杰理AC54XX SDK使用说明

目录

[1前言 5](#_Toc19211)

[1.1 概述 5](#_Toc1805)

[1.2 重要概念 5](#_Toc30403)

[1.3 目录结构 5](#_Toc20707)

[1.4 约定 6](#_Toc18800)

[2系统初始化 7](#_Toc753)

[2.1 概述 7](#_Toc1318)

[2.2 重要概念 7](#_Toc27089)

[2.3 API参考 7](#_Toc21407)

[early\_initcall 7](#_Toc437)

[platform\_initcall 8](#_Toc23586)

[board\_init 9](#_Toc26940)

[initcall 9](#_Toc19837)

[module\_initcall 10](#_Toc7165)

[late\_initcall 11](#_Toc9444)

[3板级配置 13](#_Toc14961)

[3.1 概述 13](#_Toc178)

[3.2 重要概念 13](#_Toc4641)

[3.3 API参考 13](#_Toc1642)

[UART\_PLATFORM\_DATA 13](#_Toc28737)

[SD\_PLATFORM\_DATA 14](#_Toc23899)

[HW\_IIC\_PLATFORM\_DATA 16](#_Toc23068)

[SW\_IIC\_PLATFORM\_DATA 17](#_Toc3036)

[LCD\_PLATFORM\_DATA 18](#_Toc25092)

[USB\_PLATFORM\_DATA 20](#_Toc13055)

[ADKEY\_PLATFORM\_DATA 21](#_Toc17573)

[IOKEY\_PLATFORM\_DATA 23](#_Toc17696)

[VIDEO\_PLATFORM\_DATA 25](#_Toc6622)

[SPI\_PLATFORM\_DATA 27](#_Toc30149)

[SW\_SPI\_PLATFORM\_DATA 28](#_Toc32155)

[POWER\_PLATFORM\_DATA 29](#_Toc21929)

[4设备 31](#_Toc30785)

[4.1 概述 31](#_Toc25572)

[4.2 重要概念 31](#_Toc27558)

[4.3 API参考 31](#_Toc25999)

[REGISTER\_DEVICES 31](#_Toc12467)

[device\_init 32](#_Toc8026)

[dev\_online 33](#_Toc13084)

[dev\_open 34](#_Toc10469)

[dev\_read 35](#_Toc978)

[dev\_write 35](#_Toc4065)

[dev\_bulk\_read 36](#_Toc24609)

[dev\_bulk\_write 37](#_Toc28999)

[dev\_ioctl 38](#_Toc23189)

[dev\_close 39](#_Toc8479)

[5文件系统 41](#_Toc31763)

[5.1 概述 41](#_Toc32178)

[5.2 重要概念 41](#_Toc9940)

[5.3 API参考 41](#_Toc12573)

[6系统事件 42](#_Toc4645)

[6.1 概述 42](#_Toc31896)

[6.2 重要概念 42](#_Toc11531)

[6.3 API参考 42](#_Toc17146)

[7定时器 43](#_Toc26441)

[7.1 概述 43](#_Toc29748)

[7.2 重要概念 43](#_Toc7003)

[7.3 API参考 43](#_Toc18913)

[8录像 44](#_Toc26964)

[8.1 概述 44](#_Toc1308)

[8.2 重要概念 44](#_Toc4561)

[8.3 API参考 44](#_Toc8248)

[9拍照 45](#_Toc6492)

[9.1 概述 45](#_Toc15125)

[9.2 重要概念 45](#_Toc32747)

[9.3 API参考 45](#_Toc32538)

[10回放 46](#_Toc30053)

[10.1 概述 46](#_Toc24618)

[10.2 重要概念 46](#_Toc31576)

[10.3 API参考 46](#_Toc2568)

[11从机 47](#_Toc9533)

[11.1 概述 47](#_Toc1536)

[11.2 重要概念 47](#_Toc29696)

[11.3 API参考 47](#_Toc17115)

[12界面 48](#_Toc28325)

[12.1 概述 48](#_Toc14692)

[12.2 重要概念 48](#_Toc26948)

[12.3 API参考 48](#_Toc31229)

[12.4 开机动画 48](#_Toc13489)

[开机图片 48](#_Toc5158)

[开机动画 48](#_Toc1159)

[13系统配置项 50](#_Toc3801)

[13.1 概述 50](#_Toc1650)

[13.2 重要概念 50](#_Toc8490)

[13.3 API参考 50](#_Toc15597)

[6.2 video\_dec\_server接口说明 50](#_Toc5264)

[10. 摄像头YUV数据 51](#_Toc210)

[10.1 概述 51](#_Toc18108)

[10.2 获取数据步骤 51](#_Toc26722)

