3. Button control

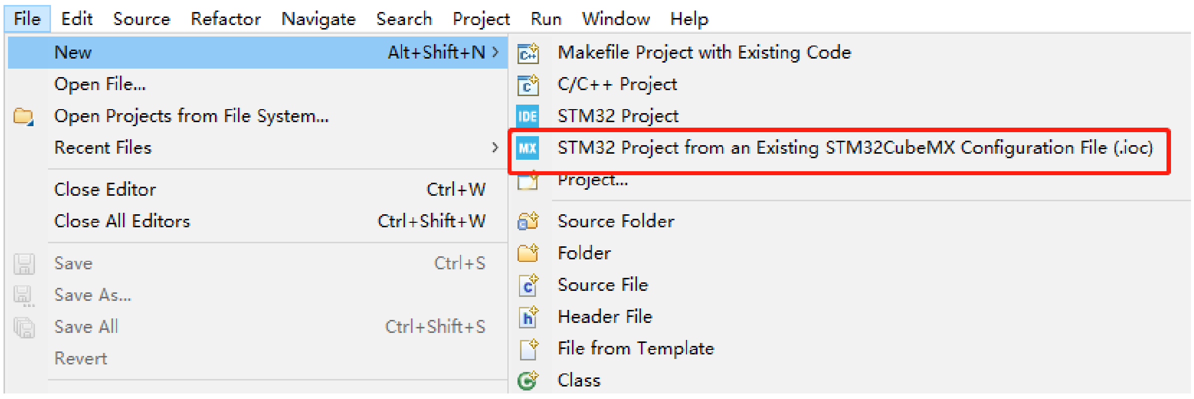
3.1. Purpose of the experiment

Detect the state of KEY1 on the expansion board and control the buzzer to sound. Each time the button is pressed, the buzzer will sound once.

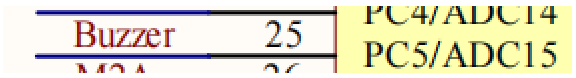
3.2. configure pin information

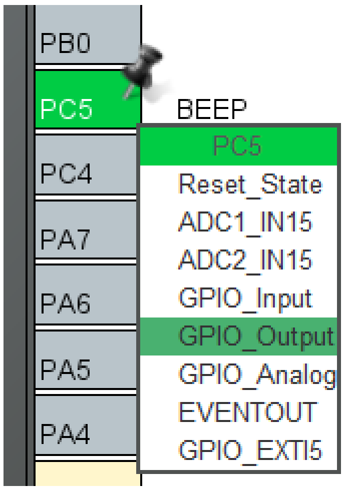
Since each new project needs configuration information, it is more troublesome. Fortunately, STM32CubeIDE provides the function of importing .ioc files, which can help us save time.

