RV1126/RV1109 Battery Product Developer Guide

ID: RK-KF-YF-397

Release Version: V1.5.0

Release Date: 2020-08-12

Security Level: □Top-Secret □Secret □Internal ■Public

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This document is going to introduce the technical points of developing low-power or quick-start related products(generally they are called battery products in this document) based on RV1126/RV1109 Linux SDK, aiming to help customers quickly get started with battery products, such as: battery IPC, smart doorbell, smart peephole, smart door lock etc.

Product Version

Chip Model	Kernel Version
RV1126/RV1109	Linux 4.19

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Version	Author	Date	Change Description
V1.0.0	Zhichao Yu, Ziyuan Xu, Hans Yang, Tao Huang	2020-12-22	Initial version
V1.1.0	Zhihua Wang	2021-02-02	Add binocular camera introduction
V1.2.0	Fenrir Lin	2021-03-24	Add oem partition introduction
V1.3.0	CWW	2021-04-28	Add secure boot
V1.4.0	Fenrir Lin	2021-07-26	Add Battery EVB configuration instructions
V1.5.0	Zhichao Guo	2021-08-12	1. Add ums configuration instructions 2. Add boot.img packaging instructions

Contents

1. Product Solution Overview

1.1 Battery Product Solution Introduction

A common feature of low power consumption products with batteries is that, in the case of a battery, it needs to be used for as long as half a year or even a year. There are already lots of these type of products, such as: battery IPC, smart peephole, smart doorbell, facial gate and so on.

In order to extend the battery life, when developing this kind of products, we require that the SoC must be power off when the device is not working, and DDR is also completely powered off. When an external condition (such as PIR or Wi-Fi wake up remotely) is triggered, it quickly enters the working mode by fast cold boot. Therefore, the cold boot time has also become a very critical indicator of this product.

RV1126/RV1109 chip adopts 14nm process, operating voltage is 0.8V, compared with the previous generation of chips, power consumption and temperature rise are greatly improved. In addition, RV1126/RV1109 has a special hardware optimization design for fast boot, which can greatly reduce the time of fast boot. For example, RV1126/RV1109 has a built-in hardware decompression module that can quickly decompress rootfs and kernel.

At present, the wake-up methods supported by our battery IPC prototypes are:

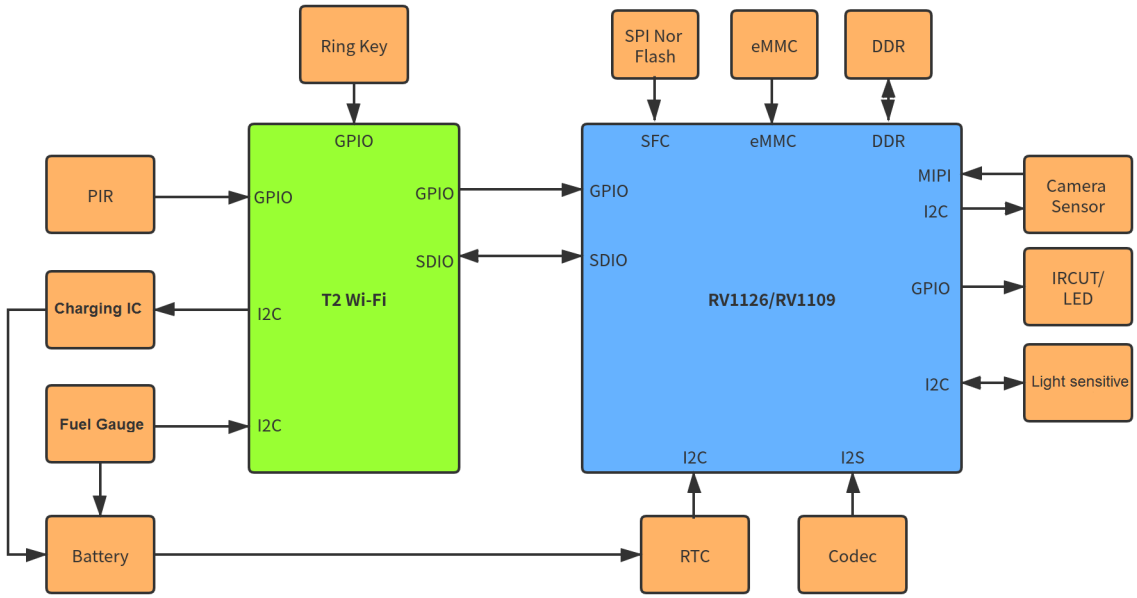
- Press key to wake up
- PIR wake up
- Wi-Fi remote wakeup

1.2 The Advantages of Developing Battery Product by RV1126/RV1109

- 14nm process, low operating voltage and low power consumption;
- Fast cold boot, quick snapshots in 250ms, application services can be started in 350ms;
- ISP2.0, enhanced de-noising module, which can realize clear color images at dim light scene;
- Support H264 smart encoding, which can achieve lower bit rate and higher picture quality;
- Support Wi-Fi low-power keep-alive and remote wake-up mechanism;
- Support full-duplex voice intercom, support 3A algorithm;
- Currently, Alibaba Cloud, Tuya cloud platforms are supported;

1.3 Battery IPC Product Block Diagram

Based on the current RV1126/RV1109 chip, the block diagram of developing battery IPC products is as follows:



In the block diagram, T2 Wi-Fi is a low-power Wi-Fi chip, which integrates a low-power MCU, so that we can save an external low-power MCU. Normally, some low-power logic needs to be executed on the external low-power MCU, so you can perform some low-power logic processing without powering on the controller, such as:

- Control of charging current;
- Judgment of wake-up source;
- Battery power reading;
- PIR enable control;
- Filter for repeated wake-up of PIR;

If customers choose other low-power Wi-Fi without MCU function, we suggest that an external low-power MCU should be considered in the design of the solution.

2. RV1126/RV1109 Fast Boot SDK Introduction

2.1 Fast Boot/Low-power Software Framework Introduction

2.1.1 Basic Board Configuration Introduction

Currently, fast boot related configurations in RV1126/RV1109 SDK are as follows:

Configuration name	Description
BoardConfig-tb-v12.mk	Used to build the board-level configuration of the RV1126 DDR3 EVB V12 board, with eMMC for storage
BoardConfig-tb-v13.mk	Used to build the board-level configuration of the RV1126 DDR3 EVB V13 board, with eMMC for storage
BoardConfig-spi-nor-tb-v13.mk	Used to build the board-level configuration of the RV1126 DDR3 EVB V13 board, with SPI Nor for storage
BoardConfig-dualcam-tb-v13.mk	Used to build the board-level configuration of the RV1126 DDR3 EVB V13 board, with eMMC for storage, and support binocular cameras
BoardConfig-battery-evb.mk	Used to build the board-level configuration of the battery IPC LPDDR4 Demo Board, with eMMC for storage, and it is used for battery IPC products
BoardConfig-snapshot.mk	Used to build the board-level configuration of the battery IPC LPDDR4 Demo Board, with eMMC for storage, and it is used for quick snapshot products

RV1126/RV1109 takes BoardConfig.mk configuration by default. When compiling the above configurations, you need to switch the board-level configuration before compiling. The reference command is as follows:

```
./build.sh BoardConfig-tb-v13.mk && ./build.sh
```

Customers can develop based on the above configuration according to their own product requirements.

2.1.2 Board-level configuration function description

	BoardConfig-battery-evb.mk	BoardConfig-snapshot.mk	BoardConfig-tb-v12/3.mk
Quick start	Support	Support	Support
Low power consumption	Support	Not turned on by default	Not turned on by default
Display	MIPI Panel	MIPI Panel	MIPI Panel
Network	Wi-Fi, Support remote wakeup	Not turned on by default	Ethernet, does not support remote wake-up
Audio	ES8311(One way is MIC, the other way is hardware callback)	Not turned on by default	RK809
Cloud platform	Default Tuya Cloud	Not turned on by default	Default Tuya Cloud
Quick snapshot	Not turned on by default	Support	Not turned on by default

2.1.3 Image Partition Introduction

The fast boot firmware image partition is different from the regular IPC firmware. For the partition configuration, please refer to the files in the device/rockchip/rv1126_rv1109 directory of the SDK:

```
parameter-tb.txt                                  // eMMC fast boot image partition configuration
parameter-spi-nor-tb-32M.txt                     // SPI Nor 32MB fast boot image partition configuration
```

Generally, fast boot will create the following partitions:

- uboot

Note: uboot partition actually packs MCU image and Trust image, they will be loaded by SPL;

The path of the configuration file is: `rkbin/RKTRUST/RV1126TOS_TB.ini`

- boot

Note: it is packaged in fit format, which includes dtb, kernel image and rootfs image. Generally, Rootfs image uses ramdisk, which is preloaded by SPL and decompressed by hardware.

The path of the configuration file is: `device/rockchip/rv1126_rv1109/boot-tb.its`

- userdata(oem)

Note: according to different needs, customers can open a separate read-write partition. Add oem Partition can be used to store files such as cloud curing information, algorithm model library, etc.

2.1.4 Fast Boot Firmware Flashing

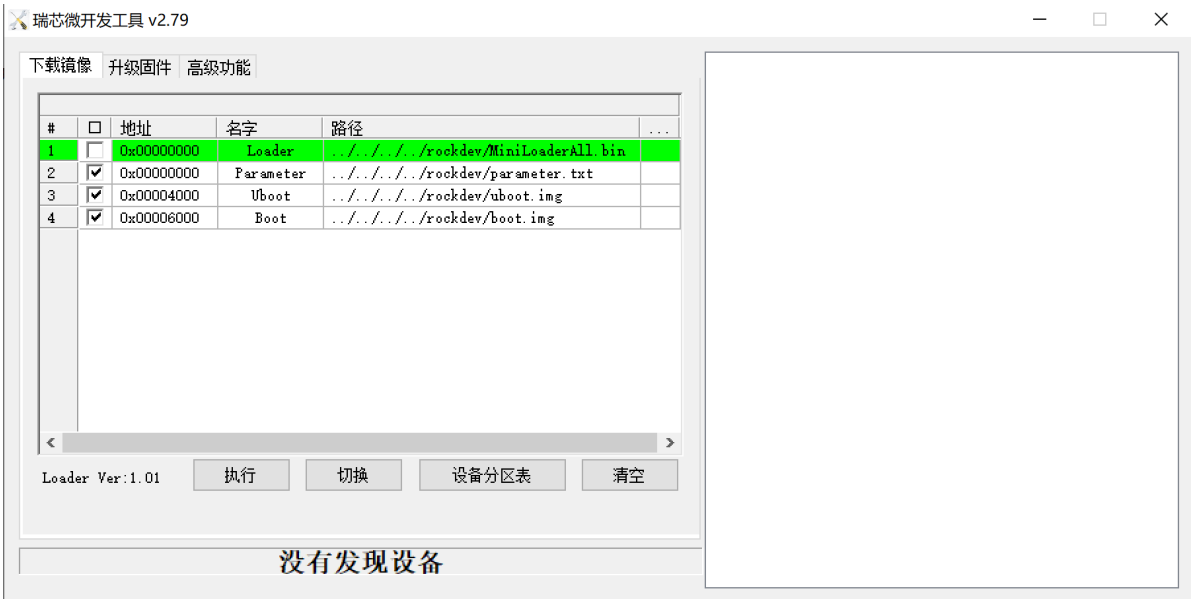
In the Linux system, you can use the following command to flash the whole firmware:

```
upgrade_tool uf rockdev/update.img
```

In the Windows system, you need to import the quick-start flashing configuration, and then do partition flashing. The Windows tool quick-start flashing configuration file is as follows:

```
rv1126_rv1109_tb-config.cfg
```

After importing the configuration, the interface is displayed as follows, generally, we only need to flash Uboot and Boot:



For more detailed instructions on firmware flashing please refer to the document: docs/RV1126_RV1109/Rockchip_RV1126_RV1109_Quick_Start_Linux_EN.pdf

3. Fast Boot

RV1126/RV1109 chipset have built-in hardware decompression module --decom, which can greatly improve the system boot speed. In addition, RV1126/RV1109 has a built-in MCU. The MCU will boot quickly after the SoC is powered on, initialize the camera and ISP quickly, and then save the first few frames as quickly as possible. This chapter mainly introduces the optimization methods and notices of RV1126/RV1109 fast boot.

