Содержание

Конспект U-Boot v2017 (next-dev) Development Guide

- https://github.com/rockchip-linux/u-boot
- https://github.com/LuckfoxTECH/luckfox-pico

1. Chapter-1 Basic Introduction

1.5 Boot-order

The RK platform currently has two startup methods depending on whether the front-end Loader code is open source:

```
// Preloader closed source
BOOTROM => ddr bin => Miniloader => TRUST => U-BOOT => KERNEL
// The front-end loader is open source
BOOTROM => TPL => SPL => TRUST => U-BOOT => KERNEL
```

TPL is equivalent to ddr bin, and SPL is equivalent to miniloader. The combination of TPL+SPL achieves the same functions as RK closed source ddr.bin+miniloader and can be replaced with each other

1.9 TPL/SPL/U-Boot-proper

U-Boot can use the same set of code to obtain Loaders with three different functions by using different compilation conditions: TPL/SPL/U-Boot-proper

TPL (Tiny Program Loader) and **SPL** (Secondary Program Loader) are loaders at an earlier stage than U-Boot:

- TPL: runs in sram and is responsible for completing ddr initialization;
- SPL: runs in ddr and is responsible for completing the lowlevel initialization of the system and loading of subsequent firmware (trust.img and uboot.img);
- U-Boot proper: runs in ddr, which is what we usually call «U-Boot», it is responsible for booting the kernel;

BOOTROM ⇒ TPL(ddr bin) ⇒ SPL(miniloader) ⇒ TRUST ⇒ U-BOOT ⇒ KERNEL

Further references: doc/README.TPL and doc/README.SPL

```
// U-Boot stage
./u-boot.map        // MAP table file
./uboot.sym     // SYMBOL table file
./u-boot        // ELF file, similar to vmlinux of the kernel
(important!)
./u-boot.dtb        // u-boot’s own dtb file

// SPL stage
```

```
./spl/u-boot-spl.map    // MAP table file
./spl/uboot-spl.sym // SYMBOL table file
./spl/u-boot-spl    // ELF file, similar to vmlinux of the kernel
(important!)
./spl/u-boot-spl.dtb    // spl's own dtb file
 ./spl/uboot-spl.bin    // Executable binary files will be packaged
into loaders for programming

// TPL stage
./tpl/u-boot-tpl.map    // MAP table file
./tpl/u-boottpl.sym     // SYMBOL table file
./tpl/u-boot-tpl    // ELF file, similar to vmlinux of the kernel
(important!)
./tpl/uboot-tpl.dtb     // tpl's own dtb file
./tpl/u-boottpl.bin // Executable binary files will be packaged into
loaders for programming
```

1.12 U-Boot DTS

U-Boot has its own DTS file, and the corresponding DTB file will be automatically generated during compilation and added to the end of u-boot.bin. File directory:

```
arch/arm/dts/
```

The specific DTS file used by each platform is specified through `CONFIG_DEFAULT_DEVICE_TREE` in `defconfig`.

2. Chapter-2 RK Architecture

2.8 Aliases

Some special aliases in U-Boot are different from the definitions in kernel DTS.

eMMC/SD are collectively called mmc devices in U-Boot, and are distinguished by numbers 0 and 1; SD has a higher startup priority than eMMC.

```
mmc1: stands for sd
mmc0: stands for emmc
```

2.20 vendor storage

U-Boot of the RK platform provides a Vendor storage area for users to save SN, MAC and other information. The storage offset is as follows (see details vendor.c)

3. Chapter-3 Compilation and programming

3.1 Preparation

config fragment introduction

Due to the differentiated needs of products on a single platform, one defconfig is no longer sufficient. Therefore, starting from RV1126, config fragment is supported, that is, overlay of defconfig.

For example: CONFIG_BASE_DEFCONFIG=«rv1126_defconfig» is specified in rv1126-emmc-tb.config . When the ./make.sh rv1126-emmc-tb command is executed, rv1126_defconfig will be used to generate .config first, and then rv1126-emmc- The configuration in tb.config overlays .config. This command is equivalent to:

```
make rv1126_defconfig rv1126-emmc-tb.config && make
```

If you want to update the config fragment file, you only need to use ./scripts/sync-fragment.sh . For example:

```
./scripts/sync-fragment.sh configs/rv1126-emmc-tb.config
```

Command effect: diff the configuration differences between the current .config and rv1126_defconfig into the rv1126-emmc-tb.config file.

4. Chapter-4 System module

4.7 DTBO/DTO

In order to facilitate users' understanding of the contents of this chapter, please read the appendix chapter first to clarify the professional terms: DTB, DTBO, DTC, DTO, DTS, FDT

The relationship between them can be described as:

- DTS is a file used to describe FDT;
- DTS can generate DTB/DTBO after being compiled by DTC;
- DTB and DTBO can be merged into a new DTB through DTO operation;

Usually, many users are accustomed to replace the action meaning of the word «DTO» with «DTBO». In the following, we avoid mixing this concept and make it clear: DTO is a verb concept, which represents operation; DTBO is A noun concept that refers to the sub-dtb used for superposition.

