



U-Boot

# 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



# Different Embedded Linux bootloader

- **U-boot**

- **UEFI**

- **Redboot**

- **Stubby (Linaro) ...**

- **Anyway, they are same target**

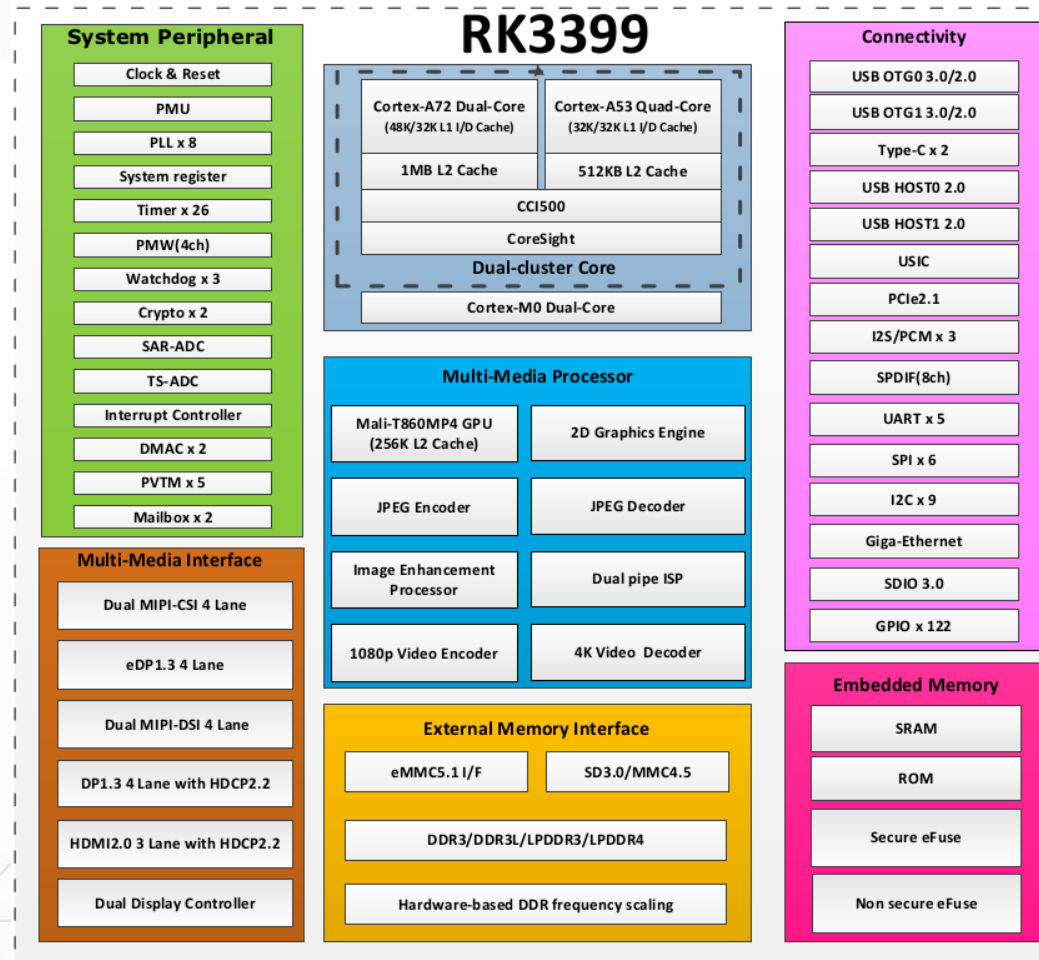
- Load and boot OS to RAM from storage



# Concepts of the Boot Loader

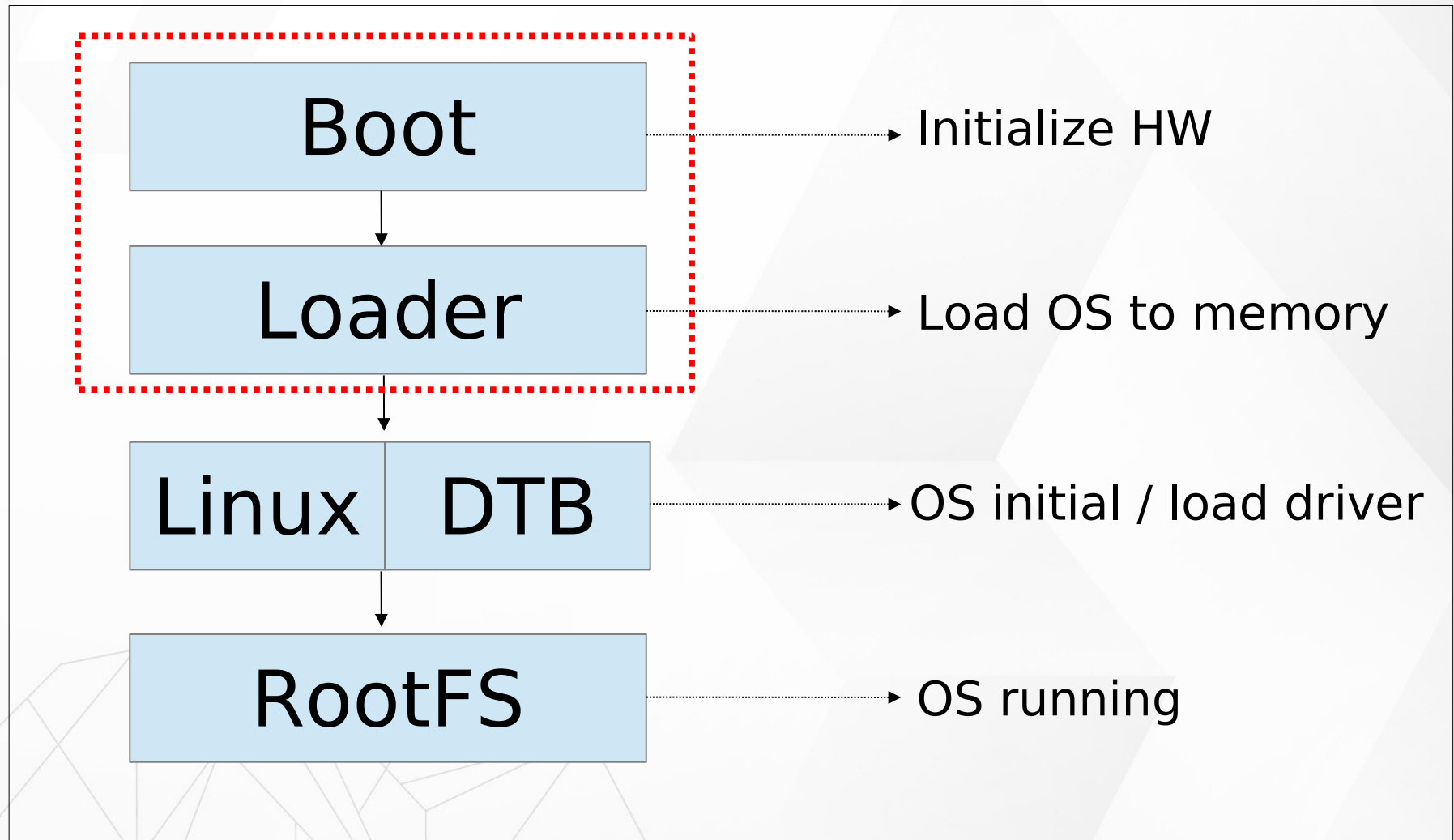
- Boot Loader is varied from CPU to CPU, from board to board.
- All the system software and data are stored in some kind of nonvolatile memory.
- Operation Mode of Boot Loader
  - **Boot : Initialize basic of SOC**
  - **Load : load OS to RAM then execute**

# RK3399 SOC



[http://wiki.friendlyarm.com/wiki/index.php/NanoPi\\_M4#Diagram.2C\\_Layout\\_and\\_Dimension](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
u-boot	Uboot.img	0x800000,0x0400000
Trust	Trust.img	0xC00000,0x0400000
Linux kernel	Linux kernel	SD Card Part List 4
RootFS	rootfs.img	SD Card Part List 5

# 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`
- Mount 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
- 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

