
arm
Firmware Handoff on FVP
Harrison Mutai and Manish Pandey
22 February 2023

Punch-Cards and Bootstraps

- Early programming involved using patch-cables or toggle switches.
- Someone proposed creating a **small program** capable of **loading larger programs** stored on punched cards or paper tape.
- Like lifting oneself by bootstraps, it initiated the system.
- Bootloaders have evolved to handle the increasing complexity of modern computing.

NASA, Public domain, via Wikimedia Commons

Contents

Firmware Handoff Technical Overview

Objectives

- Standardizing information passing between boot stages
  - Well defined and scalable data-structure where information propagated as the system boots.
  - Standardizing usage of scratch registers (i.e. x0-x3) at handoff boundaries
- Standard aims to satisfy requirements from different BL33 solutions:
  - Linux (LinuxBoot)
  - Xen
  - EDK2
  - *Partners with closed source BL33 FW
  - U-boot
  - Coreboot – it’s not used as BL33 yet, some partners plan to in the future.

Firmware Handoff Technical Overview

Core Concepts

- Spec introduces concept of a Transfer List (TL) and Transfer Entry (TE)
- Any stage in the boot process can produce information that is consumed by later boot stages
- TL resides in contiguous physical address space
- Any information produced by a firmware stage must be encapsulated by a TE.
- TE cannot be a pointer to a separate memory location

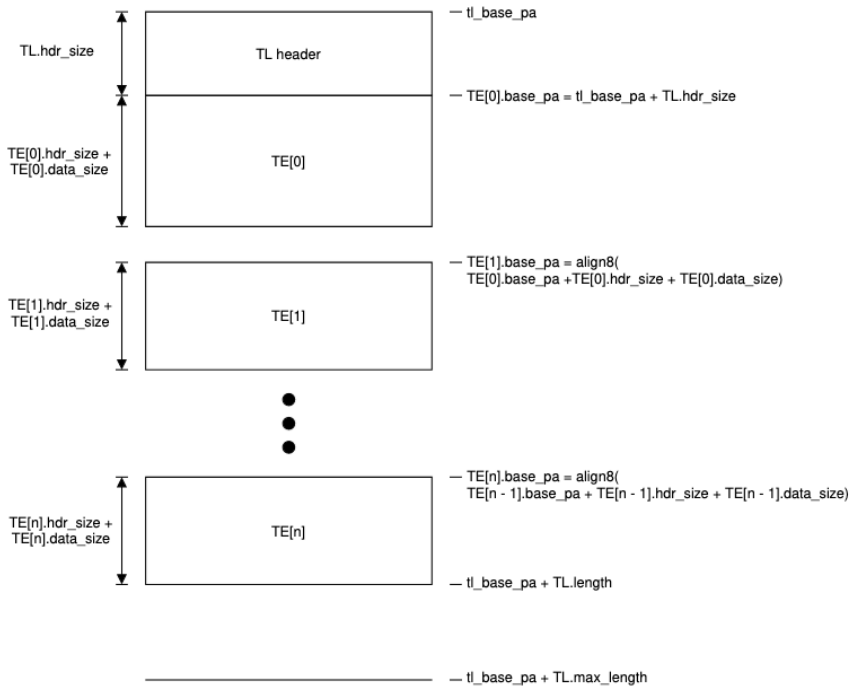


Figure 1: Transfer list example layout [1]

Firmware Handoff Technical Overview

Transfer List Requirements

- TL composed of header, followed by sequence of TE's
- TL header specifies:
  - Signature
  - Checksum
  - Version
  - Header Size
  - Alignment (maximum allowed by the TL)
  - Maximum allowable size
  - Size of the entire TL in bytes
- TL and TE headers must be 8-byte aligned.