For more information on this chapter, please refer to: https://source.android.google.cn/devices/architecture/dto

4.11 HW-ID DTB

U-Boot of the RK platform supports detecting the GPIO or ADC status on the hardware and dynamically loading different Kernel DTBs, which is temporarily called HW-ID DTB (Hardware id DTB) function.

4.11.1 Design principles

Usually hardware design will frequently update versions and some components, such as screens, wifi modules, etc. If each hardware version corresponds to a set of software, maintenance will be more troublesome. Therefore, the HW_ID function is needed to implement a set of software that can adapt to different versions of hardware.

For different hardware versions, the software needs to provide corresponding dtb files, and also provide ADC/GPIO hardware unique values to characterize the current hardware version (for example: fixed adc value, fixed certain GPIO level).

The user packages all the dtb files corresponding to the hardware version into the same resource.img. When U-Boot boots the kernel, it will detect the hardware unique value and find the dtb file matching the current hardware version from resource.img. to kernel

4.11.2 Hardware Reference

Currently, it supports ADC and GPIO methods to determine the hardware version.

ADC reference design

There are reserved voltage dividing resistors on the RK3326-EVB/PX30-EVB motherboard. Different resistor dividing voltages have different ADC values, so that different hardware versions can be determined:

- The supporting MIPI panel has another pull-down resistor reserved:
- Different mipi screens will be configured with different resistance values, and a unique ADC parameter value will be determined with the EVB motherboard.

The ADC calculation method of the current V1 version: the maximum value of the ADC parameter is 1024, which corresponds to the ADC_IN0 pin being directly pulled up to the supply voltage 1.8V. There is a 10K pull-down resistor on the MIPI screen, and the ADC behind the EVB board is connected. =1024*10K/(10K + 51K) =167.8.

GPIO reference design

There is currently no hardware reference design for GPIO, and users can customize it themselves.

8. Chapter-8 Debugging methods

8.3 io command

Function: Read and write memory.

```
// read operation
md - memory display
Usage: md [.b, .w, .l, .q] address [# of objects]

// write operation
mw - memory write (fill)
Usage: mw [.b, .w, .l, .q] address value [count]
```

Read operation. Example: Display 0x10 consecutive data starting from address

0x76000000.

```
=> md.l 0x76000000 0x10
76000000: fffffffe ffffffff ffffffff ffffffff
76000010: ffffffdf ffffffff feffffff ffffffff
76000020: ffffffff ffffffff ffffffff ffffffff
76000030: ffffffff ffffffff ffffffff ffffffff
```

8.10 dm tree

Function: View the binding and probe status between all device-drivers.

10. Chapter-10 SPL

10.1 Firmware boot

The function of SPL is to replace miniloader to complete the loading and booting of trust.img and uboot.img. SPL currently supports booting two types of firmware:

- FIT firmware: enabled by default;
- RKFW firmware: turned off by default and needs to be configured and enabled individually by the user;

10.1.1 FIT firmware

The FIT (flattened image tree) format is a relatively new firmware format supported by SPL and supports multiple image packaging and verification.

FIT uses DTS syntax to describe the packaged image. The description file is u-boot.its, and the final generated FIT firmware is u-boot.itb.

10.3.3 Startup priority

- SPL uses the boot sequence defined by u-boot, spl-boot-order , located in rkxxxx-u-boot.dtsi:
  - `chosen { stdout-path = &uart2; uboot,spl-boot-order = &sdmmc, &sfc, &nandc, &emmc; };`
- Maskrom startup priority:
  - `spi nor > spi nand > emmc > sd`
- Startup priority of Pre-loader(SPL):
  - `sd > spi nor > spi nand > emmc`

10.3.6 pinctrl

Configuration:

```
CONFIG_SPL_PINCTRL_GENERIC=y
CONFIG_SPL_PINCTRL=y
```

drive:

```
./drivers/pinctrl/pinctrl-uclass.c
./drivers/pinctrl/pinctrl-generic.c
./drivers/pinctrl/pinctrl-rockchip.c
```

DTS configuration:

Take sdmmc as an example:

```
&pinctrl { uboot,dm-spl; };

...
```

Precautions:

When SPL enables pinctrl, you need to modify the CONFIG_OF_SPL_REMOVE_PROPS definition in defconfig and delete the pinctrl-0 pinctrl-names field.

11. Chapter-11 TPL

TPL is a Loader at an earlier stage than U-Boot. TPL runs in SRAM and is used to complete the initialization of DRAM instead of ddr bin. TPL is the version with open source code, and ddr bin is the version with closed source code.

11.1 Compilation and packaging

11.1.1 Configuration

12. Chapter-12 FIT

12.3 Platform configuration

12.3.3 Image file

The FIT solution finally outputs two firmwares in FIT format for programming, namely uboot.img (without trust.img) and boot.img, and an SPL file for packaging into a loader.

- uboot.img file
  - uboot.itb = trust + u-boot.bin + mcu.bin(option)
  - uboot.img = uboot.itb * N copies (N is usually 2)
- boot.img file
  - boot.itb = kernel + fdt + resource + ramdisk(optional)
  - boot.img = boot.itb * M copies (M is usually 1)
- MCU configuration
- Firmware compression

- SPL file
- ./fit directory