HOW TO COMPILE TF-A BINARIES AND RUN HELLO_WORLD UNDER UBOOT USING LS1043ARDB

Contents

- **General boot flow……………………………………………………………………1**
- **TF-A Boot Flow………………………………………………………………………2**
- **COMPILE PBL BINARY FROM RCW SOURCE FILE……………………………5**
- **COMPILE U-BOOT BINARY ………………………………………………………6**
  - **COMPILE U-BOOT BINARY AND HELLO_WORLD STANDALONE……..6**
- **Compile TF-A binaries (bl2_.pbl and fip.bin)…………………………………7**
  - **HOW TO COMPILE BL2 BINARY………………………………………….8**
  - **HOW TO COMPILE FIP BINARY…………………………………………9**
- **Program TF-A binaries on specific boot mode………………………………10**
  - **Program TF-A binaries on IFC NOR flash……………………………10**
  - **Program TF-A binaries on NAND flash………………………………11**
  - **Program TF-A binaries on SD card…………………………………….12**
- **How to run Hello_world program under U-boot……………………………13**

General boot flow

NXP SoC Booting Principles The high-level boot flow of an ARMv8-A SoC is:

1. SoC comes out of reset and reads RCW/PBL image from a boot source, such as a NOR flash, SD card, or eMMC flash. The RCW/PBL image contains configuration bits that control:

    • Pin muxing and the protocol selected for SerDes pins.
    • Clock parameters and PLL multipliers.
    • Device containing the first software (not in an internal BootROM) to run.

2. Code in the internal BootROM starts running and configures low-level aspects of the SoC.

3. The BootROM must then load the first external software (TF-A binaries) to run from a boot device, such as NOR flash or SD/eMMC.

    a. The BootROM transfers control to BL2.
    b. BL2 loads and starts bootloader from NOR flash or SD/eMMC.

4. Usually, the bootloader must also load peripheral firmware, firmware required to make peripherals, such as Ethernet controllers work. The details of this differ from SoC to SoC.

5. When the bootloader finishes initialization, its job is to locate a Linux kernel image and a Linux device tree image. The device tree is a description of the board and SoC hardware that Linux uses, for example, to know which peripherals are available for use and to associate drivers with them. Often, bootloaders do some on-the-fly "fixups" to the device tree to pass information to Linux.

6. In summary, the bootloader reads kernel and device tree images from memory or mass storage device. Because bootloaders have many drivers, there are many possible choices for the location of the images.

    - NOR flash (serial QSPI or parallel)

- SD card/eMMC flash
- USB mass storage devices of all types
- SATA drives of all types
- Ethernet, normally via TFTP

7. After the bootloader loads the kernel and device tree and does fixups, it puts kernel boot parameters and the device tree into DDR where the kernel can find them and passes control to the kernel. One of the key kernel parameters is “root=”. It tells the Linux kernel what device contains the user space file set (userland). U-Boot stores kernel parameters in environment variable bootargs.

Notes on General Boot Principles

• Secure boot does not change the overall sequence. The significant difference is that secure boot involves each component (starting with the BootROM) validating the images it loads and starts. This sequence of image validations is called the “chain of trust”. Linux often resets peripherals and reloads their firmware. This process is specific to the SoCs.

TF-A Boot Flow

1. **BootROM (BL1)**

   a. When the CPU is released from reset, hardware executes PBL commands that copy the BL2 binary (bl2.bin) for platform initialization to OCRAM. The PBI commands also populate the BOOTLOC ptr to the location where bl2.bin is copied.

   b. Upon successful execution of the PBI commands, Boot ROM passes control to the BL2 image at EL3.

2. **BL2**

   a. BL2 initializes the DRAM, configures TZASC

   b. BL2 validates BL31, BL32, and BL33 images to the DDR memory

after validating these images. BL31, BL32, and BL33 images form FIP image, fip.bin.

c. Post validation of all the components of the FIP image, BL2 passes execution control to the EL3 runtime firmware image named as “BL31”.

