arm

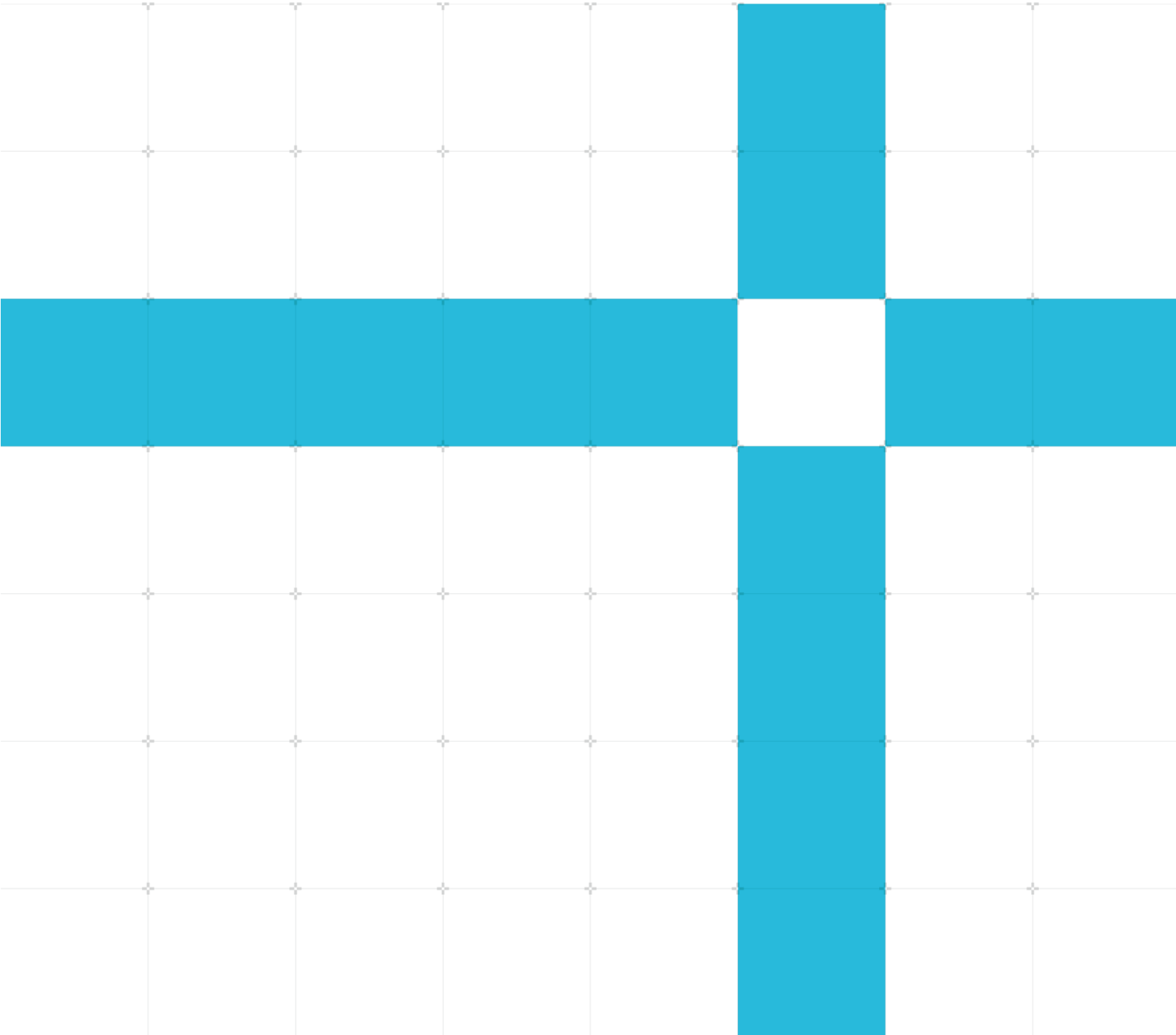
Total Compute

Version 1.0

Armv9-A OS Porting Guide

Non-Confidential

Issue 1.0
109693_1.0



Total Compute
Armv9-A OS Porting Guide

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	27-03-2024	Non-Confidential	Initial release

Non-Confidential Proprietary Notice

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

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

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

Inclusive language commitment

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

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

Table of Contents

1 Total Compute software stack components and FVP overview

1.1. Armv9-A OS porting for DTV on TC2

Arm Total Compute 2022 (TC2) provides a reference platform and software stack based on premium Arm-based mobile silicon. Because of its flexibility and scalability, you can use TC2 in non-mobile solutions such as Digital Television (DTV). This guide provides information to help you port your own software or operating system to TC2. This reduces the effort required to adapt your platform.

Software components and FVP overview

TC2 is supported by the OSS stack and FVP model generated for the reference design. The OSS stack is designed to facilitate partner software development and architecture exploration. TC2 FVP is a programmer's view model of the TC2 system architecture, supported by a basic Rest of System (RoS) implementation.

The OSS (Open-Source Software) reference stack targets the corresponding FVP and is validated in a hardware development environment. It demonstrates the following:

- Integration of multiple open-source software components.
- Upstream support for configuring the latest Arm IPs.
- Alignment with Arm specifications.
- Examples of software features including security, power, and virtualization.

The OSS stack is based on the following components:

- SCP firmware
- Trusted Firmware-A
- U-boot
- Armv9-A Linux kernel, with drivers and Device Tree-based hardware config descriptors
- Distro boot examples (Debian)
- Reference toolchain

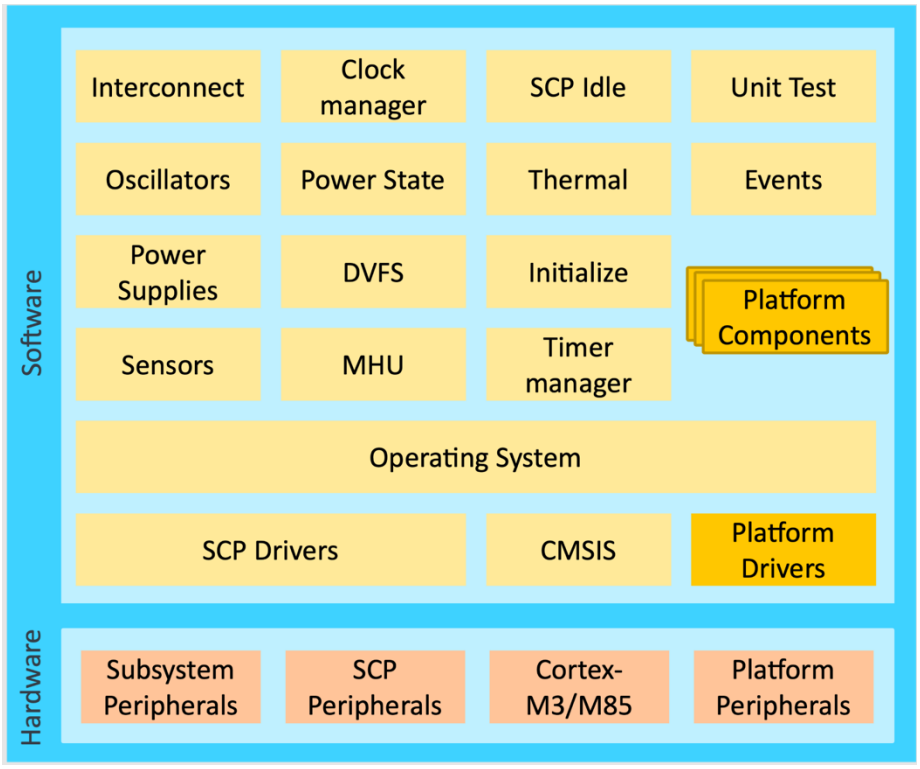
2 Porting SCP firmware

2.1 SCP firmware introduction

2.1.1 System Control Processor

The system control processor (SCP) is a processor-based capability that provides a flexible and extensible platform for the provision of power management functions and services.

Figure 2-1 SCP hardware and software stack architecture overview



2.1.2 SCP functionality

The SCP provides the following functionality:

System powerup and startup

Clock management

OS Power state requests

Wakeup events

DVFS (Dynamic Voltage and Frequency Scaling) transitions

Sensor framework

2.1.3 Hardware interfaces

The SCP provides the following hardware interfaces:

Message Handling Unit

Power Policy Unit

Sensor Monitor

System Control

2.1.4 SCP firmware directory structure

The SCP firmware directory has the following structure:

Table 2-1 SCP firmware directory structure

Folder	Description
`arch`	Architecture-specific code
`cmake`	CMake configuration files
`contrib`	External code, for example CMSIS
`debugger`	CLI debugger code
`doc`	Documentation
`framework`	Framework code
`module`	Module code
`product`	Product-specific code and configuration files
`tools`	Scripts and tools to build and validate code
`docker`	Docker image configuration files

You can find more details about SCP-firmware in the SCP-firmware User Guide[1]

2.1.5 SCP build system

The CMake build system has the following advantages:

Faster than Makefile

Possible to add multiple tools at build time

Cppcheck integration

Allows more complex build flags

Conditional build code is easier to implement

Build flags are easier to set and clear

The following code shows an example build command using CMake:

```
make -GNinja -S $tc2_workspace/build-scripts/../src/SCP-firmware -B
$tc2_workspace/build-scripts/../output/debian/tmp_build/scp//scp -DCMAKE_ASM_COMPILER=
$tc2_workspace/build-scripts/../tools/arm-gnu-toolchain-12.2.rel1-x86_64-arm-none-
eabi/bin/arm-none-eabi-gcc -DCMAKE_C_COMPILER $tc2_workspace/build-
scripts/../tools/arm-gnu-toolchain-12.2.rel1-x86_64-arm-none-eabi/bin/arm-none-eabi-gcc
-DSCP_TOOLCHAIN:STRING=GNU -DCMAKE_OBJCOPY $tc2_workspace/build-scripts/../tools/arm-
gnu-toolchain-12.2.rel1-x86_64-arm-none-eabi/bin/arm-none-eabi-objcopy -
DSCP_LOG_LEVEL=INFO -DDISABLE_CPPCHECK=true -DSCP_ENABLE_PLAT_FVP=true -
DSCP_PLATFORM_VARIANT=2 -DCMAKE_BUILD_TYPE=Release -DSCP_FIRMWARE_S
OURCE_DIR:PATH=tc2/scp_ramfw

cmake --build $tc2_workspace/build-scripts/../output/debian/tmp_build/scp//scp --
parallel 24
```

This example uses the following CMake configuration options:

- `-S`
  Source code directory
- `-B`
  Build directory
- `-DCMAKE_ASM_COMPILER`
  Location of the GNU asm compiler executable
- `-DCMAKE_C_COMPILER`
  Location of the GNU C compiler executable
- `-DCMAKE_OBJCOPY`
  Location of the objcopy executable
- `-DSCP_LOG_LEVEL`
  Sets the SCP log level. Available log levels are NONE, ERROR, NOTICE, WARNING, INFO, and VERBOSE
- `-DCMAKE_BUILD_TYPE`
  Specifies the build type. Available build types are Debug and Release

Build software stack components using included build scripts

The TC2 software stack provides a build system including component build scripts and a TC docker image for setting up a standard build environment. Build scripts are run inside the TC Docker environment which makes building more convenient than using CMAKE directly. Please note these scripts are not usable for system debugging images. Details about the TC2 software build system are available in the TC2 user guide[2].

For example, to clean, build, and deploy the SCP firmware stack, go to `$tc2_workspace/build-scripts` then run:

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

2.1.6 SCP firmware linker script for Cortex-M processors

Define SCP memory map

For each of the final build images `scp_romfw` or `scp_ramfw`, the following linker information must be provided in an `fmw_memory.h` file:

- FMW_MEM_MODE: The desired memory region configuration. The following table shows the available settings for FMW_MEM_MODE:

Table 2-2 Memory region configuration

FMW_MEM_MODE	Description
ARCH_MEM_MODE_SINGLE_REGION	Single memory region configuration
ARCH_MEM_MODE_DUAL_REGION_RELOCATION	Dual memory region configuration with read-only data section relocation to sram region
ARCH_MEM_MODE_DUAL_REGION_NO_RELOCATION	Dual memory region configuration with read-only data section executes in place

- FMW_MEM0_BASE: The base address of the MEM0 region. This setting is always used regardless of the memory region configuration specified by FMW_MEM_MODE.

- FMW_MEM0_SIZE: The size of the MEM0 region in bytes.

- FMW_MEM1_BASE: If a dual-region memory configuration is specified by FMW_MEM_MODE then FMW_MEM1_BASE and FMW_MEM1_SIZE must also be defined. FMW_MEM1_BASE is the base address of the MEM0 region.
- FMW_MEM1_SIZE: The size of the MEM1 region in bytes.

TC2 example:

```
SCP-firmware/product/tc2/scp_romfw/fmw_memory.h
```

```
#define FMW_MEM_MODE ARCH_MEM_MODE_DUAL_REGION_RELOCATION
/*
 * ROM memory
 */
#define FMW_MEM0_SIZE SCP_BOOT_ROM_SIZE
#define FMW_MEM0_BASE SCP_BOOT_ROM_BASE
/*
 * RAM memory for scp_romfw (16 KiB block at the top of the RAM)
 *
 * The last 16 KiB of SCP RAM are used for scp_romfw data regions.
 * The start of the RAM is where the scp bootloader places the scp_ramfw
 * so the 2 areas don't overlap.
 */
#define FMW_MEM1_SIZE (16 * 1024)
#define FMW_MEM1_BASE (SCP_RAM_BASE + SCP_RAM_SIZE - FMW_MEM1_SIZE)
```

Source linker script for GNU toolchain

```
SCP-firmware/arch/arm/arm-m/src/arch.ld.S
```

Source linker script for Arm Compiler

```
SCP-firmware/arch/arm/arm-m/src/arch.scatter.S
```

Example scp_romfw GNU linker script

```
/*
 * Arm SCP/MCP Software
 * Copyright (c) 2015-2022, Arm Limited and Contributors. All rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 *
 * Description:
 *      GNU LD linker script.
 */

#include <arch_scatter.h>

ENTRY(arch_exception_reset)

MEMORY {
#if FMW_MEM_MODE == ARCH_MEM_MODE_SINGLE_REGION
    /*
     * Single region memory layout:
     *  - MEM0 accepts:
     *      - Read-only sections
     *      - Read-write sections
     *      - Executable sections
     */

    mem0 (rwx) : ORIGIN = FMW_MEM0_BASE, LENGTH = FMW_MEM0_SIZE
#elif FMW_MEM_MODE == ARCH_MEM_MODE_DUAL_REGION_RELOCATION
    /*
     * Dual region memory layout with initialized data relocation:
     *  - MEM0 accepts:
     *      - Read-only sections
     *      - Executable sections
```

```
     *
     *  - MEM1 accepts:
     *      - Read-write sections
     */

    mem0 (rx) : ORIGIN = FMW_MEM0_BASE, LENGTH = FMW_MEM0_SIZE
    mem1 (w) : ORIGIN = FMW_MEM1_BASE, LENGTH = FMW_MEM1_SIZE
#elif FMW_MEM_MODE == ARCH_MEM_MODE_DUAL_REGION_NO_RELOCATION
    /*
     * Dual region memory layout without initialized data relocation:
     *  - MEM0 accepts:
     *      - Executable sections
     *
     *  - MEM1 accepts:
     *      - Read-only sections
     *      - Read-write sections
     */

    mem0 (x) : ORIGIN = FMW_MEM0_BASE, LENGTH = FMW_MEM0_SIZE
    mem1 (rw) : ORIGIN = FMW_MEM1_BASE, LENGTH = FMW_MEM1_SIZE
#endif
}

#if FMW_MEM_MODE == ARCH_MEM_MODE_SINGLE_REGION
    REGION_ALIAS("r", mem0);
    REGION_ALIAS("w", mem0);
    REGION_ALIAS("x", mem0);
#elif FMW_MEM_MODE == ARCH_MEM_MODE_DUAL_REGION_RELOCATION
    REGION_ALIAS("r", mem0);
    REGION_ALIAS("w", mem1);
    REGION_ALIAS("x", mem0);
#elif FMW_MEM_MODE == ARCH_MEM_MODE_DUAL_REGION_NO_RELOCATION
    REGION_ALIAS("r", mem1);
    REGION_ALIAS("w", mem1);
    REGION_ALIAS("x", mem0);
#endif

SECTIONS {
    /*
     * Variables defined here:
     *   - __data_load__: Load address of .data
     *   - __data_start__: Start address of .data
     *   - __data_end__: End address of .data and .data-like orphans
     *   - __bss_start__: Start address of .bss
     *   - __bss_end__: End address of .bss and .bss-like orphans
     *   - __stackheap_start__: Start address of .stackheap
     *   - __stackheap_end__: End address of .stackheap
     *   - __stack: Initial stack pointer
     */
…
```

The TC2 example uses a dual-region memory configuration with 16KB MEM0 and 16KB MEM1. The code text section and read-only data section are in MEM0 Nor flash which can execute scp_romfw in place (XIP). Writeable data in scp_romfw is in MEM1 and it will be relocated at FMW_MEM1_BASE.

2.2 SCP porting

2.2.1 Product structure hierarchy

A product in this context is a collection of firmware. You must place the files that define the product in the product directory, in the following hierarchy:

```
SCP-firmware
└──product
      └──── tc2
          ├──── include
          ├──── module
          ├──── product.mk
          ├──── scp_ramfw
          └──── scp_romfw
```

The `product.mk` file describes the product to the build system listing all build images such as `scp_romfw` which runs in place in the Nor flash memory, and `scp_ramfw` which is runtime firmware resident in RAM. In addition, a product may also contain header files that are used for product-specific definitions, for example, the definition of SCP mmap, clock systems, power domains, and timers:

```
product/tc2/include/
├── clock_soc.h
├── config_power_domain.h
├── scp_mmap.h
├── scp_tc2_mhu.h
├── system_pik.h
├── tc2_core.h
├── tc2_dvfs.h
├── tc2_power_domain.h
└── tc2_timer.h
…
```

2.2.2 SCP toolchain support

The SCP_TOOLCHAIN CMake cache variable specifies the tools used to build the firmware. Toolchain support is on a per-firmware basis, and the toolchains supported by the firmware are specified by the `Toolchain-${SCP_TOOLCHAIN}.cmake` files found in the firmware.

For example, a firmware build with both GCC and Arm Compiler 6 may offer a `GNU` toolchain option and an `ArmClang` toolchain option (`Toolchain-GNU.cmake` and `Toolchain-ArmClang.cmake`). In this situation, for GCC you might use:

```
$ cmake "${SCP_SOURCE_DIR}" -B "${SCP_BUILD_DIR}" \
  -DSCP_TOOLCHAIN:STRING="GNU"
```

Or for Arm Compiler 6:

```
$ cmake "${SCP_SOURCE_DIR}" -B "${SCP_BUILD_DIR}" \
  -DSCP_TOOLCHAIN:STRING="ArmClang"
```

2.2.3 SCP processor support

The SCP software stack is based on the ARM Common Microcontroller Software Interface Standard (CMSIS[3]) layer. The CMSIS simplifies microcontroller software development, providing a consistent and efficient interface for developers working with Cortex-M and entry-level Cortex-A, and Cortex-R processors. It promotes code reuse, portability, and interoperability, enabling developers to focus on application-level logic rather than dealing with low-level hardware details.

CMSIS provides software interfaces to processors and peripherals, real-time operating systems, and middleware components. CMSIS also includes a delivery mechanism for devices, boards, and software, and enables the combination of software components from multiple vendors.

The latest CMSIS version is 5.9.0. Check the version history for information about support for specific processors:

CMSIS version[4]

Platform specific (Toolchain-GNU.cmake for example)

TC2 SCP processor setting

```
product/tc2/scp_ramfw/Toolchain-GNU.cmake:set(CMAKE_SYSTEM_PROCESSOR "cortex-m3")
```

TC2 SCP architecture setting

```
product/tc2/scp_ramfw/Firmware.cmake:set(SCP_ARCHITECTURE "arm-m")
```

Example SCP processor setting for high end Cortex-M processors:

Cortex-M55

```
set(CMAKE_SYSTEM_PROCESSOR "cortex-m55+nodsp")
```

Cortex-M7

```
set(CMAKE_SYSTEM_PROCESSOR "cortex-m7")
```

Cortex-M85

```
set(CMAKE_SYSTEM_PROCESSOR "cortex-m85")
```

2.2.4 Add a new module to the product

To add a new module to the SCP product firmware, do either of the following:

Add a new product-specific module under the product structure.

Add a new common feature module under the module structure.

Example

The following example shows how to add a new TC2 product-specific module called tc2_system:

- Create the module folder structure under `SCP-firmware/product/tc2/module/tc2_system`. Create the module header files, source files, and document sub-directory as follows:

```
tc2
    ├─── module
    │    └─── tc2_system
    │         ├─── CMakeLists.txt
    │         ├─── Module.cmake
    │         ├─── include
    │         └─── src
```

- Create the `Module.cmake` file with the following contents, defining the SCP_MODULE and SCP_MODULE_TARGET cmake variables:

```
set(SCP_MODULE "tc2-system")
set(SCP_MODULE_TARGET "module-tc2-system")
```

- Create the CMakeLists.txt file with the following contents:

```
add_library(${SCP_MODULE_TARGET} SCP_MODULE)

target_include_directories(${SCP_MODULE_TARGET}
                           PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include")

target_sources(${SCP_MODULE_TARGET}
               PRIVATE "${CMAKE_CURRENT_SOURCE_DIR}/src/mod_tc2_system.c")

target_link_libraries(${SCP_MODULE_TARGET}
    PRIVATE module-clock module-sds module-power-domain module-ppu-v1
            module-scmi module-system-power)
```

- Enable the new module in product firmware by appending the module name to the SCP_MODULES cmake variable in `product/tc2/scp_ramfw/Firmware.cmake`.

```
list(APPEND SCP_MODULES "tc2-system")
```

  This is expanded in `CMakeLists.txt` as an SCP-Firmware library and linked to the SCP Firmware target.

Example

The following example shows how to add a new mpmm (Maximum Power Mitigation Mechanism for Armv9-A CPUs) module under the modules structure:

1. Create the module folder structure under `SCP-firmware/module/mpmm`. Create the module header files, source files, and document sub-directory as follows:

```
SCP-firmware/module/mpmm/include/
SCP-firmware/module/mpmm/src/
SCP-firmware/module/mpmm/doc/
```

2. Create the `module.cmake` file with the following contents:

```
set(SCP_MODULE "mpmm")
set(SCP_MODULE_TARGET "module-mpmm")
```

3. Create the `CMakeLists.txt` file with the following contents:

```
add_library(${SCP_MODULE_TARGET} SCP_MODULE)
target_include_directories(${SCP_MODULE_TARGET}
                           PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include")
target_include_directories(${SCP_MODULE_TARGET}
                           PRIVATE "${CMAKE_CURRENT_SOURCE_DIR}/src")
target_sources(${SCP_MODULE_TARGET}
               PRIVATE "${CMAKE_CURRENT_SOURCE_DIR}/src/mod_mpmm.c")
target_link_libraries(${SCP_MODULE_TARGET} PRIVATE module-power-domain
                                                   module-scmi-perf)
```

4. Enable the new module in product firmware by appending the module name to the SCP_MODULES cmake variable, and the module configuration source file to the firmware target source path in `product/tc2/scp_ramfw/CMakeLists.txt`.

```
    list(APPEND SCP_MODULES "mpmm")
    target_sources(tc2-bl2 PRIVATE "config_mpmm.c")
```

Note

You can enable the new module by appending the module name to the SCP_MODULES cmake variable in both `Firmware.cmake` and `CMakeLists.txt`. SCP_MODULES is expanded by the SCP firmware framework as a library, and linked to the SCP firmware target.

2.2.5 Combining modules into product firmware

The `product.mk` file defines scp_romfw and scp_ramfw firmware targets.

Each firmware target is a binary image created when building the product.

scp_romfw

The SCP boot ROM code executes after a cold reset. It performs the following functions:

- Setting up the generic timer, Universal Asynchronous Receiver/Transmitter (UART) console, and clocks system interconnect
- Powering up the primary Application Processor (AP), normally CPU0
- Copying and authenticating the SCP runtime firmware

scp_ramfw

The SCP runtime code executes after it is copied into the SCP private RAM. It performs the following functions:

- Setting up the SCP peripherals
- Responding to System Control and Management Interface (SCMI) messages through the Message Handling Unit (MHU) for processor power control and Dynamic Voltage and Frequency Scaling (DVFS)
- Handling voltage and power domain management
- Handling clock management

A firmware target is fully defined by its set of modules and their configuration data defined by the struct fwk_module_config in `scp/product/tc2/scp_romfw/CMakeLists.txt`:

```
if(SCP_ENABLE_PLAT_FVP)
    target_sources(
        tc2-bl1
        PRIVATE "${CMAKE_CURRENT_SOURCE_DIR}/config_pl011.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_ppu_v1.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_sds.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_cmn_booker.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_system_pll.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_pik_clock.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_css_clock.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_clock.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_gtimer.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_timer.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_tc2_bl1.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_bootloader.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_transport.c"
                "${CMAKE_CURRENT_SOURCE_DIR}/config_mhu2.c")
```


Note

The order of the modules in the following list is the order in which the modules are initialized, bound, and started during the pre-runtime phase. Any change in the order will cause firmware initialization errors.

The modules enabled for the TC2 scp_romfw are defined in the `scp/product/tc2/scp_romfw/Firmware.cmake` file:

```
if(SCP_ENABLE_PLAT_FVP)
    list(APPEND SCP_MODULES "pl011")
    list(APPEND SCP_MODULES "ppu-v1")
    list(APPEND SCP_MODULES "tc2-bl1")
    list(APPEND SCP_MODULES "bootloader")
    list(APPEND SCP_MODULES "system-pll")
    list(APPEND SCP_MODULES "pik-clock")
    list(APPEND SCP_MODULES "css-clock")
    list(APPEND SCP_MODULES "clock")
    list(APPEND SCP_MODULES "gtimer")
    list(APPEND SCP_MODULES "timer")
    list(APPEND SCP_MODULES "cmn-booker")
    list(APPEND SCP_MODULES "sds")
    list(APPEND SCP_MODULES "mhu2")
    list(APPEND SCP_MODULES "transport")
```

3 TF-A firmware porting

3.1 TF-A overview

The Trusted Firmware-A project provides a reference implementation of secure world software for Armv8-A and Armv9-A class processors.

3.1.1 TF-A design

For more information about the features and design of the TF-A firmware, refer to the firmware-design[5] document.

3.1.2 TF-A for TC Platform

The platforms TC0 (TARGET_PLATFORM=0), TC1 (TARGET_PLATFORM=1), and TC2 (TARGET_PLATFORM=2) support different CPUs:

TC0 supports Cortex A510, Cortex A710 and Cortex X2. Note that TC0 is now deprecated.

