Serial communication

1. Learning objectives

1. Learn the basic use of the serial port of the MSPM0G3507 motherboard.
2. Understand the basic knowledge of serial communication.

Serial communication

Serial communication refers to a communication method in which data is transmitted bit by bit between peripherals and computers through data signal lines, ground lines, etc., and belongs to the serial communication method.

The key parameters of serial communication include baud rate, data bit, stop bit and parity bit. In order to ensure normal communication, these parameters must be consistent between the two communicating parties.

Serial working mode

1. **Simplex mode**

- **Features**: Data can only be transmitted in one direction, the sender is responsible for sending, the receiver is responsible for receiving, and there is no reverse communication.
- **Application**: Applicable to scenarios where information only needs to be transmitted in one direction, such as sensor data output.

2. **Full-Duplex Mode**

- **Features**: Data can be transmitted in both directions at the same time, and sending and receiving do not interfere with each other.
- **Implementation**: Two data lines are required, one for sending (TX) and one for receiving (RX).
- **Application**: Commonly used in scenarios that require efficient and real-time communication, such as the interaction between computers and peripherals.

3. **Half-Duplex Mode**

- **Features**: Data is transmitted in both directions, but not at the same time, and must be sent and received alternately.

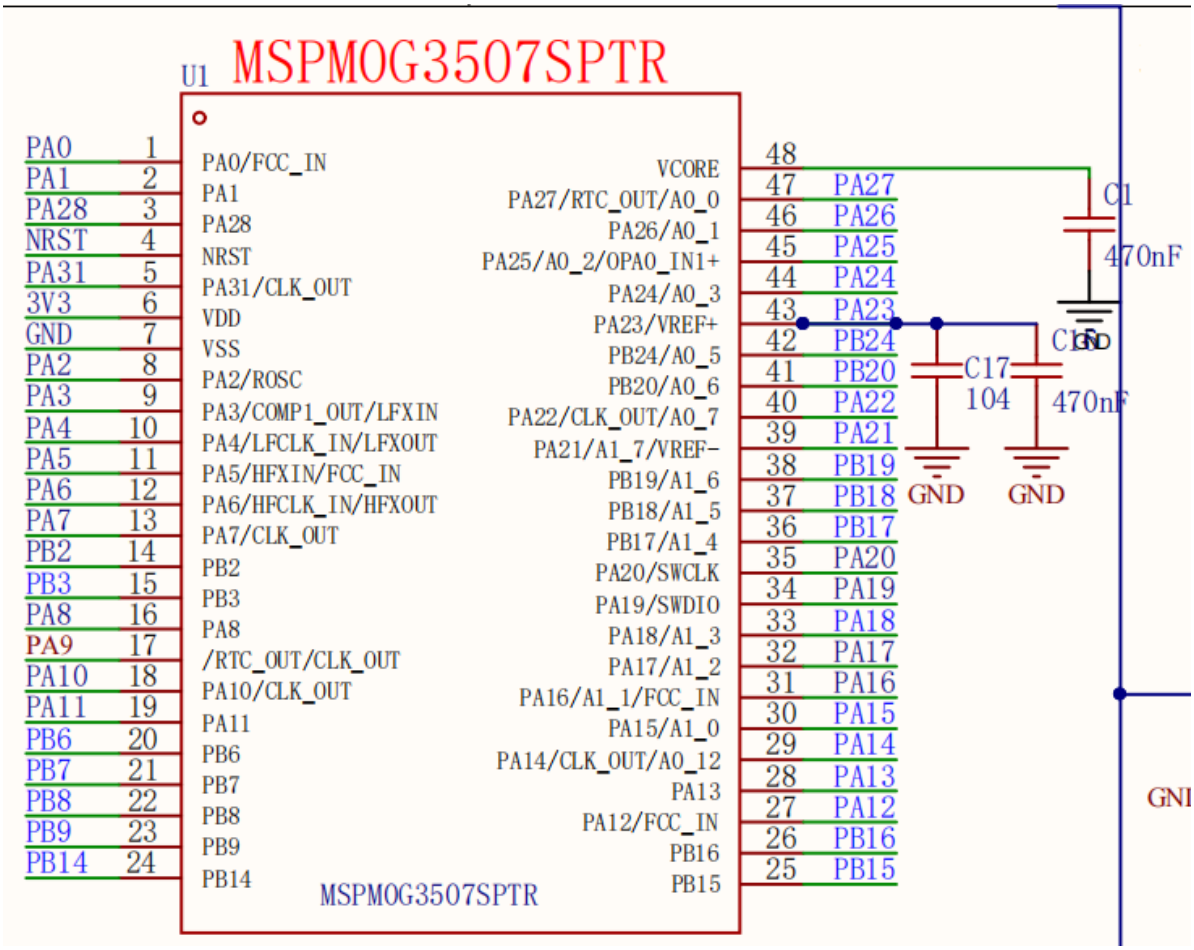
- **Implementation**: Sending and receiving share a communication line, and the direction is switched by timing.
- **Application**: Suitable for scenarios that do not require high real-time performance, such as walkie-talkie communication.

These three modes adapt to different communication needs and can be flexibly selected according to specific application scenarios.

