Trusted Firmware-A Getting Started Guide

Version 1.0

Non-Confidential

Issue 01

110431_0100_01_en

Trusted Firmware-A Getting Started Guide

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	8 April 2025	Non-Confidential	First release

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Trusted Firmware-A

This section outlines an introduction to Trusted Firmware-A.

1.1 Introduction to Trusted Firmware-A

From TrustedFirmware.org:

“Arm Trusted Firmware provides a reference implementation of secure world software for Armv8-A and Armv8-M. It provides SoC developers and OEMs with a reference trusted code base complying with the relevant Arm specifications. The code on this website is the preferred implementation of Arm specifications, allowing quick and easy porting to modern chips and platforms. This forms the foundations of a Trusted Execution Environment (TEE) on application processors, or the Secure Processing Environment (SPE) of microcontrollers.”

Trusted Firmware-A targets Armv8-A SoCs; see Trusted Firmware-M for Armv8-M SoCs.

The Trusted Firmware project is adopting open governance:

- Arm is transferring the Trusted Firmware project to be managed as an open project by Linaro
- Membership of the Trusted Firmware project is open to all
- Governance will be overseen by a board of member representatives
- Stakeholders in Trusted Firmware are encouraged to join

1.2 Getting Started with Trusted Firmware-A

You can install and run Trusted Firmware-A on some platforms as part of the reference software stacks shipped with the Arm Reference Platforms deliverables.

Alternatively, follow the official user guide.

1.3 Official sites

View the following list of official Trusted Firmware-A websites.

- TrustedFirmware.org
- Source code repo
- Documentation
- Wiki

1.4 FAQs and tutorials

View the following list of Trusted Firmware-A FAQs and tutorials.

- Official user guide
- Debugging Arm Trusted Firmware
- Using old Arm Trusted Firmware releases

2. Debugging Arm Trusted Firmware

This section outlines how to debug Arm Trusted Firmware.

2.1 Preface

This article outlines how to use DS-5 Development Studio (DS-5) to debug Arm Trusted Firmware (ATF) from cold reset through to normal world handover.

Specifically, this article discusses two of the most common obstacles encountered when trying to debug ATF:

- ATF comprises multiple individual boot stages that run at different exception levels; symbols and debug information must therefore be loaded from the correct file and into the correct virtual address space
- One must take control of the system very early on; this can be particularly difficult on hardware platforms and requires modifications to the ATF source code

These instructions are primarily written targeting the Armv8 Foundation Model FVP, with delta instructions for the Juno development platform also provided at the end.

2.2 Getting Started

Get started with the following.

2.2.1 Install Required Software

Install the following software:

- Install DS-5 Development Studio here (article written using version 5.27.1)
- Install the Armv8 Foundation Model here (article written using version 11.1 build 24)

2.2.2 Obtain ATF Sources

Follow the instructions for using the Linaro software deliverables on an FVP to download the workspace initialisation script (article written using version 1707) and then select the following configuration when prompted:

```
+-------------+-----------------------------------+
| Workspace   | <workspace>                       |
| Platform    | [64-bit] AEMv8-A Base Platform FVP |
| Build       | Build from source                 |
| Environment | Linux/Android                     |
| Kernel      | latest-armlt                      |
| Filesystem   | BusyBox Built from source            |
+--------------+--------------------------------------+
```

The ATF sources can then be found in the `<workspace>/arm-tf/` directory.

While it is possible to manually fetch the sources from GitHub, we recommend the above automated method as the instructions below depend on other files provided as part of those deliverables, such as a Linux kernel image, ramdisk image, and normal world bootloader BL33 image.

2.3 Arm Trusted Firmware overview

The ATF cold boot flow comprises up to five individual boot stages running at different exception levels:

Boot stage	Exception level	Description
BL1	EL3	Trusted bootstrap; cold/warm boot detection
BL2	EL1S	Trusted bootloader
BL31	EL3	Resident runtime firmware
BL32	EL1S	[Optional] Trusted operating system
BL33	EL2	Normal world bootloader

With these stages run in the following order:

```
BL1  -->  BL2  -->  BL1  -->  BL31  -->  BL32  -->  BL31  -->  BL33
```

We recommend reading the Arm Trusted Firmware Design document for more information (can also be found in `<workspace>/arm-tf/docs/`).

This article outlines how to debug ATF:

- From the BL1 entrypoint through to the BL33 entrypoint i.e. “normal world handover”
- In a system without a trusted operating system i.e. no BL32 present
- Using the official reference implementation sources of BL1, BL2, and BL31 (*)

(*) The `bl1/`, `bl2/`, and `bl31/` directories in `<workspace>/arm-tf/`.

When debugging ATF it is important to know which boot stage(s) contain the functionality that you are interested in; this way the correct symbols and debug information can be loaded, allowing us to set breakpoints on textual symbol names rather than raw addresses, see function call target names rather than PC relative offsets, and so on. It also means we can skip unnecessary parts of the boot flow.

