Total Compute

Arm Limited

Sep 11, 2024

TOTAL COMPUTE:

1	Total	Comput	11 12 12 12 12 12 12 12 12 12 12 12 12 1
	1.1	Total Co	ompute Platform Software Components
		1.1.1	RSE Firmware 1
		1.1.2	SCP Firmware
		1.1.3	AP Secure World Software
		1.1.4	AP Non-Secure World Software
	1.2	Instructi	ons: Obtaining Total Compute software deliverables
	1.3	TC Soft	ware Stack Overview
	1.4	User Gu	iide
		1.4.1	Notice
		1.4.2	Prerequisites
		1.4.3	Download the source code and build
		1.4.4	Provided components
		1.4.5	Obtaining the TC3 FVP
		1.4.6	Running the software on FVP
		1.4.7	Running sanity tests
		1.4.8	Debugging on Arm Development Studio
		1.4.9	Feature Guide
	1.5	System	profiling, Applications tracing and Trace analysis
		1.5.1	Simpleperf
		1.5.2	Perf
		1.5.3	Perfetto
	1.6	Security	67
		1.6.1	Assumptions and Delegated Mitigations
	1.7	Expecte	d test results
		1.7.1	SCMI unit tests
		1.7.2	TF-A unit tests
		1.7.3	TF-M unit tests
		1.7.4	OP-TEE unit tests
		1.7.5	Trusted Services and Client application unit tests
		1.7.6	Trusty unit tests
		1.7.7	Microdroid Demo unit tests
		1.7.8	Kernel selftest unit tests
		1.7.9	Rotational scheduler unit tests
		1.7.10	MPAM unit tests
		1.7.11	MPMM unit tests
		1.7.12	BTI unit tests
		1.7.13	MTE unit tests
		1.7.14	PAUTH unit tests
		1.7.15	EAS with Lisa unit tests

		1.7.16	CPU hardware capabilities	95
		1.7.17	GPU GLES Integration tests	95
		1.7.18	GPU EGL Integration tests	98
	1.8	Troubles	shooting: common problems and solutions	00
		1.8.1	Docker	01
	1.9	Release	notes - TC23.1	01
		1.9.1	Release tag	02
		1.9.2	Platform Support	02
		1.9.3	Components	02
		1.9.4	Hardware Features	02
		1.9.5	Software Features	.03
		1.9.6	Tools Support	04
		1.9.7	Optimizations	04
		1.9.8	Limitations	04
		1.9.9	Known issues	04
		1.9.10	Support	08
2	Previ	ious relea	ases 1	.09
	2.1	TC2 rele	ease tags	09
	2.2	TC1 rele	ease tags	.09
	2.3	TC0 rele	ease tags	09

CHAPTER

TOTAL COMPUTE: TC23.1

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 RSE Firmware

Runtime Security Engine (RSE) - previously known as Runtime Security SubSystem (RSS) - serves as the Root of Trust for the Total Compute platform.

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

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

The following diagram illustrates the certificate structure adopted in the TC platform:

Considering the previous diagram, the boxes do indicate the certificates, while the arrows do indicate the parent-child relationships (who loads who).

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-M85 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

It performs the following functions:

- 1. Sets up generic timer, UART console and clocks
- 2. Powers ON primary AP CPU
- 3. Responds to SCMI messages via MHUv3 for CPU power control and DVFS
- 4. Power Domain management
- 5. Clock management

1.1.3 AP Secure World Software

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

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. EL3 Runtime Software (BL31 image)
- 2. Secure Partition Manager (BL32 image)
- 3. Non-Trusted firmware U-boot (BL33 image)
- 4. Secure Partitions images (OP-TEE and Trusted Services, or Trusty)

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. System Control and Management Interface (SCMI)
- 3. Secure Monitor framework
- 4. 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, Trusted Services and Trusty 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 S-EL2 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 and Internal Trusted Storage 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 MHUv3 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

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.

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 Swift-Shader 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-Drage GPU model is 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.

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

1.2 Instructions: Obtaining Total Compute software deliverables

- To build the TC3 software stack, please refer to the *user guide*;
- For further details on the latest release and features, please refer to the *release notes*;

1.3 TC Software Stack Overview

The TC3 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. RSE (previously known as 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.

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

1.4 User Guide



- * Build command
- * More about the build system
- * Build component requirements
- Provided components
 - * Firmware and Software Components
 - Runtime Security Engine (RSE)
 - System Control Processor (SCP)
 - Trusted Firmware-A
 - U-Boot
 - Hafnium
 - \cdot OP-TEE
 - · S-EL0 trusted-services
 - · Trusty
 - Linux
 - * Distributions
 - Buildroot Linux distro
 - · Debian Linux distro
 - · Android
 - * Run scripts
- Obtaining the TC3 FVP
- Running the software on FVP
 - * Running Buildroot
 - * Running Debian
 - * Running Android
 - · Android general common run command
 - · Android with AVB enabled
 - * Expected behaviour
- Running sanity tests
 - * SCMI
 - * *TF-A*
 - * *TF-M*
 - * Validate the TensorFlow Lite ML flow
 - Prerequisites
 - *Manually uploading a TensorFlow Lite ML model for Buildroot or application for Debian distro*

- Manually uploading a TensorFlow Lite ML model, Arm Neural Network and application for Android • Running the provided TensorFlow Lite ML model examples * OP-TEE * Trusted Services and Client application * Trusty * Microdroid · Prerequisites · Run Microdroid demo • Run Microdroid instance · Connect to Microdroid instance with ADB * Kernel Selftest * Rotational Scheduler * MPAM * MPMM * BTI * *MTE* * PAUTH * EAS with LISA * pKVM SMMUv3 driver support validation * CPU hardware capabilities * GPU Integration • Initial Setup · Running GLES integration tests · Running EGL integration tests · Running Vulkan integration tests - Debugging on Arm Development Studio * Attach and Debug * Switch between SCP and AP * Enable LLVM parser (for Dwarf5 support) * Arm DS version - Feature Guide * Firmware Update · Creating Capsule · Loading Capsule
 - Updating Firmware

* AutoFDO in Android
Prerequisites
Steps to use AutoFDO
* ADB connection on Android
Connect to the running FVP-model instance
• Upload a file
• Download a file
Execute a remote command
* Set up TAP interface for Android ADB
• Steps to set up the tap interface
• Steps to graceful disable and remove the tap interface
* Running and Collecting FVP tracing information
• Getting the list of trace sources
• Executing the FVP-model with traces enabled
* DICE/DPE
Verify DPE from U-boot
Verify DPE from Microdroid

1.4.1 Notice

The Total Compute 2023 (TC3) 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 TC3 Fast Model platform (FVP) version 11.26.16.

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 64 GB of RAM. Lower amounts may lead to build failures due to out-of-memory (OOM).

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

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

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

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

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

sudo usermod -aG docker \$USER
newgrp docker

1.4.3 Download the source code and build

The TC3 software stack supports the following distros:

- Buildroot (a minimal distro containing Busybox);
- Debian (based on Debian 12 Bookworm);
- Android (both Android 14 and Android 13).

Download the source code

To download **Buildroot or Debian source code**, please define the following environment variable:

export REPO_TARGET=bsp

To download Android source code (a superset of bsp), please define the following environment variable:

```
export REPO_TARGET=android
```

Independently of the distribution to be built, create a new folder that will be your workspace (which will henceforth be referred to as <TC_WORKSPACE> in these instructions) and start the cloning code process by running the following commands:

If cloning Android, this is expected to take a very long time. Once this finishes, the current <TC_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;
- tests/: different test suites.

Note: The manifest file tc3_a14.xml is for Android 14, which is officially supported in TC23. To checkout code for Android 13, use tc3_a13.xml instead. Only minimum test is done for Android 13.

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 <TC_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:

ound distros	
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)

Table 1: List of GPU rendering variants available for Android OS based build distros

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

Android Verified Boot (AVB)

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 setup the environment to build the Buildroot distro, please run the following commands:

export PLATFORM=tc3
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 setup the environment to build the Debian distro, please run the following commands:

export PLATFORM=tc3
export FILESYSTEM=debian
export TC_TARGET_FLAVOR=fvp
cd build-scripts
./setup.sh

Android build

Note: Android SDK, which is required to build the benchmark_model application for Android, has its standalone terms and conditions. These terms and conditions are automatically accepted during Android SDK installation process and can be found in link.

By default, the Android image is built with Android Verified Boot (AVB) disabled. To override this setting and build Android with 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=tc3
export FILESYSTEM=android
export TC_ANDROID_VERSION=android14
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=tc3
export FILESYSTEM=android
export TC_ANDROID_VERSION=android14
export TC_GPU=hwr
export TC_TARGET_FLAVOR=fvp
export GPU_DDK_REPO=<PATH TO GPU DDK SOURCE CODE>
export GPU_DDK_VERSION="releases/r49p0_00eac0"

(continues on next page)

(continued from previous page)

```
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 setup the environment to build the Android distro with software rendering, please run the following commands:

```
export PLATFORM=tc3
export FILESYSTEM=android
export TC_ANDROID_VERSION=android14
export TC_GPU=swr
export TC_TARGET_FLAVOR=fvp
cd build-scripts
./setup.sh
```

Warning: If building the TC3 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 TC3 software stack for any of the supported distros, simply run:

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

The output directory (henceforth referred to as <TC_OUTPUT>) is <TC_WORKSPACE>/output/<\$PLATFORM>/ <\$FILESYSTEM>/<\$TC_TARGET_FLAVOR>.

Once the previous process finishes, <TC_OUTPUT> will have two subdirectories:

- tmp_build/ storing individual components' build files;
- deploy/ storing the final images.

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 clean, build, deploy and patch commands. The build-all.sh script also supports all, which performs a clean followed by a rebuild of all the stack.

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

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

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 \
    -a $AVB \
    -t $TC_TARGET_FLAVOR \
    -g $TC_GPU build
```

Build component requirements

The list of requirements of a specific component can be modified by editing the build_requirements.txt file. When building a specific component, both the component and the requirements specified after the equal sign will be sequentially rebuilt, considering current environment variables.

To activate this feature, use the with_reqs option appended to the desired component build command, as illustrated in the following example:

./run_docker.sh ./build-scp.sh clean build with_reqs

The with_reqs functionality adheres to the specific details mentioned above for build-all.sh.

1.4.4 Provided components

Firmware and Software Components

Runtime Security Engine (RSE)

Based on Runtime Security Engine

Script	<tc_workspace>/build-scripts/build-rse.sh</tc_workspace>	
Files	 <tc_output>/deploy/rse_encrypted_cm_provis</tc_output> <tc_output>/deploy/rse_encrypted_dm_provis</tc_output> <tc_output>/deploy/rse_rom.bin</tc_output> 	ioning_bundle_0 ioning_bundle_0

System Control Processor (SCP)

Based on SCP Firmware

Script	<tc_workspace>/build-scripts/build-scp.sh</tc_workspace>
Files	 <tc_output>/deploy/scp-boot.bin</tc_output> <tc_output>/deploy/scp-runtime.bin</tc_output>

Trusted Firmware-A

Based on Trusted Firmware-A

Script	<tc_workspace>/build-scripts/build-tfa.sh</tc_workspace>
Files	 <tc_output>/deploy/bl1-tc.bin</tc_output> <tc_output>/deploy/fip-tc.bin</tc_output> <tc_output>/deploy/fip_gpt-tc.bin</tc_output>

U-Boot

Based on U-Boot

Script	<tc_workspace>/build-scripts/build-u-boot.sh</tc_workspace>
Files	• <tc_output>/deploy/u-boot.bin</tc_output>

Hafnium

Based on Hafnium

Script	<tc_workspace>/build-scripts/build-hafnium.sh</tc_workspace>
Files	• <tc_output>/deploy/hafnium.bin</tc_output>

OP-TEE

Based on OP-TEE

Script	<tc_workspace>/build-scripts/build-optee-os.sh</tc_workspace>
Files	 <tc_output>/tmp_build/tfa_sp/tee- pager_v2.bin</tc_output>

S-EL0 trusted-services

Based on Trusted Services

Script	<tc_workspace>/build-scripts/build-trusted- services.sh</tc_workspace>
Files	 <tc_output>/tmp_build/tfa_sp/crypto.bin</tc_output> <tc_output>/tmp_build/tfa_sp/internal- trusted-storage.bin</tc_output> <tc_output>/tmp_build/tfa_sp/fwu.bin</tc_output>

Trusty

Based on Trusty

Script	<tc_workspace>/build-scripts/build-trusty.sh</tc_workspace>
Files	• <tc_output>/tmp_build/tfa_sp/lk.bin</tc_output>

Linux

The component responsible for building a 6.1 version of the Android Common kernel (ACK).

Script	<tc_workspace>/build-scripts/build-linux.sh</tc_workspace>
Files	• <tc_output>/deploy/Image</tc_output>

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	<tc_workspace>/build-scripts/build-buildroot.sh</tc_workspace>
Files	• <tc_output>/deploy/tc-fitImage.bin</tc_output>

Debian Linux distro

Script	<tc_workspace>/build-scripts/build-debian.sh</tc_workspace>
Files	<tc_output>/deploy/debian.img</tc_output><tc_output>/deploy/debian_fs.img</tc_output>

Android

Script	<tc_workspace>/build-scripts/build-android.sh</tc_workspace>
Files	 <tc_output>/deploy/android.img</tc_output> <tc_output>/deploy/ramdisk_uboot.img</tc_output> <tc_output>/deploy/system.img</tc_output> <tc_output>/deploy/userdata.img</tc_output> <tc_output>/deploy/vendor.img</tc_output> <tc_output>/deploy/boot.img (AVB only)</tc_output> <tc_output>/deploy/vbmeta.img (AVB only)</tc_output>

Run scripts

Within the <TC_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 TC3 FVP

To download the latest available TC3 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 TC3 platform must be available to run the included run scripts.

The run-scripts structure is as follows:

```
run-scripts
|--tc3
    |--run_model.sh
|-- ...
```

Ensure that all dependencies are met by running the FVP: ./path/to/FVP_TC3. 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 <TC_WORKSPACE>/run-scripts/tc3/ will launch the FVP, providing the previously built images as arguments. The following excerpt contains the command usage help retrieved when running ./ run-scripts/tc3/run_model.sh --help script:

```
$ ./run-scripts/tc3/run_model.sh --help
<path_to_run_model.sh> [OPTIONS]
REOUIRED OPTIONS:
-m, --model MODEL
                                  path to model
-d, --distro {buildroot|android|debian}
                                  distro version
OPTIONAL OPTIONS
                                  avb boot, DEFAULT: false
-a, --avb {true|false}
-t, --tap-interface
                                  tap interface
-n, --networking {user|tap|none} networking
                                  DEFAULT: tap if tap interface provided, otherwise user
    --debug {iris|cadi|none}
                                  start a debug server, print the port listening on,
                                  and wait for debugger. DEFAULT: none
-v, --no-visualisation
                                  don't spawn a model visualisation window
    --telnet
                                  don't spawn console windows, only listen on telnet
-- MODEL_ARGS
                                  pass all further options directly to the model
```

Running Buildroot

./run-scripts/tc3/run_model.sh -m <model binary path> -d buildroot

Running Debian

./run-scripts/tc3/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/tc3/run_model.sh -m <model binary path> -d android

Android with AVB enabled

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

./run-scripts/tc3/run_model.sh -m <model binary path> -d android -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 RSE logs.

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 to boot Trusted Firmware-A, Hafnium, Secure Partitions (OP-TEE, Trusted Services in Buildroot and Trusty in Android) then U-Boot, and finally the root filesystem of the corresponding 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 3 DPO 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.

SCMI

This test is supported in Buildroot only. When setup the environment to build the Buildroot distro, an extra command is needed:

export SCMI_TESTS=true

before executing the script ./setup.sh. Then build and run the Buildroot distro as normal. After the FVP is up and running, on the terminal_uart_ap run:

./scmi_test_agent

The test log will be generated with file name arm_scmi_test_log.txt.

The random test failures on test cases 409, 413 and 517 is known issue.

Note: This test is specific to Buildroot only. And the manifest file tc3_a14.xml is used when checkout the code. An example of the expected test result for this test is illustrated in the related *Total Compute Platform Expected Test Results* document section.

TF-A

This test is supported in Buildroot only. After build Buildroot, run commands:

```
export TFTF_TESTS=true
./run_docker.sh build-tftf-tests.sh all with_reqs
```

Then run Buildroot as normal. The test results is on terminal_uart_ap.

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.

TF-M

After build the selected system distro, run commands:

```
export RSE_TESTS=true
./run_docker.sh build-rse.sh all with_reqs
```

Then run the selected system distro as normal. The test results is on terminal_s1.

Note: It is expected that the boot will not complete after the rse tests are run.

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

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, the following files will be required:

- benchmark_model binary: this file is part of the TC build and is automatically built;
- <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.
- armNN folder: this folder contains the files libarmnn.so, libarmnnDelegate.so, and Arm_CpuRef_backend.so; these libraries are required by TensorFlow Lite to use ArmNN as one of its backends to delegate work.

For Buildroot distro, the binaries are automatically integrated into the buildroot distro filesystem (being located at /opt/arm/ml).

For Android and Debian distro, the benchmark_model and armNN binaries have to be manually uploaded to the running TC FVP model before the test can be executed. See sections *Manually uploading a TensorFlow Lite ML model for Buildroot or application for Debian distro*, and *Manually uploading a TensorFlow Lite ML model*, Arm Neural Network and application for Android below for more details.

Manually uploading a TensorFlow Lite ML model for Buildroot or application for Debian distro

For Buildroot distro, the application and "Mobile Object Localizer" model are automatically integrated into the buildroot distro filesystem. However, there may be situations where the developer wishes to use their own TensorFlow Lite model.

For Debian distro, the application and a model have to be manually uploaded to the running TC FVP model.

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

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 upload and 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:

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

• upload the benchmark_model application 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 benchmark_model 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 in the Running the provided TensorFlow Lite ML model examples section.

Note: For Debian distro, the directory /opt/arm/ml should be created manually from uart_ap terminal. Also, the benchmark_model application is located in host directory <TC_OUTPUT>/deploy.

Manually uploading a TensorFlow Lite ML model, Arm Neural Network and application for Android

The benchmark_model application, "Arm Neural Network" backend support library and "Mobile Object Localizer" TensorFlow Lite model are not automatically integrated into the Android filesystem.

This section describes the steps necessary to manually upload these required files to the TC FVP running Android instance, and execute the test.

• start by moving to the build folder and upload the MobileNet Graph model and benchmark_model application by the following command:

```
cd <TC_OUTPUT>/deploy/
adb connect localhost:5555
adb push benchmark_model /data/local/tmp
adb push mobile_object_localizer_v1.tflite /data/local/tmp
adb push armNN /data/local/tmp
```

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

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.

