



Technology Innovator

Puya

PY32T09x Series Touch Application Development Guide



Puya Semiconductor (Shanghai) Co., Ltd.

Content

1. Introduction	4
2. Software Library Structure.....	5
3. Introduction and Usage of the Touch Library	9
3.1. Introduction to Touch Function Categories	9
3.2. MCU resource requirements	9
3.2.1. MCU hardware resources.....	9
3.2.2. MCU software resources	9
3.3. Touch Library Interface Description	11
3.3.1. Data Interface	11
3.3.2. Function Interface	11
3.4. Touch library function options and parameter configuration	13
3.4.1. Touch key user configuration	13
3.4.2. Slider/Wheel user configuration.....	14
3.4.3. Touch function and parameter configuration	15
3.5. User-modifiable touch library callback functions	17
3.6. Touch application configuration steps and process	19
3.6.1. General key application	19
3.6.2. Wheel/Slider Application.....	27
3.6.3. Touch Library Low-Power Application.....	29
3.6.4. Floating Button Application	30
3.6.5. Waterproof Button Application (with Shield Channel)	31
3.6.6. Waterproof key application (without Shield channel)	32
3.6.7. Touch water detection application	34
4. System configuration and peripheral application analysis	36
4.1. System Configuration	36
4.1.1. Clock configuration	36
4.2. Interfaces and usage of peripheral applications.....	36
4.2.1. UART	36
4.2.2. Timers.....	39
4.2.3. PWM	43
4.2.4. Watchdogs.....	44
4.2.5. PVD (Low Voltage Detection).....	45
4.2.6. ADC	45
4.2.7. DMA.....	46
4.3. Typical Application Drivers	47
4.3.1. Digital Tube / LED Driver	47
4.3.2. User data storage	49
.....	50

4.3.3.	WS2812B-like LED driver	50
4.3.4.	Buzzer Control	51
5.	User Application Programming Interface	52
6.	Revision History	53

Puya Confidential

1. Introduction

The PY32T09x Series Standard Software Library (PY32T09x_Touch_Library_Vxx) includes system configuration, the touch library, peripheral libraries (HAL library), application examples based on the HAL library, and commonly used external device drivers. This project uses a visual configuration interface, allowing system configuration, touch, peripherals, and external device initialization and parameter configuration to be completed by selecting options and entering parameters. Based on this project for application development, users do not need to focus on the initialization process of the system, peripherals, and TK related to the chip level. They only need to call the corresponding module functions and data interfaces to use the chip's functions. This allows users to focus solely on application development related to product functions, making the development process more intuitive and efficient. Developers unfamiliar with Puya MCUs can also easily get started.

This document will introduce in detail the definitions and usage methods of each functional module in this project, enabling developers to gain a comprehensive understanding of the application development process based on the standard project.

2. Software Library Structure

The basic structure of the software library is shown in the figure below:

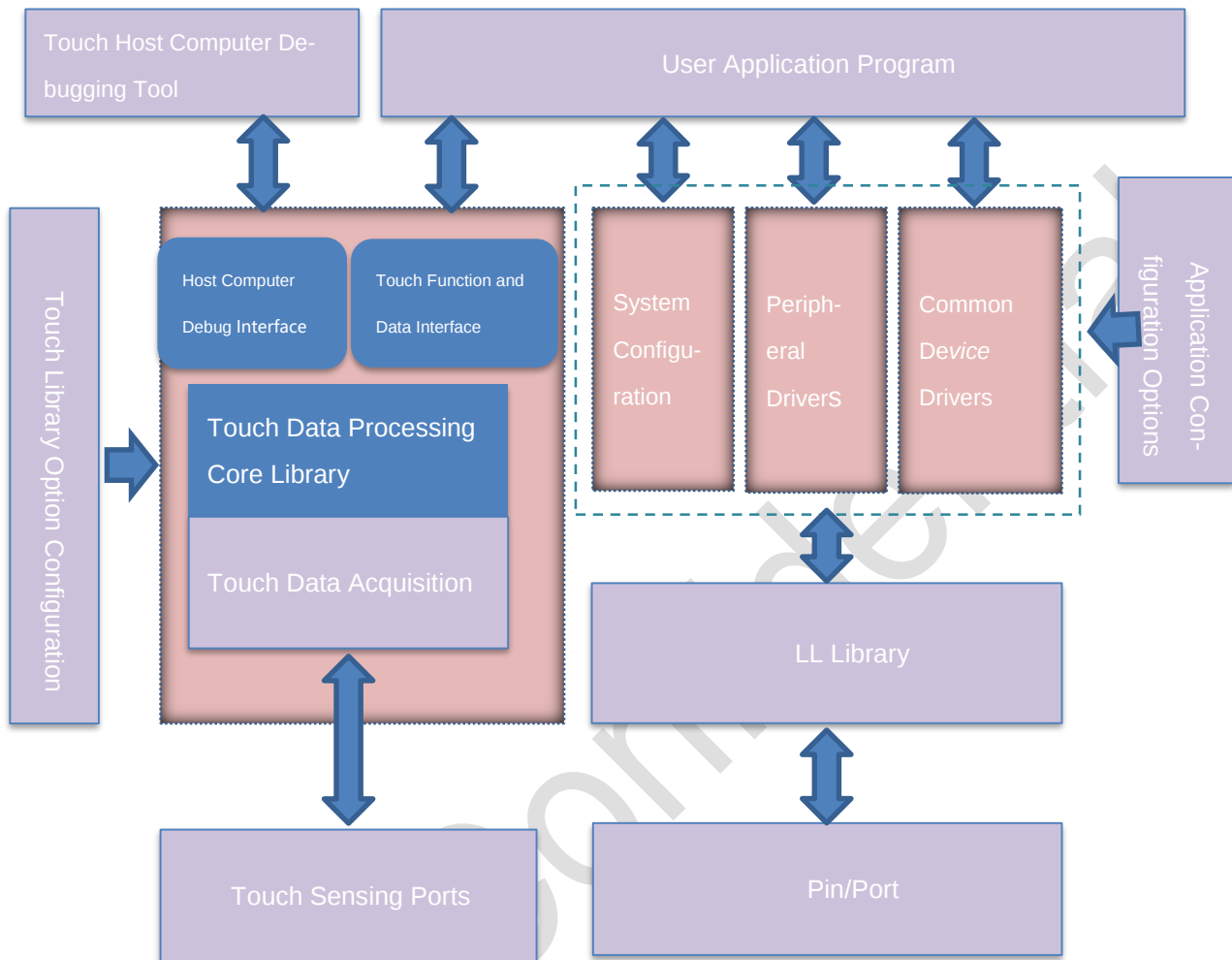


Figure 2-1 Software library basic structure

The correspondence between the main modules in the above software structure diagram and the software project directories is as follows:

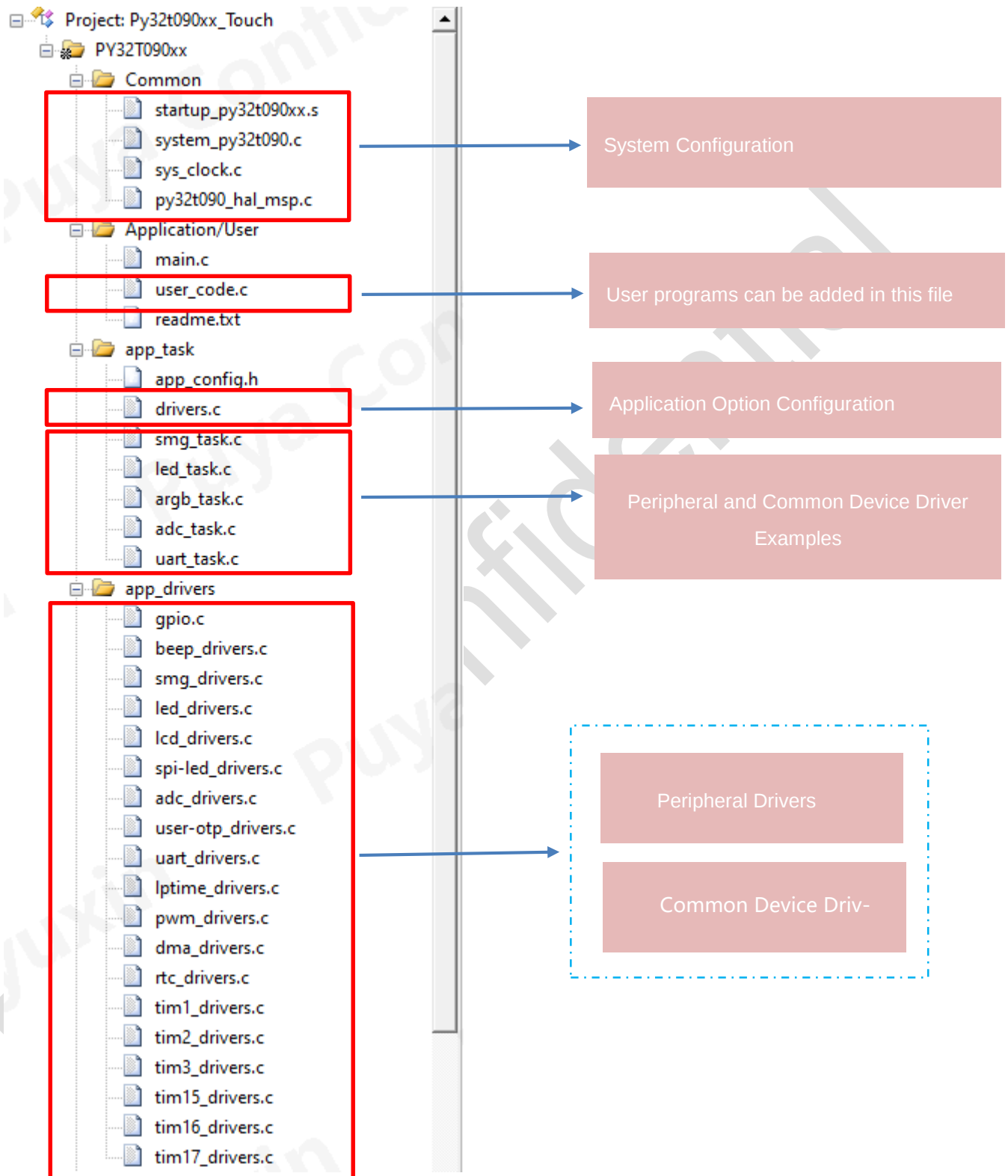


Figure 2-2 Project directory and driver layer structure

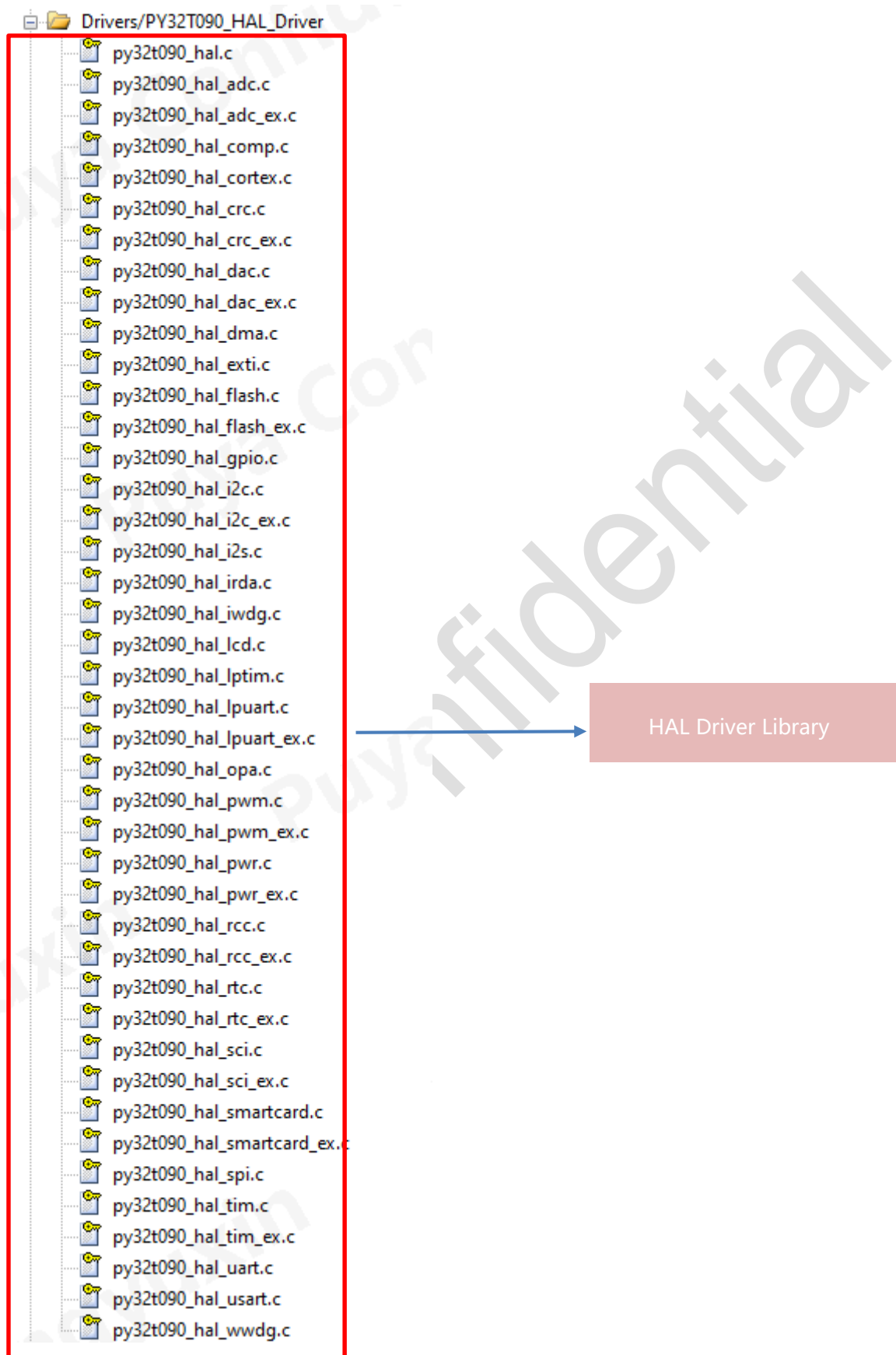


Figure 2-3 HAL driver layer file structure

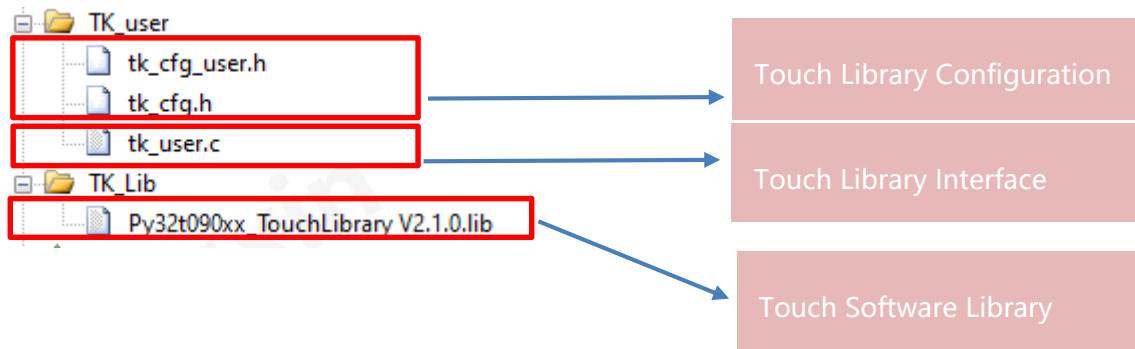


Figure 2-4 Touch library software module structure

3. Introduction and Usage of the Touch Library

3.1. Introduction to Touch Function Categories

The touch library supports the following applications:

- Conventional buttons (sensing elements: springs, PADs, conductive foam, etc.)
- Waterproof buttons
- Wheel/Slider
- Floating control
- Touch water detection
- Low-power applications (floating touch and touch water detection do not support touch wake-up)

3.2. MCU resource requirements

3.2.1. MCU hardware resources

The PY32 touch library requires the following hardware resources:

- SYSTICK timer (non-exclusive)
- One GPIO per touch channel
- The waterproof Shield channel occupies one channel IO
- RTC is required in low-power mode

3.2.2. MCU software resources

FLASH and SRAM resources used by the PY32 touch library are shown in the table below:

Table 3-1 Touch function and channel configuration

Touch function	Flash	SRAM	Channels
Conventional buttons	9.36K	1.83K	8(Key)
Floating buttons	9.34K	1.77K	6(Key)
Waterproof buttons	11.1K	2.05K	9(Key)+1(Shield)
Wheel + Slider + Button	12.3K	1.9K	2(Key)+4(Wheel)+4(Slider)
Slider + Wheel	11.02K	1.83K	4(Wheel)+4(Slider)
Touch Water Detection	9.94K	1.72K	1(Ref)+1(Detect)

Note: 1) SRAM usage in the above table includes a 512 Bytes stack

2) Each additional channel increases SRAM by about 30 Bytes

Table 3-2 Touch function timing test data

Touch Function	TK frequency Frequency (MHz)	Channel	Window Setting	Single-Channel Scan Time (us)	TK Interrupt Re- sponse Time (us)	TK Data Pro- cessing Time (us)
Spring Button	24	8	12000	1000	545	270
	48			500	390	200
Floating Button	24	6	30000	2500	405	205
	48			1250	290	150
Waterproof Button	24	8+1	12000	1000	700	502
	48			500	500	405
Slider/Slide Bar	24	4	12000	1000	250	100
	48			500	195	70
Water Detec- tion	24	2	12000	500	13	60
	48			250	10	53

Note:

1. TK sampling time is directly proportional to the window setting value and inversely proportional to the touch clock frequency.
2. With the main frequency fixed, the TK interrupt time and TK data processing time are basically proportional to the number of channels.
3. Taking conventional key scanning with a 24M main frequency as an example, the total sampling time for 8 channels is: $8 * 1000\mu s = 8000\mu s$. The interrupt response + data processing time is: $545 + 270 = 815\mu s$. Since TK sampling does not occupy CPU time, the proportion of CPU time used by the TK task is: $815/8000 = 10.1\%$.

