Table of Contents

1. What is boot loader ?....2
2. u-boot features....2
3. What is firmware....2
4. How to download u-boot, what is the version?....2
5. u-boot tree structure....2
5. u-boot -NXP – I.MX6 board....4
6. What are the boot Boot Modes -I.MX6....6
7. What is Internal Boot Mode ?....6
8. Why Do we need u-boot start from 1K (0x0400) ?....6
9.External Boot Mode....7
10.The process of booting an i.MX6 involves multiple steps:....7
11. u-boot -TI - OMAP3 board....8
12.Boot sequence (in order)....9
13.MLO....10
14.U-BOOT BOOT Sequence....10
15.Board Initialization Flow....11

1. What is boot loader ?

U-Boot is both a first-stage and second-stage boot loader. It is loaded by the system's ROM from a supported boot device, such as an SD card, SATA drive, NOR flash (e.g. using SPI or I^2C), or NAND flash. If there are size constraints, U-Boot may be split into stages: the platform would load a small SPL (Secondary Program Loader), which is a stripped-d own version of U-Boot, and the SPL would do initial hardware configuration and load the larger, fully featured version of U-Boot

2. u-boot features

U-Boot runs a command-line interface on a serial port. Using the console, users can load and boot a kernel, possibly changing parameters from the default. There are also commands to read device information, read and write flash memory, download files (kernels, boot images, etc.) from the serial port or network, manipulate device trees, and work with environment variables (which can be written to persistent storage, and are used to control U-Boot behavior such as the default boot command and timeout before auto-booting, as well as hardware data such as the Ethernet MAC address).

3. What is firmware

1. It is assembly code & C code
2. It is startup code – normally ex: FSBL , SSBL, U-BOOT
3. It will run in while(1)
4. There is no- scheduler – memory management – virtual memory
5. There is No MMU – page table concept – in u-boot

4. How to download u-boot, what is the version?

git clone git://git.denx.de/u-boot.git

u-boot version: 2019.10 , VERSION = 2019 PATCHLEVEL = 10

5. u-boot tree structure

tree -L 1

.
├── api
├── **arch**
├── **board**
├── **cmd**
├── **common**
├── config.mk
├── **configs**
├── disk
├── doc
├── Documentation
├── **drivers**
├── dts
├── env
├── examples
├── **fs**
├── i**nclude**
├── **Kbuild**
├── **Kconfig**
├── **lib**
├── Licenses
├── MAINTAINERS
├── **Makefile**
├── net
├── post
├── README
├── scripts
├── tags
├── test
└── tools