3. **BL31**

    a. Sets up exception vector table at EL3

    b. Configures security-related settings (TZPC)

    c. Provides services to both bootloader and operating system, such as controlling core power state and bringing additional cores out of reset

    d. [Optional] Passes execution control to Trusted OS (OP-TEE) image, BL32, if BL32 image is present.

4. **BL32**

    a. [Optional] After initialization, BL32 returns control to BL31.

5. **BL31**

    a. Passes execution control to bootloader U-Boot/UEFI, BL33 at EL2

6. **BL33**

    a. Loads and starts the kernel and other firmware (if any) images

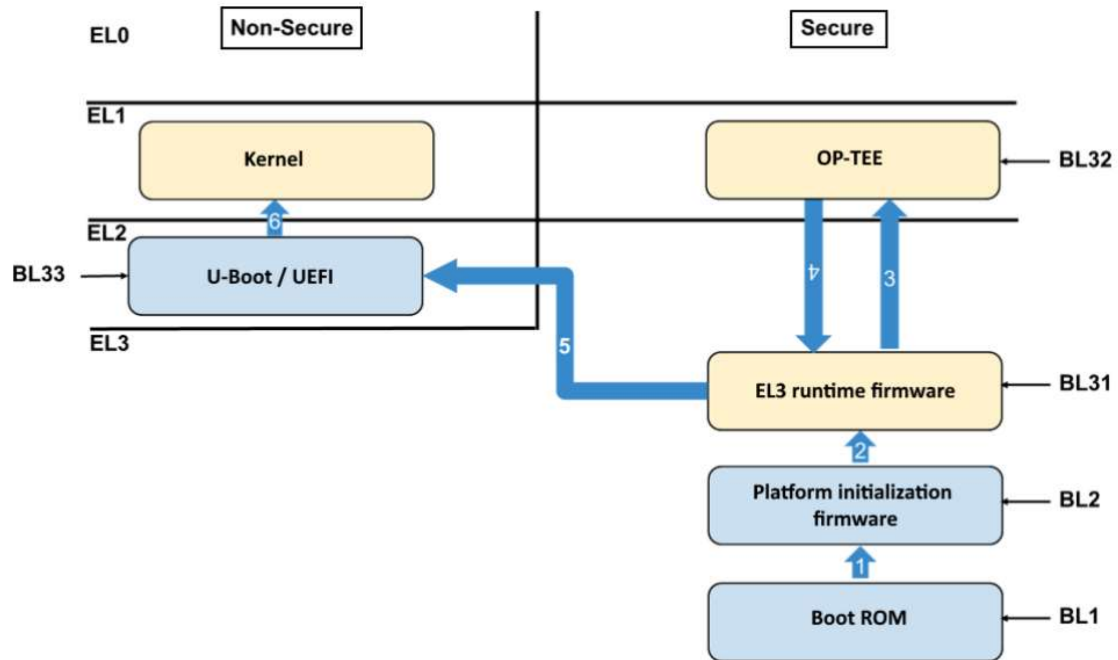

Figure 1: TF-A boot flow - stages

Follow these steps to compile and deploy TF-A binaries (bl2_<boot_mode>..pbl) on the required boot mode.

1. Compile PBL binary from RCW source file
2. Compile U-Boot binary
3. Compile TF-A binaries (bl2_.pbl and fip.bin)
4. Program TF-A binaries on specific boot mode
5. How to run Hello_world program under U-boot.

1-COMPILE PBL BINARY FROM RCW SOURCE FILE

You have to create a new directory to compile the binaries that you need to create a TF-A binary

You need to compile the rcw_<boot_mode>.bin binary to build the bl2_<boot_mode>.pbl binary.

Clone the rcw repository and compile the PBL binary.

1. $ git clone https://github.com/nxp-qoriq/rcw
2. $ cd rcw
3. $ cd ls1043ardb
4.
5. $ make

Inside of the directory called "RR_FQPP_1455" you can see some binaries in the with the next nomenclature:

rcw_<freq>.bin

Where "freq" is the frequency in MHz of the processor, the values of the frecuency are 1200MHz, 1400MHz, 1500MHz and 1600MHz

2-COMPILE U-BOOT BINARY

You need to compile the u-boot.bin binary to build the fip.bin binary.