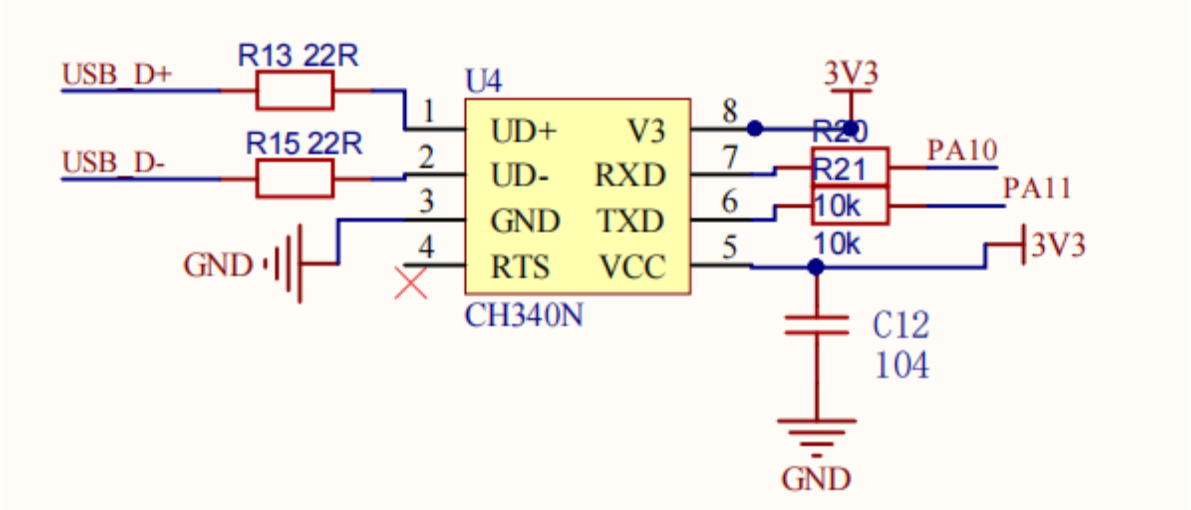
2. Hardware Construction

Since the development board comes with a CH340 serial port circuit, serial port communication can be directly realized through a USB cable without an external USB to TTL module.

MSPM0G3507 main control diagram



Type-C partial schematic diagram



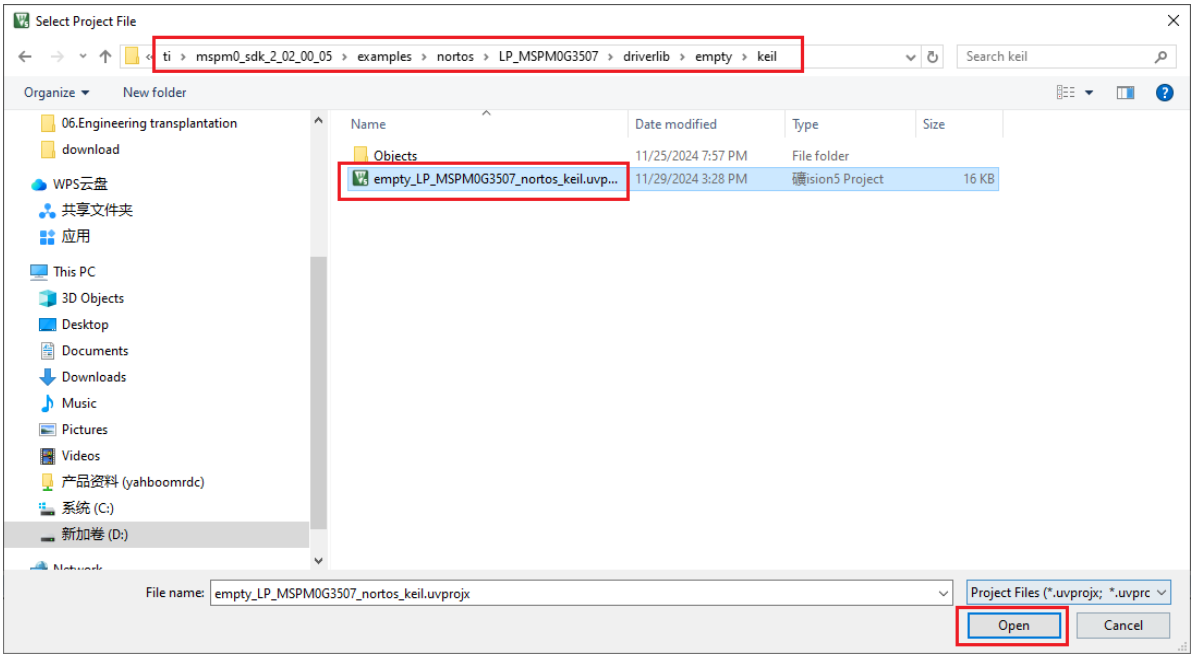
21	PA10		UART0_TX *[2]* / SPI0_POCI *[3]* / I2C0_SDA *[4]* / TIMA1_C0 *[5]* / TIMG12_C0 *[6]* / TIMA0_C2 *[7]* / I2C1_SDA *[8]* / CLK_OUT *[9]*	56	18	14	15
22	PA11		UART0_RX *[2]* / SPI0_SCK *[3]* / I2C0_SCL *[4]* / TIMA1_C1 *[5]* / COMP0_OUT *[6]*/ TIMA0_C2N *[7]* / I2C1_SCL *[8]*	57	19	15	16

3. Experimental steps

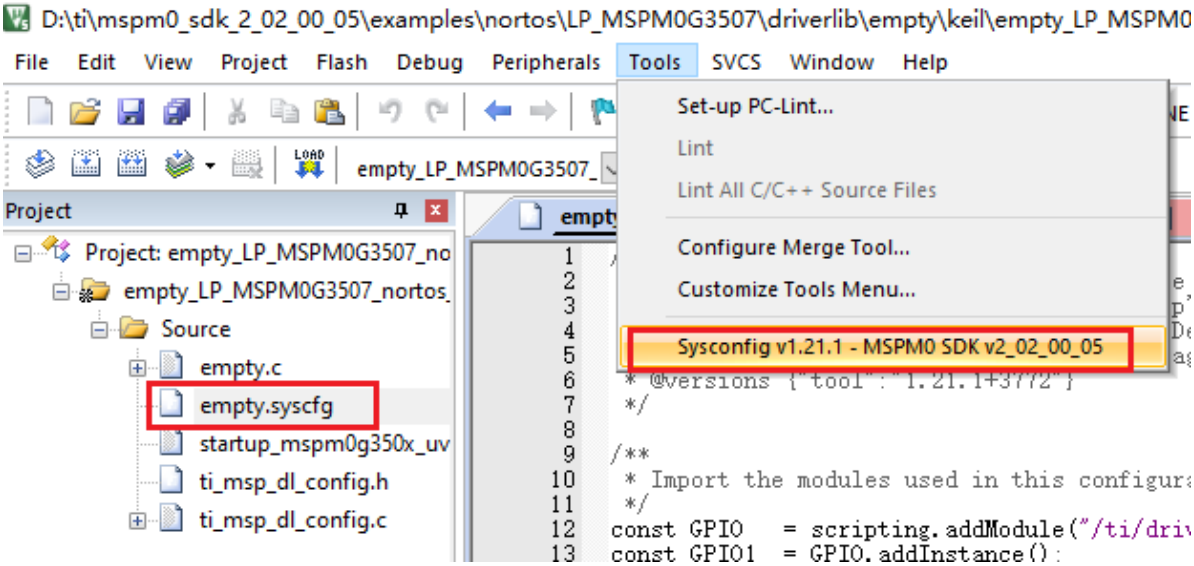
This lesson uses the UART0 peripheral on the PA10 and PA11 pins, and the CH340 USB-to-serial port chip on the development board to receive serial port data sent by the host computer, and then send the data to the host computer through the serial port sending function.

