U-boot Bootloader

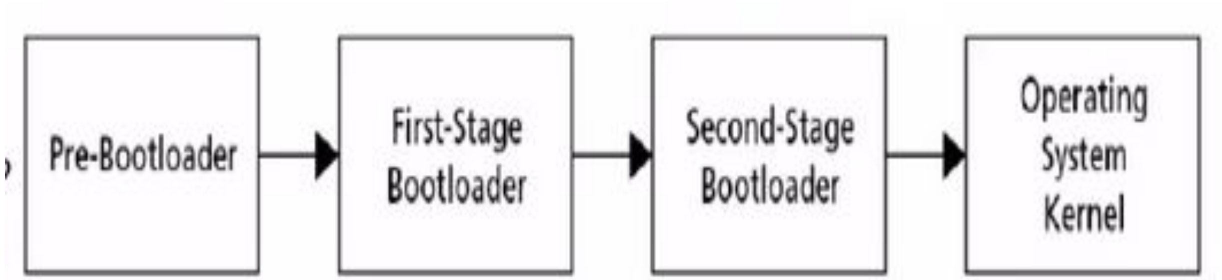
3rd July 2024

Contents

- Bootloader Functions
- U-boot Introduction
- Why U-Boot
- U-Boot bootflow
- U-Boot commands
- Boot scripts
- U-boot directory structure
- U-Boot code flow
- Building U-Boot
- Customization of U-Boot

What is Bootloader ?

- The boot loader is a software program and is responsible for **loading the operating system** into memory.



- It is typically stored in a specific location on the boot device.
- Multistage booting

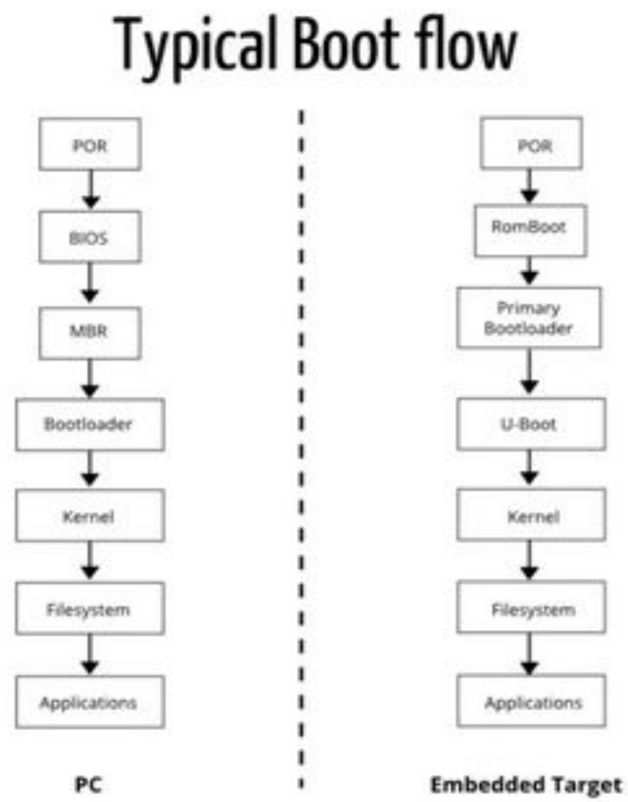
Bootloader Functions

1. Initializing the hardware, especially the memory controller

- The low-level initialization of the microprocessor, memory controllers, i/o ports,system bus (PCI, PCIe) and other board-specific hardware.
- Varies from board to board and CPU to CPU, it must be performed before an OS can execute.
- 2. Providing boot parameters for the kernel
- Like sets the initial root filesystem (root= parameter)
- Specifying the kernel's initial process (init= parameter), setting the system timezone (tz=), and specifying console behavior (console=).

3. loads the os into system memory

Bootloader Comparisons



Uboot Introduction

- U-boot is short for Universal-bootloader(also called as Das U-boot).
- U-Boot itself must be booted by the platform, and that must be done from a device that the platform's ROM is capable of booting from, which naturally depends on the platform.
- Used in **embedded devices** to perform various low-level hardware initialization tasks and boot the device's operating system kernel.
- Supports ~300 boards and ~13+ architectures, including M68000, ARM, Blackfin, MicroBlaze, IBM S360, My66, MOS 6502, ARM64, MIPS, Nios, SuperH, PPC, RISC-V and x86.
- Used as a default bootloader by several board vendors.

    Amazon Kindle & Kobo eReader devices use U-Boot as their bootloader.

    MIPS based wireless routers use U-Boot for bootloading.

    The PowerPC based series of AmigaOne computers running AmigaOS use U-Boot.

Why Uboot ?

Uboot Features

- **Open source and has vast community support.**
- **Hardware Compatibility**

  Supports variety of architectures.

  Supports variety of boards.

  Even supports variety of configurations.

- **Feature-Rich Functionality**

  Booting from various storage devices is possible(SD card, SATA drives,NOR flash, NAND flash, USB mass storage).

  Supports Network booting.

  Support for multiple filesystems (FAT, ext*, etc.),

- **Pre-boot Environment** : ***Testing*** and Debugging.

SoCtronics

Uboot Features

- **Customizable footprint**

U-Boot is highly customizable to provide a rich feature set that it supports multiple architectures and multiple configurations, with ***a small binary footprint.***

- **Command shell**

U-Boot has a command shell in which you work with U-Boot commands to create a customized boot process.

U-Boot has a set of built-in commands for booting the system, managing memory, and updating an embedded system's firmware.

Custom commands.

- **Environmental Variables & Boot scripts**
- Provides **secure booting** with public keys and digital signatures.

