

基于 PY32 软件库的触摸应用开发指南



Puya Semiconductor (Shanghai) Co., Ltd

目录

1. 简介	3
2. 软件库结构	4
3. 触摸库介绍与使用	6
3.1 触摸功能类别介绍	6
3.2 MCU 资源需求	7
3.2.1 MCU 硬件资源	7
3.2.2 MCU 软件资源	7
3.3 触摸库接口解析	8
3.3.1 数据接口	8
3.3.2 函数接口	9
3.4 触摸库功能选项及参数配置	10
3.4.1 触摸按键用户配置	10
3.4.2 滑条/滑轮用户配置	11
3.4.3 触摸功能及参数配置	12
3.5 用户可修改的触摸库回调函数	13
3.6 触摸应用配置步骤及流程	14
3.6.1 常规按键应用	14
3.6.2 滑轮/滑条应用	19
3.6.3 触摸低功耗应用	21
3.6.4 隔空按键应用	22
3.6.5 防水按键应用	22
3.6.6 触摸检水应用	24
4. 系统配置及外设应用解析	26
4.1 系统配置	26
4.2 外设应用的接口与使用	27
4.2.1 UART	27
4.2.2 定时器	28
4.2.3 PWM	29
4.2.4 ADC	30
4.2.5 I2C (Slave Mode)	30
4.2.6 RTC	31
4.2.7 看门狗	31
4.3 典型应用驱动	32
4.3.1 数码管/LED 驱动	32
4.3.2 专用驱动 IC 驱动数码管/LED	34
4.3.3 红外遥控接收	34
4.3.4 用户数据掉电存储	35
4.3.5 类 W2812B 灯珠驱动	36
4.3.6 蜂鸣器控制	37
5. 用户应用程序接口	38
6. 更新历史	39

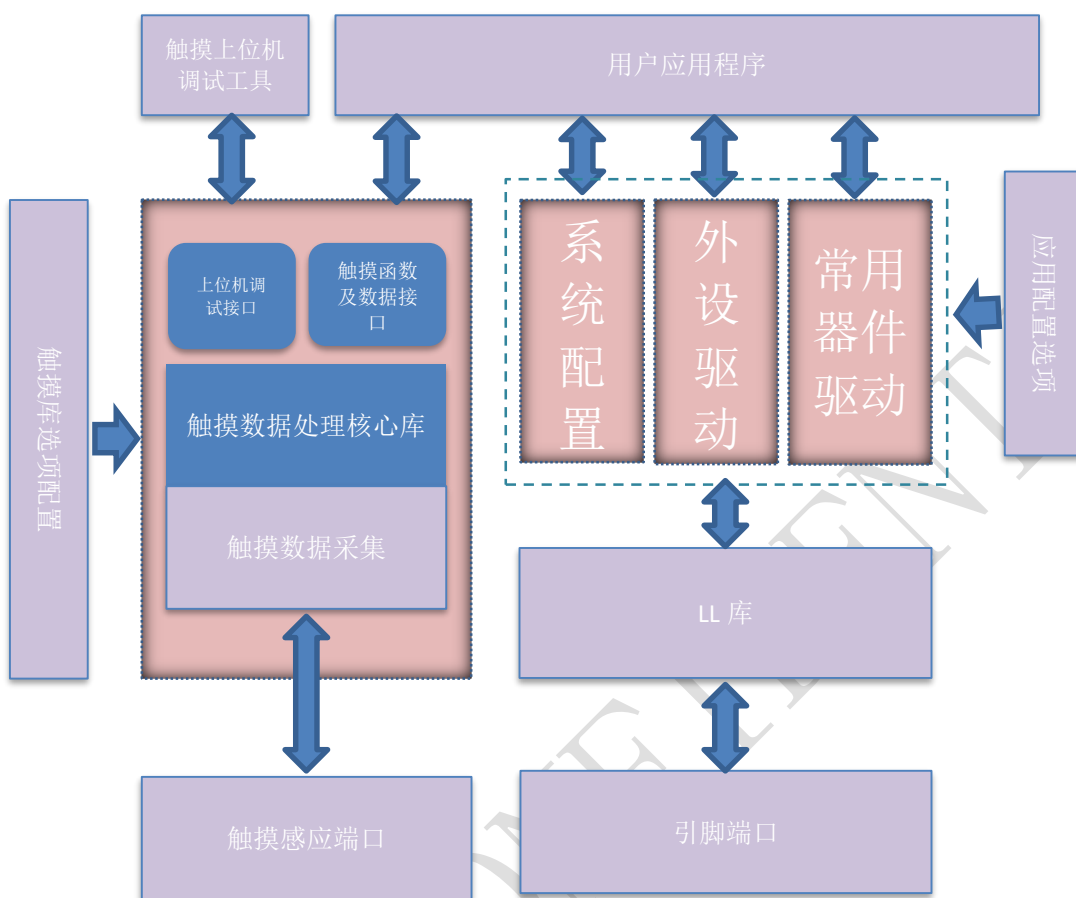
1. 简介

PY32T 标准软件库 (PY32T020_Touch_Library_Vxx) 集合了系统配置、触摸库、外设库 (LL 库)、基于 LL 库的应用范例和常用的外部器件驱动。此工程采用可视化配置界面, 可以通过勾选选项及填写参数的方式来完成系统配置、触摸、外设及外部器件的初始化及参数配置。在此工程基础上进行应用开发, 用户可无需关注与芯片层面相关的系统、外设、TK 的初始化过程, 只需调用相应模块的函数及数据接口即可实现芯片相应功能的使用, 从而可以使用户只需专注与产品功能相关的应用开发, 应用开发过程更加直观与高效, 不熟悉普冉 mcu 的开发者也可轻松入门及上手。

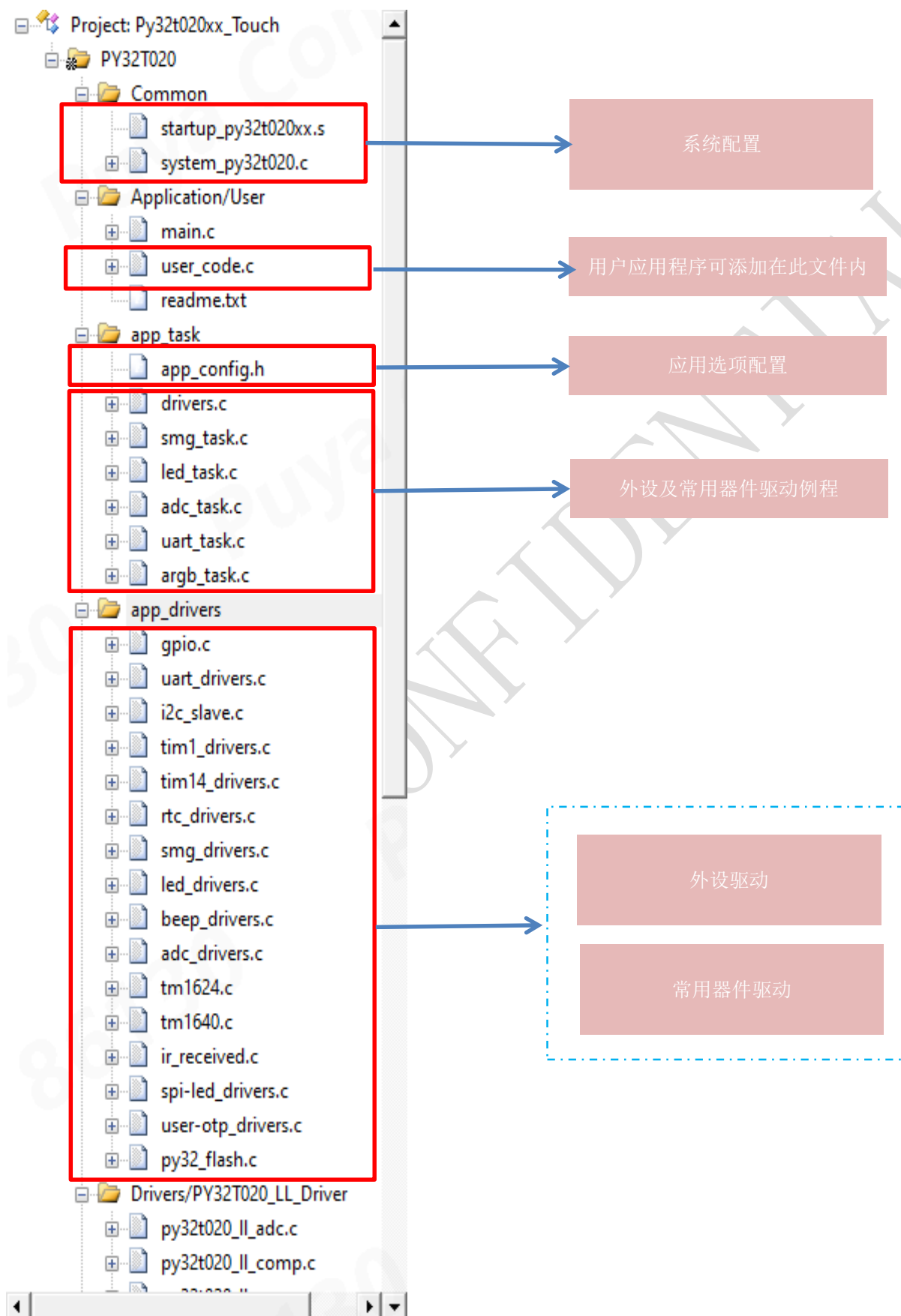
本文档将详细介绍此工程各功能模块的定义及使用方法, 使开发者对基于标准工程的应用开发过程有全面的了解。

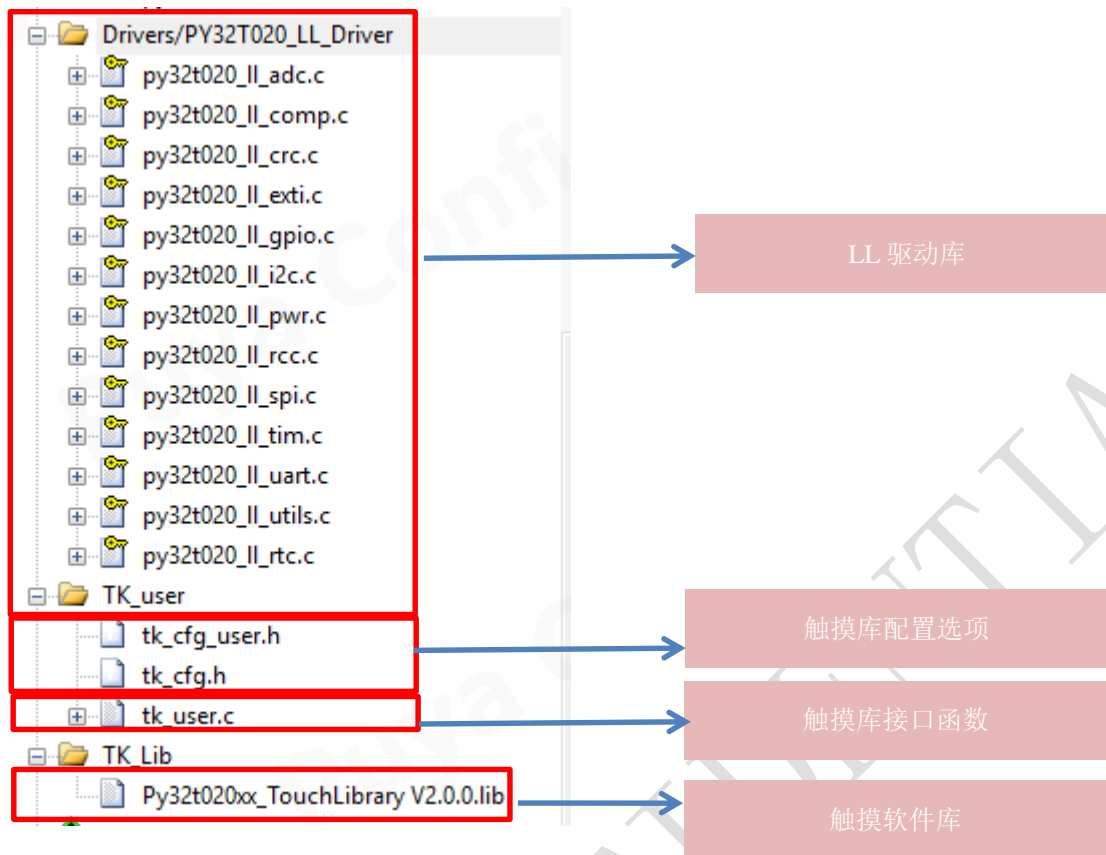
2. 软件库结构

软件库的基本结构如下图所示：



以上软件结构图主要模块与软件工程目录对应关系如下：





3. 触摸库介绍与使用

3.1 触摸功能类别介绍

触摸库共支持以下应用：

- 常规按键（感应器件为弹簧、PAD、导电泡棉等）
- 防水按键
- 滑轮/滑条
- 隔空触控
- 触摸检水
- 低功耗应用

3.2 MCU 资源需求

3.2.1 MCU 硬件资源

PY32 触摸库需使用以下硬件资源：

- SYS TICK 定时器（非独占）
- 每个触摸通道一个通用 IO
- 防水 Shield 通道需占一个通道 IO
- 低功耗模式下需用到 RTC

3.2.2 MCU 软件资源

PY32 触摸库占用 FLASH 和 SRAM 资源如下表：

触摸功能	FLASH	SRAM	通道	备注
常规按键	5.83K	1.15K	8(Key)	每增加一个按键 SRAM 约增加 30 字节
隔空按键	6.47K	1.27K	6(Key)	
防水按键	7.08K	1.38K	9(Key)+1(Shield)	
滑轮+滑条+按键	8.00K	1.29K	2(Key)+4(Wheel)+4(Slider)	
滑条+滑轮	6.96K	1.23K	4(Wheel)+4(Slider)	
触摸检水	5.79K	1.06K	1(Ref)+1(Detect)	

