Get MPU6050 data (I2C)

1. Learning objectives

1. Learn the basic knowledge of IIC communication.
2. Get MPU6050 data.

Introduction to I2C

The IIC bus is a bidirectional two-wire serial bus that provides communication lines between integrated circuits. It means a protocol that completes information exchange between integrated circuits or functional units.

The IIC module receives and sends data and converts data from serial to parallel or from parallel to serial. Interrupts can be enabled or disabled. The interface is connected to the IIC bus through the data pin (SDA) and the clock pin (SCL). It allows connection to a standard (up to 100kHz) or fast (up to 400kHz) IIC bus. (The data line SDA and the clock SCL constitute a serial bus that can send and receive data).

There are three types of signals in the IIC bus during data transmission, namely: start signal (START), stop (end) signal (STOP), and acknowledgement signal (ACK). Secondly, it is in an idle state when no data transmission is performed.

Basic parameters of I2C

Rate: The I2C bus has two transmission modes: standard mode (100 kbit/s) and fast mode (400 kbit/s), and there are also faster extended mode and high-speed mode to choose from.

Device address: Each device has a unique 7-bit or 10-bit address, and the address selection can be used to determine who to communicate with.

Bus state: The I2C bus has five states, namely idle state, start signal, end signal, response signal, and data transmission.

Data format: The I2C bus has two data formats, standard format and fast format. The standard format is an 8-bit data byte plus a 1-bit ack/nack (acknowledgement/non-acknowledgement) bit, and the fast format allows two bytes to be transmitted simultaneously.

Since the SCL and SDA lines are bidirectional, they may also have level errors due to external reasons (such as capacitance in the line), which may cause communication errors. Therefore, in the IIC bus, pull-up resistors are usually used to ensure that the signal line is at a high level in the idle state.

2. Hardware Construction

The I2C of the MSPM0G series supports master-slave mode, has 7 address bits that can be set, supports I2C standard transmission rates of 100kbps, 400kbps, and 1Mbps, and supports SMBUS. Whether it is a master or a slave, there are independent 8-byte FIFOs for sending and receiving. MSPM0 I2C has 8-byte FIFOs, generates independent interrupts for controller and target modes, and supports DMA.

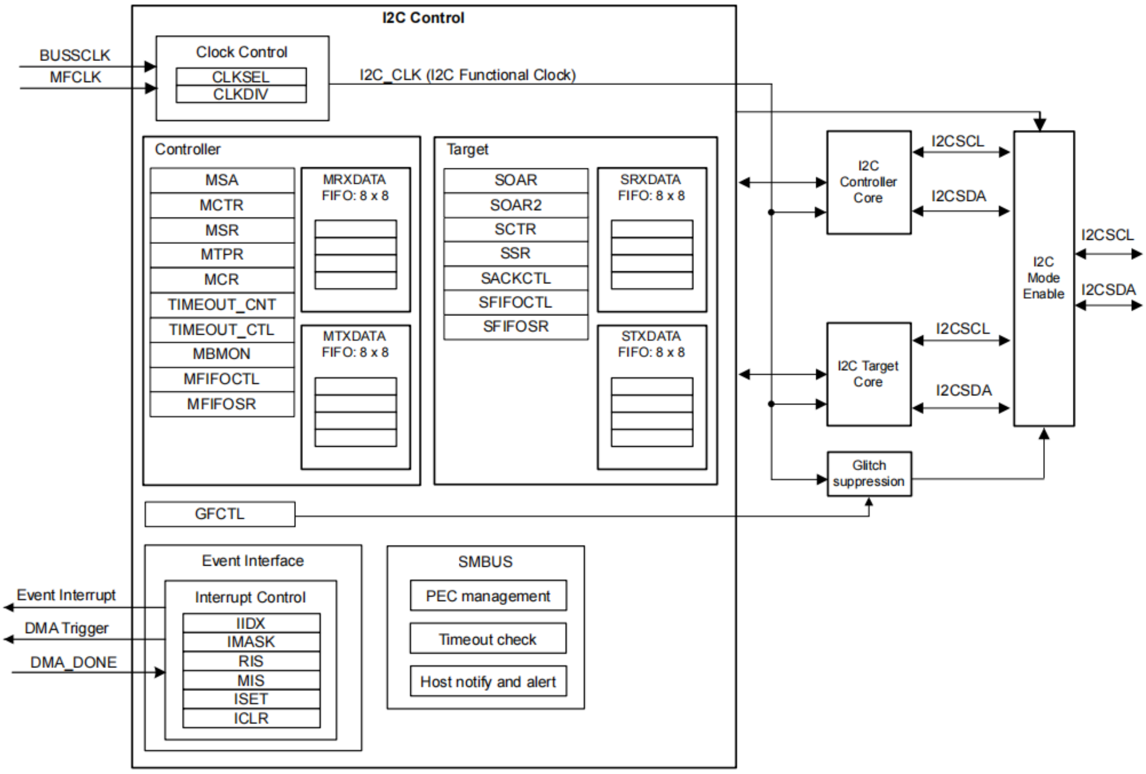


Figure 18-1. I2C Functional Block Diagram

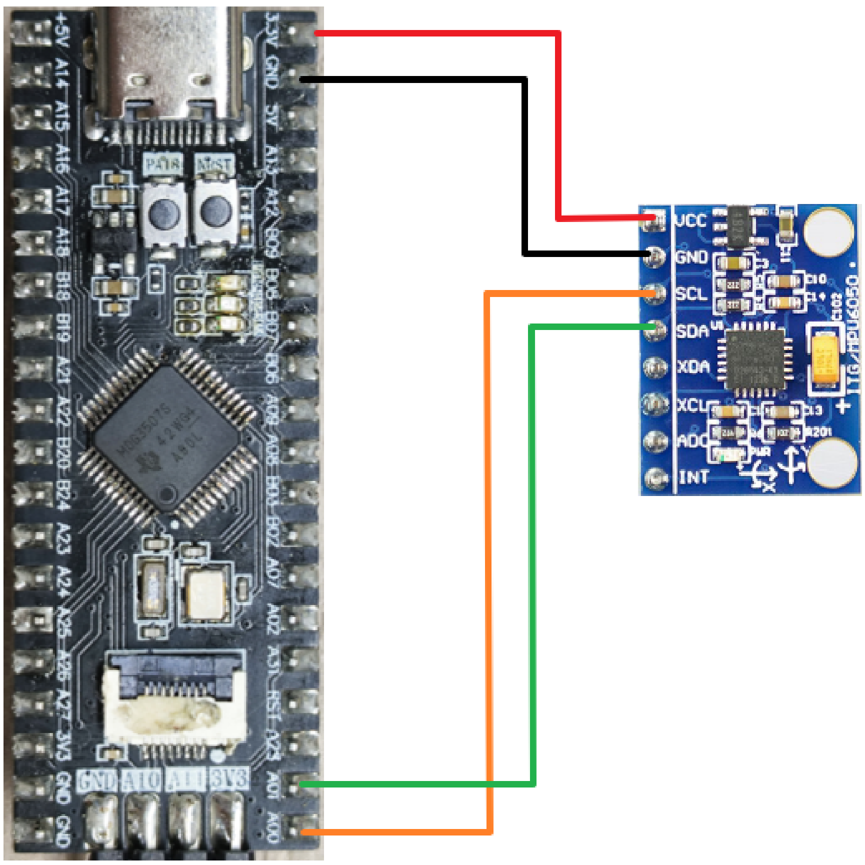
Software I2C refers to implementing the I2C communication protocol by writing code in the program. It uses general-purpose input and output (GPIO) pins to simulate the data line (SDA) and clock line (SCL) of I2C, and transmits data and generates timing signals by controlling the level changes of the pins through software. Compared with hardware I2C, the advantage of software I2C is that it does not require specific hardware support and can be implemented on any microcontroller that supports GPIO functions. It uses the general IO pins of the microcontroller to implement the I2C communication protocol.

