

# 如何基于 PY32 触摸库进行应用开发

V1.0



**Puya Semiconductor (Shanghai) Co., Ltd**

# 目录

1. 简介.....	3
2. 硬件设计注意事项.....	4
2.1 芯片烧录.....	4
2.2 PF2 复位引脚.....	5
3. 软件开发指南.....	6
3.1 触摸配置.....	6
3.1.1 通道配置 (tk_cfg_user.h) .....	6
3.1.2 触摸低功耗配置.....	7
3.1.3 内部参考通道配置.....	8
3.1.4 防水模式设置.....	9
3.1.5 隔空触摸设置.....	10
3.1.6 触摸按键处理.....	10
3.2 外设配置 (app_config.h) .....	11
3.2.1 定时器设置.....	11
3.2.2 蜂鸣器设置.....	12
3.2.3 ADC/NTC 检测.....	12
3.2.4 数码管驱动.....	13
3.2.5 UART.....	14
3.2.6 用户数据保存.....	14
3.2.7 PWM 输出.....	15
3.2.8 调试信息打印.....	15
3.2.9 RTC.....	16
3.2.10 选项字节配置 Option.....	16
3.2.11 红外接收.....	17
3.2.12 看门狗.....	17
3.2.13 类 W2812B 灯珠驱动.....	17
4. GPIO 函数接口.....	18
4.1.1 GPIO 初始化.....	18
4.1.2 GPIO 电平设置以及读取.....	18
4.1.3 GPIO 中断模式回调函数.....	18
5. user_code.c.....	19
6. 更新历史.....	20

## 1. 简介

本应用笔记介绍了如何基于 PY32 触摸库进行产品应用开发，包括硬件设计注意事项、触摸库功能及接口定义、外设使用方法等。基于 PY32 触摸库及其标准工程进行应用开发可以大幅降低开发的难度，使整体开发效率大为提高。

表格 1-1 适用产品

类型	产品系列
微型控制器系列	PY32T020、PY32T020-B、PY32C681、PY32C682

## 2. 硬件设计注意事项

### 2.1 芯片烧录

1) 程序烧录需要使用 4 根线，分别为 VCC，GND，PF3（SWCLK），PF4（SWDIO）来进行烧录，触摸上位机以及程序在线 Debug 都通过这 4 根线进行；所以在 IO 口富余的条件下尽量不使用 PF3、PF4 这两个管脚，或者只拿来做一些不影响整体逻辑的功能，点个灯之类，方便触摸调试的时候将其屏蔽掉；

2) PF3，PF4 最好别拿来驱动数码管，防止烧录的时候由于别的 IO 口可能存在电压变化串扰到这两个 IO 口，造成烧录不了；

3) 当 PF3，PF4 复用为别的功能后，需要使用 PY Touch-Link 上位机进行烧录，或者使用量产脱机烧录器；

4) PY-LINK 仿真器默认没有选择电源，所以烧录的时候需要使用一个跳线帽将 VCC 跳至 3.3V 或者 5V，如图 2-1 所示

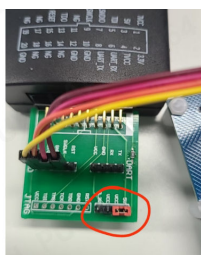


图 2-1

5) 上位机下载失败，显示如图 2-2 所示，这是因为 PCB 板子上有大电容，而烧录器烧录的时候会自动上下电一次，此时很可能会导致 USB 过流保护，造成烧录不进；需要将转接板上 3.3V 的连接线割断（如图 2-3 所示），串联一个 47R 电阻，下载的时候选择 3.3V 烧录即可；

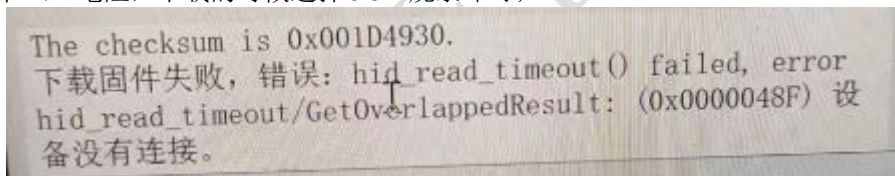


图 2-2

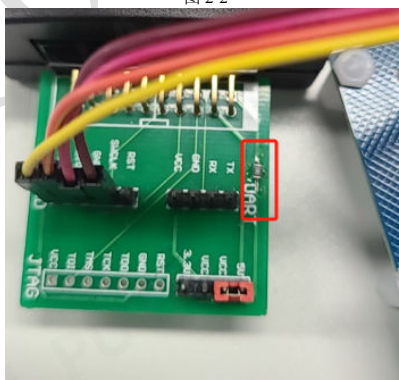


图 2-3

## 2.2 PF2 复位引脚

- 1) PF2 默认为复位功能，低电平复位，所以应用的时候需要保证 PF2 口不会被外部拉低，造成芯片无法工作；
- 2) 如果 PF2 需要用做其他功能（输入、输出、触摸等），需要在程序内将其模式改为 GPIO 模式，如图 2-4 所示

