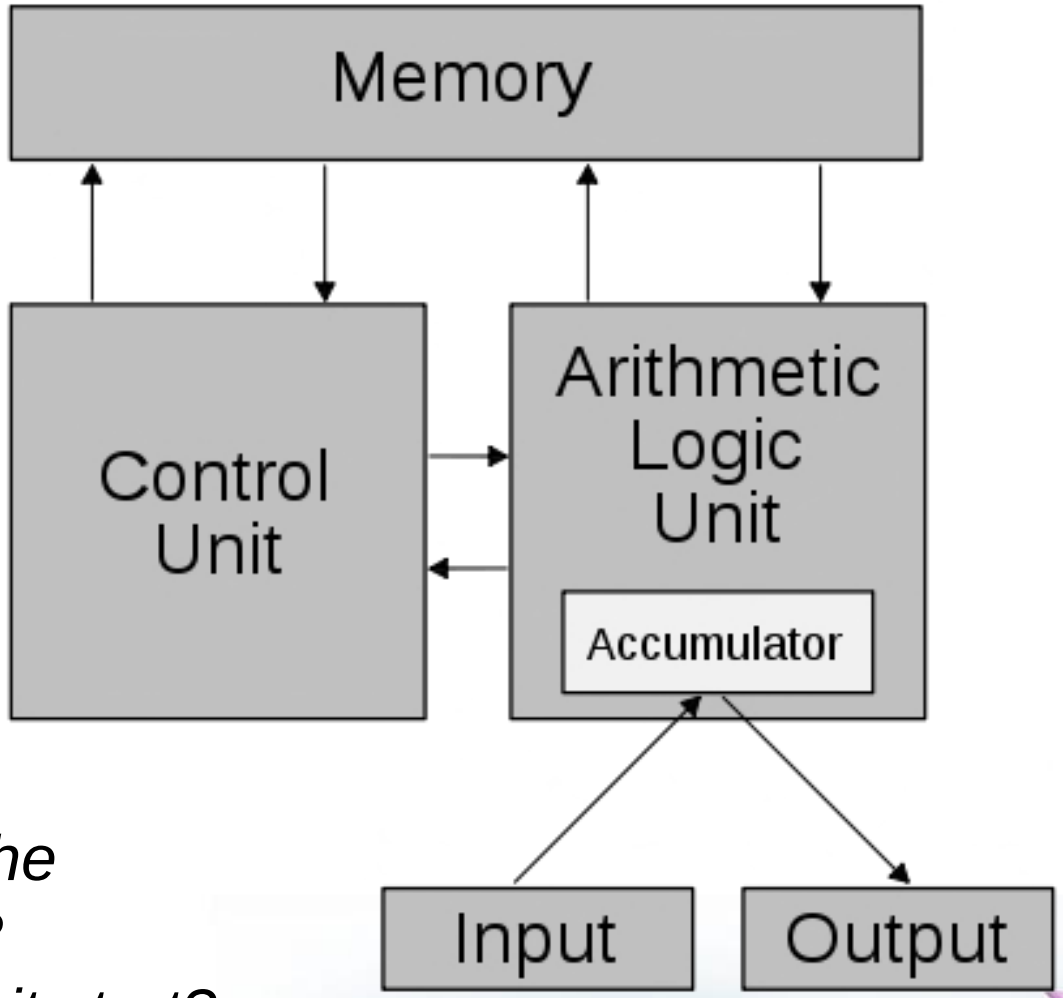
How system boot-up

Xiaoli Gong

gongxiaoli@nankai.edu.cn

Nankai University, China

System Architecture



How to get the instructions?
Where does it start?

Memory vs Storage

Memory vs Storage

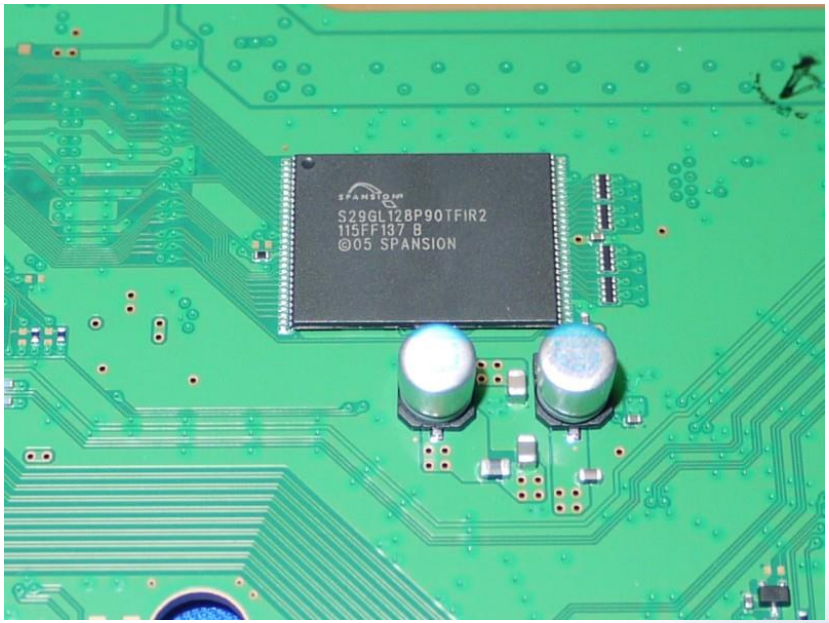
All of them are called 存储 in Chinese.
How to access them?

