LED Control

I. Learning Objectives

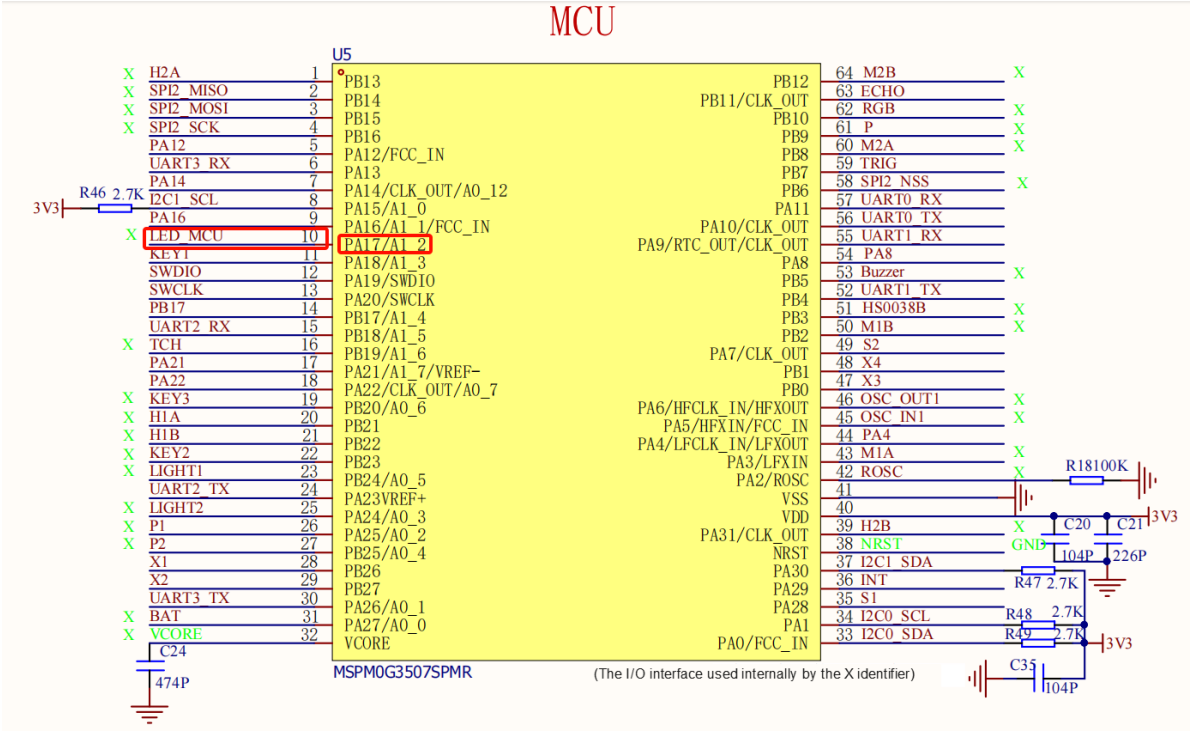
1. Learn the basic usage of MSPM0G3507 development board pins.
2. Understand how to control the onboard LED.

II. Hardware Setup

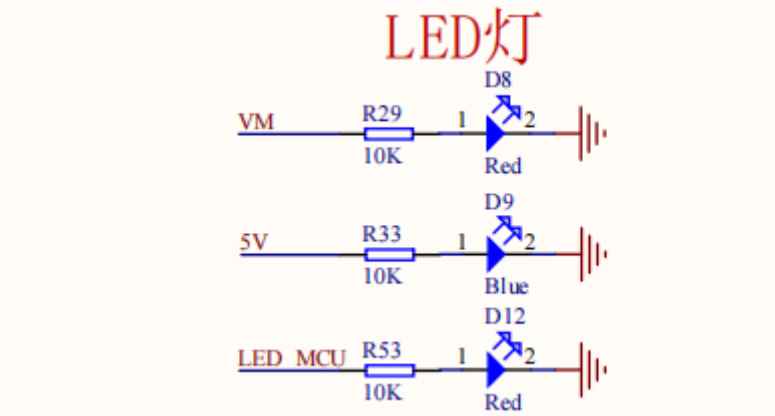
This course requires no additional hardware; the onboard LEDs on the MSPMOG3507 motherboard are sufficient.

We have set up a programmable MCU indicator light (red) on the development board.

