Audio Video Development Kits Document

Bekencorp

Jul 22, 2025

CONTENTS

This is the documentation for Beken Armino AVDK Development Framework.

This document is based on Armino IDK and helps users open pronunciation video applications;

Platform related configurations and usage, please refer to Armino IDK

CHAPTER

ONE

QUICK START GUIDE

1.1 Armino AVDK SDK Code download

We can download Armino AVDK SDK from gitlab:

```
mkdir -p ~/armino
cd ~/armino
git clone -b release/v2.0.2 --recurse-submodules https://gitlab.bekencorp.com/armino/
↪bk_avdk.git

or:

git clone -b release/v2.0.2 --recurse-submodules git@gitlab.bekencorp.com:armino/bk_
↪avdk.git
```

We can download Armino AVDK SDK from github:

```
mkdir -p ~/armino
cd ~/armino
git clone -b release/v2.0.2 --recurse-submodules https://github.bekencorp.com/armino/
↪bk_avdk.git
```

Then switch to the stable branch Tag node, such as v2.0.2.2:

```
cd ~/armino/bk_avdk
git checkout -B your_branch_name v2.0.2.2
git submodule update --init --recursive
ls
```

```
longbao.hu@shdsci0[14:20:13]~/armino/bk_avdk$ ls
bk_idk        build_avdk.sh  build.sh    docs      projects         tools
build_all.sh  build_doc.sh   components  Makefile  sdk_version.txt
```

Fig. 1: Figure 1. AVDK Directiry structure

1.2 Build Compilation Environment:

Note:

Armino, currently supports compiling in Linux environment. This chapter willtake Ubuntu 20.04 LTS as an example to introduce the construction of the entire compiling environment.

1.2.1 Install Tool Chain

Click Download to download the BK7258 toolchain.

After downloading the tool kit, decompress it to '/opt/':

```
$ sudo tar -xvjf gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2 -C /opt/
```

Note: Tool chain the default path is configured in the middleware/soc/bk7258/bk7258.defconfig, you can modify `CONFIG_TOOLCHAIN_PATH` to set to your owner toolchain path:

CONFIG_TOOLCHAIN_PATH="/opt/gcc-arm-none-eabi-10.3-2021.10/bin"

1.2.2 Install Depended libraries

Enter the following command in the terminal to install python3,CMake,Ninja:

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install build-essential cmake python3 python3-pip doxygen ninja-build␣
↪libc6:i386 libstdc++6:i386 libncurses5-dev lib32z1 -y
sudo pip3 install pycrypto click
```

1.2.3 Install python dependencies

Enter the following command to install python dependencies:

```
sudo pip3 install sphinx_rtd_theme future breathe blockdiag sphinxcontrib-seqdiag␣
↪sphinxcontrib-actdiag sphinxcontrib-nwdiag sphinxcontrib.blockdiag
```

If you default Python is Python2, please set it to Python3:

```
sudo ln -s /usr/bin/python3 /usr/bin/python
```

1.3 Build The Project

Run following commands to build BK7258 default doorbell project:

```
cd ~/armino/bk_avdk
make bk7258
```

You can also build projects with PROJECT parameter, e.g. run "make bk7258 PROJECT=media/doorbell" can build projects/media/doorbell etc.

1.4 Configuration project

We can also use the project configuration file for differentiated configuration:

```
Project Profile Override Chip Profile Override Default Configuration
Example: config >> bk7258.defconfig >> KConfig
+ Example of project configuration file:
    projects/media/doorbell/config/bk7258/config
+ Sample chip configuration file:
    middleware/soc/bk7258/bk7258.defconfig
+ Sample KConfig configuration file:
    middleware/arch/cm33/Kconfig
    components/bk_cli/Kconfig
```

1.5 Create New project

The default project is projects/media/doorbell. For new projects, please refer to the project in projects/media/

1.6 Burn Code

On the Windows platform, Armino currently supports UART burning.

For detailed burning process, please refer to IDK

CHAPTER

TWO

HW REFERENCE

CHAPTER

THREE

API REFERENCES

3.1 API Overview

The following table shows the status of the SDK public API

3.2 Multi-Media

3.2.1 Audio APIs

Important: The Audio API v1.0 is the lastest and stable Audio APIs. All new applications should use Audio API v1.0.

Audio Interface

The BK Audio Driver supports following functions:

- adc function
- dac function
- eq function

The adc and dac function can be operated independently. However the eq function need to use with dac function together.

Audio API Categories

Most of Audio APIs can be categoried as:

- Common APIs

  The common APIs are prefixed with bk_aud, the APIs may be common for adc and dac interfaces, e.g. bk_aud_driver_init() etc.

- Adc function APIs.

  The APIs provide support for adc function. e.g. bk_aud_adc_init() etc.

- Dac function APIs.

  The APIs provide support for dac function. e.g. bk_aud_dac_init() etc.

- Eq function APIs.

  The APIs provide support for eq function. e.g. bk_aud_eq_init() etc.

Common APIs:

- `bk_aud_driver_init()` - init the audio driver
- `bk_aud_driver_deinit()` - deinit the audio driver
- `bk_aud_register_aud_isr()` - register audio isr

Adc APIs:

- `bk_aud_adc_init()` - init the adc module of audio
- `bk_aud_adc_deinit()` - deinit the adc module of audio
- `bk_aud_set_adc_samp_rate()` - set the sample rate in adc work mode
- `bk_aud_get_adc_fifo_addr()` - get the adc fifo address in adc work mode
- `bk_aud_get_dtmf_fifo_addr()` - get the dtmf fifo address in adc work mode
- `bk_aud_get_adc_status()` - get the adc status information in adc work mode
- `bk_aud_get_dtmf_status()` - get the dtmf status information in dtmf work mode
- `bk_aud_enable_adc_int()` - enable adc interrupt
- `bk_aud_disable_adc_int()` - disable adc interrupt
- `bk_aud_start_adc()` - start adc function
- `bk_aud_stop_adc()` - stop adc function
- `bk_aud_get_adc_fifo_data()` - get adc data
- `bk_aud_get_dtmf_fifo_data()` - get dtmf data

Dac APIs:

- `bk_aud_dac_init()` - init the dac module of audio
- `bk_aud_dac_deinit()` - deinit the dac module of audio
- `bk_aud_set_dac_samp_rate()` - set the sample rate of audio dac function
- `bk_aud_get_dac_fifo_addr()` - get the dac fifo address
- `bk_aud_enable_dac_int()` - enable dac interrupt function
- `bk_aud_disable_dac_int()` - disable dac interrupt function
- `bk_aud_get_dac_status()` - get the dac fifo status information
- `bk_aud_start_dac()` - start dac function
- `bk_aud_stop_dac()` - stop dac function

Eq APIs:

- `bk_aud_eq_init()` - init the eq module of audio
- `bk_aud_eq_deinit()` - deinit the eq module of audio

API Reference

#.. include:: ../../_build/inc/aud.inc

API Typedefs

#.. include:: ../../_build/inc/aud_types.inc

3.2.2 Display APIs

LCD Display Interface

The BK LCD Display Driver supports following interfaces:

- **i8080 LCD interface**
  1) The screen Driving IC used in the project is ST7796S
  2) The data transmission of the 8080 interface is 8-bit
  3) The interface supports RGB565 data
- **RGB interface**
  1) The supported screens in the project include Driving IC ST7282, HX8282 and GC9503V
  2) the data supported by the interface including RGB565, YUYV, UYVY, YYUV, UVYY, VUYY

LCD API Categories

LCD 8080 Config

LCD RGB Config

LCD parcical display

Local display parameter configuration

API Reference

Header File

- lcd.h

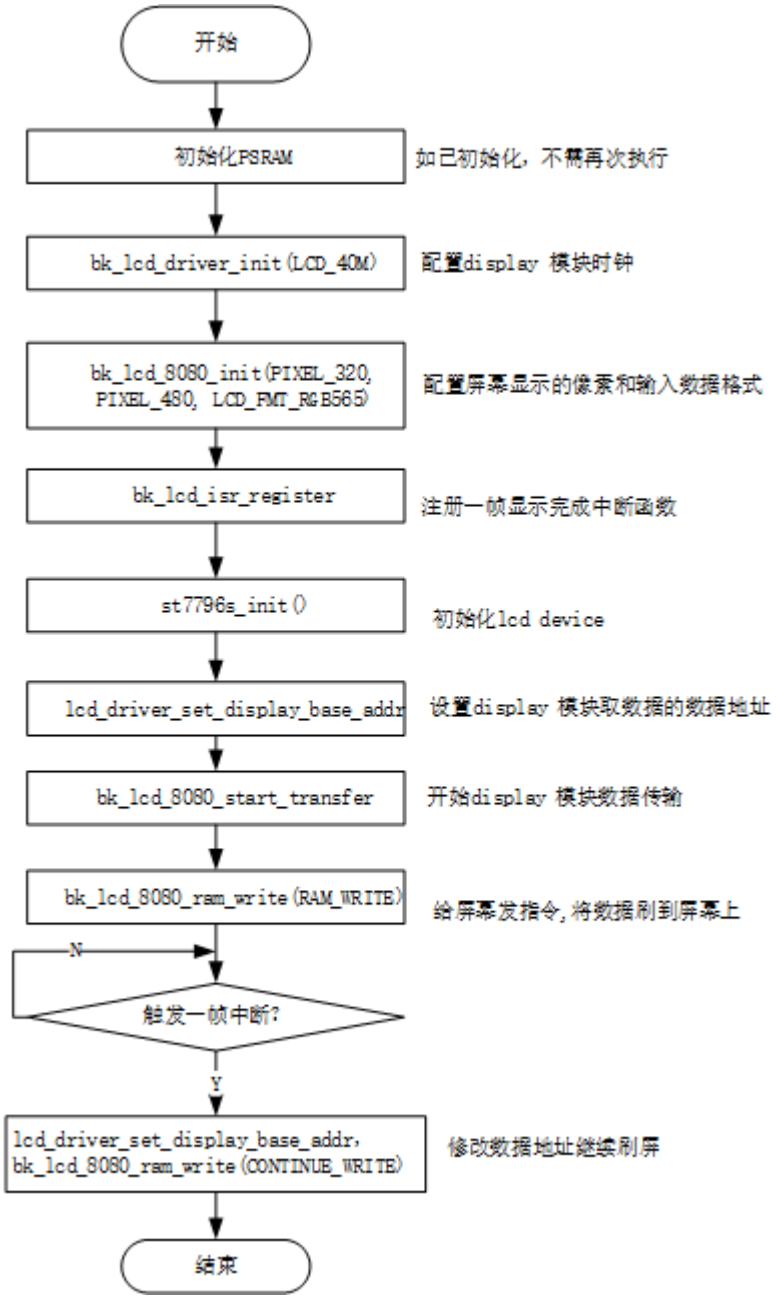


Fig. 1: Figure 1. 8080 lcd config flow

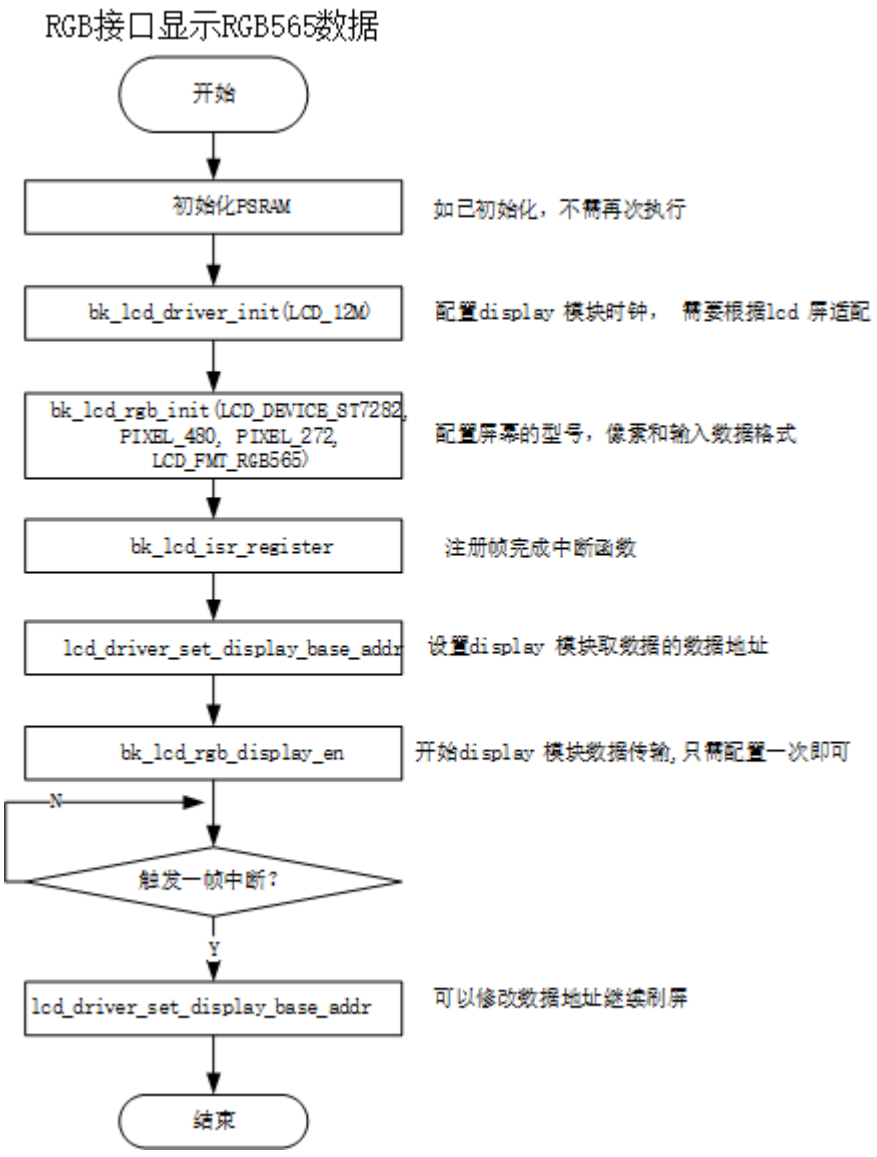


Fig. 2: Figure 2. rgb lcd config flow

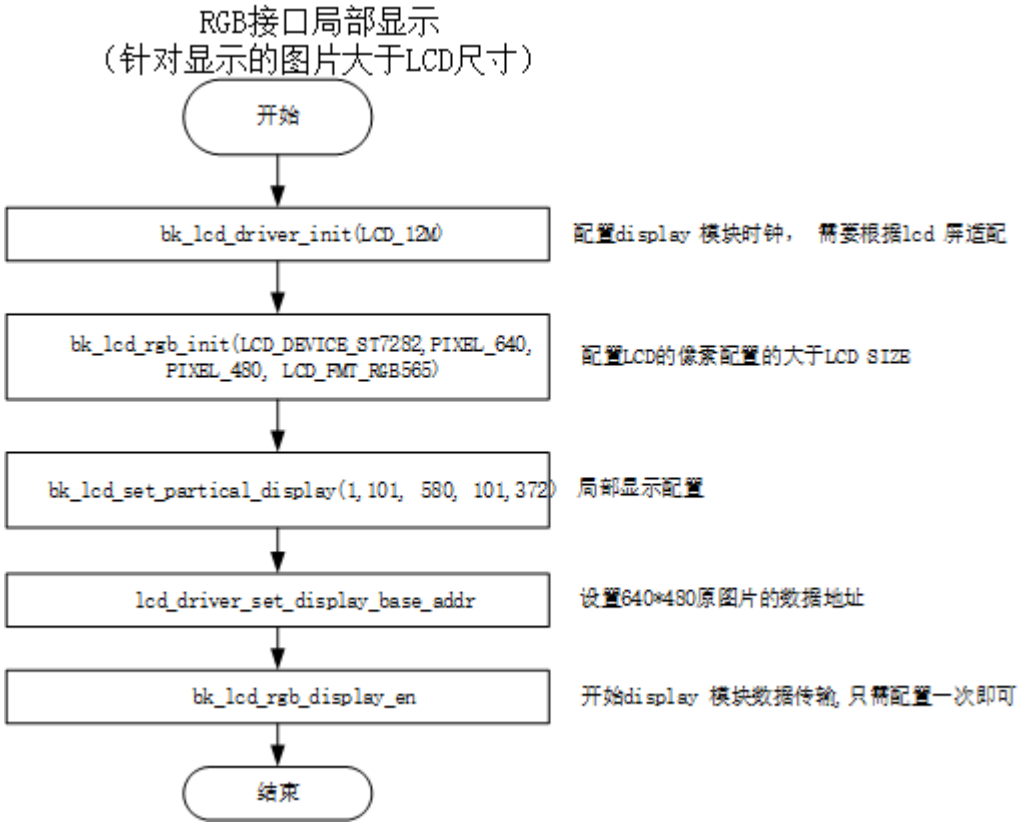


Fig. 3: Figure 3. rgb lcd config flow

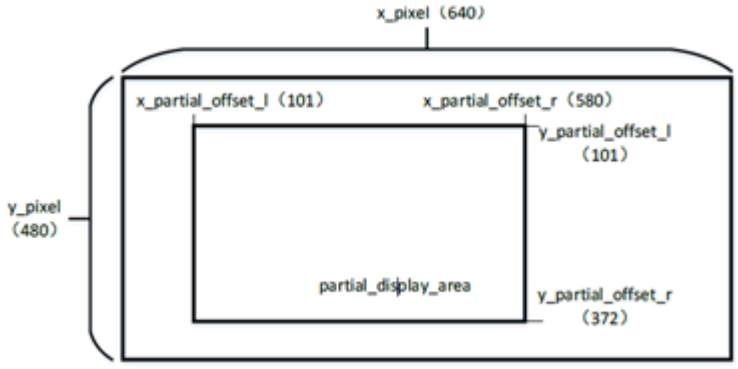


Fig. 4: Figure 4. rgb lcd config flow

Functions

void **bk_lcd_set_devices_list**(const *lcd_device_t* **list, uint16_t size)
LCD API.

This API set LCD device list

Returns

- void

bk_err_t **bk_lcd_driver_init**(*lcd_clk_t* clk)
This API select LCD module clk source.

- open lcd sys interrupt/clk enable

BK_OK: succeed

- others: other errors.

Returns

bk_err_t **bk_lcd_driver_deinit**(void)
close lcd sys interrupt/clk enable

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_8080_init**(const *lcd_device_t* *device)
This API init the 8080 lcd interface.

- Set lcd display mode is 8080 interface
- init 8080 lcd gpio
- enable 8080 display -enable 8080 end of frame interrupt
- if you want enable start of frame interrupt, please use API bk_lcd_8080_int_enable

Parameters **-** – *lcd_device_t*

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_8080_deinit**(void)
8080 lcd interface reg deinit

- This API reset all lcd reg include power
- close 8080 lcd enable and display
- reset x pixel and y pixel zero
- unregister lcd isr

Returns

- BK_OK: succeed

- others: other errors.

bk_err_t **bk_lcd_8080_start_transfer**(bool start)
This API config 8080 lcd interrupt.

This API start mcu 8080 lcd transfer data to display

Attention 8080 end of frame int is open in API bk_lcd_8080_init

Parameters

- – is_sof_en enable start of frame interrupt
  - is_eof_en enable end of frame interrupt
- **start_transfer** –
  - 1:data start transfer to lcd display on;
  - 0:stop transfer

Returns

- BK_OK: succeed
- others: other errors.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_rgb_deinit**(void)
rgb lcd interface reg deinit

- This API reset all lcd reg include power
- close rgb lcd enable and display
- reset x pixel and y pixel zero
- unregister lcd isr

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_rgb_display_en**(bool en)
enable rgb lcd display

Parameters **bool** – en

- 1: enable rgb display
- 0: disable rgb display

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_isr_register**(*lcd_isr_t* lcd_isr)
This API config rgb lcd interrupt.

This API register lcd isr, user should check int status in isr function, and clear status

Attention rgb end of frame int is open in API bk_lcd_rgb_init

Parameters

- – – is_sof_en enable start of frame interrupt
  – is_eof_en enable end of frame interrupt
- – – isr: isr function

Returns

- BK_OK: succeed
- others: other errors.

Returns

- BK_OK: succeed
- others: other errors.

uint32_t **bk_lcd_int_status_get**(void)
This API used to get lcd int status.

8080 Usage example: if (bk_lcd_int_status_get()&I8080_OUTPUT_EOF == 1), present 8080 eof int triggerd if (bk_lcd_int_status_get()&RGB_OUTPUT_EOF == 1), present rgb eof int triggerd

Returns

- status value.

bk_err_t **bk_lcd_int_status_clear**(*lcd_int_type_t* int_type)
This API used to clr lcd int status.

Parameters – – int_type value.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_pixel_config**(uint16_t x_pixel, uint16_t y_pixel)
This API config lcd display x size and y size.

Parameters – – x_pixel user can set by any value, can be lcd size x or picture size x

- y_pixel user can set by any value, can be lcd size y or picture size y

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_rgb_init**(const *lcd_device_t* *device)
This API init the rgb lcd interface.

- Set lcd display mode is rgb interface
- init rgb lcd gpio

- enable rgb display
- enable rgb end of frame interrupt

Parameters - – *lcd_device_t*: lcd type select from *lcd_device_t*

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_8080_send_cmd**(uint8_t param_count, uint32_t command, uint32_t *param)
This API used send 8080 lcd init cmd.

Usage example:

#define COMMAND_1 0xf uint32_t param_command1[2] = {0xc3, 0x29}; bk_lcd_8080_send_cmd(2, COMMAND_1, param_command1);

Parameters - – param_count: cmd parameter number

- command: command
- param: the cmd parameter

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_set_partical_display**(bool en, uint16_t partial_clum_l, uint16_t partial_clum_r, uint16_t partial_line_l, uint16_t partial_line_r)
This API used for display partical area.

8080 Usage example:

```
#define PARTICAL_XS   101
#define PARTICAL_XE   220
#define PARTICAL_YS   101
#define PARTICAL_YE   380
bk_lcd_set_partical_display(EDGE_PARTICAL_XS, EDGE_PARTICAL_XE, EDGE_PARTICAL_YS,
↪ EDGE_PARTICAL_YE);
bk_lcd_8080_send_cmd(2, COMMAND_1, param_command1);
```

Parameters - – partial_clum_l:

- partial_clum_r:
- partial_line_l:
- partial_line_r:

Returns

- BK_OK: succeed
- others: other errors.

void **lcd_driver_ppi_set**(uint16_t width, uint16_t height)
This API used to clr lcd int status.

Parameters - – int_type value.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_lcd_set_yuv_mode**(pixel_format_t input_data_format)
This API used to set lcd display data farmat.

Parameters **-** – input_data_format value.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **lcd_driver_init**(const *lcd_device_t* *device)
This API used for init lcd.

Usage example:

Parameters **lcd_config_t** –

Returns

- BK_OK: succeed
- others: other errors.

const *lcd_device_t* ***get_lcd_device_by_id**(*lcd_device_id_t* id)
this api used to get lcd device interface

Parameters **select** – lcd_device_id_t member

Returns lcd device infermation, include: .id = LCD_DEVICE_HX8282, .name = "hx8282", .type = LCD_TYPE_RGB565, .ppi = PPI_1024X600, .rgb = &lcd_rgb .init = NULL,

const *lcd_device_t* ****get_lcd_devices_list**(void)
get lcd device list

Returns

- devices list

uint32_t **get_lcd_devices_num**(void)
get lcd device number

Returns

- devices number

bk_err_t **lcd_driver_backlight_open**(void)
this api used to open lcd backlight

Parameters **none** –

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **lcd_driver_backlight_close**(void)
this api used to close lcd backlight

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **lcd_driver_backlight_set**(uint8_t percent)
this api used to set lcd backlight

Parameters `percent` – rang from 0~100

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **lcd_driver_display_disable**(void)
this api used to disable lcd display

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **lcd_driver_display_enable**(void)
this api used to enable lcd display

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **lcd_driver_display_continue**(void)
this api used to clear eof int and diaplay continue

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **lcd_driver_set_display_base_addr**(uint32_t disp_base_addr)
this api used to set lcd display addr

Parameters `disp_base_addr` – sram or psram

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **lcd_driver_deinit**(void)
this api used to close lcd

Returns

- BK_OK: succeed
- others: other errors.

const *lcd_device_t* ***get_lcd_device_by_ppi**(media_ppi_t ppi)
get lcd device struct by ppi

Parameters `lcd` – ppi

Returns

- return true is *lcd_device_t* member
- or not find device, return NULL.

const *lcd_device_t* ***get_lcd_device_by_name**(char *name)
get lcd device struct by device name

Parameters `lcd` – name

Returns

- return true is *lcd_device_t* member

- or not find device, return NULL.

bk_err_t **bk_lcd_input_pixel_hf_reverse**(bool hf_reverse)
input data halfword reverse

Parameters **enable** – enable/disable hf_reverse

Returns

- BK_OK: succeed
- others: other errors.

API Typedefs

Header File

- lcd_types.h

Structures

struct **yuv_data_t**

struct **lcd_rect_t**

struct **lcd_disp_framebuf_t**

struct **lcd_rgb_t**
rgb interface config param

Public Members

lcd_clk_t **clk**
config lcd clk

rgb_out_clk_edge_t **data_out_clk_edge**
rgb data output in clk edge, should refer lcd device spec

uint16_t **hsync_back_porch**
rang 0~0x3FF (0~1023), should refer lcd device spec

uint16_t **hsync_front_porch**
rang 0~0x3FF (0~1023), should refer lcd device spec

uint16_t **vsync_back_porch**
rang 0~0xFF (0~255), should refer lcd device spec

uint16_t **vsync_front_porch**
rang 0~0xFF (0~255), should refer lcd device spec

uint8_t **hsync_pulse_width**
rang 0~0x3F (0~7), should refer lcd device spec

uint8_t **vsync_pulse_width**
rang 0~0x3F (0~7), should refer lcd device spec

struct **lcd_mcu_t**
mcu interface config param

Public Members

lcd_clk_t **clk**
config lcd clk

void (*__set_display_area__)(uint16 xs, uint16 xe, uint16 ys, uint16 ye)
if lcd size is smaller then image, and set api bk_lcd_pixel_config is image x y, should set partical display

struct **lcd_qspi_t**
qspi interface config param

struct **lcd_spi_t**
spi interface config param

struct **lcd_device_t**

Public Members

lcd_device_id_t **id**
lcd device type, user can add if you want to add another lcd device

char ***name**
lcd device name

lcd_type_t **type**
lcd device hw interface

media_ppi_t **ppi**
lcd device x y size

pixel_format_t **src_fmt**
source data format: input to display module data format(rgb565/rgb888/yuv)

pixel_format_t **out_fmt**
display module output data format(rgb565/rgb666/rgb888), input to lcd device,

const *lcd_rgb_t* ***rgb**
RGB interface lcd device config

const *lcd_mcu_t* ***mcu**
MCU interface lcd device config

const *lcd_qspi_t* ***qspi**
QSPI interface lcd device config

const *lcd_spi_t* ***spi**
SPI interface lcd device config

void (***init**)(void)
lcd device initial function

bk_err_t (***lcd_off**)(void)
lcd off

Macros

USE_LCD_REGISTER_CALLBACKS

Type Definitions

typedef void (***lcd_isr_t**)(void)

Enumerations

enum **lcd_device_id_t**
Values:

enumerator **LCD_DEVICE_UNKNOW**

enumerator **LCD_DEVICE_ST7282**
480X270 RGB

enumerator **LCD_DEVICE_HX8282**
1024X600 RGB

enumerator **LCD_DEVICE_GC9503V**
480X800 RGB

enumerator **LCD_DEVICE_NT35510**
480X854 RGB

enumerator **LCD_DEVICE_H050IWV**
800X480 RGB

enumerator **LCD_DEVICE_MD0430R**
800X480 RGB

enumerator **LCD_DEVICE_MD0700R**
1024X600 RGB

enumerator **LCD_DEVICE_ST7701S_LY**
480X480 RGB

enumerator **LCD_DEVICE_ST7701S**
480X480 RGB

enumerator **LCD_DEVICE_ST7701SN**
480X480 RGB

enumerator **LCD_DEVICE_AML01**
720X1280 RGB

enumerator **LCD_DEVICE_ST7796S**
320X480 MCU

enumerator **LCD_DEVICE_NT35512**
480X800 MCU

enumerator **LCD_DEVICE_NT35510_MCU**
480X800 MCU

enumerator **LCD_DEVICE_ST7789V**
170X320 MCU

enumerator **LCD_DEVICE_ST7796PI_MCU16**
320X480 MCU 16bit

enumerator **LCD_DEVICE_SH8601A**
454X454 QSPI

enumerator **LCD_DEVICE_ST77903_WX20114**
400X400 QSPI

enumerator **LCD_DEVICE_ST77903_SAT61478M**
400X400 QSPI

enumerator **LCD_DEVICE_ST77903_H0165Y008T**
360X480 QSPI

enumerator **LCD_DEVICE_SPD2010**
412X412 QSPI

enumerator **LCD_DEVICE_GC9C01**
360X360 QSPI

enumerator **LCD_DEVICE_JD9855**
360X360 QSPI

enumerator **LCD_DEVICE_JD9855_K18XJ15**
360X360 QSPI

enumerator **LCD_DEVICE_ST77916**
360X360 QSPI

enumerator **LCD_DEVICE_JD9853A**
240X320 QSPI

enumerator **LCD_DEVICE_ST7796U**
320X480 SPI

enumerator **LCD_DEVICE_GC9D01**
160X160 SPI

enumerator **LCD_DEVICE_ST7789V2**
240X320 SPI

enum **lcd_type_t**
Values:

enumerator **LCD_TYPE_RGB**
lcd hardware interface is parallel RGB interface

enumerator **LCD_TYPE_RGB565**
lcd device output data hardware interface is RGB565 format

enumerator **LCD_TYPE_MCU8080**
lcd device output data hardware interface is MCU 8BIT format

enumerator **LCD_TYPE_QSPI**
lcd device hardware interface is QSPI interface

enumerator **LCD_TYPE_SPI**
lcd device hardware interface is SPI interface

enum **lcd_int_type_t**
Values:

enumerator **RGB_OUTPUT_EOF**
reg end of frame int status

enumerator **RGB_OUTPUT_SOF**
reg display output start of frame status

enumerator **I8080_OUTPUT_SOF**
8080 display output start of frame status

enumerator **I8080_OUTPUT_EOF**
8080 display output end of frame status

enumerator **DE_INT**

enumerator **FRAME_INTERVAL_INT**

enum **frame_delay_unit_t**
Values:

enumerator **DCLK_UNIT**

enumerator **HSYNC_UNIT**

enumerator **VSYNC_UNIT**

enumerator **NONE**

enum **lcd_clk_t**

rgb lcd clk select, infulence pfs, user should select according to lcd device spec

Values:

enumerator **LCD_80M**

enumerator **LCD_60M**

enumerator **LCD_54M**

enumerator **LCD_40M**

enumerator **LCD_32M**

enumerator **LCD_30M**

enumerator **LCD_26M**

enumerator **LCD_22M**

enumerator **LCD_20M**

enumerator **LCD_17M**

enumerator **LCD_15M**

enumerator **LCD_12M**

enumerator **LCD_10M**

enumerator **LCD_9M**

enumerator **LCD_8M**

enum **lcd_qspi_clk_t**

Values:

enumerator **LCD_QSPI_80M**

enumerator **LCD_QSPI_64M**

enumerator **LCD_QSPI_60M**

enumerator **LCD_QSPI_53M**

enumerator **LCD_QSPI_48M**

enumerator **LCD_QSPI_40M**

enumerator **LCD_QSPI_32M**

enumerator **LCD_QSPI_30M**

enum **rgb_out_clk_edge_t**

rgb data output in clk rising or falling

Values:

enumerator **POSEDGE_OUTPUT**

output in clk falling

enumerator **NEGEDGE_OUTPUT**

output in clk rising

enum **data_format_t**

Values:

enumerator **ARGB8888**

ARGB8888 DMA2D color mode

enumerator **RGB888**

RGB888 DMA2D color mode

enumerator **RGB565**
RGB565 DMA2D color mode

enumerator **YUYV**

enum **lcd_backlight_default_ctrl_t**
Values:

enumerator **LCD_BL_DEFAULT_CLOSE**

enumerator **LCD_BL_DEFAULT_OPEN**

3.2.3 DMA2D APIs

DMA2D Interface

DMA2D Features

- **fill specific color** Fill part or all of the target image with a specific color
- **mem to mem copy** Copy part or all of the source image to part or all of the target image
- **mem to mem with pixel conversion** Copy part or all of the source image to part or all of the target image through pixel format conversion
- **mem to mem with pixel conversionwith and blending** Part or all of the two source images with different pixel formats are mixed, and the result is copied to the part or the entire target :image with different color formats

DMA2D API Categories

DMA2D API	Description
bk_dma2d_driver_init()	initializes the DMA2D
bk_dma2d_init()	config the DMA2D
bk_dma2d_driver_deinit()	Deinitializes the DMA2D peripheral
bk_dma2d_layer_config()	Configure the background or foreground Layer
bk_dma2d_start_transfer()	Start the DMA2D Transfer
bk_dma2d_start_blending()	Start the multi-source DMA2D Transfer
bk_dma2d_is_transfer_busy()	check dma2d is transfer busy or not
bk_dma2d_int_enable()	clear dma2d int status
bk_dma2d_int_status_clear()	Lcd transfer pixel config

DMA2D Register ISR API	Description
bk_dma2d_isr_register()	register dma2d cpu int isr
bk_dma2d_register_int_callback_isr()	register dma2d int type isr

DMA2D Work Config

1.fill specific color

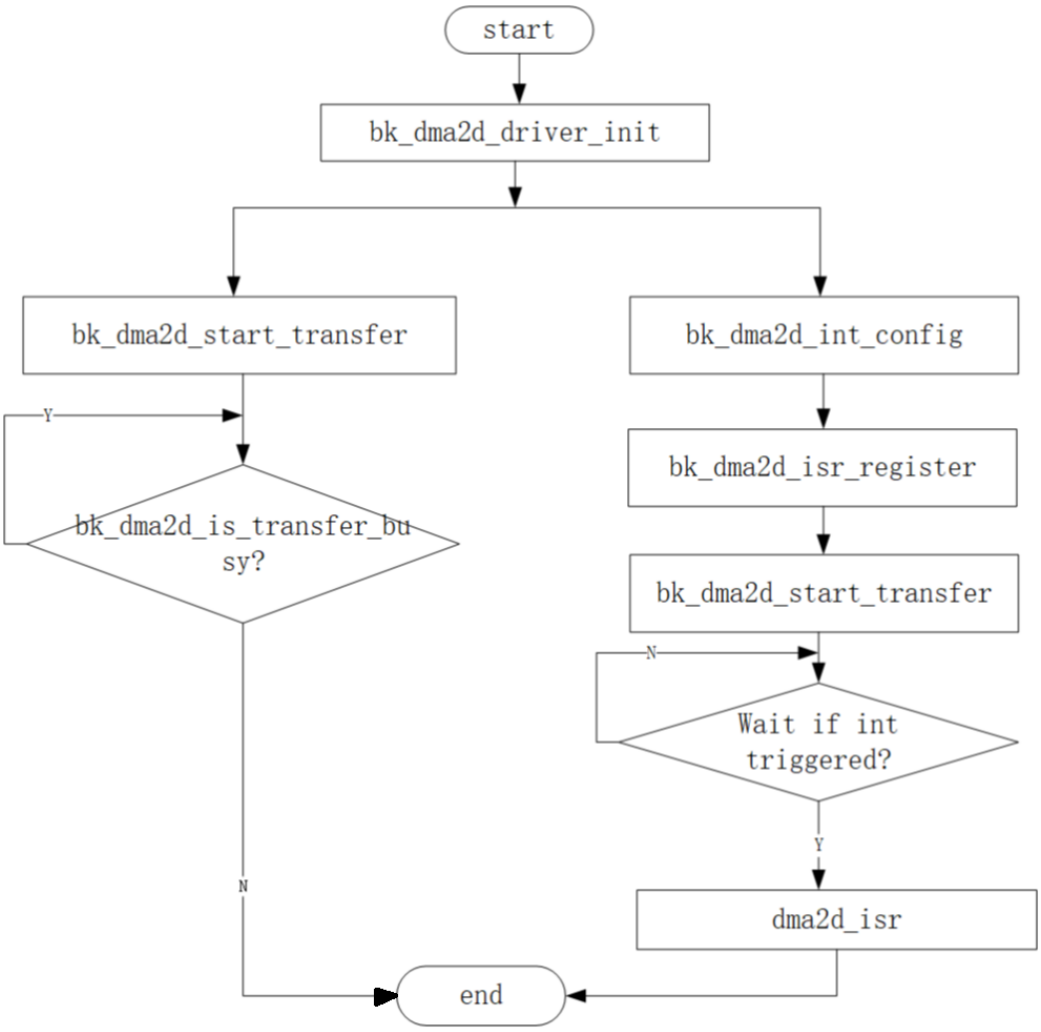


Fig. 5: Figure 1. dma2d_fill_color

2.mem to mem copy

3.mem to mem with pixel conversion

> The Figure API is the same as dma2d_mem_to_mem please reference the API Reference for the different config

4.mem to mem with pixel conversion and blending

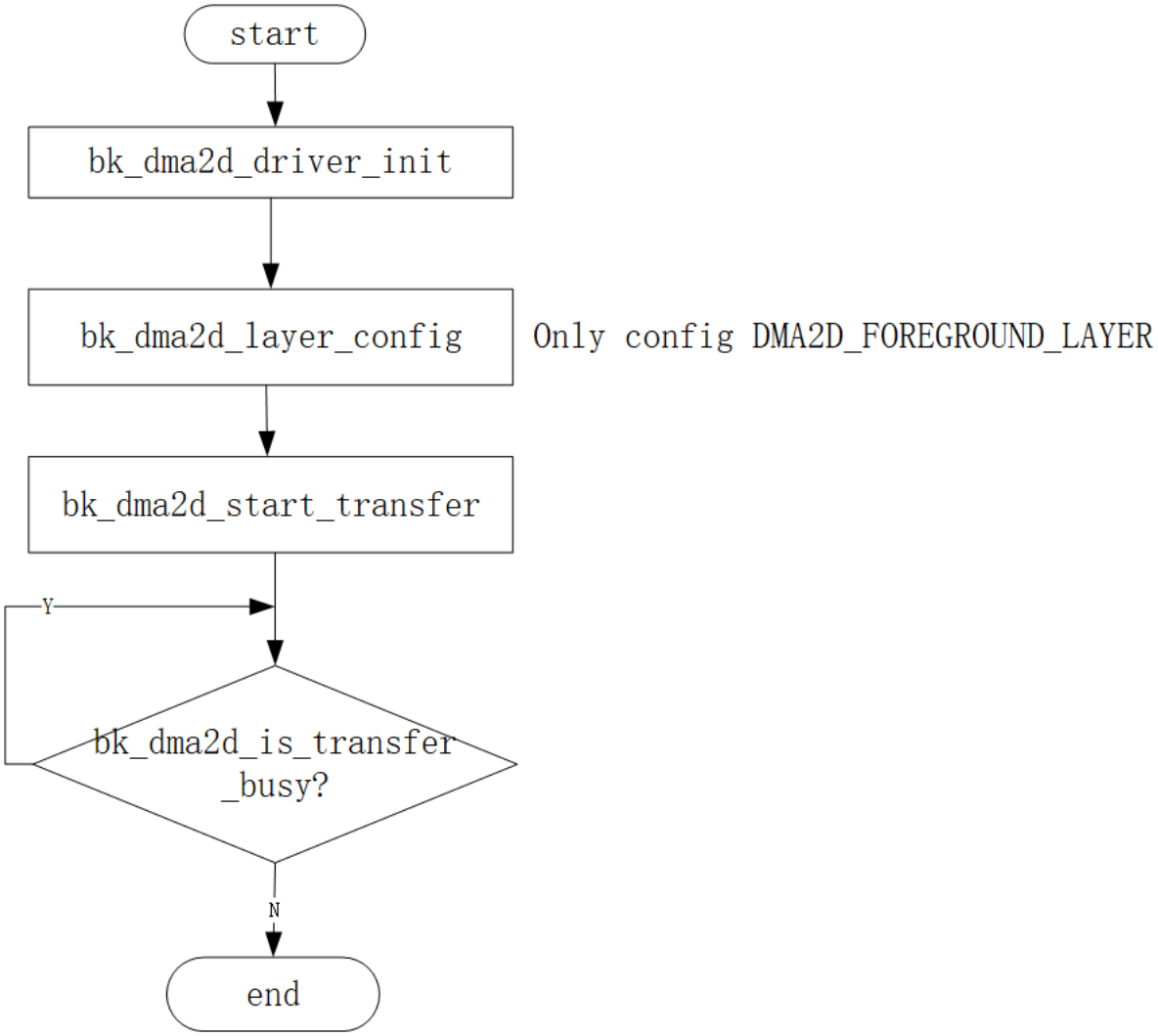


Fig. 6: Figure 2. dma2d_mem_to_mem

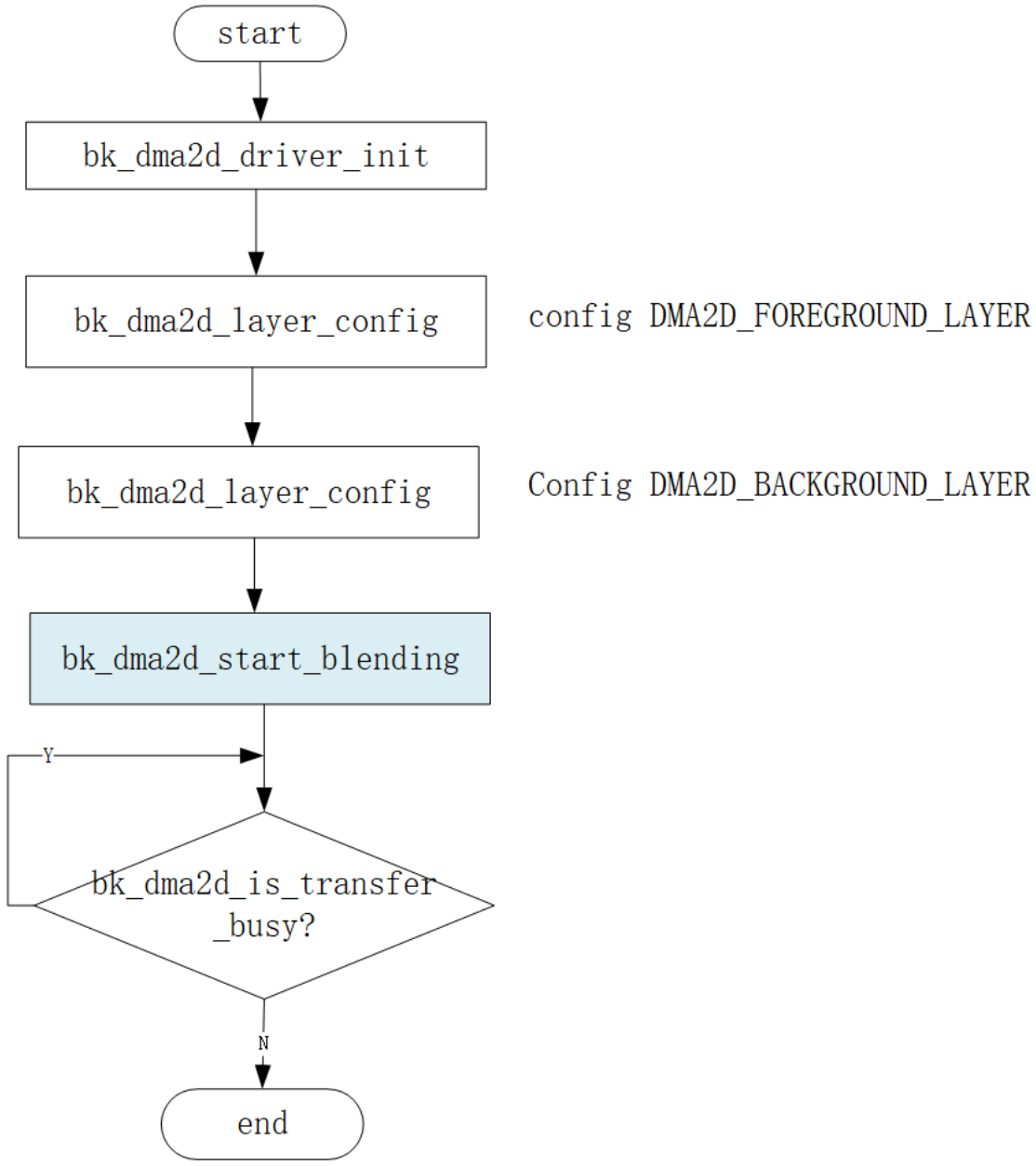


Fig. 7: Figure 3. dma2d_mem_to_mem_with_pixel_conversion_and_blending

API Reference

Header File

- dma2d.h

Functions

bk_err_t **bk_dma2d_driver_init**(void)
initializes the DMA2D system and peripheral registers

- open dma2d sys interrupt enable

Attention you can reference cli_dma2d.c for all API usage

Parameters **dma2d_config** – pointer to a *dma2d_config_t* structure that contains the configuration information for the DMA2D.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_init**(*dma2d_config_t* *dma2d)
config dma2d work mode/ data format/ offset etc.

Usage example:

```
      dma2d_config_t dma2d_config = {0};
      dma2d_config.init.mode = DMA2D_R2M;                      Mode Register to␣
↪Memory
      dma2d_config.init.color_mode = DMA2D_OUTPUT_RGB565;       DMA2D Output␣
↪color mode is ARGB4444 (16 bpp)
      dma2d_config.init.output_offset = 0;             No offset in output
      dma2d_config.init.red_blue_swap = DMA2D_RB_REGULAR;       No R&B swap for␣
↪the output image
      dma2d_config.init.alpha_inverted = DMA2D_REGULAR_ALPHA;   No alpha␣
↪inversion for the output image
      bk_dma2d_init(&dma2d_config);
```

Attention you can reference cli_dma2d.c for all API usage

Parameters **dma2d_config** – pointer to a *dma2d_config_t* structure that contains the configuration information for the DMA2D.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_driver_deinit**(void)
Deinitializes the DMA2D peripheral registers to their default reset values.

- reset the dma2d driver init reg

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_layer_config**(*dma2d_config_t* *dma2d, uint32_t layer_idx)

Configure the DMA2D background or foreground Layer include layer offset, color mode, alpha value etc.

Usage example:

```
      dma2d_config.layer_cfg[DMA2D_FOREGROUND_LAYER].alpha_mode = DMA2D_REPLACE_
↪ALPHA;
      dma2d_config.layer_cfg[DMA2D_FOREGROUND_LAYER].input_alpha = alpha_value;
      dma2d_config.layer_cfg[DMA2D_FOREGROUND_LAYER].input_color_mode = DMA2D_
↪INPUT_RGB565;
      dma2d_config.layer_cfg[DMA2D_FOREGROUND_LAYER].input_offset = fg_offline;
      dma2d_config.layer_cfg[DMA2D_FOREGROUND_LAYER].red_blue_swap  = DMA2D_RB_
↪REGULAR;
      dma2d_config.layer_cfg[DMA2D_FOREGROUND_LAYER].alpha_inverted = DMA2D_
↪REGULAR_ALPHA;

      Background layer Configuration
      dma2d_config.layer_cfg[DMA2D_BACKGROUND_LAYER].alpha_mode  = DMA2D_
↪REPLACE_ALPHA;
      dma2d_config.layer_cfg[DMA2D_BACKGROUND_LAYER].input_alpha   = 0x80;
      dma2d_config.layer_cfg[DMA2D_BACKGROUND_LAYER].input_color_mode = DMA2D_
↪INPUT_RGB565;
      dma2d_config.layer_cfg[DMA2D_BACKGROUND_LAYER].input_offset   = bg_
↪offline;
      dma2d_config.layer_cfg[DMA2D_BACKGROUND_LAYER].red_blue_swap   = DMA2D_RB_
↪REGULAR;
      dma2d_config.layer_cfg[DMA2D_BACKGROUND_LAYER].alpha_inverted  = DMA2D_
↪REGULAR_ALPHA;

      bk_dma2d_layer_config(&dma2d_config, DMA2D_FOREGROUND_LAYER);
      bk_dma2d_layer_config(&dma2d_config, DMA2D_BACKGROUND_LAYER);
```

Parameters - – dma2d Pointer to a *dma2d_config_t* structure that contains the configuration information for the DMA2D.

- LayerIdx DMA2D Layer index. This parameter can be one of the following values: DMA2D_BACKGROUND_LAYER(0) / DMA2D_FOREGROUND_LAYER(1)

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_transfer_config**(*dma2d_config_t* *dma2d, uint32_t pdata, uint32_t dst_addr, uint32_t width, uint32_t height)

Start the DMA2D Transfer when you use (bk_dma2d_driver_init) and (bk_dma2d_layer_config) API, then use this API start dma2d work.

- dst_addr: The destination memory Buffer address.

- Width: The width of data to be transferred from source to destination (expressed in number of pixels per line).
- Height: The height of data to be transferred from source to destination (expressed in number of lines).

Usage example:

```
      dma2d_config_t dma2d_config = {0};
      dma2d_config.init.mode    = DMA2D_R2M;                      Mode Register␣
↪to Memory
      dma2d_config.init.color_mode = DMA2D_OUTPUT_RGB565;       DMA2D Output␣
↪color mode is ARGB4444 (16 bpp)
      dma2d_config.init.output_offset = 0;          No offset in output
      dma2d_config.init.red_blue_swap = DMA2D_RB_REGULAR;       No R&B swap for␣
↪the output image
      dma2d_config.init.alpha_inverted = DMA2D_REGULAR_ALPHA;   No alpha␣
↪inversion for the output image

      bk_dma2d_driver_init(&dma2d_config);
      bk_dma2d_start_transfer(&dma2d_config, color, (uint32_t)dst_addr, width,␣
↪high);
```

Parameters - – dma2d: Pointer to a *dma2d_config_t* structure that contains the configuration information for the DMA2D.

- pdata: have two means:
  - 1: if the Memory-to-Memory or Memory-to-Memory with pixel format select, should Configure the source memory Buffer address
  - 2: if Register-to-Memory mode is selected, should configure the color value

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_blending_config**(*dma2d_config_t* *dma2d, uint32_t fg_addr, uint32_t bg_addr, uint32_t dst_addr, uint32_t width, uint32_t height)

Start the multi-source DMA2D Transfer. start foreground layer and background layer blending.

Usage example:

```
      1: bk_dma2d_driver_init(&dma2d_config);
      2: bk_dma2d_layer_config(&dma2d_config, DMA2D_FOREGROUND_LAYER);
      3: bk_dma2d_layer_config(&dma2d_config, DMA2D_BACKGROUND_LAYER);
      4:bk_dma2d_blending_start(&dma2d_config, (uint32_t)pFgaddr, (uint32_
↪t)pBgaddr, (uint32_t)pDst, xsize ,ysize);
```

Attention you can reference cli_dma2d.c for API usage

Parameters - – dma2d: Pointer to a *dma2d_config_t* structure that contains the configuration information for the DMA2D.

- fg_addr: The source memory Buffer address for the foreground layer.
- bg_addr: The source memory Buffer address for the background layer.
- dst_addr: The destination memory Buffer address.

- Width: The width of data to be transferred from source to destination (expressed in number of pixels per line).
- Height: The height of data to be transferred from source to destination (expressed in number of lines).

Returns

- BK_OK: succeed
- others: other errors.

bool **bk_dma2d_is_transfer_busy**(void)
this API is check dma2d is transfer busy or not

Usage example:

```
      1: bk_dma2d_driver_init(&dma2d_config);
      2: bk_dma2d_start_transfer(&dma2d_config, color, (uint32_t)dst_addr,␣
↪width, high);
      3: while (bk_dma2d_get_transfer_status()) {}
```

Returns return 0: transfer stop or transfer done;

- return 1 dma2d is work or transfer not done

bk_err_t **bk_dma2d_int_enable**(*dma2d_int_type_t* int_type, bool enable)
dma2d int enable.

Usage example:

```
      bk_dma2d_int_config(DMA2D_TRANS_ERROR | DMA2D_TRANS_COMPLETE ,1);
      always use with:  bk_dma2d_isr_register(dma2d_isr);
                        bk_dma2d_int_status_get();
                        bk_dma2d_int_status_clear(DMA2D_TRANS_ERROR_STATUS);
                        bk_dma2d_int_status_clear(DMA2D_TRANS_COMPLETE_STATUS);
```

Parameters

- **int_type** – select from dma2d_int_type_t, include int type:
  - DMA2D_CFG_ERROR
  - DMA2D_CLUT_TRANS_COMPLETE
  - DMA2D_CLUT_TRANS_ERROR
  - DMA2D_WARTERMARK_INT
  - DMA2D_TRANS_COMPLETE
  - DMA2D_TRANS_ERROR
- **enable** – 1:enable int, 0 disable int

Returns

- BK_OK: succeed
- others: other errors.

uint32_t **bk_dma2d_int_status_get**(void)
bk_dma2d_int_status_get.

Usage example:

```
uint32_t int_status;
int_status = bk_dma2d_int_status_get();
if (int_status & DMA2D_CFG_ERROR) {
   bk_dma2d_int_status_clear(DMA2D_CFG_ERROR_STATUS);
   bk_dma2d_int_config(DMA2D_CFG_ERROR, 0);
}
```

Returns return uint32_t value of all int status,

- can used by value & dma2d_int_status_t check which int triggered. typedef enum { DMA2D_TRANS_ERROR_STATUS = 0x1, DMA2D_TRANS_COMPLETE_STATUS, DMA2D_WARTERMARK_INT_STATUS, DMA2D_CLUT_TRANS_ERROR_STATUS, DMA2D_CLUT_TRANS_COMPLETE_STATUS, DMA2D_CFG_ERROR_STATUS }dma2d_int_status_t;

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_int_status_clear**(*dma2d_int_status_t* int_status)
clear dma2d int status

Parameters **int_status** – select from dma2d_int_status_t include: DMA2D_TRANS_ERROR_STATUS DMA2D_TRANS_COMPLETE_STATUS DMA2D_WARTERMARK_INT_STATUS DMA2D_CLUT_TRANS_ERROR_STATUS DMA2D_CLUT_TRANS_COMPLETE_STATUS DMA2D_CFG_ERROR_STATUS

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_isr_register**(*dma2d_isr_t* dma2d_isr)
register dma2d cpu int isr

Parameters **dma2d_isr** – the function you registr isr

Return values **bk_err_t** – status

bk_err_t **bk_dma2d_offset_blend**(*dma2d_offset_blend_t* *dma2d_blend)
config dma2d blend

Parameters **-** – *dma2d_offset_blend_t* struct to config blend params

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_blend**(*dma2d_blend_t* *dma2d_blend)
config dma2d blend

Parameters **-** – dma2d_blend struct to config blend params

Returns

- BK_OK: succeed

- others: other errors.

void **bk_dma2d_memcpy_or_pixel_convert**(*dma2d_memcpy_pfc_t* *pixel_convert)
config dma2d pixel revert

Parameters - – dma2d_pixel_convert_t struct to config pixel convert params

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **dma2d_fill**(*dma2d_fill_t* *fill)
config dma2d fill color

Parameters - – *dma2d_fill_t* struct to config dma2d fill params

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_start_transfer**(void)
start dma2d transfer

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_dma2d_stop_transfer**(void)
stop dma2d transfer

Returns

- BK_OK: succeed
- others: other errors.

void **dma2d_memcpy_psram**(void *Psrc, void *Pdst, uint32_t xsize, uint32_t ysize, uint32_t src_offline, uint32_t dest_offline)
dma2d_memcpy for src and dst addr is in psram

Parameters

- **Psrc.** – memcpy or partical copy start addr, in other word, the lcd frame addr add offset addr is the start copy addr.
- **Pdst.** – memcpy or partical copy dst addr, in other word, the dst frame addr add dst offset addr is the end copy addr.
- **xsize** – the memcpy or partical copy width
- **ysize** – the memcpy or partical copy hight
- **src_offline.** – for partical copy this is to calculate start copy addr based frame addr, uint by pixel
- **dest_offline** – for partical copy this is to calculate copy to dst addr based frame addr, uint by pixel

Returns none

void **dma2d_memcpy_psram_for_lvgl**(void *Psrc, uint32_t src_xsize, uint32_t src_ysize, void *Pdst, uint32_t dst_xsize, uint32_t dst_ysize, uint32_t src_xpos, uint32_t src_ypos, uint32_t dst_xpos, uint32_t dst_ypos)

void **dma2d_driver_transfes_ability**(*dma2d_trans_ability_t* trans_ability)
set dma2d transtfer ability

Parameters `select` – from dma2d_trans_ability_t

void **bk_dma2d_soft_reset**(void)
reset dma2d hardware logal , keep the config value

Retval

API Typedefs

Header File

- dma2d_types.h

Structures

struct **dma2d_init_t**
DMA2D Init structure definition.

Public Members

dma2d_mode_t **mode**
Configures the DMA2D transfer mode.his parameter can be one value of DMA2D_Mode.

uint32_t **color_mode**
Configures the color format of the output image. This parameter can be one value of DMA2D_Output_Color_Mode.

uint32_t **output_offset**
Specifies the Offset value. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.

uint32_t **alpha_inverted**
Select regular or inverted alpha value for the output pixel format converter. This parameter can be one value of DMA2D_Alpha_Inverted.

uint32_t **line_offset_mode**
Configures how is expressed the line offset for the foreground, background and output. This parameter can be one value of DMA2D_Line_Offset_Mode.

uint8_t **red_blue_swap**
Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR) for the output pixel format converter. This parameter can be one value of DMA2D_RB_Swap.

uint8_t **out_byte_by_byte_reverse**
DMA2D_BYTE_REVERSE or DMA2D_NO_REVERSE , normally for 32bit color of yuv format

dma2d_trans_ability_t **trans_ability**
set defult MAX_TRANS_256BYTES

struct **dma2d_layer_cfg_t**
DMA2D Layer structure definition

Public Members

uint32_t **input_offset**
Configures the DMA2D foreground or background offset. This parameter must be a number between Min_Data = 0x0000 and Max_Data = 0x3FFF.

uint32_t **input_color_mode**
Configures the DMA2D foreground or background color mode. This parameter can be one value of DMA2D_Input_Color_Mode.

uint32_t **alpha_mode**
Configures the DMA2D foreground or background alpha mode.This parameter can be one value of DMA2D_Alpha_Mode.

uint32_t **input_color**
Specifies color value in case of A8 or A4 color mode.

Note: In case of A8 or A4 color mode (ARGB), this parameter must be a number between Min_Data = 0x00000000 and Max_Data = 0xFFFFFF where

- Inputcolor[16:23] is the red value RED[0:7]
- Inputcolor[8:15] is the green value GREEN[0:7]
- Inputcolor[0:7] is the blue value BLUE[0:7].

---

uint32_t **alpha_inverted**
Select regular or inverted alpha value.This parameter can be one value of DMA2D_Alpha_Inverted.

uint32_t **red_blue_swap**
Select regular mode (RGB or ARGB) or swap mode (BGR or ABGR). This parameter can be one value of DMA2D_RB_Swap.

date_reverse_t **input_data_reverse**
input formart for yuv format, reverse data byte by byte or halfword by halfword.

struct **dma2d_config_t**

Public Members

dma2d_init_t **init**
dma2d init config, is the value of struct *dma2d_init_t*

dma2d_layer_cfg_t **layer_cfg**[MAX_DMA2D_LAYER]
dma2d layer config, is the value of struct *dma2d_layer_cfg_t* the param MAX_DMA2D_LAYER is select from DMA2D_BACKGROUND_LAYER and DMA2D_FOREGROUND_LAYER

struct **dma2d_blend_t**

Public Members

void *pfg_addr
The source memory Buffer address for the foreground layer..

void *pbg_addr
he source memory Buffer address for the background layer...

void *pdst_addr
The output memory Buffer address ..

input_color_mode_t fg_color_mode
fg color mode..

input_color_mode_t bg_color_mode
bg color mode..

out_color_mode_t dst_color_mode
out color mode..

uint32_t fg_offline
for partical copy this is to calculate start addr based on fg frame addr, uint by pixel

uint32_t bg_offline
for partical copy this is to calculate start addr based on bg frame addr, uint by pixel

uint32_t dest_offline
for partical copy this is to calculate output addr based on dst frame addr, uint by pixel

uint32 xsize
dma2d blend x size..

uint32 ysize
dma2d blend y size..

uint8_t fg_alpha_value
config fg alpha..

uint8_t bg_alpha_value
config bg alpha.

red_blue_swap_t red_bule_swap
DMA2D_RB_SWAP or DMA2D_RB_REGULAR

struct dma2d_offset_blend_t

Public Members

void *pfg_addr
The source memory Buffer address for the foreground layer..

void *pbg_addr
he source memory Buffer address for the background layer...

void *pdst_addr
The output memory Buffer address ..

input_color_mode_t fg_color_mode
fg color mode..

input_color_mode_t bg_color_mode
bg color mode..

out_color_mode_t dst_color_mode
out color mode..

uint16_t fg_frame_xpos
src img start copy/pfc x pos

uint16_t fg_frame_ypos
src img start copy/pfc y pos

uint16_t bg_frame_xpos
dma2d fill x pos based on frame_xsize

uint16_t bg_frame_ypos
dma2d fill y pos based on frame_ysize

uint16_t dst_frame_xpos
dma2d fill x pos based on frame_xsize

uint16_t dst_frame_ypos
dma2d fill y pos based on frame_ysize

uint16_t fg_frame_width
src image width

uint16_t fg_frame_height
src image height

uint16_t bg_frame_width
src image width

uint16_t bg_frame_height
src image height

uint16_t dst_frame_width
src image width

uint16_t **dst_frame_height**
src image height

uint32 **dma2d_width**
dma2d blend x size..

uint32 **dma2d_height**
dma2d blend y size..

uint8_t **fg_alpha_value**
config fg alpha..

uint8_t **bg_alpha_value**
config bg alpha.

red_blue_swap_t **fg_red_blue_swap**
fg img red blue swap, select DMA2D_RB_SWAP or DMA2D_RB_REGULAR

red_blue_swap_t **bg_red_blue_swap**
bg img red blue swap, select DMA2D_RB_SWAP or DMA2D_RB_REGULAR

red_blue_swap_t **dst_red_blue_swap**
dst img red blue swap, select DMA2D_RB_SWAP or DMA2D_RB_REGULAR

struct **dma2d_memcpy_pfc_t**

Public Members

void ***input_addr**
The image memcpy or pixel convert src addr

void ***output_addr**
The mage memcpy or pixel convert dst addr

uint16_t **src_frame_width**
memcpy or pfc src image width

uint16_t **src_frame_height**
imemcpy or pfc src image height

uint16_t **src_frame_xpos**
src img start copy/pfc x pos

uint16_t **src_frame_ypos**
src img start copy/pfc y pos

uint16_t **dst_frame_width**
memcpy to dst image, the dst image width

uint16_t **dst_frame_height**
memcpy to dst image, the dst image height

uint16_t **dst_frame_xpos**
dma2d fill x pos based on frame_xsize

uint16_t **dst_frame_ypos**
dma2d fill y pos based on frame_ysize

uint16_t **dma2d_width**
dma2d memcpy or pfc width

uint16_t **dma2d_height**
dma2d memcpy or pfc height

input_color_mode_t **input_color_mode**
The pixel convert src color mode

out_color_mode_t **output_color_mode**
The pixel convert dst color mode

uint8_t **input_alpha**
src data alpha, depend on alpha_mode

uint8_t **output_alpha**
dst data alpha,depend on alpha_mode

red_blue_swap_t **input_red_blue_swap**
src img red blue swap, select DMA2D_RB_SWAP or DMA2D_RB_REGULAR

red_blue_swap_t **output_red_blue_swap**
src img red blue swap, select DMA2D_RB_SWAP or DMA2D_RB_REGULAR

struct **dma2d_fill_t**

Public Members

void ***frameaddr**
dma2d fill frame baseaddr , normally LCD start frame addr

uint16_t **frame_xsize**
img/lcd x size

uint16_t **frame_ysize**
img/lcd y size

uint16_t **xpos**
dma2d fill x pos based on frame_xsize

uint16_t **ypos**
dma2d fill y pos based on frame_ysize

uint16_t **width**
dma2d fill width

uint16_t **height**
dma2d fill height

uint32_t **color**
dma2d fill color , if YUV datat fill , should fill RGB888 color data

Macros

BK_ERR_DMA2D_NOT_INIT
dma2d macos define

LCD driver not init

USE_HAL_DMA2D_REGISTER_CALLBACKS
if use int type isr register, set this value 1. always use with API @refs bk_dma2d_register_int_callback_isr

COLOR_BLACK

COLOR_NAVY

COLOR_DARKGREEN

COLOR_DARKCYAN

COLOR_MAROON

COLOR_PURPLE

COLOR_OLIVE

COLOR_LIGHTGREY

COLOR_DARKGREY

COLOR_BLUE

COLOR_GREEN

COLOR_CYAN

COLOR_RED

COLOR_MAGENTA

COLOR_YELLOW

COLOR_WHITE

COLOR_ORANGE

COLOR_GREENYELLOW

COLOR_PINK

COLOR_SILVER

COLOR_GRAY

COLOR_LIME

COLOR_TEAL

COLOR_FUCHSIA

COLOR_ESP_BKGD

DMA2D_OCOLR_BLUE_1
: define Mode_ARGB8888/RGB888 Blue Value

DMA2D_OCOLR_GREEN_1
: define Mode_ARGB8888/RGB888 Green Value

DMA2D_OCOLR_RED_1
: define Mode_ARGB8888/RGB888 Red Value

DMA2D_OCOLR_YELLOW_1
: define Mode_ARGB8888/RGB888 yellow Value

DMA2D_OCOLR_ALPHA_1
: define Mode_ARGB8888/RGB888Alpha Channel Value

DMA2D_OCOLR_BLUE_3
define Mode_ARGB1555 Blue Value

DMA2D_OCOLR_GREEN_3
define Mode_ARGB1555Green Value

DMA2D_OCOLR_RED_3
define Mode_ARGB1555 Red Value

DMA2D_OCOLR_ALPHA_3
define Mode_ARGB1555 Alpha Channel Value

DMA2D_OCOLR_BLUE_4
define Mode_ARGB4444 Blue Value

DMA2D_OCOLR_GREEN_4
define Mode_ARGB4444 Green Value

DMA2D_OCOLR_RED_4
define Mode_ARGB4444 Red Value

DMA2D_OCOLR_ALPHA_4
define Mode_ARGB4444 Alpha Channel Value

DMA2D_BACKGROUND_LAYER
DMA2D Background Layer (layer 0)

DMA2D_FOREGROUND_LAYER
DMA2D Foreground Layer (layer 1)

MAX_DMA2D_LAYER
DMA2D maximum number of layers

DMA2D_REGULAR_ALPHA
No modification of the alpha channel value

DMA2D_INVERTED_ALPHA
Invert the alpha channel value

DMA2D_NO_REVERSE
dma2d in/out put data in order not flip

DMA2D_BYTE_REVERSE
input/output data reerse byte by byte for 32 bits color of YUV

DMA2D_HFWORD_REVERSE
input/output data reerse half-word by half-word for 32 bits color of YUV

DMA2D_ISR_NUM

Type Definitions

typedef void (*dma2d_isr_t)(void)

Enumerations

enum dm2d_isr_id_t
if USE_HAL_DMA2D_REGISTER_CALLBACKS = 1, define int isr register id

Values:

enumerator DMA2D_CFG_ERROR_ISR

enumerator DMA2D_CLUT_TRANS_COMPLETE_ISR

enumerator DMA2D_CLUT_TRANS_ERROR_ISR

enumerator DMA2D_WARTERMARK_INT_ISR

enumerator DMA2D_TRANS_COMPLETE_ISR

enumerator DMA2D_TRANS_ERROR_ISR

enum dma2d_mode_t
DMA2D enum defines.

DMA2D_Mode

Values:

enumerator DMA2D_M2M
DMA2D memory to memory transfer mode

enumerator DMA2D_M2M_PFC
DMA2D memory to memory with pixel format conversion transfer mode

enumerator DMA2D_M2M_BLEND
DMA2D memory to memory with blending transfer mode

enumerator DMA2D_R2M
DMA2D register to memory transfer mode

enumerator DMA2D_M2M_BLEND_FG
DMA2D memory to memory with blending transfer mode and fixed color FG

enumerator DMA2D_M2M_BLEND_BG
DMA2D memory to memory with blending transfer mode and fixed color BG

enum **date_reverse_t**

Values:

enumerator **NO_REVERSE**
in output 32bit color(yuv) formart, not reverse data byte by byte

enumerator **BYTE_BY_BYTE_REVERSE**
in output 32bit color(yuv) formart, reverse data byte by byte

enumerator **HFWORD_BY_HFWORD_REVERSE**
in output 32bit color(yuv) formart, reverse data byte by byte

enum **out_color_mode_t**

DMA2D_Output_Color_Mode , used for fillfmt / memcpy dst fmt/ blend output fmt

Values:

enumerator **DMA2D_OUTPUT_ARGB8888**
ARGB8888 DMA2D color mode

enumerator **DMA2D_OUTPUT_RGB888**
RGB888 DMA2D color mode

enumerator **DMA2D_OUTPUT_RGB565**
RGB565 DMA2D color mode

enumerator **DMA2D_OUTPUT_ARGB1555**
ARGB1555 DMA2D color mode

enumerator **DMA2D_OUTPUT_ARGB4444**
ARGB4444 DMA2D color mode

enumerator **DMA2D_OUTPUT_YUYV**

enumerator **DMA2D_OUTPUT_UYVY**

enumerator **DMA2D_OUTPUT_YYUV**

enumerator **DMA2D_OUTPUT_UVYY**

enumerator **DMA2D_OUTPUT_VUYY**

enum **input_color_mode_t**

DMA2D_Input_Color_Mode, the members in order

Values:

enumerator **DMA2D_INPUT_ARGB8888**
ARGB8888 DMA2D color mode

enumerator **DMA2D_INPUT_RGB888**
RGB888 DMA2D color mode

enumerator **DMA2D_INPUT_RGB565**
RGB565 DMA2D color mode

enumerator **DMA2D_INPUT_ARGB1555**
ARGB1555 DMA2D color mode

enumerator **DMA2D_INPUT_ARGB4444**
ARGB4444 DMA2D color mode

enumerator **DMA2D_INPUT_L8**

enumerator **DMA2D_INPUT_AL44**

enumerator **DMA2D_INPUT_AL88**

enumerator **DMA2D_INPUT_L4**

enumerator **DMA2D_INPUT_A8**

enumerator **DMA2D_INPUT_A4**

enumerator **DMA2D_INPUT_YUYV**

enumerator **DMA2D_INPUT_UYVY**

enumerator **DMA2D_INPUT_YYUV**

enumerator **DMA2D_INPUT_UVYY**

enumerator **DMA2D_INPUT_VUYY**

enum **dma2d_int_type_t**
dma2d int type

Values:

enumerator **DMA2D_CFG_ERROR**

enumerator DMA2D_CLUT_TRANS_COMPLETE

enumerator DMA2D_CLUT_TRANS_ERROR

enumerator DMA2D_WARTERMARK_INT

enumerator DMA2D_TRANS_COMPLETE

enumerator DMA2D_TRANS_ERROR

enumerator DMA2D_ALL_INI

enum dma2d_int_status_t
dma2d int status

Values:

enumerator DMA2D_TRANS_ERROR_STATUS

enumerator DMA2D_TRANS_COMPLETE_STATUS

enumerator DMA2D_WARTERMARK_INT_STATUS

enumerator DMA2D_CLUT_TRANS_ERROR_STATUS

enumerator DMA2D_CLUT_TRANS_COMPLETE_STATU

enumerator DMA2D_CFG_ERROR_STATUS

enum dma2d_trans_ability_t
Values:

enumerator MAX_TRANS_256BYTES

enumerator TRANS_128BYTES

enumerator TRANS_64BYTES

enumerator **TRANS_32BYTES**

enumerator **TRANS_16BYTES**

enum **blend_alpha_mode_t**
Values:

enumerator **DMA2D_NO_MODIF_ALPHA**

enumerator **DMA2D_REPLACE_ALPHA**

enumerator **DMA2D_COMBINE_ALPHA**

enum **red_blue_swap_t**
Values:

enumerator **DMA2D_RB_REGULAR**
Select regular mode (RGB or ARGB)

enumerator **DMA2D_RB_SWAP**
Select swap mode (BGR or ABGR) * = 0

enum **color_bytes_t**
Values:

enumerator **TWO_BYTES**

enumerator **THREE_BYTES**

enumerator **FOUR_BYTES**

3.2.4 Rotate APIs

Rotate Interface

The main function of the Rotate module is to rotate the yuv422 image stored in memory into an image in rgb565 format, and store it in another memory after completion

The modules work in three modes:

1. Rotate clockwise mode
2. Counterclockwise rotation mode
3. Format conversion without rotation

- For rotation, if the input data is RGB565 data, then only rotation is performed

demo run dependent macro configuration:

Name	Description	File	value
CONFIG_HW_ROTATE_PFC	Configures the Rotate function	`\middleware\soc\bk7236\bk7236.defconfig`	y

Rotate Categories

API Reference

Header File

- rott_driver.h

Functions

bk_err_t **bk_rott_driver_init**(void)
This API used config sysclk.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_soft_reset**(void)
This API used reset rotate.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_driver_deinit**(void)
This API used disable rotate cpu int.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_int_enable**(*rott_int_type_t* int_type, bool en)
This API used to config rotate int type and int enable.

Parameters `1:int_type` – include:

```
      - ROTATE_COMPLETE_INT: rotate complete int
      - ROTATE_WARTERMARK_INT: if enable, wtmk_block need config,␣
↪and not set 0, you can set wtmk_block by api bk_rott_wartermark_
↪block_config config or rott_config
      - ROTATE_CFG_ERR_INT, the following will trigger cfg err int,
↪like:
                        - picture_xpixel & picture_ypixel be zero␣
↪while working.
                        - ROTATE_WARTERMARK_INT=1, but wtmk_block=0
                        - block_cnt=0
                        - block_xpixel > picture_xpixel,  block_
↪ypixel > picture_ypixel
2: en enable or disable
```

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_block_delay_config**(bool en, uint16_t delay_hclk_count1, uint16_t delay_hclk_count2)

This API used config rotate delay cycle in block and block interval.

Parameters `-` – en,enable the delay function

- delay_hclk_count1 (range 0 ~ 0x7fff), the delay pclk cycle is calculate by (delay_hclk_count1 * delay_hclk_count2)
- delay_hclk_count2 (range 0 ~ 0xffff), the delay pclk cycle is calculate by (delay_hclk_count1 * delay_hclk_count2)

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_mode_config**(media_rotate_t rot_mode)

This API used config rotate mode.

Parameters `struct` – rott_mode_t

- ROTT_ONLY_YUV2RGB565, only use yuv data pixel convert to rgb565 pixel, no rotate
- ROTT_ROTATE90, if rot_input_fmt = RGB565, just rotate; if rot_input_fmt = yuv, not only rotate but convert to rgb565.
- ROTT_ROTATE170,if rot_input_fmt = RGB565, just rotate; if rot_input_fmt = yuv, not only rotate but convert to rgb565.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_input_data_format**(pixel_format_t rot_input_fmt)

This API used config input data format, output data format is default rgb565.

Attention if your data rotate or pixel convert not display normally,you can use api bk_rott_data_reverse modify input/output data flow to see see

Parameters `struct` – rott_input_fmt_t

- RGB565, input format is rgb565 be

- RGB565le, input format is rgb565 le
- ROTT_FMT_YUYV
- ROTT_FMT_VUYY
- others

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_data_reverse**(*rott_input_data_flow_t* input_flow, *rott_output_data_flow_t* output_flow)

This API used config input data or output data flip(reverse)

Attention if your data rotate or pixel convert not display normally,you can use api bk_rott_data_reverse modify input/output data flow to see see

Parameters

- **struct** – rott_input_data_flow_t
  - ROTT_INPUT_NORMAL just input data, no flip
  - ROTT_INPUT_REVESE_BYTE_BY_BYTE, input data input convert or flip byte by byte
  - ROTT_INPUT_REVESE_HALFWORD_BY_HALFWORD, the input convert or flip halfword by halfword
- **struct** – rott_output_data_flow_t
  - ROTT_OUTPUT_NORMAL just output data, no flip outpuy
  - ROTT_OUTPUT_REVESE_HALFWORD_BY_HALFWORD, convert output data reversed(flip) halfword by halfword

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_block_rotate_config**(uint16_t picture_xpixel, uint16_t picture_ypixel, uint16_t block_xpixel, uint16_t block_ypixel, uint16_t block_cnt)

This API used config rotate param, rotate is by block, a block is max 4800 pixel,so (block_xpixel * block_ypixel) not over then 4800.

Param

bk_err_t **bk_rott_wr_addr_config**(uint32_t rd_base_addr, uint32_t wr_base_addr)

This API used config rotate src/dst addr.

Param

bk_err_t **bk_rott_wartermark_block_config**(uint16_t wtmk_block)

This API used config block threshold value,must make sure you enable ROTATE_WARTERMARK_INT int.

Usage example:

```
    A 640X480 picture, the param can set [640,480,80,60,64], you can set wtmk_
↪block 32 or 16 or 48 or other less then block_cnt value
```

Parameters **wtmk_block** – when rotate block is wtmk_block, will trigger ROTATE_WARTERMARK_INT

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_enable**(void)
This API used enable rotate, when rotate complete once, you should enable again to rotate.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_soft_rst**(void)
This API used reset ROTATE hardware logic.

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_isr_register**(*rott_isr_t* rott_isr)
This API register rotate isr.

Param

bk_err_t **bk_rott_int_status_clear**(*rott_int_type_t* int_type)
This API used to clear int status.

Usage example:

```
if ((bk_rott_int_status_get() & ROTATE_COMPLETE_INT))
 bk_rott_int_status_clear(ROTATE_COMPLETE_INT);
```

Parameters **-** – int type: you should used with bk_rott_int_status_get() api

Returns

- BK_OK: succeed
- others: other errors.

uint32_t **bk_rott_int_status_get**(void)
This API get int status.

Returns

- the int status value

bk_err_t **rott_config**(*rott_config_t* *rott)
This API is concentrate multiple api to realize rotate config.

Usage example:

```
- you can realize only yuv to  rgb565 no rotate mode like this:
    rott.rot_mode = ROTT_ONLY_YUV2RGB565
    rott.input_addr = (void*)&yuyv_640_480[0];
    rott.output_addr = (uint8_t *)0x60000000;
    rott.input_fmt = PIXEL_FMT_YUYV;
```

(continues on next page)

(continued from previous page)

```
    rott.input_flow = ROTT_INPUT_NORMAL;
    rott.output_flow = ROTT_OUTPUT_REVESE_HALFWORD_BY_HALFWORD;
    rott.picture_xpixel = 640;
    rott.picture_ypixel = 480;
    rott.block_xpixel = 80;
    rott.block_ypixel = 60;
    rott.block_cnt = 64;
    rott.watermark_blk = 32;
    rott_config(&rott);

- you can realize only yuyv to  rgb565 rotate 270 mode like this:
    rott.rot_mode = ROTT_ROTATE270;
    rott.input_addr = (void*)&yuyv_640_480[0];
    rott.output_addr = (uint8_t *)0x60000000;
    rott.input_fmt = PIXEL_FMT_YUYV;
    rott.input_flow = ROTT_INPUT_NORMAL;
    rott.output_flow = ROTT_OUTPUT_REVESE_HALFWORD_BY_HALFWORD;
    rott.picture_xpixel = ROTT_XPIXE;
    rott.picture_ypixel = ROTT_YPIXE;
    rott.block_xpixel = ROTT_XBLOCK;
    rott.block_ypixel = ROTT_YBLOCK;
    rott.block_cnt = ROTT_BLOCK_NUM;
    rott.watermark_blk = ROTT_BLOCK_NUM >> 1;
    rott_config(&rott);


- you can realize only  rgb565 rotate 270 mode like this:
    rott.rot_mode = ROTT_ROTATE270;
    rott.input_fmt = PIXEL_FMT_RGB565;
    rott.input_addr = XXXX;
    ................
```

Parameters **struct** – rott_input_data_flow_t

- those params is the same as every solo api

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_rott_transfer_ability**(*rott_trans_ability_t* ability)

set ahb master busrt len.

Parameters **0:64** – word per busrt; 1:32;2:16; 3:8; 4:4

API Typedefs

Header File

- rott_types.h

Structures

struct **rott_config_t**

Public Members

media_rotate_t **rot_mode**
bypass rotating, only use yuv2rgb565 pixel formart convert function

void ***input_addr**
input data addr

void ***output_addr**
output data addr

pixel_format_t **input_fmt**
pixel format, output rgb565 always

rott_input_data_flow_t **input_flow**
input data convert or not

rott_output_data_flow_t **output_flow**
output data convert or not

uint16_t **picture_xpixel**
image width

uint16_t **picture_ypixel**
image height

uint16_t **block_xpixel**
the following is optional, but should make sure rott_driver.c's TABLE_ROTT_BLOCK table is include used x/y pixel image width separated block width , must be devisible by picture_xpixel

uint16_t **block_ypixel**
image height separated block height, must be devisible by picture_ypixel

uint16_t **block_cnt**
block counts, must be (picture_xpixel/block_xpixel)*(picture_ypixel/block_ypixel) cnt

uint16_t **watermark_blk**
if enable watermark enable, you should config watermark_blk, normally defult one-half block cnt

Macros

USE_ROTT_REGISTER_CALLBACKS

BK_ERR_ROTT_NOT_INIT
rotate driver not init

Type Definitions

typedef void (*__rott_isr_t__)(void)

Enumerations

enum **rott_trans_ability_t**
Values:

enumerator **ROTATE_BURST_64**

enumerator **ROTATE_BURST_32**

enumerator **ROTATE_BURST_16**

enumerator **ROTATE_BURST_8**

enumerator **ROTATE_BURST_4**

enum **rott_int_type_t**
Values:

enumerator **ROTATE_COMPLETE_INT**
rotate finish int enable

enumerator **ROTATE_WARTERMARK_INT**
rotate watermark int enable.

enumerator **ROTATE_CFG_ERR_INT**
rotate config error int enable

enum **rott_input_data_flow_t**
Values:

enumerator **ROTT_INPUT_NORMAL**
input data not flip

enumerator **ROTT_INPUT_REVESE_BYTE_BY_BYTE**
convert input data revese byte by byte.

enumerator **ROTT_INPUT_REVESE_HALFWORD_BY_HALFWORD**
convert input data revese halfword by halfword

enum **rott_output_data_flow_t**
Values:

enumerator **ROTT_OUTPUT_NORMAL**
output data not flip

enumerator **ROTT_OUTPUT_REVESE_HALFWORD_BY_HALFWORD**
convert output data reverse halfword by halfword

3.2.5 SBC APIs

Important: The SBC Driver is just only APIs for SBC decoding function. And the SBC API v1.0 is the lastest and stable SBC decoder APIs. All new applications should use SBC API v1.0.

SBC User Guide

The sbc and msbc are used as follow:

SBC Interface

The BK SBC Driver supports following functions:

- sbc function
- msbc function

The SBC Driver support msbc decoder function. If the application need to use msbc decoder function, Just use the bk_sbc_decoder_support msbc() to enable msbc function.

SBC API Categories

Most of SBC APIs can be categoried as:

- Common or sbc APIs

  The common APIs are prefixed with bk_sbc, the APIs may be common for sbc and msbc interfaces, e.g. bk_sbc_decoder_init() etc.

- MSBC function APIs.

  The APIs provide support for msbc function. e.g. bk_sbc_decoder_support msbc().

Common APIs:

- *bk_sbc_decoder_init()* - init the sbc module
- *bk_sbc_decoder_deinit()* - deinit the sbc module
- *bk_sbc_decoder_bit_allocation()* - calculate the bit allocation

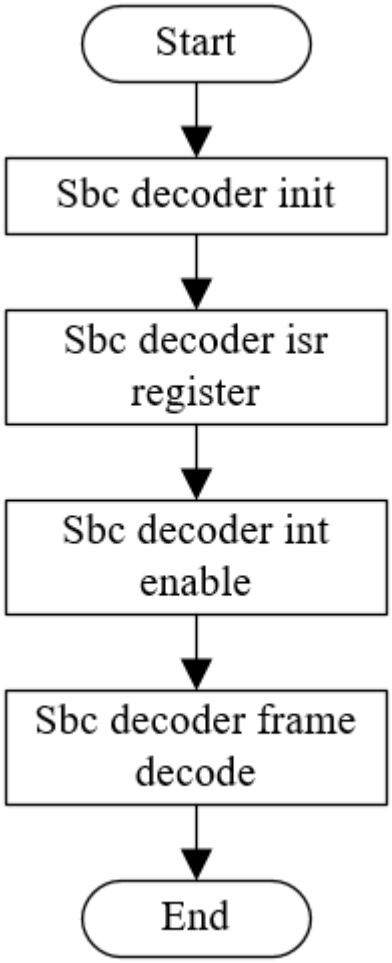


Fig. 8: SBC User Guide Flow

- *bk_sbc_decoder_frame_decode()* - decode one sbc frame
- *bk_sbc_decoder_interrupt_enable()* - enable or disable the sbc interrupt
- *bk_sbc_decoder_support_msbc()* - enable or disable msbc decoder function
- *bk_sbc_decoder_register_sbc_isr()* - register sbc isr

API Reference

Header File

- sbc.h

Functions

bk_err_t **bk_sbc_decoder_frame_decode**(*sbcdecodercontext_t* *sbc, const uint8_t *data, uint32_t length)

SBC decoder decode one frame.

This API decode one frame by sbc decoder.

Usage example:

sbcdecodercontext_t sbc_decoder; bk_sbc_decoder_frame_decode(&sbc_decoder, sbc_data, 512);

Attention 1. the output PCM data please refer to the follows variables:

- sbc->pcm_sample: means output PCM data address
- sbc->pcm_length: means output PCM data length in sample
- sbc->channel_number: means output PCM data channels

Parameters – – sbc: sbc decoder context pointer;

- data: buffer to be decoded;
- length: the length of input buffer;

Returns

- consumed: buffer length by decoder if no error ocurs, else error code (always small than 0) will be return.

bk_err_t **bk_sbc_decoder_bit_allocation**(*sbccommoncontext_t* *sbc)

SBC bit allocation calculate for both encoder and decoder.

This API calculate sbc bit allocation.

Parameters – – sbc: sbc decoder context pointer

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_sbc_decoder_init**(*sbcdecodercontext_t* *sbc)

SBC decoder initialize.

This API init the sbc decoder function.

Parameters – – sbc: sbc decoder context pointer

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_sbc_decoder_deinit**(void)
SBC decoder deinit.

This API deinit the sbc decoder function.

Parameters - – None

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_sbc_decoder_interrupt_enable**(bool enable)
enable/disable sbc interrupt

This API enable or disable sbc interrupt:

- reigster interrupt service handle
- enable or disable sbc interrupt

Parameters - – enable: enable —— 1, disable —— 0

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_sbc_decoder_support_msbc**(bool enable)
enable/disable msbc decoder

This API enable/disable msbc decoder.

Parameters - – enable: enable —— 1, disable —— 0

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_sbc_decoder_register_sbc_isr**(*sbc_decoder_isr_t* isr, void *param)
Register sbc decoder isr.

This API register sbc decoder isr.

Parameters - – isr: sbc decoder isr callback;

- param: sbc decoder isr callback parameter;

Returns

- BK_OK: succeed
- others: other errors.

API Typedefs

Header File

- sbc_types.h

Structures

struct **sbccommoncontext_t**
SBC decoder context.

Public Members

int8_t **blocks**
block number

int8_t **subbands**
subbands number

uint8_t **join**
bit number x set means joint stereo has been used in subband x

uint8_t **bitpool**
indicate the size of the bit allocation pool that has been used for encoding the stream

int8_t **channel_mode**
channel mode

int8_t **sample_rate_index**
sample rate index, 0:16000, 1:32000, 2:44100, 3:48000

int8_t **allocation_method**
allocation method

int8_t **reserved8**
dummy, reserved for byte align

int8_t **bits**[2][8]
calculate result by bit allocation

int8_t **scale_factor**[2][8]
only the lower 4 bits of every element are to be used

int32_t **mem**[2][8]
Memory used as bit need and levels

struct **sbcdecodercontext_t**
SBC decoder context.

Public Members

int8_t **channel_number**
channels number

uint8_t **pcm_length**
PCM length

uint16_t **sample_rate**
sample rate

int32_t **pcm_sample**[2][128]
PCM frame buffer

int32_t **vfifo**[2][170]
FIFO V for subbands synthesis calculation.

struct **sbc_config_t**

Public Members

sbc_channel_num_t **channel_num**
channels number

sbc_subband_number_t **subbands**
subband number

sbc_chn_comb_t **chn_comb**
pcm output struct

sbc_blocks_number_t **blocks**
block number

Macros

SBC_SYNCWORD
SBC synchronize word

MSBC_SYNCWORD
MSBC synchronize word

SBC_CHANNEL_MODE_MONO
SBC channel mode : MONO

SBC_CHANNEL_MODE_DUAL
SBC channel mode : Dual Channels

SBC_CHANNEL_MODE_STEREO
SBC channel mode : Stereo

SBC_CHANNEL_MODE_JOINT_STEREO
SBC channel mode : Joint Stereo

SBC_DECODER_ERRORS

SBC_DECODER_ERROR_BUFFER_OVERFLOW
buffer overflow

SBC_DECODER_ERROR_SYNC_INCORRECT
synchronize incorrect

SBC_DECODER_ERROR_BITPOOL_OUT_BOUNDS
bitpool out of bounds

SBC_DECODER_ERROR_STREAM_EMPTY
stream empty

SBC_DECODER_ERROR_OK
no error

Type Definitions

typedef void (*__sbc_decoder_isr_t__)(void *param)

Enumerations

enum **sbc_sample_rates_t**
Values:

enumerator **SBC_SAMPLE_RATE_16000**
SBC sampling frequency : 16.0KHz

enumerator **SBC_SAMPLE_RATE_32000**
SBC sampling frequency : 32.0KHz

enumerator **SBC_SAMPLE_RATE_44100**
SBC sampling frequency : 44.1KHz

enumerator **SBC_SAMPLE_RATE_48000**
SBC sampling frequency : 48.0KHz

enumerator **SBC_SAMPLE_RATE_MAX**

enum **sbc_channel_num_t**
Values:

enumerator **SBC_CHANNEL_NUM_ONE**
SBC channel number : 1 channel

enumerator **SBC_CHANNEL_NUM_TWO**
SBC channel number : 2 channels

enumerator **SBC_CHANNEL_MODE_INVALID**

enum **sbc_subband_number_t**
Values:

enumerator **SBC_SUBBAND_NUMBER_4**
SBC subbands number 4

enumerator **SBC_SUBBAND_NUMBER_8**
SBC subbands number 8

enumerator **SBC_SUBBAND_MAX**

enum **sbc_chn_comb_t**
Values:

enumerator **SBC_DECODE_OUTPUT_SINGLE**
SBC decoder output struct:{16'd0, pcm}

enumerator **SBC_DECODE_OUTPUT_DOUBLE**
SBC decoder output struct:{pcm_chn2, pcm_chn1}

enumerator **SBC_DECODE_OUTPUT_INVALID**

enum **sbc_blocks_number_t**
Values:

enumerator **SBC_BLOCK_NUMBER_4**
SBC blocks number 4

enumerator **SBC_BLOCK_NUMBER_8**
SBC blocks number 8

enumerator **SBC_BLOCK_NUMBER_12**
SBC blocks number 12

enumerator **SBC_BLOCK_NUMBER_16**
SBC blocks number 16

enumerator **SBC_BLOCK_NUMBER_MAX**

enum **sbc_allocation_method_t**
Values:

enumerator **SBC_ALLOCATION_METHOD_LOUDNESS**
SBC allocation method : Loudness

enumerator **SBC_ALLOCATION_METHOD_SNR**
SBC allocation method : SNR

enumerator **SBC_ALLOCATION_METHOD_INVALID**

3.2.6 MP3 APIs

Important: The MP3 API v1.0 is the lastest stable MP3 APIs. It supports MPEG-1, MPEG-2 and MPEG-2.5 standard Layer3 decoding. All new applications should use MP3 API v1.0.

MP3 API Categories

MP3 APIs:

- *MP3InitDecoder()* - allocate memory for platform-specific data and clear all the user-accessible fields
- *MP3FreeDecoder()* - free platform-specific data allocated by InitMP3Decoder and zero out the contents of MP3DecInfo struct
- *MP3FindSyncWord()* - locate the next byte-alinged sync word in the raw mp3 stream
- *MP3Decode()* - decode one frame of MP3 data
- *MP3GetLastFrameInfo()* - get info about last MP3 frame decoded (number of sampled decoded, sample rate, bitrate, etc.)
- *MP3GetNextFrameInfo()* - parse MP3 frame header

API Reference

Header File

- mp3dec.h

Functions

HMP3Decoder **MP3InitDecoder**(void)
Init the decoder of MP3.

This API init the mp3 decoder module:

- allocate memory for platform-specific data
- clear all the user-accessible fields

Parameters **None** –

Returns None

void **MP3FreeDecoder**(*HMP3Decoder* hMP3Decoder)
Free the decoder of MP3.

This API free the mp3 decoder module:

- free platform-specific data allocated by InitMP3Decoder

- zero out the contents of MP3DecInfo struct

Parameters **HMP3Decoder** – valid MP3 decoder instance pointer

Returns None

int **MP3Decode**(*HMP3Decoder* hMP3Decoder, unsigned char **inbuf, int *bytesLeft, short *outbuf, int useSize)

Decode one frame of MP3 data.

Parameters

- **hMP3Decoder** – valid MP3 decoder instance pointer
- **inbuf** – double pointer to buffer of MP3 data (containing headers + mainData)
- **bytesLeft** – number of valid bytes remaining in inbuf
- **outbuf** – pointer to outbuf, big enough to hold one frame of decoded PCM samples
- **useSize** – flag indicating whether MP3 data is normal MPEG format (useSize = 0) or reformatted as "self-contained" frames (useSize = 1)

Returns error code, defined in mp3dec.h (0 means no error, < 0 means error)

void **MP3GetLastFrameInfo**(*HMP3Decoder* hMP3Decoder, *MP3FrameInfo* *mp3FrameInfo)

Get info about last MP3 frame decoded.

This API get the info about last mp3 decoded:

- number of sampled decoded, sample rate, bitrate, etc.

Parameters

- **hMP3Decoder** – valid MP3 decoder instance pointer
- **mp3FrameInfo** – pointer to MP3FrameInfo struct

Returns None

int **MP3GetNextFrameInfo**(*HMP3Decoder* hMP3Decoder, *MP3FrameInfo* *mp3FrameInfo, unsigned char *buf)

Parse MP3 frame header.

Parameters

- **hMP3Decoder** – valid MP3 decoder instance pointer
- **mp3FrameInfo** – pointer to MP3FrameInfo struct
- **buf** – pointer to buffer containing valid MP3 frame header

Returns error code, defined in mp3dec.h (0 means no error, < 0 means error)

int **MP3FindSyncWord**(unsigned char *buf, int nBytes)

locate the next byte-alinged sync word in the raw mp3 stream

Parameters

- **buf** – buffer to search for sync word
- **nBytes** – max number of bytes to search in buffer

Returns

- offset to first sync word (bytes from start of buf)
- -1 if sync not found after searching nBytes

int **MP3SetBuffMethod**(void *(*alloc)(size_t size), void (*free)(void *buff), void *(*mset)(void *s, unsigned char c, size_t n))

indicate decoder which function to alloc free and memset This function must call before MP3InitDecoder or after MP3FreeDecoder.

Parameters

- **alloc** – alloc function
- **free** – free function
- **mset** – memset function

Returns

- 0: success
- other: err

int **MP3SetBuffMethodAlwaysFourAlignedAccess**(void *(*alloc)(size_t size), void (*free)(void *buff), void *(*mset)(void *s, unsigned char c, size_t n))

indicate decoder which function to alloc free and memset This function must call before MP3InitDecoder or after MP3FreeDecoder. This function will be used always 4 bytes access. This function is high priority than MP3SetBuffMethod

Parameters

- **alloc** – alloc function
- **free** – free function
- **mset** – memset function

Returns

- 0: success
- other: err

Structures

struct **_MP3FrameInfo**

Macros

MAINBUF_SIZE

MAX_NGRAN

MAX_NCHAN

MAX_NSAMP

Type Definitions

typedef void *HMP3Decoder

typedef struct *_MP3FrameInfo* MP3FrameInfo

Enumerations

enum MPEGVersion
Values:

enumerator MPEG1

enumerator MPEG2

enumerator MPEG25

enum err_mp3_t
Values:

enumerator ERR_MP3_NONE

enumerator ERR_MP3_INDATA_UNDERFLOW

enumerator ERR_MP3_MAINDATA_UNDERFLOW

enumerator ERR_MP3_FREE_BITRATE_SYNC

enumerator ERR_MP3_OUT_OF_MEMORY

enumerator ERR_MP3_NULL_POINTER

enumerator ERR_MP3_INVALID_FRAMEHEADER

enumerator ERR_MP3_INVALID_SIDEINFO

enumerator ERR_MP3_INVALID_SCALEFACT

enumerator **ERR_MP3_INVALID_HUFFCODES**

enumerator **ERR_MP3_INVALID_DEQUANTIZE**

enumerator **ERR_MP3_INVALID_IMDCT**

enumerator **ERR_MP3_INVALID_SUBBAND**

enumerator **ERR_UNKNOWN**

3.2.7 DVP_CAMERA APIs

Important: The DVP_CAMERA API v1.0 is the lastest stable DVP_CAMERA APIs. All new applications should use DVP_CAMERA API v1.0.

DVP_CAMERA Interface

Current software version support two types dvp sensor: gc0328c and hm1055. they are comunicate by i2c with chip. gc0328c have 20pins interface, hm1055 have 24pins interface.

gc0328c support different image resolution and frame rate: 1. WQVGA(480*272) 2. QVGA(320*240) 3. VGA(640*480) frame rate: 5fps/10fps/20fps/25fps

hm1055 only support 720p(1280*720) image resolution, and different frame rates: 5fps/15fps/20fps

DVP_CAMERA API Categories

Most of DVP_CAMERA APIs can be categoried as:

- DVP_CAMERA APIs

  The common APIs are prefixed with bk_camera, e.g. bk_camera_driver_init() etc.

DVP_CAMERA APIs:

- `bk_dvp_camera_driver_init()` - init dvp camera(power on sensor, begin to sample image)
- `bk_dvp_camera_driver_deinit()` - deinit dvp camera(power off sensor)
- *`bk_dvp_camera_get_device()`* - get current dvp dendor config
- `bk_dvp_camera_encode_config()` - change encode image upper and lower size or close auto encode
- `dvp_camera_i2c_read_uint8()` - master read dvp senor register value by byte
- `dvp_camera_i2c_read_uint16()` - master read dvp senor register value by double byte
- `dvp_camera_i2c_read_uint8()` - master write dvp senor register value by byte
- `dvp_camera_i2c_write_uint16()` - master write dvp senor register value by double byte

API Reference

Header File

- dvp_camera.h

Functions

const *dvp_sensor_config_t* ****get_sensor_config_devices_list**(void)

int **get_sensor_config_devices_num**(void)

const *dvp_sensor_config_t* ***get_sensor_config_interface_by_id**(*sensor_id_t* id)

void **bk_dvp_set_devices_list**(const *dvp_sensor_config_t* **list, uint16_t size)

This API set dvp camera device list.

Returns

- void

const *dvp_sensor_config_t* ***bk_dvp_detect**(void)

enumerate dvp camera

This API will auto detect dvp sensor, and init sensor, i2c module, psram, dma, ect.

@attation 1. bk_dvp_camera_driver_init api include bk_dvp_camera_enumerate function

Parameters **config** – process for sensor data

Returns

- dvp_sensor_config: sensor ptr
- NULL: not found

bk_err_t **bk_dvp_init**(camera_handle_t *handle, *dvp_config_t* *cfg, *bk_dvp_callback_t* *cb)

Init the camera.

This API will auto detect dvp sensor, and init sensor, jpeg module, i2c module, psram, dma, ect.

Parameters

- **handle** – output handle
- **config** – process for sensor data
- **cb** – frame buffer ops callback @attation 1. you need make sure upper module exist.

Returns

- kNoErr: succeed
- others: other errors.

bk_err_t **bk_dvp_deinit**(camera_handle_t *handle)

Deinit the camera.

This API will deinit sensor, jpeg module, i2c module, psram, dma, ect.

Parameters **handle** – output handle when open

Returns

- kNoErr: succeed

- others: other errors.

media_camera_device_t *bk_dvp_camera_get_device(void)
Get current use camera.

This API will called after bk_dvp_camera_driver_init

Returns

- dvp_camera_device_t *: succeed
- NULL: current no sensor work.

bk_err_t bk_dvp_camera_power_enable(uint8_t enable)
dvp power on

This API called by user, before calling bk_dvp_driver_init, you should power on dvp

@attation 1. This api config ctrl by marco VIDEO_GPIO_CTRL_LDO_ENABLE

Parameters enable – power up/down:1/0

Returns

- BK_OK: succeed
- others: other errors.

const *dvp_sensor_config_t* *bk_dvp_get_sensor_auto_detect(void)
auto detect dvp camera

This API called by user, before use dvp camera

@attation 1. This api return a pointer for dvp camera sensor config

Returns

- get current dvp config(type)
- return NULL

bk_err_t bk_dvp_h264_idr_reset(void)
regenerate idr frame

This API called by user, once call this api, will regenerate idr frame

@attation 1. This api only effect in camera is working, and work in h264 mode or h264&yuv mode,

Returns

- BK_OK: succeed
- others: other errors.

API Typedefs

Header File

- dvp_camera_types.h

Structures

struct **dvp_config_t**

struct **bk_dvp_callback_t**

Public Members

void *(***frame_init**)(uint16_t sensor_id, uint8_t camera_type, uint16_t image_format)
init frame buffer

int (***frame_deinit**)(void *node)
deinit frame buffer

int (***frame_clear**)(void *node)
clear frame buffer

int (***frame_complete**)(uint16_t image_format, void *node, frame_buffer_t *buffer)
dvp notify upper layer a complete frame ready

frame_buffer_t *(***frame_malloc**)(uint16_t image_format, void *node, uint32_t size)
malloc jpeg psram buffer

struct **dvp_sensor_config_t**
dvp sensor configure

Public Members

char ***name**
sensor name

media_ppi_t **def_ppi**
sensor default resolution

frame_fps_t **def_fps**
sensor default fps

mclk_freq_t **clk**
sensor work clk in config fps and ppi

pixel_format_t **fmt**
sensor input data format

sync_level_t **vsync**
sensor vsync active level

sync_level_t **hsync**
sensor hsync active level

uint16_t **id**
sensor type, sensor_id_t

uint16_t **address**
sensor write register address by i2c

uint16_t **fps_cap**
sensor support fps

uint16_t **ppi_cap**
sensor support resoultions

bool (*__detect__)(void)
auto detect used dvp sensor

int (*__init__)(void)
init dvp sensor

int (*__set_ppi__)(media_ppi_t ppi)
set resolution of sensor

int (*__set_fps__)(frame_fps_t fps)
set fps of sensor

int (*__power_down__)(void)
power down or reset of sensor

int (*__dump_register__)(media_ppi_t ppi)
dump sensor register

void (*__read_register__)(bool enable)
read sensor register when write

Macros

GC_QVGA_USE_SUBSAMPLE

Enumerations

enum **sensor_id_t**
dvp sensor id type

Values:

enumerator **ID_UNKNOW**

enumerator **ID_PAS6329**

enumerator **ID_OV7670**

enumerator **ID_PAS6375**

enumerator **ID_GC0328C**

enumerator **ID_BF2013**

enumerator **ID_GC0308C**

enumerator **ID_HM1055**

enumerator **ID_GC2145**

enumerator **ID_OV2640**

enumerator **ID_GC0308**

enumerator **ID_TVP5150**

enumerator **ID_SC101**

3.2.8 UVC_CAMERA APIs

Important: The UVC_CAMERA API v1.0 is the lastest stable UVC_CAMERA APIs. All new applications should use UVC_CAMERA API v1.0.

UVC_CAMERA Interface

UVC camera only support full speed type

UVC_CAMERA API Categories

Most of UVC_CAMERA APIs can be categoried as:

- UVC_CAMERA APIs

  The common APIs are prefixed with bk_uvc, e.g. bk_uvc_init() etc.

UVC_CAMERA APIs:

- `bk_uvc_camera_driver_init()` - init uvc camera(open uvc device)
- `bk_uvc_camera_driver_deinit()` - deinit uvc camera(close uvc device)
- `bk_uvc_camera_power_on()` - power on uvc
- `bk_uvc_camera_get_config()` - set uvc camera pps(image resolution) and fps(frame rate)
- `bk_uvc_camera_set_config()` - set uvc camera pps(image resolution) and fps(frame rate)
- `bk_uvc_camera_driver_start()` - start uvc, start simple image information
- `bk_uvc_camera_driver_stop()` - stop uvc, stop simple image information
- `bk_uvc_camera_get_device()` - get current working uvc device

API Reference

Header File

- uvc_camera.h

Functions

bk_err_t **bk_uvc_power_on**(uint32_t format, uint32_t timeout)

uvc power on

This API called by user, power on all port and register connect/disconnect cb

Parameters **trigger** – 0:not wait uvc connect, 1: wait uvc connect 4 seconds

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_uvc_power_off**(void)

uvc power on

This API called by user, power off all port

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_uvc_init**(camera_handle_t *handle, uvc_config_t *config, bk_uvc_callback_t *cb)

Init the uvc.

This API open uvc

Parameters

- **open** – handle
- **config** – The port config
- **cb** – The frame_buffer callback

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_uvc_deinit**(camera_handle_t *handle)
Deinit the uvc.

This API stop and close uvc

@attation: port must 1 - other

Parameters **handle** – The uvc open handle output

Returns

- BK_OK: succeed
- others: other errors.

bk_usb_hub_port_info ***bk_uvc_get_enum_info**(uint8_t port, uint16_t format)
Get uvc config.

This API called by user, get port info

Parameters

- **port** – The port user want to know
- **format** – The image format you want to know(image_format_t)

Returns

- PTR: succeed
- NULL: other errors.

bk_err_t **bk_uvc_set_start**(camera_handle_t *handle, uvc_config_t *config)
Set uvc config.

This API called by user, Set uvc support fps and resolutions and start stream

Parameters

- **handle** – the uvc init device handle
- **config** – the uvc need support param the user set

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_uvc_set_stop**(camera_handle_t *handle)
Set uvc config.

This API called by user, Set stop uvc stream output

Parameters **handle** – the uvc init device handle

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_uvc_register_packet_cb**(void *cb)
register uvc packet analyse callback

This API will register uvc analyse callabck

Parameters **cb** – @attation 1. this api called before bk_uvc_init. if not register, analyse uvc packet by default adk function

Returns

- BK_OK: set success
- others: other errors.

bk_err_t **bk_uvc_register_connect_state_cb**(camera_state_cb_t cb)
register uvc connect state cb

This API will register uvc connect state cb, include uvc state change cb, and other ops error

Parameters **cb** – @attation 1. this api called before bk_uvc_power_on.

Returns

- BK_OK: set success
- others: other errors.

bk_err_t **bk_uvc_register_separate_packet_callback**(uvc_separate_config_t *cb)
register separate packet cb

This API will register separate packet data, some uvc device a endpoint will output mjpeg and h264 data meantime, need register this callback

Parameters **cb** – @attation 1. this api called before bk_uvc_init, not all uvc need register

Returns

- BK_OK: set success
- others: other errors.

API Typedefs

3.2.9 VIDEO_TRANSFER APIs

Important: The VIDEO_TRANSFER API v1.0 is the lastest stable VIDEO_TRANSFER APIs. All new applications should use VIDEO_TRANSFER API v1.0.

VIDEO_TRANSFER API Categories

Most of VIDEO_TRANSFER APIs can be categoried as:

- VIDEO_TRANSFER APIs

  The common APIs are prefixed with bk_video, e.g. bk_video_transfer_init() etc.

VIDEO_TRANSFER APIs:

- *bk_video_transfer_init()* - init video transfer
- *bk_video_transfer_deinit()* - deinit video transfer
- *bk_video_buffer_open()* - open video buff
- *bk_video_buffer_close()* - close video buff
- *bk_video_buffer_read_frame()* - read fix length video data

API Reference

Header File

- video_transfer.h

Functions

bk_err_t **bk_video_transfer_init**(*video_setup_t* *setup_cfg)
video transfer init

This API will create video thread, init msg queue, and excute camera init

Parameters **setup_cfg** – configure of the video transfer include packet process, method of transfer, etc.

Returns

- kNoErr: succeed
- others: other errors.

bk_err_t **bk_video_transfer_deinit**(void)
video transfer deinit

This API will quit video thread, and free all reasource

Returns

- kNoErr: succeed
- others: other errors.

bk_err_t **bk_video_buffer_open**(media_camera_device_t *device)
set video transfer param

This API will open video tranfer data_buffer, and star transfer

Attention 1. when call this function, the video date will transfer to wifi(UDP/others)

Parameters **device** – referenc type : meida_camera_device_t

Returns

- kNoErr: succeed
- others: other errors.

bk_err_t **bk_video_buffer_close**(void)
set video buffer close

This API will deinit video tranfer, stop transfer video data

Returns

- kNoErr: succeed
- others: other errors.

uint32_t **bk_video_buffer_read_frame**(uint8_t *buf, uint32_t buf_len, int *err_code, uint32_t timeout)
read video buffer frame

This API will malloc a data_buffer, and save video data to this buffer

Parameters

- **buf** – malloc buf pointer

- **buf_len** – buf length
- **err_code** –
  - 0: success
  - 1: param error
  - 2: buffer full
  - 3: frame data err
  - 4: timeout
  - 5: unknow err
- **timeout** – read frame data timeout

Returns

- 0: failed
- other: frame_length.

bk_err_t **bk_video_transfer_stop**(void)

read video buffer frame

This API will stop camera, but not free malloc memory for transfer

Returns

- kNoErr: succeed
- others: other errors.

bk_err_t **bk_video_transfer_start**(void)

read video buffer frame

This API will start camera after stopping camera

@attation 1. this function must called after bk_video_transfer_stop

Returns

- kNoErr: succeed
- others: other errors.

API Typedefs

Header File

- video_types.h

Structures

struct **video_config_t**

Public Members

uint8_t *rxbuf
the buffer save camera data

void (*node_full_handler)(void *curptr, uint32_t newlen, uint32_t is_eof, uint32_t frame_len)
node full handler

This is a transfer camera data to uplayer api, when transfer node_len jpeg data finish , this function will be called

Param curptr the start address of transfer data.

Param newlen the transfer data length

Param is_eof 0/1: whether this packet data is the last packet of this frame, will called in jpeg_end_frame isr

Param frame_len the complete jpeg frame size, if is_eof=1, the frame_len is the true value of jpeg frame size, is_eof=0, the frame_len=0, in other words, only when transfer really frame_len at the last packet in jpeg_end_frame isr

void (*data_end_handler)(void)
brief data_end_handler

This api use to inforamte video transfer thread to deal transfer camera data

media_camera_device_t *device
config of camera

uint16_t rxbuf_len
The length of receiving camera data buff

uint16_t rx_read_len
manage the node_full_handler callback function input params

uint32_t node_len
video transfer network comunication protocol length a time

struct video_packet_t

Public Members

uint32_t ptklen
The current packet length

uint32_t frame_id
The current packet frame id

uint32_t is_eof
The current packet is the last packet

uint32_t frame_len
The frame length

struct **video_setup_t**

Public Members

media_camera_device_t ***device**
config of camera

uint16_t **open_type**
video transfer network comunication protocol type, video_open_type_t

uint16_t **send_type**
video transfer network comunication protocol type, video_send_type_t

uint16_t **pkt_header_size**
packet header size

uint16_t **pkt_size**
packet size

video_transfer_send_func **send_func**
function ptr for send to uplayer

video_transfer_start_cb **start_cb**
function ptr for start to send to uplayer

video_transfer_start_cb **end_cb**
function ptr for end to send to uplayer

tvideo_add_pkt_header **add_pkt_header**
function ptr for add packet header

struct **video_header_t**

Public Members

uint8_t **id**
the frame id

uint8_t **is_eof**
the flag of end frame, 1 for end

uint8_t **pkt_cnt**
the packet count of one frame

uint8_t **pkt_seq**
the packet header's count of one frame

struct **video_buff_t**

Public Members

beken_semaphore_t **aready_semaphore**
: the video data receive complete

uint8_t ***buf_base**
: the receive video data, malloc by user

uint32_t **buf_len**
: video buff length, malloc by user

uint32_t **frame_id**
: frame id

uint32_t **frame_pkt_cnt**
: the packet count of one frame

uint8_t ***buf_ptr**
: recoder the buff ptr of every time receive video packte

uint32_t **frame_len**
: the length of receive one frame

uint32_t **start_buf**
: video buff receive state

Macros

MEDIA_UDP_TRAN_LEN

MEDIA_TCP_TRAN_LEN

MEDIA_NET_TRAN_MAX_LEN

MEDIA_RETRY_DELAY_TIME

MEDIA_TRAN_DELAY_TIME_MS

Type Definitions

typedef void (*tvideo_add_pkt_header)(*video_packet_t* *param)

typedef int (*video_transfer_send_func)(uint8_t *data, uint32_t len)

typedef void (*video_transfer_start_cb)(void)

typedef void (*video_transfer_end_cb)(void)

Enumerations

enum video_open_type_t
video sample module protocol type

Values:

enumerator TVIDEO_OPEN_NONE
not sample module

enumerator TVIDEO_OPEN_SCCB
sample module follow sccb protocol

enumerator TVIDEO_OPEN_SPIDMA
sample module follow spidma protocol

enumerator TVIDEO_OPEN_RTSP
sample module follow rtsp protocol

enum video_send_type_t
video transfer network comunication protocol type

Values:

enumerator TVIDEO_SND_NONE
not transfer

enumerator TVIDEO_SND_UDP
follow udp protocol

enumerator TVIDEO_SND_TCP
follow tcp protocol

enumerator TVIDEO_SND_INTF
transfer to inter frame

enumerator TVIDEO_SND_BUFFER
transfer to buffer

enum **video_buff_state_t**
Values:

enumerator **BUF_STA_INIT**
video buff init

enumerator **BUF_STA_COPY**
video buff begin copy

enumerator **BUF_STA_GET**
video frame get

enumerator **BUF_STA_FULL**
video buff full

enumerator **BUF_STA_DEINIT**
video buff deinit

enumerator **BUF_STA_ERR**
other error

3.2.10 JPEGENC APIs

Important: The JPEGENC API v1.0 is the lastest stable JPEGENC APIs. All new applications should use JPEGENC API v1.0.

JPEGENC Interface

The JPEGENC module can work two different mode

- 0: JPEG ENCODER mode, output jpeg data
- 1: YUV mode, output camera sample data, yuv data

JPEGENC API Categories

Most of JPEGENC APIs can be categoried as:

- JPEGENC APIs

  The common APIs are prefixed with bk_jpeg_enc, e.g. bk_jpeg_enc_driver_init() etc.

JPEGENC APIs:

- *bk_jpeg_enc_driver_init()* - init jpeg encode module driver
- *bk_jpeg_enc_driver_deinit()* - deinit jpeg encode module driver
- *bk_jpeg_enc_init()* - init jpeg encode module
- *bk_jpeg_enc_deinit()* - deinit jpeg encode module
- *bk_jpeg_enc_get_frame_size()* - get a frame size output from jpeg encode module

API Reference

Header File

- jpeg_enc.h

Functions

bk_err_t **bk_jpeg_enc_driver_init**(void)
Init the jpeg encode driver.

This API init the resoure common:

- Init jpegenc driver control memory
- register jpegenc isr

Attention 1. This API should be called before any other jpeg APIs.

Attention 2. This API will power up video when soc is bk7256XX

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_driver_deinit**(void)
Deinit the jpegenc driver.

This API free all resource related to jpegenc and disable jpegenc.

Attention 1. This API will power down video when soc is bk7256XX

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_init**(const jpeg_config_t *config)
Init the jpeg encode.

This API init the jpegenc

- Configure clock and Power up the jpegenc
- Enable Jpegenc interrupt
- Map the jpegenc to dedicated GPIO port(MCLK and PCLK)
- Set the jpegenc parameters
- Enable jpeg encode module

Usage example:

```
jpeg_config_t init_config = {
                    .rx_buf = xxx, // the address point to mem for receiving␣
↪jpeg data
                    .yuv_mode = 0,
                    .sys_clk_div = 3, // jpeg_clk = 480 / (1+3) =120MHz
```

(continues on next page)

(continued from previous page)

```
                    .mclk_div = 0, // MCLK=jpeg_clk/4 = 30MHz
                    .rx_buf_len = xxx, // the length of receive jpeg data
                    .node_len = xxx, // dma transfer length
                    .x_pixel = 80, // jpeg image resolution of width: 80 * 8 =␣
↪640
                    .y_pixel = 60, // jpeg image resolution of height: 60 * 8 =␣
↪480
                    .dma_rx_finish_handler = cb, // dma transfer finish callback
                };
BK_LOG_ON_ERR(bk_jpeg_init(&init_config);
```

Attention 1. only work for imaging transfer

Parameters `config` – jpeg encode parameter settings

Returns

- BK_OK: succeed
- BK_ERR_NULL_PARAM: jpegenc config paramter is NULL
- BK_ERR_JPEG_NOT_INIT: jpegenc driver not init
- others: other errors.

bk_err_t **bk_jpeg_enc_deinit**(void)

Deinit the jpeg.

This API stop jpeg encode, close jpeg gpio, power off jpeg module at last.

Attention 1. only work for imaging transfer

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_start**(yuv_mode_t mode)

set jpegenc start

This API will set jpegenc work start.

Attention 1. if work in jpegenc mode, this api can enable jpegenc mode

Attention 2. if work in yuv mode, this api can enable yuv_mode.

Parameters `mode` – 0/1:jpeg encode mode/yuv mode

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_stop**(yuv_mode_t mode)

set jpegenc stop

This API will set jpegenc work stop.

Attention 1. if work in jpegenc mode, this api can disable jpegenc mode

Attention 2. if work in yuv mode, this api can disable yuv_mode.

Parameters `mode` – 0/1:jpeg encode mode/yuv mode

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_yuv_fmt_sel**(yuv_format_t fmt)

yuv format select

This API will set yuv output format.

Attention 1. this function only used when jpeg at yuv mode

Parameters `value` – 0:yuyv, 1:uyvy, 2:yyuv, 3:uvyy

Returns

- BK_OK: succeed
- others: other errors.

uint32_t **bk_jpeg_enc_get_frame_size**(void)

get a frame size output from jpegenc module, uint byte

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_register_isr**(jpeg_isr_type_t type_id, *jpeg_isr_t* isr, void *param)

register frame end isr

This API will register jpeg encode isr function, need user defined.The isr function will execute when the isr be triggered.

Parameters

- `type_id` – the type of the isr
  - JPEG_EOY: (eoy)when jpeg encode work in yuv mode, once a complete frame have been transfer by jpeg encode module, the isr will trigger.
  - JPEG_HEAD_OUTPUT: jpeg encode output head the isrt will be triggered, work in jpeg encode mode.
  - START_OFFRAME: (sof) when jpeg encode module start reveice a frame the isr will be triggered.
  - JPEG_EOF: (eof) when jpeg encode module receive a complete frame the isr will be triggered, this only effect in jpeg encode mode.
  - JPEG_VSYNC_NEGEDGE: (vng) when jpeg encode module receive a vsync negedge the isr will be triggered.
- `isr` – isr_func
- `param` – other value(default set NULL)

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_unregister_isr**(jpeg_isr_type_t type_id)
unregister frame end isr

This API will unregister jpeg encode isr function, need user unregister, when deinit jpeg module

Parameters **type_id** – the type of the isr

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_get_fifo_addr**(uint32_t *fifo_addr)
get jpeg enc fifo addr

This API will get jpeg encode fifo addr in param

Attention 1. only work in jpeg encode mode is effective

Parameters **fifo_addr** – fifo addr

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_set_auxs**(uint32_t cksel, uint32_t ckdiv)
set mclk output

This API will use for camera config

@attenation 1. this api only used for bk7256xx_mp chip, and when use this api, this api 'bk_jpeg_enc_mclk_enable' will not useful

Parameters

- **cksel** – 0-3:0:DCO/1:APLL/2:320M/3:480M,use 2 or 3
- **ckdiv** – 0-15, div = 1/(ckdiv+1)

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_partial_display_init**(const jpeg_partial_offset_config_t *offset_config)
partial display init

This API will use for partial display configure of jpeg encode

Parameters **offset_config** – x_pixel left/right offset and y_pixel left/right offset configure

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_partial_display_deinit**(void)
partial display deinit

This API will use for partial display deinit

Returns

- BK_OK: succeed

- others: other errors.

bk_err_t **bk_jpeg_set_em_base_addr**(uint8_t *address)
jpeg em base set

This API will use for jpeg em base address, 64KB memory align.

Parameters **address** – set a mem used for jpeg encode

Returns

- BK_OK: succeed
- others: other errors.

uint32_t **bk_jpeg_get_em_base_addr**(void)
jpeg em base get

This API will use for get jpeg em base address.

Returns

- jpeg em base

bk_err_t **bk_jpeg_enc_encode_config**(uint8_t enable, uint16_t up_size, uint16_t low_size)
jpeg encode enable encode auto contrl

This API will use for enable/disable for auto encode size. only valid in jpeh encode mode

@attenation 1. this api only called in jpeg_eof_isr function, and only bk_jpeg_enc_enable_encode_auto_ctrl have been set enable

Parameters

- **up_size** – the jpeg image upper limit, unit byte
- **low_size** – the jpeg image lower limit, unit byte

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_mode_switch**(jpeg_config_t *config)
jpeg encode mode and yuv mode switch

This API will use for switch jpeg mode

@attenation 1. this api must called in jpeg_eof isr or yuv_eof isr

Parameters **config** – jpeg encode config or jpeg yuv config

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_enc_set_mclk_div**(mclk_div_t div)

bk_err_t **bk_jpeg_enc_soft_reset**(void)
jpeg encode module softreset

This API will use for resetting jpeg encode module

@attenation 1. this api only used in bk7236xx

Returns

- BK_OK: succeed
- others: other errors.

API Typedefs

Header File

- jpeg_enc_types.h

Macros

Y_PIXEL_272
JPEGENC APIs Version 1.0.

image resolution for hight: Y * 8 = 272

Y_PIXEL_320
image resolution for hight: Y * 8 = 320

X_PIXEL_480
image resolution for width: X * 8 = 480

Y_PIXEL_480
image resolution for hight: Y * 8 = 480

X_PIXEL_640
image resolution for hight: X * 8 = 640

Y_PIXEL_240
image resolution for hight: Y * 8 = 240

X_PIXEL_320
image resolution for hight: X * 8 = 320

Y_PIXEL_600
image resolution for hight: Y * 8 = 600

X_PIXEL_800
image resolution for hight: X * 8 = 800

Y_PIXEL_720
image resolution for hight: Y * 8 = 720

X_PIXEL_1280
image resolution for hight: X * 8 = 1280

X_PIXEL_1600
image resolution for hight: X * 8 = 1600

Y_PIXEL_1200
image resolution for hight: Y * 8 = 1200

Type Definitions

typedef void (*`jpeg_isr_t`)(jpeg_unit_t id, void *param)
jpeg isr callback relay jpeg driver

Param id only 0 useful, set 0 default

Param param NULL default

3.2.11 Hardware JPEGDEC APIs

JPEGDEC API Categories

JPEGDEC APIs:

- *bk_jpeg_dec_driver_init()* - init jpeg decode hardware module
- *bk_jpeg_dec_driver_deinit()* - deinit jpeg decode hardware module
- *bk_jpeg_dec_hw_start()* - start jpeg decode
- *bk_jpeg_dec_stop()* - stop jpeg decode
- *bk_jpeg_dec_isr_register()* - register jpeg dec complete callback

HW JPEG DECODE Config

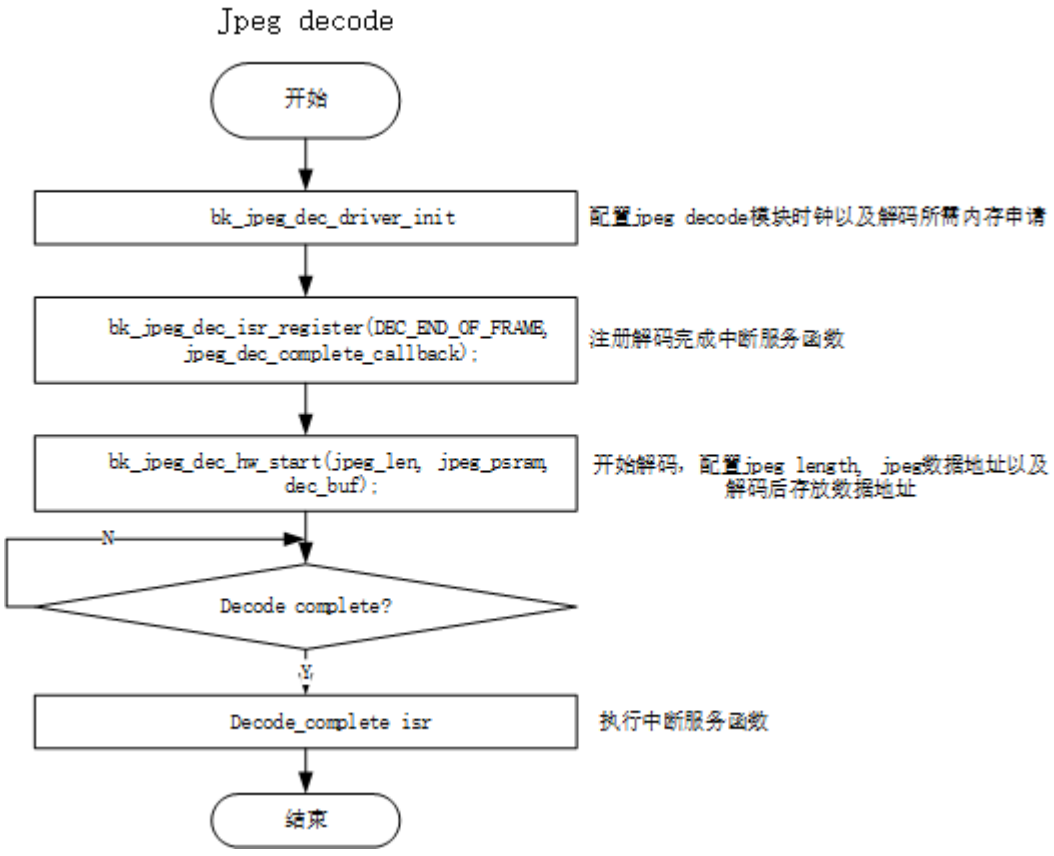


Fig. 9: Figure 1. jpeg decode config flow

API Reference

Header File

- jpeg_dec.h

Functions

bk_err_t **bk_jpeg_dec_driver_init**(void)
HW JPEG DEC API.

this api open the jpeg decode clk

bk_err_t **bk_jpeg_dec_driver_deinit**(void)
this api close the jpeg decode clk

- disable jpeg decode

bk_err_t **bk_jpeg_dec_line_int_dis**(void)
this api used for jpeg decode config

this api used for jpeg decode line interrupt config decode can select a frame complete done isr(defult) , or this decode lin_num enter int isr

Usage example:

```
if x pixel 320, you set bk_jpeg_dec_line_int_en(32), every 32 line will enter␣
↪isr, in all ten times
if x pixel 240, you set bk_jpeg_dec_line_int_en(24), every 24 line will enter␣
↪isr, in all ten times
```

this api used for disable jpeg decode line num int

Parameters

- **xpixel** – jpeg decode picture x pixel
  - input_buf decode input data addr
    - output_buf decode output data addr
- **lin_num** – decode line num ,should multiple of 8,

Returns

- BK_OK: succeed
- others: other errors.

Returns

Returns

- BK_OK: succeed
- others: other errors.

Returns

Returns

- BK_OK: succeed
- others: other errors.

bk_err_t **bk_jpeg_dec_hw_start**(uint32_t length, unsigned char *input_buf, unsigned char *output_buf)
this api start jpeg decode

Parameters **length** – jpeg length

- input_buf jpeg data src addr
- output_buf jpeg decode output addr

Returns 0: jpeg decode ok; others: error

bk_err_t **bk_jpeg_dec_dma_start**(uint32_t length, unsigned char *input_buf, unsigned char *output_buf)
this api start jpeg decode

Parameters **length** – jpeg length

- input_buf jpeg data src addr
- output_buf jpeg decode output addr

Returns 0: jpeg decode ok; others: error

bk_err_t **bk_jpeg_dec_stop**(void)
stop jpeg decode

Returns 0: jpeg decode stop ok; others: error

bk_err_t **bk_jpeg_dec_out_format**(pixel_format_t pixel_fmt)
set jpeg decode out format, useful in bk7258

Parameters **pixel_fmt** – select, PIXEL_FMT_YUYV(default), PIXEL_FMT_VUYY, PIXEL_FMT_VYUY (not commonly used),PIXEL_FMT_YYUV(not commonly used)

Returns BK_OK

void **bk_jpeg_dec_line_num_set**(*line_num_t* line_num)
when set decode by line, use this api config decode by how many lines

Parameters **select** – according to line_num_t enum,like 8/16/24/32….etc. multiple of 8 line

void **bk_jpeg_dec_by_line_start**(void)
start decode in every line decode isr

bk_err_t **jpeg_dec_set_dst_addr**(unsigned char *output_buf)
set decode output addr,this api is changed by line decode mode isr

bool **bk_jpeg_dec_by_line_is_last_line_complete**(void)
check decode by line is decode last line complete

Returns return true: last line complete

bk_err_t **bk_jpeg_dec_isr_register**(jpeg_dec_isr_t jpeg_dec_isr)
This API is direct registerjpeg dec cpu isr.

Parameters **isr** – isr function

Returns

- BK_OK: succeed
- others: other errors.

API Typedefs

Header File

- jpeg_dec_types.h

Structures

struct **jpeg_dec_res_t**
jpeg dec isr return value

Public Members

uint8_t **ok**
jpeg decoder success

uint16_t **pixel_x**
jpeg x pixel

uint16_t **pixel_y**
jpeg y pixel

uint32_t **size**
jpeg size

jpeg_encode_mode_t **jpeg_enc_mode**
jpeg encode mode

Macros

USE_JPEG_DEC_COMPLETE_CALLBACKS
set 1, register jpeg decode complete callback, set 0, register jpeg dec cpu isr

Type Definitions

typedef enum *_line_num* **line_num_t**

typedef void (***jpeg_dec_isr_cb_t**)(*jpeg_dec_res_t* *result)
refistrer type jpeg decode complete isr function

param return jpeg x pixel , y pixel , and jpeg size

Return

- BK_OK: succeed
- others: other errors. jpegdec int isr register func type

Enumerations

enum **jpeg_encode_mode_t**
Values:

enumerator **JPEG_ENCODE_UNSTD**

enumerator **JPEG_ENCODE_YUV420**

enumerator **JPEG_ENCODE_YUV422**

enumerator **JPEG_ENCODE_YUV444**

enum **JRESULT**
Error code

Values:

enumerator **JDR_OK**
0: Succeeded

enumerator **JDR_INTR**
1: Interrupted by output function

enumerator **JDR_INP**
2: Device error or wrong termination of input stream

enumerator **JDR_MEM1**
3: Insufficient memory pool for the image

enumerator **JDR_MEM2**
4: Insufficient stream input buffer

enumerator **JDR_PAR**
5: Parameter error

enumerator **JDR_FMT1**
6: Data format error (may be damaged data)

enumerator **JDR_FMT2**
7: Right format but not supported

enumerator **JDR_FMT3**
8: Not supported JPEG standard

enum **_line_num**
Values:

enumerator **LINE_8**

enumerator **LINE_16**

enumerator **LINE_24**

enumerator **LINE_32**

enumerator **LINE_40**

enumerator **LINE_48**

enum **jpeg_dec_isr_type_t**
Values:

enumerator **DEC_END_OF_FRAME**
select jpeg decode a complete frame enter isr callback

enumerator **DEC_EVERY_LINE_INT**
select jpeg decode by line num enter isr callback

enumerator **DEC_ERR**

enumerator **DEC_ISR_MAX**

3.2.12 JPEGDEC APIs

Important: The JPEGDEC API v1.0 is the lastest stable JPEGDEC APIs. All new applications should use JPEGDEC API v1.0. This component is software decode not hardware decode

JPEGDEC API Categories

Most of JPEGDEC APIs can be categoried as:

- JPEGDEC APIs

  The common APIs are prefixed with bk_jpeg_dec_sw, e.g. bk_jpeg_dec_sw_init() etc.

JPEGDEC APIs:

- *bk_jpeg_dec_sw_init()* - init jpeg decode software module
- *bk_jpeg_dec_sw_deinit()* - deinit jpeg decode software module
- *bk_jpeg_dec_sw_start()* - excute jpeg decode
- *bk_jpeg_dec_sw_register_finish_callback()* - register jpeg decode finish callback
- *bk_jpeg_dec_sw_start_one_time()* - excute jpeg decode without relying on initalization

API Reference

Header File

- jpeg_decode_sw.h

Functions

bk_err_t **bk_jpeg_dec_sw_init**(void *jd_in, uint32_t len)

JPEG_DEC APIs Version 1.0.

Init the jpeg_dec

This API init the jpeg_dec

- config jpeg_dec image resoult, and simple_rate
- malloc buffer for jpeg_dec

Attention 1. only for software jpeg_dec

Parameters

- **jd_in** – inter buffer
- **len** – buffer length

Returns

- BK_OK: succeed
- other: errors.

bk_err_t **bk_jpeg_dec_sw_deinit**(void)

Deinit the jpeg_dec.

This API deinit the jpeg_dec

- clear all configure for jpeg_dec
- free buffer for jpeg_dec

Attention 1. only for software jpeg_dec

Returns

- BK_OK: succeed
- other: errors.

bk_err_t **bk_jpeg_dec_sw_start**(uint8_t decode_type, uint8_t *src_buf, uint8_t *dst_buf, uint32_t jpeg_size, uint32_t outbuf_size, sw_jpeg_dec_res_t *result)

jpeg_decoder_fun

This API jpeg_decoder_fun

- jpeg_prepare for jpeg_dec
- start jpeg_dec

Attention 1. only for software jpeg_dec

Attention 2. outbuf_size only work for decode_type=0, decode by frame, and outbuf size must 2 times for decode 16line

Parameters

- **decode_type** – 0:decode by line, one time decode 16line; 1:decode by frame
- **src_buf** – jpeg image address
- **dst_buf** – decode output buffer address
- **outbuf_size** – decode output buffer size
- **result** – decode image info

Returns

- BK_OK: succeed
- other: errors.

bk_err_t **bk_jpeg_dec_sw_start_line**(void)
start line jpeg decode sw

This API is used to decode go on

Attention 1. only for software jpeg_dec by line

Returns

- BK_OK: succeed
- other: errors.

bk_err_t **bk_jpeg_get_img_info**(uint32_t frame_size, uint8_t *src_buf, sw_jpeg_dec_res_t *result)
jpeg_decoder_fun

This API is used to get img width and size

Attention 1. only for software jpeg_dec

Parameters

- **frame_size** – input image size
- **src_buf** – jpeg image address
- **result** – decode image info

Returns

- BK_OK: succeed
- other: errors.

void **bk_jpeg_dec_sw_register_finish_callback**(void *cb)
jpegdec register finish icallback_func

This API jpeg_decoder_fun, the callback function wiil execute when decode success

Attention 1. only for software jpeg_dec

Parameters **cb** – finish jpeg_dec callback function

Returns

bk_err_t **bk_jpeg_dec_sw_start_one_time**(uint8_t decode_type, uint8_t *src_buf, uint8_t *dst_buf, uint32_t jpeg_size, uint32_t outbuf_size, sw_jpeg_dec_res_t *result, uint8_t scale, JD_FORMAT_OUTPUT format, media_rotate_t rotate_angle, uint8_t *work_buffer, uint8_t *rotate_buffer)

jpeg_decoder_fun

This API jpeg_decoder_fun

- jpeg_prepare for jpeg_dec
- start jpeg_dec

Attention 1. only for software jpeg_dec

Attention 2. outbuf_size only work for decode_type=0, decode by frame, and outbuf size must 2 times for decode 16line

Parameters

- **decode_type** – 0:decode by line, one time decode 16line; 1:decode by frame
- **src_buf** – jpeg image address
- **dst_buf** – decode output buffer address
- **outbuf_size** – decode output buffer size
- **result** – decode image info
- **scaling** – of decode image
- **format** – of decode image
- **rotate_angle** – of decode image
- **work_buffer** – inter workbuffer
- **rotate_buffer** – rotate need buffer

Returns

- BK_OK: succeed
- other: errors.

API Typedefs

3.2.13 JPEGENC APIs

Important: This component is for software encode not hardware encode

JPEGENC API Categories

Most of JPEGENC APIs can be categoried as:

- JPEGENC APIs

JPEGENC APIs:

- *jpeg_sw_encoder_init()* - init jpeg encode software module

API Reference

Header File

- jpeg_enc_sw.h

Functions

void **jpeg_sw_encoder_init**(void)
Initialize the JPEG softeware encoder.

Attention 1. only for software jpeg encode

Returns

Structures

struct **_jpeg_sw_encoder**
JPEG_SW_ENC APIs Version 1.0.

Type Definitions

typedef struct *_jpeg_sw_encoder* **jpeg_sw_encoder_t**
JPEG_SW_ENC APIs Version 1.0.

API Typedefs

3.2.14 AEC APIs

Important: The AEC API v1.0 is the lastest stable AEC APIs. All new applications should use AEC API v1.0.

AEC API Categories

AEC APIs:

- *aec_size()* - get AECContext size (byte)
- *aec_init()* - init aec module
- *aec_ctrl()* - get and set aec parameters
- *aec_proc()* - excute aec function

API Reference

Header File

- aec.h

Functions

uint32_t **aec_size**(uint32_t delay)
Get AECContext size (byte)

Parameters **delay** – delay samples

Returns

- uint32_t: size(byte)

void **aec_init**(*AECContext* *aec, int16_t fs)
Init aec.

This API init aec function :

- Set aec parameters
- Init fft parameters

Usage example:

```
AECContext* aec = NULL;

//malloc aec memory
aec = (AECContext*)os_malloc(aec_size());

//init aec
aec_init(aec, 16000);
```

Parameters

- **aec** – aec parameters
- **fs** – frame sample 8K/16K

Returns none

void **aec_ctrl**(*AECContext* *aec, uint32_t cmd, uint32_t arg)
Control aec.

This API control aec function :

- Set aec parameters
- Get data addr

Usage example:

```
AECContext* aec = NULL;

//malloc aec memory
aec = (AECContext*)os_malloc(aec_size());

//init aec
aec_init(aec, 16000);

//set mic delay
aec_ctrl(aec, AEC_CTRL_CMD_SET_MIC_DELAY, 10);
```

Parameters

- **aec** – aec parameters
- **cmd** – control command in enum AEC_CTRL_CMD
- **arg** – value

Returns none

void **aec_proc**(*AECContext* *aec, int16_t *rin, int16_t *sin, int16_t *out)

Control aec.

This API control aec function :

- Set aec parameters
- Get data addr

Usage example:

```
AECContext* aec = NULL;

//malloc aec memory
aec = (AECContext*)os_malloc(aec_size());

//init aec
aec_init(aec, 16000);

//aec excute work
aec_proc (aec, rin, sin, out);
```

Parameters

- **aec** – aec parameters
- **rin** – reference data
- **sin** – source data
- **out** – output data

Returns none

Type Definitions

typedef struct _AECContext **AECContext**

Enumerations

enum **AEC_CTRL_CMD**
Values:

enumerator **AEC_CTRL_CMD_NULL**

enumerator **AEC_CTRL_CMD_GET_MIC_DELAY**
Get mic delay

enumerator **AEC_CTRL_CMD_SET_MIC_DELAY**
Set mic delay

enumerator **AEC_CTRL_CMD_GET_EC_DEPTH**
Get ec depth

enumerator **AEC_CTRL_CMD_SET_EC_DEPTH**
Set ec depth

enumerator **AEC_CTRL_CMD_GET_NS_LEVEL**
Get ns level

enumerator **AEC_CTRL_CMD_SET_NS_LEVEL**
Set ns level

enumerator **AEC_CTRL_CMD_GET_DRC**
Get drc

enumerator **AEC_CTRL_CMD_SET_DRC**
Set drc

enumerator **AEC_CTRL_CMD_GET_FLAGS**
Get flags

enumerator **AEC_CTRL_CMD_SET_FLAGS**
Set flags

enumerator **AEC_CTRL_CMD_SET_VOL**
Set volum

enumerator **AEC_CTRL_CMD_SET_REF_SCALE**
Set reference scale

enumerator **AEC_CTRL_CMD_SET_TxRxThr**
Set TxRxThr

enumerator **AEC_CTRL_CMD_SET_TxRxFlr**
Set TxRxFlr

enumerator **AEC_CTRL_CMD_SET_ECRSD1**
Set ECRSD1

enumerator **AEC_CTRL_CMD_SET_ECRSD2**
Set ECRSD2

enumerator **AEC_CTRL_CMD_SET_RSDBAND**
Set RSDBAND

enumerator **AEC_CTRL_CMD_SET_NS_PARA**
Set ns parameter

enumerator **AEC_CTRL_CMD_SET_PARAMS**
Set parameters

enumerator **AEC_CTRL_CMD_SET_EC_COE**
Set EC_COE

enumerator **AEC_CTRL_CMD_SET_DRC_TAB**
Set DRC_TAB

enumerator **AEC_CTRL_CMD_GET_RX_BUF**
Get rx buffer addr

enumerator **AEC_CTRL_CMD_GET_TX_BUF**
Get tx buffer addr

enumerator **AEC_CTRL_CMD_GET_OUT_BUF**
Get output buffer addr

enumerator **AEC_CTRL_CMD_GET_FRAME_SAMPLE**
Get frame sample

Reference link

API Reference : Introduced the AEC API interface

User and Developer Guide : Introduced common usage scenarios of AEC

3.2.15 G711 APIs

Important: The G711 API v1.0 is the lastest stable G711 APIs. All new applications should use G711 API v1.0.

G711 API Categories

G711 APIs:

- *linear2alaw()* - G711A encodec pcm data to a-law data
- *alaw2linear()* - G711A decodec a-law data to pcm data
- *linear2ulaw()* - G711U encodec pcm data to u-law data
- *ulaw2linear()* - G711U decodec u-law data to pcm data

API Reference

Header File

- g711.h

Functions

unsigned char **linear2alaw**(int pcm_val)
G711A encodec pcm data to a-law data.

Parameters **pcm_val** – 16 bit pcm data

Returns

- unsigned char: a-law data after pcm data has been encoded by G711A

int **alaw2linear**(unsigned char a_val)
G711A decodec a-law data to pcm data.

Parameters **a_val** – 8 bit a-law data

Returns

- int: pcm data after a-law data has been decoded by G711A

unsigned char **linear2ulaw**(int pcm_val)
G711U encodec pcm data to u-law data.

Parameters **pcm_val** – 16 bit pcm data

Returns

- unsigned char: u-law data after pcm data has been encoded by G711U

int **ulaw2linear**(unsigned char u_val)
G711U decodec u-law data to pcm data.

Parameters **u_val** – 8 bit u-law data

Returns

- int: pcm data after u-law data has been decoded by G711U

3.2.16 Aud_Intf APIs

Important: The Aud_Intf API v1.0 is the lastest and stable Audio interface APIs. All new applications should use Audio interface API v1.0. And all Aud_Intf API cannot be called in mic and speaker callback registed in *bk_aud_intf_drv_init* .

Audio Interface

The BK Audio interface supports two work mode:

- General mode
- Voice mode

The general work mode supports general functions, and the Voice mode supports voice transfer functions. However, the audio only work in one mode at a time.

The General mode supports following functions:

- record function
- play function

The record and play function can be operated independently.

The Voice mode supports following features:

- voice transfer by G711A data
- voice transfer by PCM data
- 8k and 16k sample rate
- software AEC
- set mic gain
- set speaker gain
- onboard mic and speaker
- UAC mic and speaker

Audio Interface API Categories

Most of Audio APIs can be categoried as:

- Common APIs

  The common APIs are prefixed with bk_aud_intf, the APIs may be common for General mode and Voice mode, e.g. bk_aud_intf_drv_init() etc.

- Mic function APIs.

  The APIs provide support for record function. e.g. bk_aud_intf_mic_init() etc.

- Spk function APIs.

  The APIs provide support for play function. e.g. bk_aud_intf_spk_init() etc.

- Voc function APIs.

  The APIs provide support for voice transfer function. e.g. bk_aud_intf_voc_init() etc.

Common APIs:

- bk_aud_intf_drv_init() - init the audio interface driver
- bk_aud_intf_drv_deinit() - deinit the audio interface driver
- bk_aud_intf_set_mode() - set audio interface work mode
- bk_aud_intf_set_mic_gain() - set audio mic gain
- bk_aud_intf_set_spk_gain() - set audio speaker gain
- bk_aud_intf_write_spk_data() - write data to sperker

Mic APIs:

- bk_aud_intf_mic_init() - init the mic function
- bk_aud_intf_mic_deinit() - deinit the mic function
- bk_aud_intf_mic_start() - start mic work
- bk_aud_intf_mic_pause() - pause mic work
- bk_aud_intf_mic_stop() - stop mic work
- bk_aud_intf_set_mic_chl() - set mic channel in work
- bk_aud_intf_get_mic_chl() - get mic channel in work
- bk_aud_intf_set_mic_samp_rate() - set mic sample rate
- bk_aud_intf_get_mic_samp_rate() - get mic sample rate

Spk APIs:

- bk_aud_intf_spk_init() - init the speaker function
- bk_aud_intf_spk_deinit() - deinit the speaker function
- bk_aud_intf_spk_start() - start speaker work
- bk_aud_intf_spk_pause() - pause speaker work
- bk_aud_intf_spk_stop() - stop speaker work
- bk_aud_intf_set_spk_chl() - set speaker channel in work
- bk_aud_intf_get_spk_chl() - get speaker channel in work
- bk_aud_intf_set_spk_samp_rate() - set speaker sample rate
- bk_aud_intf_get_spk_samp_rate() - get speaker sample rate

Voc APIs:

- bk_aud_intf_voc_init() - init the voice transfer function
- bk_aud_intf_voc_deinit() - deinit the voice transfer function
- bk_aud_intf_voc_start() - start voice transfer
- bk_aud_intf_voc_stop() - stop voice transfer
- bk_aud_intf_set_aec_para() - set aec parameter
- bk_aud_intf_get_aec_para() - get aec parameter
- bk_aud_intf_voc_tx_debug() - register voice tx debug callback
- bk_aud_intf_voc_rx_debug() - register voice rx debug callback
- bk_aud_intf_voc_aec_debug() - register voice aec debug callback

API Reference

#.. include:: ../../_build/inc/aud_intf.inc

API Typedefs

#.. include:: ../../_build/inc/aud_intf_types.inc

CHAPTER

FOUR

SUPPORT PERIPHERALS

This following are the peripherals supported by software platform adjustment verification:

4.1 1.LCD

Reference file path: `.\components\bk_peripheral\src\lcd`

4.1.1 1.1.RGB LCD

Reference file path: `.\components\bk_peripheral\src\lcd\rgb`
The supported RGB LCD models and resolutions are as shown in the table below:

models	Resolutions	SPI Interface
ST7282	480*272	NO
ST7701S	480*480	YES
ST7701SN	480*800	YES
H050IWV	800*480	NO
GC9503V	480*800	YES
NT35510	480*854	YES
NT35512	480*800	YES
HX8282	1024*600	NO
NV3052CGRB	720*1280	YES

4.1.2 1.2.Intel 8080[MCU] LCD

Reference file path: `.\components\bk_peripheral\src\lcd\mcu`
The supported Intel 8080[MCU] LCD models and resolutions are as shown in the table below:

Models	Resolutions
ST7796S	320*480
NT35510	480*800
ST7789V	172*320

4.1.3 1.3.QSPI LCD

Reference file path: `.\components\bk_peripheral\src\lcd\qspi`
The supported QSPI LCD models and resolutions are as shown in the table below:

Models	Maximum Resolutions	Has RAM
SH8601A_FLSAMO139	454*454	YES
ST77903_WX20114	400*400	NO
ST77903_SAT61478M	360*800	NO
ST77903_H0165Y008T	400*400	NO
SPD2010_H0146Y005T	412*412	YES

4.2 2.DVP Camera

Reference file path: `.\components\bk_peripheral\src\dvp`
The supported DVP Camera models and maximum resolutions are as shown in the table below:

Models	Maximum Resolutions
GC0308	640*480
GC0328C	640*480
GC2145	1280*720
HM1055	1280*720
OV2640	1280*720
SC101	1280*720

4.3 3.UVC Camera

The supported UVC Camera models, resolutions and frame rates are as shown in the table below:

Models	Resolutions - Frame Rates
USBVID_0C45&PID_64AB&REV_0822	1280*720-15 800*480-15 640*480-15 480*320-15
USBVID_13D3&PID_78AB&REV_0826	640*480-15
USBVID_13D3&PID_78AB&REV_0829	1280*720-10 640*480-30 480*800-30 480*320-30 320*240-30
USBVID_0C45&PID_64AB&REV_0811	640*480-25
USBVID_1871&PID_FF50&REV_0100	640*480-15 480*320-15 352*288-15 320*240-15
USBVID_0C45&PID_64AB&REV_1202	640*480-25 480*320-30 320*240-30
USBVID_0C45&PID_64A0&REV_0100	800*480-15 640*480-15 480*320-15 352*288-15
USBVID_13D3&PID_784B&REV_1001	640*480-15
USBVID_05A3&PID_9230&REV_0909	1280*720-25 640*480-15 480*800-25
USBVID_0C45&PID_636B&REV_0100	800*480-25 640*320-25 480*320-25
USBVID_174F&PID_2459&REV_0008	1280*720-30 960*540-30

4.4 4.Touch Panel

Reference file path: `.\components\bk_peripheral\src\tp`

The supported Touch Panel models and resolutions are as shown in the table below:

Models	Resolutions
FT6336	Customizable —- ————- Customizable
GT991	
GT1151	Customizable
HY4633	Customizable

CHAPTER

FIVE

LCD

LCD screen is the abbreviation of Liquid Crystal Display. The structure of LCD is to place liquid crystals between two parallel pieces of glass. There are many vertical and horizontal small wires between the two pieces of glass. By controlling whether the rod-shaped crystal molecules are energized or not, the direction is changed, and the light is refracted to create a picture. Liquid crystal is an organic compound composed of long rod-shaped molecules. In their natural state, the long axes of these rod-like molecules are roughly parallel.

5.1 LCD physical structure

LCD screen panels are mainly composed of glass substrates, polarizing plates, color filters, liquid crystal materials, alignmentLayer, optical films, drive circuits and other components.

1.glass substrates A glass substrate is actually a thin piece of glass with a flat surface. There is a transparent conductive layer of In2O3 or SnO2 evaporated on the surface, which is the ITO film layer. After photolithography and processing, a transparent conductive pattern is made. These graphics are composed of the smallest image unit, pixel graphics and outer lead graphics of all the chromaticity and brightness of an image. Therefore, the outer leads must not be soldered by traditional soldering. Connections can only be made via conductive rubber strips or conductive tape, etc. If it is scratched, cut or corroded, the device will be scrapped.

2.color filters The reason why the LCD panel can display colors is because the light passes through the color filter. Then the liquid crystal panel is changed by the voltage of the driving chip, so that the liquid crystal molecules stand in a row or appear in a twisted state. After forming the gate, you then choose whether the light from the backlight source penetrates or not, and finally the picture is produced. But this is only a difference in the degree of light transmission, and the colors produced are only black and white. If you want to form a colorful picture, It needs to rely on a combination of three light sources: red, green, and blue.

3.alignment film Alignment film is the most critical material to control the quality of LCD display. In order to achieve a good rotation effect of the liquid crystal material, it is necessary to apply an alignment film on the inside of the upper and lower electrode substrates of the LCD display screen. After the alignment film is coated, the friction process will be carried out. The surface of the alignment film will form a groove arranged in a certain direction due to friction. The liquid crystal material on the alignment film will also reach the desired height due to the force between molecules. A directional effect that produces alignment. In this way, we can control the predetermined direction and predetermined tilt angle arrangement of the liquid crystal molecules, which is very conducive to the movement of the LCD display.

4.liquid crystal material Liquid crystal material is the main material of LCD display screen. Most liquid crystal materials are mixed from several or even more than a dozen types of single liquid crystal materials. Each liquid crystal material has its own fixed clearing point TL and crystallization point TS. Therefore, each type of LCD display must be used or stored within a certain temperature range between TS and TL. If the temperature is too low, crystallization will destroy the orientation layer of the LCD display; and If the temperature is too high, The liquid crystal will lose its liquid crystal state, and then it will lose all the functions of the LCD display.

5.polarizer The main purpose of polarizers is to polarize light that passes through the dichroic medium of the polarizing film.

6.drive circuit The biggest function of the drive circuit is to build on the drive circuit by adjusting a series of parameters such as voltage, phase, peak value, frequency, timing, effective value, duty cycle, etc. applied to the pixel electrode.

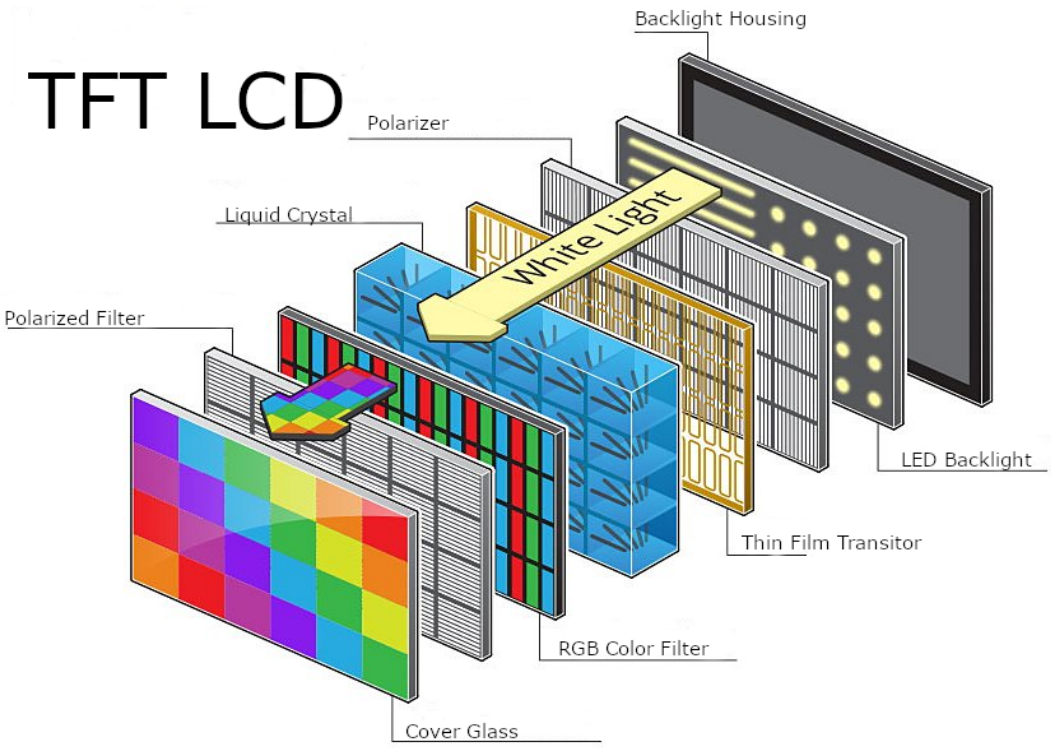


Fig. 1: Figure 1. lcd

5.2 LCD driver interface

The current mainstream LCD driver interfaces include SPI, I8080, QSPI, RGB, etc.

5.3 8080 LCD hardware interface

The data transmission of the 8080 interface is 8-bit, 9-bit, 16-bit and 18-bit. The interface bus introduced in this document is 8-bit. The hardware schematic diagram is as follows:

- 3.5”(diagonal), 320x3 RGB x 480 dots, 262K colors TFT LCD module
- Driving IC: ST7796S
- RGB serial interface

The chip IO resources occupied by the Display 8080 interface are as follows:

类型	描述	占用 IO 数量	并行数据位数	数据传输带宽	GRAM 位置
SPI	串行接口，以 SPI 总线协议为基础，通常采用 4 线或 3 线模式	最少	1	最小	LCD
QSPI（Quad-SPI）	SPI 接口的一种扩展，可以使用 4 根数据线并行传输	较少	4	较小	LCD 或主控
I80（MCU、DBI）	并行接口，以 I80 总线协议为基础	较多	8/16	较大	LCD
RGB（DPI）	并行接口，一般需搭配 3-wire SPI 接口	最多	8/16/18/24	较大	主控

Fig. 2: Figure 2. driver interface

5.4 8080 LCD interface driving principle

The data transmission of the I8080 protocol is carried out in the format of cmd+parameter. Through the method of cmd + param, the initialization of the screen, the configuration of module functions, and the transmission of image data are realized. The following is a timing diagram for writing commands and data:

- RESET is set to 1 when transmitting
- When sending DB data, CSX is set low and WRX is pulled low.
- If it is COMMAND data, DCX is set low; if it is DATA data, DCX is set high.
- During write data transfer, the DB line needs to remain stable on the rising edge of WR.

8080 hardware implementation of LCD is as follows:

5.5 Tearing Effect

The Tearing Effect output line supplies to the MPU a Panel synchronization signal.The signal can be used by the MPU to synchronize Frame Memory Writing when displaying video images. if the lcd not have TE single, the controller should select adapt clk to reduce tearing

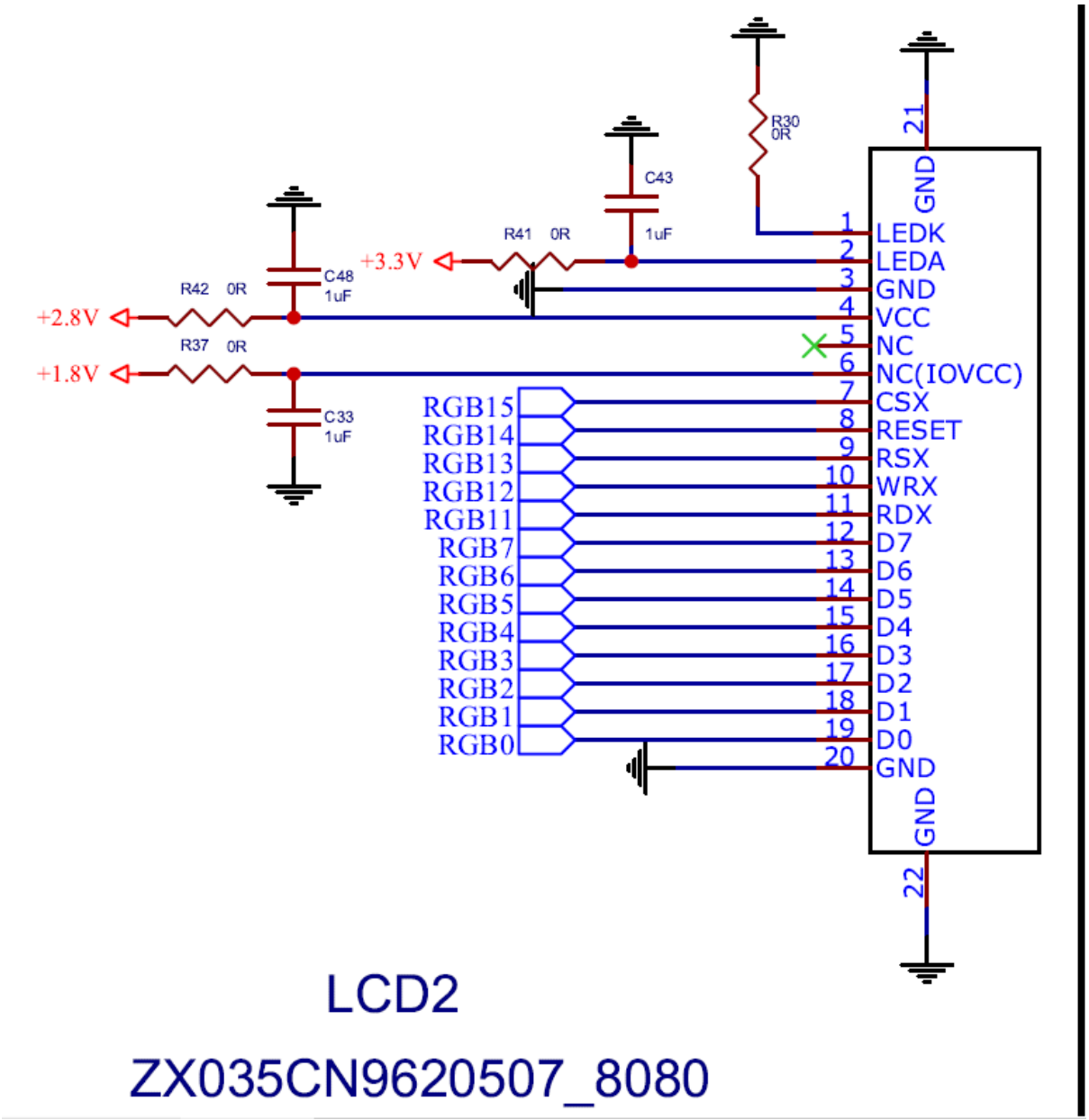


Fig. 3: Figure 3. 8080 lcd pin

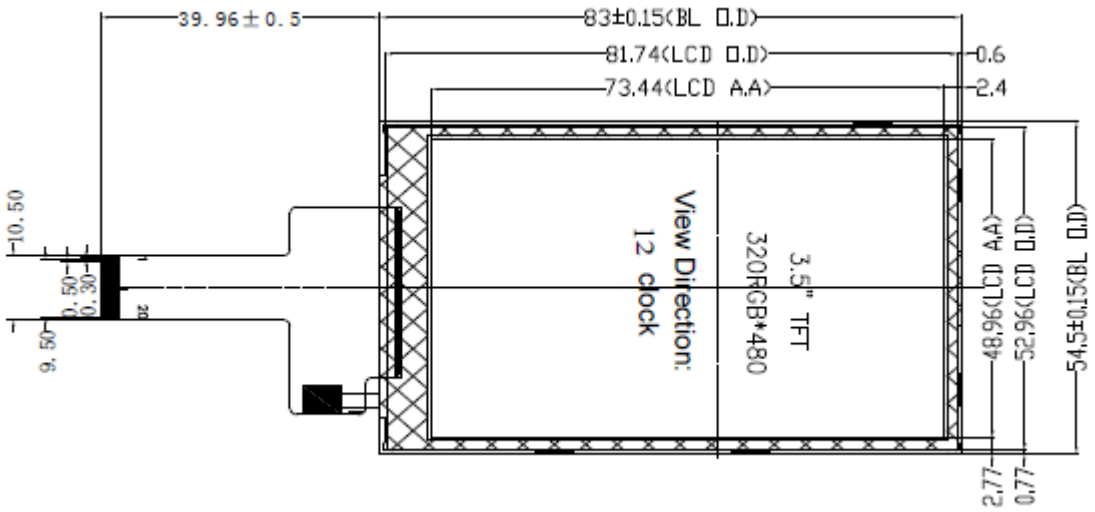


Fig. 4: Figure 4. 8080 lcd floor plan

GPIO	IO Function 5	Describtion
GPIO 19	CSX	Chip select signal, Low-active.
GPIO 20	RESET	屏幕的reset信号，在启动和初始化时需要置低。
GPIO 21	DCX	Display data/command selection pin in MCU interface DCX='1': display data or parameter. DCX='0': register index / command.
GPIO 22	WRX	Write enable in MCU parallel interface. 数据在上升沿保持稳定。
GPIO 23	RDX	Read enable in 8080 MCU parallel interface. Low-active
GPIO47 ~GPIO40	DB7-DB0	8-bit I/F: DB[0] ~ DB[7] 数据输出 IO

Fig. 5: Figure 5. 8080 lcd pin function

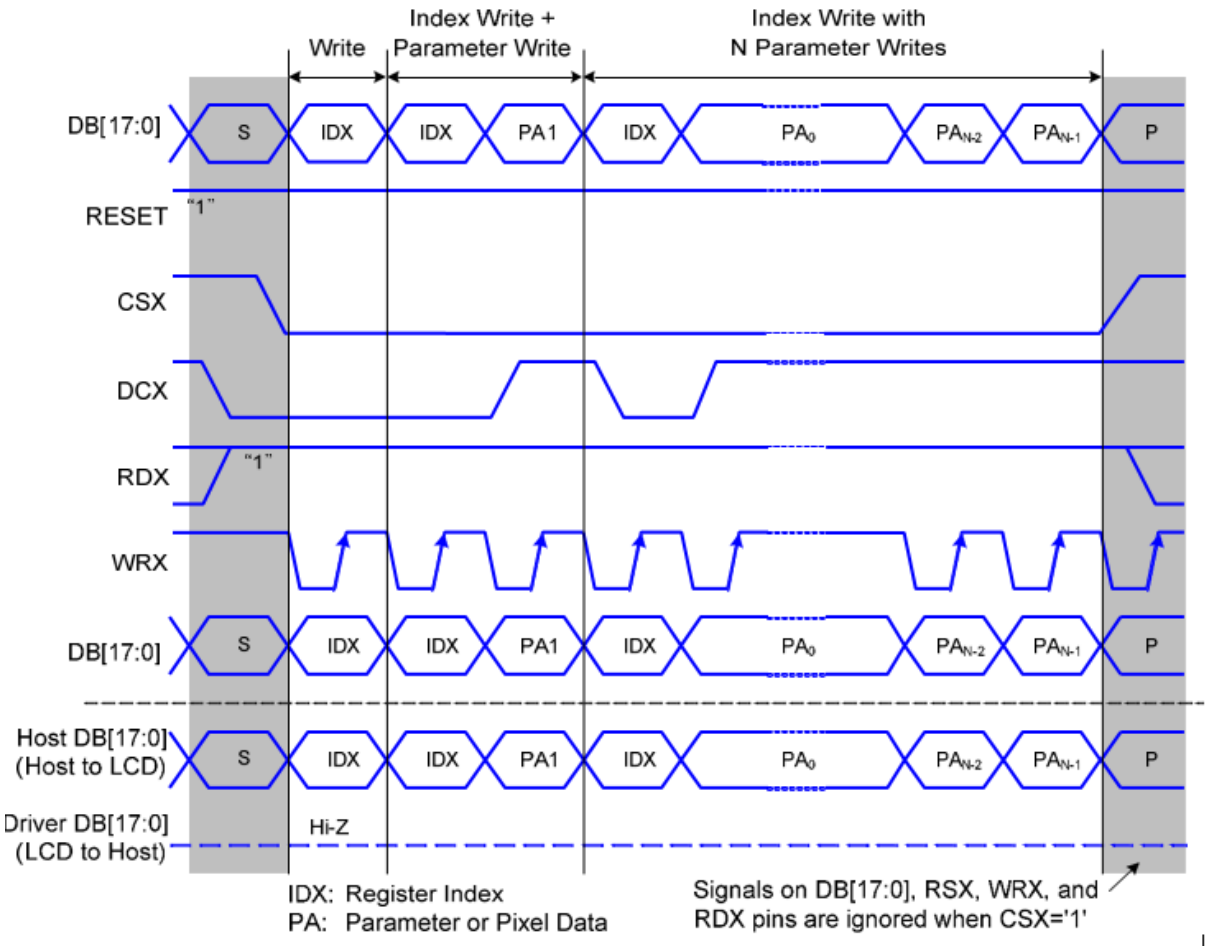


Fig. 6: Figure 6. 8080 lcd timing

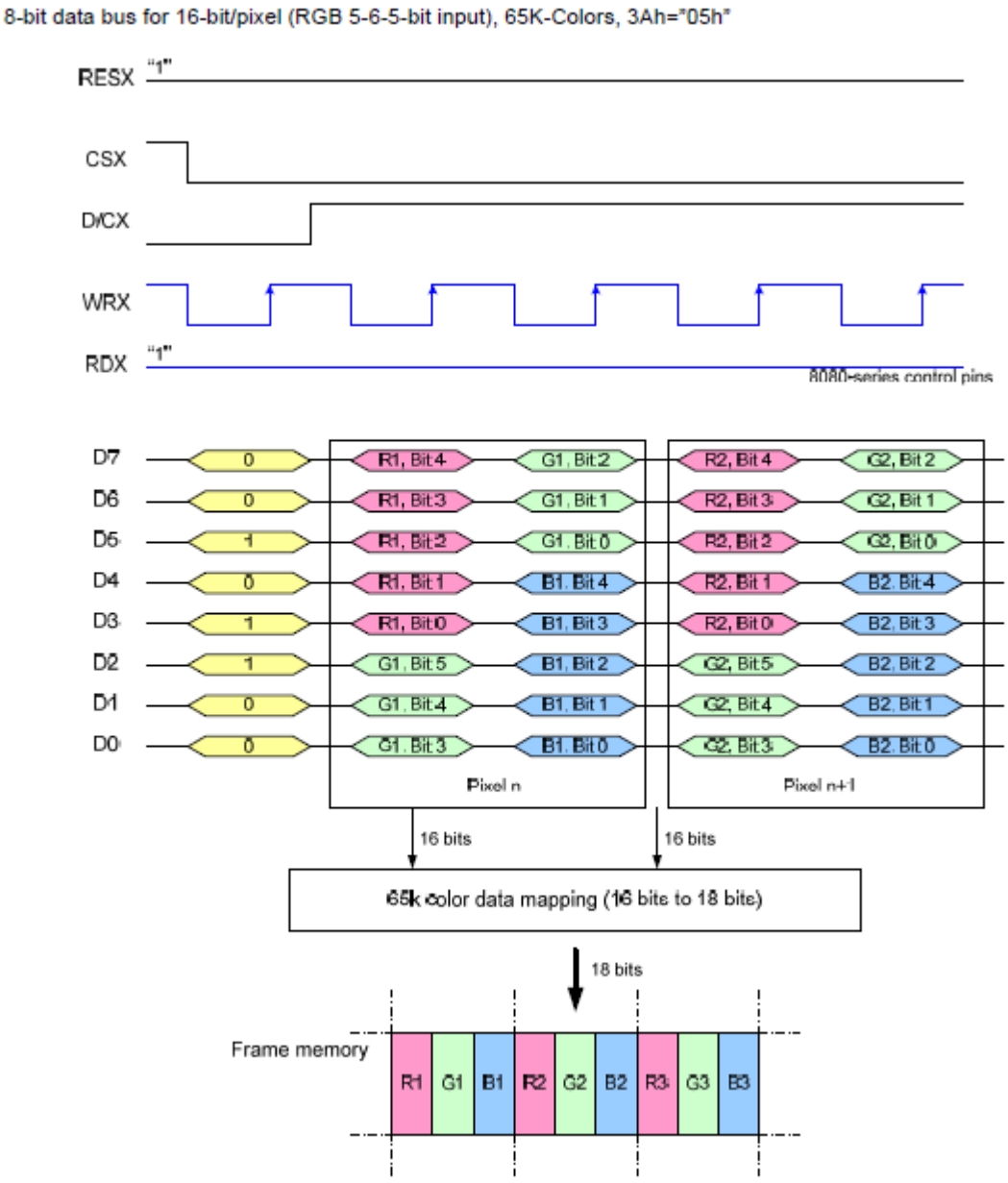


Fig. 7: Figure 7. 8080 lcd data transmission

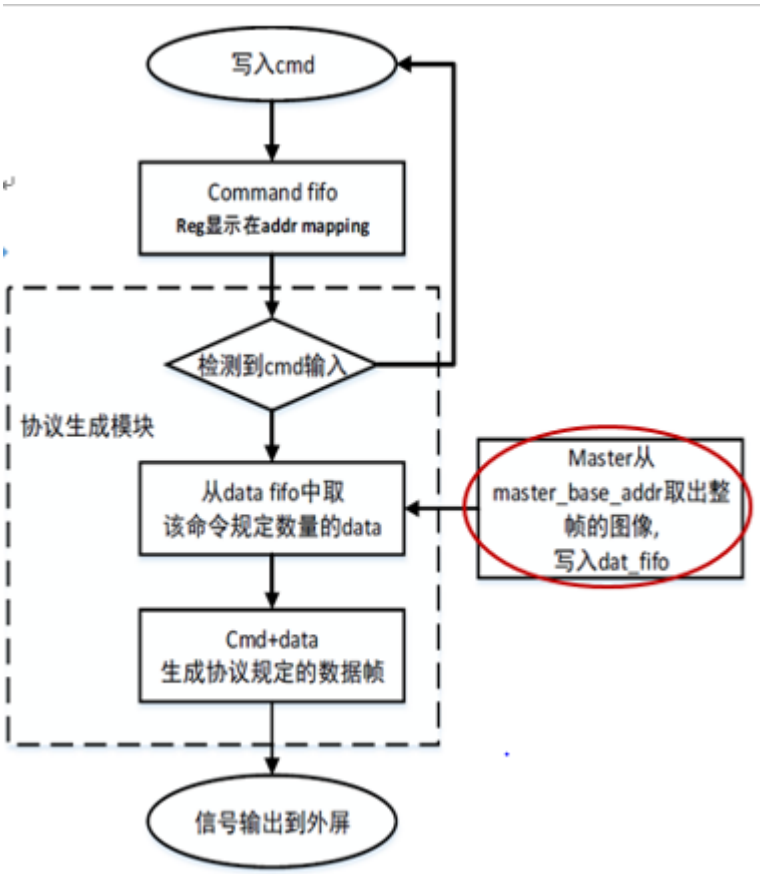


Fig. 8: Figure 8. 8080 lcd hardware process

5.6 RGB LCD hardware interface

The RGB interface, also known as the DPI (Display Pixel Iterface) interface, is a parallel interface that uses ordinary synchronous clock and signal lines to transmit data. The data lines and control lines of its interface are separated. Because there is no GRAM inside the screen, the protocol data speed is fast and the cost is low. The screen can be refreshed directly. It is usually used for driving large screens. The data types of the RGB protocol include RGB565, RGB88, RGB666, etc. The color components are red, green, and blue. By changing the three color channels, the colors are superimposed on each other to obtain a variety of colors. This module uses RGB565 data type.

- Resolution : 480(H) x 3(RGB) x 272(V) pixels
- Input Data: Parallel RGB565 16-bit
- Driver IC: ST7282

The chip IO resources occupied by the Display rgb interface are as follows:

5.7 RGB LCD interface driving principle

The RGB LCD protocol timing diagram is as follows:

- DCLK pixel clock signal: output to the driving clock of the external screen to ensure the correctness of data transmission, and read RGB data on the falling edge (or rising edge) of the clock
- VSYNC indicates the beginning of scanning a frame. During the data transmission of a frame, VSYNC will be set to 1 until the transmission of a frame is completed.
- HSYNC represents the beginning of scanning a line, will be set to 1 before each line of data transmission, and will be set to 0 at the end of each line of data transmission.

The RGB protocol data format is as follows:

Hardware implementation of RGB LCD:

5.8 QSPI LCD hardware interface

- Resolution: 454(W) x RGB x 454(H)
- Driver IC: SH8601A
- Interface: QSPI
- Display mode: MOLED

The chip IO resources occupied by the Display qspi interface are as follows:

5.9 QSPI LCD interface driving principle

The QSPI LCD protocol timing diagram is as follows:

The qspi driver interface is similar to spi. The difference lies in the number of data pins. qspi has two more data transmission pins and is faster than spi.

Hardware implementation of QSPI LCD:

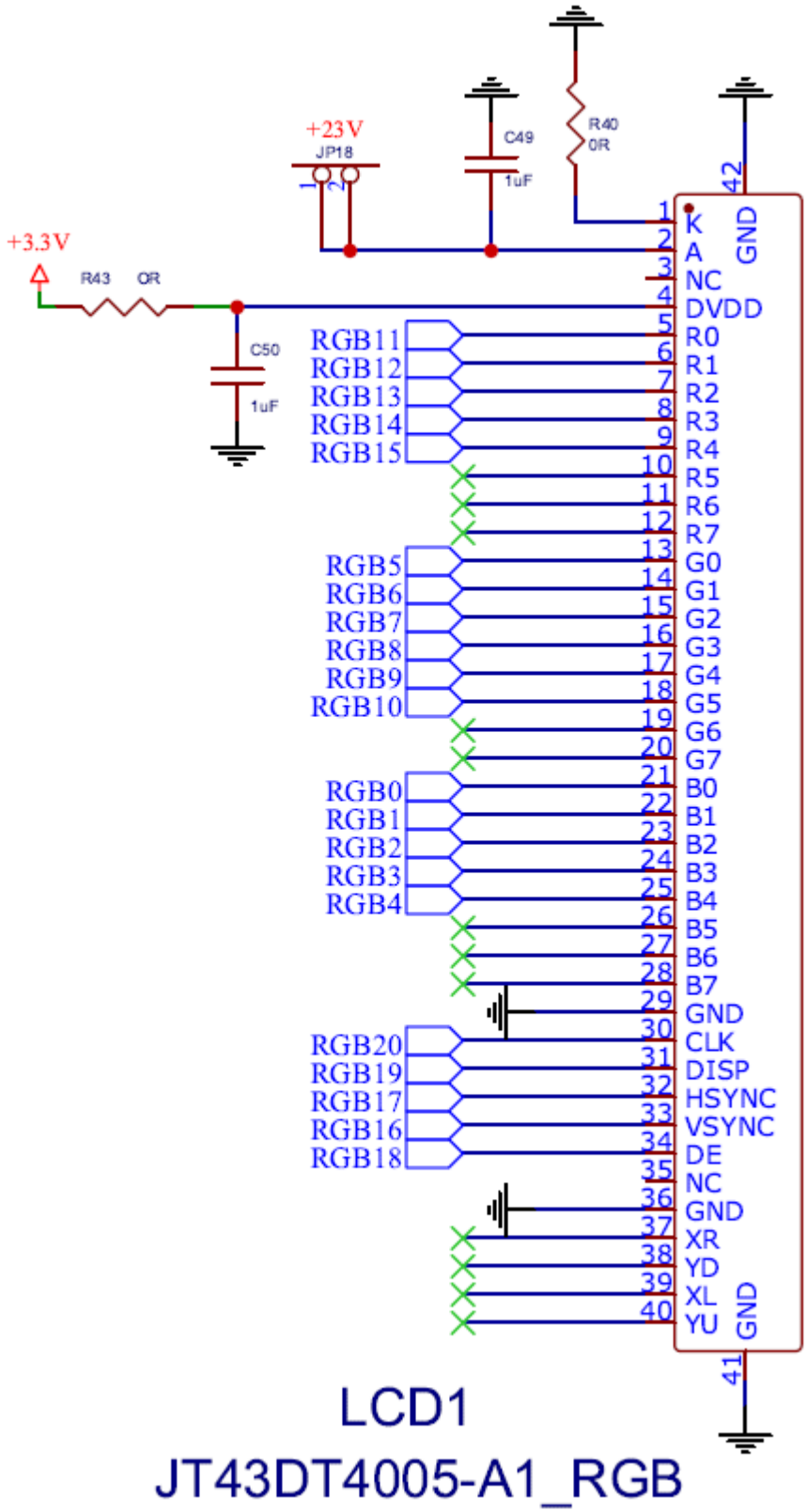


Fig. 9: Figure 9. rgb lcd pin

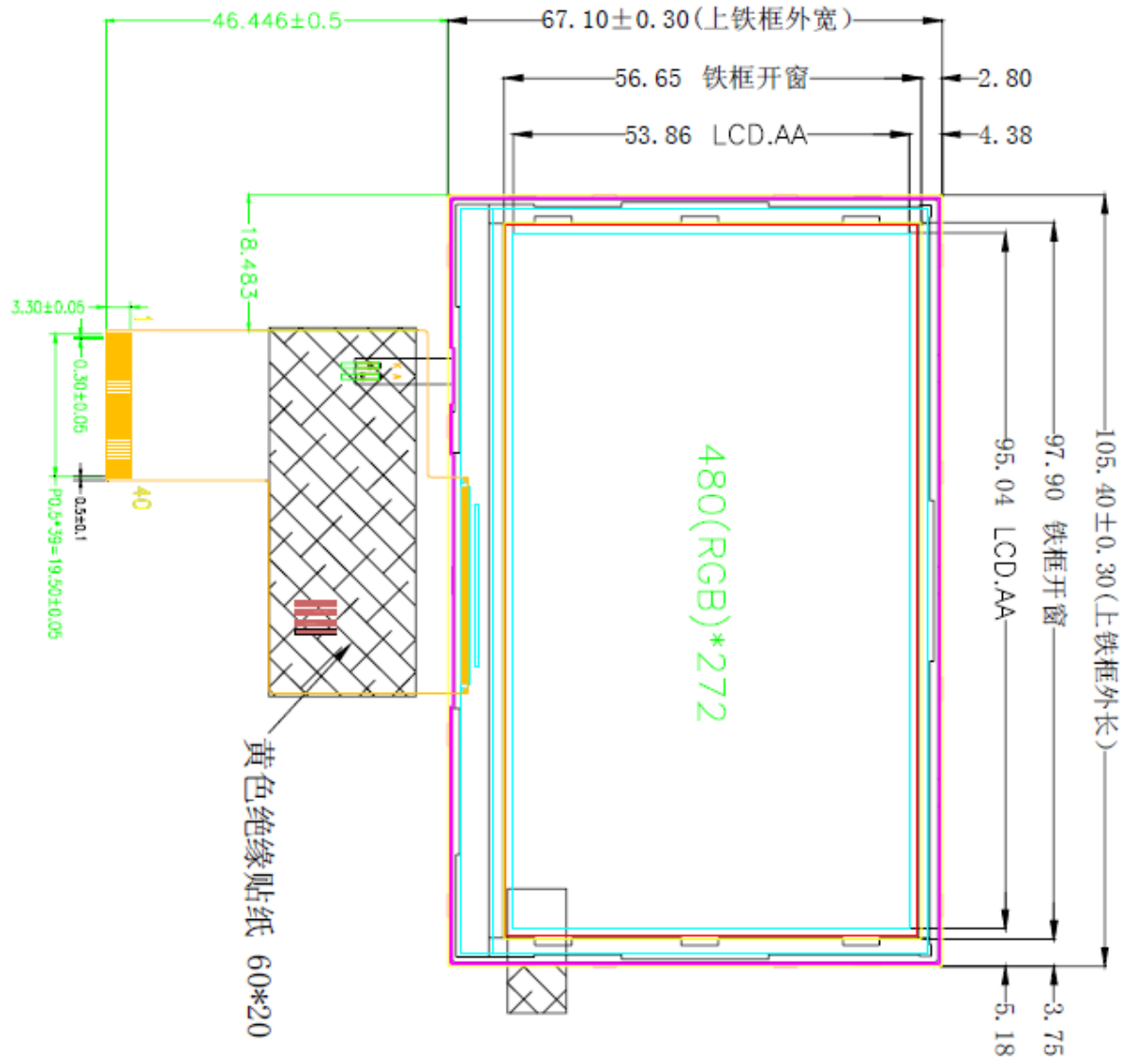


Fig. 10: Figure 10. rgb lcd floor plan

GPIO	IO Function 5	Describtion
RGB11-15	R0-R4	Red data
RGB5-10	G0-G5	Green data
RGB0-4	B0-B4	Blue data
RGB20	DCLK	输出给外屏的驱动时钟
RGB19	DISP	Display on/off，置 1 打开屏幕，置 0 关闭屏幕。
RGB17	HSYNC	Horizontal sync signal,HSYNC 会在每行数据传输前置 1，在每行数据传输 结束置 0。
RGB16	VSYNC	Vertical sync signal，在一帧数据传输期间，VSYNC 会一直置 1，直到一帧传输完成后。
RGB18	DE	Data enable, 1 数据有效

Fig. 11: Figure 11. rgb lcd pin function

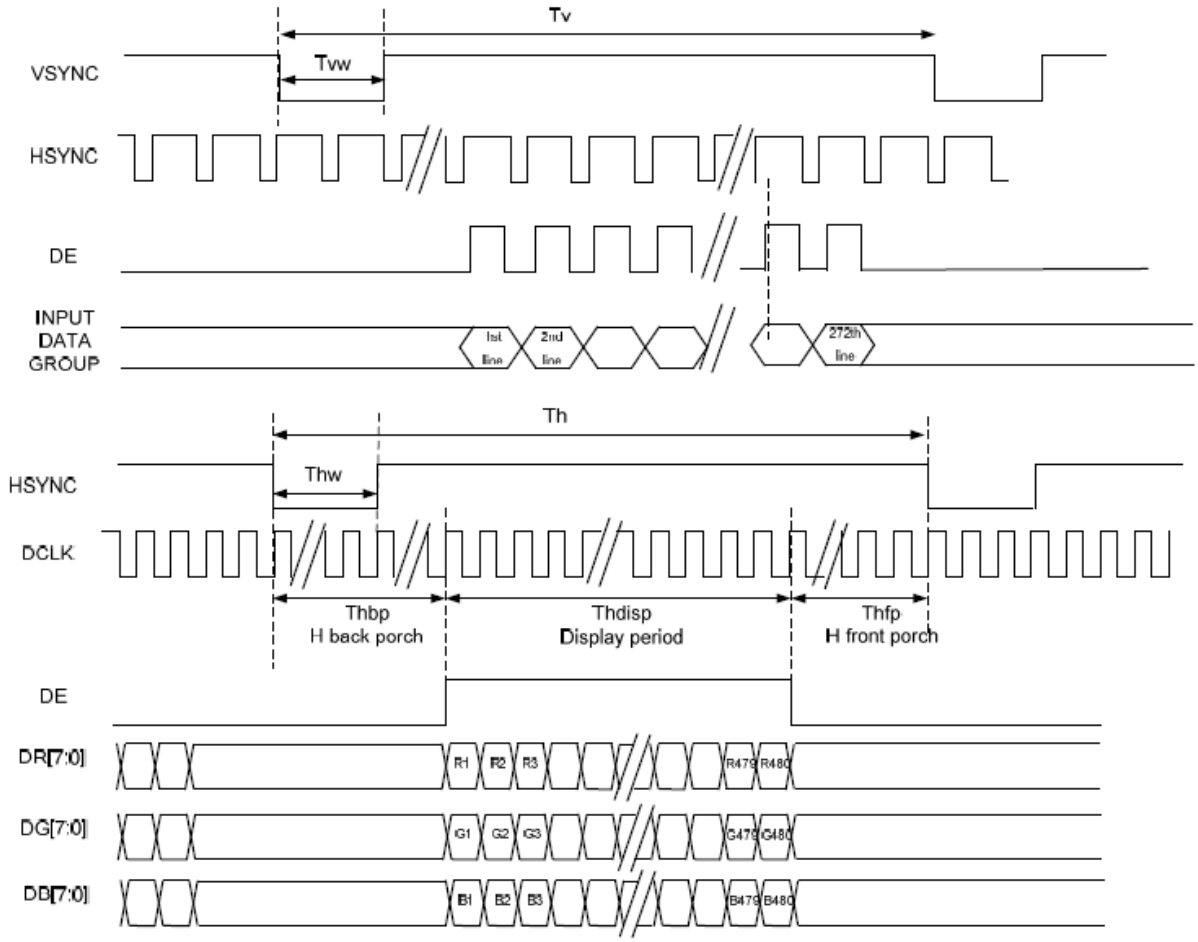


Fig. 12: Figure 12. rgb lcd timing

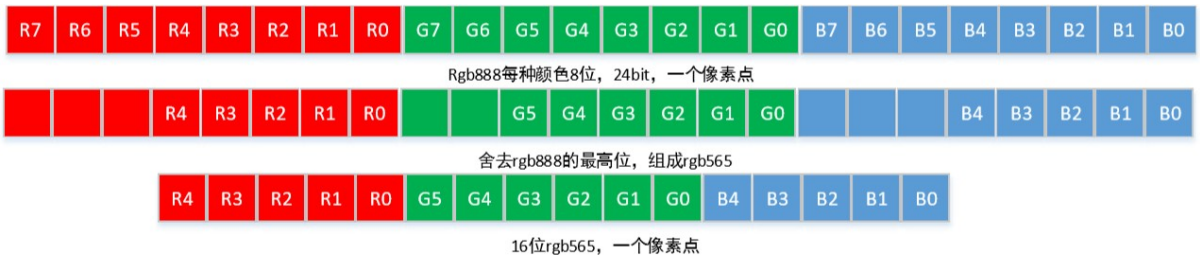


Fig. 13: Figure 13. rgb lcd protocol

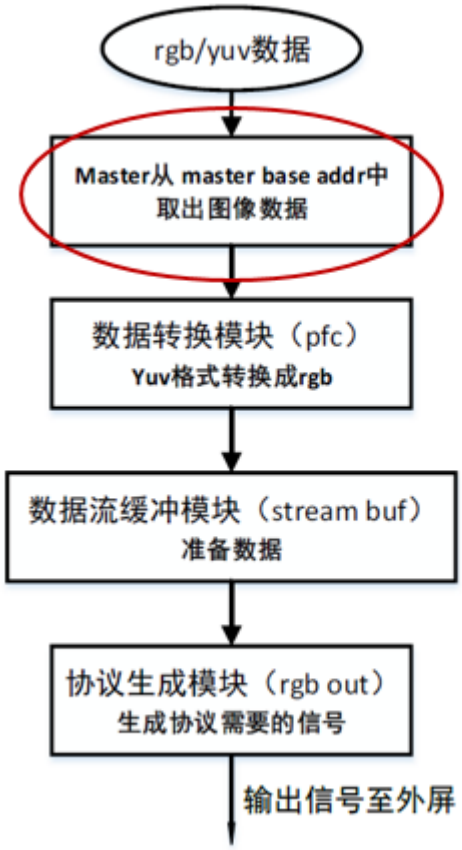


Fig. 14: Figure 14. rgb lcd hardware process

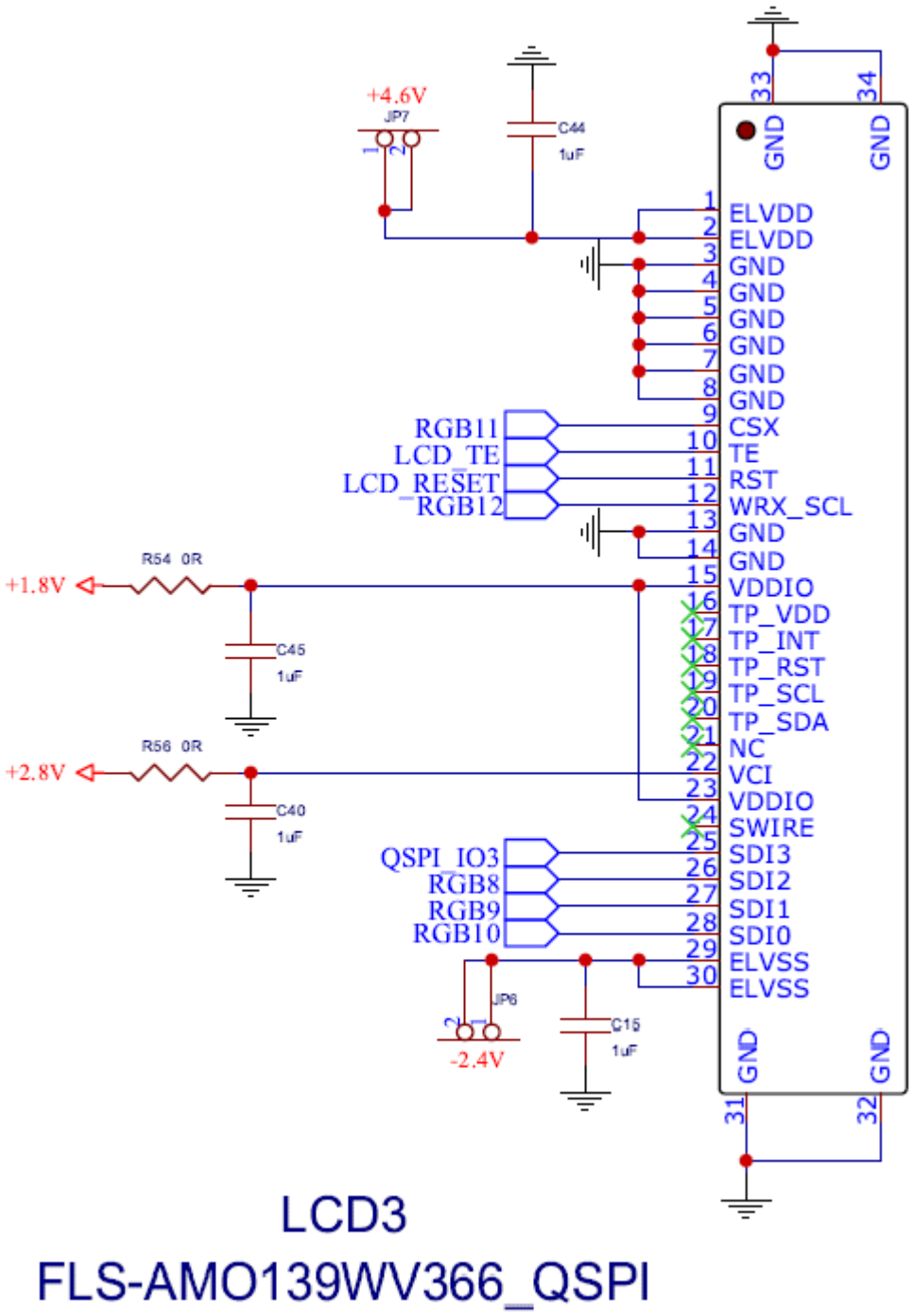
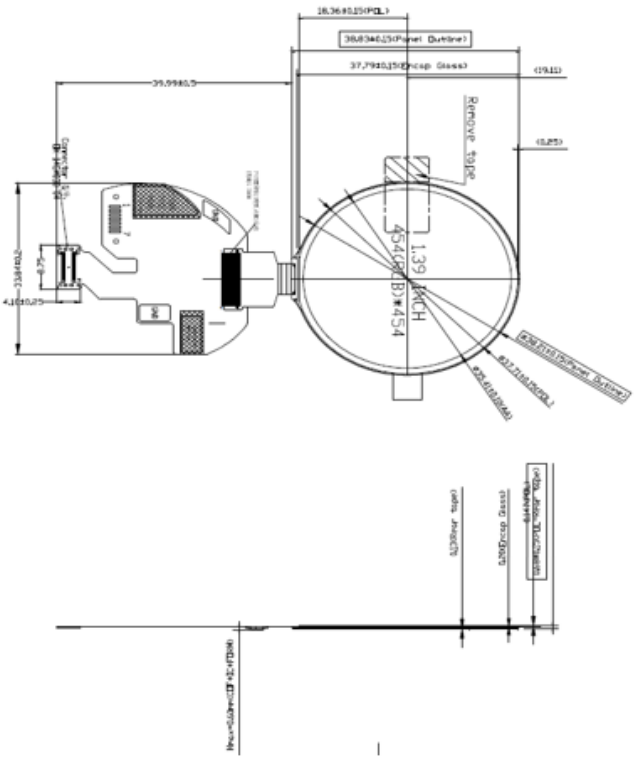


Fig. 15: Figure 15. qspi lcd pin



No.	Symbol
1	GND
2	TP_SDA
3	TP_SCL
4	TP_RST
5	TP_INT
6	TP_VDD
7	GND

Fig. 16: Figure 16. qspi lcd floor plan

GPIO	IO Function 5	Describtion
RGB11	CSX	Chip Select Signal ,Active Low
LCD_TE	TE	Tearing effect output pin to synchronize MCU to frame
LCD_RESET	RST	Device reset signal 0 : Enable ; 1: Disable
RGB12	WRX_SCL	Synchronous Clock,Serial clock, Rising edge
RGB10	SDI0	Serial Data Input in Quad-SPI
RGB 9	SDI1	Serial Data Input in Quad-SPI
RGB 8	SDI2	Serial Data Input in Quad-SPI
QSPI_IO3	SD13	Serial Data Input in Quad-SPI

Fig. 17: Figure 17. qspi lcd pin function

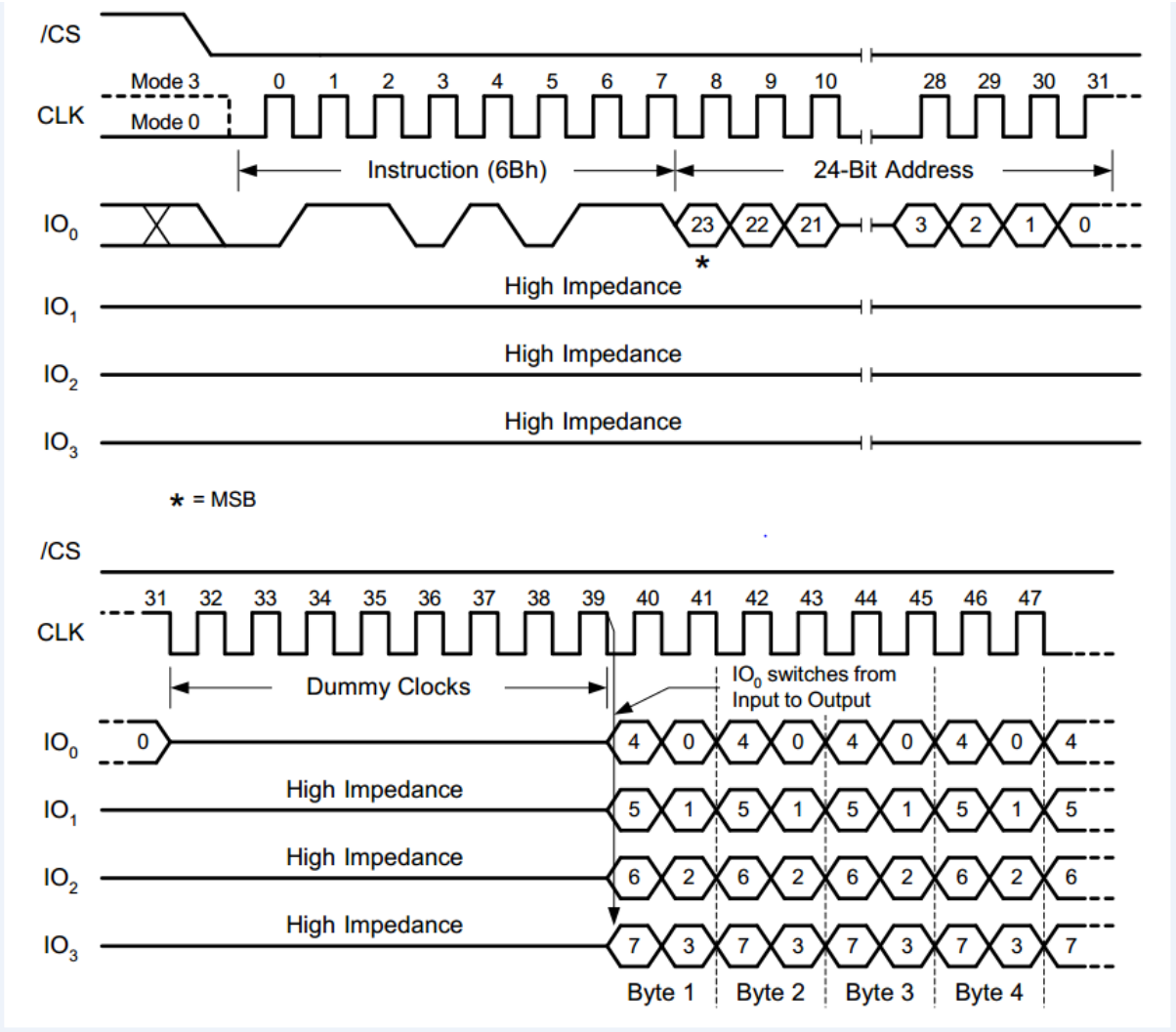


Fig. 18: Figure 18. qspi lcd timing

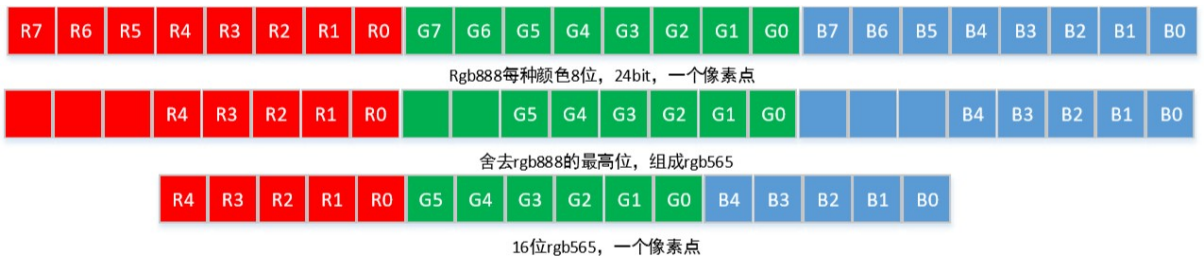


Fig. 19: Figure 19. qspi lcd hardware process

5.10 software design

The design layering idea of the software code in the project is as follows:

Fig. 20: Figure 20. software architecture

Driver layer meaning: For different chip boards, even if the SOC layer is different, the LCD driver interface called is the same. Code design idea: Since the LCD has three different interfaces, the APIs that need to be independently packaged for each interface need to be named 8080_lcd, rgb_lcd, qspi_lcd to distinguish them, while the public API names are not distinguished.

5.11 related data structures

enumeration definition of image format:

```
typedef enum {
    PIXEL_FMT_UNKNOW,         /**< unknow image format */
    PIXEL_FMT_JPEG,           /**< image foramt jpeg */
    PIXEL_FMT_H264,
    PIXEL_FMT_H265,
    PIXEL_FMT_YUV444,
    PIXEL_FMT_YUYV,  /**< lcd/jpeg_decode support */
    PIXEL_FMT_VYUY,  /**< jpeg_decode support */
    PIXEL_FMT_UYVY,
    PIXEL_FMT_YYUV,   /**< jpeg_decode support */
    PIXEL_FMT_VUYY,   /**< jpeg_decode support */
    PIXEL_FMT_UVYY,
    PIXEL_FMT_YUV422,
    PIXEL_FMT_I420,
    PIXEL_FMT_YV12,
    PIEXL_FMT_YUV420P,
    PIXEL_FMT_NV12,
    PIXEL_FMT_NV21,
    PIXEL_FMT_YUV420SP,
    PIXEL_FMT_YUV420,
    PIXEL_FMT_RGB444,
    PIXEL_FMT_RGB555,
    PIXEL_FMT_RGB565,     /**< input data format is rgb565(big endian), high pixel is bit[31-16], low pixel is bit[15-0] (PIXEL BIG ENDIAN)*/
    PIXEL_FMT_RGB565_LE,  /**< input data format is rgb565(big endian), high pixel is bit[15-0], low pixel is bit[31-16] (PIXEL little ENDIAN)*/
    PIXEL_FMT_BGR565,
    PIXEL_FMT_RGB666,
    PIXEL_FMT_RGB888,
    PIXEL_FMT_BGR888,
    PIXEL_FMT_ARGB8888,
    PIXEL_FMT_GRAY,
    PIXEL_FMT_RAW,
    PIXEL_FMT_PNG,
} pixel_format_t;
```

Fig. 21: Figure 21. format enum

enumeration definition of LCD screen device:

enumeration definition of LCD pixels:

enumeration definition of FPS:

```
typedef enum {
    LCD_DEVICE_UNKNOW,
    LCD_DEVICE_ST7282,  /**< 480X270 RGB  */
    LCD_DEVICE_HX8282,  /**< 1024X600 RGB  */
    LCD_DEVICE_GC9503V, /**< 480X800 RGB  */
    LCD_DEVICE_NT35510, /**< 480X854 RGB  */
    LCD_DEVICE_H050IWV, /**< 800X480 RGB  */
    LCD_DEVICE_MD0430R, /**< 800X480 RGB  */
    LCD_DEVICE_MD0700R, /**< 1024X600 RGB  */
    LCD_DEVICE_ST7701S_LY, /**< 480X480 RGB  */
    LCD_DEVICE_ST7701S, /**< 480X480 RGB  */
    LCD_DEVICE_ST7701SN, /**< 480X480 RGB  */
    LCD_DEVICE_AML01,   /**< 720X1280 RGB  */

    LCD_DEVICE_ST7796S, /**< 320X480 MCU  */
    LCD_DEVICE_NT35512, /**< 480X800 MCU  */
    LCD_DEVICE_NT35510_MCU, /**< 480X800 MCU  */
    LCD_DEVICE_ST7789V, /**< 170X320 MCU  */

    LCD_DEVICE_SH8601A, /**< 454X454 QSPI  */
    LCD_DEVICE_ST77903_WX20114, /**< 400X400 QSPI  */
    LCD_DEVICE_ST77903_SAT61478M, /**< 400X400 QSPI  */
    LCD_DEVICE_ST77903_H0165Y008T, /**< 360X480 QSPI  */
    LCD_DEVICE_SPD2010, /**< 412X412 QSPI  */
} lcd_device_id_t;
```

Fig. 22: Figure 22. lcd device model enumeration

```
typedef enum
{
    PPI_DEFAULT     = 0,
    PPI_170X320     = (PIXEL_170 << 16) | PIXEL_320,
    PPI_320X240     = (PIXEL_320 << 16) | PIXEL_240,
    PPI_320X480     = (PIXEL_320 << 16) | PIXEL_480,
    PPI_360X480     = (PIXEL_360 << 16) | PIXEL_480,
    PPI_400X400     = (PIXEL_400 << 16) | PIXEL_400,
    PPI_412X412     = (PIXEL_412 << 16) | PIXEL_412,
    PPI_454X454     = (PIXEL_454 << 16) | PIXEL_454,
    PPI_480X272     = (PIXEL_480 << 16) | PIXEL_272,
    PPI_480X320     = (PIXEL_480 << 16) | PIXEL_320,
    PPI_480X480     = (PIXEL_480 << 16) | PIXEL_480,
    PPI_640X480     = (PIXEL_640 << 16) | PIXEL_480,
    PPI_480X800     = (PIXEL_480 << 16) | PIXEL_800,
    PPI_480X854     = (PIXEL_480 << 16) | PIXEL_854,
    PPI_480X864     = (PIXEL_480 << 16) | PIXEL_864,
    PPI_720X288     = (PIXEL_720 << 16) | PIXEL_288,
    PPI_720X576     = (PIXEL_720 << 16) | PIXEL_576,
    PPI_720X1280    = (PIXEL_720 << 16) | PIXEL_1280,
    PPI_854X480     = (PIXEL_854 << 16) | PIXEL_480,
    PPI_800X480     = (PIXEL_800 << 16) | PIXEL_480,
    PPI_864X480     = (PIXEL_864 << 16) | PIXEL_480,
    PPI_960X480     = (PIXEL_960 << 16) | PIXEL_480,
    PPI_800X600     = (PIXEL_800 << 16) | PIXEL_600,
    PPI_1024X600    = (PIXEL_1024 << 16) | PIXEL_600,
    PPI_1280X720    = (PIXEL_1280 << 16) | PIXEL_720,
    PPI_1600X1200   = (PIXEL_1600 << 16) | PIXEL_1200,
    PPI_1920X1080   = (PIXEL_1920 << 16) | PIXEL_1080,
} media_ppi_t;
```

Fig. 23: Figure 23. pixel enum

```
typedef enum
{
    FPS0    = 0,        /**< 0fps */
    FPS5    = (1 << 0), /**< 5fps */
    FPS10   = (1 << 1), /**< 10fps */
    FPS15   = (1 << 2), /**< 15fps */
    FPS20   = (1 << 3), /**< 20fps */
    FPS25   = (1 << 4), /**< 25fps */
    FPS30   = (1 << 5), /**< 30fps */
} frame_fps_t;
```

Fig. 24: Figure 24. fps enum

5.12 code interface

code API as follows:

```
bk_err_t bk_lcd_driver_init(lcd_clk_t clk);

/**
* @brief    This API select LCD module clk source
*          - open lcd sys interrupt/clk enable
* @return
*     - BK_OK: succeed
*     - others: other errors.
*/

bk_err_t bk_lcd_driver_deinit(void);

/**
* @brief  close lcd sys interrupt/clk enable
* @return
*     - BK_OK: succeed
*     - others: other errors.
*/


bk_err_t bk_lcd_8080_init(const lcd_device_t *device);

/**
 * @brief This API init the 8080 lcd interface
 *    - Set lcd display mode is 8080 interface
 *    - init 8080 lcd gpio
 *    - enable 8080 display
 *    -enable 8080 end of frame interrupt
 *    - if you want enable start of frame interrupt, please use API bk_lcd_8080_int_enable
 *
 * @param
 *     - lcd_device_t
 *
 * @return
 *     - BK_OK: succeed
 *     - others: other errors.
 */
```

Fig. 25: Figure 25. api_1

5.13 code configuration process

LCD 8080 configuration flow chart is as follows:

LCD RGB configuration flow chart is as follows:

The flow chart for setting up special area display is as follows:

partial display diagram:

for the use of the RGB, please refer to the project “https://docs.bekencorp.com/arminodoc/bk_avdk/bk7258/en/v2.0.1/projects_work/media/lcd_rgb/index.htmll”

for the use of the 8080, please refer to the project “https://docs.bekencorp.com/arminodoc/bk_avdk/bk7258/en/v2.0.1/projects_work/media/lcd_8080/index.html”

for the use of the QSPI, please refer to the project “https://docs.bekencorp.com/arminodoc/bk_avdk/bk7258/en/v2.0.1/projects_work/media/lcd_qspi/index.html”

```
bk_err_t bk_lcd_8080_deinit(void);

/**
* @brief 8080 lcd interface reg deinit
*      - This API reset all lcd reg include power
*      - close 8080 lcd enable and display
*      - reset x pixel and y pixel zero
*      - unregister lcd isr
*
* @return
*      - BK_OK: succeed
*      - others: other errors.
*/


/**
 * @brief This API config 8080 lcd interrupt

 * @param
 *      - is_sof_en enable start of frame interrupt
 *      - is_eof_en enable end of frame interrupt
 *
 * @attention 8080 end of frame int is open in API bk_lcd_8080_init
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
//bk_err_t bk_lcd_8080_int_enable(bool is_sof_en, bool is_eof_en);

bk_err_t bk_lcd_8080_start_transfer(bool start);

/**
 * @brief This API start mcu 8080 lcd transfer data to display
 *
 * @param start_transfer
 *       - 1:data start transfer to lcd display on;
 *       - 0:stop  transfer
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
```

Fig. 26: Figure 25. api_2

```
bk_err_t bk_lcd_rgb_deinit(void);

/**
 * @brief     rgb lcd interface reg deinit
 *           - This API reset all lcd reg include power
 *           - close rgb lcd enable and display
 *           - reset x pixel and y pixel zero
 *           - unregister lcd isr
 *
 * @return
 *     - BK_OK: succeed
 *     - others: other errors.
 */


bk_err_t bk_lcd_rgb_display_en(bool en);
/**
 * @brief  enable rgb lcd display
 * @param  bool en
 *       - 1: enable rgb display
 *       - 0: disable rgb display
 *
 * @return
 *     - BK_OK: succeed
 *     - others: other errors.
 */
```

Fig. 27: Figure 25. api_3

```
#if (USE_LCD_REGISTER_CALLBACKS == 1)
bk_err_t bk_lcd_isr_register(lcd_int_type_t int_type, lcd_isr_t isr);

/**
 * @brief This API register  8080/rgb lcd int isr
 *
 * @param
 *      - int_type  include  8080/rgb  end of frame int and  8080/rgb  start of frame int
 *      - isr: isr function
 *
 * Usage example:
 *
 *       bk_lcd_isr_register(I8080_OUTPUT_EOF, lcd_i8080_isr); //register 8080 end of frame isr
 *
  * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */

#else ...
#endif
uint32_t bk_lcd_int_status_get(void);
/**
 * @brief This API used to get lcd int status
 *
 * @return
 *      - status value.
 *
 * 8080 Usage example:
 *      if (bk_lcd_int_status_get()&I8080_OUTPUT_EOF == 1), present 8080 eof int triggerd
 *      if (bk_lcd_int_status_get()&RGB_OUTPUT_EOF == 1), present rgb eof int triggerd
 *
 */
bk_err_t bk_lcd_int_status_clear(lcd_int_type_t int_type);
/**
 * @brief This API used to clr lcd int status
 *
 * @param
 *      - int_type value.
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
```

Fig. 28: Figure 25. api_4

```
bk_err_t bk_lcd_pixel_config(uint16_t x_pixel, uint16_t y_pixel);

/**
 * @brief This API config lcd display x size and y size
 *
 * @param
 *      - x_pixel  user can set by any value, can be lcd size x or picture size x
 *      - y_pixel  user can set by any value, can be lcd size y or picture size y
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
bk_err_t bk_lcd_rgb_init(const lcd_device_t *device);

/**
 * @brief This API init the rgb lcd interface
 *     - Set lcd display mode is rgb interface
 *     - init rgb lcd gpio
 *     - enable rgb display
 *     - enable rgb end of frame interrupt
 *
 * @param
 *      - lcd_device_t: lcd type select from lcd_device_t
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
```

Fig. 29: Figure 25. api_5

```
bk_err_t bk_lcd_8080_send_cmd(uint8_t param_count, uint32_t command, uint32_t *param);

/**
 * @brief This API used send 8080 lcd init cmd
 *
 * @param
 *      - param_count: cmd parameter number
 *      - command: command
 *      - param: the cmd parameter
 *
 * Usage example:
 *
 *    #define COMMAND_1 0xf
 *    uint32_t param_command1[2]   = {0xc3, 0x29};
 *    bk_lcd_8080_send_cmd(2, COMMAND_1, param_command1);
 *
  * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
bk_err_t bk_lcd_set_partical_display(bool en, uint16_t partial_clum_l, uint16_t partial_clum_r,
                                     uint16_t partial_line_l, uint16_t partial_line_r);

/**
 * @brief This API used for display partical area
 *
 * @param
 *      - partial_clum_l:
 *      - partial_clum_r:
 *      - partial_line_l:
 *      - partial_line_r:
 *
 * 8080 Usage example:
 *
 *    #define PARTICAL_XS   101
 *    #define PARTICAL_XE   220
 *    #define PARTICAL_YS   101
 *    #define PARTICAL_YE   380
 *    bk_lcd_set_partical_display(EDGE_PARTICAL_XS, EDGE_PARTICAL_XE, EDGE_PARTICAL_YS, EDGE_PARTICAL_YE);
 *    bk_lcd_8080_send_cmd(2, COMMAND_1, param_command1);
 *
  * @return
 *      - BK_OK: succeed
 *      - others: other errors.
```

Fig. 30: Figure 25. api_6

```
void lcd_driver_ppi_set(uint16_t width, uint16_t height);

/**
 * @brief This API used to clr lcd int status
 *
 * @param
 *      - int_type value.
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */

bk_err_t bk_lcd_set_yuv_mode(pixel_format_t input_data_format);

/**
 * @brief This API used to set lcd display data farmat
 *
 * @param
 *      - input_data_format value.
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */

bk_err_t lcd_driver_init(const lcd_device_t *device);

/**
 * @brief This API used for init lcd
 *
 *
 * @param lcd_config_t
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 *
```

Fig. 31: Figure 25. api_7

```
const lcd_device_t *get_lcd_device_by_id(lcd_device_id_t id);

/**
 * @brief this api used to get lcd device interface
 *
 * @param select lcd_device_id_t member
 * @return lcd device information, include:
 *   .id = LCD_DEVICE_HX8282,
 *   .name = "hx8282",
 *   .type = LCD_TYPE_RGB565,
 *   .ppi = PPI_1024X600,
 *   .rgb = &lcd_rgb
 *   .init = NULL,
 */

const lcd_device_t **get_lcd_devices_list(void);

/**
 * @brief get lcd device list
 * @return
 *      - devices list
 */

uint32_t get_lcd_devices_num(void);

/**
 * @brief get lcd device number
 * @return
 *      - devices number
 */

bk_err_t lcd_driver_backlight_open(void);

/**
 * @brief this api used to open lcd backlight
 *
 * @param none
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
```

Fig. 32: Figure 25. api_8

```
bk_err_t lcd_driver_backlight_close(void);

/**
 * @brief this api used to close lcd backlight
 *
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */

bk_err_t lcd_driver_backlight_set(uint8_t percent);

/**
 * @brief this api used to set lcd backlight
 *
 * @param percent rang from 0~100
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */

bk_err_t lcd_driver_display_disable(void);

/**
 * @brief  this api used to disable lcd display
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */

bk_err_t lcd_driver_display_enable(void);

/**
 * @brief this api used to enable lcd display
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
```

Fig. 33: Figure 25. api_9

```
bk_err_t lcd_driver_display_continue(void);

/**
 * @brief this api used to clear eof int and diaplay continue
 *
 * @return
 *     - BK_OK: succeed
 *     - others: other errors.
 */

bk_err_t lcd_driver_set_display_base_addr(uint32_t disp_base_addr);

/**
 * @brief this api used to set lcd display addr
 *
 * @param disp_base_addr sram or psram
 *
 * @return
 *     - BK_OK: succeed
 *     - others: other errors.
 */

bk_err_t lcd_driver_deinit(void);

/**
 * @brief this api used to close lcd
 *
 * @return
 *     - BK_OK: succeed
 *     - others: other errors.
 */

const lcd_device_t *get_lcd_device_by_ppi(media_ppi_t ppi);

/**
 * @brief get lcd device struct by ppi
 *
 *
 * @param lcd ppi
 *
 * @return
 *     - return true is lcd_device_t member
 *     - or not find device, return NULL.
 */
```

Fig. 34: Figure 25. api_10

```
const lcd_device_t * get_lcd_device_by_name(char * name);
/**
 * @brief get lcd device struct by device name
 *
 *
 * @param lcd name
 *
 * @return
 *      - return true is lcd_device_t member
 *      - or not find device, return NULL.
 */


bk_err_t bk_lcd_input_pixel_hf_reverse(bool hf_reverse);
/**
 * @brief input data halfword reverse
 *
 *
 * @param enable enable/disable hf_reverse
 *
 * @return
 *      - BK_OK: succeed
 *      - others: other errors.
 */
```

Fig. 35: Figure 25. api_11

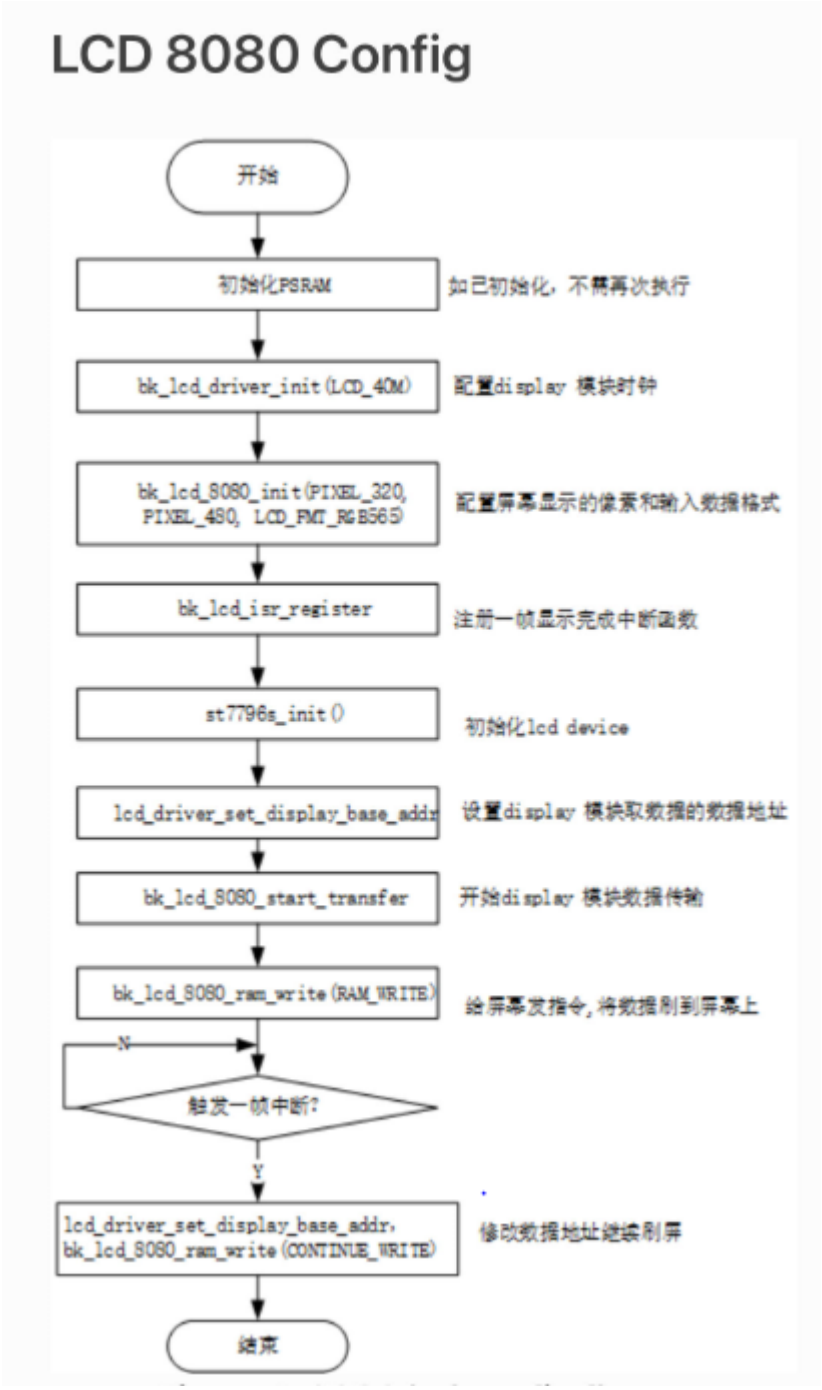


Fig. 36: Figure 26. 8080 configuration flow chart

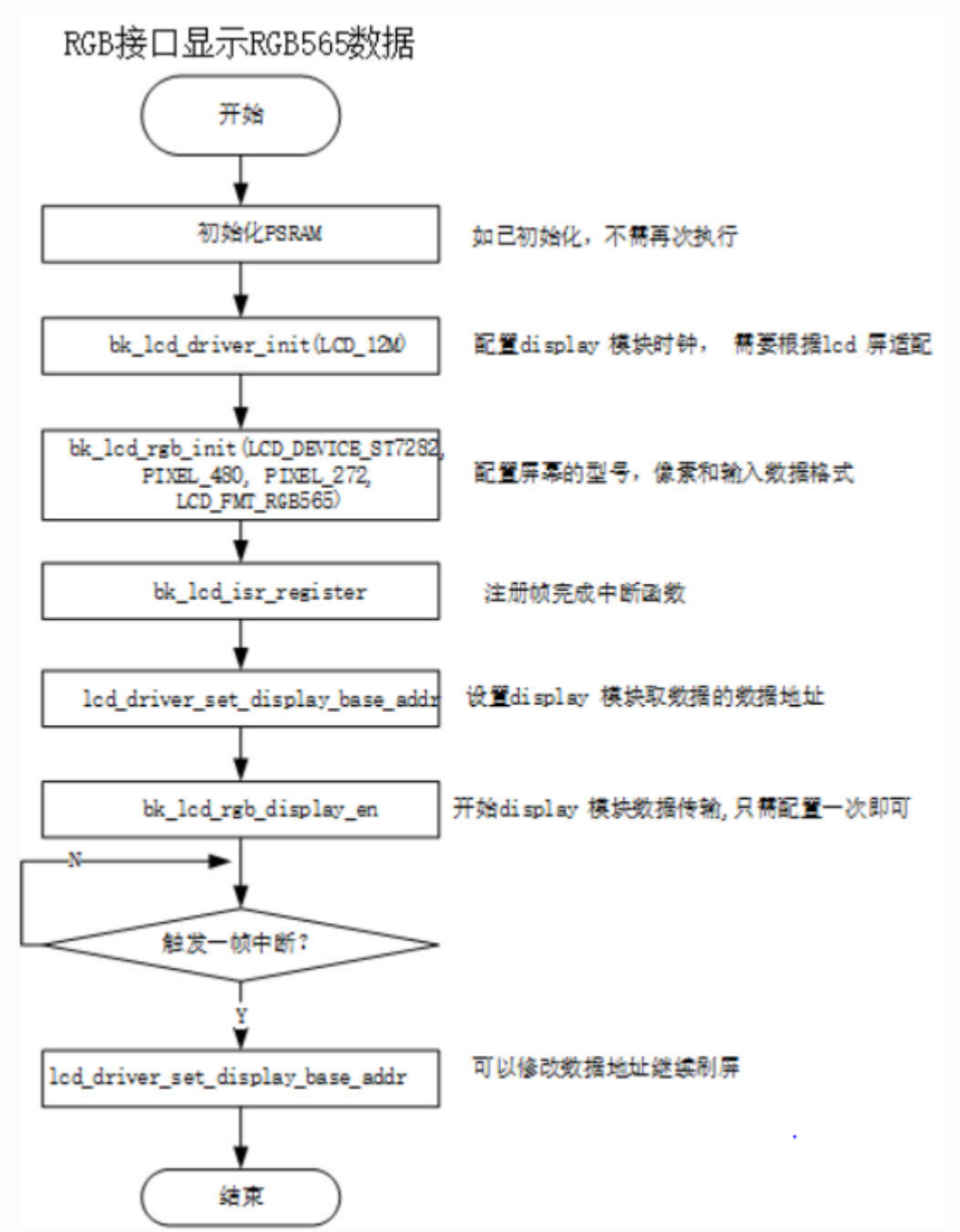


Fig. 37: Figure 27. rgb configuration flow chart

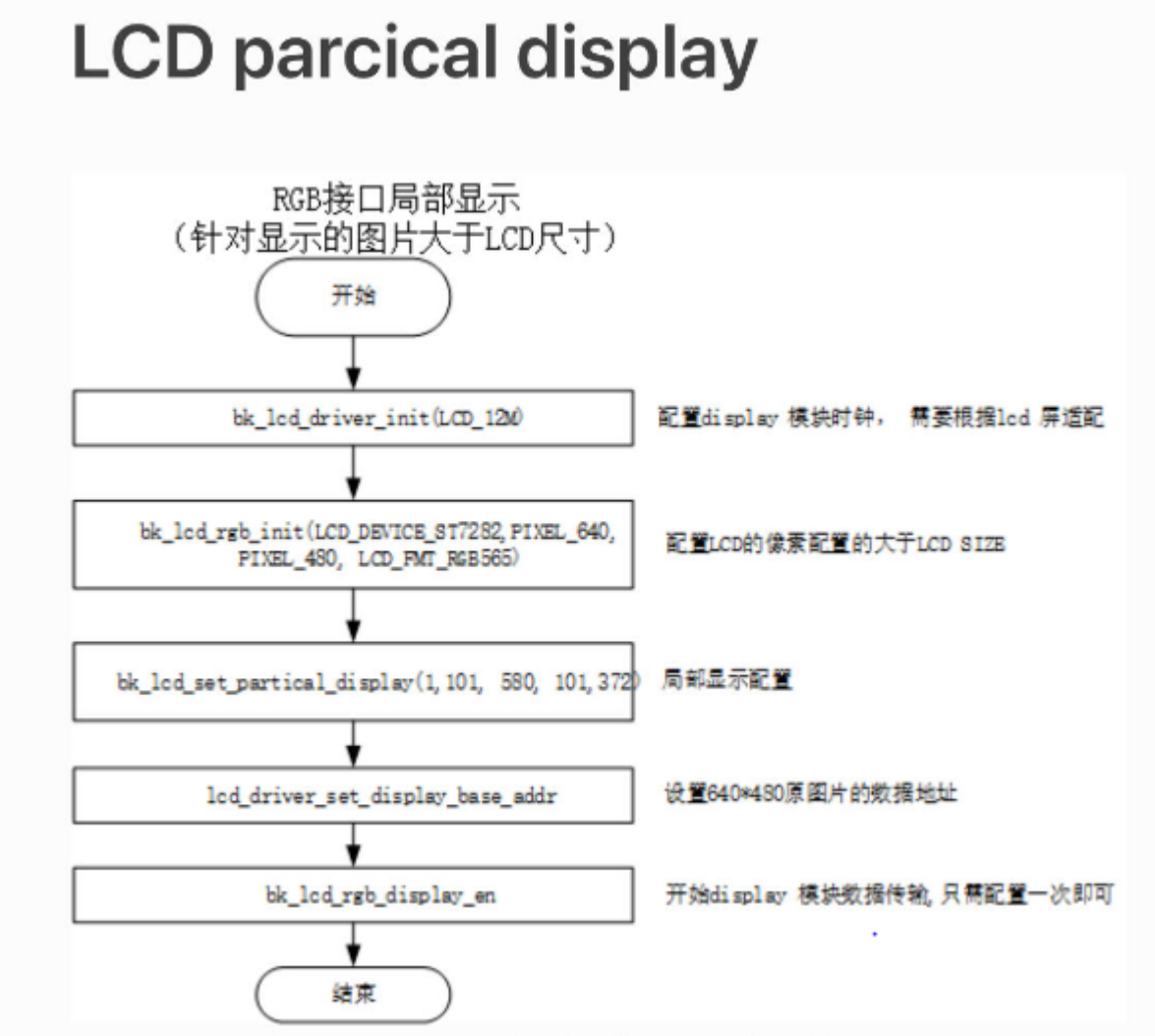


Fig. 38: Figure 28. special area configuration diagram

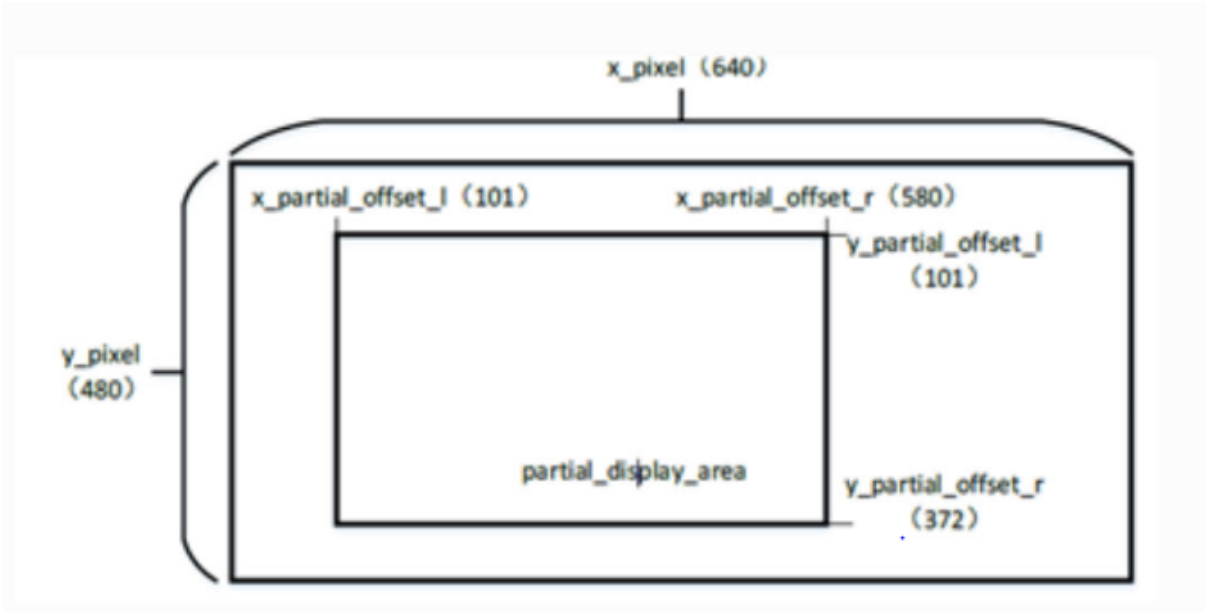


Fig. 39: Figure 29. special area display map

CHAPTER

SIX

CAMERA

6.1 Camera Overview

6.1.1 1 Working Principle

The scene is projected to the surface of the image SENSOR through the optical image generated by the LENS, and then converted into an electrical signal, converted into A digital image signal after A/D (analog-to-digital conversion), and then sent to the digital signal processing chip (DSP) for processing, after output YUV or RGB format data.

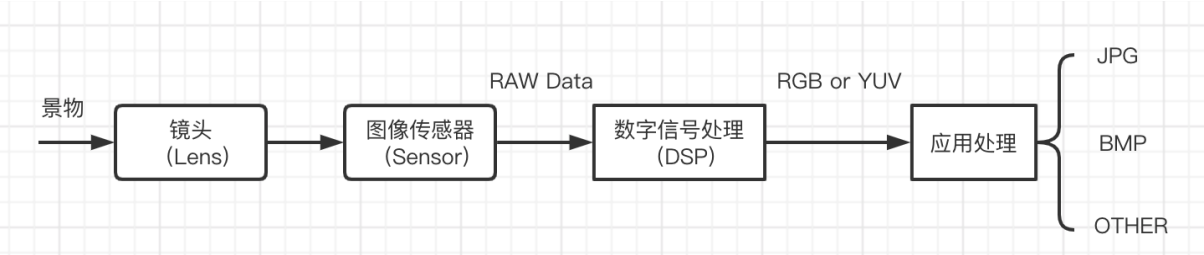


Fig. 1: Figure 1.Working principle and process

6.1.2 2 Main Components

Generally speaking, camera is mainly composed of lens and sensor IC two parts, of which some sensor IC integrated DSP, some not integrated, but also need external DSP processing.

1.lens

The lens structure of the camera is composed of several lenses, divided into Plastic lenses and Glass lenses, usually the lens structure is: 1P,2P,1G1P,1G3P,2G2P,4G and so on.

2. sensor (Image sensor)

- Sensor is a Semiconductor chip that comes in two types: CCD (Charge Coupled Device), short for charge-coupled device, and CMOS (Complementary Metal-Oxide Semiconductor), complementary metal-oxide semiconductor.
- The Sensor converts the light transmitted from the lens into an electrical signal, and then converts it into a digital signal through the internal AD. Since each pixel of the sensor can only be sensitive to R light or B light or G light, each pixel is stored in monochrome at this time, which is called RAW DATA. In order to

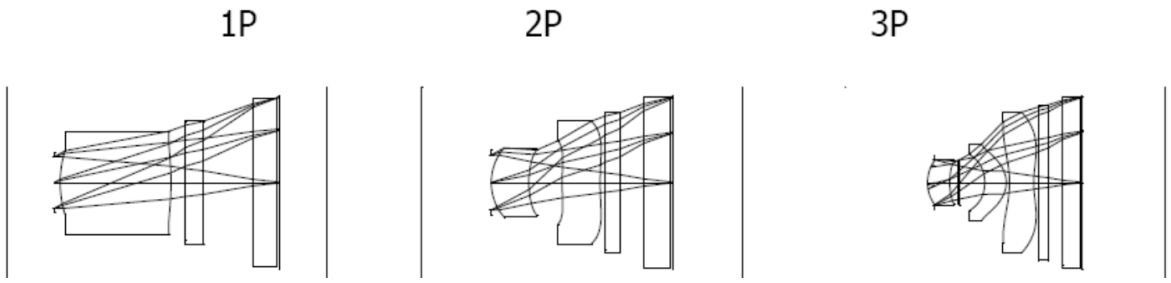


Fig. 2: Figure 2.Lens structure

restore the RAW DATA data of each pixel to the three primary colors, it needs to be processed by the ISP. ISP (Image signal processing) mainly completes the processing of digital images, and converts the original data collected by the sensor into the format supported by the display.

Note:

- 1. CCD sensor, charge signal first transmitted, after amplification, and then A/D, imaging quality high sensitivity, good resolution, low noise; Slow processing speed; The cost is high and the process is complex.
- 2. CMOS sensor, charge signal first amplified, after A/D, and then transmitted; The imaging quality sensitivity is low and the noise is obvious. Fast processing speed; Low cost, simple process.

3.CAMIF (camera controller)

The camera interface circuit on the chip controls the device, receives the data collected by the sensor, sends it to the CPU, and sends it to the LCD for display.

6.1.3 3 Camera Ports

There are two common camera interfaces: UVC interface and DVP interface

- USB port, only data cable, no clock cable.

- DVP(Digital Video Port/Parally Port)interface, mainly composed of power bus, input bus, output bus.
  - Input bus

PWDN: camera enable pin, can be configured in two modes, one is standby, one is normal work. When the camera is configured as standby, all operations including reset are invalid for the camera, so the pin must be set to normal work to reset it.

MCLK: The working clock provided to the camera.

IIC_SDA/IIC_SCL: Registers used to read and write sensors.

- Output bus

PCLK: pixel sync signal pin.

VSYNC: indicates the frame synchronization signal.

HSYN: indicates the line synchronization signal.

DATA[0-7] : Output data pin.

- Power bus

AVDD: The analog voltage of the camera, which mainly supplies power to the photosensitive region of the camera and the ADC part.

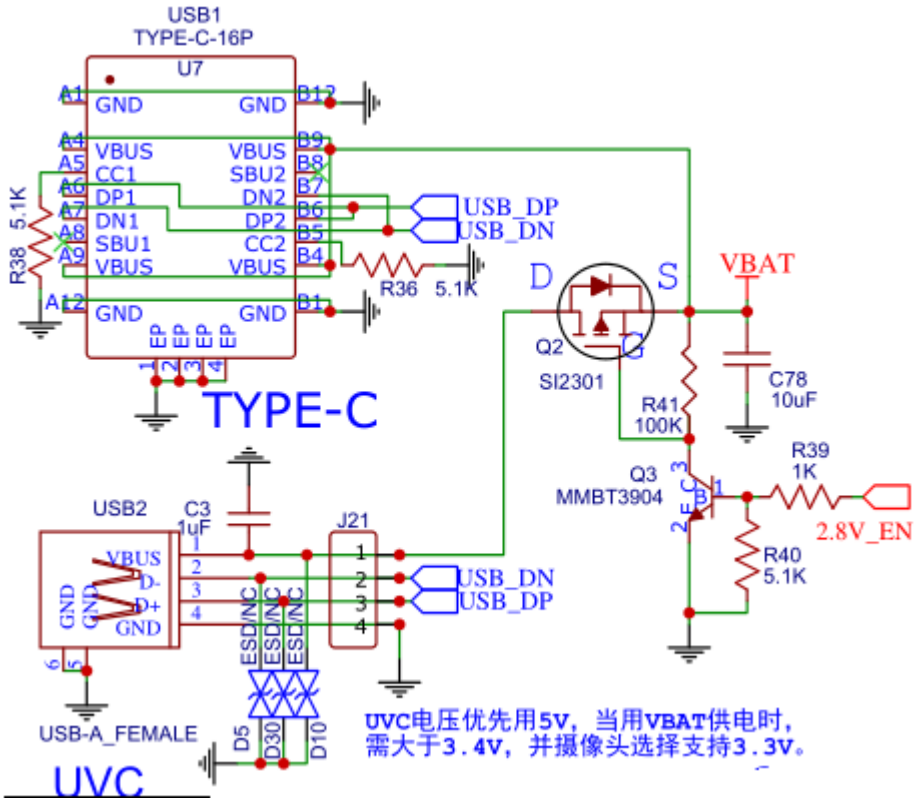


Fig. 3: Figure 3.UVC interface

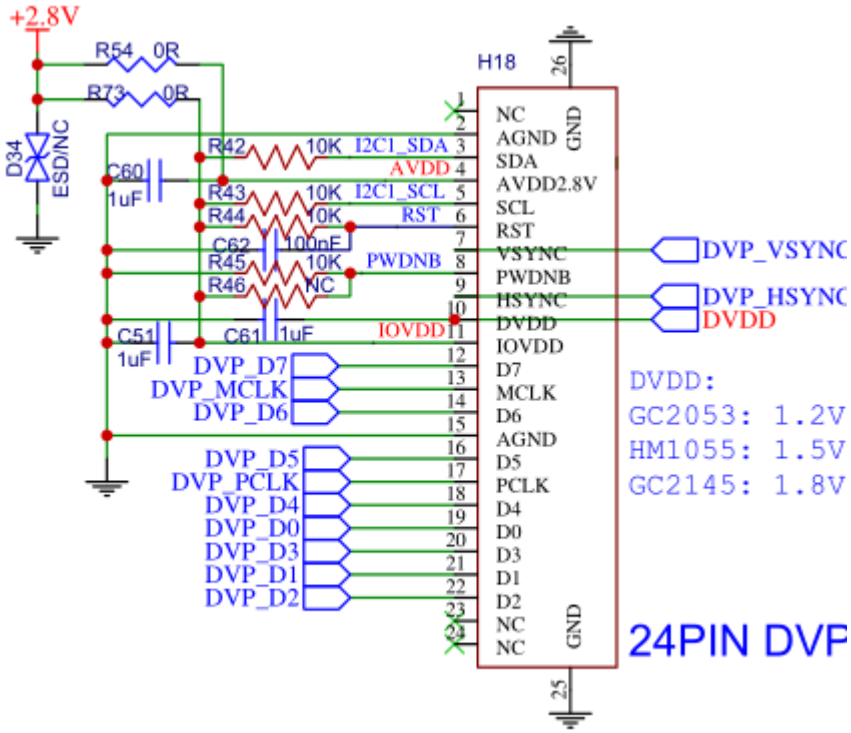


Fig. 4: Figure 4.DVP interface

IOVDD: The analog voltage of the GPIO port of the camera mainly supplies power to the IIC or DVP part.

DVDD: The digital operating voltage of the camera, if the power supply is unstable, it may cause the screen to burn.

6.2 DVP Camera Introduction

6.2.1 1 Introduction to DVP

The current DVP camera is based on the CIS interface and only supports 20/24Pin 8bit data output. The internal register model of the sensor is configured through the I2C protocol to achieve the required output resolution, frame rate, exposure and other performance adjustments. In addition, for the output data of the sensor, it is necessary to use the hardware module for further editing. For example, the YUV_BUF module is used to cut and convert the YUV422 data output of the sensor into YUV420, and then the H264 module is used for encoding to reduce the use of memory space and perform more functions. Or directly for YUV422 data using JPEG module encoding, output is more stable and memory space use smaller image.

6.2.2 2 Types and specifications supported by DVP

Supported peripherals, please refer to Support Peripherals

6.2.3 3 DVP use process

- 1.Power on and clock configure each multimedia hardware module, including YUV_BUF/JPEG/H264 module, and supply power to the sensor;
- 2.Initialize I2C, because I2C needs to be used to configure the DVP register and specify the write/read address of the DVP
- 3.configure the DVP GPIO second function, so that the output can be detected in real time
- 4.Configure the input clock of the sensor. There are two sources of the clock: clock division for YUV_BUF and clock division for AUXS
- 5.Configure the register value of YUV_BUF/JPEG/H264 according to the specific application, and configure DMA to process the data, and finally enable related modules;
- 6.Configure the register value of the sensor through I2C to achieve the ideal output data;

6.2.4 4 Main applications of DVP working mode

DVP cameras can rely on different hardware modules to meet different needs, and several main applications are listed below.

1.DVP works in YUV (or GRAY) mode

2.DVP works in JPEG(or JPEG&YUV) mode

3.DVP works in H264(or h264&YUV) mode

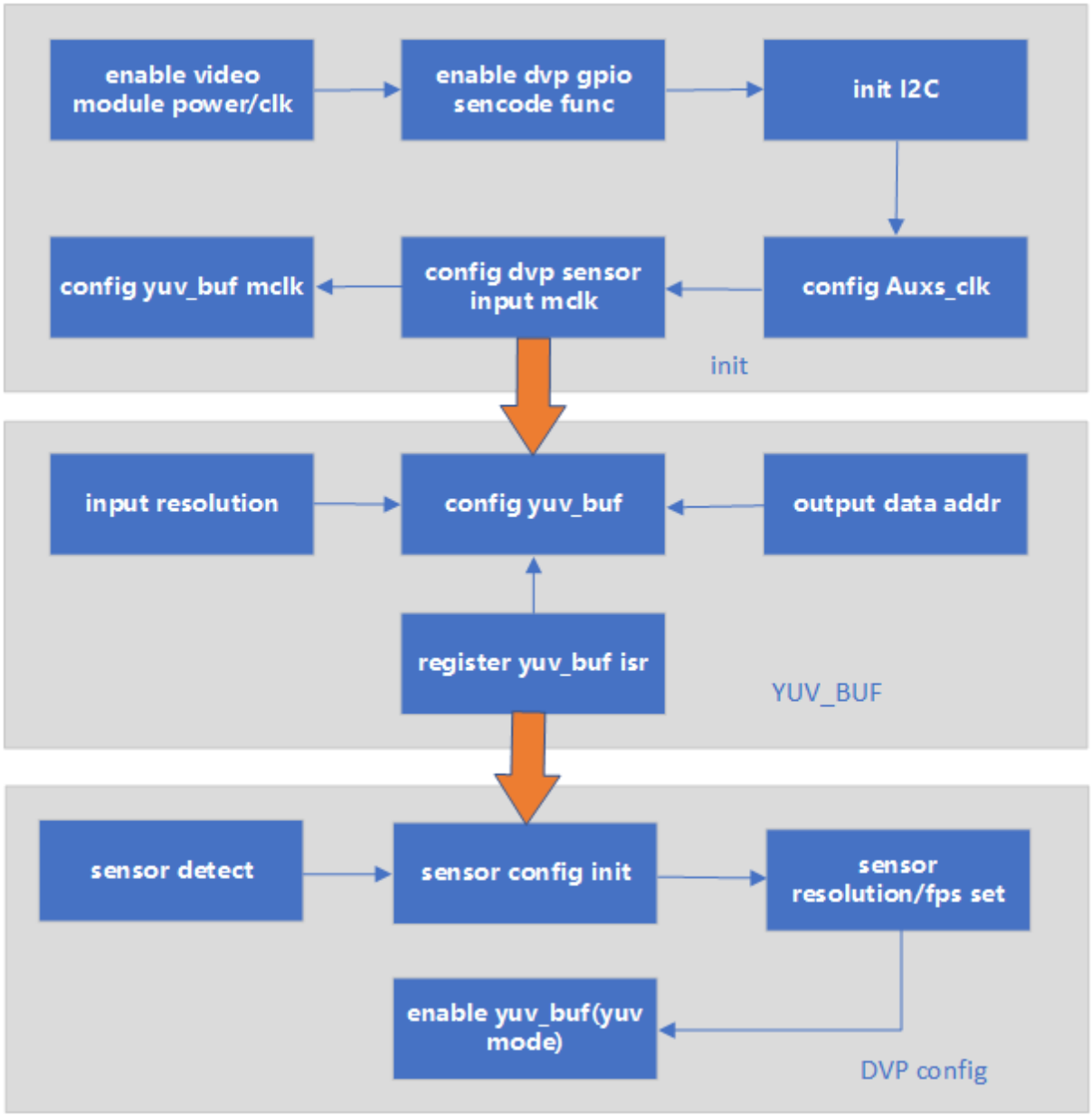


Fig. 5: Figure 1.DVP works in YUV (or GRAY) mode

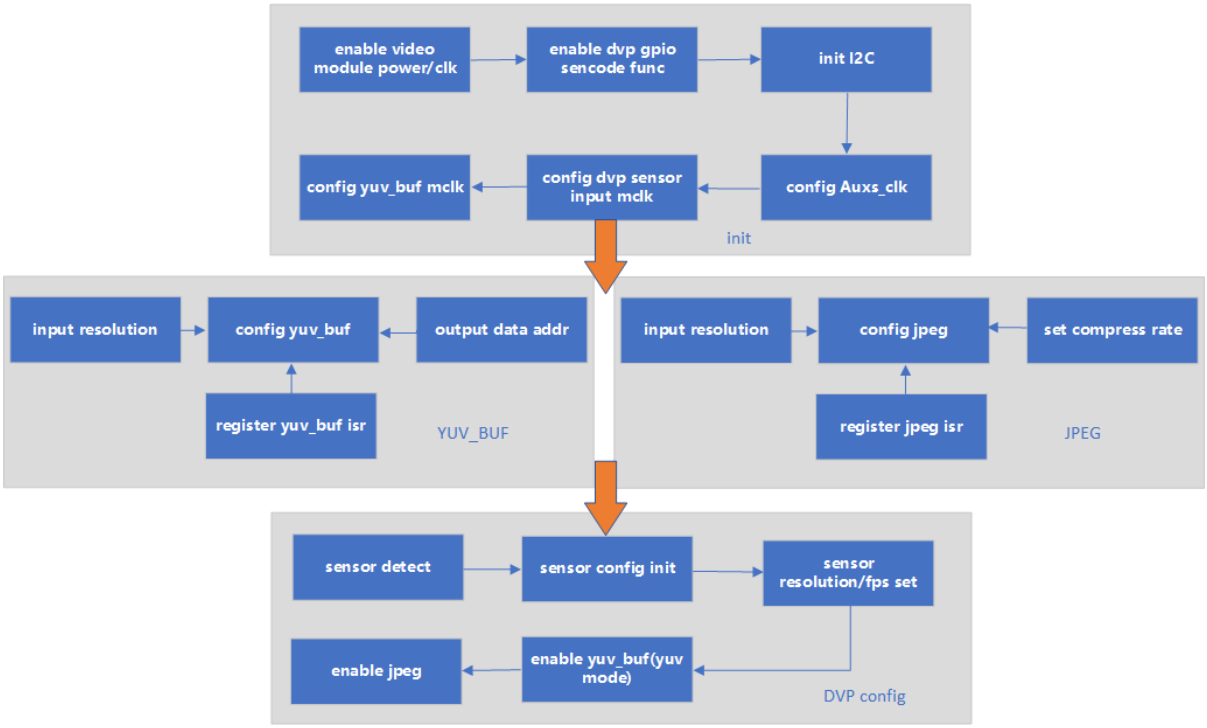


Fig. 6: Figure 2.DVP works in JPEG mode

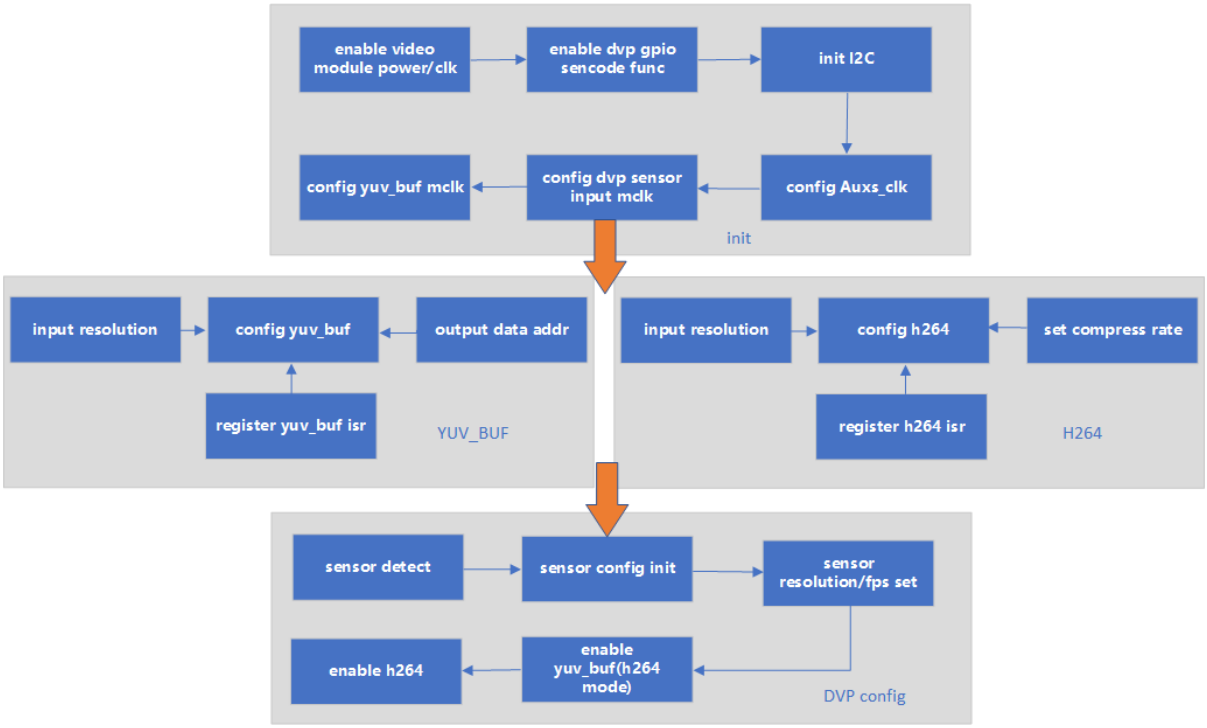


Fig. 7: Figure 3.DVP works in H264 mode

6.3 UVC Camera Introduction

6.3.1 1 Introduction to UVC

USB Video Control (UVC) is a USB device class specification in the USB industry specification, and a unified data exchange specification for video devices with USB interfaces. In the UVC specification, it is clearly required to have two interfaces: VC Interface (video control class Interface) and the VS Interface (video stream interface). The VC interface is responsible for the configuration of UVC equipment, and the VS interface is responsible for the transmission of video stream data, and the two cooperate with each other to complete the functions of UVC equipment. The current hardware supports USB2.0, including Full-speed and High-speed. In addition to the 1.5Mbps and 12Mbps transmission modes specified in USB1.1, USB2.0 also adds a high-speed transmission mode of 480Mbps. The USB1.1 unit data transfer time is 1 millisecond, while the USB2.0 unit data transfer time is 125 microseconds. The UVC output data format is determined by its own firmware, which may support H264, MJPEG, H265, etc.

6.3.2 2 Types and specifications supported by UVC

USB specification	UVC specification	Transmission	transmission speed	MaxPacketSize	resolution	fps	output format
USB2.0	UVC1.5/UVC1.0	ISO/BULK	High(Full)-speed	1024 and below	1280X720 and below	30 and below	JPEG/H264/H265

6.3.3 3 UVC Usage process

- 1.initialize the USB hardware module, that is, initialize the host, and start its work usb_task;
- 2.host Detects whether the device is connected and enumerates.
- 3.After detecting that the connection is successful, the upper layer reads the device descriptor and sets UVC parameters based on it
- 4.Start the UVC
- 5.Process UVC packets
- 6.Pause UVC transmission
- 7.restore UVC transmission, etc., do a good job of state machine switching

CHAPTER

SEVEN

PSRAM MEM CONFIG

7.1 1. Function overview

PSRAM provides enough memory space for different modules to use, the current BK7258 supports two types of PSRAM, their memory size is different, respectively, 8Mbyte and 16Mbyte, using different macros to distinguish. There are four types of modules in multimedia that need to use PSRAM. Since PSRAM is also used as heap on different cpus, not all of the PSRAM space is used by multimedia, and the length and space need to be distinguished.

7.2 2. PSRAM Type

The BK7258 supports two types of PSRAM, which are configured as follows:

name	size(Mbyte)
APS6408L_O	8
APS128XXO_OB9	16

Note: Different projects use different psram types by default, you need to see the compiled sdkconfig file.

7.3 3. Division of PSRAM

PSRAM is mainly divided into two parts, one part is used as data memory, the other part is used as heap space of different cpus, you need to refer to the following macro definition to identify the size of their use:

marco	value	implication
CONFIG_PSRAM	Y	enable PSRAM module
CONFIG_PSRAM_AS_SYS_MEMORY	Y	enable PSRAM as Heap
CONFIG_USE_PSRAM_HEAP_AT_SRAM_OOM	Y	enable alloc from sram fail, then alloc from psram
CONFIG_PSRAM_HEAP_BASE	0x60700000	the start address of psram as heap in current cpu
CONFIG_PSRAM_HEAP_SIZE	0x80000	the size psram as heap in cluurent cpu
CONFIG_PSRAM_HEAP_CPU0_BASE_ADDR	0x60700000	the start address of psram as heap in this chip

- Function switch CONFIG_PSRAM_AS_SYS_MEMORY depends on CONFIG_PSRAM and CONFIG_FREERTOS_V10

Note: The above values are defined differently on different cores, CONFIG_PSRAM_HEAP_CPU0_BASE_ADDER indicates that PSRAM serves as the heap starting address, and that psram serves as the heap space from this address. By default, the starting address of PSRAM (0x60000000) through CONFIG_PSRAM_HEAP_CPU0_BASE_ADDER is allocated for multimedia use, and the rest is allocated for Heap use. Use the macro CONFIG_PSRAM_HEAP_BASE Define PSRAM on the current kernel as the Heap start address, and use CONFIG_PSRAM_HEAP_SIZE to define the space size. Different cores have different defined values, CONFIG_PSRAM_HEAP_CPU0_BASE_ADDER has the same value on different cores. This should be consistent with the minimum value of CONFIG_PSRAM_HEAP_BASE on different cores, such as the default BK7258 project, CONFIG_PSRAM_HEAP_BASE=0x60700000 on CPU0, CONFIG_PSRAM_HEAP_BASE=0x60700000, CONFIG_PSRAM_HEAP_SIZE=0x80000, then CPU1 CONFIG_PSRAM_HEAP_BASE=0x60780000, CONFIG_PSRAM_HEAP_SIZE=0x80000, PSRAM is not used as Heap by default on CPU2. So CONFIG_PSRAM_HEAP_CPU0_BASE_ADDER=0x60700000. Whether psram is required as heap on a kernel is controlled by CONFIG_PSRAM_AS_SYS_MEMORY.

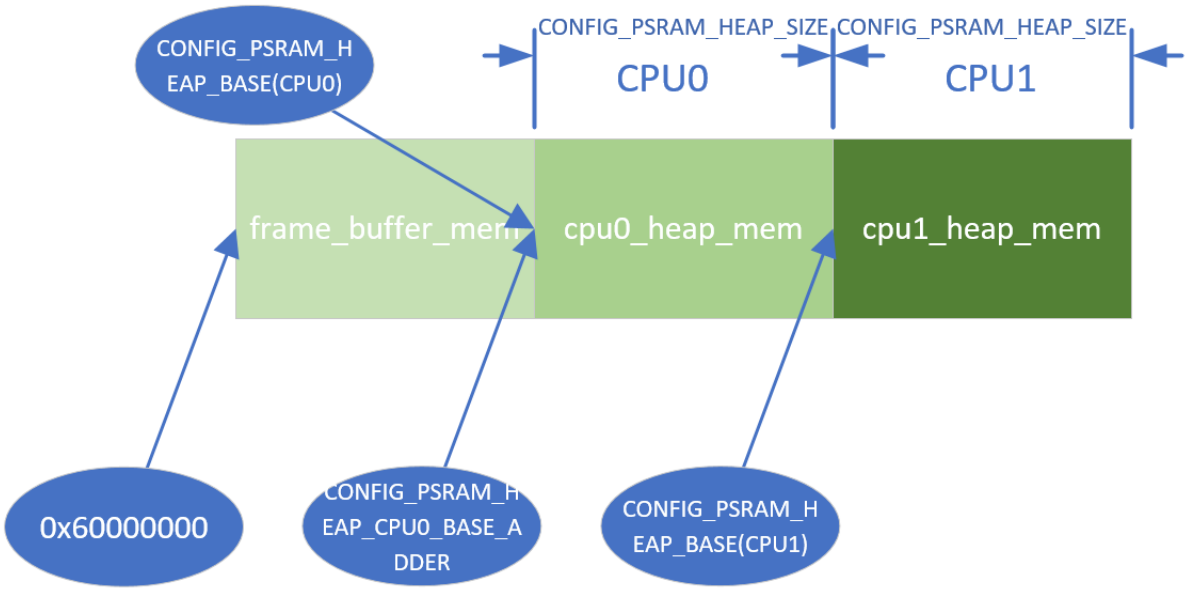


Fig. 1: Figure 1. psram architecture

7.4 4. PSRAM used for heap

- Configure the starting address and size of psram heap through the following two configuration items:

```
CONFIG_PSRAM_HEAP_BASE configures the starting address of psram heap
CONFIG_PSRAM_HEAP_SIZE configures the starting address of psram size
```

- Memory application and release interface:

```
void *psram_malloc(size_t size);
void  psram_free(void *p);
```

- The low-voltage function PARAM of the BK7258 chip will be powered off. If the psram memory used by the upper layer is not released, it will result in the inability to enter low voltage. You can print the current psram memory usage through the following interface:

```
void bk_psram_heap_get_used_state(void);
```

- The API of create task on psram:

```
bk_err_t rtos_create_psram_thread(beken_thread_t *thread, uint8_t␣
↪priority, const char *name, beken_thread_function_t function, uint32_
↪t stack_size, beken_thread_arg_t arg);
bk_err_t rtos_delete_thread(beken_thread_t *thread); // The API of␣
↪delete task is not change
```

7.5 5. PSRAM in the multimedia division

PSRAM is used in multimedia and is divided into four modules according to function, reference structure: *psram_heap_type_t*, position: *bk_idk/include/driver/psram_types.h*.

The memory used by each module is defined by the structure: *psram_mem_slab_mapping*. The amount of memory allocated by each module is controlled by macros, reference position: *bk_avdk_main/components/media_utils*, as follows:

marco	value(byte)	implication
CONFIG_PSRAM_MEM_SLAB_USER_SIZE	102400	the size alloc to user
CONFIG_PSRAM_MEM_SLAB_AUDIO_SIZE	102400	the size alloc to audio
CONFIG_PSRAM_MEM_SLAB_ENCODE_SIZE	1433600	the size alloc to encode(jpeg/h264)
CONFIG_PSRAM_MEM_SLAB_DISPLAY_SIZE	5701632	the size alloc to display

Note: The value defined by the above macro is defined by default, which can be dynamically adjusted according to its own needs when used, and can be modified directly in the cpu config of the corresponding project. However, note that the length added up above cannot exceed the address used by the Heap(CONFIG_PSRAM_HEAP_CPU0_BASE_ADDER), otherwise there will be problems.

7.6 6. Each module of multimedia memory adjustment

According to the previous section, psram is divided into four modules, different modules store different types of data, as follows:

- UASER: allocated to users. The allocated size is defined by the macro CONFIG_PSRAM_MEM_SLAB_USER_SIZE.
- AUDIO: allocated to audio. The allocated size is defined by the macro CONFIG_PSRAM_MEM_SLAB_AUDIO_SIZE. It stores audio data;
- ENCODE: allocated to encoding, the allocated size is defined by the macro CONFIG_PSRAM_MEM_SLAB_ENCODE_SIZE, which stores complete JPEG images or H264 images;
- DISPLAY: allocated to the display. The allocated size is defined by the macro CONFIG_PSRAM_MEM_SLAB_DISPLAY_SIZE, which stores the displayed data type, such as YUV, RGB565, RGB888, etc.

The amount of data stored varies according to the function and size of the module above. For example, the ENCODE module can store more than one frame of JPEG image or H264 image. The size of the system also defines a frame of macros, reference files: `./bk_idk/middleware/driver/camera/Kconfig`:

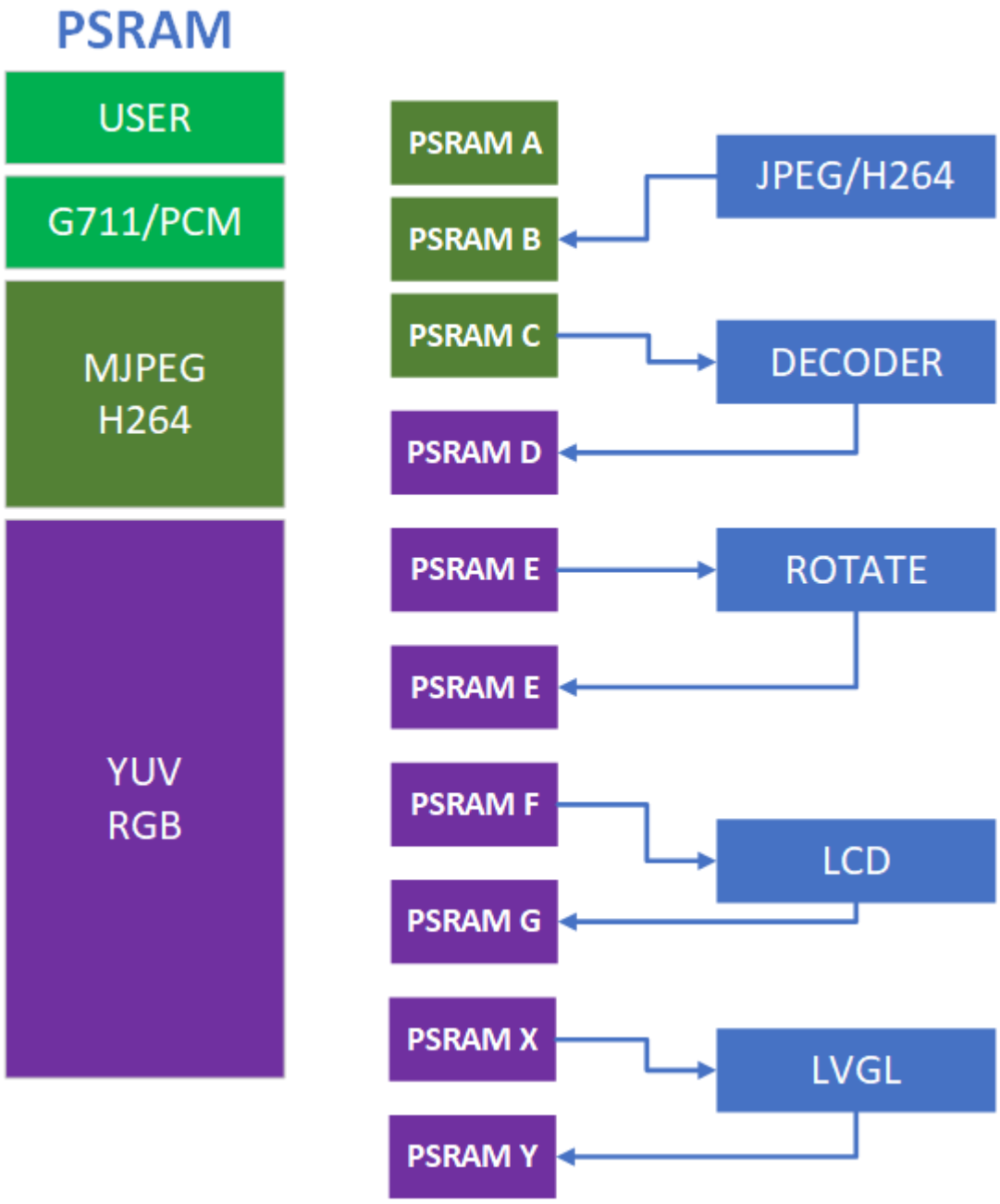


Fig. 2: Figure 2. psram multimedia architecture

marco	value(byte)	implication	range
CONFIG_JPEG_FRAME_SIZE	153600	the size of one complete jpeg frame	[0, 204800]
CONFIG_H264_FRAME_SIZE	65536	the size of one complete h264 frame	[0, 204800]

The above size needs to be adjusted according to their own needs, such as the need to store 1280X720 JPEG images, 150K space may not be enough, need to be changed to 200K(204800), or even larger, according to the actual use of adjustment. Also for H264 data, sometimes need to adjust the compression rate of H264, in order to achieve a clearer picture quality, the default 64K May not be enough, need to continue to increase, so also need to adjust according to the actual situation.

According to the size defined above, the number of different block storage can be calculated. Assuming the RGB565 used by the DISPLAY module, and the resolution is 800X480, then the size of an image is 800*480*2=768000. The number that can be stored is: CONFIG_PSRAM_MEM_SLAB_DISPLAY_SIZE/768000=7, which means that the maximum storage capacity is 7 frames of 800X480 RGB565 images.

Assuming the ENCODE module is used to store JPEG images, the maximum number of stores is: CONFIG_PSRAM_MEM_SLAB_ENCODE_SIZE/CONFIG_JPEG_FRAME_SIZE=9; However, the actual situation will store both JPEG and H264 data, which is defined in the code the largest number, each image module reference: `./bk_avdk_main/components/multimedia/comm/frame_buffer.c`, definition in the following statement:

```
uint8_t fb_count[FB_INDEX_MAX] = {5, 4, H264_GOP_FRAME_CNT * 2};
```

It means that the maximum storage is 5 frames of DISPLAY data, 4 frames of JPEG data, H264_GOP_FRAME_CNT*2 frames of H264 data. The above quantity can be adjusted, as long as the total amount of data does not exceed the size of the respective module.

7.7 7. Using psram on multimedia

Because all multimedia functions are used in CPU1, the use of PSRAM can only be directly invoked in CPU1. After CPU1 is started, the system will automatically initialize the entire PSRAM for the multimedia, and users do not need to invoke the implementation themselves. When CPU1 is powered down, multimedia do not use PSRAM, and no additional call to the logged out interface is required to free up the corresponding memory.

- Memory initialization interface

```
bk_psram_frame_buffer_init
```

- Memory request and release interface

```
void *bk_psram_frame_buffer_malloc(psram_heap_type_t type, uint32_t size);
void bk_psram_frame_buffer_free(void* mem_ptr);
```

Note: When used by customers, it is recommended to use the system interface to apply for and release psram memory (psram_mallocpsram_free), and it is not recommended to use the above multimedia module defined interface to apply for and release psram memory.

CHAPTER

EIGHT

VIDEO CODEC

8.1 H264 Encoding

8.1.1 1. Function overview

h264 coding is mainly used to encode YUV422 data through hardware, output h264 image data, and compress image data

8.1.2 2. Develop materials

Currently avdk provides a detailed sdk interface, refer to the file path:.*bk_idkmiddlewaredriverh264*

In order to facilitate the use of customers, re-packaging the coding function, the current only need to switch the camera, and then for the image quality, also provide an api interface to adjust, reference file:.componentsmultimediaappmedia_app.c ‘

8.1.3 3. Interface description

Debugging description based on h264 image quality: media_app_set_compression_ratio

Structure parameter description: `compress_ratio_t`

- yuv_mode_t mode: The current mode to be adjusted. Only JPEG and H264 are supported. Select H264
- h264_qp_t h264_qp: The qp value of h264 encode value
- uint8_t enable: enables compression adjustment, which is supported only in JPEG mode
- uint16_t jpeg_up: Upper limit of data JPEG image size, in byte, valid only in JPEG mode
- uint16_t jpeg_low: Lower limit of data JPEG image size, in byte, valid only in JPEG mode
- uint16_t imb_bits: The size of the I frame macro block encoded by H264, in byte. This parameter is valid only in H264 mode. The larger the value, the smaller the compression rate, the higher the image quality and the larger the encoded output image, range [1, 4095].
- uint16_t pmb_bits: The size of the H264 output P-frame macro block, in byte, is valid only in H264 mode. The larger the value, the smaller the compression rate, the higher the image quality, and the larger the encoded output image, range [1, 4095].

Structure parameter description: `h264_qp_t`

- uint8_t init_qp; The init qp value,range [0, 51];

- uint8_t i_min_qp; The minimum qp of I frame encode, range [0, 51];
- uint8_t i_max_qp; The maximum qp of I frame encode, range [0, 51];
- uint8_t p_min_qp; The minimum qp of P frame encode, range [0, 51];
- uint8_t p_max_qp; The maximum qp of P frame encode, range [0, 51];

Note: In addition to paying attention to the valid range of the above values, it is also necessary to pay attention to the maximum value of the corresponding encoding must be greater than the minimum value. The image quality can be adjusted through the above interface. It should be noted that improving the image quality will undoubtedly increase the amount of data. At present, the maximum output space of a frame of yuv422 data encoding is 64KB. The encoded H264 image output is larger than 64K, and there will be instability.

8.1.4 4. Code gear adjust

The default SDK provides the configuration of h264 encoded gear adjustment, which is defined by the configuration macro: `CONFIG_H264_QUALITY_LEVEL`, which is described as follows:

- Value range: [0, 3], define three gears, the values are 1/2/3, corresponding to the h264 compressed image quality from low to high, the clearer the image is.
- If CONFIG_H264_QUALITY_LEVEL=0, the default value is used instead of the three-gear parameter. The default value reference path: `.\bk_idk\middleware\soc\bk7258\hal\h264_default_config.h`, you can change the default value to achieve the desired effect.
- The default SDK gear is defined in the middle gear, CONFIG_H264_QUALITY_LEVEL=2. You can change this value in project config.

8.1.5 5.Adjustment of P-Frame Count

Under the default SDK configuration, the number of P-frames per GOP in H264 encoding is 5, which can be adjusted through the macro: `CONFIG_H264_P_FRAME_CNT`. The valid range is [0, 1023]. Increasing the number of P-frames can lead to a reduction in the overall bitrate of the encoded output.

8.1.6 6.Suggestions for Adjusting Image Quality

The SDK provides three predefined H264 image quality levels, which can be controlled through the macro: `CONFIG_H264_QUALITY_LEVEL`. Adjusting image quality primarily involves parameters within the structure of: `compress_ratio_t`.

Debugging Steps:

1.Adjust the number of P-frames: Increase the number of P-frames based on the variation in image content to reduce bitrate.

2.Adjust the frame buffer size: The default is 64K. Adjust it as needed to prevent I-frames from being too large and causing the output to fail. It is recommended to set the CONFIG_H264_FRAME_SIZE to 102400 (100K).

3.Configure the H264 image quality level: Use CONFIG_H264_QUALITY_LEVEL=0 to use the SDK's original encoding parameters.

4.Real-time compression rate adjustment: Monitor image quality and bitrate simultaneously, and use a controlled variable method to change one parameter at a time.

5.Obtain/configure encoding parameters: Use the command-line tool to get encoding parameters. To obtain: media h264 get_config, and to configure: media compress h264 init_qp iframe_max_qp pframe_max_qp num_ibits num_pbits.

6.Adjust the initial quantization parameter (init_qp): Start from 1 and increase by 5 each time, with a maximum value of no more than 51 until the image does not show obvious macroblock motion.

7.Adjust the I-frame encoding parameter (num_ibits): Increase this value by 20 each time until the image quality is satisfactory, but be cautious not to increase it too much to avoid increased bitrate, with a suggested maximum value of no more than 200.

8.Adjust the P-frame encoding parameter (num_pbits): Adjust this similarly, but be cautious not to increase it too much to avoid increased bitrate; a suggested maximum value is no more than 150.

9.Repeat steps 7-8: Continue adjusting until the best parameters are found.

10.Adjust the maximum quantization parameters for I-frames and P-frames (iframe_max_qp and pframe_max_qp): Reducing these values can improve image quality, but will also increase bitrate. These should not be set lower than 32.

11.Adjust the frame buffer size: After achieving the final effect, if the I-frame size does not exceed 64K, you can revert it to the default value CONFIG_H264_FRAME_SIZE=65536 (64K).

Through these steps, you can optimize the number of P-frames and image quality in the H264 encoding process to meet specific application requirements.

8.2 H264 Decoding

8.2.1 1.H264 Decoding

The H264 decoding part is implemented on the mobile phone. This article uses the ffmpeg library for decoding

8.2.2 2.ffmpeg Official Website

https://ffmpeg.org/

8.2.3 3.Decoding Process Diagram

8.2.4 4.Process Explanation

1. Register Decoder:`avcodec_register_all`

   Used to register all compiled codecs.

2. Find Decoder:`avcodec_find_decoder`

   Searches for the decoder corresponding to the specified decoder ID. The decoder ID for H.264 is: AV_CODEC_ID_H264.

3. Initialize Parser:`av_parser_init`

   Initializes the parser context. The parser can be used for preprocessing input data before processing the data stream, such as finding frame boundaries, extracting key information, etc.

4. Allocate Decoder Context:`avcodec_alloc_context3`

   Allocates an AVCodecContext structure and sets default values. AVCodecContext is a data structure in ffmpeg used to store the context information of the codec.

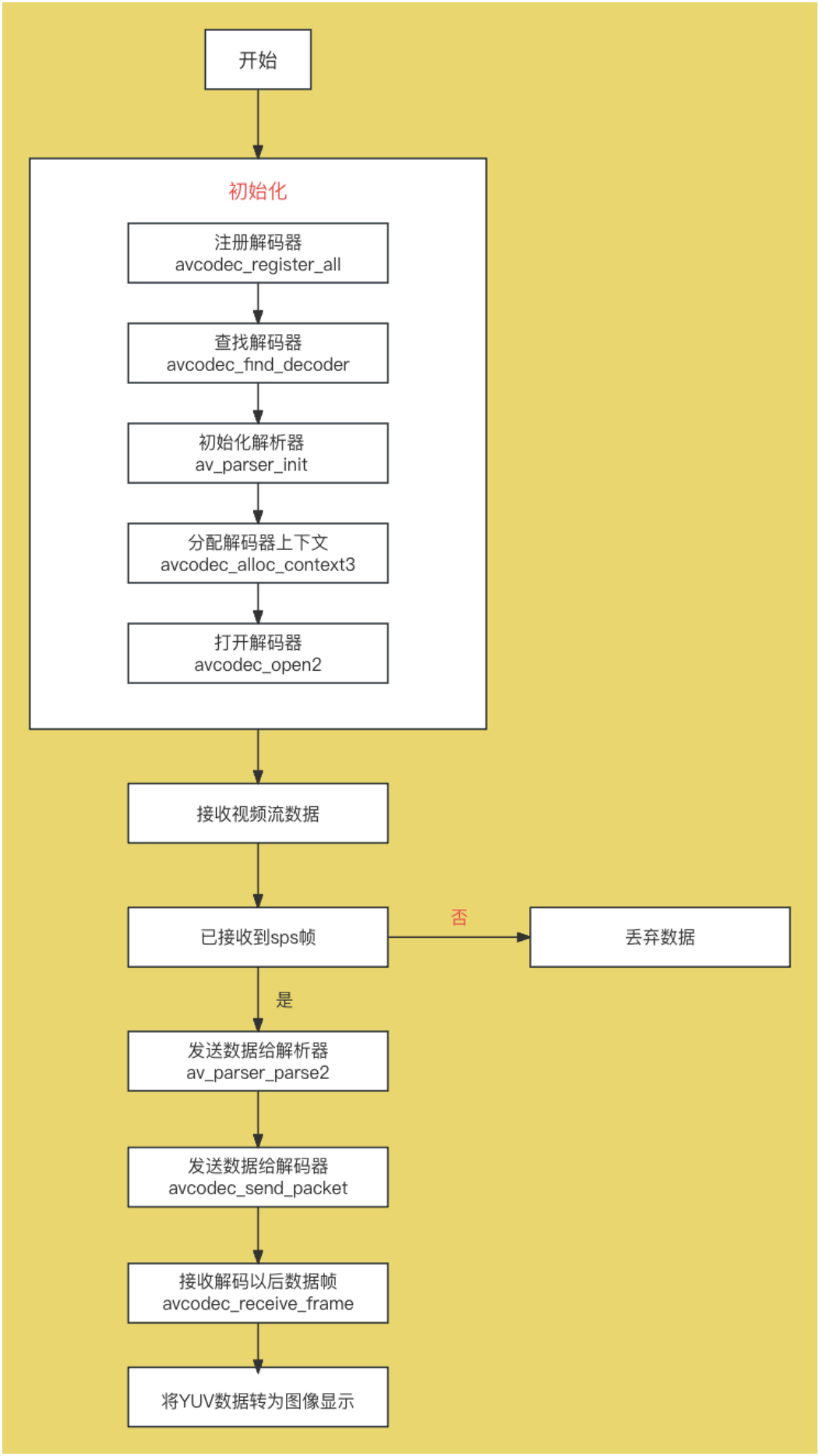


Fig. 1: Figure 1. decode_flow

5. Open Decoder:`avcodec_open2`

   Opens the decoder and initializes its context (AVCodecContext). After initializing the decoder context, this function opens the decoder to start the decoding process.

6. Receive Data

   Before starting decoding each time, check if the data starts with an sps frame. Only start decoding when the data with sps beginning is found; otherwise, discard the data without decoding.

7. Send Data to Parser:`av_parser_parse2`

   Parses the input media data stream, usually used to find frame boundaries.

8. Send Data to Decoder:`avcodec_send_packet`

   Used to send data packets to the decoder for decoding. It sends the input data packet to the decoder context for decoding.

9. Receive Decoded Data Frames:`avcodec_receive_frame`

   Used to receive the decoded frames from the decoder. After using the `avcodec_send_packet` function to send data packets for decoding, `avcodec_receive_frame` can be used to obtain the decoded image frames from the decoder.

8.3 JPEG Encoding

8.3.1 1. Function overview

JPEG encoding is mainly used to encode YUV422 data through hardware, output JPEG image data, and compress image data. At present, JPEG function is mainly used in DVP camera, DVP camera output YUV422 data to

Chip, chip compression into JPEG image. It also supports compression of external YUV422 data into JPEG images. Since BK7258 has a module, YUV_BUF, that deals specifically with YUV data, the JPEG module also needs to rely on it.

8.3.2 2. Develop materials

Currently avdk provides a detailed sdk interface, reference file path:`.\bk_idk\include\driver\jpeg_enc.h`, `.\bk_idk\include\driver\jpeg_enc.h`.

In order to facilitate the use of customers, re-packaging the coding function, the current only need to switch the camera, and then for the image quality, also provide an api interface to adjust, reference file:`.\components\multimedia\app\media_app.c`.

8.3.3 3. Interface description

BK7258 supports the encoding of DVP camera data into JPEG, H264 and synchronous output of YUV422 data for LCD display.

- Open DVP camera interface:

```
bk_err_t media_app_camera_open(media_camera_device_t *device);
```

- Close DVP camera interface:

```
bk_err_t media_app_camera_close(camera_type_t type);
```

You need to pay attention to the data structure passed into the parameter

```
typedef enum
{
        UNKNOW_CAMERA,
        DVP_CAMERA,     /**< dvp camera */
        UVC_CAMERA,     /**< uvc camera */
        NET_CAMERA,     /**< net camera, use dual chip transfer each other*/
} camera_type_t;

typedef enum
{
        UNKNOW_MODE = 0,
        YUV_MODE,      /**< output yuv data */
        GRAY_MODE,     /**< output gray data */
        JPEG_MODE,     /**< output jpeg data */
        H264_MODE,     /**< output h264 data */
        H265_MODE,     /**< output h265 data */
        JPEG_YUV_MODE,/**< output jpeg & yuv data */
        H264_YUV_MODE,/**< output h264 & yuv data */
} yuv_mode_t;

typedef struct {
        camera_type_t  type; /**< camera type */
        yuv_mode_t     mode; /**< work mode */
        pixel_format_t fmt;  /**< camera output data format */
        frame_info_t   info; /**< camera output resolution and fps */
} media_camera_device_t;
```

According to the above data structure, the corresponding parameters are configured and passed into the interface to achieve the corresponding expected result. For example, the current use of DVP camera, then the camera type is DVP_CAMERA, if you want to output JPEG and YUV data, Then the yuv mode is set to JPEG_YUV_MODE, or if only JPEG images are needed, the yuv mode is set to YUV_MODE.

- Interface for adjusting compression rate

```
bk_err_t media_app_set_compression_ratio(compress_ratio_t *ratio);
```

According to the data structure of the passed parameters, the compression rate adjustment is supported in both JPEG and H264, and this interface supports calling during the encoding process.

```
typedef struct {
        yuv_mode_t mode;   /**< The mode to be adjusted. Only JPEG_MODE and H264_MODE␣
↪*/ are supported
        h264_qp_t qp;      /**< The initial qp value of the h264 encoding, the␣
↪smaller the qp value, the larger the compression rate */
        uint8_t  enable;   /**< Enable encoding adjustment */
        uint16_t jpeg_up;  /**< Adjust the upper limit of jpeg compression, target␣
↪image output size, in byte */
        uint16_t jpeg_low; /**< Adjust the upper limit of jpeg compression, target␣
↪image output size, in byte */
        uint16_t imb_bits; /**< h264 encoding, upper limit of target I frame output␣
↪size, unit byte */
        uint16_t pmb_bits; /**< h264 encoding, upper limit of the output size of the␣
↪target P-frame, unit byte */
} compress_ratio_t;
```

According to the above data structure, the corresponding parameters are configured and passed into the interface to achieve the corresponding expected result. For example, adjust the JPEG encoding compression rate, mode set to JPEG_MODE, jpeg encoding process, when output a frame. If the size of the image exceeds jpeg_up, it is further compressed so that the amount of data is no more than jpeg_up and no less than jpeg_low.

8.3.4 4. Driver code

DVP camera driver code path:`.\bk_idk\middleware\camera\dvp_camera.c`.

Underlying driver ocde path:`.\bk_idk\middleware\jpeg`, `.\bk_idk\middleware\yuv_buf`.

8.4 JPEG Decode SW

8.4.1 1. JPEG Soft Decoding Process

The main process of JPEG soft decoding is as follows:

1) Pre decoding, parsing the header of a JPEG data to obtain image format, image length and width, quantization table, and Huffman table;
2) Search for quantization tables and Huffman tables to parse a block of data;
3) Convert a block data into the image format that needs to be output;
4) Write the converted image data into the output image;
5) Repeat 2-4 until an exception occurs or decoding is complete;

8.4.2 2. Memory requirements

JPEG soft decoding relies on threads within 1K, and the main buffers and purposes required are as follows:

1) A 10240 buffer for storing Huffman tables and intermediate data processing;
2) A buffer of size 0xB0, used to store intermediate running pointers and variables;
3) A 16 * 16 * 2 sized buffer for image rotation after soft decoding;

8.4.3 3. Use of JPEG soft decoding module

1) Call *bk_jpeg_dec_sw_init* to initialize the soft decoding module;
2) Call *bk_jpeg_dec_sw_start* to decode the incoming image;
3) Call *bk_jpeg_dec_sw_deinit* to release the internal buffer of soft decoding;

Please refer to API Reference : for function parameters.

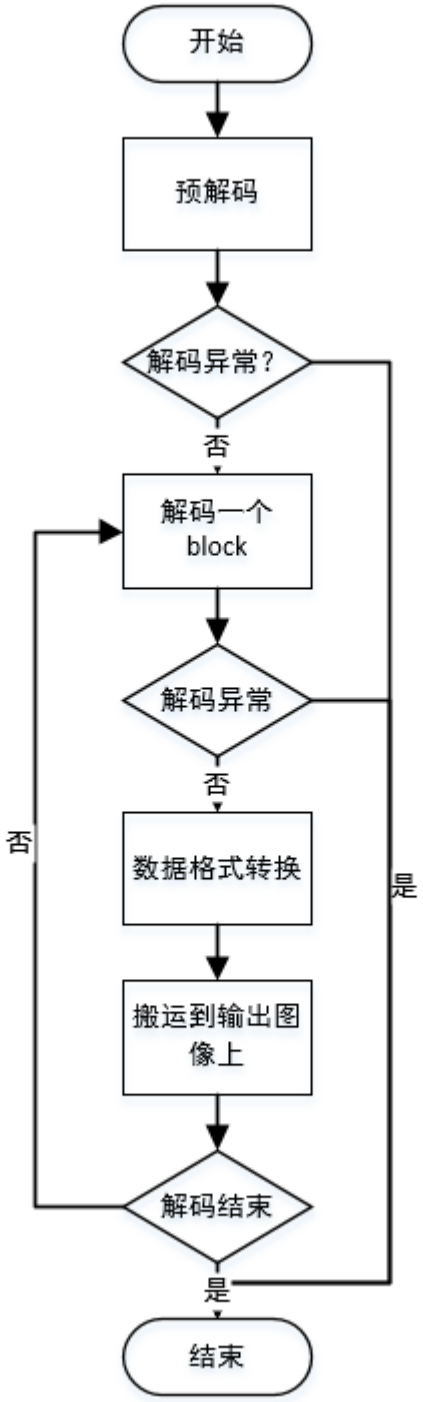


Fig. 2: Figure 1. jpeg soft decode process

8.4.4 4. One time decoding function

For some scenarios, a one-time decoding interface *bk_jpeg_dec_sw_start_one_time* is provided, Before using this interface, there is no need to call *bk_jpeg_dec_sw_init*, and the buffer is passed in externally;

If NULL is passed in externally, it will be automatically requested by the interface internally; Automatically release the internal requested buffer after decoding is completed;

Note: This interface can be decoded in parallel with the soft decoding module, but the decoding speed will be reduced when parallel;

8.4.5 5. Decoding speed optimization

When DTCM is sufficient, *CONFIG_SOFTWARE_DECODE_SRAM_MAPPING* can be set to y on CP1 and CP2 to speed up decoding; The optimization method is to put the decoding thread and part of the buffer used for decoding on dtcm, and at the same time, change the original decoding block and save it on psram to buffer 16 lines and then save them together on psram, reducing the time of frequent psram address switching in actual operation;

Note: The soft decoding and rotation modules cannot work at the same time because the buffer used to buffer 16 lines in soft decoding is the buffer used in the rotation module;

8.5 JPEG ENCODE SW

8.5.1 1.JPEG ENCODE SW FLOW

The JPEG software encoding flow is as follows:

1) Input data format convert(This step is optional:it is necessary to convert input data format if it is not YVYU422);
2) Initialization /apply inner encoding buffer/image output buffer;
3) Fill the JPEG header;
4) excute DCT/quantization/huffman coding with 16x8 block size,fill the compressed bits to output buffer;
5) repeat step 4 until error or end of encoding;

8.5.2 2.Memory requirement

JPEG software encoding thread need less than 3K bytes stack,other buffers and description of them is as follows:

1) one 512bytes inner buffer to save 16x8 block output compressed bits;
2) one 64k bytes buffer as image output buffer(This buffer size is modifiable according to the requirement)

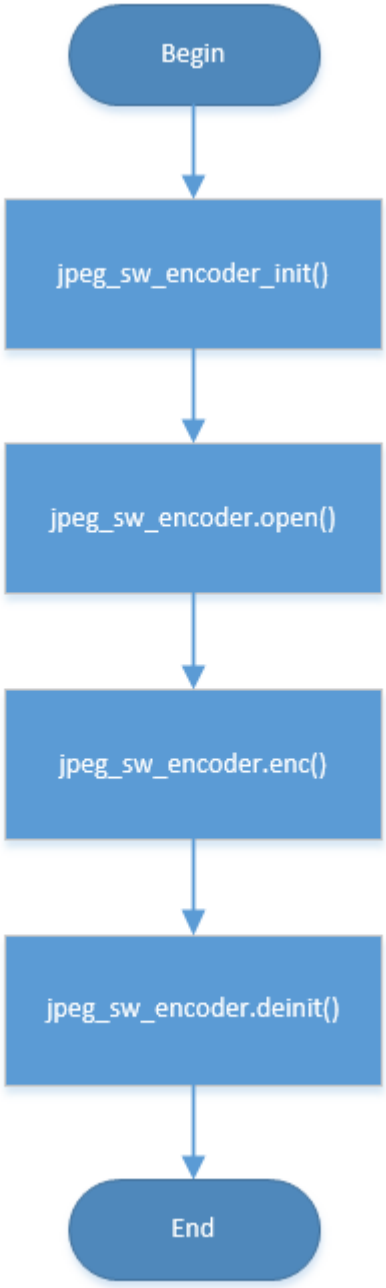


Fig. 3: Figure 1. jpeg soft encode process

8.5.3 3.JPEG softeware API usage

1) call *jpeg_sw_encoder_init* ,initialize JPEG software encoding module/register callback functions like:open/enc/reset/deinit;
2) call *jpeg_sw_encoder.open* ,initialize codec inner parameters /apply inner buffer;
3) call *jpeg_sw_encoder.enc* ,excute JPEG software encoding;
4) call *jpeg_sw_encoder.deinit* ,release inner buffer/close JPEG software encoder;

API Reference API Reference :

8.5.4 4.JPEG SW encoding example

See jpeg_sw_enc_test.c for details

CHAPTER

NINE

GUI

9.1 LVGL

9.1.1 1. Function overview

LVGL(Light and Versatile Graphics Library)is a free, open source and low-cost embedded graphics library with rich and powerful modular graphics components. It can be used to create beautiful interface for any MCU, MPU and display type and supports multiple input devices. It has become a popular graphics user interface library. LVGL official website address is https://lvgl.io.

9.1.2 2. Reference project

Currently avdk provides LVGL demo reference projects of various types and purposes. The code path of the reference project is `\projects\lvgl\...` and the compilation command is `make bk7258 PROJECT=lvgl/xxx`. For specific project introduction, see Reference projects

9.1.3 3. Development steps

Since BK7258 multimedia is executed on CPU1, the execution of lvgl needs to start CPU1 on CPU0 first, then send a mailbox message to CPU1 to execute the relevant code, and finally return to the execution state after the execution is completed on CPU1. This part of the execution code has been encapsulated into an independent interface. Please follow the following steps:

- Set the value of the macro `CONFIG_LV_COLOR_DEPTH` according to the data type of the LCD, and open the corresponding macro configuration of the functions of the functions provided by LVGL as required.
- Call the interface of `media_app_lvgl_open()` on CPU0 to start CPU1 and send a message to execute LVGL event.
- Define the inferface of `void lvgl_event_handle(media_mailbox_msg_t *msg)` on CPU1 and execute the corresponding open or close code according to the `msg->event` parameter.
- Obtain the LCD structure information according to `msg->param` and select the memory type of draw_buffer and configure the corresponding parameters of the `lv_vnd_config_t` sturcture.
- Call the interface of `lv_vendor_init(lv_vnd_config_t *config)` to initialize the relevant configuration of LVGL.
- Call the interface of `lcd_display_open(lcd_open_t *config)` to initialize and open the LCD display.
- If the touch function exists, call the interface `drv_tp_open(int hor_size, int ver_size, tp_mirror_type_t tp_mirror)` to initialize the tp function.

- Call the component interface provided by LVGL to draw the corresponding UI.
- Call the interface of `lv_vendor_start()` to create an LVGL task and start scheduling execution.
- Call the interface of `msg_send_rsp_to_media_major_mailbox(media_mailbox_msg_t msg, uint32_t result, uint32_t dest)` to return the execution status. The parameter of dest is APP_MODULE.

Note: When draw_buffer of LVGL chooses to use sram memory, its size is 1/10 screen size according to official recommendations, which can save memory without losing frame rate.

9.1.4 4. Development description

- Set draw_buffer to use psram memory or sram memory through `CONFIG_LVGL_USE_PSRAM`.
- Use the `lv_vendor_disp_lock()` and `lv_vendor_disp_unlock()` interfaces for code protection when calling the component interface provided by LVGL to draw the corresponding UI.
- LVGL source code itself provides many additional third-party libraries, including file system interface, JPG decoder, BMP decoder, PNG decoder and GIF decoder etc. Due to the limitations of the system SRAM memory, these decoders can only decode small resolution pictures for display. For images with large resolution, PSRAM memory can be used for decoding.
- SDK currently supports both fatfs and littlefs, which is based on bk_vfs's posix interface (refer to cli_vfs.c). When using it, open `CONFIG_VFS` and select littlefs or fatfs according to needs(`CONFIG_FATFS / CONFIG_LITTLEFS`). It can alse use another way that set the macro `LV_USE_FS_FATFS` in the lv_conf.h file to 1 to use the fatfs file system.
- When LVGL use PNG, JPG and GIF decoders to decode, you need to open the corresponding macros in the lv_conf.h file, which are `LV_USE_PNG` `LV_USE_SJPG` and `LV_USE_GIF`.
- There is no need to open the `CONFIG_LVGL_USE_PSRAM` macro when selecting PSRAM memory for decoding using PNG, JPG and GIF decoders.
- Regarding the usage of LVGL rotation function, LVGL itself has its own software rotation function, which can be executed through the function `lv_disp_set_rotation()`. The function pass the parameters of `LV_DISP_ROT_90` , `LV_DISP_ROT_180` or `LV_DISP_ROT_270` to achieve image rotation at 90, 180 and 270 degrees. However, due to the limitations of this feature, it is only applicable to screens with the same width and height. For the screen with different width and height, rotating 90 degrees and 270 degrees will cause abnormal image display. To solve this issue, and in the case where the display hardware cannot change the display direction, the SDK provides an additional rotation function, which can achieve image rotation display at 90 degrees, 180 degrees and 270 degrees. The specific implementation is as follows: When initializing LVGL by call the function `lv_vendor_init(lv_vnd_config_t *config)` in main function, the `rotation` parameter in the structure `lv_vnd_config_t` can be assigned by passing `ROTATE_NONE`, `ROTATE_90`, `ROTATE_180``and ``ROTATE_270` to indicate no rotation, rotation 90, rotation 180 and rotation 270 respectively.
- Regarding how to package resource files into bin file, select the corresponding packaging tool according to the selected file system type. For tools, please consult FAE.
- LVGL's project has been added to the automated partition list.If you want to reconfigure the partition size, just set the `bk7258_partition.csv` file in project directly and pay attention to some alignment requirements.

CHAPTER

TEN

REFERENCE PROJECTS

10.1 Bluetooth Project

10.1.1 Central Project

1 Function Overview

This project show how pad work with main function: bluetooth a2dp source/avrcp tg ct/ble

2 Code Path

demo:./projects/bluetooth/central

build cmd:`make bk7258 PROJECT=bluetooth/central`

2 project path

- central: `project/bluetooth/central`

3 a2dp source cli introduce

a2dp_player connect <xx:xx:xx:xx:xx:xx>	connect to soundbar
a2dp_player disconnect <xx:xx:xx:xx:xx:xx>	disconnect
a2dp_player play <xxx.mp3>	play mp3
a2dp_player stop	stop play
a2dp_player pause	pause
a2dp_player resume	resume
a2dp_player abs_vol <xxx>	set remote absolute volume 0 ~ 127 (depend on peer impl)

4 a2dp source test procedure

1.Find a sdcard format exfat, copy project/bluetooth/central/1_qcs.mp3 to root dir. (Must be 16bits mp3)

2.Insert sdcard and power on.

3.Make soundbar enter pair mode.

4.Input `a2dp_player connect xx:xx:xx:xx:xx:xx` (xx means soundbar addr).

5.Input `a2dp_player play xxx.mp3`

6.Now you can here soundbar playing music.(You should play after connection complete as soon as possible, and don't stop, ref chapter 5)

7.You can stop or pause when play, play when stop.

5 compatibility describes

1.In music playing scenario, some soundbars(e.g. JBL) would disconnect(JBL would power up bluetooth) link when stop(local a2dp_player stop or peer avdtp suspend) too long. log will show "bt_api_event_cb:Disconnected from xx:xx:xx:xx:xx:xx"

2.Some soundbars(e.g. xiaomi) would not register avrcp playback and could express inconformitily status between local and peer.

3.Some soundbars(e.g. xiaomi) would not report avrcp volume changed evt, so central could not know. | 4.Some soundbars(e.g. xiaomi) does not support absolute volume.

6 Attention

1.Log would show "f_mount failed" or "read data crc error" when sdcard failed.

10.1.2 Headset

1. Function Overview

This project show how headset work with main function:

1.Receive music data transmitted by the peer as a2dp sink

2.Control the peer as avrcp ct/tg (play pause,etc.)

3.Receive voice data transmitted by the peer as hfp hf

4.ble gatt server/gatt client(see the introduction of Central project for details)

1.1 Specification

- **a2dp:**
  - avdtp sink
- **avrcp:**
  - tg
  - ct
- **hfp:**
  - hf
- **ble:**
  - gap

- gatt server
- gatt client
- smp legacy pair/secure connection pair

1.2 Code Path and build cmd

demo:./projects/bluetooth/headset

build cmd:`make bk7258 PROJECT=bluetooth/headset`

2. Introduction to cli commands

2.1 a2dp/avrcp

headset connect <xx:xx:xx:xx:xx:xx>	connect
headset disconnect <xx:xx:xx:xx:xx:xx>	disconnect
headset play	play
headset pause	pause
headset next	next song
headset prev	prev song
headset rewind [msec]	rewind
headset fast_forward [msec]	fast forward
headset vol_up	volume up
headset vol_down	volume down
headset pair_mode	enter pairing mode

3. Frame Diagram

3.1 Software Module Architecture Diagram

As shown in the figure below,bluetooth is responsible for receiving music data, receiving and sending voice data; audio is responsible for decoding and data,playing music and voice data, and collecting and encoding MIC data; storage is responsible for storing bluetooth information,such as encryption keys.

3.2 Flowchart

A2DP workflow and HFP workflow are shown in the figure below:

4. Configuration

4.1 Demo enable configuration

In the project path config/bk7258/config,modify the macro to configure whether to enable A2DP and HFP demo.Currently,A2DP demo is enabled by default;

//enable A2DP demo
CONFIG_A2DP_SINK_DEMO=y

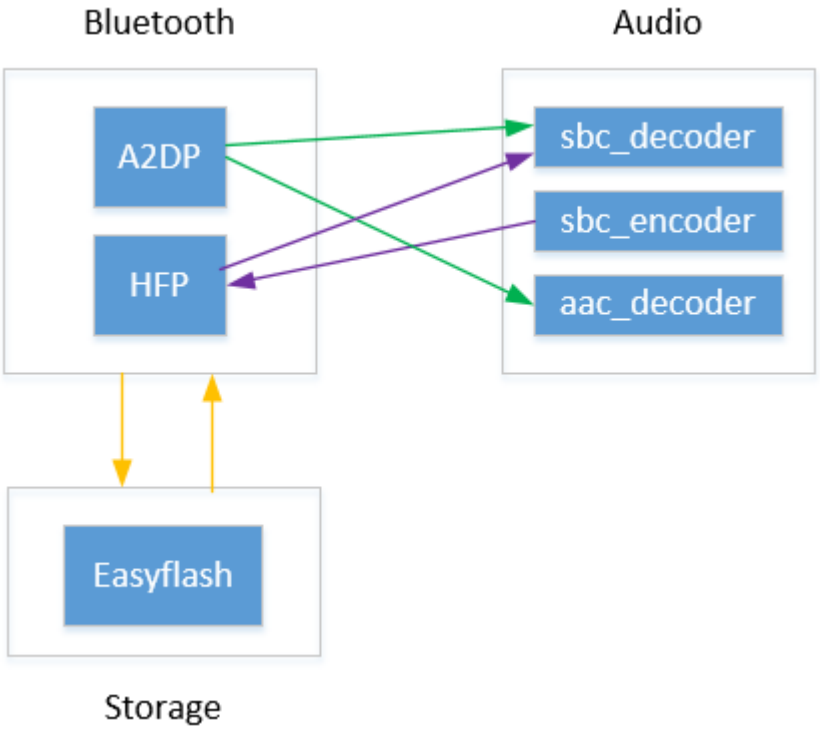


Fig. 1: Figure 1. software module architecture

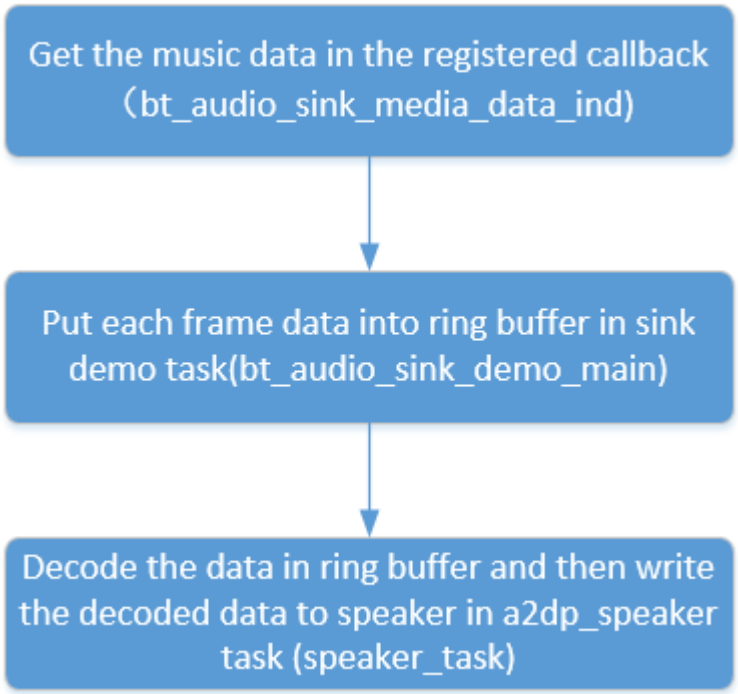


Fig. 2: Figure 2. a2dp flow chart

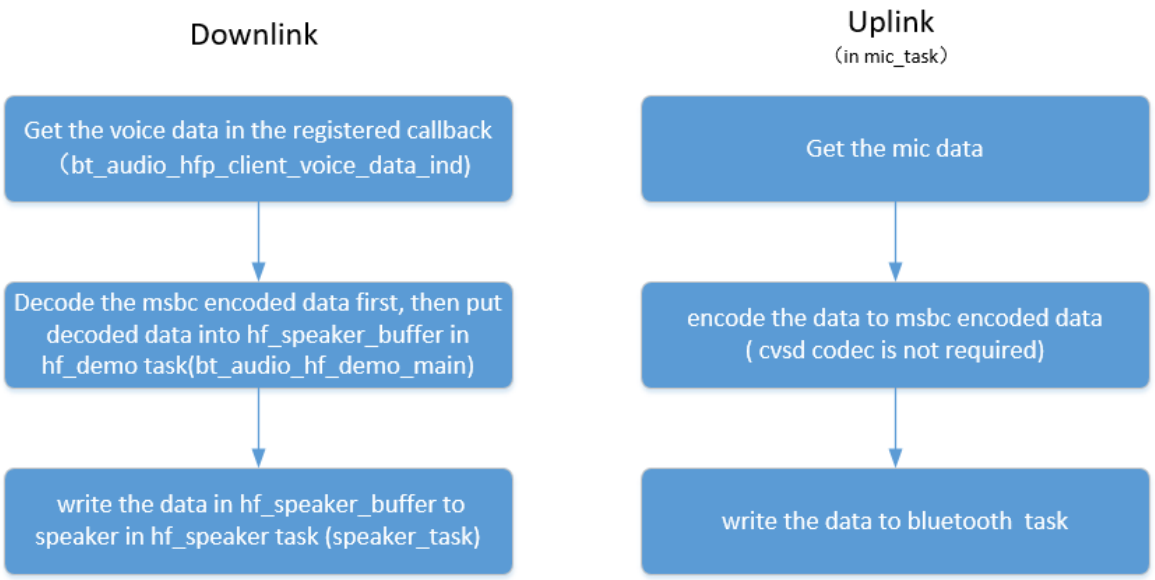


Fig. 3: Figure 3. hfp flow chart

//enable HFP demo
CONFIG_HFP_HF_DEMO=y

4.2 Codec configuration

4.2.1 A2DP

	CONFIG_SBC	CONFIG_AAC_DECODER	parameter of a2dp_sink_demo_init
support SBC	Y	N	0
support AAC	Y	Y	1

4.2.1 HFP

	CONFIG_SBC	parameter of hfp_hf_demo_init
support CVSD	N	0
support mSBC	Y	1

5. Code Explanation

5.1 A2DP demo

5.1.1 Get music data reported by BT

```
void bt_audio_sink_media_data_ind(const uint8_t *data, uint16_t data_len)
{
        bt_audio_sink_demo_msg_t demo_msg;
        int rc = -1;

        os_memset(&demo_msg, 0x0, sizeof(bt_audio_sink_demo_msg_t));

        if (bt_audio_sink_demo_msg_que == NULL)
        {
                return;
        }

        demo_msg.data = (char *) os_malloc(data_len);

        if (demo_msg.data == NULL)
        {
                LOGE("%s, malloc failed\r\n", __func__);
                return;
        }

        os_memcpy(demo_msg.data, data, data_len);
        demo_msg.type = BT_AUDIO_D2DP_DATA_IND_MSG;
        demo_msg.len = data_len;

        //send to audio_sink_demo task
        rc = rtos_push_to_queue(&bt_audio_sink_demo_msg_que, &demo_msg, BEKEN_NO_
↪WAIT);

        ...
}
```

5.1.2 Store music data in ring buffer

```
void bt_audio_sink_demo_main(void *arg)
{
        ...

        case BT_AUDIO_D2DP_DATA_IND_MSG:
        {
                ...

                uint8 *fb = (uint8_t *)msg.data;
                if(s_spk_is_started)
                {
                        //store sbc format data into the ring buffer frame by frame
                        if (CODEC_AUDIO_SBC == bt_audio_a2dp_sink_codec.type)
                        {
                                uint8_t payload_header = *fb++;
                                uint8_t frame_num = payload_header&0xF;
```

(continues on next page)

(continued from previous page)

```
                                if(msg.len - 1 != frame_length*frame_num)
                                {
                                        LOGI("recv undef sbc, payload_header %d,␣
↪payload_len: %d, frame_num:%d \r\n", payload_header, msg.len - 1, frame_num);
                                }
                                for(uint8_t i=0;i<frame_num;i++)
                                {
                                        if (ring_buffer_node_get_free_nodes(&s_a2dp_
↪frame_nodes))
                                        {
                                                uint8_t *node = ring_buffer_node_get_
↪write_node(&s_a2dp_frame_nodes);
                                                os_memcpy(node, fb, frame_length);
                                                fb += frame_length;
                                        }
                                        else
                                        {
                                                LOGI("A2DP frame nodes buffer(sbc) is␣
↪full\n");
                                                //os_free(msg.data);
                                                break;
                                        }
                                }
                        }
#if CONFIG_AAC_DECODER
                        //First parse the frame length of the AAC format data, and␣
↪then store it in the ring buffer according to the frame length
                        else if(CODEC_AUDIO_AAC == bt_audio_a2dp_sink_codec.type)
                        {
                                // LOGI("-> %d \n", msg.len);
                                uint8_t *inbuf = &fb[9];
                                uint32_t inlen = 0;
                                uint8_t  len   = 255;
                                do
                                {
                                        inlen += len = *inbuf++;
                                }
                                while (len == 255);
                                {
                                        if (ring_buffer_node_get_free_nodes(&s_a2dp_
↪frame_nodes))
                                        {
                                                uint8_t *node = ring_buffer_node_get_
↪write_node(&s_a2dp_frame_nodes);
                                                *((uint32_t *)node) = inlen;
                                                os_memcpy(node + 4, inbuf, inlen);
                                        }
                                        else
                                        {
                                                LOGI("A2DP frame nodes buffer(aac) is␣
↪full\n");
                                                os_free(msg.data);
                                                break;
                                        }
                                }
                        }
```

(continues on next page)

(continued from previous page)

```
#endif
                        else
                        {
                                LOGE("%s, Unsupported a2dp codec %d \r\n", __func__,␣
↪bt_audio_a2dp_sink_codec.type);
                        }
                        if(a2dp_speaker_sema)
                        {
                                rtos_set_semaphore(&a2dp_speaker_sema);
                        }
                }

                ...