1. Open the SYSCONFIG configuration tool

Open the blank project empty in the SDK in KEIL.



Select and open, open the empty.syscfg file in the keil interface, **with the empty.syscfg file open**, then select to open the SYSCONFIG GUI interface in the Tools column.



2. Serial port clock configuration

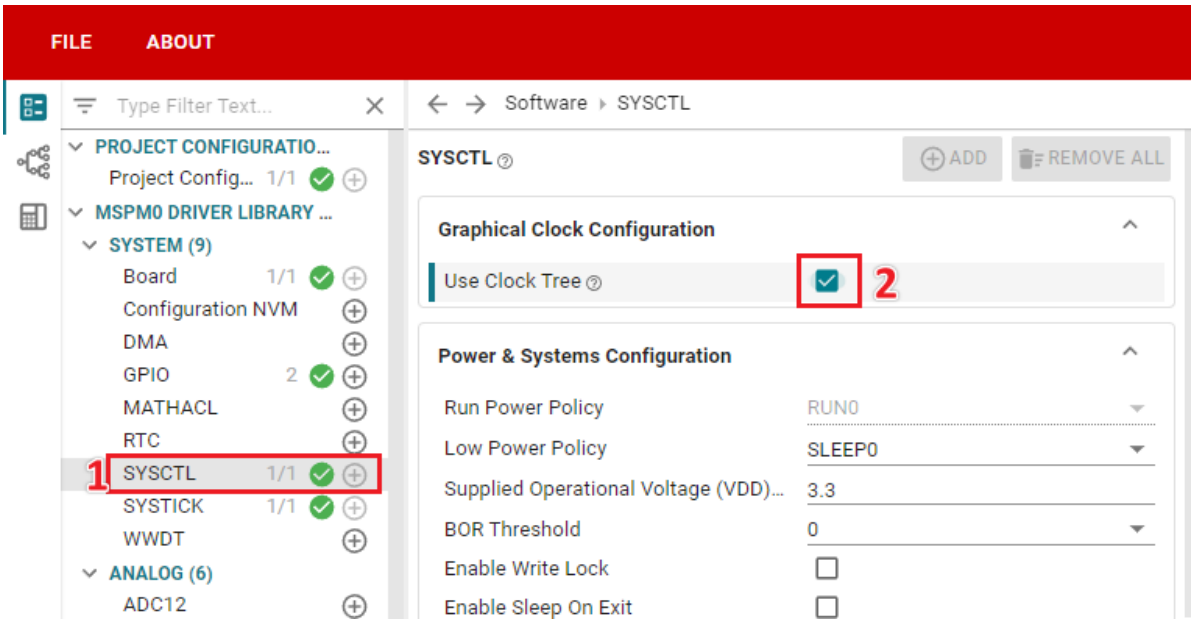
The serial port has three clock sources:

1.**BUSCLK**: CPU clock provided by the internal high-frequency oscillator, usually the chip is set to 32MHz when it leaves the factory.

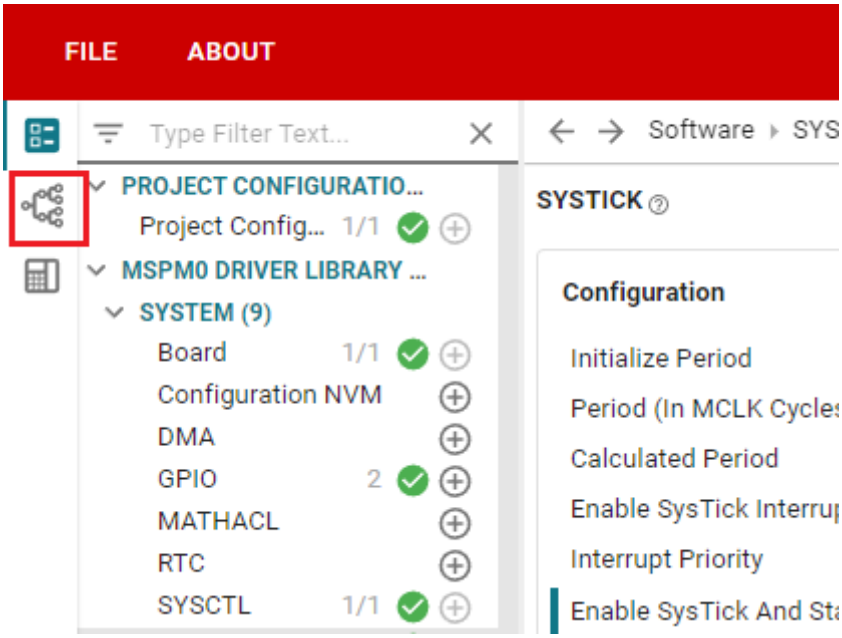
2.**MFCLK**: Only a fixed 4MHz clock can be used. If you want to enable it, you need to configure the SYSOSC_4M branch of the clock tree to enable it normally.

3. **LFCLK**: The clock (32KHz) is provided by the internal low-frequency oscillator. It is valid in run, sleep, stop and standby modes. Using this clock can achieve lower power consumption.

