



## TXW81x 视频方案开发指南



珠海泰芯半导体有限公司  
Zhuhai Taixin Semiconductor Co., Ltd

珠海市高新区港湾一号科创园港 11 栋 3 楼

保密等级	A	TXW81x 视频方案开发指南	文件编号	TX-0000
发行日期	2023-11-10		文件版本	V1.0

# 责任与版权

## 责任限制

由于产品版本升级或者其他原因，本文档会不定期更新。除非另行约定，泰芯半导体有限公司对本文档所有内容不提供任何担保或授权。

客户应在遵守法律、法规和安全要求的前提下进行产品设计，并做充分验证。泰芯半导体有限公司对应用帮助或客户产品设计不承担任何义务。客户应对其使用泰芯半导体有限公司的产品和应用自行负责。

在适用法律允许的范围内，泰芯半导体有限公司在任何情况下，都不对因使用本文档相关内容及本文档描述的产品而产生的损失和损害进行超过购买支付价款的赔偿（除在涉及人身伤害的情况中根据适用的法律规定的损害赔偿外）。

## 版权申明

泰芯半导体有限公司保留随时修改本文档中任何信息的权利，无需提前通知且不承担任何责任。

未经泰芯半导体有限公司书面同意，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。除非获得相关权利人的许可，否则，任何人不能以任何形式对前述软件进行复制、分发、修改、摘录、反编译、反汇编、解密、反向工程、出租、转让、分许可等侵犯本文档描述的享有版权的软件版权的行为，但是适用法禁止此类限制的除外。



珠海泰芯半导体有限公司  
Zhuhai Taixin Semiconductor Co., Ltd

珠海市高新区港湾一号科技园港 11 栋 3 楼

版权所有 侵权必究  
Copyright © 2023 by Tai Xin All rights reserved


保密等级	A	TXW81x 视频方案开发指南	文件编号	TX-0000
发行日期	2023-11-10		文件版本	V1.0


修订记录

日期	版本	描 述	修订人
2023-11-10	V1.0	初始版本	TX

	珠海泰芯半导体有限公司 Zhuhai Taixin Semiconductor Co.,Ltd	珠海市高新区港湾一号科创园港 11 栋 3 楼
---	--	-------------------------

版权所有 侵权必究 Copyright © 2023 by Tai Xin All rights reserved
--

保密等级	A	TXW81x 视频方案开发指南	文件编号	TX-0000
发行日期	2023-11-10		文件版本	V1.0
<div>目录</div> <div><div>TXW81x 视频方案开发指南..... 1</div><div><div>1. 概述..... 1</div><div>2. 硬件开发板..... 2</div><div><div>2.1. 音视频开发板..... 2</div><div><div>2.1.1. 音视频开发板接口介绍..... 2</div></div></div><div>3. 视频开发配置流程..... 3</div><div><div>3.1. 视频方案相关配置..... 3</div></div><div>4. DVP 摄像头..... 4</div><div><div>4.1. DVP 摄像头相关参数配置..... 4</div><div>4.2. DVP 摄像头添加..... 5</div><div>4.3. VPP 功能说明..... 5</div><div><div>4.3.1. VPP 已有功能优化..... 5</div><div>4.3.2. VPP 新增加功能..... 6</div><div>4.3.3. VPP 中断..... 7</div><div>4.3.4. api 接口说明..... 7</div></div><div>4.4. mjpeg 编码参数配置..... 7</div><div>4.5. DVP 图片的获取流程..... 9</div><div>4.6. DVP 数据流说明..... 12</div><div>4.7. DVP 重要打印说明..... 12</div></div><div>5. SPI 摄像头..... 14</div><div><div>5.1. SPI 摄像头配置..... 14</div></div></div></div>				
<div></div>		<div>珠海泰芯半导体有限公司</div> <div>Zhuhai Taixin Semiconductor Co.,Ltd</div>	<div>珠海市高新区港湾一号科创园港 11 栋 3 楼</div>	
<div>版权所有 侵权必究</div> <div>Copyright © 2023 by Tai Xin All rights reserved</div>				

保密等级	A	TXW81x 视频方案开发指南	文件编号	TX-0000
发行日期	2023-11-10		文件版本	V1.0
<div>5.2. PRC 功能联合 SPI 摄像头..... 15</div> <div>6. SCALE 模块..... 17</div> <div>6.1. scale1..... 17</div> <div>6.2. scale2..... 18</div> <div>6.3. scale3..... 18</div> <div>7. LCDC 屏显说明..... 19</div> <div>7.1. osd 推屏..... 19</div> <div>7.2. video 推屏..... 20</div> <div>7.3. jpeg 推屏..... 20</div> <div>7.4. 源数据推屏..... 21</div> <div>7.5. 屏配置..... 21</div> <div>7.6. 旋转问题..... 23</div> <div>7.7. 屏驱动 api..... 23</div> <div>8. USB 摄像头..... 24</div> <div>8.1. USB 摄像头参数配置..... 24</div> <div>8.2. USB 摄像头获取过程..... 24</div> <div>8.3. USB 重要打印说明..... 26</div>				
		珠海泰芯半导体有限公司 Zhuhai Taixin Semiconductor Co.,Ltd	珠海市高新区港湾一号科创园港 11 栋 3 楼	
版权所有 侵权必究 Copyright © 2023 by Tai Xin All rights reserved				