TC1 supports Cortex A510, Cortex A715 and Cortex X3. Note that TC1 is now deprecated.

TC2 supports Cortex A520, Cortex A720 and Cortex X4.

For more information about the TF-A support for TC platform, refer to the TC Platform[6] documentation.

3.2 Build system

This guide uses the software stack that corresponds to the TC2-2023.10.04 release tag as the example. Refer to the TC2 software user guide[7] for information about installing the software stack. The resulting directory has the following structure:

```
build-scripts output run-scripts src tc-venv tools
```

The build system is included in the `build-scripts` directory. Specifically for TF-A, the `build-scripts/config/common.config` file contains the configuration described below. This is a common configuration that is shared with other platforms:

```
#Trusted-Firmware A
TFA_COMPILER=$AARCH64_BARE_METAL/aarch64-none-elf
TFA_SRC=$SRC_DIR/trusted-firmware-a
TFA_OUTDIR=$OUTPUT_DIR/tmp_build/tfa
TFA_FILES=$FILES_DIR/tfa
TFA_OPENSSL_DIR=$TOOLS_DIR/openssl
TFA_SP_DIR=$OUTPUT_DIR/tmp_build/tfa_sp
```

For the TC2 platform, there is a platform-defined configuration in the `$tc2_workspace/build-scripts/config/tc2.config` file as follows:

```
# TF-A
TFA_PLATFORM=tc
TC_VERSION=2
TC_FWU_SUPPORT=1
TFA_DEBUG=1
# Log levels are LOG_LEVEL_NONE, LOG_LEVEL_ERROR, LOG_LEVEL_NOTICE, LOG_LEVEL_WARNING,L
OG_LEVEL_INFO, LOG_LEVEL_VERBOSE
TFA_LOG_LEVEL=LOG_LEVEL_NOTICE
make_opts_tfa=(
        PLAT=$TFA_PLATFORM
        TARGET_VERSION=$TC_VERSION
        TARGET_PLATFORM=$TC_TARGET_FLAVOR
        ARCH=aarch64
        BL33=$OUTPUT_DIR/tmp_build/u-boot/u-boot.bin
        BL32=$OUTPUT_DIR/tmp_build/hafnium/secure_tc_clang/hafnium.bin
        SCP_BL2="$SCP_OUTDIR/scp/bin/tc2-bl2.bin"
        MBEDTLS_DIR="$SRC_DIR/mbedtls"
        SPD=spmd
        SPMD_SPM_AT_SEL2=1
        CTX_INCLUDE_EL2_REGS=1
        LD_LIBRARY_PATH=$TFA_OPENSSL_DIR/lib:$LD_LIBRARY_PATH
        CROSS_COMPILE=$TFA_COMPILER-
        TRUSTED_BOARD_BOOT=1
        GENERATE_COT=1
        ARM_ROTPK_LOCATION=devel_rsa
        ROT_KEY="$TFA_SRC/plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem"
        DEBUG=$TFA_DEBUG
        LOG_LEVEL=$TFA_LOG_LEVEL
        ARM_ARCH_MINOR=7
        HOSTCC=gcc
        CTX_INCLUDE_PAUTH_REGS=1
        BRANCH_PROTECTION=1
        ENABLE_SVE_FOR_NS=2
        ENABLE_SVE_FOR_SWD=1
        ENABLE_MPAM_FOR_LOWER_ELS=1
        CTX_INCLUDE_MTE_REGS=1
        OPENSSL_DIR=$TFA_OPENSSL_DIR
)
```

The TC platform version is defined by the build options:

- TFA_PLATFORM is `tc`
- TC_VERSION is `2`,
- TC_TARGET_FLAVOR is `fvp`.

```
PLAT=$TFA_PLATFORM
TARGET_VERSION=$TC_VERSION
TARGET_PLATFORM=$TC_TARGET_FLAVOR
```

For example, if you need to modify or add more build options for the TC platform, you can modify the `$tc2_workspace/build-scripts/config/tc2.config` file. For example, the TFA_LOG_LEVEL is defined with the following values:

```
#define LOG_LEVEL_NONE                  U(0)
#define LOG_LEVEL_ERROR                 U(10)
#define LOG_LEVEL_NOTICE                U(20)
#define LOG_LEVEL_WARNING               U(30)
#define LOG_LEVEL_INFO                  U(40)
#define LOG_LEVEL_VERBOSE               U(50)

TFA_LOG_LEVEL=LOG_LEVEL_NOTICE
```

This example uses LOG_LEVEL_NOTICE as the default. If you need to dump more detailed log information, you can change the default to LOG_LEVEL_VERBOSE. Because the value is bigger, more log information is generated.

3.3 Porting guidelines

Porting Trusted Firmware-A (TF-A) to a new platform involves making some mandatory and optional modifications for both the cold and warm boot paths. Modifications could include the following:

Implementing a platform-specific function or variable

Setting up the execution context in a certain way

Defining certain constants (for example #defines)

The platform-specific functions and variables are declared in the `trusted-firmware-a/include/plat/common/platform.h` file. The firmware provides a default implementation of variables and functions to fulfill the optional requirements to ease the porting effort. Each platform port can use them as is or provide their own implementation if the default implementation is inadequate.

Only Arm development platforms such as the FVP and the TC platform can use the functions and definitions in `trusted-firmware-a/include/plat/arm/common/` and the corresponding source files in `trusted-firmware-a/plat/arm/common/`.

Specifically for the TC platform:

The related platform specific source files can be found in `trusted-firmware-a/plat/arm/board/tc/`.

The related platform defined constants are implemented in `trusted-firmware-a/plat/arm/board/tc/include/platform_def.h`.

The configuration of the makefile is in `trusted-firmware-a/plat/arm/board/tc/platform.mk`.

For information about the API list for generic TF-A porting requirements, refer to the TF-A porting guide[8] .

3.4 CPU and architecture

3.4.1 CPU

The TC platforms support hardware cache coherence management, based on the TC platform version. The related specific CPU support is specified in the `trusted-firmware-a/plat/arm/board/tc/platform.mk` file as follows:

```
# System coherency is managed in hardware
HW_ASSISTED_COHERENCY   :=      1
# CPU libraries for TARGET_VERSION=1
ifeq (${TARGET_VERSION}, 1)
TC_CPU_SOURCES  +=      lib/cpus/aarch64/cortex_a510.S \
                        lib/cpus/aarch64/cortex_a715.S \
                        lib/cpus/aarch64/cortex_x3.S
endif
# CPU libraries for TARGET_VERSION=2
ifeq (${TARGET_VERSION}, 2)
TC_CPU_SOURCES  +=      lib/cpus/aarch64/cortex_a520.S \
                        lib/cpus/aarch64/cortex_a720.S \
                        lib/cpus/aarch64/cortex_x4.S
endif
```

The TC2 platform supports Cortex-A520, Cortex-A720, and Cortex-X4 CPUs.

3.4.2 Architecture

The TC2 CPUs implement the Armv9.2-A architecture. The Armv9.2-A architecture extends the architecture defined in the Armv8-A architectures up to Armv8.7A. The related specific architecture support for the TC platform is defined by macros in the `trusted-firmware-a plat/arm/board/tc/platform.mk` file.

Mandatory architecture features

By default, for the TF-A, the architecture is defined as ARMv8.0 in the `trusted-firmware-a /make_helpers/defaults.mk` file:

```
# ARM Architecture major and minor versions: 8.0 by default.
ARM_ARCH_MAJOR                  := 8
ARM_ARCH_MINOR                  := 0
```

The architecture of ARMv8.7-A/ARMv9.2-A on the TC2 platform is enabled by the following build option in `$tc2_workspace/build-scripts/config/tc2.config`:

```
ARM_ARCH_MINOR=7
```

All the mandatory architectural features up to `ARM_ARCH_MAJOR.ARM_ARCH_MINOR` are included and unconditionally enabled by the TF-A build system. More information is included in the `trusted-firmware-a/make_helpers/arch_features.mk` file:

```
# Enable the features which are mandatory from ARCH version 8.1 and upwards.
ifeq "8.1" "$(word 1, $(sort 8.1 $(ARM_ARCH_MAJOR).$(ARM_ARCH_MINOR)))"
ENABLE_FEAT_PAN         =       1
ENABLE_FEAT_VHE         =       1
endif
# Enable the features which are mandatory from ARCH version 8.2 and upwards.
ifeq "8.2" "$(word 1, $(sort 8.2 $(ARM_ARCH_MAJOR).$(ARM_ARCH_MINOR)))"
ENABLE_FEAT_RAS         =       1
endif
# Enable the features which are mandatory from ARCH version 8.4 and upwards.
ifeq "8.4" "$(word 1, $(sort 8.4 $(ARM_ARCH_MAJOR).$(ARM_ARCH_MINOR)))"
ENABLE_FEAT_DIT         =       1
```

```
ENABLE_FEAT_SEL2        =       1
endif
# Enable the features which are mandatory from ARCH version 8.5 and upwards.
ifeq "8.5" "$(word 1, $(sort 8.5 $(ARM_ARCH_MAJOR).$(ARM_ARCH_MINOR)))"
ENABLE_FEAT_SB          =       1
endif
# Enable the features which are mandatory from ARCH version 8.6 and upwards.
ifeq "8.6" "$(word 1, $(sort 8.6 $(ARM_ARCH_MAJOR).$(ARM_ARCH_MINOR)))"
ENABLE_FEAT_FGT         =       1
ENABLE_FEAT_ECV         =       1
endif
# Enable the features which are mandatory from ARCH version 8.7 and upwards.
ifeq "8.7" "$(word 1, $(sort 8.7 $(ARM_ARCH_MAJOR).$(ARM_ARCH_MINOR)))"
ENABLE_FEAT_HCX         =       1
endif
```

Optional architecture features

The following table lists the optional architectural feature configurations available:

Table 3-1: Optional architecture feature configurations

Feature Control Option	Value for TC2	**Additional Information**
BRANCH_PROTECTION	1	Enable ARMv8.3 Pointer Authentication and ARMv8.5 Branch Target Identification support for TF-A BL images themselves
ENABLE_SVE_FOR_NS	2	Enable Scalable Vector Extension for non-secure world.
ENABLE_SVE_FOR_SWD	1	Enable Scalable Vector Extension for secure world.
ENABLE_MPAM_FOR_LOWER_ELS	1	Enable lower ELs to use the optional ARMv8.4 MPAM feature.
CTX_INCLUDE_MTE_REGS	1	Enable including Memory Tagging Extension registers in the CPU context. The platform uses this feature in the Secure world and MTE is enabled at ELx.

3.5 Platform

3.5.1 Platform memory layout

The firmware uses secure boot ROM, secure SRAM, and DRAM in the TC2 platform, accessing memory resources as shown in the following table:

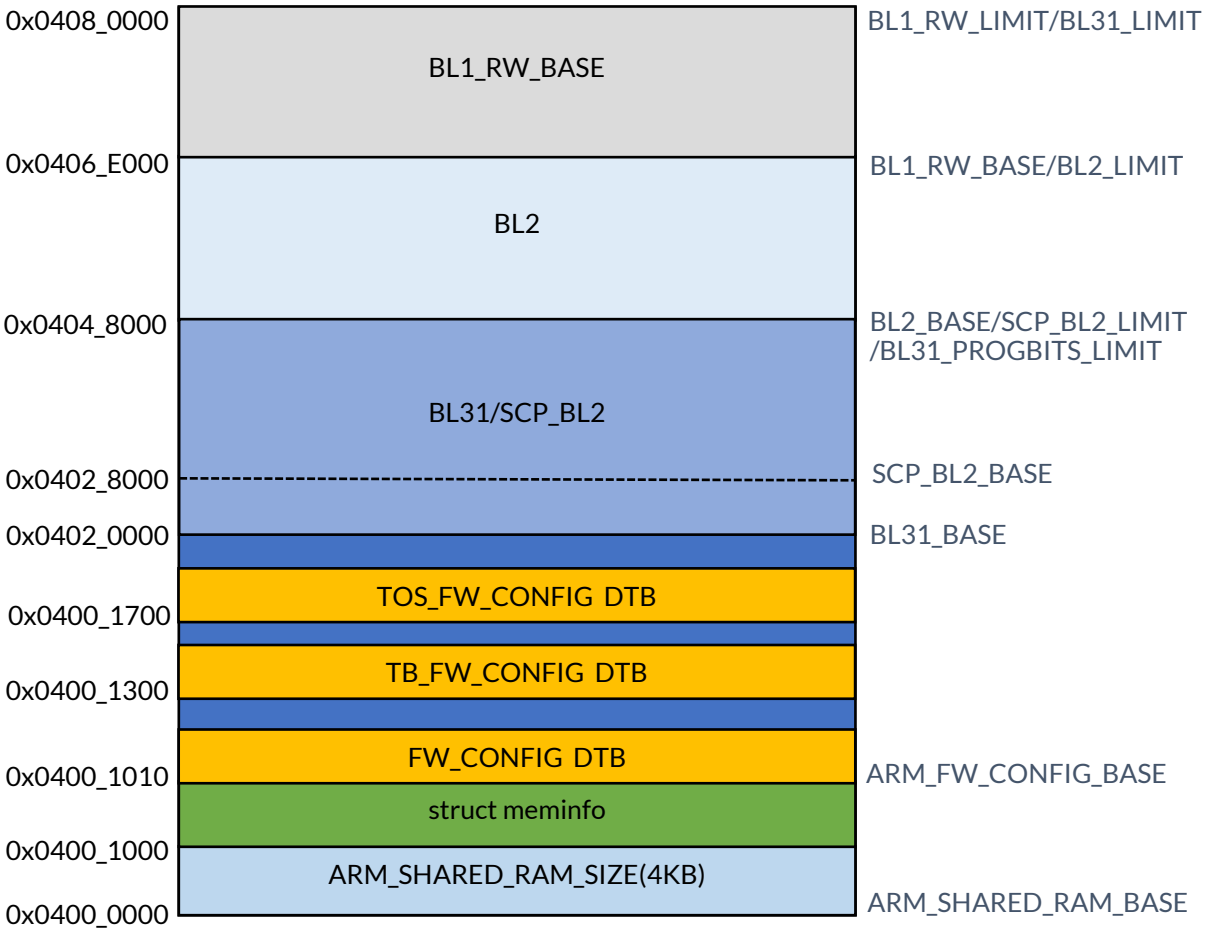
Table 3-2: Memory Map on TC2 Platform

Start address	End address	Region	Additional Information
0x0000_0000	0x0007FFFF	Secure Boot ROM	512KB
0x04000000	0x0407FFFF	Secure SRAM	512KB
0x80000000	0xFFFFFFFF	DRAM1	2GB

Specifically, BL1 runs from the 0x0 address, which is the secure ROM address. The BL2/BL31 image uses Secure SRAM, and the BL32 and other secure partition packages are loaded into Secure DRAM, which is protected by the TZC400 Trust Zone controller.

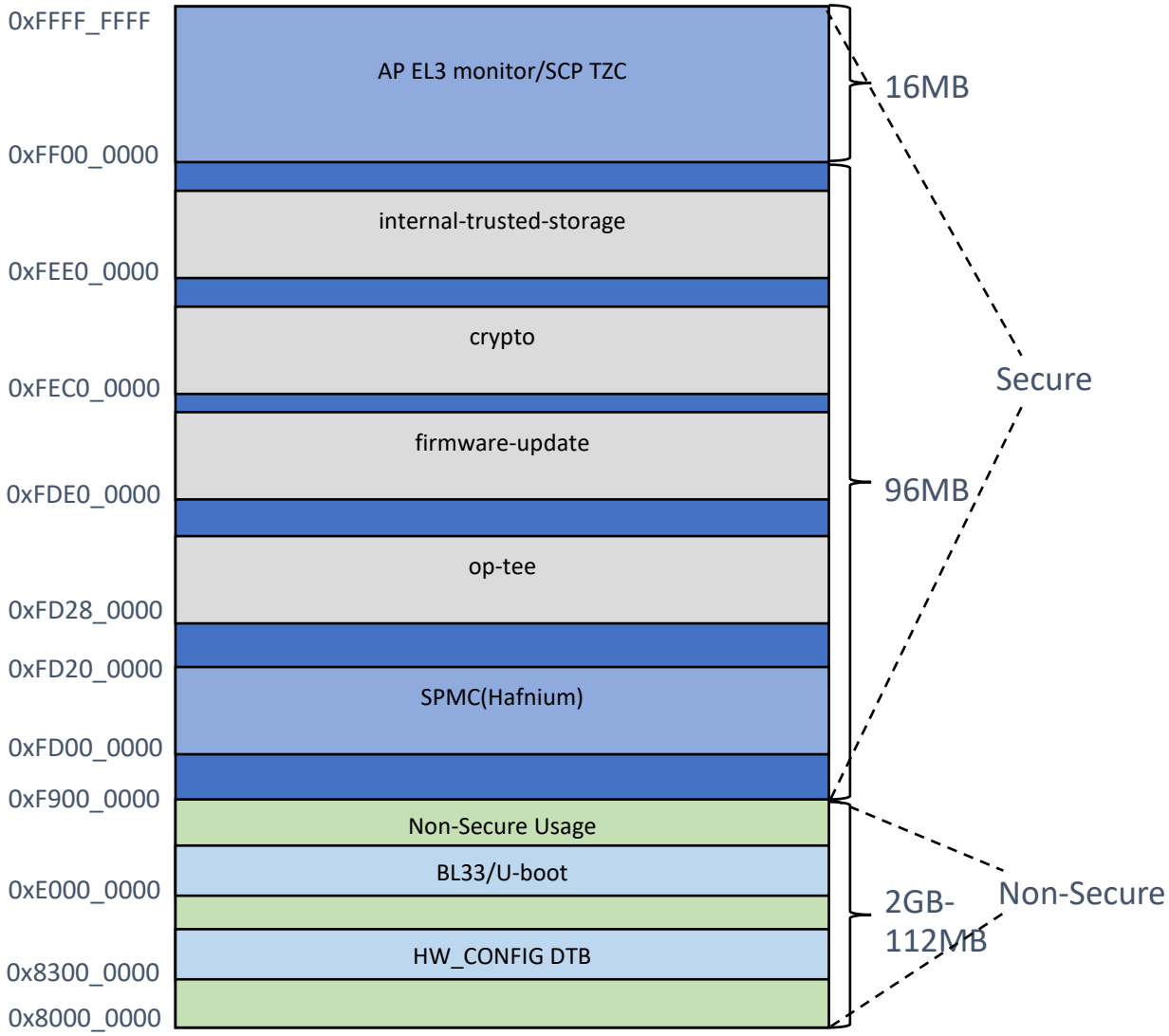
The following figure shows SRAM usage:

Figure 31 -Secure SRAM usage on the TC2 platform



The DRAM usage is as below:

Figure 3-2 DRAM Usage by Firmware on TC2 Platform



The related address definitions can be found in the `trusted-firmware-a /plat/arm/board/tc/include/platform_def.h` and `trusted-firmware-a /include/plat/arm/common/arm_def.h` files.

The DTB's address is defined in the `trusted-firmware-a/plat/arm/board/tc/fdts/tc_fw_config.dts` file, and the secure partition package loading address can is defined in the `trusted-firmware-a /plat/arm/board/tc/fdts/tc_tb_fw_config.dts` file.

3.5.2 UART

The TC22 specification defines secure and non-secure UART as shown in the following table:

Table 3-3: UART memory map on the TC2 platform

Start address	End address	Region	Additional Information
0x00_2A40_0000	0x00_2A40_FFFFF	Non-secure Universal Asynchronous Receiver/Transmitter (UART)	-
0x00_2A41_0000	0x00_2A41_FFFFF	Secure UART	-

For the TC platform, the UART is redefined compared to other ARM CSS platforms. The secure UART is used for the TF-A boot console, which is the uart1_ap terminal for the TC FVP platform. The TF-A runtime console uses the same console with Linux, which is the uart_ap terminal for the TC FVP platform.

In the TF-A, UART is defined in the `trusted-firmware-a/plat/arm/board/tc/include/platform_def.h` file as follows:

```
#ifdef TARGET_PLATFORM_FVP
#undef PLAT_ARM_BOOT_UART_BASE
#define PLAT_ARM_BOOT_UART_BASE         0x2A410000

#undef PLAT_ARM_RUN_UART_BASE
#define PLAT_ARM_RUN_UART_BASE          0x2A400000
```

TF-A uses a different UART between boot and runtime, and the same UART with Linux when system is in the runtime stage. You can use the UART to output debug log information about the interaction between TF-A and Linux, as defined in DTS in the `trusted-firmware-a/fdts/tc.dts` file :

```
        ap_ns_uart: uart@2A400000 {
                compatible = "arm,pl011", "arm,primecell";
                reg = <0x0 0x2A400000 0x0 0x1000>;
                interrupts = <0x0 63 0x4>;
                clocks = <&soc_uartclk>, <&soc_refclk100mhz>;
                clock-names = "uartclk", "apb_pclk";
                status = "okay";
        };
```

3.5.3 GIC interrupt controller

The TC platform includes the GIC700 component as the interrupt controller. It supports GICv3, GICv3.1, and GICv4.1. The TF-A enables the related driver support with the `trusted-firmware-a/plat/arm/board/tc/platform.mk` build options:

```
GIC_ENABLE_V4_EXTN      :=      1
# GIC-600 configuration
GICV3_SUPPORT_GIC600    :=      1
# Include GICv3 driver files
include drivers/arm/gic/v3/gicv3.mk
```

```
ENT_GIC_SOURCES         :=      ${GICV3_SOURCES}                \
                                plat/common/plat_gicv3.c        \
                                plat/arm/common/arm_gicv3.c
```

The GIC600 and GIC700 share some power management function routines, so the GICV3_SUPPORT_GIC600 is used. GIC_ENABLE_V4_EXTN is for the GICv4 related change.

The GIC driver is listed below:

```
GICV3_SOURCES   +=      drivers/arm/gic/v3/gicv3_main.c         \
                        drivers/arm/gic/v3/gicv3_helpers.c      \
                        drivers/arm/gic/v3/gicdv3_helpers.c     \
                        drivers/arm/gic/v3/gicrv3_helpers.c
GICV3_SOURCES   +=      drivers/arm/gic/v3/gic-x00.c
```

3.6 Device tree

3.6.1 Device tree introduction

Each of the Boot Loader stages may be dynamically configured if required by the platform. The Boot Loader stage can optionally specify a firmware configuration file and/or hardware configuration file as listed below:

FW_CONFIG - The firmware configuration file. Holds properties shared across all BLx images. An example is the dtb-registry node, which contains the information about the other device tree configurations including load-address, size, and image_id. It is set to `tc_fw_config.dts` for the TC platform.

HW_CONFIG - The hardware configuration file. Can be shared by all Boot Loader stages and also by the Normal World Rich OS, it is set to `tc.dts` for the TC platform.

TB_FW_CONFIG - Trusted Boot Firmware configuration file. Shared between BL1 and BL2, it is set to `tc_tb_fw_config.dts` for the TC platform.

SOC_FW_CONFIG - SoC Firmware configuration file used by BL31, Not required for the TC platform.

TOS_FW_CONFIG – SPM (Hafnium) configuration file. Set to `tc_spmc_optee_sp_manifest.dts` or `tc_spmc_trusty_sp_manifest.dts` for the TC platform.

NT_FW_CONFIG - Non-Trusted Firmware configuration file. Used by non-trusted firmware (BL33), Not required for the TC platform.

The following example, shows the device tree usage for the TC platform:

```
/* BL1 load, used for load SPs by TC platform */
./plat/arm/board/tc/fdts/tc_tb_fw_config.dts
/* BL2 load for TFTF usage  */
./plat/arm/board/tc/fdts/tc_spmc_manifest.dts
/* BL2 load for Android Trusty usage */
./plat/arm/board/tc/fdts/tc_spmc_trusty_sp_manifest.dts
/* BL2 load for OPTEE usage */
./plat/arm/board/tc/fdts/tc_spmc_optee_sp_manifest.dts
/* BL1 load, used by TC platform  */
./plat/arm/board/tc/fdts/tc_fw_config.dts
/* HW config DTS for kernel usage, used by TC FVP platform */
./fdts/tc.dts
/* HW fpga config DTS for kernel usage */
./fdts/tc_fpga.dts
```

The Image ID for the DTB definition is as follows:

```
/* FW_CONFIG, BL1 load, used by TC platform */
#define FW_CONFIG_ID                    U(31)
/* HW_CONFIG (e.g. Kernel DT), Loaded by BL2, but aligned with uboot on the loading add
res, to let Kernel know, used by TC platform */
#define HW_CONFIG_ID                    U(23)
/* TB_FW_CONFIG,  BL2 use it to load SP package, used for TC platform */
#define TB_FW_CONFIG_ID                 U(24)
/* SOC_FW_CONFIG , BL31 use if needed, NOT use for TC*/
#define SOC_FW_CONFIG_ID                U(25)
/* TOS_FW_CONFIG ,  BL32 use, it is SPMC manifest for TC platform*/
#define TOS_FW_CONFIG_ID                U(26)
/* NT_FW_CONFIG,  BL33 use, NOT used for TC*/
#define NT_FW_CONFIG_ID                 U (27)
```

3.6.2 Enabling DTB for the TC Platform

The required DTS files need to be defined in the platform makefile. For example, `trusted-firmware-a/plat/arm/board/tc/platform.mk` for the TC platforms defines the following:

```
FDT_SOURCES     +=  ${TC_BASE}/fdts/${PLAT}_fw_config.dts   \
                ${TC_BASE}/fdts/${PLAT}_tb_fw_config.dts
FW_CONFIG       :=  ${BUILD_PLAT}/fdts/${PLAT}_fw_config.dtb
TB_FW_CONFIG        :=  ${BUILD_PLAT}/fdts/${PLAT}_tb_fw_config.dtb

ifeq (${SPD},spmd)
ifeq ($(ARM_SPMC_MANIFEST_DTS),)
ARM_SPMC_MANIFEST_DTS   :=  ${TC_BASE}/fdts/${PLAT}_spmc_manifest.dts
endif

FDT_SOURCES     +=  ${ARM_SPMC_MANIFEST_DTS}
TC_TOS_FW_CONFIG    :=  ${BUILD_PLAT}/fdts/$(notdir $(basename ${ARM_SPMC_MANIFEST_DTS}
)).dtb

#Device tree
TC_HW_CONFIG_DTS    :=  fdts/tc.dts
TC_HW_CONFIG        :=  ${BUILD_PLAT}/fdts/${PLAT}.dtb
FDT_SOURCES     +=  ${TC_HW_CONFIG_DTS}
```

3.6.3 DTB used by the Linux kernel

The HW_CONFIG file is used for Linux kernel. The DTS file for the TC platform is `trusted-firmware-a/fdts/tc.dts`. The corresponding DTB is loaded by BL2, at the load address defined in `tc_fw_config.dts`:

```
hw-config {
    load-address = <0x0 0x83000000>;
    max-size = <0x8000>;
    id = <HW_CONFIG_ID>;
};
```

The load address is 0x83000000, which is the part of the DRAM address for the TC platform according to the TC2 specification.

