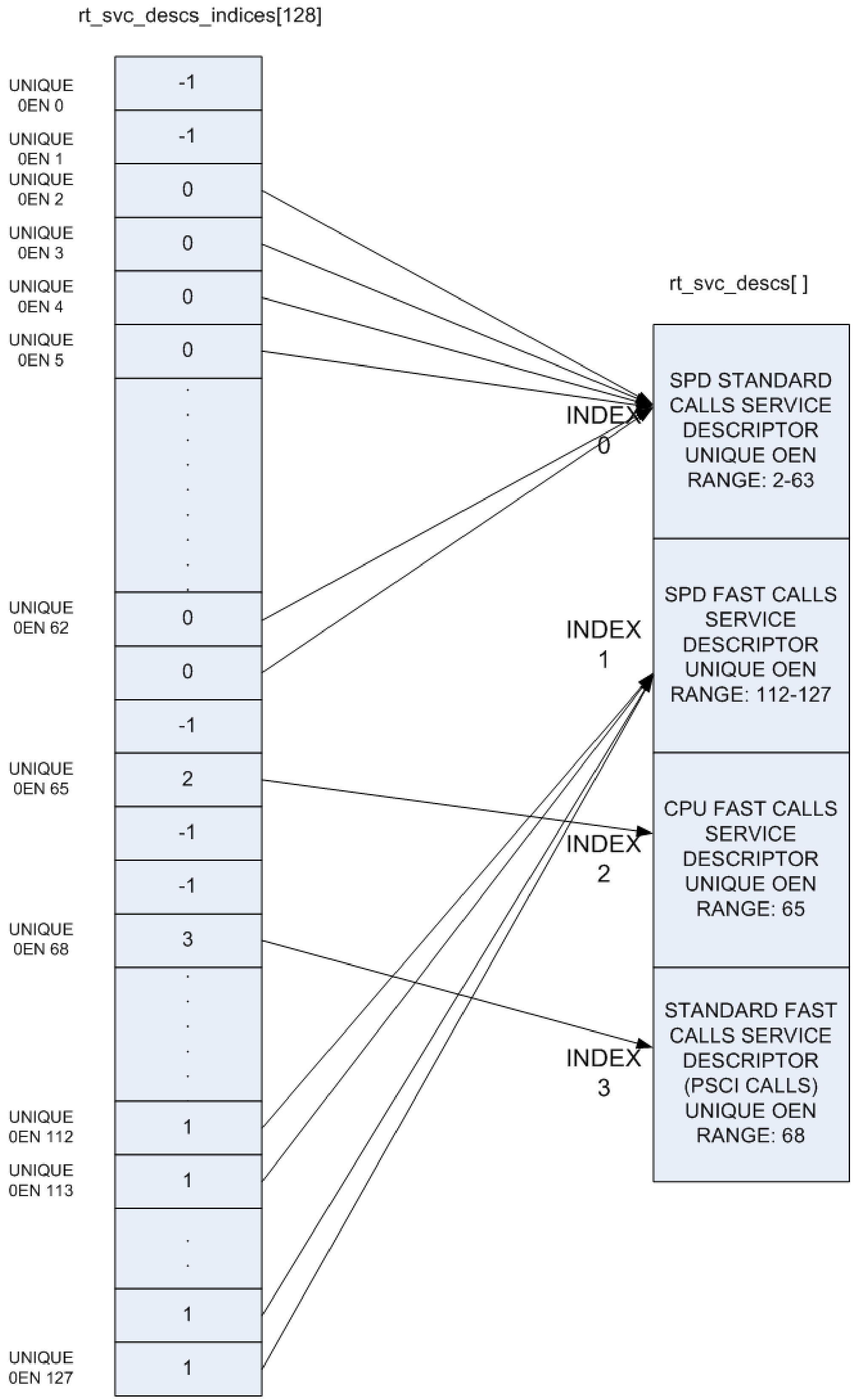
1. Overlapping OEN ranges
2. Multiple descriptors for the same range of OENs and `call_type`
3. Incorrect range of owning entity numbers for a given `call_type`

Once validated, the service `init()` callback is invoked. This function carries out any essential EL3 initialization before servicing requests. The `init()` function is only invoked on the primary CPU during cold boot. If the service uses per-CPU data this must either be initialized for all CPUs during this call, or be done lazily when a CPU first issues an SMC call to that service. If `init()` returns anything other than `0`, this is treated as an initialization error and the service is ignored: this does not cause the firmware to halt.

The OEN and call type fields present in the SMC Function ID cover a total of 128 distinct services, but in practice a single descriptor can cover a range of OENs, e.g. SMCs to call a Trusted OS function. To optimize the lookup of a service handler, the framework uses an array of 128 indices that map every distinct OEN/call-type combination either to one of the declared services or to indicate the service is not handled.

This `rt_svc_descs_indices[]` array is populated for all of the OENs covered by a service after the service `init()` function has reported success. So a service that fails to initialize will never have it's `handle()` function invoked.

The following figure shows how the `rt_svc_descs_indices[]` index maps the SMC Function ID call type and OEN onto a specific service handler in the `rt_svc_descs[]` array.


rt_svc_descs_indices[128]
UNIQUE 0EN 0
-1
UNIQUE 0EN 1
-1
UNIQUE 0EN 2
0
UNIQUE 0EN 3
0
UNIQUE 0EN 4
0
UNIQUE 0EN 5
0
UNIQUE 0EN 62
0
0
-1
UNIQUE 0EN 65
2
-1
-1
UNIQUE 0EN 68
3
UNIQUE 0EN 112
1
UNIQUE 0EN 113
1
1
UNIQUE 0EN 127
1
rt_svc_descs[ ]
INDEX 0
SPD STANDARD CALLS SERVICE DESCRIPTOR UNIQUE OEN RANGE: 2-63
INDEX 1
SPD FAST CALLS SERVICE DESCRIPTOR UNIQUE OEN RANGE: 112-127
INDEX 2
CPU FAST CALLS SERVICE DESCRIPTOR UNIQUE OEN RANGE: 65
INDEX 3
STANDARD FAST CALLS SERVICE DESCRIPTOR (PSCI CALLS) UNIQUE OEN RANGE: 68

Handling an SMC

When the EL3 runtime services framework receives a Secure Monitor Call, the SMC Function ID is passed in W0 from the lower exception level (as per the SMCCC). If the calling register width is AArch32, it is invalid to invoke an SMC Function which indicates the SMC64 calling convention: such calls are ignored and return the Unknown SMC Function Identifier result code `0xFFFFFFFF` in R0/X0.

Bit[31] (fast/yielding call) and bits[29:24] (owning entity number) of the SMC Function ID are combined to index into the `rt_svc_descs_indices[]` array. The resulting value might indicate a service that has no handler, in this case the framework will also report an Unknown SMC Function ID. Otherwise, the value is used as a further index into the `rt_svc_descs[]` array to locate the required service and handler.

The service's `handle()` callback is provided with five of the SMC parameters directly, the others are saved into memory for retrieval (if needed) by the handler. The handler is also provided with an opaque `handle` for use with the supporting library for parameter retrieval, setting return values and context manipulation. The `flags` parameter indicates the security state of the caller and the state of the SVE hint bit per the SMCCCv1.3. The framework finally sets up the execution stack for the handler, and invokes the services `handle()` function.

On return from the handler the result registers are populated in X0-X7 as needed before restoring the stack and CPU state and returning from the original SMC.

5.4.3 Exception Handling Framework

Please refer to the *Exception Handling Framework* document.

5.4.4 Power State Coordination Interface

TODO: Provide design walkthrough of PSCI implementation.

The PSCI v1.1 specification categorizes APIs as optional and mandatory. All the mandatory APIs in PSCI v1.1, PSCI v1.0 and in PSCI v0.2 draft specification PSCI are implemented. The table lists the PSCI v1.1 APIs and their support in generic code.

An API implementation might have a dependency on platform code e.g. CPU_SUSPEND requires the platform to export a part of the implementation. Hence the level of support of the mandatory APIs depends upon the support exported by the platform port as well. The Juno and FVP (all variants) platforms export all the required support.

PSCI v1.1 API	Supported	Comments
PSCI_VERSION	Yes	The version returned is 1.1
CPU_SUSPEND	Yes*	
CPU_OFF	Yes*	
CPU_ON	Yes*	
AFFINITY_INFO	Yes	
MIGRATE	Yes**	
MIGRATE_INFO_TYPE	Yes**	
MIGRATE_INFO_CPU	Yes**	
SYSTEM_OFF	Yes*	
SYSTEM_RESET	Yes*	
PSCI_FEATURES	Yes	
CPU_FREEZE	No	
CPU_DEFAULT_SUSPEND	No	
NODE_HW_STATE	Yes*	
SYSTEM_SUSPEND	Yes*	
PSCI_SET_SUSPEND_MODE	No	
PSCI_STAT_RESIDENCY	Yes*	
PSCI_STAT_COUNT	Yes*	
SYSTEM_RESET2	Yes*	
MEM_PROTECT	Yes*	
MEM_PROTECT_CHECK_RANGE	Yes*	

*Note : These PSCI APIs require platform power management hooks to be registered with the generic PSCI code to be supported.

**Note : These PSCI APIs require appropriate Secure Payload Dispatcher hooks to be registered with the generic PSCI code to be supported.

The PSCI implementation in TF-A is a library which can be integrated with AArch64 or AArch32 EL3 Runtime Software for Armv8-A systems. A guide to integrating the PSCI library for EL3 Runtime Software can be found at *Porting Guide*.

5.4.5 DSU driver

Platforms that include a DSU (DynamIQ Shared Unit) can define the `USE_DSU_DRIVER` build flag to enable the DSU driver. This driver is responsible for configuring DSU-related powerdown and power feature settings, enabling access to PMU registers at EL1 using `dsu_driver_init()` and for preserving the context of DSU PMU system registers.

To support the DSU driver, platforms must define the `plat_dsu_data` structure.

5.4.6 Secure-EL1 Payloads and Dispatchers

On a production system that includes a Trusted OS running in Secure-EL1/EL0, the Trusted OS is coupled with a companion runtime service in the BL31 firmware. This service is responsible for the initialisation of the Trusted OS and all communications with it. The Trusted OS is the BL32 stage of the boot flow in TF-A. The firmware will attempt to locate, load and execute a BL32 image.

TF-A uses a more general term for the BL32 software that runs at Secure-EL1 - the *Secure-EL1 Payload* - as it is not always a Trusted OS.

TF-A provides a Test Secure-EL1 Payload (TSP) and a Test Secure-EL1 Payload Dispatcher (TSPD) service as an example of how a Trusted OS is supported on a production system using the Runtime Services Framework. On such a system, the Test BL32 image and service are replaced by the Trusted OS and its dispatcher service. The TF-A build system expects that the dispatcher will define the build flag `NEED_BL32` to enable it to include the BL32 in the build either as a binary or to compile from source depending on whether the `BL32` build option is specified or not.

The TSP runs in Secure-EL1. It is designed to demonstrate synchronous communication with the normal-world software running in EL1/EL2. Communication is initiated by the normal-world software

- either directly through a Fast SMC (as defined in the SMCCC)
- or indirectly through a PSCI SMC. The PSCI implementation in turn informs the TSPD about the requested power management operation. This allows the TSP to prepare for or respond to the power state change

The TSPD service is responsible for.

- Initializing the TSP
- Routing requests and responses between the secure and the non-secure states during the two types of communications just described

Initializing a BL32 Image

The Secure-EL1 Payload Dispatcher (SPD) service is responsible for initializing the BL32 image. It needs access to the information passed by BL2 to BL31 to do so. This is provided by:

```
entry_point_info_t *bl31_plat_get_next_image_ep_info(uint32_t);
```

which returns a reference to the `entry_point_info` structure corresponding to the image which will be run in the specified security state. The SPD uses this API to get entry point information for the SECURE image, BL32.

In the absence of a BL32 image, BL31 passes control to the normal world bootloader image (BL33). When the BL32 image is present, it is typical that the SPD wants control to be passed to BL32 first and then later to BL33.

To do this the SPD has to register a BL32 initialization function during initialization of the SPD service. The BL32 initialization function has this prototype:

```
int32_t init(void);
```

and is registered using the `bl31_register_bl32_init()` function.

TF-A supports two approaches for the SPD to pass control to BL32 before returning through EL3 and running the non-trusted firmware (BL33):

1. In the BL32 setup function, use `bl31_set_next_image_type()` to request that the exit from `bl31_main()` is to the BL32 entrypoint in Secure-EL1. BL31 will exit to BL32 using the asynchronous method by calling `bl31_prepare_next_image_entry()` and `el3_exit()`.

   When the BL32 has completed initialization at Secure-EL1, it returns to BL31 by issuing an SMC, using a Function ID allocated to the SPD. On receipt of this SMC, the SPD service handler should switch the CPU context from trusted to normal world and use the `bl31_set_next_image_type()` and `bl31_prepare_next_image_entry()` functions to set up the initial return to the normal world firmware BL33. On return from the handler the framework will exit to EL2 and run BL33.

2. The BL32 setup function registers an initialization function using `bl31_register_bl32_init()` which provides a SPD-defined mechanism to invoke a 'world-switch synchronous call' to Secure-EL1 to run the BL32 entrypoint.

   Note: The Test SPD service included with TF-A provides one implementation of such a mechanism.

   On completion BL32 returns control to BL31 via a SMC, and on receipt the SPD service handler invokes the synchronous call return mechanism to return to the BL32 initialization function. On return from this function, `bl31_main()` will set up the return to the normal world firmware BL33 and continue the boot process in the normal world.

5.4.7 Exception handling in BL31

When exception occurs, PE must execute handler corresponding to exception. The location in memory where the handler is stored is called the exception vector. For ARM architecture, exception vectors are stored in a table, called the exception vector table.

Each EL (except EL0) has its own vector table, VBAR_ELn register stores the base of vector table. Refer to AArch64 exception vector table

Current EL with SP_EL0

- Sync exception : Not expected except for BRK instruction, its debugging tool which a programmer may place at specific points in a program, to check the state of processor flags at these points in the code.
- IRQ/FIQ : Unexpected exception, panic
- SError : "plat_handle_el3_ea", defaults to panic

Current EL with SP_ELx

- Sync exception : Unexpected exception, panic
- IRQ/FIQ : Unexpected exception, panic
- SError : "plat_handle_el3_ea" Except for special handling of lower EL's SError exception which gets triggered in EL3 when PSTATE.A is unmasked. Its only applicable when lower EL's EA is routed to EL3 (FFH_SUPPORT=1).

Lower EL Exceptions

Applies to all the exceptions in both AArch64/AArch32 mode of lower EL.

Before handling any lower EL exception, we synchronize the errors at EL3 entry to ensure that any errors pertaining to lower EL is isolated/identified. If we continue without identifying these errors early on then these errors will trigger in EL3 (as SError from current EL) any time after PSTATE.A is unmasked. This is wrong because the error originated in lower EL but exception happened in EL3.

To solve this problem, synchronize the errors at EL3 entry and check for any pending errors (async EA). If there is no pending error then continue with original exception. If there is a pending error then, handle them based on routing model of EA's. Refer to *Reliability, Availability, and Serviceability (RAS) Extensions* for details about routing models.

- KFH : Reflect it back to lower EL using **reflect_pending_async_ea_to_lower_el()**
- FFH : Handle the synchronized error first using **handle_pending_async_ea()** after that continue with original exception. It is the only scenario where EL3 is capable of doing nested exception handling.

After synchronizing and handling lower EL SErrors, unmask EA (PSTATE.A) to ensure that any further EA's caused by EL3 are caught.

5.4.8 Crash Reporting in BL31

BL31 implements a scheme for reporting the processor state when an unhandled exception is encountered. The reporting mechanism attempts to preserve all the register contents and report it via a dedicated UART (PL011 console). BL31 reports the general purpose, EL3, Secure EL1 and some EL2 state registers.

A dedicated per-CPU crash stack is maintained by BL31 and this is retrieved via the per-CPU pointer cache. The implementation attempts to minimise the memory required for this feature. The file `crash_reporting.S` contains the implementation for crash reporting.

The sample crash output is shown below.

```
x0              = 0x000000002a4a0000
x1              = 0x0000000000000001
x2              = 0x0000000000000002
x3              = 0x0000000000000003
x4              = 0x0000000000000004
x5              = 0x0000000000000005
x6              = 0x0000000000000006
```

(continues on next page)

(continued from previous page)

```
x7             = 0x0000000000000007
x8             = 0x0000000000000008
x9             = 0x0000000000000009
x10            = 0x0000000000000010
x11            = 0x0000000000000011
x12            = 0x0000000000000012
x13            = 0x0000000000000013
x14            = 0x0000000000000014
x15            = 0x0000000000000015
x16            = 0x0000000000000016
x17            = 0x0000000000000017
x18            = 0x0000000000000018
x19            = 0x0000000000000019
x20            = 0x0000000000000020
x21            = 0x0000000000000021
x22            = 0x0000000000000022
x23            = 0x0000000000000023
x24            = 0x0000000000000024
x25            = 0x0000000000000025
x26            = 0x0000000000000026
x27            = 0x0000000000000027
x28            = 0x0000000000000028
x29            = 0x0000000000000029
x30            = 0x0000000088000b78
scr_el3        = 0x000000000003073d
sctlr_el3      = 0x00000000b0cd183f
cptr_el3       = 0x0000000000000000
tcr_el3        = 0x000000008080351c
daif           = 0x00000000000002c0
mair_el3       = 0x00000000004404ff
spsr_el3       = 0x0000000060000349
elr_el3        = 0x0000000088000114
ttbr0_el3      = 0x0000000004018201
esr_el3        = 0x00000000be000000
far_el3        = 0x0000000000000000
spsr_el1       = 0x0000000000000000
elr_el1        = 0x0000000000000000
spsr_abt       = 0x0000000000000000
spsr_und       = 0x0000000000000000
spsr_irq       = 0x0000000000000000
spsr_fiq       = 0x0000000000000000
sctlr_el1      = 0x0000000030d00800
actlr_el1      = 0x0000000000000000
cpacr_el1      = 0x0000000000000000
csselr_el1     = 0x0000000000000000
sp_el1         = 0x0000000000000000
esr_el1        = 0x0000000000000000
ttbr0_el1      = 0x0000000000000000
ttbr1_el1      = 0x0000000000000000
mair_el1       = 0x0000000000000000
amair_el1      = 0x0000000000000000
tcr_el1        = 0x0000000000000000
```

(continues on next page)

(continued from previous page)

```
tpidr_el1        = 0x0000000000000000
tpidr_el0        = 0x0000000000000000
tpidrro_el0      = 0x0000000000000000
par_el1          = 0x0000000000000000
mpidr_el1        = 0x0000000080000000
afsr0_el1        = 0x0000000000000000
afsr1_el1        = 0x0000000000000000
contextidr_el1 = 0x0000000000000000
vbar_el1         = 0x0000000000000000
cntp_ctl_el0     = 0x0000000000000000
cntp_cval_el0    = 0x0000000000000000
cntv_ctl_el0     = 0x0000000000000000
cntv_cval_el0    = 0x0000000000000000
cntkctl_el1      = 0x0000000000000000
sp_el0           = 0x0000000004014940
isr_el1          = 0x0000000000000000
dacr32_el2       = 0x0000000000000000
ifsr32_el2       = 0x0000000000000000
icc_hppir0_el1 = 0x00000000000003ff
icc_hppir1_el1 = 0x00000000000003ff
icc_ctlr_el3     = 0x0000000000080400
gicd_ispendr regs (Offsets 0x200-0x278)
Offset                  Value
0x200:           0x0000000000000000
0x208:           0x0000000000000000
0x210:           0x0000000000000000
0x218:           0x0000000000000000
0x220:           0x0000000000000000
0x228:           0x0000000000000000
0x230:           0x0000000000000000
0x238:           0x0000000000000000
0x240:           0x0000000000000000
0x248:           0x0000000000000000
0x250:           0x0000000000000000
0x258:           0x0000000000000000
0x260:           0x0000000000000000
0x268:           0x0000000000000000
0x270:           0x0000000000000000
0x278:           0x0000000000000000
```

5.4.9 Guidelines for Reset Handlers

TF-A implements a framework that allows CPU and platform ports to perform actions very early after a CPU is released from reset in both the cold and warm boot paths. This is done by calling the `reset_handler` macro/function in both the BL1 and BL31 images. It in turn calls the platform and CPU specific reset handling functions.

Details for implementing a CPU specific reset handler can be found in *CPU specific Reset Handling*. Details for implementing a platform specific reset handler can be found in the *Porting Guide* (see the``plat_reset_handler()`` function).

When adding functionality to a reset handler, keep in mind that if a different reset handling behavior is required between the first and the subsequent invocations of the reset handling code, this should be detected at runtime. In other words, the reset handler should be able to detect whether an action has already been performed and act as appropriate. Possible courses of actions are, e.g. skip the action the second time, or undo/redo it.

5.4.10 Configuring secure interrupts

The GIC driver is responsible for performing initial configuration of secure interrupts on the platform. To this end, the platform is expected to provide the GIC driver (either GICv2 or GICv3, as selected by the platform) with the interrupt configuration during the driver initialisation.

Secure interrupt configuration are specified in an array of secure interrupt properties. In this scheme, in both GICv2 and GICv3 driver data structures, the `interrupt_props` member points to an array of interrupt properties. Each element of the array specifies the interrupt number and its attributes (priority, group, configuration). Each element of the array shall be populated by the macro `INTR_PROP_DESC()`. The macro takes the following arguments:

- 13-bit interrupt number,
- 8-bit interrupt priority,
- Interrupt type (one of `INTR_TYPE_EL3`, `INTR_TYPE_S_EL1`, `INTR_TYPE_NS`),
- Interrupt configuration (either `GIC_INTR_CFG_LEVEL` or `GIC_INTR_CFG_EDGE`).

5.4.11 CPU specific operations framework

Certain aspects of the Armv8-A architecture are implementation defined, that is, certain behaviours are not architecturally defined, but must be defined and documented by individual processor implementations. TF-A implements a framework which categorises the common implementation defined behaviours and allows a processor to export its implementation of that behaviour. The categories are:

1. Processor specific reset sequence.
2. Processor specific power down sequences.
3. Processor specific register dumping as a part of crash reporting.
4. Errata status reporting.

Each of the above categories fulfils a different requirement.

1. allows any processor specific initialization before the caches and MMU are turned on, like implementation of errata workarounds, entry into the intra-cluster coherency domain etc.
2. allows each processor to implement the power down sequence mandated in its Technical Reference Manual (TRM).
3. allows a processor to provide additional information to the developer in the event of a crash, for example Cortex-A53 has registers which can expose the data cache contents.
4. allows a processor to define a function that inspects and reports the status of all errata workarounds on that processor.

Please note that only 2. is mandated by the TRM.

The CPU specific operations framework scales to accommodate a large number of different CPUs during power down and reset handling. The platform can specify any CPU optimization it wants to enable for each CPU. It can also specify the CPU errata workarounds to be applied for each CPU type during reset handling by defining CPU errata compile time macros. Details on these macros can be found in the *Arm CPU Specific Build Macros* document.

The CPU specific operations framework depends on the `cpu_ops` structure which needs to be exported for each type of CPU in the platform. It is defined in `include/lib/cpus/aarch64/cpu_macros.S` and has the following fields : `midr`, `reset_func()`, `cpu_pwr_down_ops` (array of power down functions) and `cpu_reg_dump()`.

The CPU specific files in `lib/cpus` export a `cpu_ops` data structure with suitable handlers for that CPU. For example, `lib/cpus/aarch64/cortex_a53.S` exports the `cpu_ops` for Cortex-A53 CPU. According to the platform configuration, these CPU specific files must be included in the build by the platform makefile. The generic CPU specific operations framework code exists in `lib/cpus/aarch64/cpu_helpers.S`.

CPU PCS

All assembly functions in CPU files are asked to follow a modified version of the Procedure Call Standard (PCS) in their internals. This is done to ensure calling these functions from outside the file doesn't unexpectedly corrupt registers in the very early environment and to help the internals to be easier to understand. Please see the *CPU errata implementation* for any function specific restrictions.

register	use
x0 - x15	scratch
x16, x17	do not use (used by the linker)
x18	do not use (platform register)
x19 - x28	callee saved
x29, x30	FP, LR

CPU specific Reset Handling

After a reset, the state of the CPU when it calls generic reset handler is: MMU turned off, both instruction and data caches turned off, not part of any coherency domain and no stack.

The BL entrypoint code first invokes the `plat_reset_handler()` to allow the platform to perform any system initialization required and any system errata workarounds that needs to be applied. The `get_cpu_ops_ptr()` reads the current CPU midr, finds the matching `cpu_ops` entry in the `cpu_ops` array and returns it. Note that only the part number and implementer fields in midr are used to find the matching `cpu_ops` entry. The `reset_func()` in the returned `cpu_ops` is then invoked which executes the required reset handling for that CPU and also any errata workarounds enabled by the platform.

It should be defined using the `cpu_reset_func_{start,end}` macros and its body may only clobber x0 to x14 with x14 being the cpu_rev parameter. The cpu file should also include a call to `cpu_reset_prologue` at the start of the file for errata to work correctly.

CPU specific power down sequence

During the BL31 initialization sequence, the pointer to the matching `cpu_ops` entry is stored in per-CPU data by `cpu_data_init_cpu_ops()` so that it can be quickly retrieved during power down sequences.

Various CPU drivers register handlers to perform power down at certain power levels for that specific CPU. The PSCI service, upon receiving a power down request, determines the highest power level at which to execute power down sequence for a particular CPU. It uses the `prepare_cpu_pwr_dwn()` function to pick the right power down handler for the requested level. The function retrieves `cpu_ops` pointer member of per-CPU data, and from that, further retrieves `cpu_pwr_down_ops` array, and indexes into the required level. If the requested power level is higher than what a CPU driver supports, the handler registered for highest level is invoked.

At runtime the platform hooks for power down are invoked by the PSCI service to perform platform specific operations during a power down sequence, for example turning off CCI coherency during a cluster power down.

Newer CPUs include a feature called "powerdown abandon". The feature is based on the observation that events like GIC wakeups have a high likelihood of happening while the core is in the middle of its powerdown sequence (at `wfi`). Older cores will powerdown and immediately power back up when this happens. To save on the work and latency involved, the newer cores will "give up" mid way through if no context has been lost yet. This is possible as the powerdown operation is lengthy and a large part of it does not lose context.

To cater for this possibility, the powerdown hook will be called a second time after a wakeup. The expectation is that the first call will operate as before, while the second call will undo anything the first call did. This should be done statelessly, for example by toggling the relevant bits.

CPU specific register reporting during crash

If the crash reporting is enabled in BL31, when a crash occurs, the crash reporting framework calls `do_cpu_reg_dump` which retrieves the matching `cpu_ops` using `get_cpu_ops_ptr()` function. The `cpu_reg_dump()` in `cpu_ops` is invoked, which then returns the CPU specific register values to be reported and a pointer to the ASCII list of register names in a format expected by the crash reporting framework.

CPU errata implementation

Errata workarounds for CPUs supported in TF-A are applied during both cold and warm boots, shortly after reset. Individual Errata workarounds are enabled as build options. Some errata workarounds have potential run-time implications; therefore some are enabled by default, others not. Platform ports shall override build options to enable or disable errata as appropriate. The CPU drivers take care of applying errata workarounds that are enabled and applicable to a given CPU.

Each erratum has a build flag in `lib/cpus/cpu-ops.mk` of the form: `ERRATA_<cpu_num>_<erratum_id>`. It also has a short description in *CPU Errata Workarounds* on when it should apply.

Errata framework

The errata framework is a convention and a small library to allow errata to be automatically discovered. It enables compliant errata to be automatically applied and reported at runtime (either by status reporting or the errata ABI).

To write a compliant mitigation for erratum number erratum_id on a cpu that declared itself (with declare_cpu_ops) as cpu_name one needs 3 things:

1. A CPU revision checker function: check_erratum_<cpu_name>_<erratum_id>

   It should check whether this erratum applies on this revision of this CPU. It will be called with the CPU revision as its first parameter (x0) and should return one of ERRATA_APPLIES or ERRATA_NOT_APPLIES.

   It may only clobber x0 to x4. The rest should be treated as callee-saved.

2. A workaround function: erratum_<cpu_name>_<erratum_id>_wa

   It should obtain the cpu revision (with cpu_get_rev_var), call its revision checker, and perform the mitigation, should the erratum apply.

   It may only clobber x0 to x8. The rest should be treated as callee-saved.

3. Register itself to the framework

   Do this with add_erratum_entry <cpu_name>, ERRATUM(<erratum_id>), <errata_flag> where the errata_flag is the enable flag in cpu-ops.mk described above.

See the next section on how to do this easily.

Note: CVEs have the format CVE_<year>_<number>. To fit them in the framework, the erratum_id for the checker and the workaround functions become the number part of its name and the ERRATUM(<number>) part of the registration should instead be CVE(<year>, <number>). In the extremely unlikely scenario where a CVE and an erratum numbers clash, the CVE number should be prefixed with a zero.

Also, their build flag should be WORKAROUND_CVE_<year>_<number>.

Note: AArch32 uses the legacy convention. The checker function has the format check_errata_<erratum_id> and the workaround has the format errata_<cpu_number>_<erratum_id>_wa where cpu_number is the shortform letter and number name of the CPU.

For CVEs the erratum_id also becomes cve_<year>_<number>.

Errata framework helpers

Writing these errata involves lots of boilerplate and repetitive code. On AArch64 there are helpers to omit most of this. They are located in `include/lib/cpus/aarch64/cpu_macros.S` and the preferred way to implement errata. Please see their comments on how to use them.

The most common type of erratum workaround, one that just sets a "chicken" bit in some arbitrary register, would have an implementation for the Cortex-A77, erratum #1925769 like:

```
workaround_reset_start cortex_a77, ERRATUM(1925769), ERRATA_A77_1925769
    sysreg_bit_set CORTEX_A77_CPUECTLR_EL1, CORTEX_A77_CPUECTLR_EL1_BIT_8
workaround_reset_end cortex_a77, ERRATUM(1925769)

check_erratum_ls cortex_a77, ERRATUM(1925769), CPU_REV(1, 1)
```

Status reporting

In a debug build of TF-A, on a CPU that comes out of reset, both BL1 and the runtime firmware (BL31 in AArch64, and BL32 in AArch32) will invoke a generic errata status reporting function. It will read the `errata_entries` list of that cpu and will report whether each known erratum was applied and, if not, whether it should have been.

Reporting the status of errata workaround is for informational purpose only; it has no functional significance.

5.4.12 Memory layout of BL images

Each bootloader image can be divided in 2 parts:

- the static contents of the image. These are data actually stored in the binary on the disk. In the ELF terminology, they are called `PROGBITS` sections;
- the run-time contents of the image. These are data that don't occupy any space in the binary on the disk. The ELF binary just contains some metadata indicating where these data will be stored at run-time and the corresponding sections need to be allocated and initialized at run-time. In the ELF terminology, they are called `NOBITS` sections.

All PROGBITS sections are grouped together at the beginning of the image, followed by all NOBITS sections. This is true for all TF-A images and it is governed by the linker scripts. This ensures that the raw binary images are as small as possible. If a NOBITS section was inserted in between PROGBITS sections then the resulting binary file would contain zero bytes in place of this NOBITS section, making the image unnecessarily bigger. Smaller images allow faster loading from the FIP to the main memory.

For BL31, a platform can specify an alternate location for NOBITS sections (other than immediately following PROGBITS sections) by setting `SEPARATE_NOBITS_REGION` to 1 and defining `BL31_NOBITS_BASE` and `BL31_NOBITS_LIMIT`.

Linker scripts and symbols

Each bootloader stage image layout is described by its own linker script. The linker scripts export some symbols into the program symbol table. Their values correspond to particular addresses. TF-A code can refer to these symbols to figure out the image memory layout.

Linker symbols follow the following naming convention in TF-A.

- __<SECTION>_START__

  Start address of a given section named <SECTION>.

- __<SECTION>_END__

  End address of a given section named <SECTION>. If there is an alignment constraint on the section's end address then __<SECTION>_END__ corresponds to the end address of the section's actual contents, rounded up to the right boundary. Refer to the value of __<SECTION>_UNALIGNED_END__ to know the actual end address of the section's contents.

- __<SECTION>_UNALIGNED_END__

  End address of a given section named <SECTION> without any padding or rounding up due to some alignment constraint.

- __<SECTION>_SIZE__

  Size (in bytes) of a given section named <SECTION>. If there is an alignment constraint on the section's end address then __<SECTION>_SIZE__ corresponds to the size of the section's actual contents, rounded up to the right boundary. In other words, __<SECTION>_SIZE__ = __<SECTION>_END__ - _<SECTION>_START__. Refer to the value of __<SECTION>_UNALIGNED_SIZE__ to know the actual size of the section's contents.

- __<SECTION>_UNALIGNED_SIZE__

  Size (in bytes) of a given section named <SECTION> without any padding or rounding up due to some alignment constraint. In other words, __<SECTION>_UNALIGNED_SIZE__ = __<SECTION>_UNALIGNED_END__ - __<SECTION>_START__.

Some of the linker symbols are mandatory as TF-A code relies on them to be defined. They are listed in the following subsections. Some of them must be provided for each bootloader stage and some are specific to a given bootloader stage.

The linker scripts define some extra, optional symbols. They are not actually used by any code but they help in understanding the bootloader images' memory layout as they are easy to spot in the link map files.

Common linker symbols

All BL images share the following requirements:

- The BSS section must be zero-initialised before executing any C code.
- The coherent memory section (if enabled) must be zero-initialised as well.
- The MMU setup code needs to know the extents of the coherent and read-only memory regions to set the right memory attributes. When `SEPARATE_CODE_AND_RODATA=1`, it needs to know more specifically how the read-only memory region is divided between code and data.

The following linker symbols are defined for this purpose:

- `__BSS_START__`
- `__BSS_SIZE__`
- `__COHERENT_RAM_START__` Must be aligned on a page-size boundary.
- `__COHERENT_RAM_END__` Must be aligned on a page-size boundary.
- `__COHERENT_RAM_UNALIGNED_SIZE__`
- `__RO_START__`
- `__RO_END__`
- `__TEXT_START__`
- `__TEXT_END_UNALIGNED__`
- `__TEXT_END__`
- `__RODATA_START__`
- `__RODATA_END_UNALIGNED__`
- `__RODATA_END__`

BL1's linker symbols

BL1 being the ROM image, it has additional requirements. BL1 resides in ROM and it is entirely executed in place but it needs some read-write memory for its mutable data. Its `.data` section (i.e. its allocated read-write data) must be relocated from ROM to RAM before executing any C code.

The following additional linker symbols are defined for BL1:

- `__BL1_ROM_END__` End address of BL1's ROM contents, covering its code and `.data` section in ROM.
- `__DATA_ROM_START__` Start address of the `.data` section in ROM. Must be aligned on a 16-byte boundary.
- `__DATA_RAM_START__` Address in RAM where the `.data` section should be copied over. Must be aligned on a 16-byte boundary.

- `__DATA_SIZE__` Size of the `.data` section (in ROM or RAM).
- `__BL1_RAM_START__` Start address of BL1 read-write data.
- `__BL1_RAM_END__` End address of BL1 read-write data.

How to choose the right base addresses for each bootloader stage image

There is currently no support for dynamic image loading in TF-A. This means that all bootloader images need to be linked against their ultimate runtime locations and the base addresses of each image must be chosen carefully such that images don't overlap each other in an undesired way. As the code grows, the base addresses might need adjustments to cope with the new memory layout.

The memory layout is completely specific to the platform and so there is no general recipe for choosing the right base addresses for each bootloader image. However, there are tools to aid in understanding the memory layout. These are the link map files: `build/<platform>/<build-type>/bl<x>/bl<x>.map`, with `<x>` being the stage bootloader. They provide a detailed view of the memory usage of each image. Among other useful information, they provide the end address of each image.

- `bl1.map` link map file provides `__BL1_RAM_END__` address.
- `bl2.map` link map file provides `__BL2_END__` address.
- `bl31.map` link map file provides `__BL31_END__` address.
- `bl32.map` link map file provides `__BL32_END__` address.

For each bootloader image, the platform code must provide its start address as well as a limit address that it must not overstep. The latter is used in the linker scripts to check that the image doesn't grow past that address. If that happens, the linker will issue a message similar to the following:

```
aarch64-none-elf-ld: BLx has exceeded its limit.
```

Additionally, if the platform memory layout implies some image overlaying like on FVP, BL31 and TSP need to know the limit address that their PROGBITS sections must not overstep. The platform code must provide those.

TF-A does not provide any mechanism to verify at boot time that the memory to load a new image is free to prevent overwriting a previously loaded image. The platform must specify the memory available in the system for all the relevant BL images to be loaded.

For example, in the case of BL1 loading BL2, `bl1_plat_sec_mem_layout()` will return the region defined by the platform where BL1 intends to load BL2. The `load_image()` function performs bounds check for the image size based on the base and maximum image size provided by the platforms. Platforms must take this behaviour into account when defining the base/size for each of the images.

Memory layout on Arm development platforms

The following list describes the memory layout on the Arm development platforms:

- A 4KB page of shared memory is used for communication between Trusted Firmware and the platform's power controller. This is located at the base of Trusted SRAM. The amount of Trusted SRAM available to load the bootloader images is reduced by the size of the shared memory.

  The shared memory is used to store the CPUs' entrypoint mailbox. On Juno, this is also used for the MHU payload when passing messages to and from the SCP.

- Another 4 KB page is reserved for passing memory layout between BL1 and BL2 and also the dynamic firmware configurations.
- On FVP, BL1 is originally sitting in the Trusted ROM at address `0x0`. On Juno, BL1 resides in flash memory at address `0x0BEC0000`. BL1 read-write data are relocated to the top of Trusted SRAM at runtime.
- BL2 is loaded below BL1 RW
- EL3 Runtime Software, BL31 for AArch64 and BL32 for AArch32 (e.g. SP_MIN), is loaded at the top of the Trusted SRAM, such that its NOBITS sections will overwrite BL1 R/W data and BL2. This implies that BL1 global variables remain valid only until execution reaches the EL3 Runtime Software entry point during a cold boot.
- On Juno, SCP_BL2 is loaded temporarily into the EL3 Runtime Software memory region and transferred to the SCP before being overwritten by EL3 Runtime Software.
- BL32 (for AArch64) can be loaded in one of the following locations:
  - Trusted SRAM
  - Trusted DRAM (FVP only)
  - Secure region of DRAM (top 16MB of DRAM configured by the TrustZone controller)

  When BL32 (for AArch64) is loaded into Trusted SRAM, it is loaded below BL31.

The location of the BL32 image will result in different memory maps. This is illustrated for both FVP and Juno in the following diagrams, using the TSP as an example.

Note: Loading the BL32 image in TZC secured DRAM doesn't change the memory layout of the other images in Trusted SRAM.

CONFIG section in memory layouts shown below contains:

```
+--------------------+
|bl2_mem_params_descs|
|--------------------|
|     fw_configs     |
+--------------------+
```

`bl2_mem_params_descs` contains parameters passed from BL2 to next the BL image during boot.

`fw_configs` includes soc_fw_config, tos_fw_config, tb_fw_config and fw_config.

FVP with TSP in Trusted SRAM with firmware configs : (These diagrams only cover the AArch64 case)

```
               DRAM
0xffffffff +----------+
           | EL3 TZC  |
0xffe00000 |----------|  (secure)
           | AP TZC   |
0xff000000 +----------+
           :          :
0x82100000 |----------|
           |HW_CONFIG |
0x82000000 |----------|   (non-secure)
           |          |
0x80000000 +----------+

           Trusted DRAM
0x08000000 +----------+
           |HW_CONFIG |
0x07f00000 |----------|
           :          :
           |          |
0x06000000 +----------+

           Trusted SRAM
0x04040000 +----------+  loaded by BL2  +----------------+
           | BL1 (rw) |  <<<<<<<<<<<<<  |                |
           |----------|  <<<<<<<<<<<<<  |  BL31 NOBITS   |
           |   BL2    |  <<<<<<<<<<<<<  |                |
           |----------|  <<<<<<<<<<<<<  |----------------|
           |          |  <<<<<<<<<<<<<  | BL31 PROGBITS  |
           |          |  <<<<<<<<<<<<<  |----------------|
           |          |  <<<<<<<<<<<<<  |     BL32       |
0x04003000 +----------+                 +----------------+
           |  CONFIG  |
0x04001000 +----------+
           |  Shared  |
0x04000000 +----------+

           Trusted ROM
0x04000000 +----------+
           | BL1 (ro) |
0x00000000 +----------+
```

FVP with TSP in Trusted DRAM with firmware configs (default option):

```
                 DRAM
0xffffffff +--------------+
           |   EL3 TZC    |
0xffe00000 |--------------|  (secure)
           |   AP TZC     |
0xff000000 +--------------+
```

(continues on next page)

(continued from previous page)

```
           :               :
0x82100000 |---------------|
           |   HW_CONFIG   |
0x82000000 |---------------|  (non-secure)
           |               |
0x80000000 +---------------+

             Trusted DRAM
0x08000000 +---------------+
           |   HW_CONFIG   |
0x07f00000 |---------------|
           :               :
           |     BL32      |
0x06000000 +---------------+

             Trusted SRAM
0x04040000 +---------------+  loaded by BL2  +-----------------+
           |    BL1 (rw)   |  <<<<<<<<<<<<<  |                 |
           |---------------|  <<<<<<<<<<<<<  |   BL31 NOBITS   |
           |      BL2      |  <<<<<<<<<<<<<  |                 |
           |---------------|  <<<<<<<<<<<<<  |-----------------|
           |               |  <<<<<<<<<<<<<  | BL31 PROGBITS   |
           |               |                 +-----------------+
0x04003000 +---------------+
           |     CONFIG    |
0x04001000 +---------------+
           |     Shared    |
0x04000000 +---------------+

             Trusted ROM
0x04000000 +---------------+
           |    BL1 (ro)   |
0x00000000 +---------------+
```

FVP with TSP in TZC-Secured DRAM with firmware configs :

```
               DRAM
0xffffffff +----------+
           |  EL3 TZC |
0xffe00000 |----------|  (secure)
           |  AP TZC  |
           |  (BL32)  |
0xff000000 +----------+
           |          |
0x82100000 |----------|
           |HW_CONFIG |
0x82000000 |----------|  (non-secure)
           |          |
0x80000000 +----------+

           Trusted DRAM
0x08000000 +----------+
```

(continues on next page)

(continued from previous page)

```
           |HW_CONFIG |
0x7f000000 |----------|
           :          :
           |          |
0x06000000 +----------+

           Trusted SRAM
0x04040000 +----------+  loaded by BL2  +----------------+
           | BL1 (rw) |  <<<<<<<<<<<<<  |                |
           |----------|  <<<<<<<<<<<<<  |  BL31 NOBITS   |
           |   BL2    |  <<<<<<<<<<<<<  |                |
           |----------|  <<<<<<<<<<<<<  |----------------|
           |          |  <<<<<<<<<<<<<  | BL31 PROGBITS  |
           |          |                 +----------------+
0x04003000 +----------+
           |  CONFIG  |
0x04001000 +----------+
           |  Shared  |
0x04000000 +----------+

           Trusted ROM
0x04000000 +----------+
           | BL1 (ro) |
0x00000000 +----------+
```

Juno with BL32 in Trusted SRAM :

```
              DRAM
0xFFFFFFFF +----------+
           |  SCP TZC |
0xFFE00000 |----------|
           |  EL3 TZC |
0xFFC00000 |----------|  (secure)
           |  AP TZC  |
0xFF000000 +----------+
           |          |
           :          :  (non-secure)
           |          |
0x80000000 +----------+


              Flash0
0x0C000000 +----------+
           :          :
0x0BED0000 |----------|
           | BL1 (ro) |
0x0BEC0000 |----------|
           :          :
0x08000000 +----------+                  BL31 is loaded
                                         after SCP_BL2 has
           Trusted SRAM                  been sent to SCP
0x04040000 +----------+  loaded by BL2  +----------------+
```

(continues on next page)

(continued from previous page)

```
            | BL1 (rw) |  <<<<<<<<<<<<<  |                |
            |----------|  <<<<<<<<<<<<<  |  BL31 NOBITS   |
            |   BL2    |  <<<<<<<<<<<<<  |                |
            |----------|  <<<<<<<<<<<<<  |----------------|
            | SCP_BL2  |  <<<<<<<<<<<<<  | BL31 PROGBITS  |
            |          |  <<<<<<<<<<<<<  |----------------|
            |          |  <<<<<<<<<<<<<  |     BL32       |
            |          |                 +----------------+
            |          |
0x04001000 +----------+
            |   MHU    |
0x04000000 +----------+
```

Juno with BL32 in TZC-secured DRAM :

```
               DRAM
0xFFFFFFFF +----------+
            |  SCP TZC |
0xFFE00000 |----------|
            |  EL3 TZC |
0xFFC00000 |----------|  (secure)
            |  AP TZC  |
            |  (BL32)  |
0xFF000000 +----------+
            |          |
            :          :  (non-secure)
            |          |
0x80000000 +----------+

              Flash0
0x0C000000 +----------+
            :          :
0x0BED0000 |----------|
            | BL1 (ro) |
0x0BEC0000 |----------|
            :          :
0x08000000 +----------+                  BL31 is loaded
                                          after SCP_BL2 has
            Trusted SRAM                  been sent to SCP
0x04040000 +----------+  loaded by BL2  +----------------+
            | BL1 (rw) |  <<<<<<<<<<<<<  |                |
            |----------|  <<<<<<<<<<<<<  |  BL31 NOBITS   |
            |   BL2    |  <<<<<<<<<<<<<  |                |
            |----------|  <<<<<<<<<<<<<  |----------------|
            | SCP_BL2  |  <<<<<<<<<<<<<  | BL31 PROGBITS  |
            |          |                 +----------------+
0x04001000 +----------+
            |   MHU    |
0x04000000 +----------+
```

5.4.13 Firmware Image Package (FIP)

Using a Firmware Image Package (FIP) allows for packing bootloader images (and potentially other payloads) into a single archive that can be loaded by TF-A from non-volatile platform storage. A driver to load images from a FIP has been added to the storage layer and allows a package to be read from supported platform storage. A tool to create Firmware Image Packages is also provided and described below.

Firmware Image Package layout

The FIP layout consists of a table of contents (ToC) followed by payload data. The ToC itself has a header followed by one or more table entries. The ToC is terminated by an end marker entry, and since the size of the ToC is 0 bytes, the offset equals the total size of the FIP file. All ToC entries describe some payload data that has been appended to the end of the binary package. With the information provided in the ToC entry the corresponding payload data can be retrieved.

```
------------------
| ToC Header     |
|----------------|
| ToC Entry 0    |
|----------------|
| ToC Entry 1    |
|----------------|
| ToC End Marker |
|----------------|
|                |
|     Data 0     |
|                |
|----------------|
|                |
|     Data 1     |
|                |
------------------
```

The ToC header and entry formats are described in the header file `include/tools_share/firmware_image_package.h`. This file is used by both the tool and TF-A.

The ToC header has the following fields:

```
`name`: The name of the ToC. This is currently used to validate the header.
`serial_number`: A non-zero number provided by the creation tool
`flags`: Flags associated with this data.
    Bits 0-31: Reserved
    Bits 32-47: Platform defined
    Bits 48-63: Reserved
```

A ToC entry has the following fields:

```
`uuid`: All files are referred to by a pre-defined Universally Unique
    IDentifier [UUID] . The UUIDs are defined in
    `include/tools_share/firmware_image_package.h`. The platform translates
```

(continues on next page)

(continued from previous page)

```
    the requested image name into the corresponding UUID when accessing the
    package.
`offset_address`: The offset address at which the corresponding payload data
    can be found. The offset is calculated from the ToC base address.
`size`: The size of the corresponding payload data in bytes.
`flags`: Flags associated with this entry. None are yet defined.
```

Firmware Image Package creation tool

The FIP creation tool can be used to pack specified images into a binary package that can be loaded by TF-A from platform storage. The tool currently only supports packing bootloader images. Additional image definitions can be added to the tool as required.

The tool can be found in `tools/fiptool`.

Loading from a Firmware Image Package (FIP)

The Firmware Image Package (FIP) driver can load images from a binary package on non-volatile platform storage. For the Arm development platforms, this is currently NOR FLASH.

Bootloader images are loaded according to the platform policy as specified by the function `plat_get_image_source()`. For the Arm development platforms, this means the platform will attempt to load images from a Firmware Image Package located at the start of NOR FLASH0.

The Arm development platforms' policy is to only allow loading of a known set of images. The platform policy can be modified to allow additional images.

5.4.14 Use of coherent memory in TF-A

There might be loss of coherency when physical memory with mismatched shareability, cacheability and memory attributes is accessed by multiple CPUs (refer to section B2.9 of Arm ARM for more details). This possibility occurs in TF-A during power up/down sequences when coherency, MMU and caches are turned on/off incrementally.

TF-A defines coherent memory as a region of memory with Device nGnRE attributes in the translation tables. The translation granule size in TF-A is 4KB. This is the smallest possible size of the coherent memory region.

By default, all data structures which are susceptible to accesses with mismatched attributes from various CPUs are allocated in a coherent memory region (refer to section 2.1 of *Porting Guide*). The coherent memory region accesses are Outer Shareable, non-cacheable and they can be accessed with the Device nGnRE attributes when the MMU is turned on. Hence, at the expense of at least an extra page of memory, TF-A is able to work around coherency issues due to mismatched memory attributes.

The alternative to the above approach is to allocate the susceptible data structures in Normal WriteBack WriteAllocate Inner shareable memory. This approach requires the data structures to be designed so that it is possible to work around the issue of mismatched memory attributes by performing software cache maintenance on them.

Disabling the use of coherent memory in TF-A

It might be desirable to avoid the cost of allocating coherent memory on platforms which are memory constrained. TF-A enables inclusion of coherent memory in firmware images through the build flag `USE_COHERENT_MEM`. This flag is enabled by default. It can be disabled to choose the second approach described above.

The below sections analyze the data structures allocated in the coherent memory region and the changes required to allocate them in normal memory.

Coherent memory usage in PSCI implementation

The `psci_non_cpu_pd_nodes` data structure stores the platform's power domain tree information for state management of power domains. By default, this data structure is allocated in the coherent memory region in TF-A because it can be accessed by multiple CPUs, either with caches enabled or disabled.

```
typedef struct non_cpu_pwr_domain_node {
    /*
     * Index of the first CPU power domain node level 0 which has this node
     * as its parent.
     */
    unsigned int cpu_start_idx;

    /*
     * Number of CPU power domains which are siblings of the domain indexed
     * by 'cpu_start_idx' i.e. all the domains in the range 'cpu_start_idx
     * -> cpu_start_idx + ncpus' have this node as their parent.
     */
    unsigned int ncpus;

    /*
     * Index of the parent power domain node.
     */
    unsigned int parent_node;

    plat_local_state_t local_state;

    unsigned char level;

    /* For indexing the psci_lock array*/
    unsigned char lock_index;
} non_cpu_pd_node_t;
```

In order to move this data structure to normal memory, the use of each of its fields must be analyzed. Fields like `cpu_start_idx`, `ncpus`, `parent_node` `level` and `lock_index` are only written once during cold boot. Hence removing them from coherent memory involves only doing a clean and invalidate of the cache lines after these fields are written.

The field `local_state` can be concurrently accessed by multiple CPUs in different cache states. A Lamport's Bakery lock `psci_locks` is used to ensure mutual exclusion to this field and a clean and invalidate is needed after it is written.

Bakery lock data

The bakery lock data structure `bakery_lock_t` is allocated in coherent memory and is accessed by multiple CPUs with mismatched attributes. `bakery_lock_t` is defined as follows:

```
typedef struct bakery_lock {
    /*
     * The lock_data is a bit-field of 2 members:
     * Bit[0]       : choosing. This field is set when the CPU is
     *                choosing its bakery number.
     * Bits[1 - 15] : number. This is the bakery number allocated.
     */
    volatile uint16_t lock_data[BAKERY_LOCK_MAX_CPUS];
} bakery_lock_t;
```

It is a characteristic of Lamport's Bakery algorithm that the volatile per-CPU fields can be read by all CPUs but only written to by the owning CPU.

Depending upon the data cache line size, the per-CPU fields of the `bakery_lock_t` structure for multiple CPUs may exist on a single cache line. These per-CPU fields can be read and written during lock contention by multiple CPUs with mismatched memory attributes. Since these fields are a part of the lock implementation, they do not have access to any other locking primitive to safeguard against the resulting coherency issues. As a result, simple software cache maintenance is not enough to allocate them in coherent memory. Consider the following example.

CPU0 updates its per-CPU field with data cache enabled. This write updates a local cache line which contains a copy of the fields for other CPUs as well. Now CPU1 updates its per-CPU field of the `bakery_lock_t` structure with data cache disabled. CPU1 then issues a DCIVAC operation to invalidate any stale copies of its field in any other cache line in the system. This operation will invalidate the update made by CPU0 as well.

To use bakery locks when `USE_COHERENT_MEM` is disabled, the lock data structure has been redesigned. The changes utilise the characteristic of Lamport's Bakery algorithm mentioned earlier. The bakery_lock structure only allocates the memory for a single CPU. The macro `DEFINE_BAKERY_LOCK` allocates all the bakery locks needed for a CPU into a section `.bakery_lock`. The linker allocates the memory for other cores by using the total size allocated for the bakery_lock section and multiplying it with (PLATFORM_CORE_COUNT - 1). This enables software to perform software cache maintenance on the lock data structure without running into coherency issues associated with mismatched attributes.

The bakery lock data structure `bakery_info_t` is defined for use when `USE_COHERENT_MEM` is disabled as follows:

```
typedef struct bakery_info {
    /*
     * The lock_data is a bit-field of 2 members:
     * Bit[0]       : choosing. This field is set when the CPU is
     *                choosing its bakery number.
     * Bits[1 - 15] : number. This is the bakery number allocated.
     */
     volatile uint16_t lock_data;
} bakery_info_t;
```

The `bakery_info_t` represents a single per-CPU field of one lock and the combination of corresponding `bakery_info_t` structures for all CPUs in the system represents the complete bakery lock. The view in memory for a system with n bakery locks are:

```
.bakery_lock section start
|----------------|
| `bakery_info_t`| <-- Lock_0 per-CPU field
|    Lock_0      |     for CPU0
|----------------|
| `bakery_info_t`| <-- Lock_1 per-CPU field
|    Lock_1      |     for CPU0
|----------------|
| ....           |
|----------------|
| `bakery_info_t`| <-- Lock_N per-CPU field
|    Lock_N      |     for CPU0
------------------
|    XXXXX       |
| Padding to     |
| next Cache WB  | <--- Calculate PERCPU_BAKERY_LOCK_SIZE, allocate
|  Granule       |       continuous memory for remaining CPUs.
------------------
| `bakery_info_t`| <-- Lock_0 per-CPU field
|    Lock_0      |     for CPU1
|----------------|
| `bakery_info_t`| <-- Lock_1 per-CPU field
|    Lock_1      |     for CPU1
|----------------|
| ....           |
|----------------|
| `bakery_info_t`| <-- Lock_N per-CPU field
|    Lock_N      |     for CPU1
------------------
|    XXXXX       |
| Padding to     |
| next Cache WB  |
|  Granule       |
------------------
```

Consider a system of 2 CPUs with 'N' bakery locks as shown above. For an operation on Lock_N, the corresponding `bakery_info_t` in both CPU0 and CPU1 `.bakery_lock` section need to be fetched and appropriate cache operations need to be performed for each access.

On Arm Platforms, bakery locks are used in psci (`psci_locks`) and power controller driver (`arm_lock`).

Non Functional Impact of removing coherent memory

Removal of the coherent memory region leads to the additional software overhead of performing cache maintenance for the affected data structures. However, since the memory where the data structures are allocated is cacheable, the overhead is mostly mitigated by an increase in performance.

There is however a performance impact for bakery locks, due to:

- Additional cache maintenance operations, and
- Multiple cache line reads for each lock operation, since the bakery locks for each CPU are distributed across different cache lines.

The implementation has been optimized to minimize this additional overhead. Measurements indicate that when bakery locks are allocated in Normal memory, the minimum latency of acquiring a lock is on an average 3-4 micro seconds whereas in Device memory the same is 2 micro seconds. The measurements were done on the Juno Arm development platform.

As mentioned earlier, almost a page of memory can be saved by disabling `USE_COHERENT_MEM`. Each platform needs to consider these trade-offs to decide whether coherent memory should be used. If a platform disables `USE_COHERENT_MEM` and needs to use bakery locks in the porting layer, it can optionally define macro `PLAT_PERCPU_BAKERY_LOCK_SIZE` (see the *Porting Guide*). Refer to the reference platform code for examples.

5.4.15 Isolating code and read-only data on separate memory pages

In the Armv8-A VMSA, translation table entries include fields that define the properties of the target memory region, such as its access permissions. The smallest unit of memory that can be addressed by a translation table entry is a memory page. Therefore, if software needs to set different permissions on two memory regions then it needs to map them using different memory pages.

The default memory layout for each BL image is as follows:

```
|        ...        |
+-------------------+
|  Read-write data  |
+-------------------+ Page boundary
|     <Padding>     |
+-------------------+
| Exception vectors |
+-------------------+ 2 KB boundary
|     <Padding>     |
+-------------------+
|  Read-only data   |
+-------------------+
|       Code        |
+-------------------+ BLx_BASE
```

Note: The 2KB alignment for the exception vectors is an architectural requirement.

The read-write data start on a new memory page so that they can be mapped with read-write permissions, whereas the code and read-only data below are configured as read-only.

However, the read-only data are not aligned on a page boundary. They are contiguous to the code. Therefore, the end of the code section and the beginning of the read-only data one might share a memory page. This forces both to be mapped with the same memory attributes. As the code needs to be executable, this means that the read-only data stored on the same memory page as the code are executable as well. This could potentially be exploited as part of a security attack.

TF provides the build flag `SEPARATE_CODE_AND_RODATA` to isolate the code and read-only data on separate memory pages. This in turn allows independent control of the access permissions for the code and read-only data. In this case, platform code gets a finer-grained view of the image layout and can appropriately map the code region as executable and the read-only data as execute-never.

This has an impact on memory footprint, as padding bytes need to be introduced between the code and read-only data to ensure the segregation of the two. To limit the memory cost, this flag also changes the memory layout such that the code and exception vectors are now contiguous, like so:

```
|         ...        |
+--------------------+
|  Read-write data   |
+--------------------+ Page boundary
|      <Padding>     |
+--------------------+
|  Read-only data    |
+--------------------+ Page boundary
|      <Padding>     |
+--------------------+
| Exception vectors  |
+--------------------+ 2 KB boundary
|      <Padding>     |
+--------------------+
|        Code        |
+--------------------+ BLx_BASE
```

With this more condensed memory layout, the separation of read-only data will add zero or one page to the memory footprint of each BL image. Each platform should consider the trade-off between memory footprint and security.

This build flag is disabled by default, minimising memory footprint. On Arm platforms, it is enabled.

5.4.16 Publish and Subscribe Framework

The Publish and Subscribe Framework allows EL3 components to define and publish events, to which other EL3 components can subscribe.

The following macros are provided by the framework:

- `REGISTER_PUBSUB_EVENT(event)`: Defines an event, and takes one argument, the event name, which must be a valid C identifier. All calls to `REGISTER_PUBSUB_EVENT` macro must be placed in the file `pubsub_events.h`.

- `PUBLISH_EVENT_ARG(event, arg)`: Publishes a defined event, by iterating subscribed handlers and calling them in turn. The handlers will be passed the parameter `arg`. The expected use-case is to broadcast an event.
- `PUBLISH_EVENT(event)`: Like `PUBLISH_EVENT_ARG`, except that the value `NULL` is passed to subscribed handlers.
- `SUBSCRIBE_TO_EVENT(event, handler)`: Registers the `handler` to subscribe to `event`. The handler will be executed whenever the `event` is published.
- `for_each_subscriber(event, subscriber)`: Iterates through all handlers subscribed for `event`. `subscriber` must be a local variable of type `pubsub_cb_t *`, and will point to each subscribed handler in turn during iteration. This macro can be used for those patterns that none of the `PUBLISH_EVENT_*()` macros cover.

Publishing an event that wasn't defined using `REGISTER_PUBSUB_EVENT` will result in build error. Subscribing to an undefined event however won't.

Subscribed handlers must be of type `pubsub_cb_t`, with following function signature:

```
typedef void* (*pubsub_cb_t)(const void *arg);
```

There may be arbitrary number of handlers registered to the same event. The order in which subscribed handlers are notified when that event is published is not defined. Subscribed handlers may be executed in any order; handlers should not assume any relative ordering amongst them.

Publishing an event on a PE will result in subscribed handlers executing on that PE only; it won't cause handlers to execute on a different PE.

Note that publishing an event on a PE blocks until all the subscribed handlers finish executing on the PE.

TF-A generic code publishes and subscribes to some events within. Platform ports are discouraged from subscribing to them. These events may be withdrawn, renamed, or have their semantics altered in the future. Platforms may however register, publish, and subscribe to platform-specific events.

Publish and Subscribe Example

A publisher that wants to publish event `foo` would:

- Define the event `foo` in the `pubsub_events.h`.

  ```
  REGISTER_PUBSUB_EVENT(foo);
  ```

- Depending on the nature of event, use one of `PUBLISH_EVENT_*()` macros to publish the event at the appropriate path and time of execution.

A subscriber that wants to subscribe to event `foo` published above would implement:

```
void *foo_handler(const void *arg)
{
     void *result;

     /* Do handling ... */
```

(continues on next page)

(continued from previous page)

```
    return result;
}

SUBSCRIBE_TO_EVENT(foo, foo_handler);
```

Reclaiming the BL31 initialization code

A significant amount of the code used for the initialization of BL31 is never needed again after boot time. In order to reduce the runtime memory footprint, the memory used for this code can be reclaimed after initialization has finished and be used for runtime data.

The build option `RECLAIM_INIT_CODE` can be set to mark this boot time code with a `.text.init.*` attribute which can be filtered and placed suitably within the BL image for later reclamation by the platform. The platform can specify the filter and the memory region for this init section in BL31 via the plat.ld.S linker script. For example, on the FVP, this section is placed overlapping the secondary CPU stacks so that after the cold boot is done, this memory can be reclaimed for the stacks. The init memory section is initially mapped with `RO`, `EXECUTE` attributes. After BL31 initialization has completed, the FVP changes the attributes of this section to `RW`, `EXECUTE_NEVER` allowing it to be used for runtime data. The memory attributes are changed within the `bl31_plat_runtime_setup` platform hook. The init section section can be reclaimed for any data which is accessed after cold boot initialization and it is upto the platform to make the decision.

Please note that this will disable inlining for any functions with the __init attribute.

5.4.17 Performance Measurement Framework

The Performance Measurement Framework (PMF) facilitates collection of timestamps by registered services and provides interfaces to retrieve them from within TF-A. A platform can choose to expose appropriate SMCs to retrieve these collected timestamps.

By default, the global physical counter is used for the timestamp value and is read via `CNTPCT_EL0`. The framework allows to retrieve timestamps captured by other CPUs.

Timestamp identifier format

A PMF timestamp is uniquely identified across the system via the timestamp ID or `tid`. The `tid` is composed as follows:

```
Bits 0-7: The local timestamp identifier.
Bits 8-9: Reserved.
Bits 10-15: The service identifier.
Bits 16-31: Reserved.
```

1. The service identifier. Each PMF service is identified by a service name and a service identifier. Both the service name and identifier are unique within the system as a whole.
2. The local timestamp identifier. This identifier is unique within a given service.

Registering a PMF service

To register a PMF service, the `PMF_REGISTER_SERVICE()` macro from `pmf.h` is used. The arguments required are the service name, the service ID, the total number of local timestamps to be captured and a set of flags.

The `flags` field can be specified as a bitwise-OR of the following values:

```
PMF_STORE_ENABLE: The timestamp is stored in memory for later retrieval.
PMF_DUMP_ENABLE: The timestamp is dumped on the serial console.
```

The `PMF_REGISTER_SERVICE()` reserves memory to store captured timestamps in a PMF specific linker section at build time. Additionally, it defines necessary functions to capture and retrieve a particular timestamp for the given service at runtime.

The macro `PMF_REGISTER_SERVICE()` only enables capturing PMF timestamps from within TF-A. In order to retrieve timestamps from outside of TF-A, the `PMF_REGISTER_SERVICE_SMC()` macro must be used instead. This macro accepts the same set of arguments as the `PMF_REGISTER_SERVICE()` macro but additionally supports retrieving timestamps using SMCs.

Capturing a timestamp

PMF timestamps are stored in a per-service timestamp region. On a system with multiple CPUs, each timestamp is captured and stored in a per-CPU cache line aligned memory region.

Having registered the service, the `PMF_CAPTURE_TIMESTAMP()` macro can be used to capture a timestamp at the location where it is used. The macro takes the service name, a local timestamp identifier and a flag as arguments.

The `flags` field argument can be zero, or `PMF_CACHE_MAINT` which instructs PMF to do cache maintenance following the capture. Cache maintenance is required if any of the service's timestamps are captured with data cache disabled.

To capture a timestamp in assembly code, the caller should use `pmf_calc_timestamp_addr` macro (defined in `pmf_asm_macros.S`) to calculate the address of where the timestamp would be stored. The caller should then read `CNTPCT_EL0` register to obtain the timestamp and store it at the determined address for later retrieval.

Retrieving a timestamp

From within TF-A, timestamps for individual CPUs can be retrieved using either `PMF_GET_TIMESTAMP_BY_MPIDR()` or `PMF_GET_TIMESTAMP_BY_INDEX()` macros. These macros accept the CPU's MPIDR value, or its ordinal position respectively.

From outside TF-A, timestamps for individual CPUs can be retrieved by calling into `pmf_smc_handler()`.

```
Interface : pmf_smc_handler()
Argument  : unsigned int smc_fid, u_register_t x1,
            u_register_t x2, u_register_t x3,
```

(continues on next page)

(continued from previous page)

```
            u_register_t x4, void *cookie,
            void *handle, u_register_t flags
Return    : uintptr_t

smc_fid: Holds the SMC identifier which is either `PMF_SMC_GET_TIMESTAMP_32`
    when the caller of the SMC is running in AArch32 mode
    or `PMF_SMC_GET_TIMESTAMP_64` when the caller is running in AArch64 mode.
x1: Timestamp identifier.
x2: The `mpidr` of the CPU for which the timestamp has to be retrieved.
    This can be the `mpidr` of a different core to the one initiating
    the SMC.  In that case, service specific cache maintenance may be
    required to ensure the updated copy of the timestamp is returned.
x3: A flags value that is either 0 or `PMF_CACHE_MAINT`.  If
    `PMF_CACHE_MAINT` is passed, then the PMF code will perform a
    cache invalidate before reading the timestamp.  This ensures
    an updated copy is returned.
```

The remaining arguments, `x4`, `cookie`, `handle` and `flags` are unused in this implementation.

PMF code structure

1. `pmf_main.c` consists of core functions that implement service registration, initialization, storing, dumping and retrieving timestamps.
2. `pmf_smc.c` contains the SMC handling for registered PMF services.
3. `pmf.h` contains the public interface to Performance Measurement Framework.
4. `pmf_asm_macros.S` consists of macros to facilitate capturing timestamps in assembly code.
5. `pmf_helpers.h` is an internal header used by `pmf.h`.

5.4.18 Armv8-A Architecture Extensions

TF-A makes use of Armv8-A Architecture Extensions where applicable. This section lists the usage of Architecture Extensions, and build flags controlling them.

Build options

`ARM_ARCH_MAJOR` and `ARM_ARCH_MINOR`

These build options serve dual purpose

- Determine the architecture extension support in TF-A build: All the mandatory architectural features up to `ARM_ARCH_MAJOR.ARM_ARCH_MINOR` are included and unconditionally enabled by TF-A build system.
- `ARM_ARCH_MAJOR` and `ARM_ARCH_MINOR` are passed to a march.mk build utility this will try to come up with an appropriate -march value to be passed to compiler by probing the compiler and

checking what's supported by the compiler and what's best that can be used. But if platform provides a `MARCH_DIRECTIVE` then it will used directly and compiler probing will be skipped. Use `ARM_ARCH_FEATURE` to provide a list of optional features that can be relied upon.

The build system requires that the platform provides a valid numeric value based on CPU architecture extension, otherwise it defaults to base Armv8.0-A architecture. Subsequent Arm Architecture versions also support extensions which were introduced in previous versions.

See also:

Build Options

For details on the Architecture Extension and available features, please refer to the respective Architecture Extension Supplement.

Armv8.1-A

This Architecture Extension is targeted when `ARM_ARCH_MAJOR` >= 8, or when `ARM_ARCH_MAJOR` == 8 and `ARM_ARCH_MINOR` >= 1.

- By default, a load-/store-exclusive instruction pair is used to implement spinlocks. The `USE_SPINLOCK_CAS` build option when set to 1 selects the spinlock implementation using the ARMv8.1-LSE Compare and Swap instruction. Notice this instruction is only available in AArch64 execution state, so the option is only available to AArch64 builds.

Armv8.2-A

- The presence of ARMv8.2-TTCNP is detected at runtime. When it is present, the Common not Private (TTBRn_ELx.CnP) bit is enabled to indicate that multiple Processing Elements in the same Inner Shareable domain use the same translation table entries for a given stage of translation for a particular translation regime.

Armv8.3-A

- Pointer authentication features of Armv8.3-A are unconditionally enabled in the Non-secure world so that lower ELs are allowed to use them without causing a trap to EL3.

  In order to enable the Secure world to use it, `CTX_INCLUDE_PAUTH_REGS` must be set to 1. This will add all pointer authentication system registers to the context that is saved when doing a world switch.

  The TF-A itself has support for pointer authentication at runtime that can be enabled by setting `BRANCH_PROTECTION` option to non-zero and `CTX_INCLUDE_PAUTH_REGS` to 1. This enables pointer authentication in BL1, BL2, BL31, and the TSP if it is used.

  Note that Pointer Authentication is enabled for Non-secure world irrespective of the value of these build flags if the CPU supports it.

  If `ARM_ARCH_MAJOR == 8` and `ARM_ARCH_MINOR >= 3` the code footprint of enabling PAuth is lower because the compiler will use the optimized PAuth instructions rather than the backwards-compatible ones.

Armv8.5-A

- Branch Target Identification feature is selected by `BRANCH_PROTECTION` option set to 1. This option defaults to 0.
- Memory Tagging Extension feature has few variants but not all of them require enablement from EL3 to be used at lower EL. e.g. Memory tagging only at EL0(MTE) does not require EL3 configuration however memory tagging at EL2/EL1 (MTE2) does require EL3 enablement and we need to set this option `ENABLE_FEAT_MTE2` to 1. This option defaults to 0.

Armv7-A

This Architecture Extension is targeted when `ARM_ARCH_MAJOR` == 7.

There are several Armv7-A extensions available. Obviously the TrustZone extension is mandatory to support the TF-A bootloader and runtime services.

Platform implementing an Armv7-A system can to define from its target Cortex-A architecture through `ARM_CORTEX_A<X> = yes` in their `platform.mk` script. For example `ARM_CORTEX_A15=yes` for a Cortex-A15 target.

Platform can also set `ARM_WITH_NEON=yes` to enable neon support. Note that using neon at runtime has constraints on non secure world context. TF-A does not yet provide VFP context management.

Directive `ARM_CORTEX_A<x>` and `ARM_WITH_NEON` are used to set the toolchain target architecture directive.

Platform may choose to not define straight the toolchain target architecture directive by defining `MARCH_DIRECTIVE`. I.e:

```
MARCH_DIRECTIVE := -march=armv7-a
```

5.4.19 Code Structure

TF-A code is logically divided between the three boot loader stages mentioned in the previous sections. The code is also divided into the following categories (present as directories in the source code):

- **Platform specific.** Choice of architecture specific code depends upon the platform.
- **Common code.** This is platform and architecture agnostic code.
- **Library code.** This code comprises of functionality commonly used by all other code. The PSCI implementation and other EL3 runtime frameworks reside as Library components.
- **Stage specific.** Code specific to a boot stage.
- **Drivers.**
- **Services.** EL3 runtime services (eg: SPD). Specific SPD services reside in the `services/spd` directory (e.g. `services/spd/tspd`).

Each boot loader stage uses code from one or more of the above mentioned categories. Based upon the above, the code layout looks like this:

```
Directory     Used by BL1?     Used by BL2?     Used by BL31?
bl1           Yes              No               No
bl2           No               Yes              No
bl31          No               No               Yes
plat          Yes              Yes              Yes
drivers       Yes              No               Yes
common        Yes              Yes              Yes
lib           Yes              Yes              Yes
services      No               No               Yes
```

The build system provides a non configurable build option IMAGE_BLx for each boot loader stage (where x = BL stage). e.g. for BL1 , IMAGE_BL1 will be defined by the build system. This enables TF-A to compile certain code only for specific boot loader stages

All assembler files have the `.S` extension. The linker source files for each boot stage have the extension `.ld.S`. These are processed by GCC to create the linker scripts which have the extension `.ld`.

FDTs provide a description of the hardware platform and are used by the Linux kernel at boot time. These can be found in the `fdts` directory.

References

- Trusted Board Boot Requirements CLIENT (TBBR-CLIENT) Armv8-A (ARM DEN0006D)
- PSCI
- SMC Calling Convention
- *Interrupt Management Framework*

5.5 Interrupt Management Framework

This framework is responsible for managing interrupts routed to EL3. It also allows EL3 software to configure the interrupt routing behavior. Its main objective is to implement the following two requirements.

1. It should be possible to route interrupts meant to be handled by secure software (Secure interrupts) to EL3, when execution is in non-secure state (normal world). The framework should then take care of handing control of the interrupt to either software in EL3 or Secure-EL1 depending upon the software configuration and the GIC implementation. This requirement ensures that secure interrupts are under the control of the secure software with respect to their delivery and handling without the possibility of intervention from non-secure software.
2. It should be possible to route interrupts meant to be handled by non-secure software (Non-secure interrupts) to the last executed exception level in the normal world when the execution is in secure world at

exception levels lower than EL3. This could be done with or without the knowledge of software executing in Secure-EL1/Secure-EL0. The choice of approach should be governed by the secure software. This requirement ensures that non-secure software is able to execute in tandem with the secure software without overriding it.

5.5.1 Concepts

Interrupt types

The framework categorises an interrupt to be one of the following depending upon the exception level(s) it is handled in.

1. Secure EL1 interrupt. This type of interrupt can be routed to EL3 or Secure-EL1 depending upon the security state of the current execution context. It is always handled in Secure-EL1.
2. Non-secure interrupt. This type of interrupt can be routed to EL3, Secure-EL1, Non-secure EL1 or EL2 depending upon the security state of the current execution context. It is always handled in either Non-secure EL1 or EL2.
3. EL3 interrupt. This type of interrupt can be routed to EL3 or Secure-EL1 depending upon the security state of the current execution context. It is always handled in EL3.

The following constants define the various interrupt types in the framework implementation.

```
#define INTR_TYPE_S_EL1      0
#define INTR_TYPE_EL3        1
#define INTR_TYPE_NS         2
```

Routing model

A type of interrupt can be either generated as an FIQ or an IRQ. The target exception level of an interrupt type is configured through the FIQ and IRQ bits in the Secure Configuration Register at EL3 (`SCR_EL3.FIQ` and `SCR_EL3.IRQ` bits). When `SCR_EL3.FIQ`=1, FIQs are routed to EL3. Otherwise they are routed to the First Exception Level (FEL) capable of handling interrupts. When `SCR_EL3.IRQ`=1, IRQs are routed to EL3. Otherwise they are routed to the FEL. This register is configured independently by EL3 software for each security state prior to entry into a lower exception level in that security state.

A routing model for a type of interrupt (generated as FIQ or IRQ) is defined as its target exception level for each security state. It is represented by a single bit for each security state. A value of `0` means that the interrupt should be routed to the FEL. A value of `1` means that the interrupt should be routed to EL3. A routing model is applicable only when execution is not in EL3.

The default routing model for an interrupt type is to route it to the FEL in either security state.

Valid routing models

The framework considers certain routing models for each type of interrupt to be incorrect as they conflict with the requirements mentioned in Section 1. The following sub-sections describe all the possible routing models and specify which ones are valid or invalid. EL3 interrupts are currently supported only for GIC version 3.0 (Arm GICv3) and only the Secure-EL1 and Non-secure interrupt types are supported for GIC version 2.0 (Arm GICv2) (see *Assumptions in Interrupt Management Framework*). The terminology used in the following sub-sections is explained below.

1. **CSS**. Current Security State. `0` when secure and `1` when non-secure
2. **TEL3**. Target Exception Level 3. `0` when targeted to the FEL. `1` when targeted to EL3.

Secure-EL1 interrupts

1. **CSS=0, TEL3=0**. Interrupt is routed to the FEL when execution is in secure state. This is a valid routing model as secure software is in control of handling secure interrupts.
2. **CSS=0, TEL3=1**. Interrupt is routed to EL3 when execution is in secure state. This is a valid routing model as secure software in EL3 can handover the interrupt to Secure-EL1 for handling.
3. **CSS=1, TEL3=0**. Interrupt is routed to the FEL when execution is in non-secure state. This is an invalid routing model as a secure interrupt is not visible to the secure software which violates the motivation behind the Arm Security Extensions.
4. **CSS=1, TEL3=1**. Interrupt is routed to EL3 when execution is in non-secure state. This is a valid routing model as secure software in EL3 can handover the interrupt to Secure-EL1 for handling.

Non-secure interrupts

1. **CSS=0, TEL3=0**. Interrupt is routed to the FEL when execution is in secure state. This allows the secure software to trap non-secure interrupts, perform its book-keeping and hand the interrupt to the non-secure software through EL3. This is a valid routing model as secure software is in control of how its execution is preempted by non-secure interrupts.
2. **CSS=0, TEL3=1**. Interrupt is routed to EL3 when execution is in secure state. This is a valid routing model as secure software in EL3 can save the state of software in Secure-EL1/Secure-EL0 before handing the interrupt to non-secure software. This model requires additional coordination between Secure-EL1 and EL3 software to ensure that the former's state is correctly saved by the latter.
3. **CSS=1, TEL3=0**. Interrupt is routed to FEL when execution is in non-secure state. This is a valid routing model as a non-secure interrupt is handled by non-secure software.
4. **CSS=1, TEL3=1**. Interrupt is routed to EL3 when execution is in non-secure state. This is an invalid routing model as there is no valid reason to route the interrupt to EL3 software and then hand it back to non-secure software for handling.

EL3 interrupts

1. **CSS=0, TEL3=0**. Interrupt is routed to the FEL when execution is in Secure-EL1/Secure-EL0. This is a valid routing model as secure software in Secure-EL1/Secure-EL0 is in control of how its execution is preempted by EL3 interrupt and can handover the interrupt to EL3 for handling.

   However, when `EL3_EXCEPTION_HANDLING` is `1`, this routing model is invalid as EL3 interrupts are unconditionally routed to EL3, and EL3 interrupts will always preempt Secure EL1/EL0 execution. See *exception handling* documentation.

2. **CSS=0, TEL3=1**. Interrupt is routed to EL3 when execution is in Secure-EL1/Secure-EL0. This is a valid routing model as secure software in EL3 can handle the interrupt.

3. **CSS=1, TEL3=0**. Interrupt is routed to the FEL when execution is in non-secure state. This is an invalid routing model as a secure interrupt is not visible to the secure software which violates the motivation behind the Arm Security Extensions.

4. **CSS=1, TEL3=1**. Interrupt is routed to EL3 when execution is in non-secure state. This is a valid routing model as secure software in EL3 can handle the interrupt.

Mapping of interrupt type to signal

The framework is meant to work with any interrupt controller implemented by a platform. A interrupt controller could generate a type of interrupt as either an FIQ or IRQ signal to the CPU depending upon the current security state. The mapping between the type and signal is known only to the platform. The framework uses this information to determine whether the IRQ or the FIQ bit should be programmed in `SCR_EL3` while applying the routing model for a type of interrupt. The platform provides this information through the `plat_interrupt_type_to_line()` API (described in the *Porting Guide*). For example, on the FVP port when the platform uses an Arm GICv2 interrupt controller, Secure-EL1 interrupts are signaled through the FIQ signal while Non-secure interrupts are signaled through the IRQ signal. This applies when execution is in either security state.

Effect of mapping of several interrupt types to one signal

It should be noted that if more than one interrupt type maps to a single interrupt signal, and if any one of the interrupt type sets **TEL3=1** for a particular security state, then interrupt signal will be routed to EL3 when in that security state. This means that all the other interrupt types using the same interrupt signal will be forced to the same routing model. This should be borne in mind when choosing the routing model for an interrupt type.

For example, in Arm GICv3, when the execution context is Secure-EL1/ Secure-EL0, both the EL3 and the non secure interrupt types map to the FIQ signal. So if either one of the interrupt type sets the routing model so that **TEL3=1** when **CSS=0**, the FIQ bit in `SCR_EL3` will be programmed to route the FIQ signal to EL3 when executing in Secure-EL1/Secure-EL0, thereby effectively routing the other interrupt type also to EL3.

5.5.2 Assumptions in Interrupt Management Framework

The framework makes the following assumptions to simplify its implementation.

1. Although the framework has support for 2 types of secure interrupts (EL3 and Secure-EL1 interrupt), only interrupt controller architectures like Arm GICv3 has architectural support for EL3 interrupts in the form of Group 0 interrupts. In Arm GICv2, all secure interrupts are assumed to be handled in Secure-EL1. They can be delivered to Secure-EL1 via EL3 but they cannot be handled in EL3.

2. Interrupt exceptions (`PSTATE.I` and `F` bits) are masked during execution in EL3.

3. Interrupt management: the following sections describe how interrupts are managed by the interrupt handling framework. This entails:

   1. Providing an interface to allow registration of a handler and specification of the routing model for a type of interrupt.

   2. Implementing support to hand control of an interrupt type to its registered handler when the interrupt is generated.

Both aspects of interrupt management involve various components in the secure software stack spanning from EL3 to Secure-EL1. These components are described in the section *Software components*. The framework stores information associated with each type of interrupt in the following data structure.

```
typedef struct intr_type_desc {
        interrupt_type_handler_t handler;
        uint32_t flags;
        uint32_t scr_el3[2];
} intr_type_desc_t;
```

The `flags` field stores the routing model for the interrupt type in bits[1:0]. Bit[0] stores the routing model when execution is in the secure state. Bit[1] stores the routing model when execution is in the non-secure state. As mentioned in Section *Routing model*, a value of `0` implies that the interrupt should be targeted to the FEL. A value of `1` implies that it should be targeted to EL3. The remaining bits are reserved and SBZ. The helper macro `set_interrupt_rm_flag()` should be used to set the bits in the `flags` parameter.

The `scr_el3[2]` field also stores the routing model but as a mapping of the model in the `flags` field to the corresponding bit in the `SCR_EL3` for each security state.

The framework also depends upon the platform port to configure the interrupt controller to distinguish between secure and non-secure interrupts. The platform is expected to be aware of the secure devices present in the system and their associated interrupt numbers. It should configure the interrupt controller to enable the secure interrupts, ensure that their priority is always higher than the non-secure interrupts and target them to the primary CPU. It should also export the interface described in the *Porting Guide* to enable handling of interrupts.

In the remainder of this document, for the sake of simplicity a Arm GICv2 system is considered and it is assumed that the FIQ signal is used to generate Secure-EL1 interrupts and the IRQ signal is used to generate non-secure interrupts in either security state. EL3 interrupts are not considered.

5.5.3 Software components

Roles and responsibilities for interrupt management are sub-divided between the following components of software running in EL3 and Secure-EL1. Each component is briefly described below.

1. EL3 Runtime Firmware. This component is common to all ports of TF-A.
2. Secure Payload Dispatcher (SPD) service. This service interfaces with the Secure Payload (SP) software which runs in Secure-EL1/Secure-EL0 and is responsible for switching execution between secure and non-secure states. A switch is triggered by a Secure Monitor Call and it uses the APIs exported by the Context management library to implement this functionality. Switching execution between the two security states is a requirement for interrupt management as well. This results in a significant dependency on the SPD service. TF-A implements an example Test Secure Payload Dispatcher (TSPD) service.

   An SPD service plugs into the EL3 runtime firmware and could be common to some ports of TF-A.
3. Secure Payload (SP). On a production system, the Secure Payload corresponds to a Secure OS which runs in Secure-EL1/Secure-EL0. It interfaces with the SPD service to manage communication with non-secure software. TF-A implements an example secure payload called Test Secure Payload (TSP) which runs only in Secure-EL1.

   A Secure payload implementation could be common to some ports of TF-A, just like the SPD service.

5.5.4 Interrupt registration

This section describes in detail the role of each software component (see *Software components*) during the registration of a handler for an interrupt type.

EL3 runtime firmware

This component declares the following prototype for a handler of an interrupt type.

```
typedef uint64_t (*interrupt_type_handler_t)(uint32_t id,
                                             uint32_t flags,
                                             void *handle,
                                             void *cookie);
```

The `id` is parameter is reserved and could be used in the future for passing the interrupt id of the highest pending interrupt only if there is a foolproof way of determining the id. Currently it contains `INTR_ID_UNAVAILABLE`.

The `flags` parameter contains miscellaneous information as follows.

1. Security state, bit[0]. This bit indicates the security state of the lower exception level when the interrupt was generated. A value of `1` means that it was in the non-secure state. A value of `0` indicates that it was in the secure state. This bit can be used by the handler to ensure that interrupt was generated and routed as per the routing model specified during registration.
2. Reserved, bits[31:1]. The remaining bits are reserved for future use.

The handle parameter points to the cpu_context structure of the current CPU for the security state specified in the flags parameter.

Once the handler routine completes, execution will return to either the secure or non-secure state. The handler routine must return a pointer to cpu_context structure of the current CPU for the target security state. On AArch64, this return value is currently ignored by the caller as the appropriate cpu_context to be used is expected to be set by the handler via the context management library APIs. A portable interrupt handler implementation must set the target context both in the structure pointed to by the returned pointer and via the context management library APIs. The handler should treat all error conditions as critical errors and take appropriate action within its implementation e.g. use assertion failures.

The runtime firmware provides the following API for registering a handler for a particular type of interrupt. A Secure Payload Dispatcher service should use this API to register a handler for Secure-EL1 and optionally for non-secure interrupts. This API also requires the caller to specify the routing model for the type of interrupt.

```
int32_t register_interrupt_type_handler(uint32_t type,
                                        interrupt_type_handler handler,
                                        uint64_t flags);
```

The type parameter can be one of the three interrupt types listed above i.e. INTR_TYPE_S_EL1, INTR_TYPE_NS & INTR_TYPE_EL3. The flags parameter is as described in Section 2.

The function will return 0 upon a successful registration. It will return -EALREADY in case a handler for the interrupt type has already been registered. If the type is unrecognised or the flags or the handler are invalid it will return -EINVAL.

Interrupt routing is governed by the configuration of the SCR_EL3.FIQ/IRQ bits prior to entry into a lower exception level in either security state. The context management library maintains a copy of the SCR_EL3 system register for each security state in the cpu_context structure of each CPU. It exports the following APIs to let EL3 Runtime Firmware program and retrieve the routing model for each security state for the current CPU. The value of SCR_EL3 stored in the cpu_context is used by the el3_exit() function to program the SCR_EL3 register prior to returning from the EL3 exception level.

```
uint32_t cm_get_scr_el3(uint32_t security_state);
void cm_write_scr_el3_bit(uint32_t security_state,
                          uint32_t bit_pos,
                          uint32_t value);
```

cm_get_scr_el3() returns the value of the SCR_EL3 register for the specified security state of the current CPU. cm_write_scr_el3_bit() writes a 0 or 1 to the bit specified by bit_pos. register_interrupt_type_handler() invokes set_routing_model() API which programs the SCR_EL3 according to the routing model using the cm_get_scr_el3() and cm_write_scr_el3_bit() APIs.

It is worth noting that in the current implementation of the framework, the EL3 runtime firmware is responsible for programming the routing model. The SPD is responsible for ensuring that the routing model has been adhered to upon receiving an interrupt.

Secure payload dispatcher

A SPD service is responsible for determining and maintaining the interrupt routing model supported by itself and the Secure Payload. It is also responsible for ferrying interrupts between secure and non-secure software depending upon the routing model. It could determine the routing model at build time or at runtime. It must use this information to register a handler for each interrupt type using the `register_interrupt_type_handler()` API in EL3 runtime firmware.

If the routing model is not known to the SPD service at build time, then it must be provided by the SP as the result of its initialisation. The SPD should program the routing model only after SP initialisation has completed e.g. in the SPD initialisation function pointed to by the `bl32_init` variable.

The SPD should determine the mechanism to pass control to the Secure Payload after receiving an interrupt from the EL3 runtime firmware. This information could either be provided to the SPD service at build time or by the SP at runtime.

Test secure payload dispatcher behavior

Note: Where this document discusses `TSP_NS_INTR_ASYNC_PREEMPT` as being `1`, the same results also apply when `EL3_EXCEPTION_HANDLING` is `1`.

The TSPD only handles Secure-EL1 interrupts and is provided with the following routing model at build time.

- Secure-EL1 interrupts are routed to EL3 when execution is in non-secure state and are routed to the FEL when execution is in the secure state i.e **CSS=0, TEL3=0** & **CSS=1, TEL3=1** for Secure-EL1 interrupts
- When the build flag `TSP_NS_INTR_ASYNC_PREEMPT` is zero, the default routing model is used for non-secure interrupts. They are routed to the FEL in either security state i.e **CSS=0, TEL3=0** & **CSS=1, TEL3=0** for Non-secure interrupts.
- When the build flag `TSP_NS_INTR_ASYNC_PREEMPT` is defined to 1, then the non secure interrupts are routed to EL3 when execution is in secure state i.e **CSS=0, TEL3=1** for non-secure interrupts. This effectively preempts Secure-EL1. The default routing model is used for non secure interrupts in non-secure state. i.e **CSS=1, TEL3=0**.

It performs the following actions in the `tspd_init()` function to fulfill the requirements mentioned earlier.

1. It passes control to the Test Secure Payload to perform its initialisation. The TSP provides the address of the vector table `tsp_vectors` in the SP which also includes the handler for Secure-EL1 interrupts in the `sel1_intr_entry` field. The TSPD passes control to the TSP at this address when it receives a Secure-EL1 interrupt.

   The handover agreement between the TSP and the TSPD requires that the TSPD masks all interrupts (`PSTATE.DAIF` bits) when it calls `tsp_sel1_intr_entry()`. The TSP has to preserve the callee saved general purpose, SP_EL1/Secure-EL0, LR, VFP and system registers. It can use `x0-x18` to enable its C runtime.

2. The TSPD implements a handler function for Secure-EL1 interrupts. This function is registered with the EL3 runtime firmware using the `register_interrupt_type_handler()` API as follows

```
/* Forward declaration */
interrupt_type_handler tspd_secure_el1_interrupt_handler;
int32_t rc, flags = 0;
set_interrupt_rm_flag(flags, NON_SECURE);
rc = register_interrupt_type_handler(INTR_TYPE_S_EL1,
                                     tspd_secure_el1_interrupt_handler,
                                     flags);
if (rc)
    panic();
```

3. When the build flag `TSP_NS_INTR_ASYNC_PREEMPT` is defined to 1, the TSPD implements a handler function for non-secure interrupts. This function is registered with the EL3 runtime firmware using the `register_interrupt_type_handler()` API as follows

```
/* Forward declaration */
interrupt_type_handler tspd_ns_interrupt_handler;
int32_t rc, flags = 0;
set_interrupt_rm_flag(flags, SECURE);
rc = register_interrupt_type_handler(INTR_TYPE_NS,
                                    tspd_ns_interrupt_handler,
                                    flags);
if (rc)
    panic();
```

Secure payload

A Secure Payload must implement an interrupt handling framework at Secure-EL1 (Secure-EL1 IHF) to support its chosen interrupt routing model. Secure payload execution will alternate between the below cases.

1. In the code where IRQ, FIQ or both interrupts are enabled, if an interrupt type is targeted to the FEL, then it will be routed to the Secure-EL1 exception vector table. This is defined as the **asynchronous mode** of handling interrupts. This mode applies to both Secure-EL1 and non-secure interrupts.
2. In the code where both interrupts are disabled, if an interrupt type is targeted to the FEL, then execution will eventually migrate to the non-secure state. Any non-secure interrupts will be handled as described in the routing model where **CSS=1 and TEL3=0**. Secure-EL1 interrupts will be routed to EL3 (as per the routing model where **CSS=1 and TEL3=1**) where the SPD service will hand them to the SP. This is defined as the **synchronous mode** of handling interrupts.

The interrupt handling framework implemented by the SP should support one or both these interrupt handling models depending upon the chosen routing model.

The following list briefly describes how the choice of a valid routing model (see *Valid routing models*) effects the implementation of the Secure-EL1 IHF. If the choice of the interrupt routing model is not known to the SPD service at compile time, then the SP should pass this information to the SPD service at runtime during its initialisation phase.

As mentioned earlier, an Arm GICv2 system is considered and it is assumed that the FIQ signal is used to generate Secure-EL1 interrupts and the IRQ signal is used to generate non-secure interrupts in either security state.

Secure payload IHF design w.r.t secure-EL1 interrupts

1. **CSS=0, TEL3=0**. If `PSTATE.F=0`, Secure-EL1 interrupts will be triggered at one of the Secure-EL1 FIQ exception vectors. The Secure-EL1 IHF should implement support for handling FIQ interrupts asynchronously.

   If `PSTATE.F=1` then Secure-EL1 interrupts will be handled as per the synchronous interrupt handling model. The SP could implement this scenario by exporting a separate entrypoint for Secure-EL1 interrupts to the SPD service during the registration phase. The SPD service would also need to know the state of the system, general purpose and the `PSTATE` registers in which it should arrange to return execution to the SP. The SP should provide this information in an implementation defined way during the registration phase if it is not known to the SPD service at build time.

2. **CSS=1, TEL3=1**. Interrupts are routed to EL3 when execution is in non-secure state. They should be handled through the synchronous interrupt handling model as described in 1. above.

3. **CSS=0, TEL3=1**. Secure-EL1 interrupts are routed to EL3 when execution is in secure state. They will not be visible to the SP. The `PSTATE.F` bit in Secure-EL1/Secure-EL0 will not mask FIQs. The EL3 runtime firmware will call the handler registered by the SPD service for Secure-EL1 interrupts. Secure-EL1 IHF should then handle all Secure-EL1 interrupt through the synchronous interrupt handling model described in 1. above.

Secure payload IHF design w.r.t non-secure interrupts

1. **CSS=0, TEL3=0**. If `PSTATE.I=0`, non-secure interrupts will be triggered at one of the Secure-EL1 IRQ exception vectors . The Secure-EL1 IHF should co-ordinate with the SPD service to transfer execution to the non-secure state where the interrupt should be handled e.g the SP could allocate a function identifier to issue a SMC64 or SMC32 to the SPD service which indicates that the SP execution has been preempted by a non-secure interrupt. If this function identifier is not known to the SPD service at compile time then the SP could provide it during the registration phase.

   If `PSTATE.I=1` then the non-secure interrupt will pend until execution resumes in the non-secure state.

2. **CSS=0, TEL3=1**. Non-secure interrupts are routed to EL3. They will not be visible to the SP. The `PSTATE.I` bit in Secure-EL1/Secure-EL0 will have not effect. The SPD service should register a non-secure interrupt handler which should save the SP state correctly and resume execution in the non-secure state where the interrupt will be handled. The Secure-EL1 IHF does not need to take any action.

3. **CSS=1, TEL3=0**. Non-secure interrupts are handled in the FEL in non-secure state (EL1/EL2) and are not visible to the SP. This routing model does not affect the SP behavior.

A Secure Payload must also ensure that all Secure-EL1 interrupts are correctly configured at the interrupt controller by the platform port of the EL3 runtime firmware. It should configure any additional Secure-EL1 interrupts which the EL3 runtime firmware is not aware of through its platform port.

Test secure payload behavior

The routing model for Secure-EL1 and non-secure interrupts chosen by the TSP is described in Section *Secure Payload Dispatcher*. It is known to the TSPD service at build time.

The TSP implements an entrypoint (`tsp_sel1_intr_entry()`) for handling Secure-EL1 interrupts taken in non-secure state and routed through the TSPD service (synchronous handling model). It passes the reference to this entrypoint via `tsp_vectors` to the TSPD service.

The TSP also replaces the default exception vector table referenced through the `early_exceptions` variable, with a vector table capable of handling FIQ and IRQ exceptions taken at the same (Secure-EL1) exception level. This table is referenced through the `tsp_exceptions` variable and programmed into the VBAR_EL1. It caters for the asynchronous handling model.

The TSP also programs the Secure Physical Timer in the Arm Generic Timer block to raise a periodic interrupt (every half a second) for the purpose of testing interrupt management across all the software components listed in *Software components*.

5.5.5 Interrupt handling

This section describes in detail the role of each software component (see Section *Software components*) in handling an interrupt of a particular type.

EL3 runtime firmware

The EL3 runtime firmware populates the IRQ and FIQ exception vectors referenced by the `runtime_exceptions` variable as follows.

1. IRQ and FIQ exceptions taken from the current exception level with `SP_EL0` or `SP_EL3` are reported as irrecoverable error conditions. As mentioned earlier, EL3 runtime firmware always executes with the `PSTATE.I` and `PSTATE.F` bits set.
2. The following text describes how the IRQ and FIQ exceptions taken from a lower exception level using AArch64 or AArch32 are handled.

When an interrupt is generated, the vector for each interrupt type is responsible for:

1. Saving the entire general purpose register context (x0-x30) immediately upon exception entry. The registers are saved in the per-cpu `cpu_context` data structure referenced by the `SP_EL3`register.
2. Saving the `ELR_EL3`, `SP_EL0` and `SPSR_EL3` system registers in the per-cpu `cpu_context` data structure referenced by the `SP_EL3` register.
3. Switching to the C runtime stack by restoring the `CTX_RUNTIME_SP` value from the per-cpu `cpu_context` data structure in `SP_EL0` and executing the `msr spsel, #0` instruction.
4. Determining the type of interrupt. Secure-EL1 interrupts will be signaled at the FIQ vector. Non-secure interrupts will be signaled at the IRQ vector. The platform should implement the following API to determine the type of the pending interrupt.

```
uint32_t plat_ic_get_pending_interrupt_type(void);
```

It should return either `INTR_TYPE_S_EL1` or `INTR_TYPE_NS`.

5. Determining the handler for the type of interrupt that has been generated. The following API has been added for this purpose.

```
interrupt_type_handler get_interrupt_type_handler(uint32_t interrupt_
↪type);
```

It returns the reference to the registered handler for this interrupt type. The `handler` is retrieved from the `intr_type_desc_t` structure as described in Section 2. `NULL` is returned if no handler has been registered for this type of interrupt. This scenario is reported as an irrecoverable error condition.

6. Calling the registered handler function for the interrupt type generated. The `id` parameter is set to `INTR_ID_UNAVAILABLE` currently. The id along with the current security state and a reference to the `cpu_context_t` structure for the current security state are passed to the handler function as its arguments.

   The handler function returns a reference to the per-cpu `cpu_context_t` structure for the target security state.

7. Calling `el3_exit()` to return from EL3 into a lower exception level in the security state determined by the handler routine. The `el3_exit()` function is responsible for restoring the register context from the `cpu_context_t` data structure for the target security state.

Secure payload dispatcher

Interrupt entry

The SPD service begins handling an interrupt when the EL3 runtime firmware calls the handler function for that type of interrupt. The SPD service is responsible for the following:

1. Validating the interrupt. This involves ensuring that the interrupt was generated according to the interrupt routing model specified by the SPD service during registration. It should use the security state of the exception level (passed in the `flags` parameter of the handler) where the interrupt was taken from to determine this. If the interrupt is not recognised then the handler should treat it as an irrecoverable error condition.

   An SPD service can register a handler for Secure-EL1 and/or Non-secure interrupts. A non-secure interrupt should never be routed to EL3 from from non-secure state. Also if a routing model is chosen where Secure-EL1 interrupts are routed to S-EL1 when execution is in Secure state, then a S-EL1 interrupt should never be routed to EL3 from secure state. The handler could use the security state flag to check this.

2. Determining whether a context switch is required. This depends upon the routing model and interrupt type. For non secure and S-EL1 interrupt, if the security state of the execution context where the interrupt was generated is not the same as the security state required for handling the interrupt, a context switch is required. The following 2 cases require a context switch from secure to non-secure or vice-versa:

   1. A Secure-EL1 interrupt taken from the non-secure state should be routed to the Secure Payload.

2. A non-secure interrupt taken from the secure state should be routed to the last known non-secure exception level.

   The SPD service must save the system register context of the current security state. It must then restore the system register context of the target security state. It should use the `cm_set_next_eret_context()` API to ensure that the next `cpu_context` to be restored is of the target security state.

   If the target state is secure then execution should be handed to the SP as per the synchronous interrupt handling model it implements. A Secure-EL1 interrupt can be routed to EL3 while execution is in the SP. This implies that SP execution can be preempted while handling an interrupt by a another higher priority Secure-EL1 interrupt or a EL3 interrupt. The SPD service should be able to handle this preemption or manage secure interrupt priorities before handing control to the SP.

3. Setting the return value of the handler to the per-cpu `cpu_context` if the interrupt has been successfully validated and ready to be handled at a lower exception level.

The routing model allows non-secure interrupts to interrupt Secure-EL1 when in secure state if it has been configured to do so. The SPD service and the SP should implement a mechanism for routing these interrupts to the last known exception level in the non-secure state. The former should save the SP context, restore the non-secure context and arrange for entry into the non-secure state so that the interrupt can be handled.

Interrupt exit

When the Secure Payload has finished handling a Secure-EL1 interrupt, it could return control back to the SPD service through a SMC32 or SMC64. The SPD service should handle this secure monitor call so that execution resumes in the exception level and the security state from where the Secure-EL1 interrupt was originally taken.

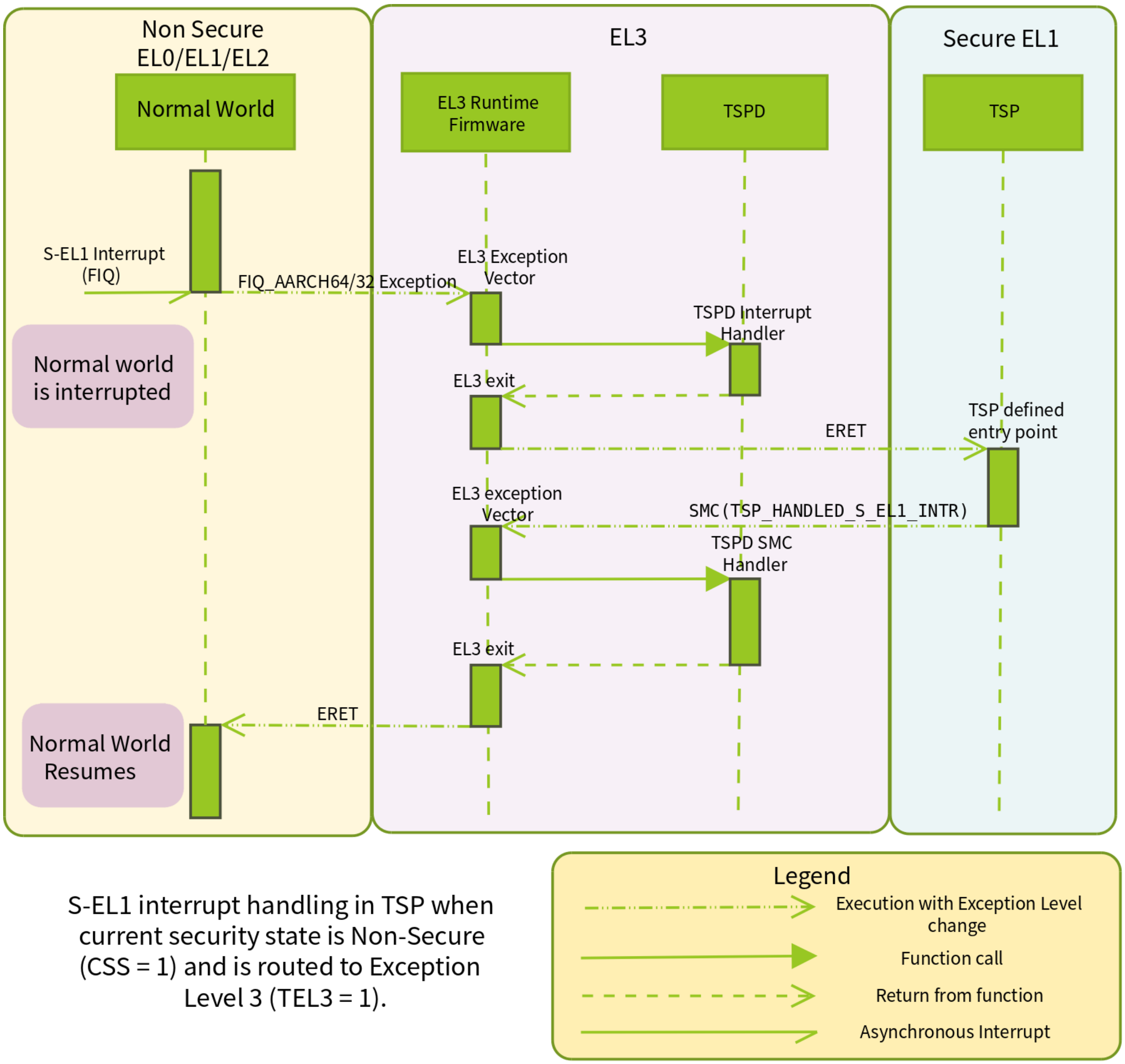
Test secure payload dispatcher Secure-EL1 interrupt handling

The example TSPD service registers a handler for Secure-EL1 interrupts taken from the non-secure state. During execution in S-EL1, the TSPD expects that the Secure-EL1 interrupts are handled in S-EL1 by TSP. Its handler `tspd_secure_el1_interrupt_handler()` expects only to be invoked for Secure-EL1 originating from the non-secure state. It takes the following actions upon being invoked.

1. It uses the security state provided in the `flags` parameter to ensure that the secure interrupt originated from the non-secure state. It asserts if this is not the case.
2. It saves the system register context for the non-secure state by calling `cm_el1_sysregs_context_save(NON_SECURE);`.
3. It sets the `ELR_EL3` system register to `tsp_sel1_intr_entry` and sets the `SPSR_EL3.DAIF` bits in the secure CPU context. It sets `x0` to `TSP_HANDLE_SEL1_INTR_AND_RETURN`. If the TSP was preempted earlier by a non secure interrupt during `yielding` SMC processing, save the registers that will be trashed, which is the `ELR_EL3` and `SPSR_EL3`, in order to be able to re-enter TSP for Secure-EL1 interrupt processing. It does not need to save any other secure context since the TSP is expected to preserve it (see section *Test secure payload dispatcher behavior*).
4. It restores the system register context for the secure state by calling `cm_el1_sysregs_context_restore(SECURE);`.

5. It ensures that the secure CPU context is used to program the next exception return from EL3 by calling `cm_set_next_eret_context(SECURE);`.
6. It returns the per-cpu `cpu_context` to indicate that the interrupt can now be handled by the SP. `x1` is written with the value of `elr_el3` register for the non-secure state. This information is used by the SP for debugging purposes.

The figure below describes how the interrupt handling is implemented by the TSPD when a Secure-EL1 interrupt is generated when execution is in the non-secure state.



S-EL1 interrupt handling in TSP when current security state is Non-Secure (CSS = 1) and is routed to Exception Level 3 (TEL3 = 1).

The TSP issues an SMC with `TSP_HANDLED_S_EL1_INTR` as the function identifier to signal completion of interrupt handling.

The TSPD service takes the following actions in `tspd_smc_handler()` function upon receiving an SMC with `TSP_HANDLED_S_EL1_INTR` as the function identifier:

1. It ensures that the call originated from the secure state otherwise execution returns to the non-secure state with `SMC_UNK` in `x0`.
2. It restores the saved `ELR_EL3` and `SPSR_EL3` system registers back to the secure CPU context (see step 3 above) in case the TSP had been preempted by a non secure interrupt earlier.
3. It restores the system register context for the non-secure state by calling `cm_el1_sysregs_context_restore(NON_SECURE)`.
4. It ensures that the non-secure CPU context is used to program the next exception return from EL3 by calling `cm_set_next_eret_context(NON_SECURE)`.
5. `tspd_smc_handler()` returns a reference to the non-secure `cpu_context` as the return value.

Test secure payload dispatcher non-secure interrupt handling

The TSP in Secure-EL1 can be preempted by a non-secure interrupt during `yielding` SMC processing or by a higher priority EL3 interrupt during Secure-EL1 interrupt processing. When `EL3_EXCEPTION_HANDLING` is `0`, only non-secure interrupts can cause preemption of TSP since there are no EL3 interrupts in the system. With `EL3_EXCEPTION_HANDLING=1` however, any EL3 interrupt may preempt Secure execution.

It should be noted that while TSP is preempted, the TSPD only allows entry into the TSP either for Secure-EL1 interrupt handling or for resuming the preempted `yielding` SMC in response to the `TSP_FID_RESUME` SMC from the normal world. (See Section *Implication of preempted SMC on Non-Secure Software*).

The non-secure interrupt triggered in Secure-EL1 during `yielding` SMC processing can be routed to either EL3 or Secure-EL1 and is controlled by build option `TSP_NS_INTR_ASYNC_PREEMPT` (see Section *Test secure payload dispatcher behavior*). If the build option is set, the TSPD will set the routing model for the non-secure interrupt to be routed to EL3 from secure state i.e. **TEL3=1, CSS=0** and registers `tspd_ns_interrupt_handler()` as the non-secure interrupt handler. The `tspd_ns_interrupt_handler()` on being invoked ensures that the interrupt originated from the secure state and disables routing of non-secure interrupts from secure state to EL3. This is to prevent further preemption (by a non-secure interrupt) when TSP is reentered for handling Secure-EL1 interrupts that triggered while execution was in the normal world. The `tspd_ns_interrupt_handler()` then invokes `tspd_handle_sp_preemption()` for further handling.

If the `TSP_NS_INTR_ASYNC_PREEMPT` build option is zero (default), the default routing model for non-secure interrupt in secure state is in effect i.e. **TEL3=0, CSS=0**. During `yielding` SMC processing, the IRQ exceptions are unmasked i.e. `PSTATE.I=0`, and a non-secure interrupt will trigger at Secure-EL1 IRQ exception vector. The TSP saves the general purpose register context and issues an SMC with `TSP_PREEMPTED` as the function identifier to signal preemption of TSP. The TSPD SMC handler, `tspd_smc_handler()`, ensures that the SMC call originated from the secure state otherwise execution returns to the non-secure state with `SMC_UNK` in `x0`. It then invokes `tspd_handle_sp_preemption()` for further handling.

The `tspd_handle_sp_preemption()` takes the following actions upon being invoked:

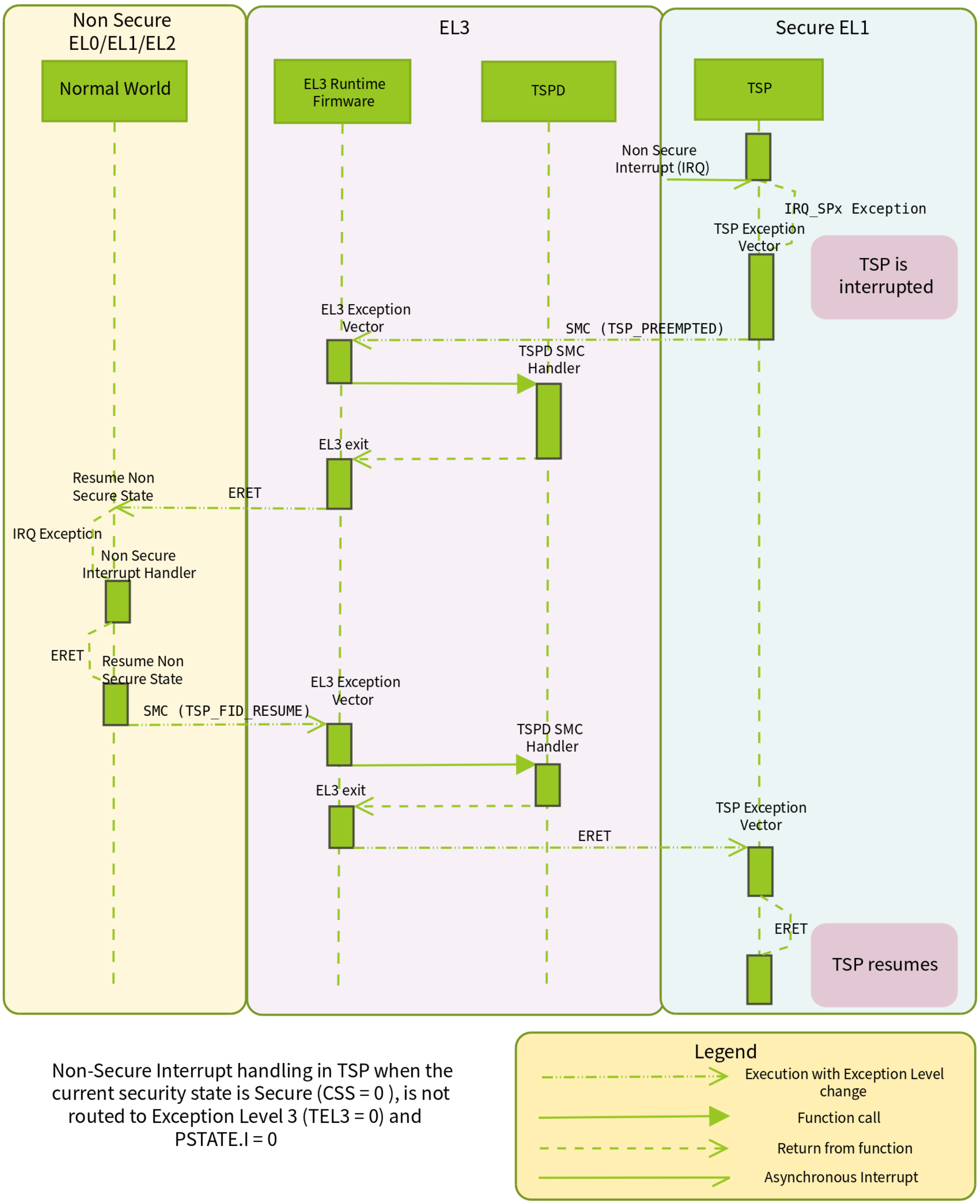
1. It saves the system register context for the secure state by calling `cm_el1_sysregs_context_save(SECURE)`.

2. It restores the system register context for the non-secure state by calling `cm_el1_sysregs_context_restore(NON_SECURE)`.
3. It ensures that the non-secure CPU context is used to program the next exception return from EL3 by calling `cm_set_next_eret_context(NON_SECURE)`.
4. `SMC_PREEMPTED` is set in x0 and return to non secure state after restoring non secure context.

The Normal World is expected to resume the TSP after the `yielding` SMC preemption by issuing an SMC with `TSP_FID_RESUME` as the function identifier (see section *Implication of preempted SMC on Non-Secure Software*). The TSPD service takes the following actions in `tspd_smc_handler()` function upon receiving this SMC:

1. It ensures that the call originated from the non secure state. An assertion is raised otherwise.
2. Checks whether the TSP needs a resume i.e check if it was preempted. It then saves the system register context for the non-secure state by calling `cm_el1_sysregs_context_save(NON_SECURE)`.
3. Restores the secure context by calling `cm_el1_sysregs_context_restore(SECURE)`
4. It ensures that the secure CPU context is used to program the next exception return from EL3 by calling `cm_set_next_eret_context(SECURE)`.
5. `tspd_smc_handler()` returns a reference to the secure `cpu_context` as the return value.

The figure below describes how the TSP/TSPD handle a non-secure interrupt when it is generated during execution in the TSP with `PSTATE.I` = 0 when the `TSP_NS_INTR_ASYNC_PREEMPT` build flag is 0.


Non Secure
EL0/EL1/EL2
EL3
Secure EL1
Normal World
EL3 Runtime Firmware
TSPD
TSP
Non Secure Interrupt (IRQ)
IRQ_SPx Exception
TSP Exception Vector
TSP is interrupted
EL3 Exception Vector
SMC (TSP_PREEMPTED)
TSPD SMC Handler
EL3 exit
Resume Non Secure State
ERET
IRQ Exception
Non Secure Interrupt Handler
ERET
Resume Non Secure State
EL3 Exception Vector
SMC (TSP_FID_RESUME)
TSPD SMC Handler
EL3 exit
TSP Exception Vector
ERET
ERET
TSP resumes
Non-Secure Interrupt handling in TSP when the current security state is Secure (CSS = 0), is not routed to Exception Level 3 (TEL3 = 0) and PSTATE.I = 0
Legend
Execution with Exception Level change
Function call
Return from function
Asynchronous Interrupt

Secure payload interrupt handling

The SP should implement one or both of the synchronous and asynchronous interrupt handling models depending upon the interrupt routing model it has chosen (as described in section *Secure Payload*).

In the synchronous model, it should begin handling a Secure-EL1 interrupt after receiving control from the SPD service at an entrypoint agreed upon during build time or during the registration phase. Before handling the interrupt, the SP should save any Secure-EL1 system register context which is needed for resuming normal execution in the SP later e.g. `SPSR_EL1`, `ELR_EL1`. After handling the interrupt, the SP could return control back to the exception level and security state where the interrupt was originally taken from. The SP should use an SMC32 or SMC64 to ask the SPD service to do this.

In the asynchronous model, the Secure Payload is responsible for handling non-secure and Secure-EL1 interrupts at the IRQ and FIQ vectors in its exception vector table when `PSTATE.I` and `PSTATE.F` bits are 0. As described earlier, when a non-secure interrupt is generated, the SP should coordinate with the SPD service to pass control back to the non-secure state in the last known exception level. This will allow the non-secure interrupt to be handled in the non-secure state.

Test secure payload behavior

The TSPD hands control of a Secure-EL1 interrupt to the TSP at the `tsp_sel1_intr_entry()`. The TSP handles the interrupt while ensuring that the handover agreement described in Section *Test secure payload dispatcher behavior* is maintained. It updates some statistics by calling `tsp_update_sync_sel1_intr_stats()`. It then calls `tsp_common_int_handler()` which.

1. Checks whether the interrupt is the secure physical timer interrupt. It uses the platform API `plat_ic_get_pending_interrupt_id()` to get the interrupt number. If it is not the secure physical timer interrupt, then that means that a higher priority interrupt has preempted it. Invoke `tsp_handle_preemption()` to handover control back to EL3 by issuing an SMC with `TSP_PREEMPTED` as the function identifier.
2. Handles the secure timer interrupt interrupt by acknowledging it using the `plat_ic_acknowledge_interrupt()` platform API, calling `tsp_generic_timer_handler()` to reprogram the secure physical generic timer and calling the `plat_ic_end_of_interrupt()` platform API to signal end of interrupt processing.

The TSP passes control back to the TSPD by issuing an SMC64 with `TSP_HANDLED_S_EL1_INTR` as the function identifier.

The TSP handles interrupts under the asynchronous model as follows.

1. Secure-EL1 interrupts are handled by calling the `tsp_common_int_handler()` function. The function has been described above.
2. Non-secure interrupts are handled by calling the `tsp_common_int_handler()` function which ends up invoking `tsp_handle_preemption()` and issuing an SMC64 with `TSP_PREEMPTED` as the function identifier. Execution resumes at the instruction that follows this SMC instruction when the TSPD hands control to the TSP in response to an SMC with `TSP_FID_RESUME` as the function identifier from the non-secure state (see section *Test secure payload dispatcher non-secure interrupt handling*).

5.5.6 Other considerations

Implication of preempted SMC on Non-Secure Software

A `yielding` SMC call to Secure payload can be preempted by a non-secure interrupt and the execution can return to the non-secure world for handling the interrupt (For details on `yielding` SMC refer SMC calling convention). In this case, the SMC call has not completed its execution and the execution must return back to the secure payload to resume the preempted SMC call. This can be achieved by issuing an SMC call which instructs to resume the preempted SMC.

A `fast` SMC cannot be preempted and hence this case will not happen for a fast SMC call.

In the Test Secure Payload implementation, `TSP_FID_RESUME` is designated as the resume SMC FID. It is important to note that `TSP_FID_RESUME` is a `yielding` SMC which means it too can be be preempted. The typical non secure software sequence for issuing a `yielding` SMC would look like this, assuming `P.STATE.I=0` in the non secure state :

```
int rc;
rc = smc(TSP_YIELD_SMC_FID, ...);      /* Issue a Yielding SMC call */
/* The pending non-secure interrupt is handled by the interrupt handler
   and returns back here. */
while (rc == SMC_PREEMPTED) {          /* Check if the SMC call is preempted */
    rc = smc(TSP_FID_RESUME);          /* Issue resume SMC call */
}
```

The `TSP_YIELD_SMC_FID` is any `yielding` SMC function identifier and the smc() function invokes a SMC call with the required arguments. The pending non-secure interrupt causes an IRQ exception and the IRQ handler registered at the exception vector handles the non-secure interrupt and returns. The return value from the SMC call is tested for `SMC_PREEMPTED` to check whether it is preempted. If it is, then the resume SMC call `TSP_FID_RESUME` is issued. The return value of the SMC call is tested again to check if it is preempted. This is done in a loop till the SMC call succeeds or fails. If a `yielding` SMC is preempted, until it is resumed using `TSP_FID_RESUME` SMC and completed, the current TSPD prevents any other SMC call from re-entering TSP by returning `SMC_UNK` error.

5.6 PSCI Power Domain Tree Structure

5.6.1 Requirements

1. A platform must export the `plat_get_aff_count()` and `plat_get_aff_state()` APIs to enable the generic PSCI code to populate a tree that describes the hierarchy of power domains in the system. This approach is inflexible because a change to the topology requires a change in the code.

   It would be much simpler for the platform to describe its power domain tree in a data structure.

2. The generic PSCI code generates MPIDRs in order to populate the power domain tree. It also uses an MPIDR to find a node in the tree. The assumption that a platform will use exactly the same MPIDRs as

generated by the generic PSCI code is not scalable. The use of an MPIDR also restricts the number of levels in the power domain tree to four.

Therefore, there is a need to decouple allocation of MPIDRs from the mechanism used to populate the power domain topology tree.

3. The current arrangement of the power domain tree requires a binary search over the sibling nodes at a particular level to find a specified power domain node. During a power management operation, the tree is traversed from a 'start' to an 'end' power level. The binary search is required to find the node at each level. The natural way to perform this traversal is to start from a leaf node and follow the parent node pointer to reach the end level.

   Therefore, there is a need to define data structures that implement the tree in a way which facilitates such a traversal.

4. The attributes of a core power domain differ from the attributes of power domains at higher levels. For example, only a core power domain can be identified using an MPIDR. There is no requirement to perform state coordination while performing a power management operation on the core power domain.

   Therefore, there is a need to implement the tree in a way which facilitates this distinction between a leaf and non-leaf node and any associated optimizations.

5.6.2 Design

Describing a power domain tree

To fulfill requirement 1., the existing platform APIs `plat_get_aff_count()` and `plat_get_aff_state()` have been removed. A platform must define an array of unsigned chars such that:

1. The first entry in the array specifies the number of power domains at the highest power level implemented in the platform. This caters for platforms where the power domain tree does not have a single root node, for example, the FVP has two cluster power domains at the highest level (1).
2. Each subsequent entry corresponds to a power domain and contains the number of power domains that are its direct children.
3. The size of the array minus the first entry will be equal to the number of non-leaf power domains.
4. The value in each entry in the array is used to find the number of entries to consider at the next level. The sum of the values (number of children) of all the entries at a level specifies the number of entries in the array for the next level.

The following example power domain topology tree will be used to describe the above text further. The leaf and non-leaf nodes in this tree have been numbered separately.

```
                                         +-+
                                         |0|
                                         +-+
                                        /   \
```

(continues on next page)

(continued from previous page)

```
                             /     \
                            /       \
                           /         \
                          /           \
                         /             \
                        /               \
                       /                 \
                      /                   \
                     /                     \
                  +-+                       +-+
                  |1|                       |2|
                  +-+                       +-+
                 /   \                     /   \
                /     \                   /     \
               /       \                 /       \
              /         \               /         \
           +-+           +-+         +-+           +-+
           |3|           |4|         |5|           |6|
           +-+           +-+         +-+           +-+
  +---+-----+    +----+----|     +----+----+     +----+-----+-----+
  |   |     |    |    |    |     |    |    |     |    |     |     |
  |   |     |    |    |    |     |    |    |     |    |     |     |
  v   v     v    v    v    v     v    v    v     v    v     v     v
+-+  +-+   +-+  +-+  +-+  +-+   +-+  +-+  +-+   +-+  +--+  +--+  +--+
|0|  |1|   |2|  |3|  |4|  |5|   |6|  |7|  |8|   |9|  |10|  |11|  |12|
+-+  +-+   +-+  +-+  +-+  +-+   +-+  +-+  +-+   +-+  +--+  +--+  +--+
```

This tree is defined by the platform as the array described above as follows:

```
#define PLAT_NUM_POWER_DOMAINS       20
#define PLATFORM_CORE_COUNT          13
#define PSCI_NUM_NON_CPU_PWR_DOMAINS \
                   (PLAT_NUM_POWER_DOMAINS - PLATFORM_CORE_COUNT)

unsigned char plat_power_domain_tree_desc[] = { 1, 2, 2, 2, 3, 3, 3, 4};
```

Removing assumptions about MPIDRs used in a platform

To fulfill requirement 2., it is assumed that the platform assigns a unique number (core index) between `0` and `PLAT_CORE_COUNT - 1` to each core power domain. MPIDRs could be allocated in any manner and will not be used to populate the tree.

`plat_core_pos_by_mpidr(mpidr)` will return the core index for the core corresponding to the MPIDR. It will return an error (-1) if an MPIDR is passed which is not allocated or corresponds to an absent core. The semantics of this platform API have changed since it is required to validate the passed MPIDR. It has been made a mandatory API as a result.

Another mandatory API, `plat_my_core_pos()` has been added to return the core index for the calling core. This API provides a more lightweight mechanism to get the index since there is no need to validate the MPIDR of the calling core.

The platform should assign the core indices (as illustrated in the diagram above) such that, if the core nodes are numbered from left to right, then the index for a core domain will be the same as the index returned by `plat_core_pos_by_mpidr()` or `plat_my_core_pos()` for that core. This relationship allows the core nodes to be allocated in a separate array (requirement 4.) during `psci_setup()` in such an order that the index of the core in the array is the same as the return value from these APIs.

Dealing with holes in MPIDR allocation

For platforms where the number of allocated MPIDRs is equal to the number of core power domains, for example, Juno and FVPs, the logic to convert an MPIDR to a core index should remain unchanged. Both Juno and FVP use a simple collision proof hash function to do this.

It is possible that on some platforms, the allocation of MPIDRs is not contiguous or certain cores have been disabled. This essentially means that the MPIDRs have been sparsely allocated, that is, the size of the range of MPIDRs used by the platform is not equal to the number of core power domains.

The platform could adopt one of the following approaches to deal with this scenario:

1. Implement more complex logic to convert a valid MPIDR to a core index while maintaining the relationship described earlier. This means that the power domain tree descriptor will not describe any core power domains which are disabled or absent. Entries will not be allocated in the tree for these domains.
2. Treat unallocated MPIDRs and disabled cores as absent but still describe them in the power domain descriptor, that is, the number of core nodes described is equal to the size of the range of MPIDRs allocated. This approach will lead to memory wastage since entries will be allocated in the tree but will allow use of a simpler logic to convert an MPIDR to a core index.

Traversing through and distinguishing between core and non-core power domains

To fulfill requirement 3 and 4, separate data structures have been defined to represent leaf and non-leaf power domain nodes in the tree.

```
/
↪*****************************************************************************
 * The following two data structures implement the power domain tree. The tree
 * is used to track the state of all the nodes i.e. power domain instances
 * described by the platform. The tree consists of nodes that describe CPU␣
↪power
 * domains i.e. leaf nodes and all other power domains which are parents of a
 * CPU power domain i.e. non-leaf nodes.
␣
↪****************************************************************************/
↪
typedef struct non_cpu_pwr_domain_node {
    /*
     * Index of the first CPU power domain node level 0 which has this node
     * as its parent.
     */
    unsigned int cpu_start_idx;
```

(continues on next page)

(continued from previous page)

```
    /*
     * Number of CPU power domains which are siblings of the domain indexed
     * by 'cpu_start_idx' i.e. all the domains in the range 'cpu_start_idx
     * -> cpu_start_idx + ncpus' have this node as their parent.
     */
    unsigned int ncpus;

    /* Index of the parent power domain node */
    unsigned int parent_node;

    -----
} non_cpu_pd_node_t;

typedef struct cpu_pwr_domain_node {
    u_register_t mpidr;

    /* Index of the parent power domain node */
    unsigned int parent_node;

    -----
} cpu_pd_node_t;
```

The power domain tree is implemented as a combination of the following data structures.

```
non_cpu_pd_node_t psci_non_cpu_pd_nodes[PSCI_NUM_NON_CPU_PWR_DOMAINS];
PER_CPU_DEFINE(cpu_pd_node_t, psci_cpu_pd_nodes);
```

Populating the power domain tree

The `populate_power_domain_tree()` function in `psci_setup.c` implements the algorithm to parse the power domain descriptor exported by the platform to populate the two arrays. It is essentially a breadth-first-search. The nodes for each level starting from the root are laid out one after another in the `psci_non_cpu_pd_nodes` and `psci_cpu_pd_nodes` arrays as follows:

```
psci_non_cpu_pd_nodes -> [[Level 3 nodes][Level 2 nodes][Level 1 nodes]]
psci_cpu_pd_nodes -> [Level 0 nodes]
```

For the example power domain tree illustrated above, the `psci_cpu_pd_nodes` will be populated as follows. The value in each entry is the index of the parent node. Other fields have been ignored for simplicity.

```
      +-------------+     ^
CPU0  |      3      |     |
      +-------------+     |
CPU1  |      3      |     |
      +-------------+     |
CPU2  |      3      |     |
      +-------------+     |
CPU3  |      4      |     |
```

(continues on next page)

(continued from previous page)

```
      +-------------+       |
CPU4  |      4      |       |
      +-------------+       |
CPU5  |      4      |       | PLATFORM_CORE_COUNT
      +-------------+       |
CPU6  |      5      |       |
      +-------------+       |
CPU7  |      5      |       |
      +-------------+       |
CPU8  |      5      |       |
      +-------------+       |
CPU9  |      6      |       |
      +-------------+       |
CPU10 |      6      |       |
      +-------------+       |
CPU11 |      6      |       |
      +-------------+       |
CPU12 |      6      |       v
      +-------------+
```

The `psci_non_cpu_pd_nodes` array will be populated as follows. The value in each entry is the index of the parent node.

```
      +-------------+       ^
PD0   |     -1      |       |
      +-------------+       |
PD1   |      0      |       |
      +-------------+       |
PD2   |      0      |       |
      +-------------+       |
PD3   |      1      |       | PLAT_NUM_POWER_DOMAINS -
      +-------------+       | PLATFORM_CORE_COUNT
PD4   |      1      |       |
      +-------------+       |
PD5   |      2      |       |
      +-------------+       |
PD6   |      2      |       |
      +-------------+       v
```

Each core can find its node in the `psci_cpu_pd_nodes` array using the `plat_my_core_pos()` function. When a core is turned on, the normal world provides an MPIDR. The `plat_core_pos_by_mpidr()` function is used to validate the MPIDR before using it to find the corresponding core node. The non-core power domain nodes do not need to be identified.

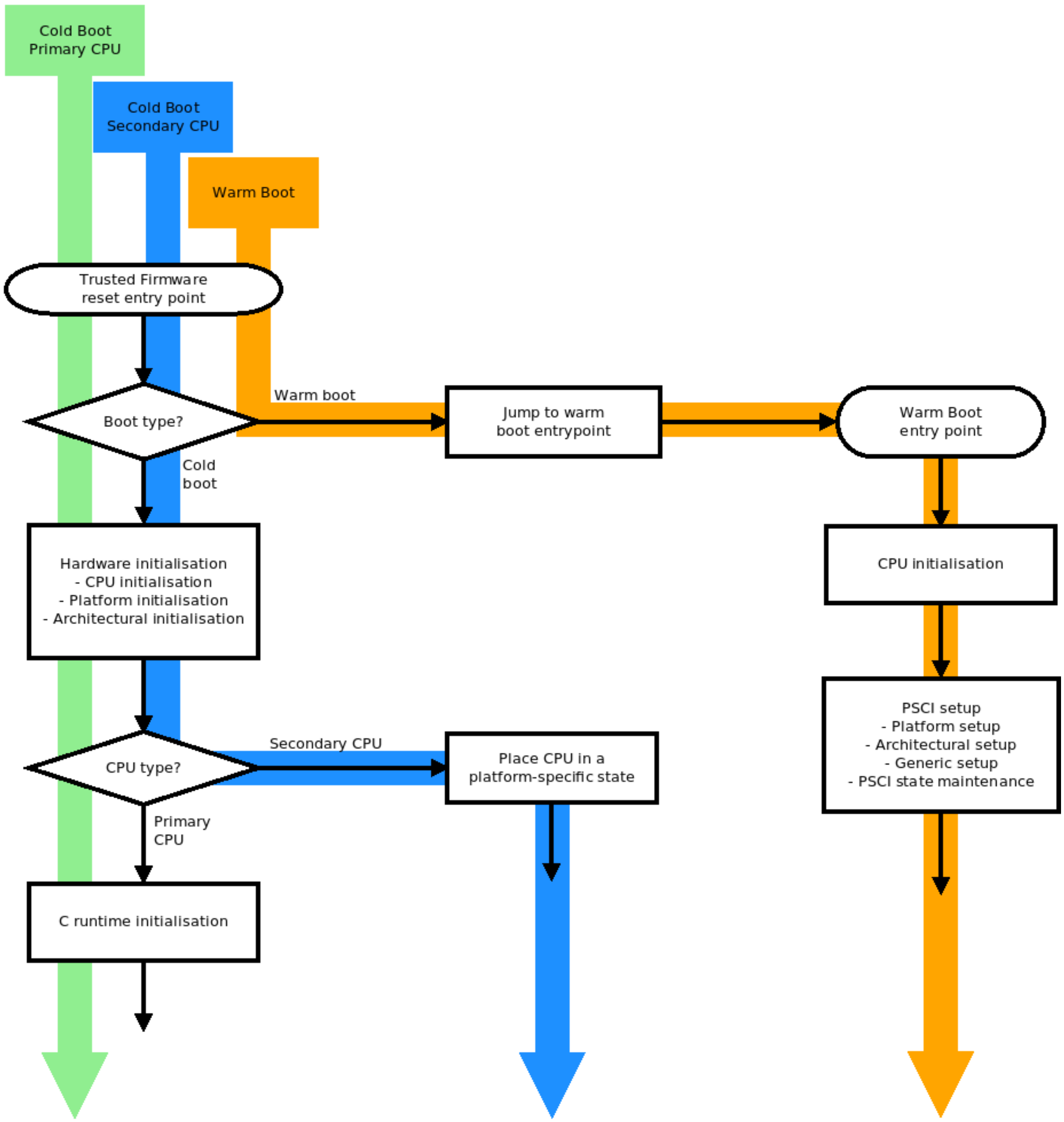
5.7 CPU Reset

This document describes the high-level design of the framework to handle CPU resets in Trusted Firmware-A (TF-A). It also describes how the platform integrator can tailor this code to the system configuration to some extent, resulting in a simplified and more optimised boot flow.

This document should be used in conjunction with the *Firmware Design* document which provides greater implementation details around the reset code, specifically for the cold boot path.

5.7.1 General reset code flow

The TF-A reset code is implemented in BL1 by default. The following high-level diagram illustrates this:



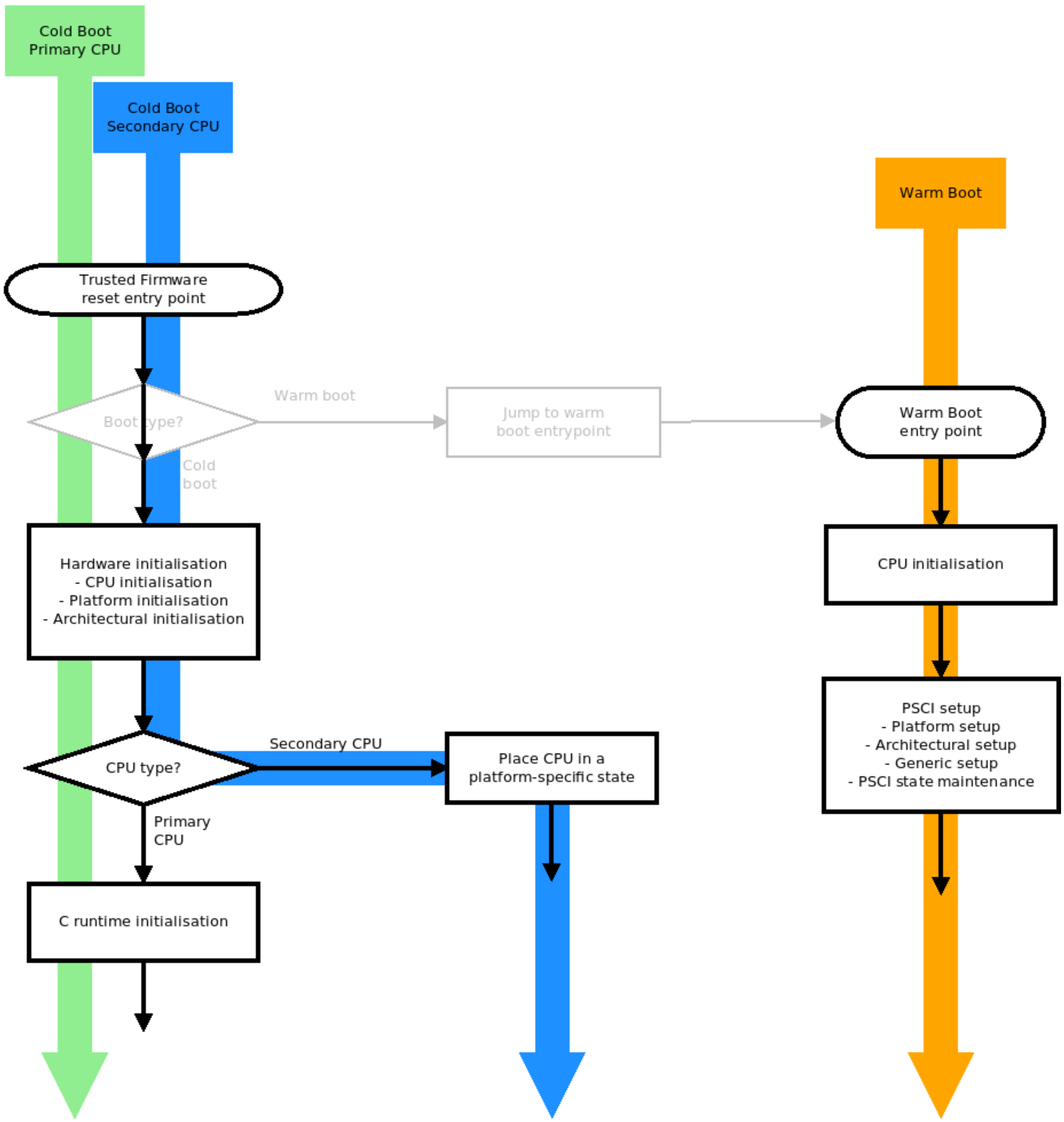
This diagram shows the default, unoptimised reset flow. Depending on the system configuration, some of these steps might be unnecessary. The following sections guide the platform integrator by indicating which build options exclude which steps, depending on the capability of the platform.

Note: If BL31 is used as the TF-A entry point instead of BL1, the diagram above is still relevant, as all these operations will occur in BL31 in this case. Please refer to section 6 "Using BL31 entrypoint as the reset address" for more information.

5.7.2 Programmable CPU reset address

By default, TF-A assumes that the CPU reset address is not programmable. Therefore, all CPUs start at the same address (typically address 0) whenever they reset. Further logic is then required to identify whether it is a cold or warm boot to direct CPUs to the right execution path.

If the reset vector address (reflected in the reset vector base address register `RVBAR_EL3`) is programmable then it is possible to make each CPU start directly at the right address, both on a cold and warm reset. Therefore, the boot type detection can be skipped, resulting in the following boot flow:



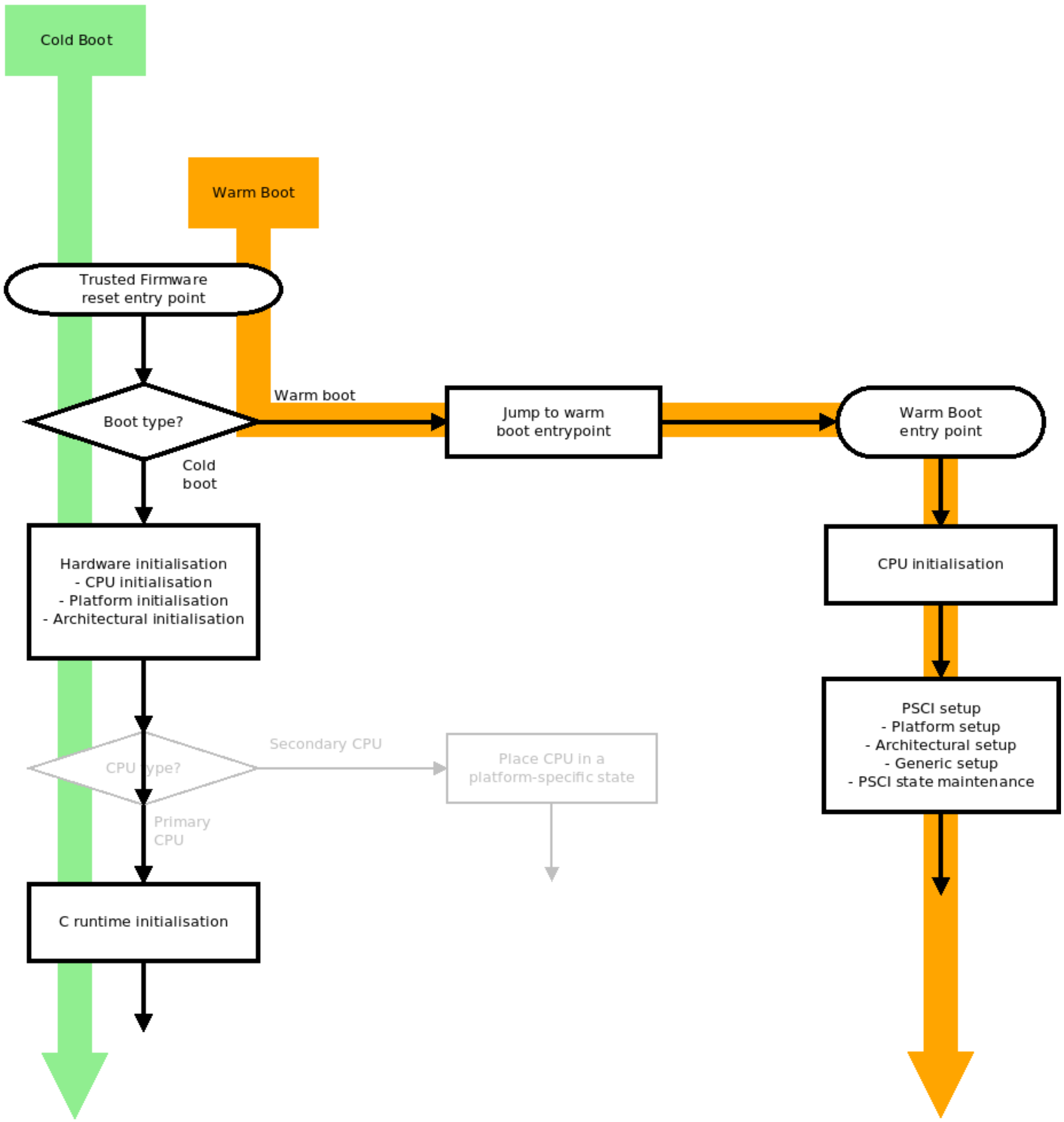
To enable this boot flow, compile TF-A with `PROGRAMMABLE_RESET_ADDRESS=1`. This option only affects the TF-A reset image, which is BL1 by default or BL31 if `RESET_TO_BL31=1`.

On both the FVP and Juno platforms, the reset vector address is not programmable so both ports use `PROGRAMMABLE_RESET_ADDRESS=0`.

5.7.3 Cold boot on a single CPU

By default, TF-A assumes that several CPUs may be released out of reset. Therefore, the cold boot code has to arbitrate access to hardware resources shared amongst CPUs. This is done by nominating one of the CPUs as the primary, which is responsible for initialising shared hardware and coordinating the boot flow with the other CPUs.

If the platform guarantees that only a single CPU will ever be brought up then no arbitration is required. The notion of primary/secondary CPU itself no longer applies. This results in the following boot flow:

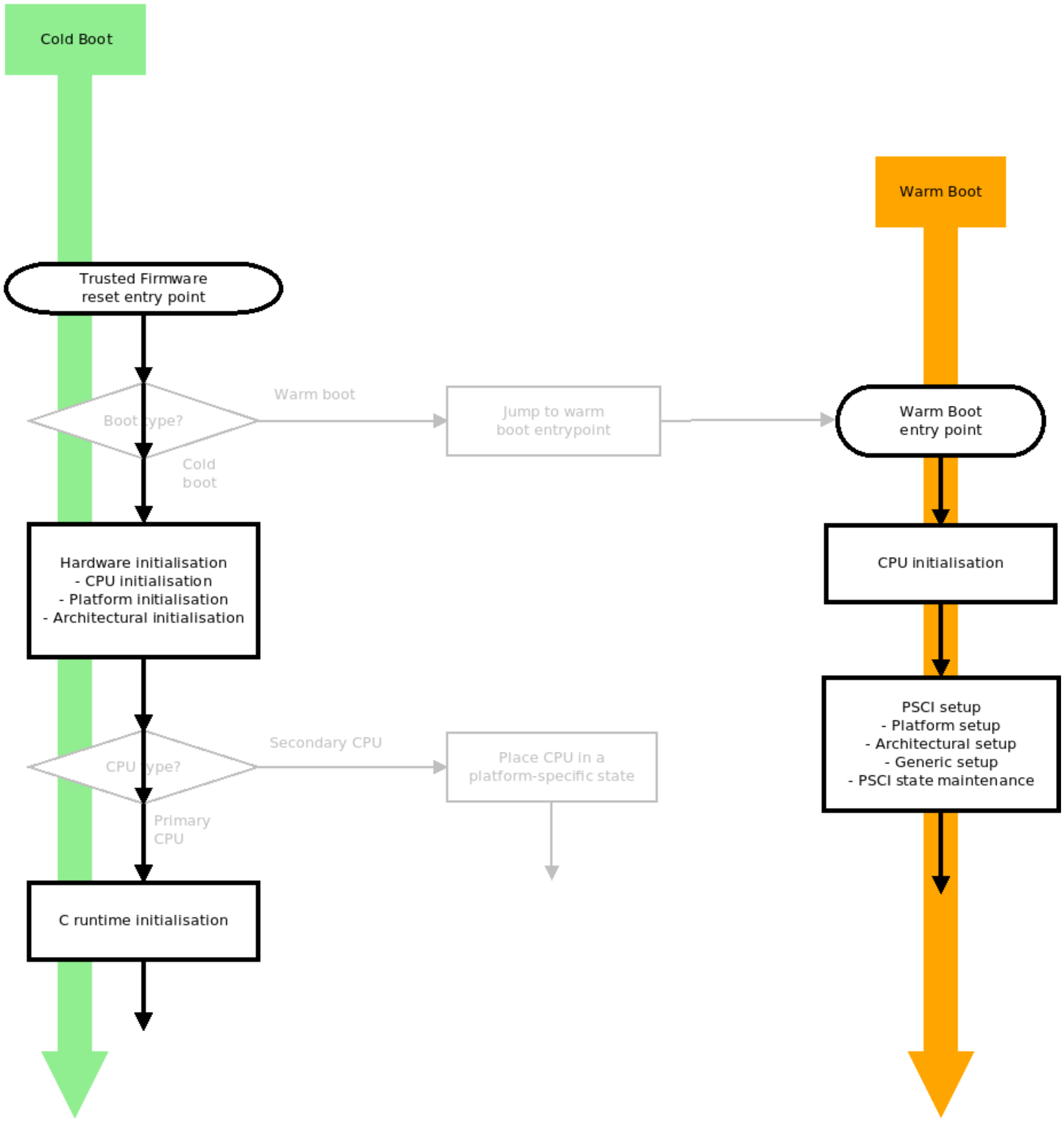


To enable this boot flow, compile TF-A with `COLD_BOOT_SINGLE_CPU=1`. This option only affects the TF-A reset image, which is BL1 by default or BL31 if `RESET_TO_BL31=1`.

On both the FVP and Juno platforms, although only one core is powered up by default, there are platform-specific ways to release any number of cores out of reset. Therefore, both platform ports use `COLD_BOOT_SINGLE_CPU=0`.

5.7.4 Programmable CPU reset address, Cold boot on a single CPU

It is obviously possible to combine both optimisations on platforms that have a programmable CPU reset address and which release a single CPU out of reset. This results in the following boot flow:



To enable this boot flow, compile TF-A with both `COLD_BOOT_SINGLE_CPU=1` and `PROGRAMMABLE_RESET_ADDRESS=1`. These options only affect the TF-A reset image, which is BL1 by default or BL31 if `RESET_TO_BL31=1`.

5.7.5 Using BL31 entrypoint as the reset address

On some platforms the runtime firmware (BL3x images) for the application processors are loaded by some firmware running on a secure system processor on the SoC, rather than by BL1 and BL2 running on the primary application processor. For this type of SoC it is desirable for the application processor to always reset to BL31 which eliminates the need for BL1 and BL2.

TF-A provides a build-time option `RESET_TO_BL31` that includes some additional logic in the BL31 entry point to support this use case.

In this configuration, the platform's Trusted Boot Firmware must ensure that BL31 is loaded to its runtime address, which must match the CPU's `RVBAR_EL3` reset vector base address, before the application processor is powered on. Additionally, platform software is responsible for loading the other BL3x images required and providing entry point information for them to BL31. Loading these images might be done by the Trusted Boot Firmware or by platform code in BL31.

Although the Arm FVP platform does not support programming the reset base address dynamically at run-time, it is possible to set the initial value of the `RVBAR_EL3` register at start-up. This feature is provided on the Base FVP only.

It allows the Arm FVP port to support the `RESET_TO_BL31` configuration, in which case the `bl31.bin` image must be loaded to its run address in Trusted SRAM and all CPU reset vectors be changed from the default `0x0` to this run address. See the *Arm Fixed Virtual Platforms (FVP)* for details of running the FVP models in this way.

Although technically it would be possible to program the reset base address with the right support in the SCP firmware, this is currently not implemented so the Juno port doesn't support the `RESET_TO_BL31` configuration.

The `RESET_TO_BL31` configuration requires some additions and changes in the BL31 functionality:

Determination of boot path

In this configuration, BL31 uses the same reset framework and code as the one described for BL1 above. Therefore, it is affected by the `PROGRAMMABLE_RESET_ADDRESS` and `COLD_BOOT_SINGLE_CPU` build options in the same way.

In the default, unoptimised BL31 reset flow, on a warm boot a CPU is directed to the PSCI implementation via a platform defined mechanism. On a cold boot, the platform must place any secondary CPUs into a safe state while the primary CPU executes a modified BL31 initialization, as described below.

Platform initialization

In this configuration, since the CPU resets to BL31, no parameters are expected to be passed to BL31 (see notes below for clarification). Instead, the platform code in BL31 needs to know, or be able to determine, the location of the BL32 (if required) and BL33 images and provide this information in response to the `bl31_plat_get_next_image_ep_info()` function.

Additionally, platform software is responsible for carrying out any security initialisation, for example programming a TrustZone address space controller. This might be done by the Trusted Boot Firmware or by platform code in BL31.

Note: Even though RESET_TO_BL31 is designed such that BL31 is the reset BL image, some platforms may wish to pass some arguments to BL31 as per the defined contract between BL31 and previous bootloaders. Previous bootloaders can pass arguments through registers x0 through x3. BL31 will preserve them and propagate them to platform code, which will handle these arguments in an IMPDEF manner.

5.8 Console Framework

The TF-A console framework is used to register consoles for different boot states so that user's output can be displayed on physical consoles throughout the different boot stages. The framework also supports debug mode for general debugging purposes.

The console framework supports a number of different UARTs, it is highly likely that the driver of the UART that is needed is already implemented. If not, a driver will need to be written for the new UART in TF-A. Current supported UARTs are:

- Amlogic Meson
- Arm PL011
- Cadence CDNS
- Coreboot CBMEM
- Marvell A3700
- **NXP**
  - i.MX LPUART
  - i.MX UART
  - Linflex
- Nvidia SPE
- Qualcomm UARTDM
- Renesas RCAR

- STMicroelectronics STM32
- Texas Instruments 16550

Note: The supported UART list is non-exhaustive. Check if the UART driver has already been written before writing a new one.

```
Console scopes and flags

Scope   : Flag
BOOT    : CONSOLE_FLAG_BOOT
RUNTIME : CONSOLE_FLAG_RUNTIME
CRASH   : CONSOLE_FLAG_CRASH
```

The console framework supports multiple consoles. Multiple instances of a UART can be registered at any given moment. Any registered console can have a single scope or multiple scopes. In single scope for example, setting three different consoles with each having BOOT, RUNTIME, and CRASH states respectively, the boot console will display only boot logs, the runtime console will display only the runtime output, while the crash console will be used to print the crash log in the event of a crash. Similarly, a console with all three scopes will display any and all output destined for BOOT, RUNTIME, or CRASH consoles.

These multiple scopes can be useful in many ways, for example:

- Having different consoles for Boot and Runtime messages
- Having a single console for both Runtime and Boot messages
- Having no runtime console at all and just having a single Boot console.
- Having a separate console for crash reporting when debugging.

5.8.1 Registering a console

To register a console in TF-A check if the hardware (UART) that is going to be used is already defined, if not we will need to define it, for example, the **PL011** UART driver API is defined in `include/drivers/arm/pl011.h`.

A skeleton console driver (assembly) is provided in TF-A `drivers/console/aarch64/skeleton_console.S`, this skeleton sets the rules for writing a new console_driver. Have a look at `drivers/arm/pl011/aarch64/pl011_console.S` for an actual implementation using this skeleton.

Function : console_xxx_register

```
Argument : console_t *, ...
Return   : int
```

This ASM Function is used to initialize and register a console. The caller needs to pass an empty `console_t` struct which *MUST* be allocated in persistent memory (e.g. a global or static local variable, *NOT* on the stack).

This function takes a `console_t` struct placed in x0 and additional arguments placed in x1 - x7. It returns x0 with either a 0 on failure or 1 on success.

See `console_pl011_register` ASM function for an implementation of this function.

Note: The `xxx` in the function name is replaced with the console driver name, for example, `console_xxx_register` becomes `console_pl011_register` in the driver for pl011.

Function : console_xxx_putc

```
Argument : int, console_t *
Return   : int
```

This ASM function is used to send a character to the UART's Transmit FIFO. It takes two arguments, a character as int stored in w0, and the `console_t` struct pointer stored in x1. It returns w0 with either the character on successs or a negative value on error. In a crash context this function must only clobber x0 - x2, x16 - x17.

See `console_pl011_putc` ASM function for an implementation.

Note: Avoid the direct use of this function for printing to the console, instead use the `debug.h` print macros, such as: VERBOSE(…), INFO(…), WARN(…), NOTICE(…) and ERROR(…).

Function : console_xxx_getc

```
Argument : console_t *
Return   : int
```

This ASM function is used to read a character from the receive FIFO. It takes a pointer to the console_struct as an argument and returns a character on success or a negative value below -2 on failure. This function is dependent on the `ENABLE_CONSOLE_GETC` flag, which is optional and is left to the platform because there may be security implications.

See `console_pl011_getc` ASM function for an implementation.

Function : console_xxx_flush

```
Argument : console_t *
Return   : void
```

This ASM function flushes any characters, that are still in the Transmit FIFO but haven't been printed yet to the console. It takes a pointer to the console_struct but doesn't return any value. In a crash context this function must only clobber x0 - x5, x16 - x17.

See `console_pl011_flush` ASM function for an implementation.

Macro : finish_console_register xxx putc=1 getc=ENABLE_CONSOLE_GETC flush=1

```
Callbacks
xxx      : name of the console driver
putc     : 0 for off, 1 to turn on putc function
getc     : 0 for off, ENABLE_CONSOLE_GETC to turn on the getc function
flush    : 0 for off, 1 to turn on flush function
```

This assembly macro function is called by the `console_xxx_register` to encapsulate the common setup that has to be done at the end of a console driver's register function. It takes `putc`, `getc` and `flush` macro arguments. It will register all of the driver's callbacks in the `console_t` struct and initialize the `flags` field (by default consoles are enabled for "boot" and "crash" states, this can be changed after registration using the `console_set_scope` function). This macro ends with a tail call that will include return to the caller.

This macro requires `console_t` pointer in x0 and a valid return address in x30.

See `include/arch/aarch64/console_macros.S`.

5.8.2 Registering a console using C

A console can be implemented in pure C, which is much easier than using assembly. Currently there is no C template for implementing a console driver in C but it can easily be implemented using existing examples. See `drivers/arm/dcc/dcc_console.c` for an implementation of a console driver in C.

The assembly functions in *Registering a console* section can be written in C when implementing a console driver using C.

> **Note:** A crash callback needs to be able to run without a stack. If crash mode support is required then the console driver will need to be written in Assembly (only the putc and flush functions are needed in a crash context).

5.8.3 Multi Console API

TF-A uses the multi-console API to manage the registered console instances and the characters print queue. This can be found in `drivers/console/multi_console.c`.

The multi-console API stores all registered consoles in a struct list `console_list`. Consoles can be removed from the console_list if no longer needed.

Consoles are registered with BOOT and CRASH scopes by default. These scopes can be changed after registration using `console_set_scope` function, as per the platform requirement.

This API also helps print characters to the specified consoles, characters can also be retrieved from the receive FIFO (this implementation is disabled by default but can be enabled if there is a need for it). The API can also help flush the transmit FIFO to get rid of any lingering characters from the queue when switching from secure world to the non-secure world.

The following functions are defined in the multi_console API.

Function : console_register()

```
Argument : console_t*
Return   : int
```

This function adds a console to the `console_list` declared in `include/drivers/console.h` and makes sure that there is only one instance of a specific console in this list. This function is called by the `finish_console_register` asm macro function, at the end of the console registration process.

This function always return 1. If the console is already present in the `console_list` it will return immediately with a value of 1, otherwise it will add the console to the `console_list` and then return 1.

Note: The `console_list` is a list of type `console_t`, it is an **extern** variable declared in `include/drivers/console.h`.

Function : console_unregister()

```
Argument : console_t*
Return   : console_t* or NULL
```

This function removes a console from the `console_list`. It will return the removed console on success or a `NULL` character upon failure.

Function : console_set_scope()

```
Argument : console_t*, int
Return   : void
```

This function is used to set the scope of the registered console. A console can be registered with upto three states (called the scope). These states are

- BOOT - set using the flag `CONSOLE_FLAG_BOOT`
- RUNTIME - set using the flag `CONSOLE_FLAG_RUNTIME`
- CRASH - set using the flag `CONSOLE_FLAG_CRASH`

It takes a pointer to the console and an int value (which is provided as the FLAG value) as its arguments. This function does not return anything.

Function : console_switch_state()

```
Argument : int
Return   : void
```

This function sets the console state (scope) for printing, i.e, TF-A will start sending all logs (INFO, WARNING, ERROR, NOTICE, VERBOSE) to the consoles that are registered with this new state (scope). For example, calling `console_switch_state(CONSOLE_FLAG_RUNTIME)`, TF-A will start sending all log messages to all consoles marked with the RUNTIME flag. BOOT is the default console state.

This function takes a console state as the function's only argument. This function does not return a value.

Function : console_putc()

```
Argument : int
Return   : int
```

Invoking this function sends a character to the `console->putc` (struct member) function of all consoles registered for the current scope, for example, BOOT logs will only be printed on consoles set with a BOOT scope. In the PL011 implementation `console->putc` call points to the `console_pl011_putc()` function.

This function takes the int value of a character as an argument and returns the int value of the character back on success or a negative int value on error.

> **Note:** Do not use this function in TF-A release builds, instead use the log prefixes, for example, `INFO("Print information here.")` to print messages on the active console.

Function : console_getc()

```
Argument : void
Return   : int
```

This function is used to fetch a character from the receive FIFO that has not been printed to the console yet. This function is disabled by default for security reasons but can be enabled using the `ENABLE_CONSOLE_GETC` macro if there is a need for it.

This function doesn't take any argument but returns a character as an int.

Function : console_flush()

```
Argument : void
Return   : void
```

This function flushes all the characters pending in the transmit FIFO of the active UART thus removing them from the print queue.

This function has no arguments and do not return a value.

Function : putchar()

```
Argument : int
Return   : int
```

This function overrides the weak implementation of the putchar library. It is used to send a character to the `console_putc()` function to be printed to the active console.

This function will either return the character on success or an **EOF** character otherwise.

5.9 Trusted Board Boot

The *Trusted Board Boot* (TBB) feature prevents malicious firmware from running on the platform by authenticating all firmware images up to and including the normal world bootloader. It does this by establishing a *Chain of Trust* using Public-Key-Cryptography Standards (PKCS).

This document describes the design of Trusted Firmware-A (TF-A) TBB, which is an implementation of the Trusted Board Boot Requirements (TBBR) specification, Arm DEN0006D. It should be used in conjunction with the *Firmware Update (FWU)* design document, which implements a specific aspect of the TBBR.

5.9.1 Chain of Trust

A Chain of Trust (CoT) starts with a set of implicitly trusted components, which are used to establish trust in the next layer of components, and so on, in a *chained* manner.

The chain of trust depends on several factors, including:

- The set of firmware images in use on this platform. Typically, most platforms share a common set of firmware images (BL1, BL2, BL31, BL33) but extra platform-specific images might be required.
- The key provisioning scheme: which keys need to programmed into the device and at which stage during the platform's manufacturing lifecycle.
- The key ownership model: who owns which key.

As these vary across platforms, chains of trust also vary across platforms. Although each platform is free to define its own CoT based on its needs, TF-A provides a set of "default" CoTs fitting some typical trust models, which platforms may reuse. The rest of this section presents general concepts which apply to all these default CoTs.

The implicitly trusted components forming the trust anchor are:

- A Root of Trust Public Key (ROTPK), or a hash of it.

  On Arm development platforms, a hash of the ROTPK (hash algorithm selected by the `HASH_ALG` build option, with sha256 as default) is stored in the trusted root-key storage registers. Alternatively, a development ROTPK might be used and its hash embedded into the BL1 and BL2 images (only for development purposes).

- The BL1 image, on the assumption that it resides in ROM so cannot be tampered with.

The remaining components in the CoT are either certificates or boot loader images. The certificates follow the X.509 v3 standard. This standard enables adding custom extensions to the certificates, which are used to store essential information to establish the CoT.

All certificates are self-signed. There is no need for a Certificate Authority (CA) because the CoT is not established by verifying the validity of a certificate's issuer but by the content of the certificate extensions. To sign the certificates, different signature schemes are available, please refer to the *Build Options* for more details.

The certificates are categorised as "Key" and "Content" certificates. Key certificates are used to verify public keys which have been used to sign content certificates. Content certificates are used to store the hash of a boot loader image. An image can be authenticated by calculating its hash and matching it with the hash extracted from the content certificate. Various hash algorithms are supported to calculate all hashes, please refer to the *Build Options* for more details. The public keys and hashes are included as non-standard extension fields in the X.509 v3 certificates.

The next sections now present specificities of each default CoT provided in TF-A.

Default CoT #1: TBBR

The *TBBR* CoT is named after the specification it follows to the letter.

In the TBBR CoT, all firmware binaries and certificates are (directly or indirectly) linked to the Root of Trust Public Key (ROTPK). Typically, the same vendor owns the ROTPK, the Trusted key and the Non-Trusted Key. Thus, this vendor is involved in signing every BL3x Key Certificate.

The keys used to establish this CoT are:

- **Root of trust key**

  The private part of this key is used to sign the trusted boot firmware certificate and the trusted key certificate. The public part is the ROTPK.

- **Trusted world key**

  The private part is used to sign the key certificates corresponding to the secure world images (SCP_BL2, BL31 and BL32). The public part is stored in one of the extension fields in the trusted key certificate.

- **Non-trusted world key**

  The private part is used to sign the key certificate corresponding to the non-secure world image (BL33). The public part is stored in one of the extension fields in the trusted key certificate.

- **BL3X keys**

  For each of SCP_BL2, BL31, BL32 and BL33, the private part is used to sign the content certificate for the BL3X image. The public part is stored in one of the extension fields in the corresponding key certificate.

The following images are included in the CoT:

- BL1
- BL2
- SCP_BL2 (optional)
- BL31
- BL33
- BL32 (optional)

The following certificates are used to authenticate the images.

- **Trusted boot firmware certificate**

  It is self-signed with the private part of the ROT key. It contains a hash of the BL2 image and hashes of various firmware configuration files (TB_FW_CONFIG, HW_CONFIG, FW_CONFIG).

- **Trusted key certificate**

  It is self-signed with the private part of the ROT key. It contains the public part of the trusted world key and the public part of the non-trusted world key.

- **SCP firmware key certificate**

It is self-signed with the trusted world key. It contains the public part of the SCP_BL2 key.

- **SCP firmware content certificate**

  It is self-signed with the SCP_BL2 key. It contains a hash of the SCP_BL2 image.

- **SoC firmware key certificate**

  It is self-signed with the trusted world key. It contains the public part of the BL31 key.

- **SoC firmware content certificate**

  It is self-signed with the BL31 key. It contains hashes of the BL31 image and its configuration file (SOC_FW_CONFIG).

- **Trusted OS key certificate**

  It is self-signed with the trusted world key. It contains the public part of the BL32 key.

- **Trusted OS content certificate**

  It is self-signed with the BL32 key. It contains hashes of the BL32 image(s) and its configuration file(s) (TOS_FW_CONFIG).

- **Non-trusted firmware key certificate**

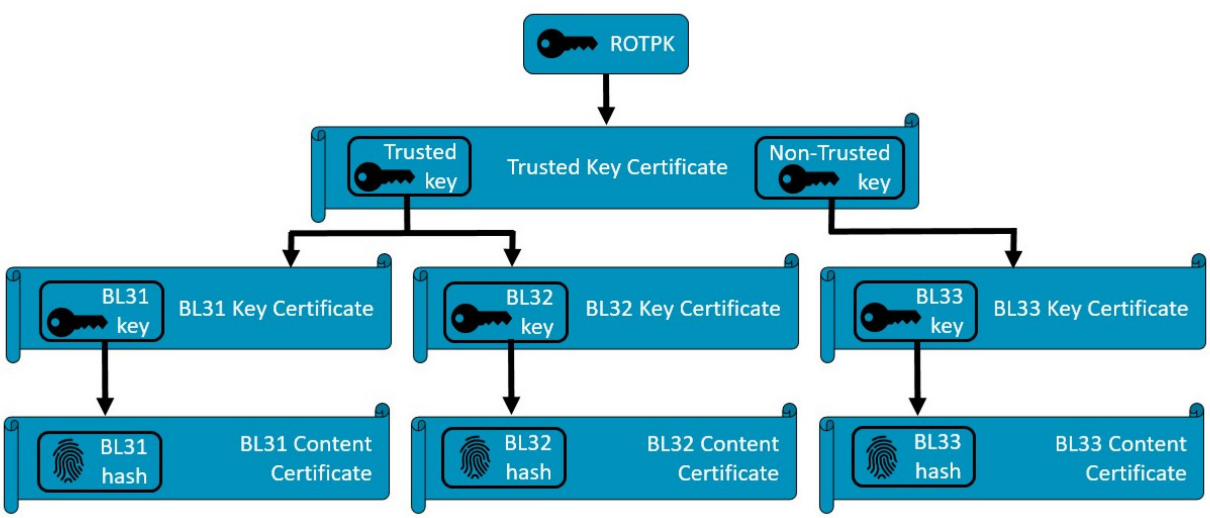
  It is self-signed with the non-trusted world key. It contains the public part of the BL33 key.

- **Non-trusted firmware content certificate**

  It is self-signed with the BL33 key. It contains hashes of the BL33 image and its configuration file (NT_FW_CONFIG).

The SCP firmware and Trusted OS certificates are optional, but they must be present if the corresponding SCP_BL2 or BL32 images are present.

The following diagram summarizes the part of the TBBR CoT enforced by BL2. Some images (SCP, debug certificates, secure partitions, configuration files) are not shown here for conciseness:

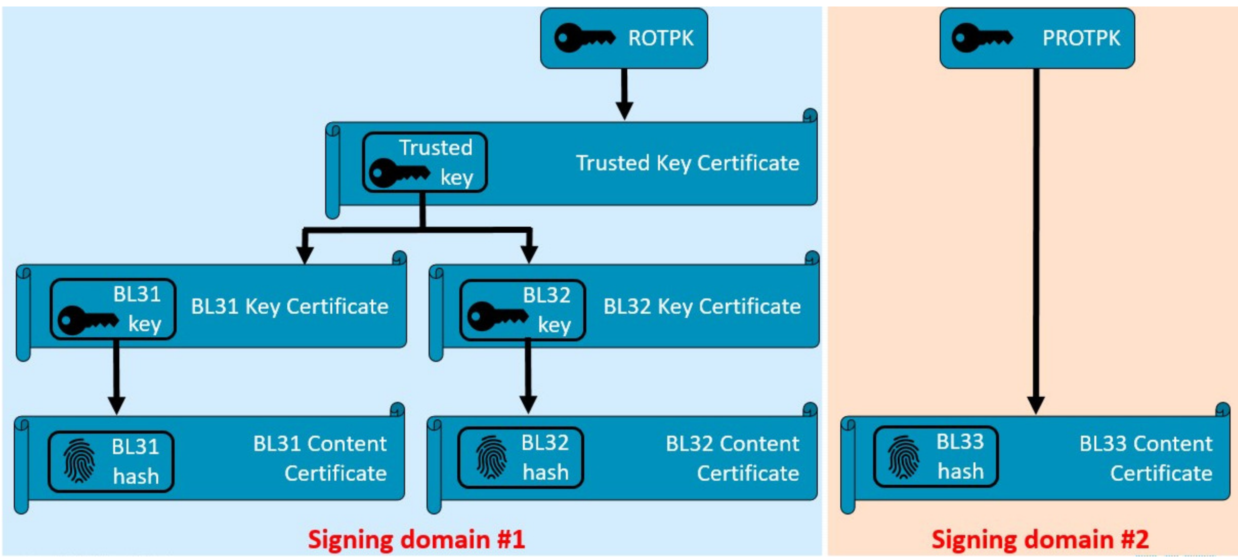


Default CoT #2: Dualroot

The *dualroot* CoT is targeted at systems where the Normal World firmware is owned by a different entity than the Secure World Firmware, and those 2 entities do not wish to share any keys or have any dependency between each other when it comes to signing their respective images. It establishes 2 separate signing domains, each with its own Root of Trust key. In that sense, this CoT has 2 roots of trust, hence the *dualroot* name.

Although the dualroot CoT reuses some of the TBBR CoT components and concepts, it differs on the BL33 image's chain of trust, which is rooted into a new key, called *Platform ROTPK*, or *PROTPK* for short.

The following diagram summarizes the part of the dualroot CoT enforced by BL2. Some images (SCP, debug certificates, secure partitions, configuration files) are not shown here for conciseness:



Default CoT #3: CCA

This CoT is targeted at Arm CCA systems. The Arm CCA security model recommends making supply chains for the Arm CCA firmware, the secure world firmware and the platform owner firmware, independent. Hence, this CoT has 3 roots of trust, one for each supply chain.

5.9.2 Trusted Board Boot Sequence

The CoT is verified through the following sequence of steps. The system panics if any of the steps fail.

- BL1 loads and verifies the BL2 content certificate. The issuer public key is read from the verified certificate. A hash of that key is calculated and compared with the hash of the ROTPK read from the trusted root-key storage registers. If they match, the BL2 hash is read from the certificate.

Note: The matching operation is platform specific and is currently unimplemented on the Arm development platforms.

- BL1 loads the BL2 image. Its hash is calculated and compared with the hash read from the certificate. Control is transferred to the BL2 image if all the comparisons succeed.
- BL2 loads and verifies the trusted key certificate. The issuer public key is read from the verified certificate. A hash of that key is calculated and compared with the hash of the ROTPK read from the trusted root-key storage registers. If the comparison succeeds, BL2 reads and saves the trusted and non-trusted world public keys from the verified certificate.

The next two steps are executed for each of the SCP_BL2, BL31 & BL32 images. The steps for the optional SCP_BL2 and BL32 images are skipped if these images are not present.

- BL2 loads and verifies the BL3x key certificate. The certificate signature is verified using the trusted world public key. If the signature verification succeeds, BL2 reads and saves the BL3x public key from the certificate.
- BL2 loads and verifies the BL3x content certificate. The signature is verified using the BL3x public key. If the signature verification succeeds, BL2 reads and saves the BL3x image hash from the certificate.

The next two steps are executed only for the BL33 image.

- BL2 loads and verifies the BL33 key certificate. If the signature verification succeeds, BL2 reads and saves the BL33 public key from the certificate.
- BL2 loads and verifies the BL33 content certificate. If the signature verification succeeds, BL2 reads and saves the BL33 image hash from the certificate.

The next step is executed for all the boot loader images.

- BL2 calculates the hash of each image. It compares it with the hash obtained from the corresponding content certificate. The image authentication succeeds if the hashes match.

The Trusted Board Boot implementation spans both generic and platform-specific BL1 and BL2 code, and in tool code on the host build machine. The feature is enabled through use of specific build flags as described in *Build Options*.

On the host machine, a tool generates the certificates, which are included in the FIP along with the boot loader images. These certificates are loaded in Trusted SRAM using the IO storage framework. They are then verified by an Authentication module included in TF-A.

The mechanism used for generating the FIP and the Authentication module are described in the following sections.

5.9.3 Authentication Framework

The authentication framework included in TF-A provides support to implement the desired trusted boot sequence. Arm platforms use this framework to implement the boot requirements specified in the Trusted Board Boot Requirements (TBBR) document.

More information about the authentication framework can be found in the *Authentication Framework & Chain of Trust* document.

5.9.4 Certificate Generation Tool

The `cert_create` tool is built and runs on the host machine as part of the TF-A build process when `GENERATE_COT=1`. It takes the boot loader images and keys as inputs and generates the certificates (in DER format) required to establish the CoT. The input keys must either be a file in PEM format or a PKCS11 URI in case a HSM is used. New keys can be generated by the tool in case they are not provided. The certificates are then passed as inputs to the `fiptool` utility for creating the FIP.

The certificates are also stored individually in the output build directory.

The tool resides in the `tools/cert_create` directory. It uses the OpenSSL SSL library version to generate the X.509 certificates. The specific version of the library that is required is given in the *Prerequisites* document.

Instructions for building and using the tool can be found at *Building the Certificate Generation Tool*.

5.9.5 Authenticated Encryption Framework

The authenticated encryption framework included in TF-A provides support to implement the optional firmware encryption feature. This feature can be optionally enabled on platforms to implement the optional requirement: R060_TBBR_FUNCTION as specified in the Trusted Board Boot Requirements (TBBR) document.

5.9.6 Firmware Encryption Tool

The `encrypt_fw` tool is built and runs on the host machine as part of the TF-A build process when `DECRYPTION_SUPPORT != none`. It takes the plain firmware image as input and generates the encrypted firmware image which can then be passed as input to the `fiptool` utility for creating the FIP.

The encrypted firmwares are also stored individually in the output build directory.

The tool resides in the `tools/encrypt_fw` directory. It uses OpenSSL SSL library version 1.0.1 or later to do authenticated encryption operation. Instructions for building and using the tool can be found in the *Building the Firmware Encryption Tool*.

5.10 Building FIP images with support for Trusted Board Boot

Trusted Board Boot primarily consists of the following two features:

- Image Authentication, described in *Trusted Board Boot*, and
- Firmware Update, described in *Firmware Update (FWU)*

The following steps should be followed to build FIP and (optionally) FWU_FIP images with support for these features:

1. Fulfill the dependencies of the `mbedtls` cryptographic and image parser modules by checking out a recent version of the mbed TLS Repository. It is important to use a version that is compatible with TF-A and fixes any known security vulnerabilities. See mbed TLS Security Center for more information. See the *Prerequisites* document for the appropriate version of mbed TLS to use.

   The `drivers/auth/mbedtls/mbedtls_*.mk` files contain the list of mbed TLS source files the modules depend upon. `include/drivers/auth/mbedtls/mbedtls_config.h` contains the configuration options required to build the mbed TLS sources.

   Note that the mbed TLS library is licensed under the Apache version 2.0 license. Using mbed TLS source code will affect the licensing of TF-A binaries that are built using this library.

2. To build the FIP image, ensure the following command line variables are set while invoking `make` to build TF-A:

   - `TRUSTED_BOARD_BOOT=1`
   - `GENERATE_COT=1`

   By default, this will use the Chain of Trust described in the TBBR-client document. To select a different one, use the `COT` build option.

   If using a custom build of OpenSSL, set the `OPENSSL_DIR` variable accordingly so it points at the OpenSSL installation path, as explained in *Build Options*. In addition, set the `LD_LIBRARY_PATH` variable when running to point at the custom OpenSSL path, so the OpenSSL libraries are loaded from that path instead of the default OS path. Export this variable if necessary.

   In the case of Arm platforms, the location of the ROTPK must also be specified at build time. The following locations are currently supported (see `ARM_ROTPK_LOCATION` build option):

   - `ARM_ROTPK_LOCATION=regs`: the ROTPK hash is obtained from the Trusted root-key storage registers present in the platform. On Juno, these registers are read-only. On FVP Base and Cortex models, the registers are also read-only, but the value can be specified using the command line option `bp.trusted_key_storage.public_key` when launching the model. On Juno board, the default value corresponds to an ECDSA-SECP256R1 public key hash, whose private part is not currently available.
   - `ARM_ROTPK_LOCATION=devel_rsa`: the ROTPK is a hash of the RSA public key corresponding to the private key specified by `ROT_KEY`. If `ROT_KEY` is not specified, the private key is the development key `plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem`. There are also 3k and 4k RSA development keys in `plat/arm/board/common/rotpk/`. The hashing algorithm is selected by `HASH_ALG`; sha256 is used if `HASH_ALG` is not specified.
   - `ARM_ROTPK_LOCATION=devel_ecdsa`: the ROTPK is a hash of the ECDSA public key corresponding to the private key specified by `ROT_KEY`. If `ROT_KEY` is not specified, the private key is the development key `plat/arm/board/common/rotpk/arm_rotprivk_ecdsa.pem` by default, a 384 bit key `plat/arm/board/common/rotpk/arm_rotprivk_ecdsa_secp384r1.pem` also exists, and can be specified by `ROT_KEY`. The hashing algorithm is selected by `HASH_ALG`; sha256 is used if `HASH_ALG` is not specified.
   - `ARM_ROTPK_LOCATION=devel_full_dev_rsa_key`: the ROTPK is an unhashed RSA public key corresponding to the private key specified by `ROT_KEY`. If `ROT_KEY` is not

specified, the private key is the development key `plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem`. There are also 3k and 4k RSA development keys in `plat/arm/board/common/rotpk/`.

- `ARM_ROTPK_LOCATION=devel_full_dev_ecdsa_key`: the ROTPK is an unhashed RSA public key corresponding to the private key specified by `ROT_KEY`. If `ROT_KEY` is not specified, the private key is the development key `plat/arm/board/common/rotpk/arm_rotprivk_ecdsa.pem`, a 384 bit key `plat/arm/board/common/rotpk/arm_rotprivk_ecdsa_secp384r1.pem` also exists, and can be specified by `ROT_KEY`.

Example of command line using RSA development keys:

```
make PLAT=<platform> TRUSTED_BOARD_BOOT=1 GENERATE_COT=1        \
ARM_ROTPK_LOCATION=devel_rsa                                    \
ROT_KEY=plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem        \
BL33=<path-to>/<bl33_image> OPENSSL_DIR=<path-to>/<openssl>     \
all fip
```

The result of this build will be the bl1.bin and the fip.bin binaries. This FIP will include the certificates corresponding to the selected Chain of Trust. These certificates can also be found in the output build directory.

3. The optional FWU_FIP contains any additional images to be loaded from Non-Volatile storage during the *Firmware Update (FWU)* process. To build the FWU_FIP, any FWU images required by the platform must be specified on the command line. On Arm development platforms like Juno, these are:

   - NS_BL2U. The AP non-secure Firmware Updater image.
   - SCP_BL2U. The SCP Firmware Update Configuration image.

   Example of Juno command line for generating both `fwu` and `fwu_fip` targets using RSA development:

```
make PLAT=juno TRUSTED_BOARD_BOOT=1 GENERATE_COT=1              \
ARM_ROTPK_LOCATION=devel_rsa                                    \
ROT_KEY=plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem        \
BL33=<path-to>/<bl33_image> OPENSSL_DIR=<path-to>/<openssl>     \
SCP_BL2=<path-to>/<scp_bl2_image>                               \
SCP_BL2U=<path-to>/<scp_bl2u_image>                             \
NS_BL2U=<path-to>/<ns_bl2u_image>                               \
all fip fwu_fip
```

Note: The BL2U image will be built by default and added to the FWU_FIP. The user may override this by adding `BL2U=<path-to>/<bl2u_image>` to the command line above.

Note: Building and installing the non-secure and SCP FWU images (NS_BL1U, NS_BL2U and SCP_BL2U) is outside the scope of this document.

The result of this build will be bl1.bin, fip.bin and fwu_fip.bin binaries. Both the FIP and FWU_FIP will include the certificates corresponding to the selected Chain of Trust. These certificates can also be found in the output build directory.

CHAPTER

SIX

PORTING GUIDE

6.1 Introduction

Porting Trusted Firmware-A (TF-A) to a new platform involves making some mandatory and optional modifications for both the cold and warm boot paths. Modifications consist of:

- Implementing a platform-specific function or variable,
- Setting up the execution context in a certain way, or
- Defining certain constants (for example #defines).

The platform-specific functions and variables are declared in `include/plat/common/platform.h`. The firmware provides a default implementation of variables and functions to fulfill the optional requirements in order to ease the porting effort. Each platform port can use them as is or provide their own implementation if the default implementation is inadequate.

> **Note:** TF-A historically provided default implementations of platform interfaces as *weak* functions. This practice is now discouraged and new platform interfaces as they get introduced in the code base should be *strongly* defined. We intend to convert existing weak functions over time. Until then, you will find references to *weak* functions in this document.

Please review the *Threat Model* documents as part of the porting effort. Some platform interfaces play a key role in mitigating against some of the threats. Failing to fulfill these expectations could undermine the security guarantees offered by TF-A. These platform responsibilities are highlighted in the threat assessment section, under the "*Mitigations implemented?*" box for each threat.

Some modifications are common to all Boot Loader (BL) stages. Section 2 discusses these in detail. The subsequent sections discuss the remaining modifications for each BL stage in detail.

Please refer to the *Platform Ports Policy* for the policy regarding compatibility and deprecation of these porting interfaces.

Only Arm development platforms (such as FVP and Juno) may use the functions/definitions in `include/plat/arm/common/` and the corresponding source files in `plat/arm/common/`. This is done so that there are no dependencies between platforms maintained by different people/companies. If you want to use any of the functionality present in `plat/arm` files, please propose a patch that moves the code to `plat/common` so that it can be discussed.

6.2 Common modifications

This section covers the modifications that should be made by the platform for each BL stage to correctly port the firmware stack. They are categorized as either mandatory or optional.

6.3 Common mandatory modifications

A platform port must enable the Memory Management Unit (MMU) as well as the instruction and data caches for each BL stage. Setting up the translation tables is the responsibility of the platform port because memory maps differ across platforms. A memory translation library (see `lib/xlat_tables_v2/`) is provided to help in this setup.

Note that although this library supports non-identity mappings, this is intended only for re-mapping peripheral physical addresses and allows platforms with high I/O addresses to reduce their virtual address space. All other addresses corresponding to code and data must currently use an identity mapping.

Also, the only translation granule size supported in TF-A is 4KB, as various parts of the code assume that is the case. It is not possible to switch to 16 KB or 64 KB granule sizes at the moment.

In Arm standard platforms, each BL stage configures the MMU in the platform-specific architecture setup function, `blX_plat_arch_setup()`, and uses an identity mapping for all addresses.

If the build option `USE_COHERENT_MEM` is enabled, each platform can allocate a block of identity mapped secure memory with Device-nGnRE attributes aligned to page boundary (4K) for each BL stage. All sections which allocate coherent memory are grouped under `.coherent_ram`. For ex: Bakery locks are placed in a section identified by name `.bakery_lock` inside `.coherent_ram` so that its possible for the firmware to place variables in it using the following C code directive:

```
__section(".bakery_lock")
```

Or alternatively the following assembler code directive:

```
.section .bakery_lock
```

The `.coherent_ram` section is a sum of all sections like `.bakery_lock` which are used to allocate any data structures that are accessed both when a CPU is executing with its MMU and caches enabled, and when it's running with its MMU and caches disabled. Examples are given below.

The following variables, functions and constants must be defined by the platform for the firmware to work correctly.

6.3.1 File : platform_def.h [mandatory]

Each platform must ensure that a header file of this name is in the system include path with the following constants defined. This will require updating the list of `PLAT_INCLUDES` in the `platform.mk` file.

Platform ports may optionally use the file `include/plat/common/common_def.h`, which provides typical values for some of the constants below. These values are likely to be suitable for all platform ports.

- **#define : PLATFORM_LINKER_FORMAT**

  Defines the linker format used by the platform, for example `elf64-littleaarch64`.

- **#define : PLATFORM_LINKER_ARCH**

  Defines the processor architecture for the linker by the platform, for example `aarch64`.

- **#define : PLATFORM_STACK_SIZE**

  Defines the normal stack memory available to each CPU. This constant is used by `plat/common/aarch64/platform_mp_stack.S` and `plat/common/aarch64/platform_up_stack.S`.

- **#define : CACHE_WRITEBACK_GRANULE**

  Defines the size in bytes of the largest cache line across all the cache levels in the platform.

- **#define : FIRMWARE_WELCOME_STR**

  Defines the character string printed by BL1 upon entry into the `bl1_main()` function.

- **#define : PLATFORM_CORE_COUNT**

  Defines the total number of CPUs implemented by the platform across all clusters in the system.

- **#define : PLAT_NUM_PWR_DOMAINS**

  Defines the total number of nodes in the power domain topology tree at all the power domain levels used by the platform. This macro is used by the PSCI implementation to allocate data structures to represent power domain topology.

- **#define : PLAT_MAX_PWR_LVL**

  Defines the maximum power domain level that the power management operations should apply to. More often, but not always, the power domain level corresponds to affinity level. This macro allows the PSCI implementation to know the highest power domain level that it should consider for power management operations in the system that the platform implements. For example, the Base AEM FVP implements two clusters with a configurable number of CPUs and it reports the maximum power domain level as 1.

- **#define : PLAT_MAX_OFF_STATE**

  Defines the local power state corresponding to the deepest power down possible at every power domain level in the platform. The local power states for each level may be sparsely allocated between 0 and this value with 0 being reserved for the RUN state. The PSCI implementation uses this value to initialize the local power states of the power domain nodes and to specify the requested power state for a PSCI_CPU_OFF call.

- **#define : PLAT_MAX_RET_STATE**

Defines the local power state corresponding to the deepest retention state possible at every power domain level in the platform. This macro should be a value less than PLAT_MAX_OFF_STATE and greater than 0. It is used by the PSCI implementation to distinguish between retention and power down local power states within PSCI_CPU_SUSPEND call.

- **#define : PLAT_MAX_PWR_LVL_STATES**

  Defines the maximum number of local power states per power domain level that the platform supports. The default value of this macro is 2 since most platforms just support a maximum of two local power states at each power domain level (power-down and retention). If the platform needs to account for more local power states, then it must redefine this macro.

  Currently, this macro is used by the Generic PSCI implementation to size the array used for PSCI_STAT_COUNT/RESIDENCY accounting.

- **#define : BL1_RO_BASE**

  Defines the base address in secure ROM where BL1 originally lives. Must be aligned on a page-size boundary.

- **#define : BL1_RO_LIMIT**

  Defines the maximum address in secure ROM that BL1's actual content (i.e. excluding any data section allocated at runtime) can occupy.

- **#define : BL1_RW_BASE**

  Defines the base address in secure RAM where BL1's read-write data will live at runtime. Must be aligned on a page-size boundary.

- **#define : BL1_RW_LIMIT**

  Defines the maximum address in secure RAM that BL1's read-write data can occupy at runtime.

- **#define : BL2_BASE**

  Defines the base address in secure RAM where BL1 loads the BL2 binary image. Must be aligned on a page-size boundary. This constant is not applicable when BL2_IN_XIP_MEM is set to '1'.

- **#define : BL2_LIMIT**

  Defines the maximum address in secure RAM that the BL2 image can occupy. This constant is not applicable when BL2_IN_XIP_MEM is set to '1'.

- **#define : BL2_RO_BASE**

  Defines the base address in secure XIP memory where BL2 RO section originally lives. Must be aligned on a page-size boundary. This constant is only needed when BL2_IN_XIP_MEM is set to '1'.