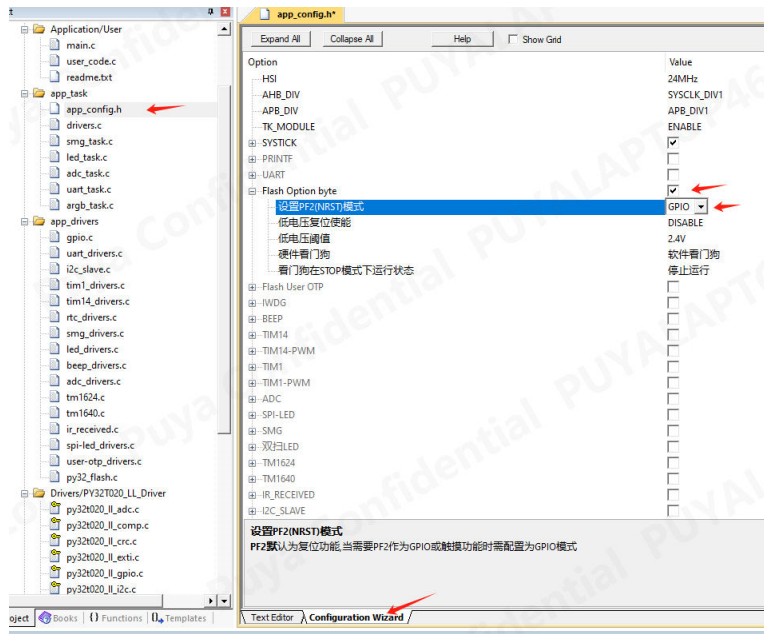


图 2-4

### 3. 软件开发指南

### 3.1 触摸配置

### 3.1.1 通道配置 (tk\_cfg\_user.h)

该文件用于配置触摸按键通道，按键触发阈值；滑条通道，滑条类型、分辨率以及滑条触发阈值；

- 1) 触摸按键配置, 如图 3-1 所示, 按照实际情况更改, 以 `TK_CH_NONE` 结尾; GPIO 对应的触摸通道号可查看数据手册引脚定义这部分, 如图 3-2 所示;

Pin#	Pin Name	IO Mode	IO Type	IO Direction	Notes	接口连接	
						管脚功能	引脚连接
CH_KEY0	TK_CH17						
CH_KEY1	TK_CH15						
CH_KEY2	TK_CH11						
CH_KEY3	TK_CH15						
CH_KEY4	TK_CH16						
CH_KEY5	TK_CH10						
CH_KEY6	TK_CH7						
CH_KEY7	TK_CH6						
CH_KEY8	TK_CH_NONE						
CH_KEY9	TK_CH_NONE						
CH_KEY10	TK_CH_NONE						
CH_KEY11	TK_CH_NONE						
CH_KEY12	TK_CH_NONE						
CH_KEY13	TK_CH_NONE						
CH_KEY14	TK_CH_NONE						
CH_KEY15	TK_CH_NONE						
CH_KEY16	TK_CH_NONE						
CH_KEY17	TK_CH_NONE						
CH_KEY18	TK_CH_NONE						
CH_KEY19	TK_CH_NONE						
CH_KEY20	TK_CH_NONE						
CH_KEY21	TK_CH_NONE						
CH_KEY22	TK_CH_NONE						
CH_KEY23	TK_CH_NONE						
CH_KEY24	TK_CH_NONE						
CH_KEY25	TK_CH_NONE						

图 3-1

图 3-2

- 2) 触摸按键门限值设置, 如图 3-3 所示, 当按键的数据增量超过该值且稳定超过滤波时间后, 认为按键触发

```
#define KEY0 THD 100
#define KEY1 THD 100
#define KEY2 THD 100
#define KEY3 THD 100
#define KEY4 THD 100
#define KEY5 THD 100
#define KEY6 THD 100
#define KEY7 THD 100
#define KEY8 THD 100
#define KEY9 THD 40
#define KEY10 THD 40
#define KEY11 THD 40
#define KEY12 THD 40
#define KEY13 THD 40
#define KEY14 THD 40
#define KEY15 THD 40
#define KEY16 THD 40
#define KEY17 THD 40
#define KEY18 THD 40
#define KEY19 THD 40
#define KEY20 THD 40
#define KEY21 THD 40
#define KEY22 THD 40
#define KEY23 THD 40
#define KEY24 THD 40
#define KEY25 THD 40
```

图 3-3

- 3) 滑条功能设置, 如图 3-4 所示, 目前触摸库支持 4 个滑条, 每个滑条最多支持 8 个通道;

当需要滑条类型时，将 `SLIDER_OR_WHEEL0_TYPE` 定义为 `TK_APP_SLIDER`，当需要滑环类型时，定义为 `TK_APP_WHEEL`；其他情况定义为 `TK_APP_NONE`；

滑条分辨率设置，设置数据输出的最大值；

滑条门限值，设置滑条触发的门限值，当相邻的 3 个通道的数据增量超过该值且稳定超过滤波时间后，认为滑条触发；

滑条通道设置,按照实际情况更改,以 TK CH NONE 结尾;

```

//=====
#define SLIDER_OR_WHEELO_TYPE          TK_APP_NONE           //滑条类型
#define SLIDER_OR_WHEELO_RESOLUTION    100                  //滑条分辨率
#define SLIDER_OR_WHEELO_THD           80                   //滑条门限值
#define SLIDER_OR_WHEELO_CH0           TK_CH4               //滑条通道，按顺序填写
#define SLIDER_OR_WHEELO_CH1           TK_CH8
#define SLIDER_OR_WHEELO_CH2           TK_CH5
#define SLIDER_OR_WHEELO_CH3           TK_CH6
#define SLIDER_OR_WHEELO_CH4           TK_CH7
#define SLIDER_OR_WHEELO_CH5           TK_CH_NONE
#define SLIDER_OR_WHEELO_CH6           TK_CH_NONE
#define SLIDER_OR_WHEELO_CH7           TK_CH_NONE