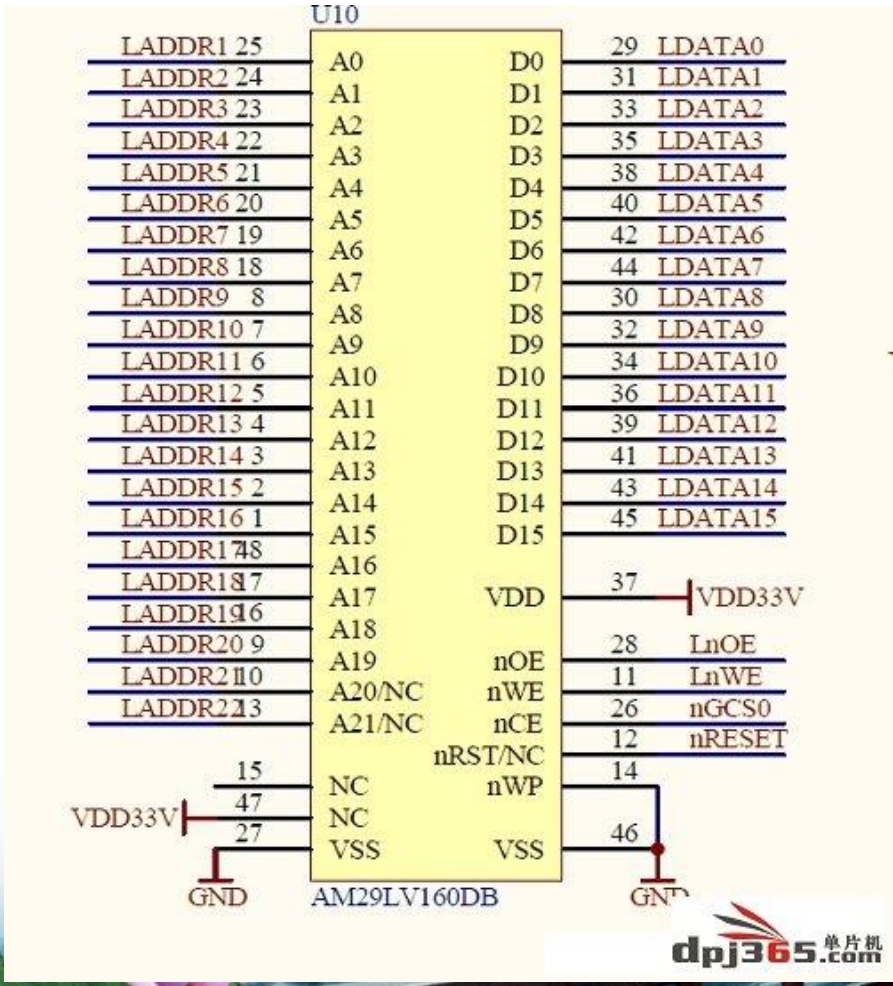
A specialized Storage device

- Can keep the data without battery
- Can be read as RAM
- Set to reset address
- Our option:
  - ROM
  - Eeprom
  - Nor Flash

However……

- Nor flash are expensive and difficult to deploy.