MSPM0G3507 Main Control Diagram



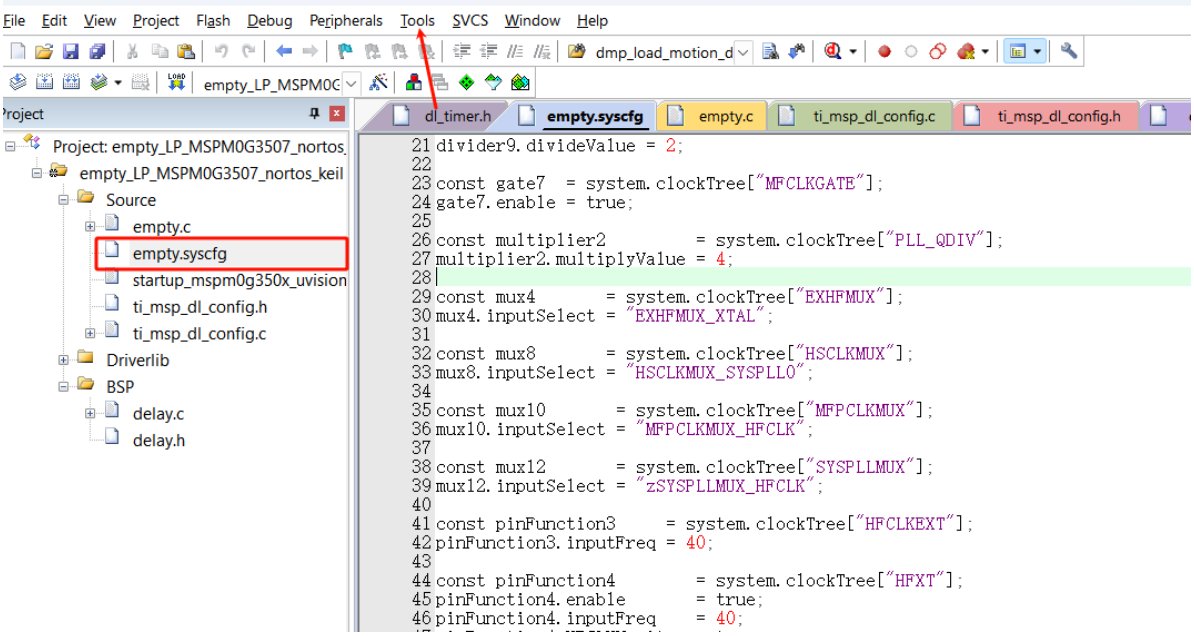
LED Part Circuit Diagram

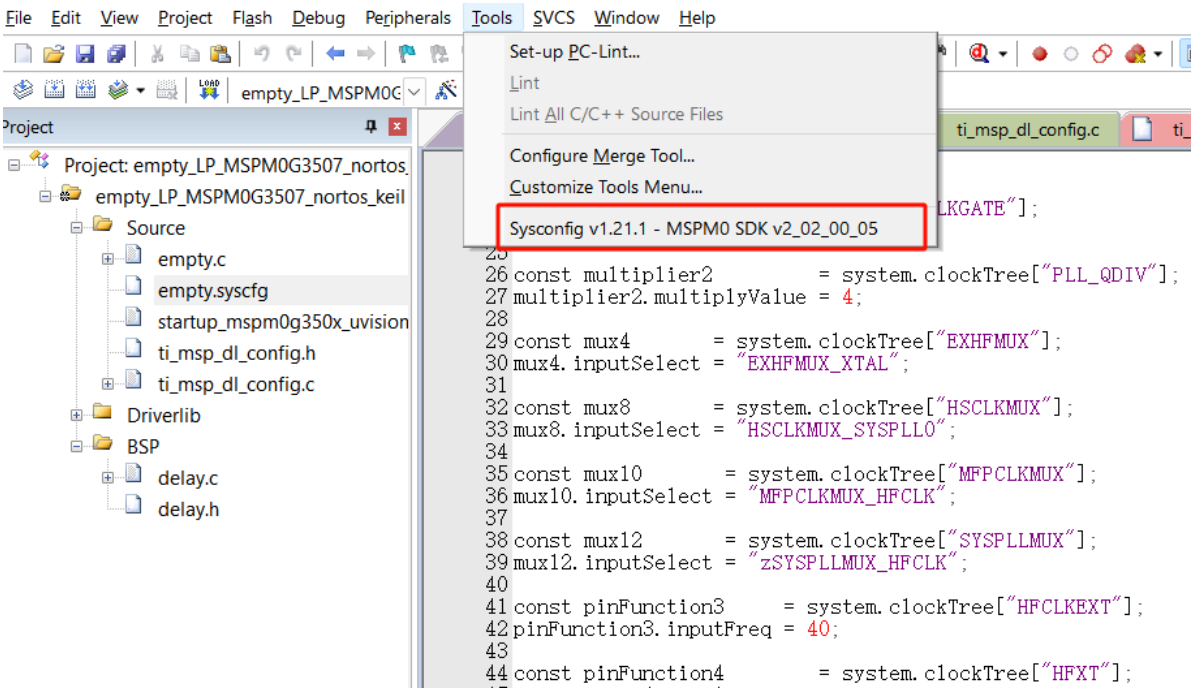


III. Experiment Steps

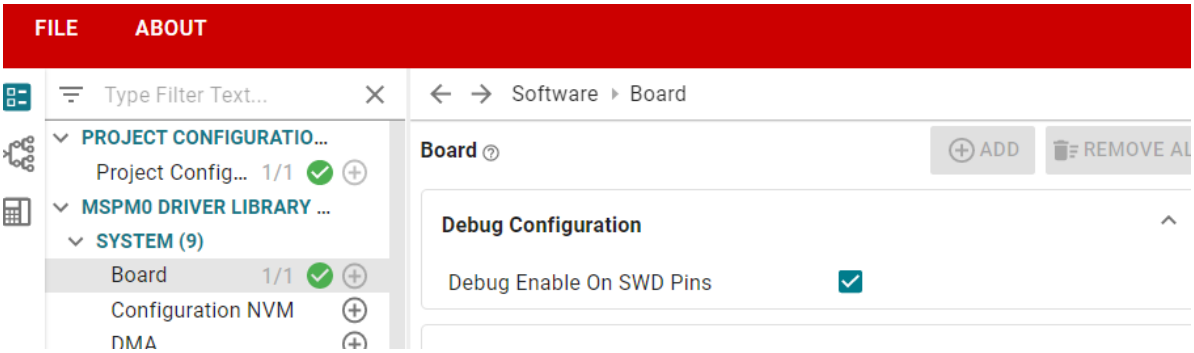
1. Open SYSCONFIG Configuration Tool

Before starting this tutorial, ensure that the SDK environment is successfully installed and the sysconfig tool configuration is also successful. For details, see [3. Development Environment Setup and Usage]. In KEIL, open our project configured