



## TXW80X 音频方案开发指南



### 注意

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，本文档仅作为使用指导，不做任何担保。


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	TXW80X SDK 快速入门手册	文件编号	TX-0000
发行日期	2022-08-08		文件版本	V1.1

修订记录

日期	版本	描 述	修订人
2022-08-08	V1.1	增加 I2S 开发指南	TX
2022-05-15	V1.0	初始版本	TX

 泰芯半导体 TaiXin Semiconductor	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 © 2021 by Tai Xin All rights reserved
--

保密等级	A	TXW80X SDK 快速入门手册	文件编号	TX-0000
发行日期	2022-08-08		文件版本	V1.1

目录

1. 概述.....

1

2. 硬件开发板.....

1

2.1. 音视频开发板.....

2

2.1.1. 音视频开发板接口介绍.....

2

3. 音频开发配置流程.....

3

3.1. 音频方案相关配置.....

3

3.2. PDM 麦克风.....

3

3.2.1. PDM 音频相关参数配置.....

3

3.2.2. PDM 音频接口说明.....

3

3.3. I2S(8211)播放音频.....


6

3.3.1. I2S(8211)相关参数配置.....

6

3.3.2. I2S(8211)接口说明.....

6

	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
---	--	-------------------------

## 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
TXW806	QFN56/48/40	

## 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

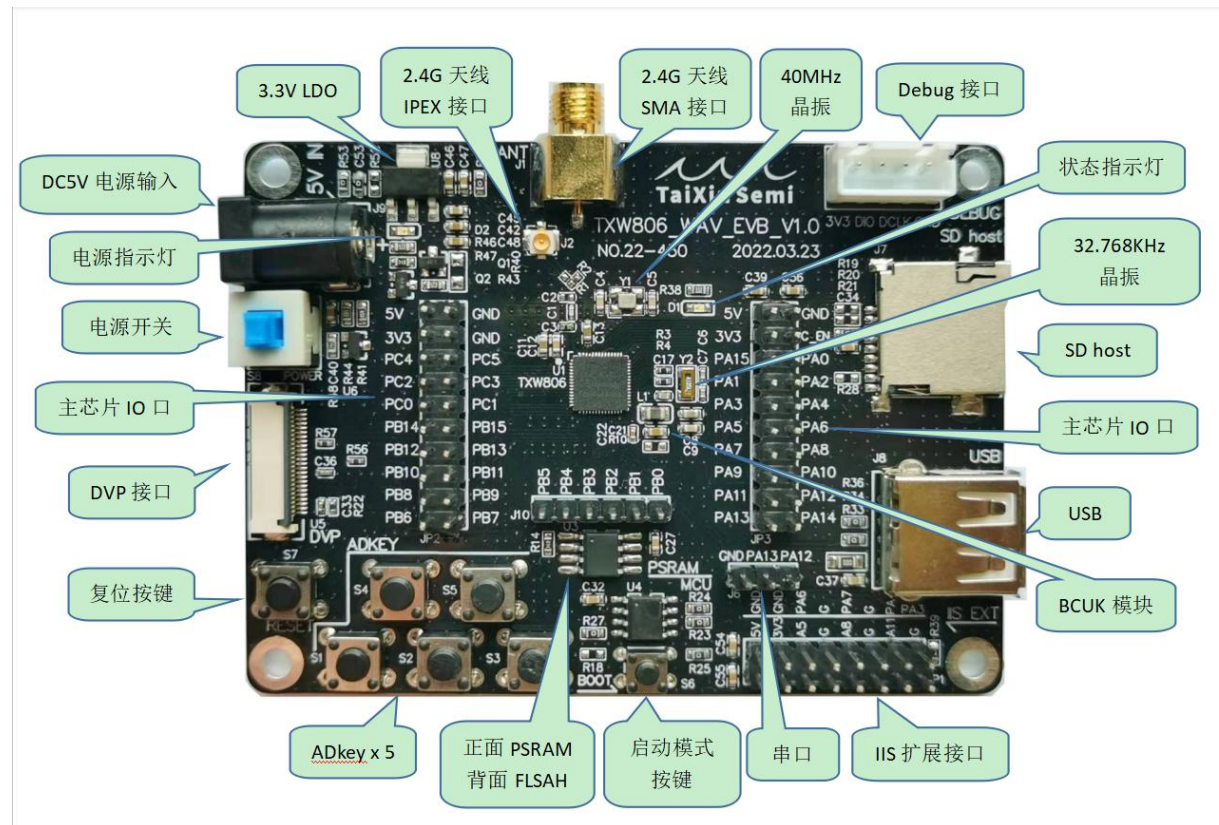


图 3.1.1.1 - 音视频开发板主视图

#### 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。

**IIS 扩展接口：**此接口注意是为了扩展音频子板。

## 3. 音频开发配置流程

### 3.1. 音频方案相关配置

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。

```
#define USB_HOST_EN          1
#define MACBUS_USB

#define USB_EN              0
#define DVP_EN              0
#define JPG_EN              (1 && DVP_EN)
#define SDH_EN              1
#define FS_EN               1
#define VCM_EN              (0 || DVP_EN)

#define FRAME_MANAGEMENT    1
#define OPENDML_EN          1
#define UART_INPUT_EN       1
#define UART_FLY_CTRL_EN    0