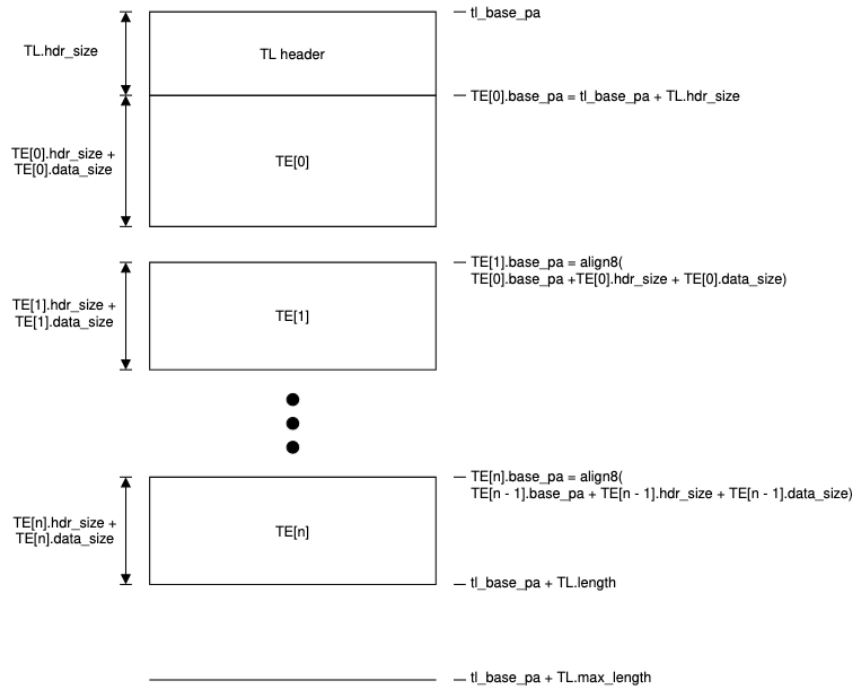


Figure 1: Transfer list example layout [1]

Firmware Handoff Technical Overview

Transfer Entry Requirements

- TE's start with an entry header followed by a data section.
- TE header contains:
  - Unique tag to identify contents
  - Header size
  - Exact size of data contents
- TE's can be added or removed at any stage.
- Tag examples: FDT, HOB block, HOB list, ACPI table.
- Tags must be allocated in specification before use!

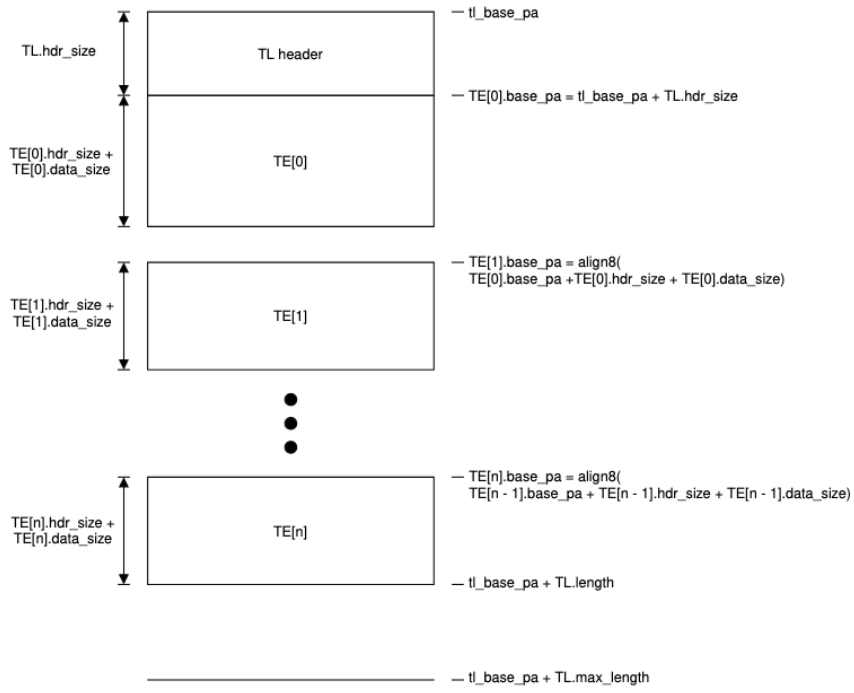


Figure 1: Transfer list example layout [1]

Firmware Handoff Technical Overview

Tag Allocation

- New tag ID allocated by submitting a PR to GitHub repository: https://github.com/FirmwareHandoff/firmware_handoff
- There is a generic range applicable to all projects.
- Spec allows IMPDEF tags to be added in separate range:
  - Encouraged to group tags in logical clusters at 16- or 256-byte boundaries (i.e. tags related to firmware project or chipset).
  - Trusted Firmware related projects have their own range.
  - The {0xff_f000, . . . , 0xff_ffff} range is reserved for non-standardized use, don't need to raise a PR (strongly discouraged except for local experiments!)
- Tag should have a simple layout representable by a C structure.