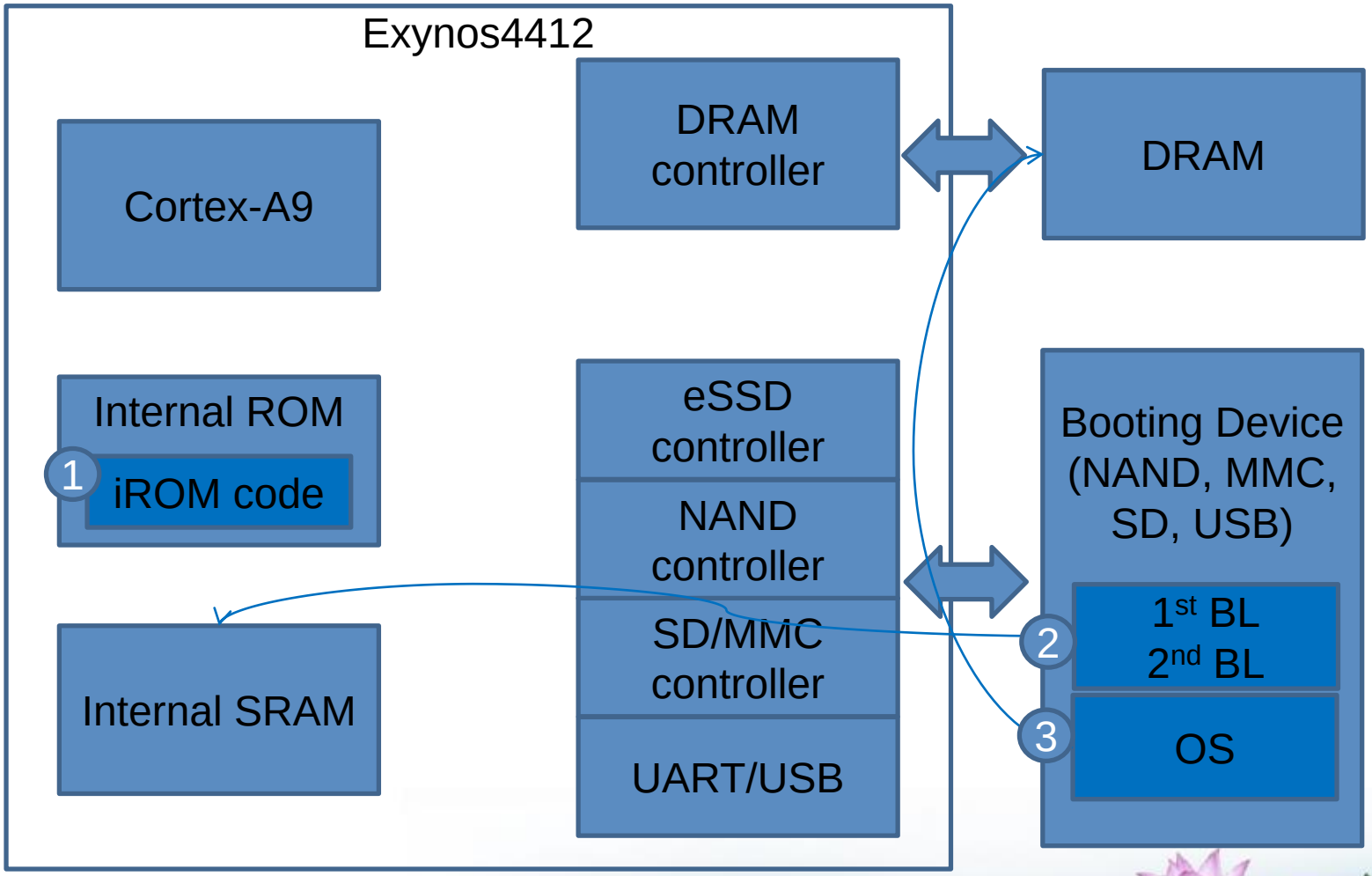




New solutions

- spi flash, nand flash, sdcard, USB storage…..
- We need a Stepstone.
- We need some helper program.

U-boot loading process

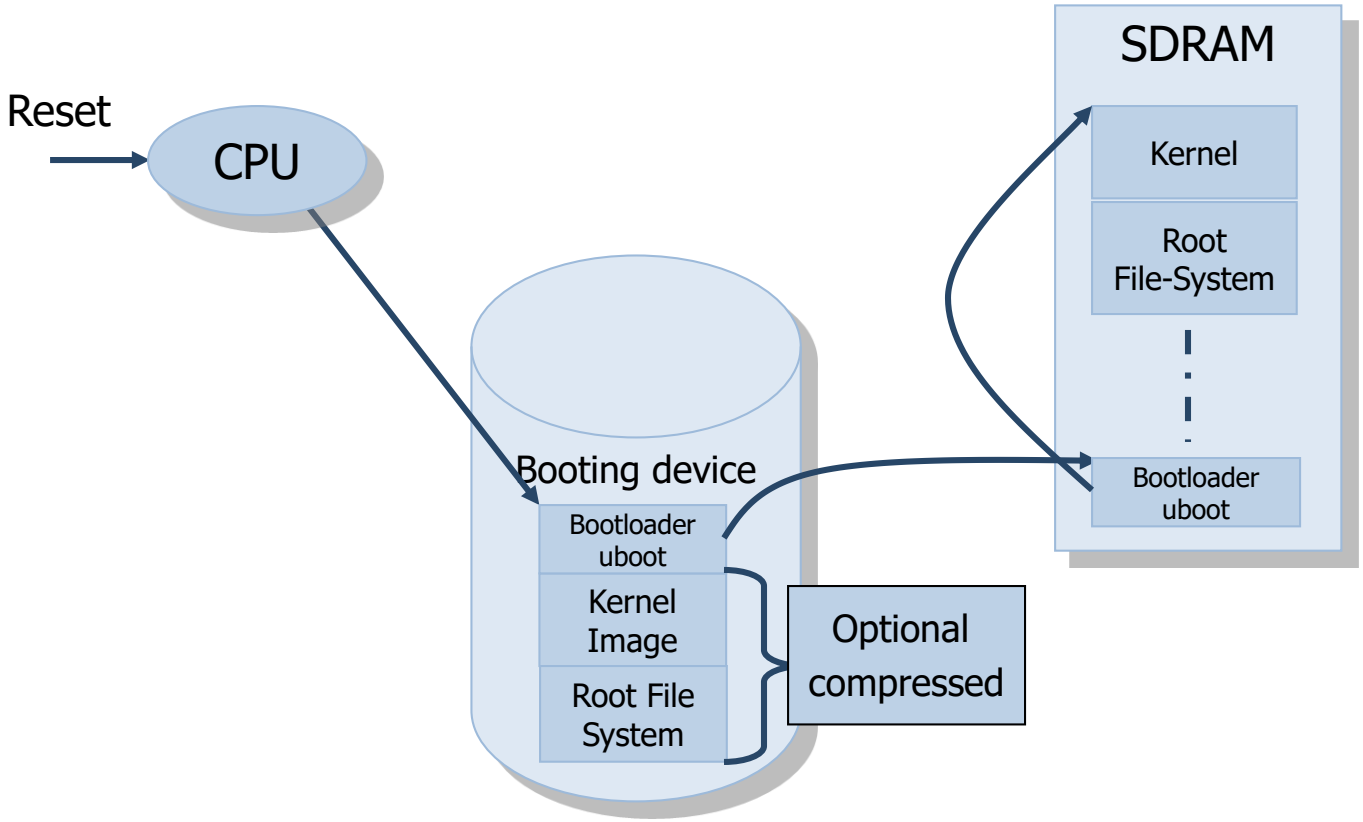


What we do when boot up

- Disable watchdog
- Initialize LED and/or UART
- Initialize DRAM
- Copy to DRAM (Optional)
- Initialize stack
- Get into stage-2

