

PY32T09x 系列触摸应用 开发指南



Puya Semiconductor (Shanghai) Co., Ltd.

目录

1. 简介	4
2. 软件库结构	5
3. 触摸库介绍与使用	9
3.1. 触摸功能类别介绍	9
3.2. MCU 资源需求	9
3.2.1. MCU 硬件资源	9
3.2.2. MCU 软件资源	9
3.3. 触摸库接口解析	11
3.3.1. 数据接口	11
3.3.2. 函数接口	11
3.4. 触摸库功能选项及参数配置	13
3.4.1. 触摸按键用户配置	13
3.4.2. 滑条/滑轮用户配置	14
3.4.3. 触摸功能及参数配置	14
3.5. 用户可修改的触摸库回调函数	16
3.6. 触摸应用配置步骤及流程	18
3.6.1. 常规按键应用	18
3.6.2. 滑轮/滑条应用	24
3.6.3. 触摸库低功耗应用	26
3.6.4. 隔空按键应用	27
3.6.5. 防水按键应用（有 Shield 通道）	28
3.6.6. 防水按键应用（无 Shield 通道）	29
3.6.7. 触摸检水应用	31
4. 系统配置及外设应用解析	33
4.1. 系统配置	33
4.1.1. 时钟配置	33
4.2. 外设应用的接口与使用	33
4.2.1. UART	33
4.2.2. 定时器	35
4.2.3. PWM	40
4.2.4. 看门狗	41
4.2.5. 低电压检测（PVD）	41
4.2.6. ADC	42
4.2.7. DMA	43
4.3. 典型应用驱动	43
4.3.1. 数码管/LED 驱动	43
4.3.2. 用户数据存储	46
4.3.3. 类 WS2812B 灯珠驱动	47
4.3.4. 蜂鸣器控制	47
5. 用户应用程序接口	49

6. 更新历史.....50

Puya Confidential

1. 简介

PY32T09x 系列标准软件库 (PY32T09x_Touch_Library_Vxx) 集合了系统配置、触摸库、外设库 (HAL 库)、基于 HAL 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了解。