Currently, fast boot supports the following storage media:

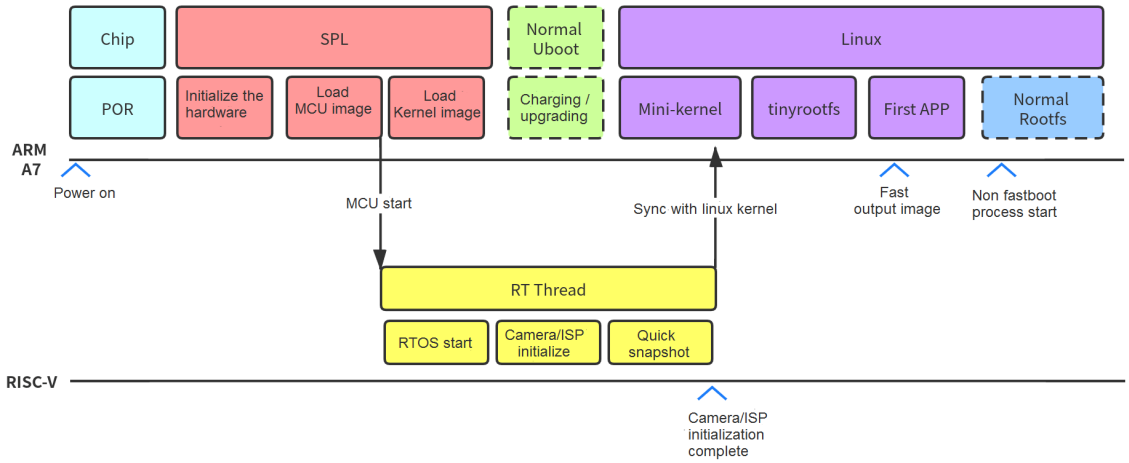
- eMMC
- SPI Nor Flash

Different storage media have different reading speeds for the kernel and rootfs image. The following are the typical reading speed of different storage media. Therefore, we do not recommend using SPI Nand Flash as a fast boot storage medium:

Storage media type	**Read speed**	**Whether fast boot is supported**	**Reference board configuration**
eMMC	120MB/S	Yes	BoardConfig-tb-v13.mk
SPI Nor Flash	30MB/S	Yes	BoardConfig-spi-nor-tb-v13.mk
SPI Nand Flash	10.8MB/S	No	Not Support

3.1 Basic Process of Fast Boot

The basic process of RV1126/RV1109 fast boot is shown in the figure below:



As seen from the basic process in the figure, the boot process of the fast boot version does not run Uboot normally, and the kernel, rootfs, and MCU systems are all loaded through SPL. The configuration of kernel and rootfs has been greatly simplified.

RV1126/RV1109 takes many methods to optimize fast boot, the typical optimization methods are as follows:

- The kernel and rootfs are loaded through SPL, and Uboot is cropped;
- The kernel and rootfs are decompressed by hardware;
- MCU is used to initialize ISP/Camera;
- Drives initialize in parallel;
- Cropped kernel;
- Cropped rootfs;
- Algorithm model pre-loading;
- Parallel initialization of the main process in user mode;
- Wi-Fi network connection optimization;

Next, we will give a more in-depth introduction to the above optimization methods.

3.2 BootRom Support for Fast Boot

At present, the SPI Nor driver of bootrom supports four-channel DMA mode to load the lower-level firmware, which has been configured directly when using USBPlug for flashing firmware, and customers do not need to configure it again. eMMC currently does not have this optimization.

3.3 SPL Fast Boot Mechanism Introduction

Under U-Boot SPL, it supports fast boot in fit format, as well as pressing keys to enter loader mode and low battery detection.

Configuration:

```
CONFIG_SPL_KERNEL_BOOT=y                    // Turn on the fast boot function
CONFIG_SPL_BLK_READ_PREPARE=y               // Turn on the preload function
CONFIG_SPL_MISC_DECOMPRESS=y                // Turn on the decompression
function
CONFIG_SPL_ROCKCHIP_HW_DECOMPRESS=y
```

U-Boot SPL supports the preload function. After the preload function is enabled, the firmware can be loaded while executing other programs. Currently, it is mainly used to preload ramdisk.

For example, to preload the ramdisk compressed by gzip, the compression command is:

```
cat ramdisk | gzip -n -f -9> ramdisk.gz
```

The configuration of the "its" file is as follows:

```
ramdisk {
        data = /incbin/("./images-tb/ramdisk.gz");
        compression = "gzip";       // compression format
        type = "ramdisk";
        arch = "arm";
        os = "linux";
        preload = <1>;              // preload flag
        comp = <0x5800000>;         // load address
        load = <0x2800000>;         // Decompression address
        decomp-async;               // Asynchronous decompression
        hash {
                algo = "sha256";
                uboot-ignore = <1>; // No hash check
        };
};
```

Build the firmware, take building the RV1126 eMMC firmware as an example:

```
./make.sh rv1126-emmc-tb && ./make.sh --spl
```

3.4 Kernel Fast Boot Mechanism Introduction

Configuration:

```
CONFIG_ROCKCHIP_THUNDER_BOOT=y                  // Enable the fast boot function
CONFIG_ROCKCHIP_THUNDER_BOOT_MMC=y              // Enable eMMC fast boot optimization function
CONFIG_ROCKCHIP_THUNDER_BOOT_SFC=y              // Enable SPI Nor fast boot optimization function
CONFIG_VIDEO_ROCKCHIP_THUNDER_BOOT_ISP=y        // Enable ISP fast boot optimization function
```

In order to boot quickly, SPL will not modify the parameters of the kernel dtb based on the actual hardware parameters, so some parameters need to be configured by yourselves. The detailed parameters that need to be configured are:

- Memory
- The size of ramdisk before and after decompression

For details, please refer to: `kernel/arch/arm/boot/dts/rv1126-thunder-boot.dtsi`

```
memory: memory {
        device_type = "memory";
```

```
        reg = <0x00000000 0x20000000>;       // Actual memory size
};

reserved-memory {
        trust@0 {
                reg = <0x00000000 0x00200000>;   // trust space
                no-map;
        };

        trust@200000 {
                reg = <0x00200000 0x00008000>;
        };

        ramoops@210000 {
                compatible = "ramoops";
                reg = <0x00210000 0x000f0000>;
                record-size = <0x20000>;
                console-size = <0x20000>;
                ftrace-size = <0x00000>;
                pmsg-size = <0x50000>;
        };

        rtos@300000 {
                reg = <0x00300000 0x00100000>;        // Reserved for users, can be deleted if not used
                no-map;
        };

        ramdisk_r: ramdisk@2800000 {
                reg = <0x02800000 (48 * 0x00100000)>; // The source address and size of the decompression can be changed according to the actual size
        };

        ramdisk_c: ramdisk@5800000 {
                reg = <0x05800000 (20 * 0x00100000)>; // Compression source address and size, can be changed according to the actual size
        };

        rkisp_thunderboot: rkisp@08000000 {
                reg = <0x08000000 (144 * 0x00100000)>; // The reserved memory for offline frames is used for MCU snapshot, The size of reserved memory can be changed according to the actual needs. and it is recommended to delete it when MCU quick snapshot function is not needed.
        };
};
```

Configuration for eMMC:

```
/ {
        reserved-memory {
                mmc_ecsd: mmc@20f000 {
                        reg = <0x0020f000 0x00001000>;   // SPL uploads the ecsd area to kernel
                };

                mmc_idmac: mmc@500000 {
```

```
                        reg = <0x00500000 0x00100000>;    // When ramdisk is
preloaded, idmac memory area is reserved. After the preload is completed,
the memory in this area is released
                };
        };

        thunder_boot_mmc: thunder-boot-mmc {
                compatible = "rockchip,thunder-boot-mmc";
                reg = <0xffc50000 0x4000>;
                memory-region-src = <&ramdisk_c>;
                memory-region-dst = <&ramdisk_r>;
                memory-region-idmac = <&mmc_idmac>;
        };
};
```

Configuration for SPI Nor:

```
/ {
        thunder_boot_spi_nor: thunder-boot-spi-nor {
                compatible = "rockchip,thunder-boot-sfc";
                reg = <0xffc90000 0x4000>;
                memory-region-src = <&ramdisk_c>;
                memory-region-dst = <&ramdisk_r>;
        };
};
```

3.5 Driver Parallel Loading Mechanism

In order to make full use of the advantages of multi-core, initcalls of the same level are run in parallel during the kernel boot process of fast boot.

The function requires `CONFIG_INITCALL_ASYNC=y`, which is enabled by default in `arch/arm/configs/rv1126-tb.config`.

The kernel parameter `initcall_nr_threads` controls parallel threads. `initcall_nr_threads <0` means to use the default number of threads of the algorithm; `initcall_nr_threads == 0` means to turn off this function; `initcall_nr_threads> 0` means to use the specified number of threads.

The kernel default bool parameter `initcall_debug` controls whether to print initcall debugging information. For details, please refer to `Documentation/admin-guide/kernel-parameters.txt`.

Through these two parameters, you can observe the initcall calling sequence, time-consuming and other information. In the dts provided by default, these two parameters are set to `initcall_nr_threads=-1 initcall_debug=0`, that is, the algorithm default thread number is used, and debugging information printing is disabled.

Modifications that may be required to use this feature are:

1. If the two drivers A and B actually have a dependency (for example, the initialization process of B requires a variable initialization of A to be completed), but the initcall is at the same level, it will appear that B is initialized earlier than A. The **kernel crash** occurred due to access to uninitialized variables. At this time, you need to manually adjust the initcall level of the two to ensure that the level of A is earlier than that of B. The reason why there was no problem before is that although the code was written incorrectly, the compiler puts the initialization function of A before B, which in fact guarantees the order. For the detailed

level usage, please refer to `include/linux/init.h`, pure_initcall is the earliest, and late_initcall_sync is the latest.

