Serial Communication

I. Learning Objectives

1. Learn the basic usage of MSPM0G3507 development board serial port.
2. Understand basic knowledge of serial communication.

Serial Communication

Serial Communication refers to a communication method where peripherals and computers transmit data bit by bit through data signal lines, ground lines, etc., belonging to serial communication methods.

Key parameters of serial communication include baud rate, data bits, stop bits, and parity bits. To ensure normal communication, these parameters must be consistent between both communication parties.

Serial Communication Working Modes

1. **Simplex Mode**
   - **Characteristics**: Data can only be transmitted in one direction, the sending end is responsible for sending, the receiving end is responsible for receiving, with no reverse communication.
   - **Applications**: Suitable for scenarios that only require one-way information transmission, such as sensor data output.
2. **Full-Duplex Mode**
   - **Characteristics**: Data can be transmitted simultaneously in both directions, with sending and receiving not interfering with each other.
   - **Implementation**: Requires two data lines, respectively used for sending (TX) and receiving (RX).
   - **Applications**: Commonly used in scenarios requiring efficient, real-time communication, such as computer and peripheral interaction.
3. **Half-Duplex Mode**
   - **Characteristics**: Data is transmitted in both directions, but cannot be simultaneous; sending and receiving must alternate.
   - **Implementation**: Sending and receiving share one communication line, relying on timing to switch directions.

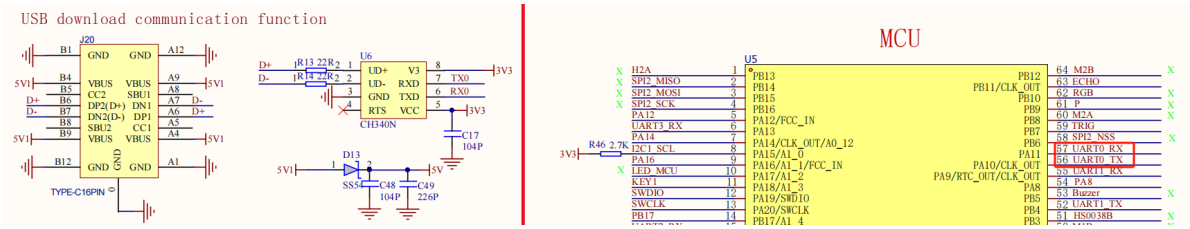
- **Applications**: Suitable for scenarios with low real-time requirements, such as walkie-talkie communication.

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

II. Hardware Setup

Since the development board has a built-in CH340 serial port circuit, serial communication can be achieved directly through USB cable without external USB to TTL module.

Serial Port Part Circuit Diagram



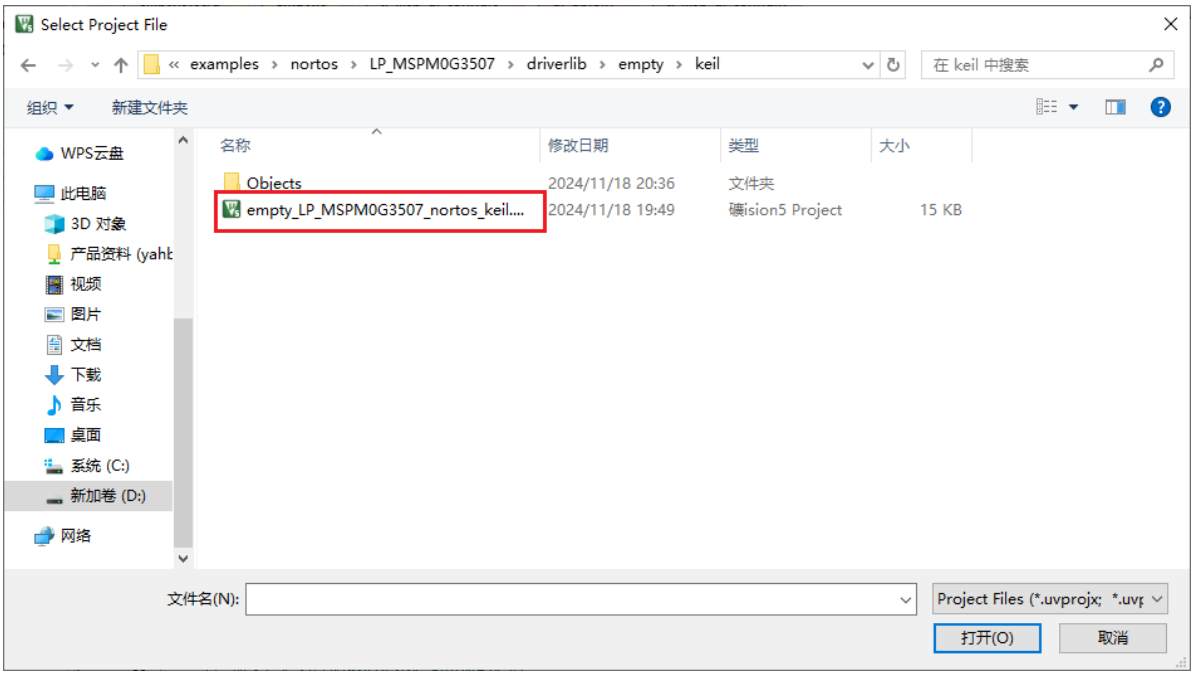
21	PA10		UART0_TX *[2]* / SPI0_POCI *[3]* / I2C0_SDA *[4]* / TIMA1_C0 *[5]* / TIMG12_C0 *[6]* / TIMA0_C2 *[7]* / I2C1_SDA *[8]* / CLK_OUT *[9]*/(Default BSL UART_TX)	56	18	14	15	High-Drive
22	PA11		UART0_RX *[2]* / SPI0_SCK *[3]* / I2C0_SCL *[4]* / TIMA1_C1 *[5]* / COMP0_OUT *[6]*/ TIMA0_C2N *[7]* /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