U-boot uses this address to pass information to the Linux kernel. For example, u-boot defines this address in `trusted-firmware-a/include/configs/total_compute_fvp.h`:

```
#define CFG_EXTRA_ENV_SETTINGS  \
                                "bootm_size=0x20000000\0"       \
                                "load_addr=0xa0000000\0"        \
                                "kernel_addr_r=0x80080000\0"    \
                                "initrd_addr_r=0x88000000\0"    \
                                "fdt_addr_r=0x83000000\0"
```

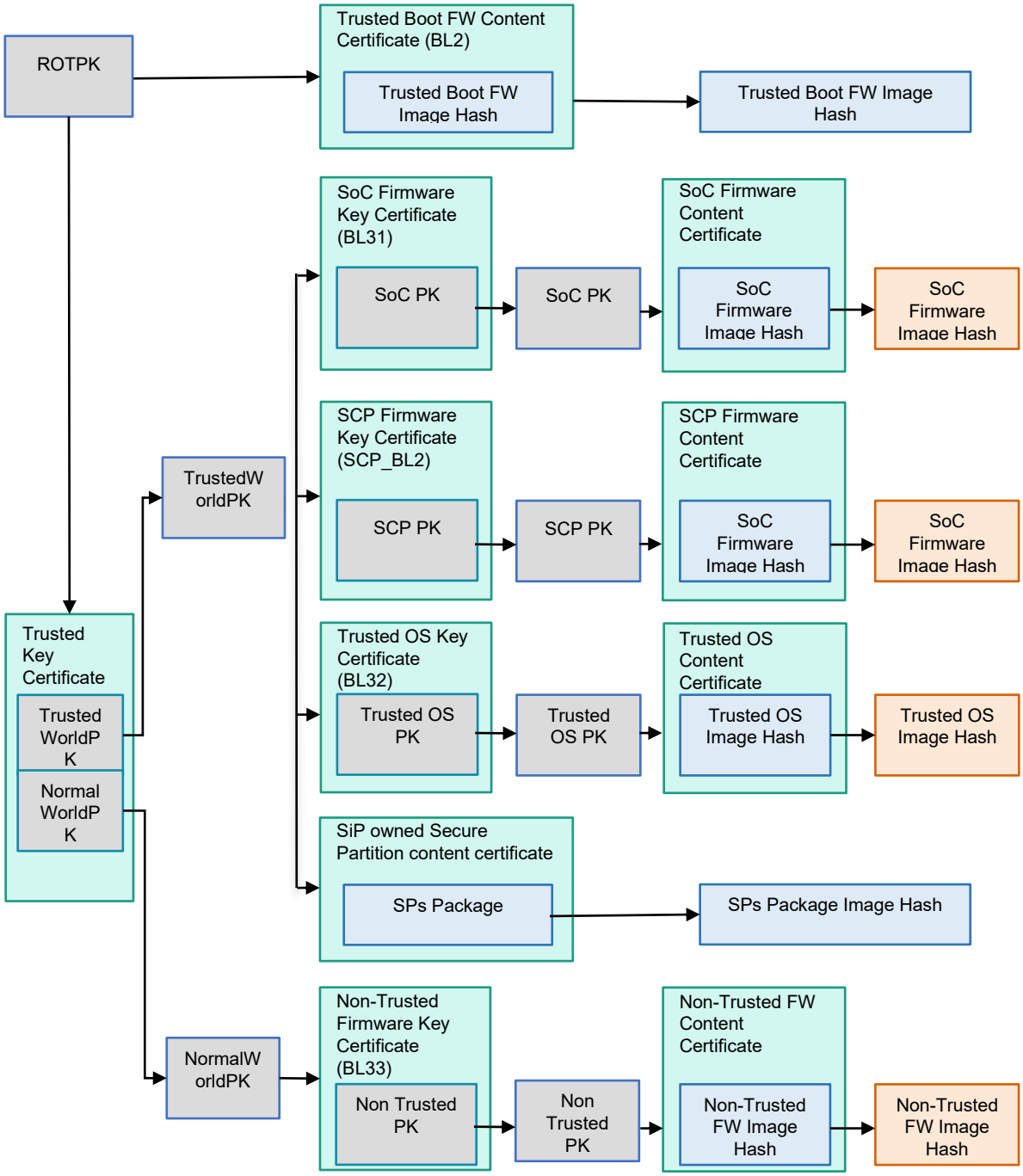
3.7 Secure boot

The Trusted Board Boot (TBB) feature prevents malicious firmware from running on the platform by authenticating all firmware images up to and including the normal world bootloader. The TF-A supports a reference implementation of the Trusted Board Boot Requirements (TBBR)[9] specification, Arm DEN0006D.

3.7.1 Chain of trust

The certificates of the chain of trust used by TF-A are as follows:

Figure 3-3 Certificates of the chain of trust



3.7.2 Enabling TBBR

The TC platform enables the TBBR feature with the build option defined in the `$tc2_workspace/build-scripts/config/tc2.config` file:

```
MBEDTLS_DIR="$SRC_DIR/mbedtls"
LD_LIBRARY_PATH=$TFA_OPENSSL_DIR/lib:$LD_LIBRARY_PATH
TRUSTED_BOARD_BOOT=1
GENERATE_COT=1
ARM_ROTPK_LOCATION=devel_rsa
ROT_KEY="$TFA_SRC/plat/arm/board/common/rotpk/arm_rotprivk_rsa.pem"
OPENSSL_DIR=$TFA_OPENSSL_DIR
```

3.7.3 Related certificate images

For the TC platform, the related certificate images are built as follows:

```
Trusted key certificate: offset=0x11ED9E, size=0x616, cmdline="--trusted-key-cert"
SCP Firmware key certificate: offset=0x11F3B4, size=0x4E2, cmdline="--scp-fw-key-cert"
SoC Firmware key certificate: offset=0x11F896, size=0x4E2, cmdline="--soc-fw-key-cert"
Trusted OS Firmware key certificate: offset=0x11FD78, size=0x4F0, cmdline="--tos-fw-
key-cert"
Non-Trusted Firmware key certificate: offset=0x120268, size=0x4F3, cmdline="--nt-fw-
key-cert"
Trusted Boot Firmware BL2 certificate: offset=0x12075B, size=0x4BE, cmdline="--tb-fw-
cert"
SCP Firmware content certificate: offset=0x120C19, size=0x3F1, cmdline="--scp-fw-cert"
SoC Firmware content certificate: offset=0x12100A, size=0x438, cmdline="--soc-fw-cert"
Trusted OS Firmware content certificate: offset=0x121442, size=0x4D6, cmdline="--tos-
fw-cert"
Non-Trusted Firmware content certificate: offset=0x121918, size=0x449, cmdline="--nt-
fw-cert"
SiP owned Secure Partition content certificate: offset=0x121D61, size=0x600, cmdline="-
-sip-sp-cert"
```

3.8 Power management

The TF-A implementation of the PSCI API is based on the concept of a power domain. A power domain is a CPU or a logical group of CPUs which share a state on which power management operations can be performed as specified by the PSCI[10].

3.8.1 TC Platform topology

Each CPU in the system is assigned a CPU index which is a unique number between 0 and PLATFORM_CORE_COUNT - 1.

The power domains are arranged in a hierarchical tree structure. Each power domain can be identified by the CPU index of any CPU that is part of that domain, and a power domain level. A processing element (for example, a CPU) is at level 0. If the power domain node above a CPU is a logical grouping of CPUs that share some state, then level 1 is that group of CPUs (for example, a cluster), and level 2 is a group of clusters (for example, the system).

For the TC platform, the system power domain tree is defined in `trusted-firmware-a/plat/arm/board/tc/tc_topology.c`.

For a specific platform, you must implement `plat_arm_calc_core_pos` to get the CPU index. For example, the TC platform implements this function in `trusted-firmware-a/plat/arm/board/tc/include/tc_helpers.S`.

3.8.2 CSS platform PSCI OPS

Each platform needs to port the platform specific function `plat_setup_psci_ops` to support PSCI power management. The function is only called by the primary CPU.

The `plat_setup_psci_ops` function is called by PSCI initialization code. It does the following:

- Notifies the platform layer about the warm boot entry point using the first argument, sec_entrypoint.
- Exports the handler routines for platform specific PSCI power management actions using a pointer to BL31's private `plat_psci_ops` structure.

If the platform you are using includes the SCP for power management, it uses the CSS routine defined in the `trusted-firmware-a/plat/arm/css/common/css_common.mk` make file as follows:

```
BL31_SOURCES            +=      plat/arm/css/common/css_pm.c                    \
                                plat/arm/css/common/css_topology.c
```

The `plat_psci_ops` structure for the TCSS platform is as follows:

```
plat_psci_ops_t plat_arm_psci_pm_ops = {
    .pwr_domain_on      = css_pwr_domain_on,
    .pwr_domain_on_finish   = css_pwr_domain_on_finish,
    .pwr_domain_on_finish_late = css_pwr_domain_on_finish_late,
    .pwr_domain_off     = css_pwr_domain_off,
    .cpu_standby        = css_cpu_standby,
    .pwr_domain_suspend = css_pwr_domain_suspend,
    .pwr_domain_suspend_finish  = css_pwr_domain_suspend_finish,
    .system_off     = css_system_off,
```

```
    .system_reset       = css_system_reset,
    .validate_power_state   = css_validate_power_state,
    .validate_ns_entrypoint = arm_validate_psci_entrypoint,
    .translate_power_state_by_mpidr = css_translate_power_state_by_mpidr,
    .get_node_hw_state  = css_node_hw_state,
    .get_sys_suspend_power_state = css_get_sys_suspend_power_state,

#if defined(PLAT_ARM_MEM_PROT_ADDR)
    .mem_protect_chk    = arm_psci_mem_protect_chk,
    .read_mem_protect   = arm_psci_read_mem_protect,
    .write_mem_protect  = arm_nor_psci_write_mem_protect,
#endif
#if CSS_USE_SCMI_SDS_DRIVER
    .system_reset2      = css_system_reset2,
#endif
};
```

If the platform you are using does not include the SCP, the platform needs to implement the specific operation of the `plat_psci_ops` structure. For example, the BASE FVP platform adds the support into the `trusted-firmware-a/plat/arm/board/fvp/fvp_pm.c` file.

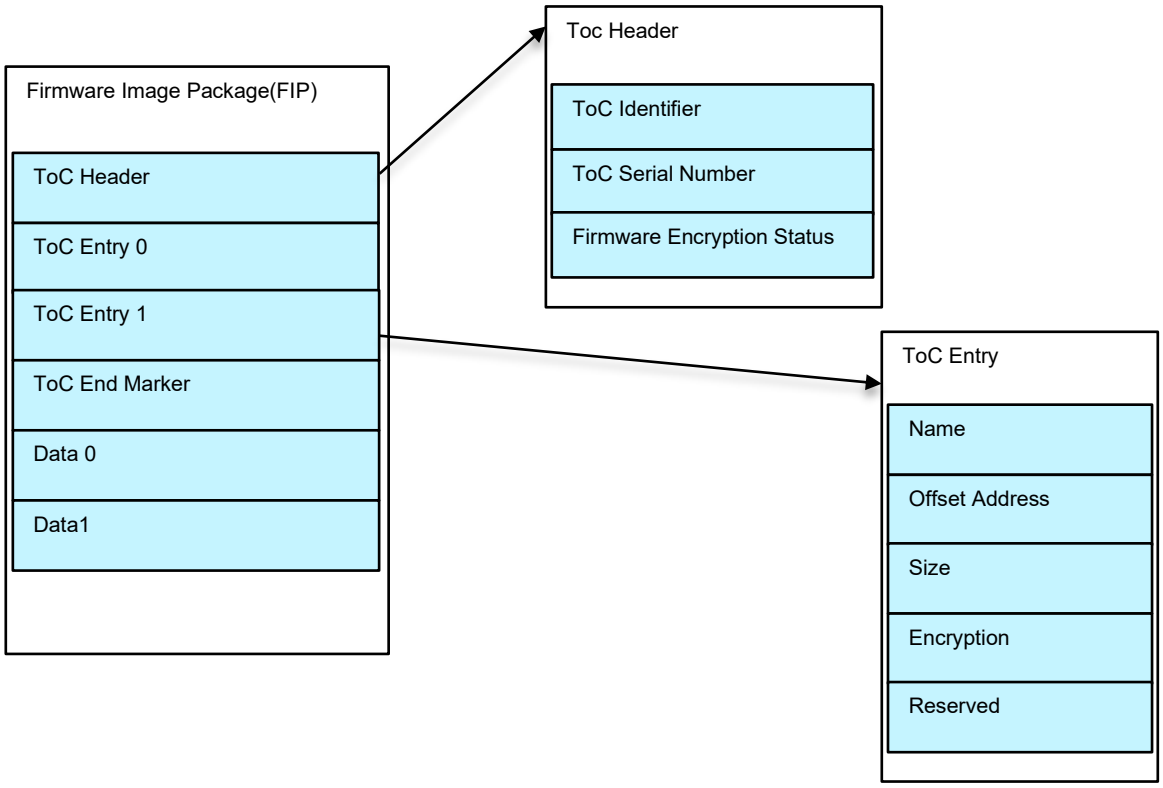
3.9 Firmware image

The TC2 platform includes the SCP and AP processor, both processors have their own separated firmware images. The Trusted firmware-A is for AP processor, the build script of TC2 platform software stack will package these firmware images together as one package, the platform can load the package into memory to run when system boot.

3.9.1 TF-A FIP format

For TF-A, you can use a Firmware Image Package (FIP) to create a single archive with bootloader images, and potentially other payloads as well. TF-A then loads this image from non-volatile platform storage. For more information about the FIP format, see the FIP document[11]. The following figure details the FIP format.

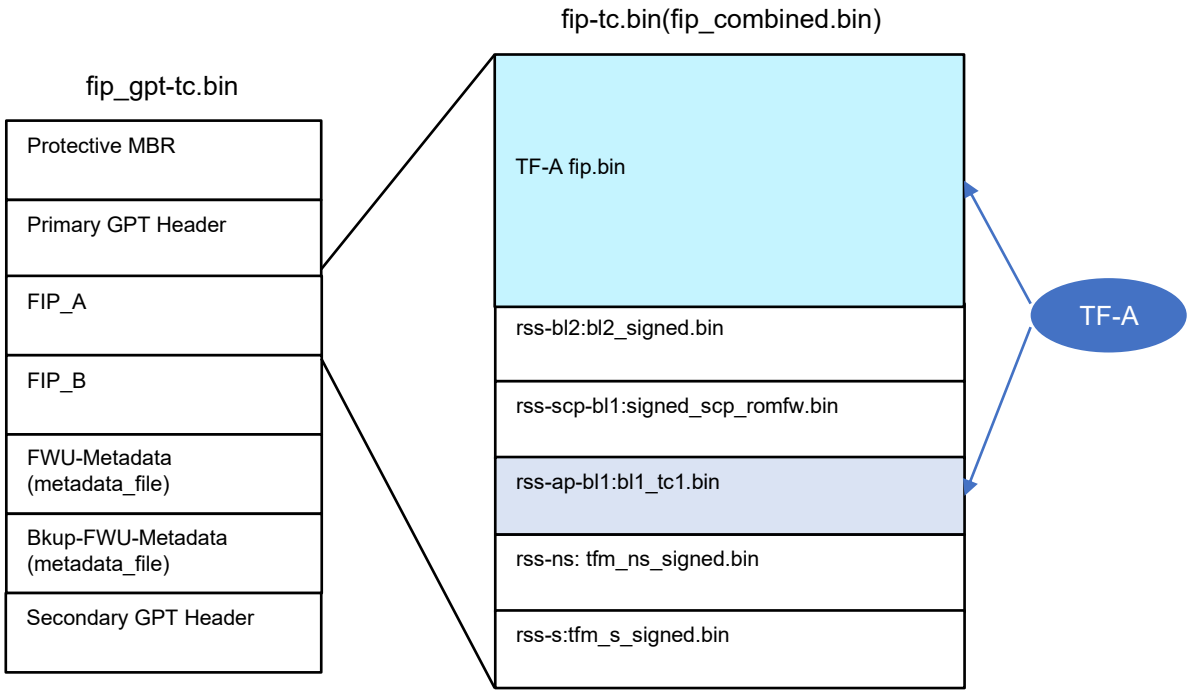
Figure 3-4: FIP format



3.9.2 TC firmware image

The following diagram shows the TC layout of the resulting GPT image file `$tc2workspace/output/deploy/fip_gpt-tc.bin`.

Figure 3-5: TC2 FIP GPT image



The TF-A code base is used to create the `fip_bin` file. The `fip_combined.bin` file is created by adding `$tc2_workspace/output/tmp_build/tfa/build/tc/debug/fip.bin` and `rss binaries`. The metadata_file is `$tc2workspace/output/tmp_build/metadata.bin`.

The TF-A `fip.bin` is built using a command defined in `$tc2workspace/build-scripts/build-tfa.sh`:

```
“make "${make_opts_tfa[@]}" "${make_opts_tfa_optee[@]}" all fip”
```

The BL1/BL2 and BL31 images are built from the TF-A code base. However, the BL32/BL33 and SCP BL2 are not in the TF-A code base, so their locations are defined as follows:

```
BL33=$OUTPUT_DIR/tmp_build/u-boot/u-boot.bin
      BL32=$OUTPUT_DIR/tmp_build/hafnium/secure_tc_clang/hafnium.bin
SCP_BL2="$SCP_OUTDIR/scp/bin/tc2-bl2.bin"
```

The following example shows the TF-A `fip.bin` being created on the TC platform:

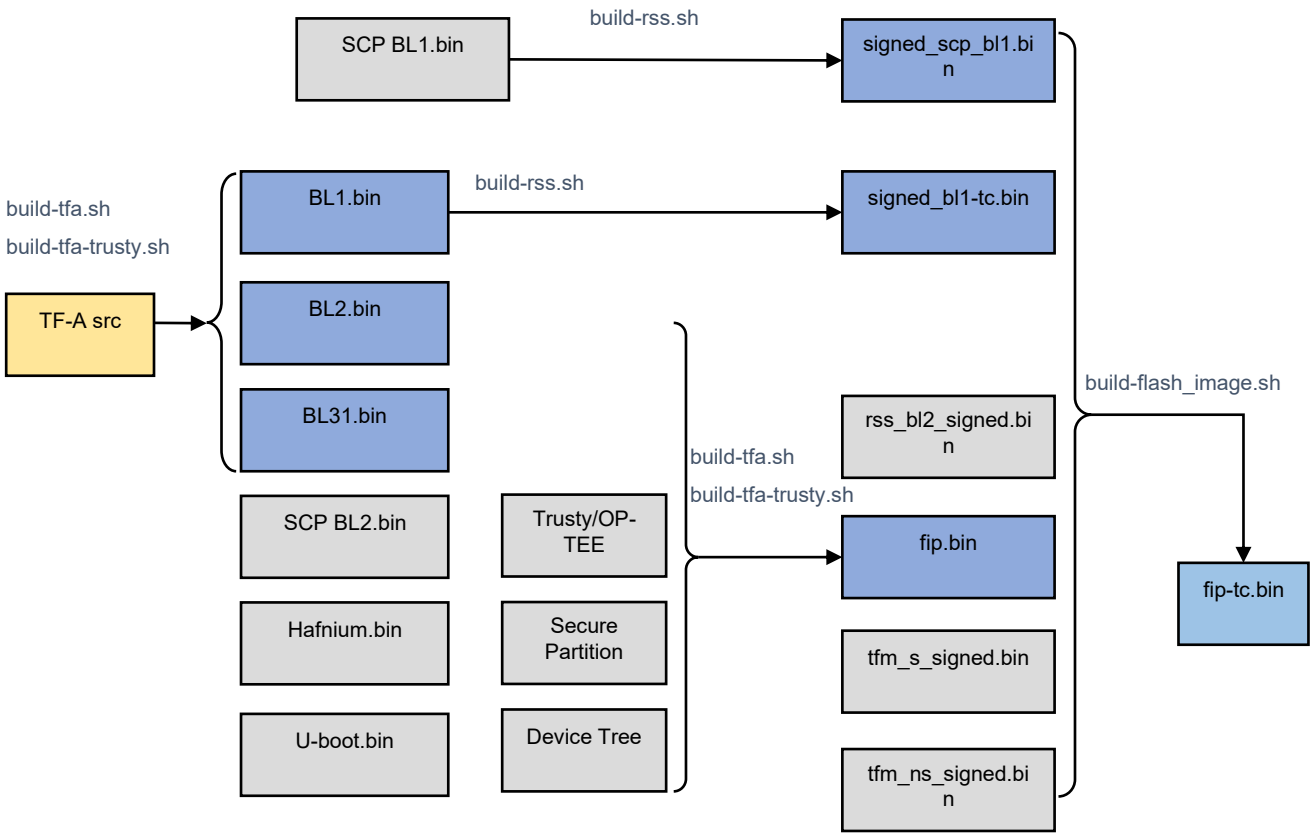
Figure 3-6: TC2 TF-A FIP image list

```
Trusted Boot Firmware BL2: offset=0x3F8, size=0x14ED1, cmdline="--tb-fw"
SCP Firmware SCP_BL2: offset=0x152C9, size=0x161F4, cmdline="--scp-fw"
EL3 Runtime Firmware BL31: offset=0x2B4BD, size=0x101D1, cmdline="--soc-fw"
Secure Payload BL32 (Trusted OS): offset=0x3B68E, size=0x2E2D0, cmdline="--tos-fw"
Non-Trusted Firmware BL33: offset=0x6995E, size=0xB1F30, cmdline="--nt-fw"
FW_CONFIG: offset=0x11B88E, size=0x18C, cmdline="--fw-config"
HW_CONFIG: offset=0x11BA1A, size=0x297F, cmdline="--hw-config"
TB_FW_CONFIG: offset=0x11E399, size=0x296, cmdline="--tb-fw-config"
TOS_FW_CONFIG: offset=0x11E62F, size=0x76F, cmdline="--tos-fw-config"
Trusted key certificate: offset=0x11ED9E, size=0x616, cmdline="--trusted-key-cert"
SCP Firmware key certificate: offset=0x11F3B4, size=0x4E2, cmdline="--scp-fw-key-cert"
SoC Firmware key certificate: offset=0x11F896, size=0x4E2, cmdline="--soc-fw-key-cert"
Trusted OS Firmware key certificate: offset=0x11FD78, size=0x4F0, cmdline="--tos-fw-key-cert"
Non-Trusted Firmware key certificate: offset=0x120268, size=0x4F3, cmdline="--nt-fw-key-cert"
Trusted Boot Firmware BL2 certificate: offset=0x12075B, size=0x4BE, cmdline="--tb-fw-cert"
SCP Firmware content certificate: offset=0x120C19, size=0x3F1, cmdline="--scp-fw-cert"
SoC Firmware content certificate: offset=0x12100A, size=0x438, cmdline="--soc-fw-cert"
Trusted OS Firmware content certificate: offset=0x121442, size=0x4D6, cmdline="--tos-fw-cert"
Non-Trusted Firmware content certificate: offset=0x121918, size=0x449, cmdline="--nt-fw-cert"
SiP owned Secure Partition content certificate: offset=0x121D61, size=0x600, cmdline="--sip-sp-cert"
DC1EEF48-B17A-4CCF-AC8B-DFCFF7711B14: offset=0x122361, size=0x12346, cmdline="--blob"
D9DF52D5-16A2-4BB2-9AA4-D26D3B84E8C0: offset=0x1346A7, size=0x97D5E, cmdline="--blob"
6823A838-1B06-470E-9774-0CCE8BFB53FD: offset=0x1CC405, size=0xC350, cmdline="--blob"
486178E0-E7F8-11E3-BC5E-0002A5D5C51B: offset=0x1D8755, size=0x74038, cmdline="--blob"
```

3.9.3 FIP creation process

The following figure summarizes the FIP creation process.

Figure 3-7: TC2 FIP creation process



`TF-A fip.bin`

- Created by TF-A. No controls exposed to TC build scripts. Use TOOL_ADD_PAYLOAD to add files into `fip.bin`, for example `trusted-firmware-a/plat/arm/board/tc/platform.mk`:

```
$(eval $(call TOOL_ADD_PAYLOAD, ${BUILD_PLAT}/tb_fw.crt, --tb-fw-cert))
```

Combined FIP

1. Adds RSS support for the GPT partition. RSS keeps its files in `fip_combined.bin`.
   - TC `build-flash-image.sh` uses `assemble_fip()` to add RSS related binaries into `fip_combined.bin`. If additional binaries are needed, they can be added in `fip_combined.bin`. The `fip_combined.bin` creates a symbolic link as `fip-tc.bin`.
   - Note: Be aware that the maximum FIP size (12MB) is hardcoded in the script.
2. The GPT image is created by the `build-flash-image.sh` script.
   - TC `build-flash-image.sh` uses `do_generate_gpt ()` to create the GPT image from `fip_combined.bin`.
3. The resulting GPI image file is `fip_gpt-tc.bin`, used by the FVP as a flash0.

The combined FIP is stored in both partitions FIP_A and FIP_B.

3.10 Working with TF-A mainline

3.10.1 TF-A version

Generally, the TF-A mainline project releases a new version once every 6 months. The TC2-2023.10.04 release tag of the TC2 software stack included v2.9 of TF-A, specifically commit e87102f32bbdf0fa4b2892394cb4f2766321b9d4. You can check the TF-A version in the Makefile as follows:

```
# Trusted Firmware Version
VERSION_MAJOR                   := 2
VERSION_MINOR                   := 9
VERSION                         := ${VERSION_MAJOR}.${VERSION_MINOR}
```

The TF-A component of the TC2-2023.10.04 release also included some offline patches and the sp config file `sp_laylout.json` which are not present in the mainline of TF-A These are located in `$tc2_workspace/build-scripts/files/tfa/tc2`:

```
0001-Makefile-add-trusty_sp_fw_config-build-option.patch
0002-fix-spmd-Enable-NS_TIMER_SWITCH.patch
0003-fix-plat-arm-increase-sp-max-image-size.patch
0004-feat-plat-tc-add-spmc-manifest-with-trusty-sp.patch
0005-feat-plat-tc-update-dts-with-trusty-compatible-strin.patch
0006-feat-arm-tc-Update-trusty-load-address-in-dts-files.patchcd
0007-feat-plat-tc-add-firmware-update-secure-partition.patch
0008-feat-plat-tc-reserve-4-MB-for-stmm-communication-use.patch
0009-refactor-tc-remap-console-logs.patch
0010-Add-Total-compute-FPGA-support.patch
0011-fix-tc-increase-the-firmware-update-partition-size.patch
0012-tc2-fpga-Add-superset-CPU-support.patch
0013-feat-plat-tc-enable-mpmm-and-amu-counters-for-the-FP.patch
0014-tc2-Do-not-enable-MPMM-and-Aux-AMU-counters-always.patch
0015-update-fix-SPMC-manifest.patch
0016-tc2-Enable-gpu-dpu-scmi-power-domain-and-also-gpu-pe.patch
0017-Interrrupt-numbers-for-smmu_700.patch
0018-fix-tc-Specify-non-secure-memory-regions-when-using-.patch
0019-tc-enable-FVP-.dts-support-for-1920x1080-60fps-resol.patch
sp_layout.json
```

If you use a different version to the TF-A mainline, for example the v2.10 release, you must apply these offline patches.