Clone the u-boot repository and compile the U-Boot binary for TF-A

1. $ git clone https://github.com/nxp-qoriq/u-boot
2. $ cd u-boot
3. $ git checkout -b LSDK-21.08 LSDK-21.08
4. $ export ARCH=arm64
5. $ export CROSS_COMPILE=aarch64-linux-gnu-
6. $ make distclean
7. $ make ls1043ardb_tfa_defconfig
8. $ make

2.1 COMPILE U-BOOT BINARY AND HELLO_WORLD STANDALONE

1. $ git clone https://github.com/nxp-qoriq/u-boot.git
2. $ cd u-boot
3. $ git checkout -b LSDK-21.08 LSDK-21.08
4. $ export ARCH=arm64
5. $ export CROSS_COMPILE=aarch64-linux-gnu-
6. $ make distclean
   6. 1 Open with nano the file located in "configs/ ls1043ardb_tfa_defconfig
   6.2 Add to the file in the last line the next configuration "CONFIG_EXAMPLES=y"
   6.3 Close and save the file.
   6.4 Return to folder "U-boot"
7. $ make ls1043ardb_tfa_defconfig
8. $ make

3 Compile TF-A binaries (bl2_.pbl and fip.bin)

1. $ git clone https://github.com/nxp-qoriq/atf
2. $ cd atf
3. $ git checkout -b LSDK-21.08 LSDK-21.08
4. $ export ARCH=arm64
5. $ export CROSS_COMPILE=aarch64-linux-gnu-

The compiled BL2 binaries, bl2.bin and bl2_<boot mode>.pbl are available at atf/build/ls1043ardb/release/.

NOTE: For any update in the BL2 source code or RCW binary, the bl2_<boot mode>.pbl binary needs to be recompiled.

3.1 HOW TO COMPILE BL2 BINARY

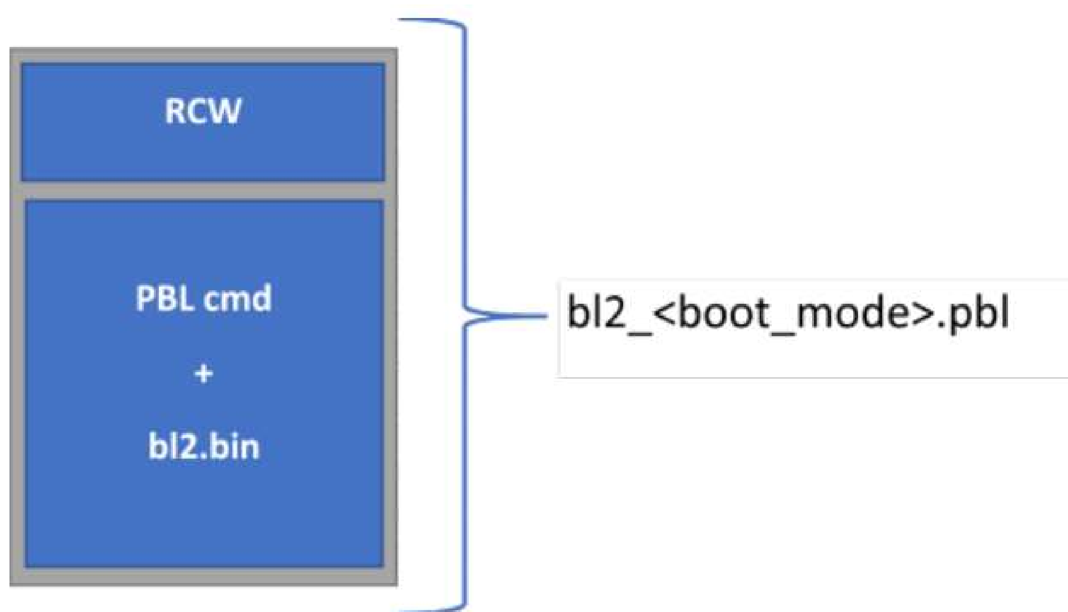

Figure 2: bl2.pbl

To compile the BL2 binary without OPTEE:

make PLAT=<platform> bl2 BOOT_MODE=<boot_mode> pbl
RCW=<path_to_rcw_binary>/<rcw_binary_for_specific_boot_mode>

