Arm reference solutions

Arunachalam Ganapathy

Sep 21, 2023

TOTAL COMPUTE:

Warning: This release is now considered obsolete and has been replaced by a newer TC release. Please refer to the latest TC release documentation for more details.

CHAPTER

ONE

TOTAL COMPUTE

Warning: This release is now considered obsolete and has been replaced by a newer TC release. Please refer to the latest TC release documentation for more details.

1.1 Total Compute: TC1-2022.05.12

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

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

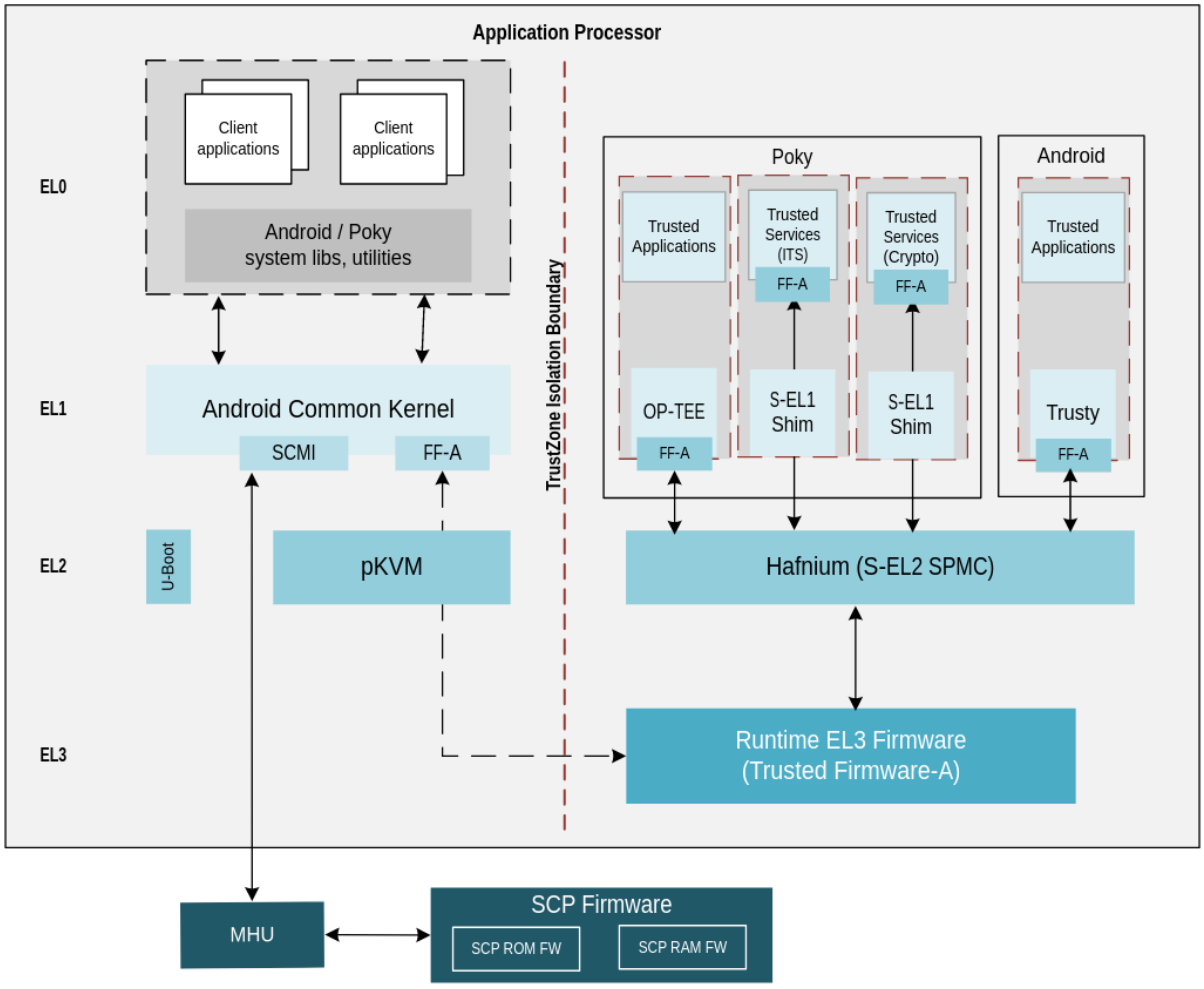
Warning: This release is now considered obsolete and has been replaced by a newer TC release. Please refer to the latest TC release documentation for more details.

1.1.1 Total Compute Platform Software Components

SCP Firmware

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

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


Application Processor
EL0
EL1
EL2
EL3
Client applications
Client applications
Android / Poky system libs, utilities
Android Common Kernel
SCMI
FF-A
U-Boot
pKVM
TrustZone Isolation Boundary
Poky
Trusted Applications
Trusted Services (ITS)
FF-A
Trusted Services (Crypto)
FF-A
OP-TEE
FF-A
S-EL1 Shim
S-EL1 Shim
Android
Trusted Applications
Trusty
FF-A
Hafnium (S-EL2 SPMC)
Runtime EL3 Firmware (Trusted Firmware-A)
MHU
SCP Firmware
SCP ROM FW
SCP RAM FW

SCP Boot ROM

SCP Boot ROM code is the first software that executes right after a cold reset or Power-on. It performs the following functions:

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

SCP Runtime Firmware

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

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

Secure Software

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

AP firmware

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

Trusted Firmware-A (TF-A) BL1

AP Trusted ROM contains an on-chip trusted ROM that runs the boot code on Total Compute platform. BL1 performs minimal architectural initialization (like exception vectors, CPU initialization) and Platform initialization. It loads the BL2 image and passes control to it.

Trusted Firmware-A (TF-A) BL2

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

1. SCP BL2 image
2. EL3 Runtime Software (BL31 image)
3. Secure Partition Manager (BL32 image)
4. Non-Trusted firmware - U-boot (BL33 image)

5. Secure Partitions images (OP-TEE and Trusted Services)

Trusted Firmware-A (TF-A) BL31

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

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

Secure Partition Manager

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

Secure Partitions

Software image isolated using SPM is Secure Partition. Total Compute enables OP-TEE and Trusted Services (Crypto, Internal Trusted Storage) as Secure Partitions.

OP-TEE

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

Trusted Services

Trusted Services like Crypto Service and Internal Trusted Storage runs as S-EL0 Secure Partitions using a Shim layer at S-EL1. These services along with S-EL1 Shim layer are built as a single image. The Shim layer forwards FF-A calls from S-EL0 to S-EL2.

U-Boot

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

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

Kernel

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

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

Android

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

1. Software rendering: This profile has support for Android UI and boots Android to home screen. It uses SwiftShader to achieve this. Swiftshader is a CPU base implementation of the Vulkan graphics API by Google.

Warning: This release is now considered obsolete and has been replaced by a newer TC release. Please refer to the latest TC release documentation for more details.

1.1.2 Instructions: Obtaining Total Compute software deliverables

- To build the TC1 software stack please refer to *user-guide*
- For the list of changes and features added please refer to *change-log*
- For further details on the latest release and features please refer to *release_notes*

1.1.3 TC Software Stack Overview

The TC1 software consists of firmware, kernel and file system components that can run on the associated FVP. Following are the Software components:

1. SCP firmware – System initialization, Clock and Power control
2. AP firmware – Trusted Firmware-A (TF-A)
3. Secure Partition Manager
4. Secure Partitions
    - OP-TEE Trusted OS in Poky
    - Trusted Services with Shim layer in Poky
    - Trusty Trusted OS in Android
5. U-Boot – loads and verifies the fitImage for poky boot, containing kernel and filesystem or boot Image for Android Verified Boot, containing kernel and ramdisk.
6. Kernel – supports the following hardware features

- Mailbox hardware unit
- PAC/MTE/BTI features

7. Android
    - Supports PAC/MTE/BTI features

Total Compute Platform Software Components

Warning: This release is now considered obsolete and has been replaced by a newer TC release. Please refer to the latest TC release documentation for more details.

1.1.4 User Guide

Contents

Notice

The Total Compute 2021 (TC1) software stack uses the Yocto project to build a Board Support Package (BSP) and a choice of Poky Linux distribution or Android userspace. The Yocto project uses Bitbake to build the software stack.

Prerequisites

These instructions assume that:

- Your host PC is running Ubuntu Linux 18.04 LTS.
- You are running the provided scripts in a `bash` shell environment.
- You are using git version of 2.30.0 or higher.

To resolve the dependencies, run:

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
sudo apt-get install chrpath gawk texinfo libssl-dev diffstat wget git-core unzip gcc-
↪multilib \
 build-essential socat cpio python python3 python3-pip python3-pexpect xz-utils␣
↪debianutils \
 iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3 xterm git-
↪lfs openssl curl \
 libncurses5 lib32ncurses5-dev libz-dev python-pip u-boot-tools m4 zip gcc-9 libstdc++6␣
↪liblz4-tool zstd
```

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

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

If syncing and building android, the minimum requirements for the host machine can be found at https://source.android.com/se

- At least 250GB of free disk space to check out the code and an extra 150 GB to build it. If you conduct multiple builds, you need additional space.
- At least 16 GB of available RAM/swap.
- Git configured properly using "git config" otherwise it may throw error while fetching the code.

Syncing and building the source code

There are two distros supported in the TC1 software stack: poky (a minimal distro containing busybox) and android.

To sync code for poky, please follow the steps in "Syncing code" section for BSP only. To sync code for android, please follow the steps for syncing both BSP and Android.

To build the required binaries for poky, please follow the steps in "Board Support Package build" section only. To build the binaries for Android, please follow the steps in both "Board Support Package build" and "Android OS build" sections.

Syncing code

Create a new folder that will be your workspace, which will henceforth be referred to as <tc1_workspace> in these instructions.

```
mkdir <tc1_workspace>
cd <tc1_workspace>
export TC1_RELEASE=refs/tags/TC1-2022.05.12
```

To sync BSP only without Android, run the repo command.

```
repo init -u https://gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest -m tc1.xml -b ${TC1_RELEASE} -g bsp
repo sync -j$(nproc)
```

To sync both the BSP and Android, run the repo command.

```
repo init -u https://gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest -m tc1.xml -b ${TC1_RELEASE} -g android
repo sync -j$(nproc)
```

Board Support Package build

```
cd <tc1_workspace>/bsp
export DISTRO="poky"
export MACHINE="tc1"
source setup-environment
bitbake tc-artifacts-image
```

The initial clean build will be lengthy, given that all host utilities are to be built as well as the target images. This includes host programs (python, cmake, etc.) and the required toolchain(s).

Once the build is successful, all images will be placed in the <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1 directory.

Note that the BSP includes the Poky Linux distribution, which offers BusyBox-like utilities.

Android OS build

- tc1_swr : This supports Android display with swiftshader (software rendering).

The android images can be built with or without authentication enabled using Android Verified Boot(AVB). AVB build is done in userdebug mode and takes a longer time to boot as the images are verified.

The `build-scripts/tc1/build_android.sh` script in `<tc1_workspace>/android` will patch and build android. This can be passed 2 parameters, `-d` for deciding which profile to build and `-a` for enabling AVB. The following command shows the help menu for the script:

```
cd <tc1_workspace>/android
./build-scripts/tc1/build_android.sh  -h
Incorrect script use, call script as:
<path to build_android.sh> [OPTIONS]
OPTIONS:
-d, --distro                              distro version, values supported [android-swr]
-a, --avb                                 [OPTIONAL] avb boot, values supported [true,␣
↪false], DEFAULT: false
```

The `--avb` option does not influence the way the system boots rather it adds an optional sanity check on the prerequisite images.

As an example, to build Android with software rendering and AVB disabled, run the command:

```
./build-scripts/tc1/build_android.sh -d android-swr
```

To build Android with software rendering and AVB enabled, run the command:

```
./build-scripts/tc1/build_android.sh -d android-swr -a true
```

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

Provided components

Within the Yocto project, each component included in the TC1 software stack is specified as a Bitbake recipe. The TC1 recipes are located at `<tc1_workspace>/bsp/layers/meta-arm/`.

Yocto allows modifying the fetched source code of each recipe component in the workspace, by applying patches. This is however not a convenient approach for developers, since creating patches and updating recipes is time-consuming. To make that easier, Yocto provides the devtool utility. Devtool creates a new workspace, in which you can edit the fetched source code and bake images with the modifications

```
cd <tc1_workspace>/bsp
MACHINE=tc1
DISTRO=poky
. ./conf/setup-environment

# create a workspace for a given recipe component
# recipe-component-name can be of:
# trusted-firmware-a / scp-firmware / u-boot / linux-arm64-ack
devtool modify <recipe-component-name>

# This creates a new workspace for recipe-component-name and fetches source code
# into "<tc1_workspace>/build-poky/workspace/sources/{trusted-firmware-a,scp-firmware,u-
↪boot,linux-arm64-ack}"
```

(continues on next page)

(continued from previous page)

```
# edit the source code in the newly created workspace
# build images with changes on workspace
# recipe-component-name can be of: trusted-firmware-a / scp-firmware / u-boot / linux-
↪arm64-ack
bitbake <recipe-component-name>
```

Firmware Components

Trusted Firmware-A

Based on Trusted Firmware-A

Recipe	<tc1_workspace>/bsp/layers/meta-arm/meta-arm-bsp/recipes-bsp/trusted-firmware-a/trusted-firmware-a-tc.inc
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/bl1-tc.bin • <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/fip_gpt.bin

System Control Processor (SCP)

Based on SCP Firmware

Recipe	<tc1_workspace>/bsp/layers/meta-arm/meta-arm-bsp/recipes-bsp/scp-firmware/scp-firmware-tc.inc
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/scp_ramfw.bin • <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/scp_romfw.bin

U-Boot

Based on U-Boot gitlab

Recipe	<tc1_workspace>/bsp/layers/meta-arm/meta-arm-bsp/recipes-bsp/u-boot/u-boot_%.bbappend
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/u-boot.bin

Hafnium

Based on Hafnium

Recipe	<tc1_workspace>/bsp/layers/meta-arm/meta-arm-bsp/recipes-bsp/hafnium/hafnium-tc.inc
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/hafnium.bin

OP-TEE

Based on OP-TEE

Recipe	<tc1_workspace>/bsp/layers/meta-arm/meta-arm-bsp/recipes-security/optee/optee-os-tc.inc
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/optee/tee-pager_v2.bin

S-EL0 trusted-services

Based on Trusted Services

Recipe	<tc1_workspace>/bsp/layers/meta-tc/recipes-security/trusted-services/secure-partitions_git.bb
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/crypto-sp.bin • <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/internal-trusted-storage.bin

Linux

The recipe responsible for building a 5.10 version of the Android Common kernel (ACK).

Recipe	<tc1_workspace>/bsp/layers/meta-arm/meta-arm-bsp/recipes-kernel/linux/linux-arm-platforms.inc
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/Image

Trusty

Based on Trusty

Recipe	<tc1_workspace>/bsp/layers/meta-tc/recipes-security/trusty/trusty_git.bb
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/lk.bin

Distributions

Poky Linux distro

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

Recipe	<tc1_workspace>/bsp/layers/openembedded-core/meta/recipes-core/images/core-image-minimal.bb
Files	• <tc1_workspace>/bsp/build-poky/tmp-poky/deploy/images/tc1/fitImage-core-image-minimal-tc1-tc1

Android

Android S is supported in this release with device profiles suitable for TC1 machine configuration. Android is built as a separate project and then booted with the BSP built by Yocto.

Run scripts

Within the `<tc1_workspace>/bsp/run-scripts/` are several convenience functions for testing the software stack. Usage descriptions for the various scripts are provided in the following sections.

Obtaining the TC1 FVP

The TC1 FVP is available to partners for build and run on Linux host environments. Please contact Arm to have access (support@arm.com).

Running the software on FVP

A Fixed Virtual Platform (FVP) of the TC1 platform must be available to run the included run scripts.

The run-scripts structure is as follows:

```
run-scripts
|--tc1
   |--run_model.sh
   |-- ...
```

Ensure that all dependencies are met by running the FVP: `./path/to/FVP_TC1`. You should see the FVP launch, presenting a graphical interface showing information about the current state of the FVP.

The `run_model.sh` script in `<tc1_workspace>/bsp/run-scripts/tc1` will launch the FVP, providing the previously built images as arguments. Run the `run_model.sh` script:

```
./run_model.sh
Incorrect script use, call script as:
<path_to_run_model.sh> [OPTIONS]
OPTIONS:
-m, --model                     path to model
-d, --distro                    distro version, values supported [poky, android-swr]
-a, --avb                       [OPTIONAL] avb boot, values supported [true, false],␣
↪DEFAULT: false
-t, --tap-interface             [OPTIONAL] enable TAP interface
-e, --extra-model-params        [OPTIONAL] extra model parameters
```

Running Poky

```
./run-scripts/tc1/run_model.sh -m <model binary path> -d poky
```

Running Android