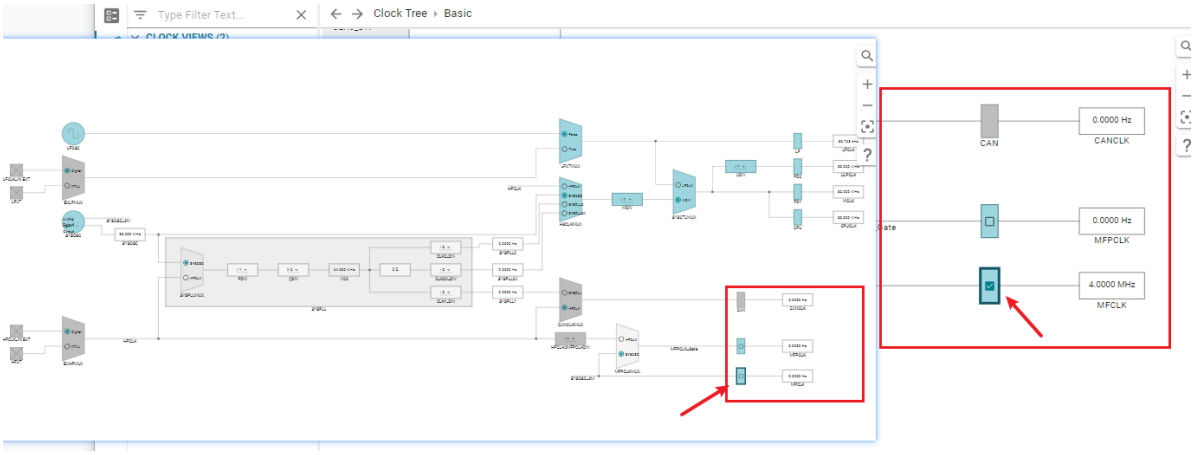
This case uses MFCLK as the clock source of the serial port. To enable MFCLK, you need to configure the clock tree in SYSCONFIG. Find the SYSCTL option in the tab on the left side of SYSCONFIG, find Use Clock Tree in the option page and check it to enable the configuration of the clock tree.



Click the second icon on the far left of the SYSCONFIG interface to open the clock tree.

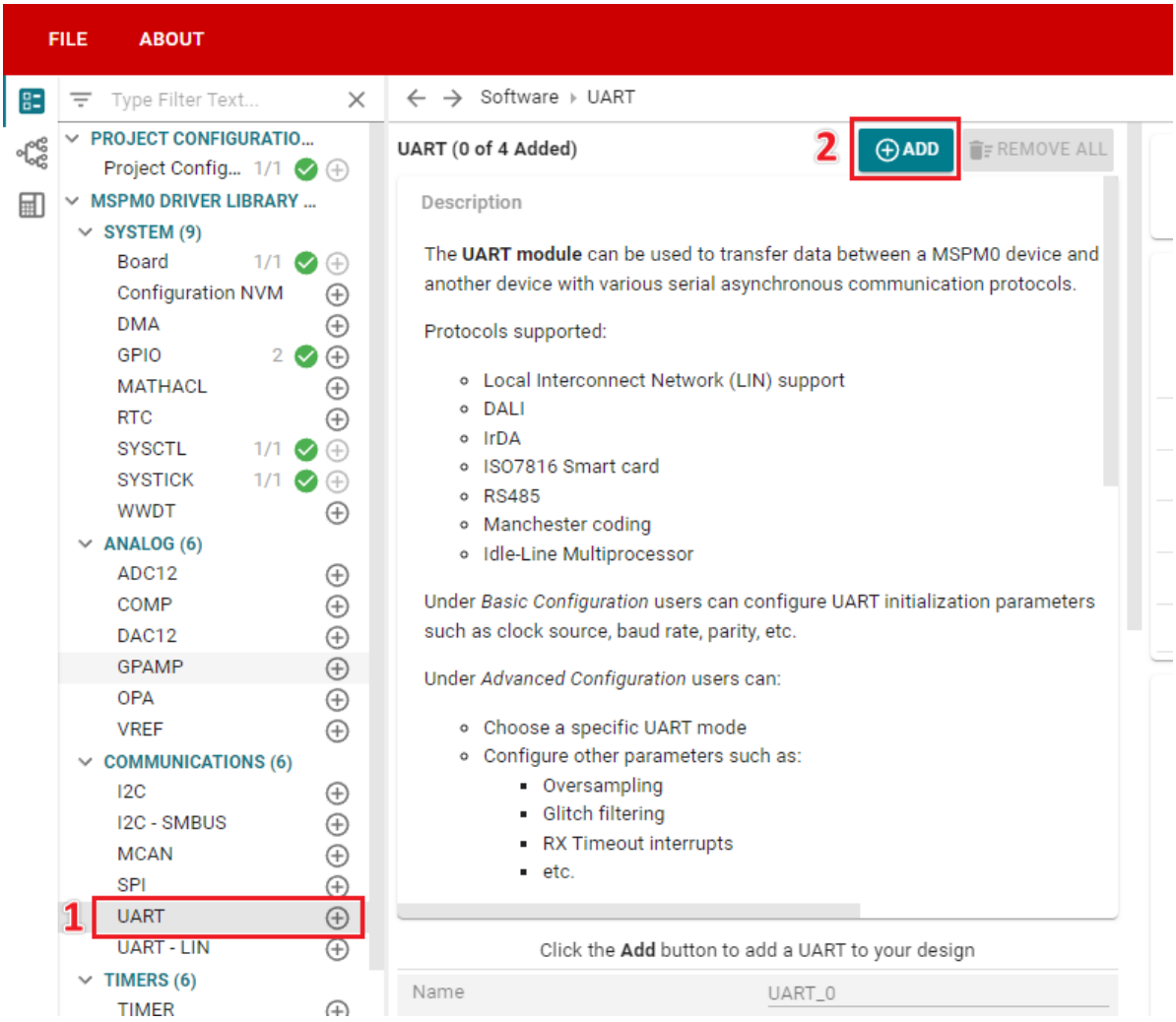


Turn on the MFCLK switch in the clock tree, as shown in the figure below, click the option block on the SYSOSC_4M branch to turn on the MFCLK clock. When 4.0000Mhz appears on the right, the setting is complete.

