Button control

1. Learning objectives

1. Learn the basic working principle of the buttons of the MSPM0G3507 motherboard.
2. Use the onboard buttons to control the on/off of the onboard LED lights.

2. Hardware construction

Independent button principle

Independent buttons are usually implemented through **mechanical switches**, and their hardware structure consists of buttons, connecting lines, and pull-up/pull-down resistors. When the button is closed, the line is connected and the signal level changes; when the button is released, the line is disconnected and the signal returns to its initial state.

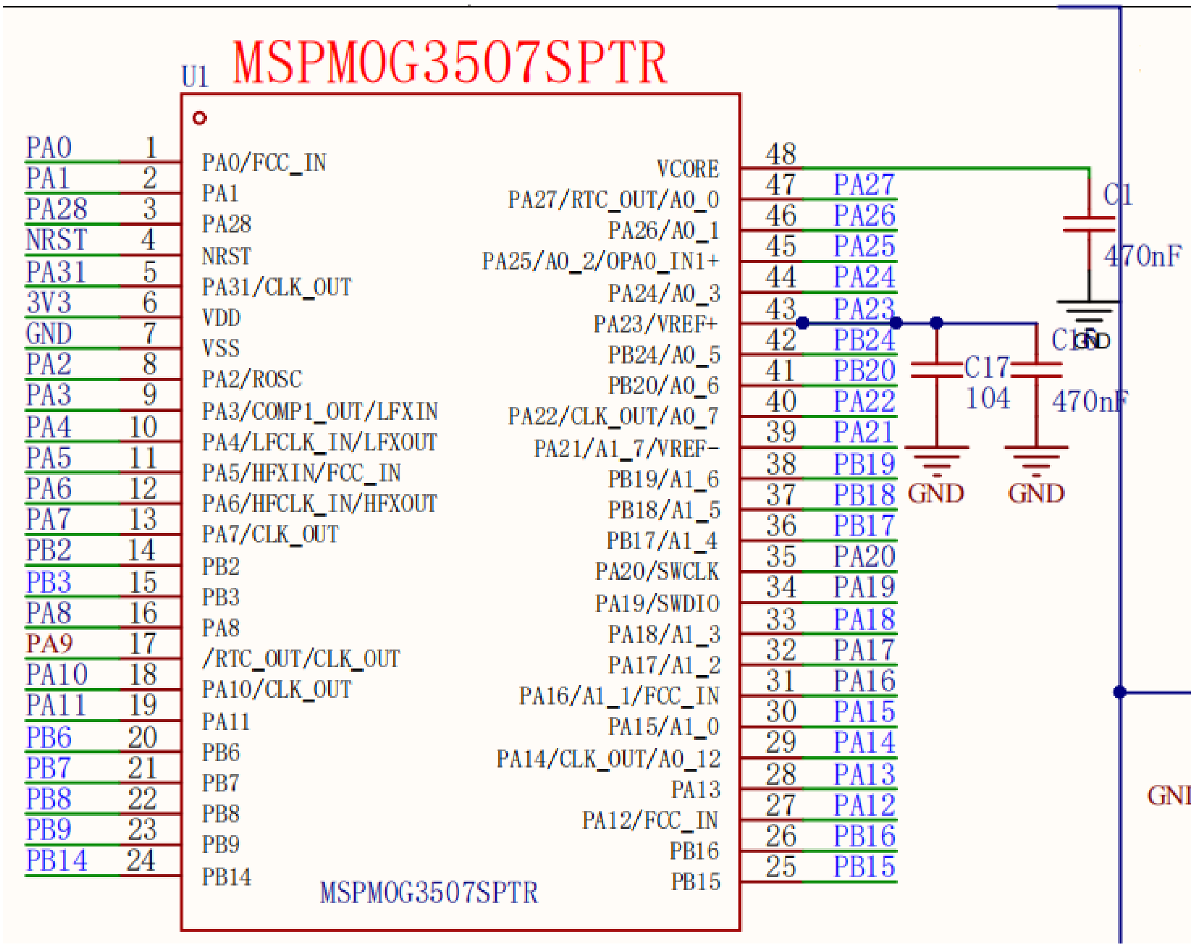
Key debounce

The mechanical elasticity of the button will cause the signal to jump multiple times in a short period of time when the button is closed or released, which is called **button jitter**. Usually, jitter is eliminated by **software delay debounce** or hardware plus **RC filter circuit**.

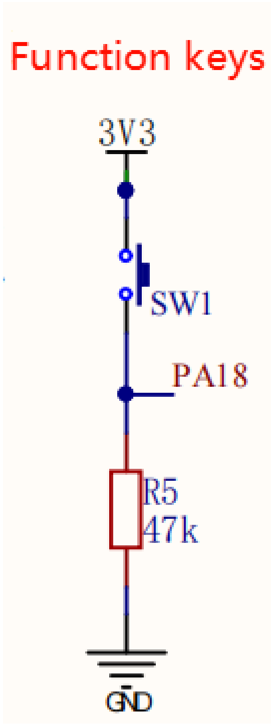
This course does not require additional hardware equipment, and directly uses the onboard function buttons on the MSPM0G3507 motherboard.