2. 软件库结构

软件库的基本结构如下图所示：

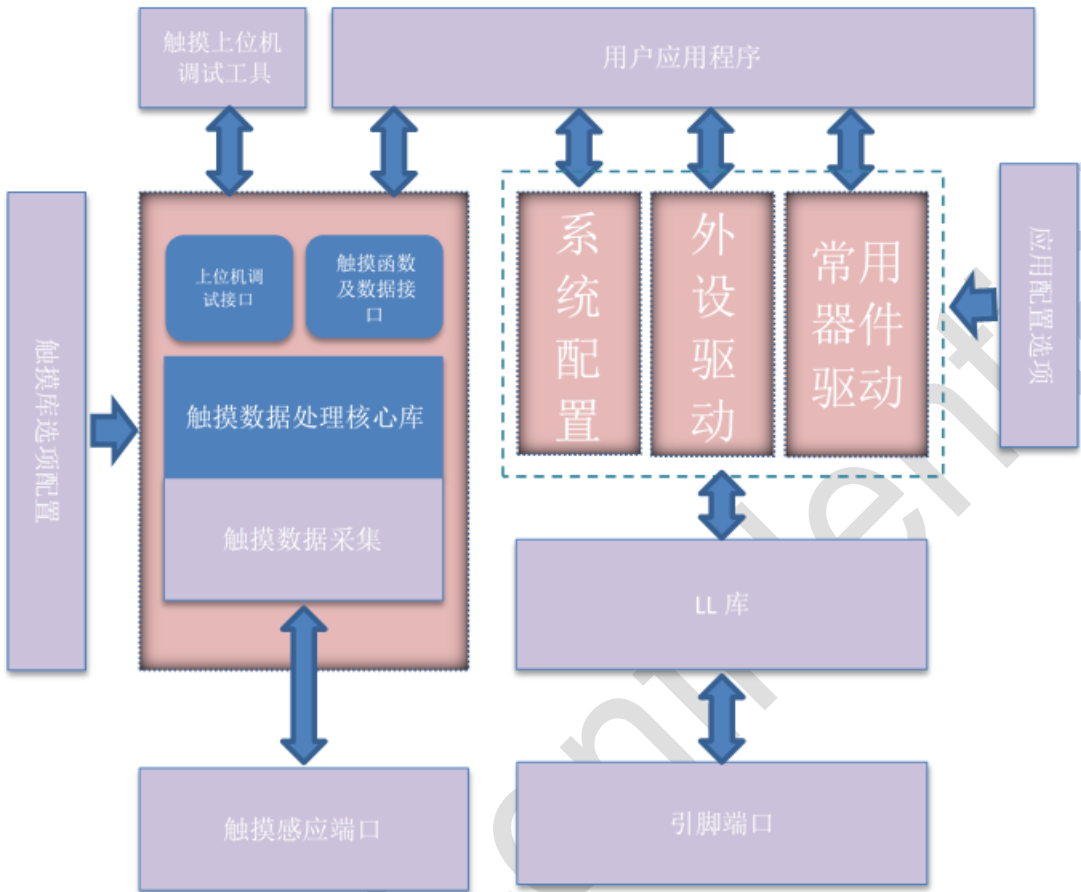


图 2-1 软件库基本结构

以上软件结构图中主要模块与软件工程目录的对应关系如下：

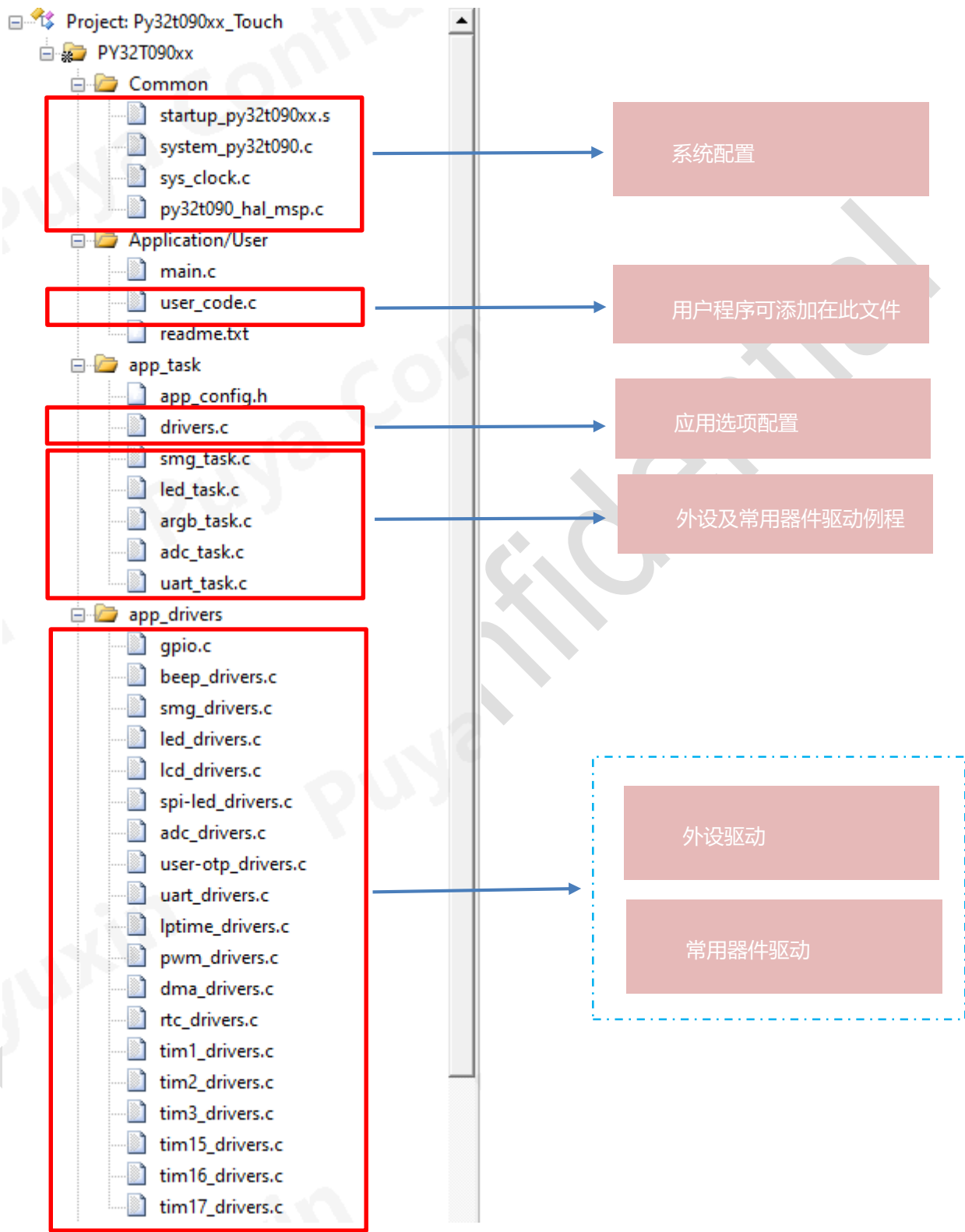


图 2-2 工程目录与驱动分层结构图

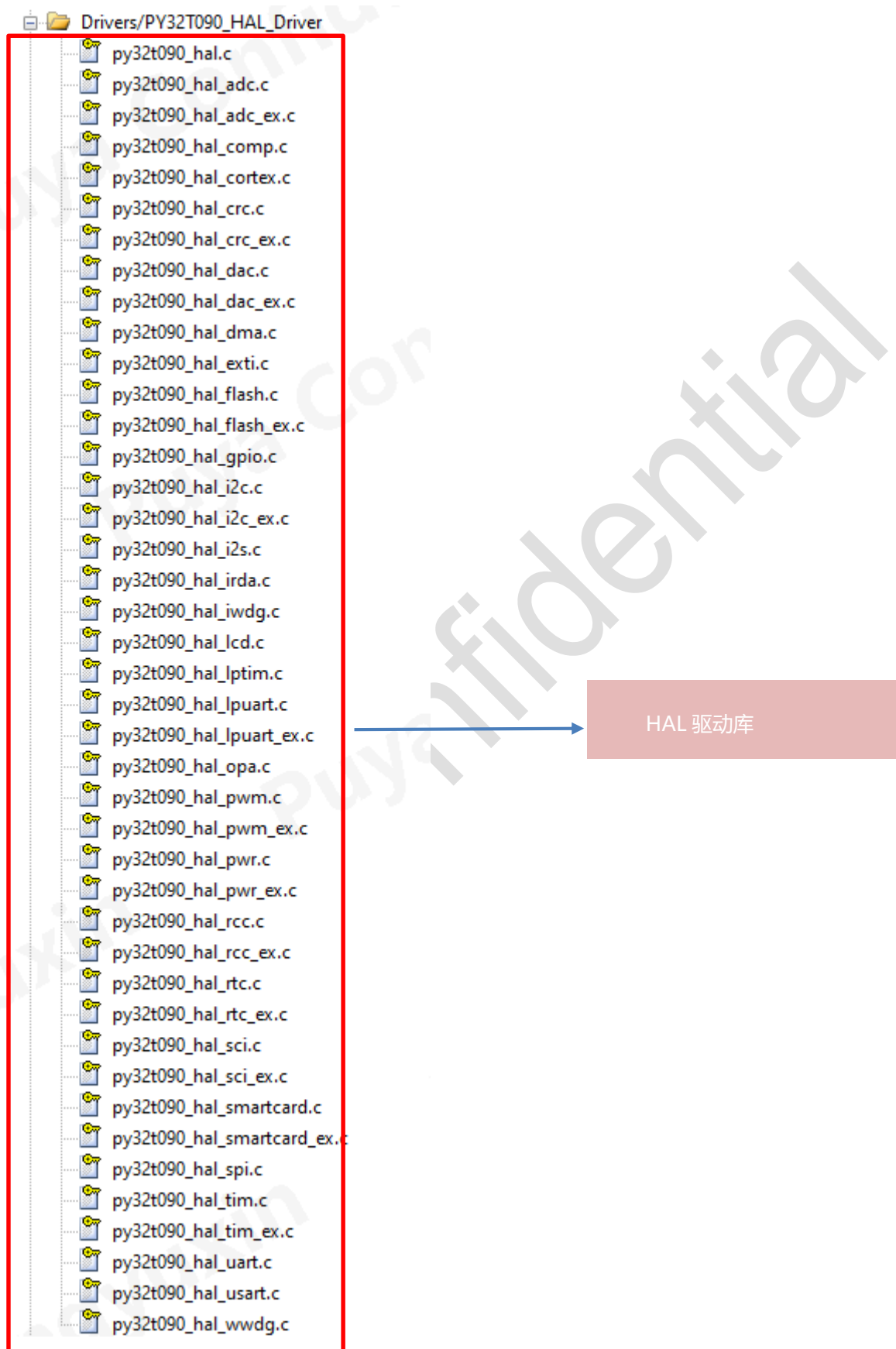


图 2-3 HAL 驱动层文件结构图

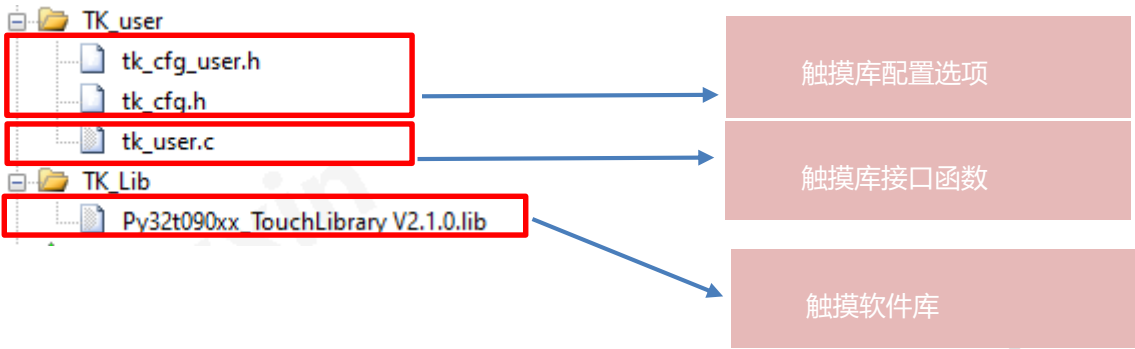


图 2-4 触摸库软件模块结构图

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（隔空触摸、触摸检水不支持触摸唤醒功能）

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：

- SYSTICK 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：

表 3-1 触摸功能与通道配置表

触摸功能	FLASH	SRAM	通道
常规按键	9.36K	1.83K	8(Key)
隔空按键	9.34K	1.77K	6(Key)
防水按键	11.1K	2.05K	9(Key)+1(Shield)
滑轮+滑条+按键	12.3K	1.9K	2(Key)+4(Wheel)+4(Slider)
滑条+滑轮	11.02K	1.83K	4(Wheel)+4(Slider)
触摸检水	9.94K	1.72K	1(Ref)+1(Detect)

注：

- 1) 上表中 SRAM 占用包括堆栈 1024 Byte
- 2) 每增加一个通道 SRAM 约增加 32 Byte

表 3-2 触摸功能时间测试数据表

触摸功能	TK 频率 (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)
弹簧按键	24	8	12000	1000	545	270
	48			500	390	200
隔空按键	24	6	30000	2500	405	205
	48			1250	290	150
防水按键	24	8+1	12000	1000	700	502
	48			500	500	405
滑轮/滑条	24	4	12000	1000	250	100
	48			500	195	70
检水	24	2	12000	500	13	60
	48			250	10	53

注：

- 1. TK 采样时间与窗口设置值成正比，与触摸时钟频率成反比。
- 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： $8 * 1000us=8000 us$ ，中断响应+数据处理时间共为： $545+ 270= 815us$ ，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： $815/8000 = 10.1\%$ 。

3.3. 触摸库接口解析

3.3.1. 数据接口

表 3-3 触摸库数据接口表

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，bit 0~bit 31 分别对应 KEY0~KEY31，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.SliderOrWheel- Position[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。

3.3.2. 函数接口

表 3-4 触摸库函数接口表

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，为触摸功能提供时基。

```

    int main(void)
    {
        /*优先初始化打印*/
        #if (APP_TK_ENABLE && DEBUG_ENABLE && (DEBUG_MODE == 0))
            log_init();
        #endif
        HAL_Init();
        /* System clock configuration */
        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*/
        #if APP_TK_ENABLE
            TK_Init();
        #endif
        user_init();
        /* infinite loop */
        log_printf("Start Loop\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
            HAL_IncTick();
        }
        #if APP_BEEP_ENABLE
            BEEP_Toggle();
        #endif
        #if APP_LED_ENABLE
            LED_Scan();
        #endif
        user_timer();
    }

```

图 3-1 触摸库函数调用代码示例图

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
#define CH_KEY0      TK_CH27
#define CH_KEY1      TK_CH24
#define CH_KEY2      TK_CH20
#define CH_KEY3      TK_CH19
#define CH_KEY4      TK_CH26
#define CH_KEY5      TK_CH25
#define CH_KEY6      TK_CH22
#define CH_KEY7      TK_CH21
#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
#define CH_KEY26     TK_CH_NONE
#define CH_KEY27     TK_CH_NONE
#define CH_KEY28     TK_CH_NONE
#define CH_KEY29     TK_CH_NONE
#define CH_KEY30     TK_CH_NONE
#define CH_KEY31     TK_CH_NONE
#define CH_KEY32     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      50
#define KEY9_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
KEYn_THD 时，KEYn 将被触发。

图 3-2 触摸按键通道与门限配置代码示例

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：

```
//
#define SLIDER_OR_WHEEL0_TYPE TK_APP_WHEEL
#define SLIDER_OR_WHEEL0_RESOLUTION 360 //滑条分辨率
#define SLIDER_OR_WHEEL0_THD 101 //滑条门限值
//滑条通道，按顺序填写
#define SLIDER_OR_WHEEL0_CH0 TK_CH0
#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

#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

#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

#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
//
```

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

图 3-3 滑条/滑轮配置示意图

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。



图 3-4 触摸常规功能配置



图 3-5 触摸特殊功能配置

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：

表 3-5 触摸库用户回调函数表

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 参数名： raw 数据类型： uint16_t 参数含义： 触摸数据	返回值： 补偿后的触摸数据	用于对触摸数据进行补偿，例如：当 LED 灯亮灭变化时，触摸数据可能会同步变化，在此情况下，可在此函数内根据 LED 的状态对触摸数据进行同步补偿。
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，用于获取是否需对触摸模式切换的信息。
TK_UpdatBaseLineData	参数名： chs 数据类型： uint8_t 参数含义： 通道索引	无	调用后强制更新该通道基线值，适用于检测到异常时主动释放按键状态。