To LS1043ARDB for SD boot:

make PLAT=ls1043ardb bl2 BOOT_MODE=sd pbl
RCW=<path_to_rcw_binary>/<rcw_freq.bin>

To LS1043ARDB for NOR boot:

make PLAT=ls1043ardb bl2 BOOT_MODE=nor pbl
RCW=<path_to_rcw_binary>/<rcw_freq.bin>

To LS1043ARDB for NAND boot:

make PLAT=ls1043ardb bl2 BOOT_MODE=nand pbl
RCW=<path_to_rcw_binary>/<rcw_freq.bin>

3.2 HOW TO COMPILE FIP BINARY

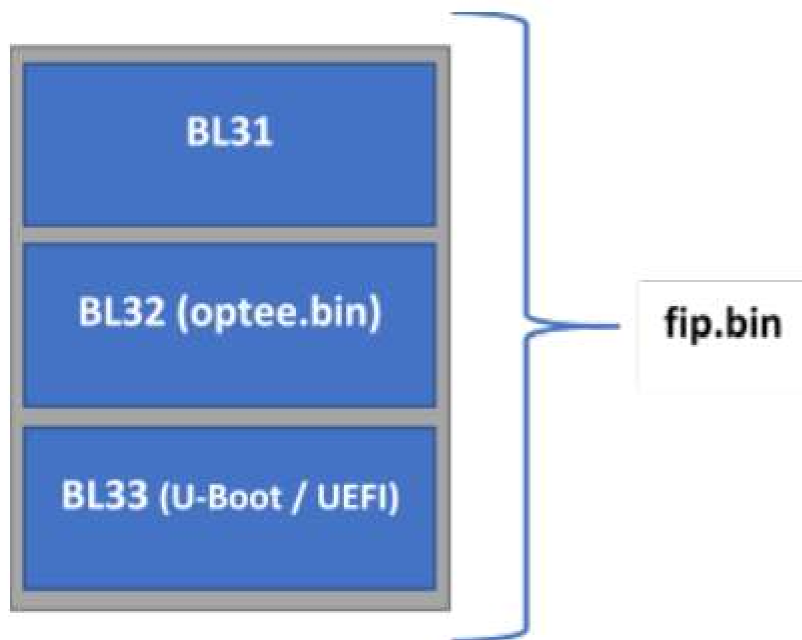

Figure 3: fip.bin

To compile the FIP binary without OPTEE and without trusted board boot:

$make PLAT=<platform> fip BL33=<path_to_u-boot_binary>/u-boot.bin

For LS1043ARDB:

$make PLAT=ls1043ardb fip BL33=<path_to_u-boot_binary>/u-boot.bin

The compiled BL31 and FIP binaries (bl31.bin, fip.bin) are available at atf/build/ls1043ardb/release/. For any update in the BL31, BL32, or BL33 binaries, the fip.bin binary needs to be recompiled.

4 Program TF-A binaries on specific boot mode

For that step you can use a tftp server, but it is easier with a USB formatted on FAT32.

You have to put the files " bl2_<boot_mode>.pbl" and "fip.bin" in the usb and follow the steps to your boot mode.