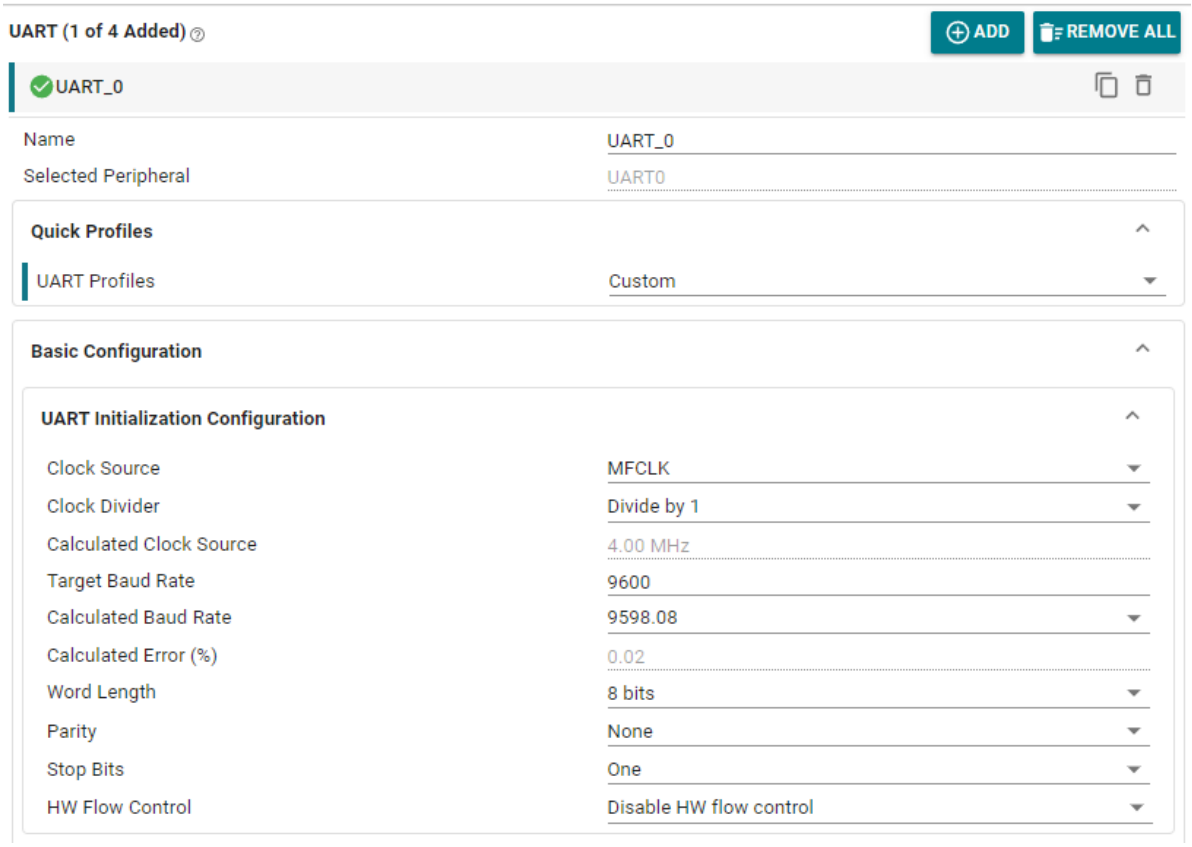


3. Serial port parameter configuration

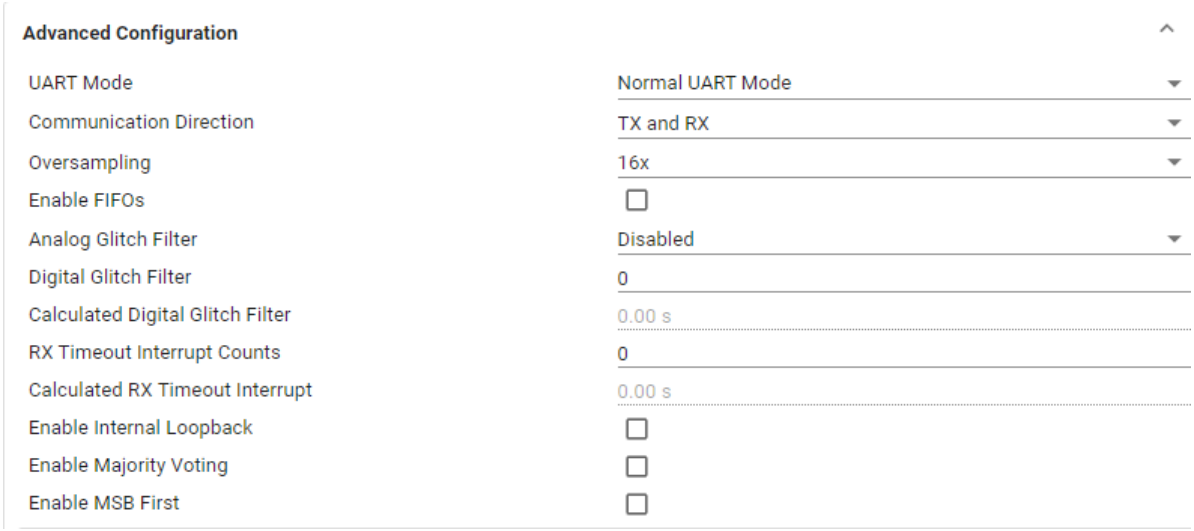
In SYSCONFIG, select MCU peripherals on the left, find the UART option and click Enter. Click ADD in UART to add a group of serial port peripherals.



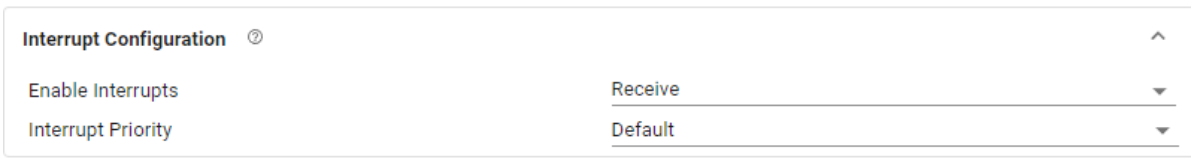
Basic configuration: Customize the serial port name, here set it to UART_0, set the Clock Source to MFCLK, select Divide by 1 for Clock Divider, and the software will automatically calculate the clock source frequency. Set the baud rate to 9600, data bits to 8, stop bits to 1, parity bit to none, and do not use hardware flow control.



Advanced configuration: Use the default options. The sampling period Oversampling has three options: 3, 8, and 16. Usually, 3 and 8 are sufficient. If the clock deviation is large or the clock speed is too fast, resulting in abnormal calculation of the baud rate division coefficient, select 16. **Just don't let sysconfig report an error.** Here our serial port clock frequency is 4MHz, so we choose 16 gears.