        }
        break;

        ...

}
```

5.1.3 Decode music data

```
static void speaker_task(void *arg)
{
        ...

        for (; i < s_frame; i++)
        {
                //get data from ring buffer
                uint8_t *inbuf = ring_buffer_node_get_read_node(&s_a2dp_frame_nodes);
                bk_err_t ret;

                if (CODEC_AUDIO_SBC == bt_audio_a2dp_sink_codec.type)
                {
                        //perform sbc decoding
                        ret = bk_sbc_decoder_frame_decode(&bt_audio_sink_sbc_decoder,␣
↪inbuf, frame_length);
                        if (ret < 0)
                        {
                                LOGE("sbc_decoder_decode error <%d>\n", ret);
                                continue;
                        }
                        int16_t *dst = (int16_t*)bt_audio_sink_sbc_decoder.pcm_sample;
                        int16_t w_len = bt_audio_sink_sbc_decoder.pcm_length * 4;
                        if(CONFIG_BOARD_AUDIO_CHANNLE_NUM == 1)
                        {
                                for(int i=0; i<bt_audio_sink_sbc_decoder.pcm_length *␣
↪2; i++)
                                {
                                        dst[i] = dst[i*2];
                                }
                                w_len = bt_audio_sink_sbc_decoder.pcm_length * 2;
                        }
```

(continues on next page)

(continued from previous page)

```
                        //send decoded data to the speaker
                        int size = raw_stream_write(raw_write, (char *)dst, w_len);
                        if (size <= 0)
                        {
                                LOGE("raw_stream_write size fail: %d \n", size);
                                break;
                        }
                        else
                        {
                                //LOGI("raw_stream_write size: %d \n", size);
                        }
                }
#if (CONFIG_AAC_DECODER)
                else if(CODEC_AUDIO_AAC == bt_audio_a2dp_sink_codec.type)
                {
                        uint32_t len = *(uint32_t *)inbuf;
                        // LOGI("<- %d \n", len);
                        uint8_t *out_buf = 0;
                        uint32_t out_len = 0;

                        //perform aac decoding
                        ret = bk_aac_decoder_decode(&bt_audio_sink_aac_decoder,␣
↪(inbuf+4), len, &out_buf, &out_len);
                        if(ret == 0)
                        {
                                int16_t *dst = (int16_t*)out_buf;
                                int16_t w_len = out_len;
                                if(CONFIG_BOARD_AUDIO_CHANNLE_NUM == 1)
                                {
                                        for(uint16_t i=0;i<out_len/2;i++)
                                        {
                                                dst[i] = dst[i*2];
                                        }
                                        w_len = out_len/2;
                                }
                                //send decoded data to the speaker
                                int size = raw_stream_write(raw_write, (char *)dst, w_
↪len);

                                if (size <= 0)
                                {
                                        LOGE("raw_stream_write size fail: %d \n",␣
↪size);

                                        break;
                                }
                                else
                                {
                                        // LOGI("raw_stream_write size: %d \n", size);
                                }
                        }else
                        {
                                LOGE("sbc_decoder_decode error <%d>\n", ret);
                        }
                }
#endif
                else
                {
```

(continues on next page)

(continued from previous page)

```
                        LOGE("unsupported codec!! /n");
                }
        }

        ...
}
```

5.2 HFP demo

5.2.1 Get voice data reported by BT

```
void bt_audio_hfp_client_voice_data_ind(const uint8_t *data, uint16_t data_len)
{
        bt_audio_hf_demo_msg_t demo_msg;
        int rc = -1;

        os_memset(&demo_msg, 0x0, sizeof(bt_audio_hf_demo_msg_t));
        if (bt_audio_hf_demo_msg_que == NULL)
                return;

        demo_msg.data = (char *) os_malloc(data_len);
        if (demo_msg.data == NULL)
        {
                LOGI("%s, malloc failed\r\n", __func__);
                return;
        }

        os_memcpy(demo_msg.data, data, data_len);
        demo_msg.type = BT_AUDIO_VOICE_IND_MSG;
        demo_msg.len = data_len;

        //send to audio_hf_demo task
        rc = rtos_push_to_queue(&bt_audio_hf_demo_msg_que, &demo_msg, BEKEN_NO_WAIT);
        ...
}
```

5.2.2 Decode voice data

```
void bt_audio_hf_demo_main(void *arg)
{
        ...

        case BT_AUDIO_VOICE_IND_MSG:
        {
                ...
                //decode msbc format voice data
                if (CODEC_VOICE_MSBC == bt_audio_hfp_hf_codec)
                {
                        fb += 2; //Skip Synchronization Header
                        ret = bk_sbc_decoder_frame_decode(&bt_audio_hf_sbc_decoder,␣
↪fb, msg.len - 2);
//                        LOGI("sbc decod %d \n", ret);
```

(continues on next page)

(continued from previous page)

```
                        if (ret < 0)
                        {
                                LOGE("msbc decode fail, ret:%d\n", ret);
                        }
                        else
                        {
                                ret = BK_OK;
                                fb = (uint8_t*)bt_audio_hf_sbc_decoder.pcm_sample;
                                packet_len = r_len = SCO_MSBC_SAMPLES_PER_FRAME*2;
                                packet_num = 4;
                        }
                }
                else
                {
                        packet_len = r_len = SCO_CVSD_SAMPLES_PER_FRAME * 2;
                        packet_num = 8;
                }

                if(ret == BK_OK)
                {
                        ...
                        //store the original voice data into hf_speaker_buffer
                        os_memcpy(hf_speaker_buffer + hf_speaker_data_count, fb, r_
↪len);
                        hf_speaker_data_count += r_len;
                        if (hf_speaker_data_count >= packet_len * packet_num)
                        {
                                if (hf_speaker_sema)
                                {
                                        rtos_set_semaphore(&hf_speaker_sema);
                                }
                                hf_speaker_data_count -= packet_len * packet_num;
                        }
                }else
                {
                        //LOGE("write spk data fail \r\n");
                }

                os_free(msg.data);
        }
        break;

        ...
}
```

5.2.3 Play voice data

```
static void speaker_task(void *arg)
{
        ...

        while (hf_auido_start)
        {
                rtos_get_semaphore(&hf_speaker_sema, BEKEN_WAIT_FOREVER);
```

(continues on next page)

(continued from previous page)

```
                //send the voice data in hf_speaker_buffer to SPEAKER
                int size = raw_stream_write(raw_write, (char *)hf_speaker_buffer,␣
↪packet_len);

                if (size <= 0)
                {
                        LOGE("raw_stream_write size: %d \n", size);
                        break;
                }
                else
                {
                        //LOGI("raw_stream_write size: %d \n", size);
                }
        }

    ...
}
```

5.2.4 Get uplink MIC data

```
static void mic_task(void *arg)
{
        ...

        while (hf_auido_start)
        {
                if (hf_mic_data_count+read_size < sizeof(hf_mic_sco_data))
                {
                        //get MIC data
                        int size = raw_stream_read(raw_read, (char *)(hf_mic_sco_data␣
↪+ hf_mic_data_count), read_size);
                        if (size > 0)
                        {
                                ...

                                while (hf_mic_data_count >= send_len)
                                {
                                        //decide whether to perform msbc encoding␣
↪based on the selected encoding format
                                        if (CODEC_VOICE_MSBC == bt_audio_hfp_hf_codec)
                                        {
                                                int32_t produced = sbc_encoder_
↪encode(&bt_audio_hf_sbc_encoder, (int16_t *)(hf_mic_sco_data + send_len * i));
//                          LOGI("[send_mic_data_to_air_msbc]  %d \r\n",produced);
                                                //send the encoded voice data to BT
                                                bk_bt_hf_client_voice_out_write(hfp_
↪peer_addr, (uint8_t *)&bt_audio_hf_sbc_encoder.stream [ -2 ], produced + 2);
                                        }else
                                        {
                                                //send uncoded voice data to BT
                                                bk_bt_hf_client_voice_out_write(hfp_
↪peer_addr, hf_mic_sco_data + send_len * i, send_len);
                                        }
```

(continues on next page)

(continued from previous page)

```
                                        i++;
                                        hf_mic_data_count -= send_len;
                                }
                                if (hf_mic_data_count)
                                {
                                        os_memmove(hf_mic_sco_data, hf_mic_sco_data +␣
↪send_len * i, hf_mic_data_count);
                                }
                        }
                }
                else
                {
                        LOGE("MIC BUFFER FULL \r\n");
                        hf_mic_data_count = 0;
                }
        }

        ...
}
```

10.2 Multi-media Projects

10.2.1 Doorbell

1. Introduction

This project is a demo of a USB camera door lock, supporting end-to-end (BK7258 device) to mobile app demonstrations. Support multi camera switch for door lock, include 1 dvp and 2 uvc. The default use 16M psram.

1.1 Specifications

- **Hardware configuration:**
  - Core board, **BK7258_QFN88_9X9_V3.2**
  - Display adapter board, **BK7258_LCD_interface_V3.0**
  - MIC small board, **BK_Module_Microphone_V1.1**
  - SPEAKER small board, **BK_Module_Speaker_V1.1**
  - PSRAM 8M/16M
- **Support, UVC**
  - Reference peripherals, UVC resolution of **864 * 480**
- **Support, DVP**
  - Reference peripherals, gc2145 resolution of **864 * 480**
- Support, UAC
- Support, TCP LAN image transmission

- Support UDP LAN image transmission
- Support, Shangyun, P2P image transfer
- **Support, LCD RGB/MCU I8080 display**
  - Reference peripherals, **ST7701SN**, 480 * 854 RGB LCD
  - RGB565/RGB888
- **Support, hardware/software rotation**
  - 0°, 90°, 180°, 270°
- Support, onboard speaker
- Support, Micro
- **Support, MJPEG hardware decoding**
  - YUV422
- **Support, MJPEG software decoding**
  - YUV420
- Support, H264 hardware decoding
- **Support, OSD display**
  - ARGB888[PNG]
  - Custom Font

Warning: Please use reference peripherals to familiarize and learn about demo projects. If the specifications of the peripherals are different, the code may need to be reconfigured.

1.2 Path

<bk_avdk source code path>/projects/media/doorbell

2. Framework diagram

2.1 Software Module Architecture Diagram

As shown in the following figure, BK7258 has multiple CPUs:

- CPU0, running WIFI/BLE as a low-power CPU.
- CPU1, running multimedia and serves as a high-performance multimedia CPU.

- In the UVC scheme, we adopt a pipeline approach to improve overall performance.
- **The images output by UVC cameras can be divided into two types: YUV420 MJPEG and YUV422 MJPEG.**
  - The software will automatically recognize and use a hardware decoder to decode YUV422 MJPEG. YUV420 MJPEG uses CPU1 and CPU2 for software decoding.
  - When decoding hardware, the image resolution needs to be a multiple of 32 for wide resolution and a multiple of 16 for high resolution.
  - YUV pixel arrangement is divided into planar format, packed format, and semi planar format. Hardware encoded data, in packet format when required.
- **MJPEG HW Decoder, in pipeline mode, due to the encoding data of H264, it needs to be decoded based on MJPEG be**

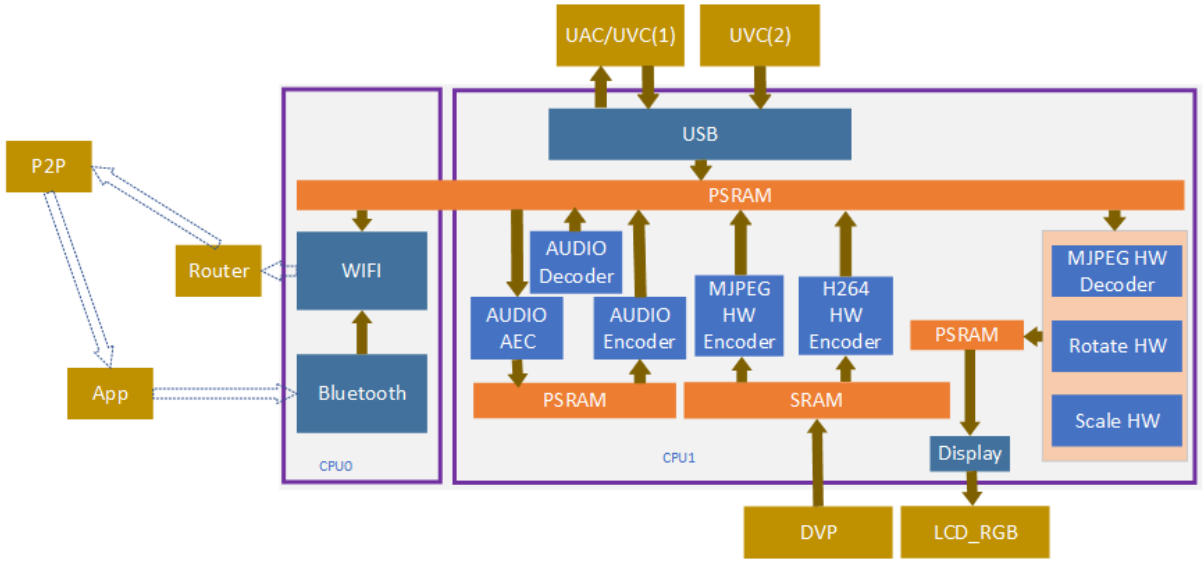


Fig. 4: Figure 1. software module architecture

  - When closing, it is important to note that both the display and image transfer are closed before closing this module. The default demo already includes this logic.

- **MJPEG SW Decoder, two decoders will not work simultaneously at the same time.**
  - Once the image is confirmed to be YUV420 or YUV422, it is decided whether to use software decoding or hardware decoding.
  - When work camera switch, one camera may output JPEG(YUV422) and another may output JPEG(YUV420), the system can recognize format and readapt decode method, customer not need do other work.
- **Rota HW and Rota SW will only use one type of rotating module at the same time.**
  - Rota HW, supports RGB 565 image output, supports 0°, 90°, 270°.
  - Rota SW supports 0°, 90°, 180°, and 270°.
  - If you need to use RGB888 output or support 180°, and meet one of the conditions, you need to switch to software decoding.
  - How to make decisions on Rota HW and Rota SW currently is determined by the SDK software. Users only need to input the rotation angle and output image format parameters into the corresponding interface when opening the LCD.

2.2 Code Module Relationship Diagram

As shown in the following figure, multimedia interfaces are defined in **media_app.h** and **aud_intf.h**.

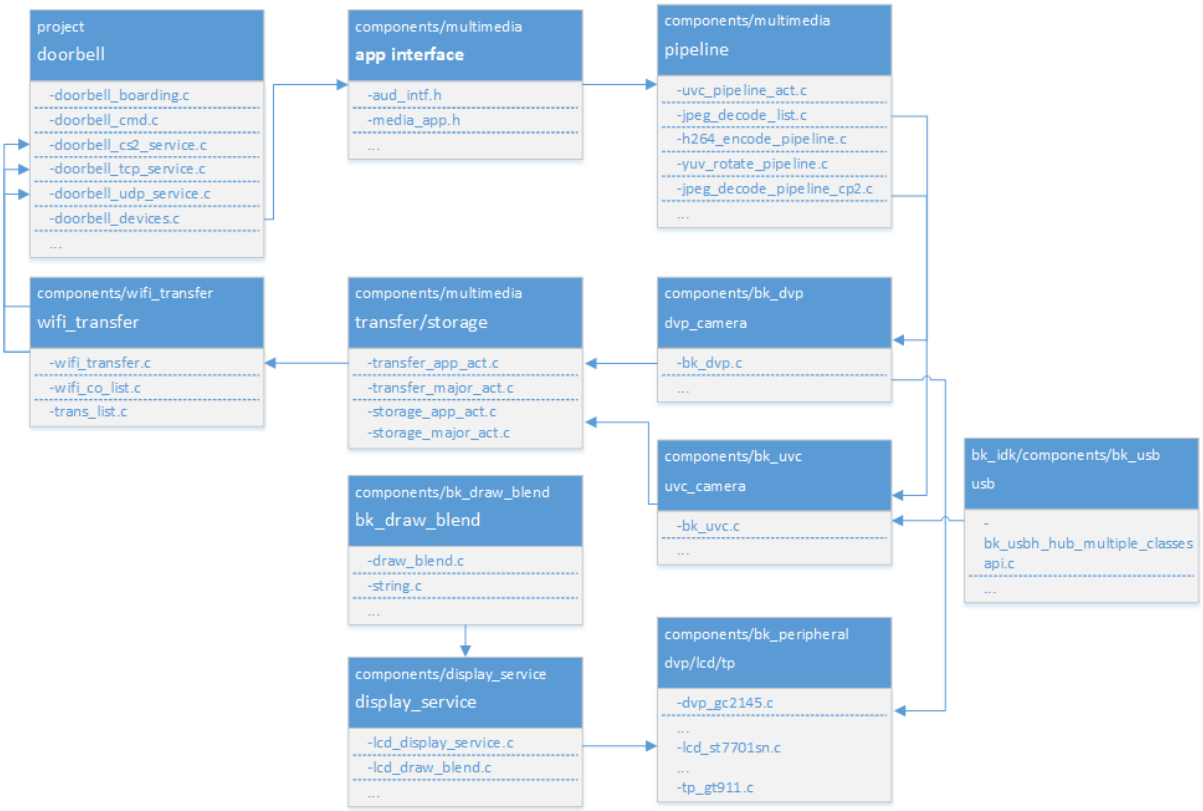


Fig. 5: Figure 2. module relationship diagram

3. Configuration

3.1 Bluetooth and Multimedia Memory Reuse

In order to further save memory, in the default project, the multimedia memory encoding and decoding memory and Bluetooth memory are multiplexed, mainly using the following two macros. If you want to use two modules in parallel, you can close them yourself. Please confirm if the overall memory is sufficient before closing.

Kconfig	CPU	Format	Value
CONFIG_BT_REUSE_MEDIA_MEMORY	CPU0 && CPU1	bool	y
CONFIG_BT_REUSE_MEDIA_MEM_SIZE	CPU0 && CPU1	hex	0x1B000

- In order to solve memory reuse conflicts during actual use, it is necessary to check the status of Bluetooth and disable or uninstall Bluetooth before using the multimedia module.
- If the multimedia modules have already been turned off and you want to use them again, you need to reinitialize Bluetooth. Please refer to the following code.
- The range of values is based on the maximum memory required by the Bluetooth hardware module and the maximum memory required for multimedia hardware encoding, with one value being the maximum.
- The hardware memory requirements for general Bluetooth are relatively small [actual statistics need to be calculated based on the compiled map program]. Because it is generally configured according to the maximum memory capacity of multimedia hardware.

3.1.1 Uninstalling Bluetooth

```
#ifdef CONFIG_BT_REUSE_MEDIA_MEMORY
#if CONFIG_BLUETOOTH
    bk_bluetooth_deinit();
#endif
#endif
```

3.1.2 Initialize Bluetooth

```
bk_bluetooth_init();
```

3.2 Hardware Decoding Memory Configuration Instructions

A hardware accelerator requires a portion of memory, which is optimized based on the actual resolution. The default configuration parameters are LCD with a 480 * 854 vertical screen and Camera with an 864 * 480 MJPEG image.

```
//The recommended output resolution and width for Camera are multiples of 32. When␣
↪the default configuration of the screen and camera is small, memory can be␣
↪optimized by modifying the configuration macro.
#define IMAGE_MAX_WIDTH (864)
#define IMAGE_MAX_HEIGHT (480)

//When starting the scaling module, it is necessary to pay attention to these two␣
↪sets of parameters. The default recommendation is that the width should be slightly␣
↪larger than the screen.
#define DISPLAY_MAX_WIDTH (864)
#define DISPLAY_MAX_HEIGHT (480)

typedef struct {
#if SUPPORTED_IMAGE_MAX_720P
    uint8_t decoder[DECODE_MAX_PIPELINE_LINE_SIZE * 2];
    uint8_t scale[SCALE_MAX_PIPELINE_LINE_SIZE * 2];
    uint8_t rotate[ROTATE_MAX_PIPELINE_LINE_SIZE * 2];
#else
    uint8_t decoder[DECODE_MAX_PIPELINE_LINE_SIZE * 2];
    uint8_t rotate[ROTATE_MAX_PIPELINE_LINE_SIZE * 2];
#endif

} mux_sram_buffer_t;

* If rotation is not required, the memory of the rotating part can be saved.
* Attention should be paid to the resolution of scaling. The scaled resolution, width,
↪ and height must all be multiples of 8.
```

Caution: When the VNet BT-REUSEUMEDIA.MMORY macro is opened, this portion of memory will be reused with Bluetooth hardware memory.

4. Demonstration explanation

Please visit APP Usage Document.

Hint: If you do not have cloud account permissions, you can use debug mode to set the local area network TCP/UDP/CS2 image transmission method.

---

5. Code explanation

5.1 UVC Camera

Supported peripherals, please refer to Support Peripherals

5.1.1 Turn on UVC

5.1.1.1 Application Code

```
//Path      : projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

int doorbell_camera_turn_on(camera_parameters_t *parameters)
{
    ...

    //turn on camera
    ret = media_app_camera_open(&db_device_info->video_handle, &device);

    //set rotate angle
    //attentions:
    //    1.while MJPEG be YUV422 MJPEG, only rotate on local lcd, the h264 stream␣
↪not rotate.
    //    2.while MJPEG be UV420 MJPEG, rotate do the same time with jpeg decode,
    media_app_set_rotate(rot_angle);

    //enable h264 encode pipeline
    ret = media_app_h264_pipeline_open();

    if (device.type == UVC_CAMERA)
    {
        // uvc output jpeg frame and open jpeg decode, decode by 16 lines outout yuv␣
↪format.
        media_app_pipeline_jdec_open();
    }
    else if (device.type == DVP_CAMERA)
    {
        // dvp output yuv format data, enable yuv deal task
        media_app_frame_jdec_open(NULL);
    }


    ...
}
```

5.1.1.2 Interface Code

```
//Path      :  components/multimedia/app/media_app.c
//Loaction  :  CPU0

bk_err_t media_app_camera_open(camera_handle_t *handle, media_camera_device_t *device)
{
    ...

    //Uninstall Bluetooth
    #ifdef CONFIG_BT_REUSE_MEDIA_MEMORY
    #if CONFIG_BLUETOOTH
        bk_bluetooth_deinit();
    #endif
    #endif

    //Vote to activate CPU1. The purpose of voting is to ensure that CPU1 can be␣
↪automatically turned off when not in use, in order to achieve the goal of low power␣
↪consumption.
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_JPEG_EN, PM_POWER_
↪MODULE_STATE_ON);

    //Notify CPU1 to turn on the UVC camera.
    media_device_t media_device = {0};
    media_device.param1 = (uint32_t)handle;
    media_device.param2 = (uint32_t)device;
    //notify cpu1 open camera
    ret = media_send_msg_sync(EVENT_CAM_UVC_OPEN_IND, (uint32_t)media_device);

    ...
}

typedef struct {
    camera_type_t type; // camera type
    uint16_t port;      // camera port index(uvc:[1,3], dvp:0)
    uint16_t format;    // camera output image format, reference image_format_t
    uint16_t width;     // camera output image width
    uint16_t height;    // camera output image height
    uint32_t fps;       // camera output image fps
    media_rotate_t rotate;// reserve
} media_camera_device_t;
```

5.1.2 Obtain an image

5.1.2.1 Enable interface

```
//Path      : components/multimedia/app/media_app.c
//Loaction  :  CPU0

bk_err_t media_app_register_read_frame_callback(image_format_t fmt, frame_cb_t cb)
{
    ...

    //cb: register image process callback, will return a frame buffer in callback␣
↪function
```

(continues on next page)

(continued from previous page)

```
    //fmt: the image format you want to read

    ...
}
```

5.1.2.2 Disable interface

```
//Path      : components/multimedia/app/media_app.c
//Loaction  :  CPU0
bk_err_t media_app_unregister_read_frame_callback(void)
{
    ...

    //called this function, will stop read frame, and the callback that be register␣
↪not been called again

    ...
}
```

Attention: Here introduces how to read images, through the above interface, users can get the image, but in the callback function, do not handle it too long, it is recommended to copy the image data to the customer thread for processing in the callback function, otherwise, the task of reading images will be stuck, leading to frame loss.

5.1.3 Turn off UVC

5.1.3.1 Application Code

```
//Path      :  projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

int doorbell_camera_turn_off(void)
{
    ...

    //Disable H264 encoding
    media_app_h264_pipeline_close();

    //close jpeg decode pipeline function(maybe not open before)
    media_app_pipeline_jdec_close();
    //close yuv data process task(maybe not open before)
    media_app_frame_jdec_close();

    //close all camera have been opened
    do {
        //get current camera handle
        db_device_info->video_handle = bk_camera_handle_node_pop();
        if (db_device_info->video_handle)
        {
            LOGI("%s, %d, %p\n", __func__, __LINE__, db_device_info->video_handle);
            media_app_camera_close(&db_device_info->video_handle);
```

(continues on next page)

(continued from previous page)

```
        }
        else
        {
            break;
        }
    } while (1);

    ...
}
```

5.1.3.2 Interface Code

```
//Path      :  components/multimedia/app/media_app.c
//Loaction  :  CPU0

bk_err_t media_app_camera_close(camera_handle_t *handle)
{
    ...

    //Turn off UVC camera by handle
    ret = media_send_msg_sync(EVENT_CAM_UVC_CLOSE_IND, (uint32_t)handle);

    //Vote to allow CPU1 to be turned off. The purpose of voting is to ensure that␣
↪CPU1 can be automatically turned off when not in use, in order to achieve the goal␣
↪of low power consumption.
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_JPEG_EN, PM_POWER_
↪MODULE_STATE_OFF);

    ...
}

bk_err_t media_app_pipeline_jdec_open(void)
{
    ...

    //vote to enable cpu1
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_JPEG_DE, PM_POWER_
↪MODULE_STATE_ON);

    //set jpeg decode yuv data need rotate or not
    ret = media_send_msg_sync(EVENT_PIPELINE_SET_ROTATE_IND, jpeg_decode_pipeline_
↪param.rotate);

    //enable jpeg decode pipeline function
    ret = media_send_msg_sync(EVENT_PIPELINE_LCD_JDEC_OPEN_IND, 0);

    ...
}
```

Warning:

- All operations involving multimedia require attention to the requirement of low power consumption. To turn on the device, it must be turned off, otherwise the entire system cannot enter low-power mode.

- The operation involving CPU1 voting, opening and closing, must appear in pairs, otherwise there will be a problem of CPU1 being unable to close and increasing power consumption.
- You can refer to the chapter on low power consumption

5.2 LCD Display

Supported peripherals, please refer to Support Peripherals

5.2.1 Open LCD

5.2.1.1 Application Code

```
//Path      : projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

int doorbell_display_turn_on(uint16_t id, uint16_t rotate, uint16_t fmt)
{
    ...

    //Set the pixel format for display
    if (fmt == 0)
    {
        media_app_lcd_fmt(PIXEL_FMT_RGB565_LE);
    }
    else if (fmt == 1)
    {
        media_app_lcd_fmt(PIXEL_FMT_RGB888);
    }

    //Set the rotation angle.
    switch (rotate)
    {
        case 90:
            rot_angle = ROTATE_90;
            break;
        case 180:
            rot_angle = ROTATE_180;
            break;
        case 270:
            rot_angle = ROTATE_270;
            break;
        case 0:
        default:
            rot_angle = ROTATE_NONE;
            break;
    }

    media_app_set_rotate(rot_angle);

    //Open local LCD display
   media_app_lcd_disp_open(&lcd_open);
```

(continues on next page)

(continued from previous page)

```
    ...
}
```

5.2.1.2 Interface Code

```
//Path      :  components/multimedia/app/media_app.c
//Loaction  :  CPU0

bk_err_t media_app_lcd_pipeline_open(void *lcd_open)
{
    ...

    //
    ret = media_app_lcd_pipeline_disp_open(config);

    //
    ret = media_app_lcd_pipeline_jdec_open();

    ...
}

bk_err_t media_app_lcd_disp_open(void *config)
{
    ...

    //Vote to activate CPU1. The purpose of voting is to ensure that CPU1 can be␣
↪automatically turned off when not in use, in order to achieve the goal of low power␣
↪consumption.
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_LCD, PM_POWER_MODULE_
↪STATE_ON);

    //Notify CPU1 to turn on the LCD
    ret = media_send_msg_sync(EVENT_PIPELINE_LCD_DISP_OPEN_IND, (uint32_t)config);

    ...
}
```

5.2.2 Turn off LCD

5.2.2.1 Application Code

```
//Path      : projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

int doorbell_display_turn_off(void)
{
    ...

    //Turn off local LCD display
    media_app_lcd_pipeline_close();

    ...
}
```

5.2.2.2 Interface Code

```
//Path      : components/multimedia/app/media_app.c
//Loaction  :  CPU0

bk_err_t media_app_lcd_disp_close(void)
{
    ...

    //Turn off the display LCD
    ret = media_send_msg_sync(EVENT_PIPELINE_LCD_DISP_CLOSE_IND, 0);

    //vote to close cpu1
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_LCD, PM_POWER_MODULE_
↪STATE_OFF)

    ...
}
```

5.2.3 OSD Display

5.3 Audio

5.3.1 Open UAC, onboard MIC/SPEAKER

```
//Path      :  projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

int doorbell_audio_turn_on(audio_parameters_t *parameters)
{
    ...

    //Enable AEC
    if (parameters->aec == 1)
    {
        aud_voc_setup.aec_enable = true;
    }
    else
    {
        aud_voc_setup.aec_enable = false;
    }

    //Set SPEAKER single ended mode
    ud_voc_setup.spk_mode = AUD_DAC_WORK_MODE_SIGNAL_END;

    //Turn on UAC
    if (parameters->uac == 1)
    {
        aud_voc_setup.mic_type = AUD_INTF_MIC_TYPE_UAC;
        aud_voc_setup.spk_type = AUD_INTF_SPK_TYPE_UAC;
    }
    else //Activate onboard MIC and SPEAKER
    {
        aud_voc_setup.mic_type = AUD_INTF_MIC_TYPE_BOARD;
```

(continues on next page)

(continued from previous page)

```
        aud_voc_setup.spk_type = AUD_INTF_SPK_TYPE_BOARD;
    }

    if (aud_voc_setup.mic_type == AUD_INTF_MIC_TYPE_BOARD && aud_voc_setup.spk_type␣
↪== AUD_INTF_SPK_TYPE_BOARD) {
        aud_voc_setup.data_type = parameters->rmt_recoder_fmt - 1;
    }

    //Set sampling rate
    switch (parameters->rmt_recorder_sample_rate)
    {
        case DB_SAMPLE_RARE_8K:
            aud_voc_setup.samp_rate = 8000;
        break;

        case DB_SAMPLE_RARE_16K:
            aud_voc_setup.samp_rate = 16000;
        break;

        default:
            aud_voc_setup.samp_rate = 8000;
        break;
    }

    //Register MIC data callback
    aud_intf_drv_setup.aud_intf_tx_mic_data = doorbell_udp_voice_send_callback;

    ...
}
```

5.3.2 Obtaining uplink MIC data

```
//Path      :  projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

//Register MIC callback
aud_intf_drv_setup.aud_intf_tx_mic_data = doorbell_udp_voice_send_callback;
ret = bk_aud_intf_drv_init(&aud_intf_drv_setup);

int doorbell_udp_voice_send_callback(unsigned char *data, unsigned int len)
{
    ...

    //The usual callback is to transmit in the direction of WIFI.
    return db_device_info->audio_transfer_cb->send(buffer, len, &retry_cnt);
}
```

5.3.3 Play downstream SPEAKER data

```
//Path      :  projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

void doorbell_audio_data_callback(uint8_t *data, uint32_t length)
{
    ...

    //Send data to SPEAKER
    ret = bk_aud_intf_write_spk_data(data, length);

    ...
}
```

5.3.4 AEC/Noise Reduction Treatment

Please refer to AEC Debug

5.3.7 Turn off UAC, onboard MIC/SPEAKER

```
//Path      :  projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

int doorbell_audio_turn_off(void)
{
    ...

    bk_aud_intf_voc_stop();
    bk_aud_intf_voc_deinit();
    /* deinit aud_tras task */
    aud_work_mode = AUD_INTF_WORK_MODE_NULL;
    bk_aud_intf_set_mode(aud_work_mode);
    bk_aud_intf_drv_deinit();

    ...
}
```

5.4 H264 Encoding and Decoding

Please refer to the H264 encoding

5.5 WIFI transmission

5.5.1 Setting up WIFI network data transmission callback

```
//Path      :  projects/media/doorbell/main/src/doorbell_udp_service.c
//Loaction  :  CPU0

bk_err_t doorbell_udp_service_init(void)
{
```

(continues on next page)

(continued from previous page)

```
    ...

    //Here, we have set up callbacks for image and audio data to WIFI
    doorbell_devices_set_camera_transfer_callback(&doorbell_udp_img_channel);
    doorbell_devices_set_audio_transfer_callback(&doorbell_udp_aud_channel);

    ...
}

typedef struct {
    //The callback for the final data transmission
    media_transfer_send_cb send;

    //Packaging of Head and Payload before Data Transmission
    media_transfer_prepare_cb prepare;

    //Optimize packet loss handling for latency optimization
    media_transfer_drop_check_cb drop_check;

    //Obtain the TX data buffer that needs to be filled
    media_transfer_get_tx_buf_cb get_tx_buf;

    //Get the size of the TX buffer to be filled
    media_transfer_get_tx_size_cb get_tx_size;

    //Set the data format of the image
    pixel_format_t fmt;
} media_transfer_cb_t;
```

5.5.1 Obtaining H264 image data

```
//Path      :  components/wifi_transfer/src/wifi_transfer.c
//Loaction  :  CPU0

bk_err_t bk_wifi_transfer_frame_open(const media_transfer_cb_t *cb, uint16_t img_
↪format)
{
    ...

    //Improve the performance of network image transmission
    bk_wifi_set_wifi_media_mode(true);
    bk_wifi_set_video_quality(WIFI_VIDEO_QUALITY_SD);

    ...

    //Register H264 image data and obtain callback(if need h264 fmt=IMAGE_H264)
    ret = media_app_register_read_frame_callback(img_format, wifi_transfer_read_frame_
↪callback);

    ...
}
```

5.5.2 Open Image Data Image Transfer

```
//Path      :  projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

int doorbell_video_transfer_turn_on(void)
{
    ...

    //Open image transfer
    if (db_device_info->camera_transfer_cb)
    {
        if (db_device_info->h264_transfer)
        {
            ret = bk_wifi_transfer_frame_open(db_device_info->camera_transfer_cb,␣
↪IMAGE_H264);
        }
        else
        {
            ret = bk_wifi_transfer_frame_open(db_device_info->camera_transfer_cb,␣
↪IMAGE_MJPEG);
        }
    }
    else
    {
        LOGE("media_transfer_cb: NULL\n");
    }

    ...
}
```

5.5.2 Close Image Data Image Transfer

```
//Path      :  projects/media/doorbell/main/src/doorbell_devices.c
//Loaction  :  CPU0

int doorbell_video_transfer_turn_off(void)
{
    ...

    //Close image transfer
    ret = bk_wifi_transfer_frame_close();

    ...
}


5.6 Camera switch
.................................

::

    //Path      : projects/media/doorbell/main/src/app_main.c
    //Loaction  : CPU0
```

(continues on next page)

(continued from previous page)

```
static void media_app_camera_switch(media_camera_device_t *device)
{
    os_printf("%s\r\n", __func__);
    bk_err_t ret;

    //judge current camera is woring
    if (db_device_info->video_handle != NULL) {
        //close h264 pipeline function
        ret = media_app_pipeline_h264_close();
        if (ret != BK_OK)
        {
            os_printf("media_app_pipeline_h264_close failed\n");
            return;
        }

        //close dvp yuv image display function
        ret = media_app_frame_jdec_close();
        if (ret != BK_OK) {
            os_printf("media_app_frame_jdec_close failed\r\n");
            return;
        }

        //close jpegdec pipeline function(include yuv rotate pipeline function)
        ret = media_app_pipeline_jdec_close();
        if (ret != BK_OK) {
            os_printf("media_app_pipeline_jdec_close failed\r\n");
            return;
        }

        //close the camera is working
        ret = media_app_camera_close(&db_device_info->video_handle);
        if (ret != BK_OK) {
            os_printf("media_app_camera_close failed\r\n");
            return;
        }
    }

    //set yuv rotate angle
    media_app_set_rotate(ROTATE_90);

    //open the camera wanted switch
    ret = media_app_camera_open(&db_device_info->video_handle, device);
    if (ret != BK_OK) {
        os_printf("media_app_camera_open failed\r\n");
        return;
    }

    if (device->type == DVP_CAMERA) {
        //open dvp yuv image display function
        ret = media_app_frame_jdec_open(NULL);
        if (ret != BK_OK) {
            os_printf("media_app_frame_jdec_open failed\r\n");
            return;
        }
    } else {
        //open jpegdec pipeline function(include yuv rotate pipeline function)
```

(continues on next page)

(continued from previous page)

```
            ret = media_app_pipeline_jdec_open();
            if (ret != BK_OK) {
                os_printf("media_app_pipeline_jdec_open failed\r\n");
                return;
            }

