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.

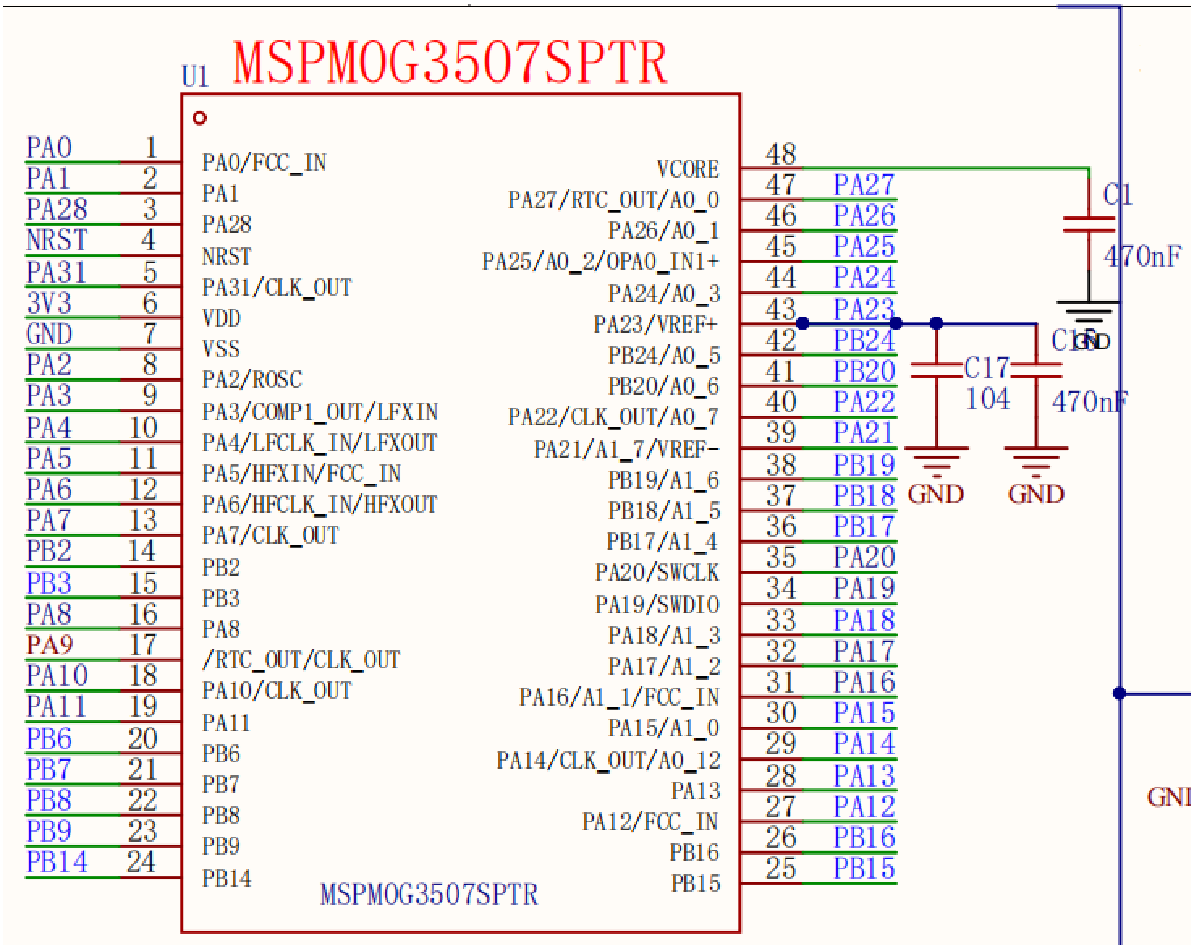
- **Implementation**: Sending and receiving share a communication line, and the direction is switched by timing.
- **Application**: Suitable for scenarios that do not require high real-time performance, such as walkie-talkie communication.

