Total Compute

Arm Limited

Oct 26, 2023

TOTAL COMPUTE:

CHAPTER

ONE

TOTAL COMPUTE : TC2

1.1 Total Compute Platform

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.

1.1.1 Latest TC release: TC2-2023.04.21

Total Compute Platform Software Components

RSS Firmware

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

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

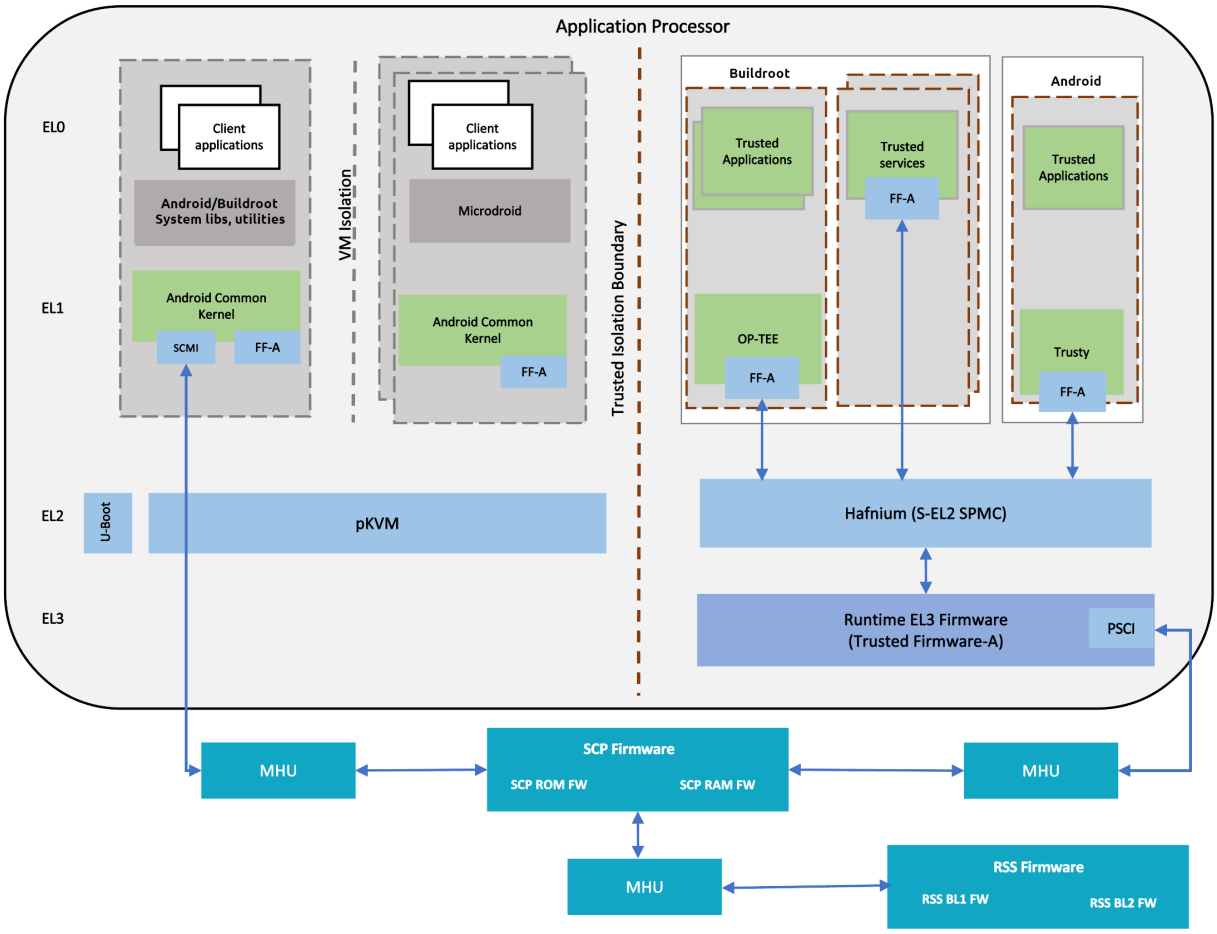
RSS initially boots from immutable code (BL1_1) in its internal ROM, before jumping to BL1_2, which is provisioned and hash-locked in RSS OTP. The updatable MCUBoot BL2 boot stage is loaded from the flash into RSS SRAM, where it is authenticated. BL2 loads and authenticates the TF-M runtime into RSS 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

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


Application Processor
EL0
EL1
EL2
EL3
Client applications
Android/Buildroot System libs, utilities
Android Common Kernel
SCMI
FF-A
VM Isolation
Client applications
Microdroid
Android Common Kernel
FF-A
Trusted Isolation Boundary
Buildroot
Trusted Applications
OP-TEE
FF-A
Trusted services
FF-A
Android
Trusted Applications
Trusty
FF-A
U-Boot
pKVM
Hafnium (S-EL2 SPMC)
Runtime EL3 Firmware (Trusted Firmware-A)
PSCI
MHU
SCP Firmware
SCP ROM FW
SCP RAM FW
MHU
MHU
RSS Firmware
RSS BL1 FW
RSS BL2 FW

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

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 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 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.

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 TTIx 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.