# 1前言

## 1.1 概述

杰理AC54系列SDK是基于杰理科技AC54系列芯片开发的通用型视频项目

平台，可用于行车记录仪、带WiFi行车记录仪、运动DV、飞控等项目的开发。

AC54系列芯片支持2路H264编码和1路JPG编码。双路编码最大规格为：

1：前视1080P H264 + 后视720P H264

2：前视1296P H264 + 后视720P JPG

H264编码封装为mov格式视频文件，JPG编码封装为avi格式视频文件。

## 1.2 重要概念

**APP层**

系统中的应用程序层，是整个系统的最顶层，一个APP对应一种模式。

APP层通过组合调用服务层和系统层提供的API函数,来实现不同的功能。

**服务层**

服务层在系统层之上，通过将系统层提供的API进行组合、封装来为APP层

提供模块化的功能，大大简化了APP层的业务逻辑。

**系统层**

系统层集合了操作系统、设备接口、文件系统、事件分发、定时器等功能，

为服务层和APP层提供所需要的API。

## 1.3 目录结构

**app：** 应用层程序目录，一个子目录表示一个方案

**cpu：** 各型号芯片相关模块驱动和下载工具目录。

**camera：** 各型号摄像头的驱动程序目录

**include\_lib：** lib文件和lib的头文件目录，**请勿修改**

**doc：** SDK文档目录

**UI\_工程**： 各种风格的UI工程目录

## 1.4 约定

1. 板级相关配置在对应的board\_xxx.c文件中
2. SDK配置宏在app\_config.h文件中
3. 设备video0 代表前视，video1代表模拟后拉，video3/video4代表usb后拉
4. video0使用设备iic0通讯，video1使用设备iic1通讯
5. video2为H264和JPG解码设备
6. fb0 为UI框架使用的显示设备，fb1为摄像头使用的显示设备
7. fb3为虚拟的显示设备，可用于获取摄像头数据

# 2系统初始化

## 2.1 概述

板子上电后，代码的执行顺序为maskrom-->uboot-->sdk。sdk启动后会

按照顺序依次执行初始化代码，完成必要的初始化工作。

## 2.2 重要概念

## 2.3 API参考

**early\_initcall**

【描述】

第一阶段初始化，负责一些模块的变量初始化。

【语法】

early\_initcall(int (\*func)(void));

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| func | 返回值为int型，参数为空的函数 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 失败，其值为错误码 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：system/init.h

【注意】

【举例】

static int timer1\_init(void)

{

return 0;

}

*early\_initcall(timer1\_init);*

**platform\_initcall**

【描述】

第二阶段初始化，负责部分外设模块的初始化。

【语法】

platform\_initcall(int (\*func)(void));

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| func | 返回值为int型，参数为空的函数 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 失败，其值为错误码 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：system/init.h

【注意】

【举例】

static int test\_func(void)

{

return 0;

}

*platform\_initcall(test\_func);*

**board\_init**

【描述】

第三阶段初始化，负责板子相关IO口初始化和设备注册。

【语法】

board\_init();

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
|  |  |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

【注意】

在函数board\_init被调用前，不能调用dev\_open函数打开设备。

【举例】

*void board\_init(void)*

{

gpio\_direction\_output(IO\_PORTB\_11, 0);

device\_init();

}

**initcall**

【描述】

第四阶段初始化。

【语法】

\_\_initcall(int (\*func)(void));

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| func | 返回值为int型，参数为空的函数 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 失败，其值为错误码 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：system/init.h

【注意】

【举例】

static int test\_func(void)

{

return 0;

}

*\_\_initcall(test\_func);*

**module\_initcall**

【描述】

第五阶段初始化。

【语法】

module\_initcall(int (\*func)(void));

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| func | 返回值为int型，参数为空的函数 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 失败，其值为错误码 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：system/init.h

【注意】

【举例】

static int test\_func(void)

{

return 0;

}

*module\_initcall(test\_func);*

**late\_initcall**

【描述】

第六阶段初始化。

【语法】

late\_initcall(int (\*func)(void));

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| func | 返回值为int型，参数为空的函数 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 失败，其值为错误码 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：system/init.h

【注意】

【举例】

static int test\_func(void)

{

return 0;

}

*late\_initcall(test\_func);*

# 3板级配置

## 3.1 概述

系统中所有跟板子相关的外设的IO口配置信息都在对应的board\_xxx.c文件中

指定，选择板子的宏在app\_config.h中定义。

## 3.2 重要概念

**设备**

系统中的设备并不等同于设备驱动，而是在设备驱动之上再加一层抽象化的接

口抽象出来的设备。这样上层的服务或APP调用的就是抽象化的接口而不依

赖于具体的硬件特性，保证了上层模块的稳定性。

## 3.3 API参考

**UART\_PLATFORM\_DATA**

【描述】

串口配置信息。

【语法】

UART*x*\_PLATFORM\_DATA\_BEGIN(uartx\_data)

.baudrate = ,

.tx\_pin = ,

.flags = ,

UART*x*\_PLATFORM\_DATA\_END()

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| uartx\_data | 变量名 |  |
| baudrate | 波特率 |  |
| tx\_pin | 串口的TX脚 |  |
| flags | 串口配置选项  UART\_DEBUG 用作调试串口 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：asm/uart.h

【注意】

【举例】

UART0\_PLATFORM\_DATA\_BEGIN(uart0\_data)

.baudrate = 115200,

.tx\_pin = IO\_PORTG\_14,

.flags = UART\_DEBUG,

UART0\_PLATFORM\_DATA\_END();

**SD\_PLATFORM\_DATA**

【描述】

SD卡配置信息。

【语法】

SD*x*\_PLATFORM\_DATA\_BEGIN(sd*x*\_data)

.port = ,

.priority = ,

.data\_width = ,

.speed = ,

.detect\_mode = ,

.detect\_func = ,

.power = ,

SDx\_PLATFORM\_DATA\_END

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| sdx\_data | 变量名 |  |
| port | 端口选择 |  |
| priority | 中断优先级 |  |
| data\_width | 几线模式 |  |
| speed | SD卡时钟速度 |  |
| detect\_mode | 检测模式 |  |
| detect\_func | 检测函数， 返回值1表示在线 |  |
| power | 电源控制 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：asm/sdmmc.h