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.
\rightarrow example ML models
# For Buildroot and Debian distro
cd /opt/arm/ml
# For Android
cd /data/local/tmp
# With XNNPack for CPU path
./benchmark_model --graph=mobile_object_localizer_v1.tflite \
    --num_threads=4 --num_runs=1 --min_secs=0.01 --use_xnnpack=true
# With ArmNN for GPU path (Only available for Android)
LD_LIBRARY_PATH=/vendor/lib64/egl:armNN/ \
./benchmark_model \
    --graph=mobile_object_localizer_v1.tflite \
    --num_threads=4 \setminus
    --num_runs=1 \setminus
    --min\_secs=0.01 \setminus
    --external_delegate_path="armNN/libarmnnDelegate.so" \
    --external_delegate_options="backends:GpuAcc;logging-severity:info"
```

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.

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 -g FwuServiceTests -g ItsServiceTests -g CryptoKeyDerivationServicePackedcTests -g CryptoMacServicePackedcTests -g CryptoCipherServicePackedcTests -g CryptoBervicePackedcTests -g CryptoServicePackedcTests -g CryptoServicePackedcTests -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

On the Android distribution, Virtualization service provides support to run Microdroid based pVM (Protected VM). In TC, it supports running both simple Microdroid demo and real Microdroid instance.

Prerequisites

Boot TC FVP with Android distribution to completely up. Leave it for some time (about 30 minutes) after homescreen is rendered for adbd service to work. From one host terminal, run the following commands:

```
export TC_ANDROID_VERSION=android14
export ANDROID_PRODUCT_OUT=<TC_WORKSPACE>/src/android/out/target/product/tc_fvp/
```

Note: The document below is for Android 14. android13 can be used to run the test on Android 13. There are different behaviours for Android 13. The differences will be explained end of this chapter.

Run Microdroid demo

On the same host terminal, run command:

./run-scripts/run_microdroid_demo.sh run-tc-app

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

Run Microdroid instance

On the same host terminal, run command:

./run-scripts/run_microdroid_demo.sh start-microdroid

The terminal will be pending and waiting for ADB connection to it.

Connect to Microdroid instance with ADB

There are two options using ADB to connect to Microdroid instance.

- If there is only one Microdroid instance to be run, connect to it when it starts running. Run the command:
 - ./run-scripts/run_microdroid_demo.sh start-microdroid --auto-connect
- If there is more than one Microdroid instance to be run, start the Microdroid instances firstly, then connect to them from another host terminal. Run the command:

./run_microdroid_demo.sh vm-connect <CID>

The CID for the Microdroid instance is shown when the instance starts running. Also the script will prompt the user to select between the running instances.

Note: This test is specific to Android only. The ADB connection uses the default ADB port 5555. If ADB connect failed, check the ADB port in use and make change to the script manually.

Note: There are two differences for Android 13. When using the run-tc-app command, the test is not expected to terminate immediately. This allows you to access the shell from another terminal; To access the VM shell for Microdroid, the build type must be userdebug when building Android. Accessing the VM shell with an eng build (the default build option) is not possible. To enable userdebug mode, use the command export TC_ANDROID_BUILD_TYPE=userdebug before building Android.

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 TC3 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.

Rotational Scheduler

Rotating scheduler is a vendor module in the Linux kernel that will allow to use the CPUs optimally on an asymmetric platform. Typically, on an asymmetric platform, tasks running on big CPUs will finish sooner. The resulting scheduling pattern is not optimal, little/medium CPUs are unused once the big CPUs finish their task, as the tasks running on little/medium CPUs are migrated to big CPU and little/medium CPUs will be in a idle state.

The rotating scheduler:

- Starts when one CPU reaches the Rotate state.
- Ends when there are no CPU in the Rotate state anymore.

Rotating scheduler will

- rotate task between CPUs to have all the tasks finishing approximately at the same time.
- no idle time from any CPU.

There are sysfs interface to configure rotating scheduler:

• Enable

Enable/disable the rotating scheduler.

• Max_latency_us

Keep track of the amount of work each rotating task has achieved. At any time, if the task the most ahead finishes, all the rotating tasks should finish within the next max_latency_us.

• Min_residency_us

Tasks are guaranteed a minimum residency time after a rotation. This prevents from having tasks constantly switching on a CPU. Min_residency_us is stronger than max_latency_us, meaning that min_residency_us is strictly respected and max_latency_us is a soft target.

To run the test, on the terminal_uart_ap run:

test_rotational_scheduler.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.

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

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 tc3 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

For FVP with buildroot, boot the FVP model to buildroot as you normally would, making sure user networking is enabled (include -n user option):

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

The following excerpt illustrates the contents of the target_conf_linux.yml file:

```
target-conf:
  kind: linux
  name: tc
  host: localhost
  port: 8022
```

(continues on next page)

(continued from previous page)

```
username: root
password: ""
strict-host-check: false
kernel:
   src: <TC_OUTPUT>/tmp_build/linux
   modules:
      make-variables:
      CC: clang
   build-env: alpine
wait-boot:
   enable: false
devlib:
   file-xfer: scp
   max-async: 1
```

An intermittent failure on test case TwoBigThreeSmall[board=tc]:test_task_placement is known issue.

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.iommu/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.iommu/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.iommu/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:

```
(...)
Ε
    0.033341][
                   T1] iommu: Default domain type: Translated
Ε
    0.033349][
                   T1] iommu: DMA domain TLB invalidation policy: strict mode
(...)
    0.059858][
                   T1] kvm [1]: IPA Size Limit: 40 bits
Γ
    0.068132][
                   T1] kvm-arm-smmu-v3 4002a00000.iommu: ias 40-bit, oas 40-bit
Ε
\rightarrow (features 0x0000dfef)
                   T1] kvm-arm-smmu-v3 4002a00000.iommu: allocated 65536 entries for cmdq
Г
     0.068562][
    0.068574][
                   T1] kvm-arm-smmu-v3 4002a00000.iommu: 2-level strtab only covers 23/
Γ
\rightarrow 32 bits of SID
Г
     0.070775][
                   T1] kvm-arm-smmu-v3 3f000000.iommu: ias 40-bit, oas 40-bit (features_
\rightarrow 0x0000dfef)
     0.071061][
                   T1] kvm-arm-smmu-v3 3f000000.iommu: allocated 65536 entries for cmdq
Γ
                   T1] kvm-arm-smmu-v3 3f000000.iommu: 2-level strtab only covers 23/32
Γ
     0.071071][
→bits of SID
    0.086915][
                  T69] Freeing initrd memory: 1428K
Γ
    0.094720][
                   T1] kvm [1]: GICv4 support disabled
Γ
Γ
    0.094727][
                   T1] kvm [1]: GICv3: no GICV resource entry
Γ
                   T1] kvm [1]: disabling GICv2 emulation
    0.094734]
Γ
    0.094742][
                   T1] kvm [1]: GIC system register CPU interface enabled
                   T1] kvm [1]: vgic interrupt IRQ18
Ε
    0.094803][
                   T1] kvm [1]: Protected nVHE mode initialized successfully
Ε
    0.095008][
(...)
    0.196354][
                  T69] komeda 4000000000.display: Adding to iommu group 0
Ε
(...)
                  T69] mali 2d000000.gpu: Adding to iommu group 1
Г
     3.792147][
(...)
```

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.

GPU Integration

When Android is built with the Mali DDK (hardware rendering) based on DDK source code, it supports integration tests for GLES, Vulkan and EGL. These are built by default as part of the DDK and can be run from the Android command line (aka terminal_uart_ap) once the system has booted.

The following subsection includes the common steps and commands required to run any of the GPU integration test suites.

Initial Setup

Android enforces linker namespaces based on file paths and therefore the files must be copied to an unrestricted namespace. On the terminal_uart_ap run:

su

```
mkdir -p /data/nativetest/unrestricted
cp /data/nativetest64/vendor/mali_tests64/* /data/nativetest/unrestricted/
cd /data/nativetest/unrestricted
```

To prevent potential failures during the start of tests, specify the following environment variable:

export LD_PRELOAD=/vendor/lib64/egl/libGLES_mali.so

Running GLES integration tests

On the terminal_uart_ap run:

```
./mali_gles_integration_suite
```

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

Running EGL integration tests

On the terminal_uart_ap run:

./mali_egl_integration_tests

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

Warning: Please note that, the EGL test takes considerable time to finish (approx. 2 days).

Running Vulkan integration tests

On the terminal_uart_ap run:

./mali_vulkan_integration_suite

Warning: When running the full Vulkan Integration test suite, the test vulkan_wsi_external_memory_dma_buf_32k_image is expected to fail at some point (please refer to the *Total Compute Platform Expected Test Results* for more details). To avoid facing this error or having the GPU Integration test fail, the user is highly suggested to run the tests individually.

Warning: Please note that, depending on the unitary test selection but especially considering the full Vulkan test suite, the test execution time may take quite considerable time to run (approx. 2 weeks considering the worst scenario for the full test suite).

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

- Build the target with debug enabled (the file <TC_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/tc3/run_model.sh -m <model binary path> -d <distro> -- -I

- 3. Import the model Add a new model... -> Select Model Interface -> Select Model Connection Method -> Model Running on Local Host. Change the unrecognised CPU type to A-Generic.
- 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.b or later.
□ TC3 ×								
Simulation Name :	TC3							
Manufacturer Name :	Imported				Platform Nam	e : TC3_1		
Model Devices and Clus	ster Configuration Debug Cor	nections Model La	unch Configuration	Advanced Configuration				
Executable Devices	ster eeninguration <u>seeting een</u>		garation	araneea eoringaroaori	Associations			
Instance Name	Dovice Name	Type	tor Paramotors		Drag an item fro	m the list on right	hand side and drop it into the left hand side	
css.cluster0.subclust css.cluster0.subclust css.cluster0.subclust css.cluster0.subclust css.cluster0.subclust css.cluster0.subclust css.cluster0.subclust css.cluster0.subclust css.cluster0.subclust css.cluster0.subclust	Device Name Device Name ARM_Cortex-A520 ter0.cpu1 ARM_Cortex-A520 ter1.cpu1 ARM_Cortex-A725 ter1.cpu3 ARM_Cortex-A725 ter1.cpu3 ARM_Cortex-A725 ter2.cpu0 ARM_Cortex-X925 ter2.cpu1 ARM_Cortex-X925 ARM_Cortex-M95 ARM_Cortex-M55	Type Clus Cortex-AS2 0 Acceneric 1 Cortex-MS3 3 Cortex-M55 4	er Parameters		Comparing an left in the construction of	r0.subcluster0.cg r0.subcluster1.cg r0.subcluster1.cg r0.subcluster1.cg r0.subcluster1.cg r0.subcluster1.cg r0.subcluster2.cg r0.subcluster2.cg r0.subcluster2.cg r0.subcluster2.cg r0.subcluster2.cg r0.subcluster2.cg	Filter: framework.SimulationEngine css.sms.scp.cpu.ll.cache css.sms.scp.cpu.ll.cache css.sms.scp.cpu.ll.cache css.sms.scp.cpu.ll.cache css.sms.scp.cpu.ll.cache css.sms.rse.cpu.ll.incoherent_interconnect css.sms.rse.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster0.cpu.ll.cache css.cluster0.subcluster1.cpu0.ll.cache css.cluster0.subcluster1.cpu0.ll.cache	
							css.cluster0.subcluster1.cpu1.ltcache css.cluster0.subcluster1.cpu1.ltcache css.cluster0.subcluster1.cpu1.ltcache css.cluster0.subcluster1.cpu1.ltcache css.cluster0.subcluster1.cpu2.ltdcache css.cluster0.subcluster1.cpu2.ltcache	
				* 🖉 🔶		×	css.ciusteru.subciuster1.cpu2.l2cache	
								Save Import Debug

Switch between SCP and AP

- 1. Right click on target and select Debug Configurations;
- 2. Under Connection, select Cortex-M85 for SCP or any of the remaining targets to attach to a specific AP;
- 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).

Pebug Control × +		
🧏 🤻 👻 🕨 💷 👁 🖉 👻 🛩 😂	•	≡
👻 🗽 tc3-RSE connected		
ARM_Cortex-M55 #1 stopped (Secure Privileged Thread)		
→		
ARM_Cortex-M85 #1 stopped (Secure Privileged Thread)		
→ k tc3-cluster0 connected		
ARM_Cortex-A520_0 #0 stopped (EL2h)		
ARM_Cortex-A520_1 #1 powered down		
→		
ARM_Cortex-A725_0 #0 powered down		
ARM_Cortex-A725_1 #1 powered down		
ARM_Cortex-A725_2 #2 powered down		
ARM_Cortex-A725_3 #3 powered down		
→		
ARM_Cortex-X925_0 #0 powered down		
ARM_Cortex-X925_1 #1 powered down		
Status: connected		

Name: tc3-cluster0
🗇 Connection 🔚 Files 👫 Debugger 🦃 OS Awareness 🕺 Arguments 🚾 Environment 🛃 Export
Select target
Select the manufacturer, board, project type and debug operation to use.
Currently selected: Imported / TC3_1 / Bare Metal Debug / ARM_Cortex-M85
Filter platforms
P 1UZ_Z
▶ TC3
▼ TC3_1
✓ Bare Metal Debug
ARM_Cortex-A520_0
ARM_Cortex-A520_1
ARM_Cortex-A520x2 SMP Cluster 0
ARM_Cortex-A725_0
ARM_Cortex-A725_1
ARM_Cortex-A725_2
ARM_Cortex-A725_3
ARM_Cortex-A725x4 SMP Cluster 1
ARM_Cortex-M55
ARM_Cortex-M85
ARM_Cortex-X925_0
ARM_Cortex-X925_1
ARM_Cortex-X925x2 SMP Cluster 2
Linux Kernel Debug





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. Clone the edk2 project with the command:

git clone --depth 1 --branch edk2-stable202405 https://github.com/tianocore/edk2.git

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

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:

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 reset command:

TOTAL_COMPUTE# reset

Note: The new Firmware Update solution is compatible with the latest TF-A and aligns with the PSA Firmware update spec.

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:

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):

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 3 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
# 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> ${\hookrightarrow}{<}local host location where to save file>$

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 $\mbox{-s}\ \mbox{-sv}\ \mbox{shell}\ \mbox{ls}\ \mbox{-la}$

There is a script adb_verify.sh under TC directory build-scripts/unit_test. It can be used to test all adb commands on TC Android.

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-{\scriptstyle \hookrightarrow}{\sf 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 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)
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/tc3/run_model.sh -m <model binary path> -d android -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 3 DPO 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:

console:/ lo	<pre># ifconfig Link encap:Local Loopback inet addr:127.0.0.1 Mask:255.0.0.0 inet6 addr: ::1/128 Scope: Host UP LOOPBACK RUNNING MTU:65536 Metric:1 RX packets:27 errors:0 dropped:0 overruns:0 frame:0 TX packets:27 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:1800 TX bytes:1800</pre>
∃ummyO	Link encap:Ethernet HWaddr Contract Scope: Link UP BROADCAST RUNNING NOARP MTU:1500 Metric:1 RX packets:0 errors:0 dropped:0 overruns:0 frame:0 TX packets:50 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:0 TX bytes:9189
eth0	Link encap:Ethernet HWaddr Brack Briver smc91x inet addr:192.168.122.62 Bcast:192.168.122.255 Mask:255.255.255.0 inet6 addr: Brack B
console:/	#

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 instead.

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

Total Compute

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-Yeats 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_TC3 \ --plugin <fvp-model plugin path/ListTraceSources.so> \ >& /tmp/trace-sources-fvp-tc3.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-Yeats:

```
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:

Multiple trace sources can be requested by separating the trace-sources strings with commas, as exemplified on the previous command listing.

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:

Warning: Please note that the trace information output can be very verbose depending on the component and filtering options. This has the potential to produce a large amount of information, which in case of redirecting to a file, can easily achieve file sizes of GB or TB magnitude in a short period of time.

The following output excerpt illustrates an example of the trace information captured for the DPU (streamid=0x00000000) and GPU (streamid=0x00000200):

```
(...)
cpnss.smmu_rp0_tcu.start_ptw_read: trans_id=0x0000000000000000000000000000000
...
streamid=0x00000000 substreamid=0xffffffff ttb_grain_stage_and_
...
elevel=0x00000202 pa_address=0x000000088ea5bfe0 input_
...
address=0x00000000ff800000 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 (continues on next page)
```

```
(continued from previous page)
```

```
cpnss.smmu_rp0_tcu.verbose_commentary: output="Performing a Table Walk read_
→as:-"
cpnss.smmu_rp0_tcu.verbose_commentary: output=" trans_id:121-st2-final-l2-
→aa64-ttb0-vmid:0-ns-sid:0"
cpnss.smmu_rp0_tcu.verbose_commentary: output="to ns-0x000000088ea5bfe0-PND-
→u0x5300000a-m0xffffffff-ish-osh-rawaC-rawaC of size 8B"
cpnss.smmu_rp0_tcu.verbose_commentary: output="Table Walk finished:-"
cpnss.smmu_rp0_tcu.verbose_commentary: output=" trans_id:121-st2-final-l2-
→aa64-ttb0-vmid:0-ns-sid:0"
cpnss.smmu_rp0_tcu.verbose_commentary: output="got:-"
                                                0x000000088ea5bfe0:
cpnss.smmu_rp0_tcu.verbose_commentary: output="
→0x00000088f2006d5"
cpnss.smmu_rp0_tcu.ptw_read: trans_id=0x00000000000000079 streamid=0x00000000_
--substreamid=0xffffffff ttb_grain_stage_and_level=0x00000202 pa_
→address=0x000000088ea5bfe0 input_address=0x00000000ff800000 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=0x000000088f2006d5 ssd=ns pas=ns
→mecid=0xfffffff
cpnss.smmu_rp0_tcu.ptw_read_st2_leaf_descriptor: trans_id=0x000000000000000079_
ustreamid=0x00000000 substreamid=0xffffffff ttb_grain_stage_and_
→level=0x00000202 pa_address=0x000000088ea5bfe0 input_
→address=0x00000000ff800000 ssd_ns=ssd_ns ns=bus-ns desckind=el2_or_st2_
→aarch64 XN=N contiguous=N AF=Y SH10=sh10_osh DBM=N HAP21=hap21_read_write_
--MemAttr3_0=memattr_oNC_iNC output_address=0x000000088f200000 nT=N s2hwu_
→pbha=0x00 NS=n/a AMEC=MEC not supported. ssd=ns pas=ns mecid=0xffffffff PIE_
→PIIndex=0xffff PIE_Dirty=n/a POE_POIndex=0xffff AssuredOnly=n/a
(...)
css.smmu.start_ptw_read: trans_id=0x00000000000000000040 streamid=0x00000200_
--substreamid=0xffffffff ttb_grain_stage_and_level=0x00000201 pa_
→address=0x0000000883794110 input_address=0x00000008899ad000 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=0xfffffff
css.smmu.verbose_commentary: output="Performing a Table Walk read as:-"
css.smmu.verbose_commentary: output=" trans_id:64-st2-final-l1-aa64-ttb0-
→vmid:1-ns-sid:512"
css.smmu.verbose_commentary: output="to ns-0x0000000883794110-PND-u0x53000109-
css.smmu.verbose_commentary: output="Table Walk finished:-"
css.smmu.verbose_commentary: output="
                                        trans_id:64-st2-final-l1-aa64-ttb0-
→vmid:1-ns-sid:512"
css.smmu.verbose_commentary: output="got:-"
css.smmu.verbose_commentary: output="
                                        0x000000883794110: 0x0000008899aa003
\rightarrow"
css.smmu.ptw_read: trans_id=0x000000000000000000 streamid=0x00000200
→substreamid=0xffffffffftttb_grain_stage_and_level=0x00000201 pa_
→address=0x0000000883794110 input_address=0x00000008899ad000 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=0x00000008899aa003 ssd=ns pas=ns
→mecid=0xfffffff
css.smmu.ptw_read_st2_table_descriptor: trans_id=0x00000000000000040
--streamid=0x00000200 substreamid=0xffffffff ttb_grain_stage_and_
→level=0x00000201 pa_address=0x0000000883794110 input_
                                                                (continues on next page)
→address=0x00000008899ad000 ssd_ns=ssd_ns ns=bus-ns desckind=el2_or_st2_
 <u>→aarch64 APTable=aptable_no_effect XNTable=N PXNTable=N.</u>
```

(...)

DICE/DPE

Verify DPE from U-boot

To verify DPE is working, run Android distro with AVB (the authentication option) enabled. Refer to the *Build variants configuration* section for AVB.

It should build and run successfully. And on terminal_uart_ap, there is output:

```
PVMFW load addr 84000000 size 426 KiB
Loading PVMFW to f1973000, end f19df5bf ... OK
```

which shows that PVMFW image is verified and loaded successfully.

Verify DPE from Microdroid

On Android 14, with AVB enabled, the protected VM is supported. To verify this, run Microdroid with protected option.

Refer to the *Microdroid* section on how to run Microdroid instance. Based on that, to use the protected option, run the command:

for one Microdroid instance
./run-scripts/run_microdroid_demo.sh start-microdroid --auto-connect --protected

::

for more than one Microdroid instance # start the instance ./run-scripts/run_microdroid_demo.sh startmicrodroid –protected # from a new terminal, connect to the instance ./run_microdroid_demo.sh vm-connect <CID>

It should run to Microdroid VM shell prompt. Which verifies that DPE is working.

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

1.5 System profiling, Applications tracing and Trace analysis

This section provides information related with tools, methodologies and features present and supported in the current Total Compute release, that allow to profile the performance and behaviour of the system and/or applications.

1.5.1 Simpleperf

Simpleperf is a native CPU profiling tool for Android, which can be used to profile both Android applications and native processes running on Android. Detailed documentation about this tool can be found on link.

The Linux Kernel exposes several Performance Monitoring Unit (PMU) - CPU and non-CPU - events, as well as software and tracepoint events to user space via the perf_event_open system call, which is used by simpleperf to collect and process the event's data.

The following subsections do present some simpleperf command examples that can be used to obtain useful information to troubleshoot and validate the development. Detailed information on the available commands supported, and their usage can be obtained on link.

Note: Although not all, most of the following commands do require root privileges in order to retrieve some of the system wide information. Failing to do so, will result in the following error message *"System wide profiling needs root privilege"*. To obtain root privileges, simply run the command su **0** before running any of the following examples.