- **#define : BL2_RO_LIMIT**

  Defines the maximum address in secure XIP memory that BL2's actual content (i.e. excluding any data section allocated at runtime) can occupy. This constant is only needed when BL2_IN_XIP_MEM is set to '1'.

- **#define : BL2_RW_BASE**

Defines the base address in secure RAM where BL2's read-write data will live at runtime. Must be aligned on a page-size boundary. This constant is only needed when BL2_IN_XIP_MEM is set to '1'.

- **#define : BL2_RW_LIMIT**

  Defines the maximum address in secure RAM that BL2's read-write data can occupy at runtime. This constant is only needed when BL2_IN_XIP_MEM is set to '1'.

- **#define : BL31_BASE**

  Defines the base address in secure RAM where BL2 loads the BL31 binary image. Must be aligned on a page-size boundary.

- **#define : BL31_LIMIT**

  Defines the maximum address in secure RAM that the BL31 image can occupy.

- **#define : PLAT_RSE_COMMS_PAYLOAD_MAX_SIZE**

  Defines the maximum message size between AP and RSE. Need to define if platform supports RSE.

For every image, the platform must define individual identifiers that will be used by BL1 or BL2 to load the corresponding image into memory from non-volatile storage. For the sake of performance, integer numbers will be used as identifiers. The platform will use those identifiers to return the relevant information about the image to be loaded (file handler, load address, authentication information, etc.). The following image identifiers are mandatory:

- **#define : BL2_IMAGE_ID**

  BL2 image identifier, used by BL1 to load BL2.

- **#define : BL31_IMAGE_ID**

  BL31 image identifier, used by BL2 to load BL31.

- **#define : BL33_IMAGE_ID**

  BL33 image identifier, used by BL2 to load BL33.

If Trusted Board Boot is enabled, the following certificate identifiers must also be defined:

- **#define : TRUSTED_BOOT_FW_CERT_ID**

  BL2 content certificate identifier, used by BL1 to load the BL2 content certificate.

- **#define : TRUSTED_KEY_CERT_ID**

  Trusted key certificate identifier, used by BL2 to load the trusted key certificate.

- **#define : SOC_FW_KEY_CERT_ID**

  BL31 key certificate identifier, used by BL2 to load the BL31 key certificate.

- **#define : SOC_FW_CONTENT_CERT_ID**

  BL31 content certificate identifier, used by BL2 to load the BL31 content certificate.

- **#define : NON_TRUSTED_FW_KEY_CERT_ID**

  BL33 key certificate identifier, used by BL2 to load the BL33 key certificate.

- **#define : NON_TRUSTED_FW_CONTENT_CERT_ID**

  BL33 content certificate identifier, used by BL2 to load the BL33 content certificate.

- **#define : FWU_CERT_ID**

  Firmware Update (FWU) certificate identifier, used by NS_BL1U to load the FWU content certificate.

If the AP Firmware Updater Configuration image, BL2U is used, the following must also be defined:

- **#define : BL2U_BASE**

  Defines the base address in secure memory where BL1 copies the BL2U binary image. Must be aligned on a page-size boundary.

- **#define : BL2U_LIMIT**

  Defines the maximum address in secure memory that the BL2U image can occupy.

- **#define : BL2U_IMAGE_ID**

  BL2U image identifier, used by BL1 to fetch an image descriptor corresponding to BL2U.

If the SCP Firmware Update Configuration Image, SCP_BL2U is used, the following must also be defined:

- **#define : SCP_BL2U_IMAGE_ID**

  SCP_BL2U image identifier, used by BL1 to fetch an image descriptor corresponding to SCP_BL2U.

  Note: TF-A does not provide source code for this image.

If the Non-Secure Firmware Updater ROM, NS_BL1U is used, the following must also be defined:

- **#define : NS_BL1U_BASE**

  Defines the base address in non-secure ROM where NS_BL1U executes. Must be aligned on a page-size boundary.

  Note: TF-A does not provide source code for this image.

- **#define : NS_BL1U_IMAGE_ID**

  NS_BL1U image identifier, used by BL1 to fetch an image descriptor corresponding to NS_BL1U.

If the Non-Secure Firmware Updater, NS_BL2U is used, the following must also be defined:

- **#define : NS_BL2U_BASE**

  Defines the base address in non-secure memory where NS_BL2U executes. Must be aligned on a page-size boundary.

  Note: TF-A does not provide source code for this image.

- **#define : NS_BL2U_IMAGE_ID**

  NS_BL2U image identifier, used by BL1 to fetch an image descriptor corresponding to NS_BL2U.

For the the Firmware update capability of TRUSTED BOARD BOOT, the following macros may also be defined:

- **#define : PLAT_FWU_MAX_SIMULTANEOUS_IMAGES**

  Total number of images that can be loaded simultaneously. If the platform doesn't specify any value, it defaults to 10.

If a SCP_BL2 image is supported by the platform, the following constants must also be defined:

- **#define : SCP_BL2_IMAGE_ID**

  SCP_BL2 image identifier, used by BL2 to load SCP_BL2 into secure memory from platform storage before being transferred to the SCP.

- **#define : SCP_FW_KEY_CERT_ID**

  SCP_BL2 key certificate identifier, used by BL2 to load the SCP_BL2 key certificate (mandatory when Trusted Board Boot is enabled).

- **#define : SCP_FW_CONTENT_CERT_ID**

  SCP_BL2 content certificate identifier, used by BL2 to load the SCP_BL2 content certificate (mandatory when Trusted Board Boot is enabled).

If a BL32 image is supported by the platform, the following constants must also be defined:

- **#define : BL32_IMAGE_ID**

  BL32 image identifier, used by BL2 to load BL32.

- **#define : TRUSTED_OS_FW_KEY_CERT_ID**

  BL32 key certificate identifier, used by BL2 to load the BL32 key certificate (mandatory when Trusted Board Boot is enabled).

- **#define : TRUSTED_OS_FW_CONTENT_CERT_ID**

  BL32 content certificate identifier, used by BL2 to load the BL32 content certificate (mandatory when Trusted Board Boot is enabled).

- **#define : BL32_BASE**

  Defines the base address in secure memory where BL2 loads the BL32 binary image. Must be aligned on a page-size boundary.

- **#define : BL32_LIMIT**

  Defines the maximum address that the BL32 image can occupy.

If the Test Secure-EL1 Payload (TSP) instantiation of BL32 is supported by the platform, the following constants must also be defined:

- **#define : TSP_SEC_MEM_BASE**

Defines the base address of the secure memory used by the TSP image on the platform. This must be at the same address or below `BL32_BASE`.

- **#define : TSP_SEC_MEM_SIZE**

  Defines the size of the secure memory used by the BL32 image on the platform. `TSP_SEC_MEM_BASE` and `TSP_SEC_MEM_SIZE` must fully accommodate the memory required by the BL32 image, defined by `BL32_BASE` and `BL32_LIMIT`.

- **#define : TSP_IRQ_SEC_PHY_TIMER**

  Defines the ID of the secure physical generic timer interrupt used by the TSP's interrupt handling code.

If the platform port uses the translation table library code, the following constants must also be defined:

- **#define : PLAT_XLAT_TABLES_DYNAMIC**

  Optional flag that can be set per-image to enable the dynamic allocation of regions even when the MMU is enabled. If not defined, only static functionality will be available, if defined and set to 1 it will also include the dynamic functionality.

- **#define : MAX_XLAT_TABLES**

  Defines the maximum number of translation tables that are allocated by the translation table library code. To minimize the amount of runtime memory used, choose the smallest value needed to map the required virtual addresses for each BL stage. If `PLAT_XLAT_TABLES_DYNAMIC` flag is enabled for a BL image, `MAX_XLAT_TABLES` must be defined to accommodate the dynamic regions as well.

- **#define : MAX_MMAP_REGIONS**

  Defines the maximum number of regions that are allocated by the translation table library code. A region consists of physical base address, virtual base address, size and attributes (Device/Memory, RO/RW, Secure/Non-Secure), as defined in the `mmap_region_t` structure. The platform defines the regions that should be mapped. Then, the translation table library will create the corresponding tables and descriptors at runtime. To minimize the amount of runtime memory used, choose the smallest value needed to register the required regions for each BL stage. If `PLAT_XLAT_TABLES_DYNAMIC` flag is enabled for a BL image, `MAX_MMAP_REGIONS` must be defined to accommodate the dynamic regions as well.

- **#define : PLAT_VIRT_ADDR_SPACE_SIZE**

  Defines the total size of the virtual address space in bytes. For example, for a 32 bit virtual address space, this value should be `(1ULL << 32)`.

- **#define : PLAT_PHY_ADDR_SPACE_SIZE**

  Defines the total size of the physical address space in bytes. For example, for a 32 bit physical address space, this value should be `(1ULL << 32)`.

If the platform port uses the IO storage framework, the following constants must also be defined:

- **#define : MAX_IO_DEVICES**

  Defines the maximum number of registered IO devices. Attempting to register more devices than this value using `io_register_device()` will fail with -ENOMEM.

- **#define : MAX_IO_HANDLES**

Defines the maximum number of open IO handles. Attempting to open more IO entities than this value using `io_open()` will fail with -ENOMEM.

- **#define : MAX_IO_BLOCK_DEVICES**

  Defines the maximum number of registered IO block devices. Attempting to register more devices this value using `io_dev_open()` will fail with -ENOMEM. MAX_IO_BLOCK_DEVICES should be less than MAX_IO_DEVICES. With this macro, multiple block devices could be supported at the same time.

If the platform needs to allocate data within the per-cpu data framework in BL31, it should define the following macro. Currently this is only required if the platform decides not to use the coherent memory section by undefining the `USE_COHERENT_MEM` build flag. In this case, the framework allocates the required memory within the the per-cpu data to minimize wastage.

- **#define : PLAT_PCPU_DATA_SIZE**

  Defines the memory (in bytes) to be reserved within the per-cpu data structure for use by the platform layer.

The following constants are optional. They should be defined when the platform memory layout implies some image overlaying like in Arm standard platforms.

- **#define : BL31_PROGBITS_LIMIT**

  Defines the maximum address in secure RAM that the BL31's progbits sections can occupy.

- **#define : TSP_PROGBITS_LIMIT**

  Defines the maximum address that the TSP's progbits sections can occupy.

If the platform supports OS-initiated mode, i.e. the build option `PSCI_OS_INIT_MODE` is enabled, and if the platform's maximum power domain level for PSCI_CPU_SUSPEND differs from `PLAT_MAX_PWR_LVL`, the following constant must be defined.

- **#define : PLAT_MAX_CPU_SUSPEND_PWR_LVL**

  Defines the maximum power domain level that PSCI_CPU_SUSPEND should apply to.

If the platform port uses the PL061 GPIO driver, the following constant may optionally be defined:

- **PLAT_PL061_MAX_GPIOS** Maximum number of GPIOs required by the platform. This allows control how much memory is allocated for PL061 GPIO controllers. The default value is

  1. $(eval $(call add_define,PLAT_PL061_MAX_GPIOS))

If the platform port uses the partition driver, the following constant may optionally be defined:

- **PLAT_PARTITION_MAX_ENTRIES** Maximum number of partition entries required by the platform. This allows control how much memory is allocated for partition entries. The default value is 128. For example, define the build flag in `platform.mk`: PLAT_PARTITION_MAX_ENTRIES := 12 $(eval $(call add_define,PLAT_PARTITION_MAX_ENTRIES))
- **PLAT_PARTITION_BLOCK_SIZE** The size of partition block. It could be either 512 bytes or 4096 bytes. The default value is 512. For example, define the build flag in `platform.mk`: PLAT_PARTITION_BLOCK_SIZE := 4096 $(eval $(call add_define,PLAT_PARTITION_BLOCK_SIZE))

If the platform port uses the Arm® Ethos™-N NPU driver, the following configuration must be performed:

- The NPU SiP service handler must be hooked up. This consists of both the initial setup (`ethosn_smc_setup`) and the handler itself (`ethosn_smc_handler`)

If the platform port uses the Arm® Ethos™-N NPU driver with TZMP1 support enabled, the following constants and configuration must also be defined:

- **ETHOSN_NPU_PROT_FW_NSAID**

  Defines the Non-secure Access IDentity (NSAID) that the NPU shall use to access the protected memory that contains the NPU's firmware.

- **ETHOSN_NPU_PROT_DATA_RW_NSAID**

  Defines the Non-secure Access IDentity (NSAID) that the NPU shall use for read/write access to the protected memory that contains inference data.

- **ETHOSN_NPU_PROT_DATA_RO_NSAID**

  Defines the Non-secure Access IDentity (NSAID) that the NPU shall use for read-only access to the protected memory that contains inference data.

- **ETHOSN_NPU_NS_RW_DATA_NSAID**

  Defines the Non-secure Access IDentity (NSAID) that the NPU shall use for read/write access to the non-protected memory.

- **ETHOSN_NPU_NS_RO_DATA_NSAID**

  Defines the Non-secure Access IDentity (NSAID) that the NPU shall use for read-only access to the non-protected memory.

- **ETHOSN_NPU_FW_IMAGE_BASE** and **ETHOSN_NPU_FW_IMAGE_LIMIT**

  Defines the physical address range that the NPU's firmware will be loaded into and executed from.

- Configure the platforms TrustZone Controller (TZC) with appropriate regions of protected memory. At minimum this must include a region for the NPU's firmware code and a region for protected inference data, and these must be accessible using the NSAIDs defined above.
- Include the NPU firmware and certificates in the FIP.
- Provide FCONF entries to configure the image source for the NPU firmware and certificates.
- Add MMU mappings such that:
- BL2 can write the NPU firmware into the region defined by `ETHOSN_NPU_FW_IMAGE_BASE` and `ETHOSN_NPU_FW_IMAGE_LIMIT`
- BL31 (SiP service) can read the NPU firmware from the same region
- Add the firmware image ID `ETHOSN_NPU_FW_IMAGE_ID` to the list of images loaded by BL2.

Please see the reference implementation code for the Juno platform as an example.

The following constant is optional. It should be defined to override the default behaviour of the `assert()` function (for example, to save memory).

- **PLAT_LOG_LEVEL_ASSERT** If `PLAT_LOG_LEVEL_ASSERT` is higher or equal than `LOG_LEVEL_VERBOSE`, `assert()` prints the name of the file, the line number and the asserted expression. Else if it is higher than `LOG_LEVEL_INFO`, it prints the file name and the line number. Else if it is lower than `LOG_LEVEL_INFO`, it doesn't print anything to the console. If `PLAT_LOG_LEVEL_ASSERT` isn't defined, it defaults to `LOG_LEVEL`.

If the platform port uses the DRTM feature, the following constants must be defined:

- **#define : PLAT_DRTM_EVENT_LOG_MAX_SIZE**

  Maximum Event Log size used by the platform. Platform can decide the maximum size of the Event Log buffer, depending upon the highest hash algorithm chosen and the number of components selected to measure during the DRTM execution flow.

- **#define : PLAT_DRTM_MMAP_ENTRIES**

  Number of the MMAP entries used by the DRTM implementation to calculate the size of address map region of the platform.

6.3.2 File : plat_macros.S [mandatory]

Each platform must ensure a file of this name is in the system include path with the following macro defined. In the Arm development platforms, this file is found in `plat/arm/board/<plat_name>/include/plat_macros.S`.

- **Macro : plat_crash_print_regs**

  This macro allows the crash reporting routine to print relevant platform registers in case of an unhandled exception in BL31. This aids in debugging and this macro can be defined to be empty in case register reporting is not desired.

  For instance, GIC or interconnect registers may be helpful for troubleshooting.

6.4 Handling Reset

BL1 by default implements the reset vector where execution starts from a cold or warm boot. BL31 can be optionally set as a reset vector using the `RESET_TO_BL31` make variable.

For each CPU, the reset vector code is responsible for the following tasks:

1. Distinguishing between a cold boot and a warm boot.
2. In the case of a cold boot and the CPU being a secondary CPU, ensuring that the CPU is placed in a platform-specific state until the primary CPU performs the necessary steps to remove it from this state.
3. In the case of a warm boot, ensuring that the CPU jumps to a platform- specific address in the BL31 image in the same processor mode as it was when released from reset.

The following functions need to be implemented by the platform port to enable reset vector code to perform the above tasks.

6.4.1 Function : plat_get_my_entrypoint() [mandatory when PROGRAMMABLE_RESET_ADDRESS == 0]

```
Argument : void
Return   : uintptr_t
```

This function is called with the MMU and caches disabled (`SCTLR_EL3.M` = 0 and `SCTLR_EL3.C` = 0). The function is responsible for distinguishing between a warm and cold reset for the current CPU using platform-specific means. If it's a warm reset, then it returns the warm reset entrypoint point provided to `plat_setup_psci_ops()` during BL31 initialization. If it's a cold reset then this function must return zero.

This function does not follow the Procedure Call Standard used by the Application Binary Interface for the Arm 64-bit architecture. The caller should not assume that callee saved registers are preserved across a call to this function.

This function fulfills requirement 1 and 3 listed above.

Note that for platforms that support programming the reset address, it is expected that a CPU will start executing code directly at the right address, both on a cold and warm reset. In this case, there is no need to identify the type of reset nor to query the warm reset entrypoint. Therefore, implementing this function is not required on such platforms.

6.4.2 Function : plat_secondary_cold_boot_setup() [mandatory when COLD_BOOT_SINGLE_CPU == 0]

```
Argument : void
```

This function is called with the MMU and data caches disabled. It is responsible for placing the executing secondary CPU in a platform-specific state until the primary CPU performs the necessary actions to bring it out of that state and allow entry into the OS. This function must not return.

In the Arm FVP port, when using the normal boot flow, each secondary CPU powers itself off. The primary CPU is responsible for powering up the secondary CPUs when normal world software requires them. When booting an EL3 payload instead, they stay powered on and are put in a holding pen until their mailbox gets populated.

This function fulfills requirement 2 above.

Note that for platforms that can't release secondary CPUs out of reset, only the primary CPU will execute the cold boot code. Therefore, implementing this function is not required on such platforms.

6.4.3 Generic hold pen helpers

When `COLD_BOOT_SINGLE_CPU` is not set, TF-A provides a generic hold pen implementation that platforms can use for SMP secondary CPU bring-up. Each core gets a cache-line-aligned slot containing an entry field and two magic tags, avoiding common pitfalls with cache coherency during the boot handshake between primary and secondary CPUs.

The C API consists of `plat_hold_pen_init()`, to be called during platform setup (typically from `plat_setup_psci_ops()`), `plat_hold_pen_signal()` to be called from `pwr_domain_on()`, and the assembly macro `plat_hold_pen_wait_and_jump` to be used by waking cores, usually from `plat_secondary_cold_boot_setup()`. Detailed documentation for each is in `include/plat/common/plat_hold_pen.h` and `include/plat/common/plat_hold_pen.S`. See `plat/arm/board/corstone1000/` and `plat/qemu/` for usage examples.

Platforms using this helper must add `plat/common/plat_hold_pen.c` to `BL31_SOURCES`.

6.4.4 Function : plat_is_my_cpu_primary() [mandatory when COLD_BOOT_SINGLE_CPU == 0]

```
Argument : void
Return   : unsigned int
```

This function identifies whether the current CPU is the primary CPU or a secondary CPU. A return value of zero indicates that the CPU is not the primary CPU, while a non-zero return value indicates that the CPU is the primary CPU.

Note that for platforms that can't release secondary CPUs out of reset, only the primary CPU will execute the cold boot code. Therefore, there is no need to distinguish between primary and secondary CPUs and implementing this function is not required.

6.4.5 Function : platform_mem_init() [mandatory]

```
Argument : void
Return   : void
```

This function is called before any access to data is made by the firmware, in order to carry out any essential memory initialization.

6.4.6 Function: plat_get_rotpk_info()

```
Argument : void *, void **, unsigned int *, unsigned int *
Return   : int
```

This function is mandatory when Trusted Board Boot is enabled. It returns a pointer to the ROTPK stored in the platform (or a hash of it) and its length. The ROTPK must be encoded in DER format according to the following ASN.1 structure:

```
AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm         OBJECT IDENTIFIER,
    parameters        ANY DEFINED BY algorithm OPTIONAL
}

SubjectPublicKeyInfo  ::=  SEQUENCE  {
    algorithm         AlgorithmIdentifier,
    subjectPublicKey  BIT STRING
}
```

In case the function returns a hash of the key:

```
DigestInfo ::= SEQUENCE {
    digestAlgorithm   AlgorithmIdentifier,
    digest            OCTET STRING
}
```

The function returns 0 on success. Any other value is treated as error by the Trusted Board Boot. The function also reports extra information related to the ROTPK in the flags parameter:

```
ROTPK_IS_HASH      : Indicates that the ROTPK returned by the platform is a
                     hash.
ROTPK_NOT_DEPLOYED : This allows the platform to skip certificate ROTPK
                     verification while the platform ROTPK is not deployed.
                     When this flag is set, the function does not need to
                     return a platform ROTPK, and the authentication
                     framework uses the ROTPK in the certificate without
                     verifying it against the platform value. This flag
                     must not be used in a deployed production environment.
```

6.4.7 Function: plat_get_nv_ctr()

```
Argument : void *, unsigned int *
Return   : int
```

This function is mandatory when Trusted Board Boot is enabled. It returns the non-volatile counter value stored in the platform in the second argument. The cookie in the first argument may be used to select the counter in case the platform provides more than one (for example, on platforms that use the default TBBR CoT, the cookie will correspond to the OID values defined in TRUSTED_FW_NVCOUNTER_OID or NON_TRUSTED_FW_NVCOUNTER_OID).

The function returns 0 on success. Any other value means the counter value could not be retrieved from the platform.

6.4.8 Function: plat_set_nv_ctr()

```
Argument : void *, unsigned int
Return   : int
```

This function is mandatory when Trusted Board Boot is enabled. It sets a new counter value in the platform. The cookie in the first argument may be used to select the counter (as explained in plat_get_nv_ctr()). The second argument is the updated counter value to be written to the NV counter.

The function returns 0 on success. Any other value means the counter value could not be updated.

6.4.9 Function: plat_set_nv_ctr2()

```
Argument : void *, const auth_img_desc_t *, unsigned int
Return   : int
```

This function is optional when Trusted Board Boot is enabled. If this interface is defined, then `plat_set_nv_ctr()` need not be defined. The first argument passed is a cookie and is typically used to differentiate between a Non Trusted NV Counter and a Trusted NV Counter. The second argument is a pointer to an authentication image descriptor and may be used to decide if the counter is allowed to be updated or not. The third argument is the updated counter value to be written to the NV counter.

The function returns 0 on success. Any other value means the counter value either could not be updated or the authentication image descriptor indicates that it is not allowed to be updated.

6.5 Dynamic Root of Trust for Measurement support (in BL31)

The functions mentioned in this section are mandatory, when platform enables DRTM_SUPPORT build flag.

6.5.1 Function : plat_get_addr_mmap()

```
Argument : void
Return   : const mmap_region_t *
```

This function is used to return the address of the platform *address-map* table, which describes the regions of normal memory, memory mapped I/O and non-volatile memory.

6.5.2 Function : plat_has_non_host_platforms()

```
Argument : void
Return   : bool
```

This function returns *true* if the platform has any trusted devices capable of DMA, otherwise returns *false*.

6.5.3 Function : plat_has_unmanaged_dma_peripherals()

```
Argument : void
Return   : bool
```

This function returns *true* if platform uses peripherals whose DMA is not managed by an SMMU, otherwise returns *false*.

Note - If the platform has peripherals that are not managed by the SMMU, then the platform should investigate such peripherals to determine whether they can be trusted, and such peripherals should be moved under “Non-host platforms” if they can be trusted.

6.5.4 Function : plat_get_total_num_smmus()

```
Argument : void
Return   : unsigned int
```

This function returns the total number of SMMUs in the platform.

6.5.5 Function : plat_enumerate_smmus()

```
Argument : void
Return   : const uintptr_t *, size_t
```

This function returns an array of SMMU addresses and the actual number of SMMUs reported by the platform.

6.5.6 Function : plat_drtm_get_dma_prot_features()

```
Argument : void
Return   : const plat_drtm_dma_prot_features_t*
```

This function returns the address of plat_drtm_dma_prot_features_t structure containing the maximum number of protected regions and bitmap with the types of DMA protection supported by the platform. For more details see section 3.3 Table 6 of DRTM specification.

6.5.7 Function : plat_drtm_dma_prot_get_max_table_bytes()

```
Argument : void
Return   : uint64_t
```

This function returns the maximum size of DMA protected regions table in bytes.

6.5.8 Function : plat_drtm_get_tpm_features()

```
Argument : void
Return   : const plat_drtm_tpm_features_t*
```

This function returns the address of *plat_drtm_tpm_features_t* structure containing PCR usage schema, TPM-based hash, and firmware hash algorithm supported by the platform.

6.5.9 Function : plat_drtm_get_min_size_normal_world_dce()

```
Argument : void
Return   : uint64_t
```

This function returns the size normal-world DCE of the platform.

6.5.10 Function : plat_drtm_get_imp_def_dlme_region_size()

```
Argument : void
Return   : uint64_t
```

This function returns the size of implementation defined DLME region of the platform.

6.5.11 Function : plat_drtm_get_tcb_hash_table_size()

```
Argument : void
Return   : uint64_t
```

This function returns the size of TCB hash table of the platform.

6.5.12 Function : plat_drtm_get_acpi_tables_region_size()

```
Argument : void
Return   : uint64_t
```

This function returns the size of ACPI tables region of the platform.

6.5.13 Function : plat_drtm_get_tcb_hash_features()

```
Argument : void
Return   : uint64_t
```

This function returns the Maximum number of TCB hashes recorded by the platform. For more details see section 3.3 Table 6 of DRTM specification.

6.5.14 Function : plat_drtm_get_dlme_img_auth_features()

```
Argument : void
Return   : uint64_t
```

This function returns the DLME image authentication features. For more details see section 3.3 Table 6 of DRTM specification.

6.5.15 Function : plat_drtm_validate_ns_region()

```
Argument : uintptr_t, uintptr_t
Return   : int
```

This function validates that given region is within the Non-Secure region of DRAM. This function takes a region start address and size an input arguments, and returns 0 on success and -1 on failure.

6.5.16 Function : plat_set_drtm_error()

```
Argument : uint64_t
Return   : int
```

This function writes a 64 bit error code received as input into non-volatile storage and returns 0 on success and -1 on failure.

6.5.17 Function : plat_get_drtm_error()

```
Argument : uint64_t*
Return   : int
```

This function reads a 64 bit error code from the non-volatile storage into the received address, and returns 0 on success and -1 on failure.

6.6 Common mandatory function modifications

The following functions are mandatory functions which need to be implemented by the platform port.

6.6.1 Function : plat_my_core_pos()

```
Argument : void
Return   : unsigned int
```

This function returns the index of the calling CPU which is used as a CPU-specific linear index into blocks of memory (for example while allocating per-CPU stacks). This function will be invoked very early in the initialization sequence which mandates that this function should be implemented in assembly and should not rely on the availability of a C runtime environment. This function can clobber x0 - x8 and must preserve x9 - x29.

This function plays a crucial role in the power domain topology framework in PSCI and details of this can be found in *PSCI Power Domain Tree Structure*.

6.6.2 Function : plat_core_pos_by_mpidr()

```
Argument : u_register_t
Return   : int
```

This function validates the `MPIDR` of a CPU and converts it to an index, which can be used as a CPU-specific linear index into blocks of memory. In case the `MPIDR` is invalid, this function returns -1. This function will only be invoked by BL31 after the power domain topology is initialized and can utilize the C runtime environment. For further details about how TF-A represents the power domain topology and how this relates to the linear CPU index, please refer *PSCI Power Domain Tree Structure*.

6.6.3 Function : plat_get_mbedtls_heap() [when TRUSTED_BOARD_BOOT == 1]

```
Arguments : void **heap_addr, size_t *heap_size
Return    : int
```

This function is invoked during Mbed TLS library initialisation to get a heap, by means of a starting address and a size. This heap will then be used internally by the Mbed TLS library. Hence, each BL stage that utilises Mbed TLS must be able to provide a heap to it.

A helper function can be found in *drivers/auth/mbedtls/mbedtls_common.c* in which a heap is statically reserved during compile time inside every image (i.e. every BL stage) that utilises Mbed TLS. In this default implementation, the function simply returns the address and size of this "pre-allocated" heap. For a platform to use this default implementation, only a call to the helper from inside plat_get_mbedtls_heap() body is enough and nothing else is needed.

However, by writting their own implementation, platforms have the potential to optimise memory usage. For example, on some Arm platforms, the Mbed TLS heap is shared between BL1 and BL2 stages and, thus, the necessary space is not reserved twice.

On success the function should return 0 and a negative error code otherwise.

6.6.4 Function : plat_get_enc_key_info() [when FW_ENC_STATUS == 0 or 1]

```
Arguments : enum fw_enc_status_t fw_enc_status, uint8_t *key,
            size_t *key_len, unsigned int *flags, const uint8_t *img_id,
            size_t img_id_len
Return    : int
```

This function provides a symmetric key (either SSK or BSSK depending on fw_enc_status) which is invoked during runtime decryption of encrypted firmware images. *plat/common/plat_bl_common.c* provides a dummy weak implementation for testing purposes which must be overridden by the platform trying to implement a real world firmware encryption use-case.

It also allows the platform to pass symmetric key identifier rather than actual symmetric key which is useful in cases where the crypto backend provides secure storage for the symmetric key. So in this case `ENC_KEY_IS_IDENTIFIER` flag must be set in `flags`.

In addition to above a platform may also choose to provide an image specific symmetric key/identifier using img_id.

On success the function should return 0 and a negative error code otherwise.

Note that this API depends on `DECRYPTION_SUPPORT` build flag.

6.6.5 Function : plat_fwu_set_images_source() [when PSA_FWU_SUPPORT == 1]

```
Argument : const struct fwu_metadata *metadata
Return   : void
```

This function is mandatory when PSA_FWU_SUPPORT is enabled. It provides a means to retrieve image specification (offset in non-volatile storage and length) of active/updated images using the passed FWU metadata, and update I/O policies of active/updated images using retrieved image specification information. Further I/O layer operations such as I/O open, I/O read, etc. on these images rely on this function call.

In Arm platforms, this function is used to set an I/O policy of the FIP image, container of all active/updated secure and non-secure images.

6.6.6 Function : plat_fwu_set_metadata_image_source() [when PSA_FWU_SUPPORT == 1]

```
Argument : unsigned int image_id, uintptr_t *dev_handle,
           uintptr_t *image_spec
Return   : int
```

This function is mandatory when PSA_FWU_SUPPORT is enabled. It is responsible for setting up the platform I/O policy of the requested metadata image (either FWU_METADATA_IMAGE_ID or BKUP_FWU_METADATA_IMAGE_ID) that will be used to load this image from the platform's non-volatile storage.

FWU metadata can not be always stored as a raw image in non-volatile storage to define its image specification (offset in non-volatile storage and length) statically in I/O policy. For example, the FWU metadata image is stored as a partition inside the GUID partition table image. Its specification is defined in the partition table that needs to be parsed dynamically. This function provides a means to retrieve such dynamic information to set the I/O policy of the FWU metadata image. Further I/O layer operations such as I/O open, I/O read, etc. on FWU metadata image relies on this function call.

It returns '0' on success, otherwise a negative error value on error. Alongside, returns device handle and image specification from the I/O policy of the requested FWU metadata image.

6.6.7 Function : plat_fwu_get_boot_idx() [when PSA_FWU_SUPPORT == 1]

```
Argument : void
Return   : uint32_t
```

This function is mandatory when PSA_FWU_SUPPORT is enabled. It provides the means to retrieve the boot index value from the platform. The boot index is the bank from which the platform has booted the firmware images.

By default, the platform will read the metadata structure and try to boot from the active bank. If the platform fails to boot from the active bank due to reasons like an Authentication failure, or on crossing a set number of watchdog resets while booting from the active bank, the platform can then switch to boot from a different bank. This function then returns the bank that the platform should boot its images from.

6.7 Common optional modifications

The following are helper functions implemented by the firmware that perform common platform-specific tasks. A platform may choose to override these definitions.

6.7.1 Function : plat_set_my_stack()

```
Argument : void
Return   : void
```

This function sets the current stack pointer to the normal memory stack that has been allocated for the current CPU. For BL images that only require a stack for the primary CPU, the UP version of the function is used. The size of the stack allocated to each CPU is specified by the platform defined constant `PLATFORM_STACK_SIZE`.

Common implementations of this function for the UP and MP BL images are provided in `plat/common/aarch64/platform_up_stack.S` and `plat/common/aarch64/platform_mp_stack.S`

6.7.2 Function : plat_get_my_stack()

```
Argument : void
Return   : uintptr_t
```

This function returns the base address of the normal memory stack that has been allocated for the current CPU. For BL images that only require a stack for the primary CPU, the UP version of the function is used. The size of the stack allocated to each CPU is specified by the platform defined constant `PLATFORM_STACK_SIZE`.

Common implementations of this function for the UP and MP BL images are provided in `plat/common/aarch64/platform_up_stack.S` and `plat/common/aarch64/platform_mp_stack.S`

6.7.3 Function : plat_report_exception()

```
Argument : unsigned int
Return   : void
```

A platform may need to report various information about its status when an exception is taken, for example the current exception level, the CPU security state (secure/non-secure), the exception type, and so on. This function is called in the following circumstances:

- In BL1, whenever an exception is taken.
- In BL2, whenever an exception is taken.

The default implementation doesn't do anything, to avoid making assumptions about the way the platform displays its status information.

For AArch64, this function receives the exception type as its argument. Possible values for exceptions types are listed in the `include/common/bl_common.h` header file. Note that these constants are not related to any architectural exception code; they are just a TF-A convention.

For AArch32, this function receives the exception mode as its argument. Possible values for exception modes are listed in the `include/lib/aarch32/arch.h` header file.

6.7.4 Function : plat_reset_handler()

```
Argument : void
Return   : void
```

A platform may need to do additional initialization after reset. This function allows the platform to do the platform specific initializations. Platform specific errata workarounds could also be implemented here. The API should preserve the values of callee saved registers x19 to x29.

The default implementation doesn't do anything. If a platform needs to override the default implementation, refer to the *Firmware Design* for general guidelines.

6.7.5 Function : plat_disable_acp()

```
Argument : void
Return   : void
```

This API allows a platform to disable the Accelerator Coherency Port (if present) during a cluster power down sequence. The default weak implementation doesn't do anything. Since this API is called during the power down sequence, it has restrictions for stack usage and it can use the registers x0 - x17 as scratch registers. It should preserve the value in x18 register as it is used by the caller to store the return address.

6.7.6 Function : plat_error_handler()

```
Argument : int
Return   : void
```

This API is called when the generic code encounters an error situation from which it cannot continue. It allows the platform to perform error reporting or recovery actions (for example, reset the system). This function must not return.

The parameter indicates the type of error using standard codes from `errno.h`. Possible errors reported by the generic code are:

- `-EAUTH`: a certificate or image could not be authenticated (when Trusted Board Boot is enabled)
- `-ENOENT`: the requested image or certificate could not be found or an IO error was detected
- `-ENOMEM`: resources exhausted. TF-A does not use dynamic memory, so this error is usually an indication of an incorrect array size

The default implementation simply spins.

6.7.7 Function : plat_panic_handler()

```
Argument : void
Return   : void
```

This API is called when the generic code encounters an unexpected error situation from which it cannot recover. This function must not return, and must be implemented in assembly because it may be called before the C environment is initialized.

Note: The address from where it was called is stored in x30 (Link Register). The default implementation simply spins.

6.7.8 Function : plat_get_bl_image_load_info()

```
Argument : void
Return   : bl_load_info_t *
```

This function returns pointer to the list of images that the platform has populated to load. This function is invoked in BL2 to load the BL3xx images.

6.7.9 Function : plat_get_next_bl_params()

```
Argument : void
Return   : bl_params_t *
```

This function returns a pointer to the shared memory that the platform has kept aside to pass TF-A related information that next BL image needs. This function is invoked in BL2 to pass this information to the next BL image.

6.7.10 Function : plat_get_stack_protector_canary()

```
Argument : void
Return   : u_register_t
```

This function returns a random value that is used to initialize the canary used when the stack protector is enabled with ENABLE_STACK_PROTECTOR. A predictable value will weaken the protection as the attacker could easily write the right value as part of the attack most of the time. Therefore, it should return a true random number.

Warning: For the protection to be effective, the global data need to be placed at a lower address than the stack bases. Failure to do so would allow an attacker to overwrite the canary as part of the stack buffer overflow attack.

6.7.11 Function : plat_flush_next_bl_params()

```
Argument : void
Return   : void
```

This function flushes to main memory all the image params that are passed to next image. This function is invoked in BL2 to flush this information to the next BL image.

6.7.12 Function : plat_log_get_prefix()

```
Argument : unsigned int
Return   : const char *
```

This function defines the prefix string corresponding to the *log_level* to be prepended to all the log output from TF-A. The *log_level* (argument) will correspond to one of the standard log levels defined in debug.h. The platform can override the common implementation to define a different prefix string for the log output. The implementation should be robust to future changes that increase the number of log levels.

6.7.13 Function : plat_get_soc_version()

```
Argument : void
Return   : int32_t
```

This function returns soc version which mainly consist of below fields

```
soc_version[30:24] = JEP-106 continuation code for the SiP
soc_version[23:16] = JEP-106 identification code with parity bit for the SiP
soc_version[15:0]  = Implementation defined SoC ID
```

6.7.14 Function : plat_get_soc_revision()

```
Argument : void
Return   : int32_t
```

This function returns soc revision in below format

```
soc_revision[0:30] = SOC revision of specific SOC
```

6.7.15 Function : plat_get_soc_name()

```
Argument : char **
Return   : int32_t
```

The plat_get_soc_name() function allows a platform to expose the SoC name to the firmware. It takes a pointer to a character pointer as an argument, which must be set to point to a static, null-terminated SoC name string. The string must be encoded in UTF-8 and should use only printable ASCII characters for compatibility. It must not exceed 136 bytes, including the null terminator. On success, the function returns SMC_ARCH_CALL_SUCCESS. If the platform does not support SoC name retrieval, it returns SMC_ARCH_CALL_NOT_SUPPORTED. This API allows platforms to support SoC name queries via SMCCC_ARCH_SOC_ID.

6.7.16 Function : plat_is_smccc_feature_available()

```
Argument : u_register_t
Return   : int32_t
```

This function returns SMC_ARCH_CALL_SUCCESS if the platform supports the SMCCC function specified in the argument; otherwise returns SMC_ARCH_CALL_NOT_SUPPORTED.

6.7.17 Function : plat_can_cmo()

```
Argument : void
Return   : uint64_t
```

When CONDITIONAL_CMO flag is enabled:

- This function indicates whether cache management operations should be performed. It returns 0 if CMOs should be skipped and non-zero otherwise.
- The function must not clobber x1, x2 and x3. It's also not safe to rely on stack. Otherwise obey AAPCS.

6.7.18 Struct: plat_try_images_ops [optional]

This optional structure holds platform hooks for alternative images load. It has to be defined in platform code and registered by calling plat_setup_try_img_ops() function, passing it the address of the plat_try_images_ops struct.

Function : plat_setup_try_img_ops [optional]

```
Argument : const struct plat_try_images_ops *
Return   : void
```

This optional function is called to register platform try images ops, given as argument.

Function : plat_try_images_ops.next_instance [optional]

```
Argument : unsigned int image_id
Return   : int
```

This optional function tries to load images from alternative places. In case PSA FWU is not used, it can be any instance or media. If PSA FWU is used, it is mandatory that the backup image is on the same media. This is required for MTD devices like NAND. The argument is the ID of the image for which we are looking for an alternative place. It returns 0 in case of success and a negative errno value otherwise.

Function : plat_setup_log_gpt_corrupted [optional]

```
Argument : const struct plat_log_gpt_corrupted *
Return   : void
```

This optional function is called to register platform log GPT corrupted functions, given as argument.

Function : plat_setup_log_gpt_corrupted.plat_set_gpt_corruption [optional]

```
Argument : uintptr_t gpt_corrupted_info_ptr, uint8_t flags
Return   : void
```

This optional function will log error information if the GPT is corrupted, by setting the address passed by gpt_corrupted_info_ptr to flags value, currently bit[0] is used for logging primary GPT corruption, bit[7:1] are reserved. The data type passed by reference is uint8_t.

Function : plat_setup_log_gpt_corrupted.plat_log_gpt_corruption [optional]

```
Argument : uintptr_t log_address, uint8_t gpt_corrupted_info
Return   : void
```

This optional function will log if the primary GPT is corrupted, by writing the value of gpt_corrupted_info to the address passed by log_address.

6.8 Modifications specific to a Boot Loader stage

6.9 Boot Loader Stage 1 (BL1)

BL1 implements the reset vector where execution starts from after a cold or warm boot. For each CPU, BL1 is responsible for the following tasks:

1. Handling the reset as described in section 2.2
2. In the case of a cold boot and the CPU being the primary CPU, ensuring that only this CPU executes the remaining BL1 code, including loading and passing control to the BL2 stage.
3. Identifying and starting the Firmware Update process (if required).
4. Loading the BL2 image from non-volatile storage into secure memory at the address specified by the platform defined constant `BL2_BASE`.
5. Populating a `meminfo` structure with the following information in memory, accessible by BL2 immediately upon entry.

   ```
   meminfo.total_base = Base address of secure RAM visible to BL2
   meminfo.total_size = Size of secure RAM visible to BL2
   ```

   By default, BL1 places this `meminfo` structure at the end of secure memory visible to BL2.

   It is possible for the platform to decide where it wants to place the `meminfo` structure for BL2 or restrict the amount of memory visible to BL2 by overriding the weak default implementation of `bl1_plat_handle_post_image_load` API.

The following functions need to be implemented by the platform port to enable BL1 to perform the above tasks.

6.9.1 Function : bl1_early_platform_setup() [mandatory]

```
Argument : void
Return   : void
```

This function executes with the MMU and data caches disabled. It is only called by the primary CPU.

On Arm standard platforms, this function:

- Enables a secure instance of SP805 to act as the Trusted Watchdog.
- Initializes a UART (PL011 console), which enables access to the `printf` family of functions in BL1.
- Enables issuing of snoop and DVM (Distributed Virtual Memory) requests to the CCI slave interface corresponding to the cluster that includes the primary CPU.

6.9.2 Function : bl1_plat_arch_setup() [mandatory]

```
Argument : void
Return   : void
```

This function performs any platform-specific and architectural setup that the platform requires. Platform-specific setup might include configuration of memory controllers and the interconnect.

In Arm standard platforms, this function enables the MMU.

This function helps fulfill requirement 2 above.

6.9.3 Function : bl1_platform_setup() [mandatory]

```
Argument : void
Return   : void
```

This function executes with the MMU and data caches enabled. It is responsible for performing any remaining platform-specific setup that can occur after the MMU and data cache have been enabled.

In Arm standard platforms, this function initializes the storage abstraction layer used to load the next bootloader image.

This function helps fulfill requirement 4 above.

6.9.4 Function : bl1_plat_sec_mem_layout() [mandatory]

```
Argument : void
Return   : meminfo *
```

This function should only be called on the cold boot path. It executes with the MMU and data caches enabled. The pointer returned by this function must point to a `meminfo` structure containing the extents and availability of secure RAM for the BL1 stage.

```
meminfo.total_base = Base address of secure RAM visible to BL1
meminfo.total_size = Size of secure RAM visible to BL1
```

This information is used by BL1 to load the BL2 image in secure RAM. BL1 also populates a similar structure to tell BL2 the extents of memory available for its own use.

This function helps fulfill requirements 4 and 5 above.

6.9.5 Function : bl1_plat_prepare_exit() [optional]

```
Argument : entry_point_info_t *
Return   : void
```

This function is called prior to exiting BL1 in response to the `BL1_SMC_RUN_IMAGE` SMC request raised by BL2. It should be used to perform platform specific clean up or bookkeeping operations before transferring control to the next image. It receives the address of the `entry_point_info_t` structure passed from BL2. This function runs with MMU disabled.

6.9.6 Function : bl1_plat_set_ep_info() [optional]

```
Argument : unsigned int image_id, entry_point_info_t *ep_info
Return   : void
```

This function allows platforms to override `ep_info` for the given `image_id`.

The default implementation just returns.

6.9.7 Function : bl1_plat_get_next_image_id() [optional]

```
Argument : void
Return   : unsigned int
```

This and the following function must be overridden to enable the FWU feature.

BL1 calls this function after platform setup to identify the next image to be loaded and executed. If the platform returns `BL2_IMAGE_ID` then BL1 proceeds with the normal boot sequence, which loads and executes BL2. If the platform returns a different image id, BL1 assumes that Firmware Update is required.

The default implementation always returns `BL2_IMAGE_ID`. The Arm development platforms override this function to detect if firmware update is required, and if so, return the first image in the firmware update process.

6.9.8 Function : bl1_plat_get_image_desc() [optional]

```
Argument : unsigned int image_id
Return   : image_desc_t *
```

BL1 calls this function to get the image descriptor information `image_desc_t` for the provided `image_id` from the platform.

The default implementation always returns a common BL2 image descriptor. Arm standard platforms return an image descriptor corresponding to BL2 or one of the firmware update images defined in the Trusted Board Boot Requirements specification.

6.9.9 Function : bl1_plat_handle_pre_image_load() [optional]

```
Argument : unsigned int image_id
Return   : int
```

This function can be used by the platforms to update/use image information corresponding to `image_id`. This function is invoked in BL1, both in cold boot and FWU code path, before loading the image.

6.9.10 Function : bl1_plat_calc_bl2_layout() [optional]

```
Argument : const meminfo_t *bl1_mem_layout, meminfo_t *bl2_mem_layout
Return   : void
```

This utility function calculates the memory layout of BL2, representing it in a *meminfo_t* structure. The default implementation derives this layout from the positioning of BL1's RW data at the top of the memory layout.

6.9.11 Function : bl1_plat_handle_post_image_load() [optional]

```
Argument : unsigned int image_id
Return   : int
```

This function can be used by the platforms to update/use image information corresponding to `image_id`. This function is invoked in BL1, both in cold boot and FWU code path, after loading and authenticating the image.

The default weak implementation of this function calculates the amount of Trusted SRAM that can be used by BL2 and allocates a `meminfo_t` structure at the beginning of this free memory and populates it. The address of `meminfo_t` structure is updated in `arg1` of the entrypoint information to BL2.

6.9.12 Function : bl1_plat_fwu_done() [optional]

```
Argument : unsigned int image_id, uintptr_t image_src,
           unsigned int image_size
Return   : void
```

BL1 calls this function when the FWU process is complete. It must not return. The platform may override this function to take platform specific action, for example to initiate the normal boot flow.

The default implementation spins forever.

6.9.13 Function : bl1_plat_mem_check() [mandatory]

```
Argument : uintptr_t mem_base, unsigned int mem_size,
           unsigned int flags
Return   : int
```

BL1 calls this function while handling FWU related SMCs, more specifically when copying or authenticating an image. Its responsibility is to ensure that the region of memory identified by `mem_base` and `mem_size` is mapped in BL1, and that this memory corresponds to either a secure or non-secure memory region as indicated by the security state of the `flags` argument.

This function can safely assume that the value resulting from the addition of `mem_base` and `mem_size` fits into a `uintptr_t` type variable and does not overflow.

This function must return 0 on success, a non-null error code otherwise.

The default implementation of this function asserts therefore platforms must override it when using the FWU feature.

6.9.14 Function : bl1_plat_is_shared_nv_ctr() [optional]

```
Argument : void
Return   : bool
```

This function inquiry the platform if the non-volatile counter is shared across all secure images (BL2, BL31, BL32, etc.). It is used only in BL1 and when *PSA_FWU_SUPPORT* is enabled.

6.10 Boot Loader Stage 2 (BL2)

The BL2 stage is executed only by the primary CPU, which is determined in BL1 using the `platform_is_primary_cpu()` function. BL1 passed control to BL2 at `BL2_BASE`. BL2 executes in Secure EL1 and and invokes `plat_get_bl_image_load_info()` to retrieve the list of images to load from non-volatile storage to secure/non-secure RAM. After all the images are loaded then BL2 invokes `plat_get_next_bl_params()` to get the list of executable images to be passed to the next BL image.

The following functions must be implemented by the platform port to enable BL2 to perform the above tasks.

6.10.1 Function : bl2_early_platform_setup2() [mandatory]

```
Argument : u_register_t, u_register_t, u_register_t, u_register_t
Return   : void
```

This function executes with the MMU and data caches disabled. It is only called by the primary CPU. The 4 arguments are passed by BL1 to BL2 and these arguments are platform specific.

On Arm standard platforms when RESET_TO_BL2 is not set, the arguments received are :

arg0 - Points to load address of FW_CONFIG

arg1 - `meminfo` structure populated by BL1. The platform copies the contents of `meminfo` as it may be subsequently overwritten by BL2.

On Arm standard platforms, this function also:

- Initializes a UART (PL011 console), which enables access to the `printf` family of functions in BL2.
- Initializes the storage abstraction layer used to load further bootloader images. It is necessary to do this early on platforms with a SCP_BL2 image, since the later `bl2_platform_setup` must be done after SCP_BL2 is loaded.

6.10.2 Function : bl2_plat_arch_setup() [mandatory]

```
Argument : void
Return   : void
```

This function executes with the MMU and data caches disabled. It is only called by the primary CPU.

The purpose of this function is to perform any architectural initialization that varies across platforms.

On Arm standard platforms, this function enables the MMU.

6.10.3 Function : bl2_platform_setup() [mandatory]

```
Argument : void
Return   : void
```

This function may execute with the MMU and data caches enabled if the platform port does the necessary initialization in `bl2_plat_arch_setup()`. It is only called by the primary CPU.

The purpose of this function is to perform any platform initialization specific to BL2.

In Arm standard platforms, this function performs security setup, including configuration of the TrustZone controller to allow non-secure masters access to most of DRAM. Part of DRAM is reserved for secure world use.

6.10.4 Function : bl2_plat_handle_pre_image_load() [optional]

```
Argument : unsigned int
Return   : int
```

This function can be used by the platforms to update/use image information for given `image_id`. This function is currently invoked in BL2 before loading each image.

6.10.5 Function : bl2_plat_handle_post_image_load() [optional]

```
Argument : unsigned int
Return   : int
```

This function can be used by the platforms to update/use image information for given `image_id`. This function is currently invoked in BL2 after loading each image.

6.10.6 Function : bl2_plat_preload_setup [optional]

```
Argument : void
Return   : void
```

This optional function performs any BL2 platform initialization required before image loading, that is not done later in bl2_platform_setup().

6.11 Boot Loader Stage 2 (BL2) at EL3

When the platform has a non-TF-A Boot ROM it is desirable to jump directly to BL2 instead of TF-A BL1. In this case BL2 is expected to execute at EL3 instead of executing at EL1. Also on RME enabled systems BL2 runs at EL3. Refer to the *Firmware Design* document for more information.

The existing mandatory functions of BL2 must handle BL2 running at EL3.

6.12 FWU Boot Loader Stage 2 (BL2U)

The AP Firmware Updater Configuration, BL2U, is an optional part of the FWU process and is executed only by the primary CPU. BL1 passes control to BL2U at `BL2U_BASE`. BL2U executes in Secure-EL1 and is responsible for:

1. (Optional) Transferring the optional SCP_BL2U binary image from AP secure memory to SCP RAM. BL2U uses the SCP_BL2U `image_info` passed by BL1. `SCP_BL2U_BASE` defines the address in AP secure memory where SCP_BL2U should be copied from. Subsequent handling of the SCP_BL2U image is implemented by the platform specific `bl2u_plat_handle_scp_bl2u()` function. If `SCP_BL2U_BASE` is not defined then this step is not performed.
2. Any platform specific setup required to perform the FWU process. For example, Arm standard platforms initialize the TZC controller so that the normal world can access DDR memory.

The following functions must be implemented by the platform port to enable BL2U to perform the tasks mentioned above.

6.12.1 Function : bl2u_early_platform_setup() [mandatory]

```
Argument : meminfo *mem_info, void *plat_info
Return   : void
```

This function executes with the MMU and data caches disabled. It is only called by the primary CPU. The arguments to this function is the address of the `meminfo` structure and platform specific info provided by BL1.

The platform may copy the contents of the `mem_info` and `plat_info` into private storage as the original memory may be subsequently overwritten by BL2U.

On Arm CSS platforms `plat_info` is interpreted as an `image_info_t` structure, to extract SCP_BL2U image information, which is then copied into a private variable.

6.12.2 Function : bl2u_plat_arch_setup() [mandatory]

```
Argument : void
Return   : void
```

This function executes with the MMU and data caches disabled. It is only called by the primary CPU.

The purpose of this function is to perform any architectural initialization that varies across platforms, for example enabling the MMU (since the memory map differs across platforms).

6.12.3 Function : bl2u_platform_setup() [mandatory]

```
Argument : void
Return   : void
```

This function may execute with the MMU and data caches enabled if the platform port does the necessary initialization in `bl2u_plat_arch_setup()`. It is only called by the primary CPU.

The purpose of this function is to perform any platform initialization specific to BL2U.

In Arm standard platforms, this function performs security setup, including configuration of the TrustZone controller to allow non-secure masters access to most of DRAM. Part of DRAM is reserved for secure world use.

6.12.4 Function : bl2u_plat_handle_scp_bl2u() [optional]

```
Argument : void
Return   : int
```

This function is used to perform any platform-specific actions required to handle the SCP firmware. Typically it transfers the image into SCP memory using a platform-specific protocol and waits until SCP executes it and signals to the Application Processor (AP) for BL2U execution to continue.

This function returns 0 on success, a negative error code otherwise. This function is included if SCP_BL2U_BASE is defined.

6.13 Boot Loader Stage 3-1 (BL31)

During cold boot, the BL31 stage is executed only by the primary CPU. This is determined in BL1 using the `platform_is_primary_cpu()` function. BL1 passes control to BL31 at `BL31_BASE`. During warm boot, BL31 is executed by all CPUs. BL31 executes at EL3 and is responsible for:

1. Re-initializing all architectural and platform state. Although BL1 performs some of this initialization, BL31 remains resident in EL3 and must ensure that EL3 architectural and platform state is completely initialized. It should make no assumptions about the system state when it receives control.
2. Passing control to a normal world BL image, pre-loaded at a platform- specific address by BL2. On ARM platforms, BL31 uses the `bl_params` list populated by BL2 in memory to do this.
3. Providing runtime firmware services. Currently, BL31 only implements a subset of the Power State Coordination Interface (PSCI) API as a runtime service. See *Power State Coordination Interface (PSCI)* below for details of porting the PSCI implementation.
4. Optionally passing control to the BL32 image, pre-loaded at a platform- specific address by BL2. BL31 exports a set of APIs that allow runtime services to specify the security state in which the next image should be executed and run the corresponding image. On ARM platforms, BL31 uses the `bl_params` list populated by BL2 in memory to do this.

If BL31 is a reset vector, It also needs to handle the reset as specified in section 2.2 before the tasks described above.

The following functions must be implemented by the platform port to enable BL31 to perform the above tasks.

6.13.1 Function : bl31_early_platform_setup2() [mandatory]

```
Argument : u_register_t, u_register_t, u_register_t, u_register_t
Return   : void
```

This function executes with the MMU and data caches disabled. It is only called by the primary CPU. BL2 can pass 4 arguments to BL31 and these arguments are platform specific.

In Arm standard platforms, the arguments received are :

arg0 - The pointer to the head of *bl_params_t* list which is list of executable images following BL31,

arg1 - Points to load address of SOC_FW_CONFIG if present
except in case of Arm FVP and Juno platform.

In case of Arm FVP and Juno platform, points to load address of FW_CONFIG.

arg2 - Points to load address of HW_CONFIG if present

arg3 - A special value to verify platform parameters from BL2 to BL31. Not used in release builds.

The function runs through the *bl_param_t* list and extracts the entry point information for BL32 and BL33. It also performs the following:

- Initialize a UART (PL011 console), which enables access to the `printf` family of functions in BL31.
- Enable issuing of snoop and DVM (Distributed Virtual Memory) requests to the CCI slave interface corresponding to the cluster that includes the primary CPU.

6.13.2 Function : bl31_plat_arch_setup() [mandatory]

```
Argument : void
Return   : void
```

This function executes with the MMU and data caches disabled. It is only called by the primary CPU.

The purpose of this function is to perform any architectural initialization that varies across platforms.

On Arm standard platforms, this function enables the MMU.

6.13.3 Function : bl31_platform_setup() [mandatory]

```
Argument : void
Return   : void
```

This function may execute with the MMU and data caches enabled if the platform port does the necessary initialization in `bl31_plat_arch_setup()`. It is only called by the primary CPU.

The purpose of this function is to complete platform initialization so that both BL31 runtime services and normal world software can function correctly.

On Arm standard platforms, this function does the following:

- Initialize the generic interrupt controller.

  Depending on the GIC driver selected by the platform, the appropriate GICv2 or GICv3 initialization will be done, which mainly consists of:

  - Enable secure interrupts in the GIC CPU interface.
  - Disable the legacy interrupt bypass mechanism.
  - Configure the priority mask register to allow interrupts of all priorities to be signaled to the CPU interface.
  - Mark SGIs 8-15 and the other secure interrupts on the platform as secure.
  - Target all secure SPIs to CPU0.
  - Enable these secure interrupts in the GIC distributor.
  - Configure all other interrupts as non-secure.
  - Enable signaling of secure interrupts in the GIC distributor.
- Enable system-level implementation of the generic timer counter through the memory mapped interface.

- Grant access to the system counter timer module
- Initialize the power controller device.

  In particular, initialise the locks that prevent concurrent accesses to the power controller device.

6.13.4 Function : bl31_plat_runtime_setup() [optional]

```
Argument : void
Return   : void
```

The purpose of this function is to allow the platform to perform any BL31 runtime setup just prior to BL31 exit during cold boot. The default weak implementation of this function is empty. Any platform that needs to perform additional runtime setup, before BL31 exits, will need to override this function.

6.13.5 Function : bl31_plat_get_next_image_ep_info() [mandatory]

```
Argument : uint32_t
Return   : entry_point_info *
```

This function may execute with the MMU and data caches enabled if the platform port does the necessary initializations in `bl31_plat_arch_setup()`.

This function is called by `bl31_main()` to retrieve information provided by BL2 for the next image in the security state specified by the argument. BL31 uses this information to pass control to that image in the specified security state. This function must return a pointer to the `entry_point_info` structure (that was copied during `bl31_early_platform_setup()`) if the image exists. It should return NULL otherwise.

6.13.6 Function : plat_rmmd_get_cca_attest_token() [mandatory when ENABLE_RMM == 1]

```
Argument : uintptr_t, size_t *, uintptr_t, size_t, size_t *
Return   : int
```

This function returns the Platform attestation token. If the full token does not fit in the buffer, the function will return a hunk of the token and indicate how many bytes were copied and how many are pending. Multiple calls to this function may be needed to retrieve the entire token.

The parameters of the function are:

arg0 - A pointer to the buffer where the Platform token should be copied by
this function. If the platform token does not completely fit in the buffer, the function may return a piece of the token only.

arg1 - Contains the size (in bytes) of the buffer passed in arg0. In
addition, this parameter is used by the function to return the size of the platform token length hunk copied to the buffer.

arg2 - A pointer to the buffer where the challenge object is stored.

arg3 - The length of the challenge object in bytes. Possible values are 32,
48 and 64. This argument must be zero for subsequent calls to retrieve the remaining hunks of the token.

arg4 - Returns the remaining length of the token (in bytes) that is yet to
be returned in further calls.

The function returns 0 on success, -EINVAL on failure and -EAGAIN if the resource associated with the platform token retrieval is busy.

6.13.7 Function : plat_rmmd_get_cca_realm_attest_key() [mandatory when ENABLE_RMM == 1]

```
Argument : uintptr_t, size_t *, unsigned int
Return   : int
```

This function returns the delegated realm attestation key which will be used to sign Realm attestation token. The API currently only supports P-384 ECC curve key.

The parameters of the function are:

arg0 - A pointer to the buffer where the attestation key should be copied
by this function. The buffer must be big enough to hold the attestation key.

arg1 - Contains the size (in bytes) of the buffer passed in arg0. The
function returns the attestation key length in this parameter.

arg2 - The type of the elliptic curve to which the requested attestation key
belongs.

The function returns 0 on success, -EINVAL on failure.

6.13.8 Function : plat_rmmd_get_el3_rmm_shared_mem() [when ENABLE_RMM == 1]

```
Argument : uintptr_t *
Return   : size_t
```

This function returns the size of the shared area between EL3 and RMM (or 0 on failure). A pointer to the shared area (or a NULL pointer on failure) is stored in the pointer passed as argument.

6.13.9 Function : plat_rmmd_load_manifest() [when ENABLE_RMM == 1]

```
Arguments : rmm_manifest_t *manifest
Return    : int
```

When ENABLE_RMM is enabled, this function populates a boot manifest for the RMM image and stores it in the area specified by manifest.

When ENABLE_RMM is disabled, this function is not used.

6.13.10 Function : plat_rmmd_mecid_key_update() [when ENABLE_RMM == 1]

```
Argument : uint16_t, unsigned int
Return   : int
```

This function is invoked by BL31's RMMD when there is a request from the RMM monitor to update the tweak for the encryption key associated to a MECID.

The first parameter (`uint16_t mecid`) contains the MECID for which the encryption key is to be updated. The second argument specifies the reason for key update. Possible values are: 0 - Realm creation, 1 - Realm destruction.

Return value is 0 upon success and -EFAULT otherwise.

This function needs to be implemented by a platform if it enables RME.

6.13.11 Function : plat_rmmd_reserve_memory() [when ENABLE_RMM == 1]

```
Arguments : size_t size, unsigned long alignment
Return    : uintptr_t
```

Reserve memory to be used by the RMM. This could be memory simply taken from a pool of reserved memory, for instance from a carveout dedicated to RMM.

Return value is the physical address of a memory region of at least `size` bytes, which needs to be aligned to `alignment` bytes.

This function needs to be implemented if a platform enables RME and the RMM requires the memory reservation feature.

6.13.12 Function : plat_rmmd_el3_token_sign_push_req() [mandatory when RMMD_ENABLE_EL3_TOKEN_SIGN == 1]

```
Arguments : const struct el3_token_sign_request *req
Return    : int
```

Queue realm attestation token signing request from the RMM in EL3. The interface between the RMM and EL3 is modeled as a queue but the underlying implementation may be different, so long as the semantics of queuing and the error codes are used as defined below.

See *EL3 Token Sign Request structure* for definition of the request structure.

Optional interface from the RMM-EL3 interface v0.4 onwards.

The parameters of the functions are:
arg0: Pointer to the token sign request to be pushed to EL3. The structure must be located in the RMM-EL3 shared memory buffer and must be locked before use.

Return codes:

- E_RMM_OK On Success.
- E_RMM_INVAL If the arguments are invalid.
- E_RMM_AGAIN Indicates that the request was not queued since the queue in EL3 is full. This may also be returned for any reason or situation in the system, that prevents accepting the request from the RMM.
- E_RMM_UNK If the SMC is not implemented or if interface version is < 0.4.

6.13.13 Function : plat_rmmd_el3_token_sign_pull_resp() [mandatory when RMMD_ENABLE_EL3_TOKEN_SIGN == 1]

```
Arguments : struct el3_token_sign_response *resp
Return    : int
```

Populate the attestation signing response in the `resp` parameter. The interface between the RMM and EL3 is modeled as a queue for responses but the underlying implementation may be different, so long as the semantics of queuing and the error codes are used as defined below.

See *EL3 Token Sign Response structure* for definition of the response structure.

Optional interface from the RMM-EL3 interface v0.4 onwards.

The parameters of the functions are:
resp: Pointer to the token sign response to get from EL3. The structure must be located in the RMM-EL3 shared memory buffer and must be locked before use.

Return:

- E_RMM_OK On Success.
- E_RMM_INVAL If the arguments are invalid.
- E_RMM_AGAIN Indicates that a response is not ready yet.
- E_RMM_UNK If the SMC is not implemented or if interface version is < 0.4.

6.13.14 Function : plat_rmmd_el3_token_sign_get_rak_pub() [mandatory when RMMD_ENABLE_EL3_TOKEN_SIGN == 1]

```
Argument : uintptr_t, size_t *, unsigned int
Return   : int
```

This function returns the public portion of the realm attestation key which will be used to sign Realm attestation token. Typically, with delegated attestation, the private key is returned, however, there may be platforms where the private key bits are better protected in a platform specific manner such that the private key is not exposed. In such cases, the RMM will only cache the public key and forward any requests such as signing, that uses the private key to EL3. The API currently only supports P-384 ECC curve key.

This is an optional interface from the RMM-EL3 interface v0.4 onwards.

The parameters of the function are:

> arg0 - A pointer to the buffer where the public key should be copied by this function. The buffer must be big enough to hold the attestation key.
>
> arg1 - Contains the size (in bytes) of the buffer passed in arg0. The function returns the attestation key length in this parameter.
>
> arg2 - The type of the elliptic curve to which the requested attestation key belongs.

The function returns E_RMM_OK on success, RMM_E_INVAL if arguments are invalid and E_RMM_UNK if the SMC is not implemented or if interface version is < 0.4.

6.13.15 Function : plat_rmmd_el3_ide_key_program() [mandatory when RMMD_ENABLE_IDE_KEY_PROG == 1]

```
Argument : uint64_t, uint64_t, uint64_t, struct rp_ide_key_info_t *, uint64_t,
↪ uint64_t
Return   : int
```

This function sets the key/IV info for an IDE stream at the Root port. The key is 256 bits and IV is 96 bits. The caller calls this SMC to program this key to the Rx and Tx ports and for each substream corresponding to a single keyset. The platform should validate the arguments *Ecam address* and *Rootport ID* before acting on it. The arguments *request ID* and *cookie* are to be ignored for blocking mode and are pass-through to the response for non-blocking mode.

The platform needs to ensure proper exclusives are in place when accessed from multiple CPUs. Depending on the expected latency for IDE-KM interface, the platform should choose blocking or non-blocking semantics. More details about IDE Setup flow can be found in this RFC.

The parameters of the function are:

> arg0 - The ecam address, to access and configure PCI devices in a system.
>
> arg1 - The rootport ID used to identify the PCIe rootport of a connected device.
>
> arg2 - The IDE stream info associated with a physical device, this parameter packs the the keyset, direction, substream and stream ID info.

arg3 - Structure with key and IV info.

arg4 - The request ID, is used in non-blocking mode only and can be ignored in blocking mode.

arg5 - The cookie variable, is used in non-blocking mode only and can be ignored in blocking mode.

The function returns E_RMM_OK on success, E_RMM_INVAL if arguments are invalid, E_RMM_FAULT if the key programming is unsuccesful, E_RMM_UNK for an unknown error, E_RMM_AGAIN returned only for non-blocking mode if the IDE-KM interface is busy or the request queue is full. E_RMM_INPROGRESS returned if the request is queued successfully and used only in non-blocking mode.

6.13.16 Function : plat_rmmd_el3_ide_key_set_go() [mandatory when RMMD_ENABLE_IDE_KEY_PROG == 1]

```
Argument : uint64_t, uint64_t, uint64_t, uint64_t, uint64_t
Return   : int
```

This function activates the IDE stream at the Root Port once all the keys have been programmed. The platform should validate the arguments *Ecam address* and *Rootport ID* before acting on it. The arguments *request ID* and *cookie* are to be ignored for blocking mode and are pass-through to the response for non-blocking mode.

The platform needs to ensure proper exclusives are in place when accessed from multiple CPUs. Depending on the expected latency for IDE-KM interface, the platform should choose blocking or non-blocking semantics. More details about IDE Setup flow can be found in this RFC.

The parameters of the function are:

arg0 - The ecam address, to access and configure PCI devices in a system.

arg1 - The rootport ID used to identify the PCIe rootport of a connected device.

arg2 - The IDE stream info associated with a physical device, this parameter packs the the keyset, direction, substream and stream ID info.

arg3 - The request ID, is used in non-blocking mode only and can be ignored in blocking mode.

arg4 - The cookie variable, is used in non-blocking mode only and can be ignored in blocking mode.

The function returns E_RMM_OK on success, E_RMM_INVAL if arguments are invalid, E_RMM_FAULT if the key programming is unsuccesful, E_RMM_UNK for an unknown error, E_RMM_AGAIN returned only for non-blocking mode if the IDE-KM interface is busy or the request queue is full. E_RMM_INPROGRESS returned if the request is queued successfully and used only in non-blocking mode.

6.13.17 Function : plat_rmmd_el3_ide_key_set_stop() [mandatory when RMMD_ENABLE_IDE_KEY_PROG == 1]

```
Argument : uint64_t, uint64_t, uint64_t, uint64_t, uint64_t
Return   : int
```

This function stops the IDE stream and is used to tear down the IDE stream at Root Port. The platform should validate the arguments *Ecam address* and *Rootport ID* before acting on it. The arguments *request ID* and *cookie* are to be ignored for blocking mode and are pass-through to the response for non-blocking mode.

The platform needs to ensure proper exclusives are in place when accessed from multiple CPUs. Depending on the expected latency for IDE-KM interface, the platform should choose blocking or non-blocking semantics. More details about IDE Setup flow can be found in this RFC.

The parameters of the function are:

> arg0 - The ecam address, to access and configure PCI devices in a system.
>
> arg1 - The rootport ID used to identify the PCIe rootport of a connected device.
>
> arg2 - The IDE stream info associated with a physical device, this parameter packs the the keyset, direction, substream and stream ID info.
>
> arg3 - The request ID, is used in non-blocking mode only and can be ignored in blocking mode.
>
> arg4 - The cookie variable, is used in non-blocking mode only and can be ignored in blocking mode.

The function returns E_RMM_OK on success, E_RMM_INVAL if arguments are invalid, E_RMM_FAULT if the key programming is unsuccesful, E_RMM_UNK for an unknown error, E_RMM_AGAIN returned only for non-blocking mode if the IDE-KM interface is busy or the request queue is full. E_RMM_INPROGRESS returned if the request is queued successfully and used only in non-blocking mode.

6.13.18 Function : plat_rmmd_el3_ide_km_pull_response() [mandatory when RMMD_ENABLE_IDE_KEY_PROG == 1]

```
Argument : uint64_t, uint64_t, uint64_t *, uint64_t *, uint64_t *
Return   : int
```

This function retrieves a reponse for any of the prior non-blocking IDE-KM requests. The caller has to identify the request and populate the accurate response. For blocking calls, this function always returns E_RMM_UNK.

The platform needs to ensure proper exclusives are in place when accessed from multiple CPUs. Depending on the expected latency for IDE-KM interface, the platform should choose blocking or non-blocking semantics. More details about IDE Setup flow can be found in this RFC.

The parameters of the function are:

> arg0 - The ecam address, to access and configure PCI devices in a system.
>
> arg1 - The rootport ID used to identify the PCIe rootport of a connected device.
>
> arg2 - Retrieved response corresponding to the previous IDE_KM request.

arg3 - returns the passthrough request ID of the retrieved response.

arg4 - returns the passthrough cookie of the retrieved response.

The function returns E_RMM_OK if response is retrieved successfully, E_RMM_INVAL if arguments to this function are invalid, E_RMM_UNK if response retrieval failed for an unknown error or IDE-KM interface is having blocking semantics, E_RMM_AGAIN if the response queue is empty.

The *arg2* return parameter can return the following values: E_RMM_OK - The previous request was successful. E_RMM_FAULT - The previous request was not successful. E_RMM_INVAL - Arguments to previous request were incorrect. E_RMM_UNK - Previous request returned Unknown error.

6.13.19 Function : bl31_plat_enable_mmu [optional]

```
Argument : uint32_t
Return   : void
```

This function enables the MMU. The boot code calls this function with MMU and caches disabled. This function should program necessary registers to enable translation, and upon return, the MMU on the calling PE must be enabled.

The function must honor flags passed in the first argument. These flags are defined by the translation library, and can be found in the file `include/lib/xlat_tables/xlat_mmu_helpers.h`.

On DynamIQ systems, this function must not use stack while enabling MMU, which is how the function in xlat table library version 2 is implemented.

6.13.20 Function : plat_init_apkey [optional]

```
Argument : void
Return   : uint128_t
```

This function returns the 128-bit value which can be used to program ARMv8.3 pointer authentication keys.

The value should be obtained from a reliable source of randomness. It will be called each time a core powers up and it is the platform's responsibility to decide when to regenerate the keys if generating them is an expensive operation.

This function is only needed if ARMv8.3 pointer authentication is used in the Trusted Firmware by building with `BRANCH_PROTECTION` option set to 1, 2 or 3.

6.13.21 Function : plat_get_syscnt_freq2() [mandatory]

```
Argument : void
Return   : unsigned int
```

This function is used by the architecture setup code to retrieve the counter frequency for the CPU's generic timer. This value will be programmed into the `CNTFRQ_EL0` register. In Arm standard platforms, it returns the base frequency of the system counter, which is retrieved from the first entry in the frequency modes table.

6.13.22 #define : PLAT_PERCPU_BAKERY_LOCK_SIZE [optional]

When `USE_COHERENT_MEM = 0`, this constant defines the total memory (in bytes) aligned to the cache line boundary that should be allocated per-cpu to accommodate all the bakery locks.

If this constant is not defined when `USE_COHERENT_MEM = 0`, the linker calculates the size of the `.bakery_lock` input section, aligns it to the nearest `CACHE_WRITEBACK_GRANULE`, multiplies it with `PLATFORM_CORE_COUNT` and stores the result in a linker symbol. This constant prevents a platform from relying on the linker and provide a more efficient mechanism for accessing per-cpu bakery lock information.

If this constant is defined and its value is not equal to the value calculated by the linker then a link time assertion is raised. A compile time assertion is raised if the value of the constant is not aligned to the cache line boundary.

6.13.23 SDEI porting requirements

The *SDEI* dispatcher requires the platform to provide the following macros and functions, of which some are optional, and some others mandatory.

Macros

Macro: PLAT_SDEI_NORMAL_PRI [mandatory]

This macro must be defined to the EL3 exception priority level associated with Normal *SDEI* events on the platform. This must have a higher value (therefore of lower priority) than `PLAT_SDEI_CRITICAL_PRI`.

Macro: PLAT_SDEI_CRITICAL_PRI [mandatory]

This macro must be defined to the EL3 exception priority level associated with Critical *SDEI* events on the platform. This must have a lower value (therefore of higher priority) than `PLAT_SDEI_NORMAL_PRI`.

Note: *SDEI* exception priorities must be the lowest among Secure priorities. Among the *SDEI* exceptions, Critical *SDEI* priority must be higher than Normal *SDEI* priority.

Functions

Function: int plat_sdei_validate_entry_point() [optional]

```
Argument: uintptr_t ep, unsigned int client_mode
Return: int
```

This function validates the entry point address of the event handler provided by the client for both event registration and *Complete and Resume* *SDEI* calls. The function ensures that the address is valid in the client translation regime.

The second argument is the exception level that the client is executing in. It can be Non-Secure EL1 or Non-Secure EL2.

The function must return `0` for successful validation, or `-1` upon failure.

The default implementation always returns `0`. On Arm platforms, this function translates the entry point address within the client translation regime and further ensures that the resulting physical address is located in Non-secure DRAM.

Function: void plat_sdei_handle_masked_trigger(uint64_t mpidr, unsigned int intr) [optional]

```
Argument: uint64_t
Argument: unsigned int
Return: void
```

SDEI specification requires that a PE comes out of reset with the events masked. The client therefore is expected to call `PE_UNMASK` to unmask *SDEI* events on the PE. No *SDEI* events can be dispatched until such time.

Should a PE receive an interrupt that was bound to an *SDEI* event while the events are masked on the PE, the dispatcher implementation invokes the function `plat_sdei_handle_masked_trigger`. The MPIDR of the PE that received the interrupt and the interrupt ID are passed as parameters.

The default implementation only prints out a warning message.

6.13.24 TRNG porting requirements

The *TRNG* backend requires the platform to provide the following values and mandatory functions.

Values

value: uuid_t plat_trng_uuid [mandatory]

This value must be defined to the UUID of the TRNG backend that is specific to the hardware after `plat_entropy_setup` function is called. This value must conform to the SMCCC calling convention; The most significant 32 bits of the UUID must not equal `0xffffffff` or the signed integer `-1` as this value in w0 indicates failure to get a TRNG source.

Functions

Function: void plat_entropy_setup(void) [mandatory]

```
Argument: none
Return: none
```

This function is expected to do platform-specific initialization of any TRNG hardware. This may include generating a UUID from a hardware-specific seed.

Function: bool plat_get_entropy(uint64_t *out) [mandatory]

```
Argument: uint64_t *
Return: bool
Out : when the return value is true, the entropy has been written into the
storage pointed to
```

This function writes entropy into storage provided by the caller. If no entropy is available, it must return false and the storage must not be written.

6.14 Power State Coordination Interface (PSCI)

The TF-A implementation of the PSCI API is based around the concept of a *power domain*. A *power domain* is a CPU or a logical group of CPUs which share some state on which power management operations can be performed as specified by PSCI. Each CPU in the system is assigned a cpu index which is a unique number between `0` and `PLATFORM_CORE_COUNT - 1`. The *power domains* are arranged in a hierarchical tree structure and each *power domain* can be identified in a system by the cpu index of any CPU that is part of that domain and a *power domain level*. A processing element (for example, a CPU) is at level 0. If the *power domain* node above a CPU is a logical grouping of CPUs that share some state, then level 1 is that group of CPUs (for example, a cluster), and level 2 is a group of clusters (for example, the system). More details on the power domain topology and its organization can be found in *PSCI Power Domain Tree Structure*.

BL31's platform initialization code exports a pointer to the platform-specific power management operations required for the PSCI implementation to function correctly. This information is populated in the `plat_psci_ops` structure. The PSCI implementation calls members of the `plat_psci_ops` structure for performing power management operations on the power domains. For example, the target CPU is specified by its MPIDR in a PSCI CPU_ON call. The pwr_domain_on() handler (if present) is called for the CPU power domain.

The power_state parameter of a PSCI CPU_SUSPEND call can be used to describe composite power states specific to a platform. The PSCI implementation defines a generic representation of the power-state parameter, which is an array of local power states where each index corresponds to a power domain level. Each entry contains the local power state the power domain at that power level could enter. It depends on the validate_power_state() handler to convert the power-state parameter (possibly encoding a composite power state) passed in a PSCI CPU_SUSPEND call to this representation.

This 32-bit power_state parameter has two encoding formats (as described in the ARM PSCI spec), controlled by the PSCI_EXTENDED_STATE_ID build option.

The state ID field in this is entirely platform-defined. Platforms must document their state ID conventions for OS developers. The power_state value is passed from the consumers (Like OSs) through generic PSCI layers to the respective platform handler.

The following functions form part of platform port of PSCI functionality.

6.14.1 Function : plat_psci_stat_accounting_start() [optional]

```
Argument : const psci_power_state_t *
Return   : void
```

This is an optional hook that platforms can implement for residency statistics accounting before entering a low power state. The pwr_domain_state field of state_info (first argument) can be inspected if stat accounting is done differently at CPU level versus higher levels. As an example, if the element at index 0 (CPU power level) in the pwr_domain_state array indicates a power down state, special hardware logic may be programmed in order to keep track of the residency statistics. For higher levels (array indices > 0), the residency statistics could be tracked in software using PMF. If ENABLE_PMF is set, the default implementation will use PMF to capture timestamps.

6.14.2 Function : plat_psci_stat_accounting_stop() [optional]

```
Argument : const psci_power_state_t *
Return   : void
```

This is an optional hook that platforms can implement for residency statistics accounting after exiting from a low power state. The pwr_domain_state field of state_info (first argument) can be inspected if stat accounting is done differently at CPU level versus higher levels. As an example, if the element at index 0 (CPU power level) in the pwr_domain_state array indicates a power down state, special hardware logic may be programmed in order to keep track of the residency statistics. For higher levels (array indices > 0), the residency statistics could be tracked in software using PMF. If ENABLE_PMF is set, the default implementation will use PMF to capture timestamps.

6.14.3 Function : plat_psci_stat_get_residency() [optional]

```
Argument : unsigned int, const psci_power_state_t *, unsigned int
Return   : u_register_t
```

This is an optional interface that is is invoked after resuming from a low power state and provides the time spent resident in that low power state by the power domain at a particular power domain level. When a CPU wakes up from suspend, all its parent power domain levels are also woken up. The generic PSCI code invokes this function for each parent power domain that is resumed and it identified by the `lvl` (first argument) parameter. The `state_info` (second argument) describes the low power state that the power domain has resumed from. The current CPU is the first CPU in the power domain to resume from the low power state and the `last_cpu_idx` (third parameter) is the index of the last CPU in the power domain to suspend and may be needed to calculate the residency for that power domain.

6.14.4 Function : plat_get_target_pwr_state() [optional]

```
Argument : unsigned int, const plat_local_state_t *, unsigned int
Return   : plat_local_state_t
```

The PSCI generic code uses this function to let the platform participate in state coordination during a power management operation. The function is passed a pointer to an array of platform specific local power state `states` (second argument) which contains the requested power state for each CPU at a particular power domain level `lvl` (first argument) within the power domain. The function is expected to traverse this array of upto `ncpus` (third argument) and return a coordinated target power state by the comparing all the requested power states. The target power state should not be deeper than any of the requested power states.

A weak definition of this API is provided by default wherein it assumes that the platform assigns a local state value in order of increasing depth of the power state i.e. for two power states X & Y, if X < Y then X represents a shallower power state than Y. As a result, the coordinated target local power state for a power domain will be the minimum of the requested local power state values.

6.14.5 Function : plat_get_power_domain_tree_desc() [mandatory]

```
Argument : void
Return   : const unsigned char *
```

This function returns a pointer to the byte array containing the power domain topology tree description. The format and method to construct this array are described in *PSCI Power Domain Tree Structure*. The BL31 PSCI initialization code requires this array to be described by the platform, either statically or dynamically, to initialize the power domain topology tree. In case the array is populated dynamically, then plat_core_pos_by_mpidr() and plat_my_core_pos() should also be implemented suitably so that the topology tree description matches the CPU indices returned by these APIs. These APIs together form the platform interface for the PSCI topology framework.

6.14.6 Function : plat_setup_psci_ops() [mandatory]

```
Argument : uintptr_t, const plat_psci_ops **
Return   : int
```

This function may execute with the MMU and data caches enabled if the platform port does the necessary initializations in `bl31_plat_arch_setup()`. It is only called by the primary CPU.

This function is called by PSCI initialization code. Its purpose is to let the platform layer know about the warm boot entrypoint through the `sec_entrypoint` (first argument) and to export handler routines for platform-specific psci power management actions by populating the passed pointer with a pointer to BL31's private `plat_psci_ops` structure.

A description of each member of this structure is given below. Please refer to the Arm FVP specific implementation of these handlers in `plat/arm/board/fvp/fvp_pm.c` as an example. For each PSCI function that the platform wants to support, the associated operation or operations in this structure must be provided and implemented (Refer section 4 of *Firmware Design* for the PSCI API supported in TF-A). To disable a PSCI function in a platform port, the operation should be removed from this structure instead of providing an empty implementation.

plat_psci_ops.cpu_standby()

Perform the platform-specific actions to enter the standby state for a cpu indicated by the passed argument. This provides a fast path for CPU standby wherein overheads of PSCI state management and lock acquisition is avoided. For this handler to be invoked by the PSCI `CPU_SUSPEND` API implementation, the suspend state type specified in the `power-state` parameter should be STANDBY and the target power domain level specified should be the CPU. The handler should put the CPU into a low power retention state (usually by issuing a wfi instruction) and ensure that it can be woken up from that state by a normal interrupt. The generic code expects the handler to succeed.

plat_psci_ops.pwr_domain_on()

Perform the platform specific actions to power on a CPU, specified by the `MPIDR` (first argument). The generic code expects the platform to return PSCI_E_SUCCESS on success or PSCI_E_INTERN_FAIL for any failure.

plat_psci_ops.pwr_domain_off_early() [optional]

This optional function performs the platform specific actions to check if powering off the calling CPU and its higher parent power domain levels as indicated by the `target_state` (first argument) is possible or allowed.

The `target_state` encodes the platform coordinated target local power states for the CPU power domain and its parent power domain levels.

For this handler, the local power state for the CPU power domain will be a power down state where as it could be either power down, retention or run state for the higher power domain levels depending on the result of state coordination. The generic code expects PSCI_E_DENIED return code if the platform thinks that CPU_OFF should not proceed on the calling CPU.

plat_psci_ops.pwr_domain_off()

Perform the platform specific actions to prepare to power off the calling CPU and its higher parent power domain levels as indicated by the `target_state` (first argument). It is called by the PSCI `CPU_OFF` API implementation.

The `target_state` encodes the platform coordinated target local power states for the CPU power domain and its parent power domain levels. The handler needs to perform power management operation corresponding to the local state at each power level.

For this handler, the local power state for the CPU power domain will be a power down state where as it could be either power down, retention or run state for the higher power domain levels depending on the result of state coordination. The generic code expects the handler to succeed.

plat_psci_ops.pwr_domain_validate_suspend() [optional]

This is an optional function that is only compiled into the build if the build option `PSCI_OS_INIT_MODE` is enabled.

If implemented, this function allows the platform to perform platform specific validations based on hardware states. The generic code expects this function to return PSCI_E_SUCCESS on success, or either PSCI_E_DENIED or PSCI_E_INVALID_PARAMS as appropriate for any invalid requests.

plat_psci_ops.pwr_domain_suspend_pwrdown_early() [optional]

This optional function may be used as a performance optimization to replace or complement pwr_domain_suspend() on some platforms. Its calling semantics are identical to pwr_domain_suspend(), except the PSCI implementation only calls this function when suspending to a power down state, and it guarantees that data caches are enabled.

When HW_ASSISTED_COHERENCY = 0, the PSCI implementation disables data caches before calling pwr_domain_suspend(). If the target_state corresponds to a power down state and it is safe to perform some or all of the platform specific actions in that function with data caches enabled, it may be more efficient to move those actions to this function. When HW_ASSISTED_COHERENCY = 1, data caches remain enabled throughout, and so there is no advantage to moving platform specific actions to this function.

plat_psci_ops.pwr_domain_suspend()

Perform the platform specific actions to prepare to suspend the calling CPU and its higher parent power domain levels as indicated by the `target_state` (first argument). It is called by the PSCI `CPU_SUSPEND` API implementation.

The `target_state` has a similar meaning as described in the `pwr_domain_off()` operation. It encodes the platform coordinated target local power states for the CPU power domain and its parent power domain levels. The handler needs to perform power management operation corresponding to the local state at each power level. The generic code expects the handler to succeed.

The difference between turning a power domain off versus suspending it is that in the former case, the power domain is expected to re-initialize its state when it is next powered on (see `pwr_domain_on_finish()`). In the latter case, the power domain is expected to save enough state so that it can resume execution by restoring this state when its powered on (see `pwr_domain_suspend_finish()`).

When suspending a core, the platform can also choose to power off the GICv3 Redistributor and ITS through an implementation-defined sequence. To achieve this safely, the ITS context must be saved first. The architectural part is implemented by the `gicv3_its_save_disable()` helper, but most of the needed sequence is implementation defined and it is therefore the responsibility of the platform code to implement the necessary sequence. Then the GIC Redistributor context can be saved using the `gicv3_rdistif_save()` helper. Powering off the Redistributor requires the implementation to support it and it is the responsibility of the platform code to execute the right implementation defined sequence.

When a system suspend is requested, the platform can also make use of the `gicv3_distif_save()` helper to save the context of the GIC Distributor after it has saved the context of the Redistributors and ITS of all the cores in the system. The context of the Distributor can be large and may require it to be allocated in a special area if it cannot fit in the platform's global static data, for example in DRAM. The Distributor can then be powered down using an implementation-defined sequence.

plat_psci_ops.pwr_domain_pwr_down()

This is an optional function and, if implemented, is expected to perform platform specific actions before the CPU is powered down. Since this function is invoked outside the PSCI locks, the actions performed in this hook must be local to the CPU or the platform must ensure that races between multiple CPUs cannot occur.

The `target_state` has a similar meaning as described in the `pwr_domain_off()` operation and it encodes the platform coordinated target local power states for the CPU power domain and its parent power domain levels.

It is preferred that this function returns. The caller will invoke `wfi()` to powerdown the CPU, mitigate any powerdown errata, and handle any wakeups that may arise. Previously, this function did not return and instead called `wfi` (in an infinite loop) directly. This is still possible on platforms where this is guaranteed to be terminal, however, it is strongly discouraged going forward.

Previously this function was called `pwr_domain_pwr_down_wfi()` and invoked `psci_power_down_wfi()` (now removed).

plat_psci_ops.pwr_domain_on_finish()

This function is called by the PSCI implementation after the calling CPU is powered on and released from reset in response to an earlier PSCI `CPU_ON` call. It performs the platform-specific setup required to initialize enough state for this CPU to enter the normal world and also provide secure runtime firmware services.

The `target_state` (first argument) is the prior state of the power domains immediately before the CPU was turned on. It indicates which power domains above the CPU might require initialization due to having previously been in low power states. The generic code expects the handler to succeed.

plat_psci_ops.pwr_domain_on_finish_late() [optional]

This optional function is called by the PSCI implementation after the calling CPU is fully powered on with respective data caches enabled. The calling CPU and the associated cluster are guaranteed to be participating in coherency. This function gives the flexibility to perform any platform-specific actions safely, such as initialization or modification of shared data structures, without the overhead of explicit cache maintainace operations.

The `target_state` has a similar meaning as described in the `pwr_domain_on_finish()` operation. The generic code expects the handler to succeed.

plat_psci_ops.pwr_domain_suspend_finish()

This function is called by the PSCI implementation after the calling CPU is powered on and released from reset in response to an asynchronous wakeup event, for example a timer interrupt that was programmed by the CPU during the `CPU_SUSPEND` call or `SYSTEM_SUSPEND` call. It performs the platform-specific setup required to restore the saved state for this CPU to resume execution in the normal world and also provide secure runtime firmware services.

The `target_state` (first argument) has a similar meaning as described in the `pwr_domain_on_finish()` operation. The generic code expects the platform to succeed.

If the Distributor, Redistributors or ITS have been powered off as part of a suspend, their context must be restored in this function in the reverse order to how they were saved during suspend sequence.

plat_psci_ops.system_off()

This function is called by PSCI implementation in response to a `SYSTEM_OFF` call. It performs the platform-specific system poweroff sequence after notifying the Secure Payload Dispatcher. The caller will call `wfi` if this function returns, similar to *plat_psci_ops.pwr_domain_pwr_down()*.

plat_psci_ops.system_reset()

This function is called by PSCI implementation in response to a `SYSTEM_RESET` call. It performs the platform-specific system reset sequence after notifying the Secure Payload Dispatcher. The caller will call `wfi` if this function returns, similar to *plat_psci_ops.pwr_domain_pwr_down()*.

plat_psci_ops.validate_power_state()

This function is called by the PSCI implementation during the `CPU_SUSPEND` call to validate the `power_state` parameter of the PSCI API and if valid, populate it in `req_state` (second argument) array as power domain level specific local states. If the `power_state` is invalid, the platform must return PSCI_E_INVALID_PARAMS as error, which is propagated back to the normal world PSCI client.

plat_psci_ops.validate_ns_entrypoint()

This function is called by the PSCI implementation during the `CPU_SUSPEND`, `SYSTEM_SUSPEND` and `CPU_ON` calls to validate the non-secure `entry_point` parameter passed by the normal world. If the `entry_point` is invalid, the platform must return PSCI_E_INVALID_ADDRESS as error, which is propagated back to the normal world PSCI client.

plat_psci_ops.get_sys_suspend_power_state()

This function is called by the PSCI implementation during the `SYSTEM_SUSPEND` call to get the `req_state` parameter from platform which encodes the power domain level specific local states to suspend to system affinity level. The `req_state` will be utilized to do the PSCI state coordination and `pwr_domain_suspend()` will be invoked with the coordinated target state to enter system suspend.

plat_psci_ops.get_pwr_lvl_state_idx()

This is an optional function and, if implemented, is invoked by the PSCI implementation to convert the `local_state` (first argument) at a specified `pwr_lvl` (second argument) to an index between 0 and `PLAT_MAX_PWR_LVL_STATES` - 1. This function is only needed if the platform supports more than two local power states at each power domain level, that is `PLAT_MAX_PWR_LVL_STATES` is greater than 2, and needs to account for these local power states.

plat_psci_ops.translate_power_state_by_mpidr()

This is an optional function and, if implemented, verifies the `power_state` (second argument) parameter of the PSCI API corresponding to a target power domain. The target power domain is identified by using both `MPIDR` (first argument) and the power domain level encoded in `power_state`. The power domain level specific local states are to be extracted from `power_state` and be populated in the `output_state` (third argument) array. The functionality is similar to the `validate_power_state` function described above and is envisaged to be used in case the validity of `power_state` depend on the targeted power domain. If the `power_state` is invalid for the targeted power domain, the platform must return PSCI_E_INVALID_PARAMS as error. If this function is not implemented, then the generic implementation relies on `validate_power_state` function to translate the `power_state`.

This function can also be used in case the platform wants to support local power state encoding for `power_state` parameter of PSCI_STAT_COUNT/RESIDENCY APIs as described in Section 5.18 of PSCI.

plat_psci_ops.get_node_hw_state()

This is an optional function. If implemented this function is intended to return the power state of a node (identified by the first parameter, the MPIDR) in the power domain topology (identified by the second parameter, power_level), as retrieved from a power controller or equivalent component on the platform. Upon successful completion, the implementation must map and return the final status among HW_ON, HW_OFF or HW_STANDBY. Upon encountering failures, it must return either PSCI_E_INVALID_PARAMS or PSCI_E_NOT_SUPPORTED as appropriate.

Implementations are not expected to handle power_levels greater than PLAT_MAX_PWR_LVL.

plat_psci_ops.system_reset2()

This is an optional function. If implemented this function is called during the SYSTEM_RESET2 call to perform a reset based on the first parameter reset_type as specified in PSCI. The parameter cookie can be used to pass additional reset information. If the reset_type is not supported, the function must return PSCI_E_NOT_SUPPORTED. For architectural resets, all failures must return PSCI_E_INVALID_PARAMETERS and vendor reset can return other PSCI error codes as defined in PSCI. If this function returns success, the caller will call wfi similar to *plat_psci_ops.pwr_domain_pwr_down()*.

plat_psci_ops.write_mem_protect()

This is an optional function. If implemented it enables or disables the MEM_PROTECT functionality based on the value of val. A non-zero value enables MEM_PROTECT and a value of zero disables it. Upon encountering failures it must return a negative value and on success it must return 0.

plat_psci_ops.read_mem_protect()

This is an optional function. If implemented it returns the current state of MEM_PROTECT via the val parameter. Upon encountering failures it must return a negative value and on success it must return 0.

plat_psci_ops.mem_protect_chk()

This is an optional function. If implemented it checks if a memory region defined by a base address base and with a size of length bytes is protected by MEM_PROTECT. If the region is protected then it must return 0, otherwise it must return a negative number.

6.14.7 Secure payload power management callback

During PSCI power management operations, the EL3 Runtime Software may need to perform some bookkeeping, and PSCI library provides `spd_pm_ops_t` callbacks for this purpose. These hooks must be populated and registered by using `psci_register_spd_pm_hook()` PSCI library interface.

Typical bookkeeping during PSCI power management calls include save/restore of the EL3 Runtime Software context. Also if the EL3 Runtime Software makes use of secure interrupts, then these interrupts must also be managed appropriately during CPU power down/power up. Any secure interrupt targeted to the current CPU must be disabled or re-targeted to other running CPU prior to power down of the current CPU. During power up, these interrupt can be enabled/re-targeted back to the current CPU.

```
typedef struct spd_pm_ops {
        void (*svc_on)(u_register_t target_cpu);
        int32_t (*svc_off)(u_register_t __unused);
        void (*svc_suspend)(u_register_t max_off_pwrlvl);
        void (*svc_on_finish)(u_register_t __unused);
        void (*svc_suspend_finish)(u_register_t max_off_pwrlvl);
        int32_t (*svc_migrate)(u_register_t from_cpu, u_register_t to_cpu);
        int32_t (*svc_migrate_info)(u_register_t *resident_cpu);
        void (*svc_system_off)(void);
        void (*svc_system_reset)(void);
} spd_pm_ops_t;
```

A brief description of each callback is given below:

- svc_on, svc_off, svc_on_finish

  The `svc_on`, `svc_off` callbacks are called during PSCI_CPU_ON, PSCI_CPU_OFF APIs respectively. The `svc_on_finish` is called when the target CPU of PSCI_CPU_ON API powers up and executes the `psci_warmboot_entrypoint()` PSCI library interface.

- svc_suspend, svc_suspend_finish

  The `svc_suspend` callback is called during power down bu either PSCI_SUSPEND or PSCI_SYSTEM_SUSPEND APIs. The `svc_suspend_finish` is called when the CPU wakes up from suspend and executes the `psci_warmboot_entrypoint()` PSCI library interface. The `max_off_pwrlvl` (first parameter) denotes the highest power domain level being powered down to or woken up from suspend.

- svc_system_off, svc_system_reset

  These callbacks are called during PSCI_SYSTEM_OFF and PSCI_SYSTEM_RESET PSCI APIs respectively.

- svc_migrate_info

  This callback is called in response to PSCI_MIGRATE_INFO_TYPE or PSCI_MIGRATE_INFO_UP_CPU APIs. The return value of this callback must correspond to the return value of PSCI_MIGRATE_INFO_TYPE API as described in PSCI. If the secure payload is a Uniprocessor (UP) implementation, then it must update the mpidr of the CPU it is resident in via `resident_cpu` (first argument). The updates to `resident_cpu` is ignored if the secure payload is a multiprocessor (MP) implementation.

- svc_migrate

  This callback is only relevant if the secure payload in EL3 Runtime Software is a Uniprocessor (UP) implementation and supports migration from the current CPU `from_cpu` (first argument) to another CPU `to_cpu` (second argument). This callback is called in response to PSCI_MIGRATE API. This callback is never called if the secure payload is a Multiprocessor (MP) implementation.

6.15 Interrupt Management framework (in BL31)

BL31 implements an Interrupt Management Framework (IMF) to manage interrupts generated in either security state and targeted to EL1 or EL2 in the non-secure state or EL3/S-EL1 in the secure state. The design of this framework is described in the *Interrupt Management Framework*

A platform should export the following APIs to support the IMF. The following text briefly describes each API and its implementation in Arm standard platforms. The API implementation depends upon the type of interrupt controller present in the platform. Arm standard platform layer supports both Arm Generic Interrupt Controller version 2.0 (GICv2) and 3.0 (GICv3). Juno builds the Arm platform layer to use GICv2 and the FVP can be configured to use either GICv2 or GICv3 depending on the build flag `FVP_USE_GIC_DRIVER` (See *Arm FVP Platform Specific Build Options* for more details).

See also: *Interrupt Controller Abstraction APIs*.

6.15.1 Function : plat_interrupt_type_to_line() [mandatory]

```
Argument : uint32_t, uint32_t
Return   : uint32_t
```

The Arm processor signals an interrupt exception either through the IRQ or FIQ interrupt line. The specific line that is signaled depends on how the interrupt controller (IC) reports different interrupt types from an execution context in either security state. The IMF uses this API to determine which interrupt line the platform IC uses to signal each type of interrupt supported by the framework from a given security state. This API must be invoked at EL3.

The first parameter will be one of the `INTR_TYPE_*` values (see *Interrupt Management Framework*) indicating the target type of the interrupt, the second parameter is the security state of the originating execution context. The return result is the bit position in the `SCR_EL3` register of the respective interrupt trap: IRQ=1, FIQ=2.

In the case of Arm standard platforms using GICv2, S-EL1 interrupts are configured as FIQs and Non-secure interrupts as IRQs from either security state.

In the case of Arm standard platforms using GICv3, the interrupt line to be configured depends on the security state of the execution context when the interrupt is signalled and are as follows:

- The S-EL1 interrupts are signaled as IRQ in S-EL0/1 context and as FIQ in NS-EL0/1/2 context.
- The Non secure interrupts are signaled as FIQ in S-EL0/1 context and as IRQ in the NS-EL0/1/2 context.
- The EL3 interrupts are signaled as FIQ in both S-EL0/1 and NS-EL0/1/2 context.

6.15.2 Function : plat_ic_get_pending_interrupt_type() [mandatory]

```
Argument : void
Return   : uint32_t
```

This API returns the type of the highest priority pending interrupt at the platform IC. The IMF uses the interrupt type to retrieve the corresponding handler function. `INTR_TYPE_INVAL` is returned when there is no interrupt pending. The valid interrupt types that can be returned are `INTR_TYPE_EL3`, `INTR_TYPE_S_EL1` and `INTR_TYPE_NS`. This API must be invoked at EL3.

In the case of Arm standard platforms using GICv2, the *Highest Priority Pending Interrupt Register* (`GICC_HPPIR`) is read to determine the id of the pending interrupt. The type of interrupt depends upon the id value as follows.

1. id < 1022 is reported as a S-EL1 interrupt
2. id = 1022 is reported as a Non-secure interrupt.
3. id = 1023 is reported as an invalid interrupt type.

In the case of Arm standard platforms using GICv3, the system register `ICC_HPPIR0_EL1`, *Highest Priority Pending group 0 Interrupt Register*, is read to determine the id of the pending interrupt. The type of interrupt depends upon the id value as follows.

1. id = `PENDING_G1S_INTID` (1020) is reported as a S-EL1 interrupt
2. id = `PENDING_G1NS_INTID` (1021) is reported as a Non-secure interrupt.
3. id = `GIC_SPURIOUS_INTERRUPT` (1023) is reported as an invalid interrupt type.
4. All other interrupt id's are reported as EL3 interrupt.

6.15.3 Function : plat_ic_get_pending_interrupt_id() [mandatory]

```
Argument : void
Return   : uint32_t
```

This API returns the id of the highest priority pending interrupt at the platform IC. `INTR_ID_UNAVAILABLE` is returned when there is no interrupt pending.

In the case of Arm standard platforms using GICv2, the *Highest Priority Pending Interrupt Register* (`GICC_HPPIR`) is read to determine the id of the pending interrupt. The id that is returned by API depends upon the value of the id read from the interrupt controller as follows.

1. id < 1022. id is returned as is.
2. id = 1022. The *Aliased Highest Priority Pending Interrupt Register* (`GICC_AHPPIR`) is read to determine the id of the non-secure interrupt. This id is returned by the API.
3. id = 1023. `INTR_ID_UNAVAILABLE` is returned.

In the case of Arm standard platforms using GICv3, if the API is invoked from EL3, the system register `ICC_HPPIR0_EL1`, *Highest Priority Pending Interrupt group 0 Register*, is read to determine the id of the pending interrupt. The id that is returned by API depends upon the value of the id read from the interrupt controller as follows.

1. id < `PENDING_G1S_INTID` (1020). id is returned as is.
2. id = `PENDING_G1S_INTID` (1020) or `PENDING_G1NS_INTID` (1021). The system register `ICC_HPPIR1_EL1`, *Highest Priority Pending Interrupt group 1 Register* is read to determine the id of the group 1 interrupt. This id is returned by the API as long as it is a valid interrupt id
3. If the id is any of the special interrupt identifiers, `INTR_ID_UNAVAILABLE` is returned.

When the API invoked from S-EL1 for GICv3 systems, the id read from system register `ICC_HPPIR1_EL1`, *Highest Priority Pending group 1 Interrupt Register*, is returned if is not equal to GIC_SPURIOUS_INTERRUPT (1023) else `INTR_ID_UNAVAILABLE` is returned.

6.15.4 Function : plat_ic_acknowledge_interrupt() [mandatory]

```
Argument : void
Return   : uint32_t
```

This API is used by the CPU to indicate to the platform IC that processing of the highest pending interrupt has begun. It should return the raw, unmodified value obtained from the interrupt controller when acknowledging an interrupt. The actual interrupt number shall be extracted from this raw value using the API *plat_ic_get_interrupt_id()<plat_ic_get_interrupt_id>*.

This function in Arm standard platforms using GICv2, reads the *Interrupt Acknowledge Register* (`GICC_IAR`). This changes the state of the highest priority pending interrupt from pending to active in the interrupt controller. It returns the value read from the `GICC_IAR`, unmodified.

In the case of Arm standard platforms using GICv3, if the API is invoked from EL3, the function reads the system register `ICC_IAR0_EL1`, *Interrupt Acknowledge Register group 0*. If the API is invoked from S-EL1, the function reads the system register `ICC_IAR1_EL1`, *Interrupt Acknowledge Register group 1*. The read changes the state of the highest pending interrupt from pending to active in the interrupt controller. The value read is returned unmodified.

The TSP uses this API to start processing of the secure physical timer interrupt.

6.15.5 Function : plat_ic_end_of_interrupt() [mandatory]

```
Argument : uint32_t
Return   : void
```

This API is used by the CPU to indicate to the platform IC that processing of the interrupt corresponding to the id (passed as the parameter) has finished. The id should be the same as the id returned by the `plat_ic_acknowledge_interrupt()` API.

Arm standard platforms write the id to the *End of Interrupt Register* (`GICC_EOIR`) in case of GICv2, and to `ICC_EOIR0_EL1` or `ICC_EOIR1_EL1` system register in case of GICv3 depending on where the API is invoked from, EL3 or S-EL1. This deactivates the corresponding interrupt in the interrupt controller.

The TSP uses this API to finish processing of the secure physical timer interrupt.

6.15.6 Function : plat_ic_get_interrupt_type() [mandatory]

```
Argument : uint32_t
Return   : uint32_t
```

This API returns the type of the interrupt id passed as the parameter. `INTR_TYPE_INVAL` is returned if the id is invalid. If the id is valid, a valid interrupt type (one of `INTR_TYPE_EL3`, `INTR_TYPE_S_EL1` and `INTR_TYPE_NS`) is returned depending upon how the interrupt has been configured by the platform IC. This API must be invoked at EL3.

Arm standard platforms using GICv2 configures S-EL1 interrupts as Group0 interrupts and Non-secure interrupts as Group1 interrupts. It reads the group value corresponding to the interrupt id from the relevant *Interrupt Group Register* (`GICD_IGROUPRn`). It uses the group value to determine the type of interrupt.

In the case of Arm standard platforms using GICv3, both the *Interrupt Group Register* (`GICD_IGROUPRn`) and *Interrupt Group Modifier Register* (`GICD_IGRPMODRn`) is read to figure out whether the interrupt is configured as Group 0 secure interrupt, Group 1 secure interrupt or Group 1 NS interrupt.

6.16 Registering a console

Platforms will need to implement the TF-A console framework to register and use a console for visual data output in TF-A. These can be used for data output during the different stages of the firmware boot process and also for debugging purposes.

The console framework can be used to output data on to a console using a number of TF-A supported UARTs. Multiple consoles can be registered at the same time with different output scopes (BOOT, RUNTIME, CRASH) so that data can be displayed on their respective consoles without unnecessary cluttering of a single console.

Information for registering a console can be found in the *Console Framework* section of the *System Design* documentation.

6.17 Common helper functions

6.17.1 Function : elx_panic()

```
Argument : void
Return   : void
```

This API is called from assembly files when reporting a critical failure that has occured in lower EL and is been trapped in EL3. This call **must not** return.

6.17.2 Function : el3_panic()

```
Argument : void
Return   : void
```

This API is called from assembly files when encountering a critical failure that cannot be recovered from. This function assumes that it is invoked from a C runtime environment i.e. valid stack exists. This call **must not** return.

6.17.3 Function : panic()

```
Argument : void
Return   : void
```

This API called from C files when encountering a critical failure that cannot be recovered from. This function in turn prints backtrace (if enabled) and calls el3_panic(). This call **must not** return.

6.18 Crash Reporting mechanism (in BL31)

BL31 implements a crash reporting mechanism which prints the various registers of the CPU to enable quick crash analysis and debugging. This mechanism relies on the platform implementing `plat_crash_console_init`, `plat_crash_console_putc` and `plat_crash_console_flush`.

The file `plat/common/aarch64/crash_console_helpers.S` contains sample implementation of all of them. Platforms may include this file to their makefiles in order to benefit from them. By default, they will cause the crash output to be routed over the normal console infrastructure and get printed on consoles configured to output in crash state. `console_set_scope()` can be used to control whether a console is used for crash output.

Note: Platforms are responsible for making sure that they only mark consoles for use in the crash scope that are able to support this, i.e. that are written in assembly and conform with the register clobber rules for putc() (x0-x2, x16-x17) and flush() (x0-x3, x16-x17) crash callbacks.

In some cases (such as debugging very early crashes that happen before the normal boot console can be set up), platforms may want to control crash output more explicitly. These platforms may instead provide custom implementations for these. They are executed outside of a C environment and without a stack. Many console drivers provide functions named `console_xxx_core_init/putc/flush` that are designed to be used by these functions. See Arm platforms (like juno) for an example of this.

6.18.1 Function : plat_crash_console_init [mandatory]

```
Argument : void
Return   : int
```

This API is used by the crash reporting mechanism to initialize the crash console. It must only use the general purpose registers x0 through x7 to do the initialization and returns 1 on success.

6.18.2 Function : plat_crash_console_putc [mandatory]

```
Argument : int
Return   : int
```

This API is used by the crash reporting mechanism to print a character on the designated crash console. It must only use general purpose registers x1 and x2 to do its work. The parameter and the return value are in general purpose register x0.

6.18.3 Function : plat_crash_console_flush [mandatory]

```
Argument : void
Return   : void
```

This API is used by the crash reporting mechanism to force write of all buffered data on the designated crash console. It should only use general purpose registers x0 through x5 to do its work.

6.18.4 Function : plat_setup_early_console [optional]

```
Argument : void
Return   : void
```

This API is used to setup the early console, it is required only if the flag `EARLY_CONSOLE` is enabled.

6.19 External Abort handling and RAS Support

If any cores on the platform support powerdown abandon (check the "Core powerup and powerdown sequence" in their TRMs), then these functions should be able to handle being called with power domains off and after the powerdown `wfi`. In other words it may run after a call to `pwr_domain_suspend()` and before a call to `pwr_domain_suspend_finish()` (and their power off counterparts).

Should this not be desirable, or if there is no powerdown abandon support, then RAS errors should be masked by writing any relevant error records in any powerdown hooks to prevent deadlocks due to a RAS error after the point of no return. See the core's TRM for further information.

6.19.1 Function : plat_ea_handler

```
Argument : int
Argument : uint64_t
Argument : void *
Argument : void *
Argument : uint64_t
Return   : void
```

This function is invoked by the runtime exception handling framework for the platform to handle an External Abort received at EL3. The intention of the function is to attempt to resolve the cause of External Abort and return; if that's not possible then an orderly shutdown of the system is initiated.

The first parameter (`int ea_reason`) indicates the reason for External Abort. Its value is one of `ERROR_EA_*` constants defined in `ea_handle.h`.

The second parameter (`uint64_t syndrome`) is the respective syndrome presented to EL3 after having received the External Abort. Depending on the nature of the abort (as can be inferred from the `ea_reason` parameter), this can be the content of either `ESR_EL3` or `DISR_EL1`.

The third parameter (`void *cookie`) is unused for now. The fourth parameter (`void *handle`) is a pointer to the preempted context. The fifth parameter (`uint64_t flags`) indicates the preempted security state. These parameters are received from the top-level exception handler.

This function must be implemented if a platform expects Firmware First handling of External Aborts.

6.19.2 Function : plat_handle_uncontainable_ea

```
Argument : int
Argument : uint64_t
Return   : void
```

This function is invoked by the RAS framework when an External Abort of Uncontainable type is received at EL3. Due to the critical nature of Uncontainable errors, the intention of this function is to initiate orderly shutdown of the system, and is not expected to return.

This function must be implemented in assembly.

The first and second parameters are the same as that of `plat_ea_handler`.

The default implementation of this function calls `report_unhandled_exception`.

6.19.3 Function : plat_handle_double_fault

```
Argument : int
Argument : uint64_t
Return   : void
```

This function is invoked by the RAS framework when another External Abort is received at EL3 while one is already being handled. I.e., a call to `plat_ea_handler` is outstanding. Due to its critical nature, the intention of this function is to initiate orderly shutdown of the system, and is not expected recover or return.

This function must be implemented in assembly.

The first and second parameters are the same as that of `plat_ea_handler`.

The default implementation of this function calls `report_unhandled_exception`.

6.19.4 Function : plat_handle_el3_ea

```
Return   : void
```

This function is invoked when an External Abort is received while executing in EL3. Due to its critical nature, the intention of this function is to initiate orderly shutdown of the system, and is not expected recover or return.

This function must be implemented in assembly.

The default implementation of this function calls `report_unhandled_exception`.

6.19.5 Function : plat_handle_rng_trap

```
Argument : u_register_t *
Argument : bool
Return   : int
```

This function is invoked by BL31's exception handler when there is a synchronous system register trap caused by access to the RNDR or RNDRRS registers. It allows platforms enabling `ENABLE_FEAT_RNG_TRAP` to emulate those system registers by returning back some entropy to the lower EL.

The first parameter (`u_register_t * data`) is a pointer to where the entropy should be written to.

The second parameter (`bool rndrrs`) represents whether the register access is `RNDR` or `RNDRRS`.

The return value indicates how to proceed:

- When returning `TRAP_RET_UNHANDLED` (-1), entropy was unavailable.
- When returning `TRAP_RET_CONTINUE` (1), entropy was available and written to `data`.

This function needs to be implemented by a platform if it enables FEAT_RNG_TRAP.

6.19.6 Function : plat_handle_impdef_trap

```
Argument : uint64_t
Argument : cpu_context_t *
Return   : int
```

This function is invoked by BL31's exception handler when there is a synchronous system register trap caused by access to the implementation defined registers. It allows platforms enabling `IMPDEF_SYSREG_TRAP` to emulate those system registers choosing to program bits of their choice. If using in combination with `ARCH_FEATURE_AVAILABILITY`, the macros {SCR,MDCR,CPTR}_PLAT_{BITS,IGNORED,FLIPPED} should be defined to report correct results.

The first parameter (`uint64_t esr_el3`) contains the content of the ESR_EL3 syndrome register, which encodes the instruction that was trapped.

The second parameter (`cpu_context_t *ctx`) represents the CPU state in the lower exception level, at the time when the execution of the `mrs` instruction was trapped.

The return value indicates how to proceed:

- When returning `TRAP_RET_UNHANDLED` (-1), the machine will panic.
- When returning `TRAP_RET_REPEAT` (0), the exception handler will return to the same instruction, so its execution will be repeated.
- When returning `TRAP_RET_CONTINUE` (1), the exception handler will return to the next instruction.

This function needs to be implemented by a platform if it enables IMPDEF_SYSREG_TRAP.

6.20 Build flags

There are some build flags which can be defined by the platform to control inclusion or exclusion of certain BL stages from the FIP image. These flags need to be defined in the platform makefile which will get included by the build system.

- **NEED_BL33** By default, this flag is defined `yes` by the build system and `BL33` build option should be supplied as a build option. The platform has the option of excluding the BL33 image in the `fip` image by defining this flag to `no`. If any of the options `EL3_PAYLOAD_BASE` or `PRELOADED_BL33_BASE` are used, this flag will be set to `no` automatically.
- **ARM_ARCH_MAJOR and ARM_ARCH_MINOR** By default, ARM_ARCH_MAJOR.ARM_ARCH_MINOR is set to 8.0 in `defaults.mk`, if the platform makefile/build defines or uses the correct ARM_ARCH_MAJOR and ARM_ARCH_MINOR then mandatory Architectural features available for that Arch version will be enabled by default and any optional Arch feature supported by the Architecture and available in TF-A can be enabled from platform specific makefile. Look up to `arch_features.mk` for details pertaining to mandatory and optional Arch specific features.

6.21 Platform include paths

Platforms are allowed to add more include paths to be passed to the compiler. The `PLAT_INCLUDES` variable is used for this purpose. This is needed in particular for the file `platform_def.h`.

Example:

```
PLAT_INCLUDES  += -Iinclude/plat/myplat/include
```

6.22 C Library

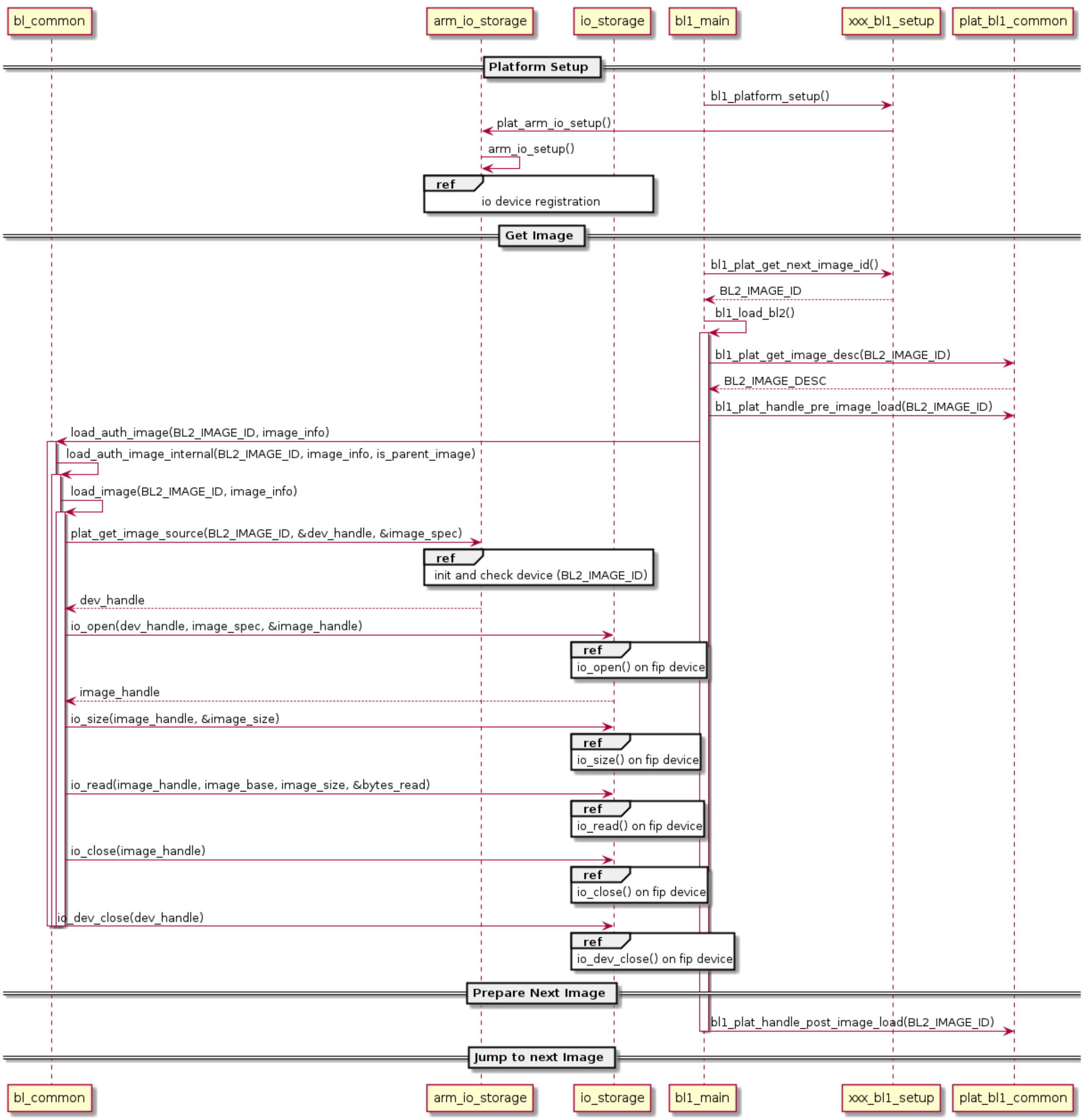
To avoid subtle toolchain behavioral dependencies, the header files provided by the compiler are not used. The software is built with the `-nostdinc` flag to ensure no headers are included from the toolchain inadvertently. Instead the required headers are included in the TF-A source tree. The library only contains those C library definitions required by the local implementation. If more functionality is required, the needed library functions will need to be added to the local implementation.

Some C headers have been obtained from FreeBSD and SCC, while others have been written specifically for TF-A. Some implementation files have been obtained from FreeBSD, others have been written specifically for TF-A as well. The files can be found in `include/lib/libc` and `lib/libc`.

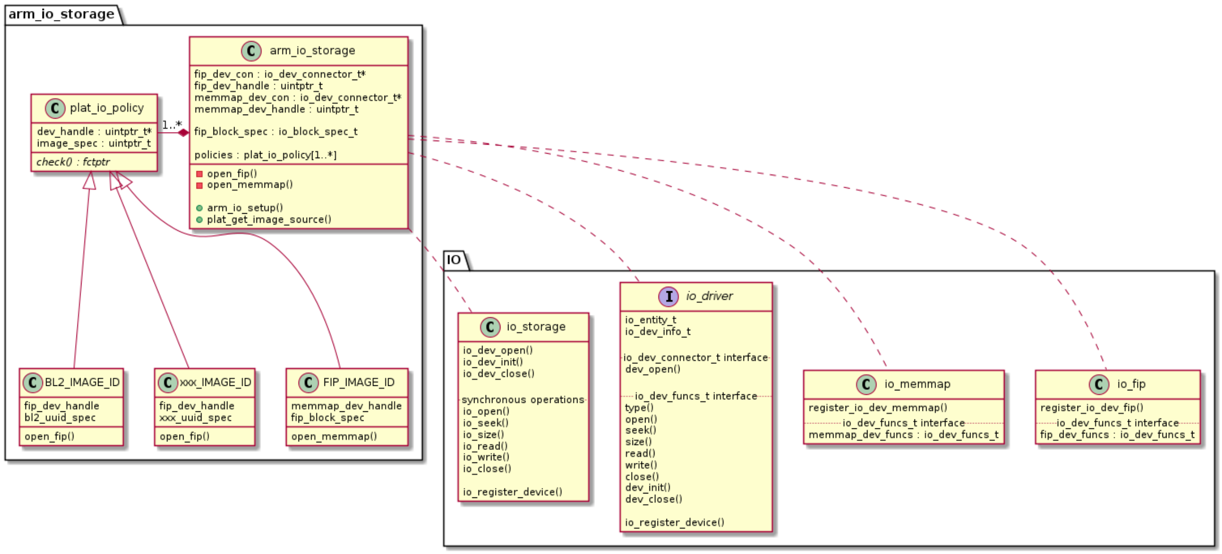
SCC can be found in http://www.simple-cc.org/. A copy of the FreeBSD sources can be obtained from http://github.com/freebsd/freebsd.

6.23 Storage abstraction layer

In order to improve platform independence and portability a storage abstraction layer is used to load data from non-volatile platform storage. Currently storage access is only required by BL1 and BL2 phases and performed inside the `load_image()` function in `bl_common.c`.

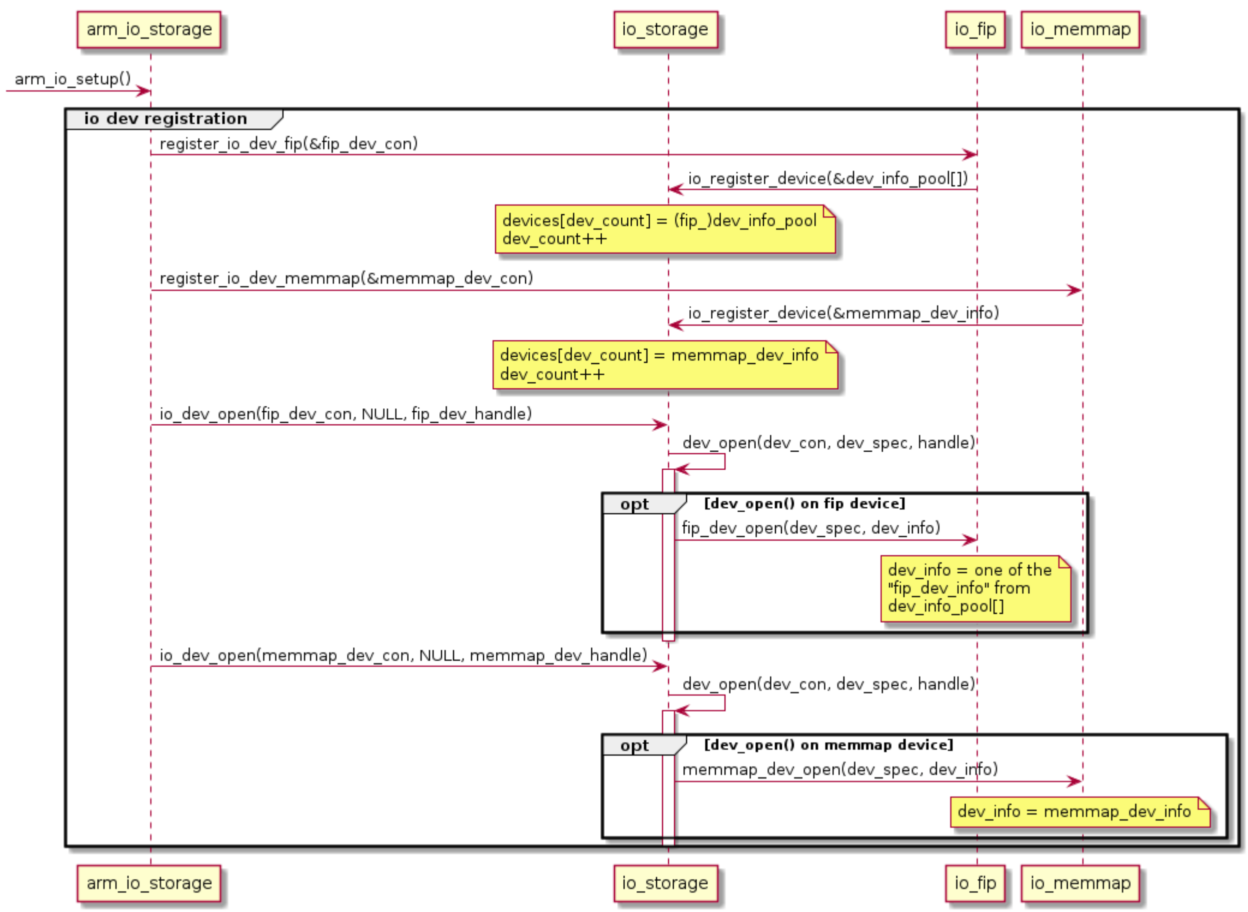


It is mandatory to implement at least one storage driver. For the Arm development platforms the Firmware Image Package (FIP) driver is provided as the default means to load data from storage (see *Firmware Image Package (FIP)*). The storage layer is described in the header file `include/drivers/io/io_storage.h`. The implementation of the common library is in `drivers/io/io_storage.c` and the driver files are located in `drivers/io/`.

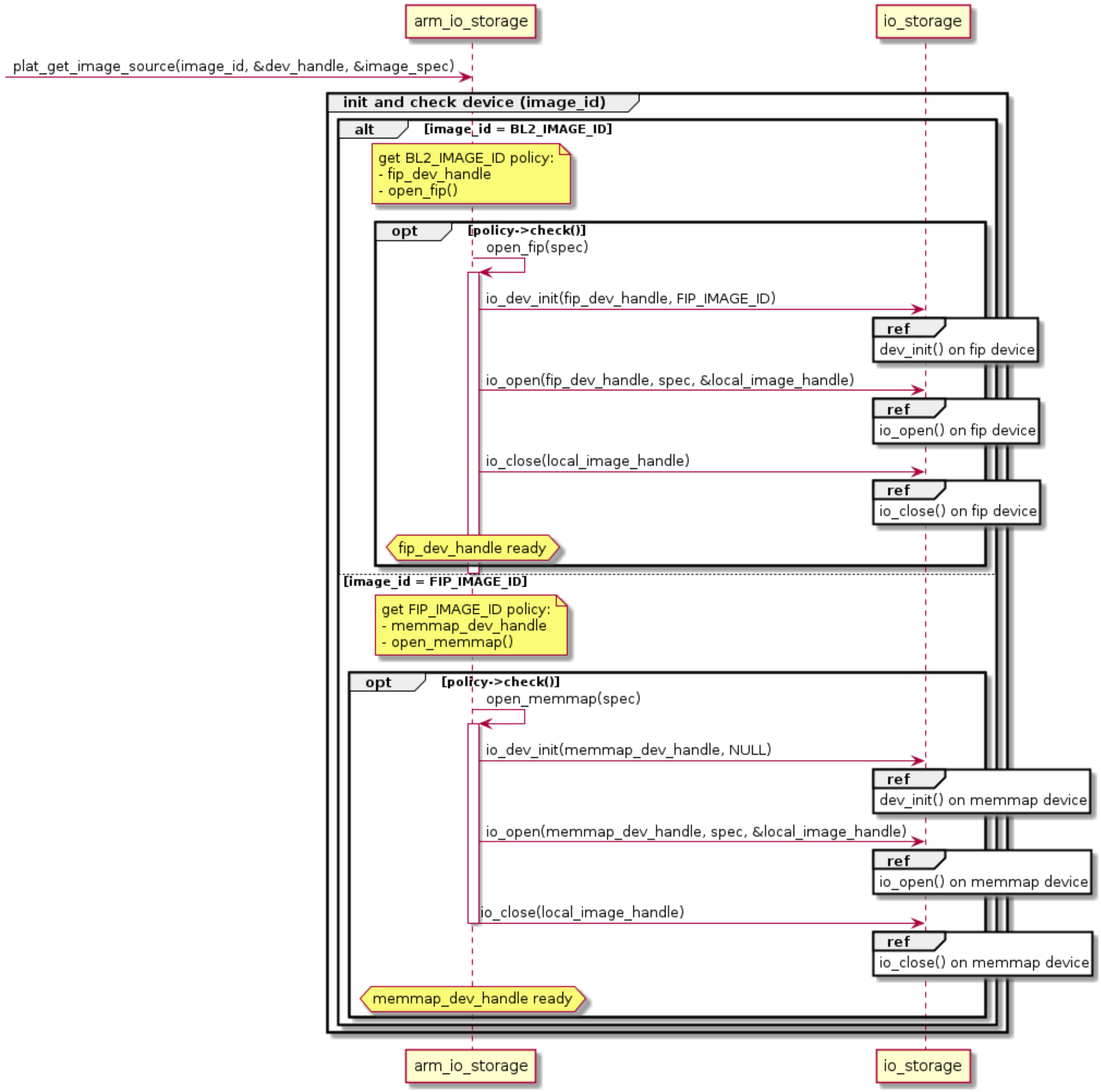


Each IO driver must provide `io_dev_*` structures, as described in `drivers/io/io_driver.h`. These are returned via a mandatory registration function that is called on platform initialization. The semi-hosting driver implementation in `io_semihosting.c` can be used as an example.

Each platform should register devices and their drivers via the storage abstraction layer. These drivers then need to be initialized by bootloader phases as required in their respective `blx_platform_setup()` functions.



The storage abstraction layer provides mechanisms (`io_dev_init()`) to initialize storage devices before IO operations are called.



The basic operations supported by the layer include `open()`, `close()`, `read()`, `write()`, `size()` and `seek()`. Drivers do not have to implement all operations, but each platform must provide at least one driver for a device capable of supporting generic operations such as loading a bootloader image.

The current implementation only allows for known images to be loaded by the firmware. These images are specified by using their identifiers, as defined in `include/plat/common/common_def.h` (or a separate header file included from there). The platform layer (`plat_get_image_source()`) then returns a reference to a device and a driver-specific `spec` which will be understood by the driver to allow access to the image data.

The layer is designed in such a way that is it possible to chain drivers with other drivers. For example, filesystem drivers may be implemented on top of physical block devices, both represented by IO devices with corresponding drivers. In such a case, the file-system "binding" with the block device may be deferred until the file-system device is initialised.

The abstraction currently depends on structures being statically allocated by the drivers and callers, as the system does not yet provide a means of dynamically allocating memory. This may also have the affect of limiting the amount of open resources per driver.

6.24 Measured Boot Platform Interface

Enabling the MEASURED_BOOT flag adds extra platform requirements. Please refer to *Measured Boot Design* for more details.

6.25 Live Firmware Activation Interface

6.25.1 Function : plat_lfa_get_components()

```
Argument : plat_lfa_component_info_t **
Return   : int
```

This platform API provides the list of LFA components available for activation. It populates a pointer to an array of `plat_lfa_component_info_t` structures, which contain information about each component (like UUID, ID, etc.). It returns 0 on success, or a standard error code on failure.

6.25.2 Function : is_plat_lfa_activation_pending()

```
Argument : uint32_t
Return   : bool
```

This platform API checks if the specified LFA component, identified by its `lfa_component_id`, is available for activation. It returns true if available, otherwise false.

6.25.3 Function : plat_lfa_cancel()

```
Argument : uint32_t
Return   : int
```

This platform API allows the platform to cancel an ongoing update or activation process for the specified `lfa_component_id`. It returns 0 on success or a standard error code on failure.

6.25.4 Function : plat_lfa_load_auth_image()

```
Argument : uint32_t
Return   : int
```

The platform uses this API to load, authenticate and measure the component specified by `lfa_component_id`. It should return 0 on success or appropriate error codes for load/authentication failures.

NOTE: The error code -EAGAIN is not treated as an error, it means the operation is incomplete and the function should be called again by the caller.

6.25.5 Function : plat_lfa_notify_activate()

```
Argument : uint32_t
Return   : int
```

This API is invoked by the platform to notify its security engine to initiate the required steps for component activation. The function takes the component identifier `lfa_component_id` as an argument. It should return 0 on success or appropriate negative error codes on failures.

CHAPTER

SEVEN

PLATFORM PORTS

7.1 Allwinner ARMv8 SoCs

Trusted Firmware-A (TF-A) implements the EL3 firmware layer for Allwinner SoCs with ARMv8 cores. Only BL31 is used to provide proper EL3 setup and PSCI runtime services.

7.1.1 Building TF-A

There is one build target per supported SoC:

TF-A build target	Supported SoCs
sun50i_a133	A100, A133, A133P, R818, T800
sun50i_a64	A64, H5, H64
sun50i_h6	H6
sun50i_h616	H313, H616, H700, T507
sun50i_r329	R329

To build with the default settings for a particular SoC:

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=<build target> DEBUG=1
```

So for instance to build for a board with the Allwinner A64 SoC:

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=sun50i_a64 DEBUG=1
```

Platform-specific build options

The default build options should generate a working firmware image. There are some build options that allow to fine-tune the firmware, or to disable support for optional features.

- SUNXI_PSCI_USE_NATIVE : Support direct control of the CPU cores powerdown and powerup sequence by BL31. This requires either support for a code snippet to be loaded into the ARISC SCP (A64, H5), or the power sequence control registers to be programmed directly (H6, H616). This supports only basic control, like core on/off and system off/reset. This option defaults to 1. If an active SCP supporting the SCPI protocol is detected at runtime, this control scheme will be ignored, and SCPI will be used instead, unless support has been explicitly disabled.

- `SUNXI_PSCI_USE_SCPI` : Support control of the CPU cores powerdown and powerup sequence by talking to the SCP processor via the SCPI protocol. This allows more advanced power saving techniques, like suspend to RAM. This option defaults to 1 on SoCs that feature an SCP. If no SCP firmware using the SCPI protocol is detected, the native sequence will be used instead. If both native and SCPI methods are included, SCPI will be favoured if SCP support is detected.
- `SUNXI_SETUP_REGULATORS` : On SoCs that typically ship with a PMIC power management controller, BL31 tries to set up all needed power rails, programming them to their respective voltages. That allows bootloader software like U-Boot to ignore power control via the PMIC. This setting defaults to 1. In some situations that enables too many regulators, or some regulators need to be enabled in a very specific sequence. To avoid problems with those boards, `SUNXI_SETUP_REGULATORS` can bet set to `0` on the build command line, to skip the PMIC setup entirely. Any bootloader or OS would need to setup the PMIC on its own then.

7.1.2 Installation

U-Boot's SPL acts as a loader, loading both BL31 and BL33 (typically U-Boot). Loading is done from SD card, eMMC or SPI flash, also via an USB debug interface (FEL).

After building bl31.bin, the binary must be fed to the U-Boot build system to include it in the FIT image that the SPL loader will process. bl31.bin can be either copied (or sym-linked) into U-Boot's root directory, or the environment variable BL31 must contain the binary's path. See the respective U-Boot documentation for more details.

7.1.3 Memory layout

A64, H5 and H6 SoCs

BL31 lives in SRAM A2, which is documented to be accessible from secure world only. Since this SRAM region is very limited (48 KB), we take several measures to reduce memory consumption. One of them is to confine BL31 to only 28 bits of virtual address space, which reduces the number of required page tables (each occupying 4KB of memory). The mapping we use on those SoCs is as follows:

```
0 64K         16M              1GB         1G+160M     physical address
+-+------+-+---+------+--...---+-------+----+------+----------
|B|      |S|///|      |//...///|       |////|      |
|R| SRAM |C|///| dev  |//...///| (sec) |////| BL33 |  DRAM ...
|O|      |P|///| MMIO |//...///| DRAM  |////|      |
|M|      | |///|      |//...///| (32M) |////|      |
+-+------+-+---+------+--...---+-------+----+------+----------
| |      | |   |      |       /       /   /      /
| |      | |   |      |      /       /  /      /
| |      | |   |      |     /       / /      /
| |      | |   |      |    /       //      /
| |      | |   |      |   /       /      /
```

(continues on next page)

(continued from previous page)

```
+-+------+-+---+------+--+-------+------+
|B|      |S|///|      |//|       |      |
|R| SRAM |C|///| dev  |//|  sec  | BL33 |
|O|      |P|///| MMIO |//| DRAM  |      |
|M|      | |///|      |//|       |      |
+-+------+-+---+------+--+-------+------+
0 64K         16M       160M    192M  256M             virtual address
```

A133, H616 SoCs

The H616 lacks the secure SRAM region present on the other SoCs, also lacks the “ARISC” management processor (SCP) we use.

In A133 the size of SRAM A2 section is limited to 64KB.

BL31 thus needs to run from DRAM, which prevents our compressed virtual memory map described above. Since running in DRAM also lifts the restriction of the limited SRAM size, we use the normal 1:1 mapping with 32 bits worth of virtual address space. So the virtual addresses used in BL31 match the physical addresses as presented above.

7.1.4 Trusted OS dispatcher

One can boot Trusted OS(OP-TEE OS, bl32 image) along side bl31 image on Allwinner A64.

In order to include the ‘opteed’ dispatcher in the image, pass ‘SPD=opteed’ on the command line while compiling the bl31 image and make sure the loader (SPL) loads the Trusted OS binary to the beginning of DRAM (0x40000000).

7.2 AMD Versal Gen 2

Trusted Firmware-A implements the EL3 firmware layer for AMD Versal Gen 2. The platform only uses the runtime part of TF-A as AMD Versal Gen 2 already has a BootROM (BL1) and PMC FW (BL2).

BL31 is TF-A. BL32 is an optional Secure Payload. BL33 is the non-secure world software (U-Boot, Linux etc).

To build: `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal2 bl31 ` ``

To build TF-A for JTAG DCC console: `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal2 CONSOLE=dcc bl31 ` ``

To build TF-A with Errata management interface `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal2 bl31 ERRATA_ABI_SUPPORT=1 ` ``

To build TF-A with IPI CRC check: `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal2 bl31 IPI_CRC_CHECK=1 ` ``

7.2.1 AMD Versal Gen 2 platform specific build options

- *MEM_BASE*: Specifies the base address of the bl31 binary.
- *MEM_SIZE*: Specifies the size of the memory region of the bl31 binary.
- *BL32_MEM_BASE*: Specifies the base address of the bl32 binary.
- *BL32_MEM_SIZE*: Specifies the size of the memory region of the bl32 binary.
- *CONSOLE*: Select the console driver. Options: - *pl011*, *pl011_0*: ARM pl011 UART 0 (default) - *pl011_1* : ARM pl011 UART 1 - *dcc* : JTAG Debug Communication Channel(DCC)
- *NS_FW_HANDOFF_BASE*: When the handoff protocol using transfer lists is enabled (`TRANSFER_LIST=1`), the build system optionally accepts a build-time override for the non-secure firmware handoff base address via this argument. If not provided, the platform falls back to the default base address defined in the Versal Gen 2 platform header (`PLAT_ARM_NS_IMAGE_BASE` minus 10 MB).

7.2.2 Configurable Stack Size

The stack size in TF-A for the Versal platform is configurable. The custom package can define the desired stack size as per the requirement in the makefile as follows:

```
PLATFORM_STACK_SIZE := <value>

$(eval $(call add_define,PLATFORM_STACK_SIZE))
```

7.2.3 CUSTOM SIP Service Support

- Dedicated SMC FID `SOC_SIP_SVC_CUSTOM(0x82002000)` (32-bit) / `(0xC2002000)` (64-bit) is used by a custom package for providing CUSTOM SIP service.
- By default, the platform provides a bare minimum definition for `custom_smc_handler` in this service.
- To use this service, the custom package should implement its own SMC handler named `custom_smc_handler`. Once the custom package is included in the TF-A build, its definition of `custom_smc_handler` is enabled.

7.2.4 Configurable APU IPI ID

The APU IPI ID in TF-A for the Versal Gen 2 platform is configurable as per the design.

To build PLAT_IPI_ID_APU:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=versal2 RESET_TO_BL31=1 bl31 PLAT_
↪IPI_ID_APU=<value>

$(eval $(call add_define,PLAT_IPI_ID_APU))
```

7.2.5 Custom Package Makefile Fragment Inclusion in TF-A Build

- Custom package is not directly part of the TF-A source.
- <CUSTOM_PKG_PATH> is the location where the user clones a custom package locally.
- The custom package must implement a makefile fragment named `custom_pkg.mk` so it can be included in the TF-A build.
- `custom_pkg.mk` should specify all the rules to include custom package specific header files, dependent libraries, and source files that are required to be part of the TF-A build.
- When <CUSTOM_PKG_PATH> is specified in the TF-A build command, `custom_pkg.mk` is included from <CUSTOM_PKG_PATH>.
- Example TF-A build command:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=versal2 RESET_TO_BL31=1 bl31 CUSTOM_
↪PKG_PATH=<...>
```

7.2.6 Reference DEN0028E SMC calling convention

7.2.7 Allocated subranges of Function Identifier to SIP services

SMC Function	Identifier Service type
0xC2000000-0xC200FFFF	Fast SMC64 SiP Service Calls as per SMCCC Section 6.1

7.2.8 IPI SMC call ranges

SMC Function Identifier	Service type
0xc2001000-0xc2001FFF	Fast SMC64 SiP Service call range used for AMD-Xilinx IPI

7.2.9 PM SMC call ranges

SMC Function Identifier	Service type
0xc2000000-0xc2000FFF	Fast SMC64 SiP Service call range used for AMD-Xilinx Platform Management

7.2.10 SMC function IDs for SiP Service queries

Service	Call UID	Revision
SiP Service	0x8200_FF01	0x8200_FF03

Call UID Query – Returns a unique identifier of the service provider.

Revision Query – Returns revision details of the service implementor.

7.3 Arm Development Platforms

7.3.1 Arm Juno Development Platform

Platform-specific build options

- `JUNO_TZMP1` : Boolean option to configure Juno to be used for TrustZone Media Protection (TZ-MP1). Default value of this flag is 0.

Running software on Juno

This version of TF-A has been tested on variants r0, r1 and r2 of Juno.

To run TF-A on Juno, you need to first prepare an SD card with Juno software stack that includes TF-A. This version of TF-A is tested with pre-built Linaro release software stack version 20.01. You can alternatively build the software stack yourself by following the Juno platform software user guide. Once you prepare the software stack on an SD card, you can replace the `bl1.bin` and `fip.bin` binaries in the `SOFTWARE/` directory with custom built TF-A binaries.

Preparing TF-A images

This section provides Juno and FVP specific instructions to build Trusted Firmware, obtain the additional required firmware, and pack it all together in a single FIP binary. It assumes that a Linaro release software stack has been installed.

Note: Pre-built binaries for AArch32 are available from Linaro Release 16.12 onwards. Before that release, pre-built binaries are only available for AArch64.

Warning: Follow the full instructions for one platform before switching to a different one. Mixing instructions for different platforms may result in corrupted binaries.

Warning: The uboot image downloaded by the Linaro workspace script does not always match the uboot image packaged as BL33 in the corresponding fip file. It is recommended to use the version that is packaged in the fip file using the instructions below.

Note: For the FVP, the kernel FDT is packaged in FIP during build and loaded by the firmware at runtime.

1. Clean the working directory

```
make realclean
```

2. Obtain SCP binaries (Juno)

   This version of TF-A is tested with SCP version 2.14.0 on Juno. You can download pre-built SCP binaries (`scp_bl1.bin` and `scp_bl2.bin`) from TF-A downloads page. Alternatively, you can build the binaries from source.

3. Obtain BL33 (all platforms)

   Use the fiptool to extract the BL33 image from the FIP package included in the Linaro release:

```
# Build the fiptool
make [DEBUG=1] [V=1] fiptool

# Unpack firmware images from Linaro FIP
./tools/fiptool/fiptool unpack <path-to-linaro-release>/[SOFTWARE]/fip.
↪bin
```

   The unpack operation will result in a set of binary images extracted to the current working directory. BL33 corresponds to `nt-fw.bin`.

   Note: The fiptool will complain if the images to be unpacked already exist in the current directory. If that is the case, either delete those files or use the `--force` option to overwrite.

   Note: For AArch32, the instructions below assume that nt-fw.bin is a normal world boot loader that supports AArch32.

4. Build TF-A images and create a new FIP for FVP

```
# AArch64
make PLAT=fvp BL33=nt-fw.bin all fip

# AArch32
make PLAT=fvp ARCH=aarch32 AARCH32_SP=sp_min BL33=nt-fw.bin all fip
```

5. Build TF-A images and create a new FIP for Juno

For AArch64:

Building for AArch64 on Juno simply requires the addition of `SCP_BL2` as a build parameter.

```
make PLAT=juno BL33=nt-fw.bin SCP_BL2=scp_bl2.bin all fip
```

For AArch32:

Hardware restrictions on Juno prevent cold reset into AArch32 execution mode, therefore BL1 and BL2 must be compiled for AArch64, and BL32 is compiled separately for AArch32.

- Before building BL32, the environment variable `CROSS_COMPILE` must point to the AArch32 Linaro cross compiler.

  ```
  export CROSS_COMPILE=<path-to-aarch32-gcc>/bin/arm-linux-gnueabihf-
  ```

- Build BL32 in AArch32.

  ```
  make ARCH=aarch32 PLAT=juno AARCH32_SP=sp_min \
  RESET_TO_SP_MIN=1 JUNO_AARCH32_EL3_RUNTIME=1 bl32
  ```

- Save `bl32.bin` to a temporary location and clean the build products.

  ```
  cp <path-to-build>/bl32.bin <path-to-temporary>
  make realclean
  ```

- Before building BL1 and BL2, the environment variable `CROSS_COMPILE` must point to the AArch64 Linaro cross compiler.

  ```
  export CROSS_COMPILE=<path-to-aarch64-gcc>/bin/aarch64-none-elf-
  ```

- The following parameters should be used to build BL1 and BL2 in AArch64 and point to the BL32 file.

  ```
  make ARCH=aarch64 PLAT=juno JUNO_AARCH32_EL3_RUNTIME=1 \
  BL33=nt-fw.bin SCP_BL2=scp_bl2.bin \
  BL32=<path-to-temporary>/bl32.bin all fip
  ```

The resulting BL1 and FIP images may be found in:

```
# Juno
./build/juno/release/bl1.bin
./build/juno/release/fip.bin

# FVP
./build/fvp/release/bl1.bin
./build/fvp/release/fip.bin
```

After building TF-A, the files `bl1.bin`, `fip.bin` and `scp_bl1.bin` need to be copied to the `SOFTWARE/` directory on the Juno SD card.

Booting Firmware Update images

The new images must be programmed in flash memory by adding an entry in the `SITE1/HBI0262x/images.txt` configuration file on the Juno SD card (where `x` depends on the revision of the Juno board). Refer to the Juno Getting Started Guide, section 2.3 "Flash memory programming" for more information. User should ensure these do not overlap with any other entries in the file.

```
NOR10UPDATE: AUTO                       ;Image Update:NONE/AUTO/FORCE
NOR10ADDRESS: 0x00400000                ;Image Flash Address [ns_bl2u_base_
↪address]
NOR10FILE: \SOFTWARE\fwu_fip.bin        ;Image File Name
NOR10LOAD: 00000000                     ;Image Load Address
NOR10ENTRY: 00000000                    ;Image Entry Point

NOR11UPDATE: AUTO                       ;Image Update:NONE/AUTO/FORCE
NOR11ADDRESS: 0x03EB8000                ;Image Flash Address [ns_bl1u_base_
↪address]
NOR11FILE: \SOFTWARE\ns_bl1u.bin        ;Image File Name
NOR11LOAD: 00000000                     ;Image Load Address
```

The address ns_bl1u_base_address is the value of NS_BL1U_BASE - 0x8000000. In the same way, the address ns_bl2u_base_address is the value of NS_BL2U_BASE - 0x8000000.

Booting an EL3 payload

If the EL3 payload is able to execute in place, it may be programmed in flash memory by adding an entry in the `SITE1/HBI0262x/images.txt` configuration file on the Juno SD card (where `x` depends on the revision of the Juno board). Refer to the Juno Getting Started Guide, section 2.3 "Flash memory programming" for more information.

Alternatively, the same DS-5 command mentioned in the FVP section above can be used to load the EL3 payload's ELF file over JTAG on Juno.

For more information on EL3 payloads in general, see *Booting an EL3 payload*.

Booting a preloaded kernel image

The Trusted Firmware must be compiled in a similar way as for FVP explained above. The process to load binaries to memory is the one explained in *plat_juno_booting_el3_payload*.

Testing System Suspend

The SYSTEM SUSPEND is a PSCI API which can be used to implement system suspend to RAM. For more details refer to section 5.16 of PSCI. To test system suspend on Juno, at the linux shell prompt, issue the following command:

```
echo +10 > /sys/class/rtc/rtc0/wakealarm
echo -n mem > /sys/power/state
```

The Juno board should suspend to RAM and then wakeup after 10 seconds due to wakeup interrupt from RTC.

Additional Resources

Please visit the Arm Platforms Portal to get support and obtain any other Juno software information. Please also refer to the Juno Getting Started Guide to get more detailed information about the Juno Arm development platform and how to configure it.

7.3.2 Arm Fixed Virtual Platforms (FVP)

Arm *FVP*s are complete simulations of an Arm system, including processor, memory and peripherals. They enable software development without the need for real hardware.

There exists many types of FVPs. This page provides details on how to build and run TF-A on some of these FVPs.

Please also refer to the TF-A CI scripts under the model/ directory for an exhaustive list of *FVP*s which TF-A is regularly tested on as part of our continuous integration strategy.

Fixed Virtual Platform (FVP) Support

An *FVP* provides a complete simulation of an Arm system. This is a generic term used for all kinds of vastly different and incompatible systems. One category of these systems are the `FVP_Base` family of FVPs. These are entirely virtual platforms, largely used for early feature development. They offer a large degrees of customisation but share a lot of similarities. The `fvp` platform in TF-A supports these platforms only. Despite the generic name, other FVPs (eg FVP_TC4) have their own dedicated TF-A platforms and will not work with this one.

Please refer to each FVP's documentation for a detailed description of the model parameter options.

The latest version of the AArch64 build of TF-A has been tested on the following Arm FVPs without shifted affinities, and that do not support threaded CPU cores (64-bit host machine only).

Note: The FVP models used are version 11.31 Build 28.

- FVP_Base_AEMvA-AEMvA
- FVP_Base_RevC-2xAEMvA
- FVP_Base_Cortex-A32
- FVP_Base_Cortex-A35
- FVP_Base_Cortex-A53
- FVP_Base_Cortex-A55
- FVP_Base_Cortex-A57x1-A53x1
- FVP_Base_Cortex-A57x2-A53x4
- FVP_Base_Cortex-A57
- FVP_Base_Cortex-A57x4-A53x4
- FVP_Base_Cortex-A65
- FVP_Base_Cortex-A65AE
- FVP_Base_Cortex-A72
- FVP_Base_Cortex-A73
- FVP_Base_Cortex-A73x4-A53x4
- FVP_Base_Cortex-A75
- FVP_Base_Cortex-A76
- FVP_Base_Cortex-A76AE
- FVP_Base_Cortex-A77
- FVP_Base_Cortex-A78
- FVP_Base_Cortex-A78AE
- FVP_Base_Cortex-A78C
- FVP_Base_Cortex-A710
- FVP_Base_Cortex-X2
- FVP_Base_Cortex-X4
- FVP_Base_Cortex-X925
- FVP_Base_Neoverse-E1
- FVP_Base_Neoverse-N1

- FVP_Base_Neoverse-N2
- FVP_Base_Neoverse-V1

The latest version of the AArch32 build of TF-A has been tested on the following Arm FVPs without shifted affinities, and that do not support threaded CPU cores (64-bit host machine only).

- FVP_Base_AEMvA
- FVP_Base_AEMvA-AEMvA
- FVP_Base_Cortex-A32

Note: The FVP_Base_RevC-2xAEMv8A FVP only supports shifted affinities, which is not compatible with legacy GIC configurations. Therefore this FVP does not support these legacy GIC configurations.

The *Foundation* and *Base* FVPs can be downloaded free of charge. See the Arm's website for download options of all FVPs.

Note: The build numbers quoted above are those reported by launching the FVP with the --version parameter.

Note: Linaro provides a ramdisk image in prebuilt FVP configurations and full file systems that can be downloaded separately. To run an FVP with a virtio file system image an additional FVP configuration option -C bp.virtioblockdevice.image_path="<path-to>/<file-system-image> can be used.

Note: The software will not work on Version 1.0 of the Foundation FVP. The commands below would report an unhandled argument error in this case.

Note: FVPs can be launched with --cadi-server option such that a CADI-compliant debugger (for example, Arm DS-5) can connect to and control its execution.

Warning: Since FVP model Version 11.0 Build 11.0.34 and Version 8.5 Build 0.8.5202 the internal synchronisation timings changed compared to older versions of the models. The models can be launched with -Q 100 option if they are required to match the run time characteristics of the older versions.

All the above platforms have been tested with Linaro Release 20.01.

Arm FVP Platform Specific Build Options

- `FVP_CLUSTER_COUNT` : Configures the cluster count to be used to build the topology tree within TF-A. By default TF-A is configured for dual cluster topology and this option can be used to override the default value.
- `FVP_INTERCONNECT_DRIVER`: Selects the interconnect driver to be built. The default interconnect driver depends on the value of `FVP_CLUSTER_COUNT` as explained in the options below:
  - `FVP_CCI` : The CCI driver is selected. This is the default if 0 < `FVP_CLUSTER_COUNT` <= 2.
  - `FVP_CCN` : The CCN driver is selected. This is the default if `FVP_CLUSTER_COUNT` > 2.
- `FVP_MAX_CPUS_PER_CLUSTER`: Sets the maximum number of CPUs implemented in a single cluster. This option defaults to 4.
- `FVP_MAX_PE_PER_CPU`: Sets the maximum number of PEs implemented on any CPU in the system. This option defaults to 1. Note that the build option `ARM_PLAT_MT` doesn't have any effect on FVP platforms.
- `FVP_USE_GIC_DRIVER` : Selects the GIC driver to be built. Options:
  - `FVP_GICV2` : The GICv2 only driver is selected
  - `FVP_GICV3` : The GICv3 only driver is selected (default option)
  - `FVP_GICV5` : The GICv5 only driver is selected
- `FVP_HW_CONFIG_DTS` : Specify the path to the DTS file to be compiled to DTB and packaged in FIP as the HW_CONFIG. See *Firmware Design* for details on HW_CONFIG. By default, this is initialized to a sensible DTS file in `fdts/` folder depending on other build options. But some cases, like shifted affinity format for MPIDR, cannot be detected at build time and this option is needed to specify the appropriate DTS file.
- `FVP_HW_CONFIG` : Specify the path to the HW_CONFIG blob to be packaged in FIP. See *Firmware Design* for details on HW_CONFIG. This option is similar to the `FVP_HW_CONFIG_DTS` option, but it directly specifies the HW_CONFIG blob instead of the DTS file. This option is useful to override the default HW_CONFIG selected by the build system.
- `FVP_GICR_REGION_PROTECTION`: Mark the redistributor pages of inactive/fused CPU cores as read-only. The default value of this option is `0`, which means the redistributor pages of all CPU cores are marked as read and write.
- `INITRD_SIZE` : Enable the insertion of initrd properties to the device tree blob at build time. Takes an initrd size value in hex format
- `INITRD_PATH` : Enable the insertion of initrd properties to the device tree blob at build time. Takes a path to an initrd file. Can be used as an alternative to `INITRD_SIZE`. `INITRD_SIZE` takes precedence over `INITRD_PATH` if both values are provided.
- `INITRD_BASE` : Provide the preloaded initrd base address in memory (hex format).

Running on the AEMv8 Base FVP

AArch64 with reset to BL1 entrypoint

The following `FVP_Base_RevC-2xAEMv8A` parameters should be used to boot Linux with 8 CPUs using the AArch64 build of TF-A.

```
<path-to>/FVP_Base_RevC-2xAEMv8A                          \
-C pctl.startup=0.0.0.0                                   \
-C bp.secure_memory=1                                     \
-C bp.tzc_400.diagnostics=1                               \
-C cluster0.NUM_CORES=4                                   \
-C cluster1.NUM_CORES=4                                   \
-C cache_state_modelled=1                                 \
-C bp.secureflashloader.fname="<path-to>/<bl1-binary>"    \
-C bp.flashloader0.fname="<path-to>/<FIP-binary>"         \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000 \
--data cluster0.cpu0="<path-to>/<ramdisk>"@0x84000000
```

Note: The `FVP_Base_RevC-2xAEMv8A` has shifted affinities and requires a specific DTS for all the CPUs to be loaded.

AArch32 with reset to BL1 entrypoint

The following `FVP_Base_AEMv8A-AEMv8A` parameters should be used to boot Linux with 8 CPUs using the AArch32 build of TF-A.

```
<path-to>/FVP_Base_AEMv8A-AEMv8A                          \
-C pctl.startup=0.0.0.0                                   \
-C bp.secure_memory=1                                     \
-C bp.tzc_400.diagnostics=1                               \
-C cluster0.NUM_CORES=4                                   \
-C cluster1.NUM_CORES=4                                   \
-C cache_state_modelled=1                                 \
-C cluster0.cpu0.CONFIG64=0                               \
-C cluster0.cpu1.CONFIG64=0                               \
-C cluster0.cpu2.CONFIG64=0                               \
-C cluster0.cpu3.CONFIG64=0                               \
-C cluster1.cpu0.CONFIG64=0                               \
-C cluster1.cpu1.CONFIG64=0                               \
-C cluster1.cpu2.CONFIG64=0                               \
-C cluster1.cpu3.CONFIG64=0                               \
-C bp.secureflashloader.fname="<path-to>/<bl1-binary>"    \
-C bp.flashloader0.fname="<path-to>/<FIP-binary>"         \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000 \
--data cluster0.cpu0="<path-to>/<ramdisk>"@0x84000000
```

AArch64 with reset to BL31 entrypoint

The following `FVP_Base_RevC-2xAEMv8A` parameters should be used to boot Linux with 8 CPUs using the AArch64 build of TF-A.

```
<path-to>/FVP_Base_RevC-2xAEMv8A                                       \
-C pctl.startup=0.0.0.0                                                \
-C bp.secure_memory=1                                                  \
-C bp.tzc_400.diagnostics=1                                            \
-C cluster0.NUM_CORES=4                                                \
-C cluster1.NUM_CORES=4                                                \
-C cache_state_modelled=1                                              \
-C cluster0.cpu0.RVBAR=0x04010000                                      \
-C cluster0.cpu1.RVBAR=0x04010000                                      \
-C cluster0.cpu2.RVBAR=0x04010000                                      \
-C cluster0.cpu3.RVBAR=0x04010000                                      \
-C cluster1.cpu0.RVBAR=0x04010000                                      \
-C cluster1.cpu1.RVBAR=0x04010000                                      \
-C cluster1.cpu2.RVBAR=0x04010000                                      \
-C cluster1.cpu3.RVBAR=0x04010000                                      \
--data cluster0.cpu0="<path-to>/<bl31-binary>"@0x04010000              \
--data cluster0.cpu0="<path-to>/<bl32-binary>"@0xff000000              \
--data cluster0.cpu0="<path-to>/<bl33-binary>"@0x88000000              \
--data cluster0.cpu0="<path-to>/<fdt>"@0x82000000                      \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000            \
--data cluster0.cpu0="<path-to>/<ramdisk>"@0x84000000
```

Notes:

- Position Independent Executable (PIE) support is enabled in this config allowing BL31 to be loaded at any valid address for execution.
- Since a FIP is not loaded when using BL31 as reset entrypoint, the `--data="<path-to><bl31|bl32|bl33-binary>"@<base-address-of-binary>` parameter is needed to load the individual bootloader images in memory. BL32 image is only needed if BL31 has been built to expect a Secure-EL1 Payload. For the same reason, the FDT needs to be compiled from the DT source and loaded via the `--data cluster0.cpu0="<path-to>/<fdt>"@0x82000000` parameter.
- The `FVP_Base_RevC-2xAEMv8A` has shifted affinities and requires a specific DTS for all the CPUs to be loaded.
- The `-C cluster<X>.cpu<Y>.RVBAR=@<base-address-of-bl31>` parameter, where X and Y are the cluster and CPU numbers respectively, is used to set the reset vector for each core.
- Changing the default value of `ARM_TSP_RAM_LOCATION` will also require changing the value of `--data="<path-to><bl32-binary>"@<base-address-of-bl32>` to the new value of `BL32_BASE`.

AArch32 with reset to SP_MIN entrypoint

The following `FVP_Base_AEMv8A-AEMv8A` parameters should be used to boot Linux with 8 CPUs using the AArch32 build of TF-A.

```
<path-to>/FVP_Base_AEMv8A-AEMv8A                                \
-C pctl.startup=0.0.0.0                                         \
-C bp.secure_memory=1                                           \
-C bp.tzc_400.diagnostics=1                                     \
-C cluster0.NUM_CORES=4                                         \
-C cluster1.NUM_CORES=4                                         \
-C cache_state_modelled=1                                       \
-C cluster0.cpu0.CONFIG64=0                                     \
-C cluster0.cpu1.CONFIG64=0                                     \
-C cluster0.cpu2.CONFIG64=0                                     \
-C cluster0.cpu3.CONFIG64=0                                     \
-C cluster1.cpu0.CONFIG64=0                                     \
-C cluster1.cpu1.CONFIG64=0                                     \
-C cluster1.cpu2.CONFIG64=0                                     \
-C cluster1.cpu3.CONFIG64=0                                     \
-C cluster0.cpu0.RVBAR=0x04002000                               \
-C cluster0.cpu1.RVBAR=0x04002000                               \
-C cluster0.cpu2.RVBAR=0x04002000                               \
-C cluster0.cpu3.RVBAR=0x04002000                               \
-C cluster1.cpu0.RVBAR=0x04002000                               \
-C cluster1.cpu1.RVBAR=0x04002000                               \
-C cluster1.cpu2.RVBAR=0x04002000                               \
-C cluster1.cpu3.RVBAR=0x04002000                               \
--data cluster0.cpu0="<path-to>/<bl32-binary>"@0x04002000       \
--data cluster0.cpu0="<path-to>/<bl33-binary>"@0x88000000       \
--data cluster0.cpu0="<path-to>/<fdt>"@0x82000000               \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000     \
--data cluster0.cpu0="<path-to>/<ramdisk>"@0x84000000
```

Note: Position Independent Executable (PIE) support is enabled in this config allowing SP_MIN to be loaded at any valid address for execution.

Running on the Cortex-A57-A53 Base FVP

With reset to BL1 entrypoint

The following `FVP_Base_Cortex-A57x4-A53x4` model parameters should be used to boot Linux with 8 CPUs using the AArch64 build of TF-A.

```
<path-to>/FVP_Base_Cortex-A57x4-A53x4                           \
-C pctl.startup=0.0.0.0                                         \
-C bp.secure_memory=1                                           \
-C bp.tzc_400.diagnostics=1                                     \
-C cache_state_modelled=1                                       \
-C bp.secureflashloader.fname="<path-to>/<bl1-binary>"          \
-C bp.flashloader0.fname="<path-to>/<FIP-binary>"               \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000 \
--data cluster0.cpu0="<path-to>/<ramdisk>"@0x84000000
```

With reset to BL31 entrypoint

The following `FVP_Base_Cortex-A57x4-A53x4` model parameters should be used to boot Linux with 8 CPUs using the AArch64 build of TF-A.

```
<path-to>/FVP_Base_Cortex-A57x4-A53x4                            \
-C pctl.startup=0.0.0.0                                          \
-C bp.secure_memory=1                                            \
-C bp.tzc_400.diagnostics=1                                      \
-C cache_state_modelled=1                                        \
-C cluster0.cpu0.RVBARADDR=0x04010000                            \
-C cluster0.cpu1.RVBARADDR=0x04010000                            \
-C cluster0.cpu2.RVBARADDR=0x04010000                            \
-C cluster0.cpu3.RVBARADDR=0x04010000                            \
-C cluster1.cpu0.RVBARADDR=0x04010000                            \
-C cluster1.cpu1.RVBARADDR=0x04010000                            \
-C cluster1.cpu2.RVBARADDR=0x04010000                            \
-C cluster1.cpu3.RVBARADDR=0x04010000                            \
--data cluster0.cpu0="<path-to>/<bl31-binary>"@0x04010000    \
--data cluster0.cpu0="<path-to>/<bl32-binary>"@0xff000000    \
--data cluster0.cpu0="<path-to>/<bl33-binary>"@0x88000000    \
--data cluster0.cpu0="<path-to>/<fdt>"@0x82000000            \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000  \
--data cluster0.cpu0="<path-to>/<ramdisk>"@0x84000000
```


Running on the Cortex-A32 Base FVP (AArch32)

With reset to BL1 entrypoint

The following `FVP_Base_Cortex-A32x4` model parameters should be used to boot Linux with 4 CPUs using the AArch32 build of TF-A.

```
<path-to>/FVP_Base_Cortex-A32x4                                 \
-C pctl.startup=0.0.0.0                                         \
-C bp.secure_memory=1                                           \
```

(continues on next page)

(continued from previous page)

```
-C bp.tzc_400.diagnostics=1                                  \
-C cache_state_modelled=1                                    \
-C bp.secureflashloader.fname="<path-to>/<bl1-binary>"       \
-C bp.flashloader0.fname="<path-to>/<FIP-binary>"            \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000 \
--data cluster0.cpu0="<path-to>/<ramdisk>"@0x84000000
```

With reset to SP_MIN entrypoint

The following `FVP_Base_Cortex-A32x4` model parameters should be used to boot Linux with 4 CPUs using the AArch32 build of TF-A.

```
<path-to>/FVP_Base_Cortex-A32x4                              \
-C pctl.startup=0.0.0.0                                      \
-C bp.secure_memory=1                                        \
-C bp.tzc_400.diagnostics=1                                  \
-C cache_state_modelled=1                                    \
-C cluster0.cpu0.RVBARADDR=0x04002000                        \
-C cluster0.cpu1.RVBARADDR=0x04002000                        \
-C cluster0.cpu2.RVBARADDR=0x04002000                        \
-C cluster0.cpu3.RVBARADDR=0x04002000                        \
--data cluster0.cpu0="<path-to>/<bl32-binary>"@0x04002000   \
--data cluster0.cpu0="<path-to>/<bl33-binary>"@0x88000000   \
--data cluster0.cpu0="<path-to>/<fdt>"@0x82000000           \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000 \
--data cluster0.cpu0="<path-to>/<ramdisk>"@0x84000000
```

Booting Firmware Update images

When Firmware Update (FWU) is enabled there are at least 2 new images that have to be loaded, the Non-Secure FWU ROM (NS-BL1U), and the FWU FIP.

The additional fip images must be loaded with:

```
--data cluster0.cpu0="<path_to>/ns_bl1u.bin"@0x0beb8000     [ns_bl1u_base_
↪address]
--data cluster0.cpu0="<path_to>/fwu_fip.bin"@0x08400000     [ns_bl2u_base_
↪address]
```

The address ns_bl1u_base_address is the value of NS_BL1U_BASE. In the same way, the address ns_bl2u_base_address is the value of NS_BL2U_BASE.

Booting an EL3 payload

The EL3 payloads boot flow requires the CPU's mailbox to be cleared at reset for the secondary CPUs holding pen to work properly. Unfortunately, its reset value is undefined on the FVP platform and the FVP platform code doesn't clear it. Therefore, one must modify the way the model is normally invoked in order to clear the mailbox at start-up.

One way to do that is to create an 8-byte file containing all zero bytes using the following command:

```
dd if=/dev/zero of=mailbox.dat bs=1 count=8
```

and pre-load it into the FVP memory at the mailbox address (i.e. `0x04000000`) using the following model parameters:

```
--data cluster0.cpu0=mailbox.dat@0x04000000   [Base FVPs]
```

To provide the model with the EL3 payload image, the following methods may be used:

1. If the EL3 payload is able to execute in place, it may be programmed into flash memory. On Base Cortex and AEM FVPs, the following model parameter loads it at the base address of the NOR FLASH1 (the NOR FLASH0 is already used for the FIP):

   ```
   -C bp.flashloader1.fname="<path-to>/<el3-payload>"
   ```

2. When using the `SPIN_ON_BL1_EXIT=1` loading method, the following DS-5 command may be used to load the EL3 payload ELF image over JTAG:

   ```
   load <path-to>/el3-payload.elf
   ```

3. The EL3 payload may be pre-loaded in volatile memory using the following model parameters:

   ```
   --data cluster0.cpu0="<path-to>/el3-payload>"@address   [Base FVPs]
   ```

   The address provided to the FVP must match the `EL3_PAYLOAD_BASE` address used when building TF-A.

Booting a kernel image in BL33

TF-A can boot a Linux kernel, which uses a ramdisk as a filesystem. The required initrd properties are injected in to the device tree blob (DTB) at build time.

Kernel image packaged in fip as a BL33 image

A Linux kernel image can be packaged in the fip as a BL33 image and then booted in TF-A.

For example, the firmware can be built as:

```
make PLAT=fvp DEBUG=1                \
ARM_LINUX_KERNEL_AS_BL33             \
BL33=<path-to-kernel-binary>         \
INITRD_SIZE=0x8000000                \
all fip
```

The options `INITRD_SIZE` or `INITRD_PATH` triggers the insertion of initrd properties in to the DTB. `INITRD_BASE` is also required but a default value is set by the FVP platform.

The options available here are:

```
INITRD_BASE: Set the initrd base address in memory. Defaults to 0x90000000 in␣
↪FVP.
INITRD_SIZE: Set the initrd size in dec or hex format. Hex format must␣
↪precede with '0x'.
INITRD_PATH: Provide an initrd path for the build time to determine its exact␣
↪size.
```

Users can provide either `INITRD_SIZE` or `INITRD_PATH` to set the initrd size value. `INITRD_SIZE` takes prioty over `INITRD_PATH`.

Now the fvp binary can be run as:

```
<path-to>/FVP_Base_AEMv8A-AEMv8A                          \
-C bp.secureflashloader.fname=<path-to>/bl1.bin           \
-C bp.flashloader0.fname=<path-to>/fip.bin                \
--data cluster0.cpu0="<path-to>/<initrd.bin>"@0x90000000
```

Note: Providing a higher value for an initrd size than the actual size of the file is supported but it will trigger a non-breaking "Initramfs unpacking failed" error by the kernel at runtime. This error can be ignored because initrd's can be stacked one after another, when the kernel unpacks the first initrd it looks for another in the extra space which it won't find, hence the error.

Preloaded kernel image - Normal flow

The following example uses a simplified boot flow to boot a Linux kernel using TF-A. This can be useful if the kernel is already present in memory (like in FVP).

For example, if the kernel is loaded at `0x80080000` the firmware can be built like this:

```
make PLAT=fvp DEBUG=1                \
ARM_LINUX_KERNEL_AS_BL33=1           \
PRELOADED_BL33_BASE=0x80080000       \
```

(continues on next page)

(continued from previous page)

```
INITRD_SIZE=0x8000000             \
all fip
```

Now the FVP binary can be run with the following command:

```
<path-to>/FVP_Base_AEMv8A-AEMv8A                           \
-C bp.secureflashloader.fname=<path-to>/bl1.bin            \
-C bp.flashloader0.fname=<path-to>/fip.bin                 \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000 \
--data cluster0.cpu0="<path-to>/<initrd.bin>"@0x90000000
```

Preloaded kernel image - Reset to BL31

We can also boot a Linux kernel by jumping directly to BL31 `RESET_TO_BL31=1`. This requires preloading a DTB into memory. We can inject the initrd start and end properties into the DTB (HW_CONFIG) at build time which is then stored by TF-A in `build/fvp/<build-type>/fdts/` directory.

For example, we can build the firmware as:

```
make PLAT=fvp DEBUG=1                   \
RESET_TO_BL31=1                         \
ARM_LINUX_KERNEL_AS_BL33=1              \
PRELOADED_BL33_BASE=0x80080000          \
ARM_PRELOADED_DTB_BASE=0x87F00000       \
INITRD_BASE=0x88000000                  \
INITRD_PATH=<path-to>/initrd.bin
```

Now we can run the binary as:

```
<path-to>/FVP_Base_AEMv8A-AEMv8A                              \
-C cluster0.NUM_CORES=4                                       \
-C cluster0.cpu0.RVBAR=0x04001000                             \
-C cluster0.cpu1.RVBAR=0x04001000                             \
-C cluster0.cpu2.RVBAR=0x04001000                             \
-C cluster0.cpu3.RVBAR=0x04001000                             \
--data cluster0.cpu0="<path-to>/bl31.bin"@0x04001000          \
--data cluster0.cpu0="<path-to>/<kernel-binary>"@0x80080000   \
--data cluster0.cpu0="<path-to>/<initrd.bin>"@0x88000000      \
--data cluster0.cpu0="<path-to>/fdts/fvp-base-gicv3-psci.dtb"@87F00000
```

Obtaining the Flattened Device Trees

Depending on the FVP configuration and Linux configuration used, different FDT files are required.

Note: It is not recommended to use the FDTs built along the kernel because not all FDTs are available from there.

The dynamic configuration capability is enabled in the firmware for FVPs. This means that the firmware can authenticate and load the FDT if present in FIP. A default FDT is packaged into FIP during the build based on the build configuration. This can be overridden by using the `FVP_HW_CONFIG` or `FVP_HW_CONFIG_DTS` build options (refer to *Arm FVP Platform Specific Build Options* for details on the options).

- `fvp-base-gicv2-psci.dts`

  For use with models such as the Cortex-A57-A53 or Cortex-A32 Base FVPs without shifted affinities and with Base memory map configuration.

- `fvp-base-gicv3-psci.dts`

  For use with models such as the Cortex-A57-A53 or Cortex-A32 Base FVPs without shifted affinities and with Base memory map configuration and Linux GICv3 support.

- `fvp-base-gicv3-psci-1t.dts`

  For use with models such as the AEMv8-RevC Base FVP with shifted affinities, single threaded CPUs, Base memory map configuration and Linux GICv3 support.

- `fvp-base-gicv3-psci-dynamiq.dts`

  For use with models as the Cortex-A55-A75 Base FVPs with shifted affinities, single cluster, single threaded CPUs, Base memory map configuration and Linux GICv3 support.

GICv5 Support

GICv5 support in TF-A is currently **experimental** and provided only for early development and testing purposes. A simplified build configuration is available to allow booting the Linux kernel as a BL33 payload on the FVP platform.

Key notes:

- The support is **not production-ready** and is intended to assist with upstream kernel development and validation.
- The device tree bindings are **not finalized**
- Use this configuration at your own discretion, understanding that the design and register usage may change in future revisions.

This configuration is **temporary** and may be removed once full GICv5 support is integrated upstream.

```
make PLAT=fvp DEBUG=1                            \
CTX_INCLUDE_AARCH32_REGS=0                       \
FVP_USE_GIC_DRIVER=FVP_GICV5                     \
ARM_LINUX_KERNEL_AS_BL33=1                       \
PRELOADED_BL33_BASE=0x84000000                   \
FVP_HW_CONFIG_DTS=<PROVIDE_YOUR_OWN_DT>          \
```

Copyright (c) 2019-2025, Arm Limited. All rights reserved.

7.3.3 Arm Versatile Express

Versatile Express (VE) family development platform provides an ultra fast environment for prototyping Armv7 System-on-Chip designs. VE Fixed Virtual Platforms (FVP) are simulations of Versatile Express boards. The platform in Trusted Firmware-A has been verified with Arm Cortex-A5 and Cortex-A7 VE FVP's. This platform is tested on and only expected to work with single core models.

Boot Sequence

BL1 –> BL2 –> BL32(sp_min) –> BL33(u-boot) –> Linux kernel

How to build

Code Locations

- U-boot
- Trusted Firmware-A

Build Procedure

- Obtain arm toolchain. The software stack has been verified with linaro 6.2 arm-linux-gnueabihf. Set the CROSS_COMPILE environment variable to point to the toolchain folder.
- Fetch and build u-boot. Make the .config file using the command:

  ```
  make ARCH=arm vexpress_aemv8a_aarch32_config
  ```

  Make the u-boot binary for Cortex-A5 using the command:

  ```
  make ARCH=arm SUPPORT_ARCH_TIMER=no
  ```

  Make the u-boot binary for Cortex-A7 using the command:

```
make ARCH=arm
```

- Build TF-A:

  The make command for Cortex-A5 is:

```
make PLAT=fvp_ve ARCH=aarch32 ARM_ARCH_MAJOR=7 ARM_CORTEX_A5=yes \
AARCH32_SP=sp_min FVP_HW_CONFIG_DTS=fdts/fvp-ve-Cortex-A5x1.dts \
ARM_XLAT_TABLES_LIB_V1=1 BL33=<path_to_u-boot.bin> all fip
```

  The make command for Cortex-A7 is:

```
make PLAT=fvp_ve ARCH=aarch32 ARM_ARCH_MAJOR=7 ARM_CORTEX_A7=yes \
AARCH32_SP=sp_min FVP_HW_CONFIG_DTS=fdts/fvp-ve-Cortex-A7x1.dts  \
BL33=<path_to_u-boot.bin> all fip
```

Run Procedure

The following model parameters should be used to boot Linux using the build of Trusted Firmware-A made using the above make commands:

```
./<path_to_model> <path_to_bl1.elf> \
      -C motherboard.flashloader1.fname=<path_to_fip.bin> \
      --data cluster.cpu0=<path_to_zImage>@0x80080000  \
      --data cluster.cpu0=<path_to_ramdisk>@0x84000000
```

7.3.4 TC Total Compute Platform

Some of the features of TC platform referenced in TF-A include:

- A System Control Processor to abstract power and system management tasks away from application processors. The RAM firmware for SCP is included in the TF-A FIP and is loaded by AP BL2 from FIP in flash to SRAM for copying by SCP (SCP has access to AP SRAM).
- GICv4
- Trusted Board Boot
- SCMI
- MHUv2

The TF-A build is specified by the option *TARGET_PLATFORM* which represents the Total Compute platform number.

The TC4 platform supports the following CPUs:

- C1-Nano

- C1-Pro
- C1-Ultra

Boot Sequence

The execution begins from SCP_BL1. SCP_BL1 powers up the AP which starts executing AP_BL1 and then executes AP_BL2 which loads the SCP_BL2 from FIP to SRAM. The SCP has access to AP SRAM. The address and size of SCP_BL2 is communicated to SCP using SDS. SCP copies SCP_BL2 from SRAM to its own RAM and starts executing it. The AP then continues executing the rest of TF-A stages including BL31 runtime stage and hands off executing to Non-secure world (u-boot).

Build Procedure (TF-A only)

- Obtain Arm toolchain and set the CROSS_COMPILE environment variable to point to the toolchain folder.
- Build TF-A:

```
make PLAT=tc BL33=<path_to_uboot.bin> \
SCP_BL2=<path_to_scp_ramfw.bin> TARGET_PLATFORM=4 all fip
```

  Enable TBBR by adding the following options to the make command:

```
TRUSTED_BOARD_BOOT=1 \
GENERATE_COT=1 \
ARM_ROTPK_LOCATION=devel_rsa  \
ROT_KEY=plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem
```

7.3.5 Arm FPGA Platform

This platform supports FPGA images used internally in Arm Ltd., for testing and bringup of new cores. With that focus, peripheral support is minimal: there is no mass storage or display output, for instance. Also this port ignores any power management features of the platform. Some interconnect setup is done internally by the platform, so the TF-A code just needs to setup UART and GIC.

The FPGA platform requires to pass on a DTB for the non-secure payload (mostly Linux), so we let TF-A use information from the DTB for dynamic configuration: the UART and GIC base addresses are read from there.

As a result this port is a fairly generic BL31-only port, which can serve as a template for a minimal new (and possibly DT-based) platform port.

The aim of this port is to support as many FPGA images as possible with a single build. Image specific data must be described in the DTB or should be auto-detected at runtime.

As the number and topology layout of the CPU cores differs significantly across the various images, this is detected at runtime by BL31. The /cpus node in the DT will be added and filled accordingly, as long as it does not exist already.

Platform-specific build options

- `SUPPORT_UNKNOWN_MPID` : Boolean option to allow unknown MPIDR registers. Normally TF-A panics if it encounters a MPID value not matched to its internal list, but for new or experimental cores this creates a lot of churn. With this option, the code will fall back to some basic CPU support code (only architectural system registers, and no errata). Default value of this flag is 1.
- `PRELOADED_BL33_BASE` : Physical address of the BL33 non-secure payload. It must have been loaded into DRAM already, typically this is done by the script that also loads BL31 and the DTB. It defaults to 0x80080000, which is the traditional load address for an arm64 Linux kernel.
- `FPGA_PRELOADED_DTB_BASE` : Physical address of the flattened device tree blob (DTB). This DT will be used by TF-A for dynamic configuration, so it must describe at least the UART and a GICv3 interrupt controller. The DT gets amended by the code, to potentially add a command line and fill the CPU topology nodes. It will also be passed on to BL33, by putting its address into the x0 register before jumping to the entry point (following the Linux kernel boot protocol). It defaults to 0x80070000, which is 64KB before the BL33 load address.
- `FPGA_PRELOADED_CMD_LINE` : Physical address of the command line to put into the devicetree blob. Due to the lack of a proper bootloader, a command line can be put somewhere into memory, so that BL31 will detect it and copy it into the DTB passed on to BL33. To avoid random garbage, there needs to be a "CMD:" signature before the actual command line. Defaults to 0x1000, which is normally in the "ROM" space of the typical FPGA image (which can be written by the FPGA payload uploader, but is read-only to the CPU). The FPGA payload tool should be given a text file containing the desired command line, prefixed by the "CMD:" signature.

Building the TF-A image

```
make PLAT=arm_fgpa DEBUG=1
```

This will use the default load addresses as described above. When those addresses need to differ for a certain setup, they can be passed on the make command line:

```
make PLAT=arm_fgpa DEBUG=1 PRELOADED_BL33_BASE=0x80200000 FPGA_
↪PRELOADED_DTB_BASE=0x80180000 bl31
```

Running the TF-A image

After building TF-A, the actual TF-A code will be located in `bl31.bin` in the build directory. Additionally there is a `bl31.axf` ELF file, which contains BL31, as well as some simple ROM trampoline code (required by the Arm FPGA boot flow) and a generic DTB to support most of the FPGA images. This can be simply handed over to the FPGA payload uploader, which will take care of loading the components at their respective load addresses. In addition to this file you need at least a BL33 payload (typically a Linux kernel image), optionally a Linux initrd image file and possibly a command line:

```
fpga-run ... -m bl31.axf -l auto -m Image -l 0x80080000 -m initrd.gz␣
↪-l 0x84000000 -m cmdline.txt -l 0x1000
```

7.3.6 Arm Development Platform Build Options

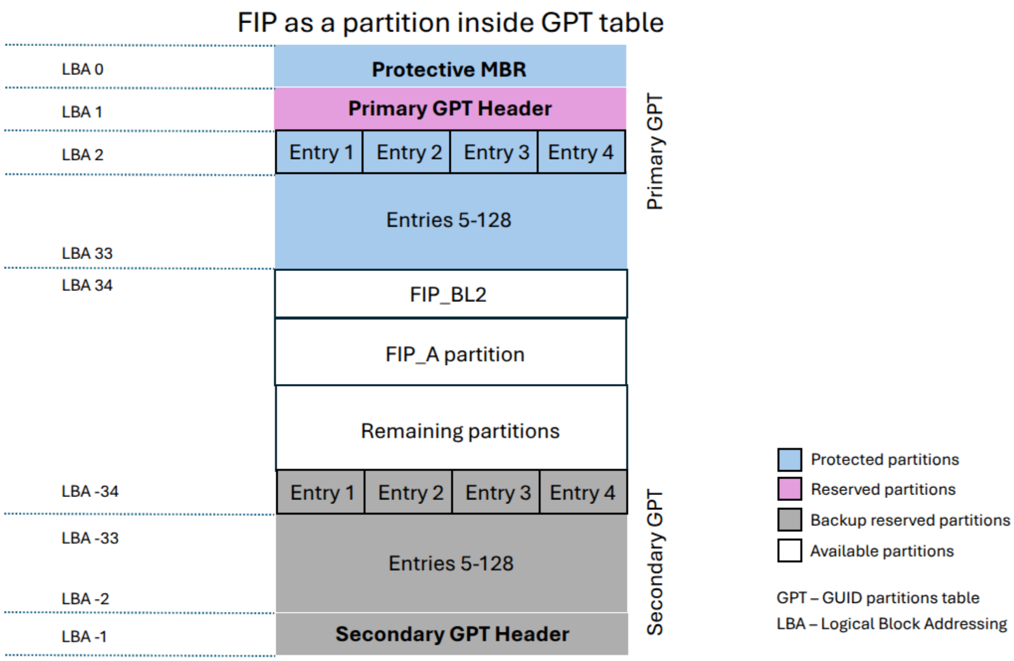
Arm Platform Build Options

- `ARM_BL31_IN_DRAM`: Boolean option to select loading of BL31 in TZC secured DRAM. By default, BL31 is in the secure SRAM. Set this flag to 1 to load BL31 in TZC secured DRAM. If TSP is present, then setting this option also sets the TSP location to DRAM and ignores the `ARM_TSP_RAM_LOCATION` build flag.
- `ARM_CONFIG_CNTACR`: boolean option to unlock access to the `CNTBase<N>` frame registers by setting the `CNTCTLBase.CNTACR<N>` register bits. The frame number `<N>` is defined by `PLAT_ARM_NSTIMER_FRAME_ID`, which should match the frame used by the Non-Secure image (normally the Linux kernel). Default is true (access to the frame is allowed).
- `ARM_FW_CONFIG_LOAD_ENABLE`: Boolean option to enable the loading of FW_CONFIG device trees from the Firmware Image Package (FIP). When enabled, BL2 is responsible for loading, parsing, and validating the FW_CONFIG device trees from the FIP. The option depends on RESET_TO_BL2.
- `ARM_DISABLE_TRUSTED_WDOG`: boolean option to disable the Trusted Watchdog. By default, Arm platforms use a watchdog to trigger a system reset in case an error is encountered during the boot process (for example, when an image could not be loaded or authenticated). The watchdog is enabled in the early platform setup hook at BL1 and disabled in the BL1 prepare exit hook. The Trusted Watchdog may be disabled at build time for testing or development purposes.
- `ARM_LINUX_KERNEL_AS_BL33`: The Linux kernel expects registers x0-x3 to have specific values at boot. This boolean option allows the Trusted Firmware to have a Linux kernel image as BL33 by preparing the registers to these values before jumping to BL33. This option defaults to 0 (disabled). When enabled (1), the address of the Linux image must be provided via the `PRELOADED_BL33_BASE` option. Additionally, either the `HW_CONFIG_BASE` or `ARM_PRELOADED_DTB_BASE` option must specify the memory location of a preloaded device tree blob (DTB). This option implies `USE_KERNEL_DT_CONVENTION`.
- `ARM_PLAT_MT`: This flag determines whether the Arm platform layer has to cater for the multi-threading `MT` bit when accessing MPIDR. When this flag is set, the functions which deal with MPIDR

assume that the MT bit in MPIDR is set and access the bit-fields in MPIDR accordingly. Default value of this flag is 0. Note that this option is not used on FVP platforms.