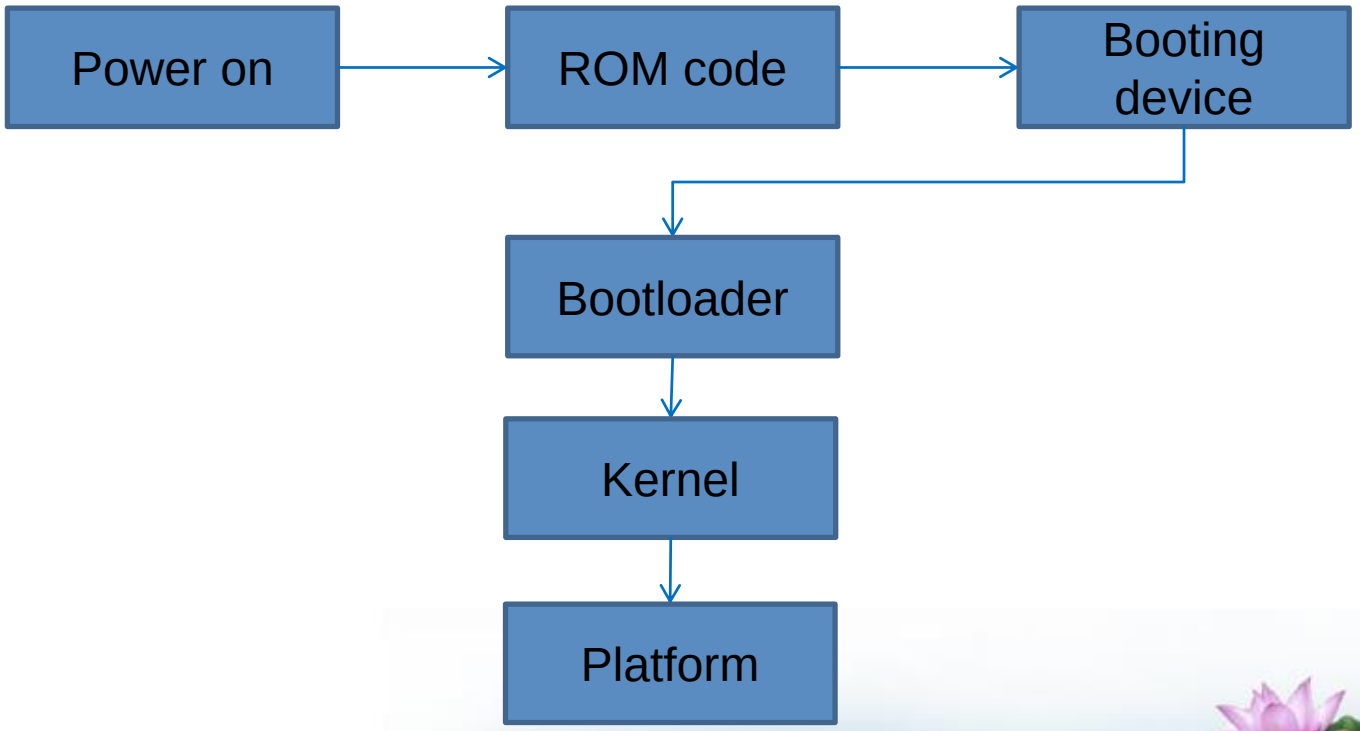
Boot-up program of Tizen

Loading through bootloader



System bootup overview

- **Bootloader**
  - **Memory Init**
  - **Pass Machine ID and the boot arguments to kernel**



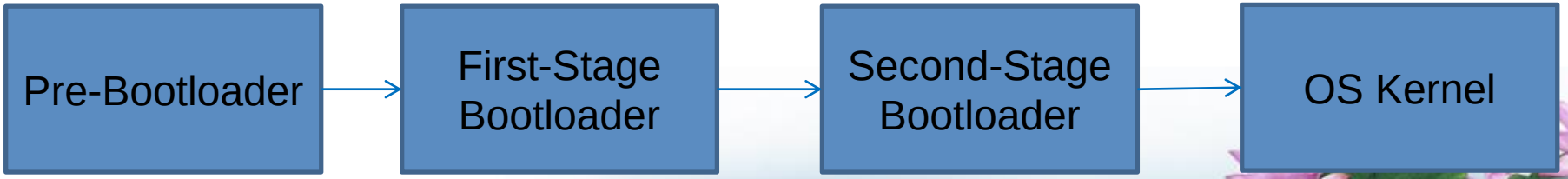
What is bootloader

- **What is boot loader ?**
  - **A boot loader is the first program which executes (before the main program i.e. kernel) whenever a system is initialized**
  - **A boot loader is a computer program that loads the main program (i.e. operating system, kernel) for the board after completion of the self-tests**