Simpleperf List

A list of the different supported events can be obtained by running the command simpleperf list. An example of the output provided by the command is presented below for reference:

console:/ \$ simpleperf list

```
List of hw-cache events:
  # More cache events are available in `simpleperf list raw`.
  branch-load-misses
  branch-loads
  dTLB-load-misses
  dTLB-loads
  iTLB-load-misses
  iTLB-loads
  L1-dcache-load-misses
  L1-dcache-loads
  L1-icache-load-misses
  L1-icache-loads
  LLC-load-misses
  LLC-loads
List of coresight etm events:
  cs_etm/autofdo/
  cs-etm
                        # CoreSight ETM instruction tracing
List of hardware events:
  branch-instructions
  branch-misses
  bus-cvcles
  cache-misses
  cache-references
  cpu-cycles
  instructions
  stalled-cycles-backend
```

```
stalled-cycles-frontend
List of pmu events:
  arm_cspmu_0/cycles/
  arm_cspmu_1/cycles/
  arm_cspmu_2/cycles/
  arm_cspmu_3/cycles/
  arm_dsu_0/bus_access/
  arm_dsu_0/bus_cycles/
  arm_dsu_0/cycles/
  arm_dsu_0/memory_error/
  armv9_cortex_a520/br_immed_retired/
  armv9_cortex_a520/br_mis_pred/
  (...)
  armv9_cortex_a520/trcextout3/
  armv9_cortex_a520/ttbr_write_retired/
  armv9_cortex_a725/br_immed_retired/
  armv9_cortex_a725/br_mis_pred/
  (...)
  armv9_cortex_a725/ttbr_write_retired/
  armv9_cortex_a725/unaligned_ldst_retired/
  armv9_cortex_x925/br_immed_retired/
  armv9_cortex_x925/br_mis_pred/
  (...)
  armv9_cortex_x925/ttbr_write_retired/
  armv9_cortex_x925/unaligned_ldst_retired/
List of raw events provided by cpu pmu:
  # Please refer to "PMU common architectural and microarchitectural event numbers"
  # and "ARM recommendations for IMPLEMENTATION DEFINED event numbers" listed in
  # ARMv8 manual for details.
  # A possible link is https://developer.arm.com/docs/ddi0487/latest/arm-architecture-
→reference-manual-armv8-for-armv8-a-architecture-profile.
  raw-ase-spec (may not supported)
                                                 # Operation speculatively executed,__
→Advanced SIMD instruction
  raw-br-immed-retired (may not supported)
                                                 # Instruction architecturally executed, __
\rightarrow immediate branch
  raw-br-immed-spec (may not supported)
                                                 # Branch speculatively executed,__
\rightarrow immediate branch
  (...)
  raw-unaligned-st-spec (may not supported)
                                                 # Unaligned access, write
  raw-vfp-spec (may not supported)
                                                 # Operation speculatively executed,__
→floating-point instruction
List of software events:
  alignment-faults
  context-switches
  cpu-clock
  cpu-migrations
  emulation-faults
  maior-faults
  minor-faults
  page-faults
                                                                             (continues on next page)
```

task-clock			
()			

Simpleperf Stat

The stat command can be used to get event counter values of the profiled processes. The command can be customised to filter which events to use, which processes/threads to monitor, how to monitor and what print interval to adopt.

Some command examples are presented below.

Get system wide event counts for a specific duration and print at a specific interval

The following command allows to get the system wide default event counts, considering a duration of 1s and printing counts every 50ms:

```
console:/ # simpleperf stat -a --duration 1 --interval 0.05
Performance counter statistics:
#
           count event_name
                                                # count / runtime
                                               # 0.012290 GHz
       1,562,624 cpu-cycles
                0 stalled-cycles-frontend # 0.000 /sec
                0 stalled-cycles-backend # 0.000 /sec
       1,427,862 instructions
                                               # 1.094380 cycles per instruction
          427,802 INSTRUCTIONS # 1.094380 cycle
45,105 branch-instructions # 392.411 K/sec

0 branch-misses

                                              # 0.000000% miss rate
  106.490720(ms) task-clock
                                              # 88.921592 cpus used
               14 context-switches
                                              # 139.296 /sec
               11 page-faults
                                              # 114.277 /sec
Total test time: 0.001198 seconds.
Performance counter statistics:
#
           count event_name
                                               # count / runtime
       3,394,688 cpu-cycles
                                               # 0.015067 GHz
                0 stalled-cycles-frontend # 0.000 /sec
                0 stalled-cycles-backend # 0.000 /sec
       3,309,558 instructions
                                              # 1.025722 cycles per instruction

      162,493
      branch-instructions
      # 1.025/22 cycle

      0
      branch-misses
      # 770.683 K/sec

      110(max)
      # 0.000000% miss

                                              # 0.000000% miss rate
  199.783110(ms) task-clock
                                              # 13.914760 cpus used
               54 context-switches
                                              # 278.607 /sec
               13 page-faults
                                              # 68.572 /sec
Total test time: 0.014358 seconds.
(...)
```

Get event counts for a specific process within a duration

The following command allows to get the default event counts for the process system_server considering a duration of 50ms:

```
console:/ # ps -A | grep system_server
svstem
             477 318 19313020 338596 do_epoll_wait 0 S system_server
console:/ #
console:/ # simpleperf stat -p 477 --duration 0.05
Performance counter statistics:
                                        # count / runtime
#
        count event_name
            ◊ cpu-cycles
                                        #
            $ stalled-cycles-frontend
                                        #

    stalled-cycles-backend

                                        #

0 instructions

                                        #

    branch-instructions

                                        #

    branch-misses

                                        #
                                      # 0.000000 cpus used
 0.000000(ms) task-clock
            © context-switches
                                      #

    page-faults

                                        #
Total test time: 0.052116 seconds.
console:/sdcard #
```

Get specific events for a particular process

```
# this example assumes the previous "system_server" process with PID 477
console:/ # simpleperf stat -e cpu-cycles -p 477 --duration 0.05
Performance counter statistics:
# count event_name # count / runtime
6,351,580 cpu-cycles # 0.099210 GHz
Total test time: 0.050210 seconds.
console:/sdcard #
# Additional examples:
console:/ # simpleperf stat -e cache-references, cache-misses -p 477 --duration 0.05
console:/ # simpleperf stat -e cache-references, cache-misses ls
```

Similarly to filtering events for a particular process using -p <PID> option, filtering events for specific threads can be achieved by using -t <TID> argument.

Get non-CPU PMU events

```
console:/ # simpleperf stat -a -e arm_dsu_0/cycles/ -- sleep 0.01
Performance counter statistics:
# count event_name # count / runtime
9,223,372,036,854,775,809 arm_dsu_0/cycles/ # 469025720716.176 G/sec
Total test time: 0.019609 seconds.
console:/ # simpleperf stat -a -e arm_cspmu_0/cycles/ -- sleep 0.01
Performance counter statistics:
# count event_name # count / runtime
9,223,372,036,854,775,809 arm_cspmu_0/cycles/ # 469169585366.791 G/sec
Total test time: 0.019612 seconds.
console:/sdcard #
```

Note: Non-CPU PMU events are not supported in per-process due to perf or simpleperf not being able to attach events to a process.

Collect event counters using event-groups

```
console:/ # simpleperf stat --group cpu-cycles, instructions -- ls
            debug_ramdisk
                                  lost+found
acct
                                               second_stage_resources
            dev
apex
                                  mnt
                                               storage
bin
            etc
                                  odm
                                               sys
bugreports fstab.total_compute
                                  odm_dlkm
                                               system
cache
                                  oem
                                               system_dlkm
            init
config
            init.common.rc
                                  postinstall system_ext
            init.environ.rc
                                               vendor
d
                                  proc
            init.total_compute.rc product
                                               vendor_dlkm
data
data_mirror linkerconfig
                                  sdcard
Performance counter statistics:
      count event_name # count / runtime
 27,147,316 cpu-cycles # 2.603942 GHz
 27,147,250 instructions # 1.000002 cycles per instruction
Total test time: 0.010935 seconds.
console:/sdcard #
```

Simpleperf Record

The **record** command is used to dump samples of the profiled processes. The following example provides a very basic usage scenario of the command:

```
console:/sdcard # pwd
/sdcard
console:/sdcard # simpleperf record ls
Alarms
           DCIM
                     Movies
                                    Pictures
                                                Ringtones
Android
           Documents Music
                                    Podcasts
                                                TemporaryFile-t57Mnj
Audiobooks Download Notifications Recordings perf.data
simpleperf I cmd_record.cpp:798] Recorded for 0.0108992 seconds. Start post processing.
simpleperf I cmd_record.cpp:891] Samples recorded: 37. Samples lost: 0.
console:/sdcard #
```

Some additional command usage examples may include:

```
# Record individual process for a specific duration:
console:/ # simpleperf record -p <PID> --duration <DURATION IN SECONDS>
# Record set of processes for a specific duration:
console:/ # simpleperf record -p <PID1>,<PID2> --duration <DURATION IN SECONDS>
# Spawn workload as a child process and record it:
console:/ # simpleperf record <WORKLOAD APPLICATION>
# Frequency of the record can be set using -f or -c option, where
# '-f 1000' means collecting 1000 records every second, and
# '-c 1000' means collecting 1 record when 1000 events are hit.
console:/ # simpleperf record -f <FREQUENCY> -p <PID> --duration <DURATION IN SECONDS>
console:/ # simpleperf record -c <COUNT> -p <PID> --duration <DURATION IN SECONDS>
```

Simpleperf Report

The report command is used to report profiling data generated by the record command. The following example assumes being executed following the previous simpleperf record 1s command example:

```
# this example assumes and follows the run of the previous "simpleperf record ls" example
console:/sdcard # simpleperf report
Cmdline: /system/bin/simpleperf record ls
Arch: arm64
Event: cpu-cycles (type 0, config 0)
Samples: 37
Event count: 25996499
Overhead Command
                           Tid
                                 Shared Object
                                                                                Svmbol
                     Pid
38.72%
         ls
                     2145 2145 /system/lib64/libcrypto.so
                                                                                sha256_
→block_data_order
16.70%
        ls
                     2145 2145 [kernel.kallsyms]
                                                                                invoke
⇔syscall
7.56%
         ls
                     2145 2145 [kernel.kallsyms]
                                                                                perf_
→output_end
6.67%
         ls
                     2145 2145 /apex/com.android.runtime/bin/linker64
--[linker]soinfo::lookup_version_info(VersionTracker const&, unsigned int, (chairu@ons@tpage)
→version_info const**)
```

⇔area_free	
4.64% ls 2145 2145 /apex/com.android.runtime/bin/linker64	
\rightarrow [linker]ElfReader::ReadDynamicSection()	
3.45% ls 2145 [kernel.kallsyms] el0_da	la
3.20% ls 2145 2145 /apex/com.android.runtime/lib64/bionic/libc.so	
⇔aarch64_cas4_acq	
3.16% ls 2145 2145 [kernel.kallsyms] mt_fir	nd
3.13% ls 2145 2145 [kernel.kallsyms] mas_wr	r_
⇔walk	
2.97% ls 2145 2145 [kernel.kallsyms] mas_	
⇔next_node	
2.91% ls 2145 2145 [kernel.kallsyms]rcu_	L
⇔read_unlock	
1.56% ls 2145 2145 [kernel.kallsyms] mas_	
⇔destroy	
0.10% ls 2145 2145 [kernel.kallsyms] mas_	
⇔walk	
0.01% ls 2145 2145 [kernel.kallsyms]pte_	_
→alloc	
0.00% ls 2145 2145 [kernel.kallsyms] down_	-
⇔write_killable	
0.00% ls 2145 2145 [kernel.kallsyms] setup_	
⇔new_exec	
0.00% simpleperf 2145 2145 [kernel.kallsyms]rcu_	L
⇔read_lock	
<pre>console:/sdcard #</pre>	

1.5.2 Perf

Perf is a profiler tool for Linux based systems that abstracts CPU hardware differences in Linux performance measurements, while presenting a simple command-line interface.

More information on the tool can be found on link.

The Linux Kernel exposes several Performance Monitoring Unit (PMU) - CPU and non-CPU - events, as well as software and tracepoint events to user space via the perf_event_open system call, which is used by perf to collect and process the event's data.

List of available events

A list of the different supported events can be obtained by running the command perf list. An example of the output provided by the command is presented below for reference:

```
# perf list
List of pre-defined events (to be used in -e or -M):
branch-instructions OR branches [Hardware event]
branch-misses [Hardware event]
bus-cycles [Hardware event]
cache-misses [Hardware event]
```

cache-references	[Hardware event]
cpu-cycles OR cycles	[Hardware event]
instructions	[Hardware event]
stalled-cycles-backend OR idle-cycles-backend	[Hardware event]
stalled-cycles-frontend OR idle-cycles-frontend	[Hardware event]
alignment-faults	[Software event]
bpf-output	[Software event]
cgroup-switches	[Software event]
context-switches OR cs	[Software event]
cpu-clock	[Software event]
cpu-migrations OR migrations	[Software event]
dummy	[Software event]
emulation-faults	[Software event]
major-faults	[Software event]
minor-faults	[Software event]
page-faults OR faults	[Software event]
task-clock	[Software event]
duration_time	[Tool event]
user_time	[Tool event]
system_time	[Tool event]
Il_dcache_load_misses	[Hardware cache event]
L1 deache load misses	[Hardware cache event]
L1 ucache load-misses	[Hardware cache event]
L1-icache-loads	[Hardware cache event]
LI-Icache-Ioaus	[Hardware cache event]
LLC-loads	[Hardware cache event]
hranch_load_misses	[Hardware cache event]
branch loads	[Hardware cache event]
dTLP load misson	[Hardware cache event]
dTLP_loade	[Hardware cache event]
iTLP load micros	[Hardware cache event]
IILD-IUdu-MISSES	[Hardware cache event]
IILD-IUdus hr immed retired OP armyQ cortex a520/hr immed ret	inod/ [Kornol DML ovent]
br immed netined OR armyO contex_3220/br immed_net	ined/ [Kernel PMU event]
br_immed_retired_OR_armyQ_cortex_a725/br_immed_ret	ined/ [Kernel PMU event]
br_mic_pred_OR_army0_contex_x925/bl_1mmed_ret	[Kornel DML event]
br_mis_pred_OR_armv0_cortex_a320/br_mis_pred/	[Kernel PMU event]
br_mis_pred_OR_armv9_cortex_a/25/br_mis_pred/	[Kernel PMU event]
br_mis_pred ok armv9_corcex_x925/br_mis_pred/	[Kerner Pho event]
()	matimad / [Karna] DMII avant]
tthe units noticed OR armu9_cortex_d520/tthe units	retired/ [Kernel PHU event]
tthe units noticed OR armu0 contex_d/25/tthe units	
unaligned ldst notined OP army contex_X925/UDF_WF110	
unaligned ldst retired OD army contex_a320/ unaligned	med_ldst_retired/[Kernel_DMU_evert]
anarighed_rust_rethed of armv9_cortex_d/23/unding	[Kernel DMI event]
$a_1m_c_pmu_v/cycres/$	[Kernel DMI event]
$a_1m_c_pmu_1/cycres/$	[Kernel PMU event]
$a_{1}m_{c} = c_{2}/c_{1}c_{2}/c_{2}c_{2}$	[Kerne] DMI event]
$a_1m_c_pm_d_p/cycres/$	[Kernel PHU event]
alm_usu_V/Jus_access/	[Vermer FIIO event]

arm_dsu_0/bus_cycles/ [Kernel PMU event] arm_dsu_0/cycles/ [Kernel PMU event] arm_dsu_0/memory_error/ [Kernel PMU event] [Kernel PMU event] arm_spe_0// arm_spe_1// [Kernel PMU event] [Kernel PMU event] cs_etm// cs_etm/autofdo/ [Kernel PMU event] (...) alarmtimer:alarmtimer_cancel [Tracepoint event] alarmtimer:alarmtimer_fired [Tracepoint event] alarmtimer:alarmtimer_start [Tracepoint event] alarmtimer:alarmtimer_suspend [Tracepoint event] (...)

Note: The previous command may present its output in an unformatted way when running on the FVP. It may be desirable to instead redirect its output to a file and then list the contents of that file by running the following command sequence:

```
# perf list > perf_list.txt
# cat perf_list.txt
```

Perf Stat

The stat command can be used to get event counter values of the profiled processes. Some examples of its usage are presented following, as well as some considerations to take into account when considering TC3, with direct implications on the perf stat command.

Special considerations considering TC3 and implications on the perf stat command

TC3 defines per-microarchitecture PMU instances. As a result, the Kernel CPU PMU events will be displayed for each CPU micro-architecture during perf list, as illustrated on the following excerpt:

```
(...)
    cpu_cycles OR armv9_cortex_a520/cpu_cycles/
    cpu_cycles OR armv9_cortex_a725/cpu_cycles/
    cpu_cycles OR armv9_cortex_x925/cpu_cycles/
    [Kernel PMU event]
    (...)
```

When considering Kernel 6.1 and for situations where the perf command is executed as a task-bound (cpu==-1), the event is opened on an arbitrary CPU PMU and will only count on a subset of CPUs. This means, for example, that it might open on a "big" PMU and only count while the task is running on "big" CPUs, but not while the task is running on "little" CPUs. The following excerpt illustrates one such situation, where cycles are not counted for the command 1s, as the command did execute on the CPUs whose PMU were not selected by perf to open the events:

```
# perf stat -e cycles -- ls
arm-ffa-tee.ko
Performance counter stats for 'ls':
```

(continues on next page)

(continued from previous page)

(0.

(continued from previous page)

To overcome this implication and always ensure the retrieval of meaningful data, perf commands should be executed in one of two possible ways:

1. providing to the perf command the individual CPU PMU events to count:

```
# perf stat -e armv9_cortex_x925/cpu_cycles/,armv9_cortex_a725/cpu_cycles/,
→armv9_cortex_a520/cpu_cycles/ -- ls
arm-ffa-tee.ko
Performance counter stats for 'ls':
           1224520
                          armv9_cortex_x925/cpu_cycles/
     <not counted>
                          armv9_cortex_a725/cpu_cycles/
                                                                                  <u>ц</u>
                 (0.00\%)
\rightarrow
     <not counted>
                          armv9_cortex_a520/cpu_cycles/
                                                                                  <u>ц</u>
                 (0.00\%)
       0.000970750 seconds time elapsed
       0.001074000 seconds user
       0.000000000 seconds sys
#
```

2. providing to the perf command a CPU mask so that the event is opened on all CPU PMUs:

As can also be seen on the previous example, the instructions are not broken down to specific PMU type (CPU type). This might be ambiguous for users to read the result, as instructions/cycles on different CPUs do have different performance meaning.

This issue seems to have been fixed in newer Kernel versions (>=6.6) and when running the perf command with the default event names (without providing the CPU mask). Therefore, a possible solution could be to compile perf from newer source code, and copy the resulting binary into the rootfs before booting the image, or alternatively use the scp

command to upload the binary to a booted system.

Additional perf stat command examples are illustrated following, where the $-C \ 0-7$ argument was used as a workaround for the above-mentioned issue (TC3 FVP has 8 CPUs):

```
## Single event
# perf stat -C 0-7 -e <EVENT> -- <WORKLOAD>
## Multiple events
# perf stat -C 0-7 -e <EVENT1>,<EVENT2>,...,<EVENT-N> -- <WORKLOAD>
## Event grouping
# perf stat -C 0-7 -e '{<EVENT1>,<EVENT2>,...,<EVENT-N>}' -- <WORKLOAD>
## Attaching to the existing process; 'sleep X' is passed to run perf for a specific...
-duration
# perf stat -C 0-7 -e <EVENT> -p <PID> -- sleep 1
```

Note: DSU and MCN PMU driver do not support all possible events by name. For cases where data for a particular event is not visible, perf stat can be used with a raw event ID. Some examples of how to read the non-CPU PMU event counters are presented below (the values 0xa2 and 0x182 are obtained from the respective component TRM documentation):

Perf Record, Report and Annotate

Running perf record will collect and generate a perf.data file containing the sampling data of one or more events. This data can be later analysed using perf report or perf annotate commands. By default, the perf record uses cycles as a default event.

To modify the sampling period while running perf record, two approaches can be followed:

- 1. frequency: specifies the average rate of samples/sec (-F option);
- 2. count: enforces sampling at the specifies event period (-c option).

Some command examples illustrating this usage are presented below:

```
# Sample on event cycles at the default frequency
perf record -C 0-7 <WORKLOAD>
# Sample on event instructions at 1000 samples/sec
perf record -C 0-7 -e instructions -F 1000 <WORKLOAD>
# Sample on event instructions at every 2000 occurrences of event
perf record -C 0-7 -e instructions -c 2000 <WORKLOAD>
```

Perf and Arm SPE extension

The Arm Statistical Profiling Extension (SPE) feature provides a hardware assisted CPU operation profiling mechanism. This provides accurate attribution of latencies and events down to individual instructions.

The general perf record command usage with SPE on TC23 platform looks like:

TC23 supports SPE only on Mid and Big CPUs and not on small CPUs, there are 2 SPE instances, arm_spe_0 for Mid CPUs (CPUs 2-5) and arm_spe_1 for big CPUs (CPUs 6-7). When workload needs to be analyzed using SPE, it should be bound to CPUs which have the SPE capability using taskset. So on TC23 platform workloads should be bound to CPUs 2-5 when using arm_spe_0 and workloads should be bound to CPUs 6-7 when using arm_spe_1. min_latency=0 config parameter is mandatory to provide with any perf-spe command.