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

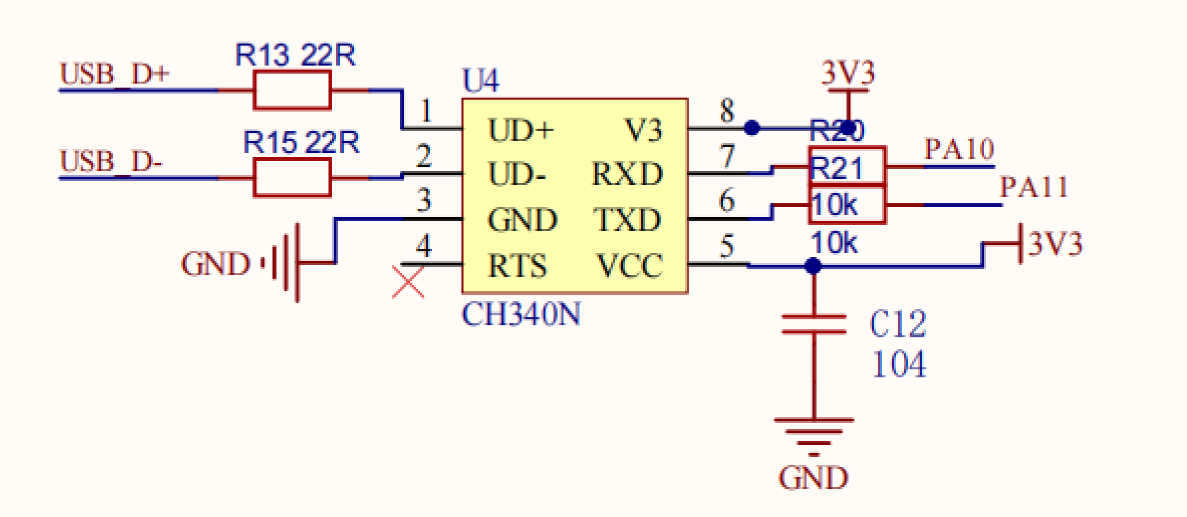
2. Hardware Construction

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



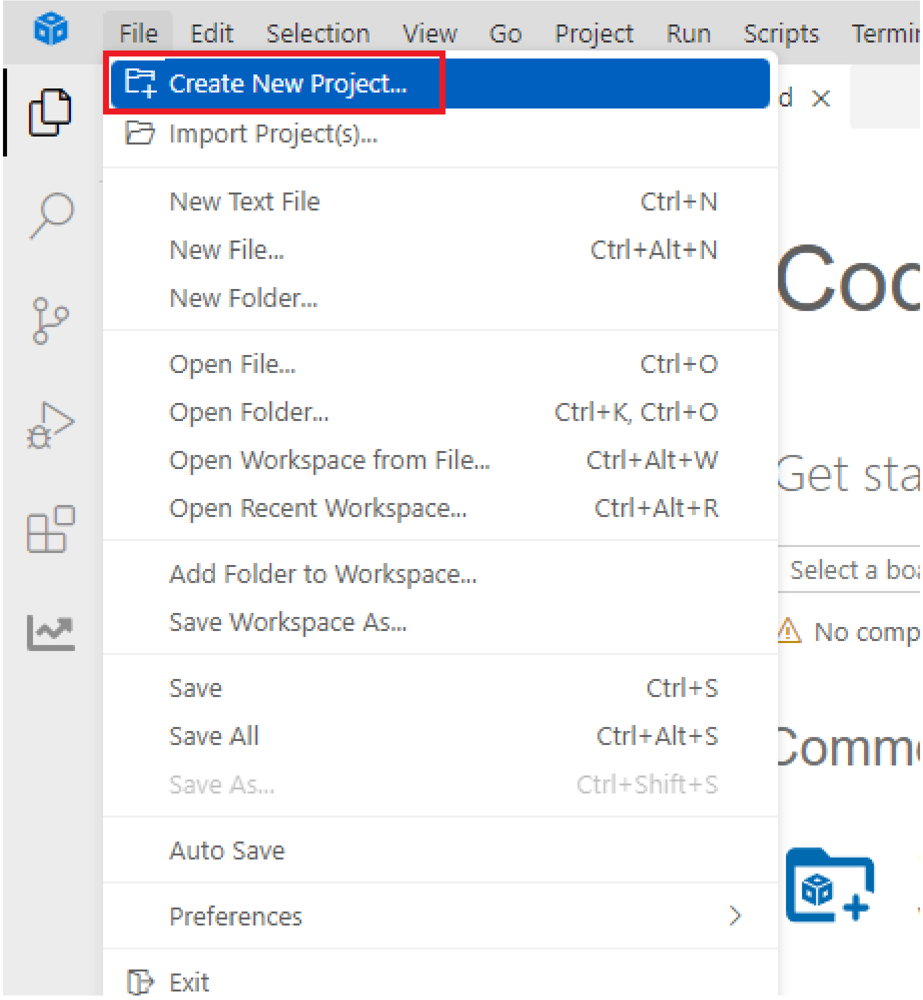
21	PA10		UART0_TX *[2]* / SPI0_POCI *[3]* / I2C0_SDA *[4]* / TIMA1_C0 *[5]* / TIMG12_C0 *[6]* / TIMA0_C2 *[7]* / I2C1_SDA *[8]* / CLK_OUT *[9]*	56	18	14	15	
22	PA11		UART0_RX *[2]* / SPI0_SCK *[3]* / I2C0_SCL *[4]* / TIMA1_C1 *[5]* / COMP0_OUT *[6]*/ TIMA0_C2N *[7]* / I2C1_SCL *[8]*	57	19	15	16	

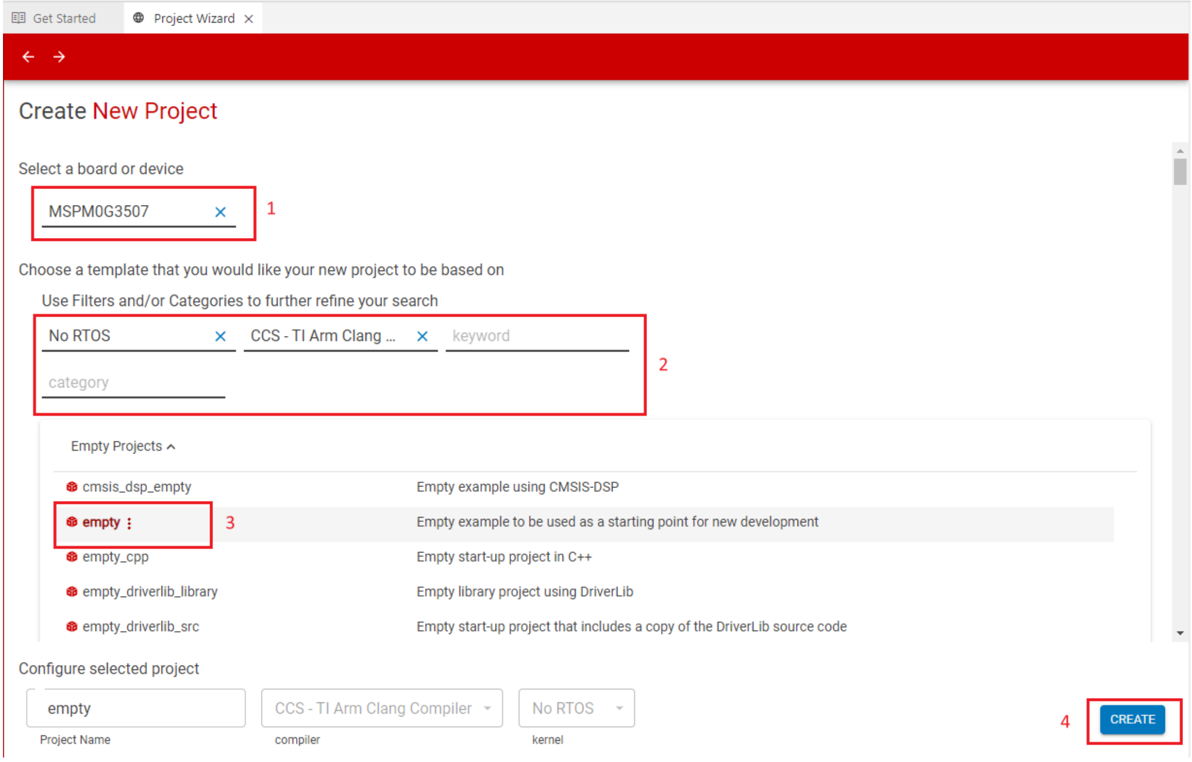
3. Experimental steps

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

Create a blank project empty in CCS.