In the schematic diagram of the development board, a button is connected to the PA18 pin. One end of the button is connected to a high level (3.3V), and the other end is connected to a GPIO pin. At the same time, the PA18 pin is connected to the ground through a 47kΩ pull-down resistor R5. When the button is not pressed, the PA18 pin level is pulled down to 0V due to the pull-down resistor; when the button is pressed, the PA18 pin is directly connected to a high level of 3.3V, thereby achieving a change in high and low levels. The development board can determine whether the button is pressed by detecting the level state of the PA18 pin. This design is simple and stable, and can effectively avoid signal interference caused by pin suspension.

MSPM0G3507 main control diagram



Key schematic diagram

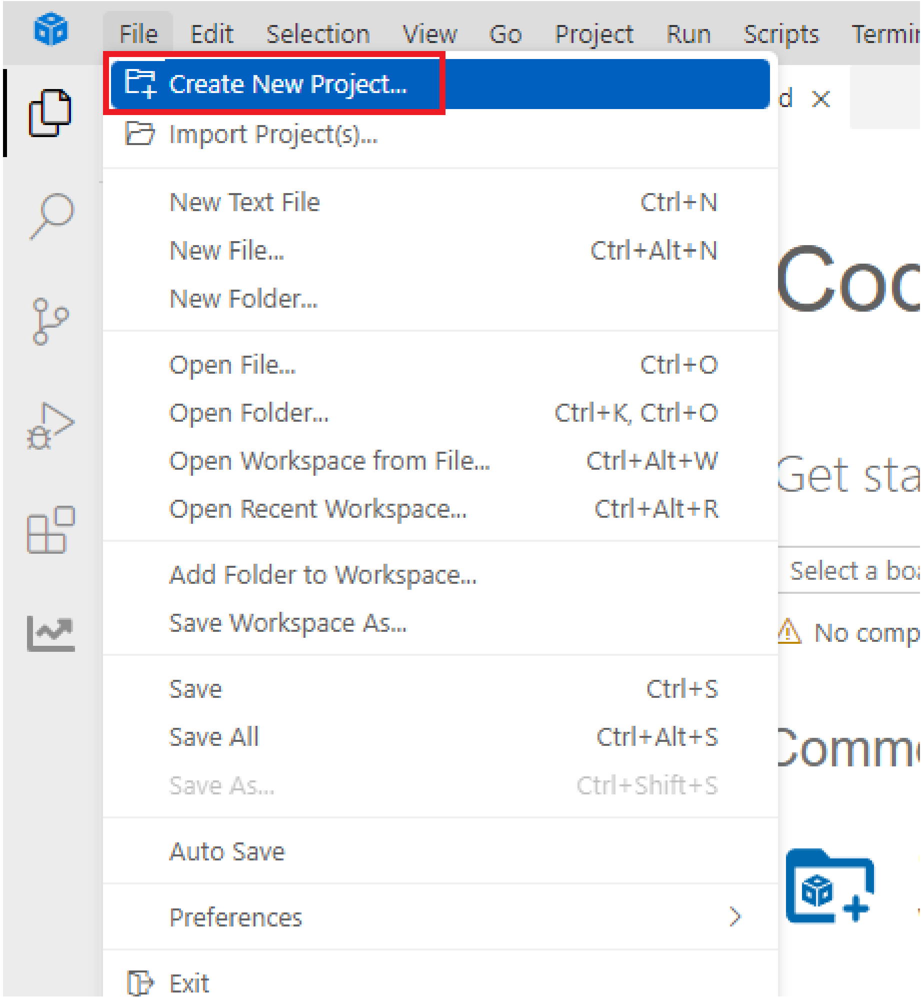


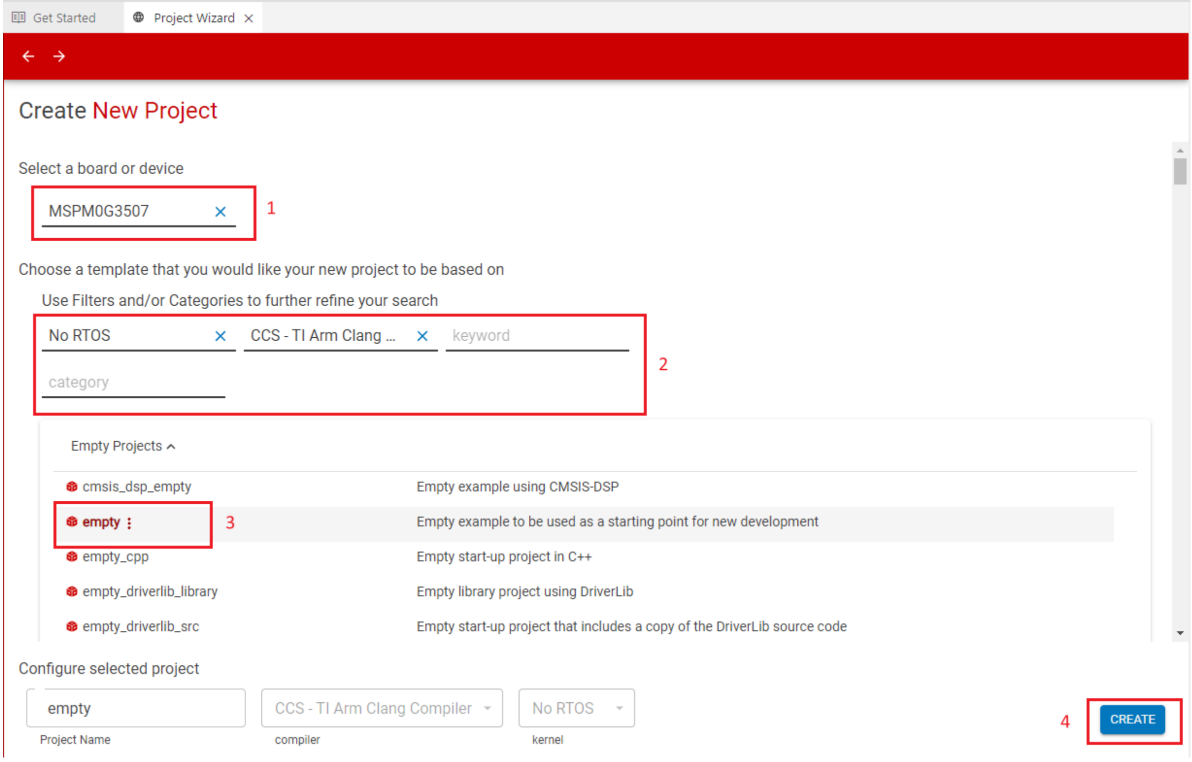
3. Experimental steps

In the microcontroller, the input mode of the pin refers to configuring a pin as an input function to receive changes in external signals. The working principle of the key is based on this point: by setting the pin connected to the key to input mode, the microcontroller can detect the level change on the pin. When the key is not pressed, the pin level remains at a low level; when the key is pressed, the pin level becomes a high level. This change in high and low levels is the embodiment of the key input signal, and it is also the basis for the microcontroller to judge the key status. Therefore, the process of receiving external signals is actually the process of detecting the high and low level changes of the key through the input mode.

1. Open the SYSCONFIG configuration tool

Create a blank project in CCS called empty.





Create New Project

Select a board or device

MSPM0G3507 1

Choose a template that you would like your new project to be based on

Use Filters and/or Categories to further refine your search

No RTOS CCS - TI Arm Clang ... keyword 2

category

Empty Projects

cmsis_dsp_empty	Empty example using CMSIS-DSP
empty 3	Empty example to be used as a starting point for new development
empty_cpp	Empty start-up project in C++
empty_driverlib_library	Empty library project using DriverLib
empty_driverlib_src	Empty start-up project that includes a copy of the DriverLib source code