The following listing illustrates how to record SPE samples on Mid CPUs with arm_spe_0:

```
# perf record -e arm_spe_0/min_latency=0/ -- taskset -c 2-5 ls
arm-tstee.ko build_env.cfg perf.data perf.data.old
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.225 MB perf.data ]
#
```

The previously recorded data (perf.data) can then be analyzed using the perf report command as follows:

```
# perf report
Warning:
Please install libunwind or libdw development packages during the perf build.
Only instruction-based sampling period is currently supported by Arm SPE.
# To display the perf.data header info, please use --header/--header-only option
#
#
# Total Lost Samples: 0
#
# Samples: 853 of event 'l1d-access'
# Event count (approx.): 853
#
# Children
               Self Command Shared Object
                                                   Symbol
#
  . . . . . . . .
           . . . . . . . .
                   . . . . . . .
                              #
   31.42%
             31.42% ls
                             [kernel.kallsyms]
                                                   [k] __sanitizer_cov_trace_
   9.03%
             9.03% taskset [kernel.kallsyms]
                                                   [k] __sanitizer_cov_trace_
   1.88%
             1.88% ls
                            [kernel.kallsyms]
                                                   [k] next_uptodate_page
```

1.52%	1.52%	ls	[kernel.kallsyms]	[k] do_set_pte
1.41%	1.41%	ls	[kernel.kallsyms]	<pre>[k] search_cmp_ftr_reg</pre>
1.41%	1.41%	ls	[kernel.kallsyms]	[k] unmap_page_range
1.06%	1.06%	ls	[kernel.kallsyms]	[k]pi_copy_page
1.06%	1.06%	ls	[kernel.kallsyms]	[k]rcu_read_unlock
()				

The previous example only specify min_latency=0 required config parameter. However, there can be situations where making usage of the other config parameters may help to filter profiling information. Complementing the previous example, let's assume it would be desirable to make usage of the config parameter event_filter=2, which discards all samples which do not have retired instructions events. The following command listing illustrates the command usage considering this scenario:

```
# perf record -e arm_spe_0/min_latency=0,event_filter=2/ -- taskset -c 2 ls
arm-tstee.ko build_env.cfg perf.data perf.data.old
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.225 MB perf.data ]
#
```

Detailed information regarding the config parameters can be found at link.

Kernel config and prerequisites for enabling Arm SPE can also be found in the Kernel documentation.

1.5.3 Perfetto

Perfetto is an open-source stack developed for performance instrumentation and trace analysis. It offers services and libraries for recording system-level and app-level traces, native and Java heap profiling, a library for analysing traces using SQL, and a web-based user interface (UI) that allows to visualize and explore the collected traces.

It has support for ftrace, atrace, /proc/{stat,vmstat,pid}/* and perf_event as data sources to collect system level traces. A data source can be seen as the capability, exposed by a producer, of providing some tracing data. The producer is an entity that offers the ability to contribute to the trace, advertising this ability with one or more data sources. A consumer is an entity that controls the tracing service, provides to the tracing service the trace configuration and reads back the trace buffers. The tracing service is a long-lived entity (i.e. a system daemon on Linux/Android) which handles the tracing sessions, routes trace configuration from consumer to producers, and manages trace buffers.

The data source defines its own schema (a protobuf) consisting of data source trace config (what kind of input config it would expect from the consumer) and trace packets (what kind of data it would output into the trace).

Some examples of data sources advertised by different producers to collect system-level traces are listed below:

- linux.process_stats
- linux.ftrace
- linux.sys_stats
- linux.perf

Recording and Visualising Traces with Perfetto

Perfetto can record traces, using either the UI (available at https://ui.perfetto.dev/#!/record) or the command line. An example of how perfetto can be used to collect traces using the command line is present below:

Some complementing considerations regarding the previous presented command listing:

- the use of -s localhost:<PORT> can be ignored if there is only one ADB instance available for debug to the host;
- the default ADB port is 5555; however, in cases where there are more than one ADB instance available for debug to the host, the port may change; in these situations refer to the output of the adb devices command or to the FVP-model start up log information to understand which port was assigned as replacement; the *ADB connection on Android* section provides additional information that can be useful to troubleshoot the connection;
- config.txt contains the perfetto trace config; some examples of this config will be presented in the *Trace* config examples subsection.

Once the perfetto trace file is collected and downloaded to the host, it can be loaded into the perfetto UI (available at https://ui.perfetto.dev/) using the option "open trace file", as illustrated on the following image:



Detailed information regarding perfetto can be found on the official documentation available at https://perfetto.dev/ docs/.

Trace config examples

This subsection provides three trace config examples that can be used to control the tracing service and influence the sampled data on the TC3 platform. Alongside to each trace configuration, examples of the visualisation of the respective captured trace data using the Perfetto UI are also included for reference.

Additional examples of data source trace configurations for different supported data sources can be found at https: //perfetto.dev/docs/ (please refer to the "Data sources" section). Some additional config examples can be found in test/configs/ directory in perfetto source code.

A full list of supported ftrace events can be found in file protos/perfetto/trace/ftrace/ftrace_event. proto in perfetto source code.

A full list of supported meminfo and vmstat counters can be found in file protos/perfetto/common/ sys_stats_counters.proto in perfetto source code.

Example 1: collect ftrace scheduling events, process stats and system stats counters every 1000ms:

Trace configuration file:

```
buffers {
  size kb: 16384
  fill_policy: RING_BUFFER
}
buffers {
  size_kb: 16384
  fill_policy: RING_BUFFER
}
data_sources {
  config {
   name: "linux.ftrace"
   target_buffer: 0
    ftrace_config {
      # Scheduling information and process tracking. Useful for:
      # - what is happening on each CPU at each moment
      # - why a thread was de-scheduled
      # - parent/child relationships between processes and threads.
      ftrace_events: "sched/sched_switch"
      ftrace_events: "power/suspend_resume"
      ftrace_events: "sched/sched_process_exit"
      ftrace_events: "sched/sched_process_free"
      ftrace_events: "task/task_newtask"
      ftrace_events: "task/task_rename"
      # Wakeup info. Allows to compute how long a task was
      # blocked due to CPU contention.
      ftrace_events: "sched/sched_wakeup"
```

```
# os.Trace markers:
      ftrace_events: "ftrace/print"
      # RSS and ION buffer events:
      ftrace_events: "mm_event/mm_event_record"
      ftrace_events: "kmem/rss_stat"
      ftrace_events: "kmem/ion_heap_grow"
      ftrace_events: "kmem/ion_heap_shrink"
   }
 }
}
data_sources {
  config {
   name: "linux.sys_stats"
   target_buffer: 1
    sys_stats_config {
      meminfo_period_ms: 100
      meminfo_counters: MEMINFO_MEM_AVAILABLE
      meminfo_counters: MEMINFO_BUFFERS
      meminfo_counters: MEMINFO_CACHED
      meminfo_counters: MEMINFO_SWAP_CACHED
      meminfo_counters: MEMINFO_ACTIVE
      meminfo_counters: MEMINFO_INACTIVE
      meminfo_counters: MEMINFO_ACTIVE_ANON
      meminfo_counters: MEMINFO_INACTIVE_ANON
      meminfo_counters: MEMINFO_ACTIVE_FILE
      meminfo_counters: MEMINFO_INACTIVE_FILE
      meminfo_counters: MEMINFO_UNEVICTABLE
      vmstat_period_ms: 100
      vmstat_counters: VMSTAT_NR_FREE_PAGES
      vmstat_counters: VMSTAT_NR_ALLOC_BATCH
      vmstat_counters: VMSTAT_NR_INACTIVE_ANON
      vmstat_counters: VMSTAT_NR_VMSCAN_WRITE
      vmstat_counters: VMSTAT_NR_VMSCAN_IMMEDIATE_RECLAIM
      vmstat_counters: VMSTAT_NR_WRITEBACK_TEMP
      stat_period_ms: 100
      stat_counters: STAT_CPU_TIMES
      stat_counters: STAT_IRQ_COUNTS
      stat_counters: STAT_FORK_COUNT
   }
  }
}
data_sources: {
   config {
       name: "linux.process_stats"
        target_buffer: 0
        process_stats_config {
            scan_all_processes_on_start: true
```

```
proc_stats_poll_ms: 1000
}
duration_ms: 1000
```

Perfetto UI visualisation:

Perfette	
Normality	
Outstand Difference	
Control Second Control Contro Control Control	
Outloop Sinter Sintere Sinter Sinter Sinter Sinter Sinter Sinter Sinter Sin	
Incl Madeba	
Current on Participation Partiteates Participation <th< th=""><th></th></th<>	
No.4 Sector Sector <th></th>	
Control of the second sec	
2 Owned Dec Image: Control of the state of the s	
© 0703L 0x1 Parafas Image: Contract of the contract of	
Original Control President Control	
Outcome Mac idea frage Control	
	· · · · · · · · · · · · · · · · · · ·
🕼 Sendh te kapey lit 🔹 🔺 tuzed poles 38	
Crowty june Products	hat here for
L Converte system at the syste	
Lungle Taxes A remolation A rem	
Does Astald cample	
Chan Channe cauragée Pretruis d'erem 🖌 🧧	
Report A	
Q Dogmentation	
P Riga	
🕐 Reporta bug 🗸 🗸 ensity dampéd 1120	
C 600640102 · 000.0000 ·	
✓ 000.0.004 / HH	
▼ 628,0.00,04 128	
V 020,029(413)	
▼ interLeter 64	
v cst,ångå HBA	
✓ cmit_damp4/182	the
Durier feletion	ŦŸ
CPU sched Size	
Verifie Detail Verifie Terministration of the field of	

Example 2: collect cpu_cycles and instructions CPU PMU counters on all CPUs:

Trace configuration file:

```
buffers {
   size_kb: 10240
   fill_policy: RING_BUFFER
}
data_sources {
   config {
    name: "linux.perf"
    target_buffer: 0
    perf_event_config {
      all_cpus: true
      timebase {
        frequency: 99
        counter: HW_CPU_CYCLES
        timestamp_clock: PERF_CLOCK_MONOTONIC
```

```
}
    }
 }
}
data_sources {
  config {
   name: "linux.perf"
    target_buffer: 0
    perf_event_config {
      all_cpus: true
      timebase {
        frequency: 99
        counter: HW_INSTRUCTIONS
        timestamp_clock: PERF_CLOCK_MONOTONIC
      }
    }
  }
}
duration_ms: 1000
```

Perfetto UI visualisation:

🗈 🔒 trace_file perfetto-trac	*(1 × +																			a 🔇
< → C @	O A https://ul.perfetto.de	v /#i/viewer?local_caci	ie_key=49fd1099-f070-	b996-8641-2d95d080f														ŵ		න =
nerfetto 🛛 🚍								Q. Search o												
Navigation ^		00000	00:00:00	000000	00.000	00.0000	00.0000	00:00:00	00.00.00	000000	00.00.00	00:00:00	00.0000	40.0000	00.00.00	00.0000	00:00:00	00 20 20	00.00.00	00.00.00
Den trace file	80.05.25 + 334.919.330	00.30.00	00.00.00	00.38.00 300.000.000	00.00.00	00.38.00 330.000.008	00.00.00 250.000 000	00.380.00 300-000-008	00.00-00 254 200 000	00.30.00 430-000-000	80.00-08 454 100-000	00.00.00 800.000 000	00.38.00 980-000-000	00.00.00	00.381.00 410-000-000	00.00.00 790.0000	00.38.00 750-009-008	00:00:00 800 100:000	00.80.00	00.00.00 Y00.000.0
Dpen with legacy UI	0 =																			
Q) Record new trace	Cipa di epurcycles 🖌 🖌	304																		
	Cpu 1 epu-cycles 🛛 🖌	2504																		
Current Trace	Cpu 2 opurcycles 🛛 🥓	TM												1					1	
trace, file perfetto-trace (1 MB)	Cipu 3 opurcipales 🛛 🖌	600M																		
Download	Cpu 4 epu cycles 🖌 📈	35						_								1				
S Query (SQL)	Cox Sequencies #	38					_													
ia vα	Dau & cource cardes and	354														_				
Metrics	Per Terracelles d	454							_			1								_
Info and stats	Part Conceptor	204													2					
		1014																		
Convert trace	Cpu 1 instructions 📈																			
Switch to legacy UI	Cpu 2 instructions 🖌	134										-								
🛓 Convert to įson	Cpu 3 instructions 🖌	6004								-										
Formate Terror	Cpu 4 instructions 📈	20									L									
Example fraces	Cpu S instructions 🛛 🖌	34													_					
Con Android example	Cpu 6 instructions 🖌 🖌	754																		
Open Criterie example	Cpu 7 instructions 🖌 🖌	434		_																
Support ^	✓ Misc Bobal Tracks																			
C Keyboard shortcuts	✓ Process 776																			
Documentation	← Process 1663																			
P Flags	✓ Process 454																			
Report a bug	✓ Process 1208																			
C) Record metabaloa	✓ Process 308																			
	✓ Process 638																			
	✓ Process 935																			
	V Process 389																			
	✓ Process 165																			
	✓ Process 311																			
	Const failution																			Ŧv
	Profile: Saveles														8	lected function: (none) Snarshot tin	er 915ms 588us 710ns	Form	1.1
	roat: 2																			
	perfetto::ServiceMain[int, char*	*}																		
	poll appll	wari j								std::1::_	function:func-pe	rfette::TrasingServi	iceEmpl::NotifyFlush	DoneForProducer(uns:	igned short, unsigne	llong)::8,0, std::.	_1::allocatorsperfe	to::TracingServiceIm	pl::MetifyFlushDe	nefarfr.
										perfetto::T	racingGerviceImpl::S	crapeSharedMemorySut	ffers[perfetto::Tra:	ingGerviceImpl::Tra	cingSection*, perfet	o::TracingServiceIn	pl::ProducerEndpoin	(Impl*)		

Example 3: call stack sampling of processes:

Trace configuration file:

```
buffers {
  size_kb: 10240
  fill_policy: RING_BUFFER
}
data_sources {
  config {
    name: "linux.perf"
    target_buffer: 0
    perf_event_config {
      timebase {
        frequency: 99
        timestamp_clock: PERF_CLOCK_MONOTONIC
      }
      callstack_sampling {
        kernel_frames: true
      }
    }
  }
}
duration_ms: 1000
```

Perfetto UI visualisation:



Copyright (c) 2022-2024, Arm Limited. All rights reserved.

1.6 Security

1.6.1 Assumptions and Delegated Mitigations

ID	01
Assumption	Physical attack is out of scope.
Delegated Mitigation	None. None.
Impact if assumption is wrong	Unhandled vulnerabilities.

ID	02
Assumption	Part of NOR flash access restriction is controlled by end platform providers.
Delegated	The current hardware and software implementations lack protection to NOR Flash. The vendor
Mitigation	should add protection before NOR Flash and program the access restrictions similar to the Memory
	Side Filter (MSF).
Impact if	Secure keys/FIP image can be overwritten.
assumption is	
wrong	

ID	03
Assumption	Part of DDR region access restriction is controlled by end platform providers.
Delegated Miti-	The current hardware and software implementations lack protection to DDR. The vendor should
gation	add protection before DDR and program the access restrictions similar to the Memory Side Filter
	(MSF).
Impact if the	Can be overwritten by Malicious software.
assumption is	
wrong	

ID	04
Assump-	The firmware update capsule is generated from a working set of firmware image blobs (TF-A BL2,
tion	SCP BL2, etc.) and not tampered in DDR. (The Firmware Update Capsule contains the FIP image
	and additional FWU specific headers. On the TC SW stack reference design, the update capsule is
	preloaded manually to the DDR before execution.)
Dele-	None.
gated	
Mitiga-	
tion	
Impact	Boot hangs.
if the as-	
sumption	
is wrong	

ID	05
Assumption	Not using the dummy key provided with the TC software stack for the platform
	secure boot.
Delegated Mitigation	The vendor should follow RSE guidelines to create custom keys and securely store
	them.
Impact if the assumption is	The keys are exposed and the secure boot is compromised.
wrong	

ID	06
Assumption	Securely handle debug and trace for production releases.
Delegated Mitiga-	The vendor should disable the debug and trace capability for production releases or enable
tion	proper debug authentication as recommended by DEN0034.
Impact if the as-	Secure data can be exposed. Arbitrary code can be executed.
sumption is wrong	

1.7 Expected test results


1.7.1 SCMI unit tests

```
# cat arm_scmi_test_log.txt
      **** SCMI Compliance Suite ****
 Using SCMI kernel Raw transport rooted at:/sys/kernel/debug/scmi/0/raw
 Resetting SCMI kernel Raw queues.
 Using *strict* SCMI protocol version checking
        *** Starting BASE tests ***
101: Base protocol version check
   [Check 1] Query protocol version
     MSG HDR : 0x00004000
                  : 0
     NUM PARAM
     CHECK STATUS : PASSED [SCMI_STATUS_SUCCES]
     CHECK HEADER : PASSED [0x00004000]
     RETURN COUNT : 1
     RETURN[00]
                  : 0x00020000
     VERSION
                   : 0x00020000
                                                         : CONFORMANT
102: Base protocol attributes check
    [Check 1] Query protocol attributes
     MSG HDR
                : 0x00044001
     NUM PARAM
                   : 0
     CHECK STATUS : PASSED [SCMI_STATUS_SUCCES]
     CHECK HEADER : PASSED [0x00044001]
     RETURN COUNT : 1
     RETURN [00]
                   : 0x00000204
     CHECK RSVD BITS: PASSED
     CHECK NUM AGENTS: PASSED [0x0000002]
     CHECK NUM PROTOCOLS: PASSED [0x0000004]
                                                     : CONFORMANT
(...output truncated...)
517: Clock config set check
     NUM CLOCKS
                  : 3
   CLOCK ID: 0
    [Check 1] Set config with attributes :0
     MSG HDR : 0x05885007
                  : 2
     NUM PARAM
     PARAMETER[00] : 0x00000000
     PARAMETER[01] : 0x00000000
     CHECK HEADER : PASSED [0x05885007]
     CHECK STATUS : FAILED
         EXPECTED : SCMI_DENIED_ERROR
         RECEIVED : SCMI_NOT_SUPPORTED
     CHECK STATUS : PASSED [SCMI_NOT_SUPPORTED]
   CLOCK ID: 1
   [Check 1] Set config with attributes :1
     MSG HDR : 0x058c5007
                   : 2
     NUM PARAM
     PARAMETER[00] : 0x00000001
     PARAMETER[01] : 0x00000001
     CHECK HEADER : PASSED [0x058c5007]
     CHECK STATUS : FAILED
```

```
EXPECTED : SCMI_DENIED_ERROR
                  : SCMI_STATUS_SUCCES
         RECEIVED
      CHECK STATUS
                  : FAILED
         EXPECTED : SCMI_NOT_SUPPORTED
         RECEIVED : SCMI_STATUS_SUCCES
      CHECK STATUS
                   : PASSED [SCMI_STATUS_SUCCES]
      RETURN COUNT
                  : 0
    [Check 2] Verify the changed attribute
      MSG HDR
                  : 0x05905003
      NUM PARAM
                    : 1
      PARAMETER[00] : 0x00000001
      CHECK STATUS
                  : PASSED [SCMI_STATUS_SUCCES]
      CHECK HEADER : PASSED [0x05905003]
      CHECK RSVD BITS: PASSED
      RETURN COUNT : 5
      RETURN [00]
                  : 0x00000001
      RETURN[01]
                   : 0x45584950
      RETURN[02]
                  : 0x00305f4c
      RETURN[03]
                  : 0x00000000
      RETURN[04]
                  : 0x00000000
      CHECK CLOCK STATUS : PASSED [0x0000001]
    [Check 3] Restore the original attributes :0
      MSG HDR
                 : 0x05945007
      NUM PARAM
                   : 2
      PARAMETER[00] : 0x00000001
      PARAMETER[01] : 0x0000000
      CHECK HEADER : PASSED [0x05945007]
      CHECK STATUS : FAILED
         EXPECTED : SCMI_STATUS_SUCCES
         RECEIVED : SCMI_NOT_SUPPORTED
                                                        : NON CONFORMANT
         *** Starting SENSOR tests ***
Calling agent have no access to SENSOR protocol
         *** Starting RESET tests ***
Calling agent have no access to RESET protocol
         *** Starting VOLTAGE tests ***
Calling agent have no access to Voltage protocol
*************************
 TOTAL TESTS: 86
                  PASSED: 75
                              FAILED: 1
                                           SKIPPED: 10
**** SCMI tests complete ****
```

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.