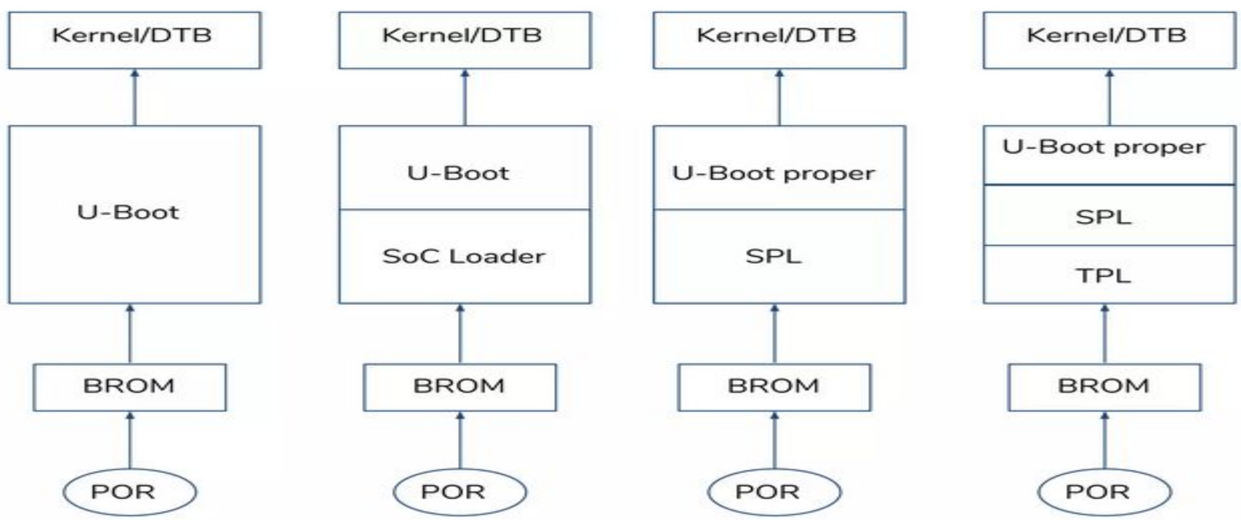
U-boot features

U-boot Supports Multi-stage booting

- U-Boot may be split into two stages: the platform would load a small SPL (Secondary Program Loader), which is a stripped-down version of U-Boot.
- Regardless of whether the SPL is used, U-Boot performs both

First-stage (e.g., configuring memory controllers and SDRAM)

Second-stage booting (performing multiple steps to load a modern operating system from a variety of devices that must be configured, presenting a menu for users to interact with and control the boot process, etc.).



U-boot features

Uboot supports device tree overlays

- This feature aims to make it possible for a single U-Boot binary to support multiple configurations.
- Allows dynamic hardware configuration based on attached peripherals (like HATs on Raspberry Pi).
- DTOs are valuable when working with hardware platforms that support multiple configurations or when integrating new hardware components into an existing system.
- This capability is particularly valuable during the evaluation phase of a project when multiple configurations are being tested
- Add the following line to your config.txt to apply the overlay:

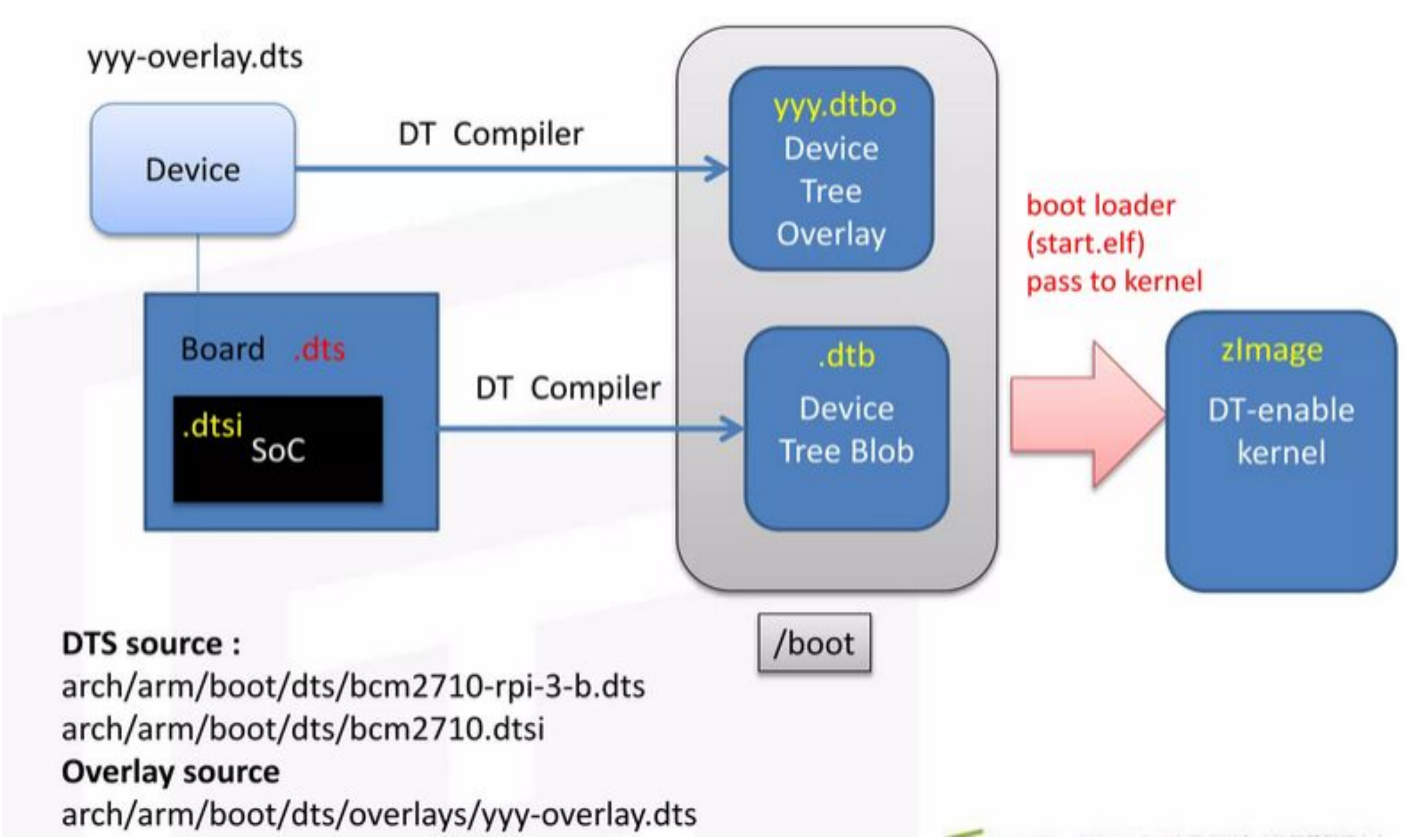
    dtoverlay=uart1-overlay

U-boot features

Device tree

- Describe the hardware layout, and how it works.
- For a given piece of HW, Device Tree should be the same for U-Boot, or Linux.
- There should be no need to change the Device Tree when updating the OS.
- The Device tree describes how the device/IP block is connected/integrated with the rest of the system: IRQ lines, DMA channels, clocks, reset lines, etc.
- Can be linked directly inside a bootloader binary (U-Boot, Barebox).
- Can be passed to the operating system by the bootloader (Linux).
- One or more nodes defining the buses on SoC and on-board devices.

U-boot features



U-boot features

Device tree

```
/ {
        compatible = "raspberrypi,3-model-b-plus", "brcm,bcm2837";
        model = "Raspberry Pi 3 Model B+";
        cpus: cpus {
                #address-cells = <1>;
                #size-cells = <0>;
                enable-method = "brcm,bcm2836-smp"; // for ARM 32-bit

                 cpu0: cpu@0 {
                        device_type = "cpu";
                        compatible = "arm,cortex-a53";
                        reg = <0>;
                        enable-method = "spin-table";
                        cpu-release-addr = <0x0 0x000000d8>;
                };
                cpu1:cpu@1{.................

        chosen {
                stdout-path = "serial1:115200n8";
        };

        memory@0 {
                device_type = "memory";
                reg = <0 0x40000000>;
        };
        i2c2: i2c@7e805000 {
                compatible = "brcm,bcm2835-i2c";
                reg = <0x7e805000 0x1000>;
                interrupts = <2 21>;
                clocks = <&clocks BCM2835_CLOCK_VPU>;
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";
        };................
```

U-boot features

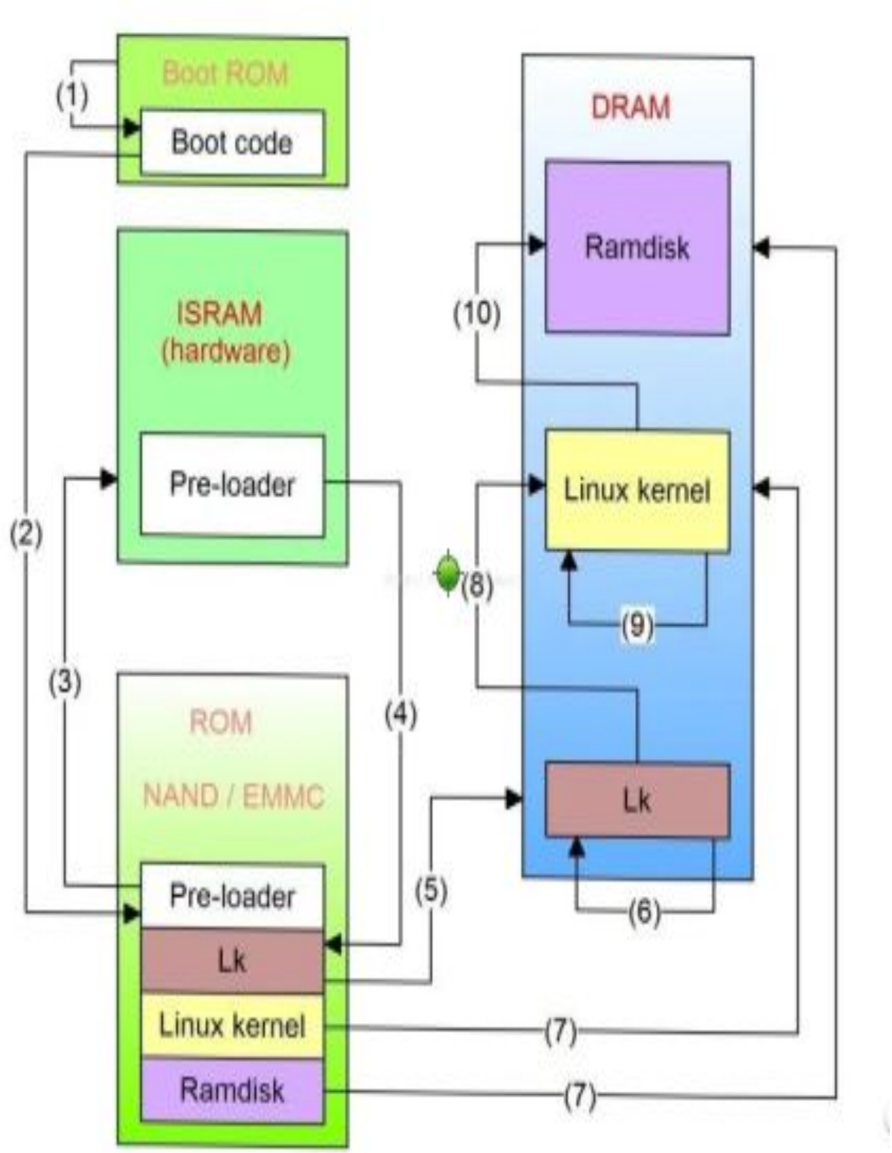
```
/dts-v1/;
/plugin/;

/ {
    compatible = "brcm,bcm2835";

    fragment@0 {
        target = <&uart1>;
        __overlay__ {
            status = "okay";
        };
    };

    fragment@1 {
        target-path = "/aliases";
        __overlay__ {
            serial1 = &uart1;
        };
    };
};
```