Hardware I2C refers to processing the I2C communication protocol through a dedicated hardware module. Most modern microcontrollers and some external devices have integrated hardware I2C modules, which are responsible for handling the details of I2C communication, including generating correct timing signals, automatically handling signal conflicts, data transmission and error detection, etc. You can directly use the hardware pin connection without writing timing code.

This experiment uses software IIC to read the data of the MPU6050 module.

Hardware connection

MSPM0G3507	MPU6050
PA0	SCL
PA1	SDA
3V3	GND
GND	VCC

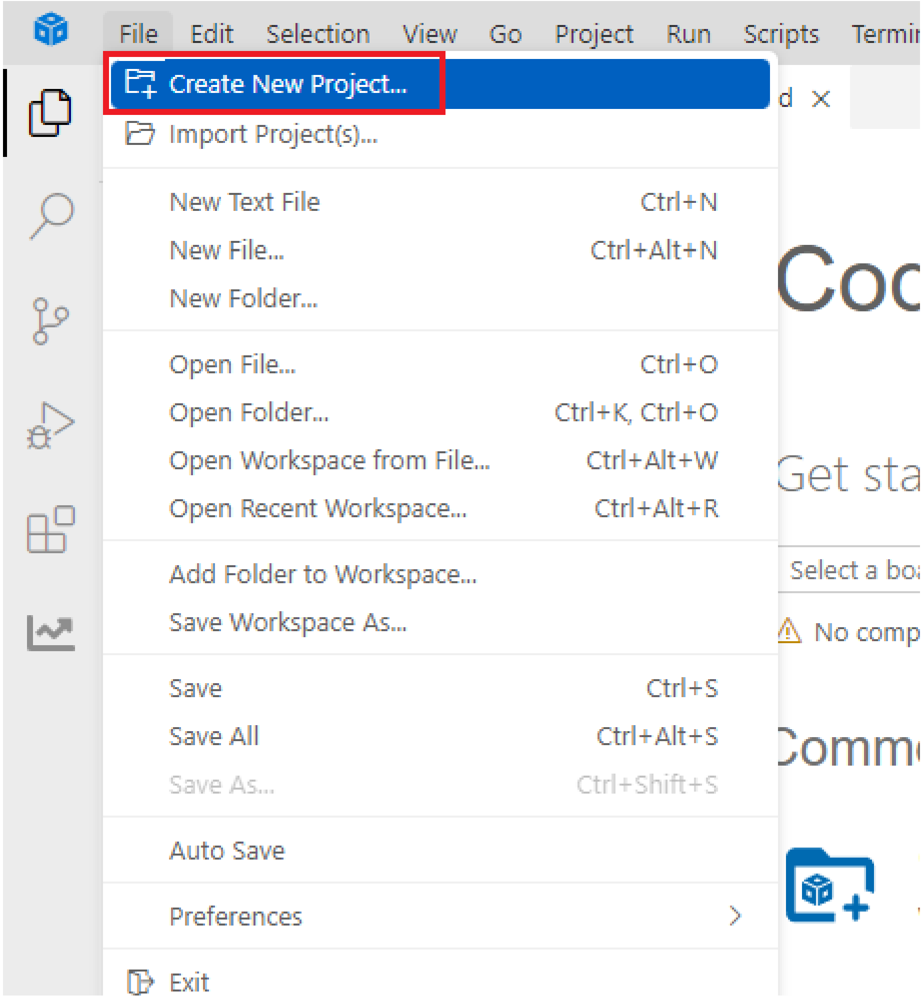


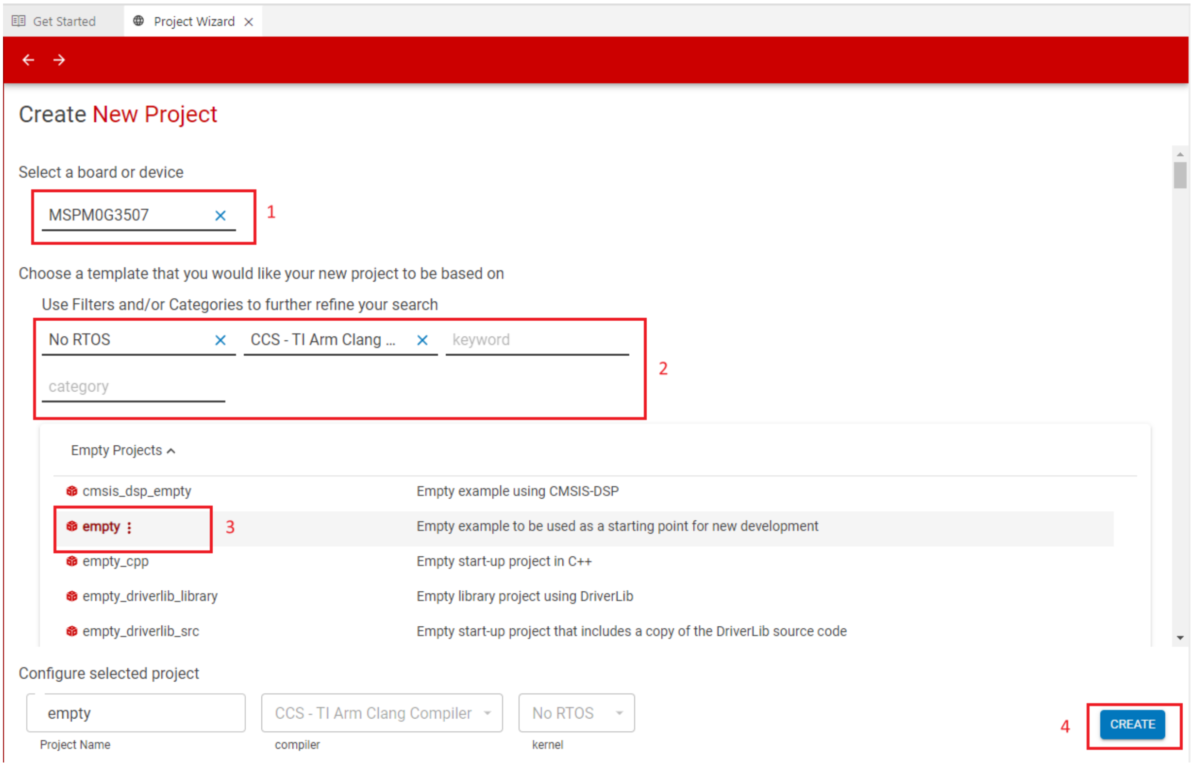
3. Experimental steps

This course configures the PA0 and PA1 pins as SCL and SDA to read the data of the MPU6050 six-axis sensor module.