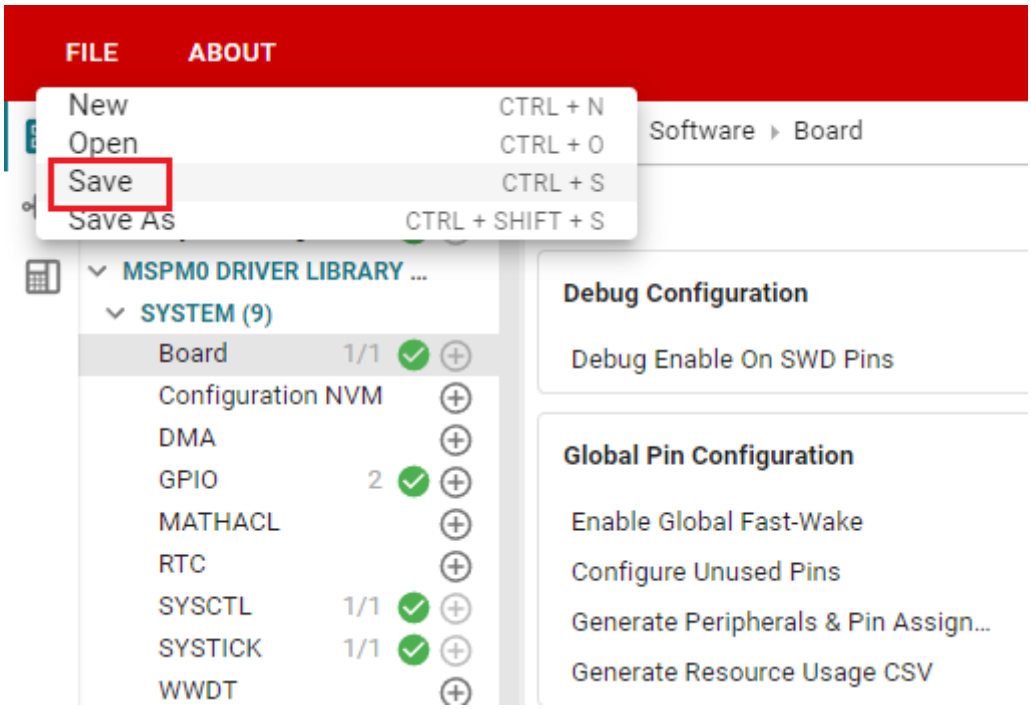


Interrupt configuration: The case uses interrupt serial port reception, and each time a byte is received, it will enter the receive interrupt.

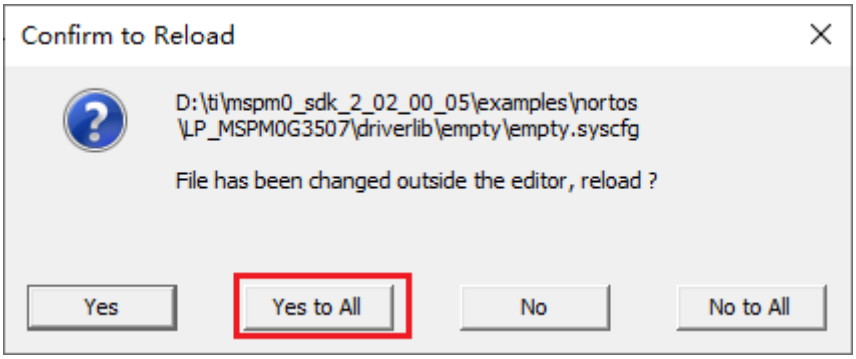


Pin configuration: According to the schematic diagram, PA10 is TX and PA11 is RX, which is serial port 0

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



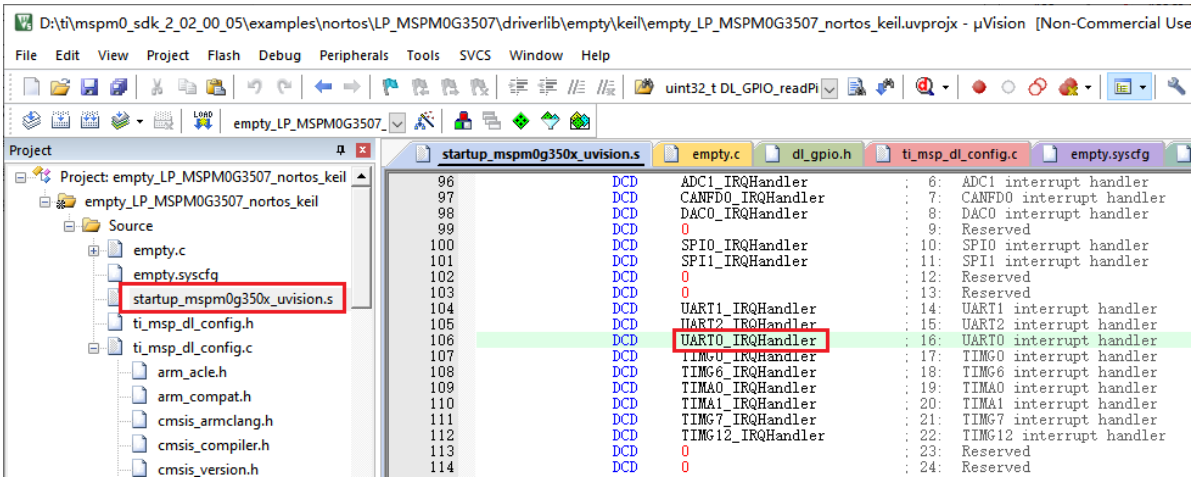
Select `Yes to All` in the pop-up window