Create New Project

Select a board or device

MSPM0G3507

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

category

Empty Projects

cmsis_dsp_empty	Empty example using CMSIS-DSP
empty	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

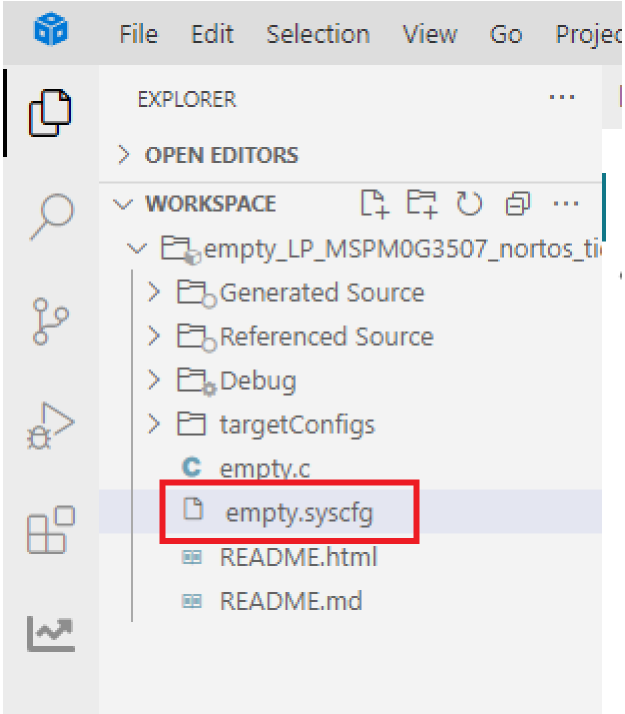
Configure selected project

Project Name: empty

compiler: CCS - TI Arm Clang Compiler

kernel: No RTOS

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



2. Serial port clock configuration

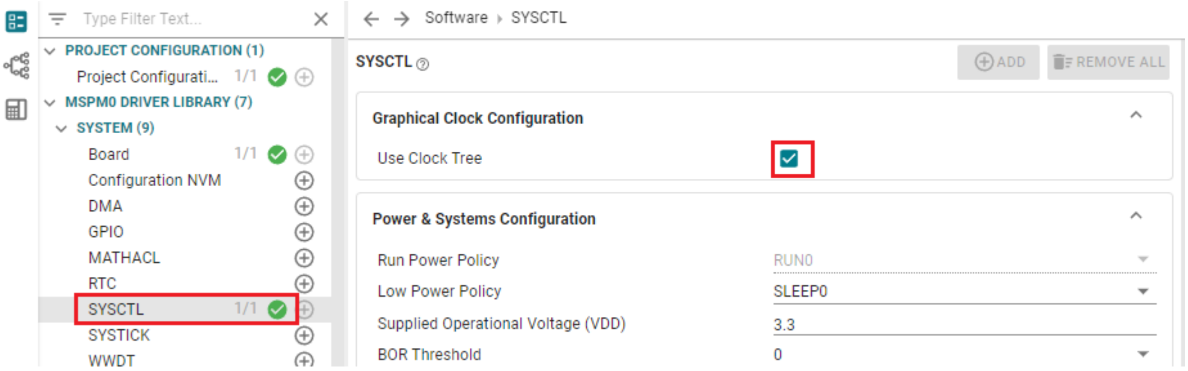
There are three sources of serial port clock, namely:

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

MFCLK: Only a fixed 4MHz clock can be used (refer to page 132 of the user manual). If enabled, the SYSOSC_4M branch of the clock tree needs to be configured to enable it normally.

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.