Instructions: Obtaining Total Compute software deliverables

- To build the TC2 software stack please refer to *user-guide*
- For the list of changes and features added please refer to *change-log*
- For further details on the latest release and features please refer to *release_notes*

TC Software Stack Overview

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

1. SCP firmware – System initialization, Clock and Power control
2. RSS firmware – 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

Total Compute Platform Software Components

User Guide

Contents

Notice

The Total Compute 2022 (TC2) software stack uses bash scripts to build a Board Support Package (BSP) and a choice of Buildroot Linux distribution or Android.

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.

To get the latest repo tool from google, 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/b

- At least 250GB 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.
- Git configured properly using "git config" otherwise it may throw error while fetching the code.

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

```
sudo apt install docker.io
# ensure docker service is properly started and running
sudo systemctl restart docker
sudo chmod 777 /var/run/docker.sock
```

NOTE: to manage Docker as a non-root user, run the following command:

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

Download the source code and build

There are two distros supported in the TC2 software stack:

- Buildroot (a minimal distro containing Busybox);
- Android.

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/TC2-2023.04.21
```

To sync Buildroot source code, run the following repo command:

```
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, run the following repo command:

```
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
```

NOTE: 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 times. 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 4, by adopting the following command `repo sync -j 4 --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 environmet 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, run the following listed commands.

For the Buildroot build:

```
export PLATFORM=tc2
export FILESYSTEM=buildroot
./setup.sh
```

For the Android build with hardware rendering:

```
export PLATFORM=tc2
export FILESYSTEM=android-fvp
export TC_GPU=true
export TC_TARGET_FLAVOR=fvp
export GPU_DDK_REPO=<PATH TO GPU DDK SOURCE CODE>
export GPU_DDK_VERSION=r40p0_01eac0
export LM_LICENSE_FILE=<LICENSE FILE>
export ARM_PRODUCT_DEF=<PATH TO ELMAP FILE IN ARMCLANG>
export ARMLMD_LICENSE_FILE=<LICENSE FILE>
export ANDROID_TEST_EXAMPLES=<PATH TO GPU DDK TEST EXAMPLES>
export ARMCLANG_TOOL=<PATH TO ARMCLANG TOOLCHAIN>
./setup.sh
```

For the Android build with software rendering:

```
export PLATFORM=tc2
export TC_GPU=false
export TC_TARGET_FLAVOR=fvp
export FILESYSTEM=android-fvp
./setup.sh
```

The various tools will be installed in the `tools/` directory at the root of the workspace.

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

```
export AVB=true
```

NOTES:

- 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).
- 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.

- 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 options

Android OS build

- tc2_fvp with `TC_GPU=false` : this supports Android display with swiftshader (software rendering);
- tc2_fvp with `TC_GPU=true` : this supports Android display with Mali GPU (hardware rendering). GPU DDK source code is available only to licensee partners (please contact support@arm.com).

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.

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

Build command

To build the whole TC2 software stack, simply run:

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

Once the previous process finishes, the current <tc2_workspace> should have the following structure:

- build files are stored in <tc2_workspace>/build-scripts/output/tmp_build/;
- final images will be placed in <tc2_workspace>/build-script/output/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.

Provided components

Firmware Components

Trusted Firmware-A

Based on Trusted Firmware-A

Script	<tc2_workspace>/build-scripts/build-tfa.sh
Files	• <tc2_workspace>/build-scripts/output/deploy/tc2/bl1-tc.bin • <tc2_workspace>/build-scripts/output/deploy/tc2/fip-tc.bin

System Control Processor (SCP)

Based on SCP Firmware

Script	<tc2_workspace>/build-scripts/build-scp.sh
Files	• <tc2_workspace>/build-scripts/output/deploy/tc2/scp_ramfw.bin • <tc2_workspace>/build-scripts/output/deploy/tc2/scp_romfw.bin

U-Boot

Based on U-Boot gitlab

Script	<tc2_workspace>/build-scripts/build-u-boot.sh
Files	• <tc2_workspace>/build-scripts/output/deploy/tc2/u-boot.bin

Hafnium

Based on Hafnium

Script	<tc2_workspace>/build-scripts/build-hafnium.sh
Files	• <tc2_workspace>/build-scripts/output/deploy/tc2/hafnium.bin

OP-TEE

Based on OP-TEE

Script	<tc2_workspace>/build-scripts/build-optee-os.sh
Files	• <tc2_workspace>/build-scripts/output/tmp_build/tfa_sp/tee-pager_v2.bin

S-EL0 trusted-services

Based on Trusted Services

Script	<tc2_workspace>/build-scripts/build-trusted-services.sh
Files	• <tc2_workspace>/build-scripts/output/tmp_build/tfa_sp/crypto-sp.bin • <tc2_workspace>/build-scripts/output/tmp_build/tfa_sp/internal-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>/build-scripts/output/deploy/tc2/Image

Trusty

Based on Trusty

Script	<tc2_workspace>/build-scripts/build-trusty.sh
Files	• <tc2_workspace>/build-scripts/output/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>/build-scripts/output/deploy/tc2/tc-fitImage.bin

Android

Script	<tc2_workspace>/build-scripts/build-android.sh
Files	• <tc2_workspace>/build-scripts/output/deploy/tc2/android.img • <tc2_workspace>/build-scripts/output/deploy/tc2/ramdisk_uboot.img • <tc2_workspace>/build-scripts/output/deploy/tc2/system.img • <tc2_workspace>/build-scripts/output/deploy/tc2/userdata.img • <tc2_workspace>/build-scripts/output/deploy/tc2/boot.img (AVB only) • <tc2_workspace>/build-scripts/output/deploy/tc2/vbmeta.img (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.