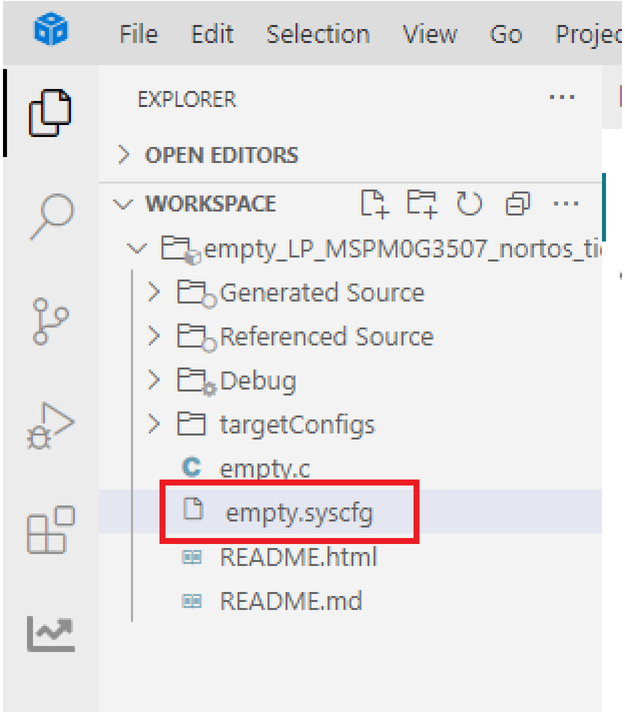
1. Open the SYSCONFIG configuration tool

Create a blank project in CCS.





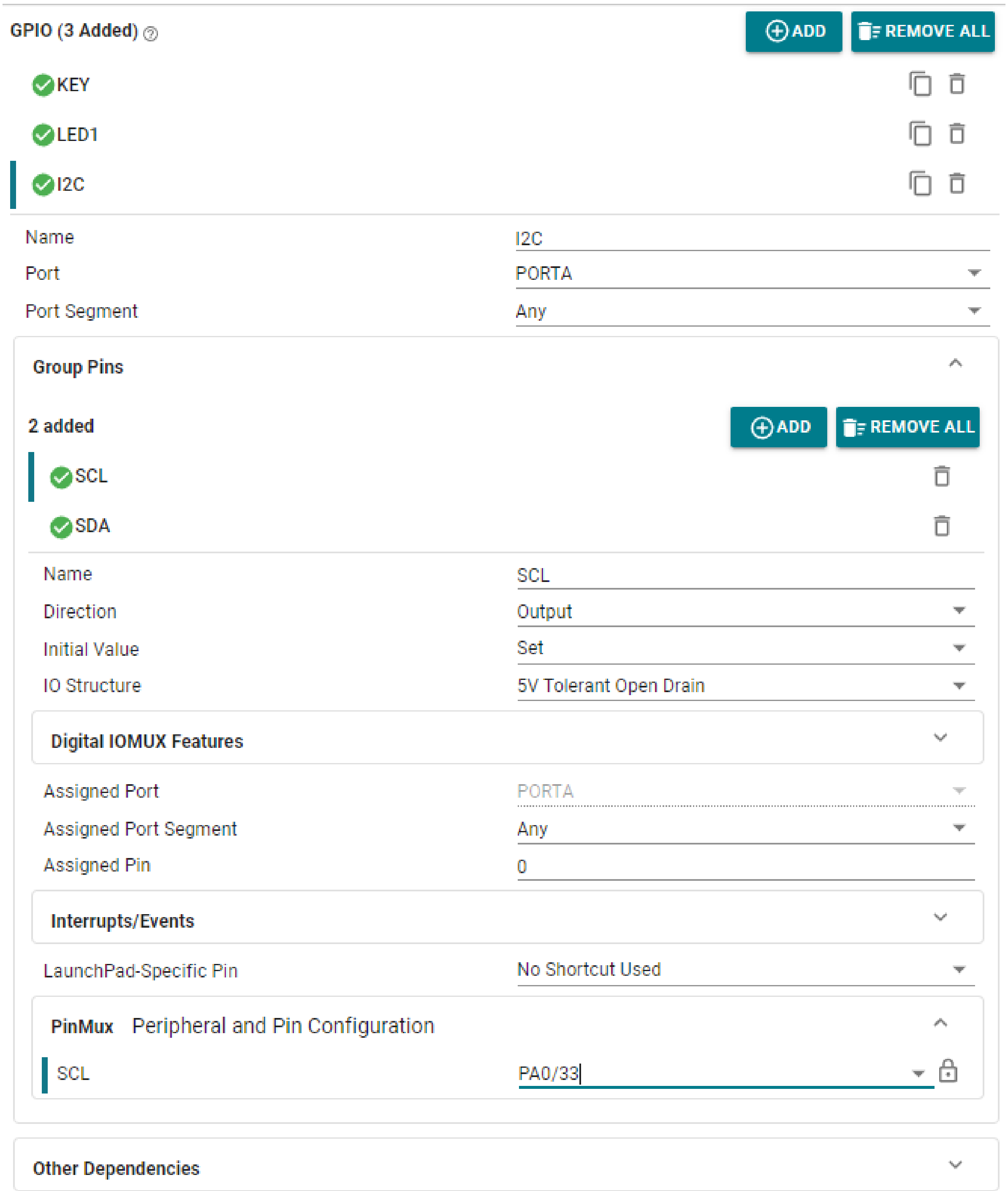
Find and open the empty.syscfg file in the left workspace of CCS.



