
# U-boot

# Bootloader

- What is bootloader
- Boot : short bootstrap
  - Initialize basic of SOC (CPU, RAM, CLK)
  - BL1, BL2
- Loader
  - Load OS to RAM(DDR) form storage
  - u-boot

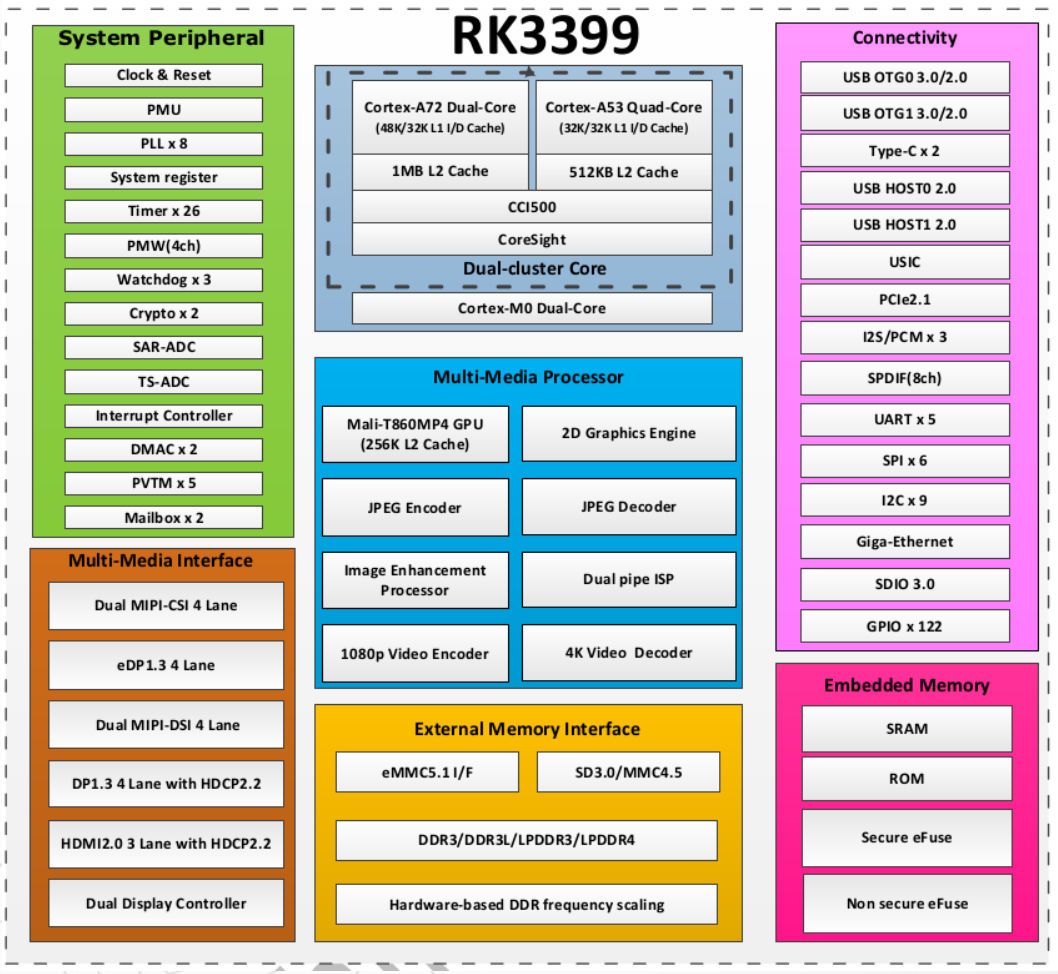
# All kinds of embedded Linux bootloader

- **U-boot**
- UEFI
- Redboot
- Stubby (Linaro) …
- Anyway, they are same target
  - Load and boot OS to RAM from storage