2. The regulator driver needs to complete driver registration before subsys_initcall_sync, so that other drivers that need regulator will not fail to probe for cannot get the regulator. Please refer to:

```
drivers/regulator/rk808-
regulator.c:subsys_initcall(rk808_regulator_driver_init);
drivers/regulator/pwm-
regulator.c:subsys_initcall_sync(pwm_regulator_driver_init);
```

3. The camera driver needs to be in front of device_initcall (usually registered through module_i2c_driver) instead of using device_initcall_sync. Because no other drivers are initialized at the moment of device_initcall_sync and parallel is not possible. Please refer to:

```
drivers/media/i2c/ov02k10.c:module_i2c_driver(ov02k10_i2c_driver);
drivers/media/i2c/sc2232.c:module_i2c_driver(sc2232_i2c_driver);
drivers/media/i2c/sc4238.c:module_i2c_driver(sc4238_i2c_driver);
drivers/media/i2c/sc2310.c:module_i2c_driver(sc2310_i2c_driver);
drivers/media/i2c/ov02b10.c:module_i2c_driver(ov02b10_i2c_driver);
drivers/media/i2c/os05a20.c:module_i2c_driver(os05a20_i2c_driver);
drivers/media/i2c/os04a10.c:module_i2c_driver(os04a10_i2c_driver);
drivers/media/i2c/ov12d2q.c:module_i2c_driver(ov12d2q_i2c_driver);
```

3.6 Introduction to the Simplified Kernel

The fast boot kernel is cropped through the config fragment mechanism of kernel. The config fragment will overwrite the configuration of the same name in the defconfig configuration. The path of the fast boot config fragment is as follows:

```
kernel/arch/arm/configs/rv1126-tb.config
```

Assuming that the kernel configuration name of the product currently in use is: rv1126_ipc_defconfig, the command to build kernel using config fragment is as follows:

```
make rv1126_ipc_defconfig rv1126-tb.config
```

Through the above command, a simplified kernel configuration .config will be generated. We can see from the rv1126-tb.config file that most of the kernel functions that are not needed are set to be disabled, and most of the required drivers are configured as m, that is, the modules are loaded after the rootfs is loaded, which can speed up the boot speed.

Take rv1126-battery-evb.config as an example, the operation to update the kernel configuration is as follows:

```
make ARCH=arm rv1126_defconfig rv1126-tb.config
cp .config rv1126.config
make ARCH=arm rv1126_defconfig rv1126-tb.config rv1126-battery-evb.config
ARCH=arm make menuconfig
# to modify
make savedefconfig
scripts/diffconfig -m rv1126.config .config > arch/arm/configs/rv1126-
battery-evb.config
```

3.7 Introduction to Compact Version of rootfs

The size of the rootfs will greatly influence the speed of reading and decompressing rootfs image when booting. In the fast boot, our rootfs adopts the ramdisk file system, we recommend to control the ramdisk file size as much as possible. Some drivers ko, executable programs, and libraries that can be loaded with delay need to be placed in other partitions.

The rootfs configuration reference for fast boot is in the following path:

```
buildroot/configs/rockchip_rv1126_evb_tb_defconfig
```

It can be seen that the rootfs configuration for fast boot has been greatly simplified, and many non-essential functions have been closed. Customers can adjust rootfs related configuration according to their own needs.

It should be noted that the busybox command is also cropped according to the needs of fast boot. The configuration of busybox for fast boot is as follows:

```
buildroot/board/rockchip/common/tinyrootfs/busybox.config
```

At present, the maximum supported size of SDK rootfs is 20MB after compression, and the size after decompression is 48MB. Customers can refer to the previous introduction to modify this memory size according to actual product needs to avoid waste of DDR or insufficient memory allocated:

```
vim kernel/arch/arm/boot/dts/rv1126-thunder-boot.dtsi
```

```
ramdisk_r: ramdisk@2800000 {
        reg = <0x02800000 (48 * 0x00100000)>;        // The decompression source address and size can be changed according to the actual size
};

ramdisk_c: ramdisk@5800000 {
        reg = <0x05800000 (20 * 0x00100000)>;        // Compression source address and size, can be changed according to the actual size
};
```

If you are not sure about the size of your rootfs image, you can check it with the following command:

```
me@my-ubuntu:~/rv1126/sdk$ ls -al buildroot/output/rockchip_rv1126_evb_tb/images/
-rw-r--r-- 1 me me 10777600 January 14 19:31 ramboot.img
-rw-r--r-- 1 me me 16209920 January 14 19:31 rootfs.romfs
            // size before compression
-rw-r--r-- 1 me me 6634740 January 14 19:31 rootfs.romfs.gz
              // size after compression
-rw-r--r-- 1 me me 16465920 January 14 19:31 rootfs.tar
```

3.8 Quick Snapshot

After the device is powered on, the MCU will immediately collect the data of the first few frames of the camera into DDR. After the device system is started, the application can actively send the data to ISP+ISPP for processing. The mechanism by which ISP/ISPP reads RKRAW data from DDR for processing is called offline frame processing.

3.8.1 MCU Boot Mechanism

When the chip is powered on, MCU is in reset state, which need ARM load firmware for it, and then cancel the reset signal, MCU is powered on and executed, and at the same time ARM starts Linux.

The MCU firmware is located in: rkbin\bin\rv11\rv1126_riscv_sc210iot_1920_1080_SBGGR10_compact.bin, please delete rv1126_riscv_v1.02.bin before compiling,and modify the name of rv1126_riscv_sc210iot_1920_1080_SBGGR10_compact.bin to rv1126_riscv_v1.02.bin. At present, the firmware only supports 1080P 30fps, and please contact our FAE for supporting other sensors.

The time consumption of each stage of MCU is as follows::

Test conditions	**Time consumption for system reset SENSOR to output images**	**Time consumption to output 10 frames of images**
LP3 924M MCU has no IRCUT and with illumination supplement The system resets directly during normal operation and triggers the oscilloscope to capture time sequence. The storage medium is eMMC	174ms	340ms

The following changes will affect the speed of MCU snapshot:

1. Change DDR frequency
2. Increase the function of MCU system
3. Replace sensor
4. Replace the storage medium

3.8.2 Offline Frame Processing Introduction

After the device is powered on, MCU will immediately collect 10 frames of RAW data from the camera into DDR. After the device system is started, the application can read the previously saved RAW image from DDR and send it to ISP and ISPP for processing and output YUV image. The data format of offline frame processing must be RKRAW format file. The basic process is as follows:

```
DDR -> RKRAW -> ISP -> ISPP -> YUV
```

dts configuration

To add a new rkisp, rksipp virtual nodes:

```
&rkisp_vir1 {
	status = "okay";
};

&rkispp_vir1 {
	status = "okay";
};
```

Buildroot configuration

Before processing offline frames, you need to configure the IQ file of offline frames. You can modify the buildroot configuration: add in buildroot/configs/rockchip_rv1126_rv1109_defconfig:

```
BR2_PACKAGE_CAMERA_ENGINE_RKAIQ_FAKE_CAMERA_IQFILE="sc210iot_YT-SC210-V2_M12-30IRC-2MP-F18.xml"
```

Offline Frame Demo Test Program

The code is located in: `external/rkmedia/examples/rkmedia_fake_vi_test.c`

Offline Frame Test Command

```
rkmedia_fake_vi_test -a /etc/iqfiles -d /dev/video38 -s /dev/v4l-subdev6 -w 1920 -h 1080 -v 1
```

Introduction to test command parameters

- a: specify the path of the iq file FakeCamera.xml
- d: the rkispp_m_bypass/rkispp_scale0/rkispp_scale1/rkispp_scale2 node of FakeCamera
- s: store rkisp-isp-subdev of reserve mem
- w: the width of FakeCamera
- h: the height of FakeCamera
- j: save the data type. When the value is 1, save JPEG data
- y: save the data type. When the value is 1, save YUV data
- v: save the data type. When the value is 1, display locally

3.8.3 FAST AE Function of MCU

Usually, sensor will have a default exposure value and gain value after initialization, but the adaptable scene is single, and the captured image is easily to be overexposed or too dark. If you want to adapt to a variety of environments with different light intensities, you need to assist the algorithm that automatically adjusts the exposure value and gain value after brightness statistics, to do rapid adjustments. We call the adjustment process exposure convergence. The MCU adopts RK's FAST AE algorithm, which can be used with the photosensitive sensor on the hardware for brightness calibration to realize the adjustment function of rapid convergence of exposure between frames. If there is no photosensitive device, the MCU will be the normal exposure in the daytime indoors by default, and after a few frames of exposure convergence to achieve rapid exposure adjustment.

3.9 Fast Boot Optimization

After optimizing through the above methods, current fast boot can achieve the following indicators:

- Using eMMC storage, the fast boot indicators are as follows:

Boot phase	**Achievement of indicators**
Quick snapshot	210ms
Camera streaming out	600ms
Wi-Fi access to IP	1S
Wi-Fi extremely fast streaming	2S

- Using SPI Nor storage, the fast boot indicators are as follows:

Boot phase	**Target achievement status**
Quick snapshot	TBD
Camera streaming out	950ms
Wi-Fi access to IP	1.3S
Wi-Fi extremely fast streaming	2.5S

Note: due to the different configuration of different products, the fast boot indicators will be different from the above-mentioned indicators. Rockchip does not guarantee that any product type can reach the above-mentioned optimization indicators.

4. Secure Fast Boot

Security verification process of fast boot:

- Maskrom verifies loader (including SPL, ddr, usbplug)
- SPL verifies uboot.img (including trust, U-Boot...)
- U-Boot verifies boot.img (including kernel, fdt, ramdisk...)

Note: after flashing the signed MiniLoaderAll.bin, the security verification will be started, and the unsigned firmware cannot be started.