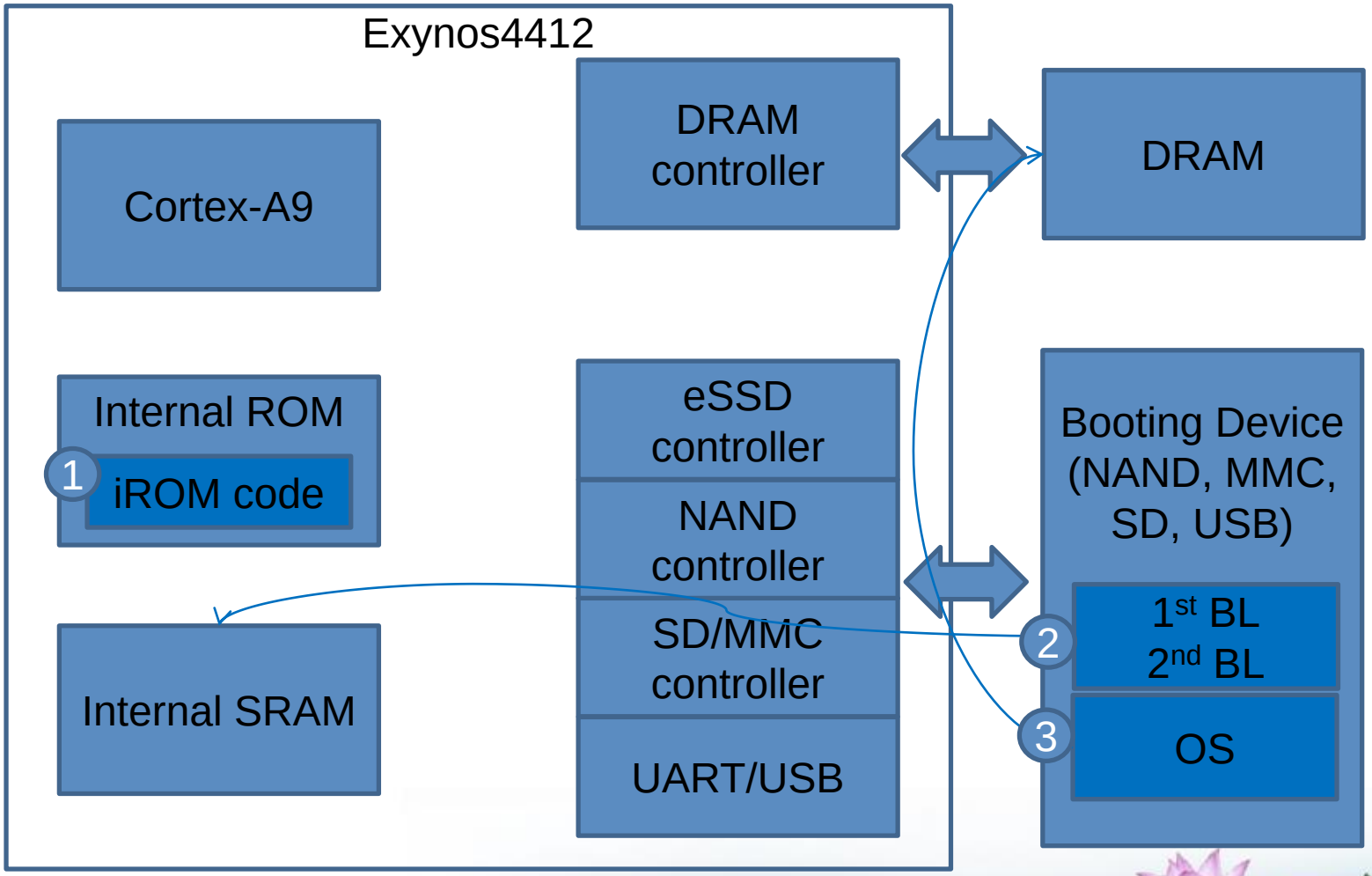
- **Why boot loader is needed ?**
  - **Gives a development environment**
  - **Saves cost of flash writers**
  - **Setup the basic running environment for operating system**
  - **Give flexibility to load any operating system**

Steps of bootloader

- What are different types of boot loaders ?
  - Boot-ROM (or Pre-Boot Loader)
    - Small code which loads First stage boot loader
  - First Stage Boot Loader
    - Small Piece of code that initialize the NAND/MMC & DRAM controller.
  - Second Stage Boot Loader
    - Primary function of the second-stage boot loader is to Loading the kernel into RAM or jumping directly to the start of the kernel.