1.7.2 TF-A unit tests

```
NOTICE: Booting trusted firmware test framework
NOTICE: Built : 13:11:16, Aug 8 2024
NOTICE: v2.10(tc,release):8c2ca7e
NOTICE: Running at NS-EL2
NOTICE: Starting a new test session
Running test suite 'Framework Validation'
Description: Validate the core features of the test framework
> Executing 'NVM support'
 TEST COMPLETE
                                                            Passed
> Executing 'NVM serialisation'
 TEST COMPLETE
                                                            Passed
> Executing 'Events API'
 TEST COMPLETE
                                                            Passed
> Executing 'IRQ handling'
 TEST COMPLETE
                                                            Passed
> Executing 'SGI support'
 TEST COMPLETE
                                                            Passed
Running test suite 'Timer framework Validation'
Description: Validate the timer driver and timer framework
> Executing 'Verify the timer interrupt generation'
 TEST COMPLETE
                                                            Passed
> Executing 'Target timer to a power down cpu'
 TEST COMPLETE
                                                            Passed
> Executing 'Test scenario where multiple CPUs call same timeout'
 TEST COMPLETE
                                                            Passed
(output trancated)
> Executing 'Check if DIT Bit is preserved in RL/NS'
 TEST COMPLETE
                                                            Skipped
FEAT_RME not supported
> Test suite 'Framework Validation'
                                                            Passed
> Test suite 'Timer framework Validation'
                                                            Passed
```

(continued	from	previous	page)
(continued	monn	previous	pu ₅ c)

<pre>> Test suite 'Boot requirement tests'</pre>	
<pre>> Test suite 'Query runtime services'</pre>	Passed
<pre>> Test suite 'PSCI Version'</pre>	Passed
> Test suite 'PSCI Affinity Info'	Passed
Test suite ICDU Jeteluel	Passed
> lest suite 'CPU Hotplug'	Passed
> Test suite 'PSCI CPU Suspend'	Passed
> Test suite 'PSCI STAT'	Passed
<pre>> Test suite 'PSCI NODE_HW_STATE'</pre>	Passed
<pre>> Test suite 'PSCI Features'</pre>	Passed
<pre>> Test suite 'PSCI MIGRATE_INFO_TYPE'</pre>	Dagged
<pre>> Test suite 'PSCI mem_protect_check'</pre>	rasseu
<pre>> Test suite 'SDEI'</pre>	Passed
> Test suite 'Runtime Instrumentation Validation'	Passed
<pre>> Test suite 'TRNG'</pre>	Passed
> Test suite 'EM-ABI'	Passed
> Test suite 'IRO support in TSP'	Passed
Test suite ITCD handler standard functions result test!	Passed
> lest suite TSP handler standard functions result test	Passed
> Test suite 'Stress test TSP functionality'	Passed
> Test suite 'TSP PSTATE test'	Passed
<pre>> Test suite 'EL3 power state parser validation'</pre>	Passed
<pre>> Test suite 'State switch'</pre>	Passed
<pre>> Test suite 'CPU extensions'</pre>	Passed
<pre>> Test suite 'ARM_ARCH_SVC'</pre>	Passed
<pre>> Test suite 'Performance tests'</pre>	Passea
<pre>> Test suite 'SMC calling convention'</pre>	Passed
<pre>> Test suite 'FF-A Setup and Discovery'</pre>	Passed
	Passed

> Test suite 'FF-A SMCCC compliance'	
	Passed
> Test suite 'SP exceptions'	Passed
<pre>> Test suite 'FF-A Direct messaging'</pre>	Tusseu
	Passed
<pre>> Test suite 'FF-A Group0 interrupts'</pre>	
> Test suite 'FF-A Power management'	Passed
	Passed
<pre>> Test suite 'FF-A Memory Sharing'</pre>	
> Test suite 'SIMD SVE Registers context'	Passed
v lest surce sing, svi keyisters context	Passed
<pre>> Test suite 'FF-A Notifications'</pre>	
Test suite 'DML Leakage'	Passed
> lest suite Pho Leakage	Passed
<pre>> Test suite 'DebugFS'</pre>	
Test suits IDMI and SDM tests!	Passed
> lest suite kii ald SPA tests	Passed
> Test suite 'Realm payload at EL1'	
	Passed
Tests Skipped : 154	
Tests Passed : 85	
Tests Failed : 0	
Tests Crashed : 0	
lotal tests : 239	
NOTICE: Exiting 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.

1.7.3 TF-M unit tests

```
#### Execute test suites for the Secure area ####
Running Test Suite IPC secure interface test (TFM_S_IPC_TEST_1XXX)...
> Executing 'TFM_S_IPC_TEST_1001'
    Description: 'Get PSA framework version'
    TEST: TFM_S_IPC_TEST_1001 - PASSED!
> Executing 'TFM_S_IPC_TEST_1002'
    Description: 'Get version of an RoT Service'
    TEST: TFM_S_IPC_TEST_1002 - PASSED!
> Executing 'TFM_S_IPC_TEST_1004'
    Description: 'Request connection-based RoT Service'
    TEST: TFM_S_IPC_TEST_1004 - PASSED!
```

```
> Executing 'TFM_S_IPC_TEST_1006'
 Description: 'Call PSA RoT access APP RoT memory test service'
Connect success!
Call success!
  TEST: TFM_S_IPC_TEST_1006 - PASSED!
> Executing 'TFM_S_IPC_TEST_1012'
 Description: 'Request stateless service'
  TEST: TFM_S_IPC_TEST_1012 - PASSED!
TESTSUITE PASSED!
    (output trancated)
TEST: DPE_S_TEST_MUST_BE_THE_LAST - PASSED!
TESTSUITE PASSED!
*** Secure test suites summary ***
Test suite 'IPC secure interface test (TFM_S_IPC_TEST_1XXX)' has PASSED
Test suite 'Crypto secure interface tests (TFM_S_CRYPTO_TEST_1XXX)' has PASSED
Test suite 'Platform Service Secure interface tests(TFM_S_PLATFORM_TEST_1XXX)' has PASSED
Test suite 'DPE Secure Tests (DPE_S_TEST_1XXX)' has PASSED
*** End of Secure test suites ***
```

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.

1.7.4 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
         skip test, pseudo TA not found
- 1001 -
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
```

```
29678 subtests of which 0 failed
107 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.

1.7.5 Trusted Services and Client application unit tests

Expected command output for the Trusted Services:

```
# ts-service-test -g FwuServiceTests -g ItsServiceTests -g_
→ CryptoKeyDerivationServicePackedcTests -g CryptoMacServicePackedcTests -g
→ CryptoCipherServicePackedcTests -g CryptoHashServicePackedcTests -g
→CryptoServicePackedcTests -g CryptoServiceProtobufTests -g CryptoServiceLimitTests -v
TEST(FwuServiceTests, checkMetadataAccess) - 962 ms
TEST(FwuServiceTests, checkImgDirAccess) - 894 ms
TEST(ItsServiceTests, storeNewItem) - 977 ms
TEST(CryptoKeyDerivationServicePackedcTests, deriveAbort) - 908 ms
TEST(CryptoKeyDerivationServicePackedcTests, hkdfDeriveBytes) - 968 ms
TEST(CryptoKeyDerivationServicePackedcTests, hkdfDeriveKey) - 949 ms
TEST(CryptoMacServicePackedcTests, macAbort) - 891 ms
TEST(CryptoMacServicePackedcTests, signAndVerify) - 3208 ms
TEST(CryptoCipherServicePackedcTests, cipherAbort) - 896 ms
TEST(CryptoCipherServicePackedcTests, encryptDecryptRoundtrip) - 1715 ms
TEST(CryptoHashServicePackedcTests, hashAbort) - 713 ms
TEST(CryptoHashServicePackedcTests, hashAndVerify) - 858 ms
TEST(CryptoHashServicePackedcTests, calculateHash) - 641 ms
TEST(CryptoServicePackedcTests, getUefiPrivAuthVarFingerprint) - 666 ms
TEST(CryptoServicePackedcTests, verifyPkcs7Signature) - 5661 ms
TEST(CryptoServicePackedcTests, generateRandomNumbers) - 700 ms
TEST(CryptoServicePackedcTests, asymEncryptDecryptWithSalt) - 73067 ms
TEST(CryptoServicePackedcTests, asymEncryptDecrypt) - 62412 ms
TEST(CryptoServicePackedcTests, signAndVerifyEat) - 12352 ms
TEST(CryptoServicePackedcTests, signAndVerifyMessage) - 12439 ms
TEST(CryptoServicePackedcTests, signAndVerifyHash) - 12465 ms
TEST(CryptoServicePackedcTests, exportAndImportKeyPair) - 1646 ms
TEST(CryptoServicePackedcTests, exportPublicKey) - 2582 ms
TEST(CryptoServicePackedcTests, purgeKey) - 1584 ms
TEST(CryptoServicePackedcTests, copyKey) - 87618 ms
TEST(CryptoServicePackedcTests, generatePersistentKeys) - 2622 ms
TEST(CryptoServicePackedcTests, generateVolatileKeys) - 2504 ms
TEST(CryptoServiceProtobufTests, generateRandomNumbers) - 621 ms
TEST(CryptoServiceProtobufTests, asymEncryptDecryptWithSalt) - 38097 ms
TEST(CryptoServiceProtobufTests, asymEncryptDecrypt) - 55196 ms
TEST(CryptoServiceProtobufTests, signAndVerifyMessage) - 12452 ms
```

```
TEST(CryptoServiceProtobufTests, signAndVerifyHash) - 12486 ms
TEST(CryptoServiceProtobufTests, exportAndImportKeyPair) - 1662 ms
TEST(CryptoServiceProtobufTests, exportPublicKey) - 2558 ms
TEST(CryptoServiceProtobufTests, generatePersistentKeys) - 2640 ms
TEST(CryptoServiceProtobufTests, generateVolatileKeys) - 2525 ms
TEST(CryptoServiceLimitTests, volatileRsaKeyPairLimit) - 2101315 ms
TEST(CryptoServiceLimitTests, volatileEccKeyPairLimit) - 84459 ms
OK (46 tests, 38 ran, 328 checks, 0 ignored, 8 filtered out, 2607650 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: AE Generating random bytes length: 7 Operation successful Random bytes: 41 D0 FA 7E 9A 6B 46 Generating random bytes length: 128 Operation successful Random bytes: DF 76 FB 96 A7 C3 CE 98 90 FD 51 50 20 EF CB 9E CA 70 0A D1 7A 4E 02 FA 0B FA 10 88 5A 1B 84 B7 72 58 7E B7 9F 76 BD E0 FE 40 45 16 EF 10 82 93 2B 64 BB D3 62 AE A9 46 B9 3F 0E FF 07 23 36 CF 40 95 8B E5 D5 CA 1A C6 8A C1 AB 59 D9 9F 18 3F EB E5 38 8D BB 2A 7D 32 4A 69 1F D6 24 5F 53 2F 5C 77 BC E8 63 16 9E D3 08 CD 52 F8 89 B6 0A 82 21 05 60 42 ED 8B 76 0F 79 CE CA 09 4F 29 95 57 Generating ECC signing key Operation successful Signing message: "The quick brown fox" using key: 256

Operation successful Signature bytes: 13 03 0E 48 E6 88 1B 16 90 D0 48 E4 36 37 E9 AB 9E 1E 84 AB 2A E8 D7 84 4B 03 1A 00 06 D1 52 52 36 40 0D BD D3 B2 DD E5 00 10 08 96 14 8C DE 99 CF DC AF 98 1F 38 16 55 53 ED 01 6F EE D3 82 32 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: 12 44 8C 20 DE D8 3F 66 9F 6B 84 ED 6D 16 DA D3 70 DD 44 2C B9 4C 72 52 CD 61 4B 38 9F 99 35 F4 FC E2 82 B7 5E 7C 65 A3 DB 01 36 1B 56 C4 A4 EF FD 98 6E 81 88 A7 0E A7 0F 72 E2 8D 8D 1B A7 2A 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: AC B8 A4 68 74 9C AA 56 F6 76 E5 09 1D A4 04 D6 C2 19 B4 63 F1 64 C5 AC BB FB 81 9D 4F 2D 12 0F Decrypting message using RSA key: 257 Operation successful Decrypted message: "Top secret" Exporting public key: 256 Operation successful Public key bytes: 04 74 D1 5F 50 01 8F CB 35 02 FA E8 00 1B 33 1C A2 31 75 FE E4 EF 07 3B 11 79 80 DA 04 CO 32 2E 98 8F 2E 14 2F 02 27 11 88 54 7F CA 1A BC 7C 35 EC 64 35 5F B4 4B EB 57 CD 84 A5 F9 50 FE 64 5C

```
24
Destroying signing key: 256
Operation successful
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.

1.7.6 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
Ε
[ RUN
           ] ipc.close_handle_negative
        OK ] ipc.close_handle_negative
Ε
[ RUN
           ] ipc.set_cookie_negative
Γ
        OK ] ipc.set_cookie_negative
[ RUN
           ] ipc.port_create_negative
Γ
        OK ] ipc.port_create_negative
[ RUN
           ] ipc.port_create
Γ
        OK ] ipc.port_create
[ RUN
           ] ipc.connect_negative
Γ
        OK ] ipc.connect_negative
[ RUN
           ] ipc.connect_close
        OK ] ipc.connect_close
Γ
[ RUN
           ] ipc.connect_access
        OK ] ipc.connect_access
Γ
[ RUN
           ] ipc.accept_negative
        OK ] ipc.accept_negative
Γ
[ DISABLED ] ipc.DISABLED_accept
[ RUN
           ] ipc.get_msg_negative
        OK ] ipc.get_msg_negative
Ε
[ RUN
           ] ipc.put_msg_negative
        OK ] ipc.put_msg_negative
Γ
[ RUN
           ] ipc.send_msg
Γ
        OK ] ipc.send_msg
[ RUN
           ] ipc.send_msg_negative
        OK ] ipc.send_msg_negative
Γ
[ RUN
           ] ipc.read_msg_negative
        OK ] ipc read_msg_negative
Γ
[ RUN
           ] ipc.end_to_end_msg
        OK ] ipc.end_to_end_msg
Γ
[ RUN
           ] ipc.hset_create
```

Ε		OK	ipc.hset_create	
Ε	RUN]	ipc.hset_add_mod_del	
Ε		OK]	ipc.hset_add_mod_del	
Ε	RUN]	ipc.hset_add_self	
Ε		OK]	ipc.hset_add_self	
Ε	RUN]	ipc.hset_add_loop	
Ε		OK]	ipc.hset_add_loop	
Ε	RUN]	ipc.hset_add_duplicate	
Ε		OK]	ipc.hset_add_duplicate	
Ε	RUN]	ipc.hset_wait_on_empty_set	
Ε		OK]	ipc.hset_wait_on_empty_set	
Γ	DISA	BLED	ipc.DISABLED_hset_add_chan	
Ε	RUN]	ipc.send_handle_negative	
Ε		OK]	ipc.send_handle_negative	
Ε	RUN]	ipc.recv_handle	
Ε		OK]	ipc.recv_handle	
Ε	RUN]	ipc.recv_handle_negative	
Ε		OK]	ipc.recv_handle_negative	
Ε	RUN]	ipc.echo_handle_bulk	
Ε		OK _	ipc.echo_handle_bulk	
Ε	RUN]	ipc.tipc_connect	
Ε		OK]	ipc.tipc_connect	
Ε	RUN]	ipc.tipc_send_recv_1	
Ε		OK]	ipc.tipc_send_recv_1	
Ε	RUN]	ipc.tipc_send_recv_hdr_payload	
Ε		OK]	ipc.tipc_send_recv_hdr_payload	
E		=====]	28 tests ran.	
Ε	PAS	SED]	28 tests.	
Γ	DISA	BLED	2 tests.	
C	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.

1.7.7 Microdroid Demo unit tests

[2024-04-03T09:17:23.149928488+00:00 INFO crosvm::crosvm::sys::unix::device_helpers]_ →Trying to attach block device: /proc/self/fd/49 [2024-04-03T09:17:23.150215128+00:00 INFO crosvm::crosvm::sys::unix::device_helpers] →Trying to attach block device: /proc/self/fd/54 [2024-04-03T09:17:23.196606144+00:00 INFO crosvm::crosvm::sys::unix::device_helpers] →Trying to attach block device: /proc/self/fd/63 0.089283][T21] Freeing initrd memory: 1652K Ε T17] cacheinfo: Unable to detect cache hierarchy for CPU 0 0.089863][Γ T1] brd: module loaded Γ 0.091308][T1] loop: module loaded 0.093654][Г T1] virtio_blk virtio3: 1/0/0 default/read/poll queues Γ 0.093768][Γ 0.093971][T1] virtio_blk virtio3: [vda] 100992 512-byte logical blocks (51.7 MB/ →49.3 MiB) 0.114159][T1] GPT:Primary header thinks Alt. header is not at the end of the Ε →disk. T1] GPT:100872 != 100991 Γ 0.114352][Γ 0.114438][T1] GPT:Alternate GPT header not at the end of the disk. 0.114582][T1] GPT:100872 != 100991 Γ T1] GPT: Use GNU Parted to correct GPT errors. Γ 0.114626][T1] vda: vda1 vda2 Γ 0.114801][T1] virtio_blk virtio4: 1/0/0 default/read/poll queues Г 0.114968][Γ 0.115253][T1] virtio_blk virtio4: [vdb] 20608 512-byte logical blocks (10.6 MB/ $\rightarrow 10.1$ MiB) 0.115994][T1] GPT:Primary header thinks Alt. header is not at the end of the Ε ⊸disk. T1] GPT:20552 != 20607 0.116152][Ε Г 0.116242][T1] GPT:Alternate GPT header not at the end of the disk. 0.116396][T1] GPT:20552 != 20607 Γ 0.116477][T1] GPT: Use GNU Parted to correct GPT errors. Γ T1] vdb: vdb1 Γ 0.116636][T1] virtio_blk virtio5: 1/0/0 default/read/poll queues Γ 0.116834][T1] virtio_blk virtio5: [vdc] 20608 512-byte logical blocks (10.6 MB/ Γ 0.117146][$\rightarrow 10.1$ MiB) Γ 0.117898][T1] GPT:Primary header thinks Alt. header is not at the end of the →disk. Ε 0.118063][T1] GPT:20592 != 20607 T1] GPT:Alternate GPT header not at the end of the disk. Γ 0.118135][T1] GPT:20592 != 20607 Г 0.118264][Γ T1] GPT: Use GNU Parted to correct GPT errors. 0.118345][Γ 0.118466][T1] vdc: vdc1 vdc2 vdc3 vdc4 vdc5 T1] zram: Added device: zram0 Γ 0.118871][0.119260][T1] rtc-pl030 2000.rtc: registered as rtc0 Г T1] rtc-pl030 2000.rtc: setting system clock to 2024-04-03T09:17:23_ Γ 0.119381][→UTC (1712135843) T1] device-mapper: uevent: version 1.0.3 0.119609][[Γ 0.119775][T1] device-mapper: ioctl: 4.47.0-ioctl (2022-07-28) initialised: dm-→devel@redhat.com 0.120060][T1] ipip: IPv4 and MPLS over IPv4 tunneling driver Г 0.120294][T1] gre: GRE over IPv4 demultiplexor driver Γ T1] ip_gre: GRE over IPv4 tunneling driver Г 0.120409][Γ 0.120824][T1] IPv4 over IPsec tunneling driver Ε 0.121037][T1] Initializing XFRM netlink socket