3.3. Touch Library Interface Description

3.3.1. Data Interface

Table 3-3 Touch library data interface

Application Category	Variable Name	Variable Type	Description
Standard Keys	TKCtr.KeyFlags	unsigned int	TKCtr.KeyFlags is a 32-bit unsigned number. Each bit represents the trigger status of a touch key. bit0~bit31 correspond to KEY0~KEY31 respectively. The user program can check the value of the corresponding bit to obtain the key status and perform the corresponding operation.
Waterproof Button			
Floating Button			
Slider/Wheel	TKCtr.SliderOrWheelPosition[n]	signed short int	TKCtr.SliderOrWheelPosition[n] indicates the position information of the nth slider/wheel. When the slider/wheel is triggered, the position range is: 0 to the set resolution. When the value of TKCtr.SliderOrWheelPosition[n] is -1, it indicates that the slider/wheel is not triggered.
Touch Water Detection	TKCtr.DetectOut	unsigned char	TKCtr.DetectOut is an 8-bit unsigned number. Each bit represents the water level status of a water detection channel. When it is 1, it indicates that the corresponding channel has water.

3.3.2. Function Interface

Table 3-4 Touch library function interface

Function Name	Input Parameters	Return Value	Descriptions
TK_Init	None	None	Touch function initialization API function, called before entering the main loop.
TK_MainFsm	None	None	Touch function main state machine API function, called in the main loop.
TK_Timer-Handler	uint8_t ms, ms indicates the time interval for calling this function, in milliseconds.	None	This function is called in the timer interrupt service routine to provide the time base for the touch function.

```

int main(void)
{
    /*Prioritize initialization of printing*/
    #if (APP_TK_ENABLE && DEBUG_ENABLE && (DEBUG_MODE == 0))
        log_init();
    #endif
    HAL_Init();
    /* System clock configuration */
    SystemClockConfig();
    /* System configuration initialization code begins */
    app_drivers_init();
    /* System configuration initialization code ends */
    #if APP_TK_ENABLE
        TK_Init();
    #endif
    user_init();
    /* infinite loop */
    log_printf("Start Loop\n");
    while (1)
    {
        #if APP_TK_ENABLE
            /* Touch state machine processing */
            TK_MainFsm();
            /* Touch key processing */
            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();
}

```

Figure 3-1 Touch library function call code example

3.4. Touch library function options and parameter configuration

3.4.1. Touch key user configuration

For touch key applications, the touch channels and key trigger threshold values should first be configured. These two configurations are implemented in `tk_cfg_user.h`, as shown below:

```
#ifndef _TK_CFG_USER_H
#define _TK_CFG_USER_H

//Key touch channel definition, fill in according to the
#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
//-----

//-----
//Key trigger threshold definition
#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
```

CH_KEYn corresponds to KEYn_THD. CH_KEYn is the channel definition corresponding to KEYn, and KEYn_THD is the trigger threshold value of KEYn. When the channel data variation defined by CH_KEYn is greater than KEYn_THD, KEYn will be triggered.

Figure 3-2 Touch key channel and threshold configuration code example

3.4.2. Slider/Wheel user configuration

Slider/Wheel applications need to configure the application type, channels, resolution, and trigger threshold values. These are defined in `tk_cfg_user.h`, as shown below:

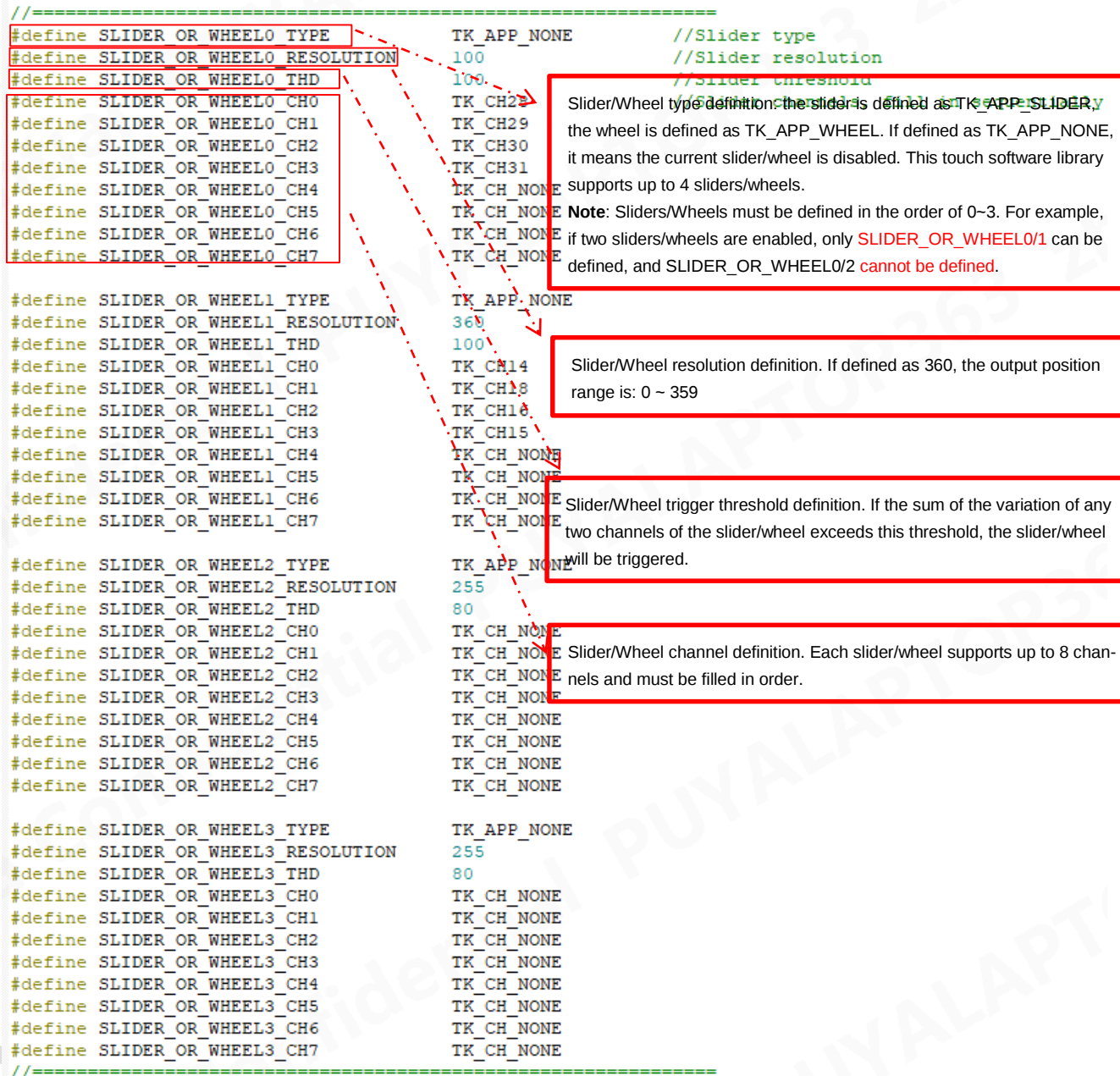


Figure 3-3 Slider/wheel configuration diagram

3.4.3. Touch function and parameter configuration

Touch parameters and function selection can be defined in tk_cfg.h. In actual applications, it is not necessary to directly edit the contents of tk_cfg.h; instead, configuration is done in its configuration wizard. The configuration wizard is a graphical configuration menu where settings can be completed by checking options, selecting from drop-down menus, or entering parameters.

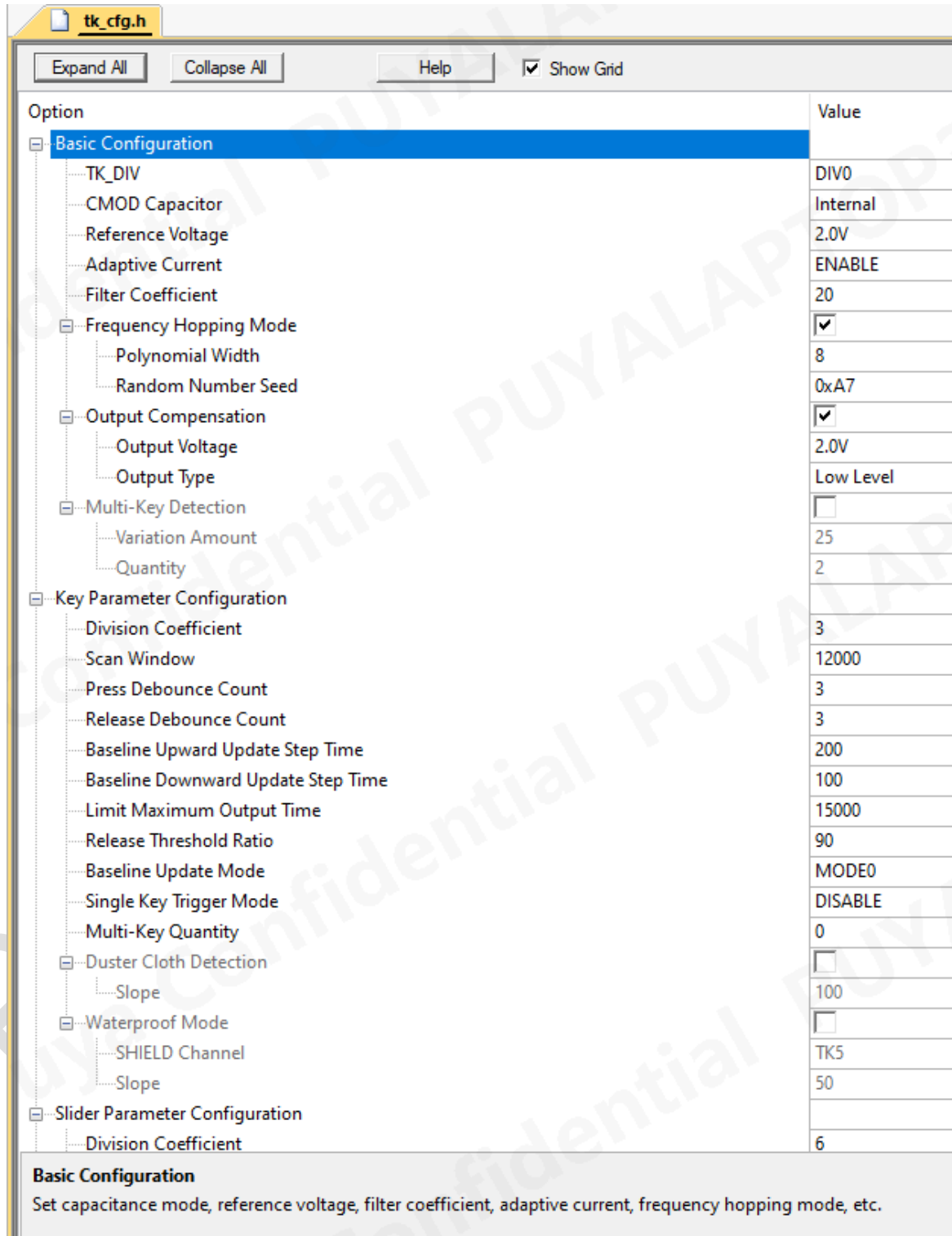


Figure 3-4 Touch regular function configuration

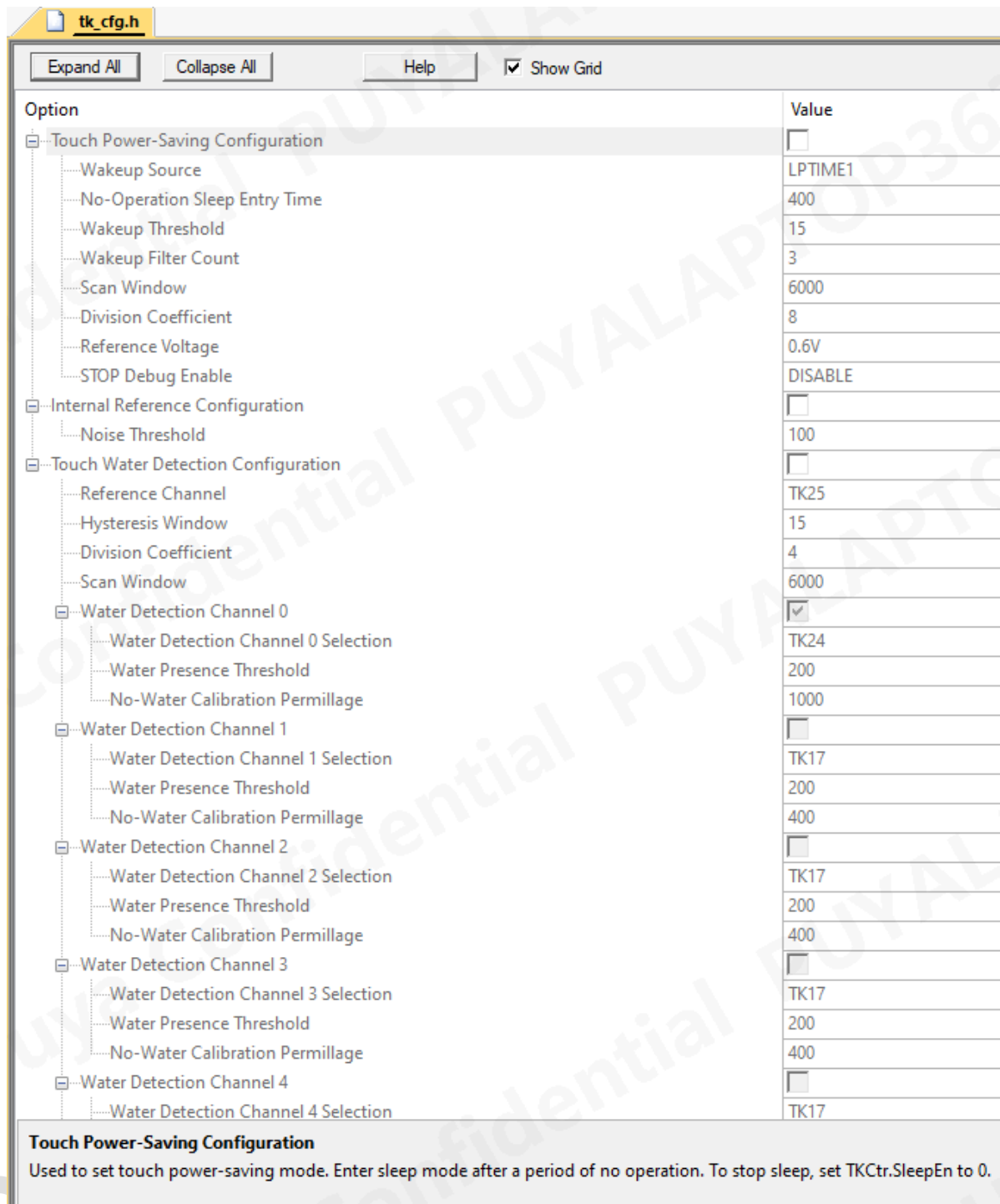


Figure 3-5 Touch special function configuration

3.5. User-modifiable touch library callback functions

The touch software library exposes some function interfaces to users in the form of callback functions so they can be modified to meet more application requirements. These callback functions are stored in the tk_user.c file. Common callback functions are listed in the table below:

Table 3-5 Touch library user callback functions

Function Name	Input Parameters	Return Value	Descriptions
EnterStop_Callback	None	None	Called before the touch enters low-power mode. If the user program needs to perform operations before entering low-power mode, the corresponding code can be added in this function.
ExitStop_Callback	None	None	Called after the touch exits low-power mode. If the user program needs to perform operations after exiting low-power mode, the corresponding code can be added in this function.
LoopStop_Callback	None	None	Called after data acquisition wakes up in low-power mode. If the user program has tasks that need to be processed periodically during low-power mode, the corresponding code can be added in this function.
TK_RawDataCompensate	Parameter name: chs Data type: uint8_t Parameter meaning: channel index Parameter name: raw Data type: uint16_t Parameter meaning: touch data	Return value: Compensated touch data	Used to compensate touch data. For example, when the LED turns on or off, the touch data may change accordingly. In this case, the touch data can be synchronously compensated in this function according to the LED state.
FilterExDeal	Parameter name: chs Data type: uint8_t Parameter meaning: channel index	None	This function is called back when filtering touch data; the filter parameters can be adjusted in this function according to actual measurements.
APP_TouchKeyFlagsMask	None	None	This function is used to determine and handle abnormal conditions. For example, after a key event is generated, check whether interference exists at that moment; if there is interference, discard the generated key event.