```

图 3-4

### 3.1.2 触摸低功耗配置

1) 低功耗相关函数，如表 2 所示

函数名	功能
void LoopStop_Callback(void)	芯片唤醒后执行后的函数，无论是外部中断唤醒、RTC 唤醒、触摸唤醒都会执行这个函数；如果达不到退出休眠的条件则会继续进入 STOP 模式，可以在该函数内添加自己的处理，当需要退出休眠时，手动执行 TKCtr.TK_state = TK_GO 语句即可退出休眠，同时需要将 TKCtr.SleepTime 变量清 0，不然又会很快进入省电模式；
void EnterStop_Callback(void)	芯片进入省电模式前调用的函数，主要用于将一些外设关闭，设置 IO 口状态等降低功耗等操作；
void ExitStop_Callback(void)	芯片退出省电模式后调用的函数，主要用于恢复外设等操作；

2) 在 app\_config.h 中开启 RTC 模块，选择时钟源，设置秒中断时间。如果需要定时唤醒起来检测一些外部动作，则需要开启秒中断以及 RTC 总中断（仅需要触摸唤醒的时候不用开启 RTC 相关中断）；如图 3-5 所示



图 3-5

3) 在 tk\_cfg.h 中开启触摸省电选项，如图 3-6 所示，配置进入休眠的时间以及唤醒门限值，唤醒门限值设置可打开 STOP 调试模式，连接上位机观察休眠后手指按下 TK-COM 通道的差值，如图 3-7 所示；（调试完成后需要把 STOP 调试模式关闭，不然芯片不是真正的进入低功耗模式）；

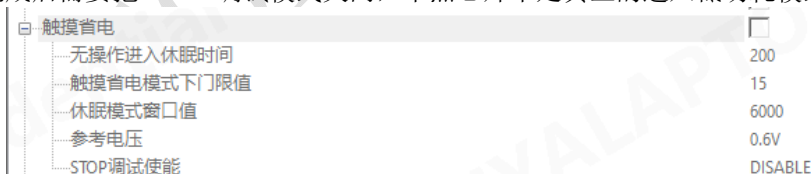


图 3-6

触摸按键	滑条/滑轮	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	
按键序号		KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	
通道	TK17	TK15	TK11	TK5	TK16	TK10	TK_COM...	
基准值	8404	8422	8405	8433	8401	8436	5409	
原始值	8406	8425	8402	8437	8402	8434	5412	
差值	2	3	-3	4	1	-2	3	
门限值	100	100	100	100	100	100	15	
按键次数	0	0	0	0	0	0	0	
状态								

图 3-7

3.1.3 内部参考通道配置

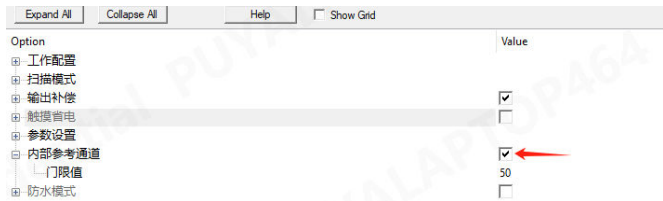


图 3-8

- 1) 芯片内部存在一个参考电容，该通道的采样数据理论上不会有过多的数据变化；
- 2) 打开内部参考通道后，上位机显示会多出一个 TK-REF 的通道，用于显示内部通道的数据，如图 3-9 所示

按键序号	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7	
通道	TK17	TK15	TK11	TK5	TK16	TK10	TK7	TK6	TK-REF
基准值	8406	8388	8404	8430	8401	8397	8430	83	6581
原始值	8406	8388	8405	8429	8398	8398	8433	83	6576
差值	0	0	1	-1	-3	1	3	0	-5
门限值	100	100	100	100	100	100	100	100	50
按键次数	0	0	0	0	0	0	0	0	0
状态									

图 3-9

- 3) 正如第一点所说的，因为该通道存在内部，所以会该通道数据造成异常波动的条件只能来源于电源以及一些辐射干扰；可以通过设置该通道的门限值来做一些在环境异常时做的一些保护性动作，避免按键误触发；（当 VCC 电压变化时该数据也会发生变化，特别是灯光全亮跟全灭交替变化的时候，设置门限值的时候需要查看这段时间的数据变化范围）