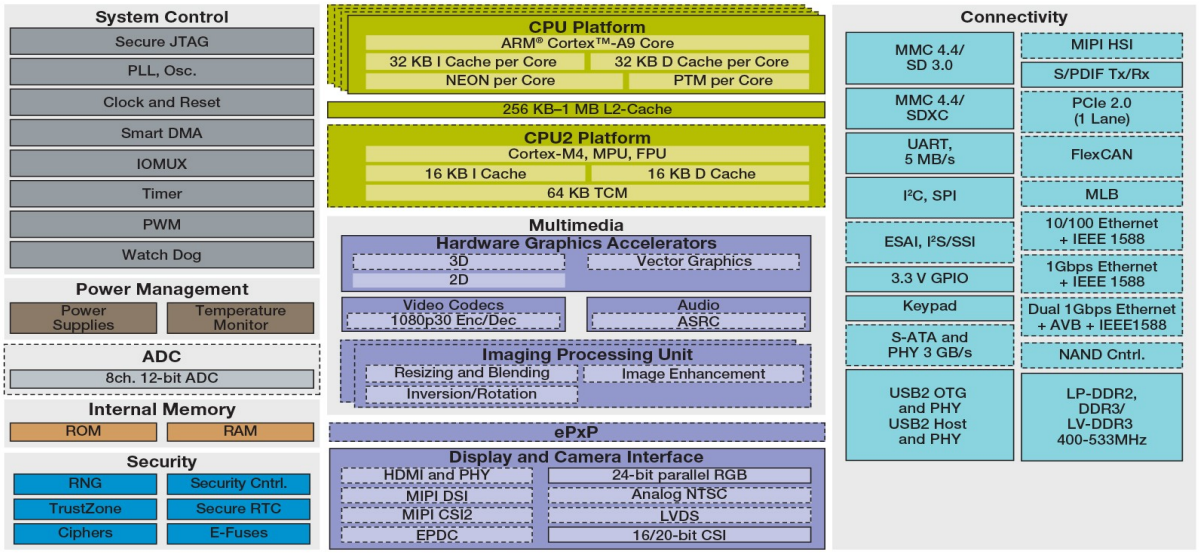
Directory Hierarchy:
==========

/arch	**Architecture specific files**
/arc	Files generic to ARC architecture
/arm	**Files generic to ARM architecture**
/m68k	Files generic to m68k architecture
/microblaze	Files generic to microblaze architecture
/mips	Files generic to MIPS architecture
/nds32	Files generic to NDS32 architecture
/nios2	Files generic to Altera NIOS2 architecture
/openrisc	Files generic to OpenRISC architecture
/powerpc	Files generic to PowerPC architecture
/riscv	Files generic to RISC-V architecture
/sandbox	Files generic to HW-independent "sandbox"
/sh	Files generic to SH architecture
/x86	Files generic to x86 architecture

/api	Machine/arch independent API for external apps
/board	**Board dependent files**
/cmd	**U-Boot commands functions**
/common	**Misc architecture independent functions**
/configs	**Board default configuration files**
/disk	**Code for disk drive partition handling**
/doc	**Documentation (don't expect too much)**
/drivers	**Commonly used device drivers**
/dts	**Contains Makefile for building internal U-Boot fdt.**
/examples	**Example code for standalone applications, etc.**
/fs	**Filesystem code (cramfs, ext2, jffs2, etc.)**
/include	**Header Files**
/lib	**Library routines generic to all architectures**
/Licenses	**Various license files**
/net	**Networking code**
/post	**Power On Self Test**
/scripts	**Various build scripts and Makefiles**
/test	**Various unit test files**
/tools	**Tools to build S-Record or U-Boot images, etc.**

5. u-boot -NXP – I.MX6 board

i.MX 6 Series Applications Processor Block Diagram



ARCH	ARM	u-boot-master/arch/arm
CPU	ARMV7 , CORTEX-A9	u-boot-master/arch/arm/cpu/armv7
SOC – FAMILY (**manufacturer)**	I.MX	u-boot-master/arch/arm/mach-imx/
SOC – Specific - family	I.MX6	u-boot-master/arch/arm/mach-imx/mx6
BOARD – SOC - Manufacturer	freescale	u-boot-master/board/freescale
BOARD - VENDOR	Sabre-sd	u-boot-master/board/freescale/mx6sabresd/
Default-Configuration File	Sabre-sd	u-boot-master/configs/mx6sabresd_defconfig
Macro - Definition	Sabre-sd	U-boot-master/include/configs/mx6sabresd.h
Supporting – Common-Macro- Definition	Sabre-sd	U-boot-master/include/configs/imx6_spl.h U-boot-master/include/configs/mx6sabre_common.h U-boot-master/include/configs/mx6_common.h

U-BOOT : Configuration	make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- mx6sabresd_defconfig
Build	make all
SD BOOT	`sudo dd if=u-boot.imx of=/dev/mmcblk0 bs=512 seek=2 conv=fsync`

6. What are the boot Boot Modes -I.MX6

SoCs can be configured for internal or external Boot Mode with the internal boot mode being the more popular mode.

7. What is Internal Boot Mode ?

The Internal Boot Mode is supported on:

- i.MX25
- i.MX35
- i.MX50
- i.MX51
- i.MX53
- i.MX6
- i.MX7
- i.MX8MQ

With the Internal Boot Mode, the images contain a header which describes where the binary shall be loaded and started. These headers also contain a so-called DCD table which consists of register/value pairs. These are executed by the Boot ROM and are used to configure the SDRAM. In barebox, the i.MX images are generated with the **scripts/imx/imx-image tool.** Normally it’s not necessary to call this tool manually, it is executed automatically at the end of the build process

8. Why Do we need u-boot start from 1K (0x0400) ?

This requirement comes from i.MX6. The i.MX6 will look at the address 1024 to load the data into its ROM and execute from there. Notice that at this point we have not created any partition whatsoever and the board will still boot. The u-boot does not care about the Master Boot

Record. Though we will need some kid of partition information information to read file system later in the boot process.