Configure selected project

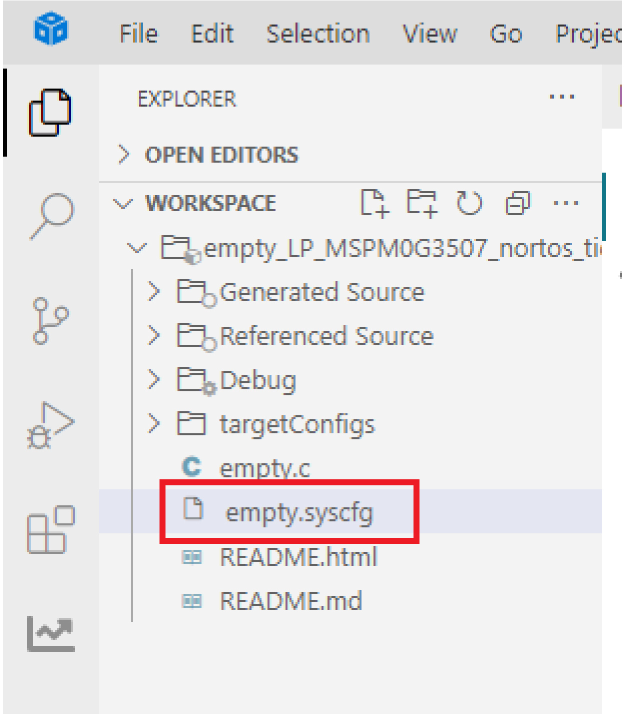
empty — Project Name

CCS - TI Arm Clang Compiler — compiler

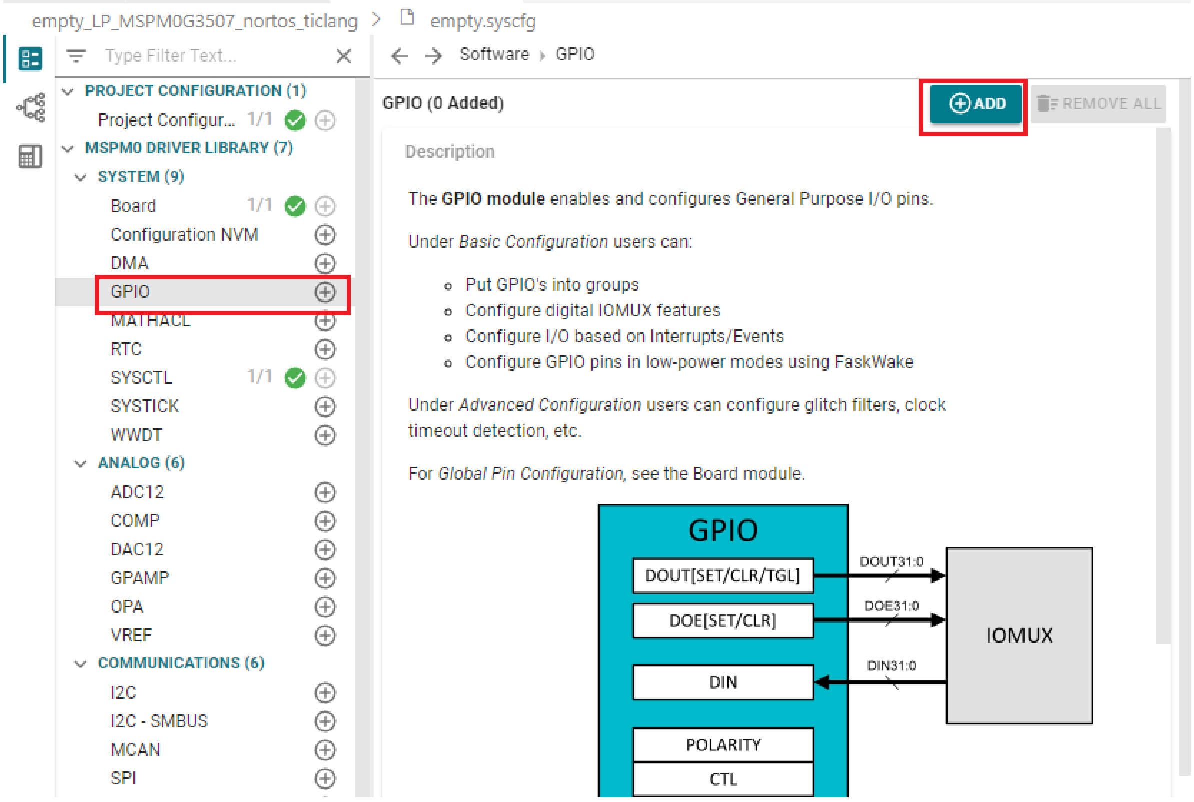
No RTOS — kernel

4 CREATE

Find and open the empty.syscfg file in the left workspace of CCS.



In SYSCONFIG, select MCU peripherals on the left, find the GPIO option and click to enter. Click ADD in GPIO to add two sets of pin peripherals, one set of LED pins and one set of button pins. For LED pins, please refer to the previous **Light up LED** course. Here we only talk about the configuration of key pins.



The **GPIO module** enables and configures General Purpose I/O pins.

Under *Basic Configuration* users can:

- Put GPIO's into groups
- Configure digital IOMUX features
- Configure I/O based on Interrupts/Events
- Configure GPIO pins in low-power modes using FaskWake