Firmware Handoff Technical Overview

Register Convention (Aarch64)

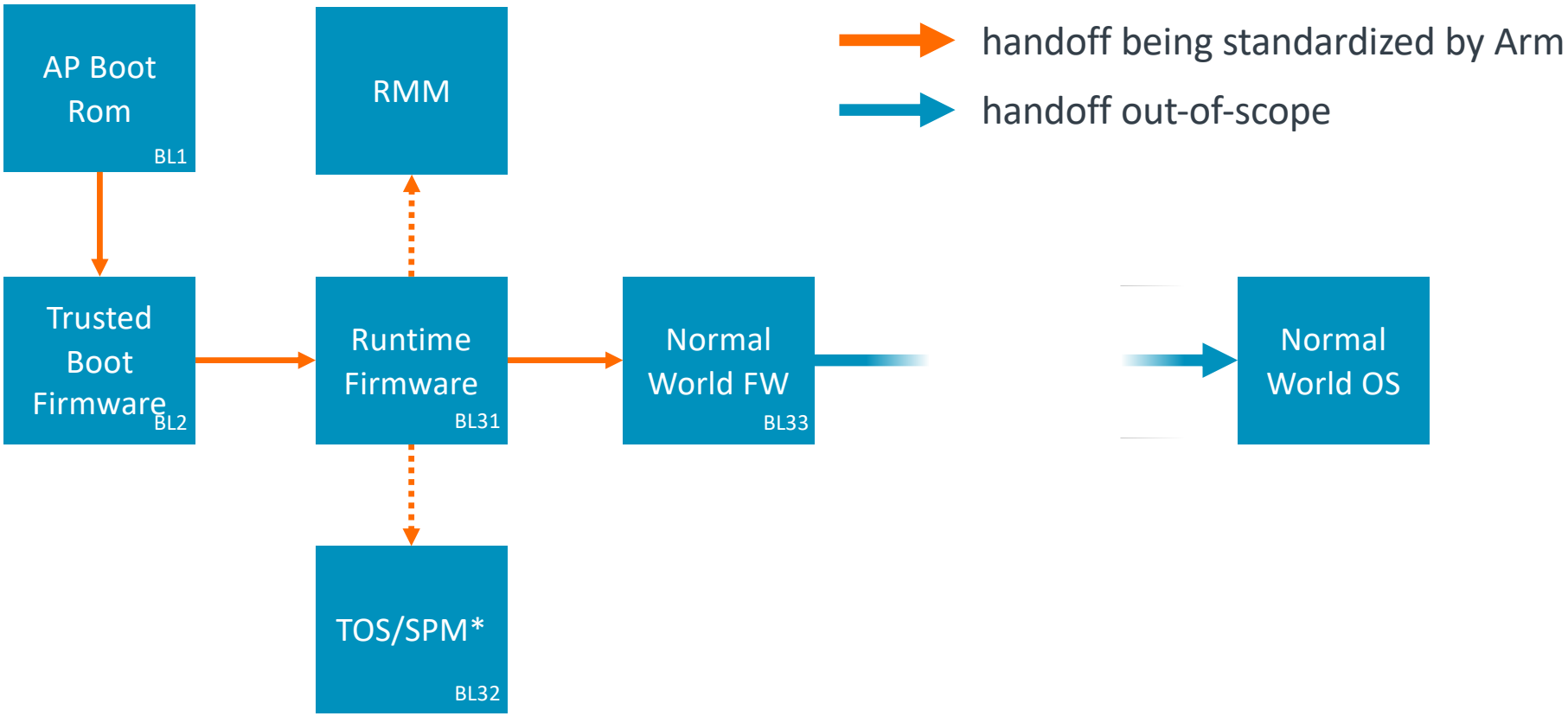
Register	Contents
x0	Base address of FDT if it exists in TL, otherwise 0.
x1	X1 is divided into the following fields: • X1[23:0]: set to the TL signature (0x4a0f_b10b) • X1[31:24]: version of register convention • X1[63:32]: reserved, must be zero.
x2	Reserved, must be zero.
x3	TL Base Address

Firmware Handoff Technical Overview

Register Convention (Aarch32)

Register	Contents
r0	Reserved, must be zero.
r1	R1 is divided into the following fields: • R1[23:0]: set to the TL signature (4a0f_b10b) • R1[31:24]: version of the register convention used.
r2	Base address of FDT if it exists in TL, otherwise 0.
r3	TL Base Address

Which boundaries are we standardizing?