# Concepts of the Boot Loader

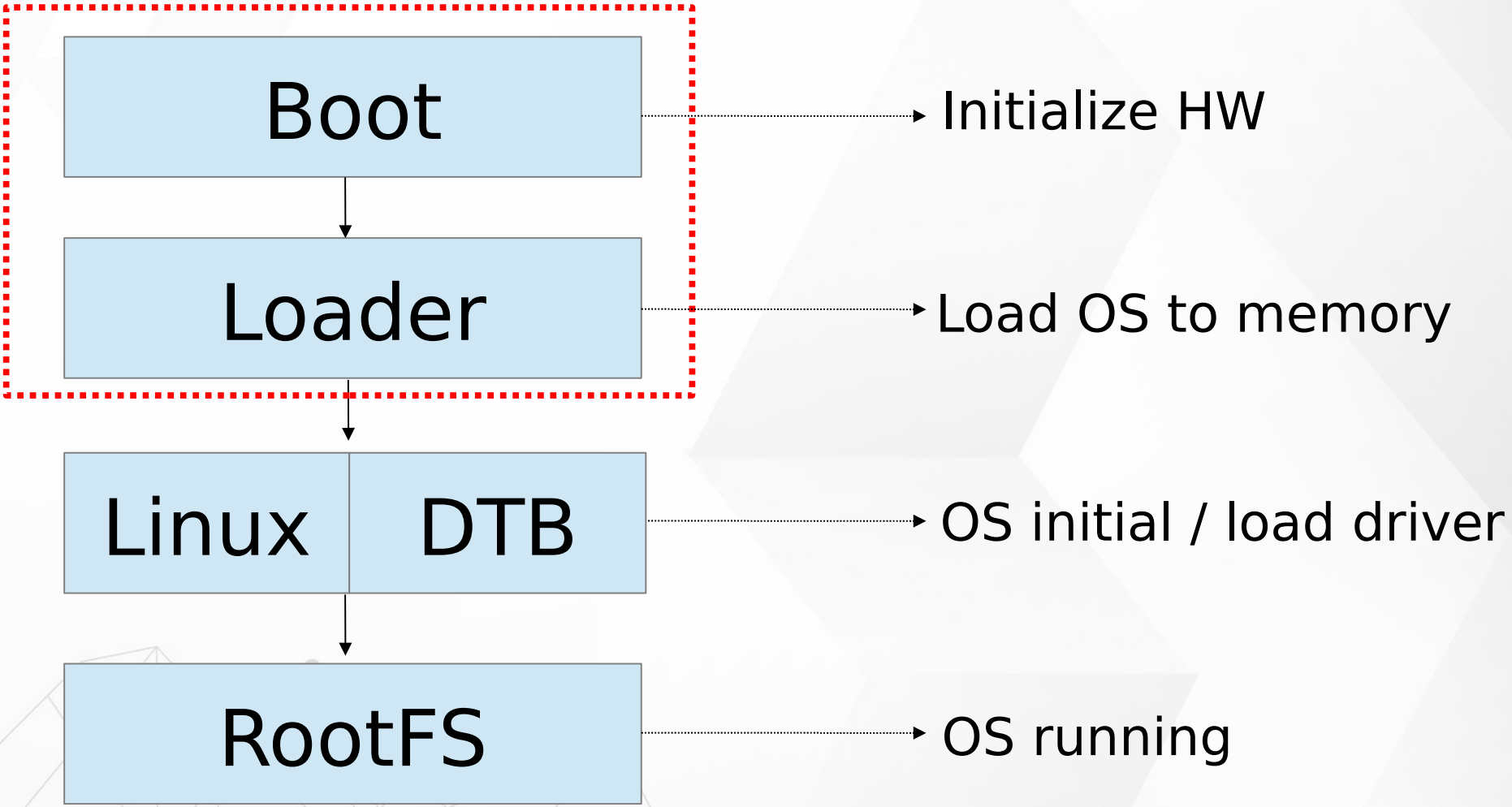
- Boot Loader is varied from CPU to CPU,from board to board.
- All the system software and data are stored in some kind of nonvolatile memory.
- Operation Mode of Boot Loader
  - **Boot : Initialize basic of SOC**
  - **Load : load OS to RAM then execute**

# RK3399 SOC




http://wiki.friendlyarm.com/wiki/index.php/NanoPi_M4#Diagram.2C_Layout_and_Dimension

# Embedded Linux System Booting



# Image Partition

partmap.txt – Image layout in SD card

| | | |
|---|---|---|
| Loader | idbloader.img | 0x8000,0x280000 |
| Environment | | 0x3F8000,0x8000 |
| Parameter | param4sd.txt | 0x400000,0x0400000 |
| u-boot | Uboot.img | 0x800000,0x0400000 |
| Trust | Trust.img | 0xC00000,0x0400000 |
| Misc | | 0x1000000,0x0400000 |
| Resource | Resource.img | 0x1400000,0x0C00000 |
| Linux kernel | Linux kernel | 0x2000000,0x2000000 |
| Boot | Boot.img | 0x4000000,0x2000000 |
| RootFS | rootfs.img | 0x6000000,RootFS Size |
| User data | | |

# Boot

- Power On BootROM code (work in cache)
  - Load BL1
- BL1 (work in cache)
  - Initial simple exception vectors, PLL (clock)
  - Initial Multi-CPU
  - Load BL2
- BL2 (work in cache)
  - Initial DDR memory
  - Initial C environment (stack, heap, )
  - Load BL31

# Boot

- BL31 (work in DDR)
  - Initial exception vectors
  - Load BL32 (u-boot)
- **BL32 U-boot**
  - Initial storage device
  - Load Linux Kernel
- Kernel
  - kernel/Documentation/arm64/booting.txt
  - Load RootFS

# Introduce U-boot

# U-boot

- Das U-Boot -- the Universal Boot Loader
- http://www.denx.de/wiki/U-Boot
- GitHub for u-boot
- Open Source follow GPL
- Supply many CPU
  - PPC, ARM, x86, MIPS, AVR32 …
- Supply basic periphery devices
  - UART, Flash, SD/MMC ….

# u-boot directory structure

- Arch → Many types CPU : Arm, mips, i386 ...
- Board → Many types develop board : Samsung, ti, davinci ...
- Tools → Make Image (u-boot, linux ) or S-Record image tool
- Drivers → Some HW control code
- Fs → Supply file system : fat, jffs2, ext2, cramfs
- Lib → General public library : CRC32/16, bzlib, ldiv ..
- Disk→ Supply disk driver and partition handling

# u-boot directory structure

- Common → Major command and relation environment setting source code
- Api → Implement unrelated hardware code
- nand_spl, onenand_ipl
  → Related nand/onenand flash control
- Example → Standalone application

# u-boot directory structure about rk3399

- **arch/arm/cpu/armv8/**
  - ARMv8 relate
- **arch/arm/cpu/armv8/rk33xx/**
  - rk3399 related
  - Clock, i2c, irom, mmc, emmc ...
- **board/rockchip/rk33xx/**
  - Board level related
  - Peripheral initial

# u-boot directory structure about rk3399

- Common
  - u-boot command related
- **include/configs/**
  - rk_default_config.h
  - rk33plat.h

# How to build u-boot

- Clear : $ **make distclean**
- Configure : $ **make nanopi-m4-rk3399_slash_defconfig**
- Build : $ **make -j4**
- Create Image : $ tools/loaderimage --pack --uboot ./u-boot-dtb.bin \
  uboot.img