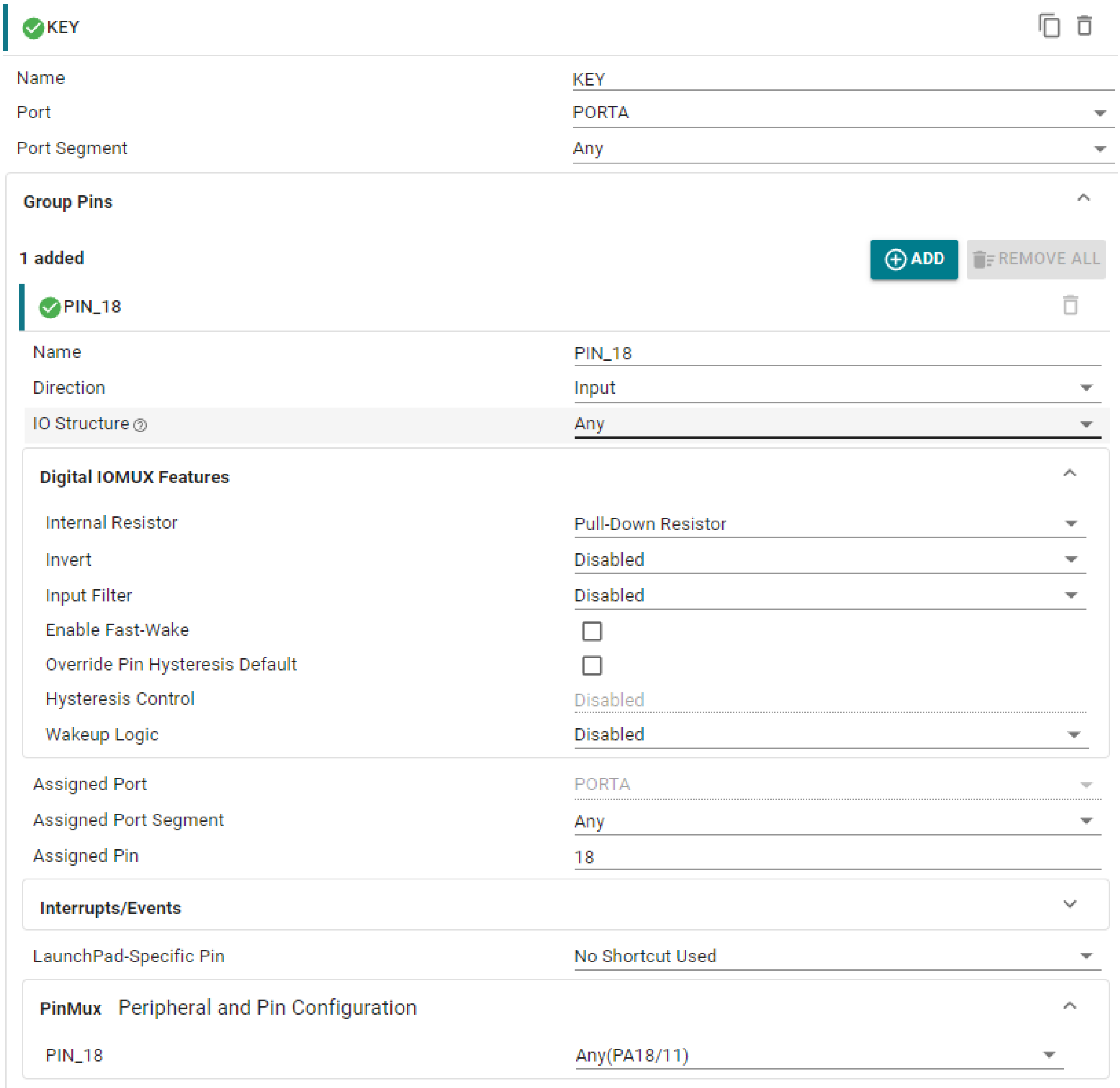
Under *Advanced Configuration* users can configure glitch filters, clock timeout detection, etc.

For *Global Pin Configuration*, see the Board module.

2. Configure pin mode

Configure pin parameters and name the GPIO instance as `KEY`. Since PA18 is connected, select `PORTA` for Port.

Set the pin name to `PIN_18` and set it to input mode `Input`. Set Internal Resistor to pull-down resistor `Pull-Down Resistor`, and set the pin to be controlled in Assigned Pin to 18.



✓ KEY

Name	KEY
Port	PORTA
Port Segment	Any

Group Pins

1 added

✓ PIN_18

Name	PIN_18
Direction	Input
IO Structure	Any

Digital IOMUX Features

Internal Resistor	Pull-Down Resistor
Invert	Disabled
Input Filter	Disabled
Enable Fast-Wake	□
Override Pin Hysteresis Default	□
Hysteresis Control	Disabled
Wakeup Logic	Disabled