Uboot : Booting Process



1. Boot Rom code executes
2. searching boot device and locating primary boot loader(SPL)
3. loading SPL(Secondary program loader)
4. SPL locating uboot
5. SPL loading uboot into DRAM(main memory)
6. uboot starts executing, it relocates itself.
7. uboot loads kernel and ramdisk
8. uboot boots the kernel by passing the boot arguments
9. kernel starts executing
10. kernel mounts root file system

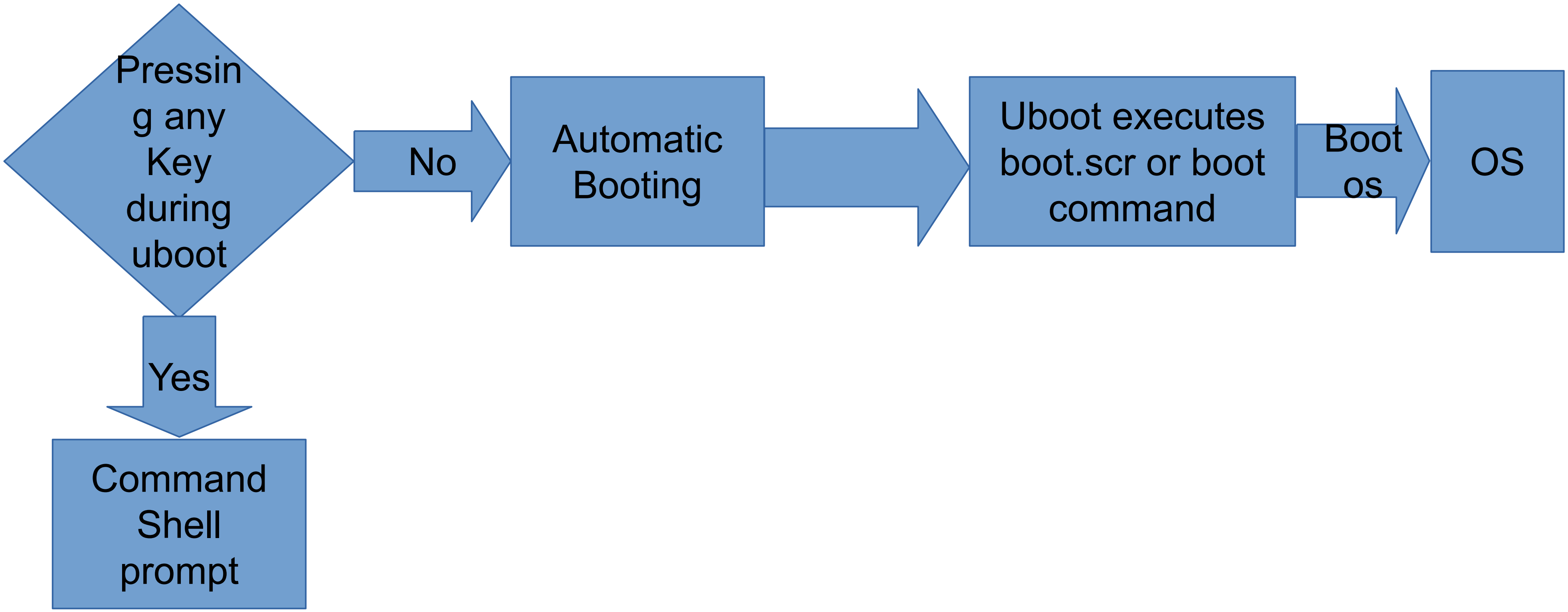
Uboot : Booting Process

```
U-Boot 2024.07-rc4-00012-g1ebd659cf0-dirty (Jun 13 2024 - 14:16:59 +0530)

DRAM:  948 MiB
RPI 3 Model B+ (0xa020d3)
Core:  86 devices, 13 uclasses, devicetree: board
MMC:   mmc@7e202000: 0, mmcnr@7e300000: 1
Loading Environment from FAT... Unable to read "uboot.env" from mmc0:1...
In:    serial,usbkbd
Out:   serial,vidconsole
Err:   serial,vidconsole
Net:   No ethernet found.

starting USB...
Bus usb@7e980000: USB DWC2
scanning bus usb@7e980000 for devices... 4 USB Device(s) found
       scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot:  0
U-Boot>
```

Uboot : Booting process



Uboot shell commands

- ‘help’ command

```
=> help
?         - alias for 'help'
bdinfo    - print Board Info structure
bootm     - boot application image from memory
cmp       - memory compare
coninfo - print console devices and information
```