3.10.2 Port to mainline

Using v2.10.0 as example, the v2.10 LTS release branch is located here:
v2.10 LTS[12]
The commit of LTS v2.10.0 is as follows:

```
commit b6c0948400594e3cc4dbb5a4ef04b815d2675808 (tag: v2.10.0, tag: v2.10)
Merge: ccd8c0230 987358099
Author: Manish Pandey <manish.pandey2@arm.com>
Date:   Wed Nov 22 17:31:48 2023 +0100

    Merge "docs(changelog): changelog for v2.10 release" into integration
```

Do the following:

1. Back up the original TF-A code base, then clone the code repository from upstream mainline into the `$tc2_workspace/src/` directory:

```
cd $tc2_workspace/src/
```

```
mv trusted-firmware-a trusted-firmware-a-TC2-back
git clone https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git
```

2. Reset to the v2.10 commit `b6c0948400594e3cc4dbb5a4ef04b815d2675808`:

```
git reset --hard b6c0948400594e3cc4dbb5a4ef04b815d2675808
```

3. Apply the offline patches into the code base.

Use the build script `$tc2_workspace/build-scripts/build-tfa.sh patch` to apply the offline patches. For the v2.10 version, there are 5 patches that conflict with the code base. Apply the 5 patches manually as follows:

```
0001-Makefile-add-trusty_sp_fw_config-build-option.patch
0010-Add-Total-compute-FPGA-support.patch
0012-tc2-fpga-Add-superset-CPU-support.patch
0013-feat-plat-tc-enable-mpmm-and-amu-counters-for-the-FP.patch
0016-tc2-Enable-gpu-dpu-scmi-power-domain-and-also-gpu-pe.patch
```

Note: Because the target platform is the TC2 FVP, patches `0012` and `0013` which are for the TC2_fpga cannot be used. Patch `0016` has already been merged into mainline which is included in the v2.10 code base, so it is also not used.

4. Revert the impact patch with the commit `20629b3153bccdda32116ed5c4861e61fa1fba95`.

Failing to revert this patch results in a build failure.

```
git revert 20629b3153bccdda32116ed5c4861e61fa1fba95
```

5. Build the new TF-A code into the software stack:

```
export PLATFORM=tc2
export TC_TARGET_FLAVOR=fvp
export FILESYSTEM=buildroot
build-scripts/build-tfa.sh clean
build-scripts/build-tfa.sh build
build-scripts/build-tfa.sh deploy
#signed BL1 and SCP_BL1
build-scripts/build-rss.sh deploy
#update the fip image
build-scripts/build-flash-image.sh deploy
```

3.10.3 Running the image on the TC2 FVP

After porting and building, you can run the new firmware image. Use the appropriate command from the user guide[13].

You can see in the boot log that the TF-A version is 2.10 instead of the version of v2.9 of the TC2-2023.10.04 TC2 software stack release:

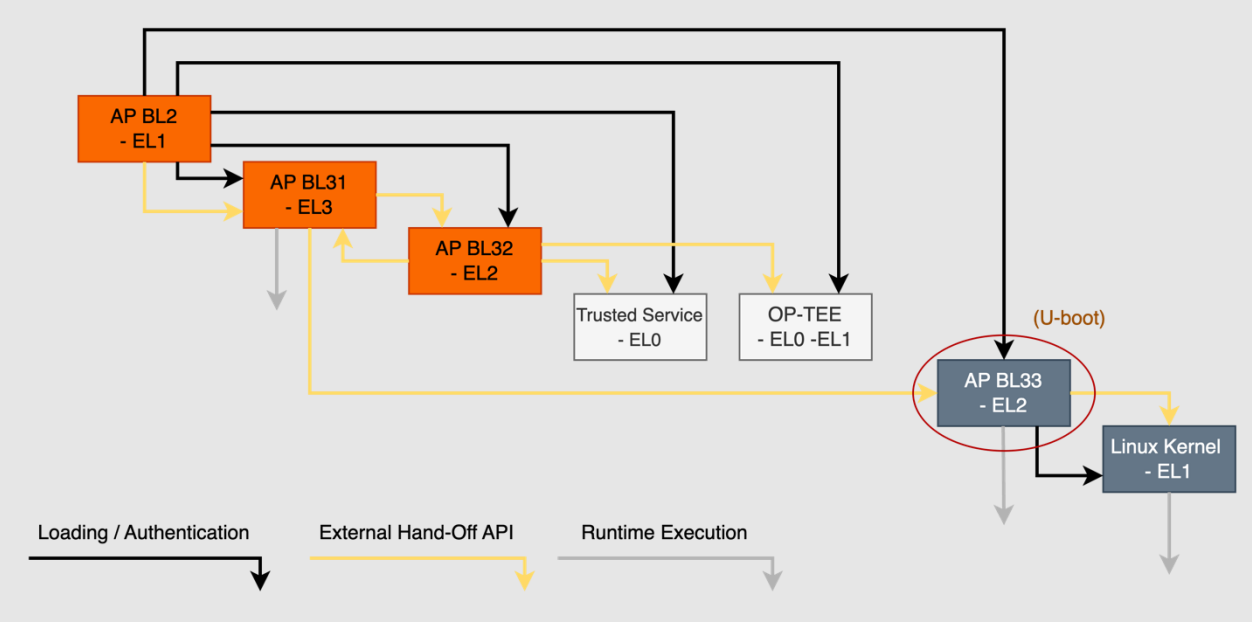
```
NOTICE:  Booting Trusted Firmware
NOTICE:  BL1: v2.10.0   (debug):v2.10.0-17-gf0088d79b
NOTICE:  BL1: Built : 07:43:08, Jan 24 2024
NOTICE:  BL1: Booting BL2
NOTICE:  BL2: v2.10.0   (debug):v2.10.0-17-gf0088d79b
NOTICE:  BL2: Built : 07:43:16, Jan 24 2024
NOTICE:  BL1: Booting BL31
NOTICE:  BL31: v2.10.0  (debug):v2.10.0-17-gf0088d79b
NOTICE:  BL31: Built : 07:43:27, Jan 24 2024
```

U-Boot porting

U-Boot for the TC system

The TC solution uses a boot flow sequence based on RSS serving as the Root of Trust. The following diagram shows the U-Boot (Universal Boot Loader) reference part ：

Figure 0-1 U-Boot flow



The U-Boot image loads as non-trusted firmware at the BL2 stage of the AP, during the Trusted Firmware-A's BL2 stage, after it configures the TrustZone controller and assigns memory regions in DRAM for secure and non-secure use. The U-Boot image is loaded as non-Trusted firmware. Execution control is passed by Trust Firmware-A after it completes booting the runtime software, which is the BL31 stage in the overall TC boot flow. Finally, the U-Boot handles the BL33 stage.

The main tasks of U-Boot are completing hardware initialization, memory mapping, and loading the subsequent kernel for execution. To accomplish these tasks and make the TC software stack work properly on the target board, check the following modules: low level driver, board config, device tree, kconfig, and header files.

The following sections discuss these modules in detail.

3.11 Fetch code and build

Total Compute has already provided the whole software stack including the `u-boot` example. The code is in the directory `$tc2_workspace/src/u-boot`. We will start by fetching the u-boot mainline code.

3.11.1 Fetch mainline u-boot code

To fetch all the `u-boot` source code, enter the following command:

```
git clone https://source.denx.de/u-boot/u-boot
```

Arm recommends using the latest release branch, which is compatible with most features and more stable than the development branch. Use a tag later than v2020.10.rc3, at which point the Total Compute Solution code package was integrated. Here is an example of a suitable branch:

```
remotes/origin/u-boot-2023.07.y
```

Build mainline

After successfully fetching the code, perform the following steps to generate a workable code base.

```
export CROSS_COMPILE=<path_of_toolchain>/aarch64-none-linux-gnu-
make distclean
make total_compute_defconfig
make -j8 V=1 #use V=1 to check the detail log info during make
```

Here `total_compute_defconfig` is the name of the default project name for TC.

3.12 Porting U-Boot on the target device

3.12.1 Code hierarchy for U-Boot

The following table shows the file hierarchy for the U-boot code, especially those files and directories that you might want to modify during porting.

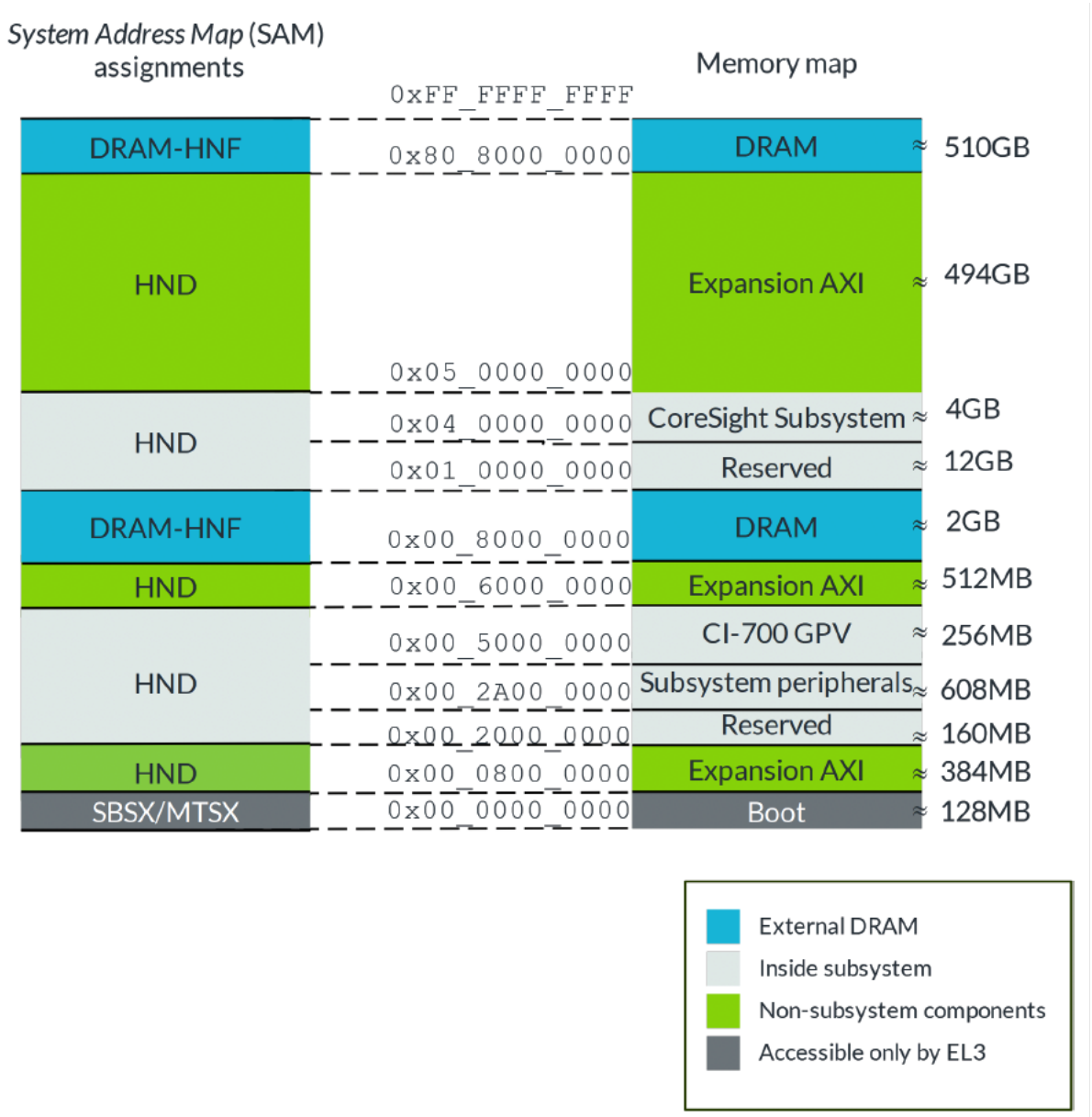
Table 0-1 U-boot code hierarchy

Directory	Files	Usage	Porting Related Features
arch	arch/arm/Kconfig arch/arm/dts/Makefile arch/arm/dts/total_compute.dts arch/arm/cpu/armv8/smccc-call.S arch/arm/lib/asm-offsets.c arch/arm/include/asm/gpio.h	The contents in the Arch folder mainly pertain to features related to processor architecture. Also, it is important to note that Device Tree Source (DTS) files related to the board are also in this directory.	• ARM architecture-related feature configurations in U-Boot. • Project's Device Tree Source (DTS) configurations.
Board	board/armltd/total_compute/Kconfig board/armltd/total_compute/Makefile board/armltd/total_compute/total_compute.c	In the Board folder, you will find content related to the development board, including the definition and configuration implementation of the board-level project. The directory structure is as following: board/<vendor>/<board_name>	Configuration related to the Total_Compute project, including Kconfig and Makefile.
Common	common/board_r.c	Processor architecture-independent generic modules.	
Configs	configs/total_compute_defconfig	Configuration files for board-level devices, where major U-Boot macro features are defined here.	Most of the Total Compute Macro feature configuration.
Drivers	drivers/Kconfig drivers/Makefile drivers/arm-ffa/Kconfig drivers/arm-ffa/Makefile drivers/arm-ffa/arm-ffa-uclass.c drivers/arm-ffa/arm_ffa_prv.h drivers/arm-ffa/core.c	All driver-related functionality is stored here, including network cards, USB, serial ports, and LCD.	Most features are already present on the mainline, but you can add additional FFA functionality here.
Include	include/arm_ffa.h include/arm_ffa_helper.h include/dm/uclass-id.h include/efi_firmware_arm_psa.h include/efi_loader.h include/linux/arm-smccc.h include/configs/total_compute.h	The folder of header files, where definitions related to various features are located.	For porting-related features of Total Compute, modifications to the header files include FFA feature, EFI capsule update, and definitions specific to Total Compute itself.
Lib	lib/Kconfig lib/Makefile lib/arm-ffa/* lib/efi_loader/*	The specific function libraries are placed here.	Add the lib of arm-ffa and efi-loader.

3.12.2 Memory map

The following figure shows the System Address Map (SAM) of TC22 FVP.

Figure 0-2 SAM for U-boot



U-Boot only configures and works with the AP non-secure memory map. The following table shows the memory reference configurations that might need modification during the `u-boot` porting process. Depending on the specific configurations, Total Compute FVP settings, and board hardware definitions, you might need to confirm the configuration with your hardware colleagues.

Table 0-2 memory regions

Start address	End address	Region	Additional Information
0x000 0C00 0000	0x000 0FFF FFFF	CS4 – flash	Non-secure Storage
0x000 2A40 0000	0x000 2A40 FFFF	Non-secure UART	
0x000 8000 0000	0x000 FFFF FFFF	DRAM1	
0x080 8000 0000	0x0FF FFFF FFFF	DRAM2	

In the DRAM region, configure the memory buffer which is needed for u-boot:

The configuration in `configs/total_compute_defconfig`

```
CONFIG_TEXT_BASE=0xe0000000                 //config the default text code base address
CONFIG_SYS_MALLOC_LEN=0x3200000             //config the mem size for malloc function
CONFIG_CUSTOM_SYS_INIT_SP_ADDR=0x8007fff0   //config the stack address, mandatory
CONFIG_ENV_SIZE=0x2a00000                   //config the environment size, mandatory
CONFIG_SYS_LOAD_ADDR=0x90000000             //config the image load address, mandatory

CONFIG_SYS_MEMTEST_START=0x80000000         //config the memory test arrange, optional
CONFIG_SYS_MEMTEST_END=0xff000000

CONFIG_AVB_BUF_ADDR=0x90000000              //config the android verified boot buffer
                                            //address, mandatory for AVB feature
CONFIG_AVB_BUF_SIZE=0x10000000
```

The configuration in `include/configs/total_compute_fvp.h`

```
#define UART0_BASE          0x2A400000      //uart control register base
#define PHYS_SDRAM_1        0x80000000      //RAM1 address
#define DRAM_SEC_SIZE       0x07000000      //reserved for secure world use
#define PHYS_SDRAM_1_SIZE   0x80000000 - DRAM_SEC_SIZE  //RAM1 size
#define PHYS_SDRAM_2        0x8080000000    //RAM2 address
#define PHYS_SDRAM_2_SIZE   0x180000000     //RAM2 size

/*************************************************************
*CFG_EXTRA_ENV_SETTINGS is the extra environments configuration
*Here mainly about the boot address configuration of OS image
*bootm_size     - kernel boot memory available range
*load_addr      - FIT image load address by bootm
*kernel_addr_r  - kernel image load address by booti
*initrd_add_r   - the address of RAMdisk
*fdt_addr_r     - device tree blob file load address
*************************************************************/
#define CFG_EXTRA_ENV_SETTINGS    \
"bootm_size=0x20000000\0"  \
"load_addr=0xa0000000\0"   \
"kernel_addr_r=0x80080000\0"      \
"initrd_addr_r=0x88000000\0"      \
"fdt_addr_r=0x83000000\0"

#define CFG_SYS_FLASH_BASE 0x0C000000

//Shared buffer used for communication between u-boot and the FWU SP
#define FFA_SHARED_MM_BUFFER_SIZE    4 * 1024 * 1024 /* 4 MB */
#define FFA_SHARED_MM_BUFFER_ADDR    (0xFCA00000)
```

For details, reference the above content in the configuration and header files, especially the comments in the code. Typically, you need to do the following:

1. Define the default text code start address with `CONFIG_TEXT_BASE`
2. Define `CONFIG_SYS_MALLOC_LEN` as the size of memory region for `malloc()` in `u-boot`.

3. Define the available RAM range using `PHYS_SDRAM1`, `DRAM_SEC_SIZE`, `PHYS_SDRAM_1_SIZE`, `CFG_SYS_SDRAM_BASE`, and so on. The `u-boot` software calculates the detailed memory distribution values such as the relocate address and offset, based on these RAM range variables.
4. Save the values to the `u-boot` global data `gd` and use them when booting. For other attributes which need to be customized based on the specific hardware, refer to the sample code and the U-boot Documentation[14]. To use extra environment settings, you can customize the following:
    - Available boot memory range with `bootm_size`
    - The FIT image load address with `load_addr`
    - The kernel image load address with `kernel_addr_r`
    - The device tree blob file address with the `fdr_addr_r`, and so on.

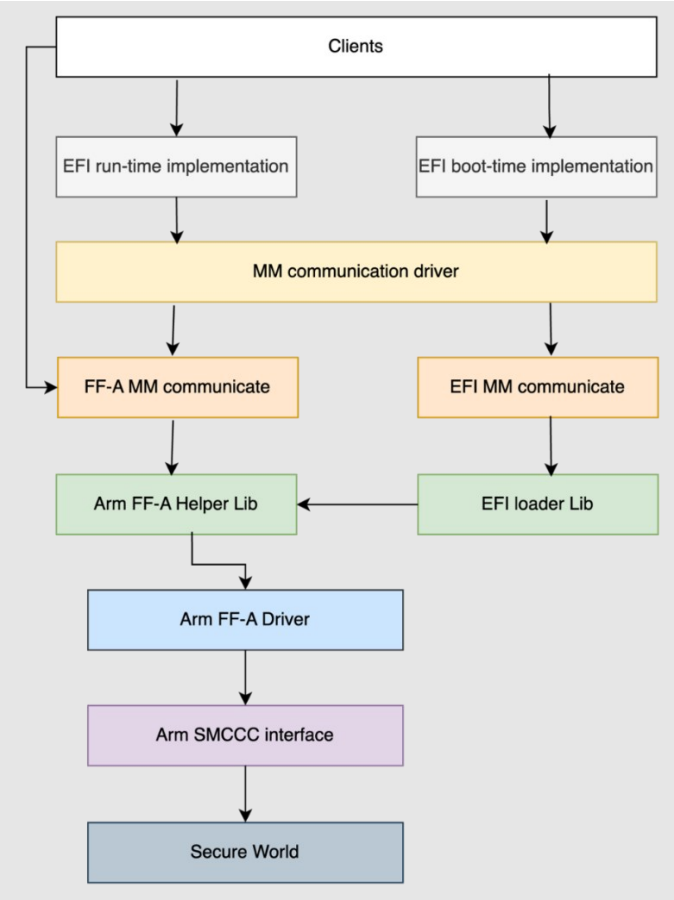
3.12.3 FF-A support

FF-A (Firmware Framework for Arm A-profile) is a software architecture which provides an isolation mechanism. This mechanism enables the isolation of mutually mistrusted software components in the Secure state. FF-A implements the principle of least privilege, which means a software component must be able to access only regions in the physical address space and system resources for its correct operation.

To achieve this, Arm has developed the Arm FFA low-level software module, which implements the Arm Firmware Framework for Armv8-A on U-Boot. This module provides helper FF-A interfaces for user layers. These helper functions allow clients to pass data and select the FF-A function to use for communication with secure world. This module is based on the FF-A specification v1.0 and uses the SMC32 calling convention.

The following figure shows the implementation hierarchy for Arm FF-A support in U-Boot:

Figure 0-3 U-Boot FF-A implementation



To integrate the Arm FFA software module to the mainline code, do the following:

1. Add the SMC (Secure Monitor Call) call logic for FF-A transport in the Arm SMCCC (SMC Calling Convention) implementation.
2. Implement the Arm FF-A driver, located at `<u-boot_src>/driver/`.
3. Implement the FF-A help library, located at `<u-boot_src>/lib`.
4. Register the FF-A driver and library to the `u-boot` module.

Arm provides several patches for the FF-A reference support, in the following path:

```
$tc2_workspace/build-scripts/files/u-boot/tc2/
0002-arm_ffa-introducing-Arm-FF-A-low-level-driver.patch
0003-arm_ffa-rxtx_map-should-use-64-bit-calls.patch
0006-arm_ffa-unmap-rxtx-buffer-before-exiting-u-boot.patch
```

You can update the `u-boot` code by taking the FF-A patches as reference, or by directly applying them. The patches update the following files:

```
MAINTAINERS
arch/arm/cpu/armv8/smccc-call.S
arch/arm/lib/asm-offsets.c
common/board_r.c
drivers/Kconfig
drivers/Makefile
drivers/arm-ffa/Kconfig
drivers/arm-ffa/Makefile
drivers/arm-ffa/arm-ffa-uclass.c
drivers/arm-ffa/arm_ffa_prv.h
drivers/arm-ffa/core.c
include/arm_ffa.h
include/arm_ffa_helper.h
include/dm/uclass-id.h
include/linux/arm-smccc.h
lib/Kconfig
lib/Makefile
lib/arm-ffa/Kconfig
lib/arm-ffa/Makefile
lib/arm-ffa/arm_ffa_helper.c
lib/efi_loader/efi_boottime.c
```

For the current version, TC2-2023.10.04, Arm supports the following runtime FF-A invoke interfaces:
`FFA_PARTITION_INFO_GET`
`FFA_RXTX_UNMAP`
`FFA_MSG_SEND_DIRECT_REQ`

Accessing other interfaces displays the following error message:
`"Undefined FF-A interface <your_interface_id>"`

Firmware update

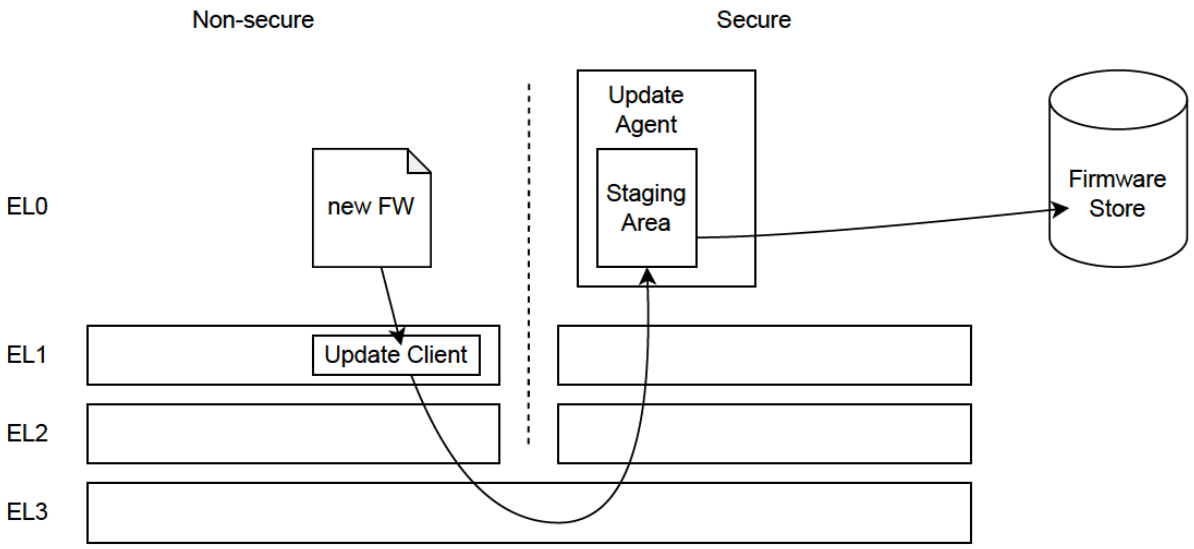
Arm has released an architecture specification for updating A-profile firmware, https://developer.arm.com/documentation/den0118/b/?lang=en. The firmware update flow performs two activities:

Update firmware images in the Firmware Store.

Activate the newly updated firmware images.