# Rk3399 Configure File

## Common

- u-boot command related

## include/configs/

- config\_distro\_bootcmd.h
- evb\_rk3399.h
- rk3399\_common.h

# How to Build u-boot

## ➤ Easy Build :

[CMD] `cd ./rockchip-bsp`

[CMD] `./build/mk-uboot.sh rockpi4b`

## ➤ Clear :

[CMD] `make distclean`

## ➤ Configure :

[CMD] `make rock-pi-4b-rk3399_defconfig`

## ➤ Build :

[CMD] `make -j4`

## ➤ Create Image :

[CMD] `cd rockchip-bsp/rkbin/tool/`

[CMD] `loaderimage --pack --uboot ../../u-boot/u-boot-dtb.bin /uboot.img`





# U-boot Configure



## Change u-boot about

- Command
  - [cmd] help
- Boot parameter
  - Kernel load address
- Function
  - Enable LED



# U-boot Configure

## ▶ Edit Configure

### ▶ Method 1

- menuconfig
  - [cmd] make menuconfig

### ▶ Method 2

- Edit configure file
  - config\_distro\_bootcmd.h
  - evb\_rk3399.h
  - rk3399\_common.h


# 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      xV4.1 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 List

## fatls

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

```
=> fatls mmc 1:4
 155639 config-4.4.154-113-rockchip-gdb9dfc2cdd25
20371464 vmlinuz-4.4.154-113-rockchip-gdb9dfc2cdd25
          extlinux/
          dtbs/
          overlays/
0371464  vmlinuz-4.4.154
   1968  hw_intfc.conf
4786387 System.map-4.4.154-00039-g00fccd3
 156441 config-4.4.154
4786387 System.map-4.4.154
5038020 initrd.img-4.4.154
```

# FAT Load

## fatload

- fatload - load binary file from a FAT filesystem
- fatload <interface> [<dev[:part]> [addr [filename [bytes [pos]]]]]  
load binary file 'filename' from 'dev' on 'interface'  
to address 'addr' from fat filesystem

```
=> fatload mmc 1:4 ${kernel_addr_r} vmlinuz-4.4.154  
reading vmlinuz-4.4.154  
20371464 bytes read in 858 ms (22.6 MiB/s)
```

```
=> fatload mmc 1:4 ${fdt_addr_r} dtbs/4.4.154/rockchip/rockpi-4b-linux.dtb  
reading dtbs/4.4.154/rockchip/rockpi-4b-linux.dtb  
94603 bytes read in 15 ms (6 MiB/s)
```

```
=> fatload mmc 1:4 ${ramdisk_addr_r} initrd.img-4.4.154  
reading initrd.img-4.4.154  
5038020 bytes read in 215 ms (22.3 MiB/s)
```

# 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

➤ saveenv

➤ setenv

→ set environment variables

→ setenv name value

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



# Simple Script

LED Blank

Script

```
1=> while true; do; gpio toggle 125; sleep 1; done
2gpio: pin 125 (gpio 125) value is 0
3gpio: pin 125 (gpio 125) value is 1
4gpio: pin 125 (gpio 125) value is 0
5gpio: pin 125 (gpio 125) value is 1
6gpio: pin 125 (gpio 125) value is 0
7gpio: pin 125 (gpio 125) value is 1
```

# run

 run

- run commands in an environment variable
- run var [...]

```
=> run bootcmd
switch to partitions #0, OK
mmc1 is current device
Scanning mmc 1:4...
Found /extlinux/extlinux.conf
pxefile_addr_str = 0x00500000
bootfile = /extlinux/extlinux.conf
Retrieving file: /extlinux/extlinux.conf
reading /extlinux/extlinux.conf
1646 bytes read in 5 ms (321.3 KiB/s)
select kernel
1:  kernel-4.4.154
2:  kernel-4.4.154-999-rockchip-gcfa47f25e
3:  kernel-4.4.154-113-rockchip-gdb9dfc2cdd25
4:  kernel-4.4.154-00039-g00fccd3
5:  kernel-4.4.154
Enter choice: Retrieving file: /hw_intfc.conf
reading /hw_intfc.conf
1968 bytes read in 4 ms (480.5 KiB/s)
```

command

# booti

## booti

- booti - boot Linux kernel 'Image' format from memory
- booti [addr [initrd[:size]] [fdt]]