with 80MHz crystal, or you can use the empty project from the SDK. For details, see [3. Development Environment Setup and Usage] -> [5. Project Migration]. After selecting and opening, in the KEIL interface, open the empty.syscfg file. **With the empty.syscfg file open**, select Open SYSCONFIG GUI interface from the Tools menu.

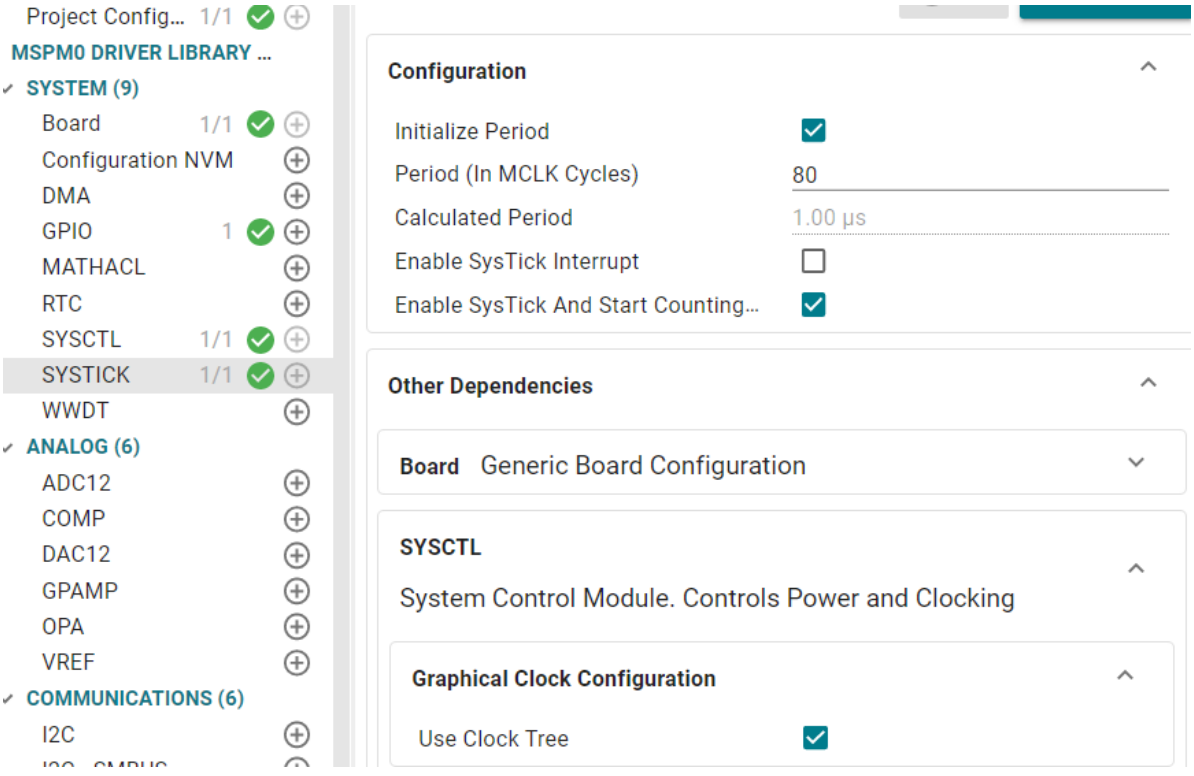




We first enable SWD debugging so that we can use debugging tools (DAP-Link, Jlink) to burn and debug the project.