III. Experiment Steps

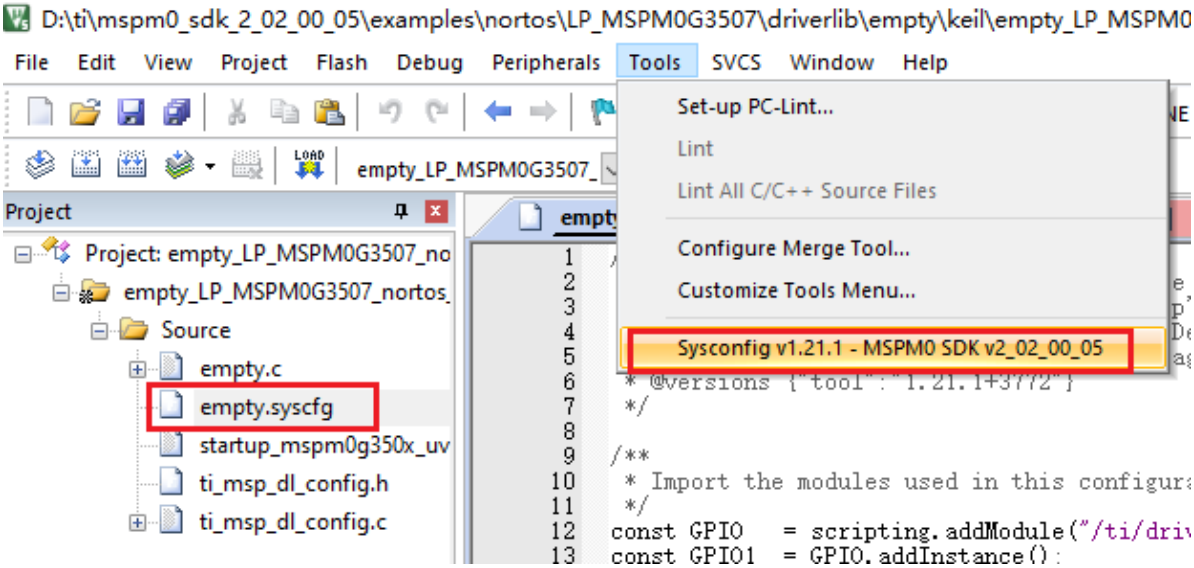
This lesson uses the UART0 peripheral on pins PA10 and PA11, combined with the onboard CH340 USB to serial chip, to achieve the operation of receiving serial data sent by the host computer and then sending the data back to the host computer through the serial port sending function.

1. Open SYSCONFIG Configuration Tool

In KEIL, open the empty project from the SDK.



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.