```
If using an android distro, export ANDROID_PRODUCT_OUT variable to point to android out␣
↪directory
for eg. ANDROID_PRODUCT_OUT=<tc1_workspace>/android/out/target/product/tc_swr

For running android with AVB disabled:
 ./run-scripts/tc1/run_model.sh -m <model binary path> -d android-swr

For running android with AVB enabled:
 ./run-scripts/tc1/run_model.sh -m <model binary path> -d android-swr -a true
```

When the script is run, two terminal instances will be launched. terminal_s0 used for the SCP, TF-A, OP-TEE core logs and terminal_s1 used by TF-A early boot, Hafnium, U-boot and Linux.

Once the FVP is running, the SCP will be the first to boot, bringing the AP out of reset. The AP will start booting from its ROM and then proceed to boot Trusted Firmware-A, Hafnium, Secure Partitions (OP-TEE, Trusted Services in Poky and Trusty in Android) then U-Boot, and then Linux and Poky/Android.

When booting Poky the model will boot Linux and present a login prompt. Login using the username `root`. You may need to hit Enter for the prompt to appear.

The OP-TEE and Trusted Services are initialized in Poky distribution. The functionality of OP-TEE and core set of trusted services such as Crypto and Internal Trusted Storage can be invoked only on Poky distribution. For OP-TEE, the TEE sanity test suite can be run using command `xtest`. For Trusted Services, run command `ts-service-test -sg ItsServiceTests -sg PsaCryptoApiTests -sg CryptoServicePackedcTests -sg CryptoServiceProtobufTests -sg CryptoServiceLimitTests -v` for Service API level tests and run command `ts-demo` for the demonstration client application.

On Android distribution, Trusty provides a Trusted Execution Environment (TEE). The functionality of Trusty IPC can be tested using command `tipc-test -t ta2ta-ipc` with root privilege.

Warning: This release is now considered obsolete and has been replaced by a newer TC release. Please refer to the latest TC release documentation for more details.

1.1.5 Release notes - 2022.05.12

Contents

Release tag

The manifest tag for this release is TC1-2022.05.12

Components

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

- Yocto based BSP build supporting Android and Poky distro.
- Trusted firmware-A for secure boot.
- System control processor(SCP) firmware for programming the interconnect, doing power control etc.
- U-Boot bootloader.
- Hafnium for S-EL2 Secure Partition Manager core.
- OP-TEE for Trusted Execution Environment (TEE) in Poky.

- Trusted Services (Crypto and Internal Trusted Storage) in Poky.
- Trusty for Trusted Execution Environment (TEE) with FF-A messaging in Android.

Hardware Features

- Booker CI with Memory Tagging Unit(MTU) support driver in SCP firmware.
- GIC Clayton Initialization in Trusted Firmware-A.
- Mali-D71 DPU and virtual encoder support for display in Linux.
- MHUv2 Driver for SCP and AP communication.
- UARTs, Timers, Flash, PIK, Clock drivers.
- PL180 MMC.
- DynamIQ Shared Unit (DSU) with 8 cores. 1 Makalu ELP + 3 Makalu + 4 Cortex-A510 (R1) cores configuration.

Software Features

- Poky Distribution support.
- Android S Support.
- Android Common Kernel 5.10 with PAC/BTI/MTE
- With Android S support, the KVM default mode of operation is set to `protected`. This is a nVHE based mode with kernel running at EL1.
- Trusted Firmware-A & Hafnium v2.6
- OP-TEE 3.14.0
- Trusty with FF-A messaging
- CI700-PMU enabled for profiling
- Support secure boot based on TBBR specification https://developer.arm.com/documentation/den0006/latest
- System Control Processor (SCP) firmware v2.10
- Build system based on Yocto master
- U-Boot bootloader v2022.01
- Power management features: cpufreq and cpuidle.
- SCMI (System Control and Management Interface) support.
- Verified u-boot for authenticating fit image (containing kernel + ramdisk) during poky boot.
- Android Verified Boot (AVB) for authenticating boot and system image during Android boot.
- Software rendering on Android with DRM Hardware Composer offloading composition to Mali D71 DPU.
- Hafnium as Secure Partition Manager (SPM) at S-EL2.
- OP-TEE as Secure Partition at S-EL1, managed by S-EL2 SPMC (Hafnium)
- Arm FF-A driver and FF-A Transport support for OP-TEE driver in Android Common Kernel.
- OP-TEE Support in Poky distribution. This includes OP-TEE client and OP-TEE test suite.
- Trusted Services (Crypto and Internal Trusted Storage) running at S-EL0.

- Trusted Services test suite added to poky distribution.
- Shim Layer at S-EL1 running on top of S-EL2 SPMC (Hafnium) used by Trusted Services running in S-EL0.
- Tracing - Added support for ETE and TRBE v1.0 in TF-A, kernel and simpleperf. Traces can be captured with simpleperf. However, to enable tracing, the libete plugin has to be loaded while executing the FVP with `--plugin <path to plugin>/libete-plugin.so`

Platform Support

- This Software release is tested on TC1 Fast Model platform (FVP). - Supported Fast model version for this release is 11.17.33

Known issues or Limitations

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

Support

For support email: support-arch@arm.com

Warning: This release is now considered obsolete and has been replaced by a newer TC release. Please refer to the latest TC release documentation for more details.

1.1.6 Change Log

Contents

This document contains a summary of the new features, changes and fixes in each release of TC1 software stack.

Version 2022.05.12

Features added

- Trusty support in Android
- CI700-PMU enabled for profiling
- Added support for ETE and TRBE

Changes

- Updated Android to S
- Updated Trusted Firmware-A & Hafnium to v2.6
- Updated SCP firmware to v2.10
- Updated U-Boot to v2022.01
- Updated Trusted Services
- Updated Yocto repositories
- Deprecated android-nano profile

Version 2021.08.17

Features added

- Memory Tagging Extension (MTE)
- Pointer Authentication Code (PAC)
- Branch Target Identification (BTI)
- Android AOSP to master (May21)
- Android Common Kernel to v5.10
- Trusted Firmware-A & Hafnium to v2.5
- OP-TEE to v3.14.0
- SCP firmware to v2.8
- U-boot to v2021.07
- Yocto to master

Warning: This release is now considered obsolete and has been replaced by a newer TC release. Please refer to the latest TC release documentation for more details.

1.2 Previous releases

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

1.2.1 TC1 release tags

TC1-2022.05.12

TC1-2021.08.17

1.2.2 TC0 release tags

TC0-2022.02.25

TC0-2021.07.31

TC0-2021.04.23

TC0-2021.02.09

CHAPTER

TWO

N1SDP

2.1 Change Log

Contents

This document contains a summary of the incremental features, changes, fixes and known issues in each release of the N1SDP stack. It is listed in the order of latest to oldest.

2.1.1 Tagged Version - N1SDP-2021.10.12

Features and Fixes

- Migration of source code repository from ARM linaro GIT to ARM Gitlab.
- Migration of build system from Yocto to BASH based scripting environment.
- Minimal BusyBox based root filesystem support.
- Rebase to the latest stable Linux kernel version `5.10.61`.
- Migrate to the latest stable EDK2 version `edk2-stable202108`.
- Update Grub to latest release tag grub-2.06.
- Migrate to the latest master version of System Control Processor (SCP) firmware and Trusted Firmware-A components.
- MCC Refresh to v117 to include:
  - QSPI programming speed improvements.

  - Case fans can be controlled through FAN_SPEED IOFPGA register.

Precautions

- The system thermal monitoring and control features are not yet calibrated, therefore do not operate the unit above room temperature (approximately 25°C).
- The N1SDP is intended for use within a laboratory or engineering development environment. Do not use the N1SDP near equipment that is sensitive to electromagnetic emissions, for example, medical equipment.
- Never subject the board to high electrostatic potentials. Observe Electrostatic Discharge (ESD) precautions when handling any board.
  - Always wear a grounding strap when handling the board.
  - Avoid touching the component pins or any other metallic elements.
- Update/Change board firmware only if MCC FW ask to do so, refer to potential damage page for more information
- Kindly note, the USB 3.0 ports and the audio jacks available on the front panel of the N1SDP case are NOT connected and are not usable. They will be removed in the later versions.

Known Issues and Limitations

- Patches providing proof of concept support for Xilinx CCIX accelerator endpoints are no longer included in this release.
- PCIe root port is limited to GEN3 speed due to the on-board PCIe switch itself only supporting up to GEN3 speed.
- Page Request Interface (PRI) feature is not available in both SMMUs interfacing with the PCIe root ports.
- Currently only Micron 8GB single Rank DIMMs (part number: MTA9ASF1G72PZ-2G6D1) and 16GB dual Rank DIMMs (part number:MTA18ASF2G72PDZ-2G6E1) are supported.
- Stability issues have been observed on long stress tests of the system.
- On-board HDMI connection is not supported for graphics output. A PCIe graphics card can be used for graphics support.
- If either of the two boards needs to boot up in a single chip mode with a C2C setup, then the other board should be powered off.
- CCIX port on N1SDP as a PCIe root host is not supported in UEFI EDK2.

Disclaimer

- Limited Testing for now due to the current global scenario, to be revisited once we get back on site.

Support

For support email: support-subsystem-enterprise@arm.com For reporting security vulnerabilities, please refer Vulnerability reporting page.

2.1.2 Change logs for the previous releases

- Refer to the Old Change Log for detailed information on software features and changes for the previous releases.

2.2 CMN-600 perf example on Neoverse N1 SDP

Contents

The goal of this document is to give a short introduction on CMN-600 performance analysis on N1SDP. This includes driver load verification and Linux perf usage examples.

The examples also include system level cache access and traffic to and from PCIe devices from the view of the interconnect.

2.2.1 Support in Arm's Neoverse N1 SDP software release

The software support for CMN-600 performance analysis can be divided into three components:

- The user space Linux perf tool
- The Linux kernel arm-cmn driver
- EDK2 (DSDT table entry)

The default build of the supplied N1SDP software stack will include all necessary changes and patches to test and explore CMN-600 performance analysis.

2.2.2 CMN-600 Topology and NodeIDs on Neoverse N1 SDP

The PMUs in CMN-600 are distributed to the nodes of the mesh interconnect. NodeType specific events are configured per node. Event counting is done by local counters in the XP attached to the node. Global counters are in the Debug Trace Controller (DTC). The arm-cmn driver uses local/global register pairing to provide 64-bit event counters (see "Counter Allocation" section below).

All the nodes are referenced by NodeID and NodeType. PMU events must specify the NodeID of the node on which it is to be counted using the nodeid= parameter. A summary of NodeID can be found in the table below. For more details contact support (support-subsystem-enterprise@arm.com).

Purpose	Node Type	NodeID	Event Name
System-Level Cache slices (SLC)	HN-F	0x24 0x28 0x44 0x48	arm_cmn/hnf
PCI_CCIX (Expansion slot 4)	RN-D	0x08	arm_cmn/rnid
PCI_0 (All other PCI-E)	RN-D	0x0c	arm_cmn/rnid
Mesh interconnections	XP	0x00 0x08 0x20 0x28 0x40 0x48 0x60 0x68	arm_cmn/mxp
Debug Trace Controller	DTC	0x68	arm_cmn/dtc_cycles
ACE-lite slave	SBSX	0x64	arm_cmn/sbsx

For details on what is connected to PCI_0 check the N1SDP TRM (Figure 2-9 PCI Express and CCIX system).

2.2.3 Software components

Linux perf tool

No modifications of `perf` source is needed. The user can opt to use any perf compatible with the built kernel or use the included script `build-scripts/build-perf.sh` to build a static linked binary from the included kernel source (binary is created as `output/n1sdp/perf/perf`).

ACPI DSDT modification

The Linux driver expects a DSDT entry that describe the location of the CMN-600 configuration space. This is included in the supplied N1SDP software stack.

Linux perf driver (arm-cmn)

The included arm-cmn driver is a work-in-progress. A Snapshot of this driver is included in the supplied N1SDP software stack. The driver is controlled by `CONFIG_ARM_CMN` (enabled in default software stack build).

Counter Allocation/Limitation

The arm-cmn driver provides 64-bit event counts for any given event. It accomplishes this using a combination of combined-pair local counters (in a DTM/XP) and uncombined global counters (in the DTC):

- **DTM/XP** Can provide up to two 32-bit local counters (built from paired 16-bit counters por_dtm_pmevcnt0+1, and 2+3) for events from itself and/or up to two devices that are connected to its ports.

  Overflows from these counters are sent to its DTC's global counters. This means only up to 2 events from any of the devices connected to an XP can be counted at the same time without sampling.

- **DTC** Each DTC can provide up to 8 global counters (por_dt_pmevcntA .. H). This means only up to 8 events in a DTC domain can be counted at the same time without sampling.

For example, the N1SDP's two PCI-Express root complexes RND (PCI_CCIX on RND3 at NodeID 0x8 and PCI0 on RND4 at NodeID 0xC), hang off of the same XP (0,1). Only up to 2 RND events from either of the two PCI-E domains can be measured simultaneously without sampling; 3 or more will require sampling.

In the following example, we try to measure 4 RND events, but perf is only giving 50% sampling time for each count because the events have to share local counters in the XP.

```
$ perf stat -a \
    -e arm_cmn/rnid_txdat_flits,nodeid=8/ \
    -e arm_cmn/rnid_txdat_flits,nodeid=12/ \
    -e arm_cmn/rnid_rxdat_flits,nodeid=8/ \
    -e arm_cmn/rnid_rxdat_flits,nodeid=12/ \
    -I 1000
#   time       counts                unit events
1.000089438       0      arm_cmn/rnid_txdat_flits,nodeid=8/     (50.00%)
1.000089438       0      arm_cmn/rnid_txdat_flits,nodeid=12/    (50.00%)
1.000089438       0      arm_cmn/rnid_rxdat_flits,nodeid=8/     (50.00%)
1.000089438       0      arm_cmn/rnid_rxdat_flits,nodeid=12/    (50.00%)
2.000231897      79      arm_cmn/rnid_txdat_flits,nodeid=8/     (50.01%)
2.000231897       0      arm_cmn/rnid_txdat_flits,nodeid=12/    (50.01%)
2.000231897       0      arm_cmn/rnid_rxdat_flits,nodeid=8/     (49.99%)
```

PMU Events

`perf list` shows the perfmon events for the node types that are detected by the arm-cmn driver. If a node type is not detected, perf list will not show the events for that node type.

```
# perf list | grep arm_cmn/hnf
arm_cmn/hnf_brd_snoops_sent/                       [Kernel PMU event]
arm_cmn/hnf_cache_fill/                            [Kernel PMU event]
arm_cmn/hnf_cache_miss/                            [Kernel PMU event]
arm_cmn/hnf_cmp_adq_full/                          [Kernel PMU event]
arm_cmn/hnf_dir_snoops_sent/                       [Kernel PMU event]
arm_cmn/hnf_intv_dirty/                            [Kernel PMU event]
arm_cmn/hnf_ld_st_swp_adq_full/                    [Kernel PMU event]
arm_cmn/hnf_mc_reqs/                               [Kernel PMU event]
arm_cmn/hnf_mc_retries/                            [Kernel PMU event]
[...]
```

The perfmon events are described in the CMN-600 TRM in the register description section for each node type's perf event selection register (at offset 0x2000 of each node that has a PMU).

CMN-600 TRM register summary links to all of the node types and offset registers.

Specifying NodeID to events in perf

To program the CMN-600's PMUs, the NodeIDs of the components need to be specified for each event using a nodeid= parameter. Example:

```
$ perf stat -a -I 1000 -e arm_cmn/hnf_mc_reqs,nodeid=0x24/
```

Multiple nodes can be specified for an event as shown below :

```
$ perf stat -a -I 1000 \
    -e arm_cmn/hnf_mc_reqs,nodeid=0x24/ \
    -e arm_cmn/hnf_mc_reqs,nodeid=0x28/ \
    -e arm_cmn/hnf_mc_reqs,nodeid=0x44/ \
    -e arm_cmn/hnf_mc_reqs,nodeid=0x48/
```

Separate events on the same nodes can be specified as shown below :

```
$ perf stat -a -I 1000 \
    -e arm_cmn/hnf_mc_reqs,nodeid=0x24/ \
    -e arm_cmn/hnf_mc_reqs,nodeid=0x28/ \
    -e arm_cmn/hnf_mc_reqs,nodeid=0x44/ \
    -e arm_cmn/hnf_mc_reqs,nodeid=0x48/ \
    -e arm_cmn/hnf_mc_retries,nodeid=0x24/ \
    -e arm_cmn/hnf_mc_retries,nodeid=0x28/ \
    -e arm_cmn/hnf_mc_retries,nodeid=0x44/ \
    -e arm_cmn/hnf_mc_retries,nodeid=0x48/
```

2.2.4 Driver verification

To verify that the arm-cmn has successfully loaded different ways:

- **Check if any arm_cmn entires is available**

```
$ perf list | grep arm_cmn
arm_cmn/dn_rxreq_dvmop/                              [Kernel PMU event]
arm_cmn/dn_rxreq_dvmop_vmid_filtered/                [Kernel PMU event]
arm_cmn/dn_rxreq_dvmsync/                            [Kernel PMU event]
arm_cmn/dn_rxreq_retried/                            [Kernel PMU event]
arm_cmn/dn_rxreq_trk_occupancy_all/                  [Kernel PMU event]
arm_cmn/dn_rxreq_trk_occupancy_dvmop/                [Kernel PMU event]
[...]
```

- **Sysfs entries**

```
$ ls -x /sys/bus/event_source/devices/arm_cmn/
cpumask
dtc_domain_0
events
format
perf_event_mux_interval_ms
power
subsystem
type
uevent
```

2.2.5 Example

HN-F PMU

Make sure to issue some memory load operation(s) in parallel, such as memtester, while executing the following perf example.

Memory Bandwidth using hnf_mc_reqs

Measure memory bandwidth using hnf_mc_reqs; assumes bandwidth comes from SLC misses.