4.1 Program TF-A binaries on IFC NOR flash

For LS1043A, the steps to program TF-A binaries on IFC NOR flash are as follows:

1. **Boot the board from default bank.**

2. **Under U-boot prompt:**
   => usb start

3. **Flash bl2_nor.pbl:**

   => fatload usb 0:1 $load_addr bl2_nor.pbl

   a. **Alternate bank:**
      => protect off 64000000 +$filesize && erase 64000000 +$filesize && cp.b $load_addr 64000000 $filesize

   b. **Current bank:**
      => protect off 60000000 +$filesize && erase 60000000 +$filesize && cp.b $load_addr 60000000 $filesize

4. **Flash fip.bin:**

   => fatload usb 0:1 $load_addr fip.bin

   a. **Alternate bank:**
      => protect off 64100000 +$filesize && erase 64100000 +$filesize && cp.b $load_addr 64100000 $filesize

   b. **Current bank:**
      => protect off 60100000 +$filesize && erase 60100000 +$filesize && cp.b $load_addr 60100000 $filesize

5. **Reset your board:**

   a. **Alternate bank:**
      cpld reset altbank

   b. **Current bank:**
      cpld reset

4.2 Program TF-A binaries on NAND flash

1. **Boot the board from default bank.**

2. **Under U-boot prompt:**
   => usb start

3. **Flash bl2_nand.pbl to NAND flash:**

   => fatload usb 0:1 $load_addr bl2_nand.pbl

   => nand erase 0x0 $filesize;nand write $load_addr 0x0 $filesize;

4. **Flash fip_uboot.bin to NAND flash:**

   => fatload usb 0:1 $load_addr fip.bin

   => nand erase 0x100000 $filesize;nand write $load_addr 0x100000 $filesize;

5. **Reset your board:**
   => cpld reset nand

4.3 Program TF-A binaries on SD card

To program TF-A binaries on SD card, follow these steps:

1. **Boot the board from default bank.**

2. **Under U-boot prompt:**
   => usb start

3. **Flash bl2_sd.pbl to SD card:**
   => fatload usb 0:1 $load_addr bl2_sd.pbl

   => mmc write $ load_addr 8 A1
4. **Flash fip.bin to SD card:**

   => fatload usb 0:1 $load_addr bl2_sd.pbl

   => mmc write $load_addr 800 A1

5. **Reset your board:**
   => cpld reset sd

5 How to run Hello_world program under U-boot.

From your host machine go to the folder “u-boot/examples/standalone” and execute the next command.

$readelf hello_world -a |grep hello_world

the result is the next:

12: 0000000000000000 0 FILE LOCAL DEFAULT ABS hello_world.c
63: **0000000080300000** 220 FUNC GLOBAL DEFAULT 1 hello_world

The memory direction to be loaded the Hello_World binary in the board must be **0x80300000.**

For that step, you can use a tftp server, but it is easier with a USB formatted on FAT32.

You have to put the file “Hello_world.bin” in the usb and follow those steps.

Note: The Hello_world binary is on to folder “u-boot/examples/standalone” of the host machine.

Under U-boot:

1. **To do more easier to remember the memory direction to be loaded the program you have to create an environment variable with the direction.**

   =>setenv load_addr_standalone 0x80300000

2. **Save the environment variable.**

   =>saveenv

3. **Start the usb port to load the binary from usb.**

    =>usb start

4. **Load the binary in the direction of memory obtained on the Linux host.**

    =>fatload usb 0:1 $load_addr_standalone hello_world.bin

5. **Execute the program.**

    =>go $load_addr_standalone

```
=> usb start
starting USB...
Bus usb3@2f00000: Register 200017f NbrPorts 2
Starting the controller
USB XHCI 1.00
Bus usb3@3000000: Register 200017f NbrPorts 2
Starting the controller
USB XHCI 1.00
Bus usb3@3100000: Register 200017f NbrPorts 2
Starting the controller
USB XHCI 1.00
scanning bus usb3@2f00000 for devices... 1 USB Device(s) found
scanning bus usb3@3000000 for devices... 2 USB Device(s) found
scanning bus usb3@3100000 for devices... 1 USB Device(s) found
       scanning usb for storage devices... 1 Storage Device(s) found
=> fatload usb 0:1 $load_addr_standalone hello_world.bin
1432 bytes read in 2 ms (699.2 KiB/s)
=> go $load_addr_standalone
## Starting application at 0x80300000 ...
Example expects ABI version 9
Actual U-Boot ABI version 9
Hello World
argc = 1
argv[0] = "0x80300000"
argv[1] = "<NULL>"
Hit any key to exit ...
```