U-boot loading process



U-boot loading process

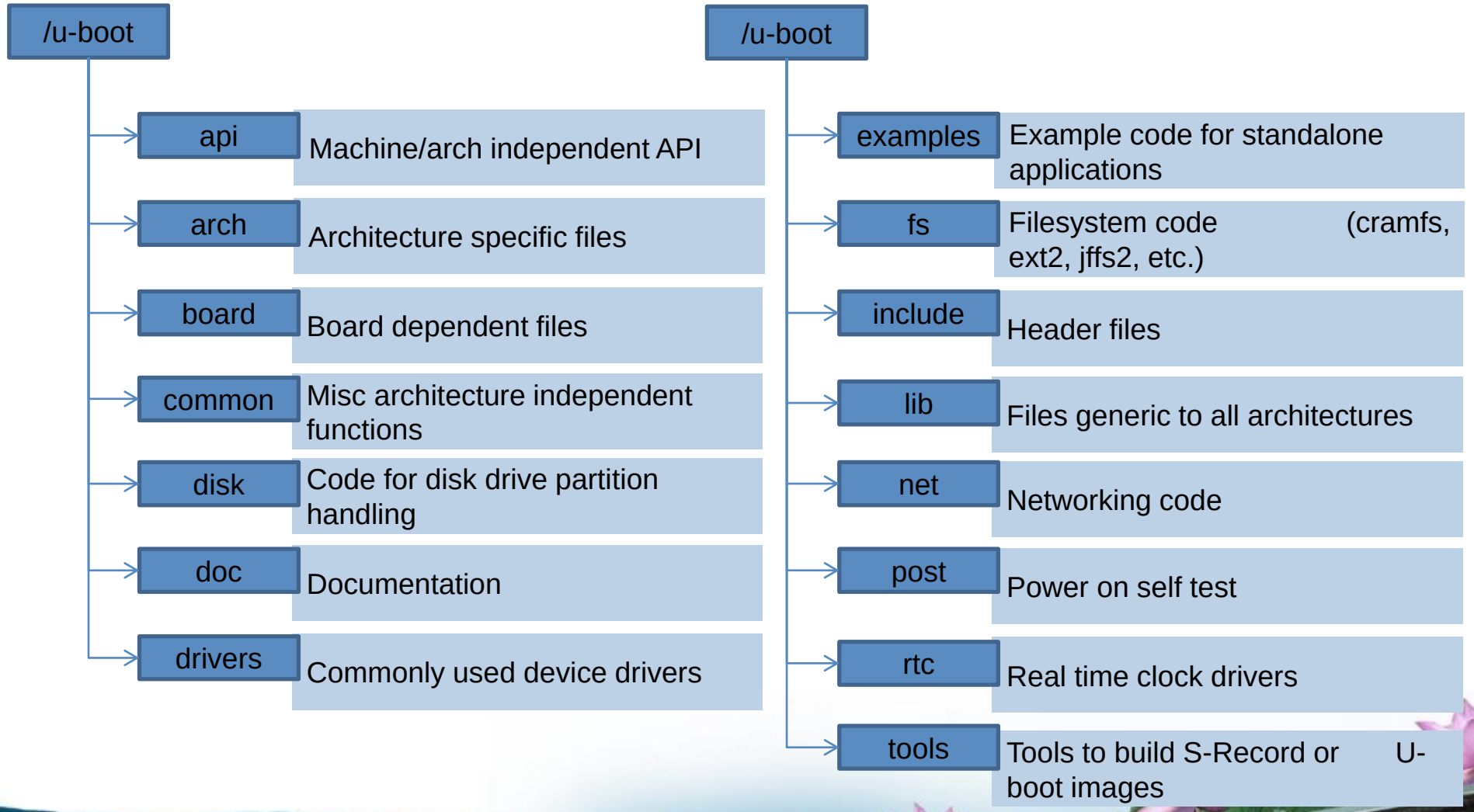
- **BL0; iRom code (boot-rom or pre-boot loader)**
  - **Simple platform independent code, stored in internal ROM**
  - **Initialize basic functions (clock, stack, heap, etc)**
  - **Load first stage boot loader (from booting device to internal SRAM)**
- **BL1; first boot loader**
  - **Simple platform independent code, stored in external memory(booting device)**
  - **Load second boot loader**
  - **Initialize NAND/MMC and DRAM controller**
- **BL2; second boot loader**
  - **Platform dependent complex code, stored in external memory**
  - **Initialize clock, UART**
  - **Load OS image from booting device to DRAM**
  - **Jump to operating system**

Uboot source code

- $ git clone ssh://your_id@review.tizen.org:29418/platform/kernel/u-boot.git u-boot-tizen
- $ cd u-boot-tizen
- $ git checkout -b tizen origin/tizen

