Total Compute

Arm Limited

May 03, 2024

TOTAL COMPUTE:

CHAPTER

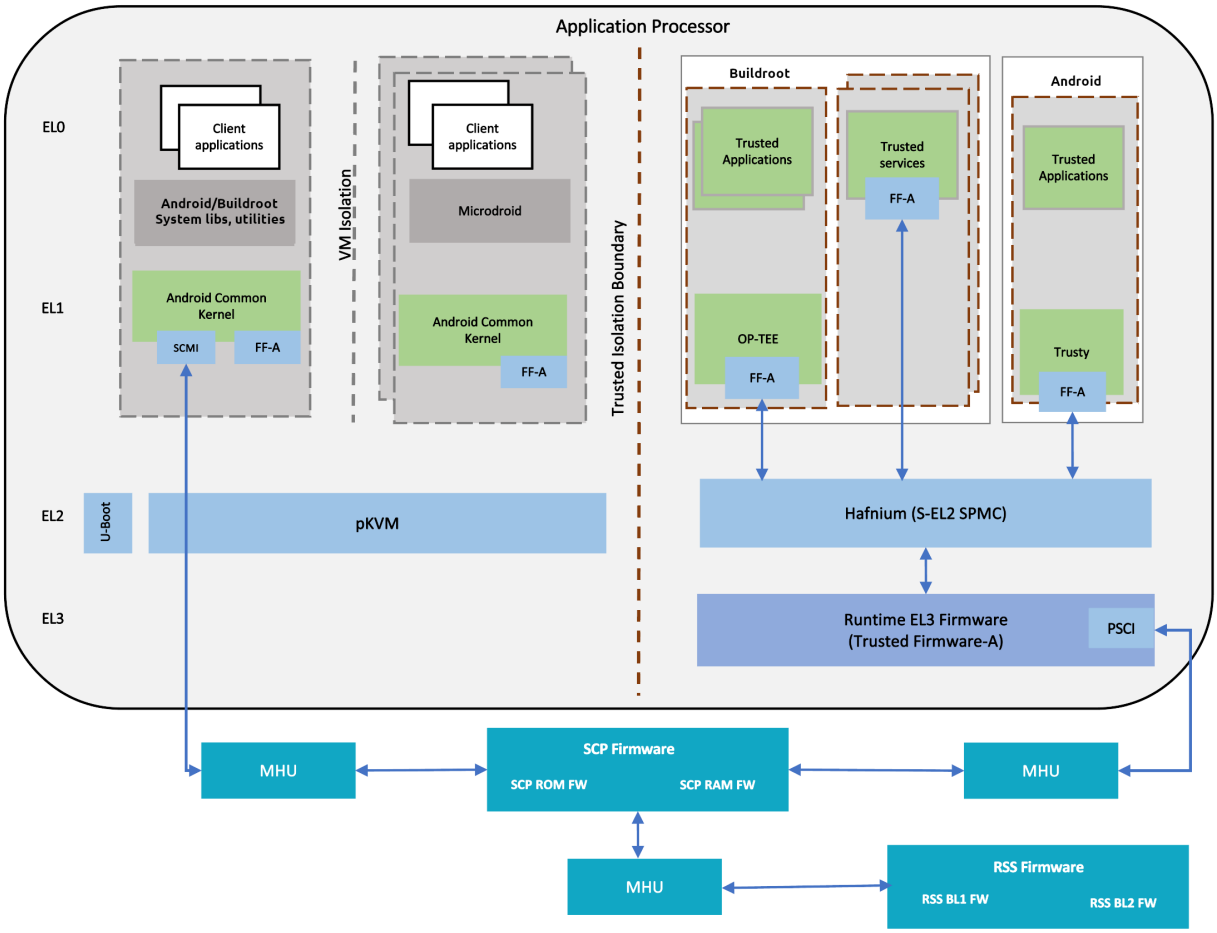
ONE

TOTAL COMPUTE: TC2-2023.10.04

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

System MMU (aka SMMU or IOMMU)

System MMU, also known as SMMUv3 or IOMMU, is the Arm IP that isolates direct memory accesses from devices (DMA), and enables devices to access non-contiguous physical memory with configurable memory attributes.

Linux has two SMMUv3 drivers:

- `CONFIG_ARM_SMMU_V3` enables the normal kernel driver that executes at `EL1`;
- `CONFIG_ARM_SMMU_V3_PKVM` enables a split driver that executes partly at `EL2`, in the pKVM hypervisor.

When pKVM is enabled (`kvm-arm.mode=protected`), the pKVM SMMU driver takes precedence over the normal driver, and protects hypervisor and guest VMs from host DMA. Host device drivers still configure DMA using the Linux DMA API, and the hypervisor installs the requested virtual-to-physical translations into the SMMU stage-2 page tables, after making sure that a compromised host is not attempting via DMA to access memory it does not own.

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 - U-boot (BL33 image)
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 and Firmware Update runs as S-EL0 Secure Partitions.

Trusty

Trusty is a secure Operating System (OS) that provides a Trusted Execution Environment (TEE) for Android. Trusty is virtualized using Hafnium at S-EL2. FF-A support is added for Total Compute. Trusty runs as a Secure Partition running in an isolated address space managed by Hafnium. The Trusty kernel runs at S-EL1 with Trusted applications running at S-EL0.

1.1.4 AP Non-Secure World Software

U-Boot

TF-A BL31 passes execution control to U-boot bootloader (BL33). U-boot in Total Compute has support for multiple image formats:

1. FitImage format: this contains the Linux kernel and Buildroot ramdisk which are authenticated and loaded in their respective positions in DRAM and execution is handed off to the kernel.
2. Android boot image: This contains the Linux kernel and Android ramdisk. If using Android Verified Boot (AVB) boot.img is loaded via virtio to DRAM, authenticated and then execution is handed off to the kernel.

Linux Kernel

Linux Kernel in Total Compute contains the subsystem-specific features that demonstrate the capabilities of Total Compute. Apart from default configuration, it enables:

1. Arm MHUv2 controller driver
2. Arm FF-A driver
3. OP-TEE driver with FF-A Transport Support
4. Arm FF-A user space interface driver
5. Trusty driver with FF-A Transport Support
6. Virtualization using pKVM

Android

Total Compute has support for Android Open-Source Project (AOSP), which contains the Android framework, Native Libraries, Android Runtime and the Hardware Abstraction Layers (HALs) for Android Operating system. The Total Compute device profile defines the required variables for Android such as partition size and product packages and has support for the below configuration of Android:

1. Software rendering: This profile has support for Android UI and boots Android to home screen. It uses SwiftShader to achieve this. Swiftshader is a CPU base implementation of the Vulkan graphics API by Google.
2. Hardware rendering: This profile also has support for Android UI and boots Android to home screen. The Mali-G720 GPU model used for rendering.

Microdroid

Microdroid is a lightweight version of Android that runs in a protected virtual machine (pVM) and is managed by Android using CrosVM.

Buildroot

A minimal rootfs that is useful for testing the bsp and boots quickly. The interface is text only and no graphics are supported.

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.

TensorFlow Lite Machine Learning

A minimal CMake wrapper project for building TensorFlow Lite applications for Total Compute targets is provided. By default, this project will build the `benchmark_model` application, which allows to profile and validate ML inference flows. However, the developer can easily adapt the project and build any application exposed by TensorFlow Lite.

1.2 Instructions: Obtaining Total Compute software deliverables

- To build the TC2 software stack, please refer to the *user guide*;
- For the list of changes and features added, please refer to the *changelog*;
- 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 in Buildroot;
   - Trusted Services in Buildroot;
   - Trusty Trusted OS in Android;
6. U-Boot – loads and verifies the fitImage for buildroot boot, containing kernel and filesystem or boot Image for Android Verified Boot, containing kernel and ramdisk;
7. Kernel – supports the following hardware features:
   - Message Handling Unit;
   - PAC/MTE/BTI features;
8. Android;
   - Supports PAC/MTE/BTI features;
9. Buildroot;
10. Debian;
11. TensorFlow Lite Machine Learning;

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 a Board Support Package (BSP) and a choice of three possible distributions including Buildroot, Debian or Android.

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 11.23.17.

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 build and run Android, the minimum requirements for the host machine can be found at https://source.android.com/setup/build/requirements. These include:

- at least 250 GB of free disk space to check out the code and an extra 150 GB to build it. If you conduct multiple builds, you need additional space;
- at least 32 GB of available RAM/swap.

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
sudo chmod 777 /var/run/docker.sock
```

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

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

1.4.3 Download the source code and build

The TC2 software stack supports the following distros:

- Buildroot (a minimal distro containing Busybox);
- Debian (based on Debian 12 Bookworm);
- Android (based on Android 13).

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/TC2-2023.10.04
```

To sync **Buildroot or 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
```

To sync **Android 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 android
repo sync -j `nproc` --fetch-submodules
```

Warning: Synchronization of the Android code from Google servers may fail due to connection problems and/or to an enforced rate limit related with the maximum number of concurrent fetching jobs. The previous commands assume that the maximum number of jobs concurrently fetching code will be a perfect match of the number of CPU cores available, which should work fine most of the time. If experiencing constant errors on consecutive fetch code attempts, please do consider deleting your entire workspace (which will ensure a clean of the support `.repo` folder containing the previously partial fetched files), by running the command `cd .. ; rm -rf <TC2_WORKSPACE>` and repeat the previous commands listed in this section to recreate the workspace (optionally, also reducing the number of jobs, for example to a maximum of 6, by adopting the following command `repo sync -j 6 --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.

Android OS build variants

Note: Android based stack takes considerable time to build, so start the build and go grab a cup of coffee!

Hardware vs Software rendering

The Android OS based build distro supports the following variants regarding the use of the GPU rendering:

TC_GPU value	Description
swr	Android display with Swiftshader (software rendering)
hwr	Mali GPU (hardware rendering based on DDK source code - please see below note)
hwr-prebuilt	Mali GPU (hardware rendering based on prebuilt binaries)

Note: GPU DDK source code is available only to licensee partners (please contact support@arm.com).

Android Verified Boot (AVB) with/without authentication

The Android images can be built with or without authentication enabled using Android Verified Boot (AVB) through the use of the `-a` option. AVB build is done in userdebug mode and takes a longer time to boot as the images are verified. This option does not influence the way the system boots, rather it adds an optional sanity check on the prerequisite images.

Build variants configuration

This section provides a quick guide on how to build the different TC build variants using the most common options.

Buildroot build

To build the Buildroot distro, please run the following commands:

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

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 (without software or GPU hardware rendering support)

To build the Debian distro, please run the following commands:

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

Android build

To build Android with Android Verified Boot (AVB) enabled, please run the next command to enable the corresponding flag in addition to any of the following Android command variants (please note that this needs to be run before running `./setup.sh`):

```
export AVB=true
```

Android can be built with or without GPU hardware rendering support by setting the `TC_GPU` environment variable accordingly, as described in the following command usage examples.

Android build with hardware rendering support based on prebuilt binaries

To build the Android distro with hardware rendering based on prebuilt binaries, please run the following commands:

```
export PLATFORM=tc2
export FILESYSTEM=android-fvp
export TC_GPU=hwr-prebuilt
export TC_TARGET_FLAVOR=fvp
cd build-scripts
./setup.sh
```

Android build with hardware rendering support based on DDK source code

To build the Android distro with hardware rendering based on DDK source code, please run the following commands:

```
export PLATFORM=tc2
export FILESYSTEM=android-fvp
export TC_GPU=hwr
export TC_TARGET_FLAVOR=fvp
export GPU_DDK_REPO=<PATH TO GPU DDK SOURCE CODE>
export GPU_DDK_VERSION="releases/r41p0_01eac0"
export LM_LICENSE_FILE=<LICENSE FILE>
export ARMLMD_LICENSE_FILE=<LICENSE FILE>
export ARMCLANG_TOOL=<PATH TO ARMCLANG TOOLCHAIN>
cd build-scripts
./setup.sh
```

Note: GPU DDK source code is available only to licensee partners (please contact support@arm.com).

Android build with software rendering support

To build the Android distro with software rendering, please run the following commands:

```
export PLATFORM=tc2
export TC_GPU=swr
export TC_TARGET_FLAVOR=fvp
export FILESYSTEM=android-fvp
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 for any of the supported distros, 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 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 ./$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

U-Boot

Based on U-Boot

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

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/<$FILESYSTEM>/tmp_build/tfa_sp/ sp.bin • <TC2_WORKSPACE>/output/<$FILESYSTEM>/tmp_build/tfa_sp/i trusted-storage.bin

Linux

The component responsible for building a 5.15 version of the Android Common kernel (ACK).

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

Trusty

Based on Trusty

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

Distributions

Buildroot Linux distro

The layer is based on the Buildroot Linux distribution. The provided distribution is based on BusyBox and built using `glibc`.

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

Debian Linux distro

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

Android

Script	<TC2_WORKSPACE>/build-scripts/build-android.sh
Files	• <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/androi • <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/ramdis • <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/system • <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/userdat • <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/boot.im (AVB only) • <TC2_WORKSPACE>/output/<$FILESYSTEM>/deploy/tc2/vbmeta (AVB only)

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:

```
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]
-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 Buildroot

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d buildroot
```

Running Debian

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

Running Android

Android general common run command

The following command is common to Android builds with AVB disabled, software or any of the hardware rendering variants. To run any of the mentioned Android variants, please run the following command:

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d android-fvp
```

Android with AVB enabled

To run Android with AVB enabled, please run the following command:

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d android-fvp -a true
```

Expected behaviour

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

- `terminal_uart_ap` used by the non-secure world components U-boot, Linux Kernel and filesystem (Buildroot/Debian/Android);
- `terminal_uart1_ap` used by the secure world components TF-A, Hafnium, Trusty 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, Trusted Services in Buildroot and Trusty in Android) then U-Boot, and finally the corresponding Linux Kernel distro.

When booting Buildroot or Debian, the model will boot the Linux kernel and present a login prompt on the `terminal_uart_ap` window. Login using the username `root` and the password `root` (password is only required for Debian). You may need to hit `Enter` for the prompt to appear.

When booting Android, the GUI window `Fast Models - Total Compute 2 DP0` shows the Android logo and on boot completion, the window will show the typical Android home screen.

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.

Validate the TensorFlow Lite ML flow

A typical Machine Learning (ML) inference flow can be validated using the TensorFlow Lite's model benchmarking application.

This application can consume any TensorFlow Lite neural network model file and run a user specified number of inferences on it, allowing to benchmark performance for the whole graph and for individual operators.

More information on the Model Benchmark tool can be found here.

Prerequisites

For this test, two files will be required:

- `benchmark_model` binary: this file is part of the TC build and is automatically built when targeting Buildroot;
- `<any model>.tflite` model: there is no requirement for a specific model file as long as it is specified in a valid `.tflite` format; for the simplicity of just running a sanity test, two models are provided with the build and are automatically integrated into the distro filesystem (being located at `/opt/arm/ml`).

Running the provided TensorFlow Lite ML model examples

The following command describes how to run the `benchmark_model` application to profile the "Mobile Object Localizer" TensorFlow Lite model, which is one of the provided TensorFlow Lite ML model examples.

Although the command arguments are expected to greatly vary according to different use cases and models, this example provides the typical command usage skeleton for most of the models.

More information on the Model Benchmark Tool and command usage examples can be found here.

To run the `benchmark_model` to profile the "Mobile Object Localizer" model, please follow the following steps:

- using `terminal_uart_ap`, login to the device/FVP model running TC and run the following commands:

```
# the following command ensures correct path location to load the provided␣
↪example ML models
cd /opt/arm/ml
benchmark_model --graph=mobile_object_localizer_v1.tflite \
        --num_threads=4 --num_runs=1 --min_secs=0.01
```

The benchmark model application will run profiling the Mobile Object Localizer model and after a few seconds, some statistics and execution info will be presented on the terminal.

Note: This test is specific to Buildroot.

Manually uploading a TensorFlow Lite ML model

There may be situations where the developer wishes to use their own TensorFlow Lite model.

This section describes the steps necessary to manually upload a model to the running TC FVP model and run it.

To the purpose of demonstrating this process, an old MobileNet Graph model version will be taken as example (the model can be downloaded from here). To run the `benchmark_model` application and profile the "MobileNet Graph" model, please proceed as described:

- start by downloading and decompressing the MobileNet graph model to your local host machine using the following command:

```
# any host path location can be used (as long it has writable permissions)
mkdir MobileNetGraphTFModel && cd MobileNetGraphTFModel
wget https://storage.googleapis.com/download.tensorflow.org/models/tflite/
↪mobilenet_v1_224_android_quant_2017_11_08.zip
unzip mobilenet_v1_224_android_quant_2017_11_08.zip
```

- upload the MobileNet Graph model to the TC FVP model using the following command:

```
# the following command assumes that the port 8022 is being used as␣
↪specified in the run_model.sh script
scp -P 8022 mobilenet_quant_v1_224.tflite root@localhost:/opt/arm/ml/
# password (if required): root
```

- once the model has been uploaded to the remote TC FVP model, the `benchmark_model` can be run as described previously in the `Running the provided TensorFlow Lite ML model examples` section.

OP-TEE

For OP-TEE, the TEE sanity test suite can be run using command `xtest` on the `terminal_uart_ap`.

Please be aware that this test suite will take some time to run all its related tests.

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

Trusted Services and Client application

For Trusted Services, please run the command `ts-service-test -sg ItsServiceTests -sg PsaCryptoApiTests -sg CryptoServicePackedcTests -sg CryptoServiceProtobufTests -sg CryptoServiceLimitTests -v` for Service API level tests, and run `ts-demo` for the demonstration of the client application.

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

Trusty

On the Android distribution, Trusty provides a Trusted Execution Environment (TEE). The functionality of Trusty IPC can be tested using the command `tipc-test -t ta2ta-ipc` with root privilege (once Android boots to prompt, run `su 0` for root access).

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

Microdroid demo

On the Android distribution, Virtualization service provides support to run Microdroid based pVM (Protected VM). For running a demo Microdroid, boot TC FVP with Android distribution. Once the Android is completely up, please run the following commands:

```
export ANDROID_PRODUCT_OUT=<TC2_WORKSPACE>/src/android/out/target/product/tc_fvp/
./run-scripts/tc2/run_microdroid_demo.sh
```

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

Kernel Selftest

Tests are located at `/usr/bin/selftest` on the device.

To run all the tests in one go, use `./run_kselftest.sh` script. Tests can also be run individually.

```
./run_kselftest.sh --summary
```

Warning: KSM driver is not a part of the TC2 kernel. Hence, one of the MTE Kselftests will fail for the `check_ksm_options` test.

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

MPAM

The hardware and the software requirements required for the MPAM feature can be verified by running the command `testing_mpam.sh` on `terminal_uart_ap` (this script is located inside the `/bin` folder, which is part of the default `$PATH` environment variable, allowing this command to be executed from any location in the device filesystem).

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

MPMM

Note: The following two tests require to execute the FVP-model enforcing the additional load of the `ScalableVectorExtension.so` plugin (which is provided and part of your FVP bundle). The following command demonstrates the typical command skeleton required to execute the fvp-model in this situation:

```
./run-scripts/tc2/run_model.sh -m <fvp-model binary path> -d buildroot \
                -- \
                --plugin <fvp-model plugin path/ScalableVectorExtension.so>
```

The functionality of the MPMM module in the SCP firmware can be leveraged to:

- set the proper gear for each core based on the workload. This functionality can be verified by checking the `INFO` level SCP logs while executing the `vector_workload` test application on the `terminal_uart_ap` window as follows:

  ```
  vector_workload
  ```

- enforce the maximum clock frequency for a group of cores of the same type, based on the current gear set for each core in that group. This functionality can be exercised by running the provided shell script `test_mpmm.sh` which will run `vector_workload` on the different cores. This test ensures that the maximum clock frequency for a group of cores of the same type does not exceed the values set in Perf Constraint Lookup Table (PCT) of the MPMM module in the SCP firmware.

  To run this test, please run the following command in the `terminal_uart_ap` window:

  ```
  test_mpmm.sh fvp
  ```

Note: These tests are specific to Buildroot only. An example of the expected test result for the second test is illustrated in the related *Total Compute Platform Expected Test Results* document section.

BTI

On the `terminal_uart_ap` run:

```
su
cd /data/nativetest64/bti-unit-tests/
./bti-unit-tests
```

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

MTE

On the `terminal_uart_ap` run:

```
su
cd /data/nativetest64/mte-unit-tests/
./mte-unit-tests
```

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

PAUTH

On the `terminal_uart_ap` run:

```
su
cd /data/nativetest64/pauth-unit-tests/
./pauth-unit-tests
```

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

EAS with LISA

This test requires Lisa to be installed. Please refer to the LISA documentation to get more information about the requirements, dependencies and installation process of LISA on your system.

To setup Lisa, please run the following commands:

```
git clone --depth=1 --branch=v3.1.0 https://github.com/ARM-software/lisa.git
cd lisa
sudo ./install_base.sh --install-all
```

The following commands should be run each time LISA is run:

```
source init_env
export TC_WORKSPACE=<TC2_WORKSPACE>
export FILESYSTEM=buildroot
```

For FVP with buildroot, boot the FVP model to buildroot as you normally would, making sure user networking is enabled:

```
exekall run lisa.tests.scheduler.eas_behaviour --conf <path to target_conf_linux.yml>
```

The following excerpt illustrates the contents of the `target_conf_buildroot.yml` file:

```
target-conf:
  kind: linux
  name: tc
```

(continues on next page)

(continued from previous page)

```
  host: localhost
  port: 8022
  username: root
  password: ""
  strict-host-check: false

  kernel:
    src: ${TC_WORKSPACE}/output/${FILESYSTEM}/tmp_build/linux

    modules:
      make-variables:
        CC: clang
      build-env: alpine

  wait-boot:
    enable: false

  devlib:
    file-xfer: scp
    max-async: 1
```

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

pKVM SMMUv3 driver support validation

The SMMUv3 driver support can be validated by checking the bootlog messages or by running the following presented command. This section describes and educates what output to expect for both situations where the driver is loaded and enabled, or when it fails or is disabled.

On the `terminal_uart_ap` run:

```
realpath /sys/bus/platform/devices/3f000000.smmu_700/driver
```

When the **pKVM driver is loaded and enabled with success**, the previous command should report an output similar to the following one:

```
$ realpath /sys/bus/platform/devices/3f000000.smmu_700/driver
/sys/bus/platform/drivers/kvm-arm-smmu-v3
```

If the **pKVM driver fails to load or is disabled**, the previous command should report an output similar to the following one:

```
$ realpath /sys/bus/platform/devices/3f000000.smmu_700/driver
/sys/bus/platform/drivers/arm-smmu-v3
```

More information about the pKVM driver loading, initialisation phase and it being used by a device driver can be checked during the bootlog messages or by running the command `dmesg`, which should contain entries similar to the following:

```
(...)
[    0.035500][    T1] iommu: Default domain type: Translated
[    0.035506][    T1] iommu: DMA domain TLB invalidation policy: strict mode
(...)
[    0.073258][    T1] kvm [1]: IPA Size Limit: 40 bits
[    0.091014][    T1] kvm-arm-smmu-v3 3f000000.smmu_700: ias 40-bit, oas 40-bit␣
↪(features 0x0000dfef)
[    0.091426][    T1] kvm-arm-smmu-v3 3f000000.smmu_700: allocated 65536 entries for␣
↪cmdq
[    0.091435][    T1] kvm-arm-smmu-v3 3f000000.smmu_700: 2-level strtab only covers 23/
↪32 bits of SID
[    0.092569][    T9] Freeing initrd memory: 1328K
[    0.096695][    T1] kvm [1]: GICv4 support disabled
[    0.096702][    T1] kvm [1]: GICv3: no GICV resource entry
[    0.096709][    T1] kvm [1]: disabling GICv2 emulation
[    0.096731][    T1] kvm [1]: GIC system register CPU interface enabled
[    0.096788][    T1] kvm [1]: vgic interrupt IRQ9
[    0.096861][    T1] kvm [1]: Protected nVHE mode initialized successfully
(...)
[    0.151372][    T7] komeda 2cc00000.display: Adding to iommu group 0
(...)
[   34.986406][    T7] mali 2d000000.gpu: Adding to iommu group 1
(...)
```

Considering the previous output excerpt, the last line confirms that the system is using pKVM instead of the classic KVM driver.

Note: This test is applicable to all TC build distro variants.

CPU hardware capabilities

The Buildroot build variant provides a script that allows to validate the advertisement for the `FEAT_AFP`, `FEAT_ECV` and `FEAT_WFxT` CPU hardware capabilities.

On the `terminal_uart_ap` run:

```
test_feats_arch.sh
```

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

1.4.8 Debugging on Arm Development Studio

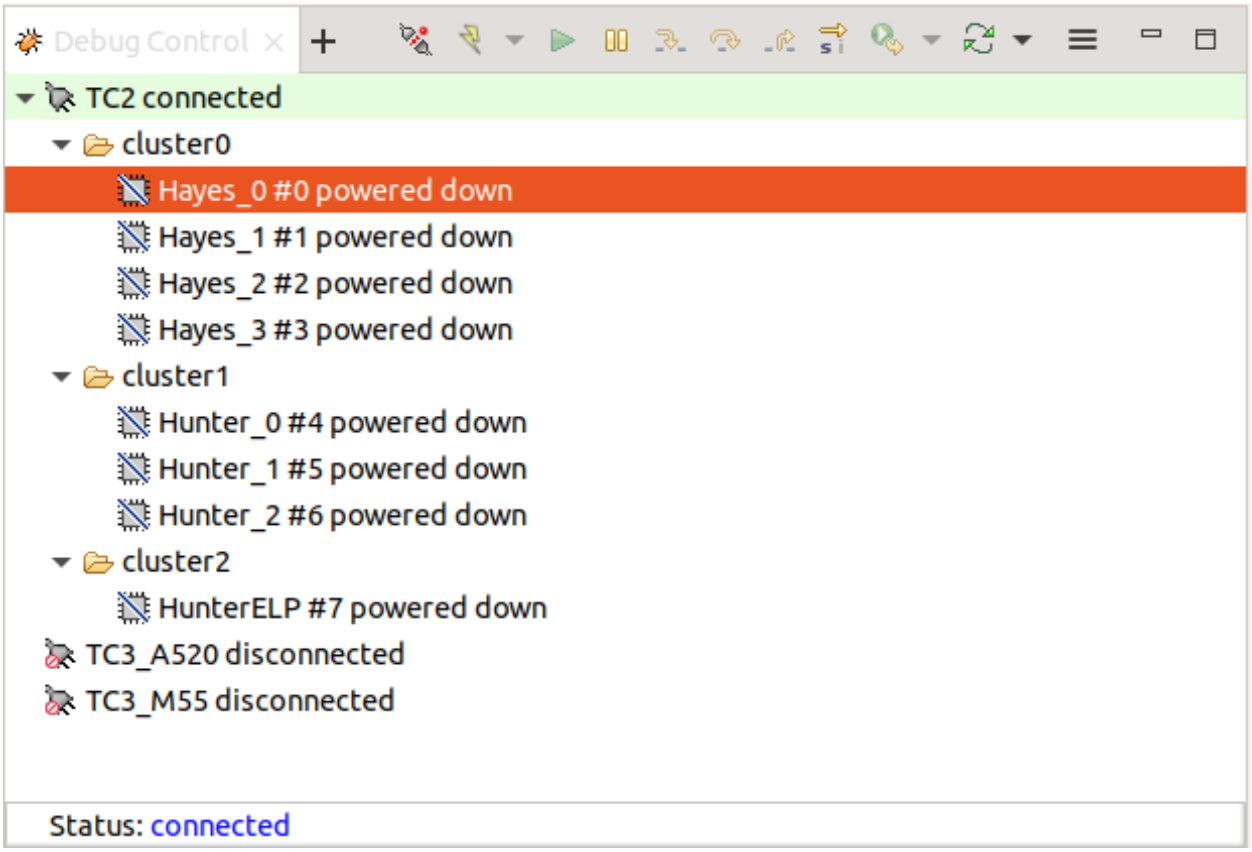
This section describes the steps to debug the TC software stack using Arm Development Studio.

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 <distro> -- -I
```

3. Select the target `Arm FVP -> TC2 -> Bare Metal Debug -> Hayesx4/Hunterx3/HunterELP SMP`
4. After connection, use 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

Firmware Update

Currently, the firmware update functionality is only supported with the buildroot distro.