3.1.4 防水模式设置



图 3-10

- 1) 因为防水模式会牺牲一定的按键响应速度，所以没有防水需求的时候把防水功能关闭；
- 2) 开启防水功能后，需要键输出补偿设置为同相，如图 3-11 所示

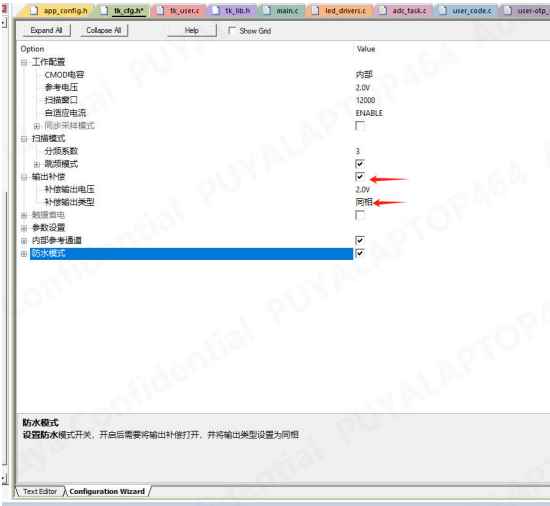


图 3-11

- 3) 开启防水模式后，触摸库会自动添加一个屏蔽通道，上位机会显示一个 TK-COM 通道，用于显示屏蔽通道的数据，如图 3-12 所示

按键序号	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7		
通道	TK17	TK15	TK11	TK5	TK16	TK10	TK7	TK6	TK_COM...	TK-REF
基准值	8433	8429	8411	8440	8436	8425	8408	82	8399	6652
原始值	8431	8434	8408	8440	8440	8425	8409	83	8399	6647
差值	-2	5	-3	0	4	0	1	1	0	-5
门限值	100	100	100	100	100	100	100	100	85	50
按键次数	0	0	0	0	0	0	0	0	0	0
状态										

图 3-12

- 4) SHIELD 通道选择，根据硬件选择对应的通道号
- 5) 有水按键处理，可以设置为正常触发跟屏蔽按键，当检测到有水时会进行相对应的逻辑处理；而检测是否有水需要调整 tk\_user.c 中 Ratio\_Tbl 的数值，如图 3-13 所示

```
const uint16_t Ratio_Tbl[9][2] =  
{  
    {25, 70},  
    {25, 70},  
    {25, 70},  
    {25, 70},  
    {25, 70},  
    {25, 70},  
    {25, 70},  
    {25, 70},  
    {25, 70},  
};
```

图 3-13

- 6) 如图 13 所示，该数组为一个二维数组，数组长度为按键数量，触发按键会有 3 种状态，

- 第一种，覆水状态，即面板上铺满水，此状态输出的按键比例相对较小；（对应图 13 的数据 25），该参数需要根据实际情况进行调整；
- 第二种，无水状态，即面板上无水，此状态输出的按键比例居中；
- 第三种，泼水状态，即有水流冲过，此状态输出的按键比例较大；（对应图 13 的数据 70），该参数需要根据实际情况进行调整；
- 7) 防抹布功能，实现逻辑为当屏蔽通道的数据变化量大于门限值时，屏蔽按键，该逻辑优先于有水按键响应；

3.1.5 隔空触摸设置

1) 打开同步采样模式，如图 3-14 所示

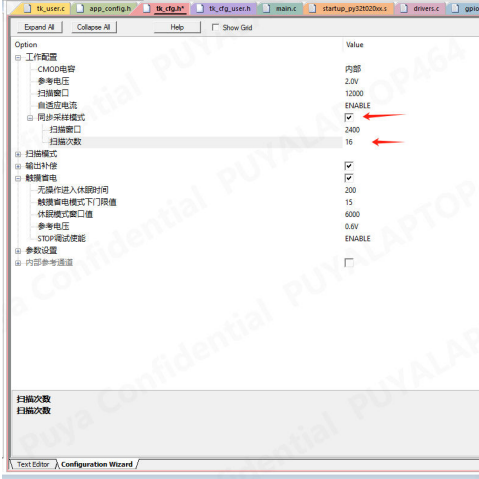


图 3-14

- 2) 设置根据隔空距离设置合适的扫描次数
- 3) 通过上位机进行数据观察，手指触摸查看是否达到触发阈值（隔空触摸根据经验阈值设置为 50 较合适），相应的减少或增加扫描次数；

3.1.6 触摸按键处理

当有触发按键产生时，会先调用 APP\_TouchKeyFlagsMask 函数，如下图所示，按键输出数据为该函数的返回值，可以在该函数内执行一些特殊的逻辑处理，例如防水按键处理，单按键触发，多按键触发根线基线等操作；

```
/**
 * @brief TK按键特殊处理
 * @param None
 * @retval None
 */
uint32_t APP_TouchKeyFlagsMask(void)
{
    #if (SLEEP_EN)
        TKCtr.SleepTime = 0; // 按键变化，计时清0
    #endif
    #if WATER_PROOF_EN
        static uint8_t last_key = TK_CH_NONE;
        static uint16_t last_Differ;
        uint32_t KeyFlags = 0;
        uint8_t now_key = TK_CH_NONE;
        int16_t Differ = 0;
        int16_t ratio;
    #if MAX_TRIGGER_KEY_CNT
        uint32_t num = 0;
    #endif
    for (uint32_t i = 0; i < TKCtr.TouchKeyChCnt; i++)
```

## 3.2 外设配置 (app\_config.h)

### 3.2.1 定时器设置

- 1) 芯片内部有 3 个定时器，分别为 SYSTICK、TIM1、TIM14；因为 TIM1 跟 TIM14 有可能需要输出 PWM 信号，所以推荐使用 SYSTICK 定时器；
- 2) 打开 SYSTICK 选项，如图 3-15 所示，可配置项为定时时间跟优先级，定时时间默认为 1ms，也可设置为 100us, 125us, 200us, 250us, 500us，能够被 1ms 整除的时间，因为该定时器也为触摸库提供时基；

