External Interrupt Experiment

1. Learning Objectives

1. Learn about the external interrupts of the MSPM0G3507 motherboard.
2. Use the onboard buttons to control the on/off of the onboard LED light.

External Interrupt

An interrupt is when the CPU is executing a program and there are other events that need to occupy the CPU. After receiving the interrupt request, the CPU pauses the execution of the original program and executes the interrupt function. After the interrupt function is executed, it returns to the original program and continues to execute.

An external interrupt is an interrupt triggered by an external hardware event (usually a signal from outside the microcontroller). This interrupt can interrupt the current program execution flow of the microcontroller, thereby quickly responding to external hardware events.

The triggering of external interrupts is usually associated with specific pins (GPIO pins), such as buttons, sensor signals, timer signals, etc. These pins are usually configured to detect specific signal changes (such as rising edges, falling edges, or level changes) to trigger interrupts.

Advantages and functions of external interrupts

Advantages:

- Fast response: External interrupts can respond immediately when a trigger event occurs without polling the signal.
- Save resources: There is no need to continuously detect the signal status in the main loop, saving CPU resources.
- Improve real-time performance: The interrupt mechanism can ensure timely processing of key events, thereby improving the real-time performance of the system.
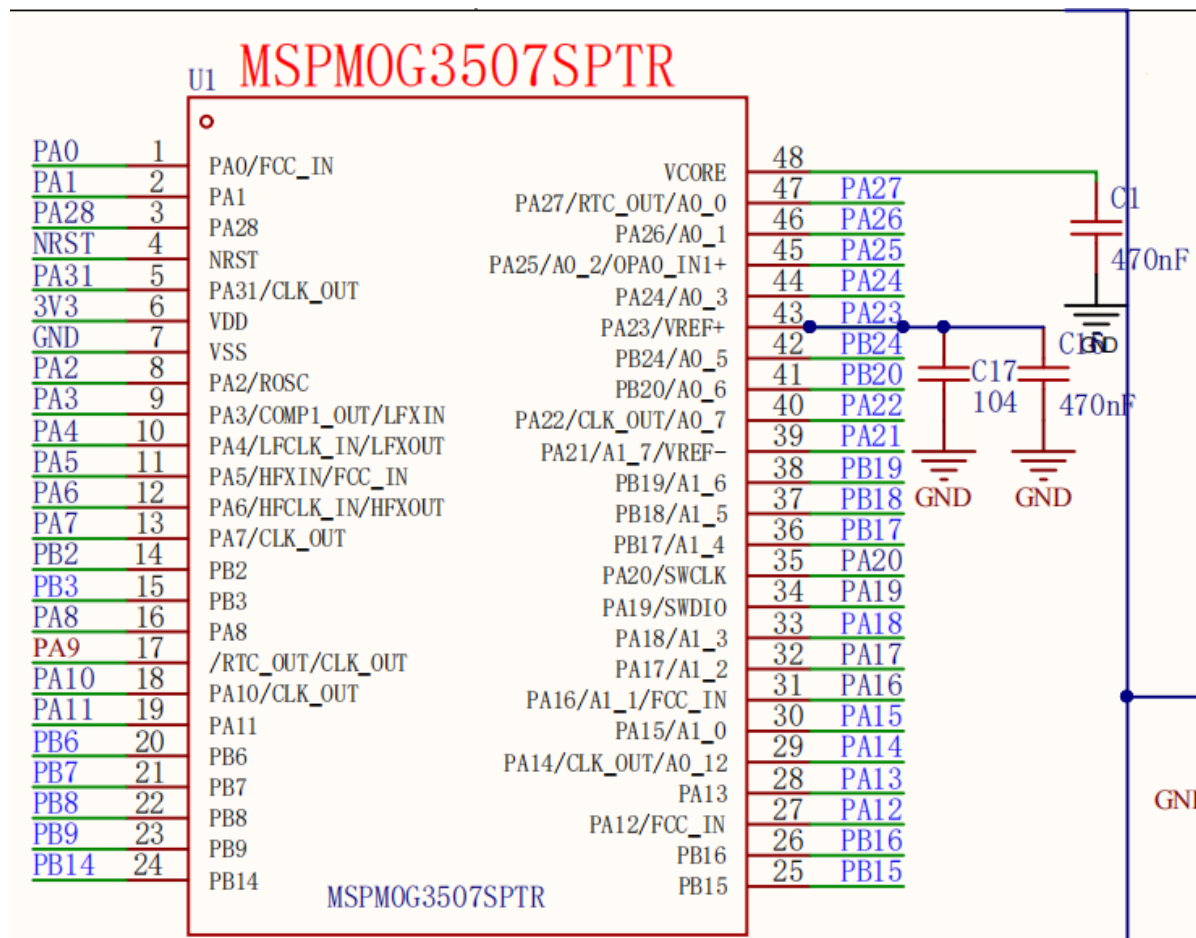
Function:

- Quick response to external events: When a hardware event occurs, the execution of the current program can be interrupted immediately and the event can be processed quickly without polling to check the signal status. It avoids the program's polling of events, greatly saves CPU resources, and improves the system's response speed to external events.

- Improve system real-time performance: In scenarios that require real-time response, external interrupts can ensure that key events are processed in a timely manner without waiting for the system main program to complete the current task, thereby improving the system's real-time performance.
- Reduce the burden on the main program: External interrupts can separate tasks related to external events from the main program, avoiding polling to check the signal status in the main program, improving the execution efficiency of the main program, and allowing the main program to focus on core tasks.
- The core of event-driven system: it can realize efficient event-driven mechanism and reduce system complexity.
- Realize hardware interaction and control: external interrupts enable the microcontroller to process and respond to external hardware signals in real time to ensure the normal operation of the system.
- Data acquisition and processing: in many real-time data acquisition systems, external interrupts are used to trigger data acquisition or processing.
- Trigger mechanism for low-power scenarios: through external interrupts, efficient low-power design can be realized, so that the system remains in sleep state when not working and only runs when necessary.
- Ensure system security: handle security-related events in a timely manner to ensure the reliability and security of the system.
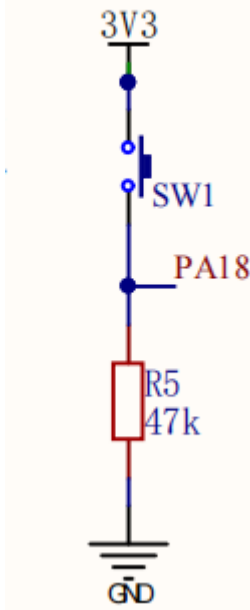
2. Hardware construction

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

MSPM0G3507 main control diagram


Button schematic diagram

Function keys

3. Experimental steps

In this lesson, the PA18 button will be used to trigger the LED1 of PB2 to turn on and off, and the CPU interrupt method will be used in the software. For the configuration of the PB2 pin of LED1, please refer to the **Light up the LED** course.