```
   $ perf stat -a -I 1000 \
       -e arm_cmn/hnf_mc_reqs,nodeid=0x24/ \
       -e arm_cmn/hnf_mc_reqs,nodeid=0x28/ \
       -e arm_cmn/hnf_mc_reqs,nodeid=0x44/ \
       -e arm_cmn/hnf_mc_reqs,nodeid=0x48/
2.000394365        121,713,206      arm_cmn/hnf_mc_reqs,nodeid=0x24/
2.000394365        121,715,680      arm_cmn/hnf_mc_reqs,nodeid=0x28/
2.000394365        121,712,781      arm_cmn/hnf_mc_reqs,nodeid=0x44/
2.000394365        121,715,432      arm_cmn/hnf_mc_reqs,nodeid=0x48/
3.000644408        121,683,890      arm_cmn/hnf_mc_reqs,nodeid=0x24/
3.000644408        121,685,839      arm_cmn/hnf_mc_reqs,nodeid=0x28/
3.000644408        121,682,684      arm_cmn/hnf_mc_reqs,nodeid=0x44/
3.000644408        121,685,669      arm_cmn/hnf_mc_reqs,nodeid=0x48/
```

Generic bandwith formula:

```
hnf_mc_reqs/second/hnf node  * 64 bytes = X MB/sec
```

Subsitute with data from perf output:

```
(121713206 + 121715680 + 121712781 + 121715432) * 64 = 29715 MB/sec
```

PCI-E RX/TX bandwidth

The RN-I/RN-D events are defined from the perspective of the bridge to the interconnect, so the "rdata" events are outbound writes to the PCI-E device and "wdata" events are inbound reads from PCI-E.

Measure RND (PCI-E) bandwidth to/from NVMe SSD when running fio

The NVMe SSD (Optane SSD 900P Series) is on PCI-E Root Complex 0 (PCI0, the Gen3 slot, behind the PCI-E switch).

Run `fio` to read from NVME SSD using 64KB block size for 1000 seconds in one terminal:

```
$ fio \
    --ioengine=libaio --randrepeat=1 --direct=1 --gtod_reduce=1 \
    --time_based --readwrite=read --bs=64k --iodepth=64k --name=r0 \
    --filename=/dev/nvme0n1p5 --numjobs=1 --runtime=10000
r0: (g=0): rw=read, bs=(R) 64.0KiB-64.0KiB, (W) 64.0KiB-64.0KiB, (T) 64.0KiB-64.0KiB,␣
↪ioengine=libaio, iodepth=65536
fio-3.1
Starting 1 process
^Cbs: 1 (f=1): [R(1)][0.5%][r=2586MiB/s,w=0KiB/s][r=41.4k,w=0 IOPS][eta 16m:35s]
fio: terminating on signal 2

r0: (groupid=0, jobs=1): err= 0: pid=1443: Thu Dec 19 12:12:10 2019
   read: IOPS=41.3k, BW=2581MiB/s (2706MB/s)(12.3GiB/4894msec) <--------------------------
↪------ read bandwidth = 2706 MB/sec
   bw (  MiB/s): min= 2276, max= 2587, per=98.10%, avg=2532.02, stdev=125.43, samples=6
   iops        : min=36418, max=41392, avg=40512.33, stdev=2006.90, samples=6
  cpu          : usr=3.15%, sys=35.15%, ctx=16686, majf=0, minf=1049353
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=0.1%, >=64=100.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.1%
     issued rwt: total=202101,0,0, short=0,0,0, dropped=0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=65536

Run status group 0 (all jobs):
   READ: bw=2581MiB/s (2706MB/s), 2581MiB/s-2581MiB/s (2706MB/s-2706MB/s), io=12.3GiB␣
↪(13.2GB), run=4894-4894msec

Disk stats (read/write):
  nvme0n1: ios=202009/2, merge=0/19, ticks=4874362/51, in_queue=3934760, util=98.06%
```

Measure with `perf` in an other terminal. Measure rdata/wdata beats. Each beat is 32 bytes.

```
$ perf stat -a -I 1000 -e arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/
#    time                 counts                  unit events
1.001019030                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (49.92%)
2.001297917                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (49.61%)
3.002325835                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (50.35%)
4.003352327                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (50.15%)
5.004378592                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (49.65%)
6.005297570                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (50.04%)
7.006323364                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (50.35%)
8.007349052                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (49.75%)
9.008374834                  0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (49.74%)
10.009297068                 0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (50.35%)
11.010322898                 0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (50.05%)
12.011349043                 0      arm_cmn/rnid_s0_rdata_beats,nodeid=0xc/           ␣
↪                         (49.65%)
```

Calculate read bandwidth from perf measurement:

```
84.74e6 rdata beats * 32 bytes per beat = 2711 MB/sec
```

Measure RND (PCI-E) bandwidth from Ethernet NIC

`netperf` is executed on the N1SDP to generate network traffic.

`netperf` executing in on terminal…

```
$ netperf -D 10 -H remote-server-in-astlab -t TCP_MAERTS -l 0
Interim result:  941.52 10^6bits/s over 10.000 seconds ending at 1576269135.608
Interim result:  941.52 10^6bits/s over 10.000 seconds ending at 1576269145.608
Interim result:  941.52 10^6bits/s over 10.000 seconds ending at 1576269155.608
Interim result:  941.52 10^6bits/s over 10.000 seconds ending at 1576269165.608
```

…and `perf` in an other at the same time.

```
$ perf stat -e arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/ -I 1000

#    time                 counts                  unit events
1.001024520                  0      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
2.002071830                  9      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
3.003109931                  0      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
4.004134740                  9      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
```

(continues on next page)

(continued from previous page)

```
5.005170700                0      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
6.006232432                0      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
7.007267874                9      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
8.008294424                0      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
9.009329587                9      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
10.010391542               0      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
11.011426046               9      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
12.012453000               0      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
13.013488283               9      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
14.014549938               0      arm_cmn/rnid_s0_wdata_beats,nodeid=0xc/
```

Calculate bandwidth from perf measurement:

```
4.024e6 wdata beats/second * 32 bytes/beat * 8 bits/byte = 1030e6 bits/second
```


2.3 CoreSight support on Neoverse N1 SDP

Contents

2.3.1 CoreSight Introduction

CoreSight Debug and Trace components are invariably described in a graph-like topology which describes the port connections amongst the various components. The topology includes, but is not limited to, the following major components.

- ETM(s)
- ETF(s)/ETB(s)
- Dynamic/Static Funnel(s)
- Dynamic/Static Replicator(s)
- TPIU(s)/ETR(s)

In CoreSight terminology, a component can be roughly described as a source - that produces/generates the debug & trace data, and/or a sink - that consumes/stores the data, or a link - which routes the data. Depending on the "type" of component, it may have 1 or more input port(s), 1 or more output port(s), a single input port only, or a single output port only.

For instance -

- ETM is a source component which only has a single output port.
- TPIU/ETR is a sink component which only has a single input port.
- Funnel is a link component which can be thought of as a MUX having multiple input ports and a single output port.
- Replicator is a link component which can be thought of as a DEMUX having a single input port and multiple output ports.
- ETF is a link component which can act both as a source and a sink (it has a FIFO buffer to store the data) and thus has a single input and a single output port.

Refer to CoreSight_Technical_Introduction for more information about CoreSight Debug and Trace elements.

It is worth mentioning that all static components in CoreSight world are not addressable and only facilitate intermediate connections between the other non-static/Dynamic CoreSight components. They are, however, described in the DT/ACPI and get discovered by their respective kernel drivers.

CoreSight components can be described either/both in a device tree or/and in a DSDT table within ACPI - similar to other components/peripherals, to get enumerated by the Linux kernel.

Every CoreSight component has a corresponding kernel driver which takes care of its initialization. There are configuration changes required in the kernel to build the appropriate CoreSight components' drivers.

Upon a successful probe of a CoreSight component driver, the particular component gets enumerated under `/dev` and appears under `/sys/bus/coresight/devices/` in the booted kernel.

2.3.2 Enabling CoreSight on N1SDP

- ACPI bindings:

  CoreSight topology for N1SDP has been described as a _DSD graph within DSDT table for N1SDP package - the support is enabled by default.

- Linux Kernel:

  The default configuration for the kernel is set to build with the CoreSight drivers. The build flag to enable CoreSight kernel configuration option to build CoreSight kernel drivers is `LINUX_CORESIGHT_BUILD_ENABLED` which is set to `1` by default.

2.3.3 Verifying CoreSight support

Assuming the kernel has been built with the CoreSight configuration, the booted kernel should have CoreSight components enumerated under sysfs within `/sys/bus/coresight/devices/`. The components should also be seen listed under `/dev`.

2.3.4 Running perf with CoreSight

CoreSight framework has been integrated with the standard perf core in the kernel to assist with trace capturing and decoding. To exercise this, perf needs to be built with openCSD (Open CoreSight Decoding) library support.

Execute the following script to build the perf executable with open CSD library

```
./build-scripts/build-perf.sh
```

This will generate the perf executable in the output/n1sdp/build_artifact/ directory which can then be transferred to the kernel running on the N1SDP board.

Here's an example showcasing trace capture and decode of a simple application:

1. Build the demo application code:

   aarch64-linux-gnu-gcc -static -o test.out main.c

```
#include <stdio.h>

int main()
{
        printf("N1SDP\n");

        return 0;
}
```

2. Disassemble the binary:

   aarch64-linux-gnu-objdump -D test.out