We need to enable the SYSTICK clock for delay, set it to 1us here. Whether to enable SYSTICK interrupt depends on needs, it is disabled by default here (the period can be adjusted according to the clock tree configuration).



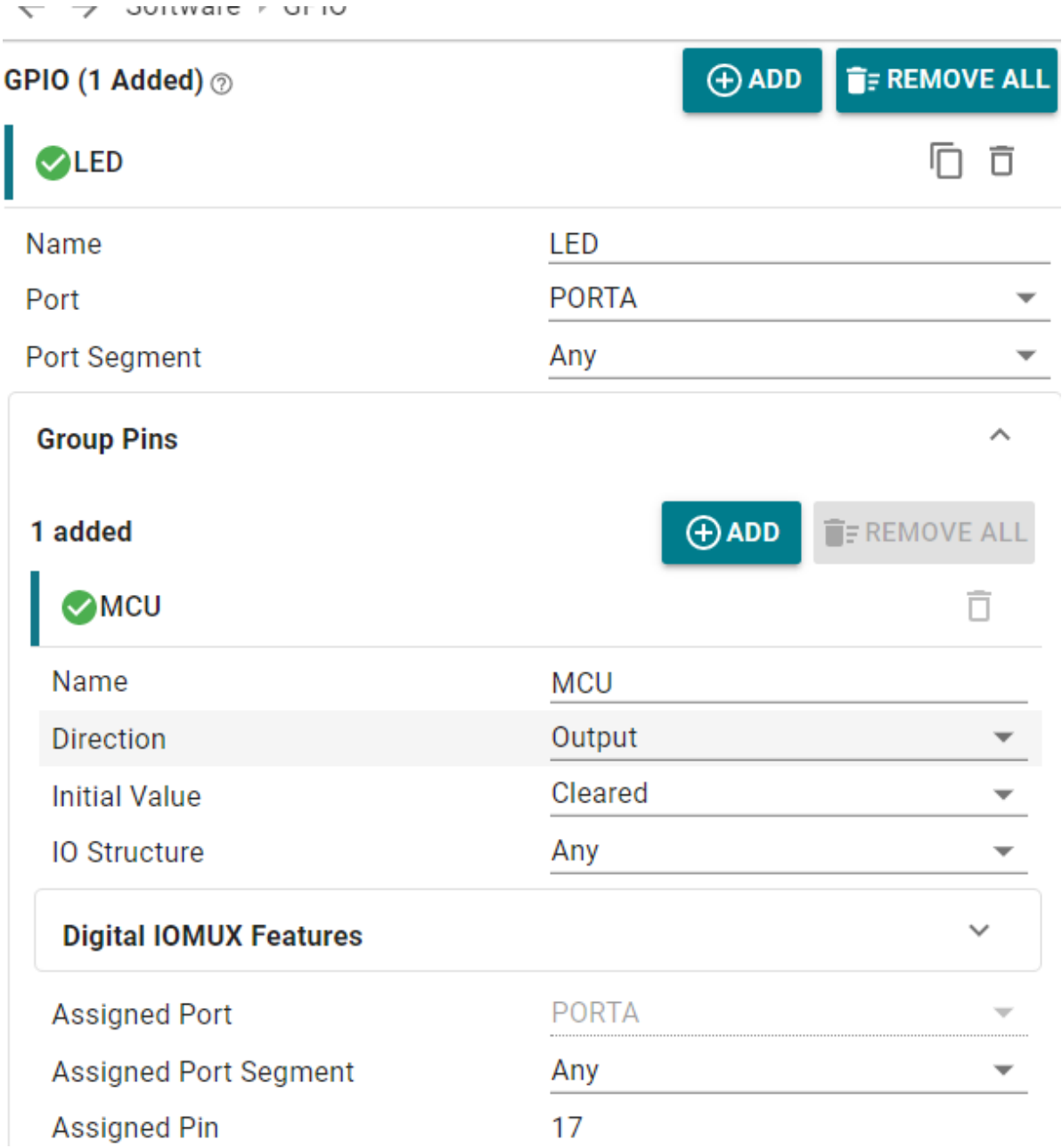
2. Parameter Settings

2.1 Set Port Parameters

In SYSCONFIG, select MCU peripherals on the left, find GPIO option and click to enter. In GPIO, click ADD to add a GPIO group, the name can be customized, here it is LED.