The document defines the concept of an Update Agent that controls the Firmware Store. The Update Agent can be implemented within a Secure Partition, in the Secure World. The Firmware Store update ABI defined in this document allows the firmware blobs to be communicated to the Update Agent by a Client in the Host OS. Refer to the following diagrams:

Figure 0-4 U-boot FWU PSA



To support the feature, Arm has developed an FMP (Firmware Management Protocol) driver supporting the Arm FWU PSA specification. This driver communicates to the firmware update secure partition executing in the secure world which is an implementation of the arm PSA specification for the firmware update. The communication to the firmware updating secure partition is based on the Arm FF-A framework. With this, you can update and activate the firmware package in U-Boot.

To integrate the Arm FMP driver to the mainline code, do the following:

1. Add the FMP driver logic to the `efi_loader` library at `$tc2_workspace/src/u-boot/lib/efi_loader`.
2. Register the FMP driver interface and kconfig setting to the existing `efi_loader` library code.
3. Reserve a shared buffer of at least 4 MB to use for communication between `u-boot` and the FWU SP (secure partition).
4. Enable following configuration in the board `defconfig` file: `$tc2_workspace/src/u-boot/configs/total_compute_defconfig`.

```
CONFIG_ARM_FFA_TRANSPORT=y
CONFIG_CMD_EFIDEBUG=y
CONFIG_EFI_CAPSULE_FIRMWARE_ARM_PSA=y
```

```
CONFIG_EFI_CAPSULE_ON_DISK=y
CONFIG_EFI_RUNTIME_UPDATE_CAPSULE=y
```

5. Enable the FF-A transport driver by adding to it to device tree file `total_compute.dts`.

Arm provides two patches for the FFA reference, in the following location:

```
$tc2_workspace/build-scripts/files/u-boot/tc2/
0004-efi_firmware-add-new-fmp-driver-that-supports-arm-fw.patch
0005-arm-total_compute-enable-capsule-update.patch
```

You can update the U-Boot code by taking the FMP patches as reference, or by directly applying them. The patches update following files:

```
include/efi_firmware_arm_psa.h
include/efi_loader.h
lib/efi_loader/Kconfig
lib/efi_loader/Makefile
lib/efi_loader/efi_capsule.c
lib/efi_loader/efi_firmware.c
lib/efi_loader/efi_firmware_arm_psa.c

include/configs/total_compute.h
arch/arm/dts/total_compute.dts
configs/total_compute_defconfig
```

After porting is complete, you can test the capsule firmware update feature. Refer to the TC User Guide section on firmware-update[15].

3.12.4 Configurations

Besides the memory map and driver configuration, you must also configure other board options for successful porting:

`defconfig` file

The content in `$tc2_workspace/src/u-boot/configs/total_compute_defconfig` is a configuration example of the options for a board based on TC. The following table lists the relevant options:

Table 0-3 U-boot Config Parameters

Configuration	Description
CONFIG_USE_BOOTARGS	If configured, uses the boot arguments in U-Boot.
CONFIG_BOOTARGS	The command line arguments passed for OS boot
CONFIG_BOOTDELAY	The boot delay time to auto load the Linux kernel.
CONFIG_BOOTCOMMAND	The command or script which will be executed during the boot delay when the user does not enter the shell.

For `bootargs`, the default setting of `total_compute` is as follows:

```
CONFIG_BOOTARGS="console=ttyAMA0 debug user_debug=31 earlycon=pl011,0x2A400000
loglevel=9 androidboot.hardware=total_compute androidboot.boot_devices=1c050000.mmci
ip=dhcp androidboot.selinux=permissive allow_mismatched_32bit_el0
systemd.log_level=info"
```

You can modify the optional kernel parameters as needed. For example, the early console device option `earlycon`, the user debug event mask option `user_debug`, and so on. For more details, refer to the kernel documentation.

For `bootcmd`, the default setting of `total_compute` is as follows:

```
CONFIG_BOOTARGS=
"virtio scan;
 if part number virtio 0 vbmeta is_avb; then
    echo virtio with vbmeta partition detected.;
    echo starting Android Verified boot.;
    avb init virtio 0;
    if avb verify; then
         set bootargs $bootargs $avb_bootargs;
         part start virtio 0 boot boot_start;
         part size virtio 0 boot boot_size;
         virtio read ${load_addr} ${boot_start} ${boot_size};
         bootm ${load_addr} ${load_addr} ${fdt_addr_r};
    else;
        echo AVB verification failed.;
        exit;
    fi;
elif part number virtio 0 system is_non_avb_android; then
    echo booting non-avb android;
    booti ${kernel_addr_r} ${initrd_addr_r} ${fdt_addr_r};
elif iminfo ${load_addr}; then
    echo Booting Buildroot FIT image;
    bootm ${load_addr} ${load_addr} ${fdt_addr_r};
else;
    set bootargs $bootargs root=/dev/mmcblk0p1 rw ;
    echo Booting Debian;
    booti ${kernel_addr_r} - ${fdt_addr_r};
fi;"
```

The `autoboot` command boots the specific OS image with specific address option. As described in the Chapter Memory Map, the value of these addresses can be customized in `total_compute_fvp.h`.

For other platform and board feature configuration, you can modify the configuration here. For example, for the Total Compute FVP board example, the `virtio` devices are configured as follows:

```
CONFIG_VIRTIO=y
CONFIG_VIRTIO_MMIO=y
CONFIG_VIRTIO_BLK=y
CONFIG_VIRTIO_NET=n
CONFIG_CMD_VIRTIO=y
```

`Kconfig` file

You need to add the following board information to the Arm `Kconfig` file. Remember to change references such as `FVP` to your specific board name:

In `$tc2_workspace/src/u-boot/arch/arm/Kconfig`:

```
config TARGET_TOTAL_COMPUTE_FVP
       bool "Support Total Compute FVP Platform"
       select ARM64
       select PL01X_SERIAL
       select DM
       select DM_SERIAL
       select DM_MMC
       select DM_GPIO
```

Configuration on the board `Kconfig` needs to be handled as follows:

In `$tc2_workspace/src/u-boot/board/armltd/total_compute/Kconfig`

```
if TARGET_TOTAL_COMPUTE_<customize_board>

config SYS_BOARD
       default "total_compute"
```

```
config SYS_VENDOR
       default "<vendor_name>"

config SYS_CONFIG_NAME
       default "total_compute_<customize_board"

endif
```

Device tree file

The content in the TC22 `$tc2_workspace/src/u-boot/arch/arm/dts/total_compute.dts` file already has the information for the node you need to configure for the device tree:

```
sysreg: sysreg@1c010000 {
    compatible = "arm,vexpress-sysreg";
    reg = <0x0 0x001c010000 0x0 0x1000>;
    gpio-controller;
    #gpio-cells = <2>;
};

fixed_3v3: v2m-3v3 {
    compatible = "regulator-fixed";
    regulator-name = "3V3";
    regulator-min-microvolt = <3300000>;
    regulator-max-microvolt = <3300000>;
    regulator-always-on;
};

mmci@1c050000 {
    compatible = "arm,pl180", "arm,primecell";
    reg = <0x0 0x001c050000 0x0 0x1000>;
    cd-gpios = <&sysreg 0 0>;
    arm,primecell-periphid = <0x00880180>;
    wp-gpios = <&sysreg 1 0>;
    bus-width = <8>;
    max-frequency = <12000000>;
    vmmc-supply = <&fixed_3v3>;
    clocks = <&clock24mhz>, <&clock24mhz>;
    clock-names = "mclk", "apb_pclk";
};

clock24mhz: clock24mhz {
    compatible = "fixed-clock";
    #clock-cells = <0>;
    clock-frequency = <24000000>;
    clock-output-names = "bp:clock24mhz";
};

psci {
    compatible = "arm,psci-1.0", "arm,psci-0.2";
    method = "smc";
};
```

Confirm the address information of `sysreg` and `mmci` with the hardware engineer for the system register and Arm Primecell address mapping. Then make the value in the dts file correspond with the hardware.

The Chapter FF-A support showed how to add the FF-A module for U-Boot. You also need to add the FF-A node into the dts file as follows:

```
    arm_ffa {
          compatible = "arm,ffa";
          method = "smc";
          status = "okay";
    };
```

You can add other device nodes as needed. For example, the TC software stack shows how to add a `virtio` block device. Use this as an example to mount your own disk image as a `virtio` block device.

```
    virtio_block@1c130000 {
          compatible = "virtio,mmio";
          reg = <0x0 0x1c130000 0x0 0x200>;
          interrupts = <0 170 4>;
    };
```

Finally, add these to the make file as a dts resource, in `$tc2_workspace/src/u-boot/arch/arm/dts/Makefile`:

```
dtb-$(CONFIG_TARGET_TOTAL_COMPUTE_FVP) += total_compute.dtb
```

`total_compute_fvp.h`

Besides the memory size customization option, you need to configure the following options in the header file `$tc2_workspace/src/u-boot/include/configs/total_compute_fvp.h`

```
/* PL011 Serial Configuration */
#define CFG_PL011_CLOCK    7372800
```

3.13 Summary

3.13.1 Arm release patches

As mentioned in previous chapters, Arm provides the following patches for you to create a base version easily and quickly.

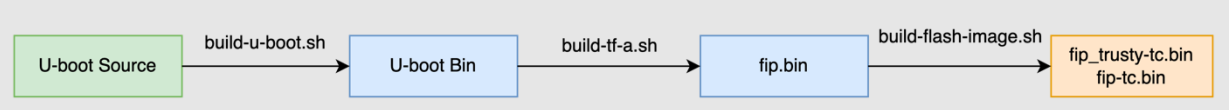
In `$tc2_workspace/build_scripts/files/u-boot/tc2`:

```
0001-arm-total_compute-update-secure-dram-size.patch
0002-arm_ffa-introducing-Arm-FF-A-low-level-driver.patch
0003-arm_ffa-rxtx_map-should-use-64-bit-calls.patch
0004-efi_firmware-add-new-fmp-driver-that-supports-arm-fw.patch
0005-arm-total_compute-enable-capsule-update.patch
0006-arm_ffa-unmap-rxtx-buffer-before-exiting-u-boot.patch
0007-arm-total_compute-create-different-product-type-for-.patch
0008-virtio-add-virtio-block-device.patch
0009-avb-Extend-support-to-non-eMMC-interfaces.patch
```

3.13.2 Build

Use the method in Chapter Build Mainline to build the U-Boot image. Then package it into the FIP (Firmware Image Package), as follows:

Figure 0-5 Package of U-Boot



Refer to the TC user guide, the section on build-variants-configuration[16], and use following commands to build the FIP TC image:

```
#set environment for build
export PLATFORM=TC
export TC_TARGET_FLAVOR=fvp                #specific board name, like "fvp"
export FILESYSTEM=debian                   #specific platform name, like "debian"

#TC intergrated build scripts
./build-scripts/build-u-boot.sh clean              #uboot clean and build
./build-scripts/build-u-boot.sh build
./build-scripts/build-u-boot.sh deploy
./build-scripts/build-tfa.sh build                 #TF-A package
./build-scripts/build-tfa.sh deploy
./build-scripts/build-flash-image.sh deploy        #update the fip image
```

3.13.3 Run

After you port and build, you can enter the U-Boot shell by stopping autoboot at the booting stage, in the `terminal_uart_ap` interface. In the terminal you can use `u-boot` commands to work in the U-Boot environment. For example, you can get the U-Boot version, get environment information, boot, update firmware, and so on.

```
Hit any key to stop autoboot:
TOTAL_COMPUTE#
TOTAL_COMPUTE# version
U-Boot 2023.07.02 (Dec 25 2023 - 16:03:17 +0800)
aarch64-none-linux-gnu-gcc (Arm GNU Toolchain 12.2.Rel1 (Build arm-12.24)) 12.2.1
20221205
GNU ld (Arm GNU Toolchain 12.2.Rel1 (Build arm-12.24)) 2.39.0.20221210
TOTAL_COMPUTE# virtio scan
Disk virtio-blk#0 not ready
TOTAL_COMPUTE# virtio info
Device 0: 0000 VirtIO Block Device
            Type: Hard Disk
            Capacity: not available
TOTAL_COMPUTE#
TOTAL_COMPUTE# printenv
arch=arm
baudrate=115200
board=total_compute
board_name=total_compute
bootargs=console=ttyAMA0 debug user_debug=31 earlycon=pl011,0x2A400000 loglevel=9
androidboot.hardware=total_compute androidboot.boot_devices=1c050000.mmci ip=dhcp
androidboot.selinux=permissive allow_mismatched_32bit_el0 systemd.log_level=info
bootcmd=virtio scan;if iminfo ${load_addr}; then echo Booting Buildroot FIT image;
bootm ${load_addr} ${load_addr} ${fdt_addr_r}; else;  set bootargs $bootargs
root=/dev/mmcblk0p1 rw ; echo Booting Debian;booti ${kernel_addr_r} - ${fdt_addr_r};
fi;
bootdelay=1
bootm_size=0x20000000
cpu=armv8
fdt_addr_r=0x83000000
fdtcontroladdr=f2b357d0
initrd_addr_r=0x88000000
kernel_addr_r=0x80080000
load_addr=0xa0000000
loadaddr=0x90000000
stderr=serial_pl01x
stdin=serial_pl01x
stdout=serial_pl01x
vendor=armltd
Environment size: 1425/44040188 bytes
TOTAL_COMPUTE#
TOTAL_COMPUTE#boot
```

4 TC Linux Kernel overview

The Linux Kernel in the Total Compute 2022 OSS software stack release originates from the Android common kernel and contains the subsystem-specific features that demonstrate the capabilities of Total Compute. Apart from the default configuration, it enables the following:

Arm MHUv2 controller driver

Arm FF-A driver

Arm FF-A user space interface driver

OP-TEE driver with FF-A Transport Support

Trusty driver with FF-A Transport Support

Virtualization using pKVM

The TC Kernel is adapted for TC hardware. Linux kernel porting involves the following:

Hardware descriptions and adaption (peripherals, processor cores).
: Device Tree

Different kernel release, version, and config (including device drivers, like the Mali DDK)
: 4K, 16K, or 64K page size The Android kernel is expected to adopt 16K.

Different File Systems
: Buildroot(based on Busybox), Android, Debian

This guide focuses on porting one standard kernel with the Debian Filesystem onto the TC platform.

4.1 Fetch the TC kernel source code

You can fetch the entire TC software stack, together with TC Linux kernel. Alternatively, you can sync the Linux kernel alone:

```
git clone https://git.gitlab.arm.com/arm-reference-solutions/linux.git -b TC2-
2023.10.04
```

Note that this tag is updated when there are new releases.

4.2 TC kernel configuration

The TC kernel's kconfig is combined with `arch/arm64/configs/gki_defconfig` and several other configuration files:

`build-scripts/config/tc2.config`

```
# Kernel
declare -A LINUX_defconfig
LINUX_defconfig[path]="linux"
LINUX_defconfig[config]="base.cfg dhcp.cfg devtmpfs.cfg gralloc.cfg mali.cfg rtc.cfg
ffa.cfg optee.cfg virtio.cfg autofdo.cfg ci700.cfg trusty.cfg disable_mpam.cfg pkvm.cfg
mpam.cfg input.cfg scmi.cfg"
source $SCRIPT_DIR/config/debian_ddk.config
```

Individual configurations under `build-scripts/files/kernel/tc2/`

```
autofdo.cfg  ci700.cfg  dhcp.cfg  ffa.cfg   gralloc.cfg  mali.cfg  optee.cfg  rtc.cfg
trusty.cfg base.cfg devtmpfs.cfg  disable_mpam.cfg  fpga.cfg  input.cfg    mpam.cfg
pkvm.cfg scmi.cfg virtio.cfg
```

Merge all the configuration files into one `.config` file using `build-scripts/build-linux.sh`

```
local lconfig=LINUX_defconfig[config]

    CFG_DIR=$FILES_DIR/kernel/$PLATFORM
    mkdir -p $LINUX_OUTDIR
    info_echo "Building using config fragments"
    CONFIG=""
    for config in ${!lconfig}; do
        CONFIG=$CONFIG"$CFG_DIR/$config "
    done
    if [ "$TC_TARGET_FLAVOR" == "fpga" ]; then
        CONFIG=$CONFIG"$CFG_DIR/fpga.cfg "
    fi
    CONFIG="$LINUX_SRC/arch/arm64/configs/gki_defconfig "$CONFIG
    pushd $LINUX_SRC
    scripts/kconfig/merge_config.sh -O $LINUX_OUTDIR -m $CONFIG
```

Architecture and platform configurations:

```
CONFIG_ARM64=y
# CONFIG_TC_FPGA is not set
CONFIG_64BIT=y
CONFIG_MMU=y
CONFIG_ARM64_PAGE_SHIFT=12
CONFIG_ARM64_CONT_PTE_SHIFT=4
CONFIG_ARM64_CONT_PMD_SHIFT=4
CONFIG_SMP=y
CONFIG_KERNEL_MODE_NEON=y
```

```
#
# Platform selection
CONFIG_ARCH_SUNXI=y
CONFIG_ARCH_HISI=y
CONFIG_ARCH_QCOM=y
CONFIG_ARCH_VEXPRESS=y

CONFIG_ARM64_4K_PAGES=y
# CONFIG_ARM64_16K_PAGES is not set
# CONFIG_ARM64_64K_PAGES is not set
# CONFIG_ARM64_VA_BITS_39 is not set
CONFIG_ARM64_VA_BITS_48=y
CONFIG_ARM64_VA_BITS=48
CONFIG_ARM64_PA_BITS_48=y
CONFIG_ARM64_PA_BITS=48
# CONFIG_CPU_BIG_ENDIAN is not set
CONFIG_CPU_LITTLE_ENDIAN=y
CONFIG_SCHED_MC=y
# CONFIG_SCHED_SMT is not set
CONFIG_NR_CPUS=32
CONFIG_HOTPLUG_CPU=y
# CONFIG_NUMA is not set
# CONFIG_HZ_100 is not set
CONFIG_HZ_250=y
# CONFIG_HZ_300 is not set
# CONFIG_HZ_1000 is not set
CONFIG_HZ=250
CONFIG_UNMAP_KERNEL_AT_EL0=y
CONFIG_MITIGATE_SPECTRE_BRANCH_HISTORY=y
CONFIG_RODATA_FULL_DEFAULT_ENABLED=y
CONFIG_ARM64_SW_TTBR0_PAN=y
CONFIG_ARM64_TAGGED_ADDR_ABI=y
CONFIG_COMPAT=y
CONFIG_KUSER_HELPERS=y
CONFIG_ARMV8_DEPRECATED=y
CONFIG_SWP_EMULATION=y
CONFIG_CP15_BARRIER_EMULATION=y
CONFIG_SETEND_EMULATION=y
#
# ARMv8.1 architectural features
#
CONFIG_ARM64_HW_AFDBM=y
CONFIG_ARM64_PAN=y
CONFIG_AS_HAS_LDAPR=y
CONFIG_AS_HAS_LSE_ATOMICS=y
CONFIG_ARM64_LSE_ATOMICS=y
CONFIG_ARM64_USE_LSE_ATOMICS=y

# ARMv8.2 architectural features
#
# CONFIG_ARM64_PMEM is not set
CONFIG_ARM64_RAS_EXTN=y
CONFIG_ARM64_CNP=y

# ARMv8.3 architectural features
#
CONFIG_ARM64_PTR_AUTH=y
CONFIG_ARM64_PTR_AUTH_KERNEL=y
CONFIG_CC_HAS_BRANCH_PROT_PAC_RET=y
CONFIG_CC_HAS_SIGN_RETURN_ADDRESS=y
CONFIG_AS_HAS_PAC=y
CONFIG_AS_HAS_CFI_NEGATE_RA_STATE=y

# ARMv8.4 architectural features
#
CONFIG_ARM64_AMU_EXTN=y
```

```
CONFIG_AS_HAS_ARMV8_4=y
CONFIG_ARM64_TLB_RANGE=y

# ARMv8.5 architectural features
#
CONFIG_AS_HAS_ARMV8_5=y
CONFIG_ARM64_BTI=y
CONFIG_ARM64_BTI_KERNEL=y
CONFIG_CC_HAS_BRANCH_PROT_PAC_RET_BTI=y
CONFIG_ARM64_E0PD=y
CONFIG_ARCH_RANDOM=y
CONFIG_ARM64_AS_HAS_MTE=y
CONFIG_ARM64_MTE=y

# ARMv8.7 architectural features
#
CONFIG_ARM64_EPAN=y
# end of ARMv8.7 architectural features

CONFIG_ARM64_SVE=y
CONFIG_ARM64_MODULE_PLTS=y
# CONFIG_ARM64_PSEUDO_NMI is not set
CONFIG_RELOCATABLE=y
CONFIG_RANDOMIZE_BASE=y
# CONFIG_RANDOMIZE_MODULE_REGION_FULL is not set
CONFIG_CC_HAVE_STACKPROTECTOR_SYSREG=y
CONFIG_STACKPROTECTOR_PER_TASK=y
```

The RD-TC22 FVP hardware includes Arm TC22 processors such as Cortex-X4 (Hunter-ELP), Cortex-A720 (Hunter), and Cortex-A520 (Hayes) which are Arm v9.2-A architecture compliant. TC2-related features need to be enabled.

4.3 TC kernel device tree configuration

The device tree describes hardware devices in a tree structure. The devices are described in DTS (Device Tree Source) files. They are compiled by the device tree compiler to a binary DTB (Device Tree Blob) file.

4.3.1 Device tree in TF-A trusted firmware on TC platform

Normally DTS files are in the Linux kernel tree (`arch/arm64/boot/dts/arm/`), however, the TC platform puts them in the TF-A tree at the following location:

```
trusted-firmware-a/fdts/tc.dts
```

In this device tree file, you can see that the TC hardware consists of:

- Eight cores in one cluster
- Mali GPU
- DPU
- SMMU
- GIC interrupt controller
- Timer and clock
- Arm tracer
- UART
- Ethernet
- MHU
- SCMI memory information

Details of these devices are in the following sections.

4.3.2 Device tree for CPU node

Under the node `cpus`, there are `cpu-map`, `idle-states`, `amus`, and `CPU0` to `CPU7`:

```
    cpus {
                #address-cells = <1>;
                #size-cells = <0>;

                cpu-map {
                        cluster0 {
                                core0 {
                                        cpu = <&CPU0>;
                                };
                                core1 {
                                        cpu = <&CPU1>;
                                };
                                core2 {
                                        cpu = <&CPU2>;
                                };
                                core3 {
                                        cpu = <&CPU3>;
                                };
                                core4 {
                                        cpu = <&CPU4>;
                                };
```

```
                        core5 {
                                cpu = <&CPU5>;
                        };
                        core6 {
                                cpu = <&CPU6>;
                        };
                        core7 {
                                cpu = <&CPU7>;
                        };
                };
        };
        /*
         * The timings below are just to demonstrate working cpuidle.
         * These values may be inaccurate.
         */
        idle-states {
                entry-method = "arm,psci";
                CPU_SLEEP_0: cpu-sleep-0 {
                        compatible = "arm,idle-state";
                        arm,psci-suspend-param = <0x0010000>;
                        local-timer-stop;
                        entry-latency-us = <300>;
                        exit-latency-us = <1200>;
                        min-residency-us = <2000>;
                };
                CLUSTER_SLEEP_0: cluster-sleep-0 {
                        compatible = "arm,idle-state";
                        arm,psci-suspend-param = <0x1010000>;
                        local-timer-stop;
                        entry-latency-us = <400>;
                        exit-latency-us = <1200>;
                        min-residency-us = <2500>;
                };
        };
        amus {
                amu: amu-0 {
                        #address-cells = <1>;
                        #size-cells = <0>;

                        mpmm_gear0: counter@0 {
                                reg = <0>;

                                enable-at-el3;
                        };

                        mpmm_gear1: counter@1 {
                                reg = <1>;

                                enable-at-el3;
                        };

                        mpmm_gear2: counter@2 {
                                reg = <2>;

                                enable-at-el3;
                        };
                };
        };
        CPU0:cpu@0 {
                device_type = "cpu";
                compatible = "arm,armv8";
                reg = <0x0>;
                enable-method = "psci";
                clocks = <&scmi_dvfs 0>;
                cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
                capacity-dmips-mhz = <406>;
```

```
                amu = <&amu>;
                supports-mpmm;
        };
        CPU1:cpu@100 {
                device_type = "cpu";
                compatible = "arm,armv8";
                reg = <0x100>;
                enable-method = "psci";
                clocks = <&scmi_dvfs 0>;
                cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
                capacity-dmips-mhz = <406>;
                amu = <&amu>;
                supports-mpmm;
        };
       CPU2:cpu@200 {
                device_type = "cpu";
                compatible = "arm,armv8";
                reg = <0x200>;
                enable-method = "psci";
                clocks = <&scmi_dvfs 0>;
                cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
                capacity-dmips-mhz = <406>;
                amu = <&amu>;
                supports-mpmm;
        };

        CPU3:cpu@300 {
                device_type = "cpu";
                compatible = "arm,armv8";
                reg = <0x300>;
                enable-method = "psci";
                clocks = <&scmi_dvfs 0>;
                cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
                capacity-dmips-mhz = <406>;
                amu = <&amu>;
                supports-mpmm;
        };
        CPU4:cpu@400 {
                device_type = "cpu";
                compatible = "arm,armv8";
                reg = <0x400>;
                enable-method = "psci";
                clocks = <&scmi_dvfs 1>;
                cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
                capacity-dmips-mhz = <912>;
                amu = <&amu>;
                supports-mpmm;
        };
      CPU5:cpu@500 {
                device_type = "cpu";
                compatible = "arm,armv8";
                reg = <0x500>;
                enable-method = "psci";
                clocks = <&scmi_dvfs 1>;
                cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
                capacity-dmips-mhz = <912>;
                amu = <&amu>;
                supports-mpmm;
        };
        CPU6:cpu@600 {
                device_type = "cpu";
                compatible = "arm,armv8";
                reg = <0x600>;
                enable-method = "psci";
                clocks = <&scmi_dvfs 1>;
                cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
```

```
                        capacity-dmips-mhz = <912>;
                        amu = <&amu>;
                        supports-mpmm;
                };
                CPU7:cpu@700 {
                        device_type = "cpu";
                        compatible = "arm,armv8";
                        reg = <0x700>;
                        enable-method = "psci";
                        clocks = <&scmi_dvfs 2>;
                        cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
                        capacity-dmips-mhz = <1024>;
                        amu = <&amu>;
                        supports-mpmm;
                };
        };

    psci {
        compatible = "arm,psci-1.0", "arm,psci-0.2";
        method = "smc";
    };


    ete0 {
        compatible = "arm,embedded-trace-extension";
        cpu = <&CPU0>;
    };

    ete1 {
        compatible = "arm,embedded-trace-extension";
        cpu = <&CPU1>;
    };

    ete2 {
        compatible = "arm,embedded-trace-extension";
        cpu = <&CPU2>;
    };

    ete3 {
        compatible = "arm,embedded-trace-extension";
        cpu = <&CPU3>;
    };

    ete4 {
        compatible = "arm,embedded-trace-extension";
        cpu = <&CPU4>;
    };

    ete5 {
        compatible = "arm,embedded-trace-extension";
        cpu = <&CPU5>;
    };

    ete6 {
        compatible = "arm,embedded-trace-extension";
        cpu = <&CPU6>;
    };

    ete7 {
        compatible = "arm,embedded-trace-extension";
        cpu = <&CPU7>;
    };

    trbe0 {
        compatible = "arm,trace-buffer-extension";
        interrupts = <1 2 4>;
```

```
    };
```