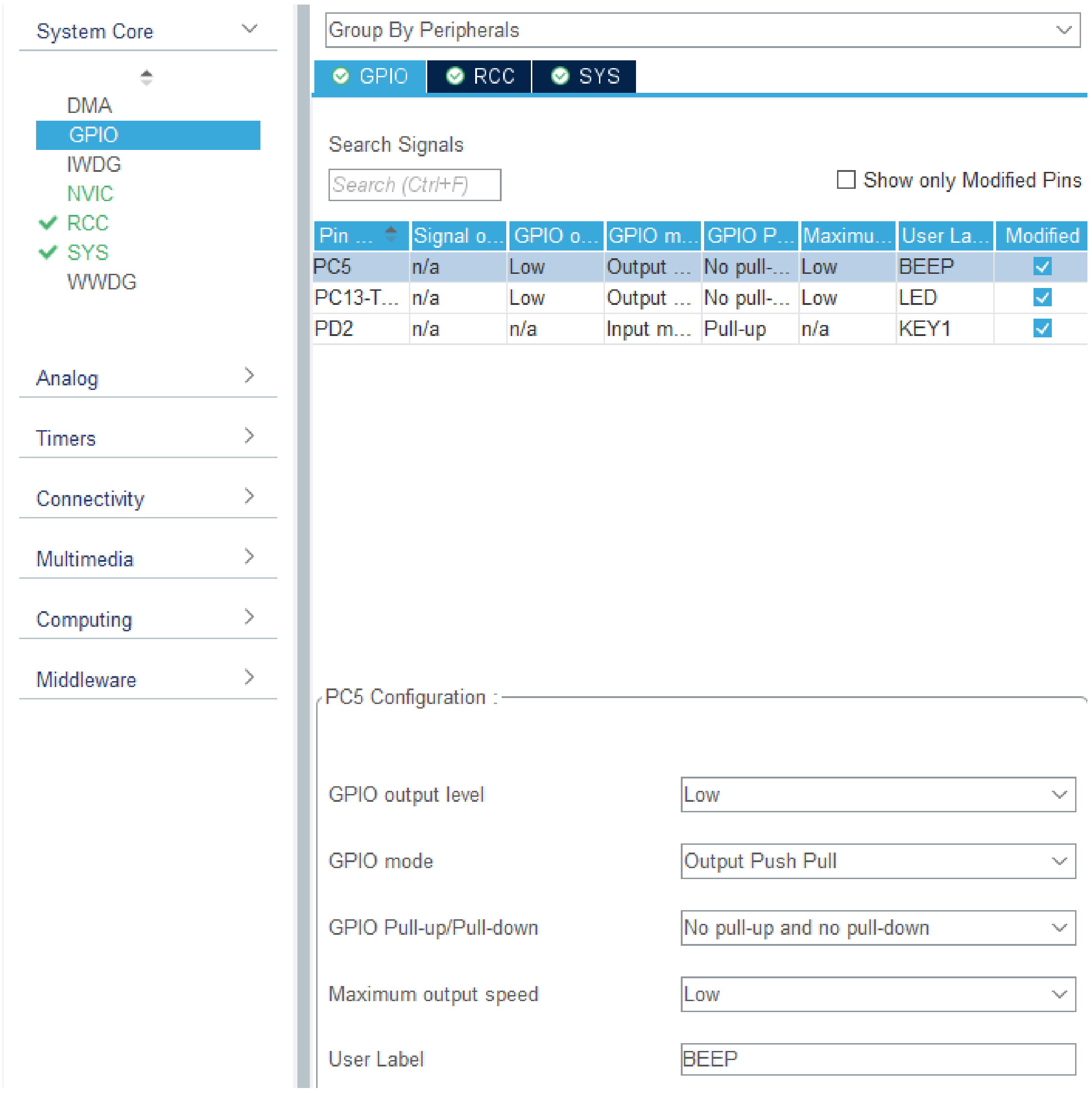
1. Import the ioc file from the LED project and name it BEEP.



2. According to the schematic diagram, the control pin of the buzzer is connected to the PC5 pin of the STM32 chip. It is necessary to set PC5 to GPIO_Output mode, and modify the label (Label) to BEEP. Other configurations are shown in the following figure.





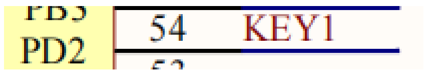


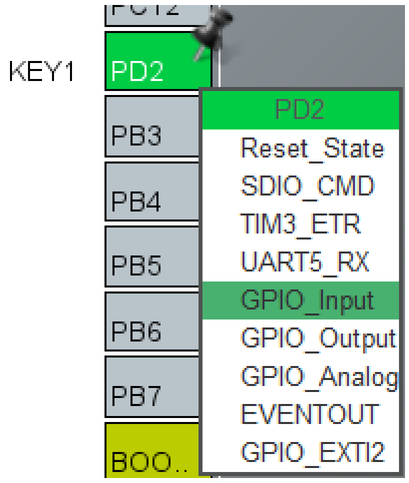
Pin ...	Signal o...	GPIO o...	GPIO m...	GPIO P...	Maximu...	User La...	Modified
PC5	n/a	Low	Output ...	No pull-...	Low	BEEP	✓
PC13-T...	n/a	Low	Output ...	No pull-...	Low	LED	✓
PD2	n/a	n/a	Input m...	Pull-up	n/a	KEY1	✓