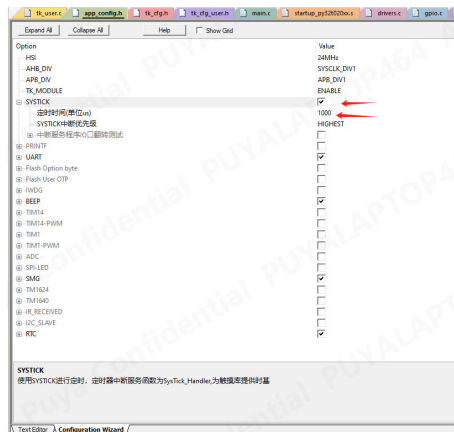


图 3-15

- 3) 打开 TIM1 选项，如图 3-16 所示，TIM1 的时间设置没有要求，任意时间都可以，当打开翻转测试后，GPIO 会在定时器中断内进行 IO 翻转，可通过示波器查看时间是否正确；定时器回调函数为 TIM1\_PeriodElapsedCallback，如图 3-17 所示，TIM14 类似；



图 3-16

```

/*****
** 函数名 void TIM1_PeriodElapsedCallback(void)
** 描述   : 定时器1回调函数，处理定时事件
** 传入   : 无
** 返回   : 无
*****/
void TIM1_PeriodElapsedCallback(void)
{
    #if (APP_IR_RECEIVED_ENABLE == 1 && D_IR_TIM == 1)
        IR_Received_Scan();
    #endif
}

```

图 3-17

### 3.2.2 蜂鸣器设置

打开 BEEP 选项，如图 3-17 置蜂鸣器输出管脚以及输出类型，当蜂鸣器与 ADC 管脚复用时，将复用选项勾选上；（如果是无源蜂鸣器，需要开启 SYSTICK 定时器，并将定时时间设置合适，因为无源蜂鸣器的 IO 翻转是在 SYSTICK 定时器里面做的翻转）

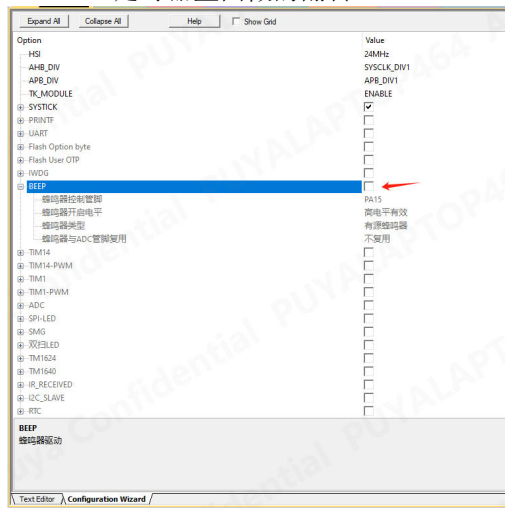


图 3-17

### 3.2.3 ADC/NTC 检测

打开 ADC 的选项，如图 3-18 置采样间隔时间，配置 ADC 的检测管脚；初始化完成后 ADC 会定时进行转换，转换完成之后会调用 ADC\_Callback()函数，所以用户可以在该函数中添加 ADC 的处理函数，采集的数据存放在 ADCxConvertedData 这个数组之中，如图 3-19 所示；

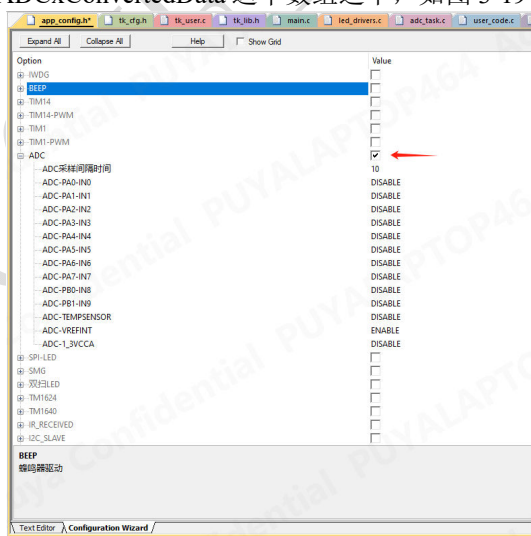


图 3-18

```
#if APP_ADC_ENABLE
/* ***** */
** 函数名 void ADC_Callback(void)
** 描述 : ADC转换完成回调，在该函数内处理ADC数据
** 传入 : 无
** 返回 : 无
** ***** */
void ADC_Callback(void)
{
    #if 0
    /* 读取通道数据 */
    log_printf("VREFINT = %d\r\n", ADCxConvertedData[ADC_CHANNEL_VREFINT]);
    log_printf("CH8 = %d\r\n", ADCxConvertedData[ADC_CHANNEL_8]);
    #endif
}
#endif
```

图 3-19

### 3.2.4 数码管驱动

- 1) 常规数码管使用 SEG 跟 COM 通过单片机直接驱动，打开 SMG 选项，如图 3-20 所示，可通过配置 COM-GPIO 以及 SEG-GPIO 进行数码管的初始化，后续可通过更改 `smg_data` 这个数组来更改显示内容；当数码管扫描会严重影响触摸数据时，可开启使用 TK 扫描 LED 选项，用于降低干扰；