```
0000000000400580 <main>:
400580:        a9bf7bfd        stp        x29, x30, [sp,#-16]!
400584:        910003fd        mov        x29, sp
400588:        90000000        adrp        x0, 400000 <_init-0x3b0>
40058c:        9118e000        add        x0, x0, #0x638
400590:        97ffffa4        bl        400420 <puts@plt>
400594:        52800000        mov        w0, #0x0
400598:        a8c17bfd        ldp        x29, x30, [sp],#16
40059c:        d65f03c0        ret
```

3. Trace the application from the start address `0x400580` to `0x4006f0`

   ./perf record -e cs_etm/@tmc_etr0/u –filter 'start 0x400580@test, stop 0x4006f0@test' –per-thread ./test

   This step will generate *perf.data* in the current working directory.

4. Decode the trace data with perf

   ./perf report –stdio –dump

   This step will dump all the captured trace data on stdio.

Refer to the man page of `perf-record` for more information on perf options, filters, etc.

2.3.5 References

- http://infocenter.arm.com/help/topic/com.arm.doc.epm039795/coresight_technical_introduction_EPM_039795.pdf?_ga=2.263237196.1385850732.1581332707-419757503.1576059061
- http://infocenter.arm.com/help/topic/com.arm.doc.101489_0000_01_en/arm_neoverse_n1_system_development_platform_technical_reference_manual_101489_0000_01_en.pdf
- https://www.linaro.org/blog/stm-and-its-usage/

2.4 Errata-1315703 WA disabled in Neoverse N1 SDP

In Trusted Firmware-A, all erratum are disabled by default including the 1315703. If we want any erratum to be enabled, then we have to explicitly enable them in the platform Makefile.

The N1SDP stack disables the workaround for Erratum 1315703 by default, so that the N1 CPU performance in N1SDP better reflects that of released versions of the N1 for software that does not require mitigation for Spectre Variant 4.

N1SDP uses N1 version r1p0, which is affected by Erratum 1315703, which is fixed in N1 r3p1. The workaround for r1p0 disables the CPU performance feature of bypassing of stores by younger loads. This can significantly affect performance. The Erratum is classified "Cat A (Rare)" and requires a specific sequence of events to occur.

Disabling this CPU performance feature is also the mitigation for Spectre Variant 4 (CVE-2018-3639). On CPUs that provide the PSTATE.SBSS feature, the OS selectively applies the mitigation only to programs that require it, leaving the performance of other programs unaffected. However, N1 r1p0 does not have the PSTATE.SBSS feature (which is introduced in N1 r3p1), and TF-A does not provide the interface to dynamically disable the CPU performance feature. Therefore, applying the workaround penalizes ALL software running on N1SDP, including those that do not require the mitigation.

Disabling Errata-1315703 is meant for performance evaluation purposes ONLY and should not be used for software that requires a seccomp computing environment.

2.5 Multichip SMP boot on Neoverse N1 SDP with CCIX

Contents

- *References*

2.5.1 Introduction

The Neoverse N1 System Development Platform (N1SDP) supports dual socket operation wherein two N1SDP boards are connected over a CCIX enabled PCIe link. One board is designated as the master board whose CCIX enabled PCIe controller is configured in root port mode and the other board is configured as slave board whose CCIX enabled PCIe controller is configured in endpoint mode. Linux boots in the master board and powers ON the slave board cores such that the operating system sees CPU cores and memories from both master and slave boards in separate NUMA nodes making a dual socket SMP configuration.

Note: Only cores and DDR memory from slave chip are exposed to the operating system. No I/O peripherals from slave chip are exposed to the operating system.

2.5.2 Connection Details

For the purpose of connecting two N1SDP boards over PCIe link, a specialized PCIe riser card (Part No: V2M-N1SDP-0343A) has to be used. The riser card package includes two PCIe form factor riser cards, two high speed low latency PCIe cables and one USB cable (for PCIe REFCLK). A 20-pin flat ribbon cable that comes along with N1SDP board accessories should also be used. This is used for I2C connection between master and slave board SCP & MCP and other timer synchronization handshake signals.

Setup has to be made following the steps given below:

1. Designate one N1SDP board as master board and the other N1SDP board as slave board.
2. Ensure that both the boards are powered off.
3. Plug in the riser cards, one in each board, in the CCIX slot (the last x16 PCIe slot close to the edge of the board).
4. Connect the high speed PCIe cables between the riser cards in respective slots (make one to one connection and not criss-cross connection).
5. Connect the USB cable between the riser cards.
6. Connect the 20-pin flat ribbon cable between the boards to the connector named as 'C2C' which should be in the back side of the board.
7. Connect the USB/SATA boot media to the respective slot in the master board.

2.5.3 Board Files Setup

1. Power ON the master board and open the MCC console of master board using a terminal application (like minicom or putty).
2. Run USB_ON command which mounts the micro SD card content of master board in host machine.
3. Assuming the micro SD card is mounted with the name 'MASTER' in host. Open the file MASTER/MB/HBI0316A/board.txt file and note the APPFILE name. It should be like 'io_v123f.txt' or similar.
4. Now close the board.txt file and open the io_v123f.txt (the one noted in step 3) which will be in the same folder.
5. Set the C2C_ENABLE flag as TRUE and C2C_SIDE flag as MASTER.
6. Set the SCC PLATFORM_CTRL register to enable multichip mode and CHIPID=0. For this, uncomment the SOCCON with offset 0x1170 and set the value 0x00000100. The line should now read: **SOCCON: 0x1170 0x00000100 ; SoC SCC PLATFORM_CTRL**

7. Save and close the file. If the host is Linux run a 'sync' command in one of the terminals to be safe.
8. Repeat from step 1 to 4 for slave board. Let's assume host has mounted the slave board's micro SD card with name 'SLAVE'.
9. Set the C2C_ENABLE flag as TRUE and C2C_SIDE flag as SLAVE.
10. Set the PLATFORM_CTRL register to enable multichip mode and CHIPID=1. For this, uncomment the SOCCON with offset 0x1170 and set the value 0x00000101. Save and close the file. Run sync command in case of Linux host. The line should now read: **SOCCON: 0x1170 0x00000101 ; SoC SCC PLATFORM_CTRL**

2.5.4 Booting the Setup

1. Copy all the firmware binaries to SOFTWARE folder in both master and slave board's micro SD card. Run sync command in case of Linux host. Note that uefi.bin file is not required to be copied to slave micro SD card as UEFI will be run only in the master chip. For getting the sources and building the binaries please refer to user guide
2. Shutdown both the boards.
3. Reboot the slave board first and let it boot. Then reboot the master board. This is done because the SCP firmware running in master board expects the slave board to respond to the I2C command when it boots. If the slave board is not responding for the I2C command then master assumes that it is running in single chip environment and continues to boot in single chip mode. This is explicitly done to avoid any delays in waiting for slave to respond that will affect single chip environment where there is no slave connected at all.
4. If master SCP finds a slave connected and responding then master SCP will perform several handshakes with slave SCP to bring-up the CCIX link and boot the slave chip's cores.

2.5.5 After Booting

1. UEFI's Dynamic ACPI framework exposes both master and slave chip's processors and memories to Linux. Assuming 16GB DDR memory connected each on master and slave boards, Linux will see 8 cores (4 cores in master chip + 4 cores in slave chip) and 32GB DDR memory space.
2. Once Linux has booted, /proc/cpuinfo and /proc/meminfo can be dumped to see the total core and memory information that Linux currently sees.
3. Also the slave board resources (slave CPUs and slave DDR memory) is treated as separate NUMA node in Linux which can be seen using the numactl –hardware command.

2.5.6 Limitations

1. Though the multichip high speed connection is made using CCIX enabled PCIe controllers which supports GEN4 speed, only GEN3 speed has been validated for multichip operation. This affects the cross-chip memory access latency.
2. The timer synchronization internal logic doesn't account for the external sync signals pad timings. So the PIK clock for timer synchronization module has to be reduced to 150MHz from actual 800MHz.
3. The REFCLK counter values on both master and slave chips has to be reset before starting the synchronization process.
4. Timer synchronization interrupt flag has no information on the source of interrupt so the synchronization is retriggered everytime an interrupt is hit.

2.5.7 References

- http://infocenter.arm.com/help/topic/com.arm.doc.101489_0000_01_en/arm_neoverse_n1_system_development_platform_technical_reference_manual_101489_0000_01_en.pdf
- https://www.ccixconsortium.com/ccix-library/

2.6 PCIe SR-IOV on Neoverse N1 SDP

Contents

2.6.1 Introduction

The Neoverse N1 System Development Platform (N1SDP) supports two PCIe root ports each supporting the standard PCIe features including the Single Root I/O Virtualization (SR-IOV) feature. This document gives an overview of how to enable and test the SR-IOV feature on N1SDP.

Note: Due to the PCIe limitations in N1SDP platform (pcie-support), the SR-IOV feature has not been completely validated.

2.6.2 Pre-Requisite

1. Latest software stack synced by following steps given in user guide
2. Ensure that the Linux tree in the synced workspace contains the SR-IOV and PCI ACS override patches. This should have been already applied when syncing the code.

2.6.3 Steps to verify SR-IOV

1. Add the kernel config `CONFIG_IXGBEVF=y` to the file `linux/arch/arm64/configs/defconfig`. This is required to enable the drivers for the mentioned Intel card.
2. Build the software stack and flash the Ubuntu image onto the boot device. Do initial boot of the board which installs Ubuntu and perform second boot which boots Ubuntu kernel.
3. Login to target Ubuntu console and edit the file /etc/default/grub. Add pcie_acs_override=id:13b5:0100 option to the GRUB_CMDLINE_LINUX_DEFAULT. Save this file and run update-grub command.

4. Reboot the board from Ubuntu console using reboot now command.
5. Now Linux probes and assigns separate IOMMU groups for all PCIe devices.
6. Virtual functions can be enabled from sysfs using following command:

   echo 63 > /sys/bus/pci/devices/0001:01:00.0/sriov_numvfs

   Note that in test environment the Intel card's ethernet port 0 is identified in Segment:1 Bus:1 Dev:0 Function:0

2.6.4 Limitations

1. SR-IOV feature is only supported in CCIX slot and not in the PCIe slots. This is due to the on-board PCIe switch not supporting the ARI capability to which the PCIe slots are connected.
2. Only Intel X540-T2 card has been validated for the SR-IOV feature.

2.6.5 References

- http://infocenter.arm.com/help/topic/com.arm.doc.101489_0000_01_en/arm_neoverse_n1_system_development_platform_technical_reference_manual_101489_0000_01_en.pdf

---

2.7 PCIe support on Neoverse N1 SDP

Contents

2.7.1 Support for PCIe in Arm's Neoverse N1 SDP software releases

The supplied Neoverse software stack includes support for PCIe. However there are two issues (detailed below) with the PCIe root port hardware IP present on Neoverse N1 SDP. These impact generic code in EDK II UEFI and Linux.

Arm is implementing workarounds and maintaining these workaround patches to Linux Kernel in the Linux Arm git-lab repository. The patches will be rebased when the Linux kernel is migrated.

The patches are applied automatically as part of the Arm reference platform software workspace sync process, providing a software stack with full PCIe support. However, note that it is not possible to use an unmodified Linux distribution release on Neoverse N1SDP, without patching and rebuilding the kernel shipped with that distribution.

2.7.2 SLVERR on PCIe device and function enumeration

Linux and EDK II UEFI use standardized PCIe enumeration code based on the assumption that PCIe compliant root ports must return magic number 0xFFFFFFFF when either a device is not connected or a function is not implemented. The PCIe root port hardware IP on Neoverse N1 SDP boards instead asserts an AXI SLVERR response, triggering a bus fault on the applications processor performing PCIe enumeration.

A software workaround has been implemented in the System Control Processor (SCP) that performs minimal PCIe enumeration and generates a bus/device/function (BDF) table that it places in shared Non-secure SRAM. During this minimal enumeration, the SCP ignores any bus faults from accessing PCIe configuration space, effectively suppressing the non-compliant AXI SLVERR responses generated by the PCIe root port hardware IP. Patches have also been applied to EDK II UEFI and Linux to use the generated BDF table during their own PCIe enumeration routines.

2.7.3 Non-contiguous configuration space

Linux and EDK II UEFI use standardized PCIe enumeration code based on the assumption that the PCIe Enhanced Configuration Access Mechanism (ECAM) base address is the same as the base address of the root port's configuration space. These assumptions are not valid for N1SDP leading to issues identifying the root port during PCIe enumeration.

A software workaround has been added to the SCP that inserts the base address of the PCIe root port's configuration space as the first word in the BDF table in shared Non-secure SRAM in both EDK II UEFI and Linux. The generic PCIe code has been patched to read the base address of the PCIe root port configuration space from the BDF table; this base address is then automatically used whenever a given BDF is all zeroes.

2.7.4 PCIE Root Port config space supports only 32 bits R/W

The root port configuration space supports only 32-bit accesses. However, standard UEFI and Linux drivers may perform 8-bit and 16-bit read/write to this space which will end up in erroneous result. To avoid this, software workaround has been added to convert the 8-bit/16-bit access to 32-bit access using read-modify-write method.

2.8 Arm Reference Platforms

Contents

2.8.1 Introduction

Arm produces open source software stacks for a variety of platforms, serving as a reference to help enable product development based on Arm IP for a range of target markets and applications.

We use these software stacks to demonstrate:

- Integration of multiple open source software components, including firmware, operating systems, applications, and services.
- Upstream support for configuring the latest Arm IP.
- Software features including secure services and power management.
- Alignment with Arm specifications and open source community initiatives.

2.8.2 Neoverse N1 Software Development Platform

Neoverse N1 SDP is an infrastructure segment development platform available for licensing to Arm partners. The platform supports a reference open-source software stack based on Trusted Firmware-A, SCP-firmware, EDK II UEFI, GRUB, Linux, and user-space components. Supporting code is available either directly in the relevant upstream projects, or VIA public-facing git repositories.

The software release supports Linux stable kernel booting Ubuntu server distribution or an optional minimal BusyBox filesystem.

Please follow the user guide to sync, build, and run the software stack on the N1SDP.

More details are available in the following Neoverse N1 SDP getting started guide.

For questions and discussions, visit our Arm community forums For technical support, please contact us at support@arm.com

---

2.9 User Guide

Contents

2.9.1 Introduction

The Neoverse N1 System Development Platform (N1SDP) is an enterprise class reference board based on the Neoverse N1 core.

This document is a guide on how to fetch, build from source, and run an Arm Reference Platforms software stack on N1SDP, including a Linux distribution based on either the Ubuntu server distribution or a minimal BusyBox based root filesystem.

The synced workspace includes:

- Scripts to build the board firmware, Linux kernel, and Ubuntu server distribution image.
- Ubuntu server distribution, sources for Linux Kernel, EDK2, firmware and BusyBox.
- Other supporting board files and prebuilt binaries.

2.9.2 Host prerequisites

These instructions assume that:

- Your host PC is running Ubuntu Linux 18.04 LTS and should have at least 50GB free storage space
- You are running the provided scripts in a `bash` shell environment.

The following utilities must be available on your host PC:

- chrpath
- compression library
- diffstat
- gawk
- makeinfo
- openssl headers
- pip
- repo

To install all the required packages, run:

```
sudo apt-get update
sudo apt-get install autoconf autopoint bc bison build-essential \
   curl debianutils device-tree-compiler dosfstools flex gettext-base git \
   libssl-dev m4 mtools parted pkg-config python python3 python3-distutils \
   rsync unzip uuid-dev wget
```

Configure Git, run:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

Install repo tool

Follow the instructions provided in the repo README file to install the `repo tool`.

NOTE: The repo tool which gets installed using apt-get command sometimes return errors, in such a case it is recommended to install repo using the `curl` method:

```
mkdir -p ~/.bin
PATH="${HOME}/.bin:${PATH}"
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/.bin/repo
chmod a+rx ~/.bin/repo
```

2.9.3 Syncing and building the source code

The N1SDP software stack supports two software distributions:

- A minimal BusyBox root filesystem.
- Ubuntu server.

The instructions below provide the necessary steps to sync and build these distributions.

Create a new folder that will be your workspace and will henceforth be referred to as <n1sdp_workspace> in these instructions:

```
mkdir <n1sdp_workspace>
cd <n1sdp_workspace>
export N1SDP_RELEASE=refs/tags/N1SDP-2021.10.12
```

NOTE: Sometimes new features and additional bug fixes will be made available in the git repositories and will not yet have been tagged as part of a release. To pickup these latest changes you can drop the `-b <release tag>` option from the `repo init` commands below. However, please be aware that such untagged changes have not yet been formally verified and should be considered unstable until they are tagged in an official release.

Building

Firmware only

Build the firmware files.

```
repo init \
    -u https://git.gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest.git \
    -b ${N1SDP_RELEASE} \
    -m pinned-n1sdp.xml \
    -g bsp \
    --depth=6
repo sync
./build-scripts/build-all.sh -f none
```

Ubuntu Distribution

N1SDP provides support for installation and boot of standard Ubuntu 18.04 distribution. The distribution is installed on a disk and since the installed image is persistent it can be used for multiple boots.

Build the firmware and Ubuntu distribtution:

```
repo init \
    -u https://git.gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest.git \
    -b ${N1SDP_RELEASE} \
    -m pinned-n1sdp.xml \
    -g ubuntu \
    --depth=6
repo sync
./build-scripts/build-all.sh -f ubuntu
```

The bootable image will be available at the location <n1sdp_workspace>/output/n1sdp/ubuntu.img>

Minimal BusyBox

Build the firmware and BusyBox based root filesystem:

```
repo init \
    -u https://git.gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest.git \
    -b ${N1SDP_RELEASE} \
    -m pinned-n1sdp.xml \
    -g busybox \
    --depth=6
repo sync
./build-scripts/build-all.sh -f busybox
```

The bootable image will be available at the location <n1sdp_workspace>/output/n1sdp/busybox.img>

NOTE: To sync the source code with the complete history of components, drop the `depth` option from the `repo init` command for any of the distributions/filesystem mentioned above.

Check dependencies

To ensure that all the required dependent packages were installed, run:

```
./build-scripts/check_dep.sh

"no missing dependencies detected" should get displayed.
```

Fetch Tools

The tools required to build the software components are fetched using build-scripts/fetch-tools.sh, it is executed as part of build-all.sh.

The script can also be run separately to fetch the tools as shown :

```
./build-scripts/fetch-tools.sh -f none
```

Prebuilt board firmware

The board firmware prebuilt binaries are made available as part of the release in the directory `bsp/n1sdp-board-firmware/`, these binaries can be copied to the onboard SD card to boot the platform to UEFI EDK2 shell console.

There are two methods for updating the microSD card with the firmware binaries:

1. The microSD card from the N1SDP can be removed from N1SDP and can be mounted on a host machine using a card reader.
2. The USB debug cable when connected to host machine will show the microSD partition on host machine which can be mounted.

```
$> sudo mount /dev/sdx1 /mnt
$> rm -rf /mnt/*
$> sudo cp -a bsp/n1sdp-board-firmware/* /mnt/
$> sudo umount /mnt
```

NOTE: replace `sdx1` with the device and partition of the SD card. Option (2) above is typically preferred, as removing the microSD card requires physical access to the motherboard inside the N1SDP's tower case.

NOTE:

Please ensure to use the recommended PMIC binary. Refer to page potential-damage for more info. If a PMIC binary mismatch is detected, a warning message is printed in the MCC console recommending the user to switch to appropriate PMIC image. On MCC recommendation *ONLY*, please update the `MB/HBI0316A/io_v123f.txt` file on the microSD using the below commands.

Example command to switch to 300k_8c2.bin from the host PC

```
$> sudo mount /dev/sdx1 /mnt
$> sudo sed -i '/^MBPMIC: pms_0V85.bin/s/^/;/g' /mnt/MB/HBI0316A/io_v123f.txt
$> sudo sed -i '/^;MBPMIC: 300k_8c2.bin/s/^;//g' /mnt/MB/HBI0316A/io_v123f.txt
$> sudo umount /mnt
```

2.9.4 Software Components

Firmware

Each firmware component has a separate script to build that component. After the firmware components have been built they must be packaged using `build-firmware-image.sh`.

To make customizations to the firmware

```
# first make sure to build all the firmware components
<n1sdp_workspace>/build-scripts/build-all.sh -f none

# modify the firmware component

# build the component
<n1sdp_workspace>/build-scripts/build-[COMPONENT].sh -f none

# repackage the firmware files
<n1sdp_workspace>/build-scripts/build-firmware-image.sh -f none
```

As an alternative the one-liner `<n1sdp_workspace>/build-scripts/build-all.sh -f none` will trigger the build of all firmware scripts.

NOTE: The valid firmware [COMPONENT] names are :

```
"arm-tf" : To build Trusted Firmware-A
"scp"    : To build SCP and MCP ROM and RAM firmware binaries
"uefi"   : To build EDK2 for N1SDP
```

Firmware image creation

Packages the firmware files in a format suitable for the N1SDP.

Build script	<n1sdp_workspace>/build-scripts/build-firmware-image.sh
Output	<n1sdp_workspace>/output/n1sdp/firmware/scp_rom.bin <n1sdp_workspace>/output/n1sdp/firmware/scp_fw.bin <n1sdp_workspace>/output/n1sdp/firmware/mcp_rom.bin <n1sdp_workspace>/output/n1sdp/firmware/mcp_fw.bin <n1sdp_workspace>/output/n1sdp/firmware/uefi.bin <n1sdp_workspace>/output/n1sdp/firmware/n1sdp-board-firmware_primary.tar.gz <n1sdp_workspace>/output/n1sdp/firmware/n1sdp-board-firmware_secondary.tar.gz

Trusted Firmware-A

Based on Trusted Firmware-A.

Build script	<n1sdp_workspace>/build-scripts/build-arm-tf.sh
Checkout path	<n1sdp_workspace>/bsp/arm-tf
Output	<n1sdp_workspace>/output/n1sdp/intermediates/bl31.bin

System Control Processor (SCP)

Based on SCP Firmware.

Build script	<n1sdp_workspace>/build-scripts/build-scp.sh
Checkout path	<n1sdp_workspace>/bsp/scp
Output	<n1sdp_workspace>/output/n1sdp/intermediates/scp-ram.bin <n1sdp_workspace>/output/n1sdp/intermediates/scp_rom.bin <n1sdp_workspace>/output/n1sdp/intermediates/mcp-ram.bin <n1sdp_workspace>/output/n1sdp/intermediates/mcp_rom.bin

UEFI EDK2

Based on UEFI EDK2.

Build script	<n1sdp_workspace>/build-scripts/build-uefi.sh
Checkout path	<n1sdp_workspace>/bsp/uefi/edk2
Output	<n1sdp_workspace>/output/n1sdp/intermediates/uefi.bin

File-system

Each file-system component has a separate script to build that component. Some build scripts depend on the value of `-f <FILESYSTEM>`. After the file-system components have been built they must be packaged into a disk image using `build-disk-image.sh`.