            if (db_device_info->h264_transfer) {
                //if enable h264 wifi transfer, need open h264 encode pipeline␣
↪function
                ret = media_app_pipeline_h264_open();
                if (ret != BK_OK)
                {
                    os_printf("media_app_pipeline_h264_open failed\n");
                    return;
                }
            }
        }
    }
```

5.6.1 Camera Switching Interface Call Flow

1. jpeg (864x480) + Wi-Fi Transmission + LCD Rotating Display (480x854)

    - Open the first camera, assumed to be DVP, with image formats of IMAGE_YUV & IMAGE_MJPEG (supporting simultaneous output of MJPEG and YUV), using media_app_camera_open().
    - Open the jpeg transmission, assuming the network port and channel are already configured, read the jpeg image using the interface, format=IMAGE_MJPEG, using media_app_register_read_frame_callback().
    - If you need to display on the LCD screen, open the hardware display function using media_app_lcd_disp_open().
    - If you need to display on the LCD screen and require rotation, since DVP supports YUV output by default, rotate the YUV image and let the display module show it, configure the rotation angle using media_app_set_rotate().
    - If you need to display on the LCD screen, send the YUV image (which may already be rotated) to the hardware display using the YUV processing function using media_app_frame_jdec_open().
    - If you need to open SD card storage for MJPEG images, currently supporting two storage modes:

        1) Single shot MJPEG capture, store one frame of MJPEG image per call, using media_app_capture(). Close storage when no longer needed using media_app_storage_close().
        2) Continuous MJPEG storage, store every frame captured by the camera to the SD card, using media_app_save_start(). Pause storage using media_app_save_stop(), (the storage task remains open and can be restarted). Close storage when no longer needed using media_app_storage_close().

    When switching to another camera (UVC), close the current camera's processes first, then start the target camera, and preferably start other functions.

    - If you need to display on the LCD screen, close the YUV image processing function using media_app_frame_jdec_close().
    - Close the current camera using media_app_camera_close().

- Open the other camera (UVC) using media_app_camera_open().
- If you need to display on the LCD, open the JPEG decoding and rotation function using media_app_pipeline_jdec_open(), which may require setting the rotation angle.

When switching to another camera (UVC), close the current camera's processes first, then start the target camera, and preferably start other functions.

- If you need to display on the LCD screen, close the decoding and rotation function using media_app_pipeline_jdec_close().
- Close the current camera using media_app_camera_close().
- Open the other camera (UVC) using media_app_camera_open().
- If you need to display on the LCD, open the JPEG decoding and rotation function using media_app_pipeline_jdec_open(), which may require setting the rotation angle.

When switching to another camera (DVP), close the current camera's processes first, then start the target camera, and preferably start other functions.

- If you need to display on the LCD screen, close the decoding and rotation function using media_app_pipeline_jdec_close().
- Close the current camera using media_app_camera_close().
- Open the other camera (UVC) using media_app_camera_open().
- If you need to display on the LCD, send the YUV image (which may already be rotated) to the hardware display using the YUV processing function using media_app_frame_jdec_open().

During the switching process, follow this flow arbitrarily.

- When closing multimedia functions, ensure all called functions are closed. All closing interfaces are protected, meaning they can be called even if the function was not opened:
  1) If you need to display on the LCD screen, close the decoding and rotation function using media_app_pipeline_jdec_close().
  2) If you need to display on the LCD screen, close the YUV image processing function using media_app_frame_jdec_close().
  3) Close the transmission using media_app_unregister_read_frame_callback().
  4) Close storage using media_app_storage_close().
  5) Close all opened cameras, obtain the opened camera using bk_camera_handle_node_pop() and close it using media_app_camera_close(), until no more cameras can be obtained from bk_camera_handle_node_pop().

2. h264 (864x480) + Wi-Fi Transmission + LCD Rotating Display (480x854)
   - Open the first camera, assumed to be DVP, with image formats of IMAGE_YUV & IMAGE_H264 (supporting simultaneous output of H264 and YUV), using media_app_camera_open().
   - Open the h264 transmission, assume the network port and channel are already configured, read the h264 image using the interface, format=IMAGE_H264, using media_app_register_read_frame_callback().
   - If you need to display on the LCD screen, open the hardware display function using media_app_lcd_disp_open().
   - If you need to display on the LCD screen and require rotation, since DVP supports YUV output by default, rotate the YUV image and let the display module show it, configure the rotation angle using media_app_set_rotate().
   - If you need to display on the LCD screen, send the YUV image (which may already be rotated) to the hardware display using the YUV processing function using media_app_frame_jdec_open().

- If you need to open SD card storage for H264 images, currently supporting continuous H264 storage, store the H264 bitstream continuously, using media_app_save_start(). Pause storage using media_app_save_stop(), (the storage task remains open and can be restarted). Close storage when no longer needed using media_app_storage_close().

When switching to another camera (UVC), close the current camera's processes first, then start the target camera, and preferably start other functions.

- If you need to display on the LCD screen, close the YUV image processing function using media_app_frame_jdec_close().
- Close the current camera using media_app_camera_close().
- Open the other camera (UVC) using media_app_camera_open().
- Open the H264 encoding function using media_app_h264_pipeline_open().
- If you need to display on the LCD, open the JPEG decoding and rotation function using media_app_pipeline_jdec_open(), which may require setting the rotation angle.

When switching to another camera (UVC), close the current camera's processes first, then start the target camera, and preferably start other functions.

- Close the H264 encoding function using media_app_h264_pipeline_close().
- If you need to display on the LCD screen, close the decoding and rotation function using media_app_pipeline_jdec_close().
- Close the current camera using media_app_camera_close().
- Open the other camera (UVC) using media_app_camera_open().
- Open the H264 encoding function using media_app_h264_pipeline_open().
- If you need to display on the LCD, open the JPEG decoding and rotation function using media_app_pipeline_jdec_open(), which may require setting the rotation angle.

When switching to another camera (DVP), close the current camera's processes first, then start the target camera, and preferably start other functions.

- Close the H264 encoding function using media_app_h264_pipeline_close().
- If you need to display on the LCD screen, close the decoding and rotation function using media_app_pipeline_jdec_close().
- Close the current camera using media_app_camera_close().
- Open the other camera (UVC) using media_app_camera_open().
- If you need to display on the LCD, send the YUV image (which may already be rotated) to the hardware display using the YUV processing function using media_app_frame_jdec_open().

During the switching process, follow this flow arbitrarily.

When closing multimedia functions, ensure all called functions are closed. All closing interfaces are protected, meaning they can be called even if the function was not opened:

1) Close the H264 encoding function using media_app_h264_pipeline_close().
2) If you need to display on the LCD screen, close the decoding and rotation function using media_app_pipeline_jdec_close().
3) If you need to display on the LCD screen, close the YUV image processing function using media_app_frame_jdec_close().
4) Close the transmission using media_app_unregister_read_frame_callback().
5) Close storage using media_app_storage_close().
6) Close all opened cameras, obtain the opened camera using bk_camera_handle_node_pop() and close it using media_app_camera_close(), until no more cameras can be obtained from bk_camera_handle_node_pop().

Warning:

- All multimedia operations must ensure low power consumption requirements. That is, open devices must be closed, otherwise, the entire system cannot enter low power mode.
- Operations involving CPU1 voting (opening and closing) must be paired, otherwise, CPU1 cannot be turned off, leading to increased power consumption.
- If the system fails to enter low power mode or CPU1 cannot drop power, use the command media_debug 8 to check if any module has not voted.

6 Doorbell

Below flowchart provides a simple introduction to the start-up process, camera switching process, and shutting-down process of the video component in doorbell, involving functional modules: wifi_transfer, sdcard_storage, lcd_display.

6.1 Enable video function

Video-related features include modules involving images throughout the entire application.

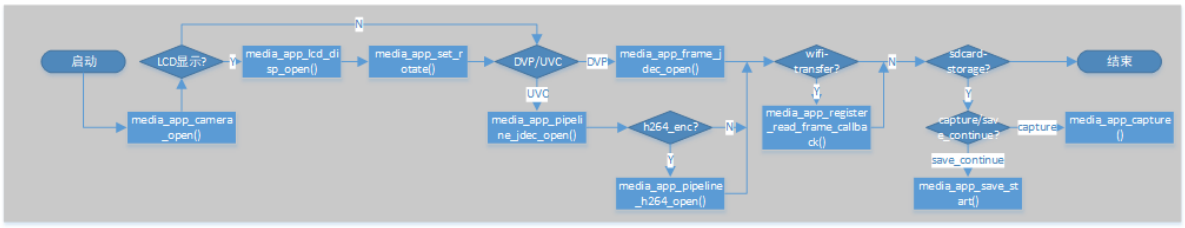


Fig. 6: Figure 3. doorbell video open diagram

6.2 Camera switch

As shown in the flowchart, when switching cameras, first check if there is currently an active camera. If there is, shut down some of the existing processes, close the currently running camera,then open the new camera, and finally restart the image processing flow at the end.

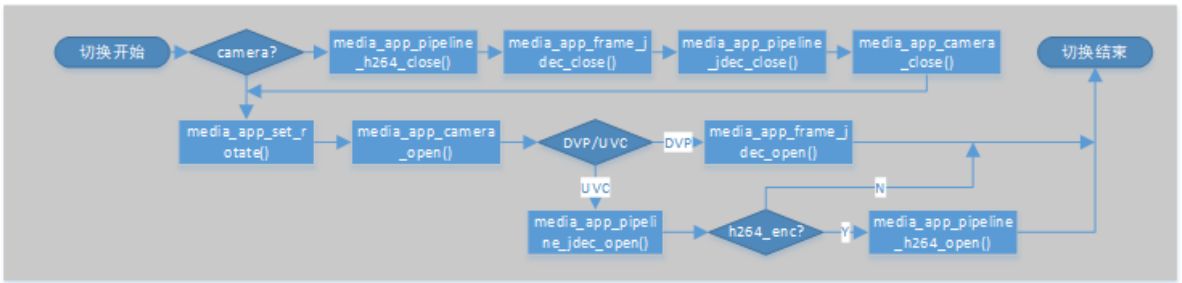


Fig. 7: Figure 4. doorbell camera switch diagram

6.3 Disable video function

When video-related functions are not in use, the image transmission tasks, image storage tasks, image processing tasks, screen display tasks, and all peripherals (such as cameras/screens) should be closed. By default, all video functions will be turned off, but customers can disable specific functions based on their own needs.

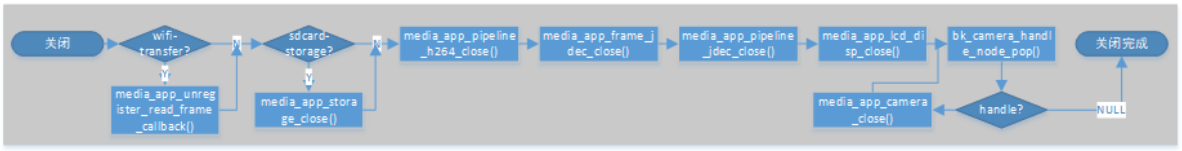


Fig. 8: Figure 5. doorbell video close diagram

7 frame_buffer

For complete image data in multimedia, all are stored in PSRAM and stored in the “frame_buffer_t” structure. And managed in the form of a linked list stream, with the specific structure as follows:

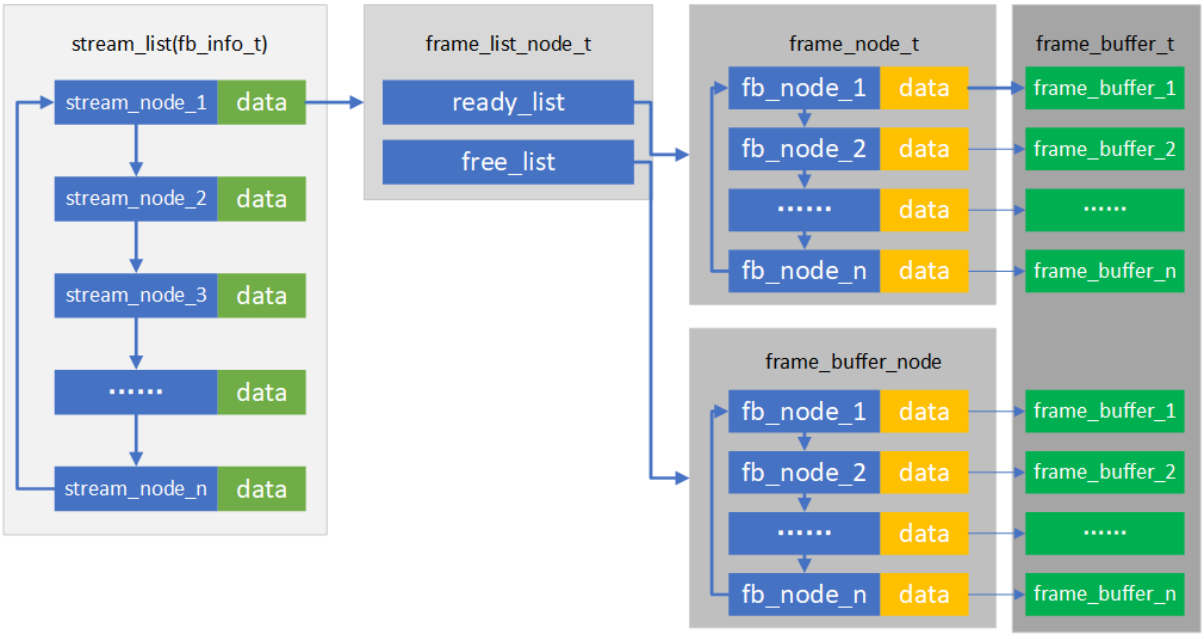


Fig. 9: Figure 6. frame_buffer stream list diagram

10.2.2 Doorbell_4M

1. Introduction

This project is a demo of a USB camera door lock, supporting end-to-end (BK7258 device) to mobile app demonstrations. By default, it supports Shangyun for network transmission.

1.1 Specifications

- **Hardware configuration:**
  - Core board, **BK7258_QFN88_9X9_V3.2**
  - Display adapter board, **BK7258_LCD_interface_V3.0**
  - Speaker small board, **BK_Module_Speaker_V1.1**
  - PSRAM 8M/16M
- **Support, UVC**
  - Reference peripherals, UVC resolution of **864 * 480**
- Support, UAC
- Support, TCP LAN image transmission
- Support UDP LAN image transmission
- Support, Shangyun, P2P image transfer
- **Support, LCD RGB/MCU I8080 display**
  - Reference peripherals, **ST7701SN**, 480 * 854 RGB LCD
  - RGB565/RGB888
- **Support, hardware/software rotation**
  - 0°, 90°, 180°, 270°
- Support, onboard speaker
- **Support, MJPEG hardware decoding**
  - YUV422
- **Support, MJPEG software decoding**
  - YUV420
- Support, H264 hardware decoding
- **Support, OSD display**
  - ARGB888[PNG]
  - Custom Font

1.2 Path

<bk_avdk source code path>/projects/media/doorbell_4M

2. Framework diagram

Please refer to Framework diagram

3. Configuration

Please refer to Configuration

3.1 Differences

The difference between doorbell_4M and doorbell_8M is that the former does not support DVP cameras and does not support the on-board microphone.

The macros supporting 4M are configured on CPU1, The path is media/doorbell_4M/config/bk7258_cp1/config, and the macro configuration differences between the two projects are as follows:

project	marco	value	implication
doorbell_4M	CONFIG_MEDIA_PSRAM_SIZE_4M	Y	PSRAM is 4M
doorbell_8M	CONFIG_MEDIA_PSRAM_SIZE_4M	N	PSRAM is 8M

The allocation of PSRAM with different sizes is shown in the following table. The config file needs to be modified, with the file path as follows:

media/doorbell_4M/config/bk7258/config

media/doorbell_4M/config/bk7258_cp1/config

media/doorbell_4M/config/bk7258_cp2/config

project	doorbell_4M	doorbell_8M
CONFIG_PSRAM_MEM_SLAB_USER_SIZE	0	102400
CONFIG_PSRAM_MEM_SLAB_AUDIO_SIZE	51200	102400
CONFIG_PSRAM_MEM_SLAB_ENCODE_SIZE	946176	1433600
CONFIG_PSRAM_MEM_SLAB_DISPLAY_SIZE	2490368	5701632
CONFIG_PSRAM_HEAP_SIZE (CP0)	0x7D000	0x80000
CONFIG_PSRAM_HEAP_SIZE (CP1)	0x2F000	0x80000

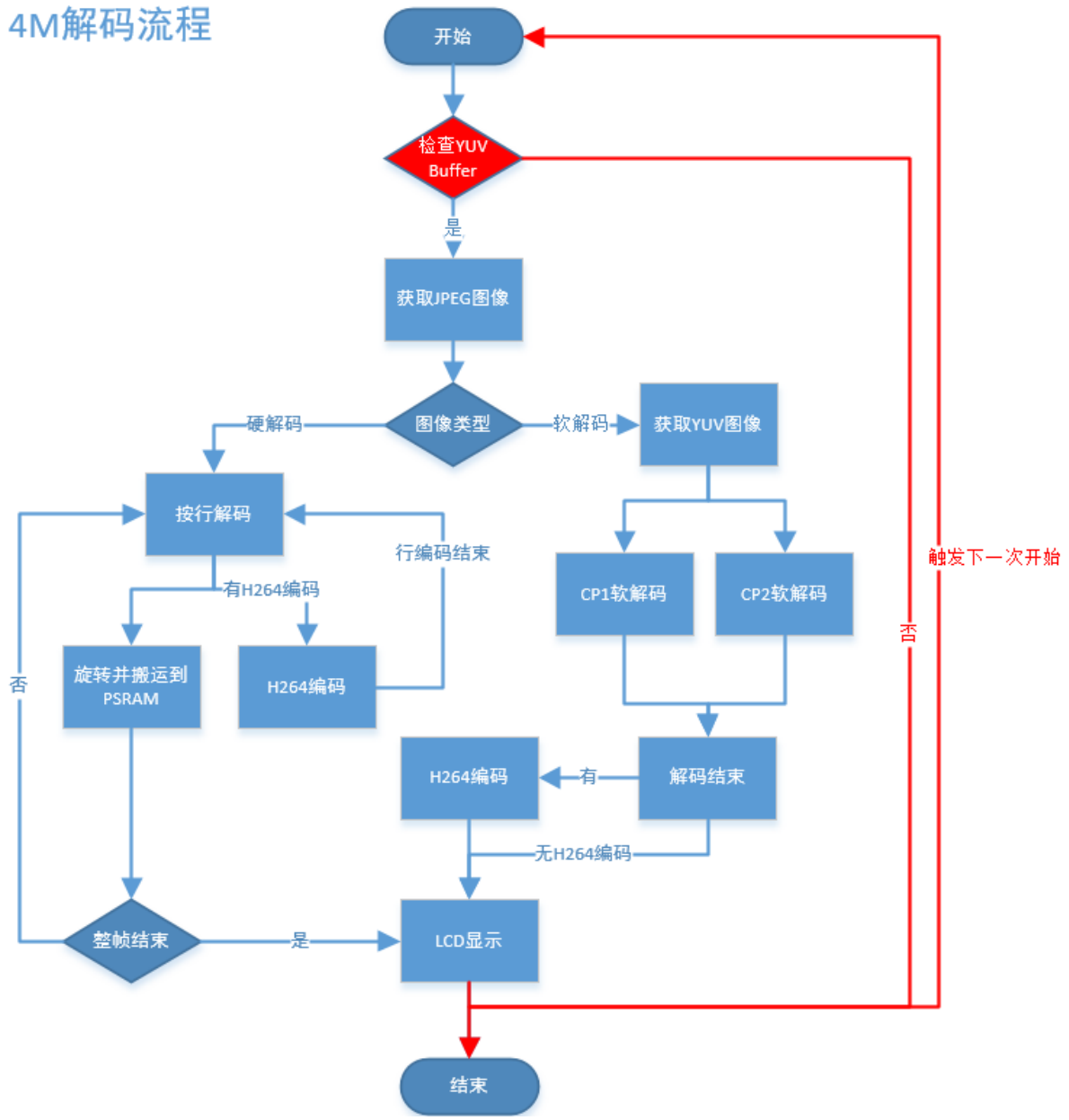
The flow is as shown in the following diagram:

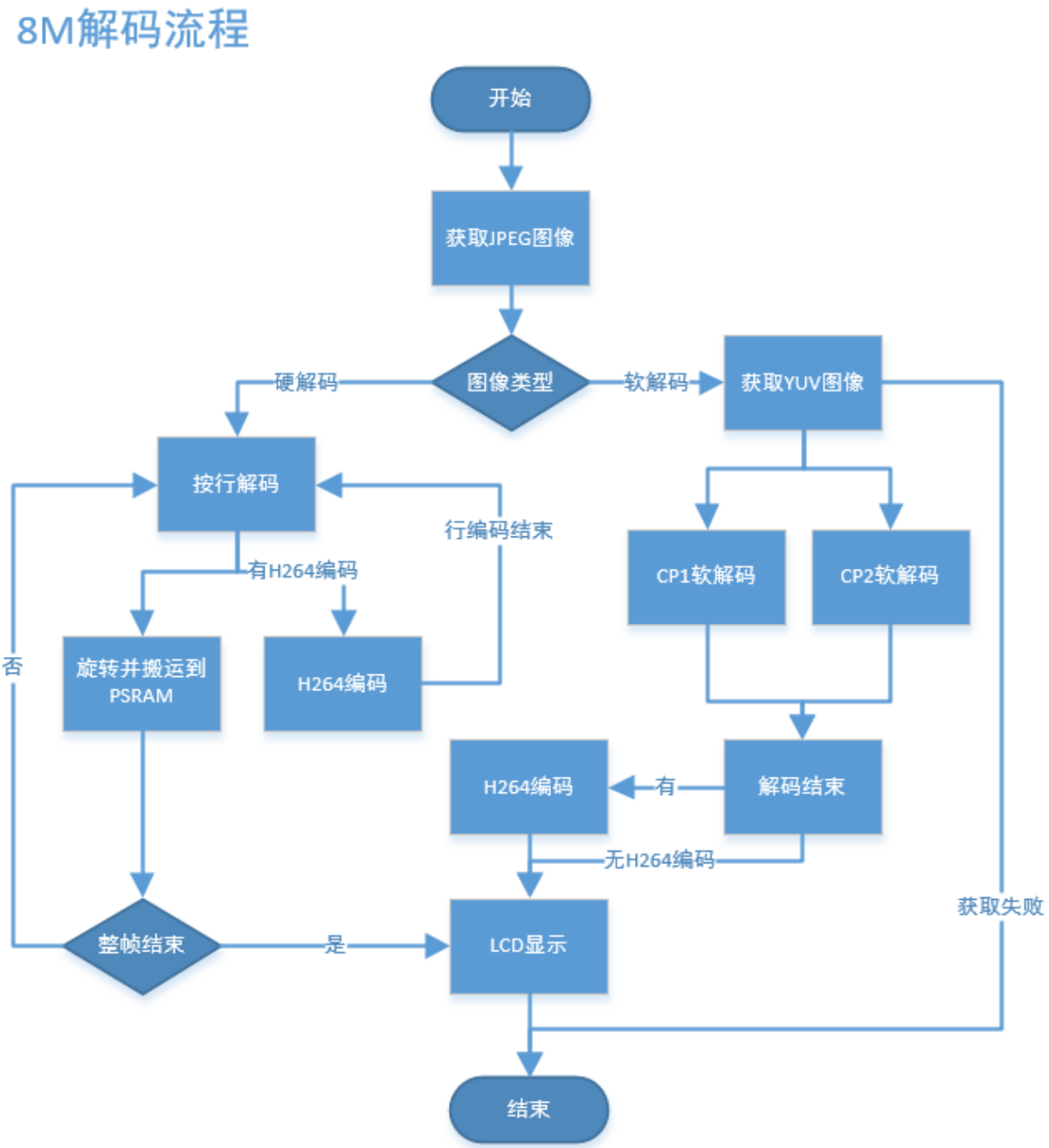
Figure 1. doorbell_4M decode process

Figure 2. doorbell_8M decode process

The main differences in the process are as shown in the following table:

project	decode process
doorbell_4M	Firstly, attempt to obtain the YUV image, and continue with the decoding only after the allocation is successful. The LCD display triggers the next image capture process immediately after completion.
doorbell_8M	Directly decode, and upon failure to obtain the YUV image, immediately release the JPEG and wait for the next frame of JPEG.


4M解码流程
开始
检查YUV Buffer
是
获取JPEG图像
图像类型
硬解码
软解码
获取YUV图像
按行解码
行编码结束
有H264编码
CP1软解码
CP2软解码
触发下一次开始
否
旋转并搬运到PSRAM
H264编码
否
H264编码
有
解码结束
无H264编码
整帧结束
是
LCD显示
结束


8M解码流程
开始
获取JPEG图像
图像类型
硬解码
软解码
获取YUV图像
按行解码
行编码结束
有H264编码
CP1软解码
CP2软解码
否
旋转并搬运到PSRAM
H264编码
H264编码
有
解码结束
无H264编码
获取失败
整帧结束
是
LCD显示
结束

4. Demonstration explanation

Please visit APP Usage Document

Demo result: During runtime, UVC, LCD, and AUDIO will be activated. The LCD will display UVC and output JPEG (864X480) images that have been decoded and rotated 90 degrees before being displayed on the LCD (480X854), After decoding, the YUV is encoded with H264 and transmitted to the mobile phone for display via WIFI (864X480).

Hint: If you do not have cloud account permissions, you can use debug mode to set the local area network TCP/UDP image transmission method.

---

5. Code explanation

Please refer to Code explanation

6. Porting Instructions

For the media module, the biggest difference between the 4M and 8M configurations is the reduction in PSRAM size, which in turn reduces the number of internal buffer images, as shown in the following table:

project	YUV images	JPEG images	H264 images
doorbell_4M	3	4	4
doorbell_8M	5	4	8

To modify the project from 8M FLASH + 8M PSRAM to 4M FLASH + 4M PSRAM, follow the steps below:

Step 1:

Merge the platform code into the project.

Synchronize modifications according to the patch, with the patch commit title being “adapter for new 4+4 psram of W955D8MKY”,

There are a total of four commits, and the code directory and involved files are as shown in the following table:

Step 2:

Synchronize the modifications according to the patch. The patch’s commit title is “PSRAM configuration for image transmission-related buffers when the size is 4M.”

There are three commits in total, and the code directory and involved files are as shown in the following table:

Code Directory	Involved Files
components	display_service/src/lcd_display_service.c media_utils/src/psram_mem_slab.c multimedia/comm/frame_buffer.c multimedia/Kconfig multimedia/pipeline/h264_encode_pipeline.c multimedia/pipeline/h264_encode_pipeline.c multimedia/pipeline/jpeg_get_pipeline.c
bk_idk/components/bk_partition	CMakeLists.txt part_table.mk
projects	media/doorbell_4M/config/bk7258_cp1/config media/doorbell_4M/config/bk7258_cp2/config media/doorbell_4M/config/bk7258/config media/doorbell_4M/config/bk7258/bk7258_partitions.csv

Key modification points are as shown in the following table:

Involved Files	Key modification points
display_service/src/lcd_display_service.c	Retrieve JPEG image immediately after the display is complete
media_utils/src/psram_mem_slab.c	Prevent circular search in the buffer
multimedia/comm/frame_buffer.c	Reduce the number of internal image buffers
multimedia/Kconfig	Enable the macro CONFIG_MEDIA_PSRAM_SIZE_4M
multimedia/pipeline/h264_encode_pipeline.c multimedia/pipeline/h264_encode_pipeline.c multimedia/pipeline/jpeg_get_pipeline.c	Modify the pipeline flow Reduce the impact of YUV image resizing on the frame rate of the software decoding
CMakeLists.txt part_table.mk	add doorbell_4M project
media/doorbell_4M/config/bk7258_cp1/config media/doorbell_4M/config/bk7258_cp2/config media/doorbell_4M/config/bk7258/config	Enable the macro CONFIG_MEDIA_PSRAM_SIZE_4M Modify PSRAM allocation, moving the USB to run on CP1 The specific modifications to the PSRAM are detailed in the table above.
media/doorbell_4M/config/bk7258/bk7258_partitions.csv	Modify the FLASH space allocation to 4M

10.2.3 Doorbell_ab_4M

1. Introduction

This project is a demo of a USB camera door lock, supporting end-to-end (BK7258 device) to mobile app demonstrations. By default, it supports Shangyun for network transmission.

1.1 Specifications

- **Hardware configuration:**
  - Core board, **BK7258_QFN88_9X9_V3.2**
  - Display adapter board, **BK7258_LCD_interface_V3.0**
  - Speaker small board, **BK_Module_Speaker_V1.1**
  - PSRAM 8M/16M
- **Support, UVC**
  - Reference peripherals, UVC resolution of **864 * 480**
- Support, UAC
- Support, TCP LAN image transmission
- Support UDP LAN image transmission
- Support, Shangyun, P2P image transfer
- **Support, LCD RGB/MCU I8080 display**
  - Reference peripherals, **ST7701SN**, 480 * 854 RGB LCD
  - RGB565/RGB888
- **Support, hardware/software rotation**
  - 0°, 90°, 180°, 270°
- Support, onboard speaker
- **Support, MJPEG hardware decoding**
  - YUV422
- **Support, MJPEG software decoding**
  - YUV420
- Support, H264 hardware decoding
- **Support, OSD display**
  - ARGB888[PNG]
  - Custom Font

1.2 Path

<bk_avdk source code path>/projects/media/doorbell_ab_4M

2. Framework diagram

Please refer to Framework diagram

3. Configuration

Please refer to Configuration

3.1 Differences

The difference between doorbell_ab_4M and doorbell_8M is that the former does not support DVP camera, nor does it support onboard MIC. The PSRAM size and model are also inconsistent.

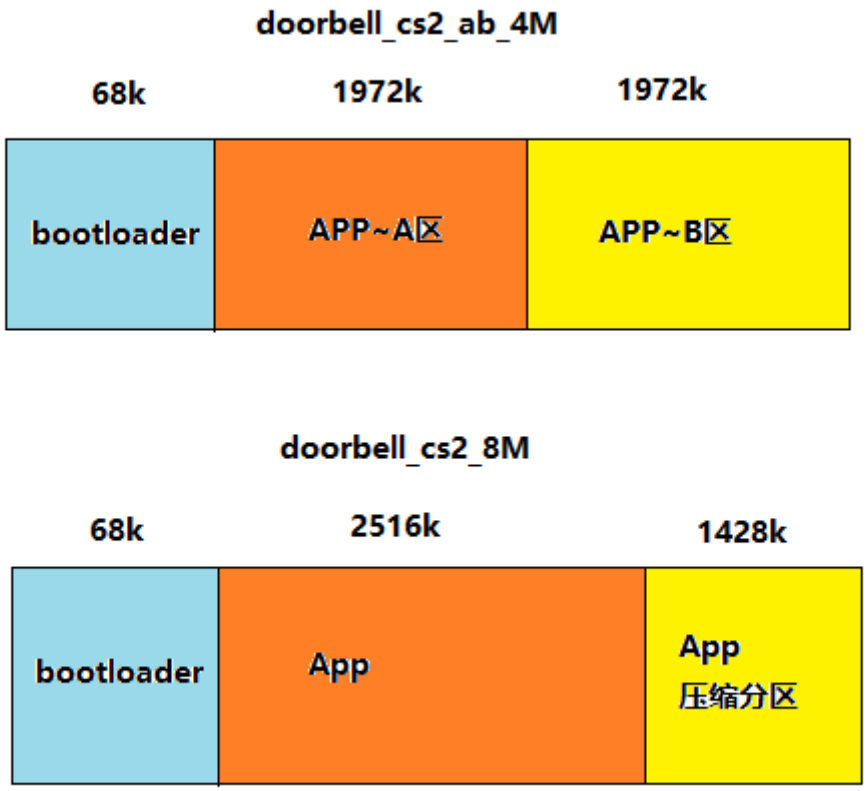


Figure 1. The main partition difference diagram of 4M & 8M

To modify the PSRAM size from 8M to 4M_ab, you need to make changes to the config file. The file path is:

doorbell_ab_4M/config/bk7258/config

doorbell_ab_4M/config/bk7258_cp1/config

doorbell_ab_4M/config/bk7258_cp2/config

The main differences in configuration parameters between doorbell_8M and doorbell_ab_4M are as follows (key configurations with no differences are also listed):

project	door-bell_8M	door-bell_ab_4M
CONFIG_PSRAM_MEM_SLAB_USER_SIZE	102400	0
CONFIG_PSRAM_MEM_SLAB_AUDIO_SIZE	102400	51200
CONFIG_PSRAM_MEM_SLAB_ENCODE_SIZE	1433600	946176
CONFIG_PSRAM_MEM_SLAB_DISPLAY_SIZE	5701632	2490368
CONFIG_MEDIA_PSRAM_SIZE_4M	N	Y
CONFIG_BUCK_ANALOG_DISABLE	N	Y
CONFIG_PSRAM_W955D8MKY_5J	N	Y
CONFIG_PSRAM_APS6408L_O	N	N
CONFIG_CPU0_SPE_RAM_SIZE	0X56000	0X5E000
CONFIG_CPU1_APP_RAM_SIZE	0X3F000	0X38000
CONFIG_CPU2_APP_RAM_SIZE	0XB000	0XA000
CONFIG_PSRAM_HEAP_BASE	0x60700000	0x60354000
CONFIG_PSRAM_HEAP_SIZE	0x80000	0x7D000
CONFIG_PSRAM_HEAP_CPU0_BASE_ADDER	0x60700000	0x60354000
CONFIG_H264_P_FRAME_CNT	5	3
CONFIG_ALI_MQTT	N	N

The flow is as shown in the following diagram:

Figure 2. doorbell_ab_4M decode process

Figure 3. doorbell_8M decode process

The main differences in the process are as shown in the following table:

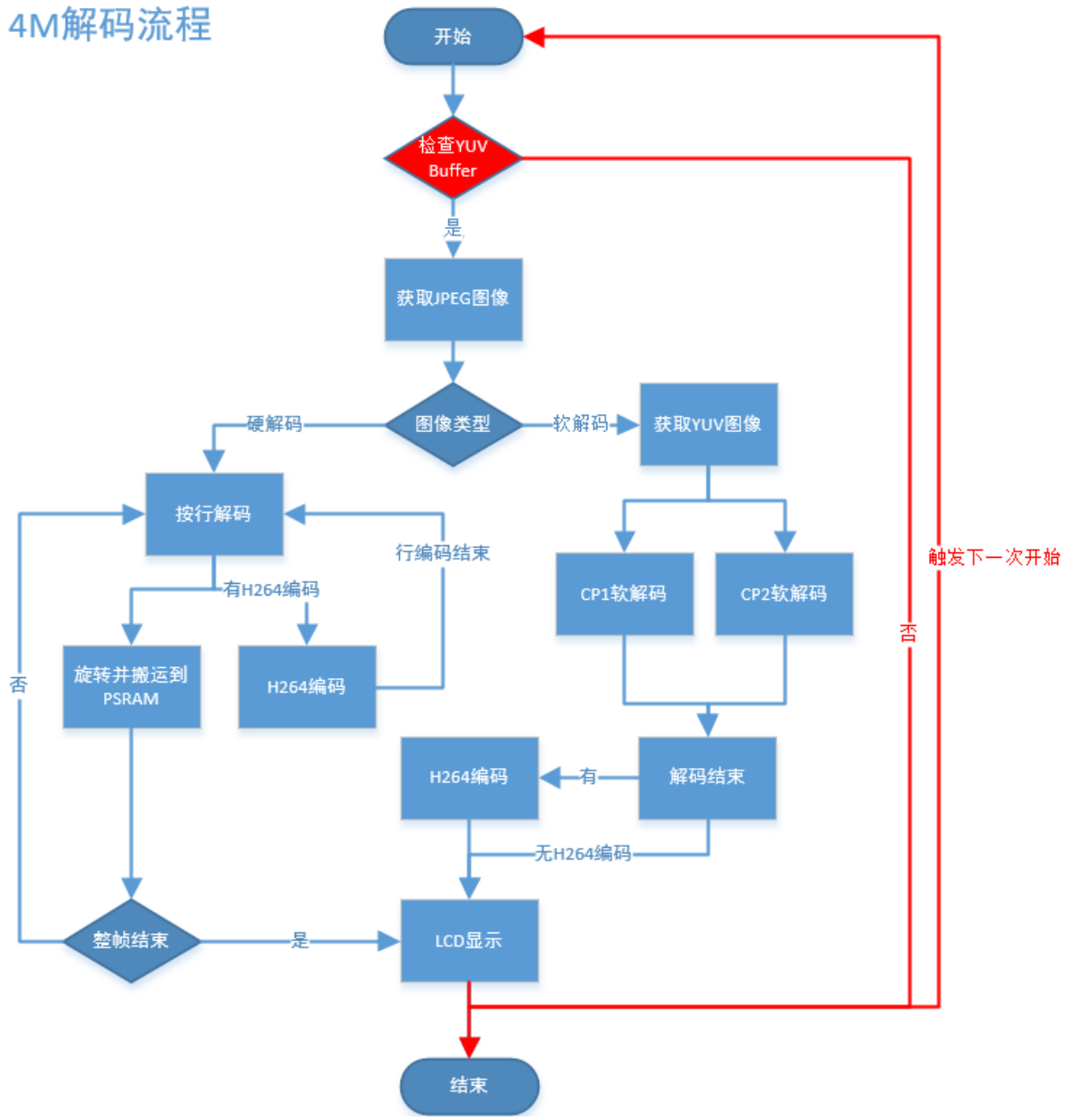
project	decode process
door-bell_ab_4M	Firstly, attempt to obtain the YUV image, and continue with the decoding only after the allocation is successful. The LCD display triggers the next image capture process immediately after completion.
door-bell_8M	Directly decode, and upon failure to obtain the YUV image, immediately release the JPEG and wait for the next frame of JPEG.

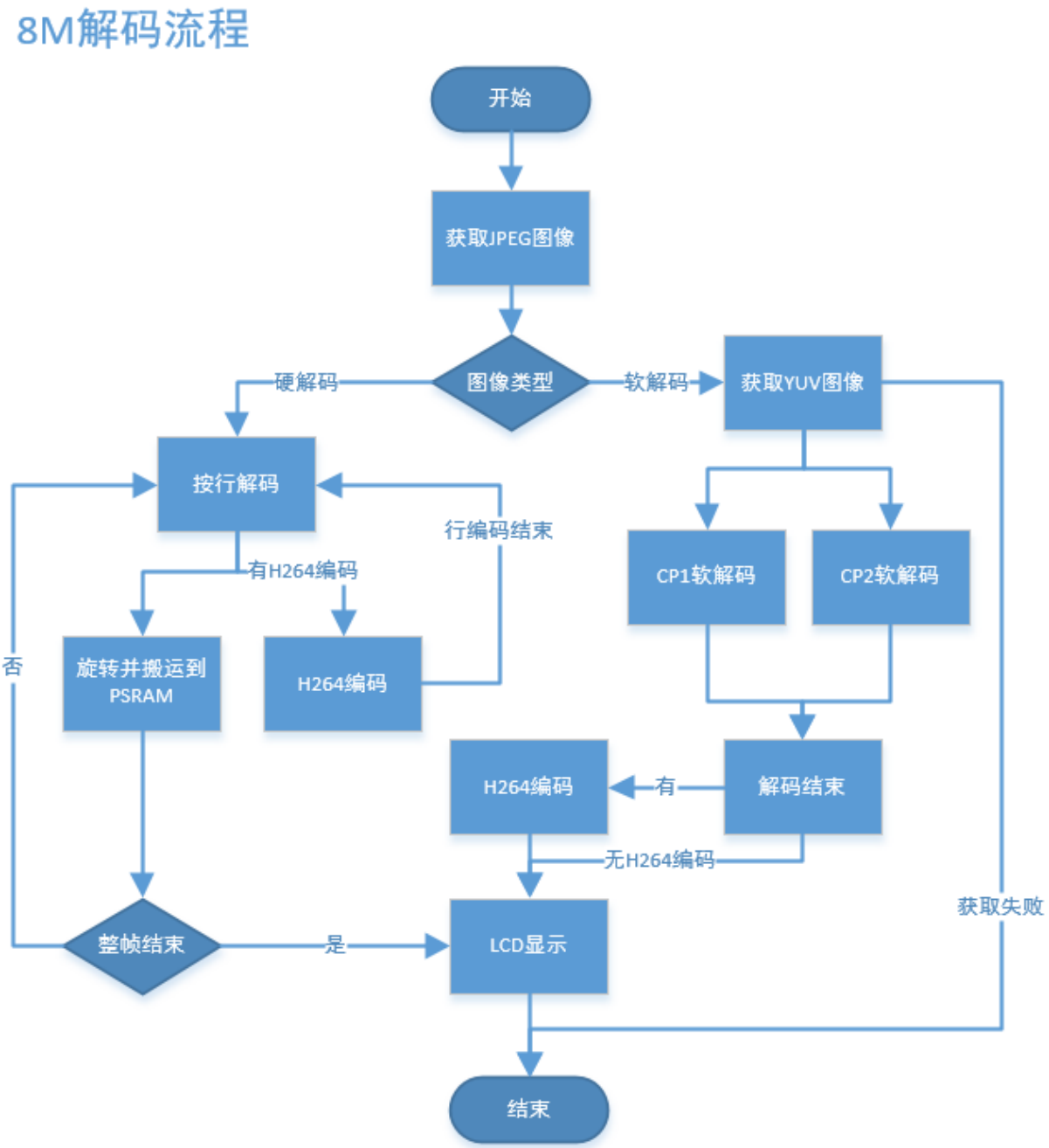
4. Demonstration explanation

Please visit APP Usage Document

Demo result: During runtime, UVC, LCD, and AUDIO will be activated. The LCD will display UVC and output JPEG (864X480) images that have been decoded and rotated 90 degrees before being displayed on the LCD (480X854), After decoding, the YUV is encoded with H264 and transmitted to the mobile phone for display via WIFI (864X480).

Hint: If you do not have cloud account permissions, you can use debug mode to set the local area network TCP/UDP image transmission method.


4M解码流程
开始
检查YUV Buffer
是
获取JPEG图像
图像类型
硬解码
软解码
获取YUV图像
按行解码
行编码结束
有H264编码
CP1软解码
CP2软解码
触发下一次开始
否
旋转并搬运到PSRAM
H264编码
否
H264编码
有
解码结束
无H264编码
整帧结束
是
LCD显示
结束


8M解码流程
开始
获取JPEG图像
图像类型
硬解码
软解码
获取YUV图像
按行解码
行编码结束
有H264编码
CP1软解码
CP2软解码
否
旋转并搬运到PSRAM
H264编码
H264编码
有
解码结束
无H264编码
获取失败
整帧结束
是
LCD显示
结束

5. Code explanation

Please refer to Code explanation

6. Porting Instructions

For the media module, the biggest difference between the 4M and 8M configurations is the reduction in PSRAM size, which in turn reduces the number of internal buffer images, as shown in the following table:

project	YUV images	JPEG images	H264 images
doorbell_ab_4M	3	4	4
doorbell_8M	5	4	8

To modify the project from 8M FLASH + 8M PSRAM to 4M FLASH + 4M PSRAM, follow the steps below:

Step 1:

Merge the platform code into the project.

Synchronize modifications according to the patch, with the patch commit title being “adapter for new 4+4 psram of W955D8MKY”,

There are a total of four commits, and the code directory and involved files are as shown in the following table:

Code Directory	Involved Files
middleware	driver/pwr_clk/Kconfig soc/bk7258/hal/sys_pm_hal.c soc/common/hal/include/psram_hal.h soc/common/hal/psram_hal.c
tools/build_tools	tools/build_tools/otherScript/special_project_deal.py
bk_idk/components/partition_table	CMakeLists.txt part_table.mk
projects	media/doorbell_ab_4M/CMakeLists.txt media/doorbell_ab_4M/config/bk7258_cp1/config media/doorbell_ab_4M/config/bk7258_cp2/config media/doorbell_ab_4M/config/bk7258/ab_position_independent.csv media/doorbell_ab_4M/config/bk7258/bk7258_partitions.csv media/doorbell_ab_4M/config/bk7258/config media/doorbell_ab_4M/config/bk7258/configuration.json media/doorbell_ab_4M/config/bk7258/configurationab.json media/doorbell_ab_4M/config/bk7258/partitions.csv media/doorbell_ab_4M/config/ota_rbl.config media/doorbell_ab_4M/main/app_main.c media/doorbell_ab_4M/main/CMakeLists.txt media/doorbell_ab_4M/main/Kconfig.projbuild media/doorbell_ab_4M/main/vendor_flash.c media/doorbell_ab_4M/main/vendor_flash_partition.h media/doorbell_ab_4M/pj_config.mk media/doorbell_ab_4M/README.md

Key modification points are as shown in the following table:

Involved Files	Key modification points
driver/pwr_clk/Kconfig	Introduce BUCK_ANALOG_DISABLE macro to disable analog domain BUCK
soc/bk7258/hal/sys_pm_hal.c	Configuring the actual code to disable the mock domain BUCK
soc/common/hal/include/psram_hal.h	Add new configuration mode and ID information for increasing 4M PSRAM
soc/common/hal/psram_hal.c	Adding the initialization process for 4M PSRAM
part_table_tools/otherScript/special_project_deal.py	Add compilation processing for the doorbell_ab_4M partition project
CMakeLists.txt part_table.mk	add doorbell_ab_4M project
media/doorbell_ab_4M/config/bk7258_cp1/config tmedia/doorbell_ab_4M/config/bk7258_cp2/config media/doorbell_ab_4M/config/bk7258/config	Enable the macro CONFIG_MEDIA_PSRAM_SIZE_4M
media/doorbell_ab_4M/config/bk7258/bk7258_partitions.csv	Modify the FLASH space allocation to 4M

10.2.4 Doorbell_8M

1. Introduction

This project is a demo of a USB camera door lock, supporting end-to-end (BK7258 device) to mobile app demonstrations. Support LVGL and multi camera switch, 1 dvp camera and 2 uvc (need connect with usb hub). The default PSRAM used is 8Mbyte.

1.1 Specifications

Please refer to Specifications

1.2 Path

<bk_avdk source code path>/projects/media/doorbell_8M

2. Framework diagram

Please refer to Framework diagram

3. Configuration

Please refer to Configuration

3.1 Differences

The difference between doorbell_8M and doorbell is that the previous configuration uses 8M PSRAM, while the latter configuration uses 16M PSRAM. The basic macro configuration of PSRAM on doorbell_8M project CPU0 is as follows:

marco	value	implication
CONFIG_PSRAM	Y	Enable PSRAM module
CONFIG_PSRAM_AS_SYS_MEMORY	Y	Enable PSRAM as Heap
CONFIG_USE_PSRAM_HEAP_AT_SRAM_OOM	N	Enable if cannot malloc mem from sram, can malloc from psram
CONFIG_PSRAM_HEAP_BASE	0x60700000	The start addres of psram as heap in current cpu(cpu0)
CONFIG_PSRAM_HEAP_SIZE	0x80000	The size of psram as heap in current cpu
CONFIG_PSRAM_HEAP_CPU0_BASE_ADDER	0x60700000	The start address of psram as heap

The basic macro configuration of PSRAM on doorbell_8M project CPU1 is as follows:

marco	value	implication
CONFIG_PSRAM	Y	Enable PSRAM module
CONFIG_PSRAM_AS_SYS_MEMORY	Y	Enable PSRAM as Heap
CONFIG_USE_PSRAM_HEAP_AT_SRAM_OOM	N	Enable if cannot malloc mem from sram, can malloc from psram
CONFIG_PSRAM_HEAP_BASE	0x60780000	The start addres of psram as heap in current cpu(cpu1)
CONFIG_PSRAM_HEAP_SIZE	0x80000	The size of psram as heap in current cpu
CONFIG_PSRAM_HEAP_CPU0_BASE_ADDER	0x60700000	The start address of psram as heap

From the above analysis, it can be concluded that for 8M psram, 0x60700000(CONFIG-PSRAM_SEAP_CPU0_SASE-ADDER) - 0x6800000 is used as a heap, On both cores, 0x80000 was allocated.

Warning: Please pay attention to whether the project and PSRAM size (model) are compatible when using it. If a 16M PSRAM configuration is used to run on an 8M board, the PSRAM used may exceed the maximum range, which may cause a dump.

4. Demonstration explanation

Please visit APP Usage Document

Demo result: During runtime, UVC, LCD, and AUDIO will be activated. The LCD will display UVC and output JPEG (864X480) images that have been decoded and rotated 90 degrees before being displayed on the LCD (480X854), After decoding, the YUV is encoded with H264 and transmitted to the mobile phone for display via WIFI (864X480).

Hint: If you do not have cloud account permissions, you can use debug mode to set the local area network TCP image transmission method.

---

5. Code explanation

Please refer to Code explanation

10.2.5 Doorbell_720p

1. Introduction

This project is a demo of a USB camera door lock, supporting end-to-end (BK7258 device) to mobile app demonstrations. The default PSRAM used is 8Mbyte. Support uvc output 1280X720 image transfer to mobilephone, and scale display to 480X854 lcd screen. This project no adapt to camera switch funcion. Customer can reference project “doorbell” if need.

1.1 Specifications

Please refer to Specifications

1.2 Path

<bk_avdk source code path>/projects/media/doorbell_720p

2. Framework diagram

Please refer to Framework diagram

3. Configuration

Please refer to Configuration

3.1 Differences

The difference between doorbell_720p and doorbell is that:

- the previous configuration uvc output 1280X720 mjpeg image, while the latter configuration uvc output 864X480 mjpeg image.

Because the multimedia module runs on CPU1, the macro configurations of the two projects differ as follows:

project	marco	value	implication
doorbell	CONFIG_BT_REUSE_MEDIA_MEMORY	Y	Multimedia and Bluetooth share one SRAM (time-division multiplexing)
doorbell_720p	CONFIG_BT_REUSE_MEDIA_MEMORY	Y	Multimedia and Bluetooth share one SRAM (time-division multiplexing)
doorbell	CONFIG_BT_REUSE_MEDIA_MEM_SIZE	0x1B000	Multimedia and Bluetooth share the same SRAM size
doorbell_720p	CONFIG_BT_REUSE_MEDIA_MEM_SIZE	0x2F000	Multimedia and Bluetooth share the same SRAM size
doorbell	CONFIG_SUPPORTED_IMAGE_MAX_720P	N	Does not support maximum image resolution of 1280X720
doorbell_720p	CONFIG_SUPPORTED_IMAGE_MAX_720P	Y	Support maximum image resolution of 1280X720

4. Demonstration explanation

Please visit APP Usage Document

Demo result: During runtime, UVC, LCD, and AUDIO will be activated. The LCD will display UVC and output JPEG (1280X720) images that have been decoded and rotated 90° before being displayed on the LCD (480X854), After decoding, the YUV is encoded with H264 and transmitted to the mobile phone for display via WIFI (1280X720).

Hint: If you do not have cloud account permissions, you can use debug mode to set the local area network TCP image transmission method.

---

5. Code explanation

Please refer to Code explanation

10.2.6 Audio Player

1 Function Overview

The project shows the function of network music player, which mainly realizes the function of network music download and play, pause, continue to play, set the volume, the previous song and the next song.

2 Code Path

Demo path: `./projects/media/doorbell/main/src/doorbell_udp_service.c`

Compile command: `make bk7258 PROJECT=media/audio_player`

3 Cli Overview

The project first needs to send command `sta {ssid} {password}` to connect to the available network, and the commands for other functions are as follows:

audio_player start	start play
audio_player pause	pause play
audio_player play	continue play
audio_player {volume_value}	set volume
audio_player last	previous song
audio_player next	next song

The volume value ranges from 0x00 to 0x3F, and the current volume value is 0x2D by default.

10.2.7 Double-Board Intertransfer Mission (Client)

1 Function Overview

This project demonstrates the function of video unidirectional transmission and audio bidirectional transmission between two BK7258 boards; the client side connects to an actual camera with DVP/UVC interface, and supports real-time display of the camera's captured images on the LCD screen connected to the current board.

1.1 Specification

- **Hardware Configuration:**
  - Core board: **BK7258_QFN88_9X9_V3.2**
  - Display adapter board: **BK7258_LCD_Interface_V3.0**
  - Microphone module board: **BK_Module_Microphone_V1.1**
  - Speaker module board: **BK_Module_Speaker_V1.1**
  - PSRAM: 8M/16M
- **Support, UVC**
  - Reference peripheral: **864 * 480** resolution UVC
- Support, UAC
- Support, TCP LAN Image transmission
- Support, UDP LAN Image transmission

- **Support, LCD RGB/MCU I8080 display**
  - Reference peripheral: **ST7701SN**, 480 * 854 RGB LCD
  - RGB565/RGB888
- **Support, Hardware rotation**
  - 0°, 90°, 180°, 270°
- Support, Onboard speaker
- Support, Onboard microphone
- **Support, MJPEG Hardware Decoding**
  - YUV422
- **Support, MJPEG Software Decoding**
  - YUV420

1.2 Path

<bk_avdk Source Code Path>/projects/media/av_client

2. Frame Diagram

2.1 Software Module Architecture Diagram

As shown in the following figure, av_client is responsible for collecting images from the USB camera and transmitting them to av_server via Wi-Fi. At the same time, the av_client's local lcd displays the collected images, while av_server displays the images transmitted by av_client.

- CPU0 runs Wi-Fi/BLE and serves as the low-power consumption CPU.
- CPU1 runs multimedia and serves as the high-performance multimedia CPU.

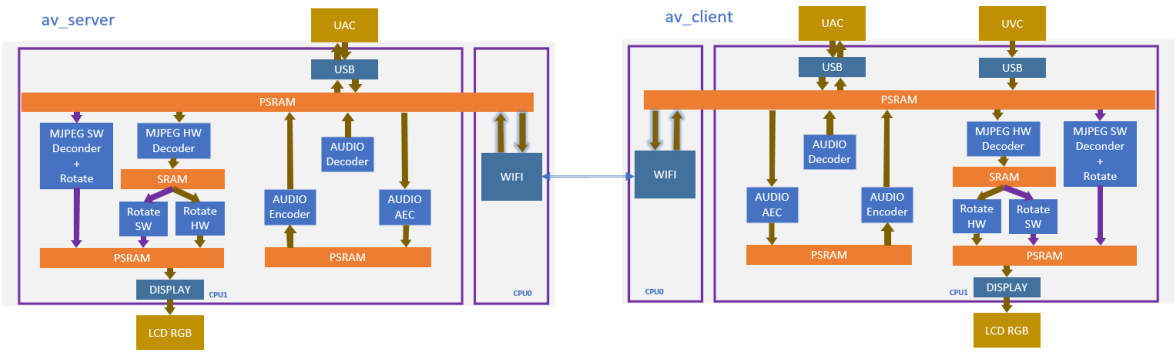


Fig. 10: Figure 1. software module architecture

2.2 Code Module Relationship Diagram

As shown in the figure below, the interfaces for multimedia are all defined in **media_app.h** and **aud_intf.h**.

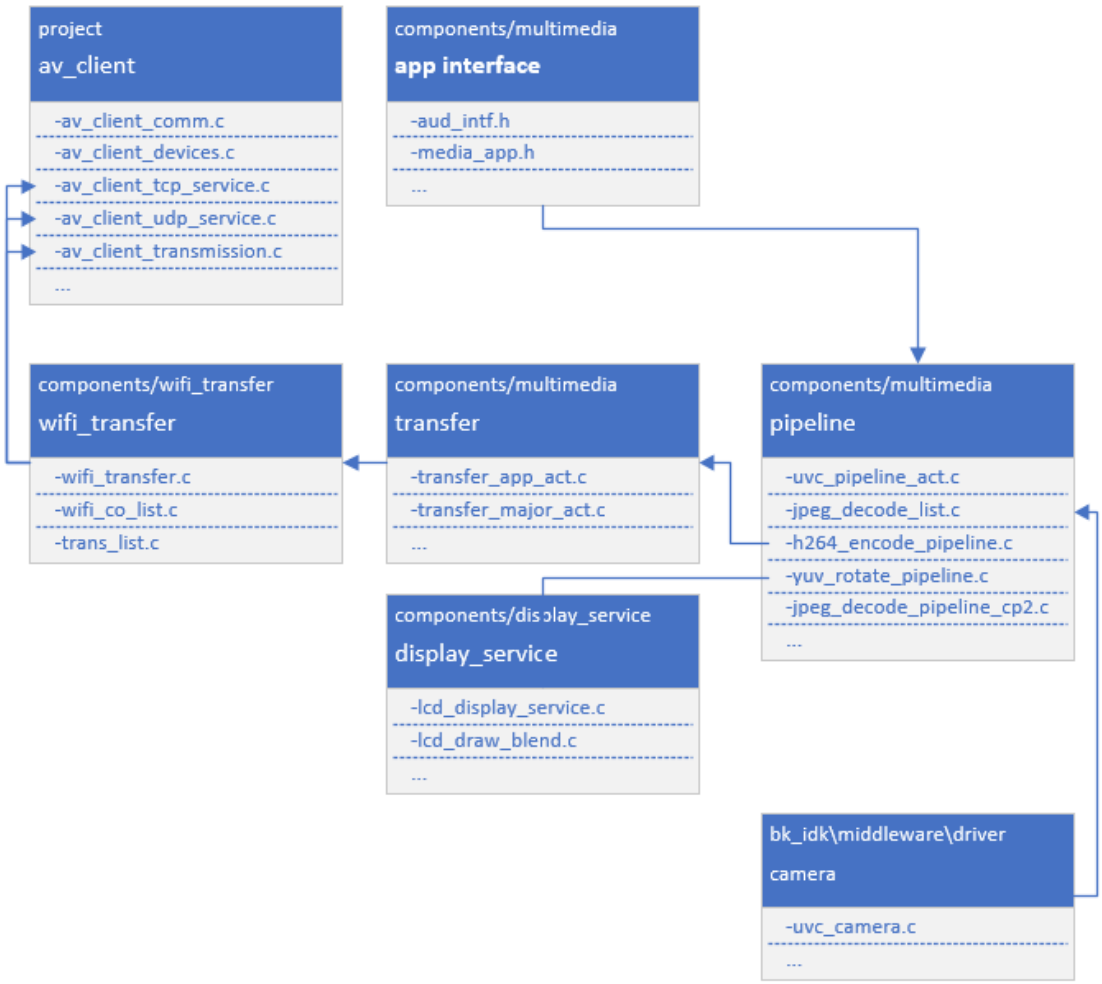


Fig. 11: Figure 2. module relationship diagram

3. Configuration

3.1 Configuration of Image Transmission Mode

Under the engineering path at config/bk7258/config, modify the macros to configure the image transmission mode; the current default transmission mode is UDP.

//Transfer using UDP connections CONFIG_AV_DEMO_MODE_UDP=y CONFIG_AV_DEMO_MODE_TCP=n

//Transfer using TCP connections CONFIG_AV_DEMO_MODE_UDP=n CONFIG_AV_DEMO_MODE_TCP=y

4. Demonstration Instructions

Two development boards are required for the two-board transmission, with av_client downloaded on Board A and av_server on Board B;

Board B acts as an AP, with a fixed SSID of “av_demo”, no password, and needs to be powered on in front of Board A; it requires connection to the LCD.

Board A acts as a STA, fixedly connected to the SSID “av_demo”; it requires connection to the USB camera and LCD.

After powering on Board A, it defaults to connecting to the WiFi on Board B. After successful connection, the camera and LCD on Board A are turned on. Once Board A is successfully connected, the LCD on Board B displays the image output from the USB on Board A.

5. Code Explanation

5.1 UVC Camera

Supported Peripherals, please refer to Supported Peripherals

5.1.1 Open UVC

5.1.1.1 Application code

```
//Path      :  projects/media/av_client/main/src/av_client_devices.c
//Loaction  :  CPU0

int av_client_camera_turn_on(camera_parameters_t *parameters)
{
    ...

    //Open UVC Camera
    ret = media_app_camera_open(&device);

    //Set local display rotation
    media_app_pipline_set_rotate(rot_angle);

    ...
}
```

5.1.1.2 Interface Code

```
//Path      :  components/multimedia/app/media_app.c
//Loaction  :  CPU0

bk_err_t media_app_camera_open(media_camera_device_t *device)
{
    ...

    //uninstall Bluetooth
    #ifdef CONFIG_BT_REUSE_MEDIA_MEMORY
    #if CONFIG_BLUETOOTH
            bk_bluetooth_deinit();
```

(continues on next page)

(continued from previous page)

```
    #endif
    #endif

    //Vote to start CPU1. The purpose of the vote is to ensure that CPU1 can be␣
↪automatically turned off when it is not in use, in order to achieve the goal of low␣
↪power consumption.
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_JPEG_EN, PM_POWER_
↪MODULE_STATE_ON);

    //Notify CPU1 to turn on the UVC camera.
    ret = media_send_msg_sync(EVENT_CAM_UVC_OPEN_IND, (uint32_t)device);

    ...
}
```

5.2 LCD Display

Supported Peripherals, please refer to Supported Peripherals

5.2.1 Open LCD

5.2.1.1 Application code

```
//Path      :  projects/media/av_client/main/src/av_client_devices.c
//Loaction  :  CPU0

int av_client_display_turn_on(uint16_t id, uint16_t rotate, uint16_t fmt)
{
    ...

    //Set the pixel format for display
    if (fmt == 0)
    {
        media_app_lcd_fmt(PIXEL_FMT_RGB565_LE);
    }
    else if (fmt == 1)
    {
        media_app_lcd_fmt(PIXEL_FMT_RGB888);
    }

    //Set the rotation angle
    switch (rotate)
    {
        case 90:
            rot_angle = ROTATE_90;
            break;
        case 180:
            rot_angle = ROTATE_180;
            break;
        case 270:
            rot_angle = ROTATE_270;
            break;
        case 0:
```

(continues on next page)

(continued from previous page)

```
        default:
            rot_angle = ROTATE_NONE;
            break;
    }

    media_app_pipline_set_rotate(rot_angle);

    //Open the local LCD display
            media_app_lcd_pipeline_open(&lcd_open);

    ...
}
```

5.2.1.2 Interface Code

```
//Path      :  components/multimedia/app/media_app.c
//Loaction  :  CPU0

bk_err_t media_app_lcd_pipeline_open(void *lcd_open)
{
    ...

    //
    ret = media_app_lcd_pipeline_disp_open(config);

    //
    ret = media_app_lcd_pipeline_jdec_open();

    ...
}

bk_err_t media_app_lcd_pipeline_disp_open(void *config)
{
    ...

    //Vote to start CPU1. The purpose of the vote is to ensure that CPU1 can be␣
↪automatically turned off when it is not in use, in order to achieve the goal of low␣
↪power consumption.
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_LCD, PM_POWER_MODULE_
↪STATE_ON);

    //Notify CPU1 to turn on the LCD.
    ret = media_send_msg_sync(EVENT_PIPELINE_LCD_DISP_OPEN_IND, (uint32_t)ptr);

    ...
}

bk_err_t media_app_lcd_pipeline_jdec_open(void)
{
    int ret = BK_OK;

    //Vote to start CPU1. The purpose of the vote is to ensure that CPU1 can be␣
↪automatically turned off when it is not in use, in order to achieve the goal of low␣
↪power consumption.
```

(continues on next page)

(continued from previous page)

```
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_JPEG_DE, PM_POWER_
↪MODULE_STATE_ON);

    //Set the rotation angle
    ret = media_send_msg_sync(EVENT_PIPELINE_SET_ROTATE_IND, jpeg_decode_pipeline_
↪param.rotate);

    //Turn on rotation, scale, and decode modules.
    ret = media_send_msg_sync(EVENT_PIPELINE_LCD_JDEC_OPEN_IND, 0);

    return ret;
}
```

5.3 Audio

5.3.1 Open UAC or Onboard MIC/SPEAKER

```
//Path      : projects/media/av_client/main/src/av_client_devices.c
//Loaction  :  CPU0

int av_client_audio_turn_on(audio_parameters_t *parameters)
{
    ...

    //Enable AEC
    if (parameters->aec == 1)
    {
        aud_voc_setup.aec_enable = true;
    }
    else
    {
        aud_voc_setup.aec_enable = false;
    }


    //Set the SPEAKER to single-ended mode.
    ud_voc_setup.spk_mode = AUD_DAC_WORK_MODE_SIGNAL_END;

    //Enable UAC
    if (parameters->uac == 1)
    {
        aud_voc_setup.mic_type = AUD_INTF_MIC_TYPE_UAC;
        aud_voc_setup.spk_type = AUD_INTF_SPK_TYPE_UAC;
    }
    else //Enable Onboard MIC and SPEAKER
    {
        aud_voc_setup.mic_type = AUD_INTF_MIC_TYPE_BOARD;
        aud_voc_setup.spk_type = AUD_INTF_SPK_TYPE_BOARD;
    }

    if (aud_voc_setup.mic_type == AUD_INTF_MIC_TYPE_BOARD && aud_voc_setup.spk_type␣
↪== AUD_INTF_SPK_TYPE_BOARD) {
        aud_voc_setup.data_type = parameters->rmt_recoder_fmt - 1;
    }
```

(continues on next page)

(continued from previous page)

```
    //Set the sampling rate
    switch (parameters->rmt_recorder_sample_rate)
    {
        case DB_SAMPLE_RARE_8K:
            aud_voc_setup.samp_rate = 8000;
        break;

        case DB_SAMPLE_RARE_16K:
            aud_voc_setup.samp_rate = 16000;
        break;

        default:
            aud_voc_setup.samp_rate = 8000;
        break;
    }

    //Register MIC data callback
    aud_intf_drv_setup.aud_intf_tx_mic_data = av_client_udp_voice_send_callback;

    ...
}
```

5.3.2 Fetch uplink MIC data

```
//Path      :  projects/media/av_client/main/src/av_client_devices.c
//Loaction  :  CPU0

//Register MIC data callback
    aud_intf_drv_setup.aud_intf_tx_mic_data = av_client_udp_voice_send_callback;
    ret = bk_aud_intf_drv_init(&aud_intf_drv_setup);

int av_client_udp_voice_send_callback(unsigned char *data, unsigned int len)
{
    ...

    //The commonly implemented callback typically involves transmitting in the WiFi␣
↪direction.
    return db_device_info->audio_transfer_cb->send(buffer, len, &retry_cnt);
}
```

5.3.3 Play downlink SPEAKER data

```
//Path      :  projects/media/av_client/main/src/av_client_devices.c
//Loaction  :  CPU0

void av_client_audio_data_callback(uint8_t *data, uint32_t length)
{
    ...

    //Send data to the SPEAKER.
    ret = bk_aud_intf_write_spk_data(data, length);
```

(continues on next page)

(continued from previous page)

```
    ...
}
```

10.2.8 Double-Board Intertransfer Mission (Server)

1 Function Overview

This project demonstrates the functionality of unidirectional video transmission and bidirectional audio transmission between two BK7258 boards; the server side does not connect to an actual camera (dvp/uvc) and supports the current board to connect to an LCD screen for real-time display of the camera images transmitted from the client side.

1.1 Specification

- **Hardware Configuration:**
  - Core board: **BK7258_QFN88_9X9_V3.2**
  - Display adapter board: **BK7258_LCD_Interface_V3.0**
  - Microphone module board: **BK_Module_Microphone_V1.1**
  - Speaker module board: **BK_Module_Speaker_V1.1**
  - PSRAM: 8M/16M
- Support, TCP LAN Image transmission
- Support, UDP LAN Image transmission
- **Support, LCD RGB/MCU I8080 display**
  - Reference peripheral: **ST7701SN**, 480 * 854 RGB LCD
  - RGB565/RGB888
- **Support, Hardware rotation**
  - 0°, 90°, 180°, 270°
- Support, Onboard speaker
- Support, Onboard microphone
- **Support, MJPEG Hardware Decoding**
  - YUV422
- **Support, MJPEG Software Decoding**
  - YUV420

2. Frame Diagram

2.1 Software Module Architecture Diagram

As shown in the following figure, av_client is responsible for collecting images from the USB camera and transmitting them to av_server via Wi-Fi. At the same time, the av_client's local lcd displays the collected images, while av_server displays the images transmitted by av_client.

- CPU0 runs Wi-Fi/BLE and serves as the low-power consumption CPU.
- CPU1 runs multimedia and serves as the high-performance multimedia CPU.

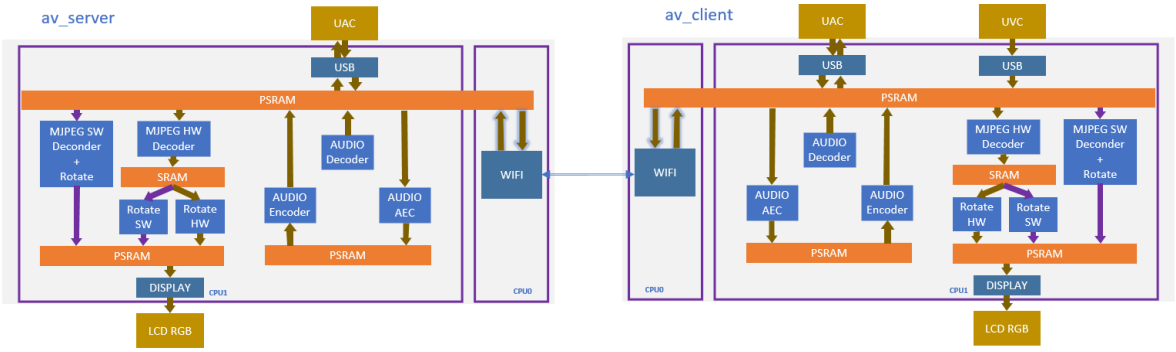


Fig. 12: Figure 1. software module architecture

2.2 Code Module Relationship Diagram

As shown in the figure below, the interfaces for multimedia are all defined in **media_app.h** and **aud_intf.h**.

3. Configuration

3.1 Configuration of Image Transmission Mode

Under the engineering path at config/bk7258/config, modify the macros to configure the image transmission mode; the current default transmission mode is UDP.

//Transfer using UDP connections CONFIG_AV_DEMO_MODE_UDP=y CONFIG_AV_DEMO_MODE_TCP=n

//Transfer using TCP connections CONFIG_AV_DEMO_MODE_UDP=n CONFIG_AV_DEMO_MODE_TCP=y

4. Demonstration Instructions

Two development boards are required for the two-board transmission, with av_client downloaded on Board A and av_server on Board B;

Board B acts as an AP, with a fixed SSID of "av_demo", no password, and needs to be powered on in front of Board A; it requires connection to the LCD.

Board A acts as a STA, fixedly connected to the SSID "av_demo"; it requires connection to the USB camera and LCD.

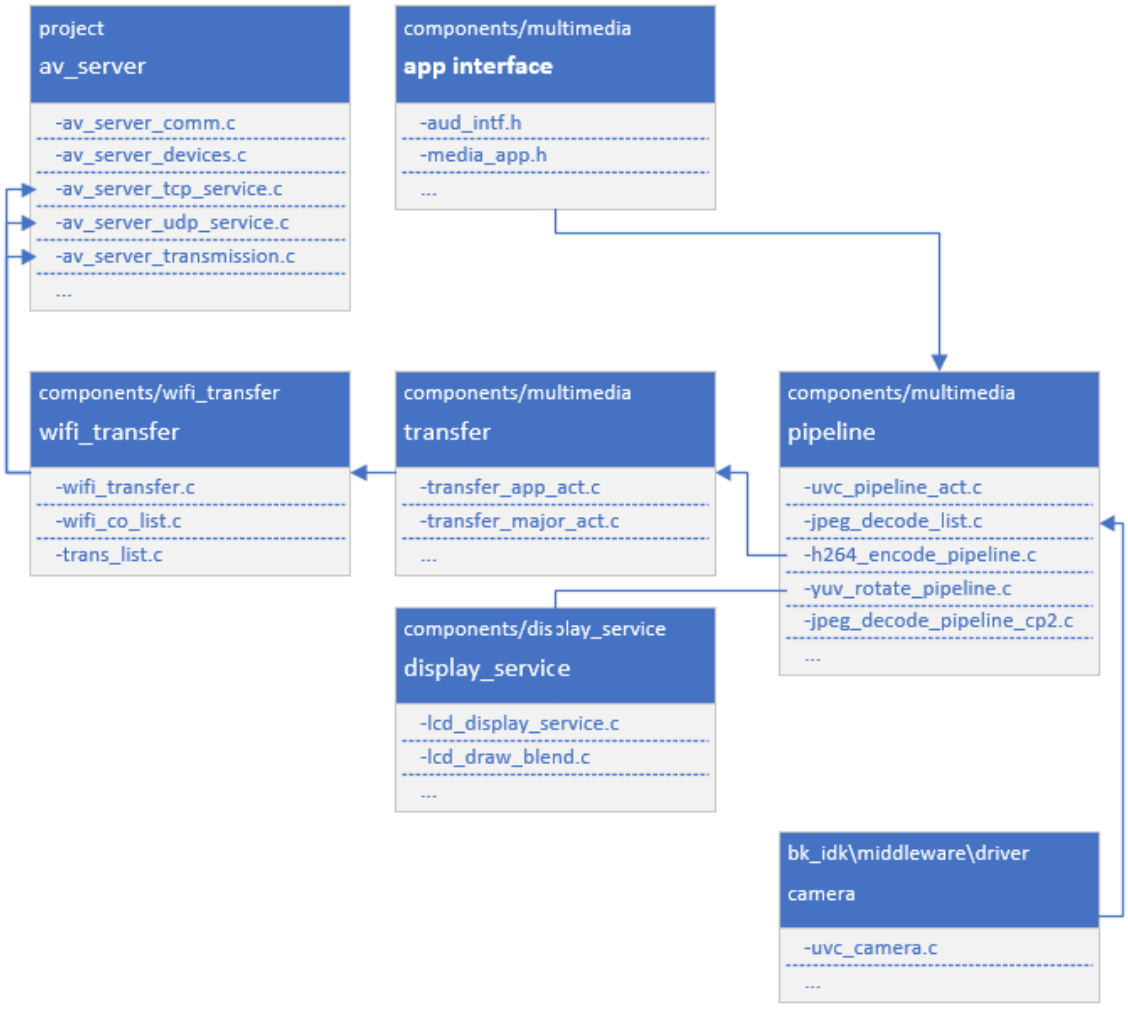


Fig. 13: Figure 2. module relationship diagram

After powering on Board A, it defaults to connecting to the WiFi on Board B. After successful connection, the camera and LCD on Board A are turned on. Once Board A is successfully connected, the LCD on Board B displays the image output from the USB on Board A.

5. Code Explanation

5.1 LCD Display

Supported Peripherals, please refer to Supported Peripherals

5.1.1 Open LCD

5.1.1.1 Application code

```
//Path      :  projects/media/av_server/main/src/av_server_devices.c
//Loaction  :  CPU0

int av_server_display_turn_on(uint16_t id, uint16_t rotate, uint16_t fmt)
{
    ...

    //Set the pixel format for display
    if (fmt == 0)
    {
        media_app_lcd_fmt(PIXEL_FMT_RGB565_LE);
    }
    else if (fmt == 1)
    {
        media_app_lcd_fmt(PIXEL_FMT_RGB888);
    }

    //Set the rotation angle
    switch (rotate)
    {
        case 90:
            rot_angle = ROTATE_90;
            break;
        case 180:
            rot_angle = ROTATE_180;
            break;
        case 270:
            rot_angle = ROTATE_270;
            break;
        case 0:
        default:
            rot_angle = ROTATE_NONE;
            break;
    }

    media_app_pipline_set_rotate(rot_angle);

    //Open the local LCD display
            media_app_lcd_pipeline_open(&lcd_open);
```

(continues on next page)

(continued from previous page)

```
    ...
}
```

5.1.1.2 Interface Code

```
//Path      :  components/multimedia/app/media_app.c
//Loaction  :  CPU0

bk_err_t media_app_lcd_pipeline_open(void *lcd_open)
{
    ...

    //
    ret = media_app_lcd_pipeline_disp_open(config);

    //
    ret = media_app_lcd_pipeline_jdec_open();

    ...
}

bk_err_t media_app_lcd_pipeline_disp_open(void *config)
{
    ...

    //Vote to start CPU1. The purpose of the vote is to ensure that CPU1 can be␣
↪automatically turned off when it is not in use, in order to achieve the goal of low␣
↪power consumption.
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_LCD, PM_POWER_MODULE_
↪STATE_ON);

    //Notify CPU1 to turn on the LCD.
    ret = media_send_msg_sync(EVENT_PIPELINE_LCD_DISP_OPEN_IND, (uint32_t)ptr);

    ...
}

bk_err_t media_app_lcd_pipeline_jdec_open(void)
{
    int ret = BK_OK;

    //Vote to start CPU1. The purpose of the vote is to ensure that CPU1 can be␣
↪automatically turned off when it is not in use, in order to achieve the goal of low␣
↪power consumption.
    bk_pm_module_vote_boot_cp1_ctrl(PM_BOOT_CP1_MODULE_NAME_VIDP_JPEG_DE, PM_POWER_
↪MODULE_STATE_ON);

    //Set the rotation angle
    ret = media_send_msg_sync(EVENT_PIPELINE_SET_ROTATE_IND, jpeg_decode_pipeline_
↪param.rotate);

    //Turn on rotation, scale, and decode modules.
    ret = media_send_msg_sync(EVENT_PIPELINE_LCD_JDEC_OPEN_IND, 0);
```

(continues on next page)

(continued from previous page)

```
    return ret;
}
```

5.2 Audio

5.2.1 Open UAC or Onboard MIC/SPEAKER

```
//Path      :  projects/media/av_client/main/src/av_server_devices.c
//Loaction  :  CPU0

int av_server_audio_turn_on(audio_parameters_t *parameters)
{
    ...

    //Enable AEC
    if (parameters->aec == 1)
    {
        aud_voc_setup.aec_enable = true;
    }
    else
    {
        aud_voc_setup.aec_enable = false;
    }


    //Set the SPEAKER to single-ended mode.
    ud_voc_setup.spk_mode = AUD_DAC_WORK_MODE_SIGNAL_END;

    //Enable UAC
    if (parameters->uac == 1)
    {
        aud_voc_setup.mic_type = AUD_INTF_MIC_TYPE_UAC;
        aud_voc_setup.spk_type = AUD_INTF_SPK_TYPE_UAC;
    }
    else //Enable Onboard MIC and SPEAKER
    {
        aud_voc_setup.mic_type = AUD_INTF_MIC_TYPE_BOARD;
        aud_voc_setup.spk_type = AUD_INTF_SPK_TYPE_BOARD;
    }

    if (aud_voc_setup.mic_type == AUD_INTF_MIC_TYPE_BOARD && aud_voc_setup.spk_type␣
↪== AUD_INTF_SPK_TYPE_BOARD) {
        aud_voc_setup.data_type = parameters->rmt_recoder_fmt - 1;
    }

    //Set the sampling rate
    switch (parameters->rmt_recorder_sample_rate)
    {
        case DB_SAMPLE_RARE_8K:
            aud_voc_setup.samp_rate = 8000;
        break;

        case DB_SAMPLE_RARE_16K:
            aud_voc_setup.samp_rate = 16000;
```

(continues on next page)

(continued from previous page)

```
        break;

        default:
            aud_voc_setup.samp_rate = 8000;
        break;
    }

    //Register MIC data callback
    aud_intf_drv_setup.aud_intf_tx_mic_data = av_client_udp_voice_send_callback;

    ...
}
```

5.2.2 Fetch uplink MIC data

```
//Path      :  projects/media/av_server/main/src/av_server_devices.c
//Loaction  :  CPU0

//Register MIC data callback
    aud_intf_drv_setup.aud_intf_tx_mic_data = av_client_udp_voice_send_callback;
    ret = bk_aud_intf_drv_init(&aud_intf_drv_setup);

int av_server_udp_voice_send_callback(unsigned char *data, unsigned int len)
{
    ...

    //The commonly implemented callback typically involves transmitting in the WiFi␣
↪direction.
    return db_device_info->audio_transfer_cb->send(buffer, len, &retry_cnt);
}
```

5.2.3 Play downlink SPEAKER data

```
//Path      :  projects/media/av_server/main/src/av_server_devices.c
//Loaction  :  CPU0

void av_server_audio_data_callback(uint8_t *data, uint32_t length)
{
    ...

