Total Compute

Arm Limited

Apr 04, 2024

TOTAL COMPUTE:

CHAPTER

ONE

TOTAL COMPUTE: TC2-2024.02.22-LSC

Total Compute is an approach to moving beyond optimizing individual IP to take a system-level solution view of the SoC that puts use cases and experiences at the heart of the designs.

Total Compute focuses on optimizing Performance, Security, and Developer Access across Arm's IP, software, and tools. This means higher-performing, more immersive, and more secure experiences on devices coupled with an easier app and software development process.

1.1 Total Compute Platform Software Components

1.1.1 RSS Firmware

Runtime Security Subsystem (RSS) serves as the Root of Trust for the Total Compute platform.

RSS BL1 code is the first software that executes right after a cold reset or Power-on.

RSS initially boots from immutable code (BL1_1) in its internal ROM, before jumping to BL1_2, which is provisioned and hash-locked in RSS OTP. The updatable MCUboot BL2 boot stage is loaded from the flash into RSS SRAM, where it is authenticated. BL2 loads and authenticates the TF-M runtime into RSS SRAM from host flash. BL2 is also responsible for loading initial boot code into other subsystems within Total Compute as below.

1. SCP BL1;
2. AP BL1.

The following diagram illustrates the boot flow sequence:

1.1.2 SCP Firmware

The System Control Processor (SCP) is a compute unit of Total Compute and is responsible for low-level system management. The SCP is a Cortex-M3 processor with a set of dedicated peripherals and interfaces that you can extend. SCP firmware supports:

1. Power-up sequence and system start-up;
2. Initial hardware configuration;
3. Clock management;
4. Servicing power state requests from the OS Power Management (OSPM) software.

SCP BL1

It performs the following functions:

1. Sets up generic timer, UART console and clocks;
2. Initializes the Coherent Interconnect;
3. Powers ON primary AP CPU;
4. Loads SCP Runtime Firmware.

SCP Runtime Firmware

SCP runtime code starts execution after TF-A BL2 has authenticated and copied it from flash. It performs the following functions:

1. Responds to SCMI messages via MHUv2 for CPU power control and DVFS;
2. Power Domain management;
3. Clock management.

1.1.3 AP Secure World Software

Secure software/firmware is a trusted software component that runs in the AP secure world. It mainly consists of AP firmware, Secure Partition Manager and Secure Partitions (OP-TEE, Trusted Services).

AP firmware

The AP firmware consists of the code that is required to boot Total Compute platform up to the point where the OS execution starts. This firmware performs architecture and platform initialization. It also loads and initializes secure world images like Secure partition manager and Trusted OS.

Trusted Firmware-A (TF-A) BL1

BL1 performs minimal architectural initialization (like exception vectors, CPU initialization) and Platform initialization. It loads the BL2 image and passes control to it.

Trusted Firmware-A (TF-A) BL2

BL2 runs at S-EL1 and performs architectural initialization required for subsequent stages of TF-A and normal world software. It configures the TrustZone Controller and carves out memory region in DRAM for secure and non-secure use. BL2 loads below images:

1. SCP BL2 image;
2. EL3 Runtime Software (BL31 image);
3. Secure Partition Manager (BL32 image);
4. Non-Trusted firmware - EDK2 (BL33 image) and Grub;
5. Secure Partitions images (OP-TEE and Trusted Services).

Trusted Firmware-A (TF-A) BL31

BL2 loads EL3 Runtime Software (BL31) and BL1 passes control to BL31 at EL3. In Total Compute BL31 runs at trusted SRAM. It provides the below mentioned runtime services:

1. Power State Coordination Interface (PSCI);
2. Secure Monitor framework;
3. Secure Partition Manager Dispatcher.

Secure Partition Manager

Total Compute enables FEAT S-EL2 architectural extension, and it uses Hafnium as Secure Partition Manager Core (SPMC). BL32 option in TF-A is re-purposed to specify the SPMC image. The SPMC component runs at S-EL2 exception level.

Secure Partitions

Software image isolated using SPM is Secure Partition. Total Compute enables OP-TEE and Trusted Services as Secure Partitions.

OP-TEE

OP-TEE Trusted OS is virtualized using Hafnium at S-EL2. OP-TEE OS for Total Compute is built with FF-A and SEL2 SPMC support. This enables OP-TEE as a Secure Partition running in an isolated address space managed by Hafnium. The OP-TEE kernel runs at S-EL1 with Trusted applications running at S-EL0.

Trusted Services

Trusted Services like Crypto Service, Internal Trusted Storage runs as S-EL0 Secure Partitions.

1.1.4 AP Non-Secure World Software

EDK2 (BL33)

TF-A BL31 passes execution control to EDK2 UEFI FW bootloader (BL33). EDK2 UEFI FW will load and verify signature of Grub.

Grub bootloader

Grub bootloader provides flexibility to configure some boot parameters, and ultimately loads and boots the Linux Kernel.

Linux Kernel

Linux Kernel in Total Compute contains the subsystem-specific features that demonstrate the capabilities of Total Compute.

Debian

This variant is based on the Debian 12 (aka Bookworm) filesystem. This image can be used for development or validation work that does not imply pixel rendering, as currently there is no support for software or hardware rendering.

1.2 Instructions: Obtaining Total Compute software deliverables

- To build the TC2 software stack, please refer to the *user guide*;
- For further details on the latest release and features, please refer to the *release notes*;
- To get detailed information on the system architecture and each of its components, please refer to the .

1.3 TC Software Stack Overview

The TC2 software consists of firmware, kernel and file system components that can run on the associated FVP.

Following is presented the high-level list of the software components:

1. SCP firmware – responsible for system initialization, clock and power control;
2. RSS firmware – provides Hardware Root of Trust;
3. AP firmware – Trusted Firmware-A (TF-A);
4. Secure Partition Manager - Hafnium;
5. Secure Partitions:
   - OP-TEE Trusted OS;
   - Trusted Services;
6. EDK2 UEFI FW and Grub – loads and verifies the fitImage for Debian boot, containing kernel and filesystem;
7. Kernel – supports the following hardware features:
   - Message Handling Unit;
   - PAC/MTE/BTI features;
8. Debian;

For more information on each of the stack components, please refer to the *Total Compute Platform Software Components* section.

1.4 User Guide

Contents

1.4.1 Notice

The Total Compute 2022 (TC2) software stack uses bash scripts to build an integrated solution comprising Board Support Package (BSP) and Debian distribution.

1.4.2 Prerequisites

These instructions assume that:

- Your host PC is running Ubuntu Linux 20.04;
- You are running the provided scripts in a `bash` shell environment;
- This release requires TC2 Fast Model platform (FVP) version 11.23.28.

To get the latest repo tool from Google, please run the following commands:

```
mkdir -p ~/bin
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
export PATH=~/bin:$PATH
```

To avoid errors while attempting to clone/fetch the different TC software components, your system should have a proper minimum `git config` configuration. The following command exemplifies the typical `git config` configuration required:

```
git config --global user.name "<user name>"
git config --global user.email "<email>"
git config --global protocol.version 2
```

To install and allow access to docker, please run the following commands:

```
sudo apt install docker.io
# ensure docker service is properly started and running
sudo systemctl restart docker
```

To manage Docker as a non-root user, please run the following commands:

```
sudo usermod -aG docker $USER
newgrp docker
```

1.4.3 Download the source code and build

The TC2 software stack supports the following distro:

- Debian (based on Debian 12 Bookworm);

Download the source code

Create a new folder that will be your workspace, which will henceforth be referred to as <TC2_WORKSPACE> in these instructions.

```
mkdir <TC2_WORKSPACE>
cd <TC2_WORKSPACE>
export TC2_RELEASE=refs/tags/TC2-2024.02.22-LSC
```

To sync **Debian source code**, please run the following repo commands:

```
repo init -u https://gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest \
            -m tc2.xml \
            -b ${TC2_RELEASE} \
            -g bsp
repo sync -j `nproc` --fetch-submodules
```

Once the previous process finishes, the current <TC2_WORKSPACE> should have the following structure:

- `build-scripts/`: the components build scripts;
- `run-scripts/`: scripts to run the FVP;
- `src/`: each component's git repository.

Initial Setup

The setup includes two parts:

1. setup a docker image;
2. setup the environment to build TC images.

Setting up a docker image involves pulling the prebuilt docker image from a docker registry. If that fails, it will build a local docker image.

To setup a docker image, patch the components, install the toolchains and build tools, please run the commands mentioned in the following *Build variants configuration* section, according to the distro and variant of interest.

The various tools will be installed in the `<TC2_WORKSPACE>/tools/` directory.

Build options

Debian OS build variant

Currently, the Debian OS build distro does not support software or hardware rendering. Considering this limitation, this build variant should be only used for development or validation work that does not imply pixel rendering.

Build variants configuration

This section provides a quick guide on how to build the TC software stack considering the Debian build variant, using the most common options.

Debian build

Currently, the Debian build does not support software or hardware rendering. As such, the `TC_GPU` variable value should not be defined. The Debian build can still be a valuable resource when just considering other types of development or validation work, which do not involve pixel rendering.

Debian build (UEFI boot with ACPI Support)

To build the Debian with UEFI based boot which has ACPI support, please run the following commands:

```
export PLATFORM=tc2
export FILESYSTEM=debian
export TC_TARGET_FLAVOR=fvp
export TC_BL33=uefi
cd build-scripts
./setup.sh
```

Warning: If building the TC2 software stack for more than one target, please ensure you run a clean build between each different build to avoid setup/building errors (refer to the next section "*More about the build system*" for command usage examples on how to do this).

Warning: If running `repo sync` again is needed at some point, then the `setup.sh` script also needs to be run again, as `repo sync` can discard the patches.

Note: Most builds will be done in parallel using all the available cores by default. To change this number, run `export PARALLELISM=<number of cores>`

Build command

To build the whole TC2 software stack, simply run:

```
./run_docker.sh ./build-all.sh build
```

Once the previous process finishes, the previously defined environment variable `$FILESYSTEM` will be automatically used and the current `<TC2_WORKSPACE>` should have the following structure:

- build files are stored in `<TC2_WORKSPACE>/output/<$FILESYSTEM>/tmp_build/`;
- final images will be placed in `<TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/`.

More about the build system

The `build-all.sh` script will build all the components, but each component has its own script, allowing it to be built, cleaned and deployed separately. All scripts support the `build`, `clean`, `deploy`, `patch` commands. `build-all.sh` also supports `all`, which performs a clean followed by a rebuild of all the stack.