```
Γ
    0.121148][
                  T1] IPsec XFRM device driver
                  T1] NET: Registered PF_INET6 protocol family
Γ
    0.121435][
    0.121963][
                  T1] Segment Routing with IPv6
Γ
Γ
    0.122083][
                  T1] In-situ OAM (IOAM) with IPv6
                  T1] mip6: Mobile IPv6
Γ
    0.122229][
Γ
    0.122430][
                  T1] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
Γ
    0.122818][
                  T1] ip6_gre: GRE over IPv6 tunneling driver
Γ
    0.123047][
                  T1] NET: Registered PF_PACKET protocol family
Γ
                  T1] NET: Registered PF_KEY protocol family
    0.123182][
Γ
    0.123323][
                  T1] NET: Registered PF_VSOCK protocol family
Γ
    0.124165][
                  T1] page_owner is disabled
Γ
    0.124614][
                  T1] Freeing unused kernel memory: 1088K
Γ
                  T1] Run /init as init process
    0.133509][
                  T1] init: init first stage started!
Γ
    0.134628][
                  T1] init: Unable to open /lib/modules, skipping module loading.
Γ
    0.134767][
                  T1] init: [libfs_mgr] ReadFstabFromDt(): failed to read fstab from dt
Γ
    0.135053][
Γ
    0.135518][
                  T1] init: Using Android DT directory /proc/device-tree/firmware/
→android/
                  T1] init: [libfs_mgr] Created logical partition system_a on device /
Ε
    0.137395][
\rightarrow dev/block/dm-0
                  T1] init: [libfs_mgr] Created logical partition vendor_a on device /
Γ
    0.137780][
\rightarrow dev/block/dm-1
    0.138578][
                 T39] init: Attempting to run /first_stage.sh...
Ε
                 T40] init: unable to execv /first_stage.sh, returned -1 errno 2
Γ
    0.139359][
Γ
    0.139567][
                 T40] init: unable to execv, returned -1 errno 2
                 T39] init: /first_stage.sh exited with status 0
Γ
    0.139942][
Г
    0.140128][
                 T39] init: unable to execv, returned -1 errno 2
Γ
    0.140452][
                 T39] init (39) used greatest stack depth: 13392 bytes left
Γ
    0.140652][
                  T1] init: console shell exited with status 0
                  T1] init: Switching root to '/first_stage_ramdisk'
Γ
    0.140924][
                  T1] init: [libfs_mgr] ReadFstabFromDt(): failed to read fstab from dt
Γ
    0.141145][
                  T1] init: DSU not detected, proceeding with normal boot
Γ
    0.141644][
                  T1] init: [libfs_avb] Returning avb_handle with status: Success
Γ
    0.144962][
Γ
    0.145280][
                  T1] init: [libfs_avb] Built verity table: '1 /dev/block/dm-0 /dev/
→block/dm-0 4096 4096 10592 10592 sha256
\rightarrow b7f93850c7581f2354dc3f038df030beaeee5de2f28192e2c32d8049ae58fdba_
→5e9304c3065e2ff6849d58e4d214d664b6a8972366f94079d8d37a500bf83e46 2 restart_on_
Ε
    0.146029][
                  T1] device-mapper: verity: sha256 using implementation "sha256-generic
⊶"
Ε
    0.152161][
                  T1] init: [libfs_mgr] superblock s_max_mnt_count:65535,/dev/block/dm-2
    0.154789][
                  T1] EXT4-fs (dm-2): mounted filesystem with ordered data mode. Quota
Г
→mode: disabled.
    0.155092][
                  T1] init: [libfs_mgr] __mount(source=/dev/block/dm-2,target=/system,
L
→type=ext4)=0: Success
Γ
    0.155474][
                  T1] init: Switching root to '/system'
                  T1] init: [libfs_avb] Built verity table: '1 /dev/block/dm-1 /dev/
    0.156126][
Г
→block/dm-1 4096 4096 1095 1095 sha256
→5536d032d54ca7a61feb92600f10dcc42a002e02be1b22926c0cd4b55a84e976 2 restart on
→corruption ignore_zero_blocks'
    0.156922][
                  T1] device-mapper: verity: sha256 using implementation "sha256-generic
Ε
                                                                          (continues on next page)
```

```
Γ
     0.158980][
                   T1] init: [libfs_mgr] superblock s_max_mnt_count:65535,/dev/block/dm-3
     0.159662][
                   T1] EXT4-fs (dm-3): mounted filesystem without journal. Quota mode:
Ε
\rightarrow disabled.
                   T1] init: [libfs_mgr] __mount(source=/dev/block/dm-3,target=/vendor,
Г
    0.159911][
→type=ext4)=0: Success
Γ
     0.160494][
                   T1] init: Skipped setting INIT_AVB_VERSION (not in recovery mode)
     0.308305][
                   T1] init: DM_DEV_STATUS failed for system_ext_a: No such device or_
Γ
\rightarrow address
                   T1] init: Could not update logical partition
Γ
     0.308453][
     0.308614][
                   T1] init: DM_DEV_STATUS failed for product_a: No such device or
Г
⊶address
Γ
     0.308726][
                   T1] init: Could not update logical partition
                   T1] init: Opening SELinux policy
Γ
     0.308862][
                   T1] init: Error: Apex SEPolicy failed signature check
Γ
     0.309137][
                   T1] init: Loading APEX Sepolicy from /system/etc/selinux/apex/
    0.309254][
Γ
→SEPolicy.zip
Ε
     0.309361][
                   T1] init: Failed to open package /system/etc/selinux/apex/SEPolicy.
→ zip: No such file or directory
                   T1] init: Loading SELinux policy
Ε
     0.311394][
                   T1] SELinux: Permission nlmsg_getneigh in class netlink_route_socket_
     0.312782][
Г
\rightarrownot defined in policy.
Ε
     0.312995][
                   T1] SELinux: Permission bpf in class capability2 not defined in.
\rightarrow policy.
     0.313093][
                   T1] SELinux: Permission checkpoint_restore in class capability2 not_
Ε
\rightarrow defined in policy.
    0.313238][
                   T1] SELinux: Permission bpf in class cap2_userns not defined in_
Ε
\rightarrow policy.
Γ
     0.313355][
                   T1] SELinux: Permission checkpoint_restore in class cap2_userns not_
\rightarrow defined in policy.
     0.313555][
                   T1] SELinux: Class mctp_socket not defined in policy.
Γ
                   T1] SELinux: Class io_uring not defined in policy.
Γ
     0.313645][
Γ
                   T1] SELinux: Class user_namespace not defined in policy.
     0.313756][
Γ
     0.313859][
                   T1] SELinux: the above unknown classes and permissions will be denied
                   T1] SELinux: policy capability network_peer_controls=1
Γ
     0.318081][
Γ
     0.318197][
                   T1] SELinux: policy capability open_perms=1
Ε
                   T1] SELinux: policy capability extended_socket_class=1
     0.318307][
                                 policy capability always_check_network=0
Γ
     0.318424][
                   T1] SELinux:
Γ
     0.318510][
                   T1] SELinux:
                                 policy capability cgroup_seclabel=0
Γ
                   T1] SELinux: policy capability nnp_nosuid_transition=1
     0.318600][
Γ
     0.318715][
                   T1] SELinux:
                                 policy capability genfs_seclabel_symlinks=0
                   T1] SELinux: policy capability ioctl_skip_cloexec=0
Γ
     0.318818][
Γ
     0.361504][
                  T19] audit: type=1403 audit(1712135843.740:2): auid=4294967295_
→ses=4294967295 lsm=selinux res=1
Γ
     0.362349][
                   T1] selinux: SELinux: Loaded file context from:
                                             /system/etc/selinux/plat_file_contexts
                   T1] selinux:
Γ
     0.362577][
Γ
    0.362827][
                   T1] selinux: SELinux: Could not stat /dev/selinux: No such file or
\rightarrow directory.
                  T19] audit: type=1404 audit(1712135843.740:3): enforcing=1 old_
Γ
     0.363240][
→enforcing=0 auid=4294967295 ses=4294967295 enabled=1 old-enabled=1 lsm=selinux res=1
                   T1] init: init second stage started!
Г
    0.392339][
Γ
     0.394310][
                   T1] selinux: SELinux: Loaded file context from:
Ε
     0.394524][
                   T1] selinux:
                                             /system/etc/selinux/plat_file_contexts
```

```
Γ
     0.396143][
                   T1] init: Using Android DT directory /proc/device-tree/firmware/
\rightarrow and roid/
    0.397222][
                   T1] init: Setting property 'ro.build.fingerprint' to 'unknown/unknown/
Ε
→unknown:unknown/.4349bf74/unknown:unknown/unknown'
                   T1] selinux: SELinux: Loaded file context from:
Г
     0.397930][
Γ
     0.398164][
                   T1] selinux:
                                             /system/etc/selinux/plat_file_contexts
    0.398375][
                   T1] init: Running restorecon...
Ε
     0.400021][
                   T1] selinux: SELinux: Could not get canonical path for /metadata/ota/
Ε
→rollback-indicator restorecon: No such file or directory.
    0.400442][
                   T1] selinux: SELinux: Could not get canonical path for /metadata/gsi
Г
→restorecon: No such file or directory.
Γ
    0.400845][
                   T1] init: Created socket '/dev/socket/property_service', mode 666,
\rightarrowuser 0, group 0
                   T1] init: [libfs_mgr] vendor overlay: vndk version not defined
     0.401388][
Γ
                   T1] init: SetupMountNamespaces done
     0.401890][
Γ
                   T1] init: Not using subcontext for microdroid
Γ
    0.402051][
Ε
     0.402506][
                   T1] init: Parsing file /system/etc/init/hw/init.rc...
    0.402947][
                   T1] init: Added '/init.environ.rc' to import list
[
                   T1] init: Parsing file /init.environ.rc...
Γ
     0.403334][
Γ
     0.403500][
                   T1] init: Unable to read config file '/init.environ.rc': open()
\rightarrow failed: No such file or directory
(...output truncated...)
     0.487280][
                   T1] init: starting service 'microdroid_manager'...
Γ
                   T1] init: Created socket '/dev/socket/vm_payload_service', mode 666,
    0.487467][
Г
→user 1000, group 1000
     0.488369][
                   T1] init: ... started service 'microdroid_manager' has pid 54
Γ
Γ
     0.488570][
                  T49] ueventd: Coldboot took 0.041 seconds
(...output truncated...)
Γ
    0.550323][
                  T54] microdroid_manager[54]: started.
     0.550495][
                  T54] microdroid_manager[54]: ramdump supported: true
Г
     0.550615][
                  T54] microdroid_manager[54]: Os { code: 2, kind: NotFound, message:
Γ
\leftrightarrow "No such file or directory" }. Assumes <0>
(...output truncated...)
Г
     0.643333][
                  T54] microdroid_manager[54]: ramdump is loaded: debuggable=true,
→ramdump=false
     0.643757][
                  T54] zram0: detected capacity change from 0 to 454240
Г
                  T54] Adding 227116k swap on /dev/block/zram0. Priority:-2 extents:1.
Γ
     0.643944][
→across:227116k SS
    0.644101][
                  T54] microdroid_manager[54]: swap enabled.
[
Γ
     0.644248][
                  T55] kexec_load (55) used greatest stack depth: 11968 bytes left
                  T54] microdroid_manager::payload[54]: loading payload metadata...
Γ
     0.645184][
                  T54] microdroid_manager::ioutil[54]: waiting for "/dev/block/by-name/
Γ
    0.645326][
→payload-metadata"...
                  T54] microdroid_manager::dice[54]: Using sample DICE values
Г
    0.645526][
(...output truncated...)
```

```
T54] microdroid_manager[54]: payload verification successful. took 35.
Ε
     0.691145][
→677936ms
                  T54] microdroid_manager[54]: Saved data is verified.
     0.691307][
Г
                  T54] microdroid_manager[54]: DICE derivation for payload
Г
     0.691435][
Γ
     0.698650][
                  T54] microdroid_manager[54]: loading config from "/mnt/apk/assets/vm_

→config.json"...

     0.698819][
                  T54] microdroid_manager::ioutil[54]: waiting for "/mnt/apk/assets/vm_
Ε

→config.json"...

(...output truncated...)
     0.809250][
                  T54] microdroid_manager::vm_payload_service[54]: The RPC server 'vm_
Γ
→payload_service' is running.
(...output truncated...)
     0.817050][
                  T54] microdroid_manager[54]: boot completed, time to run payload
Г
                  T54] microdroid_manager[54]: executing main task Task { type_:_
Ε
     0.817176][
→MicrodroidLauncher, command: "TCMicrodroidApp.so" }...
Γ
                  T54] microdroid_manager[54]: notifying payload started
     0.817391][
Ε
     0.817551][
                   T1] init: processing action (enable_property_trigger) from (<Builtin_
\rightarrow Action>:0)
payload started
     0.818000][
                   T1] init: processing action (microdroid_manager.init_done=1) from (/
Г
→system/etc/init/hw/init.rc:49)
Г
     0.818196][
                   T1] init: Sending signal 9 to service 'ueventd' (pid 49) process_
→group...
     0.818872][
                   T1] libprocessgroup: Successfully killed process cgroup uid 0 pid 49_
Γ
\rightarrowin Oms
     0.819032][
                   T1] init: Service 'ueventd' (pid 49) received signal 9
Γ
                   T1] init: processing action (init_debug_policy.adbd.enabled=1) from (/
Γ
     0.819238][
→ system/etc/init/hw/init.rc:57)
     0.819539][
                   T1] init: starting service 'adbd'...
Г
     0.819637][
                   T1] init: Created socket '/dev/socket/adbd', mode 660, user 1000,
Γ
→group 1000
                   T1] init: ... started service 'adbd' has pid 74
     0.820284][
Г
Ε
     0.820686][
                  T73] microdroid_manager[73]: dropping capabilities before executing_
→payload
     0.893050][
                  T73] microdroid_launcher: Hello Microdroid!
Ε
                  T73] microdroid_launcher:
     0.893130][
Γ
     0.893189][
                  T73] microdroid_launcher:
Ε
                  T54] microdroid_manager[54]: task successfully finished
Ε
     0.893552][
     0.893648][
                  T54] microdroid_manager[54]: notifying payload finished
Γ
payload finished with exit code {\tt 0}
Γ
     0.893841][
                  T54] microdroid_manager[54]: Shutting down...
                  T48] init: Received sys.powerctl='shutdown' from pid: 54 (/system/bin/
Γ
     0.894009][

→microdroid_manager)

     0.894193][
                   T1] init: Got shutdown_command 'shutdown' Calling_
Γ
\rightarrow HandlePowerctlMessage()
Γ
     0.894317][
                   T1] init: Clear action queue and start shutdown trigger
Ε
     0.894430][
                   T1] init: Entering shutdown mode
```

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

```
[2024-04-03T09:17:24.547673048+00:00 INFO crosvm] exiting with success VM ended: Shutdown
```

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

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.

1.7.8 Kernel selftest unit tests

```
# ./run_kselftest.sh --summary
[ 520.082187][ T177] kselftest: Running tests in arm64
TAP version 13
1..17
# selftests: arm64: check_prctl
ok 1 selftests: arm64: check_prctl
# selftests: arm64: check_gcr_el1_cswitch
ok 2 selftests: arm64: check_gcr_el1_cswitch
# selftests: arm64: check_ksm_options
not ok 3 selftests: arm64: check_ksm_options # exit=1
# selftests: arm64: check_tags_inclusion
ok 4 selftests: arm64: check_tags_inclusion
# selftests: arm64: check_user_mem
ok 5 selftests: arm64: check_user_mem
# selftests: arm64: check_mmap_options
ok 6 selftests: arm64: check_mmap_options
# selftests: arm64: check_child_memory
ok 7 selftests: arm64: check_child_memory
# selftests: arm64: check_buffer_fill
ok 8 selftests: arm64: check_buffer_fill
# selftests: arm64: btitest
ok 9 selftests: arm64: btitest
# selftests: arm64: nobtitest
ok 10 selftests: arm64: nobtitest
# selftests: arm64: pac
ok 11 selftests: arm64: pac
# selftests: arm64: fp-stress
ok 12 selftests: arm64: fp-stress
# selftests: arm64: sve-ptrace
ok 13 selftests: arm64: sve-ptrace
# selftests: arm64: sve-probe-vls
ok 14 selftests: arm64: sve-probe-vls
# selftests: arm64: vec-syscfg
ok 15 selftests: arm64: vec-syscfg
# selftests: arm64: za-fork
ok 16 selftests: arm64: za-fork
```

```
# selftests: arm64: za-ptrace
ok 17 selftests: arm64: za-ptrace
#
```

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.

1.7.9 Rotational scheduler unit tests

```
# test_rotational_scheduler.sh
Enable The Rotational Scheduler
Pass
Checking the value of max_latency_us
Pass
Checking the value of max_residency_us
Pass
Checking the value of min_residency_us
Pass
Checking the value of hysteresis_active_tick
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.

1.7.10 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 1 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
[18748.735944][ T491] MPAM_arch: PART_SEL: 0x0
```

```
Pass
Setting the cpbm 4-5 in DSU for partition 45 and reading it back
[18748.736091][ T487] MPAM_arch: PART_SEL: 0x2d
[18748.736097][ T487] MPAM_arch: CPBM: 0x30 @ffff800008963000
[18748.737773][ T492] 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.