```
# first make sure that all the components have been built at least once
<n1sdp_workspace>/build-scripts/build-all.sh -f <FILESYSTEM>

# modify the component

# build the component
<n1sdp_workspace>/build-scripts/build-[COMPONENT].sh -f <FILESYSTEM>

# repackage the disk image
<n1sdp_workspace>/build-scripts/build-disk-image.sh -f <FILESYSTEM>
```

NOTE:

The valid filesystem [COMPONENT] names are :

```
"grub" : To build the GRUB utilities
"linux": To build the Linux kernel image
"perf" : To build the perf tool
```

The valid [FILESYSTEM] names are :

```
"busybox" : To build for BusyBox filesystem
"ubuntu"  : To build for Ubuntu filesystem distribution
```

Disk image creation

Create a disk image for `-f <FILESYSTEM>`. FILESYSTEM is either `busybox` or `ubuntu`.

Build script	<n1sdp_workspace>/build-scripts/build-disk-image.sh
Output	<n1sdp_workspace>/output/n1sdp/<FILESYSTEM>.img <n1sdp_workspace>/output/n1sdp/intermediates/<FILESYSTEM

GRUB

Based on *grub*.

Build script	<n1sdp_workspace>/build-scripts/build-grub.sh
Checkout path	<n1sdp_workspace>/grub
Output	<n1sdp_workspace>/output/n1sdp/intermediates/grub/output/grubaa64.efi

Linux

Based on Linux 5.10.61 for N1SDP.

Build script	<n1sdp_workspace>/build-scripts/build-linux.sh
Checkout path	<n1sdp_workspace>/linux
Output	<n1sdp_workspace>/output/n1sdp/intermediates/kernel_Image_<FILESYSTEM>

BusyBox

Build a minimal root filesystem based on BusyBox

Build script	<n1sdp_workspace>/build-scripts/build-busybox.sh
Checkout path	<n1sdp_workspace>/busybox
Output	<n1sdp_workspace>/output/n1sdp/intermediates/busybox.initramfs <n1sdp_workspace>/output/n1sdp/intermediates/bus

Ubuntu

Build Ubuntu server distribution based on Ubuntu 18.04

Build script	<n1sdp_workspace>/build-scripts/build-ubuntu.sh
Checkout path	<n1sdp_workspace>/tools/ubuntu_bionic
Output	<n1sdp_workspace>/output/n1sdp/intermediates/ubuntu.esp.img <n1sdp_workspace>/output/n1sdp/intermediates/ubunt

2.9.5 Running the software distribution on N1SDP

This section provides steps for:

- Setting up the N1SDP with the required board firmware
- Preparing a bootable disk
- Boot the supported software distributions (Minimal BusyBox or Ubuntu Server).

Setting up the N1SDP

After powering up or rebooting the board, any firmware images placed on the board's microSD will be flashed into either on-board QSPI flash or copied into the DDR3 memory via the IOFPGA.

Configure COM Ports

Connect a USB-B cable between your host PC and N1SDP's DBG USB port, then power ON the board. The DBG USB connection will enumerate as four virtual COM ports assigned to the following processing entities, in order

```
COM<n>   - Motherboard Configuration Controller (MCC)
COM<n+1> - Application Processor (AP)
COM<n+2> - System Control Processor (SCP)
COM<n+3> - Manageability Control Processor (MCP)
```

Please check Device Manager in Windows or `ls /dev/ttyUSB*` in Linux to identify <n>.

Use a serial port application such as *PuTTy* or *minicom* to connect to all virtual COM ports with the following settings:

```
115200 baud
8-bit word length
No parity
1 stop bit
No flow control
```

Note: Some serial port applications refer to this as "115200 8N1" or similar.

Before running the deliverables on N1SDP, ensure both BOOT CONF switches are in the OFF position, then issue the following command in the MCC console:

Cmd> USB_ON

This will mount the on-board microSD card as a USB Mass Storage Device on the host PC with the name "N1SDP".

Enter the following command on the MCC console window to ensure time is correctly set. This is required for the first distribution boot to succeed:

```
Cmd> debug
Debug> time
Debug> date
Debug> exit
```

2.9.6 Update firmware on microSD card

The board firmware files are located in <n1sdp_workspace/output/n1sdp/firmware/> after the firmware source build.

Single chip mode:

```
n1sdp-board-firmware_primary.tar.gz    : firmware to be copied to microSD of N1SDP board␣
↪in single chip mode.
```

Multi chip mode:

```
n1sdp-board-firmware_primary.tar.gz    : firmware to be copied to microSD of primary␣
↪board.
n1sdp-board-firmware_secondary.tar.gz  : firmware to be copied to microSD of secondary␣
↪board.
```

There are two methods for populating the microSD card:

1. The microSD card from the N1SDP can be removed from N1SDP and can be mounted on a host machine using a card reader,
2. The USB debug cable when connected to host machine will show the microSD partition on host machine which can be mounted.

Option (2) above is typically preferred, as removing the microSD card requires physical access to the motherboard inside the N1SDP's tower case.

The instructions to extract the board firmware package onto the microSD card is as shown below:

```
$> sudo mount /dev/sdx1 /mnt
$> sudo rm -rf /mnt/*
$> sudo tar --no-same-owner -xzf n1sdp-board-firmware_primary.tar.gz -C /mnt/
$> sudo umount /mnt
```

NOTE:

- Replace `sdx1` with the device and partition of the SD card.
- Follow the similar steps to install the firmware on secondary board with n1sdp-board-firmware_secondary.tar.gz for multi chip mode.

Firmware tarball package contains IOFPGA configuration files, SCP, TF-A, and UEFI binaries.

NOTE: Please ensure to use the recommended PMIC binary. Refer to page potential-damage for more info.

If a PMIC binary mismatch is detected, a warning message is printed in the MCC console recommending the user to switch to appropriate PMIC image. On MCC recommendation *ONLY*, please update the `MB/HBI0316A/io_v123f.txt` file on the microSD using the below commands.

Example command to switch to 300k_8c2.bin from the host PC

```
$> sudo mount /dev/sdx1 /mnt
$> sudo sed -i '/^MBPMIC: pms_0V85.bin/s/^/;/g' /mnt/MB/HBI0316A/io_v123f.txt
$> sudo sed -i '/^;MBPMIC: 300k_8c2.bin/s/^;//g' /mnt/MB/HBI0316A/io_v123f.txt
$> sudo umount /mnt
```

L3 Cache Enablement

By default, L3 cache is disabled for use. To enable/disable L3 cache support, follow these steps:

1. Run USB_ON command to mount the on-board microSD card on the host PC.
2. Open the file "MB/HBI0316A/io_v123f.txt".
3. **For user to enable/disable L3 cache support, edit the SCC BOOT_GPR1 register in the following manner.**

    - **To enable L3 cache, update the SOCCON with offset 0x1184 and set the value 0x00000001. The line should now**
      `SOCCON: 0x1184 0x00000001 ; SoC SCC BOOT_GPR1`
    - **To disable L3 cache, update the SOCCON with offset 0x1184 and set the value 0x00000000. The line should now**
      `SOCCON: 0x1184 0x00000000 ; SoC SCC BOOT_GPR1`

4. Save and close the file.

Boot Minimal BusyBox Image on N1SDP

Preparing a bootable disk with BusyBox root filesystem

A bootable disk (USB stick or SATA drive) can be prepared by flashing the image generated from the source build. The image will be available at the location <n1sdp_workspace>/output/n1sdp/busybox.img>

This is a bootable GRUB image comprising Linux kernel and BusyBox binaries. The partitioning and packaging is performed during the build phase.

Use the following commands to prepare the GRUB image on a USB stick or SATA drive:

```
$ lsblk
$ sudo dd if=busybox.img of=/dev/sdx conv=fsync bs=1M
$ sync
```

Note: Replace `/dev/sdx` with the handle corresponding to your USB stick or SATA drive, as identified by the `lsblk` command.

Booting the board with BusyBox image

Insert the bootable disk created earlier. Shutdown and reboot the board by issuing the following commands to the MCC console:

```
Cmd> SHUTDOWN
Cmd> REBOOT
```

Enter the UEFI menu by pressing Esc on the AP console as the EDK2 logs start appearing; from here, enter the UEFI Boot Manager menu and then select the disk.

By default the Linux kernel will boot with ACPI configurations:

```
* BusyBox N1SDP (ACPI)
  BusyBox N1SDP SINGLE CHIP (Device Tree)
  BusyBox N1SDP MULTI CHIP (Device Tree)
```

The system will boot into a minimal BusyBox Linux image environment.

Boot Ubuntu on N1SDP

Preparing a bootable Ubuntu disk

A bootable disk (USB stick or SATA drive) can be prepared by formatting it with the distribution image created during source build. The image will be available at the location <n1sdp_workspace/output/n1sdp/ubuntu.img>.

This is a bootable GRUB image comprising Linux kernel and an Ubuntu Server 18.04 file system. The partitioning and packaging is performed during the build.

Use the following commands to burn the GRUB image to a USB stick or SATA drive:

```
$ lsblk
$ sudo dd if=ubuntu.img of=/dev/sdX bs=1M
$ sync
```

Note: Replace `/dev/sdX` with the handle corresponding to your USB stick or SATA drive, as identified by the `lsblk` command.

Booting the board with Ubuntu image

Insert the bootable disk created earlier and connect the ethernet cable to a working internet connection. This is *REQUIRED* on first boot in order to successfully download and install necessary Ubuntu packages. Installation will fail if an internet connection is not available.

NOTE: It is also observed that the installation may fail if more than one storage device is present on N1SDP, the error log is as shown below:

```
Booting `Install Ubuntu on N1SDP Platform'
error: disk `hd1,msdos2' not found.
error: you need to load the kernel first.

Press any key to continue...

Failed to boot both default and fallback entries.

Press any key to continue...
```

Therefore, it is always recommended to have only one storage device on N1SDP on which you want to install the Ubuntu software.

Shutdown and reboot the board by issuing the following commands to the MCC console:

```
Cmd> SHUTDOWN
Cmd> REBOOT
```

Enter the UEFI menu by pressing `Esc` on the AP console as the EDK2 logs start appearing; from here, enter the `UEFI Boot Manager menu` and then select the burned disk.

Ubuntu 18.04 will boot in two stages; the first boot is an installation pass, after which a second boot is required to actually enter the Ubuntu Server environment.

To reboot the board after the first boot installation pass has completed, from MCC console:

```
Cmd> REBOOT
```

The system will boot into a minimal Ubuntu 18.04 environment.

Login as user `root` with password *root*, and install any desired packages from the console:

```
# apt-get install <package-name>
```

Building Kernel Modules Natively

Native building of kernel modules typically requires kernel headers to be installed on the platform. However, a bug in deb-pkg currently causes host executables to be packed rather than the target executables.

This can be worked around by building and installing the kernel natively on the platform.

Boot the N1SDP board with Ubuntu filesystem and login as root.

```
apt-get install -y git build-essential bc bison flex libssl-dev
git clone -b n1sdp https://git.gitlab.arm.com/arm-reference-solutions/linux.git
cd kernel-release/
mkdir out
cp -v /boot/config-5.10.61+  out/.config
make O=out -j4
```

(continues on next page)

(continued from previous page)

```
make O=out modules_install
make O=out install
update-grub
sync
```

Reboot the board and when Grub menu appears, select the Advanced Boot Options -> 5.10.61 kernel for booting.

2.10 Guidelines to Vulnerability reporting

- The Neoverse N1 System Development Platform (N1SDP) stack is a collection of open source software repositories. If you think you have found a security vulnerability in a specific open source project which is part of the software stack, it is recommended to follow the vulnerability reporting guidelines specified by the respective project.
- If you think you have found a security vulnerability as part of the Neoverse N1 System Development Platform (N1SDP) software stack and it does not fall into any specific open source project, then please report by email at arm-security@arm.com specifying the project name as "Neoverse N1 System Development Platform (N1SDP) Software". More details can be found at Arm Developer website.

CHAPTER

THREE

ARM CORSTONE1000

3.1 1. User Guide

Contents

3.1.1 1.1. Notice

The Corstone-1000 software stack uses the Yocto project to build a tiny Linux distribution suitable for the Corstone-1000 platform. The yocto project relies on the Bitbake tool as its build tool. Please see Yocto mega manual for more information.

3.1.2 1.2. Prerequisites

These instructions assume your host PC is running Ubuntu Linux 18.04 or 20.04 LTS, with at least 32GB of free disk space and 16GB of RAM as minimum requirement. The following instructions expect that you are using a bash shell.

The following prerequisites must be available on the host system. To resolve these dependencies, run:

```
sudo apt-get update
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
 build-essential chrpath socat cpio python3 python3-pip python3-pexpect \
 xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev␣
↪\
 pylint3 xterm zstd liblz4-tool picocom
sudo apt-get upgrade libstdc++6
```

3.1.3 1.3. Provided components

Within the Yocto project, each component included in the Corstone-1000 software stack is specified as a bitbake recipe. The recipes specific to the Corstone-1000 BSP are located at: `<_workspace>/meta-arm/meta-arm-bsp/`. The Yocto machine config files for the Corstone-1000 FVP and FPGA are:

- `<_workspace>/meta-arm/meta-arm-bsp/conf/machine/include/corstone1000.inc`
- `<_workspace>/meta-arm/meta-arm-bsp/conf/machine/corstone1000-fvp.conf`
- `<_workspace>/meta-arm/meta-arm-bsp/conf/machine/corstone1000-mps3.conf`

1.3.1. Software for Host

1.3.1.1. Trusted Firmware-A

Based on Trusted Firmware-A

bbappend	<_workspace>/meta-arm/meta-arm-bsp/recipes-bsp/trusted-firmware-a/trusted-firmware-a_2.5.bbappend
Recipe	<_workspace>/meta-arm/meta-arm/recipes-bsp/trusted-firmware-a/trusted-firmware-a_2.5.bb

1.3.1.2. OP-TEE

Based on OP-TEE

bbappend	<_workspace>/meta-arm/meta-arm-bsp/recipes-security/optee/optee-os_3.14.0.bbappend
Recipe	<_workspace>/meta-arm/meta-arm/recipes-security/optee/optee-os_3.14.0.bb

1.3.1.3. U-Boot

Based on U-Boot

bbappend	<_workspace>/meta-arm/meta-arm/recipes-bsp/u-boot/u-boot_%.bbappend
Recipe	<_workspace>/poky/meta/recipes-bsp/u-boot/u-boot_2021.07.bb

1.3.1.4. Linux

The distro is based on the poky-tiny distribution which is a Linux distribution stripped down to a minimal configuration.

The provided distribution is based on busybox and built using muslibc. The recipe responsible for building a tiny version of linux is listed below.

bbappend	<_workspace>/meta-arm/meta-arm-bsp/recipes-kernel/linux/linux-yocto_%.bbappend
Recipe	<_workspace>/poky/meta/recipes-kernel/linux/linux-yocto_5.10.bb
defconfig	<_workspace>/meta-arm/meta-arm-bsp/recipes-kernel/linux/files/corstone1000/defconfig

1.3.2. Software for Boot Processor (a.k.a Secure Enclave)

Based on Trusted Firmware-M

bbappend	<_workspace>/meta-arm/meta-arm-bsp/recipes-bsp/trusted-firmware-m/trusted-firmware-m_%.bbappend
Recipe	<_workspace>/meta-arm/meta-arm/recipes-bsp/trusted-firmware-m/trusted-firmware-m_1.4.0.bb

The TF-M version has been bumped to 1.5, however the trusted-firmware-m_1.4.0.bb recipe file is used.

3.1.4 1.4. Building the software stack

Create a new folder that will be your workspace and will henceforth be referred to as <_workspace> in these instructions. To create the folder, run:

```
mkdir <_workspace>
cd <_workspace>
```

Corstone-1000 is a Bitbake based Yocto distro which uses kas and bitbake commands to build the stack. To install kas tool, run:

```
sudo pip3 install kas
```

In the top directory of the workspace <_workspace>, run:

```
git clone https://git.yoctoproject.org/git/meta-arm -b CORSTONE1000-2022.04.07
```

To build corstone1000 image for MPS3 FPGA, run:

```
kas shell meta-arm/kas/corstone1000-mps3.yml
bitbake corstone1000-image
```

Alternatively, to build corstone1000 image for FVP, run:

```
kas shell meta-arm/kas/corstone1000-fvp.yml
bitbake corstone1000-image
```

The initial clean build will be lengthy, given that all host utilities are to be built as well as the target images. This includes host executables (python, cmake, etc.) and the required toolchain(s).

Once the build is successful, all output binaries will be placed in the following folders:

- <_workspace>/build/tmp/deploy/images/corstone1000-fvp/ folder for FVP build;
- <_workspace>/build/tmp/deploy/images/corstone1000-mps3/ folder for FPGA build.

Everything apart from the ROM firmware is bundled into a single binary, the corstone1000-image-corstone1000-{mps3,fvp}.wic.nopt file. The ROM firmware is the bl1.bin file.

The output binaries used by FVP are the following:

- The ROM firmware: <_workspace>/build/tmp/deploy/images/corstone1000-fvp/bl1.bin
- The flash image: <_workspace>/build/tmp/deploy/images/corstone1000-fvp/corstone1000-image-corstone1000-fvp.wic.nopt

The output binaries used by FPGA are the following:

- The ROM firmware: <_workspace>/build/tmp/deploy/images/corstone1000-mps3/bl1.bin
- The flash image: <_workspace>/build/tmp/deploy/images/corstone1000-mps3/corstone1000-image-corstone1000-mps3.wic.nopt

3.1.5 1.5. Flash the firmware image on FPGA

The user should download the FPGA bit file image from this link and under the section Arm® Corstone™-1000 for MPS3.

The directory structure of the FPGA bundle is shown below.