4.1 Configuration

```
# Select the board level configuration
./build.sh BoardConfig-tb-v13.mk
```

Modify the file: device/rockchip/rv1126_rv1109/BoardConfig-tb-v13.mk Add the following configuration:

```
# Enable secure boot-up, JUST use for ramdisk
export RK_RAMDISK_SECURITY_BOOTUP=true
# Set boot.img (kernel + ramdisk) rollback index
export RK_ROLLBACK_INDEX_BOOT=1
# Set uboot.img rollback index
export RK_ROLLBACK_INDEX_UBOOT=1
```

Modify the file: kernel/arch/arm/boot/dts/rv1126-evb-ddr3-v13-tb-emmc.dts Add the following configuration:

```
diff --git a/arch/arm/boot/dts/rv1126-evb-ddr3-v13-tb-emmc.dts
b/arch/arm/boot/dts/rv1126-evb-ddr3-v13-tb-emmc.dts
index a9aad6045f01..5246d2d21ccc 100644
--- a/arch/arm/boot/dts/rv1126-evb-ddr3-v13-tb-emmc.dts
+++ b/arch/arm/boot/dts/rv1126-evb-ddr3-v13-tb-emmc.dts
@@ -17,3 +17,12 @@
                bootargs = "loglevel=0 initcall_nr_threads=-1
initcall_debug=0 printk.devkmsg=on root=/dev/rd0 console=ttyFIQ0
snd_aloop.index=7 driver_async_probe=dwmmc_rockchip rk.root2nd=/de
        };
 };
+
+&power {
+       /delete-node/ pd_crypto@RV1126_PD_CRYPTO;
+};
+
+&crypto {
+       /delete-property/ power-domains;
+       status = "okay";
+};
```

Modify the file: kernel/arch/arm/configs/rv1126-tb.config Add the following configuration:

```
diff --git a/arch/arm/configs/rv1126-tb.config b/arch/arm/configs/rv1126-
tb.config
index 729df48a8cb0..b7b1b5632727 100644
--- a/arch/arm/configs/rv1126-tb.config
+++ b/arch/arm/configs/rv1126-tb.config
@@ -80,3 +80,4 @@ CONFIG_ROMFS_ON_BLOCK=y
 # CONFIG_USB_KBD is not set
 # CONFIG_USB_MOUSE is not set
 CONFIG_VIDEO_ROCKCHIP_THUNDER_BOOT_ISP=y
+CONFIG_ROCKCHIP_THUNDER_BOOT_CRYPTO=y
```

Modify the file: u-boot/configs/rv1126-emmc-tb.config Add the following configuration:

```
diff --git a/configs/rv1126-emmc-tb.config b/configs/rv1126-emmc-tb.config
index 967d78e0b8..d1000dae5d 100644
--- a/configs/rv1126-emmc-tb.config
+++ b/configs/rv1126-emmc-tb.config
@@ -32,3 +32,9 @@ CONFIG_SPL_POWER_LOW_VOLTAGE_THRESHOLD=3400
 # CONFIG_SPL_SPI_FLASH_SUPPORT is not set
 # CONFIG_SPL_SPI_SUPPORT is not set
 CONFIG_TRUST_INI="RV1126TOS_TB.ini"
+CONFIG_ROCKCHIP_CIPHER=y
+CONFIG_SPL_ROCKCHIP_CIPHER=y
```

```
+CONFIG_FIT_SIGNATURE=y
+CONFIG_SPL_FIT_SIGNATURE=y
+CONFIG_FIT_ROLLBACK_PROTECT=y      # Enable boot.img version rollback protection, optional function
+CONFIG_SPL_FIT_ROLLBACK_PROTECT=y  # Enable uboot.img version rollback protection, optional function
```

4.2 Key Generation

Execute the following three commands under the U-Boot project to generate a pair of RSA keys for signing. Normally, you only have to generate the pair of keys once. Because after generation, this pair of keys will be used to sign and verify the firmware, please keep it properly.

```
cd tools/linux/rk_sign_tool
./rk_sign_tool cc --chip 1126
rk_sign_tool kk --out .

cd -
mkdir -p u-boot/keys

cp tools/linux/rk_sign_tool/privateKey.pem u-boot/keys/dev.key
cp tools/linux/rk_sign_tool/publicKey.pem u-boot/keys/dev.pubkey

# Use -x509 and private key to generate a self-signed certificate: keys/dev.crt (it is essentially equivalent to the public key)
cd u-boot
openssl req -batch -new -x509 -key keys/dev.key -out keys/dev.crt
```

```
#If there is no .rnd file in the user directory:
Can't load /home/rv1126/.rnd into RNG
140522933268928:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/randfile.c:88:Filename=/home/rv1126/.rnd

# Please create manually first:
touch ~/.rnd

# ls keys/ to see the results:
dev.crt  dev.key  dev.pubkey
```

> Note: The names of "keys", "dev.key", "dev.crt" and "dev.pubkey" mentioned above are not changeable. Because these names have been statically defined in the its file. If they are changed you will fail to package.

4.3 Build

```
./build.sh ramboot   # Build kernel and ramdisk, generate unsigned boot.img
./build.sh uboot     # Build uboot and loader, and sign the firmware
```

Or use the one-click build command: `./build.sh`

4.3.1 Package boot.img separately

The SDK packages the kernel and ramdisk as boot.img in fit format. If customers need to flash the kernel separately, please package zboot.img and rootfs together as boot.img. Run the following command to package boot.img.

```
cat kernel/arch/arm/boot/Image | gzip -n -f -9 >
kernel/arch/arm/boot/Image.gz
device/rockchip/common/mk-fitimage.sh rockdev/boot.img
device/rockchip/rv1126_rv1109/boot-tb.its `realpath rootfs.romfs.gz`
`realpath kernel/arch/arm/boot/Image.gz`
```

4.4 Firmware Generation

```
ls rockdev/
boot.img  MiniLoaderAll.bin  oem.img  parameter.txt  uboot.img  userdata.img
```

For detailed secure verification functions, please refer to the FIT chapter in this document: Rockchip_Developer_Guide_UBoot_Nextdev_EN.pdf.

5. Power Optimization

When designing low-power battery product, power consumption is a very important indicator. This chapter will focus on the methods and notices about power optimization.

5.1 Introduction to Power Optimization

About power optimization, we have summarized the following methods at present:

- According to product requirements, reduce the actual operating voltage and frequency of some modules, such as CPU, DDR, VEPU and NPU;
- For all Clock and Power, turn off unnecessary modules, and only keep 3 PLLs;
- Deep optimization of DDR bandwidth, reducing memory copy, and turning off unnecessary functions;
- Optimize software application to reduce CPU usage
- Optimize power consumption in hardware design;

5.2 Frequency Voltmeter Optimization

Different from traditional product solutions, the frequency of battery products is not that the higher the frequency, the better, but the frequency and voltage of the product need to be limited according to the product usage scene, to meet the principle of "the frequency and voltage are enough". Therefore, according to the actual scene of the battery IPC product, we re-examined the frequency voltmeter, and the frequency and voltage of each module have been reduced to a certain extent.

The following table is the frequency voltmeter after optimization:

- Optimized frequency

CPU	DDR	Encoder	ISP	ISPP
600MHz	528MHz	297MHz	396MHz(Max)	396MHz(Max)

- Optimized voltage

VDD_ARM	VDD_LOGIC	VCC_DDR
0.724V	0.769V	1.232V

5.3 System Bandwidth Optimization

In the process of optimizing low-power product solutions, we found that bandwidth has a very large impact on power consumption. According to the power consumption requirements of battery IPC products, we need to reduce bandwidth usage as much as possible. Currently, we use the following methods to optimize DDR bandwidth:

- TNR in 3DNR adopts 2-frame mode;
- Close HDR and enable linear mode;
- Close ISPP Dehaze module;
- Camera frame rate is reduced from 30fps to 25fps;
- Camera data changed from RAW12 to RAW10;
- Video frames should be compressed in FBC format as far as possible;
- Camera MIPI data capture is switched from ISP to VICAP;

After optimization by the above methods, the bandwidth of 200M Wi-Fi streaming is reduced obviously. The following is the bandwidth data before and after optimization(Sensor: 200M@25FPS):

Bandwidth before optimization	Bandwidth after optimization
1368MB/S	870MB/S

The particular tool rk-msch-probe for bandwidth testing can be obtained from Rockchip FAE.

5.4 Method of Enabling SoC Clock and Ppower Domain Optimization

BoardConfig-tb-v12/3.mk does not enable SoC clock and power domain optimization functions by default. You can enable this function by enabling the kernel configuration.

```
+CONFIG_ROCKCHIP_LOW_PERFORMANCE=y
```

Note: This optimization turns off the CPLL (500MHz) of the SoC, causing the Gigabit network function to fail to accurately divide the frequency to 125MHz, causing abnormal Ethernet functions.

5.5 Hardware Design Considerations on Power Optimization

5.5.1 Discrete Power Supply Design

Discrete power supply design has more advantages than using PMU (like: RK809) in the following aspects:

- You can choose DCDC or LDO with higher power efficiency to optimize power consumption more flexibly;
- Save RK809's default anti-shake time of 500ms, and speed up boot speed;
- It is easier to control cost;

Therefore, the reference power supply design solution of a low-power battery product released by Rockchip is discrete power supply.

5.5.2 Hardware Selection Guidelines

When developing low-power products, the power consumption of peripherals also needs to be evaluated. In order to help customers quickly achieve the purpose of low-power in their own solutions, we provide a list of devices that we currently consider to have relatively low power consumption. Customers can choose according to your own product (other peripherals not listed here do not mean that we do not support them, but their power consumption is at a normal level and does not need to be listed here):

Camera selection list

Sensor model	Resolution	Reference power consumption
SC210IoT	200M	63mW
GC2053	200M	93mW
OS04C10	400M	148mW

Wi-Fi Selection List

Wi-Fi model	Low-power keep-alive power consumption	Wi-Fi push-stream power consumption
CYW43438	350uA	TBD
AP6203	350uA	TBD

Note: the above Wi-Fi low-power keep-alive power consumption is tested in a shielded room with DTIM=10, and the power consumption will be higher in a normal environment.

DDR Selection

In terms of DDR, we chose LPDDR3 and LPDDR4 in our reference design, and their power consumption will be smaller than that of DDR3. The comparison of DDR power consumption is as follows:

DDR3	LPDDR3
2101mW	1809mW

Test condition: based on RV1126 EVB V13, without display, HDR on, NR on, NPU off, VEPU 396MHz, DDR bandwidth 3100±50MB/S.

5.6 RV1126/RV1109 Power Optimization Achievement Status

After optimizing through the aforementioned methods, the current power consumption data that the product solution can achieve is as follows (based on Rockchip IPC prototype test):

Power Consumption Condition	**Achieved Power Consumption (Power Consumption of the Whole Machine)**
Wi-Fi Low Power Keep Alive	1.5mW
1080P@25fps Wi-Fi streaming	720mW
2K@25fps Wi-Fi streaming	TBD
4K@15fps Wi-Fi streaming	1.4W

Note: due to the different configuration of different products, the power consumption indicators will be different from the above indicators. Rockchip does not guarantee that any product type can achieve the above optimization indicators.

6. Common Functions Configuration

The fast boot configuration has been greatly simplified, and customers may probably encounter various problems when debugging, such as the lack of some libraries or tools. The convenience of debugging has a contradictory relationship with the size of the rootfs image. To achieve the fastest boot speed, it will definitely increase the difficulty of product debugging and development. Therefore, this chapter will focus on how to enable and configure some frequently used functions.

6.1 Add MiniGUI Supported

For specific introduction, please refer to the document "Rockchip_RV1126_RV1109_Quick_Start_Linux_EN.pdf".

6.2 Binocular Cameras Support

At present, the SDK already supports the configuration of fast boot + binocular cameras. Based on this configuration, customers can develop smart door locks and other products that require binocular cameras. The configuration files are as follows:

```
BoardConfig-dualcam-tb-v13.mk
```

After compiling, you can use rkmedia_vi_double_cameras_test to preview.