Function Name	Input Parameters	Return Value	Descriptions
TK_ScanDone	None	Return value: Whether it is necessary to switch the touch mode and reinitialize	This function is called after touch data acquisition is completed to obtain information on whether the touch mode needs to be switched.
TK_UpdatBase-LineData	Parameter name: chs Data type: uint8_t Parameter meaning: channel index	None	After calling, the baseline value of this channel is forcibly updated, suitable for actively releasing the key state when an abnormal condition is detected.

3.6. Touch application configuration steps and process

3.6.1. General key application

Step 1: Set the TK channel corresponding to the key. There are two methods:

Method 1: Directly fill in the TK channels in order in tk_cfg_user.h, as follows:

The screenshot shows the `tk_cfg_user.h` file with the following definitions:

```

1 #ifndef _TK_CFG_USER_H
2 #define _TK_CFG_USER_H
3
4 // Key touch channel definition, fill in ac
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
  
```

A red box highlights `TK_CH9` in the definition, and a red dashed arrow points to the `TK_IN9` pin in the pin configuration table below.

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

Method 2: Configure the TK channels in the touch debugging host tool PY Touch - Link.exe, then export `tk_cfg_user.h` and overwrite the original file.

The screenshot shows the PY Touch - Link v1.0.9 software interface. The 'Select Model' dropdown is set to 'PY32T020G16S7'. The pin configuration table on the left shows the following connections:

Pin	Function
VCC 1	TD0
TK25 2	TK1 KEY2
VSS 3	TK2 KEY5
TK24 4	TK3 KEY4
TK23 5	TK4
TK22 6	TK5 KEY1
TK21 7	TK6
TK20 8	TK7
TK19 9	TK8 KEY3
TK18 10	TK9 KEY0
TK17 11	TK10
TK16 12	TK11
TK15 13	TK12
TK14 14	TK13

Red arrows point from the 'TK9 KEY0' entry in the table to the 'KEY0' column in the 'Touch Key' settings table and to the 'TK9' channel selection in the 'Channel' dropdown.

The 'Touch Key' settings table is as follows:

Key Number	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6
Channel	TK9	TK5	TK1	TK8	TK3	TK2	TK17
Baseline	0	0	0	0	0	0	0
Raw Data	0	0	0	0	0	0	0
Differ	0	0	0	0	0	0	0
Threshold	50	50	50	50	50	50	50
Key Count	0	0	0	0	0	0	0

Red annotations include:

- ① Select chip model
- ② Double-click the mouse on the selected TK channel pins in order
- ③ Double-click the threshold value cell of the corresponding key to set the initial threshold value

Figure 3-6 Touch key channel configuration method 1

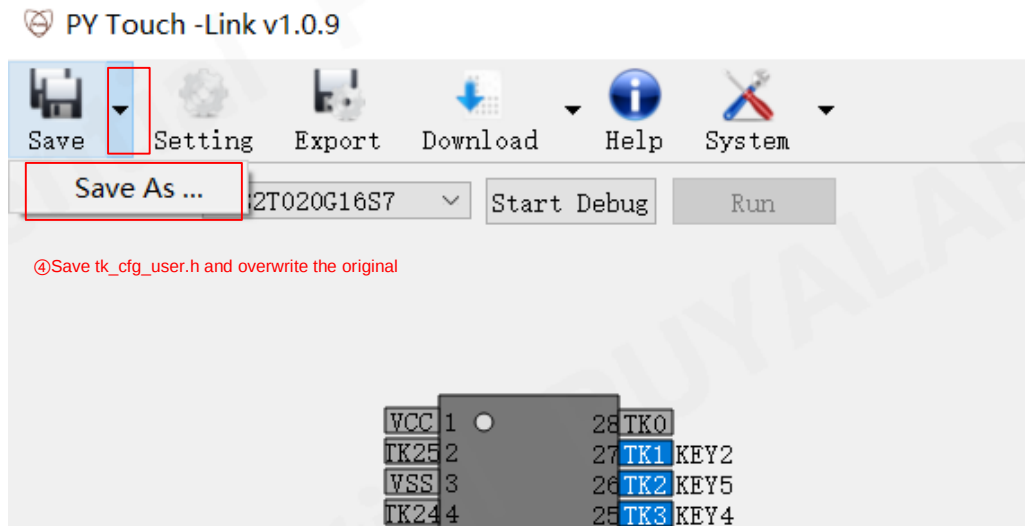


Figure 3-7 Touch key channel configuration method 2

Step 2: Compile the program. After compilation is completed, download it to the target board. There are two methods to download the program:

Method 1: Download directly within the KEIL environment, as shown below: (Note: this download method requires the download pins to remain configured as the SWD port function)

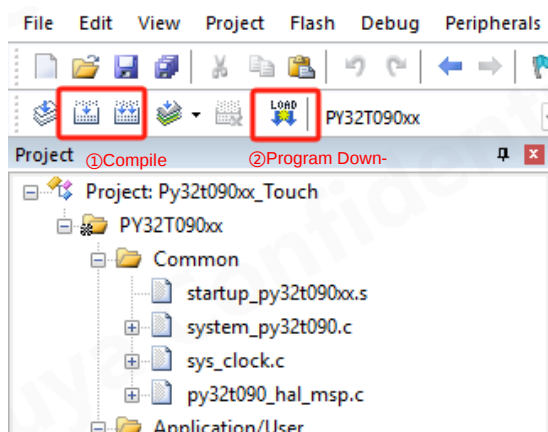


Figure 3-8 Program download method - KEIL download

Method 2: Use PY Touch - Link.exe to download, as shown below:

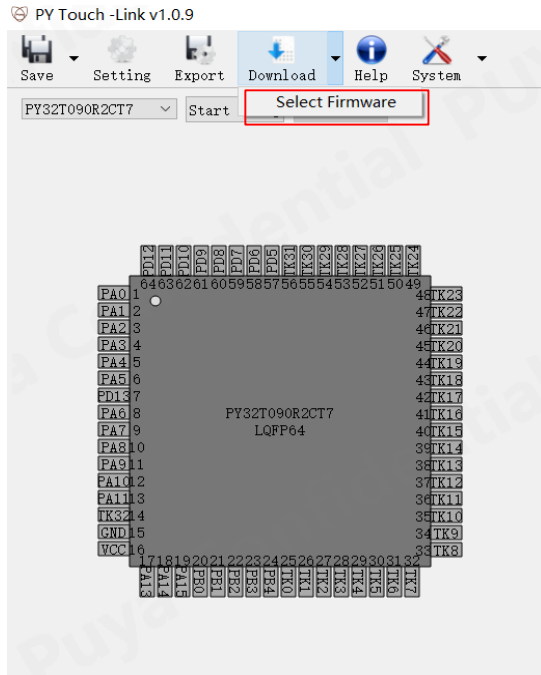
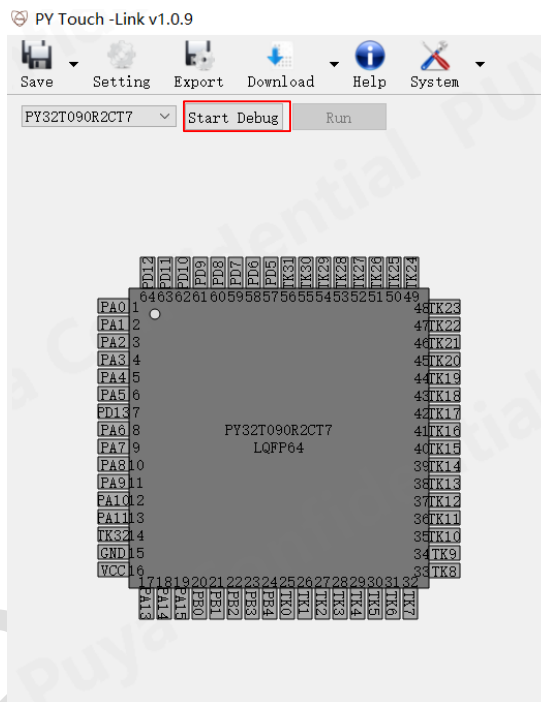


Figure 3-9 Program download method 2 - Host computer download

Step 3: Connect the host debugging tool PY Touch - Link and enter touch debugging mode.



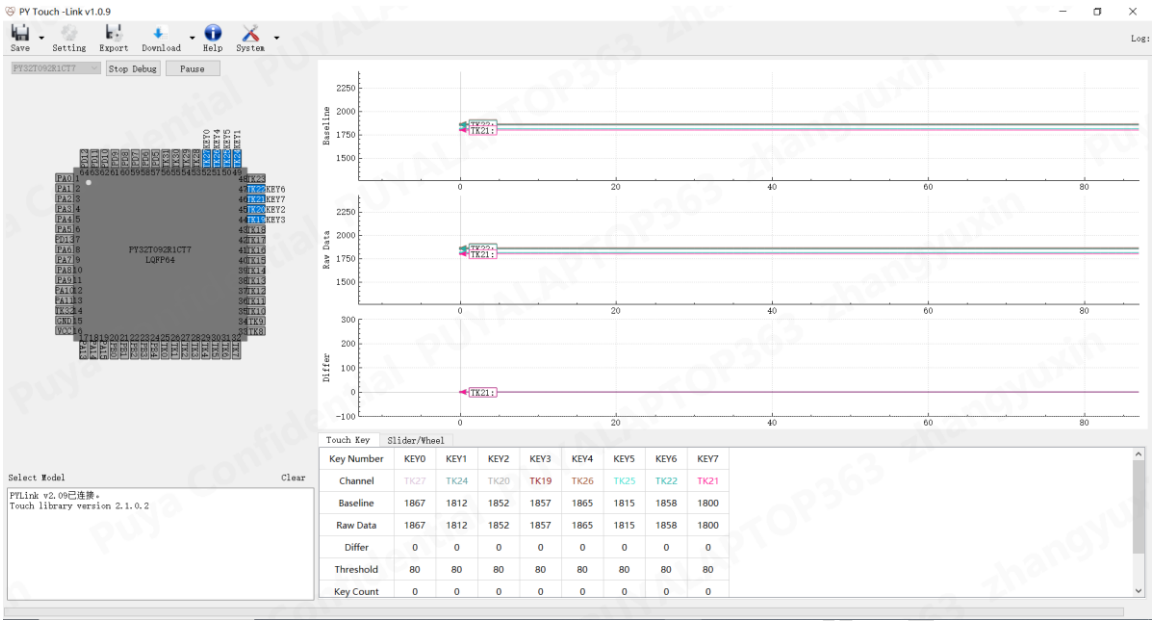


Figure 3-10 Touch host computer debugging example

Step 4: Touch the key normally with a finger (or copper post), observe the change in touch data, and set 60%–70% of the change amount as the trigger threshold.

Touch Key	Slider/Wheel							
Channel	TK27	TK24	TK20	TK19	TK26	TK25	TK22	TK21
Baseline	1879	1812	1852	1857	1865	1815	1858	1800
Raw Data	1993	1812	1852	1857	1865	1815	1858	1800
Differ	114	0	0	0	0	0	0	0
Threshold	80	80	80	80	80	80	80	80
Key Count	10	0	0	0	0	0	0	0
Status								

Figure 3-11 Touch key delta and threshold calculation example

Example: As shown above, when a finger normally touches KEY0, the change amount is about 114, and the threshold can be set to: $114 * 70\% = 80$. The setting method is as follows:

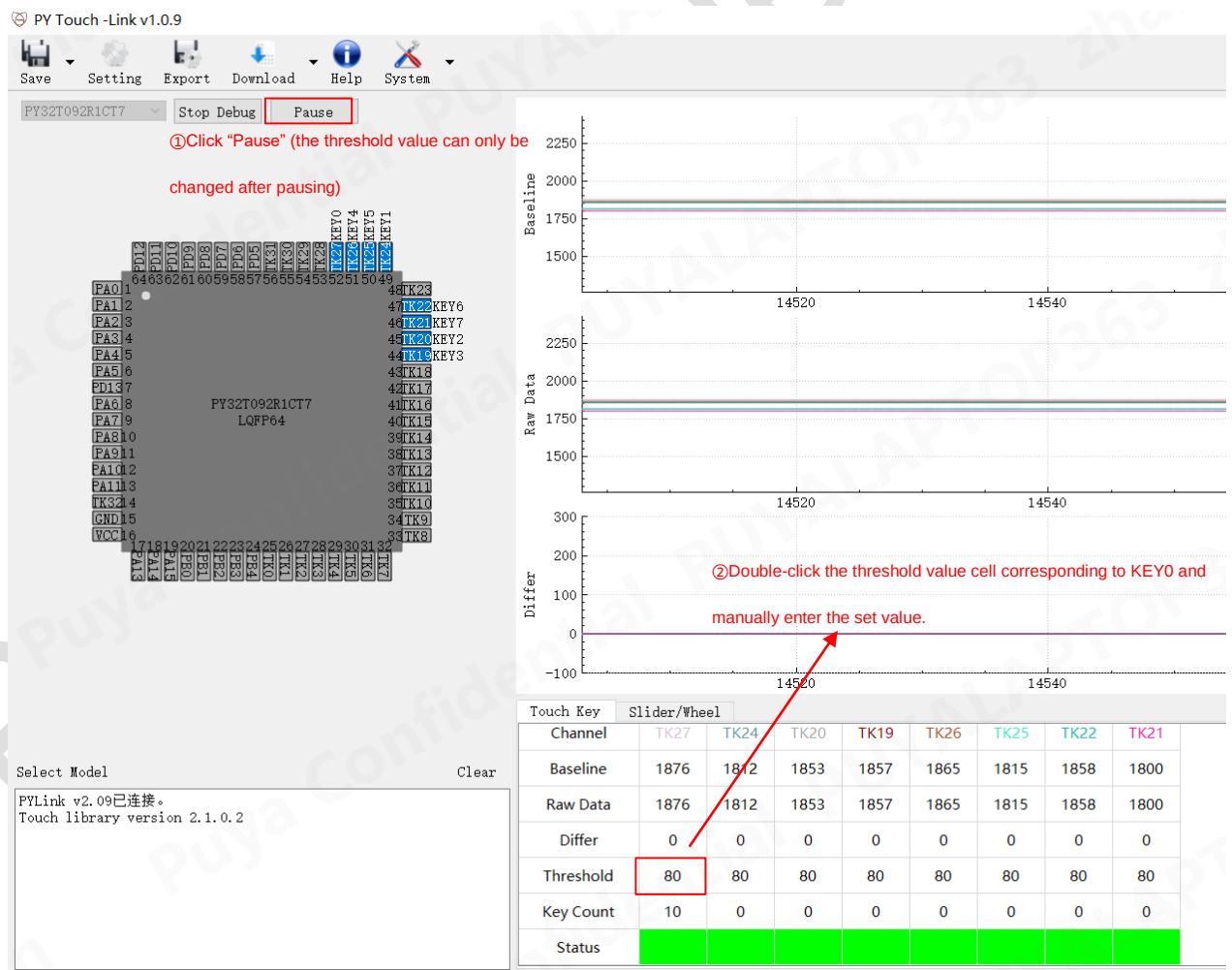


Figure 3-12 Host computer threshold setting interface example

Step5: After setting all keys according to Step4, export tk_cfg_user.h and overwrite the original file.