```
Boardfiles
├── MB
│   ├── BRD_LOG.TXT
│   ├── HBI0309B
│   │   ├── AN550
│   │   │   ├── AN550_v1.bit
│   │   │   ├── an550_v1.txt
│   │   │   └── images.txt
│   │   ├── board.txt
│   │   └── mbb_v210.ebf
│   └── HBI0309C
│       ├── AN550
```

(continues on next page)

(continued from previous page)

```
│       │   ├── AN550_v1.bit
│       │   ├── an550_v1.txt
│       │   └── images.txt
│       ├── board.txt
│       └── mbb_v210.ebf
├── SOFTWARE
│   ├── ES0.bin
│   ├── SE.bin
│   └── an550_st.axf
└── config.txt
```

Depending upon the MPS3 board version (printed on the MPS3 board) you should update the images.txt file (in corresponding HBI0309x folder) so that the file points to the images under SOFTWARE directory.

Here is an example

```
;*********************************************
;       Preload port mapping                  *
;*********************************************
;  PORT 0 & ADDRESS: 0x00_0000_0000 QSPI Flash (XNVM) (32MB)
;  PORT 0 & ADDRESS: 0x00_8000_0000 OCVM (DDR4 2GB)
;  PORT 1        Secure Enclave (M0+) ROM (64KB)
;  PORT 2        External System 0 (M3) Code RAM (256KB)
;  PORT 3        Secure Enclave OTP memory (8KB)
;  PORT 4        CVM (4MB)
;*********************************************

[IMAGES]
TOTALIMAGES: 2      ;Number of Images (Max: 32)

IMAGE0PORT: 1
IMAGE0ADDRESS: 0x00_0000_0000
IMAGE0UPDATE: RAM
IMAGE0FILE: \SOFTWARE\bl1.bin

IMAGE1PORT: 0
IMAGE1ADDRESS: 0x00_0001_0000
IMAGE1UPDATE: AUTOQSPI
IMAGE1FILE: \SOFTWARE\cs1000.bin
```

OUTPUT_DIR = <_workspace>/build/tmp/deploy/images/corstone1000-mps3

1. Copy `bl1.bin` from OUTPUT_DIR directory to SOFTWARE directory of the FPGA bundle.
2. Copy `corstone1000-image-corstone1000-mps3.wic.nopt` from OUTPUT_DIR directory to SOFTWARE directory of the FPGA bundle and rename the wic image to `cs1000.bin`.

NOTE: Renaming of the images are required because MCC firmware has limitation of 8 characters before .(dot) and 3 characters after .(dot).

Now, copy the entire folder to board's SDCard and reboot the board.

3.1.6 1.6. Running the software on FPGA

On the host machine, open 3 minicom sessions. In case of Linux machine it will be ttyUSB0, ttyUSB1, ttyUSB2 and it might be different on Window machine.

- ttyUSB0 for MCC, OP-TEE and Secure Partition
- ttyUSB1 for Boot Processor (Cortex-M0+)
- ttyUSB2 for Host Processor (Cortex-A35)

Run following commands to open minicom sessions on Linux:

```
sudo picocom -b 115200 /dev/ttyUSB0  # in one terminal
sudo picocom -b 115200 /dev/ttyUSB1  # in another terminal
sudo picocom -b 115200 /dev/ttyUSB2  # in another terminal.
```

Once the system boot is completed, you should see console logs on the minicom sessions. Once the HOST(Cortex-A35) is booted completely, user can login to the shell using **"root"** login.

3.1.7 1.7. Running the software on FVP

An FVP (Fixed Virtual Platform) of the Corstone-1000 platform must be available to execute the included run script.

The fixed virtual platform (FVP) version 11.17_23 can be downloaded from the Arm Ecosystem FVPs page. On this page, navigate to "Corstone IoT FVPs" section to download the Corstone1000 platform FVP installer. Follow the instructions of the installer and setup the FVP.

The run-scripts structure is as below:

```
run-scripts
|— corstone1000
    └— run_model.sh
|— ...
```

Ensure that the FVP has its dependencies met by executing the FVP from its installation folder: `./<Corstone-1000 Model Binary>`.

All dependencies are met if the FVP launches without any errors, presenting a graphical interface showing information about the current state of the FVP.

The `run_model.sh` script in "<_workspace>/run-scripts/corstone1000/" folder runs the FVP with the previously built images as arguments. Execute the `run_model.sh` script:

```
./run_model.sh ${FVP installation path/<Corstone-1000 FVP Binary>} -D ""
```

When the script is executed, three terminal instances will be launched, one for the boot processor (aka Secure Enclave) processing element and two for the Host processing element. Once the FVP is executing, the Boot Processor will start to boot, wherein the relevant memory contents of the .wic file are copied to their respective memory locations within the model, enforce firewall policies on memories and peripherals and then, bring the host out of reset.

The host will boot trusted-firmware-a, OP-TEE, U-Boot and then Linux, and present a login prompt (FVP host_terminal_0):

```
corstone1000-fvp login:
```

Login using the username root.

3.1.8 1.8. Running test applications

NOTE: Running the SystemReady-IR tests described below requires the user to work with USB sticks. In our testing, not all USB stick models work well with MPS3 FPGA. Here are the USB sticks models that are stable in our test environment.

- HP V165W 8 GB USB Flash Drive
- SanDisk Ultra 32GB Dual USB Flash Drive USB M3.0
- SanDisk Ultra 16GB Dual USB Flash Drive USB M3.0

NOTE: Before running each of the tests in this chapter, the user should follow the steps described in following section "Clean Secure Flash Before Testing" to erase the SecureEnclave flash cleanly and prepare a clean board environment for the testing.

1.8.1. Clean Secure Flash Before Testing (applicable to FPGA only)

To prepare a clean board environment with clean secure flash for the testing, the user should prepare an image that erases the secure flash cleanly during boot. Run following commands to build such image.

```
cd <_workspace>
git clone https://git.yoctoproject.org/git/meta-arm -b CORSTONE1000-2022.02.18
git clone https://git.gitlab.arm.com/arm-reference-solutions/systemready-patch.git
cp -f systemready-patch/embedded-a/corstone1000/erase_flash/0001-arm-bsp-trusted-
↪firmware-m-corstone1000-Clean-Secure.patch meta-arm
cd meta-arm
git apply 0001-arm-bsp-trusted-firmware-m-corstone1000-Clean-Secure.patch
cd ..
kas shell meta-arm/kas/corstone1000-mps3.yml
bitbake corstone1000-image
```

Replace the bl1.bin and cs1000.bin files on the SD card with following files:

- The ROM firmware: <_workspace>/build/tmp/deploy/images/corstone1000-mps3/bl1.bin
- The flash image: <_workspace>/build/tmp/deploy/images/corstone1000-mps3/corstone1000-image-corstone1000-mps3.wic.nopt

Now reboot the board. This step erases the Corstone1000 SecureEnclave flash completely, the user should expect following message from TF-M log:

```
!!!SECURE FLASH HAS BEEN CLEANED!!!
NOW YOU CAN FLASH THE ACTUAL CORSTONE1000 IMAGE
PLEASE REMOVE THE LATEST ERASE SECURE FLASH PATCH AND BUILD THE IMAGE AGAIN
```

Then the user should follow "Building the software stack" to build a clean software stack and flash the FPGA as normal. And continue the testing.

1.8.2. Run SystemReady-IR ACS tests

ACS image contains two partitions. BOOT partition and RESULTS partition. Following packages are under BOOT partition:

- SCT
- FWTS
- BSA uefi
- BSA linux
- grub
- uefi manual capsule application

RESULTS partition is used to store the test results. PLEASE MAKE SURE THAT THE RESULTS PARTITION IS EMPTY BEFORE YOU START THE TESTING. OTHERWISE THE TEST RESULTS WILL NOT BE CONSISTENT

1.8.2.1. FPGA instructions for ACS image

This section describes how the user can build and run Architecture Compliance Suite (ACS) tests on Corstone1000.

First, the user should download the Arm SystemReady ACS repository. This repository contains the infrastructure to build the Architecture Compliance Suite (ACS) and the bootable prebuilt images to be used for the certifications of SystemReady-IR. To download the repository, run command:

```
cd <_workspace>
git clone https://github.com/ARM-software/arm-systemready.git -b v21.09_REL1.0
```

Once the repository is successfully downloaded, the prebuilt ACS live image can be found in:

- `<_workspace>/arm-systemready/IR/prebuilt_images/v21.07_0.9_BETA/ir_acs_live_image.img.xz`

NOTE: This prebuilt ACS image includes v5.13 kernel, which doesn't provide USB driver support for Corstone1000. The ACS image with newer kernel version and with full USB support for Corstone1000 will be available in the next SystemReady release in this repository.

Then, the user should prepare a USB stick with ACS image. In the given example here, we assume the USB device is `/dev/sdb` (the user should use `lsblk` command to confirm). Be cautious here and don't confuse your host PC's own hard drive with the USB drive. Run the following commands to prepare the ACS image in USB stick:

```
cd <_workspace>/arm-systemready/IR/scripts/output/
unxz ir_acs_live_image.img.xz
sudo dd if=ir_acs_live_image.img of=/dev/sdb iflag=direct oflag=direct bs=1M␣
↪status=progress; sync
```

Once the USB stick with ACS image is prepared, the user should make sure that ensure that only the USB stick with the ACS image is connected to the board, and then boot the board.

1.8.2.2. FVP instructions for ACS image and run

Download acs image from:

- https://gitlab.arm.com/systemready/acs/arm-systemready/-/tree/linux-5.17-rc7/IR/prebuilt_images/v22.04_1.0-Linux-v5.17-rc7

Use the below command to run the FVP with acs image support in the SD card.

```
unxz ${<path-to-img>/ir_acs_live_image.img.xz}
./run_model.sh ${FVP installation path/<Corstone-1000 FVP Binary>} -D ${<path-to-img>/ir_
↪acs_live_image.img}
```

The test results can be fetched following below commands:

```
sudo mkdir /mnt/test
sudo mount -o rw,offset=<offset_2nd_partition> <path-to-img>/ir_acs_live_image.img /mnt/
↪test/
fdisk -lu <path-to-img>/ir_acs_live_image.img
->  Device                                                  Start     End Sectors ␣
↪Size Type
    /home/emeara01/Downloads/ir_acs_live_image_modified.img1    2048 1050622 1048575 ␣
↪512M Microsoft basic data
    /home/emeara01/Downloads/ir_acs_live_image_modified.img2 1050624 1153022  102399  ␣
↪50M Microsoft basic data

->   <offset_2nd_partition> = 1050624 * 512 (sector size) = 537919488
```

The FVP will reset multiple times during the test, and it might take up to 1 day to finish the test. At the end of test, the FVP host terminal will halt showing a shell prompt. Once test is finished, the FVP can be stoped, and result can be copied following above instructions.

1.8.2.3. Common to FVP and FPGA

U-Boot should be able to boot the grub bootloader from the 1st partition and if grub is not interrupted, tests are executed automatically in the following sequence:

- SCT
- UEFI BSA
- FWTS
- BSA Linux

The results can be fetched from the `acs_results` partition of the USB stick (FPGA) / SD Card (FVP).

1.8.3. Manual capsule update test

The following steps describe running manual capsule update with the `direct` method.

Check the "Run SystemReady-IR ACS tests" section above to download and unpack the acs image file

- `ir_acs_live_image.img.xz`

Download edk2 and generate capsule file:

```
git clone https://github.com/tianocore/edk2.git
edk2/BaseTools/BinWrappers/PosixLike/GenerateCapsule -e -o \
  cs1k_cap --fw-version 1 --lsv 0 --guid \
  e2bb9c06-70e9-4b14-97a3-5a7913176e3f --verbose --update-image-index \
  0 --verbose <binary_file>
```

The <binary_file> here should be a corstone1000-image-corstone1000-fvp.wic.nopt image for FVP and corstone1000-image-corstone1000-mps3.wic.nopt for FPGA. And this input binary file (capsule) should be less than 15 MB.

Based on the user's requirement, the user can change the firmware version number given to `--fw-version` option (the version number needs to be >= 1).

1.8.3.1. Capsule Copy instructions for FPGA

The user should prepare a USB stick as explained in ACS image section (see above). Place the generated `cs1k_cap` file in the root directory of the boot partition in the USB stick. Note: As we are running the direct method, the `cs1k_cap` file should not be under the EFI/UpdateCapsule directory as this may or may not trigger the on disk method.

1.8.3.2. Capsule Copy instructions for FVP

Run below commands to copy capsule into the image file and run FVP software.

```
sudo mkdir /mnt/test
sudo mount -o rw,offset=<offset_1st_partition> <path-to-img>/ir_acs_live_image.img /mnt/
↪test/
sudo cp cs1k_cap /mnt/test/
sudo umount /mnt/test
exit
./run_model.sh ${FVP installation path/<Corstone-1000 FVP Binary>} -D ${<path-to-img>/ir_
↪acs_live_image.img}
```

Size of first partition in the image file is calculated in the following way. The data is just an example and might vary with different ir_acs_live_image.img files.

```
fdisk -lu <path-to-img>/ir_acs_live_image.img
->  Device                                                  Start     End Sectors ␣
↪Size Type
    /home/emeara01/Downloads/ir_acs_live_image_modified.img1    2048 1050622 1048575 ␣
↪512M Microsoft basic data
    /home/emeara01/Downloads/ir_acs_live_image_modified.img2 1050624 1153022  102399  ␣
↪50M Microsoft basic data

->  <offset_1st_partition> = 2048 * 512 (sector size) = 1048576
```

1.8.3.3. Common to FVP and FPGA

Reach u-boot then interrupt shell to reach EFI shell. Use below command at EFI shell.

```
FS0:
EFI/BOOT/app/CapsuleApp.efi cs1k_cap
```

For this test, the user can provide two capsules for testing: a positive test case capsule which boots the board correctly, and a negative test case with an incorrect capsule which fails to boot the host software.

In the positive case scenario, the user should see following log in TF-M log, indicating the new capsule image is successfully applied, and the board boots correctly.

```
...
SysTick_Handler: counted = 10, expiring on = 360
SysTick_Handler: counted = 20, expiring on = 360
SysTick_Handler: counted = 30, expiring on = 360
...
metadata_write: success: active = 1, previous = 0
accept_full_capsule: exit: fwu state is changed to regular
...
```

In the negative case scenario, the user should see appropriate logs in the secure enclave terminal. If capsule pass initial verification, but fails verifications performed during boot time, secure enclave will try new images predetermined number of times (defined in the code), before reverting back to the previous good bank.

```
...
metadata_write: success: active = 0, previous = 1
fwu_select_previous: in regular state by choosing previous active bank
...
```

1.8.4. Linux distro install and boot (applicable to FPGA only)

To test Linux distro install and boot, the user should prepare two empty USB sticks.

Download one of following Linux distro images:

- Debian installer image: https://cdimage.debian.org/cdimage/weekly-builds/arm64/iso-dvd/
- OpenSUSE Tumbleweed installer image: http://download.opensuse.org/ports/aarch64/tumbleweed/iso/ - The user should look for a DVD Snapshot like openSUSE-Tumbleweed-DVD-aarch64-Snapshot20211125-Media.iso

Once the .iso file is downloaded, the .iso file needs to be flashed to your USB drive.

In the given example here, we assume the USB device is `/dev/sdb` (the user should use *lsblk* command to confirm). Be cautious here and don't confuse your host PC's own hard drive with the USB drive. Then copy the contents of an iso file into the first USB stick, run:

```
sudo dd if=</path/to/iso_file> of=/dev/sdb iflag=direct oflag=direct status=progress␣
↪bs=1M; sync;
```

Boot the MSP3 board with the first USB stick connected. Open following minicom sessions:

```
sudo picocom -b 115200 /dev/ttyUSB0  # in one terminal
sudo picocom -b 115200 /dev/ttyUSB2  # in another terminal.
```

Press <Ctrl+x>.

Now plug in the second USB stick, the distro installation process will start.

NOTE: Due to the performance limitation of Corstone1000 MPS3 FPGA, the distro installation process can take up to 24 hours to complete.

Once installation is complete, unplug the first USB stick and reboot the board. After successfully installing and booting the Linux distro, the user should see a login prompt:

```
debian login:
```

Login with the username root.

1.8.5. Run psa-arch-test (applicable to both FPGA and FVP)

When running psa-arch-test on MPS3 FPGA, the user should make sure there is no USB stick connected to the board. Power on the board and boot the board to Linux. Then, the user should follow the steps below to run the psa_arch_tests.

When running psa-arch-test on Corstone1000 FVP, the user should follow the instructions in *Running the software on FVP* section to boot Linux in FVP host_terminal_0, and login using the username `root`.

As a reference for the user's test results, the psa-arch-test report for Corstone1000 software (CORSTONE1000-2022.02.18) can be found in here.

First, create a file containing SE_PROXY_SP UUID. Run:

```
echo 46bb39d1-b4d9-45b5-88ff-040027dab249 > sp_uuid_list.txt
```

Then, load FFA driver module into Linux kernel. Run:

```
load_ffa_debugfs.sh .
```

Then, check whether the FFA driver loaded correctly by using the following command:

```
cat /proc/modules | grep arm_ffa_user
```

The output should be:

```
arm_ffa_user 16384 - - Live 0xffffffc0084b0000 (O)
```

Now, run the PSA arch tests with following commands. The user should run the tests in following order:

```
psa-iat-api-test
psa-crypto-api-test
psa-its-api-test
psa-ps-api-test
```

1.8.6. Linux distro: OpenSUSE Raw image installation (FVP Only)

Steps to download openSUSE Tumbleweed raw image:

- Go to: http://download.opensuse.org/ports/aarch64/tumbleweed/appliances/
- The user should look for a Tumbleweed-ARM-JeOS-efi.aarch64-* Snapshot, for example, `openSUSE-Tumbleweed-ARM-JeOS-efi.aarch64-2022.03.18-Snapshot20220331.raw.xz`

Once the .raw.xz file is downloaded, the raw image file needs to be extracted:

```
unxz <file-name.raw.xz>
```

The above command will generate a file ending with extension .raw image. Now, use the following command to run FVP with raw image installation process.