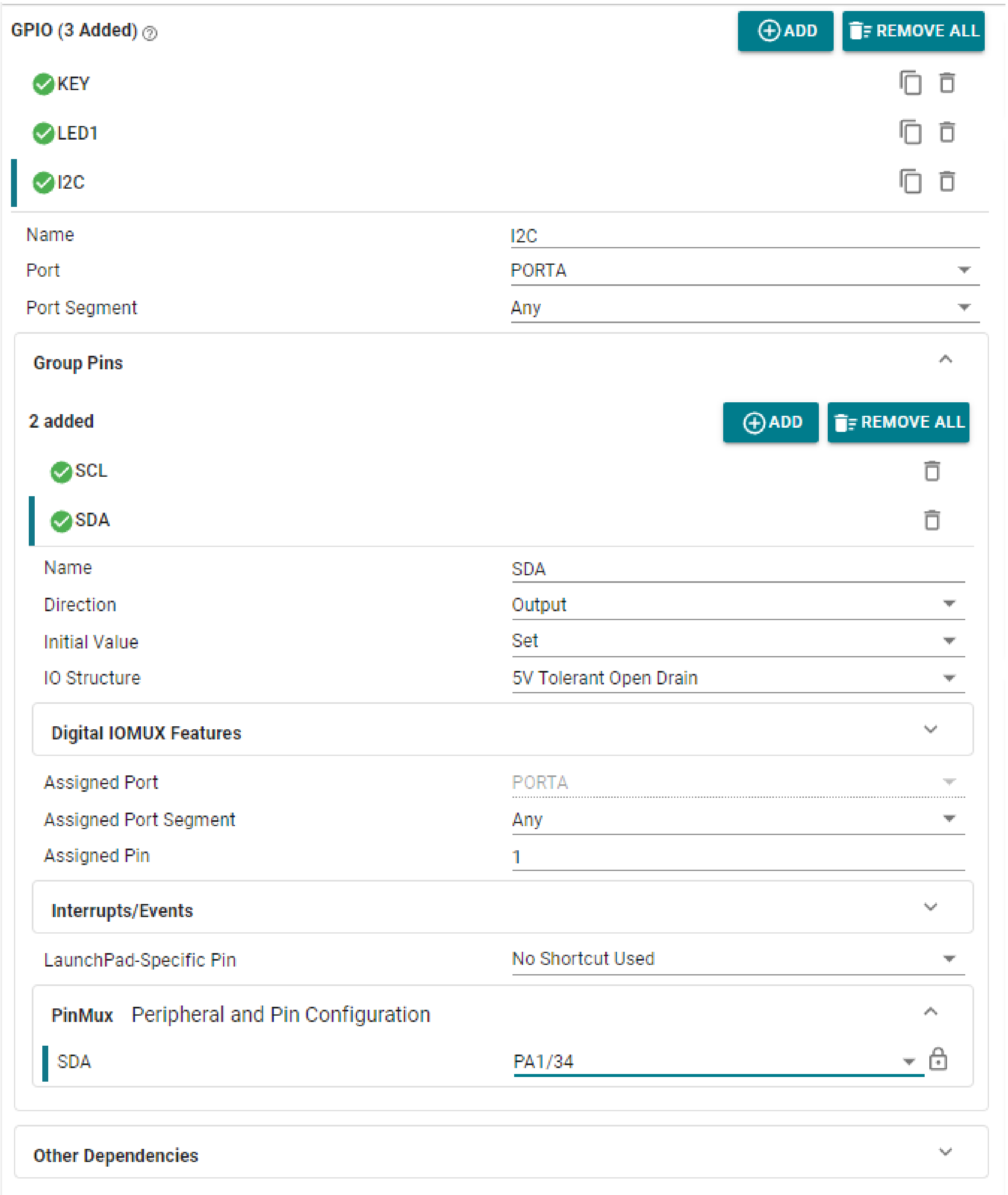
2. Pin parameter configuration

For the pin configuration of LED lights and buttons, please refer to the previous course. Here, only the I2C related pin configuration is posted

SCL:

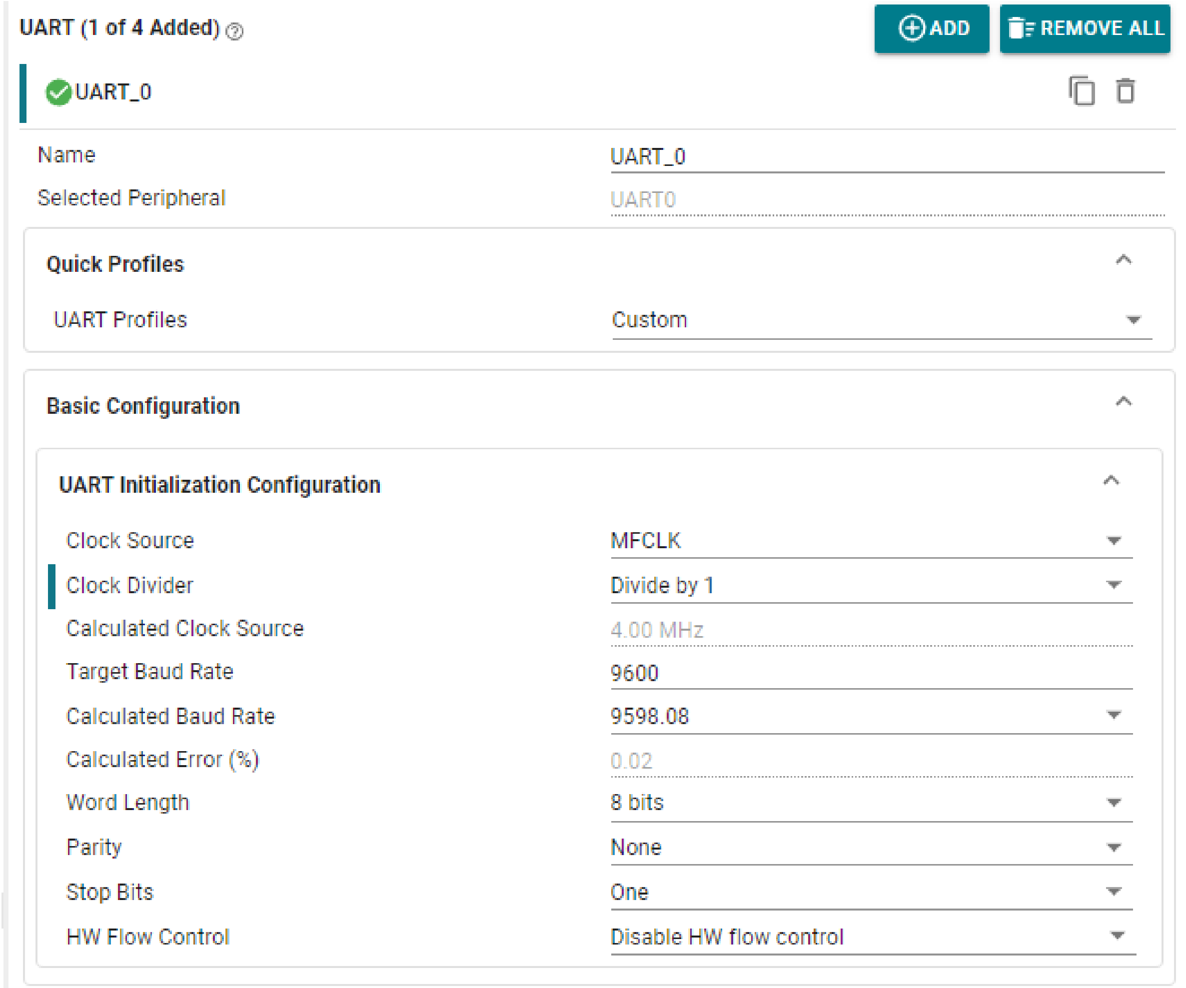

GPIO (3 Added)
ADD
REMOVE ALL
KEY
LED1
I2C
Name
I2C
Port
PORTA
Port Segment
Any
Group Pins
2 added
ADD
REMOVE ALL
SCL
SDA
Name
SCL
Direction
Output
Initial Value
Set
IO Structure
5V Tolerant Open Drain
Digital IOMUX Features
Assigned Port
PORTA
Assigned Port Segment
Any
Assigned Pin
0
Interrupts/Events
LaunchPad-Specific Pin
No Shortcut Used
PinMux Peripheral and Pin Configuration
SCL
PA0/33
Other Dependencies