1.7.11 MPMM unit tests

```
# test_mpmm.sh tc3 fvp
Testing MPMM in FVP
Testing the MPMM of Cortex A520 cores
******
According to the PCT, the max frequency should be 2152000
Current set frequency of the cpu0 is 768000
PASS
Starting a vector intensive workload on cpu0
According to the PCT, the max frequency should be 2152000
Current set frequency of the cpu0 is 2152000
PASS
Starting a vector intensive workload on cpu1
According to the PCT, the max frequency should be 1844000
Current set frequency of the cpu0 is 1844000
PASS
Testing the MPMM of Cortex A725 cores
*****************
According to the PCT, the max frequency should be 2650000
Current set frequency of the cpu2 is 1893000
PASS
Starting a vector intensive workload on cpu2
According to the PCT, the max frequency should be 2271000
Current set frequency of the cpu2 is 2271000
PASS
Starting a vector intensive workload on cpu3
According to the PCT, the max frequency should be 1893000
Current set frequency of the cpu2 is 1893000
```

```
PASS
Starting a vector intensive workload on cpu4
According to the PCT, the max frequency should be 1419000
Current set frequency of the cpu2 is 1419000
PASS
Starting a vector intensive workload on cpu5
According to the PCT, the max frequency should be 1419000
Current set frequency of the cpu2 is 1419000
PASS
Testing the MPMM of Cortex X925 cores
*****
According to the PCT, the max frequency should be 3047000
Current set frequency of the cpu6 is 3047000
PASS
Starting a vector intensive workload on cpu6
According to the PCT, the max frequency should be 2612000
Current set frequency of the cpu6 is 2176000
PASS
Starting a vector intensive workload on cpu7
According to the PCT, the max frequency should be 2176000
Current set frequency of the cpu6 is 2176000
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.

1.7.12 BTI unit tests

<pre>console:/data/nativetest64/bti-unit-tests # ./bti-unit-tests</pre>			
[======================================	==]]	Running 17 tests from 7 test suites. Global test environment set-up.	
[]	3 tests from BR_Test	
[RUN]	BR_Test.GuardedMemoryWithX160rX17	
[0	К]	<pre>BR_Test.GuardedMemoryWithX160rX17 (206 ms)</pre>	
[RUN]	BR_Test.NonGuardedMemoryAnyRegister	
[0	К]	<pre>BR_Test.NonGuardedMemoryAnyRegister (0 ms)</pre>	
[RUN]	BR_Test.GuardedMemoryOtherRegisters	
[0	К]	<pre>BR_Test.GuardedMemoryOtherRegisters (155 ms)</pre>	
[]	3 tests from BR_Test (362 ms total)	
[]	3 tests from BRAA_Test	
[RUN]	BRAA_Test.GuardedMemoryWithX160rX17	

Ε	OK]	BRAA_Test.GuardedMemoryWithX160rX17 (429 ms)
[RU	N]	BRAA_Test.NonGuardedMemoryAnyRegister
[OK]	BRAA_Test.NonGuardedMemoryAnyRegister (0 ms)
[RU	N]	BRAA_Test.GuardedMemoryOtherRegisters
L	OK]	BRAA_Test.GuardedMemoryOtherRegisters (283 ms)
L	J	3 tests irom BRAA_Test (713 ms total)
г	٦	2 tasts from PDAP Test
	j NI J	PDAP Test CuardedMomenyWithV160rV17
		BRAD_Test GuardedMemoryWithY160rY17 (385 ms)
		BRAB Test NonGuardedMemoryAnyRegister
Г	01	BRAB Test NonGuardedMemoryAnyRegister (0 ms)
		BRAB Test GuardedMemoryOtherRegisters
Г	0 1	BRAB Test GuardedMemoryOtherRegisters (297 ms)
Г]	3 tests from BRAB Test (682 ms total)
L		
[]]	2 tests from BLR_Test
[RU	N]	BLR_Test.GuardedMemoryAnyRegister
Γ	OK]	BLR_Test.GuardedMemoryAnyRegister (427 ms)
[RU	N]	BLR_Test.NonGuardedMemoryAnyRegister
Γ	OK]	BLR_Test.NonGuardedMemoryAnyRegister (0 ms)
[]]	2 tests from BLR_Test (427 ms total)
[]]	2 tests from BLRAA_Test
[RU	N]	BLRAA_Test.GuardedMemoryAnyRegister
Γ	OK]	BLRAA_Test.GuardedMemoryAnyRegister (936 ms)
[RU	N]	BLRAA_Test.NonGuardedMemoryAnyRegister
Ε	OK]	BLRAA_Test.NonGuardedMemoryAnyRegister (0 ms)
[]]	2 tests from BLRAA_Test (937 ms total)
F	-	2 tests from BLDAD Test
	J	2 tests from BLRAB_lest
		BLRAB_TEST.GuardedMemoryAnyRegister
L FDIT		PLPAP Test NonCuardedMemoryAnyRegister (749 ms)
		PLPAP Test NonGuardedMemoryAnyPegister
L [1	2 tests from RIRAR Test (749 ms total)
L]	
Г	1	2 tests from BTI LinkerTest
Γ RU	ר מ	BTI LinkerTest.CallBasicFunction
Γ	OK]	BTI LinkerTest.CallBasicFunction (0 ms)
[RU	N 1	BTI_LinkerTest_BypassLandingPad
Γ	OK 1	BTI LinkerTest.BypassLandingPad (55 ms)
[]	2 tests from BTI_LinkerTest (55 ms total)
-	-	
[]]	Global test environment tear-down
[====	=====]	17 tests from 7 test suites ran. (3929 ms total)
[P.	ASSED]	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.

1.7.13 MTE unit tests

С	<pre>console:/data/nativetest64/mte-unit-tests # ./mte-unit-tests</pre>			
Г		1	Pupping 12 tosts from 1 tost suite	
L' E		ر د	Clobal test environment set un	
L		נ ר	12 tests from MTETest	
L		נ ר	MTETast CreatePandomTag	
L	KON		MTETest (reatePandomTag (0 ms)	
L	DIIN		MTETest IncrementTag	
L T	RON	0K] I	MTETest IncrementTag (0 ms)	
г Г	RIIN		MTFTest FxcludedTags	
Г	Ron	0 1	MTETest ExcludedTags (0 ms)	
Г	RUN	1	MTETest PointerSubtraction	
Ē		OK]	MTETest.PointerSubtraction (0 ms)	
Ē	RUN]	MTETest.TagStoreAndLoad	
Ē		OK]	MTETest.TagStoreAndLoad (0 ms)	
Ε	RUN]	MTETest.DCGZVA	
Ε		OK]	MTETest.DCGZVA (0 ms)	
Ε	RUN]	MTETest.DCGVA	
E		OK]	MTETest.DCGVA (0 ms)	
E	RUN]	MTETest.Segfault	
E		OK]	MTETest.Segfault (41 ms)	
E	RUN]	MTETest.UseAfterFree	
E		OK]	MTETest.UseAfterFree (0 ms)	
E	RUN]	MTETest.CopyOnWrite	
E		OK]	MTETest.CopyOnWrite (0 ms)	
E	RUN]	MTETest.mmapTempfile	
Ε		OK]	MTETest.mmapTempfile (5 ms)	
Γ	RUN]	MTETest.MTEIsEnabled	
L		OK]	MTETest.MTEIsEnabled (0 ms)	
L]	12 tests irom miElest (48 ms total)	
Γ		1	Global test environment tear-down	
Ē		====]	12 tests from 1 test suite ran. (48 ms total)	
Ε	PASS	ED]	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.

1.7.14 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 21 tests from 3 test suites.
[------] Global test environment set-up.
[-----] 3 tests from PAuthDeathTest
[RUN] PAuthDeathTest.SignFailure

Γ	OK] PAuthDeathTest.SignFailure (431 ms)
[RU	N] PAuthDeathTest.AuthFailureNoFpac
vend	or/arm/	examples/pauth/pauth_unit_tests/pauth_unit_tests.cpp:598: Skipped
[S	KIPPED] PAuthDeathTest.AuthFailureNoFpac (0 ms)
- Γ RU	N] PAuthDeathTest.AuthFailureFpac
Г	OK	PAuthDeathTest.AuthFailureFpac (411 ms)
Г		3 tests from PAuthDeathTest (842 ms total)
-		
[] 14 tests from PAuthTest
[RU	N] PAuthTest.Signing
Γ	OK] PAuthTest.Signing (0 ms)
[RU	N] PAuthTest.AuthenticationFpac
Ē	OK] PAuthTest.AuthenticationFpac (494 ms)
- Γ RU	N	PAuthTest.AuthenticationNoFpac
vend	or/arm/	examples/pauth/pauth_unit_tests/pauth_unit_tests.cpp:225: Skipped
[S	KIPPED] PAuthTest.AuthenticationNoFpac (0 ms)
[RU	N] PAuthTest.Stripping
vend	or/arm/	examples/pauth/pauth_unit_tests/pauth_unit_tests.cpp:269: Skipped
[S	KIPPED] PAuthTest.Stripping (0 ms)
[RU	N] PAuthTest.Roundtrip
Γ	OK] PAuthTest.Roundtrip (0 ms)
[RU	N] PAuthTest.StrippingWithBuiltinReturnAddress
Γ	OK] PAuthTest.StrippingWithBuiltinReturnAddress (0 ms)
[RU	N] PAuthTest.ExtractPAC
Γ	OK] PAuthTest.ExtractPAC (0 ms)
[RU	N] PAuthTest.PACMask
Γ	OK] PAuthTest.PACMask (0 ms)
[RU	N] PAuthTest.KeyChange
Γ	OK] PAuthTest.KeyChange (1 ms)
[RU	N] PAuthTest.GenericAuthentication
Γ	OK] PAuthTest.GenericAuthentication (0 ms)
[RU	N] PAuthTest.Unwind
Γ	OK] PAuthTest.Unwind (50 ms)
[RU	N] PAuthTest.CheckReturnAddressSigned
Γ	OK] PAuthTest.CheckReturnAddressSigned (0 ms)
[RU	N] PAuthTest.AuthenticateThenReturn
Γ	OK] PAuthTest.AuthenticateThenReturn (196 ms)
[RU	N] PAuthTest.CheckHWCAP
Γ	OK] PAuthTest.CheckHWCAP (0 ms)
[]] 14 tests from PAuthTest (743 ms total)
[] 4 tests from PAuthTestData
[RU	N] PAuthTestData.Signing
Ε	OK] PAuthTestData.Signing (0 ms)
[RU	N] PAuthTestData.AuthenticationFpac
Γ	OK] PAuthTestData.AuthenticationFpac (197 ms)
[RU	N] PAuthTestData.AuthenticationNoFpac
vend	or/arm/	examples/pauth/pauth_unit_tests/pauth_unit_tests.cpp:257: Skipped

Ε	SKIPPED]	PAuthTestData.AuthenticationNoFpac (0 ms)
[]]	RUN	PAuthTestData.Roundtrip
Ε	OK]	PAuthTestData.Roundtrip (≬ ms)
[-		4 tests from PAuthTestData (197 ms total)
[-		Global test environment tear-down
[==	=======	21 tests from 3 test suites ran. (1784 ms total)
Ε	PASSED	17 tests.
Ε	SKIPPED	4 tests, listed below:
Ε	SKIPPED	PAuthDeathTest.AuthFailureNoFpac
Ε	SKIPPED	PAuthTest.AuthenticationNoFpac
Ε	SKIPPED	PAuthTest.Stripping
Ε	SKIPPED	PAuthTestData.AuthenticationNoFpac

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.

1.7.15 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...)
[2024-04-09 08:13:33,473][EXEKALL] INFO Result summary:
EnergyModelWakeMigration[board=tc]:test_dmesg
→UUID=4dd0d0a7269e4fdc9be62a622f21f5b5 PASSED: dmesg output:
EnergyModelWakeMigration[board=tc]:test_slack
→UUID=e415d484124149479d4615981e94038a PASSED
        emwm_0-0 delayed activations: 1.7964071856287425 %
        emwm_1-1 delayed activations: 1.596806387225549 %
EnergyModelWakeMigration[board=tc]:test_task_placement_
\rightarrow UUID=27db420e26604de4b5eb93b5f7f415c4 PASSED
        energy threshold: 9917896.808890518 bogo-joules
       estimated energy: 9494907.774697093 bogo-joules
       noisiest task:
            comm: kworker/7:2
            duration (abs): 0.0005616000271402299 s
            duration (rel): 0.006970336884132867 %
            pid: 86
OneSmallTask[board=tc]:test_dmesg
→UUID=e403fdd6beef422288d9702265c568e8 PASSED: dmesg output:
OneSmallTask[board=tc]:test_slack
→UUID=35c9f262e8fe4f2abd6f2efc7f8ab3dc PASSED: small-0 delayed activations: 0.0 %
OneSmallTask[board=tc]:test_task_placement
→UUID=ad6bbe0e724e4c918c24651b283d0c58 PASSED
        energy threshold: 47367.4972158616 bogo-joules
       estimated energy: 45111.90211034438 bogo-joules
       noisiest task:
            comm: kworker/7:2
            duration (abs): 7.127999560907483e-05 s
            duration (rel): 0.00718544356093443 %
            pid: 86
RampDown[board=tc]:test_dmesg
→UUID=9e0eefb9240740ffaddec5a2079faa0c PASSED: dmesg output:
RampDown[board=tc]:test_slack
→UUID=a3290caf3e9948ea9faf66581ce24cdb PASSED: down-0 delayed activations: 0.0 %
RampDown[board=tc]:test_task_placement
→UUID=cec7149293f64823a57e1a3bbd3c7c22 PASSED
        energy threshold: 3254207.3701920374 bogo-joules
       estimated energy: 2724142.579732631 bogo-joules
       noisiest task:
            comm: kworker/7:2
            duration (abs): 0.00039719196502119303 s
            duration (rel): 0.0066732320449603404 %
            pid: 86
RampUp[board=tc]:test_dmesg
→UUID=467a049c6bcf41fdad6af63165332e64 PASSED: dmesg output:
```

```
RampUp[board=tc]:test_slack
{\hookrightarrow} UUID{=}349a857829c34c79a949b402bd4fd903 PASSED: up-0 delayed activations: 0.
→2680965147453083 %
RampUp[board=tc]:test_task_placement
→UUID=40f9064f78a049aa9ac471318887ed91 PASSED
        energy threshold: 2929607.402691307 bogo-joules
        estimated energy: 2590048.4205345856 bogo-joules
        noisiest task:
            comm: kworker/7:2
            duration (abs): 0.0004323198809288442 s
            duration (rel): 0.007262193744866007 %
            pid: 86
ThreeSmallTasks[board=tc]:test_dmesg
→UUID=f254d321ef294e0ba6e2135474173a18 PASSED: dmesg output:
ThreeSmallTasks[board=tc]:test slack
→UUID=89e4ba29744c405c8500971607adea94 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
\hookrightarrow UUID{=}445dd167367f4e6db82a563c271c8f8d\ PASSED
        energy threshold: 162402.8567740885 bogo-joules
        estimated energy: 136972.06192055883 bogo-joules
        noisiest task:
            comm: kworker/7:2
            duration (abs): 7.199199171736836e-05 s
            duration (rel): 0.007257161247512859 %
            pid: 86
TwoBigTasks[board=tc]:test_dmesg
→UUID=adbb1ada47084815af6ca34c66564bc7 PASSED: dmesg output:
TwoBigTasks[board=tc]:test_slack
→UUID=987995ebef644717b95348deca1df879 PASSED
        big_0-0 delayed activations: 3.1746031746031744 %
        big_1-1 delayed activations: 4.761904761904762 %
TwoBigTasks[board=tc]:test_task_placement
→UUID=56dfbebc469e49f995a7c65b4eb7779c PASSED
        energy threshold: 2389919.2425884334 bogo-joules
        estimated energy: 2336061.2754204613 bogo-joules
        noisiest task:
            comm: init
            duration (abs): 0.00018922402523458004 s
            duration (rel): 0.01868509877871718 %
            pid: 1
TwoBigThreeSmall[board=tc]:test_dmesg
→UUID=db563f87f9cf4803b405cbb6414e9322 PASSED: dmesg output:
TwoBigThreeSmall[board=tc]:test_slack
→UUID=e8809db1a86045b99ad30c031f5c0e2a PASSED
```

```
big_0-0 delayed activations: 4.838709677419355 %
big_1-1 delayed activations: 6.349206349206349 %
small_0-2 delayed activations: 0.0 %
small_1-3 delayed activations: 0.0 %
small_2-4 delayed activations: 0.0 %
TwoBigThreeSmall[board=tc]:test_task_placement _______
UUID=17fc2798916b4b1f9dae399c20dd3e63 FAILED
energy threshold: 2472495.7617728077 bogo-joules
estimated energy: 2699550.767562539 bogo-joules
noisiest task:
    comm: sshd
    duration (abs): 0.00029410398565232754 s
    duration (rel): 0.029381488120311203 %
    pid: 175
```

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.

1.7.16 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.

1.7.17 GPU GLES Integration tests

UTF: Platform Provenance _____ Hardware: TDRX r0p0 OS: Android UTF: Running gles1_api_integration _____ Initializing: S:{gles1_api_integration} T:{Triangles [0x0001]} D:{0} System time: Fri Aug 2 20:49:38 2024 [8754.400713] [T179] servicemanager: Found android.hardware.graphics.allocator. →IAllocator/default in device VINTF manifest. [8754.401104][T179] servicemanager: Found android.hardware.graphics.allocator. →IAllocator/default in device VINTF manifest. Running: S:{gles1_api_integration} T:{Triangles [0x0001]} D:{0} Terminating: S:{gles1_api_integration} T:{Triangles [0x0001]} D:{0} Pass: S:{gles1_api_integration} T:{Triangles [0x0001]} F:{ [0x00]} D:{662} - (implicit_ \rightarrow pass) Initializing: S:{gles1_api_integration} T:{glGetString [0x0002]} D:{0} System time: Fri Aug 2 20:49:39 2024 Running: S:{gles1_api_integration} T:{glGetString [0x0002]} D:{0} Terminating: S:{gles1_api_integration} T:{glGetString [0x0002]} D:{0} Pass: S:{gles1_api_integration} T:{glGetString [0x0002]} F:{ [0x00]} D:{268} - (implicit_ \rightarrow pass) Initializing: S:{gles1_api_integration} T:{gcc49_format_conversion [0x0003]} D:{0} System time: Fri Aug 2 20:49:39 2024 Running: S:{gles1_api_integration} T:{gcc49_format_conversion [0x0003]} D:{0} Terminating: S:{gles1_api_integration} T:{gcc49_format_conversion [0x0003]} D:{0} Pass: S:{gles1_api_integration} T:{gcc49_format_conversion [0x0003]} F:{ [0x00]} D:{157}_ \rightarrow - (implicit **pass**) Initializing: S:{gles1_api_integration} T:{gcc49_memory_allocation [0x0004]} D:{0} System time: Fri Aug 2 20:49:39 2024 Running: S:{gles1_api_integration} T:{gcc49_memory_allocation [0x0004]} D:{0} Terminating: S:{gles1_api_integration} T:{gcc49_memory_allocation [0x0004]} D:{0} Pass: S:{gles1_api_integration} T:{gcc49_memory_allocation [0x0004]} F:{ [0x00]} D:{140}_ \rightarrow - (implicit **pass**) _____ UTF: Running gles2_api_integration Initializing: S:{gles2_api_integration} T:{glGetString [0x0001]} D:{0} System time: Fri Aug 2 20:49:39 2024 Running: S:{gles2_api_integration} T:{glGetString [0x0001]} D:{0} Terminating: S:{gles2_api_integration} T:{glGetString [0x0001]} D:{0} Pass: S:{gles2_api_integration} T:{glGetString [0x0001]} F:{ [0x00]} D:{225} - (implicit_ \rightarrow pass) Initializing: S:{gles2_api_integration} T:{glClearColor_basic [0x0002]} D:{0} System time: Fri Aug 2 20:49:40 2024 Running: S:{gles2_api_integration} T:{glClearColor_basic [0x0002]} D:{0} Terminating: S:{gles2_api_integration} T:{glClearColor_basic [0x0002]} D:{0} Pass: S:{gles2_api_integration} T:{glClearColor_basic [0x0002]} F:{ [0x00]} D:{476} - \rightarrow (implicit **pass**)

```
Initializing: S:{gles2_api_integration} T:{glLinkProgram [0x0003]} D:{0}
System time: Fri Aug 2 20:49:40 2024
Running: S:{gles2_api_integration} T:{glLinkProgram [0x0003]} D:{0}
Terminating: S:{gles2_api_integration} T:{glLinkProgram [0x0003]} D:{0}
Pass: S:{gles2_api_integration} T:{glLinkProgram [0x0003]} F:{ [0x00]} D:{167} -
→(implicit pass)
Initializing: S:{gles2_api_integration} T:{untextured triangle [0x0004]} D:{0}
System time: Fri Aug 2 20:49:40 2024
Running: S:{gles2_api_integration} T:{untextured triangle [0x0004]} D:{0}
Terminating: S:{gles2_api_integration} T:{untextured triangle [0x0004]} D:{0}
Pass: S:{gles2_api_integration} T:{untextured triangle [0x0004]} F:{ [0x00]} D:{396} -_
\rightarrow(implicit pass)
Initializing: S:{gles2_api_integration} T:{textured triangle [0x0005]} D:{0}
System time: Fri Aug 2 20:49:41 2024
Running: S:{gles2_api_integration} T:{textured triangle [0x0005]} D:{0}
Terminating: S:{gles2_api_integration} T:{textured triangle [0x0005]} D:{0}
Pass: S:{gles2_api_integration} T:{textured triangle [0x0005]} F:{ [0x00]} D:{432} -_
\rightarrow(implicit pass)
Initializing: S:{gles2_api_integration} T:{untextured triangle one ibo with diff types_
\rightarrow [0x0006] } D: {0}
System time: Fri Aug 2 20:49:41 2024
Running: S:{gles2_api_integration} T:{untextured triangle one ibo with diff types_
\rightarrow [0x0006] } D: {0}
Terminating: S:{gles2_api_integration} T:{untextured triangle one ibo with diff types_
\rightarrow [0x0006] } D: {0}
Pass: S:{gles2_api_integration} T:{untextured triangle one ibo with diff types [0x0006]}_
\rightarrowF:{ [0x00]} D:{467} - (implicit pass)
Initializing: S:{gles2_api_integration} T:{HW SHA1 crypto extension [0x0007]} D:{0}
System time: Fri Aug 2 20:49:42 2024
Running: S:{gles2_api_integration} T:{HW SHA1 crypto extension [0x0007]} D:{0}
Terminating: S:{gles2_api_integration} T:{HW SHA1 crypto extension [0x0007]} D:{0}
  _____
UTF: Running gles2_api_integration_large_fbo
Initializing: S:{gles2_api_integration_large_fbo} T:{glReadPixels_partial [0x0001]} D:{0}
System time: Fri Aug 2 20:49:42 2024
Running: S:{gles2_api_integration_large_fbo} T:{glReadPixels_partial [0x0001]} D:{0}
Terminating: S:{gles2_api_integration_large_fbo} T:{glReadPixels_partial [0x0001]} D:{0}
Pass: S:{gles2_api_integration_large_fbo} T:{glReadPixels_partial [0x0001]} F:{ [0x00]}
\rightarrowD:{406} - (implicit pass)
_____
UTF: Running gles3_api_integration
(output trancated)
UTF: Result Summary
All assertions passed
 21 tests considered
```

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.