【注意】

1. SD卡的时钟要能够被clk\_get(“sd”)的返回值整除

【举例】

int sdmmc\_1\_io\_detect(const struct sdmmc\_platform\_data \*data)

{

static u8 init = 0;

if (!init) {

init = 1;

gpio\_direction\_input(IO\_PORTA\_11);

gpio\_set\_pull\_up(IO\_PORTA\_11, 1);

gpio\_set\_pull\_down(IO\_PORTA\_11, 0);

}

return !gpio\_read(IO\_PORTA\_11);

}

static void sdmmc\_power(int on)

{

gpio\_direction\_output(IO\_PORTB\_11, !on);

}

*SD1\_PLATFORM\_DATA\_BEGIN(sd1\_data)*

.port = 'C',

.priority = 3,

.data\_width = 4,

.speed = 30000000,

.detect\_mode = SD\_IO\_DECT,

.detect\_func = sdmmc\_1\_io\_detect,

.power = sdmmc\_power,

*SD1\_PLATFORM\_DATA\_END()*

**HW\_IIC\_PLATFORM\_DATA**

【描述】

硬件iic配置信息。

【语法】

HW\_IIC*x*\_PLATFORM\_DATA\_BEGIN(hw\_iic*x*\_data)

.clk\_pin = ,

.dat\_pin = ,

.baudrate = ,

HW\_IIC*x*\_PLATFORM\_DATA\_END

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| hw\_iic*x*\_data | 变量名 |  |
| clk\_pin | clk脚 |  |
| dat\_pin | data脚 |  |
| baudrate | 时钟分频，取值范围0 -- 255 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：asm/iic.h

【注意】

【举例】

*HW\_IIC0\_PLATFORM\_DATA\_BEGIN(hw\_iic0\_data)*

.clk\_pin = IO\_PORTB\_04,

.dat\_pin = IO\_PORTB\_03,

.baudrate = 0x1f,

*HW\_IIC0\_PLATFORM\_DATA\_END()*

**SW\_IIC\_PLATFORM\_DATA**

【描述】

软件iic配置信息。

【语法】

SW\_IIC\_PLATFORM\_DATA\_BEGIN(sw\_iic*x*\_data)

.clk\_pin = ,

.dat\_pin = ,

.sw\_iic\_delay = ,

SW\_IIC\_PLATFORM\_DATA\_END

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| sw\_iic*x*\_data | 变量名 |  |
| clk\_pin | clk脚 |  |
| dat\_pin | data脚 |  |
| sw\_iic\_delay | IO翻转延时,单位为 1/system clk，相当于波特率 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/iic.h

【注意】

* + - * 软件IIC可以定义多个，只要参数sw\_iic*x*\_data不同即可

【举例】

*SW\_IIC\_PLATFORM\_DATA\_BEGIN(sw\_iic0\_data)*

.clk\_pin = IO\_PORTB\_00,

.dat\_pin = IO\_PORTB\_01,

.baudrate = 0x50,

*SW\_IIC\_PLATFORM\_DATA\_END()*

**LCD\_PLATFORM\_DATA**

【描述】

LCD配置信息。

【语法】

LCD\_PLATFORM\_DATA\_BEGIN(lcd\_data)

.lcd\_name = ,

.interface = ,

.lcd\_io = {

.backlight = ,

.backlight\_value = ,

.lcd\_reset = ,

.lcd\_cs = ,

.lcd\_rs = ,

.lcd\_spi\_ck = ,

.lcd\_spi\_di = ,

.lcd\_spi\_do = ,

}

LCD\_PLATFORM\_DATA\_END

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| lcd\_data | 变量名 |  |
| lcd\_name | lcd驱动名 |  |
| interface | LCD的接口类型 |  |
| backlight | 背光控制脚 |  |
| backlight\_value | 0：背光控制脚输出低背光打开  1：背光控制脚输出高背光打开 |  |
| lcd\_reset | LCD reset脚 |  |
| lcd\_cs | LCD cs脚，-1表示没有此脚 |  |
| lcd\_rs |  |  |
| lcd\_spi\_ck | LCD spi clk脚，-1表示没有此脚 |  |
| lcd\_spi\_di | LCD spi data in脚，-1表示没有此脚 |  |
| lcd\_spi\_do | LCD spi data out脚，-1表示没有此脚 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/lcd\_driver.h

【注意】

【举例】

*LCD\_PLATFORM\_DATA\_BEGIN(lcd\_data)*

.lcd\_name = (u8\*)"mipi\_4lane\_720p\_rm68200",

.interface = LCD\_MIPI,

.lcd\_io = {

.backlight = IO\_PORTG\_02,

.backlight\_value = 1,

.lcd\_reset = IO\_PORTG\_04,

.lcd\_cs = -1,

.lcd\_rs = -1,

.lcd\_spi\_ck= -1,

.lcd\_spi\_di= -1,

.lcd\_spi\_do= -1,

}

*LCD\_PLATFORM\_DATA\_END()*

**USB\_PLATFORM\_DATA**

【描述】

USB配置信息。

【语法】

USB\_PLATFORM\_DATA\_BEGIN(usb*x*\_data)

.id = ,

.online\_check\_cnt = ,

.offline\_check\_cnt = ,

.isr\_priority = ,

.host\_ot = ,

.host\_speed = ,

.slave\_ot = ,

.ctl\_irq\_int = ,

USB\_PLATFORM\_DATA\_END()