Set GPIO parameters, name this pin as MCU

From the LED part circuit diagram, we can see that LED_MCU is located on port PA17, so set Port to PORTA.



Parameter Description:

Name: Custom name for the GPIO instance. By default, the name starts with numeric suffix "0"; we can customize the name to reflect the module purpose (for example, naming GPIO as "MCU", so we know this pin is specifically for controlling LED_MCU).

Initial Value: Configure the initial level of GPIO, can only be set when configured as output mode. Here it is set to Cleared, default low level. High level is (Set)

Direction: Configure GPIO input/output, here it is output Output, Input is optional

Port: The port where the GPIO instance is located. LED is connected to GPIOA pin, can only select PORTA.

Port Segment: Set port pull-up/pull-down resistor. Note that this is port pull-up/pull-down, setting the entire GPIOA port.

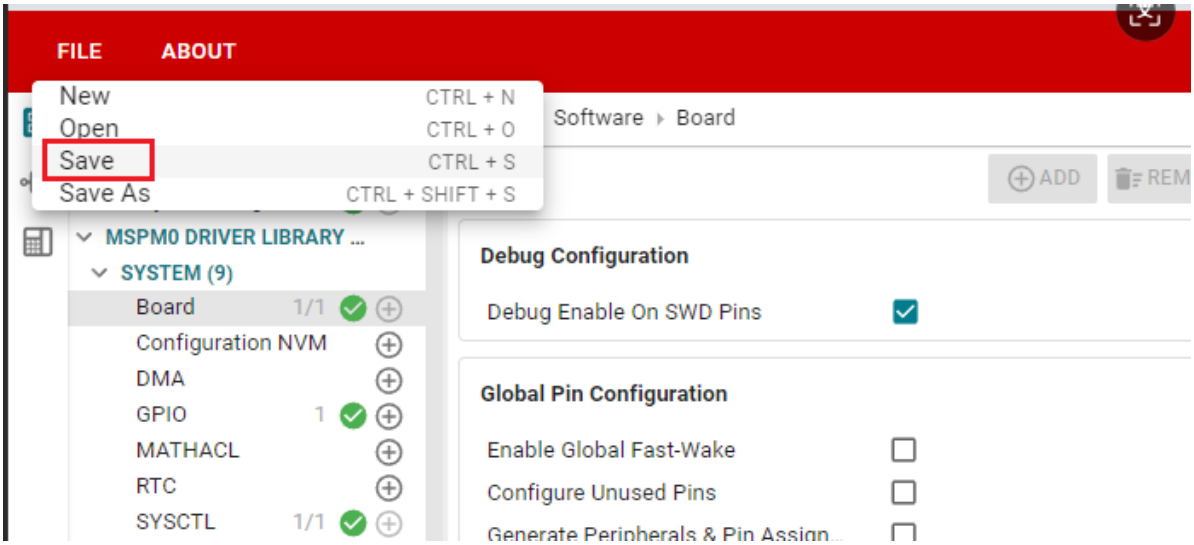
IO Structure: Set IO structure. There are multiple options, default (Any), standard (Standard), wake-up (Standard with Wake), high-speed (High-Speed) and 5V tolerant (5V Tolerant Open Drain) structures.

Internal Resistor: Set pin pull-up/pull-down resistor. There are three options, no setting, set pull-up and set pull-down resistor. Here according to the LED connection, pull-down resistor is selected.

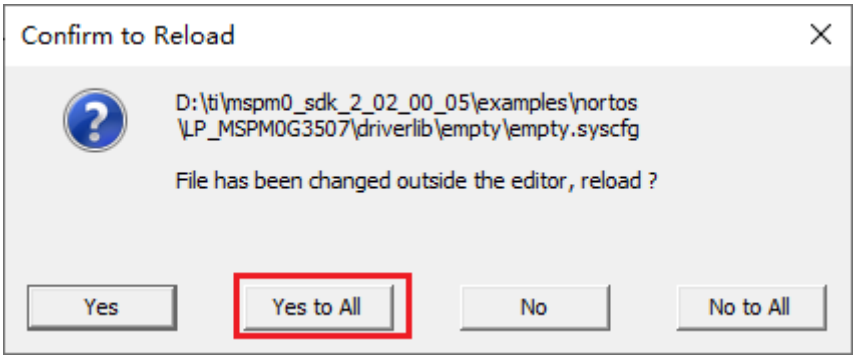
Assigned Pin: Set pin number. To control which pin, fill in the corresponding pin number, for example, the LED on the development board is connected to GPIOA17, then fill in 17.