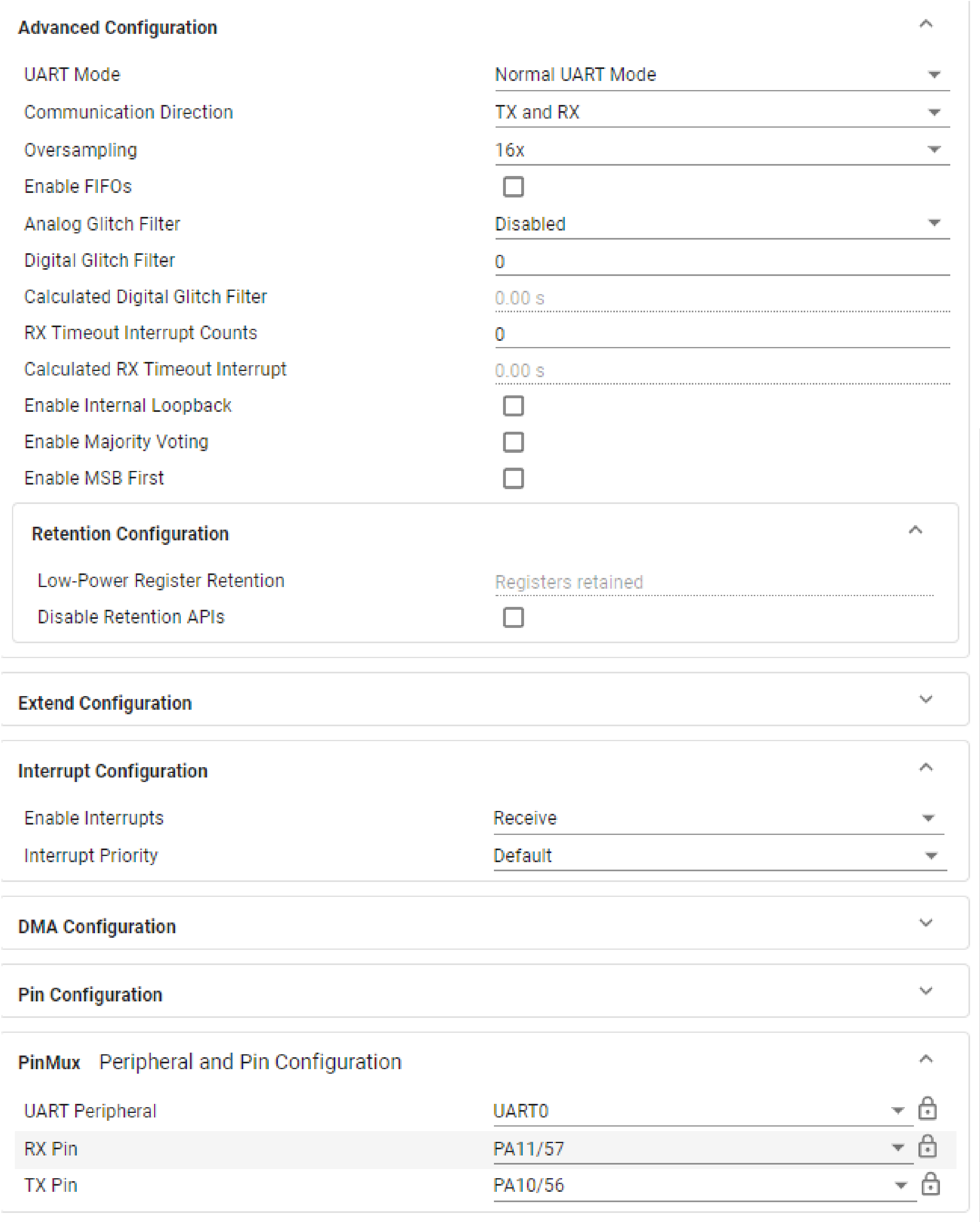
SDA:



3. Serial port parameter configuration

The undisplayed part is the default option.

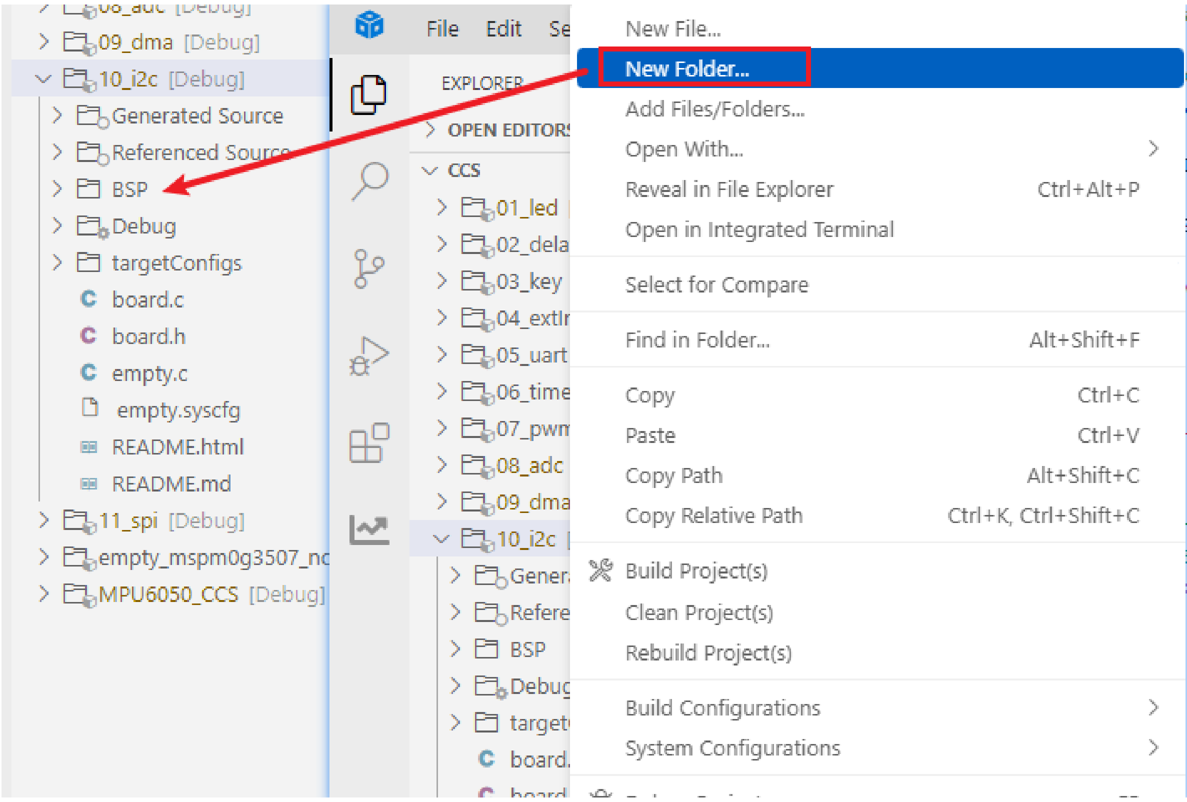

UART (1 of 4 Added)
ADD
REMOVE ALL
UART_0
Name
UART_0
Selected Peripheral
UART0
Quick Profiles
UART Profiles
Custom
Basic Configuration
UART Initialization Configuration
Clock Source
MFCLK
Clock Divider
Divide by 1
Calculated Clock Source
4.00 MHz
Target Baud Rate
9600
Calculated Baud Rate
9598.08
Calculated Error (%)
0.02
Word Length
8 bits
Parity
None
Stop Bits
One
HW Flow Control
Disable HW flow control



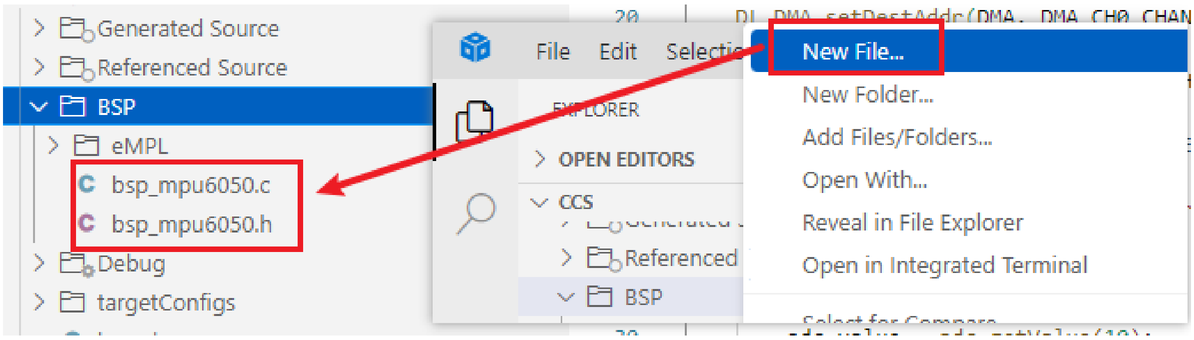
Use the shortcut key Ctrl+S to save the configuration in the .syscfg file.