Obtaining the TC2 FVP

The TC2 FVP is available to partners for build and run on Linux host environments. Please contact Arm to have access (support@arm.com).

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]
-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
-e, --extra-model-params        [OPTIONAL] extra model parameters
```

Running Buildroot

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

Running Android

For running Android with AVB disabled:

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

For running Android with AVB enabled:

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

For running Android with hardware rendering enabled:

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d android-fvp -e '--plugin=
↪<crypto.so>'
```

NOTE: `crypto.so` is part of your FVP bundle.

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/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, 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 then Linux and Buildroot/Android.

When booting Buildroot, the model will boot Linux and present a login prompt on terminal_uart_ap. Login using the username `root`. 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 Android home screen.

When booting Android with Android Verified Boot (`AVB=true`), the GUI window will display an error, as illustrated in the related *Total Compute Platform Expected Test Results* document section. This is expected with the current TC release.

Running sanity tests

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 ilustrated in the related *Total Compute Platform Expected Test Results* document section.

Trusted Services and Client application

For Trusted Services, 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 ilustrated 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 ilustrated 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, run the following command:

```
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 ilustrated 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 be run individually also.

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

NOTE 1: KSM driver is not a part of the TC2 kernel. Hence, one of the MTE Kselftests will fail for `check_ksm_options` test.

NOTE 2: This test is specific to Buildroot only. An example of the expected test result for this test is ilustrated 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 ilustrated in the related *Total Compute Platform Expected Test Results* document section.

BTI

To run the BTI unit test, navigate to <tc2_workspace> and run:

```
adb connect 127.0.0.1:5555
cd <tc2_workspace>/src/android/out/target/product/tc_fvp/testcases/bti-unit-tests/arm64
adb push bti-unit-tests /data/local/tmp
cd <tc2_workspace>/src/android/out/target
adb push ./product/tc_fvp/obj/SHARED_LIBRARIES/libbti_basic_function_intermediates/
↪libbti_basic_function.so /data/local/tmp
```

On the `terminal_uart_ap` run:

```
cd /data/local/tmp
./bti-unit-tests
```

NOTE: This test is specific to Android builds with hardware rendering configuration enabled (i.e. *TC_GPU=true*). An example of the expected test result for this test is ilustrated in the related *Total Compute Platform Expected Test Results* document section.

MTE

To run the MTE unit test, navigate to <tc2_workspace> and run:

```
adb connect 127.0.0.1:5555
cd <tc2_workspace>/src/android/out/target/product/tc_fvp/testcases/mte-unit-tests/arm64
adb push mte-unit-tests /data/local/tmp
```

On the `terminal_uart_ap` run:

```
cd /data/local/tmp
./mte-unit-tests
```

NOTE: This test is specific to Android builds with hardware rendering configuration enabled (i.e. *TC_GPU=true*). An example of the expected test result for this test is ilustrated in the related *Total Compute Platform Expected Test Results* document section.

PAUTH

To run the PAUTH unit test, navigate to <tc2_workspace> and run:

```
adb connect 127.0.0.1:5555
cd <tc2_workspace>/src/android/out/target/product/tc_fvp/testcases/pauth-unit-tests/arm64
adb push pauth-unit-tests /data/local/tmp
```

On the terminal_uart_ap run:

```
cd /data/local/tmp
./pauth-unit-tests
```

NOTE: This test is specific to Android builds with hardware rendering configuration enabled (i.e. *TC_GPU=true*). An example of the expected test result for this test is ilustrated 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 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>
```

For FVP with buildroot, boot the FVP model to buildroot as you normally would making user 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
  host: localhost
  port: 8022
  username: root
  password: ""
  strict-host-check: false

  kernel:
    src: ${TC_WORKSPACE}/build-scripts/output/tmp_build/linux

    modules:
      make-variables:
```

(continues on next page)

(continued from previous page)

```
      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 ilustrated in the related *Total Compute Platform Expected Test Results* document section.

Debugging on Arm Development Studio

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

Creating a new connection

To create a new connection, please follow the next steps:

1. Select `File->New->Model Connection`;
2. Provide the name for the new `Debug Connection` and click the `next` button;
3. Click on the `Add a new model...` button;
4. Select `CADI` as the model interface and click the `next` button;
5. Select `Launch and connect to specific model`;
6. Select the location on your system containing the TC2 FVP model path and click the `Finish` button;
7. Once the import process of the model finishes, you can close the `Model Connection` window (used to add the new model).