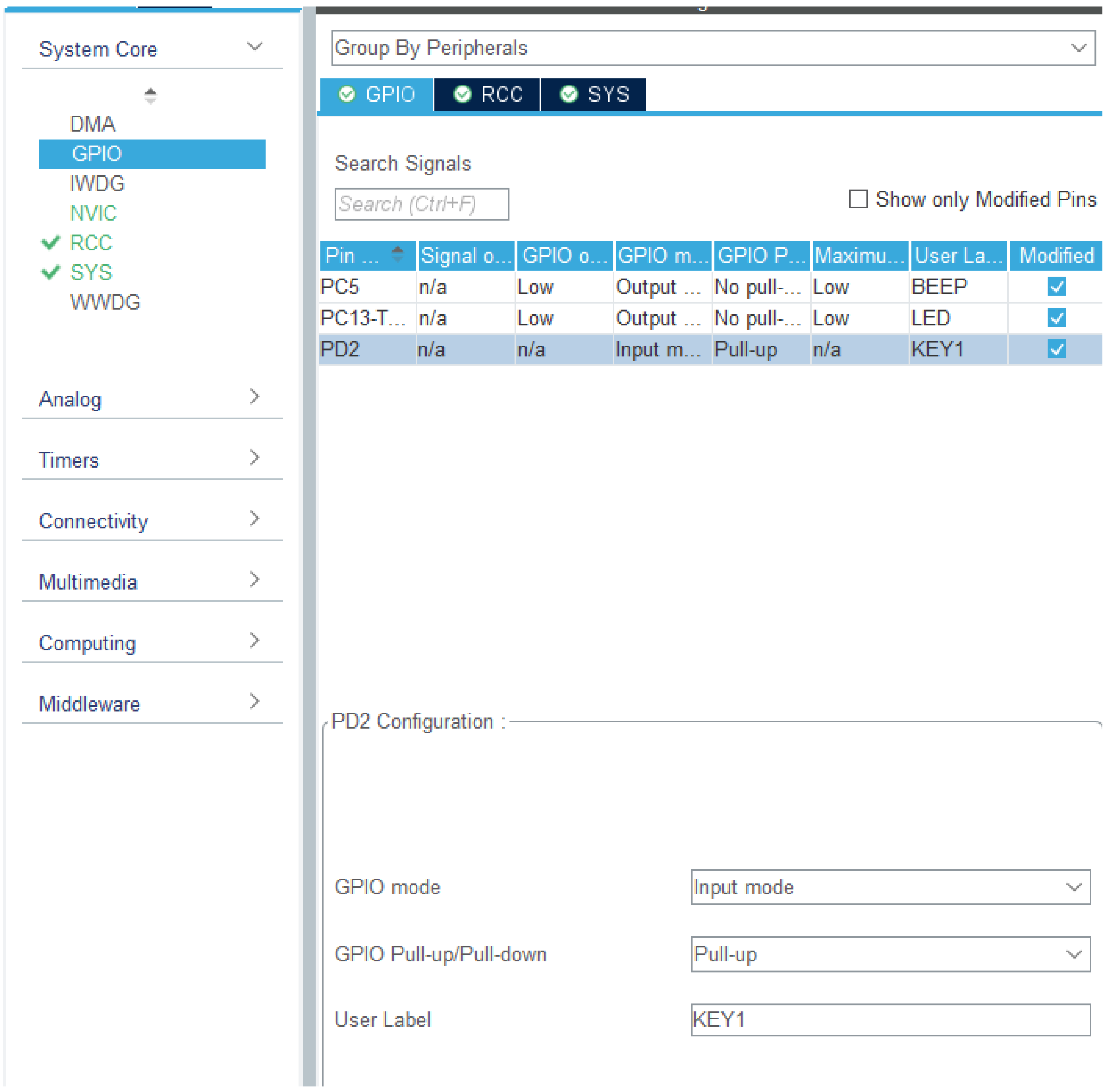
PC5 Configuration :

GPIO output level	Low
GPIO mode	Output Push Pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	Low
User Label	BEEP

3. The button KEY1 is connected to the PD2 pin. It is necessary to set PD2 to GPIO_Input mode, and modify the label (Label) to KEY1. Other configurations are shown in the following figure.