## 1. 概述

本文主要描述视频开发流程。

本文档主要适用于以下工程师：

- 技术支持工程师
- 方案软件开发工程师

本文档适用的产品范围：

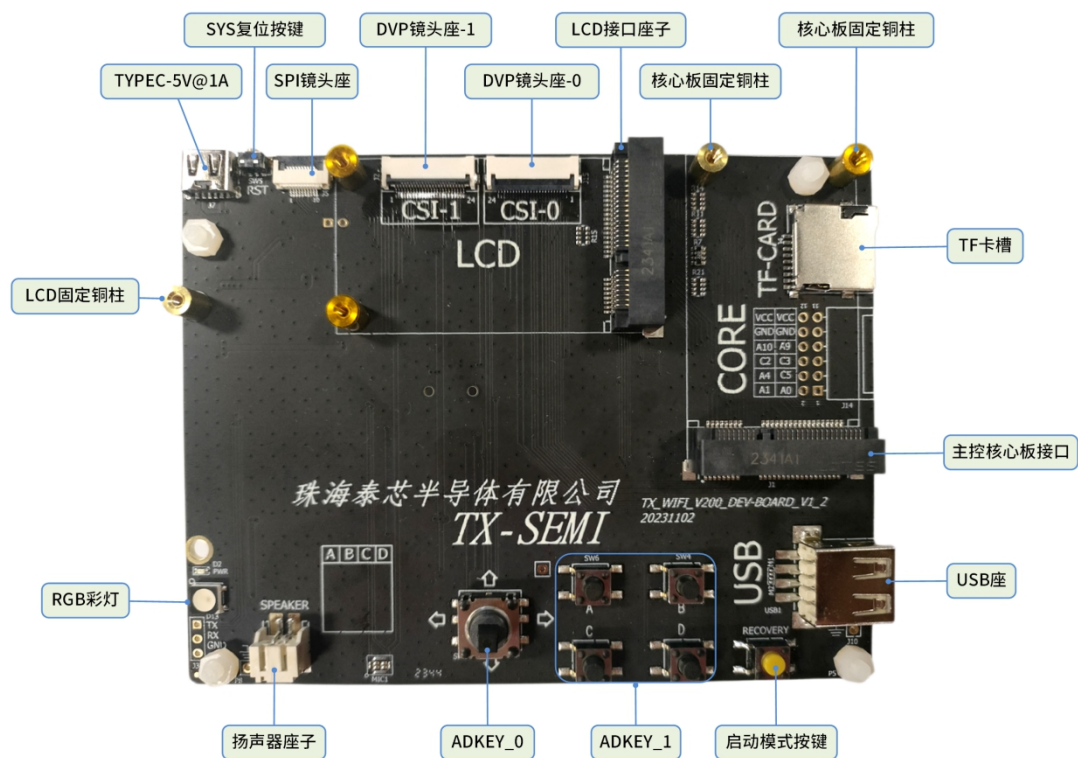
型号	封装	包装
TXW81x		

## 2. 硬件开发板

为了快速入门和方案评估，我们提供各种应用场景的开发板。

### 2.1. 音视频开发板

#### 2.1.1. 音视频开发板接口介绍



#### 特殊说明

**模式启动按键：**此按键可以一键拯救系统，在芯片上电即跑死，cklink 烧写和其他升级都失效的情况下使用。

### 3. 视频开发配置流程

#### 3.1. 视频方案相关配置

方案配置主要看 project\_config.h 配置, 以下配置都是基于 sdk 的图传协议。

```
17:
18: #define USB_HOST_EN          1
19: #define MACBUS_USB
20: #define BLE_PAIR_NET        0
21: #define WIRELESS_PAIR_CODE  0
22:
23:
24: #define USB_EN              1
25: #define DVP_EN              1
26: #define VPP_EN              DVP_EN
27: #define PRC_EN              1
28: #define JPG_EN              1
29: #define LCD_EN              1
30: #define SCALE_EN            1
31: #define SDH_EN              1
32: #define FS_EN               1
33: #define SD_SAVE              (0&&SDH_EN&&FS_EN&&JPG_EN)
34:
35: #define VCAM_EN              (0 || DVP_EN)
36:
37:
38: #define NEW_FRAME            1
39: #define OPENDML_EN           1
40: #define UART_INPUT_EN        1
41: #define UART_FLY_CTRL_EN     0
42: #define PWM_EN               0
43:
44:
```

- 1) USB\_EN 启动USB摄像头
- 2) DVP\_EN 启动DVP摄像头
- 3) VPP\_EN 为DVP的下层接口模块, 方便后面增加mipi接口
- 4) PRC\_EN 为PRC功能, 用于将摄像头(如SPI摄像头, 或者固定的yuv数据图像)输入的yuv数据自动传输到JPEG中
- 5) JPG\_EN 启动JPG的功能, 两个JPG均能使用
- 6) LCD\_EN 启动LCD控制器
- 7) SCALE\_EN SCALE功能打开, SCALE包含 1、2、3 个模块, 请细读scale说明
- 8) SDH\_EN、FS\_EN是sd卡与文件系统初始化
- 9) NEW\_FRAME 用于支持数据流处理功能, 默认打开
- 10) OPENDML\_EN 在需要录像的时候, 需要启动(仅仅支持sdk的录像格式, 其他格式需要自己实现)



## 4. DVP 摄像头

### 4.1. DVP 摄像头相关参数配置

采用VGA摄像头的时候，IMAGE\_W与IMAGE\_H配置为VGA。对应宏定义#define CMOS\_AUTO\_LOAD 1打开(位于project\_config.h),可以自动选择所有支持的摄像头,如果不使用自动选择,只希望支持某一款摄像头,则将对摄像头宏打开,例如:#define DEV\_SENSOR\_GC0308 1(位于project\_config.h),支持的摄像头以及宏定义在csi.h,如图:

```
#ifndef DEV_SENSOR_OV7725
#define DEV_SENSOR_OV7725          (0 || CMOS_AUTO_LOAD)
#endif

#ifndef DEV_SENSOR_OV7670
#define DEV_SENSOR_OV7670          (0 || CMOS_AUTO_LOAD)
#endif

#ifndef DEV_SENSOR_GC0308
#define DEV_SENSOR_GC0308          (0 || CMOS_AUTO_LOAD)
#endif

#ifndef DEV_SENSOR_GC0309
#define DEV_SENSOR_GC0309          (0 || CMOS_AUTO_LOAD)
#endif

#ifndef DEV_SENSOR_GC0328
#define DEV_SENSOR_GC0328          (0 || CMOS_AUTO_LOAD)
#endif

#ifndef DEV_SENSOR_BF3A03
#define DEV_SENSOR_BF3A03          (0 || CMOS_AUTO_LOAD)
#endif

#ifndef DEV_SENSOR_BF3703
#define DEV_SENSOR_BF3703          (0 || CMOS_AUTO_LOAD)
#endif

#ifndef DEV_SENSOR_OV2640
#define DEV_SENSOR_OV2640          0
#endif

#ifndef DEV_SENSOR_BF2013
#define DEV_SENSOR_BF2013          (0 || CMOS_AUTO_LOAD)
#endif

#ifndef DEV_SENSOR_XC7016_H63
#define DEV_SENSOR_XC7016_H63      0
#endif

#ifndef DEV_SENSOR_XC7011_H63
#define DEV_SENSOR_XC7011_H63      0
#endif

#ifndef DEV_SENSOR_XC7011_GC1054
#define DEV_SENSOR_XC7011_GC1054   0
#endif

#ifndef DEV_SENSOR_XCG532
#define DEV_SENSOR_XCG532          0
#endif

#ifndef DEV_SENSOR_GC2145
#define DEV_SENSOR_GC2145          0
#endif
```

720p摄像头处理形式分两种：

- 1) 外接isp芯片
- 2) 自带isp功能摄像头

如果所需支持摄像头为无 isp 功能摄像头，则需外接 isp 芯片，目前已适合两款 isp 芯片，XC7016 和 XC7011。当然，如果芯片直接支持 isp 功能，能正常输出已处理好的 YUV 数据，则可直接接到本主控上。在配置上 IMAGE\_W 与 IMAGE\_H 配置为 720P。

## 4.2. DVP 摄像头添加

请参考已有摄像头的配置文件 `sensor_xxx.c`，先添加摄像头的配置文件，并记录对的数据结构体 `_Sensor_Adpt_` 与 `_Sensor_Ident`。

`_Sensor_Ident` 包含当前摄像头的读写命令，命令与数据操作长度，ID 地址与 ID 值，在 `sensorCheckId` 中使用此配置信息去判断，当前插入的摄像头是否吻合此配置。

`_Sensor_Adpt_` 包含的重要信息有 `pixelw, pixelh, hsyn, vsyn, init, mclk`。`pixelw` 与 `pixelh` 为当前图像配置的输入分辨率。`hsyn` 与 `vsyn` 为当前摄像头的配置中，`hsync` 与 `vsync` 的有效电平是高电平还是低电平。`mclk` 为当前配置摄像头被插入时，系统给此摄像头提供的 `mclk` 频率。`init` 为当前摄像头的初始化配置信息，默认配置输出为 `yuyv`。

配置摄像头的配置文件 `sensor_xxx.c` 配置好后，请在 `csi.h` 中加入对应的声明，并在 `devSensorInitTable` 与 `devSensorOPTable` 添加对应摄像头的信息。

## 4.3. VPP 功能说明

VPP 模块其实是从 80x 中的 DVP 模块中抽离出来，并再进行更多功能优化的独立模块。此模块方便后面外设增加除 DVP 以外，如 MIPI 接口等外设时，可用作通用接口。