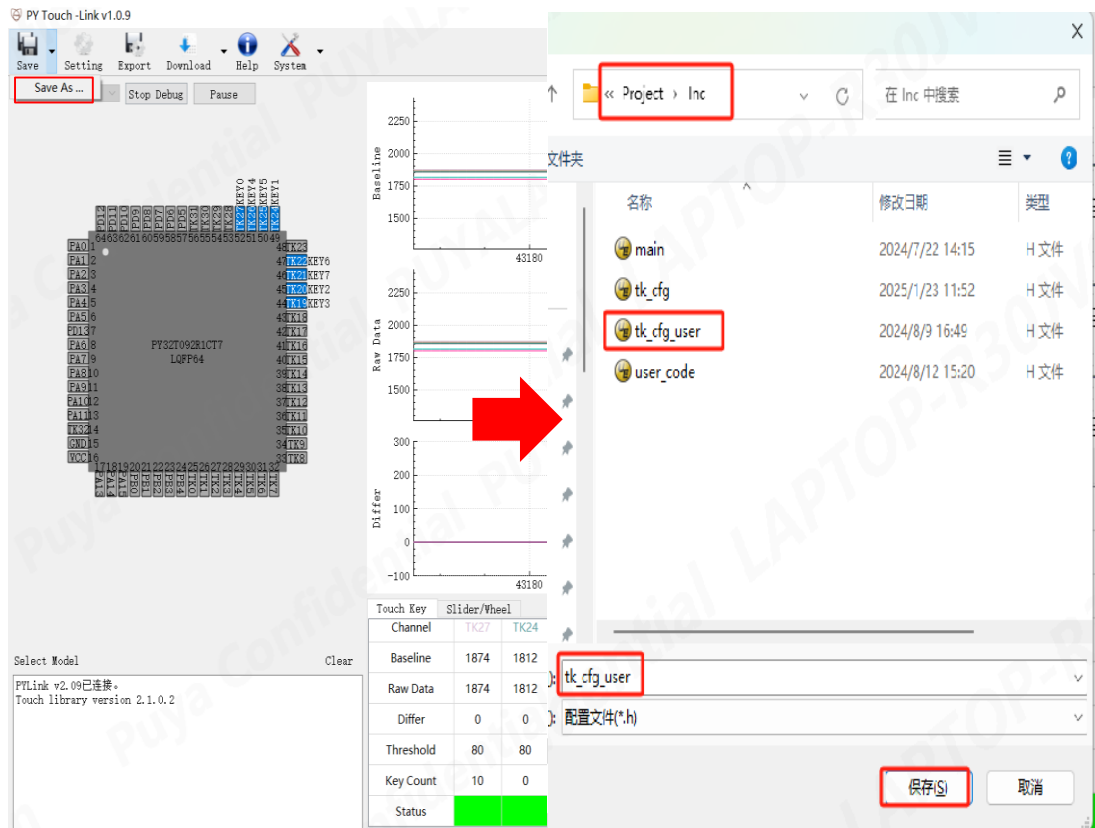


Figure 3-13 Touch configuration file export interface example

Step6 (Optional): Modify the configuration parameters in tk_cfg.h. As shown in the figure, the parameters in the red box can be adjusted appropriately according to actual conditions, while the remaining parameters can generally keep the default settings.

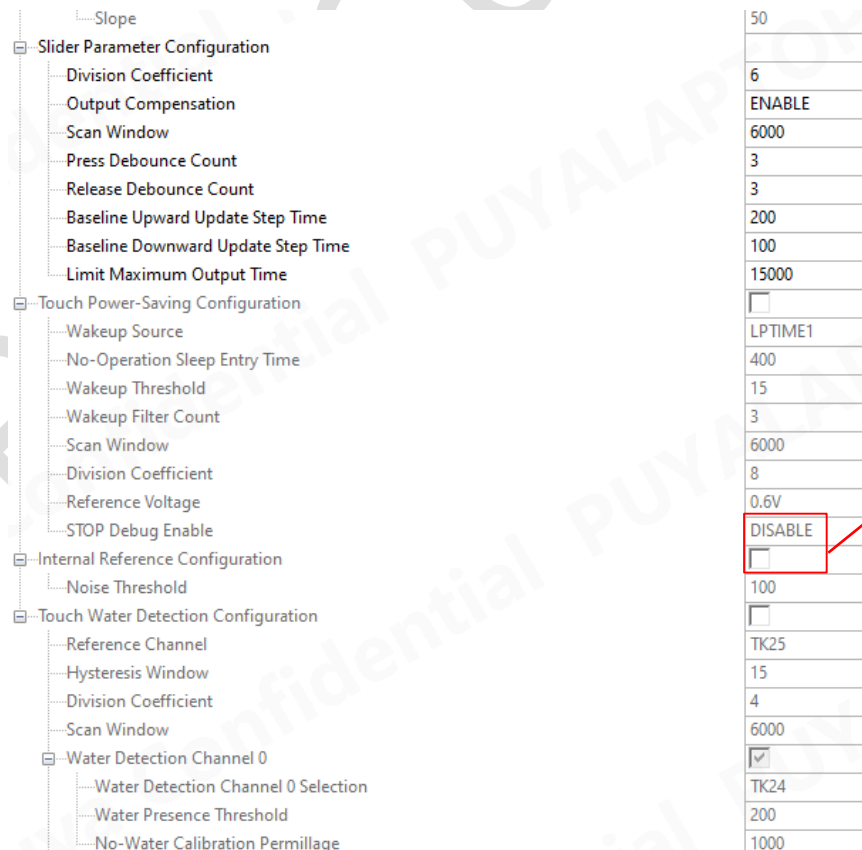
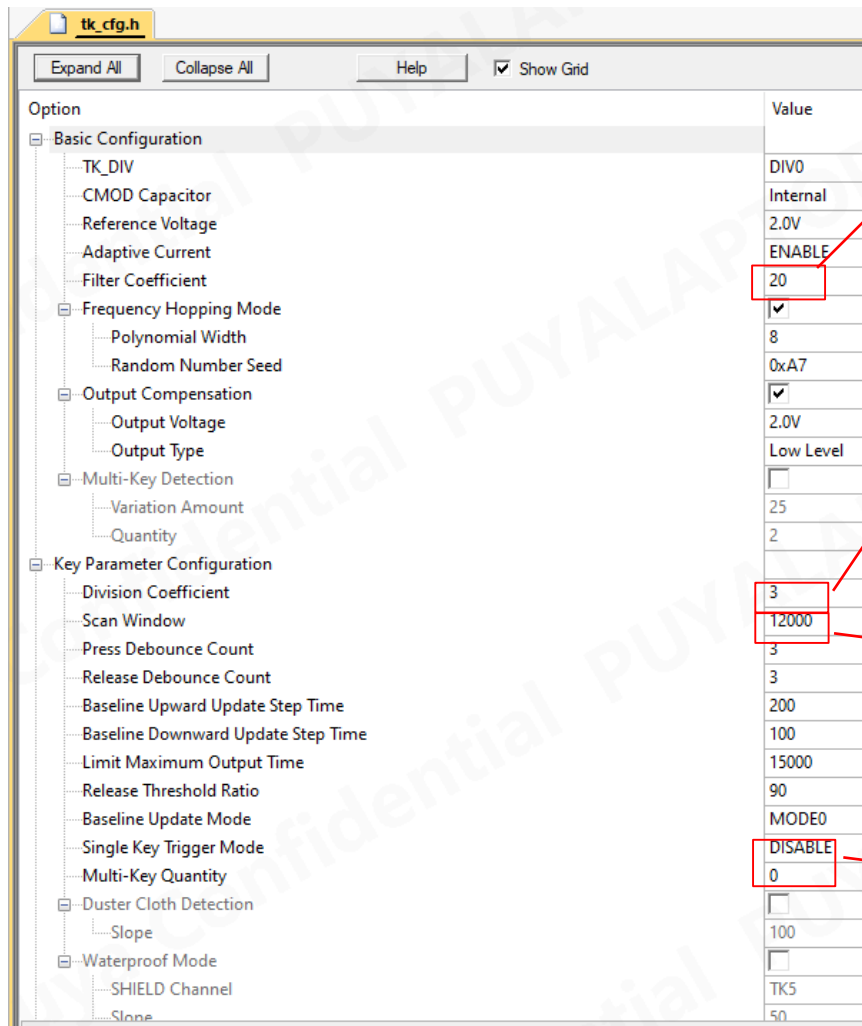


Figure 3-14 Touch parameter configuration wizard interface example

Step7: At this point, the configuration of conventional keys is complete. When a key event occurs, the corresponding bit of TKCtr.KeyFlags will be set. The user program can check the corresponding key flag bit and perform the appropriate operation.

Puya Confidential

3.6.2. Wheel/Slider Application

Step1: Configure the wheel/slider channels, type, and resolution. There are two methods to achieve this.

Method 1: Select channels, set the wheel/slider type and resolution in PY Touch - Link. After completing the settings, export tk_cfg_user.h and overwrite the original file (refer to the key application section for the export method), as shown in the figure.

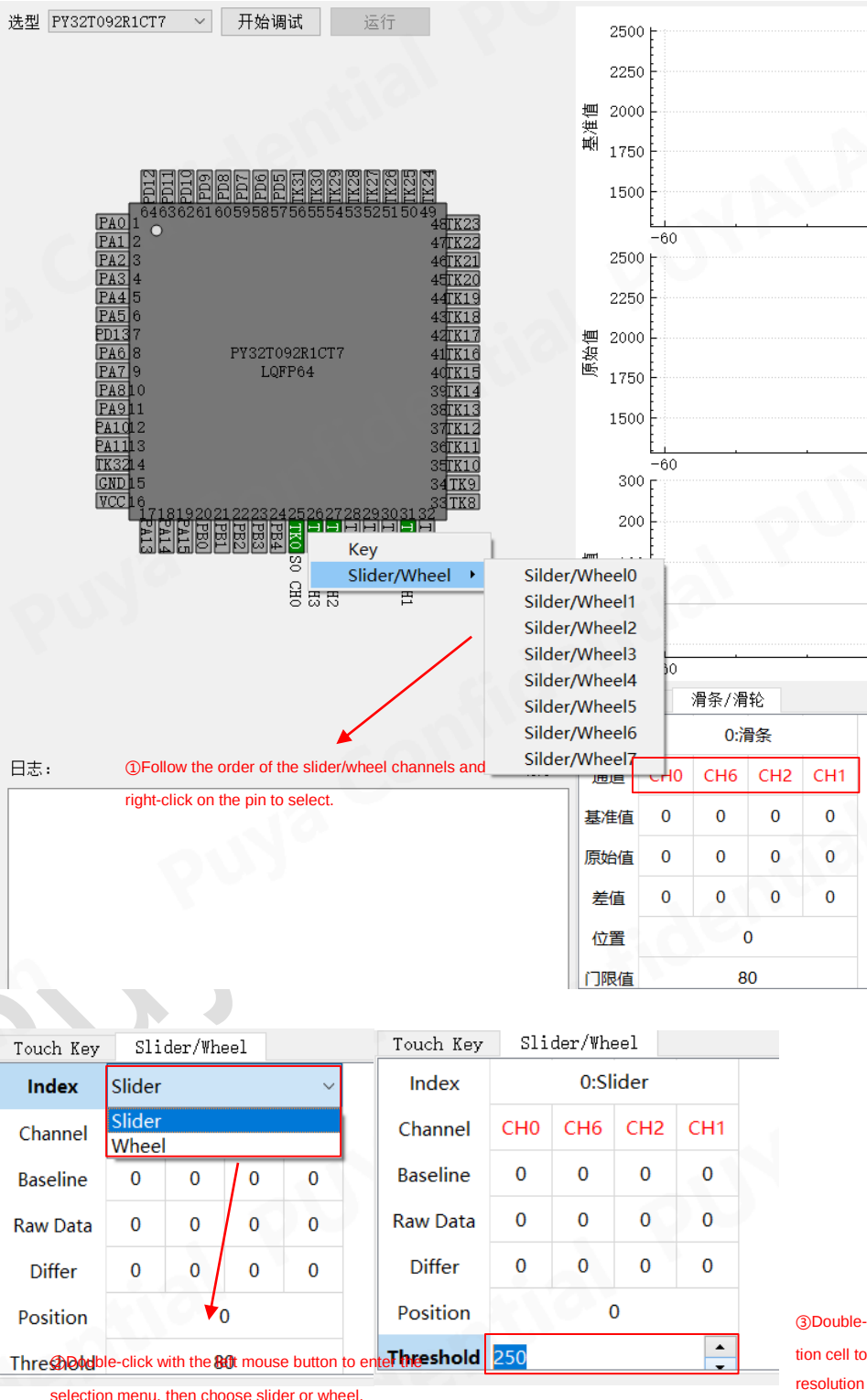


Figure 3-15 Slider/wheel configuration interface example

Method 2: Directly modify it in tk_cfg_user.h, as shown in the figure below.

```
//-----
#define SLIDER_OR_WHEEL0_TYPE      TK_APP_WHEEL    // Slider type
#define SLIDER_OR_WHEEL0_RESOLUTION 360           // Slider resolution
#define SLIDER_OR_WHEEL0_THD      100             // Slider threshold
#define SLIDER_OR_WHEEL0_CH0      TK_CH0          // Slider channels, fill in sequentially
#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
```

Figure 3-16 Header file slider/wheel configuration code example

Step 2: After completing the basic configuration of the slider/wheel, compile the program and download it to the target board, connect PY Touch - Link for debugging, and set an appropriate trigger threshold.

Touch Key	Slider/Wheel							
Index	0:Wheel				1:Slider			
Channel	CH0	CH6	CH2	CH1	CH19	CH23	CH24	CH25
Baseline	3609	3598	3593	3528	3601	3595	3583	3581
Raw Data	3709	3604	3600	3533	3605	3595	3584	3581
Differ	100	6	7	5	4	0	1	0
Position	359				-1			
Threshold	80				100			

Touch Key	Slider/Wheel							
Channel	CH0	CH6	CH2	CH1	CH19	CH23	CH24	CH25
Baseline	3607	3599	3591	3526	3601	3593	3580	3582
Raw Data	3607	3599	3591	3526	3601	3593	3580	3582
Differ	0	0	0	0	0	0	0	0
Position	-1				-1			
Threshold	80				100			
Resolution	360				255			

①Slide your finger normally on the slider/wheel and observe the variation of each channel.

②Double-click to enter the threshold setting interface, and repeatedly modify the threshold value until the best operating experience is achieved.

```
//-----
#define SLIDER_OR_WHEEL0_TYPE      TK_APP_WHEEL    // Slider type
#define SLIDER_OR_WHEEL0_RESOLUTION 360           // Slider resolution
#define SLIDER_OR_WHEEL0_THD      100             // Slider threshold
#define SLIDER_OR_WHEEL0_CH0      TK_CH0          // Slider channels, fill in sequentially
#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
```

③Fill the final determined threshold value into tk_cfg_user.h.

Figure 3-17 Slider/wheel configuration interface example

Step 3: At this point, the slider/wheel configuration is complete. The generated position information is stored in TKCtr.SliderOrWheelPosition[n]. The user program can read this value and perform corresponding operations.

```
if (TKCtr.SliderOrWheelPosition[0] != -1)
{
    // User Application Program
}
if (TKCtr.SliderOrWheelPosition[1] != -1)
{
    // User Application Program
}
```

Figure 3-18 Slider/wheel position information reading code example

3.6.3. Touch Library Low-Power Application

Step 1: Enable the touch low-power mode in tk_cfg.h and configure the basic parameters.

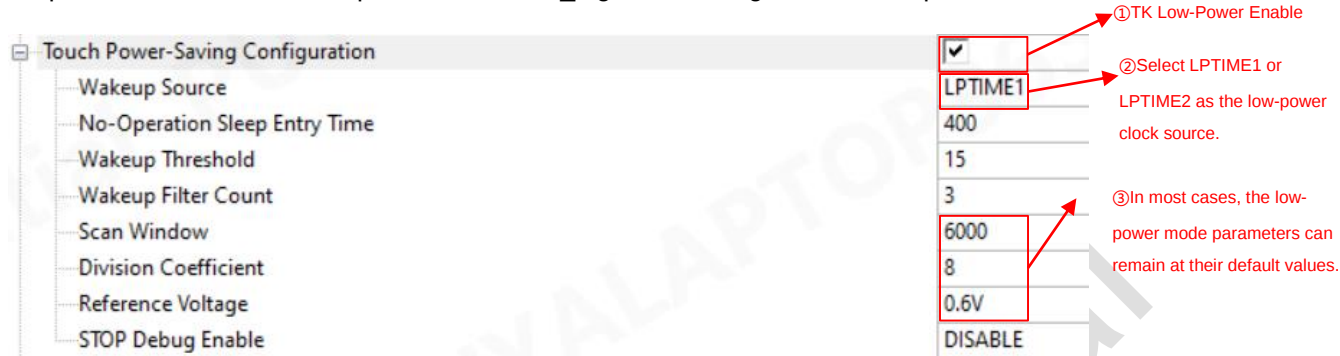


Figure 3-19 Low power configuration interface example

Since touch low power uses LPTIME1 or LPTIME2 as the time base, LPTIME must also be enabled and configured first. LPTIME is configured in app_config.h.