4.3.3 Device tree for Memory node

The memory and reserved memory node are as follow:

```
   memory {
        device_type = "memory";
        reg = <0x00 0x80000000 0x00 0x79000000 0x80 0x80000000 0x01 0x80000000>;
    };
   reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        linux,cma {
            compatible = "shared-dma-pool";
            reusable;
            size = <0x0 0x8000000>;
            linux,cma-default;
        };

        optee@0xf8e00000 {
            compatible = "restricted-dma-pool";
            reg = <0x00000000 0xf8e00000 0 0x00200000>;
        };

        fwu_mm@0xfca00000 {
            reg = <0x00000000 0xfca00000 0 0x00400000>;
            no-map;
        };
    };
```


Note

The `memory` node is generated by u-boot code dynamically. On a running TC22 system, you can dump the DTS by entering the following command from the console:

```
dtc -I fs -O dts -o ./extracted.dts /proc/device-tree
```

4.3.4 Device tree for Arm core peripheral IPs

Arm GIC-700 interrupt controller:

```
    gic: interrupt-controller@2c010000 {
        compatible = "arm,gic-600", "arm,gic-v3";
        #address-cells = <2>;
        #interrupt-cells = <3>;
        #size-cells = <2>;
        ranges;
        interrupt-controller;
        reg = <0x0 0x30000000 0 0x10000>, /* GICD */
              <0x0 0x30080000 0 0x200000>; /* GICR */
```

```
        interrupts = <0x1 0x9 0x4>;
    };
```

Arm generic timer:

```
    timer {
        compatible = "arm,armv8-timer";
        interrupts = <0x1 13 0x8>,
                 <0x1 14 0x8>,
                 <0x1 11 0x8>,
                 <0x1 10 0x8>;
    };
```

MHU shared memory communication:

```
    sram: sram@6000000 {
        compatible = "mmio-sram";
        reg = <0x0 0x06000000 0x0 0x8000>;

        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0 0x0 0x06000000 0x8000>;

        cpu_scp_scmi_mem: scp-shmem@0 {
            compatible = "arm,scmi-shmem";
            reg = <0x0 0x80>;
        };
    };

    mbox_db_rx: mhu@45010000 {
        compatible = "arm,mhuv2-rx","arm,primecell";
        reg = <0x0 0x45010000 0x0 0x1000>;
        clocks = <&soc_refclk100mhz>;
        clock-names = "apb_pclk";
        #mbox-cells = <2>;
        interrupts = <0 317 4>;
        interrupt-names = "mhu_rx";
        mhu-protocol = "doorbell";
        arm,mhuv2-protocols = <0 1>;
    };

    mbox_db_tx: mhu@45000000 {
        compatible = "arm,mhuv2-tx","arm,primecell";
        reg = <0x0 0x45000000 0x0 0x1000>;
        clocks = <&soc_refclk100mhz>;
        clock-names = "apb_pclk";
        #mbox-cells = <2>;
        interrupt-names = "mhu_tx";
        mhu-protocol = "doorbell";
        arm,mhuv2-protocols = <0 1>;
    };

    scmi {
        compatible = "arm,scmi";
        mbox-names = "tx", "rx";
        mboxes = <&mbox_db_tx 0 0 &mbox_db_rx 0 0 >;
        shmem = <&cpu_scp_scmi_mem &cpu_scp_scmi_mem>;
        #address-cells = <1>;
        #size-cells = <0>;

        scmi_devpd: protocol@11 {
            reg = <0x11>;
            #power-domain-cells = <1>;
        };
```

```
        scmi_dvfs: protocol@13 {
            reg = <0x13>;
            #clock-cells = <1>;
        };

        scmi_clk: protocol@14 {
            reg = <0x14>;
            #clock-cells = <1>;
        };
    };
```

SMMU-700:

```
    smmu_700: smmu_700@3f000000 {
        #iommu-cells = <1>;
        compatible = "arm,smmu-v3";
        reg = <0x0 0x3f000000 0x0 0x5000000>;
        interrupts = <GIC_SPI 228 IRQ_TYPE_EDGE_RISING>,
            <GIC_SPI 229 IRQ_TYPE_EDGE_RISING>,
            <GIC_SPI 230 IRQ_TYPE_EDGE_RISING>;
        interrupt-names = "eventq", "cmdq-sync", "gerror";
        dma-coherent;
    };
```

CI-700 PMU:

```
    cmn-pmu {
        compatible = "arm,ci-700";
        reg = <0x0 0x50000000 0x0 0x10000000>;
        interrupts = <0x0 460 0x4>;
    };
```

Arm PL011 UART:

```
    aliases {
        serial0 = &ap_ns_uart;
    };

    chosen {
        stdout-path = "serial0:115200n8";
};

    soc_refclk100mhz: refclk100mhz {
        compatible = "fixed-clock";
        #clock-cells = <0>;
        clock-frequency = <100000000>;
        clock-output-names = "apb_pclk";
    };

    soc_uartclk:  uartclk {
        compatible = "fixed-clock";
        #clock-cells = <0>;
        clock-frequency = <50000000>;
        clock-output-names = "uartclk";
    };

    ap_ns_uart: uart@2A400000 {
        compatible = "arm,pl011", "arm,primecell";
        reg = <0x0 0x2A400000 0x0 0x1000>;
        interrupts = <0x0 63 0x4>;
```

```
        clocks = <&soc_uartclk>, <&soc_refclk100mhz>;
        clock-names = "uartclk", "apb_pclk";
        status = "okay";
    };
```

Arm Mali GPU:

```
    gpu_clk: gpu_clk {
        compatible = "fixed-clock";
        #clock-cells = <0>;
        clock-frequency = <1000000000>;
    };

    gpu_core_clk: gpu_core_clk {
        compatible = "fixed-clock";
        #clock-cells = <0>;
        clock-frequency = <1000000000>;
    };

    gpu: gpu@2d000000 {
        compatible = "arm,mali-midgard";
        reg = <0x0 0x2d000000 0x0 0x200000>;
        interrupts = <0 66 4>, <0 67 4>, <0 65 4>;
        interrupt-names = "JOB", "MMU", "GPU";
        clocks = <&gpu_core_clk>;
        clock-names = "shadercores";
        power-domains = <&scmi_devpd 9>;
        scmi-perf-domain = <3>;
        iommus = <&smmu_700 0x200>;
    };

    power_model@simple {
        /*
         * Numbers used are irrelevant to Titan,
         * it helps suppressing the kernel warnings.
         */
        compatible = "arm,mali-simple-power-model";
        static-coefficient = <2427750>;
        dynamic-coefficient = <4687>;
        ts = <20000 2000 (-20) 2>;
        thermal-zone = "";
    };
```

Arm Mali D71:

```
    dp0: display@2cc00000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "arm,mali-d71";
        reg = <0 0x2cc00000 0 0x20000>;
        interrupts = <0 69 4>;
        interrupt-names = "DPU";
        clocks = <&scmi_clk 0>;
        clock-names = "aclk";
        iommus = <&smmu_700 0x100>;
        power-domains = <&scmi_devpd 10>;
        pl0: pipeline@0 {
            reg = <0>;
            clocks = <&scmi_clk 1>;
            clock-names = "pxclk";
            pl_id = <0>;
            ports {
                #address-cells = <1>;
```

```
                #size-cells = <0>;
                port@0 {
                    reg = <0>;
                    dp_pl0_out0: endpoint {
                        remote-endpoint = <&vencoder_in>;
                    };
                };
            };
        };

        pl1: pipeline@1 {
            reg = <1>;
            clocks = <&scmi_clk 2>;
            clock-names = "pxclk";
            pl_id = <1>;
            ports {
                #address-cells = <1>;
                #size-cells = <0>;
                port@0 {
                    reg = <0>;
                };
            };
        };
    };
```

4.3.5 Device tree for SoC peripherals

Ethernet:

```
    ethernet@18000000 {
        compatible = "smsc,lan91c111";
        reg = <0x0 0x18000000 0x0 0x10000>;
        interrupts = <0 109 4>;
    };
```

Arm MMC PL180:

```
    mmci@1c050000 {
        compatible = "arm,pl180", "arm,primecell";
        reg = <0x0 0x001c050000 0x0 0x1000>;
        interrupts = <0 107 0x4>,
                 <0 108 0x4>;
        cd-gpios = <&sysreg 0 0>;
        wp-gpios = <&sysreg 1 0>;
        bus-width = <8>;
        max-frequency = <12000000>;
        vmmc-supply = <&fixed_3v3>;
        clocks = <&bp_clock24mhz>, <&bp_clock24mhz>;
        clock-names = "mclk", "apb_pclk";
    };
```

Arm PL050:

```
    kmi@1c060000 {
        compatible = "arm,pl050", "arm,primecell";
        reg = <0x0 0x001c060000 0x0 0x1000>;
        interrupts = <0 197 4>;
        clocks = <&bp_clock24mhz>, <&bp_clock24mhz>;
        clock-names = "KMIREFCLK", "apb_pclk";
    };
```

```
kmi@1c070000 {
    compatible = "arm,pl050", "arm,primecell";
    reg = <0x0 0x001c070000 0x0 0x1000>;
    interrupts = <0 103 4>;
    clocks = <&bp_clock24mhz>, <&bp_clock24mhz>;
    clock-names = "KMIREFCLK", "apb_pclk";
};
```

```
virtio_block@1c130000 {
    compatible = "virtio,mmio";
    reg = <0x0 0x1c130000 0x0 0x200>;
    interrupts = <0 204 4>;
};

sysreg: sysreg@1c010000 {
    compatible = "arm,vexpress-sysreg";
    reg = <0x0 0x001c010000 0x0 0x1000>;
    gpio-controller;
    #gpio-cells = <2>;
};
```

4.4 Build and package the TC kernel

Before building the kernel, make sure you have generated the configuration file as described in kernel configuration section 4.2. Then configure the compiler as follows:

build-scripts/config/common.config

```
# Kernel
LINUX_SRC=$SRC_DIR/linux
LINUX_OUTDIR="$OUTPUT_DIR/tmp_build/linux/"
LINUX_COMPILER=$AARCH64_LINUX/aarch64-none-linux-gnu
LINUX_IMAGE_TYPE="Image"
# Kernel selftest
KSELFTEST_SRC=$LINUX_SRC/tools/testing/selftests
KSELFTEST_LIST="mte bti pauth"
KSELFTEST_ROOTFS_OVERLAY=$BUILDROOT_ROOTFS_OVERLAY/usr/bin
```

Build the code using the following script:

```
./build-scripts/build-linux.sh -f FILESYSTEM [-p PLATFORM] [-t
TC_TARGET_FLAVOR] -g [TC_GPU] [CMD...]
```

```
    make O=$LINUX_OUTDIR ARCH=arm64 CROSS_COMPILE=$LINUX_COMPILER- olddefconfig
    make O=$LINUX_OUTDIR ARCH=arm64 CROSS_COMPILE=$LINUX_COMPILER- -j $PARALLELISM
$LINUX_IMAGE_TYPE
    make O=$LINUX_OUTDIR ARCH=arm64 CROSS_COMPILE=$LINUX_COMPILER- -j $PARALLELISM
$LINUX_IMAGE_TYPE modules
    # GCC12.2 patch: specify EXTRA_CFLAGS to disable Werror=format-truncation for
'util/machine.c'.
    # Note: re-evaluate the need for this patch if GCC version changes or Werror is
removed.
    EXTRA_CFLAGS="${EXTRA_CFLAGS} -Wno-error=format-truncation"
    make EXTRA_CFLAGS="${EXTRA_CFLAGS}" O=$LINUX_OUTDIR ARCH=arm64
CROSS_COMPILE=$LINUX_COMPILER- -j $PARALLELISM tools/perf
    install -D $LINUX_OUTDIR/tools/perf/perf $BUILDROOT_ROOTFS_OVERLAY/bin/perf

    info_echo "Building kernel selftest"
    make O=$LINUX_OUTDIR ARCH=arm64 KBUILD_OUTPUT=$LINUX_OUTDIR
CROSS_COMPILE=$LINUX_COMPILER- -C $KSELFTEST_SRC TARGETS=arm64
ARM64_SUBTARGETS="$KSELFTEST_LIST" INSTALL_PATH=$LINUX_OUTDIR/selftest install
    mkdir -p $KSELFTEST_ROOTFS_OVERLAY
    cp -r $LINUX_OUTDIR/selftest $KSELFTEST_ROOTFS_OVERLAY/
    popd

    pushd $ARM_FFA_TEE_SRC
    make KDIR=$LINUX_OUTDIR CROSS_COMPILE=$LINUX_COMPILER-
BUILD_DIR=$ARM_FFA_TEE_OUTDIR module
    # signing the module
    $LINUX_OUTDIR/scripts/sign-file sha1 $LINUX_OUTDIR/certs/signing_key.pem
$LINUX_OUTDIR/certs/signing_key.x509 $ARM_FFA_TEE_OUTDIR/arm-ffa-tee.ko
    install -D $ARM_FFA_TEE_OUTDIR/arm-ffa-tee.ko $BUILDROOT_ROOTFS_OVERLAY/root/arm-
ffa-tee.ko
    popd
```

Debian example:

```
./run_docker.sh ./build-linux.sh  -f debian -p tc2 -t fvp  -g hwr build
```

Generated kernel image and debug ELF:

```
output/debian/tmp_build/linux/arch/arm64/boot/Image
output/debian/tmp_build/linux/vmlinux
```

4.5 Build the Linux file system and run Linux

4.5.1 Run the TC kernel with Debian

This chapter describes how Debian is supported in TC2. It describes the default configuration with no graphics support. Later sections include a graphics and display demonstration using the Mali DDK.

To run the TC2 software stack, refer to the TC2 User Guide. There are four stages:

1. Fetch the TC2 software stack source code.
2. Setup the environment for software compilation.
3. Build each software component and organize the binary file.
4. Run the software stack on TC2 FVP.

To support Debian, let us examine the third stage and fourth stage in the previous process in detail.

In the third stage, the software components are built in sequence. From the software stack perspective, they are: RSS, SCP, TF-A, Hafnium, OPTEE, Trusted Services, U-boot, and the Linux kernel. Debian and other OSs have the same software components. After all the software components mentioned are built, Debian is built as the last build step.

Before the build, some setup is necessary:

```
export PLATFORM=tc2
export FILESYSTEM=debian
export TC_TARGET_FLAVOR=fvp
cd build-scripts
```

This Debian file system is created by the build script `build-debian.sh`.

```
do_build()
{

    info_echo "Downloading Debian 12 image from Linaro"
    TEMP_FILE=.tmp
    pushd $DEPLOY_DIR/tc2

    if [ ! -f $DEBIAN_IMG ]; then
        wget --output-document=${TEMP_FILE}  ${DEBIAN_LINARO_PATH}/${DEBIAN_IMG}
        checksum=`md5sum ${TEMP_FILE} | sed 's/\|/ /'|awk '{print $1}'`

        if [ $checksum != $DEBIAN_CHECKSUM ]; then
            error_echo "Debian 12 Image Checksum validation failed"
            rm ${TEMP_FILE}
            exit
        else
            mv ${TEMP_FILE} ${DEBIAN_IMG}
        fi
    else
        info_echo "Debian 12 image already exists; skipping the download process."
    fi

    $SCRIPT_DIR/create_mmc_image.sh
    popd
}
```

The script does the following:

Downloads the Debian image from Linaro website.

```
https://releases.linaro.org/debian/images/developer-arm64/debian12/debian.img
```

Puts the OS image to MMC storage, in order to build the `mmc` image. This functionality is implemented in the build script `create_mmc_image.sh` as follows:

```
echo "Creating MMC bootable debian image"

# MMC Bootable image
OUT_IMG=${OUT_IMG:-debian_fs.img}

size_in_mb() {
        local size_in_bytes
        size_in_bytes=$(wc -c $1)
        size_in_bytes=${size_in_bytes%% *}
        echo $((size_in_bytes / 1024 / 1024 + 1))
}

# Debian Raw filesystem
DEBIAN_IMG=${DEBIAN_IMG:-debian.img}
DEBIAN_SIZE=$(size_in_mb ${DEBIAN_IMG})
IMAGE_LEN=$((DEBIAN_SIZE + 2 ))

# measured in MBytes
PART1_START=1
PART1_END=$((PART1_START + DEBIAN_SIZE))

PARTED="parted -a min "

# Create an empty disk image file
dd if=/dev/zero of=$OUT_IMG bs=1M count=3K

# Create a partition table
$PARTED $OUT_IMG unit s mktable gpt

# Create partitions
SEC_PER_MB=$((1024*2))
$PARTED $OUT_IMG unit s mkpart debian ext4 $((PART1_START * SEC_PER_MB)) $((PART1_END *
SEC_PER_MB - 1))

# Assemble all the images into one final image (There is one debian image as of Today)
dd if=$DEBIAN_IMG of=$OUT_IMG bs=1M seek=${PART1_START} conv=notrunc
```

In the fourth stage, the prepared `mmc` image for Debian is passed to the FVP using a runtime parameter, `-C board.mmc.p_mmc_file` as in the following `run_model.sh` script:

```
DEPLOY_DIR=$RUN_SCRIPTS_DIR/../../output/${DISTRO}/deploy/tc2/
DEB_MMC_IMAGE_NAME=debian_fs.img
...
     debian)
        DISTRO_MODEL_PARAMS="--data board.dram=${DEPLOY_DIR}/Image@0x80000 \
                    -C board.mmc.p_mmc_file=$DEPLOY_DIR/$DEB_MMC_IMAGE_NAME"
        BL1_IMAGE_FILE="$DEPLOY_DIR/bl1-tc.bin"
        FIP_IMAGE_FILE="$DEPLOY_DIR/fip_gpt-tc.bin"
        RSS_ROM_FILE="$DEPLOY_DIR/rss_rom.bin"
        RSS_CM_PROV_BUNDLE="$DEPLOY_DIR/rss_encrypted_cm_provisioning_bundle_0.bin"
        RSS_DM_PROV_BUNDLE="$DEPLOY_DIR/rss_encrypted_dm_provisioning_bundle.bin"
```

When the TC2 software stack runs on the TC2 FVP, the TC22 FVP, the kernel, and the Debian file system run in sequence:

```
./run-scripts/tc2/run_model.sh  -m <FVP_TC2> -d debian
```

After performing the previous steps, Debian boots and runs. You can log in to the Debian OS and run shell commands but there is no graphics display. There are two ways you can show graphics on a terminal:

Software rendering (`TC_GPU=swr`)

GPU hardware rendering (`TC_GPU=hwr`)

The difference between software rendering and hardware rendering is that, for software rendering the instructions are running on the CPU, but for hardware rendering, the rendering functionality is offloaded to the GPU.

To support hardware rendering, you need the GPU hardware as well as the GPU driver in the Linux kernel. The current Mali GPU driver for Arm Mali GPU supports Linux mainline kernel v5.10, and Android Common Kernel v5.15 and v6.1. If you intend to use a different newer mainline kernel other than v5.10, software rendering is one option.

In this section, we describe the different steps required for software rendering.

The Debian image used in TC2 is downloaded from Linaro website without any modification. To use software rendering, the three packages listed below are needed. The default Debian image does not contain the required packages, so you must prepare the Debian image with the required packages.

```
weston mesa-utils libgl1-mesa-glx
```

Before executing `build-debian.sh`, put the prepared Debian image with the name `debian.img` in the TC2 deploy directory `$tc2_workspace/output/debian/deploy/tc2/`. Then build Debian as normal.

You may need to change the size of the MMC image which contains the Debian image. To do this, in `create_mmc_image.sh`, change the following line:

```
dd if=/dev/zero of=$OUT_IMG bs=1M count=3K
```

Change the value from 3K to 6K, or a larger number depending on the size of image.

After the TC2 software stack is up and running, log into Debian and run the following commands:

```
mkdir -p /tmp/wayland && chmod 700 /tmp/wayland
export XDG_RUNTIME_DIR=/tmp/wayland/
export WAYLAND_DISPLAY=wayland-1
export LD_LIBRARY_PATH=/usr/lib/aarch64-linux-gnu/
# Launch weston
weston --backend=drm-backend.so --tty=1 --idle-time=0 --drm-device=card0&
```

You should be able to see that the display terminal is rendered from a black to a grey background.

The following sections show how to use GPU Hardware Rendering. In the FVP model, there is a GPU hardware model of the Arm Mali Titan GPU.

For the graphics software, in TC2, two parts are relevant:

The first related part is the third stage (the build stage) mentioned above. In this stage, after the Debian image is built, add another step to build the graphics software. This functionality is implemented in build script `build-debian-ddk.sh`.

The following commands set up and run the script:

```
export GPU_DDK_REPO=<PATH TO GPU DDK SOURCE CODE>
export GPU_DDK_VERSION="releases/r41p0_01eac0"
```

```
export LM_LICENSE_FILE=<LICENSE FILE>
export ARMLMD_LICENSE_FILE=<LICENSE FILE>
export ARMCLANG_TOOL=<PATH TO ARMCLANG TOOLCHAIN>
cd build-scripts

./run_docker.sh build-debian-ddk.sh build
./run_docker.sh build-debian-ddk.sh deploy
```

The script does three things:

Downloads the GPU driver source code from the git repo:

```
 <PATH TO GPU DDK SOURCE CODE>
 branch: releases/r41p0_01eac0
```

Builds the source code. It builds a kernel module, user space libraries, and binaries:

```
mali_kbase.ko
src/debian/mali/product/build_mali/install/lib/
src/debian/mali/product/build_mali/install/bin/
```

Combines the generated files to a tarball. It includes libraries, binaries, a kernel module, and a user space script.

```
output/debian/deploy/tc2/ddk/lib/aarch64-linux-gnu/mali/wayland/lib*
output/debian/deploy/tc2/ddk/lib/aarch64-linux-gnu/mali/wayland/bin
output/debian/deploy/tc2/ddk/lib/aarch64-linux-gnu/mali/wayland/mali_kbase.ko
output/debian/deploy/tc2/ddk/lib/aarch64-linux-gnu/mali/wayland/run_weston.sh
```

The following points apply after Debian Linux is up and running.

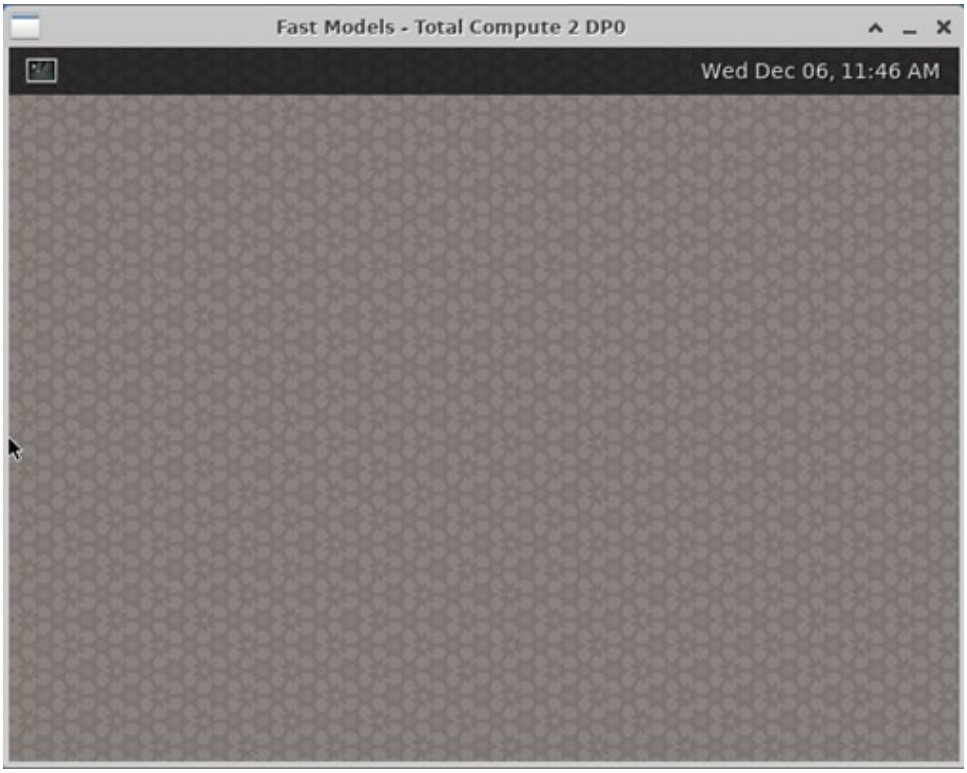
```
./run-scripts/tc2/run_model.sh -m <model_binary_path> -d debian
```

When Debian is running, remote copy the prepared Mali archive file to the running system.

```
scp -P 8022 <tc_workspace>/output/debian/deploy/tc2/ddk/lib/aarch64-linux-
gnu/mali.tar.xz root@localhost:/lib/aarch64-linux-gnu/
Password: root
```

Log in to Debian, extract the file, and run the user space script `run_weston.sh`. This script inserts the kernel module and launches Weston, which is used to render the graphics terminal.

```
cd /lib/aarch64-linux-gnu/
tar -xvf mali.tar.xz
./mali/wayland/run_weston.sh hwr
```



Copy the Mali DDK kernel driver and library.

Copying over the secure shell can be time consuming but Arm recommends this method because it does not require root permissions. Alternatively, to speed up the process, you can mount the Debian image and copy these files to the image. There are multiple ways to do this. For example:

```
# Get the first free loop device, let's say /dev/loopX
losetup -f
   => /dev/loopX: /dev/loop15
# Create a partitioned loop device with the image
sudo losetup -P /dev/loopX ./output/debian/deploy/tc2/debian_fs.img

# Mount the first partition to any location (for example: /mnt)
sudo mount /dev/loopXp1 /mnt/

# Copy the file
sudo cp -avr ./output/debian/deploy/tc2/ddk/lib/aarch64-linux-gnu/mali
/mnt/lib/aarch64-linux-gnu/

# Umount and release the loop device (don't forget to do this or the FS might be
corrupted)
sudo umount /mnt
sudo losetup -d /dev/loopX
```

4.5.2 Run the TC kernel with Buildroot

This chapter explains how the TC platform supports the Buildroot OS, and examines when the TC software stack is built and how Buildroot is integrated into TC software stack. You can use this as an example, to integrate other OSs into the TC software stack.