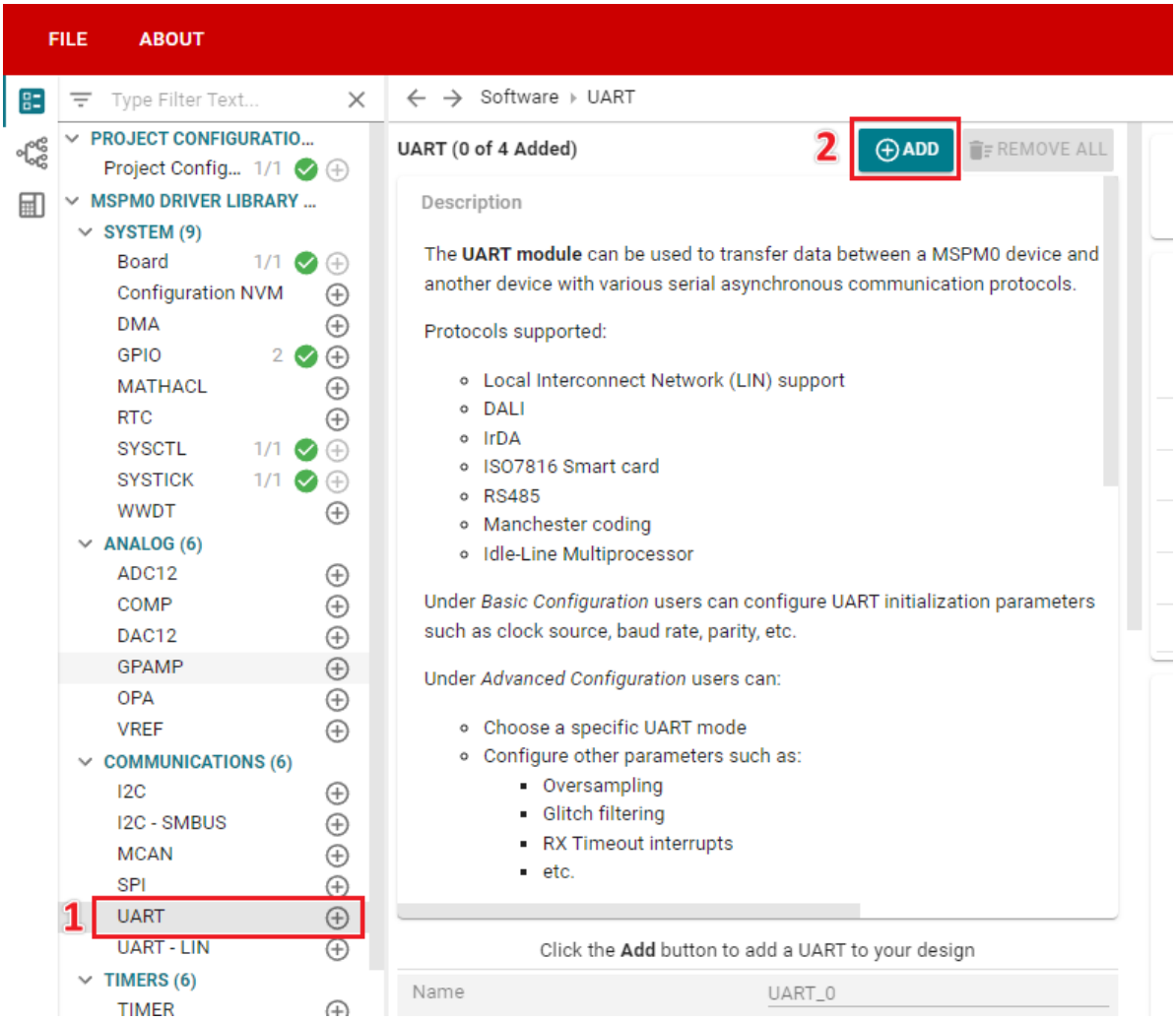
2. Serial Clock Configuration

There are three clock sources for the serial port:

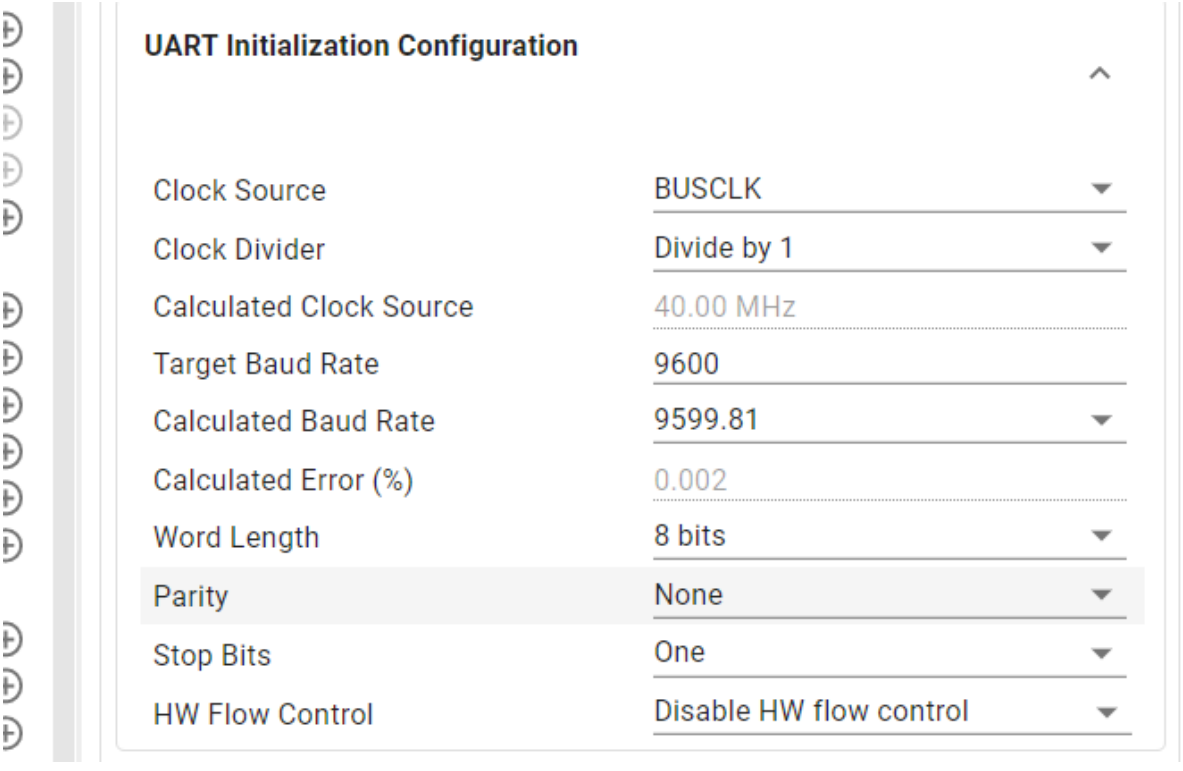
1. **BUSCLK**: CPU clock provided by the internal high-frequency oscillator, usually set to 32MHz at chip factory.
2. **MFCLK**: Can only use a fixed 4MHz clock. To enable it, you need to configure the SYSOSC_4M branch of the clock tree to enable it properly.
3. **LFCLK**: Clock provided by the internal low-frequency oscillator (32KHz). Effective in run, sleep, stop, and standby modes, using this clock can achieve lower power consumption.