3.6. 触摸应用配置步骤及流程

3.6.1. 常规按键应用

Step1: 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：

tk_cfg_user.h

```
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	UART3_RX	TM14_CH1	CMP1_INP TK_CH9
----	----	----	----	----	---	---	---	-----	----	-------	---	----------	----------	----------	--------------------

图 3-6 触摸按键通道配置方法一

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。

PY Touch-Link v1.0.9

保存 设置 导出 下载 帮助 系统设置

选择 PY32T09G1657 开始调试 运行

选择芯片型号

①选择芯片型号

KEY0 28TK0
KEY1 27TK1
KEY2 26TK2
KEY3 25TK3
KEY4 24TK4
KEY5 23TK5
KEY6 22TK6
KEY7 21TK7
KEY8 20TK8
KEY9 19TK9
KEY10 18TK10
KEY11 17TK11
KEY12 16TK12
KEY13 15TK13

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

触摸按键

通道/值

KEY0 KEY1 KEY2 KEY3 KEY4 KEY5 KEY6 KEY7

通道 TK17 TK15 TK11 TK5 TK16 TK10 TK7 TK6

初始值 0 0 0 0 0 0 0 0

门限值 100 100 100 100 100 100 100 100

按键次数 0 0 0 0 0 0 0 0

③双击对应按键的门限值单元格，设置初始门限值

日志

PY-Link已断开连接



图 3-7 触摸按键通道配置方法二

Step2: 程序编译，编译完成后下载到目标板，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）

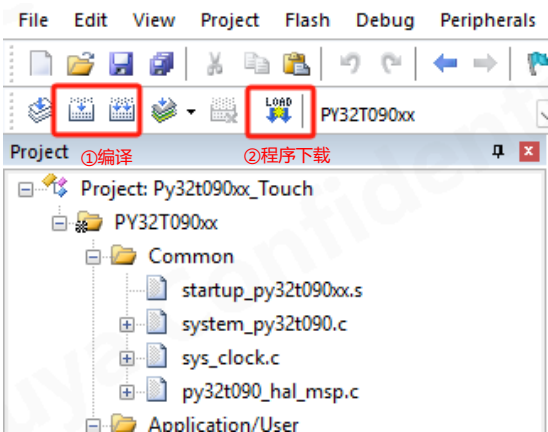


图 3-8 下载程序方法——KEIL 下载

方法二：使用 PY Touch - Link.exe 下载，如下图：

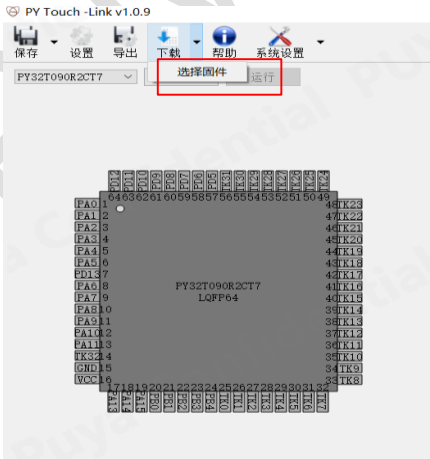


图 3-9 下载程序方法二——上位机下载

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

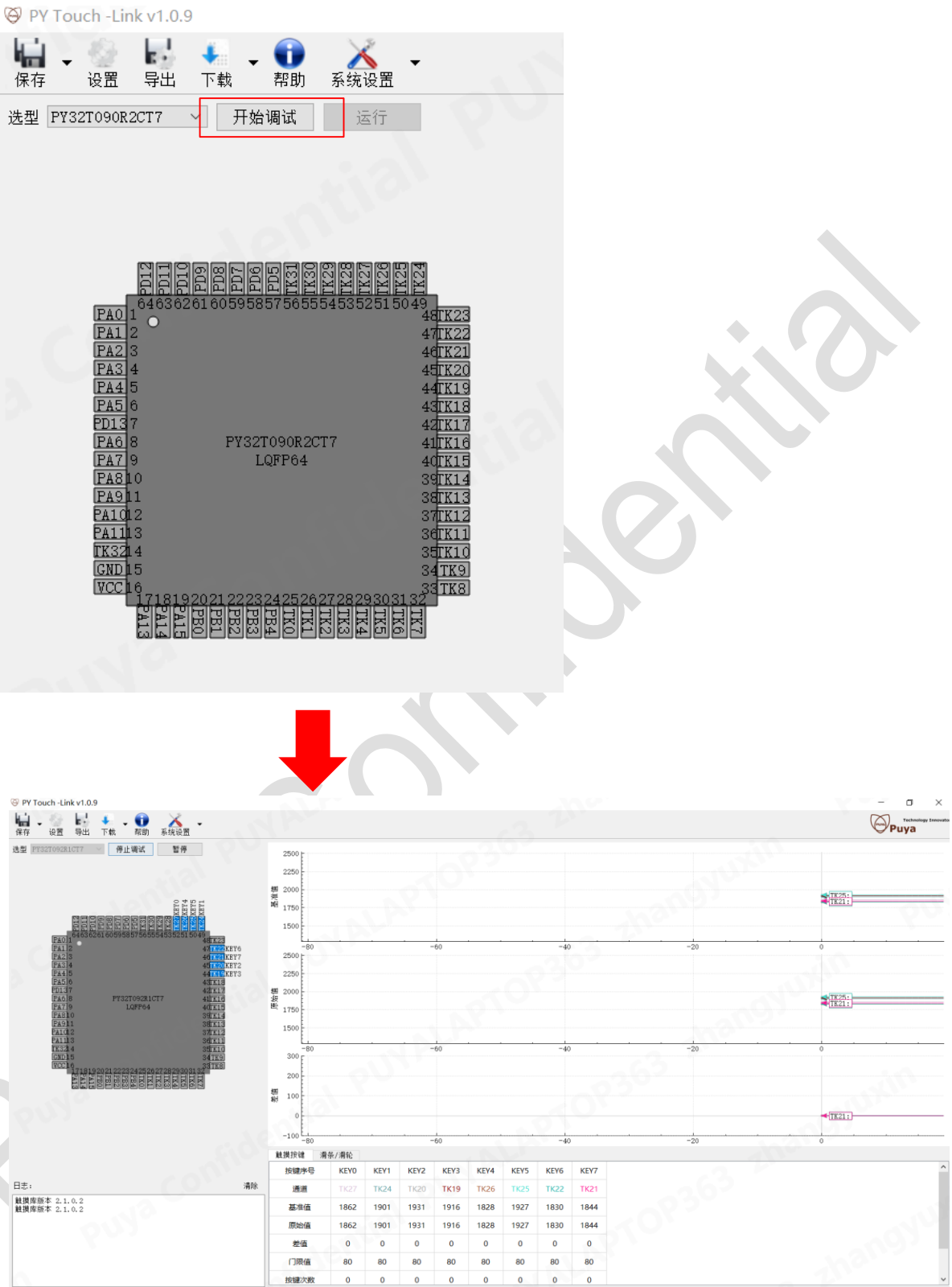


图 3-10 触摸上位机调试示例

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							
通道	TK27	TK24	TK20	TK19	TK26	TK25	TK22	TK21
基准值	1895	1900	1928	1914	1827	1927	1828	1841
原始值	2009	1900	1928	1914	1827	1927	1828	1841
差值	114	0	0	0	0	0	0	0
门限值	80	80	80	80	80	80	80	80
按键次数	9	0	0	0	0	0	0	0
状态								

图 3-11 触摸按键变化量与门限值计算示例

例：如上图，手指正常触摸 KEY0 时，变化量约为 114，门限值可设为： $114 \times 70\% = 80$ 。设置方法如下：

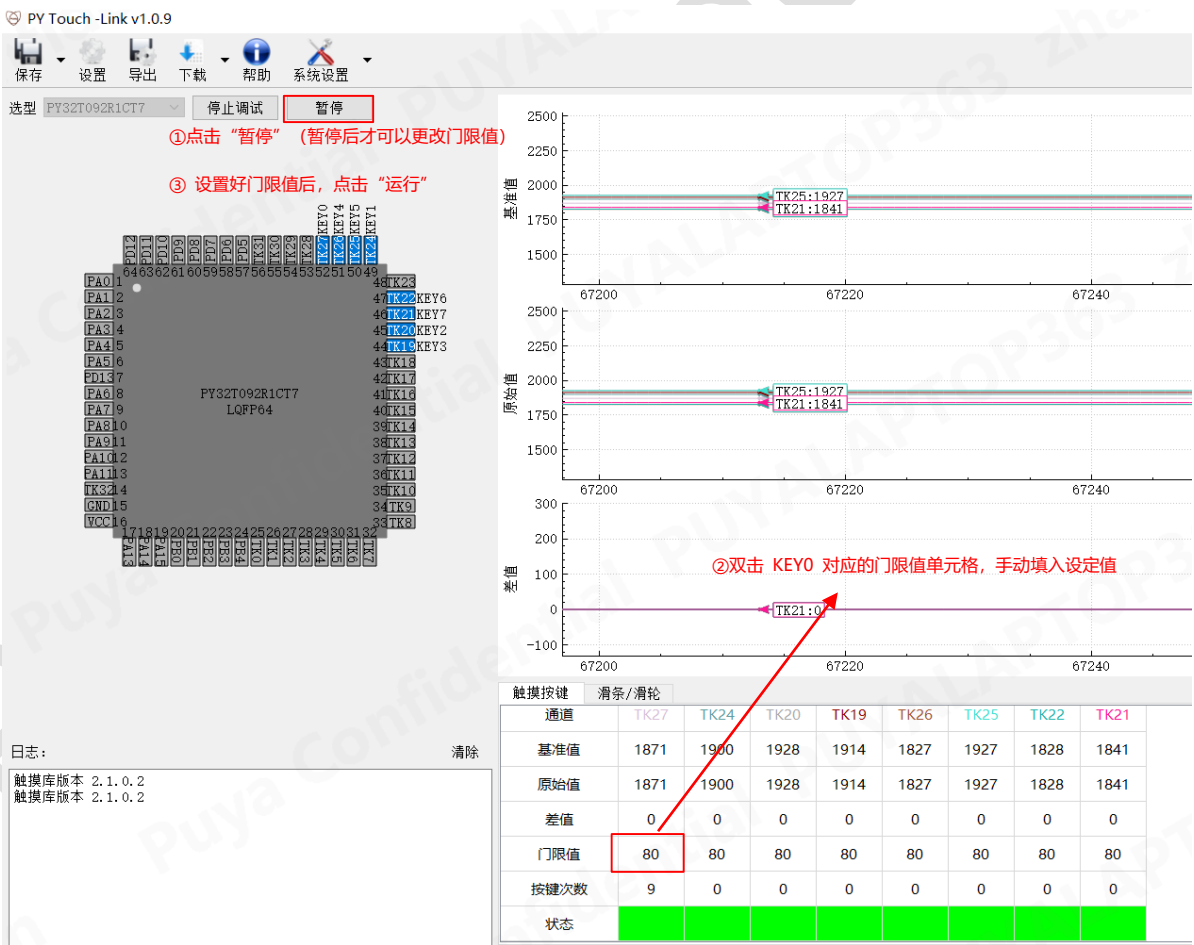


图 3-12 上位机门限值设置操作界面示例

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。

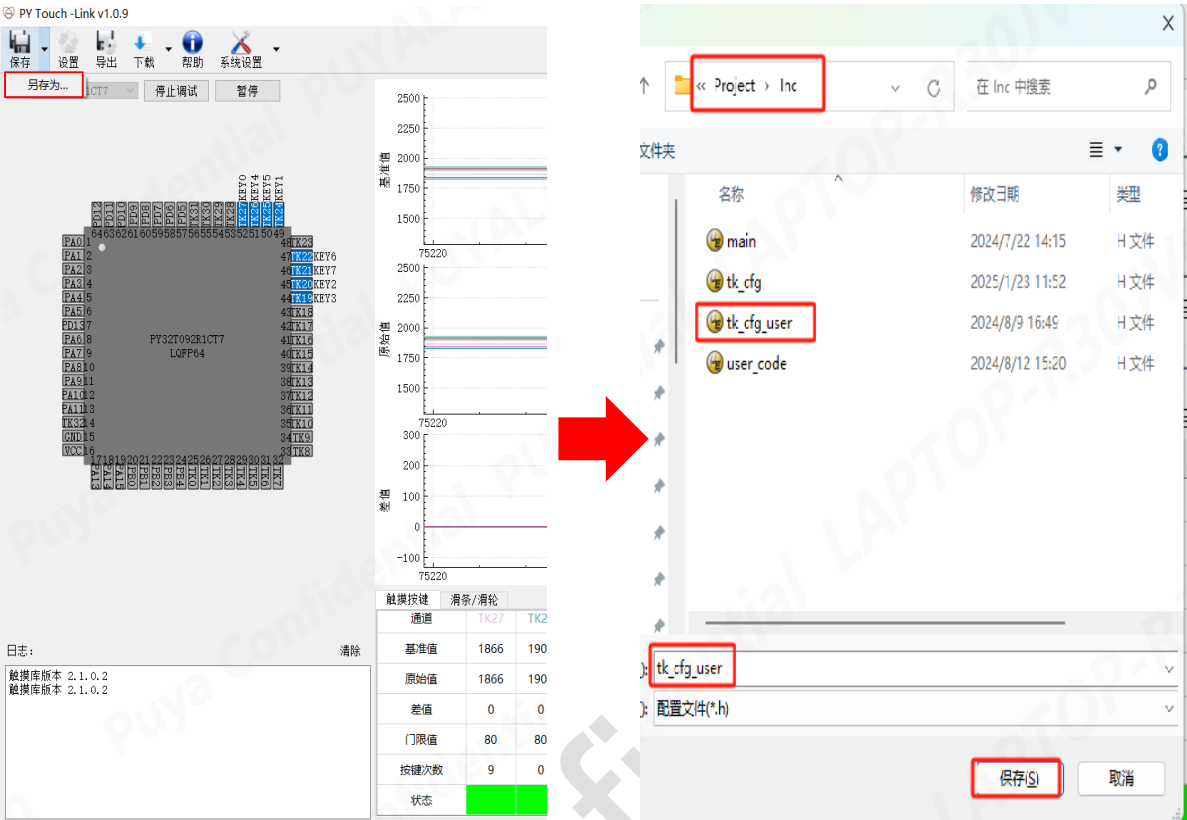


图 3-13 触摸配置文件导出操作界面示例

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。

tk_user.c

tk_cfg.h

Expand All

Collapse All

Help

Show Grid

Option	Value
基础配置	
TK_DIV	DIV0
CMOD电容	内部
参考电压	2.0V
自适应电流	ENABLE
滤波系数	20
跳频模式	☒
多项式宽度	8
随机数种子	0xA7
输出补偿	☒
输出电压	2.0V
输出类型	低电平
多按键检测	☐
变化量	25
数量	2
按键参数配置	
分频系数	3
扫描窗口	12000
按下消抖次数	3
释放消抖次数	3
基线向上更新步进时间	200
基线向下更新步进时间	100
限制最长输出时间	15000
释放门限值比例	90
基线更新模式	MODE0
单键触发模式	DISABLE
多按键数量	0
抹布检测	☐
斜率	100
防水模式	☐
SHIELD通道	TK14
斜率	50

滤波系数取值范围为：5~50，数值越小，滤波能力越强，但 按键响应速度会变慢。实际应用时，可在保证按键体验感的前提下，把滤波系数尽量设小一些。

在按键寄生电容较大且按键灵敏度偏低的情况，可将分频系数适当调大一些。实际调试时，可逐步调整参数并同步观察 按键变化量，取使变化量达最大值的 90%左右的分频系数即可。

窗口越大，扫描时间越长，按键时触摸数据变化量也越大。如果按键响应速度偏慢，把窗口适当调小一些可提高响应速度。

某些应用需设置单键或限制多按键的数量，可根据实际需求进行设置。

Option

SHIELD通道

斜率

滑条参数配置

分频系数

输出补偿

扫描窗口

按下消抖次数

释放消抖次数

基线向上更新步进时间

基线向下更新步进时间

限制最长输出时间

触摸省电配置

无操作进入休眠时间

唤醒门限值

唤醒滤波次数

唤醒滤波间隔

扫描窗口

分频系数

参考电压

STOP调试使能

内部参考配置

噪音门限值

SHIELD通道	TK14
斜率	50
分频系数	6
输出补偿	ENABLE
扫描窗口	6000
按下消抖次数	3
释放消抖次数	3
基线向上更新步进时间	200
基线向下更新步进时间	100
限制最长输出时间	15000
无操作进入休眠时间	☐
唤醒门限值	400
唤醒滤波次数	15
唤醒滤波间隔	3
扫描窗口	5
分频系数	6000
参考电压	8
STOP调试使能	0.6V
噪音门限值	DISABLE
	☒
	100

内部参考通道可用于检测电源干扰，使能内参通道并设置合适的门限值可以过滤因电源异常波动而误产生的按键，如对讲机干扰等场景。注意：此门限值必须留较大的余量，否则 可能会影响按键正常操作或在 EMC 认证时屏蔽按键。

图 3-14 触摸参数配置向导界面示例

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。

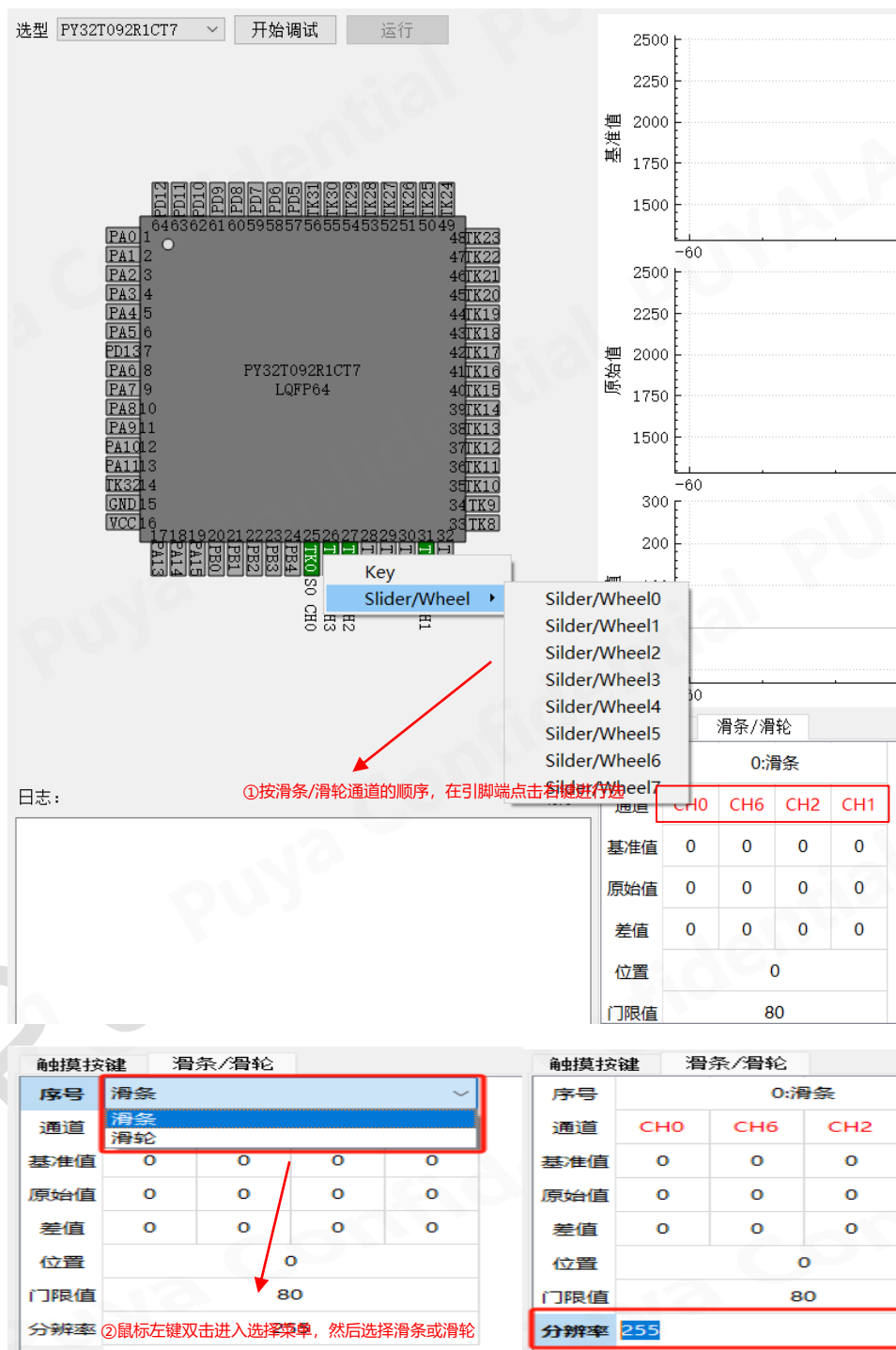


图 3-15 滑条/滑轮配置界面示例

方法二：在 tk_cfg_user.h 中直接修改，如下图。