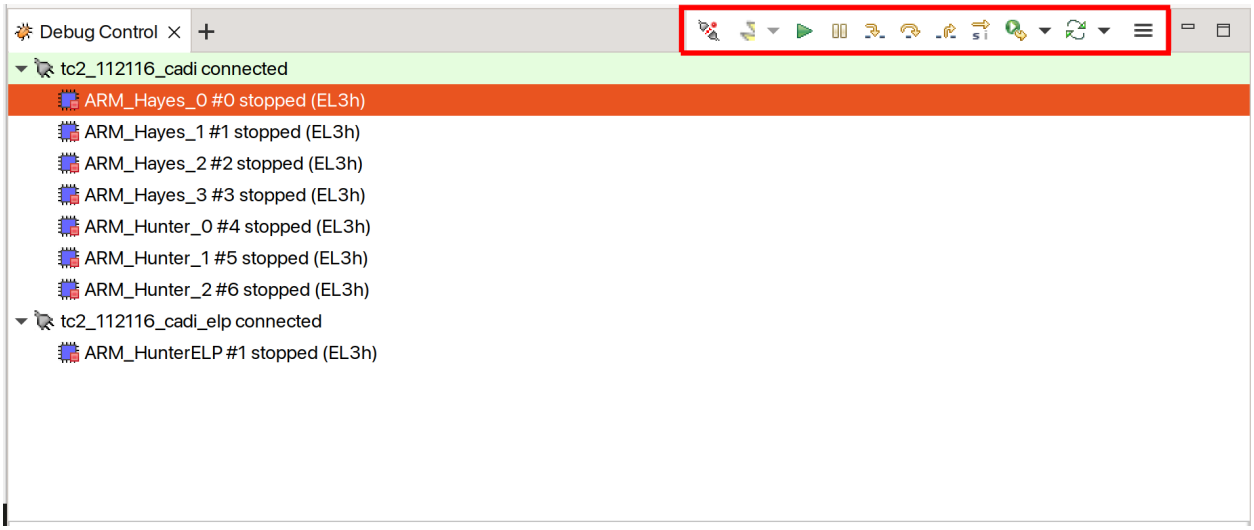
Attach and Debug

1. Build the target with debug enabled (the file `<tc2_workspace>/build-scripts/config` can be configured to enable debug);
2. Run Buildroot/Android as described in the section `Running the software on FVP` with the extra parameters `-e -S` to attach to the debugger. The full command should look like the following:

```
./run-scripts/tc2/run_model.sh -m <model binary path> -d <buildroot|android-
↪fvp> -e -S
```

3. Select the target created as mentioned in `Creating a new connection` and `connect to target` from debug control console.
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.

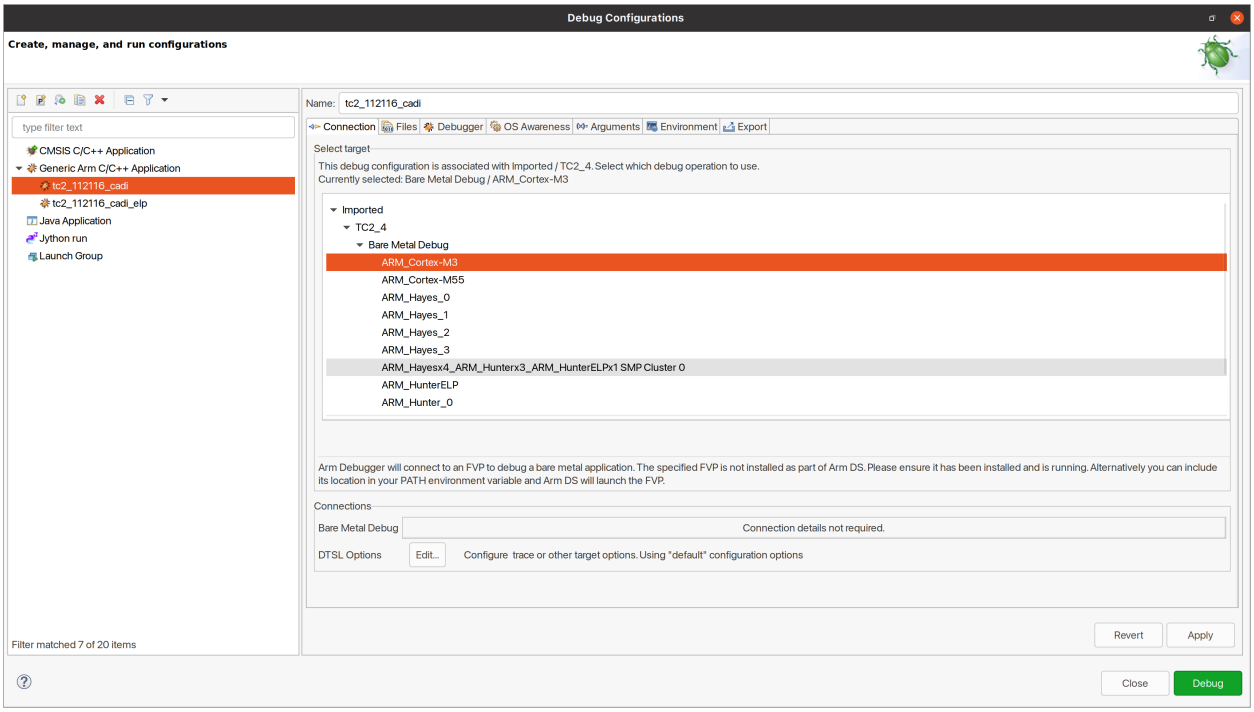
6. Debug options such as `break points`, `variable watch`, `memory view` and so on can be used.



NOTE: There is a known issue in connecting all AP cores together. The Hunter ELP core is missing from the cluster view. As a workaround, you can create two target connections as described in the `Creating a new connection` section: one for ELP core alone and the other one for the rest of AP cores.