2.3 Save and Update Configuration File

Click SAVE to save the configuration in SYSCONFIG, then close SYSCONFIG and return to keil.

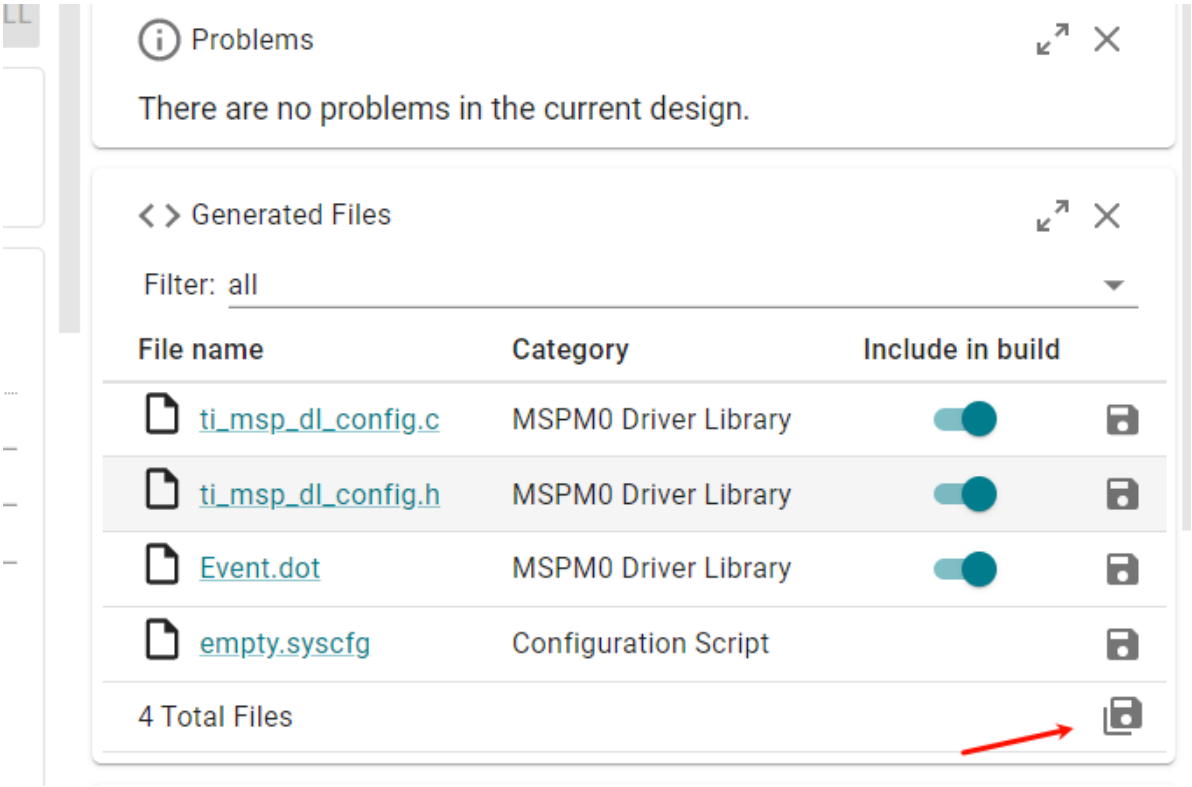


Select `Yes to All` in the pop-up window

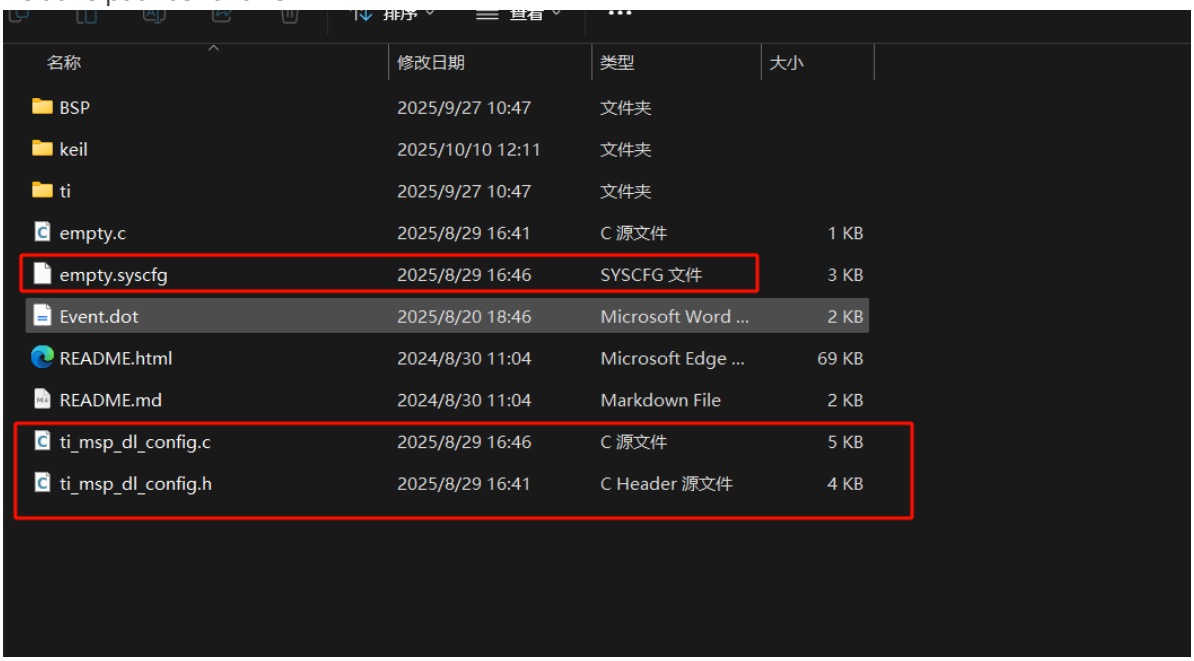


After that, we can see that the empty.syscfg file has been updated with our set parameters.

We also need to confirm whether ti_msp_dl_config.c and ti_msp_dl_config.h files are updated. Directly compile, compilation will automatically update to keil. If not updated, we manually select the project path to save



Relative path as follows



3. Write Program

In the empty.c file, write the following code

```
#include "ti_msp_dl_config.h"
#include "delay.h"

int main(void)
{
    SYSCFG_DL_init();


    while (1)
    {
        DL_GPIO_clearPins(LED_PORT,LED_MCU_PIN);//output low level o
```

```
        delay_ms(1000);//delay about 1S
        //output high level
                DL_GPIO_setPins(LED_PORT,LED_MCU_PIN);
        delay_ms(1000);//delay about 1S
    }

}
```

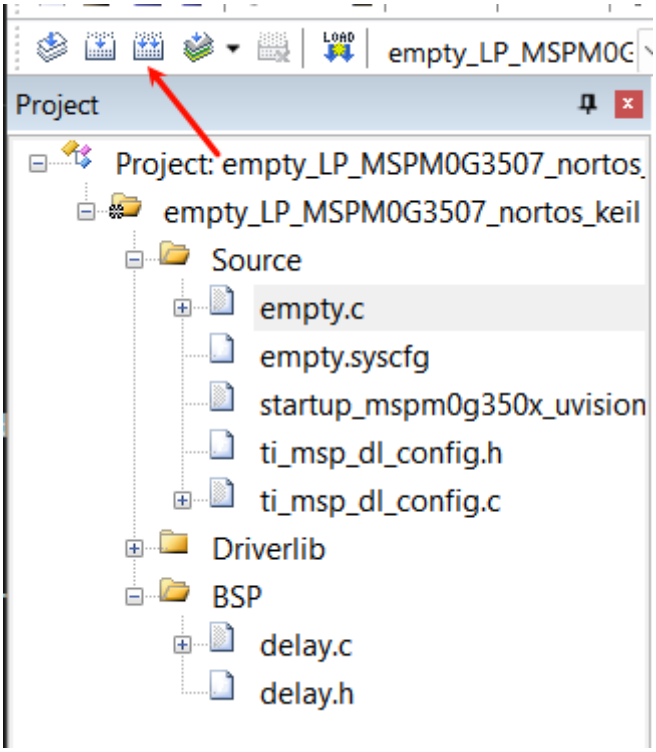
delay.c

```
#include "ti_msp_dl_config.h"
//Using method 0, 1 to implement delay requires disabling systick interrupt
//The following implements delay using three methods: 0: using a loop of no
operations to implement delay 1: using polling to read systick count to implement
delay  2: using systick interrupt to implement delay (default) If using the first
two methods, disable systick interrupt


//You can also use the macro DL_Common_delayCycles provided by TI, see dl_core.h
for details
//void delay_cycles(uint32_t cycles) {
//    while (cycles--) {
//        __NOP(); // Execute no operation (No Operation)
//    }
//}
#define DELAY_SELECT 1
volatile unsigned int delay_times = 0;