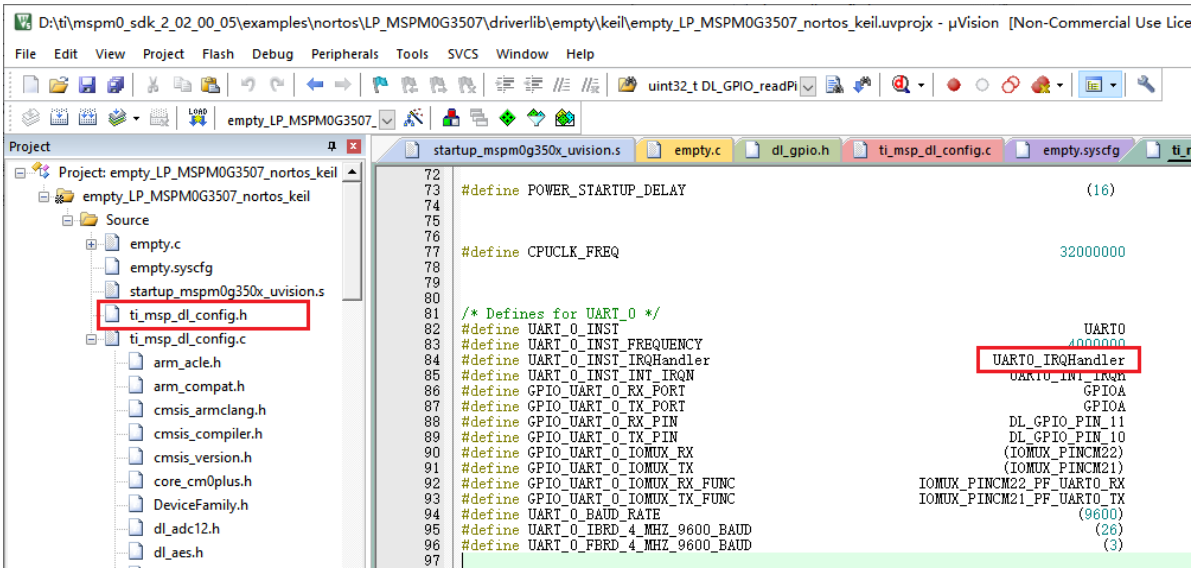
Similarly, we also need to confirm whether the ti_msp_dl_config.c and ti_msp_dl_config.h files are updated. Compile directly, and the compilation will automatically update to keil. If there is no update, we can also copy the file content in SYSCONFIG.

4. Write the program

After configuring the serial port, we also need to manually write the interrupt service function of the serial port. Because we have enabled the serial port receive interrupt, when the serial port receives data, it will trigger an interrupt, and the interrupt service function will be executed when the interrupt is triggered. The interrupt service function name 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.



You can also see the macro definition of the serial port 0 interrupt service function in the ti_msp_dl_config.h file we generated.



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

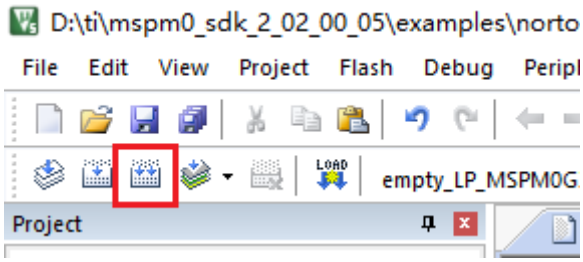
```
#include "ti_msp_dl_config.h"

volatile unsigned int delay_times = 0;
volatile unsigned char uart_data = 0;

void delay_ms(unsigned int ms);
void uart0_send_char(char ch);
void uart0_send_string(char* str);

int main(void)
{
    SYSCFG_DL_init();
    //清除串口中断标志 Clear the serial port interrupt flag
    NVIC_ClearPendingIRQ(UART_0_INST_INT_IRQN);
    //使能串口中断 Enable serial port interrupt
    NVIC_EnableIRQ(UART_0_INST_INT_IRQN);

    while (1)
    {
        //LED引脚输出高电平 LED pin outputs high level
        DL_GPIO_setPins(LED1_PORT, LED1_PIN_2_PIN);
        delay_ms(500);
```

```
        //LED引脚输出低电平 LED pin outputs low level
        DL_GPIO_clearPins(LED1_PORT, LED1_PIN_2_PIN);
        delay_ms(500);
    }
}

//搭配滴答定时器实现的精确ms延时 Accurate ms delay with tick timer
void delay_ms(unsigned int ms)
{
    delay_times = ms;
    while( delay_times != 0 );
}
//串口发送单个字符 Send a single character through the serial port
void uart0_send_char(char ch)
{
    //当串口0忙的时候等待，不忙的时候再发送传进来的字符
    // Wait when serial port 0 is busy, and send the incoming characters when it
is not busy
    while( DL_UART_isBusy(UART_0_INST) == true );
    //发送单个字符 Send a single character
    DL_UART_Main_transmitData(UART_0_INST, ch);
}
//串口发送字符串 Send string via serial port
void uart0_send_string(char* str)
{
    //当前字符串地址不在结尾 并且 字符串首地址不为空
    // The current string address is not at the end and the string first address
is not empty
    while(*str!=0&&str!=0)
    {
        //发送字符串首地址中的字符，并且在发送完成之后首地址自增
        // Send the characters in the first address of the string, and increment
the first address after sending.
        uart0_send_char(*str++);
    }
}

//滴答定时器的中断服务函数 Tick ??timer interrupt service function
void SysTick_Handler(void)
{
    if( delay_times != 0 )
    {
        delay_times--;
    }
}

//串口的中断服务函数 Serial port interrupt service function
void UART_0_INST_IRQHandler(void)
{
    //如果产生了串口中断 If a serial port interrupt occurs
    switch( DL_UART_getPendingInterrupt(UART_0_INST) )
    {
        case DL_UART_IIDX_RX://如果是接收中断 If it is a receive interrupt
            //接发送过来的数据保存在变量中 The data sent is saved in the variable
            uart_data = DL_UART_Main_receiveData(UART_0_INST);
            //将保存的数据再发送出去 Send the saved data again
            uart0_send_char(uart_data);
            break;
```

```
        default://其他的串口中断 Other serial port interrupts
            break;
    }
}
```

5. 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

- empty.c

```c
int main(void)
{
    SYSCFG_DL_init();
    //清除串口中断标志 Clear the serial port interrupt flag
    NVIC_ClearPendingIRQ(UART_0_INST_INT_IRQN);
    //使能串口中断 Enable serial port interrupt
    NVIC_EnableIRQ(UART_0_INST_INT_IRQN);

    while (1)
    {
        //LED引脚输出高电平 LED pin outputs high level
        DL_GPIO_setPins(LED1_PORT, LED1_PIN_2_PIN);
        delay_ms(500);
        //LED引脚输出低电平 LED pin outputs low level
        DL_GPIO_clearPins(LED1_PORT, LED1_PIN_2_PIN);
        delay_ms(500);
    }
}
```