For example, to build, deploy, and clean SCP, run:

```
./run_docker.sh ./build-scp.sh build
./run_docker.sh ./build-scp.sh deploy
./run_docker.sh ./build-scp.sh clean
```

The platform and filesystem used should be defined as described previously, but they can also be specified as the following example:

```
./run_docker.sh ./build-all.sh \
            -p $PLATFORM \
            -f $FILESYSTEM \
            -t $TC_TARGET_FLAVOR \
            -g $TC_GPU \
            -b ${TC_BL33} build
```

Build Components and its dependencies

A new dependency to a component can be added in the form of `$component=$dependency` in the `dependencies.txt` file

To build a component and rebuild those components that depend on it, run:

```
./run_docker.sh ./<BUILD-SCRIPT-FILENAME> build with_reqs
```

Those options work for all the `build-*.sh` scripts.

1.4.4 Provided components

Firmware Components

Trusted Firmware-A

Based on Trusted Firmware-A

Script	<TC2_WORKSPACE>/build-scripts/build-tfa.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/bl1-tc.bin • <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/fip-tc.bin

System Control Processor (SCP)

Based on SCP Firmware

Script	<TC2_WORKSPACE>/build-scripts/build-scp.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/scp_ra • <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/scp_ro

Hafnium

Based on Hafnium

Script	<TC2_WORKSPACE>/build-scripts/build-hafnium.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/hafniu

OP-TEE

Based on OP-TEE

Script	<TC2_WORKSPACE>/build-scripts/build-optee-os.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/tmp_build/tfa_sp/t pager_v2.bin

S-EL0 trusted-services

Based on Trusted Services

Script	<TC2_WORKSPACE>/build-scripts/build-trusted-services.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/tmp_build/tfa_sp/c sp.bin • <TC2_WORKSPACE>/output/<$FILESYSTEM>/tmp_build/tfa_sp/i trusted-storage.bin

Linux

The component responsible for building a 6.1 version of the mainline kernel (Linux).

Script	<TC2_WORKSPACE>/build-scripts/build-linux.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/Image

Distributions

Debian Linux distro

Script	<TC2_WORKSPACE>/build-scripts/build-debian.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/debian-12-nocloud-arm64-20230612-1409.raw.img

UEFI

Script	<TC2_WORKSPACE>/build-scripts/build-uefi.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/uefi.bin

GRUB

Script	<TC2_WORKSPACE>/build-scripts/build-grub.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/grubaa

Run scripts

Within the `<TC2_WORKSPACE>/run-scripts/` there are several convenience functions for testing the software stack. Usage descriptions for the various scripts are provided in the following sections.

1.4.5 Obtaining the TC2 FVP

The TC2 FVP is available to partners to build and run on Linux host environments.

To download the latest publicly available TC2 FVP model, please visit the webpage or contact Arm (support@arm.com).

1.4.6 Running the software on FVP

A Fixed Virtual Platform (FVP) of the TC2 platform must be available to run the included run scripts.

The run-scripts structure is as follows (assuming <TC2_WORKSPACE> location):

```
run-scripts
|--tc2
   |--run_model.sh
   |-- ...
```

Ensure that all dependencies are met by running the FVP: `./path/to/FVP_TC2`. You should see the FVP launch, presenting a graphical interface showing information about the current state of the FVP.

The `run_model.sh` script in `<TC2_WORKSPACE>/run-scripts/tc2` will launch the FVP, providing the previously built images as arguments. Run the `./run_model.sh` script:

```
./run_model.sh
Incorrect script use, call script as:
<path_to_run_model.sh> [OPTIONS]
OPTIONS:
-m, --model                    path to model
-d, --distro                   distro version, values supported [buildroot, android-
↪fvp, debian]
```

(continues on next page)

(continued from previous page)

```
-b, --bl33                          BL33 software, values supported [uefi, u-boot]
-a, --avb                           [OPTIONAL] avb boot, values supported [true, false],␣
↪DEFAULT: false
-t, --tap-interface                 [OPTIONAL] enable TAP interface
-n, --networking                    [OPTIONAL] networking, values supported [user, tap,␣
↪none]

                                    DEFAULT: tap if tap interface provided, otherwise user
--                                  [OPTIONAL] After -- pass all further options directly␣
↪to the model
```

Running Debian (UEFI boot with ACPI support)

The TC2 FVP with Debian (UEFI boot with ACPI support) will require to enable the tap interface, since `systemd` services of Debian require network access while booting. This can be ensured using the following command:

```
# following command does assume that current location is <TC2_WORKSPACE>
./run-scripts/tc2/run_model.sh -m <model binary path> -d debian -b uefi -t tap0
```

Expected behaviour

When the script is run, four terminal instances will be launched:

- `terminal_uart_ap` used by the non-secure world components EDK2, Grub, Linux Kernel and filesystem (Debian);
- `terminal_uart1_ap` used by the secure world components TF-A, Hafnium and OP-TEE;
- `terminal_s0` used for the SCP logs;
- `terminal_s1` used by RSS logs (no output by default).

Once the FVP is running, the hardware Root of Trust will verify AP and SCP images, initialize various crypto services and then handover execution to the SCP. SCP will bring the AP out of reset. The AP will start booting from its ROM and then proceed to boot Trusted Firmware-A, Hafnium, Secure Partitions (OP-TEE and Trusted Services). Following this stage, EDK2 UEFI FW and Grub bootloader will take place, and finally the corresponding Linux Kernel distro boot will happen.

When booting Debian, the model will boot the Linux kernel and present a login prompt on the `terminal_uart_ap` window. Login using the username `root` (no password is required). You may need to hit `Enter` for the prompt to appear.

The GUI window `Fast Models - Total Compute 2 DP0` is intended to show any rendered pixels, but this feature is not currently supported for the provided Debian image on the current release.

1.4.7 Running sanity tests

This section provides information on some of the suggested sanity tests that can be executed to exercise and validate the TC Software stack functionality, as well as information regarding the expected behaviour and test results.

Note: The information presented for any of the sanity tests described in this section should NOT be considered as indicative of hardware performance. These tests and the FVP model are only intended to validate the functional flow and behaviour for each of the features.

ACS (UEFI boot with ACPI support)

To run ACS (UEFI boot with ACPI support), please proceed as follows:

1. build the stack for UEFI enabled Debian distro;
2. download the latest ACS disk image from here. This will download a compressed ACS disk prebuilt image called `sr_acs_live_image.img.xz`;

```
# download sr_acs_live_image.img.xz to the root folder of <TC2_WORKSPACE>␣
↪using wget util
cd <TC2_WORKSPACE>
wget --show-progress -O sr_acs_live_image.img.xz https://github.com/ARM-
↪software/arm-systemready/raw/main/SR/prebuilt_images/v23.09_2.0.0/sr_acs_
↪live_image.img.xz
```

3. extract the compressed ACS disk image by running the following command:

```
xz -d sr_acs_live_image.img.xz
```

4. setup the stack for running the ACS test suite by running the following command:

```
# following commands do assume that current location is <TC2_WORKSPACE>
mkdir -p ./output/acs-test-suite/deploy
ln -sf $(pwd)/output/debian/deploy/* ./output/acs-test-suite/deploy/
cp sr_acs_live_image.img ./output/acs-test-suite/deploy/tc2/
```

5. ACS test suite can be executed running the following command:

```
# following command does assume that current location is <TC2_WORKSPACE>
./run-scripts/tc2/run_model.sh -m <model binary path> -d acs-test-suite -b␣
↪uefi
```

Note: An example of the expected test result for this sanity test is illustrated in the related *Total Compute Platform Expected Test Results* document section.

ACPI Test Suite

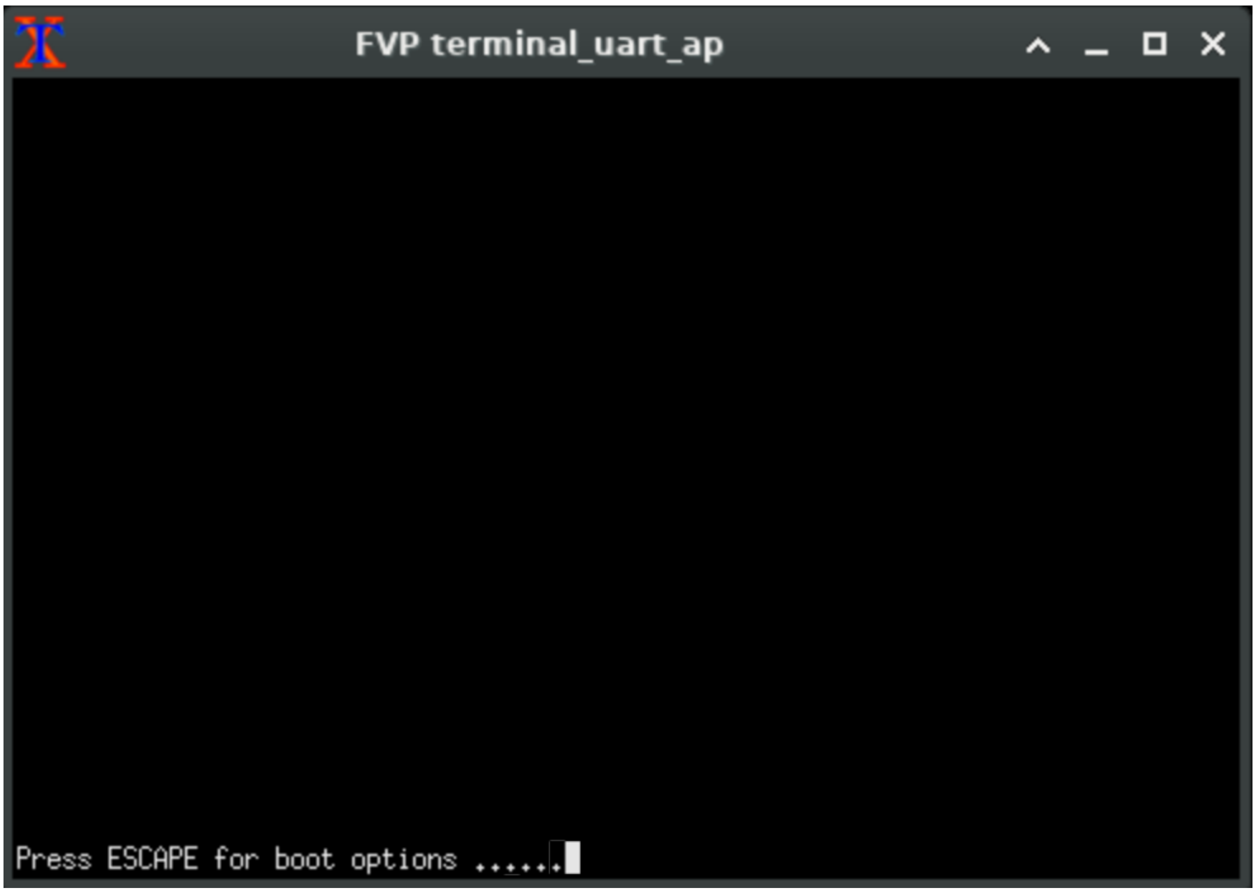
Verify the ACPI tables in UEFI shell

To verify all the ACPI tables in UEFI shell, please proceed as described below:

1. start the TC2 FVP model running Debian and pay close attention to the FVP `terminal_uart_ap` window (as you need to be very quick to succeed on the next step):

```
# following command does assume that current location is <TC2_WORKSPACE>
./run-scripts/tc2/run_model.sh -m <model binary path> -d debian -b uefi -t tap0
```

2. once the **Press ESCAPE for boot options …** message appears, quickly press `ESC` key to interrupt the initial boot and launch the boot options menu:



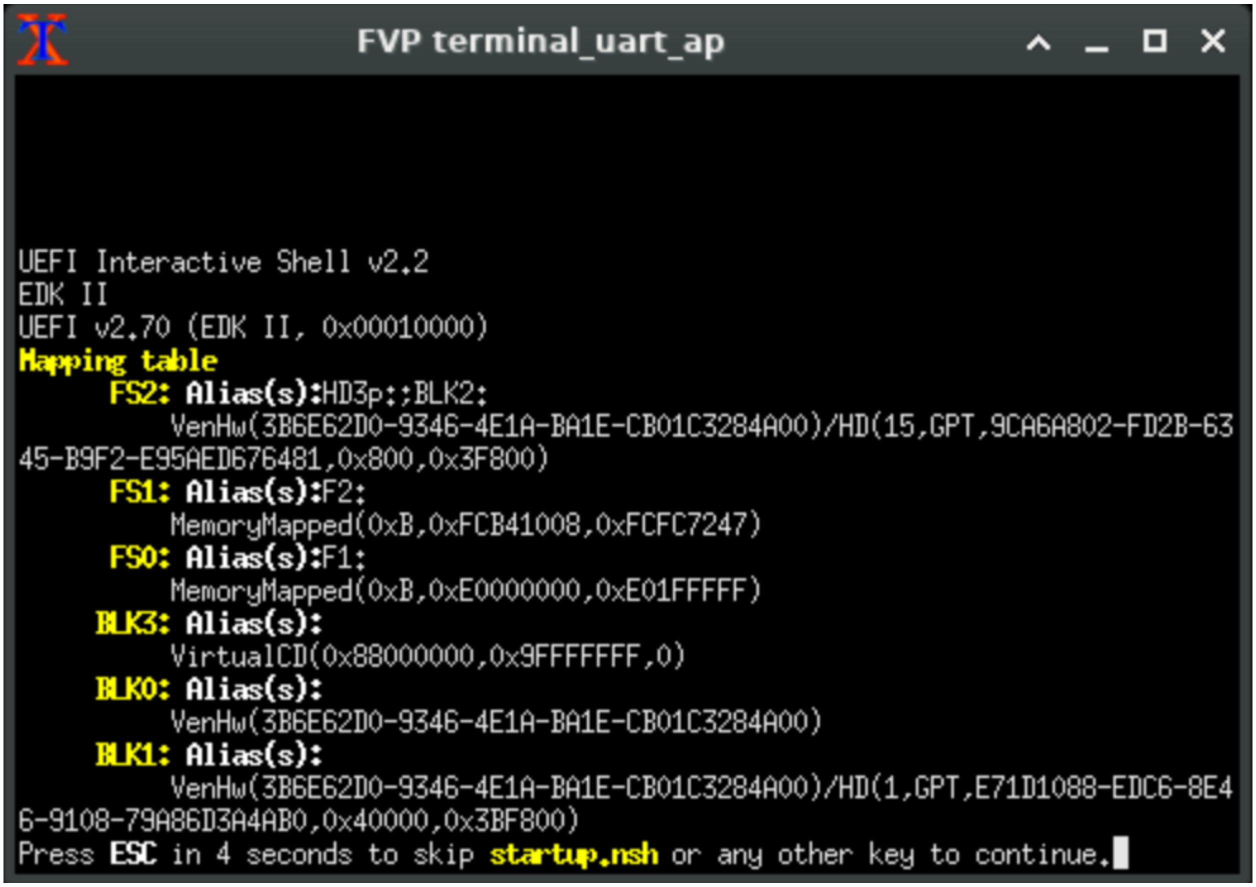
3. using the navigation keys on your keyboard, select the `Boot Manager` option as illustrated on the next image and press `ENTER` key to select it:

4. select the `UEFI Shell` option and press the `ENTER` key:


FVP terminal_uart_ap
Select Language
<Standard English>
This selection will take you to the Boot Manager
> Device Manager
> Boot Manager
> Boot Maintenance Manager
Continue
Reset
^v=Move Highlight
<Enter>=Select Entry


FVP terminal_uart_ap
Boot Manager
Boot Manager Menu
UEFI Shell
UEFI Misc Device
UEFI Misc Device 2
UEFI Non-Block Boot Device
UEFI Non-Block Boot Device 2
Use the <^> and <v> keys to choose a boot option,
the <Enter> key to select a boot option, and the
<Esc> key to exit the Boot Manager Menu.
Device Path :
MemoryMapped(0xB,0xFCB
41008,0xFCFC7247)/FvFi
le(7C04A583-9E3E-4F1C-
AD65-E05268D0B4D1)
^v=Move Highlight
<Enter>=Select Entry
Esc=Exit

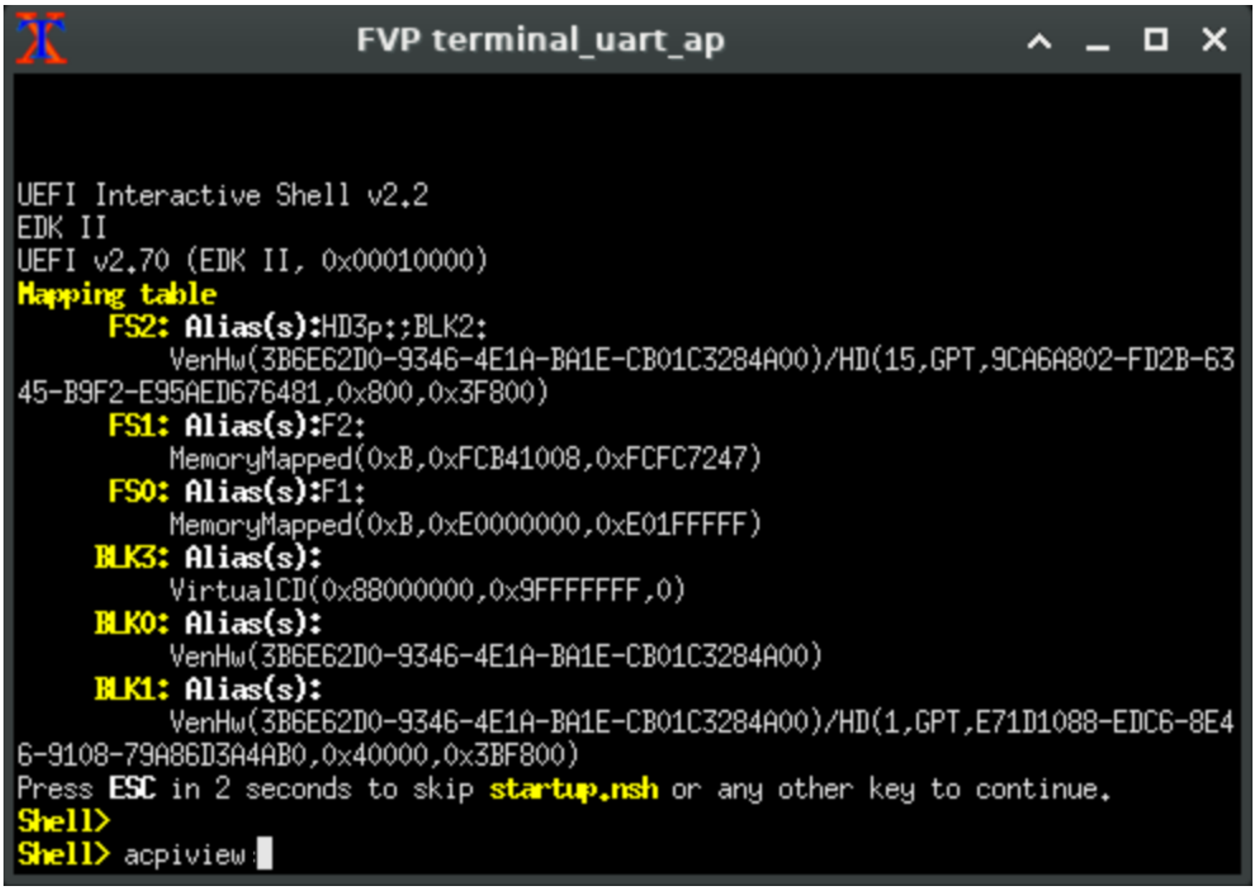
5. allow the platform to boot into the UEFI shell (the ENTER key can be pressed to skip the 5 seconds waiting if desired):



6. once the UEFI shell prompt appears, dump the ACPI content by running the command acpiview as illustrated on the next image:

It is possible to filter the output to a single ACPI table by specifying the respective table name of interest. This can be achieved by running the command acpiview -s <TABLE-NAME>, where <TABLE-NAME> can be any of the following values: FACP, DSDT, DBG2, GTDT, SPCR, APIC, PPTT or SSDT.

Note: This test is specific to Debian with UEFI ACPI support only. An example of the expected test result for this test is illustrated in the related *Total Compute Platform Expected Test Results* document section.