#define PWM_EN              0

#define AUDIO_EN             (0 && FRAME_MANAGEMENT)
#define IIS_AUDIO_EN        (0 && FRAME_MANAGEMENT)
#define RTP_SOUND           (1 && (AUDIO_EN || IIS_AUDIO_EN))

#define MJPEG_VIDEO          (1 && FRAME_MANAGEMENT && OPENDML_EN && FS_EN && SDH_EN && JPG_EN) //基于框架的mjpeg录像
#define UVC_VIDEO           (1 && FRAME_MANAGEMENT && OPENDML_EN && FS_EN && SDH_EN && USB_EN) //基于框架的uvc录像
```

- 1、USB\_EN 启动 USB 摄像头(usb 与 dvp 暂时不能共存)
- 2、DVP\_EN 启动 dvp 摄像头(usb 与 dvp 暂时不能共存)
- 3、SDH\_EN、FS\_EN 是 sd 卡与文件系统初始化
- 4、FRAME\_MANAGEMENT(仅仅支持 sdk 的图传协议) 框架启动, 在外挂 psram 的时候, 需要使能
- 5、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)

### 3.2. PDM 麦克风

#### 3.2.1. PDM 音频相关参数配置

在 sdk 中有自带音频的 demo, 主要是通过宏 #define AUDIO\_EN 和 #define FRAME\_MANAGEMENT 来打开(project\_config.h), 默认是适配 sdk, 可以图传和音频一起传输, 如果是需要自定义协议, 需要自行开发

#### 3.2.2. PDM 音频接口说明

在 sdk 中已经封装了 pdm 的音频接口, 需要注册对应接口, 只需要关注以下几个接口, 如图:

```

7:
8: void    pdm_audio_close(void *audio_hdl);
9: int     pdm_audio_start(void *audio_hdl);
0: void    pdm_audio_register(void *audio_hdl,void *priv_el,int play_size,set_buf set_buf,get_buf get_buf);
1: void    *pdm_audio_open(enum pdm_sample_freq freq,enum pdm_channel channel);
2:
3:

```

- 1、pdm\_audio\_open 打开 pdm 硬件接口, 参数是音频采样率和通道, 参考枚举 pdm\_sample\_freq 和 pdm\_channel, 返回是一个 pdm 音频句柄 (后续需要用到)
- 2、pdm\_audio\_register 注册接口, 参数: audio\_hdl 是 pdm 音频的句柄, priv\_el 是私有结构体(自己注册, 可有可无, 看应用层实现), play\_size 设置音频播放的长度(配置后不能修改, 否则出错), set\_buf(必须注册) 返回是录音的 buf 地址, get\_buf(必须要注册) 在录音完成后, 返回对应的 buf 或者可以寻找到 buf 的结构体指针, 通过参数传入(参考 audio\_demo.c 的实现)
- 3、pdm\_audio\_start 录音正式开始, 内部实现会调用 set\_buf
- 4、pdm\_audio\_close 关闭 pdm 录音