## 1. Open the SYSCONFIG configuration tool

Open the blank project empty in the SDK in KEIL.
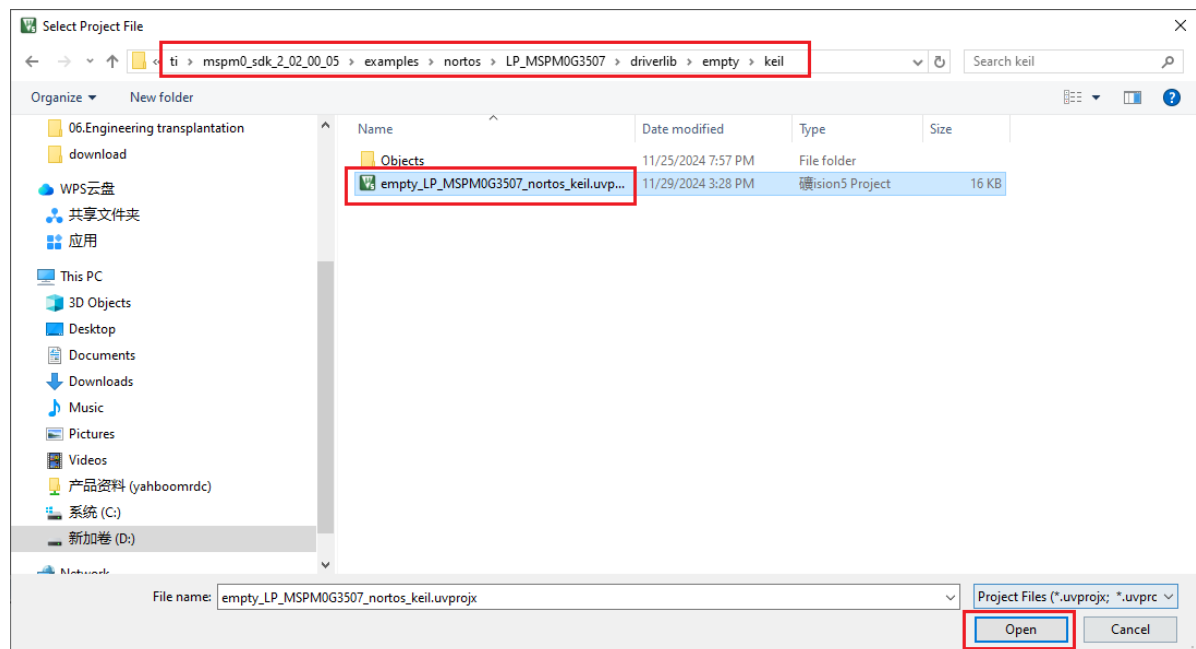
Open after selecting, open the empty.syscfg file in the keil interface, **when the empty.syscfg file is open**, then select to open the SYSCONFIG GUI interface in the Tools column.
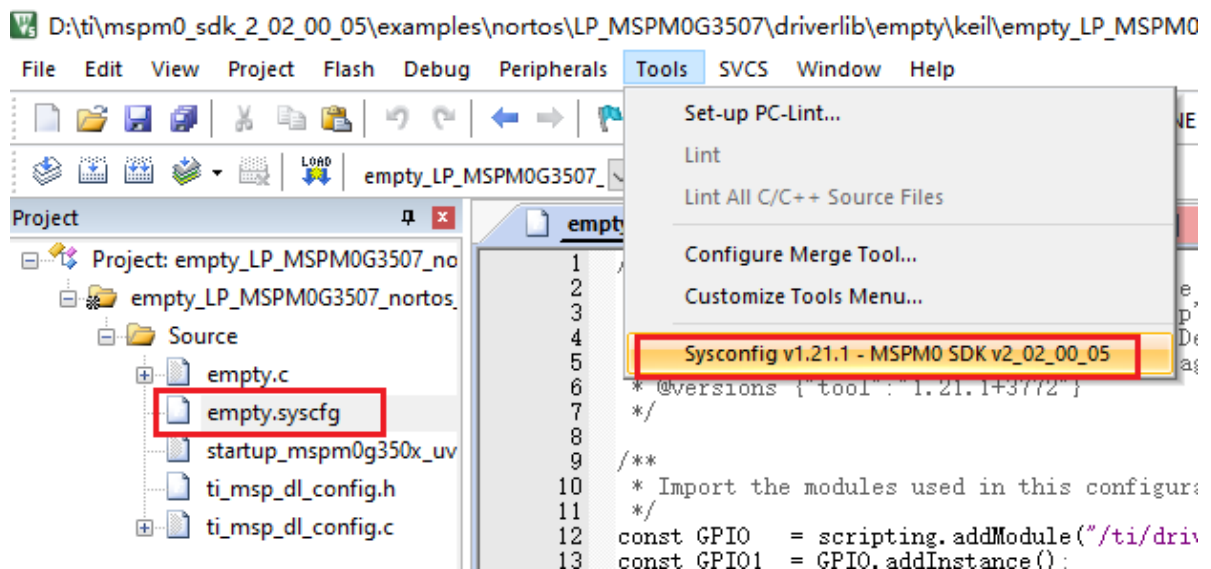
In SYSCONFIG, select MCU peripherals on the left, find the GPIO option and click to enter. Click ADD in GPIO to add a group of GPIO. This GPIO_GRP_0 is the newly added one.
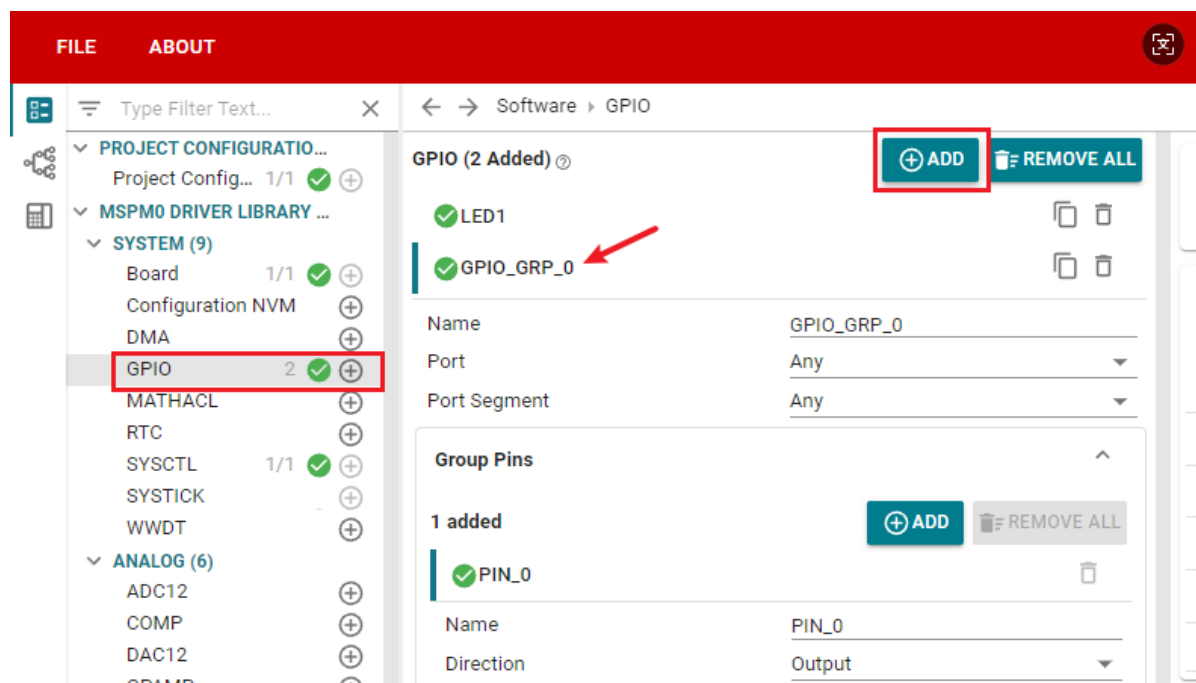
2. Configure pin mode