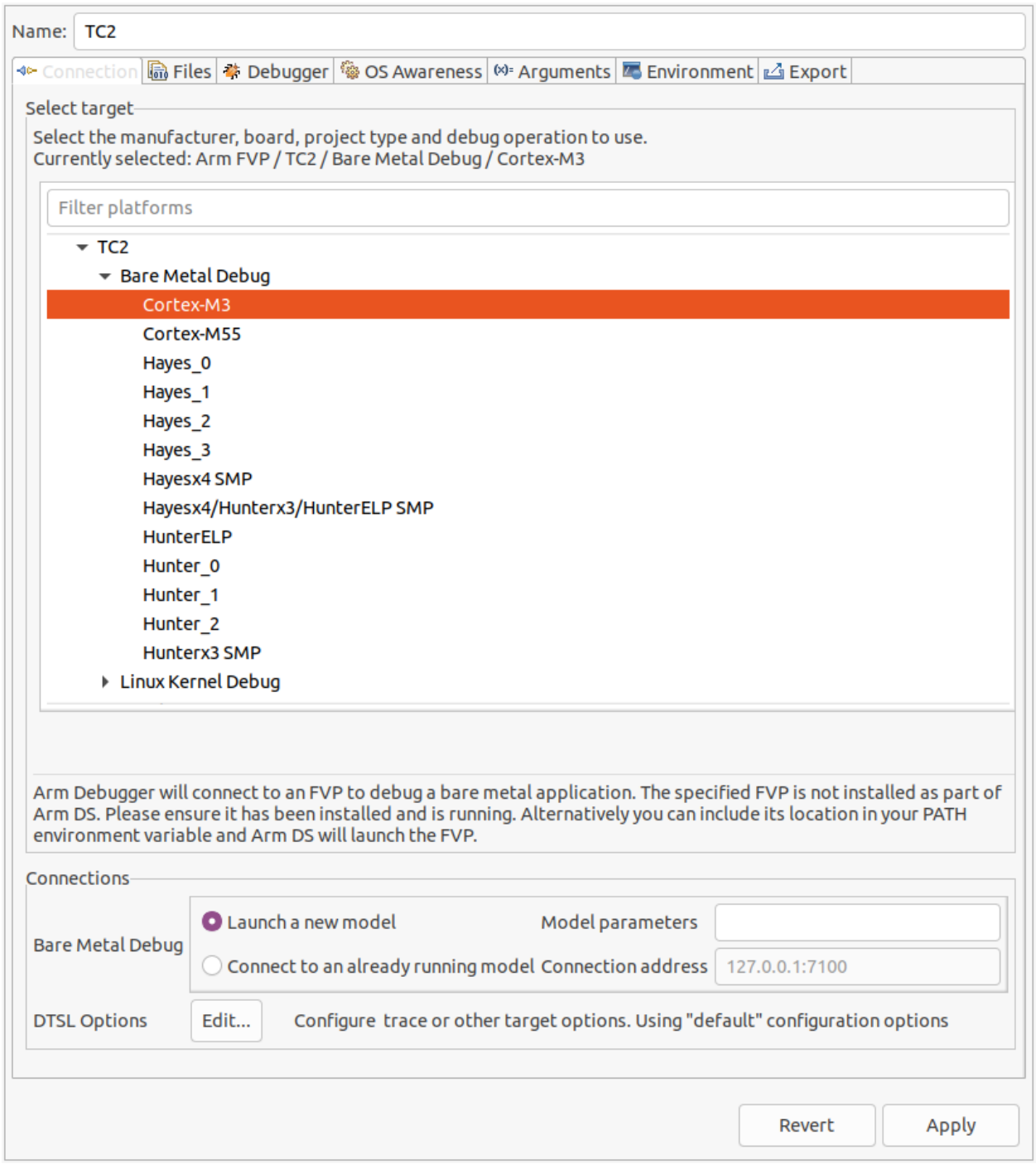
Creating Capsule

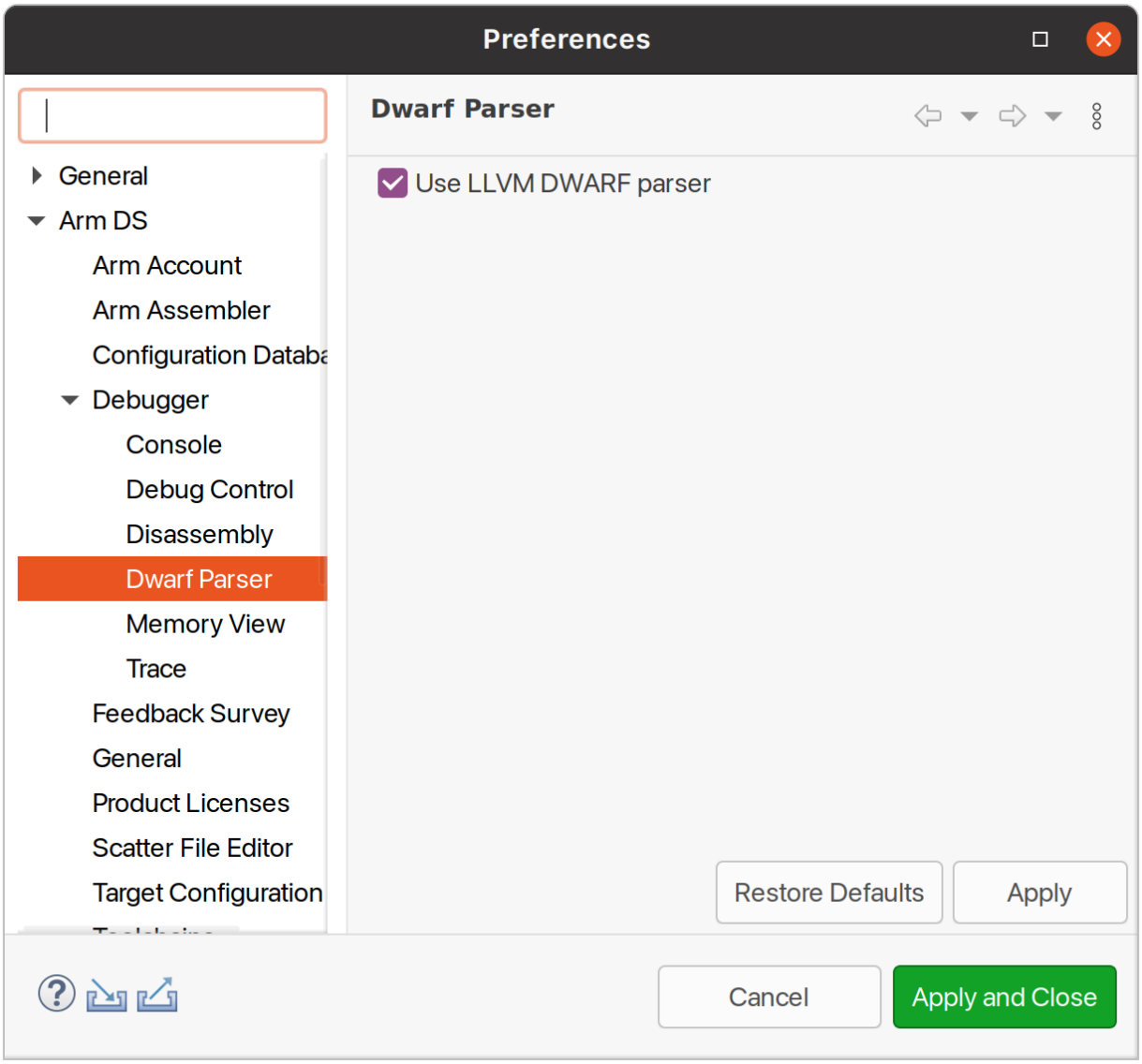
Firmware Update in the total compute platform uses the capsule update mechanism. Hence, the Firmware Image Package (FIP) binary has to be converted to a capsule. This can be done with `GenerateCapsule` which is present in `BaseTools/BinWrappers/PosixLike` of the edk2 project.

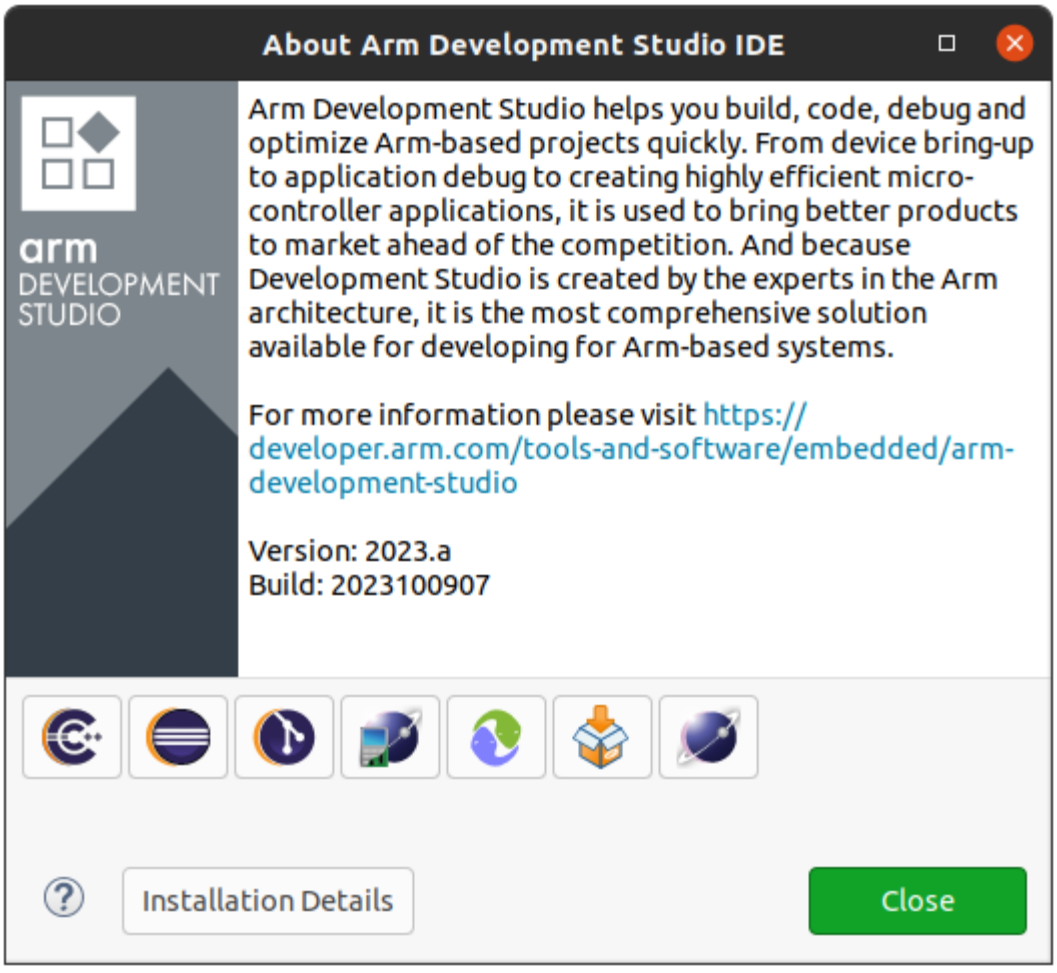
To generate the capsule from the fip binary, run the following command:

```
./GenerateCapsule -e -o efi_capsule \
            --fw-version 1 \
            --lsv 0 \
            --guid 0d5c011f-0776-5b38-8e81-36fbdf6743e2 \
```

(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
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
Use LLVM DWARF parser
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)

```
            --update-image-index 0 \
            --verbose fip-tc.bin
```

Command argument's explanation:

- `fip-tc.bin` is the input fip file that has the firmware binaries of the total compute platform;
- `efi_capsule` is the name of capsule to be generated;
- `0d5c011f-0776-5b38-8e81-36fbdf6743e2` is the image type UUID for the FIP image.

Loading Capsule

The capsule generated using the above steps has to be loaded into memory during the execution of the model by providing the below FVP arguments:

```
--data board.dram=<location of capsule>/efi_capsule@0x2000000
```

This will load the capsule to be updated at address `0x82000000`.

The final command to run the model for buildroot should look like the following:

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d buildroot \
            -- \
            --data board.dram=<location of capsule>/efi_capsule@0x2000000
```

Updating Firmware

During the normal boot of the platform, stop at the U-Boot prompt and execute the following command:

```
TOTAL_COMPUTE# efidebug capsule update -v 0x82000000
```

This will update the firmware. After it is completed, reboot the platform using the FVP GUI.

AutoFDO in Android

Feedback Directed Optimization (FDO), also known as Profile Guided Optimization (PGO), uses the profile of a program's execution to guide the optimizations performed by the compiler.

More information about the AutoFDO process in ARM can be found here.

Prerequisites

To make use of this feature, the following requisites should be observed:

- the application must be compiled to include sufficient debug information to map instructions back to source lines. For `clang/llvm`, this translates into adding the `-fdebug-info-for-profiling` and `-gline-tables-only` compiler options;
- `simpleperf` will identify the active program or library using the build identifier stored in the elf file. This requires the use of the following compiler flag `-Wl,--build-id=sha1` to be added during link time.
- download Android NDK from Android NDK downloads page and extract its contents.

The following example demonstrates how to compile a sample C program named `program.c` using `clang` from Android NDK:

```
<ndk-path>/toolchains/llvm/prebuilt/linux-x86_64/bin/clang --target=aarch64-linux-
↪android31 --sysroot=<ndk-path>/toolchains/llvm/prebuilt/linux-x86_64/sysroot -fdebug-
↪info-for-profiling -gline-tables-only -Wl,--build-id=sha1 -Wl,--no-rosegment program.c␣
↪-o program
```

Steps to use AutoFDO

The following steps describe how to upload the resulting `program` binary object to the fvp-model, how to generate and convert the execution trace into source level profiles, and how to download and reuse that to optimize the next compiler builds:

1. connect to the fvp-model running instance;

   Please refer to the *ADB - Connect to the running FVP-model instance* section for more info how to do this.

2. upload the previous resulting `program` binary object to the remote `/vendor/bin` path location;

   Please refer to the *ADB - Upload a file* section for more info how to do this.

3. using the `terminal_uart_ap` window, navigate into `/storage/self` path location and elevate your privilege level to `root` (required and crucial for next steps). This can be achieved by running the following commands on the specified terminal window:

   ```
   cd /storage/self
   su
   chmod a+x /vendor/bin/program
   ```

4. record the execution trace of the program;

   The `simpleperf` application in Android is used to record the execution trace of the application. This trace will be captured by collecting the `cs_etm` event from `simpleperf` and will be stored in a `perf.data` file.

   The following command demonstrates how to make use of the `simpleperf` application to record the execution trace of the `program` application (this command is intended to be run on the fvp-model via the `terminal_uart_ap` window):

   ```
   simpleperf record -e cs-etm program
   ```

   More info on the `simpleperf` tool can be found here.

5. convert the execution trace to instruction samples with branch histories;

   The execution trace can be converted to an instruction profile using the `simpleperf` application. The following `simpleperf inject` command will decode the execution trace and generate branch histories in text format accepted by AutoFDO (this command is intended to be run on the fvp-model via the `terminal_uart_ap` window):

   ```
   simpleperf inject -i perf.data -o inj.data --output autofdo --binary program
   ```

6. convert the instruction samples to source level profiles;

   The AutoFDO tool is used to convert the instruction profiles to source profiles for the `GCC` and `clang/llvm` compilers. It can be installed in the host machine with the following command:

```
sudo apt-get install autofdo
```

The conversion of the instruction samples to source level profiles requires to pull the instruction profile (generated in the previous step and saved as `inj.data` file), from the model to the host machine using the `adb` command (please refer to the *ADB - Download a file* section for more info how to do this).

The instruction samples produced by `simpleperf inject` will be passed to the AutoFDO tool to generate source level profiles for the compiler. The following line demonstrates the usage command for `clang/llvm` (this command is intended to be run on the host machine):

```
create_llvm_prof --binary program --profile inj.data --profiler text --out␣
↪program.llvmprof --format text
```

7. use the source level profile with the compiler;

   The profile produced by the above steps can now be provided to the compiler to optimize the next build of the `program` application. For `clang`, use the `-fprofile-sample-use` compiler option as follows (this command is intended to be run on the host machine):

```
<ndk-path>/toolchains/llvm/prebuilt/linux-x86_64/bin/clang --target=aarch64-
↪linux-android31 --sysroot=<ndk-path>/toolchains/llvm/prebuilt/linux-x86_
↪64/sysroot -O2 -fprofile-sample-use=program.llvmprof -o program program.c
```

ADB connection on Android

This section applies to Android distros and describes the steps required to use ADB protocol to perform the following actions (always considering a remote running FVP-model Android instance):

- connect to a running fvp-model instance;
- upload a file;
- download a file;
- execute a command via ADB shell.

Connect to the running FVP-model instance

1. run the fvp-model and wait for the instance to fully boot up (this may take a considerable amount of time depending on the distro under test and the host hardware specification);
2. once the Android distro boot completes (and the `Fast Models - Total Compute 2 DP0` window shows the complete Android home screen), run the following commands on a new host terminal session to connect to the fvp-model running instance via the `adb` protocol:

```
adb connect 127.0.0.1:5555
adb devices
```

The following excerpt capture demonstrates the execution and expected output from the previous commands:

```
# adb connect 127.0.0.1:5555
* daemon not running; starting now at tcp:5037
* daemon started successfully
connected to 127.0.0.1:5555
```

(continues on next page)

(continued from previous page)

```
# adb devices
List of devices attached
127.0.0.1:5555  offline
```

Note: If the previous command fails to connect, please wait a few more minutes and retry. Due to the indeterministic services boot flow nature, this may circumvent situations where the fvp-model Android instance takes a bit longer to start all the required services and correctly allow communications to happen.

Warning: If running more than one FVP-model on the same host, each instance will get a different ADB port assigned. The assigned ADB port is mentioned during the FVP-model start up phase. Please ensure you are using the correct assigned/mentioned ADB port and adapt the commands mentioned in this entire section as needed (i.e. replacing default port `5555` or `<fvp adb port>` mentions with the correct port being used).

Upload a file

1. connect or ensure that an ADB connection to the fvp-model is established;
2. run the following command to upload a local file to the remote fvp-model Android running instance:

```
adb -s <fvp adb port> push <local host location for original file> <remote␣
↪absolute path location to save file>
```

Note: It may happen that the ADB connection is lost between the connection moment and the moment that the previous command is run. If that happens, please repeat the connection step and the previous command.

Download a file

1. connect or ensure that an ADB connection to the fvp-model is established;
2. run the following command to download a remote file to your local host system:

```
adb -s <fvp adb port> pull <remote absolute path location for original file>
↪<local host location where to save file>
```

Note: It may happen that the ADB connection is lost between the connection moment and the moment that the previous command is run. If that happens, please repeat the connection step and the previous command.

Execute a remote command

```
adb -s <fvp adb port> shell <command>
```

Example:

```
adb -s <fvp adb port> shell ls -la
```

Note: It may happen that the ADB connection is lost between the connection moment and the moment that the previous command is run. If that happens, please repeat the connection step and the previous command.

Set up TAP interface for Android ADB

This section applies to Android and details the steps required to set up the tap interface on the host for model networking for ADB.

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)
   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. download and install the Android SDK from here or, alternatively, install the `adb` tool package as follows:

```
sudo apt-get install adb
```

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

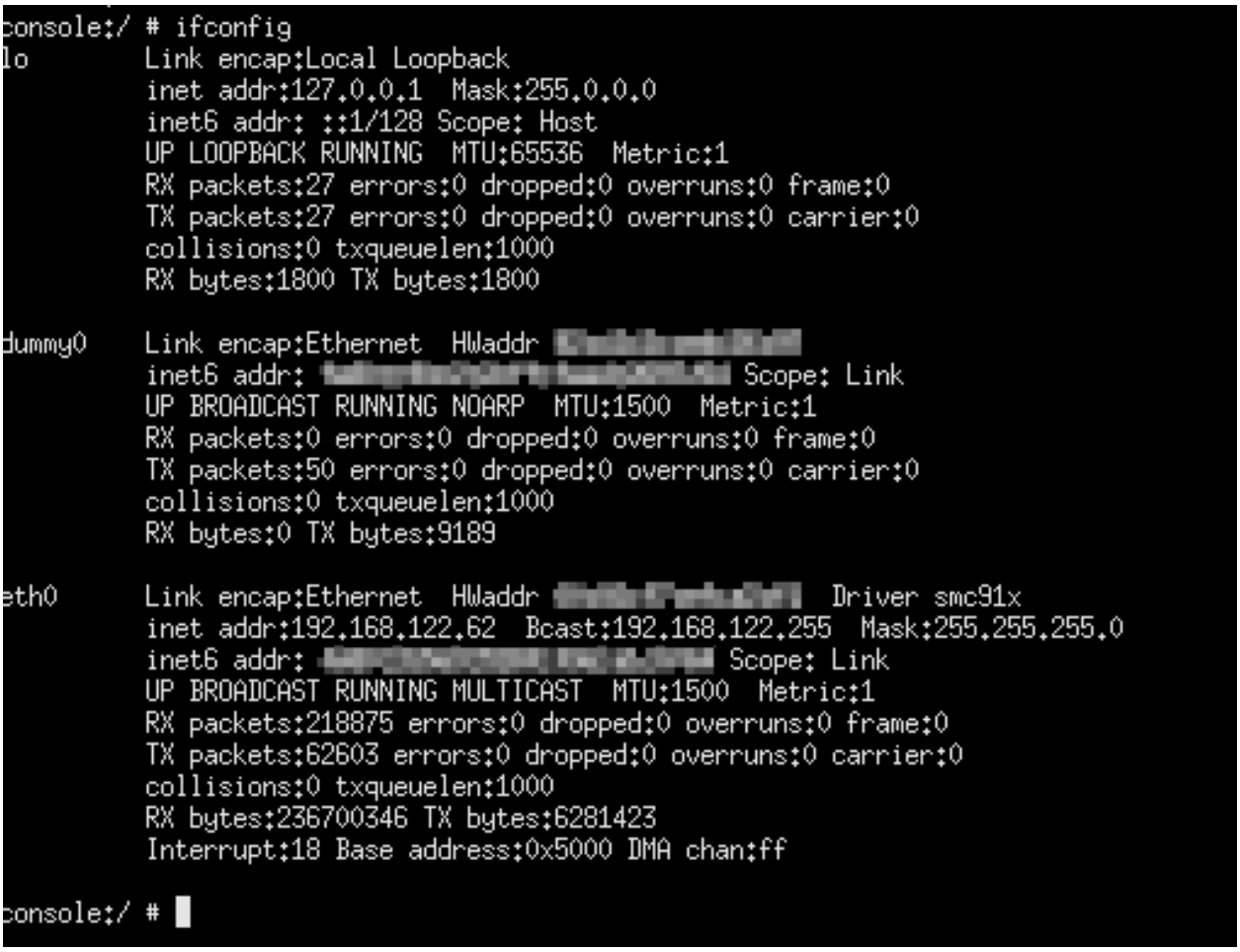
```
./run-scripts/tc2/run_model.sh -m <model binary path> -d android-fvp -t
↪"tap0"
```

Before proceeding, please allow Android FVP model to fully boot and the Android home screen display to be visible on the `Fast Models - Total Compute 2 DP0` window.

Note: Running and booting the Android FVP model will take considerable time, potentially taking easily 2-3+ hours depending on your host system hardware specification. Please grab a coffee and relax.

---

5. once the Android FVP model boots, the Android instance should get an IP address similar to `192.168.122.62`, as illustrated in the next figure:



6. validate the connection between the host `tap0` interface and the Android 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.

7. at this stage, you should also be able to establish an ADB connection and upload/download files as described in section *ADB connection on Android*.

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, 11.23.17)
================================================================================
Component is of type "MMU_700"
Version is "11.23.17"
#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 <distro> \
        -- \
        --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 <distro> \
        -- \
        --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
```

The following output excerpt illustrates an example of the trace information captured for the DPU (`streamid=0x00000100`) and GPU (`streamid=0x00000200`):

```
(...)
start_ptw_read: trans_id=0x000000000000020f streamid=0x00000100␣
↪substreamid=0xffffffff ttb_grain_stage_and_level=0x00000201 pa_
↪address=0x0000008083fdc018 input_address=0x00000000ffe00000 ssd_ns=ssd_ns␣
↪ns=bus-ns desckind=el2_or_st2_aarch64 inner_cache=rawaWB outer_cache=rawaWB␣
↪aprot=DNP adomain=ish mpam_pmg_and_partid=0x00000000 ssd=ns pas=ns␣
↪mecid=0xffffffff
verbose_commentary: output="Performing a Table Walk read as:-"
verbose_commentary: output="    trans_id:527-st2-final-l1-aa64-ttb0-vmid:0-ns-
↪sid:256"
verbose_commentary: output="to ns-0x0000008083fdc018-PND-u0x53000009-
↪m0xffffffff-ish-osh-rawaC-rawaC of size 8B"
verbose_commentary: output="Table Walk finished:-"
verbose_commentary: output="    trans_id:527-st2-final-l1-aa64-ttb0-vmid:0-ns-
↪sid:256"
verbose_commentary: output="got:-"
verbose_commentary: output="    0x0000008083fdc018: 0x0000008085c31003"
ptw_read: trans_id=0x000000000000020f streamid=0x00000100␣
↪substreamid=0xffffffff ttb_grain_stage_and_level=0x00000201 pa_
↪address=0x0000008083fdc018 input_address=0x00000000ffe00000 ssd_ns=ssd_ns␣
↪ns=bus-ns desckind=el2_or_st2_aarch64 inner_cache=rawaWB outer_cache=rawaWB␣
↪aprot=DNP adomain=ish abort=ok data=0x0000008085c31003 ssd=ns pas=ns␣
↪mecid=0xffffffff
ptw_read_st2_table_descriptor: trans_id=0x000000000000020f streamid=0x00000100␣
↪substreamid=0xffffffff ttb_grain_stage_and_level=0x00000201 pa_
↪address=0x0000008083fdc018 input_address=0x00000000ffe00000 ssd_ns=ssd_ns␣
↪ns=bus-ns desckind=el2_or_st2_aarch64 APTable=aptable_no_effect XNTable=N␣
↪PXNTable=N TableAddress=0x0000008085c31000 ssd=ns pas=ns mecid=0xffffffff
(...)
start_ptw_read: trans_id=0x000000000000033b streamid=0x00000200␣
↪substreamid=0xffffffff ttb_grain_stage_and_level=0x00000201 pa_
↪address=0x00000080872a7010 input_address=0x00000080844db000 ssd_ns=ssd_ns␣
↪ns=bus-ns desckind=el2_or_st2_aarch64 inner_cache=rawaWB outer_cache=rawaWB␣
↪aprot=DNP adomain=ish mpam_pmg_and_partid=0x00000000 ssd=ns pas=ns␣
↪mecid=0xffffffff
verbose_commentary: output="Performing a Table Walk read as:-"
verbose_commentary: output="    trans_id:827-st2-final-l1-aa64-ttb0-vmid:1-ns-
↪sid:512"
verbose_commentary: output="to ns-0x00000080872a7010-PND-u0x53000109-
↪m0xffffffff-ish-osh-rawaC-rawaC of size 8B"
verbose_commentary: output="Table Walk finished:-"
verbose_commentary: output="    trans_id:827-st2-final-l1-aa64-ttb0-vmid:1-ns-
↪sid:512"
verbose_commentary: output="got:-"
verbose_commentary: output="    0x00000080872a7010: 0x000000808a52d003"
ptw_read: trans_id=0x000000000000033b streamid=0x00000200␣
↪substreamid=0xffffffff ttb_grain_stage_and_level=0x00000201 pa_
↪address=0x00000080872a7010 input_address=0x00000080844db000 ssd_ns=ssd_ns␣
↪ns=bus-ns desckind=el2_or_st2_aarch64 inner_cache=rawaWB outer_cache=rawaWB␣
↪aprot=DNP adomain=ish abort=ok data=0x000000808a52d003 ssd=ns pas=ns␣
```

(continues on next page)

(continued from previous page)

```
↪mecid=0xffffffff
ptw_read_st2_table_descriptor: trans_id=0x000000000000033b streamid=0x00000200␣
↪substreamid=0xffffffff ttb_grain_stage_and_level=0x00000201 pa_
↪address=0x00000080872a7010 input_address=0x00000080844db000 ssd_ns=ssd_ns␣
↪ns=bus-ns desckind=el2_or_st2_aarch64 APTable=aptable_no_effect XNTable=N␣
↪PXNTable=N TableAddress=0x000000808a52d000 ssd=ns pas=ns mecid=0xffffffff
(...)
```


1.5 Expected test results

Contents

1.5.1 OP-TEE unit tests

```
# xtest
Run test suite with level=0

TEE test application started over default TEE instance
##########################################################
#
# regression
#
##########################################################
```

(continues on next page)

(continued from previous page)

```
* regression_1001 Core self tests
 - 1001 -   skip test, pseudo TA not found
  regression_1001 OK

* regression_1002 PTA parameters
 - 1002 -   skip test, pseudo TA not found
  regression_1002 OK

(...output truncated...)

regression_8101 OK
regression_8102 OK
regression_8103 OK
+-----------------------------------------------------
26197 subtests of which 0 failed
104 test cases of which 0 failed
0 test cases were skipped
TEE test application done!
#
```

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.