注意: 主要实现的接口是 set\_buf 和 get\_buf 两个函数, priv\_el 是辅助使用, set\_buf 和 get\_buf 基本都是在中断调用, 因此实现不能是耗时操作, 耗时操作可以放到线程当中

### 3.2.2.1. PDM 录音 demo 部分实现代码参考(audio\_demo.c)

- 1、set\_buf 的函数实现, 如图:

```

//配置录音的buf地址
static void *audio_set_buf(void *priv_el,void *el_point)
{
    struct audio_frame_priv *priv = (struct audio_frame_priv *)priv_el;
    struct list_head* get_analy_node;
    struct list_head **point = (struct list_head**)el_point;
    void *buf = NULL;
    get_analy_node = get_app_analyze_node(&(priv->pdm_app));
    if(get_analy_node)
    {
        buf = get_real_buf_adr(get_analy_node);
        if(!buf)
        {
            *point = NULL;
            return NULL;
        }
        *point = get_analy_node;
        set_app_buf_len(get_analy_node,AUDIOLEN);
    }
    return buf;
}
} « end audio_set_buf »

```

priv\_el 是私有结构体, 所有 buf 的申请都是从私有结构体获取

audio\_set\_buf 返回时一个 buf 地址, 用于录音数据填充使用

\*el\_point 是一个指针引用, **必须赋值**, 这个可以是 buf 地址可以是其他地址, 最后会返回给到 get\_buf 中, 这里返回的是 get\_analy\_node, get\_buf 会获取到 get\_analy\_node, 通过 get\_analy\_node 索引到 buf 地址, 所以 get\_analy\_node 是与 buf 地址是绑定的

- 2、get\_buf 的函数实现, 如图:



```

//获取录音的buf地址
static void audio_get_buf(void *priv_el, void *el_point)
{
    struct audio_frame_priv *priv = (struct audio_frame_priv *)priv_el;
    struct list_head *get_analy_node = (struct list_head *)el_point;
    int res;
    if(!get_analy_node)
    {
        printf("%s:%d err\n", __FUNCTION__, __LINE__);
        return;
    }
    res = csi_kernel_msgq_put(priv->pdm_msgq, &get_analy_node, 0, 0);
    //正常应该保证不进这里,如果进来代表任务没有获取队列,直接配置下一个buf导致的
    if(res)
    {
        printf("%s:%d err res:%d!!!!!!!!!!!!!!!!!!!!\n", __FUNCTION__, __LINE__, res);
        force_del_frame(get_analy_node);
    }

    return;
}
} « end audio_get_buf »

```

priv\_el 是私有结构体

\*el\_point 是 get\_analy\_node 的地址, 参考 set\_buf, 是可以寻找到录音 buf 的地址, 这里是通过将 get\_analy\_node 通过消息队列发送, 在其他线程调用, 因为 set\_buf 与 get\_buf 都是在中断去回调, 所以不适宜做一些音频处理或者耗时的操作, res 代表是否发送消息队列成功, 不成功, 则将 get\_analy\_node 删除, 释放资源

### 3、音频处理任务实现, 如图:

```

static void audio_deal_task(void *arg)
{
    struct audio_frame_priv *priv = (struct audio_frame_priv *)arg;
    int16* addr;
    int res;
    int i;
    struct list_head* get_analy_node;
    TYPE_FIRST_ORDER_FILTER_TYPE filter_ctl;
    rm_dc_filter_init(&filter_ctl);
    while(1)
    {
        res = csi_kernel_msgq_get(priv->pdm_msgq, &get_analy_node, -1);
        if(!res)
        {
            //中断有get_analy_node传出来,代表可以发送,如果是NULL,则去设置下一个buf
            if(get_analy_node)
            {
                #if 1
                if(!priv->stable_flag)
                {
                    priv->stable_num--;
                    if(priv->stable_num <= 0)
                    {
                        //等待pdm设备稳定,所以将buf重新给到pdm硬件使用
                        priv->stable_flag = 1;
                    }
                    force_del_frame(get_analy_node);
                    continue;
                }
                #endif

                //将音频数据滤波处理
                addr = (int16*)get_real_buf_addr(get_analy_node);
                for(i=0; i<AUDIOLEN/2; i++) {
                    addr[i] = rm_dc_filter(&filter_ctl, addr[i]);
                    addr[i] = pcm_volum_gain(addr[i], PCM_GAIN);
                }

                //设置处理完毕的时间戳
                set_real_buf_timestamp(get_analy_node, csi_kernel_get_ticks());
                printf("N");
                map_realnode_2_app_node_msg(&(priv->pdm_app), get_analy_node);
                wake_up_analyze_list_app(&(priv->pdm_app));
            }
        }
    }
} « end audio_deal_task »