3. Serial Parameter Configuration

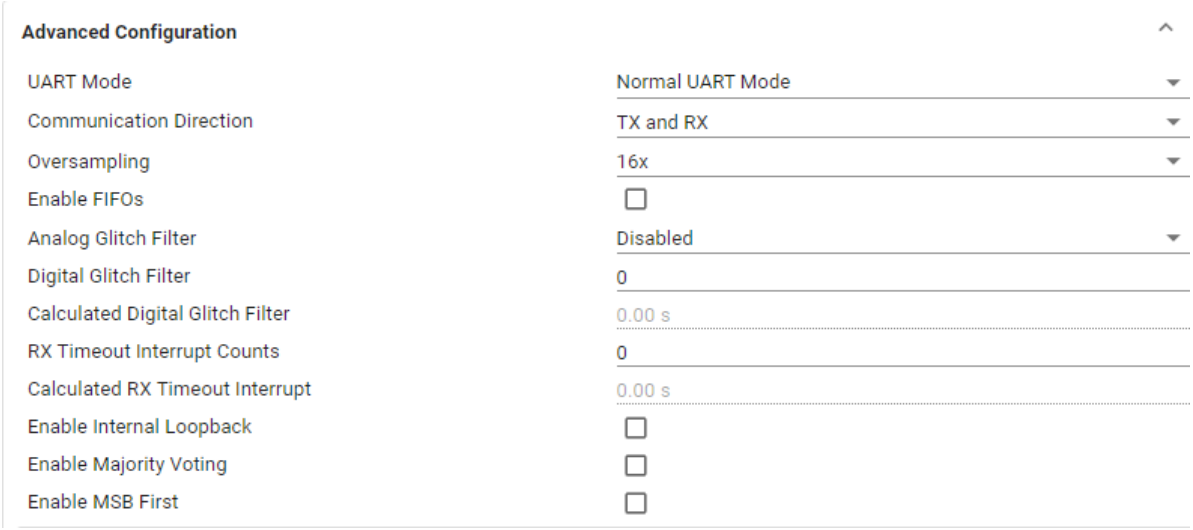
In SYSCONFIG, select MCU peripherals on the left, find UART option and click to enter. In UART, click ADD to add a serial peripheral group.



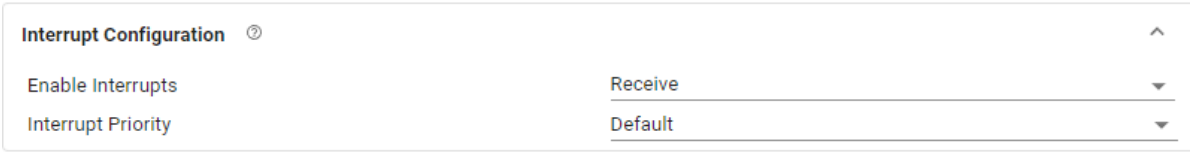
Basic Configuration: Customize the serial port name, here set as UART_0, Clock Source set to BUSCLK, Clock Divider select Divide by 1, the software will automatically calculate the clock source frequency. Set baud rate to 9600, 8 data bits, 1 stop bit, no parity, and no hardware flow control configuration.



Advanced Configuration: Use default options. Oversampling has three options: 3, 8, 16, etc. Usually, 3 and 8 are sufficient. If the clock deviation is large or the clock speed is too fast causing abnormal baud rate division coefficient calculation, choose 16. **Just don't let sysconfig report errors.** Here our serial clock frequency is 4MHz, so we choose 16.

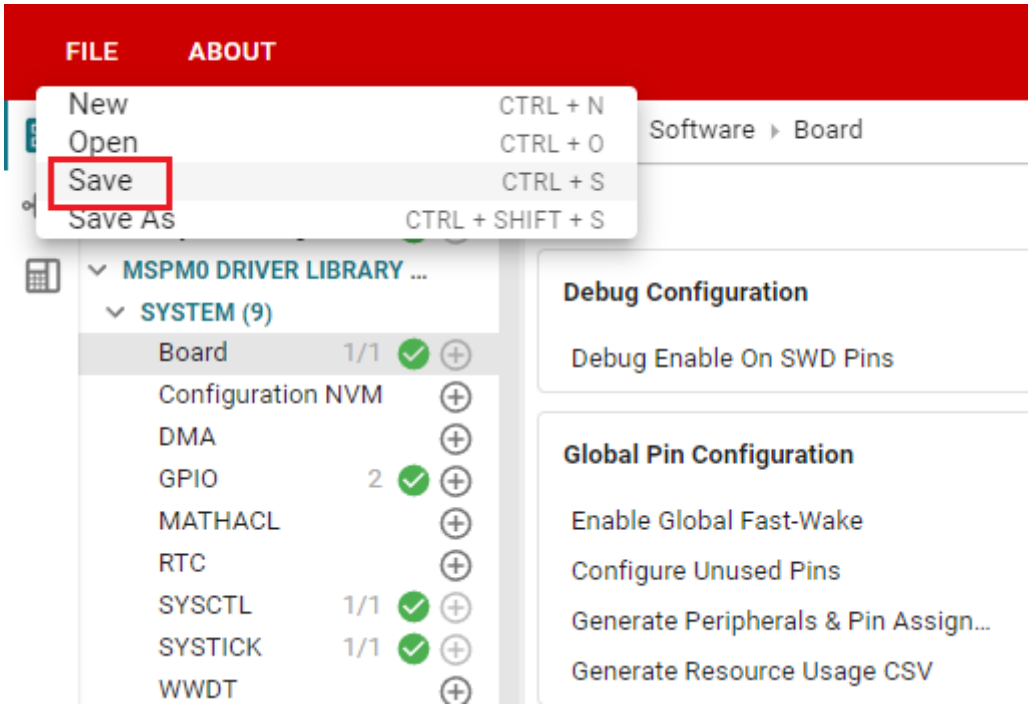


Interrupt Configuration: This example uses interrupt-based serial reception. Whenever a byte is received, it will enter the receive interrupt.