1.5.2 Trusted Services and Client application unit tests

Expected command output for the Trusted Services:

```
# ts-service-test -sg ItsServiceTests -sg PsaCryptoApiTests -sg␣
↪CryptoServicePackedcTests -sg CryptoServiceProtobufTests -sg CryptoServiceLimitTests -v
TEST(ItsServiceTests, storeNewItem) - 3903 ms
TEST(CryptoServicePackedcTests, generateRandomNumbers) - 8063 ms
TEST(CryptoServicePackedcTests, asymEncryptDecryptWithSalt) - 46995 ms
TEST(CryptoServicePackedcTests, asymEncryptDecrypt) - 11187 ms
TEST(CryptoServicePackedcTests, signAndVerifyEat) - 36934 ms
TEST(CryptoServicePackedcTests, signAndVerifyMessage) - 37118 ms
TEST(CryptoServicePackedcTests, signAndVerifyHash) - 37121 ms
TEST(CryptoServicePackedcTests, exportAndImportKeyPair) - 5506 ms
TEST(CryptoServicePackedcTests, exportPublicKey) - 7416 ms
TEST(CryptoServicePackedcTests, purgeKey) - 4631 ms
TEST(CryptoServicePackedcTests, copyKey) - 12366 ms
TEST(CryptoServicePackedcTests, generatePersistentKeys) - 8316 ms
TEST(CryptoServicePackedcTests, generateVolatileKeys) - 7886 ms
TEST(CryptoServiceProtobufTests, generateRandomNumbers) - 5785 ms
TEST(CryptoServiceProtobufTests, asymEncryptDecryptWithSalt) - 59963 ms
TEST(CryptoServiceProtobufTests, asymEncryptDecrypt) - 15982 ms
TEST(CryptoServiceProtobufTests, signAndVerifyMessage) - 37117 ms
TEST(CryptoServiceProtobufTests, signAndVerifyHash) - 37177 ms
TEST(CryptoServiceProtobufTests, exportAndImportKeyPair) - 5562 ms
TEST(CryptoServiceProtobufTests, exportPublicKey) - 7467 ms
```

(continues on next page)

(continued from previous page)

```
TEST(CryptoServiceProtobufTests, generatePersistentKeys) - 8378 ms
TEST(CryptoServiceProtobufTests, generateVolatileKeys) - 7896 ms
TEST(CryptoServiceLimitTests, volatileRsaKeyPairLimit) - 814715 ms
TEST(CryptoServiceLimitTests, volatileEccKeyPairLimit) - 197333 ms

OK (43 tests, 24 ran, 206 checks, 0 ignored, 19 filtered out, 1425193 ms)

#
```

Expected command output for the Client application:

```
# ts-demo

Demonstrates use of trusted services from an application
------------------------------------------------------------
A client requests a set of crypto operations performed by
the Crypto service.  Key storage for persistent keys is
provided by the Secure Storage service via the ITS client.

Generating random bytes length: 1
            Operation successful
            Random bytes:
                    2B
Generating random bytes length: 7
            Operation successful
            Random bytes:
                    68 CF 0C 5D 87 C7 11
Generating random bytes length: 128
            Operation successful
            Random bytes:
                    BF C6 85 27 81 02 5F 83
                    60 97 E9 2C A6 30 8E F7
                    C6 81 44 CB 26 32 8D F5
                    62 BA 0F DE B8 2C 69 E2
                    DD C0 FF A0 04 E2 D0 C0
                    DC EA 11 CE DD 7E 33 87
                    62 07 89 02 00 68 FC 24
                    AD D2 E4 86 40 3F 6E 65
                    83 46 33 9A F8 84 14 3B
                    72 11 8D 63 59 6F 69 96
                    70 D2 83 8D 60 6D 9F A2
                    B3 54 F6 3E 5E B3 FE 07
                    C9 51 F1 6A F5 B0 0E AA
                    08 B3 AE F5 06 73 6C 8B
                    95 73 B2 FF 72 C6 CF 84
                    12 7A 7A 1F 07 F2 58 71
Generating ECC signing key
            Operation successful
Signing message: "The quick brown fox" using key: 256
            Operation successful
            Signature bytes:
                    F9 F7 0E D0 4A B2 77 DF
```

(continues on next page)

(continued from previous page)

```
                    67 40 F5 36 4D 92 38 A3
                    13 5B 04 A0 6C BD 84 40
                    03 E2 43 EE BF 6F C6 C4
                    5B 5D A4 21 D9 EB 17 86
                    B9 71 0D C9 84 0C FE 55
                    71 8E 5C F7 D4 7D EB 04
                    9B 5A 11 D7 46 96 BD A6
Verify signature using original message: "The quick brown fox"
            Operation successful
Verify signature using modified message: "!he quick brown fox"
            Successfully detected modified message
Signing message: "jumps over the lazy dog" using key: 256
            Operation successful
            Signature bytes:
                    45 40 14 E3 39 0C 3B 8A
                    5F 05 C8 0C E0 B6 A6 D2
                    8B 5E E3 76 49 DD F1 9E
                    50 A0 77 6F 1B FA FF C8
                    38 66 6A 2D 40 B1 79 9C
                    43 BE 59 F4 48 45 A2 0E
                    D0 17 3F 1F D3 D7 C0 84
                    65 AC 9B 8A FB 6E B6 B6
Verify signature using original message: "jumps over the lazy dog"
            Operation successful
Verify signature using modified message: "!umps over the lazy dog"
            Successfully detected modified message
Generating RSA encryption key
            Operation successful
Encrypting message: "Top secret" using RSA key: 257
            Operation successful
            Encrypted message:
                    42 B6 53 D8 A3 03 BB 64
                    66 C0 31 A5 42 2C F8 F3
                    B8 E3 9C 58 42 7C 2C E0
                    19 43 F6 02 EB 60 6A DC
Decrypting message using RSA key: 257
            Operation successful
            Decrypted message: "Top secret"
Exporting public key: 256
            Operation successful
            Public key bytes:
                    04 D0 9A AF 76 18 9B 3B
                    08 38 65 BA 5F 81 B0 85
                    6A 39 42 19 5F 0D 17 86
                    CD 7E 2A E6 A4 CC A2 E4
                    B3 78 89 76 F6 CA 02 12
                    CB 07 2B AB CF 03 59 B3
                    34 8D 5D 0F 31 53 E0 68
                    9D 25 E2 AF 2E 0C 2C BE
                    51
Destroying signing key: 256
            Operation successful
```

(continues on next page)

(continued from previous page)

```
Destroying encryption key: 257
            Operation successful
#
```

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.

1.5.3 Trusty unit tests

```
console:/ # tipc-test -t ta2ta-ipc
ta2ta_ipc_test:
ipc-unittest-main: 2556: first_free_handle_index: 3
ipc-unittest-main: 2540: retry ret 0, event handle 1000, event 0x1
ipc-unittest-main: 2543: nested ret -13, event handle 1000, event 0x1
[ RUN      ] ipc.wait_negative
[       OK ] ipc.wait_negative
[ RUN      ] ipc.close_handle_negative
[       OK ] ipc.close_handle_negative
[ RUN      ] ipc.set_cookie_negative
[       OK ] ipc.set_cookie_negative
[ RUN      ] ipc.port_create_negative
[       OK ] ipc.port_create_negative
[ RUN      ] ipc.port_create
[       OK ] ipc.port_create
[ RUN      ] ipc.connect_negative
[       OK ] ipc.connect_negative
[ RUN      ] ipc.connect_close
[       OK ] ipc.connect_close
[ RUN      ] ipc.connect_access
[       OK ] ipc.connect_access
[ RUN      ] ipc.accept_negative
[       OK ] ipc.accept_negative
[ DISABLED ] ipc.DISABLED_accept
[ RUN      ] ipc.get_msg_negative
[       OK ] ipc.get_msg_negative
[ RUN      ] ipc.put_msg_negative
[       OK ] ipc.put_msg_negative
[ RUN      ] ipc.send_msg
[       OK ] ipc.send_msg
[ RUN      ] ipc.send_msg_negative
[       OK ] ipc.send_msg_negative
[ RUN      ] ipc.read_msg_negative
[       OK ] ipc.read_msg_negative
[ RUN      ] ipc.end_to_end_msg
[       OK ] ipc.end_to_end_msg
[ RUN      ] ipc.hset_create
[       OK ] ipc.hset_create
[ RUN      ] ipc.hset_add_mod_del
[       OK ] ipc.hset_add_mod_del
```

(continues on next page)

(continued from previous page)

```
[ RUN      ] ipc.hset_add_self
[       OK ] ipc.hset_add_self
[ RUN      ] ipc.hset_add_loop
[       OK ] ipc.hset_add_loop
[ RUN      ] ipc.hset_add_duplicate
[       OK ] ipc.hset_add_duplicate
[ RUN      ] ipc.hset_wait_on_empty_set
[       OK ] ipc.hset_wait_on_empty_set
[ DISABLED ] ipc.DISABLED_hset_add_chan
[ RUN      ] ipc.send_handle_negative
[       OK ] ipc.send_handle_negative
[ RUN      ] ipc.recv_handle
[       OK ] ipc.recv_handle
[ RUN      ] ipc.recv_handle_negative
[       OK ] ipc.recv_handle_negative
[ RUN      ] ipc.echo_handle_bulk
[       OK ] ipc.echo_handle_bulk
[ RUN      ] ipc.tipc_connect
[       OK ] ipc.tipc_connect
[ RUN      ] ipc.tipc_send_recv_1
[       OK ] ipc.tipc_send_recv_1
[ RUN      ] ipc.tipc_send_recv_hdr_payload
[       OK ] ipc.tipc_send_recv_hdr_payload
[==========] 28 tests ran.
[  PASSED  ] 28 tests.
[ DISABLED ] 2 tests.
console:/ #
```

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.

1.5.4 Microdroid Demo unit tests

```
INFO: ADB connecting to 127.0.0.1:5555
INFO: ADB connected to 127.0.0.1:5555
INFO: Checking ro.product.name
INFO: ro.product.name matches tc_fvp
INFO: Checking path of com.android.microdroid.tc
INFO: APK Installed path is: /system/app/TCMicrodroidDemoApp/TCMicrodroidDemoApp.apk
Created VM from "/system/app/TCMicrodroidDemoApp/TCMicrodroidDemoApp.apk"!"assets/vm_
↪config.json" with CID 10, state is NOT_STARTED.
Started VM, state now STARTING.


U-Boot 2022.01-15068-g240b124907 (Apr 14 2022 - 14:14:27 +0000)

DRAM:  256 MiB
## Android Verified Boot 2.0 version 1.1.0
read_is_device_unlocked not supported yet
```

(continues on next page)

(continued from previous page)

```
read_rollback_index not supported yet
read_rollback_index not supported yet
read_rollback_index not supported yet
read_is_device_unlocked not supported yet
Verification passed successfully
Imported supplementary environment
Could not find "misc" partition
## Android Verified Boot 2.0 version 1.1.0
read_is_device_unlocked not supported yet
read_rollback_index not supported yet
read_is_device_unlocked not supported yet
Verification passed successfully
## Android Verified Boot 2.0 version 1.1.0
read_is_device_unlocked not supported yet
read_rollback_index not supported yet
read_rollback_index not supported yet
read_rollback_index not supported yet
read_is_device_unlocked not supported yet
Verification passed successfully
ANDROID: Loading vendor ramdisk from "vendor_boot_a", partition 3.
Booting kernel at 0x80200000 with fdt at 80000000 ramdisk 0x84200000:0x00195c30...


## Flattened Device Tree blob at 80000000
   Booting using the fdt blob at 0x80000000
   Loading Ramdisk to 8eadb000, end 8ec70c30 ... OK
   Loading Device Tree to 000000008ead7000, end 000000008eadab80 ... OK

Starting kernel ...

[    0.136679][    T1] virtio_blk virtio3: [vda] 192768 512-byte logical blocks (98.7 MB/
↪94.1 MiB)
[    0.136743][    T1] vda: detected capacity change from 0 to 98697216
[    0.153152][    T1] GPT:Primary header thinks Alt. header is not at the end of the␣
↪disk.
[    0.153207][    T1] GPT:192712 != 192767
[    0.153244][    T1] GPT:Alternate GPT header not at the end of the disk.
[    0.153312][    T1] GPT:192712 != 192767
[    0.153348][    T1] GPT: Use GNU Parted to correct GPT errors.
[    0.153393][    T1]  vda: vda1 vda2 vda3 vda4 vda5
[    0.156140][    T1] virtio_blk virtio4: [vdb] 20992 512-byte logical blocks (10.7 MB/
↪10.3 MiB)
[    0.156265][    T1] vdb: detected capacity change from 0 to 10747904
[    0.197172][    T1] GPT:Primary header thinks Alt. header is not at the end of the␣
↪disk.
[    0.197566][    T1] GPT:20968 != 20991
[    0.197817][    T1] GPT:Alternate GPT header not at the end of the disk.
[    0.198281][    T1] GPT:20968 != 20991
[    0.198585][    T1] GPT: Use GNU Parted to correct GPT errors.
[    0.198969][    T1]  vdb: vdb1 vdb2 vdb3 vdb4
[    0.201812][    T1] virtio_blk virtio5: [vdc] 3968 512-byte logical blocks (2.03 MB/1.
↪94 MiB)
```

(continues on next page)