Switch between SCP and AP

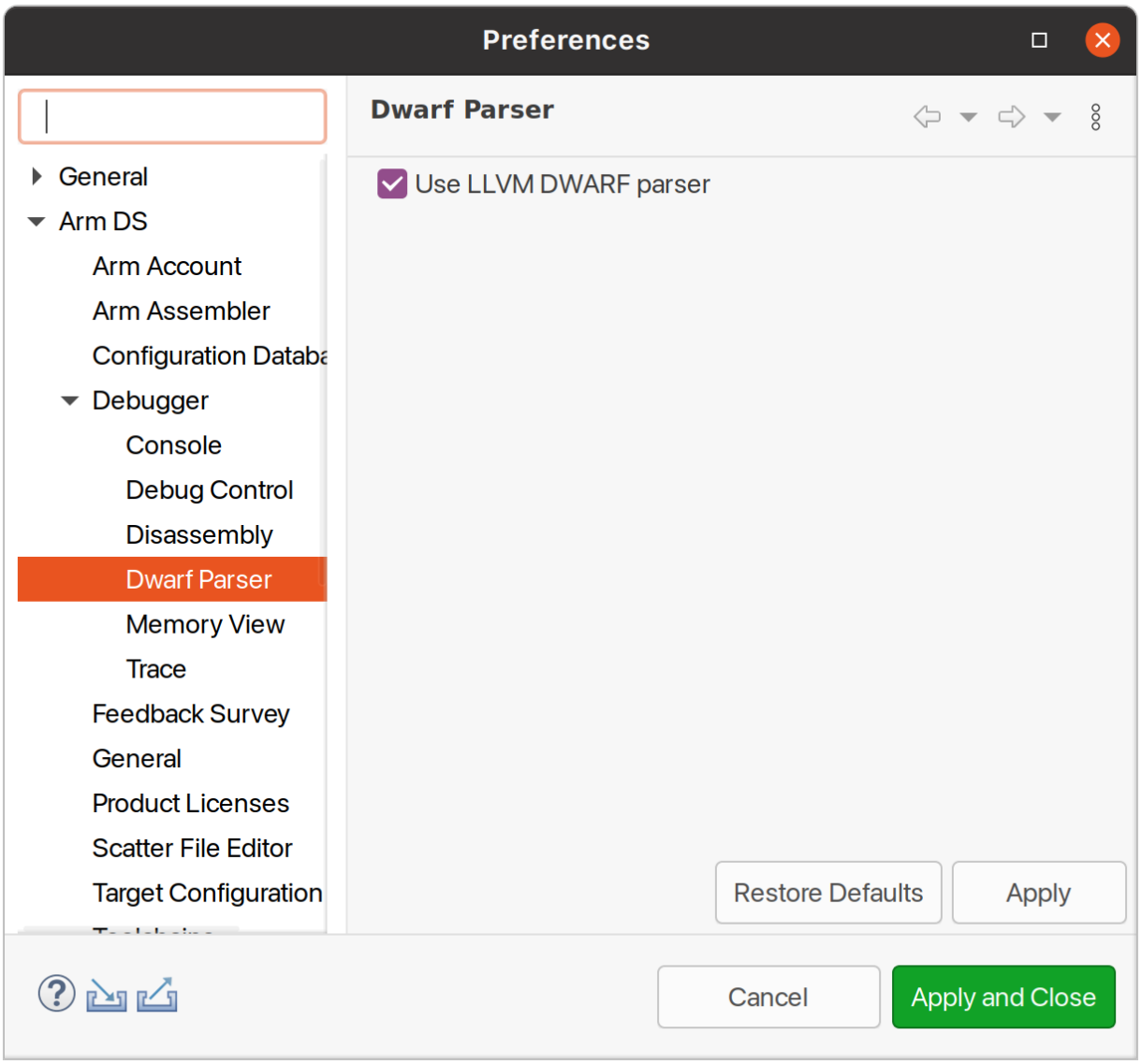
1. Right click on target and select `Debug Configurations`;
2. Under `Connection`, select `Cortex-M3` for SCP and `Arm-Hayes_x/Arm-Hunter_x` for AP core x and then debug.



Enable LLVM parser (for Dwarf5 support)