1.7.18 GPU EGL Integration tests

```
console:/data/nativetest/unrestricted # ./mali_egl_integration_tests
UTF: Platform Provenance
  Hardware: TDRX r0p0
  OS:
            Android
UTF: Running egl_surface_suite
_____
Initializing: S:{egl_surface_suite} T:{surface_eglCreateWindowSurface_defaults [0x0000]}_
\rightarrow D: \{\emptyset\}
    99.932162] [ T173] servicemanager: Found android.hardware.graphics.allocator.
Г
→IAllocator/default in device VINTF manifest.
    99.932458] [ T173] servicemanager: Found android.hardware.graphics.allocator.
→IAllocator/default in device VINTF manifest.
Running: S:{eql_surface_suite} T:{surface_eqlCreateWindowSurface_defaults [0x0000]} D:{0}
Terminating: S:{egl_surface_suite} T:{surface_eglCreateWindowSurface_defaults [0x0000]}_
\rightarrow D: \{\emptyset\}
Pass: S:{egl_surface_suite} T:{surface_eglCreateWindowSurface_defaults [0x0000]} F:{_
\rightarrow [0x00] } D: {256} - (implicit pass)
Initializing: S:{egl_surface_suite} T:{surface_drawing_to_window_surface_with_GLES_
\rightarrow [0x0001] } D: {0}
Running: S:{egl_surface_suite} T:{surface_drawing_to_window_surface_with_GLES [0x0001]}_
\rightarrow D: \{\emptyset\}
[ 230.164353][ T184] type=1400 audit(1712332638.569:611): avc: denied { getattr }_
→ for comm="RenderEngine" name="/" dev="dmabuf" ino=1 scontext=u:r:surfaceflinger:s0.
ucontext=u:object_r:unlabeled:s0 tclass=filesystem permissive=1
Terminating: S:{egl_surface_suite} T:{surface_drawing_to_window_surface_with_GLES_
\rightarrow [0x0001] } D: {0}
Pass: S:{egl_surface_suite} T:{surface_drawing_to_window_surface_with_GLES [0x0001]} F:{___
→[0x00]} D:{142255} - (implicit pass)
                                                                               (continues on next page)
```

```
Initializing: S:{egl_surface_suite} T:{surface_drawing_to_pbuffer_surface_with_GLES_
\rightarrow [0x0002] } D: {0}
Running: S:{egl_surface_suite} T:{surface_drawing_to_pbuffer_surface_with_GLES [0x0002]}_
\rightarrow D: \{\emptyset\}
Terminating: S:{egl_surface_suite} T:{surface_drawing_to_pbuffer_surface_with_GLES_
\rightarrow [0x0002] } D: {0}
Pass: S:{egl_surface_suite} T:{surface_drawing_to_pbuffer_surface_with_GLES [0x0002]} F:
→{ [0x00]} D:{39324} - (implicit pass)
Initializing: S:{egl_surface_suite} T:{surface_drawing_to_pbuffer_surface_with_GLES_
\rightarrow 16bit_config [0x0003]} D:{0}
Running: S:{egl_surface_suite} T:{surface_drawing_to_pbuffer_surface_with_GLES_16bit_
\rightarrow config [0x0003]} D:{0}
Terminating: S:{egl_surface_suite} T:{surface_drawing_to_pbuffer_surface_with_GLES_16bit_
\rightarrow config [0x0003]} D:{0}
Pass: S:{egl_surface_suite} T:{surface_drawing_to_pbuffer_surface_with_GLES_16bit_config_
→[0x0003]} F:{ [0x00]} D:{1699} - (implicit pass)
Initializing: S:{egl_surface_suite} T:{surface_drawing_to_window_surface_with_GLES3_
\rightarrow context [0x0004]} D:{0}
Running: S:{egl_surface_suite} T:{surface_drawing_to_window_surface_with_GLES3_context_
\rightarrow [0x0004]} D:{0}
Terminating: S:{egl_surface_suite} T:{surface_drawing_to_window_surface_with_GLES3_
\rightarrow context [0x0004]} D:{0}
Pass: S:{egl_surface_suite} T:{surface_drawing_to_window_surface_with_GLES3_context_
→[0x0004]} F:{ [0x00]} D:{152582} - (implicit pass)
Initializing: S:{egl_surface_suite} T:{surface_stability_simple_content_nonfs [0x0005]}_
→D:{0}
Running: S:{egl_surface_suite} T:{surface_stability_simple_content_nonfs [0x0005]} D:{0}
surface_stability_simple_content: Swapped 8 frames in 5654331408 nanosecs (1.415 fps)
surface_stability_simple_content: Swapped 7 frames in 6200350568 nanosecs (1.129 fps)
surface_stability_simple_content: Swapped 5 frames in 5483275000 nanosecs (0.912 fps)
surface_stability_simple_content: Swapped 6 frames in 6126052032 nanosecs (0.979 fps)
surface_stability_simple_content: Swapped 6 frames in 5480066528 nanosecs (1.095 fps)
surface_stability_simple_content: Total swapped 33 frames in 30079161696 nanosecs (1.097
\rightarrow fps)
Terminating: S:{egl_surface_suite} T:{surface_stability_simple_content_nonfs [0x0005]} D:
→{0}
Pass: S:{egl_surface_suite} T:{surface_stability_simple_content_nonfs [0x0005]} F:{_
→[0x00]} D:{31180} - (implicit pass)
Initializing: S:{egl_surface_suite} T:{surface_stability_simple_content_fs [0x0006]} D:
\leftrightarrow \{0\}
Running: S:{egl_surface_suite} T:{surface_stability_simple_content_fs [0x0006]} D:{0}
surface_stability_simple_content: Swapped 30 frames in 5035421000 nanosecs (5.958 fps)
surface_stability_simple_content: Swapped 28 frames in 5051054472 nanosecs (5.543 fps)
surface_stability_simple_content: Swapped 29 frames in 5011064568 nanosecs (5.787 fps)
surface_stability_simple_content: Swapped 29 frames in 5142590568 nanosecs (5.639 fps)
surface_stability_simple_content: Swapped 28 frames in 5050210456 nanosecs (5.544 fps)
surface_stability_simple_content: Total swapped 171 frames in 30067253608 nanosecs (5.
→687 fps)
Terminating: S:{egl_surface_suite} T:{surface_stability_simple_content_fs [0x0006]} D:{0}
Pass: S:{egl_surface_suite} T:{surface_stability_simple_content_fs [0x0006]} F:{ [0x00]}
→D:{30976} - (implicit pass)
Initializing: S:{egl_surface_suite} T:{surface_stability_simple_content_nonfs_checked_
→[0x0007]} D:{0}
                                                                              (continues on next page)
```

```
Running: S:{egl_surface_suite} T:{surface_stability_simple_content_nonfs_checked_
\rightarrow [0x0007] } D: {0}
surface_stability_simple_content: Total swapped 239 frames in 300293819600 nanosecs (0.
\rightarrow796 fps)
Terminating: S:{egl_surface_suite} T:{surface_stability_simple_content_nonfs_checked_
\rightarrow [0x0007] } D: {0}
Pass: S:{egl_surface_suite} T:{surface_stability_simple_content_nonfs_checked [0x0007]}_
→F:{ [0x00]} D:{301339} - (implicit pass)
Initializing: S:{egl_surface_suite} T:{surface_stability_simple_content_fs_checked_
\rightarrow [0x0008]} D:{0}
Running: S:{egl_surface_suite} T:{surface_stability_simple_content_fs_checked [0x0008]}_
\rightarrow D: \{\emptyset\}
surface_stability_simple_content: Total swapped 270 frames in 301065171768 nanosecs (0.
→897 fps)
Terminating: S:{egl_surface_suite} T:{surface_stability_simple_content_fs_checked_
\rightarrow [0x0008] } D: {0}
Pass: S:{egl_surface_suite} T:{surface_stability_simple_content_fs_checked [0x0008]} F:{___
→[0x00]} D:{302138} - (implicit pass)
Initializing: S:{egl_surface_suite} T:{surface_drawing_to_pixmap_surface_with_GLES_
\rightarrow [0x0009] } D: {0}
Running: S:{egl_surface_suite} T:{surface_drawing_to_pixmap_surface_with_GLES [0x0009]}_
\rightarrow D: \{\emptyset\}
Terminating: S:{egl_surface_suite} T:{surface_drawing_to_pixmap_surface_with_GLES_
\rightarrow [0x0009] } D: {0}
Initializing: S:{egl_surface_suite} T:{surface_yuv_android_recordable [0x000a]} D:{0}
Running: S:{egl_surface_suite} T:{surface_yuv_android_recordable [0x000a]} D:{0}
Testing MALI_TPI_FORMAT_YV12_BT601_NARROW
Testing MALI_TPI_FORMAT_YV12_BT601_WIDE
Testing MALI_TPI_FORMAT_YV12_BT709_NARROW
Testing MALI_TPI_FORMAT_YV12_BT709_WIDE
(...)
```

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.

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

1.8 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

• Troubleshooting: common problems and solutions



1.8.1 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.

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

1.9 Release notes - TC23.1

- * Development Host OS Support
- Known issues
- Support

1.9.1 Release tag

The manifest tag for this release is TC23.1.

1.9.2 Platform Support

• This software release is tested on TC3 Fixed Virtual Platform (FVP) version 11.26.16.

1.9.3 Components

The following is a summary of the key components of the release:

- Board Support Package (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 and Internal Trusted Storage) 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, and so on;
- Runtime Security Engine (RSE) previously known as Runtime Security SubSystem (RSS) firmware for providing hardware Root-of-Trust (RoT);
- TensorFlow Lite Machine Learning.

1.9.4 Hardware Features

This software release provides the following high-level hardware features:

- Arm® Tower MCN and NCI Interconnect with Memory Tagging Unit (MTU) support driver in SCP firmware;
- Arm® CoreLink[™] GIC-700 Generic Interrupt Controller in Trusted Firmware-A;
- Mali-G725 GPU;
- Arm® Mali[™]-D71 Display Processor and virtual encoder support for display on Linux;
- MHUv3 Driver for SCP and Application Processor (AP) communication;
- UARTs, Timers, Flash, Clock drivers, CCSM (Clock Control State Machine);
- PL180 MMC;
- DynamIQ Shared Unit (DSU) with 8 cores (2x Cortex-X925 + 4x Cortex-A725 + 2x Cortex-A520 cores configuration);
- Cortex®-M55-based Runtime Security Engine (RSE);
- Cortex®-M85-based System Control Processor (SCP).

1.9.5 Software Features

- Buildroot distribution support;
- Debian 12 (aka Bookworm);
- Android 14 and Android 13 support;
- Android Common Kernel 6.1.25;
- Android Kernel is built with Bazel (instead of Make) which is referred to Kleaf;
- Android Hardware Rendering with Mali-G725 GPU DDK r49p0_00eac0 (source code or prebuilt binaries);
- Android Software rendering with DRM Hardware Composer offloading composition to Mali D71 DPU;
- Arm implementation for Hardware Composer 3 is used for Android Hardware Rendering with Mali-G725 GPU DDK r49p0_00eac0 (source code or prebuilt binaries);
- Hardware Graphics Composer 3 Ranchu is used for Software Rendering (implementation is tweaked);
- 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;
- ADB connection from host machine to protected VM Microdroid;
- GPU and DPU support for S1 and S2 translation squashed with SMMU-700;
- Maximum Power Mitigation Mechanism (MPMM) support;
- GPU Dynamic Voltage Frequency Scaling (DVFS)/Idle power states support;
- Support for Memory System Resource Partitioning and Monitoring (MPAM) (see link);
- Support for Energy Aware Scheduling (EAS) (see link);
- Trusted Firmware-A v2.10;
- Hafnium v2.10 as Secure Partition Manager (SPM) at S-EL2;
- OP-TEE 4.2.0 as Secure Partition at S-EL1, managed by S-EL2 SPMC (Hafnium), support in Buildroot distribution. This includes OP-TEE client and OP-TEE test suite;
- Trusty with FF-A messaging FF-A v1.0;
- Tower Interconnect PMU's enabled for profiling;
- Support for secure boot based on Trusted Boot Board Requirements (TBBR) specification (see link);
- System Control Processor (SCP) firmware v2.14;
- Runtime Security Engine (RSE) firmware v2.0.0;
- U-Boot bootloader v2024.02;
- Power management features: cpufreq and cpuidle;
- System Control and Management Interface (SCMI) 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;
- Arm FF-A driver and FF-A Transport support for OP-TEE driver in Android Common Kernel;
- Trusted Services (Crypto and Internal Trusted Storage) running at S-EL0;

- Trusted Services test suite added to Buildroot distribution;
- PAC/BTI is enabled in Hafnium, OP-TEE and Trusted Services;
- Tracing support, based on ETE and TRBE v1.1 in TF-A, kernel and simpleperf. Traces can be captured with simpleperf;
- DICE Protection Environment (DPE) support;
- Full-HD (1920x1080-60fps) resolution support for use with the FVP model.

1.9.6 Tools Support

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

1.9.7 Optimizations

U-Boot boot time optimization

To speed up the boot process, you can interrupt the auto-boot process:

- 1. When the terminal displays Hit any key to stop autoboot: X, press ENTER.
- 2. At the resulting command prompt, type boot and press ENTER. This continues the boot process. Although the configured delay is 1-3 seconds, it takes considerably longer (approximately 15 seconds) because of the time difference between the CPU frequency and the FVP operating frequency.

1.9.8 Limitations

Development Host OS Support

Ubuntu 22.04 is not supported in this release;

1.9.9 Known issues

- 1. Ray tracing is currently not supported by the GPU DDK (hardware rendering);
- 2. The Kernel Selftest sanity test reports a failure for the check_ksm_options test as illustrated on the following excerpt. This is expected as the KSM driver is not part of the TC3 kernel.

```
(...)
# selftests: arm64: check_ksm_options
not ok 3 selftests: arm64: check_ksm_options # exit=1
(...)
```

3. When running the EAS for LISA, the test TwoBigThreeSmall:test_task_placement may fail with an output similar to the following:

```
(...)
TwoBigThreeSmall[board=tc]:test_task_placement
    UUID=17fc2798916b4b1f9dae399c20dd3e63 FAILED
energy threshold: 2472495.7617728077 bogo-joules
estimated energy: 2699550.767562539 bogo-joules
```
(continued from previous page)

```
noisiest task:
    comm: sshd
    duration (abs): 0.00029410398565232754 s
    duration (rel): 0.029381488120311203 %
    pid: 175
(...)
```

- 4. For Android builds which use the TAP network interface, the default browser available in Android (webview_shell) is not able to open HTTPS URLs. You can work around this limitation by getting the ARM64 specific Android Application Package (APK) package for other browsers (for example, Mozilla Firefox), installing it using Android Debug Bridge (ADB), and using it to browse HTTPS URLs;
- 5. The Android PAUTH sanity test may sometimes report inconsistent failing test results (this behaviour is currently under investigation). If experiencing this situation, repeat the test a few times to validate the feature;
- 6. TensorFlow application (benchmark_model) needs to download extra dependencies during the build process, which may lead to a failure because of network related issues. If experiencing this issue, try to rebuild TensorFlow application alone a few times with ./run_docker.sh ./build-ml-app.sh clean build deploy to finish the build.
- 7. The TC FVP model running Android may report a crash if the TensorFlow application benchmark_model is interrupted during execution. To prevent this situation, please wait until the TensorFlow application has finished executing.
- 8. When running the GPU GLES Integration tests, one of the tests belonging to the gles3_api_integration set may fail with output similar to the following:

```
(...)
UTF: Running gles3_api_integration
(...)
Initializing: S:{gles3_api_integration} T:{afrc_sample [0x0006]} D:{0}
System time: Fri Apr 5 15:54:57 2024
Running: S:{gles3_api_integration} T:{afrc_sample [0x0006]} D:{0}
Fail: S:{gles3_api_integration} T:{afrc_sample [0x0006]} F:{ [0x00]} D:{294}
\rightarrow - file!=((void*)0) fail [0x0!=0x0] (<unknown>)
Terminating: S:{gles3_api_integration} T:{afrc_sample [0x0006]} D:{0}
(...)
_____
UTF: Result Details
_____
(...)
Info: S:{gles3_api_integration} T:{afrc_render [0x0005]} F:{ [0x00]} D:
⇔{9000} - -----
Fail: S:{gles3_api_integration} T:{afrc_sample [0x0006]} F:{ [0x00]} D:{294}
\rightarrow - file!=((void*)0) fail [0x0!=0x0] (<unknown>)
(...)
```

9. When running the GPU EGL Integration tests, some tests may fail, resulting in output similar to the following:

```
(...)
____
                      _____
UTF: Result Details
(...)
Fail: S:{egl_surface_suite} T:{surface_depth_readback_valid_swap [0x0015]}_
\rightarrowF:{ [0x00]} D:{11441} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_depth_readback_valid_swap [0x0015]}_
→F:{ [0x00]} D:{21467} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_depth_readback_valid_swap [0x0015]}_
\rightarrowF:{ [0x00]} D:{31473} - Fail check_res != 0 fail (<unknown>)
Fail: S:{eql_surface_suite} T:{surface_depth_readback_valid_swap [0x0015]}_
\rightarrowF:{ [0x00]} D:{41673} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_depth_readback_valid_swap [0x0015]}.
→F:{ [0x00]} D:{51919} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_depth_readback_valid_swap [0x0015]}_
\rightarrowF:{ [0x00]} D:{62043} - Fail check_res != 0 fail (<unknown>)
Fail: S:{eql_surface_suite} T:{surface_depth_readback_valid_swap [0x0015]}_
\rightarrowF:{ [0x00]} D:{72164} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_depth_readback_valid_swap [0x0015]}_
\rightarrowF:{ [0x00]} D:{82463} - Fail check_res != 0 fail (<unknown>)
(...)
Fail: S:{egl_surface_suite} T:{surface_color_readback_valid_msaa_swap_
\rightarrow [0x0017]} F:{ [0x00]} D:{11605} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_color_readback_valid_msaa_swap_
\rightarrow [0x0017]} F:{ [0x00]} D:{21859} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_color_readback_valid_msaa_swap_
→ [0x0017]} F:{ [0x00]} D:{32137} - Fail check_res != 0 fail (<unknown>)
Fail: S:{eql_surface_suite} T:{surface_color_readback_valid_msaa_swap_
→ [0x0017]} F:{ [0x00]} D:{42144} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_color_readback_valid_msaa_swap_
→ [0x0017]} F:{ [0x00]} D:{52193} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_color_readback_valid_msaa_swap_
→ [0x0017]} F:{ [0x00]} D:{62272} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_color_readback_valid_msaa_swap_
→ [0x0017]} F:{ [0x00]} D:{72636} - Fail check_res != 0 fail (<unknown>)
Fail: S:{egl_surface_suite} T:{surface_color_readback_valid_msaa_swap_
→ [0x0017]} F:{ [0x00]} D:{82977} - Fail check_res != 0 fail (<unknown>)
(...)
Fail: S:{egl_image_suite} T:{afbc_bch_external_image_test [0x0008]} F:{_
\rightarrow [0x00] D:{135} - file!=((void*)0) fail [0x0!=0x0] (<unknown>)
Fail: S:{egl_image_suite} T:{afbc_usm_external_image_test [0x0009]} F:{_
(...)
Fail: S:{egl_damage_suite} T:{partial_update_entire_surface_with_buffer_age_
→zero [0x0002]} F:{ [0x00]} D:{30687} - check_res==1 fail [0==1] (<unknown>
⇒)
(...)
Fail: S:{egl_damage_suite} T:{partial_update_glReadPixel_in_damage_region_
→between_drawcalls_ms [0x0007]} F:{ [0x00]} D:{6764} - data_out==ref_color_
→out fail [00==0xff @ item 0] (<unknown>)
(...)
Fail: S:{egl_damage_suite} T:{partial_update_prerotate_fullscreen [0x000e]}_
\rightarrowF:{ [0x00]} D:{68555} - check_res==1 fail [0==1] (<unknown>)<sup>continues on next page)</sup>
```

(continued from previous page)

```
(...)
Fail: S:{egl_damage_suite} T:{partial_update_set_region_after_drawcall_
→ [0x0016]} F:{ [0x00]} D:{38224} - check_res==1 fail [0==1] (<unknown>)
(...)
UTF: Result Summary
  22
       assertions Fail
  200 tests considered
 165 tests passed
 27
      tests skipped
 0
       tests expected to fail
 8
      tests failed
      suites considered
 6
  3
       suites did not pass
 Run time 70m 7s
(...)
```

10. When running the GPU Vulkan Integration tests, the test vulkan_wsi_external_memory_dma_buf_32k_image fails and aborts execution as shown in the following output. The exact cause is currently under investigation. To work around this error and prevent the GPU Integration test failing, run the tests individually.

```
(...)
02-20 21:26:32.471 3197 3197 I mali_test: [INSTANCE EXTENSION]
\rightarrow [12] 'VK_EXT_debug_report' version 10
02-20 21:26:32.471 3197 3197 I mali_test: [INSTANCE EXTENSION]
\rightarrow [13] 'VK_EXT_debug_utils' version 2
02-20 21:26:32.471 3197 3197 I mali_test: [DEVICE QUEUE] count 1
02-20 21:26:32.471 3197 3197 I mali_test: [DEVICE QUEUE]
                                                                  [0] flags =
\rightarrow 0xc07, count = 2, timestamp valid bits = 64
02-20 21:26:32.474 3197 3197 I mali_test: Testing format R8G8B8A8, linear_
⇔case.
02-20 21:26:32.484 3197 3197 I mali_test: DRM format modifier 0:
02-20 21:26:32.484 3197 3197 I mali_test:
                                                 type = LINEAR
02-20 22:03:21.584 3197 3199 I mali_test: UTF not progressing.
02-20 22:23:21.584 3197 3199 I mali_test: UTF not progressing.
02-20 22:43:21.585 3197 3199 I mali_test: UTF not progressing after 3.
\rightarrow checks. Aborting
(...)
```

- 11. The SSH connection to fvp running Buildroot may fail due to a "Host key verification failed..." error. To resolve this issue, you can use the following command to remove the localhost entry and try again: ssh-keygen -f "\$HOME/.ssh/known_hosts" -R "[localhost]:8022"
- 12. The CPU hotplug works well with Buildroot and Debian distributions. However, in Android, CPU hotplug operations cause the system to hang. The issue is due to the integration of Hafnium and Trusty OS in the system and the PSCI CPU ON and OFF APIs are not yet supported in this case.

1.9.10 Support

For support email: support@arm.com.

Copyright (c) 2022-2024, Arm Limited. All rights reserved.

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 TC2 release tags

TC2-2023.10.04 TC2-2023.08.15 TC2-2023.04.21 TC2-2022.12.07 TC2-2022.08.12

2.2 TC1 release tags

TC1-2022.10.07 TC1-2022.05.12 TC1-2021.08.17

2.3 TC0 release tags

TC0-2022.02.25 TC0-2021.07.31 TC0-2021.04.23 TC0-2021.02.09

Copyright (c) 2022-2024, Arm Limited. All rights reserved.