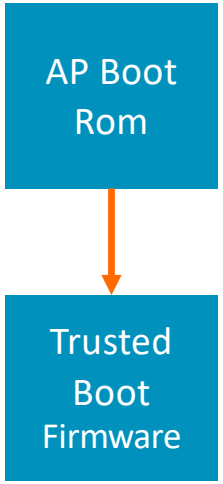


BL1 to BL2

Overview of Changes

- Firmware Configuration (FW_CONFIG) contains load info for other configuration files
  - BL2 uses this to access the Trusted Boot Firmware Configuration (TB_FW_CONFIG)
  - TB_FW_CONFIG is a device tree containing firmware settings i.e. io policies, Mbed-TLS heap info
- SRAM layout provides BL2 with read-write memory it can allocate to its memory map in Trusted SRAM

Register	Contents
x0	FW_CONFIG
x1	SRAM Layout
x2	0
x3	0



BL1 to BL2

Overview of Changes

- Re-arrange memory map
  - Memory region accessible to both stages
  - Ensure TL won't be reclaimed during image loading
- Relocate data structures shared between BL1 and BL2
  - Trusted SRAM Memory layout #36
  - Trusted Board Firmware configuration #37
- Do we still need the firmware configuration tree if all device trees reside in a TL?
- https://review.trustedfirmware.org/c/TF-A/trusted-firmware-a/+/26638