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| usbx\_data | 变量名 |  |
| id |  |  |
| online\_check\_cnt |  |  |
| offline\_check\_cnt |  |  |
| isr\_priority |  |  |
| host\_ot |  |  |
| host\_speed |  |  |
| slave\_ot |  |  |
| ctl\_irq\_int |  |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/usb.h

【注意】

【举例】

USB\_PLATFORM\_DATA\_BEGIN(usb0\_data)

.id = 0,

.online\_check\_cnt = 3,

.offline\_check\_cnt = 20,

.isr\_priority = 6,

.host\_ot = 20,

.host\_speed = 1,

.slave\_ot = 10,

.ctl\_irq\_int = HUSB0\_CTL\_INT,

USB\_PLATFORM\_DATA\_END()

**ADKEY\_PLATFORM\_DATA**

【描述】

AD按键配置信息。

【语法】

ADKEY\_PLATFORM\_DATA\_BEGIN(adkey\_data)

.io = ,

.ad\_channel = ,

.table = {

.ad\_value = {

},

.key\_value = {

},

},

ADKEY\_PLATFORM\_DATA\_END()

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| adkey\_data | 变量名 |  |
| io | ADKEY采样脚 |  |
| ad\_channel | 采样脚所对应的ad采样channel，  请参考芯片手册 |  |
| ad\_value | 数组变量，按键采样值 |  |
| key\_value | 数组变量，采样值对应的消息 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：asm/adkey.h

【注意】

* 按键个由宏ADKEY\_MAX\_NUM定义，默认为16
* 按键去抖默认值在ADKEY\_PLATFORM\_DATA\_END中定义

【举例】

ADKEY\_PLATFORM\_DATA\_BEGIN(adkey\_data)

.io = IO\_PORTB\_14,

.ad\_channel = 4,

.table = {

.ad\_value = {

ADKEY\_V\_0,

ADKEY\_V\_1,

ADKEY\_V\_2,

ADKEY\_V\_3,

ADKEY\_V\_4,

ADKEY\_V\_5,

ADKEY\_V\_6,

ADKEY\_V\_7,

ADKEY\_V\_8,

ADKEY\_V\_9,

ADKEY\_V\_10,

},

.key\_value = {

NO\_KEY, /\*0\*/

NO\_KEY,

NO\_KEY,

NO\_KEY,

NO\_KEY,

NO\_KEY, /\*5\*/

KEY\_UP,

KEY\_DOWN,

KEY\_OK,

KEY\_MODE,

NO\_KEY,

},

},

ADKEY\_PLATFORM\_DATA\_END()

**IOKEY\_PLATFORM\_DATA**

【描述】

IO按键配置信息。

【语法】

IOKEY\_PLATFORM\_DATA\_BEGIN(iokey\_data)

.num = ,

.port = ,

IOKEY\_PLATFORM\_DATA\_END()

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| iokey\_data | 变量名 |  |
| num | 按键个数 |  |
| port | struct iokey\_port类型指针  {  port: 按键IO口  press\_value: 按键按下时IO的值  key\_value: 按键对应的消息  } |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/iokey.h

【注意】

【举例】

const struct iokey\_port iokey\_list[] = {

{

.port = IO\_PORTA\_13,

.press\_value = 1,

.key\_value = KEY\_POWER,

},

{

.port = IO\_PORTA\_14,

.press\_value = 1,

.key\_value = KEY\_OK,

},

};

IOKEY\_PLATFORM\_DATA\_BEGIN(iokey\_data)

.num = ARRAY\_SIZE(iokey\_list),

.port = iokey\_list,

IOKEY\_PLATFORM\_DATA\_END()

**VIDEO\_PLATFORM\_DATA**

【描述】

VIDEO配置信息。

【语法】

VIDEO\_PLATFORM\_DATA\_BEGIN(videox\_data)

.data = ,

.num = ,

VIDEO\_PLATFORM\_DATA\_END()

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| videox\_data | 变量名 |  |
| data | struct video\_subdevice\_data 类型变量指针  {  tag： 标识符  data: 标识符对应的变量  }  tag 为VIDEO\_TAG\_CAMERA时，data指向  struct camera\_platform\_data 类型  {  xclk\_gpio: xclk引脚  reset\_gpio: 摄像头reset脚  online\_detect: 在线检测函数指针  pwdn\_gpio: 电源控制脚  power\_value: 0输出低上电，1输出高上电  interface：接口类型  csi2 {  data\_lane\_num:  clk\_inv:  d0\_rmap:  d0\_inv:  d1\_rmap:  d1\_inv:  tval\_hstt:  tval\_stto:  }  dvp {  pclk\_gpio:  hsync\_gpio  vsync\_gpio:  io\_function\_sel:  data\_gpio:  }  } |  |
| num | data指向数组项个数 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/camera.h

【注意】

【举例】

static const struct camera\_platform\_data camera0\_data = {

.xclk\_gpio = IO\_PORTA\_14,

.reset\_gpio = IO\_PORTA\_15,

.online\_detect = NULL,

.pwdn\_gpio = IO\_PORTE\_01,

.power\_value = 1,

.interface = SEN\_INTERFACE\_CSI2,

.csi2 = {

.data\_lane\_num = 2,

.clk\_inv = 1,

.d0\_rmap = CSI2\_X4\_LANE,

.d0\_inv = 1,

.d1\_rmap = CSI2\_X3\_LANE,

.d1\_inv = 1,

.tval\_hstt = 12,

.tval\_stto = 30,

}

};

static const struct video\_subdevice\_data video0\_subdev\_data[] = {

{ VIDEO\_TAG\_CAMERA, (void \*)&camera0\_data },

};