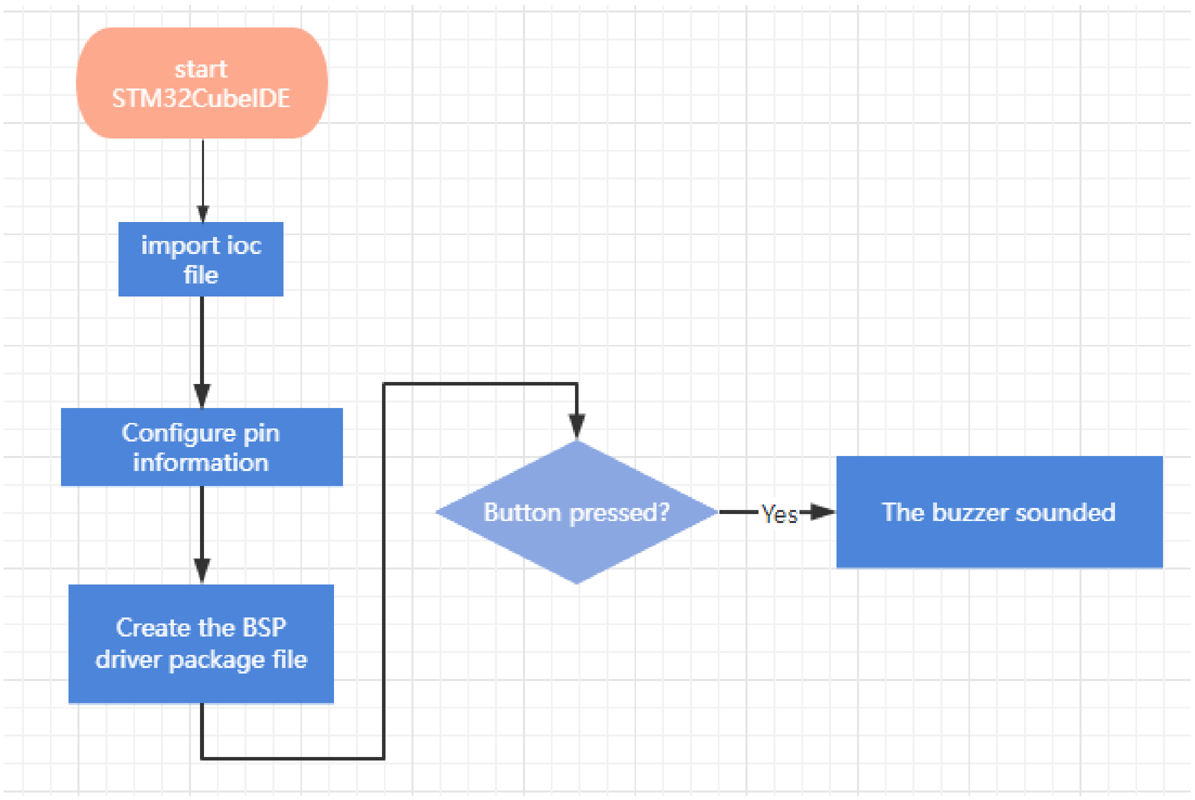
Pin ...	Signal o...	GPIO o...	GPIO m...	GPIO P...	Maximu...	User La...	Modified
PC5	n/a	Low	Output ...	No pull-...	Low	BEEP	✓
PC13-T...	n/a	Low	Output ...	No pull-...	Low	LED	✓
PD2	n/a	n/a	Input m...	Pull-up	n/a	KEY1	✓

PD2 Configuration :

GPIO mode	Input mode
GPIO Pull-up/Pull-down	Pull-up
User Label	KEY1

Save and generate code.

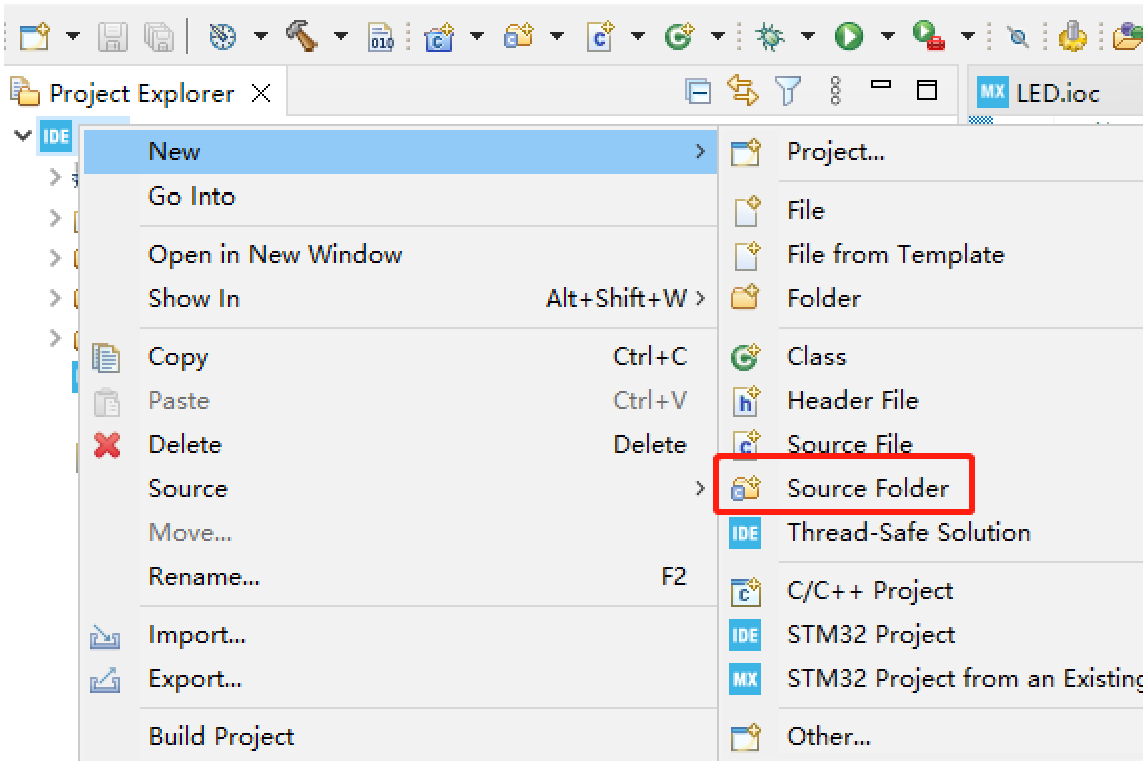
3.3. Analysis of the experimental flow chart