# Configure

include/configs/rk_default_config.h

include/configs/rk33plat.h

```
#define CONFIG_LCD_LOGO
#define CONFIG_LCD_BMP_RLE8
#define CONFIG_CMD_BMP

/* CONFIG_COMPRESS_LOGO_RLE8 or CONFIG_COMPRESS_LOGO_RLE16 */
#undef CONFIG_COMPRESS_LOGO_RLE8
#undef CONFIG_COMPRESS_LOGO_RLE16

#define CONFIG_BMP_16BPP
#define CONFIG_BMP_24BPP
#define CONFIG_BMP_32BPP
#define CONFIG_SYS_WHITE_ON_BLACK
#define LCD_BPP                 LCD_COLOR16

#define CONFIG_LCD_MAX_WIDTH    4096
#define CONFIG_LCD_MAX_HEIGHT   2048

/* rk lcd size at the end of ddr address */
#define CONFIG_RK_FB_DDREND

#ifdef CONFIG_RK_FB_DDREND
/* support load bmp files for kernel logo */
#define CONFIG_KERNEL_LOGO

/* rk lcd total size = fb size + kernel logo size */
#define CONFIG_RK_LCD_SIZE      SZ_32M
#define CONFIG_RK_FB_SIZE       SZ_16M
#endif
```

# Exercise

- Compile a u-boot for nanopi-m4
- Update u-boot for nanopim-m4

# U-boot Common Function

# Operating U-boot

- Understand and use command
- Understand and modify parameters

# Help

- $ help

  → help mm

- $ ?

```
=> help mm
mm - memory modify (auto-incrementing address)

Usage:
mm [.b, .w, .l, .q] address
=>
```

# Help

- help

  → print command description/usage

- $ help

  → help mm

- $ ?

```
=> help mm
mm - memory modify (auto-incrementing address)

Usage:
mm [.b, .w, .l, .q] address
=>
```

# md

- md

  → memory display

  → md [.b, .w, .l, .q] address [# of objects]

```
=> md 0x02080000
02080000: 4e04e260 e461206c 0808a115 2a666646    `..Nl a.....Fff*
02080010: 88689ca1 224002e2 2e000a62 20a26262    ..h...@"b...bb.
02080020: a8207386 60a626e4 00016006 62a40642    .s ..&.`.`..B..b
02080030: e000a239 62476067 a3284802 24e66242    9...g`Gb.H(.Bb.$
02080040: 22300806 32a0c270 41620081 62042664    ..0"p..2..bAd&.b
```

# mw

- mw

  → memory write

  → mw [.b, .w, .l, .q] address value [count]

```
=> mw.l 0x02080000 0x12345678 1
=> md.l 0x02080000
02080000: 12345678 e461206c 0808a115 2a666646    xV4.l a.....Fff*
02080010: 88689ca1 224002e2 2e000a62 20a26262    ..h...@"b...bb.
02080020: a8207386 60a626e4 00016006 62a40642    .s ..&.`.`..B..b
02080030: e000a239 62476067 a3284802 24e66242    9...g`Gb.H(.Bb.$
02080040: 22300806 32a0c270 41620081 62042664    ..0"p..2..bAd&.b
```

# mmc

- mmc list

  → lists available devices

```
=> mmc list
mmc@fe310000: 2
mmc@fe320000: 1 (SD)
sdhci@fe330000: 0
```

# mmc

- mmc info

  → display info of the current MMC device

```
=> mmc dev 1
switch to partitions #0, OK
mmc1 is current device
=> mmcinfo
Device: mmc@fe320000
Manufacturer ID: 3
OEM: 5344
Name: SU04G
Bus Speed: 50000000
Mode: SD High Speed (50MHz)
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 3.7 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
```

# mmc

- mmc part

  → lists available partition on current mmc device

```
=> mmc part

Partition Map for MMC device 1  --   Partition Type: DOS

Part    Start Sector    Num Sectors     UUID            Type
  1     196608          7547904         97ddff01-01     83
```

# mmc

- mmc dev

  → show or set current mmc device [partition]

  → mmc dev [dev] [part]

```
=> mmc dev 1
switch to partitions #0, OK
mmc1 is current device
=> mmcinfo
Device: mmc@fe320000
Manufacturer ID: 3
OEM: 5344
Name: SU04G
Bus Speed: 50000000
Mode: SD High Speed (50MHz)
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 3.7 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
```

# FAT

- fatls

  → list files in a directory

  → fatls <interface> [<dev[:part]>] [directory]
  - list files from 'dev' on 'interface' in a 'directory'

```
=> fatls mmc 1:1 boot
<DIR>       4096 .
<DIR>       4096 ..
        18385408 Image
          104828 rk3399-nanopi4-rev01.dtb
```

# FAT

- fatload

  → fatload - load binary file from a dos filesystem

  → Load binary file 'filename' from 'dev' on 'interface'
  to address 'addr' from dos filesystem.

```
=> fatload mmc 1:1 0x02080000 boot/Image
18385408 bytes read in 1639 ms (10.7 MiB/s)

=> md.l 0x02080000
02080000: 91005a4d 143c7fff 00080000 00000000    MZ....<.........
02080010: 01370000 00000000 0000000a 00000000    ..7.............
02080020: 00000000 00000000 00000000 00000000    ................
02080030: 00000000 00000000 644d5241 00000040    ........ARMd@...
02080040: 00004550 0002aa64 00000000 00000000    PE..d...........
02080050: 00000001 020600a0 1402020b 0136f000    .............6..
02080060: 00000000 00000000 00f25a60 00001000    ........`Z......
```

# EXT4

- ext4ls

  → list files in a directory with ext4

  → ext4ls <interface> <dev[:part]> [directory]
  - list files from 'dev' on 'interface' in a 'directory'

```
=> ext4ls mmc 1:1
<DIR>       4096 .
<DIR>       4096 ..
<DIR>      16384 lost+found
<DIR>       4096 bin
           30195 busybox.config
<SYM>          8 data
<DIR>       4096 dev
<DIR>       4096 etc
<DIR>       4096 home
             178 init
```