### 4.3.1. VPP 已有功能优化

- 1) `linebuf` 空间优化，相对于 80x 的固定 32 行 `linebuf`，81x 的 `linebuf` 数量可配置为 16 行

以上,最高 32 行,在效率影响不大的情况下,可尽量减少linebuf的配置,更有效地节省空间。

- 2) 裁剪功能去除, 80x是有输入裁剪功能,即摄像头配置为VGA,但DVP的接收分辨率可减少提取。81x此功能取消,并增加缩放功能到scale1 中, scale1 数据可直达jpeg,并不占用sram空间。scale1 的注意事项请查看scale章节。
- 3) RGB格式支持取消,目前使用上基本以yuv422 为主, 80x支持的RGB到jpg通路取消,只支持接收yuv422 与raw数据,并且只有yuv422 格式能到mjpeg。
- 4) 数据流输出源增加,相对于 80x的DVP只能输出到memory和mjpeg两个方向外, 81x可以输出到屏端, SCALE1 和SCALE3 端。scale1 为图像到mjpeg前的图像伸缩功能模块。scaler3 为图像到lcd前的图像伸缩功能模块。具体请查看scale章节。VPP只负责采集数据,通过输出端的使能来自动获取对应的数据。

#### 4.3.2. VPP 新增加功能

- 1) VPP在保留配置图像size的基础上,增加了双路VPP buf1 的可选模式, buf0 模式为常开状态, buf1 模式可选择打开关闭, buf1 相对buf0,增加了图像缩小功能,缩小为固定的 1/2 或者 1/3,如需使用qvga,建议使用此功能,不使用scaler进行缩小是因为大比例缩小,需要linebuf更多,在空间使用角度上考虑,并不值得。
- 2) 水印功能。水印功能增加,添加了两种模式的水印操作方式,一为画图模式,即直接定义水印的长宽,根据bit的情况进行显示,可用于图标上显示。二为列表模式,列表模式是在画图的基础上,增加了index寄存器,最大 16 个index,用于时间水印方面的操作,软件只需将对应的显示素材准备好,在VPP完成中断时去修改index的指向,即可方便地修改对应的时间水印。素材需 4byte对齐,并且存放到sram上,否则可能存在提取速度不够的可能性。
- 3) 相框功能。相对于水印的颜色单调,相框功能可让图像显示出色彩丰富的图框,并且数据经过简单的压缩,且增加了透明色功能,可用于儿童相机或者色彩要求高的图标需求。

其压缩的方式和透明色的标识，与屏端的osd压缩共用同一套系统，逻辑请查看“推屏类.ppt”中关于osd压缩功能的介绍。

- 4) 移动监测功能。移动监测功能是将图像划分为多个大小相等的区域，并设置一个监测的阈值，通过监测到是否有足够多的区域超过所设置的阈值，进行中断提醒。
- 5) 增加了拍照模式，为了方便大比例的图像放大，如VGA到 8K, 增加了拍照模式功能，此功能一旦打开，VPP会将图像数据从Linebuf自动拷贝到psram，让psram能自动组成一张yuv原图，后期再经过scale1 进行拉伸处理，有效地提高图像处理速度。

### 4.3.3. VPP 中断

VPP 中断在继承 80x 上的行中断，帧中断外，还增加了数据完成中断，拍照模式异常中断，拍照模式完成中断，相框异常中断，移动监测中断，水印异常中断。对应中断号请查看csi\_v2.c 相关代码。

### 4.3.4. api 接口说明

请查看对应文档“视频-摄像头.ppt”中关于 api 部分说明。

## 4.4. mjpeg 编码参数配置

```
200:
201: #define JPG0_HEAD_RESERVER 0//0 //每
202: #define JPG0_BUF_LEN 8192//8192//130
203: #define JPG0_NODE 80//20//18//30
204: #define JPG0_TAIL_RESERVER 0
205:
206:
207: //////////////////////////////////////
208: #define JPG1_HEAD_RESERVER 0 //每段
209: #define JPG1_BUF_LEN 1308//2880//40
210: #define JPG1_NODE 40//18//30*2
211: #define JPG1_TAIL_RESERVER 0
212:
```

TXW81x 总共有两路 mjpeg，mjpeg0/mjpeg1。两路 mjpeg 均可编码，数据源可来自vpp0,vpp1,prc,scale1。mjpeg1 有解码功能，解码后经过 scale2 伸缩后，数据可到 memory，用于推屏。

mjpeg 的总 size 配置为  $\text{JPG\_NODE} * (\text{JPG\_BUF\_LEN} + \text{JPG\_TAIL\_RESERVER})$ , 宏定义参考 project\_config.h 文件, JPG\_BUF\_LEN 不能过小, 这个是决定 mjpeg 的中断频率, JPG\_BUF\_LEN 越大(根据剩余内存空间配置), 一张完整图的中断越少, 效率以及出错情况越少, JPG\_NODE 是节点数, mjpeg 中断会从 JPG\_NODE 的节点池获取节点, 如果节点不够, 那么当前编码的图片会由于内存不足丢掉, JPG\_TAIL\_RESERVER 是用于自定义参数, 如果不需要, 设置为 0, JPG\_HEAD\_RESERVER 是头保留, 也是自定义, 不需要可以设置为 0, 在官方 sdk 中是默认 0, 会在 sdk 本身图传协议上使用, 空间由可存放在 sram 或者 psram, 当前均采用空间申请的方式进行处理。

```
void jpg_room_init(uint8 jpgnum){
    //INIT_LIST_HEAD(&jpg_node);
    uint8 *buf;

    if(jpgnum == 0)
    {
        if(global_jpg0_buf)
        {
            os_printf("0please custom_free jpg_buf when custom_malloc again\n");
            return ;
        }

        #ifdef PSRAM_HEAP
        buf = (uint8*)custom_malloc_psram(JPG0_NODE*(JPG0_BUF_LEN+JPG0_TAIL_RESERVER));
        #else
        buf = (uint8*)custom_malloc(JPG0_NODE*(JPG0_BUF_LEN+JPG0_TAIL_RESERVER));
        #endif
        global_jpg0_buf = buf;
        //sys_dcache_clean_invalid_range((void *)buf, JPG0_NODE*(JPG0_BUF_LEN+JPG0_TAIL_RESERVER));
    }
    else
    {
        if(global_jpg1_buf)
        {
            os_printf("1please custom_free jpg_buf when custom_malloc again\n");
            return ;
        }

        #ifdef PSRAM_HEAP
        buf = (uint8*)custom_malloc_psram(JPG1_NODE*(JPG1_BUF_LEN+JPG1_TAIL_RESERVER));
        #else
        buf = (uint8*)custom_malloc(JPG1_NODE*(JPG1_BUF_LEN+JPG1_TAIL_RESERVER));
        #endif
        global_jpg1_buf = buf;
        //sys_dcache_clean_invalid_range((void *)buf, JPG1_NODE*(JPG1_BUF_LEN+JPG1_TAIL_RESERVER));
    }
}
```

图片的压缩情况, 请参考函数 jpg\_quality\_tidy, 此函数在中断中调用, 用于通过当前图片的大小来控制图像质量的调整方向, 可理解为此函数为 jpg 图片大小的控制范围。但此函数只是期望值, 并不是一定能控制到距离实际太大的范围, 如将 720P 的图片压缩大小控制在 20K 以内, 这种是无法压缩出来这种大小的, 压缩的大小与当前使用的量化表与微调值相关, 量化表请参考 quality\_tab, 目前提供了 6 张量化表, 客户有需要, 可自行修改。微调是指在此量化表外, 还能进行依据当前使用量化表进行数据 16 个等级的微调, 等级 8 为量化表不变, 1~7 质量向上优化, 9~F 质量向下调整。