- **ARM_PLAT_PROVIDES_BL2_MEM_PARAMS: This flag can be overriden to 1 in the Arm**
  platform's platform.mk to indicate that the platform supplies its own bl2_mem_params_desc.c implementation. When enabled, the common implementation in plat/arm/common/ is omitted, and the platform must add its own bl2_mem_params_desc.c file to BL2_SOURCES. The default value is 0.
- ARM_RECOM_STATE_ID_ENC: The PSCI1.0 specification recommends an encoding for the construction of composite state-ID in the power-state parameter. The existing PSCI clients currently do not support this encoding of State-ID yet. Hence this flag is used to configure whether to use the recommended State-ID encoding or not. The default value of this flag is 0, in which case the platform is configured to expect NULL in the State-ID field of power-state parameter.
- ARM_ROTPK_LOCATION: used when TRUSTED_BOARD_BOOT=1. It specifies the location of the ROTPK returned by the function plat_get_rotpk_info() for Arm platforms. Depending on the selected option, the proper private key must be specified using the ROT_KEY option when building the Trusted Firmware. This private key will be used by the certificate generation tool to sign the BL2 and Trusted Key certificates. Available options for ARM_ROTPK_LOCATION are:
  - regs : return the ROTPK hash stored in the Trusted root-key storage registers.
  - devel_rsa : return a development public key hash embedded in the BL1 and BL2 binaries. This hash corresponds to the development private key plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem. The hashing algorithm is selected by HASH_ALG; sha256 is used if HASH_ALG is not specified. A different RSA key can be specified by setting ROT_KEY, there are 3k and 4k RSA keys in plat/arm/board/common/rotpk/.
  - devel_ecdsa : return a development public key hash embedded in the BL1 and BL2 binaries. This hash corresponds to the development private key plat/arm/board/common/rotpk/arm_rotprivk_ecdsa.pem unless a different key is specified with ROT_KEY, such as the 384 bit key in the same directory. he hashing algorithm is selected by HASH_ALG; sha256 is used if HASH_ALG is not specified.
  - devel_full_dev_rsa_key : return a development public key embedded in the BL1 and BL2 binaries. This key corresponds to the RSA private key plat/arm/board/common/rotpk/arm_rotprivk.pem by default, but can be changed by setting ROT_KEY, there are 3k and 4k RSA keys in plat/arm/board/common/rotpk/.
  - **devel_full_dev_ecdsa_key**
    [return a development public key embedded in] the BL1 and BL2 binaries. This key corresponds to the EC private key plat/arm/board/common/rotpk/arm_rotprivk_ecdsa.pem, unless a different ECDSA key is specified by ROT_KEY, such as the 384 bit key in the same directory.
- ARM_TSP_RAM_LOCATION: location of the TSP binary. Options:
  - tsram : Trusted SRAM (default option when TBB is not enabled)
  - tdram : Trusted DRAM (if available)
  - dram : Secure region in DRAM (default option when TBB is enabled, configured by the TrustZone controller)

- `ARM_XLAT_TABLES_LIB_V1`: boolean option to compile TF-A with version 1 of the translation tables library instead of version 2. It is set to 0 by default, which selects version 2.
- `ARM_GPT_SUPPORT`: Enable GPT parser to get the entry address and length of the various partitions present in the GPT image. This support is available only for the BL2 component, and it is disabled by default. The following diagram shows the view of the FIP partition inside the GPT image:



For a better understanding of these options, the Arm development platform memory map is explained in the *Firmware Design*.

Arm CSS Platform-Specific Build Options

- `CSS_DETECT_PRE_1_7_0_SCP`: Boolean flag to detect SCP version incompatibility. Version 1.7.0 of the SCP firmware made a non-backwards compatible change to the MTL protocol, used for AP/SCP communication. TF-A no longer supports earlier SCP versions. If this option is set to 1 then TF-A will detect if an earlier version is in use. Default is 1.
- `CSS_LOAD_SCP_IMAGES`: Boolean flag, which when set, adds SCP_BL2 and SCP_BL2U to the FIP and FWU_FIP respectively, and enables them to be loaded during boot. Default is 1.
- `CSS_USE_SCMI_SDS_DRIVER`: Boolean flag which selects SCMI/SDS drivers instead of SCPI/BOM driver for communicating with the SCP during power management operations and for SCP RAM Firmware transfer. If this option is set to 1, then SCMI/SDS drivers will be used. Default is 0.
- **`CSS_SYSTEM_GRACEFUL_RESET`: Build option to enable graceful powerdown of**

  CPU core on reset. This build option can be used on CSS platforms that require all the CPUs to execute the CPU specific power down sequence to complete a warm reboot sequence in which only the CPUs are power cycled.
- **`CSS_ERRATA_SME_POWER_DOWN`: Boolean option to disconnect the SME unit**

  (PSTATE.{ZA,SM}=0) before power down and downgrade a suspend to power down request to a normal suspend request. This is necessary when software running at lower ELs requests power down without first clearing these bits. On affected cores, the CME connected to it will reject its power down request.

Arm FVP Build Options

- `FVP_TRUSTED_SRAM_SIZE`: Size (in kilobytes) of the Trusted SRAM region to utilize when building for the FVP platform. This option defaults to 384.

Arm Juno Build Options

- `JUNO_AARCH32_EL3_RUNTIME`: This build flag enables you to execute EL3 runtime software in AArch32 mode, which is required to run AArch32 on Juno. By default this flag is set to '0'. Enabling this flag builds BL1 and BL2 in AArch64 and facilitates the loading of `SP_MIN` and BL33 as AArch32 executable images.

Arm Neoverse RD Platform Build Options

- `NRD_CHIP_COUNT`: Configures the number of chips on a Neoverse RD platform which supports multi-chip operation. If `NRD_CHIP_COUNT` is set to any valid value greater than 1, the platform code performs required configuration to support multi-chip operation.
- `NRD_PLATFORM_VARIANT`: Selects the variant of a Neoverse RD platform. A particular Neoverse RD platform may have multiple variants which may differ in core count, cluster count or other peripherals. This build option is used to select the appropriate platform variant for the build. The range of valid values is platform specific.

7.3.7 Morello Platform

Morello is an ARMv8-A platform that implements the capability architecture extension. The platform port present at site provides ARMv8-A architecture enablement.

Capability architecture specific changes have been added in upstream.

Further information on Morello Platform is available at info

Boot Sequence

The SCP initializes the RVBAR registers to point to the AP_BL1. Once RVBAR is initialized, the primary core is powered on. The primary core boots the AP_BL1. It performs minimum initialization necessary to load and authenticate the AP firmware image (the FIP image) from the AP QSPI NOR Flash Memory into the Trusted SRAM.

AP_BL1 authenticates and loads the AP_BL2 image. AP_BL2 performs additional initializations, and then authenticates and loads the AP_BL31 and AP_BL33. AP_BL2 then transfers execution control to AP_BL31, which is the EL3 runtime firmware. Execution is finally handed off to AP_BL33, which is the non-secure world (UEFI).

SCP -> AP_BL1 -> AP_BL2 -> AP_BL31 -> AP_BL33

Build Procedure (TF-A only)

- Obtain arm toolchain. Set the CROSS_COMPILE environment variable to point to the toolchain folder.
- For building morello with capability architecture specific changes use clang with morello changes which can be obtained from here. Set the CROSS_COMPILE enviroment variable to point to the toolchain folder.
- Build TF-A (without capabilities):

```
export CROSS_COMPILE=<path-to-aarch64-gcc>/bin/aarch64-none-elf-

make PLAT=morello TARGET_PLATFORM=<fvp or soc> all
```

- Build TF-A (with capabilities):

```
export CROSS_COMPILE=<path-to-clang>/bin/llvm-

make PLAT=morello TARGET_PLATFORM=<fvp or soc> ENABLE_FEAT_MORELLO=<1 or␣
↪2> all
```


7.3.8 Corstone1000 Platform

Some of the features of the Corstone1000 platform referenced in TF-A include:

- Cortex-A35 application processor (64-bit mode)
- Secure Enclave
- GIC-400
- Trusted Board Boot

Boot Sequence

The board boot relies on CoT (chain of trust). The trusted-firmware-a BL2 is extracted from the FIP and verified by the Secure Enclave processor. BL2 verification relies on the signature area at the beginning of the BL2 image. This area is needed by the SecureEnclave bootloader.

Then, the application processor is released from reset and starts by executing BL2.

BL2 performs the actions described in the trusted-firmware-a TBB design document.

Build Procedure (TF-A only)

- Obtain AArch64 ELF bare-metal target toolchain. Set the CROSS_COMPILE environment variable to point to the toolchain folder.
- Build TF-A:

```
make LD=aarch64-none-elf-ld \
CC=aarch64-none-elf-gcc \
V=1 \
BUILD_BASE=<path to the build folder> \
PLAT=corstone1000 \
SPD=spmd \
SPMD_SPM_AT_SEL2=0 \
DEBUG=1 \
OPENSSL_DIR=<path to openssl usr folder> \
RUNTIME_SYSROOT=<path to the sysroot> \
ARCH=aarch64 \
TARGET_PLATFORM=<fpga or fvp> \
ENABLE_PIE=1 \
RESET_TO_BL2=1 \
CREATE_KEYS=1 \
GENERATE_COT=1 \
TRUSTED_BOARD_BOOT=1 \
COT=tbbr \
ARM_ROTPK_LOCATION=devel_rsa \
ROT_KEY=plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem \
BL32=<path to optee binary> \
BL33=<path to u-boot binary> \
bl2
```

7.3.9 Automotive RD Platforms

RD-1 AE (Kronos) Platform

Some of the features of the RD-1 AE platform referenced in TF-A include:

- Neoverse-V3AE, Arm9.2-A application processor (64-bit mode)
- A GICv4-compatible GIC-720AE

Further information on RD1-AE is available at rd1ae

Boot Sequence

The boot process starts from RSE (Runtime Security Engine) that loads the Application Processor (AP) BL2 image and signals the System Control Processor (SCP) to power up the AP. The AP then runs AP BL2

The primary compute boot flow follows the following steps:

1. AP BL2:
   - Performs the actions described in the Trusted Board Boot (TBB) document.
   - Copies the AP BL31 image from Secure Flash to Secure SRAM
   - Copies the AP BL32 (OP-TEE) image from Secure Flash to Secure DRAM
   - Copies the AP BL33 (U-Boot) image from Secure Flash to Normal DRAM
   - Transfers the execution to AP BL31
2. AP BL31:
   - Initializes Trusted Firmware-A Services
   - Transfers the execution to AP BL32 and then transfers the execution to AP BL33
   - During runtime, acts as the Secure Monitor, handling SMC calls, and context switching between secure and non-secure worlds.
3. AP BL32:
   - Initializes OP-TEE environment
   - Initializes Secure Partitions
   - Transfers the execution back to AP BL31
   - During runtime, it facilitates secure communication between the normal world environment (e.g. Linux) and the Trusted Execution Environment.

Build Procedure (TF-A only)

- Obtain Arm toolchain and set the CROSS_COMPILE environment variable to point to the toolchain folder.
- Build TF-A:

```
make \
PLAT=rd1ae \
ARCH=aarch64 \
CREATE_KEYS=1 \
GENERATE_COT=1 \
TRUSTED_BOARD_BOOT=1 \
COT=tbbr \
ARM_ROTPK_LOCATION=devel_rsa \
ROT_KEY=plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem \
BL32=<path to optee binary> \
SPD=spmd \
SPMD_SPM_AT_SEL2=0 \
BL33=<path to u-boot binary> \
```


RD-Aspen (Zena CSS) Platform

The RD-Aspen platform, as referenced in TF-A, includes the following features:

- Primary Compute with four processor clusters, each containing:
  - Four Cortex-A720AE cores (Armv9.2-A application processor, 64-bit mode)
  - A DynamIQ Shared Unit (DSU-120AE)
- A GIC-720AE, which is GICv4-compatible and supports GICv3 mode as well.

Further information on RD-Aspen is available at Zena CSS

Boot Sequence

The boot process begins with the Runtime Security Engine (RSE), which loads the Application Processor (AP) BL2 image into the Trusted SRAM at a fixed address. Once loaded, the RSE signals the System Control Processor firmware (SCP-firmware) running on Safety Island Cluster 0 (SI CL0) to initiate the AP power-up sequence.

The SCP-firmware then sets the reset vector base address (RVBAR) for the AP, ensuring it starts executing BL2 from the designated address. Following this, the SCP-firmware powers on AP Cluster 0, allowing the AP to run AP BL2.

The following tasks are executed for each AP BL stage:

1. AP BL2:

- Performs the actions described in the Trusted Board Boot (TBB) document.
- (Optional step) Find the FIP image in a GPT partition, incase the FIP lies within in a GPT image.
- Copies the FW_CONFIG from Secure Flash to Trusted SRAM.
- (Optional step) Computes FW_CONFIG image measurement and invokes extend operation of TF-M Measured Boot secure partition.
- Completes its dynamic configuration from the FW_CONFIG loaded. This includes:
  - Parsing the configuration data.
  - Setting up the required system parameters.
- Reads and loads AP BL31 image into the Trusted SRAM.
- (Optional step) Computes AP BL31 image measurement and invokes extend operation of TF-M Measured Boot secure partition
- Copies the HW_CONFIG from Secure Flash to DRAM.
- (Optional step) Computes HW_CONFIG (DTB) image measurement and invokes extend operation of TF-M Measured Boot secure partition.
- (If present) Reads and loads AP BL32 (Secure Payload) image into Secure DRAM
- **(Optional step, if present) Computes AP BL32 image measurement and invokes extend**
  operation of TF-M Measured Boot secure partition
- (If present) Reads and loads the SPMC manifest (for S-EL2 firmware configuration) into Trusted SRAM and passes its location to BL31.
- Copies AP BL33 and Device tree blob from Secure Flash to Normal DRAM.
- (Optional step) Computes AP BL33 image measurement and invokes extend operation of TF-M Measured Boot secure partition
- Transfers the execution to AP BL31.

2. AP BL31:
   - Initializes Trusted Firmware-A Services.
   - Transfers the execution to AP BL32 and then transfers the execution to AP BL33
3. AP BL32:
   - Initializes Trusted OS (OP-TEE) environment
   - Initializes Secure Partitions
   - Transfers the execution back to AP BL31
   - During runtime, it facilitates secure communication between the normal world environment (e.g. Linux) and the Trusted Execution Environment.

Build Procedure (TF-A only)

- Ensure all Prerequisites are met, and the CROSS_COMPILE environment variable is properly set.
- Build TF-A:

```
make \
PLAT=rdaspen \
CREATE_KEYS=1 \
GENERATE_COT=1 \
TRUSTED_BOARD_BOOT=1 \
COT=tbbr \
ARM_ROTPK_LOCATION=devel_rsa \
ROT_KEY=plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem \
BL32=<path to optee binary> \
ARM_GPT_SUPPORT=1 \
BL33=<PATH-TO-BL33-BINARY> \
MEASURED_BOOT=1 \
```

Note: The BL32 flag is optional and should be set only if a Trusted OS is required. If it is not set, then BL33 will be loaded directly after BL31.

The ARM_GPT_SUPPORT flag is also optional. It must be enabled when the FIP image resides inside a GPT partition on Secure Flash.

The MEASURED_BOOT flag is also optional. It must be enabled in order to enable measurement of FW images and data.

This chapter holds documentation related to Arm's development platforms, including both software models (FVPs) and hardware development boards such as Juno.

7.4 Aspeed AST2700

Aspeed AST2700 is a 64-bit ARM SoC with 4-cores Cortex-A35 integrated. Each core operates at 1.6GHz.

7.4.1 Boot Flow

BootRom –> TF-A BL31 –> BL32 –> BL33 –> Linux Kernel

7.4.2 How to build

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=ast2700 SPD=opteed
```

7.5 Amlogic Meson A113D (AXG)

The Amlogic Meson A113D is a SoC with a quad core Arm Cortex-A53 running at ~1.2GHz. It also contains a Cortex-M3 used as SCP.

This port is a minimal implementation of BL31 capable of booting mainline U-Boot and Linux:

- SCPI support.
- Basic PSCI support (CPU_ON, CPU_OFF, SYSTEM_RESET, SYSTEM_OFF). Note that CPU0 can't be turned off, so there is a workaround to hide this from the caller.
- GICv2 driver set up.
- Basic SIP services (read efuse data, enable/disable JTAG).

In order to build it:

```
CROSS_COMPILE=aarch64-none-elf- make DEBUG=1 PLAT=axg [SPD=opteed]
     [AML_USE_ATOS=1 when using ATOS as BL32]
```

This port has been tested on a A113D board. After building it, follow the instructions in the U-Boot repository, replacing the mentioned **bl31.img** by the one built from this port.

7.6 Amlogic Meson S905 (GXBB)

The Amlogic Meson S905 is a SoC with a quad core Arm Cortex-A53 running at 1.5Ghz. It also contains a Cortex-M3 used as SCP.

This port is a minimal implementation of BL31 capable of booting mainline U-Boot and Linux:

- SCPI support.
- Basic PSCI support (CPU_ON, CPU_OFF, SYSTEM_RESET, SYSTEM_OFF). Note that CPU0 can't be turned off, so there is a workaround to hide this from the caller.
- GICv2 driver set up.
- Basic SIP services (read efuse data, enable/disable JTAG).

In order to build it:

```
CROSS_COMPILE=aarch64-linux-gnu- make DEBUG=1 PLAT=gxbb bl31
```

This port has been tested in a ODROID-C2. After building it, follow the instructions in the U-Boot repository, replacing the mentioned **bl31.bin** by the one built from this port.

7.7 Amlogic Meson S905x (GXL)

The Amlogic Meson S905x is a SoC with a quad core Arm Cortex-A53 running at 1.5Ghz. It also contains a Cortex-M3 used as SCP.

This port is a minimal implementation of BL31 capable of booting mainline U-Boot and Linux:

- SCPI support.
- Basic PSCI support (CPU_ON, CPU_OFF, SYSTEM_RESET, SYSTEM_OFF). Note that CPU0 can't be turned off, so there is a workaround to hide this from the caller.
- GICv2 driver set up.
- Basic SIP services (read efuse data, enable/disable JTAG).

In order to build it:

```
CROSS_COMPILE=aarch64-linux-gnu- make DEBUG=1 PLAT=gxl
```

Or, if willing to boot from U-Boot SPL (using standard params handling):

```
CROSS_COMPILE=aarch64-linux-gnu- make DEBUG=1 PLAT=gxl AML_STDPARAMS=1
```

This port has been tested on a Lepotato. After building it, follow the instructions in the gxlimg repository or U-Boot repository, replacing the mentioned **bl31.img** by the one built from this port.

7.8 Amlogic Meson S905X2 (G12A)

The Amlogic Meson S905X2 is a SoC with a quad core Arm Cortex-A53 running at ~1.8GHz. It also contains a Cortex-M3 used as SCP.

This port is a minimal implementation of BL31 capable of booting mainline U-Boot and Linux:

- SCPI support.
- Basic PSCI support (CPU_ON, CPU_OFF, SYSTEM_RESET, SYSTEM_OFF). Note that CPU0 can't be turned off, so there is a workaround to hide this from the caller.
- GICv2 driver set up.
- Basic SIP services (read efuse data, enable/disable JTAG).

In order to build it:

```
CROSS_COMPILE=aarch64-linux-gnu- make DEBUG=1 PLAT=g12a
```

This port has been tested on a SEI510 board. After building it, follow the instructions in the gxlimg repository or U-Boot repository, replacing the mentioned **bl31.img** by the one built from this port.

7.9 HiKey

HiKey is one of 96boards. Hisilicon Kirin6220 processor is installed on HiKey.

More information are listed in link.

7.9.1 How to build

Code Locations

- Trusted Firmware-A: link
- OP-TEE link
- edk2: link
- OpenPlatformPkg: link
- l-loader: link
- atf-fastboot: link

Build Procedure

- Fetch all the above repositories into local host. Make all the repositories in the same ${BUILD_PATH}.

```
git clone https://github.com/ARM-software/arm-trusted-firmware -b␣
↪integration
git clone https://github.com/OP-TEE/optee_os
git clone https://github.com/96boards-hikey/edk2 -b testing/hikey960_
↪v2.5
git clone https://github.com/96boards-hikey/OpenPlatformPkg -b␣
↪testing/hikey960_v1.3.4
git clone https://github.com/96boards-hikey/l-loader -b testing/
↪hikey960_v1.2
git clone https://github.com/96boards-hikey/atf-fastboot
```

- Create the symbol link to OpenPlatformPkg in edk2.

```
$cd ${BUILD_PATH}/edk2
$ln -sf ../OpenPlatformPkg
```

- Prepare AARCH64 && AARCH32 toolchain. Prepare python.

- If your hikey hardware is built by CircuitCo, update *OpenPlatformPkg/Platforms/Hisilicon/HiKey/HiKey.dsc* first. *(optional)* console on hikey.**

  ```
  DEFINE SERIAL_BASE=0xF8015000
  ```

  If your hikey hardware is built by LeMaker, nothing to do.

- Build it as debug mode. Create your own build script file or you could refer to **build_uefi.sh** in l-loader git repository.

  ```
  cd {BUILD_PATH}/arm-trusted-firmware
  sh ../l-loader/build_uefi.sh hikey
  ```

- Generate l-loader.bin and partition table for aosp. The eMMC capacity is either 8GB or 4GB. Just change "aosp-8g" to "linux-8g" for debian.

  ```
  cd ${BUILD_PATH}/l-loader
  ln -sf ${EDK2_OUTPUT_DIR}/FV/bl1.bin
  ln -sf ${EDK2_OUTPUT_DIR}/FV/bl2.bin
  ln -sf ${BUILD_PATH}/atf-fastboot/build/hikey/${FASTBOOT_BUILD_OPTION}/
  ↪bl1.bin fastboot.bin
  make hikey PTABLE_LST=aosp-8g
  ```

7.9.2 Setup Console

- Install ser2net. Use telnet as the console since UEFI fails to display Boot Manager GUI in minicom. **If you don't need Boot Manager GUI, just ignore this section.**

  ```
  $sudo apt-get install ser2net
  ```

- Configure ser2net.

  ```
  $sudo vi /etc/ser2net.conf
  ```

  Append one line for serial-over-USB in below. *#ser2net.conf*

  ```
  2004:telnet:0:/dev/ttyUSB0:115200 8DATABITS NONE 1STOPBIT banner
  ```

- Start ser2net

  ```
  $sudo killall ser2net
  $sudo ser2net -u
  ```

- Open the console.

  ```
  $telnet localhost 2004
  ```

  And you could open the console remotely, too.

7.9.3 Flash images in recovery mode

- Make sure Pin3-Pin4 on J15 are connected for recovery mode. Then power on HiKey.
- Remove the modemmanager package. This package may cause the idt tool failure.

```
$sudo apt-get purge modemmanager
```

- Run the command to download recovery.bin into HiKey.

```
$sudo python hisi-idt.py -d /dev/ttyUSB1 --img1 recovery.bin
```

- Update images. All aosp or debian images could be fetched from link.

```
$sudo fastboot flash ptable prm_ptable.img
$sudo fastboot flash loader l-loader.bin
$sudo fastboot flash fastboot fip.bin
$sudo fastboot flash boot boot.img
$sudo fastboot flash cache cache.img
$sudo fastboot flash system system.img
$sudo  fastboot flash userdata userdata.img
```

7.9.4 Boot UEFI in normal mode

- Make sure Pin3-Pin4 on J15 are open for normal boot mode. Then power on HiKey.
- Reference link

7.10 HiKey960

HiKey960 is one of 96boards. Hisilicon Hi3660 processor is installed on HiKey960.

More information are listed in link.

7.10.1 How to build

Code Locations

- Trusted Firmware-A: link
- OP-TEE: link
- edk2: link
- OpenPlatformPkg: link
- l-loader: link

Build Procedure

- Fetch all the above 5 repositories into local host. Make all the repositories in the same ${BUILD_PATH}.

```
git clone https://github.com/ARM-software/arm-trusted-firmware -b␣
↪integration
git clone https://github.com/OP-TEE/optee_os
git clone https://github.com/96boards-hikey/edk2 -b testing/hikey960_
↪v2.5
git clone https://github.com/96boards-hikey/OpenPlatformPkg -b␣
↪testing/hikey960_v1.3.4
git clone https://github.com/96boards-hikey/l-loader -b testing/
↪hikey960_v1.2
```

- Create the symbol link to OpenPlatformPkg in edk2.

```
$cd ${BUILD_PATH}/edk2
$ln -sf ../OpenPlatformPkg
```

- Prepare AARCH64 toolchain.
- If your hikey960 hardware is v1, update *OpenPlatformPkg/Platforms/Hisilicon/HiKey960/HiKey960.dsc* first. *(optional)*

```
DEFINE SERIAL_BASE=0xFDF05000
```

  If your hikey960 hardware is v2 or newer, nothing to do.

- Build it as debug mode. Create script file for build.

```
cd {BUILD_PATH}/arm-trusted-firmware
sh ../l-loader/build_uefi.sh hikey960
```

- Generate l-loader.bin and partition table. *Make sure that you're using the sgdisk in the l-loader directory.*

```
cd ${BUILD_PATH}/l-loader
ln -sf ${EDK2_OUTPUT_DIR}/FV/bl1.bin
ln -sf ${EDK2_OUTPUT_DIR}/FV/bl2.bin
ln -sf ${EDK2_OUTPUT_DIR}/FV/fip.bin
ln -sf ${EDK2_OUTPUT_DIR}/FV/BL33_AP_UEFI.fd
make hikey960
```

7.10.2 Setup Console

- Install ser2net. Use telnet as the console since UEFI will output window that fails to display in minicom.

```
$sudo apt-get install ser2net
```

- Configure ser2net.

```
$sudo vi /etc/ser2net.conf
```

Append one line for serial-over-USB in *#ser2net.conf*

```
2004:telnet:0:/dev/ttyUSB0:115200 8DATABITS NONE 1STOPBIT banner
```

- Start ser2net

```
$sudo killall ser2net
$sudo ser2net -u
```

- Open the console.

```
$telnet localhost 2004
```

And you could open the console remotely, too.

7.10.3 Boot UEFI in recovery mode

- Fetch that are used in recovery mode. The code location is in below. link
- Prepare recovery binary.

```
$cd tools-images-hikey960
$ln -sf ${BUILD_PATH}/l-loader/l-loader.bin
$ln -sf ${BUILD_PATH}/l-loader/fip.bin
$ln -sf ${BUILD_PATH}/l-loader/recovery.bin
```

- Prepare config file.

```
$vi config
# The content of config file
./sec_usb_xloader.img 0x00020000
./sec_uce_boot.img 0x6A908000
./recovery.bin 0x1AC00000
```

- Remove the modemmanager package. This package may causes hikey_idt tool failure.

```
$sudo apt-get purge modemmanager
```

- Run the command to download recovery.bin into HiKey960.

```
$sudo ./hikey_idt -c config -p /dev/ttyUSB1
```

- UEFI running in recovery mode. When prompt '.' is displayed on console, press hotkey 'f' in keyboard. Then Android fastboot app is running. The timeout of prompt '.' is 10 seconds.
- Update images.

```
$sudo fastboot flash ptable prm_ptable.img
$sudo fastboot flash xloader sec_xloader.img
$sudo fastboot flash fastboot l-loader.bin
$sudo fastboot flash fip fip.bin
$sudo fastboot flash boot boot.img
$sudo fastboot flash cache cache.img
$sudo fastboot flash system system.img
$sudo fastboot flash userdata userdata.img
```

- Notice: UEFI could also boot kernel in recovery mode, but BL31 isn't loaded in recovery mode.

7.10.4 Boot UEFI in normal mode

- Make sure "Boot Mode" switch is OFF for normal boot mode. Then power on HiKey960.
- Reference link

7.11 Intel Agilex SoCFPGA

Agilex SoCFPGA is a FPGA with integrated quad-core 64-bit Arm Cortex A53 processor.

Upon boot, Boot ROM loads bl2 into OCRAM. Bl2 subsequently initializes the hardware, then loads bl31 and bl33 (UEFI) into DDR and boots to bl33.

```
Boot ROM --> Trusted Firmware-A --> UEFI
```

7.11.1 How to build

Code Locations

- Trusted Firmware-A: link
- UEFI (to be updated with new upstreamed UEFI): link

Build Procedure

- Fetch all the above 2 repositories into local host. Make all the repositories in the same ${BUILD_PATH}.
- Prepare the AARCH64 toolchain.
- Build UEFI using Agilex platform as configuration This will be updated to use an updated UEFI using the latest EDK2 source

```
make CROSS_COMPILE=aarch64-linux-gnu- device=agx
```

- Build atf providing the previously generated UEFI as the BL33 image

```
make CROSS_COMPILE=aarch64-linux-gnu- bl2 fip PLAT=agilex
BL33=PEI.ROM
```

Install Procedure

- dd fip.bin to a A2 partition on the MMC drive to be booted in Agilex board.
- Generate a SOF containing bl2

```
aarch64-linux-gnu-objcopy -I binary -O ihex --change-addresses 0xffe00000 bl2.
↪bin bl2.hex
quartus_cpf --bootloader bl2.hex <quartus_generated_sof> <output_sof_with_bl2>
```

- Configure SOF to board

```
nios2-configure-sof <output_sof_with_bl2>
```

7.11.2 Boot trace

```
INFO:    DDR: DRAM calibration success.
INFO:    ECC is disabled.
NOTICE:  BL2: v2.1(debug)
NOTICE:  BL2: Built
INFO:    BL2: Doing platform setup
NOTICE:  BL2: Booting BL31
INFO:    Entry point address = 0xffe1c000
INFO:    SPSR = 0x3cd
NOTICE:  BL31: v2.1(debug)
NOTICE:  BL31: Built
INFO:    ARM GICv2 driver initialized
INFO:    BL31: Initializing runtime services
WARNING: BL31: cortex_a53
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0x50000
INFO:    SPSR = 0x3c9
```

7.12 Intel Stratix 10 SoCFPGA

Stratix 10 SoCFPGA is a FPGA with integrated quad-core 64-bit Arm Cortex A53 processor.

Upon boot, Boot ROM loads bl2 into OCRAM. Bl2 subsequently initializes the hardware, then loads bl31 and bl33 (UEFI) into DDR and boots to bl33.

```
Boot ROM --> Trusted Firmware-A --> UEFI
```

7.12.1 How to build

Code Locations

- Trusted Firmware-A: link
- UEFI (to be updated with new upstreamed UEFI): link

Build Procedure

- Fetch all the above 2 repositories into local host. Make all the repositories in the same ${BUILD_PATH}.
- Prepare the AARCH64 toolchain.
- Build UEFI using Stratix 10 platform as configuration This will be updated to use an updated UEFI using the latest EDK2 source

```
make CROSS_COMPILE=aarch64-linux-gnu- device=s10
```

- Build atf providing the previously generated UEFI as the BL33 image

```
make CROSS_COMPILE=aarch64-linux-gnu- bl2 fip PLAT=stratix10
BL33=PEI.ROM
```

Install Procedure

- dd fip.bin to a A2 partition on the MMC drive to be booted in Stratix 10 board.
- Generate a SOF containing bl2

```
aarch64-linux-gnu-objcopy -I binary -O ihex --change-addresses 0xffe00000 bl2.
↪bin bl2.hex
quartus_cpf --bootloader bl2.hex <quartus_generated_sof> <output_sof_with_bl2>
```

- Configure SOF to board

```
nios2-configure-sof <output_sof_with_bl2>
```

7.12.2 Boot trace

```
INFO:    DDR: DRAM calibration success.
INFO:    ECC is disabled.
INFO:    Init HPS NOC's DDR Scheduler.
NOTICE:  BL2: v2.0(debug):v2.0-809-g7f8474a-dirty
NOTICE:  BL2: Built : 17:38:19, Feb 18 2019
INFO:    BL2: Doing platform setup
INFO:    BL2: Loading image id 3
INFO:    Loading image id=3 at address 0xffe1c000
```

(continues on next page)

(continued from previous page)

```
INFO:    Image id=3 loaded: 0xffe1c000 - 0xffe24034
INFO:    BL2: Loading image id 5
INFO:    Loading image id=5 at address 0x50000
INFO:    Image id=5 loaded: 0x50000 - 0x550000
NOTICE:  BL2: Booting BL31
INFO:    Entry point address = 0xffe1c000
INFO:    SPSR = 0x3cd
NOTICE:  BL31: v2.0(debug):v2.0-810-g788c436-dirty
NOTICE:  BL31: Built : 15:17:16, Feb 20 2019
INFO:    ARM GICv2 driver initialized
INFO:    BL31: Initializing runtime services
WARNING: BL31: cortex_a53: CPU workaround for 855873 was missing!
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0x50000
INFO:    SPSR = 0x3c9
UEFI firmware (version 1.0 built at 11:26:18 on Nov  7 2018)
```

7.13 Marvell

7.13.1 Marvell Armada 8040 NBX Platform

The a80x0_nbx platform provides TF-A support for the Free Mobile Nodebox10G (NBX), a network appliance based on the Marvell Armada 8040 SoC with dual CP110 companion processors.

Hardware Configuration

- SoC: Marvell Armada 8040 (AP806 + 2x CP110)
- CPU: Quad-core ARM Cortex-A72 @ 1.3GHz
- DDR4 memory with ECC support (single channel, 32-bit)
- Dual CP110 companion processors (CP0 and CP1)
- SGMII 1G Ethernet on CP0 lane 5
- I2C buses for peripheral access (CP0: 100kHz, CP1: 400kHz)
- NS16550 UART console at 115200 baud
- eMMC boot via Xenon SDHCI controller

Building

For general build instructions (toolchain, environment variables, build options), refer to *TF-A Build Instructions for Marvell Platforms*.

```
make PLAT=a80x0_nbx DEBUG=1 LOG_LEVEL=40 USE_COHERENT_MEM=0 \
    PLAT_RECOVERY_IMAGE_ENABLE=1 all fip mrvl_flash
```

The final flash image is located at `build/a80x0_nbx/<debug|release>/flash-image.bin`.

Platform-Specific Features

Ramoops Buffer Preservation (ramoopsies)

On ECC-enabled DDR configurations, the memory controller must scrub all memory during initialization to establish valid parity bits. This would normally destroy the Linux kernel ramoops buffer containing crash logs from the previous boot.

The ramoopsies driver intercepts the DDR scrubbing function using the GNU linker `--wrap` feature, saving the 32KB ramoops buffer to SRAM before scrubbing and restoring it afterward. The driver also handles pending SError exceptions that occur when reading uninitialized ECC memory by installing a minimal exception vector that acknowledges and clears them.

This is implemented using `--wrap=mv_ddr_mem_scrubbing` without modifying the upstream mv-ddr-marvell repository.

Source: `plat/marvell/armada/a8k/a80x0_nbx/board/ramoopsies.c`

Memory layout:

```
Ramoops buffer address:   0x3FFF8000
Ramoops buffer size:      32 KB (0x8000)
End address:              0x3FFFFFFF (just below 1GB boundary)
```

Linux kernel configuration:

```
CONFIG_PSTORE=y
CONFIG_PSTORE_RAM=y
CONFIG_PSTORE_CONSOLE=y      # Optional: capture kernel console
CONFIG_PSTORE_PMSG=y         # Optional: capture userspace messages
```

Device tree configuration:

```
/ {
    reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        ramoops@3fff8000 {
            compatible = "ramoops";
```

(continues on next page)

(continued from previous page)

```
            reg = <0x0 0x3fff8000 0x0 0x8000>;
            record-size      = <0x2000>;
            console-size     = <0x2000>;
            pmsg-size        = <0x2000>;
            ftrace-size      = <0x0>;
            ecc-size         = <0x0>;
        };
    };
};
```

Alternatively, pass via kernel command line:

```
ramoops.mem_address=0x3fff8000 ramoops.mem_size=0x8000 \
    ramoops.record_size=0x2000
```

After a kernel crash and reboot, logs are available via pstore:

```
mount -t pstore pstore /sys/fs/pstore
cat /sys/fs/pstore/dmesg-ramoops-0
```

Note: Only cold boot (power cycle) triggers DDR scrubbing; warm reboot preserves RAM content without this mechanism.

UART Skip Image Recovery

The platform uses the `USER_DEFINED` skip image detection callback to implement software-based boot recovery without requiring a dedicated GPIO button. Users can trigger recovery mode by holding the `s` key during early boot, causing the bootloader to load from the secondary/recovery image instead of the primary firmware.

Enabled with `PLAT_RECOVERY_IMAGE_ENABLE=1`.

Usage:

1. Connect serial console (115200 8N1)
2. Power on or reset the board
3. Hold `s` key during boot
4. System will skip to recovery/secondary image

DDR Configuration

Memory is configured dynamically via SPD (Serial Presence Detect) EEPROM:

SPD I2C Address	0x53
SPD Page Select	0x36
I2C Controller	CP0 I2C (MPP 37/38)
Bus Width	64-bit with ECC
Speed Bin	DDR4-2400T

The topology defaults to Micron 8Gbit x8 devices but actual configuration is read from the DIMM SPD at runtime.

Flashing

Via SPI (from U-Boot):

```
tftpboot $loadaddr flash-image.bin
sf probe
sf erase 0 0x100000
sf write $loadaddr 0 $filesize
```

For UART recovery, use the Marvell WTP tools or a compatible UART download script when the board is in UART boot mode. See *TF-A UART Booting Instructions for Marvell Platforms* for details.

Board Files

Platform-specific source files are located under `plat/marvell/armada/a8k/a80x0_nbx/board/`:

- `dram_port.c` - DDR topology and SPD configuration
- `marvell_plat_config.c` - Memory maps, power management, skip image
- `phy-porting-layer.h` - SerDes PHY parameters
- `ramoopsies.c` - ECC memory crash log preservation

7.13.2 TF-A Build Instructions for Marvell Platforms

This section describes how to compile the Trusted Firmware-A (TF-A) project for Marvell's platforms.

Build Instructions

(1) Set the cross compiler

```
> export CROSS_COMPILE=/path/to/toolchain/aarch64-linux-gnu-
```

(2) Set path for FIP images:

Set U-Boot image path (relatively to TF-A root or absolute path)

```
> export BL33=path/to/u-boot.bin
```

For example: if U-Boot project (and its images) is located at `~/project/u-boot`, BL33 should be `~/project/u-boot/u-boot.bin`

Note: *u-boot.bin* should be used and not *u-boot-spl.bin*

Set MSS/SCP image path (mandatory only for A7K/A8K/CN913x when MSS_SUPPORT=1)

```
> export SCP_BL2=path/to/mrvl_scp_bl2*.img
```

(3) Armada-37x0 build requires WTP tools installation.

See below in the section "Tools and external components installation". Install ARM 32-bit cross compiler, which is required for building WTMI image for CM3

```
> sudo apt-get install gcc-arm-linux-gnueabi
```

(4) Clean previous build residuals (if any)

```
> make distclean
```

(5) Build TF-A

There are several build options:

- PLAT

  Supported Marvell platforms are:

  - a3700 - A3720 DB, EspressoBin and Turris MOX
  - a70x0
  - a70x0_amc - AMC board
  - a70x0_mochabin - Globalscale MOCHAbin
  - a80x0
  - a80x0_mcbin - MacchiatoBin
  - a80x0_nbx - Free Mobile Nodebox10G
  - a80x0_puzzle - IEI Puzzle-M801

  - t9130 - CN913x
  - t9130_cex7_eval - CN913x CEx7 Evaluation Board

- DEBUG

  Default is without debug information (=0). in order to enable it use `DEBUG=1`. Can be enabled also when building UART recovery images, there is no issue with it.

  Production TF-A images should be built without this debug option!

- LOG_LEVEL

  Defines the level of logging which will be purged to the default output port.

  - 0 - LOG_LEVEL_NONE
  - 10 - LOG_LEVEL_ERROR
  - 20 - LOG_LEVEL_NOTICE (default for DEBUG=0)
  - 30 - LOG_LEVEL_WARNING
  - 40 - LOG_LEVEL_INFO (default for DEBUG=1)
  - 50 - LOG_LEVEL_VERBOSE

- USE_COHERENT_MEM

  This flag determines whether to include the coherent memory region in the BL memory map or not. Enabled by default.

- LLC_ENABLE

  Flag defining the LLC (L3) cache state. The cache is enabled by default (`LLC_ENABLE=1`).

- LLC_SRAM

  Flag enabling the LLC (L3) cache SRAM support. The LLC SRAM is activated and used by Trusted OS (OP-TEE OS, BL32). The TF-A only prepares CCU address translation windows for SRAM address range at BL31 execution stage with window target set to DRAM-0. When Trusted OS activates LLC SRAM, the CCU window target is changed to SRAM. There is no reason to enable this feature if OP-TEE OS built with CFG_WITH_PAGER=n. Only set LLC_SRAM=1 if OP-TEE OS is built with CFG_WITH_PAGER=y.

- MARVELL_SECURE_BOOT

  Build trusted(=1)/non trusted(=0) image, default is non trusted. This parameter is used only for `mrvl_flash` and `mrvl_uart` targets.

- MV_DDR_PATH

  This parameter is required for `mrvl_flash` and `mrvl_uart` targets. For A7K/A8K/CN913x it is used for BLE build and for Armada37x0 it used for ddr_tool build.

  Specify path to the full checkout of Marvell mv-ddr-marvell git repository. Checkout must contain also .git subdirectory because mv-ddr build process calls git commands.

  Do not remove any parts of git checkout becuase build process and other applications need them for correct building and version determination.

CN913x specific build options:

- CP_NUM

  Total amount of CPs (South Bridge) connected to AP. When the parameter is omitted, the build uses the default number of CPs, which is a number of embedded CPs inside the package: 1 or 2 depending on the SoC used. The parameter is valid for OcteonTX2 CN913x SoC family (PLAT=t9130), which can have external CPs connected to the MCI ports. Valid values with CP_NUM are in a range of 1 to 3.

A7K/A8K/CN913x specific build options:

- BLE_PATH

  Points to BLE (Binary ROM extension) sources folder. The parameter is optional, its default value is `plat/marvell/armada/a8k/common/ble` which uses TF-A in-tree BLE implementation.

- MSS_SUPPORT

  When `MSS_SUPPORT=1`, then TF-A includes support for Management SubSystem (MSS). When enabled it is required to specify path to the MSS firmware image via `SCP_BL2` option.

  This option is by default enabled.

- SCP_BL2

  Specify path to the MSS fimware image binary which will run on Cortex-M3 coprocessor. It is available in Marvell binaries-marvell git repository. Required when `MSS_SUPPORT=1`.

Globalscale MOCHAbin specific build options:

- DDR_TOPOLOGY

  The DDR topology map index/name, default is 0.

  Supported Options:

  - 0 - DDR4 1CS 2GB
  - 1 - DDR4 1CS 4GB
  - 2 - DDR4 2CS 8GB

Armada37x0 specific build options:

- HANDLE_EA_EL3_FIRST_NS

  When `HANDLE_EA_EL3_FIRST_NS=1`, External Aborts and SError Interrupts, resulting from errors in NS world, will be always trapped in TF-A. TF-A in this case enables dirty hack / workaround for a bug found in U-Boot and Linux kernel PCIe controller driver pci-aardvark.c, traps and then masks SError interrupt caused by AXI SLVERR on external access (syndrome 0xbf000002).

  Otherwise when `HANDLE_EA_EL3_FIRST_NS=0`, these exceptions will be trapped in the current exception level (or in EL1 if the current exception level is EL0). So exceptions caused by U-Boot will be trapped in U-Boot, exceptions caused by Linux kernel (or user applications) will be trapped in Linux kernel.

Mentioned bug in pci-aardvark.c driver is fixed in U-Boot version v2021.07 and Linux kernel version v5.13 (workarounded since Linux kernel version 5.9) and also backported in Linux kernel stable releases since versions v5.12.13, v5.10.46, v5.4.128, v4.19.198, v4.14.240.

If target system has already patched version of U-Boot and Linux kernel then it is strongly recommended to not enable this workaround as it disallows propagating of all External Aborts to running Linux kernel and makes correctable errors as fatal aborts.

This option is now disabled by default. In past this option has different name "HANDLE_EA_EL3_FIRST" and was enabled by default in TF-A versions v2.2, v2.3, v2.4 and v2.5.

- CM3_SYSTEM_RESET

  When `CM3_SYSTEM_RESET=1`, the Cortex-M3 secure coprocessor will be used for system reset.

  TF-A will send command 0x0009 with a magic value via the rWTM mailbox interface to the Cortex-M3 secure coprocessor. The firmware running in the coprocessor must either implement this functionality or ignore the 0x0009 command (which is true for the firmware from A3700-utils-marvell repository). If this option is enabled but the firmware does not support this command, an error message will be printed prior trying to reboot via the usual way.

  This option is needed on Turris MOX as a workaround to a HW bug which causes reset to sometime hang the board.

- A3720_DB_PM_WAKEUP_SRC

  For Armada 3720 Development Board only, when `A3720_DB_PM_WAKEUP_SRC=1`, TF-A will setup PM wake up src configuration. This option is disabled by default.

Armada37x0 specific build options for `mrvl_flash` and `mrvl_uart` targets:

- DDR_TOPOLOGY

  The DDR topology map index/name, default is 0.

  Supported Options:

  - 0 - DDR3 1CS 512MB (DB-88F3720-DDR3-Modular, EspressoBin V3-V5)
  - 1 - DDR4 1CS 512MB (DB-88F3720-DDR4-Modular)
  - 2 - DDR3 2CS 1GB (EspressoBin V3-V5)
  - 3 - DDR4 2CS 4GB (DB-88F3720-DDR4-Modular)
  - 4 - DDR3 1CS 1GB (DB-88F3720-DDR3-Modular, EspressoBin V3-V5)
  - 5 - DDR4 1CS 1GB (EspressoBin V7, EspressoBin-Ultra)
  - 6 - DDR4 2CS 2GB (EspressoBin V7)
  - 7 - DDR3 2CS 2GB (EspressoBin V3-V5)
  - CUST - CUSTOMER BOARD (Customer board settings)

- CLOCKSPRESET

The clock tree configuration preset including CPU and DDR frequency, default is CPU_800_DDR_800.

- CPU_600_DDR_600 - CPU at 600 MHz, DDR at 600 MHz
- CPU_800_DDR_800 - CPU at 800 MHz, DDR at 800 MHz
- CPU_1000_DDR_800 - CPU at 1000 MHz, DDR at 800 MHz
- CPU_1200_DDR_750 - CPU at 1200 MHz, DDR at 750 MHz

Look at Armada37x0 chip package marking on board to identify correct CPU frequency. The last line on package marking (next line after the 88F37x0 line) should contain:

- C080 or I080 - chip with 800 MHz CPU - use `CLOCKSPRESET=CPU_800_DDR_800`
- C100 or I100 - chip with 1000 MHz CPU - use `CLOCKSPRESET=CPU_1000_DDR_800`
- C120 - chip with 1200 MHz CPU - use `CLOCKSPRESET=CPU_1200_DDR_750`

• BOOTDEV

The flash boot device, default is `SPINOR`.

Currently, Armada37x0 only supports `SPINOR`, `SPINAND`, `EMMCNORM` and `SATA`:

- SPINOR - SPI NOR flash boot
- SPINAND - SPI NAND flash boot
- EMMCNORM - eMMC Download Mode

  Download boot loader or program code from eMMC flash into CM3 or CA53
  Requires full initialization and command sequence

- SATA - SATA device boot

  Image needs to be stored at disk LBA 0 or at disk partition with MBR type 0x4d (ASCII 'M' as in Marvell) or at disk partition with GPT partition type GUID `6828311A-BA55-42A4-BCDE-A89BB5EDECAE`.

• PARTNUM

The boot partition number, default is 0.

To boot from eMMC, the value should be aligned with the parameter in U-Boot with name of `CONFIG_SYS_MMC_ENV_PART`, whose value by default is 1. For details about CONFIG_SYS_MMC_ENV_PART, please refer to the U-Boot build instructions.

• WTMI_IMG

The path of the binary can point to an image which does nothing, an image which supports EFUSE or a customized CM3 firmware binary. The default image is `fuse.bin` that built from sources in WTP folder, which is the next option. If the default image is OK, then this option should be skipped.

Please note that this is not a full WTMI image, just a main loop without hardware initialization code. Final WTMI image is built from this WTMI_IMG binary and sys-init code from the WTP directory which sets DDR and CPU clocks according to DDR_TOPOLOGY and CLOCKSPRESET options.

CZ.NIC as part of Turris project released free and open source WTMI application firmware `wtmi_app.bin` for all Armada 3720 devices. This firmware includes additional features like access to Hardware Random Number Generator of Armada 3720 SoC which original Marvell's `fuse.bin` image does not have.

CZ.NIC's Armada 3720 Secure Firmware is available at website:

https://gitlab.nic.cz/turris/mox-boot-builder/

- WTP

  Specify path to the full checkout of Marvell A3700-utils-marvell git repository. Checkout must contain also .git subdirectory because WTP build process calls git commands.

  WTP build process uses also Marvell mv-ddr-marvell git repository specified in MV_DDR_PATH option.

  Do not remove any parts of git checkout becuase build process and other applications need them for correct building and version determination.

- CRYPTOPP_PATH

  Use this parameter to point to Crypto++ source code directory. If this option is specified then Crypto++ source code in CRYPTOPP_PATH directory will be automatically compiled. Crypto++ library is required for building WTP image tool. Either CRYPTOPP_PATH or CRYPTOPP_LIBDIR with CRYPTOPP_INCDIR needs to be specified for Armada37x0.

- CRYPTOPP_LIBDIR

  Use this parameter to point to the directory with compiled Crypto++ library. By default it points to the CRYPTOPP_PATH.

  On Debian systems it is possible to install system-wide Crypto++ library via command `apt install libcrypto++-dev` and specify CRYPTOPP_LIBDIR to `/usr/lib/`.

- CRYPTOPP_INCDIR

  Use this parameter to point to the directory with header files of Crypto++ library. By default it points to the CRYPTOPP_PATH.

  On Debian systems it is possible to install system-wide Crypto++ library via command `apt install libcrypto++-dev` and specify CRYPTOPP_INCDIR to `/usr/include/crypto++/`.

For example, in order to build the image in debug mode with log level up to 'notice' level run

```
> make DEBUG=1 USE_COHERENT_MEM=0 LOG_LEVEL=20 PLAT=<MARVELL_PLATFORM> mrvl_
↪flash
```

And if we want to build a Armada37x0 image in debug mode with log level up to 'notice' level, the image has the preset CPU at 1000 MHz, preset DDR3 at 800 MHz, the DDR topology of DDR4 2CS, the image boot from SPI NOR flash partition 0, and the image is non trusted in WTP, the command line is as following

```
> make DEBUG=1 USE_COHERENT_MEM=0 LOG_LEVEL=20 CLOCKSPRESET=CPU_1000_DDR_800 \
    MARVELL_SECURE_BOOT=0 DDR_TOPOLOGY=3 BOOTDEV=SPINOR PARTNUM=0 PLAT=a3700 \
    MV_DDR_PATH=/path/to/mv-ddr-marvell/ WTP=/path/to/A3700-utils-marvell/ \
    CRYPTOPP_PATH=/path/to/cryptopp/ BL33=/path/to/u-boot.bin \
    all fip mrvl_bootimage mrvl_flash mrvl_uart
```

To build just TF-A without WTMI image (useful for A3720 Turris MOX board), run following command:

```
> make USE_COHERENT_MEM=0 PLAT=a3700 CM3_SYSTEM_RESET=1 BL33=/path/to/u-boot.
↪bin \
    CROSS_COMPILE=aarch64-linux-gnu- mrvl_bootimage
```

Here is full example how to build production release of Marvell firmware image (concatenated binary of Marvell's A3720 sys-init, CZ.NIC's Armada 3720 Secure Firmware, TF-A and U-Boot) for EspressoBin board (PLAT=a3700) with 1GHz CPU (CLOCKSPRESET=CPU_1000_DDR_800) and 1GB DDR4 RAM (DDR_TOPOLOGY=5):

```
> git clone https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git
> git clone https://source.denx.de/u-boot/u-boot.git
> git clone https://github.com/weidai11/cryptopp.git
> git clone https://github.com/MarvellEmbeddedProcessors/mv-ddr-marvell.git
> git clone https://github.com/MarvellEmbeddedProcessors/A3700-utils-marvell.
↪git
> git clone https://gitlab.nic.cz/turris/mox-boot-builder.git
> make -C u-boot CROSS_COMPILE=aarch64-linux-gnu- mvebu_espressobin-88f3720_
↪defconfig u-boot.bin
> make -C mox-boot-builder CROSS_CM3=arm-linux-gnueabi- wtmi_app.bin
> make -C trusted-firmware-a CROSS_COMPILE=aarch64-linux-gnu- CROSS_CM3=arm-
↪linux-gnueabi- \
    USE_COHERENT_MEM=0 PLAT=a3700 CLOCKSPRESET=CPU_1000_DDR_800 DDR_
↪TOPOLOGY=5 \
    MV_DDR_PATH=$PWD/mv-ddr-marvell/ WTP=$PWD/A3700-utils-marvell/ \
    CRYPTOPP_PATH=$PWD/cryptopp/ BL33=$PWD/u-boot/u-boot.bin \
    WTMI_IMG=$PWD/mox-boot-builder/wtmi_app.bin FIP_ALIGN=0x100 mrvl_flash
```

Produced Marvell firmware flash image: `trusted-firmware-a/build/a3700/release/flash-image.bin`

Special Build Flags

- **PLAT_RECOVERY_IMAGE_ENABLE**
  When set this option to enable secondary recovery function when build atf. In order to build UART recovery image this operation should be disabled for A7K/A8K/CN913x because of hardware limitation (boot from secondary image can interrupt UART recovery process). This MACRO definition is set in `plat/marvell/armada/a8k/common/include/platform_def.h` file.

- **DDR32**

  In order to work in 32bit DDR, instead of the default 64bit ECC DDR, this flag should be set to 1.

For more information about build options, please refer to the *Build Options* document.

Build output

Marvell's TF-A compilation generates 8 files:

- ble.bin - BLe image (not available for Armada37x0)
- bl1.bin - BL1 image
- bl2.bin - BL2 image
- bl31.bin - BL31 image
- fip.bin - FIP image (contains BL2, BL31 & BL33 (U-Boot) images)
- boot-image.bin - TF-A image (contains BL1 and FIP images)
- flash-image.bin - Flashable Marvell firmware image. For Armada37x0 it contains TIM, WTMI and boot-image.bin images. For other platforms it contains BLe and boot-image.bin images. Should be placed on the boot flash/device.
- uart-images.tgz.bin - GZIPed TAR archive which contains Armada37x0 images for booting via UART. Could be loaded via Marvell's WtpDownload tool from A3700-utils-marvell repository.

Additional make target `mrvl_bootimage` produce `boot-image.bin` file. Target `mrvl_flash` produce final `flash-image.bin` file and target `mrvl_uart` produce `uart-images.tgz.bin` file.

Tools and external components installation

Armada37x0 Builds require installation of additional components

(1) ARM cross compiler capable of building images for the service CPU (CM3). This component is usually included in the Linux host packages. On Debian/Ubuntu hosts the default GNU ARM tool chain can be installed using the following command

```
> sudo apt-get install gcc-arm-linux-gnueabi
```

Only if required, the default tool chain prefix `arm-linux-gnueabi-` can be overwritten using the environment variable `CROSS_CM3`. Example for BASH shell

```
> export CROSS_CM3=/opt/arm-cross/bin/arm-linux-gnueabi
```

(2) DDR initialization library sources (mv_ddr) available at the following repository (use the "master" branch):

https://github.com/MarvellEmbeddedProcessors/mv-ddr-marvell.git

(3) Armada3700 tools available at the following repository (use the "master" branch):

https://github.com/MarvellEmbeddedProcessors/A3700-utils-marvell.git

(4) Crypto++ library available at the following repository:

https://github.com/weidai11/cryptopp.git

(5) Optional CZ.NIC's Armada 3720 Secure Firmware:

https://gitlab.nic.cz/turris/mox-boot-builder.git

Armada70x0, Armada80x0 and CN913x Builds require installation of additional components

(1) DDR initialization library sources (mv_ddr) available at the following repository (use the "master" branch):

https://github.com/MarvellEmbeddedProcessors/mv-ddr-marvell.git

(2) MSS Management SubSystem Firmware available at the following repository (use the "binaries-marvell-armada-SDK10.0.1.0" branch):

https://github.com/MarvellEmbeddedProcessors/binaries-marvell.git

7.13.3 TF-A UART Booting Instructions for Marvell Platforms

This section describes how to temporary boot the Trusted Firmware-A (TF-A) project over UART without flashing it to non-volatile storage for Marvell's platforms.

See *TF-A Build Instructions for Marvell Platforms* how to build `mrvl_uart` and `mrvl_flash` targets used in this section.

Armada37x0 UART image downloading

There are two options how to download UART image into any Armada37x0 board.

Marvell Wtpdownloader

Marvell Wtpdownloader works only with UART images stored in separate files and supports only upload speed with 115200 bauds. Target `mrvl_uart` produces GZIPed TAR archive `uart-images.tgz.bin` with either three files `TIM_ATF.bin`, `wtmi_h.bin` and `boot-image_h.bin` for non-secure boot or with four files `TIM_ATF_TRUSTED.bin`, `TIMN_ATF_TRUSTED.bin`, `wtmi_h.bin` and `boot-image_h.bin` when secure boot is enabled.

Compilation:

```
> git clone https://github.com/MarvellEmbeddedProcessors/A3700-utils-marvell.
↪git
> make -C A3700-utils-marvell/wtptp/src/Wtpdownloader_Linux -f makefile.mk
```

It produces executable binary `A3700-utils-marvell/wtptp/src/Wtpdownloader_Linux/WtpDownload_linux`

To download images from `uart-images.tgz.bin` archive unpack it and for non-secure boot variant run:

```
> stty -F /dev/ttyUSB<port#> clocal
> WtpDownload_linux -P UART -C <port#> -E -B TIM_ATF.bin -I wtmi_h.bin -I␣
↪boot-image_h.bin
```

After that immediately start terminal on `/dev/ttyUSB<port#>` to see boot output.

CZ.NIC mox-imager

CZ.NIC mox-imager supports all Armada37x0 boards (not only Turris MOX as name suggests). It works with either with separate files from `uart-images.tgz.bin` archive (like Marvell Wtpdownloader) produced by `mrvl_uart` target or also with `flash-image.bin` file produced by `mrvl_flash` target, which is the exactly same file as used for flashing. So when using CZ.NIC mox-imager there is no need to build separate files for UART flashing like in case with Marvell Wtpdownloader.

CZ.NIC mox-imager moreover supports higher upload speeds up to the 6000000 bauds (which seems to be limit of Armada37x0 SoC) which is much higher and faster than Marvell Wtpdownloader.

Compilation:

```
> git clone https://gitlab.nic.cz/turris/mox-imager.git
> make -C mox-imager
```

It produces executable binary `mox-imager/mox-imager`

To download single file image built by `mrvl_flash` target at the highest speed, run:

```
> mox-imager -D /dev/ttyUSB<port#> -E -b 6000000 -t flash-image.bin
```

To download images from `uart-images.tgz.bin` archive built by `mrvl_uart` target for non-secure boot variant (like Wtpdownloader) but at the highest speed, first unpack `uart-images.tgz.bin` archive and then run:

```
> mox-imager -D /dev/ttyUSB<port#> -E -b 6000000 -t TIM_ATF.bin wtmi_h.bin␣
↪boot-image_h.bin
```

CZ.NIC mox-imager after successful download will start its own mini terminal (option `-t`) to not loose any boot output. It also prints boot output which is sent either by image files or by bootrom during transferring of image files. This mini terminal can be quit by CTRL-\ + C keypress.

A7K/A8K/CN913x UART image downloading

A7K/A8K/CN913x uses same image `flash-image.bin` for both flashing and booting over UART. For downloading image over UART it is possible to use mvebu64boot tool.

Compilation:

```
> git clone https://github.com/pali/mvebu64boot.git
> make -C mvebu64boot
```

It produces executable binary `mvebu64boot/mvebu64boot`

To download `flash-image.bin` image run:

```
> mvebu64boot -t -b flash-image.bin /dev/ttyUSB0
```

After successful download it will start own mini terminal (option `-t`) like CZ.NIC mox-imager.

7.13.4 TF-A Porting Guide for Marvell Platforms

This section describes how to port TF-A to a customer board, assuming that the SoC being used is already supported in TF-A.

Source Code Structure

- The customer platform specific code shall reside under `plat/marvell/armada/<soc family>/<soc>_cust` (e.g. 'plat/marvell/armada/a8k/a7040_cust').
- The platform name for build purposes is called `<soc>_cust` (e.g. `a7040_cust`).
- The build system will reuse all files from within the soc directory, and take only the porting files from the customer platform directory.

Files that require porting are located at `plat/marvell/armada/<soc family>/<soc>_cust` directory.

Armada-70x0/Armada-80x0 Porting

SoC Physical Address Map (marvell_plat_config.c)

This file describes the SoC physical memory mapping to be used for the CCU, IOWIN, AXI-MBUS and IOB address decode units (Refer to the functional spec for more details).

In most cases, using the default address decode windows should work OK.

In cases where a special physical address map is needed (e.g. Special size for PCIe MEM windows, large memory mapped SPI flash…), then porting of the SoC memory map is required.

Note: For a detailed information on how CCU, IOWIN, AXI-MBUS & IOB work, please refer to the SoC functional spec, and under `docs/plat/marvell/armada/misc/mvebu-[ccu/iob/amb/io-win].rst` files.

boot loader recovery (marvell_plat_config.c)

- Background:

  Boot rom can skip the current image and choose to boot from next position if a specific value (`0xDEADB002`) is returned by the ble main function. This feature is used for boot loader recovery by booting from a valid flash-image saved in next position on flash (e.g. address 2M in SPI flash).

  Supported options to implement the skip request are:

  - GPIO
  - I2C
  - User defined

- Porting:

  Under marvell_plat_config.c, implement struct skip_image that includes specific board parameters.

  Warning: To disable this feature make sure the struct skip_image is not implemented.

- Example:

In A7040-DB specific implementation (`plat/marvell/armada/a8k/a70x0/board/marvell_plat_config.c`), the image skip is implemented using GPIO: mpp 33 (SW5).

Before resetting the board make sure there is a valid image on the next flash address:

> -tftp [valid address] flash-image.bin -sf update [valid address] 0x2000000 [size]

Press reset and keep pressing the button connected to the chosen GPIO pin. A skip image request message is printed on the screen and boot rom boots from the saved image at the next position.

DDR Porting (dram_port.c)

This file defines the dram topology and parameters of the target board.

The DDR code is part of the BLE component, which is an extension of ARM Trusted Firmware (TF-A).

The DDR driver called mv_ddr is released separately apart from TF-A sources.

The BLE and consequently, the DDR init code is executed at the early stage of the boot process.

Each supported platform of the TF-A has its own DDR porting file called dram_port.c located at `atf/plat/marvell/armada/a8k/<platform>/board` directory.

Please refer to '<path_to_mv_ddr_sources>/doc/porting_guide.txt' for detailed porting description.

The build target directory is "build/<platform>/release/ble".

Comphy Porting (phy-porting-layer.h or phy-default-porting-layer.h)

- **Background:**

  Some of the comphy's parameters value depend on the HW connection between the SoC and the PHY. Every board type has specific HW characteristics like wire length. Due to those differences some comphy parameters vary between board types. Therefore each board type can have its own list of values for all relevant comphy parameters. The PHY porting layer specifies which parameters need to be suited and the board designer should provide relevant values.

  The PHY porting layer simplifies updating static values per board type, which are now grouped in one place.

  Note: The parameters for the same type of comphy may vary even for the same board type, it is because the lanes from comphy-x to some PHY may have different HW characteristic than lanes from comphy-y to the same (multiplexed) or other PHY.

- **Porting:**

  The porting layer for PHY was introduced in TF-A. There is one file `drivers/marvell/comphy/phy-default-porting-layer.h` which contains the defaults. Those default parameters are used only if there is no appropriate phy-porting-layer.h file under: `plat/marvell/armada/<soc family>/<platform>/board/phy-porting-layer.h`. If the phy-porting-layer.h exists, the phy-default-porting-layer.h is not going to be included.

  Warning: Not all comphy types are already reworked to support the PHY porting layer, currently the porting layer is supported for XFI/SFI and SATA comphy types.

  The easiest way to prepare the PHY porting layer for custom board is to copy existing example to a new platform:

  - cp `plat/marvell/armada/a8k/a80x0/board/phy-porting-layer.h` "plat/marvell/armada/<soc family>/<platform>/board/phy-porting-layer.h"
  - adjust relevant parameters or
  - if different comphy index is used for specific feature, move it to proper table entry and then adjust.

  Note: The final table size with comphy parameters can be different, depending on the CP module count for given SoC type.

- **Example:**

  Example porting layer for armada-8040-db is under: `plat/marvell/armada/a8k/a80x0/board/phy-porting-layer.h`

  Note: If there is no PHY porting layer for new platform (missing phy-porting-layer.h), the default values are used (drivers/marvell/comphy/phy-default-porting-layer.h) and the user is warned:

  Warning: “Using default comphy parameters - it may be required to suit them for your board”.

7.13.5 Address decoding flow and address translation units of Marvell Armada 8K SoC family

```
+--------------------------------------------------------------------------------
↪---------------------+
|                                                                          +-------------
↪+    +---------------+ |
|                                                                          | Memory      ␣
↪+-----   DRAM CS    | |
|+------------+ +------------+ +------------+                             | Controller ␣
↪|    +---------------+ |
||  AP DMA    | |            | |            |                             +-------------
↪+                       |
||  SD/eMMC   | | CA72 CPUs | |   AP MSS   |                             +-------------
↪+                       |
||  MCI-0/1   | |            | |            |                             | Memory      ␣
↪|                       |
|+------+------+ +--+---------+ +--------+--+  +------------+             | Controller ␣
↪|     +--------------+ |
|       |          |                   |     |             +----- Translaton ␣
↪|     |AP            | |
|       |          |                   |     |             |    +-------------
↪+     |Configuration| |
|       |          |                   +-----+             +------------------
↪------Space        | |
|       |          | +--------------+          |   CCU       |                   ␣
↪     +-------------+ |
|       |          | | MMU         +----------+   Windows   |   +------------+  ␣
↪     +-------------+ |
|       |         +-| translation |          |   Lookup    +----          +---
↪------   AP SPI     | |
|       |            +--------------+          |             |   |           |  ␣
↪     +-------------+ |
|       |            +--------------+          |             |   |  IO       |  ␣
↪     +-------------+ |
|       +------------| SMMU        +----------+             |   |  Windows  +---
↪------   AP MCI0/1   | |
|                    | translation |          +------------+   |  Lookup   |  ␣
```

(continues on next page)

(continued from previous page)

```
↪      +--------------+  |
|                      +----------+---+                          |             |   ␣
↪      +--------------+  |
|             -              |                               |             +---
↪------    AP STM     |  |
|            +-----------------                              |             |   ␣
↪      +--------------+  |
| AP            |                |                           +--+----------+   ␣
↪                        |
+--------------------------------------------------------------|-------------
↪---------------------+
+--------------|-----------------------------------------------|-------------
↪---------------------+
| CP            |            +-------------+              +-----+-----+      ␣
↪+-------------------+  |
|               |            |             |              |            +-------
↪   SB CFG Space    |  |
|               |            |   DIOB      |              |            |      ␣
↪+-------------------+  |
|               |            |   Windows   ---------------- IOB        |      ␣
↪+-------------------+  |
|               |            |   Control   |              |  Windows   +------
↪| SB PCIe-0 - PCIe2 |  |
|               |            |             |              |  Lookup    |      ␣
↪+-------------------+  |
|               |            +------+------+              |            |      ␣
↪+-------------------+  |
|               |                   |                     |            +------
↪+      SB NAND      |  |
|               |                   |                     +-----+-----+      ␣
↪+-------------------+  |
|               |                   |                           |              ␣
↪                        |
|               |                   |                           |              ␣
↪                        |
|   +-------------------+   +-------------+               +-----+-----+      ␣
↪+-------------------+  |
|   | Network Engine    |   |             |               |            +-------
↪  SB SPI-0/SPI-1    |  |
|   | Security Engine   |   | PCIe, MSS   |               |  RUNIT     |      ␣
↪+-------------------+  |
|   | SATA, USB         |   | DMA         |               |  Windows   |      ␣
↪+-------------------+  |
|   | SD/eMMC           |   |             |               |  Lookup    +-------
↪   SB Device Bus   |  |
|   | TDM, I2C          |   |             |               |            |      ␣
↪+-------------------+  |
|   +-------------------+   +-------------+               +------------+       ␣
↪                        |
|                                                                              ␣
↪                        |
+-----------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪---------------------+
```

7.13.6 AMB - AXI MBUS address decoding

AXI to M-bridge decoding unit driver for Marvell Armada 8K and 8K+ SoCs.

The Runit offers a second level of address windows lookup. It is used to map transaction towards the CD BootROM, SPI0, SPI1 and Device bus (NOR).

The Runit contains eight configurable windows. Each window defines a contiguous, address space and the properties associated with that address space.

```
Unit         Bank          ATTR
Device-Bus   DEV_BOOT_CS   0x2F
             DEV_CS0       0x3E
             DEV_CS1       0x3D
             DEV_CS2       0x3B
             DEV_CS3       0x37
SPI-0        SPI_A_CS0     0x1E
             SPI_A_CS1     0x5E
             SPI_A_CS2     0x9E
             SPI_A_CS3     0xDE
             SPI_A_CS4     0x1F
             SPI_A_CS5     0x5F
             SPI_A_CS6     0x9F
             SPI_A_CS7     0xDF
SPI          SPI_B_CS0     0x1A
             SPI_B_CS1     0x5A
             SPI_B_CS2     0x9A
             SPI_B_CS3     0xDA
BOOT_ROM     BOOT_ROM      0x1D
UART         UART          0x01
```

Mandatory functions

- **marvell_get_amb_memory_map**
  Returns the AMB windows configuration and the number of windows

Mandatory structures

- **amb_memory_map**
  Array that include the configuration of the windows. Every window/entry is a struct which has 2 parameters:
  - Base address of the window
  - Attribute of the window

Examples

```
struct addr_map_win amb_memory_map[] = {
    {0xf900,          AMB_DEV_CS0_ID},
};
```

7.13.7 Marvell CCU address decoding bindings

CCU configuration driver (1st stage address translation) for Marvell Armada 8K and 8K+ SoCs.

The CCU node includes a description of the address decoding configuration.

Mandatory functions

- **marvell_get_ccu_memory_map**
  Return the CCU windows configuration and the number of windows of the specific AP.

Mandatory structures

- **ccu_memory_map**
  Array that includes the configuration of the windows. Every window/entry is a struct which has 3 parameters:
  - Base address of the window
  - Size of the window
  - Target-ID of the window

Example

```
struct addr_map_win ccu_memory_map[] = {
        {0x00000000f2000000,     0x00000000e000000,      IO_0_TID}, /* IO␣
↪window */
};
```

7.13.8 Marvell IO WIN address decoding bindings

IO Window configuration driver (2nd stage address translation) for Marvell Armada 8K and 8K+ SoCs.

The IO WIN includes a description of the address decoding configuration.

Transactions that are decoded by CCU windows as IO peripheral, have an additional layer of decoding. This additional address decoding layer defines one of the following targets:

- **0x0** = BootRom

- **0x1** = STM (Serial Trace Macro-cell, a programmer's port into trace stream)
- **0x2** = SPI direct access
- **0x3** = PCIe registers
- **0x4** = MCI Port
- **0x5** = PCIe port

Mandatory functions

- **marvell_get_io_win_memory_map**
  Returns the IO windows configuration and the number of windows of the specific AP.

Mandatory structures

- **io_win_memory_map**
  Array that include the configuration of the windows. Every window/entry is a struct which has 3 parameters:
  - Base address of the window
  - Size of the window
  - Target-ID of the window

Example

```
struct addr_map_win io_win_memory_map[] = {
        {0x00000000fe000000,    0x000000001f00000,      PCIE_PORT_TID}, /*␣
↪PCIe window 31Mb for PCIe port*/
        {0x00000000ffe00000,    0x000000000100000,      PCIE_REGS_TID}, /*␣
↪PCI-REG window 64Kb for PCIe-reg*/
        {0x00000000f6000000,    0x000000000100000,      MCIPHY_TID},    /*␣
↪MCI window  1Mb for PHY-reg*/
};
```

7.13.9 Marvell IOB address decoding bindings

IO bridge configuration driver (3rd stage address translation) for Marvell Armada 8K and 8K+ SoCs.

The IOB includes a description of the address decoding configuration.

IOB supports up to n (in CP110 n=24) windows for external memory transaction. When a transaction passes through the IOB, its address is compared to each of the enabled windows. If there is a hit and it passes the security checks, it is advanced to the target port.

Mandatory functions

- **marvell_get_iob_memory_map**
  Returns the IOB windows configuration and the number of windows

Mandatory structures

- **iob_memory_map**
  Array that includes the configuration of the windows. Every window/entry is a struct which has 3 parameters:
  - Base address of the window
  - Size of the window
  - Target-ID of the window

Target ID options

- **0x0** = Internal configuration space
- **0x1** = MCI0
- **0x2** = PEX1_X1
- **0x3** = PEX2_X1
- **0x4** = PEX0_X4
- **0x5** = NAND flash
- **0x6** = RUNIT (NOR/SPI/BootRoom)
- **0x7** = MCI1

Example

```
struct addr_map_win iob_memory_map[] = {
        {0x00000000f7000000,    0x0000000001000000,     PEX1_TID}, /* PEX1_X1␣
↪window */
        {0x00000000f8000000,    0x0000000001000000,     PEX2_TID}, /* PEX2_X1␣
↪window */
        {0x00000000f6000000,    0x0000000001000000,     PEX0_TID}, /* PEX0_X4␣
↪window */
        {0x00000000f9000000,    0x0000000001000000,     NAND_TID}  /* NAND␣
↪window */
};
```

7.14 MediaTek 8183

MediaTek 8183 (MT8183) is a 64-bit ARM SoC introduced by MediaTek in early 2018. The chip incorporates eight cores - four Cortex-A53 little cores and Cortex-A73. Both clusters can operate at up to 2 GHz.

7.14.1 Boot Sequence

```
Boot Rom --> Coreboot --> TF-A BL31 --> Depthcharge --> Linux Kernel
```

7.14.2 How to Build

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=mt8183 DEBUG=1
```

7.15 MediaTek 8186

MediaTek 8186 (MT8186) is a 64-bit ARM SoC introduced by MediaTek in 2021. The chip incorporates eight cores - six Cortex-A55 little cores and two Cortex-A76. Cortex-A76 can operate at up to 2.05 GHz. Cortex-A55 can operate at up to 2.0 GHz.

7.15.1 Boot Sequence

```
Boot Rom --> Coreboot --> TF-A BL31 --> Depthcharge --> Linux Kernel
```

7.15.2 How to Build

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=mt8186 DEBUG=1 COREBOOT=1
```

7.16 MediaTek 8188

MediaTek 8188 (MT8188) is a 64-bit ARM SoC introduced by MediaTek in 2022. The chip incorporates eight cores - six Cortex-A55 little cores and two Cortex-A78. Cortex-A78 can operate at up to 2.6 GHz. Cortex-A55 can operate at up to 2.0 GHz.

7.16.1 Boot Sequence

```
Boot Rom --> Coreboot --> TF-A BL31 --> Depthcharge --> Linux Kernel
```

7.16.2 How to Build

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=mt8188 DEBUG=1 COREBOOT=1
```

7.17 MediaTek 8189

MediaTek 8189 (MT8189) is a 64-bit ARM SoC introduced by MediaTek in 2024. The chip incorporates eight cores - six Cortex-A55 cores, two Cortex-A78 cores. Cortex-A55 can operate at up to 2.0 GHz. Cortex-A78 can operate at up to 3.0 GHz.

7.17.1 Boot Sequence

```
Boot Rom --> Coreboot --> TF-A BL31 --> Depthcharge --> Linux Kernel
```

7.17.2 How to Build

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=mt8189 DEBUG=1 COREBOOT=1
```

7.18 MediaTek 8192

MediaTek 8192 (MT8192) is a 64-bit ARM SoC introduced by MediaTek in 2020. The chip incorporates eight cores - four Cortex-A55 little cores and Cortex-A76. Cortex-A76 can operate at up to 2.2 GHz. Cortex-A55 can operate at up to 2 GHz.

7.18.1 Boot Sequence

```
Boot Rom --> Coreboot --> TF-A BL31 --> Depthcharge --> Linux Kernel
```

7.18.2 How to Build

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=mt8192 DEBUG=1 COREBOOT=1
```

7.19 MediaTek 8195

MediaTek 8195 (MT8195) is a 64-bit ARM SoC introduced by MediaTek in 2021. The chip incorporates eight cores - four Cortex-A55 little cores and Cortex-A78. Cortex-A78 can operate at up to 2.6 GHz. Cortex-A55 can operate at up to 2.0 GHz.

7.19.1 Boot Sequence

```
Boot Rom --> Coreboot --> TF-A BL31 --> Depthcharge --> Linux Kernel
```

7.19.2 How to Build

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=mt8195 DEBUG=1 COREBOOT=1
```

7.20 MediaTek 8196

MediaTek 8196 (MT8196) is a 64-bit ARM SoC introduced by MediaTek in 2024. The chip incorporates eight cores - four Cortex-A720 cores, three Cortex-X4 cores and one Cortex-X925 core. Cortex-A720 can operate at up to 2.1 GHz. Cortex-X4 can operate at up to 2.8 GHz. Cortex-X925 can operate at up to 3.6 GHz.

7.20.1 Boot Sequence

```
Boot Rom --> Coreboot --> TF-A BL31 --> Depthcharge --> Linux Kernel
```

7.20.2 How to Build

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=mt8196 DEBUG=1 COREBOOT=1
```

7.21 NVIDIA Tegra

- **T194**

T194 has eight NVIDIA Carmel CPU cores in a coherent multi-processor configuration. The Carmel cores support the ARM Architecture version 8.2, executing both 64-bit AArch64 code, and 32-bit AArch32 code. The Carmel processors are organized as four dual-core clusters, where each cluster has a dedicated 2 MiB Level-2 unified cache. A high speed coherency fabric connects these processor complexes and allows heterogeneous multi-processing with all eight cores if required.

7.21.1 Directory structure

- plat/nvidia/tegra/common - Common code for all Tegra SoCs
- plat/nvidia/tegra/soc/txxx - Chip specific code

7.21.2 Trusted OS dispatcher

Tegra supports multiple Trusted OS'.

- Trusted Little Kernel (TLK): In order to include the 'tlkd' dispatcher in the image, pass 'SPD=tlkd' on the command line while preparing a bl31 image.
- Trusty: In order to include the 'trusty' dispatcher in the image, pass 'SPD=trusty' on the command line while preparing a bl31 image.

This allows other Trusted OS vendors to use the upstream code and include their dispatchers in the image without changing any makefiles.

These are the supported Trusted OS' by Tegra platforms.

- Tegra194: Trusty

7.21.3 Scatter files

Tegra platforms currently support scatter files and ld.S scripts. The scatter files help support ARMLINK linker to generate BL31 binaries. For now, there exists a common scatter file, plat/nvidia/tegra/scat/bl31.scat, for all Tegra SoCs. The *LINKER* build variable needs to point to the ARMLINK binary for the scatter file to be used. Tegra platforms have verified BL31 image generation with ARMCLANG (compilation) and ARMLINK (linking) for the Tegra186 platforms.

7.21.4 Preparing the BL31 image to run on Tegra SoCs

```
CROSS_COMPILE=<path-to-aarch64-gcc>/bin/aarch64-none-elf- make PLAT=tegra \
TARGET_SOC=<target-soc e.g. t194> SPD=<dispatcher e.g. trusty|tlkd>
bl31
```

Note that all Tegra platforms only support compiling with GCC or ARMCLANG. Clang is not supported. Images will compile with clang, but will not boot.

Platforms wanting to use different TZDRAM_BASE, can add `TZDRAM_BASE=<value>` to the build command line.

The Tegra platform code expects a pointer to the following platform specific structure via 'x1' register from the BL2 layer which is used by the bl31_early_platform_setup() handler to extract the TZDRAM carveout base and size for loading the Trusted OS and the UART port ID to be used. The Tegra memory controller driver programs this base/size in order to restrict NS accesses.

typedef struct plat_params_from_bl2 { /* TZ memory size / *uint64_t tzdram_size;* / TZ memory base / *uint64_t tzdram_base;* / UART port ID */ int uart_id; /* L2 ECC parity protection disable flag */ int l2_ecc_parity_prot_dis; /* SHMEM base address for storing the boot logs */ uint64_t boot_profiler_shmem_base; } plat_params_from_bl2_t;

7.21.5 Power Management

The PSCI implementation expects each platform to expose the 'power state' parameter to be used during the 'SYSTEM SUSPEND' call. The state-id field is implementation defined on Tegra SoCs and is preferably defined by tegra_def.h.

7.21.6 Tegra configs

- 'tegra_enable_l2_ecc_parity_prot': This flag enables the L2 ECC and Parity Protection bit, for Arm Cortex-A57 CPUs, during CPU boot. This flag will be enabled by Tegrs SoCs during 'Cluster power up' or 'System Suspend' exit.

7.22 NXP i.MX7 WaRP7

The Trusted Firmware-A port for the i.MX7Solo WaRP7 implements BL2 at EL3. The i.MX7S contains a BootROM with a High Assurance Boot (HAB) functionality. This functionality provides a mechanism for establishing a root-of-trust from the reset vector to the command-line in user-space.

7.22.1 Boot Flow

BootROM –> TF-A BL2 –> BL32(OP-TEE) –> BL33(U-Boot) –> Linux

In the WaRP7 port we encapsulate OP-TEE, DTB and U-Boot into a FIP. This FIP is expected and required

7.22.2 Build Instructions

We need to use a file generated by u-boot in order to generate a .imx image the BootROM will boot. It is therefore _required_ to build u-boot before TF-A and furthermore it is _recommended_ to use the mkimage in the u-boot/tools directory to generate the TF-A .imx image.

U-Boot

https://git.linaro.org/landing-teams/working/mbl/u-boot.git

```
git checkout -b rms-atf-optee-uboot linaro-mbl/rms-atf-optee-uboot
make warp7_bl33_defconfig;
make u-boot.imx arch=ARM CROSS_COMPILE=arm-linux-gnueabihf-
```

OP-TEE

https://github.com/OP-TEE/optee_os.git

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- PLATFORM=imx PLATFORM_
↪FLAVOR=mx7swarp7 ARCH=arm CFG_PAGEABLE_ADDR=0 CFG_DT_ADDR=0x83000000 CFG_NS_
↪ENTRY_ADDR=0x87800000
```

TF-A

https://github.com/ARM-software/arm-trusted-firmware.git

The following commands assume that a directory exits in the top-level TFA build directory "fiptool_images". "fiptool_images" contains

- u-boot.bin The binary output from the u-boot instructions above
- tee-header_v2.bin
- tee-pager_v2.bin
- tee-pageable_v2.bin Binary outputs from the previous OPTEE build steps

It is also assumed copy of mbedtls is available on the path path ../mbedtls
https://github.com/ARMmbed/mbedtls.git At the time of writing HEAD points to 0592ea772aee48ca1e6d9eb84eca8e143033d973

```
mkdir fiptool_images
cp /path/to/optee/out/arm-plat-imx/core/tee-header_v2.bin fiptool_images
cp /path/to/optee/out/arm-plat-imx/core/tee-pager_v2.bin fiptool_images
cp /path/to/optee/out/arm-plat-imx/core/tee-pageable_v2.bin fiptool_images

make CROSS_COMPILE=${CROSS_COMPILE} PLAT=warp7 ARCH=aarch32 ARM_ARCH_MAJOR=7 \
     ARM_CORTEX_A7=yes AARCH32_SP=optee PLAT_WARP7_UART=1 GENERATE_COT=1 \
     TRUSTED_BOARD_BOOT=1 USE_TBBR_DEFS=1 NEED_BL32=yes \
     BL32=fiptool_images/tee-header_v2.bin \
     BL32_EXTRA1=fiptool_images/tee-pager_v2.bin \
     BL32_EXTRA2=fiptool_images/tee-pageable_v2.bin \
     BL33=fiptool_images/u-boot.bin certificates all

/path/to/u-boot/tools/mkimage -n /path/to/u-boot/u-boot.cfgout -T imximage -e␣
↪0x9df00000 -d ./build/warp7/debug/bl2.bin ./build/warp7/debug/bl2.bin.imx
```

FIP

```
cp /path/to/uboot/u-boot.bin fiptool_images
cp /path/to/linux/arch/boot/dts/imx7s-warp.dtb fiptool_images

tools/cert_create/cert_create -n --rot-key "build/warp7/debug/rot_key.pem" \
           --tfw-nvctr 0 \
           --ntfw-nvctr 0 \
           --trusted-key-cert fiptool_images/trusted-key-cert.key-crt \
           --tb-fw=build/warp7/debug/bl2.bin \
           --tb-fw-cert fiptool_images/trusted-boot-fw.key-crt\
           --tos-fw fiptool_images/tee-header_v2.bin \
           --tos-fw-cert fiptool_images/tee-header_v2.bin.crt \
           --tos-fw-key-cert fiptool_images/tee-header_v2.bin.key-crt \
           --tos-fw-extra1 fiptool_images/tee-pager_v2.bin \
           --tos-fw-extra2 fiptool_images/tee-pageable_v2.bin \
           --nt-fw fiptool_images/u-boot.bin \
           --nt-fw-cert fiptool_images/u-boot.bin.crt \
           --nt-fw-key-cert fiptool_images/u-boot.bin.key-crt \
           --hw-config fiptool_images/imx7s-warp.dtb

tools/fiptool/fiptool create --tos-fw fiptool_images/tee-header_v2.bin \
          --tos-fw-extra1 fiptool_images/tee-pager_v2.bin \
          --tos-fw-extra2 fiptool_images/tee-pageable_v2.bin \
          --nt-fw fiptool_images/u-boot.bin \
          --hw-config fiptool_images/imx7s-warp.dtb \
          --tos-fw-cert fiptool_images/tee-header_v2.bin.crt \
          --tos-fw-key-cert fiptool_images/tee-header_v2.bin.key-crt \
          --nt-fw-cert fiptool_images/u-boot.bin.crt \
          --nt-fw-key-cert fiptool_images/u-boot.bin.key-crt \
          --trusted-key-cert fiptool_images/trusted-key-cert.key-crt \
          --tb-fw-cert fiptool_images/trusted-boot-fw.key-crt warp7.fip
```

7.22.3 Deploy Images

First place the WaRP7 into UMS mode in u-boot this should produce an entry in /dev like /dev/disk/by-id/usb-Linux_UMS_disk_0_WaRP7-0xf42400d3000001d4-0:0

```
=> ums 0 mmc 0
```

Next flash bl2.imx and warp7.fip

bl2.imx is flashed @ 1024 bytes warp7.fip is flash @ 1048576 bytes

```
sudo dd if=bl2.bin.imx of=/dev/disk/by-id/usb-Linux_UMS_disk_0_WaRP7-
↪0xf42400d3000001d4-0\:0 bs=512 seek=2 conv=notrunc
# Offset is 1MB 1048576 => 1048576 / 512 = 2048
sudo dd if=./warp7.fip of=/dev/disk/by-id/usb-Linux_UMS_disk_0_WaRP7-
↪0xf42400d3000001d4-0\:0 bs=512 seek=2048 conv=notrunc
```

Remember to umount the USB device pefore proceeding

```
sudo umount /dev/disk/by-id/usb-Linux_UMS_disk_0_WaRP7-0xf42400d3000001d4-0\
↪:0*
```

7.22.4 Signing BL2

A further step is to sign BL2.

The image_sign.sh and bl2_sign.csf files alluded to blow are available here.

https://github.com/bryanodonoghue/atf-code-signing

It is suggested you use this script plus the example CSF file in order to avoid hard-coding data into your CSF files.

Download both “image_sign.sh” and “bl2_sign.csf” to your arm-trusted-firmware top-level directory.

```
#!/bin/bash
SIGN=image_sign.sh
TEMP=`pwd`/temp
BL2_CSF=bl2_sign.csf
BL2_IMX=bl2.bin.imx
CST_PATH=/path/to/cst-2.3.2
CST_BIN=${CST_PATH}/linux64/cst

#Remove temp
rm -rf ${TEMP}
mkdir ${TEMP}

# Generate IMX header
/path/to/u-boot/tools/mkimage -n u-boot.cfgout.warp7 -T imximage -e␣
↪0x9df00000 -d ./build/warp7/debug/bl2.bin ./build/warp7/debug/bl2.bin.imx >
↪${TEMP}/${BL2_IMX}.log
```

(continues on next page)

(continued from previous page)

```
# Copy required items to $TEMP
cp build/warp7/debug/bl2.bin.imx ${TEMP}
cp ${CST_PATH}/keys/* ${TEMP}
cp ${CST_PATH}/crts/* ${TEMP}
cp ${BL2_CSF} ${TEMP}

# Generate signed BL2 image
./${SIGN} image_sign_mbl_binary ${TEMP} ${BL2_CSF} ${BL2_IMX} ${CST_BIN}

# Copy signed BL2 to top-level directory
cp ${TEMP}/${BL2_IMX}-signed .
cp ${BL2_RECOVER_CSF} ${TEMP}
```

The resulting bl2.bin.imx-signed can replace bl2.bin.imx in the Deploy Images section above, once done.

Suggested flow for verifying.

1. Followed all previous steps above and verify a non-secure ATF boot
2. Down the NXP Code Singing Tool
3. Generate keys
4. Program the fuses on your board
5. Replace bl2.bin.imx with bl2.bin.imx-signed
6. Verify inside u-boot that "hab_status" shows no events
7. Subsequently close your board.

If you have HAB events @ step 6 - do not lock your board.

To get a good over-view of generating keys and programming the fuses on the board read "High Assurance Boot for Dummies" by Boundary Devices.

https://boundarydevices.com/high-assurance-boot-hab-dummies/

7.23 NXP i.MX 8 Series

The i.MX 8 series of applications processors is a feature- and performance-scalable multi-core platform that includes single-, dual-, and quad-core families based on the Arm® Cortex® architecture—including combined Cortex-A72 + Cortex-A53, Cortex-A35, and Cortex-M4 based solutions for advanced graphics, imaging, machine vision, audio, voice, video, and safety-critical applications.

The i.MX8QM is with 2 Cortex-A72 ARM core, 4 Cortex-A53 ARM core and 1 Cortex-M4 system controller.

The i.MX8QX is with 4 Cortex-A35 ARM core and 1 Cortex-M4 system controller.

The System Controller (SC) represents the evolution of centralized control for system-level resources on i.MX8. The heart of the system controller is a Cortex-M4 that executes system controller firmware.

7.23.1 Boot Sequence

Bootrom –> BL31 –> BL33(u-boot) –> Linux kernel

7.23.2 How to build

Build Procedure

- Prepare AARCH64 toolchain.
- Build System Controller Firmware and u-boot firstly, and get binary images: scfw_tcm.bin and u-boot.bin
- Build TF-A

  Build bl31:

  ```
  CROSS_COMPILE=aarch64-linux-gnu- make PLAT=<Target_SoC> bl31
  ```

  Target_SoC should be “imx8qm” for i.MX8QM SoC. Target_SoC should be “imx8qx” for i.MX8QX SoC.

Deploy TF-A Images

TF-A binary(bl31.bin), scfw_tcm.bin and u-boot.bin are combined together to generate a binary file called flash.bin, the imx-mkimage tool is used to generate flash.bin, and flash.bin needs to be flashed into SD card with certain offset for BOOT ROM. The system controller firmware, u-boot and imx-mkimage will be upstreamed soon, this doc will be updated once they are ready, and the link will be posted.

7.24 NXP i.MX 8M Series

The i.MX 8M family of applications processors based on Arm Corte-A53 and Cortex-M4 cores provide high-performance computing, power efficiency, enhanced system reliability and embedded security needed to drive the growth of fast-growing edge node computing, streaming multimedia, and machine learning applications.

imx8mq is dropped in TF-A CI build due to the small OCRAM size, but still actively maintained in NXP official release.

7.24.1 Boot Sequence

Bootrom –> SPL –> BL31 –> BL33(u-boot) –> Linux kernel

7.24.2 How to build

Build Procedure

- Prepare AARCH64 toolchain.
- Build spl and u-boot firstly, and get binary images: u-boot-spl.bin, u-boot-nodtb.bin and dtb for the target board.
- Build TF-A

  Build bl31:

  ```
  CROSS_COMPILE=aarch64-linux-gnu- make PLAT=<Target_SoC> bl31
  ```

  Target_SoC should be "imx8mq" for i.MX8MQ SoC. Target_SoC should be "imx8mm" for i.MX8MM SoC. Target_SoC should be "imx8mn" for i.MX8MN SoC. Target_SoC should be "imx8mp" for i.MX8MP SoC.

Deploy TF-A Images

TF-A binary(bl31.bin), u-boot-spl.bin u-boot-nodtb.bin and dtb are combined together to generate a binary file called flash.bin, the imx-mkimage tool is used to generate flash.bin, and flash.bin needs to be flashed into SD card with certain offset for BOOT ROM. the u-boot and imx-mkimage will be upstreamed soon, this doc will be updated once they are ready, and the link will be posted.

7.24.3 TBBR Boot Sequence

When setting NEED_BL2=1 on imx8mm. We support an alternative way of boot sequence to support TBBR.

Bootrom –> SPL –> BL2 –> BL31 –> BL33(u-boot with UEFI) –> grub

This helps us to fulfill the SystemReady EBBR standard. BL2 will be in the FIT image and SPL will verify it. All of the BL3x will be put in the FIP image. BL2 will verify them. In U-boot we turn on the UEFI secure boot features so it can verify grub. And we use grub to verify linux kernel.

7.24.4 Measured Boot

When setting MEASURED_BOOT=1 on imx8mm we can let TF-A generate event logs with a DTB overlay. The overlay will be put at PLAT_IMX8M_DTO_BASE with maximum size PLAT_IMX8M_DTO_MAX_SIZE. Then in U-boot we can apply the DTB overlay and let U-boot to parse the event log and update the PCRs.

7.24.5 High Assurance Boot (HABv4)

All actively maintained platforms have a support for High Assurance Boot (HABv4), which is implemented via ROM Vector Table (RVT) API to extend the Root-of-Trust beyond the SPL. Those calls are done via SMC and are executed in EL3, with results returned back to original caller.

Note on DRAM Memory Mapping

There is a special case of mapping the DRAM: entire DRAM available on the platform is mapped into the EL3 with MT_RW attributes.

Mapping the entire DRAM allows the usage of 2MB block mapping in Level-2 Translation Table entries, which use less Page Table Entries (PTEs). If Level-3 PTE mapping is used instead then additional PTEs would be required, which leads to the increase of translation table size.

Due to the fact that the size of SRAM is limited on some platforms in the family it should rather be avoided creating additional Level-3 mapping and introduce more PTEs, hence the implementation uses Level-2 mapping which maps entire DRAM space.

The reason for the MT_RW attribute mapping scheme is the fact that the SMC API to get the status and events is called from NS world passing destination pointers which are located in DRAM. Mapping DRAM without MT_RW permissions causes those locations not to be filled, which in turn causing EL1&0 software not to receive replies.

Therefore, DRAM mapping is done with MT_RW attributes, as it is required for data exchange between EL3 and EL1&0 software.

Reference Documentation

Details on HABv4 usage and implementation could be found in following documents:

- AN4581: "i.MX Secure Boot on HABv4 Supported Devices", Rev. 4 - June 2020
- AN12263: "HABv4 RVT Guidelines and Recommendations", Rev. 1 - 06/2020
- "HABv4 API Reference Manual". This document in the part of NXP Code Signing Tool (CST) distribution.

7.25 NXP i.MX 8ULP

i.MX 8ULP is part of the ULP family with emphasis on extreme low-power techniques using the 28 nm fully depleted silicon on insulator process. Like i.MX 7ULP, i.MX 8ULP continues to be based on asymmetric architecture.

The i.MX 8ULP family of processors features NXP's advanced implementation of the dual Arm Cortex-A35 cores alongside an Arm Cortex-M33. This combined architecture enables the device to run a rich operating system (such as Linux) on the Cortex-A35 core and an RTOS (such as FreeRTOS) on the Cortex-M33 core. It also includes a Cadence Tensilica Fusion DSP for low-power audio and a HiFi4 DSP for advanced audio and machine learning applications.

The design enables clean separation between two processing domains, where each has separate power, clocking and peripheral islands, but the bus fabric of each domain is tightly integrated for efficient communication. The part is streamlined to minimize pin count, enabling small packages and simple system integration. This microprocessor is intended for applications where efficiency and simple system integration is important. i.MX8ULP Applications Processors.

7.25.1 Boot Sequence

BootROM –> SPL –> BL31 –> BL33(u-boot) –> Linux kernel

7.25.2 How to build

Build Procedure

- Prepare AARCH64 toolchain.
- Get the ELE FW image from NXP linux SDK package
- Build SPL and u-boot firstly, and get binary images: u-boot-spl.bin, u-boot.bin and dtb
- Build TF-A

  Build bl31:

  ```
  CROSS_COMPILE=aarch64-linux-gnu- make PLAT=<Target_SoC> bl31
  ```

  Target_SoC should be "imx8ulp" for i.MX8ULP SoC.

Deploy TF-A Images

TF-A binary(bl31.bin), u-boot-spl.bin u-boot.bin, ELE FW image are combined together to generate a binary file called flash.bin, the imx-mkimage tool is used to generate flash.bin, and flash.bin needs to be flashed into SD card with certain offset for BOOT ROM.

Reference Documentation

Details on how to prepare, generate & deploy the boot image be found in following documents:

- i.MX Linux User's Guide link
- i.MX Linux Reference Manual link

7.26 NXP i.MX 9 Series

Building on the market-proven i.MX 6 and i.MX 8 series, i.MX 9 series applications processors bring together higher performance applications cores, an independent MCU-like real-time domain, Energy Flex architecture, state-of-the-art security with EdgeLock® secure enclave and dedicated multi-sensory data processing engines (graphics, image, display, audio and voice). The i.MX 9 series, part of the EdgeVerse™ edge computing platform, integrates hardware neural processing units across many members of the series for acceleration of machine learning applications at the edge i.MX9 Applications Processors.

7.26.1 Boot Sequence

BootROM –> SPL –> BL31 –> BL33(u-boot) –> Linux kernel

7.26.2 How to build

Build Procedure

- Prepare AARCH64 toolchain.
- Get the ELE FW image from NXP linux SDK package
- Build SPL and u-boot firstly, and get binary images: u-boot-spl.bin, u-boot.bin and dtb
- Build TF-A

  Build bl31:

  ```
  CROSS_COMPILE=aarch64-linux-gnu- make PLAT=<Target_SoC> bl31
  ```

  Target_SoC should be "imx93" for i.MX93 SoC.

Deploy TF-A Images

TF-A binary(bl31.bin), u-boot-spl.bin u-boot.bin, ELE FW image are combined together to generate a binary file called flash.bin, the imx-mkimage tool is used to generate flash.bin, and flash.bin needs to be flashed into SD card with certain offset for BOOT ROM.

Reference Documentation

Details on how to prepare, generate & deploy the boot image be found in following documents:

- i.MX Linux User's Guide link
- i.MX Linux Reference Manual link

7.27 NXP S32G274A

S32G2 is an NXP vehicle network processor combining ASIL D safety, hardware security, high-performance real-time and application processing and network acceleration. S32G2 supports the needs of new vehicle architectures: service-oriented gateways, domain controllers, zonal processors, safety processors and more. It is equipped with 4 Cortex-A53 cores operating at 1.0GHz.

The TF-A includes support for one single S32G2-based board called S32G274ARDB2. The S32G-VNP-RDB2 is a compact, highly optimized and integrated board engineering for vehicle service-oriented gateway (SoG), domain control applications, high-performance processing, safety and security applications. More details about this board can be found at s32g274ardb2.

7.27.1 Boot Flow

```
BootROM -> BL2 (SRAM) -> BL31 (DDR) -> BL33 (DDR)
```

Warning: This boot flow is a preliminary version that will serve as a foundation for upcoming S32G2 contributions.

7.27.2 Code Locations

- Downstream TF-A: link:
- Downstream U-Boot: link
- Downstream Linux: link

7.27.3 How to build

The port currently available on the S32G274ARDB2 platform is in its initial stage. The boot process takes advantage of both SRAM and DDR memory. BootROM loads BL2 into SRAM, and BL2 is responsible for initializing DDR and loading subsequent stages (such as BL31 and BL33) into their designated memory regions. To create a bootable image, the script below should be used. This script makes use of the `mkimage` tool, which is part of the U-Boot drop for S32G274A SoCs.

> **Warning:** The `DDR_FW_BIN_PATH` parameter must point to the location of the folder containing the DDR firmware binary specific to the S32G274ARDB2 platform. This binary is essential for proper DDR initialization. It is included in the Linux BSP releases for the S32G274A-RDB board and can be obtained from https://www.nxp.com/.

```
#!/bin/bash -xe
TF_A="${TF_A:-`pwd`}"
UBOOT="${UBOOT:-${TF_A}/../u-boot}"
DDR_FW="${DDR_FW:-${TF-A}/../ddr-firmware}"
DEBUG="${DEBUG:-1}"

FIP_BASE="0x34100000"

if [ "${DEBUG}" -eq "1" ]; then
        BUILD="debug"
else
        BUILD="release"
fi

BOOT_IMAGE="build/s32g274ardb2/${BUILD}/BOOT_IMAGE.bin"
BL2_BIN="build/s32g274ardb2/${BUILD}/bl2.bin"
FIP_BIN="build/s32g274ardb2/${BUILD}/fip.bin"

# Generate bl2, bl31 and fip image
make -C "${TF_A}" -j9 'PLAT=s32g274ardb2' \
        BL33="${UBOOT}/u-boot-nodtb.bin" DEBUG="${DEBUG}" clean DDR_FW_BIN_
↪PATH="${DDR_FW}/ddr_fw_ecc_off.bin"
make -C "${TF_A}" -j9 'PLAT=s32g274ardb2' \
        BL33="${UBOOT}/u-boot-nodtb.bin" DEBUG="${DEBUG}" bl2 DDR_FW_BIN_PATH=
↪"${DDR_FW}/ddr_fw_ecc_off.bin"
make -C "${TF_A}" -j9 'PLAT=s32g274ardb2' \
        BL33="${UBOOT}/u-boot-nodtb.bin" DEBUG="${DEBUG}" fip DDR_FW_BIN_PATH=
↪"${DDR_FW}/ddr_fw_ecc_off.bin"

# Extract BL2 entry
BL2_START="0x$(poetry run memory -p s32g274ardb2 -b debug footprint | \
                        grep BL2 | awk -F'|' '{print $3}' | xargs)"
# BL2 bin file size in bytes
BL2_SIZE="$(stat -c "%s" "${BL2_BIN}")"

# Pack bl2.bin and fip.bin by ensuring that the FIP image will start at FIP_
```

(continues on next page)

(continued from previous page)

```
↪BASE
cp -vf "${BL2_BIN}" "${BOOT_IMAGE}"
dd if="${FIP_BIN}" of="${BOOT_IMAGE}" seek="$((FIP_BASE - BL2_START))" bs=1

# Build a bootable image by appending the IVT
"${UBOOT}/tools/mkimage" \
        -a "${BL2_START}" \
        -e "${BL2_START}" \
        -T s32ccimage \
        -n "${UBOOT}/u-boot-s32.cfgout" \
        -d "${BOOT_IMAGE}" \
        fip.s32
```

7.27.4 SoC Errata Workarounds

The S32G274A port of the TF-A includes compilation flags that can be used to control the workaround for the SoC. These flags are used similarly to how the *CPU Errata Workarounds* are used. The list of workarounds includes the following switches:

- `ERRATA_S32_051700`: This applies erratum ERR051700 workaround to SoCs part of the S32 Common Chassis family, and therefore it needs to be enabled for the S32G and S32R devices.

7.28 Nuvoton NPCM845X

Nuvoton NPCM845X is the Nuvoton Arbel NPCM8XX Board Management controller (BMC) SoC.

The Nuvoton Arbel NPCM845X SoC is a fourth-generation BMC. The NPCM845X computing subsystem comprises a quadcore Arm Cortex-A35 CPU.

This SoC includes secured components, i.e., bootblock stored in ROM, u-boot, OPTEE-OS, trusted-firmware-a and Linux. Every stage is measured and validated by the bootblock. This SoC was tested on the Arbel NPCM845X evaluation board.

7.28.1 How to Build

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=npcm845x all SPD=opteed
```

7.29 NXP Reference Development Platforms

7.29.1 1. NXP SoCs - Overview

The QorIQ family of ARM based SoCs that are supported on TF-A are:

1. LX2160A

- SoC Overview:

The LX2160A multicore processor, the highest-performance member of the Layerscape family, combines FinFET process technology's low power and sixteen Arm® Cortex®-A72 cores with datapath acceleration optimized for L2/3 packet processing, together with security offload, robust traffic management and quality of service.

Details about LX2160A can be found at lx2160a.

- LX2160ARDB Board:

The LX2160A reference design board provides a comprehensive platform that enables design and evaluation of the LX2160A or LX2162A processors. It comes preloaded with a board support package (BSP) based on a standard Linux kernel.

Board details can be fetched from the link: lx2160ardb.

2. LS1028A

- SoC Overview:

The Layerscape LS1028A applications processor for industrial and automotive includes a time-sensitive networking (TSN) -enabled Ethernet switch and Ethernet controllers to support converged IT and OT networks. Two powerful 64-bit Arm®v8 cores support real-time processing for industrial control and virtual machines for edge computing in the IoT. The integrated GPU and LCD controller enable Human-Machine Interface (HMI) systems with next-generation interfaces.

Details about LS1028A can be found at ls1028a.

- LS1028ARDB Board:

The LS1028A reference design board (RDB) is a computing, evaluation, and development platform that supports industrial IoT applications, human machine interface solutions, and industrial networking.

Details about LS1028A RDB board can be found at ls1028ardb.

3. LS1043A

- SoC Overview:

The Layerscape LS1043A processor is NXP's first quad-core, 64-bit Arm®-based processor for embedded networking. The LS1023A (two core version) and the LS1043A (four core version) deliver greater than 10 Gbps of performance in a flexible I/O package supporting fanless designs. This SoC is a purpose-built solution for small-form-factor networking and industrial applications with BOM optimizations for economic low layer PCB, lower cost power supply and single clock design. The new 0.9V versions of the LS1043A and LS1023A deliver addition power savings for applications such as Wireless LAN and to Power over Ethernet systems.

Details about LS1043A can be found at ls1043a.

- LS1043ARDB Board:

The LS1043A reference design board (RDB) is a computing, evaluation, and development platform that supports the Layerscape LS1043A architecture processor. The LS1043A-RDB can help shorten your time to market by providing the following features:

Memory subsystem:

- 2GByte DDR4 SDRAM (32bit bus)
- 128 Mbyte NOR flash single-chip memory
- 512 Mbyte NAND flash
- 16 Mbyte high-speed SPI flash
- SD connector to interface with the SD memory card

Ethernet:

- XFI 10G port
- QSGMII with 4x 1G ports
- Two RGMII ports

PCIe:

- PCIe2 (Lanes C) to mini-PCIe slot
- PCIe3 (Lanes D) to PCIe slot

USB 3.0: two super speed USB 3.0 type A ports

UART: supports two UARTs up to 115200 bps for console

Details about LS1043A RDB board can be found at ls1043ardb.

4. LS1046A

- SoC Overview:

The LS1046A is a cost-effective, power-efficient, and highly integrated system-on-chip (SoC) design that extends the reach of the NXP value-performance line of QorIQ communications processors. Featuring power-efficient 64-bit Arm Cortex-A72 cores with ECC-protected L1 and L2 cache memories for high reliability, running up to 1.8 GHz.

Details about LS1046A can be found at ls1046a.

- LS1046ARDB Board:

The LS1046A reference design board (RDB) is a high-performance computing, evaluation, and development platform that supports the Layerscape LS1046A architecture processor. The LS1046ARDB board supports the Layerscape LS1046A processor and is optimized to support the DDR4 memory and a full complement of high-speed SerDes ports.

Details about LS1046A RDB board can be found at ls1046ardb.

- LS1046AFRWY Board:

The LS1046A Freeway board (FRWY) is a high-performance computing, evaluation, and development platform that supports the LS1046A architecture processor capable of support more than 32,000 CoreMark performance. The FRWY-LS1046A board supports the LS1046A processor, onboard DDR4 memory, multiple Gigabit Ethernet, USB3.0 and M2_Type_E interfaces for Wi-Fi, FRWY-LS1046A-AC includes the Wi-Fi card.

Details about LS1046A FRWY board can be found at ls1046afrwy.

5. LS1088A

- SoC Overview:

The LS1088A family of multicore communications processors combines up to and eight Arm Cortex-A53 cores with the advanced, high-performance data path and network peripheral interfaces required for wireless access points, networking infrastructure, intelligent edge access, including virtual customer premise equipment (vCPE) and high-performance industrial applications.

Details about LS1088A can be found at ls1088a.

- LS1088ARDB Board:

The LS1088A reference design board provides a comprehensive platform that enables design and evaluation of the product (LS1088A processor). This RDB comes pre-loaded with a board support package (BSP) based on a standard Linux kernel.

Details about LS1088A RDB board can be found at ls1088ardb.

1.1. Table of supported boot-modes by each platform & platform that needs FIP-DDR:

PLAT	BOOT_MODE							fip_ddr_needed
	sd	qspi	nor	nand	emmc	flexspi_nor	flexspi_nand	
lx2160ardb	yes				yes	yes		yes
ls1028ardb	yes				yes	yes		no
ls1043ardb	yes		yes	yes				no
ls1046ardb	yes	yes			yes			no
ls1046afrwy	yes	yes						no
ls1088ardb	yes	yes						no

1.2. Boot Sequence

```
+                              Secure World        |      Normal World
+ EL0                                              |
+                                                  |
+ EL1                              BL32(Tee OS)    |      kernel
+                                   ^  |           |         ^
+                                   |  |           |         |
+ EL2                               |  |           |      BL33(u-boot)
+                                   |  |           |        ^
+                                   |  v           |       /
```

(continues on next page)

(continued from previous page)

```
+ EL3          BootROM --> BL2 --> BL31 ---------------/
+
```

1.3. Boot Sequence with FIP-DDR

```
+                             Secure World         |      Normal World
+ EL0                                              |
+                                                  |
+ EL1                fip-ddr     BL32(Tee OS)      |      kernel
+                      ^ |          ^ |            |         ^
+                      | |          | |            |         |
+ EL2                  | |          | |            |      BL33(u-boot)
+                      | |          | |            |        ^
+                      | v          | v            |       /
+ EL3     BootROM --> BL2 -----> BL31 ---------------/
+
```

1.4. DDR Memory Layout

NXP Platforms divide DRAM into banks:

- DRAM0 Bank: Maximum size of this bank is fixed to 2GB, DRAM0 size is defined in platform_def.h if it is less than 2GB.
- DRAM1 ~ DRAMn Bank: Greater than 2GB belongs to DRAM1 and following banks, and size of DRAMn Bank varies for one platform to others.

The following diagram is default DRAM0 memory layout in which secure memory is at top of DRAM0.

```
high  +-----------------------------------------------+
      |                                               |
      |   Secure EL1 Payload Shared Memory (2 MB)     |
      |                                               |
      +-----------------------------------------------+
      |                                               |
      |            Secure Memory (64 MB)              |
      |                                               |
      +-----------------------------------------------+
      |                                               |
      |             Non Secure Memory                 |
      |                                               |
low   +-----------------------------------------------+
```

7.29.2 2. How to build

2.1. Code Locations

- OP-TEE: link
- U-Boot: link
- RCW: link
- ddr-phy-binary: Required by platforms that need fip-ddr. link
- cst: Required for TBBR. link

2.2. Build Procedure

- Fetch all the above repositories into local host.
- Prepare AARCH64 toolchain and set the environment variable "CROSS_COMPILE".

```
export CROSS_COMPILE=.../bin/aarch64-linux-gnu-
```

- Build RCW. Refer README from the respective cloned folder for more details.
- Build u-boot and OPTee firstly, and get binary images: u-boot.bin and tee.bin. For u-boot you can use the <platform>_tfa_defconfig for build.
- Copy/clone the repo "ddr-phy-binary" to the tfa directory for platform needing ddr-fip.
- Below are the steps to build TF-A images for the supported platforms.

2.2.1. Compilation steps without BL32

BUILD BL2:

-To compile

```
make PLAT=$PLAT \
BOOT_MODE=<platform_supported_boot_mode> \
RCW=$RCW_BIN \
pbl
```

BUILD FIP:

```
make PLAT=$PLAT \
BOOT_MODE=<platform_supported_boot_mode> \
RCW=$RCW_BIN \
BL33=$UBOOT_SECURE_BIN \
pbl \
fip
```

2.2.2. Compilation steps with BL32

BUILD BL2:

-To compile

```
make PLAT=$PLAT \
BOOT_MODE=<platform_supported_boot_mode> \
RCW=$RCW_BIN \
BL32=$TEE_BIN SPD=opteed\
pbl
```

BUILD FIP:

```
make PLAT=$PLAT \
BOOT_MODE=<platform_supported_boot_mode> \
RCW=$RCW_BIN \
BL32=$TEE_BIN SPD=opteed\
BL33=$UBOOT_SECURE_BIN \
pbl \
fip
```

2.2.3. BUILD fip-ddr (Mandatory for certain platforms, refer table above):

-To compile additional fip-ddr for selected platforms(Refer above table if the platform needs fip-ddr).

```
make PLAT=<platform_name> fip-ddr
```

7.29.3 3. Deploy ATF Images

Note: The size in the standard uboot commands for copy to nor, qspi, nand or sd should be modified based on the binary size of the image to be copied.

- Deploy ATF images on flexspi-Nor or QSPI flash Alt Bank from U-Boot prompt.

  – Commands to flash images for bl2_xxx.pbl and fip.bin

  Notes: ls1028ardb has no flexspi-Nor Alt Bank, so use "sf probe 0:0" for current bank.

```
tftp 82000000  $path/bl2_xxx.pbl;

i2c mw 66 50 20;sf probe 0:1; sf erase 0 +$filesize; sf write 0x82000000␣
↪0x0 $filesize;

tftp 82000000  $path/fip.bin;
i2c mw 66 50 20;sf probe 0:1; sf erase 0x100000 +$filesize; sf write␣
↪0x82000000 0x100000 $filesize;
```

  – Next step is valid for platform where FIP-DDR is needed.

```
tftp 82000000  $path/ddr_fip.bin;
i2c mw 66 50 20;sf probe 0:1; sf erase 0x800000 +$filesize; sf write␣
↪0x82000000 0x800000 $filesize;
```

– Then reset to alternate bank to boot up ATF.

Command for lx2160a, ls1088a and ls1028a platforms:

```
qixisreset altbank;
```

Command for ls1046a platforms:

```
cpld reset altbank;
```

- Deploy ATF images on SD/eMMC from U-Boot prompt. – file_size_in_block_sizeof_512 = (Size_of_bytes_tftp / 512)

```
    mmc dev <idx>; (idx = 1 for eMMC; idx = 0 for SD)

    tftp 82000000  $path/bl2_<sd>_or_<emmc>.pbl;
    mmc write 82000000 8 <file_size_in_block_sizeof_512>;

    tftp 82000000  $path/fip.bin;
    mmc write 82000000 0x800 <file_size_in_block_sizeof_512>;

--  Next step is valid for platform that needs FIP-DDR.
```

```
tftp 82000000  $path/ddr_fip.bin;
mmc write 82000000 0x4000 <file_size_in_block_sizeof_512>;
```

– Then reset to sd/emmc to boot up ATF from sd/emmc as boot-source.

Command for lx2160A, ls1088a and ls1028a platforms:

```
qixisreset <sd or emmc>;
```

Command for ls1043a and ls1046a platform:

```
cpld reset <sd or emmc>;
```

- Deploy ATF images on IFC nor flash from U-Boot prompt.

```
tftp 82000000  $path/bl2_nor.pbl;
protect off 64000000 +$filesize; erase 64000000 +$filesize; cp.b␣
↪82000000 64000000 $filesize;

tftp 82000000  $path/fip.bin;
protect off 64100000 +$filesize; erase 64100000 +$filesize; cp.b␣
↪82000000 64100000 $filesize;
```

– Then reset to alternate bank to boot up ATF.

Command for ls1043a platform:

```
cpld reset altbank;
```

- Deploy ATF images on IFC nand flash from U-Boot prompt.

```
tftp 82000000  $path/bl2_nand.pbl;
nand erase 0x0 $filesize; nand write 82000000 0x0 $filesize;

tftp 82000000  $path/fip.bin;
nand erase 0x100000 $filesize;nand write 82000000 0x100000 $filesize;
```

– Then reset to nand flash to boot up ATF.

Command for ls1043a platform:

```
cpld reset nand;
```

7.29.4 4. Trusted Board Boot:

For TBBR, the binary name changes:

Boot Type	BL2	FIP	FIP-DDR
Normal Boot	bl2_<boot_mode>.pbl	fip.bin	ddr_fip.bin
TBBR Boot	bl2_<boot_mode>_sec.pbl	fip.bin	ddr_fip_sec.bin

Refer nxp-ls-tbbr.rst for detailed user steps.

7.29.5 Steps to blow fuses on NXP LS SoC:

- Enable POVDD – Refer board GSG(Getting Started Guide) for the steps to enable POVDD. – Once the POVDD is enabled, make sure to set variable POVDD_ENABLE := yes, in the platform.mk.

	Platform	Jumper	Switch	LED to Verify	Through GPIO Pin (=number)
1.	lx2160ardb	J9			no
2.	lx2160aqds	J35			no
3.	lx2162aqds	J35	SW9[4] = 1	D15	no

- SFP registers to be written to:

	Platform	OTPMKR0..OTPMKR7	SRKHR0..SRKHR7
1.	lx2160ardb/lx2160aqds/lx2162aqds	0x1e80234..0x1e80250	0x1e80254..0x1e80270

- At U-Boot prompt, verify that SNVS register - HPSR, whether OTPMK was written, already:

	Platform	OTPMK_ZERO_BIT(=value)	SNVS_HPSR_REG
1.	lx2160ardb/lx2160aqds/lx2162aqds	1 (=1 means not blown, =0 means blown)	0x01E90014

From u-boot prompt:

– Check for the OTPMK.

```
  md $SNVS_HPSR_REG

Command Output:
    01e90014: 88000900

    In case it is read as 00000000, then read this register␣
↪using jtag (in development mode only through CW tap).
                 +0        +4        +8        +C
    [0x01E90014] 88000900

    Note: OTPMK_ZERO_BIT is 1, indicating that the OTPMK is not␣
↪blown.
```

---Check for the SRK Hash. .. code:: shell

md $SRKHR0 0x10

Command Output:

01e80254: 00000000 00000000 00000000 00000000 …………….. 01e80264: 00000000 00000000 00000000 00000000 ……………

Note: Zero means that SRK hash is not blown.

- If not blown, then from the U-Boot prompt, using following commands: – Provision the OTPMK.

```
mw.l $OTPMKR0  <OTMPKR_0_32Bit_val>
mw.l $OTPMKR1  <OTMPKR_1_32Bit_val>
mw.l $OTPMKR2  <OTMPKR_2_32Bit_val>
mw.l $OTPMKR3  <OTMPKR_3_32Bit_val>
mw.l $OTPMKR4  <OTMPKR_4_32Bit_val>
mw.l $OTPMKR5  <OTMPKR_5_32Bit_val>
mw.l $OTPMKR6  <OTMPKR_6_32Bit_val>
mw.l $OTPMKR7  <OTMPKR_7_32Bit_val>
```

– Provision the SRK Hash.

```
  mw.l $SRKHR0  <SRKHR_0_32Bit_val>
  mw.l $SRKHR1  <SRKHR_1_32Bit_val>
  mw.l $SRKHR2  <SRKHR_2_32Bit_val>
  mw.l $SRKHR3  <SRKHR_3_32Bit_val>
  mw.l $SRKHR4  <SRKHR_4_32Bit_val>
  mw.l $SRKHR5  <SRKHR_5_32Bit_val>
  mw.l $SRKHR6  <SRKHR_6_32Bit_val>
  mw.l $SRKHR7  <SRKHR_7_32Bit_val>

Note: SRK Hash should be carefully written keeping in mind the␣
↪SFP Block Endianness.
```

- At U-Boot prompt, verify that SNVS registers for OTPMK are correctly written:

  – Check for the OTPMK.

```
  md $SNVS_HPSR_REG

Command Output:
    01e90014: 80000900

    OTPMK_ZERO_BIT is zero, indicating that the OTPMK is blown.

    Note: In case it is read as 00000000, then read this register␣
↪using jtag (in development mode only through CW tap).
```

```
  md $OTPMKR0 0x10

Command Output:
    01e80234: ffffffff ffffffff ffffffff ffffffff    ................
    01e80244: ffffffff ffffffff ffffffff ffffffff    ................

    Note: OTPMK will never be visible in plain.
```

  – Check for the SRK Hash. For example, if following SRK hash is written:

    SFP SRKHR0 = fdc2fed4 SFP SRKHR1 = 317f569e SFP SRKHR2 = 1828425c SFP SRKHR3 = e87b5cfd SFP SRKHR4 = 34beab8f SFP SRKHR5 = df792a70 SFP SRKHR6 = 2dff85e1 SFP SRKHR7 = 32a29687,

    then following would be the value on dumping SRK hash.

```
  md $SRKHR0 0x10

Command Output:
    01e80254: d4fec2fd 9e567f31 5c422818 fd5c7be8    ....1.V..(B\
↪.{\.
    01e80264: 8fabbe34 702a79df e185ff2d 8796a232    4....y*p-...
↪2...

    Note: SRK Hash is visible in plain based on the SFP Block␣
↪Endianness.
```

- Caution: Donot proceed to the next step, until you are sure that OTPMK and SRKH are correctly blown from above steps. – After the next step, there is no turning back. – Fuses will be burnt, which cannot be undo.
- Write SFP_INGR[INST] with the PROGFB(0x2) instruction to blow the fuses. – User need to save the SRK key pair and OTPMK Key forever, to continue using this board.

	Platform	SFP_INGR_REG | SFP_WRITE_DATE_FRM_MIRROR_REG_TO_FUSE
1.	lx2160ardb/lx2160aqds	0x01E80020 | 0x2

```
md $SFP_INGR_REG  $SFP_WRITE_DATE_FRM_MIRROR_REG_TO_FUSE
```

- On reset, if the SFP register were read from u-boot, it will show the following: – Check for the OTPMK.

```
  md $SNVS_HPSR_REG

Command Output:
    01e90014: 80000900

    In case it is read as 00000000, then read this register␣
↪using jtag (in development mode only through CW tap).
                 +0        +4        +8        +C
    [0x01E90014] 80000900

    Note: OTPMK_ZERO_BIT is zero, indicating that the OTPMK is␣
↪blown.
```

```
  md $OTPMKR0 0x10

Command Output:
    01e80234: ffffffff ffffffff ffffffff ffffffff    ............
↪....
    01e80244: ffffffff ffffffff ffffffff ffffffff    ............
↪....

    Note: OTPMK will never be visible in plain.
```

– SRK Hash

```
  md $SRKHR0 0x10

Command Output:
    01e80254: d4fec2fd 9e567f31 5c422818 fd5c7be8    ....1.V..(B\
↪.{\.
    01e80264: 8fabbe34 702a79df e185ff2d 8796a232    4....y*p-...
↪2...

    Note: SRK Hash is visible in plain based on the SFP Block␣
↪Endianness.
```

7.29.6 Second method to do the fuse provsioning:

This method is used for quick way to provision fuses. Typically used by those who needs to provision number of boards.

- Enable POVDD: – Refer the table above to enable POVDD.

  Note: If GPIO Pin supports enabling POVDD, it can be done through the below input_fuse_file.

  – Once the POVDD is enabled, make sure to set variable POVDD_ENABLE := yes, in the platform.mk.

- User need to populate the "input_fuse_file", corresponding to the platform for:

  – OTPMK – SRKH

  Table of fuse provisioning input file for every supported platform:

	Platform	FUSE_PROV_FILE
1.	lx2160ardb/lx2160aqds/lx2162aqds	${CST_DIR}/input_files/gen_fusescr/ls2088_1088/input_fuse_file

- Create the TF-A binary with FUSE_PROG=1.

```
make PLAT=$PLAT FUSE_PROG=1\
  BOOT_MODE=<platform_supported_boot_mode> \
  RCW=$RCW_BIN \
  BL32=$TEE_BIN SPD=opteed\
  BL33=$UBOOT_SECURE_BIN \
  pbl \
  fip \
  fip_fuse \
  FUSE_PROV_FILE=../../apps/security/cst/input_files/gen_fusescr/
↪ls2088_1088/input_fuse_file
```

- Deployment: – Refer the nxp-layerscape.rst for deploying TF-A images. – Deploying fip_fuse.bin:

  For Flexspi-Nor:

```
  tftp 82000000  $path/fuse_fip.bin;
  i2c mw 66 50 20;sf probe 0:0; sf erase 0x880000 +$filesize; sf␣
↪write 0x82000000 0x880000 $filesize;

For SD or eMMC [file_size_in_block_sizeof_512 = (Size_of_bytes_
↪tftp / 512)]:
```

```
tftp 82000000  $path/fuse_fip.bin;
mmc write 82000000 0x4408 <file_size_in_block_sizeof_512>;
```

- Valiation:

	Platform	Error_Register | Error_Register_Address
1.	lx2160ardb/lx2160aqds/lx2162aqds	DCFG scratch 4 register | 0x01EE020C

At the U-Boot prompt, check DCFG scratch 4 register for any error.

```
  md $Error_Register_Address 1

Command Ouput:
    01ee020c: 00000000

Note:
 - 0x00000000 shows no error, then fuse provisioning is successful.
 - For non-zero value, refer the code header file ".../drivers/nxp/
↪sfp/sfp_error_codes.h"
```

7.29.7 NXP Platforms:

TRUSTED_BOARD_BOOT option can be enabled by specifying TRUSTED_BOARD_BOOT=1 on command line during make.

Bare-Minimum Preparation to run TBBR on NXP Platforms:

- OTPMK(One Time Programable Key) needs to be burnt in fuses. – It is the 256 bit key that stores a secret value used by the NXP SEC 4.0 IP in Trusted or Secure mode.

  Note: It is primarily for the purpose of decrypting additional secrets stored in system non-volatile memory.

  – NXP CST tool gives an option to generate it.

  Use the below command from directory 'cst', with correct options.

  ```
  ./gen_otpmk_drbg
  ```

- SRKH (Super Root Key Hash) needs to be burnt in fuses. – It is the 256 bit hash of the list of the public keys of the SRK key pair. – NXP CST tool gives an option to generate the RSA key pair and its hash.

  Use the below command from directory 'cst', with correct options.

  ```
  ./gen_keys
  ```

Refer fuse frovisioning readme 'nxp-ls-fuse-prov.rst' for steps to blow these keys.

Two options are provided for TRUSTED_BOARD_BOOT:

7.29.8 Option 1: CoT using X 509 certificates

- This CoT is as provided by ARM.
- Enable this option by setting `NXP_TBBR_USE_X509=1` on the make command line.
- To generate CSF header, path of CST repository needs to be specified as CST_DIR
- CSF header is embedded to each of the BL2 image.
- GENERATE_COT=1 adds the tool 'cert_create' to the build environment to generate: – X509 Certificates as (.crt) files. – X509 Pem key file as (.pem) files.
- SAVE_KEYS=1 saves the keys and certificates, if GENERATE_COT=1. – For this to work, file name for cert and keys are provided as part of compilation or build command.

  — default file names will be used, incase not provided as part compilation or build command.
  — default folder 'BUILD_PLAT' will be used to store them.
- ROTPK for x.509 certificates is generated and embedded in bl2.bin and verified as part of CoT by Boot ROM during secure boot.
- Compilation steps:

All Images

```
make PLAT=$PLAT TRUSTED_BOARD_BOOT=1 GENERATE_COT=1 CST_DIR=$CST_DIR_
↪PATH \
NXP_TBBR_USE_X509=1 \
BOOT_MODE=<platform_supported_boot_mode> \
RCW=$RCW_BIN \
BL32=$TEE_BIN SPD=opteed\
BL33=$UBOOT_SECURE_BIN \
pbl \
fip
```

Additional FIP_DDR Image (For NXP platforms like lx2160a)

```
 make PLAT=$PLAT TRUSTED_BOARD_BOOT=1 GENERATE_COT=1 \
 NXP_TBBR_USE_X509=1 fip_ddr

Note: make target 'fip_ddr' should never be combine with other make␣
↪target 'fip', 'pbl' & 'bl2'.
```

7.29.9 Option 2: CoT using NXP CSF headers.

- CSF header is embedded to each of the BL31, BL32 and BL33 image.
- This is the default flow when `NXP_TBBR_USE_X509` is not set or is `0`.
- To generate CSF header, path of CST repository needs to be specified as CST_DIR
- Default input files for CSF header generation is added in this repo.
- Default input file requires user to generate RSA key pair named – srk.pri, and – srk.pub, and add them in ATF repo. – These keys can be generated using gen_keys tool of CST.
- To change the input file , user can use the options BL33_INPUT_FILE, BL32_INPUT_FILE, BL31_INPUT_FILE
- There are 2 paths in secure boot flow : – Development Mode (sb_en in RCW = 1, SFP->OSPR, ITS = 0)

  — In this flow , even on ROTPK comparison failure, flow would continue. — However SNVS is transitioned to non-secure state

  – Production mode (SFP->OSPR, ITS = 1)

  — Any failure is fatal failure
- Compilation steps:

All Images

```
make PLAT=$PLAT TRUSTED_BOARD_BOOT=1 CST_DIR=$CST_DIR_PATH \
BOOT_MODE=<platform_supported_boot_mode> \
RCW=$RCW_BIN \
BL32=$TEE_BIN SPD=opteed\
BL33=$UBOOT_SECURE_BIN \
pbl \
fip
```

Additional FIP_DDR Image (For NXP platforms like lx2160a)

```
make PLAT=$PLAT TRUSTED_BOARD_BOOT=1 CST_DIR=$CST_DIR_PATH fip_ddr
```

- Compilation Steps with build option for generic image processing filters to prepend CSF header: – Generic image processing filters to prepend CSF header

  BL32_INPUT_FILE = < file name> BL33_INPUT_FILE = <file name>

  ```
  make PLAT=$PLAT TRUSTED_BOARD_BOOT=1 CST_DIR=$CST_DIR_PATH \
  BOOT_MODE=<platform_supported_boot_mode> \
  RCW=$RCW_BIN \
  BL32=$TEE_BIN SPD=opteed\
  BL33=$UBOOT_SECURE_BIN \
  BL33_INPUT_FILE = <ip file> \
  BL32_INPUT_FILE = <ip_file> \
  BL31_INPUT_FILE = <ip file> \
  ```

(continues on next page)

(continued from previous page)

```
pbl \
fip
```

Deploy ATF Images

Same steps as mentioned in the readme “nxp-layerscape.rst”.

Verification to check if Secure state is achieved:

	Platform	SNVS_HPSR_REG	SYS_SECURE_BIT(=value)	SYSTEM_SECURE_CONFIG_BIT(=value)	SSM_STATE
1.	lx2160ardb or lx2160aqds or lx2162aqds	0x01E90014	15 (= 1, BootROM Booted)	14-12 (= 010 means Intent to Secure, (= 000 Unsecure)	11-8 (=1111 means secure boot) (=1011 means Non-secure Boot)

- Production mode (SFP->OSPR, ITS = 1) – Linux prompt will successfully come. if the TBBR is successful.

  — Else, Linux boot will be successful.

  – For secure-boot status, read SNVS Register $SNVS_HPSR_REG from u-boot prompt:

```
  md $SNVS_HPSR_REG

Command Output:
    1e90014: 8000AF00

    In case it is read as 00000000, then read this register using jtag (in development mode only through CW tap).
                 +0        +4        +8        +C
    [0x01E90014] 8000AF00
```

- Development Mode (sb_en in RCW = 1, SFP->OSPR, ITS = 0) – Refer the SoC specific table to read the register to interpret whether the secure boot is achieved or not. – Using JTAG (in development environment only, using CW tap):

  — For secure-boot status, read SNVS Register $SNVS_HPSR_REG

```
  ccs::display_regs 86 0x01E90014 4 0 1

Command Output:
    Using the SAP chain position number 86, following is the output.

                 +0        +4        +8        +C
```

(continues on next page)

(continued from previous page)

```
    [0x01E90014] 8000AF00

    Note: Chain position number will vary from one SoC to other␣
↪SoC.
```

- Interpretation of the value:

  – 0xA indicates BootROM booted, with intent to secure. – 0xF = secure boot, as SSM_STATE.

This chapter holds documentation related to NXP reference development platforms. It includes details on image flashing, fuse provisioning and trusted board boot-up.

7.30 Poplar

Poplar is the first development board compliant with the 96Boards Enterprise Edition TV Platform specification.

The board features the Hi3798C V200 with an integrated quad-core 64-bit Arm Cortex A53 processor and high performance Mali T720 GPU, making it capable of running any commercial set-top solution based on Linux or Android.

It supports a premium user experience with up to H.265 HEVC decoding of 4K video at 60 frames per second.

```
SOC Hisilicon Hi3798CV200
CPU Quad-core Arm Cortex-A53 64 bit
DRAM DDR3/3L/4 SDRAM interface, maximum 32-bit data width 2 GB
USB Two USB 2.0 ports One USB 3.0 ports
CONSOLE USB-micro port for console support
ETHERNET 1 GBe Ethernet
PCIE One PCIe 2.0 interfaces
JTAG 8-Pin JTAG
EXPANSION INTERFACE Linaro 96Boards Low Speed Expansion slot
DIMENSION Standard 160×120 mm 96Boards Enterprice Edition form factor
WIFI 802.11AC 2*2 with Bluetooth
CONNECTORS One connector for Smart Card One connector for TSI
```

At the start of the boot sequence, the bootROM executes the so called l-loader binary whose main role is to change the processor state to 64bit mode. This must happen prior to invoking Trusted Firmware-A:

```
l-loader --> Trusted Firmware-A --> u-boot
```

7.30.1 How to build

Code Locations

- Trusted Firmware-A: link
- l-loader: link
- u-boot: link

Build Procedure

- Fetch all the above 3 repositories into local host. Make all the repositories in the same ${BUILD_PATH}.
- Prepare the AARCH64 toolchain.
- Build u-boot using poplar_defconfig

```
make CROSS_COMPILE=aarch64-linux-gnu- poplar_defconfig
make CROSS_COMPILE=aarch64-linux-gnu-
```

- Build atf providing the previously generated u-boot.bin as the BL33 image

```
make CROSS_COMPILE=aarch64-linux-gnu-  all fip SPD=none PLAT=poplar
BL33=u-boot.bin
```

- **Build l-loader (generated the final fastboot.bin)**
  1. copy the atf generated files fip.bin and bl1.bin to l-loader/atf/
  2. export ARM_TRUSTED_FIRMWARE=${ATF_SOURCE_PATH)
  3. make

7.30.2 Install Procedure

- Copy l-loader/fastboot.bin to a FAT partition on a USB pen drive.
- Plug the USB pen drive to any of the USB2 ports
- Power the board while keeping S3 pressed (usb_boot)

The system will boot into a u-boot shell which you can then use to write the working firmware to eMMC.

7.30.3 Boot trace

```
Bootrom start
Boot Media: eMMC
Decrypt auxiliary code ...OK

lsadc voltage min: 000000FE, max: 000000FF, aver: 000000FE, index: 00000000

Entry boot auxiliary code

Auxiliary code - v1.00
DDR code - V1.1.2 20160205
Build: Mar 24 2016 - 17:09:44
Reg Version:  v134
Reg Time:     2016/03/18 09:44:55
Reg Name:     hi3798cv2dmb_hi3798cv200_ddr3_2gbyte_8bitx4_4layers.reg

Boot auxiliary code success
Bootrom success

LOADER:  Switched to aarch64 mode
LOADER:  Entering ARM TRUSTED FIRMWARE
LOADER:  CPU0 executes at 0x000ce000

INFO:    BL1: 0xe1000 - 0xe7000 [size = 24576]
NOTICE:  Booting Trusted Firmware
NOTICE:  BL1: v1.3(debug):v1.3-372-g1ba9c60
NOTICE:  BL1: Built : 17:51:33, Apr 30 2017
INFO:    BL1: RAM 0xe1000 - 0xe7000
INFO:    BL1: Loading BL2
INFO:    Loading image id=1 at address 0xe9000
INFO:    Image id=1 loaded at address 0xe9000, size = 0x5008
NOTICE:  BL1: Booting BL2
INFO:    Entry point address = 0xe9000
INFO:    SPSR = 0x3c5
NOTICE:  BL2: v1.3(debug):v1.3-372-g1ba9c60
NOTICE:  BL2: Built : 17:51:33, Apr 30 2017
INFO:    BL2: Loading BL31
INFO:    Loading image id=3 at address 0x129000
INFO:    Image id=3 loaded at address 0x129000, size = 0x8038
INFO:    BL2: Loading BL33
INFO:    Loading image id=5 at address 0x37000000
INFO:    Image id=5 loaded at address 0x37000000, size = 0x58f17
NOTICE:  BL1: Booting BL31
INFO:    Entry point address = 0x129000
INFO:    SPSR = 0x3cd
INFO:    Boot bl33 from 0x37000000 for 364311 Bytes
NOTICE:  BL31: v1.3(debug):v1.3-372-g1ba9c60
NOTICE:  BL31: Built : 17:51:33, Apr 30 2017
INFO:    BL31: Initializing runtime services
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0x37000000
```

(continues on next page)

(continued from previous page)

```
INFO:    SPSR = 0x3c9


U-Boot 2017.05-rc2-00130-gd2255b0 (Apr 30 2017 - 17:51:28 +0200)poplar

Model: HiSilicon Poplar Development Board
BOARD: Hisilicon HI3798cv200 Poplar
DRAM:  1 GiB
MMC:   Hisilicon DWMMC: 0
In:    serial@f8b00000
Out:   serial@f8b00000
Err:   serial@f8b00000
Net:   Net Initialization Skipped
No ethernet found.

Hit any key to stop autoboot:  0
starting USB...
USB0:   USB EHCI 1.00
scanning bus 0 for devices... 1 USB Device(s) found
USB1:   USB EHCI 1.00
scanning bus 1 for devices... 4 USB Device(s) found
       scanning usb for storage devices... 1 Storage Device(s) found
       scanning usb for ethernet devices... 1 Ethernet Device(s) found

USB device 0:
    Device 0: Vendor: SanDisk Rev: 1.00 Prod: Cruzer Blade
            Type: Removable Hard Disk
            Capacity: 7632.0 MB = 7.4 GB (15630336 x 512)
... is now current device
Scanning usb 0:1...
=>
```

7.31 QEMU virt Armv8-A

Trusted Firmware-A (TF-A) implements the EL3 firmware layer for QEMU virt Armv8-A. BL1 is used as the BootROM, supplied with the -bios argument. When QEMU starts all CPUs are released simultaneously, BL1 selects a primary CPU to handle the boot and the secondaries are placed in a polling loop to be released by normal world via PSCI.

BL2 edits the Flattened Device Tree, FDT, generated by QEMU at run-time to add a node describing PSCI and also enable methods for the CPUs.

If `ARM_LINUX_KERNEL_AS_BL33` is set to 1 then this FDT will be passed to BL33 via register x0, as expected by a Linux kernel. This allows a Linux kernel image to be booted directly as BL33 rather than using a bootloader.

An ARM64 defconfig v5.5 Linux kernel is known to boot, FDT doesn't need to be provided as it's generated by QEMU.

Current limitations:

- Only cold boot is supported

7.31.1 Getting non-TF images

QEMU_EFI.fd can be downloaded from http://snapshots.linaro.org/components/kernel/leg-virt-tianocore-edk2-upstream/latest/QEMU-KERNEL-AARCH64/RELEASE_GCC5/QEMU_EFI.fd

or, can be built as follows:

```
git clone https://github.com/tianocore/edk2.git
cd edk2
git submodule update --init
make -C BaseTools
source edksetup.sh
export GCC5_AARCH64_PREFIX=aarch64-linux-gnu-
build -a AARCH64 -t GCC5 -p ArmVirtPkg/ArmVirtQemuKernel.dsc
```

Then, you will get Build/ArmVirtQemuKernel-AARCH64/DEBUG_GCC5/FV/QEMU_EFI.fd

Please note you do not need to use GCC 5 in spite of the environment variable GCC5_AARCH64_PREFIX.

The rootfs can be built by using Buildroot as follows:

```
git clone git://git.buildroot.net/buildroot.git
cd buildroot
make qemu_aarch64_virt_defconfig
utils/config -e BR2_TARGET_ROOTFS_CPIO
utils/config -e BR2_TARGET_ROOTFS_CPIO_GZIP
make olddefconfig
make
```

Then, you will get output/images/rootfs.cpio.gz.

7.31.2 Booting via semi-hosting option

Boot binaries, except BL1, are primarily loaded via semi-hosting so all binaries has to reside in the same directory as QEMU is started from. This is conveniently achieved with symlinks the local names as:

- bl2.bin -> BL2
- bl31.bin -> BL31
- bl33.bin -> BL33 (QEMU_EFI.fd)
- Image -> linux/arch/arm64/boot/Image

To build:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=qemu
```

To start (QEMU v5.0.0):

```
qemu-system-aarch64 -nographic -machine virt,secure=on -cpu cortex-a57  \
    -kernel Image                           \
    -append "console=ttyAMA0,38400 keep_bootcon"   \
    -initrd rootfs.cpio.gz -smp 2 -m 1024 -bios bl1.bin   \
    -d unimp -semihosting-config enable,target=native
```

7.31.3 Booting via flash based firmware

An alternate approach to deploy a full system stack on QEMU is to load the firmware via a secure flash device. This involves concatenating `bl1.bin` and `fip.bin` to create a boot ROM that is flashed onto secure FLASH0 with the `-bios` option.

For example, to test the following firmware stack:

- BL32 - `bl32.bin` -> `tee-header_v2.bin`
- BL32 Extra1 - `bl32_extra1.bin` -> `tee-pager_v2.bin`
- BL32 Extra2 - `bl32_extra2.bin` -> `tee-pageable_v2.bin`
- BL33 - `bl33.bin` -> `QEMU_EFI.fd` (EDK II)
- `Image` -> linux/arch/arm64/boot/Image

1. Compile TF-A

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=qemu BL32=bl32.bin \
    BL32_EXTRA1=bl32_extra1.bin BL32_EXTRA2=bl32_extra2.bin \
    BL33=bl33.bin BL32_RAM_LOCATION=tdram SPD=opteed all fip
```

Or, alternatively, to build with TBBR enabled, as well as, BL31 and BL32 encrypted with test key:

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=qemu BL32=bl32.bin \
    BL32_EXTRA1=bl32_extra1.bin BL32_EXTRA2=bl32_extra2.bin \
    BL33=bl33.bin BL32_RAM_LOCATION=tdram SPD=opteed all fip \
    TRUSTED_BOARD_BOOT=1 GENERATE_COT=1 DECRYPTION_SUPPORT=aes_gcm \
    FW_ENC_STATUS=0 ENCRYPT_BL31=1 ENCRYPT_BL32=1
```

2. Concatenate `bl1.bin` and `fip.bin` to create the boot ROM

```
dd if=build/qemu/release/bl1.bin of=flash.bin bs=4096 conv=notrunc
dd if=build/qemu/release/fip.bin of=flash.bin seek=64 bs=4096␣
↪conv=notrunc
```

3. Launch QEMU

```
qemu-system-aarch64 -nographic -machine virt,secure=on
    -cpu cortex-a57  -kernel Image   \
    -append 'console=ttyAMA0,38400 keep_bootcon'  \
    -initrd rootfs.cpio.gz -smp 2 -m 1024 -bios flash.bin   \
    -d unimp
```

The `-bios` option abstracts the loading of raw bare metal binaries into flash or ROM memory. QEMU loads the binary into the region corresponding to the hardware's entrypoint, from which the binary is executed upon a platform "reset". In addition to this, it places the information about the kernel provided with option `-kernel`, and the RamDisk provided with `-initrd`, into the firmware configuration `fw_cfg`. In this setup, EDK II is responsible for extracting and launching these from `fw_cfg`.

Note: QEMU may be launched with or without ACPI (`-acpi`/`-no-acpi`). In either case, ensure that the kernel build options are aligned with the parameters passed to QEMU.

7.31.4 Running QEMU in OpenCI

Linaro's continuous integration platform OpenCI supports running emulated tests on QEMU. The tests are kicked off on Jenkins and deployed through the Linaro Automation and Validation Architecture LAVA.

There are a set of Linux boot tests provided in OpenCI. They rely on prebuilt binaries for UEFI, the kernel, root file system, as well as, any other TF-A dependencies, and are run as part of the OpenCI TF-A daily job. To run them manually, a builder job may be triggered with the test configuration `qemu-boot-tests`.

You may see the following warning repeated several times in the boot logs:

```
pflash_write: Write to buffer emulation is flawed
```

Please ignore this as it is an unresolved issue in QEMU, it is an internal QEMU warning that logs flawed use of "write to buffer".

Note: For more information on how to trigger jobs in OpenCI, please refer to Linaro's CI documentation, which explains how to trigger a manual job.

7.32 QEMU SBSA Target

Trusted Firmware-A (TF-A) implements the EL3 firmware layer for QEMU SBSA Armv8-A. While running Qemu from command line, we need to supply two Flash images. First Secure BootRom is supplied by -pflash argument. This Flash image is made by EDK2 build system by composing BL1 and FIP. Second parameter for Qemu is responsible for Non-secure rom which also given with -pflash argument and contains of UEFI and EFI variables (also made by EDK2 build system). Semihosting is not used

When QEMU starts all CPUs are released simultaneously, BL1 selects a primary CPU to handle the boot and the secondaries are placed in a polling loop to be released by normal world via PSCI.

BL2 edits the FDT, generated by QEMU at run-time to add a node describing PSCI and also enable methods for the CPUs.

Current limitations:

- Only cold boot is supported

To build TF-A:

```
git clone https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git tfa
cd tfa
export CROSS_COMPILE=aarch64-none-elf-
make PLAT=qemu_sbsa all fip
```

To build TF-A with BL32 and SPM enabled(StandaloneMM as a Secure Payload):

```
git clone https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git tfa
cd tfa
export CROSS_COMPILE=aarch64-none-elf-
make PLAT=qemu_sbsa BL32=../STANDALONE_MM.fd SPM_MM=1 EL3_EXCEPTION_
↪HANDLING=1 all fip
```

Images will be placed at build/qemu_sbsa/release (bl1.bin and fip.bin). Need to copy them into top directory for EDK2 compilation.

```
cp build/qemu_sbsa/release/bl1.bin ../
cp build/qemu_sbsa/release/fip.bin ../
```

Those images cannot be used by itself (no semihosing support). Flash images are built by EDK2 build system, refer to edk2-platform repo for full build instructions.

```
git clone https://github.com/tianocore/edk2-platforms.git
Platform/Qemu/SbsaQemu/Readme.md
```

7.33 Qualcomm Platforms

7.33.1 Qualcomm Chrome Targets

Trusted Firmware-A (TF-A) implements the EL3 firmware layer for QTI SC7180, SC7280 and QCS615.

Boot Trace

Bootrom –> BL1/BL2 –> BL31 –> BL33 –> Linux kernel

BL1/2 and BL33 can currently be supplied from Coreboot + Depthcharge

How to build

Code Locations

- Trusted Firmware-A: link

Build Procedure

QTI SoC expects TF-A's BL31 to get integrated with other boot software Coreboot, so only bl31.elf need to get build from the TF-A repository.

The build command looks like

> make CROSS_COMPILE=aarch64-linux-gnu- PLAT=sc7180 COREBOOT=1

Update value of CROSS_COMPILE argument with your cross-compilation toolchain. Update the value of PLAT to be either of sc7180, sc7280_chrome or qcs615

Additional QTISECLIB_PATH=<path to qtiseclib> can be added in build command. if QTISECLIB_PATH is not added in build command stub implementation of qtiseclib is picked. qtiseclib with stub implementation doesn't boot device. This was added to satisfy compilation.

QTISELIB for SC7180 is available at link QTISELIB for SC7280 is available at link QTISECLIB for QCS615 is not available yet and currently compile with stubs only.

7.33.2 Qualcomm MSM8916

The MSM8916 platform port in TF-A supports multiple similar Qualcomm SoCs:

System-on-Chip (SoC)	TF-A Platform	Application CPU	Supports
Snapdragon 410 (MSM8x16, APQ8016(E)) (DragonBoard 410c)	`PLAT=msm8916`	4x ARM Cortex-A53	AArch64/AArch32
Snapdragon 615 (MSM8x39, APQ8039)	`PLAT=msm8939`	4x ARM Cortex-A53 4x ARM Cortex-A53	AArch64/AArch32
Snapdragon 210 (MSM8x09, APQ8009)	`PLAT=msm8909`	4x ARM Cortex-A7	AArch32 only
Snapdragon X5 Modem (MDM9x07)	`PLAT=mdm9607`	1x ARM Cortex-A7	AArch32 only

It provides a minimal, community-maintained EL3 firmware and PSCI implementation, based on information from the public Snapdragon 410E Technical Reference Manual combined with a lot of trial and error to actually make it work.

Note: Unlike the *QTI SC7180/SC7280* ports, this port does **not** make use of a proprietary binary components (QTISECLIB). It is fully open-source but therefore limited to publicly documented hardware components.

Functionality

The TF-A port is much more minimal compared to the original firmware and therefore expects the non-secure world (e.g. Linux) to manage more hardware, such as the SMMUs and all remote processors (RPM, WCNSS, Venus, Modem). Everything except modem is currently functional with a slightly modified version of mainline Linux.

> **Warning:** This port is **not secure**. There is no special secure memory and the used DRAM is available from both the non-secure and secure worlds. Unfortunately, the hardware used for memory protection is not described in the APQ8016E documentation.

The port is primarily intended as a minimal PSCI implementation (without a separate secure world) where this limitation is not a big problem. Booting secondary CPU cores (PSCI `CPU_ON`) is supported. Basic CPU core power management (`CPU_SUSPEND`) is functional but still work-in-progress and will be added later once ready.

Boot Flow

BL31 (AArch64) or BL32/SP_MIN (AArch32) replaces the original `tz` firmware in the boot flow:

```
Boot ROM (PBL) -> SBL -> BL31 (EL3) -> U-Boot (EL2) -> Linux (EL2)
```

After initialization the normal world starts at a fixed entry address in EL2/HYP mode, configured using `PRELOADED_BL33_BASE`. At runtime, it is expected that the normal world bootloader was already loaded into RAM by a previous firmware component (usually SBL) and that it is capable of running in EL2/HYP mode.

U-Boot for DragonBoard 410c is recommended if possible. The original Little Kernel-based bootloader from Qualcomm does not support EL2/HYP, but can be booted using an additional shim loader such as tfalkstub.

Build

It is possible to build for either AArch64 or AArch32. Some platforms use 32-bit CPUs that only support AArch32 (see table above). For all others AArch64 is the preferred build option.

AArch64 (BL31)

Setup the cross compiler for AArch64 and build BL31 for one of the platforms in the table above:

```
$ make CROSS_COMPILE=aarch64-none-elf- PLAT=...
```

The BL31 ELF image is generated in `build/$PLAT/release/bl31/bl31.elf`.

AArch32 (BL32/SP_MIN)

Setup the cross compiler for AArch32 and build BL32 with SP_MIN for one of the platforms in the table above:

```
$ make CROSS_COMPILE=arm-none-eabi- PLAT=... ARCH=aarch32 AARCH32_SP=sp_min
```

The BL32 ELF image is generated in `build/$PLAT/release/bl32/bl32.elf`.

Build Options

Some options can be changed at build time by adding them to the make command line:

- `QTI_UART_NUM`: Number of UART controller to use for debug output and crash reports. This must be the same UART as used by earlier boot firmware since the UART controller does not get fully initialized at the moment. Defaults to the usual debug UART used for the platform (see `platform.mk`).
- `QTI_RUNTIME_UART`: By default (`0`) the UART is only used for the boot process and critical crashes. If set to `1` it is also used for runtime messages. Note that this option can only be used if the UART is reserved in the normal world and the necessary clocks remain enabled.

The memory region used for the different firmware components is not fixed and can be changed on the make command line. The default values match the addresses used by the original firmware (see `platform.mk`):

- `PRELOADED_BL33_BASE`: The entry address for the normal world. Usually refers to the first bootloader (e.g. U-Boot).
- `BL31_BASE`: Base address for the BL31 firmware component. Must point to a 64K-aligned memory region with at least 128 KiB space that is permanently reserved in the normal world.
- `BL32_BASE`: Base address for the BL32 firmware component.
  - **AArch32:** BL32 is used in place of BL31, so the option is equivalent to `BL31_BASE`.
  - **AArch64:** Secure-EL1 Payload. Defaults to using 128 KiB of space directly after BL31. For testing only, the port is primarily intended as a minimal PSCI implementation without a separate secure world.

Installation

The ELF image must be "signed" before flashing it, even if the board has secure boot disabled. In this case the signature does not provide any security, but it provides the firmware with required metadata.

The DragonBoard 410c does not have secure boot enabled by default. In this case you can simply sign the ELF image using a randomly generated key. You can use e.g. qtestsign:

```
$ ./qtestsign.py tz build/msm8916/release/bl31/bl31.elf
```

Then install the resulting `build/msm8916/release/bl31/bl31-test-signed.mbn` to the `tz` partition on the device. BL31 should be running after a reboot.

Note: On AArch32 the ELF image is called `bl32.elf`. The installation procedure is identical.

Warning: Do not flash incorrectly signed firmware on devices that have secure boot enabled! Make sure that you have a way to recover the board in case of problems (e.g. using EDL).

Boot Trace

AArch64 (BL31)

BL31 prints some lines on the debug console, which will usually look like this (with `DEBUG=1`, otherwise only the `NOTICE` lines are shown):

```
...
S - DDR Frequency, 400 MHz
NOTICE:  BL31: v2.6(debug):v2.6
NOTICE:  BL31: Built : 20:00:00, Dec 01 2021
INFO:    BL31: Platform setup start
INFO:    ARM GICv2 driver initialized
INFO:    BL31: Platform setup done
INFO:    BL31: Initializing runtime services
INFO:    BL31: cortex_a53: CPU workaround for 819472 was applied
INFO:    BL31: cortex_a53: CPU workaround for 824069 was applied
INFO:    BL31: cortex_a53: CPU workaround for 826319 was applied
INFO:    BL31: cortex_a53: CPU workaround for 827319 was applied
INFO:    BL31: cortex_a53: CPU workaround for 835769 was applied
INFO:    BL31: cortex_a53: CPU workaround for disable_non_temporal_hint was␣
↪applied
INFO:    BL31: cortex_a53: CPU workaround for 843419 was applied
INFO:    BL31: cortex_a53: CPU workaround for 1530924 was applied
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0x8f600000
INFO:    SPSR = 0x3c9

U-Boot 2021.10 (Dec 01 2021 - 20:00:00 +0000)
Qualcomm-DragonBoard 410C
...
```

AArch32 (BL32/SP_MIN)

BL32/SP_MIN prints some lines on the debug console, which will usually look like this (with `DEBUG=1`, otherwise only the `NOTICE` lines are shown):

```
...
S - DDR Frequency, 400 MHz
NOTICE:  SP_MIN: v2.8(debug):v2.8
NOTICE:  SP_MIN: Built : 23:03:31, Mar 31 2023
INFO:    SP_MIN: Platform setup start
INFO:    ARM GICv2 driver initialized
INFO:    SP_MIN: Platform setup done
INFO:    SP_MIN: Initializing runtime services
INFO:    BL32: cortex_a53: CPU workaround for 819472 was applied
INFO:    BL32: cortex_a53: CPU workaround for 824069 was applied
INFO:    BL32: cortex_a53: CPU workaround for 826319 was applied
INFO:    BL32: cortex_a53: CPU workaround for 827319 was applied
INFO:    BL32: cortex_a53: CPU workaround for disable_non_temporal_hint was␣
↪applied
INFO:    SP_MIN: Preparing exit to normal world
INFO:    Entry point address = 0x86400000
INFO:    SPSR = 0x1da
Android Bootloader - UART_DM Initialized!!!
[0] welcome to lk
...
```

7.33.3 Dragonwing RB3 Gen 2 development platform

Trusted Firmware-A (TF-A) platform port for Kodiak SoC based RB3Gen2 platform. For details about RB3Gen2 platform, refer to this link

Boot flow

The boot flow on RB3Gen2 with TF-A BL2 replacing the TZ firmware binary and FIP payload replcacing UEFI binary as:

```
PBL (ROM) -> XBL -> TF-A BL2 -> TF-A BL31 -> BL33 -> Linux kernel
                                    |
                                    --> OP-TEE as BL32
```

BL33 can either be edk2, U-Boot, Coreboot + Depthcharge etc.

Summary:

- PBL authenticates and loads XBL on on-chip RAM (IMEM) which is responsible to initializing DRAM and has the flash storage driver.
- XBL/XBL_SEC authenticates and loads TF-A BL2 from *tz_a/b* partition and FIP payload from *uefi_a/b* partition on flash to DRAM.

- TF-A BL2 parses the FIP payload on DRAM and loads BL31, BL32 and BL33 components in corresponding locations as specified in FIP metadata.

How to build

Steps to build TF-A BL2 and FIP payload:

```
$ make CROSS_COMPILE=aarch64-none-elf- PLAT=rb3gen2 SPD=opteed \
    QTISECLIB_PATH=<qtiseclib-path> BL32=<path-to-optee-bin> \
    BL33=<path-to-os-bootloader-bin> fip all

$ ./tools/qti/generate_fip_elf.sh build/rb3gen2/release/fip.bin \
    0x9fc00000
```

QTISELIB for Kodiak/SC7280 is available at link

Note that the bl2.elf generated here is required to be signed as TZ image with QTI signing involved. Also, the fip.elf generated above is signed using qtestsign which just puts the OEM test signature to allow it to work on devices in non-secure mode. In secure mode, it is required to be signed using sectools v2

How to flash

Generate a Yocto BSP image for RB3Gen2 following instructions from meta-qcom here

It will generate a tar ball in the build folder for RB3Gen2 something like: *qcom-multimedia-image-qcs6490-rb3gen2-core-kit.rootfs.qcomflash.tar.gz*. Untar it and replace *tz.mbn* with *bl2.mbn* and *uefi.elf* with *fip.elf* that you have generated by yourself.

Then flash the board following the meta-qcom instructions here

Boot Trace

Boot logs on RB3Gen2:

```
Format: Log Type - Time(microsec) - Message - Optional Info
Log Type: B - Since Boot(Power On Reset),  D - Delta,  S - Statistic
<snip-XBL-logs>
NOTICE:  BL2: v2.13.0(release):v2.13.0-825-gc17b22110
NOTICE:  BL2: Built : 15:54:07, Sep 25 2025
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: v2.13.0(release):v2.13.0-825-gc17b22110
NOTICE:  BL31: Built : 15:58:59, Sep 25 2025
QTISECLIB [0019dc805]QC_IMAGE_VERSION_STRING=QTISECLIB.CB.1.0-00069-
↪SC7280AAAAAANAZT-1
QTISECLIB [001a0817c]IMAGE_VARIANT_STRING=SC7180LC
QTISECLIB [001a221d8]OEM_IMAGE_VERSION_STRING=crm-ubuntu54
QTISECLIB [001a40244]QC_IMAGE_TS_STRING=23:42:01, Aug 31 2022
I/TC:
I/TC: OP-TEE version: 4.7.0-79-g8a8fab26c (gcc version 14.2.0 (Ubuntu 14.2.0-
```

(continues on next page)

(continued from previous page)

```
↪19ubuntu2)) #7 Thu Sep 25 10:28:22 UTC 2025 aarch64
I/TC: WARNING: This OP-TEE configuration might be insecure!
I/TC: WARNING: Please check https://optee.readthedocs.io/en/latest/
↪architecture/porting_guidelines.html
I/TC: Primary CPU initializing
I/TC: Primary CPU switching to normal world boot
<snip>
```

7.33.4 Dragonwing IQ-9075 development platform

Trusted Firmware-A (TF-A) platform port for Lemans SoC based EVK platform also known as IQ-9075 as the marketing name. For details about IQ-9075 platform, refer to this link

Boot flow

Similar to *Dragonwing RB3 Gen 2 development platform*.

How to build

Steps to build TF-A BL2 and FIP payload:

```
$ make CROSS_COMPILE=aarch64-none-elf- PLAT=lemans_evk SPD=opteed \
    QTISECLIB_PATH=<qtiseclib-path> BL32=<path-to-optee-bin> \
    BL33=<path-to-os-bootloader-bin> fip all

$ ./tools/qti/generate_fip_elf.sh build/lemans_evk/release/fip.bin \
    0xaf000000
```

QTISECLIB for lemans is still work under progress, the current port allows to boot OP-TEE OS.

Note that the bl2.elf generated here is required to be signed as TZ image with QTI signing involved. Also, the fip.elf generated above is signed using qtestsign which just puts the OEM test signature to allow it to work on devices in non-secure mode. In secure mode, it is required to be signed using sectools v2

How to flash

Generate a Yocto BSP image for iq-9075-evk following instructions from meta-qcom here

It will generate a tar ball in the build folder for iq-9075-evk something like: *qcom-multimedia-image-iq-9075-evk.rootfs.qcomflash.tar.gz*. Untar it and replace *tz.mbn* with *bl2.mbn* and *uefi.elf* with *fip.elf* that you have generated by yourself.

Then flash the board following the meta-qcom instructions here

Boot Trace

Boot logs on lemans_evk:

```
Format: Log Type - Time(microsec) - Message - Optional Info
Log Type: B - Since Boot(Power On Reset),  D - Delta,  S - Statistic
S - QC_IMAGE_VERSION_STRING=BOOT.MXF.1.0.c1-00430-KODIAKLA-1
S - IMAGE_VARIANT_STRING=SocLeMansLAA
S - OEM_IMAGE_VERSION_STRING=258af405f4c7
<snip-XBL-logs>
NOTICE:  BL2: v2.14.0(release):v2.13.0-1873-g0c07fd50a
NOTICE:  BL2: Built : 17:52:29, Feb 17 2026
NOTICE:  BL2: Booting BL31
NOTICE:  BL31: v2.14.0(release):v2.13.0-1873-g0c07fd50a
NOTICE:  BL31: Built : 17:52:29, Feb 17 2026
ERROR:   Please use QTISECLIB_PATH while building TF-A
ERROR:   Please refer docs/plat/qti/* for more details.
I/TC:
I/TC: OP-TEE version: 4.9.0-1-gc6cb871cc (gcc version 11.4.0 (Ubuntu 11.4.0-
↪1ubuntu1~22.04)) #1 Tue Feb 17 12:22:27 UTC 2026 aarch64
I/TC: WARNING: This OP-TEE configuration might be insecure!
I/TC: WARNING: Please check https://optee.readthedocs.io/en/latest/
↪architecture/porting_guidelines.html
I/TC: Primary CPU initializing
I/TC: Platform Qualcomm: Flavor lemans
I/TC: Primary CPU switching to normal world boot
PANIC at PC : 0x000000001c200d70
```

7.34 Raspberry Pi 3

The Raspberry Pi 3 is an inexpensive single-board computer that contains four Arm Cortex-A53 cores.

The following instructions explain how to use this port of the TF-A with the default distribution of Raspbian because that's the distribution officially supported by the Raspberry Pi Foundation. At the moment of writing this, the officially supported kernel is a AArch32 kernel. This doesn't mean that this port of TF-A can't boot a AArch64 kernel. The Linux tree fork maintained by the Foundation can be compiled for AArch64 by following the steps in *AArch64 kernel build instructions*.

IMPORTANT NOTE: This port isn't secure. All of the memory used is DRAM, which is available from both the Non-secure and Secure worlds. This port shouldn't be considered more than a prototype to play with and implement elements like PSCI to support the Linux kernel.

7.34.1 Design

The SoC used by the Raspberry Pi 3 is the Broadcom BCM2837. It is a SoC with a VideoCore IV that acts as primary processor (and loads everything from the SD card) and is located between all Arm cores and the DRAM. Check the Raspberry Pi 3 documentation for more information.

This explains why it is possible to change the execution state (AArch64/AArch32) depending on a few files on the SD card. We only care about the cases in which the cores boot in AArch64 mode.

The rules are simple:

- If a file called `kernel8.img` is located on the `boot` partition of the SD card, it will load it and execute in EL2 in AArch64. Basically, it executes a default AArch64 stub at address **0x0** that jumps to the kernel.
- If there is also a file called `armstub8.bin`, it will load it at address **0x0** (instead of the default stub) and execute it in EL3 in AArch64. All the cores are powered on at the same time and start at address **0x0**.

This means that we can use the default AArch32 kernel provided in the official Raspbian distribution by renaming it to `kernel8.img`, while TF-A and anything else we need is in `armstub8.bin`. This way we can forget about the default bootstrap code. When using a AArch64 kernel, it is only needed to make sure that the name on the SD card is `kernel8.img`.

Ideally, we want to load the kernel and have all cores available, which means that we need to make the secondary cores work in the way the kernel expects, as explained in *Secondary cores*. In practice, a small bootstrap is needed between TF-A and the kernel.

To get the most out of a AArch32 kernel, we want to boot it in Hypervisor mode in AArch32. This means that BL33 can't be in EL2 in AArch64 mode. The architecture specifies that AArch32 Hypervisor mode isn't present when AArch64 is used for EL2. When using a AArch64 kernel, it should simply start in EL2.

Placement of images

The file `armstub8.bin` contains BL1 and the FIP. It is needed to add padding between them so that the addresses they are loaded to match the ones specified when compiling TF-A. This is done automatically by the build system.

The device tree block is loaded by the VideoCore loader from an appropriate file, but we can specify the address it is loaded to in `config.txt`.

The file `kernel8.img` contains a kernel image that is loaded to the address specified in `config.txt`. The Linux kernel tree has information about how a AArch32 Linux kernel image is loaded in `Documentation/arm/Booting`:

```
The zImage may also be placed in system RAM and called there.  The
kernel should be placed in the first 128MiB of RAM.  It is recommended
that it is loaded above 32MiB in order to avoid the need to relocate
prior to decompression, which will make the boot process slightly
faster.
```

There are no similar restrictions for AArch64 kernels, as specified in the file `Documentation/arm64/booting.txt`.

This means that we need to avoid the first 128 MiB of RAM when placing the TF-A images (and specially the first 32 MiB, as they are directly used to place the uncompressed AArch32 kernel image. This way, both AArch32 and AArch64 kernels can be placed at the same address.

In the end, the images look like the following diagram when placed in memory. All addresses are Physical Addresses from the point of view of the Arm cores. Again, note that this is all just part of the same DRAM that goes from **0x00000000** to **0x3F000000**, it just has different names to simulate a real secure platform!

```
0x00000000 +-----------------+
           |       ROM       | BL1
0x00020000 +-----------------+
           |       FIP       |
0x00200000 +-----------------+
           |                 |
           |       ...       |
           |                 |
0x01000000 +-----------------+
           |       DTB       | (Loaded by the VideoCore)
           +-----------------+
           |                 |
           |       ...       |
           |                 |
0x02000000 +-----------------+
           |     Kernel      | (Loaded by the VideoCore)
           +-----------------+
           |                 |
           |       ...       |
           |                 |
0x10000000 +-----------------+
           |   Secure SRAM   | BL2, BL31
0x10100000 +-----------------+
           |   Secure DRAM   | BL32 (Secure payload)
0x11000000 +-----------------+
           | Non-secure DRAM | BL33
           +-----------------+
           |                 |
           |       ...       |
           |                 |
0x3F000000 +-----------------+
           |       I/O       |
0x40000000 +-----------------+
```

The area between **0x10000000** and **0x11000000** has to be manually protected so that the kernel doesn't use it. The current port tries to modify the live DTB to add a memreserve region that reserves the previously mentioned area.

If this is not possible, the user may manually add `memmap=16M$256M` to the command line passed to the kernel in `cmdline.txt`. See the *Setup SD card* instructions to see how to do it. This system is strongly discouraged.

The last 16 MiB of DRAM can only be accessed by the VideoCore, that has different mappings than the Arm cores in which the I/O addresses don't overlap the DRAM. The memory reserved to be used by the VideoCore is always placed at the end of the DRAM, so this space isn't wasted.

Considering the 128 MiB allocated to the GPU and the 16 MiB allocated for TF-A, there are 880 MiB available for Linux.

Boot sequence

The boot sequence of TF-A is the usual one except when booting an AArch32 kernel. In that case, BL33 is booted in AArch32 Hypervisor mode so that it can jump to the kernel in the same mode and let it take over that privilege level. If BL33 was running in EL2 in AArch64 (as in the default bootflow of TF-A) it could only jump to the kernel in AArch32 in Supervisor mode.

The Linux kernel tree has instructions on how to jump to the Linux kernel in `Documentation/arm/Booting` and `Documentation/arm64/booting.txt`. The bootstrap should take care of this.

This port support a direct boot of the Linux kernel from the firmware (as a BL33 image). Alternatively, U-Boot or other bootloaders may be used.

Secondary cores

This port of the Trusted Firmware-A supports `PSCI_CPU_ON`, `PSCI_SYSTEM_RESET` and `PSCI_SYSTEM_OFF`. The last one doesn't really turn the system off, it simply reboots it and asks the VideoCore firmware to keep it in a low power mode permanently.

The kernel used by Raspbian doesn't have support for PSCI, so it is needed to use mailboxes to trap the secondary cores until they are ready to jump to the kernel. This mailbox is located at a different address in the AArch32 default kernel than in the AArch64 kernel.

Kernels with PSCI support can use the PSCI calls instead for a cleaner boot.

Also, this port of TF-A has another Trusted Mailbox in Shared BL RAM. During cold boot, all secondary cores wait in a loop until they are given given an address to jump to in this Mailbox (`bl31_warm_entrypoint`).

Once BL31 has finished and the primary core has jumped to the BL33 payload, it has to call `PSCI_CPU_ON` to release the secondary CPUs from the wait loop. The payload then makes them wait in another waitloop listening from messages from the kernel. When the primary CPU jumps into the kernel, it will send an address to the mailbox so that the secondary CPUs jump to it and are recognised by the kernel.

7.34.2 Build Instructions

To boot a AArch64 kernel, only the AArch64 toolchain is required.

To boot a AArch32 kernel, both AArch64 and AArch32 toolchains are required. The AArch32 toolchain is needed for the AArch32 bootstrap needed to load a 32-bit kernel.

The build system concatenates BL1 and the FIP so that the addresses match the ones in the memory map. The resulting file is `armstub8.bin`, located in the build folder (e.g. `build/rpi3/debug/armstub8.bin`). To know how to use this file, follow the instructions in *Setup SD card*.

The following build options are supported:

- RPI3_BL33_IN_AARCH32: This port can load a AArch64 or AArch32 BL33 image. By default this option is 0, which means that TF-A will jump to BL33 in EL2 in AArch64 mode. If set to 1, it will jump to BL33 in Hypervisor in AArch32 mode.
- PRELOADED_BL33_BASE: Used to specify the address of a BL33 binary that has been preloaded by any other system than using the firmware. BL33 isn't needed in the build command line if this option is used. Specially useful because the file kernel8.img can be loaded anywhere by modifying the file config.txt. It doesn't have to contain a kernel, it could have any arbitrary payload.
- RPI3_DIRECT_LINUX_BOOT: Disabled by default. Set to 1 to enable the direct boot of the Linux kernel from the firmware. Option RPI3_PRELOADED_DTB_BASE is mandatory when the direct Linux kernel boot is used. Options PRELOADED_BL33_BASE will most likely be needed as well because it is unlikely that the kernel image will fit in the space reserved for BL33 images. This option can be combined with RPI3_BL33_IN_AARCH32 in order to boot a 32-bit kernel. The only thing this option does is to set the arguments in registers x0-x3 or r0-r2 as expected by the kernel.
- RPI3_PRELOADED_DTB_BASE: Auxiliary build option needed when using RPI3_DIRECT_LINUX_BOOT=1. This option allows to specify the location of a DTB in memory.
- RPI3_RUNTIME_UART: Indicates whether the UART should be used at runtime or disabled. -1 (default) disables the runtime UART. Any other value enables the default UART (currently UART1) for runtime messages.
- RPI3_USE_UEFI_MAP: Set to 1 to build ATF with the altername memory mapping required for an UEFI firmware payload. These changes are needed to be able to run Windows on ARM64. This option, which is disabled by default, results in the following memory mappings:

```
0x00000000 +-----------------+
           |       ROM       | BL1
0x00010000 +-----------------+
           |       DTB       | (Loaded by the VideoCore)
0x00020000 +-----------------+
           |       FIP       |
0x00030000 +-----------------+
           |                 |
           |  UEFI PAYLOAD   |
           |                 |
0x00200000 +-----------------+
           |   Secure SRAM   | BL2, BL31
0x00300000 +-----------------+
           |   Secure DRAM   | BL32 (Secure payload)
0x00400000 +-----------------+
           |                 |
           |                 |
           | Non-secure DRAM | BL33
           |                 |
           |                 |
0x01000000 +-----------------+
           |                 |
           |       ...       |
           |                 |
```

(continues on next page)

(continued from previous page)

```
0x3F000000 +-----------------+
           |       I/O       |
```

- BL32: This port can load and run OP-TEE. The OP-TEE image is optional. Please use the code from here. Build the Trusted Firmware with option BL32=tee-header_v2.bin BL32_EXTRA1=tee-pager_v2.bin BL32_EXTRA2=tee-pageable_v2.bin to put the binaries into the FIP.

  > **Warning:** If OP-TEE is used it may be needed to add the following options to the Linux command line so that the USB driver doesn't use FIQs: dwc_otg.fiq_enable=0 dwc_otg.fiq_fsm_enable=0 dwc_otg.nak_holdoff=0. This will unfortunately reduce the performance of the USB driver. It is needed when using Raspbian, for example.

- TRUSTED_BOARD_BOOT: This port supports TBB. Set this option to 1 to enable it. In order to use TBB, you might want to set GENERATE_COT=1 to let the contents of the FIP automatically signed by the build process. The ROT key will be generated and output to rot_key.pem in the build directory. It is able to set ROT_KEY to your own key in PEM format.
- ENABLE_STACK_PROTECTOR: Disabled by default. It uses the hardware RNG of the board.

The following is not currently supported:

- AArch32 for TF-A itself.
- EL3_PAYLOAD_BASE: The reason is that you can already load anything to any address by changing the file armstub8.bin, so there's no point in using TF-A in this case.

Building the firmware for kernels that don't support PSCI

This is the case for the 32-bit image of Raspbian, for example. 64-bit kernels always support PSCI, but they may not know that the system understands PSCI due to an incorrect DTB file.

First, clone and compile the 32-bit version of the Raspberry Pi 3 TF-A bootstrap. Choose the one needed for the architecture of your kernel.

Then compile TF-A. For a 32-bit kernel, use the following command line:

```
CROSS_COMPILE=aarch64-linux-gnu- make PLAT=rpi3          \
RPI3_BL33_IN_AARCH32=1                                   \
BL33=../rpi3-arm-tf-bootstrap/aarch32/el2-bootstrap.bin
```

For a 64-bit kernel, use this other command line:

```
CROSS_COMPILE=aarch64-linux-gnu- make PLAT=rpi3          \
BL33=../rpi3-arm-tf-bootstrap/aarch64/el2-bootstrap.bin
```

However, enabling PSCI support in a 64-bit kernel is really easy. In the repository Raspberry Pi 3 TF-A bootstrap there is a patch that can be applied to the Linux kernel tree maintained by the Raspberry Pi foundation.

It modifes the DTS to tell the kernel to use PSCI. Once this patch is applied, follow the instructions in *AArch64 kernel build instructions* to get a working 64-bit kernel image and supporting files.

Building the firmware for kernels that support PSCI

For a 64-bit kernel:

```
CROSS_COMPILE=aarch64-linux-gnu- make PLAT=rpi3                          \
PRELOADED_BL33_BASE=0x02000000                                           \
RPI3_PRELOADED_DTB_BASE=0x01000000                                       \
RPI3_DIRECT_LINUX_BOOT=1
```

For a 32-bit kernel:

```
CROSS_COMPILE=aarch64-linux-gnu- make PLAT=rpi3                          \
PRELOADED_BL33_BASE=0x02000000                                           \
RPI3_PRELOADED_DTB_BASE=0x01000000                                       \
RPI3_DIRECT_LINUX_BOOT=1                                                 \
RPI3_BL33_IN_AARCH32=1
```

7.34.3 AArch64 kernel build instructions

The following instructions show how to install and run a AArch64 kernel by using a SD card with the default Raspbian install as base. Skip them if you want to use the default 32-bit kernel.

Note that this system won't be fully 64-bit because all the tools in the filesystem are 32-bit binaries, but it's a quick way to get it working, and it allows the user to run 64-bit binaries in addition to 32-bit binaries.

1. Clone the Linux tree fork maintained by the Raspberry Pi Foundation. To speed things up, do a shallow clone of the desired branch.

```
git clone --depth=1 -b rpi-4.18.y https://github.com/raspberrypi/linux
cd linux
```

2. Configure and compile the kernel. Adapt the number after `-j` so that it is 1.5 times the number of CPUs in your computer. This may take some time to finish.

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- bcmrpi3_defconfig
make -j 6 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
```

3. Copy the kernel image and the device tree to the SD card. Replace the path by the corresponding path in your computers to the `boot` partition of the SD card.

```
cp arch/arm64/boot/Image /path/to/boot/kernel8.img
cp arch/arm64/boot/dts/broadcom/bcm2710-rpi-3-b.dtb /path/to/boot/
cp arch/arm64/boot/dts/broadcom/bcm2710-rpi-3-b-plus.dtb /path/to/boot/
```

4. Install the kernel modules. Replace the path by the corresponding path to the filesystem partition of the SD card on your computer.

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- \
INSTALL_MOD_PATH=/path/to/filesystem modules_install
```

5. Follow the instructions in *Setup SD card* except for the step of renaming the existing `kernel7.img` (we have already copied a AArch64 kernel).

7.34.4 Setup SD card

The instructions assume that you have an SD card with a fresh install of Raspbian (or that, at least, the `boot` partition is untouched, or nearly untouched). They have been tested with the image available in 2018-03-13.

1. Insert the SD card and open the `boot` partition.
2. Rename `kernel7.img` to `kernel8.img`. This tricks the VideoCore bootloader into booting the Arm cores in AArch64 mode, like TF-A needs, even though the kernel is not compiled for AArch64.
3. Copy `armstub8.bin` here. When `kernel8.img` is available, The VideoCore bootloader will look for a file called `armstub8.bin` and load it at address **0x0** instead of a predefined one.
4. To enable the serial port "Mini UART" in Linux, open `cmdline.txt` and add `console=serial0,115200 console=tty1`.
5. Open `config.txt` and add the following lines at the end (`enable_uart=1` is only needed to enable debugging through the Mini UART):

```
enable_uart=1
kernel_address=0x02000000
device_tree_address=0x01000000
```

If you connect a serial cable to the Mini UART and your computer, and connect to it (for example, with `screen /dev/ttyUSB0 115200`) you should see some text. In the case of an AArch32 kernel, you should see something like this:

```
NOTICE:  Booting Trusted Firmware
NOTICE:  BL1: v1.4(release):v1.4-329-g61e94684-dirty
NOTICE:  BL1: Built : 00:09:25, Nov  6 2017
NOTICE:  BL1: Booting BL2
NOTICE:  BL2: v1.4(release):v1.4-329-g61e94684-dirty
NOTICE:  BL2: Built : 00:09:25, Nov  6 2017
NOTICE:  BL1: Booting BL31
NOTICE:  BL31: v1.4(release):v1.4-329-g61e94684-dirty
NOTICE:  BL31: Built : 00:09:25, Nov  6 2017
[    0.266484] bcm2835-aux-uart 3f215040.serial: could not get clk: -517

Raspbian GNU/Linux 9 raspberrypi ttyS0
raspberrypi login:
```

Just enter your credentials, everything should work as expected. Note that the HDMI output won't show any text during boot.

7.35 Raspberry Pi 4

The *Raspberry Pi 4* is an inexpensive single-board computer that contains four Arm Cortex-A72 cores. Also in contrast to previous Raspberry Pi versions this model has a GICv2 interrupt controller.

This port is a minimal port to support loading non-secure EL2 payloads such as a 64-bit Linux kernel. Other payloads such as U-Boot or EDK-II should work as well, but have not been tested at this point.

IMPORTANT NOTE: This port isn't secure. All of the memory used is DRAM, which is available from both the Non-secure and Secure worlds. The SoC does not seem to feature a secure memory controller of any kind, so portions of DRAM can't be protected properly from the Non-secure world.

7.35.1 Build Instructions

There are no real configuration options at this point, so there is only one universal binary (bl31.bin), which can be built with:

```
CROSS_COMPILE=aarch64-linux-gnu- make PLAT=rpi4 DEBUG=1
```

Copy the generated build/rpi4/debug/bl31.bin to the SD card, adding an entry starting with `armstub=`, then followed by the respective file name to `config.txt`. You should have AArch64 code in the file loaded as the "kernel", as BL31 will drop into AArch64/EL2 to the respective load address. arm64 Linux kernels are known to work this way.

Other options that should be set in `config.txt` to properly boot 64-bit kernels are:

```
enable_uart=1
arm_64bit=1
enable_gic=1
```

The BL31 code will patch the provided device tree blob in memory to advertise PSCI support, also will add a reserved-memory node to the DT to tell the non-secure payload to not touch the resident TF-A code.

If you connect a serial cable between the Mini UART and your computer, and connect to it (for example, with `screen /dev/ttyUSB0 115200`) you should see some text from BL31, followed by the output of the EL2 payload. The command line provided is read from the `cmdline.txt` file on the SD card.

7.35.2 TF-A port design

In contrast to the existing Raspberry Pi 3 port this one here is a BL31-only port, also it deviates quite a lot from the RPi3 port in many other ways. There is not so much difference between the two models, so eventually those two could be (more) unified in the future.

As with the previous models, the GPU and its firmware are the first entity to run after the SoC gets its power. The on-chip Boot ROM loads the next stage (bootcode.bin) from flash (EEPROM), which is again GPU code. This part knows how to access the MMC controller and how to parse a FAT filesystem, so it will load further components and configuration files from the first FAT partition on the SD card.

To accommodate this existing way of configuring and setting up the board, we use as much of this workflow as possible. If bootcode.bin finds a file called `armstub8.bin` on the SD card or it gets pointed to such code by finding a `armstub=` key in `config.txt`, it will load this file to the beginning of DRAM (address 0) and execute it in AArch64 EL3. But before doing that, it will also load a "kernel" and the device tree into memory. The load addresses have a default, but can also be changed by setting them in `config.txt`. If the GPU firmware finds a magic value in the armstub image file, it will put those two load addresses in memory locations near the beginning of memory, where TF-A code picks them up.

To keep things simple, we will just use the kernel load address as the BL33 entry point, also put the DTB address in the x0 register, as requested by the arm64 Linux kernel boot protocol. This does not necessarily mean that the EL2 payload needs to be a Linux kernel, a bootloader or any other kernel would work as well, as long as it can cope with having the DT address in register x0. If the payload has other means of finding the device tree, it could ignore this address as well.

7.36 Raspberry Pi 5

The Raspberry Pi 5 is a single-board computer that contains four Arm Cortex-A76 cores.

This port is a minimal BL31 implementation capable of booting 64-bit EL2 payloads such as Linux and EDK2.

IMPORTANT NOTE: This port isn't secure. All of the memory used is DRAM, which is available from both the Non-secure and Secure worlds. The SoC does not seem to feature a secure memory controller of any kind, so portions of DRAM can't be protected properly from the Non-secure world.

7.36.1 Build

To build this platform, run:

```
CROSS_COMPILE=aarch64-linux-gnu- make PLAT=rpi5 DEBUG=1
```

The firmware will be generated at `build/rpi5/debug/bl31.bin`.

The following build options are supported:

- `RPI3_DIRECT_LINUX_BOOT`: Enabled by default. Allows direct boot of the Linux kernel from the firmware.
- `PRELOADED_BL33_BASE`: Used to specify the fixed address of a BL33 binary that has been preloaded by earlier boot stages (VPU). Useful for bundling BL31 and BL33 in the same `armstub` image (e.g. TF-A + EDK2).
- `RPI3_PRELOADED_DTB_BASE`: This option allows to specify the fixed address of a DTB in memory. Can only be used if `device_tree_address=` is present in config.txt.
- `RPI3_RUNTIME_UART`: Indicates whether TF-A should use the debug UART for runtime messages or not. `-1` (default) disables the option, any other value enables it.

7.36.2 Usage

Copy the firmware binary to the first FAT32 partition of a supported boot media (SD, USB) and append `armstub=bl31.bin` to config.txt, or just rename the file to `armstub8-2712.bin`.

No other config options or files are required by the firmware alone, this will depend on the payload you intend to run.

For Linux, you must also place an appropriate DTB and kernel in the boot partition. This has been validated with a copy of Raspberry Pi OS.

The VPU will preload a BL33 AArch64 image named either `kernel_2712.img` or `kernel8.img`, which can be overridden by adding a `kernel=filename` option to config.txt.

Kernel and DTB load addresses are also chosen by the VPU and can be changed with `kernel_address=` and `device_tree_address=` in config.txt. If TF-A was built with `PRELOADED_BL33_BASE` or `RPI3_PRELOADED_DTB_BASE`, setting those config options may be necessary.

By default, all boot stages print messages to the dedicated UART debug port. Configuration is `115200 8n1`.

7.36.3 Design

This port is largely based on the RPi 4 one.

The boot process is essentially the same, the only notable difference being that all VPU blobs have been moved into EEPROM (former start4.elf & fixup4.dat). There's also a custom BL31 TF-A armstub included for PSCI, which can be replaced with this port.

7.37 Renesas R-Car

"R-Car" is the nickname for Renesas' system-on-chip (SoC) family for car information systems designed for the next-generation of automotive computing for the age of autonomous vehicles.

The scalable R-Car hardware platform and flexible software platform cover the full product range, from the premium class to the entry level. Plug-ins are available for multiple open-source software tools.

7.37.1 Renesas R-Car Gen3 evaluation boards:

	Standard	Low Cost Boards (LCB)
R-Car H3	• Salvator-X • Salvator-XS	• R-Car Starter Kit Premier
R-Car M3-W	• Salvator-X • Salvator-XS	• R-Car Starter Kit Pro
R-Car M3-N	• Salvator-X • Salvator-XS	
R-Car V3M	• Eagle	• Starter Kit
R-Car V3H	• Condor	• Starter Kit
R-Car D3	• Draak	

boards info

The current TF-A port has been tested on the R-Car H3 Salvator-X Soc_id r8a7795 revision ES1.1 (uses a Secure Payload Dispatcher)

```
ARM CA57 (ARMv8) 1.5 GHz quad core, with NEON/VFPv4, L1$ I/D
48K/32K, L2$ 2MB
ARM CA53 (ARMv8) 1.2 GHz quad core, with NEON/VFPv4, L1$ I/D 32K/32K,
L2$ 512K
Memory controller for LPDDR4-3200 4GB in 2 channels, each 64-bit wide
Two- and three-dimensional graphics engines,
Video processing units,
3 channels Display Output,
6 channels Video Input,
SD card host interface,
USB3.0 and USB2.0 interfaces,
CAN interfaces
Ethernet AVB
PCI Express Interfaces
Memories
    INTERNAL 384KB SYSTEM RAM
    DDR 4 GB LPDDR4
    HYPERFLASH 64 MB HYPER FLASH (512 MBITS, 160 MHZ, 320 MBYTES/S)
    QSPI FLASH 16MB QSPI (128 MBITS,80 MHZ,80 MBYTES/S)1 HEADER QSPI
    MODULE
    EMMC 32 GB EMMC (HS400 240 MBYTES/S)
    MICROSD-CARD SLOT (SDR104 100 MBYTES/S)
```

7.37.2 Overview

On the rcar-gen3 the BOOTROM starts the cpu at EL3; for this port BL2 will therefore be entered at this exception level (the Renesas' ATF reference tree [1] resets into EL1 before entering BL2 - see its bl2.ld.S)

BL2 initializes DDR (and on some platforms i2c to interface to the PMIC) before determining the boot reason (cold or warm).

During suspend all CPUs are switched off and the DDR is put in backup mode (some kind of self-refresh mode). This means that BL2 is always entered in a cold boot scenario.

Once BL2 boots, it determines the boot reason, writes it to shared memory (BOOT_KIND_BASE) together with the BL31 parameters (PARAMS_BASE) and jumps to BL31.

To all effects, BL31 is as if it is being entered in reset mode since it still needs to initialize the rest of the cores; this is the reason behind using direct shared memory access to BOOT_KIND_BASE _and_ PARAMS_BASE instead of using registers to get to those locations (see el3_common_macros.S and bl31_entrypoint.S for the RESET_TO_BL31 use case).

Depending on the boot reason BL31 initializes the rest of the cores: in case of suspend, it uses a MBOX memory region to recover the program counters.

[1] https://github.com/renesas-rcar/arm-trusted-firmware

7.37.3 How to build

The TF-A build options depend on the target board so you will have to refer to those specific instructions. What follows is customized to the H3 SiP Salvator-X development system used in this port.

Build Tested:

RCAR_OPT="LSI=H3 RCAR_DRAM_SPLIT=1 RCAR_LOSSY_ENABLE=1"

$ make clean bl2 bl31 rcar_layout_tool PLAT=rcar ${RCAR_OPT} SPD=opteed

System Tested:

- mbed_tls: git@github.com:ARMmbed/mbedtls.git [devel]

  commit 552754a6ee82bab25d1bdf28c8261a4518e65e4d Merge: 68dbc94 f34a4c1 Author: Simon Butcher <simon.butcher@arm.com> Date: Thu Aug 30 00:57:28 2018 +0100

- optee_os: https://github.com/BayLibre/optee_os

  Until it gets merged into OP-TEE, the port requires Renesas' Trusted Environment with a modification to support power management. commit 80105192cba9e704ebe8df7ab84095edc2922f84

  Author: Jorge Ramirez-Ortiz <jramirez@baylibre.com> Date: Thu Aug 30 16:49:49 2018 +0200 plat-rcar: cpu-suspend: handle the power level Signed-off-by: Jorge Ramirez-Ortiz <jramirez@baylibre.com>

- u-boot: The port has beent tested using mainline uboot.

  commit 4cdeda511f8037015b568396e6dcc3d8fb41e8c0 Author: Fabio Estevam <festevam@gmail.com> Date: Tue Sep 4 10:23:12 2018 -0300

- linux: The port has beent tested using mainline kernel.

  commit 7876320f88802b22d4e2daf7eb027dd14175a0f8 Author: Linus Torvalds <torvalds@linux-foundation.org> Date: Sun Sep 16 11:52:37 2018 -0700 Linux 4.19-rc4

TF-A Build Procedure

- Fetch all the above 4 repositories.
- Prepare the AARCH64 toolchain.
- Build u-boot using r8a7795_salvator-x_defconfig. Result: u-boot-elf.srec

```
make CROSS_COMPILE=aarch64-linux-gnu-
   r8a7795_salvator-x_defconfig

make CROSS_COMPILE=aarch64-linux-gnu-
```

- Build atf Result: bootparam_sa0.srec, cert_header_sa6.srec, bl2.srec, bl31.srec

```
RCAR_OPT="LSI=H3 RCAR_DRAM_SPLIT=1 RCAR_LOSSY_ENABLE=1"

make clean bl2 bl31 rcar PLAT=rcar ${RCAR_OPT} SPD=opteed
```

- Build optee-os Result: tee.srec

```
make -j8 PLATFORM="rcar" CFG_ARM64_core=y
```

Install Procedure

- Boot the board in Mini-monitor mode and enable access to the Hyperflash.
- Use the XSL2 Mini-monitor utility to accept all the SREC ascii transfers over serial.

7.37.4 Boot trace

Notice that BL31 traces are not accessible via the console and that in order to verbose the BL2 output you will have to compile TF-A with LOG_LEVEL=50 and DEBUG=1

```
Initial Program Loader(CA57) Rev.1.0.22
NOTICE:  BL2: PRR is R-Car H3 Ver.1.1
NOTICE:  BL2: Board is Salvator-X Rev.1.0
NOTICE:  BL2: Boot device is HyperFlash(80MHz)
NOTICE:  BL2: LCM state is CM
NOTICE:  AVS setting succeeded. DVFS_SetVID=0x53
```

(continues on next page)

(continued from previous page)

```
NOTICE:  BL2: DDR1600(rev.0.33)NOTICE:  [COLD_BOOT]NOTICE:  ..0
NOTICE:  BL2: DRAM Split is 4ch
NOTICE:  BL2: QoS is default setting(rev.0.37)
NOTICE:  BL2: Lossy Decomp areas
NOTICE:       Entry 0: DCMPAREACRAx:0x80000540 DCMPAREACRBx:0x570
NOTICE:       Entry 1: DCMPAREACRAx:0x40000000 DCMPAREACRBx:0x0
NOTICE:       Entry 2: DCMPAREACRAx:0x20000000 DCMPAREACRBx:0x0
NOTICE:  BL2: v2.0(release):v2.0-rc0-32-gbcda69a
NOTICE:  BL2: Built : 16:41:23, Oct  2 2018
NOTICE:  BL2: Normal boot
INFO:    BL2: Doing platform setup
INFO:    BL2: Loading image id 3
NOTICE:  BL2: dst=0xe6322000 src=0x8180000 len=512(0x200)
NOTICE:  BL2: dst=0x43f00000 src=0x8180400 len=6144(0x1800)
WARNING: r-car ignoring the BL31 size from certificate,using
RCAR_TRUSTED_SRAM_SIZE instead
INFO:    Loading image id=3 at address 0x44000000
NOTICE:  rcar_file_len: len: 0x0003e000
NOTICE:  BL2: dst=0x44000000 src=0x81c0000 len=253952(0x3e000)
INFO:    Image id=3 loaded: 0x44000000 - 0x4403e000
INFO:    BL2: Loading image id 4
INFO:    Loading image id=4 at address 0x44100000
NOTICE:  rcar_file_len: len: 0x00100000
NOTICE:  BL2: dst=0x44100000 src=0x8200000 len=1048576(0x100000)
INFO:    Image id=4 loaded: 0x44100000 - 0x44200000
INFO:    BL2: Loading image id 5
INFO:    Loading image id=5 at address 0x50000000
NOTICE:  rcar_file_len: len: 0x00100000
NOTICE:  BL2: dst=0x50000000 src=0x8640000 len=1048576(0x100000)
INFO:    Image id=5 loaded: 0x50000000 - 0x50100000
NOTICE:  BL2: Booting BL31
INFO:    Entry point address = 0x44000000
INFO:    SPSR = 0x3cd
VERBOSE: Argument #0 = 0xe6325578
VERBOSE: Argument #1 = 0x0
VERBOSE: Argument #2 = 0x0
VERBOSE: Argument #3 = 0x0
VERBOSE: Argument #4 = 0x0
VERBOSE: Argument #5 = 0x0
VERBOSE: Argument #6 = 0x0
VERBOSE: Argument #7 = 0x0


U-Boot 2018.09-rc3-00028-g3711616 (Sep 27 2018 - 18:50:24 +0200)

CPU: Renesas Electronics R8A7795 rev 1.1
Model: Renesas Salvator-X board based on r8a7795 ES2.0+
DRAM:  3.5 GiB
Flash: 64 MiB
MMC:   sd@ee100000: 0, sd@ee140000: 1, sd@ee160000: 2
Loading Environment from MMC... OK
In:    serial@e6e88000
```

(continues on next page)

(continued from previous page)

```
Out:   serial@e6e88000
Err:   serial@e6e88000
Net:   eth0: ethernet@e6800000
Hit any key to stop autoboot:  0
=>
```

7.38 Renesas RZ/A3

The RZ/A series is an RTOS-based microprocessor (MPU) combining excellent real-time performance and fast boot time based on Renesas' proprietary technology and Arm® ecosystem, and is as user-friendly as Renesas MCUs.

The high-speed performance of the RZ/A MPU can quickly process graphics and high-load applications especially in the third generation RZ/A3M MPU powered by the 64-bit Arm® Cortex®-A55 CPU core with a maximum operating frequency of 1 GHz.

7.38.1 Renesas RZ/A3 reference platforms:

Board	Details
EK-RZ/A3M	Evaluation Kit for RZ/A3M MPU

7.38.2 Boot Sequence

RZ/A3 SoCs implements its internal boot ROM; which loads an image from a non-volatile storage to the on-chip RAM, and jumps over to it. TF-A provides a special mode, BL2-AT-EL3, which enables BL2 to execute at EL3 and small enough to fit on-chip RAM.

Once BL2 boots, it initializes DDR before copying the next image from flash to DDR, then transfer the excution to it which is usually required at EL3 at the entry point to run a RTOS (such as Zephyr or FreeRTOS). That is the reason why only BL2 is supported.

BootROM –> BL2 –> RTOS (Zephyr/FreeRTOS)

7.38.3 TF-A Build Procedure

The TF-A build options depend on the target board so you will have to refer to those specific instructions. What follows is customized to the EK-RZ/A3M used in this port.

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=rza3m_ek_nor all
```

7.39 Renesas RZ/G

The "RZ/G" Family of high-end 64-bit Arm®-based microprocessors (MPUs) enables the solutions required for the smart society of the future. Through a variety of Arm Cortex®-A53 and A57-based devices, engineers can easily implement high-resolution human machine interfaces (HMI), embedded vision, embedded artificial intelligence (e-AI) and real-time control and industrial ethernet connectivity.