(continued from previous page)

```
[    0.202210][    T1] vdc: detected capacity change from 0 to 2031616
[    0.226878][    T1] GPT:Primary header thinks Alt. header is not at the end of the␣
↪disk.
[    0.227043][    T1] GPT:3872 != 3967
[    0.227141][    T1] GPT:Alternate GPT header not at the end of the disk.
[    0.227301][    T1] GPT:3872 != 3967
[    0.227399][    T1] GPT: Use GNU Parted to correct GPT errors.
[    0.227544][    T1]  vdc: vdc1 vdc2 vdc3 vdc4
[    0.242286][    T1] device-mapper: verity: sha1 using implementation "sha1-generic"
[    0.250605][    T1] EXT4-fs (dm-2): mounted filesystem with ordered data mode. Opts:␣
↪errors=panic
[    0.252168][    T1] device-mapper: verity: sha1 using implementation "sha1-generic"
[    0.254868][    T1] EXT4-fs (dm-3): mounted filesystem without journal. Opts:␣
↪errors=panic
[    0.350347][    T1] SELinux:  Permission nlmsg_getneigh in class netlink_route_socket␣
↪not defined in policy.
[    0.350480][    T1] SELinux:  Permission bpf in class capability2 not defined in␣
↪policy.
[    0.350556][    T1] SELinux:  Permission checkpoint_restore in class capability2 not␣
↪defined in policy.
[    0.350652][    T1] SELinux:  Permission bpf in class cap2_userns not defined in␣
↪policy.
[    0.350765][    T1] SELinux:  Permission checkpoint_restore in class cap2_userns not␣
↪defined in policy.
[    0.350898][    T1] SELinux: the above unknown classes and permissions will be denied
[    0.353749][    T1] SELinux:  policy capability network_peer_controls=1
[    0.353824][    T1] SELinux:  policy capability open_perms=1
[    0.353878][    T1] SELinux:  policy capability extended_socket_class=1
[    0.353974][    T1] SELinux:  policy capability always_check_network=0
[    0.354040][    T1] SELinux:  policy capability cgroup_seclabel=0
[    0.354113][    T1] SELinux:  policy capability nnp_nosuid_transition=1
[    0.354210][    T1] SELinux:  policy capability genfs_seclabel_symlinks=0
[    0.500954][   T21] audit: type=1403 audit(1682216952.892:2): auid=4294967295␣
↪ses=4294967295 lsm=selinux res=1
[    0.507132][   T21] audit: type=1404 audit(1682216952.896:3): enforcing=1 old_
↪enforcing=0 auid=4294967295 ses=4294967295 enabled=1 old-enabled=1 lsm=selinux res=1
[    0.705758][  T128] binder: 128:128 transaction failed 29189/-22, size 0-0 line 2758
[    0.705896][  T129] binder: 129:129 transaction failed 29189/-22, size 0-0 line 2758
[    0.730365][  T131] device-mapper: verity: sha256 using implementation "sha256-ce"
[    0.770587][    C0] blk_update_request: I/O error, dev vdc, sector 0 op 0x1:(WRITE)␣
↪flags 0x800 phys_seg 0 prio class 0
[    0.773769][  T137] device-mapper: verity: sha256 using implementation "sha256-ce"
[    0.795051][  T137] EXT4-fs (dm-5): mounted filesystem without journal. Opts: (null)
[    0.800970][  T137] EXT4-fs (loop2): mounted filesystem without journal. Opts: (null)
libc: Access denied finding property "persist.arm64.memtag.default"
libc: Access denied finding property "libc.debug.gwp_asan.sample_rate.microdroid_launcher
↪"
libc: Access denied finding property "libc.debug.gwp_asan.sample_rate.system_default"
libc: Access denied finding property "libc.debug.gwp_asan.process_sampling.microdroid_
↪launcher"
libc: Access denied finding property "libc.debug.gwp_asan.process_sampling.system_default
↪"
```

(continues on next page)

(continued from previous page)

```
libc: Access denied finding property "libc.debug.gwp_asan.max_allocs.microdroid_launcher"
libc: Access denied finding property "libc.debug.gwp_asan.max_allocs.system_default"
libc: Access denied finding property "heapprofd.enable"
libc: Access denied finding property "ro.arch"
libc: Access denied finding property "ro.arch"
libc: Access denied finding property "ro.arch"
[    1.826111][   T21] audit: type=1400 audit(1682216954.216:4): avc:  denied  { getattr␣
↪} for  pid=152 comm="microdroid_laun" path="socket:[11462]" dev="sockfs" ino=11462␣
↪scontext=u:r:microdroid_app:s0 tcontext=u:r:microdroid_manager:s0 tclass=vsock_socket␣
↪permissive=0
Hello Microdroid!
payload finished with exit code 0
[    1.829062][   T18] binder: undelivered transaction 38, process died.
```

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.

1.5.5 Kernel selftest unit tests

```
# ./run_kselftest.sh --summary
[  407.778719][  T234] kselftest: Running tests in arm64
TAP version 13
1..10
# selftests: arm64: check_gcr_el1_cswitch
ok 1 selftests: arm64: check_gcr_el1_cswitch
# selftests: arm64: check_ksm_options
not ok 2 selftests: arm64: check_ksm_options # exit=1
# selftests: arm64: check_tags_inclusion
ok 3 selftests: arm64: check_tags_inclusion
# selftests: arm64: check_user_mem
ok 4 selftests: arm64: check_user_mem
# selftests: arm64: check_mmap_options
ok 5 selftests: arm64: check_mmap_options
# selftests: arm64: check_child_memory
ok 6 selftests: arm64: check_child_memory
# selftests: arm64: check_buffer_fill
ok 7 selftests: arm64: check_buffer_fill
# selftests: arm64: btitest
ok 8 selftests: arm64: btitest
# selftests: arm64: nobtitest
ok 9 selftests: arm64: nobtitest
# selftests: arm64: pac
ok 10 selftests: arm64: pac
#
```

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.

1.5.6 MPAM unit tests

```
# testing_mpam.sh
Testing the number of partitions supported.  It should be 0-63
Pass

Partition 0 is the default partition to which all tasks will be assigned.  Checking if␣
↪task 5 is assigned to partition 0
Pass

Testing the number of bits required to set the cache portion bitmap. It should be 8
Pass

Testing the default cpbm configured in the DSU for all the partitions.  It should be 0-7␣
↪for all the partitions
[  305.081818][  T236] MPAM_arch: PART_SEL: 0x0
Pass

Setting the cpbm 4-5 (00110000) in DSU for partition 45 and reading it back
[  305.081969][  T233] MPAM_arch: PART_SEL: 0x2d
[  305.081974][  T233] MPAM_arch: CPBM: 0x30 @ffff80000a803000
[  305.082588][  T237] MPAM_arch: PART_SEL: 0x2d
Pass

#
```

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.

1.5.7 MPMM unit tests

```
# test_mpmm.sh fvp
This is a test script to check the MPMM functionality

This is based on the PCT configured in the SCP which can be found at
product/tc2/scp_ramfw/config_mpmm.c

Testing MPMM in FVP

Testing the MPMM of A520 cores
******************************
According to the PCT, the max frequency should be 1840000
Current set frequency of the cpu0 is 1537000
PASS

Starting a vector intensive workload on cpu0
According to the PCT, the max frequency should be 1537000
Current set frequency of the cpu0 is 1537000
PASS
```

(continues on next page)

(continued from previous page)

```
Starting a vector intensive workload on cpu1
According to the PCT, the max frequency should be 1537000
Current set frequency of the cpu0 is 1537000
PASS

Starting a vector intensive workload on cpu2
According to the PCT, the max frequency should be 1153000
Current set frequency of the cpu0 is 1153000
PASS

Starting a vector intensive workload on cpu3
According to the PCT, the max frequency should be 1153000
Current set frequency of the cpu0 is 1153000
PASS

Testing the MPMM of A720 cores
******************************
According to the PCT, the max frequency should be 2271000
Current set frequency of the cpu4 is 1893000
PASS

Starting a vector intensive workload on cpu4
According to the PCT, the max frequency should be 1893000
Current set frequency of the cpu4 is 1893000
PASS

Starting a vector intensive workload on cpu5
According to the PCT, the max frequency should be 1893000
Current set frequency of the cpu4 is 1893000
PASS

Starting a vector intensive workload on cpu6
According to the PCT, the max frequency should be 1893000
Current set frequency of the cpu4 is 1893000
PASS

Testing the MPMM of X4 cores
******************************
According to the PCT, the max frequency should be 3047000
Current set frequency of the cpu7 is 1088000
PASS

Starting a vector intensive workload on cpu7
According to the PCT, the max frequency should be 2612000
Current set frequency of the cpu7 is 2612000
PASS
#
```

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.

1.5.8 BTI unit tests

```
console:/data/nativetest64/bti-unit-tests # ./bti-unit-tests

[==========] Running 17 tests from 7 test suites.
[----------] Global test environment set-up.
[----------] 3 tests from BR_Test
[ RUN      ] BR_Test.GuardedMemoryWithX16OrX17
[       OK ] BR_Test.GuardedMemoryWithX16OrX17 (181 ms)
[ RUN      ] BR_Test.NonGuardedMemoryAnyRegister
[       OK ] BR_Test.NonGuardedMemoryAnyRegister (0 ms)
[ RUN      ] BR_Test.GuardedMemoryOtherRegisters
[       OK ] BR_Test.GuardedMemoryOtherRegisters (122 ms)
[----------] 3 tests from BR_Test (304 ms total)

[----------] 3 tests from BRAA_Test
[ RUN      ] BRAA_Test.GuardedMemoryWithX16OrX17
[       OK ] BRAA_Test.GuardedMemoryWithX16OrX17 (344 ms)
[ RUN      ] BRAA_Test.NonGuardedMemoryAnyRegister
[       OK ] BRAA_Test.NonGuardedMemoryAnyRegister (0 ms)
[ RUN      ] BRAA_Test.GuardedMemoryOtherRegisters
[       OK ] BRAA_Test.GuardedMemoryOtherRegisters (233 ms)
[----------] 3 tests from BRAA_Test (578 ms total)

[----------] 3 tests from BRAB_Test
[ RUN      ] BRAB_Test.GuardedMemoryWithX16OrX17
[       OK ] BRAB_Test.GuardedMemoryWithX16OrX17 (310 ms)
[ RUN      ] BRAB_Test.NonGuardedMemoryAnyRegister
[       OK ] BRAB_Test.NonGuardedMemoryAnyRegister (0 ms)
[ RUN      ] BRAB_Test.GuardedMemoryOtherRegisters
[       OK ] BRAB_Test.GuardedMemoryOtherRegisters (297 ms)
[----------] 3 tests from BRAB_Test (608 ms total)

[----------] 2 tests from BLR_Test
[ RUN      ] BLR_Test.GuardedMemoryAnyRegister
[       OK ] BLR_Test.GuardedMemoryAnyRegister (332 ms)
[ RUN      ] BLR_Test.NonGuardedMemoryAnyRegister
[       OK ] BLR_Test.NonGuardedMemoryAnyRegister (0 ms)
[----------] 2 tests from BLR_Test (333 ms total)

[----------] 2 tests from BLRAA_Test
[ RUN      ] BLRAA_Test.GuardedMemoryAnyRegister

[       OK ] BLRAA_Test.GuardedMemoryAnyRegister (745 ms)
[ RUN      ] BLRAA_Test.NonGuardedMemoryAnyRegister
[       OK ] BLRAA_Test.NonGuardedMemoryAnyRegister (0 ms)
[----------] 2 tests from BLRAA_Test (745 ms total)

[----------] 2 tests from BLRAB_Test
[ RUN      ] BLRAB_Test.GuardedMemoryAnyRegister
[       OK ] BLRAB_Test.GuardedMemoryAnyRegister (748 ms)
[ RUN      ] BLRAB_Test.NonGuardedMemoryAnyRegister
[       OK ] BLRAB_Test.NonGuardedMemoryAnyRegister (0 ms)
```

(continues on next page)

(continued from previous page)

```
[----------] 2 tests from BLRAB_Test (748 ms total)

[----------] 2 tests from BTI_LinkerTest
[ RUN      ] BTI_LinkerTest.CallBasicFunction
[       OK ] BTI_LinkerTest.CallBasicFunction (0 ms)
[ RUN      ] BTI_LinkerTest.BypassLandingPad
[       OK ] BTI_LinkerTest.BypassLandingPad (35 ms)
[----------] 2 tests from BTI_LinkerTest (35 ms total)

[----------] Global test environment tear-down
[==========] 17 tests from 7 test suites ran. (3354 ms total)
[  PASSED  ] 17 tests.
```

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.

1.5.9 MTE unit tests

```
console:/data/nativetest64/mte-unit-tests # ./mte-unit-tests

[==========] Running 12 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 12 tests from MTETest
[ RUN      ] MTETest.CreateRandomTag
[       OK ] MTETest.CreateRandomTag (0 ms)
[ RUN      ] MTETest.IncrementTag
[       OK ] MTETest.IncrementTag (0 ms)
[ RUN      ] MTETest.ExcludedTags
[       OK ] MTETest.ExcludedTags (0 ms)
[ RUN      ] MTETest.PointerSubtraction
[       OK ] MTETest.PointerSubtraction (0 ms)
[ RUN      ] MTETest.TagStoreAndLoad
[       OK ] MTETest.TagStoreAndLoad (0 ms)
[ RUN      ] MTETest.DCGZVA
[       OK ] MTETest.DCGZVA (0 ms)
[ RUN      ] MTETest.DCGVA
[       OK ] MTETest.DCGVA (0 ms)
[ RUN      ] MTETest.Segfault
[       OK ] MTETest.Segfault (41 ms)
[ RUN      ] MTETest.UseAfterFree
[       OK ] MTETest.UseAfterFree (0 ms)
[ RUN      ] MTETest.CopyOnWrite
[       OK ] MTETest.CopyOnWrite (0 ms)
[ RUN      ] MTETest.mmapTempfile
[       OK ] MTETest.mmapTempfile (5 ms)
[ RUN      ] MTETest.MTEIsEnabled
[       OK ] MTETest.MTEIsEnabled (0 ms)
[----------] 12 tests from MTETest (48 ms total)
```

(continues on next page)

(continued from previous page)

```
[----------] Global test environment tear-down
[==========] 12 tests from 1 test suite ran. (48 ms total)
[  PASSED  ] 12 tests.
```

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.