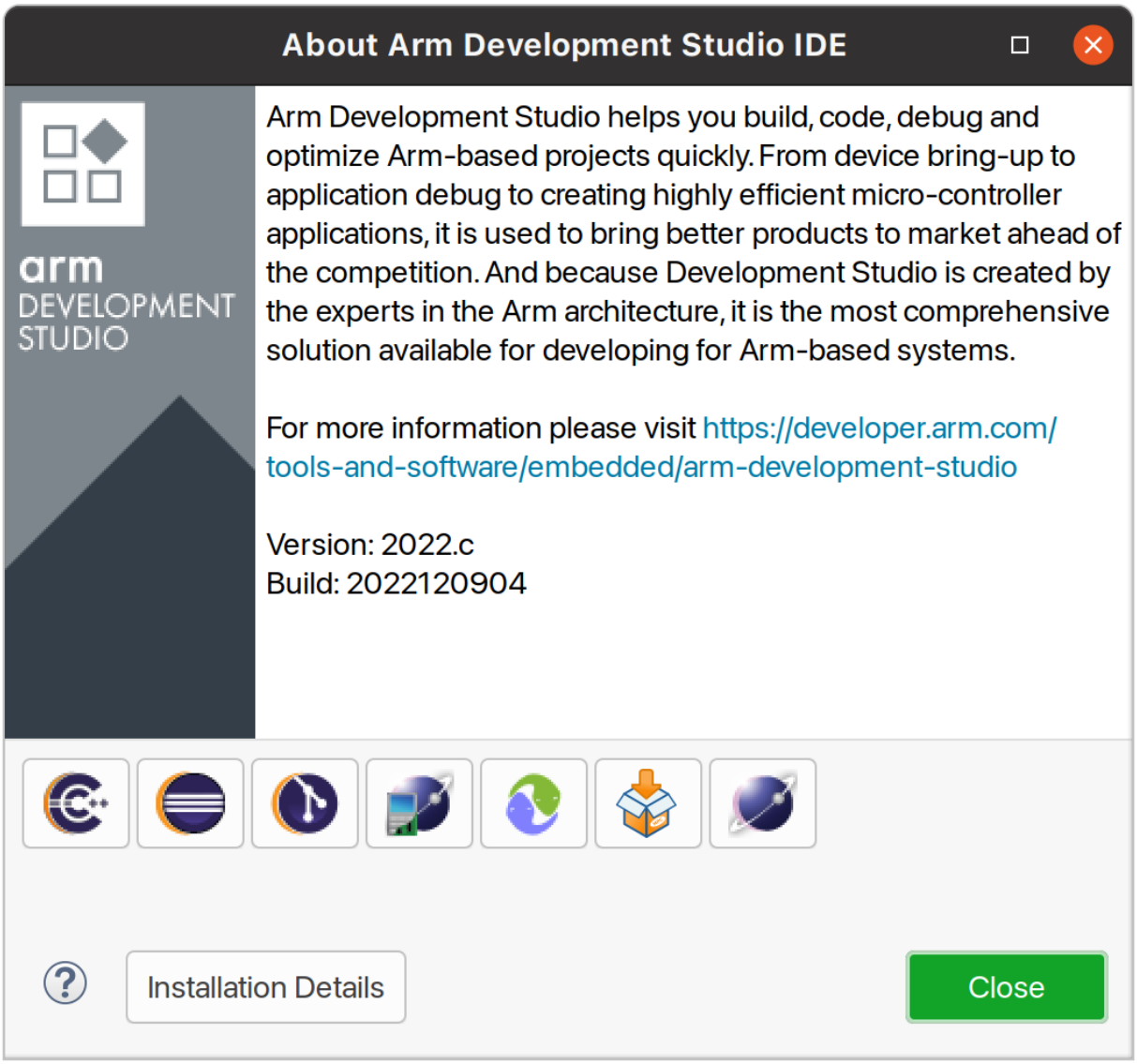
To enable LLVM parser (with Dwarf5 support), please follow the next steps:

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).

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

Command arguments 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 -e "--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.

Expected test results

Contents

OP-TEE unit tests

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

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

* 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
```

(continues on next page)

(continued from previous page)

```
104 test cases of which 0 failed
0 test cases were skipped
TEE test application done!
#
```

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
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
```

(continues on next page)

(continued from previous page)

```
            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
                    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
```

(continues on next page)

(continued from previous page)

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

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
```

(continues on next page)

(continued from previous page)

```
[       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
[ 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.
```

(continues on next page)

(continued from previous page)

```
[  PASSED  ] 28 tests.
[ DISABLED ] 2 tests.
console:/ #
```

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
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
```

(continues on next page)

(continued from previous page)

```
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)
[    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
```

(continues on next page)

(continued from previous page)

```
[    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
↪"
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.
```

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
```

(continues on next page)

(continued from previous page)

```
# 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
#
```

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

#
```

BTI unit tests

```
console:/data/local/tmp # ./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
```

(continues on next page)

(continued from previous page)

```
[       OK ] BLRAB_Test.NonGuardedMemoryAnyRegister (0 ms)
[----------] 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.
```

MTE unit tests

```
console:/data/local/tmp # ./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)

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

PAUTH unit tests

```
console:/data/local/tmp # ./pauth-unit-tests

PAC is enabled by the kernel: 1
PAC2 is implemented by the hardware: 0
FPAC is implemented by the hardware: 0
[==========] Running 18 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 2 tests from PAuthDeathTest
[ RUN      ] PAuthDeathTest.SignFailure
[       OK ] PAuthDeathTest.SignFailure (332 ms)
[ RUN      ] PAuthDeathTest.AuthFailure
Illegal instruction
```

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

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

(continues on next page)

(continued from previous page)

```
[2023-02-20 17:14:06,801][EXEKALL] INFO  Result summary:
EnergyModelWakeMigration[board=tc2]:test_dmesg
UUID=f719a77a37da4c35a287ad4f6f8fef9c PASSED: dmesg output:
EnergyModelWakeMigration[board=tc2]:test_slack
UUID=4e24d5b26b0d4020b3c2cc343082c6ac PASSED: emwm_0-0 delayed
activations: 1.3972055888223553 %
EnergyModelWakeMigration[board=tc2]:test_task_placement
UUID=aed8627987f043969c1f76ae4254e2f3 PASSED
  energy threshold: 7728.922049366228 bogo-joules
  estimated energy: 7132.289772873224 bogo-joules
  noisiest task:
    comm: kworker/5:1
    duration (abs): 0.0006251400118344463 s
    duration (rel): 0.007751385789064716 %
    pid: 69