图 3-20

- 2) 双扫类型的数码管通过单片机 IO 口直接驱动时，打开双扫 LED 选项，如图 3-21 所示，通过配置 LED-GPIO 来进行数码管的初始化，后续可通过更改 `led_data` 这个数组来更改显示内容；当数码管扫描会严重影响触摸数据时，可开启使用 TK 扫描 LED 选项，用于降低干扰；

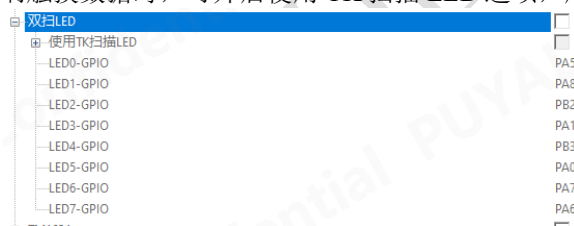


图 3-21

- 3) 使用 TM1624 以及类似芯片驱动数码管，打开 TM1624 选项，如图 3-22 所示，通过配置 CLK、DIN、STB 管脚来进行数码管初始化，然后根据需求更改显示模式以及显示亮度；可打开测试选项来进行数码管测试，打开后数码管会全亮、全灭的进行闪烁；通过调用 `TM1624_Display_Update` 进行显示刷新；



图 3-22

- 4) 使用 TM1640 以及类似芯片驱动数码管，打开 TM1640 选项，如图 3-23 所示，通过配置 CLK、DIN 管脚来进行数码管初始化，然后根据需求更改显示模式以及显示亮度；可打开测试选项来进行数码管测试，打开后数码管会全亮、全灭的进行闪烁；通过调用 `TM1640_Display_Update` 进行显示刷新；

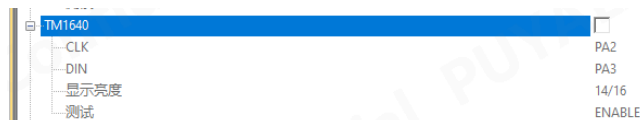


图 3-23

### 3.2.5 UART

打开 UART 选项，如图 3-24 所示，可配置 UART 波特率，RX、TX 管脚，中断使能以及中断优先级配置，打开中断后，UART 会以队列的方式进行数据发送以及读取，默认的队列长度为 64 个字节；可以手动更改大小，如图 3-25 所示，当 TX 跟 RX 管脚不注意颠倒时，可将 UART1\_SWAP 设为 1 即可；串口 1、2、3 设置类似；当打开串口收发测试后，单片机收到数据后会原封不动的回发数据，方便进行收发测试；



图 3-24

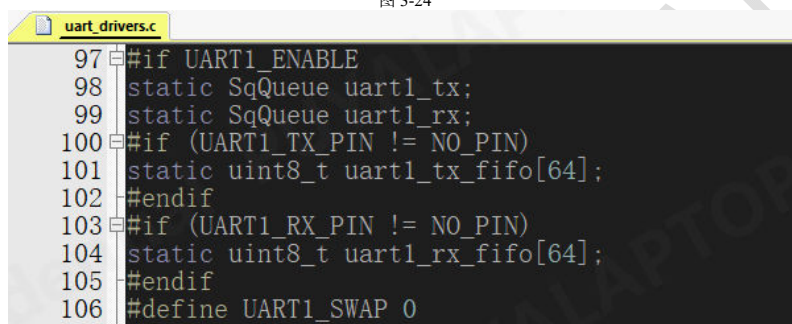


图 3-25

需要进行串口 1 发送时，调用 UART1\_QueueSend 函数发送即可，函数返回 0 时，表示发送失败，1 表示发送成功；

调用 UART1\_QueueRead 函数进行数据读取，返回 1 时表示数据接收成功，返回 0 表示队列中没有数据；

### 3.2.6 用户数据保存

打开 Flash User OTP 选项，如图 3-26 所示，打开后可调用 User\_Cache\_Read 读取数据，调用 User\_Cache\_Write 将数据写入缓存，因为 FLASH 只能页写，所以最后调用 User\_Flash\_Write 将缓存数据写入 FLASH，减少 FLASH 的擦写次数；当需要进行掉电数据保存时，可打开掉电数据保存的选项，设置对应的检测电压（需要开启 ADC 模块，并勾选 ADC-VREFINT 选项）；当发生掉电时，会回调 ADC\_LVD\_ISR 函数，如图 3-27 所示，用户在该函数内执行数据存储的操作；