VIDEO\_PLATFORM\_DATA\_BEGIN(video0\_data) = {

.data = video0\_subdev\_data,

.num = ARRAY\_SIZE(video0\_subdev\_data),

VIDEO\_PLATFORM\_DATA\_END()

**SPI\_PLATFORM\_DATA**

【描述】

SPI配置信息。

【语法】

SPIx\_PLATFORM\_DATA\_BEGIN(spix\_data)

.clk = ,

.port = ,

.mode = ,

SPIx\_PLATFORM\_DATA\_END()

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| spix\_data | 变量名 |  |
| clk | spi的时钟，要能够被clk\_get(“spi”)整除 |  |
| port | spi出口 |  |
| mode | spi的模式 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：asm/spi.h

【注意】

【举例】

SPI0\_PLATFORM\_DATA\_BEGIN(spi0\_data)

.clk = 60000000,

.mode = SPI\_DUAL\_MODE,

.port = SPI\_PORTA\_0\_4\_PORTH15,

SPI0\_PLATFORM\_DATA\_END()

**SW\_SPI\_PLATFORM\_DATA**

【描述】

软件SPI配置信息。

【语法】

SW\_SPI\_PLATFORM\_DATA\_BEGIN(sw\_spi\_data)

.pin\_cs = ,

.pin\_clk = ,

.pin\_in = ,

.pin\_out = ,

SW\_SPI\_PLATFORM\_DATA\_END()

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| sw\_spi\_data | 变量名 |  |
| pin\_cs | cs脚 |  |
| pin\_clk | clk脚 |  |
| pin\_in | data in脚 |  |
| pin\_out | data out脚 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/av10\_spi.h

【注意】

* 此SPI只是用于和AV10通讯

【举例】

#ifdef CONFIG\_AV10\_SPI\_ENABLE

SW\_SPI\_PLATFORM\_DATA\_BEGIN(sw\_spi\_data)

.pin\_cs = IO\_PORTG\_15,

.pin\_clk = IO\_PORTE\_04,

.pin\_in = IO\_PORTE\_05,

.pin\_out = IO\_PORTE\_05,

SW\_SPI\_PLATFORM\_DATA\_END()

#endif

**POWER\_PLATFORM\_DATA**

【描述】

电源配置信息。

【语法】

POWER\_PLATFORM\_DATA\_BEGIN(power\_data)

.usb\_wkup = {

.gpio = ,

.edge = ,

.port\_en = ,

},

.gsen\_int\_wkup = {

.gpio = ,

.edge = ,

.port\_en = ,

},

.en\_pr = {

.wkup\_en = ,

.wkup\_power\_gpio = ,

},

POWER\_PLATFORM\_DATA\_END()

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| power\_data | 变量名 |  |
| usb\_wkup | USB唤醒  {  gpio: 唤醒脚  edge：触发边沿  port\_en:  } |  |
| gsen\_int\_wkup | Gsensor唤醒 |  |
| en\_pr |  |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：power\_ctrl.h

【注意】

【举例】

POWER\_PLATFORM\_DATA\_BEGIN(power\_data)

.usb\_wkup = {

.gpio = IO\_PORT\_PR\_01,

.edge = 0,

.port\_en = 1,

},

.gsen\_int\_wkup = {

.gpio = IO\_PORT\_PR\_03,

.edge = 0,

.port\_en = 1,

},

.en\_pr = {

.wkup\_en = 1,

.wkup\_power\_gpio = IO\_PORT\_PR\_00,

},

POWER\_PLATFORM\_DATA\_END()

# 

# 4设备接口

## 4.1 概述

## 4.2 重要概念

## 4.3 API参考

### REGISTER\_DEVICES

【描述】

定义静态的设备信息。

【语法】

REGISTER\_DEVICES(device\_table) = {

{

.name = ,

.ops = ,

.priv\_data = ,

},

};

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| device\_table | 变量名 |  |
| name | 设备名 |  |
| ops | 设备驱动函数 |  |
| priv\_data | 设备的配置参数 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
|  |  |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

【举例】

REGISTER\_DEVICES(device\_table) = {

{ "lcd", &lcd\_dev\_ops, (void\*)&lcd\_data},

{ "iic1", &iic\_dev\_ops, (void \*)&hw\_iic0\_data },

{ "iic0", &iic\_dev\_ops, (void \*)&sw\_iic\_data },

};

### device\_init

【描述】

设备接口初始化。

【语法】

int devices\_init(void);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
|  |  |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 错误码 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

* 此函数在通常在board\_init函数中调用

【举例】

void board\_init()

{

devices\_init();

}

### dev\_online

【描述】

打开设备。

【语法】

int dev\_open(const char \*name);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| name | 设备名 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 离线 |
| 1 | 在线 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

* 只有部分设备支持此函数

【举例】

if (device\_online(“sd0”)) {

}

### dev\_open

【描述】

打开设备。

【语法】

void \*dev\_open(const char \*name, void \*arg);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| name | 设备名 | 输入 |
| arg | 设备的私有参数 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| NULL | 错误 |
| 非NULL | 成功 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

* 此函数要在devices\_init函数被调用后才能调用

【举例】

void \*iic = dev\_open(“iic0”, NULL);