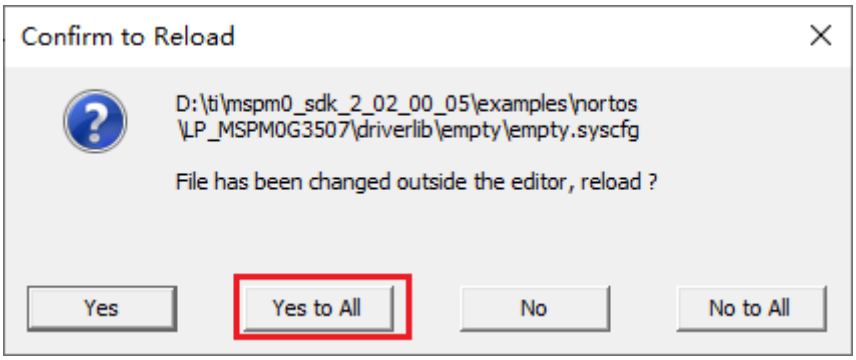


Pin Configuration: According to the schematic, PA10 is TX, PA11 is RX, which is serial port 0

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



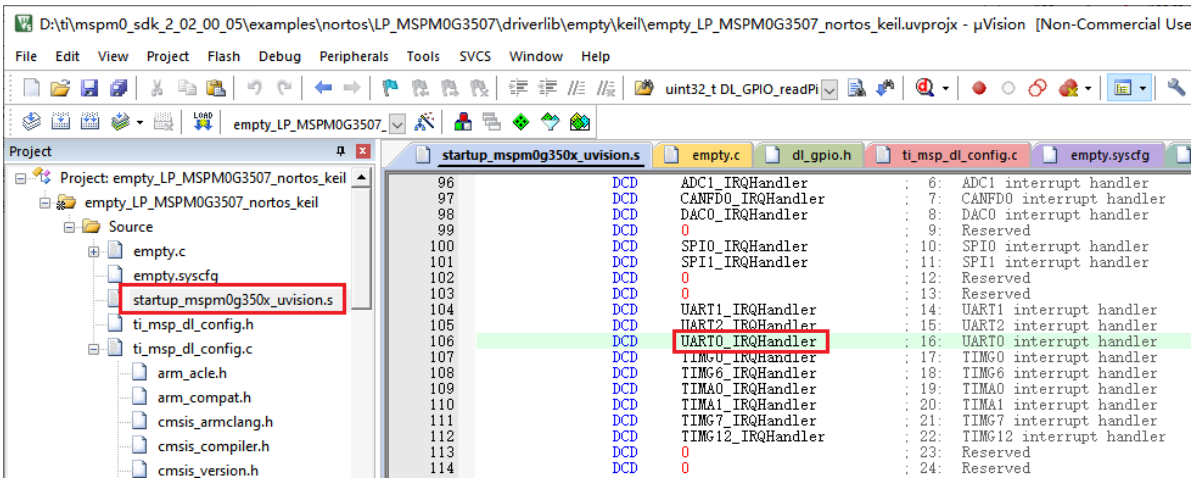
Select `Yes to All` in the pop-up window



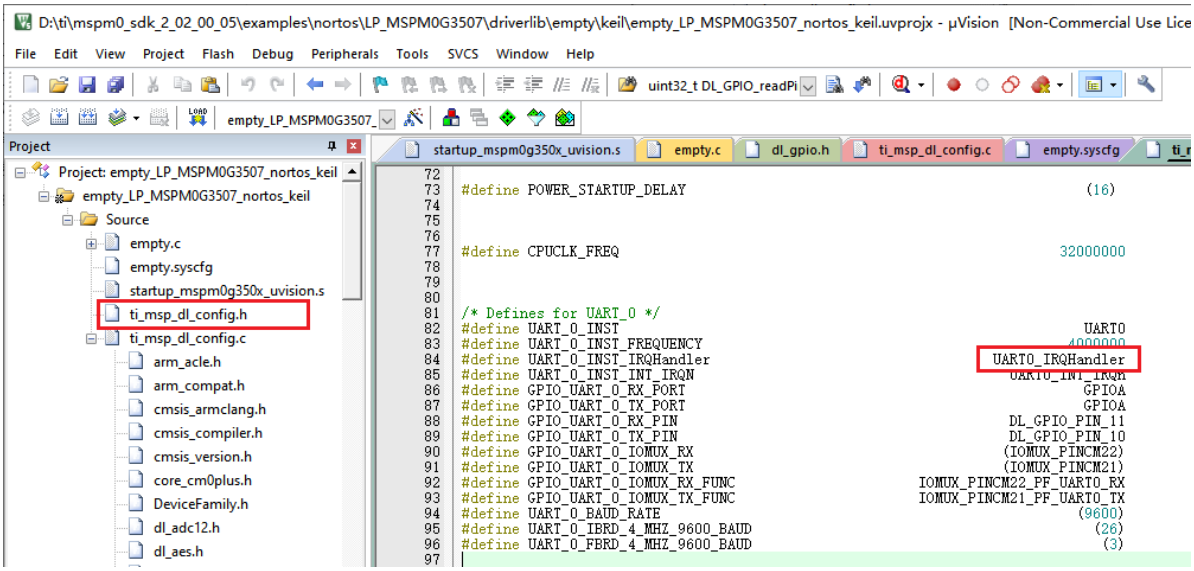
Similarly, we need to confirm whether ti_msp_dl_config.c and ti_msp_dl_config.h files are updated. Directly compile, compilation will automatically update to keil. If not updated, we can also copy the file content from SYSCONFIG.