- Running ‘help’ on a command
- Provide further details on that specific command

```
=> help bdinfo
bdinfo - print Board Info structure

Usage:
bdinfo
```

Uboot shell commands

- ‘bdinfo’ command

```
=> bdinfo
arch_number = 0x00000E05
boot_params = 0x80000100
DRAM bank   = 0x00000000
-> start    = 0x80000000
-> size     = 0x20000000
eth0name    = usb_ether
ethaddr     = 60:64:05:f4:79:7f
current eth = usb_ether
ip_addr     = 192.168.1.2
baudrate    = 115200 bps
TLB addr    = 0x9FFF0000
relocaddr   = 0x9FF44000
reloc off   = 0x1F744000
irq_sp      = 0x9DF23EC0
sp start    = 0x9DF23EB0
Early malloc usage: 2a8 / 400
```

Uboot shell commands

Uboot memory access commands

- Useful for reading or writing memory and registers – md,mw
- Support for byte/ word/ long/ quad
- md.b, md.l .....(default - md.l (32-bit))
- Support for reading multiple units at a time - 0x04

```
=> mw 0x81000000 0x1234abcd
=> md.l 0x81000000 0x8
81000000: 1234abcd 00000000 00000000 00000000 ..4.............
81000010: 00000000 00000000 00000000 00000000 ................
=> md.w 0x81000000 0x8
81000000: abcd 1234 0000 0000 0000 0000 0000 0000 ..4.............
=> md.b 0x81000000 0x8
81000000: cd ab 34 12 00 00 00 00                  ..4.....
```

Uboot shell commands

- Memory modification commands - mm
- Useful for interactively modifying registers
- ‘mm’ auto increments address
- Press ‘q’ to get back to u-boot shell
- Use ‘-’ to get previous address,
- Use ‘Enter’ to skip present address without value

```
=> mm 0x4804c134
4804c134: ffffffff ? fe1fffff
4804c138: f0002300 ?
4804c13c: 00000000 ? 00400000
4804c140: 00000000 ? q
=>
```

Uboot shell commands

- There are also commands to copy – cp, compare - cmp

```
=> mw 0x81000000 0x1234abcd 0x10
=> cp 0x81000000 0x82000000 0x8
=> cmp 0x81000000 0x82000000 0x8
Total of 8 word(s) were the same
=> cmp 0x81000000 0x82000000 0x9
word at 0x81000020 (0x1234abcd) != word at 0x82000020 \
    (0xea000003)
Total of 8 word(s) were the same
```

Uboot shell commands

‘GPIO’ commands

- Useful for toggling or sampling GPIOs
- gpio input – to set that pin as input pin
- gpio set – to set a gpio
- gpio clear , gpio toggle

```
=> gpio input 45
gpio: pin 45 (gpio 45) value is 1
=> echo $?
1
=> gpio set 53
gpio: pin 53 (gpio 53) value is 1
```

Uboot shell commands

Boot commands

- U-boot supports booting kernel from different type of images(i.e uImage, zImage,Image, .gzip)
- Bootm - Boots a Linux kernel image stored in memory, used for handling uImage.
- Bootz - Similar to bootm, but specifically for handling compressed zImage format kernels.
- Booti - The booti command is used to boot a Linux kernel in flat or compressed 'Image' format.
- Boot - Starts the boot process based on the bootcmd environment variable.
- Along with these images uboot also supports fit images.

Uboot shell commands

- dcache[on|off] , icache[on|off]
- eeprom read and eeprom write
- nand read, nand write, nand erase, nand markbad.
- Booting from different boot devices

  nboot, usbboot, tftpboot
- Reset – perform reset of cpu
- run - run commands in an environment variable
- **Memory test** - mtest {address} {bytes} {pattern}

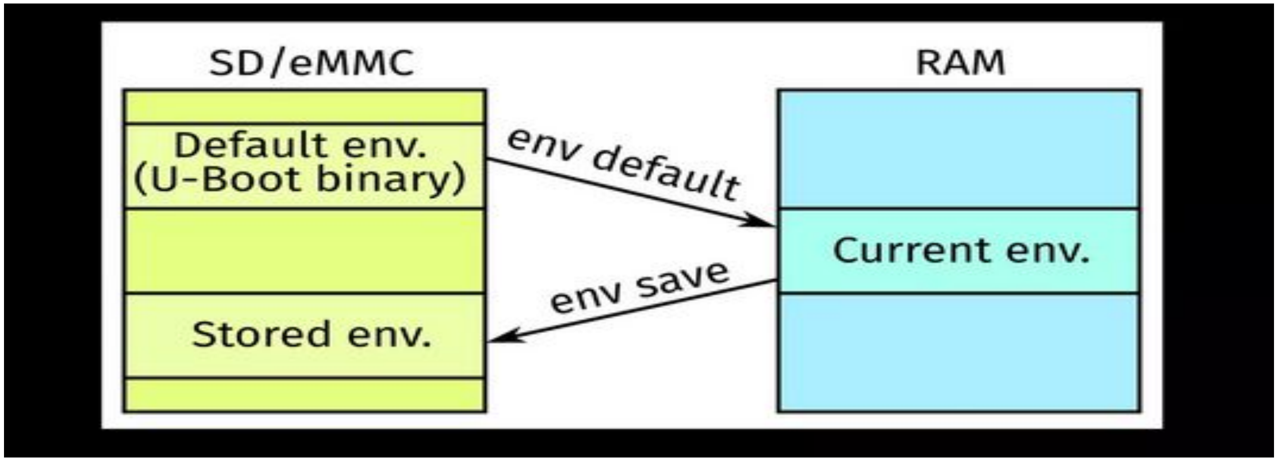
Custom commands

Adding and enabling a new command of your own

- 1. Write an API(function) that implements your command.
- 2. Register your function to the command list.
- 3. Enable your command.