9.External Boot Mode

The External Boot Mode is supported by the older i.MX SoCs:

- i.MX1
- i.MX21
- i.MX27
- i.MX31
- i.MX35

The External Boot Mode supports booting only from NOR and NAND flash. On NOR flash, the binary is started directly on its physical address in memory. Booting from NAND flash is more complicated. The NAND flash controller copies the first 2kb of the image to the NAND Controller's internal SRAM. This initial binary portion then has to:

- Set up the SDRAM
- Copy the initial binary to SDRAM to make the internal SRAM in the NAND flash controller free for use for the controller
- Copy the whole barebox image to SDRAM
- Start the image

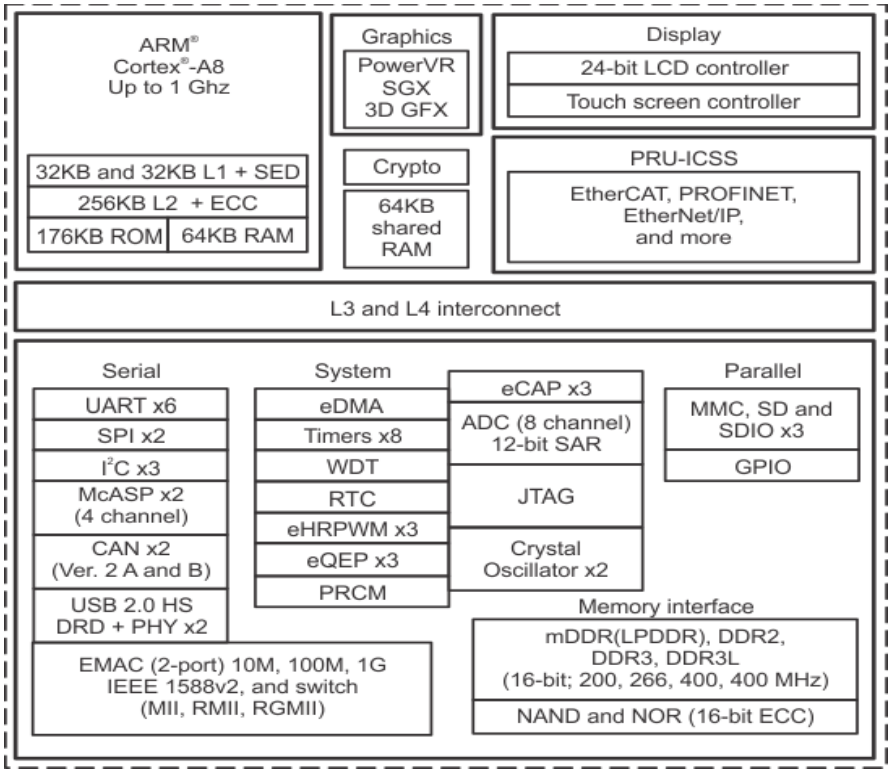
It is possible to write the image directly to NAND. However, since NAND flash can have bad blocks which must be skipped during writing the image and also by the initial loader, it is recommended to use the barebox_update - update barebox to persistent media command for writing to NAND flash.

10.The process of booting an i.MX6 involves multiple steps:

1. At power-on or reset, the processor starts executing code from on-chip ROM. This code reads the *Boot Mode* switches, and if they're in the normal position (*Boot from fuses*), goes on to read the fuses to find out which media to search for a bootable image.
   In our case, it will see that we have the fuses programmed for serial EEPROM, and should find a U-Boot image at offset 0x400 in that device.
2. The code in on-chip ROM will read the U-Boot preamble, configuring registers for DDR, then

copy U-Boot into DDR and begin U-Boot execution.

11. u-boot -TI - OMAP3 board



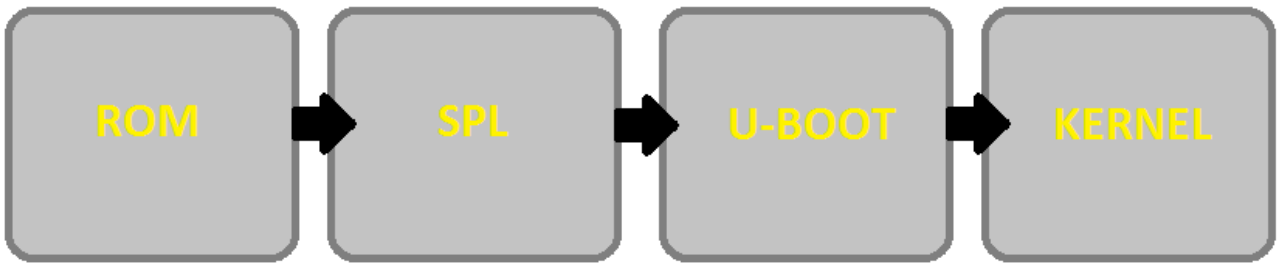
ARCH	ARM	u-boot-master/arch/arm
CPU	ARMV7 , CORTEX-A8	u-boot-master/arch/arm/cpu/armv7
SOC – FAMILY (**manufacturer)**	OMAP2	u-boot-master/arch/arm/mach-omap2/
SOC – Specific - family	OMAP3	u-boot-master/arch/arm/mach-omap2/omap3
BOARD – SOC - Manufacturer	TI	u-boot-master/board/ti
BOARD - VENDOR	evm	u-boot-master/board/ti/evm