注：上表中 SRAM 占用包括堆栈 512Byte

PY32 触摸库采样及处理所需的时间如下表：

触摸功能	扫描方式	主频 (MHz)	通道	窗口设置	单通道扫描时间 (us)	TK 中断响应时间 (us)	TK 数据处理时间 (us)
按键	常规扫描	24	8	12000	1000	195	150
		48			1000	135	105
	同步扫描	24	6	2400*20	4000	3148	143
		48			4000	2245	85
防水按键	常规扫描	24	8+1	6000	750	395	295
		48				277	205
滑轮/滑条	常规扫描	24	4	6000	500	92	102
		48				63	71
检水	常规扫描	24	2	12000	500	15	73
		48				10	52

说明：

1. TK 采样时间与窗口设置值成正比，与其他因素无关。
2. 在主频确定的前提下，TK 中断时间与 TK 数据处理时间基本与通道数量成正比。
3. 以按键常规扫描、主频 24M 为例，8 通道采样时间共为： $8 * 1000us = 8000us$ ，中断响应+数据处理时间共为： $195 + 150 = 345us$ ，由于 TK 采样并不占用 CPU 时间，所以 TK 任务只占 CPU 时间的比例为： $345/8000 = 4.3\%$ 。

3.3 触摸库接口解析

3.3.1 数据接口

应用类别	变量名	变量类型	说明
常规按键	TKCtr.KeyFlags	unsigned int	TKCtr.KeyFlags 为 32 位无符号数，其每个 bit 表示一个触摸按键的触发状态，bit0~bit25 分别对应 KEY0~KEY25，用户程序可判断相应位的值来获取按键状态，并执行相应操作。
防水按键			
隔空按键			
滑条/滑轮	TKCtr.SliderOrWheelPosition[n]	signed short int	TKCtr.SliderOrWheelPosition[n] 表示第 n 个滑条/滑轮的位置信息，当滑条/滑轮被触发时，位置范围为：0~设定的精度，当 TKCtr.SliderOrWheelPosition[n] 值为-1 时，表示滑条/滑轮无触发。
触摸检水	TKCtr.DetectOut	unsigned char	TKCtr.DetectOut 为 8 位无符号数，其每个 bit 表示一个检水通道的水位状态，当为 1 时，表示相应通道为有水。

3.2.2 函数接口

函数名	输入参数	返回值	描述
TK_Init	无	无	触摸功能初始化 API 函数，在进入主循环之前调用。
TK_MainFsm	无	无	触摸功能主状态机 API 函数，在主循环中调用。
TK_TimerHandler	uint8_t ms, ms 表示调用 该函数的时间 间隔，单位为 毫秒	无	该函数在定时器中断服务程序中调用，可为触摸功能提供 时基。

```

int main(void)
{
    SystemClockConfig();
    /* 系统配置初始化代码开始*/
    app_drivers_init();
    /* 系统配置初始化代码结束*/
    /* 触摸初始化开始 */
    #if APP_TK_ENABLE
        TK_Init();
    #endif
    /* 触摸初始化结束 */
    /* 用户初始化代码开始*/
    user_init();
    /* 用户初始化代码结束*/
    log_printf("start loop\r\n");
    while (1)
    {
        #if APP_TK_ENABLE
            /* 触摸状态机处理 */
            TK_MainFsm();
            /* 触摸按键处理 */
            TK_Loop();
        #endif
        app_drivers_loop();
        user_loop();
    }
}

```

```

void SysTick_Handler(void)
{
    static uint16_t Prescaler;
    #if (SYSTICK_DEBUG_GPIO != NO_PIN && SYSTICK_DEBUG)
        GPIO_ToggleBit(SYSTICK_DEBUG_GPIO);
    #endif
    Prescaler++;
    if (Prescaler >= (1000 / SYSTICK_TIMING_TIME))
    {
        Prescaler = 0;
        app_drivers_timer();
    }
    #if APP_TK_ENABLE
        TK_TimerHandler(1);
    #endif
    #if APP_BEEP_ENABLE
        BEEP_Toggle();
    #endif
    #if (APP_IR_RECEIVED_ENABLE == 1 && D_IR_TIM == 0)
        IR_Received_Scan();
    #endif
    #if APP_LED_ENABLE
        LED_Scan();
    #endif
    user_timer();
}

```

3.4 触摸库功能选项及参数配置

3.4.1 触摸按键用户配置

对于触摸按键应用，首先应配置触摸通道和按键触发门限值，此两项配置在 `tk_cfg_user.h` 中实现，如下图所示：

```
#ifndef _TK_CFG_USER_H
#define _TK_CFG_USER_H

//按键触摸通道定义，按KEY的顺序填写，未用到的按键必须定义为TK_CH_NONE
#define CH_KEY0    TK_CH9
#define CH_KEY1    TK_CH5
#define CH_KEY2    TK_CH1
#define CH_KEY3    TK_CH8
#define CH_KEY4    TK_CH3
#define CH_KEY5    TK_CH2
#define CH_KEY6    TK_CH17
#define CH_KEY7    TK_CH_NONE
#define CH_KEY8    TK_CH_NONE
#define CH_KEY9    TK_CH_NONE
#define CH_KEY10   TK_CH_NONE
#define CH_KEY11   TK_CH_NONE
#define CH_KEY12   TK_CH_NONE
#define CH_KEY13   TK_CH_NONE
#define CH_KEY14   TK_CH_NONE
#define CH_KEY15   TK_CH_NONE
#define CH_KEY16   TK_CH_NONE
#define CH_KEY17   TK_CH_NONE
#define CH_KEY18   TK_CH_NONE
#define CH_KEY19   TK_CH_NONE
#define CH_KEY20   TK_CH_NONE
#define CH_KEY21   TK_CH_NONE
#define CH_KEY22   TK_CH_NONE
#define CH_KEY23   TK_CH_NONE
#define CH_KEY24   TK_CH_NONE
#define CH_KEY25   TK_CH_NONE

//-----

//-----

//按键触发门限值定义
#define KEY0_THD    80
#define KEY1_THD    80
#define KEY2_THD    80
#define KEY3_THD    80
#define KEY4_THD    80
#define KEY5_THD    80
#define KEY6_THD    80
#define KEY7_THD    80
#define KEY8_THD    40
#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

//=====
```

CH_KEYn 与 KEYn_THD 相对应，CH_KEYn 为 KEYn 对应的通道定义，KEYn_THD 为 KEYn 的触发门限值。当 CH_KEYn 定义的通道数据变化量大于于 KEYn_THD 时，KEYn 将被触发。

3.4.2 滑条/滑轮用户配置

滑条/滑轮应用需配置应用类型、通道、分辨率及触发门限值，在 tk_cfg_user.h 中进行定义，如下图所示：

The diagram illustrates the configuration of sliders/wheels in the `tk_cfg_user.h` file. It shows three sections of code, each representing a different slider/wheel configuration. Red boxes highlight specific parameters, and red arrows point from these boxes to explanatory text boxes on the right.

Section 1: Slider/Wheel 0 Configuration

```

//=====
#define SLIDER_OR_WHEEL0_TYPE      TK_APP_WHEEL      //滑条类型
#define SLIDER_OR_WHEEL0_RESOLUTION 360             //滑条分辨率
#define SLIDER_OR_WHEEL0_THD      101               //滑条门限值
#define SLIDER_OR_WHEEL0_CH0      TK_CH0            //滑条通道，按顺序填写
#define SLIDER_OR_WHEEL0_CH1      TK_CH6
#define SLIDER_OR_WHEEL0_CH2      TK_CH2
#define SLIDER_OR_WHEEL0_CH3      TK_CH1
#define SLIDER_OR_WHEEL0_CH4      TK_CH_NONE
#define SLIDER_OR_WHEEL0_CH5      TK_CH_NONE
#define SLIDER_OR_WHEEL0_CH6      TK_CH_NONE
#define SLIDER_OR_WHEEL0_CH7      TK_CH_NONE

```

Section 2: Slider/Wheel 1 Configuration

```

#define SLIDER_OR_WHEEL1_TYPE      TK_APP_SLIDER
#define SLIDER_OR_WHEEL1_RESOLUTION 255
#define SLIDER_OR_WHEEL1_THD      102
#define SLIDER_OR_WHEEL1_CH0      TK_CH19
#define SLIDER_OR_WHEEL1_CH1      TK_CH23
#define SLIDER_OR_WHEEL1_CH2      TK_CH24
#define SLIDER_OR_WHEEL1_CH3      TK_CH25
#define SLIDER_OR_WHEEL1_CH4      TK_CH_NONE
#define SLIDER_OR_WHEEL1_CH5      TK_CH_NONE
#define SLIDER_OR_WHEEL1_CH6      TK_CH_NONE
#define SLIDER_OR_WHEEL1_CH7      TK_CH_NONE

```

Section 3: Slider/Wheel 2 Configuration

```

#define SLIDER_OR_WHEEL2_TYPE      TK_APP_NONE
#define SLIDER_OR_WHEEL2_RESOLUTION 255
#define SLIDER_OR_WHEEL2_THD      80
#define SLIDER_OR_WHEEL2_CH0      TK_CH_NONE
#define SLIDER_OR_WHEEL2_CH1      TK_CH_NONE
#define SLIDER_OR_WHEEL2_CH2      TK_CH_NONE
#define SLIDER_OR_WHEEL2_CH3      TK_CH_NONE
#define SLIDER_OR_WHEEL2_CH4      TK_CH_NONE
#define SLIDER_OR_WHEEL2_CH5      TK_CH_NONE
#define SLIDER_OR_WHEEL2_CH6      TK_CH_NONE
#define SLIDER_OR_WHEEL2_CH7      TK_CH_NONE

```

Section 4: Slider/Wheel 3 Configuration

```

#define SLIDER_OR_WHEEL3_TYPE      TK_APP_NONE
#define SLIDER_OR_WHEEL3_RESOLUTION 255
#define SLIDER_OR_WHEEL3_THD      80
#define SLIDER_OR_WHEEL3_CH0      TK_CH_NONE
#define SLIDER_OR_WHEEL3_CH1      TK_CH_NONE
#define SLIDER_OR_WHEEL3_CH2      TK_CH_NONE
#define SLIDER_OR_WHEEL3_CH3      TK_CH_NONE
#define SLIDER_OR_WHEEL3_CH4      TK_CH_NONE
#define SLIDER_OR_WHEEL3_CH5      TK_CH_NONE
#define SLIDER_OR_WHEEL3_CH6      TK_CH_NONE
#define SLIDER_OR_WHEEL3_CH7      TK_CH_NONE
//=====

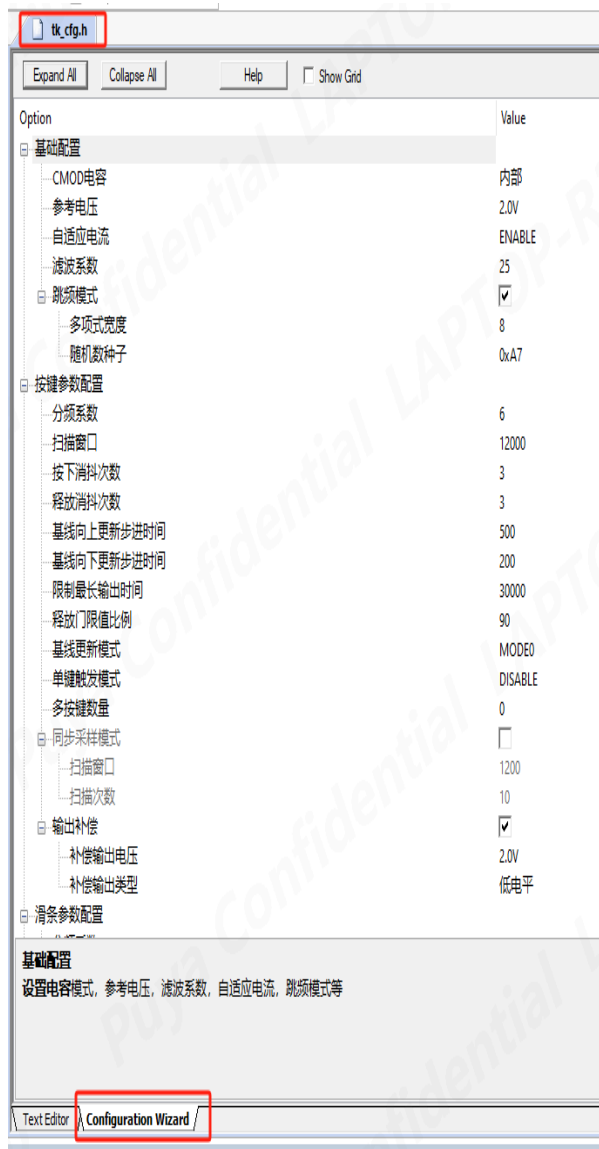
```