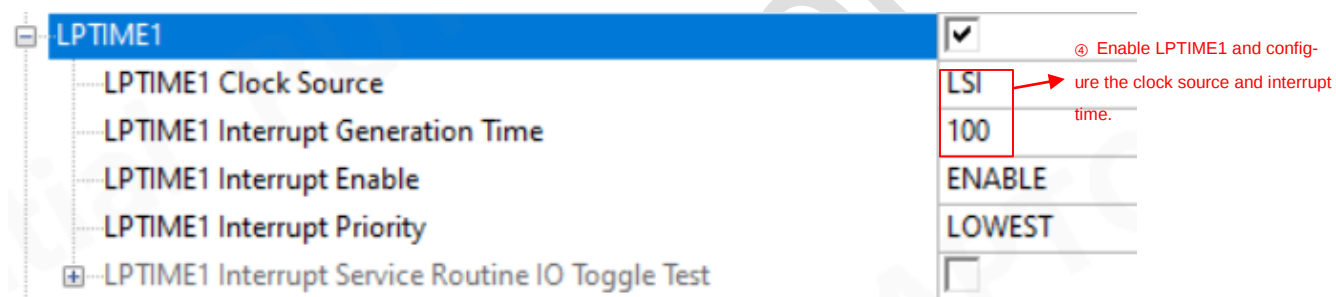


Figure 3-20 LPTIME time base configuration interface example

Step 2: Set the trigger threshold for low-power mode.



Figure 3-21 Low power threshold setting interface example

Step 3: Download the TK low-power mode program to the target board. Touch the button or slider normally with your finger and observe the data variation to determine the threshold.

Touch Key	Slider/Wheel		
Key Number	KEY0	KEY1	
Channel	TK14	TK15	TK_CO...
Baseline	3601	3607	5465
Raw Data	3602	3608	5513
Differ	1	1	48
Threshold	180	180	15
Key Count	2	1	0

Data variation when a finger touches in low-power debug mode. The threshold is set based on this variation value, typically about 50% of the variation.

Note: The variation of each key is not exactly the same. In practice, the key with the smallest variation should be used as the reference.

Figure 3-22 Low power debugging data interface example

Step 4: Set the threshold according to the debug data, disable debug mode, and re-download the program for testing. Fine-tune the threshold based on the actual test results until the best experience is achieved.

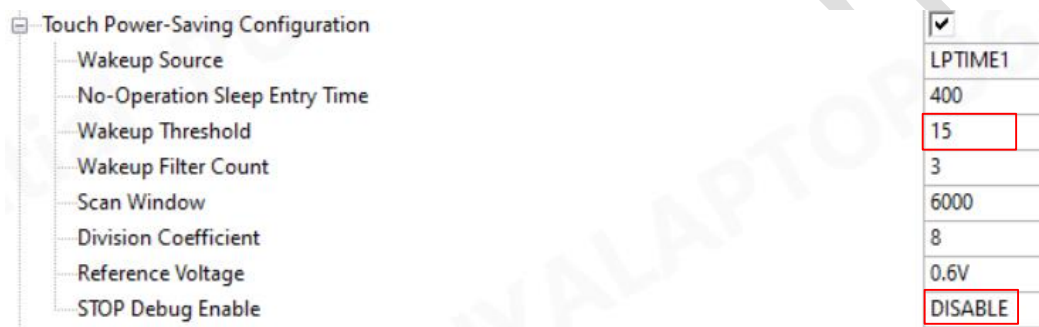


Figure 3-23 Low power debugging disable configuration example

3.6.4. Floating Button Application

The setup method for the floating button application is similar to that of regular buttons, as shown below:

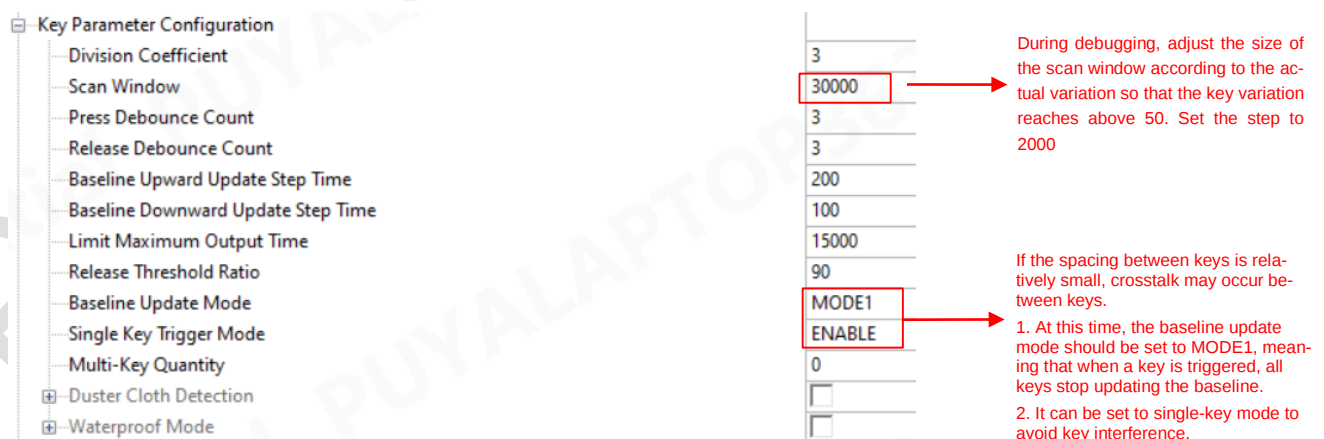


Figure 3-24 Floating button configuration interface

3.6.5. Waterproof Button Application (with Shield Channel)

The setup method for waterproof button applications is similar to that of regular buttons. The difference is that the output compensation must be set to in-phase mode, as shown below.

Step 1: Increase the division coefficient, modify the compensation output type, and set the Shield channel.

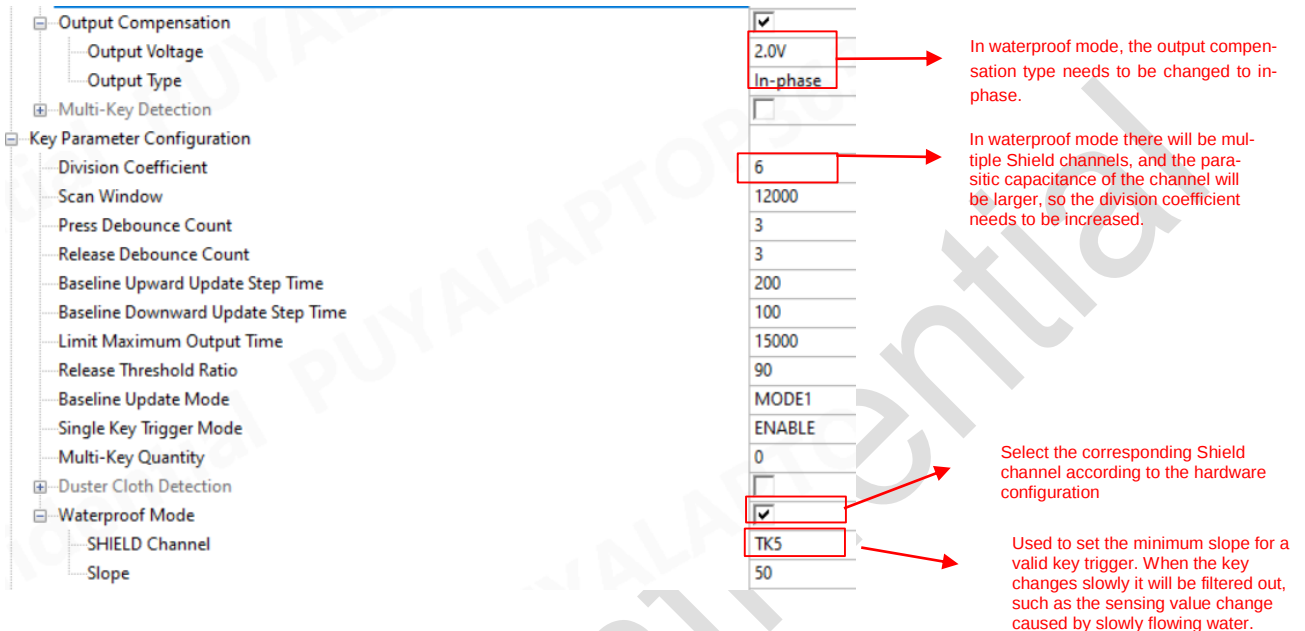


Figure 3-25 Waterproof key configuration interface

Step 2: Enable PRINTF in app_config.h.



Figure 3-26 PRINTF function enable configuration example

Step 3: After assembling the enclosure, download the program to the target board and connect PY Touch - Link. You can see that an additional channel TK_COM appears in the touch key column, which represents the Shield channel data.

触摸按键	滑条/滑轮									
按键序号	KEY0	KEY1	KEY2	KEY3	KEY4	KEY5	KEY6	KEY7	KEY8	KEY9
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										

Figure 3-27 Waterproof key debugging interface and Shield channel data

Step 4: Press the touch key when there is no water present. The log panel in the lower-left corner of the host software will print the following information.

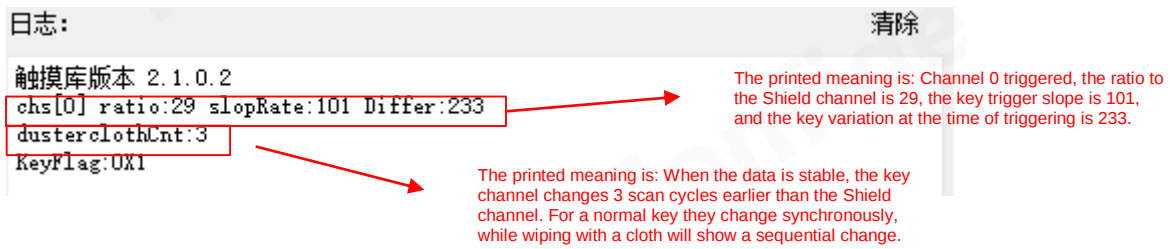


Figure 3-28 Touch key log print example in dry state

Step 5: Press the touch key when water is present. The log panel in the lower-left corner of the host software will print the following information.



Figure 3-29 Touch key log print example in wet state

Step 6: Modify the program according to the ratio obtained from actual measurements

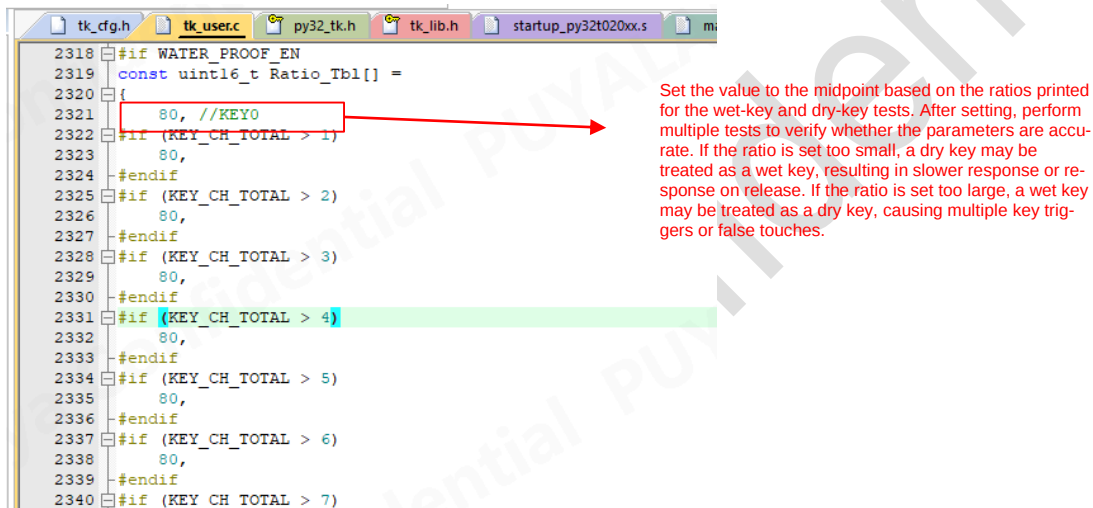


Figure 3-30 Waterproof key ratio parameter configuration code example

3.6.6. Waterproof key application (without Shield channel)

When the hardware does not have a Shield channel, basic waterproof logic processing can be implemented through software.

Step 1: Set other parameters according to the normal key mode.

Step 2: Enable wipe detection mode and multi-key detection.

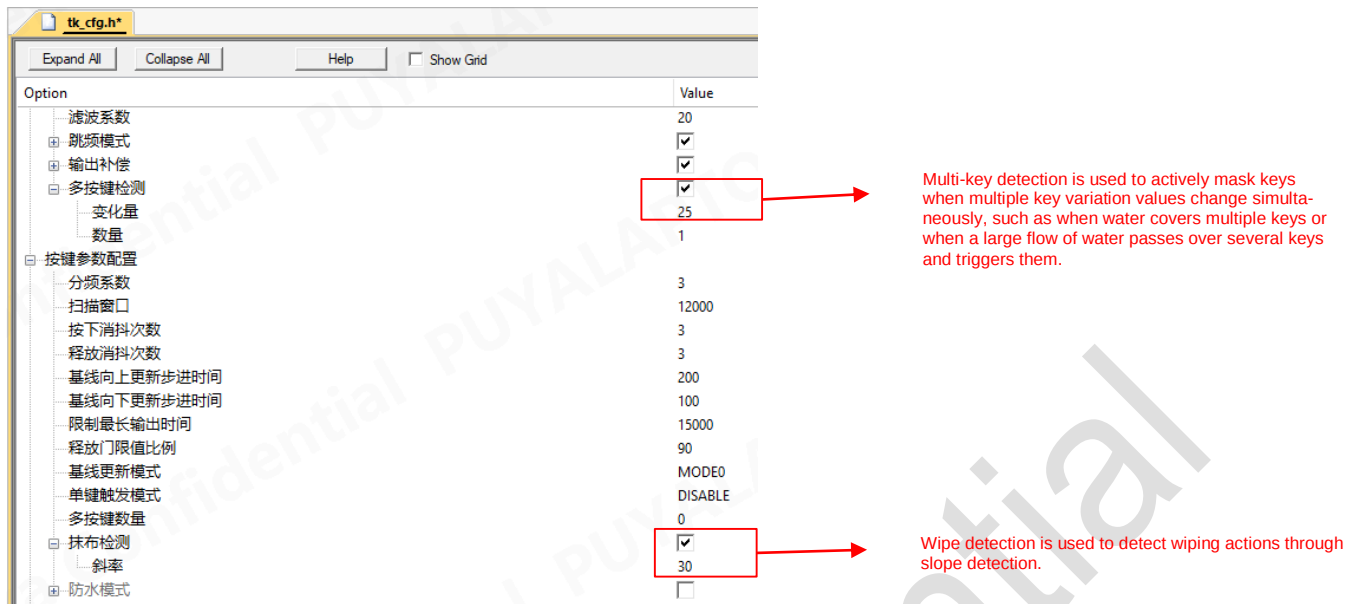


Figure 3-31 Waterproof key configuration interface without Shield channel

Step 3: Enable PRINTF in app_config.h.



Figure 3-32 Print function enable configuration example

Step 4: After assembling the enclosure, download the program to the target board, connect to PY Touch - Link, and perform key operations.

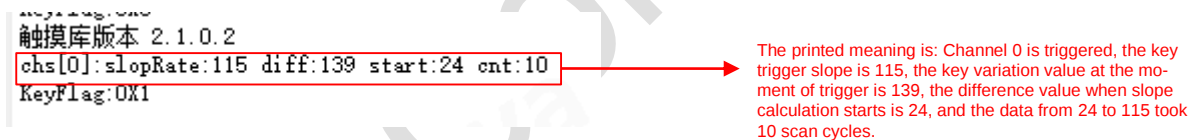


Figure 3-33 Waterproof key debugging log print example without Shield

Step 5: Use a cloth to perform simulation testing and check the printed ratio.