1.5.10 PAUTH unit tests

```
console:/data/nativetest64/pauth-unit-tests $ ./pauth-unit-tests
PAC is enabled by the kernel: 1
PAC2 is implemented by the hardware: 1
FPAC is implemented by the hardware: 1
[==========] Running 18 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 2 tests from PAuthDeathTest
[ RUN      ] PAuthDeathTest.SignFailure
[       OK ] PAuthDeathTest.SignFailure (113 ms)
[ RUN      ] PAuthDeathTest.AuthFailure
[       OK ] PAuthDeathTest.AuthFailure (137 ms)
[----------] 2 tests from PAuthDeathTest (250 ms total)

[----------] 13 tests from PAuthTest
[ RUN      ] PAuthTest.Signing
[       OK ] PAuthTest.Signing (0 ms)
[ RUN      ] PAuthTest.Authentication
[       OK ] PAuthTest.Authentication (146 ms)
[ RUN      ] PAuthTest.Stripping
vendor/arm/examples/pauth/pauth_unit_tests/pauth_unit_tests.cpp:279: Skipped

[  SKIPPED ] PAuthTest.Stripping (0 ms)
[ RUN      ] PAuthTest.Roundtrip
[       OK ] PAuthTest.Roundtrip (0 ms)
[ RUN      ] PAuthTest.StrippingWithBuiltinReturnAddress
[       OK ] PAuthTest.StrippingWithBuiltinReturnAddress (0 ms)
[ RUN      ] PAuthTest.ExtractPAC
[       OK ] PAuthTest.ExtractPAC (0 ms)
[ RUN      ] PAuthTest.PACMask
[       OK ] PAuthTest.PACMask (0 ms)
[ RUN      ] PAuthTest.KeyChange
[       OK ] PAuthTest.KeyChange (1 ms)
[ RUN      ] PAuthTest.GenericAuthentication
[       OK ] PAuthTest.GenericAuthentication (0 ms)
[ RUN      ] PAuthTest.Unwind
[       OK ] PAuthTest.Unwind (8 ms)
[ RUN      ] PAuthTest.CheckReturnAddressSigned
[       OK ] PAuthTest.CheckReturnAddressSigned (0 ms)
[ RUN      ] PAuthTest.AuthenticateThenReturn
[       OK ] PAuthTest.AuthenticateThenReturn (93 ms)
```

(continues on next page)

(continued from previous page)

```
[ RUN      ] PAuthTest.CheckHWCAP
[       OK ] PAuthTest.CheckHWCAP (0 ms)
[----------] 13 tests from PAuthTest (251 ms total)

[----------] 3 tests from PAuthTestData
[ RUN      ] PAuthTestData.Signing
[       OK ] PAuthTestData.Signing (0 ms)
[ RUN      ] PAuthTestData.Authentication
[       OK ] PAuthTestData.Authentication (92 ms)
[ RUN      ] PAuthTestData.Roundtrip
[       OK ] PAuthTestData.Roundtrip (0 ms)
[----------] 3 tests from PAuthTestData (92 ms total)

[----------] Global test environment tear-down
[==========] 18 tests from 3 test suites ran. (594 ms total)
[  PASSED  ] 17 tests.
[  SKIPPED ] 1 test, listed below:
[  SKIPPED ] PAuthTest.Stripping
```

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.

1.5.11 EAS with Lisa unit tests

```
The following expressions will be executed:

EnergyModelWakeMigration:test_dmesg
EnergyModelWakeMigration:test_slack
EnergyModelWakeMigration:test_task_placement
OneSmallTask:test_dmesg
OneSmallTask:test_slack
OneSmallTask:test_task_placement
RampDown:test_dmesg
RampDown:test_slack
RampDown:test_task_placement
RampUp:test_dmesg
RampUp:test_slack
RampUp:test_task_placement
ThreeSmallTasks:test_dmesg
ThreeSmallTasks:test_slack
ThreeSmallTasks:test_task_placement
TwoBigTasks:test_dmesg
TwoBigTasks:test_slack
TwoBigTasks:test_task_placement
TwoBigThreeSmall:test_dmesg
TwoBigThreeSmall:test_slack
TwoBigThreeSmall:test_task_placement

Used trace events:
```

(continues on next page)

(continued from previous page)

```
  - sched_switch
  - sched_wakeup
  - sched_wakeup_new
  - task_rename
  - userspace@rtapp_loop
  - userspace@rtapp_stats

(...output truncated...)

[2023-09-05 10:57:24,399][EXEKALL] INFO  Result summary:
EnergyModelWakeMigration[board=tc]:test_dmesg                    ␣
↪UUID=40d036f38fd64fbbbffbea7d2e9ddbc6 PASSED: dmesg output:
EnergyModelWakeMigration[board=tc]:test_slack                    ␣
↪UUID=0843b7d55cc5498094b2dd7856adddab PASSED: emwm_0-0 delayed activations: 2.
↪7944111776447107 %
EnergyModelWakeMigration[board=tc]:test_task_placement           ␣
↪UUID=b88941b8bc1c44b19722dc7d3ef087ba PASSED
        energy threshold: 7737.468929123666 bogo-joules
        estimated energy: 7146.545812339864 bogo-joules
        noisiest task:
            comm: kworker/5:1
            duration (abs): 0.0006058100001951061 s
            duration (rel): 0.007507425508119742 %
            pid: 69

OneSmallTask[board=tc]:test_dmesg                                ␣
↪UUID=1429aa06cf5242ef8fb9fd3929b85361 PASSED: dmesg output:
OneSmallTask[board=tc]:test_slack                                ␣
↪UUID=cf55513dad2d4bc4bfd4da4385edc1b3 PASSED: small-0 delayed activations: 0.0 %
OneSmallTask[board=tc]:test_task_placement                       ␣
↪UUID=398f2a58624841c2beeda8feb7a68844 PASSED
        energy threshold: 60.32967281693774 bogo-joules
        estimated energy: 57.456831254226415 bogo-joules
        noisiest task:
            comm: init
            duration (abs): 0.0001531899999918096 s
            duration (rel): 0.015442352118088516 %
            pid: 1

RampDown[board=tc]:test_dmesg                                    ␣
↪UUID=f4a8f3054eb34bdeaff8545408d87817 PASSED: dmesg output:
RampDown[board=tc]:test_slack                                    ␣
↪UUID=049088a94b1a401d9a9faf572dbb969e PASSED: down-0 delayed activations: 0.
↪2145922746781116 %
RampDown[board=tc]:test_task_placement                           ␣
↪UUID=44b916a863e74f1aa8fe8322d958e2cd PASSED
        energy threshold: 5075.609124026213 bogo-joules
        estimated energy: 4499.802859646464 bogo-joules
        noisiest task:
            comm: kworker/5:1
            duration (abs): 0.0005715100001566498 s
            duration (rel): 0.007681529072919605 %
```

(continues on next page)

(continued from previous page)

```
            pid: 69

RampUp[board=tc]:test_dmesg                                 ␣
↪UUID=df45eca48be84805ac07162e0313b614 PASSED: dmesg output:
RampUp[board=tc]:test_slack                                 ␣
↪UUID=c59e63e892c2436d93f09db4a9b690e0 PASSED: up-0 delayed activations: 0.0 %
RampUp[board=tc]:test_task_placement                        ␣
↪UUID=88fe93bcbc004665b92678564355e4c9 PASSED
        energy threshold: 4511.9916490978 bogo-joules
        estimated energy: 3825.2514695943355 bogo-joules
        noisiest task:
            comm: kworker/5:1
            duration (abs): 0.0005233000001680921 s
            duration (rel): 0.00703361144586148 %
            pid: 69

ThreeSmallTasks[board=tc]:test_dmesg                        ␣
↪UUID=6da8a02fbce14536a5497368576f41a1 PASSED: dmesg output:
ThreeSmallTasks[board=tc]:test_slack                        ␣
↪UUID=80d93c17789d44a79734f1caefea1571 PASSED
        small_0-0 delayed activations: 0.0 %
        small_1-1 delayed activations: 0.0 %
        small_2-2 delayed activations: 0.0 %

ThreeSmallTasks[board=tc]:test_task_placement               ␣
↪UUID=cd4d878f560d4bfcb47aeea155aa9680 PASSED
        energy threshold: 206.84414838330824 bogo-joules
        estimated energy: 172.37012365275686 bogo-joules
        noisiest task:
            comm: init
            duration (abs): 0.00015036999991480116 s
            duration (rel): 0.015158063503586424 %
            pid: 1

TwoBigTasks[board=tc]:test_dmesg                            ␣
↪UUID=c4048cd03dfa40c4ace83722b6add40f PASSED: dmesg output:
TwoBigTasks[board=tc]:test_slack                            ␣
↪UUID=03d5041fb8fb4e0ea4ca3b9ddbf33be8 SKIPPED: skipped-reason: The workload will␣
↪result in overutilized status for all possible task placement, making it unsuitable to␣
↪test EAS on this platform
TwoBigTasks[board=tc]:test_task_placement                   ␣
↪UUID=3fc72b13185c4d149a4f8057e9c9c138 SKIPPED: skipped-reason: The workload will␣
↪result in overutilized status for all possible task placement, making it unsuitable to␣
↪test EAS on this platform
TwoBigThreeSmall[board=tc]:test_dmesg                       ␣
↪UUID=bdaf2fba329b474a8377631cac3aa4d0 PASSED: dmesg output:
TwoBigThreeSmall[board=tc]:test_slack                       ␣
↪UUID=6c5fbb8dae80474aa89ed1edee65734c SKIPPED: skipped-reason: The workload will␣
↪result in overutilized status for all possible task placement, making it unsuitable to␣
↪test EAS on this platform
TwoBigThreeSmall[board=tc]:test_task_placement              ␣
↪UUID=14a242efc9184c3aaf46d41c0192b42f SKIPPED: skipped-reason: The workload will␣
```

(continues on next page)

(continued from previous page)

```
↪result in overutilized status for all possible task placement, making it unsuitable to␣
↪test EAS on this platform
```

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.

1.5.12 CPU hardware capabilities

```
# test_feats_arch.sh
Testing FEAT_AFP HW CAP
Pass

Testing FEAT_ECV HW CAP
Pass

Testing FEAT_WFXT HW CAP
Pass

#
```

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 - TC2-2023.10.04

Contents

1.7.1 Release tag

The manifest tag for this release is `TC2-2023.10.04`.

1.7.2 Components

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

- BSP build supporting Android, Buildroot and Debian distros;
- Trusted firmware-A for secure boot;
- U-Boot bootloader;
- Hafnium for S-EL2 Secure Partition Manager core;
- OP-TEE for Trusted Execution Environment (TEE) in Buildroot;

- Trusted Services (Crypto, Internal Trusted Storage and Firmware Update) in Buildroot;
- Trusty for Trusted Execution Environment (TEE) with FF-A messaging in Android;
- System control processor(SCP) firmware for programming the interconnect, power control etc;
- Runtime Security Subsystem (RSS) firmware for providing HW RoT;
- TensorFlow Lite Machine Learning;
- Enable full-HD (1920x1080-60fps) resolution support for use with the FVP model.

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

- Buildroot distribution support;
- Debian 12 (aka Bookworm);
- Android 13 support;
- Android Common Kernel 5.15.41;
- Android Hardware Rendering with Mali-G720 GPU - DDK r41p0_01eac0 (source code) or prebuilt binaries;
- Android Software rendering with DRM Hardware Composer offloading composition to Mali D71 DPU;
- KVM default mode of operation is set to `protected` by default, thus effectively enabling pKVM on the system. This is a nVHE based mode with kernel running at EL1;
- Microdroid based pVM support in Android;
- GPU and DPU support for S1 and S2 translation squashed with SMMU-700;
- Maximum Power Mitigation Mechanism (MPMM) support;
- GPU DVFS/Idle power states support;
- Support for MPAM (more info available at link);
- Support for EAS (more info available at link);
- Trusted Firmware-A v2.9;
- Hafnium v2.9;

- OP-TEE 3.22.0;
- Trusty with FF-A messaging - FF-A v1.0;
- CI700-PMU enabled for profiling;
- 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;
- U-Boot bootloader v2023.04;
- Power management features: cpufreq and cpuidle;
- SCMI (System Control and Management Interface) support;
- Virtio to mount the android image in the host machine as a storage device in the FVP;
- Verified u-boot for authenticating fit image (containing kernel + ramdisk) during Buildroot boot;
- Android Verified Boot (AVB) for authenticating boot and system image during Android boot;
- Hafnium as Secure Partition Manager (SPM) at S-EL2;
- OP-TEE as Secure Partition at S-EL1, managed by S-EL2 SPMC (Hafnium);
- Arm FF-A driver and FF-A Transport support for OP-TEE driver in Android Common Kernel;
- OP-TEE Support in Buildroot distribution. This includes OP-TEE client and OP-TEE test suite;
- Trusted Services (Crypto, Internal Trusted Storage and Firmware Update) running at S-EL0;
- Trusted Services test suite added to Buildroot distribution;
- Tracing support, based on ETE and TRBE v1.1 in TF-A, kernel and `simpleperf`. Traces can be captured with `simpleperf`.
- Firmware update support.

1.7.5 Platform Support

- This software release is tested on TC2 Fast Model platform (FVP) version 11.23.17.

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. To avoid the waiting time experienced during U-Boot boot time, the user may interrupt the auto-boot process when presented with the message `Hit any key to stop autoboot: X` by pressing the `ENTER` key and, on the presented command prompt, type `boot` followed by `ENTER` key to confirm command to immediately boot the distro kernel image. Although the configured delay is shown as 1-3 seconds, it will take considerably more time to boot (approximately 15 seconds) due to the time difference in the CPU frequency and the FVP operating frequency;
2. Ubuntu 22.04 is not supported in this release;
3. SVE2 (Scalable Vector Extension) feature is not supported with this release;

4. For Android builds which do use the TAP network interface, the default browser available in Android (`webview_shell`) is not able to open HTTPS urls. Interested users can attempt to circumvent this limitation by getting the ARM64 specific APK package for other browsers (e.g. Mozilla Firefox), install it using ADB, and use it to browse HTTPS urls;