Annotations:

- 滑条/滑轮类型定义**：滑条定义为 `TK_APP_SLIDER`，滑轮定义为 `TK_APP_WHEEL`，如定义为 `TK_APP_NONE`，表示当前滑条/滑轮不使能，本触摸软件库最多支持 4 个滑条/滑轮。
注意：滑条/滑轮必须按 0-3 的顺序来定义，例如：如使能两个滑条/滑轮，只能定义 `SLIDER_OR_WHEEL0/1`，而不能定义 `SLIDER_OR_WHEEL0/2`。
- 滑条/滑轮分辨率定义**：如定义为 360，则输出位置范围为：0 ~ 359。
- 滑条/滑轮触发门限值定义**：如果该滑条/滑轮其中两个通道变化量之和大于此门限值，该滑条/滑轮将被触发。
- 滑条/滑轮通道定义**：每个滑条/滑轮最多支持 8 个通道，必须按顺序填写。

3.4.3 触摸功能及参数配置

触摸参数及功能选择可在 tk_cfg.h 中进行定义，实际应用时，并不需要直接编辑 tk_cfg.h 的内容，而是在其 configuration wizard 中进行配置，configuration wizard 是图形化的配置菜单，可以通过勾选、下拉菜单选择或输入参数的方式完成相应配置。



3.5 用户可修改的触摸库回调函数

触摸软件库以回调函数的形式，开放了一些函数接口给用户进行修改，以适应更多的应用要求。这些回调函数存放在 tk_user.c 文件中，常用的回调函数如下表：

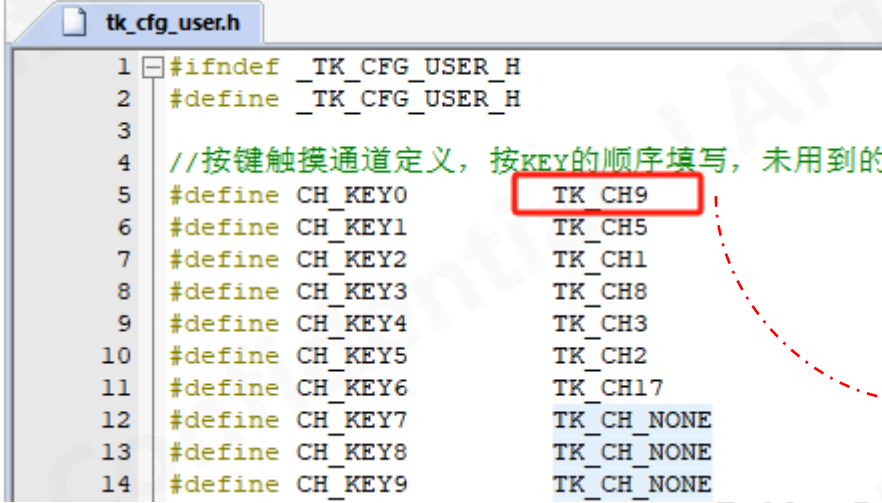
函数名	输入参数	返回值	描述
EnterStop_Callback	无	无	在触摸进入低功耗模式之前调用，用户程序如有需在进入低功耗之前进行的操作，可将相应代码添加在此函数内。
ExitStop_Callback	无	无	在触摸退出低功耗模式之后调用，用户程序如有需在退出低功耗之后进行的操作，可将相应代码添加在此函数内。
LoopStop_Callback	无	无	低功耗模式下采集完数据唤醒后调用，用户程序如有需低功耗下定时处理的任务，可将相应代码添加在此函数内。
TK_RawDataCompensate	参数名： chs 数据类型： uint8_t 参数含义： 通道索引	返回值： 补偿后的触摸数据	用于对触摸数据进行补偿，例如：在 LED 灯亮灭变化时，触摸数据可能会同步变化，在此情况下，可在此函数内判断 LED 的状态对触摸数据同步进行补偿。
FilterExDeal	参数名： chs 数据类型： uint8_t 参数含义： 通道索引	无	此函数在对触摸数据滤波时进行回调，可在此函数内根据实测情况对滤波参数进行调整。
APP_TouchKeyFlagsMask	无	无	此函数用于对异常状态进行判断及处理，例如：产生按键后，判断此时是否存在干扰，如有干扰，则把产生的按键丢弃。
TK_ScanDone	无	返回值： 是否需对触摸模式切换及重新初始化	此函数在触摸数据采集完成后调用，用于获取是否需重新对触摸模式切换的信息。
PrintfTask	无	无	打印接口函数，此函数在触摸数据采集完并进行数据处理后调用，用户如需 UART 打印触摸数据及状态，可在此函数内添加相关代码。

3.6 触摸应用配置步骤及流程

3.6.1 常规按键应用

Step 1: 设置按键对应的 TK 通道，有两种方法：

方法一：在 tk_cfg_user.h 直接按顺序填写 TK 通道，如下：




```

1 #ifndef TK_CFG_USER_H
2 #define TK_CFG_USER_H
3
4 //按键触摸通道定义，按KEY的顺序填写，未用到的
5 #define CH_KEY0 TK_CH9
6 #define CH_KEY1 TK_CH5
7 #define CH_KEY2 TK_CH1
8 #define CH_KEY3 TK_CH8
9 #define CH_KEY4 TK_CH3
10 #define CH_KEY5 TK_CH2
11 #define CH_KEY6 TK_CH17
12 #define CH_KEY7 TK_CH_NONE
13 #define CH_KEY8 TK_CH_NONE
14 #define CH_KEY9 TK_CH_NONE
  
```

19	19	19	21	28	-	1	-	PB2	IO	COM_C	-	SPI_MISO	CMP1_INP TK_IN9
												UART3_RX	
												TM14_CH1	

方法二：在触摸调试上位机 PY Touch - Link.exe 中配置 TK 通道，然后导出 tk_cfg_user.h 覆盖原文件。



① 选择芯片型号

② 按顺序在选择的 TK 通道引脚处鼠标双击，可选定相应按键的通道

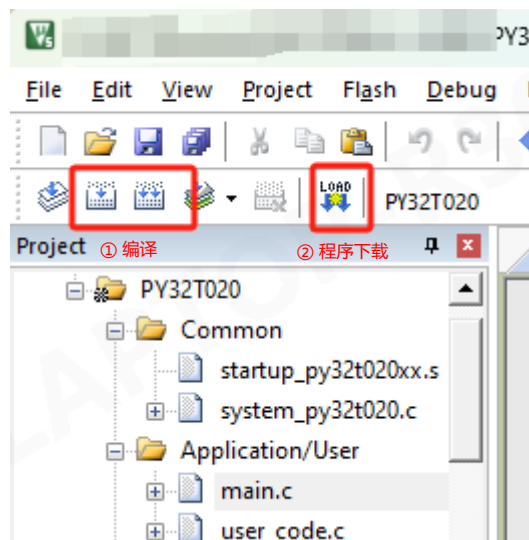
③ 双击对应按键的门限值单元格，设置初始门限值

④ 保存 tk_cfg_user.h，覆盖原文件

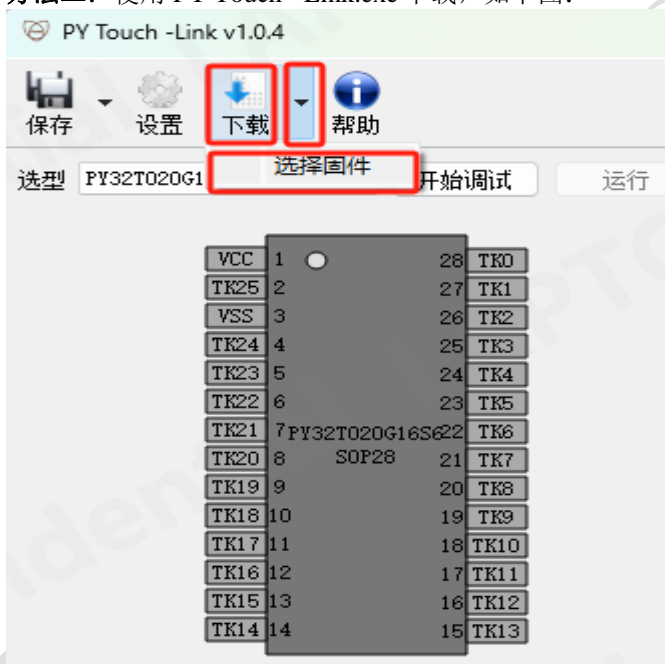
按键序号	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6
通道	TK9	TK5	TK1	TK8	TK3	TK2	TK17
基准值	0	0	0	0	0	0	0
原始值	0	0	0	0	0	0	0
差值	0	0	0	0	0	0	0
门限值	50	50	50	50	50	50	50
按键次数	0	0	0	0	0	0	0
状态							

Step2: 程序编译，编译完后下载到目标板，下载程序有两种方法：

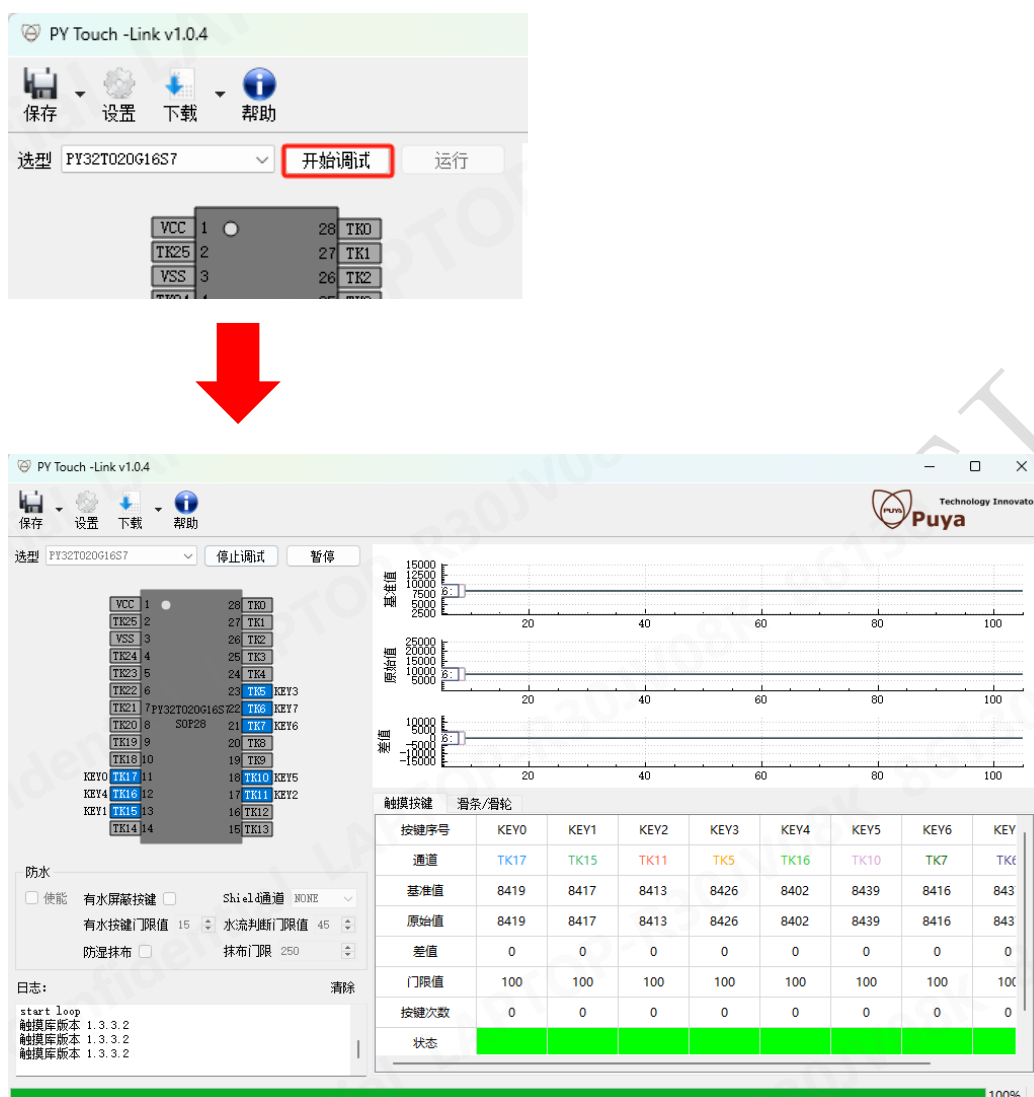
方法一：在 KEIL 环境内直接下载，如下图：（注意：此种下载方法需保持下载引脚为 SWD 端口功能）