# EXT4

- ext4load

  → ext4load - load binary file from a Ext4 filesystem

  → ext4load <interface> [<dev[:part]> [addr [filename [bytes [pos]]]]]
  load binary file 'filename' from 'dev' on 'interface'
  to address 'addr' from ext4 filesystem

```
=> ext4load mmc 1:1 0x02080000 boot/rk3399-nanopi4-rev01.dtb
104828 bytes read in 39 ms (2.6 MiB/s)
=> md 0x02080000
02080000: edfe0dd0 7c990100 38000000 fc710100    .......|...8..q.
02080010: 28000000 11000000 10000000 00000000    ...(............
02080020: 80270000 c4710100 00000000 00000000    ..'...q.........
02080030: 00000000 00000000 01000000 00000000    ................
02080040: 03000000 27000000 00000000 65697266    .......'....frie
02080050: 796c646e 63656c65 6e616e2c 2d69706f    ndlyelec,nanopi-
02080060: 7200346d 636b636f 2c706968 33336b72    m4.rockchip,rk33
02080070: 00003939 03000000 04000000 0b000000    99..............
02080080: 01000000 03000000 04000000 1c000000    ................
02080090: 02000000 03000000 04000000 2b000000    ............+...
020800a0: 02000000 03000000 17000000 37000000    ............7...
020800b0: 65697246 796c646e 63656c45 6e614e20    FriendlyElec Nan
020800c0: 2069506f 0000344d 01000000 5f726464    oPi M4......ddr_
020800d0: 696d6974 0000676e 03000000 14000000    timing..........
020800e0: 00000000 6b636f72 70696863 7264642c    ....rockchip,ddr
020800f0: 6d69742d 00676e69 03000000 04000000    -timing.........
```

# printenv

- printenv
  - → print environment variables
  - → printenv name

```
=> printenv
altbootcmd=setenv boot_syslinux_conf extlinux/extlinux-rollback.conf;run distro_bootcmd
arch=arm
baudrate=1500000
board=evb_rk3399
board_name=evb_rk3399
boot_a_script=load ${devtype} ${devnum}:${distro_bootpart} ${scriptaddr} ${prefix}${script}; so
boot_efi_binary=load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} efi/boot/bootaa64
ernel_addr_r} ${fdt_addr_r};else bootefi ${kernel_addr_r} ${fdtcontroladdr};fi
boot_efi_bootmgr=if fdt addr ${fdt_addr_r}; then bootefi bootmgr ${fdt_addr_r};else bootefi boo
```

```
=> printenv loadimage
loadimage=ext4load mmc 1:1 $kernel_addr_r boot/Image
```

# setenv

- setenv

  → set environment variables

  → setenv name value

```
=> setenv test 12345
=> printenv test
test=12345
=> setenv test
=> printenv test
## Error: "test" not defined
```

# saveenv

- setenv

  → save environment variables to persistent storage

# Simple Script

```
=> ext4ls mmc 1:1 boot
<DIR>       4096 .
<DIR>       4096 ..
        18385408 Image
          104828 rk3399-nanopi4-rev01.dtb
<SYM>           7 Image_1
<SYM>           7 Image_2
=> if test readkernel = Image_1;then
> ext4load mmc 1:1 0x02080000 boot/Image_1 ;
> echo read Image_1;
> else
> ext4load mmc 1:1 0x02080000 boot/Image_2;
> echo read Image_2;
> fi
18385408 bytes read in 1625 ms (10.8 MiB/s)
read Image_2
```

# run

- run

  → run commands in an environment variable

  → run var [...]

```
=> printenv bootcmd
bootcmd=run loadimage; run loadfdt; booti $kernel_addr_r - $fdt_addr_r
=> run bootcmd
18385408 bytes read in 1625 ms (10.8 MiB/s)
104828 bytes read in 26 ms (3.8 MiB/s)
## Flattened Device Tree blob at 01f00000
   Booting using the fdt blob at 0x1f00000
ERROR: reserving fdt memory region failed (addr=0 size=0)
   Loading Device Tree to 00000000f1f1b000, end 00000000f1f3797b ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.179 (slash@slash-ThinkPad-T420) (gcc vers
08 CST 2021
```

# booti

- booti

  → booti - boot Linux kernel 'Image' format from memory

  → booti [addr [initrd[:size]] [fdt]]