//Custom delay (not precise) Custom delay (not precise)
#if DELAY_SELECT==0
void delay_ms(unsigned int ms)
{
    unsigned int i, j;
    // The number of nested loops below is roughly calculated based on the main
control frequency and the instruction cycle generated by the compiler,
    // and needs to be adjusted through actual testing to achieve the required
delay.

    for (i = 0; i < ms; i++)
    {
        for (j = 0; j < 8000; j++)
        {
            // Only execute a simple operation that can predict its execution
time

            __asm__("nop"); // "nop" represents "no operation", which consumes
one or several clock cycles in most architectures, not fixed
        }
    }
}

#elif  DELAY_SELECT==1
//Implement delay by polling systick count
void delay_us(unsigned long __us)
{
```

```
    uint32_t ticks;
    uint32_t told, tnow, tcnt = 0;

    ticks = __us * (80000000 / 1000000);  // 80MHz clock
    told = SysTick->VAL;

    while (1)
    {
        tnow = SysTick->VAL;
        if (tnow != told)
        {
            if (tnow < told)
                tcnt += told - tnow;
            else
                tcnt += (SysTick->LOAD + 1) - tnow + told;

            told = tnow;
            if (tcnt >= ticks)
                break;
        }
    }
}
//Precise ms delay implemented with tick timer
void delay_ms(unsigned long ms)
{
    while (ms--)
    {
        delay_us(1000);  // delay 1ms each time
    }
}

#else
//Implement delay using systick interrupt, need to enable systick interrupt in
sysconfig
void delay_ms(unsigned int ms)
{
    delay_times=ms;
    while(delay_times!=0);
}


void SysTick_Handler(void)
{
        if( delay_times != 0 )
        {
                delay_times--;
        }
}
#endif
```