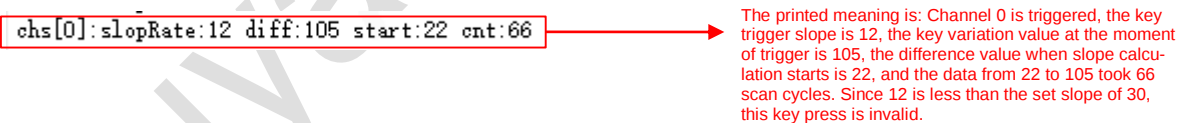


Figure 3-34 Cloth test log print example

3.6.7. Touch water detection application

Step 1: Enable the water detection function and configure the reference channel, water detection channel, and basic parameters.

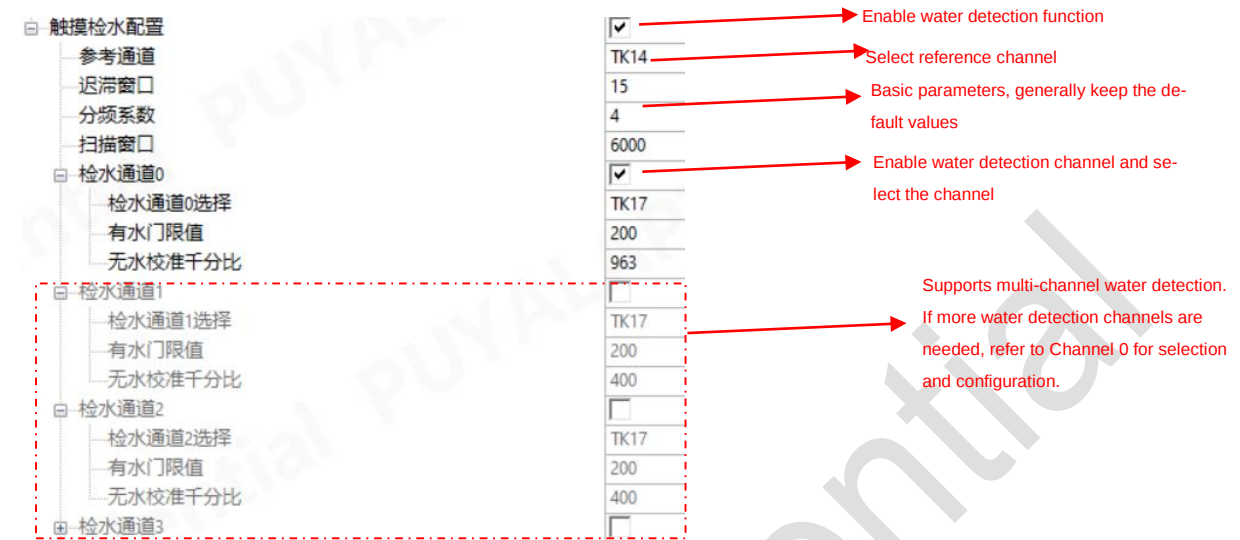


Figure 3-35 Touch water detection function configuration interface

Step 2: Enable PRINTF in app_config.h;



Figure 3-36 Water detection print enable configuration

Step 3: Assemble the enclosure and container. Keep the container without water at this time. Download the program to the target board and connect PY Touch - Link.

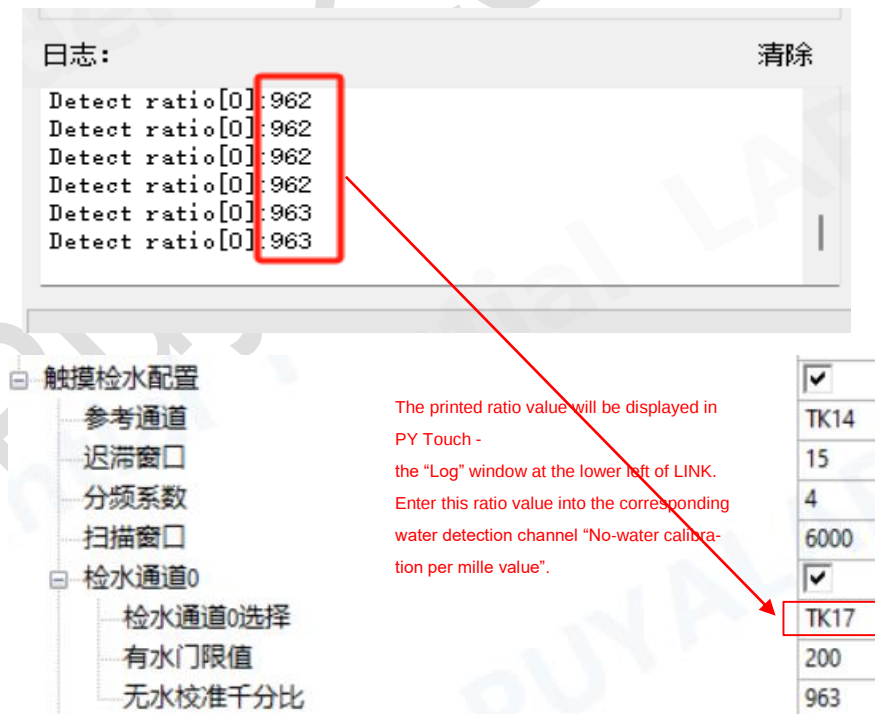


Figure 3-37 Water detection ratio and parameter setting in dry state

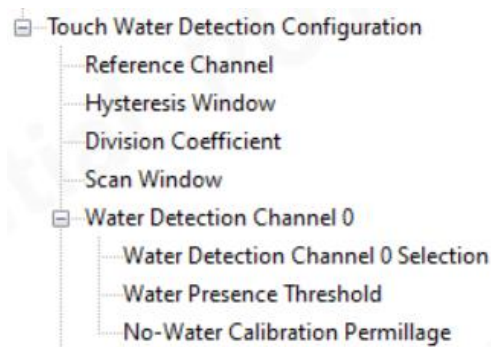
Step 4: After filling in the calibration value for the no-water state, recompile and download the program.

Touch Key	Slider/Wheel	
Channel	TK14	TK17
Baseline	5658	5658
Raw Data	5658	6012
Differ	0	354
Threshold	0	200
Key Count	0	7
Status		

Pour water into the container until the water level exceeds the position of the water detection PAD. At this time, the difference value of the detection channel is the change amount when water is present. Generally, about 50% of the water-present change amount is taken as the water detection threshold.

Figure 3-38 Water detection threshold calculation

Step 5: Take about 50% of the water-present change amount and fill it into the “Water-present threshold”.



☒	TK14
	15
	4
	6000
☒	TK17
	177
	963

Fill in the water-present threshold: $354 \times 0.5 = 177$

Figure 3-39 Water detection threshold setting

Step 6: Download the program again, repeatedly test and fine-tune the threshold until the best water detection effect is achieved. At this point, the water detection function debugging is complete. The user program can perform corresponding operations according to the water detection status.

```

for (uint8_t n = 0; n < (WATER_DETECT_CNT - 1); n++)
{
    /* Water detected */
    if (TKCtr.DetectOut & (1 << n))
    {
        // User Application Program
    }
    /* No water */
    else
    {
        // User Application Program
    }
}

```

Figure 3-40 Water detection status reading and user program code example

4. System configuration and peripheral application analysis

4.1. System Configuration

4.1.1. Clock configuration

Configure in app_config.h as shown in the figure:

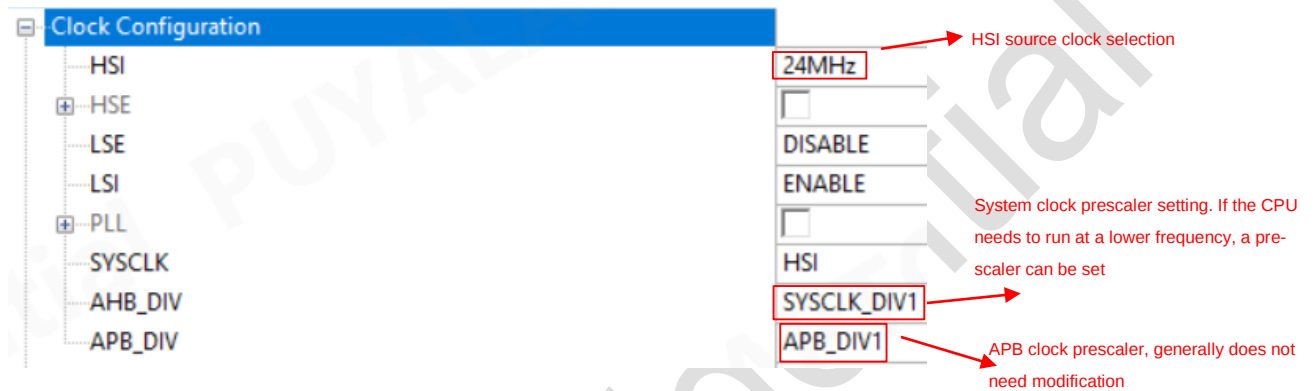


Figure 4-1 System clock configuration interface

4.2. Interfaces and usage of peripheral applications

4.2.1. UART

4.2.1.1. UART settings interface

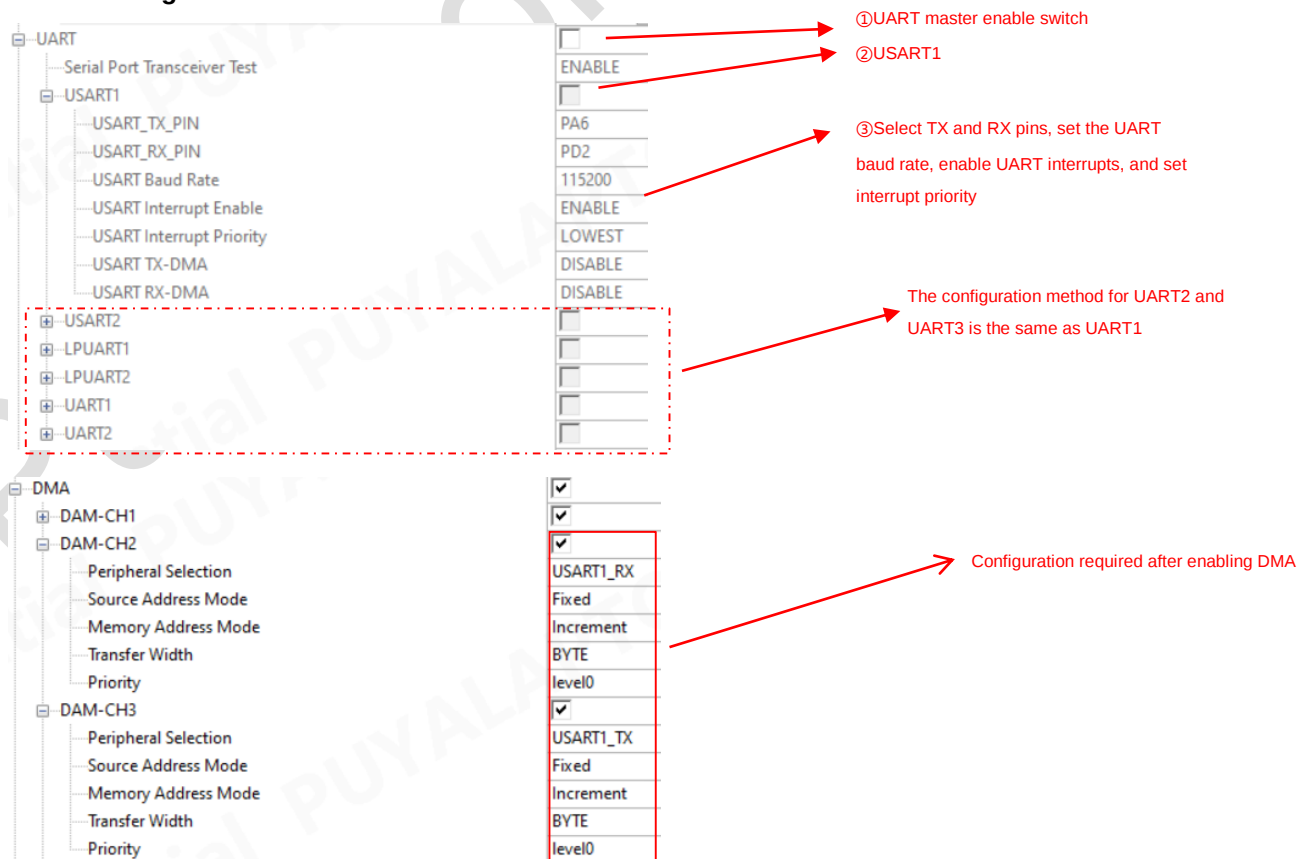


Figure 4-2 UART configuration interface

4.2.1.2. UART Function Interface Description

Table 4-1 UART function interface description

Function Name	Input Parameters			Return Value	Note
	Parameter Name	Data Type	Parameter Description		
USARTx_QueueRead	data	uint8_t *	Receive pointer; when data is available, the received data will be written to this memory	Receive Status	Calling this function reads one byte of data from USARTx; a return value of 1 indicates successful reception, and 0 indicates no data;
USARTx_QueueSend	data	uint8_t *	Data to be sent	Send Status	Calling this function sends one byte of data through USARTx; a return value of 1 indicates the send was successful, and 0 indicates the send failed (queue full);
UARTx_QueueRead	data	uint8_t *	Receive pointer; when data is available, the received data will be written to this memory	Receive Status	Calling this function reads one byte of data from USARTx; a return value of 1 indicates successful reception, and 0 indicates no data;
UARTx_QueueSend	data	uint8_t *	Data to be sent	Send status	Calling this function sends one byte of data through USARTx. A return value of 1 indicates the send was successful, while 0 indicates the send failed because the queue is full.

Function Name	Input Parameters			Return Value	Note
	Parameter Name	Data Type	Parameter Description		
LPUSARTx_QueueRead	data	uint8_t *	Receive pointer; when data is available, the received data will be written to this memory	Receive status	Calling this function reads one byte of data from USARTx. A return value of 1 indicates the receive was successful, while 0 indicates that no data is available.
LPUARTx_QueueSend	data	uint8_t *	Data to be sent	Send status	Calling this function sends one byte of data through USARTx. A return value of 1 indicates the send was successful, while 0 indicates the send failed because the queue is full.

Note: The above UART function interfaces are located in uart_drivers.c.

4.2.1.3. UART Application Example

UART
Serial Port Transceiver Test

☒ **ENABLE**

UART test enable. After enabling, the chip can connect to a PC and communicate with a serial port assistant.

```

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
}

```

Use a serial port assistant to send data; the chip receives 64 bytes of data and stores them in the rx1_data array

Send the received 64 bytes back. If the data received by the serial port assistant matches the transmitted data, it proves that UART transmission and reception are functioning correctly.