```
./run_model.sh ${FVP installation path/<Corstone-1000 FVP Binary>} -D ${openSUSE raw␣
↪image file path}
```

After successfully installing and booting the Linux distro, the user should see a openSUSE login prompt.

```
localhost login:
```

Login with the username 'root' and password 'linux'.

3.1.9 1.9. Running the software on FVP on Windows

If the user needs to run the Corstone1000 software on FVP on Windows. The user should follow the build instructions in this document to build on Linux host PC, and copy the output binaries to the Windows PC where the FVP is located, and launch the FVP binary.

Contents

3.2 1. Release notes - 2022.04.04

3.2.1 1.1. Known Issues or Limitations

- FGPA support Linux distro install and boot through installer. However, FVP only support openSUSE raw image installation and boot.
- Due to the performance uplimit of MPS3 FPGA and FVP, some Linux distros like Fedora Rawhide cannot boot on Corstone1000 (i.e. user may experience timeouts or boot hang).
- Below SCT FAILURE is a known issues in the FVP: UEFI Compliant - Boot from network protocols must be implemented – FAILURE

3.2.2 1.2. Platform Support

- This software release is tested on Corstone1000 FPGA version AN550_v1
- This software release is tested on Corstone1000 Fast Model platform (FVP) version 11.17_23 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

3.3 2. Release notes - 2022.02.25

3.3.1 2.1. Known Issues or Limitations

- The following tests only work on Corstone1000 FPGA: ACS tests (SCT, FWTS, BSA), manual capsule update test, Linux distro install and boot.

3.3.2 2.2. Platform Support

- This software release is tested on Corstone1000 FPGA version AN550_v1
- This software release is tested on Corstone1000 Fast Model platform (FVP) version 11.17_23 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

3.4 3. Release notes - 2022.02.21

3.4.1 3.1. Known Issues or Limitations

- The following tests only work on Corstone1000 FPGA: ACS tests (SCT, FWTS, BSA), manual capsule update test, Linux distro install and boot, psa-arch-test.

3.4.2 3.2. Platform Support

- This software release is tested on Corstone1000 FPGA version AN550_v1
- This software release is tested on Corstone1000 Fast Model platform (FVP) version 11.16.21 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

3.5 4. Release notes - 2022.01.18

3.5.1 4.1. Known Issues or Limitations

- Before running each SystemReady-IR tests: ACS tests (SCT, FWTS, BSA), manual capsule update test, Linux distro install and boot, etc., the SecureEnclave flash must be cleaned. See user-guide “Clean Secure Flash Before Testing” section.

3.6 5. Release notes - 2021.12.15

3.6.1 5.1. Software Features

The following components are present in the release:

- Yocto version Honister
- Linux kernel version 5.10
- U-Boot 2021.07
- OP-TEE version 3.14
- Trusted Firmware-A 2.5
- Trusted Firmware-M 1.5
- OpenAMP 347397decaa43372fc4d00f965640ebde042966d
- Trusted Services a365a04f937b9b76ebb2e0eeade226f208cbc0d2

3.6.2 5.2. Platform Support

- This software release is tested on Corstone1000 FPGA version AN550_v1
- This software release is tested on Corstone1000 Fast Model platform (FVP) version 11.16.21 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

3.6.3 5.3. Known Issues or Limitations

- The following tests only work on Corstone1000 FPGA: ACS tests (SCT, FWTS, BSA), manual capsule update test, Linux distro install and boot, and psa-arch-tests.
- Only the manual capsule update from UEFI shell is supported on FPGA.
- Due to flash size limitation and to support A/B banks,the wic image provided by the user should be smaller than 15MB.
- The failures in PSA Arch Crypto Test are known limitations with crypto library. It requires further investigation. The user can refer to PSA Arch Crypto Test Failure Analysis In TF-M V1.5 Release for the reason for each failing test.

3.7 6. Release notes - 2021.10.29

3.7.1 6.1. Software Features

This initial release of Corstone-1000 supports booting Linux on the Cortex-A35 and TF-M/MCUBOOT in the Secure Enclave. The following components are present in the release:

- Linux kernel version 5.10
- U-Boot 2021.07
- OP-TEE version 3.14
- Trusted Firmware-A 2.5
- Trusted Firmware-M 1.4

3.7.2 6.2. Platform Support

- This Software release is tested on Corstone1000 Fast Model platform (FVP) version 11.16.21 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

3.7.3 6.3. Known Issues or Limitations

- No software support for external system(Cortex M3)
- No communication established between A35 and M0+
- Very basic functionality of booting Secure Enclave, Trusted Firmware-A , OP-TEE , u-boot and Linux are performed

3.7.4 6.4. Support

For support email: support-subsystem-iot@arm.com

3.8 Change Log

This document contains a summary of the new features, changes and fixes in each release of Corstone-1000 software stack.

3.8.1 Version 2022.04.04

Features added

- Linux distro openSUSE, raw image installation and boot in the FVP.
- SCT test support in FVP.
- Manual capsule update support in FVP.

3.8.2 Version 2022.02.25

Features added

- Building and running psa-arch-tests on Corstone1000 FVP
- Enabled smm-gateway partition in Trusted Service on Corstone1000 FVP
- Enabled MHU driver in Trusted Service on Corstone1000 FVP
- Enabled OpenAMP support in SE proxy SP on Corstone1000 FVP

3.8.3 Version 2022.02.21

Changes

- psa-arch-tests: recipe is dropped and merged into the secure-partitons recipe.
- psa-arch-tests: The tests are align with latest tfm version for psa-crypto-api suite.

3.8.4 Version 2022.01.18

Changes

- psa-arch-tests: change master to main for psa-arch-tests
- U-Boot: fix null pointer exception for get_image_info
- TF-M: fix capsule instability issue for corstone1000

3.8.5 Version 2022.01.07

Changes

- Corstone1000: fix SystemReady-IR ACS test (SCT, FWTS) failures.
- U-Boot: send bootcomplete event to secure enclave.
- U-Boot: support populating corstone1000 image_info to ESRT table.
- U-Boot: add ethernet device and enable configs to support bootfromnetwork SCT.

3.8.6 Version 2021.12.15

Features added

- Enabling Corstone1000 FPGA support on: - Linux 5.10 - OP-TEE 3.14 - Trusted Firmware-A 2.5 - Trusted Firmware-M 1.5
- Building and running psa-arch-tests
- Adding openamp support in SE proxy SP
- OP-TEE: adding smm-gateway partition
- U-Boot: introducing Arm FF-A and MM support

3.8.7 Version 2021.10.29

Features added

- Enabling Corstone1000 FVP support on: - Linux 5.10 - OP-TEE 3.14 - Trusted Firmware-A 2.5 - Trusted Firmware-M 1.4
- Linux kernel: enabling EFI, adding FF-A debugfs driver, integrating ARM_FFA_TRANSPORT.
- U-Boot: Extending EFI support
- python3-imgtool: adding recipe for Trusted-firmware-m
- python3-imgtool: adding the Yocto recipe used in signing host images (based on MCUBOOT format)

Changes

CHAPTER

FOUR

AEMFVP-A

4.1 Busybox boot on Armv-A Base AEM FVP platforms

Contents

4.1.1 Overview of Busybox boot

Busybox is a lightweight executable which packages lots of POSIX compliant UNIX utilities in a single file system. Busybox boot with Armv-A base design platform software stack demonstrates the integration of various software components on the software stack resulting in the ability to boot Linux Kernel on Armv-A Base AEM FVP.

Booting to Busybox is especially helpful when porting the software stack for new platforms which are derivative of Armv-A base design platform as this can be quickly executed to ensure that the various software components are properly integrated and verify the basic functionality of various software components.

This document describes how to build the Armv-A based design platform software stack and use it to boot up to Busybox on the Armv-A Base AEM FVP.

4.1.2 Download the platform software

To obtain the required sources for the platform, follow the steps listed on the user guide page. Ensure that the platform software is downloaded before proceeding with the steps listed below. Also, note the host machine requirements listed on that page which is essential to build and execute the platform software stack.

Skip this section if the required sources already have been downloaded.

4.1.3 Build the platform software

This section describes the procedure to build the following software components for Busybox boot.

- Trusted firmware (TF-A)
- U-Boot
- UEFI
- GRUB
- Linux
- Busybox

The disk image consists of two partitions. The first partition is an EFI system partition and contains GRUB and Kernel Image. The second partition is an ext3 partition which contains the Busybox based rootfs. Examples of how to use the build command for Busybox boot are listed below.

To build the software stack, the command to be used is

```
./build-scripts/aemfvp-a/build-test-busybox.sh -p <platform name> <command>
```

Supported command line options are listed below

- <platform name>
  - `aemfvp-a`
- <command>
  - Supported commands are
    - `clean`
    - `build`
    - `package`
    - `all` (all of the three above)

Note: On networks where the git port is blocked, the build procedure might not progress. Refer to the troubleshooting guide for possible ways to resolve this issue.

Examples of the build command are

- Command to clean, build and package the software stack required for Busybox boot on Armv-A Base AEM FVP platform:

  ```
  ./build-scripts/aemfvp-a/build-test-busybox.sh -p aemfvp-a all
  ```

- Command to perform an incremental build of the software components included in the software stack for the Armv-A Base platform.

  Note: this command should be followed by the `package` command to complete the preparation of the FIP and the disk image.

  ```
  ./build-scripts/aemfvp-a/build-test-busybox.sh -p aemfvp-a build
  ```

- Command to package the previously built software stack and prepare the FIP and the disk image.

  ```
  ./build-scripts/aemfvp-a/build-test-busybox.sh -p aemfvp-a package
  ```

Individual software component build (Optional)

The user can use the following command if they want to rebuild individual software component. Each software component has a separate script to build that component. After the software components have been built they must be packaged as a disk image using `build-test-busybox.sh`.

To rebuild individual software components, the command to be used is

```
./build-scripts/build-<component>.sh -p <platform name> -f <filesystem> <command>
```

Supported command line options are listed below

- <component>
  - Supported components are
    - `arm-tf`
    - `uboot`
    - `uefi`
    - `grub`
    - `linux`
    - `busybox`
- <platform name>
  - `aemfvp-a`
- <filesystem>
  - `busybox`
- <command>
  - Supported commands are
    - `clean`
    - `build`

To repackage the disk image, the command to be used is

```
./build-scripts/aemfvp-a/build-test-busybox.sh -p <platform name> package
```

4.1.4 Boot up to Busybox

After the build of the platform software stack for Busybox is complete, the following commands can be used to start the execution of the Armv-A Base AEM FVP model and boot the platform up to the Busybox prompt. Refer to the user guide section `Obtaining the Arm-A Base AEM FVP` for information on downloading the Arm-A Base AEM FVP model. Examples of how to use the command are listed below.

To boot up to the Busybox prompt, the commands to be used are

- Set `MODEL` path before launching the model:

  ```
  export MODEL=<absolute path to the platform Armv-A AEM FVP binary>
  ```

- Launch Busybox boot:

```
./model-scripts/aemfvp-a/boot.sh -p <platform name> -b <bootloader> -n [true|false]
```

Supported command line options are listed below

- -p <platform name>
  - `aemfvp-a`
- -b <bootloader>
  - Select bootloader to boot with Busybox.
  - `uefi` (Default)
  - `u-boot`
- -n [true|false] (optional)
  - Enable or disable network controller support on the platform. If network ports have to be enabled, use 'true' as the option. Default value is set to 'false'.

Example commands for booting Busybox are listed below.

- Command to start the execution of the Armv-A Base model to boot up to the Busybox prompt using uefi:

```
./model-scripts/aemfvp-a/boot.sh -p aemfvp-a -b uefi
```

- Command to start the execution of the Armv-A Base model to boot up to the Busybox prompt using U-Boot:

```
./model-scripts/aemfvp-a/boot.sh -p aemfvp-a -b u-boot
```

- Command to start the execution of the Armv-A Base model to boot up to the Busybox prompt. The selection of the bootloader to be either UEFI or U-Boot is based on command line parameters. The model supports networking allowing the software running within the model to access the network.

```
./model-scripts/aemfvp-a/boot.sh -p aemfvp-a -b [uefi|u-boot] -n true
```

  Note: If the user stops autoboot while booting with U-Boot and enters the U-Boot command line console, the user must run the following command to continue the boot process.

```
run bootcmd
```

  Command to exit from the command line console.

```
Ctrl + ]
telnet> close
```

When the script is executed, four terminal instances will be launched. The usage of each terminal can be seen below,

- Terminal-0 is the debug console for the AP (Application Processor) and contains the booting logs of Trusted firmware-A, U-Boot, Linux, and user-space applications.
- Terminal-1 is the debug console for the AP (Application Processor) and contains the UEFI boot logs.
- Terminal-2 is the debug console that displays the localhost information.
- The fourth Terminal uses the GUI representation of the model, which contains information about the overall executed instructions of the CPUs and the status of each CPU in the clusters.

The AP will start booting Trusted Firmware-A, followed by UEFI/U-Boot, Linux, and then BusyBox.

To run terminal-only mode on hosts without graphics/display:

```
env -u DISPLAY ./model-scripts/aemfvp-a/boot.sh -p <platform name> -b <bootloader>
```

This launches FVP in the background and automatically connects to the interactive application console via telnet.

To stop the model, exit telnet:

```
Ctrl + ]
telnet> close
```

Note: The boot logs can be found at <aemfvp-a_workspace>/aemfvp-a path after booting busybox. The following logs are generated in this path.

- uart0.log: Terminal-0 debug console logs are stored in the uart0.log file. It is a symbolic link for uart0 log file with the latest timestamp.
- uart1.log: Terminal-1 debug console logs are stored in the uart1.log file. It is a symbolic link for uart1 log file with the latest timestamp.

4.2 Change Log

This document contains a summary of the incremental changes, features, fixes and known issues in each release of the Armv-A Base AEM FVP platform stack.

4.2.1 Release tag: AEMFVP-A-2021.09.20

Changed

- Migration of build system from Yocto to BASH based scripting environment.
- Migration of source code repository from ARM linaro GIT to ARM Gitlab.
- Migration of Kernel from 5.4 to 5.13.
- Update Trusted Firmware-A to version 2.5.
- Update U-boot to version 2021.4.
- Migrate to the latest stable EDK2 version `edk2-stable202108`.
- Migrate to the latest EDK2-platforms commit ID `46026ad759b718142e652618c399a1f68d4e3804`.
- Changed the Armv-A Base AEM FVP model parameters to enable all the CPU cores.
- Updated the Armv-A Base AEM FVP model parameters to enable network support.

Features

- (AARCH64) UEFI + Busybox boot supported with latest stable kernel version 5.13.
- (AARCH64) U-Boot + Busybox boot supported with latest stable kernel version 5.13.
- Linux distribution boot supported and the following distributions are verified with Armv-A Base AEM FVP.
  - Fedora-34-1.2
  - Ubuntu-20.04.1
  - Debian-11.0.0
- Add changes for accessing Terminal-0 via Telnet if the Armv-A base AEM FVP is running on a non-Xserver support.

Platform Support

- This Software release is tested on Armv-A Base RevC AEM FVP and version is [11.15.18 (Jul 14 2021)].

Known issues or Limitations

- Ubuntu 20.04.1 installation may take longer time compared to other distribution installations on Armv-A Base AEM FVP.
- The Armv-A Base AEM FVP model hangs when Linux reboot is triggered after the model is booted with U-Boot and Busybox.

4.2.2 Change logs for the previous releases

- Refer to the Old Change Log for detailed information on software features and changes for the previous releases.

4.3 Install and boot a Linux distribution on Armv-A Base AEM FVP Platforms

Contents

4.3.1 Linux distribution boot

The Armv-A Base AEM FVP platform software stack supports the installation and boot of various Linux distributions such as Debian, Ubuntu, and Fedora. The distribution is installed on a SATA disk and since the installed image is persistent it can be used for multiple boots.

4.3.2 Download the platform software

To obtain the required sources for the platform, follow the steps listed on the user guide page. Ensure that the platform software is downloaded before proceeding with the steps listed below. Also, note the host machine requirements listed on that page which is essential to build and execute the platform software stack.

Skip this section if the required sources have been downloaded.

4.3.3 Build the platform software

This section describes the procedure to build the platform firmware required to install and boot a Linux distribution on Armv-A Base AEM FVP platforms.

To build the Armv-A Base AEM FVP software stack, the command to be used is

```
./build-scripts/build-test-uefi.sh -p <platform name> <command>
```

Supported command line options are listed below

- <platform name>
  - `aemfvp-a`
- <command>
  - `clean`
  - `build`
  - `package`
  - `all` (all of the three above)

Examples of the build command are

- Command to clean, build and package the Armv8-A Base AEM FVP software stack required for the distribution installation/boot on the platform:

  ```
  ./build-scripts/build-test-uefi.sh -p aemfvp-a all
  ```

- Command to remove the generated outputs (binaries) of the software stack for the Armv8-A Base FVP platform:

  ```
  ./build-scripts/build-test-uefi.sh -p aemfvp-a clean
  ```

- Command to perform an incremental build of the software components included in the software stack for the Armv8-A Base platform:

  Note: this command should be followed by the `package` command to complete the preparation of the fip image.