4. Compile

Click the Rebuild icon. The following prompt indicates that compilation is complete and there are no errors.




```
Generating Code (empty.syscfg)...
Unchanged D:\ti\mspm0_sdk_2_02_00_05\examples\nortos\LP_MSPM0G3507\driverlib\empty\ti_msp_dl_config.c...
Unchanged D:\ti\mspm0_sdk_2_02_00_05\examples\nortos\LP_MSPM0G3507\driverlib\empty\ti_msp_dl_config.h...
Unchanged D:\ti\mspm0_sdk_2_02_00_05\examples\nortos\LP_MSPM0G3507\driverlib\empty\Event.dot...
assembling startup_mspm0g350x_uvision.s...
compiling empty.c...
compiling ti_msp_dl_config.c...
linking...
Program Size: Code=544 RO-data=208 RW-data=0 ZI-data=352
FromELF: creating hex file...
".\Objects\empty_LP_MSPM0G3507_nortos_keil.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:06
```

IV. Program Analysis

- dl_gpio.h

```
/**
 *  @brief      Set a group of GPIO pins
 *
 *  @param[in]  gpio  Pointer to the register overlay for the peripheral
 *  @param[in]  pins  Pins to set high. Bitwise OR of @ref DL_GPIO_PIN.
 */
__STATIC_INLINE void DL_GPIO_setPins(GPIO_Regs* gpio, uint32_t pins)
{
    gpio->DOUTSET31_0 = pins;
}

/**
 *  @brief      Clear a group of GPIO pins
 *
 *  @param[in]  gpio  Pointer to the register overlay for the peripheral
 *  @param[in]  pins  Pins to clear. Bitwise OR of @ref DL_GPIO_PIN.
 */
__STATIC_INLINE void DL_GPIO_clearPins(GPIO_Regs* gpio, uint32_t pins)
```

```
{
    gpio->DOUTCLR31_0 = pins;
}

/**
 *  @brief      Toggle a group of GPIO pins
 *
 *  @param[in]  gpio  Pointer to the register overlay for the peripheral
 *  @param[in]  pins  Pins to toggle. Bitwise OR of @ref DL_GPIO_PIN.
 */
__STATIC_INLINE void DL_GPIO_togglePins(GPIO_Regs* gpio, uint32_t pins)
{
    gpio->DOUTTGL31_0 = pins;
}
```

__STATIC_INLINE void DL_GPIO_setPins(GPIO_Regs* gpio, uint32_t pins): This function controls the pin to output high level.

__STATIC_INLINE void DL_GPIO_clearPins(GPIO_Regs* gpio, uint32_t pins): This function controls the pin to output low level.

__STATIC_INLINE void DL_GPIO_togglePins(GPIO_Regs* gpio, uint32_t pins): This function controls the pin level toggle, if it was high level it becomes low level, if it was low level it becomes high level.

Set PA17 to output low level, delay about 1s then output high level, then delay about 1s. Repeat this process.

V. Experiment Phenomenon

After the program download is complete, you can see the red LED on the MSPM0 development board is lit and flashes about every 1s.