4. Use of I2C protocol

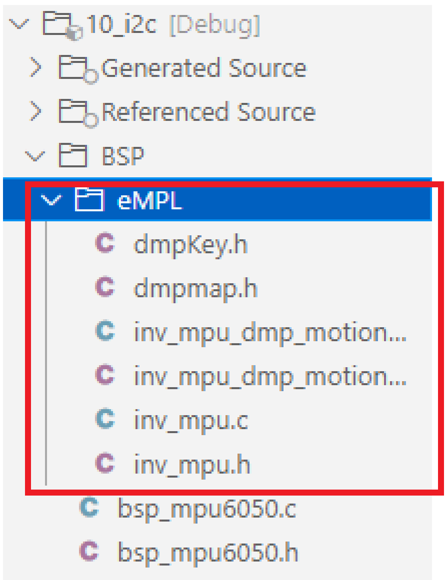
We create a new folder in the project folder: `BSP`.



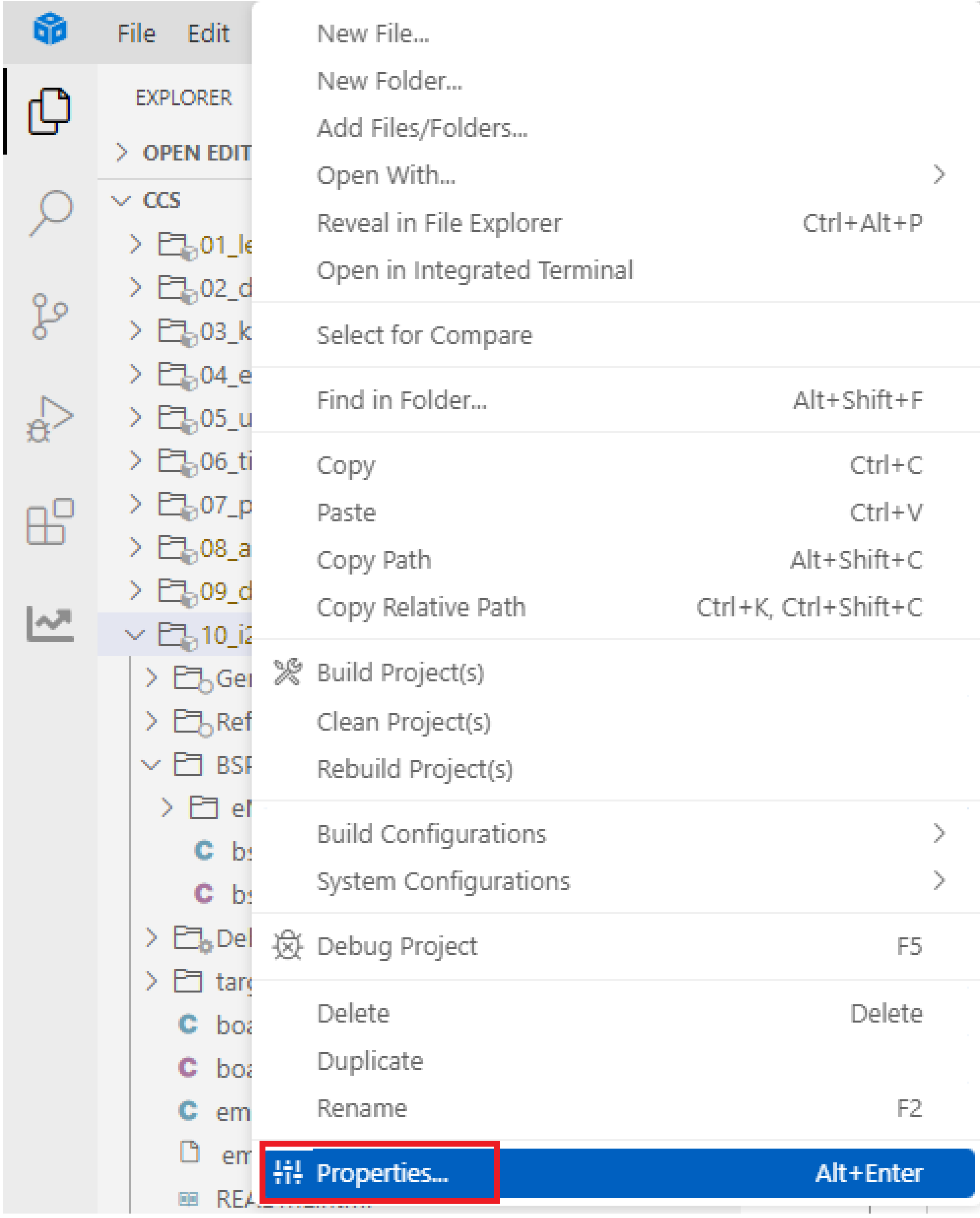
Create two more files in the BSP folder, namely `bsp_mpu6050.c` and `bsp_mpu6050.h`.

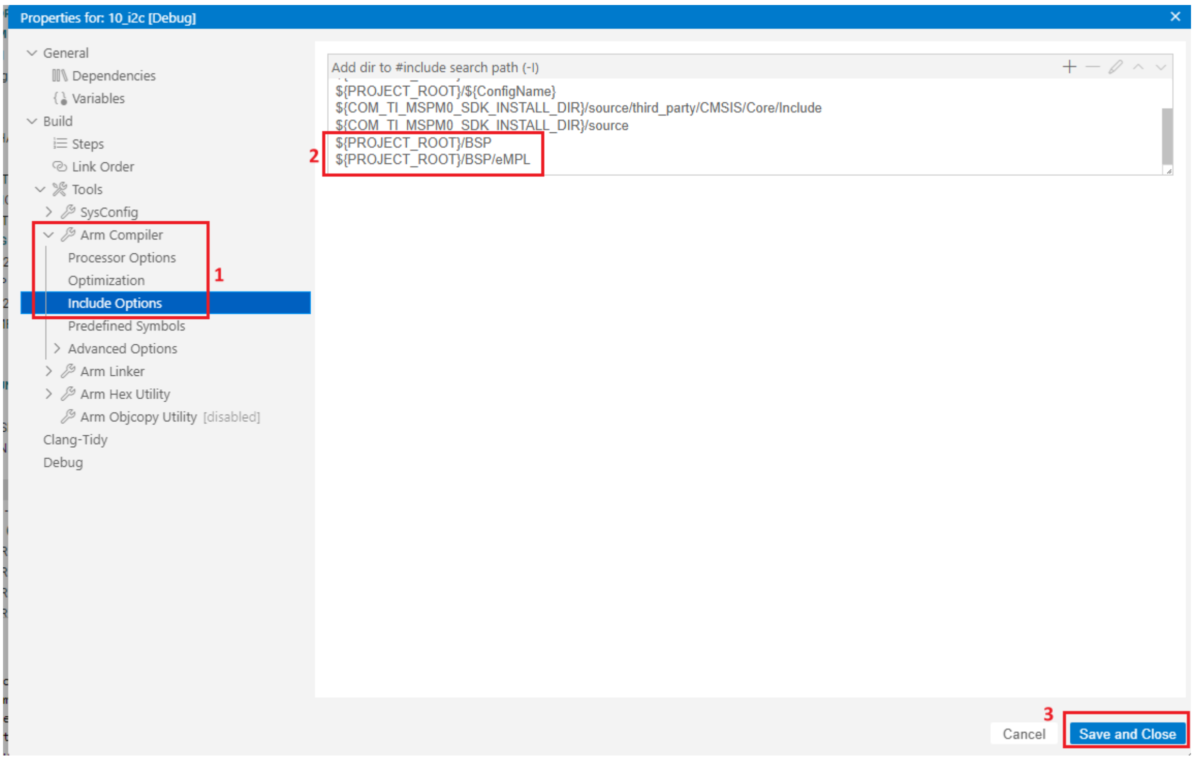


Then put the drivers designed for the MPU6050 gyroscope and accelerometer into the newly created eMPL folder.