Figure 4-3 UART application example code

4.2.2. Timers

4.2.2.1. SysTick

The SYSTICK configuration interface is shown below:

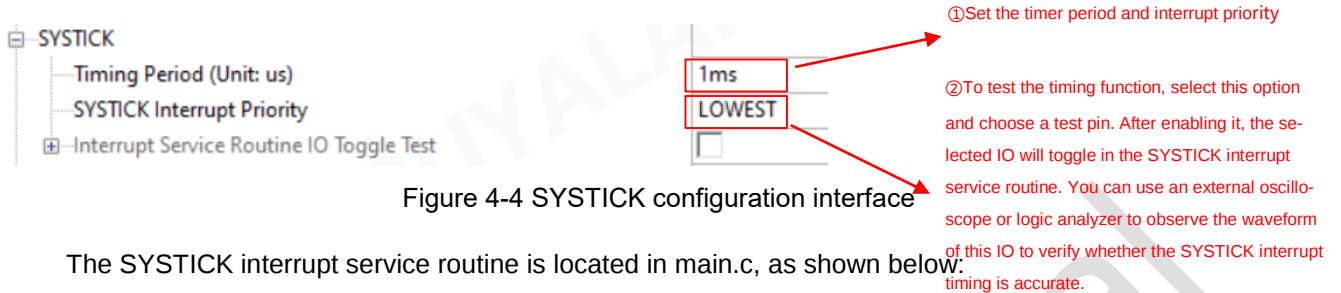


Figure 4-4 SYSTICK configuration interface

The SYSTICK interrupt service routine is located in main.c, as shown below:

```

/*****
** Function Name : void SysTick_Handler(void)
** Description : System tick timer, current timing interval is 1ms
** Input : None
** Return : None
*****/
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();
}
    
```

If the user program needs code to be executed in the SYSTICK periodic interrupt service, the code can be placed in user_timer

Figure 4-5 SYSTICK interrupt service routine code example

4.2.2.2. LPTIME1/LPTIME2

The LPTIMEx (x=1, 2) timer configuration interface is as follows:

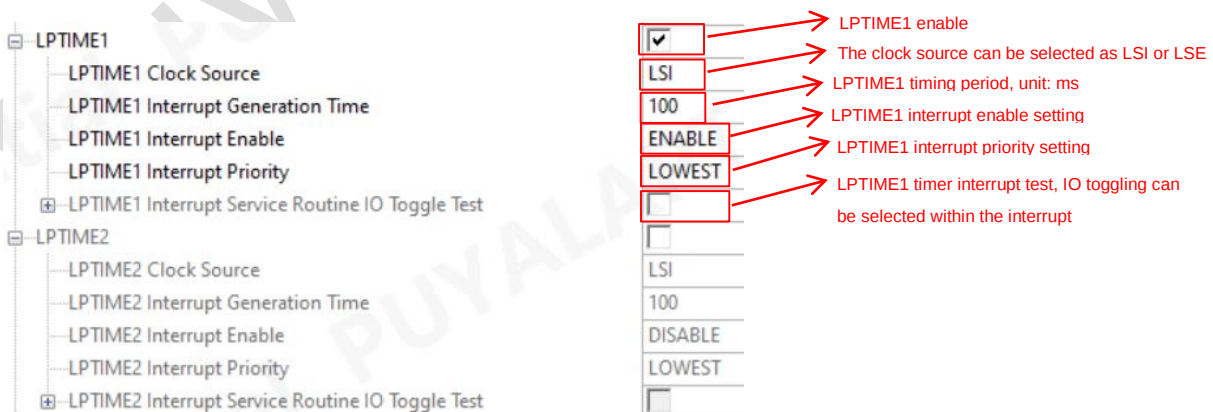


Figure 4-6 LPTIME configuration interface example

The LPTIME interrupt callback function is as follows (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 callback */
            #if (LPTIME1_DEBUG && (LPTIME1_DEBUG_GPIO != NO_PIN))
                GPIO_ToggleBit(LPTIME1_DEBUG_GPIO);
            #endif
        }
    #endif
    #if LPTIME2_ENABLE
        if(hlptim == &LPTIME2_Handle)
        {
            /* LPTIME2 callback */
            #if (LPTIME2_DEBUG && (LPTIME2_DEBUG_GPIO != NO_PIN))
                GPIO_ToggleBit(LPTIME2_DEBUG_GPIO);
            #endif
        }
    #endif
}

```

Figure 4-7 LPTIME interrupt callback function code example

4.2.2.3. TIMx (x=1, 2, 3, 15, 16, 17)

Function mutual exclusion: The PWM, timer, and input capture functions of TIMx cannot be used simultaneously.

The TIMx (x=1, 2, 3, 15, 16, 17) timer configuration interfaces are the same. TIM1 is used as an example below:

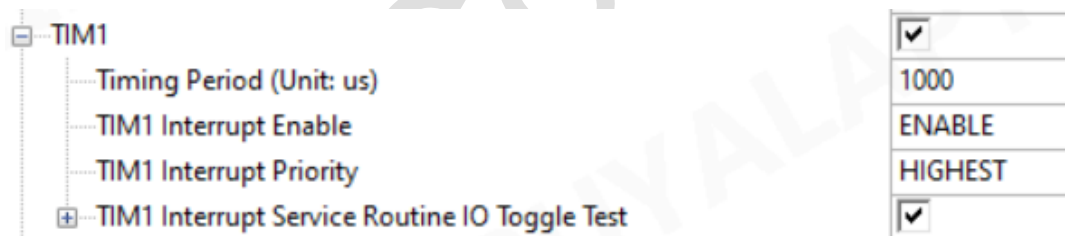


Figure 4-8 TIM1 configuration interface

Note: The TIMx timer configuration is basically the same as SYSTICK. Please refer to the related description of SYSTICK.

The TIMx timer interrupt service routines are located in timx_drivers.c, as shown below:

```

/**
 * @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);
        /* Test GPIO toggle */
        #if (TIM1_DEBUG_GPIO != NO_PIN && TIM1_DEBUG)
            GPIO_ToggleBit(TIM1_DEBUG_GPIO);
        #endif
    }
}

```

Figure 4-9 TIM1 interrupt service routine

4.2.2.4. TIMx-PWM (x=1, 2, 3, 15, 16, 17)

Function mutual exclusion: The PWM, timer, and input capture functions of TIMx cannot be used simultaneously.

The TIMx-PWM configuration interface is shown below:

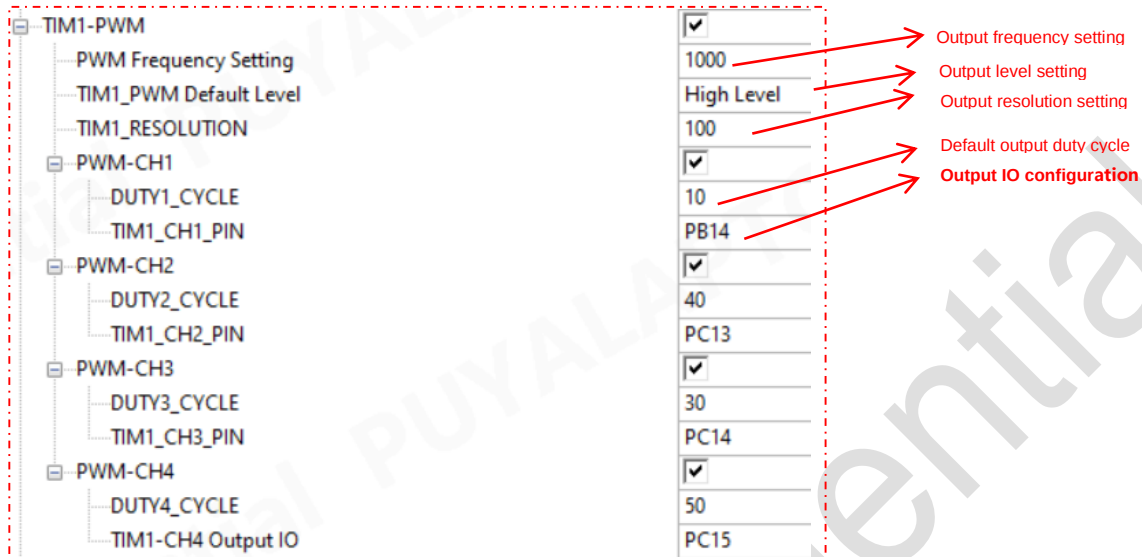


Figure 4-10 TIM1-PWM configuration interface example

If it is necessary to adjust the TIMx-PWM duty cycle in an application, the TIMx_PWM_Pulse interface function can be called for configuration. For example, the figure below shows the TIM1 duty cycle adjustment function.

```

/*****
**  Function: void TIM1_PWM_Pulse(uint32_t CHx, uint16_t percent)
**  Description: Timer 1 - PWM duty cycle setting
**  Input: CHx, channel number, TIM_CHANNEL_1~TIM_CHANNEL_4
**          percent, output percentage
**  Return: None
*****/
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;
    }
}

```

Figure 4-11 TIM1-PWM duty cycle adjustment function code example

4.2.2.5. TIMx-CAPTURE (x=1, 2, 3, 15)

Mutual exclusion: The PWM, timer, and input capture functions of TIMx cannot be used simultaneously.

The configuration interface for TIMx-CAPTURE (x=1, 2, 3, 15) is the same. TIM1 is used as an example below:



Figure 4-12 TIM1 input capture configuration interface example

Usage:

(1) Configure TIMx as input capture mode (for example, select TIM1-CAPTURE, with the signal input pin as PC13).

(2) Input the signal to be measured.

(3) Directly read the variables to obtain the result:

Frequency: Read TIMxCapture.Freq (for example: TIM1Capture.Freq)

Duty cycle: Read TIMxCapture.Duty (for example: TIM1Capture.Duty)

In the following function, the duty cycle and frequency of the input capture can be obtained.

```

void app_drivers_loop(void)
{
    /* External interrupt flag */
    if (EXTI_Flag != 0)
    {
        #if APP_IWDG_ENABLE
        #if APP_ADC_ENABLE
        #if (APP_UART_ENABLE && APP_UART_TEST)
        #if (APP_I2C1_ENABLE || APP_I2C2_ENABLE)
        #if (APP_SPI1_ENABLE || APP_SPI2_ENABLE)
        #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
        #if APP_TIM15_INPUTCAPTURE_ENABLE
            if (TIM15Captue.Time)

```

Figure 4-13 Input capture frequency and duty cycle reading code example

4.2.3. PWM

The PWM configuration interface is shown below:

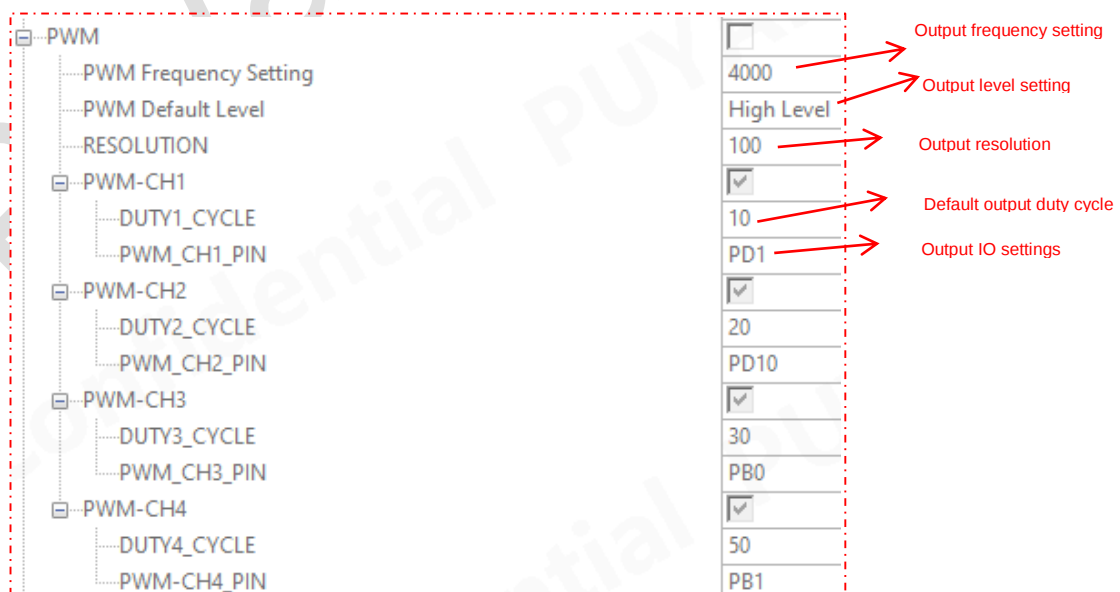


Figure 4-14 PWM configuration interface

If it is necessary to adjust the PWM duty cycle in the application, you can call the PWM_Pulse_Config interface function to configure it.

4.2.4. Watchdogs

The watchdog configuration interface is shown below:



By default, the program feeds the watchdog in the main loop.

```

/*****
** Function: void app_drivers_loop(void)
** Description: Polled in the main function's while(1) loop, used to handle tasks of each module
** Input: None
** Return: None
*****/
void app_drivers_loop(void)
{
    /* External interrupt flag */
    if (EXTI_Flag != 0)
    {
        log_printf("EXTI_Flag:0X%X\r\n", EXTI_Flag);
        EXTI_Flag = 0;
    }
}

#ifdef APP_IWDG_ENABLE
/* Feed the watchdog at half the configured watchdog time */
if (Iwdg_Time > (IDWG_RELOAD_VALUE >> 1))
{
    Iwdg_Time = 0;
    /* Refresh IWDG */
    if (HAL_IWDG_Refresh(&hiwdg) != HAL_OK)
    {
        APP_ErrorHandler();
    }
}
#endif

```

Figure 4-15 Watchdog configuration interface and feed dog code example

4.2.5. PVD (Low Voltage Detection)

The configuration interface is as follows:



The voltage detection service function is shown below.

```
void HAL_PWR_PVDCallback(void)
{
    if ( __HAL_PWR_GET_FLAG(PWR_FLAG_PVDO) )
    {
        /* Detected power supply voltage below detection voltage */
        log_printf("LVD DOWN\n");
    }
    else
    {
        log_printf("LVD UP\n");
    }
}
```

Figure 4-16 PVD configuration and callback function

4.2.6. ADC

The ADC configuration interface is as follows:

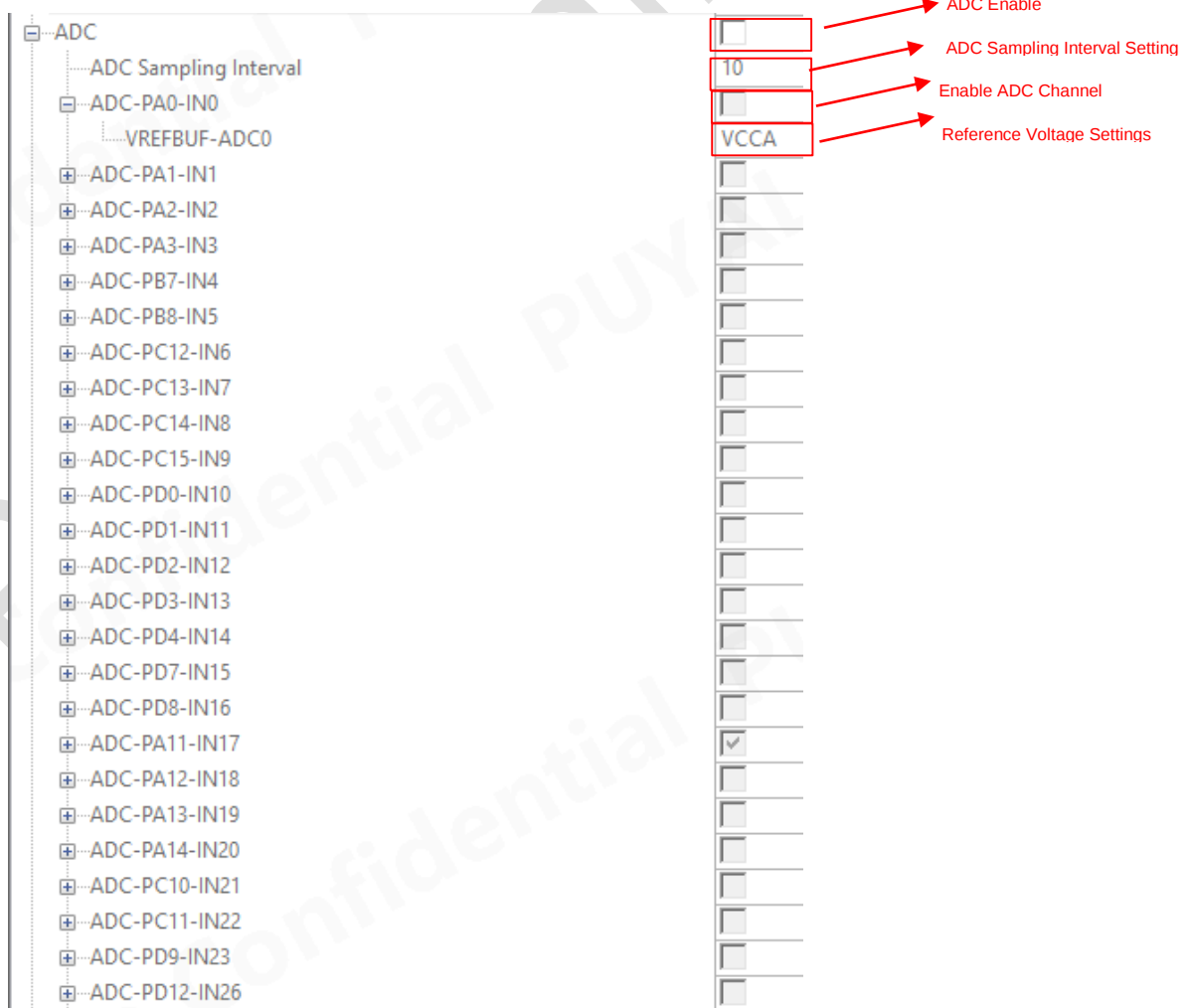


Figure 4-17 ADC channel configuration interface

After sampling is completed, the ADC_Callback function will be called.