```
//=====
#define SLIDER_OR_WHEELO_TYPE          TK_APP_WHEEL          //滑条类型
#define SLIDER_OR_WHEELO_RESOLUTION    360                  //滑条分辨率
#define SLIDER_OR_WHEELO_THD           80                   //滑条门限值
#define SLIDER_OR_WHEELO_CH0           TK_CH0               //滑条通道，按顺序填写
#define SLIDER_OR_WHEELO_CH1           TK_CH6
#define SLIDER_OR_WHEELO_CH2           TK_CH2
#define SLIDER_OR_WHEELO_CH3           TK_CH1
#define SLIDER_OR_WHEELO_CH4           TK_CH_NONE
#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 3-16 头文件滑条/滑轮配置代码示例

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

触摸按键	滑条/滑轮							
序号	0:滑轮				1:滑条			
	CH0	CH6	CH2	CH1	CH19	CH23	CH24	CH25
基准值	956	943	939	949	952	914	927	932
原始值	1091	943	940	962	952	914	927	932
差值	135	0	1	13	0	0	0	0
位置	-1				-1			
门限值	80				100			

触摸按键	滑条/滑轮							
序号	0:滑轮				1:滑条			
	CH0	CH6	CH2	CH1	CH19	CH23	CH24	CH25
基准值	951	940	937	946	952	913	925	930
原始值	951	940	937	946	952	913	925	930
差值	0	0	0	0	0	0	0	0
位置	-1				-1			
门限值	80				100			

- 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。

```
//=====
#define SLIDER_OR_WHEELO_TYPE          TK_APP_WHEEL          //滑条类型
#define SLIDER_OR_WHEELO_RESOLUTION    360                  //滑条分辨率
#define SLIDER_OR_WHEELO_THD           80                   //滑条门限值
#define SLIDER_OR_WHEELO_CH0           TK_CH0               //滑条通道，按顺序填写
#define SLIDER_OR_WHEELO_CH1           TK_CH6
#define SLIDER_OR_WHEELO_CH2           TK_CH2
#define SLIDER_OR_WHEELO_CH3           TK_CH1
#define SLIDER_OR_WHEELO_CH4           TK_CH_NONE
#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 3-17 滑条/滑轮配置界面示例

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
}
```

图 3-18 滑条/滑轮位置信息读取代码示例

3.6.3. 触摸库低功耗应用

Step1: 在 tk_cfg.h 中使能触摸低功耗模式，并配置基础参数。

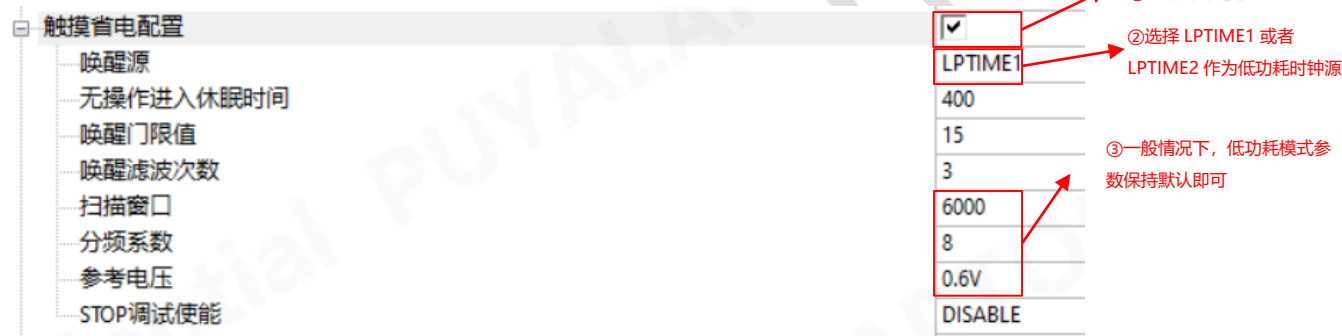


图 3-19 低功耗配置界面示例

由于触摸低功耗以 LPTIME1 或者 LPTIME2 为时基，所以也需先配置 LPTIME 使能，LPTIME 在 app_config.h 中进行配置。



图 3-20 LPTIME 时基配置界面示例

Step2：设置低功耗模式触发门限值。



图 3-21 低功耗门限值设置界面示例

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	滑条/滑轮								
按键序号	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7	
通道	TK17	TK15	TK11	TK5	TK16	TK10	TK7	TK6	TK_COM-SHIELD
基准值	1822	1786	1770	1932	1837	2232	1859	1837	5449
原始值	1822	1786	1770	1932	1837	2232	1859	1837	5479
差值	0	0	0	0	0	0	0	0	30
门限值	100	100	100	100	100	100	100	100	15
按键次数	0	2	1	0	3	2	1	0	0

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。

图 3-22 低功耗调试数据界面示例

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。



图 3-23 低功耗调试关闭配置示例

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，如下图：

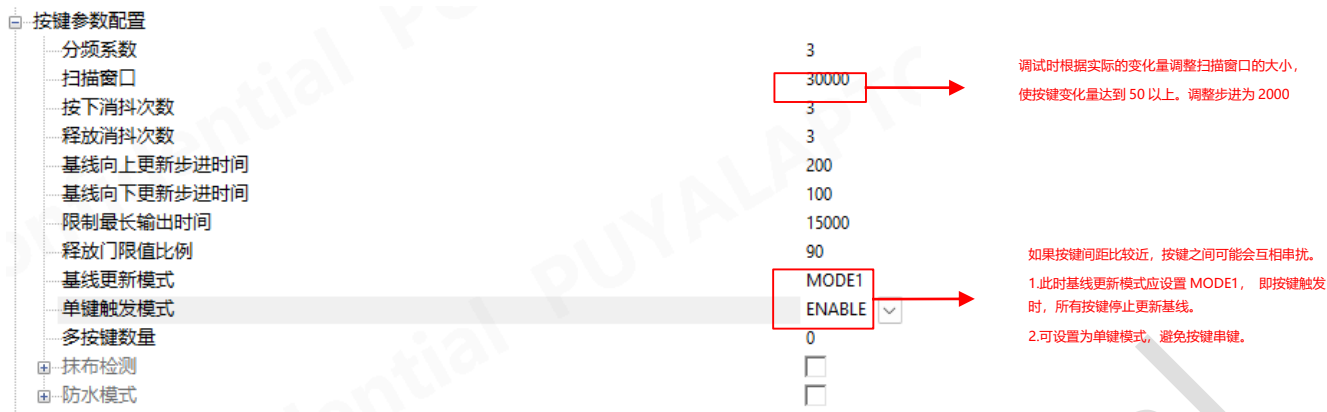


图 3-24 隔空按键配置界面

3.6.5. 防水按键应用 (有 Shield 通道)

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。

Step 1: 增大分频系数，修改补偿输出类型，设置 Shield 通道。

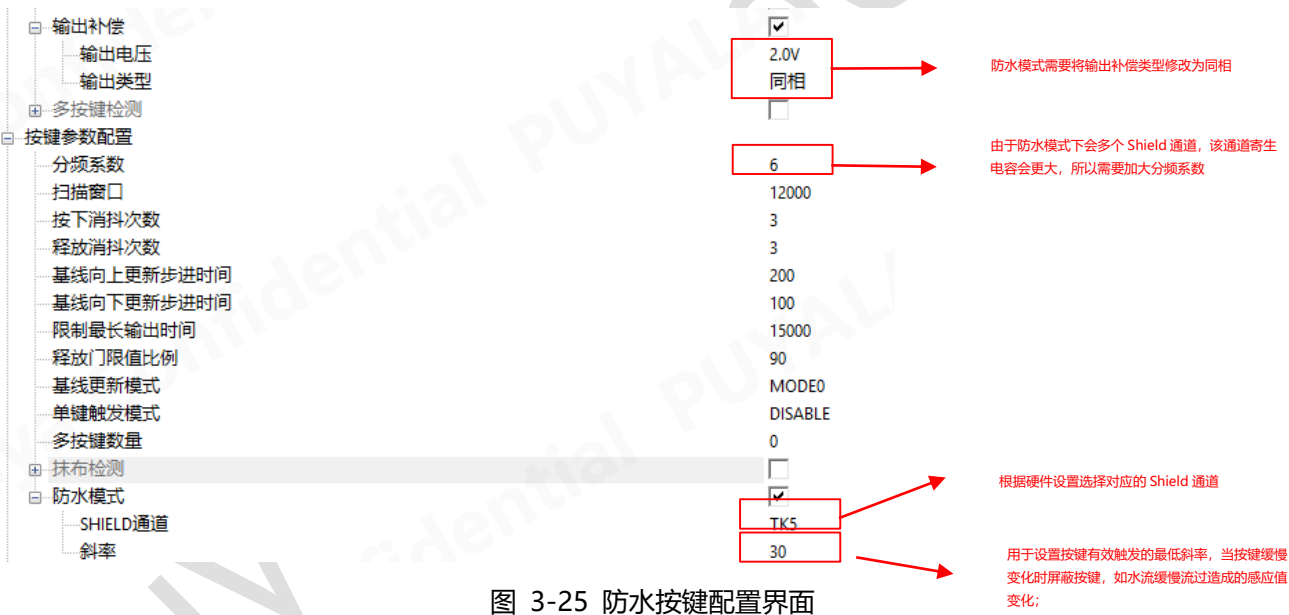


图 3-25 防水按键配置界面

Step 2: 在 app_config.h 中使能 PRINTF。



图 3-26 PRINTF 功能使能配置示例

Step 3: 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。

触摸按键	滑条/滑轮									
按键序号	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7	KEY8	
通道	TK0	TK1	TK12	TK6	TK7	TK11	TK8	TK9	TK10	TK_COM...
基准值	1934	1921	1900	1910	1891	1879	1896	1879	1903	1622
原始值	1934	1921	1900	1910	1891	1879	1895	1879	1903	1622
差值	0	0	0	0	0	0	-1	0	0	0
门限值	80	80	80	80	80	80	80	80	80	80
按键次数	1	0	0	0	0	0	0	0	0	0
状态										