Note:

```
rkmedia_vi_double_cameras_test -a /etc/iqfiles/ -u 0
```

For detailed introduction of rkmedia_vi_double_cameras_test, please refer to the document "Rockchip_Developer_Guide_Linux_RKMedia_EN.pdf".

6.3 Enable Commonly Used Debug Methods

6.3.1 Enable adb

To enable the adb function, please enable the following configuration in buildroot:

```
BR2_PACKAGE_THUNDERBOOT=y
BR2_THUNDERBOOT_USB_ADBD=y
```

6.3.2 Enable iperf

To enable the iperf3 function, please enable the following configuration in buildroot:

```
BR2_PACKAGE_IPERF3=y
```

6.3.3 Enable gdb

To enable the gdb function, please turn on the gdb function in the buildroot configuration file.

```
+#include "gdb.config"
```

6.3.4 Enable U Disk Recognition

To enable the U Disk recognition, please turn on the macro in the kernel configuration file.

```
diff --git a/arch/arm/configs/rv1126-tb.config b/arch/arm/configs/rv1126-tb.config
index 729df48a8cb0..34d7d40c6d82 100644
--- a/arch/arm/configs/rv1126-tb.config
+++ b/arch/arm/configs/rv1126-tb.config
@@ -1,4 +1,3 @@
-CONFIG_BLK_DEV_SD=m
 CONFIG_BT=m
 CONFIG_BT_HCIUART=m
 CONFIG_CFG80211=m
@@ -24,8 +23,6 @@ CONFIG_ROCKCHIP_HW_DECOMPRESS=y
 CONFIG_ROCKCHIP_RAMDISK=y
 CONFIG_ROCKCHIP_THUNDER_BOOT=y
 CONFIG_ROMFS_FS=y
```

```
-CONFIG_SCSI=m
-CONFIG_SCSI_MOD=m
 # CONFIG_SLUB_SYSFS is not set
 CONFIG_SND=m
 CONFIG_SND_ALOOP=m
@@ -47,21 +44,21 @@ CONFIG_SOUND=m
 CONFIG_STMMAC_ETH=m
 CONFIG_STMMAC_PLATFORM=m
 # CONFIG_TEE is not set
-CONFIG_USB=m
+CONFIG_USB=y
 # CONFIG_USB_CONFIGFS_F_UAC1 is not set
 # CONFIG_USB_CONFIGFS_F_UAC2 is not set
 # CONFIG_USB_CONFIGFS_F_UVC is not set
 # CONFIG_USB_CONFIGFS_RNDIS is not set
-CONFIG_USB_DWC3=m
-CONFIG_USB_DWC3_OF_SIMPLE=m
-CONFIG_USB_DWC3_ROCKCHIP_INNO=m
-CONFIG_USB_EHCI_HCD=m
-CONFIG_USB_EHCI_HCD_PLATFORM=m
+CONFIG_USB_DWC3=y
+CONFIG_USB_DWC3_OF_SIMPLE=y
+CONFIG_USB_DWC3_ROCKCHIP_INNO=y
+CONFIG_USB_EHCI_HCD=y
+CONFIG_USB_EHCI_HCD_PLATFORM=y
 CONFIG_USB_HID=m
 # CONFIG_USB_NET_DRIVERS is not set
-CONFIG_USB_OHCI_HCD=m
-CONFIG_USB_OHCI_HCD_PLATFORM=m
-CONFIG_USB_STORAGE=m
+CONFIG_USB_OHCI_HCD=y
+CONFIG_USB_OHCI_HCD_PLATFORM=y
+CONFIG_USB_STORAGE=y
 CONFIG_USB_XHCI_HCD=m
 CONFIG_USB_XHCI_PLATFORM=m
```

6.3.5 Enable RNDIS

To enable the RNDIS, please turn on the macro in the buildroot configuration file.

```
BR2_THUNDERBOOT_USB_RNDIS
```

Turn on the macro in the kernel configuration file.

```
CONFIG_USB_F_RNDIS=y
CONFIG_USB_CONFIGFS_RNDIS=y
```

And then execute.

```
./build.sh kernel && make thunderboot-reconfigure && ./build.sh ramboot
```

6.3.6 Enable UMS

1. To enable the UMS, please turn on the macro in the kernel configuration file.

```
+CONFIG_USB_CONFIGFS_MASS_STORAGE=y
+CONFIG_USB_F_MASS_STORAGE=y
```

2. Make the following modifications to the S50tb_usbdevice script file in buildroot/package/rockchip/thunderboot/.

(Modify parameters related to UMS_BLOCK according to requirements. For details, refer to Rockchip_Quick_Start_Linux_USB_Gadget_EN.pdf.)

```
diff --git a/package/rockchip/thunderboot/S50tb_usbdevice
b/package/rockchip/thunderboot/S50tb_usbdevice
index 40cd9994..2789efe5 100644
--- a/package/rockchip/thunderboot/S50tb_usbdevice
+++ b/package/rockchip/thunderboot/S50tb_usbdevice
@@ -5,6 +5,9 @@
 #
 # Load default env variables from profiles

+UMS_EN=off
 RNDIS_EN=off
 ADB_EN=off
 RNDIS_ADDR=192.168.1.100
@@ -19,6 +22,14 @@
USB_STRINGS_DIR=${USB_CONFIGFS_DIR}/strings/${USB_ATTRIBUTE}
 USB_FUNCTIONS_DIR=${USB_CONFIGFS_DIR}/functions
 USB_CONFIGS_DIR=${USB_CONFIGFS_DIR}/configs/${USB_SKELETON}

+# For VBUS_ALWAYS_ON usb otg is not support ums
+# Since the block to ums is always occupated by USB due to no disconneted
state
+UMS_BLOCK=/userdata/ums_shared.img
+UMS_BLOCK_SIZE=0       #unit M
+UMS_BLOCK_TYPE=fat
+UMS_BLOCK_AUTO_MOUNT=off
+UMS_RO=0

 syslink_function()
 {
        ln -s ${USB_FUNCTIONS_DIR}/$1
${USB_CONFIGS_DIR}/f${USB_FUNCTIONS_CNT}
@@ -31,6 +42,22 @@ bind_functions()
        test $RNDIS_EN = on && syslink_function rndis.gs0
        test $ADB_EN = on && syslink_function ffs.adb

+       if [ $UMS_EN = on ];then
+               echo ${UMS_RO} >
${USB_FUNCTIONS_DIR}/mass_storage.0/lun.0/ro
+               if [ "$UMS_BLOCK_SIZE" != "0" -a ! -e ${UMS_BLOCK} ]; then
+                       dd if=/dev/zero of=${UMS_BLOCK} bs=1M
count=${UMS_BLOCK_SIZE}
+                       mkfs.${UMS_BLOCK_TYPE} ${UMS_BLOCK}
```

```
+                       test $? && echo "Warning: failed to
mkfs.${UMS_BLOCK_TYPE} ${UMS_BLOCK}"
+               fi
+               mkdir /userdata/ums -p
+               if [ $UMS_BLOCK_AUTO_MOUNT = on ];then
+                       mount ${UMS_BLOCK} /mnt/ums
+               else
+                       echo ${UMS_BLOCK} >
${USB_FUNCTIONS_DIR}/mass_storage.0/lun.0/file
+               fi
+               syslink_function mass_storage.0
+       fi

        echo ${CONFIG_STRING} >
${USB_CONFIGS_DIR}/strings/${USB_ATTRIBUTE}/configuration
 }

@@ -43,6 +70,8 @@ function_init()
                mkdir /dev/usb-ffs/adb -m 0770
                mount -o uid=2000,gid=2000 -t functionfs adb /dev/usb-
ffs/adb
        fi

+       mkdir ${USB_FUNCTIONS_DIR}/mass_storage.0
 }

 configfs_init()
@@ -90,6 +119,39 @@ make_config_string()
        fi
 }

+parse_parameter()
+{
+       # find name and var
+       NAME=`echo $1 | awk -F "=" '{print $1}'`
+       VAR=`echo $1 | awk -F "=" '{print $2}'`
+
+       case "$NAME" in
+               ums_block)
+                       UMS_BLOCK=${VAR}
+                       ;;
+               ums_block_size)
+                       if [ ! "$VAR" -gt 0 ] 2>/dev/null ;then
+                               echo "$VAR is not a number"
+                               exit 1
+                       fi
+                       UMS_BLOCK_SIZE=${VAR}
+                       ;;
+               ums_block_type)
+                       UMS_BLOCK_TYPE=${VAR}
+                       ;;
+               ums_block_auto_mount)
+                       UMS_BLOCK_AUTO_MOUNT=${VAR}
+                       ;;
+               ums_ro)
+                       if [ "$VAR" != "off" ]; then
+                               echo "Set UMS read-only"
+                               UMS_RO=1
```

```
+                       fi
+                               UMS_RO=0
+                       ;;
+       esac
+}

 parameter_init()
 {
        while read line
@@ -99,18 +161,33 @@ parameter_init()
                                ADB_EN=on
                                make_config_string adb
                                ;;
+                       usb_ums_en)
+                               UMS_EN=on
+                               make_config_string ums
+                               ;;
                        usb_rndis_en)
                                RNDIS_EN=on
                                make_config_string rndis
                                ;;
+                       *)
+                               parse_parameter ${line}
+                               ;;
                esac
        done < $USB_CONFIG_FILE

-       if [ "$CONFIG_STRING"x = "adb"x ];then
-               PID=0x0006
-       else
-               PID=0x0019
-       fi
+       case "$CONFIG_STRING" in
+               ums)
+                       PID=0x0000
+                       ;;
+               adb)
+                       PID=0x0006
+                       ;;
+               ums_adb | adb_ums)
+                       PID=0x0018
+                       ;;
+               *)
+                       PID=0x0019
+       esac
 }
```

3. Modify buildroot/xxxxx/target/etc/preinit.d/.usb_config.

```
usb_adb_en
+usb_ums_en
```

6.4 Add oem Partition

At present, the configurations of EVB fast boot and battery IPC have enabled the oem partition by default. If other configurations want to enable the oem partition, please complete the following steps.

The following operation takes battery IPC as an example, other configurations need to modify the corresponding files in the same path.