https://wiki.tizen.org/wiki/How_to_Build_and_Load_Tizen_on_Odroid_U3

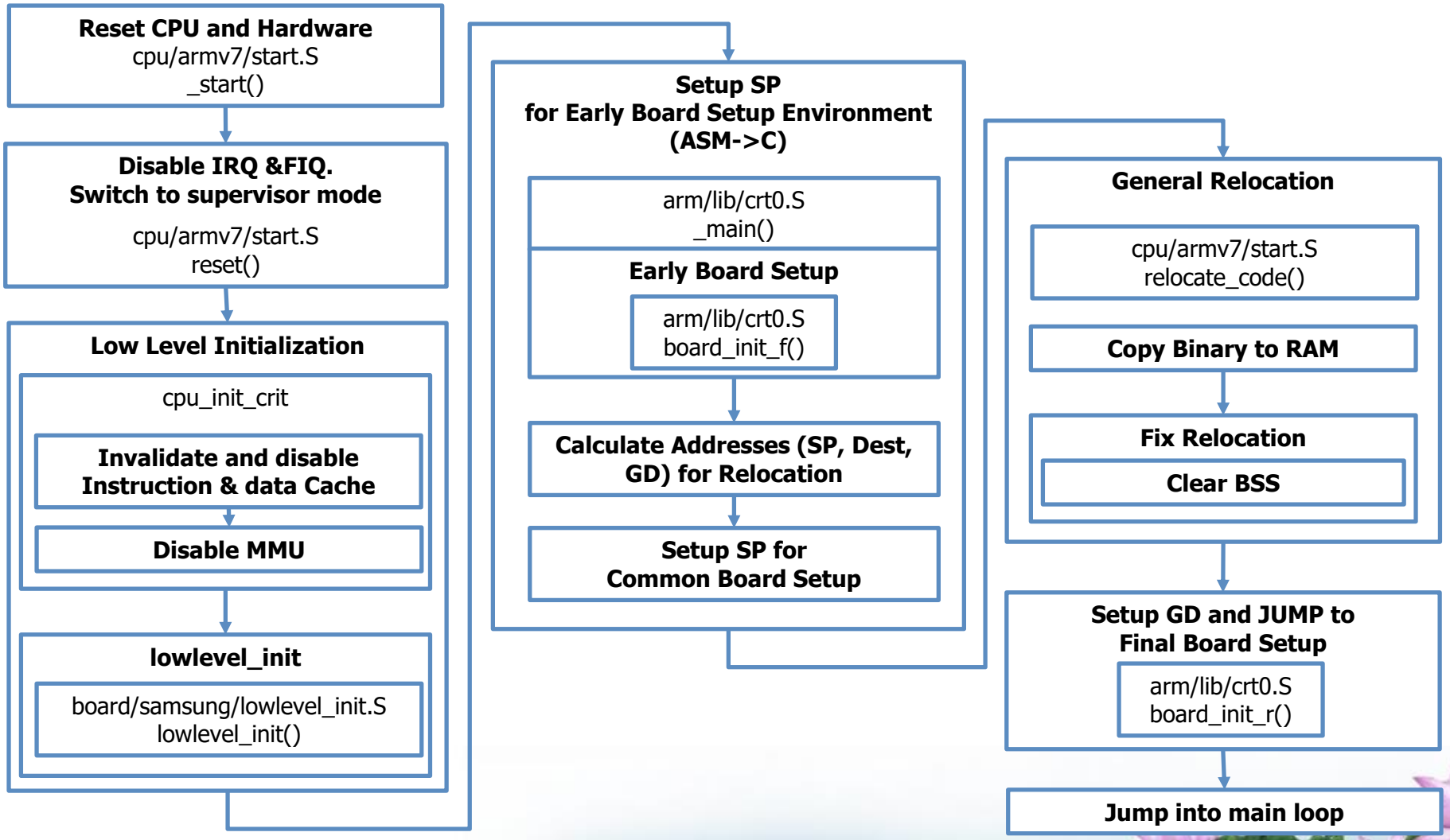
U-boot directory structure



U-boot code sequence

1. Disable all interrupts
2. Copy any initialized data from ROM to RAM
3. Zero the uninitialized data area
4. Allocate space for and initialize the stack
5. Initialize the processor’s stack pointer
6. Create and initialize the heap
7. Execute the initializers for all global data
8. Enable interrupts
9. Call main loop

U-boot code sequence



U-boot code sequence

- _start() /* Entry point function */
- reset() /* Reset CPU configurations */
  - save_boot_params() /* Save boot parameters */
  - Disable the FIQ & IRQ
  - Switch to supervisor mode
  - cpu_init_crit()
    - Invalidate I, D Cache
    - Disable I cache if CONFIG_SYS_ICACHE_OFF
    - Disable MMU
    - lowlevel_init()
- _main() /* setup initial stack & global data.
  And jump to C routine for board initialization */