Default-Configuration File	evm	u-boot-master/configs/omap3_evm_defconfig
Macro - Definition	evm	U-boot-master/include/configs/omap3_evm.h
Supporting – Common-Macro- Definition	Sabre-sd	U-boot-master/include/configs/ti_omap3_common.h U-boot-master/include/configs/ti_armv7_omap.h U-boot-master/include/configs/ti_armv7_common.h

12.Boot sequence (in order)

- Boot ROM
- X-loader
- U-boot

At power-up an OMAP3 device begins booting from internal Boot ROM. This code is fixed during the manufacturing process and cannot be altered. The Boot ROM reads boot configuration pins (SW4 on the OMAP3 EVM) which tell the Boot ROM where to look for the first external bootloader. The choices include NAND, UART, and SD/MMC Card. Control is then passed to this first external bootloader called x-loader. The x-loader application is included in the Linux PSP provided by TI and can be modified by the end user. The x-loader application passes control to u-boot. U-boot is also a bootloader and is considered the second external bootloader in this case.

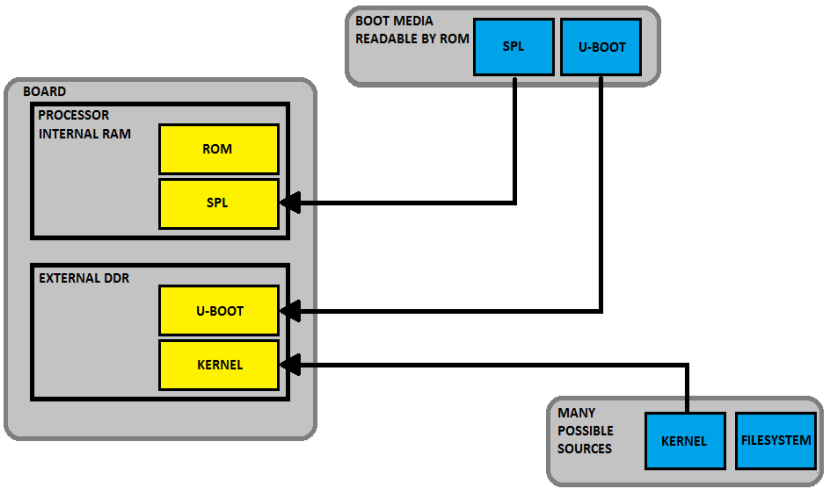


but has limited internal RAM (128 kB). Because of this limited amount of RAM, multiple bootloader stages are needed. These boot loader stages systematically unlock the full functionality of the device so that all complexities of the device are available to the kernel.