### dev\_read

【描述】

从设备读取数据。

【语法】

int dev\_read(void \*device, void \*buf, u32 len);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| device | 设备句柄 | 输入 |
| buf | 数据存放buf | 输入 |
| len | 读取长度 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 等于len | 成功 |
| 不等于len | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

【举例】

### dev\_write

【描述】

写数据到设备。

【语法】

int dev\_write(void \*device, void \*buf, u32 len);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| device | 设备句柄 | 输入 |
| buf | 数据存放buf | 输入 |
| len | 写入长度 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 等于len | 成功 |
| 不等于len | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

【举例】

### dev\_bulk\_read

【描述】

块设备读取。

【语法】

int dev\_bulk\_read(void \*device, void \*buf, u32 offset, u32 len);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| device | 设备句柄 | 输入 |
| buf | 数据存放buf | 输入 |
| offset | 读取地址 | 输入 |
| len | 读取长度 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 等于len | 成功 |
| 不等于len | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

【举例】

### dev\_bulk\_write

【描述】

块设备读取。

【语法】

int dev\_bulk\_write(void \*device, void \*buf, u32 offset, u32 len);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| device | 设备句柄 | 输入 |
| buf | 数据存放buf | 输入 |
| offset | 写入地址 | 输入 |
| len | 写入长度 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 等于len | 成功 |
| 不等于len | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

【举例】

### dev\_ioctl

【描述】

配置设备。

【语法】

int dev\_ioctl(void \*device, int cmd, u32 arg);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| device | 设备句柄 | 输入 |
| cmd | 配置命令 | 输入 |
| arg | 配置参数 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

* 通用的配置命令在device/ioctl\_cmds.h文件中定义
* 设备的私有命令在自己的.h文件中定义

【举例】

### dev\_close

【描述】

关闭设备。

【语法】

int dev\_close(void \*device);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| device | 设备句柄 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：device/device.h

【注意】

【举例】

## 4.4 添加设备

**1. 定义struct device\_operations类型变量，按需要实现其成员函数。**

【举例】

static struct device xxx\_device;

static bool xxx\_dev\_online(struct dev\_node \*node)

{

if (设备在线) {

return true;

} else {

return false;

}

}

static int xxx\_dev\_init(struct dev\_node \*node, void \*arg)

{

struct xxx\_platform\_data \*pd = (struct xxx\_platform\_data \*)arg;

//其它初始化代码

return 0;

}

static int xxx\_dev\_open(struct dev\_node \*node, struct device \*\*device, void \*arg)

{

//先执行设备打开动作

if (打开成功) {

\*device = &xxx\_device;

return 0;

} else {

return -ERROR\_CODE;

}

}

static int xxx\_dev\_ioctl(struct device \*device, u32 cmd, u32 arg)

{

switch(cmd) {

}

//返回0 表示成功， 负值表示错误码

return 0;

}

const struct device\_operations xxx\_dev\_ops = {

.online = xxx\_dev\_online,

.init = xxx\_dev\_init,

.open = xxx\_dev\_open,

.ioctl = xxx\_dev\_ioctl,

}

1. **添加设备节点**

在对应的board\_xxx.c文件中添加以下信息

//设备配置信息，如不需要可不定义

const struct xxx\_platform\_data xxx\_data = {

成员变量负值

};

REGISTER\_DEVICES(device\_table) = {

{“xxx”, &xxx\_dev\_ops, &xxx\_data},

};

# 5文件系统

## 5.1 概述

## 5.2 重要概念

## 5.3 API参考

### mount

【描述】

挂载设备。

【语法】

struct imount \*mount(const char \*dev\_name, const char \*path,

const char \*fs\_type, void \*dev\_arg);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| dev\_name | 设备名 | 输入 |
| path | 挂载到目标路径 | 输入 |
| fs\_type | 文件系统类型 | 输入 |
| dev\_arg | 调用dev\_open时传给设备的私有参数 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 非NULL | 成功 |
| NULL | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

* 如果设备具有多个分区，则挂载完成后会有多个磁盘，磁盘个数参考imount->part\_num, 盘符从大写字母’C’开始递增。

【举例】

struct imount \*mt = mount(“sd0”, “storage/sd0”, “fat”, NULL);

if (mt) {

printf(“分区数: %d\n”, mt->part\_num);

}

### unmount

【描述】

卸载设备。

【语法】

int mount(const char \*path);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| path | 卸载路径 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 非0 | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

【举例】

struct imount \*mt = mount(“sd0”, “storage/sd0”, “fat”, NULL);

unmount(“storage/sd0”);

### fopen

【描述】

打开文件。

【语法】

FILE \*fopen(const char \*path, const char \*mode);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| path | 文件完整路径 | 输入 |
| mode | 打开方式：  “r”: 只读， 文件不存在时返回失败  “w”: 只写，文件不存在时返回失败  “w+”: 创建，文件不存在时创建文件 |  |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 非NULL | 成功 |
| NULL | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

【举例】

struct imount \*mt = mount(“sd0”, “storage/sd0”, “fat”, NULL);

FILE \*file = fopen(“storage/sd0/C/test.bin”, “r”);

if (file == NULL) {

file = fopen(“storage/sd0/C/test.bin”, “w+”);

}

### fread

【描述】

读文件。

【语法】

int fread(FILE \*file, void \*buf, u32 len);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| file | 文件句柄 | 输入 |
| buf | 数据buf | 输入 |
| len | 读取数据长度 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 等于len | 成功 |
| 小于len | 文件结束或错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