The following components of the software stack running on TC are built individually in sequence:

```
scp, hafnium, linux, optee-os, u-boot, trusted-services, tfa, rss, flash image, ml app
and buildroot.
```

Hafnium, OPTEE-OS, and trusted services are for secure world, running at S-EL2, S-EL1, and S-EL0 respectively.

SCP, TFA, and RSS are firmware for processors.

A flash image is built to combine the binaries.

The ML app is for machine learning benchmarking.

The remaining components are Linux, U-Boot, and Buildroot. U-Boot is the bootloader for the OS, Linux is to build OS kernel, Buildroot is to build OS rootfs.

As an example, consider building `buildroot` for the OS rootfs .The required script is `build-scripts/build-buildroot.sh`.

```
    make BR2_EXTERNAL=$BUILDROOT_EXTERNAL_TREE O=$BUILDROOT_OUT defconfig
BR2_DEFCONFIG=$BUILDROOT_CFG/defconfig
    AARCH64_LINUX="$AARCH64_LINUX" make BR2_EXTERNAL=$BUILDROOT_EXTERNAL_TREE
O=$BUILDROOT_OUT all
```

In the `buildroot` configuration file, the instructions build the `rootfs` image. The configuration file is `build-scripts/files/buildroot/defconfig`.

```
BR2_TARGET_ROOTFS_CPIO=y
BR2_TARGET_ROOTFS_CPIO_GZIP=y
BR2_TARGET_ROOTFS_CPIO_UIMAGE=y
```

Then the second build command builds the `rootfs` image.The generated OS `rootfs` image file is `output/buildroot/tmp_build/buildroot/images/rootfs.cpio.gz`. Also, in the file `build-buildroot.sh`, the `deploy` command combines three images into a single image.

The three images are the generated OS kernel, the OS rootfs, and the device tree binary.

```
$UBOOT_OUTDIR/tools/mkimage -f $BUILDROOT_CFG/fit-image.its $DEPLOY_DIR/$PLATFORM/tc-
fitImage.bin
```

The file `build-scripts/files/buildroot/fit-image.its` shows the three images.

```
output/buildroot/tmp_build/linux/arch/arm64/boot/Image - OS kernel
output/buildroot/tmp_build/tfa/build/tc/debug/fdts/tc.dtb - device tree binary
output/buildroot/tmp_build/buildroot/images/rootfs.cpio.gz - OS rootfs
```

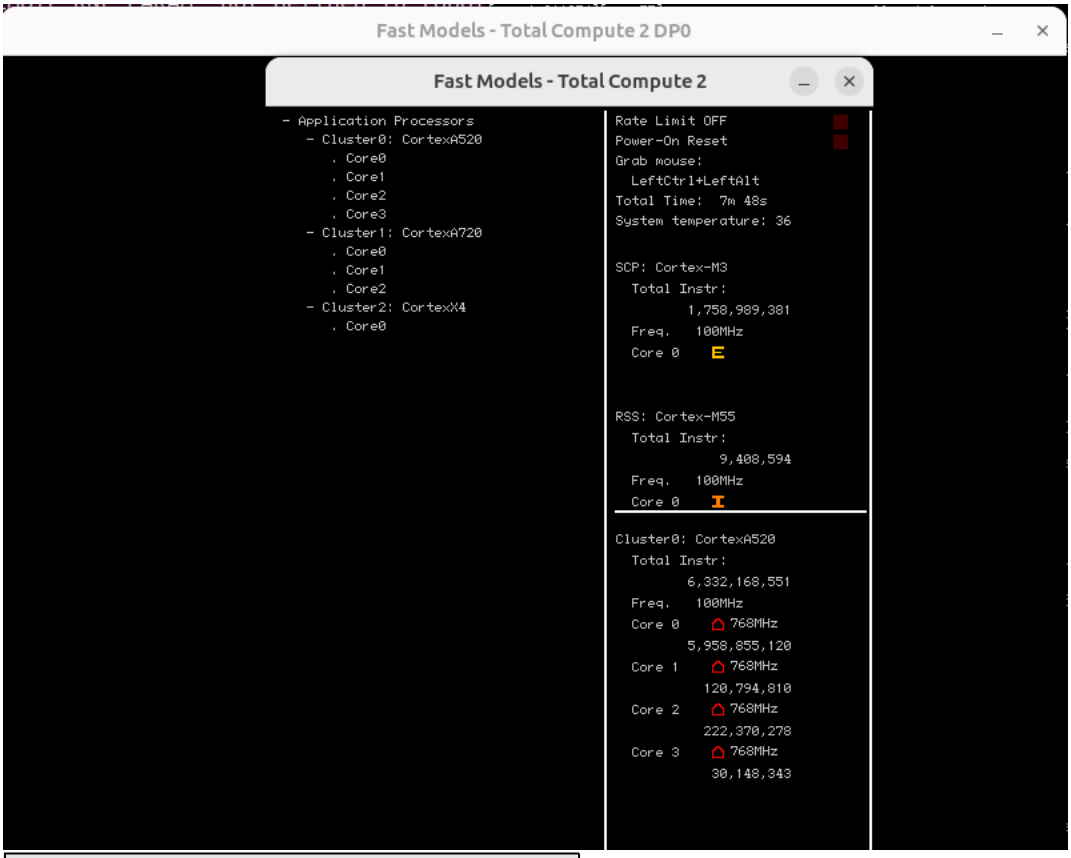
After the build process, you can run the TC software stack. The run script is `run-scripts/tc2/run_model.sh`. This script runs the FVP model with FVP model parameters, one of which points to the OS image, as follows:

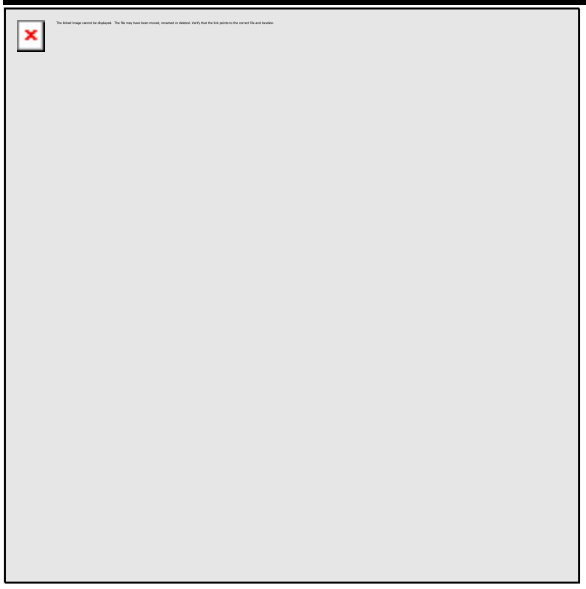
```
"--data board.dram=${DEPLOY_DIR}/tc-fitImage.bin@0x20000000"
```

When the TC software stack is running, the boot process runs the following in sequence:

1. RSS
2. SCP
3. TFA
4. U-Boot
5. Linux
6. The Buildroot filesystem.

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

./run-scripts/tc2/run_model.sh  -m
~/work/Model/FastModels/SubSystemModels/FVP_TC2_11.23_17/norm/models/Linux64_GCC-
9.3/FVP_TC2 -d buildroot
MODEL=/home/yupluo01/work/Model/FastModels/SubSystemModels/FVP_TC2_11.23_17/norm/models
/Linux64_GCC-9.3/FVP_TC2
DISTRO=buildroot
TAP_INTERFACE=
NETWORKING=user
AVB=false

Launching model: FVP_TC2

Fast Models [11.23.17 (Oct 23 2023)]
Copyright 2000-2023 ARM Limited.
All Rights Reserved.

Info: /OSCI/SystemC: Simulation stopped by user.
DISTRO_MODEL_PARAMS=--data board.dram=./run-
scripts/tc2/../../output/buildroot/deploy/tc2//tc-fitImage.bin@0x20000000
Enabling user networking
+ ~/work/Model/FastModels/SubSystemModels/FVP_TC2_11.23_17/norm/models/Linux64_GCC-
9.3/FVP_TC2 -C board.flashloader0.fname=./run-
scripts/tc2/../../output/buildroot/deploy/tc2//fip_gpt-tc.bin -C
soc.pl011_uart0.out_file=/home/yupluo01/work/dev_platform/tc2_workspace/buildroot/run-
scripts/tc2/logs/buildroot/2024_01_08_02_09_PM/uart0_soc.log -C
soc.pl011_uart0.unbuffered_output=1 -C
css.pl011_uart_ap.out_file=/home/yupluo01/work/dev_platform/tc2_workspace/buildroot/run
-scripts/tc2/logs/buildroot/2024_01_08_02_09_PM/uart_ap.log -C
css.pl011_uart_ap.unbuffered_output=1 -C
css.pl011_uart1_ap.out_file=/home/yupluo01/work/dev_platform/tc2_workspace/buildroot/ru
n-scripts/tc2/logs/buildroot/2024_01_08_02_09_PM/uart1_ap.log -C
css.pl011_uart1_ap.unbuffered_output=1 -C displayController=2 -C
css.rss.rom.raw_image=./run-scripts/tc2/../../output/buildroot/deploy/tc2//rss_rom.bin
-C css.rss.VMADDRWIDTH=19 -C css.rss.CMU0_NUM_DB_CH=16 --data css.rss.sram0=./run-
scripts/tc2/../../output/buildroot/deploy/tc2//rss_encrypted_cm_provisioning_bundle_0.b
in@0x0 --data css.rss.sram1=./run-
scripts/tc2/../../output/buildroot/deploy/tc2//rss_encrypted_dm_provisioning_bundle.bin
@0x80000 -C css.cluster0.subcluster0.has_ete=1 -C css.cluster0.subcluster1.has_ete=1 -C
css.cluster0.subcluster2.has_ete=1 -C board.smsc_91c111.enabled=1 -C
board.hostbridge.userNetworking=1 -C
'board.hostbridge.userNetPorts="5555=5555,8080=80,8022=22"' --data board.dram=./run-
scripts/tc2/../../output/buildroot/deploy/tc2//tc-fitImage.bin@0x20000000
```





Note

There is no GUI to show.

5 Porting a new Linux Kernel to the TC platform

There are many Linux kernel trees, for example, the Linux kernel tree, maintainer kernel tree, arm64 kernel tree, RT kernel tree, stable kernel tree, Google Android Common Kernel, and kernel trees maintained by SoC partners.

Because the TC hardware was initially targeted to mobile systems, the Android Common Kernel (ACK) was chosen as the base. Arm have added a lot of necessary patches, and have written new kernel drivers to empower TC hardware. For the relevant commits, check the following link under the `tc_main` branch:

tc_main[17]

This porting guide demonstrates one example for porting one different kernel version to explore on the Debian system with a software-rendering display. Usually, all TC kernel drivers and patches should be applied to the chosen kernel, however, this is not possible due to huge amount of necessary coding and debugging work. Additionally, some patches dedicated for Android may not be needed for the Debian system. Despite this, the following section demonstrates Mali D71 patches on the FVP.

5.1 Choosing a suitable kernel

One thing to consider is whether the kernel version is appropriate for architecture enablement, to match the latest Arm processors. Refer to the Linux edition[18] support for the Arm architecture.

This porting guide uses the stable Linux kernel v6.1.25 as an example:

<make use of TC code tree path >

```
cd $tc2_workspace
mv linux linux-OLD
git clone https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git -b v6.1.25
```

5.2 Update the device tree and kernel configuration

The intention of this Linux porting guide is to assist DTV partners to port a new Linux software stack from the TC kernel. For this reason, the device tree is mostly unchanged in this guide because the hardware is not changed. For more information, refer to chapter4.3. One necessary change is to edit `tc.dts` to adapt the new SCMI kernel driver:

```
src/trusted-firmware-a/fdts/tc.dts
@@ -240,7 +240,11 @@
                #size-cells = <1>;
                ranges = <0 0x0 0x06000000 0x8000>;

-               cpu_scp_scmi_mem: scp-shmem@0 {
+               cpu_scp_scmi_mem0: scp-sram-section0@0 {
+                       compatible = "arm,scmi-shmem";
+                       reg = <0x0 0x80>;
+               };
+               cpu_scp_scmi_mem1: scp-sram-section1@0 {
                        compatible = "arm,scmi-shmem";
                        reg = <0x0 0x80>;
                };
@@ -275,27 +279,29 @@
                interrupts = <0x0 460 0x4>;
        };

-       scmi {
-               compatible = "arm,scmi";
-               mbox-names = "tx", "rx";
-               mboxes = <&mbox_db_tx 0 0 &mbox_db_rx 0 0 >;
-               shmem = <&cpu_scp_scmi_mem &cpu_scp_scmi_mem>;
-               #address-cells = <1>;
-               #size-cells = <0>;
+       firmware {
+               scmi {
+                       compatible = "arm,scmi";
+                       mbox-names = "tx", "rx";
+                       mboxes = <&mbox_db_tx 0 0 &mbox_db_rx 0 0 >;
+                       shmem = <&cpu_scp_scmi_mem0 &cpu_scp_scmi_mem1>;
+                       #address-cells = <1>;
+                       #size-cells = <0>;

-               scmi_devpd: protocol@11 {
-                       reg = <0x11>;
-                       #power-domain-cells = <1>;
-               };
```

```
+                       scmi_devpd: protocol@11 {
+                               reg = <0x11>;
+                               #power-domain-cells = <1>;
+                       };

-               scmi_dvfs: protocol@13 {
-                       reg = <0x13>;
-                       #clock-cells = <1>;
-               };
```

In the future you may decide to adapt software on a new hardware platform, so the device tree might be updated. For example, you might want to use fewer CPUs. In that case, you would need to change the CPU node as described in Device tree for CPU node .

You can reduce the size of the kernel image by removing some modules and drivers in the kernel configuration. Also, the standard Linux kernel and the TC kernel have some differences that must be changed. In this porting guide, we chose to combine the `arm64` default configuration and the `TC22` configuration.

The following two minor changes were made:

```
diff --git a/files/kernel/tc2/base.cfg b/files/kernel/tc2/base.cfg
index 781ee4a..44bb735 100644
--- a/files/kernel/tc2/base.cfg
+++ b/files/kernel/tc2/base.cfg
@@ -3,6 +3,7 @@ CONFIG_ARM_MHU=y
CONFIG_ARM_MHU_V2=y
CONFIG_ARM_SMMU_V3=y
CONFIG_ARM64_VA_BITS_48=y
+CONFIG_DRM=y
CONFIG_DRM_HDLCD=y
CONFIG_DRM_KOMEDA=y
CONFIG_DRM_VIRT_ENCODER=y
diff --git a/files/kernel/tc2/ci700.cfg b/files/kernel/tc2/ci700.cfg
index 50c0153..c5a818f 100644
--- a/files/kernel/tc2/ci700.cfg
+++ b/files/kernel/tc2/ci700.cfg
@@ -1 +1 @@
-CONFIG_ARM_CMN=y
+CONFIG_ARM_CMN=n
```

Enable CONFIG_DRM=y to make CONFIG_DRM_KOMEDA for Mali D71 driver to be built as a built-in module.

Disable CONFIG_ARM_CMN to bypass one CMN perf driver issue on FVP_TC22 11.23 models.

We made changes to `build-linux.sh` as follows to support the new kernel build:

```
build-scripts/build-linux.sh
diff --git a/build-linux.sh b/build-linux.sh
index 56a8054..0f850f6 100755
--- a/build-linux.sh
+++ b/build-linux.sh
```

```
@@ -42,7 +42,7 @@ do_build() {
     if [ "$TC_TARGET_FLAVOR" == "fpga" ]; then
         CONFIG=$CONFIG"$CFG_DIR/fpga.cfg "
     fi
-    CONFIG="$LINUX_SRC/arch/arm64/configs/gki_defconfig "$CONFIG
+    CONFIG="$LINUX_SRC/arch/arm64/configs/defconfig "$CONFIG
     pushd $LINUX_SRC
     scripts/kconfig/merge_config.sh -O $LINUX_OUTDIR -m $CONFIG
     make O=$LINUX_OUTDIR ARCH=arm64 CROSS_COMPILE=$LINUX_COMPILER- olddefconfig
@@ -53,19 +53,6 @@ do_build() {
     EXTRA_CFLAGS="${EXTRA_CFLAGS} -Wno-error=format-truncation"
     make EXTRA_CFLAGS="${EXTRA_CFLAGS}" O=$LINUX_OUTDIR ARCH=arm64
CROSS_COMPILE=$LINUX_COMPILER- -j $PARALLELISM tools/perf
     install -D $LINUX_OUTDIR/tools/perf/perf $BUILDROOT_ROOTFS_OVERLAY/bin/perf
-
-    info_echo "Building kernel selftest"
-    make O=$LINUX_OUTDIR ARCH=arm64 KBUILD_OUTPUT=$LINUX_OUTDIR
CROSS_COMPILE=$LINUX_COMPILER- -C $KSELFTEST_SRC TARGETS=arm64
ARM64_SUBTARGETS="$KSELFTEST_LIST" INSTALL_PATH=$LINUX_OUTDIR/selftest install
-    mkdir -p $KSELFTEST_ROOTFS_OVERLAY
-    cp -r $LINUX_OUTDIR/selftest $KSELFTEST_ROOTFS_OVERLAY/
-    popd
-
-    pushd $ARM_FFA_TEE_SRC
-    make KDIR=$LINUX_OUTDIR CROSS_COMPILE=$LINUX_COMPILER-
BUILD_DIR=$ARM_FFA_TEE_OUTDIR module
-    # signing the module
-    $LINUX_OUTDIR/scripts/sign-file sha1 $LINUX_OUTDIR/certs/signing_key.pem
$LINUX_OUTDIR/certs/signing_key.x509 $ARM_FFA_TEE_OUTDIR/arm-ffa-tee.ko
-    install -D $ARM_FFA_TEE_OUTDIR/arm-ffa-tee.ko $BUILDROOT_ROOTFS_OVERLAY/root/arm-
ffa-tee.ko
-    popd
 }
```

To enable Mali D71 on the kernel for the TC FVP, apply the following two patches:

```
0051-drm-Add-component-aware-simple-encoder.patch

From af3cf4a02e7e5ec1fa82f4526d8803ae079b02e9 Mon Sep 17 00:00:00 2001
From: Tushar Khandelwal <tushar.khandelwal@arm.com>
Date: Tue, 16 Jun 2020 12:39:06 +0000
Subject: [PATCH 051/110] drm: Add component-aware simple encoder

This is a simple DRM encoder that gets its connector timings information
from a OF subnode in the device tree and exposes that as a "discovered"
panel. It can be used together with component-based DRM drivers in an
emulated environment where no real encoder or connector hardware exists
and the display output is configured outside the kernel.

Signed-off-by: Tushar Khandelwal <tushar.khandelwal@arm.com>

Upstream-Status: Backport [https://git.linaro.org/landing-teams/working/arm/kernel-
release.git/commit/?h=latest-armlt&id=15283f7be4b1e586702551e85b4caf06531ac2fc]
Signed-off-by: Arunachalam Ganapathy <arunachalam.ganapathy@arm.com>
Change-Id: Ic68cbba7da7d36ee23359ff53bf30eb44cb78661
---
drivers/gpu/drm/Kconfig               |  11 +
drivers/gpu/drm/Makefile              |   2 +
drivers/gpu/drm/drm_virtual_encoder.c | 299 ++++++++++++++++++++++++++
3 files changed, 312 insertions(+)
create mode 100644 drivers/gpu/drm/drm_virtual_encoder.c
```

```
diff --git a/drivers/gpu/drm/Kconfig b/drivers/gpu/drm/Kconfig
index f30f99166531..f89469bf5991 100644
--- a/drivers/gpu/drm/Kconfig
+++ b/drivers/gpu/drm/Kconfig
@@ -323,6 +323,17 @@ config DRM_VKMS

  If M is selected the module will be called vkms.

+config DRM_VIRT_ENCODER
+       tristate "Virtual OF-based encoder"
+       depends on DRM && OF
+       select VIDEOMODE_HELPERS
+       help
+         Choose this option to get a virtual encoder and its associated
+         connector that will use the device tree to read the display
+         timings information. If M is selected the module will be called
+         drm_vencoder.
+
+
source "drivers/gpu/drm/exynos/Kconfig"

source "drivers/gpu/drm/rockchip/Kconfig"
diff --git a/drivers/gpu/drm/Makefile b/drivers/gpu/drm/Makefile
index 0b283e46f28b..8ffb1718d032 100644
--- a/drivers/gpu/drm/Makefile
+++ b/drivers/gpu/drm/Makefile
@@ -72,6 +72,8 @@ drm_kms_helper-y := drm_bridge_connector.o drm_crtc_helper.o \
drm_kms_helper-$(CONFIG_DRM_PANEL_BRIDGE) += bridge/panel.o
drm_kms_helper-$(CONFIG_DRM_FBDEV_EMULATION) += drm_fb_helper.o
obj-$(CONFIG_DRM_KMS_HELPER) += drm_kms_helper.o
+drm_vencoder-y := drm_virtual_encoder.o
+obj-$(CONFIG_DRM_VIRT_ENCODER) += drm_vencoder.o

#
# Drivers and the rest
diff --git a/drivers/gpu/drm/drm_virtual_encoder.c
b/drivers/gpu/drm/drm_virtual_encoder.c
new file mode 100644
index 000000000000..39a902ecfe32
--- /dev/null
+++ b/drivers/gpu/drm/drm_virtual_encoder.c
@@ -0,0 +1,299 @@
+/*
+ * Copyright (C) 2016 ARM Limited
+ * Author: Liviu Dudau <Liviu.Dudau@arm.com>
+ *
+ * Dummy encoder and connector that use the OF to "discover" the attached
+ * display timings. Can be used in situations where the encoder and connector's
+ * functionality are emulated and no setup steps are needed, or to describe
+ * attached panels for which no driver exists but can be used without
+ * additional hardware setup.
+ *
+ * The encoder also uses the component framework so that it can be a quick
+ * replacement for existing drivers when testing in an emulated environment.
+ *
+ * This file is subject to the terms and conditions of the GNU General Public
+ * License.  See the file COPYING in the main directory of this archive
+ * for more details.
+ *
+ */
+
+#include <drm/drm_crtc.h>
+#include <drm/drm_atomic_helper.h>
+#include <drm/drm_crtc_helper.h>
+#include <drm/drm_probe_helper.h>
+#include <drm/drm_print.h>
```

```
+#include <linux/platform_device.h>
+#include <drm/drm_of.h>
+#include <linux/component.h>
+#include <video/display_timing.h>
+#include <video/of_display_timing.h>
+#include <video/videomode.h>
+
+struct drm_virt_priv {
+      struct drm_connector connector;
+      struct drm_encoder encoder;
+      struct display_timings *timings;
+};
+
+#define connector_to_drm_virt_priv(x) \
+      container_of(x, struct drm_virt_priv, connector)
+
+#define encoder_to_drm_virt_priv(x) \
+      container_of(x, struct drm_virt_priv, encoder)
+
+static void drm_virtcon_destroy(struct drm_connector *connector)
+{
+      struct drm_virt_priv *conn = connector_to_drm_virt_priv(connector);
+
+      drm_connector_cleanup(connector);
+      display_timings_release(conn->timings);
+}
+
+static enum drm_connector_status
+drm_virtcon_detect(struct drm_connector *connector, bool force)
+{
+      return connector_status_connected;
+}
+
+static const struct drm_connector_funcs drm_virtcon_funcs = {
+      .reset = drm_atomic_helper_connector_reset,
+      .detect      = drm_virtcon_detect,
+      .fill_modes = drm_helper_probe_single_connector_modes,
+      .destroy = drm_virtcon_destroy,
+      .atomic_duplicate_state = drm_atomic_helper_connector_duplicate_state,
+      .atomic_destroy_state = drm_atomic_helper_connector_destroy_state,
+};
+
+static int drm_virtcon_get_modes(struct drm_connector *connector)
+{
+      struct drm_virt_priv *conn = connector_to_drm_virt_priv(connector);
+      struct display_timings *timings = conn->timings;
+      int i;
+
+      for (i = 0; i < timings->num_timings; i++) {
+            struct drm_display_mode *mode = drm_mode_create(connector->dev);
+            struct videomode vm;
+
+            if (videomode_from_timings(timings, &vm, i))
+                  break;
+
+            drm_display_mode_from_videomode(&vm, mode);
+            mode->type = DRM_MODE_TYPE_DRIVER;
+            if (timings->native_mode == i)
+                  mode->type = DRM_MODE_TYPE_PREFERRED;
+
+            drm_mode_set_name(mode);
+            drm_mode_probed_add(connector, mode);
+      }
+
+      return i;
+}
```

```
+
+static int drm_virtcon_mode_valid(struct drm_connector *connector,
+                                  struct drm_display_mode *mode)
+{
+      return MODE_OK;
+}
+
+struct drm_encoder *drm_virtcon_best_encoder(struct drm_connector *connector)
+{
+      struct drm_virt_priv *priv = connector_to_drm_virt_priv(connector);
+
+      return &priv->encoder;
+}
+
+struct drm_encoder *
+drm_virtcon_atomic_best_encoder(struct drm_connector *connector,
+                                struct drm_atomic_state *state)
+{
+      struct drm_virt_priv *priv = connector_to_drm_virt_priv(connector);
+
+      return &priv->encoder;
+}
+
+static const struct drm_connector_helper_funcs drm_virtcon_helper_funcs = {
+      .get_modes = drm_virtcon_get_modes,
+      .mode_valid = drm_virtcon_mode_valid,
+      .best_encoder = drm_virtcon_best_encoder,
+      .atomic_best_encoder = drm_virtcon_atomic_best_encoder,
+};
+
+static void drm_vencoder_destroy(struct drm_encoder *encoder)
+{
+      drm_encoder_cleanup(encoder);
+}
+
+static const struct drm_encoder_funcs drm_vencoder_funcs = {
+      .destroy = drm_vencoder_destroy,
+};
+
+static void drm_vencoder_dpms(struct drm_encoder *encoder, int mode)
+{
+      /* nothing needed */
+}
+
+static bool drm_vencoder_mode_fixup(struct drm_encoder *encoder,
+                                    const struct drm_display_mode *mode,
+                                    struct drm_display_mode *adjusted_mode)
+{
+      /* nothing needed */
+      return true;
+}
+
+static void drm_vencoder_prepare(struct drm_encoder *encoder)
+{
+      drm_vencoder_dpms(encoder, DRM_MODE_DPMS_OFF);
+}
+
+static void drm_vencoder_commit(struct drm_encoder *encoder)
+{
+      drm_vencoder_dpms(encoder, DRM_MODE_DPMS_ON);
+}
+
+static void drm_vencoder_mode_set(struct drm_encoder *encoder,
+                                  struct drm_display_mode *mode,
+                                  struct drm_display_mode *adjusted_mode)
+{
```