Graphical Clock Configuration

Use Clock Tree ☑

Power & Systems Configuration

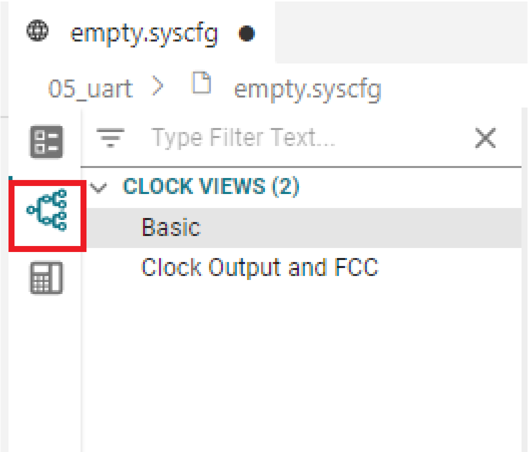
Run Power Policy: RUN0

Low Power Policy: SLEEP0

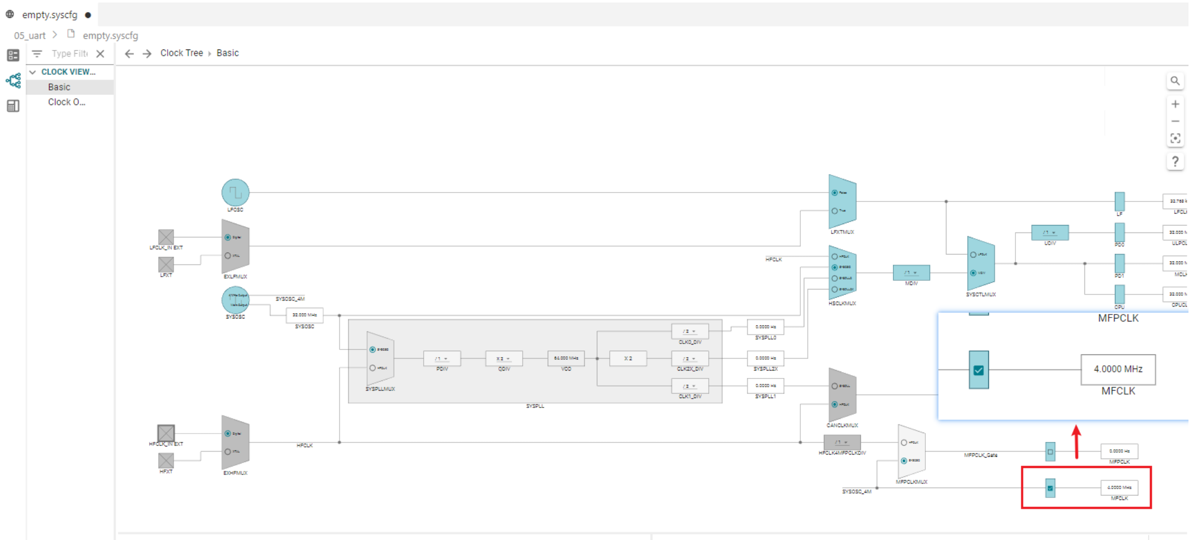
Supplied Operational Voltage (VDD): 3.3

BOR Threshold: 0

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

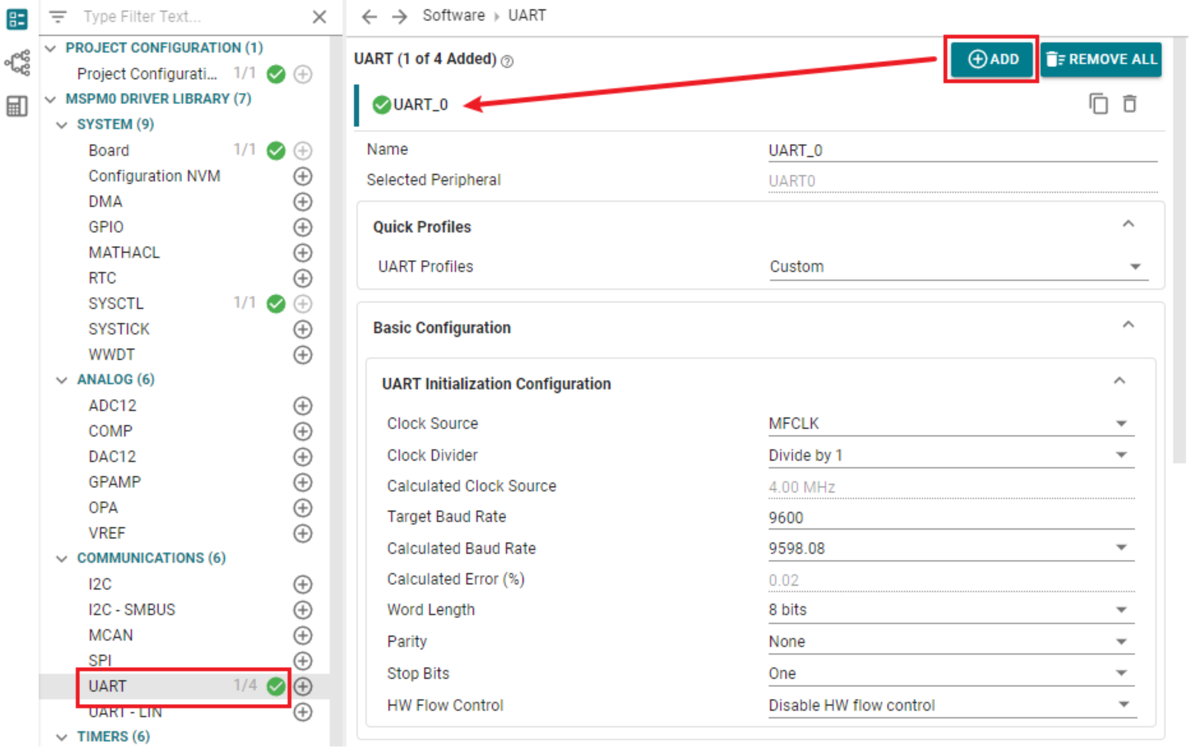


Turn on the MFCLK switch in the clock tree, as shown in the figure below, click the option block on the SYSOSC_4M branch to turn on the MFCLK clock. When 4.0000Mhz appears on the right, the setting is complete.



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.



UART (1 of 4 Added)