请查看对应文档“视频-摄像头.ppt”中关于 mjpeg api 部分说明。

## 4.5. DVP 图片的获取流程

DVP 图片获取接口如图:(jpgdef.h), 代码可以查看 AT\_save\_photo.c (拍照) 和 AT\_save.avi.c (录制视频), 这个流程是数据流的流程, 将大部分硬件配置都配置好(video\_app.c), 如果采用 sdk 默认的配置, 可以直接通过这个流程获取到对应图片。

```
#define GET_FRAME(f)          recv_real_data(f)
#define GET_DATA_LEN(f)      (uint32_t) get_stream_real_data_len(f)
#define GET_DATA_TIMESTAMP(f) (uint32_t) get_stream_data_timestamp(f)
#define GET_NODE_LEN(f)      (uint32_t) stream_data_custom_cmd_func(f,CUSTOM_GET_NODE_LEN,NULL)
#define FREE_JPG_NODE(f)     (uint32_t) stream_data_custom_cmd_func(f,CUSTOM_FREE_NODE,NULL)
#define GET_JPG_BUF(f)       (uint32_t) stream_data_custom_cmd_func(f,CUSTOM_GET_FIRST_BUF,NULL)
#define DEL_JPG_FRAME(f)     free_data(f)
#define DEL_JPEG_NODE(f,d)   (uint32_t) stream_data_custom_cmd_func(f,CUSTOM_DEL_NODE,d)
#define GET_DATA_BUF(f)      (uint32_t) get_stream_real_data(f)
#define DEL_FRAME(f)         free_data(f)
#define GET_JPG_SELF_BUF(f,d) (uint32_t) stream_data_custom_cmd_func(f,CUSTOM_GET_NODE_BUF,d)
#define GET_NODE_COUNT(f)    (uint32_t) stream_data_custom_cmd_func(f,CUSTOM_GET_NODE_COUNT,NULL)
```

以上截图的宏只是为了兼容旧版本的宏名称接口, 实际后续使用 stream\_frame.c 的函数就可以了, 以下针对函数名去解释接口, 并且结合代码分析:

- 1) 首先获取图片帧get\_f=recv\_real\_data(stream\*s)获取的是struct data\_structure结构的get\_f, 里面包含图片的数据、长度以及时间戳参数。
- 2) void \*data\_priv = get\_stream\_real\_data(get\_f), 这里获取的是实际内容, 如果是图片, 这里就是图片的所有数据(因为内存原因, 这里图片以节点形式保存, 比较特殊, 如果是音频, 则就是整个音频buf的起始地址)。
- 3) len = get\_stream\_real\_data\_len(get\_f), 获取get\_f的长度, 如果是图片, 就是图片长度, 如果是音频, 就是音频buf的长度。
- 4) node\_len = stream\_data\_custom\_cmd\_func(get\_f, CUSTOM\_GET\_NODE\_LEN, NULL), 整个是获取图片一个节点的大小(这个命令是获取图片特有), 因为图片以链表形式保存, 所以要知道节点最大的大小。
- 5) buf = stream\_data\_custom\_cmd\_func(f, CUSTOM\_GET\_NODE\_BUF, d), 这个获取的是节点的buf, 就是图片实际内容, 遍历的时候, 就是将所有buf读取出来就是图片, 注意这的f不是struct data\_structure的结构, 这个是 jpeg 的产生图片特定的结构体 struct

stream\_jpeg\_data\_s中的data(一个私有结构),最后获取到buf,然后通过node\_len读取图片内容。

```
struct stream_jpeg_data_s
{
    struct stream_jpeg_data_s *next;
    void *data;
    int ref;
};
```

- 6) stream\_data\_custom\_cmd\_func(get\_f, CUSTOM\_DEL\_NODE, d), 注意 d 是一个结构体 struct stream\_jpeg\_data\_s, 这个结合代码看才行。
- 7) free\_data(get\_f) 就是删除 get\_f, 将空间还原。

下图是一个伪代码, 显示使用流程, 细节处理可以参考 AT\_save\_photo.c 文件的拍照功能:

```

stream *s = NULL;
struct data_structure *get_f = NULL;
void *fp = NULL;
//创建接收流
s = open_stream_available(R_AT_SAVE_PHOTO,0,8,opcode_func,NULL);
//启动jpeg(内部就是创建一个发送流,会绑定R_AT_SAVE_PHOTO)
start_jpeg();
//获取图片
get_f = recv_real_data(s);
//创建文件
fp = osal_fopen(filename,"wb+");
{
    uint8_t *jpeg_buf_addr = NULL;
    uint32_t total_len;
    uint32_t uint_len;
    uint32_t jpg_len;
    struct stream_jpeg_data_s *el,*tmp;
    //获取图片数据的节点
    struct stream_jpeg_data_s *dest_list = (struct stream_jpeg_data_s *)get_stream_real_data(get_f);
    struct stream_jpeg_data_s *dest_list_tmp = dest_list;
    //获取图片长度
    total_len = get_stream_real_data_len(get_f);
    //获取图片数据一个节点的长度
    uint_len = (uint32_t)stream_data_custom_cmd_func(get_f,CUSTOM_GET_NODE_LEN,NULL)
    jpg_len = total_len;

    //遍历图片的节点
    LL_FOREACH_SAFE(dest_list,el,tmp)
    {
        if(dest_list_tmp == el)
        {
            continue;
        }
        if(fp && total_len)
        {
            //获取节点的buf起始地址
            jpeg_buf_addr = (uint8_t *)stream_data_custom_cmd_func(get_f,CUSTOM_GET_NODE_BUF,el->data);
            //写卡
            if(total_len >= uint_len)
            {
                osal_fwrite(jpeg_buf_addr,1,uint_len,fp);
                total_len -= uint_len;
            }
            else
            {
                osal_fwrite(jpeg_buf_addr,1,total_len,fp);
                total_len = 0;
            }
        }
        //使用完一个节点,就要删除改节点,快速释放空间
        stream_data_custom_cmd_func(get_f,CUSTOM_DEL_NODE,el)
    }
}
//写卡完毕后,释放get_f的节点
free_data(get_f);
//关闭文件系统句柄
osal_fclose(fp);

```



## 4.6. DVP 数据流说明

DVP → JPG 数据流的生成步骤与基本思想如下：

- 1) JPG数据buf以节点的形式分配好
- 2) DVP采集足够的行数据
- 3) JPG提取压缩
- 4) 当JPG压缩出来数据量到达所配置的数据buf长度或者图片获取结束，生成out\_buf\_full或者done中断（详情请看jpg模块api函数说明）
- 5) 中断里记录好节点的情况并分配下次jpg数据所存放的节点位置
- 6) 若done中断完成，重组jpg图片

其中，jpg 数据 buf 节点分配，请查看 jpg\_room\_init 函数，这里特说明，frame 根节点 2 个，即 jpg\_frame[2]，分别被 jpg\_p（中断）与 usr\_p（应用）指针指向，用于表示对应 frame 的使用是在中断还是在应用。除了 jpg\_frame 的两个根节点外，还有 free\_tab 一个根节点，用于管理由 free\_table\_init 中生成的众多空间子节点。每个空间子节点都会绑定自己专属的空间，图片数据存放则存放到此众多空间中，整帧图片数据并不连续，但能根据 jpg\_frame 的应用节点查找到图片数据排布并进行数据重组。