5. Android builds with software or hardware rendering support do not properly initialiase the KVM during boot and will show a kernel warning dump during boot, similar to the following excerpt:

```
(...)
[    0.079881][    T1] kvm [1]: IPA Size Limit: 40 bits
[    0.080735][    T1] ------------[ cut here ]------------
[    0.080816][    T1] WARNING: CPU: 9 PID: 1 at arch/arm64/kvm/arm.c:1675␣
↪cpu_hyp_init_context+0x154/0x160
[    0.080965][    T1] Modules linked in:
[    0.081024][    T1] CPU: 9 PID: 1 Comm: swapper/0 Tainted: G S          ␣
↪     5.15.41-g7ed92d32a9ad #1
[    0.081165][    T1] Hardware name: arm,tc (DT)
[    0.081233][    T1] pstate: 80000005 (Nzcv daif -PAN -UAO -TCO -DIT -
↪SSBS BTYPE=--)
[    0.081350][    T1] pc : cpu_hyp_init_context+0x154/0x160
[    0.081433][    T1] lr : cpu_hyp_init_context+0xec/0x160
[    0.081515][    T1] sp : ffff80000a5cbc90
[    0.081576][    T1] x29: ffff80000a5cbc90 x28: 0000000000000000 x27:␣
↪000000000000000d
[    0.081695][    T1] x26: ffff800009f6d100 x25: 0000000000000004 x24:␣
↪ffff80000a49a000
[    0.081815][    T1] x23: ffff8000098fd000 x22: 0000000000000030 x21:␣
↪ffff80000a49b000
[    0.081934][    T1] x20: ffff80000a49b000 x19: ffff00000216d100 x18:␣
↪0000000000000000
[    0.082053][    T1] x17: 6120737265746e75 x16: 0000000000000008 x15:␣
↪0000000000000000
[    0.082172][    T1] x14: 0000000000000000 x13: 0000000000000000 x12:␣
↪0000000000000001
[    0.082292][    T1] x11: 0000000000000001 x10: 000000000015f258 x9 :␣
↪ffff80000a2c5920
[    0.02411][    T1] x8 : ffff80000a5cbbd0 x7 : 0000000000000000 x6 :␣
↪00000081eefff000
[    0.082530][    T1] x5 : ffff80000a5cbbd0 x4 : 00000081eefff000 x3 :␣
↪00000000c6000000
[    0.082650][    T1] x2 : 0001000000000000 x1 : 0000008081b4d080 x0 :␣
↪ffffffffffffffff
[    0.082769][    T1] Call trace:
[    0.082818][    T1]  cpu_hyp_init_context+0x154/0x160
[    0.082895][    T1]  kvm_arch_init+0xc58/0xea0
[    0.082965][    T1]  kvm_init+0x3c/0x350
[    0.083027][    T1]  arm_init+0x20/0x30
[    0.083086][    T1]  do_one_initcall+0x44/0x290
[    0.083156][    T1]  kernel_init_freeable+0x250/0x2d4
[    0.083236][    T1]  kernel_init+0x28/0x130
[    0.083301][    T1]  ret_from_fork+0x10/0x20
[    0.083368][    T1] ---[ end trace e2459f77e453d262 ]---
[    0.083451][    T1] ------------[ cut here ]------------
[    0.083532][    T1] WARNING: CPU: 9 PID: 1 at arch/arm64/kvm/arm.c:1972␣
```

(continues on next page)

(continued from previous page)

```
↪kvm_arch_init+0xd34/0xea0
[    0.083667][    T1] Modules linked in:
[    0.083725][    T1] CPU: 9 PID: 1 Comm: swapper/0 Tainted: G S      W   ␣
↪      5.15.41-g7ed92d32a9ad #1
[    0.083866][    T1] Hardware name: arm,tc (DT)
[    0.083934][    T1] pstate: 80000005 (Nzcv daif -PAN -UAO -TCO -DIT -
↪SSBS BTYPE=--)
[    0.084050][    T1] pc : kvm_arch_init+0xd34/0xea0
[    0.084124][    T1] lr : kvm_arch_init+0xca4/0xea0
[    0.084198][    T1] sp : ffff80000a5cbca0
[    0.084259][    T1] x29: ffff80000a5cbca0 x28: 0000000000000000 x27:␣
↪000000000000000d
[    0.084379][    T1] x26: ffff800009f6d100 x25: 0000000000000004 x24:␣
↪ffff80000a49a000
[    0.084498][    T1] x23: ffff8000098fd000 x22: 0000000000000030 x21:␣
↪00000081ec200000
[    0.084617][    T1] x20: 0000000002e00000 x19: 00000081ec200000 x18:␣
↪0000000000000000
[    0.084736][    T1] x17: 6120737265746e75 x16: 0000000000000008 x15:␣
↪0000000000000000
[    0.084855][    T1] x14: 0000000000000000 x13: 0000000000000000 x12:␣
↪0000000000000001
[    0.084975][    T1] x11: 0000000000000001 x10: 000000000015f258 x9 :␣
↪ffff80000a2c5920
[    0.085094][    T1] x8 : ffff80000a5cbbd0 x7 : 0000000000000000 x6 :␣
↪0000000000000000
[    0.085213][    T1] x5 : 0000000000000030 x4 : 0000f1000216d100 x3 :␣
↪00000000c6000000
[    0.085333][    T1] x2 : 0000000002e00000 x1 : 00000081ec200000 x0 :␣
↪ffffffffffffffff
[    0.085452][    T1] Call trace:
[    0.085500][    T1]  kvm_arch_init+0xd34/0xea0
[    0.085569][    T1]  kvm_init+0x3c/0x350
[    0.085629][    T1]  arm_init+0x20/0x30
[    0.085689][    T1]  do_one_initcall+0x44/0x290
[    0.085758][   T1]  kernel_init_freeable+0x250/0x2d4
[    0.085836][    T1]  kernel_init+0x28/0x130
[    0.085900][    T1]  ret_from_fork+0x10/0x20
[    0.085965][    T1] ---[ end trace e2459f77e453d263 ]---
[    0.086052][    T1] ------------[ cut here ]------------
[    0.086133][    T1] WARNING: CPU: 9 PID: 1 at arch/arm64/kvm/arm.c:1726␣
↪cpu_set_hyp_vector+0xb0/0xd4
[    0.086272][    T1] Modules linked in:
[    0.086330][    T1] CPU: 9 PID: 1 Comm: swapper/0 Tainted: G S      W   ␣
↪      5.15.41-g7ed92d32a9ad #1
[    0.086471][    T1] Hardware name: arm,tc (DT)
[    0.086538][    T1] pstate: 80000005 (Nzcv daif -PAN -UAO -TCO -DIT -
↪SSBS BTYPE=--)
[    0.086655][    T1] pc : cpu_set_hyp_vector+0xb0/0xd4
[    0.086733][    T1] lr : cpu_set_hyp_vector+0xa8/0xd4
[    0.086810][    T1] sp : ffff80000a5cbc90
[    0.086871][    T1] x29: ffff80000a5cbc90 x28: 0000000000000000 x27:␣
```

(continues on next page)

(continued from previous page)

```
↪000000000000000d
[    0.086991][    T1] x26: ffff800009f6d100 x25: 0000000000000004 x24:␣
↪ffff80000a49a000
[    0.087110][    T1] x23: ffff8000098fd000 x22: 0000000000000030 x21:␣
↪00000081ec200000
[    0.087229][    T1] x20: 0000000002e00000 x19: 00000081ec200000 x18:␣
↪0000000000000000
[    0.087348][    T1] x17: 6120737265746e75 x16: 000000000000000a x15:␣
↪0000000000000000
[    0.087468][    T1] x14: 0000000000000000 x13: 0000000000000000 x12:␣
↪0000000000000001
[    0.087587][    T1] x11: 0000000000000001 x10: 000000000015f258 x9 :␣
↪ffff80000a2c5920
[    0.087706][    T1] x8 : ffff80000a5cbbd0 x7 : 0000000000000000 x6 :␣
↪0000000000000000
[    0.087825][    T1] x5 : 0000000000000030 x4 : 0000f1000216d100 x3 :␣
↪00000000c6000000
[    0.087945][    T1] x2 : ffff80000a5cbc90 x1 : 0000000000000000 x0 :␣
↪ffffffffffffffff
[    0.088064][    T1] Call trace:
[    0.088112][    T1]  cpu_set_hyp_vector+0xb0/0xd4
[    0.088184][    T1]  kvm_arch_init+0xcb4/0xea0
[    0.088253][    T1]  kvm_init+0x3c/0x350
[    0.088314][    T1]  arm_init+0x20/0x30
[    0.088373][    T1]  do_one_initcall+0x44/0x290
[    0.088443][    T1]  kernel_init_freeable+0x250/0x2d4
[    0.088520][    T1]  kernel_init+0x28/0x130
[    0.088585][    T1]  ret_from_fork+0x10/0x20
[    0.088650][    T1] ---[ end trace e2459f77e453d264 ]---
[    0.088734][    T1] ------------[ cut here ]------------
[    0.088815][    T1] WARNING: CPU: 9 PID: 1 at arch/arm64/kvm/debug.c:68␣
↪kvm_arm_init_debug+0x5c/0x64
[    0.088956][    T1] Modules linked in:
[    0.089013][    T1] CPU: 9 PID: 1 omm: swapper/0 Tainted: G S      W    ␣
↪    5.15.41-g7ed92d32a9ad #1
[    0.089154][    T1] Hardware name: arm,tc (DT)
[    0.089222][    T1] pstate: 80000005 (Nzcv daif -PAN -UAO -TCO -DIT -
↪SSBS BTYPE=--)
[    0.089338][    T1] pc : kvm_arm_init_debug+0x5c/0x64
[    0.089416][    T1] lr : kvm_arm_init_debug+0x24/0x64
[    0.089493][    T1] sp : ffff80000a5cbc90
[    0.089554][    T1] x29: ffff80000a5cbc90 x28: 00000000ec200000 x27:␣
↪000000000000000d
[    0.089673][    T1] x26: ffff800009f6d100 x25: 0000000000000004 x24:␣
↪ffff80000a49a000
[    0.089793][    T1] x23: ffff8000098fd000 x22: 0000000000000030 x21:␣
↪00000081ec200000
[    0.089912][    T1] x20: 0000000002e00000 x19: 00000081ec200000 x18:␣
↪0000000000000000
[    0.090031][    T1] x17: 6120737265746e75 x16: 000000000000000a x15:␣
↪0000000000000000
[    0.090150][    T1] x14: 0000000000000000 x13: 0000000000000000 x12:␣
```

(continues on next page)

(continued from previous page)

```
↪0000000000000001
[    0.090270][    T1] x11: 0000000000000001 x10: 000000000015f258 x9 :␣
↪ffff80000a2c5920
[    0.090389][    T1] x8 : ffff80000a5cbbd0 x7 : 0000000000000000 x6 :␣
↪0000000000000000
[    0.090508][    T1] x5 : 0000000000000030 x4 : 0000f1000216d100 x3 :␣
↪00000000c6000000
[    0.090627][    T1] x2 : ffff80000a5cbc90 x1 : 0000000000000000 x0 :␣
↪ffffffffffffffff
[    0.090746][    T1] Call trace:
[    0.090795][    T1]  kvm_arm_init_debug+0x5c/0x64
[    0.090867][    T1]  kvm_arch_init+0xcbc/0xea0
[    0.090936][    T1]  kvm_init+0x3c/0x350
[    0.090997][    T1]  arm_init+0x20/0x30
[    0.091056][    T1]  do_one_initcall+0x44/0x290
[    0.091125][    T1]  kernel_init_freeable+0x250/0x2d4
[    0.091203][    T1]  kernel_init+0x28/0x130
[    0.091267][    T1]  ret_from_fork+0x10/0x20
[    0.091333][    T1] ---[ end trace e2459f77e453d265 ]---
[    0.091417][    T1] kvm [1]: Failed to init hyp memory protection
[    0.091657][    T1] kvm [1]: error initializing Hyp mode: -333447168
(...)
```

6. Android builds with software or hardware rendering support will present an SMC blocked call message on FVP `terminal_uart1_ap` window, similar to the following excerpt:

```
(...)
NOTICE:  Booting Trusted Firmware
NOTICE:  BL1: v2.9(debug):v2.9.0-291-g68e93909f
NOTICE:  BL1: Built : 13:08:36, Aug 23 2023
NOTICE:  BL1: Booting BL2
NOTICE:  BL2: v2.9(debug):v2.9.0-291-g68e93909f
NOTICE:  BL2: Built : 13:08:40, Aug 23 2023
NOTICE:  BL1: Booting BL31
NOTICE:  BL31: v2.9(debug):v2.9.0-291-g68e93909f
NOTICE:  BL31: Built : 13:08:48, Aug 23 2023
INFO: Initializing Hafnium (SPMC)
INFO: text: 0xfd000000 - 0xfd027000
INFO: rodata: 0xfd027000 - 0xfd02e000
INFO: data: 0xfd02e000 - 0xfd117000
INFO: stacks: 0xfd120000 - 0xfd130000
INFO: Supported bits in physical address: 40
INFO: Stage 2 has 3 page table levels with 2 pages at the root.
INFO: Stage 1 has 4 page table levels with 1 pages at the root.
INFO: Memory range:  0xf9000000 - 0xfeffffff
INFO: Loading VM id 0x8001: trusty.
INFO: Loaded with 8 vCPUs, entry at 0xf901c000.
INFO: Hafnium initialisation completed
NOTICE: SMC 0xbd000000 attempted from VM 0x8001, blocked=1
NOTICE: SMC 0xbd000000 attempted from VM 0x8001, blocked=1
NOTICE: SMC 0xbd000000 attempted from VM 0x8001, blocked=1
(...OUTPUT TRUNCATED TO SAVE SPACE AND REPETITION...)
```

(continues on next page)

(continued from previous page)

```
NOTICE: SMC 0xbd000000 attempted from VM 0x8001, blocked=1
NOTICE: SMC 0xbd000000 attempted from VM 0x8001, blocked=1
NOTICE: SMC 0xbd000000 attempted from VM 0x8001, blocked=1
(...)
```

7. The Android PAUTH sanity test may sometimes report inconsistent failing test results (this behaviour is currently under investigation). If experiencing this situation, please repeat the test a few times to validate the feature.

1.7.8 Support

For support email: support@arm.com.

1.8 Change Log

Contents

This document contains a summary of the new features, changes and fixes in each release of TC2 software stack.

1.8.1 Version TC2-2023.10.04

Features added

- GPU and DPU support for S1 and S2 translation squashed with SMMU-700;
- Enable full-HD (1920x1080-60fps) resolution support for use with the FVP model;

1.8.2 Version TC2-2023.08.15

Features added

- Added support for TensorFlow Lite Machine Learning;
- Extended Docker build support to Debian TC build variant;
- Added support for the following Cortex A720/Cortex A520/Cortex X4 CPU architectural features: AFP, ECV and WFxt;
- Enabled support for Maximum Power Mitigation Mechanism (MPMM);
- Added support for GPU DVFS/Idle power states;
- GPU hardware rendering based on DDK source code compilation and pre-built binaries for Android;

1.8.3 Version TC2-2023.04.21

Features added

- Added support for EAS;
- Added support for MPAM;
- Added support for Mali-G720 GPU;
- Added support for Android Hardware Rendering.

Changes

- Updated to Android 13;
- GPU and DPU are using S1 translation with SMMU-600.

1.8.4 Version TC2-2022.12.07

Features added

- Added support for MTE3/EPAN;
- Added support for Firmware Update;
- Enabled VHE support in Hafnium to support S-EL0 partitions;
- Enabled S2 translation for GPU and DPU using SMMU-700;
- Enabled protected nVHE support for pKVM hypervisor.

1.8.5 Version TC2-2022.08.12

Features added

- Hardware Root of Trust;
- Updated Android to AOSP master;
- Microdroid based pVM support in Android.

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