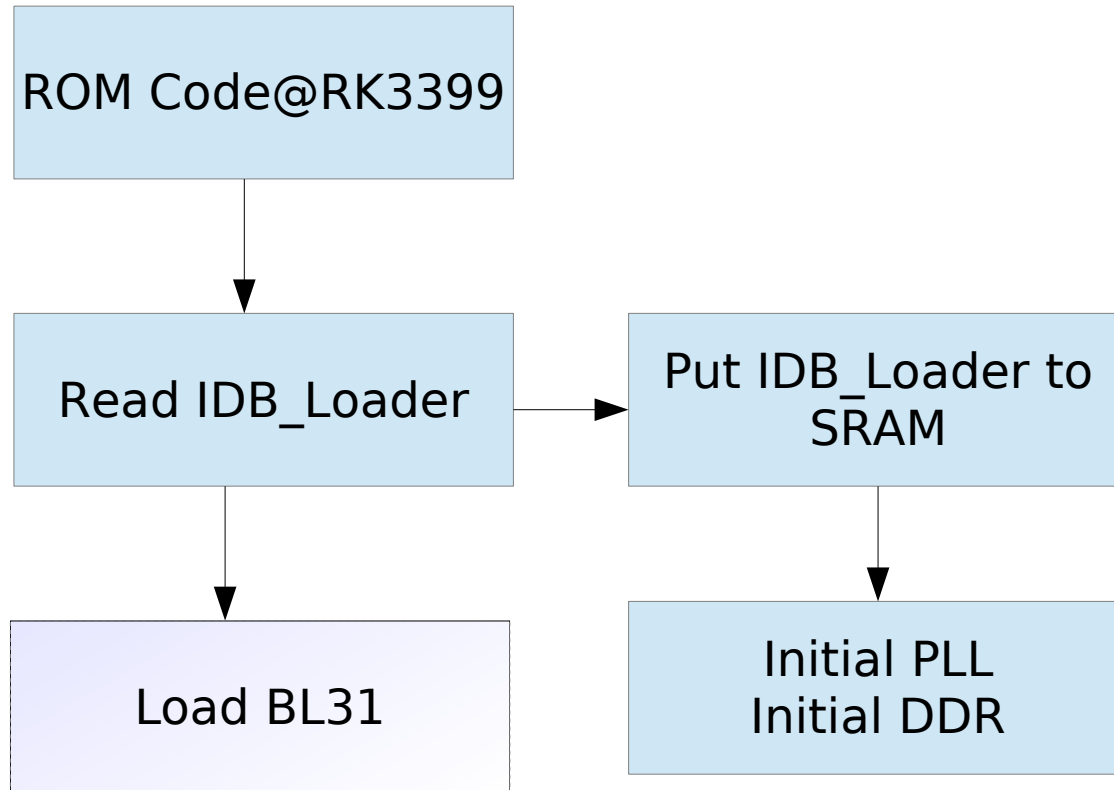
```
=> booti ${kernel_addr_r} ${ramdisk_addr_r}:5038020 ${fdt_addr_r}
## Flattened Device Tree blob at 01f00000
   Booting using the fdt blob at 0x1f00000
   Loading Ramdisk to 76db8000, end 7bdf0020 ... OK
   Loading Device Tree to 0000000076d9d000, end 0000000076db718a ... OK
Adding bank: start=0x00200000, size=0x7fe00000

Starting kernel ...

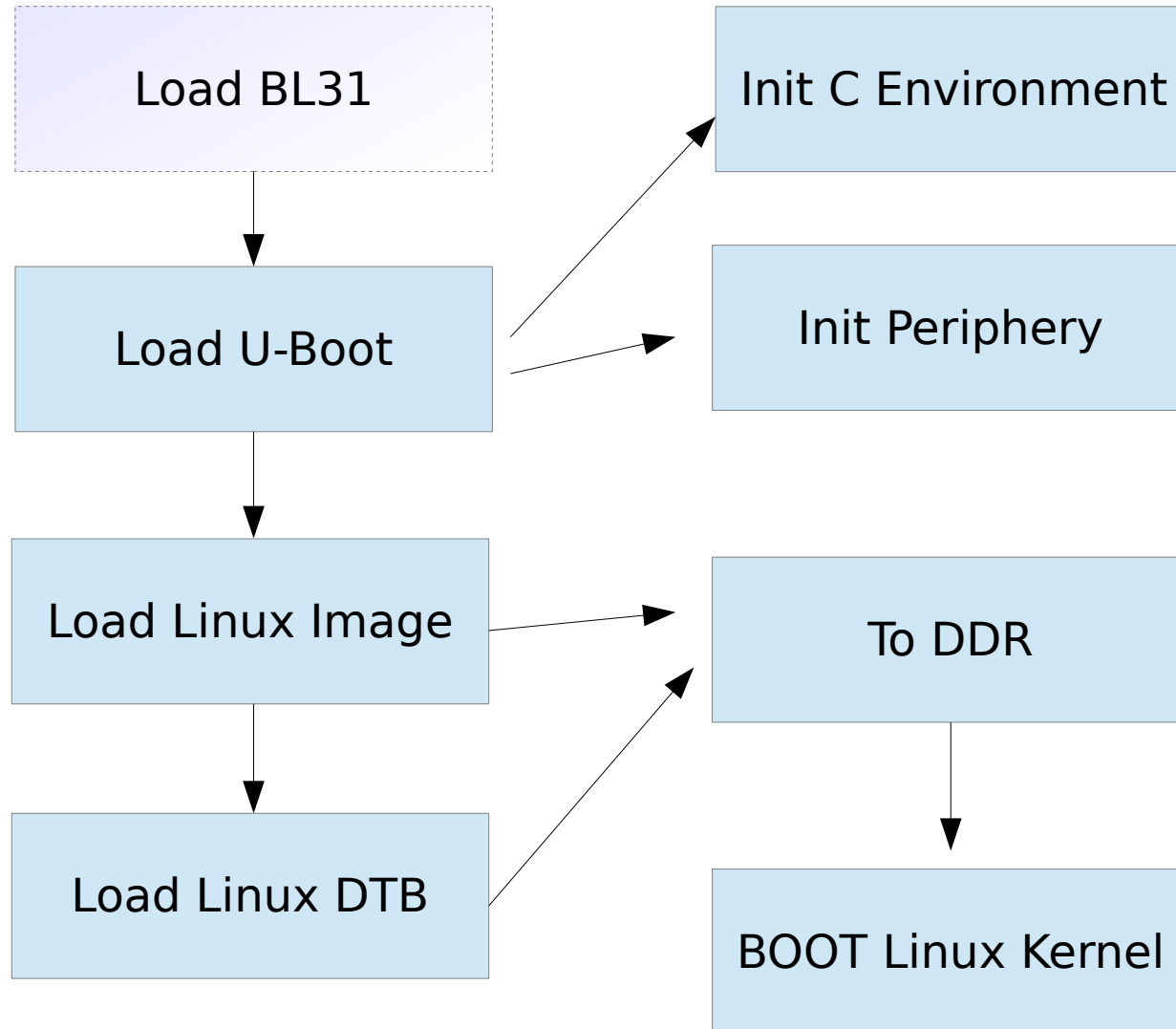
[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Initializing cgroup subsys cpuset
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Initializing cgroup subsys cpuacct
[ 0.000000] Linux version 4.4.154 (slash@slash-ThinkPad-E14-Gen-2) (gcc version 7.3.1 20180425 [linaro-7.3-2018.05
revision d29120a424ecfbc167ef90065c0eeb7f91977701] (Linaro GCC 7.3-2018.05) ) #4 SMP Sun Jan 23 15:55:32 CST 2022
```