```

由于获取到的 pdm 音频是没有经过滤波处理, 这里在 get\_buf 中将音频的 buf 通过消息队列



发送到 audio\_deal\_task 任务中,所以所有音频的数据都会在这里进行处理

- ① 先获取 get\_analy\_node, 然后通过 get\_real\_buf\_adr 获取到实际录音的 buf 地址 (在 set\_buf 已经说明, get\_analy\_node 与 buf 是一个绑定作用)
- ② Buf 数据通过 rm\_dc\_filter 与 pcm\_volum\_gain 进行处理, 最后获得的是 pcm 数据

4、Demo 初始化流程:

```
void *audio_task(const char *name)
{
    struct audio_frame_priv *priv_el;
    void *audio_priv;
    priv_el = audio_creat((void *)name);
    audio_priv = pdm_audio_open(PDM_SAMPLE_FREQ_16K, PDM_CHANNEL_RIGHT);
    if(audio_priv)
    {
        priv_el->audio_hardware_hdl = audio_priv;
        pdm_audio_register(audio_priv, priv_el, AUDIOLEN, audio_set_buf, audio_get_buf);
        pdm_audio_start(audio_priv);
    }
    return (void*)priv_el;
}
```

- ① audio\_creat, 创建 demo 的私有结构体
- ② pdm\_audio\_open 打开 pdm 的硬件, 设置 16000HZ 采样率, 设置右声道采集
- ③ pdm\_audio\_regeister 注册 set\_buf 和 get\_buf 函数
- ④ pdm\_audio\_start 开始录音

### 3.3. I2S(8211)播放音频

#### 3.3.1. I2S(8211)相关参数配置

在 sdk 中有自带音频的 demo, 主要是通过 audio\_8211\_demo\_init 调用后, 注册成功, demo 是基于 sdk 的框架去实现, 可以结合 pdm 或者 sd 卡播放音频, 如果有自己的一个音频流, 需要自行实现

#### 3.3.2. I2S(8211)接口说明

1、如图:主要关注 audio\_i2s\_config 这个结构体

- ① i2s\_dev 是注册 i2s 的句柄, 与 i2s\_devid 绑定, 不需要去配置
- ② current\_node 和 reg\_node 不需要配置
- ③ type 设置是麦克风还是喇叭功能
- ④ i2s\_devid 设置使用的是哪个 i2s 设备, 如: HG\_IIS0\_DEVID
- ⑤ sample\_freq 播放或者录音的采样率
- ⑥ sample\_bit 播放音频或者录音的位数
- ⑦ data\_fmt 是一个格式, 根据 i2s 设备去配置
- ⑧ play\_empty\_buf 在播放音频的时候需要配置, 在没有音频数据的时候, 默认播放的音频 buf 数据, 一般是将 play\_empty\_buf 数据清 0