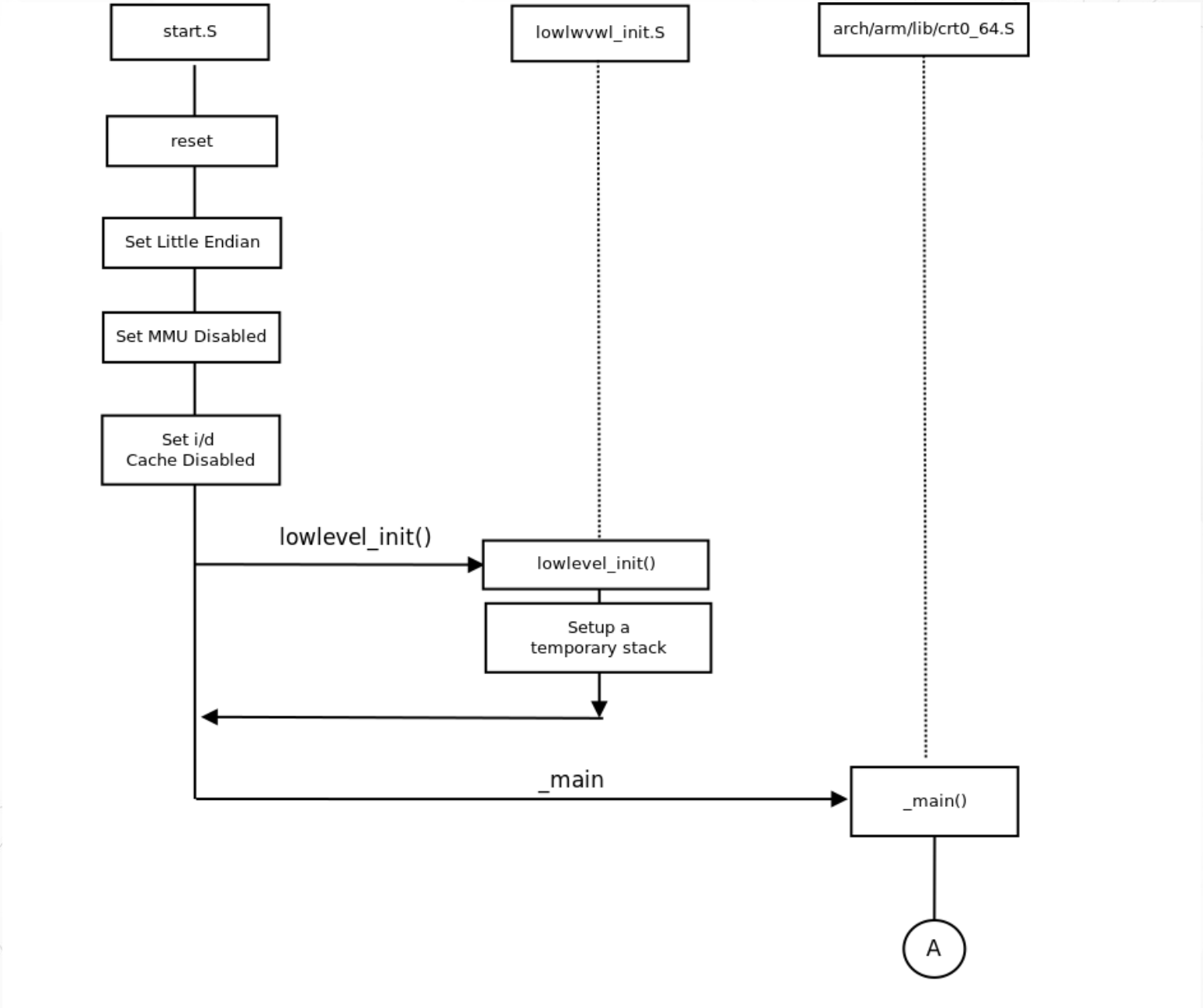
```
printenv bootcmd
bootcmd=run loadimage; run loadfdt; booti $kernel_addr_r - $fdt_addr_r
```

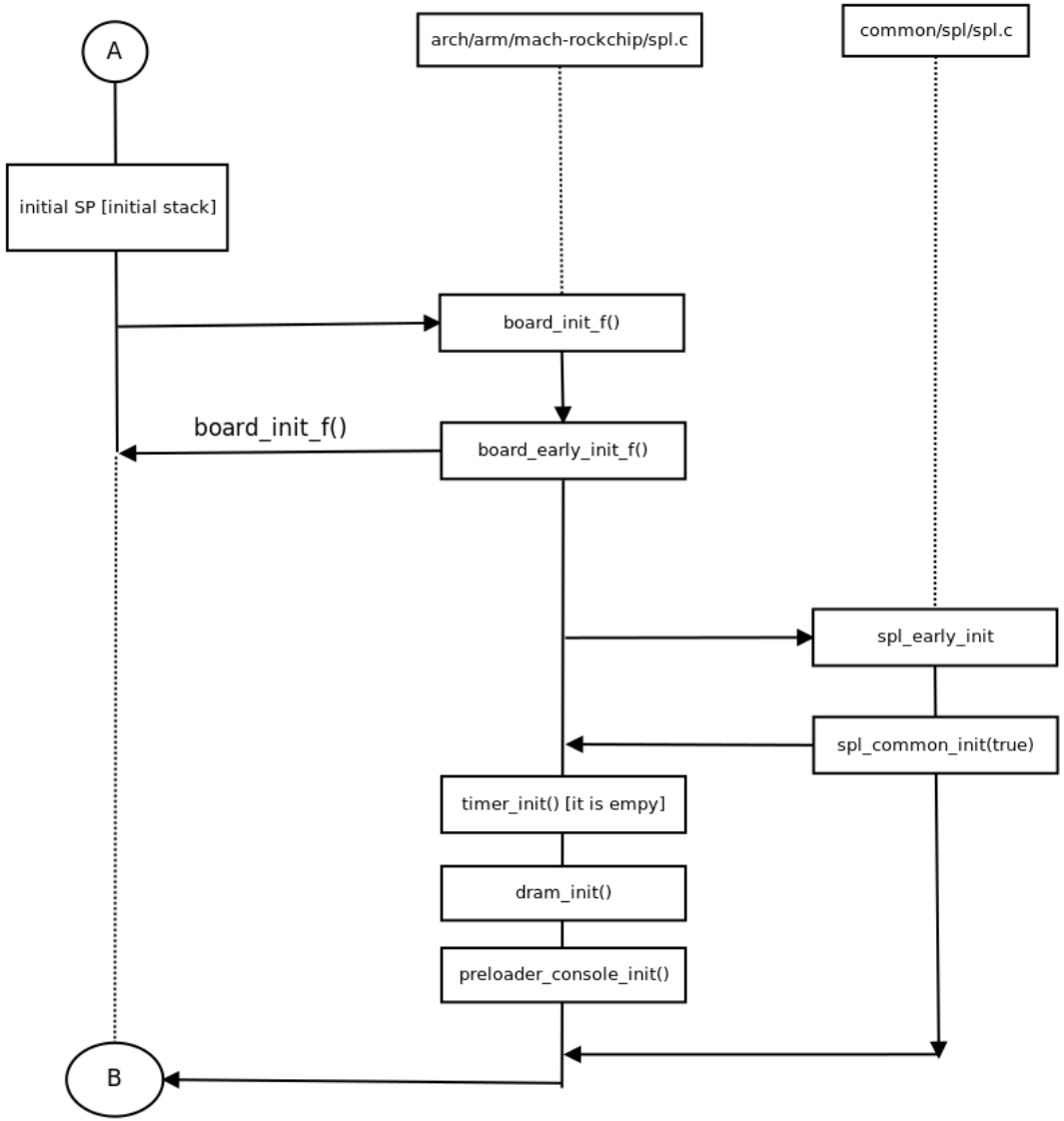
```
=> run bootcmd
18385408 bytes read in 1625 ms (10.8 MiB/s)
104828 bytes read in 26 ms (3.8 MiB/s)
## Flattened Device Tree blob at 01f00000
   Booting using the fdt blob at 0x1f00000
ERROR: reserving fdt memory region failed (addr=0 size=0)
   Loading Device Tree to 00000000f1f1b000, end 00000000f1f3797b ... OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
```