6.4.1 Modify Buildroot Configuration

In buildroot/configs/rockchip_rv1126_battery_ipc_defconfig, add:

```
BR2_PACKAGE_RK_OEM=y
BR2_PACKAGE_RK_OEM_RESOURCE_DIR="$(TOPDIR)/../device/rockchip/oem/oem_battery_ipc"
BR2_PACKAGE_RK_OEM_IMAGE_FILESYSTEM_TYPE="ext2"
```

6.4.2 Modify BoardConfig

In device/rockchip/rv1126_rv1109/BoardConfig-battery-ipc.mk, add:

```
# Set oem partition type, including ext2 squashfs
export RK_OEM_FS_TYPE=ext2
# OEM config
export RK_OEM_DIR=oem_battery_ipc
# OEM build on buildroot
export RK_OEM_BUILDIN_BUILDROOT=YES
```

6.4.3 Modify Parameter

In device/rockchip/rv1126_rv1109/parameter-tb.txt, add the oem partition information.

```
CMDLINE:mtdparts=rk29xxnand:0x00002000@0x00004000(uboot),0x00010000@0x00006000(boot),0x00020000@0x00016000(oem),-@0x00036000(userdata:grow)
```

6.4.4 Modify package-file

In tools/linux/Linux_Pack_Firmware/rockdev/package-file, add:

```
oem Image/oem.img
```

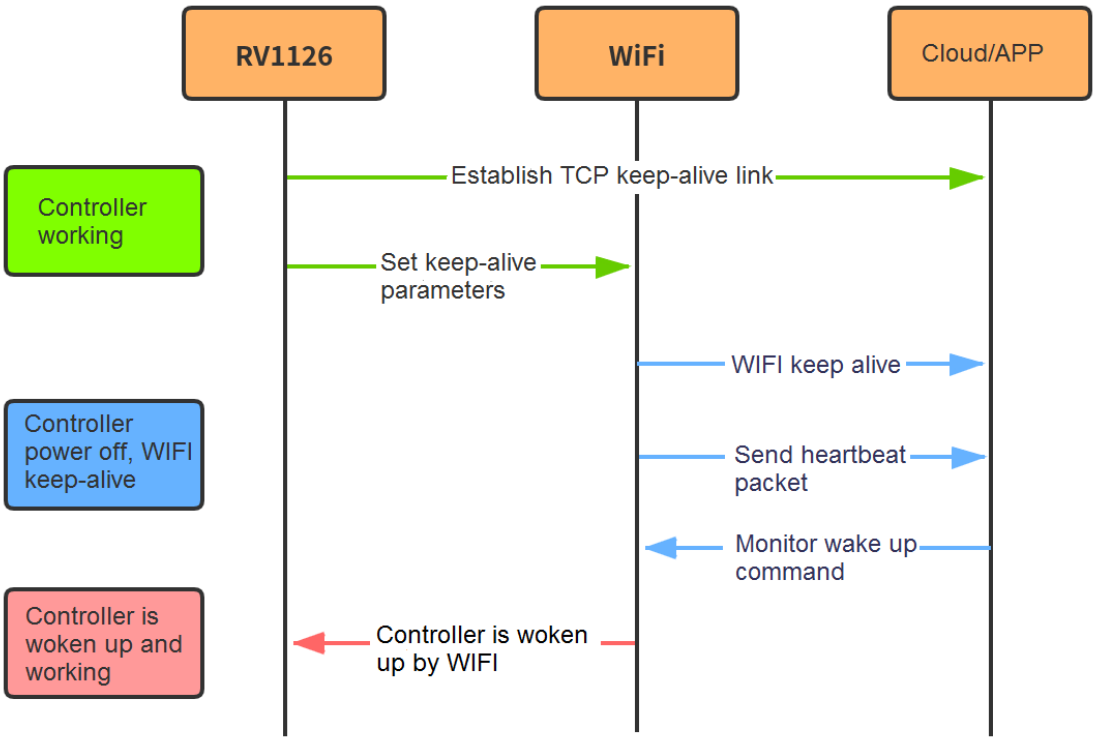
6.4.5 Boot Script to Mount oem Partition

Please refer to buildroot/package/rockchip/thunderboot/S07mountall.

7. Introduction to Wi-Fi Keep-alive and Wake-up Remotely

Low-power battery products pay great attention to the portability of the product. Therefore, such products often use Wi-Fi to transmit control commands or video streaming data. When the device is not working, the SoC is in a power-down state. At this time, in order to keep the device online, the Wi-Fi must be in a low-power keep-alive mode. Wi-Fi will wake up regularly to receive wake-up packets from the cloud in the low-power keep-alive mode. When users need to see the video on the device through the mobile phone, the cloud will send a wake-up packet to Wi-Fi. After Wi-Fi receives the wake-up packet, it will power on the SoC through GPIO, and then the SoC will quickly start and push the video stream to users.

The key process of Wi-Fi low power keep-alive and remote wake-up is as follows:



This chapter will focus on the development of Wi-Fi for low-power battery products.

7.1 Wi-Fi Configuration

Currently, RV1126/RV1109 supports the following network configuration ways:

- Command line network configuration;
- QR code network configuration;

7.1.1 Command Line Network Configuration

```
cp /etc/wpa_supplicant.conf /tmp
sed -i "s/SSID/wifi_name/g" /tmp/wpa_supplicant.conf
sed -i "s/PASSWORD/wifi_password/g" /tmp/wpa_supplicant.conf
wpa_supplicant -B -i wlan0 -c /tmp/wpa_supplicant.conf
udhcpc -i wlan0
```

7.1.2 QR Code Network Configuration

1. **Example of using QR code**

Please refer to docs/RV1126_RV1109/Rockchip_Instruction_Linux_Battery_IPC_CN.pdf for details.

2. **Introduction to QR code scanning device interface**

Please refer to docs/RV1126_RV1109/ApplicationNote/Rockchip_Instructions_Qrcode_CN.pdf for details.

7.2 Wi-Fi Low-power Keep-alive

The process of Wi-Fi low-power keep-alive solution is as follows:

1. Normal state: firstly, RV1126 IPC device establishes a video stream connection with the cloud. When the video stream is finished, or the cloud actively disconnects the live stream, the device will establish a TCP keep-alive connection with the cloud and send heartbeat packets periodically **(The content is a customize string, such as "heartbeat")** to the cloud to ensure that the cloud always knows that the device is online;
2. Trigger to sleep: the sleep state is triggered by the physical buttons of IPC camera device or the camera program logic (when detecting idle), etc.;
3. Before going to sleep: use a WCM API to transfer the "the latest TCP keep-alive connection parameters" in step one to Wi-Fi, and Wi-Fi will continue to maintain this keep-alive TCP connection and send heartbeat packets (" heartbeat"); parameters include (for example):

```
    struct tcp_keepalive_conn {
        struct ether_addr dst_mac;   /* Destinition Mac */
        struct ipv4_addr  src_ip;    /* Sorce IP */
        struct ipv4_addr  dst_ip;    /* Destinition IP */
        uint16 ipid;                 /* Ip Identification */
        uint16 srcport;              /* Source Port Address */
        uint16 dstport;              /* Destination Port Address */
        uint32 seq;                  /* TCP Sequence Number */
        uint32 ack;                  /* TCP Ack Number */
        uint16 tcpwin;               /* TCP window */
        uint32 tsval;                /* Timestamp Value */
        uint32 tsecr;                /* Timestamp Echo Reply */
        uint32 len;                  /* last packet payload len */
        uint32 ka_payload_len;       /* keep alive payload length  eg:
sizeof(heartbeat)*/
        uint8  ka_payload[1];        /* keep alive payload eg: heartbeat*/
    } tcpka_conn_t;
    //Obtain specific TCP connection parameters: You can add trace function
to linux tcp/ip protocol stack to obtain specific tcp connection
parameters,like tcpdump.
```

Then set the wake-up package content through another WCM API function (for example, "WakeUp", when Wi-Fi receives the same content as "WakeUp", wake up RV1126 device through GPIO)

4. Device sleep: RV1126 IPC camera device is power off, only the Wi-Fi chip is running and maintains a TCP heartbeat in the cloud;
5. Wake up the device: the cloud sends the wake-up package "WakeUp" to Wi-Fi through the TCP keep-alive connection, and Wi-Fi wakes up the RV1126 device through GPIO;
6. Exit sleep: RV1126 IPC device is triggered to turn on, re-establishes the connection to the cloud video streaming, and returns to the normal state of step 1;

7.3 Low-power Wi-Fi Support List

The low-power Wi-Fi modules currently supported by the SDK ar as follows:

Wi-Fi model	**Low-power keep-alive power consumption**	**Wi-Fi streaming power consumption**
CYW43438	350uA	TBD
AP6203	350uA	TBD

7.4 Cypress Wi-Fi Solution Introduction

7.4.1 Cypress Wi-Fi Configuration

```
##Drive Code:
kernel/drivers/net/wireless/rockchip_wlan/cywdhd/bcmdhd

#buildroot Configuration
BR2_PACKAGE_RKWIFIBT_AWNB197 = y

#Choose corresponding driver:
--- a/arch/arm/configs/rv1126-battery.config
+++ b/arch/arm/configs/rv1126-battery.config
@@ -1,5 +1,7 @@
 # CONFIG_AP6XXX is not set
 CONFIG_AP6XXX_INDEP_POWER=m
+CONFIG_CYW_BCMDHD=m
+CONFIG_SDIO_CIS_CYW43438=y
 CONFIG_BATTERY_CW2015=yba
 CONFIG_CHARGER_GPIO=y
 CONFIG_LEDS_PWM=y
```

7.4.2 Keep-alive Application Process Introduction

Cypress provides a complete API to setting Wi-Fi, the reference code is located in:

```
external/rkwifibt/src/CY_WL_API/
```

API Introduction

```
int WIFI_Init(void) // Wi-Fi initialization
void WIFI_Deinit(void) // Wi-Fi deinitialization
int WIFI_Connect(char* ssid_name, char* password, int useip) //Wi-Fi connection, userip parameter is not used temporarily
int WIFI_GetStatus(void) //Get connection status
int WIFI_GetWakupReason(void) //Get Wi-Fi wakeup reason
int WIFI_ClientScan(char *ssid) //Scan Wi-Fi
int WIFI_Suspend(int sock) //sleep
int WIFI_Resume(void) //Wake up
```

Introduction to Keep Alive Process

```
/* Configuration */
/* wifi.h */
#define TCPKA_INTERVAL 180 //Set the packet sending interval, which can be modified

/* wifi.c */
const char tcpka_payload[] = "helloworld"; //Set the content of the keep-alive package
const char wowl_pattern[] = "123"; //Set the content of the wake-up packet

/* Part of main.c function */
        main()
        if (WIFI_GetStatus()) //If the Wi-Fi is awakened from sleep, the Wi-Fi is already connected
        {
                pr_info("Already joined AP.\n");
                wifi_get_wakeup(); //Get the reason for Wi-Fi wakeup

                ret = WIFI_Resume(); //Restore Wi-Fi status, delete the last keep-alive configuration
                if (ret)
                {
                        pr_info("resume_enter, err = %d\n", ret);
                        goto exit;
                }
                rk_obtain_ip_from_vendor(ifname); //Get IP address from vendor
        }
        else
        {
                ret = WIFI_Connect(ssid, password, 0); //Connect to Wi-Fi

                if (ret <0)
                {
                        goto exit;
                }

                ret = rk_obtain_ip_from_udhcpc(ifname); //Use udhcpc to get the IP address
                if (ret)
                {
                        pr_info("obtain_ip, err = %d\n", ret);
                        goto exit;
```

```
                }
        }
        sock = connect_server(ip, port); //Connect the keep-alive connection
of the remote server
        if (sock <0)
        {
                goto exit;
        }

        ret = send(sock, tcpka_payload, strlen(tcpka_payload), 0); //Send
initial customize data
        if (ret <0)
        {
                pr_info("send err = %d\n", ret);
                goto exit;
        }

        ret = WIFI_Suspend(sock); //Wi-Fi sleep

        //Power off operation
        system("tb_poweroff &");
```