Update the header file path and right-click the project folder.





`${PROJECT_ROOT}/BSP` Path Description

`${PROJECT_ROOT}` indicates the current project path.

`/BSP` indicates the Hardware folder under the current project path.

5. Write the program

When using software to implement I2C communication, you need to select appropriate pins as the data line (SDA) and clock line (SCL). Generally, any programmable general-purpose input and output (GPIO) pins can be selected as software I2C pins. For software I2C, at least two pins are required for the data line (SDA) and the clock line (SCL), and ensure that these pins can meet the timing requirements of the I2C communication protocol. The following is a general pin description:

1. Data line (SDA): The pin used to transmit data. In software I2C, this pin needs to be set to output mode (for the master device to send data) and input mode (for the master device to receive data). During communication, data transmission needs to be achieved by controlling the level change of the data line.
2. Clock line (SCL): The pin used to control the clock signal for data transmission. In software I2C, this pin needs to be set to output mode, and the clock pulse is generated by controlling the level change of the clock line to control the transmission of the data line. It should be noted that the following aspects should be considered when selecting the appropriate pin:

- Support input/output configuration: The pin needs to support configuration as input or output mode in software and be able to switch dynamically through the program.
- Hardware restrictions and conflicts: Make sure that the selected pin is not assigned to other hardware functions or peripherals to avoid conflicts.
- Electrical characteristics: The electrical characteristics of the pins should meet the standard requirements of the I2C bus, such as the correct level and driving capability.

It should be noted that the implementation of software I2C requires more program code and calculations. Compared with hardware I2C, software I2C is more sensitive to processor performance and timing control. Therefore, when selecting pins, the performance and programmability of the processor also need to be considered. In order to ensure the maintainability and portability of the code, the relevant functions are defined here.

The macro definitions of the SDA pin and the SCL pin are as follows:

bsp_mpu6050.h

```
#ifndef _BSP_MPU6050_H_
#define _BSP_MPU6050_H_

#include "board.h"


//设置SDA输出模式 Set SDA output mode
#define SDA_OUT()   {                                          \
                        DL_GPIO_initDigitalOutput(I2C_SDA_IOMUX);     \
                        DL_GPIO_setPins(I2C_PORT, I2C_SDA_PIN);      \
                        DL_GPIO_enableOutput(I2C_PORT, I2C_SDA_PIN); \
                    }
//设置SDA输入模式 Set SDA input mode
#define SDA_IN()    { DL_GPIO_initDigitalInput(I2C_SDA_IOMUX); }

//获取SDA引脚的电平变化 Get the level change of the SDA pin
#define SDA_GET()   ( ( ( DL_GPIO_readPins(I2C_PORT,I2C_SDA_PIN) & I2C_SDA_PIN )
> 0 ) ? 1 : 0 )
```

```
//SDA与SCL输出 SDA and SCL output
#define SDA(x)      ( (x) ? (DL_GPIO_setPins(I2C_PORT,I2C_SDA_PIN)) :
(DL_GPIO_clearPins(I2C_PORT,I2C_SDA_PIN)) )
#define SCL(x)      ( (x) ? (DL_GPIO_setPins(I2C_PORT,I2C_SCL_PIN)) :
(DL_GPIO_clearPins(I2C_PORT,I2C_SCL_PIN)) )


//MPU6050的AD0是IIC地址引脚，接地则IIC地址为0x68,接VCC则IIC地址为0x69
//AD0 of MPU6050 is the IIC address pin. If it is grounded, the IIC address is
0x68. If it is connected to VCC, the IIC address is 0x69.


#define MPU6050_RA_SMPLRT_DIV       0x19        //陀螺仪采样率 地址  Gyro sampling
rate address
#define MPU6050_RA_CONFIG           0x1A        //设置数字低通滤波器 地址 Set
digital low-pass filter address
#define MPU6050_RA_GYRO_CONFIG      0x1B        //陀螺仪配置寄存器 Gyro
configuration register
#define MPU6050_RA_ACCEL_CONFIG     0x1C        //加速度传感器配置寄存器
Acceleration sensor configuration register
#define MPU_INT_EN_REG              0X38        //中断使能寄存器 Interrupt enable
register
#define MPU_USER_CTRL_REG           0X6A        //用户控制寄存器 User control
register
#define MPU_FIFO_EN_REG             0X23        //FIFO使能寄存器 FIFO enable
register
#define MPU_PWR_MGMT2_REG           0X6C        //电源管理寄存器2 Power management
register 2
#define MPU_GYRO_CFG_REG            0X1B        //陀螺仪配置寄存器 Gyroscope
configuration register
#define MPU_ACCEL_CFG_REG           0X1C        //加速度计配置寄存器 Accelerometer
configuration register
#define MPU_CFG_REG                 0X1A        //配置寄存器 Configuration
register
#define MPU_SAMPLE_RATE_REG         0X19        //采样频率分频器 Sampling frequency
divider
#define MPU_INTBP_CFG_REG           0X37        //中断/旁路设置寄存器
Interrupt/bypass setting register

#define MPU6050_RA_PWR_MGMT_1       0x6B
#define MPU6050_RA_PWR_MGMT_2       0x6C

#define MPU6050_WHO_AM_I            0x75
#define MPU6050_SMPLRT_DIV          0            //8000Hz
#define MPU6050_DLPF_CFG            0
#define MPU6050_GYRO_OUT            0x43         //MPU6050陀螺仪数据寄存器地址
MPU6050 gyroscope data register address
#define MPU6050_ACC_OUT             0x3B         //MPU6050加速度数据寄存器地址
MPU6050 acceleration data register address

#define MPU6050_RA_TEMP_OUT_H       0x41        //温度高位 High temperature
#define MPU6050_RA_TEMP_OUT_L       0x42        //温度低位 Low temperature

#define MPU_ACCEL_XOUTH_REG         0X3B        //加速度值,X轴高8位寄存器
Acceleration value, X-axis high 8-bit register
#define MPU_ACCEL_XOUTL_REG         0X3C        //加速度值,X轴低8位寄存器
Acceleration value, X-axis low 8-bit register
```

```
#define MPU_ACCEL_YOUTH_REG         0X3D        //加速度值,Y轴高8位寄存器
Acceleration value, Y-axis high 8-bit register
#define MPU_ACCEL_YOUTL_REG         0X3E        //加速度值,Y轴低8位寄存器
Acceleration value, Y-axis low 8-bit register
#define MPU_ACCEL_ZOUTH_REG         0X3F        //加速度值,Z轴高8位寄存器
Acceleration value, Z-axis high 8-bit register
#define MPU_ACCEL_ZOUTL_REG         0X40        //加速度值,Z轴低8位寄存器
Acceleration value, Z-axis low 8-bit register

#define MPU_TEMP_OUTH_REG           0X41        //温度值高八位寄存器 Temperature
value high eight bits register
#define MPU_TEMP_OUTL_REG           0X42        //温度值低8位寄存器 Temperature
value lower 8 bits register

#define MPU_GYRO_XOUTH_REG          0X43        //陀螺仪值,X轴高8位寄存器 Gyroscope
value, X-axis high 8-bit register
#define MPU_GYRO_XOUTL_REG          0X44        //陀螺仪值,X轴低8位寄存器 Gyroscope
value, X-axis low 8-bit register
#define MPU_GYRO_YOUTH_REG          0X45        //陀螺仪值,Y轴高8位寄存器 Gyroscope
value, Y-axis high 8-bit register
#define MPU_GYRO_YOUTL_REG          0X46        //陀螺仪值,Y轴低8位寄存器 Gyroscope
value, Y-axis low 8-bit register
#define MPU_GYRO_ZOUTH_REG          0X47        //陀螺仪值,Z轴高8位寄存器 Gyroscope
value, Z-axis high 8-bit register
#define MPU_GYRO_ZOUTL_REG          0X48        //陀螺仪值,Z轴低8位寄存器 Gyroscope
value, Z-axis low 8-bit register


char MPU6050_WriteReg(uint8_t addr,uint8_t regaddr,uint8_t num,uint8_t
*regdata);
char MPU6050_ReadData(uint8_t addr, uint8_t regaddr,uint8_t num,uint8_t* Read);

char MPU6050_Init(void);
void MPU6050ReadGyro(short *gyroData);
void MPU6050ReadAcc(short *accData);
float MPU6050_GetTemp(void);
uint8_t MPU6050ReadID(void);
#endif
```