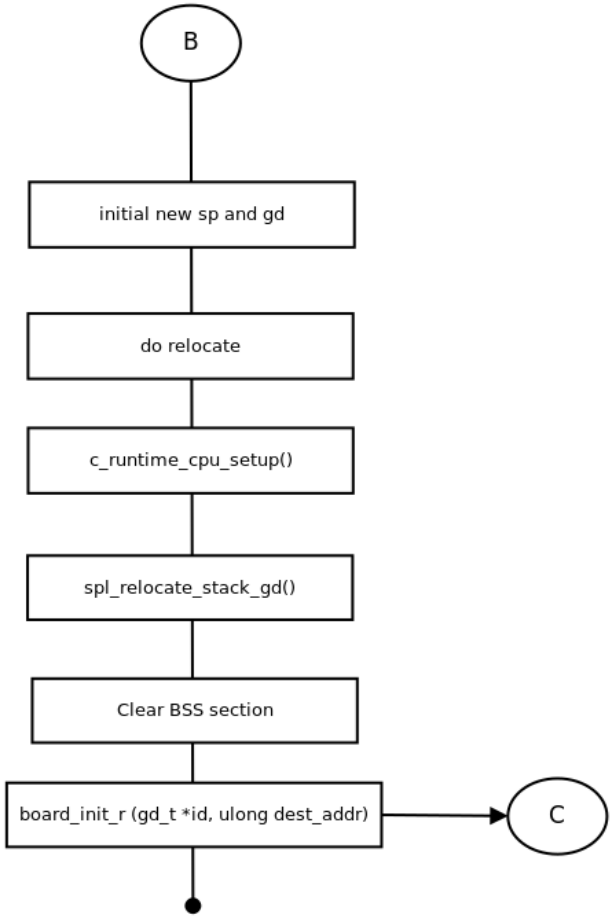
# Exercise

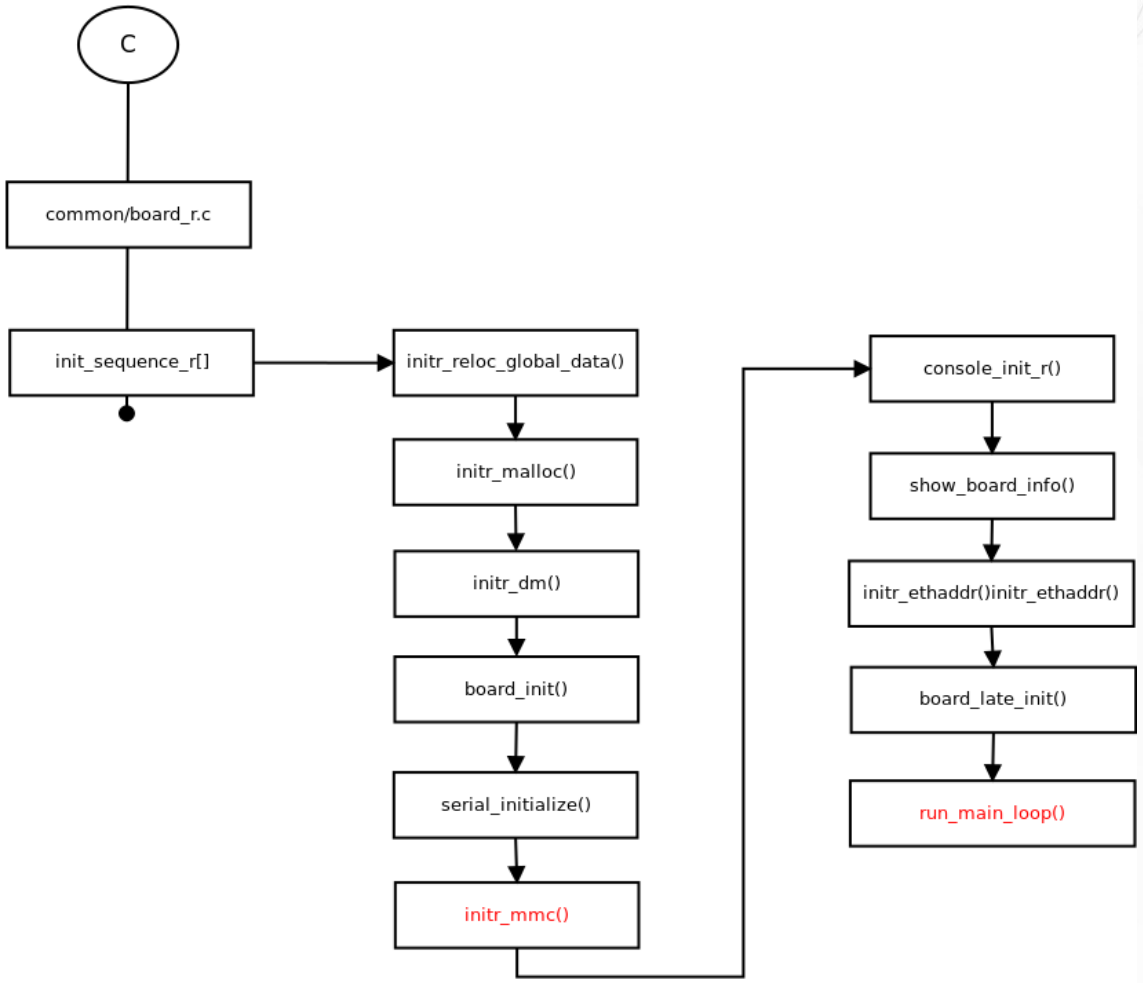
- Try to use everyone command in u-boot
- Use help function to know how to use command
- To know bootcmd and bootargs