**注：**因为是掉电过程中进行数据写入，所以为了保证数据的可靠性，需要读取的时候增加校验，防止由于数据出错造成异常，其次就是掉电速度不能过快，不然数据来不及存储；

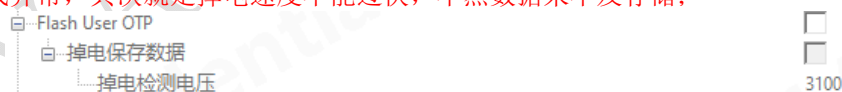


图 3-26

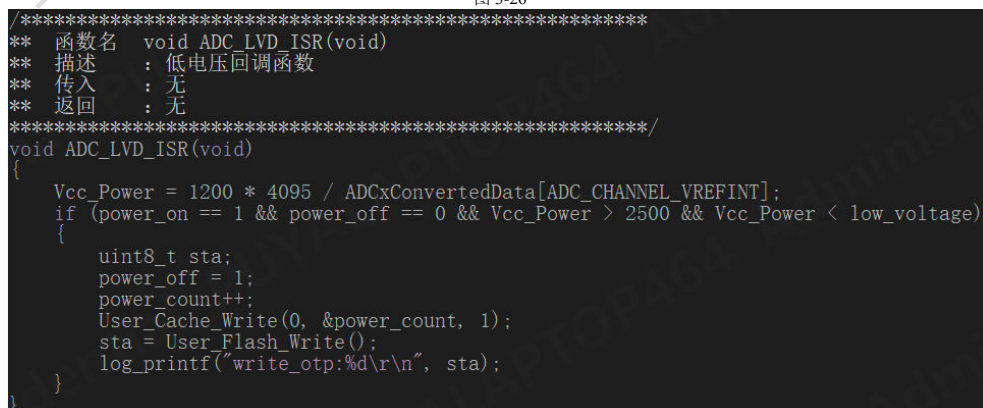


图 3-27



### 3.2.7 PWM 输出

芯片内部有 2 个定时器支持 PWM，分别为 TIM1、TIM14；其中 TIM14 支持 1 路 PWM 输出，TIM1 支持 4 路 PWM 输出；

打开 TIM1-PWM 选项，如图 3-28 所示，可以进行频率的设置，空闲电平状态，分辨率设置（即可以设置的最大输出值）；默认输出百分比设置，PWM 输出管脚设置；之后可以通过 TIM1\_PWM\_Pulse 进行 PWM 占空比调整，TIM14 类似；



图 3-28

### 3.2.8 调试信息打印

开发过程中很多情况需要打印提示信息，此时可以打开 PRINTF 选项，如图 3-29 所示，默认情况只用勾选 PRINTF 即可，使用触摸调试上位机进行数据打印，程序下载后使用上位机连接，之后日志部分就可以显示打印的数据，如图 3-30 所示；注：上位机调试需要使用 PF3，PF4 两个接口，然后打印不能过于过快，防止上位机读取数据不及时导致数据异常；

当勾选了 UART 之后，需要开启对应的 UART 模块；然后就会使用 UART 进行数据输出；

调用方法为 log\_printf，使用方法跟 printf 一致；



图 3-29



图 3-30

### 3.2.9 RTC

打开 RTC 选项，如图 3-31 所示，

- 1) 选择 RTC 的时钟源；
- 2) 配置秒中断产生时间（需要触摸唤醒时建议将秒中断产生时间设置为 100ms），开启秒中断使能后中断回调函数为 RTC\_SecCallback，如图 3-32 所示；
- 3) 闹钟中断产生计数值就是多少个秒中断产生后发生的中断，如图 3-31 所配置的为 1 秒产生一个闹钟中断，开启闹钟中断使能后中断回调函数为 RTC\_AlrCallback，如图 3-33 所示；
- 4) 根据需要开启对应的中断；



图 3-31

```

/*****
** 函数名 void RTC_SecCallback(void)
** 描述   RTC秒中断中断回调函数
** 传入   : 无
** 返回   : 无
*****/
void RTC_SecCallback(void)
{
}

```

图 3-32

```

/*****
** 函数名 void RTC_AlrCallback(void)
** 描述   RTC闹钟中断中断回调函数
** 传入   : 无
** 返回   : 无
*****/
void RTC_AlrCallback(void)
{
}

```

图 3-33

### 3.2.10 选项字节配置 Option

打开 Flash Option byte，如图 3-34 所示

- 1) 配置 PF2 的端口的模式；
- 2) 设置低电压复位使能以及低电压复位电压，开启后休眠功耗多增加 5ua 左右
- 3) 设置看门狗 STOP 模式下状态，如果开启看门狗后设置为正常运行，进入 STOP 后由于无法喂狗，则会造成芯片复位，此时需要开启 RTC 唤醒芯片进行喂狗；
- 4) 硬件看门狗，即芯片上电后自动打开看门狗，但是喂狗还是需要程序内主动喂狗；

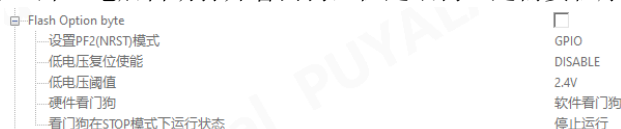


图 3-34



### 3.2.11 红外接收

打开 IR\_RECEIVED 选项，如图 3-35 所示，例程使用定时器扫描 IO 口状态，进行红外信号解码

- 1) 选择红外接收管脚
- 2) 选择相应的定时器（推荐使用 SYSTICK 作为时基，设置为 125us）；
- 3) 程序调用 IR\_Press 函数读取收到的红外数据，如图 3-36 所示，数据包含地址，命令，次数；



图 3-35

```
void app_drivers_loop(void)
{
    #if APP_IR_RECEIVED_ENABLE
        Ir_TypeDef remote;
        if (IR_Press(&remote))
        {
            log_printf("address:0X%X ", remote.ir_address); // 收到的地址
            log_printf("command:0X%X ", remote.ir_command); // 收到的命令
            log_printf("count:%d\r\n", remote.ir_count); // 收到的次数
        }
    #endif
}
```