FVP terminal_uart_ap
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS2: Alias(s):HD3p::BLK2:
VenHw(3B6E62D0-9346-4E1A-BA1E-CB01C3284A00)/HD(15,GPT,9CA6A802-FD2B-6345-B9F2-E95AED676481,0x800,0x3F800)
FS1: Alias(s):F2:
MemoryMapped(0xB,0xFCB41008,0xFCFC7247)
FS0: Alias(s):F1:
MemoryMapped(0xB,0xE0000000,0xE01FFFFF)
BLK3: Alias(s):
VirtualCD(0x88000000,0x9FFFFFFF,0)
BLK0: Alias(s):
VenHw(3B6E62D0-9346-4E1A-BA1E-CB01C3284A00)
BLK1: Alias(s):
VenHw(3B6E62D0-9346-4E1A-BA1E-CB01C3284A00)/HD(1,GPT,E71D1088-EDC6-8E46-9108-79A86D3A4AB0,0x40000,0x3BF800)
Press ESC in 2 seconds to skip startup.nsh or any other key to continue.
Shell>
Shell> acpiview

Verify PPTT ACPI table content in Debian shell

The following screenshot exemplifies how to dump the **data cache information** of the CPU cores while in Debian shell (command can be run on the `terminal_uart_ap` window):

The following screenshot exemplifies how to dump the **instruction cache information** of the CPU cores while in Debian shell (command can be run on the `terminal_uart_ap` window):

The following screenshot exemplifies how to dump the **L2 cache information** of the CPU cores while in Debian shell (command can be run on the `terminal_uart_ap` window):

Note: This test is specific to Debian with UEFI ACPI support only.

1.4.8 Debugging on Arm Development Studio

This section describes the steps to debug the TC software stack using Arm Development Studio.

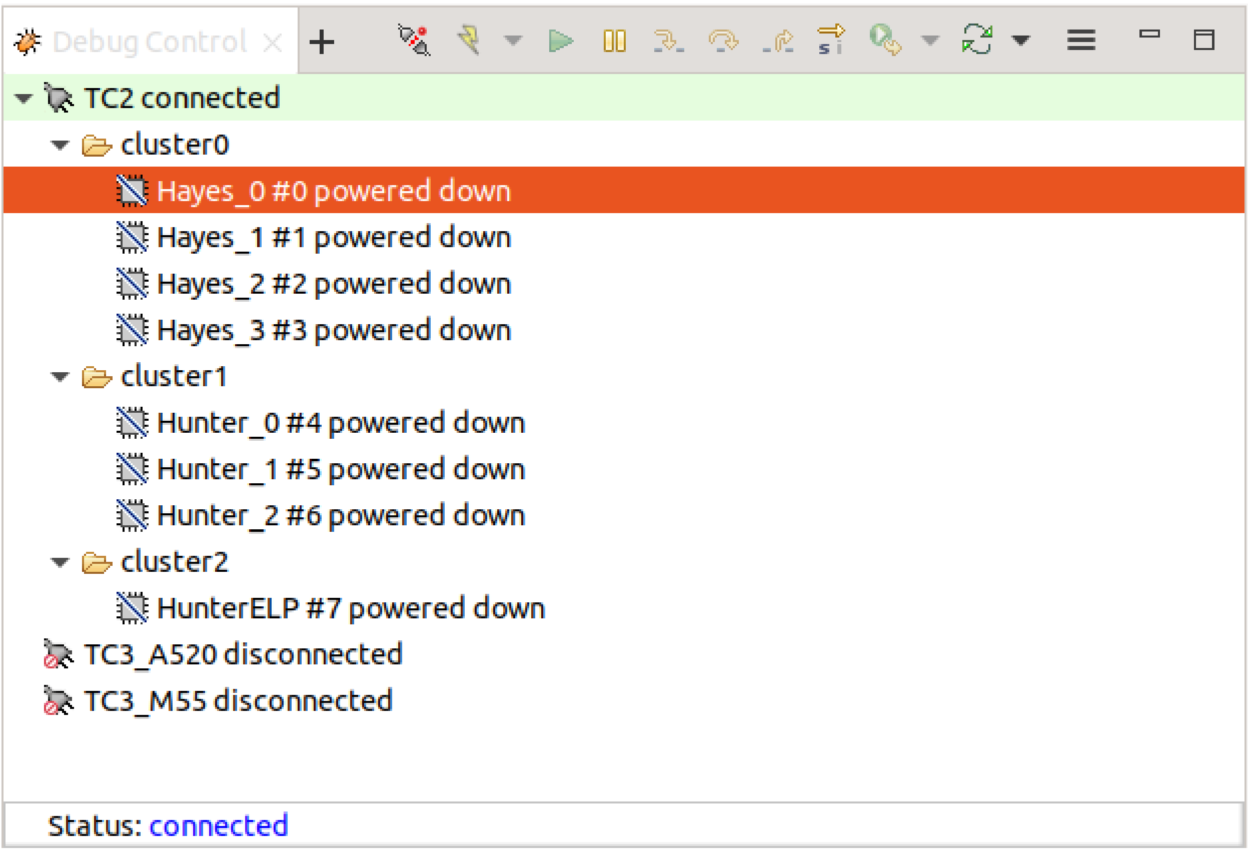
Attach and Debug

1. Build the target with debug enabled (the file <TC2_WORKSPACE>/build-scripts/config can be configured to enable debug);
2. Run the distro as described in the section `Running the software on FVP` with the extra parameters `-- -I` to attach to the debugger. The full command should look like the following:

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d debian -b uefi -- -
↪I
```

3. Select the target `Arm FVP -> TC2 -> Bare Metal Debug -> Hayesx4/Hunterx3/HunterELP SMP`;

4. After connection, use the options in debug control console (highlighted in the below diagram) or the keyboard shortcuts to `step`, `run` or `halt`;
5. To add debug symbols, right click on target -> `Debug configurations` and under `files` tab add path to `elf` files;
6. Debug options such as `break points`, `variable watch`, `memory view` and so on can be used.



Note: This configuration requires Arm DS version 2023.a or later. The names of the cores shown are based on codenames instead of product names. The mapping for the actual names follows the below described convention:

Codename	Product name
Hayes	Cortex A520
Hunter	Cortex A720
Hunter ELP	Cortex X4

Switch between SCP and AP

1. Right click on target and select `Debug Configurations`;
2. Under `Connection`, select `Cortex-M3` for SCP or any of the remaining targets to attach to a specific AP (please refer to the previous note regarding the matching between the used codenames and actual product names);
3. Press the `Debug` button to confirm and start your debug session.

Enable LLVM parser (for Dwarf5 support)

To enable LLVM parser (with Dwarf5 support), please follow the next steps:

1. Select `Window->Preferences->Arm DS->Debugger->Dwarf Parser`;
2. Tick the `Use LLVM DWARF parser` option;
3. Click the `Apply and Close` button.

Arm DS version

The previous steps apply to the following Arm DS Platinum version/build:

Note: Arm DS Platinum is only available to licensee partners. Please contact Arm to have access (support@arm.com).

1.4.9 Feature Guide

Set up TAP interface

This section details the steps required to set up the tap interface on the host to enable model networking.

The following method relies on `libvirt` handling the network bridge. This solution provides a safer approach in which, in cases where a bad configuration is used, the primary network interface should continue operational.

Steps to set up the tap interface

To set up the tap interface, please follow the next steps (unless otherwise mentioned, all commands are intended to be run on the host system):

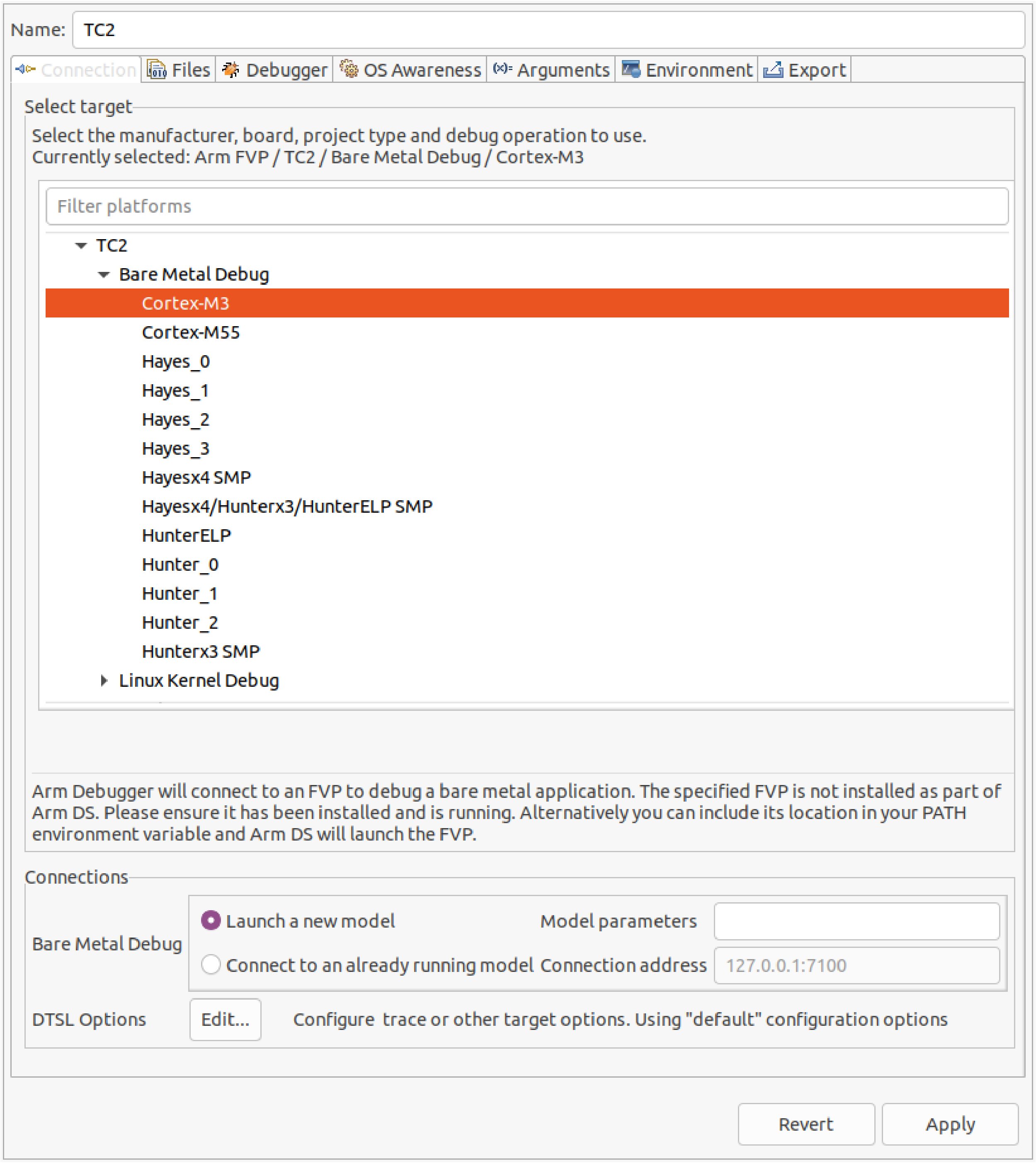
1. install `libvirt` on your development host system:

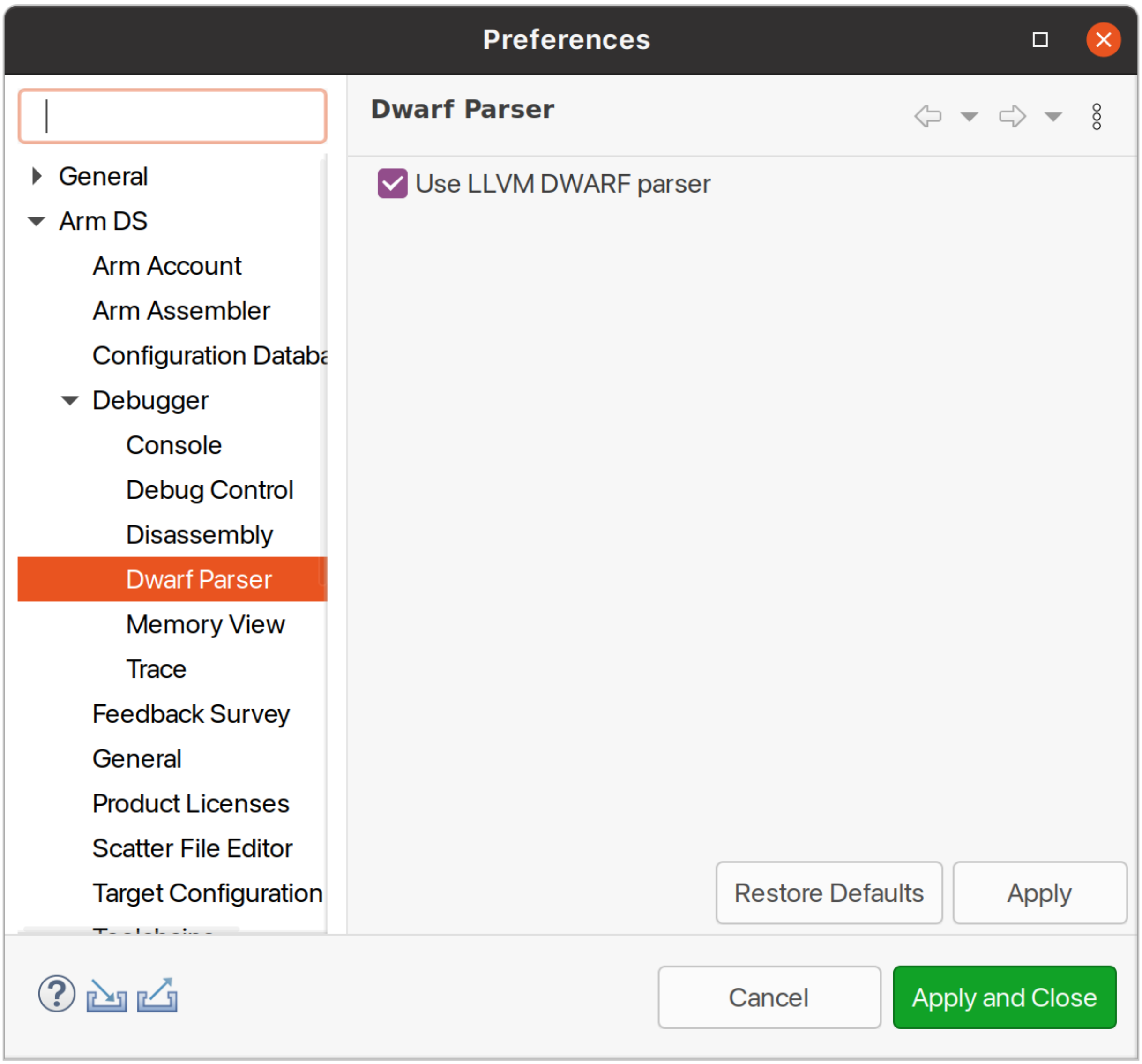
   ```
   sudo apt-get update && sudo apt-get install libvirt-daemon-system libvirt-
   ↪clients
   ```

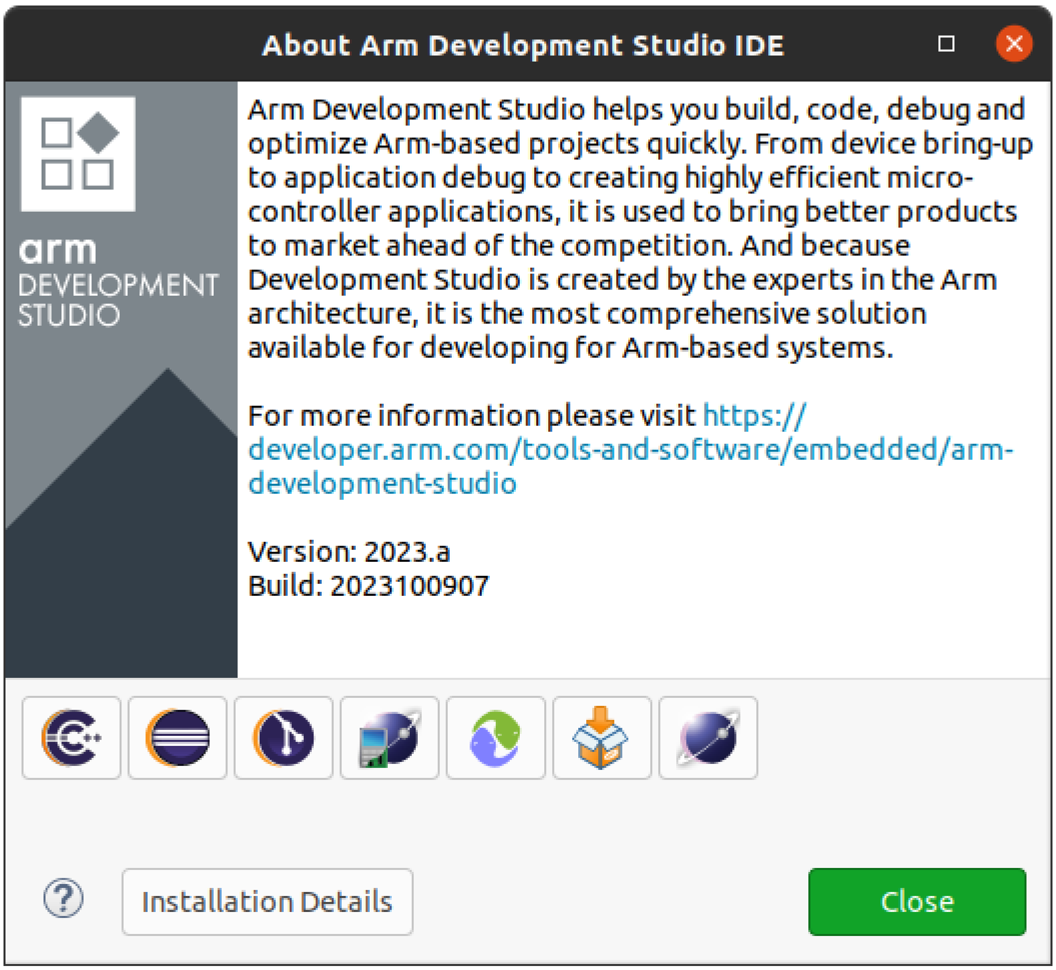
   The host system should now list a new interface with a name similar to `virbr0` and an IP address of `192.168.122.1`. This can be verified by running the command `ifconfig -a` (or alternatively `ip a s` for newer distributions) which will produce an output similar to the following:

   ```
   $ ifconfig -a
   virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
   inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
   ether XX:XX:XX:XX:XX:XX  txqueuelen 1000  (Ethernet)
   ```

(continues on next page)


Name: TC2
Connection
Files
Debugger
OS Awareness
Arguments
Environment
Export
Select target
Select the manufacturer, board, project type and debug operation to use.
Currently selected: Arm FVP / TC2 / Bare Metal Debug / Cortex-M3
Filter platforms
TC2
Bare Metal Debug
Cortex-M3
Cortex-M55
Hayes_0
Hayes_1
Hayes_2
Hayes_3
Hayesx4 SMP
Hayesx4/Hunterx3/HunterELP SMP
HunterELP
Hunter_0
Hunter_1
Hunter_2
Hunterx3 SMP
Linux Kernel Debug
Arm Debugger will connect to an FVP to debug a bare metal application. The specified FVP is not installed as part of Arm DS. Please ensure it has been installed and is running. Alternatively you can include its location in your PATH environment variable and Arm DS will launch the FVP.
Connections
Bare Metal Debug
Launch a new model
Model parameters
Connect to an already running model
Connection address
127.0.0.1:7100
DTSL Options
Edit...
Configure trace or other target options. Using "default" configuration options
Revert
Apply


Preferences
Dwarf Parser
Use LLVM DWARF parser
General
Arm DS
Arm Account
Arm Assembler
Configuration Databa
Debugger
Console
Debug Control
Disassembly
Dwarf Parser
Memory View
Trace
Feedback Survey
General
Product Licenses
Scatter File Editor
Target Configuration
Restore Defaults
Apply
Cancel
Apply and Close


About Arm Development Studio IDE
arm
DEVELOPMENT
STUDIO
Arm Development Studio helps you build, code, debug and optimize Arm-based projects quickly. From device bring-up to application debug to creating highly efficient micro-controller applications, it is used to bring better products to market ahead of the competition. And because Development Studio is created by the experts in the Arm architecture, it is the most comprehensive solution available for developing for Arm-based systems.
For more information please visit https://developer.arm.com/tools-and-software/embedded/arm-development-studio
Version: 2023.a
Build: 2023100907
Installation Details
Close

(continued from previous page)