UART_0

Name UART_0

Selected Peripheral UART0

Quick Profiles

UART Profiles Custom

Basic Configuration

UART Initialization Configuration

Clock Source MFCLK

Clock Divider Divide by 1

Calculated Clock Source 4.00 MHz

Target Baud Rate 9600

Calculated Baud Rate 9598.08

Calculated Error (%) 0.02

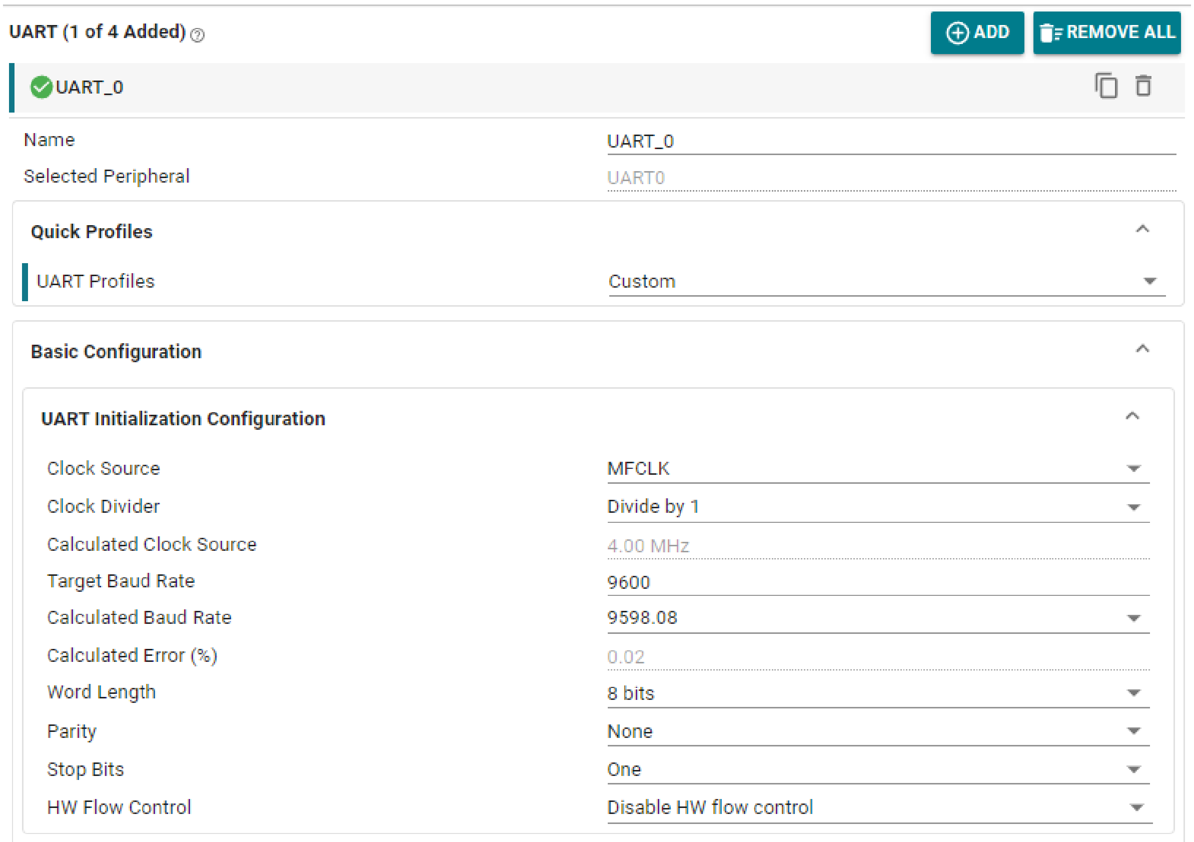
Word Length 8 bits

Parity None

Stop Bits One

HW Flow Control Disable HW flow control

The configuration is consistent with the keil environment case, so it will not be explained here.



UART (1 of 4 Added)

UART_0

Name UART_0

Selected Peripheral UART0

Quick Profiles

UART Profiles Custom

Basic Configuration

UART Initialization Configuration

Clock Source MFCLK

Clock Divider Divide by 1

Calculated Clock Source 4.00 MHz

Target Baud Rate 9600

Calculated Baud Rate 9598.08

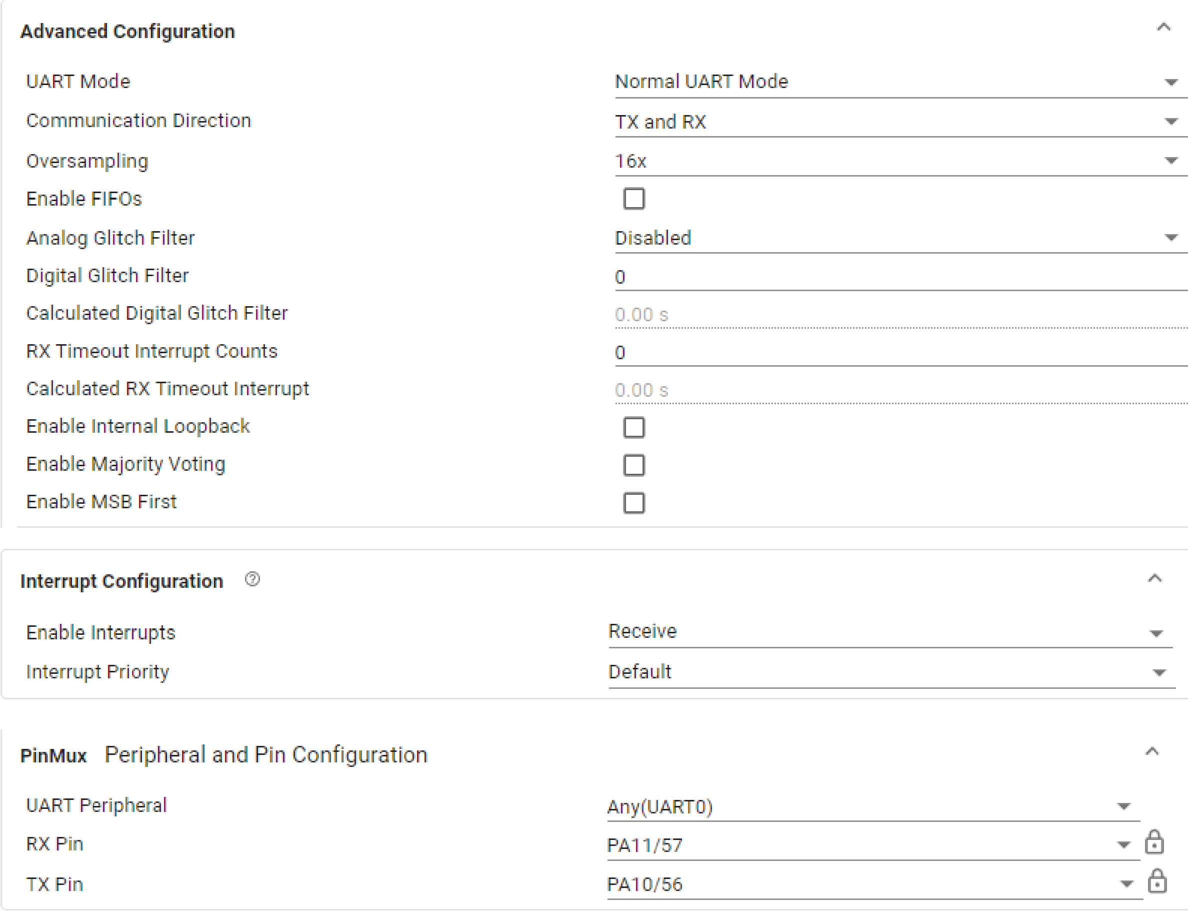
Calculated Error (%) 0.02

Word Length 8 bits

Parity None

Stop Bits One

HW Flow Control Disable HW flow control



Advanced Configuration

UART Mode	Normal UART Mode
Communication Direction	TX and RX
Oversampling	16x
Enable FIFOs	□
Analog Glitch Filter	Disabled
Digital Glitch Filter	0
Calculated Digital Glitch Filter	0.00 s
RX Timeout Interrupt Counts	0
Calculated RX Timeout Interrupt	0.00 s
Enable Internal Loopback	□
Enable Majority Voting	□
Enable MSB First	□