图 3-27 防水按键调试界面及 Shield 通道数据

Step 4: 在无水状态下按下触摸按键，上位机左下角日志栏会打印如下信息。

日志：

清除

触摸库版本 2.1.0.2

chs[0] ratio:29 slopRate:101 Differ:233

dusterclothCnt:3

KeyFlag:0X1

打印含义为：通道 0 触发，与 Shield 通道的比例为 29，按键触发斜率为 101，触发时的按键变化量为 233

打印含义为：数据稳定时，按键通道比 Shield 通道先变化 3 个扫描周期，正常按键是同步变化，有抹布动作时会有先后顺序。

图 3-28 无水状态下触摸按键日志打印示例

Step 5: 在有水状态下按下触摸按键，上位机左下角日志栏会打印如下信息。

chs[0] ratio:156 slopRate:92 Differ:191

dusterclothCnt:3

打印含义为：通道 0 触发，与 Shield 通道的比例为 156，按键触发斜率为 92，触发时的按键变化量为 191

图 3-29 有水状态下触摸按键日志打印示例

Step 6: 根据实测得到的比例修改程序

tk_cfg.h

tk_user.c

py32_tk.h

tk_lib.h

startup_py32t020xx.s

mi

2318 #if WATER_PROOF_EN

2319 const uint16_t Ratio_Tbl[] =

2320 {

2321 80, //KEY0

2322 #if (KEY_CH_TOTAL > 1)

2323 80,

2324 #endif

2325 #if (KEY_CH_TOTAL > 2)

2326 80,

2327 #endif

2328 #if (KEY_CH_TOTAL > 3)

2329 80,

2330 #endif

2331 #if (KEY_CH_TOTAL > 4)

2332 80,

2333 #endif

2334 #if (KEY_CH_TOTAL > 5)

2335 80,

2336 #endif

2337 #if (KEY_CH_TOTAL > 6)

2338 80,

2339 #endif

2340 #if (KEY_CH_TOTAL > 7)

根据有水按键以及无水按键打印的比例取中间值进行设置，设置后多次测试查看参数是否设置准确，当比例设置过小时无水按键会当成有水按键进行处理，造成响应变慢或者松手响应，设置过大时有水按键会当成无水按键进行处理，造成多按键触发或误触。

图 3-30 防水按键比例参数配置代码示例

3.6.6. 防水按键应用（无 Shield 通道）

当硬件无 Shield 通道时，可通过软件进行基础防水逻辑处理

Step 1: 其他参数按照普通按键模式进行设置;

Step 2: 打开防抹布模式以及多按键检测。

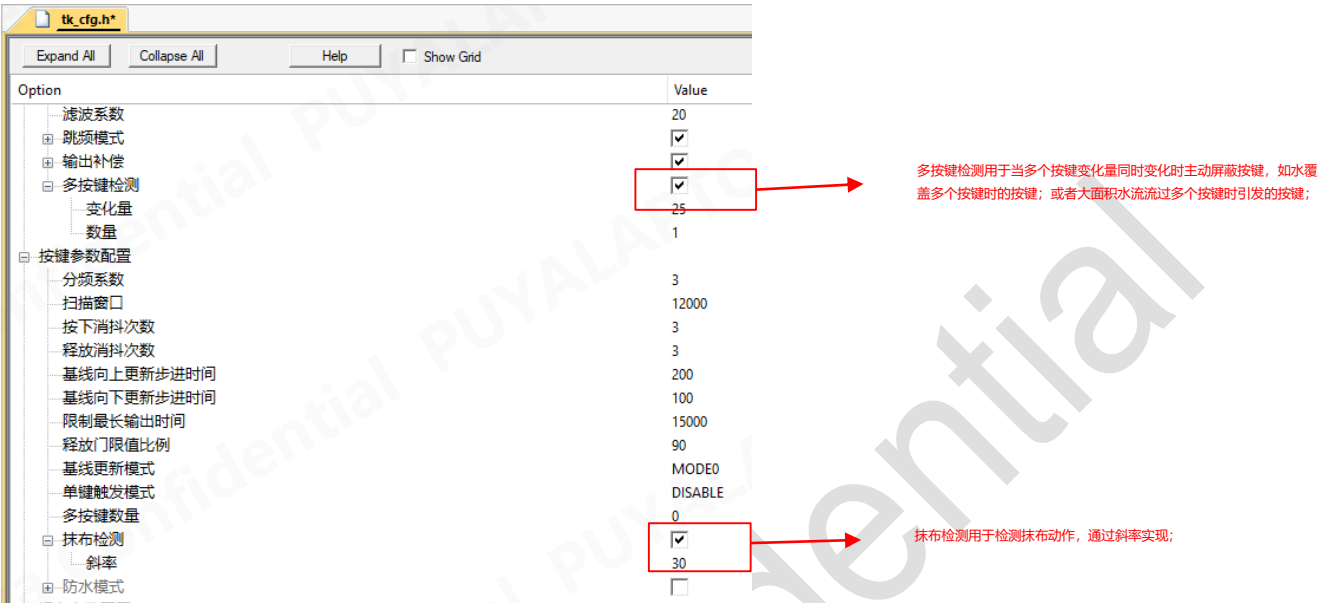


图 3-31 无 Shield 通道防水按键配置界面

Step 3: 在 app_config.h 中使能 PRINTF;



图 3-32 打印功能使能配置示例

Step 4: 装配好外壳后，将程序下载到目标板，连接 PY Touch - Link 后进行按键操作。



图 3-33 无 Shield 防水按键调试日志打印示例

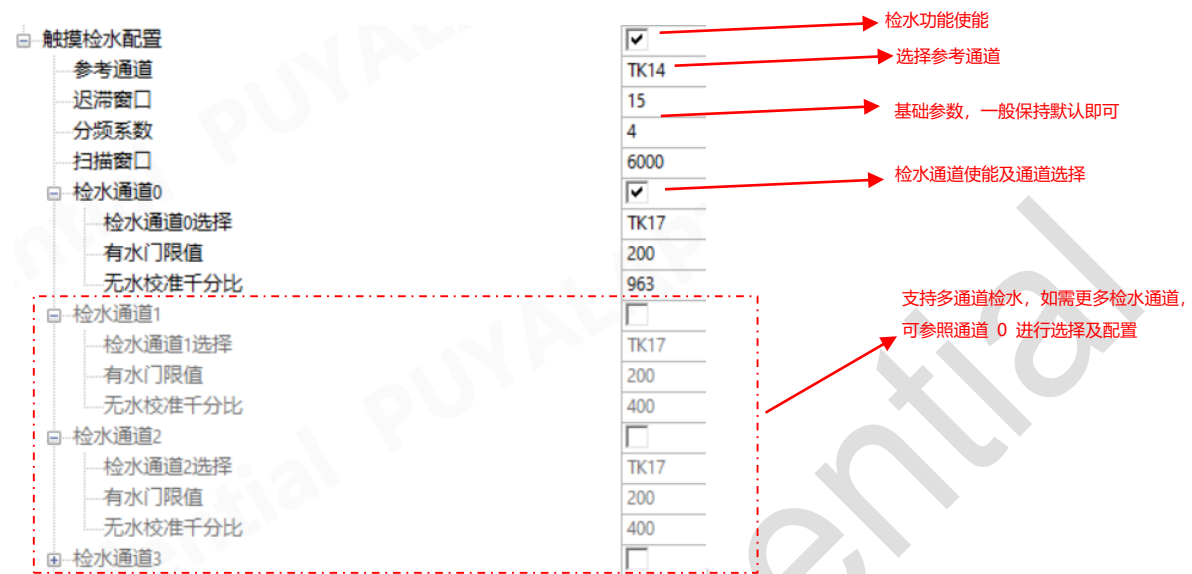
Step 5: 使用抹布进行模拟测试，查看打印的比例。



图 3-34 抹布测试日志打印示例

3.6.7. 触摸检水应用

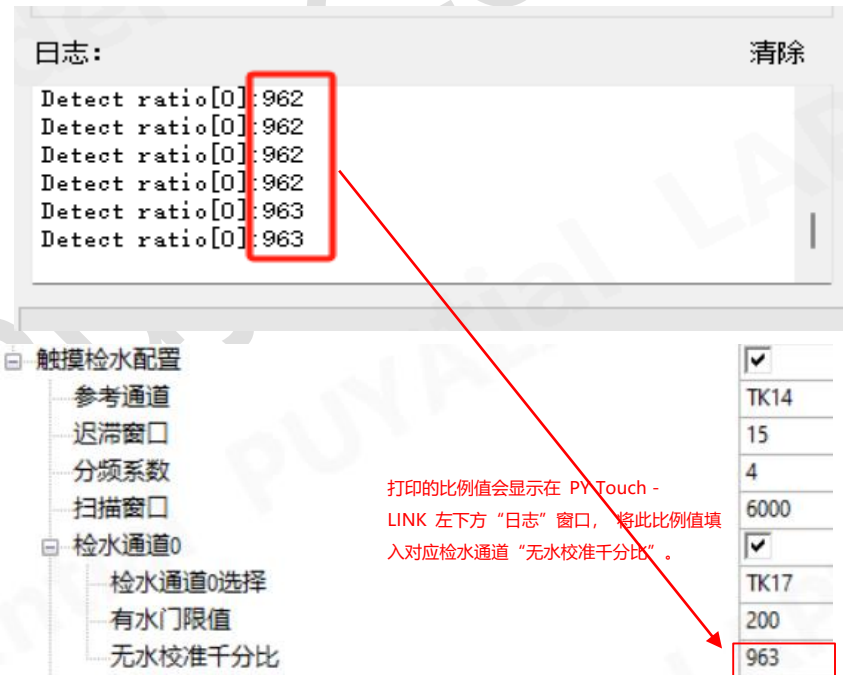
Step 1: 使能检水功能，配置参考通道、检水通道及基础参数。



Step 2: 在 app_config.h 中使能 PRINTF;



Step 3: 装配好外壳及容器，容器此时保持无水状态，将程序下载到目标板，连接 PY Touch - Link。



Step 4: 填写好无水状态的检准值，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
门限值	0	200
按键次数	0	9
状态		

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。

图 3-38 检水门限值计算

Step 5: 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: $230 \times 0.5 = 115$