```
RX packets 0  bytes 0 (0.0 B)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 0  bytes 0 (0.0 B)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0-nic: flags=4098<BROADCAST,MULTICAST>  mtu 1500
ether XX:XX:XX:XX:XX:XX  txqueuelen 1000  (Ethernet)
RX packets 0  bytes 0 (0.0 B)
RX errors 0  dropped 0  overruns 0  frame 0
TX packets 0  bytes 0 (0.0 B)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
$
```

2. create the `tap0` interface:

   ```
   sudo ip tuntap add dev tap0 mode tap user $(whoami)
   sudo ifconfig tap0 0.0.0.0 promisc up
   sudo brctl addif virbr0 tap0
   ```

3. run the FVP model providing the additional parameter `-t "tap0"` to enable the tap interface:

   ```
   ./run-scripts/tc2/run_model.sh -m <model binary path> -d debian -b uefi -t
   ↪"tap0"
   ```

   Before proceeding, please allow FVP model to fully boot.

4. once the FVP model boots, the running instance should get an IP address similar to `192.168.122.62`;

5. validate the connection between the host `tap0` interface and the FVP model by running the following command **on the fvp-model** via the `terminal_uart_ap` window:

   ```
   ping 192.168.122.1
   ```

   Alternatively, it is also possible to validate if the fvp-model can reach a valid internet gateway by pinging, for instance, the IP address `8.8.8.8` instead.

Steps to graceful disable and remove the tap interface

To revert the configuration of your host system (removing the `tap0` interface), please follow the next steps:

1. remove the `tap0` from the bridge configuration:

   ```
   sudo brctl delif virbr0 tap0
   ```

2. disable the bridge interface:

   ```
   sudo ip link set virbr0 down
   ```

3. remove the bridge interface:

   ```
   sudo brctl delbr virbr0
   ```

4. remove the `libvirt` package:

```
sudo apt-get remove libvirt-daemon-system libvirt-clients
```

Running and Collecting FVP tracing information

This section describes how to run the FVP-model, enabling the output of trace information for debug and troubleshooting purposes. To illustrate proper trace output information that can be obtained at different stages, the following command examples will use the SMMU-700 block component. However, any of the commands mentioned, can be extended or adapted easily for any other component.

Note: This functionality requires to execute the FVP-model enforcing the additional load of the `GenericTrace.so` or `ListTraceSources.so` plugins (which are provided and part of your FVP bundle).

Getting the list of trace sources

To get the list of trace sources available on the FVP-model, please run the following command:

```
<fvp-model binary path>/FVP_TC2 \
        --plugin <fvp-model plugin path/ListTraceSources.so> \
        >& /tmp/trace-sources-fvp-tc2.txt
```

This will start the model and use the `ListTraceSources.so` plugin to dump the list to a file. Please note that the file size can easily extend to tens of megabytes, as the list is quite extensive.

The following excerpt illustrates the output information related with the example component SMMU-700:

```
Component (1439) providing trace: TC2.css.smmu (MMU_700, 11.23.28)
=============================================================================
Component is of type "MMU_700"
Version is "11.23.28"
#Sources: 299

Source ArchMsg.Error.error (These messages are about activity occurring on the␣
↪SMMU that is considered an error.
Messages will only come out here if parameter all_error_messages_through_trace␣
↪is true.

DISPLAY %{output})
        Field output type:MTI_STRING size:0 max_size:120 (The stream output)

Source ArchMsg.Error.fetch_from_memory_type_not_supporting_httu (A descriptor␣
↪fetch from an HTTU-enabled translation regime to an unsupported
memory type was made.  Whilst the fetch itself may succeed, if an update to
the descriptor was attempted then it would fail.)
```

Executing the FVP-model with traces enabled

To execute the FVP-model with trace information enabled, please run the following command:

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d debian -b uefi \
        -- \
        --plugin <fvp-model plugin path/GenericTrace.so> \
        -C 'TRACE.GenericTrace.trace-sources="TC2.css.smmu.*"' \
        -C TRACE.GenericTrace.flush=true
```

Multiple trace sources can be requested by separating the trace-sources strings with commas. By default, the trace information will be displayed to the standard output (e.g. display), which due to its verbosity may not be always the ideal solution. For such situations, it is suggested to redirect and capture the trace information into a file, which can be achieved by running the following command:

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d debian -b uefi \
        -- \
        --plugin <fvp-model plugin path/GenericTrace.so> \
        -C 'TRACE.GenericTrace.trace-sources="TC2.css.smmu.*"' \
        -C TRACE.GenericTrace.flush=true \
        >& /tmp/trace-fvp-tc2.txt
```


1.5 Expected test results

Contents

1.5.1 ACS (UEFI boot with ACPI support) Test Suite unit tests

```
(...output truncated...)

remove-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-
↪test/uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/Application/StallForKey/
↪StallForKey/DEBUG/StallForKey.dll 0xF7A7B000
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestInfrastructure/SCT/Framework/Sct/
↪DEBUG/SCT.dll 0xF7422000
Loading driver at 0x000F7421000 EntryPoint=0x000F7427728 SCT.efi
Load support files ...
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestInfrastructure/SCT/Drivers/
```

(continues on next page)

(continued from previous page)

```
↪TestProfile/TestProfile/DEBUG/TestProfile.dll 0xF74DA000
Loading driver at 0x000F74D9000 EntryPoint=0x000F74DC348 TestProfile.efi
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestInfrastructure/SCT/Drivers/
↪StandardTest/StandardTest/DEBUG/StandardTest.dll 0xF7416000
Loading driver at 0x000F7415000 EntryPoint=0x000F741C164 StandardTest.efi
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestInfrastructure/SCT/Drivers/
↪TestRecovery/TestRecovery/DEBUG/TestRecovery.dll 0xF74D7000
Loading driver at 0x000F74D6000 EntryPoint=0x000F74D7784 TestRecovery.efi
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestInfrastructure/SCT/Drivers/
↪TestLogging/TestLogging/DEBUG/TestLogging.dll 0xF740F000
Loading driver at 0x000F740E000 EntryPoint=0x000F74121E0 TestLogging.efi
Load proxy files ...
Load test files ...
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestCase/UEFI/EFI/Protocol/BlockIo2/
↪BlackBoxTest/BlockIo2BBTest/DEBUG/BlockIo2BBTest.dll 0xF73F7000
Loading driver at 0x000F73F6000 EntryPoint=0x000F73FF984 BlockIo2BBTest.efi
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestCase/UEFI/EFI/Protocol/EraseBlock/
↪BlackBoxTest/EraseBlockBBTest/DEBUG/EraseBlockBBTest.dll 0xF73F1000
Loading driver at 0x000F73F0000 EntryPoint=0x000F73F2874 EraseBlockBBTest.efi
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestCase/UEFI/EFI/Protocol/
↪UFSDeviceConfig/BlackBoxTest/UFSDeviceConfigBBTest/DEBUG/UFSDeviceConfigBBTest.dll␣
↪0xF73EC000Loading driver at 0x000F73EB000 EntryPoint=0x000F73EC7FC␣
↪UFSDeviceConfigBBTest.efi
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestCase/UEFI/EFI/Protocol/
↪ComponentName2/BlackBoxTest/ComponentName2BBTest/DEBUG/ComponentName2BBTest.dll␣
↪0xF73E2000
Loading driver at 0x000F73E1000 EntryPoint=0x000F73E627C ComponentName2BBTest.efi
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestCase/UEFI/EFI/Protocol/IPsecConfig/
↪BlackBoxTest/IPsecConfigBBTest/DEBUG/IPsecConfigBBTest.dll 0xF73D6000
Loading driver at 0x000F73D5000 EntryPoint=0x000F73DA06C IPsecConfigBBTest.efi
add-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/
↪uefi-sct/Build/bbrSct/DEBUG_GCC5/AARCH64/SctPkg/TestCase/UEFI/EFI/Protocol/DiskIo2/
↪BlackBoxTest/DiskIo2BBTest/DEBUG/DiskIo2BBTest.dll 0xF73C4000
Loading driver at 0x000F73C3000 EntryPoint=0x000F73CCE80 DiskIo2BBTest.efi

(...output truncated...)

Loading driver at 0x000F737A000 EntryPoint=0x000F737E178 TimeServicesBBTest.efi

Loading driver at 0x000F70A4000 EntryPoint=0x000F70A5C18 RamDiskProtocolBBTest.efi
Test preparing...
  Remaining test cases: 330
  Generic services test: PlatformSpecificElements
  Iterations: 1/1
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------
Arm ACS Version: v2.0.0-BETA-0
PlatformSpecificElements
Revision 0x00010001
Test Entry Point GUID: A0A8BED3-3D6F-4AD8-907A-84D52EE1543B
Test Support Library GUIDs:
  1F9C2AE7-F147-4D19-A5E8-255AD005EB3E
  832C9023-8E67-453F-83EA-DF7105FA7466
------------------------------------------------------------
UEFI 2.6
Test Configuration #0
------------------------------------------------------------
Check the platform specific elements defined in the UEFI Spec Section 2.6.2
------------------------------------------------------------
Logfile: "\EFI\BOOT\bbr\SCT\Log\GenericTest\EFICompliantTest0\PlatformSpecificEl
ements_0_0_A0A8BED3-3D6F-4AD8-907A-84D52EE1543B.log"
Test Started: 03/13/24  06:02p
------------------------------------------------------------
UEFI Compliant - Console protocols must be implemented -- PASS
8F7556C2-4665-4353-A3AF-9C005A1E63E1
/home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/uefi-sct/
SctPkg/TestCase/UEFI/EFI/Generic/EfiCompliant/BlackBoxTest/EfiCompliantBBTestPla
tform_uefi.c:1022:Text Input - Yes, Text Output - Yes, Text InputEx - Yes

UEFI Compliant - Hii protocols must be implemented -- PASS
B7CD2D76-EA43-4013-B7D1-59EB2EC9BF1B
/home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/uefi-sct/
SctPkg/TestCase/UEFI/EFI/Generic/EfiCompliant/BlackBoxTest/EfiCompliantBBTestPla
tform_uefi.c:1106:HiiDatabase - Yes, HiiString - Yes, HiiConfigRouting - Yes, Hi
iConfigAccess - Yes

UEFI Compliant - Hii protocols must be implemented -- PASS
B7CD2D76-EA43-4013-B7D1-59EB2EC9BF1B
/home/runner/work/arm-systemready/arm-systemready/SR/scripts/edk2-test/uefi-sct/
SctPkg/TestCase/UEFI/EFI/Generic/EfiCompliant/BlackBoxTest/EfiCompliantBBTestPla
tform_uefi.c:1150:HiiFont - Yes

(...output truncated...)