方法二：使用 PY Touch - Link.exe 下载，如下图：



Step3: 连接上位机调试工具 PY Touch - Link, 进入触摸调试模式。



Step4: 用手指（或铜柱）正常触摸按键，查看触摸数据变化量，取变化量的 60%~70%设为触发门限值。

触摸按键	滑条/滑轮	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7
按键序号		KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7
通道		TK17	TK15	TK11	TK5	TK16	TK10	TK7	TK6
基准值		8424	8419	8413	8427	8411	8440	8416	8440
原始值		8608	8419	8413	8427	8411	8440	8416	8440
差值		184	0	0	0	0	0	0	0
门限值		100	100	100	100	100	100	100	100
按键次数		35	0	0	0	0	0	15	0
状态									

例：如上图，手指正常触摸 KEY0 时，变化量约为 184，门限值可设为： $184 * 70\% = 128$ ，设置方法如下：

① 点击“暂停”（暂停后才可以更改门限值）

③ 设置好门限值后，点击“运行”

② 双击 KEY0 对应的门限值单元格，手动填入设定值

按键序号	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7
通道	TK17	TK15	TK11	TK5	TK16	TK10	TK7	TK6
基准值	8417	8417	8412	8425	8404	8440	8415	8439
原始值	8417	8417	8412	8425	8404	8440	8416	8439
差值	0	0	0	0	0	0	1	0
门限值	128	100	100	100	100	100	100	100
按键次数	123	0	0	0	0	0	15	0
状态	绿色	绿色	绿色	绿色	绿色	绿色	绿色	绿色

Step5: 按 Step4 步骤设置完所有的按键后，导出 tk_cfg_user.h，覆盖原文件。

另存为...

文件类

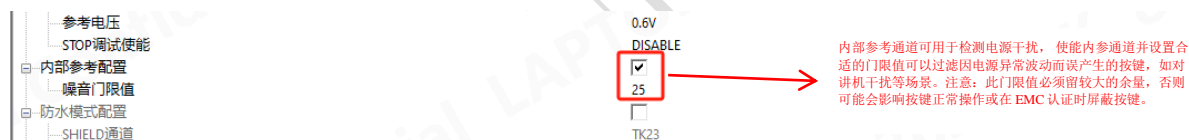
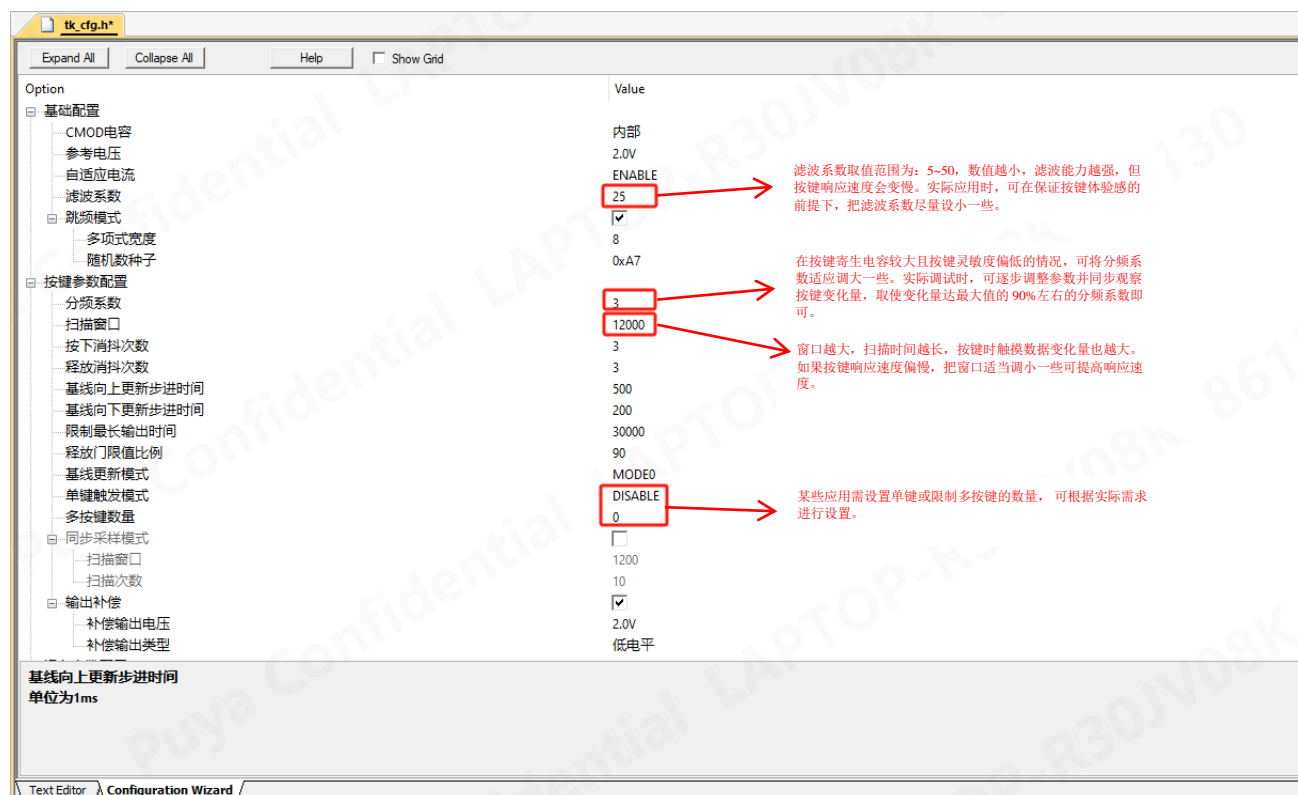
名称	修改日期	类型
main	2024/7/22 14:15	H 文件
tk_cfg	2025/1/23 11:52	H 文件
tk_cfg_user	2024/8/9 16:49	H 文件
user_code	2024/8/12 15:20	H 文件

tk_cfg_user

配置文件(*.h)

保存(S) 取消

Step6（可选）：在 tk_cfg.h 中修改配置参数，如图，红框内参数可根据实际情况适当调整，其余参数一般情况下保持默认设置即可。

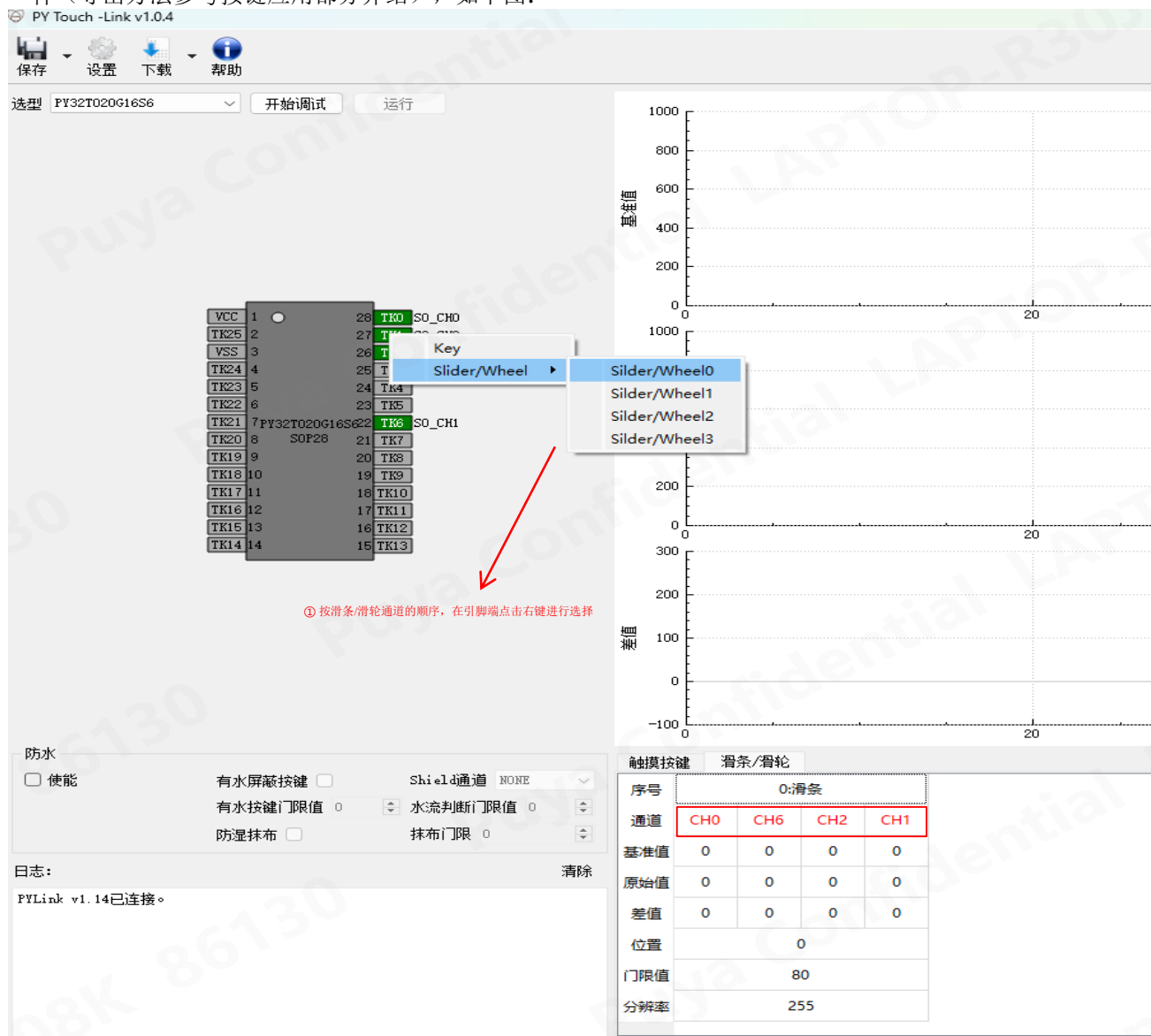


Step7：至此，常规按键的配置已完成。有按键产生时，TKCtr.KeyFlags 的相应位会被置位，用户程序可判断按键对应的标志位并执行相应的操作。

3.6.2 滑轮/滑条应用

Step1: 配置滑轮/滑条通道、类型、分辨率，有两种方法可实现：

方法一：在 PY Touch - Link 中选择通道、设置滑轮/滑条类型、分辨率，设置完成后导出 tk_cfg_user.h 覆盖原文件（导出方法参考按键应用部分介绍），如下图：



方法二：在 tk_cfg_user.h 中直接修改，如下图：

```
//=====
#define SLIDER_OR_WHEEL0_TYPE      TK_APP_WHEEL      //滑条类型
#define SLIDER_OR_WHEEL0_RESOLUTION 360              //滑条分辨率
#define SLIDER_OR_WHEEL0_THD      101                //滑条门限值
#define SLIDER_OR_WHEEL0_CH0      TK_CH0              //滑条通道，按顺序填写
#define SLIDER_OR_WHEEL0_CH1      TK_CH6
#define SLIDER_OR_WHEEL0_CH2      TK_CH2
#define SLIDER_OR_WHEEL0_CH3      TK_CH1
#define SLIDER_OR_WHEEL0_CH4      TK_CH_NONE
#define SLIDER_OR_WHEEL0_CH5      TK_CH_NONE
#define SLIDER_OR_WHEEL0_CH6      TK_CH_NONE
#define SLIDER_OR_WHEEL0_CH7      TK_CH_NONE
```