  ```
  ./build-scripts/build-test-uefi.sh -p aemfvp-a build
  ```

- Command to package the previously built software stack and prepares the fip image:

```
./build-scripts/build-test-uefi.sh -p aemfvp-a package
```

4.3.4 Installing a Linux distribution

After the platform firmware build for the distribution install/boot is complete, a distribution can be installed into a SATA disk image. Before beginning the installation process, download the CD iso image of the required distribution version.

The distribution installation images can be downloaded from the following locations (select an image built for aarch64 architecture):

- Fedora-34-1.2
- Ubuntu-20.04.1
- Debian-11.0.0

Refer to the user guide section `Obtaining the Arm-A Base AEM FVP` for information on downloading the Arm-A Base AEM FVP model.

The commands used to begin the distribution installation are:

- Set `MODEL` path before launching the model:

```
export MODEL=<absolute path to the platform FVP binary>
```

- Launch the installation:

```
./model-scripts/aemfvp-a/distro.sh -p <platform name> -i <abs_iso_image_path> -s
↪<disk size> -n [true|false]
```

Supported command line options are listed below

- -p <platform name>
  - `aemfvp-a`
- -i <abs_iso_image_path>
  - Absolute path to the downloaded distribution installer disk image.
- -s <disk_size>
  - Size of the SATA disk image (in GB) to be created. 12GB and above is good enough for most use cases.
- -n [true|false] (optional)
  - Enable or disable network controller support on the platform. If network ports have to be enabled, use 'true' as the option. Default value is set to 'false'.

An example of a command to install the Fedora distribution is listed below.

```
./model-scripts/aemfvp-a/distro.sh -p aemfvp-a -i /absolute/path/to/Fedora-Server-dvd-
↪aarch64-34-1.2.iso -s 16
```

- This command creates a 16GB SATA disk image, boots the Armv8-A Base software stack and starts the installation of Fedora distribution.
- From here on, follow the instructions of the chosen distribution installer. For more information about the installation procedure, refer online installation manuals of the chosen distribution.

- After the installation is completed, the disk image with a random name "<number>.satadisk" will be created in satadisk/ folder. Use this disk image for booting the installed distribution.

Additional distribution specific instructions (if any)

- Debian Distribution installation:
  - During installation, the installer will prompt the user with the message 'Load CD-ROM drivers from removable media' and display two options - Yes/No. Select the option 'No'. This is followed by another prompt 'Manually select a CD-ROM module and device?' and displays two options - Yes/No. Select the option 'Yes'. This brings up the module list required for accessing CD-ROM and lists two options - 'none' and 'cdrom'. Select the option 'none' and enter `/dev/vda`. The installation media on the virtio disk will be detected and installation continues.

4.3.5 Booting a Linux distribution

Refer to the user guide section `Obtaining the Arm-A Base AEM FVP` for information on downloading the Arm-A Base AEM FVP model.

To boot the installed distribution, use the following commands:

- Set `MODEL` path before launching the model:

```
export MODEL=<absolute path to the platform FVP binary>
```

- Start the distribution boot:

```
./model-scripts/aemfvp-a/distro.sh -p <platform name> -d <satadisk_path> -n␣
↪[true|false]
```

Supported command line options are listed below

- -p <platform name>
  - `aemfvp-a`
- -d <satadisk_path>
  - Absolute path to the installed distribution disk image created using the instructions listed in the previous section.
- -n [true|false] (optional)
  - Enable or disable network controller support on the platform. If network ports have to be enabled, use 'true' as the option. Default value is set to 'false'.

Example commands to boot a Linux distribution are listed below.

- Command to look for the available `.satadisk` image in the `satadisk/` folder and boots with that image. If multiple `.satadisk` images are found, it will list them all but won't boot:

```
./model-scripts/aemfvp-a/distro.sh -p aemfvp-a
```

- Command to begin the distribution boot from the `fedora.satadisk` image:

```
./model-scripts/aemfvp-a/distro.sh -p aemfvp-a -d /absolute/path/to/fedora.satadisk
```

When the script is executed, four terminal instances will be launched. The usage of each terminal can be seen below,

- Terminal-0 is the debug console for the AP (Application Processor) and contains the booting logs of Trusted firmware-A, Linux, and user-space applications.
- Terminal-1 is the debug console for the AP (Application Processor) and contains the UEFI boot logs.
- Terminal-2 is the debug console that displays the localhost information.
- The fourth Terminal uses the GUI representation of the model, which contains information about the overall executed instructions of the CPUs and the status of each CPU in the clusters.

The AP will start booting Trusted Firmware-A, followed by UEFI, Linux, and then Distribution.

To run terminal-only mode on hosts without graphics/display:

```
env -u DISPLAY ./model-scripts/aemfvp-a/distro.sh -p <platform name> -d <satadisk_path>
```

This launches FVP in the background and automatically connects to the interactive application console via telnet.

To stop the model, exit telnet:

```
Ctrl + ]
telnet> close
```

Note: The boot logs can be found at <aemfvp-a_workspace>/aemfvp-a path after booting distribution. The following logs are generated in this path.

- uart0.log: Terminal-0 debug console logs are stored in the uart0.log file. It is a symbolic link for uart0 log file with the latest timestamp.
- uart1.log: Terminal-1 debug console logs are stored in the uart1.log file. It is a symbolic link for uart1 log file with the latest timestamp.

4.4 Troubleshooting Guide

Contents

4.4.1 Introduction

The documentation for Armv-A Base AEM FVP platform software typically suffices in most cases but there could be certain host development machine dependencies that could cause failures either during the build or execution stages. This page provides solutions for known issues that could affect the use of the platform software stack.

4.4.2 Error while using repo command

The *repo init* or *repo sync* command fails with the below listed error message.

```
File "<path-to-workspace>/.repo/repo/main.py", line 79
file=sys.stderr)
    ^
SyntaxError: invalid syntax
```

The typical reason for this failure could be that the default version of python on the development machine is not Python3.6. To resolve this issue, install the latest version of python, if not already installed on the development machine and invoke the repo command from */usr/bin/* with *python3* as listed below.

```
python3 /usr/bin/repo init \
              -u https://git.gitlab.arm.com/arm-reference-solutions/arm-reference-
↪solutions-manifest.git \
              -m pinned-aemfvp-a.xml \
              -b refs/tags/AEMFVP-A-2021.09.20

python3 /usr/bin/repo sync
```

4.4.3 Builds do not progress to completion

During the build of the platform software stack, components such as GRUB download additional code from remote repositories using the git port (or the git protocol). Development machines on which git port is blocked, the build does not progress to completion, waiting for the additional code to be downloaded. This typically is observed when setting up a new platform software workspace.

As a workaround, use https instead of git protocol for cloning required git submodules of the various components in the software stack. A patch, as an example of this change in the GRUB component, is listed below.

```
diff --git a/bootstrap b/bootstrap
index 5b08e7e2d..031784582 100755
--- a/bootstrap
+++ b/bootstrap
@@ -47,7 +47,7 @@ PERL="${PERL-perl}"

 me=$0

-default_gnulib_url=git://git.sv.gnu.org/gnulib
+default_gnulib_url=https://git.savannah.gnu.org/git/gnulib.git

 usage() {
   cat <<EOF
```

4.4.4 FVP closes abruptly

Tests such as distribution installation take few hours to complete on Armv-A Base AEM FVP. If the model quits abruptly during its execution without any particular error message displayed on the model launch window, the host machine's memory requirements have to be rechecked. Refer to the model document for more information about the system requirements and follow the recommended configurations.

4.4.5 Repo sync fails when downloading Linux repo

If the download of the Linux repo fails during the execution of the 'repo sync' command, rerun the repo init command with the "–depth=1" parameter appended to the repo init command. This parameter reduces the commit history that is downloaded and can reduce the failures in downloading Linux repo.

4.5 Armv-A Base AEM FVP platform software user guide

Contents

4.5.1 Introduction

The Armv-A Base AEM FVP (Fixed Virtual Platform) is an evolution of the base platform, enhanced to support the exploration of system level virtualization. It is an Architecture Envelope Model (AEM) which incorporates two AEM clusters, each of which can be configured with up to four cores.

This document is a user guide on how to set up, build and run the software stack on the Armv-A Base AEM FVP (Fixed Virtual Platform).

4.5.2 Host prerequisites for a validated build environment

- Ubuntu Linux 18.04 LTS
- Minimum of 50 GB free storage space.
- Commands provided in this guide are executed from a `bash` shell environment.

Packages

To ensure that all the required packages are installed, run:

```
sudo apt-get update
sudo apt-get install make autoconf autopoint bc bison build-essential curl \
    device-tree-compiler dosfstools flex gettext-base git libssl-dev m4 \
    mtools parted pkg-config python python3-distutils rsync unzip uuid-dev \
    wget acpica-tools fuseext2 iasl
```

To ensure that all the required packages for FVP are installed, run:

```
sudo apt-get install telnet xterm
```

Repo

Follow the instructions provided in the repo README file to install the `repo tool`.

NOTE: The repo tool which gets installed using apt-get command sometimes return errors, in such a case it's recommended to install repo using the `curl` method.

The `repo` tool uses `git` to download the source code. It should be configured before using the `repo` tool.

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

4.5.3 Downloading the software stack

The manifest files, which contain the location of all the git repositories of Armv-A Base AEM FVP platform software stack, are available here. This section explains the procedure to sync the software stack.

- Create a new empty folder

  ```
  mkdir <aemfvp-a_workspace>
  cd <aemfvp-a_workspace>
  ```

- For fetching the latest *stable* software stack, use the following commands to sync:

```
repo init \
     -u https://git.gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest.git \
     -m pinned-aemfvp-a.xml \
     -b refs/tags/AEMFVP-A-2021.09.20 --depth=1

repo sync
```

Note: The repo tool requires at least Python 3.6 to be installed on the development machine. On machines where python3 is not used as default, the repo init command will fail to complete. Refer to the troubleshooting guide for resolving this issue.

Note: To fetch the entire commit history that is being downloaded, remove `--depth=1` from the repo init command. This will increase the time taken to download the platform software stack.

4.5.4 Verify prerequisites

Run the following command to verify all the required prerequisites to build the software stack:

```
sudo ./build-scripts/check_dep.sh
```

Note: This command checks and notifies to the user if the host machine is missing any required packages to build the software stack. The user has to install the missing packages.

4.5.5 Obtaining the Armv-A Base AEM FVP

The latest version of the Armv-A Base AEM FVP model can be downloaded from Arm Ecosystem FVPs and select `Armv-A Base RevC AEM FVP`.

Follow the instructions of the installer and set up the FVP. The installer, by default, selects the home directory to install the FVP. To opt for a different directory than the one selected by the installer, provide an absolute path to that directory when prompted for during the FVP installation process.

4.5.6 Enable network on Armv-A Base AEM FVP

The Armv-A Base AEM FVP supports virtual ethernet interface to allow networking support, used for the software executed by the Armv-A Base AEM FVP. If support for networking is required, the host TAP interface has to be set up before the Armv-A Base AEM FVP is launched. To set up the TAP interface, execute the following commands on the host machine.

- Install libvirt-bin

  ```
  sudo apt-get install libvirt-bin
  ```

  Note: The above command creates the network interface `virbr0` with the IP address `192.168.122.1`. This can be checked with the command `ifconfig`. If the interface is not created, run the following command to restart the libvirt daemon.

  ```
  sudo systemctl restart libvirt-bin.service
  ```

- Create a tap interface named 'tap0'

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

4.5.7 Supported features

Armv-A Base AEM FVP software stack supports the following features.

- Busybox Boot.
- Distribution Boot.

Follow the links above for detailed information about the build and execute steps for each of the supported features.

4.5.8 Software Components

The following software components are involved in the Armv-A Base AEM FVP booting.

1. Trusted Firmware-A (TF-A)
2. U-Boot
3. UEFI
4. GRUB
5. Linux
6. Root Filesystem (Busybox or Distribution)

Trusted Firmware-A

Based on Trusted Firmware-A.

Build script	<aemfvp-a_workspace>/build-scripts/build-arm-tf.sh
Check-out path	<aemfvp-a_workspace>/arm-tf
Output	<aemfvp-a_workspace>/output/aemfvp-a/aemfvp-a/tf-bl1.bin <aemfvp-a_workspace>/output/aemfvp-a/aemfvp-a/tf-bl2.bin <aemfvp-a_workspace>/output/aemfvp-a/aemfvp-a/tf-b31.bin

U-Boot

Based on U-Boot.

Build script	<aemfvp-a_workspace>/build-scripts/build-uboot.sh
Checkout path	<aemfvp-a_workspace>/u-boot
Output	<aemfvp-a_workspace>/output/aemfvp-a/aemfvp-a/uboot.bin

UEFI

Based on edk2 and edk2-platforms

Build script	<aemfvp-a_workspace>/build-scripts/build-uefi.sh
Checkout path	<aemfvp-a_workspace>/uefi/edk2 <aemfvp-a_workspace>/uefi/edk2/edk2-platforms
Output	<aemfvp-a_workspace>/output/aemfvp-a/aemfvp-a/uefi.bin

GRUB

Based on GRUB.

Build script	<aemfvp-a_workspace>/build-scripts/build-grub.sh
Checkout path	<aemfvp-a_workspace>/grub
Output	<aemfvp-a_workspace>/grub/output/grubaa64.efi

Linux

Based on Linux.

Build script	<aemfvp-a_workspace>/build-scripts/build-linux.sh
Checkout path	<aemfvp-a_workspace>/linux
Output	<aemfvp-a_workspace>/output/aemfvp-a/aemfvp-a/Image

Root Filesystem (Busybox)

The Busybox provides the minimal Rootfs.

Build script	<aemfvp-a_workspace>/build-scripts/build-busybox.sh
Checkout path	<aemfvp-a_workspace>/busybox
Output	<aemfvp-a_workspace>/busybox/out/arm64/_install/

Root Filesystem (Distribution)

This environment provides the base firmware (TF-A and UEFI) to boot the distribution on Armv-A Base AEM FVP and does not have a separate script to create the installable distribution images. The installable distribution images are fetched from the corresponding repository. The GRUB, Linux, and Rootfs are parts of the installable distribution images.

4.5.9 Report security vulnerabilities

For reporting security vulnerabilities, please refer Vulnerability reporting page.

4.5.10 Release Tags

The table below lists the release tags for the Armv-A Base AEM FVP software stack and the corresponding Armv-A Base AEM FVP version that is recommended to be used along with the listed release tag. The summary of the software feature and changes introduced in each release is listed in change log.

Release Tag	Armv-A Base AEM FVP Version
AEMFVP-A-2021.09.20	11.15.18

4.6 Guidelines to Vulnerability reporting

- Architecture Envelope Model(AEMFVP-A) platform software stack is a collection of open source software repositories. If you think you have found a security vulnerability in a specific open source project which is part of the software stack, it is recommended to follow the vulnerability reporting guidelines specified by the respective project.
- If you think you have found a security vulnerability as part of the Architecture Envelope Model(AEMFVP-A) platform software stack and does not fall into any specific open source project, then please report by email at arm-security@arm.com specifying the project name as "Architecture Envelope Model(AEMFVP-A) Platform Software". More details can be found at Arm Developer website.