          *** Starting Wakeup semantic tests ***

Operating System View:
 501 : Wake from EL1 PHY Timer Int
           Failed on PE -    0
           Checkpoint --  1                              : Result:  FAIL
 502 : Wake from EL1 VIR Timer Int
           Failed on PE -    0
           Checkpoint --  1                              : Result:  FAIL
 503 : Wake from EL2 PHY Timer Int
           Failed on PE -    0
           Checkpoint --  1                              : Result:  FAIL
 504 : Wake from Watchdog WS0 Int
```

(continues on next page)

(continued from previous page)

```
        Invalid SPI interrupt ID number 30     : Result:  PASS
 505 : Wake from System Timer Int            : Result:  A01F9000

          One or more Wakeup tests failed or were skipped.

          *** Starting Peripheral tests ***

Operating System View:
 601 : USB CTRL Interface
           Checkpoint --  1                          : Result:  SKIPPED
 602 : Check SATA CTRL Interface
           Checkpoint --  1                          : Result:  SKIPPED
 603 : Check Arm BSA UART register offsets        : Result:  PASS
 604 : Check Arm GENERIC UART Interrupt
           Test Message                              : Result:  PASS
 606 : 16550 compatible UART
           Checkpoint --  2                          : Result:  SKIPPED

          One or more Peripheral tests failed or were skipped.

          *** Starting Watchdog tests ***

Operating System View:
 701 : Non Secure Watchdog Access                 : Result:  PASS
 702 : Check Watchdog WS0 interrupt               : Result:  PASS

          All Watchdog tests passed.

          *** No ECAM region found, Skipping PCIE tests ***

         -------------------------------------------------------
         Total Tests run  =   45  Tests Passed  =   34  Tests Failed =    7
         -------------------------------------------------------

          *** BSA tests complete. Reset the system. ***

(...output truncated...)
```

Note: To obtain more information on how to run this test, please refer to the *Total Compute Platform User Guide - Running sanity tests* document section.

1.5.2 ACPI Test Suite unit tests

```
# acpiview -s
                              acpiview

 --------------- RSDP Table ---------------

Address  : 0xF7870018
Length   : 36

00000000 : 52 53 44 20 50 54 52 20 - 1B 41 52 4D 4C 54 44 02   RSD PTR .ARMLTD.
00000010 : 00 00 00 00 24 00 00 00 - 98 FE 87 F7 00 00 00 00   ....$...........
00000020 : C8 00 00 00                                          ....

Table Checksum : OK

RSDP                                 :
  Signature                          : RSD PTR
  Checksum                           : 0x1B
  Oem ID                             : ARMLTD
  Revision                           : 2
  RSDT Address                       : 0x0
  Length                             : 36
  XSDT Address                       : 0xF787FE98
  Extended Checksum                  : 0xC8
  Reserved                           : 0 0 0

 --------------- XSDT Table ---------------

Address  : 0xF787FE98
Length   : 92

00000000 : 58 53 44 54 5C 00 00 00 - 01 0A 41 52 4D 4C 54 44   XSDT\.....ARMLTD
00000010 : 41 52 4D 54 43 20 20 20 - 27 07 14 20 20 20 20 20   ARMTC   '..
00000020 : 13 00 00 01 98 FB 87 F7 - 00 00 00 00 98 FA 87 F7   ................
00000030 : 00 00 00 00 98 E9 87 F7 - 00 00 00 00 18 FE 87 F7   ................
00000040 : 00 00 00 00 98 F5 87 F7 - 00 00 00 00 18 EB 87 F7   ................
00000050 : 00 00 00 00 18 EF 87 F7 - 00 00 00 00               ............

Table Checksum : OK

XSDT                                 :
  Signature                          : XSDT
  Length                             : 92
  Revision                           : 1
  Checksum                           : 0xA
  Oem ID                             : ARMLTD
  Oem Table ID                       : ARMTC
  Oem Revision                       : 0x20140727


(...output truncated...)
```

(continues on next page)

(continued from previous page)

```
 --------------- SSDT Table ---------------

Address  : 0xF787EF18
Length   : 403

00000000 : 53 53 44 54 93 01 00 00 - 02 C3 41 52 4D 4C 54 44   SSDT......ARMLTD
00000010 : 41 52 4D 54 43 00 00 00 - 27 07 14 20 49 4E 54 4C   ARMTC...'.. INTL
00000020 : 28 06 23 20 10 4E 16 5F - 53 42 5F 5B 82 41 05 43   (.# .N._SB_[.A.C
00000030 : 4F 4D 30 08 5F 48 49 44 - 0D 41 52 4D 48 30 30 31   OM0._HID.ARMH001
00000040 : 31 00 08 5F 43 49 44 0D - 41 52 4D 48 30 30 31 31   1.._CID.ARMH0011
00000050 : 00 08 5F 55 49 44 00 08 - 5F 53 54 41 0A 0F 08 5F   .._UID.._STA..._
00000060 : 43 52 53 11 1A 0A 17 86 - 09 00 01 00 00 40 2A 00   CRS..........@*.
00000070 : 10 00 00 89 06 00 01 01 - 5F 00 00 00 79 00 5B 82   ........_...y.[.
00000080 : 41 05 43 4F 4D 31 08 5F - 48 49 44 0D 41 52 4D 48   A.COM1._HID.ARMH
00000090 : 30 30 31 31 00 08 5F 43 - 49 44 0D 41 52 4D 48 30   0011.._CID.ARMH0
000000A0 : 30 31 31 00 08 5F 55 49 - 44 01 08 5F 53 54 41 0A   011.._UID.._STA.
000000B0 : 0F 08 5F 43 52 53 11 1A - 0A 17 86 09 00 01 00 00   .._CRS..........
000000C0 : 40 2A 00 10 00 00 89 06 - 00 01 01 00 00 00 00 79   @*.............y
000000D0 : 00 5B 82 41 04 56 52 30 - 30 08 5F 48 49 44 0D 4C   .[.A.VR00._HID.L
000000E0 : 4E 52 4F 30 30 30 35 00 - 08 5F 55 49 44 00 08 5F   NRO0005.._UID.._
000000F0 : 43 43 41 01 08 5F 43 52 - 53 11 1A 0A 17 86 09 00   CCA.._CRS.......
00000100 : 01 00 00 13 1C 00 00 01 - 00 89 06 00 01 01 EC 00   ................
00000110 : 00 00 79 00 5B 82 41 04 - 56 52 30 31 08 5F 48 49   ..y.[.A.VR01._HI
00000120 : 44 0D 4C 4E 52 4F 30 30 - 30 35 00 08 5F 55 49 44   D.LNRO0005.._UID
00000130 : 01 08 5F 43 43 41 01 08 - 5F 43 52 53 11 1A 0A 17   .._CCA.._CRS....
00000140 : 86 09 00 01 00 00 15 1C - 00 00 01 00 89 06 00 01   ................
00000150 : 01 ED 00 00 00 79 00 5B - 82 3A 4E 45 54 30 08 5F   .....y.[.:NET0._
00000160 : 48 49 44 0D 4C 4E 52 4F - 30 30 30 33 00 08 5F 55   HID.LNRO0003.._U
00000170 : 49 44 00 08 5F 43 52 53 - 11 1A 0A 17 86 09 00 01   ID.._CRS........
00000180 : 00 00 00 18 00 00 01 00 - 89 06 00 01 01 8D 00 00   ................
00000190 : 00 79 00                                             .y.

Table Checksum : OK

ACPI Table Header                    :
Signature                          : SSDT
  Length                             : 403
  Revision                           : 2
  Checksum                           : 0xC3
  Oem ID                             : ARMLTD
  Oem Table ID                       : ARMTC
  Oem Revision                       : 0x20140727
  Creator ID                         : INTL
  Creator Revision                   : 0x20230628