# U-boot Initialize Sequence



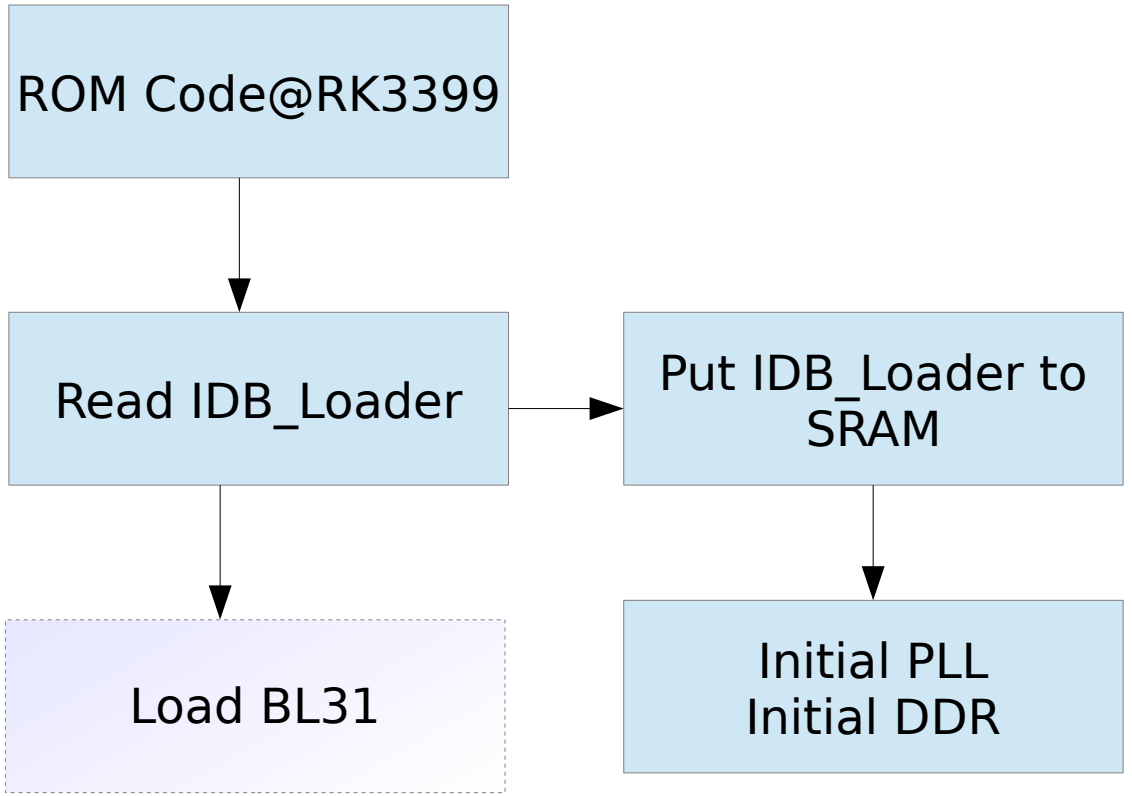




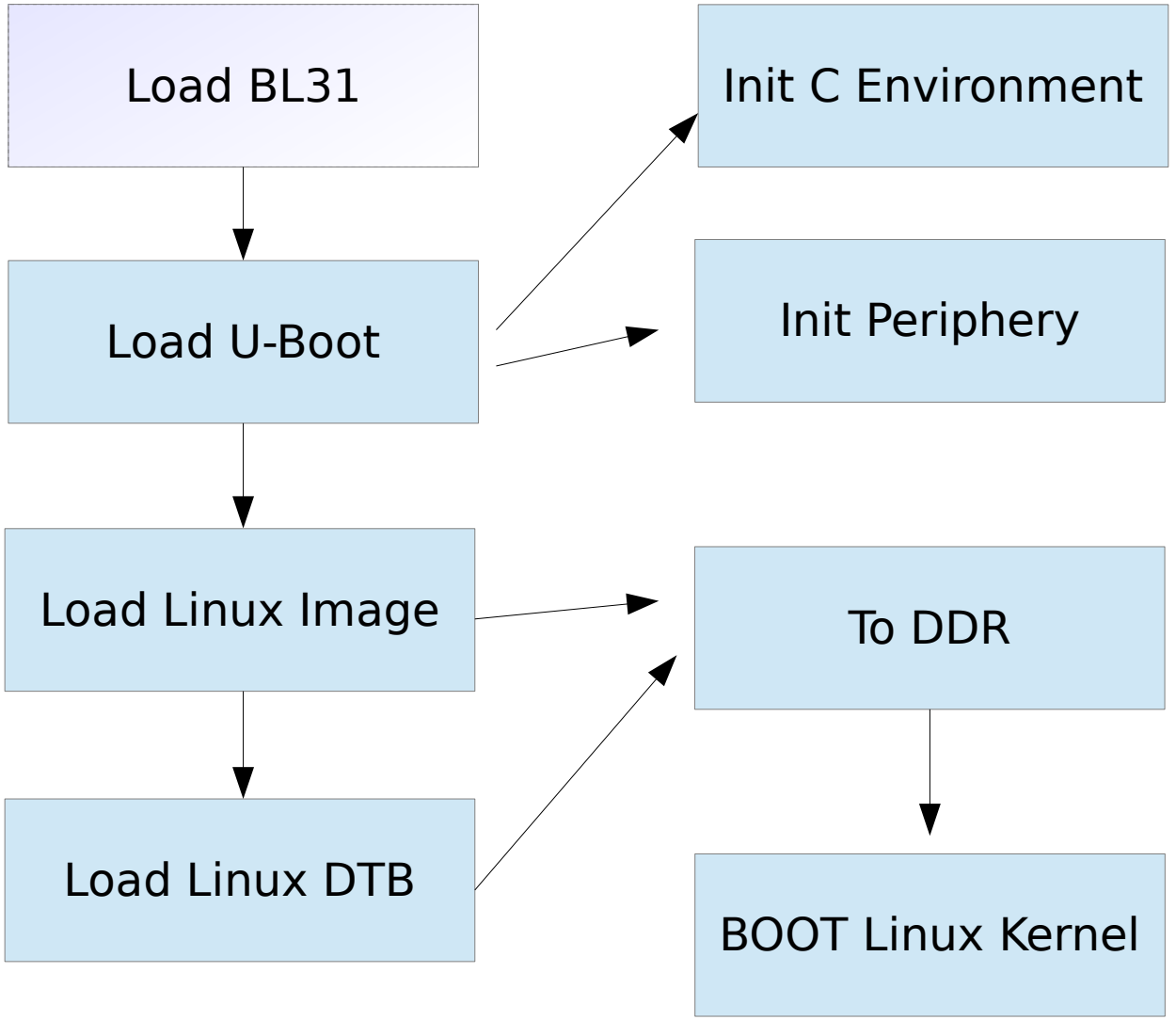


# Boot Linux kernel

# System Start Up



# System Start Up



# Boot Linux Kernel Command

- bootcmd
  - $ printenv bootcmd

```
bootcmd=run loadimage; run loadfdt;
booti $kernel_addr_r - $fdt_addr_r
```

  - $ printenv loadimage → load Linux kernel Image

```
loadimage=ext4load mmc 1:1 $kernel_addr_r boot/Image
```

  - $ printenv loadfdt → Load Linux kernel device tree blob

```
loadfdt=ext4load mmc 1:1 $fdt_addr_r boot/rk3399-nanopi4-rev01.dtb
```

# Boot OS (Linux) Command

- booti
  → boot Linux **kernel 64 bit standard** kernel **image**
- boota
  → boot **android** style kernel image
- bootrk
  → boot **rockchip** style kernel image
- Boot command
  → booti [Kernel image addr] [RAM disk adr ] [DTB addr]
  → $ booti **kernel_addr ramdisk_addr DTB_addr**

# Linux Kernel Command (bootargs)

- bootargs
  - $ printenv bootargs (Linux kernel command)

```
bootargs=earlycon=uart8250,mmio32,0xff1a0000 swiotlb=1
console=ttyFIQ0 rootwait root=/dev/mmcblk0p1 rw
```

  - earlycon : earlay console device
  - console : Linux console device
  - root : RootFS device

- Enter Linux kernel to check
  - $ cat /proc/cmd

# How to jump to kernel

- Use boot command

  → cmd/cmd_booti.c

- Jump to Linux kernel

  → arch/arm/lib/bootm.c

  → boot_os_fn *bootm_os_get_boot_func(int os)

  → int do_bootm_linux(int flag, int argc, char *const argv[],
  bootm_headers_t *images)

  → void (*kernel_entry)( void *fdt_addr,
  void *res0,
  void *res1,
  void *res2);

# Linux Enter Point

arch/arm/lib/bootm.c

```
static void boot_jump_linux(bootm_headers_t *images, int fl
{
#ifdef CONFIG_ARM64
    void (*kernel_entry)(void *fdt_addr, void *res0, void *
            void *res2);