Uboot – ENV Variables

- ## Environmental Variables :



- Uboot uses env variables to store configurations.
- Env var are used to control U-Boot behavior such as the boot targets, and timeout before auto-booting, as well as hardware data such as the baudrate, ethernet MAC address, ipaddr, tftpsrcp, tftpdtsp, tftpblocksize).
- Environment variables are stored in Non Volatile memory and loaded by uboot into ram during execution.

Uboot – ENV Variables

- Can create new variables using *setenv* and can store them in persistent memory using *saveenv*.
- Environment variables are stored as strings (**case sensitive**).
- The factory default variables and their values are stored in the U-Boot binary image itself. In this way, you can recover the variables and their values at any time with the envreset command.
- Custom variables can be created as long as there is enough space in the NVRAM.
- autostart – Yes -> boots the image by internally calling bootm

  No -> just loads and uncompress the image
- ver – stores current uboot version
- env default -a -- forcibly reset variables to their default values

Uboot – ENV Variables

- Executing environmental variables

```
#   setenv dumpaddr md.b \${addr} \${bytes}
#   printenv dumpaddr
dumpaddr=md.b ${addr} ${bytes}
#   setenv addr 2C000
#   setenv bytes 5
#   run dumpaddr
0002c000: 00 00 00 00 00     .....
```

- You must use back slash '\' before '$' so that the inner variables are not expanded in the new variable. Instead, they are expanded when the recursive variable(variable that contain one or more variables) is run as a command.
- Set bootcmd fatload mmc 0:1 \${kernel_addr_r} Image;fatload mmc 0:1 \${fdt_addr_r} \${fdtfile};bootm \${kernel_addr_r} \${fdt_addr_r}

Uboot – ENV Variables

- Some of these variables are set using *env_set* function in board files.
- Some are set in .env files.

```
U-Boot> fatload mmc 0:1 ${fdt_addr} bcm2710-rpi-3-b-plus.dtb
34228 bytes read in 4 ms (8.2 MiB/s)
U-Boot> printenv
arch=arm
baudrate=115200
board=rpi
board_name=3 Model B+
board_rev=0xD
board_rev_scheme=1
board_revision=0xA020D3
boot_targets=mmc usb pxe dhcp
bootargs=coherent_pool=1M 8250.nr_uarts=1 root=/dev/mmcblk0p2 initrd=/bin/sh
bootcmd=bootflow scan
bootdelay=20
cpu=armv8
dhcpuboot=usb start; dhcp u-boot.uimg; bootm
ethaddr=b8:27:eb:f9:b1:d9
fdt_addr=2eff7500
fdt_addr_r=0x02600000
fdt_high=ffffffffffffff
fdtcontroladdr=3af4a340
fdtfile=broadcom/bcm2837-rpi-3-b-plus.dtb
fileaddr=2eff7500
filesize=85b4
initrd_high=ffffffffffffffff
```

Uboot – ENV Variables

- The default environment for a board is created using a .env environment file using a simple text format. The base filename for this is defined by CONFIG_ENV_SOURCE_FILE, or CONFIG_SYS_BOARD if that is empty.

- Board file should be present in board / <vendor> / <board> / <board>.env

  Ex : **board / raspberry / rpi / rpi.env**

  board / ti / am64x / am64x.env

- Settings which are common to a group of boards can use #include to bring in a common file in the include/env directory.

  #include <env/ti/mmc.env>

Uboot – ENV Variables

```
/* SPDX-License-Identifier: GPL-2.0+ */

/* environment for Raspberry Pi boards */

dhcpuboot=usb start; dhcp u-boot.uimg; bootm

/* Environment */
stdin=serial,usbkbd
stdout=serial,vidconsole
stderr=serial,vidconsole

/* DFU over USB/UDC */
#ifdef CONFIG_CMD_DFU
dfu_alt_info=u-boot.bin fat 0 1;uboot.env fat 0 1;
  config.txt fat 0 1;
#ifdef CONFIG_ARM64
dfu_alt_info+=Image fat 0 1
#else
dfu_alt_info+=zImage fat 0 1
#endif
#endif  /* CONFIG_CMD_DFU */
```

Uboot : Command Shell Booting

Booting kernel by setting env variables and uboot commands

- Unlike PC bootloaders which automatically choose the memory locations of the kernel and other boot data, U-Boot requires its boot commands to explicitly specify the physical memory addresses as destinations for copying data (kernel, ramdisk, device tree, etc.)

```
=> setenv fdt_addr 0x43000000
=> setenv kernel_addr_r 0x47000000
=> setenv ramdisk_addr_r 0x48000000
=> setenv bootargs console=ttyS0,115200n8 root=/dev/sda1 waitforroot=3 rootfstype=ext4
```

```
=> ext4load mmc 0 ${fdt_addr} /boot/dtb/sun7i-a20-bananapro.dtb
29223 bytes read in 889 ms (31.3 KiB/s)
```

```
=> ext4load mmc 0 ${kernel_addr_r} /boot/zImage-armv7
4058096 bytes read in 304 ms (12.7 MiB/s)
```

```
=> ext4load mmc 0 ${ramdisk_addr_r} /boot/initrd-armv7
44930974 bytes read in 2258 ms (19 MiB/s)
```

Uboot : Command Shell Booting

```
=> bootz ${kernel_addr_r} ${ramdisk_addr_r}:${filesize} ${fdt_addr}
Kernel image @ 0x47000000 [ 0x000000 - 0x3debf0 ]
## Flattened Device Tree blob at 43000000
   Booting using the fdt blob at 0x43000000
   Loading Ramdisk to 47526000, end 49fff79e ... OK
   Loading Device Tree to 4751b000, end 47525226 ... OK

Starting kernel ...
```

- It's even possible to upgrade U-Boot using U-Boot, simply by reading the new bootloader from somewhere (local storage, or from the serial port or network) into memory, and writing that data to persistent storage where the bootloader belongs.