```
+	/* nothing needed */
+}
+
+static const struct drm_encoder_helper_funcs drm_vencoder_helper_funcs = {
+	.dpms		= drm_vencoder_dpms,
+	.mode_fixup	= drm_vencoder_mode_fixup,
+	.prepare	= drm_vencoder_prepare,
+	.commit		= drm_vencoder_commit,
+	.mode_set	= drm_vencoder_mode_set,
+};
+
+static int drm_vencoder_bind(struct device *dev, struct device *master,
+			     void *data)
+{
+	struct drm_encoder *encoder;
+	struct drm_virt_priv *con;
+	struct drm_connector *connector;
+	struct drm_device *drm = data;
+	u32 crtcs = 0;
+	int ret;
+
+	con = devm_kzalloc(dev, sizeof(*con), GFP_KERNEL);
+	if (!con)
+		return -ENOMEM;
+
+	dev_set_drvdata(dev, con);
+	connector = &con->connector;
+	encoder = &con->encoder;
+
+	if (dev->of_node) {
+		struct drm_bridge *bridge;
+		crtcs = drm_of_find_possible_crtcs(drm, dev->of_node);
+		bridge = of_drm_find_bridge(dev->of_node);
+		if (bridge) {
+			ret = drm_bridge_attach(encoder, bridge, NULL, 0);
+			if (ret) {
+				DRM_ERROR("Failed to initialize bridge\n");
+				return ret;
+			}
+		}
+		con->timings = of_get_display_timings(dev->of_node);
+		if (!con->timings) {
+			dev_err(dev, "failed to get display panel timings\n");
+			return ENXIO;
+		}
+	}
+
+	/* If no CRTCs were found, fall back to the old encoder's behaviour */
+	if (crtcs == 0) {
+		dev_warn(dev, "Falling back to first CRTC\n");
+		crtcs = 1 << 0;
+	}
+
+	encoder->possible_crtcs = crtcs ? crtcs : 1;
+	encoder->possible_clones = 0;
+
+	ret = drm_encoder_init(drm, encoder, &drm_vencoder_funcs,
+			       DRM_MODE_ENCODER_VIRTUAL, NULL);
+	if (ret)
+		goto encoder_init_err;
+
+	drm_encoder_helper_add(encoder, &drm_vencoder_helper_funcs);
+
+	/* bogus values, pretend we're a 24" screen for DPI calculations */
+	connector->display_info.width_mm = 519;
+	connector->display_info.height_mm = 324;
```

```
+	connector->interlace_allowed = false;
+	connector->doublescan_allowed = false;
+	connector->polled = 0;
+
+	ret = drm_connector_init(drm, connector, &drm_virtcon_funcs,
+				 DRM_MODE_CONNECTOR_VIRTUAL);
+	if (ret)
+		goto connector_init_err;
+
+	drm_connector_helper_add(connector, &drm_virtcon_helper_funcs);
+
+	drm_connector_register(connector);
+
+	ret = drm_connector_attach_encoder(connector, encoder);
+	if (ret)
+		goto attach_err;
+
+	return ret;
+
+attach_err:
+	drm_connector_unregister(connector);
+	drm_connector_cleanup(connector);
+connector_init_err:
+	drm_encoder_cleanup(encoder);
+encoder_init_err:
+	display_timings_release(con->timings);
+
+	return ret;
+};
+
+static void drm_vencoder_unbind(struct device *dev, struct device *master,
+				void *data)
+{
+	struct drm_virt_priv *con = dev_get_drvdata(dev);
+
+	drm_connector_unregister(&con->connector);
+	drm_connector_cleanup(&con->connector);
+	drm_encoder_cleanup(&con->encoder);
+	display_timings_release(con->timings);
+}
+
+static const struct component_ops drm_vencoder_ops = {
+	.bind = drm_vencoder_bind,
+	.unbind = drm_vencoder_unbind,
+};
+
+static int drm_vencoder_probe(struct platform_device *pdev)
+{
+	return component_add(&pdev->dev, &drm_vencoder_ops);
+}
+
+static int drm_vencoder_remove(struct platform_device *pdev)
+{
+	component_del(&pdev->dev, &drm_vencoder_ops);
+	return 0;
+}
+
+static const struct of_device_id drm_vencoder_of_match[] = {
+	{ .compatible = "drm,virtual-encoder", },
+	{},
+};
+MODULE_DEVICE_TABLE(of, drm_vencoder_of_match);
+
+static struct platform_driver drm_vencoder_driver = {
+	.probe = drm_vencoder_probe,
+	.remove = drm_vencoder_remove,
```

```
+      .driver = {
+             .name = "drm_vencoder",
+             .of_match_table = drm_vencoder_of_match,
+      },
+};
+
+module_platform_driver(drm_vencoder_driver);
+
+MODULE_AUTHOR("Liviu Dudau");
+MODULE_DESCRIPTION("Virtual DRM Encoder");
+MODULE_LICENSE("GPL v2");
--
2.40.1
```

```
0052-drm-arm-komeda-add-RENDER-capability-to-the-device-n.patch

From 0c806323dc6e7714f822ead4ba3f456ab02616ec Mon Sep 17 00:00:00 2001
From: Tushar Khandelwal <tushar.khandelwal@arm.com>
Date: Wed, 17 Jun 2020 10:49:26 +0000
Subject: [PATCH 052/110] drm: arm: komeda: add RENDER capability to the device
node

this is required to make this driver work with android framework

Signed-off-by: Tushar Khandelwal <tushar.khandelwal@arm.com>

Upstream-Status: Inappropriate [Product specific configuration]
Signed-off-by: Arunachalam Ganapathy <arunachalam.ganapathy@arm.com>
---
drivers/gpu/drm/arm/display/komeda/komeda_kms.c | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

diff --git a/drivers/gpu/drm/arm/display/komeda/komeda_kms.c
b/drivers/gpu/drm/arm/display/komeda/komeda_kms.c
index 451746ebbe71..7f79de60a60a 100644
--- a/drivers/gpu/drm/arm/display/komeda/komeda_kms.c
+++ b/drivers/gpu/drm/arm/display/komeda/komeda_kms.c
@@ -58,7 +58,7 @@ static irqreturn_t komeda_kms_irq_handler(int irq, void *data)
}

static const struct drm_driver komeda_kms_driver = {
-      .driver_features = DRIVER_GEM | DRIVER_MODESET | DRIVER_ATOMIC,
+      .driver_features = DRIVER_GEM | DRIVER_MODESET | DRIVER_ATOMIC | DRIVER_RENDER,
.lastclose                 = drm_fb_helper_lastclose,
DRM_GEM_DMA_DRIVER_OPS_WITH_DUMB_CREATE(komeda_gem_dma_dumb_create),
.fops = &komeda_cma_fops,
--
2.40.1
```

Then build the kernel with the default script command:

```
 ./run_docker.sh ./build-linux.sh
```

5.3 Off-tree kernel module build

5.3.1 OP-TEE driver with FF-A transport support

You get an FFA TEE driver build failure if you do not comment out the build, so you must update the `arm-ffa-tee` driver from the main branch to match the new Linux kernel.

```
cd $tc2_workspace
mv arm-ffa-tee arm-ffa-tee-OLD
git clone https://git.gitlab.arm.com/linux-arm/linux-trusted-services.git  arm-ffa-tee
```

5.3.2 Mali DDK driver

You have to use software rendering if booting Debian with a GUI display.

5.4 Kernel booting

The command line to boot the Kernel is as follows:

```
root@debian-tc:~# cat /proc/cmdline

console=ttyAMA0 debug user_debug=31 earlycon=pl011,0x2A400000 loglevel=9
androidboot.hardware=total_compute androidboot.boot_devices=1c050000.mmci ip=dhcp
androidboot.selinux=permissive allow_mismatched_32bit_el0 systemd.log_level=info
root=/dev/mmcblk0p1 rw
```

The stable Linux image is loaded by U-Boot (we may choose EDK2/uefi instead on TC LSC FVP).

Loading the Kernel image and the Debian boot log produces the following output (with some key information highlighted):

```
U-Boot 2023.04-00009-ge1ab7bef29 (Jan 23 2024 - 05:49:28 +0000)

DRAM:  1.9 GiB (effective 7.9 GiB)
Core:  14 devices, 12 uclasses, devicetree: separate
Flash: 64 MiB
MMC:   mmci@1c050000: 0
Loading Environment from nowhere... OK
In:    serial_pl01x
Out:   serial_pl01x
Err:   serial_pl01x
Hit any key to stop autoboot:  10
TOTAL_COMPUTE# boot
## Checking Image at a0000000 ...
Unknown image format!
Booting Debian
Moving Image from 0x80080000 to 0x80200000, end=826f0000
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
Working FDT set to 83000000
   Loading Device Tree to 000000009fffa000, end 000000009ffff9f6 ... OK
Working FDT set to 9fffa000

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd801]
[    0.000000] Linux version 6.1.25-00004-g4dddb1b7c042 (yupluo01@a015822) (aarch64-
none-linux-gnu-gcc (Arm GNU Toolchain 12.2.Rel1 (Build arm-12.24)) 12.2.1 20221205, GNU
ld (Arm GNU Toolchain 12.2.Rel1 (Build arm-12.24)) 2.39.0.20221210) #4 SMP PREEMPT Tue
Jan 23 05:48:08 UTC 2024
[    0.000000] Machine model: arm,tc
[    0.000000] earlycon: pl11 at MMIO 0x000000002a400000 (options '')
[    0.000000] printk: bootconsole [pl11] enabled
[    0.000000] efi: UEFI not found.
[    0.000000] Reserved memory: created CMA memory pool at 0x00000081f8000000, size 128
MiB
[    0.000000] OF: reserved mem: initialized node linux,cma, compatible id shared-dma-
pool
[    0.000000] NUMA: No NUMA configuration found
[    0.000000] NUMA: Faking a node at [mem 0x0000000080000000-0x00000081ffffffff]
[    0.000000] NUMA: NODE_DATA [mem 0x81f7019a00-0x81f701bfff]
[    0.000000] Zone ranges:
[    0.000000]   DMA      [mem 0x0000000080000000-0x00000000ffffffff]
[    0.000000]   DMA32    empty
[    0.000000]   Normal   [mem 0x0000000100000000-0x00000081ffffffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
```

```
[    0.000000]   node   0: [mem 0x0000000080000000-0x00000000f8ffffff]
[    0.000000]   node   0: [mem 0x0000008080000000-0x00000081ffffffff]
[    0.000000] Initmem setup node 0 [mem 0x0000000080000000-0x00000081ffffffff]
[    0.000000] On node 0, zone Normal: 28672 pages in unavailable ranges
[    0.000000] psci: probing for conduit method from DT.
[    0.000000] psci: PSCIv1.1 detected in firmware.
[    0.000000] psci: Using standard PSCI v0.2 function IDs
[    0.000000] psci: MIGRATE_INFO_TYPE not supported.
[    0.000000] psci: SMC Calling Convention v1.4
[    0.000000] percpu: Embedded 21 pages/cpu s45288 r8192 d32536 u86016
[    0.000000] pcpu-alloc: s45288 r8192 d32536 u86016 alloc=21*4096
[    0.000000] pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3 [0] 4 [0] 5 [0] 6 [0] 7
[    0.000000] Detected PIPT I-cache on CPU0
[    0.000000] CPU features: detected: Address authentication (IMP DEF algorithm)
[    0.000000] CPU features: detected: GIC system register CPU interface
[    0.000000] CPU features: detected: Virtualization Host Extensions
[    0.000000] CPU features: detected: Hardware dirty bit management
[    0.000000] CPU features: detected: Memory Tagging Extension
[    0.000000] CPU features: detected: Asymmetric MTE Tag Check Fault
[    0.000000] CPU features: detected: Spectre-v4
[    0.000000] alternatives: applying boot alternatives
[    0.000000] Fallback order for Node 0: 0
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 2035712
[    0.000000] Policy zone: Normal
[    0.000000] Kernel command line: console=ttyAMA0 debug user_debug=31
earlycon=pl011,0x2A400000 loglevel=9 androidboot.hardware=total_compute
androidboot.boot_devices=1c050000.mmci ip=dhcp androidboot.selinux=permissive
allow_mismatched_32bit_el0 systemd.log_level=info root=/dev/mmcblk0p1 rw
[    0.000000] Dentry cache hash table entries: 1048576 (order: 11, 8388608 bytes,
linear)
[    0.000000] Inode-cache hash table entries: 524288 (order: 10, 4194304 bytes,
linear)
[    0.000000] mem auto-init: stack:all(zero), heap alloc:off, heap free:off
[    0.000000] software IO TLB: area num 8.
[    0.000000] software IO TLB: mapped [mem 0x00000000f4e00000-0x00000000f8e00000]
(64MB)
[    0.000000] Memory: 7876336K/8273920K available (16576K kernel code, 3724K rwdata,
9060K rodata, 7616K init, 612K bss, 266512K reserved, 131072K cma-reserved)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=8, Nodes=1
[    0.000000] rcu: Preemptible hierarchical RCU implementation.
[    0.000000] rcu:     RCU event tracing is enabled.
[    0.000000] rcu:     RCU restricting CPUs from NR_CPUS=256 to nr_cpu_ids=8.
[    0.000000]  Trampoline variant of Tasks RCU enabled.
[    0.000000]  Tracing variant of Tasks RCU enabled.
[    0.000000] rcu: RCU calculated value of scheduler-enlistment delay is 25 jiffies.
[    0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=8
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] GICv3: GIC: Using split EOI/Deactivate mode
[    0.000000] GICv3: 960 SPIs implemented
[    0.000000] GICv3: 1024 Extended SPIs implemented
[    0.000000] Root IRQ handler: gic_handle_irq
[    0.000000] GICv3: GICv3 features: 48 PPIs, DirectLPI
[    0.000000] GICv3: GICv4 features: DirectLPI RVPEID Valid+Dirty
[    0.000000] GICv3: CPU0: found redistributor 0 region 0:0x0000000030080000
[    0.000000] ITS: No ITS available, not enabling LPIs
[    0.000000] rcu: srcu_init: Setting srcu_struct sizes based on contention.
[    0.000000] arch_timer: cp15 timer(s) running at 100.00MHz (phys).
[    0.000000] clocksource: arch_sys_counter: mask: 0x1ffffffffffffff max_cycles:
0x171024e7e0, max_idle_ns: 440795205315 ns
[    0.000000] sched_clock: 57 bits at 100MHz, resolution 10ns, wraps every
4398046511100ns
[    0.000313] Console: colour dummy device 80x25
[    0.000346] Calibrating delay loop (skipped), value calculated using timer
frequency.. 200.00 BogoMIPS (lpj=400000)
[    0.000356] pid_max: default: 32768 minimum: 301
[    0.000389] LSM: Security Framework initializing
```

```
[    0.002156] smp: Bringing up secondary CPUs ...
[    0.002399] Detected PIPT I-cache on CPU1
[    0.002414] cacheinfo: Unable to detect cache hierarchy for CPU 1
[    0.002417] GICv3: CPU1: found redistributor 100 region 0:0x00000000300c0000
[    0.002424] CPU1: Booted secondary processor 0x0000000100 [0x410fd801]
[    0.002693] Detected PIPT I-cache on CPU2
[    0.002710] cacheinfo: Unable to detect cache hierarchy for CPU 2
[    0.002712] GICv3: CPU2: found redistributor 200 region 0:0x0000000030100000
[    0.002719] CPU2: Booted secondary processor 0x0000000200 [0x410fd801]
[    0.002993] Detected PIPT I-cache on CPU3
[    0.003011] cacheinfo: Unable to detect cache hierarchy for CPU 3
[    0.003013] GICv3: CPU3: found redistributor 300 region 0:0x0000000030140000
[    0.003020] CPU3: Booted secondary processor 0x0000000300 [0x410fd801]
[    0.003290] Detected PIPT I-cache on CPU4
[    0.003291] CPU features: SANITY CHECK: Unexpected variation in
SYS_ID_AA64MMFR1_EL1. Boot CPU: 0x1001111011312122, CPU4: 0x1001111010312122
[    0.003297] CPU features: Unsupported CPU feature variation detected.
[    0.003309] cacheinfo: Unable to detect cache hierarchy for CPU 4
[    0.003311] GICv3: CPU4: found redistributor 400 region 0:0x0000000030180000
[    0.003317] CPU4: Booted secondary processor 0x0000000400 [0x410fd811]
[    0.003590] Detected PIPT I-cache on CPU5
[    0.003592] CPU features: SANITY CHECK: Unexpected variation in
SYS_ID_AA64MMFR1_EL1. Boot CPU: 0x1001111011312122, CPU5: 0x1001111010312122
[    0.003609] cacheinfo: Unable to detect cache hierarchy for CPU 5
[    0.003612] GICv3: CPU5: found redistributor 500 region 0:0x00000000301c0000
[    0.003617] CPU5: Booted secondary processor 0x0000000500 [0x410fd811]
[    0.003888] Detected PIPT I-cache on CPU6
[    0.003893] CPU features: SANITY CHECK: Unexpected variation in
SYS_ID_AA64MMFR1_EL1. Boot CPU: 0x1001111011312122, CPU6: 0x1001111010312122
[    0.003911] cacheinfo: Unable to detect cache hierarchy for CPU 6
[    0.003913] GICv3: CPU6: found redistributor 600 region 0:0x0000000030200000
[    0.003919] CPU6: Booted secondary processor 0x0000000600 [0x410fd811]
[    0.004187] Detected PIPT I-cache on CPU7
[    0.004190] CPU features: SANITY CHECK: Unexpected variation in
SYS_ID_AA64MMFR1_EL1. Boot CPU: 0x1001111011312122, CPU7: 0x1001111010312122
[    0.004207] cacheinfo: Unable to detect cache hierarchy for CPU 7
[    0.004209] GICv3: CPU7: found redistributor 700 region 0:0x0000000030240000
[    0.004213] CPU7: Booted secondary processor 0x0000000700 [0x410fd820]
[    0.004228] smp: Brought up 1 node, 8 CPUs
[    0.004330] SMP: Total of 8 processors activated.
[    0.004335] CPU features: detected: Branch Target Identification
[    0.004340] CPU features: detected: ARMv8.4 Translation Table Level
[    0.004344] CPU features: detected: Data cache clean to the PoU not required for I/D
coherence
[    0.004350] CPU features: detected: Common not Private translations
[    0.004356] CPU features: detected: CRC32 instructions
[    0.004360] CPU features: detected: E0PD
[    0.004364] CPU features: detected: Enhanced Counter Virtualization
[    0.004369] CPU features: detected: Enhanced Privileged Access Never
[    0.004374] CPU features: detected: Generic authentication (IMP DEF algorithm)
[    0.004379] CPU features: detected: RCpc load-acquire (LDAPR)
[    0.004385] CPU features: detected: LSE atomic instructions
[    0.004389] CPU features: detected: Privileged Access Never
[    0.004394] CPU features: detected: RAS Extension Support
[    0.004399] CPU features: detected: Speculation barrier (SB)
[    0.004404] CPU features: detected: Stage-2 Force Write-Back
[    0.004408] CPU features: detected: TLB range maintenance instructions
[    0.004412] CPU features: detected: WFx with timeout
[    0.004418] CPU features: detected: Speculative Store Bypassing Safe (SSBS)
[    0.004430] CPU features: detected CPU7: Activity Monitors Unit (AMU)
[    0.004430] CPU features: detected CPU6: Activity Monitors Unit (AMU)
[    0.004430] CPU features: detected CPU5: Activity Monitors Unit (AMU)
[    0.004430] CPU features: detected CPU3: Activity Monitors Unit (AMU)
[    0.004431] CPU features: detected CPU4: Activity Monitors Unit (AMU)
[    0.004430] CPU features: detected CPU1: Activity Monitors Unit (AMU)
[    0.004431] CPU features: detected CPU0: Activity Monitors Unit (AMU)
```

```
[    0.004431] CPU features: detected CPU2: Activity Monitors Unit (AMU)
[    0.004469] CPU: All CPU(s) started at EL2

...

[    0.048668] arm-smmu-v3 3f000000.smmu_700: ias 40-bit, oas 40-bit (features
0x0004dfef)
[    0.048832] arm-smmu-v3 3f000000.smmu_700: allocated 65536 entries for cmdq
[    0.048975] arm-smmu-v3 3f000000.smmu_700: allocated 32768 entries for evtq
[    0.048981] arm-smmu-v3 3f000000.smmu_700: 2-level strtab only covers 25/32 bits of
SID
[    0.050271] arm-smmu-v3 3f000000.smmu_700: msi_domain absent - falling back to wired
irqs
...
[    0.104719] cpu cpu0: EM: created perf domain
[    0.104905] cpu cpu4: EM: created perf domain
[    0.105093] cpu cpu7: EM: created perf domain
[    0.105872] [drm] Found ARM Mali-D71 version r0p0
[    0.105906] [drm] Pipeline-0: n_layers: 4, n_scalers: 2, output: single-link.
[    0.105910] [drm]    output_link[0]: vencoder.
[    0.105912] [drm]    output_link[1]: none.
[    0.105918] [drm] Pipeline-1: n_layers: 4, n_scalers: 2, output: single-link.
[    0.105922] [drm]    output_link[0]: none.
[    0.105925] [drm]    output_link[1]: none.
[    0.106043] [drm] CRTC-0: master(pipe-0) slave(pipe-1).
[    0.106046] [drm] CRTC-1: master(pipe-1) slave(pipe-0).
[    0.106585] komeda 2cc00000.display: bound vencoder (ops drm_vencoder_ops)
[    0.106749] [drm] Initialized komeda 0.1.0 20181101 for 2cc00000.display on minor 0
[    0.129202] Console: switching to colour frame buffer device 240x67
[    0.146854] komeda 2cc00000.display: [drm] fb0: komedadrmfb frame buffer device
...

[    2.879294] Run /sbin/init as init process
[    2.879300]   with arguments:
[    2.879304]     /sbin/init
[    2.879308]   with environment:
[    2.879311]     HOME=/
[    2.879316]     TERM=linux
[    2.879319]     user_debug=31
[    3.003516] systemd[1]: systemd 252.12-1~deb12u1 running in system mode (+PAM +AUDIT
+SELINUX +APPARMOR +IMA +SMACK +SECCOMP +GCRYPT -GNUTLS +OPENSSL +ACL +BLKID +CURL
+ELFUTILS +FIDO2 +IDN2 -IDN +IPTC +KMOD +LIBCRYPTSETUP +LIBFDISK +PCRE2 -PWQUALITY
+P11KIT +QRENCODE +TPM2 +BZIP2 +LZ4 +XZ +ZLIB +ZSTD -BPF_FRAMEWORK -XKBCOMMON +UTMP
+SYSVINIT default-hierarchy=unified)
[    3.003550] systemd[1]: Detected architecture arm64.

Welcome to Debian GNU/Linux 12 (bookworm)!

apt install weston mesa-utils libgl1-mesa-glx
mkdir -p /tmp/wayland && chmod 700 /tmp/wayland
export XDG_RUNTIME_DIR=/tmp/wayland/
export WAYLAND_DISPLAY=wayland-1
export LD_LIBRARY_PATH=/usr/lib/aarch64-linux-gnu/

# Launch weston
weston --backend=drm-backend.so --tty=1 --idle-time=0 --drm-device=card0&

https://wayland.freedesktop.org
Bug reports to: https://gitlab.freedesktop.org/wayland/weston/issues/
Build: 10.0.1
[07:20:43.499] Command line: weston --backend=drm-backend.so --tty=1 --idle-time=0 --
drm-device=card0
[07:20:43.499] OS: Linux, 6.1.25-00004-g4dddb1b7c042, #4 SMP PREEMPT Tue Jan 23
05:48:08 UTC 2024, aarch64
[07:20:43.499] Flight recorder: enabled
[07:20:43.499] Starting with no config file.
```

```
[07:20:43.501] Output repaint window is 7 ms maximum.
[07:20:43.501] Loading module '/usr/lib/aarch64-linux-gnu/libweston-10/drm-backend.so'
[07:20:43.521] initializing drm backend
[07:20:43.521] Trying logind launcher...
[07:20:43.521] logind: cannot find systemd session for uid: 0 -61
[07:20:43.521] logind: cannot setup systemd-logind helper error: (No data available),
using legacy fallback
[07:20:43.521] Trying weston_launch launcher...
[07:20:43.521] could not get launcher fd from env
[07:20:43.521] Trying direct launcher...
[07:20:43.521] using /dev/dri/card0
[07:20:43.521] DRM: supports atomic modesetting
[07:20:43.521] DRM: supports GBM modifiers
[07:20:43.521] DRM: supports picture aspect ratio
[07:20:43.522] Loading module '/usr/lib/aarch64-linux-gnu/libweston-10/gl-renderer.so'
```

When this finishes, you can see the same display as shown in the previous chapter.

6 References

[1] https://github.com/ARM-software/SCP-firmware/blob/master/user_guide.md

[2] https://totalcompute.docs.arm.com/en/latest/totalcompute/tc2/user-guide.html

[3] https://arm-software.github.io/CMSIS_5/General/html/index.html

[4] https://arm-software.github.io/CMSIS_5/General/html/cm_revisionHistory.html

[5] https://trustedfirmware-a.readthedocs.io/en/latest/design/firmware-design.html

[6] https://trustedfirmware-a.readthedocs.io/en/latest/plat/arm/tc/index.html

[7] https://totalcompute.docs.arm.com/en/latest/totalcompute/tc2/user-guide.html

[8] https://trustedfirmware-a.readthedocs.io/en/latest/porting-guide.html

[9] https://developer.arm.com/docs/den0006/latest/trusted-board-boot-requirements-client-tbbr-client-armv8-a

[10] https://developer.arm.com/docs/den0006/latest/trusted-board-boot-
https://developer.arm.com/documentation/den0022/latest/-client-tbbr-client-armv8-a
[11] https://trustedfirmware-a.readthedocs.io/en/latest/design/firmware-design.html#firmware-image-package-fip

[12] https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/log/?h=lts-v2.10

[13] https://totalcompute.docs.arm.com/en/tc2-2023.10.04/totalcompute/tc2/user-guide.html#running-buildroot

[14] https://docs.u-boot.org/en/latest/index.html

[15] https://totalcompute.docs.arm.com/en/latest/totalcompute/tc2/user-guide.html#firmware-update

[16] https://totalcompute.docs.arm.com/en/latest/totalcompute/tc2/user-guide.html#build-variants-configuration
[17] https://git.gitlab.arm.com/arm-reference-solutions/linux/-/commits/tc_main/

[18] https://developer.arm.com/Tools%20and%20Software/Linux%20Kernel#Editions