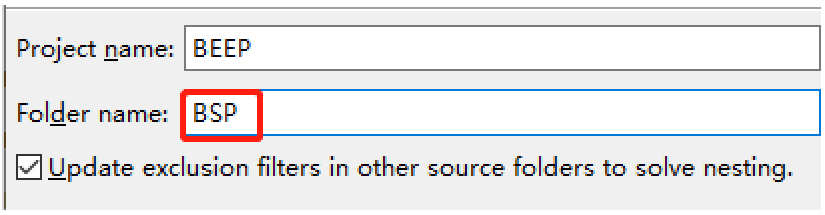


3.4. add file structure

The pin information is configured graphically, and the generated code already contains the system initialization content, so there is no need to additionally initialize the system configuration.

1. For the convenience of management, we create a new BSP source code folder. Right-click on the mouse to move the project name->New->Source Folder



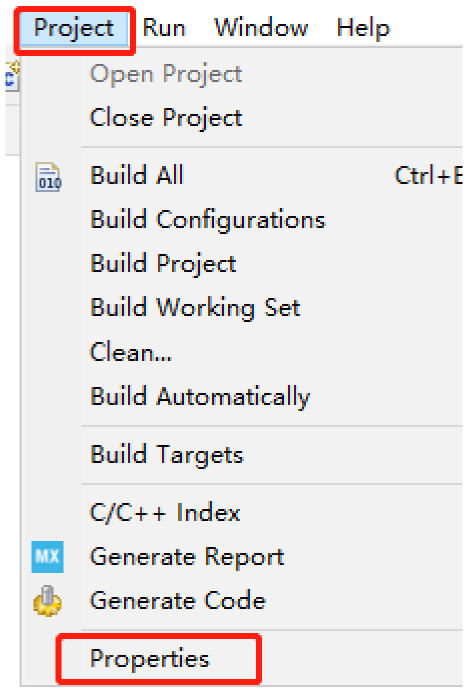


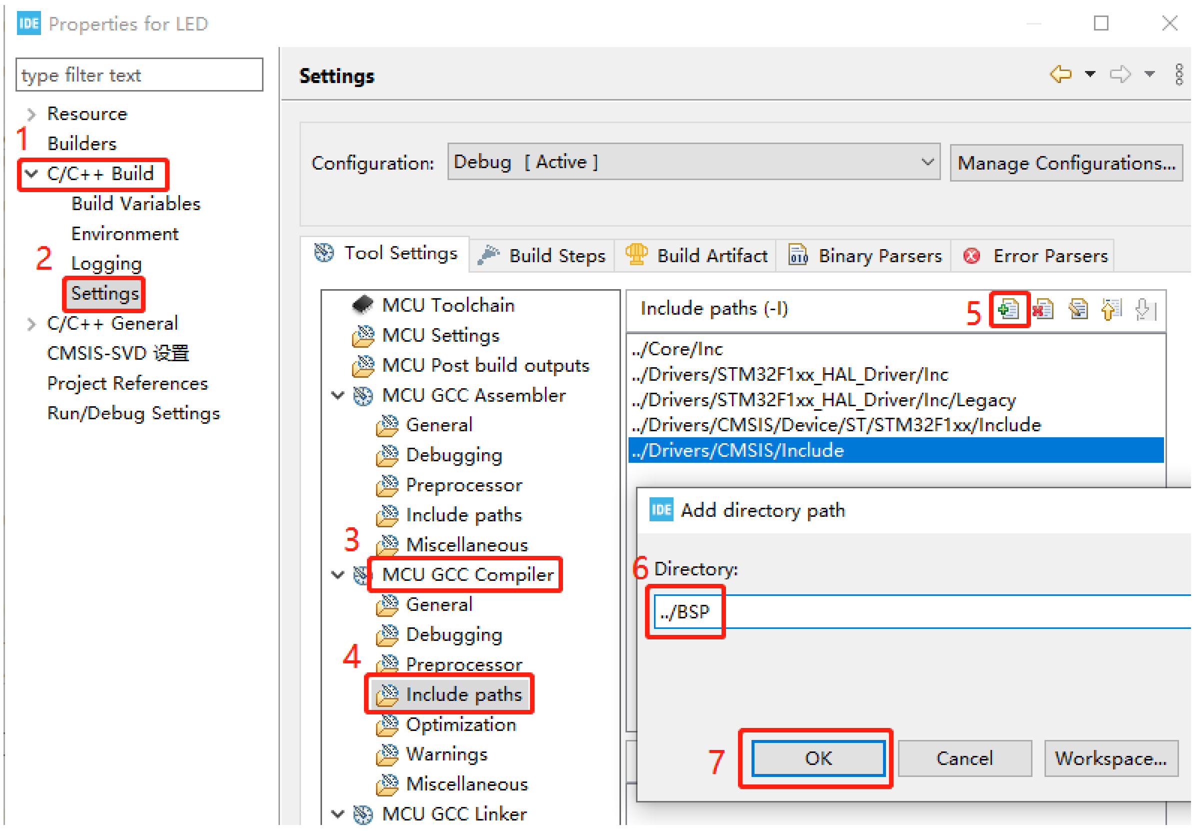
Project name: BEEP

Folder name: BSP

☑ Update exclusion filters in other source folders to solve nesting.