## 4.7. DVP 重要打印说明

- 打印信息“sip reset DVP”，DVP采集出错，请检查摄像头是否插好，或者配置表中信息是否与IMAGE\_W、IMAGE\_H一致，如果IMAGE\_W与IMAGE\_H配置为720P，配置表配置为VGA，则会出现此打印，因为采集数据上出问题。还有一种情况，配置表中的hsync和vsync的有效电平配置出错，即配置表中hsync和vsync的有效电平与\_Sensor\_Adpt\_里面的配置不匹配，这样也会造成采样出错的情况。
- 打印信息“fh”和“fv”，DVP处理速度跟不上采集速度，“fh”表示在数据没处理完的情况下，上次采集的buf已改写了一半。“fv”表示在数据没处理完的情况下，上次采集的buf已被改写。出现这种问题有可能是DVP的CLK分配过高，CPU处理不过来，或者

DVP中断中处理的事情过多，影响到处理效率。

- 打印“jpg done len err”，这种情况有两种可能性，一是CPU处理不过来，即JPG的压缩率不高，图像太大，CPU在多次中断才响应一次。第二种情况是，软件buf空间不足，节点不够（参考数据流说明）。针对第一种情况，处理方式为提高代码的运行速率（xip的运行速度, 请查看《TXW81x TXProgrammer工具使用文档》）、提升节点的空间大小或者降低图像的质量。第二种情况处理方式为提升代码运行速率，降低图像质量或者在空间允许的情况下，增加节点的数量。
- 打印“？”，这种情况为JPG出现异常了，代码会复位DVP和JPG，如持续打印这个，请检查是否图片空间过小，但图像生成太大，无法满足整张图片的存放。

## 5. SPI 摄像头

### 5.1. SPI 摄像头配置

- 1) SPI摄像头配置表跟DVP摄像头一样，配置流程也跟DVP相似，但SPI摄像头没有像DVP一样的mclk功能引脚，所以会使用PWM充当摄像头的时钟源xclk。
- 2) cs引脚与vsync引脚，此两引脚无需接上，代码硬件是通过vsync与hsync的时钟间隔（检查clk停止时间），来进行帧同步与行同步。
- 3) 摄像头的支持，我们是充当SPI从机，spi\_clk要求是在有数据的情况下才出spi\_clk，不支持spi\_clk持续输出的配置方式。
- 4) vsync与hsync支持头数据裁剪功能，用于排除非图像数据，头数据的多少请在配置好摄像头的情况下，提取摄像头输出的信号，根据信号情况，配置头裁减大小。
- 5) 内容接收，采用乒乓buf机制，即同时配置两个接收buf，满一buf数据后起一次中断。
- 6) SPI模块支持SPI摄像头均采用ioctl进行配置，配置功能如下，代码请查看

spi\_sensor.c:

```
.95:
.96:     /*
.97:     *  spi sensor
.98:     */
.99:     SPI_SENSOR_VSYNC_TIMEOUT,
:00:
:01:     SPI_SENSOR_VSYNC_HEAD_CNT,
:02:
:03:     SPI_SENSOR_VSYNC_EN,
:04:
:05:     SPI_SENSOR_HSYNC_TIMEOUT,
:06:
:07:     SPI_SENSOR_HSYNC_HEAD_CNT,
:08:
:09:     SPI_SENSOR_BUF_LEN,
:10:
:11:     SPI_SENSOR_SET_ADR0,
:12:
:13:     SPI_SENSOR_SET_ADR1,
:14:
:15:     SPI_SENSOR_PINGPANG_EN,
:16:
:17:     SPI_HIGH_SPEED_CFG,
:18:
:19:     SPI_KICK_DMA_EN,
:20:
:21:     SPI_RX_TIMEOUT_CFG,
:22: } « end spi_ioctl_cmd » ;
:23:
```

```

5:
6: void spi_sensor_cfg(){
7:     spi0_dev_g = (struct spi_device*)dev_get(HG_SPI1_DEVID);
8:     spi_open(spi0_dev_g, 1000000, SPI_SLAVE_MODE, SPI_WIRE_SINGLE_MODE, SPI_CPOL_0_CPHA_0);
9:     spi_iocctl(spi0_dev_g, SPI_CFG_SET_NONE_CS, 1, 0);
10:    spi_iocctl(spi0_dev_g, SPI_SENSOR_VSYNC_TIMEOUT, 0xea60, 0);
11:    spi_iocctl(spi0_dev_g, SPI_SENSOR_VSYNC_HEAD_CNT, 0, 0);
12:    spi_iocctl(spi0_dev_g, SPI_SENSOR_VSYNC_EN, 1, 0);
13:    spi_iocctl(spi0_dev_g, SPI_SENSOR_HSYNC_TIMEOUT, 2400, 0);
14:    spi_iocctl(spi0_dev_g, SPI_SENSOR_HSYNC_HEAD_CNT, 6, 0);
15:    spi_iocctl(spi0_dev_g, SPI_HIGH_SPEED_CFG, 0, 0);
16:    spi_iocctl(spi0_dev_g, SPI_SENSOR_PINGPANG_EN, 1, 0);
17:    spi_iocctl(spi0_dev_g, SPI_SENSOR_BUF_LEN, 480*16, 0);
18:    if(spi_dat_buf[0] == NULL){
19:        spi_dat_buf[0] = (uint8 *)custom_malloc(480*16);
20:    }
21:
22:    if(spi_dat_buf[1] == NULL){
23:        spi_dat_buf[1] = (uint8 *)custom_malloc(480*16);
24:    }
25:
26:    spi_iocctl(spi0_dev_g, SPI_SENSOR_SET_ADR0, (uint32)spi_dat_buf[0], 0);
27:    spi_iocctl(spi0_dev_g, SPI_SENSOR_SET_ADR1, (uint32)spi_dat_buf[1], 0);
28:    spi_iocctl(spi0_dev_g, SPI_RX_TIMEOUT_CFG, 1, 180000);
29:    spi_request_irq(spi0_dev_g, SPI_IRQ_FLAG_RX_DONE | SPI_IRQ_FLAG_RX_TIMEOUT, spi0_read_irq, (uint32)spi0_dev_g);
30:    spi_iocctl(spi0_dev_g, SPI_KICK_DMA_EN, 0, 0);
31: } « end spi_sensor_cfg »
32:

```

- 7) 在读取数据的配置中, 请注意SPI\_WIRE\_SINGLE\_MODE的模式配置, 这配置与摄像头输出情况相关, 如若配置错, 会出现数据错位, 图像异常的情况。

```

67: enum spi_clk_mode {
68:     /*! Compatible version: V0; V1; V2
69:     * @Describe:
70:     * Inactive state of serial clock is low, serial clock toggles in middle of first data bit
71:     */
72:     SPI_CPOL_0_CPHA_0 = 0,
73:
74:     /*! Compatible version: V0; V1; V2
75:     * @Describe:
76:     * Inactive state of serial clock is low, serial clock toggles at start of first data bit
77:     */
78:     SPI_CPOL_0_CPHA_1 = 1,
79:
80:     /*! Compatible version: V0; V1; V2
81:     * @Describe:
82:     * Inactive state of serial clock is high, serial clock toggles in middle of first data bit
83:     */
84:     SPI_CPOL_1_CPHA_0 = 2,
85:
86:     /*! Compatible version: V0; V1; V2
87:     * @Describe:
88:     * Inactive state of serial clock is high, serial clock toggles at start of first data bit
89:     */
90:     SPI_CPOL_1_CPHA_1 = 3,
91:
92:     SPI_CLK_MODE_0 = SPI_CPOL_0_CPHA_0, /*!< Equal to \ref SPI_CPOL_0_CPHA_0 */
93:     SPI_CLK_MODE_1 = SPI_CPOL_0_CPHA_1, /*!< Equal to \ref SPI_CPOL_0_CPHA_1 */
94:     SPI_CLK_MODE_2 = SPI_CPOL_1_CPHA_0, /*!< Equal to \ref SPI_CPOL_1_CPHA_0 */
95:     SPI_CLK_MODE_3 = SPI_CPOL_1_CPHA_1 /*!< Equal to \ref SPI_CPOL_1_CPHA_1 */
96: } « end spi_clk_mode » ;
97:

```

## 5.2. PRC 功能联合 SPI 摄像头

上一小节中提到, SPI 摄像头是经过乒乓 BUF 机制提取数据, 每提取满一 buf 数据, 则会响应一次中断, 此时只需我们开启 PRC 功能, 则能将 SPI 摄像头输入的 yuv 数据格式 yuyvyuyv 重新排列为 yyyyuuvv。这样重排的意义有两个, 一为可很高效地提取 y 数据到同一个位置, 方便我们去做一些图像运算, 如定高, 二维码, 条型码等。二为可让数据能以统一的格式输入到 jpg, 这样子则可使 mjpeg 很好地提取 PRC 的数据, 并进行 SPI 摄像头的图

像压缩。下图代码为将 prc 配置成跟 SPI 摄像头单 buf 数据一样的行数据，即 16 行的配置，在 SPI 接收完后再启动 prc 数据重排序。

```
void prc_module_init(){
    prc_dev_g = (struct prc_device*)dev_get(HG_PRC_DEVID);
    prc_init(prc_dev_g);
    prc_set_width(prc_dev_g,240);
    prc_set_yuv_mode(prc_dev_g,0);

    prc_set_yaddr(prc_dev_g,(uint32)prc_yuv_line);
    prc_set_uaddr(prc_dev_g,(uint32)prc_yuv_line+240*16);
    prc_set_vaddr(prc_dev_g,(uint32)prc_yuv_line+240*16+240*16/4);

    prc_request_irq(prc_dev_g,PRC_ISR,(prc_irq_hdl)prc_deal_irq,(uint32)prc_dev_g);
}

2: void spi0_read_irq(uint32 irq, uint32 irq_data){
3:
4:     if(irq == SPI_IRQ_FLAG_RX_DONE){
5:         prc_new_frame = 0;
6:         if(sync_start){
7:             if((hnum%2) == 0){
8:                 prc_set_src_addr(prc_dev_g,(uint32)spi_dat_buf[0]);
9:             }else{
10:                 prc_set_src_addr(prc_dev_g,(uint32)spi_dat_buf[1]);
11:             }
12:             prc_run(prc_dev_g);
13:         }
14:         hnum++;
15:     }
16:
17:     if(irq == SPI_IRQ_FLAG_RX_TIMEOUT){
18:         prc_new_frame = 1;
19:         prc_deal_num = 0;
20:         sync_start++;
21:         flow_data_cnt++;
22:         spi_close(spi0_dev_g);
23:         spi_sensor_cfg();
24:         hnum = 0;
25:     }
26: } « end spi0_read_irq »
```

注：

- 1) PRC跟VPP一样，只需自身启动即可，其他输出模块如需提取PRC作为输入源，则修改输出模块的输入源来源即可，无需配置PRC的输出方向。
- 2) PRC与jpeg的同步问题，因为PRC只是数据重排，无法知晓当时是否为首行，mjpeg在启动之前，PRC如果正在提取非首行的数据，则会出现图像错行的情况，请做好PRC与mjpeg的同步问题，即在启动mjpeg时，明确当前PRC是正处理帧间隔的时间下，这样同步才能保证图像无异常。

## 6. SCALE 模块

SCALE 模块分三种，分为 scale1, scale2, scale3。功能说明如下。

### 6.1. scale1

scale1 是用于将 VPP 图像数据进行伸缩功能，并传输到 mjpeg 中进行图像压缩。放大接口使用：

scale\_from\_vpp\_to\_jpg：在放大 2K 以下，均可用此接口进行放大缩小，并同步修改 photo\_msg.out0\_h 与 photo\_msg.out0\_w 的 scale 输出 size, 让 mjpeg 配置成所需的 size。

```
11:
12: void jpg_start(uint8 jpg_num){
13:     struct jpeg_device *jpeg_dev;
14:     uint32_t jpgid;
15:     jpgid = jpg_num + HG_JPG0_DEVID;
16:     jpeg_dev = (struct jpeg_device *)dev_get(jpgid);
17:
18:     //jpg csr config
19:     if(jpgid == HG_JPG0_DEVID){
20:         jpg_set_size(jpeg_dev, photo_msg.out0_h, photo_msg.out0_w);
21:     }
22:     else{
23:         jpg_set_size(jpeg_dev, photo_msg.out0_h, photo_msg.out0_w);
24:     }
25:     jpg_open(jpeg_dev);
26: }
27:
```

同步需要将 jpeg 的输入源修改为 scale。

```
11 // SCALER
12 jpg_cfg(HG_JPG0_DEVID, SCALER_DATA/*PRC_DATA*/);
13 endif

enum JPG_SRC_FROM {
    VPP_DATA0,
    VPP_DATA1,
    PRC_DATA,
    SCALER_DATA,
    SOFT_DATA,
};
```

注：缩小倍数如果过大，则需要增加 VPP 所需要的 Linebuf，否则 scale1 会报错。所以在缩小倍数过大时，可使用 VPP 的 buf1 功能，可缩小 1/2 和 1/3。

在放大超过 2K 的情况下，使用 VPP 的拍照模式，VPP 的拍照模式会将图像以 yuv420 的格式存放到 psram 中，scale1 再调用接口 scale\_from\_soft\_to\_jpg 则可以将图像拉伸到最大 8K 的 size。

## 6.2. scale2

scale2 是用于 mjpeg 解码后屏显的模块，此模块是 mjpeg 解码后，直接经过 scale2 伸缩后以 yuv420 的格式存放到 memory，屏此时即可使用 video 输出到屏上，达到显示的目的。

jpg\_decode\_scale\_config: scale2 显示初始化

jpg\_decode\_to\_lcd : 需解码的图像地址，解码前的分辨率与伸缩后的分辨率

## 6.3. scale3

scale3 是用于 VPP 进行图像伸缩后进行推屏显示。scale3 输入源可分为 VPP 与缓存输入模式，输出源分为直接到屏端或者到 memory (sram/psram)。

初始化配置到屏端的，请查看 scale\_to\_lcd\_config，用宏 SCALE\_DIRECT\_TO\_LCD 来决定是否直接输出到屏还是输出到 memory，这个初始化配置为输入源是 VPP 的情况。

输入源为缓存，即将原有的 yuv420 数据拉伸到不同分辨率的 yuv420 分辨率，使用接口 scale\_to\_lcd\_config\_soft 进行初始化，目前是配置为拉伸为满屏的操作。