图 3-36

### 3.2.12 看门狗

打开 IDWG 选项，如图 3-37 所示，程序已经在主函数内执行喂狗功能，如图 3-38 所示，所以无需用户主动喂狗

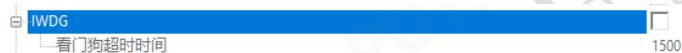


图 3-37

```
#if APP_IWDG_ENABLE
    /* 喂看门狗 */
    LL_IWDG_ReloadCounter(IWDG);
#endif
```

图 3-38

### 3.2.13 类 W2812B 灯珠驱动

打开 SPI-LED 选项，如图 3-39 所示，由于例程使用 SPI 来模拟幻彩灯珠的时序，所以需要占用 SPI 资源，而输出管脚只能选择支持 SPI-MOSI 的引脚；

- 1) 选择输出管脚；
- 2) 选择灯珠数量；
- 3) 调用 SPI\_LED\_RgbLoad 加载 RGB 数据到缓存中；
- 4) 调用 SPI\_LED\_Transmit 将数据发送出来；

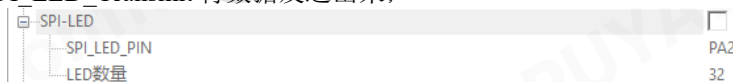


图 3-39

## 4. GPIO 函数接口

### 4.1.1 GPIO 初始化

将 GPIO 设置为模拟模式           ->     GPIO\_Init(PA0,ANALOG)  
 将 GPIO 设置为输入上拉模式       ->     GPIO\_Init(PA0,INPUT|PULL\_UP)  
 将 GPIO 设置为输入上拉下降沿中断模式 ->  
 GPIO\_Init(PA0,INPUT|PULL\_UP|EXTI\_TRIGGER\_FALLING)  
 将 GPIO 设置为推挽输出模式       ->     GPIO\_Init(PA0,OUTPUT|PUSH\_PULL)  
 将 GPIO 设置为开漏输出模式       ->     GPIO\_Init(PA0,OUTPUT|OPENDRAIN)  
 将 GPIO 设置为 TIM1\_CH3 输出       ->     GPIO\_Init(PA0,ALTERNATE | GPIO\_TIM1\_AF2)  
                                   这里选择 GPIO\_TIM1\_AF2 或者 GPIO\_TIM1\_AF5 需要查看规格书的复用功能映射  
 将 GPIO 设置为 UART 功能       ->     GPIO\_Init(PA0,ALTERNATE | GPIO\_UART2)

### 4.1.2 GPIO 电平设置以及读取

将 GPIO 设置为高电平           ->     GPIO\_SetBit(PA0)  
 将 GPIO 设置为低电平           ->     GPIO\_ClearBit (PA0)  
 将 GPIO 电平翻转               ->     GPIO\_ToggleBit (PA0)  
 读取 GPIO 电平               ->     sta = GPIO\_ReadBit (PA0)

### 4.1.3 GPIO 中断模式回调函数

如图 4-1 所示，当 GPIO 发生中断时，会回调该函数，用户需要在该函数内添加逻辑代码：

```

/**
 * 函数名 void EXTI0_15_IRQHandlerCallback(uint32_t PR)
 * 描述   : 外部中断0到15的回调函数，此函数工作在中断内，请勿在函数内进行延时等操作
 * 传入   : PR: 产生的中断标志，BIT0为外部中断0，BIT1为外部中断1，以此类推
 * 返回   : 无
 */
void EXTI0_15_IRQHandlerCallback(uint32_t PR)
{
    uint8_t i;
    for (i = 0; i < 16; i++)
    {
        /* Handle EXTI interrupt request */
        if (PR & (1 << i))
        {
            /* 产生外部中断 */
            EXTI_Flag |= 1 << i;
        }
    }
}

```

图 4-1

## 5. user\_code.c

用户初始化 `user_init`，用户补全逻辑代码；

用户主循环 `user_loop`，用户补全逻辑代码；

用户定时器 `user_timer`，用户补全逻辑代码；

当开启 ADC 功能时，当数据采集完成会回调 `ADC_Callback` 函数，用户在该函数内补全逻辑代码，ADC 采集的原始数据保存在 `ADCxConvertedData` 数组中；

6. 更新历史

Version	Content	Date
V1.0	Initial version	2024/8/16



Puya Semiconductor Co., Ltd.

声 明

普冉半导体(上海)股份有限公司（以下简称：“Puya”）保留更改、纠正、增强、修改 Puya 产品和/或本文档的权利，恕不另行通知。用户可在下单前获取产品的最新相关信息。

Puya 产品是依据订单时的销售条款和条件进行销售的。

用户对 Puya 产品的选择和使用承担全责，同时若用于其自己或指定第三方产品上的，Puya 不提供服务支持且不对此类产品承担任何责任。

Puya 在此不授予任何知识产权的明示或暗示方式许可。

Puya 产品的转售，若其条款与此处规定不一致，Puya 对此类产品的任何保修承诺无效。

任何带有 Puya 或 Puya 标识的图形或字样是普冉的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代并替换先前版本中的信息。

普冉半导体(上海)股份有限公司 - 保留所有权利