    //Send data to the SPEAKER.
    ret = bk_aud_intf_write_spk_data(data, length);

    ...
}
```

10.3 LVGL

A varity of lvgl demo projects are provided on BK7258 to demonstrate the functions of different typed and scenarios. Details are shown in the table below.

Project name	LCD resolution	Data format	Project description
86box	480*480	RGB565	86box demo
86box_smart_panel	480*480	RGB565	86box demo with speech recognition
benchmark	480*480	RGB565	LVGL performancescores demo
camera	480*854	RGB565	LVGL and camera image switching demo
keypad_encoder	800*480	RGB565	LVGL official demo
meter	400*400	RGB565	QSPI LCD animation demo
meter_rgb_16M	720*1280	RGB888	RGB888 LCD animation demo
music	720*1280	RGB565	LVGL official demo
stress	800*480	RGB565	LVGL official demo
widgets	1024*600	RGB565	LVGL official demo

The compilation command of the above demo project is `make bk7258 PROJECT=lvgl/xxx`. After the compilation and download is completed, it will be executed by default after power on.

Except for the LCD used in the camera project and meter project, which does not include touch function, the rest of the projects have turned on touch function. In addition, in addition to the normal LVGL UI display, the 86box project, 86box_smart_panel project and camera project also show another other different function. After powering on, the display need to be tested according to the following.

- 86box

  The 86box project shows how to turn off the LCD display and LVGL drawing. You can enter the command `86box` to test the shutdown.

- 86box_smart_panel

  The 86box_smart_panel project demonstrates the voice recognition function. The voice commands must be issued in Chinese. They are `a er mi nuo hui ke mo shi yong can mo shi li kai mo shi`. First, you need to issue the `a er mi nuo` wake-up word to jump to the demo page befor you can continue to issue other wake-up words for testing.

- camera

  The camera project shows the switching display of camera image and lvgl image. The specific test method is as follow.

- Install the IoT software on your mobile phone. After instation, click on the software to go to the main page.
- Use the IoT software to click on the upper right corner to add the `BK7258_DL_04` device and click on start adding. Then click `Select Wi-Fi` to connect to a network that can connect to the Internet and click next.
- Select your own Bluetooth device, which can be viewed in the power-on log or enter the `mac` command to view the MAC address of the device. Click and wait for the device's network connection progress to reach 100%.
- The phone automatically jump to the camera preview page. You can send the `lvcam_open` command to display the camera image on the LCD screen and send the `lvcam_close` command to display the LVGL image on the LCD. Continuously switch the display in sequence.

Note: The camera resolution used in the camera project is 864x480. In addition, this project can also be tested without using mobile Iot software. You can directly enter the `media uvc open 864X480` command to open the camera, and then enter the corresponding switching command.

10.4 thirdparty projects code

10.4.1 Agora-iot-sdk demo

1 Functional Overview

The sound network-based Agora-IoT-SDK connecting sound network cloud to realize the one-way diagram of equipment, mobile phone voice calls and equipment to mobile phones and rotate display

2 Code Path

demo path: `\projects\thirdparty\agora`

For the detailed description of the API interface of the Agora-iot-sdk library, please refer to the source file: `\components\bk_thirdparty\agora-iot-sdk\include\agora_rtc_api.h`

3 Cli command introduction

The commands supported by the demo are as follows:

Command	Description
agora_test {audio start|stop appid audio_type sampple_rate aec_en}	voice call
agora_test {video start|stop appid video_type video_ppi}	image transmission
agora_test {both start|stop appid audio_type sample_rate video_type video_ppi aec_en lcd_ppi lcd_driver}	voice call, image transmission and rotate display

The command parameters are described as follows:

appid	an appid for registration
audio_type	**mic and speaker type:** • 0: onboard • 1: UAC
sampple_rate	**sample rate:** • 8000 • 16000
aec_en	**AEC function control:** • 0: disable AEC • 1: enable AEC
video_type	**camera type:** • APP_CAMERA_DVP_JPEG • APP_CAMERA_DVP_YUV • APP_CAMERA_UVC_MJPEG
video_ppi	**resolution:** • PPI_320X240 • PPI_320X480 • PPI_480X320 • PPI_480X480 • PPI_480X800 • PPI_640X480 • PPI_800X480
lcd_driver	**lcd driver name:** • st7282 • gc9503v • h050iwv • hx8282 • md0430r • md0700r • nt35512 • nt35510 • st7796s • st7710s • st7701s • sn5st7701s • st7701s_ly
lcd_ppi	**resolution:** • PPI_320X240 • PPI_320X480 • PPI_480X320 • PPI_480X480 • PPI_480X800 • PPI_640X480 • PPI_800X480

Project Compilation instruction: `make bk7258 PROJECT = thirdparty/agora`

4. Demonstration introduction

The preparation of DEMO execution is as follows:

- 1.Prepare the Android machine that can be connected to the external network and install the APK `Open Live` to test the test
- 2.Go to Agora official website to register to apply for testing for testing `appid` , please refer to the information below for detailed operation
- 3.Prepare to access the 2.4GHz hotspot of the external network for Demo board connection and use

The steps of demo execution are as follows:

1. Mobile APK configuration
   - Open `Open Live` APK, enter the registration application `appid` , click `Enter` to enter
   - Enter Channel name `` hello_bk7258demo`` , click Join ROOM to enter, select `BroadCaster`

2.Demo board wifi connection

- DEMO board sending instructions `sta test xxxxxxx` connect 2.4GHz hot spots named TEST

3. Open and close voice calls
   - The demo board sends the command `agora_test audio start appid audio_type sample_rate aec_en` to open the voice call
   - The demo board sends the command `agora_test audio stop appid audio_type sample_rate aec_en` to close the voice call
4. Turn on and off video transmission
   - The demo board sends the command `agora_test video start appid video_type video_ppi` to open the image transmission
   - The demo board sends the command `agora_test video stop appid video_type video_ppi` to close the image transmission
5. Turn on and off voice call and image transmission
   - The demo board sends the command `agora_test both start appid audio_type sample_rate video_type video_ppi aec_en lcd_ppi lcd_driver` to open voice call, image transmission and rotate display
   - The demo board sends the command `agora_test both stop appid audio_type sample_rate video_type video_ppi aec_en lcd_ppi lcd_driver` to turn off voice calls, image transmission and rotate display

5. Audio configuration

The audio code in the demo is developed based on the `aud_intf` component. By configuring the input parameters of the `bk_aud_intf_voc_init` interface, the selection of mic and speaker types and the setting of the sampling rate can be realized.

The supported sample rates are as follows:

- 1.AUD_INTF_VOC_SAMP_RATE_8K: 8K sampling rate (recommended configuration)
- 2.AUD_INTF_VOC_SAMP_RATE_16K: 16K sampling rate (Currently, the SoundNet SDK does not support)

Note: 16K sampling rate will increase the load of network bandwidth, which will affect the frame rate of image transmission, so it is recommended to sample at 8K sampling rate

---

The supported mic and speaker types are as follows:

- 1.AUD_INTF_MIC_TYPE_BOARD: onboard mic (recommended configuration)
- 2.AUD_INTF_MIC_TYPE_UAC: uac type mic
- 3.AUD_INTF_SPK_TYPE_BOARD: onboard speaker (recommended configuration)
- 4.AUD_INTF_SPK_TYPE_UAC: uac type speaker

For detailed description of Aud_Intf API interface, please refer to the same page: `/api-reference/multi_media/bk_aud_intf.html`

6. Video configuration

The part of the image transmission code in the demo is developed based on the `media` component. By configuring the input parameters of the `media_app_camera_open` interface, the selection of the camera type and the setting of the resolution can be realized.

The supported types are as follows:

- 1.APP_CAMERA_DVP_JPEG: DVP camera in JPEG format (recommended configuration)
- 2.APP_CAMERA_DVP_YUV: DVP camera in YUV format (Currently, the SoundNet SDK does not support)
- 3.APP_CAMERA_UVC_MJPEG: UVC camera in MJPEG format

The supported resolutions are as follows:

- 1.PPI_320X240
- 2.PPI_320X480
- 3.PPI_480X320
- 4.PPI_480X480
- 5.PPI_480X800
- 6.PPI_640X480
- 7.PPI_800X480

Note:

- 1.The underlying hardware supports a variety of common resolutions, but it is not recommended to configure a higher sampling rate due to the limitations of the Acoustics SDK and network bandwidth

---

7. Reference materials

agora development guide

agora reference document: https://docs.agora.io/cn/Agora%20Platform/manage_projects?platform=Android

Agora appid register link:https://sso2.agora.io/cn/v5/login?_gl=1%2ardr355%2a_ga%2aMzkyNDM4ODYyLjE2NzM1MTM

Apk download link: http://dl.bekencorp.com/apk/shengwang/OpenLive_input_appid.apk

10.4.2 Wanson asr demo

1 Functional Overview

Based on the local speech recognition library provided by the third-party company Huazhen, the local offline voice wake-up word and command word recognition functions are realized.

2 Code Path

demo path: \components\audio_algorithm\wanson_asr

Wanson local speech recognition library (floating point library) path: \components\bk_thirdparty\asr\wanson

For detailed description of Wanson's local speech recognition API interface, please refer to the source file: \components\bk_thirdparty\asr\wanson\include\asr.h

DEMO Compilation instruction: make bk7258 PROJECT=thirdparty/wanson_asr

3. Demonstration introduction

After burning the firmware, the device will run real-time voice recognition function when powered on, and can be verified by saying wake-up words and command words to the mic.

The supported wake-up words and command words are as follows:

- Little Bee Steward recognizes successfully and prints the log xiao feng guan jia on the serial port
- armino recognizes successfully and prints the log a er mi nuo on the serial port
- Visitor mode recognize successful serial port print log hui ke mo shi
- dining mode recognizes successfully and prints the log yong can mo shi on the serial port
- Leave mode recognize successful serial port print log li kai mo shi
- home mode identification success serial port print log hui jia mo shi

4. wanson asr development guide

Note:

- 1. The wanson Speech Recognition Library requires the audio stream format to be: mono, 16K sampling rate, and 16bit bit width.
- 2. wanson Speech Recognition Library is based on floating-point arithmetic.
- 3. After modifying the wake word or command word, the libasr.a library needs to be replaced.

The process of developing real-time offline recognition based on the wanson speech recognition library is as follows:

- 1. Initialize speech recognition
- 2. Initialize audio sampling
- 3. Run speech recognition
- 4. Turn on audio sampling

Examples of interface calls are as follows:

```
/* init wanson asr lib */
Wanson_ASR_Init()
//reset wanson asr
Wanson_ASR_Reset();

/* init mic record */
aud_intf_drv_setup.aud_intf_tx_mic_data = aud_asr_handle;
//init audio component
bk_aud_intf_drv_init(&aud_intf_drv_setup);
aud_work_mode = AUD_INTF_WORK_MODE_GENERAL;
//set audio component work mode
bk_aud_intf_set_mode(aud_work_mode);
//init audio mic
aud_intf_mic_setup.samp_rate = AUD_ADC_SAMP_RATE_16K;
ret = bk_aud_intf_mic_init(&aud_intf_mic_setup);

/* start Speech Recognition */
//Continuously send the collected data to the algorithm for recognition
Wanson_ASR_Recog((short*)asr_buff, 480, &text, &score);

/* turn on audio sampling */
bk_aud_intf_mic_start();
```

5. Shanghai Huazhen Electronic Technology Co., Ltd.

Official website: http://www.wanson.cn/

Headquarters Address: Room 307-308, Huigaoguang Innovation Park, No. 789 Shenwang Road, Minhang District, Shanghai | Shenzhen Office Address: Room 2215-16, East Block, Building 1A, Huiyi City One Center, Xixiang, Baoan District, Shenzhen

Tel: 021-61557858

Mobile: 13524859176

13296017858

10.5 Wi-Fi projects code

10.5.1 BK-RLK Multi-media demo

1 Functional Overview

Based on BK-RLK wireless communication solution,the demo realized the function of transmitting one-way video, two-way audio and local LCD display from a single devices to multiple devices. Through this demo, user can see how to realize device discovery,connection,business data transmission and low power consumption management with the help of BK-RLK wireless communication solution.

2 Demo Introduction

BK-RLK multi-media demo is a multi-media communication system based on multiple devices supporting BK-RLK,which is divided into Client and Server.It is mainly demonstrates the use of the following functions of BK-RLK:

- Connection communication guide between Client and Server
- Power save guide between Client and Server
- Video and audio transmission guide between Client and Server

Connection communication guide between Client and Server: BK-RLK multi-media demo communication system is a process of automatic discovery of devices,connection pairing and video transmission.This process includes serveral state transitions as shown in the following table. During device initialization,the connection state will be configured to idle state by default. When the connection process starts,the connection state will be switched to probing state,and when the connection is completed,the connection state will be switched to connection state. Video and audio communication can only be carried out when the connection state is in connection state.

status	Description
idle	During device initialization,the default configuration connection state is idle state, which is used by both Client and Server.
probing	After the device initialization completed,it goes to probing state, which is used by both Client and Server.
wait probing end	After receiving the broadcast frame of Prob req, Server enters wait probing end state,which is only used by Server side.
connection	When the connection is completed,both the Client and Server enter the connection state.

Figure 1 describes the whole process of connection and communication between Client and Server in BK-RLK multi-media demo communication system. As shown in the Figure 1, both Client and Server set the connection state to idle state during initialization. Next, both Client and Server will configure BK-RLK (please refer to the Wi-Fi BK-RLK User Guide for the specific configuration of BK-RLK). After the configuration of BK-RLK is completed, Client and Server connection state will be configured as probing state. After the connection state of Client is changed to probing state, BK-RLK data send interface will be called to continuously send Prop req broadcast frames until BK-RLK receive interface received the Prob rsp unicast frame replied by Server. Client adds the information of the peer device to the BK-RLK peer link list through the received Prob rsp frame. Client changed connection state to the connection state. Then, Client starts video and audio configuration, turns on the camera, turns on the local LCD, turns on the audio, and starts calling the BK-RLK send interface to start business data communication with the Server. After the connection state of the Server is changed to probing state, it will be in standby state until the Prob req broadcast frame sent by the Client is received through the BK-RLK receive interface. Server will switched state to the wait probing end state and send the Prob rsp unicast frame through the BK-RLK send interface. And then,the connection state switch to the connection state after receiving the ACK. Subsequently, it starts to configure video and audio and carries out business data communication with the Client.

Note:When both the Client and Server are in the connection state,they will automatically connect each other when one of them is disconnected.

Power save guide between Client and Server: In BK-RLK multi-media demo communication system, the Client can enter the power save state. The Client maintain work state for a period of time and then enter the power save state periodically. The information involved is shown in the following table:

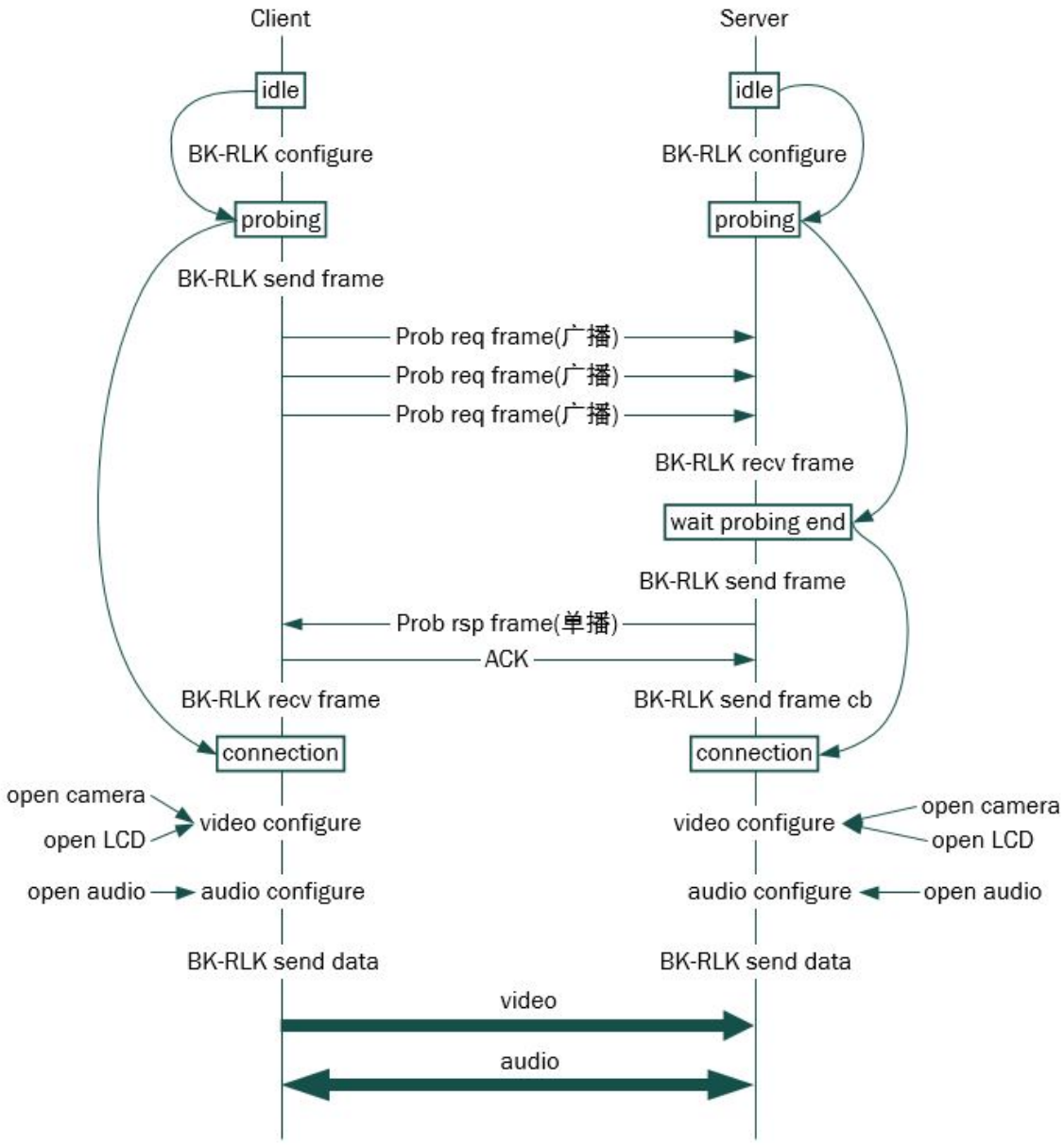


Fig. 14: Figure 1. communication flow chart of connection between Client and Server

role	information	instruction	comment
Client	RLK_RTC_LOWPOWER_PS_INTERVAL	power save period, default 2s	This value determines the sleep time of the device.The larger the value is set, the lower power consumption will be. At the same time,the wake up delay may become larger when peer device wake up Client device.
	RLK_RTC_LOWPOWER_KEEPALIVE_TIME_INTERVAL	work period,default 2 ms	After the device goes into sleep, it needs to periodically monitor whether there is any business. This value determins the monitor time of the device. The smaller the value is set,the smaller the work time and power consumption will be, and it may alse affect the wake-up delay.
	rlkc ps	sscom cmd. Input by Client, Client devices enter sleep mode.	The API function can be called to realize this function in the un-cmd situation.
Server	rlks wakeup_peer	sscom cmd. Input by Server,wake up Client device and keep work mode.	The API function can be called to realize this function in the un-cmd situation.

BK-RLK power save usage can be divided into Client and Server: Client input rlkc ps cmd,whether the cmd takes effect,there are the following forms:

- Video and audio in progress will stop immediately;
- Using the current detection tool to see that the peak current drops immediately;

Note:After input rlkc ps cmd,the client device will work with the configured sleep/work cycle. At this time ,all business is stopped. The API function rlk_client_ps_cmd_handler() can be called to realize this function in the un-cmd situation.

Server input rlks wakeup_peer cmd. Server will print WAKEUP PEER SUCCESS information when successfully wake up client.

Notes:After input rlks wakeup_peer cmd, Client device will still work in the configured sleep/work cycle untile the successful print information is received. Business has not been resumed for the Client device. After received the successful print information, Client current returns to the peak value and work again. There is a delay between the input of the cmd and the successful wake up the peer. It depends on the current air environment and the configured sleep cycle and work cycle length. The API function rlk_server_wakeup_peer() can be called to realize this function in the un-cmd situation.

The specific wake up process is shown in Figure 2 below:

In an open environment, comparison of power consumption between Standerd Wi-Fi and BK-RLK using the same set of hardware devices are shown in the following table:

devices	condition	average power consumption
Wi-Fi	Dtim20	557uA
BK-RLK	RLK_RTC_LOWPOWER_PS_INTERVAL = 2s RLK_RTC_LOWPOWER_KEEPALIVE_TIME_INTERVAL = 2ms	320uA

Video and audio transmission guide between Client and Server: After the Client and Server are successfully paired,they automatically enter video and audio transmission mode.

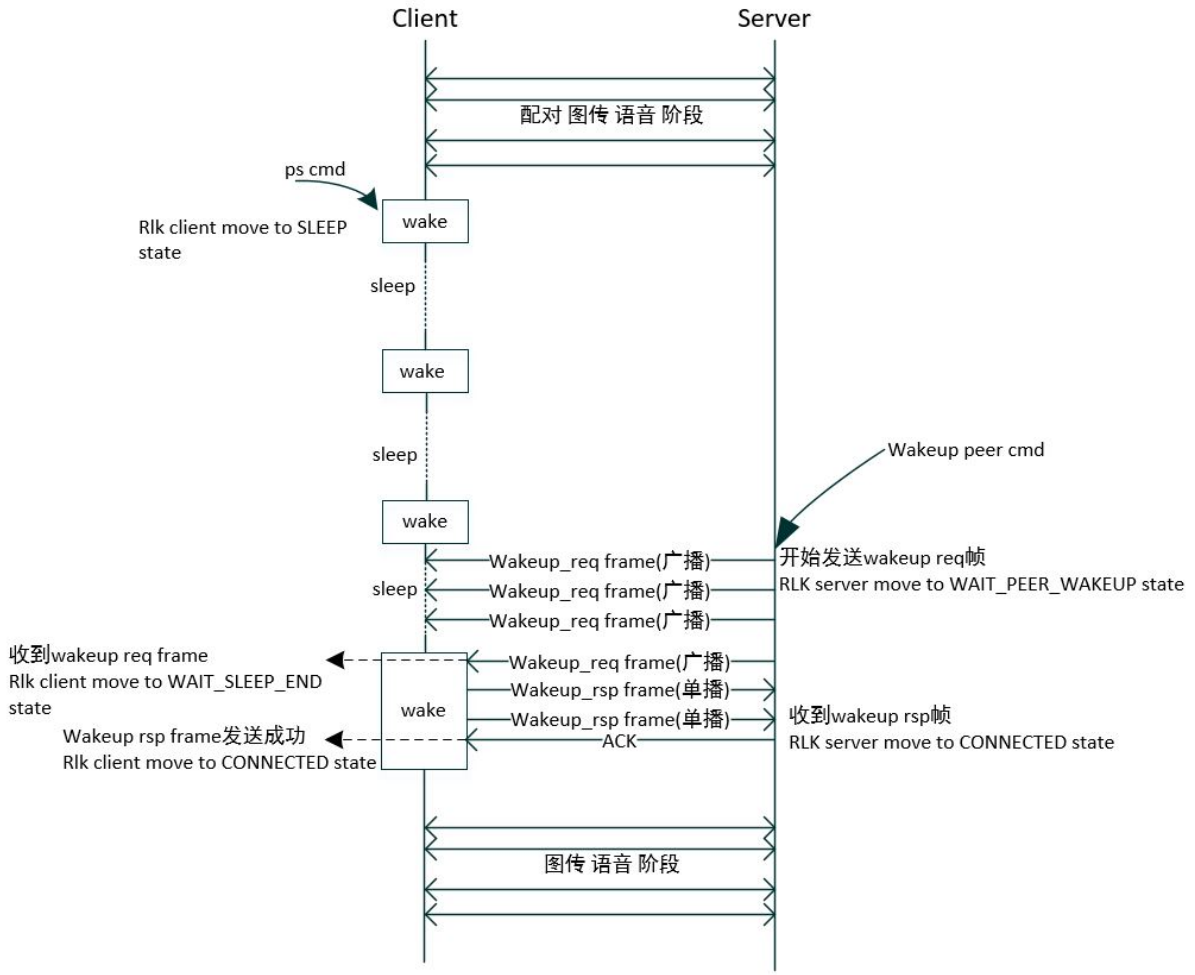


Fig. 15: Figure 2. Power save flow chart of Client and Server

3 Environment Construction

BK-RLK multi-media communication system is divided into two parts: Client and Server. And its environment construction is described in Figure 3 and Figure 4. Figure 3 is a BK7256 sample device as the Client, which adds a usb camera,an LCD display and an audio broadcaster.

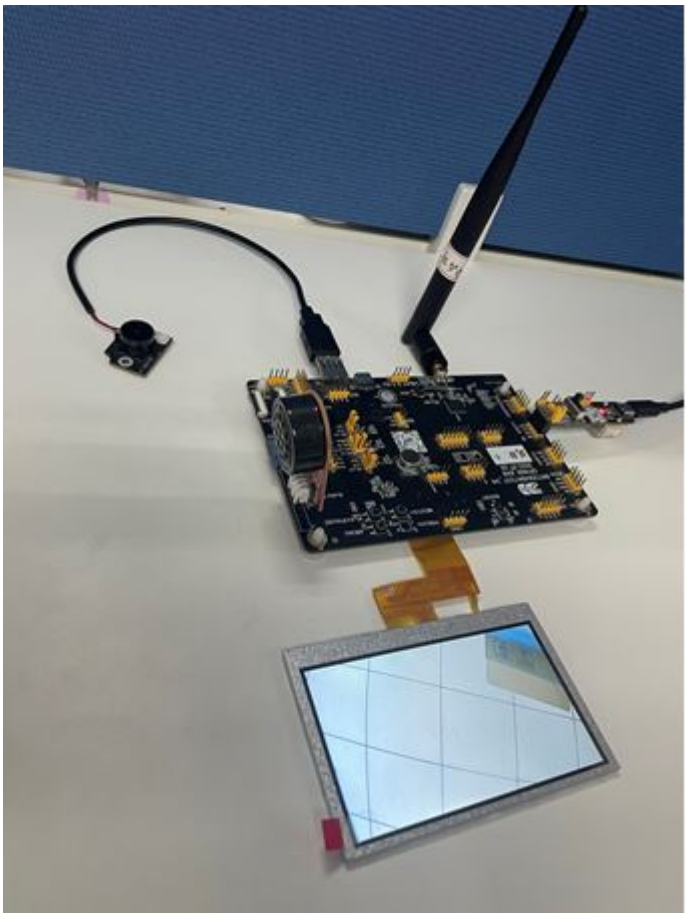


Fig. 16: Figure 3. Client environment construction

Figure 4 is a BK7256 sample device as the Server, which adds an LCD display and an audio broadcaster.

4 Code Path

BK-RLK Multi-media system demo path: `./projects/wifi`

Client demo path:`./projects/wifi/rlk_av_client`

Server demo path:`./projects/wifi/rlk_av_server`

Client build cmd:`make bk7258 PROJECT=wifi/rlk_av_client`

Server build cmd:`make bk7258 PROJECT=wifi/rlk_av_server`

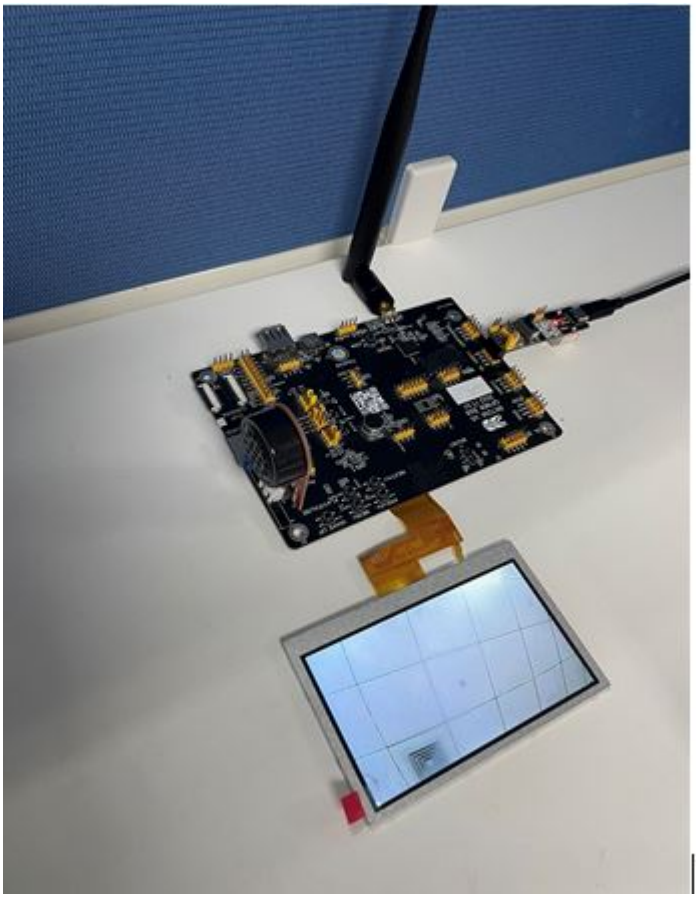


Fig. 17: Figure 4. Server environment construction

5 Reference Materials

API reference: introduce BK-RLK API interface

BK-RLK development guide: introduce BK-RLK user guide

BK-RLK project: introduce BK-RLK projects

10.6 Peripheral

10.6.1 DVP

1. Introduction

This project is the default debugging project for DVP, which can be used to test whether the inserted DVP works properly. The default is 16M PSRAM.

1.1 Specifications

Please refer to Specifications

1.2 Path

<bk_avdk source code path>/projects/peripheral/dvp

2. Framework diagram

Please refer to Framework diagram

3. Configuration

3.1 Bluetooth and Multimedia Memory Reuse

Using this project is only for verifying DVP, so the memory does not need to be shared with Bluetooth.

marco	value	implication
CONFIG_BT_REUSE_MEDIA_MEMORY	N	Multimedia and Bluetooth share one SRAM (time-division multiplexing)

3.1.1 Uninstalling Bluetooth

```
#ifdef CONFIG_BT_REUSE_MEDIA_MEMORY
#if CONFIG_BLUETOOTH
    bk_bluetooth_deinit();
#endif
#endif
```

3.1.2 Initialize Bluetooth

```
bk_bluetooth_init();
```

3.2 Differences

The difference between DVP and Doorviewer is that:

- DVP starts by default when powered on, and analyzes the output JPEG image raw data format YUV422/YUV420 and output resolution through a software decoder. Other peripherals are not involved and do not support image transfer, LCD display, voice transmission, etc.
- Doorviewer is a complete door lock solution.

4. Demonstration explanation

Default power on self start, no additional operation required, Print the following log.

```
(7598):##MJPEG:camera_type:1(1:dvp 2:uvc) frame_id:178, length:26163, frame_
↪addr:0x6011f7a0
video_co:I(7598):4:2:2, YUV:
(7598):##DECODE:pixel_x:640, pixel_y:480
(7600):rotate_angle:0(0:0 1:90 2:180 3:270)
(7600):byte_order:0(0:little endian 1:big endian)
(7600):out_fmt:YUYV
```

5. Code explanation

Please refer to Code explanation

6. Add adaptation

When adapting a new camera, the debugging steps are as follows:

Step 1: Refer to the configuration methods of other peripherals: `./components/bk_peripheral/src/dvp/dvp_gc2145.c`, Add a new dvp_xx.c file to the corresponding folder. And add the newly added peripherals to `dvp_sensor_devices.c`, modify the directory's: "./components/bk_peripheral" CMakeList.txt file, add the new file dvp_xx.c to compiler.

Step 2: Clearly specify the I2C read/write address and the hardware I2C ID used in dvp_xx.c. BK7258 defaults to I2C1, corresponding to GPIO0 and GPIO1. The I2C ID used by DVP is configured through macros.

marco	value	implication
CONFIG_DVP_CAMERA_I2C_ID	1	The index of using I2C

Step 3: Clarify the CHIP_ID of the camera in dvp_xx.c.

Step 4: Clarify the members of the structure "dvp_sensor_config. t" in dvp_xx.c. Note that the configured parameters need to be consistent with the camera configuration. For example, CLK=MCLK_24 indicates that the input clock of the sensor is 24MHz,

Instead of other values, otherwise it will be inconsistent with the expected frame rate; For example, fmt=PIXEL_FMT_YUYV indicates that the format of the sensor's output image is YUYV, not other types. Otherwise, the main control cannot recognize it, resulting in the inability to output the image or abnormal output delay; For example, vsync=SYNC=HIGHVNet, which means that the effective signal of the sensor is effective when vsync is high, not low, otherwise it will cause abnormal sampling of the main control and result in the inability to generate a graph. Other member variables are similar and require strict correspondence.

Step 5: Configure the corresponding initialization table, resolution sub table, frame rate sub table, etc. in dvp_xx.c.

Step 6: Clarify the control GPIO of the DVP hardware power supply. BK7258 also controls the LDO of the DVP through GPIO28. Pulling it high will enable the IOVDD of the DVP, and pulling it low will power down the DVP. This can be configured through macros:

marco	value	implication
CONFIG_CAMERA_CTRL_POWER_GPIO_ID	0x1C	The GPIO ID control dvp power

DVP also needs to supply power to the DVDD, which varies depending on the camera.

Step 7: Currently, there are few supported feature configurations, such as frame rate configuration. If additional support is needed for exposure adjustment, image flipping, image night mode, etc. It is necessary to add members to the above structure and assign corresponding sensor configuration tables to the corresponding members.

Step 8: Use this project to test the newly adapted camera.

```
//dvp camera struct
typedef struct
{
    char *name;  /**< sensor name */
    media_ppi_t def_ppi;  /**< sensor default resolution */
    frame_fps_t def_fps;  /**< sensor default fps */
    mclk_freq_t  clk;  /**< sensor work clk in config fps and ppi */
    pixel_format_t fmt; /**< sensor output data format */
    sync_level_t vsync; /**< sensor vsync active level  */
    sync_level_t hsync; /**< sensor hsync active level  */
    uint16_t id;  /**< sensor type, sensor_id_t */
    uint16_t address;  /**< sensor write register address by i2c */
    uint16_t fps_cap;  /**< sensor support fps */
    uint16_t ppi_cap;  /**< sensor support resoultions */
    bool (*detect)(void);  /**< auto detect used dvp sensor */
    int (*init)(void);  /**< init dvp sensor */
    int (*set_ppi)(media_ppi_t ppi);  /**< set resolution of sensor */
    int (*set_fps)(frame_fps_t fps);  /**< set fps of sensor */
    int (*power_down)(void);  /**< power down or reset of sensor */
    int (*dump_register)(media_ppi_t ppi);  /**< dump sensor register */
    void (*read_register)(bool enable);  /**< read sensor register when write*/
} dvp_sensor_config_t;

Description of some parameters:
    * clk: clock input for the camera. The default clock is 24MHz, which needs to be␣
↪configured according to the camera specification.
    * fmt: The format of camera output data to the chip. Currently, only YUV420 is␣
↪supported, and the sequence needs to be synchronized according to the camera output␣
↪sequence. The default is YUYV.
```

(continues on next page)

(continued from previous page)

```
    * vsync: camera output vsync effective level. When some cameras vsync is low, the␣
↪output valid data needs to be synchronized with the camera vsync output level. The␣
↪default high level is valid.
    * hsync: effective level of camera output vsync. When vsync of some cameras is␣
↪low, the output of valid data needs to be synchronized with the hsync output level␣
↪of the camera. The default high level is valid.
    * address: Configure the I2C slave address of the camera register. The parameter␣
↪needs to be configured according to the camera specification.
    * fps_cap: The frame rate table supported by the camera, which needs to be␣
↪configured with the corresponding register.
    * ppi_cap: resolution table supported by camera, need to configure the␣
↪corresponding register to achieve.
```

10.6.2 UVC

1. Introduction

This project is the default debugging project for UVC, which can be used to test whether the inserted UVC works properly. The default is 16M PSRAM.

1.1 Specifications

Please refer to Specifications

1.2 Path

<bk_avdk source code path>/projects/peripheral/uvc

2. Framework diagram

Please refer to Framework diagram

3. Configuration

3.1 Bluetooth and Multimedia Memory Reuse

Using this project is only for verifying UVC, so the memory does not need to be shared with Bluetooth.

marco	value	implication
CONFIG_BT_REUSE_MEDIA_MEMORY	N	Multimedia and Bluetooth share one SRAM (time-division multiplexing)

3.1.1 Uninstalling Bluetooth

```
#ifdef CONFIG_BT_REUSE_MEDIA_MEMORY
#if CONFIG_BLUETOOTH
    bk_bluetooth_deinit();
#endif
#endif
```

3.1.2 Initialize Bluetooth

```
bk_bluetooth_init();
```

3.2 Differences

The difference between UVC and Doorbell is that:

- UVC starts by default when powered on, and analyzes the output JPEG image raw data format YUV422/YUV420 and output resolution through a software decoder. Other peripherals are not involved and do not support image transfer, LCD display, voice transmission, etc.
- Doorbell is a complete door lock solution.

4. Demonstration explanation

Default power on self start, no additional operation required, Print the following log.

```
(4736):##MJPEG: frame_id:113, length:56922, frame_addr:0x6020fba0
video_co:I(4736):4:2:2, YUV:
(4736):##DECODE:pixel_x:864, pixel_y:480
(4736):rotate_angle:0(0:0 1:90 2:180 3:270)
(4736):byte_order:0(0:little endian 1:big endian)
(4736):out_fmt:YUYV
```

5. Code explanation

Please refer to Code explanation

10.6.3 QSPI_LCD

10.6.4 SPI_LCD

10.6.5 RGB565 LCD

1 Introduction

RGB565 project is verified RGB565 interface display function,randoms color are displayed every second after power on

the main differences of RGB888/RGB565/RGB666 project is whether the screen supports RGB565 or rgb666 or rgb888, the data source is not mandatory ,it can be RGB565, RGB888 or YUYV.

2 Code Path

demo path:`./projects/peripheral/lcd_rgb565`

3.Code explanation

compile commands:`make bk7258 PROJECT=peripheral/lcd_rgb565`

Attention:

LCD RGB565 or RGB666 or RGB888 is defined by lcd_device_t member "out_fmt", alse can config by API:

LCD conifig by default:

```
const lcd_device_t lcd_device_st7701sn =
{
        .id = LCD_DEVICE_ST7701SN,
        .name = "st7701sn",
        .type = LCD_TYPE_RGB565,    /**< lcd device hw interface *LCD_TYPE_RGB=LCD_
↪TYPE_RGB565 */
        .ppi = PPI_480X854,
        .rgb = &lcd_rgb,
        .src_fmt = PIXEL_FMT_RGB565,   /**< source data format: input to display␣
↪module data format(rgb565/rgb888/yuv)*/
        .out_fmt = PIXEL_FMT_RGB565, /**< display module output data format(rgb565/
↪rgb666/rgb888), input to lcd device,*/
        .init = lcd_st7701sn_init,
        .lcd_off = NULL,
};
```

user can also use API to config:

```
lcd_hal_rgb_set_in_out_format(src_fmt, out_fmt);
```

RGB565 LCD config step

- step 1: open lcd

```
media_app_lcd_pipeline_disp_open
media_app_lcd_example_display
```

- step 2:malloc psram

```
frame_buffer_display_malloc
```

after psram malloc, should config frame:

```
uint32_t size = ppi_to_pixel_x(lcd_open->device_ppi) * ppi_to_pixel_y(lcd_open->
↪device_ppi) * 2;

frame = frame_buffer_display_malloc(size);

frame->width = ppi_to_pixel_x(lcd_open->device_ppi);
frame->height = ppi_to_pixel_y(lcd_open->device_ppi);
frame->fmt = PIXEL_FMT_RGB565;
```

- step 3:fill color

```
lcd_fill_rand_color
```

- step 4:display request

```
lcd_display_frame_request
```

- step 5:cycle step2-4

10.6.6 RGB888 LCD

1 Introduction

The project is verified RGB888 interface display function,randoms color are displayed every second after power on

the main differences of RGB888/RGB565/RGB666 project is whether the screen supports RGB565 or rgb666 or rgb888, the data source is not mandatory ,it can be RGB565, RGB888 or YUYV.

2 Code Path

demo path:`./projects/peripheral/lcd_rgb888`

3.Code explanation

compile commands:`make bk7258 PROJECT=peripheral/lcd_rgb888`

Attention: LCD RGB565 or RGB666 or RGB888 is defined by lcd_device_t member "out_fmt", alse can config by API: src_fmt is the source data send to display, such as following is RGB888 send to LCD directly.

LCD conifig by default:

```
const lcd_device_t lcd_device_st7701sn =
{
        .id = LCD_DEVICE_ST7701SN,
        .name = "st7701sn",
        .type = LCD_TYPE_RGB565,    /**< lcd device hw interface *LCD_TYPE_RGB=LCD_
↪TYPE_RGB565 */
        .ppi = PPI_480X854,
        .rgb = &lcd_rgb,
        .src_fmt = PIXEL_FMT_RGB888,   /**< source data format: input to display␣
↪module data format(rgb565/rgb888/yuv)*/
        .out_fmt = PIXEL_FMT_RGB888, /**< display module output data format(rgb565/
↪rgb666/rgb888), input to lcd device,*/
        .init = lcd_st7701sn_init,
        .lcd_off = NULL,
};
```

user can also use API to config:

```
lcd_hal_rgb_set_in_out_format(src_fmt, out_fmt);
```

RGB888 LCD config step

- step 1:open lcd

```
media_app_lcd_pipeline_disp_open
media_ipc_send
```

- step 2:malloc psram

```
frame_buffer_display_malloc
```

after psram malloc, should config frame:

```
uint32_t size = ppi_to_pixel_x(lcd_open->device_ppi) * ppi_to_pixel_y(lcd_open->
↪device_ppi) * 3;
frame = frame_buffer_display_malloc(size);

frame->width = ppi_to_pixel_x(lcd_open->device_ppi);
frame->height = ppi_to_pixel_y(lcd_open->device_ppi);
frame->fmt = PIXEL_FMT_RGB888;
```

like: frame->fmt = PIXEL_FMT_RGB888; is config source data RGB888 to display module;

- step 3:fill color

```
lcd_fill_rand_color
```

- step 4:display request

```
lcd_display_frame_request
```

- step 5:cycle step2-4

10.6.7 RGB666 LCD

1 Introduction

The project is verified RGB666 interface display function,randoms color are displayed every second after power on

the main differences of RGB888/RGB565/RGB666 project is whether the screen supports RGB565 or rgb666 or rgb888, the data source is not mandatory ,it can be RGB565, RGB888 or YUYV.

2 Code Path

demo path:`./projects/peripheral/lcd_rgb666`

3.Code explanation

compile commands:`make bk7258 PROJECT=peripheral/lcd_rgb666`

Attention: LCD RGB565 or RGB666 or RGB888 is defined by lcd_device_t member "out_fmt", alse can config by API:

src_fmt is the source data send to display, such as following is YUYV send to display, display convert to RGB666 to LCD display.

LCD conifig by default:

```
const lcd_device_t lcd_device_st7701sn =
{
        .id = LCD_DEVICE_ST7701SN,
        .name = "st7701sn",
        .type = LCD_TYPE_RGB,    /**< lcd device hw interface *LCD_TYPE_RGB=LCD_
↪TYPE_RGB565 */
        .ppi = PPI_480X854,
        .rgb = &lcd_rgb,
        .src_fmt = PIXEL_FMT_YUYV,   /**< source data format: input to display␣
↪module data format(rgb565/rgb888/yuv)*/
        .out_fmt = PIXEL_FMT_RGB666, /**< display module output data format(rgb565/
↪rgb666/rgb888), input to lcd device,*/
        .init = lcd_st7701sn_init,
        .lcd_off = NULL,
};
```

user can also use API to config:

```
lcd_hal_rgb_set_in_out_format(src_fmt, out_fmt);
```

RGB666 LCD config step

- step 1:open lcd

```
media_app_lcd_pipeline_disp_open
media_ipc_send
```

- step 2:malloc psram

```
frame_buffer_display_malloc
```

after psram malloc, should config frame:

```
uint32_t size = ppi_to_pixel_x(lcd_open->device_ppi) * ppi_to_pixel_y(lcd_open->
↪device_ppi) * 2;
frame = frame_buffer_display_malloc(size);

frame->width = ppi_to_pixel_x(lcd_open->device_ppi);
frame->height = ppi_to_pixel_y(lcd_open->device_ppi);
frame->fmt = PIXEL_FMT_YUYV;
```

frame->fmt = PIXEL_FMT_YUYV; configures the data source for the display as YUV.

- step 3:fill color

```
lcd_fill_rand_color
```

- step 4:display request

```
lcd_display_frame_request
```

- step 5:cycle step2-4

10.6.8 8080 LCD

1 Introduction

The project is verified mcu interface display function,randoms color are displayed every second after power on

2 Code Path

demo path:`./projects/peripheral/lcd_8080`

3.Code explanation

compile commands:`make bk7258 PROJECT=peripheral/lcd_8080`

Attention: how to config LCD RGB interface or MCU 8080 is configed by cd_device_t member "type"

LCD conifig by default:

```
const lcd_device_t lcd_device_st7796s =
{
        .id = LCD_DEVICE_ST7796S,
        .name = "st7796s",
        .type = LCD_TYPE_MCU8080,
        .ppi = PPI_320X480,
        .mcu = &lcd_mcu,
        .init = lcd_st7796s_init,
        .lcd_off = st7796s_lcd_off,
};
```

the another params:

```
pixel_format_t src_fmt;  /**< source data format: input to display module data␣
↪format(rgb565/rgb888/yuv)*/
pixel_format_t out_fmt;   /**< display module output data format(rgb565/rgb666/
↪rgb888), input to lcd device,*/
```

is the same function as to the following API:

```
lcd_hal_rgb_set_in_out_format(src_fmt, out_fmt);
```

LCD MCU config step

- step 1: open lcd

```
media_app_lcd_pipeline_disp_open
media_app_lcd_example_display
```

- step 2:malloc psram

```
frame_buffer_display_malloc
```

after psram malloc, should config frame:

```
uint32_t size = ppi_to_pixel_x(lcd_open->device_ppi) * ppi_to_pixel_y(lcd_open->
↪device_ppi) * 2;

frame = frame_buffer_display_malloc(size);

frame->width = ppi_to_pixel_x(lcd_open->device_ppi);
frame->height = ppi_to_pixel_y(lcd_open->device_ppi);
frame->fmt = PIXEL_FMT_RGB565;
```

- step 3:fill color

```
lcd_fill_rand_color
```

- step 4:display request

```
lcd_display_frame_request
```

- step 5:cycle step2-4

CHAPTER

ELEVEN

MULTIMEDIA POWER CONSUMPTION

11.1 1 Overview of multimedia power consumption

Based on BK7258 projects, multimedia functions are implemented on CPU1, and each hardware module, such as PSRAM, LCD, H264, USB, etc., has the corresponding clock configuration, power management, etc. In order to achieve lower power consumption, the switch needs to be used symmetrically.

11.2 2 Multimedia applications

Multimedia SDK provides a feature-rich and complete interface, for the use of the user, and user does not need to consider the vote of the application of the interface, is the need to make sure that is symmetric call, interface reference address: `./components/multimedia/include/media_app.h`, The description is as follows.

Enable/disable the camera, need to pay attention to the content of the parameters:

```
bk_err_t media_app_camera_open(media_camera_device_t *device);
bk_err_t media_app_camera_close(camera_type_t type);
```

Enable/disable the interface to get images. Different types of images need to be passed different types of parameters, allowing the acquired image data to be processed directly in the callback function (parameter cb), this interface can only be effective after the camera is turned on:

```
bk_err_t media_app_register_read_frame_callback(pixel_format_t fmt, frame_
↪cb_t cb);
bk_err_t media_app_unregister_read_frame_callback(void);
```

Enable/disable h264 encoding function, this interface supports JPEG image decoding, image conversion to YUV format, and then H264 encoding output:

```
bk_err_t media_app_h264_pipeline_open(void);
bk_err_t media_app_h264_pipeline_close(void);
```

Enable/disable image rotation function, this interface supports JPEG decoded images, rotation at any Angle, and then display on the corresponding LCD screen:

```
bk_err_t media_app_lcd_pipline_rotate_open(media_rotate_t rotate);
bk_err_t media_app_lcd_pipline_rotate_close(void);
```

In addition to the above basic functions, it also supports some additional functions, such as when the UVC camera is opened, support registration callback function to obtain UVC enumeration information

and connection status, after obtaining the need to call the parameter configuration interface, so that UVC really start. Support to set parameters directly in the registered callback function to start UVC:

```
bk_err_t media_app_uvc_register_info_notify_cb(uvc_device_info_t cb);
bk_err_t media_app_set_uvc_device_param(bk_uvc_config_t *config);
```

Support in H264 encoding, adjust the compression rate to achieve a better image effect, specific debugging instructions please refer to the H264 encoding document, this interface only after the H264 encoding function is opened:

```
bk_err_t media_app_set_compression_ratio(compress_ratio_t * ratio);
```

The interface has a corresponding reference case, please refer: `./components/multimedia/cli/media_cli.c`.

11.3 3 Multimedia voting

All multimedia functions are implemented on CPU1. If the SDK interface is invoked, CPU1 will vote on by default and then process the corresponding process. The voting interface is as follows:

```
bk_err_t bk_pm_module_vote_boot_cp1_ctrl(pm_boot_cp1_module_name_e module,
↪pm_power_module_state_e power_state);
```

The interface supports different modules to vote and power on or off. For details, reference:

```
typedef enum
{
    PM_BOOT_CP1_MODULE_NAME_FFT          = 0,
    PM_BOOT_CP1_MODULE_NAME_AUDP_SBC        ,// 1
    PM_BOOT_CP1_MODULE_NAME_AUDP_AUDIO      ,// 2
    PM_BOOT_CP1_MODULE_NAME_AUDP_I2S        ,// 3
    PM_BOOT_CP1_MODULE_NAME_VIDP_JPEG_EN    ,// 4
    PM_BOOT_CP1_MODULE_NAME_VIDP_JPEG_DE    ,// 5
    PM_BOOT_CP1_MODULE_NAME_VIDP_DMA2D      ,// 6
    PM_BOOT_CP1_MODULE_NAME_VIDP_LCD        ,// 7
    PM_BOOT_CP1_MODULE_NAME_MULTIMEDIA      ,// 8
    PM_BOOT_CP1_MODULE_NAME_APP             ,// 9
    PM_BOOT_CP1_MODULE_NAME_VIDP_ROTATE     ,// 10
    PM_BOOT_CP1_MODULE_NAME_VIDP_SCALE      ,// 11
    PM_BOOT_CP1_MODULE_NAME_GET_MEDIA_MSG   ,// 12
    PM_BOOT_CP1_MODULE_NAME_MAX             ,// attention: MAX value can␣
↪not exceed 31.
}pm_boot_cp1_module_name_e;
```

The power supply of different modules also needs to be controlled by voting to ensure that the modules share the power domain. When a module needs to be used, the power supply needs to be turned on. When all modules do not need to work, the power failure process is carried out. Customers do not need to pay attention to it when using it, but they can pay attention to whether the corresponding module is really powered off when entering the low power mode. The voting interface is as follows:

```
bk_err_t bk_pm_module_vote_power_ctrl(pm_power_module_name_e module,pm_
↪power_module_state_e power_state);
```

The above interface can achieve the switch function of different modules by configuring different parameters. The type reference file of the first parameter is: `./bk_idk/include/modules/pm.h`. The following are the tickets defined by the multimedia module:

```
/*----SUB POWER DOMAIN VIDP--------*/
#define PM_POWER_SUB_MODULE_NAME_VIDP_DMA2D            (POWER_SUB_MODULE_
↪NAME_VIDP_DMA2D)
#define PM_POWER_SUB_MODULE_NAME_VIDP_YUVBUF           (POWER_SUB_MODULE_
↪NAME_VIDP_YUVBUF)
#define PM_POWER_SUB_MODULE_NAME_VIDP_JPEG_EN          (POWER_SUB_MODULE_
↪NAME_VIDP_JPEG_EN)
#define PM_POWER_SUB_MODULE_NAME_VIDP_JPEG_DE          (POWER_SUB_MODULE_
↪NAME_VIDP_JPEG_DE)
#define PM_POWER_SUB_MODULE_NAME_VIDP_LCD              (POWER_SUB_MODULE_
↪NAME_VIDP_LCD)
#define PM_POWER_SUB_MODULE_NAME_VIDP_ROTT             (POWER_SUB_MODULE_
↪NAME_VIDP_ROTT)
#define PM_POWER_SUB_MODULE_NAME_VIDP_SCAL0            (POWER_SUB_MODULE_
↪NAME_VIDP_SCAL0)
#define PM_POWER_SUB_MODULE_NAME_VIDP_SCAL1            (POWER_SUB_MODULE_
↪NAME_VIDP_SCAL1)
#define PM_POWER_SUB_MODULE_NAME_VIDP_H264             (POWER_SUB_MODULE_
↪NAME_VIDP_H264)
```

When different PASRAM modules are used, they will also vote to ensure normal function and low power consumption. The voting interface is:

```
bk_err_t bk_pm_module_vote_psram_ctrl(pm_power_psram_module_name_e module,
↪pm_power_module_state_e power_state);
```

The above interface can achieve the control of PASRAM by different modules by configuring different parameters. The enumerated values of different modules are as follows:

```
typedef enum
{
    PM_POWER_PSRAM_MODULE_NAME_FFT       = 0,
    PM_POWER_PSRAM_MODULE_NAME_AUDP_SBC     ,// 1
    PM_POWER_PSRAM_MODULE_NAME_AUDP_AUDIO   ,// 2
    PM_POWER_PSRAM_MODULE_NAME_AUDP_I2S     ,// 3
    PM_POWER_PSRAM_MODULE_NAME_VIDP_JPEG_EN ,// 4
    PM_POWER_PSRAM_MODULE_NAME_VIDP_H264_EN ,// 5
    PM_POWER_PSRAM_MODULE_NAME_VIDP_JPEG_DE ,// 6
    PM_POWER_PSRAM_MODULE_NAME_VIDP_DMA2D   ,// 7
    PM_POWER_PSRAM_MODULE_NAME_VIDP_LCD     ,// 8
    PM_POWER_PSRAM_MODULE_NAME_APP          ,// 9
    PM_POWER_PSRAM_MODULE_NAME_AS_MEM       ,// 10
    PM_POWER_PSRAM_MODULE_NAME_CPU1         ,// 11
    PM_POWER_PSRAM_MODULE_NAME_MEDIA        ,// 12
    PM_POWER_PSRAM_MODULE_NAME_LVGL_CODE_RUN,// 13
    PM_POWER_PSRAM_MODULE_NAME_MAX          ,// attention: MAX value can␣
↪not exceed 31.
}pm_power_psram_module_name_e;
```

11.4 4 multimedia low power inspection

When it is necessary to enter the low power consumption, all peripherals need to be turned off, including PSRAM, to ensure that all modules are powered off, you can send the following command to check:

```
pm_debug 8
```

By default, for example, when the command is powered on, the output is as follows:

```
1. pm video and audio state:0x0 0x0

2. pm ahpb and bakp state:0x0 0x10001

3. pm low vol[module:0xffbfefff] [need module:0xffffffff]

4. pm deepsleep[module:0xc0][need module:0x3c0]

5. pm power and pmu state[0x200000c8][0x2f1]

6. Attention the bakp not power down[modulue:0x10001]

7. pm_psram_ctrl_state:0x0 0x0

8. pm_cp1_ctrl_state:0x0

9. pm_cp1_boot_ready:0x0 0xffffffff

10. pm_module_lv_sleep_state:0x0
```

For the multimedia module, pay attention to the first line (indicating the internal power control of each multimedia module, and the corresponding ticket is pm_power_module_name_e, which should be 0 when not used) and the seventh line (indicating the power control of PSRAM, because several modules use PARAM. Line 8 (indicates that different modules voted for CPU1 to start, and the corresponding vote is pm_boot_cp1_module_name_e, which should be 0 when not applicable).

Note: The above interfaces do not require the user to encapsulate the call, when using multimedia functions, these functions have been encapsulated into the internal logic of the switch, only need to ensure that the switch is symmetric when used. In addition, the above command is a check method that depends on the CLI command function. CLI functions may not be enabled.

CHAPTER

TWELVE

AUDIO ALGORITHMS

12.1 AEC Debug

12.1.1 1.Overview

The AEC echo cancellation algorithm is mainly used in bidirectional voice call scenarios, which eliminates the audio played by the speaker collected by the device end mic through the algorithm to obtain clean human voices.

12.1.2 2.Function Overview Engineering Code Path

The reference demo path for using the AEC algorithm is as follows: `./components/audio_algorithm/aec`

The demo provides methods for testing and verifying the AEC algorithm's PCM data and interface usage.

12.1.3 3.Debugging of AEC echo cancellation effect

AEC debugging precautions:

Important: The echo situation is closely related to the device's mic sensitivity, mic gain, speaker sensitivity, speaker gain, product casing layout, sound intensity, etc. Different echo situations require different AEC parameters to handle, so before starting to debug AEC parameters, it is necessary to ensure that the following prerequisites are completed:

- 1. The mic model must be fixed, and the sensitivity of different models may vary. The higher the sensitivity, the greater the echo. Therefore, it is advisable to choose a mic with lower sensitivity while meeting the requirements of the usage scenario.
- 2. The gain of the mic must be fixed, and the magnitude of the gain will affect the amplitude of the audio signal.
- 3. The speaker model must be fixed, and the power of different models may vary. The higher the power, the louder the sound, and the greater the echo. Therefore, it is advisable to choose a speaker with lower power while meeting the requirements of the usage scenario.
- 4. The gain of the speaker (including PA and DAC gain inside the digital chip) must be fixed. The larger the gain, the louder the speaker's volume, and the greater the echo. Therefore, it is possible to lower the gain of the speaker as much as possible while meeting the requirements of the usage scenario.

- 5. The product casing and sound chamber must be fixed. Different casings and sound chambers can cause different echo conditions of the equipment, so the product casing and sound chamber must be fixed.

The voice call application developed based on the aud-intf component needs to debug the effect of AEC echo cancellation, mainly by adjusting the following parameters: `echo depth` , `max amplitude` , `min amplitude` , `noise level` , and `noise param` .

The debugging steps are as follows:

- 1.Ensure that the two devices involved in the call (usually the phone and the board) are in remote mode, ensuring that the sound of the phone's speech is not captured by the board's mic, and then open the voice call;
- 2.Based on the echo effect heard, use the UART to send a serial command `aud-intf_set-aec_param_test {param value}` to set the values of each parameter and debug the AEC parameters online;
- 3.Send the serial command `aud_intf_set_aec_param_test ec_depth {value}` to set the `echo depth` . The larger the echo, the greater the value. When the value continues to increase but cannot improve the echo cancellation effect, stop setting the value;
- 4.Send the serial commands `aud_intf_set_aec_param_test TxRxThr {value}` and `aud_intf_set_aec_param_test TxRxFlr {value}` based on the range of echo size to set the values of `max amplitude` and `min amplitude` , optimizing the echo cancellation effect;
- 5.Send the serial command `aud_intf_set_aec_param_test ns_level {value}` based on the size of the background noise heard to set the `noise level` to optimize the effect of background noise elimination. The larger the background noise, the larger the value set;
- 4.Send the serial command `aud_intf_set_aec_param_test ns_para {value}` to set the value of `noise param` to 0, 1, and 2, respectively, and select the best value;
- 5.Send the serial command `aud_intf_get_aec_param_test` to obtain and record the online debugging results of all parameters, and set them as the default parameters for AEC during voice initialization.

Note:

- 1. To ensure the effectiveness of the serial command `aud-intf_set-aec_param_test {param value}` , open the macro `PROFIG-AUD-INTF-TEST=y`.
- 2. Failure to tune AEC parameters can result in occasional abnormal sounds heard by the opposite end of the board (usually on the mobile phone), such as occasional silence or partial echo.

12.1.4 4.Reference link

API Reference : Introduced the AEC API interface

User and Developer Guide : Introduced common usage scenarios of AEC

CHAPTER

THIRTEEN

FAQ

13.1 Common Problems with DVP

13.1.1 1. Introduction

This section mainly describes the common problems and solutions encountered during debugging and using DVP cameras.

Q: Unable to recognize camera

A: Use dvp Sample project, testing the adapted sample sensor, gc2145, If it cannot be recognized, it indicates that there is an error in the parameter configuration of the newly adapted sensor in dvp_xxx.c. The most fundamental issue is the I2C read-write address and CHIP_ID of the new sensor. If even gc2145 cannot recognize it, check whether the sensor power supply DVDD and IOVDD meet the protocol requirements, whether the GPIO controlling the power on the hardware is also GPIO28, and whether I2C is also corresponding to GPO0 and GPO1. In addition, attention should be paid to contact issues, which may also be due to insufficient physical connection, resulting in malfunction.

Q: Abnormal image output and printing: “Sensor’s yuyv data resolution is not right”.

A: The DVP data collected by the main control does not match the configured resolution. The VSync/HSync/PCLK signals can be captured through a logic analyzer. The following conditions must be followed: if the resolution configured for the main control is 640X480, then a VSync must contain 480 HSync pulses internally, and an HSync must contain 640 * 2=1280 PCLK pulses internally. If it does not meet the requirements, it will inevitably result in abnormal graphics. This problem may be caused by poor physical contact, and needs to be unplugged and reinstalled. Possible data collection anomalies may be caused by inconsistent line sequence with the default BK7258. Perhaps due to electromagnetic interference from the board, the sampling of the main control may be inaccurate in PCLK. A pull-up filter capacitor 8-22pf can be connected to PLK. Because the dvp lines are too long, it is recommended to wrap them with copper wire to reduce electromagnetic interference.

Q: Abnormal image output and printing: “sensor FIFO is full”.

A: The main control is receiving DVP data too slowly, causing sensor FIFO overflow. The solution can be tried: Reduce frame rate/reduce resolution/increase YUV_BUF hardware module clock (current default JPEG: 120MHz, YUV_BUF:120MHz, H264:120MHz).

Q: Abnormal image output and printing: “JPEG code rate is slow than sensor data rate”.

A: Indicates that the encoding speed is too slow. The solution can be to try reducing the frame rate/resolution/increasing the clock of the encoding hardware module (currently default JPEG: 120MHz H264:120MHz).

Q: Abnormal image output and printing: “h264 encode error”.

A: Indicates H264 encoding error, which may be caused by the sensor's frame gap time being too low, resulting in abnormal control; It is also possible that the previous frame has not been fully encoded before the start of the H264 encoding for the new frame. In this case, the software code has already covered and the relevant hardware modules can be reset directly.

Q: I2C abnormality after switching, camera cannot communicate normally.

A: This situation usually occurs when other peripherals share a set of I2C with DVP, and this I2C is cut off and used by other peripherals; It is recommended to use software I2C to prevent the problem of not working after reuse. Open the software I2C function macro control:

marco	value	implication
CONFIG_SIM_I2C	Y	Enable software to simulate I2C functionality

Q: Configure to h264/JPEG encoding mode, abnormal printing occurs: "···· size no match ····"

A: In general, the length of data transferred by DMA is not consistent with the actual encoding length. This will result in the software discarding such erroneous frames by default to prevent screen flickering and other issues, and resetting the corresponding module.

Q: The perspective is not in the center when creating the image.

A: This problem is caused by configuring the parameters (registers) of DVP. It is recommended to find an engineer from the sensor factory to reconfigure it. Currently, the configuration provided by the SDK only guarantees normal image output.

13.2 Common Problems with UVC

13.2.1 1. Introduction

When using UVC, some common issues may arise, and here are some debugging methods provided.

Q: After calling the UVC open interface, the camera does not enumerate (no prompt of successful connection)

A: This kind of problem is usually due to the camera not being powered on or insufficient power supply. It is recommended to supply 5.0V of voltage to the USB. The BK7258 controls the USB LDO through GPIO28, which can enable the USB power by pulling it high, and can shutdown the USB power by pulling it low. This can be configured through macros:

Q: Enumeration was successful, but no image is displayed.

A: This issue usually requires checking if the parameters are configured incorrectly, such as the camera not supporting the resolution you have configured, which would then require you to reconfigure it. When the resolution is configured incorrectly, you will see the following log: "uvc_camera_set_param, not support this format or resolution, please check!". If this is not the case, you may need to capture packets to analyze whether the UVC data packets are normal, including headers and valid data, and that the valid data is within the normal range.

Q: The frame rate is lower than expected.

A: The "expected" refers to the frame rate in comparison to a PC, for example, if you configure it to 30fps but the actual output is only half that or much lower, plug it into the computer to analyze whether the frame rate is also low on the computer, which can help rule out issues with the camera itself. If it's not an issue with the camera itself, then consider whether it's a problem with the SDK. This requires log analysis or packet capture analysis using a protocol analyzer.

Q: There are abnormal screen cutting issues in the image, and the image is incomplete.

A: This situation usually requires checking if there are any misconfigurations in the transmission mode, ISO/BULK, and then checking if there are any errors in the upper layer unpacking logic.

Q: The image displays abnormal halos and distortions.

A: This is generally considered an issue with the camera itself. Try plugging it into a PC to analyze whether the same problem occurs.

Q: When using UVC for H.264 pipeline encoding, the log "h264_encode_finish_handle 26430-65536, error:1" is printed.

A: There are two reasons for this log message:

- One is that when the first number (26430) is greater than the second number (65536), it indicates that the length of the I frame is greater than the length of the frame_buffer (the second number).

  Modification method: Increase the value of the macro CONFIG_H264_FRAME_SIZE to a larger number.

- The other is that "error:" has a value of 1, indicating that the length of the decoded JPEG image is incorrect. This suggests that the JPEG image has padding bits. To prevent image anomalies, the decoder performs strict length checks on the image internally.

Modification method: It is recommended to modify the camera firmware to ensure that the UVC output JPEG image does not have padding bits. Such padding bits are usually at the end and are either 0xFF or 0x00.

Another method is to lower the internal length check mechanism of the decoder, which is not recommended as it may lead to JPEG anomalies causing the screen to display incorrectly. Reference code modification.

```
//path        : bk_idk/middleware/driver/jpeg_dec_driver.c

bool jpeg_dec_comp_status(uint8_t *src, uint32_t src_size, uint32_t dec_size)
{
    ...

    if (max > 0 && strip + dec_size == src_size - JPEG_TAIL_SIZE)
    {
        ok = true;
    }
    else if (max > 0 && src_size - dec_size == 3 && src[src_size - 3] == 0x00)
    {
        ok = true;
    }
    else
    {
        // you can open this log, and  analysis the specific reasons
        LOGD("decoder_error, %u, %u, %u, %u\n", src_size, dec_size, strip, max);
    }
    return ok;
}
```

Q: The image is displayed normally on the PC end, but not on the board, and the log prints "cpu1:uvc_stre:W(8614):uvc_id0:30[227 28KB], uvc_id1: 0[0 0KB], uvc_id2:0[0 0KB], packets:[all:62568, err:0]";

A: In the above log, the seq is always 0 after uvc_idx, indicating that no images are output. The err value is not 0, indicating that there are error data in usb. There are two reasons for this log message:

- Please check the stability of the USB connection.
- Please verify that the resolution currently in use by the UVC is supported by the camera.

    - Please ensure that the port number being used matches, as the USB currently supports up to 3 ports, with the port range being: [1, 3]. When enabling UVC, make sure the accuracy of the structure media_device_t->port.

- genindex

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

O

R

S

T

U

V

X

Y