Step2: 滑条/滑轮基础配置完成后，程序编译、下载到目标板，连接 PY Touch - Link 进行调试，并设置合适的触发门限值。

触摸按键	滑条/滑轮								触摸按键	滑条/滑轮							
序号	0:滑轮				1:滑条				序号	0:滑轮				1:滑条			
通道	CH0	CH6	CH2	CH1	CH19	CH23	CH24	CH25	通道	CH0	CH6	CH2	CH1	CH19	CH23	CH24	CH25
基准值	8029	8346	8446	7727	8450	8443	8429	8422	基准值	8028	8347	8421	7687	8451	8441	8429	8420
原始值	8027	8351	8603	7998	8451	8443	8429	8422	原始值	8028	8347	8422	7687	8451	8441	8429	8420
差值	-2	5	157	271	1	0	0	0	差值	0	0	1	0	0	0	0	0
位置	236				-1				位置	-1				-1			
门限值	100				100				门限值	100				100			
分辨率	360				255				分辨率	360				255			

① 手指正常在滑条/滑轮上滑动，观察每个通道变化量。

② 双击进入门限值设置界面，反复修改门限值直至达到最佳的操作体验感。

```
3 //=====
4
5 #define SLIDER_OR_WHEEL0_TYPE      TK_APP_WHEEL      //滑条类型
6 #define SLIDER_OR_WHEEL0_RESOLUTION 360              //滑条分辨率
7 #define SLIDER_OR_WHEEL0_THD      100                //滑条门限值
8 #define SLIDER_OR_WHEEL0_CH0      TK_CH0              //滑条通道，按顺序填写
9 #define SLIDER_OR_WHEEL0_CH1      TK_CH6
10 #define SLIDER_OR_WHEEL0_CH2      TK_CH2
11 #define SLIDER_OR_WHEEL0_CH3      TK_CH1
12 #define SLIDER_OR_WHEEL0_CH4      TK_CH_NONE
13 #define SLIDER_OR_WHEEL0_CH5      TK_CH_NONE
14 #define SLIDER_OR_WHEEL0_CH6      TK_CH_NONE
15 #define SLIDER_OR_WHEEL0_CH7      TK_CH_NONE
```

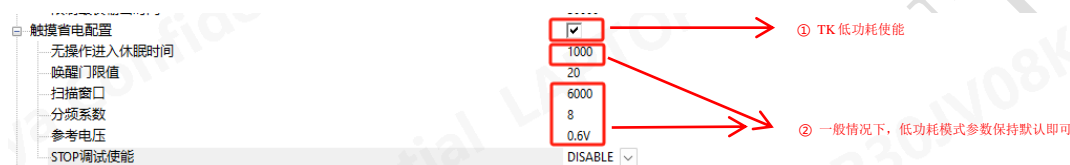
③ 最后确定的门限值数值填写到 tk_cfg_user.h

Step3: 至此, 滑条/滑轮配置完成, 产生的位置信息存放于 TKCtr.SliderOrWheelPosition[n], 用户程序可读取该数值并执行相应操作。

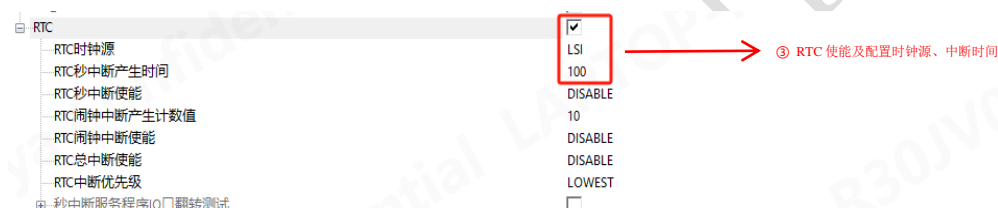
```
if (TKCtr.SliderOrWheelPosition[0] != -1)
{
    //用户应用程序
}
if (TKCtr.SliderOrWheelPosition[1] != -1)
{
    //用户应用程序
}
```

3.6.3 触摸低功耗应用

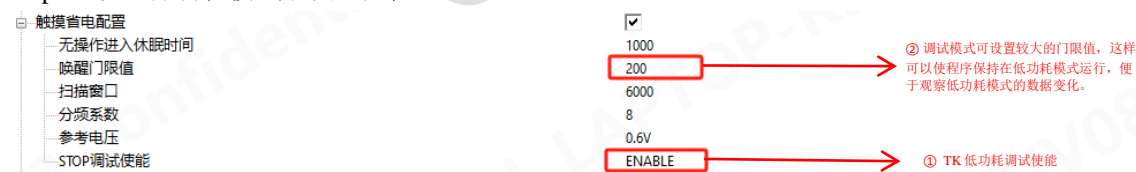
Step1: 在 tk_cfg.h 中使能触摸低功耗模式, 并配置基础参数。



由于触摸低功耗以 RTC 为时基, 所以也需先配置 RTC 使能, RTC 在 app_config.h 中进行配置。



Step2: 设置低功耗模式触发门限值。



Step3: TK 低功耗模式程序下载至目标板, 手指正常触摸按键或滑条, 观察数据变化量以确定门限值。

触摸按键			
按键序号	KEY0	KEY1	TK_COM...
通道	TK14	TK15	TK_COM...
基准值	3610	3607	5405
原始值	3610	3607	5457
差值	0	0	52
门限值	180	180	200
按键次数	0	0	0
状态			

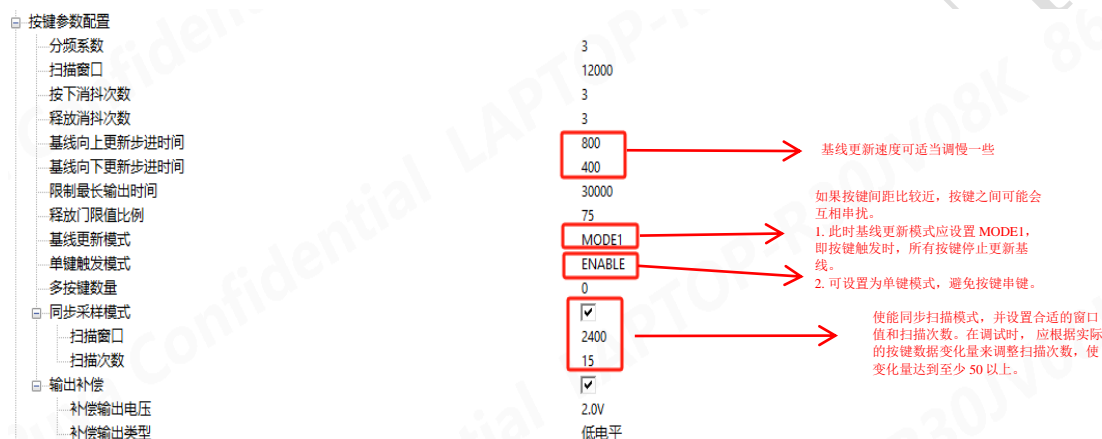
低功耗调试模式下手指触摸时的数据变化量, 门限值根据此变化量数值进行设定, 通常门限值取变化量的 50% 左右。注意: 每个按键的变化量不完全一样, 实际应以变化量最小的按键为参考。

Step4: 根据调试的数据设定门限值, 关闭调试模式, 重新下载程序进行测试, 根据实测的效果再微调门限值, 直至达到最佳体验感。



3.6.4 隔空按键应用

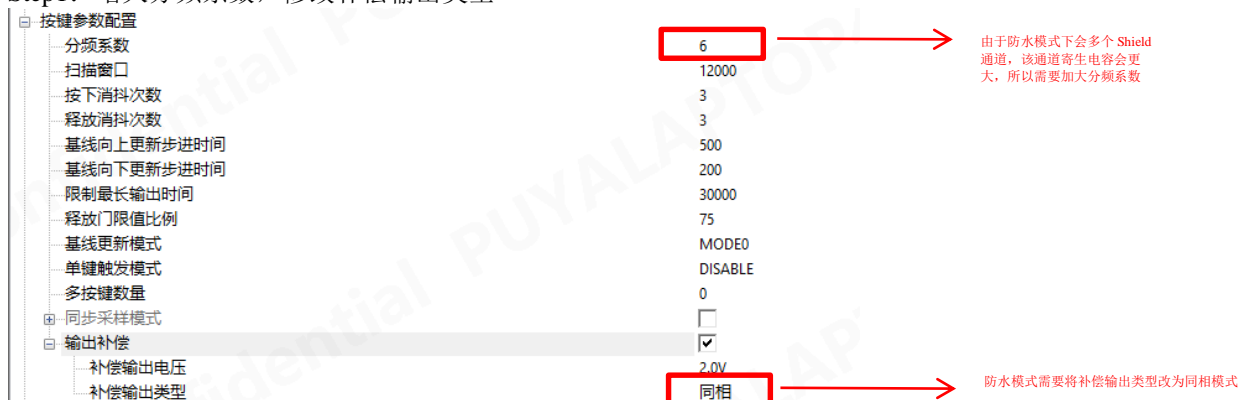
隔空按键应用的设置方法与常规按键类似, 不同的是隔空按键需使用同步扫描模式, 如下图:



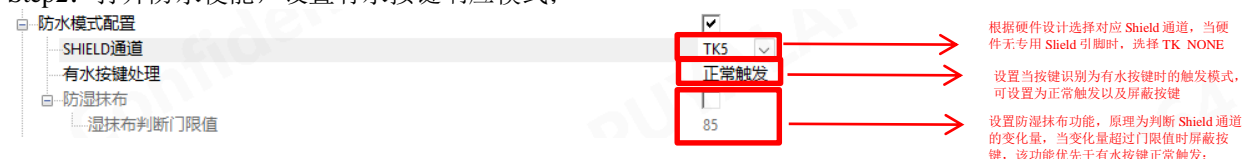
3.6.5 防水按键应用

防水按键应用的设置方法与常规按键类似, 不同的是防水按键需将输出补偿设置为同相模式, 如下图:

Step1: 增大分频系数, 修改补偿输出类型



Step2: 打开防水使能, 设置有水按键响应模式;



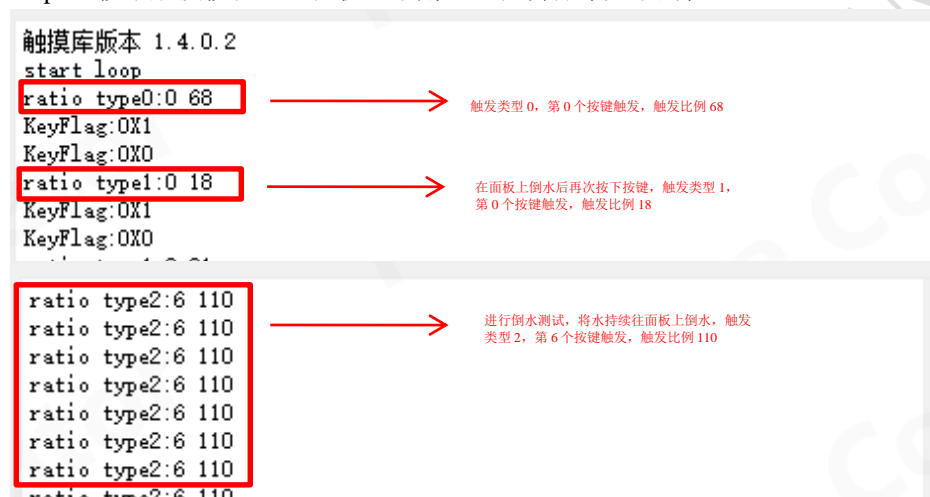
Step3: 在 app_config.h 中使能 PRINTF。



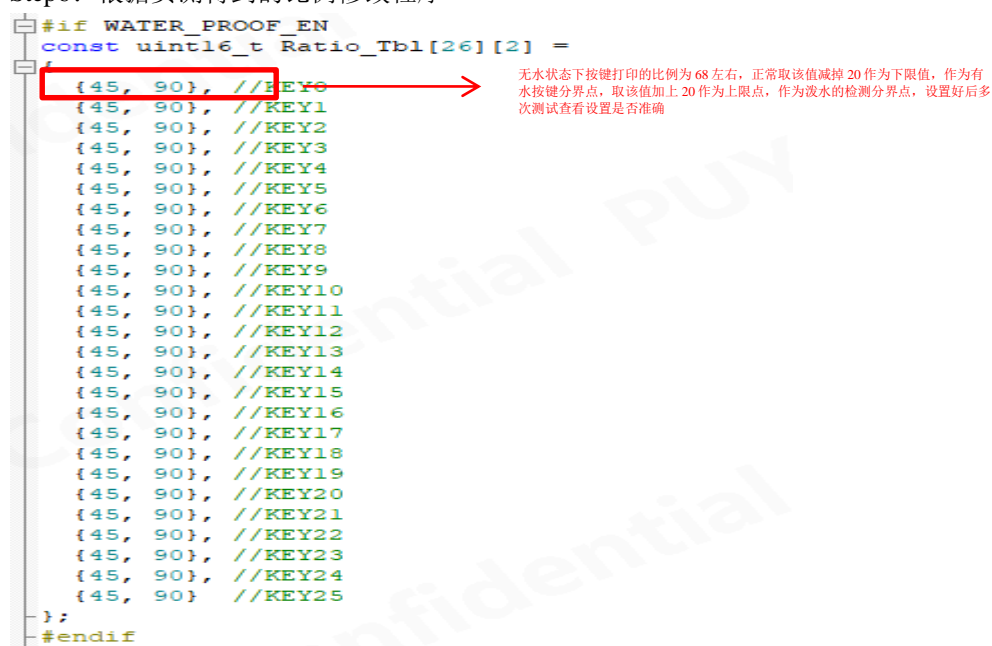
Step4: 装配好外壳后将程序下载到目标板, 连接 PY Touch - Link。可以看到触摸按键栏多了一个 TK_COM 的通道, 该通道代表 Shield 通道数据; 当需要防湿抹布功能时, 观察湿抹布动作时该通道的差值, 开启防湿抹布使能, 并将其填入湿抹布判断门限值中。