To this end we have generated the following table of “interesting” functionality with corresponding boot stage and symbol name(s):

Functionality	Boot stage	Symbols
Cold/warm boot detection	BL1	`plat_get_my_entrypoint`
CPU-specific reset handlers	BL1	`reset_handler`
Bootstrap (BL1) entrypoint and early setup	BL1	`bl1_entrypoint`
Bootstrap (BL1) main	BL1	`bl1_main`
Load Bootloader (BL2) from FIP	BL1	`bl1_load_bl2`
Bootstrap (BL1) -> Bootloader (BL2) handover	BL1	`bl1_prepare_next_image`
Bootstrap (BL1) -> Bootloader (BL2) handover	BL1	`el3_exit`
Bootloader (BL2) entrypoint and early setup	BL2	`bl2_entrypoint`
Bootloader (BL2) main	BL2	`bl2_main`
Load images from FIP	BL2	`bl2_load_images`
Bootloader (BL2) -> Bootstrap (BL1) handover	BL2	`smc`
Bootstrap (BL1) -> Firmware (BL31) handover	BL1	`bl1_plat_prepare_exit`
Firmware (BL31) cold boot entrypoint and early setup	BL31	`bl31_entrypoint`
Firmware (BL31) warm boot entrypoint	BL31	`bl31_warm_entrypoint`
Firmware (BL31) main	BL31	`bl31_main`
Initialise CPU operations	BL31	`init_cpu_ops`
Power management setup	BL31	`populate_power_domain_tree`
Power management setup	BL31	`psci_init_pwr_domain_node`
Power management setup	BL31	`psci_set_pwr_domains_to_run`
CPU power down sequence	BL31	`prepare_cpu_pwr_dwn`
Firmware BL31 -> BL33 normal world handover	BL31	`bl31_prepare_next_image`
Firmware BL31 -> BL33 normal world handover	BL31	`el3_exit`

Make a note of any of these that interest you.

2.4 Building and running ATF

Run ATF on the Armv8 Foundation Model model like so (we recommend turning this into a shell script for ease of use):

```
/path/to/Foundation_Platform \
    --cores=4 --secure-memory --visualization --gicv3 \
    --data=<workspace>/output/fvp/fvp-busybox/uboot/bl1.bin@0x0 \
    --data=<workspace>/output/fvp/fvp-busybox/uboot/fip.bin@0x08000000 \
    --data=<workspace>/output/fvp/fvp-busybox/uboot/foundation-v8-
gicv3.dtb@0x82000000 \
    --data=<workspace>/output/fvp/fvp-busybox/uboot/Image@0x80080000 \
    --data=<workspace>/output/fvp/fvp-busybox/uboot/ramdisk.img@0x84000000 \
    --cadi-server
```

Replacing `/path/to/Foundation_Model` with the path to your Armv8 Foundation Model executable and `<workspace>` with the path to your workspace directory.

Note that the command references artefacts in `<workspace>` that are only present after invoking the build script referenced in the instructions linked above.

2.5 Preparing to debug ATF

Connecting to the model

Running the model with the `--cadi-server` flag causes the simulation to pause at the first cycle waiting for a debugger to be connected.

From the DS-5 Debug perspective, navigate to:

```
File  -->  New  -->  Other  -->  DS-5 Configuration Database  -->  Configuration
 Database
```

Enter a name of your choice, such as “ATF on Armv8 Foundation Model”, then click “Finish”.

Next, navigate to:

```
File  -->  New  -->  Other  --> DS-5 Configuration Database --> Model Configuration
```

And:

1. Select the configuration database created above
2. Click “Next”
3. Select “Browse for model running on local host”
4. Click “Next”
5. Click “Browse”
6. Select the running model from the list, for example “System Generator:Foundation_AEMv8A (port=7000)”
7. Click “Finish”
8. Click “Import”
9. Click “Debug”

On the window that opens, navigate to the “Debugger” tab, tick “Connect only”, tick “Execute debugger commands”, and copy-paste the following into the text box to automatically load all symbols into the correct virtual address space each time you connect to the model:

```
add-symbol-file /arm-tf/build/fvp/debug/bl1/bl1.elf EL3:0
add-symbol-file <workspace>/arm-tf/build/fvp/debug/bl2/bl2.elf EL1S:0
add-symbol-file <workspace>/arm-tf/build/fvp/debug/bl31/bl31.elf EL3:0
add-symbol-file <workspace>/u-boot/output/vexpress_aemv8a_semi/u-boot EL2:0
add-symbol-file <workspace>/linux/out/fvp/mobile_bb/vmlinux EL2:0
```

Replacing `<workspace>` with the path to your workspace directory.

The EL and number at the end of each command (e.g. `EL3:0’) ensure the symbols are loaded into the correct virtual address space and at the correct memory offset; ATF uses absolute addresses for its symbols so we ensure an offset of 0.

Click “Apply” and then “Debug” to connect to the paused model. You can now step through the ATF code or set a breakpoint on the symbol corresponding to the functionality that you are interested in.