Assigned Port	PORTA
Assigned Port Segment	Any
Assigned Pin	18

Interrupts/Events

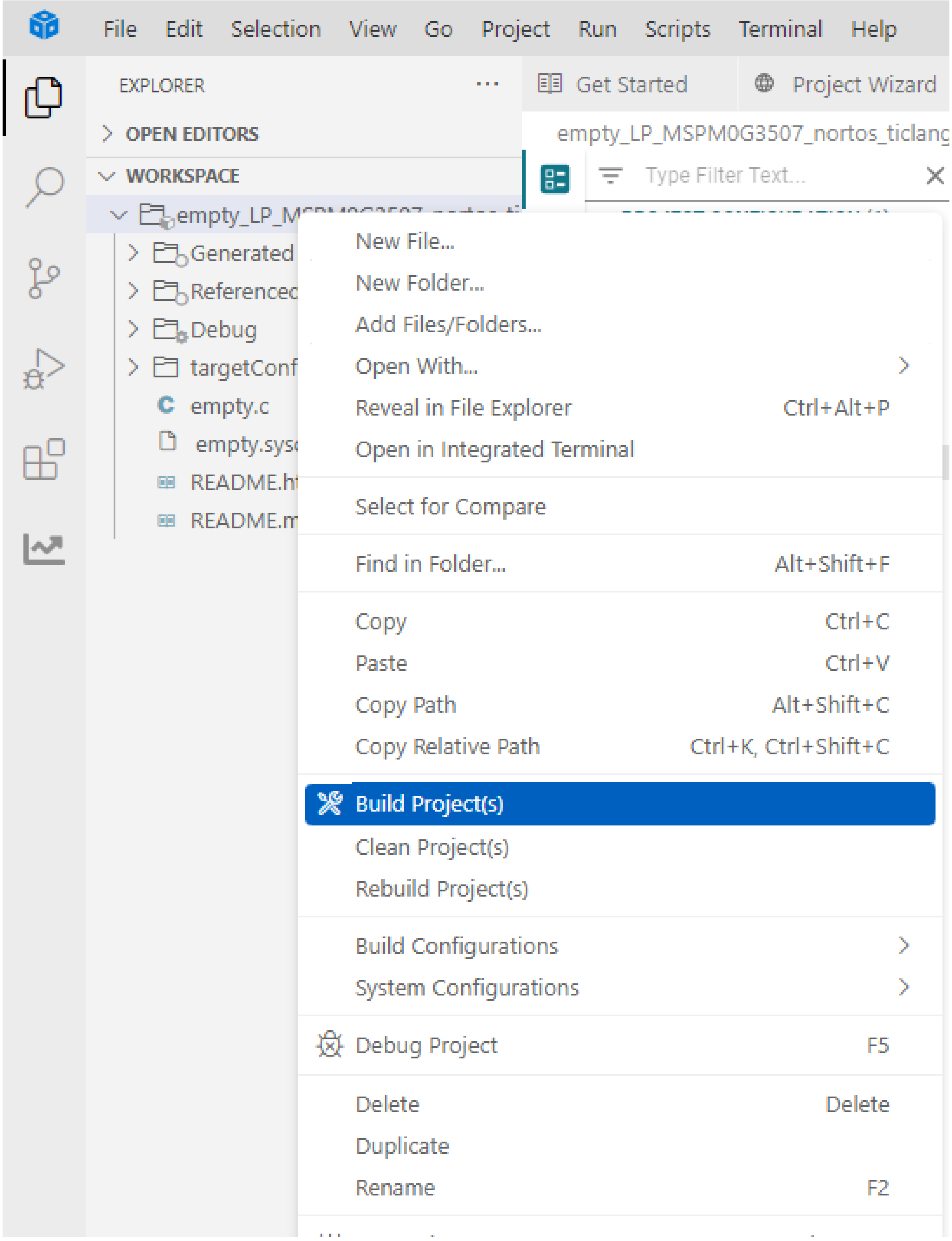
LaunchPad-Specific Pin	No Shortcut Used

PinMux Peripheral and Pin Configuration

PIN_18	Any(PA18/11)

Save the configuration in the .syscfg file using the shortcut key Ctrl+S.

Compile after saving



3. Write the program

In the empty.c file, write the following code

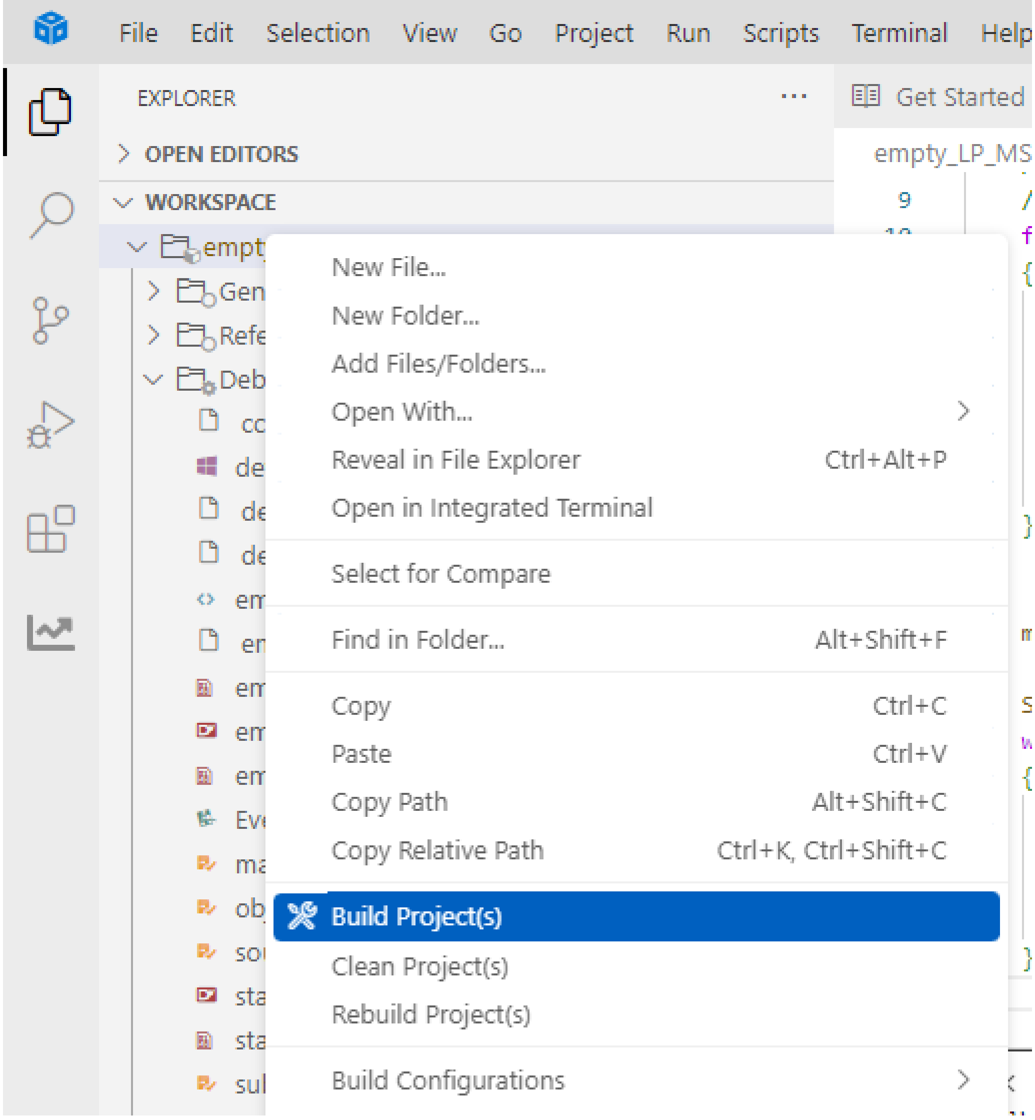
```
#include "ti_msp_dl_config.h"

int main(void)
{

        SYSCFG_DL_init();

        while (1)
        {
                //如果读取到的引脚值大于0，说明PA18引脚为高电平
                // If the pin value read is greater than 0, it means that the
PA18 pin is high level.
                if( DL_GPIO_readPins(KEY_PORT, KEY_PIN_18_PIN) > 0 )
                {
                        DL_GPIO_setPins(LED1_PORT,LED1_PIN_2_PIN);  //LED控制输出高电平 LED control output high level
                }
                else//如果PA18引脚为低电平 If the PA18 pin is low
                {
                        DL_GPIO_clearPins(LED1_PORT,LED1_PIN_2_PIN);//LED控制输出低电平 LED control output low level
                }
        }
}
```