    int fake = (flag & BOOTM_STATE_OS_FAKE_GO);

    kernel_entry = (void (*)(void *fdt_addr, void *res0, vo
            void *res2))images->ep;
```

```
    unsigned long machid = gd->bd->bi_arch_number;
    char *s;
    void (*kernel_entry)(int zero, int arch, uint params);
    unsigned long r2;
    int fake = (flag & BOOTM_STATE_OS_FAKE_GO);

    kernel_entry = (void (*)(int, int, uint))images->ep;
```

```
    if (!fake) {
#ifdef CONFIG_ARMV7_NONSEC
        if (armv7_boot_nonsec()) {
            armv7_init_nonsec();
            secure_ram_addr(_do_nonsec_entry)(kernel_entry,
                                              0, machid, r2);
        } else
#endif
            kernel_entry(0, machid, r2);
    }
#endif
}
```

```
if (IMAGE_ENABLE_OF_LIBFDT && images->ft_len)
    r2 = (unsigned long)images->ft_addr;
else
    r2 = gd->bd->bi_boot_params;
```

Assign DTB

Start To Kernel

# Add Feature To U-boot

# Add Feature

- Add command → common/
- Add driver → driver
- Add application → example

# Add Command

- How to create a command ?
- Directory
  - cmd/ → booti.c , mmc.c, mem.c ...
- U_BOOT_CMD(name,maxargs,rep,cmd,usage,help)
  - include/command.h

# How to Command

cmd/cmd_version.c

```
static int do_version(struct cmd_tbl *cmdtp, int flag, int argc,
              char *const argv[])
{
    char buf[DISPLAY_OPTIONS_BANNER_LENGTH];

    printf("hellow_wold\n");

    printf(display_options_get_banner(false, buf, sizeof(buf)));

    return 0;
}

U_BOOT_CMD(
    version,    1,      1,  do_version,
    "print monitor, compiler and linker version",
    ""
);
```

Include/command.h

#define U_BOOT_CMD(_name, _maxargs, _rep, _cmd, _usage, _help)

# Exercise

- Reference version command

  → create a helloworld command

- Reference LED command

  → create a led command for nanopi-m4