## 7. LCDC 屏显说明

屏数据源来源于 video 与 osd, video 有两路, 由 photo0 与 photo1 组成, 可独立配置。

osd 有一层, 格式支持 256 色, rgb565, rgb888。

### 7.1. osd 推屏

请查看 lvgl\_run 代码中

```
if(disp_updata == 1){  
  
    . . . . .  
  
}  
  
disp_updata = 0;
```

此部分如不使用 lvgl, 可进行移植, osd 空间为 osd\_menu565\_buf, 代码会对此空间进行压缩, 以乒乓 buf 的形式传送到 osd\_encode\_buf 与 osd\_encode\_buf1, 再由屏端通过此两 buf 进行刷屏。客户无需关注压缩与刷屏逻辑, 只需针对 osd\_menu565\_buf 进行修改即可。

注: osd 存在压缩功能与透明色功能, 会从色表中取出两个色彩进行功能确定, 函数为:

```
int32 lcdc_set_osd_enc_head(struct lcdc_device *p_lcdc, uint32 head, uint32  
head_tran)
```

head : 压缩头数据, 配置后会对此数据当成压缩数据的起始数据

head\_tran : 源数据中如存在压缩头, 则由 head 转换成 head\_tran

如配置压缩头为 0xffff, 则

源色彩数据: 8080 8080 8080 8080 8080 8080 8080 8080

压缩后数据: ffff 0008 8080

```
int32 lcdc_set_osd_enc_diap(struct lcdc_device *p_lcdc, uint32 diap, uint32
```



diap\_tran)

diap :透明色数据

diap\_tran :遇到透明色后转换成其他数据，如需透明色，则 diap\_tran 跟 diap 一样即可。

## 7.2. video 推屏

video 由 photo0\photo1 组成，可分别配置对应的 video 显示的位置，亦可选择是否使用对应的 photo，目前 photo0 用于通用 DVP 推送 video 显示，photo1 用于解码后 video 显示。

photo0\photo1 可配置任一放在高优先级

video 可由 scale3 直接推送到屏，但此模式下无法旋转，并且帧率需与 DVP 相同。常用做法为直接读取 memory 后推屏，memory 数据为 yuv420 格式。

## 7.3. jpeg 推屏

jpg 图像作为常用的图像格式，用作推屏显示接口，请使用

```
void jpg_decode_to_lcd(uint32 photo,uint32 jpg_w,uint32 jpg_h,uint32
video_w,uint32 video_h)
```

Photo :jpg 图片源数据

jpg\_w、jpg\_h :jpg 图片的长宽

video\_w、video\_h :解码 video 大小

注：此接口需提前准备好解码数据存放空间，代码中已使用 video\_decode\_mem，video\_decode\_mem1，video\_decode\_mem2 三指针做好空间处理，只需提前做好指针空间预指向即可，可参考代码 void lv\_page\_select(uint8\_t page)中 page == 0 与 page == 1 代码

## 7.4. 源数据推屏

如已准备好 yuv420 源数据,可直接通过接口 `void lcd_user_frame(uint32 frame_addr)` 进行推屏,此函数内部只做 `video_decode_mem`, `video_decode_mem1`, `video_decode_mem2` 三指针做好空间处理。

## 7.5. 屏配置

```
typedef struct _lcd_desc_s{  
  
    . . . .  
  
}lcddev_t;
```

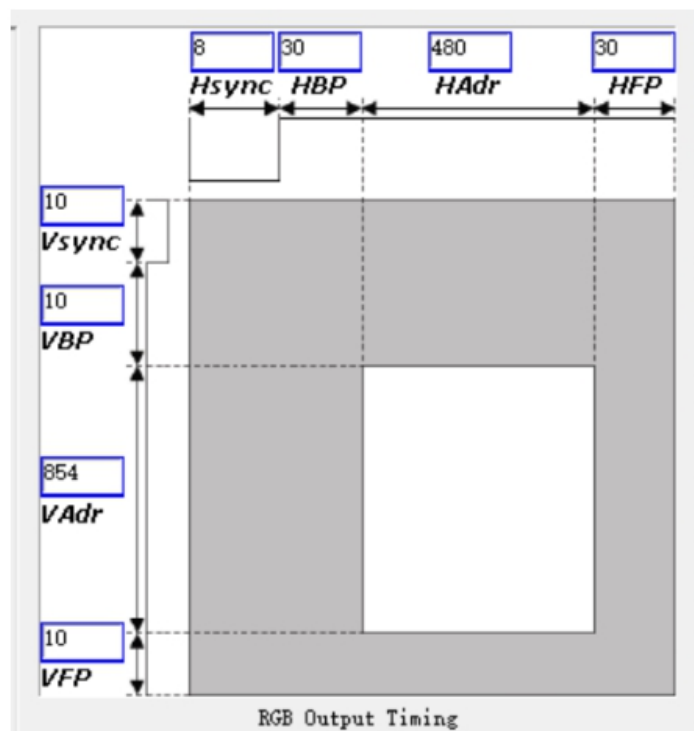
例:

```
36:  
37: lcddev_t lcdstruct = {  
38:     .name = "st7701s",  
39:     .lcd_bus_type = LCD_BUS_RGB,  
40:     .bus_width = LCD_BUS_WIDTH_16,  
41:     .color_mode = LCD_MODE_565,  
42:     .scan_mode = LCD_ROTATE_90,//rotate 90  
43:     .te_mode = 0xff,//te mode, 0xff:disable  
44:     .colrarray = 0,//0:_RGB_ 1:_RBG_ 2:_GBR_ 3:_GRB_ 4:_BRG_ 5:_BGR_  
45:     .pclk = 20000000,  
46:     .even_order = 0,  
47:     .odd_order = 0,  
48:  
49:     .screen_w = 480,  
50:     .screen_h = 854,  
51:     .video_x = 0,  
52:     .video_y = 0,  
53:     .video_w = 480,  
54:     .video_h = 854,  
55:     .osd_x = 0,  
56:     .osd_y = 0,  
57:     .osd_w = 480, // 0 : value will set to video_w , use for 4:3 LCD +16  
58:     .osd_h = 848, // 0 : value will set to video_h , use for 4:3 LCD +16  
59:     .init_table = st7701s_register_init_tab,  
60:  
61:     .clk_per_pixel = 2,  
62:  
63:     .pclk_inv = 1,  
64:  
65:     .vlw      = 10,  
66:     .vbp      = 6,  
67:     .vfp      = 10,  
68:  
69:     .hlw      = 8,  
70:     .hbp      = 8,  
71:     .hfp      = 30,  
72:  
73:  
74:     .de_inv = 0,  
75:     .hs_inv = 1,  
76:     .vs_inv = 1,  
77:  
78:  
79:     .de_en  = 1,  
80:     .vs_en  = 1,  
81:     .hs_en  = 1,  
82:  
83: };  
84:
```