按键序号	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7	KEY8	KEY9	TK_COM...
通道	TK0	TK1	TK12	TK6	TK7	TK11	TK8	TK9	TK10	TK_COM...	
基准值	3616	3608	3608	3604	3618	3601	3613	3595	3614	3613	
原始值	3616	3608	3608	3604	3618	3601	3613	3595	3614	3613	
差值	0	0	0	0	0	0	0	0	0	0	
门限值	80	80	80	80	80	80	80	80	80	80	85
按键次数	0	0	0	0	0	0	0	0	0	0	0
状态											

Step5: 按下触摸按键, 上位机左下角日志栏会打印如下的信息

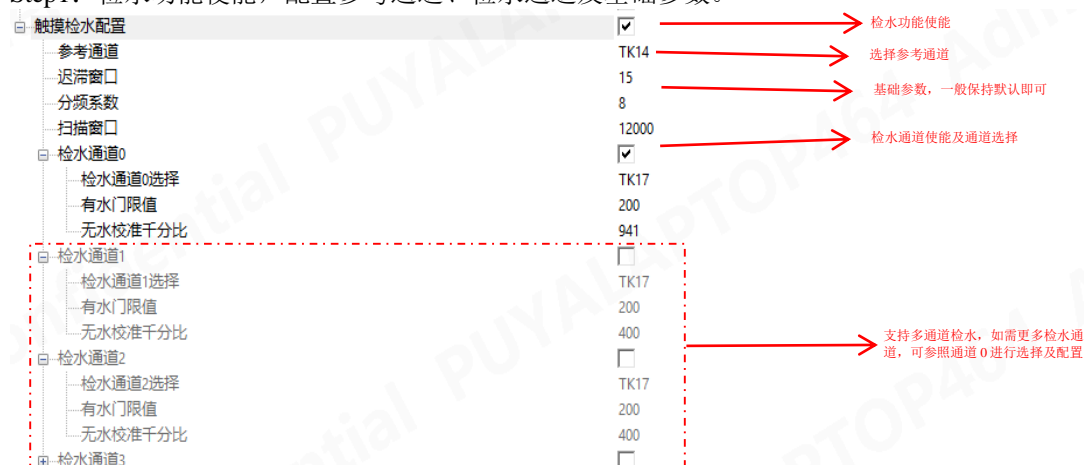


Step6: 根据实测得到的比例修改程序



3.6.6 触摸检水应用

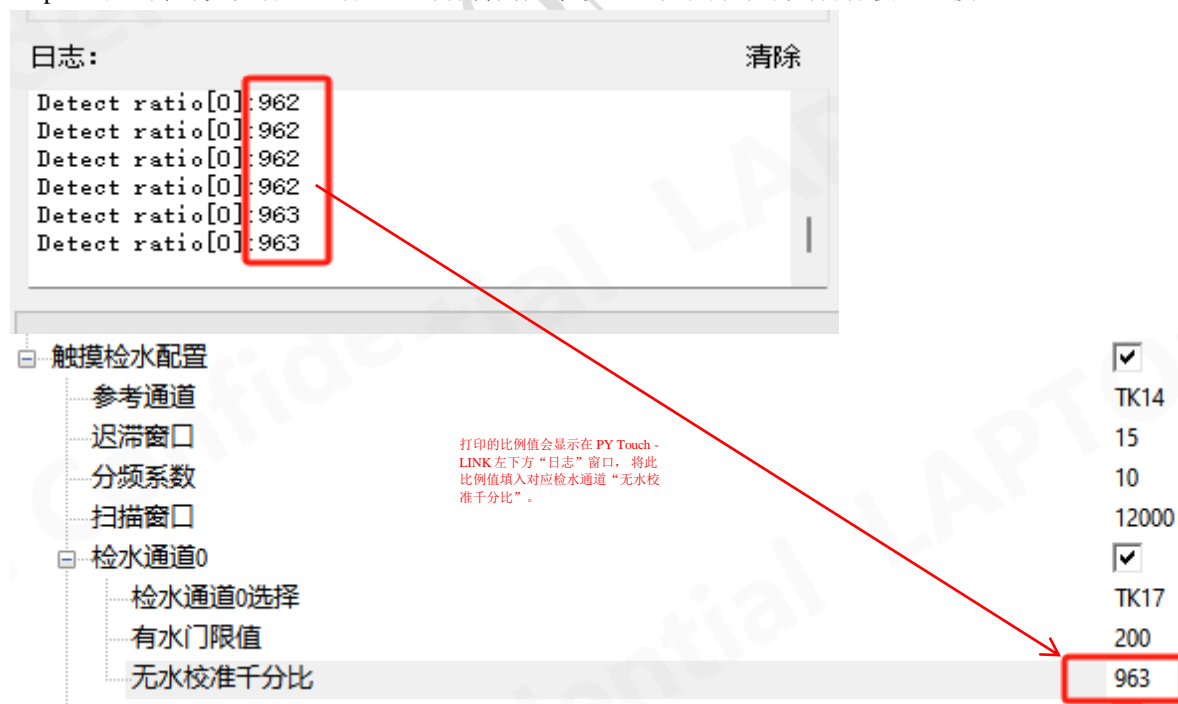
Step1: 检水功能使能，配置参考通道、检水通道及基础参数。



Step2: 在 app_config.h 中使能 PRINTF。



Step3: 装配好外壳及容器，容器此时保持为无水状态，把程序下载到目标板，连接 PY Touch - Link。



Step4: 填写好无水状态的检准值后，重新编译下载程序。

触摸按键		滑条/滑轮	
按键序号	KEY0	KEY1	
通道	TK14	TK17	
基准值	6007	6007	
原始值	6007	6237	
差值	0	230	往容器倒入水，直到水位超过检水PAD位置，此时检水通道差值即为有水变化量，一般取有水变化量的约50%作为检水门限值。
门限值	0	200	
按键次数	0	9	
状态			

Step5: 取有水变化量约 50%填入“有水门限值”。

- 触摸检水配置
 - 参考通道
 - 迟滞窗口
 - 分频系数
 - 扫描窗口
 - 检水通道0
 - 检水通道0选择
 - 有水门限值
 - 无水校准千分比

- ☒ TK14
- 15
- 10
- 12000
- ☒ TK17
- 115
- 963

填写有水门限值: $230 \times 0.5 = 115$

Step6: 重新下载程序，反复测试并微调门限值，直至达到最好的检水效果。至此，检水功能调试完成，用户程序可根据检水状态去执行相应操作。

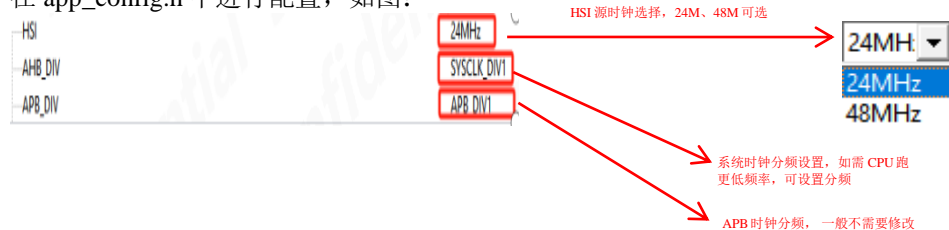
```
#if WATER_DETECT_EN
for (uint8_t n = 0; n < (WATER_DETECT_CNT - 1); n++)
{
    if(TKCtrl.DetectOut & (1 << n)) /* 检测到有水 */
    {
        //用户处理程序
    }
    else /* 无水 */
    {
        //用户处理程序
    }
}
#endif
```

4. 系统配置及外设应用解析

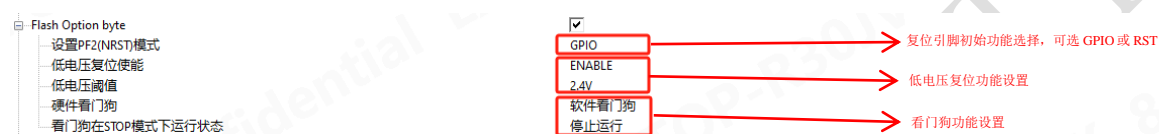
4.1 系统配置

4.1.1 时钟配置

在 app_config.h 中进行配置，如图：



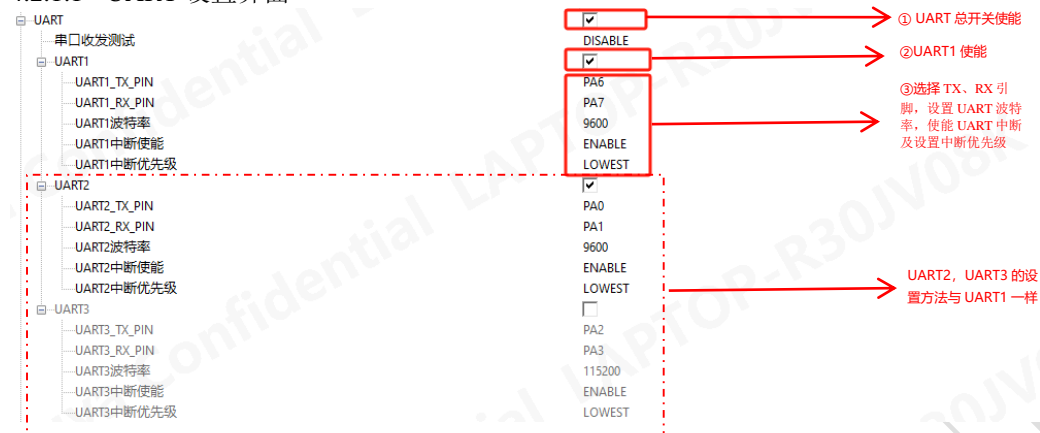
4.1.2 Option 选项



4.2 外设应用的接口与使用

4.2.1 UART

4.2.1.1 UART 设置界面

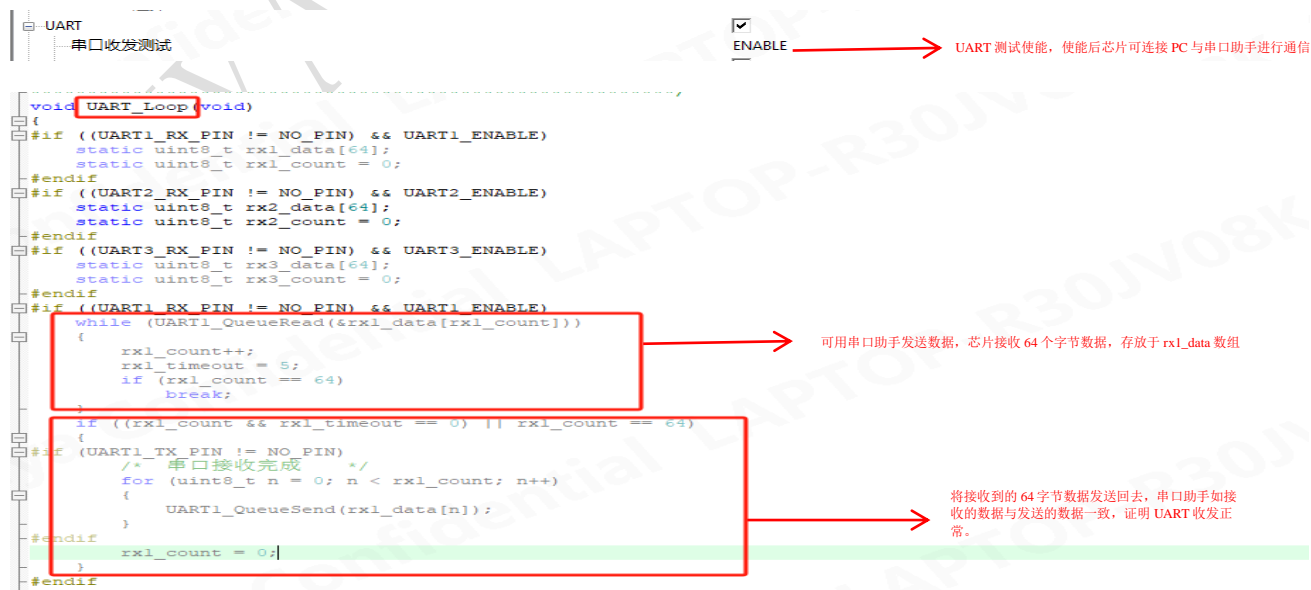


4.2.1.2 UART 函数接口说明

函数名	输入参数			返回值	描述
	参数名	数据类型	参数描述		
UART_QueueSend	*UARTx	UART_Type Def	指向 UARTx 寄存器组的指针	发送结果标志	调用该函数可实现 UARTx 一字节数据的发送
	*Queue	SqQueue	指向发送数据队形的指针		
	data	uint8_t	待发送的一字节数据		
UART_QueueRead	*Queue	SqQueue	UART 接收数据队列指针	接收队列是否为空标志	如接收队列不为空, 调用该函数可读取接收队列一字节数据
	*data	uint8_t	数据指针, 取接收队列的一字节数据赋给该指针指向的变量		