OneSmallTask[board=tc2]:test_dmesg
UUID=8feff89476b549c5b6eeaccabc1f9ecf PASSED: dmesg output:
OneSmallTask[board=tc2]:test_slack
UUID=aad102f781334c6a80c714fc30ceb1bd PASSED: small-0 delayed activations:
0.0 %
OneSmallTask[board=tc2]:test_task_placement
UUID=8e1fd6314d644a91be77824e61dd15e6 PASSED
  energy threshold: 60.32889497785198 bogo-joules
  estimated energy: 57.45609045509712 bogo-joules
  noisiest task:
    comm: init
    duration (abs): 0.00016386000061174855 s
    duration (rel): 0.016518059968414968 %
    pid: 1

RampDown[board=tc2]:test_dmesg
UUID=496f3d737eec4c8e81e1cdc96aa12982 PASSED: dmesg output:
RampDown[board=tc2]:test_slack
UUID=b788b6a6f2644e1e9a345466647c485c PASSED: down-0 delayed activations:
0.2145922746781116 %
RampDown[board=tc2]:test_task_placement
UUID=814a50bd7fac46bd80fe8e99f1a94892 PASSED
  energy threshold: 5075.673823290201 bogo-joules
  estimated energy: 4476.446074247863 bogo-joules
  noisiest task:
    comm: kworker/5:1
    duration (abs): 0.0005229099842836149 s
    duration (rel): 0.0070281984591597825 %
    pid: 69

RampUp[board=tc2]:test_task_placement
UUID=34733c655ac54a27bb11dad502fe42eb PASSED
  energy threshold: 4511.993708928746 bogo-joules
  estimated energy: 3824.991676713266 bogo-joules
  noisiest task:
    comm: kworker/5:1
```

(continues on next page)

(continued from previous page)

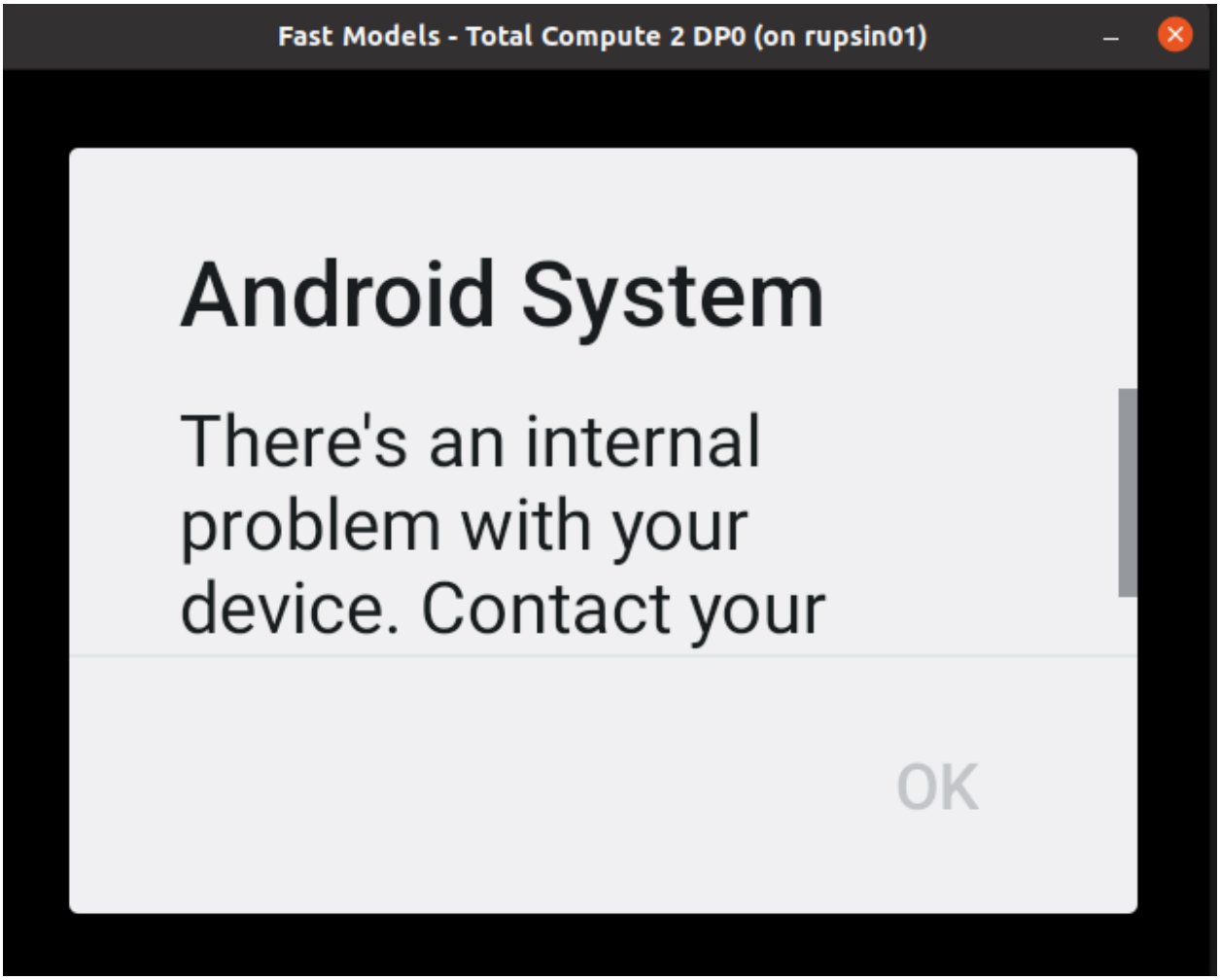
```
    duration (abs): 0.0005214800185058266 s
    duration (rel): 0.007009150123468466 %
    pid: 69