```
#if APP_ADC_ENABLE
/*****
** Function Name    void ADC_Callback(void)
** Description : ADC conversion complete callback, process ADC data within this function
** Input   : None
** Return  : None
*****/
void ADC_Callback(void)
{
    #if 0
        log_printf("Vcc_Power = %d %d\r\n", Vcc_Power, ADCxConvertedData[ADC_CHANNEL_VREFINT_INDEX]);
    #endif
}
```

Figure 4-18 ADC conversion complete callback function example

4.2.7. DMA

DMA configuration interface. It cannot be used independently and must be used together with peripherals.

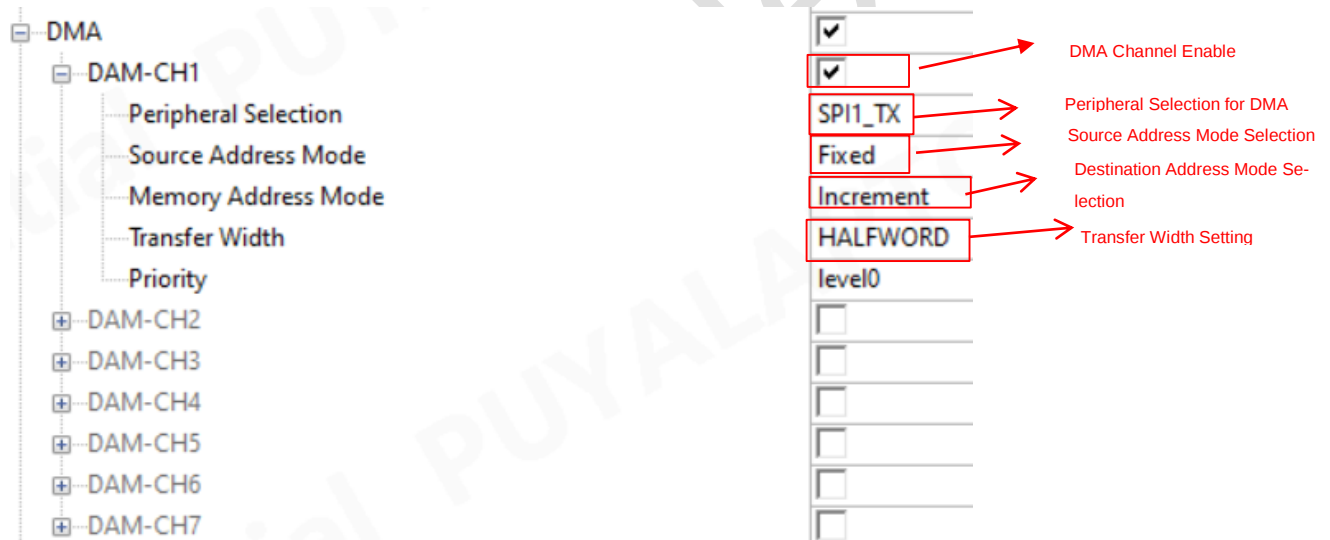


Figure 4-19 DMA channel configuration interface

4.3. Typical Application Drivers

4.3.1. Digital Tube / LED Driver

4.3.1.1. Common Cathode Digital Tube / LED Driver

The typical circuit for a common cathode digital tube / LED is as follows:

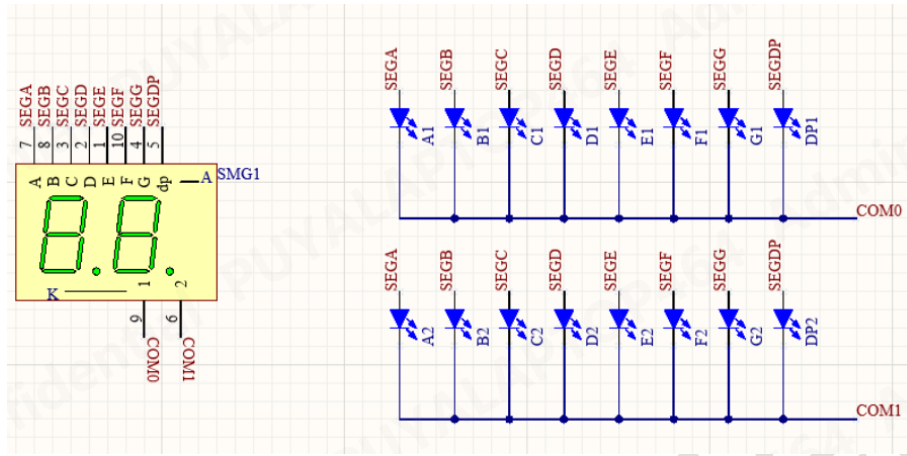


Figure 4-20 Common cathode digital tube typical circuit

The configuration interface is shown below:

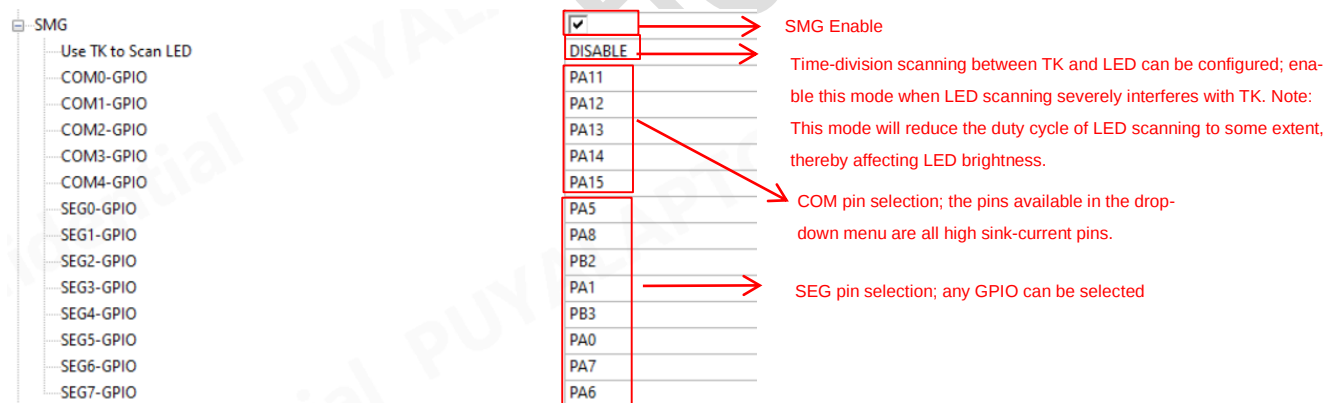


Figure 4-21 Common cathode digital tube/LED setting interface

Application example (smg_task.c) is as follows:

```

/*****
** Function name: void SMG_Show(uint8_t num)
** Description: Display decimal number using digital tube
** Input: num - number to display
** Return: None
*****/
void SMG_Show(uint16_t num)
{
    smg_data[0] = num_tbl[num / 10];
    smg_data[1] = num_tbl[num % 10];
}

/*****
** Function name: void LED_Show(uint8_t led_bit)
** Description: Button LED indicator
** Input: led_bit - bit position to display,
           bit0 indicates LED for TK0,
           bit1 indicates LED for TK1,
           .....
** Return: None
*****/
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;
}

```

The data in the array smg_data corresponds to the display content of the seven-segment display/LED. In application, simply write data to smg_data.

Figure 4-22 Common cathode digital tube/LED application example

4.3.1.2. Matrix LED Driver

A typical circuit for a matrix LED is shown below. For different matrices, the addresses corresponding to the LEDs are consistent.

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

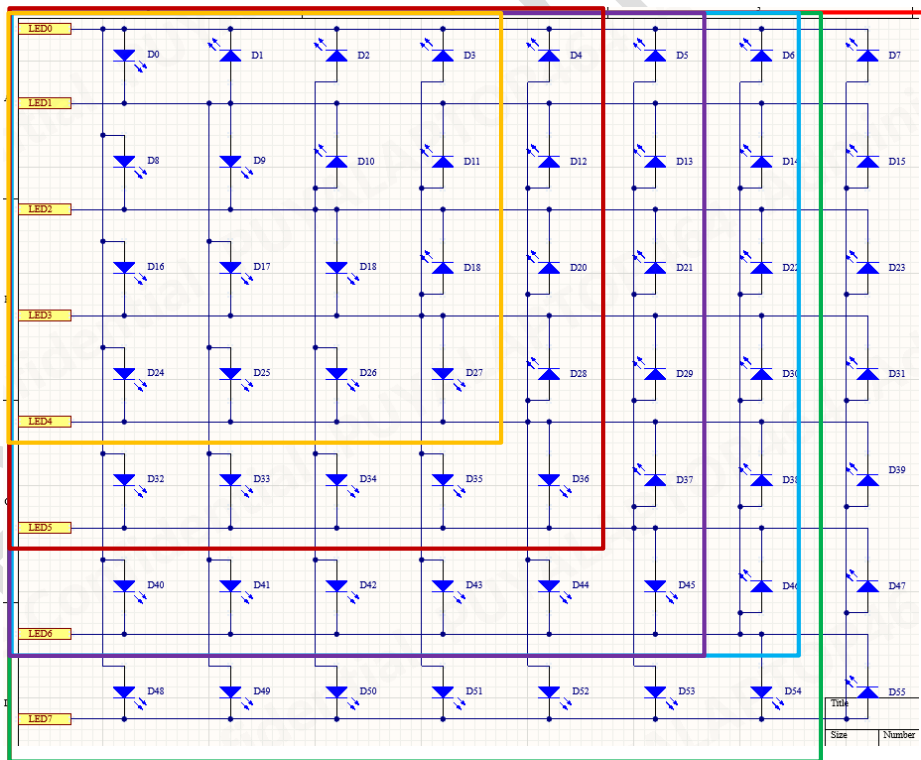


Figure 4-23 Matrix LED typical circuit diagram

LED display configuration buffer:

Control whether the LEDs are on by controlling the led_data buffer. 1: On; 0: Off.

Table 4-2 Matrix LED display configuration buffer

Buffer	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

The configuration interface is as follows:

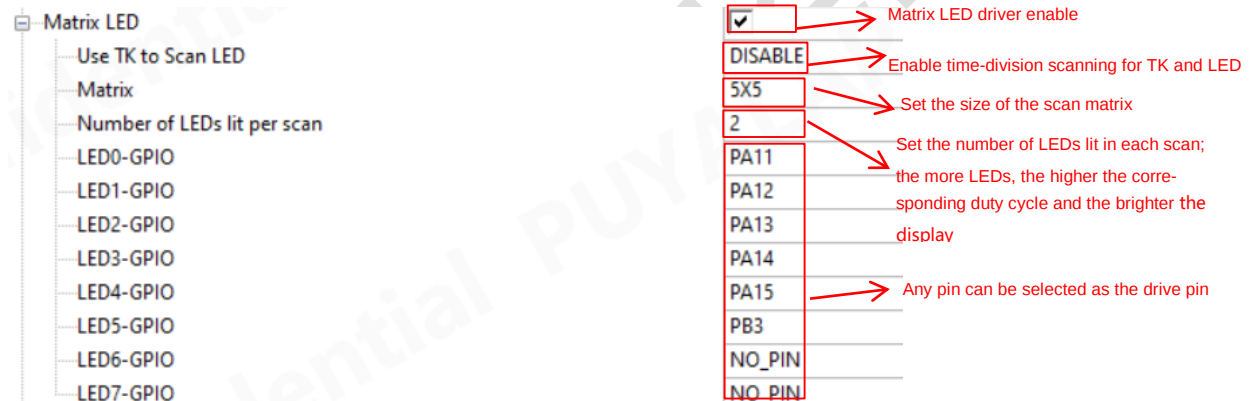


Figure 4-24 Matrix LED setting interface

4.3.2. User data storage

The setup interface and steps are as follows:



Figure 4-25 User data storage configuration and code example

```

#if APP_USER_OTP_ENABLE
uint8_t data;
/* Second power-on, read data */
if (User_Cache_Read(0, &data, 1) == 1)
{
    log_printf("data:%d\n", data);
    data++;
}
else
{
    /* First power-on, or data has never been stored */
    log_printf("first power\n");
    data = 0;
}
/* Save data */
User_Cache_Write(0, &data, 1);
if (User_Flash_Write())
    log_printf("User_Flash_Write:OK\n");
else
    log_printf("User_Flash_Write:FAIL\n");
#endif

```

After power-up, read the data at address 0 and print it out for verification.

Write data to cache

Write the cache to FLASH. Since each FLASH write requires an erase, to reduce erase/write cycles it is recommended to first write all data to the cache and then call this function once to write them at once.

Figure 4-26 WS2812B-like LED driver configuration interface

4.3.3. WS2812B-like LED driver

This example uses SPI to emulate the communication timing of addressable RGB LEDs. The setup interface is as follows.

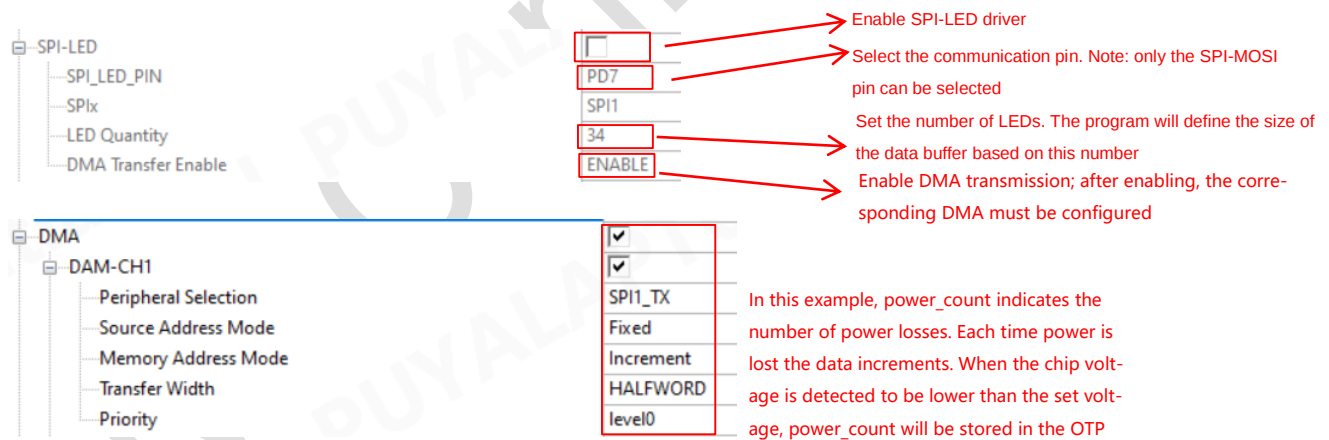


Figure 4-27 SPI_LED data transfer function

In the application, the program first loads the data to be sent into the buffer array `rgb_data` through the `SPI_LED_RgbLoad` function, and then calls the `SPI_LED_Transmit` function to send the data.

The `SPI_LED_Transmit` function is shown as follows:

```

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
}

```

Figure 4-28 Buzzer control configuration interface

4.3.4. Buzzer Control

The buzzer configuration interface is shown below:



Figure 4-29 BEEP_On function code example

In the application, the BEEP_On function can be called to control the buzzer. The input parameter is the buzzer sounding time, as shown below:

```

/*****
**  Function: void BEEP_On(uint16_t timeout)
**  Description: Turn on buzzer
**  Input: timeout, on time in ms
**  Return: None
*****/
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
}

```

Application Example:

```

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

```

Figure 4-30 Key trigger buzzer application code example

5. User Application Programming Interface

The User_code.c file contains the interface functions for the user application. The interface functions are listed in the table below:

Table 5-1 User application interface functions

Function Name	Functional overview
user_init	User application initialization. Users can place initialization code in this function.
user_loop	This function is called in the main loop. Users can place the code that runs in the main loop in this function.
user_timer	This function is called in the SYS_TICK interrupt. Users can place code that runs in the timer in this function.
ADC_Callback	This function is called after ADC conversion is complete. Users can read ADC data and add corresponding processing in this function.

6. Revision History

Version	Date	Descriptions
V1.0	2025.03.20	Initial version
V1.1	2026.05.19	Updated touch library and application-layer interface development description



Puya Semiconductor Co., Ltd.

IMPORTANT NOTICE

Puya reserve the right to make changes, corrections, enhancements, modifications to Puya products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information of Puya products before placing orders.

Puya products are sold pursuant to terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice and use of Puya products. Puya does not provide service support and assumes no responsibility when products that are used on its own or designated third party products.

Puya hereby disclaims any license to any intellectual property rights, express or implied.

Resale of Puya products with provisions inconsistent with the information set forth herein shall void any warranty granted by Puya.

Any with Puya or Puya logo are trademarks of Puya. All other product or service names are the property of their respective owners.

The information in this document supersedes and replaces the information in the previous version.

Puya Semiconductor Co., Ltd. – All rights reserved