4. Write Program

After configuring the serial port, we also need to manually write the serial port interrupt service function. Because we enabled the serial port receive interrupt, when the serial port receives data, it will trigger an interrupt, and triggering the interrupt will execute the interrupt service function. The name of each interrupt service function is usually fixed and cannot be arbitrarily modified, otherwise it will not be possible to correctly enter the interrupt service function. 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 for the serial port 0 interrupt service function in our generated ti_msp_dl_config.h file.



So in the uart0.c file, write the following code

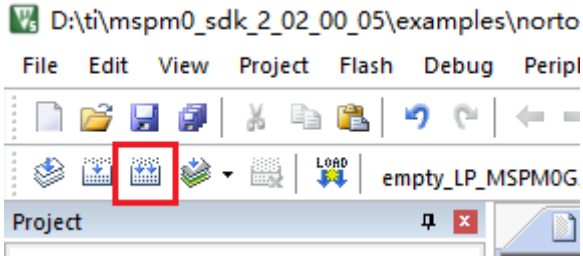
```
#include "uart0.h"


#define RE_0_BUFF_LEN_MAX   128

volatile uint8_t  recv0_buff[RE_0_BUFF_LEN_MAX] = {0};
volatile uint16_t recv0_length = 0;
volatile uint8_t  recv0_flag = 0;

void USART_Init(void)
{
    //Clear serial interrupt flag

    NVIC_ClearPendingIRQ(UART_0_INST_INT_IRQN);
    //Enable serial interrupt
    NVIC_EnableIRQ(UART_0_INST_INT_IRQN);
}

//Serial send single character

void uart0_send_char(char ch)
{
    //When serial port 0 is busy, wait, send the incoming character when not
busy

    while( DL_UART_isBusy(UART_0_INST) == true );
    //Send single character

    DL_UART_Main_transmitData(UART_0_INST, ch);

}
//Serial send string
void uart0_send_string(char* str)
{
    //Current string address is not at the end and string first address is not
null

    while(*str!=0&&str!=0)
    {
```

```
        //Send the character at the first address of the string, and increment
the first address after sending is complete

        uart0_send_char(*str++);
    }
}


#if !defined(__MICROLIB)
//If not using microlib, you need to add the following function

#if (__ARMCLIB_VERSION <= 6000000)
//If compiler is AC5, define the following structure

struct __FILE
{
    int handle;
};
#endif

FILE __stdout;

//Define _sys_exit() to avoid using semihosting mode

void _sys_exit(int x)
{
    x = x;
}
#endif
int fputc(int ch, FILE* stream)
{

    uart0_send_char(ch);

    return ch;
}


int fputs(const char* restrict s, FILE* restrict stream)
{
    uint16_t i,len;
    len=strlen(s);
    for (i=0; i<len; i++)
    {
        uart0_send_char(s[i]);
    }
    return len;
}

int puts(const char *_ptr)
{
    int count =fputs(_ptr, stdout);
    count +=fputs("\n", stdout);
    return count;
}
```

```
void UART_0_INST_IRQHandler(void)
{
    uint8_t receivedData = 0;

    //If serial interrupt occurred

    switch( DL_UART_getPendingInterrupt(UART_0_INST) )
    {
            //If it's receive interrupt
        case DL_UART_IIDX_RX:

            // Receive and save the sent data
            receivedData = DL_UART_Main_receiveData(UART_0_INST);

            // Check if buffer is full
            if (recv0_length < RE_0_BUFF_LEN_MAX - 1)
            {
                recv0_buff[recv0_length++] = receivedData;

                // Send the saved data back out, comment out if you don't want
echo back

                uart0_send_char(receivedData);
            }
            else
            {
                recv0_length = 0;
            }

            break;

        default://Other serial interrupts
            break;
    }
}
```

5. 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

- empty.c

```c
#include "ti_msp_dl_config.h"
#include "uart0.h"

int main(void)
{
        //System initialization
        SYSCFG_DL_init();
        //Serial initialization
        USART_Init();

        //Implemented output redirection

            printf("Hello,Yahboom!");


}
```

Serial interrupt must be manually enabled. The function `NVIC_EnableIRQ` specifies enabling a certain interrupt. Before enabling, you must first clear the interrupt flag, otherwise after enabling the interrupt, it will automatically enter the interrupt.

```c
//Serial send single character
void uart0_send_char(char ch)
{
    //When serial port 0 is busy, wait, send the incoming character when not
busy

    while( DL_UART_isBusy(UART_0_INST) == true );
    //Send single character
    DL_UART_Main_transmitData(UART_0_INST, ch);
}
//Serial send string
void uart0_send_string(char* str)
{
    //Current string address is not at the end and string first address is not
null

    while(*str!=0&&str!=0)
    {
        //Send the character at the first address of the string, and increment
the first address after sending is complete

        uart0_send_char(*str++);
```

```
    }
}
```