#### 元素说明:

name	:屏名字
lcd_bus_type	:屏接口 (rgb, 8080, 6800)
color_mode	:屏输出模式 (rgb565, rgb666, rgb888)
scan_mode	:屏旋转
te_mode	:mcu 屏是否存在 te IO
colrarray	:输出色彩顺序
pclk	:dotclk 频率, 从主频 240M 分频
screen_w、screen_h	:屏宽高
video_w、video_h	:video 数据源长宽
video_x、video_y	:video 的开始坐标
osd_w、osd_h	:osd 数据源长宽
osd_x、osd_y	:osd 的开始坐标
init_table	:屏配置表
pclk_inv	:dotclk 反向
de_en	:DE io 功能使能
vs_en	:vsync io 功能使能
hs_en	:hsync io 功能使能
de_inv	:de io 反向
vs_inv	:vsync io 反向
hs_inv	:hsync io 反向

vlw、vbp、vfp、hlw、hbp、hfp 配置需读取屏配置表, 相关的无效行, 无效列, 有效区域与可视区域。



## 7.6. 旋转问题

屏显模块，只有 video 有旋转功能，osd 无旋转功能。旋转角度为 0, 90, 180, 270 四种，而 180 度的旋转需要注意空间分配，YUV 数据偏移跟 0, 90, 270 有所不同，此外用宏 LCD\_SET\_ROTATE\_180 进行控制。在 video 显示中，如果不需要旋转，可以配置屏端 ioctl1 LCDC\_SET\_VIDEO\_MODE。不旋转，无需申请旋转所需的 linebuf 空间，lcdc\_set\_rotate\_linebuf\_XXX\_addr, 此空间跟屏宽与所需性能相关，屏越宽，需求性能越好，则这个空间需要更大，LCD\_ROTATE\_LINE 为控制此部分的宏。

## 7.7. 屏驱动 api

请查看文档“推屏类.ppt”，结合代码了解模块中存在的功能及其 api 的使用。

## 8. USB 摄像头

### 8.1. USB 摄像头参数配置

宏定义:UVC\_BLANK\_LEN 是节点的大小, UVC\_BLANK\_NUM 节点数量, 在无 psram 的情况下, UVC\_BLANK\_NUM 需要足够大, UVC\_BLANK\_NUM\*UVC\_BLANK\_LEN 起码需求 1.5 张图片的 size 大小, 否则图片会被丢弃, 在有 psram 的情况下, UVC\_BLANK\_NUM 可以小一点, 最后所有图片数据都会放在 psram 中, UVC\_BLANK\_LEN 不宜过小, 默认 12\*1024 byte, UVC\_FRAME\_NUM 代表图片的帧, 默认值为 2。

```
:
: #if USB_EN
: extern uint8_t uvc_dat[1024];
: #define UVC_BLANK_LEN      12*1024
: #if UVC_PSRAM == 1
: #define UVC_BLANK_NUM      3
: #else
: #define UVC_BLANK_NUM      7
: #endif
: #define UVC_FRAME_NUM      2
: UVC_BLANK uvc_blank[UVC_BLANK_NUM];
: uint8 blank_space[UVC_BLANK_NUM][UVC_BLANK_LEN+UVC_HEAD_RESERVER]__attribute__((aligned(4)));
```

在分辨率选择上, 默认分辨率请修改 uvc\_default\_dpi 里的返回值, 当前默认有两种 UVC\_VGA 和 UVC\_720P。在传输中途若想修改摄像头的分辨率, 请使用 usb 的 ioctl 功能, 例如:

```
: void uvc_ioctl_index(uint8 uvc_idx){
:     uint8 msgbuf[1];
:     uint8* pt;
:
:     struct usb_device *dev;
:     dev = (struct usb_device *)dev_get(HG_USBDEV_DEVID);
:     pt = msgbuf;
:     pt[0] = uvc_idx;
:     usb_host_uvc_ioctl(dev, USB_HOST_IO_CMD_SET_IDX, msgbuf, 0);
: }
```

如果摄像头有接入 hub, 也同样可用 IOCTL 配置, USB\_HOST\_IO\_CMD\_INSERT\_HUB 功能。

### 8.2. USB 摄像头获取过程

(实际宏定义与 DVP 摄像头的一样, 调用函数名称改成 uvc 的实现)

uvc 图片获取接口如图: (jpgdef.h)

```

extern int get_node_uvc_len();
#define UVC_HEAD_RESERVER 24 //每段JPEG BUF前面预留用于填充其他数据的数据量, 无需预留填0
#define UVC_PSRAM 0
#define GET_NODE_LEN(f) get_node_uvc_len(f)
#define GET_JPG_LEN(f) get_uvc_frame_len(f)
#define GET_JPG_buf(f) get_uvc_buf()
#define GET_JPG_FRAME() get_uvc_message_gloal()
#define FREE_JPG_NODE(f) free_uvc_blank_nopsram(f)
#define DEL_JPG_FRAME(f) uvc_sema_up()

```

- 1) 首先获取图片帧  $f = \text{GET\_JPG\_FRAME}()$ , 如果获取到代表已经获取到整一张图片, 图片的组成是以节点形式连接, 只要通过循环读取就能获取到整一张图。
- 2)  $\text{GET\_JPG\_LEN}(f)$  获取一张图片的size。
- 3)  $\text{GET\_JPG\_buf}(f)$  获取图片帧的首个节点。
- 4)  $\text{FREE\_JPG\_NODE}(f)$  释放图片帧首个节点。
- 5)  $\text{GET\_NODE\_LEN}(f)$  获取一个节点的长度(图片帧内所有节点长度都是一样的, 所以获取到图片帧最后一个节点的时候, 实际内容  $\leq \text{GET\_NODE\_LEN}(f)$ , 所以处理最后一帧的时候, 应该根据图片size计算最后节点的实际内容的大小)。
- 6)  $\text{DEL\_JPG\_FRAME}(f)$  删除图片帧, 同时归还所有节点, 在图片帧使用完毕后, 需要调用该接口去释放, 否则导致图片帧丢失, 后续获取不到图片帧。

伪代码:

```

int read_size = 0;
int photo_size_tmp, photo_size, node_len;
uint8 *buf;
//获取图片帧
void *f = GET_JPG_FRAME()
if(f)
{
    //获取节点长度
    node_len = GET_NODE_LEN(f);
    //获取图片的size
    photo_size_tmp = photo_size = GET_JPG_LEN(f);

    //循环读取图片帧的节点
    while(photo_size_tmp)
    {
        buf = GET_JPG_buf(f);
        //异常, 不能出现这种情况
        if(!buf)
        {
            printf("get buf err\n");
            while(1);
        }
        if(photo_size_tmp >= node_len)
        {
            read_size = node_len;
        }
        else
        {
            read_size = photo_size_tmp;
        }
        //保存图片的buf, 自定义实现, buf read_size

        FREE_JPG_NODE(f);
        photo_size_tmp -= read_size;
    }

    //图片帧使用完毕
    DEL_JPG_FRAME(f)
}

```

### 8.3. USB 重要打印说明

- 打印“babble error”，此打印说明信号线接触不良，系统会重新reset usb,要是持续打印babble error，则需要确定usb线是否正常。
- 打印“disconnect.....”，此打印为设备掉线打印，这种掉线有接触不良掉线或者主动排插掉线，系统会重新reset usb。
- 打印“XXX Err”和“Enum ERR”，此类打印为设备在枚举过程中枚举失败，系统会重新reset usb。
- 打印“uvc can not set resolution :720p”，此类打印为在不支持 720P的摄像头上配置了 720P输出，会默认改为VGA输出。
- 打印“\_D3\_”，此类打印为摄像头的UVC图像出错，如果这部分打印过多，请确定摄像头是否正常。
- 打印“\_D1\_”和“\_D2\_”时，表明当前的数据缓存空间不足，如果是跑无psram的方案下，持续打印这个的情况下，请确认图像是否大小超过默认size，7\*12K，如果是，请修改摄像头的输出大小。如果此打印非持续出现，但也高频率出现，表明cpu处理不过来，请提高代码的运行效率（XIP的运行速率，请查看《TXW81x TXProgrammer工具使用文档》）。