图 3-39 检水门限值设置

Step 6: 重新下载程序，反复测试并微调门限值，直至达到最佳的检水效果。至此，检水功能调试完成，用户程序可根据检水状态执行相应操作。

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

图 3-40 检水状态读取及用户程序代码示例

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：

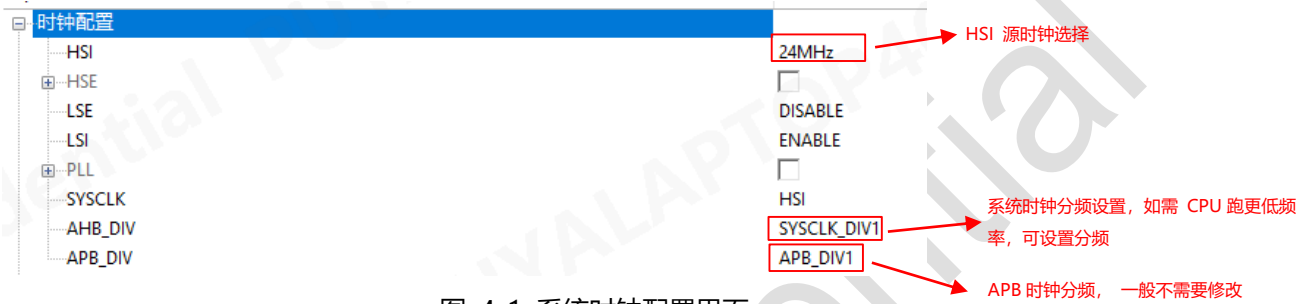


图 4-1 系统时钟配置界面

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

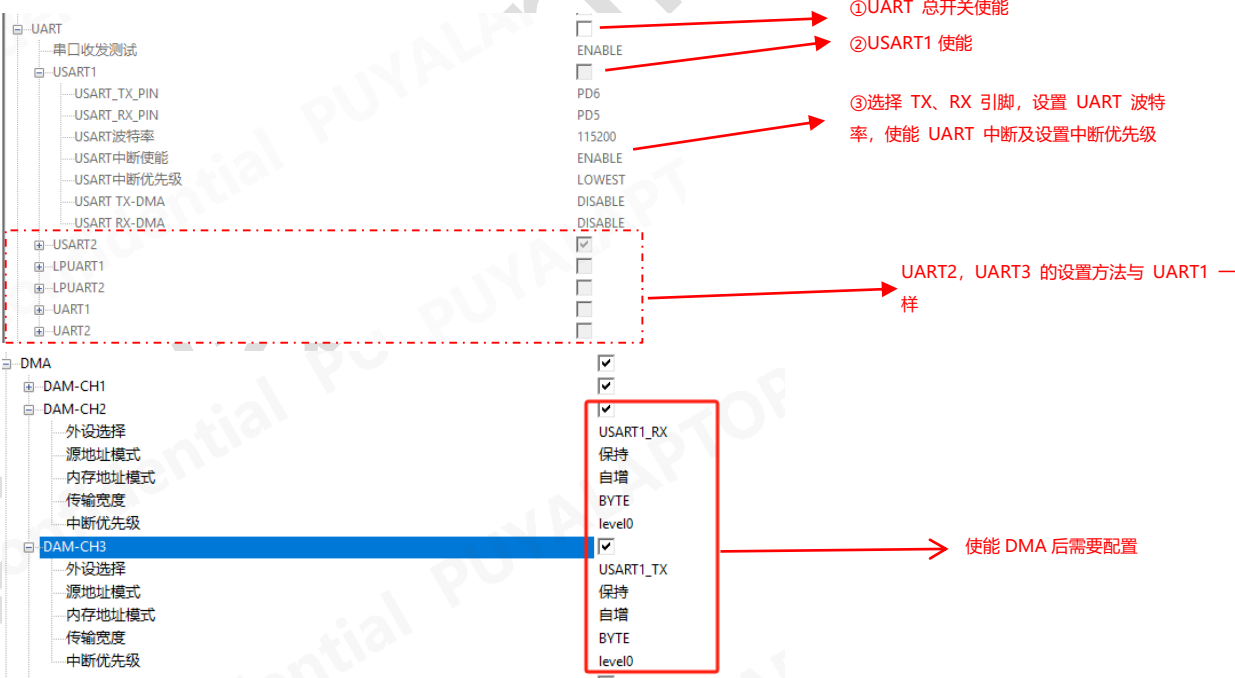


图 4-2 UART 配置界面

4.2.1.2. UART 函数接口说明

表 4-1 UART 函数接口说明

函数名	输入参数			返回值	备注
	参数名	数据类型	参数描述		
USARTx_QueueRead	data	uint8_t *	接收指针,有数据时会将收到的数据赋值到该内存	接收状态	调用该函数可实现 USARTx 一个字节数据的读取，当返回 1 时表示接收成功，返回 0 则无数据；
USARTx_QueueSend	data	uint8_t *	待发送的数据	发送状态	调用该函数可实现 USARTx 一个字节数据的发送，当返回 1 时表示发送成功，返回 0 则表示发送失败，队列满；
UARTx_QueueRead	data	uint8_t *	接收指针,有数据时会将收到的数据赋值到该内存	接收状态	调用该函数可实现 USARTx 一个字节数据的读取，当返回 1 时表示接收成功，返回 0 则无数据；
UARTx_QueueSend	data	uint8_t *	待发送的数据	发送状态	调用该函数可实现 USARTx 一个字节数据的发送，当返回 1 时表示发送成功，返回 0 则表示发送失败，队列满；
LPUSARTx_QueueRead	data	uint8_t *	接收指针,有数据时会将收到的数据赋值到该内存	接收状态	调用该函数可实现 USARTx 一个字节数据的读取，当返回 1 时表示接收成功，返回 0 则无数据；
LPUARTx_QueueSend	data	uint8_t *	待发送的数据	发送状态	调用该函数可实现 USARTx 一个字节数据的发送，当返回 1 时表示发送成功，返回 0 则表示发送失败，队列满；

注：以上 UART 函数接口存放于 uart_drivers.c。

4.2.1.3. UART 应用范例

UART

串口收发测试

ENABLE

```
void UART_Loop(void)
{
    #if ((USART1_RX_PIN != NO_PIN) && USART1_ENABLE)
        static uint8_t usartlrx_data[64];
        static uint8_t usartlrx_count = 0;
        while(USART1_QueueRead(&usartlrx_data[usartlrx_count]))
        {
            usartlrx_count++;
            usartlrx_timeout = 5;
            if (usartlrx_count == 64)
                break;
        }
        if ((usartlrx_count && usartlrx_timeout == 0) || usartlrx_count == 64)
        {
            #if (USART1_TX_PIN != NO_PIN)
                #if USART1_TXDMA_ENABLE
                    HAL_SCI_Transmit_DMA(&USART1_Handle,usartlrx_data,usartlrx_count);
                #else
                    /* 串口接收完成 */
                    for(uint8_t n = 0;n < usartlrx_count;n++)
                    {
                        USART1_QueueSend(usartlrx_data[n]);
                    }
                #endif
            #endif
            usartlrx_count = 0;
        }
    #endif
}
```

UART 测试使能，使能后芯片可连接 PC 与串口助手进行通信

可用串口助手发送数据，芯片接收 64 个字节数据，存放于 rx1_data 数组

将接收到的 64 字节数据发送回去，串口助手如接收的数据与发送的数据一致，证明 UART 收发正常。

图 4-3 UART 应用范例代码示例

4.2.2. 定时器

4.2.2.1. SYSTICK

SYSTICK 设置界面如下：

SYSTICK

定时时间(单位us)

SYSTICK中断优先级

中断服务程序IO口翻转测试

IO选择

1ms

LOWEST

☐

PC12

①设置定时时间及中断优先级设置

②如需测试定时功能，可勾选此项及选择测试引脚，勾选后，所选的 IO 会在 SYSTICK 中断服务程序里翻转，可外部用示波器或逻辑分析仪查看此 IO 波形来验证 SYSTICK 中断时间是否准确

图 4-4 SYSTICK 配置界面

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

图 4-5 SYSTICK 中断服务程序代码示例

4.2.2.2. LPTIME1、LPTIME2

LPTIME_x (x=1、2) 定时设置界面如下：

LPTIME1	☒	LPTIME1 使能
LPTIME1时钟源	LSI	时钟源可选为 LSI 或 LSE
LPTIME1中断产生时间	100	LPTIME1 定时时间，单位为 ms
LPTIME1中断使能	DISABLE	LPTIME1 中断使能设置
LPTIME1中断优先级	LOWEST	LPTIME1 中断优先级设置
☒ LPTIME1中断服务程序IO翻转测试	☐	LPTIME1 定时中断测试，可选择 IO 在中断
LPTIME2	☐	
LPTIME2时钟源	LSI	
LPTIME2中断产生时间	100	
LPTIME2中断使能	DISABLE	
LPTIME2中断优先级	LOWEST	
☒ LPTIME2中断服务程序IO翻转测试	☐	

图 4-6 LPTIME 配置界面示例

LPTIME 中断回调函数如下 (lptime_driver.c) :

```

/**
 * @brief LPTIM autoreload match interrupt callback function
 * @param None
 * @retval None
 */
void HAL_LPTIM_AutoReloadMatchCallback(LPTIM_HandleTypeDef *hlptim)
{
    #if LPTIME1_ENABLE
        if(hlptim == &LPTIME1_Handle)
        {
            /* LPTIME1回调 */
            #if (LPTIME1_DEBUG && (LPTIME1_DEBUG_GPIO != NO_PIN))
                GPIO_ToggleBit(LPTIME1_DEBUG_GPIO);
            #endif
        }
    #endif
    #if LPTIME2_ENABLE
        if(hlptim == &LPTIME2_Handle)
        {
            /* LPTIME2回调 */
            #if (LPTIME2_DEBUG && (LPTIME2_DEBUG_GPIO != NO_PIN))
                GPIO_ToggleBit(LPTIME2_DEBUG_GPIO);
            #endif
        }
    #endif
}

```

图 4-7 LPTIME 中断回调函数代码示例

4.2.2.3. TIMx (x=1、2、3、15、16、17)

功能互斥：TIMx 的 PWM、定时器和输入捕获功能不可同时使用。

TIMx (x=1、2、3、15、16、17) 定时设置界面相同，以 TIM1 举例如下：



图 4-8 TIM1 配置界面

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 timx_drivers.c，如下所示：

```

/**
 * @brief This function handles TIM1 Interrupt .
 */
void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
    /* TIM Update event */
    if (__HAL_TIM_GET_FLAG(&TIM1_Handle, TIM_FLAG_UPDATE) != RESET && __HAL_TIM_GET_IT_SOURCE(&TIM1_Handle, TIM_IT_UPDATE) != RESET)
    {
        __HAL_TIM_CLEAR_IT(&TIM1_Handle, TIM_IT_UPDATE);
        /* 测试GPIO翻转 */
        #if (TIM1_DEBUG_GPIO != NO_PIN && TIM1_DEBUG)
            GPIO_ToggleBit(TIM1_DEBUG_GPIO);
        #endif
    }
}

```

图 4-9 TIM1 中断服务程序

4.2.2.4. TIMx-PWM (x=1、2、3、15、16、17)

功能互斥：TIMx 的 PWM、定时器和输入捕获功能不可同时使用。

TIMx-PWM 设置界面如下所示：

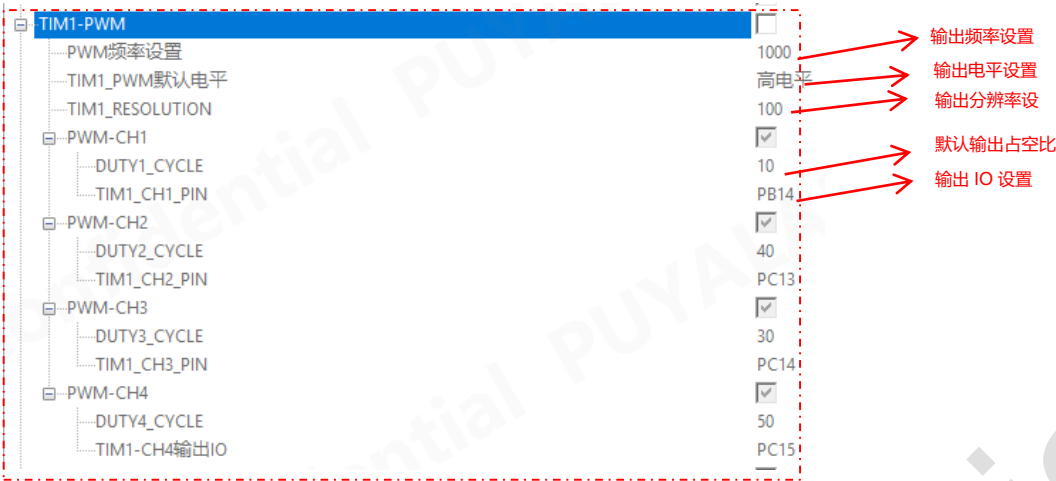


图 4-10 TIM1-PWM 配置界面示例

在应用中如需要调整 TIMx-PWM 占空比，可调用 TIMx_PWM_Pulse 接口函数进行设置。例如下图为 TIM1 占空比调节函数。

```

/*****
** 函数名   : void TIM1_PWM_Pulse(uint32_t CHx,uint16_t percent)
** 描述    : 定时器1-PWM占空比设置
** 传入    : CHx 通道号 TIM_CHANNEL_1~TIM_CHANNEL_4
              percent 输出百分比
** 返回    : 无
*****/
void TIM1_PWM_Pulse(uint32_t CHx, uint16_t percent)
{
    switch (CHx)
    {
        case TIM_CHANNEL_1:
            TIM1->CCR1 = TIM1_Autoreload * percent / TIM1_RESOLUTION;
            break;
        case TIM_CHANNEL_2:
            TIM1->CCR2 = TIM1_Autoreload * percent / TIM1_RESOLUTION;
            break;
        case TIM_CHANNEL_3:
            TIM1->CCR3 = TIM1_Autoreload * percent / TIM1_RESOLUTION;
            break;
        case TIM_CHANNEL_4:
            TIM1->CCR4 = TIM1_Autoreload * percent / TIM1_RESOLUTION;
            break;
    }
}