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

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

KEY

Name	KEY
Port	PORTA
Port Segment	Any

Group Pins

1 added [ADD] [REMOVE ALL]

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)

Configure pin external interrupts. All GPIOA ports on the chip can set external interrupt functions.

Enable Interrupts enables interrupts, **Interrupt Priority** sets the interrupt priority to default. When multiple interrupts are triggered at the same time, they are processed according to the chip's default priority rules.

Trigger Polarity The interrupt is triggered by a rising edge. When the button is pressed, the level of the GPIO pin changes from low level (0) to high level (1). This process is called a rising edge. This means that the external interrupt will be triggered when the pin state changes from low to high.

Event Publishing channel The event publishing channel is not enabled. The event publishing channel is a mechanism that publishes interrupt events to a specified channel. Other peripherals can respond by subscribing to events on the channel.
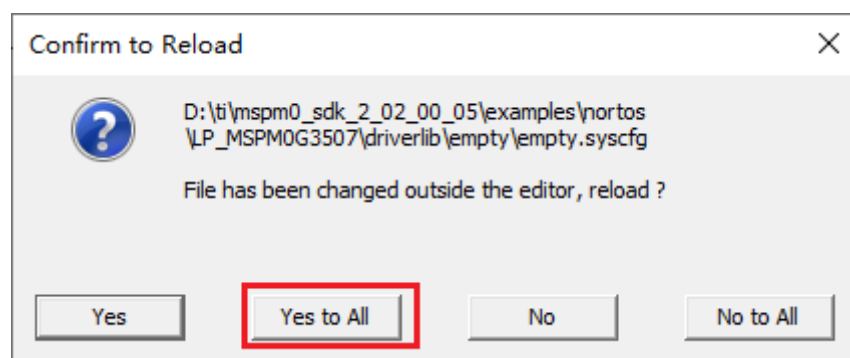
Interrupts/Events

Enable Interrupts	☑
Interrupt Priority	Default
Trigger Polarity	Trigger on Rising Edge
Event Publishing Channel	Disabled (0)