【举例】

struct imount \*mt = mount(“sd0”, “storage/sd0”, “fat”, NULL);

u8 buf[512];

FILE \*file = fopen(“storage/sd0/C/test.bin”, “r”);

if (file) {

fread(file, buf, 512);

}

### fwrite

【描述】

写文件。

【语法】

int fwrite(FILE \*file, void \*buf, u32 len);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| file | 文件句柄 | 输入 |
| buf | 数据buf | 输入 |
| len | 写入数据长度 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 等于len | 成功 |
| 小于len | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

* 对于fat文件系统，写入长度尽可能按512对齐以便于获得更快的写入速度

【举例】

struct imount \*mt = mount(“sd0”, “storage/sd0”, “fat”, NULL);

u8 buf[512];

FILE \*file = fopen(“storage/sd0/C/test.bin”, “w+”);

if (file) {

memset(buf, 0x55, 512);

fwrite(file, buf, 512);

}

### fseek

【描述】

设置文件指针。

【语法】

int fseek(FILE \*file, int offset, int orig);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| file | 文件句柄 | 输入 |
| offset | 偏移值 | 输入 |
| orig | 偏移起始位置  SEEK\_SET: 从文件开始位置seek  SEEK\_CUR: 从文件当前位置seek  SEEK\_END: 从文件结束位置往前seek | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 等于offset | 成功 |
| 负值 | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

【举例】

struct imount \*mt = mount(“sd0”, “storage/sd0”, “fat”, NULL);

u8 buf[512];

FILE \*file = fopen(“storage/sd0/C/test.bin”, “r”);

if (file) {

fseek(file, 1000, SEEK\_SET);

fread(file, buf, 200);

}

### fclose

【描述】

关闭文件。

【语法】

int fclose(FILE \*file);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| file | 文件句柄 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 负值 | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

【举例】

struct imount \*mt = mount(“sd0”, “storage/sd0”, “fat”, NULL);

u8 buf[512];

FILE \*file = fopen(“storage/sd0/C/test.bin”, “r”);

if (file) {

fseek(file, 1000, SEEK\_SET);

fread(file, buf, 200);

*fclose(file);*

}

### fdelete

【描述】

删除文件。

【语法】

int fdelete(FILE \*file);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| file | 文件句柄 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 0 | 成功 |
| 负值 | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

【举例】

FILE \*file = fopen(“storage/sd1/C/test.bin”, “r”);

if (file) {

fdelete(file);

}

### fget\_name

【描述】

获取文件名。

【语法】

int fget\_name(FILE \*file, u8 \*name, int len);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| file | 文件句柄 | 输入 |
| name | 返回的文件名 | 输入 |
| len | name的长度，小于26时，只获取短文件名 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 大于0 | 文件名的长度 |
| 负值 | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

【举例】

u8 name[64];

int len = fget\_name(file, name, sizeof(name));

### fscan

【描述】

扫描指定类型的文件。

【语法】

struct vfscan \* fscan(const char \*path, const char \*arg);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| path | 扫描的路径 | 输入 |
| arg | 扫描参数：  -t 指定文件类型  -r 包含子目录  -d 包含文件夹  -a 指定文件属性 -ar 只读文件、-a/r 非只读文件  -s 排序方式， -sn按文件号排序 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 非NULL | 成功 |
| NULL | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

【举例】

//搜索sd1 DCIM/1目录下的属性为只读的mov和avi文件，按文件序号排序

struct vfscan \*fs = fscan(“storage/sd1/C/DCIM/1”, “-tMOVAVI -d -ar -sn”);

### fselect

【描述】

打开扫描的文件。

【语法】

FILE \* fselect(struct vfscan \*fs, int sel\_mode，int arg);

【参数】

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| fs | fscan 返回值 | 输入 |
| sel\_mode | 选择方式：  FSEL\_FIRST\_FILE 选择第一个文件  FSEL\_NEXT\_FILE 选择下一个文件  FSEL\_PREV\_FILE 选择上一个文件  FSEL\_LAST\_FILE 选择最后一个文件  FSEL\_BY\_NUMBER 选择指定序号的文件 | 输入 |
| arg | sel\_mode为FSEL\_BY\_NUMBER时，用来指定文件序号，从1开始 | 输入 |

【返回值】

|  |  |
| --- | --- |
| 返回值 | 描述 |
| 非NULL | 成功 |
| NULL | 错误 |

【错误码】

|  |  |
| --- | --- |
| 返回值 | 含义 |
|  |  |

【需求】

* 头文件：fs/fs.h

【注意】

* fselect打开的文件不用时也要调用fclose来关闭文件

【举例】

//搜索sd1 DCIM/1目录下的属性为只读的mov和avi文件，按文件序号排序

struct vfscan \*fs = fscan(“storage/sd1/C/DCIM/1”, “-tMOVAVI -d -ar -sn”);

if (fs) {

FILE \*file = fselect(fs, FSEL\_FIRST\_FILE, 0);