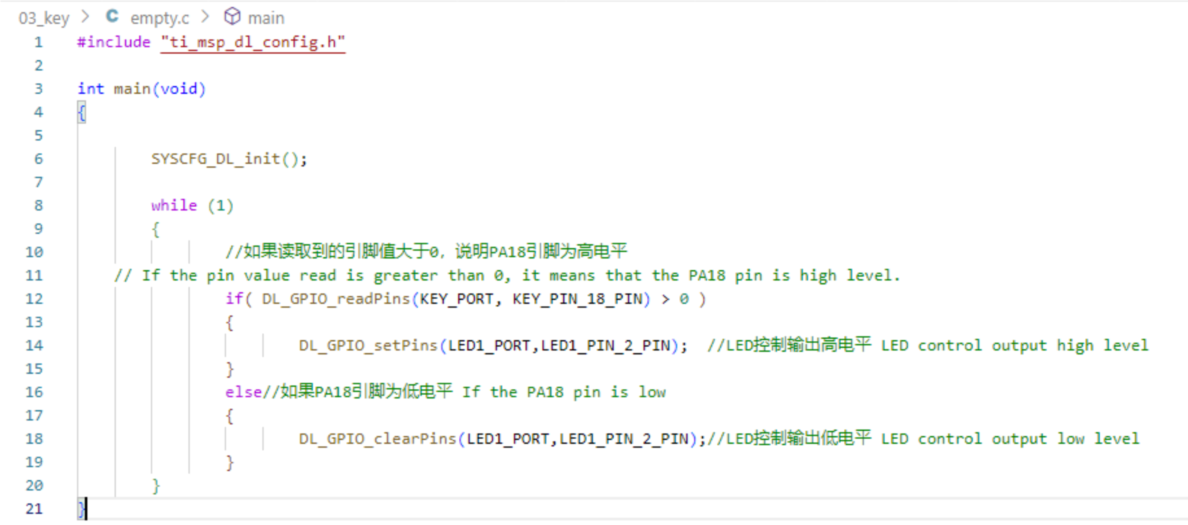
4. Compile



If the compilation is successful, you can download the program to the development board.

4. Program Analysis

- empty.c



```c
#include "ti_msp_dl_config.h"

int main(void)
{

    SYSCFG_DL_init();

    while (1)
    {
        //如果读取到的引脚值大于0，说明PA18引脚为高电平
  // If the pin value read is greater than 0, it means that the PA18 pin is high level.
        if( DL_GPIO_readPins(KEY_PORT, KEY_PIN_18_PIN) > 0 )
        {
            DL_GPIO_setPins(LED1_PORT,LED1_PIN_2_PIN);  //LED控制输出高电平 LED control output high level
        }
        else//如果PA18引脚为低电平 If the PA18 pin is low
        {
            DL_GPIO_clearPins(LED1_PORT,LED1_PIN_2_PIN);//LED控制输出低电平 LED control output low level
        }
    }
}
```

uint32_t DL_GPIO_readPins(GPIO_Regs* gpio, uint32_t pins) is used to read the level status of the specified GPIO pin. Get the status of the specified pin by reading the value of the GPIO register. Enter the GPIO port and pin number to be read, and return the level status of the pin.