```

图 4-11 TIM1-PWM 占空比调节函数代码示例

4.2.2.5. TIMx-CAPTURE (x=1、2、3、15)

功能互斥：TIMx 的 PWM、定时器和输入捕获功能不可同时使用。

TIMx-CAPTURE (x=1、2、3、15) 设置界面相同，以 TIM1 举例如下：



图 4-12 TIM1 输入捕获配置界面示例

使用方式：

- (1) 配置 TIMx 为输入捕获模式 (如选择 TIM1-CAPTURE，信号输入管脚为 PC13)。
- (2) 输入待测信号。
- (3) 直接读取变量获取结果：

频率： 读取 TIMxCaptue.Freq (例如：TIM1Captue.Freq)

占空比： 读取 TIMxCaptue.Duty (例如：TIM1Captue.Duty)

在以下函数中，可获取输入捕获的占空比和频率。

```
void app_drivers_loop(void)
{
    /* 外部中断标志位 */
    if (EXTI_Flag != 0)
    {
        #if APP_IWDG_ENABLE
        #endif
        #if APP_ADC_ENABLE
        #endif
        #if (APP_UART_ENABLE && APP_UART_TEST)
        #endif
        #if (APP_I2C1_ENABLE || APP_I2C2_ENABLE)
        #endif
        #if (APP_SPI1_ENABLE || APP_SPI2_ENABLE)
        #endif
        #if APP_TIM1_INPUTCAPTURE_ENABLE
        if (TIM1Captue.Time)
        {
            TIM1Captue.Time--;
            if (TIM1_GetCapture(&TIM1Captue))
            {
                log_printf("Fre:%d Duty:%d\n", TIM1Captue.Freq, TIM1Captue.Duty);
            }
        }
        #endif
        #if APP_TIM2_INPUTCAPTURE_ENABLE
        if (TIM2Captue.Time)
        {
            TIM2Captue.Time--;
            if (TIM2_GetCapture(&TIM2Captue))
            {
                log_printf("Fre:%d Duty:%d\n", TIM2Captue.Freq, TIM2Captue.Duty);
            }
        }
        #endif
        #if APP_TIM3_INPUTCAPTURE_ENABLE
        if (TIM3Captue.Time)
        {
            TIM3Captue.Time--;
            if (TIM3_GetCapture(&TIM3Captue))
            {
                log_printf("Fre:%d Duty:%d\n", TIM3Captue.Freq, TIM3Captue.Duty);
            }
        }
        #endif
    }
}
```

图 4-13 输入捕获频率与占空比读取代码示例

4.2.3. PWM

PWM 设置界面如下所示：



图 4-14 PWM 配置界面

在应用中如需要调整 PWM 占空比，可调用 PWM_Pulse_Config 接口函数进行设置。

4.2.4. 看门狗

看门狗设置界面如下：



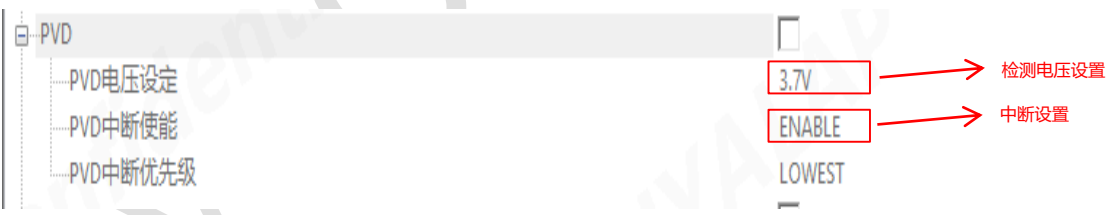
程序默认在主循环内喂狗。

```
7  }
8  /**
9  ** 函数名 void app_drivers_loop(void)
10 ** 描述 : 在主函数while(1)中轮询, 用于处理各模块的任务
11 ** 传入 : 无
12 ** 返回 : 无
13 **/
14 void app_drivers_loop(void)
15 {
16     /* 外部中断标志位 */
17     if (EXTI_Flag != 0)
18     {
19         log_printf("EXTI_Flag:0x%X\r\n", EXTI_Flag);
20         EXTI_Flag = 0;
21     }
22     #if APP_IWDG_ENABLE
23     /* 在设置喂狗的一半时间喂狗 */
24     if (Iwdg_Time > (IWDG_RELOAD_VALUE >> 1))
25     {
26         Iwdg_Time = 0;
27         /* Refresh IWDG */
28         if (HAL_IWDG_Refresh(&hiwdg) != HAL_OK)
29         {
30             APP_ErrorHandler();
31         }
32     }
33     #endif
34 }
```

图 4-15 看门狗配置界面及喂狗代码示例

4.2.5. 低电压检测 (PVD)

设置界面如下：



电压检测服务函数如下所示。

```
void HAL_PWR_PVDCallback(void)
{
    if (__HAL_PWR_GET_FLAG(PWR_FLAG_PVDO))
    {
        /* 检测到电源低于检测电压 */
        log_printf("LVD DOWN\r\n");
    }
    else
    {
        log_printf("LVD UP\r\n");
    }
}
```

图 4-16 PVD 配置及回调函数

4.2.6. ADC

ADC 的设置界面如下：

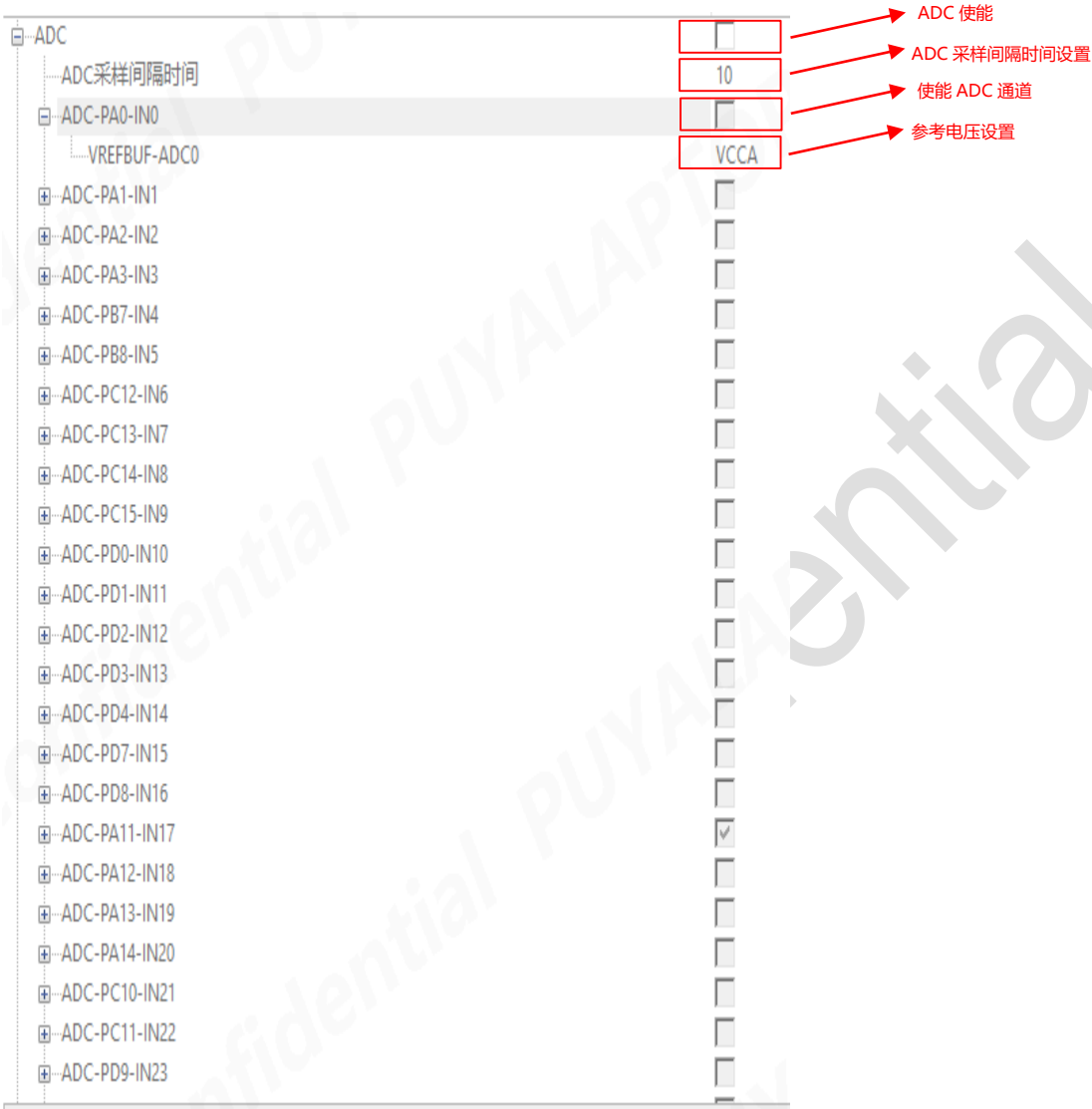


图 4-17 ADC 通道配置界面

采样完成会调用 ADC_Callback 函数。

```
#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
        log_printf("Vcc_Power = %d %d\r\n", Vcc_Power, ADCxConvertedData[ADC_CHANNEL_VREFINT_INDEX]);
    #endif
}
#endif
```