Boot Scripts

- Written using environmental variables.
- Stored in any storage device in boot partition.
- Automatically executed before the OS auto boot process.
- Use mkimage tools to create .scr files.
- Especially useful for production environment.

Boot Scripts

The bootscript works in the following way:

1. U-Boot checks the variable *loadbootsc*. If set to "no", gives uboot prompt.

2. If the variable *loadbootsc* is set to "yes" (factory default value) U-Boot tries to download the bootscript file with the filename stored in variable '*bootscript* '

The default value of the bootscript variable is <platformname>-bootscript.

3. If the bootscript file is successfully downloaded, it is executed.

4. If any of the commands in the bootscript fails, the rest of script is cancelled.

5. When the bootscript has been fully executed (or cancelled) U-Boot continues normal execution.

Boot Scripts

U-Boot commands can be put together in a text file and then the text files used to create a boot.scr. U-boot will look for the script in the root or /boot directory of the first partition on the storage device present.

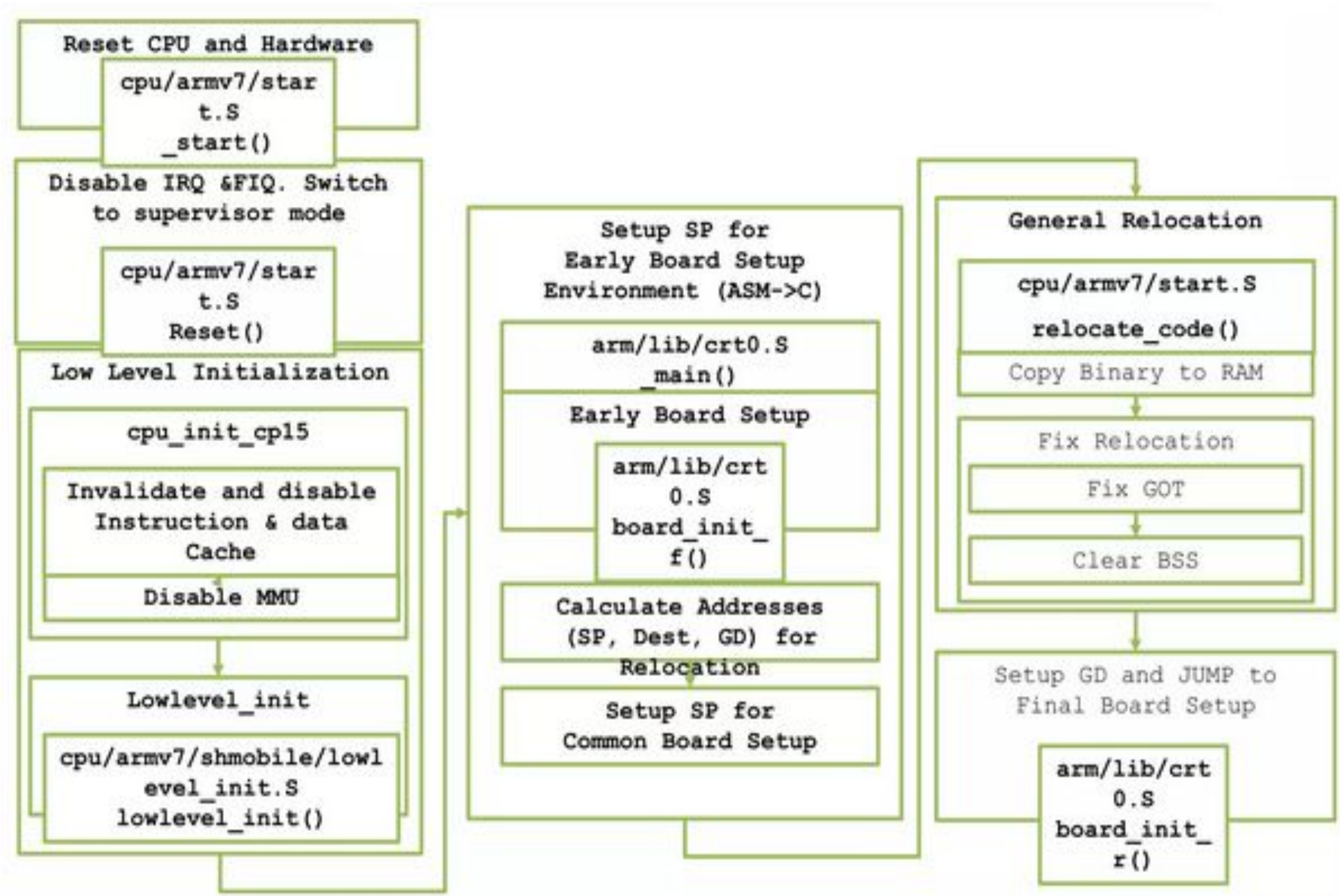
```
# sda-boot.cmd
#
setenv fdt_addr 0x43000000
setenv kernel_addr_r 0x47000000
setenv ramdisk_addr_r 0x48000000
#
setenv bootargs console=ttyS0,115200n8 root=/dev/sda1 waitforroot=3 rootfstype=ext4
#
ext4load scsi 0:1 ${fdt_addr} /boot/dtb/sun7i-a20-bananapro.dtb
ext4load scsi 0:1 ${kernel_addr_r} /boot/zImage-armv7
ext4load scsi 0:1 ${ramdisk_addr_r} /boot/initrd-armv7
#
bootz ${kernel_addr_r} ${ramdisk_addr_r}:${filesize} ${fdt_addr}
#
# Create boot.scr with:
# mkimage -C none -A arm -T script -d sda-boot.cmd boot.scr
```