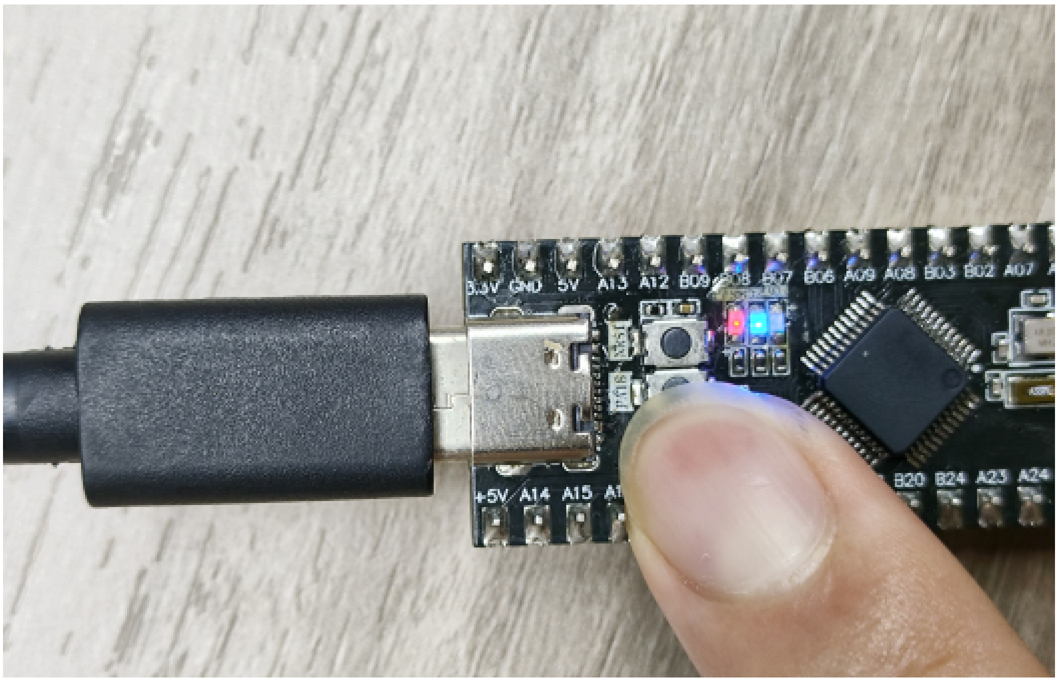
gpio: GPIO port to be read.

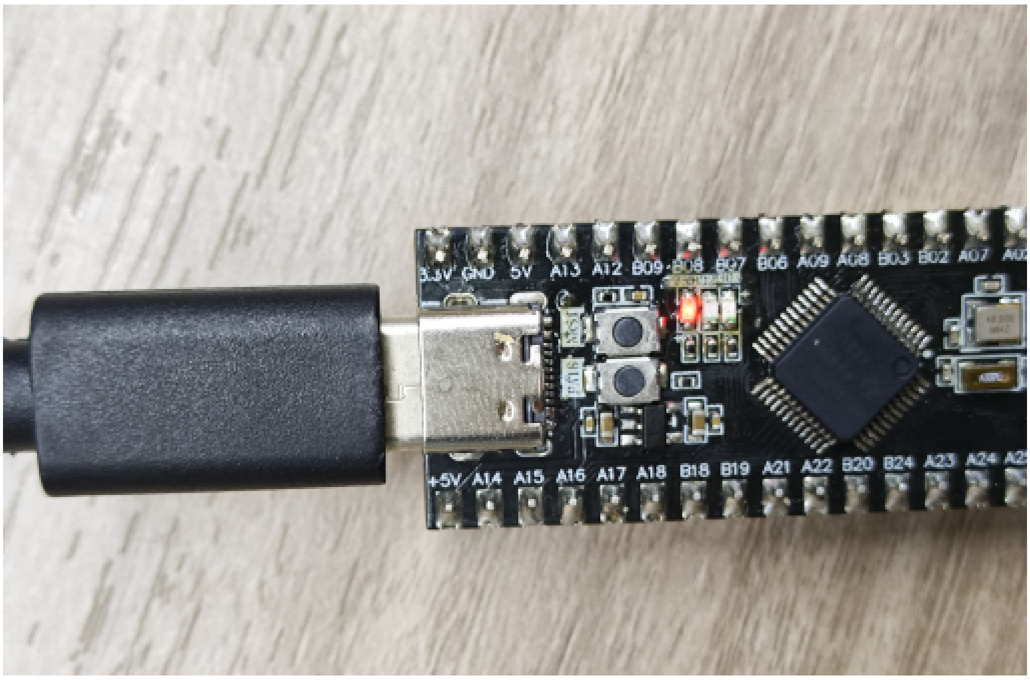
pins: pin number to be read.

This code continuously detects the level state of pin `PA18` (button access pin) and controls the output level of `LED1_PIN_2` (LED control pin) to realize the function of key control LED. When `PA18` is high level, the LED is turned on (LED pin outputs high level); when `PA18` is low level, the LED is turned off (LED pin outputs low level).

5. Experimental phenomenon

After the program is downloaded, the LED light can be controlled by the button. Press the button to turn on the LED light, and release the button to turn off the LED light.




3.3V GND 5V A13 A12 B09 B08 B07 B06 A09 A08 B03 B02 A07
+5V A14 A15 A16 A17 A18 B18 B19 A21 A22 B20 B24 A23 A24