The serial port interrupt must be enabled manually. The function `NVIC_EnableIRQ` specifies to enable a certain interrupt. Before enabling, the interrupt flag must be cleared first, otherwise the interrupt will be automatically entered after enabling.

```c
//串口发送单个字符 Send a single character through the serial port
```

```
void uart0_send_char(char ch)
{
    //当串口0忙的时候等待，不忙的时候再发送传进来的字符
    // Wait when serial port 0 is busy, and send the incoming characters when it
is not busy
    while( DL_UART_isBusy(UART_0_INST) == true );
    //发送单个字符 Send a single character
    DL_UART_Main_transmitData(UART_0_INST, ch);
}
//串口发送字符串 Send string via serial port
void uart0_send_string(char* str)
{
    //当前字符串地址不在结尾 并且 字符串首地址不为空
    // The current string address is not at the end and the string first address
is not empty
    while(*str!=0&&str!=0)
    {
        //发送字符串首地址中的字符，并且在发送完成之后首地址自增
        // Send the characters in the first address of the string, and increment
the first address after sending.
        uart0_send_char(*str++);
    }
}
```

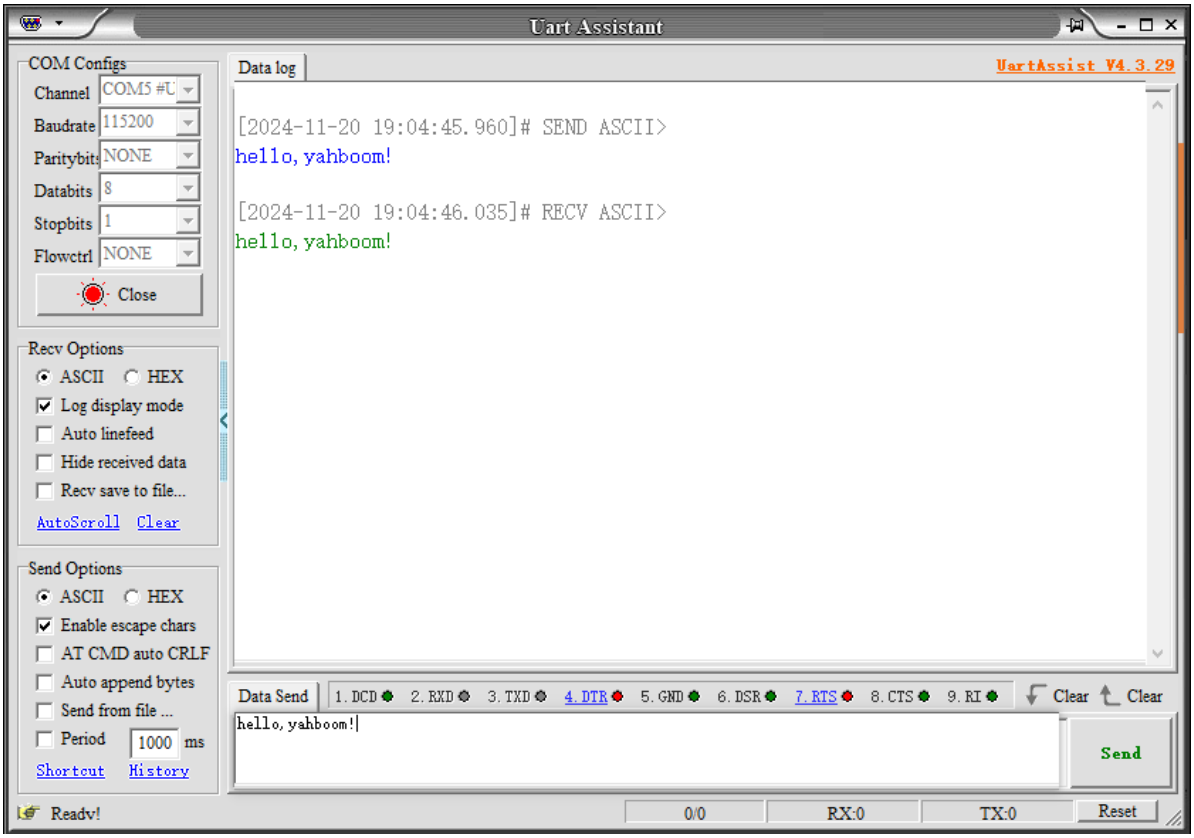
Define two functions, the `uart0_send_char` function sends a single character through the serial port, and the `uart0_send_string` function sends a string through the serial port.

```
//串口的中断服务函数 Serial port interrupt service function
void UART_0_INST_IRQHandler(void)
{
    //如果产生了串口中断 If a serial port interrupt occurs
    switch( DL_UART_getPendingInterrupt(UART_0_INST) )
    {
        case DL_UART_IIDX_RX://如果是接收中断 If it is a receive interrupt
            //接发送过来的数据保存在变量中 The data sent is saved in the variable
            uart_data = DL_UART_Main_receiveData(UART_0_INST);
            //将保存的数据再发送出去 Send the saved data again
            uart0_send_char(uart_data);
            break;

        default://其他的串口中断 Other serial port interrupts
            break;
    }
}
```

`UART_0_INST_IRQHandler` function implements the function of receiving and returning data through the serial port. Get the current interrupt type of UART0 through `DL_UART_getPendingInterrupt()`. If a UART interrupt triggered by data is received, the received data is saved in a variable and then sent out.

5. Experimental phenomenon

After the program is downloaded, you can send data to the development board through the serial port debugging assistant, and you can see the data returned by the development board.


Uart Assistant
COM Configs
Channel COM5 #U
Baudrate 115200
Paritybits NONE
Databits 8
Stopbits 1
Flowctrl NONE
Close
Recv Options
ASCII HEX
Log display mode
Auto linefeed
Hide received data
Recv save to file...
AutoScroll Clear
Send Options
ASCII HEX
Enable escape chars
AT CMD auto CRLF
Auto append bytes
Send from file ...
Period 1000 ms
Shortcut History
Data log
UartAssist V4.3.29
[2024-11-20 19:04:45.960]# SEND ASCII>
hello,yahboom!
[2024-11-20 19:04:46.035]# RECV ASCII>
hello,yahboom!
Data Send
1.DCD 2.RXD 3.TXD 4.DTR 5.GND 6.DSR 7.RTS 8.CTS 9.RI
Clear Clear
hello,yahboom!
Send
Readv!
0/0
RX:0
TX:0
Reset