The scalable RZ/G hardware platform and flexible software platform cover the full product range, from the premium class to the entry level. Plug-ins are available for multiple open-source software tools.

7.39.1 Renesas RZ/G2 reference platforms:

Board	Details
hihope-rzg2h	"96 boards" compatible board from Hoperun equipped with Renesas RZ/G2H SoC
	http://hihope.org/product/musashi
hihope-rzg2m	"96 boards" compatible board from Hoperun equipped with Renesas RZ/G2M SoC
	http://hihope.org/product/musashi
hihope-rzg2n	"96 boards" compatible board from Hoperun equipped with Renesas RZ/G2N SoC
	http://hihope.org/product/musashi
ek874	"96 boards" compatible board from Silicon Linux equipped with Renesas RZ/G2E SoC
	https://www.si-linux.co.jp/index.php?CAT%2FCAT874

boards info

The current TF-A port has been tested on the HiHope RZ/G2M SoC_id r8a774a1 revision ES1.3.

```
ARM CA57 (ARMv8) 1.5 GHz dual core, with NEON/VFPv4, L1$ I/D 48K/32K, L2$ 1MB
ARM CA53 (ARMv8) 1.2 GHz quad core, with NEON/VFPv4, L1$ I/D 32K/32K, L2$ 512K
Memory controller for LPDDR4-3200 4GB in 2 channels(32-bit bus mode)
Two- and three-dimensional graphics engines,
Video processing units,
Display Output,
Video Input,
SD card host interface,
USB3.0 and USB2.0 interfaces,
CAN interfaces,
Ethernet AVB,
Wi-Fi + BT,
PCI Express Interfaces,
Memories
    INTERNAL 384KB SYSTEM RAM
    DDR 4 GB LPDDR4
    QSPI FLASH 64MB
    EMMC 32 GB EMMC (HS400 240 MBYTES/S)
    MICROSD-CARD SLOT (SDR104 100 MBYTES/S)
```

7.39.2 Overview

On RZ/G2 SoCs the BOOTROM starts the cpu at EL3; for this port BL2 will therefore be entered at this exception level (the Renesas' ATF reference tree [1] resets into EL1 before entering BL2 - see its bl2.ld.S)

BL2 initializes DDR before determining the boot reason (cold or warm).

Once BL2 boots, it determines the boot reason, writes it to shared memory (BOOT_KIND_BASE) together with the BL31 parameters (PARAMS_BASE) and jumps to BL31.

To all effects, BL31 is as if it is being entered in reset mode since it still needs to initialize the rest of the cores; this is the reason behind using direct shared memory access to BOOT_KIND_BASE _and_ PARAMS_BASE instead of using registers to get to those locations (see el3_common_macros.S and bl31_entrypoint.S for the RESET_TO_BL31 use case).

[1] https://github.com/renesas-rz/meta-rzg2/tree/BSP-1.0.5/recipes-bsp/arm-trusted-firmware/files

7.39.3 How to build

The TF-A build options depend on the target board so you will have to refer to those specific instructions. What follows is customized to the HiHope RZ/G2M development kit used in this port.

Build Tested:

```
make bl2 bl31 rzg LOG_LEVEL=40 PLAT=rzg LSI=G2M RCAR_DRAM_SPLIT=2\
RCAR_LOSSY_ENABLE=1 SPD="none"
```

System Tested:

- mbed_tls: git@github.com:ARMmbed/mbedtls.git [devel]

commit 72ca39737f974db44723760623d1b29980c00a88
Merge: ef94c4fcf dd9ec1c57
Author: Janos Follath <janos.follath@arm.com>
Date: Wed Oct 7 09:21:01 2020 +0100

- u-boot: The port has beent tested using mainline uboot with HiHope RZ/G2M board specific patches.

commit 46ce9e777c1314ccb78906992b94001194eaa87b
Author: Heiko Schocher <hs@denx.de>
Date: Tue Nov 3 15:22:36 2020 +0100

- linux: The port has beent tested using mainline kernel.

commit f8394f232b1eab649ce2df5c5f15b0e528c92091
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date: Sun Nov 8 16:10:16 2020 -0800
Linux 5.10-rc3

TF-A Build Procedure

- Fetch all the above 3 repositories.
- Prepare the AARCH64 toolchain.
- Build u-boot using hihope_rzg2_defconfig.

  Result: u-boot-elf.srec

```
make CROSS_COMPILE=aarch64-linux-gnu-
   hihope_rzg2_defconfig

make CROSS_COMPILE=aarch64-linux-gnu-
```

- Build TF-A

  Result: bootparam_sa0.srec, cert_header_sa6.srec, bl2.srec, bl31.srec

```
make bl2 bl31 rzg LOG_LEVEL=40 PLAT=rzg LSI=G2M RCAR_DRAM_SPLIT=2\
RCAR_LOSSY_ENABLE=1 SPD="none"
```

Install Procedure

- Boot the board in Mini-monitor mode and enable access to the QSPI flash.
- Use the flash_writer utility[2] to flash all the SREC files.

[2] https://github.com/renesas-rz/rzg2_flash_writer

7.39.4 Boot trace

```
INFO:    ARM GICv2 driver initialized
NOTICE:  BL2: RZ/G2 Initial Program Loader(CA57) Rev.2.0.6
NOTICE:  BL2: PRR is RZ/G2M Ver.1.3
NOTICE:  BL2: Board is HiHope RZ/G2M Rev.4.0
NOTICE:  BL2: Boot device is QSPI Flash(40MHz)
NOTICE:  BL2: LCM state is unknown
NOTICE:  BL2: DDR3200(rev.0.40)
NOTICE:  BL2: [COLD_BOOT]
NOTICE:  BL2: DRAM Split is 2ch
NOTICE:  BL2: QoS is default setting(rev.0.19)
NOTICE:  BL2: DRAM refresh interval 1.95 usec
NOTICE:  BL2: Periodic Write DQ Training
```

(continues on next page)

(continued from previous page)

```
NOTICE:  BL2: CH0: 400000000 - 47fffffff, 2 GiB
NOTICE:  BL2: CH2: 600000000 - 67fffffff, 2 GiB
NOTICE:  BL2: Lossy Decomp areas
NOTICE:       Entry 0: DCMPAREACRAx:0x80000540 DCMPAREACRBx:0x570
NOTICE:       Entry 1: DCMPAREACRAx:0x40000000 DCMPAREACRBx:0x0
NOTICE:       Entry 2: DCMPAREACRAx:0x20000000 DCMPAREACRBx:0x0
NOTICE:  BL2: FDT at 0xe631db30
NOTICE:  BL2: v2.3(release):v2.4-rc0-2-g1433701e5
NOTICE:  BL2: Built : 13:45:26, Nov  7 2020
NOTICE:  BL2: Normal boot
INFO:    BL2: Doing platform setup
INFO:    BL2: Loading image id 3
NOTICE:  BL2: dst=0xe631d200 src=0x8180000 len=512(0x200)
NOTICE:  BL2: dst=0x43f00000 src=0x8180400 len=6144(0x1800)
WARNING: r-car ignoring the BL31 size from certificate,using RCAR_TRUSTED_
↪SRAM_SIZE instead
INFO:    Loading image id=3 at address 0x44000000
NOTICE:  rcar_file_len: len: 0x0003e000
NOTICE:  BL2: dst=0x44000000 src=0x81c0000 len=253952(0x3e000)
INFO:    Image id=3 loaded: 0x44000000 - 0x4403e000
INFO:    BL2: Loading image id 5
INFO:    Loading image id=5 at address 0x50000000
NOTICE:  rcar_file_len: len: 0x00100000
NOTICE:  BL2: dst=0x50000000 src=0x8300000 len=1048576(0x100000)
INFO:    Image id=5 loaded: 0x50000000 - 0x50100000
NOTICE:  BL2: Booting BL31
INFO:    Entry point address = 0x44000000
INFO:    SPSR = 0x3cd


U-Boot 2021.01-rc1-00244-gac37e14fbd (Nov 04 2020 - 20:03:34 +0000)

CPU: Renesas Electronics R8A774A1 rev 1.3
Model: HopeRun HiHope RZ/G2M with sub board
DRAM:  3.9 GiB
MMC:   mmc@ee100000: 0, mmc@ee160000: 1
Loading Environment from MMC... OK
In:    serial@e6e88000
Out:   serial@e6e88000
Err:   serial@e6e88000
Net:   eth0: ethernet@e6800000
Hit any key to stop autoboot:  0
=>
```

7.40 Rockchip SoCs

Trusted Firmware-A supports a number of Rockchip ARM SoCs from both AARCH32 and AARCH64 fields.

This includes right now: - px30: Quad-Core Cortex-A53 - rk3288: Quad-Core Cortex-A17 (past A12) - rk3328: Quad-Core Cortex-A53 - rk3368: Octa-Core Cortex-A53 - rk3399: Hexa-Core Cortex-A53/A72 - rk3566/rk3568: Quad-Core Cortex-A55 - rk3576: Octa-Core Cortex-A53/A72 - rk3588: Octa-Core Cortex-A55/A76

7.40.1 Boot Sequence

For AARCH32:
Bootrom –> BL1/BL2 –> BL32 –> BL33 –> Linux kernel

For AARCH64:
Bootrom –> BL1/BL2 –> BL31 –> BL33 –> Linux kernel

BL1/2 and BL33 can currently be supplied from either: - Coreboot + Depthcharge - U-Boot - either separately as TPL+SPL or only SPL

7.40.2 How to build

Rockchip SoCs expect TF-A's BL31 (AARCH64) or BL32 (AARCH32) to get integrated with other boot software like U-Boot or Coreboot, so only these images need to get build from the TF-A repository.

For AARCH64 architectures the build command looks like

make CROSS_COMPILE=aarch64-linux-gnu- PLAT=rk3399 bl31

while AARCH32 needs a slightly different command

make ARCH=aarch32 CROSS_COMPILE=arm-linux-gnueabihf- PLAT=rk3288 AARCH32_SP=sp_min bl32

Both need replacing the PLAT argument with the platform from above you want to build for and the CROSS_COMPILE argument with you cross- compilation toolchain.

7.40.3 How to deploy

Both upstream U-Boot and Coreboot projects contain instructions on where to put the built images during their respective build process. So after successfully building TF-A just follow their build instructions to continue.

7.41 Socionext UniPhier

Socionext UniPhier Armv8-A SoCs use Trusted Firmware-A (TF-A) as the secure world firmware, supporting BL2 and BL31.

UniPhier SoC family implements its internal boot ROM, which loads 64KB[1] image from a non-volatile storage to the on-chip SRAM, and jumps over to it. TF-A provides a special mode, BL2-AT-EL3, which enables BL2 to execute at EL3. It is useful for platforms with non-TF-A boot ROM, like UniPhier. Here, a problem is BL2 does not fit in the 64KB limit if *Trusted Board Boot (TBB)* is enabled. To solve this issue, Socionext provides a first stage loader called UniPhier BL. This loader runs in the on-chip SRAM, initializes the DRAM, expands BL2 there, and hands the control over to it. Therefore, all images of TF-A run in DRAM.

The UniPhier platform works with/without TBB. See below for the build process of each case. The image authentication for the UniPhier platform fully complies with the Trusted Board Boot Requirements (TBBR) specification.

The UniPhier BL does not implement the authentication functionality, that is, it can not verify the BL2 image by itself. Instead, the UniPhier BL assures the BL2 validity in a different way; BL2 is GZIP-compressed and appended to the UniPhier BL. The concatenation of the UniPhier BL and the compressed BL2 fits in the 64KB limit. The concatenated image is loaded by the internal boot ROM (and verified if the chip fuses are blown).

7.41.1 Boot Flow

1. The Boot ROM

   This is hard-wired ROM, so never corrupted. It loads the UniPhier BL (with compressed-BL2 appended) into the on-chip SRAM. If the SoC fuses are blown, the image is verified by the SoC's own method.

2. UniPhier BL

   This runs in the on-chip SRAM. After the minimum SoC initialization and DRAM setup, it decompresses the appended BL2 image into the DRAM, then jumps to the BL2 entry.

3. BL2 (at EL3)

   This runs in the DRAM. It extracts more images such as BL31, BL33 (optionally SCP_BL2, BL32 as well) from Firmware Image Package (FIP). If TBB is enabled, they are all authenticated by the standard mechanism of TF-A. After loading all the images, it jumps to the BL31 entry.

4. BL31, BL32, and BL33

   They all run in the DRAM. See *Firmware Design* for details.

[1] Some SoCs can load 80KB, but the software implementation must be aligned to the lowest common denominator.

7.41.2 Basic Build

BL2 must be compressed for the reason above. The UniPhier's platform makefile provides a build target `bl2_gzip` for this.

For a non-secure boot loader (aka BL33), U-Boot is well supported for UniPhier SoCs. The U-Boot image (`u-boot.bin`) must be built in advance. For the build procedure of U-Boot, refer to the document in the U-Boot project.

To build minimum functionality for UniPhier (without TBB):

```
make CROSS_COMPILE=<gcc-prefix> PLAT=uniphier BL33=<path-to-BL33> bl2_gzip fip
```

Output images:

- `bl2.bin.gz`
- `fip.bin`

7.41.3 Optional features

- Trusted Board Boot

  mbed TLS is needed as the cryptographic and image parser modules. Refer to the *Prerequisites* document for the appropriate version of mbed TLS.

  To enable TBB, add the following options to the build command:

  ```
  TRUSTED_BOARD_BOOT=1 GENERATE_COT=1
  ```

- System Control Processor (SCP)

  If desired, FIP can include an SCP BL2 image. If BL2 finds an SCP BL2 image in FIP, BL2 loads it into DRAM and kicks the SCP. Most of UniPhier boards still work without SCP, but SCP provides better power management support.

  To include SCP BL2, add the following option to the build command:

  ```
  SCP_BL2=<path-to-SCP>
  ```

- BL32 (Secure Payload)

  To enable BL32, add the following options to the build command:

  ```
  SPD=<spd> BL32=<path-to-BL32>
  ```

  If you use TSP for BL32, `BL32=<path-to-BL32>` is not required. Just add the following:

  ```
  SPD=tspd
  ```

7.42 Socionext Synquacer

Socionext's Synquacer SC2A11 is a multi-core processor with 24 cores of Arm Cortex-A53. The Developerbox, of 96boards, is a platform that contains this processor. This port of the Trusted Firmware only supports this platform at the moment.

More information are listed in link.

7.42.1 How to build

Code Locations

- Trusted Firmware-A: link
- edk2: link
- edk2-platforms: link
- edk2-non-osi: link

Boot Flow

SCP firmware –> TF-A BL31 –> UEFI(edk2)

Build Procedure

- Firstly, in addition to the "normal" build tools you will also need a few specialist tools. On a Debian or Ubuntu operating system try:

```
sudo apt install acpica-tools device-tree-compiler uuid-dev
```

- Secondly, create a new working directory and store the absolute path to this directory in an environment variable, WORKSPACE. It does not matter where this directory is created but as an example:

```
export WORKSPACE=$HOME/build/developerbox-firmware
mkdir -p $WORKSPACE
```

- Run the following commands to clone the source code:

```
cd $WORKSPACE
git clone https://github.com/ARM-software/arm-trusted-firmware -b master
git clone https://github.com/tianocore/edk2.git -b master
git clone https://github.com/tianocore/edk2-platforms.git -b master
git clone https://github.com/tianocore/edk2-non-osi.git -b master
```

- Build ATF:

```
cd $WORKSPACE/arm-trusted-firmware
make -j`nproc` PLAT=synquacer PRELOADED_BL33_BASE=0x8200000 bl31 fiptool
tools/fiptool/fiptool create \
      --tb-fw ./build/synquacer/release/bl31.bin \
      --soc-fw ./build/synquacer/release/bl31.bin \
      --scp-fw ./build/synquacer/release/bl31.bin \
      ../edk2-non-osi/Platform/Socionext/DeveloperBox/fip_all_arm_tf.bin
```

- Build EDK2:

```
cd $WORKSPACE
export PACKAGES_PATH=$WORKSPACE/edk2:$WORKSPACE/edk2-platforms:
↪$WORKSPACE/edk2-non-osi
export ACTIVE_PLATFORM="Platform/Socionext/DeveloperBox/DeveloperBox.dsc"
export GCC5_AARCH64_PREFIX=aarch64-linux-gnu-
unset ARCH

. edk2/edksetup.sh
make -C edk2/BaseTools

build -p $ACTIVE_PLATFORM -b RELEASE -a AARCH64 -t GCC5 -n `nproc` -D DO_
↪X86EMU=TRUE
```

- The firmware image, which comprises the option ROM, ARM trusted firmware and EDK2 itself, can be found $WORKSPACE/../Build/DeveloperBox/RELEASE_GCC5/FV/. Use SYNQUACERFIRMWAREUPDATECAPSULEFMPPKCS7.Cap for UEFI capsule update and SPI_NOR_IMAGE.fd for the serial flasher.

  Note #1: -t GCC5 can be loosely translated as "enable link-time-optimization"; any version of gcc >= 5 will support this feature and may be used to build EDK2.

  Note #2: Replace -b RELEASE with -b DEBUG to build a debug.

Install the System Firmware

- Providing your Developerbox is fully working and has on operating system installed then you can adopt your the newly compiled system firmware using the capsule update method:.

```
sudo apt install fwupdate
sudo fwupdate --apply {50b94ce5-8b63-4849-8af4-ea479356f0e3} \
              SYNQUACERFIRMWAREUPDATECAPSULEFMPPKCS7.Cap
sudo reboot
```

- Alternatively you can install SPI_NOR_IMAGE.fd using the board recovery method.

7.43 STMicroelectronics STM32 MPUs

7.43.1 STM32 MPUs

STM32 MPUs are microprocessors designed by STMicroelectronics based on Arm Cortex-A. This page presents the common configuration of STM32 MPUs, more details and dedicated configuration can be found in each STM32 MPU page (*STM32MP1* or *STM32MP2*)

Design

The STM32 MPU resets in the ROM code of the Cortex-A. The primary boot core (core 0) executes the boot sequence while secondary boot core (core 1) is kept in a holding pen loop. The ROM code boot sequence loads the TF-A binary image from boot device to embedded SRAM.

The TF-A image must be properly formatted with a STM32 header structure for ROM code is able to load this image. Tool stm32image can be used to prepend this header to the generated TF-A binary.

Boot

Only BL2 (with STM32 header) is loaded by ROM code. The other binaries are inside the FIP binary: BL31 (for Aarch64 platforms), BL32 (OP-TEE), U-Boot and their respective device tree blobs.

Boot sequence

ROM code -> BL2 (compiled with RESET_TO_BL2) -> OP-TEE -> BL33 (U-Boot)

Build Instructions

Boot media(s) supported by BL2 must be specified in the build command. Available storage medias are:

- `STM32MP_SDMMC`
- `STM32MP_EMMC`
- `STM32MP_RAW_NAND`
- `STM32MP_SPI_NAND`
- `STM32MP_SPI_NOR`

Serial boot devices:

- `STM32MP_UART_PROGRAMMER`
- `STM32MP_USB_PROGRAMMER`

Only one storage or serial device should be selected in the build command line, to save space and not overflow SYSRAM size, or else the platform won't build or boot.

Other configuration flags:

- `DTB_FILE_NAME`: to precise board device-tree blob to be used.
  Default: stm32mp157c-ev1.dtb
- `DWL_BUFFER_BASE`: the 'serial boot' load address of FIP,
  default location (end of the first 128MB) is used when absent
- `STM32MP_FORCE_MTD_START_OFFSET`: to force the FIP offset on MTD devices.
  Requires PSA_FWU_SUPPORT=0.
  Default: 0x80000 on NOR, 0x200000 on NAND
- `STM32MP_NAND_FIP_RECOVERY_OFFSET`: NAND offset of a recovery FIP, tried when the active FIP and its backup blocks are exhausted. Requires PSA_FWU_SUPPORT=0.
  Use a partition the same size as the active FIP.
  Default: 0 (disabled)
- `STM32MP_PSA_FWU_AB_SUPPORT`: to support PSA FWU A/B system.
  It requires `PSA_FWU_SUPPORT` to be enabled and is only supported on block
  devices with a GPT partition table like SD or eMMC.
  Default: 0 (disabled)
- `STM32MP_RECONFIGURE_CONSOLE`: to re-configure crash console (especially after BL2).
  Default: 0 (disabled)
- `STM32MP_UART_BAUDRATE`: to select UART baud rate.
  Default: 115200

Populate SD-card

Boot with FIP

The SD-card has to be formatted with GPT. It should contain at least those partitions:

- fsbl: to copy the tf-a-stm32mp157c-ev1.stm32 binary (BL2)
- fip (GUID 19d5df83-11b0-457b-be2c-7559c13142a5): which contains the FIP binary

Usually, two copies of fsbl are used (fsbl1 and fsbl2) instead of one partition fsbl.

7.43.2 STM32MP1

STM32MP1 is a microprocessor designed by STMicroelectronics based on Arm Cortex-A7. It is an Armv7-A platform, using dedicated code from TF-A. More information can be found on STM32MP1 Series page.

For TF-A common configuration of STM32 MPUs, please check *STM32 MPUs* page.

STM32MP1 Versions

There are 2 variants for STM32MP1: STM32MP13 and STM32MP15

STM32MP13 Versions

The STM32MP13 series is available in 3 different lines which are pin-to-pin compatible:

- STM32MP131: Single Cortex-A7 core
- STM32MP133: STM32MP131 + 2*CAN, ETH2(GMAC), ADC1
- STM32MP135: STM32MP133 + DCMIPP, LTDC

Each line comes with a security option (cryptography & secure boot) and a Cortex-A frequency option:

- A Cortex-A7 @ 650 MHz
- C Secure Boot + HW Crypto + Cortex-A7 @ 650 MHz
- D Cortex-A7 @ 900 MHz
- F Secure Boot + HW Crypto + Cortex-A7 @ 900 MHz

STM32MP15 Versions

The STM32MP15 series is available in 3 different lines which are pin-to-pin compatible:

- STM32MP157: Dual Cortex-A7 cores, Cortex-M4 core @ 209 MHz, 3D GPU, DSI display interface and CAN FD
- STM32MP153: Dual Cortex-A7 cores, Cortex-M4 core @ 209 MHz and CAN FD
- STM32MP151: Single Cortex-A7 core, Cortex-M4 core @ 209 MHz

Each line comes with a security option (cryptography & secure boot) and a Cortex-A frequency option:

- A Basic + Cortex-A7 @ 650 MHz
- C Secure Boot + HW Crypto + Cortex-A7 @ 650 MHz
- D Basic + Cortex-A7 @ 800 MHz
- F Secure Boot + HW Crypto + Cortex-A7 @ 800 MHz

The STM32MP1 part number codification page gives more information about part numbers.

Memory mapping

```
0x00000000 +-----------------+
           |                 |   ROM
0x00020000 +-----------------+
           |                 |
           |       ...       |
           |                 |
0x2FFC0000 +-----------------+ \
           |     BL32 DTB    | |
0x2FFC5000 +-----------------+ |
           |       BL32      | |
0x2FFDF000 +-----------------+ |
           |       ...       | |
0x2FFE3000 +-----------------+ |
           |     BL2 DTB     | | Embedded SRAM
0x2FFEA000 +-----------------+ |
           |       BL2       | |
0x2FFFF000 +-----------------+ |
           |   SCMI mailbox  | |
0x30000000 +-----------------+ /
           |                 |
           |       ...       |
           |                 |
0x40000000 +-----------------+
           |                 |
           |                 |   Devices
           |                 |
0xC0000000 +-----------------+ \
           |                 | |
0xC0100000 +-----------------+ |
           |       BL33      | | Non-secure RAM (DDR)
           |       ...       | |
           |                 | |
0xFFFFFFFF +-----------------+ /
```

Build Instructions

STM32MP1x specific flags

Dedicated STM32MP1 flags:

- `STM32_TF_VERSION`: to manage BL2 monotonic counter.
  Default: 0
- `STM32MP13`: to select STM32MP13 variant configuration.
  Default: 0
- `STM32MP15`: to select STM32MP15 variant configuration.
  Default: 1

- `STM32MP_STPMIC1L`: to select STPMIC1L variant configuration.

  Default: 0

Boot with FIP

You need to build BL2, BL32 (SP_min or OP-TEE) and BL33 (U-Boot) before building FIP binary.

U-Boot

```
cd <u-boot_directory>
make stm32mp15_trusted_defconfig
make DEVICE_TREE=stm32mp157c-ev1 all
```

OP-TEE (recommended)

OP-TEE is the default BL32 supported for STMicroelectronics platforms.

```
cd <optee_directory>
make CROSS_COMPILE=arm-linux-gnueabihf- ARCH=arm PLATFORM=stm32mp1 \
    CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-ev1.dts
```

TF-A BL32 (SP_min) (not recommended)

If you choose not to use OP-TEE, you can use TF-A SP_min. This is not the recommended BL32 to use, and will have very limited support. To build TF-A BL32, and its device tree file:

```
make CROSS_COMPILE=arm-none-eabi- PLAT=stm32mp1 ARCH=aarch32 ARM_ARCH_MAJOR=7␣
↪\
    AARCH32_SP=sp_min DTB_FILE_NAME=stm32mp157c-ev1.dtb bl32 dtbs
```

TF-A BL2

To build TF-A BL2 with its STM32 header for SD-card boot:

```
make CROSS_COMPILE=arm-none-eabi- PLAT=stm32mp1 ARCH=aarch32 ARM_ARCH_MAJOR=7␣
↪\
    DTB_FILE_NAME=stm32mp157c-ev1.dtb STM32MP_SDMMC=1
```

For other boot devices, you have to replace STM32MP_SDMMC in the previous command with the desired device flag.

This BL2 is independent of the BL32 used (SP_min or OP-TEE)

FIP

With BL32 SP_min:

```
make CROSS_COMPILE=arm-none-eabi- PLAT=stm32mp1 ARCH=aarch32 ARM_ARCH_MAJOR=7␣
↪\
    AARCH32_SP=sp_min \
    DTB_FILE_NAME=stm32mp157c-ev1.dtb \
    BL33=<u-boot_directory>/u-boot-nodtb.bin \
    BL33_CFG=<u-boot_directory>/u-boot.dtb \
    fip
```

With OP-TEE:

```
make CROSS_COMPILE=arm-none-eabi- PLAT=stm32mp1 ARCH=aarch32 ARM_ARCH_MAJOR=7␣
↪\
    AARCH32_SP=optee \
    DTB_FILE_NAME=stm32mp157c-ev1.dtb \
    BL33=<u-boot_directory>/u-boot-nodtb.bin \
    BL33_CFG=<u-boot_directory>/u-boot.dtb \
    BL32=<optee_directory>/tee-header_v2.bin \
    BL32_EXTRA1=<optee_directory>/tee-pager_v2.bin
    BL32_EXTRA2=<optee_directory>/tee-pageable_v2.bin
    fip
```

Trusted Boot Board

```
tools/cert_create/cert_create -n --rot-key build/stm32mp1/release/rot_key.pem␣
↪\
    --tfw-nvctr 0 \
    --ntfw-nvctr 0 \
    --key-alg ecdsa --hash-alg sha256 \
    --trusted-key-cert build/stm32mp1/release/trusted_key.crt \
    --tos-fw <optee_directory>/tee-header_v2.bin \
    --tos-fw-extra1 <optee_directory>/tee-pager_v2.bin \
    --tos-fw-extra2 <optee_directory>/tee-pageable_v2.bin \
    --tos-fw-cert build/stm32mp1/release/tos_fw_content.crt \
    --tos-fw-key-cert build/stm32mp1/release/tos_fw_key.crt \
    --nt-fw <u-boot_directory>/u-boot-nodtb.bin \
    --nt-fw-cert build/stm32mp1/release/nt_fw_content.crt \
    --nt-fw-key-cert build/stm32mp1/release/nt_fw_key.crt \
    --hw-config <u-boot_directory>/u-boot.dtb \
    --fw-config build/stm32mp1/release/fdts/fw-config.dtb \
    --stm32mp-cfg-cert build/stm32mp1/release/stm32mp_cfg_cert.crt

tools/fiptool/fiptool create --tos-fw <optee_directory>/tee-header_v2.bin \
    --tos-fw-extra1 <optee_directory>/tee-pager_v2.bin \
    --tos-fw-extra2 <optee_directory>/tee-pageable_v2.bin \
    --nt-fw <u-boot_directory>/u-boot-nodtb.bin \
    --hw-config <u-boot_directory>/u-boot.dtb \
```

(continues on next page)

(continued from previous page)

```
    --fw-config build/stm32mp1/release/fdts/fw-config.dtb \
    --trusted-key-cert build/stm32mp1/release/trusted_key.crt \
    --tos-fw-cert build/stm32mp1/release/tos_fw_content.crt \
    --tos-fw-key-cert build/stm32mp1/release/tos_fw_key.crt \
    --nt-fw-cert build/stm32mp1/release/nt_fw_content.crt \
    --nt-fw-key-cert build/stm32mp1/release/nt_fw_key.crt \
    --stm32mp-cfg-cert build/stm32mp1/release/stm32mp_cfg_cert.crt \
    build/stm32mp1/release/stm32mp1.fip
```


7.43.3 STM32MP2

STM32MP2 is a microprocessor designed by STMicroelectronics based on Arm Cortex-A35.

More information can be found on STM32MP2 Series page.

For TF-A common configuration of STM32 MPUs, please check *STM32 MPUs* page.

STM32MP2 Versions

Here are the variants for STM32MP2: - STM32MP21 - STM32MP23 - STM32MP25

STM32MP21 Versions

The STM32MP21 series is available in 3 different lines which are pin-to-pin compatible:

- STM32MP215: Single Cortex-A35 + Cortex-M33 - 2x Ethernet - 2x CAN FD - CSI - LTDC
- STM32MP213: Single Cortex-A35 + Cortex-M33 - 2x Ethernet - 2x CAN FD
- STM32MP211: Single Cortex-A35 + Cortex-M33 - 1x Ethernet

Each line comes with a security option (cryptography & secure boot) and a Cortex-A frequency option:

- A Basic + Cortex-A35 @ 1.2GHz
- C Secure Boot + HW Crypto + Cortex-A35 @ 1.2GHz
- D Basic + Cortex-A35 @ 1.5GHz
- F Secure Boot + HW Crypto + Cortex-A35 @ 1.5GHz

STM32MP23 Versions

The STM32MP23 series is available in 3 different lines which are pin-to-pin compatible:

- STM32MP235: Dual Cortex-A35 + Cortex-M33 - 2x Ethernet - 2x CAN FD - H264 - 3D GPU - AI / NN - LVDS / DSI
- STM32MP233: Dual Cortex-A35 + Cortex-M33 - 2x Ethernet - 2x CAN FD
- STM32MP231: Single Cortex-A35 + Cortex-M33 - 1x Ethernet

Each line comes with a security option (cryptography & secure boot) and a Cortex-A frequency option:

- A Basic + Cortex-A35 @ 1.2GHz
- C Secure Boot + HW Crypto + Cortex-A35 @ 1.2GHz
- D Basic + Cortex-A35 @ 1.5GHz
- F Secure Boot + HW Crypto + Cortex-A35 @ 1.5GHz

STM32MP25 Versions

The STM32MP25 series is available in 4 different lines which are pin-to-pin compatible:

- STM32MP257: Dual Cortex-A35 cores, Cortex-M33 core - 3x Ethernet (2+1 switch) - 3x CAN FD – H264 - 3D GPU – AI / NN - LVDS
- STM32MP255: Dual Cortex-A35 cores, Cortex-M33 core - 2x Ethernet – 3x CAN FD - H264 - 3D GPU – AI / NN - LVDS
- STM32MP253: Dual Cortex-A35 cores, Cortex-M33 core - 2x Ethernet – 3x CAN FD - LVDS
- STM32MP251: Single Cortex-A35 core, Cortex-M33 core - 1x Ethernet

Each line comes with a security option (cryptography & secure boot) and a Cortex-A frequency option:

- A Basic + Cortex-A35 @ 1.2GHz
- C Secure Boot + HW Crypto + Cortex-A35 @ 1.2GHz
- D Basic + Cortex-A35 @ 1.5GHz
- F Secure Boot + HW Crypto + Cortex-A35 @ 1.5GHz

The STM32MP2 part number codification page gives more information about part numbers.

Memory mapping

```
0x00000000 +-----------------+
           |                 |
           |       ...       |
           |                 |
0x0E000000 +-----------------+ \
           |       BL31      | |
           +-----------------+ |
           |       ...       | |
0x0E012000 +-----------------+ |
           |     BL2 DTB     | | Embedded SRAM
0x0E016000 +-----------------+ |
           |       BL2       | |
0x0E040000 +-----------------+ /
           |                 |
           |       ...       |
           |                 |
0x40000000 +-----------------+
           |                 |
           |                 |   Devices
           |                 |
0x80000000 +-----------------+ \
           |                 | |
           |                 | | Non-secure RAM (DDR)
           |                 | |
0xFFFFFFFF +-----------------+ /
```

Build Instructions

STM32MP2x specific flags

Dedicated STM32MP2 build flags:

- `STM32MP_DDR_FIP_IO_STORAGE`: to store DDR firmware in FIP.
  Default: 1
- `STM32MP21`: to select STM32MP21 variant configuration.
  Default: 0
- `STM32MP23`: to select STM32MP23 variant configuration.
  Default: 0
- `STM32MP25`: to select STM32MP25 variant configuration.
  Default: 1

To compile the correct DDR driver, one flag must be set among:

- `STM32MP_DDR3_TYPE`: to compile DDR3 driver and DT.
  Default: 0

- `STM32MP_DDR4_TYPE`: to compile DDR4 driver and DT.
  Default: 0
- `STM32MP_LPDDR4_TYPE`: to compile LpDDR4 driver and DT.
  Default: 0

Boot with FIP

You need to build BL2, BL31, BL32 (OP-TEE) and BL33 (U-Boot) and retrieve DDR PHY firmware before building FIP binary.

U-Boot

```
cd <u-boot_directory>
make stm32mp25_defconfig
make DEVICE_TREE=stm32mp257f-ev1 all
```

OP-TEE

```
cd <optee_directory>
make CROSS_COMPILE64=aarch64-none-elf- CROSS_COMPILE32=arm-none-eabi-
    ARCH=arm PLATFORM=stm32mp2 \
    CFG_EMBED_DTB_SOURCE_FILE=stm32mp257f-ev1.dts
```

DDR PHY firmware

DDR PHY firmware files may not be delivered inside TF-A repository, especially if you build directly from trustedfirmware.org repository. It then needs to be retrieved from STMicroelectronics DDR PHY github.

You can either clone the repository to the default directory:

```
git clone https://github.com/STMicroelectronics/stm32-ddr-phy-binary.git␣
↪drivers/st/ddr/phy/firmware/bin
```

Or clone it somewhere else, and add `STM32MP_DDR_FW_PATH=` in your make command line when building FIP.

TF-A BL2

To build TF-A BL2 with its STM32 header for SD-card boot:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=stm32mp2 \
    STM32MP_DDR4_TYPE=1 SPD=opteed \
    DTB_FILE_NAME=stm32mp257f-ev1.dtb STM32MP_SDMMC=1
```

For other boot devices, you have to replace STM32MP_SDMMC in the previous command with the desired device flag.

FIP

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=stm32mp2 \
    STM32MP_DDR4_TYPE=1 SPD=opteed \
    DTB_FILE_NAME=stm32mp257f-ev1.dtb \
    BL33=<u-boot_directory>/u-boot-nodtb.bin \
    BL33_CFG=<u-boot_directory>/u-boot.dtb \
    BL32=<optee_directory>/tee-header_v2.bin \
    BL32_EXTRA1=<optee_directory>/tee-pager_v2.bin
    fip
```

7.44 Texas Instruments K3

Trusted Firmware-A (TF-A) implements the EL3 firmware layer for Texas Instruments K3 SoCs.

7.44.1 Boot Flow

```
R5(U-Boot) --> TF-A BL31 --> BL32(OP-TEE) --> TF-A BL31 --> BL33(U-Boot) -->␣
↪Linux
                                                           \
                                                    Optional direct to Linux boot
                                                               \
                                                                --> BL33(Linux)
```

Texas Instruments K3 SoCs contain an R5 processor used as the boot master, it loads the needed images for A53 startup, because of this we do not need BL1 or BL2 TF-A stages.

7.44.2 Build Instructions

https://github.com/ARM-software/arm-trusted-firmware.git

TF-A:

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=k3 SPD=opteed all
```

OP-TEE:

```
make ARCH=arm CROSS_COMPILE64=aarch64-linux-gnu- PLATFORM=k3 CFG_ARM64_core=y␣
↪all
```

R5 U-Boot:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- am65x_evm_r5_defconfig
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- SYSFW=<path to SYSFW>
```

A53 U-Boot:

```
make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- am65x_evm_a53_defconfig
make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- ATF=<path> TEE=<path>
```

7.44.3 Deploy Images

```
cp tiboot3.bin tispl.bin u-boot.img /sdcard/boot/
```

7.45 Texas Instruments K3low

Trusted Firmware-A (TF-A) implements the EL3 firmware layer for Texas Instruments K3low SoCs. The first generation of K3 architecture based devices used a Cortex R5 processor as a primary boot loader. But the K3low SoC architecture removed the R5 processor and the MPU A53 acts as the primary bootloader. Due to the small static memory foot print on the k3low devices, a two stage boot load architecture is used. In the first stage BL1 boot loader configures and initializes the DDR4/LPDDR4 subsystem and resets back to the ROMCode for second stage booting of BL31 etc.

7.45.1 Boot Flow

```
TF-A BL1 --> TF-A BL31 --> BL32(OP-TEE) --> TF-A BL31 --> BL33(U-Boot) -->␣
↪Linux
                                                  \
                                              Optional direct to Linux boot
                                                      \
                                                      --> BL33(Linux)
```

7.45.2 Build Instructions

TF-A:

```
make CROSS_COMPILE=aarch64-none-linux-gnu- ARCH=aarch64 PLAT=k3low TARGET_
↪BOARD=am62lx-evm

For bl1 only binary builds:
```

```
make CROSS_COMPILE=aarch64-none-linux-gnu- ARCH=aarch64 PLAT=k3low TARGET_
↪BOARD=am62lx-evm am62l_bl1
```

OP-TEE:

```
make ARCH=arm CROSS_COMPILE64=aarch64-none-linux-gnu- PLATFORM=k3 CFG_ARM64_
↪core=y all
```

A53 U-Boot:

```
make ARCH=arm CROSS_COMPILE=aarch64-none-linux-gnu- am62l_evm_defconfig
make ARCH=arm CROSS_COMPILE=aarch64-none-linux-gnu- BL1=<path> BL31=<path>␣
↪TEE=<path> BINMAN_INDIRS=<path>
```

7.45.3 Deploy Images

```
cp tiboot3.bin tispl.bin u-boot.img /sdcard/boot/
```

7.46 Xilinx Versal NET

Trusted Firmware-A implements the EL3 firmware layer for Xilinx Versal NET. The platform only uses the runtime part of TF-A as Xilinx Versal NET already has a BootROM (BL1) and PMC FW (BL2).

BL31 is TF-A. BL32 is an optional Secure Payload. BL33 is the non-secure world software (U-Boot, Linux etc).

To build: `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal_net bl31 ` ``

To build bl32 TSP you have to rebuild bl31 too `` `bash make CROSS_COMPILE=aarch64-none-elf- PLAT=versal_net SPD=tspd RESET_TO_BL31=1 bl31 bl32 ` ``

To build TF-A for JTAG DCC console: `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal_net VERSAL_NET_CONSOLE=dcc bl31 ` ``

To build TF-A with SDEI_SUPPORT: `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal_net SDEI_SUPPORT=1 bl31 ` ``

7.46.1 Xilinx Versal NET platform specific build options

- *VERSAL_NET_ATF_MEM_BASE*: Specifies the base address of the bl31 binary.
- *VERSAL_NET_ATF_MEM_SIZE*: Specifies the size of the memory region of the bl31 binary.
- *VERSAL_NET_BL32_MEM_BASE*: Specifies the base address of the bl32 binary.
- *VERSAL_NET_BL32_MEM_SIZE*: Specifies the size of the memory region of the bl32 binary.
- *VERSAL_NET_CONSOLE*: Select the console driver. Options: - *pl011*, *pl011_0*: ARM pl011 UART 0 (default) - *pl011_1* : ARM pl011 UART 1 - *dcc* : JTAG Debug Communication Channel(DCC)
- *TFA_NO_PM* : Platform Management support. - 0 : Enable Platform Management (Default) - 1 : Disable Platform Management
- ***CPU_PWRDWN_SGI*: Select the SGI for triggering CPU power down request to**

  secondary cores on receiving power down callback from firmware. Options:
  - *0* : SGI 0
  - *1* : SGI 1
  - *2* : SGI 2
  - *3* : SGI 3
  - *4* : SGI 4
  - *5* : SGI 5
  - *6* : SGI 6 (Default)
  - *7* : SGI 7

7.46.2 Configurable APU IPI ID

The APU IPI ID in TF-A for the Versal NET platform is configurable as per the design.

To build PLAT_IPI_ID_APU:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=versal_net RESET_TO_BL31=1 bl31␣
↪PLAT_IPI_ID_APU=<value>

$(eval $(call add_define,PLAT_IPI_ID_APU))
```

7.46.3 Reference DEN0028E SMC calling convention

7.46.4 Allocated subranges of Function Identifier to SIP services

SMC Function	Identifier Service type
0xC2000000-0xC200FFFF	Fast SMC64 SiP Service Calls as per SMCCC Section 6.1

7.46.5 IPI SMC call ranges

SMC Function Identifier	Service type
0xc2001000-0xc2001FFF	Fast SMC64 SiP Service call range used for AMD-Xilinx IPI

7.46.6 PM SMC call ranges for SiP SVC version 0.1

SMC Function Identifier	Service type
0xc2000000-0xc2000FFF	Fast SMC64 SiP Service call range used for AMD-Xilinx Platform Management

7.46.7 PM SMC call ranges for SiP SVC version 0.2

SMC Function Identifier	Service type
0xc2000FFF	Fast SMC64 SiP Service call used for pass-through of AMD-Xilinx Platform Management APIs to firmware
0xc2000A00-0xc2000AFF	Fast SMC64 SiP Service call range used for AMD-Xilinx Platform Management specific TF-A APIs

7.46.8 SMC function IDs for SiP Service queries

Service	Call UID	Revision
SiP Service	0x8200_FF01	0x8200_FF03

Call UID Query – Returns a unique identifier of the service provider.

Revision Query – Returns revision details of the service implementor.

7.47 Xilinx Versal

Trusted Firmware-A implements the EL3 firmware layer for Xilinx Versal. The platform only uses the runtime part of TF-A as Xilinx Versal already has a BootROM (BL1) and PMC FW (BL2).

BL31 is TF-A. BL32 is an optional Secure Payload. BL33 is the non-secure world software (U-Boot, Linux etc).

To build: `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal bl31 ` ``

To build bl32 TSP you have to rebuild bl31 too `` `bash make CROSS_COMPILE=aarch64-none-elf- PLAT=versal SPD=tspd RESET_TO_BL31=1 bl31 bl32 ` ``

To build TF-A for JTAG DCC console `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal bl31 VERSAL_CONSOLE=dcc ` ``

To build TF-A with Errata management interface `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal bl31 ERRATA_ABI_SUPPORT=1 ` ``

To build TF-A with Straight-Line Speculation(SLS) `` `bash make RESET_TO_BL31=1 CROSS_COMPILE=aarch64-none-elf- PLAT=versal bl31 HARDEN_SLS_ALL=1 ` ``

7.47.1 Xilinx Versal platform specific build options

- *VERSAL_ATF_MEM_BASE*: Specifies the base address of the bl31 binary.
- *VERSAL_ATF_MEM_SIZE*: Specifies the size of the memory region of the bl31 binary.
- *VERSAL_BL32_MEM_BASE*: Specifies the base address of the bl32 binary.
- *VERSAL_BL32_MEM_SIZE*: Specifies the size of the memory region of the bl32 binary.
- *VERSAL_CONSOLE*: Select the console driver. Options: - *pl011*, *pl011_0*: ARM pl011 UART 0 - *pl011_1* : ARM pl011 UART 1
- ***CPU_PWRDWN_SGI*: Select the SGI for triggering CPU power down request to**
  secondary cores on receiving power down callback from firmware. Options:
  - *0* : SGI 0
  - *1* : SGI 1
  - *2* : SGI 2
  - *3* : SGI 3
  - *4* : SGI 4
  - *5* : SGI 5
  - *6* : SGI 6 (Default)
  - *7* : SGI 7

7.47.2 Configurable Stack Size

The stack size in TF-A for the Versal platform is configurable. The custom package can define the desired stack size as per the requirement in the makefile as follows:

```
PLATFORM_STACK_SIZE := <value>

$(eval $(call add_define,PLATFORM_STACK_SIZE))
```

7.47.3 Configurable APU IPI ID

The APU IPI ID in TF-A for the Versal platform is configurable as per the design.

To build PLAT_IPI_ID_APU:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=versal RESET_TO_BL31=1 bl31 PLAT_
↪IPI_ID_APU=<value>

$(eval $(call add_define,PLAT_IPI_ID_APU))
```

7.47.4 CUSTOM SIP Service Support

- Dedicated SMC FID `SOC_SIP_SVC_CUSTOM(0x82002000)` (32-bit) / `(0xC2002000)` (64-bit) is used by a custom package for providing CUSTOM SIP service.
- By default, the platform provides a bare minimum definition for `custom_smc_handler` in this service.
- To use this service, the custom package should implement its own SMC handler named `custom_smc_handler`. Once the custom package is included in the TF-A build, its definition of `custom_smc_handler` is enabled.

7.47.5 Custom Package Makefile Fragment Inclusion in TF-A Build

- Custom package is not directly part of the TF-A source.
- `<CUSTOM_PKG_PATH>` is the location where the user clones a custom package locally.
- The custom package must implement a makefile fragment named `custom_pkg.mk` so it can be included in the TF-A build.
- `custom_pkg.mk` should specify all the rules to include custom package specific header files, dependent libraries, and source files that are required to be part of the TF-A build.
- When `<CUSTOM_PKG_PATH>` is specified in the TF-A build command, `custom_pkg.mk` is included from `<CUSTOM_PKG_PATH>`.
- Example TF-A build command:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=versal RESET_TO_BL31=1 bl31 CUSTOM_
↪PKG_PATH=<...>
```

7.47.6 # PLM->TF-A Parameter Passing

The PLM populates a data structure with image information for the TF-A. The TF-A uses that data to hand off to the loaded images. The address of the handoff data structure is passed in the `` `PMC_GLOBAL_GLOB_GEN_STORAGE4` `` register. The register is free to be used by other software once the TF-A is bringing up further firmware images.

7.47.7 Reference DEN0028E SMC calling convention

7.47.8 Allocated subranges of Function Identifier to SIP services

SMC Function	Identifier Service type
0xC2000000-0xC200FFFF	Fast SMC64 SiP Service Calls as per SMCCC Section 6.1

7.47.9 IPI SMC call ranges

SMC Function Identifier	Service type
0xc2001000-0xc2001FFF	Fast SMC64 SiP Service call range used for AMD-Xilinx IPI

7.47.10 PM SMC call ranges for SiP SVC version 0.1

SMC Function Identifier	Service type
0xc2000000-0xc2000FFF	Fast SMC64 SiP Service call range used for AMD-Xilinx Platform Management

7.47.11 PM SMC call ranges for SiP SVC version 0.2

SMC Function Identifier	Service type
0xc2000FFF	Fast SMC64 SiP Service call used for pass-through of AMD-Xilinx Platform Management APIs to firmware
0xc2000A00-0xc2000AFF	Fast SMC64 SiP Service call range used for AMD-Xilinx Platform Management specific TF-A APIs

7.47.12 SMC function IDs for SiP Service queries

Service	Call UID	Revision
SiP Service	0x8200_FF01	0x8200_FF03

Call UID Query – Returns a unique identifier of the service provider.

Revision Query – Returns revision details of the service implementor.

7.48 Xilinx Zynq UltraScale+ MPSoC

Trusted Firmware-A (TF-A) implements the EL3 firmware layer for Xilinx Zynq UltraScale + MPSoC. The platform only uses the runtime part of TF-A as ZynqMP already has a BootROM (BL1) and FSBL (BL2).

BL31 is TF-A. BL32 is an optional Secure Payload. BL33 is the non-secure world software (U-Boot, Linux etc).

To build:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=zynqmp RESET_TO_BL31=1 bl31
```

To build bl32 TSP you have to rebuild bl31 too:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=zynqmp SPD=tspd RESET_TO_BL31=1␣
↪bl31 bl32
```

To build TF-A for JTAG DCC console:

```
make CROSS_COMPILE=aarch64-none-elf- PLAT=zynqmp RESET_TO_BL31=1 bl31 ZYNQMP_
↪CONSOLE=dcc
```

7.48.1 ZynqMP platform specific build options

- `XILINX_OF_BOARD_DTB_ADDR` : Specifies the base address of Device tree.
- `ZYNQMP_ATF_MEM_BASE`: Specifies the base address of the bl31 binary.
- `ZYNQMP_ATF_MEM_SIZE`: Specifies the size of the memory region of the bl31 binary.
- `ZYNQMP_BL32_MEM_BASE`: Specifies the base address of the bl32 binary.
- `ZYNQMP_BL32_MEM_SIZE`: Specifies the size of the memory region of the bl32 binary.
- `ZYNQMP_CONSOLE`: Select the console driver. Options:
  - `cadence`, `cadence0`: Cadence UART 0
  - `cadence1` : Cadence UART 1

7.48.2 ZynqMP Debug behavior

With DEBUG=1, TF-A for ZynqMP uses DDR memory range instead of OCM memory range due to size constraints. For DEBUG=1 configuration for ZynqMP the BL31_BASE is set to the DDR location of 0x1000 and BL31_LIMIT is set to DDR location of 0x7FFFF. By default the above memory range will NOT be reserved in device tree.

To reserve the above memory range in device tree, the device tree base address must be provided during build as,

make CROSS_COMPILE=aarch64-none-elf- PLAT=zynqmp RESET_TO_BL31=1 DEBUG=1
XILINX_OF_BOARD_DTB_ADDR=<DTB address> bl31

The default DTB base address for ZynqMP platform is 0x100000. This default value is not set in the code and to use this default address, user still needs to provide it through the build command as above.

If the user wants to move the bl31 to a different DDR location, user can provide the DDR address location using the build time parameters ZYNQMP_ATF_MEM_BASE and ZYNQMP_ATF_MEM_SIZE.

The DDR address must be reserved in the DTB by the user, either by manually adding the reserved memory node, in the device tree, with the required address range OR let TF-A modify the device tree on the run.

To let TF-A access and modify the device tree, the DTB address must be provided to the build command as follows,

make CROSS_COMPILE=aarch64-none-elf- PLAT=zynqmp RESET_TO_BL31=1 DEBUG=1
ZYNQMP_ATF_MEM_BASE=<DDR address> ZYNQMP_ATF_MEM_SIZE=<size> XILINX_OF_BOARD_DTB_ADDR=<DTB address> bl31

7.48.3 DDR Address Range Usage

When FSBL runs on RPU and TF-A is to be placed in DDR address range, then the user needs to make sure that the DDR address is beyond 256KB. In the RPU view, the first 256 KB is TCM memory.

For this use case, with the minimum base address in DDR for TF-A, the build command example is;

make CROSS_COMPILE=aarch64-none-elf- PLAT=zynqmp RESET_TO_BL31=1 DEBUG=1
ZYNQMP_ATF_MEM_BASE=0x40000 ZYNQMP_ATF_MEM_SIZE=<size>

7.48.4 Configurable Stack Size

The stack size in TF-A for ZynqMP platform is configurable. The custom package can define the desired stack size as per the requirement in the make file as follows,

PLATFORM_STACK_SIZE := <value> $(eval $(call add_define,PLATFORM_STACK_SIZE))

7.48.5 FSBL->TF-A Parameter Passing

The FSBL populates a data structure with image information for TF-A. TF-A uses that data to hand off to the loaded images. The address of the handoff data structure is passed in the `PMU_GLOBAL.GLOBAL_GEN_STORAGE6` register. The register is free to be used by other software once TF-A has brought up further firmware images.

7.48.6 Power Domain Tree

The following power domain tree represents the power domain model used by TF-A for ZynqMP:

```
                +-+
                |0|
                +-+
 +-------+---+---+-------+
 |       |       |       |
 |       |       |       |
 v       v       v       v
+-+     +-+     +-+     +-+
|0|     |1|     |2|     |3|
+-+     +-+     +-+     +-+
```

The 4 leaf power domains represent the individual A53 cores, while resources common to the cluster are grouped in the power domain on the top.

7.48.7 CUSTOM SIP service support

- Dedicated SMC FID ZYNQMP_SIP_SVC_CUSTOM(0x82002000)(32-bit)/ (0xC2002000)(64-bit) to be used by a custom package for providing CUSTOM SIP service.
- by default platform provides bare minimum definition for custom_smc_handler in this service.
- to use this service, custom package should implement their smc handler with the name custom_smc_handler. once custom package is included in TF-A build, their definition of custom_smc_handler is enabled.

7.48.8 Custom package makefile fragment inclusion in TF-A build

- custom package is not directly part of TF-A source.
- <CUSTOM_PKG_PATH> is the location at which user clones a custom package locally.
- custom package needs to implement makefile fragment named custom_pkg.mk so as to get included in TF-A build.
- custom_pkg.mk specify all the rules to include custom package specific header files, dependent libs, source files that are supposed to be included in TF-A build.
- when <CUSTOM_PKG_PATH> is specified in TF-A build command, custom_pkg.mk is included from <CUSTOM_PKG_PATH> in TF-A build.

- TF-A build command: make CROSS_COMPILE=aarch64-none-elf- PLAT=zynqmp RESET_TO_BL31=1 bl31 CUSTOM_PKG_PATH=<…>

7.48.9 Reference DEN0028E SMC calling convention

7.48.10 Allocated subranges of Function Identifier to SIP services

SMC Function	Identifier Service type
0xC2000000-0xC200FFFF	Fast SMC64 SiP Service Calls as per SMCCC Section 6.1

7.48.11 IPI SMC call ranges

SMC Function Identifier	Service type
0xc2001000-0xc2001FFF	Fast SMC64 SiP Service call range used for AMD-Xilinx IPI

7.48.12 PM SMC call ranges

SMC Function Identifier	Service type
0xc2000000-0xc2000FFF	Fast SMC64 SiP Service call range used for AMD-Xilinx Platform Management

7.48.13 SMC function IDs for SiP Service queries

Service	Call UID	Revision
SiP Service	0x8200_FF01	0x8200_FF03

Call UID Query – Returns a unique identifier of the service provider.

Revision Query – Returns revision details of the service implementor.

7.48.14 CUSTOM SIP service support

Service	32-bit	64-bit
SiP Service	0x82002000	0xC2002000

7.49 Broadcom Stingray

7.49.1 Description

Broadcom's Stingray(BCM958742t) is a multi-core processor with 8 Cortex-A72 cores. Trusted Firmware-A (TF-A) is used to implement secure world firmware, supporting BL2 and BL31 for Broadcom Stingray SoCs.

On Poweron, Boot ROM will load bl2 image and Bl2 will initialize the hardware, then loads bl31 and bl33 into DDR and boots to bl33.

7.49.2 Boot Sequence

Bootrom –> TF-A BL2 –> TF-A BL31 –> BL33(u-boot)

Code Locations

- Trusted Firmware-A: link

7.49.3 How to build

Build Procedure

- Prepare AARCH64 toolchain.
- Build u-boot first, and get the binary image: u-boot.bin,
- Build TF-A

  Build fip:

```
make CROSS_COMPILE=aarch64-linux-gnu- PLAT=stingray BOARD_CFG=bcm958742t all fip BL33=u-boot.bin
```

Deploy TF-A Images

The u-boot will be upstreamed soon, this doc will be updated once they are ready, and the link will be posted.

This section provides a list of supported upstream *platform ports* and the documentation associated with them.

Note: In addition to the platforms ports listed within the table of contents, there are several additional platforms that are supported upstream but which do not currently have associated documentation:

- Arm Neoverse N1 System Development Platform (N1SDP)
- MediaTek MT8173 SoCs

7.50 Deprecated platforms

Platform	Vendor	Deprecated version	Deleted version
RD1AE	Arm	2.14	TBD
TC3	Arm	2.15	2.15

CHAPTER

EIGHT

PERFORMANCE & TESTING

8.1 PSCI Performance Measurement

TF-A provides two instrumentation tools for performing analysis of the PSCI implementation:

- PSCI STAT
- Runtime Instrumentation

This page explains how they may be enabled and used to perform all varieties of analysis.

8.1.1 Performance Measurement Framework

The Performance Measurement Framework *PMF* is a framework that provides mechanisms for collecting and retrieving timestamps at runtime from the Performance Measurement Unit (*PMU*). The PMU is a generalized abstraction for accessing CPU hardware registers used to measure hardware events. This means, for instance, that the PMU might be used to place instrumentation points at logical locations in code for tracing purposes.

TF-A utilises the PMF as a backend for the two instrumentation services it provides–PSCI Statistics and Runtime Instrumentation. The PMF is used by these services to facilitate collection and retrieval of timestamps. For instance, the PSCI Statistics service registers the PMF service `psci_svc` to track its residency statistics.

This is reserved a unique ID, name, and space in memory by the PMF. The framework provides a convenient interface for PSCI Statistics to retrieve values from `psci_svc` at runtime. Alternatively, the service may be configured such that the PMF dumps those values to the console. A platform may choose to expose SMCs that allow retrieval of these timestamps from the service.

This feature is enabled with the Boolean flag `ENABLE_PMF`.

8.1.2 PSCI Statistics

PSCI Statistics is a runtime service that provides residency statistics for power states used by the platform. The service tracks residency time and entry count. Residency time is the total time spent in a particular power state by a PE. The entry count is the number of times the PE has entered the power state. PSCI Statistics implements the optional functions `PSCI_STAT_RESIDENCY` and `PSCI_STAT_COUNT` from the PSCI specification.

PSCI_STAT_RESIDENCY

Parameters

- **target_cpu** – Contains copy of affinity fields in the MPIDR register for identifying the target core (See section 5.1.4 of PSCI specifications for more details).
- **power_state** – identifier for a specific local state. Generally, this parameter takes the same form as the power_state parameter described for CPU_SUSPEND in section 5.4.2.

Returns

Time spent in `power_state`, in microseconds, by `target_cpu` and the highest level expressed in `power_state`.

PSCI_STAT_COUNT

Parameters

- **target_cpu** – follows the same format as `PSCI_STAT_RESIDENCY`.
- **power_state** – follows the same format as `PSCI_STAT_RESIDENCY`.

Returns

Number of times the state expressed in `power_state` has been used by `target_cpu` and the highest level expressed in `power_state`.

The implementation provides residency statistics only for low power states, and does this regardless of the entry mechanism into those states. The statistics it collects are set to 0 during shutdown or reset.

PSCI Statistics is enabled with the Boolean build flag `ENABLE_PSCI_STAT`. All Arm platforms utilise the PMF unless another collection backend is provided (`ENABLE_PMF` is implicitly enabled).

8.1.3 Runtime Instrumentation

The Runtime Instrumentation Service is an instrumentation tool that wraps around the PMF to provide timestamp data. Although the service is not restricted to PSCI, it is used primarily in TF-A to quantify the total time spent in the PSCI implementation. The tool can be used to instrument other components in TF-A as well. It is enabled with the Boolean flag `ENABLE_RUNTIME_INSTRUMENTATION`, and as with PSCI STAT, requires PMF to be enabled.

In PSCI, this service provides instrumentation points in the following code paths:

- Entry into the PSCI SMC handler
- Exit from the PSCI SMC handler

- Entry to low power state
- Exit from low power state
- Entry into cache maintenance operations in PSCI
- Exit from cache maintenance operations in PSCI

The service captures the cycle count, which allows for the time spent in the implementation to be calculated, given the frequency counter.

PSCI SMC Handler Instrumentation

The timestamp during entry into the handler is captured as early as possible during the runtime exception, prior to entry into the handler itself. All timestamps are stored in memory for later retrieval. The exit timestamp is captured after normal return from the PSCI SMC handler, or, if a low power state was requested, it is captured in the warm boot path.

8.2 PSCI Performance Measurements on Arm Juno Development Platform

This document summarises the findings of performance measurements of key operations in the Trusted Firmware-A Power State Coordination Interface (PSCI) implementation, using the in-built Performance Measurement Framework (PMF) and runtime instrumentation timestamps.

8.2.1 Method

We used the Juno R1 platform for these tests, which has 4 x Cortex-A53 and 2 x Cortex-A57 clusters running at the following frequencies:

Domain	Frequency (MHz)
Cortex-A57	900 (nominal)
Cortex-A53	650 (underdrive)
AXI subsystem	533

Juno supports CPU, cluster and system power down states, corresponding to power levels 0, 1 and 2 respectively. It does not support any retention states.

Given that runtime instrumentation using PMF is invasive, there is a small (unquantified) overhead on the results. PMF uses the generic counter for timestamps, which runs at 50MHz on Juno. Testing was done after the rc0 tag.

Please see the Runtime Instrumentation *Testing Methodology* page for more details. The tests were ran using the *tf-psci-lava-instr/juno-enable-runtime-instr,juno-instrumentation:juno-tftf* configuration in CI.

8.2.2 Results

CPU_SUSPEND to deepest power level

Table 1: CPU_SUSPEND latencies (ns) to deepest power level in parallel (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
1	0	104500 (-68.57%)	23400 (-91.35%)	5840 (-96.55%)
1	1	396120 (-36.57%)	18900 (-37.54%)	4660 (-97.21%)
1	2	540660 (+187.65%)	293920 (+1054.44%)	4520 (-95.00%)
1	3	986780 (+105.97%)	237680 (+1058.28%)	90840 (+3.82%)
0	0	686680 (-25.64%)	270720 (-7.97%)	168700 (+92.80%)
0	1	249100 (-77.48%)	27400 (-88.50%)	7480 (-91.44%)

Table 2: CPU_SUSPEND latencies (ns) to deepest power level in parallel (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	332440	270640 (+1031.44%)	169500 (+22.05%)
0	1	624520 (-1.01%)	30260 (-88.07%)	166740 (+21.76%)
1	0	187960 (+1.74%)	25460 (+9.93%)	90420 (+12.69%)
1	1	479100	20520 (+10.56%)	87500 (+14.38%)
1	2	923480 (-1.11%)	294160 (+1.58%)	87500 (+14.62%)
1	3	1106300	238320	87340 (+14.35%)

Table 3: CPU_SUSPEND latencies (ns) to deepest power level in serial (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
1	0	292320 (+9.48%)	23220 (-25.29%)	87860 (-47.86%)
1	1	100480 (-62.43%)	18260 (-40.48%)	4500 (-97.33%)
1	2	99760 (-65.75%)	18300 (-27.21%)	4480 (-94.85%)
1	3	99680 (-45.90%)	18580 (-19.71%)	4440 (-94.95%)
0	0	267300 (+44.86%)	27660 (+17.90%)	168320 (+92.32%)
0	1	267620 (+44.89%)	27860 (+19.88%)	168480 (+93.26%)

Table 4: `CPU_SUSPEND` latencies (ns) to deepest power level in serial (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	267000 (+9.39%)	31080 (+26.96%)	168520 (+22.44%)
0	1	267440 (+9.52%)	30680 (+28.69%)	168480 (+22.21%)
1	0	291300 (-1.18%)	25140 (+6.80%)	86980 (+13.52%)
1	1	184260 (+2.31%)	23140 (+9.46%)	87940 (+14.03%)
1	2	184520 (+2.20%)	23460 (+12.79%)	87520 (+14.02%)
1	3	184700 (+2.27%)	23240 (+9.62%)	87180 (+13.43%)

`CPU_SUSPEND` to power level 0

Table 5: `CPU_SUSPEND` latencies (ns) to power level 0 in parallel (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
1	0	540660 (-20.93%)	14960 (-33.69%)	8380 (-24.09%)
1	1	104540 (-87.40%)	16080 (-29.54%)	8220 (-28.40%)
1	2	250160 (+139.34%)	15240 (-11.40%)	8380 (-3.46%)
1	3	395020 (+58.52%)	15360 (-10.18%)	8040 (-5.19%)
0	0	685280 (+73.94%)	21000 (+20.14%)	11440 (+37.50%)
0	1	830900 (+54.01%)	21020 (+23.79%)	11380 (+37.11%)

Table 6: `CPU_SUSPEND` latencies (ns) to power level 0 in parallel (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	683780 (-2.74%)	22560 (+33.81%)	11040 (+38.35%)
0	1	829620 (-2.61%)	22820 (+39.15%)	11480 (+42.79%)
1	0	104520 (-74.34%)	17200 (+13.91%)	8680 (+20.56%)
1	1	249200 (+124.54%)	17100 (+10.61%)	8480 (+29.27%)
1	2	393980 (-28.95%)	17480 (+13.51%)	8320 (+19.88%)
1	3	539520 (+108.34%)	16980 (+9.13%)	8300 (+25.00%)

Table 7: CPU_SUSPEND latencies (ns) to power level 0 in serial (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
1	0	291180 (+188.01%)	15280 (-33.04%)	4460 (-39.40%)
1	1	96280 (-5.05%)	15140 (-33.36%)	4400 (-41.80%)
1	2	96380 (-66.93%)	15080 (-10.66%)	4460 (-2.62%)
1	3	96460	15620 (-7.35%)	4440 (-3.48%)
0	0	100940 (+4.00%)	21320 (+25.56%)	7340 (+58.19%)
0	1	101080 (+4.57%)	21320 (+25.71%)	7460 (+61.47%)

Table 8: CPU_SUSPEND latencies (ns) to power level 0 in serial (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	101100 (-4.73%)	22820 (+33.45%)	7360 (+39.92%)
0	1	101400 (-5.13%)	22720 (+33.18%)	7560 (+43.18%)
1	0	291440	16880 (+8.21%)	4580
1	1	96600 (-6.45%)	16860 (+9.20%)	4600 (+3.14%)
1	2	97060 (-6.40%)	16980 (+11.27%)	4640 (+3.11%)
1	3	96660 (-6.77%)	16960 (+7.89%)	4620 (+2.67%)

CPU_OFF on all non-lead CPUs

CPU_OFF on all non-lead CPUs in sequence then, CPU_SUSPEND on the lead core to the deepest power level.

Table 9: CPU_OFF latencies (ns) on all non-lead CPUs (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
1	0	185700 (-30.51%)	23240 (-29.45%)	88280 (-47.60%)
1	1	101440 (-62.06%)	22240 (-34.05%)	4260 (-97.47%)
1	2	101620 (-45.29%)	21340 (-15.05%)	4280 (-95.16%)
1	3	101460	21360 (-11.88%)	4240 (-7.83%)
0	0	267200 (+162.48%)	31020 (+34.52%)	168140 (+3508.15%)
0	1	266900 (+162.13%)	31980 (+37.02%)	168000 (+3520.69%)

Table 10: CPU_OFF latencies (ns) on all non-lead CPUs (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	267240 (+9.97%)	32940 (+24.68%)	168460 (+22.45%)
0	1	267340 (+9.46%)	33720 (+28.12%)	168500 (+22.21%)
1	0	185740 (+1.85%)	25120 (+6.17%)	88380 (+13.31%)
1	1	101940 (-5.77%)	24240 (+6.88%)	4600 (+4.07%)
1	2	101800 (-6.04%)	23060 (+6.17%)	4660 (+9.91%)
1	3	101820 (-5.91%)	23340 (+7.66%)	4640 (+6.91%)

CPU_VERSION in parallel

Table 11: CPU_VERSION latency (ns) in parallel on all cores (2.14)

Cluster	Core	Latency
1	0	680 (-43.33%)
1	1	1020 (-12.07%)
1	2	980 (+40.00%)
1	3	960 (-7.69%)
0	0	1380 (+35.29%)
0	1	1640 (+51.85%)

Table 12: CPU_VERSION latency (ns) in parallel on all cores (2.13)

Cluster	Core	Latency
0	0	1200 (+20.00%)
0	1	1160 (+9.43%)
1	0	700 (+16.67%)
1	1	1040 (+4.00%)
1	2	1020 (+4.08%)
1	3	1080 (+8.00%)

8.2.3 Annotated Historic Results

The following results are based on the upstream TF master as of 31/01/2017. TF-A was built using the same build instructions as detailed in the procedure above.

In the results below, CPUs 0-3 refer to CPUs in the little cluster (A53) and CPUs 4-5 refer to CPUs in the big cluster (A57). In all cases CPU 4 is the lead CPU.

PSCI_ENTRY corresponds to the powerdown latency, PSCI_EXIT the wakeup latency, and CFLUSH_OVERHEAD the latency of the cache flush operation.

CPU_SUSPEND to deepest power level on all CPUs in parallel

CPU	PSCI_ENTRY (us)	PSCI_EXIT (us)	CFLUSH_OVERHEAD (us)
0	27	20	5
1	114	86	5
2	202	58	5
3	375	29	94
4	20	22	6
5	290	18	206

A large variance in PSCI_ENTRY and PSCI_EXIT times across CPUs is observed due to TF PSCI lock contention. In the worst case, CPU 3 has to wait for the 3 other CPUs in the cluster (0-2) to complete PSCI_ENTRY and release the lock before proceeding.

The CFLUSH_OVERHEAD times for CPUs 3 and 5 are higher because they are the last CPUs in their respective clusters to power down, therefore both the L1 and L2 caches are flushed.

The CFLUSH_OVERHEAD time for CPU 5 is a lot larger than that for CPU 3 because the L2 cache size for the big cluster is lot larger (2MB) compared to the little cluster (1MB).

CPU_SUSPEND to power level 0 on all CPUs in parallel

CPU	PSCI_ENTRY (us)	PSCI_EXIT (us)	CFLUSH_OVERHEAD (us)
0	116	14	8
1	204	14	8
2	287	13	8
3	376	13	9
4	29	15	7
5	21	15	8

There is no lock contention in TF generic code at power level 0 but the large variance in PSCI_ENTRY times across CPUs is due to lock contention in Juno platform code. The platform lock is used to mediate access to a single SCP communication channel. This is compounded by the SCP firmware waiting for each AP CPU to enter WFI before making the channel available to other CPUs, which effectively serializes the SCP power down commands from all CPUs.

On platforms with a more efficient CPU power down mechanism, it should be possible to make the PSCI_ENTRY times smaller and consistent.

The PSCI_EXIT times are consistent across all CPUs because TF does not require locks at power level 0.

The CFLUSH_OVERHEAD times for all CPUs are small and consistent since only the cache associated with power level 0 is flushed (L1).

CPU_SUSPEND to deepest power level on all CPUs in sequence

CPU	PSCI_ENTRY (us)	PSCI_EXIT (us)	CFLUSH_OVERHEAD (us)
0	114	20	94
1	114	20	94
2	114	20	94
3	114	20	94
4	195	22	180
5	21	17	6

The CFLUSH_OVERHEAD times for lead CPU 4 and all CPUs in the non-lead cluster are large because all other CPUs in the cluster are powered down during the test. The CPU_SUSPEND call powers down to the cluster level, requiring a flush of both L1 and L2 caches.

The CFLUSH_OVERHEAD time for CPU 4 is a lot larger than those for the little CPUs because the L2 cache size for the big cluster is lot larger (2MB) compared to the little cluster (1MB).

The PSCI_ENTRY and CFLUSH_OVERHEAD times for CPU 5 are low because lead CPU 4 continues to run while CPU 5 is suspended. Hence CPU 5 only powers down to level 0, which only requires L1 cache flush.

CPU_SUSPEND to power level 0 on all CPUs in sequence

CPU	PSCI_ENTRY (us)	PSCI_EXIT (us)	CFLUSH_OVERHEAD (us)
0	22	14	5
1	22	14	5
2	21	14	5
3	22	14	5
4	17	14	6
5	18	15	6

Here the times are small and consistent since there is no contention and it is only necessary to flush the cache to power level 0 (L1). This is the best case scenario.

The PSCI_ENTRY times for CPUs in the big cluster are slightly smaller than for the CPUs in little cluster due to greater CPU performance.

The PSCI_EXIT times are generally lower than in the last test because the cluster remains powered on throughout the test and there is less code to execute on power on (for example, no need to enter CCI coherency)

`CPU_OFF` on all non-lead CPUs in sequence then `CPU_SUSPEND` on lead CPU to deepest power level

The test sequence here is as follows:

1. Call `CPU_ON` and `CPU_OFF` on each non-lead CPU in sequence.
2. Program wake up timer and suspend the lead CPU to the deepest power level.
3. Call `CPU_ON` on non-lead CPU to get the timestamps from each CPU.

CPU	`PSCI_ENTRY` (us)	`PSCI_EXIT` (us)	`CFLUSH_OVERHEAD` (us)
0	110	28	93
1	110	28	93
2	110	28	93
3	111	28	93
4	195	22	181
5	20	23	6

The `CFLUSH_OVERHEAD` times for all little CPUs are large because all other CPUs in that cluster are powerered down during the test. The `CPU_OFF` call powers down to the cluster level, requiring a flush of both L1 and L2 caches.

The `PSCI_ENTRY` and `CFLUSH_OVERHEAD` times for CPU 5 are small because lead CPU 4 is running and CPU 5 only powers down to level 0, which only requires an L1 cache flush.

The `CFLUSH_OVERHEAD` time for CPU 4 is a lot larger than those for the little CPUs because the L2 cache size for the big cluster is lot larger (2MB) compared to the little cluster (1MB).

The `PSCI_EXIT` times for CPUs in the big cluster are slightly smaller than for CPUs in the little cluster due to greater CPU performance. These times generally are greater than the `PSCI_EXIT` times in the `CPU_SUSPEND` tests because there is more code to execute in the "on finisher" compared to the "suspend finisher" (for example, GIC redistributor register programming).

`PSCI_VERSION` on all CPUs in parallel

Since very little code is associated with `PSCI_VERSION`, this test approximates the round trip latency for handling a fast SMC at EL3 in TF.

CPU	TOTAL TIME (ns)
0	3020
1	2940
2	2980
3	3060
4	520
5	720

The times for the big CPUs are less than the little CPUs due to greater CPU performance.

We suspect the time for lead CPU 4 is shorter than CPU 5 due to subtle cache effects, given that these measurements are at the nano-second level.

8.3 Runtime Instrumentation Testing - N1SDP

For this test we used the N1 System Development Platform (N1SDP), which contains an SoC consisting of two dual-core Arm N1 clusters. Testing was done after the rc0 tag.

Please see the Runtime Instrumentation *Testing Methodology* page for more details. The tests were ran using the *tf-psci-lava-instr/n1sdp-runtime-instrumentation,n1sdp-runtime-instrumentation:n1sdp-fip.tftf-firmware* configuration in CI.

8.3.1 Results

`CPU_SUSPEND` to deepest power level

Table 13: `CPU_SUSPEND` latencies (ns) to deepest power level in parallel (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1980 (-41.42%)	9420 (-24.52%)	400 (+17.65%)
1	0	2420 (-36.98%)	10460 (-41.69%)	180 (-18.18%)
0	0	3240 (+26.56%)	15060 (-12.14%)	140 (-12.50%)
1	0	2120 (-30.26%)	17020 (+70.88%)	140

Table 14: `CPU_SUSPEND` latencies (ns) to deepest power level in parallel (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	3380 (+1.81%)	12480 (-7.00%)	340 (+21.43%)
0	1	3840 (-4.48%)	17940 (-1.10%)	220 (-21.43%)
1	0	2560 (-5.19%)	17140 (-1.38%)	160 (-38.46%)
1	1	3040 (+2.70%)	9960 (-8.29%)	140 (-46.15%)

Table 15: `CPU_SUSPEND` latencies (ns) to deepest power level in serial (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1620	9340 (-7.89%)	280 (-6.67%)
1	0	1580 (-20.20%)	9060 (-13.22%)	140 (-50.00%)
0	0	1880 (+14.63%)	9580 (-7.35%)	160
1	0	1740 (-23.68%)	9640 (-8.71%)	140 (-76.67%)

Table 16: `CPU_SUSPEND` latencies (ns) to deepest power level in serial (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1640 (+1.23%)	10140	300
0	1	1980 (+6.45%)	10440 (-1.69%)	280
1	0	1640 (-8.89%)	10340 (+1.77%)	160 (-50.00%)
1	1	2280 (+5.56%)	10560	600 (+20.00%)

`CPU_SUSPEND` to power level 0

Table 17: `CPU_SUSPEND` latencies (ns) to power level 0 in parallel (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1440 (+4.35%)	10460	240
1	0	1780 (+7.23%)	13900 (-12.47%)	140
0	0	1740 (+6.10%)	14520 (-4.47%)	140
1	0	1700 (-4.49%)	9280 (-6.07%)	140

Table 18: `CPU_SUSPEND` latencies (ns) to power level 0 in parallel (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1380 (+72.50%)	10600 (-11.52%)	240 (-7.69%)
0	1	1660 (+7.79%)	15880 (-3.41%)	140 (-46.15%)
1	0	1640 (+1.23%)	15200 (-5.59%)	140 (-53.33%)
1	1	1780 (-1.11%)	9880 (-6.26%)	140 (-30.00%)

Table 19: CPU_SUSPEND latencies (ns) to power level 0 in serial (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1440 (-7.69%)	9220 (-4.75%)	280 (-6.67%)
1	0	1580 (-16.84%)	9040 (-12.91%)	180 (-35.71%)
0	0	1680 (-4.55%)	9480 (-7.24%)	140 (-58.82%)
1	0	1820 (+3.41%)	9620 (-8.38%)	260 (+85.71%)

Table 20: CPU_SUSPEND latencies (ns) to power level 0 in serial (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1560 (+8.33%)	9680 (-2.22%)	300
0	1	1900 (+9.20%)	10380	280
1	0	1760 (-2.22%)	10220 (+1.79%)	340
1	1	1760 (-10.20%)	10500	140 (-68.18%)

CPU_OFF on all non-lead CPUs

CPU_OFF on all non-lead CPUs in sequence then, CPU_SUSPEND on the lead core to the deepest power level.

Table 21: CPU_OFF latencies (ns) on all non-lead CPUs (current release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1660 (-2.35%)	9320 (-6.99%)	300
1	0	13660 (-5.40%)	15980 (-6.77%)	180 (-50.00%)
0	0	13940	16720	140 (-46.15%)
1	0	14220	17080 (-2.95%)	140 (-53.33%)

Table 22: CPU_OFF latencies (ns) on all non-lead CPUs (previous release)

Cluster	Core	Powerdown	Wakeup	Cache Flush
0	0	1700 (+3.66%)	10020 (-2.15%)	300
0	1	14440 (+3.74%)	17140 (-3.16%)	360 (+20.00%)
1	0	13680 (+1.03%)	16680	260 (-13.33%)
1	1	14260	17600 (-3.72%)	300

CPU_VERSION in parallel

Table 23: CPU_VERSION latency (ns) in parallel on all cores (current release)

Cluster	Core	Latency
0	0	120 (-14.29%)
1	0	240
0	0	340 (+54.55%)
1	0	280 (+7.69%)

Table 24: CPU_VERSION latency (ns) in parallel on all cores (previous release)

Cluster	Core	Latency
0	0	140 (+16.67%)
0	1	240 (+20.00%)
1	0	220 (+10.00%)
1	1	260 (+8.33%)

8.4 Runtime Instrumentation Methodology

This document outlines steps for undertaking performance measurements of key operations in the Trusted Firmware-A Power State Coordination Interface (PSCI) implementation, using the in-built Performance Measurement Framework (PMF) and runtime instrumentation timestamps.

8.4.1 Framework

The tests are based on the `runtime-instrumentation` test suite provided by the Trusted Firmware Test Framework (TFTF). The release build of this framework was used because the results in the debug build became skewed; the console output prevented some of the tests from executing in parallel.

The tests consist of both parallel and sequential tests, which are broadly described as follows:

- **Parallel Tests** This type of test powers on all the non-lead CPUs and brings them and the lead CPU to a common synchronization point. The lead CPU then initiates the test on all CPUs in parallel.
- **Sequential Tests** This type of test powers on each non-lead CPU in sequence. The lead CPU initiates the test on a non-lead CPU then waits for the test to complete before proceeding to the next non-lead CPU. The lead CPU then executes the test on itself.

Note there is very little variance observed in the values given (~1us), although the values for each CPU are sometimes interchanged, depending on the order in which locks are acquired. Also, there is very little variance observed between executing the tests sequentially in a single boot or rebooting between tests.

Given that runtime instrumentation using PMF is invasive, there is a small (unquantified) overhead on the results. PMF uses the generic counter for timestamps, which runs at 50MHz on Juno.

8.4.2 Metrics

Powerdown Latency
: Time taken from entering the TF PSCI implementation to the point the hardware enters the low power state (WFI). Referring to the TF runtime instrumentation points, this corresponds to: `(RT_INSTR_ENTER_HW_LOW_PWR - RT_INSTR_ENTER_PSCI)`.

Wakeup Latency
: Time taken from the point the hardware exits the low power state to exiting the TF PSCI implementation. This corresponds to: `(RT_INSTR_EXIT_PSCI - RT_INSTR_EXIT_HW_LOW_PWR)`.

Cache Flush Latency
: Time taken to flush the caches during powerdown. This corresponds to: `(RT_INSTR_EXIT_CFLUSH - RT_INSTR_ENTER_CFLUSH)`.

8.5 Test Secure Payload (TSP) and Dispatcher (TSPD)

8.5.1 Building the Test Secure Payload

The TSP is coupled with a companion runtime service in the BL31 firmware, called the TSPD. Therefore, if you intend to use the TSP, the BL31 image must be recompiled as well. For more information on SPs and SPDs, see the *Secure-EL1 Payloads and Dispatchers* section in the *Firmware Design*.

First clean the TF-A build directory to get rid of any previous BL31 binary. Then to build the TSP image use:

```
make PLAT=<platform> SPD=tspd all
```

An additional boot loader binary file is created in the `build` directory:

```
build/<platform>/<build-type>/bl32.bin
```

8.6 Performance Monitoring Unit

The Performance Monitoring Unit (PMU) allows recording of architectural and microarchitectural events for profiling purposes.

This document gives an overview of the PMU counter configuration to assist with implementation and to complement the PMU security guidelines given in the *Secure Development Guidelines* document.

Note: This section applies to Armv8-A implementations which have version 3 of the Performance Monitors Extension (PMUv3).

8.6.1 PMU Counters

The PMU makes 32 counters available at all privilege levels:

- 31 programmable event counters: `PMEVCNTR<n>`, where `n` is `0` to `30`.
- A dedicated cycle counter: `PMCCNTR`.

Architectural mappings

Counters	State	System Register Name
Programmable	AArch64	`PMEVCNTR<n>_EL0[63*:0]`
	AArch32	`PMEVCNTR<n>[31:0]`
Cycle	AArch64	`PMCCNTR_EL0[63:0]`
	AArch32	`PMCCNTR[63:0]`

Note: Bits [63:32] are only available if ARMv8.5-PMU is implemented. Refer to the Arm ARM for a detailed description of ARMv8.5-PMU features.

8.6.2 Configuring the PMU for counting events

Each programmable counter has an associated register, `PMEVTYPER<n>` which configures it. The cycle counter has the `PMCCFILTR_EL0` register, which has an identical function and bit field layout as `PMEVTYPER<n>`. In addition, the counters are enabled (permitted to increment) via the `PMCNTENSET` and `PMCR` registers. These can be accessed at all privilege levels.

Architectural mappings

AArch64	AArch32
`PMEVTYPER<n>_EL0[63*:0]`	`PMEVTYPER<n>[31:0]`
`PMCCFILTR_EL0[63*:0]`	`PMCCFILTR[31:0]`
`PMCNTENSET_EL0[63*:0]`	`PMCNTENSET[31:0]`
`PMCR_EL0[63*:0]`	`PMCR[31:0]`

Note: Bits [63:32] are reserved.

Relevant register fields

For PMEVTYPER<n>_EL0/PMEVTYPER<n> and PMCCFILTR_EL0/PMCCFILTR, the most important fields are:

- P:
  - Bit 31.
  - If set to 0, will increment the associated PMEVCNTR<n> at EL1.
- NSK:
  - Bit 29.
  - If equal to the P bit it enables the associated PMEVCNTR<n> at Non-secure EL1.
  - Reserved if EL3 not implemented.
- NSH:
  - Bit 27.
  - If set to 1, will increment the associated PMEVCNTR<n> at EL2.
  - Reserved if EL2 not implemented.
- SH:
  - Bit 24.
  - If different to the NSH bit it enables the associated PMEVCNTR<n> at Secure EL2.
  - Reserved if Secure EL2 not implemented.
- M:
  - Bit 26.
  - If equal to the P bit it enables the associated PMEVCNTR<n> at EL3.
- evtCount[15:10]:
  - Extension to evtCount[9:0]. Reserved unless ARMv8.1-PMU implemented.
- evtCount[9:0]:
  - The event number that the associated PMEVCNTR<n> will count.

For PMCNTENSET_EL0/PMCNTENSET, the most important fields are:

- P[30:0]:
  - Setting bit P[n] to 1 enables counter PMEVCNTR<n>.
  - The effects of PMEVTYPER<n> are applied on top of this. In other words, the counter will not increment at any privilege level or security state unless it is enabled here.
- C:
  - Bit 31.

  - If set to 1 enables the cycle counter PMCCNTR.

For PMCR/PMCR_EL0, the most important fields are:

- DP:
  - Bit 5.
  - If set to 1 it disables the cycle counter PMCCNTR where event counting (by PMEVCNTR<n>) is prohibited (e.g. EL2 and the Secure world).
  - If set to 0, PMCCNTR will not be affected by this bit and therefore will be able to count where the programmable counters are prohibited.
- E:
  - Bit 0.
  - Enables/disables counting altogether.
  - The effects of PMCNTENSET and PMCR.DP are applied on top of this. In other words, if this bit is 0 then no counters will increment regardless of how the other PMU system registers or bit fields are configured.

References

- Arm ARM

CHAPTER

NINE

SECURITY ADVISORIES

9.1 Advisory TFV-1 (CVE-2016-10319)

Title	Malformed Firmware Update SMC can result in copy of unexpectedly large data into secure memory
CVE ID	CVE-2016-10319
Date	18 Oct 2016
Versions Affected	v1.2 and v1.3 (since commit 48bfb88)
Configurations Affected	Platforms that use AArch64 BL1 plus untrusted normal world firmware update code executing before BL31
Impact	Copy of unexpectedly large data into the free secure memory reported by BL1 platform code
Fix Version	Pull Request #783
Credit	IOActive

Generic Trusted Firmware (TF) BL1 code contains an SMC interface that is briefly available after cold reset to support the Firmware Update (FWU) feature (also known as recovery mode). This allows most FWU functionality to be implemented in the normal world, while retaining the essential image authentication functionality in BL1. When cold boot reaches the EL3 Runtime Software (for example, BL31 on AArch64 systems), the FWU SMC interface is replaced by the EL3 Runtime SMC interface. Platforms may choose how much of this FWU functionality to use, if any.

The BL1 FWU SMC handling code, currently only supported on AArch64, contains several vulnerabilities that may be exploited when *all* the following conditions apply:

1. Platform code uses TF BL1 with the `TRUSTED_BOARD_BOOT` build option enabled.
2. Platform code arranges for untrusted normal world FWU code to be executed in the cold boot path, before BL31 starts. Untrusted in this sense means code that is not in ROM or has not been authenticated or has otherwise been executed by an attacker.
3. Platform code copies the insecure pattern described below from the ARM platform version of `bl1_plat_mem_check()`.

The vulnerabilities consist of potential integer overflows in the input validation checks while handling the `FWU_SMC_IMAGE_COPY` SMC. The SMC implementation is designed to copy an image into secure memory for subsequent authentication, but the vulnerabilities may allow an attacker to copy unexpectedly large data into secure memory. Note that a separate vulnerability is required to leverage these vulnerabilities; for example a way to get the system to change its behaviour based on the unexpected secure memory contents.

Two of the vulnerabilities are in the function `bl1_fwu_image_copy()` in `bl1/bl1_fwu.c`. These are listed below, referring to the v1.3 tagged version of the code:

- Line 155:

```
/*
 * If last block is more than expected then
 * clip the block to the required image size.
 */
if (image_desc->copied_size + block_size >
     image_desc->image_info.image_size) {
    block_size = image_desc->image_info.image_size -
        image_desc->copied_size;
    WARN("BL1-FWU: Copy argument block_size > remaining image size."
        " Clipping block_size\n");
}

/* Make sure the image src/size is mapped. */
if (bl1_plat_mem_check(image_src, block_size, flags)) {
    WARN("BL1-FWU: Copy arguments source/size not mapped\n");
    return -ENOMEM;
}

INFO("BL1-FWU: Continuing image copy in blocks\n");

/* Copy image for given block size. */
base_addr += image_desc->copied_size;
image_desc->copied_size += block_size;
memcpy((void *)base_addr, (const void *)image_src, block_size);
...
```

  This code fragment is executed when the image copy operation is performed in blocks over multiple SMCs. `block_size` is an SMC argument and therefore potentially controllable by an attacker. A very large value may result in an integer overflow in the 1st `if` statement, which would bypass the check, allowing an unclipped `block_size` to be passed into `bl1_plat_mem_check()`. If `bl1_plat_mem_check()` also passes, this may result in an unexpectedly large copy of data into secure memory.

- Line 206:

```
/* Make sure the image src/size is mapped. */
if (bl1_plat_mem_check(image_src, block_size, flags)) {
    WARN("BL1-FWU: Copy arguments source/size not mapped\n");
    return -ENOMEM;
}

/* Find out how much free trusted ram remains after BL1 load */
mem_layout = bl1_plat_sec_mem_layout();
if ((image_desc->image_info.image_base < mem_layout->free_base) ||
     (image_desc->image_info.image_base + image_size >
```

(continues on next page)

(continued from previous page)

```
        mem_layout->free_base + mem_layout->free_size)) {
    WARN("BL1-FWU: Memory not available to copy\n");
    return -ENOMEM;
}

/* Update the image size. */
image_desc->image_info.image_size = image_size;

/* Copy image for given size. */
memcpy((void *)base_addr, (const void *)image_src, block_size);
...
```

This code fragment is executed during the 1st invocation of the image copy operation. Both `block_size` and `image_size` are SMC arguments. A very large value of `image_size` may result in an integer overflow in the 2nd `if` statement, which would bypass the check, allowing execution to proceed. If `bl1_plat_mem_check()` also passes, this may result in an unexpectedly large copy of data into secure memory.

If the platform's implementation of `bl1_plat_mem_check()` is correct then it may help prevent the above 2 vulnerabilities from being exploited. However, the ARM platform version of this function contains a similar vulnerability:

- Line 88 of `plat/arm/common/arm_bl1_fwu.c` in function of `bl1_plat_mem_check()`:

```
while (mmap[index].mem_size) {
    if ((mem_base >= mmap[index].mem_base) &&
        ((mem_base + mem_size)
        <= (mmap[index].mem_base +
        mmap[index].mem_size)))
        return 0;

    index++;
}
...
```

This function checks that the passed memory region is within one of the regions mapped in by ARM platforms. Here, `mem_size` may be the `block_size` passed from `bl1_fwu_image_copy()`. A very large value of `mem_size` may result in an integer overflow and the function to incorrectly return success. Platforms that copy this insecure pattern will have the same vulnerability.

9.2 Advisory TFV-2 (CVE-2017-7564)

Title	Enabled secure self-hosted invasive debug interface can allow normal world to panic secure world
CVE ID	CVE-2017-7564
Date	02 Feb 2017
Versions Affected	All versions up to v1.3
Configurations Affected	All
Impact	Denial of Service (secure world panic)
Fix Version	15 Feb 2017 Pull Request #841
Credit	ARM

The `MDCR_EL3.SDD` bit controls AArch64 secure self-hosted invasive debug enablement. By default, the BL1 and BL31 images of the current version of ARM Trusted Firmware (TF) unconditionally assign this bit to `0` in the early entrypoint code, which enables debug exceptions from the secure world. This can be seen in the implementation of the `el3_arch_init_common` AArch64 macro . Given that TF does not currently contain support for this feature (for example, by saving and restoring the appropriate debug registers), this may allow a normal world attacker to induce a panic in the secure world.

The `MDCR_EL3.SDD` bit should be assigned to `1` to disable debug exceptions from the secure world.

Earlier versions of TF (prior to commit 495f3d3) did not assign this bit. Since the bit has an architecturally `UNKNOWN` reset value, earlier versions may or may not have the same problem, depending on the platform.

A similar issue applies to the `MDCR_EL3.SPD32` bits, which control AArch32 secure self-hosted invasive debug enablement. TF assigns these bits to `00` meaning that debug exceptions from Secure EL1 are enabled by the authentication interface. Therefore this issue only exists for AArch32 Secure EL1 code when secure privileged invasive debug is enabled by the authentication interface, at which point the device is vulnerable to other, more serious attacks anyway.

However, given that TF contains no support for handling debug exceptions, the `MDCR_EL3.SPD32` bits should be assigned to `10` to disable debug exceptions from AArch32 Secure EL1.

Finally, this also issue applies to AArch32 platforms that use the TF SP_MIN image or integrate the AArch32 equivalent of the `el3_arch_init_common` macro. Here the affected bits are `SDCR.SPD`, which should also be assigned to `10` instead of `00`

9.3 Advisory TFV-3 (CVE-2017-7563)

Title	RO memory is always executable at AArch64 Secure EL1
CVE ID	CVE-2017-7563
Date	06 Apr 2017
Versions Affected	v1.3 (since Pull Request #662)
Configurations Affected	AArch64 BL2, TSP or other users of xlat_tables library executing at AArch64 Secure EL1
Impact	Unexpected Privilege Escalation
Fix Version	Pull Request #924
Credit	ARM

The translation table library in ARM Trusted Firmware (TF) (under `lib/xlat_tables` and `lib/xlat_tables_v2`) provides APIs to help program translation tables in the MMU. The xlat_tables client specifies its required memory mappings in the form of `mmap_region` structures. Each `mmap_region` has memory attributes represented by the `mmap_attr_t` enumeration type. This contains flags to control data access permissions (`MT_RO`/`MT_RW`) and instruction execution permissions (`MT_EXECUTE`/`MT_EXECUTE_NEVER`). Thus a mapping specifying both `MT_RO` and `MT_EXECUTE_NEVER` should result in a Read-Only (RO), non-executable memory region.

This feature does not work correctly for AArch64 images executing at Secure EL1. Any memory region mapped as RO will always be executable, regardless of whether the client specified `MT_EXECUTE` or `MT_EXECUTE_NEVER`.

The vulnerability is known to affect the BL2 and Test Secure Payload (TSP) images on platforms that enable the `SEPARATE_CODE_AND_RODATA` build option, which includes all ARM standard platforms, and the upstream Xilinx and NVidia platforms. The RO data section for these images on these platforms is unexpectedly executable instead of non-executable. Other platforms or `xlat_tables` clients may also be affected.

The vulnerability primarily manifests itself after Pull Request #662. Before that, `xlat_tables` clients could not specify instruction execution permissions separately to data access permissions. All RO normal memory regions were implicitly executable. Before Pull Request #662. the vulnerability would only manifest itself for device memory mapped as RO; use of this mapping is considered rare, although the upstream QEMU platform uses this mapping when the `DEVICE2_BASE` build option is used.

Note that one or more separate vulnerabilities are also required to exploit this vulnerability.

The vulnerability is due to incorrect handling of the execute-never bits in the translation tables. The EL3 translation regime uses a single `XN` bit to determine whether a region is executable. The Secure EL1&0 translation regime handles 2 Virtual Address (VA) ranges and so uses 2 bits, `UXN` and `PXN`. The `xlat_tables` library only handles the `XN` bit, which maps to `UXN` in the Secure EL1&0 regime. As a result, this programs the Secure EL0 execution permissions but always leaves the memory as executable at Secure EL1.

The vulnerability is mitigated by the following factors:

- The xlat_tables library ensures that all Read-Write (RW) memory regions are non-executable by setting the `SCTLR_ELx.WXN` bit. This overrides any value of the `XN`, `UXN` or `PXN` bits in the translation tables. See the `enable_mmu()` function:

```
sctlr = read_sctlr_el##_el();              \
sctlr |= SCTLR_WXN_BIT | SCTLR_M_BIT;      \
```

- AArch32 configurations are unaffected. Here the XN bit controls execution privileges of the currently executing translation regime, which is the desired behaviour.
- ARM TF EL3 code (for example BL1 and BL31) ensures that all non-secure memory mapped into the secure world is non-executable by setting the SCR_EL3.SIF bit. See the el3_arch_init_common macro in el3_common_macros.S.

9.4 Advisory TFV-4 (CVE-2017-9607)

Title	Malformed Firmware Update SMC can result in copy or authentication of unexpected data in secure memory in AArch32 state
CVE ID	CVE-2017-9607
Date	20 Jun 2017
Versions Affected	None (only between 22 May 2017 and 14 June 2017)
Configurations Affected	Platforms that use AArch32 BL1 plus untrusted normal world firmware update code executing before BL31
Impact	Copy or authentication of unexpected data in the secure memory
Fix Version	Pull Request #979 (merged on 14 June 2017)
Credit	ARM

The include/lib/utils_def.h header file provides the check_uptr_overflow() macro, which aims at detecting arithmetic overflows that may occur when computing the sum of a base pointer and an offset. This macro evaluates to 1 if the sum of the given base pointer and offset would result in a value large enough to wrap around, which may lead to unpredictable behaviour.

The macro code is at line 52, referring to the version of the code as of commit c396b73:

```
/*
 * Evaluates to 1 if (ptr + inc) overflows, 0 otherwise.
 * Both arguments must be unsigned pointer values (i.e. uintptr_t).
 */
#define check_uptr_overflow(ptr, inc)        \
    (((ptr) > UINTPTR_MAX - (inc)) ? 1 : 0)
```

This macro does not work correctly for AArch32 images. It fails to detect overflows when the sum of its two parameters fall into the [2^32, 2^64 - 1] range. Therefore, any AArch32 code relying on this macro to detect such integer overflows is actually not protected.

The buggy code has been present in ARM Trusted Firmware (TF) since Pull Request #678 was merged (on 18 August 2016). However, the upstream code was not vulnerable until Pull Request #939 was merged (on 22 May 2017), which introduced AArch32 support for the Trusted Board Boot (TBB) feature. Before then, the check_uptr_overflow() macro was not used in AArch32 code.

The vulnerability resides in the BL1 FWU SMC handling code and it may be exploited when *all* the following conditions apply:

- Platform code uses TF BL1 with the `TRUSTED_BOARD_BOOT` build option.
- Platform code uses the Firmware Update (FWU) code provided in `bl1/bl1_fwu.c`, which is part of the TBB support.
- TF BL1 is compiled with the `ARCH=aarch32` build option.

In this context, the AArch32 BL1 image might fail to detect potential integer overflows in the input validation checks while handling the `FWU_SMC_IMAGE_COPY` and `FWU_SMC_IMAGE_AUTH` SMCs.

The `FWU_SMC_IMAGE_COPY` SMC handler is designed to copy an image into secure memory for subsequent authentication. This is implemented by the `bl1_fwu_image_copy()` function, which has the following function prototype:

```
static int bl1_fwu_image_copy(unsigned int image_id,
                    uintptr_t image_src,
                    unsigned int block_size,
                    unsigned int image_size,
                    unsigned int flags)
```

`image_src` is an SMC argument and therefore potentially controllable by an attacker. A very large 32-bit value, for example `2^32 -1`, may result in the sum of `image_src` and `block_size` overflowing a 32-bit type, which `check_uptr_overflow()` will fail to detect. Depending on its implementation, the platform-specific function `bl1_plat_mem_check()` might get defeated by these unsanitized values and allow the following memory copy operation, that would wrap around. This may allow an attacker to copy unexpected data into secure memory if the memory is mapped in BL1's address space, or cause a fatal exception if it's not.

The `FWU_SMC_IMAGE_AUTH` SMC handler is designed to authenticate an image resident in secure memory. This is implemented by the `bl1_fwu_image_auth()` function, which has the following function prototype:

```
static int bl1_fwu_image_auth(unsigned int image_id,
                    uintptr_t image_src,
                    unsigned int image_size,
                    unsigned int flags)
```

Similarly, if an attacker has control over the `image_src` or `image_size` arguments through the SMC interface and injects high values whose sum overflows, they might defeat the `bl1_plat_mem_check()` function and make the authentication module read data outside of what's normally allowed by the platform code or crash the platform.

Note that in both cases, a separate vulnerability is required to leverage this vulnerability; for example a way to get the system to change its behaviour based on the unexpected secure memory accesses. Moreover, the normal world FWU code would need to be compromised in order to send a malformed FWU SMC that triggers an integer overflow.

The vulnerability is known to affect all ARM standard platforms when enabling the `TRUSTED_BOARD_BOOT` and `ARCH=aarch32` build options. Other platforms may also be affected if they fulfil the above conditions.

9.5 Advisory TFV-5 (CVE-2017-15031)

Title	Not initializing or saving/restoring PMCR_EL0 can leak secure world timing information
CVE ID	CVE-2017-15031
Date	02 Oct 2017, updated on 04 Nov 2019
Versions Affected	All, up to and including v2.1
Configurations Affected	All
Impact	Leakage of sensitive secure world timing information
Fix Version	Pull Request #1127 (merged on 18 October 2017) Commit e290a8fcbc (merged on 23 August 2019) Commit c3e8b0be9b (merged on 27 September 2019)
Credit	Arm, Marek Bykowski

The `PMCR_EL0` (Performance Monitors Control Register) provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters. If the `PMCR_EL0.DP` bit is set to zero, the cycle counter (when enabled) counts during secure world execution, even when prohibited by the debug signals.

Since TF-A does not save and restore `PMCR_EL0` when switching between the normal and secure worlds, normal world code can set `PMCR_EL0.DP` to zero to cause leakage of secure world timing information. This register should be added to the list of saved/restored registers both when entering EL3 and also transitioning to S-EL1.

Furthermore, `PMCR_EL0.DP` has an architecturally `UNKNOWN` reset value. Since Arm TF does not initialize this register, it's possible that on at least some implementations, `PMCR_EL0.DP` is set to zero by default. This and other bits with an architecturally UNKNOWN reset value should be initialized to sensible default values in the secure context.

The same issue exists for the equivalent AArch32 register, `PMCR`, except that here `PMCR_EL0.DP` architecturally resets to zero.

NOTE: The original pull request referenced above only fixed the issue for S-EL1 whereas the EL3 was fixed in the later commits.

9.6 Advisory TFV-6 (CVE-2017-5753, CVE-2017-5715, CVE-2017-5754)

Title	Trusted Firmware-A exposure to speculative processor vulnerabilities using cache timing side-channels
CVE ID	CVE-2017-5753 / CVE-2017-5715 / CVE-2017-5754 / CVE-2018-3640 / CVE-2018-3639 / CVE-2022-23960
Date	03 Jan 2018 (Updated 11 Jan, 18 Jan, 26 Jan, 30 Jan, 07 June 2018 and 07 Aug 2025)
Versions Affected	All, up to and including v1.4
Configurations Affected	All
Impact	Leakage of secure world data to normal world
Fix Version	Pull Request #1214, Pull Request #1228, Pull Request #1240 and Pull Request #1405
Credit	Google / Arm

This security advisory describes the current understanding of the Trusted Firmware-A exposure to the speculative processor vulnerabilities identified by Google Project Zero. To understand the background and wider impact of these vulnerabilities on Arm systems, please refer to the Arm Processor Security Update.

9.6.1 Variant 1 (CVE-2017-5753)

At the time of writing, no vulnerable patterns have been observed in upstream TF code, therefore no workarounds have been applied or are planned.

9.6.2 Variant 2 (CVE-2017-5715)

Where possible on vulnerable CPUs, Arm recommends invalidating the branch predictor as early as possible on entry into the secure world, before any branch instruction is executed. There are a number of implementation defined ways to achieve this.

For Cortex-A57 and Cortex-A72 CPUs, the Pull Requests (PRs) in this advisory invalidate the branch predictor when entering EL3 by disabling and re-enabling the MMU.

For Cortex-A73 and Cortex-A75 CPUs, the PRs in this advisory invalidate the branch predictor when entering EL3 by temporarily dropping into AArch32 Secure-EL1 and executing the `BPIALL` instruction. This workaround is significantly more complex than the "MMU disable/enable" workaround. The latter is not effective at invalidating the branch predictor on Cortex-A73/Cortex-A75.

Note that if other privileged software, for example a Rich OS kernel, implements its own branch predictor invalidation during context switch by issuing an SMC (to execute firmware branch predictor invalidation), then there is a dependency on the PRs in this advisory being deployed in order for those workarounds to work. If that other privileged software is able to workaround the vulnerability locally (for example by implementing "MMU disable/enable" itself), there is no such dependency.

Pull Request #1240 and Pull Request #1405 optimise the earlier fixes by implementing a specified CVE-2017-5715 workaround SMC (`SMCCC_ARCH_WORKAROUND_1`) for use by normal world privileged software. This is more efficient than calling an arbitrary SMC (for example `PSCI_VERSION`). Details of SM-

CCC_ARCH_WORKAROUND_1 can be found in the CVE-2017-5715 mitigation specification. The specification and implementation also enable the normal world to discover the presence of this firmware service.

On Juno R1 we measured the round trip latency for both the PSCI_VERSION and SMCCC_ARCH_WORKAROUND_1 SMCs on Cortex-A57, using both the "MMU disable/enable" and "BPIALL at AArch32 Secure-EL1" workarounds described above. This includes the time spent in test code conforming to the SMC Calling Convention (SMCCC) from AArch64. For the SMCCC_ARCH_WORKAROUND_1 cases, the test code uses SMCCC v1.1, which reduces the number of general purpose registers it needs to save/restore. Although the BPIALL instruction is not effective at invalidating the branch predictor on Cortex-A57, the drop into Secure-EL1 with MMU disabled that this workaround entails effectively does invalidate the branch predictor. Hence this is a reasonable comparison.

The results were as follows:

Test	Time (ns)
PSCI_VERSION baseline (without PRs in this advisory)	515
PSCI_VERSION baseline (with PRs in this advisory)	527
PSCI_VERSION with "MMU disable/enable"	930
SMCCC_ARCH_WORKAROUND_1 with "MMU disable/enable"	386
PSCI_VERSION with "BPIALL at AArch32 Secure-EL1"	1276
SMCCC_ARCH_WORKAROUND_1 with "BPIALL at AArch32 Secure-EL1"	770

Due to the high severity and wide applicability of this issue, the above workarounds are enabled by default (on vulnerable CPUs only), despite some performance and code size overhead. Platforms can choose to disable them at compile time if they do not require them. Pull Request #1240 disables the workarounds for unaffected upstream platforms.

For vulnerable AArch32-only CPUs (for example Cortex-A8, Cortex-A9 and Cortex-A17), the BPIALL instruction should be used as early as possible on entry into the secure world. For Cortex-A8, also set ACTLR[6] to 1 during early processor initialization. Note that the BPIALL instruction is not effective at invalidating the branch predictor on Cortex-A15. For that CPU, set ACTLR[0] to 1 during early processor initialization, and invalidate the branch predictor by performing an ICIALLU instruction.

On AArch32 EL3 systems, the monitor and secure-SVC code is typically tightly integrated, for example as part of a Trusted OS. Therefore any Variant 2 workaround should be provided by vendors of that software and is outside the scope of TF. However, an example implementation in the minimal AArch32 Secure Payload, SP_MIN is provided in Pull Request #1228.

Refer to Arm CPU Security Bulletin-Spectre/Meltdown for list of impacted CPUs.

For more information about non-Arm CPUs, please contact the CPU vendor.

9.6.3 Variant 3 (CVE-2017-5754)

This variant is only exploitable between Exception Levels within the same translation regime, for example between EL0 and EL1, therefore this variant cannot be used to access secure memory from the non-secure world, and is not applicable for TF. However, Secure Payloads (for example, Trusted OS) should provide mitigations on vulnerable CPUs to protect themselves from exploited Secure-EL0 applications.

Refer to Arm CPU Security Bulletin-Spectre/Meltdown for list of impacted CPUs.

9.6.4 Variant 3a (CVE-2018-3640)

In general, it is not believed that software mitigations for this issue are necessary.

9.6.5 Variant 4 (CVE-2018-3639)

Refer to *Advisory TFV-7 (CVE-2018-3639)* for further details on this variant.

9.6.6 Spectre-BHB (CVE-2022-23960)

Refer to *Advisory TFV-9 (CVE-2022-23960)* for further details on this variant.

9.7 Advisory TFV-7 (CVE-2018-3639)

Title	Trusted Firmware-A exposure to cache speculation vulnerability Variant 4
CVE ID	CVE-2018-3639
Date	21 May 2018 (Updated 7 June 2018)
Versions Affected	All, up to and including v1.5
Configurations Affected	All
Impact	Leakage of secure world data to normal world
Fix Version	Pull Request #1392, Pull Request #1397
Credit	Google

This security advisory describes the current understanding of the Trusted Firmware-A (TF-A) exposure to Variant 4 of the cache speculation vulnerabilities identified by Google Project Zero. To understand the background and wider impact of these vulnerabilities on Arm systems, please refer to the Arm Processor Security Update.

At the time of writing, the TF-A project is not aware of a Variant 4 exploit that could be used against TF-A. It is likely to be very difficult to achieve an exploit against current standard configurations of TF-A, due to the limited interfaces into the secure world with attacker-controlled inputs. However, this is becoming increasingly difficult to guarantee with the introduction of complex new firmware interfaces, for example the Software Delegated Exception Interface (SDEI). Also, the TF-A project does not have visibility of all vendor-supplied interfaces. Therefore, the TF-A project takes a conservative approach by mitigating Variant 4 in hardware wherever possible during secure world execution. The mitigation is enabled by setting an implementation defined control bit to prevent the re-ordering of stores and loads.

For each affected CPU type, TF-A implements one of the two following mitigation approaches in Pull Request #1392 and Pull Request #1397. Both approaches have a system performance impact, which varies for each CPU type and use-case. The mitigation code is enabled by default, but can be disabled at compile time for platforms that are unaffected or where the risk is deemed low enough.

Arm CPUs not mentioned below are unaffected.

9.7.1 Static mitigation

For affected CPUs, this approach enables the mitigation during EL3 initialization, following every PE reset. No mechanism is provided to disable the mitigation at runtime.

This approach permanently mitigates the entire software stack and no additional mitigation code is required in other software components.

TF-A implements this approach for the following affected CPUs:

- Cortex-A57 and Cortex-A72, by setting bit 55 (Disable load pass store) of `CPUACTLR_EL1` (`S3_1_C15_C2_0`).
- Cortex-A73, by setting bit 3 of `S3_0_C15_C0_0` (not documented in the Technical Reference Manual (TRM)).
- Cortex-A75, by setting bit 35 (reserved in TRM) of `CPUACTLR_EL1` (`S3_0_C15_C1_0`).

9.7.2 Dynamic mitigation

For affected CPUs, this approach also enables the mitigation during EL3 initialization, following every PE reset. In addition, this approach implements `SMCCC_ARCH_WORKAROUND_2` in the Arm architectural range to allow callers at lower exception levels to temporarily disable the mitigation in their execution context, where the risk is deemed low enough. This approach enables mitigation on entry to EL3, and restores the mitigation state of the lower exception level on exit from EL3. For more information on this approach, see Firmware interfaces for mitigating cache speculation vulnerabilities.

This approach may be complemented by additional mitigation code in other software components, for example code that calls `SMCCC_ARCH_WORKAROUND_2`. However, even without any mitigation code in other software components, this approach will effectively permanently mitigate the entire software stack, since the default mitigation state for firmware-managed execution contexts is enabled.

Since the expectation in this approach is that more software executes with the mitigation disabled, this may result in better system performance than the static approach for some systems or use-cases. However, for other systems or use-cases, this performance saving may be outweighed by the additional overhead of `SMCCC_ARCH_WORKAROUND_2` calls and TF-A exception handling.

TF-A implements this approach for the following affected CPU:

- Cortex-A76, by setting and clearing bit 16 (reserved in TRM) of `CPUACTLR2_EL1` (`S3_0_C15_C1_1`).

9.8 Advisory TFV-8 (CVE-2018-19440)

Title	Not saving x0 to x3 registers can leak information from one Normal World SMC client to another
CVE ID	CVE-2018-19440
Date	27 Nov 2018
Versions Affected	All, up to and including v2.0
Configurations Affected	Multiple normal world SMC clients calling into AArch64 BL31
Impact	Leakage of SMC return values from one normal world SMC client to another
Fix Version	Pull Request #1710 (merged on 6 December 2018) Commit 01fc1c24b9
Credit	Secmation

When taking an exception to EL3, BL31 saves the CPU context. The aim is to restore it before returning into the lower exception level software that called into the firmware. However, for an SMC exception, the general purpose registers `x0` to `x3` are not part of the CPU context saved on the stack.

As per the SMC Calling Convention, up to 4 values may be returned to the caller in registers `x0` to `x3`. In TF-A, these return values are written into the CPU context, typically using one of the `SMC_RETx()` macros provided in the `include/lib/aarch64/smccc_helpers.h` header file.

Before returning to the caller, the `restore_gp_registers()` function is called. It restores the values of all general purpose registers taken from the CPU context stored on the stack. This includes registers `x0` to `x3`, as can be seen in the `lib/el3_runtime/aarch64/context.S` file at line 339 (referring to the version of the code as of commit c385955):

```
/*
 * This function restores all general purpose registers except x30 from the
 * CPU context. x30 register must be explicitly restored by the caller.
 */
func restore_gp_registers
    ldp x0, x1, [sp, #CTX_GPREGS_OFFSET + CTX_GPREG_X0]
    ldp x2, x3, [sp, #CTX_GPREGS_OFFSET + CTX_GPREG_X2]
```

In the case of an SMC handler that does not use all 4 return values, the remaining ones are left unchanged in the CPU context. As a result, `restore_gp_registers()` restores the stale values saved by a previous SMC request (or asynchronous exception to EL3) that used these return values.

In the presence of multiple normal world SMC clients, this behaviour might leak some of the return values from one client to another. For example, if a victim client first sends an SMC that returns 4 values, a malicious client may then send a second SMC expecting no return values (for example, a `SDEI_EVENT_COMPLETE` SMC) to get the 4 return values of the victim client.

In general, the responsibility for mitigating threats due to the presence of multiple normal world SMC clients lies with EL2 software. When present, EL2 software must trap SMC calls from EL1 software to ensure secure behaviour.

For this reason, TF-A does not save `x0` to `x3` in the CPU context on an SMC synchronous exception. It has behaved this way since the first version.

We can confirm that at least upstream KVM-based systems mitigate this threat, and are therefore unaffected by this issue. Other EL2 software should be audited to assess the impact of this threat.

EL2 software might find mitigating this threat somewhat onerous, because for all SMCs it would need to be aware of which return registers contain valid data, so it can sanitise any unused return registers. On the other hand, mitigating this in EL3 is relatively easy and cheap. Therefore, TF-A will now ensure that no information is leaked through registers `x0` to `x3`, by preserving the register state over the call.

Note that AArch32 TF-A is not affected by this issue. The SMC handling code in `SP_MIN` already saves all general purpose registers - including `r0` to `r3`, as can be seen in the `include/lib/aarch32/smccc_macros.S` file at line 19 (referring to the version of the code as of commit c385955):

```
/*
 * Macro to save the General purpose registers (r0 - r12), the banked
 * spsr, lr, sp registers and the `scr` register to the SMC context on entry
 * due a SMC call. The `lr` of the current mode (monitor) is expected to be
 * already saved. The `sp` must point to the `smc_ctx_t` to save to.
 * Additionally, also save the 'pmcr' register as this is updated whilst
 * executing in the secure world.
 */
    .macro smccc_save_gp_mode_regs
    /* Save r0 - r12 in the SMC context */
    stm sp, {r0-r12}
```

9.9 Advisory TFV-9 (CVE-2022-23960)

Title	Trusted Firmware-A exposure to speculative processor vulnerabilities with branch prediction target reuse
CVE ID	CVE-2022-23960
Date	08 Mar 2022
Versions Affected	All, up to and including v2.6
Configurations Affected	All
Impact	Potential leakage of secure world data to normal world if an attacker is able to find a TF-A exfiltration primitive that can be predicted as a valid branch target, and somehow induce misprediction onto that primitive. There are currently no known exploits.
Fix Version	Gerrit topic #spectre_bhb
Credit	Systems and Network Security Group at Vrije Universiteit Amsterdam for CVE-2022-23960, Arm for patches

This security advisory describes the current understanding of the Trusted Firmware-A exposure to the new speculative processor vulnerability. To understand the background and wider impact of these vulnerabilities on Arm systems, please refer to the Arm Processor Security Update. The whitepaper referred to below describes the Spectre attack and mitigation in more detail including implementation specific mitigation details for all impacted Arm CPUs.

9.9.1 CVE-2022-23960

Where possible on vulnerable CPUs that implement FEAT_CSV2, Arm recommends inserting a loop workaround with implementation specific number of iterations that will discard the branch history on exception entry to a higher exception level for the given CPU. This is done as early as possible on entry into EL3, before any branch instruction is executed. This is sufficient to mitigate Spectre-BHB on behalf of all secure world code, assuming that no secure world code is under attacker control.

The below table lists the CPUs that mitigate against this vulnerability in TF-A using the loop workaround(all cores that implement FEAT_CSV2 except the revisions of Cortex-A73 and Cortex-A75 that implements FEAT_CSV2).

Core
Cortex-A72(from r1p0)
Cortex-A76
Cortex-A76AE
Cortex-A77
Cortex-A78
Cortex-A78AE
Cortex-A78C
Cortex-X1
Cortex-X2
Cortex-X3
Cortex-A710
Cortex-A715
Neoverse-N1
Neoverse-N2
Neoverse-V1
Neoverse-V2

For all other cores impacted by Spectre-BHB, some of which that do not implement FEAT_CSV2 and some that do e.g. Cortex-A73, the recommended mitigation is to flush all branch predictions via an implementation specific route.

In case local workaround is not feasible, the Rich OS can invoke the SMC (`SMCCC_ARCH_WORKAROUND_3`) to apply the workaround. Refer to SMCCC Calling Convention specification for more details.

Gerrit topic #spectre_bhb This patchset implements the Spectre-BHB loop workaround for CPUs mentioned in the above table. For CPUs supporting speculative barrier instruction, the loop workaround is optimised by using SB in place of the common DSB and ISB sequence. It also mitigates against this vulnerability for Cortex-A72 CPU versions that support the CSV2 feature (from r1p0). The patch stack also includes an implementation for a specified CVE-2022-23960 workaround SMC(`SMCCC_ARCH_WORKAROUND_3`) for use by normal world privileged software. Details of `SMCCC_ARCH_WORKAROUND_3` can be found in the SMCCC Calling Convention specification. The specification and implementation also enables the normal world to discover the presence of this firmware service. This patch also implements `SMCCC_ARCH_WORKAROUND_3` for Cortex-A57, Coxtex-A72, Cortex-A73 and Cortex-A75 using the existing workaround. for CVE-2017-5715. Cortex-A15 patch extends Spectre V2 mitigation to Spectre-BHB.

The above workaround is enabled by default (on vulnerable CPUs only). Platforms can choose to disable them at compile time if they do not require them.

For more information about non-Arm CPUs, please contact the CPU vendor.

9.10 Advisory TFV-10 (CVE-2022-47630)

Title	Incorrect validation of X.509 certificate extensions can result in an out-of-bounds read.
CVE ID	CVE-2022-47630
Date	Reported on 12 Dec 2022
Versions Affected	v1.2 to v2.8
Configurations Affected	BL1 and BL2 with Trusted Boot enabled with custom, downstream usages of `get_ext()` and/or `auth_nvctr()` interfaces. Not exploitable in upstream TF-A code.
Impact	Out-of-bounds read.
Fix Version	• fd37982a19a4a291 "fix(auth): forbid junk after extensions" • 72460f50e2437a85 "fix(auth): require at least one extension to be present" • f5c51855d36e399e "fix(auth): properly validate X.509 extensions" • abb8f936fd0ad085 "fix(auth): avoid out-of-bounds read in auth_nvctr()" Note that 72460f50e2437a85 is not fixing any vulnerability per se but it is required for f5c51855d36e399e to apply cleanly.
Credit	Demi Marie Obenour, Invisible Things Lab

This security advisory describes a vulnerability in the X.509 parser used to parse boot certificates in TF-A trusted boot: it is possible for a crafted certificate to cause an out-of-bounds memory read.

Note that upstream platforms are **not** affected by this. Only downstream platforms may be, if (and only if) the interfaces described below are used in a different context than seen in upstream code. Details of such context is described in the rest of this document.

To fully understand this security advisory, it is recommended to refer to the following standards documents:

- RFC 5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.
- ITU-T X.690, *ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*

9.10.1 Bug 1: Insufficient certificate validation

The vulnerability lies in the following source file: `drivers/auth/mbedtls/mbedtls_x509_parser.c`. By design, `get_ext()` does not check the return value of the various `mbedtls_*()` functions, as `cert_parse()` is assumed to have guaranteed that they will always succeed. However, it passes the end of an extension as the end pointer to these functions, whereas `cert_parse()` passes the end of the `TBSCertificate`. Furthermore, `cert_parse()` does not check that the contents of the extension have the same length as the extension itself. It also does not check that the extension block extends to the end of the `TBSCertificate`.

This is a problem, as `mbedtls_asn1_get_tag()` leaves `*p` and `*len` undefined on failure. In practice, this results in `get_ext()` continuing to parse at different offsets than were used (and validated) by `cert_parse()`, which means that the in-bounds guarantee provided by `cert_parse()` no longer holds. The result is that it is possible for `get_ext()` to read memory past the end of the certificate. This could potentially access memory with dangerous read side effects, or leak microarchitectural state that could theoretically be retrieved through some side-channel attacks as part of a more complex attack.

9.10.2 Bug 2: Missing bounds check in `auth_nvctr()`

`auth_nvctr()` does not check that the buffer provided is long enough to hold an `ASN.1 INTEGER`. Since `auth_nvctr()` will only ever read 6 bytes, it is possible to read up to 6 bytes past the end of the buffer.

9.10.3 Exploitability Analysis

Upstream TF-A Code

In upstream TF-A code, the only caller of `auth_nvctr()` takes its input from `get_ext()`, which means that the second bug is exploitable, so is the first. Therefore, only the first bug need be considered.

All standard chains of trust provided in TF-A source tree (that is, under `drivers/auth/`) require that the certificate's signature has already been validated prior to calling `get_ext()`, or any function that calls `get_ext()`. Platforms taking their chain of trust from a dynamic configuration file (such as `fdts/tbbr_cot_descriptors.dts`) are also safe, as signature verification will always be done prior to any calls to `get_ext()` or `auth_nvctr()` in this case, no matter the order of the properties in the file. Therefore, it is not possible to exploit this vulnerability pre-authentication in upstream TF-A.

Furthermore, the data read through `get_ext()` only ever gets used by the authentication framework (`drivers/auth/auth_mod.c`), which greatly reduces the range of inputs it will ever receive and thus the impact this has. Specifically, the authentication framework uses `get_ext()` in three cases:

1. Retrieving a hash from an X.509 certificate to check the integrity of a child certificate (see `auth_hash()`).
2. Retrieving the signature details from an X.509 certificate to check its authenticity and integrity (see `auth_signature()`).
3. Retrieving the security counter value from an X.509 certificate to protect it from unauthorized rollback to a previous version (see `auth_nvctr()`).

None of these uses authentication framework write to the out-of-bounds memory, so no memory corruption is possible.

In summary, there are 2 separate issues - one in `get_ext()` and another one in `auth_nvctr()` - but neither of these can be exploited in the context of TF-A upstream code.

Only in the following 2 cases do we expect this vulnerability to be triggerable prior to authentication:

- The platform uses a custom chain of trust which uses the non-volatile counter authentication method (`AUTH_METHOD_NV_CTR`) before the cryptographic authentication method (`AUTH_METHOD_SIG`).
- The chain of trust uses a custom authentication method that calls `get_ext()` before cryptographic authentication.

Custom Image Parsers

If the platform uses a custom image parser instead of the certificate parser, the bug in the certificate parser is obviously not relevant. The bug in `auth_nvctr()` *may* be relevant, but only if the returned data is:

- Taken from an untrusted source (meaning that it is read prior to authentication).
- Not already checked to be a primitively-encoded ASN.1 tag.

In particular, if the custom image parser implementation wraps a 32-bit integer in an ASN.1 `INTEGER`, it is not affected.

9.11 Advisory TFV-11 (CVE-2023-49100)

Title	A Malformed SDEI SMC can cause out of bound memory read.
CVE ID	CVE-2023-49100
Date	Reported on 12 Oct 2023
Versions Affected	TF-A releases v1.5 to v2.9 LTS releases lts-v2.8.0 to lts-v2.8.11
Configurations Affected	Platforms with SDEI support
Impact	Denial of Service (secure world panic)
Fix Version	a7eff3477 "fix(sdei): ensure that interrupt ID is valid"
Credit	Christian Lindenmeier @_chli_ Marcel Busch @0ddc0de IT Security Infrastructures Lab

This security advisory describes a vulnerability in the SDEI services, where a rogue Non-secure caller invoking a SDEI_INTERRUPT_BIND SMC call with an invalid interrupt ID causes out of bound memory read.

SDEI_INTERRUPT_BIND is used to bind any physical interrupt into a normal priority SDEI event. The interrupt can be a private peripheral interrupt (PPI) or a shared peripheral interrupt (SPI). Refer to SDEI_INTERRUPT_BIND in the SDEI Specification for further details.

The vulnerability exists when the SDEI client passes an interrupt ID which is not implemented by the GIC. This will result in a data abort exception or a EL3 panic depending on the GIC version used in the system.

- **GICv2 systems:**

```
Call stack:
      sdei_interrupt_bind(interrupt ID)
       -> plat_ic_get_interrupt_type(interrupt ID)
         -> gicv2_get_interrupt_group(interrupt ID)
           -> gicd_get_igroupr(distributor base, interrupt ID)
             -> gicd_read_igroupr(distributor base, interrupt ID).

gicd_read_igroupr() will eventually do a MMIO read to an unimplemented IGROUPR
register. Which may cause a data abort or an access to a random EL3 memory␣
↪region.
```

- **GICv3 systems:**

```
Call stack:
     sdei_interrupt_bind(interrupt ID)
       -> plat_ic_get_interrupt_type(interrupt ID)
         -> gicv3_get_interrupt_group(interrupt ID, core ID)
           -> is_sgi_ppi(interrupt ID)

is_sgi_ppi() will end up in an EL3 panic on encountering an invalid interrupt␣
↪ID.
```

The vulnerability is fixed by ensuring that the Interrupt ID provided by the SDEI client is a valid PPI or SPI, otherwise return an error code indicating that the parameter is invalid.

```
/* Bind an SDEI event to an interrupt */
static int sdei_interrupt_bind(unsigned int intr_num)
{
     sdei_ev_map_t *map;
     bool retry = true, shared_mapping;

     /* Interrupt must be either PPI or SPI */
     if (!(plat_ic_is_ppi(intr_num) || plat_ic_is_spi(intr_num)))
           return SDEI_EINVAL;
```

9.12 Advisory TFV-12 (CVE-2024-5660)

Title	When Hardware Page Aggregation (HPA) is enabled memory accesses may be translated incorrectly.
CVE ID	CVE-2024-5660
Date	Reported on 26 Jan 2024
Versions Affected	TF-A version from v2.2 to v2.12 LTS releases lts-v2.8.0 to lts-v2.8.26 LTS releases lts-v2.10.0 to lts-v2.10.10
Configurations Affected	Arm CPUs with Hardware Page Aggregation (HPA) running in environments where a modified, untrusted guest OS may operate, especially with specific hypervisors.
Impact	Potential for a compromised guest OS to attack the host via HPA mechanism, resulting in possible information disclosure.
Fix Version	Gerrit-Patches
Credit	Arm

9.12.1 Description

A vulnerability has been identified in certain Arm CPUs implementing the Hardware Page Aggregation (HPA) feature. In environments utilizing virtualization, a specially crafted or compromised guest operating system could exploit this vulnerability to affect the host system. This could potentially lead to information disclosure depending on the deployment scenario and hypervisor configuration.

The below table lists the CPUs that mitigate against this vulnerability in TF-A.

Core
Cortex-A77
Cortex-A78
Cortex-A78C
Cortex-A78AE
Cortex-A710
Cortex-X1
Cortex-X2
Cortex-X3
Cortex-X4
Cortex-X925
Neoverse-V1
Neoverse-V2
Neoverse-V3
Neoverse-N2

9.12.2 Mitigation and Recommendations

Arm recommends following the mitigation steps and configuration changes described in the official advisory. The issue is avoided by setting CPUECTLR_EL1[46] to 1 which will disable hardware page aggregation.

Users should refer to the latest firmware updates as provided by vendors and ensure that HPA-related security mitigations are enabled where applicable.

For further technical information, affected CPUs, and detailed guidance, refer to the full Official Arm Advisory.

9.13 Advisory TFV-13 (CVE-2024-7881)

Title	An unprivileged context can trigger a data memory-dependent prefetch engine to fetch the contents of a privileged location and consume those contents as an address that is also dereferenced.
CVE ID	CVE-2024-7881
Date	Reported on 16 August 2024
Versions Affected	TF-A version from v2.2 to v2.12 LTS releases lts-v2.8.0 to lts-v2.8.28 LTS releases lts-v2.10.0 to lts-v2.10.12
Configurations Affected	All
Impact	Potential leakage of secure world data to normal world.
Fix Version	Gerrit topic #ar/smccc_arch_wa_4 Also see mitigation guidance in the Official Arm Advisory
Credit	Arm

9.13.1 Description

An issue has been identified in some Arm-based CPUs that may allow an unprivileged context to trigger a data memory-dependent prefetch engine to fetch the contents of a privileged location (for which it does not have read permission) and consume those contents as an address that is also dereferenced.

The following table identifies all affected CPUs and revisions for which a mitigation is provided in TF-A.

CPU	Affected Versions	Fix Status
cortex-x3	r0p0, r1p0, r1p1, r1p2	open
cortex-x4	r0p0, r0p1, r0p2	fixed in r0p3
cortex-x925	r0p0, r0p1	fixed in r0p2
neoverse-v2	r0p0, r0p1, r0p2	open
neoverse-v3	r0p0, r0p1	fixed in r0p2
neoverse-v3ae	r0p0, r0p1	fixed in r0p2
c1-premium	r0p0	fixed in r1p0
c1-pro	r0p0, r1p0	fixed in r1p1
c1-ultra	r0p0	fixed in r1p0

9.13.2 Mitigation and Recommendations

Arm recommends following the mitigation steps and configuration changes described in the official advisory. The mitigation for CVE-2024-7881 is implemented at EL3 and addresses vulnerabilities caused by memory-dependant speculative prefetching. This issue can be avoided by disabling the affected prefetcher. For most cores, this is done by setting CPUACTLR6_EL1[41] = 1. For C1-Pro, the affected prefetcher is instead disabled by setting IMP_CPUECTLR_EL1[49] = 1.

Arm has updated the SMC Calling Convention spec so that privileged normal world software can identify when the issue has been mitigated in firmware (SMCCC_ARCH_WORKAROUND_4). Refer to the SMC Calling Convention Specification for more details.

The above workaround is enabled by default (on vulnerable CPUs only). Platforms can choose to disable them at compile time if they do not require them.

For further technical information, affected CPUs, and detailed guidance, refer to the full Official Arm Advisory.

9.14 Advisory TFV-14 (CVE-2026-34879)

Title	BL1 FWU range-check mismatch in FWU_SMC_IMAGE_COPY can cause out-of-range memory access.
CVE ID	CVE-2026-34879
Date	Reported on 16 Jan 2026
Versions Affected	TF-A releases with BL1 FWU mode enabled prior to the fixes in 48739 and 48740.
Configurations Affected	Platforms that enable BL1 FWU mode and expose FWU_SMC_IMAGE_COPY to the Normal World.
Impact	Denial of Service (secure world data abort / reboot loop).
Fix Version	48739 "fix(bl1): harden FWU copy/auth overflow checks" 48740 "fix(arm): harden FWU address range checks"
Credit	Prasanth Sundararajan https://www.linkedin.com/in/prasanthsundararajan/

9.14.1 Description

In BL1 Firmware Update (FWU) mode, the FWU_SMC_IMAGE_COPY path validated image ranges using an overflow check based on (size - 1) but then consumed the full block size in subsequent range checks and memcpy operations. This created a mismatch where certain malformed requests could pass validation and later trigger a memcpy from an invalid or unmapped address.

A compromised Normal World component can issue a malformed FWU_SMC_IMAGE_COPY request while the device is in FWU mode, pass validation, and cause BL1 to attempt to copy from an invalid address. In testing, this results in a BL1 data abort and can lead to a secure-side reboot loop (availability impact).

9.14.2 Exploitability Details

This issue is exploitable only when the device is running in BL1 FWU mode and the *FWU_SMC_IMAGE_COPY* interface is reachable from the Normal World. An attacker needs control of a Normal World component capable of issuing SMC calls in FWU mode (for example, a compromised bootloader, recovery image, or update agent). No secure-world code execution is required; the attack is performed entirely through crafted FWU requests.

The attacker submits a request with carefully chosen source address and size parameters that pass the legacy range validation but produce a different effective range when the copy occurs. Because the copy uses the full block size, BL1 attempts to read from an invalid or unmapped address, resulting in a secure-world data abort. In practice this yields a denial of service in FWU mode, often observed as a secure-side reboot loop.

9.14.3 Mitigation and Recommendations

The issue is fixed by making overflow and range validation match the actual block size used by bl1_plat_mem_check() and memcpy(), and by validating end-exclusive ranges computed safely from (base + size).

Users should apply the fixes in the commits referenced below and ensure that all TF-A builds enabling BL1 FWU mode include these changes.

9.15 Advisory TFV-15 (CVE-2026-34878)

Title	Insufficient validation of FIP ToC offsets in BL1/BL2 can cause unintended reads and secure memory disclosure at boot.
CVE ID	CVE-2026-34878
Date	Reported on 22 Jan 2026
Versions Affected	TF-A releases prior to the fixes in 48351, 49485.
Configurations Affected	Platforms that load BL1/BL2 from a FIP and allow firmware update or boot from a modified FIP.
Impact	Secure memory disclosure during boot and potential DoS (boot halts on failed authentication).
Fix Version	48351 “fix(io): validate FIP ToC bounds and catch short reads” 49485 “feat(lib): add u64 overflow helper” checks”
Credit	Kevin Chen

9.15.1 Description

Even when Trusted Boot is enabled, the FIP metadata (header and ToC) is not cryptographically signed by design and must be treated as untrusted input. BL1/BL2 parse the FIP ToC to locate images, calculate offsets and lengths, and perform reads and copies before authentication is performed. The existing logic did not sufficiently validate FIP metadata fields (including ToC offsets, sizes, and read lengths) and did not consistently detect short reads. A crafted or corrupted FIP can therefore cause BL1/BL2 to read and copy unintended data-including secure memory mapped into the EL3 translation regime-into non-secure memory regions, potentially disclosing sensitive contents.

This issue affects boot-time images only (BL1/BL2), not runtime firmware (BL31). The exposure exists only during early boot while BL1/BL2 are parsing the FIP and copying images; once authentication fails, boot halts. In most deployments, useful secrets are not resident in memory during that early boot interval. In configurations where TF-A decrypts confidential images into Trusted SRAM, the disclosure can expose plaintext at boot. Note that BL33 is typically loaded into non-secure memory, so not all image copies target secure buffers during authentication. In practice, the same malformed metadata typically leads to authentication failure and halts the boot process, resulting in a DoS.

9.15.2 Exploitability Details

An attacker must be able to supply or modify the FIP that BL1/BL2 will parse, for example via a firmware update mechanism, removable storage, or a system configuration that allows boot from an untrusted FIP source. Because the FIP metadata is unsigned, Trusted Boot does not protect the ToC from tampering. By crafting offsets and sizes that point outside the actual FIP payload, or that trigger integer overflow, the attacker can force BL1/BL2 to issue reads that extend beyond the FIP and into adjacent regions mapped in secure address space. The incorrect or short reads can then be copied into the destination buffers for authentication, which fails, halting the boot process.

9.15.3 Mitigation and Recommendations

Apply the fixes referenced below. They add strict bounds checks, overflow-safe validation, and short-read detection for FIP metadata parsing and associated copy operations.

Where feasible, reduce exposure by ensuring the FIP is stored in a protected location and by restricting or disabling boot paths that accept externally modifiable FIP images.

9.16 Advisory TFV-16 (CVE-2026-0995)

Title	SME erratum in C1-Pro means memory accesses from the SME unit can remain outstanding after another CPU issues TLBI+DSB
CVE ID	*CVE-2026-0995*
Date	Reported on 23 September 2025
Versions Affected	TF-A version from v2.10 onwards
Configurations Affected	Arm C1-Pro prior to r1p2
Impact	SME can access memory after it has been re-allocated, potentially overwriting the new owner's data.
Fix Version	Gerrit topic #gr/CVE-2026-0995 Also see mitigation guidance in the Official Arm Advisory
Credit	Arm

9.16.1 C1-Pro / CME CVE-2026-0995 Workaround

9.16.2 Overview

C1-Pro/CME **CVE-2026-0995** affects CPUs implementing SME and Streaming mode. Under specific micro-architectural conditions, a `TLBI + DSB` sequence performed on one CPU (PE1) may not guarantee completion of certain in-flight memory accesses performed on another CPU (PE0). As a result, those accesses may complete after translation changes have taken effect, potentially resulting in memory accesses outside the expected translation or privilege boundaries.

To ensure architectural correctness, all affected CPUs must execute a local `DSB` whenever any CPU performs TLB maintenance. The TF-A workaround provides a coordinated EL3 mechanism that guarantees this synchronisation across all online C1-Pro CPUs.

9.16.3 Erratum Status

This erratum applies to C1-Pro multi-core configurations with `NUM_CME != 0` and (per TF-A runtime checks) to C1-Pro revisions up to **r1p2** (inclusive). The erratum is **mitigated in software** through EL3 coordination.

9.16.4 Erratum Description

TLBI + DSB might fail to ensure completion of memory accesses caused by FP/SIMD, SVE, and SME instructions while in Streaming mode, and by LDR/STR ZA/ZT0 instructions.

9.16.5 Description

A TLBI + DSB sequence executed on **PE1** may fail to ensure completion of some memory accesses on **PE0** associated with:

- LDR / STR to or from ZA or ZT0.
- FP/SIMD, SVE, or SME memory accesses while in **Streaming mode**.

9.16.6 Configurations Affected

The erratum affects:

- All **multi-core** configurations
- Where NUM_CME != 0 (i.e., systems using a CME complex)

9.16.7 Conditions

The erratum occurs when all of the following are true:

1. **PE0** executes: * LDR/STR to/from ZA or ZT0, or * Memory accesses tied to FP/SIMD, SVE, or SME execution while in Streaming mode.
2. **PE1** performs: * A TLB invalidate instruction affecting a page used by PE0's memory accesses, followed by A DSB instruction.
3. Complex micro-architectural timing conditions occur.

9.16.8 Implications

If the above conditions are met:

- The DSB on **PE1** may complete **before** certain affected memory accesses on **PE0**, even though those accesses are architecturally in scope for PE1's TLBI.
- As a result, stale or incomplete memory accesses may occur after translation changes have taken effect.
- This can lead to memory being accessed outside the expected translation or privilege boundaries, depending on the software context.

9.16.9 Enabling the Workaround in TF-A

Support for CVE-2026-0995 is build-time selectable and must be enabled by the platform.

To enable the workaround, the platform must:

- Set WORKAROUND_CVE_2026_0995=1.
- Include the C1-Pro workaround source in BL31_SOURCES.
- Include the CPU service sources used by the SMC interface.

For example, a platform that contains affected C1-Pro CPUs should add:

```
WORKAROUND_CVE_2026_0995 := 1

ifeq (${WORKAROUND_CVE_2026_0995},1)
BL31_SOURCES += lib/cpus/aarch64/c1_pro_pubsub.c \
                ${CPU_SVC_SRCS}
endif
```

If the option is not enabled, or if the platform does not include the workaround source and CPU service sources, TF-A will not mitigate this erratum.

9.16.10 Why the Workaround Must Be Implemented in EL3

Due to interaction of Linux, pKVM, and GIC security states:

- The non-secure world cannot reliably deliver SGIs without races.
- A CPU entering `CPU_OFF` may be unable to receive SGIs without violating PSCI rules.
- Interrupt masking and CPU power-down sequences may interfere with SGI delivery.

EL3 is the only domain capable of:

- Issuing **secure SGIs**.
- Tracking CPU on/off and suspend/resume transitions.
- Ensuring correct ordering during secure world entry/exit.
- Avoiding SGI interference during CPU power-down.

9.16.11 Workaround Mechanism

The TF-A workaround ensures that:

- Every affected CPU performs a **local DSB** whenever another CPU performs TLB maintenance.
- A coordinated, EL3-managed SGI rendezvous ensures all online C1-Pro CPUs participate.

The mechanism uses:

- A **global atomic counter** incremented by the SMC caller.
- **Local counters** on each CPU tracking participation in each epoch.
- **Secure SGIs** sent to all active C1-Pro CPUs.

  The reference implementation uses the EL3 secure SGI number `ARM_IRQ_SEC_SGI_6`.
- A **wait-for-completion loop** ensuring all CPUs have executed the mitigation before returning from SMC.

9.16.12 Global Counter and Ordering

Each SMC caller performs:

1. `atomic_inc_return(global_counter)`
2. A barrier ensuring visibility before SGIs: `dmbish`

The returned counter value is treated as that caller's **deadline**: all CPUs must update their local counter to at least this value.

9.16.13 SGI Rendezvous

After incrementing the counter:

1. The caller sends a secure SGI to all online C1-Pro CPUs.
2. Receivers execute: * A local `DSB` (the mitigation) * Update their local counter from the global counter
3. The caller waits until all CPUs have reached its deadline.

9.16.14 Handling PSCI `CPU_OFF` Races

SGIs may race with CPUs powering down.

To avoid violating PSCI semantics:

1. The caller sends a secure SGI to all online C1-Pro CPUs.
2. Receivers execute the local mitigation sequence, which: * May include a `DSB` if required by architectural configuration (for example, when `SCTLR_EL3.IESB` is not set). * Updates the local counter from the global counter.
3. The caller waits until all CPUs have reached its deadline value.

9.16.15 Tracking Active C1-Pro CPUs

TF-A maintains:

- A per-core signed bytemap tracking whether a C1-Pro CPU is currently *active* (reference count 0/1).
- Per-CPU MPIDRs for SGI targeting.

The implementation supports up to 64 cores (`PLATFORM_CORE_COUNT <= 64`); the SMC handler uses a 64-bit mask to track which CPUs were sent SGIs.

EL3 updates this information on:

- `psci_cpu_on_finish`
- `psci_cpu_off_finish`
- `psci_suspend_pwrdown_start` / `finish`

- `cm_entering_secure_world` / `exited_secure_world`

9.16.16 Memory Ordering Requirements

Strong ordering is required to make the SGI rendezvous reliable across cores and across PSCI/world-switch transitions:

`dsb()`
Drains outstanding memory accesses. On C1-Pro, the workaround performs a local `dsb` **only when** `SCTLR_EL3.IESB` is **not** set; if `SCTLR_EL3.IESB` is set, taking an exception to EL3 is sufficient and the explicit `dsb` can be skipped.

`dmbish`
Ensures ordering/visibility between: * the atomic increment of `global_counter` and subsequent SGI delivery, and * updates to the active-core bytemap before using it to decide who must participate in a rendezvous.

`isb`
Used in the SGI handler to ensure the counter load occurs after interrupt acknowledge (prevents stale speculative loads if SGIs merge).

9.16.17 Summary

CVE-2026-0995 allows **stale SME/SIMD/SVE memory accesses to persist beyond TLBI + DSB**, violating architectural expectations and potentially compromising security.

The TF-A mitigation:

- Ensures **every active C1-Pro CPU** drains outstanding accesses.
- Coordinates CPUs via **secure SGIs** and **atomic counters**.
- Handles PSCI races safely in EL3.
- Is enabled at build time via: `WORKAROUND_CVE_2026_0995=1`

The result is a robust, race-free mitigation suitable for systems deploying any security-sensitive workloads.

9.16.18 OS Coordination Requirement

Full mitigation of CVE-2026-0995 requires coordinated updates in both:

- Trusted Firmware-A (EL3) — this advisory
- The Operating System

The EL3 implementation provides a secure SGI-based rendezvous mechanism. The Operating System must invoke the defined SMC interface when performing affected TLB maintenance operations to ensure architectural ordering across all C1-Pro CPUs.

Deploying only one side of the mitigation is insufficient to guarantee architectural ordering and full protection.

For further information, affected CPUs, and detailed guidance, refer to the full Official Arm Advisory.

9.17 Advisory TFV-17 (CVE-2025-10263)

Title	TLBI+DSB might complete too early
CVE ID	CVE-2025-10263
Date	Reported on 13 August 2025
Versions Affected	All, up to and including v2.15
Configurations Affected	All platforms containing any of the following cores: (note that if even a single affected core is present that all cores must implement the workaround, which the TFA patch does) • Cortex-A76 • Cortex-A76AE • Cortex-A77 • Cortex-X1 • Cortex-X1C • Cortex-A78 • Cortex-A78C • Cortex-A78AE • Cortex-A710 • Cortex-X2 • Cortex-X3 • Cortex-X4 • Cortex-X925 • Neoverse-N1 • Neoverse-N2 • Neoverse-V1 • Neoverse-V2 • Neoverse-V3 • Neoverse-V3AE • C1-Ultra • C1-Premium
Impact	Exploitation may allow an attacker to corrupt page table entries or other memory-management data structures during translation invalidation operations.
Fix Version	*Commit 08fab0f083db* (merged on 16 June 2026) Gerrit topic #gr/CVE-2025-10263
Credit	Arm

9.17.1 Description

A race condition exists in affected Arm CPUs whereby a memory store performed by one Processing Element (PE) may remain incomplete after another PE has invalidated the corresponding translation and completed the architecturally required TLBI and DSB sequence. Under specific microarchitectural conditions, the assumption that all memory accesses made using the invalidated translation table entry have completed might not hold true.

This behavior can result in memory corruption affecting translation tables and other protected memory structures. Successful exploitation may lead to denial of service or privilege escalation within the affected translation regime.

Practical exploitation has been demonstrated to the extent of page table corruption. Feasible privilege escalation is considered possible, although full exploitation has not yet been publicly demonstrated.

9.17.2 Conditions

1. PE A executes a store that crosses a page boundary.
2. PE B executes a TLB invalidate instruction which affects at least one of pages targeted by PE A's store, followed by a DSB.
3. Complex micro-architectural conditions occur.
4. PE B's DSB may complete before the global observation of a portion of PE A's store which was affected by the TLB invalidation. The relevant TLB entries are invalidated correctly before the completion of the DSB.

9.17.3 Workaround

This can be avoided by executing the TLBI and DSB instructions a 2nd time. For code sequences that have multiple TLB invalidate instuctions followed by a single DSB, only the last TLB invalidate and DSB need to be repeated.

9.17.4 Impact on TF-A

TF-a uses an affected instruction sequence only in the Xlat library, so the workaround is inserted at the end of that function. Unlike most CVEs, this workaround is disabled by default due to it being needed when a platform contains any single CPU that is affected. So it must be explicitly enabled using the flag WORKAROUND_CVE_2025_10263 by the platform when an affected CPU is present.

For further information, affected CPUs, and detailed guidance, refer to the full Official Arm Advisory.

CHAPTER

TEN

DESIGN DOCUMENTS

10.1 TF-A CMake buildsystem

Author
Balint Dobszay

Organization
Arm Limited

Contact
Balint Dobszay <balint.dobszay@arm.com>

Status
Accepted

Table of Contents

- *TF-A CMake buildsystem*
  - *Abstract*
  - *Introduction*
  - *Main features*
    - *Structured configuration description*
    - *Target description*
    - *Compiler abstraction*
    - *External tools*
  - *Workflow*
  - *Example*

10.1.1 Abstract

This document presents a proposal for a new buildsystem for TF-A using CMake, and as part of this a reusable CMake framework for embedded projects.

10.1.2 Introduction

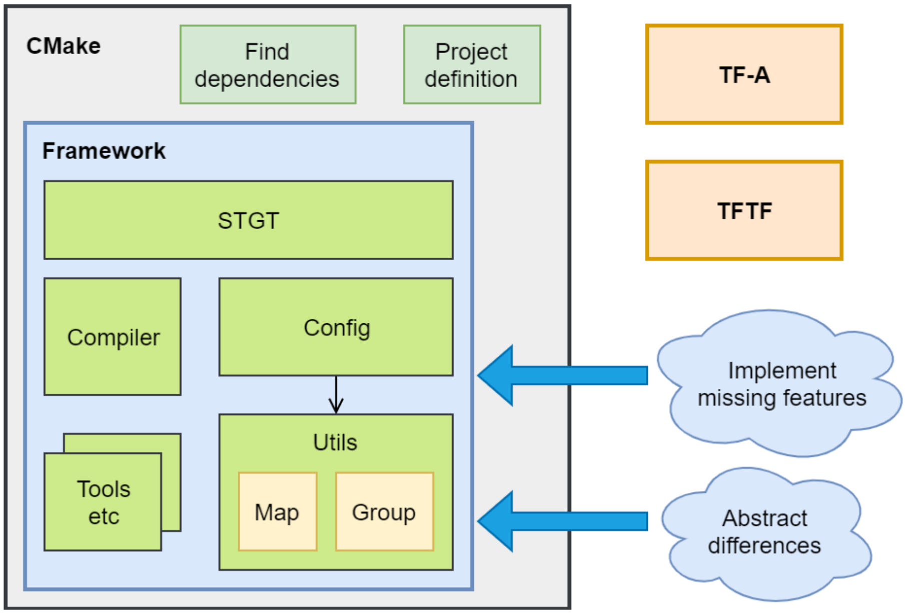
The current Makefile based buildsystem of TF-A has become complicated and hard to maintain, there is a need for a new, more flexible solution. The proposal is to use CMake language for the new buildsystem. The main reasons of this decision are the following:

- It is a well-established, mature tool, widely accepted by open-source projects.
- TF-M is already using CMake, reducing fragmentation for tf.org projects can be beneficial.
- CMake has various advantages over Make, e.g.:
  - Host and target system agnostic project.
  - CMake project is scalable, supports project modularization.
  - Supports software integration.
  - Out-of-the-box support for integration with several tools (e.g. project generation for various IDEs, integration with cppcheck, etc).

Of course there are drawbacks too:

- Language is problematic (e.g. variable scope).
- Not embedded approach.

To overcome these and other problems, we need to create workarounds for some tasks, wrap CMake functions, etc. Since this functionality can be useful in other embedded projects too, it is beneficial to collect the new code into a reusable framework and store this in a separate repository. The following diagram provides an overview of the framework structure:



10.1.3 Main features

Structured configuration description

In the current Makefile system the build configuration description, validation, processing, and the target creation, source file description are mixed and spread across several files. One of the goals of the framework is to organize this.

The framework provides a solution to describe the input build parameters, flags, macros, etc. in a structured way. It contains two utilities for this purpose:

- Map: simple key-value pair implementation.
- Group: collection of related maps.

The related parameters shall be packed into a group (or "setting group"). The setting groups shall be defined and filled with content in config files. Currently the config files are created and edited manually, but later a configuration management tool (e.g. Kconfig) shall be used to generate these files. Therefore, the framework does not contain parameter validation and conflict checking, these shall be handled by the configuration tool.

Target description

The framework provides an API called STGT ('simple target') to describe the targets, i.e. what is the build output, what source files are used, what libraries are linked, etc. The API wraps the CMake target functions, and also extends the built-in functionality, it can use the setting groups described in the previous section. A group can be applied onto a target, i.e. a collection of macros, flags, etc. can be applied onto the given output executable/library. This provides a more granular way than the current Makefile system where most of these are global and applied onto each target.

Compiler abstraction

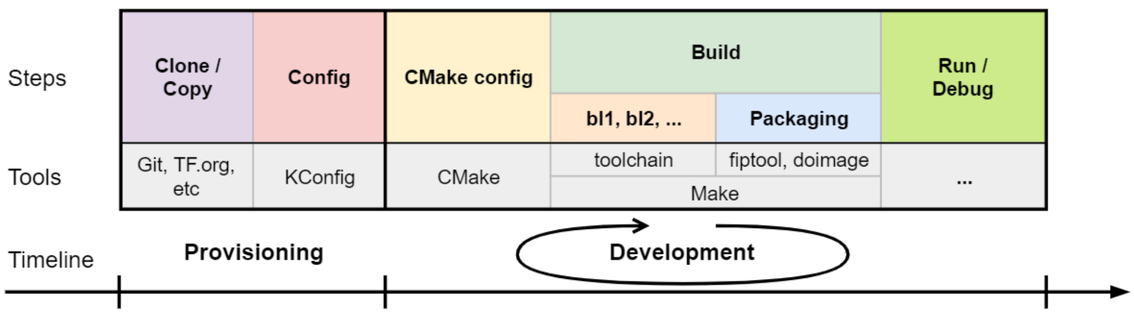
Apart from the built-in CMake usage of the compiler, there are some common tasks that CMake does not solve (e.g. preprocessing a file). For these tasks the framework uses wrapper functions instead of direct calls to the compiler. This way it is not tied to one specific compiler.

External tools

In the TF-A buildsystem some external tools are used, e.g. fiptool for image generation or dtc for device tree compilation. These tools have to be found and/or built by the framework. For this, the CMake find_package functionality is used, any other necessary tools can be added later.

10.1.4 Workflow

The following diagram demonstrates the development workflow using the framework:



The process can be split into two main phases:

In the provisioning phase, first we have to obtain the necessary resources, i.e. clone the code repository and other dependencies. Next we have to do the configuration, preferably using a config tool like KConfig.

In the development phase first we run CMake, which will generate the buildsystem using the selected generator backend (currently only the Makefile generator is supported). After this we run the selected build tool which in turn calls the compiler, linker, packaging tool, etc. Finally we can run and debug the output executables.

Usually during development only the steps in this second phase have to be repeated, while the provisioning phase needs to be done only once (or rarely).

10.1.5 Example

This is a short example for the basic framework usage.

First, we create a setting group called *mem_conf* and fill it with several parameters. It is worth noting the difference between *CONFIG* and *DEFINE* types: the former is only a CMake domain option, the latter is only a C language macro.

Next, we create a target called *fw1* and add the *mem_conf* setting group to it. This means that all source and header files used by the target will have all the parameters declared in the setting group. Then we set the target type to executable, and add some source files. Since the target has the parameters from the settings group, we can use it for conditionally adding source files. E.g. *dram_controller.c* will only be added if MEM_TYPE equals dram.

```
group_new(NAME mem_conf)
group_add(NAME mem_conf TYPE DEFINE KEY MEM_SIZE VAL 1024)
group_add(NAME mem_conf TYPE CONFIG DEFINE KEY MEM_TYPE VAL dram)
group_add(NAME mem_conf TYPE CFLAG KEY -Os)

stgt_create(NAME fw1)
stgt_add_setting(NAME fw1 GROUPS mem_conf)
stgt_set_target(NAME fw1 TYPE exe)

stgt_add_src(NAME fw1 SRC
    ${CMAKE_SOURCE_DIR}/main.c
)

stgt_add_src_cond(NAME fw1 KEY MEM_TYPE VAL dram SRC
    ${CMAKE_SOURCE_DIR}/dram_controller.c
)
```

10.2 Interaction between Measured Boot and an fTPM (PoC)

Measured Boot is the process of cryptographically measuring the code and critical data used at boot time, for example using a TPM, so that the security state can be attested later.

The current implementation of the driver included in *TF-A* supports several backends and each has a different means to store the measurements. This section focuses on the TCG event log backend, which stores measurements in secure memory.

See details of *Measured Boot Design*.

The driver also provides mechanisms to pass the Event Log to normal world if needed.

This manual provides instructions to build a proof of concept (PoC) with the sole intention of showing how Measured Boot can be used in conjunction with a firmware TPM (fTPM) service implemented on top of OP-TEE.

Note: The instructions given in this document are meant to be used to build a PoC to show how Measured Boot on TF-A can interact with a third party (f)TPM service and they try to be as general as possible. Different platforms might have different needs and configurations (e.g. different SHA algorithms) and they might also use different types of TPM services (or even a different type of service to provide the attestation) and therefore the instructions given here might not apply in such scenarios.

10.2.1 Components

The PoC is built on top of the OP-TEE Toolkit, which has support to build TF-A with support for Measured Boot enabled (and run it on a Foundation Model) since commit cf56848.

The aforementioned toolkit builds a set of images that contain all the components needed to test that the Event Log was properly created. One of these images will contain a third party fTPM service which in turn will be used to process the Event Log.

The reason to choose OP-TEE Toolkit to build our PoC around it is mostly for convenience. As the fTPM service used is an OP-TEE TA, it was easy to add build support for it to the toolkit and then build the PoC around it.

The most relevant components installed in the image that are closely related to Measured Boot/fTPM functionality are:

- **OP-TEE**: As stated earlier, the fTPM service used in this PoC is built as an OP-TEE TA and therefore we need to include the OP-TEE OS image. Support to interfacing with Measured Boot was added to version 3.9.0 of OP-TEE by implementing the `PTA_SYSTEM_GET_TPM_EVENT_LOG` syscall, which allows the former to pass a copy of the Event Log to any TA requesting it. OP-TEE knows the location of the Event Log by reading the DTB bindings received from TF-A. Visit *DTB binding for Event Log properties* for more details on this.
- **fTPM Service**: We use a third party fTPM service in order to validate the Measured Boot functionality. The chosen fTPM service is a sample implementation for Aarch32 architecture included on the ms-tpm-20-ref reference implementation from Microsoft. The service was updated in order to extend the Measured Boot Event Log at boot up and it uses the aforementioned `PTA_SYSTEM_GET_TPM_EVENT_LOG` call to retrieve a copy of the former.

Note: Arm does not provide an fTPM implementation. The fTPM service used here is a third party one which has been updated to support Measured Boot service as provided by TF-A. As such, it is beyond the scope of this manual to test and verify the correctness of the output generated by the fTPM service.

- **TPM Kernel module**: In order to interact with the fTPM service, we need a kernel module to forward the request from user space to the secure world.
- tpm2-tools: This is a set of tools that allow to interact with the fTPM service. We use this in order to read the PCRs with the measurements.

10.2.2 Building the PoC for the Arm FVP platform

As mentioned before, this PoC is based on the OP-TEE Toolkit with some extensions to enable Measured Boot and an fTPM service. Therefore, we can rely on the instructions to build the original OP-TEE Toolkit. As a general rule, the following steps should suffice:

(1) Start by following the Get and build the solution instructions to build the OP-TEE toolkit. On step 3, you need to get the manifest for FVP platform from the main branch:

```
$ repo init -u https://github.com/OP-TEE/manifest.git -m fvp.xml
```

Then proceed synching the repos as stated in step 3. Continue following the instructions and stop before step 5.

(2) Next you should obtain the Armv8-A Foundation Platform (For Linux Hosts Only). The binary should be untar'ed to the root of the repo tree, i.e., like this: `<fvp-project>/Foundation_Platformpkg`. In the end, after cloning all source code, getting the toolchains and "installing" Foundation_Platformpkg, you should have a folder structure that looks like this:

```
$ ls -la
total 80
drwxrwxr-x 20 tf-a_user tf-a_user 4096 Jul  1 12:16 .
drwxr-xr-x 23 tf-a_user tf-a_user 4096 Jul  1 10:40 ..
drwxrwxr-x 12 tf-a_user tf-a_user 4096 Jul  1 10:45 build
drwxrwxr-x 16 tf-a_user tf-a_user 4096 Jul  1 12:16 buildroot
drwxrwxr-x 51 tf-a_user tf-a_user 4096 Jul  1 10:45 edk2
drwxrwxr-x  6 tf-a_user tf-a_user 4096 Jul  1 12:14 edk2-platforms
drwxr-xr-x  7 tf-a_user tf-a_user 4096 Jul  1 10:52 Foundation_
↪Platformpkg
drwxrwxr-x 17 tf-a_user tf-a_user 4096 Jul  2 10:40 grub
drwxrwxr-x 25 tf-a_user tf-a_user 4096 Jul  2 10:39 linux
drwxrwxr-x 15 tf-a_user tf-a_user 4096 Jul  1 10:45 mbedtls
drwxrwxr-x  6 tf-a_user tf-a_user 4096 Jul  1 10:45 ms-tpm-20-ref
drwxrwxr-x  8 tf-a_user tf-a_user 4096 Jul  1 10:45 optee_client
drwxrwxr-x 10 tf-a_user tf-a_user 4096 Jul  1 10:45 optee_examples
drwxrwxr-x 12 tf-a_user tf-a_user 4096 Jul  1 12:13 optee_os
drwxrwxr-x  8 tf-a_user tf-a_user 4096 Jul  1 10:45 optee_test
drwxrwxr-x  7 tf-a_user tf-a_user 4096 Jul  1 10:45 .repo
drwxrwxr-x  4 tf-a_user tf-a_user 4096 Jul  1 12:12 toolchains
drwxrwxr-x 21 tf-a_user tf-a_user 4096 Jul  1 12:15 trusted-firmware-a
```

(3) Now enter into `ms-tpm-20-ref` and get its dependencies:

```
$ cd ms-tpm-20-ref
$ git submodule init
$ git submodule update
Submodule path 'external/wolfssl': checked out
↪'9c87f979a7f1d3a6d786b260653d566c1d31a1c4'
```

(4) Now, you should be able to continue with step 5 in "Get and build the solution" instructions. In order to enable support for Measured Boot, you need to set the following build options:

```
$ MEASURED_BOOT=y MEASURED_BOOT_FTPM=y make -j `nproc`
```

Note: The build process will likely take a long time. It is strongly recommended to pass the `-j` option to make to run the process faster.

After this step, you should be ready to run the image.

10.2.3 Running and using the PoC on the Armv8-A Foundation AEM FVP

With everything built, you can now run the image:

```
$ make run-only
```

Note: Using `make run` will build and run the image and it can be used instead of simply `make`. However, once the image is built, it is recommended to use `make run-only` to avoid re-running all the building rules, which would take time.

When FVP is launched, two terminal windows will appear. `FVP terminal_0` is the userspace terminal whereas `FVP terminal_1` is the counterpart for the secure world (where TAs will print their logs, for instance).

Log into the image shell with user `root`, no password will be required. Then we can issue the `ftpm` command, which is an alias that

(1) loads the ftpm kernel module and

(2) calls `tpm2_pcrread`, which will access the fTPM service to read the PCRs.

When loading the ftpm kernel module, the fTPM TA is loaded into the secure world. This TA then requests a copy of the Event Log generated during the booting process so it can retrieve all the entries on the log and record them first thing.

Note: For this PoC, nothing loaded after BL33 and NT_FW_CONFIG is recorded in the Event Log.

The secure world terminal should show the debug logs for the fTPM service, including all the measurements available in the Event Log as they are being processed:

```
M/TA: Preparing to extend the following TPM Event Log:
M/TA: TCG_EfiSpecIDEvent:
M/TA:   PCRIndex           : 0
M/TA:   EventType          : 3
M/TA:   Digest             : 00
M/TA:                      : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M/TA:                      : 00 00 00
M/TA:   EventSize          : 33
```

(continues on next page)

(continued from previous page)

```
M/TA:   Signature           : Spec ID Event03
M/TA:   PlatformClass       : 0
M/TA:   SpecVersion         : 2.0.2
M/TA:   UintnSize           : 1
M/TA:   NumberOfAlgorithms  : 1
M/TA:   DigestSizes         :
M/TA:     #0 AlgorithmId    : SHA256
M/TA:        DigestSize     : 32
M/TA:   VendorInfoSize      : 0
M/TA: PCR_Event2:
M/TA:   PCRIndex            : 0
M/TA:   EventType           : 3
M/TA:   Digests Count       : 1
M/TA:     #0 AlgorithmId    : SHA256
M/TA:        Digest         : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M/TA:                       : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M/TA:   EventSize           : 17
M/TA:   Signature           : StartupLocality
M/TA:   StartupLocality     : 0
M/TA: PCR_Event2:
M/TA:   PCRIndex            : 0
M/TA:   EventType           : 1
M/TA:   Digests Count       : 1
M/TA:     #0 AlgorithmId    : SHA256
M/TA:        Digest         : 58 26 32 6e 64 45 64 da 45 de 35 db 96 fd ed 63
M/TA:                       : 2a 6a d4 0d aa 94 b0 b1 55 e4 72 e7 1f 0a e0 d5
M/TA:   EventSize           : 5
M/TA:   Event               : BL_2
M/TA: PCR_Event2:
M/TA:   PCRIndex            : 0
M/TA:   EventType           : 1
M/TA:   Digests Count       : 1
M/TA:     #0 AlgorithmId    : SHA256
M/TA:        Digest         : cf f9 7d a3 5c 73 ac cb 7b a0 25 80 6a 6e 50 a5
M/TA:                       : 6b 2e d2 8c c9 36 92 7d 46 c5 b9 c3 a4 6c 51 7c
M/TA:   EventSize           : 6
M/TA:   Event               : BL_31
M/TA: PCR_Event2:
M/TA:   PCRIndex            : 0
M/TA:   EventType           : 1
M/TA:   Digests Count       : 1
M/TA:     #0 AlgorithmId    : SHA256
M/TA:        Digest         : 23 b0 a3 5d 54 d9 43 1a 5c b9 89 63 1c da 06 c2
M/TA:                       : e5 de e7 7e 99 17 52 12 7d f7 45 ca 4f 4a 39 c0
M/TA:   EventSize           : 10
M/TA:   Event               : HW_CONFIG
M/TA: PCR_Event2:
M/TA:   PCRIndex            : 0
M/TA:   EventType           : 1
M/TA:   Digests Count       : 1
M/TA:     #0 AlgorithmId    : SHA256
M/TA:        Digest         : 4e e4 8e 5a e6 50 ed e0 b5 a3 54 8a 1f d6 0e 8a
```

(continues on next page)

(continued from previous page)

```
M/TA:                           : ea 0e 71 75 0e a4 3f 82 76 ce af cd 7c b0 91 e0
M/TA:    EventSize              : 14
M/TA:    Event                  : SOC_FW_CONFIG
M/TA: PCR_Event2:
M/TA:    PCRIndex               : 0
M/TA:    EventType              : 1
M/TA:    Digests Count          : 1
M/TA:      #0 AlgorithmId       : SHA256
M/TA:         Digest            : 01 b0 80 47 a1 ce 86 cd df 89 d2 1f 2e fc 6c 22
M/TA:                           : f8 19 ec 6e 1e ec 73 ba 5a be d0 96 e3 5f 6d 75
M/TA:    EventSize              : 6
M/TA:    Event                  : BL_32
M/TA: PCR_Event2:
M/TA:    PCRIndex               : 0
M/TA:    EventType              : 1
M/TA:    Digests Count          : 1
M/TA:      #0 AlgorithmId       : SHA256
M/TA:         Digest            : 5d c6 ef 35 5a 90 81 b4 37 e6 3b 52 da 92 ab 8e
M/TA:                           : d9 6e 93 98 2d 40 87 96 1b 5a a7 ee f1 f4 40 63
M/TA:    EventSize              : 18
M/TA:    Event                  : BL32_EXTRA1_IMAGE
M/TA: PCR_Event2:
M/TA:    PCRIndex               : 0
M/TA:    EventType              : 1
M/TA:    Digests Count          : 1
M/TA:      #0 AlgorithmId       : SHA256
M/TA:         Digest            : 39 b7 13 b9 93 db 32 2f 1b 48 30 eb 2c f2 5c 25
M/TA:                           : 00 0f 38 dc 8e c8 02 cd 79 f2 48 d2 2c 25 ab e2
M/TA:    EventSize              : 6
M/TA:    Event                  : BL_33
M/TA: PCR_Event2:
M/TA:    PCRIndex               : 0
M/TA:    EventType              : 1
M/TA:    Digests Count          : 1
M/TA:      #0 AlgorithmId       : SHA256
M/TA:         Digest            : 25 10 60 5d d4 bc 9d 82 7a 16 9f 8a cc 47 95 a6
M/TA:                           : fd ca a0 c1 2b c9 99 8f 51 20 ff c6 ed 74 68 5a
M/TA:    EventSize              : 13
M/TA:    Event                  : NT_FW_CONFIG
```

These logs correspond to the measurements stored by TF-A during the measured boot process and therefore, they should match the logs dumped by the former during the boot up process. These can be seen on the terminal_0:

```
NOTICE:  Booting Trusted Firmware
NOTICE:  BL1: v2.5(release):v2.5
NOTICE:  BL1: Built : 10:41:20, Jul  2 2021
NOTICE:  BL1: Booting BL2
NOTICE:  BL2: v2.5(release):v2.5
NOTICE:  BL2: Built : 10:41:20, Jul  2 2021
NOTICE:  TCG_EfiSpecIDEvent:
```

(continues on next page)

(continued from previous page)

```
NOTICE:    PCRIndex            : 0
NOTICE:    EventType           : 3
NOTICE:    Digest              : 00
NOTICE:                        : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00
NOTICE:                        : 00 00 00
NOTICE:    EventSize           : 33
NOTICE:    Signature           : Spec ID Event03
NOTICE:    PlatformClass       : 0
NOTICE:    SpecVersion         : 2.0.2
NOTICE:    UintnSize           : 1
NOTICE:    NumberOfAlgorithms  : 1
NOTICE:    DigestSizes         :
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         DigestSize     : 32
NOTICE:    VendorInfoSize      : 0
NOTICE:  PCR_Event2:
NOTICE:    PCRIndex            : 0
NOTICE:    EventType           : 3
NOTICE:    Digests Count       : 1
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         Digest         : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00
NOTICE:                        : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00
NOTICE:    EventSize           : 17
NOTICE:    Signature           : StartupLocality
NOTICE:    StartupLocality     : 0
NOTICE:  PCR_Event2:
NOTICE:    PCRIndex            : 0
NOTICE:    EventType           : 1
NOTICE:    Digests Count       : 1
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         Digest         : 58 26 32 6e 64 45 64 da 45 de 35 db 96 fd ed␣
↪63
NOTICE:                        : 2a 6a d4 0d aa 94 b0 b1 55 e4 72 e7 1f 0a e0␣
↪d5
NOTICE:    EventSize           : 5
NOTICE:    Event               : BL_2
NOTICE:  PCR_Event2:
NOTICE:    PCRIndex            : 0
NOTICE:    EventType           : 1
NOTICE:    Digests Count       : 1
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         Digest         : cf f9 7d a3 5c 73 ac cb 7b a0 25 80 6a 6e 50␣
↪a5
NOTICE:                        : 6b 2e d2 8c c9 36 92 7d 46 c5 b9 c3 a4 6c 51␣
↪7c
NOTICE:    EventSize           : 6
NOTICE:    Event               : BL_31
NOTICE:  PCR_Event2:
NOTICE:    PCRIndex            : 0
```

(continues on next page)

(continued from previous page)

```
NOTICE:    EventType           : 1
NOTICE:    Digests Count       : 1
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         Digest         : 23 b0 a3 5d 54 d9 43 1a 5c b9 89 63 1c da 06␣
↪c2
NOTICE:                        : e5 de e7 7e 99 17 52 12 7d f7 45 ca 4f 4a 39␣
↪c0
NOTICE:    EventSize           : 10
NOTICE:    Event               : HW_CONFIG
NOTICE:  PCR_Event2:
NOTICE:    PCRIndex            : 0
NOTICE:    EventType           : 1
NOTICE:    Digests Count       : 1
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         Digest         : 4e e4 8e 5a e6 50 ed e0 b5 a3 54 8a 1f d6 0e␣
↪8a
NOTICE:                        : ea 0e 71 75 0e a4 3f 82 76 ce af cd 7c b0 91␣
↪e0
NOTICE:    EventSize           : 14
NOTICE:    Event               : SOC_FW_CONFIG
NOTICE:  PCR_Event2:
NOTICE:    PCRIndex            : 0
NOTICE:    EventType           : 1
NOTICE:    Digests Count       : 1
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         Digest         : 01 b0 80 47 a1 ce 86 cd df 89 d2 1f 2e fc 6c␣
↪22
NOTICE:                        : f8 19 ec 6e 1e ec 73 ba 5a be d0 96 e3 5f 6d␣
↪75
NOTICE:    EventSize           : 6
NOTICE:    Event               : BL_32
NOTICE:  PCR_Event2:
NOTICE:    PCRIndex            : 0
NOTICE:    EventType           : 1
NOTICE:    Digests Count       : 1
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         Digest         : 5d c6 ef 35 5a 90 81 b4 37 e6 3b 52 da 92 ab␣
↪8e
NOTICE:                        : d9 6e 93 98 2d 40 87 96 1b 5a a7 ee f1 f4 40␣
↪63
NOTICE:    EventSize           : 18
NOTICE:    Event               : BL32_EXTRA1_IMAGE
NOTICE:  PCR_Event2:
NOTICE:    PCRIndex            : 0
NOTICE:    EventType           : 1
NOTICE:    Digests Count       : 1
NOTICE:      #0 AlgorithmId    : SHA256
NOTICE:         Digest         : 39 b7 13 b9 93 db 32 2f 1b 48 30 eb 2c f2 5c␣
↪25
NOTICE:                        : 00 0f 38 dc 8e c8 02 cd 79 f2 48 d2 2c 25 ab␣
↪e2
NOTICE:    EventSize           : 6
```

(continues on next page)

(continued from previous page)

```
NOTICE:     Event                : BL_33
NOTICE:   PCR_Event2:
NOTICE:     PCRIndex             : 0
NOTICE:     EventType            : 1
NOTICE:     Digests Count        : 1
NOTICE:       #0 AlgorithmId     : SHA256
NOTICE:          Digest          : 25 10 60 5d d4 bc 9d 82 7a 16 9f 8a cc 47 95␣
↪a6
NOTICE:                          : fd ca a0 c1 2b c9 99 8f 51 20 ff c6 ed 74 68␣
↪5a
NOTICE:     EventSize            : 13
NOTICE:     Event                : NT_FW_CONFIG
NOTICE:   BL1: Booting BL31
NOTICE:   BL31: v2.5(release):v2.5
NOTICE:   BL31: Built : 10:41:20, Jul  2 2021
```

Following up with the fTPM startup process, we can see that all the measurements in the Event Log are extended and recorded in the appropriate PCR:

```
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: TPM2_PCR_EXTEND_COMMAND returned value:
M/TA:   ret_tag = 0x8002, size = 0x00000013, rc = 0x00000000
M/TA: 9 Event logs processed
```

After the fTPM TA is loaded, the call to `insmod` issued by the `ftpm` alias to load the ftpm kernel module returns, and then the TPM PCRs are read by means of `tpm_pcrread` command. Note that we are only interested in the SHA256 logs here, as this is the algorithm we used on TF-A for the measurements (see the field `AlgorithmId` on the logs above):

```
sha256:
0 : 0xA6EB3A7417B8CFA9EBA2E7C22AD5A4C03CDB8F3FBDD7667F9C3EF2EA285A8C9F
1 : 0x0000000000000000000000000000000000000000000000000000000000000000
2 : 0x0000000000000000000000000000000000000000000000000000000000000000
3 : 0x0000000000000000000000000000000000000000000000000000000000000000
4 : 0x0000000000000000000000000000000000000000000000000000000000000000
5 : 0x0000000000000000000000000000000000000000000000000000000000000000
```

(continues on next page)

(continued from previous page)

```
6 : 0x0000000000000000000000000000000000000000000000000000000000000000
7 : 0x0000000000000000000000000000000000000000000000000000000000000000
8 : 0x0000000000000000000000000000000000000000000000000000000000000000
9 : 0x0000000000000000000000000000000000000000000000000000000000000000
10: 0x0000000000000000000000000000000000000000000000000000000000000000
11: 0x0000000000000000000000000000000000000000000000000000000000000000
12: 0x0000000000000000000000000000000000000000000000000000000000000000
13: 0x0000000000000000000000000000000000000000000000000000000000000000
14: 0x0000000000000000000000000000000000000000000000000000000000000000
15: 0x0000000000000000000000000000000000000000000000000000000000000000
16: 0x0000000000000000000000000000000000000000000000000000000000000000
17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
23: 0x0000000000000000000000000000000000000000000000000000000000000000
```

In this PoC we are only interested in PCR0, which must be non-null. This is because the boot process records all the images in this PCR (see field `PCRIndex` on the Event Log above). The rest of the records must be 0 at this point.

Note: The fTPM service used has support only for 16 PCRs, therefore the content of PCRs above 15 can be ignored.

Note: As stated earlier, Arm does not provide an fTPM implementation and therefore we do not validate here if the content of PCR0 is correct or not. For this PoC, we are only focused on the fact that the event log could be passed to a third party fTPM and its records were properly extended.

10.2.4 Fine-tuning the fTPM TA

As stated earlier, the OP-TEE Toolkit includes support to build a third party fTPM service. The build options for this service are tailored for the PoC and defined in the build environment variable `FTPM_FLAGS` (see `<toolkit_home>/build/common.mk`) but they can be modified if needed to better adapt it to a specific scenario.

The most relevant options for Measured Boot support are:

- **CFG_TA_DEBUG**: Enables debug logs in the Terminal_1 console.
- **CFG_TEE_TA_LOG_LEVEL**: Defines the log level used for the debug messages.
- **CFG_TA_MEASURED_BOOT**: Enables support for measured boot on the fTPM.
- **CFG_TA_EVENT_LOG_SIZE**: Defines the size, in bytes, of the larger event log that the fTPM is able to store, as this buffer is allocated at build time. This must be at least the same as the size of the

event log generated by TF-A. If this build option is not defined, the fTPM falls back to a default value of 1024 bytes, which is enough for this PoC, so this variable is not defined in FTPM_FLAGS.

10.3 Measured Boot using a Discrete TPM (PoC)

Measured Boot is the process of cryptographically measuring the code and critical data used at boot time, for example using a TPM, so that the security state can be attested later.

The current implementation of the driver included in *TF-A* supports several backends and each has a different means to store the measurements. This section focuses on the Discrete TPM backend, which stores measurements in a PCR within the TPM. This backend can be paired with the TCG event log to provide attestation of the measurements stored in the event log. See details in *Measured Boot Design*.

This section provides instructions to setup and build a proof of concept (PoC) that showcases the use of Measured Boot with a Discrete TPM interface.

Note: The instructions given in this document are meant to build a PoC to show how Measured Boot on TF-A can interact with a Discrete TPM interface. This PoC is platform specific, and uses a SPI based Discrete TPM, the Raspberry Pi communicates with the TPM via a GPIO pin bit-banged SPI interface. For other platforms, different may be required to interface with the hardware (e.g., different hardware communication protocols) and different TPM interfaces (e.g., *FIFO* vs *CRB*).

10.3.1 Components

- **Platform**: The PoC is developed on the Raspberry Pi 3 (rpi3), due to quick driver development and the availability of GPIO pins to interface with a TPM expansion module. Measured boot capabilities using the TCG Event Log are ported to the Raspberry Pi 3 platform inside TF-A. This PoC specifically uses the Raspberry Pi 3 Model B V1.2, but this PoC is compatible with other Raspberry Pi 3 models.
- **Discrete TPM**: The TPM chip selected is a breakout board compatible with the Raspberry Pi 3 GPIO pins. This PoC uses a *SPI* based LetsTrust TPM breakout board equipped with a Infineon Optiga™ SLB 9670 TPM 2.0 chip. Link to device: https://thepihut.com/products/letstrust-tpm-for-raspberry-pi

Note: If you have another TPM breakout board that uses the same Infineon Optiga™ SLB 9670 TPM 2.0 SPI based chip, it will also work. Ensure that the correct GPIO pins are utilized on the Raspberry Pi 3 to avoid communication issues, and possible hardware failures.

- **TF-A TPM Drivers**: To interface with a physical (Discrete) TPM chip in TF-A, the PoC uses TF-A drivers that provide the command, interface, link, and platform layers required to send and receive data to and from the TPM. The drivers are located in TFA, and not in a *SP*, so that they may be used in early stages such as BL2, and in some cases, BL1. The design of the TPM Drivers is documented here: *Discrete TPM drivers*.

- **U-boot BL33**: This PoC showcases measured boot up to BL33, and for simplicity uses a U-boot image for BL33, so that the image is measured and loaded. Currently U-boot does not have Discrete TPM support for the Raspberry Pi 3 platform so the boot flow ends here.

10.3.2 Building the PoC for the Raspberry Pi 3

Build instructions for U-Boot.bin for Raspberry Pi 3.

First, the build requires a BL33 firmware image that can be packaged and measured by TF-A.

U-boot can be built for the Raspberry Pi 3, but there are some changes to be made to allow the build to succeed. First Clone U-boot and enter the repo.

```
git clone https://github.com/u-boot/u-boot.git
cd u-boot
```

Now to switch to a specific tag `v2024.04` for testing purposes, and then build the defconfig labelled `rpi_3_b_plus_defconfig`.

```
git checkout tags/v2024.04 -b tfa_dtpm_poc
make CROSS_COMPILE=aarch64-linux-gnu- rpi_3_b_plus_defconfig
```

Lastly open the `.config` and change `CONFIG_TEXT_BASE` and `CONFIG_SYS_UBOOT_START` to `0x11000000` to match the BL33 starting point.

```
vim .config
CONFIG_TEXT_BASE=0x11000000
CONFIG_SYS_UBOOT_START=0x11000000
```

To build the u-boot binary, use the following command.

```
make CROSS_COMPILE=aarch64-linux-gnu- -j$(nproc)
```

Build TF-A for Raspberry Pi 3 with Discrete TPM and Measured Boot.

Copy over the `u-boot.bin` file over to your TF-A working directory.