Table Statistics:
        0 Error(s)
        0 Warning(s)
#
```

Note: To obtain more information on how to run this sanity test, please refer to the *Total Compute Platform User*

Guide - Running sanity tests document section.

1.6 Troubleshooting: common problems and solutions

This section provides a list of potential solutions to the most common problems experienced by developers and related with the host development environment. This list is not intended to be an exhaustive list, especially due to the unpredictability and nature of some problems. The developer is, therefore, strongly encouraged to read and search for more information regarding the problem and any additional solutions (covered or not in this document).

Contents

1.6.1 Docker

Error message: `Cannot Connect to a Docker Daemon`

Solution: Ensure docker service is running, correct permissions and user group membership are properly configured (please refer to *User Guide - prerequisites* document section).

Error message: `transport: dial unix /var/run/docker/containerd/docker-containerd.sock: connect: connection refused`

Solution: Restart docker service running following command: `sudo systemctl restart docker`.

1.7 Release notes - TC2-2024.02.22-LSC

Contents

1.7.1 Release tag

The manifest tag for this release is `TC2-2024.02.22-LSC`.

1.7.2 Components

The following is a summary of the key software features of the release:

- BSP build supporting Debian bookworm distro;
- Trusted firmware-A for secure boot;
- EDK2 bootloader;
- Hafnium for S-EL2 Secure Partition Manager core;
- OP-TEE for Trusted Execution Environment (TEE);
- Trusted Services (Crypto and Internal Trusted Storage);
- System Control Processor(SCP) firmware for programming the interconnect, power control, etc;
- Runtime Security Subsystem (RSS) firmware for providing HW RoT;

1.7.3 Hardware Features

- Booker aka CoreLink CI-700 with Memory Tagging Unit (MTU) support driver in SCP firmware;
- GIC Clayton Initialization in Trusted Firmware-A;
- Mali-G720 GPU;
- Mali-D71 DPU and virtual encoder support for display on Linux;
- MHUv2 Driver for SCP and AP communication;
- UARTs, Timers, Flash, PIK, Clock drivers;
- PL180 MMC;
- DynamIQ Shared Unit (DSU) with 8 cores (1 Cortex X4 + 3 Cortex A720 + 4 Cortex A520 cores configuration);
- RSS based on Cortex M55;
- SCP based on Cortex M3;

1.7.4 Software Features

- Debian 12 (aka Bookworm);
- Linux Kernel 6.1.0;
- EDK2 v202402 (Feb 2024);
- Trusted Firmware-A v2.9;
- Hafnium v2.9 as Secure Partition Manager (SPM) at S-EL2;
- OP-TEE v4.1.0 as Secure Partition at S-EL1, managed by S-EL2 SPMC (Hafnium);
- Support for secure boot based on TBBR specification (more info available at link);
- System Control Processor (SCP) firmware v2.12;
- Runtime Security Subsystem (RSS) firmware v1.8.0;
- VirtIO to mount the Debian image in the host machine as a storage device in the FVP;
- Trusted Services (Crypto and Internal Trusted Storage) running at S-EL0;

1.7.5 Platform Support

- This software release is tested on TC2 Fast Model platform (FVP) version 11.23.28.

1.7.6 Tools Support

- This software release extends docker support to Debian distro (making it supported to all TC build variants).

1.7.7 Known issues or Limitations

1. Ubuntu 22.04 is not supported in this release;
2. SVE2 (Scalable Vector Extension) feature is not supported with this release;
3. `systemd-resolved.service` fails multiple times before successful start, which causes delay in boot time;

   This can be avoided following one of the below workaround steps:

   Method 1 - editing the Kernel command line parameters:

   1. during the boot process, once the Grub screen appears (list of options that allow to choose what to boot), press `e` key to edit the boot command;
   2. navigate to the line with the command line parameters (usually starts with `linux /boot/vmlinux*`), and append `systemd.mask=systemd-resolved.service`;
   3. press `F10` to continue to boot with the added boot parameters.

   Method 2 - make the changes to be permanent for every boot:

   1. wait for the OS to fully boot, login and get to the command prompt;
   2. on the command prompt, run the following command `vi /etc/default/grub` to edit the Grub configuration;
   3. add the following line: `GRUB_CMDLINE_LINUX_DEFAULT="systemd.mask=systemd-resolved.service"`;

4. save the changes and exit the `vi` editor;
5. on the command prompt, run the command `update-grub` to update and apply the new Grub configuration.

4. Upon the completion of running the ACS (UEFI boot with ACPI support) tests, a synchronous exception is experienced, and some dump information will be presented on the `FVP terminal_uart1_ap` window, similar to the following excerpt:

```
(...)
Operating System View:
 701 : Non Secure Watchdog Access                : Result:  PASS
 702 : Check Watchdog WS0 interrupt              : Result:  PASS

          All Watchdog tests passed.

          *** No ECAM region found, Skipping PCIE tests ***

         -------------------------------------------------------
         Total Tests run  =   45  Tests Passed  =   34  Tests Failed =    7
         -------------------------------------------------------

          *** BSA tests complete. Reset the system. ***

remove-symbol-file /home/runner/work/arm-systemready/arm-systemready/SR/
↪scripts/edk2/Build/Shell/DEBUG_GCC49/AARCH64/ShellPkg/Application/bsa-acs/
↪uefi_app/BsaAcs/DEBUG/Bsa.dll 0xF73ED000


Synchronous Exception at 0x00000000F73F9454

Recursive exception occurred while dumping the CPU state
Unhandled Exception in EL3.
x30            = 0x000000000402cb94
x0             = 0x0000000000000000
x1             = 0x000000000000003a
x2             = 0x0000000000000000
x3             = 0x00000000820003c8
x4             = 0x0000000000000000
x5             = 0x000000009600004f
x6             = 0x00000000fc8a4f90
x7             = 0x00000000fc9ccc10
x8             = 0x0000000001000000
x9             = 0x0000000004035200
x10            = 0x0000009fa226e8f3
x11            = 0x0000021e648978a7
x12            = 0x0000000000000002
x13            = 0x0000000000000002
x14            = 0x0000000000000001
x15            = 0x00000000000000ff
x16            = 0x00000000f73f9454
x17            = 0x00000000fcb41000
x18            = 0x00000000fc8a5130
x19            = 0x0000000004035a70
```

(continues on next page)

(continued from previous page)

```
x20             = 0x0000000000000000
x21             = 0x0000000000000000
x22             = 0x0000000000000000
x23             = 0x0000000000000000
x24             = 0x0000000000000000
x25             = 0x0000000000000000
x26             = 0x0000000000000000
x27             = 0x0000000000000000
x28             = 0x0000000000000000
x29             = 0x0000000000000000
scr_el3         = 0x000000401c07073d
sctlr_el3       = 0x00000000b0cd183f
cptr_el3        = 0x0000000000000000
tcr_el3         = 0x000000008081351c
daif            = 0x00000000000003c0
mair_el3        = 0x00000000004404ff
spsr_el3        = 0x00000000630002cd
elr_el3         = 0x0000000004021c30
ttbr0_el3       = 0x000000000403c001
esr_el3         = 0x00000000be000211
far_el3         = 0x0000000000000000
spsr_el1        = 0x0000000000000000
elr_el1         = 0x0000000000000000
spsr_abt        = 0x0000000000000000
spsr_und        = 0x0000000000000000
spsr_irq        = 0x0000000000000000
spsr_fiq        = 0x0000000000000000
sctlr_el1       = 0x0000000030d00980
actlr_el1       = 0x0000000000000000
cpacr_el1       = 0x0000000000300000
csselr_el1      = 0x0000000000000004
sp_el1          = 0x0000000000000000
esr_el1         = 0x0000000000000000
ttbr0_el1       = 0x0000000000000000
ttbr1_el1       = 0x0000000000000000
mair_el1        = 0x0000000000000000
amair_el1       = 0x0000000000000000
tcr_el1         = 0x0000000000000000
tpidr_el1       = 0x0000000000000000
tpidr_el0       = 0x00000000f501edc0
tpidrro_el0     = 0x0000000000000000
par_el1         = 0x0000000000000800
mpidr_el1       = 0x0000000081000000
afsr0_el1       = 0x0000000000000000
afsr1_el1       = 0x0000000000000000
contextidr_el1  = 0x0000000000000000
vbar_el1        = 0x0000000000000000
cntp_ctl_el0    = 0x0000000000000002
cntp_cval_el0   = 0x00000020f49a0984
cntv_ctl_el0    = 0x0000000000000002
cntv_cval_el0   = 0x00000020ecda7b2f
cntkctl_el1     = 0x0000000000000000
```

(continues on next page)

(continued from previous page)

```
sp_el0          = 0x00000000fc8a5130
isr_el1         = 0x0000000000000040
cpuectlr_el1    = 0x0000000000900000
```

5. Upon running the tf-a-tests, a panic in EL3 is experienced, and some dump information will be presented on the FVP `terminal_uart_ap` window, similar to the following excerpt:

```
(...)
Running test suite 'Framework Validation'
Description: Validate the core features of the test framework> Executing
↪'NVM support'
  TEST COMPLETE                                                Passed>␣
↪Executing 'NVM serialisation'
  TEST COMPLETE                                                Passed>␣
↪Executing 'Events API'
ERROR:   Unexpected affinity info state.
BACKTRACE: START: psci_warmboot_entrypoint
0: EL3: 0x4022478
1: EL3: 0x4029aa0
2: EL3: 0x40201d4
3: EL3: 0x4022550
BACKTRACE: END: psci_warmboot_entrypoint
PANIC in EL3.
x30             = 0x0000000004029aac
x0              = 0x0000000000000001
x1              = 0x0000000000000002
x2              = 0x0000000000000097
x3              = 0x00000000ffffffc8
x4              = 0x0000000000000004
x5              = 0x0000000000000000
x6              = 0x000000000402ab70
x7              = 0x0000000000000000
x8              = 0x0000000001000000
x9              = 0x00000000040201c8
x10             = 0x0000000004021cf4
x11             = 0x00000000fd030000
x12             = 0x00000000fd000800
x13             = 0x0000000080000000
x14             = 0x00000000410fd801
x15             = 0x0000000080000000
x16             = 0x0000000000000000
x17             = 0x0000000000000017
x18             = 0x000000401c070f38
x19             = 0x0000000004020140
x20             = 0x0000000000000000
x21             = 0x000000000403b000
x22             = 0x0000000000000000
x23             = 0x0000000000000000
x24             = 0x0000000000000000
x25             = 0x0000000000000000
x26             = 0x0000000000000000
x27             = 0x0000000000000000
```

(continues on next page)

(continued from previous page)

```
x28            = 0x0000000000000000
x29            = 0x0000000004030b90
scr_el3        = 0x0000000000030638
sctlr_el3      = 0x00000000b0cd183f
cptr_el3       = 0x0000000040100000
tcr_el3        = 0x000000008081351c
daif           = 0x00000000000002c0
mair_el3       = 0x00000000004404ff
spsr_el3       = 0x00000000604003c9
elr_el3        = 0x00000000fd0013d0
ttbr0_el3      = 0x000000000403c001
esr_el3        = 0x0000000000000000
far_el3        = 0x0000000000000000
spsr_el1       = 0x0000000000000000
elr_el1        = 0x0000000000000000
spsr_abt       = 0x0000000000000000
spsr_und       = 0x0000000000000000
spsr_irq       = 0x0000000000000000
spsr_fiq       = 0x0000000000000000
sctlr_el1      = 0x0000000030d80998
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        = 0x0000000080000000
tpidr_el1      = 0x0000000000000000
tpidr_el0      = 0x0000000000000000
tpidrro_el0    = 0x0000000000000000
par_el1        = 0xff0000000402a980
mpidr_el1      = 0x0000000081000000
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         = 0x0000000004030b90
isr_el1        = 0x0000000000000000
cpuectlr_el1   = 0x0000000000900000
```

1.7.8 Support

For support email: support@arm.com.

CHAPTER

TWO

PREVIOUS RELEASES

This web page provides a list of all the TotalCompute Software Stack releases, cataloged by major version, which can be used for easy historical reference.

2.1 Latest TC release

TC2-2023.10.04

2.2 TC2 release tags

TC2-2023.08.15

TC2-2023.04.21

TC2-2022.12.07

TC2-2022.08.12

2.3 TC1 release tags

TC1-2022.10.07

TC1-2022.05.12

TC1-2021.08.17

2.4 TC0 release tags

TC0-2022.02.25

TC0-2021.07.31

TC0-2021.04.23

TC0-2021.02.09