Interrupt Configuration

Enable Interrupts	Receive
Interrupt Priority	Default

PinMux Peripheral and Pin Configuration

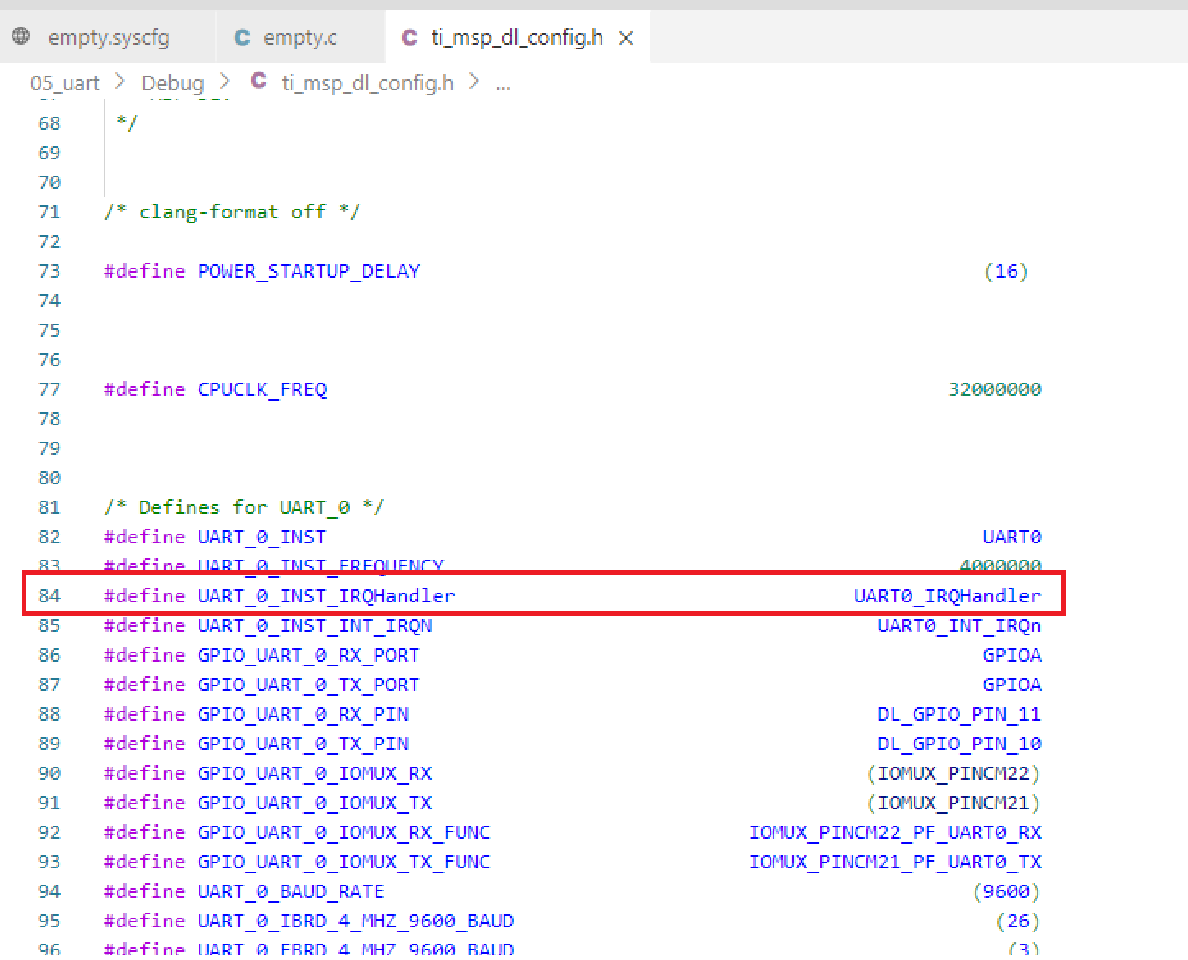
UART Peripheral	Any(UART0)
RX Pin	PA11/57
TX Pin	PA10/56

Use the shortcut keys Ctrl+S to save the configuration in the .syscfg file.

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 receiving interrupt of the serial port, when the serial port receives data, an interrupt will be triggered, and the interrupt service function will be executed when the interrupt is triggered. 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 is **UART0_IRQHandler**.

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 after compilation.