2. Add the BSP to the environment. Click Project->Properties->C/C++ Build->settings->MCU GCC Compiler->include paths, and then click the Add button to fill in ../BSP and save it.





Configuration: Debug [Active]   Manage Configurations...

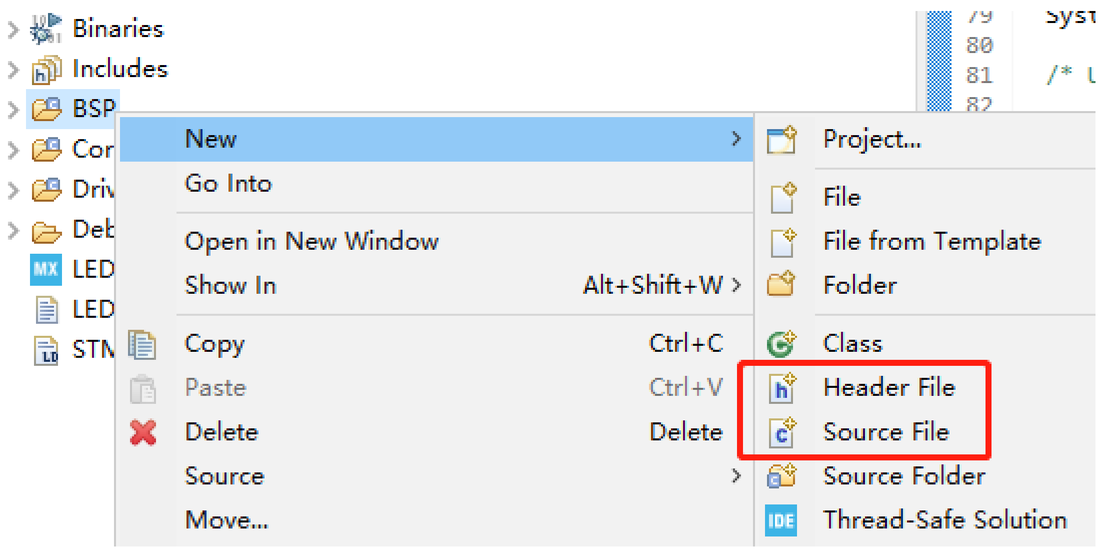
Include paths (-I)

```
../Core/Inc
../Drivers/STM32F1xx_HAL_Driver/Inc
../Drivers/STM32F1xx_HAL_Driver/Inc/Legacy
../Drivers/CMSIS/Device/ST/STM32F1xx/Include
../Drivers/CMSIS/Include
```

Directory: ../BSP

3. Create a new bsp.h and a bsp.c file, right-click BSP->New->Header File/Source File, and then enter the corresponding name.