7.5 AMPAK Wi-Fi Solution Introduction

7.5.1 AMPAK Wi-Fi Key Configuration

```
#Driver code:
kernel/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd_indep_power

#buildroot Configuration
BR2_PACKAGE_RKWIFIBT_AWNB197 = y

#Kernel Configuration
--- a/arch/arm/configs/rv1126-battery.config
+++ b/arch/arm/configs/rv1126-battery.config
@@ -1,5 +1,7 @@
 # CONFIG_AP6XXX is not set
+CONFIG_AP6XXX_INDEP_POWER=m
 CONFIG_BATTERY_CW2015=y
 CONFIG_CHARGER_GPIO=y
 CONFIG_LEDS_PWM=y
```

Modifies the content of the preset heartbeat package in kernel

```
//About the parameters of the keep-alive connection, the last two parameters need to be emphasized: 2 and 0xc000, 2 represents the content length of the keep-alive package, 0xc000 represents the content of the keep-alive package, modify it according to actual needs. Corresponds to the "/proc/tcp_params" node of the application layer
//kernel/drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd_indep_power/dhd_linux.c
static int tcp_param_show(struct seq_file *s, void *data)
{
        seq_printf(s, "dhd_priv wl tcpka_conn_add 1 %s %s %s 1 %d %d 1 1 1 1 1 2 0xc000\n", deabuf, sabuf, dabuf, source, dest);

        return 0;
}

//The port number of the keep-alive connection is modified according to the actual situation; the driver captures the socket parameters of the keep-alive connection through the port number
#define KP_PORT 5150
```

Set up wake-up package

```
external/rkwifibt/firmware/broadcom/AP6203BM/wifi/config.txt
pkt_filter_add=142 0 0 77 0xffffffffffffffffffffffffffffffff 0x2F6465766963652F77616B6575702F
# 77: it is the offset value of the wake-up content in the network packet, which can be captured and analyzed by wireshark
# 0xff....: is the mask of the network packet
# 0x2f....: is the content of the customized network package
#Note: If you don’t know how to set it, you can grab the wake-up packet sent from your server and send it to RK technician for help;
```

7.5.2 Keep-alive Application Process Example Introduction

```
//Refer to external/rkwifibt/src/tcp_client_keepalive.c
//Create a keep-alive socket:
sockfd = socket(AF_INET, SOCK_STREAM, 0)

servaddr.sin_family = AF_INET;
//Set the port number of the server keep-alive connection
servaddr.sin_port = htons(5150);
//argv[1] is the IP address of the server
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
    printf("inet_pton error for %s\n",argv[1]);
    return 0;
}
//Create a connection
if (connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) <0) {
    printf("connect error: %s (errno: %d)\n",strerror(errno),errno);
    return 0;
}

//Get the initial parameters of the keep-alive connection, corresponding to the above kernel parameters
```

```
open("/proc/tcp_params", O_RDONLY);
system(buf);
usleep(300 * 100);
system("dhd_priv wl tcpka_conn_sess_info 1");
usleep(300 * 100);
system("dhd_priv wl tcpka_conn_dump 1");

//Send the initial customized data, the parameters of tcp change
send(sockfd, "hello", 5, 0);

//Get the latest keep-alive connection parameters and set them to Wi-Fi
system("dhd_priv wl tcpka_conn_sess_info 1");
usleep(100 * 100);
system("dhd_priv wl tcpka_conn_dump 0");
usleep(100 * 100);
system("dhd_priv wl tcpka_conn_enable 1 1 30 3 8"); // 30 is the keep-alive packet interval time, in seconds
usleep(100 * 100);

//Wi-Fi sleep
system("dhd_priv setsuspendmode 1");

//power off operation
system("tb_poweroff &"); // tb_poweroff is a shutdown script, mainly used to umount and power-off operations

```

Networking script

```
external/rkwifibt/tb_start_wifi.sh
tb_start_wifi.sh ssid password #Network, execute every time you boot
```

7.6 Wi-Fi Boot Script Process Introduction

To quickly start the SDK, Wi-Fi is started through the tb_start_wifi.sh script. Next, the script will be described in details:

```
... ...
# Wake up process: The last keep-alive configuration needs to be deleted when waking up
function tcpka_del() {
        echo "tcpka_del ..."
        while true
        do
                IPID=`dhd_priv wl tcpka_conn_sess_info 1 | grep ipid`
                if ["$IPID" != "" ]; then
                        sleep 0.05
                        dhd_priv wl tcpka_conn_enable 1 0 0 0 0
                        sleep 0.05
                        dhd_priv wl tcpka_conn_del 1
                        sleep 0.05
                else
                        break
                fi

```

```
                let tcpka_cnt++
                if ["$tcpka_cnt" -gt 3 ]; then
                        echo "tcpka_conn_del failed"
                        exit
                fi

        done
}
... ...

check_wlan0                # Check whether ko is loaded
wlan_up                    # Check whether wlan0 is up
check_wakeup_cause         # Get Wi-Fi wakeup reason
tcpka_del                  # Delete the last keep-alive configuration
connect_wifi               # Connect to Wi-Fi, if it is already connected, only get the IP address
udhcpcd                    # Obtain the IP address, if the first connection is obtained with udhcpc, if it is already connected, read the configuration from the vendor partition:
```

7.7 Wi-Fi Regular Debug Methods Introduction

Compilation:

```
make menuconfig                 #Select the corresponding Wi-Fi configuration
make savedefconfig              #Save the configuration
make rkwifibt-dirclean          #clear
make rkwifibt                   #compile
```

Confirm the corresponding driver ko:

```
#The command can be seen after booting
ls /vendor/lib/modules/cywdhd.ko(bcmdhd_indep_power.ko)
#If not, please check the compilation process
```

Server program: you need to choose the corresponding solution according to the actual situation;

Wake-up: first, make sure that your hardware is in accordance with our hardware reference design. Second, you must understand the keep-alive process on the server side in details; if you have confirmed that the settings are correct, you can send us your merged code for confirmation, including the keep-alive content obtained by wireshark from server;

8. Cloud Platform Connection

At present, Rockchip’s low-power product solutions are connected to two cloud platforms: Alibaba Cloud and Tuya. The content of this chapter aims to provide customers with a guide to help customers quickly get started with the cloud platform.

8.1 Alibaba Cloud

Please refer to Link Visual Device Development-Linux SDK.

8.2 Tuya Cloud

Please refer to Tuya IPC Embedded SDK Development Guide.

8.3 Vendor Partition

Vendor partition refers to the area divided from Flash for storing vendor data. Developers can write related vendor data to the partition through the special PC tool, and the partition data will not be lost after restart or power failure. The data of the partition can be read through the related interface for display or other purposes. If the flash is erased in the whole chip, the vendor data written in the partition will be erased.

At present, the vendor partition is mainly used for two purposes, storing device certificates and Wi-Fi information.

Note: at present, each device needs to download a certificate before it connect to the cloud successfully . The tool is located in tools\windows\RKDevInfoWriteTool in the root directory of the SDK. It is necessary to ensure that the certificate of each device is unique and has been added in the cloud.

The vendor partition ID is 255, which is used to save the four-tuple certificate required for Alibaba Cloud authentication. The sample is as follows:

```
{"product_key":"a139oQFoEu6","product_secret":"LKDLOI0nJmp8m7aH","device_name":"rk10","device_secret":"77d2838182fbb2f32acc4e9298612989"}
```

The Vendor partition ID is 254, which is used to save the triple certificate required for Tuya Cloud authentication. The sample is as follows:

```
{"pid":"4wrrx6gmxh1czhcv","uuid":"tuya16c71f6e48e7a4a7","authkey":"zkjwo2tNj199MCCgdGMLEKhsu1jeHAJ8"}
```

The Vendor partition ID is 30, which is used to save Wi-Fi information and speed up the network. The sample is as follows:

```
1,fanxing,12345678,192.168.1.122,255.255.255.0,192.168.1.1,192.168.1.1
```

In addition to flashing with the RKDevInfoWriteTool tool in large quantities, you can also write certificate information on the device through the serial port, but please pay attention to the transfer and ID conversion. Take Alibaba Cloud as an example:

```
vendor_storage -w VENDOR_CUSTON_ID_FF -t string -i {\"product_key\":\"a139oQFoEu6\",\"product_secret\":\"LKDLOI0nJmp8m7aH\",\"device_name\":\"rk10\",\"device_secret \":\"77d2838182fbb2f32acc4e9298612989\"}
```

9. Optimization Method of Mistaken Wake Up

Using PIR is easily cause false triggering problems. Too many false triggers, without filtering, will have a great impact on power consumption on the one hand, and will cause frequent alarms to users on the other hand, making users feel disgusted.

There are several ways to filter false triggers in the current solution:

- Limit the number of PIR triggers, and randomly discard a certain percentage of trigger events;
- Using video motion detection algorithm, it is judged as a valid trigger only when the scene changes are detected;
- Use deep learning algorithms to detect objects in the cloud to determine whether it is a valid trigger;

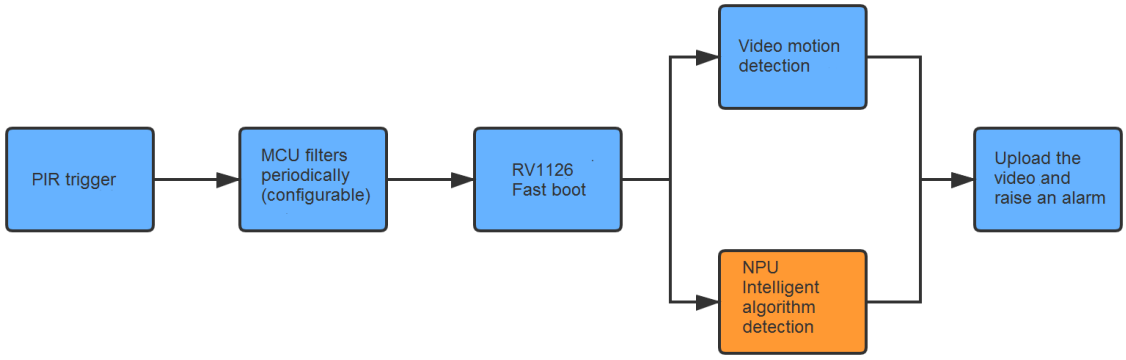
The first method can reduce the number of false triggers to a certain extent, but there is also the possibility of false alarms; the second method will often cause false alarms, some simple brightness changes on the screen will cause false alarms by the motion detection algorithm; The three methods need to send the video data after each trigger to the cloud, resulting in increased bandwidth burden.