# Boot Linux kernel

# System Start Up



# System Start Up





# Boot Linux Kernel Command



## bootcmd

- Power On will **auto run content of bootcmd**

```
[CMD] printenv bootcmd
```

```
bootcmd=run distro_bootcmd;boot_android ${devtype} ${devnum};bootrkp;
```



# Boot OS (Linux) Command

## ➤ booti

→ boot Linux **kernel 64 bit standard** kernel **image**

## ➤ boota

→ boot **android** 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)

```
earlyprintk console=ttyFIQ0,1500000n8 rw init=/sbin/init  
rootfstype=ext4 rootwait root=UUID=a54358ce-55c2-4f4f-bf6d-  
7106997cfb8f console=ttyS2,1500000n8
```

- earlycon : early 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 flag)
{
#ifdef CONFIG_ARM64
    void (*kernel_entry)(void *fdt_addr, void *res0, void *res1, void *res2);
    int fake = (flag & BOOTM_STATE_OS_FAKE_GO);

    kernel_entry = (void (*)(void *fdt_addr, void *res0, void *res1, 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 Function to Board Setting File

# evb-rk3399.c

➤ board/rockchip/evb\_rk3399/evb-rk3399.c

➤ Add function to here

```
diff --git a/board/rockchip/evb_rk3399/evb-rk3399.c b/board/rockchip/evb_rk3399/evb-rk3399.c
index 8e3fce4aa4..e5b4041288 100644
--- a/board/rockchip/evb_rk3399/evb-rk3399.c
+++ b/board/rockchip/evb_rk3399/evb-rk3399.c
@@ -30,6 +30,11 @@ int rk_board_init(void)
    struct udevice *pinctrl, *regulator;
    int ret;

+   printf("%s\n", __func__);
+
+   /* user2 led power-off */
+   run_command("gpio clear 125", 0);
+
    /*
     * The PWM does not have dedicated interrupt number in dts and can
     * not get periph_id by pinctrl framework, so let's init them here.
```

# New a Command



# Add Feature

- Add command → cmd/
- 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)
```

# Hello World Command

## Command line interface

Use the menu. <Enter> selects submenus ---> (or empty submenus ----).  
Includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit  
in [ ] excluded <M> module < > module capable

```
[*] Support U-Boot commands
[*] Command Hello World Sample
*- Use hush shell
(=> ) Shell prompt
Autoboot options --->
*** FASTBOOT ***
[*] Fastboot support --->
*** Commands ***
Info commands --->
Boot commands --->
Environment commands --->
Memory commands --->
Compression commands --->
Device access commands --->
Shell scripting commands --->
Network commands --->
[ ] Enable memtester for ddr
Misc commands --->
⌵(+)
```

# Hello World Command

cmd/Makefile

```
obj-$(CONFIG_CMD_USB_MASS_STORAGE) += usb_mass_storage.o
obj-$(CONFIG_CMD_USB_SDP) += usb_gadget_sdp.o
obj-$(CONFIG_CMD_THOR_DOWNLOAD) += thordown.o
obj-$(CONFIG_CMD_XIMG) += ximg.o
obj-$(CONFIG_CMD_YAFFS2) += yaffs2.o
obj-$(CONFIG_CMD_SPL) += spl.o
obj-$(CONFIG_CMD_ZIP) += zip.o
obj-$(CONFIG_CMD_ZFS) += zfs.o

obj-$(CONFIG_CMD_DFU) += dfu.o
obj-$(CONFIG_CMD_GPT) += gpt.o
obj-$(CONFIG_CMD_ETHSW) += ethsw.o
obj-$(CONFIG_CMD_HELLOWORLD) += helloworld.o

# Power
obj-$(CONFIG_CMD_PMIC) += pmic.o
obj-$(CONFIG_CMD_REGULATOR) += regulator.o

obj-$(CONFIG_CMD_BLOB) += blob.o
endif # !CONFIG_SPL_BUILD
```

# Hello World Command

cmd/Kconfig

```
menu "Command line interface"
```

```
config CMDLINE
```

```
bool "Support U-Boot commands"
```

```
default y
```

```
help
```

Enable U-Boot's command-line functions. This provides a means to enter commands into U-Boot for a wide variety of purposes. It also allows scripts (containing commands) to be executed. Various commands and command categories can be individually enabled. Depending on the number of commands enabled, this can add substantially to the size of U-Boot.

```
config CMD_HELLOWORLD
```

```
bool "Command Hello World Sample"
```

```
depends on CMDLINE
```

```
help
```

This option enables the Hello World as command line Sample.