- ⑨ buf\_size 是 play\_empty\_buf 的 size
- ⑩ set\_buf 和 get\_buf 是播放音频或者录音的回调函数, 非常重要, 参数说明参考 3 的例子说明
- ⑪ priv\_el 是私有结构体, set\_buf 和 get\_buf 的第一个参数, 如果不需要, 则设置为 NULL, 是为了 set\_buf 和 get\_buf 的使用

```

24
25 //在有mic和喇叭的时候, sample_freq与sample_bit应该要一致, 所以只有一个是有效值, 默认是以喇叭的配置为准
    You, 13分钟前 | 2 authors (oudi and others)
26 typedef struct audio_i2s_config
27 {
28     //无需用户配置
29     struct i2s_device *i2s_dev;
30     void *current_node;
31     void *reg_node;
32     i2s_irq_func irq_func;
33
34     //用户配置
35     uint8 type; //0:喇叭 1:麦克风
36     int i2s_dev_id;
37     int sample_freq;
38     int sample_bit;
39     int data_fmt; //I2S_DATA_FMT_I2S I2S_DATA_FMT_MSB I2S_DATA_FMT_LSB I2S_DATA_FMT_PCM
40     void *play_empty_buf; //作为喇叭的时候, 需要配置, size与buf_size一致, 可以是malloc也可以是固定, 如果是malloc, 需要自己去free
41     int buf_size;
42     audio_i2s_set_buf set_buf;
43     audio_i2s_get_buf get_buf;
44     void *priv_el;
45
46 } audio_i2s_config;
    You, 1小时前 * 1、添加8211的demo(默认双声道, 所以pdm也要双声道才...
47

```

## 2、如图:8211 初始化

```

*****/
audio_8211_frame_priv * audio_8211_demo_init(const char *name)
{
    int ret;
    audio_i2s_config *play = NULL;
    void *empty_buf = NULL;
    play = (audio_i2s_config*)malloc(sizeof(audio_i2s_config));
    empty_buf = (void*)malloc(PLAY_EMPTY_SIZE);
    if(!play || !empty_buf)
    {
        printf("%s:%d\n", __FUNCTION__, __LINE__);
        goto audio_8311_demo_init_err;
    }
    memset(play, 0, sizeof(audio_i2s_config));
    memset(empty_buf, 0, PLAY_EMPTY_SIZE);

    //OS_TASK_INIT("sim_mic", &sim_mic_task, sim_mic_thread, (uint32)NULL, OS_TASK_PRIORITY_BELOW_NORMAL, 1024);
    //os_sleep_ms(200);

    //该任务及资源初始化
    audio_8211_frame_priv *audio_8211_priv = audio_8211_creat(name);
    //8311 install, 注册对应函数
    //对play mic config初始化
    play->type = PLAY_MODE;
    play->i2s_dev_id = HG_IIS0_DEVID;
    play->sample_freq = I2S_SAMPLE_FREQ_16K;
    play->sample_bit = I2S_SAMPLE_BITS_16BITS;
    play->data_fmt = I2S_DATA_FMT_LSB;
    play->priv_el = audio_8211_priv;
    play->set_buf = audio_play_set_buf;
    play->get_buf = audio_play_get_buf;
    play->buf_size = PLAY_EMPTY_SIZE;
    play->play_empty_buf = empty_buf;
    int res = audio_i2s_install(play, NULL);
    printf("%s:%d\n", __FUNCTION__, __LINE__);
}

```

主要是将 play 的一些必要参数配置完成后, 调用 audio\_i2s\_install 正常运行, audio\_i2s\_install 有两个参数, 分别是播放音频和麦克风, 8211 只有播放音频, 所以主要注册 play 就可以了

### 3、接下来解析 set\_buf 和 get\_buf 参数意义

在 demo 中 set\_buf 对应 audio\_play\_set\_buf, get\_buf 对应 audio\_play\_get\_buf

set\_buf (void \*priv\_el, void \*el\_point, int \*buf\_size),  
priv\_el 对应 audio\_i2s\_config 中的 priv\_el, 返回值是需要配置的 buf, \*buf\_size 是值返回的 buf 的 size, 通过指针赋值返回, 用于设置播放音频的 size 或者录音的 size, \*el\_point 是一个指针, 中断完成后, 会传递到 get\_buf 中, 其中返回值和 \*buf\_size 是必须的, priv\_el 和 \*el\_point 是辅助获取 buf 和 buf\_size 的信息

get\_buf (void \*priv\_el, void \*el\_point, int \*buf\_size);  
priv\_el 对应 audio\_i2s\_config 中的 priv\_el, \*el\_point 是对应 set\_buf 中 \*el\_point, set\_buf 赋值的什么内容, 这里就能获取到什么内容, \*buf\_size 暂时没有调用(保留与 get\_buf 参数一致)