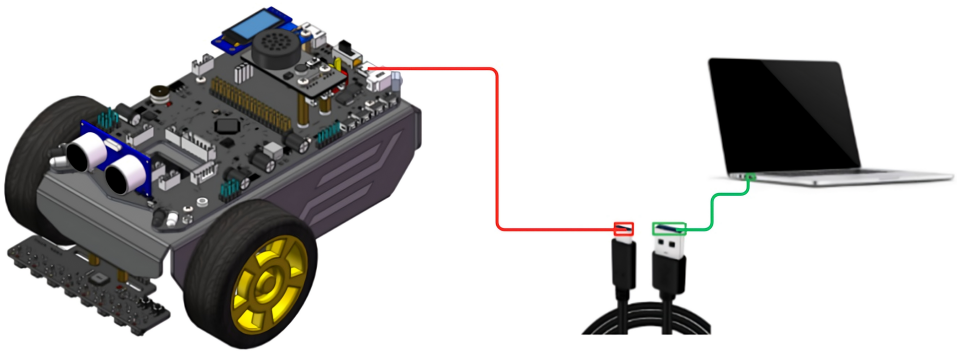
Define two functions, `uart0_send_char` function sends a single character through the serial port, `uart0_send_string` function sends a string through the serial port.

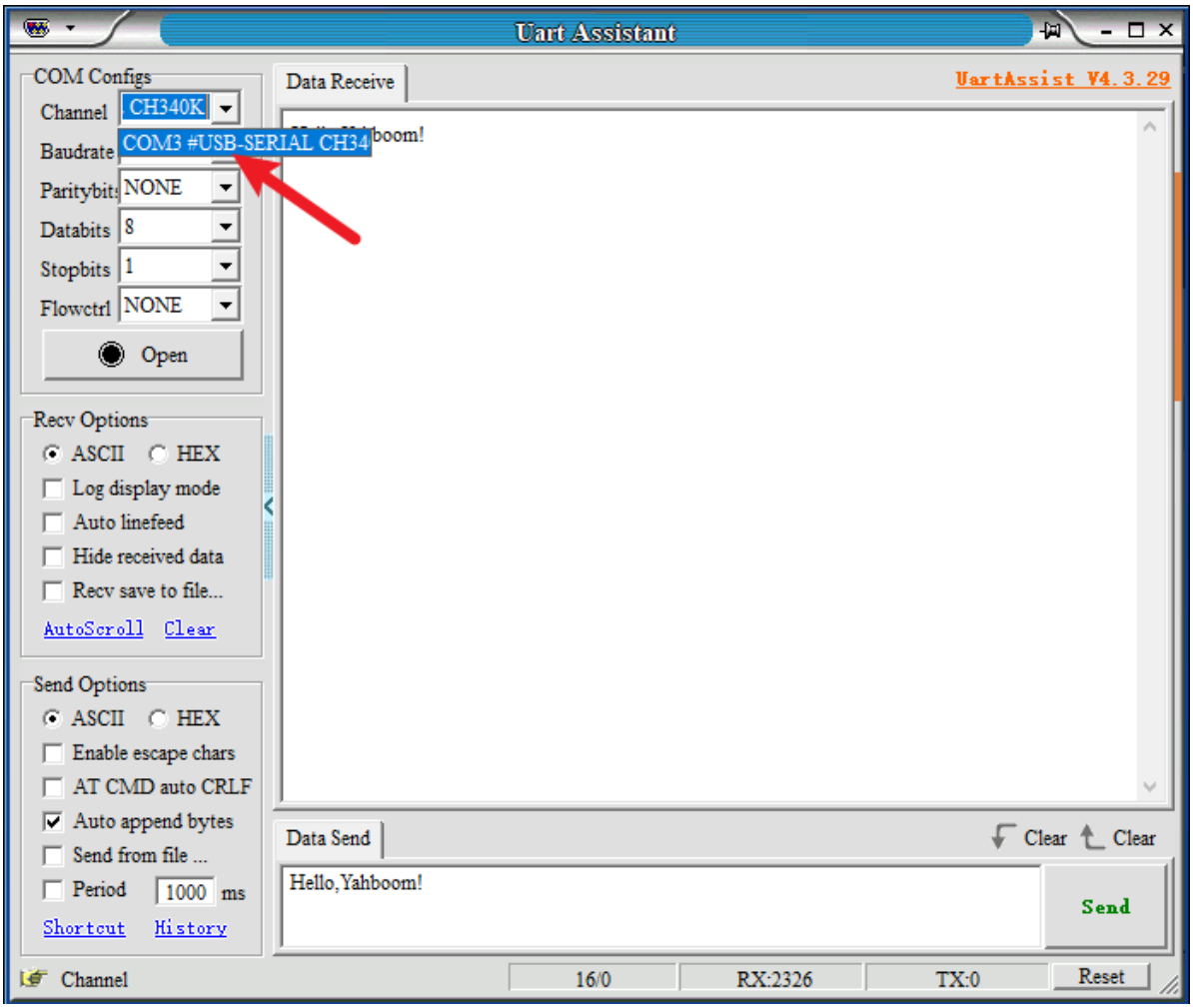
```
void UART_0_INST_IRQHandler(void)
{
    uint8_t receivedData = 0;

    //If serial interrupt occurred

    switch( DL_UART_getPendingInterrupt(UART_0_INST) )
    {
        case DL_UART_IIDX_RX://If it's receive interrupt

            // Receive and save the sent data
            receivedData = DL_UART_Main_receiveData(UART_0_INST);

            // Check if buffer is full
            if (recv0_length < RE_0_BUFF_LEN_MAX - 1)
            {
                recv0_buff[recv0_length++] = receivedData;

                // Send the saved data back out, comment out if you don't want
echo back

                uart0_send_char(receivedData);
            }
            else
            {
                recv0_length = 0;
            }

            break;

        default://Other serial interrupts
            break;
    }
}
```