U-boot directory structure

```
u-boot
    |-- api
    |-- arch
    |-- board
    |-- common
    |-- disk
    |-- doc
    |-- drivers
    |-- examples
    |-- fs
    |-- include
    |-- lib
    |-- mmc_spl
    |-- nand_spl
    |-- net
    |-- onenand_ipl
    |-- post
    |-- spl
    `-- tools
```

- api
  - API for standalone applications.
- arch
  - Architecture and SoC related basics.
- board
  - Board setup and configuration relatives.
- common
  - Commands and some middleware.
- drivers
  - Middleware and APIs for peripherals, as well as device drivers are included.
- fs
  - Supported various filesystems.
- include:
  - Board configuration files and common include headers.
- lib
  - Common libraries such as sorting, compress, crc, hashtable, etc.
- net
  - Protocol stack of network library and network APIs.
- tools:
  - Miscellaneous supporting tools.

Uboot code flow



Building uboot

- clone u-boot git repository

    git clone https://github.com/u-boot/u-boot –depth=1

Dependencies

- For building U-Boot you need a GCC or Clang compiler for your host platform.
- If you are not building on the target platform you further need a cross compiler.
- On Debian based systems the cross compiler packages are named *gcc-<architecture>-linux-gnu.*
- You could install GCC and the GCC cross compiler for the ARMv8 architecture with

    sudo apt-get install gcc gcc-aarch64-linux-gnu

Building uboot

Configuration

- Directory configs/ contains the template configuration files for the maintained boards following the naming scheme.

  <board>_defconfig

- For instance the configuration template for the raspberry pi 3B+ board is called rpi_3_defconfig. The corresponding .config file is generated by

  make ARCH=*arm* CROSS_COMPILE=*aarch64-linux-gnu- rpi_3_defconfig*

- You can adjust the configuration using

  make menuconfig

Building uboot

Building u-boot image

- When cross compiling you will have to specify the prefix of the cross-compiler. You can either specify the value of the CROSS_COMPILE variable on the make command line or export it beforehand.

  make CROSS_COMPILE=*<compiler-prefix>*

- Assuming cross compiling on Debian for ARMv8 this would be

  make -j$(nproc) ARCH=*arm* CROSS_COMPILE=*aarch64-linux-gnu-*

- U-boot.bin is created.
- Building spl,

  make -j$(nproc) SPL ARCH=*arm* CROSS_COMPILE=*aarch64-linux-gnu-*

Uboot porting

Porting uboot to a new board

- Create a board directory board/<vendor>/<board>.
- Create <board>.c files to add board specific functions, add <board>.env file for your board.
- Create board Kconfig file board/my_vendor/my_board/Kconfig

```
if TARGET_RPI_3
    config SYS_BOARD
        default "my_board"
    config SYS_VENDOR
        default "my_vendor"
    config SYS_CONFIG_NAME
        default "my_board"
endi
```

- SYS_VENDOR and SYS_BOARD are used to identify the directory where make find the files it needs to compile.

Uboot porting

- SYS_CONFIG_NAME is used to identify the board header file include/configs/SYS_CONFIG_NAME.h
- Create your board makefile and add your board files you created.

  Ex: board/my_vendor/my_board/Makefile

  obj-y := my_board.o
- Add board header file in include/configs/<board>.h
- Create defconfig file in configs/ directory

  Ex: configs/rpi_3_b_defconfig
- Put here anything that is selectable in Kconfig (menuconfig) i.e drivers, features, U-Boot behaviour etc.

U-boot porting

- Source Kconfig file in arch/ directory

  Ex: arch/arm/Kconfig or arch/arm/mach-bcm283x/Kconfig

```
source "board/raspberrypi/rpi/Kconfig"
```

- Define board’s Target Kconfig option in arch/arm/mach-bcm/rpi/Kconfig

```
config TARGET_RPI_3
        bool “Raspberry pi 3 64 bit build”
        select BCM2837_64B
```

U-boot porting

- Source Kconfig file in arch/ directory

  Ex: arch/arm/Kconfig or arch/arm/mach-bcm283x/Kconfig

```
source "board/raspberrypi/rpi/Kconfig"
```

- Define board’s Target Kconfig option in arch/arm/mach-bcm/rpi/Kconfig

```
config TARGET_RPI_3
        bool “Raspberry pi 3 64 bit build”
        select BCM2837_64B
```

U-boot porting

- Source Kconfig file in arch/ directory

    Ex: arch/arm/Kconfig or arch/arm/mach-bcm283x/Kconfig

```
source "board/raspberrypi/rpi/Kconfig"
```

- Define board’s Target Kconfig option in arch/arm/mach-bcm/rpi/Kconfig

```
config TARGET_RPI_3
        bool “Raspberry pi 3 64 bit build”
        select BCM2837_64B
```

References

- https://hub.digi.com/dp/path=/support/asset/u-boot-reference-manual/
- https://docs.u-boot.org/en/latest/usage/index.html
- https://elinux.org/RPi_U-Boot#boot.scr.uimg
- https://labs.dese.iisc.ac.in/embeddedlab/device-tree-and-boot-flow/
- https://source.android.com/docs/core/ota/ab

THANK YOU