9.1 PIR Performance Optimization

TBD

9.2 Wake Up by AI Filter

An NPU is built in RV1126/RV1109, we can add some simple detection algorithms to the device to filter the trigger events. When an object of interest is detected in the screen, an alarm will be sent to users. The key flow chart is as follows:



10. Software Function Interface

10.1 ISP

Please refer to the following document for details:
docs/Socs/RV1126_RV1109/Camera/Rockchip_Development_Guide_ISP2x_EN_v1.5.0.pdf.

The sample code for initialization and de-initialization is as follows:

```
1  char *iq_file_dir = "/etc/iqfiles/";
```

```
rk_aiq_working_mode_t enWDRMode = RK_AIQ_WORKING_MODE_NORMAL;
rk_aiq_sys_ctx_t *aiq_ctx;
rk_aiq_static_info_t aiq_static_info;

//Enumerate the static information obtained by AIQ.
rk_aiq_uapi_sysctl_enumStaticMetas(0, &aiq_static_info);
printf("sensor_name is %s, iqfiles is %s\n",
aiq_static_info.sensor_info.sensor_name, iq_file_dir);

// Initialize the AIQ context.
aiq_ctx =
rk_aiq_uapi_sysctl_init(aiq_static_info.sensor_info.sensor_name,iq_file_dir,
NULL, NULL);
// Prepare the AIQ operating environment.
if (rk_aiq_uapi_sysctl_prepare(aiq_ctx, 0, 0, enWDRMode)) {
    printf("rkaiq engine prepare failed !\n");
    return -1;
}
printf("rk_aiq_uapi_sysctl_init/prepare succeed\n");
// Start the AIQ control system. After the AIQ is started, it will obtain 3A
statistics from the ISP driver continuously, run the 3A algorithm, and apply
the calculated new parameters.
if (rk_aiq_uapi_sysctl_start(aiq_ctx)) {
    printf("rk_aiq_uapi_sysctl_start failed\n");
    return -1;
}
printf("rk_aiq_uapi_sysctl_start succeed\n");

while (true) {
    // main program
}

//Deinitialization
if (!aiq_ctx)
    return -1;
// Stop the AIQ control system.
rk_aiq_uapi_sysctl_stop(aiq_ctx, false);
// Deinitialize the AIQ context environment.
rk_aiq_uapi_sysctl_deinit(aiq_ctx);
aiq_ctx = NULL;
```

10.2 Video

Please refer to the following document for details:
docs/Socs/RV1126_RV1109/Multimedia/Rockchip_Instructions_Linux_Rkmedia_EN.pdf.

The general data flow is ISP→VI→VENC→Network stream or local save.

10.2.1 VI

The sample code for initialization and de-initialization is as follows:

```
VI_CHN_ATTR_S vi_chn_attr;
```

```
vi_chn_attr.pcVideoNode = "rkispp_scale1";
vi_chn_attr.u32BufCnt = 4;
vi_chn_attr.u32Width = 720;
vi_chn_attr.u32Height = 576;
vi_chn_attr.enPixFmt = IMAGE_TYPE_NV12;
vi_chn_attr.enWorkMode = VI_WORK_MODE_NORMAL;
VI_PIPE ViPipe = 0;
VI_CHN ViChn = 0;

// vi init
RK_MPI_VI_SetChnAttr(ViPipe, ViChn, vi_chn_attr);
RK_MPI_VI_EnableChn(ViPipe, ViChn);

// vi deinit
RK_MPI_VI_DisableChn(ViPipe, ViChn);
```

10.2.2 VENC

The sample code for initialization and de-initialization is as follows:

```
VENC_CHN_ATTR_S venc_chn_attr;
venc_chn_attr.stVencAttr.enType = RK_CODEC_TYPE_H264;
venc_chn_attr.stVencAttr.imageType = IMAGE_TYPE_NV12;
venc_chn_attr.stVencAttr.u32PicWidth = 720;
venc_chn_attr.stVencAttr.u32PicHeight = 576;
venc_chn_attr.stVencAttr.u32VirWidth = 720;
venc_chn_attr.stVencAttr.u32VirHeight = 576;
venc_chn_attr.stVencAttr.u32Profile = 77;
venc_chn_attr.stRcAttr.enRcMode = VENC_RC_MODE_H264CBR;
venc_chn_attr.stRcAttr.stH264Cbr.u32Gop = 30;
venc_chn_attr.stRcAttr.stH264Cbr.u32BitRate = 720 * 576 * 30 / 14;
venc_chn_attr.stRcAttr.stH264Cbr.fr32DstFrameRateDen = 1;
venc_chn_attr.stRcAttr.stH264Cbr.fr32DstFrameRateNum = 30;
venc_chn_attr.stRcAttr.stH264Cbr.u32SrcFrameRateDen = 1;
venc_chn_attr.stRcAttr.stH264Cbr.u32SrcFrameRateNum = 30;
// venc init
RK_MPI_VENC_CreateChn(0, &venc_chn_attr);

// venc deinit
RK_MPI_VENC_DestroyChn(VencChn);
```

Currently, there are two ways for VENC to obtain data from VI:

1. Using the RK_MPI_SYS_Bind(&ViChn, &VencChn) function, the data flow will go from VI to VENC automatically. **But note that you need to use RK_MPI_SYS_UnBind to unbind first** when de-initializing, and then destroy Venc and then Vi.
2. The working mode of VI is set to VI_WORK_MODE_GOD_MODE. You can use RK_MPI_SYS_RegisterOutCb in the output callback function of VI to send the buffer to the VENC for processing. Or call RK_MPI_SYS_GetMediaBuffer actively to get the buffer from the VI channel and then call RK_MPI_SYS_SendMediaBuffer to VENC for processing. **In this way, the path of VI→VENC can be changed to VI→frame→VENC, or one VI can be sent to multiple VENC for processing.**

The output data of VENC can also use RK_MPI_SYS_RegisterOutCb or RK_MPI_SYS_GetMediaBuffer, which can be processed by users.

10.3 Audio

Please refer to the following document for details:
docs/Socs/RV1126_RV1109/Multimedia/Rockchip_Instructions_Linux_Rkmedia_EN.pdf.

Generally, there are two types of data streams: capture and playback:

Capture: AI→AENC→Network streaming or local save.

Playback: the network receives the stream or reads the local file→ADEC→AO.

10.3.1 AI and VQE

The sample code for initialization and de-initialization is as follows:

```
mpp_chn_ai.enModId = RK_ID_AI;
mpp_chn_ai.s32ChnId = 0;
AI_CHN_ATTR_S ai_attr;
ai_attr.pcAudioNode = "default";
ai_attr.enSampleFormat = RK_SAMPLE_FMT_S16;
ai_attr.u32NbSamples = 1024;
ai_attr.u32SampleRate = 16000;
ai_attr.u32Channels = 1;
ai_attr.enAiLayout = AI_LAYOUT_MIC_REF;

// AI init
RK_MPI_AI_SetChnAttr(mpp_chn_ai.s32ChnId, &ai_attr);
RK_MPI_AI_EnableChn(mpp_chn_ai.s32ChnId);
// VQE Enable
AI_TALKVQE_CONFIG_S stAiVqeTalkAttr;
memset(&stAiVqeTalkAttr, 0, sizeof(AI_TALKVQE_CONFIG_S));
stAiVqeTalkAttr.s32WorkSampleRate = 16000;
stAiVqeTalkAttr.s32FrameSample = 320;
stAiVqeTalkAttr.aParamFilePath =
"/usr/share/rkap_aec/para/16k/RKAP_AecPara.bin";
stAiVqeTalkAttr.u32OpenMask = AI_TALKVQE_MASK_AEC | AI_TALKVQE_MASK_ANR |
AI_TALKVQE_MASK_AGC;
RK_MPI_AI_SetTalkVqeAttr(mpp_chn_ai.s32ChnId, &stAiVqeTalkAttr);
RK_MPI_AI_EnableVqe(mpp_chn_ai.s32ChnId);

// VQE Disable
RK_MPI_AI_DisableVqe(mpp_chn_ai.s32ChnId);
// AI Disable
RK_MPI_AI_DisableChn(mpp_chn_ai.s32ChnId);
```

10.3.2 AENC

The sample code for initialization and de-initialization is as follows:

```
mpp_chn_aenc.enModId = RK_ID_AENC;
mpp_chn_aenc.s32ChnId = 0;
AENC_CHN_ATTR_S aenc_attr;
aenc_attr.enCodecType = RK_CODEC_TYPE_AAC;
aenc_attr.u32Bitrate = 64000;
aenc_attr.u32Quality = 1;
aenc_attr.stAencAAC.u32Channels = 1;
aenc_attr.stAencAAC.u32SampleRate = 16000;
// AENC init
RK_MPI_AENC_CreateChn(mpp_chn_aenc.s32ChnId, &aenc_attr);

// AENC deinit
RK_MPI_AENC_DestroyChn(mpp_chn_aenc.s32ChnId);
```

Use RK_MPI_SYS_Bind(&mpp_chn_ai, &mpp_chn_aenc) for binding.

When de-initializing, RK_MPI_SYS_UnBind(&mpp_chn_ai, &mpp_chn_aenc) is required first.

10.3.3 ADEC

The sample code for initialization and de-initialization is as follows:

```
mpp_chn_adec.enModId = RK_ID_ADEC;
mpp_chn_adec.s32ChnId = 0;
ADEC_CHN_ATTR_S stAdecAttr;
stAdecAttr.enCodecType = RK_CODEC_TYPE_AAC;
if (stAdecAttr.enCodecType == G711A) {
    stAdecAttr.stAdecG711A.u32Channels = 1;
    stAdecAttr.stAdecG711A.u32SampleRate = 16000;
}
// ADEC init
RK_MPI_ADEC_CreateChn(mpp_chn_adec.s32ChnId, &stAdecAttr);

// ADEC deinit
RK_MPI_ADEC_DestroyChn(mpp_chn_adec.s32ChnId);
```

10.3.4 AO

The sample code for initialization and de-initialization is as follows:

```
mpp_chn_ao.enModId = RK_ID_AO;
mpp_chn_ao.s32ChnId = 0;
AO_CHN_ATTR_S stAoAttr;
stAoAttr.u32Channels = 1;
stAoAttr.u32SampleRate = 16000;
stAoAttr.u32NbSamples = 1024;
stAoAttr.pcAudioNode = "default";
stAoAttr.enSampleFormat = RK_SAMPLE_FMT_S16;
stAoAttr.u32NbSamples = 1024;
// AO init
RK_MPI_AO_SetChnAttr(mpp_chn_ao.s32ChnId, &stAoAttr);
RK_MPI_AO_EnableChn(mpp_chn_ao.s32ChnId);

```

```
// AO deinit
RK_MPI_AO_DisableChn(mpp_chn_ao.s32ChnId);
```

Use RK_MPI_SYS_Bind(&mpp_chn_adec, &mpp_chn_ao) for binding.

When de-initializing, RK_MPI_SYS_UnBind(&mpp_chn_adec, &mpp_chn_ao) is required first.