Click SAVE to save the configuration in SYSCONFIG, then turn off SYSCONFIG and return to keil.
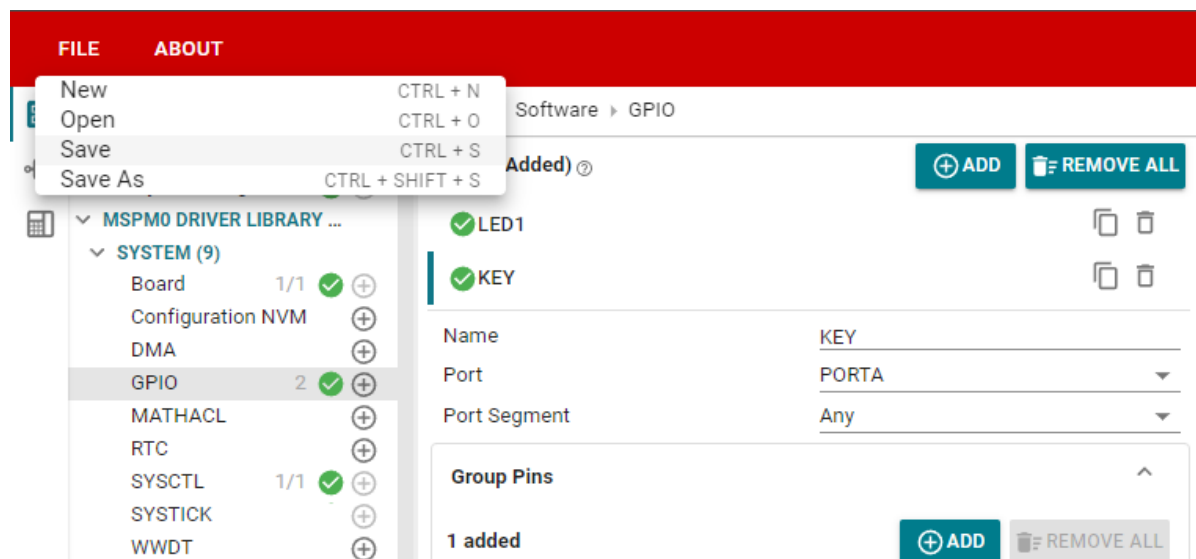
Select `Yes to All` in the pop-up window

Similarly, we also need to confirm whether the ti_msp_dl_config.c and ti_msp_dl_config.h files are updated. Compile directly, and the compilation will automatically update to keil. If there is no update, we can also copy the file content in SYSCONFIG.

3. Write the program

After configuring the pin interrupt, we also need to manually write the interrupt service function of the external interrupt. Because we have enabled the pin falling edge interrupt, when our button is pressed and released, the falling edge will trigger an interrupt, and the interrupt service function will be executed after the trigger. The name of the interrupt service function corresponding to each interrupt is usually fixed and cannot be modified at will, otherwise the interrupt service function will not be entered correctly. The specific name of the interrupt service function can be found in the project's `startup_mspm0g350x_uvision.s` file, which lists the interrupt service function names corresponding to each interrupt source.
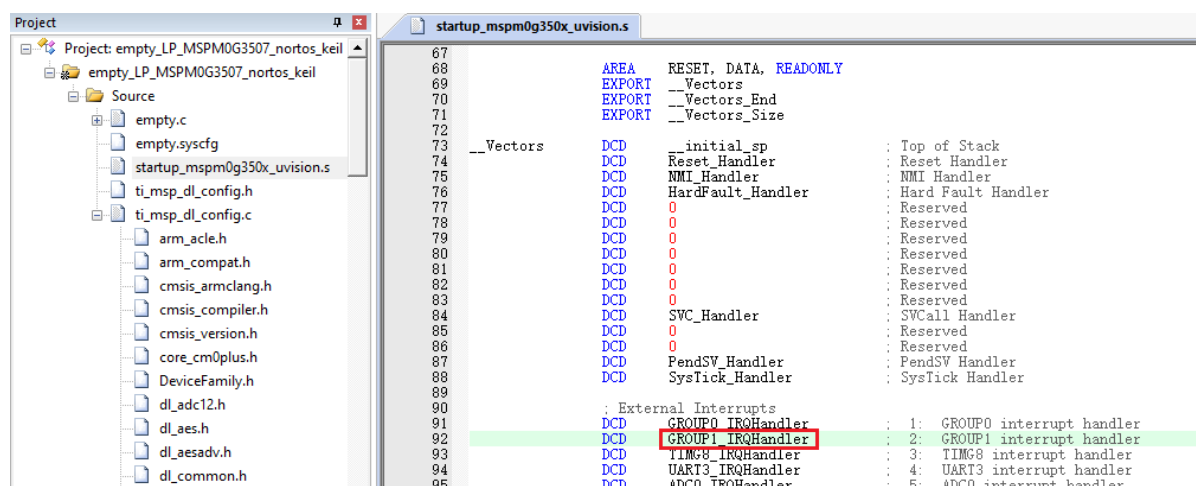
According to the user manual of the MSPM0G series, it can be known that after the GPIO interrupt is triggered, the interrupt is released to the bus through the **GRP1** line. After the bus recognizes it, it enters the interrupt service function to execute the content.