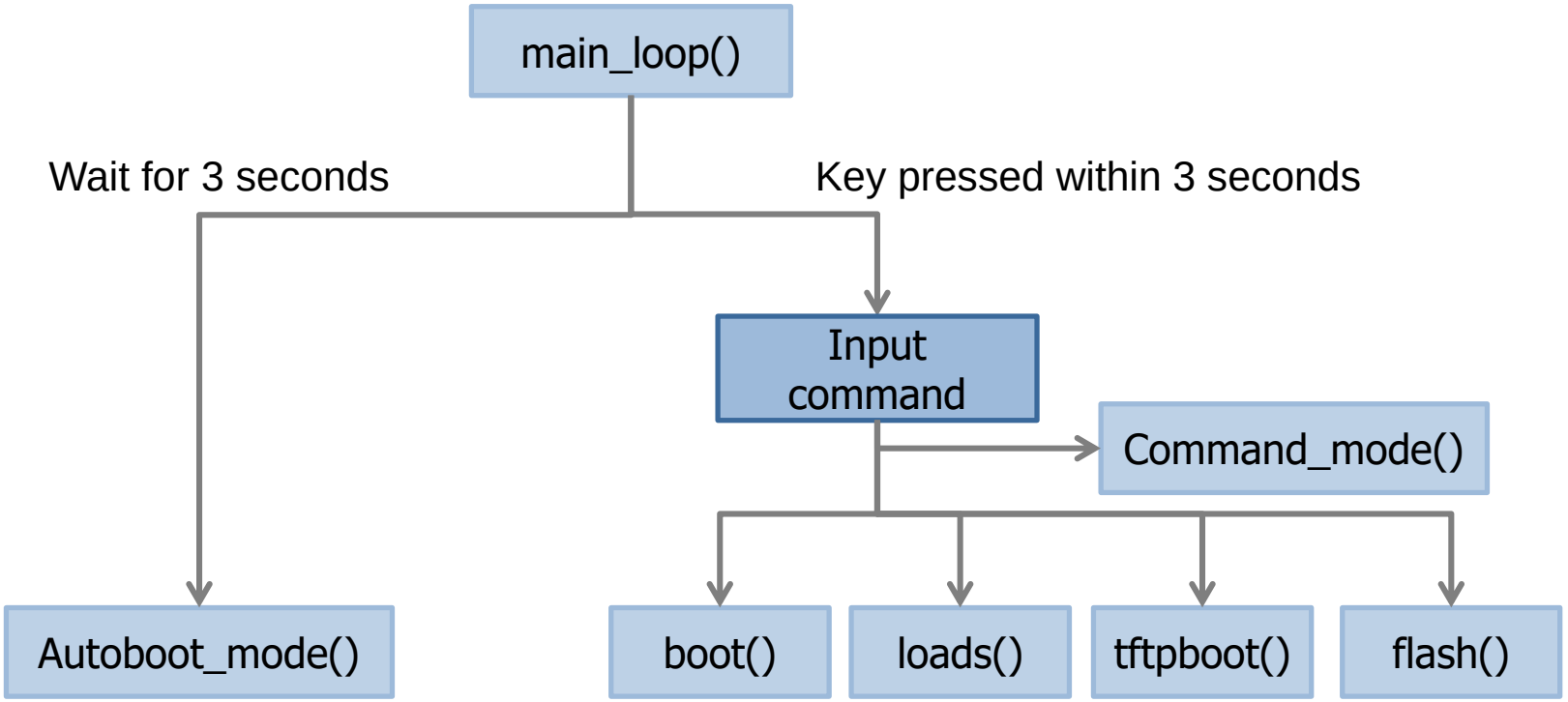
U-boot code sequence

- board_init_f()
  - arch_cpu_init // basic arch cpu dependent setup
  - board_early_init_f // early board initialization, UART pin setting
  - timer_init // initialize timer
  - env_init // initialize mmc/sdcard environment
  - init_baudrate // initialize baudrate settings
  - serial_init // serial communication setup
  - console_init_f // stage 1 init of console, configure console from console.c
  - display_banner // say that we are here, print u-boot banner
  - print_cpuinfo // display cpu info
  - checkboard // display board info
  - init_func_i2c // initialize I2C
  - dram_init // configure available RAM backs
  - arm_pci_init

U-boot code sequence

- board_init_r()
  - board_init // setup chipselects specific configuration
  - logbuff_init_ptrs // initialize default log level
  - mem_malloc_init // configure malloc area
  - nand_init mmc_init // initialize flash/mmc
  - env_relocate // relocate environment or set NULL
  - stdio_init // setup stdio ready for use
  - api_init // initialize API
  - console_init_r // fully init console as a device
  - arch_misc_init // miscellaneous arch dependent initialization
  - misc_init_r // miscellaneous platform dependent initialization
  - interrupt_init // set up exceptions
  - enable_interrupts // enable exceptions
  - board_late_init // board late initialization
  - eth_initialize // setup ethernet
  - main_loop // jump to main loop & waiting for commands from console

U-boot code sequence



Jump to kernel image address

Experiment：Create a bootable TF card

Boot images layout

Binary name	SD (blk)	eMMC boot* (blk)
bl1	1	0
bl2	31	30
u-boot-mmc.bin	63	62
tzsw	2111	2110

- Bl1, bl2, tzsw are provided by hardware vendor.

https://github.com/hardkernel/U-Boot/tree/odroid-v2010.12/sd_fuse

Compile your own u-boot

- *DTC* build from *u-boot-tizen* directory:
  - $ cd ./tools/dtc $ make install
  - $ dtc -v #Check DTC version
  - Version: DTC 1.4.0-dirty
- Build the U-Boot Tizen
- Now you can build the U-Boot image using the following commands:
  - $ CROSS_COMPILE=“arm-linux-gnueabihf-" make tizen_config
  - $ CROSS_COMPILE="arm-linux-gnueabihf-" make

Prepare your file

- First, a multiple dtbs should be appended to it (this is the one of differences with mainline U-Boot):
  - $ ./tools/mkimage_multidtb.sh u-boot.bin
- The script output is: u-boot-multi.bin
  And then make a compatible platform signature:
  - $ ./tools/mkimage_signed.sh u-boot-multi.bin tizen_config
- The script output is: u-boot-mmc.bin

Prepare your SD card

- Please follow this steps:
  - $ cd u-boot-tizen
  - $ mv u-boot-mmc.bin u-boot.bin #The script sd_fusing.sh uses u-boot.bin
  - $ mkdir fuse $ cd fuse
  - $ wget -q https://github.com/hardkernel/U-Boot/raw/odroid-v2010.12/sd_fuse/bl1.HardKernel
  - $ wget -q https://github.com/hardkernel/U-Boot/raw/odroid-v2010.12/sd_fuse/bl2.HardKernel
  - $ wget -q https://github.com/hardkernel/U-Boot/raw/odroid-v2010.12/sd_fuse/tzsw.HardKernel
  - $ wget -q https://github.com/hardkernel/U-Boot/raw/odroid-v2010.12/sd_fuse/sd_fusing.sh
  - $ chmod u+x sd_fusing.sh
- Please choose the steps suitable for your case (SD or eMMC):
- SD Card
  - Connect the SD Card Reader and insert your card.
  - Check the mass storage device (/dev/sd*) using 'dmesg | tail' commands.
- go into u-boot-tizen/fuse directory
- run:
  - $ sudo ./sd_fusing.sh /dev/sd*

Homework

- U-boot source code analysis report
- Add a new u-boot command to print your personalized information.

Acknowledgement

- This project is partially sponsored by Tianjin Samsung Electronics Co., Ltd.
- Special thanks to SKKU Embedded Software Laboratory. This material is based on the lecture provided by them, and the credit goes to the original authors.