图 4-18 ADC 转换完成回调函数示例

4.2.7. DMA

DMA 设置界面，不能单独使用，需要配合外设使用。



图 4-19 DMA 通道配置界面

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：

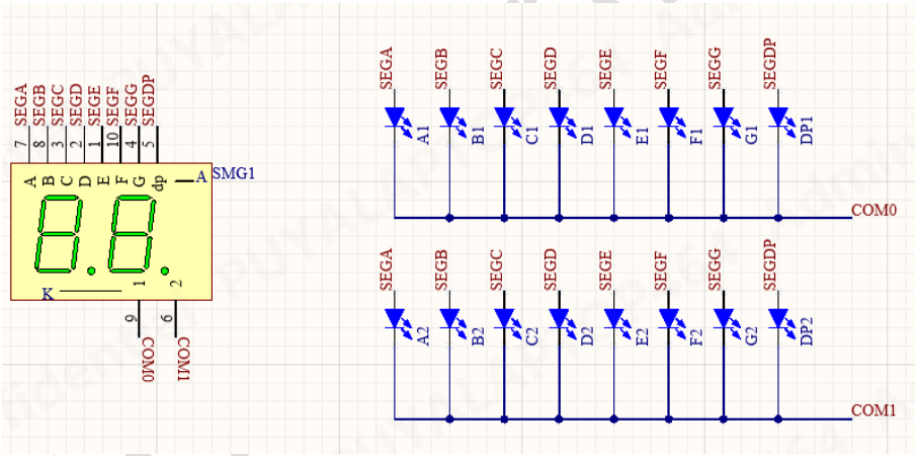


图 4-20 共阴极数码管典型电路

设置界面如下图：

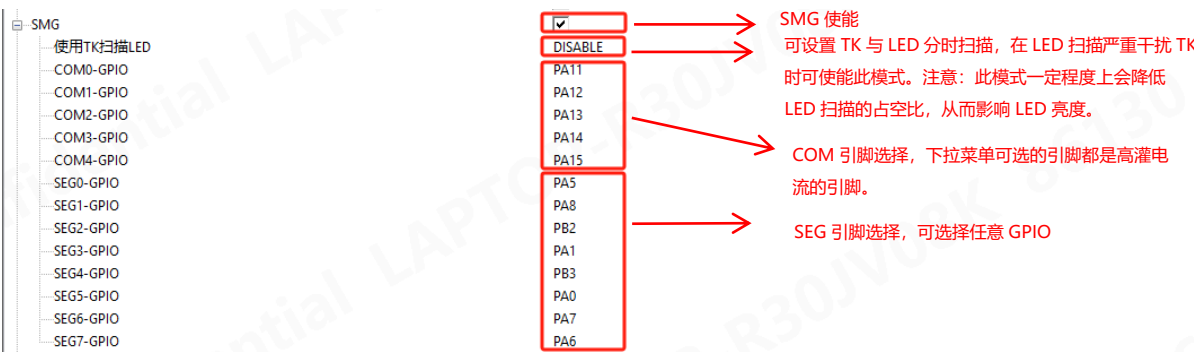


图 4-21 共阴极数码管/LED 设置界面

应用实例 (smg_task.c) 如下：



图 4-22 共阴极数码管/LED 应用实例

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

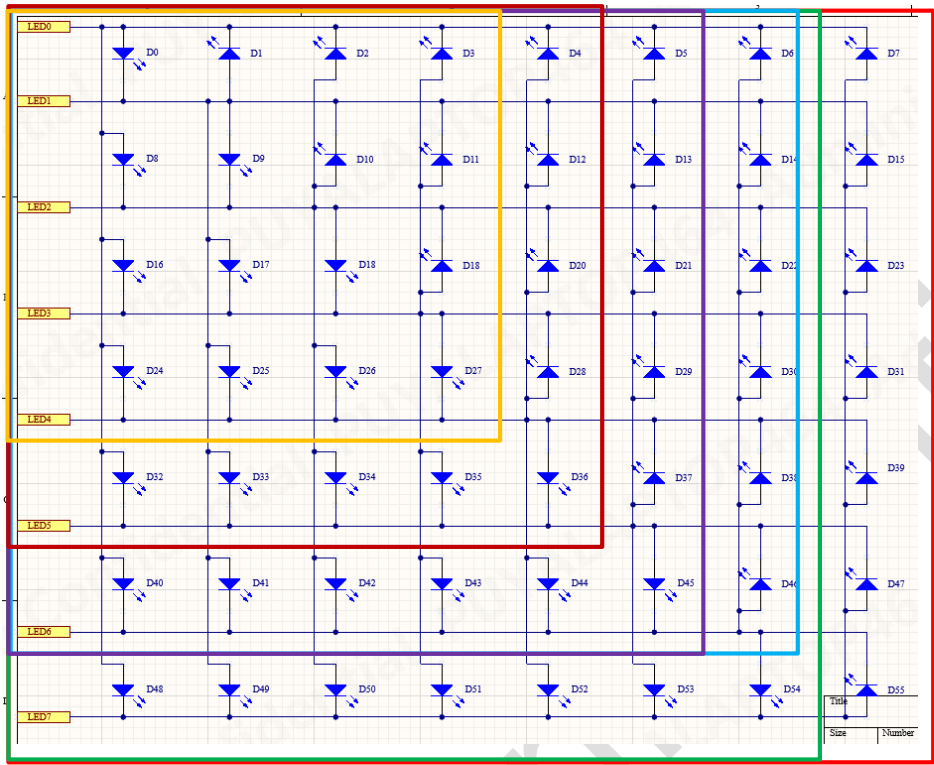


图 4-23 矩阵型 LED 典型电路图

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。

表 4-2 矩阵 LED 显示配置缓存

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：

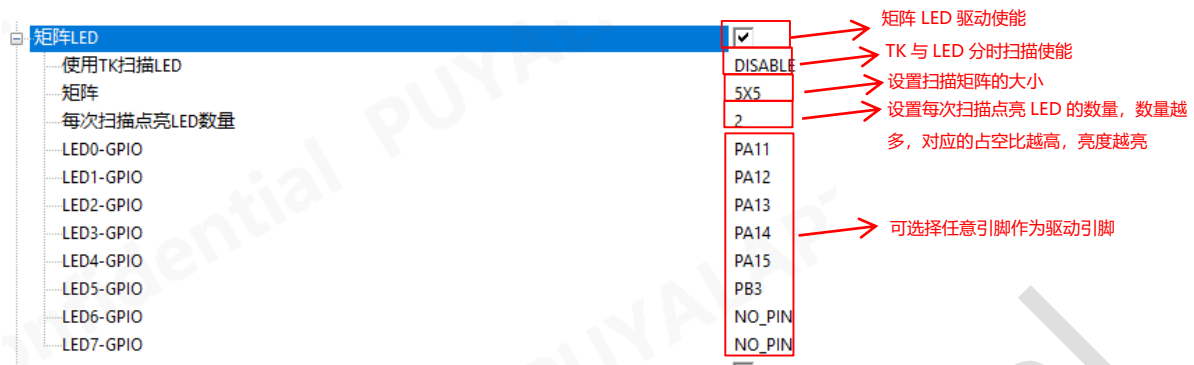


图 4-24 矩阵 LED 设置界面

4.3.2. 用户数据存储

设置界面及步骤如下：

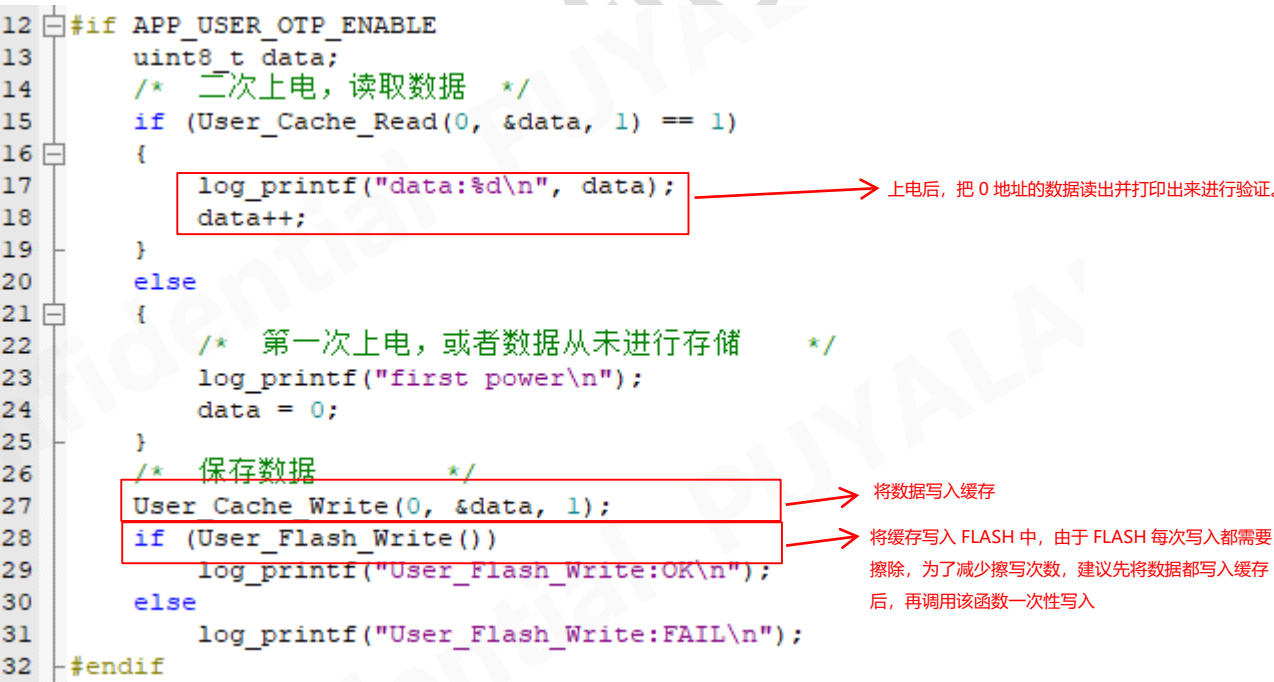
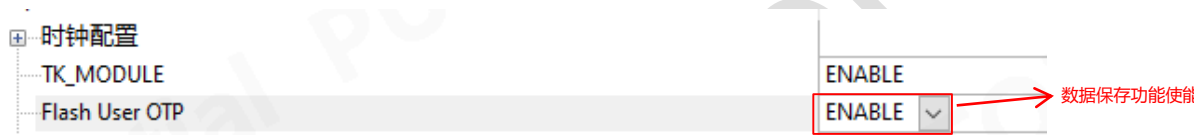


图 4-25 用户数据存储配置及代码示例

4.3.3. 类 WS2812B 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。

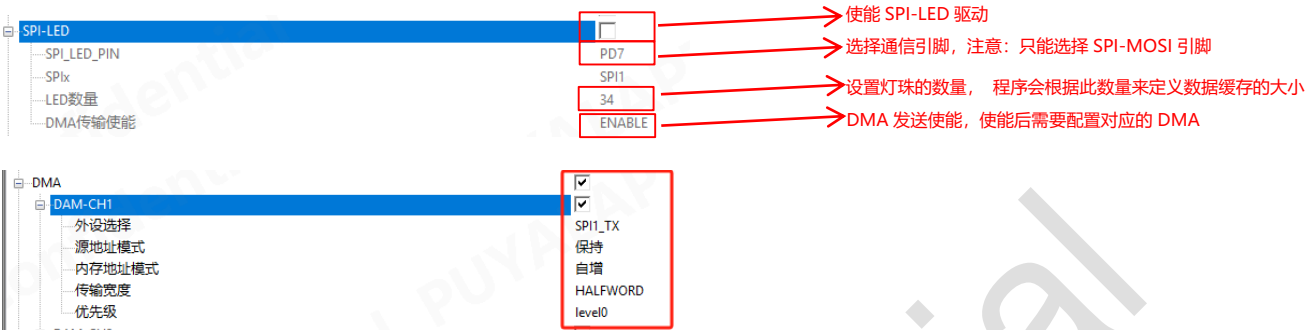


图 4-26 类 WS2812B 幻彩灯驱动配置界面

在应用中，应用程序首先将待发送的数据通过 SPI_LED_RgbLoad 函数装载于缓存数组 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)
{
    #if (SPI_LED_DMA_ENABLE == 1)
        /* Transmit and Receive an amount of data in non-blocking mode with IT */
        if (HAL_SPI_Transmit_DMA(&LED_SPI_Handle, (uint8_t *)rgb_data, SPI_LED_CNT * 6) != HAL_OK)
        {
            APP_ErrorHandler();
        }
    #else
        if (HAL_SPI_Transmit(&LED_SPI_Handle, (uint8_t *)rgb_data, SPI_LED_CNT * 6, 1000) != HAL_OK)
        {
            APP_ErrorHandler();
        }
    #endif
}
```

图 4-27 SPI_LED 数据传输函数

4.3.4. 蜂鸣器控制

蜂鸣器的设置界面如下：



图 4-28 蜂鸣器控制配置界面

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(uint16_t timeout)
** 描述    : 打开蜂鸣器
** 传入    : timeout 开启时间ms
** 返回    : 无
*****/
void BEEP_On(uint16_t timeout)
{
    beep_delay = timeout;
#ifdef BEEP_ADC
    ntc_delay = 5;
    GPIO_Init(BEEP_GPIO, GPIO_MODE_OUTPUT_PP, GPIO_NOPULL, 0);
#endif
#ifdef BEEP_GPIO_TYPE
    GPIO_SetBit(BEEP_GPIO);
#else
    GPIO_ClearBit(BEEP_GPIO);
#endif
}

```

图 4-29 BEEP_On 函数代码示例

应用实例：

```

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

```

图 4-30 按键触发蜂鸣器应用代码示例

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：

表 5-1 用户应用程序接口函数表

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

版本	日期	更新记录
V1.0	2025.03.20	1. 初版
V1.1	2026.05.19	1. 更新触摸库及应用层接口开发说明



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