These two files are mainly responsible for linking some functions in main.c, which can avoid repetitive code writing.



4. Add the following content in bsp.h: Make the control of the LED into a macro definition, which is simple and fast. The new Bsp_Init() function is mainly responsible for initialization, and Bsp_Loop() is mainly responsible for the main program content. The Bsp_Led_Show_State_Handle() function is mainly responsible for the blinking effect of the LED indicator, which is used to indicate that the system is running.

```
/* DEFINE */
#define LED_ON()          HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, SET)
#define LED_OFF()         HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, RESET)
#define LED_TOGGLE()      HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin)


/* functions */
void Bsp_Init(void);
void Bsp_Loop(void);
void Bsp_Led_Show_State_Handle(void);
```

5. Import the bsp.h header file in the main.c file.

```
/* Private includes -------------
/* USER CODE BEGIN Includes */
#include "bsp.h"

/* USER CODE END Includes */
```

6. Call Bsp_Init() in the main function.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
Bsp_Init();
/* USER CODE END 2 */
```

7. Call Bsp_Loop() in while(1).

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    Bsp_Loop();

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

3.5. core code explanation

1. Create new buzzer driver library bsp_beep.h and bsp_beep.c files in BSP. Add the following to bsp_beep.h:

```
#define BEEP_ON()        HAL_GPIO_WritePin(BEEP_GPIO_Port, BEEP_Pin, SET)
#define BEEP_OFF()       HAL_GPIO_WritePin(BEEP_GPIO_Port, BEEP_Pin, RESET)


void Beep_Timeout_Close_Handle(void);
void Beep_On_Time(uint16_t time);
```

The Beep_Timeout_Close_Handle() function needs to be called every 10 milliseconds, so as to ensure that the Beep_On_Time() function will follow the normal effect after setting the time. The time in Beep_On_Time(time) represents the time when the buzzer is turned on. If time=0, the buzzer will be turned off. If time=1, the buzzer will keep ringing. If time>=10, the buzzer will sound after time milliseconds. Auto-off (time should be a multiple of 10).

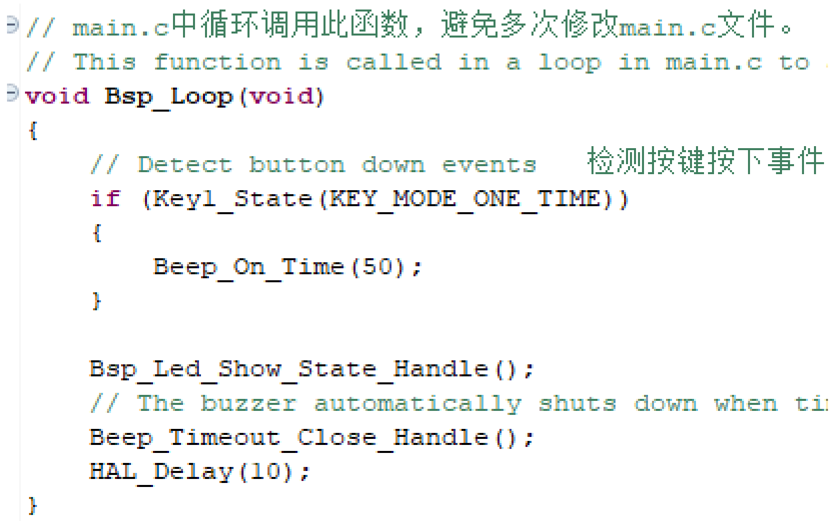
2. Create new buzzer driver library bsp_key.h and bsp_key.c files in BSP. Add the following to bsp_key.h:

```
#define KEY_PRESS           1
#define KEY_RELEASE         0

#define KEY_MODE_ONE_TIME   1
#define KEY_MODE_ALWAYS     0


uint8_t Key1_State(uint8_t mode);
```

The function of the Key1_State(mode) function is to detect whether the key is pressed, and it needs to be called every 10 milliseconds. You can enter 0 or 1 for mode, mode=0 means that pressing KEY1 will always return to KEY_PRESS, and releasing it will return to KEY_RELEASE, mode=1 means that no matter how long KEY1 is pressed, only return KEY_PRESS once, otherwise return KEY_RELEASE.

3. Add the start-up buzzer in Bsp_Init() to sound for 50 milliseconds, and detect whether the button is pressed in Bsp_Loop(). If it is pressed, it will automatically turn off after 50 milliseconds. At the bottom are the control handles for the LED lights and buzzer, which only need to be called every 10 milliseconds.

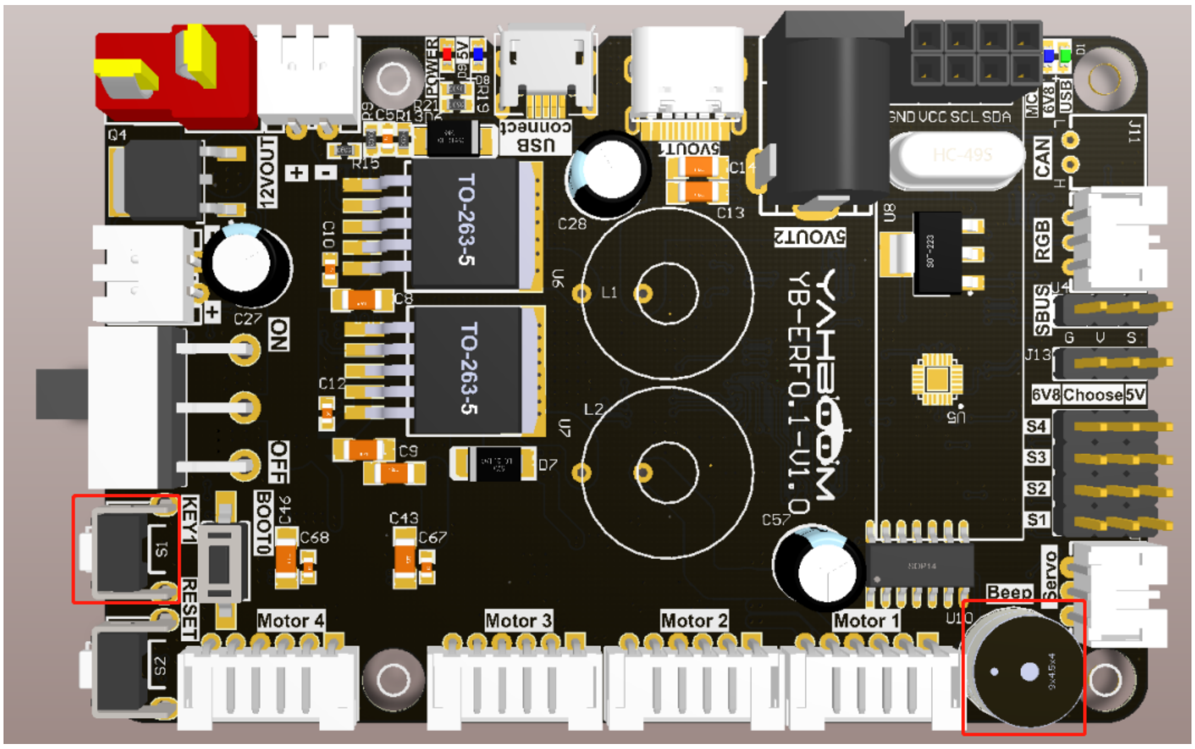
```
// The peripheral device is initialized  外设设备初始化
void Bsp_Init(void)
{
    Beep_On_Time(50);
}
```



```
// main.c中循环调用此函数，避免多次修改main.c文件。
// This function is called in a loop in main.c to
void Bsp_Loop(void)
{
    // Detect button down events    检测按键按下事件
    if (Key1_State(KEY_MODE_ONE_TIME))
    {
        Beep_On_Time(50);
    }

    Bsp_Led_Show_State_Handle();
    // The buzzer automatically shuts down when ti
    Beep_Timeout_Close_Handle();
    HAL_Delay(10);
}
```

3.6. hardware connection

The buttons KEY1 and the buzzer are all onboard components and do not need to be connected manually.



3.7. Experimental effect

After the program is programmed, the buzzer will ring for 50 milliseconds first, the LED light will flash every 200 milliseconds, and the buzzer will ring for 50 milliseconds each time the button is pressed.