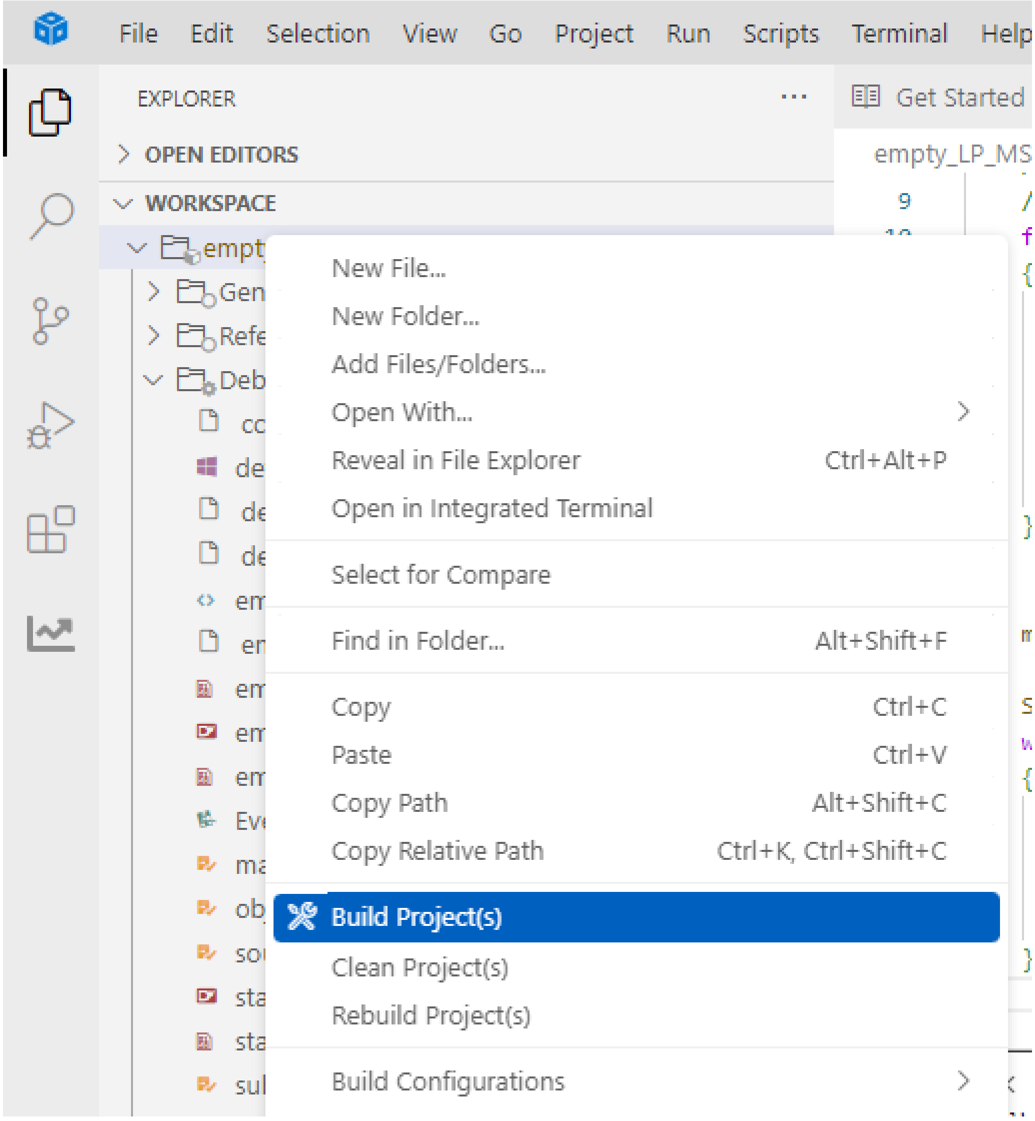
```
 */


/* clang-format off */

#define POWER_STARTUP_DELAY                                                 (16)



#define CPUCLK_FREQ                                                     32000000



/* Defines for UART_0 */
#define UART_0_INST                                                        UART0
#define UART_0_INST_FREQUENCY                                            4000000
#define UART_0_INST_IRQHandler                                  UART0_IRQHandler
#define UART_0_INST_INT_IRQN                                      UART0_INT_IRQn
#define GPIO_UART_0_RX_PORT                                                GPIOA
#define GPIO_UART_0_TX_PORT                                                GPIOA
#define GPIO_UART_0_RX_PIN                                         DL_GPIO_PIN_11
#define GPIO_UART_0_TX_PIN                                         DL_GPIO_PIN_10
#define GPIO_UART_0_IOMUX_RX                                      (IOMUX_PINCM22)
#define GPIO_UART_0_IOMUX_TX                                      (IOMUX_PINCM21)
#define GPIO_UART_0_IOMUX_RX_FUNC                       IOMUX_PINCM22_PF_UART0_RX
#define GPIO_UART_0_IOMUX_TX_FUNC                       IOMUX_PINCM21_PF_UART0_TX
#define UART_0_BAUD_RATE                                                  (9600)
#define UART_0_IBRD_4_MHZ_9600_BAUD                                         (26)
#define UART_0_FBRD_4_MHZ_9600_BAUD                                          (3)
```

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



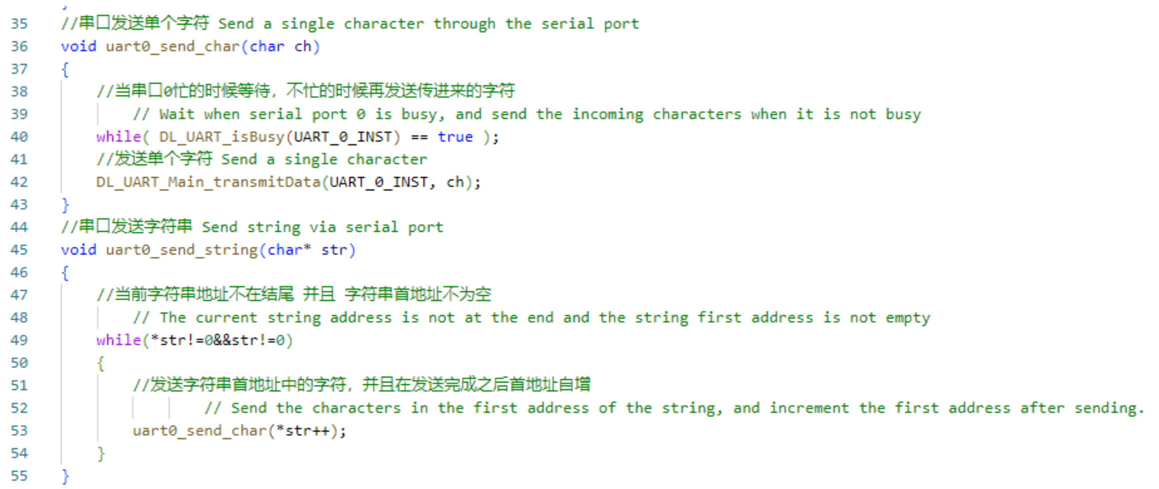
If the compilation is successful, you can download the program to the development board.