ThreeSmallTasks[board=tc2]:test_dmesg
UUID=13dbafe99b5b46d7bf4e0fe111b3ed78 PASSED: dmesg output:
ThreeSmallTasks[board=tc2]:test_slack
UUID=97567be4278f4f1188f36c3ad3ab9676 PASSED
  small_0-0 delayed activations: 0.0 %
  small_1-1 delayed activations: 0.0 %
  small_2-2 delayed activations: 0.0 %

ThreeSmallTasks[board=tc2]:test_task_placement
UUID=b518bced8a544ddd902f0c254ba9b7da PASSED
  energy threshold: 206.83944850784022 bogo-joules
  estimated energy: 172.36620708986686 bogo-joules
  noisiest task:
    comm: init
    duration (abs): 0.00016275000234600157 s
    duration (rel): 0.016406104698468295 %
    pid: 1

TwoBigTasks[board=tc2]:test_dmesg
UUID=52277de98f434f94a72d656a673339af PASSED: dmesg output:
TwoBigTasks[board=tc2]:test_slack
UUID=eb9e4cf743204830803111c56fc41787 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=tc2]:test_task_placement
UUID=76412365246741039321f2f3c0908de5 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=tc2]:test_dmesg
UUID=d7fe6333c65c44b7b801c68d4b9df74e PASSED: dmesg output:
TwoBigThreeSmall[board=tc2]:test_slack
UUID=b404c2db13064d3e9f596037159b920f 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=tc2]:test_task_placement
UUID=698d7b4fb84448248caa16db69af185f SKIPPED: skipped-reason: The
workload will result in overutilized status for all possible task
placement, making it unsuitable to test EAS on this platform
```

Booting Android with AVB



Release notes - 2023.04.21

Contents

Release tag

The manifest tag for this release is TC2-2023.04.21

Components

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

- BSP build supporting Android and Buildroot distro.
- 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.

Hardware Features

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

Software Features

- Buildroot distribution support.
- Android 13 support.
- Android Common Kernel 5.15.41
- Android Hardware Rendering with Mali TTIx GPU - DDK r40p0_01eac0
- Android Software rendering with DRM Hardware Composer offloading composition to Mali D71 DPU.
- KVM default mode of operation is set to `protected`, 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 are using S1 translation with SMMU-600
- Support for MPAM - https://developer.arm.com/documentation/107768/0100/Arm-Memory-System-Resource-Partitioning-and-Monitoring–MPAM–Extension
- Support for EAS - https://community.arm.com/oss-platforms/w/docs/530/energy-aware-scheduling-eas
- Trusted Firmware-A v2.8
- Hafnium v2.7
- OP-TEE 3.20.0
- Trusty with FF-A messaging - FF-A v1.0
- CI700-PMU enabled for profiling
- Support secure boot based on TBBR specification https://developer.arm.com/documentation/den0006/latest
- System Control Processor (SCP) firmware v2.11
- Runtime Security Subsystem (RSS) firmware v1.7.0
- Build system based on scripts which improves build times compared to Yocto
- U-Boot bootloader v2022.04
- 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 - Added support for ETE and TRBE v1.0 in TF-A, kernel and simpleperf. Traces can be captured with simpleperf. However, to enable tracing, the libete plugin has to be loaded while executing the FVP with `--plugin <path to plugin>/libete-plugin.so`

- Firmware update support.

Platform Support

- This software release is tested on TC2 Fast Model platform (FVP) version 11.21.20.

Tools Support

- This software release introduces docker support (at the moment supporting only the "image building process").

Known issues or Limitations

1. At the U-Boot prompt press enter and type "boot" to continue booting else wait for ~15 secs for boot to continue automatically. This is because of the time difference in CPU frequency and 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

Support

For support email: support-arch@arm.com

Change Log

Contents

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

Version 2023.04.21

Features added

- Added support for EAS
- Added support for MPAM
- Added support for Mali TTIx GPU
- Added support for Android Hardware Rendering

Changes

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

Version 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

Version 2022.08.12

Features added

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

CHAPTER

TWO

OLD RELEASES

2.1 Total Compute Platform

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.

2.1.1 Latest TC release

TC2-2023.04.21

2.1.2 TC2 release tags

TC2-2022.12.07

TC2-2022.08.12

2.1.3 TC1 release tags

TC1-2022.10.07

TC1-2022.05.12

TC1-2021.08.17

2.1.4 TC0 release tags

TC0-2022.02.25

TC0-2021.07.31

TC0-2021.04.23

TC0-2021.02.09