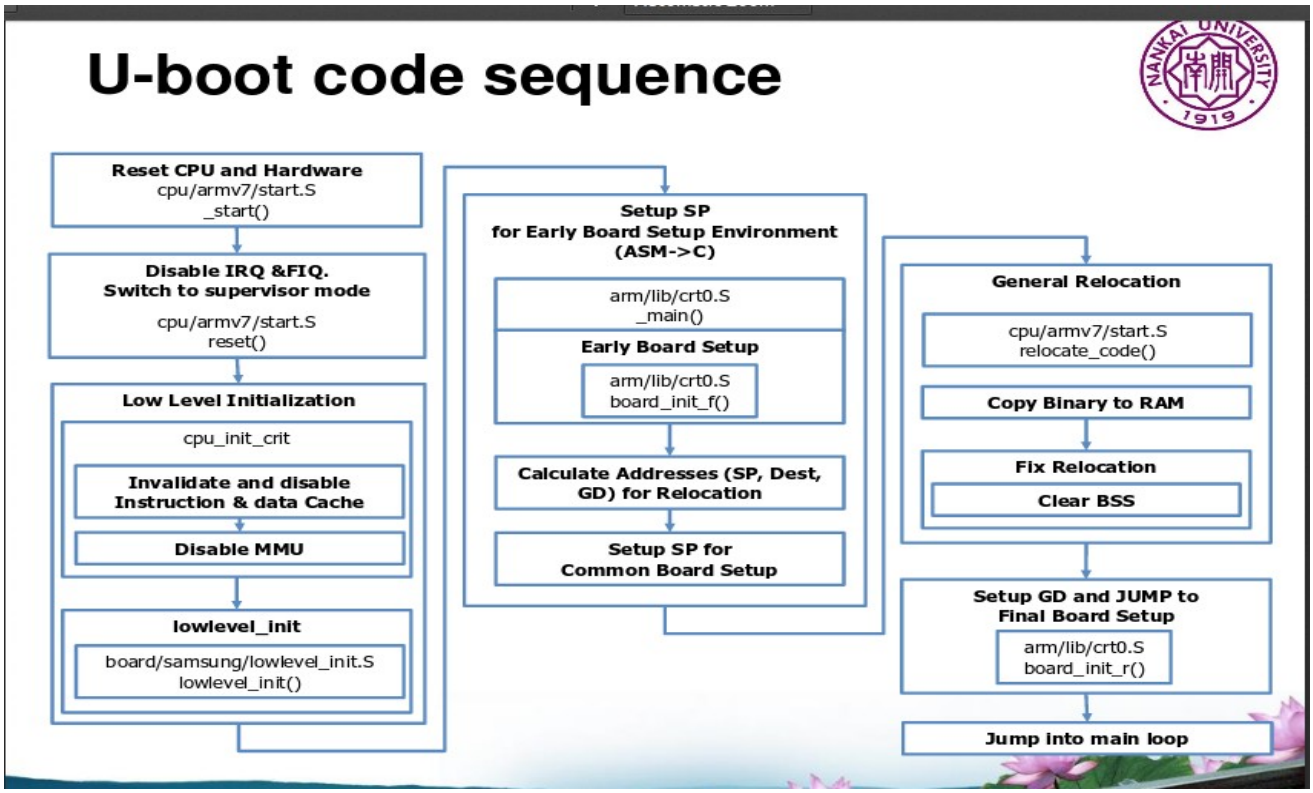
The SYSBOOT pins configure the boot device order (set by SYSBOOT[4:0]) for MMC,SPI0, UART0, USB0

13.MLO

The second stage boot loader is known as the SPL, but is sometimes referred to as the MLO. The SPL is the first stage of U-boot, and must be loaded from one of the boot sources into internal RAM. The SPL has very limited configuration or user interaction, and mainly serves to set-up the boot process for the next boot loader stage: U-boot.



14.U-BOOT BOOT Sequence



15.Board Initialization Flow

This is the intended start-up flow for boards. This should apply for both SPL and U-Boot proper (i.e. they both follow the same rules).

At present, SPL mostly uses a separate code path, but the function names and roles of each function are the same. Some boards or architectures may not conform to this. At least most ARM boards which use

CONFIG_SPL_FRAMEWORK conform to this.

Execution typically starts with an architecture-specific (and possibly CPU-specific) **start.S f**ile, such as:

- arch/arm/cpu/armv7/start.S

and so on. From there, three functions are called; the purpose and limitations of each of these functions are described below.

lowlevel_init():

- purpose: essential init to permit execution to reach **board_init_f()**
- no global_data or BSS
- there is no stack (ARMv7 may have one but it will soon be removed)
- must not set up SDRAM or use console
- must only do the bare minimum to allow execution to continue to **board_init_f()**
- this is almost never needed
- return normally from this function

board_init_f():

- purpose: set up the machine ready for r**unning board_init_r(): i.e. SDRAM and serial UART**
- global_data is available
- stack is in SRAM
- BSS is not available, so you cannot use global/static variables,

    only stack variables and global_data

Non-SPL-specific notes:

- dram_init() is called to set up DRAM. If already done in SPL this can do nothing

SPL-specific notes:

- you can override the entire board_init_f() function with your own version as needed.
- preloader_console_init() can be called here in extremis
- should set up SDRAM, and anything needed to make the UART work
- these is no need to clear BSS, it will be done by crt0.S
- must return normally from this function (don't call board_init_r()

directly)

board_init_r():

- purpose: main execution, common code
- global_data is available
- SDRAM is available
- BSS is available, all static/global variables can be used
- execution eventually continues to main_loop()

Non-SPL-specific notes:

- U-Boot is relocated to the top of memory and is now running from
  there.

SPL-specific notes:

- stack is optionally in SDRAM, if CONFIG_SPL_STACK_R is defined and
  CONFIG_SPL_STACK_R_ADDR points into SDRAM
- preloader_console_init() can be called here - typically this is
  done by selecting CONFIG_SPL_BOARD_INIT and then supplying a
  spl_board_init() function containing this call
- loads U-Boot or (in falcon mode) Linux