So in the empty.c file, write the following code

```c
#include "ti_msp_dl_config.h"

int main(void)
{
    SYSCFG_DL_init();
    NVIC_EnableIRQ(KEY_INT_IRQN);//开启按键引脚的GPIOA端口中断 Enable GPIOA port
interrupt for button pin

    while (1)
    {

    }
}


void GROUP1_IRQHandler(void)//Group1的中断服务函数 Interrupt service function of
Group1
{
    //读取Group1的中断寄存器并清除中断标志位
    // Read the interrupt register of Group1 and clear the interrupt flag
    switch( DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP_1) )
    {
        //检查是否是KEY的GPIOA端口中断，注意是INT_IIDX，不是PIN_18_IIDX
        // Check if it is the GPIOA port interrupt of KEY, note that it is
INT_IIDX, not PIN_18_IIDX
        case KEY_INT_IIDX:
            //如果按键按下变为高电平 If the button is pressed, it becomes high level
            if( DL_GPIO_readPins(KEY_PORT, KEY_PIN_18_PIN) > 0 )
            {
                //设置LED引脚状态翻转 Set LED pin state toggle
                DL_GPIO_togglePins(LED1_PORT, LED1_PIN_2_PIN);
            }
        break;
    }
}
```
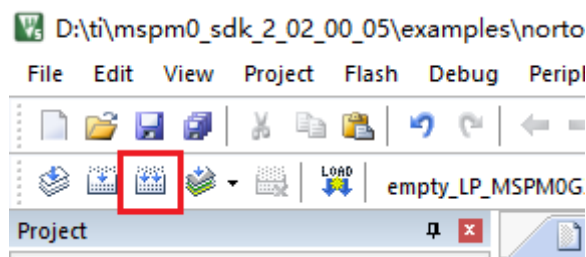
4. Compile

Click the Rebuild icon. The following prompt appears, indicating that the 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
<
```

4. Program Analysis

- dl_gpio.h

```
/**
 *  @brief      Get highest priority interrupt pending in the selected interrupt
 *              group
 *
 *  @param[in]  group  Group of interrupts to check
 *
 *  @return     The highest priority pending interrupt for the group requested.
 *              One of @ref DL_INTERRUPT_GROUP_IIDX
 */
__STATIC_INLINE uint32_t DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP group)
{
    return (CPUSS->INT_GROUP[group].IIDX);
}
```

uint32_t DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP group) is used to get the index of the highest priority pending interrupt in the specified interrupt group.

group: An enumeration type `DL_INTERRUPT_GROUP`, used to specify the interrupt group to check.

- empty.c

```
int main(void)
{
    SYSCFG_DL_init();
    NVIC_EnableIRQ(KEY_INT_IRQN);//开启按键引脚的GPIOA端口中断 Enable GPIOA port
interrupt for button pin

    while (1)
    {

    }
}


void GROUP1_IRQHandler(void)//Group1的中断服务函数 Interrupt service function of
Group1
{
    //读取Group1的中断寄存器并清除中断标志位
    // Read the interrupt register of Group1 and clear the interrupt flag
    switch( DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP_1) )
    {
        //检查是否是KEY的GPIOA端口中断，注意是INT_IIDX，不是PIN_18_IIDX
        // Check if it is the GPIOA port interrupt of KEY, note that it is
INT_IIDX, not PIN_18_IIDX
```

```
        case KEY_INT_IIDX:
            //如果按键按下变为高电平 If the button is pressed, it becomes high level
            if( DL_GPIO_readPins(KEY_PORT, KEY_PIN_18_PIN) > 0 )
            {
                //设置LED引脚状态翻转 Set LED pin state toggle
                DL_GPIO_togglePins(LED1_PORT, LED1_PIN_2_PIN);
            }
        break;
    }
}
```

This code implements a basic interrupt service function, which mainly controls the state of an LED by triggering an interrupt with a key.

After the `main()` function is initialized, the GPIOA port interrupt of the key is turned on. The `GROUP1_IRQHandler()` interrupt service function obtains the highest priority suspend flag in Group1, checks whether a key interrupt has occurred, and then calls `DL_GPIO_readPins(KEY_PORT, KEY_PIN_18_PIN)` to read the level change of the key pin. If it becomes a high level, the state of the LED pin is flipped.

5. Experimental phenomenon

After the program is downloaded, the state of the LED light can be controlled by the key. Press once to turn on the LED light, and press again to turn off the LED light.