```
cp /path/to/u-boot/build/u-boot.bin /path/to/tfa/u-boot.bin
```

TF-A build command:

```
CROSS_COMPILE=aarch64-linux-gnu-      \
make PLAT=rpi3                        \
RPI3_PRELOADED_DTB_BASE=0x200000      \
BL33=u-boot.bin                       \
SUPPORT_VFP=1                         \
DEBUG=0                               \
MEASURED_BOOT=1                       \
```

(continues on next page)

(continued from previous page)

```
DISCRETE_TPM=1                         \
MBOOT_TPM_HASH_ALG=sha256              \
TPM_INTERFACE=FIFO_SPI                 \
LOG_LEVEL=40                           \
fip all
```

This build command is similar to the one provided in the TF-A Raspberry Pi 3 platform port, To learn more about the platform and its build options, visit *Raspberry Pi 3*.

- `RPI3_PRELOADED_DTB_BASE` is given a different address to accommodate the larger BL1 and BL2 firmware sizes, this is to accommodate the TPM drivers that are packaged in BL1 and BL2 for this PoC.
- `BL33` is the non trusted firmware, in this case the U-Boot binary built earlier.
- `SUPPORT_VFP` is enabled, allows Vector Floating Point operations in EL3.
- `MEASURED_BOOT` is enabled to allow the Measured Boot flow.
- `DISCRETE_TPM=1` enables the build of Discrete TPM drivers.
- `MBOOT_TPM_HASH_ALG=sha256` sets the hash algorithm to sha256, this is the only algorithm supported by both TF-A Measured Boot and the SLB 9670 TPM 2.0.
- `TPM_INTERFACE=FIFO_SPI` specifies the use of the FIFO SPI interface.
- `LOG_LEVEL=40` ensures that eventlog is printed at the end of BL1 and BL2.

Hardware Setup:

- **TPM Connection**: Connect the LetsTrust TPM board to GPIO pins 17 - 26 on the 40-pin GPIO header on the Raspberry Pi board. The 2x5 header of the TPM module must be aligned to the pins in a specific orientation, match the 3v3 and RST pins from the TPM board to pins 17 and 18 respectively on the Raspberry Pi 3 header. See rpi3 pinout.
- **Serial Console**: Establish a serial connection to the Raspberry Pi 3 to view serial output during the boot sequence. The GND, TXD, and RXD pins, which are labelled 6, 8, and 10 on the Raspberry Pi 3 header respectively, are the required pins to establish a serial connection. The recommended way to connect to the board from another system is to use a USB to serial TTL cable to output the serial console in a easy manner.
- **SD Card Setup**: Format a SD Card as `FAT32` with a default Raspbian installation that is similar to the default Raspberry Pi 3 boot partition, this partition will utilize the default files installed in the root directory with Rasbian such as:

  ```
  bcm2710-rpi3-b.dtb
  bootcode.bin
  config.txt
  fixup.dat
  start.elf
  ```

  Open `config.txt` and overwrite the file with the following lines:

```
arm_64bit=1
disable_commandline_tags=2
enable_uart=1
armstub=armstub8.bin
device_tree_address=0x200000
device_tree_end=0x210000
```

These configurations are required to enable uart, enable 64bit mode, use the build TF binary, and the modified rpi3 device tree address and size.

Copy `armstub8.bin` from the TF-A build path to the root folder of the SD card.

The SD Card is now ready to be booted.

10.3.3 Running the PoC for the Raspberry Pi 3

Insert the SD Card into the Raspberry Pi 3 SD card port and boot the system.

To access the serial console output from the Raspberry Pi 3 you can either:

- Follow instructions to use PuTTY to connect to Raspberry Pi 3 serial console.
- Use the linux `screen` command:

```
screen /dev/ttyUSB0 115200
```

Once booted the output from the serial console will look like this:

```
Raspberry Pi Bootcode

Read File: config.txt, 153

Read File: start.elf, 2975040 (bytes)

Read File: fixup.dat, 7265 (bytes)

MESS:00:00:01.170422:0: brfs: File read: /mfs/sd/config.txt
MESS:00:00:01.174630:0: brfs: File read: 153 bytes
MESS:00:00:01.211473:0: HDMI0:EDID error reading EDID block 0 attempt 0
MESS:00:00:01.217639:0: HDMI0:EDID error reading EDID block 0 attempt 1
MESS:00:00:01.223977:0: HDMI0:EDID error reading EDID block 0 attempt 2
MESS:00:00:01.230313:0: HDMI0:EDID error reading EDID block 0 attempt 3
MESS:00:00:01.236650:0: HDMI0:EDID error reading EDID block 0 attempt 4
MESS:00:00:01.242987:0: HDMI0:EDID error reading EDID block 0 attempt 5
MESS:00:00:01.249324:0: HDMI0:EDID error reading EDID block 0 attempt 6
MESS:00:00:01.255660:0: HDMI0:EDID error reading EDID block 0 attempt 7
MESS:00:00:01.261997:0: HDMI0:EDID error reading EDID block 0 attempt 8
MESS:00:00:01.268334:0: HDMI0:EDID error reading EDID block 0 attempt 9
MESS:00:00:01.274429:0: HDMI0:EDID giving up on reading EDID block 0
MESS:00:00:01.282647:0: brfs: File read: /mfs/sd/config.txt
MESS:00:00:01.286929:0: gpioman: gpioman_get_pin_num: pin LEDS_PWR_OK not␣
↪defined
```

(continues on next page)

(continued from previous page)

```
MESS:00:00:01.487295:0: gpioman: gpioman_get_pin_num: pin DISPLAY_DSI_PORT␣
↪not defined
MESS:00:00:01.494853:0: gpioman: gpioman_get_pin_num: pin LEDS_PWR_OK not␣
↪defined
MESS:00:00:01.500763:0: *** Restart logging
MESS:00:00:01.504638:0: brfs: File read: 153 bytes
MESS:00:00:01.510139:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 0
MESS:00:00:01.517254:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 1
MESS:00:00:01.524112:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 2
MESS:00:00:01.530970:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 3
MESS:00:00:01.537826:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 4
MESS:00:00:01.544685:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 5
MESS:00:00:01.551543:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 6
MESS:00:00:01.558399:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 7
MESS:00:00:01.565258:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 8
MESS:00:00:01.572116:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 9
MESS:00:00:01.578730:0: hdmi: HDMI0:EDID giving up on reading EDID block 0
MESS:00:00:01.584634:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 0
MESS:00:00:01.592427:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 1
MESS:00:00:01.599286:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 2
MESS:00:00:01.606142:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 3
MESS:00:00:01.613001:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 4
MESS:00:00:01.619858:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 5
MESS:00:00:01.626717:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 6
MESS:00:00:01.633575:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 7
MESS:00:00:01.640431:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 8
MESS:00:00:01.647288:0: hdmi: HDMI0:EDID error reading EDID block 0 attempt 9
MESS:00:00:01.653905:0: hdmi: HDMI0:EDID giving up on reading EDID block 0
MESS:00:00:01.659769:0: hdmi: HDMI:hdmi_get_state is deprecated, use hdmi_get_
↪display_state instead
MESS:00:00:01.668264:0: HDMI0: hdmi_pixel_encoding: 162000000
MESS:00:00:01.673988:0: vec: vec_middleware_power_on: vec_base: 0x7e806000␣
↪rev-id 0x00002708 @ vec: 0x7e806100 @ 0x00000420 enc: 0x7e806060 @␣
↪0x00000220 cgmsae: 0x7e80605c @ 0x00000000
MESS:00:00:01.880234:0: dtb_file 'bcm2710-rpi-3-b.dtb'
MESS:00:00:01.889713:0: brfs: File read: /mfs/sd/bcm2710-rpi-3-b.dtb
MESS:00:00:01.894375:0: Loaded 'bcm2710-rpi-3-b.dtb' to 0x200000 size 0x7cb2
MESS:00:00:01.915761:0: brfs: File read: 31922 bytes
MESS:00:00:02.007202:0: brfs: File read: /mfs/sd/config.txt
MESS:00:00:02.017277:0: brfs: File read: 153 bytes
MESS:00:00:02.020772:0: Failed to open command line file 'cmdline.txt'
MESS:00:00:02.042302:0: gpioman: gpioman_get_pin_num: pin EMMC_ENABLE not␣
↪defined
MESS:00:00:02.398066:0: kernel=
MESS:00:00:02.455255:0: brfs: File read: /mfs/sd/armstub8.bin
MESS:00:00:02.459284:0: Loaded 'armstub8.bin' to 0x0 size 0xdbe74
MESS:00:00:02.465109:0: No compatible kernel found
MESS:00:00:02.469610:0: Device tree loaded to 0x200000 (size 0x823f)
MESS:00:00:02.476805:0: uart: Set PL011 baud rate to 103448.300000 Hz
MESS:00:00:02.483381:0: uart: Baud rate change done...
MESS:00:00:02.486793:0: uart: Baud rateNOTICE:  Booting Trusted Firmware
NOTICE:  BL1: v2.11.0(release):v2.11.0-187-g0cb1ddc9c-dirty
```

(continues on next page)

(continued from previous page)

```
NOTICE:  BL1: Built : 10:57:10, Jul  9 2024
INFO:    BL1: RAM 0x100ee000 - 0x100f9000
INFO:    Using crypto library 'mbed TLS'
NOTICE:  TPM Chip: vendor-id 0xd1, device-id 0x0, revision-id: 0x16
NOTICE:  rpi3: Detected: Raspberry Pi 3 Model B (1GB, Sony, UK) [0x00a02082]
INFO:    BL1: Loading BL2
INFO:    Loading image id=1 at address 0x100b4000
INFO:    Image id=1 loaded: 0x100b4000 - 0x100c0281
INFO:    TCG_EfiSpecIDEvent:
INFO:      PCRIndex             : 0
INFO:      EventType            : 3
INFO:      Digest               : 00
INFO:          : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
INFO:          : 00 00 00
INFO:      EventSize            : 33
INFO:      Signature            : Spec ID Event03
INFO:      PlatformClass        : 0
INFO:      SpecVersion          : 2.0.2
INFO:      UintnSize            : 1
INFO:      NumberOfAlgorithms : 1
INFO:      DigestSizes          :
INFO:        #0 AlgorithmId    : SHA256
INFO:           DigestSize     : 32
INFO:      VendorInfoSize       : 0
INFO:    PCR_Event2:
INFO:      PCRIndex             : 0
INFO:      EventType            : 3
INFO:      Digests Count        : 1
INFO:        #0 AlgorithmId    : SHA256
INFO:           Digest          : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00
INFO:          : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
INFO:      EventSize            : 17
INFO:      Signature            : StartupLocality
INFO:      StartupLocality      : 0
INFO:    PCR_Event2:
INFO:      PCRIndex             : 0
INFO:      EventType            : 1
INFO:      Digests Count        : 1
INFO:        #0 AlgorithmId    : SHA256
INFO:           Digest          : 55 11 51 d8 8b 7f 41 d3 18 16 f2 e8 80 bf 80␣
↪fa
INFO:          : b4 03 6d 96 4c a0 0a 98 45 cf 25 2f 1e a9 09 3e
INFO:      EventSize            : 5
INFO:      Event                : BL_2
NOTICE:  BL1: Booting BL2
INFO:    Entry point address = 0x100b4000
INFO:    SPSR = 0x3c5
NOTICE:  BL2: v2.11.0(release):v2.11.0-187-g0cb1ddc9c-dirty
NOTICE:  BL2: Built : 10:56:39, Jul  9 2024
INFO:    Using crypto library 'mbed TLS'
NOTICE:  TPM Chip: vendor-id 0xd1, device-id 0x0, revision-id: 0x16
```

(continues on next page)

(continued from previous page)

```
INFO:    BL2: Doing platform setup
INFO:    BL2: Loading image id 3
INFO:    Loading image id=3 at address 0x100e0000
INFO:    Image id=3 loaded: 0x100e0000 - 0x100e706b
INFO:    BL2: Loading image id 5
INFO:    Loading image id=5 at address 0x11000000
INFO:    Image id=5 loaded: 0x11000000 - 0x110a8ad8
INFO:    TCG_EfiSpecIDEvent:
INFO:      PCRIndex           : 0
INFO:      EventType          : 3
INFO:      Digest             : 00
INFO:          : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
INFO:          : 00 00 00
INFO:      EventSize          : 33
INFO:      Signature          : Spec ID Event03
INFO:      PlatformClass      : 0
INFO:      SpecVersion        : 2.0.2
INFO:      UintnSize          : 1
INFO:      NumberOfAlgorithms : 1
INFO:      DigestSizes        :
INFO:        #0 AlgorithmId   : SHA256
INFO:           DigestSize    : 32
INFO:      VendorInfoSize     : 0
INFO:    PCR_Event2:
INFO:      PCRIndex           : 0
INFO:      EventType          : 3
INFO:      Digests Count      : 1
INFO:        #0 AlgorithmId   : SHA256
INFO:           Digest        : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00␣
↪00
INFO:          : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
INFO:      EventSize          : 17
INFO:      Signature          : StartupLocality
INFO:      StartupLocality    : 0
INFO:    PCR_Event2:
INFO:      PCRIndex           : 0
INFO:      EventType          : 1
INFO:      Digests Count      : 1
INFO:        #0 AlgorithmId   : SHA256
INFO:           Digest        : 55 11 51 d8 8b 7f 41 d3 18 16 f2 e8 80 bf 80␣
↪fa
INFO:          : b4 03 6d 96 4c a0 0a 98 45 cf 25 2f 1e a9 09 3e
INFO:      EventSize          : 5
INFO:      Event              : BL_2
INFO:    PCR_Event2:
INFO:      PCRIndex           : 0
INFO:      EventType          : 1
INFO:      Digests Count      : 1
INFO:        #0 AlgorithmId   : SHA256
INFO:           Digest        : f3 00 5c ed a2 12 8b 76 b7 82 da c5 28 c3 02␣
↪52
INFO:          : 19 e4 3a 82 f2 3c ab 1e 0d 78 84 9c b5 fe e2 4f
```

(continues on next page)

(continued from previous page)

```
INFO:      EventSize          : 14
INFO:      Event              : SECURE_RT_EL3
INFO:    PCR_Event2:
INFO:      PCRIndex           : 0
INFO:      EventType          : 1
INFO:      Digests Count      : 1
INFO:        #0 AlgorithmId   : SHA256
INFO:          Digest         : 90 28 81 42 12 b7 9b ca aa 0c 40 76 33 5a 69␣
↪71
INFO:          : b6 19 2b 90 f2 d2 69 b8 de 8e 6d 05 4d c2 73 f9
INFO:      EventSize          : 6
INFO:      Event              : BL_33
NOTICE:  BL1: Booting BL31
INFO:    Entry point address = 0x100e0000
INFO:    SPSR = 0x3cd
NOTICE:  BL31: v2.11.0(release):v2.11.0-187-g0cb1ddc9c-dirty
NOTICE:  BL31: Built : 10:56:58, Jul  9 2024
INFO:    rpi3: Checking DTB...
INFO:    rpi3: Reserved 0x10000000 - 0x10100000 in DTB
INFO:    BL31: Initializing runtime services
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0x11000000
INFO:    SPSR = 0x3c9


U-Boot 2024.04-g84314330-dirty (Apr 23 2024 - 15:41:54 -0500)

DRAM:  948 MiB
RPI 3 Model B (0xa02082)
Core:  68 devices, 14 uclasses, devicetree: embed
MMC:   mmc@7e202000: 0, mmc@7e300000: 1
Loading Environment from FAT... OK
In:    serial,usbkbd
Out:   serial,vidconsole
Err:   serial,vidconsole
Net:   No ethernet found.
starting USB...
Bus usb@7e980000: USB DWC2
scanning bus usb@7e980000 for devices...
Error: smsc95xx_eth No valid MAC address found.
2 USB Device(s) found
      scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot:  2  1  0
Card did not respond to voltage select! : -110
No EFI system partition
No EFI system partition
Failed to persist EFI variables
No EFI system partition
Failed to persist EFI variables
No EFI system partition
Failed to persist EFI variables
Missing TPMv2 device for EFI_TCG_PROTOCOL
```

(continues on next page)

(continued from previous page)

```
** Booting bootflow '<NULL>' with efi_mgr
Loading Boot0000 'mmc 0' failed
EFI boot manager: Cannot load any image
Boot failed (err=-14)
Card did not respond to voltage select! : -110
No ethernet found.
No ethernet found.
U-Boot>
```

10.3.4 Next steps for Discrete TPM and Measured Boot development

In order to automatically validate the workings of the Discrete TPM, the creation of test cases that compare the eventlog image hashes with what is stored in PCR0 are a great way to test the core functionality of the Discrete TPM in Measured Boot.

Development of Discrete TPM drivers such as a reference FIFO *I2C*, MMIO, and CRB drivers has not started, these drivers will allow a larger number of platform to use a Discrete TPM in TF-A.

10.4 DRTM Proof of Concept

Dynamic Root of Trust for Measurement (DRTM) begins a new trust environment by measuring and executing a protected payload. The architectural requirements and formal definition of DRTM for Arm-based systems are detailed in the DRTM Architecture for Arm

Static Root of Trust for Measurement (SRTM)/Measured Boot implementation, currently used by TF-A covers all firmwares, from the boot ROM to the normal world bootloader. As a whole, they make up the system's TCB. These boot measurements allow attesting to what software is running on the system and enable enforcing security policies.

As the boot chain grows or firmware becomes dynamically extensible, establishing an attestable TCB becomes more challenging. DRTM provides a solution to this problem by allowing measurement chains to be started at any time. As these measurements are stored separately from the boot-time measurements, they reduce the size of the TCB, which helps reduce the attack surface and the risk of untrusted code executing, which could compromise the security of the system.

10.4.1 Components

- **DCE-Preamble**: The DCE Preamble prepares the platform for DRTM by doing any needed configuration, loading the target payload image(DLME), and preparing input parameters needed by DRTM. Finally, it invokes the DL Event to start the dynamic launch.
- **D-CRTM**: The D-CRTM is the trust anchor (or root of trust) for the DRTM boot sequence and is where the dynamic launch starts. The D-CRTM must be implemented as a trusted agent in the system. The D-CRTM initializes the TPM for DRTM and prepares the environment for the next stage of DRTM, the DCE. The D-CRTM measures the DCE, verifies its signature, and transfers control to it.

- **DCE**: The DCE executes on an application core. The DCE verifies the system's state, measures security-critical attributes of the system, prepares the memory region for the target payload, measures the payload, and finally transfers control to the payload.
- **DLME**: The protected payload is referred to as the Dynamically Launched Measured Environment, or DLME. The DLME begins execution in a safe state, with a single thread of execution, DMA protections, and interrupts disabled. The DCE provides data to the DLME that it can use to verify the configuration of the system.

In this proof of concept, DCE and D-CRTM are implemented in BL31 and DCE-Preamble and DLME are implemented in UEFI application. A DL Event is triggered as a SMC by DCE-Preamble and handled by D-CRTM, which launches the DLME via DCE.

TF-A CI pipeline already includes coverage for building TF-A with a prebuilt EDK2 and the DRTM UEFI application.

10.5 Runtime Security Engine (RSE)

This document focuses on the relationship between the Runtime Security Engine (RSE) and the application processor (AP). According to the ARM reference design the RSE is an independent core next to the AP and the SCP on the same die. It provides fundamental security guarantees and runtime services for the rest of the system (e.g.: trusted boot, measured boot, platform attestation, key management, and key derivation).

At power up RSE boots first from its private ROM code. It validates and loads its own images and the initial images of SCP and AP. When AP and SCP are released from reset and their initial code is loaded then they continue their own boot process, which is the same as on non-RSE systems. Please refer to the `RSE documentation`[1] for more details about the RSE boot flow.

The last stage of the RSE firmware is a persistent, runtime component. Much like AP_BL31, this is a passive entity which has no periodical task to do and just waits for external requests from other subsystems. RSE and other subsystems can communicate with each other over message exchange. RSE waits in idle for the incoming request, handles them, and sends a response then goes back to idle.

10.5.1 RSE communication layer

The communication between RSE and other subsystems are primarily relying on the Message Handling Unit (MHU) module.

However, this is possible to use this communication protocol with a different mailbox than MHU, by setting the flag `PLAT_MHU=NO_MHU` and implementing the APIs given in the file: `include/drivers/arm/rse_comms.h`.

The number of MHU interfaces between RSE and other cores is IMPDEF. Besides MHU other modules also could take part in the communication. RSE is capable of mapping the AP memory to its address space. Thereby

[1] https://trustedfirmware-m.readthedocs.io/en/latest/platform/arm/rse/index.html

either RSE core itself or a DMA engine if it is present, can move the data between memory belonging to RSE or AP. In this way, a bigger amount of data can be transferred in a short time.

The MHU comes in pairs. There is a sender and receiver side. They are connected to each other. An MHU interface consists of two pairs of MHUs, one sender and one receiver on both sides. Bidirectional communication is possible over an interface. One pair provides message sending from AP to RSE and the other pair from RSE to AP. The sender and receiver are connected via channels. There is an IMPDEF number of channels (e.g: 4-16) between a sender and a receiver module.

The RSE communication layer provides two ways for message exchange:

- `Embedded messaging`: The full message, including header and payload, are exchanged over the MHU channels. A channel is capable of delivering a single word. The sender writes the data to the channel register on its side and the receiver can read the data from the channel on the other side. One dedicated channel is used for signalling. It does not deliver any payload it is just meant for signalling that the sender loaded the data to the channel registers so the receiver can read them. The receiver uses the same channel to signal that data was read. Signalling happens via IRQ. If the message is longer than the data fit to the channel registers then the message is sent over in multiple rounds. Both, sender and receiver allocate a local buffer for the messages. Data is copied from/to these buffers to/from the channel registers.
- `Pointer-access messaging`: The message header and the payload are separated and they are conveyed in different ways. The header is sent over the channels, similar to the embedded messaging but the payload is copied over by RSE core (or by DMA) between the sender and the receiver. This could be useful in the case of long messages because transaction time is less compared to the embedded messaging mode. Small payloads are copied by the RSE core because setting up DMA would require more CPU cycles. The payload is either copied into an internal buffer or directly read-written by RSE. Actual behavior depends on RSE setup, whether the partition supports memory-mapped `iovec`. Therefore, the sender must handle both cases and prevent access to the memory, where payload data lives, while the RSE handles the request.

The RSE communication layer supports both ways of messaging in parallel. It is decided at runtime based on the message size which way to transfer the message.

```
+----------------------------------------------+       +-------------------+
|                                              |       |                   |
|                      AP                      |       |                   |
|                                              |  +--->|       SRAM        |
+----------------------------------------------|  |    |                   |
|              BL1 / BL2 / BL31                |  |    |                   |
+----------------------------------------------+  |    +-------------------+
         |                           ^            |        ^           ^
         |  send                 IRQ | receive    |direct  |           |
         V                           |            |access  |           |
+--------------------+    +--------------------+  |        |           |
|      MHU sender    |    |    MHU receiver    |  |        | Copy data |
+--------------------+    +--------------------+  |        |           |
   | |           | |          | |           | |   |        |           |
   | | channels  | |          | | channels  | |   |        |           |
   | | e.g: 4-16 | |          | | e.g: 4-16 | |   |        V           |
+--------------------+    +--------------------+  |    +-------+       |
|     MHU receiver   |    |     MHU sender     |  | +->|  DMA  |       |
```

(continues on next page)

(continued from previous page)

```
+--------------------+    +--------------------+  | |  +--------+       |
         |                           ^             | |     ^            |
    IRQ  |  receive                  | send        | |     | Copy data |
         V                           |             | |     V            V
+----------------------------------------------+  | |  +--------------------+
|                                              |--+-+  |                    |
|                  RSE                         |       |      SRAM          |
|                                              |       |                    |
+----------------------------------------------+       +--------------------+
```

Note: The RSE communication layer is not prepared for concurrent execution. The current use case only requires message exchange during the boot phase. In the boot phase, only a single core is running and the rest of the cores are in reset.

Message structure

A description of the message format can be found in the `RSE communication design`[2] document.

Source files

- RSE comms: `drivers/arm/rse`
- MHU driver: `drivers/arm/mhu`

API for communication over MHU

The API is defined in these header files:

- `include/drivers/arm/rse_comms.h`
- `include/drivers/arm/mhu.h`

10.5.2 RSE provided runtime services

RSE provides the following runtime services:

- `Measured boot`: Securely store the firmware measurements which were computed during the boot process and the associated metadata (image description, measurement algorithm, etc.). More info on measured boot service in RSE can be found in the `measured_boot_integration_guide`[3] .

[2] https://trustedfirmware-m.readthedocs.io/en/latest/platform/arm/rse/rse_comms.html

[3] https://trustedfirmware-m.readthedocs.io/projects/tf-m-extras/en/latest/partitions/measured_boot_integration_guide.html

- `Delegated attestation`: Query the platform attestation token and derive a delegated attestation key. More info on the delegated attestation service in RSE can be found in the `delegated_attestation_integration_guide`[4].
- `OTP assets management`: Public keys used by AP during the trusted boot process can be requested from RSE. Furthermore, AP can request RSE to increase a non-volatile counter. Please refer to the `RSE key management`[5] document for more details.
- `DICE Protection Environment`: Securely store the firmware measurements which were computed during the boot process and the associated metadata. It is also capable of representing the boot measurements in the form of a certificate chain, which is queriable. Please refer to the `DICE Protection Environment (DPE)`[8] document for more details.

Runtime service API

The RSE provided runtime services implement a PSA aligned API. The parameter encoding follows the PSA client protocol described in the `Firmware Framework for M`[6] document in chapter 4.4. The implementation is restricted to the static handle use case therefore only the `psa_call` API is implemented.

Software and API layers

```
+----------------+         +---------------------+
|   BL1 / BL2    |         |       BL31          |
+----------------+         +---------------------+
  |                          |
  | extend_measurement()     | get_delegated_key()
  |                          | get_platform_token()
  V                          V
+----------------+         +---------------------+
|  PSA protocol  |         |    PSA protocol     |
+----------------+         +---------------------+
     |                               |
     | psa_call()                    | psa_call()
     |                               |
     V                               V
+------------------------------------------------+
|         RSE communication protocol             |
+------------------------------------------------+
     |                     ^
     | mhu_send_data()     | mhu_receive_data()
     |                     |
     V                     |
+------------------------------------------------+
```

(continues on next page)

[4] https://trustedfirmware-m.readthedocs.io/projects/tf-m-extras/en/latest/partitions/delegated_attestation/delegated_attest_integration_guide.html

[5] https://trustedfirmware-m.readthedocs.io/en/latest/platform/arm/rse/rse_key_management.html

[8] https://trustedfirmware-m.readthedocs.io/projects/tf-m-extras/en/latest/partitions/dice_protection_environment/dice_protection_environment.html

[6] https://developer.arm.com/documentation/den0063

(continued from previous page)

```
|                 MHU driver                       |
+--------------------------------------------------+
           |                        ^
           | Register access        | IRQ
           V                        |
+--------------------------------------------------+
|             MHU HW on AP side                    |
+--------------------------------------------------+
                     ^
                     | Physical wires
                     |
                     V
+--------------------------------------------------+
|             MHU HW on RSE side                   |
+--------------------------------------------------+
         |                          ^
         | IRQ                      | Register access
         V                          |
+--------------------------------------------------+
|                 MHU driver                       |
+--------------------------------------------------+
         |                          |
         V                          V
+---------------+       +------------------------+
| Measured boot |       | Delegated attestation  |
| service       |       | service                |
+---------------+       +------------------------+
```

10.5.3 RSE based Measured Boot

Measured Boot is the process of cryptographically measuring (computing the hash value of a binary) the code and critical data used at boot time. The measurement must be stored in a tamper-resistant way, so the security state of the device can be attested later to an external party. RSE provides a runtime service which is meant to store measurements and associated metadata alongside.

Data is stored in internal SRAM which is only accessible by the secure runtime firmware of RSE. Data is stored in so-called measurement slots. A platform has IMPDEF number of measurement slots. The measurement storage follows extend semantics. This means that measurements are not stored directly (as it was taken) instead they contribute to the current value of the measurement slot. The extension implements this logic, where `||` stands for concatenation:

```
new_value_of_measurement_slot = Hash(old_value_of_measurement_slot ||␣
↪measurement)
```

Supported hash algorithms: sha-256, sha-512

Measured Boot API

Defined here:

- include/lib/psa/measured_boot.h

```
psa_status_t
rse_measured_boot_extend_measurement(uint8_t        index,
                                     const uint8_t *signer_id,
                                     size_t         signer_id_size,
                                     const uint8_t *version,
                                     size_t         version_size,
                                     uint32_t       measurement_algo,
                                     const uint8_t *sw_type,
                                     size_t         sw_type_size,
                                     const uint8_t *measurement_value,
                                     size_t         measurement_value_size,
                                     bool           lock_measurement);
```

Measured Boot Metadata

The following metadata can be stored alongside the measurement:

- `Signer-id`: Mandatory. The hash of the firmware image signing public key.
- `Measurement algorithm`: Optional. The hash algorithm which was used to compute the measurement (e.g.: sha-256, etc.).
- `Version info`: Optional. The firmware version info (e.g.: 2.7).
- `SW type`: Optional. Short text description (e.g.: BL1, BL2, BL31, etc.)

Note: Version info is not implemented in TF-A yet.

The caller must specify in which measurement slot to extend a certain measurement and metadata. A measurement slot can be extended by multiple measurements. The default value is IMPDEF. All measurement slot is cleared at reset, there is no other way to clear them. In the reference implementation, the measurement slots are initialized to 0. At the first call to extend the measurement in a slot, the extend operation uses the default value of the measurement slot. All upcoming extend operation on the same slot contributes to the previous value of that measurement slot.

The following rules are kept when a slot is extended multiple times:

- `Signer-id` must be the same as the previous call(s), otherwise a PSA_ERROR_NOT_PERMITTED error code is returned.
- `Measurement algorithm`: must be the same as the previous call(s), otherwise, a PSA_ERROR_NOT_PERMITTED error code is returned.

In case of error no further action is taken (slot is not locked). If there is a valid data in a sub-sequent call then measurement slot will be extended. The rest of the metadata is handled as follows when a measurement slot is extended multiple times:

- `SW type`: Cleared.
- `Version info`: Cleared.

Note: Extending multiple measurements in the same slot leads to some metadata information loss. Since RSE is not constrained on special HW resources to store the measurements and metadata, therefore it is worth considering to store all of them one by one in distinct slots. However, they are one-by-one included in the platform attestation token. So, the number of distinct firmware image measurements has an impact on the size of the attestation token.

The allocation of the measurement slot among RSE, Root and Realm worlds is platform dependent. The platform must provide an allocation of the measurement slot at build time. An example can be found in `tf-a/plat/arm/board/tc/tc_bl1_measured_boot.c` Furthermore, the memory, which holds the metadata is also statically allocated in RSE memory. Some of the fields have a static value (measurement algorithm), and some of the values have a dynamic value (measurement value) which is updated by the bootloaders when the firmware image is loaded and measured. The metadata structure is defined in `include/drivers/measured_boot/rse/rse_measured_boot.h`.

```
struct rse_mboot_metadata {
        unsigned int id;
        uint8_t slot;
        uint8_t signer_id[SIGNER_ID_MAX_SIZE];
        size_t  signer_id_size;
        uint8_t version[VERSION_MAX_SIZE];
        size_t  version_size;
        uint8_t sw_type[SW_TYPE_MAX_SIZE];
        size_t  sw_type_size;
        void    *pk_oid;
        bool    lock_measurement;
};
```

Signer-ID API

This function calculates the hash of a public key (signer-ID) using the `Measurement algorithm` and stores it in the `rse_mboot_metadata` field named `signer_id`. Prior to calling this function, the caller must ensure that the `signer_id` field points to the zero-filled buffer.

Defined here:

- `include/drivers/measured_boot/rse/rse_measured_boot.h`

```
int rse_mboot_set_signer_id(struct rse_mboot_metadata *metadata_ptr,
                            const void *pk_oid,
                            const void *pk_ptr,
                            size_t pk_len)
```

- First parameter is the pointer to the `rse_mboot_metadata` structure.
- Second parameter is the pointer to the key-OID of the public key.

- Third parameter is the pointer to the public key buffer.
- Fourth parameter is the size of public key buffer.
- This function returns 0 on success, a signed integer error code otherwise.

Build time config options

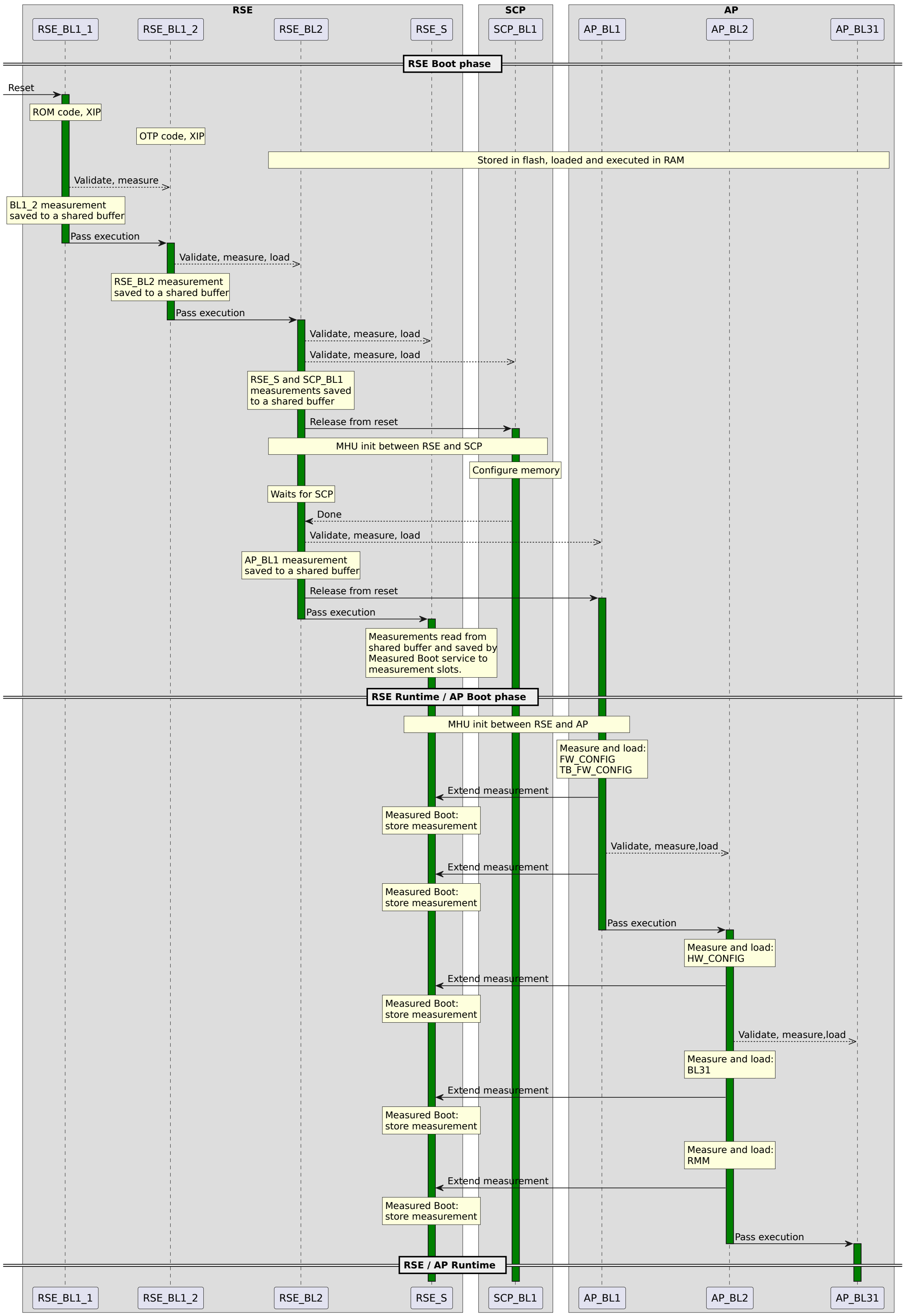
- `MEASURED_BOOT`: Enable measured boot.
- `MBOOT_RSE_HASH_ALG`: Determine the hash algorithm to measure the images. The default value is sha-256.

Measured boot flow

Sample console log

```
INFO:    Measured boot extend measurement:
INFO:     - slot        : 6
INFO:     - signer_id   : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
INFO:                   : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
INFO:     - version     :
INFO:     - version_size: 0
INFO:     - sw_type     : FW_CONFIG
INFO:     - sw_type_size: 10
INFO:     - algorithm   : 2000009
INFO:     - measurement : aa ea d3 a7 a8 e2 ab 7d 13 a6 cb 34 99 10 b9 a1
INFO:                   : 1b 9f a0 52 c5 a8 b1 d7 76 f2 c1 c1 ef ca 1a df
INFO:     - locking     : true
INFO:    FCONF: Config file with image ID:31 loaded at address = 0x4001010
INFO:    Loading image id=24 at address 0x4001300
INFO:    Image id=24 loaded: 0x4001300 - 0x400153a
INFO:    Measured boot extend measurement:
INFO:     - slot        : 7
INFO:     - signer_id   : b0 f3 82 09 12 97 d8 3a 37 7a 72 47 1b ec 32 73
INFO:                   : e9 92 32 e2 49 59 f6 5e 8b 4a 4a 46 d8 22 9a da
INFO:     - version     :
INFO:     - version_size: 0
INFO:     - sw_type     : TB_FW_CONFIG
INFO:     - sw_type_size: 13
INFO:     - algorithm   : 2000009
INFO:     - measurement : 05 b9 dc 98 62 26 a7 1c 2d e5 bb af f0 90 52 28
INFO:                   : f2 24 15 8a 3a 56 60 95 d6 51 3a 7a 1a 50 9b b7
INFO:     - locking     : true
INFO:    FCONF: Config file with image ID:24 loaded at address = 0x4001300
INFO:    BL1: Loading BL2
INFO:    Loading image id=1 at address 0x404d000
INFO:    Image id=1 loaded: 0x404d000 - 0x406412a
INFO:    Measured boot extend measurement:
INFO:     - slot        : 8
```

(continues on next page)


RSE
SCP
AP
RSE_BL1_1
RSE_BL1_2
RSE_BL2
RSE_S
SCP_BL1
AP_BL1
AP_BL2
AP_BL31
RSE Boot phase
Reset
ROM code, XIP
OTP code, XIP
Stored in flash, loaded and executed in RAM
Validate, measure
BL1_2 measurement saved to a shared buffer
Pass execution
Validate, measure, load
RSE_BL2 measurement saved to a shared buffer
Pass execution
Validate, measure, load
Validate, measure, load
RSE_S and SCP_BL1 measurements saved to a shared buffer
Release from reset
MHU init between RSE and SCP
Configure memory
Waits for SCP
Done
Validate, measure, load
AP_BL1 measurement saved to a shared buffer
Release from reset
Pass execution
Measurements read from shared buffer and saved by Measured Boot service to measurement slots.
RSE Runtime / AP Boot phase
MHU init between RSE and AP
Measure and load: FW_CONFIG TB_FW_CONFIG
Extend measurement
Measured Boot: store measurement
Validate, measure,load
Extend measurement
Measured Boot: store measurement
Pass execution
Measure and load: HW_CONFIG
Extend measurement
Measured Boot: store measurement
Validate, measure,load
Measure and load: BL31
Extend measurement
Measured Boot: store measurement
Measure and load: RMM
Extend measurement
Measured Boot: store measurement
Pass execution
RSE / AP Runtime
RSE_BL1_1
RSE_BL1_2
RSE_BL2
RSE_S
SCP_BL1
AP_BL1
AP_BL2
AP_BL31

(continued from previous page)

```
INFO:     - signer_id   : b0 f3 82 09 12 97 d8 3a 37 7a 72 47 1b ec 32 73
INFO:                   : e9 92 32 e2 49 59 f6 5e 8b 4a 4a 46 d8 22 9a da
INFO:     - version     :
INFO:     - version_size: 0
INFO:     - sw_type     : BL_2
INFO:     - sw_type_size: 5
INFO:     - algorithm   : 2000009
INFO:     - measurement : 53 a1 51 75 25 90 fb a1 d9 b8 c8 34 32 3a 01 16
INFO:                   : c9 9e 74 91 7d 28 02 56 3f 5c 40 94 37 58 50 68
INFO:     - locking     : true
```

10.5.4 Delegated Attestation

Delegated Attestation Service was mainly developed to support the attestation flow on the `ARM Confidential Compute Architecture` (ARM CCA)[7]. The detailed description of the delegated attestation service can be found in the `Delegated Attestation Service Integration Guide`[Page 807, 4] document.

In the CCA use case, the Realm Management Monitor (RMM) relies on the delegated attestation service of the RSE to get a realm attestation key and the CCA platform token. BL31 does not use the service for its own purpose, only calls it on behalf of RMM. The access to MHU interface and thereby to RSE is restricted to BL31 only. Therefore, RMM does not have direct access, all calls need to go through BL31. The RMM dispatcher module of the BL31 is responsible for delivering the calls between the two parties.

Delegated Attestation API

Defined here:

- `include/lib/psa/delegated_attestation.h`

```
psa_status_t
rse_delegated_attest_get_delegated_key(uint8_t   ecc_curve,
                                       uint32_t  key_bits,
                                       uint8_t  *key_buf,
                                       size_t    key_buf_size,
                                       size_t   *key_size,
                                       uint32_t  hash_algo);

psa_status_t
rse_delegated_attest_get_token(const uint8_t *dak_pub_hash,
                               size_t         dak_pub_hash_size,
                               uint8_t       *token_buf,
                               size_t         token_buf_size,
                               size_t        *token_size);
```

[7] https://developer.arm.com/documentation/DEN0096/A_a/?lang=en

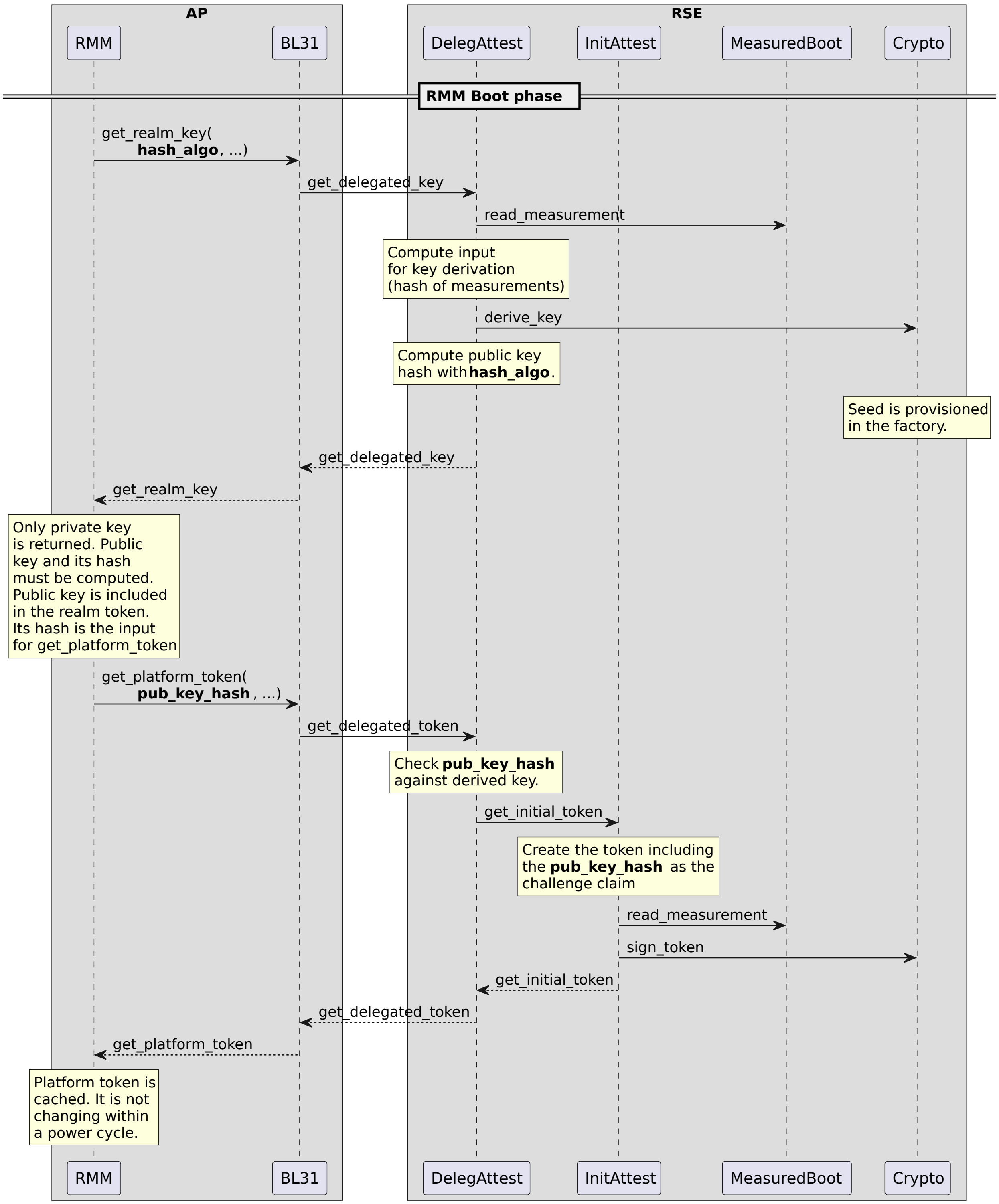
Attestation flow

Sample attestation token

Binary format:

```
INFO:    DELEGATED ATTEST TEST START
INFO:    Get delegated attestation key start
INFO:    Get delegated attest key succeeds, len: 48
INFO:    Delegated attest key:
INFO:            0d 2a 66 61 d4 89 17 e1 70 c6 73 56 df f4 11 fd
INFO:            7d 1f 3b 8a a3 30 3d 70 4c d9 06 c3 c7 ef 29 43
INFO:            0f ee b5 e7 56 e0 71 74 1b c4 39 39 fd 85 f6 7b
INFO:    Get platform token start
INFO:    Get platform token succeeds, len: 1086
INFO:    Platform attestation token:
INFO:            d2 84 44 a1 01 38 22 a0 59 05 81 a9 19 01 09 78
INFO:            23 74 61 67 3a 61 72 6d 2e 63 6f 6d 2c 32 30 32
INFO:            33 3a 63 63 61 5f 70 6c 61 74 66 6f 72 6d 23 31
INFO:            2e 30 2e 30 0a 58 20 0d 22 e0 8a 98 46 90 58 48
INFO:            63 18 28 34 89 bd b3 6f 09 db ef eb 18 64 df 43
INFO:            3f a6 e5 4e a2 d7 11 19 09 5c 58 20 7f 45 4c 46
INFO:            02 01 01 00 00 00 00 00 00 00 00 00 03 00 3e 00
INFO:            01 00 00 00 50 58 00 00 00 00 00 00 19 01 00 58
INFO:            21 01 07 06 05 04 03 02 01 00 0f 0e 0d 0c 0b 0a
INFO:            09 08 17 16 15 14 13 12 11 10 1f 1e 1d 1c 1b 1a
INFO:            19 18 19 09 61 44 cf cf cf cf 19 09 5b 19 30 03
INFO:            19 09 62 67 73 68 61 2d 32 35 36 19 09 60 78 3a
INFO:            68 74 74 70 73 3a 2f 2f 76 65 72 61 69 73 6f 6e
INFO:            2e 65 78 61 6d 70 6c 65 2f 2e 77 65 6c 6c 2d 6b
INFO:            6e 6f 77 6e 2f 76 65 72 61 69 73 6f 6e 2f 76 65
INFO:            72 69 66 69 63 61 74 69 6f 6e 19 09 5f 8d a4 01
INFO:            69 52 53 45 5f 42 4c 31 5f 32 05 58 20 53 78 79
INFO:            63 07 53 5d f3 ec 8d 8b 15 a2 e2 dc 56 41 41 9c
INFO:            3d 30 60 cf e3 22 38 c0 fa 97 3f 7a a3 02 58 20
INFO:            9a 27 1f 2a 91 6b 0b 6e e6 ce cb 24 26 f0 b3 20
INFO:            6e f0 74 57 8b e5 5d 9b c9 4f 6f 3f e3 ab 86 aa
INFO:            06 67 73 68 61 2d 32 35 36 a4 01 67 52 53 45 5f
INFO:            42 4c 32 05 58 20 53 78 79 63 07 53 5d f3 ec 8d
INFO:            8b 15 a2 e2 dc 56 41 41 9c 3d 30 60 cf e3 22 38
INFO:            c0 fa 97 3f 7a a3 02 58 20 53 c2 34 e5 e8 47 2b
INFO:            6a c5 1c 1a e1 ca b3 fe 06 fa d0 53 be b8 eb fd
INFO:            89 77 b0 10 65 5b fd d3 c3 06 67 73 68 61 2d 32
INFO:            35 36 a4 01 65 52 53 45 5f 53 05 58 20 53 78 79
INFO:            63 07 53 5d f3 ec 8d 8b 15 a2 e2 dc 56 41 41 9c
INFO:            3d 30 60 cf e3 22 38 c0 fa 97 3f 7a a3 02 58 20
INFO:            11 21 cf cc d5 91 3f 0a 63 fe c4 0a 6f fd 44 ea
INFO:            64 f9 dc 13 5c 66 63 4b a0 01 d1 0b cf 43 02 a2
INFO:            06 67 73 68 61 2d 32 35 36 a4 01 66 41 50 5f 42
INFO:            4c 31 05 58 20 53 78 79 63 07 53 5d f3 ec 8d 8b
INFO:            15 a2 e2 dc 56 41 41 9c 3d 30 60 cf e3 22 38 c0
INFO:            fa 97 3f 7a a3 02 58 20 15 71 b5 ec 78 bd 68 51
```

(continues on next page)


AP
RSE
RMM
BL31
DelegAttest
InitAttest
MeasuredBoot
Crypto
RMM Boot phase
get_realm_key(
hash_algo, ...)
get_delegated_key
read_measurement
Compute input
for key derivation
(hash of measurements)
derive_key
Compute public key
hash with hash_algo.
Seed is provisioned
in the factory.
get_delegated_key
get_realm_key
Only private key
is returned. Public
key and its hash
must be computed.
Public key is included
in the realm token.
Its hash is the input
for get_platform_token
get_platform_token(
pub_key_hash, ...)
get_delegated_token
Check pub_key_hash
against derived key.
get_initial_token
Create the token including
the pub_key_hash as the
challenge claim
read_measurement
sign_token
get_initial_token
get_delegated_token
get_platform_token
Platform token is
cached. It is not
changing within
a power cycle.
RMM
BL31
DelegAttest
InitAttest
MeasuredBoot
Crypto

(continued from previous page)

```
INFO:           2b f7 83 0b b6 a2 a4 4b 20 47 c7 df 57 bc e7 9e
INFO:           b8 a1 c0 e5 be a0 a5 01 06 67 73 68 61 2d 32 35
INFO:           36 a4 01 66 41 50 5f 42 4c 32 05 58 20 53 78 79
INFO:           63 07 53 5d f3 ec 8d 8b 15 a2 e2 dc 56 41 41 9c
INFO:           3d 30 60 cf e3 22 38 c0 fa 97 3f 7a a3 02 58 20
INFO:           10 15 9b af 26 2b 43 a9 2d 95 db 59 da e1 f7 2c
INFO:           64 51 27 30 16 61 e0 a3 ce 4e 38 b2 95 a9 7c 58
INFO:           06 67 73 68 61 2d 32 35 36 a4 01 67 53 43 50 5f
INFO:           42 4c 31 05 58 20 53 78 79 63 07 53 5d f3 ec 8d
INFO:           8b 15 a2 e2 dc 56 41 41 9c 3d 30 60 cf e3 22 38
INFO:           c0 fa 97 3f 7a a3 02 58 20 10 12 2e 85 6b 3f cd
INFO:           49 f0 63 63 63 17 47 61 49 cb 73 0a 1a a1 cf aa
INFO:           d8 18 55 2b 72 f5 6d 6f 68 06 67 73 68 61 2d 32
INFO:           35 36 a4 01 67 53 43 50 5f 42 4c 32 05 58 20 f1
INFO:           4b 49 87 90 4b cb 58 14 e4 45 9a 05 7e d4 d2 0f
INFO:           58 a6 33 15 22 88 a7 61 21 4d cd 28 78 0b 56 02
INFO:           58 20 aa 67 a1 69 b0 bb a2 17 aa 0a a8 8a 65 34
INFO:           69 20 c8 4c 42 44 7c 36 ba 5f 7e a6 5f 42 2c 1f
INFO:           e5 d8 06 67 73 68 61 2d 32 35 36 a4 01 67 41 50
INFO:           5f 42 4c 33 31 05 58 20 53 78 79 63 07 53 5d f3
INFO:           ec 8d 8b 15 a2 e2 dc 56 41 41 9c 3d 30 60 cf e3
INFO:           22 38 c0 fa 97 3f 7a a3 02 58 20 2e 6d 31 a5 98
INFO:           3a 91 25 1b fa e5 ae fa 1c 0a 19 d8 ba 3c f6 01
INFO:           d0 e8 a7 06 b4 cf a9 66 1a 6b 8a 06 67 73 68 61
INFO:           2d 32 35 36 a4 01 63 52 4d 4d 05 58 20 53 78 79
INFO:           63 07 53 5d f3 ec 8d 8b 15 a2 e2 dc 56 41 41 9c
INFO:           3d 30 60 cf e3 22 38 c0 fa 97 3f 7a a3 02 58 20
INFO:           a1 fb 50 e6 c8 6f ae 16 79 ef 33 51 29 6f d6 71
INFO:           34 11 a0 8c f8 dd 17 90 a4 fd 05 fa e8 68 81 64
INFO:           06 67 73 68 61 2d 32 35 36 a4 01 69 48 57 5f 43
INFO:           4f 4e 46 49 47 05 58 20 53 78 79 63 07 53 5d f3
INFO:           ec 8d 8b 15 a2 e2 dc 56 41 41 9c 3d 30 60 cf e3
INFO:           22 38 c0 fa 97 3f 7a a3 02 58 20 1a 25 24 02 97
INFO:           2f 60 57 fa 53 cc 17 2b 52 b9 ff ca 69 8e 18 31
INFO:           1f ac d0 f3 b0 6e ca ae f7 9e 17 06 67 73 68 61
INFO:           2d 32 35 36 a4 01 69 46 57 5f 43 4f 4e 46 49 47
INFO:           05 58 20 53 78 79 63 07 53 5d f3 ec 8d 8b 15 a2
INFO:           e2 dc 56 41 41 9c 3d 30 60 cf e3 22 38 c0 fa 97
INFO:           3f 7a a3 02 58 20 9a 92 ad bc 0c ee 38 ef 65 8c
INFO:           71 ce 1b 1b f8 c6 56 68 f1 66 bf b2 13 64 4c 89
INFO:           5c cb 1a d0 7a 25 06 67 73 68 61 2d 32 35 36 a4
INFO:           01 6c 54 42 5f 46 57 5f 43 4f 4e 46 49 47 05 58
INFO:           20 53 78 79 63 07 53 5d f3 ec 8d 8b 15 a2 e2 dc
INFO:           56 41 41 9c 3d 30 60 cf e3 22 38 c0 fa 97 3f 7a
INFO:           a3 02 58 20 23 89 03 18 0c c1 04 ec 2c 5d 8b 3f
INFO:           20 c5 bc 61 b3 89 ec 0a 96 7d f8 cc 20 8c dc 7c
INFO:           d4 54 17 4f 06 67 73 68 61 2d 32 35 36 a4 01 6d
INFO:           53 4f 43 5f 46 57 5f 43 4f 4e 46 49 47 05 58 20
INFO:           53 78 79 63 07 53 5d f3 ec 8d 8b 15 a2 e2 dc 56
INFO:           41 41 9c 3d 30 60 cf e3 22 38 c0 fa 97 3f 7a a3
INFO:           02 58 20 e6 c2 1e 8d 26 0f e7 18 82 de bd b3 39
INFO:           d2 40 2a 2c a7 64 85 29 bc 23 03 f4 86 49 bc e0
```

(continues on next page)

(continued from previous page)

```
INFO:           38 00 17 06 67 73 68 61 2d 32 35 36 58 60 31 d0
INFO:           4d 52 cc de 95 2c 1e 32 cb a1 81 88 5a 40 b8 cc
INFO:           38 e0 52 8c 1e 89 58 98 07 64 2a a5 e3 f2 bc 37
INFO:           f9 53 74 50 6b ff 4d 2e 4b e7 06 3c 4d 72 41 92
INFO:           70 c7 22 e8 d4 d9 3e e8 b6 c9 fa ce 3b 43 c9 76
INFO:           1a 49 94 1a b6 f3 8f fd ff 49 6a d4 63 b4 cb fa
INFO:           11 d8 3e 23 e3 1f 7f 62 32 9d e3 0c 1c c8
INFO:    DELEGATED ATTEST TEST END
```

JSON format:


```
{
    "CCA_ATTESTATION_PROFILE": "tag:arm.com,2023:cca_platform#1.0.0",
    "CCA_PLATFORM_CHALLENGE": "b
↪'0D22E08A98469058486318283489BDB36F09DBEFEB1864DF433FA6E54EA2D711'",
    "CCA_PLATFORM_IMPLEMENTATION_ID": "b
↪'7F454C4602010100000000000000000003003E00010000005058000000000000'",
    "CCA_PLATFORM_INSTANCE_ID": "b
↪'0107060504030201000F0E0D0C0B0A090817161514131211101F1E1D1C1B1A1918'",
    "CCA_PLATFORM_CONFIG": "b'CFCFCFCF'",
    "CCA_PLATFORM_LIFECYCLE": "secured_3003",
    "CCA_PLATFORM_HASH_ALGO_ID": "sha-256",
    "CCA_PLATFORM_VERIFICATION_SERVICE": "https://veraison.example/.well-
↪known/veraison/verification",
    "CCA_PLATFORM_SW_COMPONENTS": [
        {
            "SW_COMPONENT_TYPE": "RSE_BL1_2",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'9A271F2A916B0B6EE6CECB2426F0B3206EF074578BE55D9BC94F6F3FE3AB86AA'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "RSE_BL2",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'53C234E5E8472B6AC51C1AE1CAB3FE06FAD053BEB8EBFD8977B010655BFDD3C3'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "RSE_S",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'1121CFCCD5913F0A63FEC40A6FFD44EA64F9DC135C66634BA001D10BCF4302A2'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "AP_BL1",
            "SIGNER_ID": "b
```


(continues on next page)

(continued from previous page)


```
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'1571B5EC78BD68512BF7830BB6A2A44B2047C7DF57BCE79EB8A1C0E5BEA0A501'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "AP_BL2",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'10159BAF262B43A92D95DB59DAE1F72C645127301661E0A3CE4E38B295A97C58'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "SCP_BL1",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'10122E856B3FCD49F063636317476149CB730A1AA1CFAAD818552B72F56D6F68'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "SCP_BL2",
            "SIGNER_ID": "b
↪'F14B4987904BCB5814E4459A057ED4D20F58A633152288A761214DCD28780B56'",
            "MEASUREMENT_VALUE": "b
↪'AA67A169B0BBA217AA0AA88A65346920C84C42447C36BA5F7EA65F422C1FE5D8'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "AP_BL31",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'2E6D31A5983A91251BFAE5AEFA1C0A19D8BA3CF601D0E8A706B4CFA9661A6B8A'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "RMM",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'A1FB50E6C86FAE1679EF3351296FD6713411A08CF8DD1790A4FD05FAE8688164'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "HW_CONFIG",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'1A252402972F6057FA53CC172B52B9FFCA698E18311FACD0F3B06ECAAEF79E17'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
```


(continues on next page)

(continued from previous page)

```
        },
        {
            "SW_COMPONENT_TYPE": "FW_CONFIG",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'9A92ADBC0CEE38EF658C71CE1B1BF8C65668F166BFB213644C895CCB1AD07A25'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "TB_FW_CONFIG",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'238903180CC104EC2C5D8B3F20C5BC61B389EC0A967DF8CC208CDC7CD454174F'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        },
        {
            "SW_COMPONENT_TYPE": "SOC_FW_CONFIG",
            "SIGNER_ID": "b
↪'5378796307535DF3EC8D8B15A2E2DC5641419C3D3060CFE32238C0FA973F7AA3'",
            "MEASUREMENT_VALUE": "b
↪'E6C21E8D260FE71882DEBDB339D2402A2CA7648529BC2303F48649BCE0380017'",
            "CCA_SW_COMPONENT_HASH_ID": "sha-256"
        }
    ]
}
```

10.5.5 RSE based DICE Protection Environment

The `DICE Protection Environment (DPE)` [Page 807, 8] service makes it possible to execute *DICE* commands within an isolated execution environment. It provides clients with an interface to send DICE commands, encoded as CBOR objects, that act on opaque context handles. The *DPE* service performs *DICE* derivations and certification on its internal contexts, without exposing the *DICE* secrets (private keys and CDIs) outside of the isolated execution environment.

DPE API

Defined here:

- `include/lib/psa/dice_protection_environment.h`

```
dpe_error_t
dpe_derive_context(int      context_handle,
                   uint32_t cert_id,
                   bool     retain_parent_context,
                   bool     allow_new_context_to_derive,
                   bool     create_certificate,
```

(continues on next page)

(continued from previous page)

```
                                const DiceInputValues *dice_inputs,
                                int32_t  target_locality,
                                bool     return_certificate,
                                bool     allow_new_context_to_export,
                                bool     export_cdi,
                                int     *new_context_handle,
                                int     *new_parent_context_handle,
                                uint8_t *new_certificate_buf,
                                size_t   new_certificate_buf_size,
                                size_t  *new_certificate_actual_size,
                                uint8_t *exported_cdi_buf,
                                size_t   exported_cdi_buf_size,
                                size_t  *exported_cdi_actual_size);
```

Build time config options

- `MEASURED_BOOT`: Enable measured boot.
- `DICE_PROTECTION_ENVIRONMENT`: Boolean flag to specify the measured boot backend when *RSE* based `MEASURED_BOOT` is enabled. The default value is `0`. When set to `1` then measurements and additional metadata collected during the measured boot process are sent to the *DPE* for storage and processing.
- `DPE_ALG_ID`: Determine the hash algorithm to measure the images. The default value is sha-256.

Example certificate chain

`plat/arm/board/tc/tc_dpe.h`

10.5.6 RSE OTP Assets Management

RSE provides access for AP to assets in OTP, which include keys for image signature verification and non-volatile counters for anti-rollback protection.

Non-Volatile Counter API

AP/RSE interface for retrieving and incrementing non-volatile counters API is as follows.

Defined here:

- `include/lib/psa/rse_platform_api.h`

```
psa_status_t rse_platform_nv_counter_increment(uint32_t counter_id)

psa_status_t rse_platform_nv_counter_read(uint32_t counter_id,
        uint32_t size, uint8_t *val)
```

Through this service, we can read/increment any of the 3 non-volatile counters used on an Arm CCA platform:

- `Non-volatile counter for CCA firmware (BL2, BL31, RMM).`
- `Non-volatile counter for secure firmware.`
- `Non-volatile counter for non-secure firmware.`

Get entropy API

AP/RSE interface for reading the entropy is as follows.

Defined here:

- `include/lib/psa/rse_platform_api.h`

```
psa_status_t rse_platform_get_entropy(uint8_t *data, size_t data_size)
```

Through this service, we can read an entropy generated from RSE.

10.5.7 References

10.6 PSCI OS-initiated mode

Author
Maulik Shah & Wing Li

Organization
Qualcomm Innovation Center, Inc. & Google LLC

Contact
Maulik Shah <quic_mkshah@quicinc.com> & Wing Li <wingers@google.com>

Status
Accepted

Table of Contents

- *PSCI OS-initiated mode*
  - *Introduction*
    - *Power state coordination*
      - *Platform-coordinated*

- *OS-initiated*
- *Motivation*
  - *Scalability*
  - *Simplicity*
  - *Current vendor implementations and workarounds*
- *Requirements*
  - *PSCI_FEATURES*
    - *CPU_SUSPEND feature flags*
  - *PSCI_SET_SUSPEND_MODE*
  - *CPU_SUSPEND*
    - *Power state formats*
    - *Races in OS-initiated mode*
- *Caveats*
  - *CPU_OFF*
- *Implementation*
  - *Current implementation of platform-coordinated mode*
  - *Proposed implementation of OS-initiated mode*
- *Testing*
  - *Testing on FVP and Google platforms*
  - *Testing on STM32MP15*
  - *Testing on Qualcomm SC7280*
  - *Comparisons on Qualcomm SC7280*
    - *CPUIdle states*
    - *Results*

10.6.1 Introduction

Power state coordination

A power domain topology is a logical hierarchy of power domains in a system that arises from the physical dependencies between power domains.

Local power states describe power states for an individual node, and composite power states describe the combined power states for an individual node and its parent node(s).

Entry into low-power states for a topology node above the core level requires coordinating its children nodes. For example, in a system with a power domain that encompasses a shared cache, and a separate power domain for each core that uses the shared cache, the core power domains must be powered down before the shared cache power domain can be powered down.

PSCI supports two modes of power state coordination: platform-coordinated and OS-initiated.

Platform-coordinated

Platform-coordinated mode is the default mode of power state coordination, and is currently the only supported mode in TF-A.

In platform-coordinated mode, the platform is responsible for coordinating power states, and chooses the deepest power state for a topology node that can be tolerated by its children.

OS-initiated

OS-initiated mode is optional.

In OS-initiated mode, the calling OS is responsible for coordinating power states, and may request for a topology node to enter a low-power state when its last child enters the low-power state.

10.6.2 Motivation

There are two reasons why OS-initiated mode might be a more suitable option than platform-coordinated mode for a platform.

Scalability

In platform-coordinated mode, each core independently selects their own local power states, and doesn't account for composite power states that are shared between cores.

In OS-initiated mode, the OS has knowledge of the next wakeup event for each core, and can have more precise control over the entry, exit, and wakeup latencies when deciding if a composite power state (e.g. for a cluster) is appropriate. This is especially important for multi-cluster SMP systems and heterogeneous systems like big.LITTLE, where different processor types can have different power efficiencies.

Simplicity

In platform-coordinated mode, the OS doesn't have visibility when the last core at a power level enters a low-power state. If the OS wants to perform last man activity (e.g. powering off a shared resource when it is no longer needed), it would have to communicate with an API side channel to know when it can do so. This could result in a design smell where the platform is using platform-coordinated mode when it should be using OS-initiated mode instead.

In OS-initiated mode, the OS can perform last man activity if it selects a composite power state when the last core enters a low-power state. This eliminates the need for a side channel, and uses the well documented API between the OS and the platform.

Current vendor implementations and workarounds

- STMicroelectronics
  - For their ARM32 platforms, they're using OS-initiated mode implemented in OP-TEE.
  - For their future ARM64 platforms, they are interested in using OS-initiated mode in TF-A.
- Qualcomm
  - For their mobile platforms, they're using OS-initiated mode implemented in their own custom secure monitor firmware.
  - For their Chrome OS platforms, they're using platform-coordinated mode in TF-A with custom driver logic to perform last man activity.
- Google
  - They're using platform-coordinated mode in TF-A with custom driver logic to perform last man activity.

Both Qualcomm and Google would like to be able to use OS-initiated mode in TF-A in order to simplify custom driver logic.

10.6.3 Requirements

PSCI_FEATURES

PSCI_FEATURES is for checking whether or not a PSCI function is implemented and what its properties are.

`PSCI_FEATURES`

Parameters

- **`func_id`** – 0x8400_000A.
- **`psci_func_id`** – the function ID of a PSCI function.

Return values

- **`NOT_SUPPORTED`** – if the function is not implemented.
- **`feature flags associated with the function`** – if the function is implemented.

CPU_SUSPEND feature flags

- Reserved, bits[31:2]
- Power state parameter format, bit[1]
  - A value of 0 indicates the original format is used.
  - A value of 1 indicates the extended format is used.
- OS-initiated mode, bit[0]
  - A value of 0 indicates OS-initiated mode is not supported.
  - A value of 1 indicates OS-initiated mode is supported.

See sections 5.1.14 and 5.15 of the PSCI spec (DEN0022D.b) for more details.

PSCI_SET_SUSPEND_MODE

PSCI_SET_SUSPEND_MODE is for switching between the two different modes of power state coordination.

`PSCI_SET_SUSPEND_MODE`

Parameters

- **`func_id`** – 0x8400_000F.
- **`mode`** – 0 indicates platform-coordinated mode, 1 indicates OS-initiated mode.

Return values

- **`SUCCESS`** – if the request is successful.
- **`NOT_SUPPORTED`** – if OS-initiated mode is not supported.
- **`INVALID_PARAMETERS`** – if the requested mode is not a valid value (0 or 1).
- **`DENIED`** – if the cores are not in the correct state.

Switching from platform-coordinated to OS-initiated is only allowed if the following conditions are met:

- All cores are in one of the following states:
  - Running.
  - Off, through a call to CPU_OFF or not yet booted.
  - Suspended, through a call to CPU_DEFAULT_SUSPEND.
- None of the cores has called CPU_SUSPEND since the last change of mode or boot.

Switching from OS-initiated to platform-coordinated is only allowed if all cores other than the calling core are off, either through a call to CPU_OFF or not yet booted.

If these conditions are not met, the PSCI implementation must return DENIED.

See sections 5.1.19 and 5.20 of the PSCI spec (DEN0022D.b) for more details.

CPU_SUSPEND

CPU_SUSPEND is for moving a topology node into a low-power state.

CPU_SUSPEND

Parameters

- **func_id** – 0xC400_0001.
- **power_state** – the requested low-power state to enter.
- **entry_point_address** – the address at which the core must resume execution following wakeup from a powerdown state.
- **context_id** – this field specifies a pointer to the saved context that must be restored on a core following wakeup from a powerdown state.

Return values

- **SUCCESS** – if the request is successful.
- **INVALID_PARAMETERS** – in OS-initiated mode, this error is returned when a low-power state is requested for a topology node above the core level, and at least one of the node's children is in a local low-power state that is incompatible with the request.
- **INVALID_ADDRESS** – if the entry_point_address argument is invalid.
- **DENIED** – only in OS-initiated mode; this error is returned when a low-power state is requested for a topology node above the core level, and at least one of the node's children is running, i.e. not in a low-power state.

In platform-coordinated mode, the PSCI implementation coordinates requests from all cores to determine the deepest power state to enter.

In OS-initiated mode, the calling OS is making an explicit request for a specific power state, as opposed to expressing a vote. The PSCI implementation must comply with the request, unless the request is not consistent with the implementation's view of the system's state, in which case, the implementation must return INVALID_PARAMETERS or DENIED.

See sections 5.1.2 and 5.4 of the PSCI spec (DEN0022D.b) for more details.

Power state formats

Original format

- Power Level, bits[25:24]
  - The requested level in the power domain topology to enter a low-power state.
- State Type, bit[16]
  - A value of 0 indicates a standby or retention state.
  - A value of 1 indicates a powerdown state.

- State ID, bits[15:0]
  - Field to specify the requested composite power state.
  - The state ID encodings must uniquely describe every possible composite power state.
  - In OS-initiated mode, the state ID encoding must allow expressing the power level at which the calling core is the last to enter a powerdown state.

Extended format

- State Type, bit[30]
- State ID, bits[27:0]

Races in OS-initiated mode

In OS-initiated mode, there are race windows where the OS's view and implementation's view of the system's state differ. It is possible for the OS to make requests that are invalid given the implementation's view of the system's state. For example, the OS might request a powerdown state for a node from one core, while at the same time, the implementation observes that another core in that node is powering up.

To address potential race conditions in power state requests:

- The calling OS must specify in each CPU_SUSPEND request the deepest power level for which it sees the calling core as the last running core (last man). This is required even if the OS doesn't want the node at that power level to enter a low-power state.
- The implementation must validate that the requested power states in the CPU_SUSPEND request are consistent with the system's state, and that the calling core is the last core running at the requested power level, or deny the request otherwise.

See sections 4.2.3.2, 6.2, and 6.3 of the PSCI spec (DEN0022D.b) for more details.

10.6.4 Caveats

CPU_OFF

CPU_OFF is always platform-coordinated, regardless of whether the power state coordination mode for suspend is platform-coordinated or OS-initiated. If all cores in a topology node call CPU_OFF, the last core will power down the node.

In OS-initiated mode, if a subset of the cores in a topology node has called CPU_OFF, the last running core may call CPU_SUSPEND to request a powerdown state at or above that node's power level.

See section 5.5.2 of the PSCI spec (DEN0022D.b) for more details.

10.6.5 Implementation

Current implementation of platform-coordinated mode

Platform-coordinated is currently the only supported power state coordination mode in TF-A.

The functions of interest in the `psci_cpu_suspend` call stack are as follows:

- `psci_validate_power_state`
  - This function calls a platform specific `validate_power_state` handler, which takes the `power_state` parameter, and updates the `state_info` object with the requested states for each power level.
- `psci_find_target_suspend_lvl`
  - This function takes the `state_info` object containing the requested power states for each power level, and returns the deepest power level that was requested to enter a low power state, i.e. the target power level.
- `psci_do_state_coordination`
  - This function takes the target power level and the `state_info` object containing the requested power states for each power level, and updates the `state_info` object with the coordinated target power state for each level.
- `pwr_domain_suspend`
  - This is a platform specific handler that takes the `state_info` object containing the target power states for each power level, and transitions each power level to the specified power state.

Proposed implementation of OS-initiated mode

To add support for OS-initiated mode, the following changes are proposed:

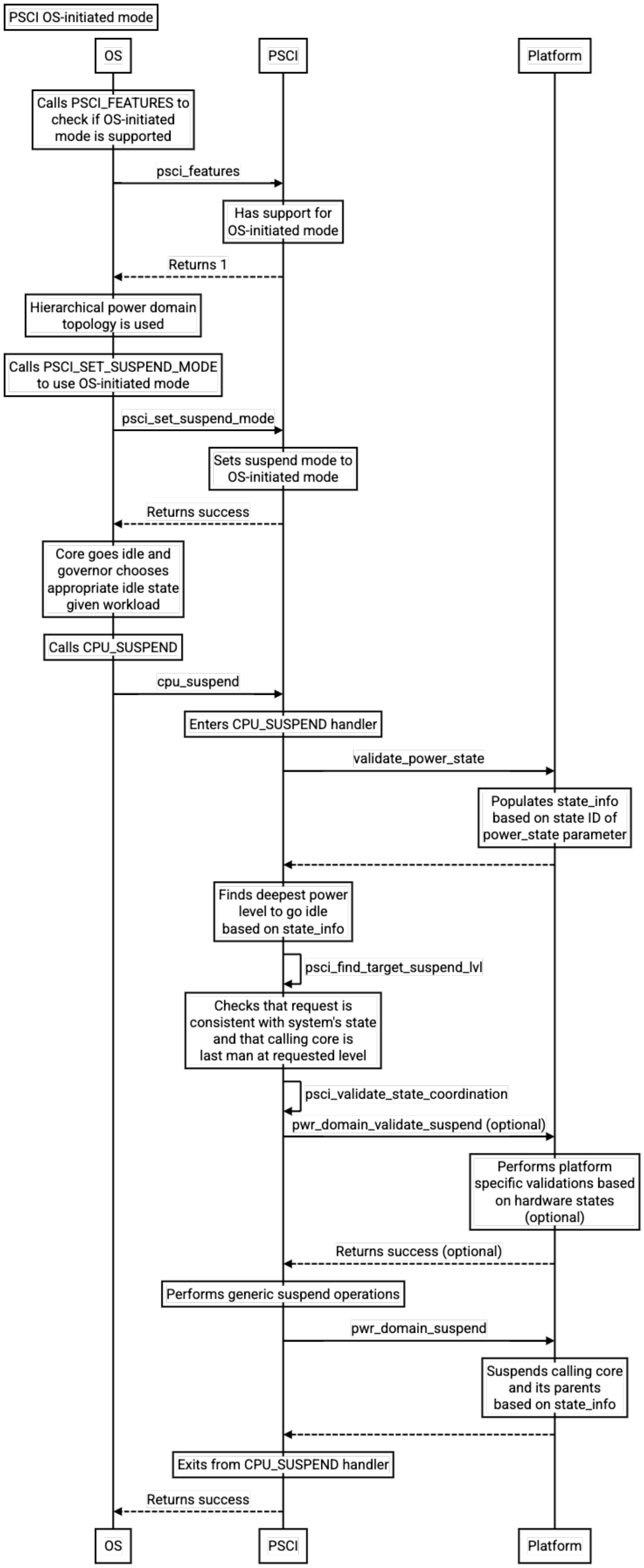
- Add a boolean build option `PSCI_OS_INIT_MODE` for a platform to enable optional support for PSCI OS-initiated mode. This build option defaults to 0.

Note: If `PSCI_OS_INIT_MODE=0`, the following changes will not be compiled into the build.

- Update `psci_features` to return 1 in bit[0] to indicate support for OS-initiated mode for CPU_SUSPEND.
- Define a `suspend_mode` enum: `PLAT_COORD` and `OS_INIT`.
- Define a `psci_suspend_mode` global variable with a default value of `PLAT_COORD`.
- Implement a new function handler `psci_set_suspend_mode` for PSCI_SET_SUSPEND_MODE.
- Since `psci_validate_power_state` calls a platform specific `validate_power_state` handler, the platform implementation should populate the `state_info` object based on the state ID from the given `power_state` parameter.

- `psci_find_target_suspend_lvl` remains unchanged.
- Implement a new function `psci_validate_state_coordination` that ensures the request satisfies the following conditions, and denies any requests that don't:
  - The requested power states for each power level are consistent with the system's state
  - The calling core is the last core running at the requested power level

  This function differs from `psci_do_state_coordination` in that:
  - The `psci_req_local_pwr_states` map is not modified if the request were to be denied
  - The `state_info` argument is never modified since it contains the power states requested by the calling OS
- Update `psci_cpu_suspend_start` to do the following:
  - If `PSCI_SUSPEND_MODE` is `PLAT_COORD`, call `psci_do_state_coordination`.
  - If `PSCI_SUSPEND_MODE` is `OS_INIT`, call `psci_validate_state_coordination`. If validation fails, propagate the error up the call stack.
- Add a new optional member `pwr_domain_validate_suspend` to `plat_psci_ops_t` to allow the platform to optionally perform validations based on hardware states.
- The platform specific `pwr_domain_suspend` handler remains unchanged.


PSCI OS-initiated mode
OS
PSCI
Platform
Calls PSCI_FEATURES to check if OS-initiated mode is supported
psci_features
Has support for OS-initiated mode
Returns 1
Hierarchical power domain topology is used
Calls PSCI_SET_SUSPEND_MODE to use OS-initiated mode
psci_set_suspend_mode
Sets suspend mode to OS-initiated mode
Returns success
Core goes idle and governor chooses appropriate idle state given workload
Calls CPU_SUSPEND
cpu_suspend
Enters CPU_SUSPEND handler
validate_power_state
Populates state_info based on state ID of power_state parameter
Finds deepest power level to go idle based on state_info
psci_find_target_suspend_lvl
Checks that request is consistent with system's state and that calling core is last man at requested level
psci_validate_state_coordination
pwr_domain_validate_suspend (optional)
Performs platform specific validations based on hardware states (optional)
Returns success (optional)
Performs generic suspend operations
pwr_domain_suspend
Suspends calling core and its parents based on state_info
Exits from CPU_SUSPEND handler
Returns success
OS
PSCI
Platform

10.6.6 Testing

The proposed patches can be found at https://review.trustedfirmware.org/q/topic:psci-osi.

Testing on FVP and Google platforms

The proposed patches add a new CPU Suspend in OSI mode test suite to TF-A Tests. This has been enabled and verified on the FVP_Base_RevC-2xAEMvA platform and Google platforms, and excluded from all other platforms via the build option `PLAT_TESTS_SKIP_LIST`.

Testing on STM32MP15

The proposed patches have been tested and verified on the STM32MP15 platform, which has a single cluster with 2 CPUs, by Gabriel Fernandez <gabriel.fernandez@st.com> from STMicroelectronics with this device tree configuration:

```
cpus {
        #address-cells = <1>;
        #size-cells = <0>;

        cpu0: cpu@0 {
                device_type = "cpu";
                compatible = "arm,cortex-a7";
                reg = <0>;
                enable-method = "psci";
                power-domains = <&CPU_PD0>;
                power-domain-names = "psci";
        };
        cpu1: cpu@1 {
                device_type = "cpu";
                compatible = "arm,cortex-a7";
                reg = <1>;
                enable-method = "psci";
                power-domains = <&CPU_PD1>;
                power-domain-names = "psci";
        };

        idle-states {
                cpu_retention: cpu-retention {
                        compatible = "arm,idle-state";
                        arm,psci-suspend-param = <0x00000001>;
                        entry-latency-us = <130>;
                        exit-latency-us = <620>;
                        min-residency-us = <700>;
                        local-timer-stop;
                };
        };

        domain-idle-states {
                CLUSTER_STOP: core-power-domain {
```

(continues on next page)

(continued from previous page)

```
                                compatible = "domain-idle-state";
                                arm,psci-suspend-param = <0x01000001>;
                                entry-latency-us = <230>;
                                exit-latency-us = <720>;
                                min-residency-us = <2000>;
                                local-timer-stop;
                        };
                };
};

psci {
        compatible = "arm,psci-1.0";
        method = "smc";

        CPU_PD0: power-domain-cpu0 {
                #power-domain-cells = <0>;
                power-domains = <&pd_core>;
                domain-idle-states = <&cpu_retention>;
        };

        CPU_PD1: power-domain-cpu1 {
                #power-domain-cells = <0>;
                power-domains = <&pd_core>;
                domain-idle-states = <&cpu_retention>;
        };

        pd_core: power-domain-cluster {
                #power-domain-cells = <0>;
                domain-idle-states = <&CLUSTER_STOP>;
        };
};
```

Testing on Qualcomm SC7280

The proposed patches have been tested and verified on the SC7280 platform by Maulik Shah <quic_mkshah@quicinc.com> from Qualcomm with this device tree configuration:

```
cpus {
        #address-cells = <2>;
        #size-cells = <0>;

        CPU0: cpu@0 {
                device_type = "cpu";
                compatible = "arm,kryo";
                reg = <0x0 0x0>;
                enable-method = "psci";
                power-domains = <&CPU_PD0>;
                power-domain-names = "psci";
        };
```

(continues on next page)

(continued from previous page)

```
        CPU1: cpu@100 {
                device_type = "cpu";
                compatible = "arm,kryo";
                reg = <0x0 0x100>;
                enable-method = "psci";
                power-domains = <&CPU_PD1>;
                power-domain-names = "psci";
        };

        CPU2: cpu@200 {
                device_type = "cpu";
                compatible = "arm,kryo";
                reg = <0x0 0x200>;
                enable-method = "psci";
                power-domains = <&CPU_PD2>;
                power-domain-names = "psci";
        };

        CPU3: cpu@300 {
                device_type = "cpu";
                compatible = "arm,kryo";
                reg = <0x0 0x300>;
                enable-method = "psci";
                power-domains = <&CPU_PD3>;
                power-domain-names = "psci";
        }

        CPU4: cpu@400 {
                device_type = "cpu";
                compatible = "arm,kryo";
                reg = <0x0 0x400>;
                enable-method = "psci";
                power-domains = <&CPU_PD4>;
                power-domain-names = "psci";
        };

        CPU5: cpu@500 {
                device_type = "cpu";
                compatible = "arm,kryo";
                reg = <0x0 0x500>;
                enable-method = "psci";
                power-domains = <&CPU_PD5>;
                power-domain-names = "psci";
        };

        CPU6: cpu@600 {
                device_type = "cpu";
                compatible = "arm,kryo";
                reg = <0x0 0x600>;
                enable-method = "psci";
                power-domains = <&CPU_PD6>;
                power-domain-names = "psci";
```

(continues on next page)

(continued from previous page)

```
        };

        CPU7: cpu@700 {
                device_type = "cpu";
                compatible = "arm,kryo";
                reg = <0x0 0x700>;
                enable-method = "psci";
                power-domains = <&CPU_PD7>;
                power-domain-names = "psci";
        };

        idle-states {
                entry-method = "psci";

                LITTLE_CPU_SLEEP_0: cpu-sleep-0-0 {
                        compatible = "arm,idle-state";
                        idle-state-name = "little-power-down";
                        arm,psci-suspend-param = <0x40000003>;
                        entry-latency-us = <549>;
                        exit-latency-us = <901>;
                        min-residency-us = <1774>;
                        local-timer-stop;
                };

                LITTLE_CPU_SLEEP_1: cpu-sleep-0-1 {
                        compatible = "arm,idle-state";
                        idle-state-name = "little-rail-power-down";
                        arm,psci-suspend-param = <0x40000004>;
                        entry-latency-us = <702>;
                        exit-latency-us = <915>;
                        min-residency-us = <4001>;
                        local-timer-stop;
                };

                BIG_CPU_SLEEP_0: cpu-sleep-1-0 {
                        compatible = "arm,idle-state";
                        idle-state-name = "big-power-down";
                        arm,psci-suspend-param = <0x40000003>;
                        entry-latency-us = <523>;
                        exit-latency-us = <1244>;
                        min-residency-us = <2207>;
                        local-timer-stop;
                };

                BIG_CPU_SLEEP_1: cpu-sleep-1-1 {
                        compatible = "arm,idle-state";
                        idle-state-name = "big-rail-power-down";
                        arm,psci-suspend-param = <0x40000004>;
                        entry-latency-us = <526>;
                        exit-latency-us = <1854>;
                        min-residency-us = <5555>;
                        local-timer-stop;
```

(continues on next page)

(continued from previous page)

```
                };
        };

        domain-idle-states {
                CLUSTER_SLEEP_0: cluster-sleep-0 {
                        compatible = "arm,idle-state";
                        idle-state-name = "cluster-power-down";
                        arm,psci-suspend-param = <0x40003444>;
                        entry-latency-us = <3263>;
                        exit-latency-us = <6562>;
                        min-residency-us = <9926>;
                        local-timer-stop;
                };
        };
};

psci {
        compatible = "arm,psci-1.0";
        method = "smc";

        CPU_PD0: cpu0 {
                #power-domain-cells = <0>;
                power-domains = <&CLUSTER_PD>;
                domain-idle-states = <&LITTLE_CPU_SLEEP_0 &LITTLE_CPU_SLEEP_1>
↪;
        };

        CPU_PD1: cpu1 {
                #power-domain-cells = <0>;
                power-domains = <&CLUSTER_PD>;
                domain-idle-states = <&LITTLE_CPU_SLEEP_0 &LITTLE_CPU_SLEEP_1>
↪;
        };

        CPU_PD2: cpu2 {
                #power-domain-cells = <0>;
                power-domains = <&CLUSTER_PD>;
                domain-idle-states = <&LITTLE_CPU_SLEEP_0 &LITTLE_CPU_SLEEP_1>
↪;
        };

        CPU_PD3: cpu3 {
                #power-domain-cells = <0>;
                power-domains = <&CLUSTER_PD>;
                domain-idle-states = <&LITTLE_CPU_SLEEP_0 &LITTLE_CPU_SLEEP_1>
↪;
        };

        CPU_PD4: cpu4 {
                #power-domain-cells = <0>;
                power-domains = <&CLUSTER_PD>;
                domain-idle-states = <&BIG_CPU_SLEEP_0 &BIG_CPU_SLEEP_1>;
```

(continues on next page)

(continued from previous page)

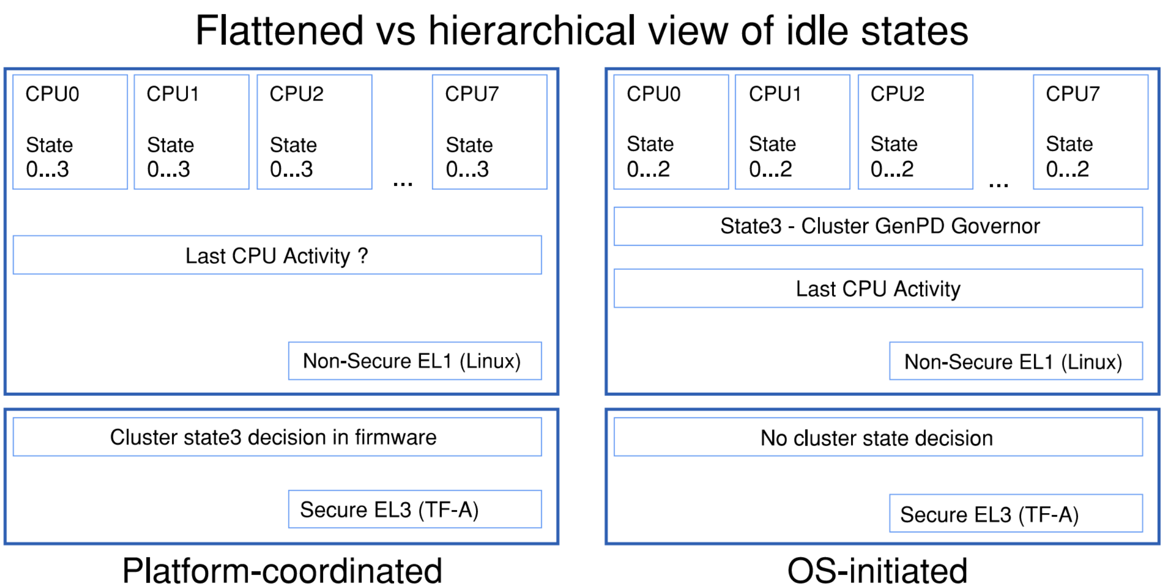
```
        };

        CPU_PD5: cpu5 {
                #power-domain-cells = <0>;
                power-domains = <&CLUSTER_PD>;
                domain-idle-states = <&BIG_CPU_SLEEP_0 &BIG_CPU_SLEEP_1>;
        };

        CPU_PD6: cpu6 {
                #power-domain-cells = <0>;
                power-domains = <&CLUSTER_PD>;
                domain-idle-states = <&BIG_CPU_SLEEP_0 &BIG_CPU_SLEEP_1>;
        };

        CPU_PD7: cpu7 {
                #power-domain-cells = <0>;
                power-domains = <&CLUSTER_PD>;
                domain-idle-states = <&BIG_CPU_SLEEP_0 &BIG_CPU_SLEEP_1>;
        };

        CLUSTER_PD: cpu-cluster0 {
                #power-domain-cells = <0>;
                domain-idle-states = <&CLUSTER_SLEEP_0>;
        };
};
```

Comparisons on Qualcomm SC7280

CPUIdle states

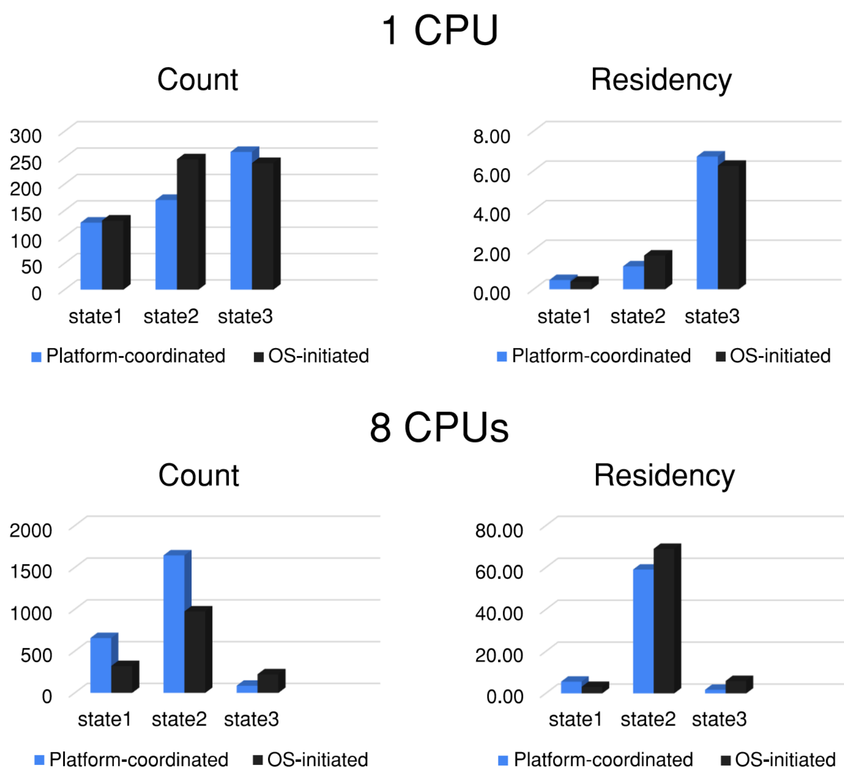
- 8 CPUs, 1 L3 cache
- Platform-coordinated mode
  - CPUIdle states
    - State0 - WFI
    - State1 - Core collapse
    - State2 - Rail collapse
    - State3 - L3 cache off and system resources voted off
- OS-initiated mode
  - CPUIdle states
    - State0 - WFI
    - State1 - Core collapse
    - State2 - Rail collapse

- Cluster domain idle state
    - State3 - L3 cache off and system resources voted off



Results

- The following stats have been captured with fixed CPU frequencies from the use case of 10 seconds of device idle with the display turned on and Wi-Fi and modem turned off.
- Count refers to the number of times a CPU or cluster entered power collapse.
- Residency refers to the time in seconds a CPU or cluster stayed in power collapse.
- The results are an average of 3 iterations of actual counts and residencies.



OS-initiated mode was able to scale better than platform-coordinated mode for multiple CPUs. The count and residency results for state3 (i.e. a cluster domain idle state) in OS-initiated mode for multiple CPUs were much closer to the results for a single CPU than in platform-coordinated mode.

10.7 Measured Boot Design

This document briefly explains the Measured-Boot design implementation in *TF-A*.

10.7.1 Introduction

Measured Boot is the process of computing and securely recording hashes of code and critical data at each stage in the boot chain before the code/data is used.

These measurements can be leveraged by other components in the system to implement a complete attestation system. For example, they could be used to enforce local attestation policies (such as releasing certain platform keys or not), or they could be securely sent to a remote challenger a.k.a. *verifier* after boot to attest to the state of the code and critical-data.

Measured Boot does not authenticate the code or critical-data, but simply records what code/critical-data was present on the system during boot.

It is assumed that BL1 is implicitly trusted (by virtue of immutability) and acts as the root of trust for measurement hence it is not measured.

The Measured Boot implementation in TF-A supports multiple backends to securely store measurements mentioned below in the *Measured Boot Backends* section.

10.7.2 Critical data

All firmware images - i.e. BLx images and their corresponding configuration files, if any - must be measured. In addition to that, there might be specific pieces of data which needs to be measured as well. These are typically different on each platform. They are referred to as *critical data*.

Critical data for the platform can be determined using the following criteria:

1. Data that influence boot flow behaviour such as -
   - Configuration parameters that alter the boot flow path.
   - Parameters that determine which firmware to load from NV-Storage to SRAM/DRAM to pass the boot process successfully.
2. Hardware configurations settings, debug settings and security policies that need to be in a valid state for a device to maintain its security posture during boot and runtime.
3. Security-sensitive data that is being updated by hardware.

Examples of Critical data:

1. The list of errata workarounds being applied at reset.
2. State of fuses such as whether an SoC is in secure mode.
3. NV counters that determine whether firmware is up-to-date and secure.

10.7.3 Measurement slot

The measurement slot resides in a Trusted Module and can be either a secure register or memory. The measurement slot is used to provide a method to cryptographically record (measure) images and critical data on a platform. The measurement slot update calculation, called an **extend** operation, is a one-way hash of all the previous measurements and the new measurement. It is the only way to change the slot value, thus no measurements can ever be removed or overwritten.

10.7.4 Measured Boot Backends

The Measured Boot implementation in TF-A supports:

1. Event Log

   The TCG Event Log holds a record of measurements made into the Measurement Slot aka PCR (Platform Configuration Register).

The TCG EFI Protocol Specification provides details on how to measure components. The Arm document Arm® Server Base Security Guide provides specific guidance for measurements on an SBSA/SBBR server system. By considering these specifications it is decided that -

1. Use PCR0 for images measurements.
2. Use PCR1 for Critical data measurements.

TCG has specified the architecture for the structure of this log in the TCG EFI Protocol Specification. The specification describes two event log event records—the legacy, fixed size SHA1 structure called TCG_PCR_EVENT and the variable length crypto agile structure called TCG_PCR_EVENT2. Event Log driver implemented in TF-A covers later part.

2. Discrete TPM

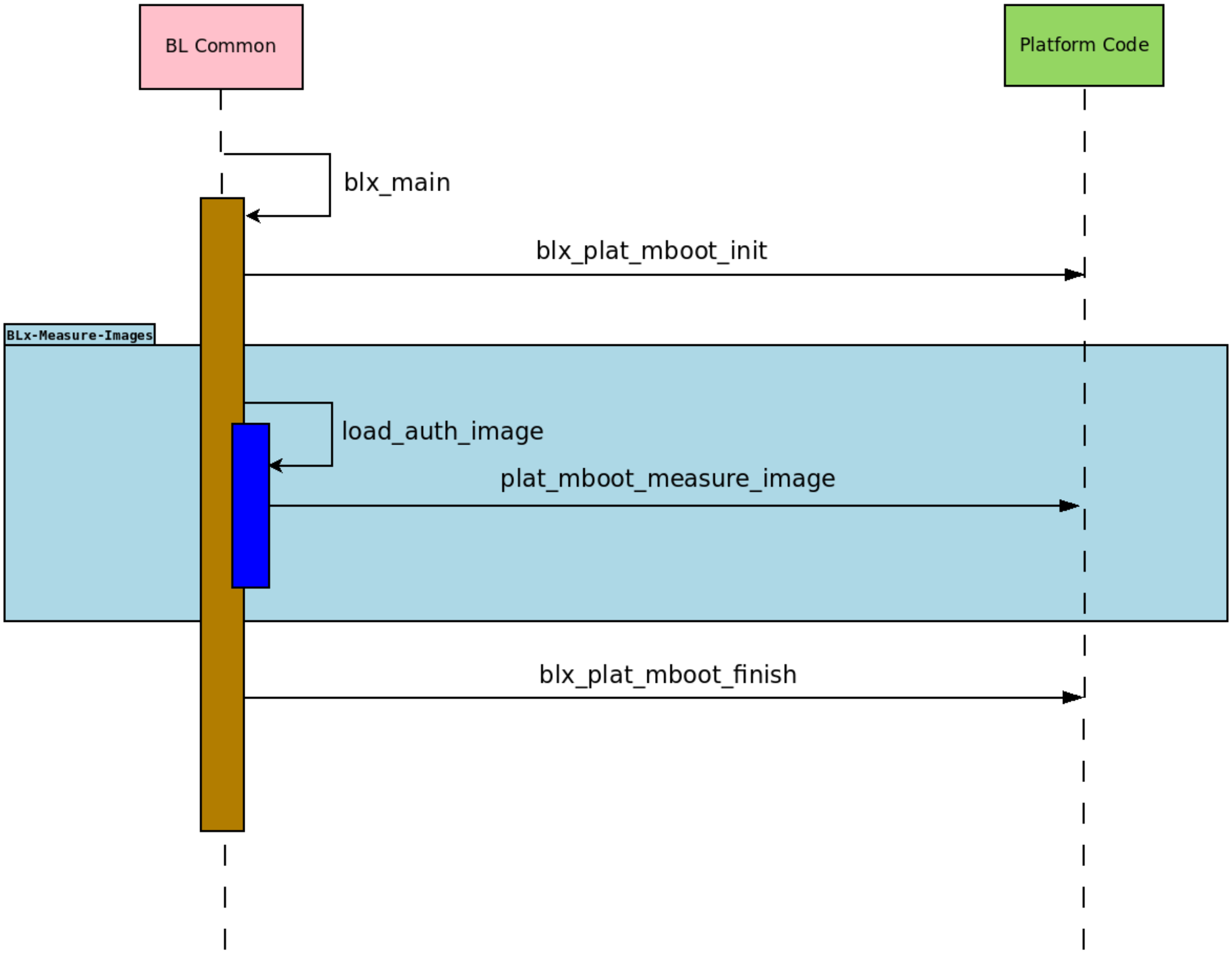
   A Discrete TPM (Trusted Platform Module) can be used alongside Event Log to extend measurements and validate Measured Boot functionality. The use of a Discrete TPM in TF-A to extend measurements of images and other critical data allows for an additional layer of security. The TPM can be used to attest the integrity of the Event Log.

3. *RSE*

   It is one of the physical backends to extend the measurements. Please refer this document *Runtime Security Engine (RSE)* for more details.

10.7.5 Platform Interface

Every image which gets successfully loaded in memory (and authenticated, if trusted boot is enabled) then gets measured. In addition to that, platforms can measure any relevant piece of critical data at any point during the boot. The following diagram outlines the call sequence for Measured Boot platform interfaces invoked from generic code:



These platform interfaces are used by BL1 and BL2 only, and are declared in `include/plat/common/platform.h`. BL31 does not load and thus does not measure any image.

Responsibilities of these platform interfaces are -

1. **Function : blx_plat_mboot_init()**

```
void bl1_plat_mboot_init(void);
void bl2_plat_mboot_init(void);
```

   Initialise all Measured Boot backends supported by the platform (e.g. Event Log buffer, *RSE*). As these functions do not return any value, the platform should deal with error management, such as logging the error somewhere, or panicking the system if this is considered a fatal error.

   - On the Arm FVP port -
     - In BL1, this function is used to initialize the Event Log backend driver, and also to write header information in the Event Log buffer.
     - In BL2, this function is used to initialize the Event Log buffer with the information received from the BL1. It results in panic on error.

2. **Function : plat_mboot_measure_image()**

```
int plat_mboot_measure_image(unsigned int image_id,
                             image_info_t *image_data);
```

- Measure the image using a hash function of the crypto module.
- Record the measurement in the corresponding backend -
  - If it is Event Log backend, then record the measurement in TCG Event Log format.
  - If it is a secure crypto-processor (like *RSE*), then extend the designated PCR (or store it in secure on-chip memory) with the given measurement.
- This function must return 0 on success, a signed integer error code otherwise.
- On the Arm FVP port, this function measures the given image and then records that measurement in the Event Log buffer. The passed id is used to retrieve information about on how to measure the image (e.g. PCR number).

3. **Function : blx_plat_mboot_finish()**

```
void bl1_plat_mboot_finish(void);
void bl2_plat_mboot_finish(void);
```

- Do all teardown operations with respect to initialised Measured Boot backends. This could be -
  - Pass the Event Log details (start address and size) to Normal world or to Secure World using any platform implementation way.
  - Measure all critical data if any.
  - As these functions do not return any value, the platform should deal with error management, such as logging the error somewhere, or panicking the system if this is considered a fatal error.
- On the Arm FVP port -
  - In BL1, this function is used to pass the base address of the Event Log buffer and its size to BL2 via tb_fw_config to extend the Event Log buffer with the measurement of various images loaded by BL2. It results in panic on error.
  - In BL2, this function is used to pass the Event Log buffer information (base address and size) to non-secure(BL33) and trusted OS(BL32) via nt_fw and tos_fw config respectively. See *DTB binding for Event Log properties* for a description of the bindings used for Event Log properties.

4. **Function : plat_mboot_measure_critical_data()**

```
int plat_mboot_measure_critical_data(unsigned int critical_data_id,
                                     const void *base,
                                     size_t size);
```

This interface is not invoked by the generic code and it is up to the platform layer to call it where appropriate.

This function measures the given critical data structure and records its measurement using the Measured Boot backend driver. This function must return 0 on success, a signed integer error code otherwise.

In FVP, Non volatile counters get measured and recorded as Critical data using the backend via this interface.

5. **Function : plat_mboot_measure_key()**

```
int plat_mboot_measure_key(const void *pk_oid, const void *pk_ptr,
                           size_t pk_len);
```

- This function is used by the platform to measure the passed key and publicise it using any of the supported backends.
- The authentication module within the trusted boot framework calls this function for every ROTPK involved in verifying the signature of a root certificate and for every subsidiary key that gets extracted from a key certificate for later authentication of a content certificate.
- A cookie, passed as the first argument, serves as a key-OID pointer associated with the public key data, passed as the second argument.
- Public key data size is passed as the third argument to this function.
- This function must return 0 on success, a signed integer error code otherwise.
- In Total Compute platform, this function is used to calculate the hash of the given key and forward this hash to *RSE* alongside the measurement of the image which the key signs.

6. **Function : crypto_mod_tcg_hash()**

```
int crypto_mod_tcg_hash(uint32_t tpm_alg_id, void *data,
                        unsigned int len, uint8_t *digest);
```

- Provided by the common crypto module, this helper translates TPM/TCG algorithm identifiers (`tpm_alg_id`) to the crypto backend so that Event Log measurements use the same digest implementation as the platform PCR backend (software crypto, hardware accelerator or a discrete TPM).
- The helper hashes the `len` bytes pointed to by `data` with the algorithm requested by `tpm_alg_id` and writes the resulting digest to `digest`. The buffer is sized according to the Event Log SpecID header, so the caller only needs to fill in the bytes.
- This function returns 0 on success. It returns a signed errno-style error code (for example `-EOPNOTSUPP` for an unsupported algorithm or `-EINVAL` for invalid arguments) if hashing fails. Generic measured boot code propagates failures to the caller.
- Register the shim by passing `crypto_mod_tcg_hash` as the `evlog_hash_func_t` argument to `event_log_init_and_reg()` (or the platform helper that wraps it) so that the common Event Log driver automatically invokes the common hashing backend whenever it needs a digest. Platforms with special hardware requirements may still provide a custom implementation if needed.

10.8 Discrete TPM drivers

This section focuses on the design and functionality of Discrete TPM drivers in *TF-A*. The *TPM* technology is designed to provide a dedicated, hardware-based solution for storing cryptographic keys and performing security-related operations.

Discrete TPMs are separate, standalone hardware components that are physically isolated from the system's main processor. This isolation helps protect sensitive information, such as encryption keys and platform credentials, from being accessed or tampered with by malicious software or unauthorized users. When a Discrete TPM interface is implemented correctly, the risk of software based attacks is reduced, further reducing the attack surface.

TPM measurements establish the security posture of a system and are used for attestation. Performing measurements using a TPM in TF-A is beneficial from a security standpoint because it ensures hardware-backed attestation earlier in the boot flow, reducing the risk of a compromised root of trust.

The design implemented in TF-A supports multiple types of TPM hardware interfaces and hardware bus types in order to be compatible with different platforms. Platforms opt to use a specific messaging interface, such as *CRB* or *FIFO*, and a specific hardware bus interface, such as *I2C* or *SPI*.

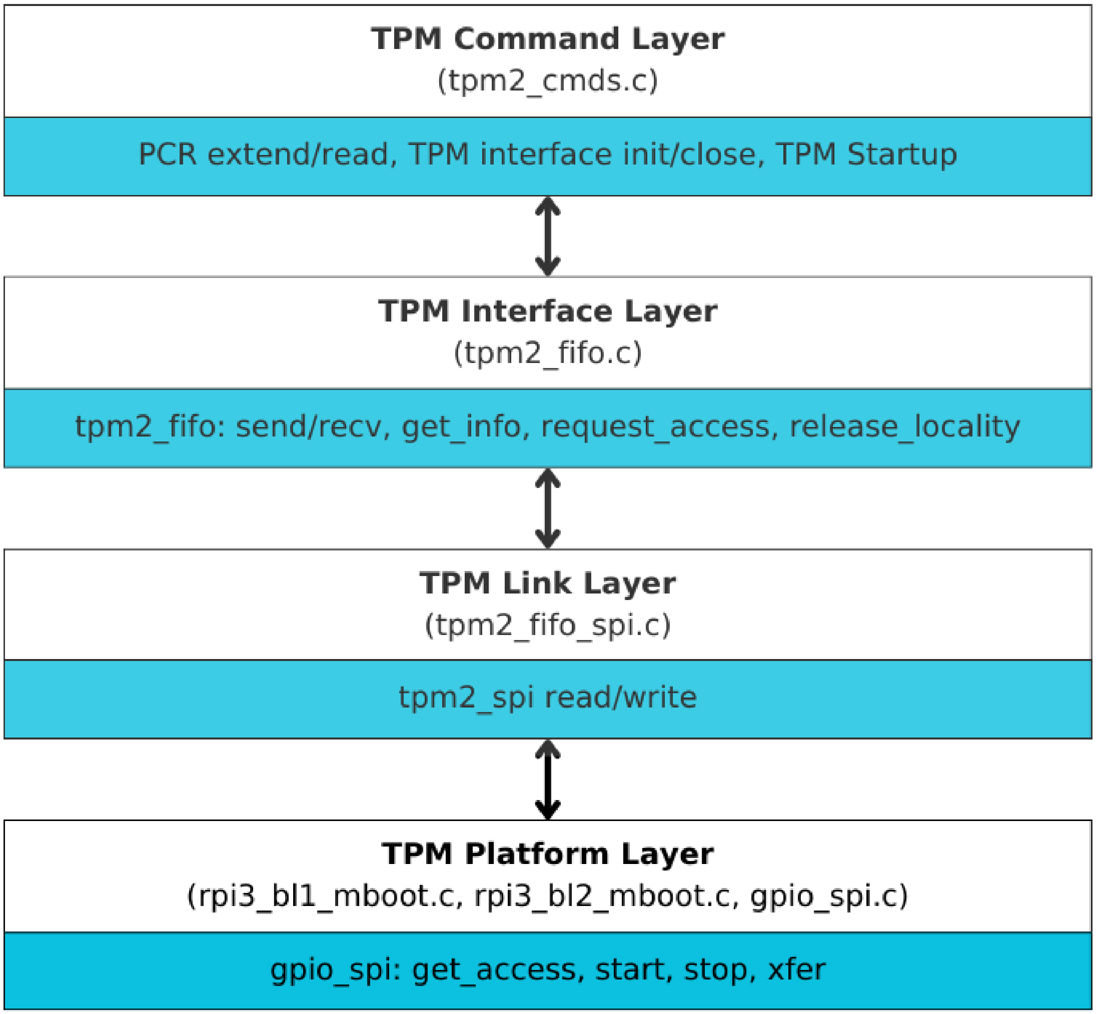
10.8.1 Driver architecture

The Discrete TPM drivers are split up into four layers, each serving a distinct purpose in the overall architecture:

- **Command Layer**: This layer provides various TPM commands based on the TCG TPM 2.0 Library Specification. It allows a system to initialize the TPM interface, perform a TPM startup, set up a locality for operations like PCR extend and read, and release the locality when finished.
- **Interface Layer**: This layer handles sending and receiving TPM commands via a specific TPM interface like FIFO or CRB. It also includes functions such as getting information, requesting access, and relinquishing access, tailored to the specific interface.
- **Link Layer**: Discrete TPMs may appear as a SPI, I2C, or memory mapped device. The link layer maps the command passed from the interface layer to the appropriate bus type. It includes hardware link read and write functions that use the platform bus interface to transfer commands.
- **Platform Layer**: The platform layer implements the details of how to communicate to the TPM chip for a specific platform. The link layer uses the platform layer to read and write to the TPM.

Note: The command, interface, and link layers are implemented in common code in TF-A. The platform layer is implemented in platform specific code.

The following diagram illustrates the Discrete TPM driver stack for the Raspberry Pi 3 platform.



Header files

- TPM Drivers: `include/drivers/tpm`

Source files

- TPM Drivers: `drivers/tpm`

10.8.2 Build time config options

- `MBOOT_TPM_HASH_ALG`: The hash algorithm to be used by the TPM, currently the only supported algorithm is `sha256`. As additional Discrete TPMs are tested and integrated in TF-A, support for more algorithms will become available.
- `DISCRETE_TPM`: Boolean flag to enable Discrete TPM support. Depending on the selected TPM interface, the appropriate drivers will be built and packaged into firmware.
- `TPM_INTERFACE`: This flag is required when `DISCRETE_TPM=1`, currently the only supported interface is `FIFO_SPI`. Ideally there should be four options:

```
FIFO_I2C
FIFO_SPI
FIFO_MMIO
CRB
```

Note: `MBOOT_TPM_HASH_ALG` will automatically overwrite `MBOOT_EL_HASH_ALG`. This is to ensure the event log and the TPM are using the same hash algorithm.

10.8.3 Discrete TPM Initialization

The TPM needs to be initialized based on the platform, the hardware interfaces need to be set up independently, and once they are setup, the TPM commands `tpm_interface_init()` and subsequently `tpm_startup()` can be called. `tpm_startup()` only needs to be called once after startup, or if the system is reset.

An example of platform specific TPM hardware initialization for the rpi3 can be found in `plat/rpi/rpi3/rpi3_bl1_setup.c` and `plat/rpi/rpi3/rpi3_bl1_mboot.c`

10.8.4 Discrete TPM PCR Extend

Once the TPM is setup, the TPM `pcr_extend` operation can be used to extend hashes and store them in PCR 0.

An example of `pcr_extend` that is used during rpi3 measured boot can be found
in `plat/rpi/rpi3/rpi3_bl1_mboot.c` and `plat/rpi/rpi3/rpi3_bl2_mboot.c`.

CHAPTER

ELEVEN

THREAT MODEL

Threat modeling is an important part of Secure Development Lifecycle (SDL) that helps us identify potential threats and mitigations affecting a system.

11.1 TF-A Firmware Threat Model

As the TF-A codebase is highly configurable to allow tailoring it best for each platform's needs, providing a holistic threat model covering all of its features is not necessarily the best approach. Instead, we provide a collection of documents which, together, form the project's threat model. These are articulated around a core document, called the *Generic Threat Model*, which focuses on the most common configuration we expect to see. The other documents typically focus on specific features not covered in the core document.

As the TF-A codebase evolves and new features get added, these threat model documents will be updated and extended in parallel to reflect at best the current status of the code from a security standpoint.

> **Note:** Although our aim is eventually to provide threat model material for all features within the project, we have not reached that point yet. We expect to gradually fill these gaps over time.
>
> Also, when a non-trivial feature gets introduced into the code base, it is often tagged as experimental for some period of time. Such experimental features typically may not be formally threat-modelled until their design and implementation stabilize. Therefore, these features are considered out of scope of TF-A's threat model. See *Experimental features* and *Experimental build options* for more details.

Each of these documents give a description of the target of evaluation using a data flow diagram, as well as a list of threats we have identified using the STRIDE threat modeling technique and corresponding mitigations.

11.1.1 Generic Threat Model

Introduction

This document provides a generic threat model for TF-A firmware.

Target of Evaluation

In this threat model, the target of evaluation is the Trusted Firmware for A-class Processors (TF-A). This includes the boot ROM (BL1), the trusted boot firmware (BL2) and the runtime EL3 firmware (BL31) as shown on Figure 1. Everything else on Figure 1 is outside of the scope of the evaluation.

TF-A can be configured in various ways. In this threat model we consider only the most basic configuration. To that end we make the following assumptions:

- All TF-A images are run from either ROM or on-chip trusted SRAM. This means TF-A is not vulnerable to an attacker that can probe or tamper with off-chip memory.
- Trusted boot is enabled. This means an attacker can't boot arbitrary images that are not approved by platform providers.
- There is no Secure-EL2. We don't consider threats that may come with Secure-EL2 software.
- There are no Root and Realm worlds. These are introduced by *Realm Management Extension (RME)*.

  The *Threat Model for TF-A with Arm CCA support* covers these types of configurations.
- No experimental features are enabled. We do not consider threats that may come from them.
- The platform's hardware complies with the PSR specification, defining the bare-minimum security pre-requisites for System-on-Chips (SoC).

Data Flow Diagram

Figure 1 shows a high-level data flow diagram for TF-A. The diagram shows a model of the different components of a TF-A-based system and their interactions with TF-A. A description of each diagram element is given on Table 1. On the diagram, the red broken lines indicate trust boundaries. Components outside of the broken lines are considered untrusted by TF-A.

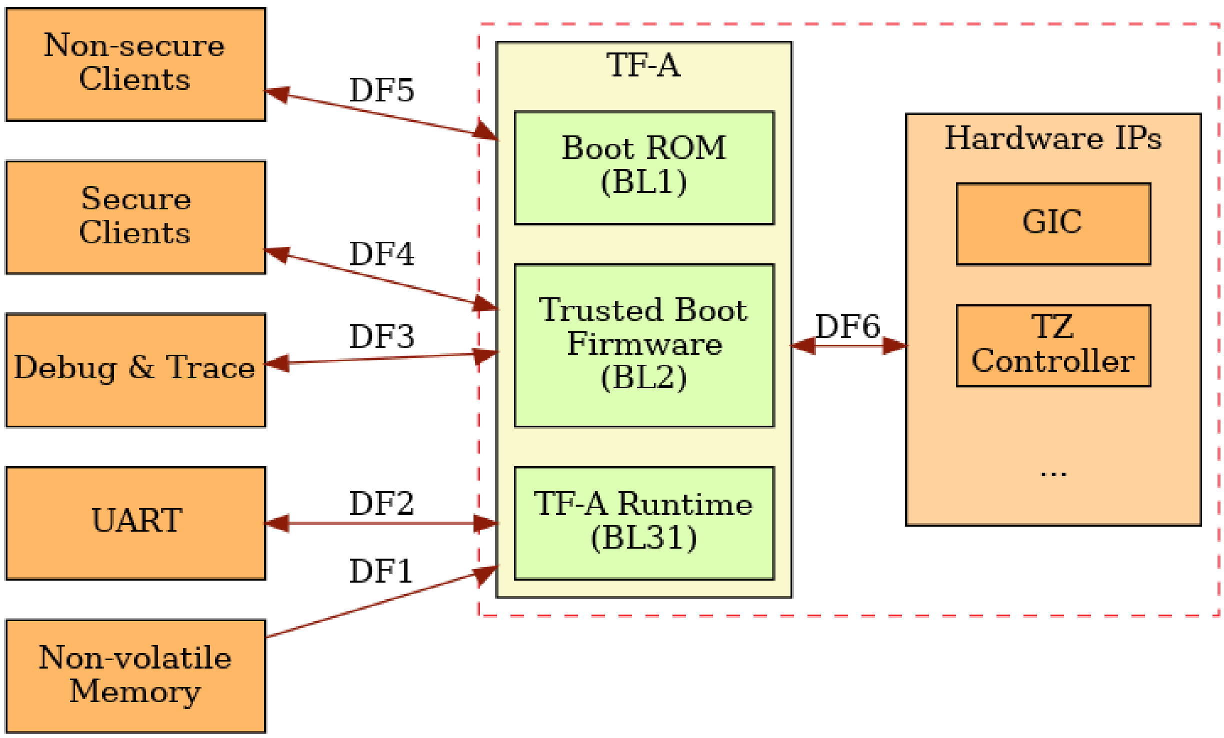


Fig. 1: Figure 1: TF-A Data Flow Diagram

Table 1: Table 1: TF-A Data Flow Diagram Description

Diagram Element	Description
DF1	At boot time, images are loaded from non-volatile memory and verified by TF-A boot firmware. These images include TF-A BL2 and BL31 images, as well as other secure and non-secure images.
DF2	TF-A log system framework outputs debug or informative messages over a UART interface. Also, characters can be read from a UART interface.
DF3	Debug and trace IP on a platform can allow access to registers and memory of TF-A.
DF4	Secure world software (e.g. trusted OS) interact with TF-A through SMC call interface and/or shared memory.
DF5	Non-secure world software (e.g. rich OS) interact

Threat Analysis

In this section we identify and provide assessment of potential threats to TF-A firmware. The threats are identified for each diagram element on the data flow diagram above.

For each threat, we identify the *asset* that is under threat, the *threat agent* and the *threat type*. Each threat is given a *risk rating* that represents the impact and likelihood of that threat. We also discuss potential mitigations.

Assets

We have identified the following assets for TF-A:

Table 2: Table 2: TF-A Assets

Asset	Description
Sensitive Data	These include sensitive data that an attacker must not be able to tamper with (e.g. the Root of Trust Public Key) or see (e.g. secure logs, debugging information such as crash reports).
Code Execution	This represents the requirement that the platform should run only TF-A code approved by the platform provider.
Availability	This represents the requirement that TF-A services should always be available for use.

Threat Agents

To understand the attack surface, it is important to identify potential attackers, i.e. attack entry points. The following threat agents are in scope of this threat model.

Table 3: Table 3: Threat Agents

Threat Agent	Description
NSCode	Malicious or faulty code running in the Non-secure world, including NS-EL0 NS-EL1 and NS-EL2 levels
SecCode	Malicious or faulty code running in the secure world, including S-EL0 and S-EL1 levels
AppDebug	Physical attacker using debug signals to access TF-A resources
PhysicalAccess	Physical attacker having access to external device communication bus and to external flash communication bus using common hardware

Note: In this threat model an advanced physical attacker that has the capability to tamper with a hardware (e.g. "rewiring" a chip using a focused ion beam (FIB) workstation or decapsulate the chip using chemicals) is considered out-of-scope.

Certain non-invasive physical attacks that do not need modifications to the chip, notably those like Power Analysis Attacks, are out-of-scope. Power analysis side-channel attacks represent a category of security threats that capitalize on information leakage through a device's power consumption during its normal operation. These attacks leverage the correlation between a device's power usage and its internal data processing activities. This correlation provides attackers with the means to extract sensitive information, including cryptographic keys.

Threat Types

In this threat model we categorize threats using the STRIDE threat analysis technique. In this technique a threat is categorized as one or more of these types: `Spoofing`, `Tampering`, `Repudiation`, `Information disclosure`, `Denial of service` or `Elevation of privilege`.

Threat Risk Ratings

For each threat identified, a risk rating that ranges from *informational* to *critical* is given based on the likelihood of the threat occurring if a mitigation is not in place, and the impact of the threat (i.e. how severe the consequences could be). Table 4 explains each rating in terms of score, impact and likelihood.

Table 4: Table 4: Rating and score as applied to impact and likelihood

Rating (Score)	Impact	Likelihood
Critical (5)	Extreme impact to entire organization if exploited.	Threat is almost certain to be exploited. Knowledge of the threat and how to exploit it are in the public domain.
High (4)	Major impact to entire organization or single line of business if exploited	Threat is relatively easy to detect and exploit by an attacker with little skill.
Medium (3)	Noticeable impact to line of business if exploited.	A knowledgeable insider or expert attacker could exploit the threat without much difficulty.
Low (2)	Minor damage if exploited or could be used in conjunction with other vulnerabilities to perform a more serious attack	Exploiting the threat would require considerable expertise and resources
Informational (1)	Poor programming practice or poor design decision that may not represent an immediate risk on its own, but may have security implications if multiplied and/or combined with other threats.	Threat is not likely to be exploited on its own, but may be used to gain information for launching another attack

Aggregate risk scores are assigned to identified threats; specifically, the impact score multiplied by the likelihood score. For example, a threat with high likelihood and low impact would have an aggregate risk score of eight (8); that is, four (4) for high likelihood multiplied by two (2) for low impact. The aggregate risk score determines the finding's overall risk level, as shown in the following table.

Table 5: Table 5: Overall risk levels and corresponding aggregate scores

Overall Risk Level	Aggregate Risk Score (Impact multiplied by Likelihood)
Critical	20–25
High	12–19
Medium	6–11
Low	2–5
Informational	1

The likelihood and impact of a threat depends on the target environment in which TF-A is running. For example, attacks that require physical access are unlikely in server environments while they are more common in Internet of Things(IoT) environments. In this threat model we consider three target environments: `Internet of Things(IoT)`, `Mobile` and `Server`.

Threat Assessment

The following threats were identified by applying STRIDE analysis on each diagram element of the data flow diagram.

For each threat, we strive to indicate whether the mitigations are currently implemented or not. However, the answer to this question is not always straight forward. Some mitigations are partially implemented in the generic code but also rely on the platform code to implement some bits of it. This threat model aims to be platform-independent and it is important to keep in mind that such threats only get mitigated if the platform code properly fulfills its responsibilities.

Also, some mitigations require enabling specific features, which must be explicitly turned on via a build flag.

When such conditions must be met, these are highlighted in the `Mitigations implemented?` box.

As our *Target of Evaluation* is made of several, distinct firmware images, some threats are confined in specific images, while others apply to each of them. To help developers implement mitigations in the right place, threats below are categorized based on the firmware image that should mitigate them.

General Threats for All Firmware Images

ID	05		
Threat	**Information leak via UART logs** During the development stages of software it is common to print all sorts of information on the console, including sensitive or confidential information such as crash reports with detailed information of the CPU state, current registers values, privilege level or stack dumps. This information is useful when debugging problems before releasing the production version but it could be used by an attacker to develop a working exploit if left enabled in the production version. This happens when directly logging sensitive information and more subtly when logging side-channel information that can be used by an attacker to learn about sensitive information.		
Diagram Elements	DF2		
Affected TF-A Components	BL1, BL2, BL31		
Assets	Sensitive Data		
Threat Agent	AppDebug		
Threat Type	Information Disclosure		
Application	Server	IoT	Mobile
Impact	N/A	Low (2)	Low (2)
Likelihood	N/A	High (4)	High (4)
Total Risk Rating	N/A	Medium (8)	Medium (8)
Mitigations	Remove sensitive information logging in production releases. Do not conditionally log information depending on potentially sensitive data. Do not log high precision timing information.		
Mitigations implemented?	Yes / Platform Specific. Requires the right build options to be used. Crash reporting is only enabled for debug builds by default, see `CRASH_REPORTING` build option. The log level can be tuned at build time, from very verbose to no output at all. See `LOG_LEVEL` build option. By default, release builds are a lot less verbose than debug ones but still produce some output. Messages produced by the platform code should use the appropriate level of		

ID	06		
Threat	**An attacker can read sensitive data and execute arbitrary code through the external debug and trace interface** Arm processors include hardware-assisted debug and trace features that can be controlled without the need for software operating on the platform. If left enabled without authentication, this feature can be used by an attacker to inspect and modify TF-A registers and memory allowing the attacker to read sensitive data and execute arbitrary code.		
Diagram Elements	DF3		
Affected TF-A Components	BL1, BL2, BL31		
Assets	Code Execution, Sensitive Data		
Threat Agent	AppDebug		
Threat Type	Tampering, Information Disclosure, Elevation of privilege		
Application	Server	IoT	Mobile
Impact	N/A	High (4)	High (4)
Likelihood	N/A	Critical (5)	Critical (5)
Total Risk Rating	N/A	Critical (20)	Critical (20)
Mitigations	Disable the debug and trace capability for production releases or enable proper debug authentication as recommended by [DEN0034].		
Mitigations implemented?	Platform specific. Configuration of debug and trace capabilities is entirely platform specific.		

ID	08		
Threat	**Memory corruption due to memory overflows and lack of boundary checking when accessing resources could allow an attacker to execute arbitrary code, modify some state variable to change the normal flow of the program, or leak sensitive information** Like in other software, TF-A has multiple points where memory corruption security errors can arise. Some of the errors include integer overflow, buffer overflow, incorrect array boundary checks, and incorrect error management. Improper use of asserts instead of proper input validations might also result in these kinds of errors in release builds.		
Diagram Elements	DF4, DF5		
Affected TF-A Components	BL1, BL2, BL31		
Assets	Code Execution, Sensitive Data		
Threat Agent	NSCode, SecCode		
Threat Type	Tampering, Information Disclosure, Elevation of Privilege		
Application	Server	IoT	Mobile
Impact	Critical (5)	Critical (5)	Critical (5)
Likelihood	Medium (3	Medium (3)	Medium (3)
Total Risk Rating	High (15)	High (15)	High (15)
Mitigations	1) Use proper input validation. 2) Code reviews, testing.		
Mitigations implemented?	1) Yes. Data received from normal world, such as addresses and sizes identifying memory regions, are sanitized before being used. These security checks make sure that the normal world software does not access memory beyond its limit. By default *asserts* are only used to check for programming errors in debug builds. Other types of errors are handled through condition checks that remain enabled in release builds. See TF-A error handling policy. TF-A provides an option to use *asserts* in release builds, however we recommend using proper runtime checks instead of relying on asserts in release builds. 2) Yes. TF-A uses a combination of manual code reviews and automated program analysis and testing to detect and fix memory corruption bugs. All TF-A code including platform code go through manual code reviews. Additionally, static code analysis is performed using Coverity Scan on all TF-A code. The code is also tested with Trusted Firmware-A Tests on Juno and FVP platforms.		

ID	11		
Threat	**Misconfiguration of the Memory Management Unit (MMU) may allow a normal world software to access sensitive data, execute arbitrary code or access otherwise restricted HW interface** A misconfiguration of the MMU could lead to an open door for software running in the normal world to access sensitive data or even execute code if the proper security mechanisms are not in place.		
Diagram Elements	DF5, DF6		
Affected TF-A Components	BL1, BL2, BL31		
Assets	Sensitive Data, Code execution		
Threat Agent	NSCode		
Threat Type	Information Disclosure, Elevation of Privilege		
Application	Server	IoT	Mobile
Impact	Critical (5)	Critical (5)	Critical (5)
Likelihood	High (4)	High (4)	High (4)
Total Risk Rating	Critical (20)	Critical (20)	Critical (20)
Mitigations	When configuring access permissions, the principle of least privilege ought to be enforced. This means we should not grant more privileges than strictly needed, e.g. code should be read-only executable, read-only data should be read-only execute-never, and so on.		
Mitigations implemented?	Platform specific. MMU configuration is platform specific, therefore platforms need to make sure that the correct attributes are assigned to memory regions. TF-A provides a library which abstracts the low-level details of MMU configuration. It provides well-defined and tested APIs. Platforms are encouraged to use it to limit the risk of misconfiguration.		

ID	13		
Threat	**Leaving sensitive information in the memory, can allow an attacker to retrieve them.** Accidentally leaving not-needed sensitive data in internal buffers can leak them if an attacker gains access to memory due to a vulnerability.		
Diagram Elements	DF4, DF5		
Affected TF-A Components	BL1, BL2, BL31		
Assets	Sensitive Data		
Threat Agent	NSCode, SecCode		
Threat Type	Information Disclosure		
Application	Server	IoT	Mobile
Impact	Critical (5)	Critical (5)	Critical (5)
Likelihood	Medium (3)	Medium (3)	Medium (3)
Total Risk Rating	High (15)	High (15)	High (15)
Mitigations	Clear the sensitive data from internal buffers as soon as they are not needed anymore.		
Mitigations implemented?	Yes / Platform specific		

ID	15		
Threat	**Improper handling of input data received over a UART interface may allow an attacker to tamper with TF-A execution environment.** The consequences of the attack depend on the the exact usage of input data received over UART. Examples are injection of arbitrary data, sensitive data tampering, influencing the execution path, denial of service (if using blocking I/O). This list may not be exhaustive.		
Diagram Elements	DF2, DF4, DF5		
Affected TF-A Components	BL1, BL2, BL31		
Assets	Sensitive Data, Code Execution, Availability		
Threat Agent	NSCode, SecCode		
Threat Type	Tampering, Information Disclosure, Denial of service, Elevation of privilege.		
Application	Server	IoT	Mobile
Impact	Critical (5)	Critical (5)	Critical (5)
Likelihood	Critical (5)	Critical (5)	Critical (5)
Total Risk Rating	Critical (25)	Critical (25)	Critical (25)
Mitigations	By default, the code to read input data from UART interfaces is disabled (see *ENABLE_CONSOLE_GETC* build option). It should only be enabled on a need basis. Data received over UART interfaces should be treated as untrusted data. As such, it should be properly sanitized and handled with caution.		
Mitigations implemented?	Platform specific. Generic code does not read any input data from UART interface(s).		

ID	16		
Threat	**An attacker could analyse the timing behaviour of implemented methods in the system to infer sensitive information.** A timing side-channel attack is a type of attack that exploits variations in the time it takes a system to perform different operations. This form of attack focuses on analyzing the time- related information leakage that occurs during the execution of cryptographic algorithms or other security-sensitive processes. By observing these timing differences, an attacker can gain insights into the internal workings of a system and potentially extract sensitive information. Sensitive information that, when revealed even partially, could heighten the susceptibility to traditional attacks like brute-force attacks.		
Diagram Elements	DF2		
Affected TF-A Components	BL1, BL2, BL31		
Assets	Sensitive Data		
Threat Agent	AppDebug		
Threat Type	Information Disclosure		
Application	Server	IoT	Mobile
Impact	Critical (5)	Critical (5)	Critical (5)
Likelihood	Critical (5)	Critical (5)	Critical (5)
Total Risk Rating	Critical (25)	Critical (25)	Critical (25)
Mitigations	Ensure that the execution time of critical operations is constant and independent of secret data. This prevents attackers from exploiting timing differences to infer information about sensitive data. Introduce random delays/timing jitter or dummy operations to make the timing behavior of program execution less predictable. This can disrupt the correlation between the execution time and sensitive data.		
Mitigations implemented?	Not implemented		

Threats to be Mitigated by the Boot Firmware

The boot firmware here refers to the boot ROM (BL1) and the trusted boot firmware (BL2). Typically it does not stay resident in memory and it is dismissed once execution has reached the runtime EL3 firmware (BL31). Thus, past that point in time, the threats below can no longer be exploited.

Note, however, that this is not necessarily true on all platforms. Platform vendors should review these threats to make sure they cannot be exploited nonetheless once execution has reached the runtime EL3 firmware.

Boot firmware threats that only affect availability, for example persistent failure to boot due to corruption of the firmware images on flash, are out of scope of this generic threat model. Such threats may be in scope for individual platforms, but those platforms are responsible for mitigating such threats, for example by protecting against physical access to flash or implementing firmware recovery mechanisms.

ID	01		
Threat	**An attacker can mangle firmware images to execute arbitrary code** Some TF-A images are loaded from external storage. It is possible for an attacker to access the external flash memory and change its contents physically, through the Rich OS, or using the updating mechanism to modify the non-volatile images to execute arbitrary code.		
Diagram Elements	DF1, DF4, DF5		
Affected TF-A Components	BL2, BL31		
Assets	Code Execution		
Threat Agent	PhysicalAccess, NSCode, SecCode		
Threat Type	Tampering, Elevation of Privilege		
Application	Server	IoT	Mobile
Impact	Critical (5)	Critical (5)	Critical (5)
Likelihood	Critical (5)	Critical (5)	Critical (5)
Total Risk Rating	Critical (25)	Critical (25)	Critical (25)
Mitigations	1) Implement the Trusted Board Boot (TBB) feature which prevents malicious firmware from running on the platform by authenticating all firmware images. 2) Perform extra checks on unauthenticated data, such as FIP metadata, prior to use.		
Mitigations implemented?	1) Yes, provided that the `TRUSTED_BOARD_BOOT` build option is set to 1. 2) Yes.		

ID	02		
Threat	**An attacker may attempt to boot outdated, potentially vulnerable firmware image** When updating firmware, an attacker may attempt to rollback to an older version that has unfixed vulnerabilities.		
Diagram Elements	DF1, DF4, DF5		
Affected TF-A Components	BL2, BL31		
Assets	Code Execution		
Threat Agent	PhysicalAccess, NSCode, SecCode		
Threat Type	Tampering		
Application	Server	IoT	Mobile
Impact	Critical (5)	Critical (5)	Critical (5)
Likelihood	Critical (5)	Critical (5)	Critical (5)
Total Risk Rating	Critical (25)	Critical (25)	Critical (25)
Mitigations	Implement anti-rollback protection using non-volatile counters (NV counters) as required by TBBR-Client specification.		
Mitigations implemented?	Yes / Platform specific. After a firmware image is validated, the image revision number taken from a certificate extension field is compared with the corresponding NV counter stored in hardware to make sure the new counter value is larger than the current counter value. **Platforms must implement this protection using platform specific hardware NV counters.**		

ID	03		
Threat	**An attacker can use Time-of-Check-Time-of-Use (TOCTOU) attack to bypass image authentication during the boot process** Time-of-Check-Time-of-Use (TOCTOU) threats occur when the security check is produced before the time the resource is accessed. If an attacker is sitting in the middle of the off-chip images, they could change the binary containing executable code right after the integrity and authentication check has been performed.		
Diagram Elements	DF1		
Affected TF-A Components	BL1, BL2		
Assets	Code Execution, Sensitive Data		
Threat Agent	PhysicalAccess		
Threat Type	Elevation of Privilege		
Application	Server	IoT	Mobile
Impact	N/A	Critical (5)	Critical (5)
Likelihood	N/A	Medium (3)	Medium (3)
Total Risk Rating	N/A	High (15)	High (15)
Mitigations	Copy image to on-chip memory before authenticating it.		
Mitigations implemented?	Platform specific. The list of images to load and their location is platform specific. Platforms are responsible for arranging images to be loaded in on-chip memory.		

ID	04		
Threat	**An attacker with physical access can execute arbitrary image by bypassing the signature verification stage using glitching techniques** Glitching (Fault injection) attacks attempt to put a hardware into a undefined state by manipulating an environmental variable such as power supply. TF-A relies on a chain of trust that starts with the ROTPK, which is the key stored inside the chip and the root of all validation processes. If an attacker can break this chain of trust, they could execute arbitrary code on the device. This could be achieved with physical access to the device by attacking the normal execution flow of the process using glitching techniques that target points where the image is validated against the signature.		
Diagram Elements	DF1		
Affected TF-A Components	BL1, BL2		
Assets	Code Execution		
Threat Agent	PhysicalAccess		
Threat Type	Tampering, Elevation of Privilege		
Application	Server	IoT	Mobile
Impact	N/A	Critical (5)	Critical (5)
Likelihood	N/A	Medium (3)	Medium (3)
Total Risk Rating	N/A	High (15)	High (15)
Mitigations	Mechanisms to detect clock glitch and power variations.		
Mitigations implemented?	No. The most effective mitigation is adding glitching detection and mitigation circuit at the hardware level. However, software techniques, such as adding redundant checks when performing conditional branches that are security sensitive, can be used to harden TF-A against such attacks. **At the moment TF-A doesn't implement such mitigations.**		

Measured Boot Threats (or lack of)

In the current Measured Boot design the following components form the *TCB*:

- BL1, BL2, BL31

- Secure world components
- RMM (if RME extension is implemented)
- The configuration data of the above components

Across various Measured Boot backends, the data recorded during the flow as well as the criticality of this data can vary. In most cases, these attributes are considered valuable assets and are protected against potential attacks:

- Image measurement: the digest value of a component produced by a hash function.
- Signer-id: the digest value of the image verification publiy key. The verification public key is part of the image metadata.

In addition to these, other metadata attributes (image version, hash algorithm identifier, etc) could be recorded during the Measured Boot process. But these are not critical data.

In this context, an attack means modifying the measurement data (image or public key hash) or recording arbitrary data as valid measurements.

The current Measured Boot design consists of two main parts. A frontend, which is responsible for taking the measurements, and a backend which is responsible for storing them. *TF-A* makes it possible to integrate various backends. Some of these are implemented by the *TF-A* projects, while others are part of different projects, and *TF-A* provides an integration layer.

- TCG-compliant Event Log: Implemented by *TF-A*. Measurements are stored in the Event Log which is located on the secure on-chip memory of the AP. The address of the Event Log buffer is handed off between boot stages and new measurements are appended to the Event Log. A limitation of the current Measured Boot implementation in *TF-A* is that it does not extend the measurements into a *PCR* of a Discrete *TPM*, where measurements would be securely stored and protected against tampering.
- Discrete *TPM*: Implemented in *TF-A* as a proof of concept, the Discrete *TPM* is used alongside the existing TCG-compliant Event Log. This Measured Boot implementation extends measurement hashes to a *PCR* in the *TPM*, which provides a hardware-backed root of trust. The measurements in the Event Log can now be hashed and compared to the value of the *PCR* to determine if tampering of the Event Log has taken place.
- CCA Measured Boot: Implemented by *TF-M*. Measurements are stored in *HES* secure on-chip memory. *HES* implements protection against tampering its on-chip memory. *HES* interface is available for BL1 and BL2.
- DICE Protection Environment (DPE): Implemented by *TF-M*. Measurements are stored in *RSE* secure on-chip memory. *RSE* implements protection against tampering its on-chip memory. DPE provides additional protection against unauthorized access by malicious actors through the use of one-time context handles and the identification of the client's target locality (location of the client).

Beyond the measurements (image digest and signer-id) there are no other assets to protect or threats to defend against that could compromise *TF-A* execution environment's security.

When considering the implementation of Measured Boot using a TCG-compliant Event Log backed by a discrete TPM, physical vulnerabilities come to mind. Platforms have many different ways of integrating a discrete TPM, and these implementations can be susceptible to man-in-the-middle attacks, where the attacker intercepts the bus traffic between the discrete TPM and the host machine. This can lead to PCR extend operations being modified, compromising Measured Boot. This vulnerability requires physical access to the host machine.

TF-A does not provide any mitigations against these physical vulnerabilities, it is the responsibility of the platform owners to address this based on their specific threat model. Mitigation of this can be achieved through dedicated hardware solutions, such as an encrypted AP/dTPM bus, or software-based approaches designed to protect sensitive data such as parameter encryption.

There are general security assets and threats associated with remote/delegated attestation. However, these are outside the *TF-A* security boundary and should be dealt with by the appropriate agent in the platform/system. Since current Measured Boot design does not use local attestation, there would be no further assets to protect (like unsealed keys).

System integrators must carefully evaluate the security requirement and capabilities of their platform and choose an appropriate Measured Boot solution.

Threats to be Mitigated by the Runtime EL3 Firmware

ID	07		
Threat	**An attacker can perform a denial-of-service attack by using a broken SMC call that causes the system to reboot or enter into unknown state.** Secure and non-secure clients access TF-A services through SMC calls. Malicious code can attempt to place the TF-A runtime into an inconsistent state by calling unimplemented SMC call or by passing invalid arguments.		
Diagram Elements	DF4, DF5		
Affected TF-A Components	BL31		
Assets	Availability		
Threat Agent	NSCode, SecCode		
Threat Type	Denial of Service		
Application	Server	IoT	Mobile
Impact	Medium (3)	Medium (3)	Medium (3)
Likelihood	High (4)	High (4)	High (4)
Total Risk Rating	High (12)	High (12)	High (12)
Mitigations	Validate SMC function ids and arguments before using them.		
Mitigations implemented?	Yes / Platform specific. For standard services, all input is validated. Platforms that implement SiP services must also validate SMC call arguments.		

ID	09		
Threat	**Improperly handled SMC calls can leak register contents** When switching between worlds, TF-A register state can leak to software in different security contexts.		
Diagram Elements	DF4, DF5		
Affected TF-A Components	BL31		
Assets	Availability, Sensitive data		
Threat Agent	NSCode, SecCode		
Threat Type	Information disclosure, Denial of Service		
Application	Server	IoT	Mobile
Impact	Medium (3)	Medium (3)	Medium (3)
Likelihood	High (4)	High (4)	High (4)
Total Risk Rating	High (12)	High (12)	High (12)
Mitigations	Save and restore registers when switching contexts.		
Mitigations implemented?	Yes / Platform specific. FEATURE_DETECTION is provided to match firmware to hardware. This is the default behaviour in TF-A. Build options are also provided to save/restore additional registers such as floating-point registers. These should be enabled if required.		

ID	10		
Threat	**SMC calls can leak sensitive information from TF-A memory via microarchitectural side channels** Microarchitectural side-channel attacks such as Spectre can be used to leak data across security boundaries. An attacker might attempt to use this kind of attack to leak sensitive data from TF-A memory.		
Diagram Elements	DF4, DF5		
Affected TF-A Components	BL31		
Assets	Sensitive Data		
Threat Agent	SecCode, NSCode		
Threat Type	Information Disclosure		
Application	Server	IoT	Mobile
Impact	Medium (3)	Medium (3)	Medium (3)
Likelihood	Medium (3)	Medium (3)	Medium (3)
Total Risk Rating	Medium (9)	Medium (9)	Medium (9)
Mitigations	Enable appropriate side-channel protections.		
Mitigations implemented?	Yes / Platform specific. TF-A implements software mitigations for Spectre type attacks as recommended by Cache Speculation Side-channels for the generic code. SiPs should implement similar mitigations for code that is deemed to be vulnerable to such attacks.		

ID	12		
Threat	**Incorrect configuration of Performance Monitor Unit (PMU) counters can allow an attacker to mount side-channel attacks using information exposed by the counters** Non-secure software can configure PMU registers to count events at any exception level and in both Secure and Non-secure states. This allows a Non-secure software (or a lower-level Secure software) to potentially carry out side-channel timing attacks against TF-A.		
Diagram Elements	DF5, DF6		
Affected TF-A Components	BL31		
Assets	Sensitive Data		
Threat Agent	NSCode		
Threat Type	Information Disclosure		
Application	Server	IoT	Mobile
Impact	Medium (3)	Medium (3)	Medium (3)
Likelihood	Low (2)	Low (2)	Low (2)
Total Risk Rating	Medium (6)	Medium (6)	Medium (6)
Mitigations	Follow mitigation strategies as described in Secure Development Guidelines.		
Mitigations implemented?	Yes / platform specific. General events and cycle counting in the Secure world is prohibited by default when applicable. However, on some implementations (e.g. PMUv3) Secure world event counting depends on external debug interface signals, i.e. Secure world event counting is enabled if external debug is enabled. Configuration of debug signals is platform specific, therefore platforms need to make sure that external debug is disabled in production or proper debug authentication is in place. This should be the case if threat #06 is properly mitigated.		

Threats to be Mitigated by an External Agent Outside of TF-A

ID	14		
Threat	**Attacker wants to execute an arbitrary or untrusted binary as the secure OS.** When the option OPTEE_ALLOW_SMC_LOAD is enabled, this trusts the non-secure world up until the point it issues the SMC call to load the Secure BL32 payload. If a compromise occurs before the SMC call is invoked, then arbitrary code execution in S-EL1 can occur or arbitrary memory in EL3 can be overwritten.		
Diagram Elements	DF5		
Affected TF-A Components	BL31, BL32		
Assets	Code Execution, Sensitive Data		
Threat Agent	NSCode		
Threat Type	Tampering, Information Disclosure, Elevation of privilege		
Application	Server	IoT	Mobile
Impact	Critical (5)	Critical (5)	Critical (5)
Likelihood	High (4)	High (4)	High (4)
Total Risk Rating	Critical (20)	Critical (20)	Critical (20)
Mitigations	When enabling the option OPTEE_ALLOW_SMC_LOAD, the non-secure OS must be considered a closed platform up until the point the SMC can be invoked to load OP-TEE.		
Mitigations implemented?	None in TF-A itself. This option is only used by ChromeOS currently which has other mechanisms to to mitigate this threat which are described in OP-TEE Dispatcher.		

11.1.2 EL3 SPMC Threat Model

Introduction

This document provides a threat model for the TF-A *EL3 Secure Partition Manager* (EL3 SPM) implementation. The EL3 SPM implementation is based on the Arm Firmware Framework for Arm A-profile specification.

Target of Evaluation

In this threat model, the target of evaluation is the `Secure Partition Manager Core` component (SPMC) within the EL3 firmware. The monitor and SPMD at EL3 are covered by the *Generic TF-A threat model*.

The scope for this threat model is:

- The TF-A implementation for the EL3 SPMC
- The implementation complies with the FF-A v1.1 specification.
- Secure partition is statically provisioned at boot time.
- Focus on the run-time part of the life-cycle (no specific emphasis on boot time, factory firmware provisioning, firmware udpate etc.)
- Not covering advanced or invasive physical attacks such as decapsulation, FIB etc.

Data Flow Diagram

Figure 1 shows a high-level data flow diagram for the SPM split into an SPMD and SPMC component at EL3. The SPMD mostly acts as a relayer/pass-through between the normal world and the secure world. It is assumed to expose small attack surface.

A description of each diagram element is given in Table 1. In the diagram, the red broken lines indicate trust boundaries.

Components outside of the broken lines are considered untrusted.

Table 6: Table 1: EL3 SPMC Data Flow Diagram Description

Diagram Element	Description
DF1	SP to SPMC communication. FF-A function invocation or implementation-defined Hypervisor call. Note:- To communicate with LSP, SP1 performs a direct message request to SPMC targeting LSP as destination.
DF2	SPMC to SPMD communication.
DF3	SPMD to NS forwarding.
DF4	SPMC to LSP communication. NWd to LSP communication happens through SPMC. LSP can send direct response SP1 or NWd through SPMC.
DF5	HW control.
DF6	Bootloader image loading.
DF7	External memory access.

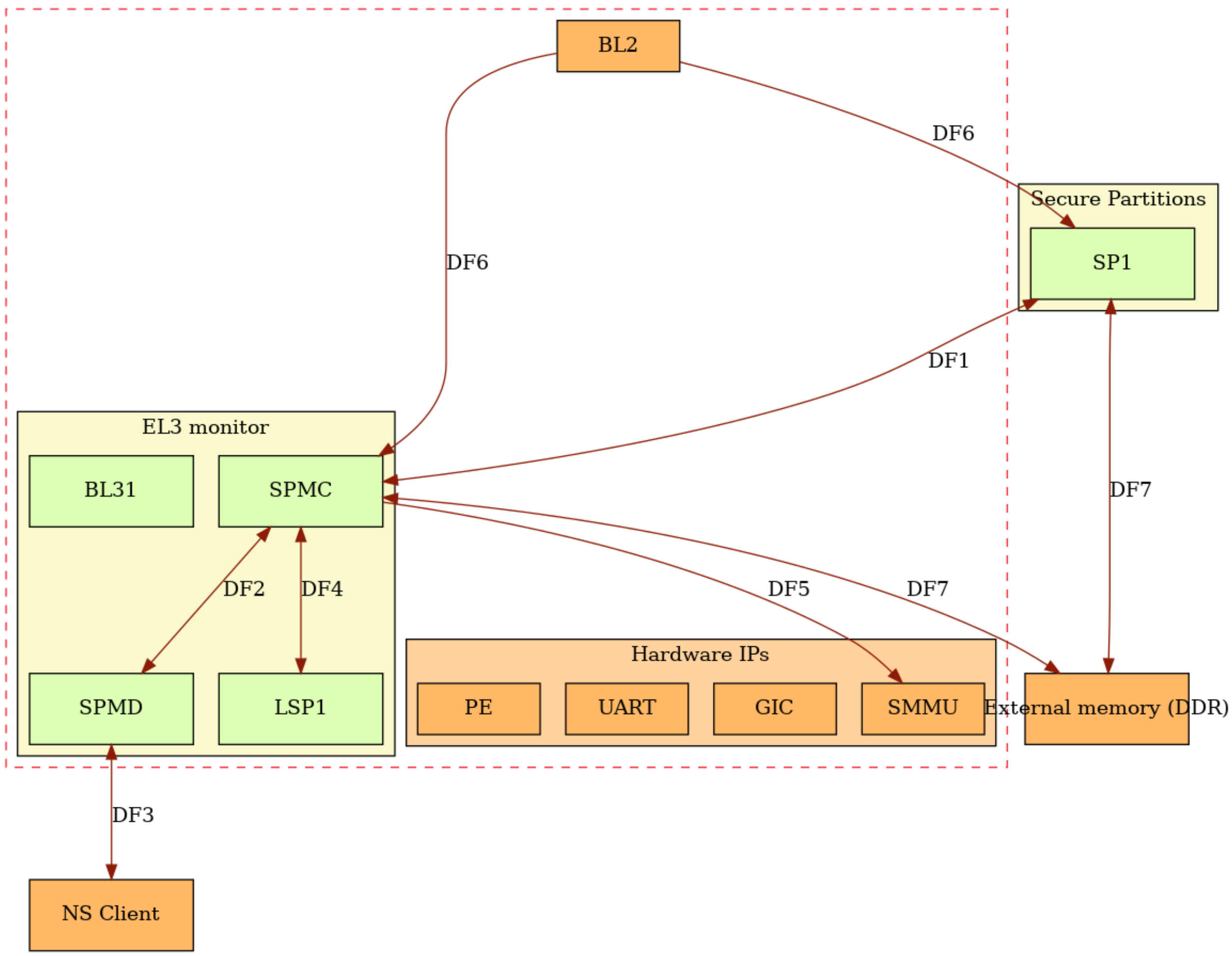


Fig. 2: Figure 1: EL3 SPMC Data Flow Diagram

Threat Analysis

This threat model follows a similar methodology to the *Generic TF-A threat model*. The following sections define:

- Trust boundaries
- Assets
- Theat agents
- Threat types

Trust boundaries

- Normal world is untrusted.
- Secure world and normal world are separate trust boundaries.
- EL3 monitor, SPMD and SPMC are trusted.
- Bootloaders (in particular BL1/BL2 if using TF-A) and run-time BL31 are implicitely trusted by the usage of trusted boot.
- EL3 monitor, SPMD, SPMC do not trust SPs.

Assets

The following assets are identified:

- SPMC state.
- SP state.
- Information exchange between endpoints (partition messages).
- SPMC secrets (e.g. pointer authentication key when enabled)
- SP secrets (e.g. application keys).
- Scheduling cycles.
- Shared memory.

Threat Agents

The following threat agents are identified:

- Non-secure endpoint (referred NS-Endpoint later): normal world client at NS-EL2 (Hypervisor) or NS-EL1 (VM or OS kernel).
- Secure endpoint (referred as S-Endpoint later): typically a secure partition.
- Hardware attacks (non-invasive) requiring a physical access to the device, such as bus probing or DRAM stress.

Threat types

The following threat categories as exposed in the *Generic TF-A threat model* are re-used:

- Spoofing
- Tampering
- Repudiation
- Information disclosure
- Denial of service
- Elevation of privileges

Similarly this threat model re-uses the same threat risk ratings. The risk analysis is evaluated based on the environment being `Server` or `Mobile`. IOT is not evaluated as the EL3 SPMC is primarily meant for use in Client.

Threat Assessment

The following threats are identified by applying STRIDE analysis on each diagram element of the data flow diagram.

ID	01	
Threat	**An endpoint impersonates the sender FF-A ID in a direct request/response invocation.**	
Diagram Elements	DF1, DF2, DF3, DF4	
Affected TF-A Components	SPMD, SPMC	
Assets	SP state	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Spoofing	
Application	Server	Mobile
Impact	Critical(5)	Critical(5)
Likelihood	Critical(5)	Critical(5)
Total Risk Rating	Critical(25)	Critical(25)
Mitigations	SPMC must be able to correctly identify an endpoint and enforce checks to disallow spoofing.	
Mitigations implemented?	Yes. The SPMC enforces checks in the direct message request/response interfaces such an endpoint cannot spoof the origin and destination worlds (e.g. a NWd originated message directed to the SWd cannot use a SWd ID as the sender ID). Also enforces check for direct response being sent only to originator of request.	

ID	02	
Threat	**An endpoint impersonates the receiver FF-A ID in a direct request/response invocation.**	
Diagram Elements	DF1, DF2, DF3, DF4	
Affected TF-A Components	SPMD, SPMC	
Assets	SP state	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Spoofing, Denial of Service	
Application	Server	Mobile
Impact	Critical(5)	Critical(5)
Likelihood	Critical(5)	Critical(5)
Total Risk Rating	Critical(25)	Critical(25)
Mitigations	Validate if endpoind has permission to send request to other endpoint by implementation defined means.	
Mitigations implemented?	Platform specific. The guidance below is left for a system integrator to implement as necessary. Additionally a software component residing in the SPMC can be added for the purpose of direct request/response filtering. It can be configured with the list of known IDs and about which interaction can occur between one and another endpoint (e.g. which NWd endpoint ID sends a direct request to which SWd endpoint ID). This component checks the sender/receiver fields for a legitimate communication between endpoints. A similar component can exist in the OS kernel driver, or Hypervisor although it remains untrusted by the SPMD/SPMC.	

ID	03	
Threat	**Tampering with memory shared between an endpoint and the SPMC.** A malicious endpoint may attempt tampering with its RX/TX buffer contents while the SPMC is processing it (TOCTOU).	
Diagram Elements	DF1, DF3, DF7	
Affected TF-A Components	SPMC	
Assets	Shared memory, Information exchange	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Tampering	
Application	Server	Mobile
Impact	High (4)	High (4)
Likelihood	High (4)	High (4)
Total Risk Rating	High (16)	High (16)
Mitigations	Validate all inputs, copy before use.	
Mitigations implemented?	Yes. In context of FF-A v1.1 this is the case of sharing the RX/TX buffer pair and usage in the PARTITION_INFO_GET or memory sharing primitives. The SPMC copies the contents of the TX buffer to an internal temporary buffer before processing its contents. The SPMC implements hardened input validation on data transmitted through the TX buffer by an untrusted endpoint. The TF-A SPMC enforces checks on data transmitted through RX/TX buffers.	

ID	04	
Threat	**An endpoint may tamper with its own state or the state of another endpoint.** A malicious endpoint may attempt violating: • its own or another SP state by using an unusual combination (or out-of-order) FF-A function invocations. This can also be an endpoint emitting FF-A function invocations to another endpoint while the latter in not in a state to receive it (e.g. SP sends a direct request to the normal world early while the normal world is not booted yet). • the SPMC state itself by employing unexpected transitions in FF-A memory sharing, direct requests and responses, or handling of interrupts This can be led by random stimuli injection or fuzzing.	
Diagram Elements	DF1, DF2, DF3	
Affected TF-A Components	SPMD, SPMC	
Assets	SP state, SPMC state	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Tampering	
Application	Server	Mobile
Impact	High (4)	High (4)
Likelihood	Medium (3)	Medium (3)
Total Risk Rating	High (12) | High (12)	
Mitigations	Follow guidelines in FF-A v1.1 specification on state transitions (run-time model).	
Mitigations implemented?	Yes. The TF-A SPMC is hardened to follow this guidance.	

ID	05	
Threat	**Replay fragments of past communication between endpoints.** A malicious endpoint may replay a message exchange that occurred between two legitimate endpoints as a matter of triggering a malfunction or extracting secrets from the receiving endpoint. In particular the memory sharing operation with fragmented messages between an endpoint and the SPMC may be replayed by a malicious agent as a matter of getting access or gaining permissions to a memory region which does not belong to this agent.	
Diagram Elements	DF2, DF3	
Affected TF-A Components	SPMC	
Assets	Information exchange	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Repudiation	
Application	Server	Mobile
Impact	Medium (3)	Medium (3)
Likelihood	High (4)	High (4)
Total Risk Rating	High (12)	High (12)
Mitigations	Strict input validation and state tracking.	
Mitigations implemented?	Platform specific.	

ID	06	
Threat	**A malicious endpoint may attempt to extract data or state information by the use of invalid or incorrect input arguments.** Lack of input parameter validation or side effects of maliciously forged input parameters might affect the SPMC.	
Diagram Elements	DF1, DF2, DF3	
Affected TF-A Components	SPMD, SPMC	
Assets	SP secrets, SPMC secrets, SP state, SPMC state	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Information discolure	
Application	Server	Mobile
Impact	High (4)	High (4)
Likelihood	Medium (3)	Medium (3)
Total Risk Rating	High (12)	High (12)
Mitigations	SPMC must be prepared to receive incorrect input data from secure partitions and reject them appropriately. The use of software (canaries) or hardware hardening techniques (XN, WXN, pointer authentication) helps detecting and stopping an exploitation early.	
Mitigations implemented?	Yes. The TF-A SPMC mitigates this threat by implementing stack protector, pointer authentication, XN, WXN, security hardening techniques.	

ID	07	
Threat	**A malicious endpoint may forge a direct message request such that it reveals the internal state of another endpoint through the direct message response.** The secure partition or SPMC replies to a partition message by a direct message response with information which may reveal its internal state (e.g. partition message response outside of allowed bounds).	
Diagram Elements	DF1, DF2, DF3	
Affected TF-A Components	SPMC	
Assets	SPMC or SP state	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Information discolure	
Application	Server	Mobile
Impact	Medium (3)	Medium (3)
Likelihood	Low (2)	Low (2)
Total Risk Rating	Medium (6)	Medium (6)
Mitigations	Follow FF-A specification about state transitions, run time model, do input validation.	
Mitigations implemented?	Yes. For the specific case of direct requests targeting the SPMC, the latter is hardened to prevent its internal state or the state of an SP to be revealed through a direct message response. Further FF-A v1.1 guidance about run time models and partition states is followed.	

ID	08	
Threat	**Probing the FF-A communication between endpoints.** SPMC and SPs are typically loaded to external memory (protected by a TrustZone memory controller). A malicious agent may use non invasive methods to probe the external memory bus and extract the traffic between an SP and the SPMC or among SPs when shared buffers are held in external memory.	
Diagram Elements	DF7	
Affected TF-A Components	SPMC	
Assets	SP/SPMC state, SP/SPMC secrets	
Threat Agent	Hardware attack	
Threat Type	Information disclosure	
Application	Server	Mobile
Impact	Medium (3)	Medium (3)
Likelihood	Low (2)	Medium (3)
Total Risk Rating	Medium (6)	Medium (9)
Mitigations	Implement DRAM protection techniques using hardware countermeasures at platform or chip level.	
Mitigations implemented?	Platform specific.	

ID	09	
Threat	**A malicious agent may attempt revealing the SPMC state or secrets by the use of software-based cache side-channel attack techniques.**	
Diagram Elements	DF7	
Affected TF-A Components	SPMC	
Assets	SP or SPMC state	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Information disclosure	
Application	Server	Mobile
Impact	Medium (3)	Medium (3)
Likelihood	Low (2)	Low (2)
Total Risk Rating	Medium (6)	Medium (6)
Mitigations	The SPMC may be hardened further with SW mitigations (e.g. speculation barriers) for the cases not covered in HW. Usage of hardened compilers and appropriate options, code inspection are recommended ways to mitigate Spectre types of attacks.	
Mitigations implemented?	No.	

ID	10	
Threat	**A malicious endpoint may attempt flooding the SPMC with requests targeting a service within an endpoint such that it denies another endpoint to access this service.** Similarly, the malicious endpoint may target a a service within an endpoint such that the latter is unable to request services from another endpoint.	
Diagram Elements	DF1, DF2, DF3	
Affected TF-A Components	SPMC	
Assets	SPMC state, Scheduling cycles	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Denial of service	
Application	Server	Mobile
Impact	Medium (3)	Medium (3)
Likelihood	Medium (3)	Medium (3)
Total Risk Rating	Medium (9)	Medium (9)
Mitigations	Bounding the time for operations to complete can be achieved by the usage of a trusted watchdog. Other quality of service monitoring can be achieved in the SPMC such as counting a number of operations in a limited timeframe.	
Mitigations implemented?	Platform specific.	

ID	11	
Threat	**Denying a lender endpoint to make progress if borrower endpoint encountered a fatal exception. Denying a new sender endpoint to make progress if receiver encountered a fatal exception.**	
Diagram Elements	DF1, DF2, DF3	
Affected TF-A Components	SPMC	
Assets	Shared resources, Scheduling cycles.	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Denial of service	
Application	Server	Mobile
Impact	Medium (3)	Medium (3)
Likelihood	Medium (3)	Medium (3)
Total Risk Rating	Medium (9)	Medium (9)
Mitigations	SPMC must be able to detect fatal error in SP and take ownership of shared resources. It should be able to relinquish the access to shared memory regions to allow lender to proceed. SPMC must return ABORTED if new direct requests are targeted to SP which has had a fatal error.	
Mitigations implemented?	Platform specific.	

ID	12	
Threat	**A malicious endpoint may attempt to donate, share, lend, relinquish or reclaim unauthorized memory region.**	
Diagram Elements	DF1, DF2, DF3	
Affected TF-A Components	SPMC	
Assets	SP secrets, SPMC secrets, SP state, SPMC state	
Threat Agent	NS-Endpoint, S-Endpoint	
Threat Type	Elevation of Privilege	
Application	Server	Mobile
Impact	High (4)	High (4)
Likelihood	High (4)	High (4)
Total Risk Rating	High (16)	High (16)
Mitigations	Follow FF-A specification guidelines on Memory management transactions.	
Mitigations implemented?	Yes. The SPMC tracks ownership and access state for memory transactions appropriately, and validating the same for all operations. SPMC follows FF-A v1.1 guidance for memory transaction lifecycle.	

11.1.3 Threat Model for RSE - AP interface

Introduction

This document is an extension for the general TF-A threat-model. It considers those platforms where a Runtime Security Engine (RSE) is included in the SoC next to the Application Processor (AP).

Target of Evaluation

The scope of this threat model only includes the interface between the RSE and AP. Otherwise, the TF-A *Generic Threat Model* document is applicable for the AP core. The threat model for the RSE firmware will be provided by the RSE firmware project in the future.

Data Flow Diagram

This diagram is different only from the general TF-A data flow diagram in that it includes the RSE and highlights the interface between the AP and the RSE cores. The interface description only focuses on the AP-RSE interface the rest is the same as in the general TF-A threat-model document.

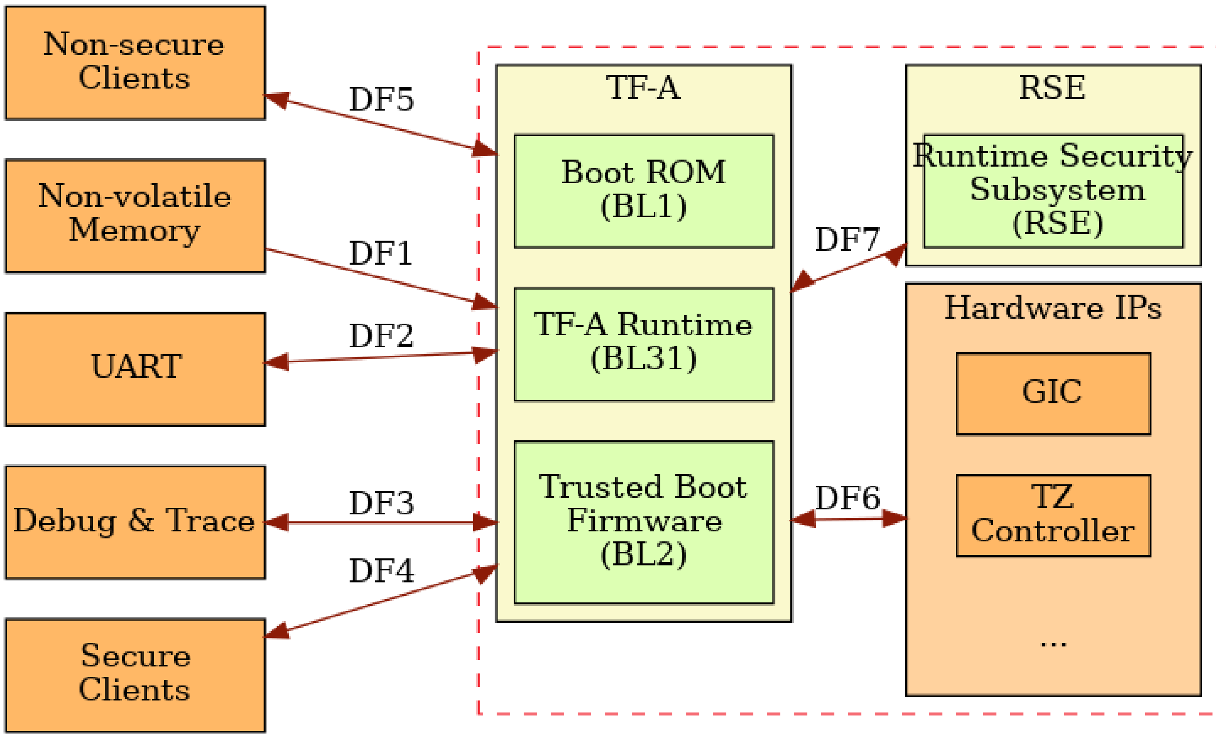


Fig. 3: Figure 1: TF-A Data Flow Diagram including RSE

Table 7: Table 1: TF-A - RSE data flow diagram

Diagram Element	Description
DF7	Boot images interact with RSE over a communication channel to record boot measurements and get image verification keys. At runtime, BL31 obtains the realm world attestation signing key from RSE.

Threat Assessment

For this section, please reference the Threat Assessment under the general TF-A threat-model document, *Generic Threat Model*. All the threats listed there are applicable for the AP core, here only the differences are highlighted.

- ID 11: The access to the communication interface between AP and RSE is allowed only for firmware running at EL3. Accidentally exposing this interface to NSCode can allow malicious code to interact with RSE and gain access to sensitive data.
- ID 13: Relevant in the context of the realm attestation key, which can be retrieved by BL31 through DF7. The RSE communication protocol layer mitigates against this by clearing its internal buffer when reply is received. The caller of the API must do the same if data is not needed anymore.

11.1.4 Threat Model for TF-A with Arm CCA support

Introduction

This document provides a threat model of TF-A firmware for platforms with Arm Realm Management Extension (RME) support which implement Arm Confidential Compute Architecture (Arm CCA).

Although it is a separate document, it references the *Generic Threat Model* in a number of places, as some of the contents is commonly applicable to TF-A with or without Arm CCA support.

Target of Evaluation

In this threat model, the target of evaluation is the Trusted Firmware for A-class Processors (TF-A) with RME support and Arm CCA support. This includes the boot ROM (BL1), the trusted boot firmware (BL2) and the runtime EL3 firmware (BL31).

Assumptions

We make the following assumptions:

- *Realm Management Extension (RME)* is enabled on the platform.
- Arm CCA Hardware Enforced Security (HES) is available on the platform, as recommended by Arm CCA security model:

  > *[R0004] Arm strongly recommends that all implementations of CCA utilize hardware enforced security (CCA HES).*

- All TF-A images run from on-chip memory. Data used by these images also live in on-chip memory. This means TF-A is not vulnerable to an attacker that can probe or tamper with off-chip memory.

  These are requirements of the Arm CCA security model:

> *[R0147] Monitor code executes entirely from on-chip memory.*
>
> *[R0149] Any monitor data that may affect the CCA security guarantee, other than GPT, is either held in on-chip memory, or in external memory but with additional integrity protection.*

Note that this threat model hardens *[R0149]* requirement by forbidding to hold data in external memory, even if it is integrity-protected - except for GPT data.

- TF-A BL1 image is immutable and thus implicitly trusted. It runs from read-only memory or write-protected memory. This could be on-chip ROM, on-chip OTP, locked on-chip flash, or write-protected on-chip RAM for example.

  This is a requirement of the Arm CCA security model:

  > *[R0158] Arm recommends that all initial boot code is immutable on a secured system.*
  >
  > *[R0050] If all or part of initial boot code is instantiated in on-chip memory then other trusted subsystems or application PE cannot modify that code before it has been executed.*

- Trusted boot and measured boot are enabled. This means an attacker can't boot arbitrary images that are not approved by platform providers.

  These are requirements of the Arm CCA security model:

  > *[R0048] A secured system can only load authorized CCA firmware.*
  >
  > *[R0079] All Monitor firmware loaded by PE initial boot is measured and verified as outlined in Verified boot.*

- No experimental features are enabled. These are typically incomplete features, which need more time to stabilize. Thus, we do not consider threats that may come from them. It is not recommended to use these features in production builds.

Data Flow Diagram

Figure 1 shows a high-level data flow diagram for TF-A. The diagram shows a model of the different components of a TF-A-based system and their interactions with TF-A. A description of each diagram element is given on Table 1. On the diagram, the red broken lines indicate trust boundaries. Components outside of the broken lines are considered untrusted by TF-A.

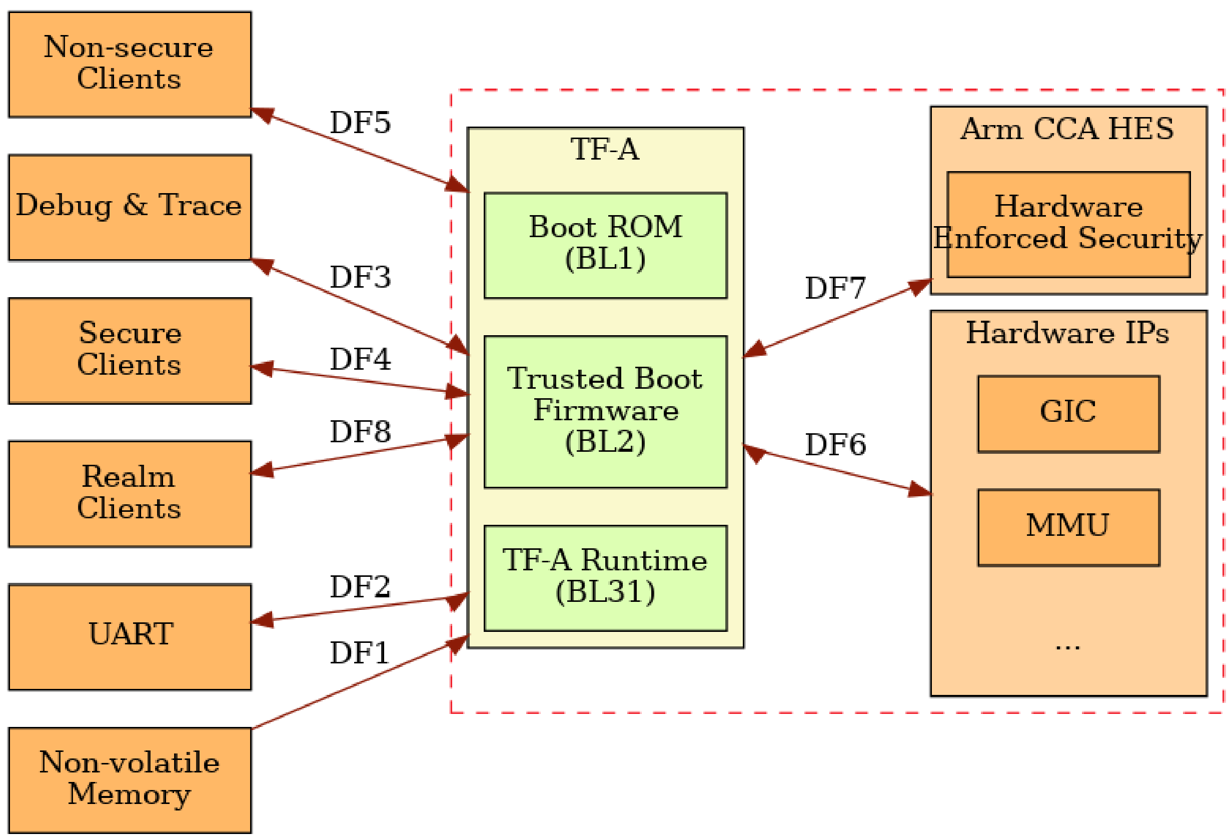


Fig. 4: Figure 1: Data Flow Diagram

Table 8: Table 1: Data Flow Diagram Description

Diagram Element	Description
DF1	Refer to DF1 description in the *Generic Threat Model*. Additionally TF-A loads realm images.
DF2-DF6	Refer to DF2-DF6 descriptions in the *Generic Threat Model*.
DF7	Boot images interact with Arm CCA HES to record boot measurements and retrieve data used for AP images authentication. The runtime firmware interacts with Arm CCA HES to obtain sensitive attestation data for the realm world.
DF8	Realm world software (e.g. TF-RMM) interact with TF-A through SMC call interface and/or shared memory.

Threat Analysis

In this threat model, we use the same method to analyse threats as in the *Generic Threat Model*. This section only points out differences where applicable.

- There is an additional threat agent: *RealmCode*. It takes the form of malicious or faulty code running in the realm world, including R-EL2, R-EL1 and R-EL0 levels.
- At this time we only consider the `Server` target environment. New threats identified in this threat model will only be given a risk rating for this environment. Other environments may be added in a future revision

Threat Assessment

General Threats for All Firmware Images

The following table analyses the *General Threats for All Firmware Images* in the context of this threat model. Only deltas are pointed out.

ID	Applicable?	Comments
05	Yes	
06	Yes	
08	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.
11	Yes	Misconfiguration of the Memory Management Unit (MMU) may allow a **normal/secure/realm** world software to access sensitive data, execute arbitrary code or access otherwise restricted HW interface. **Note that on RME systems, MMU configuration also includes Granule Protection Tables (GPT) setup.** Additional diagram elements: DF4, DF7, DF8. Additional threat agents: SecCode, RealmCode.
13	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.
15	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.

Threats to be Mitigated by the Boot Firmware

The following table analyses the *Threats to be Mitigated by the Boot Firmware* in the context of this threat model. Only deltas are pointed out.

ID	Applicable?	Comments
01	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.
02	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.
03	Yes	
04	Yes	

Threats to be Mitigated by the Runtime EL3 Firmware

The following table analyses the *Threats to be Mitigated by the Runtime EL3 Firmware* in the context of this threat model. Only deltas are pointed out.

ID	Applicable?	Comments
07	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.
09	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.
10	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.
12	Yes	Additional diagram element: DF8. Additional threat agent: RealmCode.
14	Yes	

11.1.5 Threat Model for TF-A with PSA FWU or TBBR FWU support

Introduction

This document provides a threat model of TF-A firmware for platforms with the feature PSA firmware update or TBBR firmware update or both enabled. To understand the design of the firmware update refer *Firmware Update (FWU)*.

Although it is a separate document, it references the *Generic Threat Model* in a number of places, as some of the contents are applicable to this threat model.

Target of Evaluation

In this threat model, the target of evaluation is the Trusted Firmware for A-class Processors (TF-A) when PSA FWU support is enabled or TBBR FWU mode is enabled. This includes the boot ROM (BL1), the trusted boot firmware (BL2).

Threat Assessment

For this section, please reference the Threat Assessment under the *Generic Threat Model*. Here only the differences are highlighted.

PSA FWU

Threats to be Mitigated by the Boot Firmware

The following table analyses the *Threats to be Mitigated by the Boot Firmware* in the context of this threat model. Only additional details are pointed out.

ID	Applicable?	Comments
01	Yes	Attacker can use arbitrary images to update the system.
02	Yes	Attacker tries to update the system with the vulnerable/older firmware.
03	Yes	
04	Yes	

Threats to be mitigated by platform design

PSA FWU is driven by metadata stored in non-volatile storage. This metadata is not cryptographically signed. Also, depending on the hardware design, it may be stored in untrusted storage, which makes it possible for software outside of TF-A security boundary or for a physical attacker to modify it in order to change the behaviour of the FWU process.

Below we provide some possible FWU metadata corruption scenarios:

1. The FWU metadata includes the firmware bank for booting; the attacker tries to modify it to prevent the execution of the updated firmware.

2. The FWU metadata features a field indicating the firmware's status, either in trial run or accepted run. The attacker tries to manipulate this field, ensuring the updated firmware consistently runs in trial mode, with the intention of preventing the anti-rollback update.

By design, no software mitigations exist to prevent this. The safeguarding of FWU metadata relies on the platform's hardware design to mitigate potential attacks on it, if this is a concern in the platform's threat model. For example, FWU metadata may be stored in secure storage under exclusive access from secure software, protecting it from physical, unauthenticated accesses and from non-secure software accesses.

TBBR FWU - Firmware Recovery

Threats to be Mitigated by the Boot Firmware

The following table analyses the *Threats to be Mitigated by the Boot Firmware* in the context of this threat model. Only additional details are pointed out.

ID	Applicable?	Comments
01	Yes	Attacker can use arbitrary images to recover the system.
02	Yes	Attacker tries to recover the system with the vulnerable/older firmware.
03	Yes	
04	Yes	

11.1.6 Threat Model for Firmware Handoff

Target of Evaluation

The target of this evaluation is the *Transfer List library* implementation of the Firmware Handoff specification. This library is a generic implementation shared by A-class firmware components. It is used to transfer state/metadata (e.g., Device Trees, entrypoint information) between boot stages in TF-A. This includes the boot ROM (BL1), trusted boot firmware (BL2), and runtime EL3 firmware (BL31).

It considers the generic case of handing off from more trusted component to a less trusted one (e.g. BL31 to BL33). It does not specifically consider the narrower case where the handoff is confined to TF-A stages only.

Relationship to the Generic TF-A Threat Model

This document focuses on the Transfer List (TL) library itself and should be read in conjunction with the *Generic Threat Model*, which covers TF-A as a whole.

Within the narrower TF-A context of BL1, BL2, and BL31, the TL producer and consumer are typically TF-A stages that remain within the same broader TF-A security boundary. In that context, some threats from this library-focused analysis do not necessarily need to be considered, especially where they depend on the TL being handed to a less-trusted stage outside TF-A.

Even so, it is safer to handle those threats in the same way. Doing so keeps the library robust when it is reused in other handoff paths, preserves defence-in-depth, and avoids relying on platform-specific assumptions about which stages may consume a TL.

Assumptions

The following assumptions are made about the TF-A Firmware Handoff framework implementation in TF-A:

- Within the context of Trusted Board Boot (*TBB*), the Transfer List (TL) producer is considered integrity checked.
- The shared buffer containing the TL is retained until it's consumed, and all necessary cache/coherency operations are performed to ensure the TL is visible to the receiver.
- Handoff between stages is atomic; that is, there aren't incremental updates or partial reads.

Data Flow Diagram

A generalized DFD showing information flow between a TL producer and receiver is presented below. The dataflow DF1 in this diagram is representative of dataflows in TF-A where Firmware Handoff might be used (e.g., `DF4`, and `DF5` in the generic model).

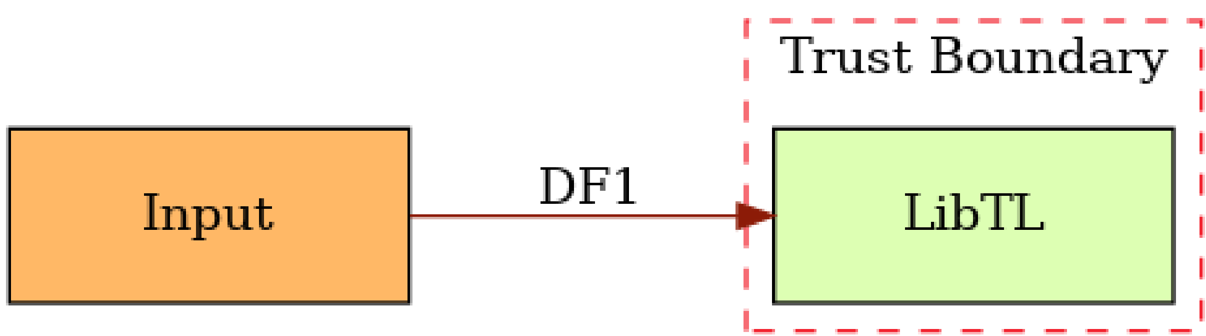


Fig. 5: Figure 1: Generalized Firmware Handoff Data Flow Diagram

Table 9: Table 1: TF-A Data Flow Diagram Description

Diagram Element	Description
DF1	Input data generated by a TL producer.

Threat Assessment

This analysis adopts the STRIDE framework to categorize and assess threats.

The primary asset is the sensitive data conveyed through the TL — including platform configuration data, firmware metadata, and potentially security-critical values exchanged between boot stages or execution environments.

Its main security responsibility is to manage the TL safely and correctly, ensuring that:

- Sensitive data is not corrupted, tampered with, or misrouted during processing.
- Confidential information within the TL is not exposed to unauthorized entities or domains.
- All data handling operations preserve integrity, confidentiality, and availability across trust boundaries (e.g., secure vs. non-secure world).

This includes protection against both malicious manipulation and accidental faults, as well as validation of inputs and outputs to prevent memory corruption or data leakage.

ID	Comments
05	**Information leak via UART logs** As in the general case, it's common to use logging to print information about a TL to the console. Because a TL can include sensitive data, the implementation must prevent said content from being written to the console. This happens when directly logging sensitive information and more subtly when logging side-channel information that can be used by an attacker to learn about sensitive information. Do not log the contents of Transfer Entries when logging is enabled. In production releases only print diagnostic error messages and ensure they do not contain sensitive information. Disable logging by default, and ensure users explicitly opt in and configure their preferred logging setup.
08	**Transfer List corruption via out-of-band copies** A producer typically performs various out-of-band memory operations on the TL. Common scenarios include authentication and loading of the platform *FDT*, and relocation of the TL from one buffer to another. Any of these operations may corrupt the TL due to copy-related bugs (e.g., errors in `memcpy` or similar memory manipulation functions). Verify the checksum before consuming a TL. Note that this mitigation does not stop an attacker from altering the data and recalculating the checksum; checksums only help detect data corruption.
13	**Sensitive data exposure from reused TLs**

11.2 TF-A Supply Chain Threat Model

11.2.1 Introduction

Software supply chain attacks aim to inject malicious code into a software product. There are several ways a malicious code can be injected into a software product (open-source project). These include:

- Malicious code commits: This attack directly injects code into a project repository. This can happen for example through developer/maintainer credential hijacks, or malicious external contributors.
- Malicious dependencies: In this case malicious code is introduced into a project through other piece of code or packages the project depends on. This can happen through for example typosquatting attack where an attacker creates a malicious package with a very similar name to a popular package and hosts it on popular package repositories.
- Malicious toolchains: This involves malicious code introduced by compromised resources used throughout the development and/or build process such as compilers and IDEs.

This document provides analysis of software supply chain attack threats for the TF-A project.

11.2.2 TF-A Overview

Figure 1 shows the different software components surrounding the TF-A project. A brief description of each component is provided below.

TF-A Repository

The TF-A repository contains generic and platform code contributed by TF-A contributors as well as libraries imported from other open-source projects, referred to as internal dependencies on Figure 1. These libraries include:

- *libfdt*: libfdt is a utility library for reading and manipulating Device Tree Binary (DTB) files. It is part of the Device Tree Compiler (DTC) toolchain[1]. DTC is used as part of the build process on the host machine to build DTB files. libfdt is used to parse the DTB files at boot time.
- *zlib*: zlib is a data compression library imported from[2].
- *compiler-rt*: This is a collection of runtime libraries from the LLVM compiler infrastructure project[3]. We import the builtins library which provides low-level, target-specific compiler builtins from compiler-rt.

---

[1] https://git.kernel.org/pub/scm/utils/dtc/dtc.git
[2] http://zlib.net/
[3] https://compiler-rt.llvm.org/

The TF-A repository also includes source code for host tools that supplement the TF-A build process. These tools include:

- *fiptool*: This tool is used to create a Firmware Image Package (FIP) which allows for packing bootloader images into a single archive that can be loaded by TF-A from non-volatile platform storage.
- *cert_create*: This tool is used to generate certificates for binary images.
- *encrypt_fw*: This tool takes the plain firmware image as input and generates the encrypted firmware image which can then be passed as input to the fiptool utility for creating the FIP.
- *sptool*: This tool is used to build the secure partition packages.

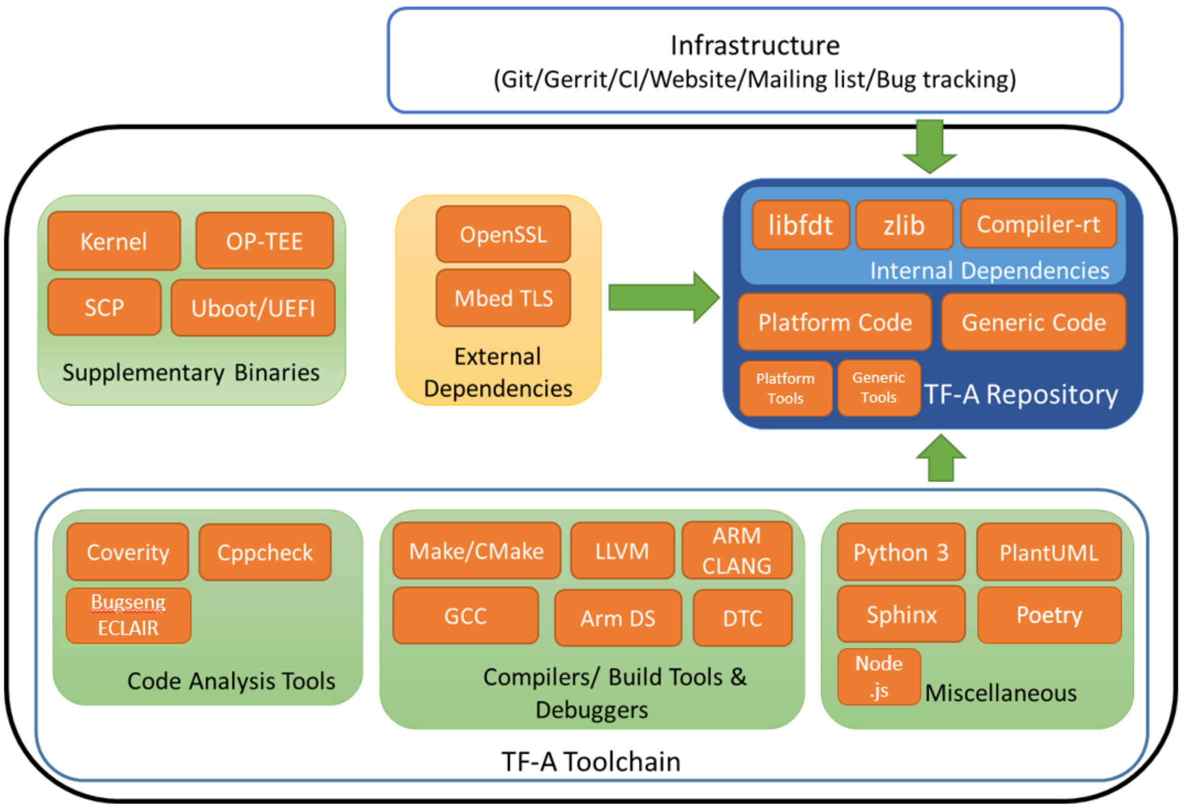


Figure 1: TF-A System Diagram

External Dependencies

These are software components that are not part of the TF-A repository but are required to build TF-A binaries and host tools.

- *Mbed TLS Library*: This is a cryptography library from trustedfirmware.org (tf.org). It is required to build TF-A binaries where cryptography features are needed, such as Trusted Board Boot (TBB).
- *OpenSSL Library*: This is another cryptography library used by TF-A host tools: fiptool, cert_create, and encrypt_fw.

The following table lists TF-A dependencies including the sources of the dependencies.

Table 10: Table 1: TF-A Dependencies

Dependency	Location of Dependency	Original Source
libfdt	Local copy	Page 902, 1
zlib	Local copy	Page 902, 2
compiler-rt	Local copy	Page 902, 3
Mbed TLS	External	4
OpenSSL	External	5

Supplementary Binaries

These are binaries used to test TF-A based systems. Below is a brief description of each component and where they are sourced from.

- *SCP-firmware*: For our tests, we use SCP-firmware binaries supplied by the Arm SCP team built from the source from the GitHub repository[6].
- *OP-TEE*: Trusted Execution Environment (TEE) from tf.org that runs as Secure EL1. We use OP-TEE built from source or binaries supplied with Arm Reference Platforms depending on the test configuration.
- *EDK2 UEFI*: Normal world bootloader from the EDK2 project[7]. We use EDK2 UEFI binaries hosted on tf.org servers for testing[8].

Other software components used to test TF-A include U-Boot, Linux kernel, RSE, MCP, and file systems, all sourced from the Arm Reference Platforms teams.

TF-A Toolchain

The TF-A project uses several tools to build, analyze and test the TF-A source code.

Node.js Tools

These are optional quality assurance and developer utility tools that are installed through the use of the Node.js package manager. They are pinned to specific versions described by the package.json file in the root of the TF-A repository, and their dependencies are downloaded from the internet at the point of installation. These tools may be installed locally on the developer machine and are installed within a Docker container in certain CI jobs. At present, these are:

- Commitlint
- Commitizen
- Husky

[4] https://tls.mbed.org/
[5] https://www.openssl.org/
[6] https://github.com/ARM-software/SCP-firmware
[7] https://github.com/tianocore/edk2
[8] https://downloads.trustedfirmware.org/tf-a/

Infrastructure

TF-A uses trustedfirmware.org (tf.org) and Arm infrastructures to host the source code, review code and run tests. Appendix A provides a security analysis of tf.org infrastructure.

11.2.3 TF-A Data Flow

Figure 2 below shows the data flow diagram for TF-A. The broken red lines indicate trust boundaries.

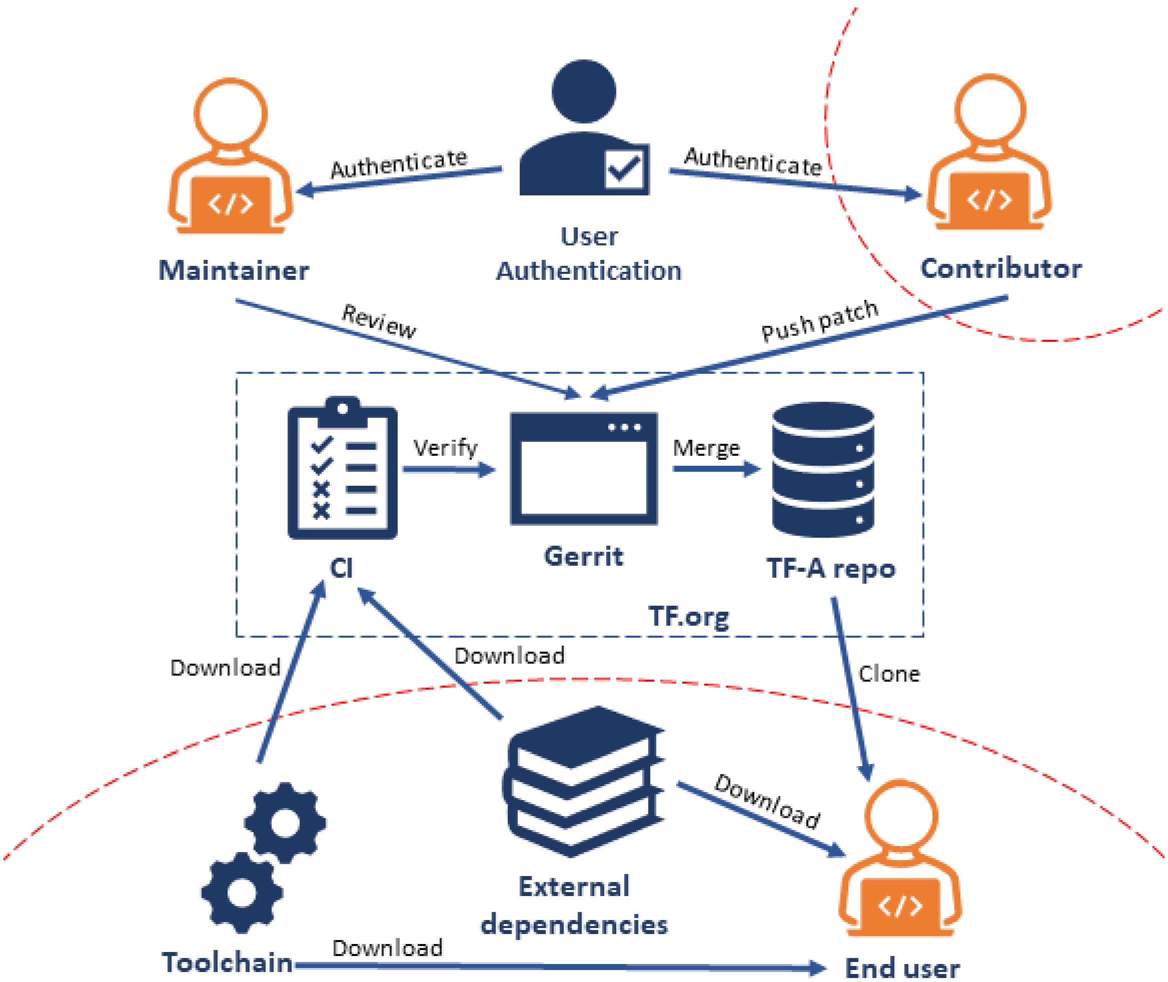


Figure 2: TF-A Data Flow Diagram

11.2.4 Attack Tree

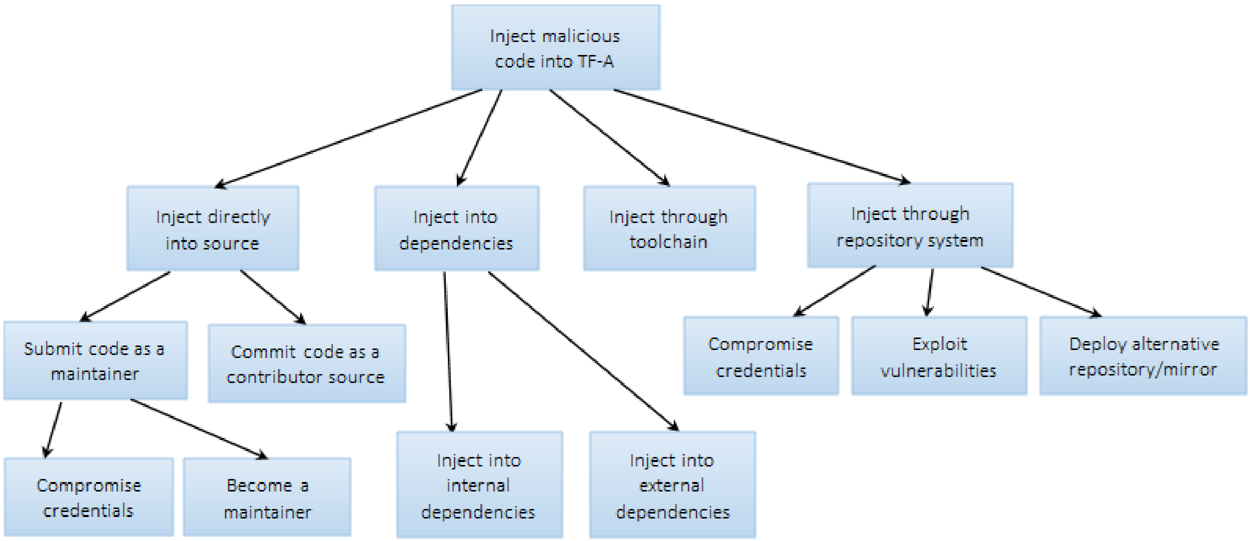


Figure 3: TF-A Attack Tree

11.2.5 Threat Assessment and Mitigations

Impact and Likelihood Ratings

Rating	Impact	Likelihood
HIGH	Major impact to entire organization or single line of business if exploited.	Threat is relatively easy to exploit by an attacker with little effort and skill.
MEDIUM	Noticeable impact to line of business if exploited.	An expert attacker could exploit the threat without much difficulty.
LOW	Minor damage if exploited or could be used in conjunction with other vulnerabilities to perform a more serious attack.	Exploiting the threat would require considerable effort and resources.

Threats and Mitigations

Threat naming convention key

- SC – Supply Chain
- SRC – Source
- DEP – Dependency
- TOOL – Toolchain

- REPO – Repository
- MAIN – Maintainer
- CONT – Contributor

Threat: TFA-SC-SRC-MAIN-01	
Description	An attacker can submit and merge malicious code by posing as a maintainer after compromising maintainers' credentials.
Impact	HIGH
Likelihood	MEDIUM
Threat and impact	In the TF-A code review process all submitted changes undergo review by a code owner and a maintainer. If the change is accepted, it will be merged (integrated) into an integration branch by a maintainer. A maintainer has the right to give a code owner review, a maintainer review and merge the submitted change. tf.org users (including maintainers) are authenticated through GitHub. The likelihood of a credential compromise depends on multiple factors. The authentication mechanism of GitHub is strong if the recommended best practices are followed[9] making credential compromise unlikely. GitHub (therefore tf.org) allows logins with two-factor authentication, requiring both a password and access to the user's authentication code. Depending on the strength of the password and factors such as whether the maintainer reuses passwords across services, the likelihood of a compromise can be higher. If an attacker manages to compromise a maintainer's credentials, posing as the maintainer, they can in theory submit a malicious change (as a maintainer or as a contributor), give all the necessary reviews and merge the change.
Mitigations	- Enforce best practices recommended by GitHub[Page 909, 9] - Not allowing a committer to both self-review and merge patches they have submitted. To achieve the commit the attacker would be required to compromise at least two credentials (reviewers and maintainer).
Mitigations implemented?	We have not disallowed self-review/merge of patches

Threat: TFA-SC-SRC-MAIN-02	
Description	An attacker can submit and merge malicious code after becoming a maintainer through social engineering techniques.
Impact	HIGH
Likelihood	LOW
Threat and impact	According to the TF project maintenance process[10], maintainers of TF-A are selected by their peers based on merit. Some of the criteria of becoming a maintainer include being an active member of the project for a minimum duration and contributing a substantial number of non-trivial and high-quality patches. However, there are some weaknesses in the process: - There is no structured mechanism to establish trust with a maintainer other than the recommendations by peers - There is no continuous monitoring of the status of a maintainer (e.g. maintainer can move from one organization to another) To perform such an attack, in addition to becoming a maintainer, an attacker also must deal with all restrictions put on maintainers.
Mitigations	- Structured mechanism to establish trust with maintainers - Not allowing a committer to both self-review and merge patches they have submitted. To achieve the commit the attacker would be required to compromise at least two credentials (reviewers and maintainer).
Mitigations implemented?	There is a structured mechanism to establish trust with maintainers, but self-review/merge of patches is not disallowed

[9] https://docs.github.com/en/github/authenticating-to-github/creating-a-strong-password

[10] https://trustedfirmware-a.readthedocs.io/en/latest/process/maintenance.html#how-to-become-a-maintainer

Threat: TFA-SC-SRC-CONT-01	
Description	An attacker can submit malicious code patch as a contributor.
Impact	HIGH
Likelihood	LOW
Threat and impact	TF-A accepts external contributions to both the generic and platform code. Unlike maintainers, contributors do not have maintainer review or merging privileges, therefore the likelihood of injecting malicious code as a contributor is lower. However, even though unlikely, it is still possible for a malicious commit to go unnoticed through the code review and verification processes. If successful, the impact can range from low to high depending on the injected code. For example, an attacker can potentially deliberately insert a memory corruption vulnerability that is hard to notice on code review and will not be detected by the verification process. This vulnerability by itself may have a low impact but can have a major impact if used in combination with other vulnerabilities.
Proposed Mitigations	– Code review and verification – Static analysis to try to pick up issues that typically end in some form of attack vector
Mitigations implemented?	Yes, contributions go through the thorough review, verification, and static analysis process automated through CI

Threat: TFA-SC-DEP-01	
Description	An attacker can inject malicious code into TF-A internal dependencies.
Impact	HIGH
Likelihood	LOW
Threat and impact	TF-A has two types of dependencies: those that are copied into the TF-A repository and shipped as part of TF-A code (referred to as *internal dependencies* here) and those that are downloaded from external repositories and used when building TF-A (referred to as *external dependencies* here). Currently TF-A has three internal dependencies: *libfdt*[Page 902, 1], *zlib*[Page 902, 2] and *compiler-rt*[Page 902, 3] libraries. These libraries are periodically updated by copying them from their source repositories. Although unlikely, it is possible for a contributor to copy the libraries from the wrong (and potentially malicious) repositories. For example, there are already multiple forks of *libfdt* (DTC) on GitHub. In addition to this, the official repositories are not immune to threats described above (TFA-SC-SRC-MAIN-01, TFA-SC-SRC-MAIN-02 and TFA-SC-SRC-CONT-01). The likelihood of an attack on TF-A through internal dependencies is lower than external dependencies for the following reasons: - Internal dependencies go through the normal code review process during upgrade - Once upgraded internal dependencies stay unchanged until the next upgrade. The upgrade window is typically long (for example *libfdt* has only changed 4 times over the past 4 years). This reduces the window of opportunity for an attacker to inject malicious code into the dependencies
Proposed Mitigations	– Explicitly document versions and official sources of dependencies – Keep a copy of a pinned version of the source code inside the TF-A tree so that the risk of getting malicious code from dependencies only arises when we upgrade them – Monitor alerts for vulnerable dependencies from GitHub[11]

Threat: TFA-SC-DEP-02	
Description	An attacker can inject malicious code into TF-A external dependencies.
Impact	HIGH
Likelihood	MEDIUM
Threat and impact	Unlike internal dependencies, external dependencies are downloaded from external repositories by end-users. Although the TF-A documentation provides information about the versions of dependencies used for testing and links to repositories, it is up to the end-user to decide where to get the dependencies from. As such, the likelihood of an attack through an external dependency is higher compared to an internal dependency. The impact of an attack ranges from low to critical depending on which dependency and what part of the dependency is affected. For example, a malicious code that affects the signature verification functions in MbedTLS is considered critical as it can be used to bypass the TBB process of TF-A.
Proposed Mitigations	– Explicitly document versions and official sources of dependencies – Provide scripts and build options to automatically fetch the latest stable release of external dependencies
Mitigations implemented?	We explicitly document versions and official sources of dependencies, but do not yet provide scripts and build options to automatically fetch the latest stable release of external dependencies

[11] https://docs.github.com/en/github/managing-security-vulnerabilities/about-alerts-for-vulnerable-dependencies

Threat: TFA-SC-REPO-01	
Description	An attacker can upload malicious versions of TF-A by compromising credentials of administrator accounts on tf.org or GitHub.
Impact	HIGH
Likelihood	LOW
Threat and impact	This attack is like TFA-SC-SRC-MAIN-01, but the likelihood and impact of the two attacks are different. The likelihood of compromising administrator credentials is lower than that of a maintainer's (assuming both use authentication methods of similar strength) as there are smaller number of administrators than maintainers. On the other hand, the impact is higher since an administrator has more privileges than a maintainer: - An administrator can upload a malicious TF-A contribution unnoticed by other reviewers - An administrator can potentially rewrite the history of the repository to evade detection
Proposed Mitigations	Strong authentication (Follow best practices recommended by GitHub[Page 909, 9])
Mitigations implemented?	Yes, strong authentication is implemented through recommended best practices

Threat: TFA-SC-REPO-02	
Description	An attacker can upload malicious versions of TF-A after getting write access to the repository by exploiting a vulnerability on tf.org or GitHub.
Impact	HIGH
Likelihood	LOW
Threat and impact	There are no reports of someone exploiting a vulnerability on GitHub or tf.org to upload malicious contributions. However, there are examples of vulnerabilities that allowed arbitrary code execution on popular hosting services[12]. Such vulnerabilities can potentially be used to upload malicious packages. In addition to being hard to exploit, vulnerabilities on popular hosting sites such as GitHub are typically detected quickly, making the window of opportunity for such attack small.
Proposed Mitigations	– Monitor alerts of any vulnerabilities that might affect TF-A repository – Ensure tf.org is up to date with latest security patches
Mitigations implemented?	Yes, alerts of vulnerabilities are monitored and tf.org is ensured to be up to date with the latest security patches

[12] "Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks"

Threat: TFA-SC-REPO-03	
Description	An attacker can host a malicious version of TF-A on an attacker-controlled repository, and trick end-users into downloading from that repository.
Impact	HIGH
Likelihood	MEDIUM
Threat and impact	It is not difficult for an attacker to create a website with a similar domain name and look as tf.org (website spoofing) and host a malicious TF-A source repository. Similarly, an attacker can create a mirror of the TF-A repository on GitHub with malicious code in it. However, for this attack to succeed the attacker needs to trick the end-user into using the attacker-controlled repositories.
Proposed Mitigations	– Users should carefully check the URL of the website before visiting it and the URL of the repository before checking it out – Accept reports of spoofing attacks on tf.org and broadcast a warning to partners
Mitigations implemented?	We accept reports of spoofing attacks on tf.org and will broadcast a warning to partners

Threat: TFA-SC-TOOL-01	
Description	Malicious code can be injected at build time through malicious tools.
Impact	HIGH
Likelihood	LOW
Threat and impact	End-users of TF-A use make (or cmake), compilers and linkers (armgcc, armclang or LLVM) to build TF-A binaries. Although TF-A documentation specifies versions and official sources of tools used to build TF-A, users can potentially be tricked into using unofficial, malicious toolchains. Similar attacks have been used in the past to inject malicious code into final products[13].
Proposed Mitigations	– Explicitly document versions and official sources of toolchains – Provide scripts to automatically fetch the latest stable release of toolchains
Mitigations implemented?	We explicitly document versions and official sources of toolchains, but have not yet provided scripts to automatically fetch the latest stable release of toolchains

[13] https://www.wired.com/story/supply-chain-hackers-videogames-asus-ccleaner/

Threat: TFA-SC-TOOL-02	
Description	Malicious code can be executed by developer's tools at installation time through malicious Node.js dependencies.
Impact	HIGH
Likelihood	LOW
Threat and impact	Users of the Node.js tools, including the CI, may be exposed to malicious dependencies that have been missed by the Node.js dependency auditor. Users of these tools could potentially be executing malicious code when using these tools, which could potentially allow a malicious actor to make silent modifications to the repository or enable retrieval of user credentials. If successful, the impact can range from low to high depending on the user's credentials. If the user is an administrator, this could imply TFA-SC-REPO-01.
Proposed Mitigations	– Limit Node.js tools to a minimal set of trusted packages – Pin Node.js packages to known versions – Update dependencies for which Node.js's auditor reports known CVEs – Execute Node.js tools in the CI only from within a trusted container
Mitigations implemented?	Yes, Node.js tools are limited to a minimal set of trusted packages, packages are pinned to known versions, dependencies are updated when there are known CVEs reported, and Node.js tools are only executed within a trusted container in CI

11.2.6 Appendix A

Summary of trustedfirmware.org security:

Table 11: Table 2: Security information of trustedfirmware.org

Software/ System	Source and integrity	Credential and permission management	Security incident response plan
Jenkins (including plugins)	• Jenkins is built using Dockerfile which is based on the official Jenkins docker image • Jenkins plugins are built using the official install-plugins.sh	• Use oauth from Github only • The password strength follows Github policy • Do not enforce using two-factor authentication • Jenkins uses matrix auth which allows users to manage "job" level ACL using Jenkins Job Builder • No API token enabled • Jenkins uses the inbuilt credential store where we store credentials for LAVA, Jenkins Job Builder, DockerHub, AWS and Gerrit tokens. The credentials are stored as a secret in Jenkins credential store. These credentials can be accessed via a Jenkins job, but someone would have to push a Jenkins Job through a Gerrit review to do this. Gerrit maintains the ACL for this and only admins and project approver can +2 a review.	• Monitor CVE's and update Jenkins LTS on a monthly cycle • Keep plugins up-to-date. But it is up to the plugin owner to maintain said plugin
Gerrit (including plugins)	• Gerrit package is installed from Linaro top level	• Use oauth from Github only • The password	• Keep plugins up-to-date. But it is up to the plugin

11.2.7 References

---

[14] https://gerrit-ci.gerritforge.com/plugin-manager/

CHAPTER TWELVE

ARCHITECTURE FEATURES

The Arm architecture evolves through versioning, with new architectural features being introduced almost every year listed in Feature_description. While most of these features require no control at EL3, some demand explicit configuration of EL3 control registers to enable their use at lower Exception Levels. For features with EL3 controls, it is relatively straightforward to examine the code and determine whether TF-A support them. However, for features that are transparent to EL3, it is more challenging to confirm whether they have been analyzed and verified as not requiring changes in TF-A.

This document serves as a central reference point to track and clarify the support status of architectural features within TF-A.

This document organizes features according to the yearly updates, while the specific architecture versions (8.X, 9.X) to which they apply can be found in Feature_description

Note: Status legend —

- `OK`: TF-A has explicit support;
- `NA`: Transparent from EL3 (no EL3 changes required);
- `NO`: Analyzed but decided not to implement;
- `WIP`: Implementation in progress;
- *(empty)*: Not analyzed.

12.1 2025 Extensions

Feature	Status
FEAT_GCIE	OK
FEAT_GCIE_LEGACY	NA
FEAT_F16F32DOT	NA
FEAT_F16F32MM	NA
FEAT_F16MM	NA
FEAT_SVE_B16MM	NA
FEAT_SME2p3	NA
FEAT_MTE_EIRG	NA
FEAT_MTETC	NA
FEAT_SVE2p3	NA
FEAT_SCR2	
FEAT_LSCP	NA
FEAT_MPAMv2_VID	NA
FEAT_MPAMv2_MSC	NA
FEAT_MPAMv2_MSC_MON_SEC	NA
FEAT_MPAMv2_ALT_ID	NA
FEAT_MPAMv2	
FEAT_FDIT	
FEAT_CMH	NA
FEAT_EAESR	NA
FEAT_TLBID	NA
FEAT_SRMASK2	
FEAT_BBML3	NA
FEAT_NV3	
FEAT_PAuth_EnhCtl	
FEAT_LORRL	NA

12.2 2024 Extensions

Feature	Status
FEAT_PMUv3_EXTPMN	NA
FEAT_SPEv1p5	OK
FEAT_SPE_EXC	OK
FEAT_SPE_nVM	OK
FEAT_TRBEv1p1	NA
FEAT_TRBE_EXC	
FEAT_SME_MOP4	NA
FEAT_SME_TMOP	NA
FEAT_PoPS	NA

continues on next page

Table 1 – continued from previous page

Feature	Status
FEAT_SME2p2	NA
FEAT_CMPBR	NA
FEAT_SSVE_AES	NA
FEAT_RME_GPC3	WIP
FEAT_RME_GDI	OK
FEAT_SVE2p2	NA
FEAT_SSVE_BitPerm	NA
FEAT_SSVE_FEXPA	NA
FEAT_SVE_AES2	NA
FEAT_LSFE	NA
FEAT_FPRCVT	NA
FEAT_SVE_F16F32MM	NA
FEAT_F8F16MM	NA
FEAT_F8F32MM	NA
FEAT_SVE_BFSCALE	NA
FEAT_OCCMO	NA
FEAT_LS64WB	NA
FEAT_AMU_EXTACR	NA
FEAT_MPAM_MSC_DOMAINS	NA
FEAT_MPAM_MSC_DCTRL	NA
FEAT_MPAM_PE_BW_CTRL	OK
FEAT_PCDPHINT	NA
FEAT_SRMASK	
FEAT_IDTE3	OK
FEAT_UINJ	OK
FEAT_NV2p1	NA
FEAT_LSUI	NA

12.3 2023 Extensions

Feature	Status
FEAT_E2H0	NA
FEAT_SME_LUTv2	NA
FEAT_LUT	NA
FEAT_FAMINMAX	NA
FEAT_FPMR	OK
FEAT_FP8	NA
FEAT_FP8FMA	NA
FEAT_SSVE_FP8FMA	NA
FEAT_FP8DOT4	NA
FEAT_SSVE_FP8DOT4	NA

continues on next page

Table 2 – continued from previous page

Feature	Status
FEAT_FP8DOT2	NA
FEAT_SSVE_FP8DOT2	NA
FEAT_SME_F8F16	NA
FEAT_SME_F8F32	NA
FEAT_CPA	NA
FEAT_CPA2	OK
FEAT_STEP2	OK
FEAT_BWE2	NA
FEAT_SPE_FPF	NA
FEAT_SPE_EFT	NA
FEAT_PMUv3_TH2	NA
FEAT_SPMU2	NA
FEAT_E3DSE	
FEAT_PMUv3_SME	NA
FEAT_SPE_SME	NA
FEAT_SPE_ALTCLK	NA
FEAT_HDBSS	OK
FEAT_HACDBS	OK
FEAT_TLBIW	NA
FEAT_ASID2	NA
FEAT_RME_GPC2	OK
FEAT_FGWTE3	OK
FEAT_PAuth_LR	OK
FEAT_ETS3	NA

12.4 2022 Extensions

Feature	Status
FEAT_BWE	NA
FEAT_ADERR	
FEAT_ANERR	
FEAT_DoubleFault2	
FEAT_PFAR	OK
FEAT_RASv2	
FEAT_CSSC	NA
FEAT_RPRFM	NA
FEAT_PRFMSLC	NA
FEAT_SPECRES2	
FEAT_CLRBHB	Ok
FEAT_ECBHB	NA
FEAT_SME2p1	NA

continues on next page

Table 3 – continued from previous page

Feature	Status
FEAT_SME_F16F16	NA
FEAT_SVE_B16B16	NA
FEAT_SME_B16B16	NA
FEAT_FGT2	OK
FEAT_MTE4	NA
FEAT_MTE_CANONICAL_TAGS	NA
FEAT_MTE_NO_ADDRESS_TAGS	NA
FEAT_MTE_TAGGED_FAR	NA
FEAT_MTE_PERM	NA
FEAT_MTE_STORE_ONLY	NA
FEAT_ITE	NA
FEAT_TRBE_EXT	NA
FEAT_TRBE_MPAM	
FEAT_ETEv1p3	NA
FEAT_GCS	OK
FEAT_CHK	NA
FEAT_SPE_DPFZS	NA
FEAT_SPE_CRR	NA
FEAT_EBEP	OK
FEAT_SEBEP	OK
FEAT_PMUv3_SS	
FEAT_SPMU	
FEAT_PMUv3_ICNTR	
FEAT_PMUv3p9	OK
FEAT_SVE2p1	NA
FEAT_Debugv8p9	OK
FEAT_ABLE	NA
FEAT_PCSRv8p9	NA
FEAT_LRCPC3	NA
FEAT_SPEv1p4	NA
FEAT_SPE_FDS	OK
FEAT_PMUv3_EDGE	NA
FEAT_D128	OK
FEAT_AIE	OK
FEAT_S1PIE	OK
FEAT_ATS1A	NA
FEAT_S1POE	OK
FEAT_S2PIE	OK
FEAT_S2POE	OK
FEAT_SYSREG128	OK
FEAT_SYSINSTR128	NA
FEAT_LSE128	NA
FEAT_HAFT	NA

continues on next page

Table 3 – continued from previous page

Feature	Status
FEAT_THE	OK
FEAT_LVA3	NA
FEAT_MTE_ASYNC	NA
FEAT_EDHSR	NA
FEAT_AMU_EXT64	NA

12.5 2021 Extensions

Feature	Status
FEAT_SME2	OK
FEAT_MEC	OK
FEAT_BRBEv1p1	WIP
FEAT_CMOW	NA
FEAT_Debugv8p8	NA
FEAT_HBC	NA
FEAT_MOPS	OK
FEAT_NMI	OK
FEAT_SPEv1p3	NA
FEAT_TIDCP1	NA
FEAT_PMUv3_TH	NA
FEAT_PMUv3p8	NA
FEAT_SCTLR2	OK
FEAT_TCR2	OK
FEAT_PMUv3_EXT64	NA

12.6 2020 Extensions

Feature	Status
FEAT_EBF16	NA
FEAT_ETEv1p1	NA
FEAT_HCX	OK
FEAT_PAN3	OK
FEAT_WFxT	NA
FEAT_XS	NO
FEAT_AFP	NA
FEAT_RPRES	NA
FEAT_LPA2	NO
FEAT_LS64	NA
FEAT_LS64_V	NA
FEAT_LS64_ACCDATA	OK
FEAT_MTE3	NA
FEAT_MTE_ASYM_FAULT	NA
FEAT_SPEv1p2	OK
FEAT_SPE_FnE	NA
FEAT_SPE_PBT	OK
FEAT_PMUv3p7	OK

12.7 The ENABLE_FEAT mechanism

The Arm architecture defines several architecture extension features, named FEAT_xxx in the architecure manual. Some of those features require setup code in higher exception levels, other features might be used by TF-A code itself.

Most of the feature flags defined in the TF-A build system are allowed to take the values 0, 1 or 2, with the following meaning:

```
ENABLE_FEAT_* = 0: Feature is unconditionally disabled at compile time.
ENABLE_FEAT_* = 1: Feature is unconditionally enabled at compile time.
ENABLE_FEAT_* = 2: Feature support is compiled in, but only enabled if the
                   feature present at runtime.
```

When setting the flag to 0, the feature is disabled during compilation, and the compiler's optimization stage and the linker will try to remove as much of this code as possible.

If it is defined to 1, the code will use the feature unconditionally, so the CPU is expected to support that feature. The FEATURE_DETECTION debug feature, if enabled, will verify this.

If the feature flag is set to 2, support for the feature will be compiled in, but its existence will be checked at runtime, so it works on CPUs with or without the feature. This is mostly useful for platforms which either support multiple different CPUs, or where the CPU is configured at runtime, like in emulators.

12.8 Adding support for a new feature

Adding support for a feature is required when:

a) a feature is added to the architecture and it includes controls at EL3

b) a feature is to be used by EL3 itself

c) EL3 wants to hide a feature from lower ELs (eg. with `FEAT_IDTE3`)

The following is a checklist with guidance on what to do to add a new feature, in this case a fictional `FEAT_ABC`. This is not an exhaustive list on how to do this, so please consult with previous such patches and/or maintainers. Please consult the Arm ARM on any specifics about the feature itself. You can speed this process up by using the `new-cpu-feature` agent skill.

- Add a feature flag to the build system.
  - The flag's name should be in the form `ENABLE_FEAT_ABC`.
  - Add it to the `assert_numerics` and `add_defines` lists in the `Makefile`.
  - Add a default of `0` for it in `make_helpers/defaults.mk`.
  - If the feature is listed as mandatory from a certain revision of the architecture, add it to the appropriate list in `make_helpers/arch_features.mk`.
  - Add any constraints in `make_helpers/constraints.mk`. This will usually be other features which this feature depends on or is incompatible with and can be found in the main description of the feature in the Arm ARM.
- Add the feature accessor functions in `include/arch/aarch64/arch_features.h`.

  A pair of functions must be defined in the form of `is_feat_abc_present()` and `is_feat_abc_supported()`. Please use the provided macros to define these.
  - The `_supported` variant is the one that should usually be used. If common code is to use it, a corresponding AArch32 version should be provided. It should return `false` if AArch32 is not to be supported.
  - In rare occasions, the feature must be checked in assembly. A `is_feat_abc_present_asm` macro must be added to `include/lib/cpus/aarch64/cpu_macros.S`.
- Register the feature in `common/feat_detect.c`.
- Register the feature in lib/extensions/idte/idte3.c.
- Add support for the feature. This will be very feature specific and exact code will depend on why support is being added. Usually, support will be a few lines in `lib/el3_runtime/aarch64/context_mgmt.c`. Some common things to do include:
  - If support requires more extensive code sequences, consider adding them to a file in `lib/extensions/abc/`. Functions in this file must be named similar to other features. Those are likely to be `abc_enable()`, `abc_enable_el3()`, or `abc_disable()`. Naming depends on the call site.

- If the feature introduces `SCR_EL3`, `MDCR_EL3`, or `CPTR_EL3` controls for lower ELs, access must be enabled for at least NS world, preferably all worlds by setting/clearing the bit in the context copy of this register.
- If the feature introduces any control bits in `SCR_EL3`, `MDCR_EL3`, `CPTR_EL3`, or `MPAM3_EL3`, those must be added to the appropriate list in `include/services/arm_arch_svc.h` to register with `ARCH_FEATURE_AVAILABILITY`. This must also be done in the `test_smccc_arch_feature_availability` test in TFTF, otherwise CI will fail.
- If the feature comes with lower EL registers:
  - those should be context switched if enabling the feature for multiple worlds.
  - if these are EL2 control registers, consider their fields' reset values. If they can reset to a value that would make a field active (like enabling a trap), a safe inactive reset value should be added to `setup_el2_context()` in `lib/el3_runtime/aarch64/context_mgmt.c`.
- Consider whether the feature introduces any potential side channels and how to close them.
- Add documentation in `docs/getting_started/build-options.rst`. This should include a brief description of what the enablement does, what values the flag can take, the default, and that it conforms to the `ENABLE_FEAT` mechanism. Adding feature enablement that does not conform to this mechanism is only done in exceptional cases. You should also update the list of features at the top of this file.
- Enable the feature for the FVP platform in `plat/arm/board/fvp/platform.mk`, setting it to `2` (`FEAT_STATE_CHECKED`).
- If the feature is optional, enable it in CI in the "allconfig" fragments. Those are currently `run_config/fvp-aemv8a.all.bmcov`, `run_config/fvp-spm.all.bmcov`, `run_config/fvp-spm.all`, and `run_config/fvp-aemv8a.all`. Mandatory features are expected to be covered by the architecture revision passed to the model.
- Optionally, although recommended, is to add a test in TFTF. This should only be done if there are observable effects from lower ELs like register traps. It is not necessary to add a test for functionality that does not have EL3 involvement (eg. whether a new instruction that cannot trap works).

CHAPTER

THIRTEEN

TOOLS

13.1 TF-A Memory Layout Tool

TF-A's memory layout tool is a Python script for analyzing the virtual memory layout of TF-A builds.

13.1.1 Prerequisites

1. Python (3.8 or later)
2. Poetry Python package manager

13.1.2 Getting Started

1. Install Poetry

```
curl -sSL https://install.python-poetry.org | python3 -
```

2. Install the required packages

```
poetry install --no-root
```

3. Verify that the tool runs in the installed virtual environment

```
poetry run memory --help
```

13.1.3 Symbol Virtual Map

The tool can be used to generate a visualisation of the symbol table. By default, it prints the symbols representing the start and end address of the main memory regions in an ELF file (i.e. text, bss, rodata) but can be modified to print any set of symbols.

```
$ poetry run memory symbols
build-path: build/fvp/release
Virtual Address Map:
           +------------__BL1_RAM_END__------------+------------------------
```

(continues on next page)

(continued from previous page)

```
↪-------------+
            +---------__COHERENT_RAM_END__----------+                                       ␣
↪               |
            +---------__COHERENT_RAM_START__---------+                                      ␣
↪               |
0x0403b000 +----------__XLAT_TABLE_END__-----------+                                        ␣
↪               |
0x04036000 +---------__XLAT_TABLE_START__----------+                                        ␣
↪               |
            +---------__BASE_XLAT_TABLE_END__--------+                                      ␣
↪               |
0x04035600 +--------------__BSS_END__--------------+                                        ␣
↪               |
            +--------__BASE_XLAT_TABLE_START__-------+                                      ␣
↪               |
            +-----__PMF_PERCPU_TIMESTAMP_END__------+                                       ␣
↪               |
            +---------__PMF_TIMESTAMP_END__---------+                                       ␣
↪               |
0x04035400 +--------__PMF_TIMESTAMP_START__--------+                                        ␣
↪               |
            +-------------__BSS_START__-------------+                                       ␣
↪               |
0x04034a00 +------------__STACKS_END__-------------+                                        ␣
↪               |
0x04034500 +-----------__STACKS_START__------------+                                        ␣
↪               |
0x040344c5 +-----------__DATA_RAM_END__------------+                                        ␣
↪               |
            +-----------__BL1_RAM_START__-----------+                                       ␣
↪               |
0x04034000 +----------__DATA_RAM_START__-----------+                                        ␣
↪               |
            |                                       +---------__COHERENT_RAM_
↪END__----------+
            |                                       +---------__COHERENT_RAM_
↪START__---------+
0x0402e000 |                                       +----------__XLAT_TABLE_
↪END__-----------+
0x04029000 |                                       +---------__XLAT_TABLE_
↪START__----------+
            |                                       +---------__BASE_XLAT_TABLE_
↪END__--------+
0x04028800 |                                       +--------------__BSS_END__-
↪-------------+
            |                                       +--------__BASE_XLAT_TABLE_
↪START__-------+
            |                                       +-----__PMF_PERCPU_
↪TIMESTAMP_END__------+
            |                                       +---------__PMF_TIMESTAMP_
↪END__---------+
0x04028580 |                                       +--------__PMF_TIMESTAMP_
```

(continues on next page)

(continued from previous page)

```
↪START__--------+
0x04028000 |                                       +-------------__BSS_START__
↪-------------+
0x04027e40 |                                       +------------__STACKS_END__
↪-------------+
0x04027840 |                                       +-----------__STACKS_START_
↪_------------+
0x04027000 |                                       +------------__RODATA_END__
↪-------------+
           |                                       +------------__CPU_OPS_END_
↪_------------+
           |                                       +-----------__CPU_OPS_
↪START__-----------+
           |                                       +--------__FCONF_POPULATOR_
↪END__--------+
           |                                       +--------------__GOT_END__-
↪-------------+
           |                                       +-------------__GOT_START__
↪-------------+
           |                                       +---------__PMF_SVC_DESCS_
↪END__---------+
0x04026c10 |                                       +--------__PMF_SVC_DESCS_
↪START__--------+
0x04026bf8 |                                       +-------__FCONF_POPULATOR_
↪START__-------+
           |                                       +-----------__RODATA_START_
↪_------------+
0x04026000 |                                       +-------------__TEXT_END__-
↪-------------+
0x04021000 |                                       +------------__TEXT_START__
↪-------------+
0x000062b5 +------------__BL1_ROM_END__------------+                         ␣
↪              |
0x00005df0 +----------__DATA_ROM_START__-----------+                         ␣
↪              |
           +------------__CPU_OPS_END__------------+                         ␣
↪              |
           +--------------__GOT_END__--------------+                         ␣
↪              |
           +-------------__GOT_START__-------------+                         ␣
↪              |
0x00005de8 +------------__RODATA_END__-------------+                         ␣
↪              |
           +-----------__CPU_OPS_START__-----------+                         ␣
↪              |
           +--------__FCONF_POPULATOR_END__--------+                         ␣
↪              |
           +---------__PMF_SVC_DESCS_END__---------+                         ␣
↪              |
0x00005c98 +--------__PMF_SVC_DESCS_START__--------+                         ␣
↪              |
0x00005c80 +-------__FCONF_POPULATOR_START__-------+                         ␣
```

(continues on next page)

(continued from previous page)

```
↪              |
          +-----------__RODATA_START__------------+                             ␣
↪              |
0x00005000 +-------------__TEXT_END__--------------+                             ␣
↪              |
0x00000000 +------------__TEXT_START__------------+-------------------------
↪-------------+
```

Addresses are displayed in hexadecimal by default but can be printed in decimal instead with the `-d` option.

Because of the length of many of the symbols, the tool defaults to a text width of 120 chars. This can be increased if needed with the `-w` option.

For more detailed help instructions, run:

```
poetry run memory --help
```

13.1.4 Memory Footprint

The tool enables users to view static memory consumption. When the `footprint` command is provided, the script analyses the ELF binaries in the build path to generate a table (per memory type), showing memory allocation and usage. This is the default output generated by the tool.

```
$ poetry run memory footprint
build-path: build/fvp/release
+----------------------------------------------------------------------------+
|                         Memory Usage (bytes) [RAM]                         |
+-----------+------------+------------+------------+------------+------------+
| Component |   Start    |   Limit    |    Size    |    Free    |   Total    |
+-----------+------------+------------+------------+------------+------------+
|    BL1    |    4034000 |    4040000 |       7000 |       5000 |       c000 |
|    BL2    |    4021000 |    4034000 |       d000 |       6000 |      13000 |
|    BL2U   |    4021000 |    4034000 |       a000 |       9000 |      13000 |
|    BL31   |    4003000 |    4040000 |      1e000 |      1f000 |      3d000 |
+-----------+------------+------------+------------+------------+------------+

+----------------------------------------------------------------------------+
|                         Memory Usage (bytes) [ROM]                         |
+-----------+------------+------------+------------+------------+------------+
| Component |   Start    |   Limit    |    Size    |    Free    |   Total    |
+-----------+------------+------------+------------+------------+------------+
|    BL1    |          0 |    4000000 |       5df0 |    3ffa210 |    4000000 |
+-----------+------------+------------+------------+------------+------------+
```

The script relies on symbols in the symbol table to determine the start, end, and limit addresses of each bootloader stage.

13.1.5 Memory Tree

A hierarchical view of the memory layout can be produced by passing the `tree` command to the tool. This gives the start, end, and size of each module, their ELF segments as well as sections.

```
$ poetry run memory tree
build-path: build/fvp/release
name                                          start        end       size
bl1                                               0    400c000    400c000
├── 00                                            0       5de0       5de0
│   ├── .text                                     0       5000       5000
│   └── .rodata                                5000       5de0        de0
├── 01                                      4034000    40344c5        4c5
│   └── .data                               4034000    40344c5        4c5
├── 02                                      4034500    4034a00        500
│   └── .stacks                             4034500    4034a00        500
├── 04                                      4034a00    4035600        c00
│   └── .bss                                4034a00    4035600        c00
└── 03                                      4036000    403b000       5000
    └── .xlat_table                         4036000    403b000       5000
bl2                                         4021000    4034000      13000
├── 00                                      4021000    4027000       6000
│   ├── .text                               4021000    4026000       5000
│   └── .rodata                             4026000    4027000       1000
└── 01                                      4027000    402e000       7000
    ├── .data                               4027000    4027809        809
    ├── .stacks                             4027840    4027e40        600
    ├── .bss                                4028000    4028800        800
    └── .xlat_table                         4029000    402e000       5000
bl2u                                        4021000    4034000      13000
├── 00                                      4021000    4025000       4000
│   ├── .text                               4021000    4024000       3000
│   └── .rodata                             4024000    4025000       1000
└── 01                                      4025000    402b000       6000
    ├── .data                               4025000    4025065         65
    ├── .stacks                             4025080    4025480        400
    ├── .bss                                4025600    4025c00        600
    └── .xlat_table                         4026000    402b000       5000
bl31                                        4003000    4040000      3d000
├── 02                                     ffe00000   ffe03000       3000
│   └── .el3_tzc_dram                      ffe00000   ffe03000       3000
├── 00                                      4003000    4010000       d000
│   └── .text                               4003000    4010000       d000
└── 01                                      4010000    4021000      11000
    ├── .rodata                             4010000    4012000       2000
    ├── .data                               4012000    401219d        19d
    ├── .stacks                             40121c0    40161c0       4000
    ├── .bss                                4016200    4018c00       2a00
    ├── .xlat_table                         4019000    4020000       7000
    └── .coherent_ram                       4020000    4021000       1000
```

The granularity of this view can be modified with the `--depth` option. For instance, if you only require the tree up to the level showing segment data, you can specify the depth with:

```
$ poetry run memory tree --depth 2
build-path: build/fvp/release
name                        start        end       size
bl1                             0    400c000    400c000
├── 00                          0       5df0       5df0
├── 01                    4034000    40344c5        4c5
├── 02                    4034500    4034a00        500
├── 04                    4034a00    4035600        c00
└── 03                    4036000    403b000       5000
bl2                       4021000    4034000      13000
├── 00                    4021000    4027000       6000
└── 01                    4027000    402e000       7000
bl2u                      4021000    4034000      13000
├── 00                    4021000    4025000       4000
└── 01                    4025000    402b000       6000
bl31                      4003000    4040000      3d000
├── 02                   ffe00000   ffe03000       3000
├── 00                    4003000    4010000       d000
└── 01                    4010000    4021000      11000
```

13.1.6 Memory Summary

The tool provides a by-translation-unit summary of the sizes (`text`, `bss`, `data`) contributed by each translation unit or group of translation units. For example, to print a table of an FVP build, with a path depth of 3:

```
$ poetry run memory summary build/fvp/debug/bl1/bl1.map  -d 3
 | Module                                  |          .text |        .data |     ␣
↪     .bss |
 |-----------------------------------------|----------------|--------------|-----
↪----------|
 | [fill]                                  |   3204(+3204)  |        0(+0) |     ␣
↪  97(+97) |
 | bl1/aem_generic.o                       |          0(+0) |        0(+0) |     ␣
↪    0(+0) |
 | bl1/arm_bl1_fwu.o                       |      224(+224) |      80(+80) |     ␣
↪    0(+0) |
 | bl1/arm_bl1_setup.o                     |      608(+608) |        0(+0) |     ␣
↪  17(+17) |
 | bl1/arm_common.o                        |      116(+116) |        0(+0) |     ␣
↪    0(+0) |
 | bl1/arm_console.o                       |      116(+116) |        0(+0) |     ␣
↪  40(+40) |
 | bl1/arm_dev_rotpk.o                     |          0(+0) |        0(+0) |     ␣
↪    0(+0) |
 | bl1/arm_dyn_cfg.o                       |      276(+276) |        0(+0) |  ␣
↪7184(+7184) |
 | bl1/arm_dyn_cfg_helpers.o               |      364(+364) |        0(+0) |     ␣
↪    0(+0) |
 | bl1/arm_err.o                           |        12(+12) |        0(+0) |     ␣
```

(continues on next page)

(continued from previous page)

```
↪    0(+0) |
| bl1/arm_fconf_io.o                         |         0(+0) |    952(+952) |     ␣
↪    0(+0) |
| bl1/arm_helpers.o                          |       44(+44) |        0(+0) |     ␣
↪    0(+0) |
| bl1/arm_io_storage.o                       |     480(+480) |        0(+0) |     ␣
↪  32(+32) |
| bl1/auth_mod.o                             |   1288(+1288) |        0(+0) |     ␣
↪    0(+0) |
| bl1/backtrace.o                            |     444(+444) |        0(+0) |     ␣
↪    0(+0) |
| bl1/bl1_arch_setup.o                       |       16(+16) |        0(+0) |     ␣
↪    0(+0) |
| bl1/bl1_context_mgmt.o                     |     340(+340) |        0(+0) |   ␣
↪1392(+1392) |
| bl1/bl1_entrypoint.o                       |     236(+236) |        0(+0) |     ␣
↪    0(+0) |
| bl1/bl1_exceptions.o                       |   2240(+2240) |        0(+0) |     ␣
↪    0(+0) |
| bl1/bl1_fwu.o                              |   2188(+2188) |      44(+44) |     ␣
↪    0(+0) |
| bl1/bl1_main.o                             |     620(+620) |        0(+0) |     ␣
↪    0(+0) |
| bl1/bl_common.o                            |     772(+772) |        0(+0) |     ␣
↪    4(+4) |
| bl1/board_arm_helpers.o                    |       44(+44) |        0(+0) |     ␣
↪    0(+0) |
| bl1/board_arm_trusted_boot.o               |       44(+44) |      16(+16) |     ␣
↪    0(+0) |
| bl1/cache_helpers.o                        |     112(+112) |        0(+0) |     ␣
↪    0(+0) |
| bl1/cci.o                                  |     408(+408) |        0(+0) |     ␣
↪  24(+24) |
| bl1/context.o                              |     348(+348) |        0(+0) |     ␣
↪    0(+0) |
| bl1/context_mgmt.o                         |   1692(+1692) |        0(+0) |     ␣
↪  48(+48) |
| bl1/cortex_a35.o                           |       96(+96) |        0(+0) |     ␣
↪    0(+0) |
| bl1/cortex_a53.o                           |     248(+248) |        0(+0) |     ␣
↪    0(+0) |
| bl1/cortex_a57.o                           |     384(+384) |        0(+0) |     ␣
↪    0(+0) |
| bl1/cortex_a72.o                           |     356(+356) |        0(+0) |     ␣
↪    0(+0) |
| bl1/cortex_a73.o                           |     304(+304) |        0(+0) |     ␣
↪    0(+0) |
| bl1/cpu_helpers.o                          |     200(+200) |        0(+0) |     ␣
↪    0(+0) |
| bl1/crypto_mod.o                           |     380(+380) |        0(+0) |     ␣
↪    0(+0) |
| bl1/debug.o                                |     224(+224) |        0(+0) |     ␣
```

(continues on next page)

(continued from previous page)

```
↪     0(+0)  |
 | bl1/delay_timer.o                       |       64(+64)  |        0(+0)  |     ␣
↪     8(+8)  |
 | bl1/enable_mmu.o                        |     112(+112)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/errata_report.o                     |     564(+564)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/fconf.o                             |     148(+148)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/fconf_dyn_cfg_getter.o              |     656(+656)  |      32(+32)  |     ␣
↪144(+144)  |
 | bl1/fconf_tbbr_getter.o                 |     332(+332)  |        0(+0)  |     ␣
↪   24(+24)  |
 | bl1/fdt_wrappers.o                      |     452(+452)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/fvp_bl1_setup.o                     |     168(+168)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/fvp_common.o                        |     512(+512)  |        0(+0)  |     ␣
↪     8(+8)  |
 | bl1/fvp_cpu_pwr.o                       |     136(+136)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/fvp_err.o                           |       44(+44)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/fvp_helpers.o                       |     148(+148)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/fvp_io_storage.o                    |     228(+228)  |        0(+0)  |     ␣
↪   16(+16)  |
 | bl1/fvp_trusted_boot.o                  |     292(+292)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/generic_delay_timer.o               |     136(+136)  |        0(+0)  |     ␣
↪   16(+16)  |
 | bl1/img_parser_mod.o                    |     588(+588)  |        0(+0)  |     ␣
↪   20(+20)  |
 | bl1/io_fip.o                            |   1332(+1332)  |        0(+0)  |     ␣
↪100(+100)  |
 | bl1/io_memmap.o                         |     736(+736)  |      16(+16)  |     ␣
↪   32(+32)  |
 | bl1/io_semihosting.o                    |     648(+648)  |      16(+16)  |     ␣
↪     0(+0)  |
 | bl1/io_storage.o                        |   1268(+1268)  |        0(+0)  |     ␣
↪104(+104)  |
 | bl1/mbedtls_common.o                    |     208(+208)  |        0(+0)  |     ␣
↪     4(+4)  |
 | bl1/mbedtls_crypto.o                    |     636(+636)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/mbedtls_x509_parser.o               |   1588(+1588)  |        0(+0)  |     ␣
↪120(+120)  |
 | bl1/misc_helpers.o                      |     392(+392)  |        0(+0)  |     ␣
↪     0(+0)  |
 | bl1/multi_console.o                     |     528(+528)  |        1(+1)  |     ␣
↪     8(+8)  |
 | bl1/pl011_console.o                     |     308(+308)  |        0(+0)  |     ␣
```

(continues on next page)

(continued from previous page)

```
↪    0(+0)  |
| bl1/plat_bl1_common.o                    |     208(+208)  |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/plat_bl_common.o                     |      40(+40)   |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/plat_common.o                        |      48(+48)   |        0(+0)  |    ␣
↪    8(+8)  |
| bl1/plat_log_common.o                    |      48(+48)   |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/plat_tbbr.o                          |     128(+128)  |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/platform_helpers.o                   |      12(+12)   |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/platform_up_stack.o                  |      16(+16)   |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/semihosting.o                        |     352(+352)  |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/semihosting_call.o                   |       8(+8)    |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/smmu_v3.o                            |     296(+296)  |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/sp805.o                              |      64(+64)   |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/tbbr_cot_bl1.o                       |       0(+0)    |      48(+48)  |    ␣
↪156(+156)  |
| bl1/tbbr_cot_common.o                    |       0(+0)    |    144(+144)  |    ␣
↪306(+306)  |
| bl1/tbbr_img_desc.o                      |       0(+0)    |    768(+768)  |    ␣
↪    0(+0)  |
| bl1/tf_log.o                             |     200(+200)  |        4(+4)  |    ␣
↪    0(+0)  |
| bl1/xlat_tables_arch.o                   |     736(+736)  |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/xlat_tables_context.o                |     192(+192)  |      96(+96)  |  ␣
↪1296(+1296)  |
| bl1/xlat_tables_core.o                   |    2112(+2112) |        0(+0)  |    ␣
↪    0(+0)  |
| bl1/xlat_tables_utils.o                  |       8(+8)    |        0(+0)  |    ␣
↪    0(+0)  |
| lib/libc.a/assert.o                      |      48(+48)   |        0(+0)  |    ␣
↪    0(+0)  |
| lib/libc.a/exit.o                        |      64(+64)   |        0(+0)  |    ␣
↪    8(+8)  |
| lib/libc.a/memchr.o                      |      44(+44)   |        0(+0)  |    ␣
↪    0(+0)  |
| lib/libc.a/memcmp.o                      |      52(+52)   |        0(+0)  |    ␣
↪    0(+0)  |
| lib/libc.a/memcpy.o                      |      32(+32)   |        0(+0)  |    ␣
↪    0(+0)  |
| lib/libc.a/memmove.o                     |      52(+52)   |        0(+0)  |    ␣
↪    0(+0)  |
| lib/libc.a/memset.o                      |     140(+140)  |        0(+0)  |    ␣
```

(continues on next page)

(continued from previous page)

```
↪    0(+0) |
| lib/libc.a/printf.o                    |   1532(+1532) |        0(+0) |    ␣
↪    0(+0) |
| lib/libc.a/snprintf.o                  |   1748(+1748) |        0(+0) |    ␣
↪    0(+0) |
| lib/libc.a/strcmp.o                    |       44(+44) |        0(+0) |    ␣
↪    0(+0) |
| lib/libc.a/strlen.o                    |       28(+28) |        0(+0) |    ␣
↪    0(+0) |
| lib/libfdt.a/fdt.o                     |   1460(+1460) |        0(+0) |    ␣
↪    0(+0) |
| lib/libfdt.a/fdt_ro.o                  |   1392(+1392) |        0(+0) |    ␣
↪    0(+0) |
| lib/libfdt.a/fdt_wip.o                 |     244(+244) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/asn1parse.o           |     956(+956) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/bignum.o              |   6796(+6796) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/bignum_core.o         |   3252(+3252) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/constant_time.o       |     280(+280) |        0(+0) |    ␣
↪    8(+8) |
| lib/libmbedtls.a/md.o                  |     504(+504) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/memory_buffer_alloc.o |   1264(+1264) |        0(+0) |    ␣
↪  40(+40) |
| lib/libmbedtls.a/oid.o                 |     752(+752) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/pk.o                  |     872(+872) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/pk_wrap.o             |     848(+848) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/pkparse.o             |     516(+516) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/platform.o            |       92(+92) |      24(+24) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/platform_util.o       |       96(+96) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/rsa.o                 |   6588(+6588) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/rsa_alt_helpers.o     |   2340(+2340) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/sha256.o              |   1448(+1448) |        0(+0) |    ␣
↪    0(+0) |
| lib/libmbedtls.a/x509.o                |   1028(+1028) |        0(+0) |    ␣
↪    0(+0) |
| Subtotals                              | 69632(+69632) | 2241(+2241) |␣
↪11264(+11264) |
Total Static RAM memory (data + bss): 13505(+13505) bytes
Total Flash memory (text + data): 71873(+71873) bytes
```

A delta between two images can be generated by passing the `--old` option with a path to the previous map file.

For example:

```
$ poetry run memory summary ../maps/fvp-tbb-mbedtls/bl1.map --old ../maps/fvp-
↪tbb-mbedtls/bl1.map.old -d 1
| Module    |          .text |     .data |          .bss |
|-----------|----------------|-----------|---------------|
| [fill]    |     780(-2424) |     0(+0) |     321(+224) |
| bl1       |    32024(+108) | 2217(+0)  |     11111(+0) |
| lib       | 45020(+10508)  |    24(+0) |   1880(+1824) |
| Subtotals |   77824(+8192) | 2241(+0)  | 13312(+2048)  |
Total Static RAM memory (data + bss): 15553(+2048) bytes
Total Flash memory (text + data): 80065(+8192) bytes
```

Note that since the old map file includes the required suffix, specifying the `--old` argument is optional here.

Under some circumstances, some executables are padded to meet certain alignments, such as a 4KB page boundary, and excluding that padding can provide more helpful diffs. Taking the last example, and adding the `-e` argument yields such a summary:

```
$ poetry run memory summary ../maps/fvp-tbb-mbedtls/bl1.map --old ../maps/fvp-
↪tbb-mbedtls/bl1.map.old  -d 1 -e
| Module    |          .text |     .data |          .bss |
|-----------|----------------|-----------|---------------|
| bl1       |    32024(+108) | 2217(+0)  |     11111(+0) |
| lib       | 45020(+10508)  |    24(+0) |   1880(+1824) |
| Subtotals | 77044(+10616)  | 2241(+0)  | 12991(+1824)  |
Total Static RAM memory (data + bss): 15232(+1824) bytes
Total Flash memory (text + data): 79285(+10616) bytes
```

13.2 TF-A CoT dt2c Tool

This tool is used to automatically generate the corresponding c file for a CoT DT file. Since currently TF-A support two type of CoT file: static c file and CoT DT binding. This is error prone and hard to maintain, therefore this tool can generate the c file for the platform that does not support CoT DT binding, given the CoT DT file so the c file can be deprecated.

13.2.1 Prerequisites

1. Python (3.8 or later)
2. Poetry Python package manager

13.2.2 Getting Started

`cot-dt2c` is installed by default with TF-A's poetry environment. All of it's dependencies are listed in tools/cot_dt2c/pyproject.toml.

`cot-dt2c` requires a standard DTS file without #ifdef, macros, or other preprocessor directives. Therefore, you need to provide a preprocessed device tree source(DTS) as input to the tool.

1. Usage of the tool

   ```
   cot-dt2c
   ```

   This command will output the following as usage for this command

   ```
   Usage: cot-dt2c [OPTIONS] COMMAND [ARGS]...

   Options:
   --version  Show the version and exit.
   --help     Show this message and exit.

   Commands:
   convert-to-c
   validate-cot
   visualize-cot
   validate-dt
   ```

13.2.3 Convert CoT descriptors to C file

To convert the CoT descriptors

This command is for the platform that does not use CoT DT parser, which can generate the C file given the CoT descriptors. Before the conversion to C file, the tool will do an implicit checks on the validity of the CoT DT file.

```
cot-dt2c convert-to-c [INPUT DTS PATH] [OUTPUT C PATH]
cot-dt2c convert-to-c fdts/tbbr_cot_descriptors.dts test.c
```

13.2.4 Validate CoT descriptors

To validate the certificate

The tests folder in the tool folder provides some bad-example of the DT file, and the tool will print out "not a valid CoT DT file" on console.

The command will check the format of the CoT file

1. The open bracket
2. The open ifdef macro
3. The missing mandatory attribute
4. Malformed DT file (cert missing parent, missing root certs. etc.)

Currently the validation is specifically for checking the CoT DT file

```
cot-dt2c validate-cot [INPUT DTS PATH]
cot-dt2c validate-cot fdts/tbbr_cot_descriptors.dts
```

13.2.5 Visualize CoT descriptors

This command create a HTML to visualize the relationship between the certificates and the image of a CoT DT file.

```
cot-dt2c visualize-cot [INPUT DTS PATH]
cot-dt2c visualize-cot fdts/tbbr_cot_descriptors.dts
```

13.2.6 Validate Other DT files

The command will transform the dtsi/dts file into a more standard dtsi/dts file inside /tmp folder that can be used as input to dt-schema for further validation. Currently the tool will perform some basic validation for the file (syntax) and dt-schema can be used for advance checks. dt-schema is not installed along with the tool.

```
cot-dt2c validate-dt [INPUT DTS PATH or INPUT DTS folder]
cot-dt2c validate-dt fdts/
cot-dt2c validate-dt fdts/fvp-bsae-gicv3.dtsi
```

CHAPTER FOURTEEN

CHANGE LOG & RELEASE NOTES

This document contains a summary of the new features, changes, fixes and known issues in each release of Trusted Firmware-A.

14.1 2.15.0 (2026-05-27)

14.1.1 ? BREAKING CHANGES

- **Architecture**
  - **CPU feature like FEAT_XXXX / ID register handling in general**
    - do not enable FEAT_RNG with v8.5

      See: do not enable FEAT_RNG with v8.5 (6d39125)
- **Bootloader Images**
  - **BL2**
    - This patch also changes all existing platform files and functions that use format bl2_el3_* to bl2_plat helpers. If any platform or out-of-tree platforms that need to support running BL2 in EL1 or EL3 must now handle it in bl2_early_platform_setup2 and bl2_plat_arch_setup.

      See: support RESET_TO_BL2 and ENABLE_RME (8c82427)
- **Services**
  - **RME**
    - **RMMD**
      - RMM v1.x compatibility is now disabled by default. Platforms which continue to rely on the RMM v1 ABI must explicitly build TF-A with `RMM_V1_COMPAT=1`.

        See: set RMM_V1_COMPAT=0 (b986631)
- **Libraries**
  - **CPU Support**

      * make ERRATA_SME_POWER_DOWN work with the recommended state ID

        See: make ERRATA_SME_POWER_DOWN work with the recommended state ID (93c7e70)

- **Drivers**

  - **Arm**

    * **RSE**

      · platforms can no longer retrieve the host ROTPK from the RSE as these are no longer provisioned.

        See: remove host ROTPK support and test (bd14181)

- **Build System**

  - Mbed TLS is now included in the TF-A repository, and it is no longer a requirement to pass `MBEDTLS_DIR` to the build system. Please run `git submodule update --init --recursive` if you encounter issues after migrating to the latest version of TF-A.

    See: add Mbed TLS submodule (bc9a699)

  - LTO has been enabled by default, which may cause unpredictable issues for platforms where the linker scripts have not been designed with LTO in mind. Please report any issues to the mailing list.

    See: enable link-time optimization by default (fa28b3a)

14.1.2 New Features

- **Architecture**

  - **CPU feature like FEAT_XXXX / ID register handling in general**

    * add support for FEAT_HACDBS (c2d6bbd)
    * add support for FEAT_HDBSS (7e58ab3)
    * add support for FEAT_STEP2 (b6cf126)
    * add support for FEAT_UINJ (4286d16)
    * add the newly analyzed features to FEATURE_DETECTION (5eceb40)
    * advertise support for FEAT_RASv2 (caf00e1)
    * bump FEATURE_DETECTION features up to 9.5 (bef4849)
    * constrain RAS_TRAP_NS_ERR_REC_ACCESS on ENABLE_FEAT_RAS (0e3e882)
    * enable FEAT_RAS for FEAT_STATE_CHECKED again (553c24c)
    * enable mandatory Armv9.4–Armv9.6 features by default (9838436)
    * enable USE_SPINLOCK_CAS to FEAT_STATE_CHECKED (38e580e)
    * update FEAT_SB's FEAT_STATE_CHECKED status (9dda408)

- **Platforms**
  - **AMD**
    - custom package linker infra (89aae9a)
    - support multiple custom packages (87daf65)
    - **Versal Gen 2**
      - add build macro support for IPI_ID_APU (6244ccf)
      - add dynamic fetching of core and cluster (31dac54)
      - add PM support for 1-cluster 4-core topology (0994aef)
      - add static topology via build arg (c0bb80e)
      - support alternate core as primary (non-cpu0) (90cdb04)
      - update plat psci based on dynamic core and cluster (ea435c8)
      - use runtime core/cluster variables for dyn topology (a6d4318)
  - **Arm**
    - allow custom BL2 mem params (7622cec)
    - prefer GUID-based GPT partition lookup (d4e8772)
    - **Common**
      - introduce SMC validation framework (92d6ae6)
    - **FVP**
      - enable discovery of two Secure Partitions for live activation (bec6bd0)
      - enable FEATURE_DETECTION on FVP (69871af)
      - extend image decryption support for FVP (d81b3bc)
      - fix PLAT_ARM_FW_HANDOFF_SIZE (ac18ce2)
      - implement SP live activation callback (5a76376)
      - introduce SP live activation component manager (4cd4918)
      - load SP_PKGs with TRANSFER_LIST (6ae88e2)
      - pack/load tb-fw cert with ARM_FW_CONFIG_LOAD_ENABLE (636654d)
    - **Juno**
      - increase BL2 stack size when MEASURED_BOOT (1e68092)
      - increase size of PLAT_ARM_EVENT_LOG_MAX_SIZE (66c0674)
    - **Morello**
      - add capability load/store/track support to MMU (6a548c3)
      - add missing is_feat_amu_present_asm function (3d3d1d8)

        - add Morello capability enablement changes (27bc138)

    - **Neoverse-RD**
        - set the correct Arm version for rdn2 (b928b7f)
        - **RD-V3**
            - use SFCP PSA call instead of RSE comms (66e46af)

    - **TC**
        - add fpga flash regions in spmc manifest (66d0d75)
        - add tc_sfcp.c (05076cb)
        - enable workaround for CVE-2026-0995 (0bc52ef)
        - set Arm CoreSight context loss property (b79eb3e)
        - support 6mid2big CPU topology for FPGA (f84905d)
        - support buildroot ddk runtime integration (743de0e)
        - use SFCP PSA call instead of RSE comms (65a4925)

    - **Corstone-1000**
        - add Cortex-A320 support (25148ce)
        - make mutlicore support platform generic (5c9b030)

    - **Automotive RD**
        - **RD-Aspen**
            - add DT buffer and IRQ setup (761d0c7)
            - dump the CPER buffer contents (96f40c7)
            - enable measured boot (0bf4d2b)
            - event handler for CPU RAS (0702fe7)
            - generate CPER at TF-A EL3 (cbad38f)
            - intr RAS handling for PC CPU (3f3b9ec)

- **Marvell**
    - add odyssey platform header files (4b8b8d7)
    - **Armada**
        - **A8K**
            - add a80x0_nbx Free Mobile board (88a6e61)
            - add user callback for skip_image (7b46c62)

- **MediaTek**
    - support write-access for DSU PMU in EL1 (eb949d9)

  * **MT8196**
    - add audio SMC cmd implementation (6866675)
    - add booker driver (997eba3)
- **NXP**
  * **i.MX**
    - **i.MX 8M**
      - keep console at runtime when building TF-A/bl31 with DEBUG (d7f0864)
    - **i.MX 9**
      - **i.MX93**
        - enable the s401 clock on/off handshake (38db815)
        - optimize power switch acknowledgment timing (7e8caac)
        - retrieve SoC info from EdgeLock Enclave (78df8f7)
  * **S32G274A**
    - **S32G274ARDB**
      - add DDR FW binary check (4c117de)
      - add DDR FW entry to storage (ced66a7)
      - add DDR init call (83971c4)
      - add DDR PHY mailbox support (30c8a20)
      - add DDR post training setup (a60aeae)
      - add DDR register accessories (a4efd42)
      - add DDR training stubs (5423906)
      - add support for DDR img in FIP (ce775d8)
      - add top-level DDR init (e0b79c5)
      - add training for 1D and 2D (47f0a59)
      - document DDR integration (a482ee2)
      - load BL31 and BL33 to DDR (7ba9e47)
      - map regions used by DDR driver (bfcb0fd)
      - use DDR reset deassertion (717ab11)
- **QEMU**
  * disable fpregs traps for QEMU in BL31 (949b0d4)
  * enable ENABLE_FEAT_RAS and ENABLE_FEAT_SB (996d08b)
- **QTI**

    * add support for Lemans SoC (b4d4a10)
    * extend interrupt dispatcher (4cf39ff)
    * integrate access control stub (c533f76)
    * integrate qtimer stub (41df20a)
    * integrate sec_core stub (88f1171)
    * integrate watchdog stub (e1ef7e6)
    * port SMMU driver from QTISECLIB (5534dad)
    * reorganize bear-based SoCs under a common family directory (22170af)
    * reorganize hoya-based SoCs under a common family directory (f575472)
    * **Kodiak**
      - add access control driver (6ae8fce)
      - add qtimer driver (9743b9d)
      - add sec_core driver (a757f5f)
      - add watchdog driver (349fe8c)
      - build XPU bypass only when bypass is enabled (0510141)
    * **Lemans**
      - add support for lemans EVK platform (cba81c3)
      - expose dcache flush all callback to QTISECLIB (a9d7f98)
  - **Raspberry Pi**
    * **Raspberry Pi 3**
      - use transfer list to carry the event log (2de6d44)
  - **Renesas**
    * **R-Car**
      - add initial BL31 support for Renesas R-Car X5H (f180a3b)
      - fold console_rcar_* functions into rcar_printf.c (4a6b037)
      - rewrite console_renesas_register() in C (ded1b9c)
      - rewrite SCIF driver from assembler to C (656a856)
      - test only for SCIF TX FIFO empty before writing the FIFO (c83fb4c)
      - **R-Car 3**
        - add R-Car Gen3 M3Le R8A779MD SoC and Geist board support (ac41faf)
        - rephrase RCAR_BL31_CRASH_BASE definition (ecf9c23)
    * **RZ**

        · **RZ/A**

            · add initial BL2 support for RZ/A platforms (2aa6335)

- **Rockchip**
  - **RK3568**
    - add early CPU reset mechanism to enable crypto function (0738365)
    - bring DDR firewall naming in line (5e4a490)
    - move existing secure init to separate files (1dc8eb9)
    - protect TF-As memory area with the DDR firewall (1248db7)
  - **RK3588**
    - report actual measured PVTPLL clocks (d2d6928)
- **ST**
  - add a halt on panic flag (041b406)
  - add stm32mp_gic_cpuif_enable/disable (7fa282a)
  - add STM32MP_SUPPORT_PM flag (0870faa)
  - allow the use of clang as linker (abf5915)
  - disable FWU on serial boot devices (25bfb4a)
  - **STM32MP1**
    - make stpmic2 usable for STM32MP1 (d404b27)
    - remove stop watchdog when going in low power (8db6a92)
    - rework ddr supplies handling (ae25a7b)
    - update STPMIC1 driver to handle more properties (f86b751)
  - **STM32MP2**
    - add a ca35ss driver (48545b3)
    - add a new stm32mp21 power compatible (8621405)
    - add RIFSC USB configuration for variants (052e0c0)
    - add watchdog support (9529933)
    - define UART8 and UART9 under STM32MP25 flag (6715534)
    - enable watchdog driver (258c471)
    - include dedicated RIFSC ID headers (c7143a4)
    - introduce MCE encryption level in RISAF dt-bindings (00da809)
    - manage core 1 enabling (376ac16)
    - manage RISAF MCE key for STM32MP21 (7a563a6)

- remove function stm32_iwdg_shadow_update() (c455c1b)
- stub PM code in serial boot (e976546)
- **STM32MP21**
  - add RIFSC IDs bindings (ad1fca2)
  - add SoC part numbers (3d797c7)
  - display package type (44b96cd)
  - enable USBOTG DFU (838de22)
  - manage STM32MP21 DT overlays (e0a6bfe)
  - remove GPIOJ and GPIOK (b112cd9)
  - update boot API header version (c91fd51)
  - update SRAM1 and RETRAM base address (185252d)
  - update STM32 header size for STM32MP21 (76043d8)
- **STM32MP23**
  - add RIFSC IDs bindings (f9c665d)
  - add SoC part numbers (d80e7fb)
  - display package type (260b1c2)
  - manage STM32MP23 DT overlays (080a672)

– **Texas Instruments**

- add clock device integration to TI clock framework (cd88365)
- add clock operation request handler (367abca)
- add device group filtering to clock initialization (574e3fe)
- add device state request handler (61dbb0c)
- add message to encrypt tfa during suspend (ea83fd8)
- add support for scmi power and clock mgmt (b5c7f3b)
- add support for TISCI firewall APIs (481aabb)
- add TI 16FFT PLL driver (757df43)
- add TI clock divider driver (243873a)
- add TI clock management framework core (6b875d3)
- add TI clock multiplexer driver (8a46d04)
- add TI core device management driver (43b5c09)
- add TI device clock driver and power domain API (6f2a879)
- add TI device power management driver (92ee37f)

    * add TI device preparation and validation driver (bc55e13)
    * add TI device PSC integration layer (a1a3a16)
    * add TI fixed clock driver (db8a399)
    * add TI generic PLL driver and control (28c0633)
    * add TI high-frequency oscillator driver (73c2e6c)
    * add TI low-frequency oscillator driver (aabf6fe)
    * add TI PSC hardware driver (f6db41e)
    * cache the response for msg_version and query_caps (a9d0d6a)
    * platform_defs: increase BL31_SZ (1022574)
    * **K3**
      - add SoC detection support (e76a0fc)
      - choose PLAT_CLUSTER_DEVICE_START_ID depending on the SoC (9b1350d)
      - handle suspend in case of LPM_BOARDCFG_MANAGED (76500ce)
    * **K3 Low Power Devices**
      - add AM62L DDR platform shim and EVM board config (9527667)
      - add BL1 platform definitions and integration for AM62L (d26945f)
      - add board specific power management drivers and data (3fd3906)
      - introduce Cadence LPDDR4 core driver for AM62L (5421f84)
      - map devctrl regs (a4e6321)
      - optimize firewall configuration to minimize boot time (ad6ffb1)
      - re-configure firewalls for TI AM62L (6af16d1)
      - scmi: init scmi server (75d75b5)
      - support SCMI protocol on k3low (1b2b4b4)
  - **Xilinx**
    * **Versal**
      - add build macro support for IPI_ID_APU (27746e4)
      - add support for TFA_NO_PM configuration (e76ac8e)
    * **Versal NET**
      - add build macro support for IPI_ID_APU (ab8120a)
    * **ZynqMP**
      - enable PM_QUERY_DATA bitmask merging with FW (dfe8d0f)
      - forward unhandled PM_QUERY_DATA to firmware (3266844)

        · sync PM query IDs with firmware (0d2ec61)

- **Bootloader Images**
  - **BL2**
    * support RESET_TO_BL2 and ENABLE_RME (8c82427)
- **Services**
  - **RME**
    * split off ENABLE_FEAT_RME (dfdbda0)
    * **TRP**
      · add support for rmm 2.0 abis (da1a043)
    * **RMMD**
      · expand RMM SMC return registers (01c4821)
      · extend RMI function ID range validation (bd5ee0b)
      · replace ENABLE_RME with ENABLE_RMM (b0ddba2)
  - **SPM**
    * **EL3 SPMC**
      · support 16 args for FFA_CONSOLE_LOG_SMC64 (cdb520e)
    * **SPMD**
      · helpers for SP live activation framework messages (0080c2c)
      · support FFA DIRECT_REQ2 for logical SP (065fa8a)
      · support for extended partition info descriptor (60cef66)
    * **SPM MM**
      · fix wrong range of SPM_MM (6c79953)
  - **DRTM**
    * check for DLME image auth plat support (c743d87)
  - **FIRME**
    * initial commit of FIRME service (c359aeb)
  - **ChromeOS**
    * add GSC, DRM, and Stable HUK SMC handlers (33fc163)
  - **Live Firmware Activation**
    * build flags for SP live activation support (f5cb144)
- **Libraries**
  - add libtpm submodule (92311ae)

- add mmio_setbits_64 inline function (184e787)
- add u64 overflow helper (6d99bc0)
- bump event log library (a04f67e)
- use C/assembler for HI/LO macros (2d05494)
- **CPU Support**
  * add support for LSC25 E-core CPU (bff6e60)
  * add support for LSC25 P-core CPU (e1fbad0)
  * add support for Rosillo cpu (c9017cb)
  * add sysreg_lazy_* macros for batched read-modify-write (b5b5769)
- **EL3 Runtime**
  * check the exception vector size (8df6a7c)
  * translate EL3 handled exceptions to C and always call prepare_el3_entry (14320bc)
  * **Context Management**
    · propagate EL2 register compatibility init to all worlds (5cadaf9)
- **Per-cpu**
  * migrate AArch32 amu_ctx to per-cpu framework (ef545e8)
- **PSCI**
  * add psci_cpu_off_start event (e0b614c)
- **Granule Protection Tables**
  * move gpt support under ENABLE_FEAT_RME (d63c296)
- **SMCCC**
  * add support for CPU Service calls (7a42d35)
- **Locks**
  * make spin_trylock with exclusives spin until it knows the state of the lock (07ba153)
- **Firmware Handoff**
  * bump implementation to spec v2.0 (933248b)

• **Drivers**
- **Authentication**
  * **Crypto**
    · build flag for SIMD crypto extensions for v8+ platform (e8cc970)
    · enable access to SIMD crypto in BL1 and BL2 (78efac7)
    · enable floating point register traps in EL3 (96f227b)

        - enable SIMD crypto extensions for S-EL1 (3f8cf8a)
        - enable the runtime instrumentation for crypto extension (993c004)
    - **mbedTLS**
        - enable AESCE under ENABLE_FEAT_CRYPTO for AES-GCM (79e9edb)
        - update mbedTLS to version 3.6.6 (28aa61d)
- **Generic Clock**
    - add get_possible_parents_num callback (e9f69b9)
- **Measured Boot**
    - enable dynamic hash provisioning (265f148)
- **GUID Partition Tables Support**
    - add type+index GPT lookup (841f85e)
- **TPM**
    - add TPM/TCG hashing helper to crypto module (7bbb008)
    - changes to support TPM lib (6963f71)
- **Arm**
    - **RSE**
        - fix iovec parameter check in rse comms (785b7df)
    - **Simple Firmware Communications Protocol (SFCP)**
        - add SFCP stack and PSA call (479e264)
- **ST**
    - **BSEC**
        - add STM32MP21 version support (b499da4)
        - update OTPCR masks for STM32MP21 (e0cd2df)
    - **Clock**
        - add barriers in clock driver (bc4ccae)
        - add ck_sys_dbg clock for STM32MP2 (ebece79)
        - add trace for unknown clock identifier (78dbe68)
        - keep TAMP_BKP_REG_CLK always on (3218d2f)
    - **Resource Isolation Framework**
        - add RISAF MCE support for STM32MP21 (f22005f)
        - add STM32MP2x support in RIFSC driver (3549f1a)
    - **ST PMIC**

          - add a switch-off function (fb04a0f)
          - add default regulator configuration for enable_ramp_delay (b1d732b)
          - add support for STPMIC1L and STPMIC2L (8325976)
          - display information only in BL2 (dfdb6c2)
          - force STPMIC1 and its I2C secure in BL2 (d72003f)
        - **USB**
          - enable USBOTG DFU for STM32MP21 (f57e492)
        - **Watchdog**
          - remove unused flags (c3ac6ca)
          - set watchdog timeout from DT (b6a26c3)
          - start the watchdog on status okay (c53dd82)
- **Miscellaneous**
  - **FDTs**
    - **ST**
      - **STM32MP1**
        - add DDR power supplies description (ad8f869)
        - introduce STM32MP21 pinctrl files (ad47dc7)
        - use iwdg1 instead of iwdg2 (84b68c6)
        - **STM32MP15**
          - add STM32M157F boards (416773d)
      - **STM32MP2**
        - add watchdog support for STM32MP2 (1582f1c)
        - remove ramp delay on ddr supply (88ae2e1)
        - **STM32MP21**
          - add fw-config file (ed717ea)
          - add pinctrl DT file (f262ebf)
          - add SoC DT files (b4cf410)
          - add STM32MP215F-DK board (5d6034a)
        - **STM32MP23**
          - add fw-config file (b1fb4d8)
          - add SoC DT files (bf4f90b)
          - add STM32MP235F-DK board (b841fb6)

            · **STM32MP25**
                · add iwdg1 on dk board (5ff6e86)
                · add iwdg1 on eval board (3674e0b)

- **Documentation**
  - document PSCI power_state (9d0f944)
  - update the feature guide to mention FEAT_IDTE3 (3ced4f3)
- **Build System**
  - add HOSTLDFLAGS to pass flags to host links (0607fb7)
  - add Mbed TLS submodule (bc9a699)
  - add size metadata for each vector_entry function (55fbbeb)
  - enable link-time optimization by default (fa28b3a)
- **Tools**
  - **STM32 Image**
    * add header version 2.3 support (9f3b4fb)
  - **Secure Partition Tool**
    * align SP uuids to byte words (a46cca0)
  - **Certificate Creation Tool**
    * make SoC AP firmware hash extension optional (62b9afa)

14.1.3 Resolved Issues

- **Architecture**
  - **CPU feature like FEAT_XXXX / ID register handling in general**
    * account for FEAT_RME registers in FEAT_FGWTE3 (98dee19)
    * add arch_extension directives to bit_lock() too (6472270)
    * add FEAT_SPE to FEATURE_DETECTION (b4f47d8)
    * always allow accesses of FEAT_RAS registers (4945956)
    * always provide pauth context helper (650bfcb)
    * correctly set PSTATE.NZCV on trapped RNDR(RS) (fb6218d)
    * disable FEAT_AMU counters on context restore (753c749)
    * disable FEAT_RME_GPC3 by default (34ca93d)
    * do not enable FEAT_RNG with v8.5 (6d39125)
    * drop feature_panic() as unused (8145e2f)

    * enable access to extended BRPs/WRPs (2edb8b6)
    * enable FEAT_FGWTE3 after FEAT_CPA (3840242)
    * fix compiler warning in feature detection (8b4b054)
    * give `stxr` distinct src and ret registers (b0c7709)
    * prevent FEAT_AMU counters 2 and 3 from counting across worlds (8cd9c18)
    * remove the feature list from arch_features.h (ab04322)
    * require FEAT_AMUv1p1 to enable the auxiliary counters (6edbd2d)
    * simplify AArch32 feature disablement (3ca44b8)
    * update feature names and comments (3a6e53c)
    * use correct inline assembly constraints (e080eee)

- **Platforms**
  - remove circular dependency on ENABLE_FEAT_RAS (813bfe5)
  - **AMD**
    * update topology before GIC init (dbba773)
    * **Versal Gen 2**
      - add prototype for external linkage (d9435e4)
      - buffer overflow due to missing bounds check (c9597b2)
      - correct NS entry point check for reserved regions (8b90e80)
      - fix misra rule 10.3 violations (1673f4d)
      - fix misra rule 10.4 violations (7645bde)
      - fix misra rule 18.1 violations (866cfa8)
      - fix misra rule 2.5 violations (4853076)
      - fix misra rule 2.7 violations (59bc7c6)
      - fix misra rule 4.6 violations (f68dc7b)
      - fix misra rule 5.7 violations (0719f9f)
      - fix misra rule 5.9 violation (155a819)
      - fix misra rule 8.13 violations (4c7f7bc)
      - fix misra rule 8.4 violation (b799622)
      - fix violations for unreferenced functions (a9e1fd1)
      - remove SCMI drivers in NO PM configuration (d27a323)
      - resolve type mismatch (9121cc9)
      - update IRQ_MAX for SDIO interrupts (b542478)

· use 64-bit macros for BL31 checks (7a7b298)

- **Arm**
  * bound backup GPT spec length (a37d078)
  * build fails on RESET_TO_BL2=1 and ARM_FW_CONFIG_LOAD_ENABLE=1 (7cc8f16)
  * clean up FEAT_RME #ifdefs (62759f7)
  * harden FWU address range checks (08c7f0e)
  * remove unused scu driver (e2504d4)
  * rearrange the include order (26fa09d)
  * support FCONF when TRANSFER_LIST and RESET_BL2 is set (a36ee52)
  * update next image's ep info with the FW config address (010f458)
  * **CSS**
    · fix SCP to AP shared memory address (221cf4b)
  * **FVP**
    · correctly describe the IRS as non-coherent (d4d6433)
    · fix SPIs for IOMMU with GICv5 (7a617d8)
    · fix the interrupt-map for the PCI DT node with GICv5 (2cd4773)
    · fully remove FVP_Foundation (dabe88c)
    · increase resident text size of BL2 (ef86015)
    · mark the ITS as dma-noncoherent (d36f109)
    · match boot manifest to PCI topology in RMM (02ca660)
    · remove erroneous comment from GICv5 DT (130fde9)
    · reword misleading error messages for GICv5+RME/SPD (d8b7717)
    · support FEAT_RNG_TRAP without FEAT_RNG (3422b84)
  * **Juno**
    · debug build fail with MEASURED_BOOT and SPMC_AT_EL3 (7eaa4fa)
    · increase BL2 maximum size to 0x15000 (07ca3c7)
    · increase BL2 maximum size to 0x16000 (36a961b)
    · raise BL2 max size for hardened IO checks (8335566)
    · restrict measured boot to a single algo (b5f6d09)
  * **Morello**
    · avoid capability tag fault on data access (3247828)
    · don't define get_mem_client_mode() when it won't be used (afe5d94)

  * **Neoverse-RD**
    - **RD-N2**
      - don't use V1 as a label (662eb59)
  * **TC**
    - add missing platform.h include (8a389ba)
    - configure mte addresses and sizes (cd76410)
    - correct register write in rng trap handler (cfbfe39)
    - don't use non-secure UART (d014579)
    - fix BL31 size for PLATFORM_TEST (853a2ad)
    - fix the platform topology (f6e46a6)
  * **Corstone-1000**
    - use generic hold pen (249c6f9)
  * **Automotive RD**
    - **RD-Aspen**
      - dts: make cache nodes DT spec compliant (75ecfa7)

- **Broadcom**
  * fix bad Mbed TLS check (f0022ae)
- **Intel**
  * allow kernel access to TSN TBU stream control registers (cc22653)
  * prevent invalid register rejection on non-A5F4 devices (d625940)
  * remove invalid SDM SMMU Stream ID register from bypass list (e928912)
  * restore overlap-safe memcpy_s to prevent SoCFPGA BL2 hang (e885959)
  * use const for socfpga_memcpy_s src (393ea86)
- **Marvell**
  * provide a default compile target (520d3f5)
  * work around uutils coreutils truncate -s %SIZE bug (9b6cb3c)
  * **Armada**
    - mv_ddr path may not be a git repo (47e2d0b)
    - **A8K**
      - add XFI params for NBX SFI 10G (72a5029)
      - declare amb_memory_map as an array (5cffee4)
      - mv_ddr path may not be a git repo (40929c2)

- **MediaTek**
  - disable ENABLE_LTO (f6ac5ad)
  - fix syntax error in mtk_vcorefs_bl_handler (16f73b9)
  - **MT8196**
    - check apusys write ce binary address (be276a8)
    - check apusys_ace_he_config address is valid (fbf9688)
    - increase apusys hardware semaphore timeout duration (6171865)
- **NVIDIA**
  - **Tegra**
    - fix receiving boot params on Tegra210 (b097e2a)
    - remove unused drivers (73ef9f9)
- **NXP**
  - move includes to where they are needed (b55caec)
  - **i.MX**
    - **i.MX 9**
      - set GICR frames (15a06e9)
      - **i.MX93**
        - add null terminator to mmap region array (3322cc1)
        - reduce BL31 limit to reserve DRAM timing area (c5c3378)
        - reduce the pmic stby off delay (2afd27f)
        - remove the unused platform helper code (ae943f1)
  - **Layerscape**
    - unlock write access SMMU_CBn_ACTLR (4040019)
  - **S32G274A**
    - **S32G274ARDB**
      - init CSR fields in DDR config (3e13402)
- **QEMU**
  - disable fpregs traps for QEMU in BL31 (39c01bc)
  - produce manifest entry inside transfer list (61f2534)
  - **SBSA**
    - fix build break when SPM set to SPMD (d2d85ed)
- **QTI**

    * do not map secure regions (d52d755)
    * don't fail to write dload mode register (8567679)
    * don't panic() without QTISECLIB (b7249aa)
    * harden mem_assign parameter handling (81671a9)
    * restrict EUD register r/w to debug builds only (fedc288)
    * **Lemans EVK**
      - move BL2 to pIMEM instead of system IMEM (23442a0)

- **Renesas**
  * **R-Car**
    - enable intialisation code for EL3 to NS-EL1 handoff (9397357)
    - **R-Car 3**
      - change the migrate information for OPTEE-OS (1394f1d)
      - disable stack protector for functions in SRAM (71b6bf7)
      - fix issue unable to access system RAM after warm boot (fb3a564)
      - fix order of DRAM DT nodes for 1x4 GiB bank (e226fa6)
      - prevent boot CPU hot unplug (460b5cf)
    - **R-Car 4**
      - change the migrate information for OPTEE-OS (178c5cf)
    - **R-Car 5**
      - enable missing FEAT_AMUv1p1 on R-Car Gen5 to fix the build (a00fee7)
      - enable SVE support for secure world (5b767b5)
      - fix linker script indent (4620c91)
      - fix multicore boot (ad5c9d0)
      - prevent boot CPU hot unplug (9a01785)
  * **RZ**
    - **RZ/A**
      - give more time to J-Link debugger (7f3481a)
      - staticize handle_debug_mode() (7fd671c)
- **Rockchip**
  * fix comment and value about TZRAM_SIZE (4ebc5a9)
  * **RK3576**
    - shorten names to fit into the allocated space (03e8e22)

- **ST**
  - avoid enormous tf-a-stm32mp13*stm32 (b66160f)
  - restore support for multiple entries in DTB_FILE_NAME (83dec00)
  - update stm32image tool to address Coverity issues (56a90ca)
  - use KEEP for .dtb_image and .bl2_image sections (9cbde74)
  - **STM32MP2**
    - correct ORR instruction in plat_crash_console_init (4fa2733)
    - remove unused macro PLAT_NB_GPIO_REGUS (1e9860a)
    - reorder DDR3 and DDR4 power supply sequences (d4346e5)
- **Texas Instruments**
  - fix invalid parameter in the function comment (af1fee4)
  - handle null parameter early in query_fw_caps (50d125b)
  - **K3 Low Power Devices**
    - add plat_get_image_source stub for BL1 (f121738)
    - extend BL1 RW region for debug builds (0b76c56)
- **Xilinx**
  - correct node idx for Versal Gen 2 (92f0f6a)
  - use bool in control expressions (b0d8470)
  - **Versal**
    - use 64-bit macros for BL31 checks (0837148)
  - **Versal NET**
    - use 64-bit macros for BL31 checks (45d5d0e)

- **Bootloader Images**
  - error out if image read size is too short (4ad5837)
  - **BL1**
    - clear loaded ID on pre-image load failure (f3f88df)
    - harden FWU copy/auth overflow checks (1d766b6)
  - **BL2**
    - disable FP traps early when crypto feature is enabled (4dd9e2b)
    - pie fixup size calculation (545e54d)
  - **BL31**
    - declare function prototype (4504b34)

    * don't enable D-cache inside a function call (5654972)

- **Services**
  - add suffix 'U' to unsigned integers (9c3faa0)
  - **RME**
    * do not force set ARM_ARCH_MAJOR/MINOR for ENABLE_RME (f85f130)
    * harden check in get PAS (af9303b)
    * increase worst-case event size (a1439c9)
    * **TRP**
      · add new RMMv2 ABI support (988eb1f)
    * **RMMD**
      · fix rmm_init() return convention and LFA activation failure (d31e090)
      · set RMM_V1_COMPAT=0 (b986631)
  - **SPM**
    * **EL3 SPMC**
      · avoid NULL deref in FFA_MEM_RETRIEVE_REQ path (850eba9)
      · fix v1.0 EMAD traversal (263561b)
      · prevent integer overflow in memory overlap validation (6a44494)
    * **SPMD**
      · add parentheses to clarify precedence (ae63fa0)
      · align parameter names in declarations (ea72d5f)
      · fix operand type mismatch (f21b831)
      · mark integer constants as unsigned (64a2189)
      · remove a racy assert in g0 int. handler (520db2b)
      · suppress unused function parameters (aea4b3d)
      · use appropriate types in expressions (21d81ac)
  - **DRTM**
    * map DLME and DRTM parameter region as execute-never (43cc99f)
    * validate NWd DCE region size to prevent overflow (4406b73)
  - **FIRME**
    * enable FIRME interface for CCA (f474a5a)
    * granule management service (776edfc)
  - **ERRATA ABI**

        * use boolean and braces (263c911)

    - **Live Firmware Activation**
        * fix verbose message for CPU hold count (6457630)
        * no need to decrement activation_count (d8abee8)
        * update activation_pending flag on SUCCESS (7bb963e)
        * validate component_id before activation check (3675bd8)

    - **Secure Payload Dispatcher**
        * **OP-TEE**
            · add boolean type for expressions (bf7901c)
            · add missing curly braces (6c61ed4)
            · add parenthesis for macro expressions (c9535e6)
            · align FDT buffer size with OP-TEE for in-place updates (d5aaf2c)
            · evaluate condition for boolean (a080ef5)
            · initialize the structure (ada9e22)
            · match variable type (827fb91)
            · move function to conditional block (18f8d11)
            · typecast operands to match data type (3b9016d)
        * **TSP**
            · don't forward declare tsp_vectors_t (cc1c867)

- **Libraries**
    - add header include guards (3daf1d1)
    - append ULL to unsigned constant (276bf69)
    - remove duplicate macro identifiers (7e68be5)
    - typecast operands to match data type (fed7bf5)
    - update utils_def.h to use assembly compatible integer literal suffixes (067c4bc)
    - **CPU Support**
        * correct comments for Cortex-A720 erratum 3711910 (597ca8a)
        * correct CVE-2024-7881 workaround and drop duplicate erratum (807d7bc)
        * enable Neoverse-V2 external LLC support (1df0bb5)
        * fix A78 workaround version for WORKAROUND_CVE_2024_5660 (ed45969)
        * fix C1 Pro powerdown abandon behavior (7783823)
        * fix feature detection for C1-Pro erratum 3686597 (98e89b1)

* fix the ordering of errata for C1 Premium (ed98a62)
* fix the ordering of errata for C1 Ultra (b8fd42a)
* fix version for Cortex-X1 CVE-2024-5660 (8a20c2d)
* fix version for Cortex-X2 CVE-2024-5660 (4273a6b)
* fix version for Cortex-X3 CVE-2024-5660 (b165cf8)
* make ERRATA_SME_POWER_DOWN work with the recommended state ID (93c7e70)
* only turn off CME when it is present (aded7f5)
* register ARCH_WORKAROUND_3 for Neoverse V2 (ecb7a36)
* remove C1-Premium erratum 3651221 (82ec67c)
* remove C1-Ultra erratum 3651221 (5b7afcb)
* reorder docs for Cortex-A76 erratum 1165522 (eaf2931)
* return ERRATA_MISSING when errata not found (284f5e7)
* sort erratum entries in A520 and X1 files (1bd44b6)
* unconstrain WORKAROUND_CVE_2025_0647 (716c864)
* update Cortex-A77 applied revision for CVE-2024-5660 (8726180)
* update Cortex-A78C WORKAROUND_CVE_2024_5660 (2012412)
* update revisions for Cortex-A76 erratum 1946160 (166c04f)
* use correct workaround for erratum N1 3324349 (54cbdcb)
* use hint instruction instead of the psb mnemonic (7905898)
* workaround for C1-Nano erratum 3392149 (cc2da10)
* workaround for C1-Nano erratum 3419531 (843c5cc)
* workaround for C1-Nano erratum 3437202 (f54c7d5)
* workaround for C1-Nano erratum 3516455 (9bce44d)
* workaround for C1-Nano erratum 3616450 (2e6594e)
* workaround for C1-Nano erratum 3630925 (c1e05df)
* workaround for C1-Nano erratum 3754876 (f077a58)
* workaround for C1-Premium erratum 3324333 (0cd6615)
* workaround for C1-Premium erratum 3502731 (37e3b5f)
* workaround for C1-Premium erratum 3651221 (e3fb210)
* workaround for C1-Premium erratum 3684152 (350a8a7)
* workaround for C1-Premium erratum 3705939 (68d095b)
* workaround for C1-Premium erratum 3815514 (20fe6fb)

* workaround for C1-Premium erratum 3865171 (f5bd742)
* workaround for C1-Premium erratum 3926381 (99b23d8)
* workaround for C1-Premium erratum 4102704 (ad01464)
* workaround for C1-Pro erratum 3300099 (740b3bb)
* workaround for C1-Pro erratum 3338470 (b7a3230)
* workaround for C1-Pro erratum 3362007 (9788d85)
* workaround for C1-Pro erratum 3619847 (89b6da0)
* workaround for C1-Pro erratum 3684268 (cddf0e7)
* workaround for C1-Pro erratum 3684268 (0d3eb4d)
* workaround for C1-Pro erratum 3686597 (429f4f6)
* workaround for C1-Pro erratum 3694158 (dd83309)
* workaround for C1-Pro erratum 3706576 (7b60fae)
* workaround for C1-Ultra erratum 3324333 (f3d5b70)
* workaround for C1-Ultra erratum 3502731 (81e845d)
* workaround for C1-Ultra erratum 3651221 (43f722d)
* workaround for C1-Ultra erratum 3658374 (3527194)
* workaround for C1-Ultra erratum 3684152 (9c72354)
* workaround for C1-Ultra erratum 3705939 (eacb047)
* workaround for C1-Ultra erratum 3815514 (8f8ee1e)
* workaround for C1-Ultra erratum 3865171 (e63111f)
* workaround for C1-Ultra erratum 3926381 (09d541b)
* workaround for C1-Ultra erratum 4102704 (f8f6f39)
* workaround for Cortex-A510 erratum 1910738 (84f6280)
* workaround for Cortex-A510 erratum 1937669 (0e37ec1)
* workaround for Cortex-A510 erratum 1942494 (f663b0e)
* workaround for Cortex-A510 erratum 1952872 (82e8616)
* workaround for Cortex-A510 erratum 1966377 (6e4321a)
* workaround for Cortex-A510 erratum 1975068 (6fb793f)
* workaround for Cortex-A510 erratum 1976290 (46bfe9b)
* workaround for Cortex-A510 erratum 2002389 (0d8d176)
* workaround for Cortex-A510 erratum 2027318 (6a022ca)
* workaround for Cortex-A510 erratum 2028010 (ccfde3f)

* workaround for Cortex-A520 erratum 2677201 (dcfdd3c)
* workaround for Cortex-A520 erratum 3631357 (056d5d1)
* workaround for Cortex-A520 erratum 3685825 (62844b5)
* workaround for Cortex-A57 erratum 817171 (5c53650)
* workaround for Cortex-A57 erratum 836019 (b6a3473)
* workaround for Cortex-A65 erratum 1179935 (015e1cd)
* workaround for Cortex-A65 erratum 1227419 (ede3a23)
* workaround for Cortex-A65 erratum 1541130 (8177e1e)
* workaround for Cortex-A65AE erratum 1638571 (7096d2b)
* workaround for Cortex-A710 erratum 1785648 (4de3978)
* workaround for Cortex-A710 erratum 1793423 (95ca058)
* workaround for Cortex-A710 erratum 1847092 (3720b1e)
* workaround for Cortex-A710 erratum 1887102 (58d1e97)
* workaround for Cortex-A710 erratum 3324338 (42c33bc)
* workaround for Cortex-A710 erratum 3888122 (2c9376c)
* workaround for Cortex-A710 erratum 4302969 (17130d2)
* workaround for Cortex-A715 erratum 2238661 (85674f7)
* workaround for Cortex-A715 erratum 2239006 (077f0ff)
* workaround for Cortex-A715 erratum 2275754 (5586a23)
* workaround for Cortex-A715 erratum 2284544 (1d8ac8b)
* workaround for Cortex-A715 erratum 2285473 (1e0da9b)
* workaround for Cortex-A715 erratum 2292761 (b246d9d)
* workaround for Cortex-A715 erratum 3456084 (af1f23a)
* workaround for Cortex-A720 erratum 3456091 (489bfa1)
* workaround for Cortex-A720AE erratum 3456103 (0a1f91a)
* workaround for Cortex-A725 erratum 2874943 (74d7575)
* workaround for Cortex-A725 erratum 2936490 (d9a21d3)
* workaround for Cortex-A725 erratum 3456106 (403ca6d)
* workaround for Cortex-A725 erratum 3711914 (ba7716b)
* workaround for Cortex-A76 erratum 1165347 (61f8953)
* workaround for Cortex-A76 erratum 1207823 (c2d99c3)
* workaround for Cortex-A76 erratum 2356586 (09f334d)

* workaround for Cortex-A76 erratum 3888013 (11b389b)
* workaround for Cortex-A76AE erratum 1931427 (46f364f)
* workaround for Cortex-A76AE erratum 1931435 (16de9fa)
* workaround for Cortex-A76AE erratum 1969401 (d428b42)
* workaround for Cortex-A76AE erratum 2371140 (f27e7f8)
* workaround for Cortex-A76AE erratum 2753838 (0e88b2c)
* workaround for Cortex-A76AE erratum 3888014 (79ddafd)
* workaround for Cortex-A77 erratum 1160841 (b8764fd)
* workaround for Cortex-A77 erratum 1204882 (98311b0)
* workaround for Cortex-A77 erratum 1220737 (ed3c064)
* workaround for Cortex-A77 erratum 1253791 (9b73520)
* workaround for Cortex-A77 erratum 1273521 (72dad2b)
* workaround for Cortex-A77 erratum 1515815 (874d48d)
* workaround for Cortex-A77 erratum 3888015 (760591c)
* workaround for Cortex-A78 erratum 1467580 (21eac8d)
* workaround for Cortex-A78 erratum 1479939 (d8b97cf)
* workaround for Cortex-A78 erratum 1492189 (2fb5979)
* workaround for Cortex-A78 erratum 1503072 (2100dec)
* workaround for Cortex-A78 erratum 1515634 (38d1be3)
* workaround for Cortex-A78 erratum 1827429 (86d299b)
* workaround for Cortex-A78 erratum 3888017 (2a9e337)
* workaround for Cortex-A78 erratum 4302972 (cdb62b6)
* workaround for Cortex-A78AE erratum 1827431 (9f2fdca)
* workaround for Cortex-A78AE erratum 1827433 (6d3be55)
* workaround for Cortex-A78AE erratum 2242639 (fa9841e)
* workaround for Cortex-A78AE erratum 2466780 (f7098b1)
* workaround for Cortex-A78AE erratum 2743229 (16ec568)
* workaround for Cortex-A78AE erratum 2779481 (183e1d7)
* workaround for Cortex-A78AE erratum 3888018 (d451275)
* workaround for Cortex-A78AE erratum 4302973 (1b89ab3)
* workaround for Cortex-A78C erratum 1941499 (8317d3e)
* workaround for Cortex-A78C erratum 1951501 (919ab77)

* workaround for Cortex-A78C erratum 2376746 (2711a0f)
* workaround for Cortex-A78C erratum 2395407 (40f28bb)
* workaround for Cortex-A78C erratum 2478780 (260e83c)
* workaround for Cortex-A78C erratum 2779483 (f11827e)
* workaround for Cortex-A78C erratum 3888019 (99a5f63)
* workaround for Cortex-A78C erratum 4302974 (be2afde)
* workaround for Cortex-X1 erratum 1467580 (8cc9b8c)
* workaround for Cortex-X1 erratum 1479939 (a16bb33)
* workaround for Cortex-X1 erratum 1492189 (e168af3)
* workaround for Cortex-X1 erratum 1503072 (95c7a1b)
* workaround for Cortex-X1 erratum 1515634 (f777172)
* workaround for Cortex-X1 erratum 1941498 (d553a48)
* workaround for Cortex-X1 erratum 1951500 (331f8fe)
* workaround for Cortex-X1 erratum 1952683 (025f274)
* workaround for Cortex-X1 erratum 2376745 (c8d0d08)
* workaround for Cortex-X1 erratum 2395406 (02bfb15)
* workaround for Cortex-X1 erratum 2742426 (1e8f40c)
* workaround for Cortex-X1 erratum 2779479 (681fec3)
* workaround for Cortex-X1 erratum 3888017 (ab76b73)
* workaround for Cortex-X1 erratum 4302972 (ffd62bf)
* workaround for Cortex-X2 erratum 1785648 (3863df6)
* workaround for Cortex-X2 erratum 1793423 (c30ff1e)
* workaround for Cortex-X2 erratum 1863568 (9764daa)
* workaround for Cortex-X2 erratum 1887102 (2c97de6)
* workaround for Cortex-X2 erratum 1887413 (991dc29)
* workaround for Cortex-X2 erratum 3324338 (df97485)
* workaround for Cortex-X2 erratum 3888122 (d0e2fb8)
* workaround for Cortex-X2 erratum 4302969 (5b77dd1)
* workaround for Cortex-X3 erratum 2138930 (157ba45)
* workaround for Cortex-X3 erratum 2147714 (045eb85)
* workaround for Cortex-X3 erratum 2184829 (530cf73)
* workaround for Cortex-X3 erratum 2214778 (231b0a6)

* workaround for Cortex-X3 erratum 3888125 (fcea95e)
* workaround for Cortex-X3 erratum 4302966 (1c0c1b5)
* workaround for Cortex-X4 erratum 2302507 (b478cb1)
* workaround for Cortex-X4 erratum 2620954 (bb3696e)
* workaround for Cortex-X4 erratum 2631888 (06c1aaf)
* workaround for Cortex-X4 erratum 2646977 (6f77394)
* workaround for Cortex-X4 erratum 3841338 (9dc81b2)
* workaround for Cortex-X925 erratum 2921199 (89725bc)
* workaround for Cortex-X925 erratum 2922378 (7c00052)
* workaround for Cortex-X925 erratum 2933290 (030b26e)
* workaround for Cortex-X925 erratum 3324334 (3232d74)
* workaround for Cortex-X925 erratum 3692980 (5d0d6e4)
* workaround for Cortex-X925 erratum 3730893 (ea24488)
* workaround for Cortex-X925 erratum 3865185 (dca40b8)
* workaround for Neoverse N1 erratum 1791580 (9df1ed7)
* workaround for Neoverse N1 erratum 3888013 (f6b03b6)
* workaround for Neoverse N1 erratum 925373 (224fe69)
* workaround for Neoverse V1 erratum 1542436 (ade85b8)
* workaround for Neoverse V1 erratum 1618634 (ec4db9e)
* workaround for Neoverse V1 erratum 1618636 (2f59dac)
* workaround for Neoverse V1 erratum 1619807 (691334a)
* workaround for Neoverse V1 erratum 1654562 (a122603)
* workaround for Neoverse V1 erratum 1674403 (9e62bd1)
* workaround for Neoverse V1 erratum 3888016 (204aebf)
* workaround for Neoverse V3 erratum 3696307 (8b1de68)
* workaround for Neoverse V3 erratum 3734562 (3d01b70)
* workaround for Neoverse V3 erratum 3782181 (742be38)
* workaround for Neoverse V3 erratum 3864536 (323f9ee)
* workaround for Neoverse V3 erratum 3878291 (281548c)
* workaround for Neoverse-N1 erratum 3324349 (8fc57d3)
* workaround for Neoverse-N2 erratum 2138953 (925661a)
* workaround for Neoverse-N2 erratum 3324339 (a6b7ed5)

    * workaround for Neoverse-N2 erratum 3888123 (35f0012)
    * workaround for Neoverse-N2 erratum 4302970 (420a059)
    * workaround for Neoverse-N3 erratum 3456111 (930a464)
    * workaround for Neoverse-V2 erratum 3442699 (a0723de)
    * workaround for Neoverse-V2 erratum 3888126 (155e87f)
    * workaround for Neoverse-V2 erratum 4302968 (af82ff2)
    * workaround for Neoverse-V3 erratum 3312417 (f4f1db3)

  - **Debug FS**
    * allocate enough space to fit all names (58a648e)
  - **EL3 Runtime**
    * actually check for the EA bit on exception entry (cf14b88)
    * do not assert the stack's location (0e96552)
    * handle SVE/SMC traps instead of panicing (59cc811)
    * remove lower_el_panic() (574db8e)
    * resolve essential-type mismatch (6de7520)
    * **Context Management**
      · do not check for AArch32 support to enable features (886f95d)
      · don't context switch GICv3 registers on NS<->RL transitions (68eacbb)
      · don't context switch GICv3 registers on NS<->RL transitions (c84cf19)
      · initialise TCR2_EL2 and HAFGRTR_EL2 for compatibility (fb345d8)
      · remove set_aapcs_args functions (252b2ff)
  - **PSCI**
    * add parentheses to clarify precedence (b1359b0)
    * adding missing curly braces (0dce7ad)
    * check PSCI_SET_SUSPEND_MODE's arguments (ee11b40)
    * do not pass unused parameters (03d0ef1)
    * fix operand type inconsistency (8a8d8e0)
    * gate suspend end_pwrlvl override in OS_INIT mode at runtime (5e4cc6d)
    * get the cpu_ops before exiting coherency (0ee188d)
    * have generic code call psci_pwrdown_cpu_start() on system off/reset (a21d2ed)
    * make sure CMOs on `struct psci_cpu_data` do not affect other data (ad6e3f8)
    * pass correct arguments to psci_suspend_to_pwrdown_start() (3ea7588)

      * set requested local power states in failure path (5e7b6c7)
  - **ROMlib**
      * gate RSA jump table entries (9b4abb0)
  - **Granule Protection Tables**
      * remove unused `gpt_disable` function (764e2bd)
  - **Translation Tables**
      * exclude security state attributes (53f44c4)
  - **C Standard Library**
      * fix coverity deadcode issue (c2316a9)
      * use const void * for memcpy_s source pointer (ded96ec)
  - **Locks**
      * restore spin_trylock's ability to acquire a lock (9c4a03f)
  - **Context Management**
      * actually clear MDCR_EL3 bits (e63e579)
- **Drivers**
  - **Authentication**
      * make key measurement failure fatal (6e221b4)
      * validate decrypt parameters and IV/tag lengths (9504113)
  - **Console**
      * adding missing curly braces (d2b32ea)
  - **I/O**
      * add NULL check for spec io_open FIP (3405337)
      * validate FIP ToC bounds and catch short reads (38110f9)
  - **MMC**
      * align sd_switch_func_status struct (f7889b3)
  - **MTD**
      * **NAND**
          · add read parameter page retry (bc308f8)
  - **Arm**
      * **GIC**
          · init the GIC before the platform with a hook (9523049)
          · **GICv5**

          · align IWB_WDOMAINR to the EAC spec (a6d2996)
        · **GICv3**
          · add an isb between the ICC_SRE_EL2 and ICC_SRE_EL1 writes (d88390a)
          · allow GIC-700 ESPI ranges in CHIPR setup (0a7ecee)
    * **MHU**
        · relax 4-byte align in buffer size (3c2ce65)
    * **RSE**
        · remove host ROTPK support and test (bd14181)
- **NXP**
    * **DDR**
        · fix improper disable of first DDRC (3c7ec07)
- **Renesas**
    * **R-Car3**
        · load B-copy SA6 certificate from correct offset (e255f2b)
- **ST**
    * **Clock**
        · check fdt_subnode_offset return (009ea37)
        · do not include platform_def.h (295f132)
        · do not return NULL in _clk_get (db91ca2)
        · increase startup timeout for oscillators for STM32MP1 (5eb683f)
        · increase startup timeout for oscillators for STM32MP2 (a239c63)
        · remove CLK_CFG/CMD_CLK support for st,kerclk (e311675)
        · remove some variables default assignment (74d6c46)
    * **DDR**
        · add missing newline (2166f9f)
        · change retention disable management (057f4f6)
        · fix coverity static analysis issues (f7d29e1)
        · improve error management during PHY init on STM32MP1 (524e694)
        · manage CID filtering when accessing PWR_CR11 (74887d7)
        · manage semaphore with CID filtering (d578883)
    * **Resource Isolation Framework**
        · do not assign an unused value for node when parsing config (2da8621)

- **ST PMIC**
  - check stpmic2_is_buck1_high_voltage return (b28feb6)
  - initialize some structs in dt_pmic2_i2c_config (51b7907)

- **Miscellaneous**
  - **Debug**
    - add debug log build option (d065020)
  - **FDTs**
    - **ST**
      - **STM32MP2**
        - fix LPDDR4 16bits swizzle (fd296ed)
  - **Security**
    - add CVE-2024-7881 mitigation to C1-Premium CPU (83ad6ba)
    - add CVE-2024-7881 mitigation to C1-Pro CPU (c09454d)
    - add CVE-2024-7881 mitigation to C1-Ultra CPU (c130f92)
    - add CVE-2025-0647 for C1-Premium (4245399)
    - add CVE-2025-0647 for C1-Ultra (2bf674e)
    - add CVE-2025-0647 for Cortex-A710 (a52dcae)
    - add CVE-2025-0647 for Cortex-X2 (9c17b3e)
    - add CVE-2025-0647 for Cortex-X3 (7fe900e)
    - add CVE-2025-0647 for Cortex-X4 (680a74b)
    - add CVE-2025-0647 for Cortex-X925 (f26fb93)
    - add CVE-2025-0647 for Neoverse-N2 (a142b10)
    - add CVE-2025-0647 for Neoverse-V2 (145603e)
    - add CVE-2025-0647 for Neoverse-V3 (efdd8ce)
    - add workaround for CVE-2025-0647 (416b861)
    - avoid CVE_2025_0647 for bl2 build (ec4f10c)
    - update Cortex-X3 fix version for CVE-2024-7881 (38db5f4)
    - update Cortex-X4 fix version for CVE-2024-7881 (13cd56d)
    - update Cortex-X925 fix version for CVE-2024-7881 (80e56ad)
    - update Neoverse-V2 fix version for CVE-2024-7881 (ef22181)
    - update Neoverse-V3/V3AE fix version for CVE-2024-7881 (a7f6d2c)
    - workaround for C1-Pro/CME CVE-2026-0995 (e9b6d2a)

- **Documentation**
  - do not describe BL arguments in as much detail (3919457)
  - fixing the name for C1 premium errata 3324333 (8d6d9c4)
  - update PSCI instrumentation unit label (80947b6)
  - update sme disconnect on power down (e3ace29)
  - use publicly accessible links for CVE-2017-15031 Advisory (9cf95e3)
  - use publicly accessible links for CVE-2022-47630 Advisory (7311283)
  - use publicly accessible links for CVE-2023-49100 Advisory (d5ca2fd)
- **Build System**
  - assign the ldflags-common variable before appending to it (c46f2d9)
  - fix BL2_CPPFLAGS when ENABLE_RME is set (6c3cfbd)
  - pass -flto-partition to the linker and not the compiler (ee0f273)
  - set defaults to feature flags before platform.mk (70a6541)
  - use ARM_ARCH_FEATURE instead of -march directly (fb0c409)
  - use assignment instead of memcpy to avoid a GCC 11 bug (1e969d0)
  - use the correct value of host-poetry (8cefbe0)
- **Dependencies**
  - **Compiler runtime libraries**
    - update compiler-rt to v22.1.5 (8fe5d8e)
  - **libfdt**
    - adding missing curly braces (66fa430)
    - fix coverity reported issue (4fc7026)
    - fix misra 14.4 and 15.6 violations (f822b3d)
    - resolve misra 10.3 violations (4b1d8a4)
    - typecast operands to match data type (11befc7)
  - **zlib**
    - fix overflow issue from coverity (e03704b)
    - update zlib to v1.3.2 (21c3672)

14.2 2.14.0 (2025-11-20)

14.2.1 ? BREAKING CHANGES

- **Services**
  - **SPM**
    - **EL3 SPMC**
      - no longer accepts invalid fragment offsets

        See: validate fragment offset (c55b519)

      - Zeroes or forwards a different set of registers, depending on the FF-A version of the source and destination. E.g. a call from a v1.1 caller to a v1.2 destination will zero out the extended registers, which is different from the old behavior of forwarding everything to EL2 SPMC, but only x0-x7 to the EL3 SPMC.

        See: fixed x8-x17 register handling for FFA 1.2 (bb9fc8c)
- **Libraries**
  - **Firmware Handoff**
    - Projects/scripts relying on the in-tree TLC tool will need to use the new, externalized location or workflow.

      See: remove in-tree TLC implementation (3ff7523)

    - LibTL is now included in TF-A as a submodule. Please run `git submodule update --init --recursive` if you encounter issues after migrating to the latest version of TF-A.

      See: switch to LibTL submodule (b5d0740)
- **Drivers**
  - **Measured Boot**
    - LibEventLog is now included in TF-A as a submodule. Please run `git submodule update --init --recursive` if you encounter issues after migrating to the latest version of TF-A.

      See: move to ext event log lib (b67e984)
- **Tools**
  - **Memory Mapping Tool**
    - The image memory map visualization tool now uses commands, rather than arguments, to determine the behaviour of the script. See the commit message for further details.

      See: change behavioural flags to commands (8daebef)

14.2.2 New Features

- **Architecture**
  - **CPU feature like FEAT_XXXX / ID register handling in general**
    - add ID register defines and read helpers (8515b43)
    - add support for FEAT_FGWTE3 (4274b52)
    - add support for FEAT_IDTE3 (f396aec)
    - add support for FEAT_RME_GPC2 (09a4bcb)
    - do feature detection on secondary cores too (d335bbb)
    - enable FEAT_AIE support (cc2523b)
    - enable FEAT_CPA2 for EL3 (a1032be)
    - enable FEAT_MPAM_PE_BW_CTRL support (c42aefd)
    - enable FEAT_PFAR support (b3bcfd1)
    - fix build with ENABLE_FEAT_PAUTH_LR=1 (3325415)
    - include enabled security state scope (d508bab)
    - introduce FEAT_RME_GDI support (5e827bf)
  - **DynamIQ Shared Unit (DSU)**
    - enable PMU registers access at EL1 (1f866fc)
    - support power control and autonomous powerdown config (d52ff2b)
- **Platforms**
  - **AMD**
    - **Versal Gen 2**
      - add hooks for mmap and early setup (4efae6a)
      - add SPMD support for SPMC at S-EL1 (c7ddb0f)
      - add support of MMI_GEM as wakeup source (4589ce0)
      - parse reserve memory subnodes (59eaed0)
      - remove handoff entry from tl (9029408)
      - validate non-secure entry addr (27e7222)
  - **Arm**
    - add pseudo CRB area (d771d57)
    - handle RMM ep_info during LFA (4ea0ebc)
    - implement arm platform GPT logging (fe3299d)

- introduce ARM_SPM_NS_MEM_BASE and move NS buffer (78a6c8f)
- load config after GPT FIP offset (6757473)
- load tos_fw_cfg using xferlist in SPMC_AT_EL3 (00c353c)
- unify Linux kernel as BL33 handling (2f5fd82)
- **Common**
  - add support for kernel DT handoff convention (291e493)
- **FVP**
  - add a GICv5 device tree (d358eb2)
  - add extra DRAM configuration for TZC (887cdf4)
  - add firmware update agent uuid in StandaloneMm (28d325c)
  - add FVP_HW_CONFIG_ADDR make variable (93fc69d)
  - add GICv5 support (e2e90fa)
  - add pseudo CRB area (235d975)
  - add SoC name support to FVP (cd802c2)
  - add StandaloneMm manifest for rust-spmc (35721cb)
  - add stub implementation for plat_lfa_notify_activate() (31e9fd9)
  - add the GICv5 config (9a099b5)
  - bump the SPMC version (4d29a8f)
  - change PLAT_SP_IMAGE_NS_BUF_BASE (b19b693)
  - enable kernel dt convention (8946bb0)
  - give bootargs on all configs (73c587e)
  - implement LFA get components API (3bb5266)
  - implement platform API for LFA activation pending check (357079c)
  - implement platform API for LFA cancel operation (b4cbf50)
  - implement platform API for load and auth image (a7fbccc)
  - increase bl2 mmap tables for handoff (25688b8)
  - increase PLAT_ARM_SPMC_SIZE for rust-spmc (1cc0294)
  - increase secure partition's table mapping count (b1f527a)
  - increase xtable for pseudo CRB for SPM_MM (8569456)
  - increase xtable for pseudo CRB for SPMC_AT_EL3 (3d35b10)
  - initialize LFA component activators in platform layer (b100e91)
  - introduce fvp_stmm_bl2_sp_list.dts for StandaloneMm (98ae901)

        - move PLAT_ARM_SPMC_SP_MANIFEST_SIZE definition (3c90095)
        - organize fvp_stmm_manifest.dts (8d66892)
        - tos_fw_config with transfer list (bc3014a)
        - update evtlog info in the xferlist's DT_SPMC_MANIFEST entry (10f6ccd)

    - **Juno**
        - add pseudo CRB area (66579ca)
        - boot with TRANSFER_LIST (fad8844)
        - change PLAT_SP_IMAGE_NS_BUF_BASE (b53b69c)
        - change preprocessor condition for plat_get_mbedtls_heap() (1776a1e)
        - change the FW_NS_HANDOFF_BASE (eee8963)
        - increase MAX_XLAT_TABLES for SPMC_AT_EL3 (c76e828)
        - increase xtable for pseudo CRB (7d142cb)
        - organize juno_stmm_manifest.dts (8706efc)
        - support image measured boot (1c199c5)
        - support StandaloneMm (e9a457f)

    - **Neoverse-RD**
        - change PLAT_SP_IMAGE_NS_BUF_BASE (c4d39b7)
        - **RD-V3**
            - enable numa aware per-cpu for RD-V3-Cfg2 (8e85be4)

    - **TC**
        - add support for configuring DSU settings (fd4e602)
        - bump SPMC version to FF-A v1.3 TC platform (838eb53)
        - change PLAT_SP_IMAGE_NS_BUF_BASE (22e97b7)

    - **Automotive RD**
        - **RD-Aspen**
            - add BL31 for RD-Aspen platform (c2cd362)
            - add DSU support (d69c3b1)
            - add DSU to the device tree (b0a8c52)
            - add support for configurable platform's CPU topology (c16a3b7)
            - enable tbb on rd-aspen platform (287e24f)
            - introduce Arm RD-Aspen platform (d1a1abe)
            - scmi gracefully shutdown system (ba4814b)

        - support BL32 (OP-TEE) (33a10dc)
        - support configurable CPU topology in device tree (b666f0a)

- **Intel**
    - migrate RSU client to SiPSVC V3 (b85b49e)
    - move common functions to common lib files (6fcd047)
    - support IO96B ECC Error Injection via SMC call (bdcd41d)
    - update CPUECTLR_EL1 to boost ethernet performance (bb9e34f)

- **MediaTek**
    - add APIs exposed to the static library (97881aa)
    - **MT8189**
        - add dfd driver (aed177c)
        - add EC pin control in SPM (ed11c2f)
        - add hotplug driver (b8d63a7)
        - add IOMMU enable control in SiP service (4c449fc)
        - add LPM v2 support (d92ee8e)
        - add mcdi driver (5f00709)
        - add mt8189 mtcmos platform data (d8c718c)
        - add mtcmos config (7e864c7)
        - add pmic driver (868b2d6)
        - add PWRAP driver (9c9324c)
        - add reset and poweroff function for PSCI call (148152f)
        - add SPM basic features support (65db67b)
        - add SPM common driver support (5f748b3)
        - add SPM features support (083cfad)
        - add SPM low power mode (3289678)
        - add SPMI driver (257aa94)
        - add support display driver (6997076)
        - add support for PTP3 (3f446df)
        - add thermal driver support (3da2d29)
        - add UFS driver support (897c2d0)
        - add UFS functions used by libbl31.a (260e18b)
        - add VCORE DVFS drivers (5e91cfd)

- enable DCM to support LP mode (f018e05)
- enable PMIC low power setting (d701cf8)
- fix the typo in the display driver (0f462e6)
- link prebuilt library (4dfb819)

* **MT8196**
  - add CPU QoS stub implementation (4488b22)
  - add MMinfra support (c33b98d)
  - add SLBC SiP handler (e86fb81)
  - add SMMU SID stub implementation (5be0e22)
  - add UFS functions used by the static library (31a69d9)
  - enable MTE2 (e2ad194)
  - extract common SPM code for reuse (532ac05)

- **NXP**
  * **i.MX**
    - **i.MX 9**
      - add full system power control option (0df6ba3)
      - add imx9 common code base (4ddfb6f)
      - add necessary ele api call support (1168465)
      - **i.MX94**
        - add initial support for imx94 (4249a4f)
      - **i.MX95**
        - add initial support for i.MX95 (d70b09f)
        - add optee support (7bde9a4)
        - implement a semaphore for GIC quiescing (f7e7ea1)
        - support trusty os (b182f70)
  * **S32G274A**
    - move fip in a dedicated partition (88b8aa9)
    - **S32G274ARDB**
      - add custom DDR FW UUID entry (4d9ac8f)
      - add DDR clock source support (c0cbf5a)
      - add mc_rgm_release_periph func (f2ab124)
      - init the generic timer for BL2 (e790512)

            · initialize the IO buffer (dbf400d)
            · initialize the uSDHC driver (d82c211)
            · set the system counter rate (3c60749)

- **QEMU**
  * add support for FEAT_GCS (d9872d7)
  * add support for FEAT_SxPxE (139ddfb)
  * add support for FEAT_TCR2 and FEAT_SCTLR2 (436cc70)
  * skip paged image info (803560d)
  * **SBSA**
    · support s-el2 and s-el1 spmc (cda0487)
- **QTI**
  * add BL32 support (48897ba)
  * add TF-A BL2 common platform framework (c48d0ae)
  * introduce basic XPU driver (292ffc0)
  * **Kodiak**
    · add support for RB3Gen2 platform (ac44b9c)
- **Renesas**
  * **R-Car**
    · add support for Renesas R-Car S4 / V4H / V4M (b45b5ba)
    · deduplicate plat_crash_print_regs (06f8eb5)
    · deduplicate PWRC SRAM trampoline (57e22e0)
    · deduplicate PWRC timer (92196d4)
    · deduplicate SCIF console_rcar_register (9979a20)
    · deduplicate stack protector (223d989)
    · replace static rcar_putc pointer with static register offsets (b21216f)
    · split common SCIF code (3c6170b)
- **Rockchip**
  * **RK3568**
    · support SCMI for clock/reset domain (4e1ccc6)
- **ST**
  * adapt .stm32 file creation for clang (fa4acc2)
  * adapt stm32 linker scripts for clang (43560d8)

* add RNG minor version (864466b)
* introduce SoC family compilation switch (701178d)
* put stm32image tool in build directory (0b4722c)
* remove unsupported option for clang (44bf952)
* update stm32 linker scripts (6778835)
* update stm32cubeprogrammer API (ef39709)
* use and override default MBedTLS config (1dd3b65)
* **STM32MP1**
  - add stm32_get_uid_otp (08252f9)
  - enable MCE driver for STM32MP13 (c7a457a)
  - prepare DDR secure area encryption for STM32MP13 (d06b375)
* **STM32MP2**
  - add HASH and RNG compilation (27b4244)
  - add RIF dt-binding defines (7b4b208)
  - add RIFSC/RISAB protection for USB3DR (ecad2c9)
  - add some platform helpers (2c831e4)
  - add STM32MP_USB_PROGRAMMER support (6d1366e)
  - add support for minimal FIP with only DDR FW (eb43024)
  - generate FIP for DDR initialization (aa63c23)
  - prepare DDR secure area encryption (f2b9807)
  - **STM32MP21**
    - add clock and reset bindings (dcb00b1)
    - add PWR registers file (2ec3cec)
    - add RCC registers file (58cf812)
  - **STM32MP25**
    - add RCC register to display all IWDG flags (e957c33)

– **Texas Instruments**
* add mmu regions for am62l soc (8853eba)
* add support for boot notification msg (7d3c700)
* am62lx init: boot notif and version msg (987c9b0)
* build generic timer (a5cf0ba)
* de-assert AINACTS at boot (879fdd0)

        - enable power state validation for K3 SoCs (7f5ff95)
    - **Xilinx**
        - deprecate PM_REQ_SUSPEND EEMI API (93434bd)
        - use common SECURE/NON_SECURE macro (4fd510e)
        - **Versal**
            - add hooks for mmap and early setup (72b9f52)
            - add support to clear PM specific data (1d4372c)
        - **ZynqMP**
            - mark IPI calls secure/non-secure (8ce93ec)
- **Bootloader Images**
    - **BL31**
        - add placeholder activator implementation for LFA (f647f56)
- **Services**
    - **FF-A**
        - bump SPMD FF-A version (dd87b73)
    - **RME**
        - **RMMD**
            - add per-CPU activation token (89d979c)
            - add RMM_RESERVE_MEMORY SMC handler (745c129)
            - add warm reset helpers for primary and secondary CPUs (22bbb59)
        - **RMM**
            - add placeholder activator callbacks for LFA (d837ccd)
    - **SPM**
        - update MM_SP_MEMORY_ATTRIBUTES_GET_AARCH64 interface (127bd5b)
        - **EL3 SPMC**
            - allow physical partitions to have multiple UUIDs (0322d7a)
            - deliver TPM event log via hob list (4e5247c)
            - enable FFA_MEM_RETRIEVE_MEM_REQ from the hypervisor (7d34c9b)
            - get sp_manifest via xferlist (aae2370)
            - parse and report VM availability messages (ce299f9)
            - update FF-A version checks (3f1c63d)
            - update FFA_MEM_PERM_GET interface (9bfe78c)

- **SPMD**
  - add FFA_NS_RES_INFO_GET ABI (08f9ba5)
  - get spmc manifest from xferlist (55fd56d)
  - support for FFA_ABORT invocation from SWd (b3dcd50)
- **DRTM**
  - remove todo that is implemented (9232d3f)
- **Live Firmware Activation**
  - add activation handler interface for component activation (9e03218)
  - add helper to check LFA prime completion status (ce78f3c)
  - add initial implementation for LFA_ACTIVATE (07de22d)
  - add initial implementation for LFA_PRIME (67fa182)
  - add LFA holding pen logic (ff7daec)
  - add platform hook for activation notification (5084b7f)
  - allow LFA_PRIME from one CPU at a time (c8e0821)
  - create LFA SMC handler template (cf48f49)
  - enable LFA of RMM (f024c57)
  - implement LFA_CANCEL SMC (3f7b286)
  - implement LFA_GET_INFO SMC and integrate LFA build (b9dee50)
  - implement LFA_GET_INVENTORY SMC (06a6f29)
  - introduce support for call_again for LFA_PRIME (59b826c)
- **Secure Payload Dispatcher**
  - **OP-TEE**
    - check paged image size (c9e91b2)

- **Libraries**
  - add mmio read with timeout (7a1970f)
  - **CPU Support**
    - add pabandon support to Nevis (6588ce0)
    - add pabandon support to the Alto cpu (2e764df)
    - add support for caddo cpu (656500f)
    - add support for Canyon CPU (5fc2895)
    - add support for Dionysus cpu library (2cdc34c)
    - add support for Neoverse-N2 prefetcher (7538438)

    * add support for venom cpu (d4c50e7)
    * add support for veymont cpu (51247cc)
    * enable Maximum Power Mitigation Mechanism (037c7a8)
    * fix external LLC presence bit in Neoverse N3 (ff90ce4)
    * update cpu_check_csv2 check (2b43216)
    * implement workaround for DSU-120 erratum 2900952 (efc945f)

- **EL3 Runtime**
    * add flags argument to handle_sysreg_trap (88655be)
- **Per-cpu**
    * add documentation for per-cpu framework (b92586a)
    * integrate per-cpu framework into BL31/BL32 (98859b9)
    * introduce framework accessors/definers (962958d)
    * introduce linker changes for NUMA aware per-cpu framework (7256cf0)
    * migrate amu_ctx to per-cpu framework (0f90f37)
    * migrate psci_cpu_pd_nodes to per-cpu framework (9f407e4)
    * migrate psci_ns_context to per-cpu framework (6d2d846)
    * migrate rmm_context to per-cpu framework (f708e9d)
    * migrate spm_core_context to per-cpu framework (f5dca2a)
- **PSCI**
    * add API to get number of CPUs currently in ON state (a7be2a5)
    * check that CPUs handled a pabandon (461b62b)
    * make pabandon support generic (04c39e4)
    * optimise clock init on a pabandon (fd914fc)
- **SMCCC**
    * add SoC name support to SMCCC_ARCH_SOC_ID (cb4ee3e)
    * clear scr_el3.nse in smccc_arch_feature_availability (847c111)
    * enable support for FEAT_MEC (a357d15)
    * introduce TPM Start SMC (96546b5)
    * introduce a new vendor_el3 service for ACS SMC handler (f69f551)
- **C Standard Library**
    * add __2snprintf alias for armclang-specific name mangling (ab0255a)
- **Locks**

    * add non-blocking spinlock_try() API (86822f2)

  - **Firmware Handoff**
    * add firmware handoff threat model (a5e9623)
    * support libtl submodule builds (06f3c70)

- **Drivers**
  - **Authentication**
    * **mbedTLS**
      · update mbedtls to version 3.6.4 (24d6ed9)
      · update mbedtls to version 3.6.5 (5affb6a)
  - **FWU**
    * add platform hook for shared NV ctr (75b5be9)
    * create flag for BL2 separation (5ce4ee1)
    * documentation for BL2 separation (cb0a4e9)
    * separate bl2 image from rest of the FIP (d57362b)
  - **MMC**
    * add define for no response (46a1167)
  - **GUID Partition Tables Support**
    * platform hook to log corrupted GPT (3537dad)
  - **SCMI**
    * add base protocol agent API (6445c83)
    * add i.MX9 SCMI vendor CPU protocol (dbfe9df)
    * add support for discovering and changing parent clocks (684952d)
    * support graceful system power set (eb113bc)
    * update version to 3.0 (e8a96bf)
  - **Arm**
    * **GIC**
      · change gic_cpuif_enable/disable to weak (34a94fa)
      · make IRQ groups optional (5f65250)
      · **GICv5**
        · add a barebones GICv5 driver (13b6281)
        · add support for building with gicv5 (8cef63d)
        · assign interrupt sources to appropriate security states (82b228b)

            · initialise the IRS (dfb37a2)
            · initialise the IWB (7179920)
            · probe components (4db6bf9)
        · **GICv3**
            · add GIC-720AE model id (0d65d5a)

  - **NXP**
    * **FLEXSPI**
        · add 128Mbytes flash info (7b370c1)
    * **uSDHC**
        · add data buffer (b61379f)
        · add dynamic mapping (cdf002d)
        · add timeout to blocking loops (01d24d6)
        · flush and invalidate buffers (7e2a434)
  - **ST**
    * add Memory Cipher Engine driver (6d79740)
    * add RIFSC driver (8934c7b)
    * add RISAF driver (399cfdd)
    * **Clock**
        · add STM32MP21 and STM32MP23 RCC variants (088238a)
        · rename RCC_USBTCCFGR register into RCC_UCPDCFGR (d3e47fb)
    * **Crypto**
        · add multi instance and error management in RNG driver (02b770a)
    * **USB**
        · add USB DWC3 driver (867cd15)

- **Miscellaneous**
  - **DT Bindings**
    * add MCE DT bindings for STM32MP13 (5f90891)
  - **FDTs**
    * **ST**
        · **STM32MP1**
            · add MCE support for STM32MP13 DK board (0bea409)
        · **STM32MP2**

      - add firewall nodes in fw-config (86d91be)
      - add memory firewall node (bb3c45d)
      - **STM32MP25**
        - add RNG node (c434b76)
        - enable rng nodes for ST boards (9598477)
    - add RISAF nodes for STM32MP25 (8f783a5)

- **Documentation**
  - platform hook for whether NV ctr is shared (95ed23a)
  - platform hook to log GPT corruption (2605cde)
  - update context management's threat model (e77cd73)
  - add Architectural Feature Support reference (5548ab9b3)
- **Build System**
  - add helpers for managing compiler launchers (5c24052)
  - add shell fragment joining utility (`shell-join`) (a75ab9a)
  - add shell fragment length utility (`shell-words`) (8165d82)
  - add shell fragment slicing utility (`shell-slice`) (9a782d4)
  - add shell program reference utility (`shell-program`) (a72154c)
  - add shell quoting utility (`shell-quote`) (10cb835)
  - add shell word extraction utility (`shell-word`) (5980fa7)
  - add shell word map utility (`shell-map`) (37cd618)
  - add temporary variable binding utility (`with`) (6eb35c6)
  - add utilities for modifying includes (0fcee05)
  - allow full LTO builds with clang (ddc918b)
  - create define macro to be used by BL (116d2c0)
  - pass cflags to the linker when LTO is enabled (885ed9e)
  - put crttool in the build directory (f4595e6)
  - put enctool in the build directory (96fbe88)
  - put fiptool in the build directory (cbd6cec)
  - set CRYPTO_SUPPORT macro per BL (6e2fe62)
  - setting CRYPTO_LIB via CRYPTO_SUPPORT (cd92fb2)
  - update `shell-map` to also pass indices (1d5ae1e)
  - use clang as a linker (6c2e5bf)

  - use LDLIBS instead of LDFLAGS for library search paths (d833129)
- **Tools**
  - **Memory Mapping Tool**
    - add summary command to memory map script (a98d466)

14.2.3 Resolved Issues

- **Architecture**
  - **CPU feature like FEAT_XXXX / ID register handling in general**
    - add ras files to the build from a common location (25fe31b)
    - configure CPTR_EL2.ZEN and CPTR_EL2.TZ to match Linux (7f471c5)
    - do feature detection before feature enablement (0f57a38)
    - don't overwrite PAuth keys with an erroneous cache clean (072e8ae)
    - extend FEAT_EBEP handling to delegate PMU control to EL2 (714a1a9)
    - remove unused FEAT_NV feature functions (3c0ebab)
    - update FEAT_PAUTH's feat detect line to tri-state (e293fca)
    - use of additional breakpoints (482fbf8)
  - **DynamIQ Shared Unit (DSU)**
    - dsu config for all cores in hot reset (98863b1)
    - fix illegal address Coverity finding (70516ab)
- **Platforms**
  - add essential bool type (d83e1f0)
  - declare unused parameters as void (ee14e1a)
  - declare unused parameters as void (877279d)
  - fix misra violation 10.1 (5d09adb)
  - typedef operands to match data type (0523d3d)
  - replace stray BL2_AT_EL3 with RESET_TO_BL2 (f856626)
  - **Allwinner**
    - avoid crash when running without DTB (69b4a59)
    - fix variable may be used uninitialized error (ce750f1)
  - **AMD**
    - **Versal Gen 2**
      - align comment about invalid console selection (bf51768)

· align IOU_SCNTR base address macro name with other platforms (18a77ba)

· enable graceful system shutdown (08ae97c)

· fix offsets for apu pcil (02210f6)

· handle debugfs specific APIs before EEMI handler (633cf6b)

· initialize counter-timer frequency register (f08dcf5)

· move plat_core_pos_by_mpidr to asm (d2244f3)

· remove extraneous parentheses (bb2b022)

· remove PM_ABORT_SUSPEND API implementation (7739450)

· runtime console in debug mode (b3555f1)

· use common function to get system counter frequency (f2ae203)

· use PM_STATE_CPU_OFF for core power down in SELF_SUSPEND (9cfc723)

· validate Node ID in PM callback events (244f9fb)

– **Arm**

* derive RMM bank size from payload (dbda614)
* don't override the gic redistributor frames (1d59d68)
* increase reserved DRAM1 mem for NS images (7a171ad)
* move lfa componet header to common and fix the helper (716deb8)
* re-enable console by default in BL31 (de46405)
* typecast operands to match data type (8a13157)
* typecast operands to match data type (3d7caf4)
* **Common**

  · add missing curly braces (618e37c)

  · add missing curly braces (3ba36ea)

  · error out if image load size is zero (854b4dd)

  · initialize the variables (929a290)

  · rename exit label (1db51f0)

  · typecast operands to match data type (f3ecd83)

* **CSS**

  · don't require the GICC frame to be defined on GICv3 (0204bc8)

* **FVP**

  · avoid stack usage in check_cpupwrctrl_el1_is_available() (36ceead)

  · do not unregister the console on system suspend (f185a54)

- fix coverity issue unsigned_compare (4824e25)
- give fvp_ras.c better dependencies (7e87f49)
- initialise the event log's size to avoid using gibberish values (b199ca1)
- pass an `interrupt-names` property for the timer when using GICv5 (8c37540)
- skip SP discovery through FFA_PARTITION_INFO_GET_REGS (0fbcef0)
- use global option for setting PLAT_ARM_MAX_BL2_SIZE (27bff0b)
- workaround when PCIe 2 region is not present in DTB (2cbea16)

- **Morello**
  - fix the incorrect order of gpu interrupts in dt (45a567a)
- **Neoverse-RD**
  - add console initialisation to BL31 (47fca89)
- **TC**
  - fix c1_pro power down abandon (2e1dff2)
  - force specifying TARGET_PLATFORM (81fece2)
- **Automotive RD**
  - **RD-Aspen**
    - enable CPU feature runtime checking (3a324c2)
    - fix timer bus cells & fix ranges (6fb6bee)

– **Intel**
  - add 5us delay before Linux reconfig to avoid HNOC hang (afae10f)
  - add memory alignment at cadence SD/eMMC driver's descriptor (2fcb37d)
  - add missing cache flush operation for hmac (1363096)
  - configure SCR1 for 32/16 non-secure SMRs and context banks (a042bb3)
  - configure usb3 system manager reg in TFA (00c1b8c)
  - correct condition to process unused entries (ef80b7a)
  - fix eMMC driver issues in boot flow on agilex5 (38636fe)
  - fix iossm driver timeout in agilex5 (5b173df)
  - fix SDMMC driver when sdmclk running at 200MHz (5482237)
  - fix socfpga_psci for cpu on off function (8f7575e)
  - fix variable may be used uninitialized error (cfde117)
  - flush the mailbox response buffer in SiPSVC V3 (cb3ceb5)
  - iossm v2 enhancement refactor (f1b1fae)

  * match declaration with definition (c90c0be)
  * remove wfi polling when performing cpu on (53791e8)
  * select the DFI interface based on the hand-off data (6993598)
  * set BIT2 of system manager MPFE Interface Select (130e88a)
  * solve agilex warm reset issue (2c03c2c)
  * solve s10 warm reset issue (7e94cc1)
  * support DDR In-line and Out-of-Band ECC handling (92d2277)
  * support generic mailbox command in SiPSVC V3 (cbb62e0)
  * support SMC 64bit return args in SiPSVC V3 (cdab401)
  * undo setting USB 3.1 reset pulse bit in BL2 (e655389)
  * unexpected DDR reset type value observed on Agilex5 (8bdfbaf)
  * update FCS AES method for GCM block modes (1e1dbad)
  * update generic mailbox command filter method (0934946)
  * update initialization to prevent warnings message (da1e000)
  * update nand driver to enable Linux OS boot (6f7f8b1)
  * update TFA to patch for Linux 6.12 rebase warning message (6e6efe8)
  * update the AES GCM/GCM_GHASH modes return data size (8e47685)
  * update the RSU version logic read (d7286ad)
  * verify data size in AES GCM and GCM-GHASH modes (34f092a)

- **MediaTek**
  * fix mtcmos build failure (68514bd)
  * **MT8188**
    - fix variable may be used uninitialized error (3599f40)
  * **MT8189**
    - fix mt_spm_pmic coverity (7c3bcb6)
    - remove unused SPM definitions and files (9fc5866)
  * **MT8195**
    - fix variable may be used uninitialized error (8914774)
  * **MT8196**
    - duplicated ETIMEDOUT definition (0e039ea)

- **NXP**
  * driver crypto caam (cacde83)

      * tbbr: adds snvs_init (ce9b87e)
    * **i.MX**
      * add static attribute for platform specific gic struct (c7294df)
      * check if the cluster id is valid (cbe42e1)
      * match function parameters to declaration (82987cb)
      * **i.MX 8M**
        * add RDC entries for all missing UARTs (a2c6e11)
        * don't reconfigure default region0 (9bf1480)
        * **i.MX 8M Plus**
          * assign wdog1 to domain0 only (8845f8b)
          * remove ERR050463 VPUMIX workaround (b1ed68d)
      * **i.MX 8ULP**
        * fix variable may be used uninitialized error (e551dbd)
    * **Layerscape**
      * **LX2**
        * **LX216**
          * **LX2160**
            * add DDRC missing DIMMs (25b410b)
            * set snoop-delayed exclusive handling on A72 cores (5acf82b)
        * put cert_create_tbbr.mk in the standard location (7c3e9a0)
  - **QEMU**
    * add reason parameter to MEC update (9c6e060)
    * fix variable may be used uninitialized error (db0d535)
  - **QTI**
    * allow secure r/w to the EUD enable register (bdd1932)
    * fix build without coreboot (e76c1b9)
    * fix config PLAT_XLAT_TABLES_DYNAMIC (7f86b63)
    * uart platform base address for SC7180 (5a12275)
  - **Renesas**
    * **R-Car**
      * **R-Car 3**
        * add missing image_base/size assignment to BL33 image loading path (e6b05fc)

      - clear TCR_EL1 at the BL2 entry point (fe87637)

    - **R-Car 4**

      - assure SCIF and HSCIF clock are always enabled (13e1e76)
      - drop unused plat_pm_scmi (3c57f96)

- **ST**

  - add build directory for stm32image (0d35f9e)
  - allow several call of stm32cubeprog_uart_load (ed88979)
  - mark INCBIN-generated sections as SHF_ALLOC (454441e)
  - replace down counter by a timeout upon dfu detach (f79ca8d)
  - **STM32MP2**

    - correct typo in definition header (f53f260)

- **Texas Instruments**

  - remove validate_power_state definition (80cfd5a)
  - specify allowable rcv_addr in mailbox (64e58ce)

- **Xilinx**

  - add API to get powerdown request status (c0719d2)
  - add macro to extract node class from node ID (2f8856f)
  - fix missing security flag in suspend path (5cac1d8)
  - incorrect usage of SECURE_FLAG for psci (3e3cdf2)
  - limit pm_feature_check deprecation warning to once per boot (eaceb37)
  - match function type as its declared (1d94b27)
  - remove PM_ABORT_SUSPEND API implementation (ecee071)
  - resolve misra rule 10.1 violations (c314a0b)
  - resolve misra rule 12.2 violations (f78c597)
  - resolve misra rule 16.3 violations (a335cd9)
  - resolve misra rule 16.4 violations (ea3ec86)
  - resolve misra rule 2.5 violations (93db9e6)
  - resolve misra rule 4.5 violations (2993166)
  - resolve misra rule 4.6 violations (6df7184)
  - resolve misra rule 8.13 violations (cd60ab7)
  - resolve violations generated with IPI_CRC_CHECK enabled (7d0eb0e)
  - typecast operands to match data type (87904ba)

    * validate Node ID in PM callback events (a6dd46a)
    * **DCC (Debug Communication Channel)**
        · typecast operands to match data type (abfcd67)
    * **Versal**
        · add external declaration (33a264c)
        · match function declaration with its definition (5b51d4d)
        · match function type as its declared (4e89096)
        · modify IPI4 and IPI5 trigger bit definitions (c96f838)
        · remove client-side implementation of PM_ABORT_SUSPEND (86d9b35)
        · replace ull with ULL to fix misra violation (f709265)
        · runtime console in debug mode (0701792)
        · switch case has only one confirming clause (325949f)
        · typecast operands to match data type (5d8831c)
        · typecast operands to match data type (c1b0a52)
        · update integer const with suffix U (1cbf6c4)
    * **Versal NET**
        · add fallback on handoff failure (05d0cb4)
        · fix coverity violation prevent buffer overrun (3ef5820)
        · move plat_core_pos_by_mpidr to asm (9127041)
        · remove client-side code of PM_ABORT_SUSPEND (eeef4ac)
        · runtime console in debug mode (18283e6)
    * **ZynqMP**
        · remove PM_ABORT_SUSPEND API implementation (c069c8e)
        · runtime console in debug mode (2546350)
        · workaround issue around cntpct_el0 (97ec082)

- **Bootloader Images**
  - **BL1**
    * use per-world context correctly (a873d26)
  - **BL2**
    * unify the BL2 EL3 and RME entrypoints (04cf04c)
  - **BL31**
    * add missing curly braces (ec93223)

    * declare function as static (d9712f9)
    * remove incorrect asserts (3e1d33d)
    * typecast operands to match data type (2fa4dee)
  - **BL32**
    * **TSP**
      · add missing include (13c9952)
      · keep the tsp D128 unaware, not the dispatcher (b77c6aa)
- **Services**
  - declare unused parameters as void (8cee7b2)
  - typecast operands to match data type (d16ad81)
  - **RME**
    * enable support for FEAT_D128 on Realm world (a3effe0)
    * fix incorrect shift operation in rmmd (c08285c)
    * **TRP**
      · report argument errors before bailing out (ab5db33)
    * **RMMD**
      · avoid race conditions in CPU finish (5782406)
      · correct activation condition check (5ba2ad3)
    * **RMM**
      · enable SMCCC_ARCH_FEATURE_AVAILABILITY for RMM (cbba59c)
  - **SPM**
    * change the SMMUv3TestEngine being used (015c76d)
    * **EL3 SPMC**
      · do not check NS bit for fragments (153eb4c)
      · fixed x8-x17 register handling for FFA 1.2 (bb9fc8c)
      · set NS bit by default and clear it as needed (cb2e574)
      · update event log related include (982e702)
      · validate fragment offset (c55b519)
    * **SPMD**
      · skip spurious id while group0 interrupt handling (5b10f25)
  - **DRTM**
    * add event name data for separator event entries (94aa3d2)

    * correct PROTECT_MEM_REGION value as per DRTM specification (65f6936)
    * measure and record the DLME EP offset instead of absolute address (f3fec61)
    * remove plat_system_reset() (30bbc4f)
    * set correct max region mask for DMA protection to 0xFFFF (3b6e594)
  - **TRNG**
    * avoid undefined behaviour in pack_entropy (f2db4eb)
  - **Live Firmware Activation**
    * check error code from plat_lfa_load_auth_image (10f81c8)
    * include string.h in lfa_main.c (c0d32ee)
  - **Secure Payload Dispatcher**
    * **OP-TEE**
      · allow opteed_handle_smc_load to handle 64-bit data_pa (0c1ca2e)
- **Libraries**
  - align round_up with MISRA 10.1 and 10.8 (fa8b749)
  - declare unused parameters as void (ccec2b9)
  - modify function to have single return (fecf325)
  - use 64-bit constants in MDCR_EL2 bit macros (a9e3195)
  - **CPU Support**
    * check minor revision before applying runtime errata (645917a)
    * correct alto library (4348393)
    * inform the compiler that struct cpu_ops is aligned (759ed94)
    * organize Cortex-X2 errata entries (f753b4a)
    * put back the global declaration for erratum #3701747 (28a0b5a)
    * use correct Makefile indentation for CVE-2018-3639 check (360460a)
    * workaround for Cortex-A510 erratum 2008766 (d64d421)
    * workaround for Cortex-A510 erratum 2169012 (124ff99)
    * workaround for Cortex-A510 erratum 2218134 (4592f4e)
    * workaround for Cortex-A510 erratum 2420992 (4fb7090)
    * workaround for Cortex-A510 erratum 3672349 (af1fa79)
    * workaround for Cortex-A510 erratum 3704847 (ea88493)
    * workaround for Cortex-A710 erratum 1901946 (4467348)
    * workaround for Cortex-A710 erratum 1916945 (df067c0)

    * workaround for Cortex-A710 erratum 1917258 (d91c417)
    * workaround for Cortex-A710 erratum 1927200 (cb2702c)
    * workaround for Cortex-A715 erratum 2376701 (4fca3ee)
    * workaround for Cortex-A715 erratum 2409570 (d6e941e)
    * workaround for Cortex-A715 erratum 3711916 (5c5b9e3)
    * workaround for Cortex-A720 erratum 2729604 (217a79c)
    * workaround for Cortex-A720 erratum 3711910 (87e69a8)
    * workaround for Cortex-X2 erratum 1901946 (ce64ea6)
    * workaround for Cortex-X2 erratum 1916945 (ff879c5)
    * workaround for Cortex-X2 erratum 1917258 (ccee7fa)
    * workaround for Cortex-X2 erratum 1927200 (e236548)
    * workaround for Cortex-X2 erratum 1934260 (2c0467a)
    * workaround for Cortex-X2 erratum 2136059 (a8e4d5a)
    * workaround for Cortex-X2 erratum 2267065 (41b9697)
    * workaround for Cortex-X2 erratum 2291219 (989c798)
    * workaround for Cortex-X3 erratum 3213672 (42920aa)
    * workaround for Cortex-X3 erratum 3692984 (f828efe)
    * workaround for Cortex-X3 erratum 3827463 (6a464ee)
    * workaround for Cortex-X4 erratum 3133195 (58148b9)
    * workaround for Cortex-X4 erratum 3887999 (5a45f0f)
    * workaround for Neoverse-V2 erratum 3701771 (98ea732)
    * workaround for Neoverse-V2 erratum 3841324 (7d94765)
    * keep leading zeros in CVE ID prints (d1ed0c3)

  - **Debug FS**

    * guard against negative ch offsets (c56a85d)
    * return negative value on error (b1f5309)
    * set debugfs smc start to vendor EL3 (4db17f4)

  - **EL3 Runtime**

    * allow RNDR access at EL3 even when RNG_TRAP is enabled (45218c6)
    * typecast operands to match data type (f05b489)
    * remove unused cpu_data related macros (2c730ee)
    * **Context Management**

- correctly restore BL2's context (780c9f0)
- deprecate use of NS_TIMER_SWITCH (ccf6796)
- disable SPE/TRBE correctly (985b6a6)
- do not restore spsr and elr twice on external aborts (f9f4944)
- do not save SCR_EL3 on external aborts (a202529)
- remove unused macro (c81b9cb)
- use safe casting in memcpy (ea5a4e9)

* **SIMD**

  - enforce FP regs context mgmt when SVE regs are enabled (da81d45)

- **Per-cpu**
  * remove redundant casts (f7ccf12)
- **PSCI**
  * add missing curly braces (bac32cc)
  * add missing curly braces (f3d9e22)
  * do not modify higher levels for standby in OSI mode (b34be5d)
  * fix coverity issue with out-of-bounds read (654ab9e)
  * initialise variable to default zero (a9eb44d)
  * initialise variable to default zero (e8c3fdd)
  * initialize the variables (df51e33)
  * typecast operands to match data type (f6166f7)
- **Granule Protection Tables**
  * fix fill_l1_cont_desc() function (9bc1e59)
- **SMCCC**
  * cleanup unused declaration (8397768)
  * don't panic on a feature availability call with FEAT_RNG_TRAP (f610c8c)
  * fixed define when ENABLE_FEAT_FPMR is disabled (57b23ea)
  * ignore SCR_EEL2_BIT (ee75a71)
  * modify feat discovery to use aarch64 value (a0fa44b)
  * resolve caller world confusion (4e4a8c5)
- **Translation Tables**
  * add missing curly braces (98a2af6)
  * change MT_DEVICE to map to nGnRnE (00669dc)

    * typecast expressions to match data type (b6c1cdf)

  - **C Standard Library**
    * add missing curly braces (60e5aee)
    * add the memcpy declaration back (f23d26d)
    * fix coverity overflowed constant (4d7238b)
    * fix coverity overflowed constant (02dbb14)
    * replace true-false with explicit comparisons (28ac195)
  - **Locks**
    * mark spin_trylock as a public function (d7e9372)
  - **DICE Protection Environment**
    * prevent compiler warnings (0c3b84c)
  - **Context Management**
    * correct guards for PIE and POE registers (f77d713)

- **Drivers**
  - **Console**
    * add missing curly braces (efea4ae)
    * create unique variable name (6d6aa1d)
    * ensured proper bitwise shift operation (1ec2c39)
    * match function parameter is decleration (acad3b0)
    * typecast operands to match data type (51df71c)
    * typecast operands to match data type (5bbe4fd)
  - **FWU**
    * fwu NV ctr upgraded on trial run (526d956)
  - **SCMI**
    * change error code output for wrong ids (48ec8d3)
    * fix coverity issue INTEGER_OVERFLOW (6ec871d)
    * fix coverity issue INTEGER_OVERFLOW (a443fbd)
  - **TPM**
    * extend timeout before reading data (6053ca4)
  - **Arm**
    * add missing curly braces (21b7d86)
    * declare unused parameters as void (d17b69d)

        * typecast operands to match data type (0cd8e55)
        * **GIC**
            - fix coverity issue INTEGER_OVERFLOW (b6f3b4f)
            - **GICv3**
                - add missing curly braces (fcaa188)
                - avoid incrementing global gicr_frames pointer (6262a3e)
                - fix misra violation 12.1 (4780c6f)
                - match function definition and declaration (d9df6b4)
                - remove plat_gicv3_base.c (07e18c2)
                - typecast operands to match data type (77189b0)
                - typecast operands to match data type (eaa454a)
        * **MHU**
            - shift by minor revision offset (9fad664)
    - **MediaTek**
        * fix incorrect source assignment (f6d7b01)
    - **NXP**
        * **NXP Crypto**
            - restricts generating nxp_mkvb via ns-world (76a9560)
        * **DDR**
            - add missing macro define for Errata A009663 (f1f04d7)
        * **uSDHC**
            - correct the usage of BIT and GENMASK (26fd068)
            - fix clk_rate and bus_width type (3d16507)
            - fix the clock rate calculation (2e90f3e)
            - handle response for CMD0 (92a7b54)
            - remove unnecessary delay (6347429)
            - wait SDSTB before changing the clock (583a544)
    - **ST**
        * remove useless field in fixed regul (adeee68)
        * **BSEC**
            - remove useless defines in BSEC3 (b43afb7)
            - rename OTPSR field (6fede18)

    * **Clock**
        - add ck_bus_risaf4 clock for STM32MP2 (2a20f3e)
        - force ARM_DIVSEL for flexgen63 config at 400MHz (40d0ceb)
        - prevent panic when external oscillator is absent (6fb522d)
    * **Crypto**
        - do not set IPRST if BUSY flag is present (6bc7c5b)
        - improve RNG health test configuration (382dff5)
        - resolve MISRA warning in HASH (84ebe2a)
    * **DDR**
        - bad refresh update level toggle sequence (6851fd9)
        - remove TODO in STM32MP2 driver (fd5e5e7)
    * **UART**
        - aarch32: remove unnecessary timeout waiting in putc (5bebf8f)
        - aarch32: wait for UART ISR register TC bit for console flush (35988c6)
        - aarch64: unwait for UART ISR register TC bit in putc function (8ad5ea0)
        - aarch64: wait for UART ISR register TC bit for console flush (65a96c0)
    * **USB**
        - correct phy_epnum type for error trace (8defd6f)
        - init endpoint with fixed value if only one is used (fce3675)
        - stub dead code (e5bafa2)
    * **Watchdog**
        - remove num_irq (9adc427)

- **Miscellaneous**
  - **AArch64**
    * do not print EL1 registers on EL3 panic (8f15231)
  - **AArch32**
    * make get_cpu_ops_ptr() PCS compliant (71f7a36)
  - **DT Bindings**
    * bad FLEXGEN configuration of pred-division for STM32MP25 (a8d4cc7)
  - **FDTs**
    * remove extra members in PCI interrupt-map (cd170ec)
    * **ST**

        - **STM32MP2**
          - set SDMMC max frequency to 166 MHz on stm32mp25 (a6665c0)
          - **STM32MP25**
            - new swizzle configuration for STM32MP257F-EV1 board (f42f2e7)
  - **Security**
    - add clrbhb support (d6affea)
    - fix Cortex-A715 CVE-2022-23960 (ad0e848)
    - fix Cortex-X3 CVE-2022-23960 (07df6c1)
    - fix Neoverse V2 CVE-2022-23960 (3084363)
    - fix spectre bhb loop count for Cortex-A720 (9fd05e6)
    - remove CVE_2022_23960 Cortex-A720 (e22ccf0)
    - remove CVE_2022_23960 Cortex-X4 (a055fdd)
    - remove CVE_2022_23960 Neoverse V3 (a2e22ac)
  - **SDEI**
    - fix coverity finding array index read (6d11328)
    - resolve coverity error for SDEI GIC (9dde2cc)
- **Documentation**
  - ensure all submodules are cloned (efb21dd)
  - fix some broken links (854d199)
  - obey the build directory (725a80d)
  - **Changelog**
    - fix platform order and add smcc to deprecated (07dc387)
- **Build System**
  - add DTB dependency for certificate generation (f02135a)
  - add include directory dependencies (19f4e19)
  - align the cpu-ops flags with all others (2897374)
  - correctly detect that an option is missing with ld_option (b45fc16)
  - correctly forward C compiler to CMake (fcada3d)
  - don't rely on Event Log build tree (76d5d32)
  - fix compiler wrapper detection (3dc69bc)
  - fix incorrect expansions in `shell-map` (46a898f)
  - fix incorrect parentheses expansion in `shell-map` (cfc2d76)

    - fix lib cflags use in build macros (d8a23ec)
    - fix Makefile syntax in constraints helpers (d42144a)
    - force CMake to cross-compile (3dc41a6)
    - forward compiler launcher to CMake (f162e3a)
    - link Event Log library directly (03032a9)
    - prevent races on the build directory (774fb37)
    - put the -target definitions in toolchain.mk (aa6edba)
    - remove redundant variables (500927e)
    - remove SUPPORT_STACK_MEMTAG (6bf7c6a)
    - scan symbols until all are resolved (bba54a3)
    - set ERRATA_SPECULATIVE_AT after platform.mk (c2dc512)
    - set linker to lld before evaluating ld_option (5ecae95)
    - shell-escape toolchain wrapper commands (78cb57c)
    - simplify the -target options (b926017)

- **Tools**
    - renesas: rzg: Fix tool build (fa0df1b)
    - **Firmware Image Package Tool**
        * skip Layerscape makefile for S32 build (ba3668f)
    - **Secure Partition Tool**
        * handle load-address-relative-offset property (5f7116a)
    - **Memory Mapping Tool**
        * ensure terminal width is respected (8ab677b)
- **Dependencies**
    - **libfdt**
        * add suffix 'U' to unsigned integers (a1321ac)

14.3 2.13.0 (2025-05-14)

The threat model for context management support is not available in the release.

14.3.1 ⚠ BREAKING CHANGES

- **Platforms**
  - **Arm**
    - rename PLAT_MHU_VERSION flag

      See: rename PLAT_MHU_VERSION flag (a773955)
- **Bootloader Images**
  - **BL32**
    - **TSP**
      - The prototype for `tsp_early_platform_setup` has been redefined. Platforms must update their implementations to match the new function signature.

        See: cascade boot arguments to platforms (32d9e8e)
- **Drivers**
  - **Arm**
    - **RSE**
      - remove rse_comms_init

        See: remove rse_comms_init (91c7a95)

14.3.2 New Features

- **Architecture**
  - **CPU feature like FEAT_XXXX / ID register handling in general**
    - add support for FEAT_PAUTH_LR (025b1b8)
    - add support for PMUv3p9 (ba9e6a3)
    - enable FEAT_BTI to FEAT_STATE_CHECKED (10ecd58)
    - enable FEAT_PAuth to FEAT_STATE_CHECKED (8d9f5f2)
    - disable FPMR trap (a57e18e)
    - enable FEAT_MOPS in EL3 when INIT_UNUSED_NS_EL2=1 (6b8df7b)
    - setup per world MDCR_EL3 (c95aa2e)
    - add support for FEAT_SPE_FDS (4fd9814)
- **Platforms**
  - **AMLogic**
    - **GXL**
      - add support for booting from U-Boot SPL/with standard params (8dca65d)

- **Arm**
  - add a macro for SPMC manifest base address (eab1ed5)
  - add initrd props to dtb at build time (1c08ff3)
  - add support for Transfer List creation (4c5ccbf)
  - convert arm platforms to expect a wakeup (da305ec)
  - enable Linux boot from fip as BL33 (eb8cb95)
  - migrate heap info to fw handoff (ada4e59)
  - port event log to firmware handoff (b30d904)
  - support AArch32 booting with handoff (abdb953)
  - support boot info handoff and event log (a852fa1)
  - use provided algs for (swd/p)rotpk (da57b6e)
  - use the provided hash alg to hash rotpk (d51981e)
  - **FVP**
    - add StandaloneMm manifest in fvp (8416e79)
    - add stub function to retrieve DLME image auth features (1733deb)
    - add stub platform function to get ACPI table region size (5d37755)
    - allocate L0 GPT at the top of SRAM (7a4a070)
    - build hob library (8740771)
    - change size of PCIe memory region 2 (2e55a3d)
    - define single Root region (665a8fd)
    - give bootargs on all configs (a507f4f)
    - increase BL1 RW for PSA Crypto (51bdb70)
    - increase bl2 mmap len for handoff (24f7830)
    - increase cactus-tertiary size (dcd8d7f)
    - increase GPT PPS to 1TB (aeec55c)
    - port event log to firmware handoff (5bf0b80)
    - set defaults for build commandline (bf9a25f)
    - support AArch32 booting with handoff (2ab298b)
  - **Neoverse-RD**
    - deprecate and remove RD-N1-Edge platform variants (71ad967)
    - deprecate and remove RD-V1 platform variants (afb3075)
    - deprecate and remove SGI-575 platform (a023201)

      - use larger stack size when S-EL2 spmc is enabled (82f4659)
      - **RD-V3**
        - add carveout for BL32 image (6823f5f)
        - add dts files to enable hafnium as BL32 (4d9b828)
        - define SPMC manifest base address (12973bc)
        - enable the support to fetch dynamic config (37cc7fa)
        - introduce platform handler for Group0 interrupt (4593b93)
    * **TC**
      - add 'kaslr-seed' node in device tree for TC3 (2d967e9)
      - add devicetree node for AP/RSE MHU (06fa4c4)
      - add dsu pmu node for TC4 (50ad0cf)
      - add MCN PMU nodes in dts for TC4 (624deb0)
      - add SLC MSC nodes to TC4 DT (99f6790)
      - allow Android load and Boot From RAM (932e64a)
      - configure UART for TC4 FPGA (84ca47a)
      - define MCN related macros for TC4 (8f61c20)
      - enable Arm SPE for TC4 (cea55c8)
      - enable DSU PMU el1 access for TC4 (00397b3)
      - enable MCN non-secure access to pmu counters on TC4 (d1062c4)
      - enable stack protector (d1de6b2)
      - enable trng (2ae197a)
      - fpga: Enable support for loading FIP image to DRAM (969b759)
      - get entropy with PSA Crypto API (8f0235f)
      - increase SCP BL2 size to support optimization 0 (3755e82)
      - initialize MHU channels with RSE (0328f34)
      - initialize the RSE communication in earlier phase (a3f9617)
      - port BL1-BL2 interface to firmware handoff framework (93c50ae)
      - port BL2-BL31 interface to firmware handoff framework (2a36dee)
      - port BL31-BL33 interface to firmware handoff framework (25a6bcd)
      - print ni-tower discovery tree (d87a856)
      - update CPU PMU nodes for tc4 (1ce2c74)
    * **Automotive RD**

- **RD-1 AE**
  - add Generic Timer in device tree (6e1bf7e)
  - add support for OP-TEE SPMC (8b27eb7)

- **Intel**
  - add FDT support for Altera products (29d1e29)
  - add support for query SDM config error and status (fcf906c)
  - implementation of SiPSVC-V3 protocol framework (204d5e6)
  - provide atf build version via smc call (d1c58d8)
  - support FCS commands with SiPSVC V3 framework (597fff5)
- **Marvell**
  - add trng driver (6d5fad8)
- **MediaTek**
  - add gic driver (d905b3d)
  - add UFS stub implementation (57c7351)
  - add vcp driver support (a1763ae)
  - update mtk_sip_def.h (ead2602)
  - **MT8189**
    - add GIC driver on MT8189 (1d193f9)
    - add GPIO support (cecbb93)
    - add reset and poweroff function for PSCI call (4e40a1f)
    - disable L3C shared SRAM if the bootloader is coreboot (7794e7c)
    - enable cirq for Mediatek MT8189 (83a5a0d)
    - initialize platform for MT8189 (6c60901)
  - **MT8196**
    - add APU kernel control operations (5e5c57d)
    - add APU power on/off functions (3ee4b2d)
    - add APU power-on init flow (0781f78)
    - add APU RCX DevAPC setting (f31932b)
    - add APU SMMU hardware semaphore operations (2d134d2)
    - add APUMMU setting (e534d4f)
    - add APUSYS AO DevAPC setting (31a0b87)
    - add CPC module for power management (75530ee)

- add DCM driver (e578702)
- add GPIO support (4cb9f2a)
- add LPM v2 support (da8cc41)
- add mcdi driver (5cb0bc0)
- add mcusys moudles for power management (95e974f)
- add Mediatek EMI stub implementation for mt8196 (39f5e27)
- add Mediatek MMinfra stub implementation (4794746)
- add mtcmos driver (1f913a6)
- add PMIC driver (d4e6f98)
- add pwr_ctrl module for CPU power management (4ba679d)
- add reset and poweroff function for PSCI call (22d74da)
- add SMMU driver for PM (86dd08d)
- add smpu protection for APU secure memory (7ed4d67)
- add SPM basic features support (fb57af7)
- add SPM common driver support (a24b53e)
- add SPM common version support (5532feb)
- add SPM features support (01ce1d5)
- add SPMI driver (adf73ae)
- add topology module for power management (da54c72)
- add vcore dvfs drivers (f0dce79)
- disable debug flag in APU driver (31137e1)
- enable appropriate errata (0d11e62)
- enable APU on mt8196 (f5a6aa0)
- enable APU spmi operation (823a57e)
- enable apusys mailbox mpu protection (83f836c)
- enable apusys security control (9059a37)
- enable cirq for MediaTek MT8196 (49d8c11)
- enable DP and eDP for mt8196 (3e43d1d)
- enable IRQ configuration (16f94b9)
- enable PMIC low power setting (e8e8768)
- enable vcore dvfsrc feature (a3c218a)
- fix MT8196 gpio driver (6f891e6)

  - initialize platform for MediaTek MT8196 (a65fadf)
  - link prebuilt library (e033943)
  - refactor LPM header include paths to use lpm_v2 (6fac00a)
  - show ERROR log if need (1ba50c3)
  - turn on APU smpu protection (5de1ace)

- **NXP**
  - **S32G274A**
    - add console mapping (a1e07b3)
    - dynamically map GIC regions (5680f81)
    - dynamically map SIUL2 and fip img (507ce7e)
    - enable MMU for BL2 stage (eb4d418)
    - enable MMU for BL31 stage (e2ae6ce)
    - enable SDHC clock (47b3a82)
    - increase the number of MMU regions (0089258)
    - map each image before its loading (34fb2b3)
    - split early clock initialization (61b5ef2)
- **QEMU**
  - add hob support for qemu platforms (648d2d8)
  - add plat_rmmd_mecid_key_update() (9c9a31e)
  - hand off TPM event log via TL (cc58f08)
  - update for renamed struct memory_bank (991f536)
  - **SBSA**
    - add support for RME on SBSA machine (acb0937)
    - adjust DT memory start address when supporting RME (99bc6cf)
    - configure GPT based on system RAM (d079d65)
    - configure RMM manifest based on system RAM (fb4edc3)
    - dissociate QEMU NS start address and NS_DRAM0_BASE (26da60e)
    - increase maximum FIP size (122dbc2)
    - relocate DT after the RMM when RME is enabled (17af959)
- **QTI**
  - platform support for qcs615 (f60617d)
- **Raspberry Pi**

    * **Raspberry Pi 3**
        - add dTPM backed measured boot (4f9894d)
        - implement eventlog handoff to BL33 (6dfcf4e)
        - implement mboot for rpi3 (c4c9e2b)
- **Rockchip**
    * increase FDT Buffer for Rockchip Devices (ab99dce)
    * update uart baudrate for rk3399 (6d7f1d4)
    * **RK3576**
        - support rk3576 (036935a)
- **Socionext**
    * **Synquacer**
        - add support Hob creation (357f28d)
- **ST**
    * use dedicated version of DT for SP_MIN (71ba164)
    * **STM32MP1**
        - rework SVC services (39b08bc)
    * **STM32MP2**
        - add a runtime service for STGEN configuration (7f41506)
        - add common SMC runtime services (f55b136)
        - add FWU support (c28c0ca)
        - disable PIE by default on STM32MP2 platform (ac9abe7)
        - use USART1 for debug console on ultra-fly boards (d59dd96)
- **Texas Instruments**
    * add support for TI mailbox driver (9347ff4)
    * introduce basic support for the AM62L (21b14fd)
    * introduce PSCI Driver for AM62L (1abdc20)
- **Xilinx**
    * **Versal NET**
        - add SDEI support (10510c9)
    * **ZynqMP**
        - add pin group for lower qspi interface (fe81d9c)
- **AMD**

    - **Versal Gen 2**
        - add bufferless IPI Support (af22b19)
        - add dependency macro for PM (aec66c3)
        - add support for platform management (414cf08)
        - extended SMCCC payload for EEMI (0cc5e21)
        - is OCM configured as coherent (c3ab09d)
        - retrieve DT address from transfer list (ea45387)
        - update platform version to versal2 (4003ac0)

- **Bootloader Images**
  - adding psa crypto - crypto_mod_finish() (055c97a)
  - **BL32**
    - enable r3 usage for boot args (3fabca7)
    - **TSP**
      - cascade boot arguments to platforms (32d9e8e)

- **Services**
  - **FF-A**
    - add FFA_MEM_PERM_GET/SET_SMC64 (ddf72e6)
    - support FFA_MSG_SEND_DIRECT_REQ2/RESP2 (09a580b)