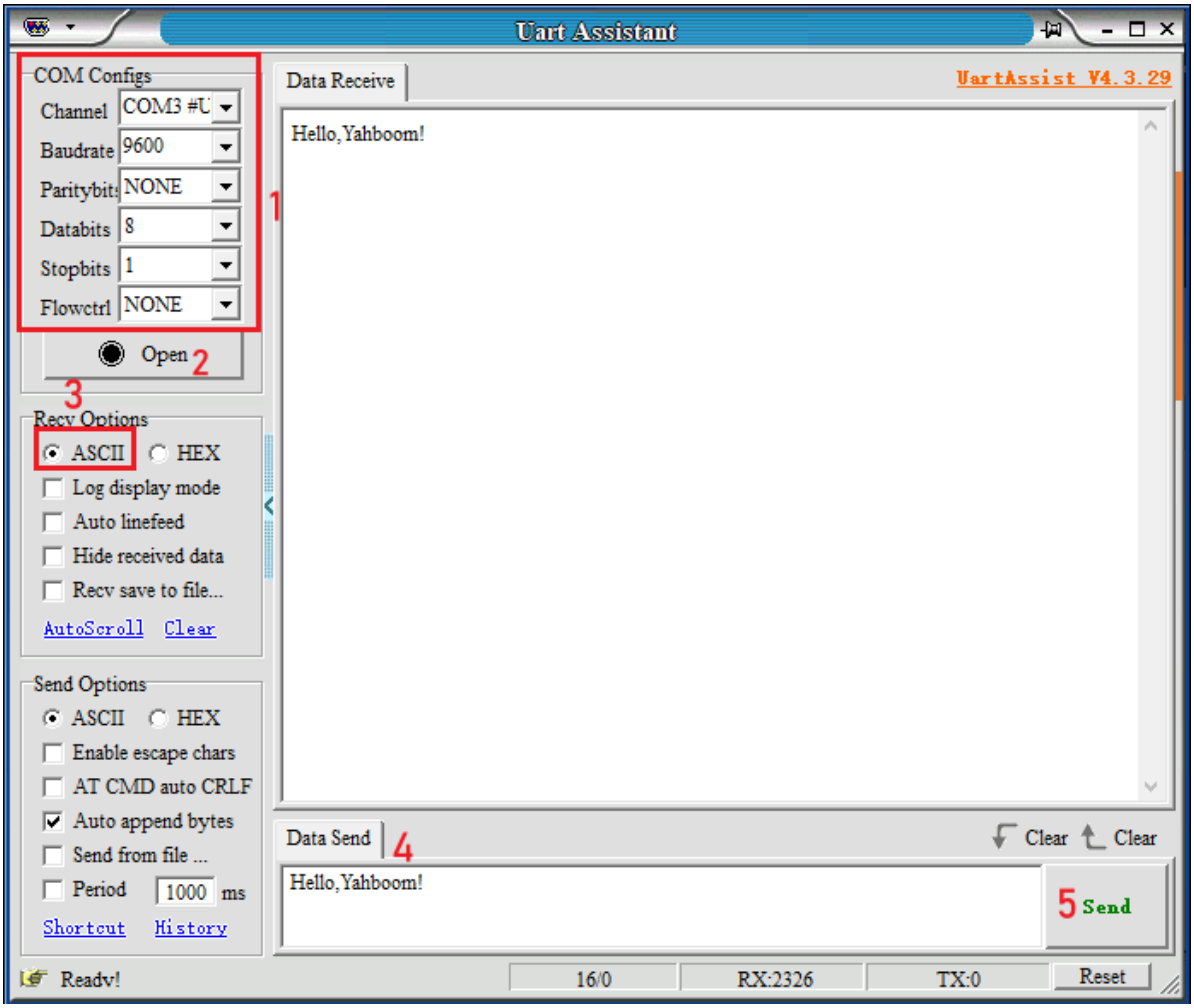
The `UART_0_INST_IRQHandler` function implements serial data reception and echo back functionality. Through `DL_UART_getPendingInterrupt()` to get the current UART0 interrupt type, if it's a UART interrupt triggered by receiving data, save the received data in a variable, then send it back out.

V. Experiment Phenomenon

After the program download is complete, we can connect the Type-C to the computer (CH340 driver needs to be installed), connection configuration as shown below, send data to the development board through the serial debugging assistant, and you can see the data returned by the development board.




Uart Assistant
COM Configs
Channel
CH340K
Baudrate
COM3 #USB-SERIAL CH34
Paritybits
NONE
Databits
8
Stopbits
1
Flowctrl
NONE
Open
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 Receive
UartAssist V4.3.29
boom!
Data Send
Clear
Clear
Hello,Yahboom!
Send
Channel
16/0
RX:2326
TX:0
Reset


Uart Assistant
COM Configs
Channel COM3 #U
Baudrate 9600
Paritybits NONE
Databits 8
Stopbits 1
Flowctrl NONE
Open
1
2
3
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 Receive
UartAssist V4.3.29
Hello,Yahboom!
Data Send
4
Clear
Clear
Hello,Yahboom!
5 Send
Readv!
16/0
RX:2326
TX:0
Reset