}

# 6系统事件

## 6.1 概述

## 6.2 重要概念

## 6.3 API参考

# 7定时器

## 7.1 概述

## 7.2 重要概念

## 7.3 API参考

# 8录像

## 8.1 概述

录像模式通过调用video\_server提供的接口，实现了前后双路显示及窗口切换等功能，通过ui\_server提供的接口显示界面。

## 8.2 重要概念

## 8.3 API参考

# 9拍照

## 9.1 概述

## 9.2 重要概念

## 9.3 API参考

# 10回放

## 10.1 概述

回放模式支持mov文件、avi文件和jpg文件的解码显示。回放模式依赖的服务为video\_dec\_server

## 10.2 重要概念

## 10.3 API参考

# 11从机

## 11.1 概述

## 11.2 重要概念

## 11.3 API参考

# 12界面

## 12.1 概述

## 12.2 重要概念

## 12.3 API参考

## 12.4 开机动画

SDK支持显示开机图片和播放开机动画功能，开机图片要求是jpg格式，开机动画要求是通过sdk提供的工具转换出来的mov格式。如果下载目录cpu/AC54xx/tools/audlogo/ 下存在mov文件则播放mov文件否则显示jpg图片并播放开机音乐。

### 开机图片

修改开机图片可通过以下两种方式进行：

1. 如果图片尺寸不变：
   1. 替换目录”UI\_工程\ui\_1280x720\_key\开机界面\project\config\开机图

片\”下面的图片。

* 1. 运行”UI\_工程\ui\_1280x720\_key\开机界面\step2-打开UI资源生成工

具.bat”。

1. 如果图片尺寸改变：
   1. 替换目录”UI\_工程\ui\_1280x720\_key\开机界面\project\config\开机图

片\”下面的图片。

* 1. 运行”UI\_工程\ui\_1280x720\_key\开机界面\step1-打开UI绘图工

具.bat”。点击”工程缩放”,输入屏幕的尺寸，确认并保存。

* 1. 运行”UI\_工程\ui\_1280x720\_key\开机界面\step2-打开UI资源生成工

具.bat”。

### 开机动画

生成开机动画步骤：

1. 打开“开机动画制作工具\vJL.exe”,点击“打开视频文件”选择带转换的

视频文件，填写配置信息，点击左侧“选择输出目录”，将目录选择为

“cpu\AC54xx\tools\audlogo\”，点击“开始转换”。

1. 将转换出来的文件重名名为boot.mov （开机动画）
2. 重复步骤1，将转换出来的文件重名名为off.mov （关机动画）

注: 如果使用开机动画，则可以将目录“cpu\AC54xx\tools\audlogo\”下面的开机音乐文件poweron.adp和poweroff.adp删除。

# 13系统配置项

## 13.1 概述

系统的配置信息已key-value的方式存储，存储介质可选择为RTC内部ram或spiflash。

## 13.2 重要概念

## 13.3 API参考

### 6.2 video\_dec\_server接口说明

1. 打开服务

struct server \*dec = server\_open(“video\_dec\_server”, “video2”);

1. 注册事件回调函数

server\_register\_event\_handler(dec, NULL, dec\_event\_handler);

3. 启动解码

union video\_dec\_req req = {0};

req.dec.fb = “fb1”;

req.dec.audio\_p.buf = NULL;

req.dec.audio\_p.buf\_len = 10\*1024;

req.dec.file = fopen(“storage/sd1/C/DCIM/1/VID\_001.MOV”, “r”);

server\_request(dec, VIDEO\_REQ\_DEC\_START, &req);

1. 暂停/继续

server\_request(dec, VIDEO\_REQ\_DEC\_PLAY\_PAUSE, NULL);

1. 快进/快退

union video\_dec\_req req = {0};

req.f.on = true; // 开始快进快退

req.f.times = 4; // 4倍速度

server\_request(dec, VIDEO\_REQ\_DEC\_FF, &req);

1. 停止解码

server\_request(dec, VIDEO\_REQ\_DEC\_STOP, NULL);

1. 关闭服务

server\_close(dec);

注：详细的参数说明请参考代码内注释

## 10. 摄像头YUV数据

### 10.1 概述

摄像头数据格式为YUV420，可借助虚拟设备fb3获取摄像头数据。

### 10.2 获取数据步骤

1. 打开服务

struct server \*display;

display = server\_open(“video\_server”, “video0”);

1. 打开设备fb3

struct fb\_var\_screeninfo info = {0};

void \*fb = dev\_open(“fb3”, NULL);

// 设置图层大小

info.buf\_xres = info.xres = 640;

info.buf\_yres = info.yres = 480;

dev\_ioctl(fb, FBSET\_VSCREENINFO, (u32)&info);

// 设置显存buf个数

dev\_ioctl(fb, FBIOSET\_FBUFFER\_NUM, 2);

3. 请求输出

union video\_req req = {0};

req.display.fb = “fb3”;

req.display.state = VIDEO\_STATE\_START;

server\_request(display, VIDEO\_REQ\_DISPLAY, &req);

1. 获取数据 (格式为YUV420)

struct fb\_map\_user map;

while (1) {

dev\_ioctl(fb, FBIOGET\_FBUFFER\_INUSED, (u32)&map);

if (map.yaddr != 0) {

/\*

map.yaddr: Y分量数据地址, 长度 width \* height

map.uaddr: U分量数据地址, 长度 width \* height / 4

map.vaddr: V分量数据地址, 长度 width \* height / 4

\*/

dev\_ioctl(fb, FBIOPUT\_FBUFFER\_INUSED, (u32)&map);

}

}

1. 关闭设备
   1. 关闭服务

union video\_req req;

req.display.state = VIDEO\_STATE\_STOP;

server\_request(display, VIDEO\_REQ\_DISPLAY, &req)

server\_close(display);

b) 关闭fb

dev\_ioctl(fb, FBIOSET\_FBUFFER\_NUM, 0);

dev\_close(fb);