```
The FFA_MSG_SEND_DIRECT_REQ2/RESP2 early implementation in the␣
↪EL3 SPMC is limited
to the use of 8 input/output registers per the SMCCC which isn't␣
↪compliant to the
FF-A v1.2 specification mandating the use of 18 registers. This␣
↪non compliance will
be fixed in the next release with the necessary versioning checks.
```

  - **RME**
    - add SMMU and PCIe information to Boot manifest (90552c6)
    - **TRP**
      - test el3-rmm ide km interface (8b3a89f)
    - **RMMD**
      - add FEAT_MEC support (7e84f3c)
      - add RMM_MECID_KEY_UPDATE call (f801fdc)
      - el3-rmm ide key management interface (2132c70)
      - verify FEAT_MEC present before calling plat hoook (609ada9)

    * **RMM**
        · add PCIe IO info to Boot manifest (bef44f6)
  - **SPM**
    * use xfer list with Hob list in SPM_MM (9ae5f67)
    * **EL3 SPMC**
        · ffa error handling in direct msg (e1168bc)
        · support Hob list to boot S-EL0 SP (4053a64)
        · use spmd_smc_switch_state after secure interrupt (a0a7f15)
    * **SPM MM**
        · move mm_communication header define to general header (b51436c)
  - **DRTM**
    * add platform API to retrieve ACPI tables region size (7792bdb)
    * ensure event types aligns with DRTM specification v1.1 (8d24a30)
    * introduce plat API for DLME authentication features (0f7ebef)
    * log No-Action Event in Event Log for DRTM measurements (2ec4488)
    * retrieve DLME image authentication features (94127ae)
    * update DLME data header with actual Event Log size (9753238)
    * validate launch features in DRTM parameters (8666bcf)
- **Libraries**
  - **CPU Support**
    * add ENABLE_ERRATA_ALL flag (593ae35)
    * add support for Alto CPU (940ecd0)
    * add sysreg_bit_toggle (bb80185)
  - **EL3 Runtime**
    * **RAS**
        · add asynchronous error type corrected (daeae49)
        · add eabort get helper function (ec6f49c)
  - **HOB**
    * copy StandaloneMm Hob creation library in edk2 (2105831)
    * introduce Hob creation library (8953568)
    * modify Hob creation code imported from edk2 (6b68b4a)
  - **PSCI**

    * allow cores to wake up from powerdown (2b5e00d)
    * remove cpu context init by index (ef738d1)
  - **ROMlib**
    * add PSA Crypto ROMLIB support (cf1b7fe)
  - **GPT**
    * statically allocate bitlocks array (b0f1c84)
  - **SMCCC**
    * add FEAT_TWED to ARCH_FEATURE_AVAILABILITY (d33ff5e)
    * implement SMCCC_ARCH_FEATURE_AVAILABILITY (8db1705)
  - **C Standard Library**
    * import qsort implementation (277713e)
  - **PSA**
    * add interface with RSE for retrieving entropy (1147a47)
  - **Firmware Handoff**
    * add 32-bit variant of ep info (7ffc1d6)
    * add 32-bit variant of SRAM layout (8001247)
    * add func to check and init a tl (f1d9459)
    * add lib to sp-min sources (79e7aae)
    * add Mbed-TLS heap info entry tag (0e932b8)
    * add transfer entry printer (937c513)
    * common API for TPM event log handoff (4d8b4ca)
    * transfer entry ID for TPM event log (9821775)
- **Drivers**
  - **Authentication**
    * add crypto_mod_finish() function (0331bd2)
    * add update of current_pk_oid in auth (9c18c35)
    * add util file for current pk_oid (1716805)
    * extend REGISTER_CRYPTO_LIB calls (95d49c6)
    * increase mbedtls heap for PSA RSA (2ffc28c)
    * introducing auth.mk (142ee34)
    * mbedtls psa key id mgmt (8a7505b)
    * **mbedTLS**

        - introduce crypto lib heap info struct (24da55e)
        - mbedtls config update for v3.6.2 (c307efc)
        - optimize SHA256 for reduced memory footprint (b57468b)
        - update mbedtls to version 3.6.3 (08f8c0a)

- **I/O**
    - add generic gpio spi bit-bang driver (3c54570)
- **Measured Boot**
    - add fw handoff event log utils (ca39163)
    - make event log lib standalone (cb03020)
- **TPM**
    - add Infineon SLB9670 GPIO SPI config (6fa56e9)
    - add tpm drivers and framework (36e3d87)
- **Arm**
    - **GIC**
        - add support for local chip addressing (c89438b)
- **NXP**
    - **Clock**
        - add a basic get_rate implementation (bd69113)
        - add base address for PERIPH_DFS (29f8a95)
        - add clock modules for uSDHC (cf6d73d)
        - add clock objects for CGM dividers (63d536f)
        - add get_rate for clock muxes (d1567da)
        - add get_rate for partition objects (a74cf75)
        - add get_rate for s32cc_clk (46de0b9)
        - add get_rate for s32cc_dfs (2fb2550)
        - add get_rate for s32cc_dfs_div (8f23e76)
        - add get_rate for s32cc_fixed_div (7c298eb)
        - add get_rate for s32cc_pll (fbebafa)
        - add get_rate for s32cc_pll_out_div (a762c50)
        - dynamic map of the clock modules (514c738)
        - enable MC_CGM dividers (2710bda)
        - get MC_CGM divider's parent (1586904)

- get MC_CGM divider's rate (ad412c0)
- get parent for the fixed dividers (35988a9)
- get pll rate using get_module_rate (43b4b29)
- restore pll output dividers rate (c23dde6)
- set MC_CGM divider's rate (f99078a)
- set the rate for partition objects (8501b1f)

– **ST**

  * **ST PMIC**

    - add defines for NVM shadow registers (c1222e7)

- **Miscellaneous**

  – add a generic EXTRACT macro (f963578)

  – add EXTRACT_FIELD macro for field extraction (af1dd6e)

  – implement strnlen secure and strcpy secure function (eb08889)

  – **AArch64**

    * add DBGPRCR_EL1 register accessors (bdcef87)

  – **FDTs**

    * **ST**

      - **STM32MP1**

        - **STM32MP15**

          - add Linux Automation GmbH Fairy Tux 2 (093c738)
          - add Linux Automation GmbH TAC (e67497f)
          - add SP_MIN versions of DT files (20544d6)

      - **STM32MP2**

        - add LPDDR4 files (64f82e5)
        - add STM32MP257F-DK board support (6a9e5ff)
        - add dual-ranked LPDDR4 config for STM32MP2 (99adf4d)
        - **STM32MP25**

          - enable WDQS for LPDDR4 (29917d3)
          - update 2GB DDR configs (388cb47)
          - add support for STM32MP257D-based ultra-fly-sbc board (a4e31cf)

- **Documentation**

  – update mboot threat model with dTPM (b00f6ec)

- **Build System**
  - rk3399: m0: add support for new binutils versions (6fbec46)
- **Tools**
  - **Secure Partition Tool**
    - add StMM memory region descriptor (3553087)
    - add the HOB list creation script (cc594af)
    - include HOB file in the TL pkg (32ecc0e)
    - invoke the HOB list creation code (2d317e8)
    - populate secure partition number in makefile (9327361)
    - specify endianness for HOB bin (49c6566)
    - transfer list to replace SP Pkg (0fe374e)
  - **Transfer List Compiler**
    - add –align argument (c4c8e26)
    - formalise random generation of TEs (157c619)

14.3.3 Resolved Issues

- **Architecture**
  - **CPU feature like FEAT_XXXX / ID register handling in general**
    - add support for 128-bit sysregs to EL3 crash handler (58fadd6)
    - add feat_hcx check before enabling FEAT_MOPS (484befb)
    - avoid using mrrs/msrr for tspd (f3e2b49)
    - improve xpaci wrapper (787977c)
    - include FEAT_MOPS declaration in aarch32 header (8656bda)
    - replace "bti" mnemonic with hint instructions (bdac600)
    - add a psb before updating context and remove context saving (f808873)
    - add a tsb before context switching (73d98e3)
- **Platforms**
  - **Arm**
    - create build directory before key generation (db69d11)
    - don't race on the build directory (9855568)
    - reinit secure and non-secure tls (2948d1f)
    - resolve build issue with ARM_ROTPK_LOCATION=regs option (4569a49)

- resolve dangling comments around macros (523c787)
- resolve misra rule R11.6 violation (307a533)
- update tsp_early_platform_setup prototype (9018b7b)
- use EL3_PAS in MAP_BL2_TOTAL definition (875423d)
- **Common**
  - add missing curly braces (7e288d1)
  - modify function to have single return (50029b9)
  - remove platform_core_pos_helper() (96e46f5)
  - remove fvp_r (2cadf21)
  - remove unused vfp code (18b129f)
  - add missing curly braces (0eeda63)
  - ignore the unused function return value (fc7a720)
- **CSS**
  - turn the redistributor off on PSCI CPU_OFF (50009f6)
- **FVP**
  - allow PSCI 0.2 in the device tree (94b500d)
  - exclude extend memory map TZC regions (06cec93)
  - increase EventLog size for OP-TEE with multiple SPs (d1a824e)
- **Juno**
  - resolve BL2 RAM overflow with RSA+ECDSA in GCC(14.2.1) (dd566a9)
- **Morello**
  - remove stray white-space in 'morello/platform.mk' (05533d9)
- **Neoverse-RD**
  - initialize CNTFRQ_EL0 for RESET_TO_BL31 (94a4383)
  - initialize timer before use in smmuv3_poll (64ff172)
  - set correct SVE vector lengths (842ba2f)
  - **RD-N2**
    - add LCA multichip data for RD-N2-Cfg2 (289578e)
    - correct RD-N2 StMM uuid format (6fb8d8c)
  - **RD-V3**
    - add console name to checksum calculation on RD-V3 (222c87e)
    - add LCA multichip data for RD-V3-Cfg2 (d0b93a0)

        · correctly define plat_mboot_measure_key() (fa0eb3c)
        · correctly handle FP regs context saving (ac05182)
        · fix comment for DRAM1 carveout size (4e2369c)
        · handle invalid build combination (fe488c3)

    * **TC**
      · define status to fix SPM tests (8d4d190)
      · eliminate unneeded MbedTLS dependency (22220e6)
      · enable certificate on the last secure partition (2e36131)
      · enable Last-level cache (LLC) for tc4 (7b41aca)
      · fix compilation error (26a520b)
      · fix SMMU streamId for tc4 gpu (bf223c7)
      · map mem_protect flash region (4bfe49e)
      · modify DPU configuration in dts for TC4 FPGA (bb9b893)
      · modify ethernet configuration for TC4 FPGA (8dec630)
      · modify gpio controller base addr for TC4 FPGA (5de9d79)
      · modify mmc configuration for TC4 FPGA (ba1faaf)
      · replace vencoder with simple panel for kernel > 6.6 (1d2d96d)
      · set console baurate to 38400 for fvp as well (5428938)
      · set system-coherency to 0(ACE-LITE) for tc4-gpu (cada6ca)

    * **Automotive RD**
      · **RD-1 AE**
        · fix rd1-ae device tree (f72eeb2)
        · rename legacy MPAM build option (7a5e580)

  – **Intel**
    * add FPGA isolation trigger when reconfiguration (6ce576c)
    * handle cold reset via physical reset switch (bf3877e)
    * redesign F2SOC bridge enable and disable flow for Agilex5 (42e9062)
    * this patch is used to solve DDR and VAB (458b40d)
    * update debug messages to appropriate class (a550aeb)
    * update ssbl naming conventions (bf2c213)
    * update warm reset routine and bootscratch register usage (646a9a1)
    * add in support for agilex5 b0 jtag id (8a0a006)

- **Marvell**
  - **Armada**
    - don't race on the UART_IMAGE (3395bd1)
- **MediaTek**
  - covert MTK_BL to uppercase for the build (c710579)
  - **MT8189**
    - fix mt8189 platform build failure (49d8678)
  - **MT8196**
    - add whole-archive option to prebuilt library (8f7d9bf)
    - fix wrong register offset of dptx on MT8196 (b38f8f7)
    - remove CPU_IDLE_SRAM_BASE entry from plat_mmap (83f37d9)
    - remove EC_SUSPEND_PIN initial setting (1185526)
    - remove SPM support for ES chip (ee2e99c)
- **NXP**
  - imx_trdc.h header guard (2e9198d)
  - **i.MX**
    - **i.MX 8M**
      - fix imx8mq build break (1b65be5)
      - **i.MX 8M Plus**
        - apply ERRATA_A53_1530924 erratum (e6d39da)
      - **i.MX 8Q**
        - fix imx8mq build break due to hab (3a36f70)
    - **i.MX 9**
      - **i.MX93**
        - trdc: restrict BLK_CTRL_S_AONMIX to secure world (c6bf928)
  - **S32G274A**
    - reduce the uSDHC clock to 200MHz (9c640e0)
- **QEMU**
  - fix register convention in BL31 for qemu (7ad6775)
  - fix RMM manifest checksum calculation (d08dca4)
  - ignore TPM error (ddb5e2f)
  - statically allocate bitlocks array (a32a77f)

    * **SBSA**
        - fix compilation error when accessing DT functions (33ac6f9)
- **QTI**
    * **MSM8916**
        - update tsp_early_platform_setup prototype (2f02426)
- **Raspberry Pi**
    * **Raspberry Pi 3**
        - expose BL1_RW to BL2 map for mboot (9acaade)
        - use correct name for include guards (5c0cbb2)
- **Renesas**
    * **R-Car**
        - use platform_def (99fe5c2)
- **Rockchip**
    * pmu: Do not mark already defined functions as weak (7f25d3c)
    * **PX30**
        - remove unused function (ce66647)
    * **RK3288**
        - remove unused function (6e38cc9)
    * **RK3399**
        - dram: Fix build with gcc 11 (f86c230)
        - fix unquoted .incbin for clang (ddd70f1)
        - m0: Makefile: fix outside array bounds warning (5049f91)
        - mark INCBIN-generated sections as SHF_ALLOC (279cad8)
    * **RK3588**
        - pmu: fix assembly symbol redefinition (f879388)
- **Socionext**
    * update tsp_early_platform_setup prototype (c45dbe7)
- **ST**
    * **STM32MP2**
        - correct early/crash console init (23647bd)
- **Texas Instruments**
    * fix UNUSED_VALUE in AM62L PSCI Driver (32302b4)

- **Xilinx**
  * avoid unexpected variable update (b3d25dc)
  * dcc console tests failing (e14ae4b)
  * modify function to have single return (906d589)
  * remove unused write_icc_asgi1r_el1() (1c12cd1)
  * resolve misra rule 10.3 violations (72eb16b)
  * resolve misra rule 10.4 violations (bdba3c8)
  * resolve misra rule 11.3 violations (c35fe29)
  * resolve misra rule 14.4 violation (a5d5cb3)
  * resolve misra rule 15.7 violations (fd44cc7)
  * resolve misra rule 2.2 violations (e5adcfc)
  * resolve misra rule 2.3 violations (09abae0)
  * resolve misra rule 2.7 violations (d87b0ce)
  * resolve misra rule 8.3 violations (3df32f8)
  * resolve misra rule 8.4 violations (4b4080d)
  * resolve misra rule 8.6 violations (eec03e9)
  * runtime console to handle dt failure (0791be8)
  * typecast expression to match data type (50ab135)
  * typecast expressions to match data type (83bcef3)
  * typecast operands to match data type (3a1a2da)
  * typecast operands to match data type (7d15b94)
  * update tsp_early_platform_setup prototype (470dd8b)
  * **Versal**
    · add unsigned suffix to match data type (4b23240)
    · handle invalid entry point in cpu hotplug scenario (435bc14)
    · modify function to have single return (890781d)
    · typecast expressions to match data type (b802b27)
    · typecast operands to match data type (8e4d5c6)
    · typecast operands to match data type (9b89de5)
  * **Versal NET**
    · add missing curly braces (9334fdf)
    · add unsigned suffix to match data type (baeeadd)

      - enable PSCI reset2 interface (5f22f57)
      - handle invalid entry point in cpu hotplug scenario (e5e417d)
      - modify function to have single return (5003a33)
      - remove_redundant_lock_defs (19799fd)
      - typecast expressions to match data type (3cbe0ae)
      - typecast operands to match data type (d51c8e4)
      - typecast operands to match data type (3dc93e5)
    - **ZynqMP**
      - add missing curly braces (e4a0c44)
      - align essential type categories (1877bf2)
      - fix length of clock name (f535068)
      - fix syscnt frequency for QEMU (55ae162)
      - handle invalid entry point in cpu hotplug scenario (df44616)
      - modify function to have single return (3f6d479)
      - typecast expression to match data type (e2cc129)
      - typecast expressions to match data type (895e802)
      - typecast operands to match data type (6ae9562)
      - typecast operands to match data type (2863b0c)
  - **AMD**
    - update transfer list args for OP-TEE (573ec22)
    - **Versal Gen 2**
      - add missing curly braces (9f51da5)
      - align QEMU APU GT frequency with silicon (f7a380e)
      - enable system reset (058edb8)
      - modify function to have single return (fb2fdcd)
      - pass tl address to bl32 (1fb3446)
      - rename console build arg to generic (2333ab4)
      - typecast expressions to match data type (fbc415d)
      - typecast operands to match data type (07be78d)
      - update DDR address map (66569a7)
      - update transfer list as optional (5cb9125)
- **Bootloader Images**

- **BL1**
  - prevent null pointer dereference (2d3b44e)
- **BL31**
  - add const qualifier (e358089)
  - add missing curly braces (88edd9c)
- **BL32**
  - **TSP**
    - use %u to display unsigned values (3b06438)

- **Services**
  - avoid altering function parameters (1a0f565)
  - **RME**
    - do not trap access to MPAM system registers in Realm mode (d048af0)
    - map DEVICE0_BASE as EL3_PAS (b577248)
    - remove ENABLE_PIE restriction (e126ed1)
    - **RMM**
      - add support for BRBCR_EL2 register for feat_brbe (41ae047)
  - **SPM**
    - **EL3 SPMC**
      - fix FF-A v1.2 version check (6feaad6)
      - move ERROR line inside conditional (bbf28dc)
    - **SPMD**
      - check pwr mgmt status for SPMC framework response (8723eaf)
      - fix build failure due to redefinition (a869e2d)
      - prevent SIMD context loss (8f60d99)
    - **SPM MM**
      - prevent excessive racing (48426cf)
  - **DRTM**
    - add missing DLME data regions for min size requirement (a65fa57)
    - adjust Event Log size in DLME (63d2020)
    - fix DLME data size check (28e8f9d)
    - sort the address-map in ascending order (7cf3784)
  - **TRNG**

        * allow FEAT_RNG_TRAP in dynamic fashion (bc30945)

    - **ERRATA ABI**

        * add support for handling split workarounds (bbff267)

- **Libraries**

    - **CPU Support**

        * add missing add_erratum_entry (a74b009)
        * avoid SME related loss of context on powerdown (45c7328)
        * clear CPUPWRCTLR_EL1.CORE_PWRDN_EN_BIT on reset (c9f352c)
        * declare reset errata correctly (5cba510)
        * drop esb from the Neoverse N1 (e75eea7)
        * fix a typo in errata doc (845213e)
        * fix clang compilation issue (bdaf0d9)
        * remove errata setting PF_MODE to conservative (ac9f4b4)
        * workaround for accessing ICH_VMCR_EL2 (7455cd1)
        * workaround for Cortex-A710 erratum 3701772 (463b5b4)
        * workaround for Cortex-A715 erratum 2804830 (fcf2ab7)
        * workaround for Cortex-A715 erratum 3699560 (26437af)
        * workaround for Cortex-A720 erratum 3699561 (050c4a3)
        * workaround for Cortex-A720-AE erratum 3699562 (af5ae9a)
        * workaround for Cortex-A725 erratum 3699564 (d732300)
        * workaround for Cortex-X2 erratum 3701772 (ae6c7c9)
        * workaround for Cortex-X3 erratum 3701769 (77feb74)
        * workaround for Cortex-X4 erratum 2923985 (cc46166)
        * workaround for Cortex-X4 erratum 2957258 (09c1edb)
        * workaround for Cortex-X4 erratum 3701758 (38401c5)
        * workaround for Cortex-X925 erratum 2963999 (29bda25)
        * workaround for Cortex-X925 erratum 3701747 (511148e)
        * workaround for CVE-2024-5660 for Cortex-A710 (0d7b503)
        * workaround for CVE-2024-5660 for Cortex-A77 (aed3e8b)
        * workaround for CVE-2024-5660 for Cortex-A78 (c818bf1)
        * workaround for CVE-2024-5660 for Cortex-A78_AE (902dc0e)
        * workaround for CVE-2024-5660 for Cortex-A78C (46a4cad)

    * workaround for CVE-2024-5660 for Cortex-X1 (26293a7)
    * workaround for CVE-2024-5660 for Cortex-X2 (5b58142)
    * workaround for CVE-2024-5660 for Cortex-X3 (b0d441b)
    * workaround for CVE-2024-5660 for Cortex-X4 (af65cbb)
    * workaround for CVE-2024-5660 for Cortex-X925 (ebc090f)
    * workaround for CVE-2024-5660 for Neoverse-N2 (26e0ff9)
    * workaround for CVE-2024-5660 for Neoverse-V1 (85709f6)
    * workaround for CVE-2024-5660 for Neoverse-V2 (878464f)
    * workaround for CVE-2024-5660 for Neoverse-V3 (ad3da01)
    * workaround for Neoverse-N2 erratum 3701773 (adea6e5)
    * workaround for Neoverse-N3 erratum 3699563 (fded839)
    * workaround for Neoverse-V3 erratum 2970647 (5f32fd2)
    * workaround for Neoverse-V3 erratum 3701767 (e25fc9d)
    * workaround for Cortex-A510 erratum 2971420 (f2bd352)

  - **EL3 Runtime**
    * add const qualifier (54c9c68)
    * add missing curly braces (858dc35)
    * for nested serrors, restore x30 to lower EL address (0bc3115)
    * replace CTX_ESR_EL3 with CTX_DOUBLE_FAULT_ESR (c722003)
    * make sure LTO doesn't garbage collect the handlers (f8d2a0e)
    * **Context Management**
      - change back owning security state when a feature is disabled (13f4a25)
      - don't access a field that doesn't exist (600717f)
      - fix context management SYSREG128 write macros (6595f4c)
    * **RAS**
      - fix status synchronous error type fields (9c17687)
      - fix typo in uncorrectable error type UEO (e5cd3e8)
    * **SIMD**
      - fix base register in fpregs_context_* (09ada2f)
  - **PSCI**
    * add const qualifier (7b97084)
    * add missing curly braces (c7b0a28)

* avoid altering function parameters (e64cdee)
* check if a core is the last one in a requested power level (71d4e03)
* initialise variables (382ba74)
* modify variable conflicting with external function (0839cfc)

- **ROMlib**
  * romlib build without MbedTLS (e4a070e)
- **SMCCC**
  * properly set RAS feature bit (04b80c1)
  * register PMUv3p5 and PMUv3p7 bits with the FEATURE_AVAILABILITY call (2bec665)
- **Translation Tables**
  * zeromem to clear all tables (aaacde4)
  * remove xlat_mpu (23302d4)
- **C Standard Library**
  * add const qualifier (b71d082)
  * explicitly check operators precedence (277d7dd)
  * make sure __init functions are garbage collected (53644fa)
  * remove __Nonnull type specifier (7ce483e)
  * typecast expressions to match data type (dd0d433)
  * typecast operands to match data type (26cc285)
- **Locks**
  * add missing curly braces (bd7ad5e)
- **PSA**
  * guard Crypto APIs with CRYPTO_SUPPORT (8a41106)
  * increase psa-mbedtls heap size for rsa (52d2934)
- **Firmware Handoff**
  * correct 8-bit modulo csum calculation (5ca0241)
  * fix message formatting of hex values (24e1ae2)
  * fix register convention in opteed (c0688c5)
  * remove XFERLIST_TB_FW_CONFIG (18be2db)

• **Drivers**
- **Console**
  * add missing curly braces (9ded5e8)

    * typecast expressions to match data type (97eefd9)

  - **Delay Timer**
    * create unique variable name (472cccb)

  - **MMC**
    * fix the length of the ocr defines (e02d365)
    * fix the length of the response type (7b4b3f2)

  - **GUID Partition Tables Support**
    * fix MBR header load (2fac89d)
    * initialise the mbr_entry variable (ec48d52)

  - **Arm**
    * add missing curly braces (03c6bb0)
    * align essential type categories (bec4a2c)
    * typecast expression to match data type (0f76d0d)
    * typecast expressions to match data type (edecc70)
    * **GIC**
      · quote the correct flag on error (df21ca0)
      · **GICv3**
        · do not assume redistributors are powered down (57f2d00)
    * **SMMU**
      · set root port CR0 GPCEN before ACCESSEN (8cc9724)

  - **Renesas**
    * **R-Car3**
      · disable A/B loader support by default (1a57115)

  - **ST**
    * **ST PMIC**
      · remove deadcode from STPMIC2 driver (bdbbf48)

- **Miscellaneous**
  - **AArch32**
    * avoid using r12 to store boot params (af61b50)

  - **FDTs**
    * **ST**
      · **STM32MP1**

        - re-enable RTC clock (33573ea)
        - **STM32MP15**
          - update clocks config for prtt1x board family (5aeb003)
      - **STM32MP2**
        - fix SDMMC slew rate (575d6dd)
  - **Security**
    - add CVE-2024-7881 mitigation to Cortex-X3 (b0521a1)
    - add CVE-2024-7881 mitigation to Cortex-X4 (6ce6aca)
    - add CVE-2024-7881 mitigation to Cortex-X925 (520c220)
    - add CVE-2024-7881 mitigation to Neoverse-V2 (56bb1d1)
    - add CVE-2024-7881 mitigation to Neoverse-V3 (037a15f)
    - add support in cpu_ops for CVE-2024-7881 (4caef42)
    - apply SMCCC_ARCH_WORKAROUND_4 to affected cpus (8ae6b1a)
    - enable WORKAROUND_CVE_2024_7881 build option (2372179)
  - **SDEI**
    - return SDEI_EINVAL if signaling state is incorrect (b142ede)
  - **TBBR**
    - remove tbbr_cot_bl1_r64.c (a2328f2)
- **Documentation**
  - fix the indent and the build command for MT8188 (cef56a5)
  - put INIT_UNUSED_NS_EL2 docs back (4557c0c)
  - update the instrumentation procedure (9b65ffe)
- **Build System**
  - do not force PLAT in plat_helpers.mk (422b181)
  - enable fp during fp save/restore (5141de1)
  - handle invalid spd build options (a0effb9)
  - include platform mk earlier (696ed16)
  - run sp_mk_gen.py with poetry (dd81623)
  - update clang target for aarch64 (af8947f)
- **Tools**
  - change data type to size_t for doimage (fbf6555)
  - **NXP Tools**

    * fix create_pbl buildroot build (634c7d8)
    * fix2 create_pbl buildroot build (bfe7f80)

  - **Certificate Creation Tool**
    * add default keysize to Brainpool ECDSA (0da16fe)
    * load openSSL configuration before PKCS11 operations (785c2c3)

  - **Firmware Encryption Tool**
    * put build_msg under LOG_LEVEL flag (7640df6)

  - **Renesas Tools**
    * **R-Car Layout Tool**
      · fix tool build (72f4b70)

  - **Transfer List Compiler**
    * add void entries to align data (03c2660)
    * pass the flags from client interface (537a25e)
    * relax entry addition from YAML files (f0e15dd)

- **Dependencies**
  - remove deprecated husky commands (b47dddd)

14.4 2.12.0 (2024-11-19)

The threat model for context management and the asymmetric CPU extension support feature is not available in the release.

14.4.1 ? BREAKING CHANGES

- **Bootloader Images**
  - remove unused plat_try_next_boot_source

    See: remove unused plat_try_next_boot_source (2c303e3)

14.4.2 Resolved Issues

- **Architecture**
  - **Branch Record Buffer Extension (FEAT_BRBE)**
    - allow RME builds with BRBE (9890eab)
  - **Memory Tagging Extension2**
    - improve ENABLE_FEAT_MTE deprecation warning (ba65e2d)
    - remove deprecated CTX_INCLUDE_MTE_REGS/FEAT_MTE (6f2b881)
- **Platforms**
  - **Allwinner**
    - dtb: check for correct error condition (7300a4d)
    - enable dtb modifications for CPU idle states to the rich OS (188a988)
    - remove unneeded header inclusion (8bb8f02)
  - **Arm**
    - **FPGA**
      - avoid stripping kernel trampoline (8292f24)
    - **FVP**
      - add DRAM memory regions that linux kernel can share (18ec9bd)
      - add optee specific mem-size attribute (75265a1)
      - add secure uart interrupt in device region (fc3a01a)
      - enable FEAT_MTE2 (d081c61)
      - fix the FF-A optee manifest by adding the boot info node (bf36351)
      - update the memory size allocated to optee at EL1 (4739372)
    - **Neoverse-RD**
      - **RD-V3**
        - remove NEED_* from RD-V3 makefile (a3eef39)
    - **TC**
      - add SCP_BL2 to RSE measured boot (7984154)
      - add stubs for soc_css_init functions (f5ae5dc)
      - correct CPU PMU binding (7aca660)
      - correct NS timer frame ID for TC (034cc80)
      - don’t enable TZC on TC3 (8ce29a7)

      · enable MTE2 unconditionally (be8eaa5)
      · fix the MHUv3 interrupt name in DT (1bf3325)
      · retain NS timer frame ID for TC2 as 0 (1ba0880)
    * **Corstone-1000**
      · fix Makefile error reporting (09bf366)
      · clean cache and disable interrupt before system reset (335c4f8)
      · include platform header file (783e5ab)
      · pass spsr value explicitly (32690ba)
      · remove unused NS_SHARED_RAM region (83c11c0)
      · update memory layout comments (d7417ad)
  – **Aspeed**
    * **AST2700**
      · fix mpll calculate statement (aa09622)
  – **HiSilicon**
    * **Poplar**
      · shutdown wdt0 before powering off (88bc65d)
      · use sysctrl module to reset (c961e68)
  – **Intel**
    * add cache invalidation during BL31 initialization (3c640c1)
    * add in JTAG ID for Linux FCS (ea906b9)
    * add in missing ECC register (4683946)
    * add in watchdog for QSPI driver (6704cba)
    * bridge ack timing issue causing fpga config hung (9a402d2)
    * correct macro naming (815245e)
    * f2sdram bridge quick write thru failed (64cf9de)
    * fix bridge enable and disable function (90f5283)
    * fix CCU for cache maintenance (f06fdb1)
    * flush L1/L2/L3/Sys cache before HPS cold reset (7ac7dad)
    * implement soc and lwsoc bridge control for burst speed (a8d81d6)
    * refactor SDMMC driver for Altera products (beba204)
    * remove redundant BIT_32 macro (7985ade)
    * software workaround for bridge timeout (e08039d)

        * update Agilex5 BL2 init flow and other misc changes (b3d2850)
        * update Agilex5 warm reset subroutines (c1253b2)
        * update all the platforms hand-off data offset value (1838a39)
        * update CCU configuration for Agilex5 platform (09330a4)
        * update mailbox SDM printout message (569a03c)
        * update memcpy to memcpy_s (e264b55)
        * update outdated code for Linux direct boot (21a01da)
        * update preloaded_bl33_base for legacy product (f29765f)
        * update sip smc config addr for agilex5 (7c72dfa)
        * update the size with addition 0x8000 0000 base (9978a3f)

    - **Marvell**
        * **Armada**
            · **A3K**
                · reset GIC before resetting via CM3 secure coprocessor (5993af4)

    - **MediaTek**
        * **MT8188**
            · remove BL32 region protection if SPD sets to none (207c447)

    - **NXP**
        * **i.MX**
            · disable DRAM retention by default on i.MX8MQ (108146c)
            · **i.MX 8M**
                · 8mq: enable imx_hab_handler (af79981)
                · ensure domain permissions for the console (f7434fa)
        * **S32G274A**
            · avoid overwriting const fields (bf01296)
            · workaround for ERR051700 erratum (b47d085)

    - **QEMU**
        * allocate space for GPT bitlock (e9bcbd7)
        * exclude GPT reserve from BL32_MEM_SIZE (7604288)
        * fix build error with spmd (1b1b40a)
        * fix EL3-SPMC data store alignment (eee52da)
        * fix L0 GPT page table mapping (147b1a6)

    * remove validate_ns_entrypoint (e5362e2)
    * update rmmd_attest_get_platform_token() (9248ee0)

- **Raspberry Pi**
  * **Raspberry Pi 3**
    · manually populate CNTFRQ reg (11dff59)
    · use correct define for GPIO reg_clr (9876baf)
- **Rockchip**
  * add parenthesis for BITS_SHIFT macro (901e94e)
  * fix "unexpected token" error with clang (52cdebb)
  * xlat: fix compatibility between v1 and v2 (d43a2e8)
- **ST**
  * set no-pie option when building ST elf file (6d26d75)
  * support device tree DDR sizes higher than 16Gbits for aarch64 (cd9c92c)
  * **STM32MP1**
    · remove unnecessary assert on GPIO_BANK_A value (5c45768)
    · skip OP-TEE header check if image base is NULL (b452e7a)
  * **STM32MP2**
    · enable timer earlier in BL31 (16a659d)
    · remove mapping of BL2 DT area (60d0758)
    · set PLAT_MAX_PWR_LVL to one (747d85e)
    · use TOOL_ADD_IMG_PAYLOAD for BL31 DT (f15f1c6)
- **Xilinx**
  * avoid altering function parameters (b21e287)
  * dcc to support runtime console scope (238eb54)
  * declare unused parameters as void (d3bb350)
  * explicitly check operators precedence (8e9a5a5)
  * fix comment about MEM_BASE/SIZE (1e2a5e2)
  * fix logic to read ipi response (03fa6f4)
  * fix OVERRUN coverity violation (e27b949)
  * handle power down event if SGI not registered (c3ffa4c)
  * map PMC_GPIO device node to interrupt for wakeup source (692d32b)
  * modify conditions to have boolean type (e223037)

- optimize logic to read IPI response (02943d0)
- register for idle callback (a3b0a34)
- rename variable to avoid conflict (aba5bf9)
- warn if reserved memory pre-exists in DT (729477f)
- **Versal**
  - add const qualifier (0f9f557)
  - add external declaration (16c611f)
  - declare unused parameters as void (ab9aab3)
  - evaluate condition for boolean (b39c82e)
  - explicitly check operators precedence (0ed8b4b)
  - kernel QEMU boot is failing on versal platform (8e5252f)
  - modify conditions to have boolean type (1247566)
  - remove check for bl32 load address (4c9ae8a)
  - variable conflicting with external linkage (e452826)
- **Versal NET**
  - evaluate condition for boolean (37c46d8)
  - declare unused parameters as void (06f63f4)
  - explicitly check operators precedence (a4ddd24)
  - ignore the unused function return value (aa6df8e)
  - modify conditions to have boolean type (83c3c36)
  - remove check for bl32 load address (c38ced2)
  - variable conflicting with external linkage (4d2b4e4)
- **ZynqMP**
  - add const qualifier (bb145c9)
  - add external declaration (6c08d1d)
  - declare unused parameters as void (1c43e36)
  - evaluate condition for boolean (aaf6e76)
  - explicitly check operators precedence (5b54231)
  - handle secure SGI at EL1 for OP-TEE (f5b2fa9)
  - ignore the unused function return value (355ccf8)
  - modify conditions to have boolean type (a42e6e4)
  - variable conflicting with external linkage (eda23fa)

- **AMD**
  - **Versal Gen 2**
    - add const qualifier (a0745f2)
    - add external declaration (17a8f41)
    - add ufs specific features support (b9c20e5)
    - correct the UFS clock rates (b048601)
    - declare unused parameters as void (851df3c)
    - explicitly check operators precedence (15a9e38)
    - ospi data integrity cases are failing (a147362)
    - update check for TRANSFER_LIST macro (7d09198)
    - variable conflicting with external linkage (ca39fd4)
- **Nuvoton**
  - fix MMU mapping settings (0a1df64)

- **Services**
  - **RME**
    - **RMMD**
      - continue boot if rmmd_setup fails (fdd8a24)
      - fail gracefully if RME is not enabled (eacbef4)
      - handle RMMD manifest loading failure (0c70781)
      - ignore SMC FID when RMM image is not present (adcd74c)
      - remove the assert check for RMM_BASE (8cb9c63)
  - **SPM**
    - **EL3 SPMC**
      - use write_el1_ctx_timer() macro to set cntkctl_el1 value (19082c2)
    - **SPMD**
      - remove spmd_handle_spmc_message (6c378c2)
    - **SPM MM**
      - carve out NS buffer TZC400 region (1922875)
  - **DRTM**
    - do cache maintenance before launching DLME (23378ae)
    - return proper values for DRTM get and set error SMCs (5e1fa57)

- **Libraries**

- **CPU Support**
  - modify the fix for Cortex-A75 erratum 764081 (7f152ea)
  - workaround for Cortex-A720 erratum 2792132 (b1bde25)
  - workaround for Cortex-A720 erratum 2844092 (1214090)
  - workaround for Cortex-X4 erratum 2816013 (1e4480b)
  - workaround for Cortex-X4 erratum 2897503 (609d08a)
  - workaround for Cortex-X4 erratum 3076789 (db7eb68)
  - workaround for Cortex-A520(2938996) and Cortex-X4(2726228) (4a97ff5)
- **EL3 Runtime**
  - correct CASSERT for cpu data size (483dc2e)
- **PSCI**
  - fix parent parsing in psci_is_last_cpu_to_idle_at_pwrlvl (01959a1)
- **ROMlib**
  - prevent race condition on the build directory (25cde5f)
  - wrap indirectly included functions (d95d56b)
- **GPT**
  - fix GPT library fill_l1_tbl() function (d024cce)
  - fix RME GPT library bug (6350aea)
- **Translation Tables**
  - correct attribute retrieval in a RME enabled system (e3c0869)
- **Authentication**
  - check the presence of the policy check function (491832f)
  - correct RSE_CRYPTO_EXPORT_PUBLIC_KEY_SID (759994a)
  - remove the bl2 static c file (ac106f2)
  - **mbedTLS**
    - fix error return code for calc_hash (885bd91)
    - sign verification issue with invalid Key/Signature (7731465)
    - add extra hash config to validate ROTPK (014975c)
  - **mbedTLS-PSA**
    - fix P-384 PSA key signature verification (12a8e95)
- **GUID Partition Tables Support**
  - fix unaligned access in load_mbr_header() (21a77e0)

- **Arm**
  - **GIC**
    - **GICv3**
      - fix GITS_CTLR.Quiescent bit definition (2da29d2)
      - incorrect impdef power down sequence (b1925dc)
      - wait rwp when gicr_ctrl.enablelpis from 1 to 0 (66668c7)
  - **MHU**
    - fix compilation error with ENABLE_ASSERTIONS=0 option (e2e8a39)
  - **RSE**
    - include lib-psa to resolve build (654ae70)
- **NXP**
  - **SFP**
    - shift gpio register offsets by 2 (d30312a)
  - **Clock**
    - broken UART clock initalization (f8490b8)
    - function parameter should not be modified (8ee0fc3)
- **ST**
  - **Clock**
    - adapt order of CSS on LSE and HSE (eca5103)
    - display proper PLL number for STM32MP13 (039b7d4)
    - do not reconfigure LSE (f4a2bb9)
  - **DDR**
    - fix coverity issue in ddrphyinit (5dd1d54)
    - move skipddc_dat definition (13cc1a5)
  - **GPIO**
    - configure each GPIO mux as secure for STM32MP2 (179a130)

- **Miscellaneous**
  - **DT Bindings**
    - update STM32MP2 clock and reset bindings (8522909)
  - **FDTs**
    - reserved memory: detect existing region (4248806)
  - **SDEI**

    * fix a crash when attempting to bind more events than are available (4096bd6)

- **Documentation**
  - fix CPU type for mt8195 (65ada75)
  - fix the example command for doc build (9db2b05)
  - point poetry readthedocs virtual env (5383a88)
  - refactor poetry dependency group (4a29299)
  - replace "ARM-TF" with "TF-A" in diagrams (c4067a9)
- **Build System**
  - correct feature assignment for ARM v8.8 compliance (94ff1d9)
  - ensure `$(ROT_KEY)` depends on correct directory rules (7a95759)
  - fix incorrectly-escaped armlink preprocessor definitions (df52e26)
  - pass the PLAT option during FIP tool compilation (40469bf)
  - string split into two lines causing error (4f32179)
- **Tools**
  - **fiptool**
    * update the fiptool and certtool to fix POSIX build (ccbfd01)
- **Dependencies**
  - **checkpatch**
    * detect issues in commit message (1a72174)

14.4.3 New Features

- **Architecture**
  - **Fine-grained Traps 2 (FEAT_FGT2).**
    * add support for FEAT_FGT2 (33e6aaa)
  - **CPU feature / ID register handling in general**
    * add ENABLE_FEAT_LS64_ACCDATA (19d52a8)
    * add new feature state for asymmetric features (43d1d95)
    * upgrade PMU to v8 (FEATURE_DETECTION) (515d2d4)
  - **Debug Extension (FEAT_Debugv8p9)**
    * add support for FEAT_Debugv8p9 (83271d5)
  - **Statistical profiling Extension (FEAT_SPE)**
    * introduce spe_disable() function (651fe50)

- **Trace Buffer Extension (FEAT_TRBE)**
  - introduce trbe_disable() function (b36e975)
- **Extension to SCTLR_ELx (FEAT_SCTLR2)**
  - enable FEAT_SCTLR2 for Realm world (b17fecd)
  - add support for FEAT_SCTLR2 (4ec4e54)
- **128-bit Translation Tables (FEAT_D128)**
  - add support for FEAT_D128 (3065513)
- **Translation Hardening Extension (FEAT_THE)**
  - add support for FEAT_THE (6d0433f)

- **Platforms**
  - **Allwinner**
    - adjust H616 L2 cache size in DTB (ee5b26f)
    - h616: add I2C PMIC support (0444589)
    - h616: add support for AXP313 PMIC (0385136)
    - h616: add support for AXP717 PMIC (646d06b)
  - **Arm**
    - **Common**
      - add support for loading CONFIG from BL2 (973e0b7)
      - add fw handoff support for RESET_TO_BL31 (1a0ebff)
      - correct the RESET_TO_BL31 x1 handoff arg (5da68cc)
      - load dt before updating entry point (c1c406a)
      - move HW_CONFIG relocation into BL31 (fe94a21)
      - remove critical handoff code from assert (cca1b72)
      - makefile invoke CoT dt2c (0e0fab0)
      - generate tbbr c file CoT dt2c (479c833)
      - add COT_DESC_IN_DTB option for Dualroot (731ac5e)
    - **FPGA**
      - enable new CPU features (1920a32)
    - **FVP**
      - change UART0-1 to NS device region (cd656a5)
      - add Cactus partition manifest for EL3 SPMC (5134623)
      - add cpu power control (d38c64d)

- add Dualroot CoT in DTB support (0af86f0)
- add flash areas for secure partition (9fb7676)
- add SPM manifest for OP-TEE at S-EL1 without S-EL2/Hafnium (41d73bf)
- allow SIMD context to be put in TZC DRAM (b4c23ad)
- fdts: add stdout-path to the Foundation FVPs (2faccab)
- replace managed-exit with ns-interrupts-action (887cec9)
- scale SP_MIN max size based on SRAM size (3b5eca9)
- update FF-A version to v1.1 supported by optee (4f37e1e)
- remove duplicate jumptable entry (180a3a9)

* **Neoverse-RD**
  - add a routine to update NT_FW_CONFIG in BL31 (c6b27c4)
  - add CSS definitions for third gen platforms (6d52713)
  - add DRAM layout for third gen platforms (10eb4c4)
  - add firmware definitions for third gen platforms (e517ccf)
  - add MHUv3 channels on third gen multichip platforms (47348b1)
  - add MHUv3 doorbell channels on third gen platforms (46d474f)
  - add multichip pas entries (c72e9dc)
  - add pas definitions for third gen platforms (896e9aa)
  - add RoS definitions for third gen platforms (fad5a20)
  - add scope for RD-Fremont variants (84973bb)
  - add SRAM layout for third gen platforms (5a37d68)
  - allow RESET_TO_BL31 for third gen platforms (4abcfd8)
  - enable RESET_TO_BL31 for RD-V3 (527fc46)
  - **RD-V3**
    - add DRAM pas entries in pas table for multichip (6a9cf0e)
    - add implementation for GPT setup (0876c74)
    - add support for measured boot at BL1 and BL2 (6182950)
    - add support for RD-Fremont (c0513e0)
    - add support for RD-Fremont-Cfg1 (6a0cb48)
    - add support for RD-Fremont-Cfg2 (eedb2d8)
    - enable AMU if present on the platform (faf98b3)
    - enable MPAM if present on the platform (e951985)

- enable MTE2 if present on the platform (f801377)
- enable SVE for SWD and NS (7e2736b)
- fetch attestation key and token from RSE (0e323ec)
- helper to initialize rse-comms with AP-RSE MHUv3 (2a35fcd)
- initialize GPT on GPC SMMU block (ba35fac)
- initialize the rse comms driver (f546113)
- integrate DTS files for RD-Fremont variants (1b96641)
- update Root registers page offset for SMMUv3 (859355f)
- set CTX_INCLUDE_SVE_REGS build flag for RD-V3 variants (1551834)

* **TC**
  - add default SLC policy for the gpu (bebefe0)
  - add device tree binding for SPE (77080f6)
  - add device tree binding for TC4 (3cedc47)
  - add DSU PMU node for tc3 (d3ae677)
  - add dts entries for MCN PMU nodes (1401a42)
  - add MHUv3 addresses between RSS and AP (5ab7a2f)
  - add MHUv3 doorbell support on TC3 (4f65c0b)
  - add MHUv3 DT binding for TC3 (6c069e7)
  - add MHUv3 register addresses for TC4 (36ffe3e)
  - add new TC4 RoS definitions (e9e83e9)
  - add NI-Tower PMU node for TC3 (169eb7d)
  - add PPI partitions in DT binding (ebc991b)
  - add system generic timer register definition for TC4 (d6b6a8b)
  - add uart node in spmc manifest (880dcd0)
  - allow TARGET_VERSION=4 (e8e1b60)
  - bind DPU SMMU on TC4 (e365479)
  - bind GPU SMMU on TC4 (11ec5de)
  - bind SCMI over MHUv3 for TC3 (f2596ff)
  - bind SMMU-600 with the DPU on TC3 FPGA (4c6960c)
  - bind SMMU-700 with DPU on TC3 (0458d3a)
  - change GIC DT property 'interrupt-cells' to 4 (1300bbc)
  - configure MCN rdalloc and wralloc mode (bb04d02)

- enable el1 access to DSU PMU registers (de8b9ce)
- enable Last-level cache (LLC) (e1b76cb)
- enable MCN non-secure access to pmu counters on TC3 (adc91a3)
- enable SME and SME2 options for TC4 (9face21)
- enable trbe errata flags for Cortex-A520 and X4 (74dc801)
- make SPE feature asymmetric (7754b77)
- make TCR2 feature asymmetric (3e8a82a)
- move flash device to own node (62269d4)
- provide target_locality info of AP FW components (3201faf)
- remove static memory used for fwu (25a2fe3)
- setup ni-tower non-secure access for TC3 (89c58a5)
- specify MHU version based on platform (04085d6)
- support full-HD resolution for the FVP model (dd5bf9c)
- update DT for Drage GPU (b3a4f8c)

* **Corstone-1000**
  - add multicore support for fvp (16f4862)
* **Automotive RD**
  - **RD-1 AE**
    - add device tree files (bb7c7e7)
    - enabling Trusted Board Boot(TBB) for RD-1 AE (2638496)
    - introduce Arm RD-1 AE platform (f661c74)
    - introduce BL31 for RD-1 AE platform (daf934c)

- **Aspeed**
  * **AST2700**
    - set up CPU clock frequency by SCU (e3d1bbd)
- **Intel**
  * add build option for boot source (ef8b05f)
  * add in SHA384 authentication (cab83c3)
  * add QSPI get devinfo mailbox cmd (8fb1b48)
  * clock manager PLL configuration for Agilex5 platform (e60bedd)
  * direct boot from TF-A to Linux for Agilex (b5c3a3f)
  * enable VAB support for Intel products (3eb5640)

    * pinmux and power manager config for Agilex5 platform (94a546a)
    * update Agilex5 DDR and IOSSM driver (ce21a1a)
    * update BL2 platform specific functions (fa1e92c)
    * update hand-off data to include agilex5 params (6875d82)

- **MediaTek**
    * change log level from INFO to VERBOSE (5f2f384)
    * configure DEV_IRQ as G1S interrupt (240a1ec)
    * move plat_helpers.h to the common folder (b741293)
    * **MT8186**
        - add common and MT8186 TRNG driver (8c1740e)
    * **MT8188**
        - add MT8188 TRNG driver (b88d1f5)
        - update SVP region ID and permission (fc77c69)
        - update SVP region ID protection flow (e66c4ea)
        - update the memory usage for SCP core0 and core1 (83112aa)
    * **MT8192**
        - update memory protect region (7587cfd)
    * **MT8195**
        - update memory protect region (4224783)
- **NXP**
    * **i.MX**
        - add helper to take params from BL2 (7eae1db)
        - **i.MX 8M**
            - **i.MX 8M Nano**
                - optionally take params from BL2 (c37a877)
            - **i.MX 8M Mini**
                - optionally take params from BL2 (11d32b3)
            - **i.MX 8M Plus**
                - optionally take params from BL2 (3d9fea9)
        - **i.MX 9**
            - **i.MX93**
                - optionally take params from BL2 (02d1813)

  - **S32G274A**
    - add ncore support (5071f7c)
    - enable BL2 early clocks (66af542)
    - enable workaround for ERR051700 (cc6e9b0)
    - use s32cc clock driver (f1e4ac5)
- **QEMU**
  - **SBSA**
    - handle the information of CPU topology (c891b4d)
- **Raspberry Pi**
  - **Raspberry Pi 5**
    - add PCI SMCCC support (682607f)
- **Renesas**
  - **R-Car**
    - **R-Car 3**
      - populate kaslr-seed in next stage DT (b9e34d1)
- **Rockchip**
  - add RK3566/RK3568 Socs support (9fd9f1d)
  - **RK3588**
    - enable crypto function (b833bbe)
    - support rk3588 (e3ec6ff)
    - support SCMI for clock/reset domain (04150fe)
- **ST**
  - add FWU with boot from NAND (795a559)
  - add stm32mp_is_wakeup_from_standby() (87cd847)
  - manage backup partitions for NAND devices (ae81d48)
  - manage BL31 FCONF load_info struct (aa7f6cd)
  - **STM32MP1**
    - always boot at 650MHz (f655922)
    - handle DDR power supplies (47e6231)
    - **STM32MP15**
      - remove OP-TEE shared mem (8dd2a64)
  - **STM32MP2**

- add BL2 boot first steps (db77f8b)
- add BL31 device tree support (27dd11d)
- add defines for the PWR peripheral (6add715)
- add fixed regulators support (c3a7534)
- add fw-config compilation (5af9369)
- add helper to get DDRDBG base address (2fd7b23)
- add minimal support for BL31 (03020b6)
- add RETRAM map/unmap capability (52f530d)
- add RISAB registers description (631c5f8)
- boot BL33 at EL1 or EL2 (c900760)
- disable unsupported features (128df96)
- display CPU info (381b2a6)
- enable DDR driver (213a08e)
- enable DDR sub-system clock (5e0be8c)
- get chip ID (154e6e6)
- handle DDR power supplies (e2d6e5e)
- improve BL31 size management (64e5a6d)
- initialize gic and delay timer in bl31_plat_arch_setup (77847f0)
- introduce DDR type compilation flags (d07e946)
- load FW binaries to DDR (9a0cad3)
- load fw-config file (a846a23)
- manage DDR FW via FIP (ae84525)
- print board info (cdaced3)

– **Texas Instruments**
  * implement DM_MANAGED suspend (9b7550f)

– **Xilinx**
  * add feature check function for TF-A specific APIs (9a0f5d1)
  * add none console (6d41398)
  * remove PM_IOCTL and PM_QUERY_DATA APIs (924f8ce)
  * update SiP SVC version number (c26aa08)
  * update TF-A to passthrough all PLM commands (4661c8f)
  * **Versal**

        · add DTB console to platform.mk (d629db2)
        · add support for QEMU COSIM platform (db827f9)
        · dedicate console for boot and runtime (d533f58)
        · deprecate build time arg VERSAL_PLATFORM (09ac1ca)

    * **Versal NET**
        · add DTB console to platform.mk (d61ba95)
        · dedicate console for boot and runtime (28ad0e0)
        · set lower cluster bus qos value (c6f6202)

    * **ZynqMP**
        · add DTB console to platform.mk (09a02ce)
        · dedicate console for boot and runtime (4557ab6)
        · enable ENABLE_LTO flag (19d8756)
        · move zynqmp platform to xlat tables v2 (fdda980)

  – **AMD**
    * populate handoff from TL (1fbe81f)
    * **Versal Gen 2**
        · add dtb & runtime console (1196474)
        · add dummy implementation for SCMI PD (095a20a)
        · add support for AMD Versal Gen 2 platform (c97857d)
        · implement USB_SET_STATE dummy IOCTL (282bce1)
        · support dynamic XLAT tables (9aa71f4)

- **Bootloader Images**
  – add plat handler for image loading (a03dafe)
  – **BL32**
    * setup GPT in BL31 in RESET_TO_BL31 boot flow (1547e5e)

- **Services**
  – **RME**
    * **RMMD**
        · el3 token sign during attestation (6a88ec8)
  – **SPM**
    * **EL3 SPMC**
        · support simd context management upon world switch (59bdcc5)

    * **SPM MM**
        · switch to simd_ctx_save/restore APIs (e6e3486)
  - **Secure Payload Dispatcher**
    * **ProvenCore**
        · switch to simd_ctx_save/restore apis (a9b64ed)
    * **Trusty**
        · switch to simd_ctx_save/restore apis (7461025)

- **Libraries**
  - **CPU Support**
    * add support for arcadia cpu (8fa5460)
    * add support for cortex-a720ae (8118078)
    * add sysreg_bitfield_insert_from_gpr macro (ad8b514)
  - **EL3 Runtime**
    * **Context Management**
        · context switch MDCR_EL3 register (123002f)
        · introduce EL3/root context (40e5f7a)
        · add Root-Context documentation(0f3cd51)
        · enhance the cpu_context memory report (781e1a4)
        · move mpam registers into el2 context (7d930c7)
        · convert el1-ctx assembly offset entries to c structure (42e35d2)
        · add explicit context entries for ERRATA_SPECULATIVE_AT (59b7c0a)
        · remove el1 context when SPMD_SPM_AT_SEL2=1 (a0674ab)
        · support for asymmetric feature among cores (2f41c9a)
        · asymmetric feature support for trbe (721249b)
        · handle asymmetry for FEAT_TCR2 (f4303d0)
        · handle asymmetry for SPE feature (188f8c4)
        · test integrity of el1_ctx registers (7623e08)
        · keep actlr_el2 value in the init context (0aa3284)
    * **SIMD**
        · add data struct for simd ctxt management (841533d)
        · add routines to save, restore sve state (6d5319a)
        · add rules to rationalize simd ctxt mgmt (3524d07)

- add sve state to simd ctxt struct (4242262)
- introduce simd context helper APIs (308ebfa)

- **GPT**
  - change the default max GPT block size to 512MB (01faa99)
  - add support for large GPT mappings (ec0088b)
  - configure memory size protected by bitlock (d766084)
- **C Standard Library**
  - avoid CWE-190 for GENMASK macros (1f0b6e7)
  - fix MISRA 12.2 violations for BIT32 and BIT64 macros (0605b7e)
- **PSA**
  - introduce generic library for CCA attestation (98d36e5)
- **Firmware Handoff**
  - fix register convention r1/x1 value on transfer list (7475815)
  - make tl generation flexible (2329e22)

- **Drivers**
  - **Generic Clock**
    - add set_parent callback (a2c6016)
    - add set_rate callback (19f9e2e)
  - **NXP**
    - add clock skeleton for s32cc (3a580e9)
    - add Linflex flush callback (95ac568)
    - **Clock**
      - add A53 clock objects (44e2130)
      - add ARM PLL enablement (b5101c4)
      - add ARM PLL ODIV enablement (84e8208)
      - add CGM0 instance (9dbca85)
      - add clock objects for ARM DFS (44ae54a)
      - add clock objects for ARM PLL (a8be748)
      - add dependencies for the XBAR clock (5692f88)
      - add DFS module enablement (4cd04c5)
      - add FXOSC clock enablement (8ab3435)
      - add get_parent callback (96e069c)

- add MC_CGM clock objects (3fa91a9)
- add MC_ME utilities (b8c68ad)
- add minimal set of S32CC clock ids (086ee20)
- add objects needed for DDR clock (4a2ca71)
- add oscillator clock objects (7c36209)
- add partition reset utilities (11a7c54)
- add partitions objects (af3020e)
- add PERIPH PLL enablement (8653352)
- add set_parent callback (12e7a2c)
- enable the A53 clock (7004f67)
- enable the DDR clock (8a4f840)
- enable the XBAR clock (b8ad880)
- enable UART clock (e4462da)
- implement set_rate for oscillators (d937351)
- refactor clock enablement (5300040)
- set parent for ARM PLL and MC_CGM muxes (83af450)
- set rate for clock fixed divider (65739db)
- set rate for clock muxes (64e0c22)
- set rate for PLL divider objects (de950ef)
- set rate for PLL objects (7ad4e23)
- setup the DDR PLL (18c2b13)

– **ST**

  * **Clock**
    - add function to restore generic timer rate (bfe8a12)
    - add STM32MP2 clock driver (615f31f)
    - don't gate/ungate an oscillator if it is not wired (f2aebab)
    - update with new bindings (ae1e503)
    - use early traces (1a25db1)
  * **DDR**
    - add STM32MP2 driver (79629b1)
  * **GPIO**
    - add set GPIO config API (bfa5f61)

      * **ST PMIC**
        - add STPMIC2 driver (817f42f)
      * **Regulator**
        - add enable ramp-delay (6897ae8)
        - support regulator_set_voltage for fixed regulator (156ed97)
      * **Reset**
        - add stm32mp2_reset driver (f829d7d)
        - add system reset management (d91d10a)

- **Miscellaneous**
  - **DT Bindings**
    * add missing SPIx bus clocks (c6d50c9)
    * describe ST GPIO banks and config (deb9c86)
    * introduce Dualroot CoT DTB (703df3a)
    * new RCC DT bindings (52b253b)
  - **FDT Wrappers**
    * add function to read uint64 with default value (bc8dfca)
  - **FDTs**
    * add DDR4 files for STM32MP2 (178aef6)
    * **STM32MP1**
      - move RNG1 to CSI to improve random generation (d594239)
      - new RCC DT bindings for STM32MP1 (4391e5e)
      - remove PLL1 settings (66d7c8b)
      - remove RTC clock configuration (703a581)
    * **STM32MP2**
      - add BL31 info in fw-config (a370c85)
      - add clock tree for STM32MP257F-EV1 (293a4f3)
      - add fw-config file (513b5cc)
      - add fw-config files for STM32MP257F-EV1 (83f571e)
      - add I2C7 pin muxing (0a08208)
      - add io_policies (53e8982)
      - add memory node (e34839b)
      - add SD-card and eMMC support on STM32MP257F-EV1 (1dafb40)

        · add sdmmc nodes in SoC DT file (3879761)
        · add sdmmc pins definition (6a85f67)
        · add UART and I2C nodes for STM32MP2 (c7cfe27)
        · describe stpmic2 power supplies (e974670)
        · remove pins-are-numbered (a1a50ef)
        · update STM32MP257F-EV1 DT (f0d6dcb)
        · **STM32MP25**
          · add DDR power supplies (7323c7f)
          · add DDRCTRL and DDRPHY settings in DDR node (56ac99a)

- **Documentation**
  - add DPE to RSE design doc (e4582e4)
  - add RMM option in build-options.rst (1b7f51e)
  - add RSE provided mboot backends to the threat model (3849d27)
  - add STM32MP2 docs links (21b6260)
  - update mboot threat model (07c2d18)
- **Build System**
  - add ability to define platform specific defaults (1b2fb6a)
  - add ctags recipes for indexing assembly files (54b773e)
- **Tools**
  - **Transfer List Compiler**
    - add command gen-header (9b05c37)
    - add host tool for static TL generation (6ac31f3)
    - add support for tox (38487c7)
    - add creating transfer lists from yaml files (3112099)
    - add option to input attr as string of flag names (4dcbba9)
    - add option to input text instead of tag id number (792e8e8)
  - **Chain of Trust device tree to C source file**
    - standalone CoT dt2c tool (4274d6f)
    - fix various breakages (73f7b7d)
    - use processed Device Tree source file as input (e19977d)
    - update documentation for cot-dt2c (b95f398)

14.5 2.11.0 (2024-05-17)

14.5.1 ? BREAKING CHANGES

- **Architecture**
  - **Memory Tagging Extension2**
    - Any platform or downstream code trying to use SCR_EL3.ATA bit(26) will see failures as this is now moved to be used only with FEAT_MTE2 with commit@ef0d0e5478a3f19cbe70a378b9b184036db38fe2

      See: remove mte, mte_perm (c282384)
- **Services**
  - **SPM**
    - **SPMD**
      - Given the optimizations made in TF-A SPMD to simplify NS EL1 context management, platform integrators must use SPMC binaries built by picking commits after 2fc6dcfa97e05159f95859fcf68db3031586f8c7 from hafnium repository.

        See: skip NS EL1 context save & restore operations (2d960a1)
- **Drivers**
  - **Arm**
    - **RSE**
      - remove PLAT_RSS_NOT_SUPPORTED build option

        See: remove PLAT_RSS_NOT_SUPPORTED build option (878354a)
  - **FWU**
    - add a config flag for including image info in the FWU metadata (11d05a7)
    - add a function to obtain an alternate FWU bank to boot (26aab79)
    - add some sanity checks for the FWU metadata (d2566cf)
    - document the config flag for including image info in the FWU metadata (7ae1619)
    - migrate FWU metadata structure to version 2 (a89d58b)

14.5.2 New Features

- **Architecture**
  - **CPU feature / ID register handling in general**
    - add cortex-a35 l2 extended control register (a727d59)
    - add feature detection for FEAT_CSV2_3 (30019d8)
    - added few helper functions (30f05b4)
  - **DynamIQ Shared Unit (DSU)**
    - save/restore DSU PMU register (f99a69c)
  - **Memory Tagging Extension2**
    - add mte2 feat (8e39788)
- **Platforms**
  - update SZ_* macros (6d511a8)
  - **Arm**
    - add COT_DESC_IN_DTB option for CCA CoT (b76a43c)
    - add trusty_sp_fw_config build option (0686a01)
    - move GPT setup to common BL source (341df6a)
    - retrieve GPT related data from platform (86e4859)
    - support FW handoff b/w BL1 & BL2 (9c11ed7)
    - support FW handoff b/w BL2 & BL31 (a5566f6)
    - add platform API that gets cluster ID (e6ae019)
    - **CSS**
      - initialise generic timer early in the boot (3447ba1)
    - **FVP**
      - add CCA CoT in DTB support (4c79b86)
      - add stdout-path (8c30a0c)
      - add support for virto-net, virtio-9p and virtio-rng (51b8b9c)
      - added calls to unprotect/protect memory (6873088)
      - delegate FFH RAS handling to SP (d07d4d6)
      - remove left-over RSS usage (a1726fa)
    - **Neoverse-RD**
      - add scope for RD-V1 (86a4949)

- add scope for RD-V1-MC (6fb16da)
- add scope for SGI-575 (18b5070)
- disable SPMD_SPM_AT_SEL2 for A75/V1/N1 platforms (b9c3273)
- disable SPMD_SPM_AT_SEL2 for N2/V2 platforms (301c017)
- enable AMU if supported by the platform (fed9368)
- remove unused SGI_PLAT build-option (2d32517)
- **SGI-575**
  - remove SGI-575 from deprecated list (f104eec)
- **RD-E1-Edge**
  - remove support for RD-E1-Edge (c69253c)
- **RD-N1-Edge**
  - remove RD-N1-Edge from deprecated list (78b7939)
- **RD-N2**
  - enable NEOVERSE_Nx_EXTERNAL_LLC flag (ab2b363)
  - add dts for secure partition (49df726)
  - enable AMU if present on the platform (2cfedfa)
  - enable MTE2 if present on the platform (3a5b375)
  - update power message value to 0 (08f6398)

* **TC**
  - add arm_ffa node in dts (4fc4e9c)
  - add DPE backend to the measured boot framework (e7f1181)
  - add DPE context handle node to device tree (1f47a71)
  - add dummy TRNG support to be able to boot pVMs (7be391d)
  - add firmware update secure partition (d062872)
  - add memory node in the device tree (5ee4deb)
  - add PMU entry (553b06b)
  - add RSS SDS region right after SCMI payload (6f503e0)
  - add save/restore DSU PMU register support (b87d7ab)
  - add SCMI power domain and IOMMU toggles (a658b46)
  - add spmc manifest with trusty sp (ba197f5)
  - add TC3 platform definitions (62320dc)
  - allow booting from DRAM (18f754a)

- choose the DPU address and irq based on the target (8e94163)
- enable gpu/dpu scmi power domain and also gpu perf domain (127eabe)
- factor in FVP/FPGA differences (1b8ed09)
- get the parent component provided DPE context_handle (467bdf2)
- group components into certificates (6df8d76)
- interrupt numbers for `smmu_700` (2c406dd)
- introduce an FPGA subvariant and TC3 CPUs (a02bb36)
- pass the DTB address to BL33 in R0 (638e4a9)
- provide a mock mbedtls-random generation function (a877818)
- share DPE context handle with child component (03d388d)

– **Intel**

* add in QSPI ECC for Linux (4d122e5)
* enable query of fip offset on RSU (6cbe2c5)
* enable SDMMC frontdoor load for ATF->Linux (32a87d4)
* increase bl2 size limit (2d46b2e)
* restructure watchdog (47ca43b)
* support QSPI ECC Linux for Agilex (d6ae69c)
* support QSPI ECC Linux for N5X (6cf16b3)
* support QSPI ECC Linux for Stratix10 (8be16e4)
* support query of fip offset using RSU (62be2a1)
* support SDM mailbox safe inject seu error for Linux (fffcb25)
* support wipe DDR after calibration (68bb3e8)

– **MediaTek**

* remove bl32 flag for mtk_bl (9c41cc1)
* **MT8188**
  - add secure iommu support (5fb5ff5)
  - remove apusys kernel handler usage constraints (0c77651)

– **NXP**

* **i.MX**
  - **i.MX 8M**
    - add 3600 MTps DDR PLL rate (f1bb459)
    - add defines for csu_sa access security (81de503)

    - add imx csu_sa enum type defines for imx8m (2ac4909)
    - make bl33 start configurable via PRELOADED_BL33_BASE (9260a8c)
    - obtain boot image set for imx8mn/mp (6d2c502)
    - **i.MX 8M Mini**
      - restrict peripheral access to secure world (1156c76)
      - set and lock almost all peripherals as non-secure (f4b11e5)
    - **i.MX 8M Plus**
      - restrict peripheral access to secure world (0324081)
      - set and lock almost all peripherals as non-secure (cba7daa)
    - **i.MX 8Q**
      - detect console base address during runtime (52ee817)
  - **i.MX 8ULP**
    - add a flag check for the ddr status (4fafccb)
    - add APD power down mode(PD) support in system suspend (478af8d)
    - add i.MX8ULP basic support (fcd41e8)
    - add memory region policy (5fd0642)
    - add OPTEE support (e7b82a7)
    - add some delay before cmc1 access (c514d3c)
    - add system power off support (891c547)
    - add the basic support for idle & system suspned (daa4478)
    - add the initial XRDC support (ac5d69b)
    - add trusty support (e853041)
    - adjust the dram mapped region (8d50c91)
    - adjust the voltage when sys dvfs enabled (416c443)
    - allocated caam did for the non secure world (7c5eedc)
    - allow RTD to reset APD through MU (ea1f7a2)
    - ddrc switch auto low power and software interface (ee25e6a)
    - enable 512KB cache after resume on imx8ulp (bcca70b)
    - enable the DDR frequency scaling support (caee273)
    - give HIFI4 DSP access to more resources (351976b)
    - not power off LPAV PD when LPAV owner is RTD (ab787db)
    - protect TEE region for secure access only (ff5e179)

        - update the upower config for power optimization (36af80c)
        - update XRDC for ELE to access DDR with CA35 DID (d159c00)
    - **S32G274A**
      - add S32G274ARDB2 board support (8b81a39)
      - enable BL31 stage (e73c3c3)
- **QEMU**
  - allow ARM_ARCH_MAJOR/MINOR override (e769f83)
  - enable FEAT_ECV when present (1b694c7)
  - enable transfer list to BL31/32 (305825b)
  - load and run RMM image (8ffe0b2)
  - setup Granule Protection Table (6cd113f)
  - setup memory map for RME (cd75693)
  - support TRP for RME (ebe82a3)
  - update mapping types for RME (a5ab1ef)
  - update to manifest v0.3 (762a1c4)
  - use mock attestation functions for RME (c69e95e)
  - **SBSA**
    - handle CPU information (42925c1)
    - handle memory information (8b7dd83)
    - mpidr needs to be present (4fc54c9)
- **Raspberry Pi**
  - add Raspberry Pi 5 support (f834b64)
- **Renesas**
  - **R-Car**
    - **R-Car 3**
      - add cache operations to boot process (7e06b06)
      - change CAM setting to improve bus latency of R-Car Gen3 (e366f8c)
      - change MMU configurations (5e8c2d8)
      - enable the stack protection (cfa466a)
      - update IPL and Secure Monitor Rev.4.0.0 (516a98e)
- **ST**
  - add a function to clear the FWU trial state counter (6e99fee)

    * add logic to boot the platform from an alternate bank (6166051)
    * do not directly call BSEC functions in common code (3007c72)
    * get the state of the active bank directly (588b01b)
    * use stm32_get_otp_value_from_idx() in BL31 (189db94)
    * **STM32MP1**
        · only fuse monotonic counter on closed devices (d6bb94f)
    * **STM32MP2**
        · add BSEC and OTP support (197ac78)
        · add ddr-fw parameter for fiptool (e494afc)
        · add plat_my_core_pos (d1c85da)
        · add STM32MP_USB_PROGRAMMER compilation (2e905c0)
        · put back core 1 in wfi after debugger's halt (2331a34)
        · use early traces (47ea303)
  – **Xilinx**
    * add handler for power down req sgi irq (ade92a6)
    * add new state to identify cpu power down (5949701)
    * add wrapper to handle cpu power down req (3dd118c)
    * power down all cores on receiving cpu pwrdwn req (c3280df)
    * request cpu power down from reset (88ee081)
    * send SGI to mailbox driver (9a7f892)
    * **Versal**
        · enable errata management feature (d766f99)
        · extend platform address space sizes (663f024)
        · **Versal NET**
            · add bufferless IPI Support (511e4a4)
    * **ZynqMP**
        · remove unused pm_get_proc_by_node() (b03ba48)

- **Bootloader Images**
  – **BL32**
    * create an sp_min_setup function (a1255c7)
- **Services**
  – **FF-A**

    * update FF-A version to v1.2 (e830e4c)

  - **RME**
    * build TF-A with ENABLE_RME for Armv9.2 (7d5fc98)
    * pass console info via RMM-EL3 ifc (3290447)

  - **SPM**
    * **EL3 SPMC**
      - add support for FFA_CONSOLE_LOG (638a6f8)
      - add support for FFA_MEM_PERM_GET and SET ABIs (1f6b2b2)
      - add support to handle power mgmt calls for s-el0 sp (5917379)
      - add support to map S-EL0 SP device regions (727ab1c)
      - add support to map S-EL0 SP memory regions (83c3da7)
      - add support to setup S-EL0 context (48db2b0)
      - synchronize access to the s-el0 sp context (5ed8e25)
    * **SPMD**
      - add FFA_MSG_SEND_DIR_REQ2 (cc6047b)
      - add FFA_MSG_SEND_DIR_RESP2 (0651b7b)
      - initialize SCR_EL3.EEL2 bit at RESET (8815cda)
      - pass SMCCCv1.3 SVE hint to lower EL (c925867)

  - **DRTM**
    * add ACPI table region size to the DLME header (5dde96b)
    * add additional return codes (89f5c75)
    * for TPM features fw hash algorithm should be 16-bits (c86cfa3)
    * update DRTM version to 1.0 (9c36b90)
    * update references to DRTM beta0 (b94d590)
    * update return code if secondary PE is not off (bc9064a)

  - **ChromeOS**
    * add ChromeOS widevine SMC handler (b22e689)

- **Libraries**
  - **CPU Support**
    * add support for Poseidon V CPU (b77f55d)
    * support to update External LLC presence in Neoverse N3 (6fbc98b)
    * support to update External LLC presence in Neoverse V2 (6aa5d1b)

- **EL3 Runtime**
  - introduce UNDEF injection to lower EL (3c789bf)
- **FCONF**
  - support signing-key in root cert node (04ac0b3)
- **OP-TEE**
  - enable transfer list in opteed (0e8def9)
- **PSCI**
  - add psci_do_manage_extensions API (160e843)
- **GPT**
  - validate CRC of GPT partition entries (7a9e9f6)
- **SMCCC**
  - add vendor specific el3 id (be5b1e2)
  - add vendor-specific el3 service (de6b79d)
  - add version FID for PMF (42cbefc)
- **C Standard Library**
  - add printf support for space padding (0926d2d)
- **Locks**
  - add bitlock (222f885)
- **DICE Protection Environment (Experimental)**
  - add cert_id argument to dpe_derive_context() (6a415bd)
  - add client API for DICE Protection Environment (b03fe8c)
  - add DPE driver to measured boot (0ae9c63)
  - add QCBOR library as a dependency of DPE (c19977b)
  - add typedefs from the Open DICE repo (584052c)
- **Context Management**
  - report context memory usage (bfef8b9)
  - add documentation for context management library (4efd219)
- **Firmware Handoff**
  - add additional TE tags (a312bfb)
  - add support for RESET_TO_BL2 (f019c80)
  - add TE's for BL1 handoff interface (0646c9b)
  - add TL source files to BL1 (469b1d8)

    * enhance transfer list library (40fd755)

- **Drivers**
  - **Authentication**
    * add explicit entries for key OIDs (2b53106)
    * **mbedTLS**
      · update config for 3.6.0 (55aed7d)
  - **Console**
    * introduce EARLY_CONSOLE (ae770fe)
  - **FWU**
    * modify the check for getting the FWU bank's state (56724d0)
    * update the URL links for the FWU specification (e106a78)
  - **SCMI**
    * add scmi sensor support (e63819f)
  - **Arm**
    * **SMMU**
      · fix to perform INV_ALL before enabling GPC (70d849c)
      · separate out smmuv3_security_init from smmuv3_init (a23710b)
    * **MHU**
      · add MHUv3 doorbell driver (bc17476)
      · add MHUv3 wrapper APIs for RSS comm driver (4b4f850)
      · use compile flag to choose mhu version (996b3af)
    * **RSE**
      · add defines for 'type' range and use them in psa_call() (002b106)
      · adjust parameter packing to match TF-M changes (5abcc83)
  - **NXP**
    * add Linflex driver (306946b)
  - **ST**
    * **BSEC**
      · add driver for the new IP version BSEC3 (ae6542f)
      · use early traces (cf237f8)
    * **Clock**
      · add function to control MCU subsystem (77b4ca0)

    * **SDMMC2**
        · set FIFO size to 1024 on STM32MP25 (d5b4d5d)

- **Miscellaneous**
  - **AArch64**
    * add functions for TLBI RPALOS (8754cc5)
  - **DT Bindings**
    * introduce CCA CoT, rename TBBR (c4b35ce)
  - **FDTs**
    * **STM32MP2**
        · add board ID OTP in STM32MP257F-EV1 (88528f5)
        · add OTP nodes in STM32MP251 SoC DT file (c238a46)
  - **Security**
    * add support for SLS mitigation (538516f)
- **Documentation**
  - update maintainer list for neoverse_rd (2d7902d)
- **Build System**
  - check that .text section starts at page boundary (3d6edc3)
  - redirect stdin to nul during toolchain detection (b9014f8)
- **Tools**
  - **Memory Mapping Tool**
    * add RELA section display (a6462e0)

14.5.3 Resolved Issues

- **Architecture**
  - **Memory Tagging Extension2**
    * remove CTX_INCLUDE_MTE_REGS usage (30788a8)
    * use ATA bit with FEAT_MTE2 (ef0d0e5)
  - **Performance Monitors Extension (FEAT_PMUv3)**
    * fix breakage on ARMv7 CPUs with SP_min as BL32 (e6f8fc7)
  - **Statistical profiling Extension (FEAT_SPE)**
    * invoke spe_disable during power domain off/suspend (777f1f6)
- **Platforms**

- **Arm**
  - move console flush/switch in common function (6bdc856)
  - only expose `arm_bl2_dyn_cfg_init` to BL2 (3b48ca1)
  - **FVP**
    - added ranges for linux (b7491c7)
    - don't check MPIDRs with the power controller in BL1 (6d8546f)
    - permit enabling SME for SPD=spmd (0b0fd0b)
  - **FPGA**
    - halve number of PEs per core (70b9204)
  - **Neoverse-RD**
    - **SGI**
      - align to misra rule for braces (cacee06)
      - apply workarounds for N2 CPU erratum (7934b68)
      - increase BL31 carveout size (0737bd3)
      - reduce cper buffer carveout size (f10d3e4)
      - update spi_id max for sgi multichip platforms (89d8577)
    - **RD-N1-Edge**
      - update RD-N1-Edge's changelog title (d239ede)
    - **RD-N2**
      - populate TOS_CONFIG only when SPMC_AT_EL3 is enabled (10dcffe)
  - **TC**
    - correct interrupts (d2e44e7)
    - do not enable MPMM and Aux AMU counters always (fc42f84)
    - do not use r0 for HW_CONFIG (a5a966b)
    - enable FEAT_MTE2 (154eb0a)
    - guard PSA crypto headers under TF-M test-suite define (d2ce6aa)
    - increase BL2 maximum size limit (19258a5)
    - increase stack size when TRUSTED_BOARD_BOOT=0 (44ddee6)
    - missing device regions in spmc manifest (5e47112)
    - remove timer interrupt from G1S (9bf31a5)
- **Intel**
  - add HPS remapper to remap base address for SDM (b727664)

    * bl31 overwrite OCRAM configuration (cfbac59)
    * fix hardcoded mpu frequency ticks (150d2be)
    * read QSPI bank buffer data in bytes (2f17ac0)
    * revert back to use L4 clock (d0e400b)
    * revert sys counter to 400MHz (460692a)
    * temporarily workaround for Zephyr SMP (68820f6)
    * update DDR range checking for Agilex5 (f4aaa9f)
    * update fcs crypto init code to check for mode (b0f4478)
    * update fcs functions to check ddr range (e8a3454)
    * update from INFO to VERBOSE when print debug message (56c8d02)
    * update HPS bridges for Agilex5 SoC FPGA (2973054)
    * update individual return result for hps and fpga bridges (82752c4)
    * update nand driver to match GHRD design (a773f41)
    * update stream id to non-secure for SDM (8fbd307)
    * update system counter back to 400MHz (a72f86a)
  - **NXP**
    * **i.MX**
      - **i.MX 8M**
        - align 3200 MTps rate with U-Boot (060fe63)
        - fix CSU_SA_REG to work with all sa registers (c13016b)
        - handle 3734 in addition to 3733 and 3732 MTps rates (cb60a87)
        - **i.MX 8M Plus**
          - uncondtionally enable only the USB power domain (ae6ce19)
      - **i.MX 8ULP**
        - add sw workaround for csi/hotplug test hang (e1d5c3c)
        - fix suspend/resume issue when DBD owner is s400 only (68f132b)
        - increase the mmap region num (047d7d1)
  - **QEMU**
    * disable FEAT_SB (59bdb42)
    * increase max FIP size (f465ac2)
  - **Raspberry Pi**
    * consider MT when calculating core index from MPIDR (6744d07)

- **Renesas**
  - **R-Car**
    - fix implicit rule invocations in tools (e068a7c)
    - **R-Car 3**
      - change RAM protection configurations (e9afde1)
      - fix load address range check (4f7e0fa)
- **Rockchip**
  - add support for building with LTO enabled (e5e9ccd)
  - fix documentation in how build bl31 in AARCH64 (6611e81)
  - **RK3328**
    - apply ERRATA_A53_1530924 erratum (dd2c888)
- **ST**
  - **STM32MP2**
    - add missing include (cb0d6b5)
    - correct early/crash console init (4da462d)
- **Texas Instruments**
  - do not stop non-secure timer on world switch (d2e1f6a)
  - **K3**
    - increment while reading trail bytes (0bdaf5c)
- **Xilinx**
  - add console_flush() before shutdown (7ec53af)
  - add FIT image check in DT console (e2d9dfe)
  - add FIT image check in prepare_dtb (046e130)
  - check proc variable before use (652c1ab)
  - deprecate SiP service count query (6a80c20)
  - fix sending sgi to linux (427e46d)
  - follow MISRA-C standards for condition check (655e62a)
  - rename macros to align with ARM (7995319)
  - update correct return types (8eb6a1d)
  - **Versal**
    - initialize cntfrq_el0 register (f000744)
    - **Versal NET**

- setup counter frequency (07625d9)
- use arm common GIC handlers (b225926)
- **ZynqMP**
  - resolve null pointer dereferencing (20fa9fc)
- **Nuvoton**
  - gfx frame buffer memory corruption during secondary boot (ae2b4a5)
  - prevent changing clock frequency (fe8cc55)
- **Bootloader Images**
  - **BL1**
    - add missing `__RW_{START,END}__` symbols (d701b48)
    - add missing spinlock dependency (e40b563)
  - **BL2**
    - make BL2 SRAM footprint flexible (e0e03a8)
- **Services**
  - **FF-A**
    - add NS memory node to fvp_spmc_optee_sp manifest (92bba3e)
  - **RME**
    - **RMMD**
      - avoid TRP when external RMM is defined (57bc3c4)
      - fix bug, raised by coverity, when zeroing manifest struct (83a4e8e)
  - **SPM**
    - add device-regions used in tf-a-tests (45716e3)
    - not defining load-address in SP config (04e7f80)
    - reduce verbosity on passing tf-a-tests (29872eb)
    - silence warning in sp_mk_generator (6a3225e)
    - **EL3 SPMC**
      - add datastore linker script markers (ba33528)
      - fix dangling pointer in FFA_CONSOLE_LOG (83129bc)
    - **SPMD**
      - register group0 handler only if supported (fca5f0e)
      - skip NS EL1 context save & restore operations (2d960a1)
- **Libraries**

- **CPU Support**
  - workaround for Cortex-A520 erratum 2630792 (f03bfc3)
  - workaround for Cortex-A520 erratum 2858100 (34db353)
  - workaround for Cortex-A710 erratum 2778471 (c9508d6)
  - workaround for Cortex-A715 erratum 2331818 (53b3cd2)
  - workaround for Cortex-A715 erratum 2344187 (33c665a)
  - workaround for Cortex-A715 erratum 2413290 (15a0461)
  - workaround for Cortex-A715 erratum 2420947 (1f73247)
  - workaround for Cortex-A715 erratum 2429384 (262dc9f)
  - workaround for Cortex-A715 erratum 2561034 (6a6b282)
  - workaround for Cortex-A715 erratum 2728106 (10134e3)
  - workaround for Cortex-A720 erratum 2926083 (152f4cf)
  - workaround for Cortex-A720 erratum 2940794 (7385213)
  - workaround for Cortex-A78C erratum 2683027 (68cac6a)
  - workaround for Cortex-A78C erratum 2743232 (81d4094)
  - workaround for Cortex-X2 erratum 2778471 (b01a93d)
  - workaround for Cortex-X3 erratum 2266875 (a65c5ba)
  - workaround for Cortex-X3 erratum 2302506 (3f9df2c)
  - workaround for Cortex-X3 erratum 2372204 (7f69a40)
  - workaround for Cortex X3 erratum 2641945 (c1aa3fa)
  - workaround for Cortex X3 erratum 2743088 (f43e9f5)
  - workaround for Cortex-X3 erratum 2779509 (355ce0a)
  - workaround for Cortex-X4 erratum 2701112 (cc41b56)
  - workaround for Cortex-X4 erratum 2740089 (c833ca6)
  - workaround for Cortex-X4 erratum 2763018 (4731211)
  - workaround for Neoverse V1 erratum 2348377 (71ed917)
  - workaround for Neoverse V2 erratum 2618597 (c0f8ce5)
  - workaround for Neoverse V2 erratum 2662553 (912c409)
  - workaround for Neoverse V2 erratum 3099206 (8815cda)
  - add Cortex-A520 definitions (ae19093)
  - workaround for Cortex-A715 erratum 2413290 re-factored with ENABLE_SPE_FOR_NS=1 (bd2f7d3)

    * fix a defect in Cortex-A715 erratum 2561034 (57ab6d8)
    * add erratum 2701951 to Cortex-X3's list (106c428)
    * update status of Cortex-X3 erratum 2615812 (f589a2a)
    * fix incorrect AMU trap settings for N2 CPU (54b86d4)
    * correct variant name for default Poseidon CPU (61a2968)
    * check for SCU before accessing DSU (5b5562b)

  - **EL3 Runtime**
    * **Context Management**
      · add more feature registers to EL1 context mgmt (d6c76e6)
      · add more system registers to EL1 context mgmt (ed9bb82)
      · hide `cm_init_context_by_index` from BL1 (a6b3643)
      · remove ENABLE_FEAT_MTE usage (a796d5a)
      · save guarded control stack registers (6aae3ac)
      · update gic el2 sysregs save/restore mechanism (937d6fd)
      · couple el2 registers with dependent feature flags (d6af234)
      · move EL1 save/restore routines into C (59f8882)

  - **FCONF**
    * boot fails using ARM_ARCH_MINOR=8 (0c86a84)

  - **OP-TEE**
    * set interrupt handler before kernel boot (0ec69a5)

  - **PSCI**
    * fix parent_idx in psci_validate_state_coordination (412d92f)
    * mask the Last in Level nibble in StateId (0a9c244)

  - **GPT**
    * declare gpt_tlbi_by_pa_ll() (832e4ed)
    * unify logging messages (b99926e)
    * use DC CIGDPAPA when MTE2 is implemented (62d6465)

  - **C Standard Library**
    * add memcpy_s source file to libc_asm mk (99db13b)
    * memset inclusion to libc makefiles (84eb3ef)

  - **PSA**
    * fix static check failure (bc0ff02)

- **Context Management**
  - align the memory address of EL2 context registers (8c56a78)
- **Firmware Handoff**
  - correct representation of tag_id (d594ace)
- **Exception Handling Framework (EHF)**
  - restrict secure world FIQ routing model to SPM_MM (7671008)
- **SMCCC**
  - correctly find pmf version (62865b4)

- **Drivers**
  - **Measured Boot**
    - add missing image identifier string (a8a09e3)
  - **SCMI**
    - induce a delay in monitoring SCMI channel status (af1ac2d)
  - **Arm**
    - **GIC**
      - **GICv3**
        - **GIC-600**
          - workaround for Part 1 of GIC600 erratum 2384374 (24a4a0a)
      - **GICv2**
        - fix SGIR_NSATT bitshift (eef240c)
    - **MHU**
      - use MHUv2 if PLAT_MHU_VERSION undefined (c34dd06)
      - provide only the usable size of memory (5cd1084)
    - **RSE**
      - fix bound check during protocol selection (f754bd4)
  - **Renesas**
    - **R-Car3**
      - add integer overflow check (ef38fb1)
      - add integer overflow check (93b8952)
      - check "rcar_image_number" variable before use (b469880)
      - check for length underflow (9778b27)
      - check loaded NS image area (ae4860b)

- **USB**
  - add missing include (f84f21f)
- **Miscellaneous**
  - **TBBR**
    - move rotpk definitions out of arm_def.h (0f0fd49)
  - code coverage optimization fix (152ad11)
  - fix MISRA defects (c42d0d8)
  - static checks on spmc dts (c35299d)
- **Documentation**
  - revise the description of REGISTER_CRYPTO_LIB (5710229)
  - typo in the romlib design (3b57ae2)
- **Build System**
  - add forgotten BL_LDFLAGS to lto command line (49ba1df)
  - don't generate build-id (304ad94)
  - don't rely on that gcc-ar is in the same directory as gcc (7ef0b83)
  - enforce single partition for LTO build (31f80ef)
  - march handling with arch-features (7275ac2)
  - move comment for VERSION_PATCH (c25d1cc)
  - mute sp_mk_generator from build log (fbd32ac)
  - properly manage versions in .versionrc.js (7f74030)
  - wrap toolchain paths in double quotes (4731c00)
- **Tools**
  - **Certificate Creation Tool**
    - add guardrails around brainpool usage (c0c280d)
    - use a salt length equal to digest length for RSA-PSS (e639ad2)
  - **Memory Mapping Tool**
    - fix footprint free space calculation (9e72d01)
    - fix memory map dump when SEPARATE_CODE_AND_RODATA=0 (6dc8ee6)
  - **Marvell Tools**
    - include mbedtls/version.h before use (8eb4efe)

14.6 2.10.0 (2023-11-21)

14.6.1 ? BREAKING CHANGES

- **Architecture**
  - **Performance Monitors Extension (FEAT_PMUv3)**
    - This patch explicitly breaks the EL2 entry path. It is currently unsupported.

      See: convert FEAT_MTPMU to C and move to persistent register init (83a4dae)
- **Libraries**
  - **EL3 Runtime**
    - **Context Management**
      - Initialisation code for handoff from EL3 to NS-EL1 disabled by default. Platforms which do that need to enable this macro going forward

        See: introduce INIT_UNUSED_NS_EL2 macro (183329a)
- **Drivers**
  - **Authentication**
    - remove CryptoCell-712/713 support

      See: remove CryptoCell-712/713 support (b65dfe4)

14.6.2 New Features

- **Architecture**
  - **CPU feature / ID register handling in general**
    - add AArch32 PAN detection support (d156c52)
    - add memory retention bit define for CLUSTERPWRDN (278beb8)
    - deny AArch64-only features when building for AArch32 (733d112)
    - initialize HFG*_EL2 registers (4a530b4)
  - **Memory Tagging Extension**
    - adds feature detection for MTE_PERM (4d0b663)
  - **Performance Monitors Extension (FEAT_PMUv3)**
    - introduce pmuv3 lib/extensions folder (c73686a)
- **Platforms**
  - **Allwinner**
    - use reset through scpi for warm/soft reset (0cf5f08)

- **Arm**
  - add IO policy to use backup gpt header (3e6d245)
  - ecdsa p384/p256 full key support (b8ae689)
  - enable FHI PPI interrupt to report CPU errors (f1e4a28)
  - reuse SPM_MM specific defines for SPMC_AT_EL3 (5df1dcc)
  - save BL32 image base and size in entry point info (821b01f)
  - add memory map entry for CPER memory region (4dc91ac)
  - firmware first error handling support for base RAMs (5b77a0e)
  - update common platform RAS implementation (7f15131)
  - **FVP**
    - add mbedtls_asn1_get_len symbol in ROMlib (0605060)
    - add public key-OID information in RSS metadata structure (bfbb1cb)
    - add spmd logical partition (5cf311f)
    - allow configurable FVP Trusted SRAM size (41e56f4)
    - capture timestamps in bl stages (ed8f06d)
    - implement platform function to measure and publish Public Key (db55d23)
    - increase BL1 RW area for PSA_CRYPTO implementation (ce18938)
    - mock support for CCA NV ctr (02552d4)
    - new SiP call to set an interrupt pending (2032401)
    - spmd logical partition smc handler (a1a9a95)
  - **Juno**
    - add mbedtls_asn1_get_len symbol in ROMlib (ec8ba97)
  - **Morello**
    - add cpuidle support (4f7330d)
    - add support for I2S audio (6bcbe43)
    - add TF-A version string to NT_FW_CONFIG (f4e64d1)
    - fdts: add CoreSight DeviceTree bindings (3e6cfa7)
    - set NT_FW_CONFIG properties for MCC, PCC and SCP version (10fd85d)
  - **RD**
    - **RD-N2**
      - enable base element RAM RAS support on RD-N2 platform (0288632)
      - add defines needed for spmc-el3 (b4bed4b)

        - add plat hook for memory transaction (f99dcba)
        - enable Neoverse N2 CPU error handling support (e802748)
        - introduce accessor function to obtain datastore (f458934)
        - introduce platform handler for Group0 interrupt (c47d049)

    - **SGI**
        - remove RAS setup call from common code (0f5e8eb)
        - firmware first error handling for Neoverse N2 CPU (31d1e4f)
        - increase sp memmap size (7c33bca)

    - **TC**
        - define memory ranges for tc platform (9be6b16)
        - implement platform function to measure and publish Public Key (eee9fb0)
        - deprecate Arm TC1 FVP platform (6a2b11c)

- **Aspeed**
    - **AST2700**
        - add Aspeed AST2700 platform support (85f199b)

- **Intel**
    - add intel_rsu_update() to sip_svc_v2 (e3c3a48)
    - ccu driver for Agilex5 SoC FPGA (02df499)
    - clock manager support for Agilex5 SoC FPGA (1b1a3eb)
    - cold/warm reset and smp support for Agilex5 SoC FPGA (79626f4)
    - ddr driver for Agilex5 SoC FPGA (29461e4)
    - mailbox and SMC support for Agilex5 SoC FPGA (8e59b9f)
    - memory controller support for Agilex5 SoC FPGA (18adb4e)
    - mmc support for Agilex5 SoC FPGA (4a577da)
    - pinmux, peripheral and Handoff support for Agilex5 SoC FPGA (fcbb5cf)
    - platform enablement for Agilex5 SoC FPGA (7931d33)
    - power manager for Agilex5 SoC FPGA (a8bf898)
    - reset manager support for Agilex5 SoC FPGA (9b8d813)
    - restructure sys mgr for Agilex (6197dc9)
    - restructure sys mgr for S10/N5X (b653f3c)
    - sdmmc/nand/combo-phy/qspi driver for Agilex5 SoC FPGA (ddaf02d)
    - setup SEU ERR read interface for FP8 (91239f2)

    * system manager support for Agilex5 SoC FPGA (7618403)
    * uart support for Agilex5 SoC FPGA (34971f8)
    * vab support for Agilex5 SoC FPGA (4754925)

  - **MediaTek**
    * add APU bootup control smc call (94a9e62)
    * add APU watchdog timeout control (baa0d45)
    * **MT8188**
      · add apusys ao devapc setting (777e3b7)
      · add backup/restore function when power on/off (233d604)
      · add devapc setting of apusys rcx (5986ae5)
      · add DSB before udelay (b254b98)
      · add emi mpu protection for APU secure memory (176846a)
      · add EMI MPU support for SCP and DSP (013006f)
      · add support for SMC from OP-TEE (34d9d61)
      · enable apusys domain remap (b5900c9)
      · enable apusys mailbox mpu protect (ad7673a)
      · increase TZRAM_SIZE from 192KB to 256KB (aa1cb27)
      · modify APU DAPC permission (d06edab)
      · update return value in mtk_emi_mpu_sip_handler (d07eee2)
    * **MT8195**
      · increase TZRAM (4f79b67)

  - **NXP**
    * **i.MX**
      · add dummy 'plat_mboot_measure_key' function (b9bceef)
      · **i.MX 8M**
        · add more dram pll setting (8947404)
        · detect console base address during runtime (df730d9)
        · enable snvs privileged registers access (8d150c9)
        · move the gpc reg & macro to a separate header file (2a6ffa9)
        · **i.MX 8M Nano**
          · add workaround for errata ERR050362 (8562564)
      · **i.MX 9**

- **i.MX93**
  - add cpuidle and basic suspend support (422d30c)
  - add OPTEE support (27a0be7)
  - add reset & poweroff support (cf7ef4c)
  - add the basic support (2368d7b)
  - add the trdc driver (2935291)
  - allow SoC masters access to system TCM (3d3b769)
  - protect OPTEE memory to secure access only (f560f84)
  - update the ocram trdc config for did10 (eb76a24)

- **QEMU**
  - add sdei support for QEMU (cef76a7)
  - add "cortex-a710" cpu support (4734a62)
  - add "neoverse-n2" cpu support (408f9cb)
  - add "neoverse-v1" cpu support (6d8d7d2)
  - add "neoverse-v1" cpu support (214de62)
  - add A55 cpu support for virt (409c20c)
  - add dummy plat_mboot_measure_key() BL1 function (8e2fd6a)
  - add dummy plat_mboot_measure_key() function (f0f11ac)
  - implement firmware handoff on qemu (322af23)
  - **SBSA**
    - handle platform version (c681d02)
    - handle GIC base (1e67b1b)
    - handle GIC ITS address (4171e98)

- **QTI**
  - **MSM8916**
    - add port for MDM9607 (78aac78)
    - add port for MSM8909 (cf0a75f)
    - add port for MSM8939 (c28e96c)
    - add SP_MIN port for AArch32 (45b2bd0)
    - add Test Secure Payload (TSP) port (6b8f9e1)
    - allow selecting which UART to use (aad23f1)
    - clear CACHE_LOCK for MMU-500 r2p0+ (d9b0442)

        · initialize CCI-400 for multiple clusters (1240dc7)
        · power on L2 caches for secondary clusters (c822d26)

- **ST**
    - add RCC registers list (4cfbb84)
    - allow AARCH64 compilation for common code (dad7181)
    - introduce new platform STM32MP2 (35527fb)
    - support gcc as linker (7762531)
    - update STM32MP DT files (4c8e8ea)
    - **STM32MP1**
        - add FWU with boot from NOR-SPI (dfbadfd)
        - **STM32MP15**
            - disable OP-TEE shared memory (fb1d3bd)
    - **STM32MP2**
        - add console configuration (87a940e)
        - generate stm32 file (e5839ed)
- **Texas Instruments**
    - add TI-SCI query firmware capabilities command support (7ab7828)
    - query firmware for suspend capability (ce1008f)
    - remove extra core counts in cluster 2 and 3 (e986845)
- **Xilinx**
    - add support to get chipid (0563601)
    - clean macro names (bfd0626)
    - fix IPI calculation for Versal/NET (69a5bee)
    - move IPI related macros to plat_ipi.h (b2258ce)
    - remove crash console unused macros (473ada6)
    - setup local/remote id in header (068b0bc)
    - switch boot console to runtime (9c1c8f0)
    - sync macro names (04a4833)
    - used console also as crash console (3e6b96e)
    - **Versal**
        - add support for SMCC ARCH SOC ID (079c6e2)
        - add tsp support (7ff4d4f)

- ddr address reservation in dtb at runtime (56d1857)
- enable assertion (0375188)
- retrieval of console information from dtb (7c36fbc)
- **Versal NET**
  - add cluster check in handoff parameters (01c8c6a)
  - add support for SMCC ARCH SOC ID (1873e7f)
  - add the IPI CRC checksum macro support (ba56b01)
  - add tsp support (639b367)
  - ddr address reservation in dtb at runtime (46a08aa)
  - enable assertion (80cb4b1)
  - get the handoff params using IPI (a36ac40)
  - remove empty crash console setup (6a14246)
  - retrieval of console information from dtb (a467e81)

* **ZynqMP**
  - enable assertion (2243ba3)
  - remove pm_ioctl_set_sgmii_mode api (7414aaa)
  - retrieval of console information from dtb (3923462)

– **Nuvoton**
  * added support for npcm845x chip (edcece1)

- **Bootloader Images**
  - **BL2**
    * add gpt support (6ed98c4)
  - **BL31**
    * reuse SPM_MM specific defines for SPMC_AT_EL3 (f5e1bed)
  - **BL32**
    * print entry point before exiting SP_MIN (94e1be2)
- **Services**
  - **RME**
    * save PAuth context when RME is enabled (13cc1aa)
    * **RMMD**
      - enable SME for RMM (f92eb7e)
      - pass SMCCCv1.3 SVE hint bit to RMM (6788963)

- **RMM**
  - update RMI VERSION command as per EAC5 (ade6000)
- **SPM**
  - separate StMM SP specifics to add support for a S-EL0 SP (549bc04)
  - **EL3 SPMC**
    - add a flag to enable support to load SEL0 SP (801cd3c)
  - **SPMD**
    - add partition info get regs (0b850e9)
    - add spmd logical partitions (890b508)
    - el3 direct message API (66bdfd6)
    - get logical partitions info (95f7f6d)
- **ERRATA ABI**
  - add support for Cortex-X3 (9c16521)

- **Libraries**
  - **CPU Support**
    - add a concise way to implement AArch64 errata (3f4c1e1)
    - add a way to automatically report errata (4f748cc)
    - add errata framework helpers (445f7b5)
    - add more errata framework helpers (94a75ad)
    - add support for Gelas CPU (02586e0)
    - add support for hermes cpu (a00e907)
    - add support for Nevis CPU (5497958)
    - add support for Travis CPU (a0594ad)
    - conform DSU errata to errata framework PCS (ee6d04d)
    - make revision procedure call optional (4d22b0e)
    - wrappers to propagate AArch32 errata info (34c51f3)
  - **EL3 Runtime**
    - modify vector entry paths (d04c04a)
    - **RAS**
      - reuse SPM_MM specific defines for SPMC_AT_EL3 (6e92a82)
      - use FEAT_IESB for error synchronization (6597fcf)
  - **Translation Tables**

        - detect 4KB and 16KB page support when FEAT_LPA2 is present (bff074d)

    - **C Standard Library**
        - add %X to printf/snprintf (483edc2)
        - implement memcpy_s in lib (f328bff)

    - **PSA**
        - interface with RSS for retrieving ROTPK (50316e2)

    - **Firmware Handoff**
        - introduce firmware handoff library (3ba2c15)
        - port BL31-BL33 interface to fw handoff framework (94c90ac)

- **Drivers**
    - **Authentication**
        - add CCA NV ctr to CCA CoT (e3b1cc0)
        - add explicit entries for key OIDs (0cffcdd)
        - create a zero-OID for Subject Public Key (9505d03)
        - ecdsa p384 key support (557f7d8)
        - measure and publicise the Public Key (9eaa5a0)
        - **mbedTLS**
            - update to 3.4.1 (e686cdb)
            - add deprecation notice (267c106)
        - **mbedTLS-PSA**
            - initialise mbedtls psa crypto (4eaaaa1)
            - introduce PSA_CRYPTO build option (5782b89)
            - mbedTLS PSA Crypto with ECDSA (255ce97)
            - register an ad-hoc PSA crypto driver (38f8936)
            - use PSA crypto API during hash calculation (484b586)
            - use PSA crypto API during signature verification (eaa62e8)
            - use PSA crypto API for hash verification (2ed061c)

    - **Measured Boot**
        - introduce platform function to measure and publish Public Key (2971bad)

    - **GUID Partition Tables Support**
        - add interface to init gpt (f08460d)
        - add support to use backup GPT header (ad2dd65)

- **Arm**
  - **Ethos-N**
    - update npu error handling (4796d2d)
  - **RSS**
    - set the signer-ID in the RSS metadata (60861a0)
- **ST**
  - **Clock**
    - allow aarch64 compilation of STGEN functions (b1718c6)
    - stub fdt_get_rcc_secure_state (19c3808)
  - **UART**
    - add AARCH64 stm32_console driver (c6d070c)

- **Miscellaneous**
  - **AArch64**
    - add stack debug information to assembly routines (f832885)
  - **DT Bindings**
    - add the STM32MP2 clock and reset bindings (3ccb708)
  - **FDTs**
    - **Morello**
      - add thermal framework (0b22160)
    - **STM32MP2**
      - add stm32mp257f-ev1 board (9aa5371)
      - introduce stm32mp25 pinctrl files (2c62cc4)
      - introduce stm32mp25 SoCs family (0dc283d)
  - **TBBR**
    - add image id for backup GPT (1051606)
    - update PK_DER_LEN for ECDSA P-384 keys (c1ec23d)

- **Documentation**
  - introduce STM32MP2 doc (ee5076f)
  - save BL32 image base and size in entry point info (31dcf23)
  - add a threat model for TF-A with Arm CCA (4463541)
  - cover threats inherent to receiving data over UART (348446a)
  - add a section for experimental build options (4885600)

- **Build System**
  - include plat header in fdt build (e03dcc8)
  - manage patch version in Makefile (055ebec)
  - march option selection (7794d6c)
  - pass CCA NV ctr option to cert_create (0f19b7a)
  - .gitignore to include memory tools (82257de)
  - allow gcc linker on Aarch32 platforms (cfe6767)
  - bump certifi to version 2023.7.22 (6cbf432)
  - convert tabs and ifdef comparisons (72f027c)
  - convert tabs to spaces (1ca73b4)
  - disable ENABLE_FEAT_MPAM for Aarch32 (a07b459)
  - include Cortex-A78AE cpu file for FVP (b996db1)
  - pass parameters through response files (430be43)
  - remove duplicated include order (c189adb)
  - remove handling of mandatory options (1ca902a)
- **Tools**
  - **Firmware Image Package Tool**
    - add ability to build statically (4d4fec2)
  - **Secure Partition Tool**
    - generate `ARM_BL2_SP_LIST_DTS` file from `sp_layout.json` (20629b3)
  - **Certificate Creation Tool**
    - add new option for CCA NV ctr (60753a6)
    - add pkcs11 engine support (616b3ce)
    - ecdsa p384 key support (c512c89)
  - **Memory Mapping Tool**
    - add tabular memory use data (d9d5eb1)
    - add topological memory view (cc60aba)

14.6.3 Resolved Issues

- **Architecture**
  - **CPU feature / ID register handling in general**
    - move nested virtualization support to optionals (8b2048c)
  - **Memory Partitioning and Monitoring (MPAM) Extension (FEAT_MPAM)**
    - refine MPAM initialization and enablement process (edebefb)
  - **Performance Monitors Extension (FEAT_PMUv3)**
    - make MDCR_EL3.MTPME=1 out of reset (33815eb)
- **Platforms**
  - register PLAT_SP_PRI only if not already registered (bf01999)
  - **Arm**
    - add Event Log area behind Trustzone Controller (d836df7)
    - correct the SPMC_AT_EL3 condition (a0ef1c0)
    - fix GIC macros for GICv4.1 support (f1df8f1)
    - add RAS_FFH_SUPPORT check for RAS EHF priority (1c01284)
    - do not program DSU CLUSTERPWRDN register (3209b35)
    - **FPGA**
      - enable CPU features required for ARMv9.2 cores (b321c24)
    - **FVP**
      - adjust BL2 maximum size as per total SRAM size (965aace)
      - adjust BL31 maximum size as per total SRAM size (24e224b)
      - conditionally increase XLAT and MMAP table entries (03cf4e9)
      - extract core id from mpidr for pwrc operations (70bc744)
      - increase maximum MMAP and XLAT entries count (12fe591)
      - increase the maximum size of Event Log (f1dfaa4)
      - resolve broken workaround reference (bcb3ea9)
      - update pwr_domain_suspend (f51d277)
      - update system suspend in OS-initiated mode (e0ef05b)
    - **Morello**
      - configure platform specific secure SPIs (80f8769)
    - **N1SDP**

      - configure platform specific secure SPIs (7b0c95a)
      - fix spi_ids range for n1sdp multichip boot (31f60a9)
    - **SGI**
      - update PLAT_SP_PRI macro definition (6f689a5)
    - **TC**
      - Correct return type (b0542b5)
      - rename macro to match PSA spec (1fc20d7)
    - **Corstone-1000**
      - add cpu_helpers.S to platform.mk (cb27274)
      - modify boot device dependencies (3ff5fc2)
      - removing the signature area (5856a91)
  - **Aspeed**
    - **AST2700**
      - add device mapping for coherent memory (cef2e92)
  - **Broadcom**
    - fix misspelled header inclusion guard (a9779c1)
  - **Cadence**
    - update console flush uart driver (e27bebb)
  - **Intel**
    - fix ncore ccu snoop dvm enable bug (106aa54)
    - resolved coverity checking (1af7bf7)
    - update boot scratch cold register to use cold 8 (655af4f)
    - update checking for memcpy and memset (c418064)
  - **MediaTek**
    - support saving/restoring GICR registers (f73466e)
  - **NVIDIA**
    - **Tegra**
      - return correct error code for plat_core_pos_by_mpidr (6bd79b1)
  - **NXP**
    - **i.MX**
      - **i.MX 8M**
        - make IMX_BOOT_UART_BASE autodetection option more obvious (101f070)

- map BL32 memory only if SPD_opteed or SPD_trusty is enabled (4827613)

- **QEMU**
  - fix 32-bit builds with stack protector (e57ca89)
  - **SBSA**
    - align FIP base to BL1 size (408cde8)
- **QTI**
  - **SC7280**
    - update pwr_domain_suspend (a43be0f)
    - update system suspend in OS-initiated mode (0a9270a)
- **Renesas**
  - **R-Car**
    - add mandatory fields in 'reserved-memory' node (f945498)
    - **R-Car 3**
      - fix CPG register code comment (69c371b)
      - update Draak and Eagle board IDs (281edfe)
- **ST**
  - allow crypto lib compilation in aarch64 (76e4fab)
  - enable RTC clock before accessing nv counter (77ce6a5)
  - flush UART at the end of uart_read() (a9cb7d0)
  - properly check LOADADDR (9f72f5e)
  - reduce MMC block_buffer (a2500ab)
  - setting default KEY_SIZE (6f3ca8a)
  - update comment on encryption key (5c506c7)
  - update dt_get_ddr_size() type (2a4abe0)
  - **STM32MP1**
    - add void entry in plat_def_toc_entries (8214ecd)
    - properly check PSCI functions return (241f874)
    - use the BSEC nodes compatible for stm32mp13 (2171bd9)
- **Texas Instruments**
  - align static device region addresses to reduce MMU table count (53a868f)
  - fix TISCI API changes during refactor (d7a7135)
  - release lock in all TI-SCI xfer return paths (e92375e)

    * remove check for zero value in BL31 boot args (44edd3b)

- **Xilinx**
    * add headers to resolve compile time issue (744d60a)
    * dcache flush for dtb region (93ed138)
    * don't reserve 1 more byte (c3b69bf)
    * dynamic mmap region for dtb (7ca7fb1)
    * remove clock_setrate and clock_getrate api (e5955d7)
    * remove console error message (f9820f2)
    * update dtb when dtb address and tf-a ddr flow is used (fdf8f92)
    * **DCC (Debug Communication Channel)**
        - add dcc console unregister function (0936abe)
        - enable DCC also for crash console (c6d9186)
    * **Versal**
        - add missing irq mapping for wakeup src (06b9c4c)
        - fix BLXX memory limits for user defined values (f123b91)
        - make pmc ipi channel as secure (96eaafa)
        - type cast addresses to fix integer overflow (bfe82cf)
        - use correct macro name for ocm base address (56afab7)
        - **Versal NET**
            - add redundant call to avoid glitches (cebb7cc)
            - change flag to increase security (e8efb65)
            - correct device node indexes (66b5620)
            - don't clear pending interrupts (fb73ea6)
            - fix BLXX memory limits for user defined values (a80da38)
            - make pmc ipi channel as secure (2c65b79)
            - use correct macro name for uart baudrate (e2ef1df)
    * **ZynqMP**
        - do not export apu_ipi (237c5a7)
        - fix BLXX memory limits for user defined values (8ce2fbf)
        - fix prepare_dtb() memory description (3efee73)
        - fix sdei arm_validate_ns_entrypoint() (3b3c70a)
        - handling of type el3 interrrupts (e8d61f7)

- make zynqmp_devices structure smaller (7e3e799)
- remove unused headers (6288636)
- resolve runtime error in TSP (81ad3b1)
- type cast addresses to fix overflow issue (9129163)
- validate clock_id to avoid OOB variable access (abc79c2)

- **Nuvoton**
  - fix typo in platform.mk (c7efb78)

- **Bootloader Images**
  - **BL2**
    - bl2 start address for RESET_TO_BL2+ENABLE_PIE (d478ac1)
  - **BL31**
    - resolve runtime console garbage in next stage (889e3d1)
  - **BL32**
    - always include arm_arch_svc in SP_MIN (cd0786c)
    - avoid clearing argument registers in RESET_TO_SP_MIN case (56055e8)
    - **TSP**
      - fix destination ID in direct request (ed23d27)
      - flush uart console (ae074b3)
- **Services**
  - **RME**
    - **RMMD**
      - enable sme using sme_enable_per_world (c0e16d3)
  - **SPM**
    - **EL3 SPM**
      - fix LSP direct message response (c040621)
      - improve direct messaging validation (48fe24c)
    - **EL3 SPMC**
      - avoid descriptor size calc overflow (27c0242)
      - correctly account for emad_offset (0c2583c)
      - fix incorrect CASSERT (1dd79f9)
      - only call spmc_shm_check_obj() on complete objects (d781959)
      - prevent total_page_count overflow (2d4da8e)

  - remove experimental flag (630a06c)
  - use uint64_t for 64-bit type (43318e4)
  - use version-dependent minimum descriptor length (52d8d50)
  - validate descriptor headers (56c052d)
  - validate memory address alignment (327b5b8)
  - validate shmem descriptor alignment (dd94372)

  - **SPMD**
    - coverity scan issues (b04343f)
    - fix FFA_VERSION forwarding (76d53ee)
    - perform G0 interrupt acknowledge and deactivation (6c91fc4)
    - relax use of EHF with SPMC at S-EL2 (bb6d0a1)

- **ERRATA ABI**
  - added Neoverse N2 to Errata ABI list (7e030b3)
  - fix the rev-var for Cortex-A710 (5c8fcc0)
  - update the Cortex-A76 errata ABI struct (92d5b50)
  - update the Cortex-A78C errata ABI struct (7f2caec)
  - update the neoverse-N1 errata ABI struct (56747a5)
  - update the Neoverse-N2 errata ABI struct (80af87e)

- **Libraries**
  - **CPU Support**
    - assert invalid cpu_ops obtained (3f721c6)
    - check for SME presence in Gelas (0bbd432)
    - fix minor issue seen with a9 cpu (af70470)
    - fix the rev-var for Cortex-A710 (2bf7939)
    - fix the rev-var of Cortex-X2 (8ae66d6)
    - fix the rev-var of Neoverse-V1 (ab2b56d)
    - flush L2 cache for Cortex-A7/12/15/17 (c5c160c)
    - integer suffix macro definition (1a56ed4)
    - reduce generic_errata_report()'s size (f43e09a)
    - revert erroneous use of override_vector_table macro in Cortex-A73 (9a0c812)
    - update the fix for Cortex-A78AE erratum 1941500 (67a2ad1)
    - update the rev-var for Cortex-A78AE (c814619)

    * workaround for Cortex-A510 erratum 2080326 (6e86475)
    * workaround for Cortex-A710 erratum 2742423 (d7bc2cb)
    * workaround for Cortex-X2 erratum 2742423 (fe06e11)
    * workaround for Cortex-X3 erratum 2070301 (2454316)
    * workaround for Cortex-X3 erratum 2742421 (5b0e443)
    * workaround for Neoverse N2 erratum 2009478 (74bfe31)
    * workaround for Neoverse N2 erratum 2340933 (68085ad)
    * workaround for Neoverse N2 erratum 2346952 (6cb8be1)
    * workaround for Neoverse N2 erratum 2743014 (eb44035)
    * workaround for Neoverse N2 erratum 2779511 (12d2806)
    * workaround for Neoverse V2 erratum 2331132 (8852fb5)
    * workaround for Neoverse V2 erratum 2719105 (b011402)
    * workaround for Neoverse V2 erratum 2743011 (58dd153)
    * workaround for Neoverse V2 erratum 2779510 (ff34264)
    * workaround for Neoverse V2 erratum 2801372 (40c81ed)
  - **EL3 Runtime**
    * leverage generic interrupt controller helpers (07f867b)
    * restrict lower el EA handlers in FFH mode (6d22b08)
    * **Context Management**
      · make ICC_SRE_EL2 fixup generic to all worlds (5e8cc72)
      · set MDCR_EL3.{NSPBE, STE} explicitly (99506fa)
    * **RAS**
      · remove RAS_FFH_SUPPORT and introduce FFH_SUPPORT (f87e54f)
      · restrict ENABLE_FEAT_RAS to have only two states (970a4a8)
  - **PSCI**
    * add optional pwr_domain_validate_suspend to plat_psci_ops_t (d348861)
  - **SMCCC**
    * ensure that mpidr passed through SMC is valid (e60c184)
    * pass SMCCCv1.3 SVE hint to internal flags (b2d8517)
  - **Translation Tables**
    * fix defects on the xlat library reported by coverity scan (2974ad8)
    * set MAX_PHYS_ADDR to total mapped physical region (1a38aaf)

- **Drivers**
  - **Authentication**
    - allow hashes of different lengths (22a5354)
    - don't overwrite pk with converted pk when rotpk is hash (1046b41)
  - **Measured Boot**
    - don't strip last non-0 char (b85bcb8)
  - **MMC**
    - initialises response buffer with zeros (b1a2c51)
  - **MTD**
    - **NAND**
      - reset the SLC NAND (f4d765a)
      - **SPI NAND**
        - add Quad Enable management (da7a33c)
  - **SCMI**
    - add parameter for plat_scmi_clock_rates_array (ca9d6ed)
  - **UFS**
    - performs unsigned shift for doorbell (e47d8a5)
    - set data segment length (9d6786c)
  - **Arm**
    - **GIC**
      - **GICv3**
        - map generic interrupt type to GICv3 group (632e5ff)
        - move invocation of gicv3_get_multichip_base function (36704d0)
        - **GIC-600**
          - fix gic600 maximum SPI ID (69ed7dc)
  - **Renesas**
    - **R-Car3**
      - update DDR setting (138ddcb)
  - **ST**
    - **Clock**
      - disabling CKPER clock is not functional on stm32mp13 (1bbcb58)
    - **Crypto**

        · do not read RNG data if it's not ready (53092a7)
        · use GENMASK_32 to define PKA registers masks (379d77b)
      * **DDR**
        · express memory size with size_t type (b4e1e8f)
      * **UART**
        · allow 64 bit compilation (6fef0f6)
        · correctly check UART enabled in flush fonction (a527380)
        · skip console flush if UART is disabled (b156d7b)

- **Miscellaneous**
  - **AArch32**
    * disable workaround discovery on aarch32 for now (d1f2748)
  - **FDTs**
    * **STM32MP1**
      · move /omit-if-no-ref/ to overlay files (f351f91)
      · **STM32MP13**
        · correct the BSEC nodes compatible (85c2ea8)
        · cosmetic fixes in PLL nodes (8b82663)
  - **SDEI**
    * ensure that interrupt ID is valid (a7eff34)
  - **TBBR**
    * guard defines under MBEDTLS_CONFIG_FILE (81c2e15)
    * unrecognised 'tos-fw-key-cert' option (f1cb5bd)
- **Documentation**
  - match boot-order size to implementation (fd1479d)
  - add missing line in the fiptool command for stm32mp1 (d526d00)
  - fix build errors for latexpdf (443d6ea)
  - remove out-dated information about CI review comments (74306b2)
  - replace deprecated urls under tfa/docs (5fdf198)
  - update maintainers list (9766f41)
  - updated certain Neoverse N2 erratum status in docs (d6d34b3)
  - use rsvg-convert as the conversion backend (c365476)
- **Tools**

- **Firmware Image Package Tool**
    - move juno plat_fiptool.mk (570a230)
- **Certificate Creation Tool**
    - fix key loading logic (bb3b0c0)
    - key: Avoid having a temporary value for pkey in key_load (ea6f845)
- **Memory Mapping Tool**
    - reintroduce support for GNU map files (d0e3053)

14.7 2.9.0 (2023-05-16)

14.7.1 ? BREAKING CHANGES

- **Libraries**
  - **EL3 Runtime**
    - **RAS**
      - The previous RAS_EXTENSION is now deprecated. The equivalent functionality can be achieved by the following 2 options:
        - ENABLE_FEAT_RAS
        - RAS_FFH_SUPPORT

        See: replace RAS_EXTENSION with FEAT_RAS (9202d51)
- **Drivers**
  - **Authentication**
    - unify REGISTER_CRYPTO_LIB

      See: unify REGISTER_CRYPTO_LIB (dee99f1)
  - **Arm**
    - **Ethos-N**
      - The Linux Kernel NPU driver can no longer directly configure and boot the NPU in a TZMP1 build. The API version has therefore been given a major version bump with this change.

        See: add protected NPU firmware setup (6dcf3e7)
      - Building the FIP when TZMP1 support is enabled in the NPU driver now requires a parameter to specify the NPU firmware file.

        See: load NPU firmware at BL2 (33bcaed)
- **Build System**

- BL2_AT_EL3 renamed to RESET_TO_BL2 across the repository.

  See: distinguish BL2 as TF-A entry point and BL2 running at EL3 (42d4d3b)

- check boolean flags are not empty

  See: check boolean flags are not empty (1369fb8)

- All input and output linker section names have been prefixed with the period character, e.g. `cpu_ops` -> `.cpu_ops`.

  See: always prefix section names with `.` (da04341)

- The `EXTRA_LINKERFILE` build system variable has been replaced with the `<IMAGE>_LINKER_SCRIPT_SOURCES` variable. See the commit message for more information.

  See: permit multiple linker scripts (a6ff006)

- The `LINKERFILE`, `BL_LINKERFILE` and `<IMAGE_LINKERFILE>` build system variables have been renamed. See the commit message for more information.

  See: clarify linker script generation (8227493)

14.7.2 Resolved Issues

- **Architecture**
  - **CPU feature / ID register handling in general**
    - context-switch: move FGT availability check to callers (de8c489)
    - make stub enable functions "static inline" (d7f3ed3)
    - resolve build errors due to compiler optimization (e8f0dd5)
  - **Memory Partitioning and Monitoring (MPAM) Extension (FEAT_MPAM)**
    - feat_detect: support major/minor (1f8be7f)
    - remove unwanted param for "endfunc" macro (0e0bd25)
    - run-time checks for mpam save/restore routines (ed80440)
  - **Pointer Authentication Extension**
    - make pauth_helpers linking generic (90ce8b8)
  - **Performance Monitors Extension (FEAT_PMUv3)**
    - switch FVP PMUv3 SPIs to PPI (d7c455d)
    - unconditionally save PMCR_EL0 (1d6d680)
  - **Scalable Matrix Extension (FEAT_SME, FEAT_SME2)**
    - disable SME for SPD=spmd (2fd2fce)
  - **Statistical profiling Extension (FEAT_SPE)**

    * drop SPE EL2 context switch code (16e3ddb)

- **Platforms**
  - **Allwinner**
    * check RSB availability in DT on H6 (658b315)
  - **Arm**
    * arm_rotpk_header undefined reference (95302e4)
    * **A5DS**
      - add default value for ARM_DISABLE_TRUSTED_WDOG (115ab63)
    * **CSS**
      - fix invalid redistributor poweroff (60719e4)
    * **FPGA**
      - include missing header file (b7253a1)
    * **FVP**
      - correct ehf priority for SPM_MM (fb2fd55)
      - incorrect UUID name in FVP tb_fw_config (7f2bf23)
      - unconditionally include lib/psa headers (72db458)
      - work around BL31 progbits exceeded (138221c)
      - work around DRTM_SUPPORT BL31 progbits exceeded (7762e5d)
    * **Morello**
      - add platform-specific power domain functions (02a5bcb)
    * **N1SDP**
      - add platform-specific power domain functions (5bdafc4)
    * **RD**
      - **RD-N1 Edge**
        - change variable type to fix gcc sign conversion error (3a3e0e5)
    * **TC**
      - increase TC_TZC_DRAM1_SIZE (7e3f6a8)
      - change the FIP offset to 8 KiB boundary (d07b8aa)
      - change the properties of optee reserved memory (2fff46c)
      - enable dynamic feature detection of FEAT_SVE for NormalWorld (67265f2)
      - enable the execution of both platform tests (657b90e)
      - only suspend booting after running plat tests (9b26655)

  · unify TC ROM start addresses (f9e11c7)
  · update the name of mbedtls config header (d5fc899)

- **Broadcom**
  - add braces around bodies of conditionals (9f58bfb)
- **Intel**
  - add mailbox error return status for FCS_DECRYPTION (76ed322)
  - agilex bitstream pre-authenticate (4b3d323)
  - fix Agilex and N5X clock manager to main PLL C0 (5f06bff)
  - fix fcs_client crashed when increased param size (c42402c)
  - fix pinmux handoff bug on Agilex (e6c0389)
  - fix print out ERROR when encounter SEU_Err (1a0bf6e)
  - fix sp_timer0 is not disabled in firewall on Agilex (8de7167)
  - fix the pointer of block memory to fill in and bytes being set (afe9fcc)
  - flash dcache before mmio read (731622f)
  - mailbox store QSPI ref clk in scratch reg (7f9e9e4)
  - missing NCORE CCU snoop filter fix in BL2 (b34a48c)
  - remove checking on TEMP and VOLT checking for HWMON (68ac5fe)
  - update boot scratch to indicate to Uboot is PSCI ON (7f7a16a)
- **NVIDIA**
  - **Tegra**
    - append major revision to the chip_id value (33c4766)
    - remove dependency on CPU registers to get boot parameters (0b9f05f)
    - **Tegra 210**
      - support legacy SMC_ID 0xC2FEFE00 (40a4e2d)
- **NXP**
  - **i.MX**
    - **i.MX 8M**
      - add ddr4 dvfs sw workaround for ERR050712 (e00fe11)
      - backup mr12/14 value from lpddr4 chip (a2655f4)
      - correct the rank info get fro mstr (5277c09)
      - fix coverity out of bound access issue (0331b1c)
      - fix the current fsp init (25c4323)

        - fix the dfiphymaster setting after dvfs (ad0cbbf)
        - fix the dram retention random hang on some imx8mq Rev2.0 (4bf5019)
        - fix the rank to rank space issue (3330084)
        - **i.MX 8Q**
          - fix compilation with gcc >= 12.x (e75a3b6)
    - **Layerscape**
      - fix errata a008850 (c45791b)
      - fix nv_storage assert checking (5d599b7)
      - unlock write access SMMU_CBn_ACTLR (0ca1d8f)
      - **LX2**
        - init global data before using it (50aa0ea)
      - **LS1046A**
        - 4 keys secureboot failure resolved (c0c157a)
  - **QEMU**
    - enable dynamic feature detection of FEAT_SVE for NormalWorld (fc259b6)
    - **SBSA**
      - enable FGT (c598692)
      - enable SVE and SME (9bff7ce)
  - **QTI**
    - **MSM8916**
      - add timeout for crash console TX flush (7e002c8)
      - drop unneeded initialization of CNTACR (d833af3)
      - flush dcache after writing msm8916_entry_point (01ba69c)
      - print \r before \n on UART console (3fb7e40)
  - **Raspberry Pi**
    - **Raspberry Pi 3**
      - initialize SD card host controller (bd96d53)
  - **Renesas**
    - align incompatible function pointers (90c4b3b)
  - **Rockchip**
    - use semicolon instead of comma (8557d49)
  - **ST**

  * add U suffix for unsigned numbers (9c1aa12)
  * explicitly check operators precedence (56048fe)
  * include utils.h to solve compilation error (377846b)
  * make metadata_block_spec static (d1d8a9b)
  * rework secure-status check in fdt_get_status() (0ebaf22)
  * use Boolean type for tests (45d2d49)
  * use indices when counting GPIOs in DT (e7d7544)
  * **STM32MP1**
    - add const for strings in stm32mp_get_soc_name() (d7f5bed)
    - add missing platform.h include (6e55f9e)
    - always define PKA algos flags (e0e2d64)
    - remove boolean check on PLAT_TBBR_IMG_DEF (231a0ad)
    - rework DWL buffer cache invalidation (127ed00)

- **Texas Instruments**
  * do not take system power reference in bl31_platform_setup() (9977948)
  * fix typo in boot authentication message name (81f525e)

- **Xilinx**
  * fix misra defects (964e559)
  * handle CRC failure in IPI (5e92be5)
  * handle CRC failure in IPI callback (6173d91)
  * initialize values to device enum members (5c62d59)
  * remove asserts around arg0/arg1 (8be2044)
  * remove unnecessary condition (c984123)
  * remove unused mailbox macros (15f49cb)
  * resolve integer handling issue (4e46db4)
  * use lib/smccc.h macros instead of trusty spd (0ee07d7)
  * **Versal**
    - check smc_fid 23:16 bits (4a50363)
    - fix incorrect regbase for PMC IPI (c4185d5)
    - initialize the variable with value 0 in pm code (cd73d62)
    - print proper atf handoff source (0fe002c)
    - replace FPD_MAINCCI* macros (245d30e)

- sync location based on IPI_ID macros (92a43bd)
- **Versal NET**
  - fix irq for IPI0 (95bbfbc)
  - clear power down bit during wakeup (5f0f7e4)
  - clear power down interrupt status before enable (2d056db)
  - correct aff level for cpu off (6ada9dc)
  - disable wakeup interrupt during client wakeup (e663f09)
  - enable wake interrupt during client suspend (39fffe5)
  - fix setting power down state (1f79bdf)
  - populate gic v3 rdist data statically (355dc3d)
  - resolve misra 10.6 warnings (8c23775)
  - resolve misra rule 20.7 warnings (21d1966)
  - use spin_lock instead of bakery_lock (0b3a2cf)

* **ZynqMP**
  - add bitmask for get_op_char API (ad4b667)
  - check return status of pm_get_api_version (c92ad36)
  - check smc_fid 23:16 bits (09b342a)
  - conditional reservation of memory in DTB (c52a142)
  - enable A53 workaround(errata 1530924) (d8133d7)
  - fix bl31_zynqmp_setup.c coding style (26ef5c2)
  - fix DT reserved allocated size (2c03915)
  - fix xck24 silicon ID (f156590)
  - initialize uint32 with value 0U in pm code (e65584a)
  - move EM SMC range to SIP range (acbae39)
  - panic w/o handoff structure in !JTAG (fbe4dbe)
  - remove redundant api_version check (d0b58c8)
  - remove unused PLAT_NUM_POWER_DOMAINS (72c3124)
  - separate EM from PM SMCs (a911396)
  - update MAX_XLAT_TABLES for DDR memory range (12446ce)
  - update the conflicting EEMI API IDs (bcc1348)
  - with DEBUG=1 move bl31 to DDR range (2537f07)

- **Bootloader Images**

- **BL31**
  - avoid clearing of argument registers in RESET_TO_BL31 case (3e14df6)
- **BL32**
  - **TSP**
    - loop / crash if mmap of region fails (8c353e0)
    - use verbose for power logs (3354915)

- **Services**
  - **RME**
    - update sample platform attestation token (19c1dce)
    - **TRP**
      - preserve RMI SMC X4 when not used as return (b96253d)
    - **RMMD**
      - add missing padding to RMM Boot Manifest and initialize it (dc0ca64)
  - **SPM**
    - **EL3 SPMC**
      - fix coverity scan warnings (1543d17)
      - improve bound check for descriptor (def7590)
      - report execution state in partition info get (62cd8f3)
    - **SPMD**
      - fix build error with spmd (fd51b21)

- **Libraries**
  - **CPU Support**
    - do not put RAS check before using esb (9ec2ca2)
    - use hint instruction for "tsb csync" (7a181b7)
    - workaround for Cortex-A510 erratum 2684597 (aea4ccf)
    - workaround for Cortex-A710 erratum 2282622 (89d85ad)
    - workaround for Cortex-A710 erratum 2768515 (b87b02c)
    - workaround for Cortex-A78 erratum 2742426 (a63332c)
    - workaround for Cortex-A78 erratum 2772019 (b10afcc)
    - workaround for Cortex-A78 erratum 2779479 (7d1700c)
    - workaround for Cortex-A78C erratum 1827430 (672eb21)
    - workaround for Cortex-A78C erratum 1827440 (b01a59e)

    * workaround for Cortex-A78C erratum 2772121 (00230e3)
    * workaround for Cortex-A78C erratum 2779484 (66bf3ba)
    * workaround for Cortex-X2 erratum 2282622 (f9c6301)
    * workaround for Cortex-X2 erratum 2768515 (1cfde82)
    * workaround for Cortex-X3 erratum 2615812 (c7e698c)
    * workaround for Neoverse N2 erratum 2743089 (1ee7c82)
    * workaround for Neoverse V1 errata 2743233 (f1c3eae)
    * workaround for Neoverse V1 errata 2779461 (2757da0)
    * workaround for Neoverse V1 erratum 2743093 (31747f0)
    * workaround platforms non-arm interconnect (ab062f0)

- **EL3 Runtime**
    * allow SErrors when executing in EL3 (1cbe42a)
    * do not save scr_el3 during EL3 entry (e61713b)
    * restore SPSR/ELR/SCR after esb (ff1d2ef)
    * **RAS**
        · do not put RAS check before esb macro (7d5036b)

- **FCONF**
    * fix FCONF_ARM_IO_UUID_NUMBER value (e208f32)
    * make struct fconf_populator static (40e740d)

- **OP-TEE**
    * address late comments and fix bad rc (8d7c80f)
    * return UUID for image loading service (85ab882)

- **PSCI**
    * do not panic on illegal MPIDR (8a6d0d2)
    * potential array overflow with cpu on (6632741)
    * remove unreachable switch/case blocks (ad27f4b)
    * tighten psci_power_down_wfi behaviour (695a48b)

- **GPT**
    * fix compilation error for gpt_rme.c (a0d5147)

- **SMCCC**
    * check smc_fid [23:17] bits (f8a3579)

- **C Standard Library**

        * properly define SCHAR_MIN (06c01b0)
        * remove __putchar alias (28dc825)
    - **Context Management**
        * enable SCXTNUM access (01cf14d)
- **Drivers**
    - **Authentication**
        * avoid out-of-bounds read in auth_nvctr() (abb8f93)
        * forbid junk after extensions (fd37982)
        * only accept v3 X.509 certificates (e9e4a2a)
        * properly validate X.509 extensions (f5c5185)
        * reject invalid padding in digests (f47547b)
        * reject junk after certificates (ca34dbc)
        * reject padding after BIT STRING in signatures (a8c8c5e)
        * require at least one extension to be present (72460f5)
        * require bit strings to have no unused bits (8816dbb)
        * use NULL instead of 0 for pointer check (654b65b)
        * **mbedTLS**
            · fix mbedtls coverity issues (a9edc32)
    - **Console**
        * correct scopes for console symbols (03bd481)
        * fix crash on spin_unlock with cache disabled (5fb6946)
    - **I/O**
        * compare function pointers with NULL (06d223c)
    - **MMC**
        * align part config type (53cbc94)
        * do not modify r_data in mmc_send_cmd() (bf78a65)
        * explicitly check operators precedence (14cda51)
        * remove redundant reset_to_idle call (bc0a738)
    - **GUID Partition Tables Support**
        * add missing curly braces (1290662)
        * add U suffix for unsigned numbers (d1c6c49)
    - **SCMI**

    * change function prototype to fix gcc error (f0f2c90)
    * fix compilation error in scmi base (7c38934)
  - **UFS**
    * device present (DP) field is set to '1' (83103d1)
    * flush the entire PRDT (83ef869)
    * only allow using one slot (56db7b8)
    * poll UCRDY for all commands (6e57b2f)
    * set the PRDT length field properly (20fdbcf)
  - **Arm**
    * **Ethos-N**
      · add workaround for erratum 2838783 (5a89947)
    * **GIC**
      · wrap cache enabled assert under plat_can_cmo (78fbb0e)
      · **GICv3**
        · fixed bug in the initialization of GICv3 SGIs/(E)PPIs interrupt priorities (5d68e89)
        · restore scr_el3 after changing it (1d0d5e4)
        · workaround for NVIDIA erratum T241-FABRIC-4 (a02a45d)
    * **RSS**
      · do not consider MHU_ERR_ALREADY_INIT as error (55a7aa9)
      · fix msg deserialization bugs in comms (dda0528)
      · remove null-terminator from RSS metadata (85a14bc)
  - **NXP**
    * fix fspi coverity issue (5199b3b)
    * fix sd secure boot failure (236ca56)
    * fix tzc380 memory regions config (07d8e34)
    * use semicolon instead of comma (50b8ea1)
    * **NXP Crypto**
      · fix coverity issue (e492299)
      · fix secure boot assert inclusion (334badb)
    * **DDR**
      · add checking return value (e83812f)
      · apply Max CDD values for warm boot (00bb8c3)

- fix coverity issue (2d541cb)
- fix underrun coverity issue (87612ea)
- use CDDWW for write to read delay (fa01056)

- **ST**
  - **Clock**
    - avoid arithmetics on pointers (4198fa1)
    - give the size for parent_mp13 and dividers_mp13 tables (ee21709)
    - remove useless switch (69a2e32)
    - use Boolean type for tests (c3ae7da)
  - **Crypto**
    - move flag control into source code (6a187a0)
    - remove platdata functions (6b3ca0a)
    - set get_plain_pk_from_asn1() static (70a422b)
  - **GPIO**
    - define shift as uint32_t (5d942ff)
  - **SDMMC2**
    - check transfer size before filling register (029f81e)
  - **ST PMIC**
    - define pmic_regs table size (3cebeec)
    - enclose macro parameter in parentheses (be7195d)
  - **Regulator**
    - enclose macro parameters in parentheses (91af163)
    - explicitly check operators precedence (68083e7)
    - rework for_each_*rdev macros (6a3ffb5)
    - use Boolean type for tests (9a00daf)
  - **USB**
    - replace redundant checks with asserts (02af589)

- **Style**
  - correct some typos (1b491ee)
- **Miscellaneous**
  - **AArch64**
    - allow build with ARM_ARCH_MINOR=4 (78f56ee)

- **FDT Wrappers**
  - use correct prototypes (e0c56fd)
- **FDTs**
  - **STM32MP1**
    - **STM32MP15**
      - use /omit-if-no-ref/ for spi and i2c (d480df2)
      - use interrupts-extended for i2c2 (600c8f7)
- **PIE**
  - pass `-fpie` to the preprocessor as well (966660e)
- **UUID**
  - add missing `#include` directives (12562af)
- add missing click dependency (ff12683)
- add parenthesis for tests in MIN, MAX and CLAMP macros (8406db1)
- increase BL32 limit (c2a7612)
- remove old-style declarations (f4b8470)
- remove useless "return" at void functions (af4d8c6)
- unify fallthrough annotations (e138400)

- **Documentation**
  - add a build.tools.python entry (4052d95)
  - add few missed links for Security Advisories (43f3a9c)
  - add plantuml as a dependency (65982a9)
  - add readthedocs configuration file (8a84776)
  - deprecate plat_convert_pk() in v2.9 (e0f58c7)
  - make required compiler version == rather than >= (415195c)
  - python version must be string (3aa919e)
  - specify python version to 3.10 (a7773c5)
- **Build System**
  - add a default value for INVERTED_MEMMAP (4d32f91)
  - allow lower address access with gcc-12 (dea23e2)
  - allow warnings when using lld (ebac692)
  - partially fix qemu aarch32 build (c68736d)
- **Tools**

- **NXP Tools**
  - fix coverity issue (4fa0f09)
- **Secure Partition Tool**
  - add dependency to SP image (4daeaf3)
- **Certificate Creation Tool**
  - change WARN to VERBOSE (76a85cf)

- **Dependencies**
  - add missing aeabi_memset.S (bdedee5)

14.7.3 New Features

- **Architecture**
  - **Extended Translation Control Register (FEAT_TCR2).**
    - add FEAT_TCR2 to the changelog (a366640)
    - support FEAT_TCR2 (d333160)
  - **CPU feature / ID register handling in general**
    - enable FEAT_SME for FEAT_STATE_CHECKED (45007ac)
    - enable FEAT_SVE for FEAT_STATE_CHECKED (2b0bc4e)
    - extend check_feature() to deal with min/max (a4cccb4)
  - **Guarded Control Stack (FEAT_GCS)**
    - support guarded control stack (688ab57)
  - **Support for the `HCRX_EL2` register (FEAT_HCX)**
    - initialize HCRX_EL2 to its default value (ddb615b)
  - **Scalable Matrix Extension (FEAT_SME, FEAT_SME2)**
    - enable SME2 functionality for NS world (03d3c0d)
- **Platforms**
  - **Allwinner**
    - add extra CPU control registers (b15e2cd)
    - add function to detect H616 die variant (fbde260)
    - add support for Allwinner T507 SoC (018c1d8)
  - **Arm**
    - add ARM_ROTPK_LOCATION variant full key (5f89928)
    - carveout DRAM1 area for Event Log (6b2e961)

- **FVP**
  - add Event Log maximum size property in DT (1cf3e2f)
  - copy the Event Log to TZC secured DRAM area (191aa5d)
  - define ns memory in the SPMC manifest (7f28179)
  - emulate trapped RNDR (1ae7552)
  - enable errata management interface (d3bed15)
  - enable FEAT_FGT by default (15107da)
  - enable FEAT_HCX by default (2e12418)
  - enable support for PSCI OS-initiated mode (e75cc24)
  - increase BL1_RW and BL2 size (dbb9c1f)
  - introduce PLATFORM_TEST_EA_FFH config (fe38cc6)
  - introduce PLATFORM_TEST_RAS_FFH config (5602ce1)
  - update device tree with load addresses of TOS_FW config (1779762)
- **Juno**
  - support ARM_IO_IN_DTB option for Juno (2fad320)
- **Morello**
  - add GPU DT node (cd94c3d)
  - add support for HW_CONFIG (be79071)
  - implement methods to retrieve soc-id information (cc266bc)
- **RD**
  - **RD-N2**
    - add platform id value for rdn2 variant 3 (028c619)
- **TC**
  - enable MPAM functionality of L3 DSU cache (b45ec8c)
  - add delegated attest and measurement tests (25dd217)
  - allow secure watchdog timer to trigger periodically (28b2d86)
  - use smmu 700 (ed80eab)

– **Intel**
- extending to support SMMU in FCS (4687021)
- fix bridge disable and reset (9ce8251)
- implement timer init divider via CPU frequency for N5X (02a9d70)
- setup FPGA interface for Agilex (3905f57)

- **MediaTek**
  - add APU init flow (5243091)
  - add new features of LPM (917abdd)
  - add SiP service for OP-TEE (621eaab)
  - add SMC handler for EMI MPU (c842cc0)
  - add SPM's SSPM notifier (c234ad1)
  - **MT8188**
    - add apu power on/off control (8e38b92)
    - add MT8188 SPM debug logs (f85b34b)
    - add MT8188 SPM support (45d5075)
    - add SPM feature support (f299efb)
    - add the register definitions accessed by SPM (1a64689)
    - enable SPM and LPM (380f64b)
    - keep infra and peri on when system suspend (e56a939)
    - update INFRA IOMMU enable flow (98415e1)
  - **MT8195**
    - add support for SMC from OP-TEE (ccc61e1)
- **NVIDIA**
  - **Tegra**
    - implement 'pwr_domain_off_early' handler (96d07af)
- **NXP**
  - **i.MX**
    - **i.MX 8M**
      - add more dram pll setting (4234b90)
      - fix the ddr4 dvfs random hang on imx8m (093888c)
      - update the ddr4 dvfs flow to include ddr3l support (0e39488)
      - use non-fast wakeup stop mode for system suspend (ef4e5f0)
      - **i.MX 8Q**
        - add anamix pll override setting for DSM mode (387a1df)
        - add BL31 PIE support (8cfa94b)
        - add the dram retention support for imx8mq (dd108c3)
        - add version for B2 (99475c5)

- add workaround code for ERR11171 on imx8mq (88a2646)
- always set up console (36be108)
- correct the slot ack setting for STOP mode (724ac3e)
- enable dram dvfs support on imx8mq (8962bdd)
- make IMX_BOOT_UART_BASE configurable via build parameter (202737e)
- remove empty bl31_plat_runtime_setup (7698dba)

- **i.MX 8**
  - add support for debug uart on lpuart1 (8406447)

* **Layerscape**
  - **LX2**
    - enable OCRAM ECC (e8faff3)
    - support more variants (c07f5e9)

- **QEMU**
  * add "neoverse-n1" cpu support (226f4c8)
  * add A76/N1 cpu support for virt (6b66693)
  * combine TF-A artefacts into ROM file (63bb905)
  * increase max cpus per cluster to 16 (73a7aca)
  * increase size of bl2 (db2bf3a)
  * make coherent memory section optional (af994ae)
  * support el3 spmc (302f053)
  * support pointer authentication (cffc956)
  * support s-el2 spmc (36802e2)
  * update abi between spmd and spmc (25ae7ad)
- **QTI**
  * **SC7280**
    - add support for PSCI_OS_INIT_MODE (e528bbe)
  * **MSM8916**
    - expose more timer frames (1781bf1)
- **ST**
  * mandate dtc version 1.4.7 (38ac8bb)
  * **STM32MP1**
    - add mbedtls-3.3 support config (c9498c8)

- **Texas Instruments**
  - add PSCI system_off support (0bdef26)
  - add sub and patch version number support (852378f)
  - disable L2 dataless UniqueClean evictions (10d5cf1)
  - do not handle EAs in EL3 (2fcd408)
  - set L2 cache data ram latency on A72 cores to 4 cycles (aee2f33)
  - set L2 cache ECC and and parity on A72 cores (81858a3)
  - set snoop-delayed exclusive handling on A72 cores (5668db7)
  - synchronize access to secure proxy threads (312eec3)
- **Xilinx**
  - add device node indexes (407eb6f)
  - sync copyright format (2774965)
  - **Versal**
    - replace irq array with switch case (0ec6c31)
    - switch to xlat_v2 (0e9f54e)
    - **Versal NET**
      - add jtag dcc support (30e8bc3)
      - add support for set wakeup source (c38d90f)
      - add support for uart1 console (2f1b4c5)
  - **ZynqMP**
    - add hooks for custom runtime setup (88a8938)
    - add hooks for mmap and early setup (7013400)
    - add SMCCC_ARCH_SOC_ID support (8f9ba3f)
    - add support for custom sip service (496d708)
    - build pm code as library (3af2ee9)
    - bump up version of query_data API (aaf5ce7)
    - make stack size configurable (5753665)

- **Services**
  - **RME**
    - read DRAM information from FVP DTB (8268590)
    - set DRAM information in Boot Manifest platform data (a97bfa5)
    - **RMM**

        - add support for the 2nd DRAM bank (346cfe2)

  - **SPM**
    - **EL3 SPMC**
      - make platform logical partition optional (555677f)
    - **SPMD**
      - add support for FFA_EL3_INTR_HANDLE_32 ABI (6671b3d)
      - copy tos_fw_config in secure region (0cea2ae)
      - fail safe if SPM fails to initialize (0d33649)
      - introduce FFA_PARTITION_INFO_GET_REGS (eaaf517)
      - introduce platform handler for Group0 interrupt (f0b64e5)
      - map SPMC manifest region as EL3_PAS (8c829a9)
      - register handler for group0 interrupt from NWd (a1e0e87)
  - **ERRATA_ABI**
    - errata management firmware interface (ffea384)
- **Libraries**
  - **CPU Support**
    - add support for blackhawk cpu (6578343)
    - add support for chaberton cpu (516a52f)
  - **EL3 Runtime**
    - handle traps for IMPDEF registers accesses (0ed3be6)
    - introduce system register trap handler (ccd81f1)
  - **FCONF**
    - rename 'ns-load-address' to 'secondary-load-address' (05e5503)
  - **OP-TEE**
    - add device tree for coreboot table (f4bbf43)
    - add loading OP-TEE image via an SMC (05c69cf)
  - **PSCI**
    - add support for OS-initiated mode (606b743)
    - add support for PSCI_SET_SUSPEND_MODE (b88a441)
    - introduce 'pwr_domain_off_early' hook (6cf4ae9)
    - update PSCI_FEATURES (9a70e69)
  - **C Standard Library**

        * add %c to printf/snprintf (44d9706)
        * add support for fallthrough statement (023f1be)
    - **PSA**
        * add read_measurement API (6d0525a)
        * interface with RSS for NV counters (8374508)
- **Drivers**
    - **Authentication**
        * compare platform and certificate ROTPK for authentication (f1e693a)
        * **mbedTLS**
            · add support for mbedtls-3.3 (51e0615)
    - **UFS**
        * adds timeout and error handling (2c5bce3)
    - **Arm**
        * **Ethos-N**
            · add check for NPU in SiP setup (a2cdbb1)
            · add event and aux control support (7820777)
            · add multiple asset allocators (8a921e3)
            · add NPU firmware validation (313b776)
            · add NPU sleeping SMC call (2a2e3e8)
            · add NPU support in fiptool (c91b08c)
            · add protected NPU firmware setup (6dcf3e7)
            · add protected NPU TZMP1 regions (d77c11e)
            · add reserved memory address support (a19a024)
            · add reset type to reset SMC calls (fa37d30)
            · add separate RO and RW NSAIDs (986c4e9)
            · add SMC call to get FW properties (e9812dd)
            · add stream extends and attr support (e64abe7)
            · add support for NPU to cert_create (f309607)
            · add support to set up NSAID (70a296e)
            · load NPU firmware at BL2 (33bcaed)
        * **GIC**
            · **GICv3**

          - enlarge the range for intr_num of structure interrupt_prop_t (d5eee8f)

      - **RSS**

        - add TC platform UUIDs for RSS images (6ef63af)

      - **SBSA**

        - helper api for refreshing watchdog timer (e8166d3)

- **Miscellaneous**

  - **AArch64**

    - make ID system register reads non-volatile (c2fb8ef)

  - **FDTs**

    - **STM32MP1**

      - use /omit-if-no-ref/ for pins nodes (0aae96c)

      - **STM32MP15**

        - add support for prtt1x board family (3812ceb)

  - **PIE/POR**

    - support permission indirection and overlay (062b6c6)

- **Documentation**

  - allow verbose build (f771a34)

- **Build System**

  - add support for new binutils versions (1f49db5)

  - allow additional CFLAGS for library build (5a65fcd)

  - **Git Hooks**

    - add pre-commit hook (cf9346c)

  - add support for poetry (793f72c)

- **Tools**

  - **Firmware Image Package Tool**

    - handle FIP in a disk partition (06e69f7)

- **Dependencies**

  - **Compiler runtime libraries**

    - update source files (658ce7a)

14.8 2.8.0 (2022-11-15)

14.8.1 ? BREAKING CHANGES

- **Drivers**
  - **Arm**
    - **Ethos-N**
      - add support for SMMU streams

        See: add support for SMMU streams (b139f1c)

14.8.2 New Features

- **Architecture**
  - pass SMCCCv1.3 SVE hint bit to dispatchers (0fe7b9f)
  - **Branch Record Buffer Extension (FEAT_BRBE)**
    - add brbe under feature detection mechanism (1298f2f)
  - **Confidential Compute Architecture (CCA)**
    - introduce new "cca" chain of trust (56b741d)
  - **Pointer Authentication Extension**
    - add/modify helpers to support QARMA3 (9ff5f75)
  - **Trapping support for RNDR/RNDRRS (FEAT_RNG_TRAP)**
    - add EL3 support for FEAT_RNG_TRAP (ff86e0b)
  - **Scalable Matrix Extension (FEAT_SME)**
    - fall back to SVE if SME is not there (26a3351)
  - **Scalable Vector Extension (FEAT_SVE)**
    - support full SVE vector length (bebcf27)
  - **Trace Buffer Extension (FEAT_TRBE)**
    - add trbe under feature detection mechanism (47c681b)
- **Platforms**
  - **Arm**
    - add support for cca CoT (f242379)
    - forbid running RME-enlightened BL31 from DRAM (1164a59)
    - provide some swd rotpk files (98662a7)
    - retrieve the right ROTPK for cca (50b4497)

* **CSS**
  - add interrupt handler for reboot request (f1fe144)
  - add per-cpu power down support for warm reset (158ed58)
* **FVP**
  - add example manifest for TSP (3cf080e)
  - add crypto support in BL31 (c9bd1ba)
  - add plat API to set and get the DRTM error (586f60c)
  - add plat API to validate that passed region is non-secure (d5f225d)
  - add platform hooks for DRTM DMA protection (d72c486)
  - build delegated attestation in BL31 (0271edd)
  - dts: drop 32-bit .dts files (b920330)
  - fdts: update rtsm_ve DT files from the Linux kernel (2716bd3)
  - increase BL31's stack size for DRTM support (44df105)
  - increase MAX_XLAT_TABLES entries for DRTM support (8a8dace)
  - support building RSS comms driver (29e6fc5)
* **RD**
  - **RD-N2**
    - add a new 'isolated-cpu-list' property (afa4157)
    - add SPI ID ranges for RD-N2 multichip platform (9f0835e)
    - enable extended SPI support (108488f)
* **SGI**
  - increase memory reserved for bl31 image (a62cc91)
  - read isolated cpu mpid list from sds (4243ef4)
  - add page table translation entry for secure uart (2a7e080)
  - bump bl1 rw size (94df8da)
  - configure SRAM and BL31 size for sgi platform (8fd820f)
  - deviate from arm css common uart related definitions (173674a)
  - enable css implementation of warm reset (18884c0)
  - remove override for `ARM_BL31_IN_DRAM` build-option (a371327)
  - route TF-A logs via secure uart (0601083)
* **TC**
  - add MHU addresses for AP-RSS comms on TC2 (6299c3a)

      - add RSS-AP message size macro (445130b)
      - add RTC PL031 device tree node (a816de5)
      - enable RSS backend based measured boot (6cb5d32)
      - increase maximum BL1/BL2/BL31 sizes (e6c1316)
      - introduce TC2 platform (eebd2c3)
      - move start address for BL1 to 0x1000 (9335c28)

- **HiSilicon**
  - **HiKey960**
    - add a FF-A logical partition (25a357f)
    - add memory sharing hooks for SPMC_AT_EL3 (5f905a2)
    - add plat-defines for SPMC_AT_EL3 (feebd4c)
    - add SP manifest for SPMC_AT_EL3 (6971642)
    - define a datastore for SPMC_AT_EL3 (e618c62)
    - increase secure workspace to 64MB (e0eea33)
    - read serial number from UFS (c371b83)
    - upgrade to xlat_tables_v2 (6cfc807)
- **MediaTek**
  - add more flexibility of mtk_pm.c (6ca2046)
  - add more options for build helper (5b95e43)
  - add smcc call for MSDC (4dbe24c)
  - extend SiP vendor subscription events (99d30b7)
  - implement generic platform port (394b920)
  - introduce mtk init framework (52035de)
  - move dp drivers to common folder (d150b62)
  - move lpm drivers back to common (cd7890d)
  - move mtk_cirq.c drivers to cirq folder (cc76896)
  - support coreboot BL31 loading (ef988ae)
  - **MT8186**
    - add EMI MPU support for SCP and DSP (3d4b6f9)
  - **MT8188**
    - add armv8.2 support (45711e4)
    - add audio support (c70f567)

- add cpu_pm driver (4fe7e6a)
- add DCM driver (bc9410e)
- add DFD control in SiP service (7079a94)
- add display port control in SiP service (a4e5023)
- add EMI MPU basic drivers (8454f0d)
- add IOMMU enable control in SiP service (be45724)
- add LPM driver support (f604e4e)
- add MCUSYS support (4cc1ff7)
- add pinctrl support (ec4cfb9)
- add pmic and pwrap support (e9310c3)
- add reset and poweroff functions (a72b9e7)
- add RTC support (af5d8e0)
- add support for PTP3 (44a1051)
- apply ERRATA for CA-78 (abb995a)
- enable MTK_PUBEVENT_ENABLE (0b1186a)
- initialize GIC (cfb0516)
- initialize platform for MediaTek MT8188 (de310e1)
- initialize systimer (215869c)

- **NXP**
  - **i.MX**
    - **i.MX 8M**
      - add dram retention flow for imx8m family (c71793c)
      - add support for high assurance boot (720e7b6)
      - add the anamix pll override setting (66d399e)
      - add the ddr frequency change support for imx8m family (9c336f6)
      - add the PU power domain support on imx8mm/mn (44dea54)
      - keep pu domains in default state during boot stage (9d3249d)
      - make psci common code pie compatible (5d2d332)
      - **i.MX 8M Nano**
        - add BL31 PIE support (62d37a4)
        - add hab and map required memory blocks (b5f06d3)
        - enable dram retention suuport on imx8mn (2003fa9)

        - **i.MX 8M Mini**
          - add BL31 PIE support (a8e6a2c)
          - add hab and map required memory blocks (5941f37)
          - enable dram retention suuport on imx8mm (b7abf48)
        - **i.MX 8M Plus**
          - add BL31 PIE support (7a443fe)
          - add hab and map required memory blocks (62a93aa)
        - **i.MX 8Q**
          - add 100us delay after USB OTG SRC bit 0 clear (66345b8)
    - **Layerscape**
      - **LS1043A**
        - **LS1043ARDB**
          - update ddr configure for ls1043ardb-pd (18af644)
- **QEMU**
  - increase size of bl31 (0e6977e)
- **QTI**
  - fix to support cpu errata (6cc743c)
  - updated soc version for sc7180 and sc7280 (39fdd3d)
- **Socionext**
  - **Synquacer**
    - add BL2 support (48ab390)
    - add FWU Multi Bank Update support (a193825)
    - add TBBR support (19aaeea)
- **ST**
  - add trace for early console (00606df)
  - enable MMC_FLAG_SD_CMD6 for SD-cards (53d5b8f)
  - properly manage early console (5223d88)
  - search pinctrl node by compatible (b14d3e2)
  - **STM32MP1**
    - add a check on TRUSTED_BOARD_BOOT with secure chip (54007c3)
    - add a stm32mp crypto library (ad3e46a)
    - add define for external scratch buffer for nand devices (9ee2510)

        - add early console in SP_min (14a0704)
        - add plat_report_*_abort functions (0423868)
        - add RNG initialization in BL2 for STM32MP13 (2742374)
        - add the decryption support (cd79116)
        - add the platform specific build for tools (461d631)
        - add the TRUSTED_BOARD_BOOT support (beb625f)
        - allow to override MTD base offset (e0bbc19)
        - configure the serial boot load address (4b2f23e)
        - extend STM32MP_EMMC_BOOT support to FIP format (95e4908)
        - manage second NAND OTP on STM32MP13 (d3434dc)
        - manage STM32MP13 rev.Y (a3f97f6)
        - optionally use paged OP-TEE (c4dbcb8)
        - remove unused function from boot API (f30034a)
        - retrieve FIP partition by type UUID (1dab28f)
        - save boot auth status and partition info (ab2b325)
        - update ROM code API for header v2 management (89c0774)
        - **STM32MP13**
            - change BL33 memory mapping (10f6dc7)
        - **STM32MP15**
            - manage OP-TEE shared memory (722ca35)
  - **Texas Instruments**
    - **K3**
        - add support for J784S4 SoCs (4a566b2)
  - **Xilinx**
    - **Versal**
        - add infrastructure to handle multiple interrupts (e497421)
        - get the handoff params using IPI (205c7ad)
        - resolve the misra 10.1 warnings (b86e1aa)
        - update macro name to generic and move to common place (f99306d)
        - **Versal NET**
            - add support for QEMU COSIM platform (6a079ef)
            - add documentation for Versal NET SoC (4efdc48)

- add SMP support for Versal NET (8529c76)
- add support for IPI (0bf622d)
- add support for platform management (0654ab7)
- add support for Xilinx Versal NET platform (1d333e6)

* **ZynqMP**
  - optimization on pinctrl_functions (314f9f7)
  - add support for ProvenCore (358aa6b)
  - add support for xck24 silicon (86869f9)
  - protect eFuses from non-secure access (d0b7286)
  - resolve the misra 10.1 warnings (bfd7c88)

- **Bootloader Images**
  - add interface to query TF-A semantic ver (dddf428)
  - **BL32**
    * **TSP**
      - add FF-A support to the TSP (4a8bfdb)
      - add ffa_helpers to enable more FF-A functionality (e9b1f30)
      - enable test cases for EL3 SPMC (15ca1ee)
      - increase stack size for tsp (5b7bd2a)
- **Services**
  - add a SPD for ProvenCore (b0980e5)
  - **RME**
    * **RMMD**
      - add support for RMM Boot interface (8c980a4)
      - add support to create a boot manifest (1d0ca40)
  - **SPM**
    * add tpm event log node to spmc manifest (054f0fe)
    * **SPMD**
      - avoid spoofing in FF-A direct request (5519f07)
  - **DRTM**
    * add a few DRTM DMA protection APIs (2b13a98)
    * add DRTM parameters structure version check (c503ded)
    * add Event Log driver support for DRTM (4081426)

    * add PCR entries for DRTM (ff1e42e)
    * add platform functions for DRTM (2a1cdee)
    * add remediation driver support in DRTM (1436e37)
    * add standard DRTM service (e62748e)
    * check drtm arguments during dynamic launch (40e1fad)
    * ensure that no SDEI event registered during dynamic launch (b1392f4)
    * ensure that passed region lies within Non-Secure region of DRAM (764aa95)
    * flush dcache before DLME launch (67471e7)
    * introduce drtm dynamic launch function (bd6cc0b)
    * invalidate icache before DLME launch (2c26597)
    * prepare DLME data for DLME launch (d42119c)
    * prepare EL state during dynamic launch (d1747e1)
    * retrieve DRTM features (e9467af)
    * take DRTM components measurements before DLME launch (2090e55)
    * update drtm setup function (d54792b)

- **Libraries**
  - **CPU Support**
    * add library support for Hunter ELP (8c87bec)
    * add a64fx cpu to tf-a (74ec90e)
    * make cache ops conditional (04c7303)
    * remove plat_can_cmo check for aarch32 (92f8be8)
    * update doc and check for plat_can_cmo (a2e0123)
  - **OP-TEE**
    * check paged_image_info (c0a11cd)
  - **PSCI**
    * add a helper function to ensure that non-boot PEs are offline (ce14a12)
  - **C Standard Library**
    * introduce __maybe_unused (351f9cd)
  - **PSA**
    * add delegated attestation partition API (4b09ffe)
    * remove initial attestation partition API (420deb5)
- **Drivers**

- **Authentication**
  - allow to verify PublicKey with platform format PK (40f9f64)
  - enable MBEDTLS_CHECK_RETURN_WARNING (a4e485d)
  - **Crypto**
    - update crypto module for DRTM support (e43caf3)
  - **mbedTLS**
    - update mbedTLS driver for DRTM support (8b65390)
- **I/O**
  - **MTD**
    - add platform function to allow using external buffer (f29c070)
- **MMC**
  - get boot partition size (f462c12)
  - manage SD Switch Function for high speed mode (e5b267b)
- **MTD**
  - add platform function to allow using external buffer (f29c070)
- **GUID Partition Tables Support**
  - allow to find partition by type UUID (564f5d4)
- **SCMI**
  - send powerdown request to online secondary cpus (14a2892)
  - set warm reboot entry point (5cf9cc1)
- **Arm**
  - **Ethos-N**
    - add support for SMMU streams (b139f1c)
  - **GIC**
    - add APIs to raise NS and S-EL1 SGIs (dcb31ff)
    - **GICv3**
      - validate multichip data for GIC-700 (a78b3b3)
  - **RSS**
    - add new comms protocols (3125901)
- **ST**
  - **Crypto**
    - add AES decrypt/auth by SAES IP (4bb4e83)

        - add ECDSA signature check with PKA (b0fbc02)
        - add STM32 RNG driver (af8dee2)
        - remove BL32 HASH driver usage (6b5fc19)
        - update HASH for new hardware version used in STM32MP13 (68039f2)
      - **SDMMC2**
        - define FIFO size (b46f74d)
        - make reset property optional (8324b16)
        - manage CMD6 (3deebd4)
      - **UART**
        - add initialization with the device tree (d99998f)
        - manage STM32MP_RECONFIGURE_CONSOLE (ea69dcd)
- **Miscellaneous**
  - **Debug**
    - add AARCH32 CP15 fault registers (bb22891)
    - add helpers for aborts on AARCH32 (6dc5979)
  - **FDTs**
    - **STM32MP1**
      - add CoT and fuse references for authentication (928fa66)
      - change pin-controller to pinctrl (44fea93)
      - **STM32MP13**
        - use STM32MP_DDR_S_SIZE in fw-config (936f29f)
      - **STM32MP15**
        - add Avenger96 board with STM32MP157A DHCOR SoM (51e2230)
        - add support for STM32MP157C based DHCOM SoM on PDK2 board (eef485a)
  - **SDEI**
    - add a function to return total number of events registered (e6381f9)
  - **TBBR**
    - increase PK_DER_LEN size (1ef303f)
- **Tools**
  - **Firmware Image Package Tool**
    - add cca, core_swd, plat cert in FIP (147f52f)
  - **Certificate Creation Tool**

    * define the cca chain of trust (0a6bf81)
    * update for ECDSA brainpoolP256r/t1 support (e78ba69)

- **Dependencies**
  - **Compiler runtime libraries**
    * update compiler-rt source files (8a6a956)
  - **libfdt**
    * add function to set MAC addresses (1aa7e30)
    * upgrade libfdt source files (94b2f94)
  - **zlib**
    * update zlib source files (a194255)

14.8.3 Resolved Issues

- **Architecture**
  - **Performance Monitors Extension (FEAT_PMUv3)**
    * add sensible default for MDCR_EL2 (7f85619)
  - **Scalable Matrix Extension (FEAT_SME)**
    * add missing ISBs (46e92f2)
- **Platforms**
  - **Arm**
    * **FVP**
      · fdts: Fix idle-states entry method (0e3d880)
      · fdts: fix memtimer subframe addressing (3fd12bb)
      · fdts: unify and fix PSCI nodes (6b2721c)
    * **FVP Versatile Express**
      · fdts: Fix vexpress,config-bus subnode names (60da130)
    * **Morello**
      · dts: add model names (30df890)
      · dts: fix DP SMMU IRQ ordering (fba729b)
      · dts: fix DT node naming (41c310b)
      · dts: fix GICv3 compatible string (982f258)
      · dts: fix SCMI shmem/mboxes grouping (8aeb1fc)
      · dts: fix SMMU IRQ ordering (5016ee4)

      - dts: fix stdout-path target (67a8a5c)
      - dts: remove #a-c and #s-c from memory node (f33e113)
      - dts: use documented DPU compatible string (3169572)
      - move BL31 to run from DRAM space (05330a4)
    - **N1SDP**
      - add numa node id for pcie controllers (2974d2f)
      - mapping Run-time UART to IOFPGA UART0 (4a81e91)
      - replace non-inclusive terms from dts file (e6ffafb)
    - **TC**
      - resolve the static-checks errors (066450a)
      - tc2 bl1 start address shifted by one page (8597a8c)
  - **Intel**
    - fix asynchronous read response by copying data to input buffer (dd7adcf)
    - fix Mac verify update and finalize for return response data (fbf7aef)
  - **MediaTek**
    - remove unused cold_boot.[c|h] (8cd3b69)
    - switch console to runtime state before leaving BL31 (fcf4dd9)
    - use uppercase for definition (810d568)
    - wrap cold_boot.h with MTK_SIP_KERNEL_BOOT_ENABLE (24476b2)
    - **MT8186**
      - fix SCP permission (8a998b5)
      - fix EMI_MPU domain setting for DSP (28a8b73)
      - fix the DRAM voltage after the system resumes (600f168)
      - move SSPM base register definition to platform_def.h (2a2b51d)
    - **MT8188**
      - add mmap entry for CPU idle SRAM (32071c0)
      - refine c-state power domain for extensibility (e35f4cb)
      - refine gic init flow after system resume (210ebbb)
  - **NXP**
    - **i.MX**
      - **i.MX 8M**
        - correct serial output for HAB JR0 (6e24d79)

            · fix dram retention fsp_table access (6c8f523)
            · move caam init after serial init (901d74b)
            · update poweroff related SNVS_LPCR bits only (ad6eb19)
            · **i.MX 8Q**
                · correct architected counter frequency (21189b8)

- **QEMU**
  - enable SVE and SME (337ff4f)
- **QTI**
  - adding secure rm flag (b5959ab)
- **Raspberry Pi**
  - **Raspberry Pi 3**
    - tighten platform pwr_domain_pwr_down_wfi behaviour (028c4e4)
- **Renesas**
  - **R-Car**
    - **R-Car 3**
      - fix RPC-IF device node name (08ae247)
- **Rockchip**
  - align fdt buffer on 8 bytes (621acbd)
  - **RK3399**
    - explicitly define the sys_sleep_flag_sram type (7a5e90a)
- **Socionext**
  - **Synquacer**
    - increase size of BL33 (a12a66d)
- **ST**
  - add max size for FIP in eMMC boot part (e7cb4a8)
  - add missing string.h include (0d33d38)
  - **STM32MP1**
    - enable crash console in FIQ handler (484e846)
    - fdts: stm32mp1: align DDR regulators with new driver (9eed71b)
    - update the FIP load address for serial boot (32f2ca0)
    - **STM32MP13**
      - correct USART addresses (de1ab9f)

- **Xilinx**
  - include missing header (28ba140)
  - miscellaneous fixes for xilinx platforms (bfc514f)
  - remove unnecessary header include (0ee2dc1)
  - update define for ZynqMP specific functions (24b5b53)
  - **Versal**
    - add SGI register call version check (5897e13)
    - enable a72 erratum 859971 and 1319367 (769446a)
    - fix code indentation issues (72583f9)
    - fix macro coding style issues (80806aa)
    - fix Misra-C violations in bl31_setup and pm_svc_main (68ffcd1)
    - remove clock related macros (47f8145)
    - resolve misra 10.1 warnings (19f92c4)
    - resolve misra 15.6 warnings (1117a16)
    - resolve misra 8.13 warnings (3d2ebe7)
    - resolve the misra 4.6 warnings (f7c48d9)
    - resolve the misra 4.6 warnings (912b7a6)
    - route GIC IPI interrupts during setup (04cc91b)
    - use only one space for indentation (dee5885)
  - **Versal NET**
    - Enable a78 errata workarounds (bcc6e4a)
    - add default values for silicon (faa22d4)
    - use api_id directly without FUNCID_MASK (b0eb6d1)
  - **ZynqMP**
    - fix coverity scan warnings (1ac6af1)
    - ensure memory write finish with dsb() (ac6c135)
    - fix for incorrect afi write mask value (4264bd3)
    - move bl31 with DEBUG=1 back to OCM (389594d)
    - move debug bl31 based address back to OCM (0ba3d7a)
    - remove additional 0x in %p print (05a6107)
    - resolve misra 4.6 warnings (cdb6211)
    - resolve misra 8.13 warnings (8695ffc)

        - resolve MISRA-C:2012 R.10.1 warnings (c889088)
        - resolve the misra 4.6 warnings (15dc3e4)
        - resolve the misra 4.6 warnings (ffa9103)
        - resolve the misra 8.6 warnings (7b1a6a0)

- **Bootloader Images**
  - **BL31**
    - allow use of EHF with S-EL2 SPMC (7c2fe62)
    - harden check in delegate_async_ea (d435238)
    - pass the EA bit to 'delegate_sync_ea' (df56e9d)
- **Services**
  - **RME**
    - refactor RME fid macros (fb00dc4)
    - relax RME compiler requirements (7670ddb)
    - update FVP platform token (364b4cd)
    - use RMM shared buffer for attest SMCs (dc65ae4)
    - xlat table setup fails for bl2 (e516ba6)
    - **RMMD**
      - return X4 output value (8e51ccc)
  - **SPM**
    - **EL3 SPMC**
      - check descriptor size for overflow (eed15e4)
      - compute full FF-A V1.1 desc size (be075c3)
      - deadlock when relinquishing memory (ac568b2)
      - error handling in allocation (cee8bb3)
      - fix detection of overlapping memory regions (0dc3518)
      - fix incomplete reclaim validation (c4adbe6)
      - fix location of fragment length check (21ed9ea)
      - fix relinquish validation check (b4c3621)
- **Libraries**
  - **CPU Support**
    - fix cpu version check for Neoverse N2, V1 (03ebf40)
    - workaround for Cortex-A510 erratum 2666669 (afb5d06)

* workaround for Cortex-A710 2216384 (b781fcf)
* workaround for Cortex-A710 erratum 2291219 (888eafa)
* workaround for Cortex-A76 erratum 2743102 (4927309)
* workaround for Cortex-A77 erratum 2743100 (4fdeaff)
* workaround for Cortex-A78C erratum 2376749 (5d3c1f5)
* workaround for Cortex-X3 erratum 2313909 (7954412)
* workaround for Neoverse N1 erratum 2743102 (8ce4050)
* workaround for Neoverse-N2 erratum 2326639 (43438ad)
* workaround for Neoverse-N2 erratum 2388450 (884d515)
* workaround for Cortex A78C erratum 2242638 (6979f47)
* workaround for Cortex-A510 erratum 2347730 (11d448c)
* workaround for Cortex-A510 erratum 2371937 (a67c1b1)
* workaround for Cortex-A710 erratum 2147715 (3280e5e)
* workaround for Cortex-A710 erratum 2371105 (3220f05)
* workaround for Cortex-A77 erratum 2356587 (7bf1a7a)
* workaround for Cortex-A78C 2132064 (8008bab)
* workaround for Cortex-A78C erratum 2395411 (4b6f002)
* workaround for Cortex-X2 erratum 2371105 (bc0f84d)
* workaround for Neoverse-N2 erratum 2376738 (e6602d4)
* workaround for Neoverse-V1 erratum 1618635 (14a6fed)
* workaround for Neoverse-V1 erratum 2294912 (39eb5dd)
* workaround for Neoverse-V1 erratum 2372203 (57b73d5)

- **EL3 Runtime**
  * **RAS**
    - restrict RAS support for NS world (46cc41d)
    - trap "RAS error record" accesses only for NS (00e8f79)
- **FCONF**
  * fix type error displaying disable_auth (381f465)
- **PSCI**
  * fix MISRA failure - Memory - illegal accesses (0551aac)
- **GPT**
  * correct the GPC enable sequence (14cddd7)

- **C Standard Library**
  - pri*ptr macros for aarch64 (d307229)
- **PSA**
  - fix Null pointer dereference error (c32ab75)
  - update measured boot handle (4d879e1)
  - add missing semicolon (d219ead)
  - align with original API in tf-m-extras (471c989)
  - extend measured boot logging (901b0a3)
- **Context Management**
  - remove explicit ICC_SRE_EL2 register read (2b28727)
- **Semihosting**
  - fix seek call failure check (7c49438)

- **Drivers**
  - **Authentication**
    - correct sign-compare warning (ed38366)
  - **Measured Boot**
    - add SP entries to event_log_metadata (e637a5e)
    - clear the entire digest array of Startup Locality event (70b1c02)
    - fix verbosity level of RSS digests traces (2abd317)
  - **MMC**
    - remove broken, unsecure, unused eMMC RPMB handling (86b015e)
    - resolve the build error (ccf8392)
  - **SCMI**
    - base: fix protocol list querying (cad90b5)
    - base: fix protocol list response size (d323f0c)
  - **UFS**
    - add retries to ufs_read_capacity (28645eb)
    - fix slot base address computation (7d9648d)
    - init utrlba/utrlbau with desc_base (9d6d1a9)
    - point utrlbau to header instead of upiu (9d3f6c4)
    - removes dp and run-stop polling loops (660c208)
    - retry commands on unit attention (3d30955)

- **Arm**
  - **GIC**
    - **GICv3**
      - fix overflow caused by left shift (6aea762)
      - update the affinity mask to 8 bit (e689048)
      - **GIC-600**
        - implement workaround to forward highest priority interrupt (e1b15b0)
  - **RSS**
    - clear the message buffer (e3a6fb8)
    - determine the size of sw_type in RSS mboot metadata (2c8f2a9)
    - fix build issues with comms protocol (ab545ef)
    - reduce input validation for measured boot (13a129e)
    - remove dependency on attestation header (6aa7154)
    - rename AP-RSS message size macro (70247dd)
- **NXP**
  - **DDR**
    - fix firmware buffer re-mapping issue (742c23a)
- **ST**
  - **Clock**
    - correct MISRA C2012 15.6 (56f895e)
    - correctly check ready bit (3b06a53)

- **Miscellaneous**
  - **AArch64**
    - make AArch64 FGT feature detection more robust (c687776)
  - **Debug**
    - backtrace stack unwind misses lr adjustment (a149eb4)
    - decouple "get_el_str()" from backtrace (0ae4a3a)
  - **FDTs**
    - **STM32MP1**
      - **STM32MP13**
        - align sdmmc pins with kernel (c7ac7d6)
        - cleanup DT files (4c07deb)

        - correct PLL nodes name (93ed4f0)
        - remove secure status (8ef8e0e)
        - update SDMMC max frequency (c9a4cb5)
  - **Security**
    - optimisations for CVE-2022-23960 (e74d658)
- **Documentation**
  - document missing RMM-EL3 runtime services (e50fedb)
  - add LTS maintainers (ab0d4d9)
  - update maintainers list (f23ce63)
  - **Changelog**
    - fix the broken link to commitlintrc.js (c1284a7)
- **Build System**
  - disable default PIE when linking (7b59241)
  - discard sections also with SEPARATE_NOBITS_REGION (64207f8)
  - ensure that the correct rule is called for tools (598b166)
  - fix arch32 build issue for clang (94eb127)
  - make TF-A use provided OpenSSL binary (e95abc4)
- **Tools**
  - **Secure Partition Tool**
    - fix concurrency issue for SP packages (0aaa382)
    - operators “is/is not” in sp_mk_gen.py (1a28f29)
    - ‘sp_mk_generator.py’ reference to undef var (0be2475)
- **Dependencies**
  - add missing aeabi_memcpy.S (93cec69)

14.9 2.7.0 (2022-05-20)

14.9.1 New Features

- **Architecture**
  - **Statistical profiling Extension (FEAT_SPE)**
    - add support for FEAT_SPEv1p2 (f20eb89)
  - **Branch Record Buffer Extension (FEAT_BRBE)**

        * add BRBE support for NS world (744ad97)

    - **Extended Cache Index (FEAT_CCIDX)**

        * update the do_dcsw_op function to support FEAT_CCIDX (d0ec1cc)

- **Platforms**

    - add SZ_* macros (1af59c4)

    - **Allwinner**

        * add SMCCC SOCID support (436cd75)
        * allow to skip PMIC regulator setup (67412e4)
        * apx803: add aldo1 regulator (a29f6e7)
        * choose PSCI states to avoid translation (159c36f)
        * provide CPU idle states to the rich OS (e2b1877)
        * simplify CPU_SUSPEND power state encoding (52466ec)

    - **Arm**

        * **FVP**

            · measure critical data (cf21064)
            · update HW_CONFIG DT loading mechanism (39f0b86)
            · enable RSS backend based measured boot (c44e50b)

        * **Morello**

            · add changes to enable TBBR boot (4af5397)
            · add DTS for Morello SoC platform (572c8ce)
            · add support for nt_fw_config (6ad6465)
            · add TARGET_PLATFORM flag (8840711)
            · configure DMC-Bing mode (9b8c431)
            · expose scmi protocols in fdts (87639aa)
            · split platform_info sds struct (4a7a9da)
            · zero out the DDR memory space (2d39b39)

        * **N1SDP**

            · add support for nt_fw_config (cf85030)
            · enable trusted board boot on n1sdp (fe2b37f)

        * **RD**

            · **RD-N2**

                · add board support for rdn2cfg2 variant (efeb438)

        - add support for rdedmunds variant (ef515f0)

    - **SGI**

      - add page table translation entry for secure uart (33d10ac)
      - deviate from arm css common uart related definitions (f2cccca)
      - enable fpregs context save and restore (18fa43f)
      - route TF-A logs via secure uart (987e2b7)

    - **TC**

      - add reserved memory region for Gralloc (ad60a42)
      - enable CI-700 PMU for profiling (fbfc598)
      - enable GPU (82117bb)
      - enable SMMU for DPU (4a6ebee)
      - enable tracing (59da207)

    - **Corstone-1000**

      - identify bank to load fip (cf89fd5)
      - implement platform specific psci reset (a599c80)
      - made changes to accommodate 3MB for optee (854d1c1)

  - **Intel**

    - add macro to switch between different UART PORT (447e699)
    - add RSU 'Max Retry' SiP SMC services (4c26957)
    - add SiP service for DCMF status (984e236)
    - add SMC for enquiring firmware version (c34b2a7)
    - add SMC support for Get USERCODE (93a5b97)
    - add SMC support for HWMON voltage and temp sensor (52cf9c2)
    - add SMC support for ROM Patch SHA384 mailbox (77902fc)
    - add SMC/PSCI services for DCMF version support (44eb782)
    - add SMPLSEL and DRVSEL setup for Stratix 10 MMC (bb0fcc7)
    - add support for F2S and S2F bridge SMC with mask to enable, disable and reset bridge (11f4f03)
    - allow to access all register addresses if DEBUG=1 (7e954df)
    - create source file for firewall configuration (afa0b1a)
    - enable firewall for OCRAM in BL31 (ae19fef)
    - enable SMC SoC FPGA bridges enable/disable (b7f3044)

- extend attestation service to Agilex family (581182c)
- implement timer init divider via cpu frequency. (#1) (f65bdf3)
- initial commit for attestation service (d174083)
- single certificate feature enablement (7facace)
- support AES Crypt Service (6726390)
- support crypto service key operation (342a061)
- support crypto service session (6dc00c2)
- support ECDH request (4944686)
- support ECDSA Get Public Key (d2fee94)
- support ECDSA HASH Signing (6925410)
- support ECDSA HASH Verification (7e25eb8)
- support ECDSA SHA-2 Data Signature Verification (5830506)
- support ECDSA SHA-2 Data Signing (07912da)
- support extended random number generation (24f9dc8)
- support HMAC SHA-2 MAC verify request (c05ea29)
- support session based SDOS encrypt and decrypt (537ff05)
- support SHA-2 hash digest generation on a blob (7e8249a)
- support SiP SVC version (f0c40b8)
- support version 2 SiP SVC SMC function ID for mailbox commands (c436707)
- support version 2 SiP SVC SMC function ID for non-mailbox commands (ad47f14)
- update to support maximum response data size (b703fac)

Marvell

- **Armada**
  - **A3K**
    - add north and south bridge reset registers (a4d35ff)

MediaTek

- introduce mtk makefile (500d40d)
- **MT8195**
  - apply erratas of CA78 for MT8195 (c21a736)
  - add EMI MPU surppot for SCP and DSP (690cb12)
  - dump EMI MPU configurations (20ef588)
  - improve SPM wakeup log (ab45305)

* **MT8186**
  - add DFD control in SiP service (e46e9df)
  - add SPM suspend driver (7ac6a76)
  - add Vcore DVFS driver (635e6b1)
  - disable 26MHz clock while suspending (9457cec)
  - initialize platform for MediaTek MT8186 (27132f1)
  - add power-off function for PSCI (a68346a)
  - add CPU hotplug (1da57e5)
  - add DCM driver (95ea87f)
  - add EMI MPU basic driver (1b17e34)
  - add MCDI drivers (06cb65e)
  - add pinctrl support (af5a0c4)
  - add pwrap and pmic driver (5bc88ec)
  - add reboot function for PSCI (24dd5a7)
  - add RTC drivers (6e5d76b)
  - add SiP service (5aab27d)
  - add sys_cirq support (109b91e)
  - apply erratas for MT8186 (572f8ad)
  - initialize delay_timer (d73e15e)
  - initialize GIC (206f125)
  - initialize systimer (a6a0af5)

- **NXP**
  * add SoC erratum a008850 (3d14a30)
  * add ifc nor and nand as io devices (b759727)
  * add RCPM2 registers definition (d374060)
  * add CORTEX A53 helper functions (3ccc8ac)
  * **i.MX**
    - **i.MX 8M**
      - add a simple csu driver for imx8m family (71c40d3)
      - add imx csu/rdc enum type defines for imx8m (0c6dfc4)
      - enable conditional build for SDEI (d2a339d)
      - enable the coram_s tz by default on imx8mn/mp (d5ede92)

- enable the csu init on imx8m (0a76495)
- do not release JR0 to NS if HAB is using it (77850c9)
- switch to xlat_tables_v2 (4f8d5b0)
- **i.MX 8M Mini**
  - enable optee fdt overlay support (9d0eed1)
  - enable Trusty OS on imx8mm (ff3acfe)
  - add support for measured boot (cb2c4f9)
- **i.MX 8M Plus**
  - add trusty for imx8mp (8b9c21b)
  - enable BL32 fdt overlay support on imx8mp (aeff146)
- **i.MX 8M Nano**
  - enable optee fdt overlay support (2612891)
  - enable Trusty OS for imx8mn (99349c8)
- **i.MX 8M Q**
  - enable optee fdt overlay support (023750c)
  - enable trusty for imx8mq (a18e393)

* **Layerscape**
  - add CHASSIS 3 support for tbbr (9550ce9)
  - add new soc errata a009660 support (785ee93)
  - add new soc errata a010539 support (85bd092)
  - add soc helper macro definition for chassis 3 (602cf53)
  - define more chassis 3 hardware address (0d396d6)
  - print DDR errata information (3412716)
  - **LS1043A**
    - add ls1043a soc support (3b0de91)
    - **LS1043ARDB**
      - add ls1043ardb board support (e4bd65f
  - **LX2**
    - enable DDR erratas for lx2 platforms (cd960f5)
  - **LS1046A**
    - add new SoC platform ls1046a (cc70859)
    - **LS1046ARDB**

- add ls1046ardb board support (bb52f75)
- **LS1046AFRWY**
  - add ls1046afrwy board support (b51dc56)
- **LS1046AQDS**
  - add board ls1046aqds support (16662dc)
- **LS1088A**
  - add new SoC platform ls1088a (9df5ba0)
  - **LS1088ARDB**
    - add ls1088ardb board support (2771dd0)
  - **LS1088AQDS**
    - add ls1088aqds board support (0b0e676)

- **QEMU**
  - add SPMD support with SPMC at S-EL1 (f58237c)
  - add support for measured boot (5e69026)
- **QTI**
  - **MSM8916**
    - allow booting secondary CPU cores (a758c0b)
    - initial platform port (dddba19)
    - setup hardware for non-secure world (af64473)
- **Renesas**
  - **R-Car**
    - **R-Car 3**
      - modify sequence for update value for WUPMSKCA57/53 (d9912cf)
      - modify type for Internal function argument (ffb725b)
      - update IPL and Secure Monitor Rev.3.0.3 (14d9727)
- **ST**
  - add a function to configure console (53612f7)
  - add STM32CubeProgrammer support on UART (fb3e798)
  - add STM32MP_UART_PROGRAMMER target (9083fa1)
  - add early console in BL2 (c768b2b)
  - disable authentication based on part_number (49abdfd)
  - get pin_count from the gpio-ranges property (d0f2cf3)

    * map 2MB for ROM code (1697ad8)
    * protect UART during platform init (acf28c2)
    * update stm32image tool for header v2 (2d8886a)
    * update the security based on new compatible (812daf9)
    * use newly introduced clock framework (33667d2)
    * **ST32MP1**
        - adaptations for STM32MP13 image header (a530874)
        - add "Boot mode" management for STM32MP13 (296ac80)
        - add a second fixed regulator (225ce48)
        - add GUID values for updatable images (8d6b476)
        - add GUID's for identifying firmware images to be booted (41bd8b9)
        - add helper to enable high speed mode in low voltage (dea02f4)
        - add logic to pass the boot index to the Update Agent (ba02add)
        - add logic to select the images to be booted (8dd7553)
        - add NVMEM layout compatibility definition (dfbdbd0)
        - add part numbers for STM32MP13 (30eea11)
        - add regulator framework compilation (bba9fde)
        - add sdmmc compatible in platform define (3331d36)
        - add sign-compare warning (c10f3a4)
        - add stm32_get_boot_interface function (a6bfa75)
        - add support for building the FWU feature (ad216c1)
        - add support for reading the metadata partition (0ca180f)
        - add timeout in IO compensation (de02e9b)
        - allow configuration of DDR AXI ports number (88f4fb8)
        - call pmic_voltages_init() in platform init (ffd1b88)
        - chip rev. Z is 0x1001 on STM32MP13 (ef0b8a6)
        - enable BL2_IN_XIP_MEM to remove relocation sections (d958d10)
        - enable format-signedness warning (cff26c1)
        - get CPU info from SYSCFG on STM32MP13 (6512c3a)
        - introduce new flag for STM32MP13 (bdec516)
        - manage HSLV on STM32MP13 (fca10a8)
        - manage monotonic counter (f5a3688)

      - new way to access platform OTP (ae3ce8b)
      - preserve the PLL4 settings for USB boot (bf1af15)
      - register fixed regulator (967a8e6)
      - remove unsupported features on STM32MP13 (111a384)
      - retry 3 times FWU trial boot (f87de90)
      - select platform compilation either by flag or DT (99a5d8d)
      - skip TOS_FW_CONFIG if not in FIP (b706608)
      - stm32mp_is_single_core() for STM32MP13 (7b48a9f)
      - update BACKUP_BOOT_MODE for STM32MP13 (4b031ab)
      - update boot API for header v2.0 (5f52eb1)
      - update CFG0 OTP for STM32MP13 (1c37d0c)
      - update console management for SP_min (aafff04)
      - update IO compensation on STM32MP13 (8e07ab5)
      - update IP addresses for STM32MP13 (52ac998)
      - update memory mapping for STM32MP13 (48ede66)
      - updates for STM32MP13 device tree compilation (d38eaf9)
      - usb descriptor update for STM32MP13 (d59b9d5)
      - use clk_enable/disable functions (c7a66e7)
      - use only one filter for TZC400 on STM32MP13 (b7d0058)
      - warn when debug enabled on secure chip (ac4b8b0)

- **Texas Instruments**
  - add enter sleep method (cf5868b)
  - add gic save and restore calls (b40a467)
  - add PSCI handlers for system suspend (2393c27)
  - allow build config of low power mode support (a9f46fa)
  - increase SEC_SRAM_SIZE to 128k (38164e6)
- **Xilinx**
  - **Versal**
    - add SPP/EMU platform support for versal (be73459)
    - add common interfaces to handle EEMI commands (1397967)
    - add SMCCC call TF_A_PM_REGISTER_SGI (fcf6f46)
    - add support to reset SGI (bf70449)

        - add UART1 as console (2c79149)
        - enhance PM_IOCTL EEMI API to support additional arg (d34a5db)
        - get version for ATF related EEMI APIs (da6e654)
        - remove the time stamp configuration (18e2a79)
    - **ZynqMP**
        - disable the -mbranch-protection flag (67abd47)
        - fix section `coherent_ram' will not fit in region` RAM' (9b4ed0a)
        - add feature check support (223a628)
        - add support to get info of xilfpga (cc077c2)
        - add uart1 as console (ea66e4a)
        - increase the max xlat tables when debug build is enabled (4c4b961)
        - pass ioctl calls to firmware (76ff8c4)
        - pm_api_clock_get_num_clocks cleanup (e682d38)

- **Bootloader Images**
  - add XLAT tables symbols in linker script (bb5b942)
  - **BL2**
    - add support to separate no-loadable sections (96a8ed1)
  - **BL31**
    - aarch64: RESET_TO_BL31_WITH_PARAMS (25844ff)
- **Services**
  - **RME**
    - add dummy platform token to RMMD (0f9159b)
    - add dummy realm attestation key to RMMD (a043510)
  - **SPM**
    - update ff-a boot protocol documentation (573ac37)
    - **EL3 SPMC**
      - allow BL32 specific defines to be used by SPMC_AT_EL3 (2d65ea1)
      - add plat hook for memory transactions (a8be4cd)
      - add EL3 SPMC #defines (44639ab)
      - introduce accessor function to obtain datastore (6a0788b)
      - add FF-A secure partition manager core (5096aeb)
      - add FFA_FEATURES handler (55a2963)

- add FFA_PARTITION_INFO_GET handler (f74e277)
- add FFA_RUN handler (aad20c8)
- add FFA_RX_RELEASE handler (f0c25a0)
- add function to determine the return path from the SPMC (20fae0a)
- add helper function to obtain endpoint mailbox (f16b6ee)
- add helper function to obtain hyp structure (a7c0050)
- add helper to obtain a partitions FF-A version (c2b1434)
- add partition mailbox structs (e1df600)
- add support for direct req/resp (9741327)
- add support for FF-A power mgmt. messages in the EL3 SPMC (59bd2ad)
- add support for FFA_MSG_WAIT (c4db76f)
- add support for FFA_SPM_ID_GET (46872e0)
- add support for forwarding a secure interrupt to the SP (729d779)
- add support for handling FFA_ERROR ABI (d663fe7)
- add support for v1.1 FF-A boot protocol (2e21921)
- add support for v1.1 FF-A memory data structures (7e804f9)
- enable building of the SPMC at EL3 (1d63ae4)
- enable checking of execution ctx count (5b0219d)
- enable handling FF-A RX/TX Mapping ABIs (1a75224)
- enable handling FFA_VERSION ABI (0c7707f)
- enable handling of the NS bit (0560b53)
- enable parsing of messaging methods from manifest (3de378f)
- enable parsing of UUID from SP Manifest (857f579)
- enable the SPMC to pass the linear core ID in a register (f014300)
- prevent read only xlat tables with the EL3 SPMC (70d986d)
- support FFA_ID_GET ABI (d5fe923)
- allow forwarding of FFA_FRAG_RX/TX calls (642db98)
- enable handling of FF-A SMCs with the SPMC at EL3 (bb01a67)
- update SPMC init flow to use EL3 implementation (6da7607)
- add logical partition framework (7affa25)
- add FF-A memory management code (e0b1a6d)
- prevent duplicated sharing of memory regions (fef85e1)

        · support multiple endpoints in memory transactions (f0244e5)

      * **SPMD**

        · forward FFA_VERSION from SPMD to SPMC (9944f55)

        · enable SPMD to forward FFA_VERSION to EL3 SPMC (9576fa9)

        · add FFA_MSG_SEND2 forwarding in SPMD (c2eba07)

        · add FFA_RX_ACQUIRE forwarding in SPMD (d555233)

      * **SPM MM**

        · add support to save and restore fp regs (15dd6f1)

- **Libraries**

  - **CPU Support**

    * add library support for Poseidon CPU (1471475)
    * add support for Cortex-X1 (6e8eca7)
    * add L1PCTL macro definiton for CPUACTLR_EL1 (8bbb1d8)

  - **EL3 Runtime**

    * add arch-features detection mechanism (6a0da73)
    * replace ARM_ARCH_AT_LEAST macro with FEAT flags (0ce220a)

  - **FCONF**

    * add a helper to get image index (9e3f409)
    * add NS load address in configuration DTB nodes (ed4bf52)

  - **Standard C Library**

    * add support for length specifiers (701e94b)

  - **PSA**

    * add initial attestation API (0848565)
    * add measured boot API (758c647)
    * mock PSA APIs (0ce2072)

- **Drivers**

  - **Generic Clock**

    * add a minimal clock framework (847c6bc)

  - **FWU**

    * add a function to pass metadata structure to platforms (9adce87)
    * add basic definitions for GUID handling (19d63df)
    * add platform hook for getting the boot index (40c175e)

    * pass a const metadata structure to platform routines (6aaf257)
    * simplify the assert to check for fwu init (40b085b)

  - **Measured Boot**
    * add RSS backend (0442ebd)

  - **GUID Partition Tables Support**
    * add a function to identify a partition by GUID (3cb1065)
    * cleanup partition and gpt headers (2029f93)
    * copy the partition GUID into the partition structure (7585ec4)
    * make provision to store partition GUID value (938e8a5)
    * verify crc while loading gpt header (a283d19)

  - **Arm**
    * **GIC**
      · allow overriding GICD_PIDR2_GICV2 address (a7521bd)
      · **GIC-600AE**
        · disable SMID for unavailable blocks (3f0094c)
        · enable all GICD, PPI, ITS SMs (6a1c17c)
        · introduce support for RAS error handling (308dce4)
    * **SMMU**
      · add SMMU abort transaction function (6c5c532)
      · configure SMMU Root interface (52a314a)
    * **MHU**
      · add MHU driver (af26d7d)
    * **RSS**
      · add RSS communication driver (ce0c40e)
    * **TZC**
      · **TZC-380**
        · add sub-region register definition (fdafe2b)

  - **Marvell**
    * **Armada**
      · **A3K**
        · **A3720**
          · preserve x1/x2 regs in console_a3700_core_init() (7c85a75)

- **MediaTek**
  - **APU**
    - add mt8195 APU clock and pll SiP call (296b590)
    - add mt8195 APU iommap regions (339e492)
    - add mt8195 APU mcu boot and stop SiP call (88906b4)
- **NXP**
  - **DCFG**
    - add Chassis 3 support (df02aee)
    - add gic address align register definition (3a8c9d7)
    - add some macro definition (1b29fe5)
  - **NXP Crypto**
    - add chassis 3 support (d60364d)
  - **DDR**
    - add rawcard 1F support (f2de48c)
    - add workaround for errata A050958 (291adf5)
  - **GIC**
    - add some macros definition for gicv3 (9755fd2)
  - **CSU**
    - add bypass bit mask definition (ec5fc50)
  - **IFC NAND**
    - add IFC NAND flash driver (28279cf)
  - **IFC NOR**
    - add IFC nor flash driver (e2fdc77)
  - **TZC-380**
    - add tzc380 platform driver support (de9e57f)
- **ST**
  - introduce fixed regulator driver (5d6a264)
  - **Clock**
    - add clock driver for STM32MP13 (9be88e7)
    - assign clocks to the correct BL (7418cf3)
    - check HSE configuration in serial boot (31e9750)
    - define secure and non-secure gate clocks (aaa09b7)

        · do not refcount on non-secure clocks in bl32 (3d69149)
        · manage disabled oscillator (bcccdac)
    * **DDR**
        · add read valid training support (5def13e)
    * **GPIO**
        · allow to set a gpio in output mode (53584e1)
        · do not apply secure config in BL2 (fc0aa10)
        · add a function to reset a pin (737ad29)
    * **SDMMC2**
        · allow compatible to be defined in platform code (6481a8f)
        · manage cards power cycle (258bef9)
    * **ST PMIC**
        · add pmic_voltages_init() function (5278ec3)
        · register the PMIC to regulator framework (85fb175)
    * **STPMIC1**
        · add new services (ea552bf)
        · add USB OTG regulators (13fbfe0)
    * **Regulator**
        · add support for regulator-always-on (9b4ca70)
        · add a regulator framework (d5b4a2c)
    * **UART**
        · manage oversampling by 8 (1f60d1b)
        · add uart driver for STM32MP1 (165ad55)

- **Miscellaneous**
  - **Debug**
    * update print_memory_map.py (d16bfe0)
  - **DT Bindings**
    * add bindings for STM32MP13 (1b8898e)
    * add TZC400 bindings for STM32MP13 (24d3da7)
  - **FDT Wrappers**
    * add function to find or add a sudnode (dea8ee0)
  - **FDTs**

        * add the ability to supply idle state information (2b2b565)
        * **STM32MP1**
            - add DDR support for STM32MP13 (e6fddbc)
            - add DT files for STM32MP13 (3b99ab6)
            - add nvmem_layout node and OTP definitions (ff8767c)
            - add st-io_policies node for STM32MP13 (2bea351)
            - add support for STM32MP13 DK board (2b7f7b7)
            - update NVMEM nodes (375b79b)

- **Documentation**
    - context management refactor proposal (3274226)
    - **Threat Model**
        * Threat Model for TF-A v8-R64 Support (dc66922)
- **Tools**
    - **Secure Partition Tool**
        * add python SpSetupActions framework (b1e6a41)
        * delete c version of the sptool (f4ec476)
        * python version of the sptool (2e82874
        * use python version of sptool (822c727)

14.9.2 Resolved Issues

- **Architecture**
    - **Activity Monitors Extension (FEAT_AMU)**
        * add default value for ENABLE_FEAT_FGT and ENABLE_FEAT_ECV flags (820371b)
        * fault handling on EL2 context switch (f74cb0b)
        * limit virtual offset register access to NS world (a4c3945)
    - **Scalable Vector Extension (FEAT_SVE)**
        * disable ENABLE_SVE_FOR_NS for AARCH32 (24ab2c0)
- **Platforms**
    - **Allwinner**
        * improve DTB patching error handling (79808f1)
    - **Arm**
        * fix fvp and juno build with USE_ROMLIB option (861250c)

    * increase ARM_BL_REGIONS count (dcb1959)
    * remove reclamation of functions starting with “init” (6c87abd)
    * use PLAT instead of TARGET_PLATFORM (c5f3de8)
    * fix SP count limit without dual root CoT (9ce15fe)
    * **FVP**
      · FCONF Trace Not Shown (0c55c10)
      · disable reclaiming init code by default (fdb9166)
      · extend memory map to include all DRAM memory regions (e803542)
      · fix NULL pointer dereference issue (a42b426)
      · op-tee sp manifest doesn’t map gicd (69cde5c)
    * **Morello**
      · change the AP runtime UART address (07302a2)
      · fix SoC reference clock frequency (e8b7a80)
      · include errata workaround for 1868343 (f94c84b)
    * **SGI**
      · disable SVE for NS to support SPM_MM builds (78d7e81)
    * **TC**
      · remove the bootargs node (68fe3ce)
    * **Corstone-1000**
      · change base address of FIP in the flash (1559450)
  - **Broadcom**
    * allow build to specify mbedTLS absolute path (903d574)
    * fix the build failure with mbedTLS config (95b5c01)
  - **Intel**
    * add flash dcache after return response for INTEL_SIP_SMC_MBOX_SEND_CMD (ac097fd)
    * allow non-secure access to FPGA Crypto Services (FCS) (4837a64)
    * always set doorbell to SDM after sending command (e93551b)
    * assert if bl_mem_params is NULL pointer (35fe7f4)
    * bit-wise configuration flag handling (276a436)
    * change SMC return arguments for INTEL_SIP_SMC_MBOX_SEND_CMD (108514f)
    * configuration status based on start request (e40910e)

      * define macros to handle buffer entries (7db1895)
      * enable HPS QSPI access by default (000267b)
      * extend SDM command to return the SDM firmware version (c026dfe)
      * extending to support large file size for AES encryption and decryption (dcb144f)
      * extending to support large file size for SHA-2 ECDSA data signing and signature verifying (1d97dd7)
      * extending to support large file size for SHA2/HMAC get digest and verifying (70a7e6a)
      * fix bit masking issue in intel_secure_reg_update (c9c0709)
      * fix configuration status based on start request (673afd6)
      * fix ddr address range checker (12d71ac)
      * fix ECC Double Bit Error handling (c703d75)
      * fix fpga config write return mechanism (ef51b09)
      * flush dcache before sending certificate to mailbox (49d44ec)
      * get config status OK status (07915a4)
      * introduce a generic response error code (651841f)
      * make FPGA memory configurations platform specific (f571183)
      * modify how configuration type is handled (ec4f28e)
      * null pointer handling for resp_len (a250c04)
      * refactor NOC header (bc1a573)
      * reject non 4-byte align request size for FPGA Crypto Service (FCS) (52ed157)
      * remove redundant NOC header declarations (58690cd)
      * remove unused printout (0d19eda)
      * update certificate mask for FPGA Attestation (fe5637f)
      * update encryption and decryption command logic (02d3ef3)
      * use macro as return value (e0fc2d1)

  - **Marvell**
      * **Armada**
          - **A3K**
              - change fatal error to warning when CM3 reset is not implemented (30cdbe7)
              - fix comment about BootROM address range (5a60efa)

  - **Mediatek**
      * **MT8186**

          · remove unused files in drivers/mcdi (bc714ba)
          · extend MMU region size (0fe7ae9)
- **NVIDIA**
  - **Tegra**
    - **Tegra 194**
      - remove incorrect erxctlr assert (e272c61)
- **NXP**
  - fix total dram size checking (0259a3e)
  - increase soc name maximum length (3ccd7e4)
  - **i.MX**
    - **i.MX 8M**
      - check the validation of domain id (eb7fb93)
      - **i.MX 8M Plus**
        - change the BL31 physical load address (32d5042)
  - **Layerscape**
    - fix build issue of mmap_add_ddr_region_dynamically (e2818d0)
    - fix coverity issue (5161cfd)
    - update WA for Errata A-050426 (72feaad)
    - **LX2**
      - drop erratum A-009810 (e36b0e4)
- **Renesas**
  - **R-Car**
    - **R-Car 3**
      - change stack size of BL31 (d544dfc)
      - fix SYSTEM_OFF processing for R-Car D3 (1b49ba0)
      - fix to bit operation for WUPMSKCA57/53 (82bb6c2)
- **Socionext**
  - **Synquacer**
    - initialise CNTFRQ in Non Secure CNTBaseN (4d4911d)
- **ST**
  - add missing header include (b1391b2)
  - don’t try to read boot partition on SD cards (9492b39)

    * fix NULL pointer dereference issues (2deff90)
    * manage UART clock and reset only in BL2 (9e52d45)
    * remove extra chars from dtc version (03d2077)
    * **ST32MP1**
        - add missing debug.h (356ed96)
        - correct dtc version check (429f10e)
        - correct include order (ff7675e)
        - correct types in messages (43bbdca)
        - deconfigure UART RX pins (d7176f0)
        - do not reopen debug features (21cfa45)
        - fix enum prints (ceab2fc)
        - include assert.h to fix build failure (570c71b)
        - remove interrupt_provider warning for dtc (ca88c76)
        - restrict DEVICE2 mapping in BL2 (db3e0ec)
        - rework switch/case for MISRA (f7130e8)
        - set reset pulse duration to 31ms (9a73a56)
- **Xilinx**
    * fix coding style violations (bb1768c)
    * fix mismatching function prototype (81333ea)
    * **Versal**
        - resolve misra R10.1 in pm services (775bf1b)
        - resolve misra R10.3 (b2bb3ef)
        - resolve misra R10.3 in pm services (5d1c211)
        - resolve misra R10.6 (93d4625)
        - resolve misra R10.6 in pm services (fa98d7f)
        - resolve misra R14.4 (a62c40d)
        - resolve misra R15.6 (b9fa2d9)
        - resolve misra R15.6 in pm services (4156719)
        - resolve misra R15.7 (bc2637e)
        - resolve misra R16.3 in pm services (27ae531)
        - resolve misra R17.7 (526a1fd)
        - resolve misra R20.7 in pm services (5dada62)

        - resolve misra R7.2 (0623dce)
        - fix coverity scan warnings (0b15187)
        - fix the incorrect log message (ea04b3f)
      - **ZynqMP**
        - define and enable ARM_XLAT_TABLES_LIB_V1 (c884c9a)
        - query node status to power up APU (b35b556)
        - resolve misra 7.2 warnings (5bcbd2d)
        - resolve misra 8.3 warnings (944e7ea)
        - resolve misra R10.3 (2b57da6)
        - resolve misra R14.4 warnings (dd1fe71)
        - resolve misra R15.6 warnings (eb0d2b1)
        - resolve misra R15.7 warnings (16de22d)
        - resolve misra R16.3 warnings (e7e5d30)
        - resolve misra R8.4 warnings (610eeac)
        - update the log message to verbose (1277af9)
        - use common interface for eemi apis (a469c1e)
- **Bootloader Images**
  - **BL1**
    - invalidate SP in data cache during secure SMC (f1cbbd6)
  - **BL2**
    - correct messages with image_id (e4c77db)
    - define RAM_NOLOAD for XIP (cc562e7)
- **Services**
  - **RME**
    - enable/disable SVE/FPU for Realms (a4cc85c)
    - align RMI and GTSI FIDs with SMCCC (b9fd2d3)
    - preserve x4-x7 as per SMCCCv1.1 (1157830)
    - **TRP**
      - Distinguish between cold and warm boot (00e8113)
  - **SPM**
    - **EL3 SPMC**
      - fix incorrect FF-A version usage (25eb2d4)

        · fix FF-A memory transaction validation (3954bc3)

- **Libraries**
  - **CPU Support**
    - workaround for Cortex-A710 2282622 (ef934cd)
    - workaround for Cortex-A710 erratum 2267065 (cfe1a8f)
    - workaround for Cortex A78 AE erratum 2376748 (92e8708)
    - workaround for Cortex A78 AE erratum 2395408 (3f4d81d)
    - workaround for Cortex X2 erratum 2002765 (34ee76d)
    - workaround for Cortex X2 erratum 2058056 (e16045d)
    - workaround for Cortex X2 erratum 2083908 (1db6cd6)
    - workaround for Cortex-A510 erratum 1922240 (8343563)
    - workaround for Cortex-A510 erratum 2041909 (e72bbe4)
    - workaround for Cortex-A510 erratum 2042739 (d48088a)
    - workaround for Cortex-A510 erratum 2172148 (c0959d2)
    - workaround for Cortex-A510 erratum 2218950 (cc79018)
    - workaround for Cortex-A510 erratum 2250311 (7f304b0)
    - workaround for Cortex-A510 erratum 2288014 (d5e2512)
    - workaround for Cortex-A710 erratum 2008768 (af220eb)
    - workaround for Cortex-A710 erratum 2136059 (8a855bd)
    - workaround for Cortex-A78 erratum 2376745 (5d796b3)
    - workaround for Cortex-A78 erratum 2395406 (3b577ed)
    - workaround for Cortex-X2 errata 2017096 (e7ca443)
    - workaround for Cortex-X2 errata 2081180 (c060b53)
    - workaround for Cortex-X2 erratum 2147715 (63446c2)
    - workaround for Cortex-X2 erratum 2216384 (4dff759)
    - workaround for DSU-110 erratum 2313941 (7e3273e)
    - workaround for Rainier erratum 1868343 (a72144f)
    - workarounds for cortex-x1 errata (7b76c20)
    - use CPU_NO_EXTRA3_FUNC for all variants (b2ed998)
  - **EL3 Runtime**
    - set unset pstate bits to default (7d33ffe)
    - **Context Management**

            · add barrier before el3 ns exit (0482503)
            · remove registers accessible only from secure state from EL2 context (7f41bcc)
            · refactor the cm_setup_context function (2bbad1d)
            · remove initialization of EL2 registers when EL2 is used (fd5da7a)
            · add cm_prepare_el3_exit_ns function (8b95e84)
            · refactor initialization of EL1 context registers (b515f54)

  - **FCONF**
    * correct image_id type in messages (cec2fb2)
  - **PSCI**
    * correct parent_node type in messages (b9338ee)
  - **GPT**
    * rework delegating/undelegating sequence (6a00e9b)
  - **Translation Tables**
    * fix bug on VERBOSE trace (956d76f)
  - **Standard C Library**
    * correct some messages (a211fde)
    * fix snprintf corner cases (c1f5a09)
    * limit snprintf radix value (b30dd40)
    * snprintf: include stdint.h (410c925)
  - **Locks**
    * add __unused for clang (5a030ce)

- **Drivers**
  - **FWU**
    * rename is_fwu_initialized (aae7c96)
  - **I/O**
    * **MTD**
      · correct types in messages (6e86b46)
  - **Measured Boot**
    * add RMM entry to event_log_metadata (f4e3e1e)
  - **MTD**
    * correct types in messages (6e86b46)
  - **SCMI**

    * add missing \n in ERROR message (0dc9f52)
    * make msg_header variable volatile (99477f0)
    * use same type for message_id (2355ebf)

- **UFS**
    * delete call to inv_dcache_range for utrd (c5ee858)
    * disables controller if enabled (b3f03b2)
    * don't zero out buf before ufs read (2ef6b8d)
    * don't zero out the write buffer (cd3ea90)
    * fix cache maintenance issues (38a5ecb)
    * move nutrs assignment to ufs_init (0956319)
    * read and write attribute based on spec (a475518)
- **Arm**
    * **GIC**
        · **GICv3**
            · fix iroute value wrong issue (65bc2d2)
    * **TZC**
        · **TZC-400**
            · correct message with filter (bdc88d2)
- **Marvell**
    * **COMPHY**
        · change reg_set() / reg_set16() to update semantics (95c26d6)
        · **Armada 3700**
            · drop MODE_REFDIV constant (9fdecc7)
            · fix comment about COMPHY status register (4bcfd8c)
            · fix comments about selector register values (71183ef)
            · fix Generation Setting registers names (e5a2aac)
            · fix PIN_PU_IVREF register name (c9f138e)
            · fix reference clock selection value names (6ba97f8)
            · fix SerDes frequency register value name (bdcf44f)
            · use reg_set() according to update semantics (4d01bfe)
    * **Armada**
        · **A3K**

- **A3720**
  - configure UART after TX FIFO reset (15546db)
  - do external reset during initialization (0ee80f3)
- **NXP**
  - ddr: corrects mapping of HNFs nodes (e3a2349)
  - **QSPI**
    - fix include path for QSPI driver (ae95b17)
  - **NXP Crypto**
    - refine code to avoid hang issue for some of toolchain (fa7fdfa)
  - **DDR**
    - fix coverity issue (f713e59)
- **ST**
  - **Clock**
    - check _clk_stm32_get_parent return (b8eab51)
    - correct stm32_clk_parse_fdt_by_name (7417cda)
    - correct types in error messages (44fb470)
    - initialize pllcfg table (175758b)
    - print enums as unsigned (9fa9a0c)
  - **DDR**
    - add missing debug.h (15ca2c5)
    - correct DDR warnings (a078134)
  - **FMC**
  - fix type in message (afcdc9d)
  - **SDMMC2**
    - check regulator enable/disable return (d50e7a7)
    - correct cmd_idx type in messages (bc1c98a)
  - **ST PMIC**
    - add static const to pmic_ops (57e6018)
    - correct verbose message (47065ff)
  - **SPI**
    - always check SR_TCF flags in stm32_qspi_wait_cmd() (55de583)
    - remove SR_BUSY bit check before sending command (5993b91)

        * **UART**
            · correctly fill BRR register (af7775a)
    - **USB**
        * correct type in message (bd9cd63)
- **Miscellaneous**
    - **AArch64**
        * fix encodings for MPAMVPM* registers (e926558)
    - **FDTs**
        * **STM32MP1**
            · correct memory mapping for STM32MP13 (99605fb)
            · remove mmc1 alias if not needed (a0e9724)
    - **PIE**
        * align fixup_gdt_reloc() for aarch64 (5ecde2a)
        * do not skip **RW_END** address during relocation (4f1a658)
    - **Security**
        * apply SMCCC_ARCH_WORKAROUND_3 to A73/A75/A72/A57 (9b2510b)
        * loop workaround for CVE-2022-23960 for Cortex-A76 (a10a5cb)
        * report CVE 2022 23960 missing for aarch32 A57 and A72 (2e5d7a4)
        * update Cortex-A15 CPU lib files for CVE-2022-23960 (187a617)
        * workaround for CVE-2022-23960 (c2a1521)
        * workaround for CVE-2022-23960 (1fe4a9d)
        * workaround for CVE-2022-23960 for A76AE, A78AE, A78C (5f802c8)
        * workaround for CVE-2022-23960 for Cortex-A57, Cortex-A72 (be9121f)
        * workaround for CVE-2022-23960 for Cortex-X1 (e81e999)
- **Tools**
    - **NXP Tools**
        * fix create_pbl print log (31af441)
        * fix tool location path for byte_swape (a89412a)
    - **Firmware Image Package Tool**
        * avoid packing the zero size images in the FIP (ab556c9)
        * respect OPENSSL_DIR (0a956f8
    - **Secure Partition Tool**

    * add leading zeroes in UUID conversion (b06344a)
    * update Optee FF-A manifest (ca0fdbd)
  - **Certificate Creation Tool**
    * let distclean Makefile target remove the cert_create tool (e15591a)
- **Dependencies**
  - **commitlint**
    * change scope-case to lower-case (804e52e)

14.10 2.6.0 (2021-11-22)

14.10.1 ? BREAKING CHANGES

- **Architecture**
  - **Activity Monitors Extension (FEAT_AMU)**
    * The public AMU API has been reduced to enablement only to facilitate refactoring work. These APIs were not previously used.

      See: privatize unused AMU APIs (b4b726e)
    * The `PLAT_AMU_GROUP1_COUNTERS_MASK` platform definition has been removed. Platforms should specify per-core AMU counter masks via FCONF or a platform-specific mechanism going forward.

      See: remove `PLAT_AMU_GROUP1_COUNTERS_MASK` (6c8dda1)
- **Libraries**
  - **FCONF**
    * FCONF is no longer added to BL1 and BL2 automatically when the FCONF Makefile (`fconf.mk`) is included. When including this Makefile, consider whether you need to add `${FCONF_SOURCES}` and `${FCONF_DYN_SOURCES}` to `BL1_SOURCES` and `BL2_SOURCES`.

      See: clean up source collection (e04da4c)
- **Drivers**
  - **Arm**
    * **Ethos-N**
      · multi-device support

        See: multi-device support (1c65989)

14.10.2 New Features

- **Architecture**
  - **Activity Monitors Extension (FEAT_AMU)**
    - enable per-core AMU auxiliary counters (742ca23)
  - **Support for the `HCRX_EL2` register (FEAT_HCX)**
    - add build option to enable FEAT_HCX (cb4ec47)
  - **Scalable Matrix Extension (FEAT_SME)**
    - enable SME functionality (dc78e62)
  - **Scalable Vector Extension (FEAT_SVE)**
    - enable SVE for the secure world (0c5e7d1)
  - **System Register Trace Extensions (FEAT_ETMv4, FEAT_ETE and FEAT_ETEv1.1)**
    - enable trace system registers access from lower NS ELs (d4582d3)
    - initialize trap settings of trace system registers access (2031d61)
  - **Trace Buffer Extension (FEAT_TRBE)**
    - enable access to trace buffer control registers from lower NS EL (813524e)
    - initialize trap settings of trace buffer control registers access (40ff907)
  - **Self-hosted Trace Extension (FEAT_TRF)**
    - enable trace filter control register access from lower NS EL (8fcd3d9)
    - initialize trap settings of trace filter control registers access (5de20ec)
  - **RME**
    - add context management changes for FEAT_RME (c5ea4f8)
    - add ENABLE_RME build option and support for RMM image (5b18de0)
    - add GPT Library (1839012)
    - add Realm security state definition (4693ff7)
    - add register definitions and helper functions for FEAT_RME (81c272b)
    - add RMM dispatcher (RMMD) (77c2775)
    - add Test Realm Payload (TRP) (50a3056)
    - add xlat table library changes for FEAT_RME (3621823)
    - disable Watchdog for Arm platforms if FEAT_RME enabled (07e96d1)
    - run BL2 in root world when FEAT_RME is enabled (6c09af9)
- **Platforms**

- **Allwinner**
  - add R329 support (13bacd3)
- **Arm**
  - add FWU support in Arm platforms (2f1177b)
  - add GPT initialization code for Arm platforms (deb4b3a)
  - add GPT parser support (ef1daa4)
  - enable PIE when RESET_TO_SP_MIN=1 (7285fd5)
  - **FPGA**
    - add ITS autodetection (d7e39c4)
    - add kernel trampoline (de9fdb9)
    - determine GICR base by probing (93b785f)
    - query PL011 to learn system frequency (d850169)
    - support GICv4 images (c69f815)
    - write UART baud base clock frequency into DTB (422b44f)
  - **FVP**
    - enable external SP images in BL2 config (33993a3)
    - add memory map for FVP platform for FEAT_RME (c872072)
    - add RMM image support for FVP platform (9d870b7)
    - enable trace extension features by default (cd3f0ae)
    - pass Event Log addr and size from BL1 to BL2 (0500f44)
  - **FVP-R**
    - support for TB-R has been added
    - configure system registers to boot rich OS (28bbbf3)
  - **RD**
    - **RD-N2**
      - add support for variant 1 of rd-n2 platform (fe5d5bb)
      - add tzc master source ids for soc dma (3139270)
  - **SGI**
    - add CPU specific handler for Neoverse N2 (d932a58)
    - add CPU specific handler for Neoverse V1 (cbee43e)
    - increase max BL2 size (7186a29)
    - enable AMU for RD-V1-MC (e8b119e)

  - enable use of PSCI extended state ID format (7bd64c7)
  - introduce platform variant build option (cfe1506)

- **TC**
  - enable MPMM (c19a82b)
  - Enable SVE for both secure and non-secure world (10198ea)
  - populate HW_CONFIG in BL31 (34a87d7)
  - introduce TC1 platform (6ec0c65)
  - add DRAM2 to TZC non-secure region (76b4a6b)
  - add bootargs node (4a840f2)
  - add cpu capacity to provide scheduling information (309f593)
  - add Ivy partition (a19bd32)
  - add support for trusted services (ca93248)
  - update Matterhorn ELP DVFS clock index (a2f6294)
  - update mhuv2 dts node to align with upstream driver (63067ce)

- **Diphda**
  - adding the diphda platform (bf3ce99)
  - disabling non volatile counters in diphda (7f70cd2)
  - enabling stack protector for diphda (c7e4f1c)

- **Marvell**
  - introduce t9130_cex7_eval (d01139f)
  - **Armada**
    - **A8K**
      - allow overriding default paths (0b702af)

- **MediaTek**
  - enable software reset for CIRQ (b3b162f)
  - **MT8192**
    - add DFD control in SiP service (5183e63)
  - **MT8195**
    - add DFD control in SiP service (3b994a7)
    - add display port control in SiP service (7eb4223)
    - remove adsp event from wakeup source (c260b32)
    - add DCM driver (49d3bd8)

            - add EMI MPU basic drivers (75edd34)
            - add SPM suspend driver (859e346)
            - add support for PTP3 (0481896)
            - add vcore-dvfs support (d562130)
            - support MCUSYS off when system suspend (d336e09)

  - **NXP**
    - add build macro for BOOT_MODE validation checking (cd1280e)
    - add CCI and EPU address definition (6cad59c)
    - add EESR register definition (8bfb168)
    - add SecMon register definition for ch_3_2 (66f7884)
    - define common macro for ARM registers (35efe7a)
    - define default PSCI features if not defined (a204785)
    - define default SD buffer (4225ce8)
    - **i.MX**
      - **i.MX 8M**
        - add sdei support for i.MX8MN (ce2be32)
        - add sdei support for i.MX8MP (6b63125)
        - add SiP call for secondary boot (9ce232f)
        - add system_reset2 implementation (60a0dde)
        - **i.MX 8M Mini**
          - enlarge BL33 (U-boot) size in FIP (d53c9db)
        - **i.MX 8M Plus**
          - add imx8mp_private.h to the build (91566d6)
          - add in BL2 with FIP (75fbf55)
          - add initial definition to facilitate FIP layout (f696843)
          - enable Trusted Boot (a16ecd2)
    - **Layerscape**
      - add ls1028a soc and board support (52a1e9f)
      - **LX2**
        - add SUPPORTED_BOOT_MODE definition (28b3221)
        - **LS1028A**
          - add ls1028a soc support (9d250f0)

- **LS1028ARDB**
  - add ls1028ardb board support (34e2112)

- **QTI**
  - **SC7280**
    - add support for pmk7325 (b8a0511)
    - support for qti sc7280 plat (46ee50e)
- **Renesas**
  - **R-Car**
    - change process for Suspend To RAM (731aa26)
    - **R-Car 3**
      - add a DRAM size setting for M3N (f95d551)
      - add new board revision for Salvator-XS/H3ULCB (4379a3e)
      - add optional support for gzip-compressed BL33 (ddf2ca0)
      - add process of SSCG setting for R-Car D3 (14f0a08)
      - add process to back up X6 and X7 register's value (7d58aed)
      - add SYSCEXTMASK bit set/clear in scu_power_up (63a7a34)
      - apply ERRATA_A53_1530924 and ERRATA_A57_1319537 (2892fed)
      - change the memory map for OP-TEE (a4d821a)
      - emit RPC status to DT fragment if RPC unlocked (12c75c8)
      - keep RWDT enabled (8991086)
      - modify LifeC register setting for R-Car D3 (5460f82)
      - modify operation register from SYSCISR to SYSCISCR (d10f876)
      - modify SWDT counter setting for R-Car D3 (053c134)
      - remove access to RMSTPCRn registers in R-Car D3 (71f2239)
      - update DDR setting for R-Car D3 (042d710)
      - update IPL and Secure Monitor Rev.3.0.0 (c5f5bb1)
      - use PRR cut to determine DRAM size on M3 (42ffd27)
- **ST**
  - add a new DDR firewall management (4584e01)
  - add a USB DFU stack (efbd65f)
  - add helper to save boot interface (7e87ba2)
  - add STM32CubeProgrammer support on USB (afad521)

      * add STM32MP_EMMC_BOOT option (214c8a8)
      * create new helper for DT access (ea97bbf)
      * implement platform functions for SMCCC_ARCH_SOC_ID (3d20178)
      * improve FIP image loading from MMC (18b415b)
      * manage io_policies with FCONF (d5a84ee)
      * use FCONF to configure platform (29332bc)
      * use FIP to load images (1d204ee)
      * **ST32MP1**
        - add STM32MP_USB_PROGRAMMER target (fa92fef)
        - add USB DFU support for STM32MP1 (942f6be)
  - **Xilinx**
    * **Versal**
      - add support for SLS mitigation (302b4df)
    * **ZynqMP**
      - add support for runtime feature config (578f468)
      - sync IOCTL IDs (38c0b25)
      - add SDEI support (4143268)
      - add support for XCK26 silicon (7a30e08)
      - extend DT description by TF-A (0a8143d)
- **Bootloader Images**
  - import BL_NOBITS_{BASE,END} when defined (9aedca0)
- **Services**
  - **FF-A**
    * adding notifications SMC IDs (fc3f480)
    * change manifest messaging method (bb320db)
    * feature retrieval through FFA_FEATURES call (96b71eb)
    * update FF-A version to v1.1 (e1c732d)
    * add Ivy partition to tb fw config (1bc02c2)
    * add support for FFA_SPM_ID_GET (70c121a)
    * route secure interrupts to SPMC (8cb99c3)
- **Libraries**
  - **CPU Support**

    - add support for Hayes CPU (7bd8dfb)
    - add support for Hunter CPU (fb9e5f7)
    - add support for Demeter CPU (f4616ef)
    - workaround for Cortex A78 AE erratum 1941500 (47d6f5f)
    - workaround for Cortex A78 AE erratum 1951502 (8913047)
  - **MPMM**
    - add support for MPMM (6812078)
  - **OP-TEE**
    - introduce optee_header_is_valid() (b84a850)
  - **PSCI**
    - require validate_power_state to expose CPU_SUSPEND (a1d5ac6)
  - **SMCCC**
    - add bit definition for SMCCC_ARCH_SOC_ID (96b0596)
- **Drivers**
  - **FWU**
    - add FWU metadata header and build options (5357f83)
    - add FWU driver (0ec3ac6)
    - avoid booting with an alternate boot source (4b48f7b)
    - avoid NV counter upgrade in trial run state (c0bfc88)
    - initialize FWU driver in BL2 (396b339)
    - introduce FWU platform-specific functions declarations (efb2ced)
  - **I/O**
    - **MTD**
      - offset management for FIP usage (9a9ea82)
  - **Measured Boot**
    - add documentation to build and run PoC (a125c55)
    - move init and teardown functions to platform layer (47bf3ac)
    - image hash measurement and recording in BL1 (48ba034)
    - update tb_fw_config with event log properties (e742bcd)
  - **MMC**
    - boot partition read support (5014b52)
  - **MTD**

- **NAND**
  - count bad blocks before a given offset (bc3eebb)
- **SCMI**
  - add power domain protocol (7e4833c)
- **Arm**
  - **Ethos-N**
    - multi-device support (1c65989)
  - **GIC**
    - **GICv3**
      - detect GICv4 feature at runtime (858f40e)
      - introduce GIC component identification (73a643e)
      - multichip: detect GIC-700 at runtime (feb7081)
      - **GIC-600AE**
        - introduce support for Fault Management Unit (2c248ad)
  - **TZC**
    - **TZC-400**
      - update filters by region (ce7ef9d)
- **MediaTek**
  - **APU**
    - add mt8192 APU device apc driver (f46e1f1)
    - add mt8192 APU iommap regions (2671f31)
    - add mt8192 APU SiP call support (ca4c0c2)
    - setup mt8192 APU_S_S_4 and APU_S_S_5 permission (77b6801)
  - **EMI MPU**
    - add MPU support for DSP (6c4973b)
- **NXP**
  - **DCFG**
    - define RSTCR_RESET_REQ (6c5d140)
  - **FLEXSPI**
    - add MT35XU02G flash info (a4f5015)
- **Renesas**
  - **R-Car3**

    - add extra offset if booting B-side (993d809)
    - add function to judge a DDR rank (726050b)

  - **ST**
    - manage boot part in io_mmc (f3d2750)
    - **USB**
      - add device driver for STM32MP1 (9a138eb)
  - **USB**
    - add a USB device stack (859bfd8)

- **Miscellaneous**
  - **Debug**
    - add new macro ERROR_NL() to print just a newline (fd1360a)
  - **CRC32**
    - **Hardware CRC32**
      - add support for HW computed CRC (a1cedad)
    - **Software CRC32**
      - add software CRC32 support (f216937)
  - **DT Bindings**
    - add STM32MP1 TZC400 bindings (43de546)
  - **FDT Wrappers**
    - add CPU enumeration utility function (2d9ea36)
  - **FDTs**
    - add for_each_compatible_node macro (ff76614)
    - introduce wrapper function to read DT UUIDs (d13dbb6)
    - add firewall regions into STM32MP1 DT (86b43c5)
    - add IO policies for STM32MP1 (21e002f)
    - add STM32MP1 fw-config DT files (d9e0586)
    - **STM32MP1**
      - align DT with latest kernel (e8a953a)
      - delete nodes for non-used boot devices (4357db5)
  - **NXP**
    - **OCRAM**
      - add driver for OCRAM initialization (10b1e13)

    * **PSCI**
        - define CPUECTLR_TIMER_2TICKS (3a2cc2e)

- **Dependencies**
  - **libfdt**
    * also allow changing base address (4d585fe)

14.10.3 Resolved Issues

- **Architecture**
- **Platforms**
  - print newline before fatal abort error message (a5fea81)
  - **Allwinner**
    * delay after enabling CPU power (86a7429)
  - **Arm**
    * correct UUID strings in FVP DT (748bdd1)
    * fix a VERBOSE trace (5869ebd)
    * remove unused memory node (be42c4b)
    * **FPGA**
      - allow build after MAKE_* changes (9d38a3e)
      - avoid re-linking from executable ELF file (a67ac76)
      - Change PL011 UART IRQ (195381a)
      - limit BL31 memory usage (d457230)
      - reserve BL31 memory (13e16fe)
      - streamline generated axf file (9177e4f)
      - enable AMU extension (d810e30)
      - increase initrd size (c3ce73b)
    * **FVP**
      - fix fvp_cpu_standby() function (3202ce8)
      - spmc optee manifest remove SMC allowlist (183725b)
      - allow changing the kernel DTB load address (672d669)
      - bump BL2 stack size (d22f1d3)
      - provide boot files via semihosting (749d0fa)
      - OP-TEE SP manifest per latest SPMC changes (b7bc51a)

        - mock support for CCA NV ctr (7423e5e)
    - **FVP-R**
        - fix compilation error in release mode (7d96e79)
    - **Morello**
        - initialise CNTFRQ in Non Secure CNTBaseN (7f2d23d)
    - **TC**
        - enable AMU extension (b5863ca)
        - change UUID to string format (1c19536)
        - remove "arm,psci" from psci node (814646b)
        - remove ffa and optee device tree node (f1b44a9)
        - set cactus-tertiary vcpu count to 1 (05f667f)
    - **SGI**
        - avoid redefinition of 'efi_guid' structure (f34322c)
- **Marvell**
    - Check the required libraries before building doimage (dd47809)
    - **Armada**
        - select correct pcie reference clock source (371648e)
        - fix MSS loader for A8K family (dceac43)
        - **A3K**
            - disable HANDLE_EA_EL3_FIRST by default (3017e93)
            - enable workaround for erratum 1530924 (975563d)
            - Fix building uart-images.tgz.bin archive (d3f8db0)
            - Fix check for external dependences (2baf503)
            - fix printing info messages on output (9f6d154)
            - update information about PCIe abort hack (068fe91)
            - Remove encryption password (076374c)
        - **A8K**
            - Add missing build dependency for BLE target (04738e6)
            - Correctly set include directories for individual targets (559ab2d)
            - Require that MV_DDR_PATH is correctly set (528dafc)
            - fix number of CPU power switches. (5cf6faf)
- **MediaTek**

* **MT8183**
  - fix out-of-bound access (420c26b)
* **MT8195**
  - use correct print format for uint64_t (964ee4e)
  - fix error setting for SPM (1f81ccc)
  - extend MMU region size (9ff8b8c)
  - fix coverity fail (85e4d14)

- **NXP**
  * **i.MX**
    - do not keep mmc_device_info in stack (99d37c8)
    - **i.MX 8M**
      - **i.MX 8M Mini**
        - fix FTBFS on SPD=opteed (10bfc77)
  * **Layerscape**
    - **LX2**
      - **LS1028A**
        - define endianness of scfg and gpio (2475f63)
        - fix compile error when enable fuse provision (a0da9c4)
- **QEMU**
  * (NS_DRAM0_BASE + NS_DRAM0_SIZE) ADDR overflow 32bit (325716c)
  * reboot/shutdown with low to high gpio (bd2ad12)
- **QTI**
  * **SC1780**
    - qti smc addition (cc35a37)
- **Raspberry Pi**
  * **Raspberry Pi 4**
    - drop /memreserve/ region (5d2793a)
- **Renesas**
  * **R-Car**
    - change process that copy code to system ram (49593cc)
    - fix cache maintenance process of reading cert header (c77ab18)
    - fix to load image when option BL2_DCACHE_ENABLE is enabled (d2ece8d)

- R-Car 3
  - fix disabling MFIS write protection for R-Car D3 (a8c0c3e)
  - fix eMMC boot support for R-Car D3 (77ab366)
  - fix source file to make about GICv2 (fb3406b)
  - fix version judgment for R-Car D3 (c3d192b)
  - generate two memory nodes for larger than 2 GiB channel 0 (21924f2)

- **Rockchip**
  - **RK3399**
    - correct LPDDR4 resume sequence (2c4b0c0)
    - fix dram section placement (f943b7c)
- **Socionext**
  - **Synquacer**
    - update scmi power domain off handling (f7f5d2c)
- **ST**
  - add STM32IMAGE_SRC (f223505)
  - add UART reset in crash console init (b38e2ed)
  - apply security at the end of BL2 (99080bd)
  - correct BSEC error code management (72c7884)
  - correct IO compensation disabling (c2d18ca)
  - correct signedness comparison issue (5657dec)
  - improve DDR get size function (91ffc1d)
  - only check header major when booting (8ce8918)
  - panic if boot interface is wrong (71693a6)
  - remove double space (306dcd6)
  - **ST32MP1**
    - add bl prefix for internal linker script (7684ddd)
- **Xilinx**
  - **Versal**
    - correct IPI buffer offset (e1e5b13)
    - use sync method for blocking calls (fa58171)
  - **ZynqMP**
    - use sync method for blocking calls (c063c5a)

- **Services**
  - drop warning on unimplemented calls (67fad51)
  - **RME**
    - fixes a shift by 64 bits bug in the RME GPT library (322b344)
  - **SPM**
    - do not compile if SVE/SME is enabled (4333f95)
    - error macro to use correct print format (0c23e6f)
    - revert workaround hafnium as hypervisor (3221fce)
    - fixing coverity issue for SPM Core. (f7fb0bf)
- **Libraries**
  - **LIBC**
    - use long for 64-bit types on aarch64 (4ce3e99)
  - **CPU Support**
    - correct Demeter CPU name (4cb576a)
    - workaround for Cortex A78 erratum 2242635 (1ea9190)
    - workaround for Cortex-A710 erratum 2058056 (744bdbf)
    - workaround for Neoverse V1 erratum 2216392 (4c8fe6b)
    - workaround for Neoverse-N2 erratum 2138953 (ef8f0c5)
    - workaround for Neoverse-N2 erratum 2138958 (c948185)
    - workaround for Neoverse-N2 erratum 2242400 (603806d)
    - workaround for Neoverse-N2 erratum 2242415 (5819e23)
    - workaround for Neoverse-N2 erratum 2280757 (0d2d999)
    - rename Matterhorn, Matterhorn ELP, and Klein CPUs (c6ac4df)
  - **EL3 Runtime**
    - correct CASSERT for pauth (b4f8d44)
    - fix SVE and AMU extension enablement flags (68ac5ed)
    - random typos in tf-a code base (2e61d68)
    - Remove save/restore of EL2 timer registers (a7cf274)
  - **OP-TEE**
    - correct signedness comparison (21d2be8)
  - **GPT**
    - add necessary barriers and remove cache clean (77612b9)

        - use correct print format for uint64_t (2461bd3)

    - **Translation Tables**

        - remove always true check in assert (74d720a)

- **Drivers**

    - **Authentication**

        - avoid NV counter upgrade without certificate validation (a2a5a94)

        - **CryptoCell-713**

            - fix a build failure with CC-713 library (e5fbee5)

    - **MTD**

        - fix MISRA issues and logic improvement (5130ad1)

        - macronix quad enable bit issue (c332740)

        - **NAND**

            - **SPI NAND**

                - check correct manufacturer id (4490b79)

                - check that parameters have been set (bc453ab)

    - **SCMI**

        - entry: add weak functions (b3c8fd5)

        - smt: fix build for aarch64 (0e223c6)

        - mention "SCMI" in driver initialisation message (e0baae7)

        - relax requirement for exact protocol version (125868c)

    - **UFS**

        - add reset before DME_LINKSTARTUP (905635d)

    - **Arm**

        - **GIC**

            - **GICv3**

                - add dsb in both disable and enable function of gicv3_cpuif (5a5e0aa)

            - **GIC-600AE**

        - fix timeout calculation (7f322f2)

        - **TZC**

            - **TZC-400**

                - never disable filter 0 (ef378d3)

    - **Marvell**

* **COMPHY**
  * fix name of 3.125G SerDes mode (a669983)
  * **Armada 3700**
    * configure phy selector also for PCIe (0f3a122)
    * fix address overflow (c074f70)
    * handle failures in power functions (49b664e)
  * **CP110**
    * fix error code in pcie power on (c0a909c)
* **Armada**
  * **A3K**
    * **A3720**
      * fix configuring UART clock (b9185c7)
      * fix UART clock rate value and divisor calculation (66a7752)
      * fix UART parent clock rate determination (5a91c43)

- **MediaTek**
  * **PMIC Wrapper**
    * update idle flow (9ed4e6f)
  * **MT8192**
    * **SPM**
      * add missing bit define for debug purpose (310c3a2)
- **NXP**
  * **FLEXSPI**
    * fix warm boot wait time for MT35XU512A (1ff7e46)
  * **SCFG**
    * fix endianness checking (fb90cfd)
  * **SFP**
    * fix compile warning (3239a17)
- **Renesas**
  * **R-Car3**
    * console: fix a return value of console_rcar_init (bb273e3)
    * ddr: update DDR setting for H3, M3, M3N (ec767c1)
    * emmc: remove CPG_CPGWPR redefinition (36d5645)

- fix CPG registers redefinition (0dae56b)
- i2c_dvfs: fix I2C operation (b757d3a)

- **ST**
  - **Clock**
    - use correct return value (8f97c4f)
    - correctly manage RTC clock source (1550909)
    - fix MCU/AXI parent clock (b8fe48b)
    - fix MPU clock rate (602ae2f)
    - fix RTC clock rating (cbd2e8a)
    - keep RTC clock always on (5b111c7)
    - keep RTCAPB clock always on (373f06b)
    - set other clocks as always on (bf39318)
  - **I/O**
    - **STM32 Image**
      - invalidate cache on local buf (a5bcf82)
      - uninitialized variable warning (c1d732d)
  - **ST PMIC**
    - initialize i2c_state (4282284)
    - missing error check (a4bcfe9)
  - **STPMIC1**
    - fix power switches activation (0161991)
    - update error cases return (ed6a852)
  - **UART**
    - **STM32 Console**
      - do not skip init for crash console (49c7f0c)
- **USB**
  - add a optional ops get_other_speed_config_desc (216c122)
  - fix Null pointer dereferences in usb_core_set_config (0cb9870)
  - remove deadcode when USBD_EP_NB = 1 (7ca4928)
  - remove unnecessary cast (025f5ef)

- **Miscellaneous**
  - use correct printf format for uint64_t (4ef449c)

- **DT Bindings**
  - fix static checks (0861fcd)
- **FDTs**
  - avoid output on missing DT property (49e789e)
  - fix OOB write in uuid parsing function (d0d6424)
  - **Morello**
    - fix scmi clock specifier to cluster mappings (387a906)
  - **STM32MP1**
    - correct copyright dates (8d26029)
    - set ETH clock on PLL4P on ST boards (3e881a8)
    - update PLL nodes for ED1/EV1 boards (cdbbb9f)
    - use 'kHz' as kilohertz abbreviation (4955d08)
- **PIE**
  - invalidate data cache in the entire image range if PIE is enabled (596d20d)
- **Security**
  - Set MDCR_EL3.MCCD bit (12f6c06)
- **SDEI**
  - fix assert while kdump issue (d39db26)
  - print event number in hex format (6b94356)
  - set SPSR for SDEI based on TakeException (37596fc)

- **Documentation**
  - fix TF-A v2.6 release date in the release information page (c90fa47)
  - fix `FF-A` substitution (a61940c)
  - fix typos in v2.5 release documentation (481c7b6)
  - remove "experimental" tag for stable features (700e768)
  - **Contribution Guidelines**
    - fix formatting for code snippet (d0bbe81)
- **Build System**
  - use space in WARNINGS list (34b508b)
  - **Git Hooks**
    - downgrade `package-lock.json` version (7434b65)
- **Tools**

- **STM32 Image**
  - improve the tool (8d0036d)
- **SPTOOL**
  - SP UUID little to big endian in TF-A build (dcdbcdd)
- **DOIMAGE**
  - Fix doimage syntax breaking secure mode build (6d55ef1)

- **Dependencies**
  - **checkpatch**
    - do not check merge commits (77a0a7f)

14.11 2.5.0 (2021-05-17)

14.11.1 New Features

- Architecture support
  - Added support for speculation barrier(`FEAT_SB`) for non-Armv8.5 platforms starting from Armv8.0
  - Added support for Activity Monitors Extension version 1.1(`FEAT_AMUv1p1`)
  - Added helper functions for Random number generator(`FEAT_RNG`) registers
  - Added support for Armv8.6 Multi-threaded PMU extensions (`FEAT_MTPMU`)
  - Added support for MTE Asymmetric Fault Handling extensions(`FEAT_MTE3`)
  - Added support for Privileged Access Never extensions(`FEAT_PANx`)
- Bootloader images
  - Added PIE support for AArch32 builds
  - Enable Trusted Random Number Generator service for BL32(sp_min)
- Build System
  - Added build option for Arm Feature Modifiers
- Drivers
  - Added support for interrupts in TZC-400 driver
  - Broadcom
    - Added support for I2C, MDIO and USB drivers
  - Marvell
    - Added support for secure read/write of dfc register-set

    * Added support for thermal sensor driver
    * Implement a3700_core_getc API in console driver
    * Added rx training on 10G port
  - Marvell Mochi
    * Added support for cn913x in PCIe mode
  - Marvell Armada A8K
    * Added support for TRNG-IP-76 driver and accessing RNG register
  - Mediatek MT8192
    * Added support for following drivers
      - MPU configuration for SCP/PCIe
      - SPM suspend
      - Vcore DVFS
      - LPM
      - PTP3
      - UART save and restore
      - Power-off
      - PMIC
      - CPU hotplug and MCDI support
      - SPMC
      - MPU
  - Mediatek MT8195
    * Added support for following drivers
      - GPIO, NCDI, SPMC drivers
      - Power-off
      - CPU hotplug, reboot and MCDI
      - Delay timer and sys timer
      - GIC
  - NXP
    * Added support for
      - non-volatile storage API
      - chain of trust and trusted board boot using two modes: MBEDTLS and CSF
      - fip-handler necessary for DDR initialization

        - SMMU and console drivers
        - crypto hardware accelerator driver
        - following drivers: SD, EMMC, QSPI, FLEXSPI, GPIO, GIC, CSU, PMU, DDR
        - NXP Security Monitor and SFP driver
        - interconnect config APIs using ARM CCN-CCI driver
        - TZC APIs to configure DDR region
        - generic timer driver
        - Device configuration driver
  - IMX
    - Added support for image loading and io-storage driver for TBBR fip booting
  - Renesas
    - Added support for PFC and EMMC driver
    - RZ Family:
      - G2N, G2E and G2H SoCs
        - Added support for watchdog, QoS, PFC and DRAM initialization
    - RZG Family:
      - G2M
        - Added support for QoS and DRAM initialization
  - Xilinx
    - Added JTAG DCC support for Versal and ZynqMP SoC family.
- Libraries
  - C standard library
    - Added support to print `%` in `snprintf()` and `printf()` APIs
    - Added support for strtoull, strtoll, strtoul, strtol APIs from FreeBSD project
  - CPU support
    - Added support for
      - Cortex_A78C CPU
      - Makalu ELP CPU
      - Makalu CPU
      - Matterhorn ELP CPU
      - Neoverse-N2 CPU
  - CPU Errata

    * Arm Cortex-A76: Added workaround for erratum 1946160
    * Arm Cortex-A77: Added workaround for erratum 1946167
    * Arm Cortex-A78: Added workaround for erratum 1941498 and 1951500
    * Arm Neoverse-N1: Added workaround for erratum 1946160
  - Flattened device tree(libfdt)
    * Added support for wrapper function to read UUIDs in string format from dtb
- Platforms
  - Added support for MediaTek MT8195
  - Added support for Arm RD-N2 board
  - Allwinner
    * Added support for H616 SoC
  - Arm
    * Added support for GPT parser
    * Protect GICR frames for fused/unused cores
  - Arm Morello
    * Added VirtIO network device to Morello FVP fdts
  - Arm RD-N2
    * Added support for variant 1 of RD-N2 platform
    * Enable AMU support
  - Arm RD-V1
    * Enable AMU support
  - Arm SGI
    * Added support for platform variant build option
  - Arm TC0
    * Added Matterhorn ELP CPU support
    * Added support for opteed
  - Arm Juno
    * Added support to use hw_config in BL31
    * Use TRNG entropy source for SMCCC TRNG interface
    * Condition Juno entropy source with CRC instructions
  - Marvell Mochi
    * Added support for detection of secure mode

- Marvell ARMADA
  - Added support for new compile option A3720_DB_PM_WAKEUP_SRC
  - Added support doing system reset via CM3 secure coprocessor
  - Made several makefile enhancements required to build WTMI_MULTI_IMG and TIMDDRTOOL
  - Added support for building DOIMAGETOOL tool
  - Added new target mrvl_bootimage
- Mediatek MT8192
  - Added support for rtc power off sequence
- Mediatek MT8195
  - Added support for SiP service
- STM32MP1
  - Added support for
    - Seeed ODYSSEY SoM and board
    - SDMMC2 and I2C2 pins in pinctrl
    - I2C2 peripheral in DTS
    - PIE for BL32
    - TZC-400 interrupt managament
    - Linux Automation MC-1 board
- Renesas RZG
  - Added support for identifying EK874 RZ/G2E board
  - Added support for identifying HopeRun HiHope RZ/G2H and RZ/G2H boards
- Rockchip
  - Added support for stack protector
- QEMU
  - Added support for `max` CPU
  - Added Cortex-A72 support to `virt` platform
  - Enabled trigger reboot from secure pl061
- QEMU SBSA
  - Added support for sbsa-ref Embedded Controller
- NXP
  - Added support for warm reset to retain ddr content

        * Added support for image loader necessary for loading fip image
        * lx2160a SoC Family
            - Added support for
                - new platform lx2160a-aqds
                - new platform lx2160a-rdb
                - new platform lx2162a-aqds
                - errata handling
    - IMX imx8mm
        * Added support for trusted board boot
    - TI K3
        * Added support for lite device board
        * Enabled Cortex-A72 erratum 1319367
        * Enabled Cortex-A53 erratum 1530924
    - Xilinx ZynqMP
        * Added support for PS and system reset on WDT restart
        * Added support for error management
        * Enable support for log messages necessary for debug
        * Added support for PM API SMC call for efuse and register access
- Processes
    - Introduced process for platform deprecation
    - Added documentation for TF-A threat model
    - Provided a copy of the MIT license to comply with the license requirements of the arm-gic.h source file (originating from the Linux kernel project and re-distributed in TF-A).
- Services
    - Added support for TRNG firmware interface service
    - Arm
        * Added SiP service to configure Ethos-N NPU
    - SPMC
        * Added documentation for SPM(Hafnium) SMMUv3 driver
    - SPMD
        * Added support for
            - FFA_INTERRUPT forwading ABI

        · FFA_SECONDARY_EP_REGISTER ABI
        · FF-A v1.0 boot time power management, SPMC secondary core boot and early run-time power management

- Tools
  - FIPTool
    - Added mechanism to allow platform specific image UUID
  - git hooks
    - Added support for conventional commits through commitlint hook, commitizen hook and husky configuration files.
  - NXP tool
    - Added support for a tool that creates pbl file from BL2
  - Renesas RZ/G2
    - Added tool support for creating bootparam and cert_header images
  - CertCreate
    - Added support for platform-defined certificates, keys, and extensions using the platform's makefile
  - shared tools
    - Added EFI_GUID representation to uuid helper data structure

14.11.2 Changed

- Common components
  - Print newline after hex address in aarch64 el3_panic function
  - Use proper `#address-cells` and `#size-cells` for reserved-memory in dtbs
- Drivers
  - Move SCMI driver from ST platform directory and make it common to all platforms
  - Arm GICv3
    - Shift eSPI register offset in GICD_OFFSET_64()
    - Use mpidr to probe GICR for current CPU
  - Arm TZC-400
    - Adjust filter tag if it set to FILTER_BIT_ALL
  - Cadence
    - Enhance UART driver APIs to put characters to fifo
  - Mediatek MT8192

        - Move timer driver to common folder
        - Enhanced sys_cirq driver to add more IC services
    - Renesas
        - Move ddr and delay driver to common directory
    - Renesas rcar
        - Treat log as device memory in console driver
    - Renesas RZ Family:
        - G2N and G2H SoCs
            - Select MMC_CH1 for eMMC channel
    - Marvell
        - Added support for checking if TRNG unit is present
    - Marvell A3K
        - Set TXDCLK_2X_SEL bit during PCIe initialization
        - Set mask parameter for every reg_set call
    - Marvell Mochi
        - Added missing stream IDs configurations
    - MbedTLS
        - Migrated to Mbed TLS v2.26.0
    - IMX imx8mp
        - Change the bl31 physical load address
    - QEMU SBSA
        - Enable secure variable storage
    - SCMI
        - Update power domain protocol version to 2.0
    - STM32
        - Remove dead code from nand FMC driver
- Libraries
    - C Standard Library
        - Use macros to reduce duplicated code between snprintf and printf
    - CPU support
        - Sanity check pointers before use in AArch32 builds
        - Arm Cortex-A78

- Remove rainier cpu workaround for errata 1542319
- Arm Makalu ELP
  - Added "_arm" suffix to Makalu ELP CPU lib

- Miscellaneous
  - Editorconfig
    - set max line length to 100
- Platforms
  - Allwinner
    - Added reserved-memory node to DT
    - Express memmap more dynamically
    - Move SEPARATE_NOBITS_REGION to platforms
    - Limit FDT checks to reduce code size
    - Use CPUIDLE hardware when available
    - Allow conditional compilation of SCPI and native PSCI ops
    - Always use a 3MHz RSB bus clock
    - Enable workaround for Cortex-A53 erratum 1530924
    - Fixed non-default PRELOADED_BL33_BASE
    - Leave CPU power alone during BL31 setup
    - Added several psci hooks enhancements to improve system shutdown/reset sequence
    - Return the PMIC to I2C mode after use
    - Separate code to power off self and other CPUs
    - Split native and SCPI-based PSCI implementations
  - Allwinner H6
    - Added R_PRCM security setup for H6 board
    - Added SPC security setup for H6 board
    - Use RSB for the PMIC connection on H6
  - Arm
    - Store UUID as a string, rather than ints
    - Replace FIP base and size macro with a generic name
    - Move compile time switch from source to dt file
    - Don't provide NT_FW_CONFIG when booting hafnium
    - Do not setup 'disabled' regulator

    * Increase SP max size
    * Remove false dependency of ARM_LINUX_KERNEL_AS_BL33 on RESET_TO_BL31 and allow it to be enabled independently

  - Arm FVP
    * Do not map GIC region in BL1 and BL2

  - Arm Juno
    * Refactor juno_getentropy() to return 64 bits on each call

  - Arm Morello
    * Remove "virtio-rng" from Morello FVP
    * Enable virtIO P9 device for Morello fvp

  - Arm RDV1
    * Allow all PSCI callbacks on RD-V1
    * Rename rddaniel to rdv1

  - Arm RDV1MC
    * Rename rddanielxlr to rdv1mc
    * Initialize TZC-400 controllers

  - Arm TC0
    * Updated GICR base address
    * Use scmi_dvfs clock index 1 for cores 4-7 through fdt
    * Added reserved-memory node for OP-TEE fdts
    * Enabled Theodul DSU in TC platform
    * OP-TEE as S-EL1 SP with SPMC at S-EL2
    * Update Matterhorm ELP DVFS clock index

  - Arm SGI
    * Allow access to TZC controller on all chips
    * Define memory regions for multi-chip platforms
    * Allow access to nor2 flash and system registers from S-EL0
    * Define default list of memory regions for DMC-620 TZC
    * Improve macros defining cper buffer memory region
    * Refactor DMC-620 error handling SMC function id
    * Refactor SDEI specific macros
    * Added platform id value for RDN2 platform

    * Refactored header file inclusions and inclusion of memory mapping

- Arm RDN2
    * Allow usage of secure partitions on RDN2 platform
    * Update GIC redistributor and TZC base address
- Arm SGM775
    * Deprecate Arm sgm775 FVP platform
- Marvell
    * Increase TX FIFO EMPTY timeout from 2ms to 3ms
    * Update delay code to be compatible with 1200 MHz CPU
- Marvell ARMADA
    * Postpone MSS CPU startup to BL31 stage
    * Allow builds without MSS support
    * Use MSS SRAM in secure mode
    * Added missing FORCE, .PHONY and clean targets
    * Cleanup MSS SRAM if used for copy
    * Move definition of mrvl_flash target to common marvell_common.mk file
    * Show informative build messages and blank lines
- Marvell ARMADA A3K
    * Added a new target mrvl_uart which builds UART image
    * Added checks that WTP, MV_DDR_PATH and CRYPTOPP_PATH are correctly defined
    * Allow use of the system Crypto++ library
    * Build $(WTMI_ENC_IMG) in $(BUILD_PLAT) directory
    * Build intermediate files in $(BUILD_PLAT) directory
    * Build UART image files directly in $(BUILD_UART) subdirectory
    * Correctly set DDR_TOPOLOGY and CLOCKSPRESET for WTMI
    * Do not use 'echo -e' in Makefile
    * Improve 4GB DRAM usage from 3.375 GB to 3.75 GB
    * Remove unused variable WTMI_SYSINIT_IMG from Makefile
    * Simplify check if WTP variable is defined
    * Split building $(WTMI_MULTI_IMG) and $(TIMDDRTOOL)
- Marvell ARMADA A8K
    * Allow CP1/CP2 mapping at BLE stage

- Mediatek MT8183
  * Added timer V20 compensation
- Nvidia Tegra
  * Rename SMC API
- TI K3
  * Make plat_get_syscnt_freq2 helper check CNT_FID0 register
  * Fill non-message data fields in sec_proxy with 0x0
  * Update ti_sci_msg_req_reboot ABI to include domain
  * Enable USE_COHERENT_MEM only for the generic board
  * Explicitly map SEC_SRAM_BASE to 0x0
  * Use BL31_SIZE instead of computing
  * Define the correct number of max table entries and increase SRAM size to account for additional table
- Raspberry Pi4
  * Switch to gicv2.mk and GICV2_SOURCES
- Renesas
  * Move headers and assembly files to common folder
- Renesas rzg
  * Added device tree memory node enhancements
- Rockchip
  * Switch to using common gicv3.mk
- STM32MP1
  * Set BL sizes regardless of flags
- QEMU
  * Include gicv2.mk for compiling GICv2 source files
  * Change DEVICE2 definition for MMU
  * Added helper to calculate the position shift from MPIDR
- QEMU SBSA
  * Include libraries for Cortex-A72
  * Increase SHARED_RAM_SIZE
  * Addes support in spm_mm for upto 512 cores
  * Added support for topology handling

- QTI
  - Mandate SMC implementation
- Xilinx
  - Rename the IPI CRC checksum macro
  - Use fno-jump-tables flag in CPPFLAGS
- Xilinx versal
  - Added the IPI CRC checksum macro support
  - Mark IPI calls secure/non-secure
  - Enable sgi to communicate with linux using IPI
  - Remove Cortex-A53 compilation
- Xilinx ZynqMP
  - Configure counter frequency during initialization
  - Filter errors related to clock gate permissions
  - Implement pinctrl request/release EEMI API
  - Reimplement pinctrl get/set config parameter EEMI API calls
  - Reimplement pinctrl set/get function EEMI API
  - Update error codes to match Linux and PMU Firmware
  - Update PM version and support PM version check
  - Update return type in query functions
  - Added missing ids for 43/46/47dr devices
  - Checked for DLL status before doing reset
  - Disable ITAPDLYENA bit for zero ITAP delay
  - Include GICv2 makefile
  - Remove the custom crash implementation

• Services
- SPMD
  - Lock the g_spmd_pm structure
  - Declare third cactus instance as UP SP
  - Provide number of vCPUs and VM size for first SP
  - Remove `chosen` node from SPMC manifests
  - Move OP-TEE SP manifest DTS to FVP platform
  - Update OP-TEE SP manifest with device-regions node

    * Remove device-memory node from SPMC manifests

  - SPM_MM

    * Use sp_boot_info to set SP context

  - SDEI

    * Updata the affinity of shared event

- Tools

  - FIPtool

    * Do not print duplicate verbose lines about building fiptool

  - CertCreate

    * Updated tool for platform defined certs, keys & extensions
    * Create only requested certificates
    * Avoid duplicates in extension stack

14.11.3 Resolved Issues

- Several fixes for typos and mis-spellings in documentation
- Build system

  - Fixed ${FIP_NAME} to be rebuilt only when needed in Makefile
  - Do not mark file targets as .PHONY target in Makefile

- Drivers

  - Authorization

    * Avoid NV counter upgrade without certificate validation

  - Arm GICv3

    * Fixed logical issue for num_eints
    * Limit SPI ID to avoid misjudgement in GICD_OFFSET()
    * Fixed potential GICD context override with ESPI enabled

  - Marvell A3700

    * Fixed configuring polarity invert bits

  - Arm TZC-400

    * Correct FAIL_CONTROL Privileged bit
    * Fixed logical error in FILTER_BIT definitions

  - Renesas rcar

    * Fixed several coding style violations reported by checkpatch

- Libraries
  - Arch helpers
    - Fixed assertions in processing dynamic relocations for AArch64 builds
  - C standard library
    - Fixed MISRA issues in memset() ABI
  - RAS
    - Fixed bug of binary search in RAS interrupt handler
- Platforms
  - Arm
    - Fixed missing copyrights in Arm-gic.h file
    - Fixed the order of header files in several dts files
    - Fixed error message printing in board makefile
    - Fixed bug of overriding the last node in image load helper API
    - Fixed stdout-path in fdts files of TC0 and N1SDP platforms
    - Turn ON/OFF redistributor in sync with GIC CPU interface ON/OFF for css platforms
  - Arm FVP
    - Fixed Generic Timer interrupt types in platform dts files
  - Arm Juno
    - Fixed parallel build issue for romlib config
  - Arm SGI
    - Fixed bug in SDEI receive event of RAS handler
  - Intel Agilex
    - Fixed PLAT_MAX_PWR_LVL value
  - Marvell
    - Fixed SPD handling in dram port
  - Marvell ARMADA
    - Fixed TRNG return SMC handling
    - Fixed the logic used for LD selector mask
    - Fixed MSS firmware loader for A8K family
  - ST
    - Fixed few violations reported by coverity static checks
  - STM32MP1

    * Fixed SELFREF_TO_X32 mask in ddr driver
    * Do not keep mmc_device_info in stack
    * Correct plat_crash_console_flush()

  - QEMU SBSA
    * Fixed memory type of secure NOR flash
  - QTI
    * Fixed NUM_APID and REG_APID_MAP() argument in SPMI driver
  - Intel
    * Do not keep mmc_device_info in stack
  - Hisilicon
    * Do not keep mmc_device_info in stack

- Services
  - EL3 runtime
    * Fixed the EL2 context save/restore routine by removing EL2 generic timer system registers
    * Added fix for exception handler in BL31 by synchronizing pending EA using DSB barrier
  - SPMD
    * Fixed error codes to use int32_t type
  - TSPD
    * Added bug fix in tspd interrupt handling when TSP_NS_INTR_ASYNC_PREEMPT is enabled
  - TRNG
    * Fixed compilation errors with -O0 compile option
  - DebugFS
    * Checked channel index before calling clone function
  - PSCI
    * Fixed limit of 256 CPUs caused by cast to unsigned char
  - TSP
    * Fixed compilation erros when built with GCC 11.0.0 toolchain
- Tools
  - FIPtool
    * Do not call `make clean` for `all` target
  - CertCreate

- Fixed bug to avoid cleaning when building the binary
- Used preallocated parts of the HASH struct to avoid leaking HASH struct fields
- Free arguments copied with strdup
- Free keys after use
- Free X509_EXTENSION structures on stack to avoid leaking them
- Optimized the code to avoid unnecessary attempts to create non-requested certificates

14.12 2.4.0 (2020-11-17)

14.12.1 New Features

- Architecture support
  - Armv8.6-A
    - Added support for Armv8.6 Enhanced Counter Virtualization (ECV)
    - Added support for Armv8.6 Fine Grained Traps (FGT)
    - Added support for Armv8.6 WFE trap delays
- Bootloader images
  - Added support for Measured Boot
- Build System
  - Added build option `COT_DESC_IN_DTB` to create Chain of Trust at runtime
  - Added build option `OPENSSL_DIR` to direct tools to OpenSSL libraries
  - Added build option `RAS_TRAP_LOWER_EL_ERR_ACCESS` to enable trapping RAS register accesses from EL1/EL2 to EL3
  - Extended build option `BRANCH_PROTECTION` to support branch target identification
- Common components
  - Added support for exporting CPU nodes to the device tree
  - Added support for single and dual-root Chains of Trust in secure partitions
- Drivers
  - Added Broadcom RNG driver
  - Added Marvell `mg_conf_cm3` driver
  - Added System Control and Management Interface (SCMI) driver
  - Added STMicroelectronics ETZPC driver
  - Arm GICv3

        - Added support for detecting topology at runtime
    - Dual Root
        - Added support for platform certificates
    - Marvell Cache LLC
        - Added support for mapping the entire LLC into SRAM
    - Marvell CCU
        - Added workaround for erratum 3033912
    - Marvell CP110 COMPHY
        - Added support for SATA COMPHY polarity inversion
        - Added support for USB COMPHY polarity inversion
        - Added workaround for erratum IPCE_COMPHY-1353
    - STM32MP1 Clocks
        - Added `RTC` as a gateable clock
        - Added support for shifted clock selector bit masks
        - Added support for using additional clocks as parents
- Libraries
    - C standard library
        - Added support for hexadecimal and pointer format specifiers in `snprint()`
        - Added assembly alternatives for various library functions
    - CPU support
        - Arm Cortex-A53
            - Added workaround for erratum 1530924
        - Arm Cortex-A55
            - Added workaround for erratum 1530923
        - Arm Cortex-A57
            - Added workaround for erratum 1319537
        - Arm Cortex-A76
            - Added workaround for erratum 1165522
            - Added workaround for erratum 1791580
            - Added workaround for erratum 1868343
        - Arm Cortex-A72
            - Added workaround for erratum 1319367

    * Arm Cortex-A77
        · Added workaround for erratum 1508412
        · Added workaround for erratum 1800714
        · Added workaround for erratum 1925769
    * Arm Neoverse-N1
        · Added workaround for erratum 1868343
  – EL3 Runtime
    * Added support for saving/restoring registers related to nested virtualization in EL2 context switches if the architecture supports it
  – FCONF
    * Added support for Measured Boot
    * Added support for populating Chain of Trust properties
    * Added support for loading the `fw_config` image
  – Measured Boot
    * Added support for event logging
- Platforms
  – Added support for Arm Morello
  – Added support for Arm TC0
  – Added support for iEi PUZZLE-M801
  – Added support for Marvell OCTEON TX2 T9130
  – Added support for MediaTek MT8192
  – Added support for NXP i.MX 8M Nano
  – Added support for NXP i.MX 8M Plus
  – Added support for QTI CHIP SC7180
  – Added support for STM32MP151F
  – Added support for STM32MP153F
  – Added support for STM32MP157F
  – Added support for STM32MP151D
  – Added support for STM32MP153D
  – Added support for STM32MP157D
  – Arm
    * Added support for platform-owned SPs

    * Added support for resetting to BL31

- Arm FPGA
    * Added support for Klein
    * Added support for Matterhorn
    * Added support for additional CPU clusters
- Arm FVP
    * Added support for performing SDEI platform setup at runtime
    * Added support for SMCCC's `SMCCC_ARCH_SOC_ID` command
    * Added an `id` field under the NV-counter node in the device tree to differentiate between trusted and non-trusted NV-counters
    * Added support for extracting the clock frequency from the timer node in the device tree
- Arm Juno
    * Added support for SMCCC's `SMCCC_ARCH_SOC_ID` command
- Arm N1SDP
    * Added support for cross-chip PCI-e
- Marvell
    * Added support for AVS reduction
- Marvell ARMADA
    * Added support for twin-die combined memory device
- Marvell ARMADA A8K
    * Added support for DDR with 32-bit bus width (both ECC and non-ECC)
- Marvell AP806
    * Added workaround for erratum FE-4265711
- Marvell AP807
    * Added workaround for erratum 3033912
- Nvidia Tegra
    * Added debug printouts indicating SC7 entry sequence completion
    * Added support for SDEI
    * Added support for stack protection
    * Added support for GICv3
    * Added support for SMCCC's `SMCCC_ARCH_SOC_ID` command
- Nvidia Tegra194

    * Added support for RAS exception handling
    * Added support for SPM

  - NXP i.MX

    * Added support for SDEI

  - QEMU SBSA

    * Added support for the Secure Partition Manager

  - QTI

    * Added RNG driver
    * Added SPMI PMIC arbitrator driver
    * Added support for SMCCC's `SMCCC_ARCH_SOC_ID` command

  - STM32MP1

    * Added support for exposing peripheral interfaces to the non-secure world at runtime
    * Added support for SCMI clock and reset services
    * Added support for STM32MP15x CPU revision Z
    * Added support for SMCCC services in `SP_MIN`

- Services

  - Secure Payload Dispatcher

    * Added a provision to allow clients to retrieve the service UUID

  - SPMC

    * Added secondary core endpoint information to the SPMC context structure

  - SPMD

    * Added support for booting OP-TEE as a guest S-EL1 Secure Partition on top of Hafnium in S-EL2
    * Added a provision for handling SPMC messages to register secondary core entry points
    * Added support for power management operations

- Tools

  - CertCreate

    * Added support for secure partitions

  - CertTool

    * Added support for the `fw_config` image

  - FIPTool

    * Added support for the `fw_config` image

14.12.2 Changed

- Architecture support
- Bootloader images
- Build System
  - The top-level Makefile now supports building FipTool on Windows
  - The default value of `KEY_SIZE` has been changed to to 2048 when RSA is in use
  - The previously-deprecated macro `__ASSEMBLY__` has now been removed
- Common components
  - Certain functions that flush the console will no longer return error information
- Drivers
  - Arm GIC
    - Usage of `drivers/arm/gic/common/gic_common.c` has now been deprecated in favour of `drivers/arm/gic/vX/gicvX.mk`
    - Added support for detecting the presence of a GIC600-AE
    - Added support for detecting the presence of a GIC-Clayton
  - Marvell MCI
    - Now performs link tuning for all MCI interfaces to improve performance
  - Marvell MoChi
    - PIDI masters are no longer forced into a non-secure access level when `LLC_SRAM` is enabled
    - The SD/MMC controllers are now accessible from guest virtual machines
  - Mbed TLS
    - Migrated to Mbed TLS v2.24.0
  - STM32 FMC2 NAND
    - Adjusted FMC node bindings to include an EBI controller node
  - STM32 Reset
    - Added an optional timeout argument to assertion functions
  - STM32MP1 Clocks
    - Enabled several additional system clocks during initialization
- Libraries
  - C Standard Library
    - Improved `memset` performance by avoiding single-byte writes
    - Added optimized assembly variants of `memset`

- CPU support
  - Renamed Cortex-Hercules to Cortex-A78
  - Renamed Cortex-Hercules AE to Cortex-A78 AE
  - Renamed Neoverse Zeus to Neoverse V1
- Coreboot
  - Updated 'coreboot_get_memory_type' API to take an extra argument as a 'memory size' that used to return a valid memory type.
- libfdt
  - Updated to latest upstream version

- Platforms
  - Allwinner
    - Disabled non-secure access to PRCM power control registers
  - Arm
    - `BL32_BASE` is now platform-dependent when `SPD_spmd` is enabled
    - Added support for loading the Chain of Trust from the device tree
    - The firmware update check is now executed only once
    - NV-counter base addresses are now loaded from the device tree when `COT_DESC_IN_DTB` is enabled
    - Now loads and populates `fw_config` and `tb_fw_config`
    - FCONF population now occurs after caches have been enabled in order to reduce boot times
  - Arm Corstone-700
    - Platform support has been split into both an FVP and an FPGA variant
  - Arm FPGA
    - DTB and BL33 load addresses have been given sensible default values
    - Now reads generic timer counter frequency, GICD and GICR base addresses, and UART address from DT
    - Now treats the primary PL011 UART as an SBSA Generic UART
  - Arm FVP
    - Secure interrupt descriptions, UART parameters, clock frequencies and GICv3 parameters are now queried through FCONF
    - UART parameters are now queried through the device tree
    - Added an owner field to Cactus secure partitions
    - Increased the maximum size of BL2 when the Chain of Trust is loaded from the device tree

    * Reduces the maximum size of BL31
    * The `FVP_USE_SP804_TIMER` and `FVP_VE_USE_SP804_TIMER` build options have been removed in favour of a common `USE_SP804_TIMER` option
    * Added a third Cactus partition to manifests
    * Device tree nodes now store UUIDs in big-endian

- Arm Juno
    * Increased the maximum size of BL2 when optimizations have not been applied
    * Reduced the maximum size of BL31 and BL32
- Marvell AP807
    * Enabled snoop filters
- Marvell ARMADA A3K
    * UART recovery images are now suffixed with `.bin`
- Marvell ARMADA A8K
    * Option `BL31_CACHE_DISABLE` is now disabled (`0`) by default
- Nvidia Tegra
    * Added VPR resize supported check when processing video memory resize requests
    * Added SMMU verification to prevent potential issues caused by undetected corruption of the SMMU configuration during boot
    * The GIC CPU interface is now properly disabled after CPU off
    * The GICv2 sources list and the `BL31_SIZE` definition have been made platform-specific
    * The SPE driver will no longer flush the console when writing individual characters
- Nvidia Tegra194
    * TZDRAM setup has been moved to platform-specific early boot handlers
    * Increased verbosity of debug prints for RAS SErrors
    * Support for powering down CPUs during CPU suspend has been removed
    * Now verifies firewall settings before using resources
- TI K3
    * The UART number has been made configurable through `K3_USART`
- Rockchip RK3368
    * The maximum number of memory map regions has been increased to 20
- Socionext Uniphier
    * The maximum size of BL33 has been increased to support larger bootloaders
- STM32

    * Removed platform-specific DT functions in favour of using existing generic alternatives

  – STM32MP1

    * Increased verbosity of exception reports in debug builds
    * Device trees have been updated to align with the Linux kernel
    * Now uses the ETZPC driver to configure secure-aware interfaces for assignment to the non-secure world
    * Finished good variants have been added to the board identifier enumerations
    * Non-secure access to clocks and reset domains now depends on their state of registration
    * NEON is now disabled in `SP_MIN`
    * The last page of `SYSRAM` is now used as SCMI shared memory
    * Checks to verify platform compatibility have been added to verify that an image is compatible with the chip ID of the running platform

  – QEMU SBSA

    * Removed support for Arm's Cortex-A53

- Services

  – Renamed SPCI to FF-A

  – SPMD

    * No longer forwards requests to the non-secure world when retrieving partition information
    * SPMC manifest size is now retrieved directly from SPMD instead of the device tree
    * The FF-A version handler now returns SPMD's version when the origin of the call is secure, and SPMC's version when the origin of the call is non-secure

  – SPMC

    * Updated the manifest to declare CPU nodes in descending order as per the SPM (Hafnium) multicore requirement
    * Updated the device tree to mark 2GB as device memory for the first partition excluding trusted DRAM region (which is reserved for SPMC)
    * Increased the number of EC contexts to the maximum number of PEs as per the FF-A specification

- Tools

  – FIPTool

    * Now returns `0` on `help` and `help <command>`

  – Marvell DoImage

    * Updated Mbed TLS support to v2.8

  – SPTool

    * Now appends CertTool arguments

14.12.3 Resolved Issues

- Bootloader images
  - Fixed compilation errors for dual-root Chains of Trust caused by symbol collision
  - BL31
    * Fixed compilation errors on platforms with fewer than 4 cores caused by initialization code exceeding the end of the stacks
    * Fixed compilation errors when building a position-independent image
- Build System
  - Fixed invalid empty version strings
  - Fixed compilation errors on Windows caused by a non-portable architecture revision comparison
- Drivers
  - Arm GIC
    * Fixed spurious interrupts caused by a missing barrier
  - STM32 Flexible Memory Controller 2 (FMC2) NAND driver
    * Fixed runtime instability caused by incorrect error detection logic
  - STM32MP1 Clock driver
    * Fixed incorrectly-formatted log messages
    * Fixed runtime instability caused by improper clock gating procedures
  - STMicroelectronics Raw NAND driver
    * Fixed runtime instability caused by incorrect unit conversion when waiting for NAND readiness
- Libraries
  - AMU
    * Fixed timeout errors caused by excess error logging
  - EL3 Runtime
    * Fixed runtime instability caused by improper register save/restore routine in EL2
  - FCONF
    * Fixed failure to initialize GICv3 caused by overly-strict device tree requirements
  - Measured Boot
    * Fixed driver errors caused by a missing default value for the `HASH_ALG` build option

- SPE
  - Fixed feature detection check that prevented CPUs supporting SVE from detecting support for SPE in the non-secure world
- Translation Tables
  - Fixed various MISRA-C 2012 static analysis violations

- Platforms
  - Allwinner A64
    - Fixed USB issues on certain battery-powered device caused by improperly activated USB power rail
  - Arm
    - Fixed compilation errors caused by increase in BL2 size
    - Fixed compilation errors caused by missing Makefile dependencies to generated files when building the FIP
    - Fixed MISRA-C 2012 static analysis violations caused by unused structures in include directives intended to be feature-gated
  - Arm FPGA
    - Fixed initialization issues caused by incorrect MPIDR topology mapping logic
  - Arm RD-N1-edge
    - Fixed compilation errors caused by mismatched parentheses in Makefile
  - Arm SGI
    - Fixed crashes due to the flash memory used for cold reboot attack protection not being mapped
  - Intel Agilex
    - Fixed initialization issues caused by several compounding bugs
  - Marvell
    - Fixed compilation warnings caused by multiple Makefile inclusions
  - Marvell ARMADA A3K
    - Fixed boot issue in debug builds caused by checks on the BL33 load address that are not appropriate for this platform
  - Nvidia Tegra
    - Fixed incorrect delay timer reads
    - Fixed spurious interrupts in the non-secure world during cold boot caused by the arbitration bit in the memory controller not being cleared
    - Fixed faulty video memory resize sequence
  - Nvidia Tegra194

    * Fixed incorrect alignment of TZDRAM base address

  - NXP iMX8M

    * Fixed CPU hot-plug issues caused by race condition

  - STM32MP1

    * Fixed compilation errors in highly-parallel builds caused by incorrect Makefile dependencies

  - STM32MP157C-ED1

    * Fixed initialization issues caused by missing device tree hash node

  - Raspberry Pi 3

    * Fixed compilation errors caused by incorrect dependency ordering in Makefile

  - Rockchip

    * Fixed initialization issues caused by non-critical errors when parsing FDT being treated as critical

  - Rockchip RK3368

    * Fixed runtime instability caused by incorrect CPUID shift value

  - QEMU

    * Fixed compilation errors caused by incorrect dependency ordering in Makefile

  - QEMU SBSA

    * Fixed initialization issues caused by FDT exceeding reserved memory size

  - QTI

    * Fixed compilation errors caused by inclusion of a non-existent file

- Services

  - FF-A (previously SPCI)

    * Fixed SPMD aborts caused by incorrect behaviour when the manifest is page-aligned

- Tools

  - Fixed compilation issues when compiling tools from within their respective directories

  - FIPTool

    * Fixed command line parsing issues on Windows when using arguments whose names also happen to be a subset of another's

  - Marvell DoImage

    * Fixed PKCS signature verification errors at boot on some platforms caused by generation of misaligned images

14.12.4 Known Issues

- Platforms
  - NVIDIA Tegra
    - Signed comparison compiler warnings occurring in libfdt are currently being worked around by disabling the warning for the platform until the underlying issue is resolved in libfdt

14.13 2.3.0 (2020-04-20)

14.13.1 New Features

- Arm Architecture
  - Add support for Armv8.4-SecEL2 extension through the SPCI defined SPMD/SPMC components.
  - Build option to support EL2 context save and restore in the secure world (CTX_INCLUDE_EL2_REGS).
  - Add support for SMCCC v1.2 (introducing the new SMCCC_ARCH_SOC_ID SMC). Note that the support is compliant, but the SVE registers save/restore will be done as part of future S-EL2/SPM development.
- BL-specific
  - Enhanced BL2 bootloader flow to load secure partitions based on firmware configuration data (fconf).
  - Changes necessary to support SEPARATE_NOBITS_REGION feature
  - TSP and BL2_AT_EL3: Add Position Independent Execution `PIE` support
- Build System
  - Add support for documentation build as a target in Makefile
  - Add `COT` build option to select the Chain of Trust to use when the Trusted Boot feature is enabled (default: `tbbr`).
  - Added creation and injection of secure partition packages into the FIP.
  - Build option to support SPMC component loading and run at S-EL1 or S-EL2 (SPMD_SPM_AT_SEL2).
  - Enable MTE support
  - Enable Link Time Optimization in GCC
  - Enable -Wredundant-decls warning check
  - Makefile: Add support to optionally encrypt BL31 and BL32
  - Add support to pass the nt_fw_config DTB to OP-TEE.
  - Introduce per-BL `CPPFLAGS`, `ASFLAGS`, and `LDFLAGS`

- build_macros: Add CREATE_SEQ function to generate sequence of numbers
- CPU Support
  - cortex-a57: Enable higher performance non-cacheable load forwarding
  - Hercules: Workaround for Errata 1688305
  - Klein: Support added for Klein CPU
  - Matterhorn: Support added for Matterhorn CPU
- Drivers
  - auth: Add `calc_hash` function for hash calculation. Used for authentication of images when measured boot is enabled.
  - cryptocell: Add authenticated decryption framework, and support for CryptoCell-713 and CryptoCell-712 RSA 3K
  - gic600: Add support for multichip configuration and Clayton
  - gicv3: Introduce makefile, Add extended PPI and SPI range, Add support for probing multiple GIC Redistributor frames
  - gicv4: Add GICv4 extension for GIC driver
  - io: Add an IO abstraction layer to load encrypted firmwares
  - mhu: Derive doorbell base address
  - mtd: Add SPI-NOR, SPI-NAND, SPI-MEM, and raw NAND framework
  - scmi: Allow use of multiple SCMI channels
  - scu: Add a driver for snoop control unit
- Libraries
  - coreboot: Add memory range parsing and use generic base address
  - compiler_rt: Import popcountdi2.c and popcountsi2.c files, aeabi_ldivmode.S file and dependencies
  - debugFS: Add DebugFS functionality
  - el3_runtime: Add support for enabling S-EL2
  - fconf: Add Firmware Configuration Framework (fconf) (experimental).
  - libc: Add memrchr function
  - locks: bakery: Use is_dcache_enabled() helper and add a DMB to the 'read_cache_op' macro
  - psci: Add support to enable different personality of the same soc.
  - xlat_tables_v2: Add support to pass shareability attribute for normal memory region, use get_current_el_maybe_constant() in is_dcache_enabled(), read-only xlat tables for BL31 memory, and add enable_mmu()
- New Platforms Support

  - arm/arm_fpga: New platform support added for FPGA
  - arm/rddaniel: New platform support added for rd-daniel platform
  - brcm/stingray: New platform support added for Broadcom stingray platform
  - nvidia/tegra194: New platform support for Nvidia Tegra194 platform
- Platforms
  - allwinner: Implement PSCI system suspend using SCPI, add a msgbox driver for use with SCPI, and reserve and map space for the SCP firmware
  - allwinner: axp: Add AXP805 support
  - allwinner: power: Add DLDO4 power rail
  - amlogic: axg: Add a build flag when using ATOS as BL32 and support for the A113D (AXG) platform
  - arm/a5ds: Add ethernet node and L2 cache node in devicetree
  - arm/common: Add support for the new `dualroot` chain of trust
  - arm/common: Add support for SEPARATE_NOBITS_REGION
  - arm/common: Re-enable PIE when RESET_TO_BL31=1
  - arm/common: Allow boards to specify second DRAM Base address and to define PLAT_ARM_TZC_FILTERS
  - arm/corstone700: Add support for mhuv2 and stack protector
  - arm/fvp: Add support for fconf in BL31 and SP_MIN. Populate power domain descriptor dynamically by leveraging fconf APIs.
  - arm/fvp: Add Cactus/Ivy Secure Partition information and use two instances of Cactus at S-EL1
  - arm/fvp: Add support to run BL32 in TDRAM and BL31 in secure DRAM
  - arm/fvp: Add support for GICv4 extension and BL2 hash calculation in BL1
  - arm/n1sdp: Setup multichip gic routing table, update platform macros for dual-chip setup, introduce platform information SDS region, add support to update presence of External LLC, and enable the NEOVERSE_N1_EXTERNAL_LLC flag
  - arm/rdn1edge: Add support for dual-chip configuration and use CREATE_SEQ helper macro to compare chip count
  - arm/sgm: Always use SCMI for SGM platforms
  - arm/sgm775: Add support for dynamic config using fconf
  - arm/sgi: Add multi-chip mode parameter in HW_CONFIG dts, macros for remote chip device region, chip_id and multi_chip_mode to platform variant info, and introduce number of chips macro
  - brcm: Add BL2 and BL31 support common across Broadcom platforms

- brcm: Add iproc SPI Nor flash support, spi driver, emmc driver, and support to retrieve plat_toc_flags
- hisilicon: hikey960: Enable system power off callback
- intel: Enable bridge access, SiP SMC secure register access, and uboot entrypoint support
- intel: Implement platform specific system reset 2
- intel: Introduce mailbox response length handling
- imx: console: Use CONSOLE_T_BASE for UART base address and generic console_t data structure
- imx8mm: Provide uart base as build option and add the support for opteed spd on imx8mq/imx8mm
- imx8qx: Provide debug uart num as build
- imx8qm: Apply clk/pinmux configuration for DEBUG_CONSOLE and provide debug uart num as build param
- marvell: a8k: Implement platform specific power off and add support for loading MG CM3 images
- mediatek: mt8183: Add Vmodem/Vcore DVS init level
- qemu: Support optional encryption of BL31 and BL32 images and ARM_LINUX_KERNEL_AS_BL33 to pass FDT address
- qemu: Define ARMV7_SUPPORTS_VFP
- qemu: Implement PSCI_CPU_OFF and qemu_system_off via semihosting
- renesas: rcar_gen3: Add new board revision for M3ULCB
- rockchip: Enable workaround for erratum 855873, claim a macro to enable hdcp feature for DP, enable power domains of rk3399 before reset, add support for UART3 as serial output, and initialize reset and poweroff GPIOs with known invalid value
- rpi: Implement PSCI CPU_OFF, use MMIO accessor, autodetect Mini-UART vs. PL011 configuration, and allow using PL011 UART for RPi3/RPi4
- rpi3: Include GPIO driver in all BL stages and use same "clock-less" setup scheme as RPi4
- rpi3/4: Add support for offlining CPUs
- st: stm32mp1: platform.mk: Support generating multiple images in one build, migrate to implicit rules, derive map file name from target name, generate linker script with fixed name, and use PHONY for the appropriate targets
- st: stm32mp1: Add support for SPI-NOR, raw NAND, and SPI-NAND boot device, QSPI, FMC2 driver
- st: stm32mp1: Use stm32mp_get_ddr_ns_size() function, set XN attribute for some areas in BL2, dynamically map DDR later and non-cacheable during its test, add a function to get non-secure DDR size, add DT helper for reg by name, and add compilation flags for boot devices
- socionext: uniphier: Turn on ENABLE_PIE

- ti: k3: Add PIE support
- xilinx: versal: Add set wakeup source, client wakeup, query data, request wakeup, PM_INIT_FINALIZE, PM_GET_TRUSTZONE_VERSION, PM IOCTL, support for suspend related, and Get_ChipID APIs
- xilinx: versal: Implement power down/restart related EEMI, SMC handler for EEMI, PLL related PM, clock related PM, pin control related PM, reset related PM, device related PM , APIs
- xilinx: versal: Enable ipi mailbox service
- xilinx: versal: Add get_api_version support and support to send PM API to PMC using IPI
- xilinx: zynqmp: Add checksum support for IPI data, GET_CALLBACK_DATA function, support to query max divisor, CLK_SET_RATE_PARENT in gem clock node, support for custom type flags, LPD WDT clock to the pm_clock structure, idcodes for new RFSoC silicons ZU48DR and ZU49DR, and id for new RFSoC device ZU39DR

- Security
  - Use Speculation Barrier instruction for v8.5+ cores
  - Add support for optional firmware encryption feature (experimental).
  - Introduce a new `dualroot` chain of trust.
  - aarch64: Prevent speculative execution past ERET
  - aarch32: Stop speculative execution past exception returns.
- SPCI
  - Introduced the Secure Partition Manager Dispatcher (SPMD) component as a new standard service.
- Tools
  - cert_create: Introduce CoT build option and TBBR CoT makefile, and define the dualroot CoT
  - encrypt_fw: Add firmware authenticated encryption tool
  - memory: Add show_memory script that prints a representation of the memory layout for the latest build

14.13.2 Changed

- Arm Architecture
  - PIE: Make call to GDT relocation fixup generalized
- BL-Specific
  - Increase maximum size of BL2 image
  - BL31: Discard .dynsym .dynstr .hash sections to make ENABLE_PIE work
  - BL31: Split into two separate memory regions
  - Unify BL linker scripts and reduce code duplication.

- Build System
  - Changes to drive cert_create for dualroot CoT
  - Enable -Wlogical-op always
  - Enable -Wshadow always
  - Refactor the warning flags
  - PIE: Pass PIE options only to BL31
  - Reduce space lost to object alignment
  - Set lld as the default linker for Clang builds
  - Remove -Wunused-const-variable and -Wpadded warning
  - Remove -Wmissing-declarations warning from WARNING1 level
- Drivers
  - authentication: Necessary fix in drivers to upgrade to mbedtls-2.18.0
  - console: Integrate UART base address in generic console_t
  - gicv3: Change API for GICR_IPRIORITYR accessors and separate GICD and GICR accessor functions
  - io: Change seek offset to signed long long and panic in case of io setup failure
  - smmu: SMMUv3: Changed retry loop to delay timer
  - tbbr: Reduce size of hash and ECDSA key buffers when possible
- Library Code
  - libc: Consolidate the size_t, unified, and NULL definitions, and unify intmax_t and uintmax_t on AArch32/64
  - ROMLIB: Optimize memory layout when ROMLIB is used
  - xlat_tables_v2: Use ARRAY_SIZE in REGISTER_XLAT_CONTEXT_FULL_SPEC, merge REGISTER_XLAT_CONTEXT_{FULL_SPEC,RO_BASE_TABLE}, and simplify end address checks in mmap_add_region_check()
- Platforms
  - allwinner: Adjust SRAM A2 base to include the ARISC vectors, clean up MMU setup, reenable USE_COHERENT_MEM, remove unused include path, move the NOBITS region to SRAM A1, convert AXP803 regulator setup code into a driver, enable clock before resetting I2C/RSB
  - allwinner: h6: power: Switch to using the AXP driver
  - allwinner: a64: power: Use fdt_for_each_subnode, remove obsolete register check, remove duplicate DT check, and make sunxi_turn_off_soc static
  - allwinner: Build PMIC bus drivers only in BL31, clean up PMIC-related error handling, and synchronize PMIC enumerations
  - arm/a5ds: Change boot address to point to DDR address

- arm/common: Check for out-of-bound accesses in the platform io policies
- arm/corstone700: Updating the kernel arguments to support initramfs, use fdts DDR memory and XIP rootfs, and set UART clocks to 32MHz
- arm/fvp: Modify multithreaded dts file of DynamIQ FVPs, slightly bump the stack size for bl1 and bl2, remove re-definition of topology related build options, stop reclaiming init code with Clang builds, and map only the needed DRAM region statically in BL31/SP_MIN
- arm/juno: Maximize space allocated to SCP_BL2
- arm/sgi: Bump bl1 RW limit, mark remote chip shared ram as non-cacheable, move GIC related constants to board files, include AFF3 affinity in core position calculation, move bl31_platform_setup to board file, and move topology information to board folder
- common: Refactor load_auth_image_internal().
- hisilicon: Remove uefi-tools in hikey and hikey960 documentation
- intel: Modify non secure access function, BL31 address mapping, mailbox's get_config_status, and stratix10 BL31 parameter handling
- intel: Remove un-needed checks for qspi driver r/w and s10 unused source code
- intel: Change all global sip function to static
- intel: Refactor common platform code
- intel: Create SiP service header file
- marvell: armada: scp_bl2: Allow loading up to 8 images
- marvell: comphy-a3700: Support SGMII COMPHY power off and fix USB3 powering on when on lane 2
- marvell: Consolidate console register calls
- mediatek: mt8183: Protect 4GB~8GB dram memory, refine GIC driver for low power scenarios, and switch PLL/CLKSQ/ck_off/axi_26m control to SPM
- qemu: Update flash address map to keep FIP in secure FLASH0
- renesas: rcar_gen3: Update IPL and Secure Monitor Rev.2.0.6, update DDR setting for H3, M3, M3N, change fixed destination address of BL31 and BL32, add missing #{address,size}-cells into generated DT, pass DT to OpTee OS, and move DDR drivers out of staging
- rockchip: Make miniloader ddr_parameter handling optional, cleanup securing of ddr regions, move secure init to separate file, use base+size for secure ddr regions, bring TZRAM_SIZE values in lined, and prevent macro expansion in paths
- rpi: Move plat_helpers.S to common
- rpi3: gpio: Simplify GPIO setup
- rpi4: Skip UART initialisation
- st: stm32m1: Use generic console_t data structure, remove second QSPI flash instance, update for FMC2 pin muxing, and reduce MAX_XLAT_TABLES to 4

- socionext: uniphier: Make on-chip SRAM and I/O register regions configurable
- socionext: uniphier: Make PSCI related, counter control, UART, pinmon, NAND controller, and eMMC controller base addresses configurable
- socionext: uniphier: Change block_addressing flag and the return value type of .is_usb_boot() to bool
- socionext: uniphier: Run BL33 at EL2, call uniphier_scp_is_running() only when on-chip STM is supported, define PLAT_XLAT_TABLES_DYNAMIC only for BL2, support read-only xlat tables, use enable_mmu() in common function, shrink UNIPHIER_ROM_REGION_SIZE, prepare uniphier_soc_info() for next SoC, extend boot device detection for future SoCs, make all BL images completely position-independent, make uniphier_mmap_setup() work with PIE, pass SCP base address as a function parameter, set buffer offset and length for io_block dynamically, and use more mmap_add_dynamic_region() for loading images
- spd/trusty: Disable error messages seen during boot, allow gic base to be specified with GICD_BASE, and allow getting trusty memsize from BL32_MEM_SIZE instead of TSP_SEC_MEM_SIZE
- ti: k3: common: Enable ARM cluster power down and rename device IDs to be more consistent
- ti: k3: drivers: ti_sci: Put sequence number in coherent memory and remove indirect structure of const data
- xilinx: Move ipi mailbox svc to xilinx common
- xilinx: zynqmp: Use GIC framework for warm restart
- xilinx: zynqmp: pm: Move custom clock flags to typeflags, remove CLK_TOPSW_LSBUS from invalid clock list and rename FPD WDT clock ID
- xilinx: versal: Increase OCM memory size for DEBUG builds and adjust cpu clock, Move versal_def.h and versal_private to include directory

• Tools
  - sptool: Updated sptool to accommodate building secure partition packages.

14.13.3 Resolved Issues

• Arm Architecture
  - Fix crash dump for lower EL
• BL-Specific
  - Bug fix: Protect TSP prints with lock
  - Fix boot failures on some builds linked with ld.lld.
• Build System
  - Fix clang build if CC is not in the path.
  - Fix 'BL stage' comment for build macros

- Code Quality
  - coverity: Fix various MISRA violations including null pointer violations, C issues in BL1/BL2/BL31 and FDT helper functions, using boolean essential, type, and removing unnecessary header file and comparisons to LONG_MAX in debugfs devfip
  - Based on coding guidelines, replace all `unsigned long` depending on if fixed based on AArch32 or AArch64.
  - Unify type of "cpu_idx" and Platform specific defines across PSCI module.
- Drivers
  - auth: Necessary fix in drivers to upgrade to mbedtls-2.18.0
  - delay_timer: Fix non-standard frequency issue in udelay
  - gicv3: Fix compiler dependent behavior
  - gic600: Fix include ordering according to the coding style and power up sequence
- Library Code
  - el3_runtime: Fix stack pointer maintenance on EA handling path, fixup 'cm_setup_context' prototype, and adds TPIDR_EL2 register to the context save restore routines
  - libc: Fix SIZE_MAX on AArch32
  - locks: T589: Fix insufficient ordering guarantees in bakery lock
  - pmf: Fix 'tautological-constant-compare' error, Make the runtime instrumentation work on AArch32, and Simplify PMF helper macro definitions across header files
  - xlat_tables_v2: Fix assembler warning of PLAT_RO_XLAT_TABLES
- Platforms
  - allwinner: Fix H6 GPIO and CCU memory map addresses and incorrect ARISC code patch offset check
  - arm/a5ds: Correct system freq and Cache Writeback Granule, and cleanup enable-method in devicetree
  - arm/fvp: Fix incorrect GIC mapping, BL31 load address and image size for RESET_TO_BL31=1, topology description of cpus for DynamIQ based FVP, and multithreaded FVP power domain tree
  - arm/fvp: spm-mm: Correcting instructions to build SPM for FVP
  - arm/common: Fix ROTPK hash generation for ECDSA encryption, BL2 bug in dynamic configuration initialisation, and current RECLAIM_INIT_CODE behavior
  - arm/rde1edge: Fix incorrect topology tree description
  - arm/sgi: Fix the incorrect check for SCMI channel ID
  - common: Flush dcache when storing timestamp
  - intel: Fix UEFI decompression issue, memory calibration, SMC SIP service, mailbox config return status, mailbox driver logic, FPGA manager on reconfiguration, and mailbox send_cmd issue

  - imx: Fix shift-overflow errors, the rdc memory region slot's offset, multiple definition of ipc_handle, missing inclusion of cdefs.h, and correct the SGIs that used for secure interrupt
  - mediatek: mt8183: Fix AARCH64 init fail on CPU0
  - rockchip: Fix definition of struct param_ddr_usage
  - rpi4: Fix documentation of armstub config entry
  - st: Correct io possible NULL pointer dereference and device_size type, nand xor_ecc.val assigned value, static analysis tool issues, and fix incorrect return value and correctly check pwr-regulators node
  - xilinx: zynqmp: Correct syscnt freq for QEMU and fix clock models and IDs of GEM-related clocks

14.13.4 Known Issues

- Build System
  - dtb: DTB creation not supported when building on a Windows host.

    This step in the build process is skipped when running on a Windows host. A known issue from the 1.6 release.
  - Intermittent assertion firing `ASSERT: services/spd/tspd/tspd_main.c:105`
- Coverity
  - Intermittent Race condition in Coverity Jenkins Build Job
- Platforms
  - arm/juno: System suspend from Linux does not function as documented in the user guide

    Following the instructions provided in the user guide document does not result in the platform entering system suspend state as expected. A message relating to the hdlcd driver failing to suspend will be emitted on the Linux terminal.
  - mediatek/mt6795: This platform does not build in this release

14.14 2.2.0 (2019-10-22)

14.14.1 New Features

- Architecture
  - Enable Pointer Authentication (PAuth) support for Secure World
    - Adds support for ARMv8.3-PAuth in BL1 SMC calls and BL2U image for firmware updates.
  - Enable Memory Tagging Extension (MTE) support in both secure and non-secure worlds

        * Adds support for the new Memory Tagging Extension arriving in ARMv8.5. MTE support is now enabled by default on systems that support it at EL0.
        * To enable it at ELx for both the non-secure and the secure world, the compiler flag `CTX_INCLUDE_MTE_REGS` includes register saving and restoring when necessary in order to prevent information leakage between the worlds.
    - Add support for Branch Target Identification (BTI)
- Build System
    - Modify FVP makefile for CPUs that support both AArch64/32
    - AArch32: Allow compiling with soft-float toolchain
    - Makefile: Add default warning flags
    - Add Makefile check for PAuth and AArch64
    - Add compile-time errors for HW_ASSISTED_COHERENCY flag
    - Apply compile-time check for AArch64-only CPUs
    - build_macros: Add mechanism to prevent bin generation.
    - Add support for default stack-protector flag
    - spd: opteed: Enable NS_TIMER_SWITCH
    - plat/arm: Skip BL2U if RESET_TO_SP_MIN flag is set
    - Add new build option to let each platform select which implementation of spinlocks it wants to use
- CPU Support
    - DSU: Workaround for erratum 798953 and 936184
    - Neoverse N1: Force cacheable atomic to near atomic
    - Neoverse N1: Workaround for erratum 1073348, 1130799, 1165347, 1207823, 1220197, 1257314, 1262606, 1262888, 1275112, 1315703, 1542419
    - Neoverse Zeus: Apply the MSR SSBS instruction
    - cortex-Hercules/HerculesAE: Support added for Cortex-Hercules and Cortex-HerculesAE CPUs
    - cortex-Hercules/HerculesAE: Enable AMU for Cortex-Hercules and Cortex-HerculesAE
    - cortex-a76AE: Support added for Cortex-A76AE CPU
    - cortex-a76: Workaround for erratum 1257314, 1262606, 1262888, 1275112, 1286807
    - cortex-a65/a65AE: Support added for Cortex-A65 and Cortex-A65AE CPUs
    - cortex-a65: Enable AMU for Cortex-A65
    - cortex-a55: Workaround for erratum 1221012
    - cortex-a35: Workaround for erratum 855472
    - cortex-a9: Workaround for erratum 794073

- Drivers
  - console: Allow the console to register multiple times
  - delay: Timeout detection support
  - gicv3: Enabled multi-socket GIC redistributor frame discovery and migrated ARM platforms to the new API
    - Adds `gicv3_rdistif_probe` function that delegates the responsibility of discovering the corresponding redistributor base frame to each CPU itself.
  - sbsa: Add SBSA watchdog driver
  - st/stm32_hash: Add HASH driver
  - ti/uart: Add an AArch32 variant
- Library at ROM (romlib)
  - Introduce BTI support in Library at ROM (romlib)
- New Platforms Support
  - amlogic: g12a: New platform support added for the S905X2 (G12A) platform
  - amlogic: meson/gxl: New platform support added for Amlogic Meson S905x (GXL)
  - arm/a5ds: New platform support added for A5 DesignStart
  - arm/corstone: New platform support added for Corstone-700
  - intel: New platform support added for Agilex
  - mediatek: New platform support added for MediaTek mt8183
  - qemu/qemu_sbsa: New platform support added for QEMU SBSA platform
  - renesas/rcar_gen3: plat: New platform support added for D3
  - rockchip: New platform support added for px30
  - rockchip: New platform support added for rk3288
  - rpi: New platform support added for Raspberry Pi 4
- Platforms
  - arm/common: Introduce wrapper functions to setup secure watchdog
  - arm/fvp: Add Delay Timer driver to BL1 and BL31 and option for defining platform DRAM2 base
  - arm/fvp: Add Linux DTS files for 32 bit threaded FVPs
  - arm/n1sdp: Add code for DDR ECC enablement and BL33 copy to DDR, Initialise CNTFRQ in Non Secure CNTBaseN
  - arm/juno: Use shared mbedtls heap between BL1 and BL2 and add basic support for dynamic config
  - imx: Basic support for PicoPi iMX7D, rdc module init, caam module init, aipstz init, IMX_SIP_GET_SOC_INFO, IMX_SIP_BUILDINFO added

  - intel: Add ncore ccu driver
  - mediatek/mt81*: Use new bl31_params_parse() helper
  - nvidia: tegra: Add support for multi console interface
  - qemu/qemu_sbsa: Adding memory mapping for both FLASH0/FLASH1
  - qemu: Added gicv3 support, new console interface in AArch32, and sub-platforms
  - renesas/rcar_gen3: plat: Add R-Car V3M support, new board revision for H3ULCB, DBSC4 setting before self-refresh mode
  - socionext/uniphier: Support console based on multi-console
  - st: stm32mp1: Add OP-TEE, Avenger96, watchdog, LpDDR3, authentication support and general SYSCFG management
  - ti/k3: common: Add support for J721E, Use coherent memory for shared data, Trap all asynchronous bus errors to EL3
  - xilinx/zynqmp: Add support for multi console interface, Initialize IPI table from zynqmp_config_setup()
- PSCI
  - Adding new optional PSCI hook `pwr_domain_on_finish_late`
    - This PSCI hook `pwr_domain_on_finish_late` is similar to `pwr_domain_on_finish` but is guaranteed to be invoked when the respective core and cluster are participating in coherency.
- Security
  - Speculative Store Bypass Safe (SSBS): Further enhance protection against Spectre variant 4 by disabling speculative loads/stores (SPSR.SSBS bit) by default.
  - UBSAN support and handlers
    - Adds support for the Undefined Behaviour sanitizer. There are two types of support offered - minimalistic trapping support which essentially immediately crashes on undefined behaviour and full support with full debug messages.
- Tools
  - cert_create: Add support for bigger RSA key sizes (3KB and 4KB), previously the maximum size was 2KB.
  - fiptool: Add support to build fiptool on Windows.

14.14.2 Changed

- Architecture
  - Refactor ARMv8.3 Pointer Authentication support code
  - backtrace: Strip PAC field when PAUTH is enabled
  - Prettify crash reporting output on AArch64.
  - Rework smc_unknown return code path in smc_handler
    * Leverage the existing `el3_exit()` return routine for smc_unknown return path rather than a custom set of instructions.
- BL-Specific
  - Invalidate dcache build option for BL2 entry at EL3
  - Add missing support for BL2_AT_EL3 in XIP memory
- Boot Flow
  - Add helper to parse BL31 parameters (both versions)
  - Factor out cross-BL API into export headers suitable for 3rd party code
  - Introduce lightweight BL platform parameter library
- Drivers
  - auth: Memory optimization for Chain of Trust (CoT) description
  - bsec: Move bsec_mode_is_closed_device() service to platform
  - cryptocell: Move Cryptocell specific API into driver
  - gicv3: Prevent pending G1S interrupt from becoming G0 interrupt
  - mbedtls: Remove weak heap implementation
  - mmc: Increase delay between ACMD41 retries
  - mmc: stm32_sdmmc2: Correctly manage block size
  - mmc: stm32_sdmmc2: Manage max-frequency property from DT
  - synopsys/emmc: Do not change FIFO TH as this breaks some platforms
  - synopsys: Update synopsys drivers to not rely on undefined overflow behaviour
  - ufs: Extend the delay after reset to wait for some slower chips
- Platforms
  - amlogic/meson/gxl: Remove BL2 dependency from BL31
  - arm/common: Shorten the Firmware Update (FWU) process
  - arm/fvp: Remove GIC initialisation from secondary core cold boot
  - arm/sgm: Temporarily disable shared Mbed TLS heap for SGM

  - hisilicon: Update hisilicon drivers to not rely on undefined overflow behaviour
  - imx: imx8: Replace PLAT_IMX8* with PLAT_imx8*, remove duplicated linker symbols and deprecated code include, keep only IRQ 32 unmasked, enable all power domain by default
  - marvell: Prevent SError accessing PCIe link, Switch to xlat_tables_v2, do not rely on argument passed via smc, make sure that comphy init will use correct address
  - mediatek: mt8173: Refactor RTC and PMIC drivers
  - mediatek: mt8173: Apply MULTI_CONSOLE framework
  - nvidia: Tegra: memctrl_v2: fix "overflow before widen" coverity issue
  - qemu: Simplify the image size calculation, Move and generalise FDT PSCI fixup, move gicv2 codes to separate file
  - renesas/rcar_gen3: Convert to multi-console API, update QoS setting, Update IPL and Secure Monitor Rev2.0.4, Change to restore timer counter value at resume, Update DDR setting rev.0.35, qos: change subslot cycle, Change periodic write DQ training option.
  - rockchip: Allow SOCs with undefined wfe check bits, Streamline and complete UARTn_BASE macros, drop rockchip-specific imported linker symbols for bl31, Disable binary generation for all SOCs, Allow console device to be set by DTB, Use new bl31_params_parse functions
  - rpi/rpi3: Move shared rpi3 files into common directory
  - socionext/uniphier: Set CONSOLE_FLAG_TRANSLATE_CRLF and clean up console driver
  - socionext/uniphier: Replace DIV_ROUND_UP() with div_round_up() from utils_def.h
  - st/stm32mp: Split stm32mp_io_setup function, move stm32_get_gpio_bank_clock() to private file, correctly handle Clock Spreading Generator, move oscillator functions to generic file, realign device tree files with internal devs, enable RTCAPB clock for dual-core chips, use a common function to check spinlock is available, move check_header() to common code
  - ti/k3: Enable SEPARATE_CODE_AND_RODATA by default, Remove shared RAM space, Drop _ADDRESS from K3_USART_BASE to match other defines, Remove MSMC port definitions, Allow USE_COHERENT_MEM for K3, Set L2 latency on A72 cores

- PSCI
  - PSCI: Lookup list of parent nodes to lock only once
- Secure Partition Manager (SPM): SPCI Prototype
  - Fix service UUID lookup
  - Adjust size of virtual address space per partition
  - Refactor xlat context creation
  - Move shim layer to TTBR1_EL1
  - Ignore empty regions in resource description
- Security
  - Refactor SPSR initialisation code

- SMMUv3: Abort DMA transactions
  - For security DMA should be blocked at the SMMU by default unless explicitly enabled for a device. SMMU is disabled after reset with all streams bypassing the SMMU, and abortion of all incoming transactions implements a default deny policy on reset.
  - Moves `bl1_platform_setup()` function from arm_bl1_setup.c to FVP platforms' fvp_bl1_setup.c and fvp_ve_bl1_setup.c files.

- Tools
  - cert_create: Remove RSA PKCS#1 v1.5 support

14.14.3 Resolved Issues

- Architecture
  - Fix the CAS spinlock implementation by adding a missing DSB in `spin_unlock()`
  - AArch64: Fix SCTLR bit definitions
    - Removes incorrect `SCTLR_V_BIT` definition and adds definitions for ARMv8.3-Pauth `EnIB`, `EnDA` and `EnDB` bits.
  - Fix restoration of PAuth context
    - Replace call to `pauth_context_save()` with `pauth_context_restore()` in case of unknown SMC call.
- BL-Specific Issues
  - Fix BL31 crash reporting on AArch64 only platforms
- Build System
  - Remove several warnings reported with W=2 and W=1
- Code Quality Issues
  - SCTLR and ACTLR are 32-bit for AArch32 and 64-bit for AArch64
  - Unify type of "cpu_idx" across PSCI module.
  - Assert if power level value greater then PSCI_INVALID_PWR_LVL
  - Unsigned long should not be used as per coding guidelines
  - Reduce the number of memory leaks in cert_create
  - Fix type of cot_desc_ptr
  - Use explicit-width data types in AAPCS parameter structs
  - Add python configuration for editorconfig
  - BL1: Fix type consistency
  - Enable -Wshift-overflow=2 to check for undefined shift behavior

  - Updated upstream platforms to not rely on undefined overflow behaviour
- Coverity Quality Issues
  - Remove GGC ignore -Warray-bounds
  - Fix Coverity #261967, Infinite loop
  - Fix Coverity #343017, Missing unlock
  - Fix Coverity #343008, Side affect in assertion
  - Fix Coverity #342970, Uninitialized scalar variable
- CPU Support
  - cortex-a12: Fix MIDR mask
- Drivers
  - console: Remove Arm console unregister on suspend
  - gicv3: Fix support for full SPI range
  - scmi: Fix wrong payload length
- Library Code
  - libc: Fix sparse warning for __assert()
  - libc: Fix memchr implementation
- Platforms
  - rpi: rpi3: Fix compilation error when stack protector is enabled
  - socionext/uniphier: Fix compilation fail for SPM support build config
  - st/stm32mp1: Fix TZC400 configuration against non-secure DDR
  - ti/k3: common: Fix RO data area size calculation
- Security
  - AArch32: Disable Secure Cycle Counter
    * Changes the implementation for disabling Secure Cycle Counter. For ARMv8.5 the counter gets disabled by setting `SDCR.SCCD` bit on CPU cold/warm boot. For the earlier architectures PMCR register is saved/restored on secure world entry/exit from/to Non-secure state, and cycle counting gets disabled by setting PMCR.DP bit.
  - AArch64: Disable Secure Cycle Counter
    * For ARMv8.5 the counter gets disabled by setting `MDCR_El3.SCCD` bit on CPU cold/warm boot. For the earlier architectures PMCR_EL0 register is saved/restored on secure world entry/exit from/to Non-secure state, and cycle counting gets disabled by setting PMCR_EL0.DP bit.

14.14.4 Deprecations

- Common Code
  - Remove MULTI_CONSOLE_API flag and references to it
  - Remove deprecated `plat_crash_console_*`
  - Remove deprecated interfaces `get_afflvl_shift`, `mpidr_mask_lower_afflvls`, `eret`
  - AARCH32/AARCH64 macros are now deprecated in favor of `__aarch64__`
  - `__ASSEMBLY__` macro is now deprecated in favor of `__ASSEMBLER__`
- Drivers
  - console: Removed legacy console API
  - console: Remove deprecated finish_console_register
  - tzc: Remove deprecated types `tzc_action_t` and `tzc_region_attributes_t`
- Secure Partition Manager (SPM):
  - Prototype SPCI-based SPM (services/std_svc/spm) will be replaced with alternative methods of secure partitioning support.

14.14.5 Known Issues

- Build System Issues
  - dtb: DTB creation not supported when building on a Windows host.

    This step in the build process is skipped when running on a Windows host. A known issue from the 1.6 release.
- Platform Issues
  - arm/juno: System suspend from Linux does not function as documented in the user guide

    Following the instructions provided in the user guide document does not result in the platform entering system suspend state as expected. A message relating to the hdlcd driver failing to suspend will be emitted on the Linux terminal.
  - mediatek/mt6795: This platform does not build in this release

14.15 2.1.0 (2019-03-29)

14.15.1 New Features

- Architecture
  - Support for ARMv8.3 pointer authentication in the normal and secure worlds

    The use of pointer authentication in the normal world is enabled whenever architectural support is available, without the need for additional build flags.

    Use of pointer authentication in the secure world remains an experimental configuration at this time. Using both the `ENABLE_PAUTH` and `CTX_INCLUDE_PAUTH_REGS` build flags, pointer authentication can be enabled in EL3 and S-EL1/0.

    See the *Firmware Design* document for additional details on the use of pointer authentication.
  - Enable Data Independent Timing (DIT) in EL3, where supported
- Build System
  - Support for BL-specific build flags
  - Support setting compiler target architecture based on `ARM_ARCH_MINOR` build option.
  - New `RECLAIM_INIT_CODE` build flag:

    A significant amount of the code used for the initialization of BL31 is not needed again after boot time. In order to reduce the runtime memory footprint, the memory used for this code can be reclaimed after initialization.

    Certain boot-time functions were marked with the `__init` attribute to enable this reclamation.
- CPU Support
  - cortex-a76: Workaround for erratum 1073348
  - cortex-a76: Workaround for erratum 1220197
  - cortex-a76: Workaround for erratum 1130799
  - cortex-a75: Workaround for erratum 790748
  - cortex-a75: Workaround for erratum 764081
  - cortex-a73: Workaround for erratum 852427
  - cortex-a73: Workaround for erratum 855423
  - cortex-a57: Workaround for erratum 817169
  - cortex-a57: Workaround for erratum 814670
  - cortex-a55: Workaround for erratum 903758
  - cortex-a55: Workaround for erratum 846532
  - cortex-a55: Workaround for erratum 798797

- cortex-a55: Workaround for erratum 778703
- cortex-a55: Workaround for erratum 768277
- cortex-a53: Workaround for erratum 819472
- cortex-a53: Workaround for erratum 824069
- cortex-a53: Workaround for erratum 827319
- cortex-a17: Workaround for erratum 852423
- cortex-a17: Workaround for erratum 852421
- cortex-a15: Workaround for erratum 816470
- cortex-a15: Workaround for erratum 827671

- Documentation
  - Exception Handling Framework documentation
  - Library at ROM (romlib) documentation
  - RAS framework documentation
  - Coding Guidelines document
- Drivers
  - ccn: Add API for setting and reading node registers
    - Adds `ccn_read_node_reg` function
    - Adds `ccn_write_node_reg` function
  - partition: Support MBR partition entries
  - scmi: Add `plat_css_get_scmi_info` function

    Adds a new API `plat_css_get_scmi_info` which lets the platform register a platform-specific instance of `scmi_channel_plat_info_t` and remove the default values
  - tzc380: Add TZC-380 TrustZone Controller driver
  - tzc-dmc620: Add driver to manage the TrustZone Controller within the DMC-620 Dynamic Memory Controller
- Library at ROM (romlib)
  - Add platform-specific jump table list
  - Allow patching of romlib functions

    This change allows patching of functions in the romlib. This can be done by adding "patch" at the end of the jump table entry for the function that needs to be patched in the file jmptbl.i.
- Library Code
  - Support non-LPAE-enabled MMU tables in AArch32
  - mmio: Add `mmio_clrsetbits_16` function

      * 16-bit variant of `mmio_clrsetbits`

  - object_pool: Add Object Pool Allocator

      * Manages object allocation using a fixed-size static array
      * Adds `pool_alloc` and `pool_alloc_n` functions
      * Does not provide any functions to free allocated objects (by design)

  - libc: Added `strlcpy` function
  - libc: Import `strrchr` function from FreeBSD
  - xlat_tables: Add support for ARMv8.4-TTST
  - xlat_tables: Support mapping regions without an explicitly specified VA

- Math

  - Added softudiv macro to support software division

- Memory Partitioning And Monitoring (MPAM)

  - Enabled MPAM EL2 traps (`MPAMHCR_EL2` and `MPAM_EL2`)

- Platforms

  - amlogic: Add support for Meson S905 (GXBB)
  - arm/fvp_ve: Add support for FVP Versatile Express platform
  - arm/n1sdp: Add support for Neoverse N1 System Development platform
  - arm/rde1edge: Add support for Neoverse E1 platform
  - arm/rdn1edge: Add support for Neoverse N1 platform
  - arm: Add support for booting directly to Linux without an intermediate loader (AArch32)
  - arm/juno: Enable new CPU errata workarounds for A53 and A57
  - arm/juno: Add romlib support

    Building a combined BL1 and ROMLIB binary file with the correct page alignment is now supported on the Juno platform. When `USE_ROMLIB` is set for Juno, it generates the combined file `bl1_romlib.bin` which needs to be used instead of bl1.bin.

  - intel/stratix: Add support for Intel Stratix 10 SoC FPGA platform
  - marvell: Add support for Armada-37xx SoC platform
  - nxp: Add support for i.MX8M and i.MX7 Warp7 platforms
  - renesas: Add support for R-Car Gen3 platform
  - xilinx: Add support for Versal ACAP platforms

- Position-Independent Executable (PIE)

  PIE support has initially been added to BL31. The `ENABLE_PIE` build flag is used to enable or disable this functionality as required.

- Secure Partition Manager
  - New SPM implementation based on SPCI Alpha 1 draft specification

    A new version of SPM has been implemented, based on the SPCI (Secure Partition Client Interface) and SPRT (Secure Partition Runtime) draft specifications.

    The new implementation is a prototype that is expected to undergo intensive rework as the specifications change. It has basic support for multiple Secure Partitions and Resource Descriptions.

    The older version of SPM, based on MM (ARM Management Mode Interface Specification), is still present in the codebase. A new build flag, SPM_MM has been added to allow selection of the desired implementation. This flag defaults to 1, selecting the MM-based implementation.
- Security
  - Spectre Variant-1 mitigations (CVE-2017-5753)
  - Use Speculation Store Bypass Safe (SSBS) functionality where available

    Provides mitigation against CVE-2018-19440 (Not saving x0 to x3 registers can leak information from one Normal World SMC client to another)

14.15.2 Changed

- Build System
  - Warning levels are now selectable with W=<1,2,3>
  - Removed unneeded include paths in PLAT_INCLUDES
  - "Warnings as errors" (Werror) can be disabled using E=0
  - Support totally quiet output with -s flag
  - Support passing options to checkpatch using CHECKPATCH_OPTS=<opts>
  - Invoke host compiler with HOSTCC / HOSTCCFLAGS instead of CC / CFLAGS
  - Make device tree pre-processing similar to U-boot/Linux by:
    * Creating separate CPPFLAGS for DT preprocessing so that compiler options specific to it can be accommodated.
    * Replacing CPP with PP for DT pre-processing
- CPU Support
  - Errata report function definition is now mandatory for CPU support files

    CPU operation files must now define a <name>_errata_report function to print errata status. This is no longer a weak reference.
- Documentation
  - Migrated some content from GitHub wiki to docs/ directory
  - Security advisories now have CVE links

- Updated copyright guidelines

- Drivers
  - console: The `MULTI_CONSOLE_API` framework has been rewritten in C
  - console: Ported multi-console driver to AArch32
  - gic: Remove 'lowest priority' constants

    Removed `GIC_LOWEST_SEC_PRIORITY` and `GIC_LOWEST_NS_PRIORITY`. Platforms should define these if required, or instead determine the correct priority values at runtime.
  - delay_timer: Check that the Generic Timer extension is present
  - mmc: Increase command reply timeout to 10 milliseconds
  - mmc: Poll eMMC device status to ensure `EXT_CSD` command completion
  - mmc: Correctly check return code from `mmc_fill_device_info`

- External Libraries
  - libfdt: Upgraded from 1.4.2 to 1.4.6-9
  - mbed TLS: Upgraded from 2.12 to 2.16

  This change incorporates fixes for security issues that should be reviewed to determine if they are relevant for software implementations using Trusted Firmware-A. See the mbed TLS releases page for details on changes from the 2.12 to the 2.16 release.

- Library Code
  - compiler-rt: Updated `lshrdi3.c` and `int_lib.h` with changes from LLVM master branch (r345645)
  - cpu: Updated macro that checks need for `CVE-2017-5715` mitigation
  - libc: Made setjmp and longjmp C standard compliant
  - libc: Allowed overriding the default libc (use `OVERRIDE_LIBC`)
  - libc: Moved setjmp and longjmp to the `libc/` directory

- Platforms
  - Removed Mbed TLS dependency from plat_bl_common.c
  - arm: Removed unused `ARM_MAP_BL_ROMLIB` macro
  - arm: Removed `ARM_BOARD_OPTIMISE_MEM` feature and build flag
  - arm: Moved several components into `drivers/` directory

    This affects the SDS, SCP, SCPI, MHU and SCMI components
  - arm/juno: Increased maximum BL2 image size to `0xF000`

    This change was required to accommodate a larger `libfdt` library

- SCMI

- Optimized bakery locks when hardware-assisted coherency is enabled using the `HW_ASSISTED_COHERENCY` build flag

- SDEI
  - Added support for unconditionally resuming secure world execution after {{ SDEI }} event processing completes

    {{ SDEI }} interrupts, although targeting EL3, occur on behalf of the non-secure world, and may have higher priority than secure world interrupts. Therefore they might preempt secure execution and yield execution to the non-secure {{ SDEI }} handler. Upon completion of {{ SDEI }} event handling, resume secure execution if it was preempted.

- Translation Tables (XLAT)
  - Dynamically detect need for `Common not Private (TTBRn_ELx.CnP)` bit

    Properly handle the case where `ARMv8.2-TTCNP` is implemented in a CPU that does not implement all mandatory v8.2 features (and so must claim to implement a lower architecture version).

14.15.3 Resolved Issues

- Architecture
  - Incorrect check for SSBS feature detection
  - Unintentional register clobber in AArch32 reset_handler function
- Build System
  - Dependency issue during DTB image build
  - Incorrect variable expansion in Arm platform makefiles
  - Building on Windows with verbose mode (`V=1`) enabled is broken
  - AArch32 compilation flags is missing `$(march32-directive)`
- BL-Specific Issues
  - bl2: `uintptr_t is not defined` error when `BL2_IN_XIP_MEM` is defined
  - bl2: Missing prototype warning in `bl2_arch_setup`
  - bl31: Omission of Global Offset Table (GOT) section
- Code Quality Issues
  - Multiple MISRA compliance issues
  - Potential NULL pointer dereference (Coverity-detected)
- Drivers
  - mmc: Local declaration of `scr` variable causes a cache issue when invalidating after the read DMA transfer completes

- mmc: `ACMD41` does not send voltage information during initialization, resulting in the command being treated as a query. This prevents the command from initializing the controller.
- mmc: When checking device state using `mmc_device_state()` there are no retries attempted in the event of an error
- ccn: Incorrect Region ID calculation for RN-I nodes
- console: `Fix MULTI_CONSOLE_API` when used as a crash console
- partition: Improper NULL checking in gpt.c
- partition: Compilation failure in `VERBOSE` mode (`V=1`)

• Library Code

- common: Incorrect check for Address Authentication support
- xlat: Fix XLAT_V1 / XLAT_V2 incompatibility

  The file `arm_xlat_tables.h` has been renamed to `xlat_tables_compat.h` and has been moved to a common folder. This header can be used to guarantee compatibility, as it includes the correct header based on `XLAT_TABLES_LIB_V2`.
- xlat: armclang unused-function warning on `xlat_clean_dcache_range`
- xlat: Invalid `mm_cursor` checks in `mmap_add` and `mmap_add_ctx`
- sdei: Missing `context.h` header

• Platforms

- common: Missing prototype warning for `plat_log_get_prefix`
- arm: Insufficient maximum BL33 image size
- arm: Potential memory corruption during BL2-BL31 transition

  On Arm platforms, the BL2 memory can be overlaid by BL31/BL32. The memory descriptors describing the list of executable images are created in BL2 R/W memory, which could be possibly corrupted later on by BL31/BL32 due to overlay. This patch creates a reserved location in SRAM for these descriptors and are copied over by BL2 before handing over to next BL image.
- juno: Invalid behaviour when `CSS_USE_SCMI_SDS_DRIVER` is not set

  In `juno_pm.c` the `css_scmi_override_pm_ops` function was used regardless of whether the build flag was set. The original behaviour has been restored in the case where the build flag is not set.

• Tools

- fiptool: Incorrect UUID parsing of blob parameters
- doimage: Incorrect object rules in Makefile

14.15.4 Deprecations

- Common Code
  - `plat_crash_console_init` function
  - `plat_crash_console_putc` function
  - `plat_crash_console_flush` function
  - `finish_console_register` macro
- AArch64-specific Code
  - helpers: `get_afflvl_shift`
  - helpers: `mpidr_mask_lower_afflvls`
  - helpers: `eret`
- Secure Partition Manager (SPM)
  - Boot-info structure

14.15.5 Known Issues

- Build System Issues
  - dtb: DTB creation not supported when building on a Windows host.

    This step in the build process is skipped when running on a Windows host. A known issue from the 1.6 release.
- Platform Issues
  - arm/juno: System suspend from Linux does not function as documented in the user guide

    Following the instructions provided in the user guide document does not result in the platform entering system suspend state as expected. A message relating to the hdlcd driver failing to suspend will be emitted on the Linux terminal.
  - arm/juno: The firmware update use-cases do not work with motherboard firmware version < v1.5.0 (the reset reason is not preserved). The Linaro 18.04 release has MB v1.4.9. The MB v1.5.0 is available in Linaro 18.10 release.
  - mediatek/mt6795: This platform does not build in this release

14.16 2.0.0 (2018-10-02)

14.16.1 New Features

- Removal of a number of deprecated APIs
  - A new Platform Compatibility Policy document has been created which references a wiki page that maintains a listing of deprecated interfaces and the release after which they will be removed.
  - All deprecated interfaces except the MULTI_CONSOLE_API have been removed from the code base.
  - Various Arm and partner platforms have been updated to remove the use of removed APIs in this release.
  - This release is otherwise unchanged from 1.6 release

14.16.2 Issues resolved since last release

- No issues known at 1.6 release resolved in 2.0 release

14.16.3 Known Issues

- DTB creation not supported when building on a Windows host. This step in the build process is skipped when running on a Windows host. Known issue from 1.6 version.
- As a result of removal of deprecated interfaces the Nvidia Tegra, Marvell Armada 8K and MediaTek MT6795 platforms do not build in this release. Also MediaTek MT8173, NXP QorIQ LS1043A, NXP i.MX8QX, NXP i.MX8QMa, Rockchip RK3328, Rockchip RK3368 and Rockchip RK3399 platforms have not been confirmed to be working after the removal of the deprecated interfaces although they do build.

14.17 1.6.0 (2018-09-21)

14.17.1 New Features

- Addressing Speculation Security Vulnerabilities
  - Implement static workaround for CVE-2018-3639 for AArch32 and AArch64
  - Add support for dynamic mitigation for CVE-2018-3639
  - Implement dynamic mitigation for CVE-2018-3639 on Cortex-A76
  - Ensure {{ SDEI }} handler executes with CVE-2018-3639 mitigation enabled
- Introduce RAS handling on AArch64

  - Some RAS extensions are mandatory for Armv8.2 CPUs, with others mandatory for Armv8.4 CPUs however, all extensions are also optional extensions to the base Armv8.0 architecture.
  - The Armv8 RAS Extensions introduced Standard Error Records which are a set of standard registers to configure RAS node policy and allow RAS Nodes to record and expose error information for error handling agents.
  - Capabilities are provided to support RAS Node enumeration and iteration along with individual interrupt registrations and fault injections support.
  - Introduce handlers for Uncontainable errors, Double Faults and EL3 External Aborts
- Enable Memory Partitioning And Monitoring (MPAM) for lower EL's
  - Memory Partitioning And Monitoring is an Armv8.4 feature that enables various memory system components and resources to define partitions. Software running at various ELs can then assign themselves to the desired partition to control their performance aspects.
  - When ENABLE_MPAM_FOR_LOWER_ELS is set to 1, EL3 allows lower ELs to access their own MPAM registers without trapping to EL3. This patch however, doesn't make use of partitioning in EL3; platform initialisation code should configure and use partitions in EL3 if required.
- Introduce ROM Lib Feature
  - Support combining several libraries into a self-called "romlib" image, that may be shared across images to reduce memory footprint. The romlib image is stored in ROM but is accessed through a jump-table that may be stored in read-write memory, allowing for the library code to be patched.
- Introduce Backtrace Feature
  - This function displays the backtrace, the current EL and security state to allow a post-processing tool to choose the right binary to interpret the dump.
  - Print backtrace in assert() and panic() to the console.
- Code hygiene changes and alignment with MISRA C-2012 guideline with fixes addressing issues complying to the following rules:
  - MISRA rules 4.9, 5.1, 5.3, 5.7, 8.2-8.5, 8.8, 8.13, 9.3, 10.1, 10.3-10.4, 10.8, 11.3, 11.6, 12.1, 14.4, 15.7, 16.1-16.7, 17.7-17.8, 20.7, 20.10, 20.12, 21.1, 21.15, 22.7
  - Clean up the usage of void pointers to access symbols
  - Increase usage of static qualifier to locally used functions and data
  - Migrated to use of u_register_t for register read/write to better match AArch32 and AArch64 type sizes
  - Use int-ll64 for both AArch32 and AArch64 to assist in consistent format strings between architectures
  - Clean up TF-A libc by removing non arm copyrighted implementations and replacing them with modified FreeBSD and SCC implementations
- Various changes to support Clang linker and assembler

  - The clang assembler/preprocessor is used when Clang is selected. However, the clang linker is not used because it is unable to link TF-A objects due to immaturity of clang linker functionality at this time.
- Refactor support APIs into Libraries
  - Evolve libfdt, mbed TLS library and standard C library sources as proper libraries that TF-A may be linked against.
- CPU Enhancements
  - Add CPU support for Cortex-Ares and Cortex-A76
  - Add AMU support for Cortex-Ares
  - Add initial CPU support for Cortex-Deimos
  - Add initial CPU support for Cortex-Helios
  - Implement dynamic mitigation for CVE-2018-3639 on Cortex-A76
  - Implement Cortex-Ares erratum 1043202 workaround
  - Implement DSU erratum 936184 workaround
  - Check presence of fix for errata 843419 in Cortex-A53
  - Check presence of fix for errata 835769 in Cortex-A53
- Translation Tables Enhancements
  - The xlat v2 library has been refactored in order to be reused by different TF components at different EL's including the addition of EL2. Some refactoring to make the code more generic and less specific to TF, in order to reuse the library outside of this project.
- SPM Enhancements
  - General cleanups and refactoring to pave the way to multiple partitions support
- SDEI Enhancements
  - Allow platforms to define explicit events
  - Determine client EL from NS context's SCR_EL3
  - Make dispatches synchronous
  - Introduce jump primitives for BL31
  - Mask events after CPU wakeup in {{ SDEI }} dispatcher to conform to the specification
- Misc TF-A Core Common Code Enhancements
  - Add support for eXecute In Place (XIP) memory in BL2
  - Add support for the SMC Calling Convention 2.0
  - Introduce External Abort handling on AArch64 External Abort routed to EL3 was reported as an unhandled exception and caused a panic. This change enables Trusted Firmware-A to handle External Aborts routed to EL3.

  - Save value of ACTLR_EL1 implementation-defined register in the CPU context structure rather than forcing it to 0.
  - Introduce ARM_LINUX_KERNEL_AS_BL33 build option, which allows BL31 to directly jump to a Linux kernel. This makes for a quicker and simpler boot flow, which might be useful in some test environments.
  - Add dynamic configurations for BL31, BL32 and BL33 enabling support for Chain of Trust (COT).
  - Make TF UUID RFC 4122 compliant
- New Platform Support
  - Arm SGI-575
  - Arm SGM-775
  - Allwinner sun50i_64
  - Allwinner sun50i_h6
  - NXP QorIQ LS1043A
  - NXP i.MX8QX
  - NXP i.MX8QM
  - NXP i.MX7Solo WaRP7
  - TI K3
  - Socionext Synquacer SC2A11
  - Marvell Armada 8K
  - STMicroelectronics STM32MP1
- Misc Generic Platform Common Code Enhancements
  - Add MMC framework that supports both eMMC and SD card devices
- Misc Arm Platform Common Code Enhancements
  - Demonstrate PSCI MEM_PROTECT from el3_runtime
  - Provide RAS support
  - Migrate AArch64 port to the multi console driver. The old API is deprecated and will eventually be removed.
  - Move BL31 below BL2 to enable BL2 overlay resulting in changes in the layout of BL images in memory to enable more efficient use of available space.
  - Add cpp build processing for dtb that allows processing device tree with external includes.
  - Extend FIP io driver to support multiple FIP devices
  - Add support for SCMI AP core configuration protocol v1.0
  - Use SCMI AP core protocol to set the warm boot entrypoint

  - Add support to Mbed TLS drivers for shared heap among different BL images to help optimise memory usage
  - Enable non-secure access to UART1 through a build option to support a serial debug port for debugger connection
- Enhancements for Arm Juno Platform
  - Add support for TrustZone Media Protection 1 (TZMP1)
- Enhancements for Arm FVP Platform
  - Dynamic_config: remove the FVP dtb files
  - Set DYNAMIC_WORKAROUND_CVE_2018_3639=1 on FVP by default
  - Set the ability to dynamically disable Trusted Boot Board authentication to be off by default with DYN_DISABLE_AUTH
  - Add librom enhancement support in FVP
  - Support shared Mbed TLS heap between BL1 and BL2 that allow a reduction in BL2 size for FVP
- Enhancements for Arm SGI/SGM Platform
  - Enable ARM_PLAT_MT flag for SGI-575
  - Add dts files to enable support for dynamic config
  - Add RAS support
  - Support shared Mbed TLS heap for SGI and SGM between BL1 and BL2
- Enhancements for Non Arm Platforms
  - Raspberry Pi Platform
  - Hikey Platforms
  - Xilinx Platforms
  - QEMU Platform
  - Rockchip rk3399 Platform
  - TI Platforms
  - Socionext Platforms
  - Allwinner Platforms
  - NXP Platforms
  - NVIDIA Tegra Platform
  - Marvell Platforms
  - STMicroelectronics STM32MP1 Platform

14.17.2 Issues resolved since last release

- No issues known at 1.5 release resolved in 1.6 release

14.17.3 Known Issues

- DTB creation not supported when building on a Windows host. This step in the build process is skipped when running on a Windows host. Known issue from 1.5 version.

14.18 1.5.0 (2018-03-20)

14.18.1 New features

- Added new firmware support to enable RAS (Reliability, Availability, and Serviceability) functionality.
  - Secure Partition Manager (SPM): A Secure Partition is a software execution environment instantiated in S-EL0 that can be used to implement simple management and security services. The SPM is the firmware component that is responsible for managing a Secure Partition.
  - SDEI dispatcher: Support for interrupt-based {{ SDEI }} events and all interfaces as defined by the {{ SDEI }} specification v1.0, see SDEI Specification
  - Exception Handling Framework (EHF): Framework that allows dispatching of EL3 interrupts to their registered handlers which are registered based on their priorities. Facilitates firmware-first error handling policy where asynchronous exceptions may be routed to EL3.

    Integrated the TSPD with EHF.
- Updated PSCI support:
  - Implemented PSCI v1.1 optional features `MEM_PROTECT` and `SYSTEM_RESET2`. The supported PSCI version was updated to v1.1.
  - Improved PSCI STAT timestamp collection, including moving accounting for retention states to be inside the locks and fixing handling of wrap-around when calculating residency in AArch32 execution state.
  - Added optional handler for early suspend that executes when suspending to a power-down state and with data caches enabled.

    This may provide a performance improvement on platforms where it is safe to perform some or all of the platform actions from `pwr_domain_suspend` with the data caches enabled.
- Enabled build option, BL2_AT_EL3, for BL2 to allow execution at EL3 without any dependency on TF BL1.

  This allows platforms which already have a non-TF Boot ROM to directly load and execute BL2 and subsequent BL stages without need for BL1. This was not previously possible because BL2 executes at S-EL1 and cannot jump straight to EL3.

- Implemented support for SMCCC v1.1, including `SMCCC_VERSION` and `SMCCC_ARCH_FEATURES`.

  Additionally, added support for `SMCCC_VERSION` in PSCI features to enable discovery of the SMCCC version via PSCI feature call.

- Added Dynamic Configuration framework which enables each of the boot loader stages to be dynamically configured at runtime if required by the platform. The boot loader stage may optionally specify a firmware configuration file and/or hardware configuration file that can then be shared with the next boot loader stage.

  Introduced a new BL handover interface that essentially allows passing of 4 arguments between the different BL stages.

  Updated cert_create and fip_tool to support the dynamic configuration files. The COT also updated to support these new files.

- Code hygiene changes and alignment with MISRA guideline:
  - Fix use of undefined macros.
  - Achieved compliance with Mandatory MISRA coding rules.
  - Achieved compliance for following Required MISRA rules for the default build configurations on FVP and Juno platforms : 7.3, 8.3, 8.4, 8.5 and 8.8.
- Added support for Armv8.2-A architectural features:
  - Updated translation table set-up to set the CnP (Common not Private) bit for secure page tables so that multiple PEs in the same Inner Shareable domain can use the same translation table entries for a given stage of translation in a particular translation regime.
  - Extended the supported values of ID_AA64MMFR0_EL1.PARange to include the 52-bit Physical Address range.
  - Added support for the Scalable Vector Extension to allow Normal world software to access SVE functionality but disable access to SVE, SIMD and floating point functionality from the Secure world in order to prevent corruption of the Z-registers.
- Added support for Armv8.4-A architectural feature Activity Monitor Unit (AMU)

  extensions.

  In addition to the v8.4 architectural extension, AMU support on Cortex-A75 was implemented.

- Enhanced OP-TEE support to enable use of pageable OP-TEE image. The Arm standard platforms are updated to load up to 3 images for OP-TEE; header, pager image and paged image.

  The chain of trust is extended to support the additional images.

- Enhancements to the translation table library:
  - Introduced APIs to get and set the memory attributes of a region.
  - Added support to manage both privilege levels in translation regimes that describe translations for 2 Exception levels, specifically the EL1&0 translation regime, and extended the memory map region attributes to include specifying Non-privileged access.

- Added support to specify the granularity of the mappings of each region, for instance a 2MB region can be specified to be mapped with 4KB page tables instead of a 2MB block.
- Disabled the higher VA range to avoid unpredictable behaviour if there is an attempt to access addresses in the higher VA range.
- Added helpers for Device and Normal memory MAIR encodings that align with the Arm Architecture Reference Manual for Armv8-A (Arm DDI0487B.b).
- Code hygiene including fixing type length and signedness of constants, refactoring of function to enable the MMU, removing all instances where the virtual address space is hardcoded and added comments that document alignment needed between memory attributes and attributes specified in TCR_ELx.

- Updated GIC support:
  - Introduce new APIs for GICv2 and GICv3 that provide the capability to specify interrupt properties rather than list of interrupt numbers alone. The Arm platforms and other upstream platforms are migrated to use interrupt properties.
  - Added helpers to save / restore the GICv3 context, specifically the Distributor and Redistributor contexts and architectural parts of the ITS power management. The Distributor and Redistributor helpers also support the implementation-defined part of GIC-500 and GIC-600.

    Updated the Arm FVP platform to save / restore the GICv3 context on system suspend / resume as an example of how to use the helpers.

    Introduced a new TZC secured DDR carve-out for use by Arm platforms for storing EL3 runtime data such as the GICv3 register context.
- Added support for Armv7-A architecture via build option ARM_ARCH_MAJOR=7. This includes following features:
  - Updates GICv2 driver to manage GICv1 with security extensions.
  - Software implementation for 32bit division.
  - Enabled use of generic timer for platforms that do not set ARM_CORTEX_Ax=yes.
  - Support for Armv7-A Virtualization extensions [DDI0406C_C].
  - Support for both Armv7-A platforms that only have 32-bit addressing and Armv7-A platforms that support large page addressing.
  - Included support for following Armv7 CPUs: Cortex-A12, Cortex-A17, Cortex-A7, Cortex-A5, Cortex-A9, Cortex-A15.
  - Added support in QEMU for Armv7-A/Cortex-A15.
- Enhancements to Firmware Update feature:
  - Updated the FWU documentation to describe the additional images needed for Firmware update, and how they are used for both the Juno platform and the Arm FVP platforms.
- Enhancements to Trusted Board Boot feature:
  - Added support to cert_create tool for RSA PKCS1# v1.5 and SHA384, SHA512 and SHA256.

  - For Arm platforms added support to use ECDSA keys.
  - Enhanced the mbed TLS wrapper layer to include support for both RSA and ECDSA to enable runtime selection between RSA and ECDSA keys.
- Added support for secure interrupt handling in AArch32 sp_min, hardcoded to only handle FIQs.
- Added support to allow a platform to load images from multiple boot sources, for example from a second flash drive.
- Added a logging framework that allows platforms to reduce the logging level at runtime and additionally the prefix string can be defined by the platform.
- Further improvements to register initialisation:
  - Control register PMCR_EL0 / PMCR is set to prohibit cycle counting in the secure world. This register is added to the list of registers that are saved and restored during world switch.
  - When EL3 is running in AArch32 execution state, the Non-secure version of SCTLR is explicitly initialised during the warmboot flow rather than relying on the hardware to set the correct reset values.
- Enhanced support for Arm platforms:
  - Introduced driver for Shared-Data-Structure (SDS) framework which is used for communication between SCP and the AP CPU, replacing Boot-Over_MHU (BOM) protocol.

    The Juno platform is migrated to use SDS with the SCMI support added in v1.3 and is set as default.

    The driver can be found in the plat/arm/css/drivers folder.
  - Improved memory usage by only mapping TSP memory region when the TSPD has been included in the build. This reduces the memory footprint and avoids unnecessary memory being mapped.
  - Updated support for multi-threading CPUs for FVP platforms - always check the MT field in MPDIR and access the bit fields accordingly.
  - Support building for platforms that model DynamIQ configuration by implementing all CPUs in a single cluster.
  - Improved nor flash driver, for instance clearing status registers before sending commands. Driver can be found plat/arm/board/common folder.
- Enhancements to QEMU platform:
  - Added support for TBB.
  - Added support for using OP-TEE pageable image.
  - Added support for LOAD_IMAGE_V2.
  - Migrated to use translation table library v2 by default.
  - Added support for SEPARATE_CODE_AND_RODATA.
- Applied workarounds CVE-2017-5715 on Arm Cortex-A57, -A72, -A73 and -A75, and for Armv7-A CPUs Cortex-A9, -A15 and -A17.
- Applied errata workaround for Arm Cortex-A57: 859972.

- Applied errata workaround for Arm Cortex-A72: 859971.
- Added support for Poplar 96Board platform.
- Added support for Raspberry Pi 3 platform.
- Added Call Frame Information (CFI) assembler directives to the vector entries which enables debuggers to display the backtrace of functions that triggered a synchronous abort.
- Added ability to build dtb.
- Added support for pre-tool (cert_create and fiptool) image processing enabling compression of the image files before processing by cert_create and fiptool.

  This can reduce fip size and may also speed up loading of images. The image verification will also get faster because certificates are generated based on compressed images.

  Imported zlib 1.2.11 to implement gunzip() for data compression.
- Enhancements to fiptool:
  - Enabled the fiptool to be built using Visual Studio.
  - Added padding bytes at the end of the last image in the fip to be facilitate transfer by DMA.

14.18.2 Issues resolved since last release

- TF-A can be built with optimisations disabled (-O0).
- Memory layout updated to enable Trusted Board Boot on Juno platform when running TF-A in AArch32 execution mode (resolving tf-issue#501).

14.18.3 Known Issues

- DTB creation not supported when building on a Windows host. This step in the build process is skipped when running on a Windows host.

14.19 1.4.0 (2017-07-07)

14.19.1 New features

- Enabled support for platforms with hardware assisted coherency.

  A new build option HW_ASSISTED_COHERENCY allows platforms to take advantage of the following optimisations:
  - Skip performing cache maintenance during power-up and power-down.
  - Use spin-locks instead of bakery locks.
  - Enable data caches early on warm-booted CPUs.

- Added support for Cortex-A75 and Cortex-A55 processors.

  Both Cortex-A75 and Cortex-A55 processors use the Arm DynamIQ Shared Unit (DSU). The power-down and power-up sequences are therefore mostly managed in hardware, reducing complexity of the software operations.

- Introduced Arm GIC-600 driver.

  Arm GIC-600 IP complies with Arm GICv3 architecture. For FVP platforms, the GIC-600 driver is chosen when FVP_USE_GIC_DRIVER is set to FVP_GIC600.

- Updated GICv3 support:
  - Introduced power management APIs for GICv3 Redistributor. These APIs allow platforms to power down the Redistributor during CPU power on/off. Requires the GICv3 implementations to have power management operations.

    Implemented the power management APIs for FVP.
  - GIC driver data is flushed by the primary CPU so that secondary CPU do not read stale GIC data.

- Added support for Arm System Control and Management Interface v1.0 (SCMI).

  The SCMI driver implements the power domain management and system power management protocol of the SCMI specification (Arm DEN 0056ASCMI) for communicating with any compliant power controller.

  Support is added for the Juno platform. The driver can be found in the plat/arm/css/drivers folder.

- Added support to enable pre-integration of TBB with the Arm TrustZone CryptoCell product, to take advantage of its hardware Root of Trust and crypto acceleration services.

- Enabled Statistical Profiling Extensions for lower ELs.

  The firmware support is limited to the use of SPE in the Non-secure state and accesses to the SPE specific registers from S-EL1 will trap to EL3.

  The SPE are architecturally specified for AArch64 only.

- Code hygiene changes aligned with MISRA guidelines:
  - Fixed signed / unsigned comparison warnings in the translation table library.
  - Added U(_x) macro and together with the existing ULL(_x) macro fixed some of the signed-ness defects flagged by the MISRA scanner.

- Enhancements to Firmware Update feature:
  - The FWU logic now checks for overlapping images to prevent execution of unauthenticated arbitrary code.
  - Introduced new FWU_SMC_IMAGE_RESET SMC that changes the image loading state machine to go from COPYING, COPIED or AUTHENTICATED states to RESET state. Previously, this was only possible when the authentication of an image failed or when the execution of the image finished.
  - Fixed integer overflow which addressed TFV-1: Malformed Firmware Update SMC can result in copy of unexpectedly large data into secure memory.

- Introduced support for Arm Compiler 6 and LLVM (clang).

  TF-A can now also be built with the Arm Compiler 6 or the clang compilers. The assembler and linker must be provided by the GNU toolchain.

  Tested with Arm CC 6.7 and clang 3.9.x and 4.0.x.

- Memory footprint improvements:

  – Introduced `tf_snprintf`, a reduced version of `snprintf` which has support for a limited set of formats.

    The mbedtls driver is updated to optionally use `tf_snprintf` instead of `snprintf`.

  – The `assert()` is updated to no longer print the function name, and additional logging options are supported via an optional platform define `PLAT_LOG_LEVEL_ASSERT`, which controls how verbose the assert output is.

- Enhancements to TF-A support when running in AArch32 execution state:

  – Support booting SP_MIN and BL33 in AArch32 execution mode on Juno. Due to hardware limitations, BL1 and BL2 boot in AArch64 state and there is additional trampoline code to warm reset into SP_MIN in AArch32 execution state.

  – Added support for Arm Cortex-A53/57/72 MPCore processors including the errata workarounds that are already implemented for AArch64 execution state.

  – For FVP platforms, added AArch32 Trusted Board Boot support, including the Firmware Update feature.

- Introduced Arm SiP service for use by Arm standard platforms.

  – Added new Arm SiP Service SMCs to enable the Non-secure world to read PMF timestamps.

    Added PMF instrumentation points in TF-A in order to quantify the overall time spent in the PSCI software implementation.

  – Added new Arm SiP service SMC to switch execution state.

    This allows the lower exception level to change its execution state from AArch64 to AArch32, or vice verse, via a request to EL3.

- Migrated to use SPDX[0] license identifiers to make software license auditing simpler.

  :::{note} Files that have been imported by FreeBSD have not been modified. :::

  [0]: https://spdx.org/

- Enhancements to the translation table library:

  – Added version 2 of translation table library that allows different translation tables to be modified by using different 'contexts'. Version 1 of the translation table library only allows the current EL's translation tables to be modified.

    Version 2 of the translation table also added support for dynamic regions; regions that can be added and removed dynamically whilst the MMU is enabled. Static regions can only be added or removed before the MMU is enabled.

    The dynamic mapping functionality is enabled or disabled when compiling by setting the build option PLAT_XLAT_TABLES_DYNAMIC to 1 or 0. This can be done per-image.

  - Added support for translation regimes with two virtual address spaces such as the one shared by EL1 and EL0.

    The library does not support initializing translation tables for EL0 software.

  - Added support to mark the translation tables as non-cacheable using an additional build option `XLAT_TABLE_NC`.

- Added support for GCC stack protection. A new build option ENABLE_STACK_PROTECTOR was introduced that enables compilation of all BL images with one of the GCC -fstack-protector-* options.

  A new platform function plat_get_stack_protector_canary() was introduced that returns a value used to initialize the canary for stack corruption detection. For increased effectiveness of protection platforms must provide an implementation that returns a random value.

- Enhanced support for Arm platforms:

  - Added support for multi-threading CPUs, indicated by `MT` field in MPDIR. A new build flag `ARM_PLAT_MT` is added, and when enabled, the functions accessing MPIDR assume that the `MT` bit is set for the platform and access the bit fields accordingly.

    Also, a new API `plat_arm_get_cpu_pe_count` is added when `ARM_PLAT_MT` is enabled, returning the Processing Element count within the physical CPU corresponding to `mpidr`.

  - The Arm platforms migrated to use version 2 of the translation tables.

  - Introduced a new Arm platform layer API `plat_arm_psci_override_pm_ops` which allows Arm platforms to modify `plat_arm_psci_pm_ops` and therefore dynamically define PSCI capability.

  - The Arm platforms migrated to use IMAGE_LOAD_V2 by default.

- Enhanced reporting of errata workaround status with the following policy:

  - If an errata workaround is enabled:

    * If it applies (i.e. the CPU is affected by the errata), an INFO message is printed, confirming that the errata workaround has been applied.

    * If it does not apply, a VERBOSE message is printed, confirming that the errata workaround has been skipped.

  - If an errata workaround is not enabled, but would have applied had it been, a WARN message is printed, alerting that errata workaround is missing.

- Added build options ARM_ARCH_MAJOR and ARM_ARM_MINOR to choose the architecture version to target TF-A.

- Updated the spin lock implementation to use the more efficient CAS (Compare And Swap) instruction when available. This instruction was introduced in Armv8.1-A.

- Applied errata workaround for Arm Cortex-A53: 855873.

- Applied errata workaround for Arm-Cortex-A57: 813419.

- Enabled all A53 and A57 errata workarounds for Juno, both in AArch64 and AArch32 execution states.
- Added support for Socionext UniPhier SoC platform.
- Added support for Hikey960 and Hikey platforms.
- Added support for Rockchip RK3328 platform.
- Added support for NVidia Tegra T186 platform.
- Added support for Designware emmc driver.
- Imported libfdt v1.4.2 that addresses buffer overflow in fdt_offset_ptr().
- Enhanced the CPU operations framework to allow power handlers to be registered on per-level basis. This enables support for future CPUs that have multiple threads which might need powering down individually.
- Updated register initialisation to prevent unexpected behaviour:
  - Debug registers MDCR-EL3/SDCR and MDCR_EL2/HDCR are initialised to avoid unexpected traps into the higher exception levels and disable secure self-hosted debug. Additionally, secure privileged external debug on Juno is disabled by programming the appropriate Juno SoC registers.
  - EL2 and EL3 configurable controls are initialised to avoid unexpected traps in the higher exception levels.
  - Essential control registers are fully initialised on EL3 start-up, when initialising the non-secure and secure context structures and when preparing to leave EL3 for a lower EL. This gives better alignment with the Arm ARM which states that software must initialise RES0 and RES1 fields with 0 / 1.
- Enhanced PSCI support:
  - Introduced new platform interfaces that decouple PSCI stat residency calculation from PMF, enabling platforms to use alternative methods of capturing timestamps.
  - PSCI stat accounting performed for retention/standby states when requested at multiple power levels.
- Simplified fiptool to have a single linked list of image descriptors.
- For the TSP, resolved corruption of pre-empted secure context by aborting any pre-empted SMC during PSCI power management requests.

14.19.2 Issues resolved since last release

- TF-A can be built with the latest mbed TLS version (v2.4.2). The earlier version 2.3.0 cannot be used due to build warnings that the TF-A build system interprets as errors.
- TBBR, including the Firmware Update feature is now supported on FVP platforms when running TF-A in AArch32 state.
- The version of the AEMv8 Base FVP used in this release has resolved the issue of the model executing a reset instead of terminating in response to a shutdown request using the PSCI SYSTEM_OFF API.

14.19.3 Known Issues

- Building TF-A with compiler optimisations disabled (-O0) fails.
- Trusted Board Boot currently does not work on Juno when running Trusted Firmware in AArch32 execution state due to error when loading the sp_min to memory because of lack of free space available. See tf-issue#501 for more details.
- The errata workaround for A53 errata 843419 is only available from binutils 2.26 and is not present in GCC4.9. If this errata is applicable to the platform, please use GCC compiler version of at least 5.0. See PR#1002 for more details.

14.20 1.3.0 (2016-10-13)

14.20.1 New features

- Added support for running TF-A in AArch32 execution state.

  The PSCI library has been refactored to allow integration with **EL3 Runtime Software**. This is software that is executing at the highest secure privilege which is EL3 in AArch64 or Secure SVC/Monitor mode in AArch32. See {ref}`Porting Guide`.

  Included is a minimal AArch32 Secure Payload, **SP-MIN**, that illustrates the usage and integration of the PSCI library with EL3 Runtime Software running in AArch32 state.

  Booting to the BL1/BL2 images as well as booting straight to the Secure Payload is supported.

- Improvements to the initialization framework for the PSCI service and Arm Standard Services in general.

  The PSCI service is now initialized as part of Arm Standard Service initialization. This consolidates the initializations of any Arm Standard Service that may be added in the future.

  A new function `get_arm_std_svc_args()` is introduced to get arguments corresponding to each standard service and must be implemented by the EL3 Runtime Software.

  For PSCI, a new versioned structure `psci_lib_args_t` is introduced to initialize the PSCI Library. **Note** this is a compatibility break due to the change in the prototype of `psci_setup()`.

- To support AArch32 builds of BL1 and BL2, implemented a new, alternative firmware image loading mechanism that adds flexibility.

  The current mechanism has a hard-coded set of images and execution order (BL31, BL32, etc). The new mechanism is data-driven by a list of image descriptors provided by the platform code.

  Arm platforms have been updated to support the new loading mechanism.

  The new mechanism is enabled by a build flag (`LOAD_IMAGE_V2`) which is currently off by default for the AArch64 build.

  Note `TRUSTED_BOARD_BOOT` is currently not supported when `LOAD_IMAGE_V2` is enabled.

- Updated requirements for making contributions to TF-A.

Commits now must have a 'Signed-off-by:' field to certify that the contribution has been made under the terms of the `Developer Certificate of Origin`.

A signed CLA is no longer required.

The *Contributor's Guide* has been updated to reflect this change.

- Introduced Performance Measurement Framework (PMF) which provides support for capturing, storing, dumping and retrieving time-stamps to measure the execution time of critical paths in the firmware. This relies on defining fixed sample points at key places in the code.
- To support the QEMU platform port, imported libfdt v1.4.1 from https://git.kernel.org/pub/scm/utils/dtc/dtc.git
- Updated PSCI support:
  - Added support for PSCI NODE_HW_STATE API for Arm platforms.
  - New optional platform hook, `pwr_domain_pwr_down_wfi()`, in `plat_psci_ops` to enable platforms to perform platform-specific actions needed to enter powerdown, including the 'wfi' invocation.
  - PSCI STAT residency and count functions have been added on Arm platforms by using PMF.
- Enhancements to the translation table library:
  - Limited memory mapping support for region overlaps to only allow regions to overlap that are identity mapped or have the same virtual to physical address offset, and overlap completely but must not cover the same area.

    This limitation will enable future enhancements without having to support complex edge cases that may not be necessary.
  - The initial translation lookup level is now inferred from the virtual address space size. Previously, it was hard-coded.
  - Added support for mapping Normal, Inner Non-cacheable, Outer Non-cacheable memory in the translation table library.

    This can be useful to map a non-cacheable memory region, such as a DMA buffer.
  - Introduced the MT_EXECUTE/MT_EXECUTE_NEVER memory mapping attributes to specify the access permissions for instruction execution of a memory region.
- Enabled support to isolate code and read-only data on separate memory pages, allowing independent access control to be applied to each.
- Enabled SCR_EL3.SIF (Secure Instruction Fetch) bit in BL1 and BL31 common architectural setup code, preventing fetching instructions from non-secure memory when in secure state.
- Enhancements to FIP support:
  - Replaced `fip_create` with `fiptool` which provides a more consistent and intuitive interface as well as additional support to remove an image from a FIP file.
  - Enabled printing the SHA256 digest with info command, allowing quick verification of an image within a FIP without having to extract the image and running sha256sum on it.

  - Added support for unpacking the contents of an existing FIP file into the working directory.
  - Aligned command line options for specifying images to use same naming convention as specified by TBBR and already used in cert_create tool.
- Refactored the TZC-400 driver to also support memory controllers that integrate TZC functionality, for example Arm CoreLink DMC-500. Also added DMC-500 specific support.
- Implemented generic delay timer based on the system generic counter and migrated all platforms to use it.
- Enhanced support for Arm platforms:
  - Updated image loading support to make SCP images (SCP_BL2 and SCP_BL2U) optional.
  - Enhanced topology description support to allow multi-cluster topology definitions.
  - Added interconnect abstraction layer to help platform ports select the right interconnect driver, CCI or CCN, for the platform.
  - Added support to allow loading BL31 in the TZC-secured DRAM instead of the default secure SRAM.
  - Added support to use a System Security Control (SSC) Registers Unit enabling TF-A to be compiled to support multiple Arm platforms and then select one at runtime.
  - Restricted mapping of Trusted ROM in BL1 to what is actually needed by BL1 rather than entire Trusted ROM region.
  - Flash is now mapped as execute-never by default. This increases security by restricting the executable region to what is strictly needed.
- Applied following erratum workarounds for Cortex-A57: 833471, 826977, 829520, 828024 and 826974.
- Added support for Mediatek MT6795 platform.
- Added support for QEMU virtualization Armv8-A target.
- Added support for Rockchip RK3368 and RK3399 platforms.
- Added support for Xilinx Zynq UltraScale+ MPSoC platform.
- Added support for Arm Cortex-A73 MPCore Processor.
- Added support for Arm Cortex-A72 processor.
- Added support for Arm Cortex-A35 processor.
- Added support for Arm Cortex-A32 MPCore Processor.
- Enabled preloaded BL33 alternative boot flow, in which BL2 does not load BL33 from non-volatile storage and BL31 hands execution over to a preloaded BL33. The User Guide has been updated with an example of how to use this option with a bootwrapped kernel.
- Added support to build TF-A on a Windows-based host machine.
- Updated Trusted Board Boot prototype implementation:

- Enabled the ability for a production ROM with TBBR enabled to boot test software before a real ROTPK is deployed (e.g. manufacturing mode). Added support to use ROTPK in certificate without verifying against the platform value when `ROTPK_NOT_DEPLOYED` bit is set.
- Added support for non-volatile counter authentication to the Authentication Module to protect against roll-back.

- Updated GICv3 support:
  - Enabled processor power-down and automatic power-on using GICv3.
  - Enabled G1S or G0 interrupts to be configured independently.
  - Changed FVP default interrupt driver to be the GICv3-only driver. **Note** the default build of TF-A will not be able to boot Linux kernel with GICv2 FDT blob.
  - Enabled wake-up from CPU_SUSPEND to stand-by by temporarily re-routing interrupts and then restoring after resume.

14.20.2 Issues resolved since last release

14.20.3 Known issues

- The version of the AEMv8 Base FVP used in this release resets the model instead of terminating its execution in response to a shutdown request using the PSCI `SYSTEM_OFF` API. This issue will be fixed in a future version of the model.
- Building TF-A with compiler optimisations disabled (`-O0`) fails.
- TF-A cannot be built with mbed TLS version v2.3.0 due to build warnings that the TF-A build system interprets as errors.
- TBBR is not currently supported when running TF-A in AArch32 state.

14.21 1.2.0 (2015-12-22)

14.21.1 New features

- The Trusted Board Boot implementation on Arm platforms now conforms to the mandatory requirements of the TBBR specification.

  In particular, the boot process is now guarded by a Trusted Watchdog, which will reset the system in case of an authentication or loading error. On Arm platforms, a secure instance of Arm SP805 is used as the Trusted Watchdog.

  Also, a firmware update process has been implemented. It enables authenticated firmware to update firmware images from external interfaces to SoC Non-Volatile memories. This feature functions even when the current firmware in the system is corrupt or missing; it therefore may be used as a recovery mode.

- Improvements have been made to the Certificate Generation Tool (`cert_create`) as follows.

- Added support for the Firmware Update process by extending the Chain of Trust definition in the tool to include the Firmware Update certificate and the required extensions.
- Introduced a new API that allows one to specify command line options in the Chain of Trust description. This makes the declaration of the tool's arguments more flexible and easier to extend.
- The tool has been reworked to follow a data driven approach, which makes it easier to maintain and extend.

- Extended the FIP tool (`fip_create`) to support the new set of images involved in the Firmware Update process.
- Various memory footprint improvements. In particular:
  - The bakery lock structure for coherent memory has been optimised.
  - The mbed TLS SHA1 functions are not needed, as SHA256 is used to generate the certificate signature. Therefore, they have been compiled out, reducing the memory footprint of BL1 and BL2 by approximately 6 KB.
  - On Arm development platforms, each BL stage now individually defines the number of regions that it needs to map in the MMU.
- Added the following new design documents:
  - *Authentication Framework & Chain of Trust*
  - *Firmware Update (FWU)*
  - *CPU Reset*
  - *PSCI Power Domain Tree Structure*
- Applied the new image terminology to the code base and documentation, as described in the *Image Terminology* document.
- The build system has been reworked to improve readability and facilitate adding future extensions.
- On Arm standard platforms, BL31 uses the boot console during cold boot but switches to the runtime console for any later logs at runtime. The TSP uses the runtime console for all output.
- Implemented a basic NOR flash driver for Arm platforms. It programs the device using CFI (Common Flash Interface) standard commands.
- Implemented support for booting EL3 payloads on Arm platforms, which reduces the complexity of developing EL3 baremetal code by doing essential baremetal initialization.
- Provided separate drivers for GICv3 and GICv2. These expect the entire software stack to use either GICv2 or GICv3; hybrid GIC software systems are no longer supported and the legacy Arm GIC driver has been deprecated.
- Added support for Juno r1 and r2. A single set of Juno TF-A binaries can run on Juno r0, r1 and r2 boards. Note that this TF-A version depends on a Linaro release that does *not* contain Juno r2 support.
- Added support for MediaTek mt8173 platform.
- Implemented a generic driver for Arm CCN IP.
- Major rework of the PSCI implementation.

  - Added framework to handle composite power states.
  - Decoupled the notions of affinity instances (which describes the hierarchical arrangement of cores) and of power domain topology, instead of assuming a one-to-one mapping.
  - Better alignment with version 1.0 of the PSCI specification.
- Added support for the SYSTEM_SUSPEND PSCI API on Arm platforms. When invoked on the last running core on a supported platform, this puts the system into a low power mode with memory retention.
- Unified the reset handling code as much as possible across BL stages. Also introduced some build options to enable optimization of the reset path on platforms that support it.
- Added a simple delay timer API, as well as an SP804 timer driver, which is enabled on FVP.
- Added support for NVidia Tegra T210 and T132 SoCs.
- Reorganised Arm platforms ports to greatly improve code shareability and facilitate the reuse of some of this code by other platforms.
- Added support for Arm Cortex-A72 processor in the CPU specific framework.
- Provided better error handling. Platform ports can now define their own error handling, for example to perform platform specific bookkeeping or post-error actions.
- Implemented a unified driver for Arm Cache Coherent Interconnects used for both CCI-400 & CCI-500 IPs. Arm platforms ports have been migrated to this common driver. The standalone CCI-400 driver has been deprecated.

14.21.2 Issues resolved since last release

- The Trusted Board Boot implementation has been redesigned to provide greater modularity and scalability. See the {ref}`Authentication Framework & Chain of Trust` document. All missing mandatory features are now implemented.
- The FVP and Juno ports may now use the hash of the ROTPK stored in the Trusted Key Storage registers to verify the ROTPK. Alternatively, a development public key hash embedded in the BL1 and BL2 binaries might be used instead. The location of the ROTPK is chosen at build-time using the `ARM_ROTPK_LOCATION` build option.
- GICv3 is now fully supported and stable.

14.21.3 Known issues

- The version of the AEMv8 Base FVP used in this release resets the model instead of terminating its execution in response to a shutdown request using the PSCI `SYSTEM_OFF` API. This issue will be fixed in a future version of the model.
- While this version has low on-chip RAM requirements, there are further RAM usage enhancements that could be made.
- The upstream documentation could be improved for structural consistency, clarity and completeness. In particular, the design documentation is incomplete for PSCI, the TSP(D) and the Juno platform.

- Building TF-A with compiler optimisations disabled (`-O0`) fails.

14.22 1.1.0 (2015-02-04)

14.22.1 New features

- A prototype implementation of Trusted Board Boot has been added. Boot loader images are verified by BL1 and BL2 during the cold boot path. BL1 and BL2 use the PolarSSL SSL library to verify certificates and images. The OpenSSL library is used to create the X.509 certificates. Support has been added to `fip_create` tool to package the certificates in a FIP.
- Support for calling CPU and platform specific reset handlers upon entry into BL3-1 during the cold and warm boot paths has been added. This happens after another Boot ROM `reset_handler()` has already run. This enables a developer to perform additional actions or undo actions already performed during the first call of the reset handlers e.g. apply additional errata workarounds.
- Support has been added to demonstrate routing of IRQs to EL3 instead of S-EL1 when execution is in secure world.
- The PSCI implementation now conforms to version 1.0 of the PSCI specification. All the mandatory APIs and selected optional APIs are supported. In particular, support for the `PSCI_FEATURES` API has been added. A capability variable is constructed during initialization by examining the `plat_pm_ops` and `spd_pm_ops` exported by the platform and the Secure Payload Dispatcher. This is used by the PSCI FEATURES function to determine which PSCI APIs are supported by the platform.
- Improvements have been made to the PSCI code as follows.
  - The code has been refactored to remove redundant parameters from internal functions.
  - Changes have been made to the code for PSCI `CPU_SUSPEND`, `CPU_ON` and `CPU_OFF` calls to facilitate an early return to the caller in case a failure condition is detected. For example, a PSCI `CPU_SUSPEND` call returns `SUCCESS` to the caller if a pending interrupt is detected early in the code path.
  - Optional platform APIs have been added to validate the `power_state` and `entrypoint` parameters early in PSCI `CPU_ON` and `CPU_SUSPEND` code paths.
  - PSCI migrate APIs have been reworked to invoke the SPD hook to determine the type of Trusted OS and the CPU it is resident on (if applicable). Also, during a PSCI `MIGRATE` call, the SPD hook to migrate the Trusted OS is invoked.
- It is now possible to build TF-A without marking at least an extra page of memory as coherent. The build flag `USE_COHERENT_MEM` can be used to choose between the two implementations. This has been made possible through these changes.
  - An implementation of Bakery locks, where the locks are not allocated in coherent memory has been added.
  - Memory which was previously marked as coherent is now kept coherent through the use of software cache maintenance operations.

Approximately, 4K worth of memory is saved for each boot loader stage when `USE_COHERENT_MEM=0`. Enabling this option increases the latencies associated with acquire and release of locks. It also requires changes to the platform ports.

- It is now possible to specify the name of the FIP at build time by defining the `FIP_NAME` variable.
- Issues with dependencies on the 'fiptool' makefile target have been rectified. The `fip_create` tool is now rebuilt whenever its source files change.
- The BL3-1 runtime console is now also used as the crash console. The crash console is changed to SoC UART0 (UART2) from the previous FPGA UART0 (UART0) on Juno. In FVP, it is changed from UART0 to UART1.
- CPU errata workarounds are applied only when the revision and part number match. This behaviour has been made consistent across the debug and release builds. The debug build additionally prints a warning if a mismatch is detected.
- It is now possible to issue cache maintenance operations by set/way for a particular level of data cache. Levels 1-3 are currently supported.
- The following improvements have been made to the FVP port.
  - The build option `FVP_SHARED_DATA_LOCATION` which allowed relocation of shared data into the Trusted DRAM has been deprecated. Shared data is now always located at the base of Trusted SRAM.
  - BL2 Translation tables have been updated to map only the region of DRAM which is accessible to normal world. This is the region of the 2GB DDR-DRAM memory at 0x80000000 excluding the top 16MB. The top 16MB is accessible to only the secure world.
  - BL3-2 can now reside in the top 16MB of DRAM which is accessible only to the secure world. This can be done by setting the build flag `FVP_TSP_RAM_LOCATION` to the value `dram`.
- Separate translation tables are created for each boot loader image. The `IMAGE_BLx` build options are used to do this. This allows each stage to create mappings only for areas in the memory map that it needs.
- A Secure Payload Dispatcher (OPTEED) for the OP-TEE Trusted OS has been added. Details of using it with TF-A can be found in *OP-TEE Dispatcher*

14.22.2 Issues resolved since last release

- The Juno port has been aligned with the FVP port as follows.
  - Support for reclaiming all BL1 RW memory and BL2 memory by overlaying the BL3-1/BL3-2 NOBITS sections on top of them has been added to the Juno port.
  - The top 16MB of the 2GB DDR-DRAM memory at 0x80000000 is configured using the TZC-400 controller to be accessible only to the secure world.
  - The Arm GIC driver is used to configure the GIC-400 instead of using a GIC driver private to the Juno port.
  - PSCI `CPU_SUSPEND` calls that target a standby state are now supported.
  - The TZC-400 driver is used to configure the controller instead of direct accesses to the registers.

- The Linux kernel version referred to in the user guide has DVFS and HMP support enabled.
- DS-5 v5.19 did not detect Version 5.8 of the Cortex-A57-A53 Base FVPs in CADI server mode. This issue is not seen with DS-5 v5.20 and Version 6.2 of the Cortex-A57-A53 Base FVPs.

14.22.3 Known issues

- The Trusted Board Boot implementation is a prototype. There are issues with the modularity and scalability of the design. Support for a Trusted Watchdog, firmware update mechanism, recovery images and Trusted debug is absent. These issues will be addressed in future releases.
- The FVP and Juno ports do not use the hash of the ROTPK stored in the Trusted Key Storage registers to verify the ROTPK in the `plat_match_rotpk()` function. This prevents the correct establishment of the Chain of Trust at the first step in the Trusted Board Boot process.
- The version of the AEMv8 Base FVP used in this release resets the model instead of terminating its execution in response to a shutdown request using the PSCI `SYSTEM_OFF` API. This issue will be fixed in a future version of the model.
- GICv3 support is experimental. There are known issues with GICv3 initialization in the TF-A.
- While this version greatly reduces the on-chip RAM requirements, there are further RAM usage enhancements that could be made.
- The firmware design documentation for the Test Secure-EL1 Payload (TSP) and its dispatcher (TSPD) is incomplete. Similarly for the PSCI section.
- The Juno-specific firmware design documentation is incomplete.

14.23 1.0.0 (2014-08-28)

14.23.1 New features

- It is now possible to map higher physical addresses using non-flat virtual to physical address mappings in the MMU setup.
- Wider use is now made of the per-CPU data cache in BL3-1 to store:
  - Pointers to the non-secure and secure security state contexts.
  - A pointer to the CPU-specific operations.
  - A pointer to PSCI specific information (for example the current power state).
  - A crash reporting buffer.
- The following RAM usage improvements result in a BL3-1 RAM usage reduction from 96KB to 56KB (for FVP with TSPD), and a total RAM usage reduction across all images from 208KB to 88KB, compared to the previous release.
  - Removed the separate `early_exception` vectors from BL3-1 (2KB code size saving).

  - Removed NSRAM from the FVP memory map, allowing the removal of one (4KB) translation table.
  - Eliminated the internal `psci_suspend_context` array, saving 2KB.
  - Correctly dimensioned the PSCI `aff_map_node` array, saving 1.5KB in the FVP port.
  - Removed calling CPU mpidr from the bakery lock API, saving 160 bytes.
  - Removed current CPU mpidr from PSCI common code, saving 160 bytes.
  - Inlined the mmio accessor functions, saving 360 bytes.
  - Fully reclaimed all BL1 RW memory and BL2 memory on the FVP port by overlaying the BL3-1/BL3-2 NOBITS sections on top of these at runtime.
  - Made storing the FP register context optional, saving 0.5KB per context (8KB on the FVP port, with TSPD enabled and running on 8 CPUs).
  - Implemented a leaner `tf_printf()` function, allowing the stack to be greatly reduced.
  - Removed coherent stacks from the codebase. Stacks allocated in normal memory are now used before and after the MMU is enabled. This saves 768 bytes per CPU in BL3-1.
  - Reworked the crash reporting in BL3-1 to use less stack.
  - Optimized the EL3 register state stored in the `cpu_context` structure so that registers that do not change during normal execution are re-initialized each time during cold/warm boot, rather than restored from memory. This saves about 1.2KB.
  - As a result of some of the above, reduced the runtime stack size in all BL images. For BL3-1, this saves 1KB per CPU.
- PSCI SMC handler improvements to correctly handle calls from secure states and from AArch32.
- CPU contexts are now initialized from the `entry_point_info`. BL3-1 fully determines the exception level to use for the non-trusted firmware (BL3-3) based on the SPSR value provided by the BL2 platform code (or otherwise provided to BL3-1). This allows platform code to directly run non-trusted firmware payloads at either EL2 or EL1 without requiring an EL2 stub or OS loader.
- Code refactoring improvements:
  - Refactored `fvp_config` into a common platform header.
  - Refactored the fvp gic code to be a generic driver that no longer has an explicit dependency on platform code.
  - Refactored the CCI-400 driver to not have dependency on platform code.
  - Simplified the IO driver so it's no longer necessary to call `io_init()` and moved all the IO storage framework code to one place.
  - Simplified the interface the the TZC-400 driver.
  - Clarified the platform porting interface to the TSP.
  - Reworked the TSPD setup code to support the alternate BL3-2 initialization flow where BL3-1 generic code hands control to BL3-2, rather than expecting the TSPD to hand control directly to BL3-2.

  - Considerable rework to PSCI generic code to support CPU specific operations.

- Improved console log output, by:
  - Adding the concept of debug log levels.
  - Rationalizing the existing debug messages and adding new ones.
  - Printing out the version of each BL stage at runtime.
  - Adding support for printing console output from assembler code, including when a crash occurs before the C runtime is initialized.
- Moved up to the latest versions of the FVPs, toolchain, EDK2, kernel, Linaro file system and DS-5.
- On the FVP port, made the use of the Trusted DRAM region optional at build time (off by default). Normal platforms will not have such a "ready-to-use" DRAM area so it is not a good example to use it.
- Added support for PSCI `SYSTEM_OFF` and `SYSTEM_RESET` APIs.
- Added support for CPU specific reset sequences, power down sequences and register dumping during crash reporting. The CPU specific reset sequences include support for errata workarounds.
- Merged the Juno port into the master branch. Added support for CPU hotplug and CPU idle. Updated the user guide to describe how to build and run on the Juno platform.

14.23.2 Issues resolved since last release

- Removed the concept of top/bottom image loading. The image loader now automatically detects the position of the image inside the current memory layout and updates the layout to minimize fragmentation. This resolves the image loader limitations of previously releases. There are currently no plans to support dynamic image loading.
- CPU idle now works on the publicized version of the Foundation FVP.
- All known issues relating to the compiler version used have now been resolved. This TF-A version uses Linaro toolchain 14.07 (based on GCC 4.9).

14.23.3 Known issues

- GICv3 support is experimental. The Linux kernel patches to support this are not widely available. There are known issues with GICv3 initialization in the TF-A.
- While this version greatly reduces the on-chip RAM requirements, there are further RAM usage enhancements that could be made.
- The firmware design documentation for the Test Secure-EL1 Payload (TSP) and its dispatcher (TSPD) is incomplete. Similarly for the PSCI section.
- The Juno-specific firmware design documentation is incomplete.
- Some recent enhancements to the FVP port have not yet been translated into the Juno port. These will be tracked via the tf-issues project.

- The Linux kernel version referred to in the user guide has DVFS and HMP support disabled due to some known instabilities at the time of this release. A future kernel version will re-enable these features.
- DS-5 v5.19 does not detect Version 5.8 of the Cortex-A57-A53 Base FVPs in CADI server mode. This is because the `<SimName>` reported by the FVP in this version has changed. For example, for the Cortex-A57x4-A53x4 Base FVP, the `<SimName>` reported by the FVP is `FVP_Base_Cortex_A57x4_A53x4`, while DS-5 expects it to be `FVP_Base_A57x4_A53x4`.

  The temporary fix to this problem is to change the name of the FVP in `sw/debugger/configdb/Boards/ARM FVP/Base_A57x4_A53x4/cadi_config.xml`. Change the following line:

  ```
  <SimName>System Generator:FVP_Base_A57x4_A53x4</SimName>
  ```

  to System Generator:FVP_Base_Cortex-A57x4_A53x4

  A similar change can be made to the other Cortex-A57-A53 Base FVP variants.

14.24 0.4.0 (2014-06-03)

14.24.1 New features

- Makefile improvements:
  - Improved dependency checking when building.
  - Removed `dump` target (build now always produces dump files).
  - Enabled platform ports to optionally make use of parts of the Trusted Firmware (e.g. BL3-1 only), rather than being forced to use all parts. Also made the `fip` target optional.
  - Specified the full path to source files and removed use of the `vpath` keyword.
- Provided translation table library code for potential re-use by platforms other than the FVPs.
- Moved architectural timer setup to platform-specific code.
- Added standby state support to PSCI cpu_suspend implementation.
- SRAM usage improvements:
  - Started using the `-ffunction-sections`, `-fdata-sections` and `--gc-sections` compiler/linker options to remove unused code and data from the images. Previously, all common functions were being built into all binary images, whether or not they were actually used.
  - Placed all assembler functions in their own section to allow more unused functions to be removed from images.
  - Updated BL1 and BL2 to use a single coherent stack each, rather than one per CPU.
  - Changed variables that were unnecessarily declared and initialized as non-const (i.e. in the .data section) so they are either uninitialized (zero init) or const.
- Moved the Test Secure-EL1 Payload (BL3-2) to execute in Trusted SRAM by default. The option for it to run in Trusted DRAM remains.

- Implemented a TrustZone Address Space Controller (TZC-400) driver. A default configuration is provided for the Base FVPs. This means the model parameter `-C bp.secure_memory=1` is now supported.
- Started saving the PSCI cpu_suspend 'power_state' parameter prior to suspending a CPU. This allows platforms that implement multiple power-down states at the same affinity level to identify a specific state.
- Refactored the entire codebase to reduce the amount of nesting in header files and to make the use of system/user includes more consistent. Also split platform.h to separate out the platform porting declarations from the required platform porting definitions and the definitions/declarations specific to the platform port.
- Optimized the data cache clean/invalidate operations.
- Improved the BL3-1 unhandled exception handling and reporting. Unhandled exceptions now result in a dump of registers to the console.
- Major rework to the handover interface between BL stages, in particular the interface to BL3-1. The interface now conforms to a specification and is more future proof.
- Added support for optionally making the BL3-1 entrypoint a reset handler (instead of BL1). This allows platforms with an alternative image loading architecture to re-use BL3-1 with fewer modifications to generic code.
- Reserved some DDR DRAM for secure use on FVP platforms to avoid future compatibility problems with non-secure software.
- Added support for secure interrupts targeting the Secure-EL1 Payload (SP) (using GICv2 routing only). Demonstrated this working by adding an interrupt target and supporting test code to the TSP. Also demonstrated non-secure interrupt handling during TSP processing.

14.24.2 Issues resolved since last release

- Now support use of the model parameter `-C bp.secure_memory=1` in the Base FVPs (see **New features**).
- Support for secure world interrupt handling now available (see **New features**).
- Made enough SRAM savings (see **New features**) to enable the Test Secure-EL1 Payload (BL3-2) to execute in Trusted SRAM by default.
- The tested filesystem used for this release (Linaro AArch64 OpenEmbedded 14.04) now correctly reports progress in the console.
- Improved the Makefile structure to make it easier to separate out parts of the TF-A for re-use in platform ports. Also, improved target dependency checking.

14.24.3 Known issues

- GICv3 support is experimental. The Linux kernel patches to support this are not widely available. There are known issues with GICv3 initialization in the TF-A.
- Dynamic image loading is not available yet. The current image loader implementation (used to load BL2 and all subsequent images) has some limitations. Changing BL2 or BL3-1 load addresses in certain ways can lead to loading errors, even if the images should theoretically fit in memory.
- TF-A still uses too much on-chip Trusted SRAM. A number of RAM usage enhancements have been identified to rectify this situation.
- CPU idle does not work on the advertised version of the Foundation FVP. Some FVP fixes are required that are not available externally at the time of writing. This can be worked around by disabling CPU idle in the Linux kernel.
- Various bugs in TF-A, UEFI and the Linux kernel have been observed when using Linaro toolchain versions later than 13.11. Although most of these have been fixed, some remain at the time of writing. These mainly seem to relate to a subtle change in the way the compiler converts between 64-bit and 32-bit values (e.g. during casting operations), which reveals previously hidden bugs in client code.
- The firmware design documentation for the Test Secure-EL1 Payload (TSP) and its dispatcher (TSPD) is incomplete. Similarly for the PSCI section.

14.25 0.3.0 (2014-02-28)

14.25.1 New features

- Support for Foundation FVP Version 2.0 added. The documented UEFI configuration disables some devices that are unavailable in the Foundation FVP, including MMC and CLCD. The resultant UEFI binary can be used on the AEMv8 and Cortex-A57-A53 Base FVPs, as well as the Foundation FVP.

  :::{note} The software will not work on Version 1.0 of the Foundation FVP. :::
- Enabled third party contributions. Added a new contributing.md containing instructions for how to contribute and updated copyright text in all files to acknowledge contributors.
- The PSCI CPU_SUSPEND API has been stabilised to the extent where it can be used for entry into power down states with the following restrictions:
  - Entry into standby states is not supported.
  - The API is only supported on the AEMv8 and Cortex-A57-A53 Base FVPs.
- The PSCI AFFINITY_INFO api has undergone limited testing on the Base FVPs to allow experimental use.
- Required C library and runtime header files are now included locally in TF-A instead of depending on the toolchain standard include paths. The local implementation has been cleaned up and reduced in scope.

- Added I/O abstraction framework, primarily to allow generic code to load images in a platform-independent way. The existing image loading code has been reworked to use the new framework. Semi-hosting and NOR flash I/O drivers are provided.
- Introduced Firmware Image Package (FIP) handling code and tools. A FIP combines multiple firmware images with a Table of Contents (ToC) into a single binary image. The new FIP driver is another type of I/O driver. The Makefile builds a FIP by default and the FVP platform code expect to load a FIP from NOR flash, although some support for image loading using semi- hosting is retained.

  :::{note} Building a FIP by default is a non-backwards-compatible change. :::

  :::{note} Generic BL2 code now loads a BL3-3 (non-trusted firmware) image into DRAM instead of expecting this to be pre-loaded at known location. This is also a non-backwards-compatible change. :::

  :::{note} Some non-trusted firmware (e.g. UEFI) will need to be rebuilt so that it knows the new location to execute from and no longer needs to copy particular code modules to DRAM itself. :::
- Reworked BL2 to BL3-1 handover interface. A new composite structure (bl31_args) holds the superset of information that needs to be passed from BL2 to BL3-1, including information on how handover execution control to BL3-2 (if present) and BL3-3 (non-trusted firmware).
- Added library support for CPU context management, allowing the saving and restoring of
  - Shared system registers between Secure-EL1 and EL1.
  - VFP registers.
  - Essential EL3 system registers.
- Added a framework for implementing EL3 runtime services. Reworked the PSCI implementation to be one such runtime service.
- Reworked the exception handling logic, making use of both SP_EL0 and SP_EL3 stack pointers for determining the type of exception, managing general purpose and system register context on exception entry/exit, and handling SMCs. SMCs are directed to the correct EL3 runtime service.
- Added support for a Test Secure-EL1 Payload (TSP) and a corresponding Dispatcher (TSPD), which is loaded as an EL3 runtime service. The TSPD implements Secure Monitor functionality such as world switching and EL1 context management, and is responsible for communication with the TSP.

  :::{note} The TSPD does not yet contain support for secure world interrupts. :::

  :::{note} The TSP/TSPD is not built by default. :::

14.25.2 Issues resolved since last release

- Support has been added for switching context between secure and normal worlds in EL3.
- PSCI API calls `AFFINITY_INFO` & `PSCI_VERSION` have now been tested (to a limited extent).
- The TF-A build artifacts are now placed in the `./build` directory and sub-directories instead of being placed in the root of the project.
- TF-A is now free from build warnings. Build warnings are now treated as errors.

- TF-A now provides C library support locally within the project to maintain compatibility between toolchains/systems.
- The PSCI locking code has been reworked so it no longer takes locks in an incorrect sequence.
- The RAM-disk method of loading a Linux file-system has been confirmed to work with the TF-A and Linux kernel version (based on version 3.13) used in this release, for both Foundation and Base FVPs.

14.25.3 Known issues

The following is a list of issues which are expected to be fixed in the future releases of TF-A.

- The TrustZone Address Space Controller (TZC-400) is not being programmed yet. Use of model parameter `-C bp.secure_memory=1` is not supported.
- No support yet for secure world interrupt handling.
- GICv3 support is experimental. The Linux kernel patches to support this are not widely available. There are known issues with GICv3 initialization in TF-A.
- Dynamic image loading is not available yet. The current image loader implementation (used to load BL2 and all subsequent images) has some limitations. Changing BL2 or BL3-1 load addresses in certain ways can lead to loading errors, even if the images should theoretically fit in memory.
- TF-A uses too much on-chip Trusted SRAM. Currently the Test Secure-EL1 Payload (BL3-2) executes in Trusted DRAM since there is not enough SRAM. A number of RAM usage enhancements have been identified to rectify this situation.
- CPU idle does not work on the advertised version of the Foundation FVP. Some FVP fixes are required that are not available externally at the time of writing.
- Various bugs in TF-A, UEFI and the Linux kernel have been observed when using Linaro toolchain versions later than 13.11. Although most of these have been fixed, some remain at the time of writing. These mainly seem to relate to a subtle change in the way the compiler converts between 64-bit and 32-bit values (e.g. during casting operations), which reveals previously hidden bugs in client code.
- The tested filesystem used for this release (Linaro AArch64 OpenEmbedded 14.01) does not report progress correctly in the console. It only seems to produce error output, not standard output. It otherwise appears to function correctly. Other filesystem versions on the same software stack do not exhibit the problem.
- The Makefile structure doesn't make it easy to separate out parts of the TF-A for re-use in platform ports, for example if only BL3-1 is required in a platform port. Also, dependency checking in the Makefile is flawed.
- The firmware design documentation for the Test Secure-EL1 Payload (TSP) and its dispatcher (TSPD) is incomplete. Similarly for the PSCI section.

14.26 0.2.0 (2013-10-25)

14.26.1 New features

- First source release.
- Code for the PSCI suspend feature is supplied, although this is not enabled by default since there are known issues (see below).

14.26.2 Issues resolved since last release

- The "psci" nodes in the FDTs provided in this release now fully comply with the recommendations made in the PSCI specification.

14.26.3 Known issues

The following is a list of issues which are expected to be fixed in the future releases of TF-A.

- The TrustZone Address Space Controller (TZC-400) is not being programmed yet. Use of model parameter `-C bp.secure_memory=1` is not supported.
- No support yet for secure world interrupt handling or for switching context between secure and normal worlds in EL3.
- GICv3 support is experimental. The Linux kernel patches to support this are not widely available. There are known issues with GICv3 initialization in TF-A.
- Dynamic image loading is not available yet. The current image loader implementation (used to load BL2 and all subsequent images) has some limitations. Changing BL2 or BL3-1 load addresses in certain ways can lead to loading errors, even if the images should theoretically fit in memory.
- Although support for PSCI `CPU_SUSPEND` is present, it is not yet stable and ready for use.
- PSCI API calls `AFFINITY_INFO` & `PSCI_VERSION` are implemented but have not been tested.
- The TF-A make files result in all build artifacts being placed in the root of the project. These should be placed in appropriate sub-directories.
- The compilation of TF-A is not free from compilation warnings. Some of these warnings have not been investigated yet so they could mask real bugs.
- TF-A currently uses toolchain/system include files like stdio.h. It should provide versions of these within the project to maintain compatibility between toolchains/systems.
- The PSCI code takes some locks in an incorrect sequence. This may cause problems with suspend and hotplug in certain conditions.
- The Linux kernel used in this release is based on version 3.12-rc4. Using this kernel with the TF-A fails to start the file-system as a RAM-disk. It fails to execute user-space `init` from the RAM-disk. As an alternative, the VirtioBlock mechanism can be used to provide a file-system to the kernel.

CHAPTER

FIFTEEN

GLOSSARY

This glossary provides definitions for terms and abbreviations used in the TF-A documentation.

You can find additional definitions in the Arm Glossary.

AArch32
: 32-bit execution state of the ARMv8 ISA

AArch64
: 64-bit execution state of the ARMv8 ISA

AMU
: Activity Monitor Unit, a hardware monitoring unit introduced by FEAT_AMUv1 that exposes CPU core runtime metrics as a set of counter registers.

API
: Application Programming Interface

AT
: Address Translation

BTI
: Branch Target Identification. An Armv8.5 extension providing additional control flow integrity around indirect branches and their targets.

CoT
COT
: Chain of Trust

CRB
: Command Response Buffer

CSS
: Compute Sub-System

CVE
: Common Vulnerabilities and Exposures. A CVE document is commonly used to describe a publicly-known security vulnerability.

D-CRTM
: Dynamic Code Root of Trust for Measurement

DCE
DRTM Configuration Environment

DICE
Device Identifier Composition Engine

DLME
Dynamically Launched Measured Environment

DPE
DICE Protection Environment

DRTM
Dynamic Root of Trust for Measurement

DS-5
Arm Development Studio 5

DSU
DynamIQ Shared Unit

DT
Device Tree

DTB
Device Tree Blob

EHF
Exception Handling Framework

EL
Exception Level

ERRATA_ABI
Errata management firmware interface

FCONF
Firmware Configuration Framework

FDT
Flattened Device Tree

FF-A
Firmware Framework for Arm A-profile

FIFO
First In, First Out

FIP
Firmware Image Package

FVP
Fixed Virtual Platform

FWU
FirmWare Update

GIC
Generic Interrupt Controller

HES
Arm CCA Hardware Enforced Security

I2C
Inter-Integrated Circuit Protocol

ISA
Instruction Set Architecture

Linaro
A collaborative engineering organization consolidating and optimizing open source software and tools for the Arm architecture.

LSP
A logical secure partition managed by SPM

LTS
Long-Term Support

MMU
Memory Management Unit

MPAM
Memory Partitioning And Monitoring. An optional Armv8.4 extension.

MPIDR
Multiprocessor Affinity Register

MPMM
Maximum Power Mitigation Mechanism, an optional power management mechanism supported by some Arm Armv9-A cores.

MTE
Memory Tagging Extension. An optional Armv8.5 extension that enables hardware-assisted memory tagging.

OEN
Owning Entity Number

OP-TEE
Open Portable Trusted Execution Environment. An example of a *TEE*

OTE
Open-source Trusted Execution Environment

PAUTH
Pointer Authentication. An optional extension introduced in Armv8.3.

PCR
Platform Configuration Register

PDD
Platform Design Document

PMF
Performance Measurement Framework

PSA
Platform Security Architecture

PSCI
Power State Coordination Interface

PSR
Platform Security Requirements

RAS
Reliability, Availability, and Serviceability extensions. A mandatory extension for the Armv8.2 architecture and later. An optional extension to the base Armv8 architecture.

ROT
Root of Trust

RSE
Runtime Security Engine

SCMI
System Control and Management Interface

SCP
System Control Processor

SDEI
Software Delegated Exception Interface

SDS
Shared Data Storage

SEA
Synchronous External Abort

SiP
SIP
Silicon Provider

SMC
Secure Monitor Call

SMCCC
SMC Calling Convention

SoC
System on Chip

SP
Secure Partition

SPD
Secure Payload Dispatcher

SPI
Serial Peripheral Interface

SPM
Secure Partition Manager

SRTM
Static Root of Trust for Measurement

SSBS
Speculative Store Bypass Safe. Introduced in Armv8.5, this configuration bit can be set by software to allow or prevent the hardware from performing speculative operations.

SVE
Scalable Vector Extension

TBB
Trusted Board Boot

TBBR
Trusted Board Boot Requirements

TCB
Trusted Compute Base

TCG
Trusted Computing Group

TEE
Trusted Execution Environment

TF-A
Trusted Firmware-A

TF-M
Trusted Firmware-M

TLB
Translation Lookaside Buffer

TLK
Trusted Little Kernel. A Trusted OS from NVIDIA.

TPM
Trusted Platform Module

TRNG
True Random Number Generator (hardware based)

TSP
Test Secure Payload

TZC
TrustZone Controller

UBSAN
Undefined Behavior Sanitizer

UEFI
Unified Extensible Firmware Interface

WDOG
Watchdog

XLAT
Translation (abbr.). For example, "XLAT table".

CHAPTER SIXTEEN

LICENSE

The software is provided under a BSD-3-Clause license (below). Contributions to this project are accepted under the same license with developer sign-off as described in the *Contributor's Guide*.

```
Copyright (c) [XXXX-]YYYY, <OWNER>. All rights reserved.

Redistribution and use in source and binary forms, with or without␣
↪modification,
are permitted provided that the following conditions are met:

-  Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

-  Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

-  Neither the name of Arm nor the names of its contributors may be used to
endorse or promote products derived from this software without specific
prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"␣
↪AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE␣
↪FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

16.1 SPDX Identifiers

Individual files contain the following tag instead of the full license text.

```
SPDX-License-Identifier:    BSD-3-Clause
```

This enables machine processing of license information based on the SPDX License Identifiers that are here available: http://spdx.org/licenses/

16.2 Other Projects

This project contains code from other projects as listed below. The original license text is included in those source files.

- The libc source code is derived from FreeBSD and SCC. FreeBSD uses various BSD licenses, including BSD-3-Clause and BSD-2-Clause. The SCC code is used under the BSD-3-Clause license with the author's permission.
- The libfdt source code is disjunctively dual licensed (GPL-2.0+ OR BSD-2-Clause). It is used by this project under the terms of the BSD-2-Clause license. Any contributions to this code must be made under the terms of both licenses.
- The LLVM compiler-rt source code is disjunctively dual licensed (NCSA OR MIT). It is used by this project under the terms of the NCSA license (also known as the University of Illinois/NCSA Open Source License), which is a permissive license compatible with BSD-3-Clause. Any contributions to this code must be made under the terms of both licenses.
- The zlib source code is licensed under the Zlib license, which is a permissive license compatible with BSD-3-Clause.
- Some STMicroelectronics platform source code is disjunctively dual licensed (GPL-2.0+ OR BSD-3-Clause). It is used by this project under the terms of the BSD-3-Clause license. Any contributions to this code must be made under the terms of both licenses.
- Some source files originating from the Linux source tree, which are disjunctively dual licensed (GPL-2.0 OR MIT), are redistributed under the terms of the MIT license. These files are:
  - `include/dt-bindings/interrupt-controller/arm-gic.h`
  - `include/dt-bindings/interrupt-controller/irq.h`

  See the original Linux MIT license.
- Some source files originating from the Open Profile for DICE project. These files are licensed under the Apache License, Version 2.0, which is a permissive license compatible with BSD-3-Clause. Any contributions to this code must also be made under the terms of Apache License 2.0. These files are:
  - `include/lib/dice/dice.h`
- Some source files originating from the pydevicetree project. These files are licensed under the Apache License, Version 2.0, which is a permissive license compatible with BSD-3-Clause. Any contributions to this code must also be made under the terms of Apache License 2.0. These files are:

- tools/cot_dt2c/cot_dt2c/pydevicetree/ast/__init__.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/ast/directive.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/ast/helpers.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/ast/node.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/ast/property.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/ast/reference.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/source/__init__.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/source/grammar.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/source/parser.py
- tools/cot_dt2c/cot_dt2c/pydevicetree/__init__.py

- Some source files originating from the edk2 project. These files are licensed under the BSD-2-Clause. Any contributions to this code must also be made under the terms of BSD-2-Clause. These files are:
  - lib/hob/hob.c
  - include/lib/hob/mmram.h
  - include/lib/hob/mpinfo.h
- Some source files originating from the mbed OS project. These files are licensed under the Apache License, Version 2.0, which is a permissive license compatible with BSD-3-Clause. Any contributions to this code must also be made under the terms of Apache License 2.0. These files are:
  - tools/memory/memory/mapsummary.py
  - tools/memory/memory/mapsummary_flamegraph.hmtl

Trusted Firmware-A (TF-A) provides a reference implementation of secure world software for Armv7-A and Armv8-A, including a Secure Monitor executing at Exception Level 3 (EL3). It implements various Arm interface standards, such as:

- The Power State Coordination Interface (PSCI)
- Trusted Board Boot Requirements CLIENT (TBBR-CLIENT)
- SMC Calling Convention
- System Control and Management Interface (SCMI)
- Software Delegated Exception Interface (SDEI)
- PSA FW update specification

Where possible, the code is designed for reuse or porting to other Armv7-A and Armv8-A model and hardware platforms.

This release provides a suitable starting point for productization of secure world boot and runtime firmware, in either the AArch32 or AArch64 execution states.

Users are encouraged to do their own security validation, including penetration testing, on any secure world code derived from TF-A.

In collaboration with interested parties, we will continue to enhance *TF-A* with reference implementations of Arm standards to benefit developers working with Armv7-A and Armv8-A TrustZone technology.

CHAPTER

SEVENTEEN

GETTING STARTED

The *TF-A* documentation contains guidance for obtaining and building the software for existing, supported platforms, as well as supporting information for porting the software to a new platform.

The **About** chapter gives a high-level overview of *TF-A* features as well as some information on the project and how it is organized.

Refer to the documents in the **Getting Started** chapter for information about the prerequisites and requirements for building *TF-A*.

The **Processes & Policies** chapter explains the project's release schedule and process, how security disclosures are handled, and the guidelines for contributing to the project (including the coding style).

The **Components** chapter holds documents that explain specific components that make up the *TF-A* software, the *Exception Handling Framework*, for example.

In the **System Design** chapter you will find documents that explain the design of portions of the software that involve more than one component, such as the *Trusted Board Boot* process.

Platform Ports provides a list of the supported hardware and software-model platforms that are supported upstream in *TF-A*. Most of these platforms also have additional documentation that has been provided by the maintainers of the platform.

The results of any performance evaluations are added to the **Performance & Testing** chapter.

Security Advisories holds a list of documents relating to *CVE* entries that have previously been raised against the software.