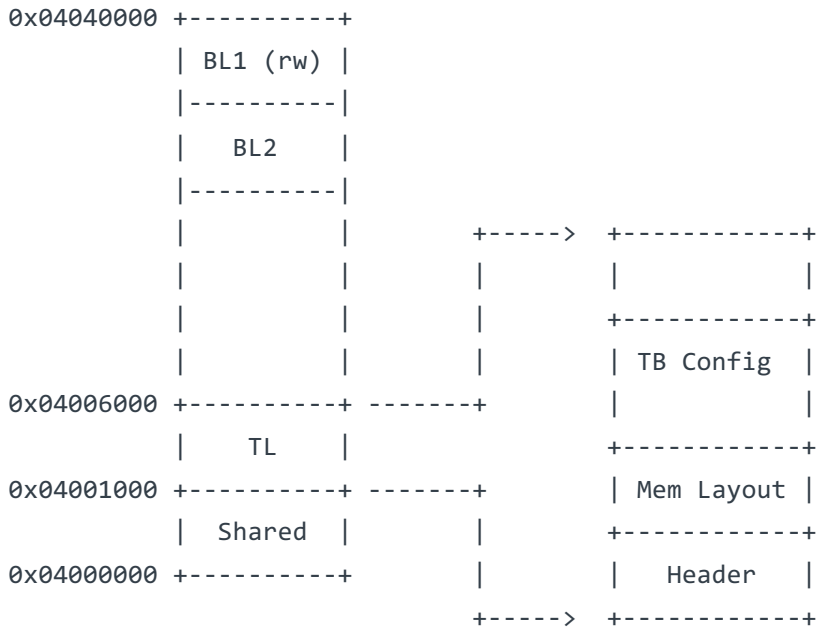


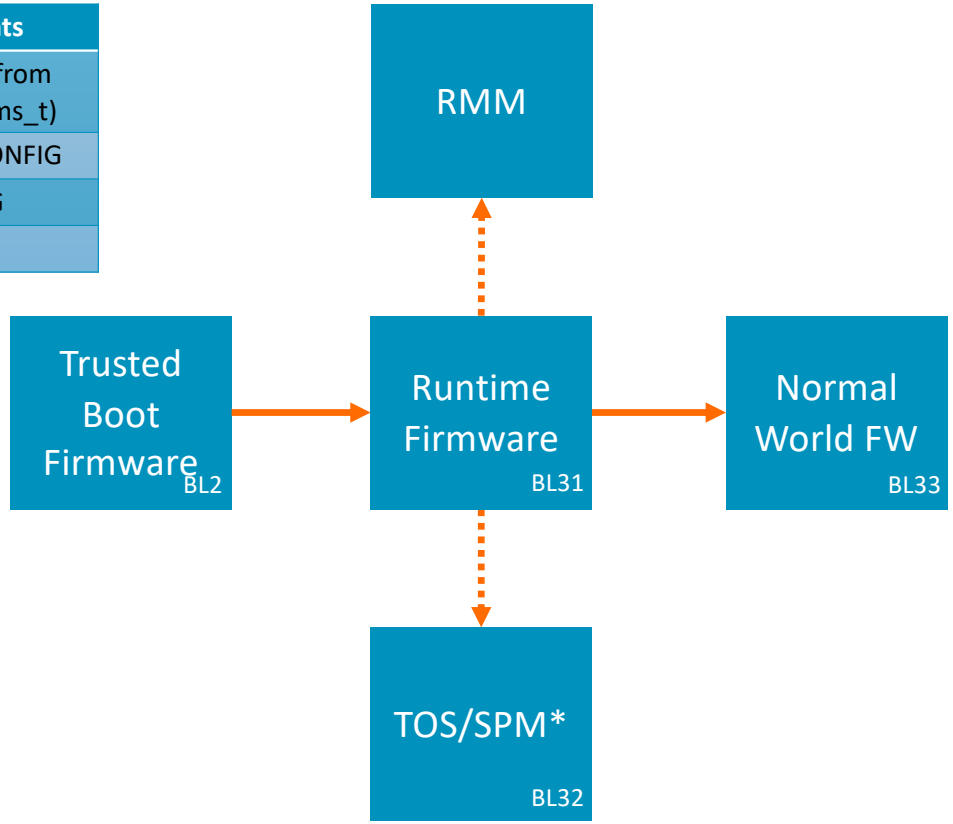
Figure 2: FVP Trusted SRAM Layout

BL2 to BL31

Overview of Changes

- x0 contains linked list of executable image info
  - Only needs information on how to execute them
  - i.e. state of general-purpose registers, PC, and SPSR
  - entry_point_info_t
- HW_CONFIG is a copy of device tree passed to BL33
- SOC_FW_CONFIG is a subset of the HW_CONFIG

Register	Contents
x0	Parameters from B2 (bl_params_t)
x1	SOC_FW_CONFIG
x2	HW_CONFIG
x3	0



BL2 to BL31

Overview of Changes

- Pass executable image info directly to BL31 instead of linked list:
  - Create new TE to encapsulate an entry_point_info structure.
  - Multiple TE's can have the same tag ID to support multiple executables.
  - Distinguish images using the attributes field in the structure header (param_header).
  - XFERLIST_EXEC_IMG_EP_INFO64 [PR - #31]
- SOC_FW_CONFIG currently unused, no need to currently support it.
  - Coud we conditionally compile it from the HW_CONFIG?
- Pass copy of the hardware description device tree also passed to BL31 as a TE using the standard FDT entry

Register	Contents
x0	FDT Address
x1	TL Metadata
x2	0
x3	TL Base Address

Next Steps

- Testing BL2-BL31 interface with four worlds
- Investigating BL31 and BL32 (TOS || SPM)
- Investigating BL31 and RMM boundary, determining whether any work is required
- Testing entire boot flow in TF-A + EDK2/U-Boot, and any other BL33 solutions
- Extending support to other Arm platforms and making it the default choice for FVP…
- Supporting partner adoption
- Support for TL library in other firmware projects
  - OP-TEE, u-boot, edk-II, Hafnium, etc.
  - Hosting place for the library?

References

1. https://github.com/FirmwareHandoff/firmware_handoff/releases/download/v0.9/firmware_handoff.pdf
2. https://static.linaro.org/connect/sfo17/Presentations/SFO17-310%20Dynamic%20secure%20firmware%20configuration%20v1.0.pdf
3. https://learn.adafruit.com/bootloader-basics/a-brief-history-of-bootloading

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు

- The Firmware Configuration Framework (FCONF)[9] is a way to offer more flexibility in the firmware. It is used to provide most of the platform-specific data that were previously hard coded inside the firmware. This framework uses device tree (one or multiple) that are passed to the firmware during load processing. BL2 uses it to describe the chain of trust and the images list to be loaded.
- Thanks to device tree usage, the configuration becomes dynamic at boot time. The current implementation uses the following device tree as framework entry:
  - FW_CONFIG - The firmware configuration file. Hold properties shared across all BLx images. An example is the dtb-registry node, which contains the information about other binaries configuration (load-address, size, image_id).
  - HW_CONFIG - The hardware configuration file. Can be shared by all Boot Loader stages and also by the Normal World Rich OS.
  - TB_FW_CONFIG - Trusted Boot Firmware configuration file. Shared between BL1 and BL2.
  - SOC_FW_CONFIG - SoC Firmware configuration file. Used by BL31.
  - TOS_FW_CONFIG - Trusted OS Firmware configuration file. Used by Trusted OS (BL32).
  - NT_FW_CONFIG - Non Trusted Firmware configuration file. Used by Non-trusted firmware (BL33).

SPMC Manifest

- SPMC manifest contains following info
  - Attributes (spmc_id, versions, exec state, load_address, entry_point, binary_size)
  - Hypervisor (related with VMs)
  - CPUs
  - Memory
- Can we divide this manifest in two parts? One consumed by SPMD and the other by SPMC.
- The one passed to SPMC can be added as a TE to be passed at BL31 -> BL32 interface.
- For the attributes part we can use **XFERLIST_DT_SPMC_MANIFEST**
- The simpler solution is to keep everything as part of **XFERLIST_DT_SPMC_MANIFEST**