2.6 Instruction delta for Juno

This section highlights the differences between the above instructions, which target the Armv8 Foundation Model, and the steps required to debug ATF on the Juno hwardware development platform.

2.6.1 Obtaining the sources

Run the workspace initialisation script to sync a new workspace as outlined earlier, but this time targeting the `[64-bit] Juno` platform.

2.6.2 Modifying the sources

Unlike the Armv8 Foundation Model, which will be paused on the first cycle of the simulation waiting for a debugger to be connected, the Juno hardware development platform will immediately begin booting ATF when power cycled. Due to the application processor debug access ports (DAPs) not being powered up until that same moment, we cannot connect a debugger until the board has already progressed some of the way through the ATF boot flow.

Due to a known issue, the way you get around this will depend on which boot stage(s) you want to debug.

2.6.3 To debug up to BL1 –> BL31 handover

Navigate to `<workspace>/arm-tf/bl1/aarch64/bl1_entrypoint.S`, find the `bl1_entrypoint` function, and add a `b .` instruction:

```
func bl1_entrypoint

    b   .   // <-- Branch-to-self added here

    /* ---------------------------------------------------------------------
    * If the reset address is programmable then bl1_entrypoint() is
    * executed only on the cold boot path. Therefore, we can skip the warm
    * boot mailbox mechanism.
    * ---------------------------------------------------------------------
    */
    el3_entrypoint_common                                 \
        _init_sctlr=1                                     \
        _warm_boot_mailbox=!PROGRAMMABLE_RESET_ADDRESS    \
        _secondary_cold_boot=!COLD_BOOT_SINGLE_CPU        \
        _init_memory=1                                    \
        _init_c_runtime=1                                 \
        _exception_vectors=bl1_exceptions
```

2.6.4 To debug from BL31 entrypoint onwards

NOTE: Ensure you do not have a `b .` instruction in the code path leading up to the BL31 entrypoint; due to the known issue at time of writing this will cause the board to panic.

Navigate to `<workspace>/arm-tf/make_helpers/defaults.mk` and modify this line to set the switch to `1`:

```
# Flag to introduce an infinite loop in BL1 just before it exits into the next
# image. This is meant to help debugging the post-BL2 phase.
SPIN_ON_BL1_EXIT
```

Additionally, add the following new lines:

```
# Flag to disable the Trusted Watchdog
ARM_DISABLE_TRUSTED_WDOG     := 1
```

Then perform a `make realclean` from the `<workspace>/arm-tf/` directory before rebuilding the software in the usual way (using the `<workspace>/build-scripts/build-all all` script).

2.6.5 Debugging

When you connect the debugger, the primary CPU will be “spinning” on a `b .` instruction; either the one you manually added to the `bl1_entrypoint` function or one that was compiled into the BL1 -> BL31 handover as a result of setting the `SPIN_ON_BL1_EXIT` flag.

Simply interrupt the CPU and enter debug command `set $pc += 4`; you can now step through and debug the ATF boot flow just like on the Armv8 Foundation Model.

3. Using old Arm Trusted Firmware releases

To obtain the latest Arm Trusted Firmware sources you can clone the git repository:

```
$ git clone https://github.com/ARM-software/arm-trusted-firmware.git
```

The version of Arm Trusted Firmware included in the [Arm Reference Platforms deliverables] under the `/arm-tf/` directory is the latest that has been validated on the supported Arm-supplied development platforms, but may not necessarily be the same as the most up-to-date development version.

To forcefully synchronise your workspace's Arm Trusted Firmware sources to the latest version:

```
$ cd <workspace>/arm-tf/
$ git fetch github
$ git checkout --track github/master
```

Note that while we expect the latest Linaro software stack deliverables and latest Arm Trusted Firmware development version will always be compatible, this may not actually be the case. Please keep this in mind when attempting to use a more recent version of the Arm Trusted Firmware sources.

4. Trusted Firmware-M

This section introduces Trusted Firmware-M.

4.1 Introduction to Trusted Firmware-M

From TrustedFirmware.org:

“Arm Trusted Firmware provides a reference implementation of secure world software for Armv8-A and Armv8-M. It provides SoC developers and OEMs with a reference trusted code base complying with the relevant Arm specifications. The code on this website is the preferred implementation of Arm specifications, allowing quick and easy porting to modern chips and platforms. This forms the foundations of a Trusted Execution Environment (TEE) on application processors, or the Secure Processing Environment (SPE) of microcontrollers.”

Trusted Firmware-M targets Armv8-M SoCs; see Trusted Firmware-A for Armv8-A SoCs.

The Trusted Firmware project is adopting open governance:

- Arm is transferring the Trusted Firmware project to be managed as an open project by Linaro
- Membership of the Trusted Firmware project is open to all
- Governance will be overseen by a board of member representatives
- Stakeholders in Trusted Firmware are encouraged to join

4.2 Useful links

Official sites:

- TrustedFirmware.org
- Source code
- Documentation
- Wiki

4.3 FAQs and tutorials

View the following resources:

- Offical user guide
- Running TF-M on Musca