备注: 以上 UART 函数接口存放于 `uart_drivers.c`。

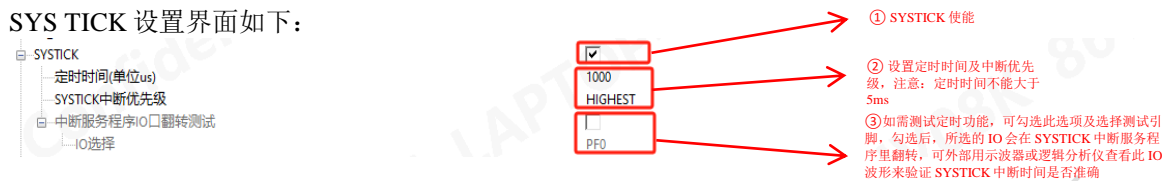
4.2.1.3 UART 应用范例



4.2.2 定时器

4.2.2.1 SYS TICK

SYSTICK 设置界面如下：



SYSTICK 中断服务程序位于 main.c，如下所示：

```

/*****
** 函数名   void SysTick_Handler(void)
** 描述    : 系统滴答定时器，目前定时时间为1ms
** 传入    : 无
** 返回    : 无
*****/
void SysTick_Handler(void)
{
    static uint16_t Prescaler;
    #if (SYSTICK_DEBUG_GPIO != NO_PIN && SYSTICK_DEBUG)
        GPIO_ToggleBit(SYSTICK_DEBUG_GPIO);
    #endif
    Prescaler++;
    if (Prescaler >= (1000 / SYSTICK_TIMING_TIME))
    {
        Prescaler = 0;
        app_drivers_timer();
    }
    #if APP_TK_ENABLE
        TK_TimerHandler(1);
    #endif
    #if APP_BEEP_ENABLE
        BEEP_Toggle();
    #endif
    #if (APP_IR_RECEIVED_ENABLE == 1 && D_IR_TIM == 0)
        IR_Received_Scan();
    #endif
    #if APP_LED_ENABLE
        LED_Scan();
    #endif
    user_timer();
}

```

用户程序如有需在 SYSTICK 定时中断服务执行的程序，可将程序放于 user_timer

4.2.2.2 TIM1、TIM14

TIMx (x=1、14) 定时设置界面如下：



备注：TIMx 定时设置与 SYSTICK 基本一致，可参考 SYSTICK 的相关描述。

TIMx 定时器中断服务程序分别位于 tim1_drivers.c、tim14_drivers.c，如下所示：

```

void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
    /* TIM Update event */
    if (LL_TIM_IsActiveFlag_UPDATE(TIM1) != 0)
    {
        LL_TIM_ClearFlag_UPDATE(TIM1);
        /* 测试GPIO翻转 */
        #if (TIM1_DEBUG_GPIO != NO_PIN && TIM1_DEBUG)
            GPIO_ToggleBit(TIM1_DEBUG_GPIO);
        #endif
        TIM1_PeriodElapsedCallback();
    }
}

```

```

void TIM14_IRQHandler(void)
{
    /* TIM Update event */
    if (LL_TIM_IsActiveFlag_UPDATE(TIM14) != 0)
    {
        LL_TIM_ClearFlag_UPDATE(TIM14);
        /* 测试GPIO翻转 */
        #if (TIM14_DEBUG_GPIO != NO_PIN && TIM14_DEBUG)
            GPIO_ToggleBit(TIM14_DEBUG_GPIO);
        #endif
        TIM14_PeriodElapsedCallback();
    }
}

```

4.2.3 PWM

PWM 功能集成于 TIMx，是 TIMx 的一个应用模式。注意：对于 TIMx 来说，PWM 功能和定时功能不能同时使用。

PWM 设置界面如下所示：

→ TIM14 只支持一个 PWM 通道

→ TIM1 共支持 4 个 PWM 通道

在应用中如需要调整 PWM 占空比，可调用以下接口函数进行设置。

```

void TIM14_PWM_Pulse(uint32_t CHx, uint16_t percent)
{
    /* CH1 compare value:250 */
    TIM_OC_Initstruct.CompareValue = TIM14_Autoreload * percent / TIM14_RESOLUTION;
    /* Configure CH1 */
    LL_TIM_OC_Init(TIM14, CHx, &TIM_OC_Initstruct);
}

void TIM1_PWM_Pulse(uint32_t CHx, uint16_t percent)
{
    /* CH1 compare value:250 */
    TIM_OC_Initstruct.CompareValue = TIM1_Autoreload * percent / TIM1_RESOLUTION;
    /* Configure CH1 */
    LL_TIM_OC_Init(TIM1, CHx, &TIM_OC_Initstruct);
}

```

4.2.4 ADC

ADC 的设置界面如下：



ADC 初始化设置完成后，程序将按设定的时间间隔对使能的 ADC 通道按顺序进行 ADC 转换，采集的 ADC 数据存放于数组 ADCxConvertedData 中，用户程序可直接从数组 ADCxConvertedData 中获取 ADC 数据并进行相应操作。

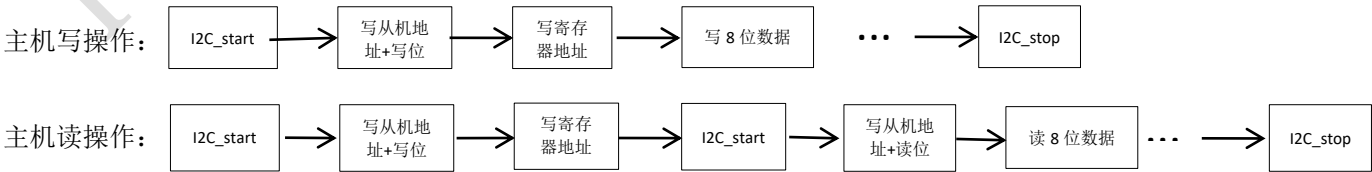


4.2.5 I2C (Slave Mode)

I2C 从机模式设置界面如下所示：



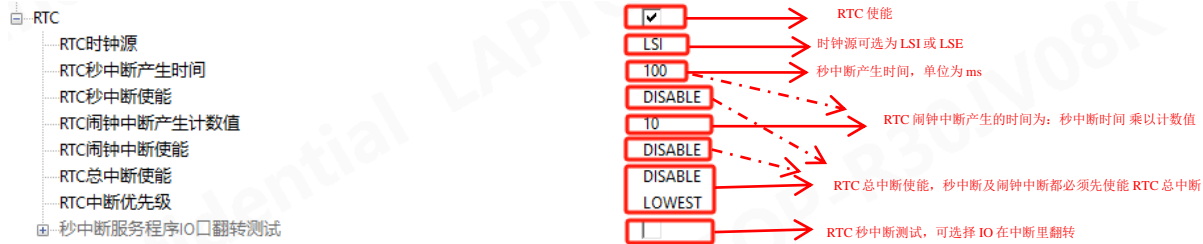
在从机模式，主机可对从机进行数据读写操作，从机的数据存放于 I2C_SRAM 数组中。I2C 主机通信应遵循以下的时序：



备注：“寄存器地址”即为 I2C_SRAM 数组的索引，如：I2C_SRAM[0] 的地址为 0。

4.2.6 RTC

RTC 设置界面如下:



RTC 中断回调函数如下(driver.c):

```

/*****
** 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)
{
}

/*****
** 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)
{
}

```

4.2.7 看门狗

看门狗设置界面:



程序默认在主循环内喂狗

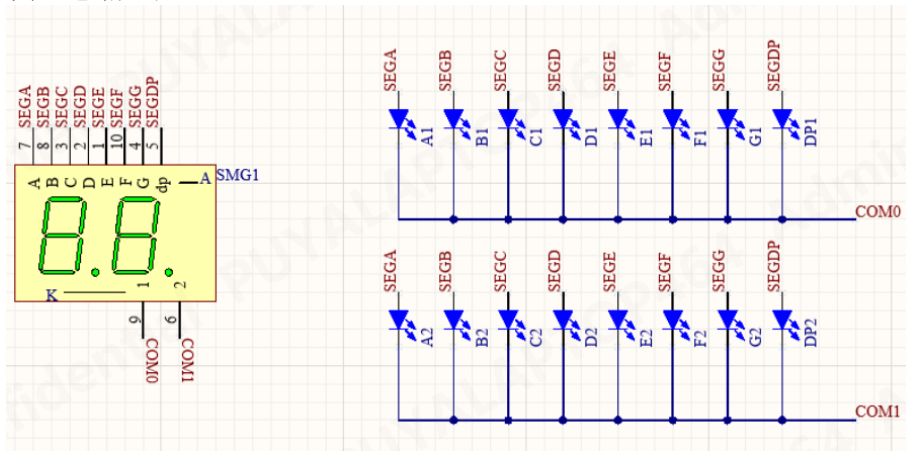


4.3 典型应用驱动

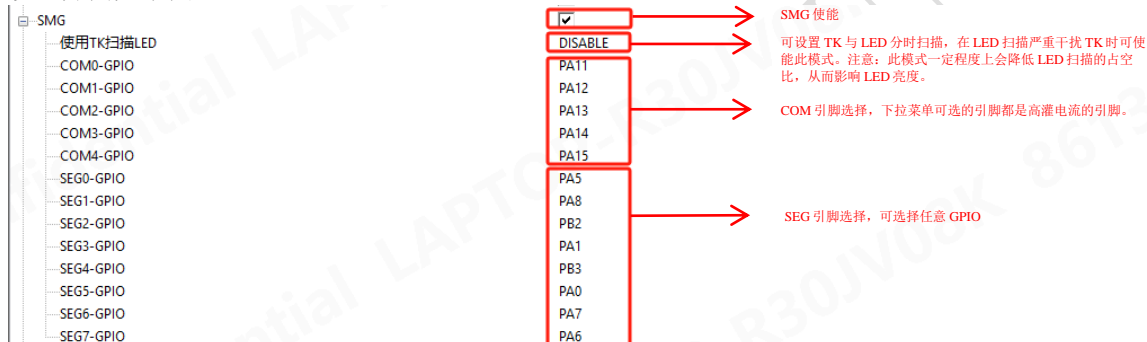
4.3.1 数码管/LED 驱动

4.3.1.1 共阴极数码管/LED 驱动

共阴数码管/LED 典型电路如下：



设置界面如下图：



应用实例(smgl_task.c)如下：

```

/*****
** 函数名 void SMG_Show(uint8_t num)
** 描述 : 使用数码管显示10进制数字
** 传入 : num 需要显示的数
** 返回 : 无
*****/
void SMG_Show(uint16_t num)
{
    smg_data[0] = num_tbl[num / 10];
    smg_data[1] = num_tbl[num % 10];
}

/*****
** 函数名 void LED_Show(uint8_t led_bit)
** 描述 : 按键LED指示
** 传入 : led_bit 需要显示的bit位置,
          bit0表示TK0的LED,
          bit1表示TK1的LED,
          .....
** 返回 : 无
*****/
void LED_Show(uint16_t led_bit)
{
    uint8_t i;
    uint8_t bit = 0;
    for (i = 0; i < SEG_COUNT; i++)
    {
        if (led_bit & 0X01)
            bit |= TK_LED[i];
        led_bit >>= 1;
    }
    smg_data[2] = bit;
}

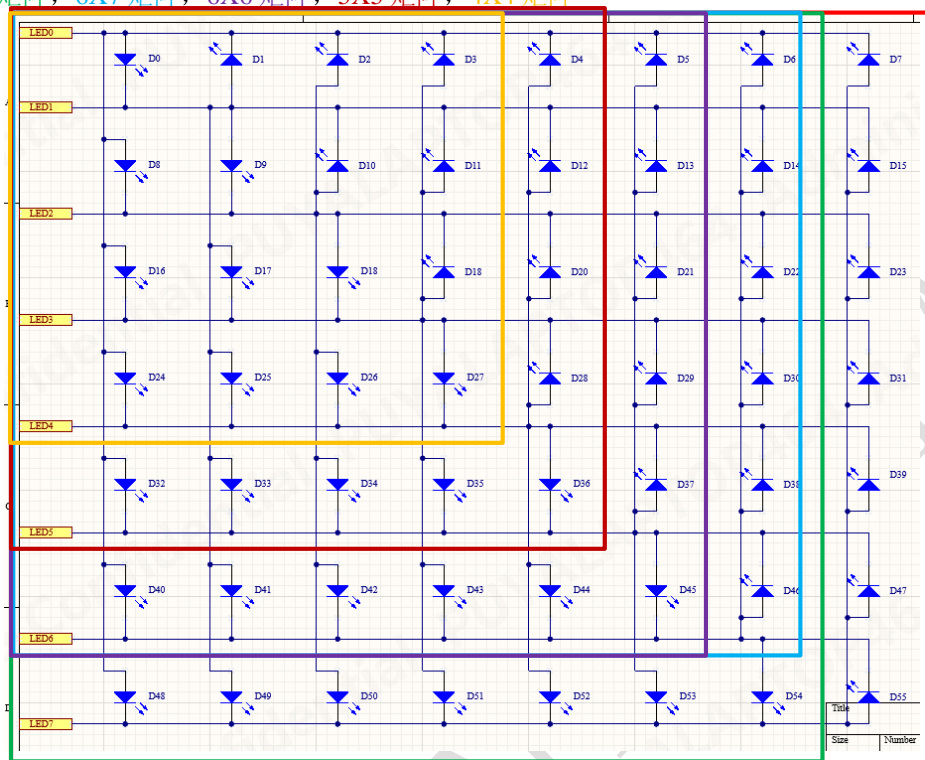
```

数组 smg_data 的数据对应数码管/LED 的显示内容，应用时只需向 smg_data 写入数据即可。

4.3.1.2 矩阵型 LED 驱动

矩阵型 LED 典型电路如下，针对不同的矩阵，LED 对应的地址是一致的；

7X8 矩阵，7X7 矩阵，6X7 矩阵，6X6 矩阵，5X5 矩阵，4X4 矩阵

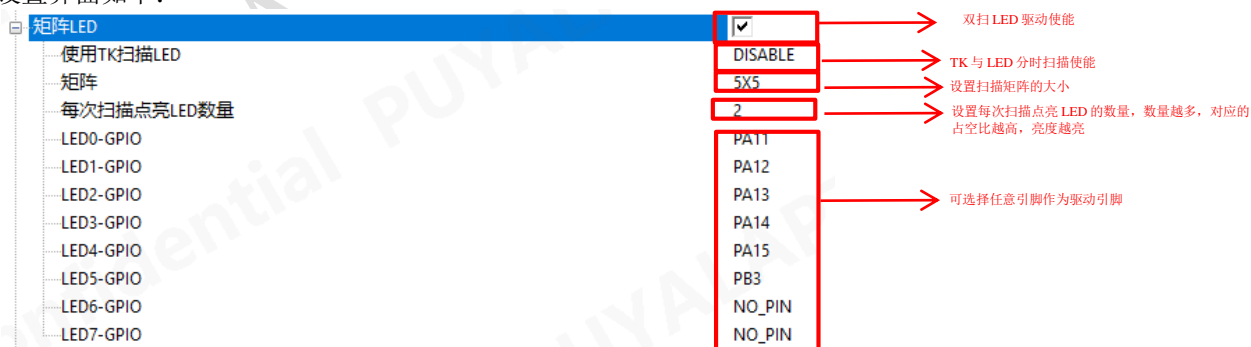


LED 显示配置缓存:

通过控制 led_data 缓存来控制是否亮灯，1: 亮，0: 熄灭

缓存	BIT0	BIT1	BIT2	BIT3	BIT4	BIT5	BIT6	BIT7
led_data[0]	D0	D1	D2	D3	D4	D5	D6	D7
led_data[1]	D8	D9	D10	D11	D12	D13	D14	D15
led_data[2]	D16	D17	D18	D19	D20	D21	D22	D23
led_data[3]	D24	D25	D26	D27	D28	D29	D30	D31
led_data[4]	D32	D33	D34	D35	D36	D37	D38	D39
led_data[5]	D40	D41	D42	D43	D44	D45	D46	D47
led_data[6]	D48	D49	D50	D51	D52	D53	D54	D55

设置界面如下:



4.3.2 专用驱动 IC 驱动数码管/LED

TM1624、TM1640 为常用的数码管/LED 驱动 IC，在使用驱动 IC 来驱动数码管/LED 时，只需要实现 mcu 与驱动 IC 间的通信并设置相应的配置命令及传输数据即可。其设置界面如下：

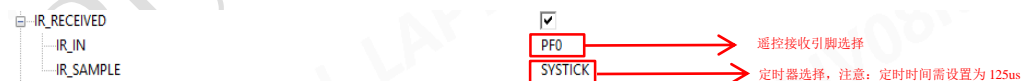


在应用时，只需要调用函数 TM1624_Display_Update/TM1640_Display_Update 即可对显示内容进行更新，如下图所示：

```
#if (APP_TM1624_ENABLE && TM1624_DISP_TEST)
    if (TM1624_Time > 500)
    {
        uint8_t i;
        TM1624_Time = 0;
        for (i = 0; i < 14; i++)
        {
            TM1624_Show[i] = ~TM1624_Show[i];
        }
        TM1624_Display_Update(TM1624_Show, 14, TM1624_ON | TM1624_DISP_DIM);
    }
#endif
#if (APP_TM1640_ENABLE && TM1640_DISP_TEST)
    if (TM1640_Time > 500)
    {
        uint8_t i;
        TM1640_Time = 0;
        for (i = 0; i < 16; i++)
        {
            TM1640_Show[i] = ~TM1640_Show[i];
        }
        TM1640_Display_Update(TM1640_Show, 16, TM1640_ON | TM1640_DISP_DIM);
    }
#endif
```

4.3.3 红外遥控接收

红外遥控接收功能使用定时器扫描 IO 口状态，进行红外信号解码，设置界面如下：



应用程序可调用函数 IR_Press 读取接收的红外数据，如下图所示：

```

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
}

```

4.3.4 用户数据掉电存储

此功能使用 ADC 定时检测芯片供电电压，当供电电压低于所设置的阈值，程序会把用户需保存的数据存入 FLASH。

设置界面及步骤如下：

Flash User OTP

掉电保存数据

掉电检测电压

ADC

ADC采样间隔时间

ADC-PA0-IN0

ADC-PA1-IN1

ADC-PA2-IN2

ADC-PA3-IN3

ADC-PA4-IN4

ADC-PA5-IN5

ADC-PA6-IN6

ADC-PA7-IN7

ADC-PB0-IN8

ADC-PB1-IN9

ADC-TEMPSENSOR

ADC-VREFINT

ADC-1_3VCCA

☒

3100

STEP1: 掉电数据保存功能使能

STEP2: 设置低电电压，当电压低于此电压，将触发数据保存功能

☒

10

DISABLE

DISABLE

DISABLE

DISABLE

DISABLE

DISABLE

DISABLE

DISABLE

DISABLE

DISABLE

DISABLE

ENABLE

ENABLE

DISABLE

STEP3: 使能 ADC，并设置采样间隔时间，注意：采样时间应根据实际掉电时间来设置，须保证在掉电过程中完成数据存储。

STEP4: 使能 ADC 检测芯片供电电压功能

应用实例：

```

10 #if LVD_WRITE_USER_DATA
11 /*****
12 ** 函数名 void ADC_LVD_ISR(void)
13 ** 描述   : 低电压回调函数
14 ** 传入   : 无
15 ** 返回   : 无
16 *****/
17 void ADC_LVD_ISR(void)
18 {
19     Vcc_Power = 1200 * 4095 / ADC_ConvertedData[ADC_CHANNEL_VREFINT];
20     if (power_on == 1 && power_off == 0 && Vcc_Power > 2500 && Vcc_Power < low_voltage)
21     {
22         uint8_t sta;
23         power_off = 1;
24         power_count++;
25         User_Cache_Write(0, &power_count, 1);
26         sta = User_Flash_Write();
27         log_printf("write_otp:%d\r\n", sta);
28     }
29 }

```

此例程中，power_count 表示掉电的次数，每次掉电数据递增，当检测到芯片电压低于设置的电压时，power_count 将被存入 OTP 区域（注：OTP 为专用 FLASH 扇区，容量为 128 字节），在实际应用中可将用户数据替换即可。

```

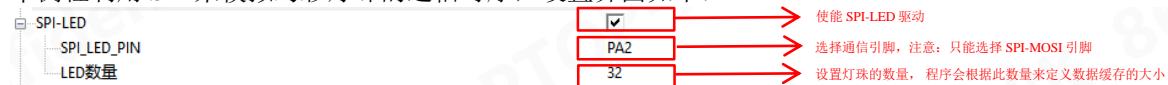
30 #if APP_USER_OTP_ENABLE
31 /* 用户数据读取 */
32 #if LVD_WRITE_USER_DATA
33     if (User_Cache_Read(0, &power_count, 1))
34     {
35         log_printf("power_count:%d\r\n", power_count);
36     }
37     else
38     {
39         log_printf("first power\r\n");
40         power_count = 0;
41     }
42 #endif
43 #endif

```

上电后，把 power_count 读出并打印出来进行验证。

4.3.5 类 W2812B 灯珠驱动

本例程利用 SPI 来模拟幻彩灯珠的通信时序，设置界面如下：

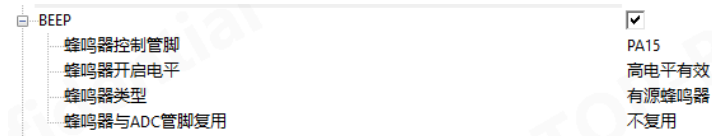


在应用中，应用程序首先将待发送的数据装载于缓存数组 `rgb_data`，然后再调用函数 `SPI_LED_Transmit` 进行数据发送，函数 `SPI_LED_Transmit` 如下所示：

```
*****  
void SPI_LED_Transmit(void)  
{  
    for (uint16_t i = 0; i < SPI_LED_CNT * 6; i++)  
    {  
        while (LL_SPI_IsActiveFlag_TXE(SPI1) == 0)  
        {  
            ;  
        }  
        LL_SPI_TransmitData16(SPI1, rgb_data[i]);  
    }  
}
```

4.3.6 蜂鸣器控制

蜂鸣器的设置界面如下：



在应用中，可调用函数 BEEP_On 来控制蜂鸣器，输入参数为蜂鸣器响的时间，如下所示：

```

/*****
** 函数名   void BEEP_On(uint8_t timeout)
** 描述    : 打开蜂鸣器
** 传入    : timeout 开启时间ms
** 返回    : 无
*****/
void BEEP_On(uint8_t timeout)
{
    beep_delay = timeout;
}
#ifdef BEEP_ADC
#ifdef APP_ADC_ENABLE
    while (adc_state == 1)        //复用状态下如果ADC在采集中，等待采集完成
    {
        __WFI();
    }
#endif
#endif
    ntc_delay = 5;
    GPIO_Init(BEEP_GPIO, OUTPUT | PUSH_PULL);
    #elif BEEP_TYPE
        GPIO_Init(BEEP_GPIO, OUTPUT | PUSH_PULL);
    #endif
    #if BEEP_GPIO_TYPE
        GPIO_SetBit(BEEP_GPIO);
    #else
        GPIO_ClearBit(BEEP_GPIO);
    #endif
}

```

应用实例：

```

#ifdef
    for (i = 0; i < TKCtr.TouchKeyChCnt; i++)
    {
        if (temp & 0X01)
        {
            if (KeyFlag & ((0X01) << i)) // 按键按下
            {
                #if APP_BEEP_ENABLE
                    BEEP_On(100);
                #endif
            }
        }
    }
    #if APP_SMG_ENABLE

```

5. 用户应用程序接口

User_code.c 文件包含了用户应用程序的接口函数，接口函数如下表：

函数名	功能描述
user_init	用户应用程序初始化，用户可将初始化程序放置于此函数内。
user_loop	此函数在主循环中调用，用户可将在主循环中运行的程序放置于此函数。
user_timer	此函数在 SYS_TICK 中断中调用，用户可将在定时器中运行的程序放置于此函数。
ADC_Callback	此函数在 ADC 转换完成后调用，用户可在此函数内读取 ADC 数据并添加相应处理程序。

6. 更新历史

Version	Content	Date
V1.0	Initial version	2025/3/20



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