4. Program Analysis

- empty.c

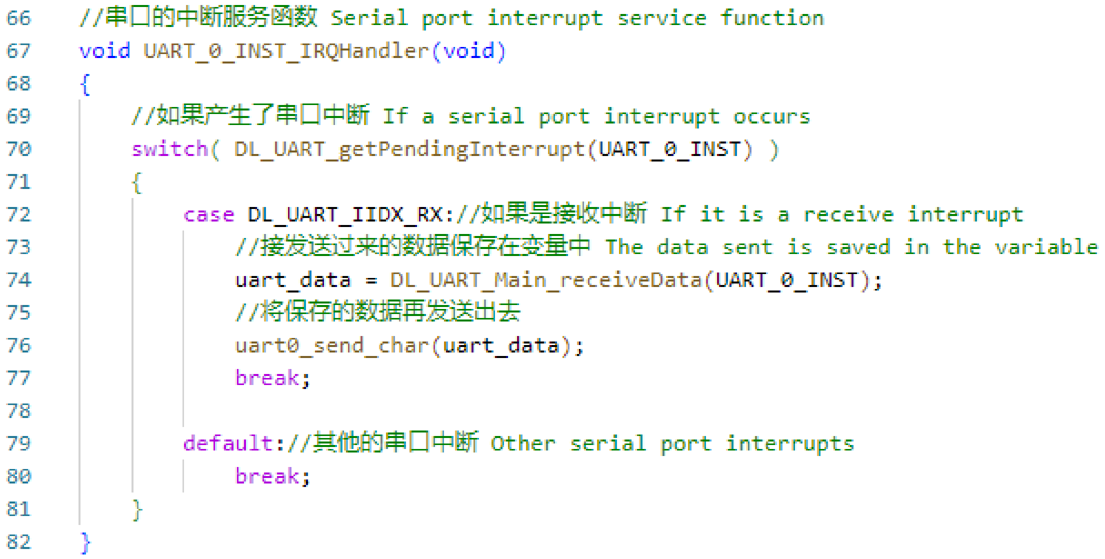
```c
int main(void)
{
    SYSCFG_DL_init();
    //清除串口中断标志 Clear the serial port interrupt flag
    NVIC_ClearPendingIRQ(UART_0_INST_INT_IRQN);
    //使能串口中断 Enable serial port interrupt
    NVIC_EnableIRQ(UART_0_INST_INT_IRQN);

    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);
    }
}
```

Serial port interrupts 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 automatically enter 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, `uart0_send_char` function sends a single character through the serial port, and `uart0_send_string` function sends a string through the serial port.

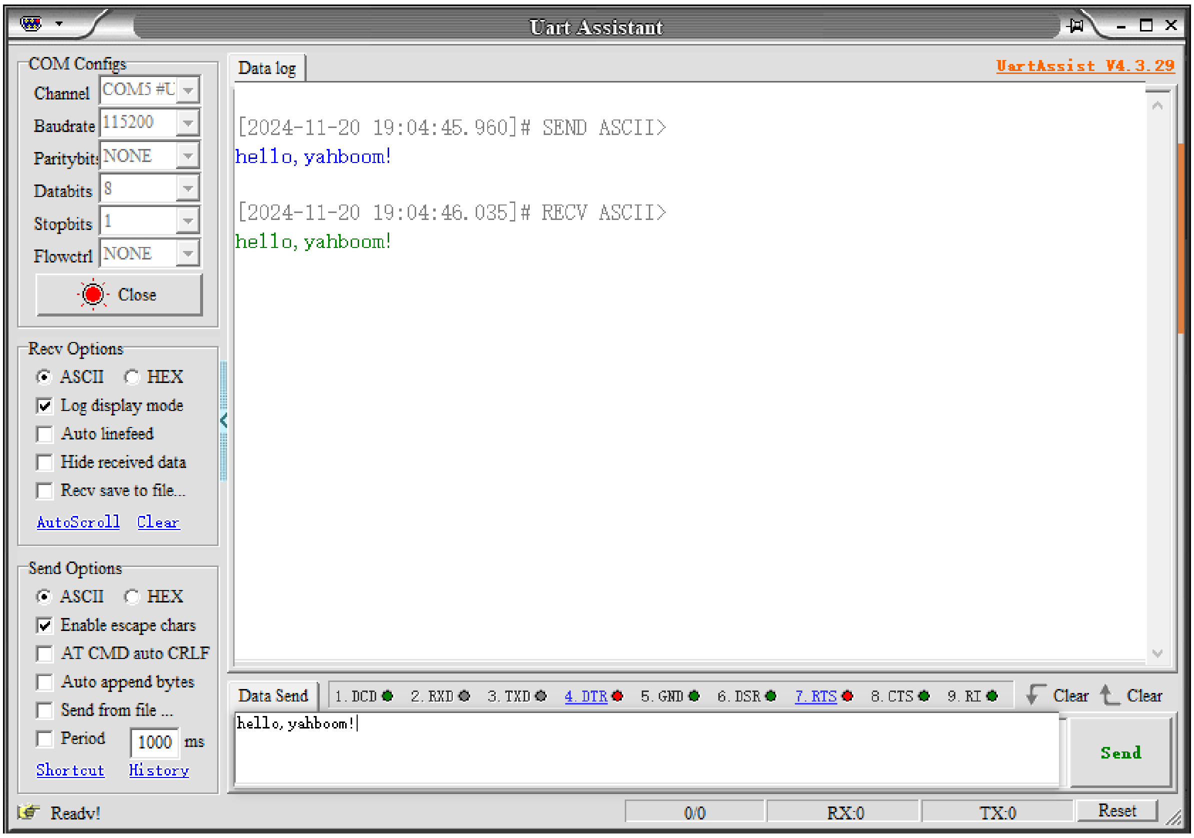


```c
//串口的中断服务函数 Serial port interrupt service function
void UART_0_INST_IRQHandler(void)
{
    //如果产生了串口中断 If a serial port interrupt occurs
    switch( DL_UART_getPendingInterrupt(UART_0_INST) )
    {
        case DL_UART_IIDX_RX://如果是接收中断 If it is a receive interrupt
            //接发送过来的数据保存在变量中 The 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
            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 interrupt type of the current 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.



COM Configs
Channel COM5 #U
Baudrate 115200
Paritybits NONE
Databits 8
Stopbits 1
Flowctrl NONE

Recv Options
● ASCII ○ HEX
☑ Log display mode
□ Auto linefeed
□ Hide received data
□ Recv save to file...

Send Options
● ASCII ○ HEX
☑ Enable escape chars
□ AT CMD auto CRLF
□ Auto append bytes
□ Send from file ...
□ Period 1000 ms

```
[2024-11-20 19:04:45.960]# SEND ASCII>
hello,yahboom!

[2024-11-20 19:04:46.035]# RECV ASCII>
hello,yahboom!
```

hello,yahboom!

Readv! 0/0 RX:0 TX:0