The next step is to configure the timing part of I2C,

bsp_mpu6050.c (only part is intercepted here, please check the project source code for details)

```
#include "bsp_mpu6050.h"
#include "stdio.h"


/******************************************************************
 * 函 数 名 称：IIC_Start
 * 函 数 说 明：IIC起始时序
 * 函 数 形 参：无
 * 函 数 返 回：无
 * 作       者：LC
 * 备       注：无
 * Function name: IIC_Start
 * Function description: IIC start timing
 * Function parameter: None
```

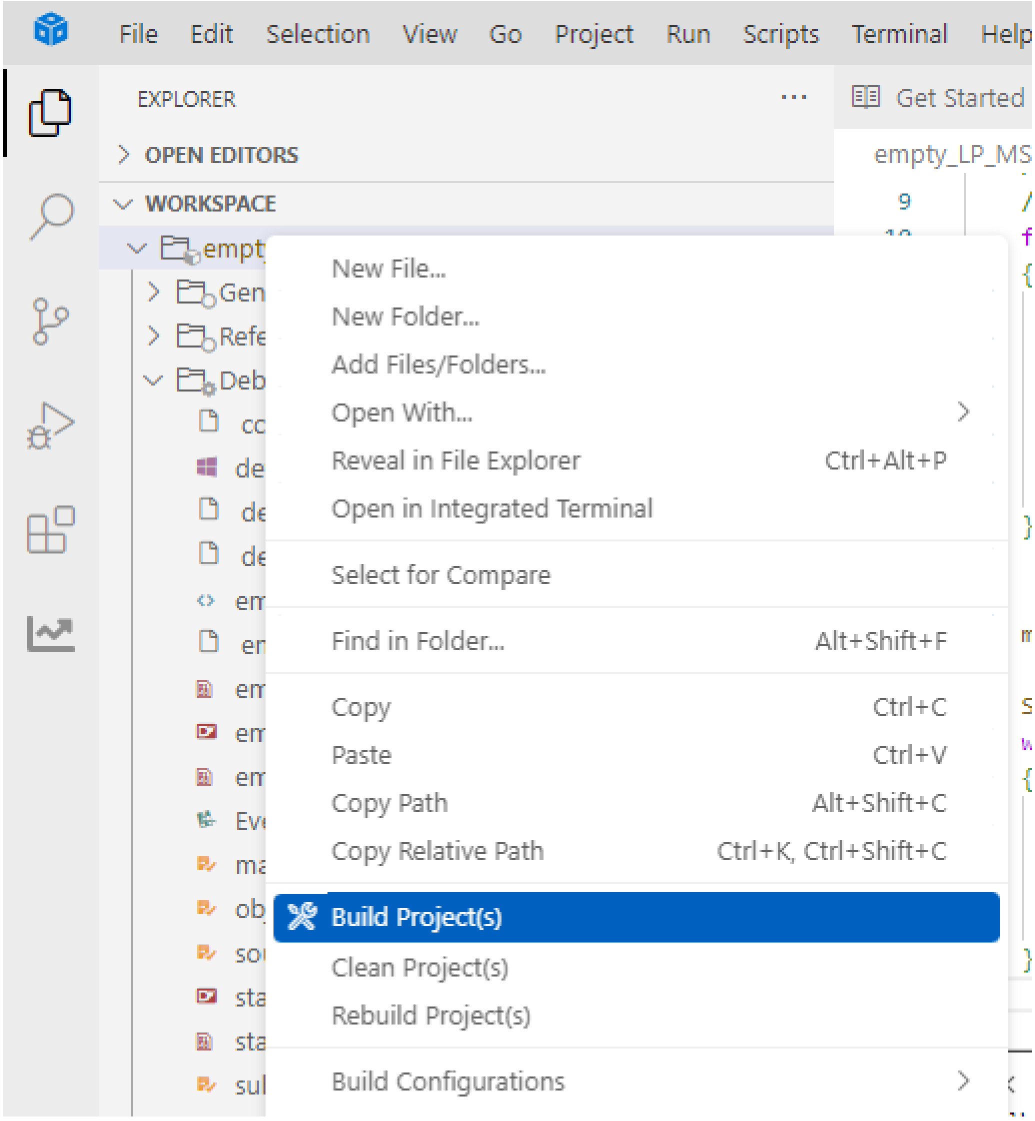
```
 * Function return: None
 * Author: LC
 * Remarks: None
******************************************************************/
void IIC_Start(void)
{
        SDA_OUT();
        SCL(1);
        SDA(0);

        SDA(1);
        delay_us(5);
        SDA(0);
        delay_us(5);

        SCL(0);
}
/******************************************************************
 * 函 数 名 称：IIC_Stop
 * 函 数 说 明：IIC停止信号
 * 函 数 形 参：无
 * 函 数 返 回：无
 * 作       者：LC
 * 备       注：无
 * Function name: IIC_Stop
 * Function description: IIC stop signal
 * Function parameters: None
 * Function return: None
 * Author: LC
 * Notes: None
******************************************************************/
void IIC_Stop(void)
{
        SDA_OUT();
        SCL(0);
        SDA(0);

        SCL(1);
        delay_us(5);
        SDA(1);
        delay_us(5);

}

...
```

Then write the following code in the empty.c file

```
#include "board.h"
#include <stdio.h>
#include "bsp_mpu6050.h"
#include "inv_mpu.h"

char buf[100];  // 定义缓冲区 Defining buffers

int main(void)
{
```

```c
    // 开发板初始化 Development board initialization
    board_init();

    // 初始化 MPU6050 Initialize MPU6050
    MPU6050_Init();

    uint8_t ret = 1;
    float pitch = 0, roll = 0, yaw = 0;  // 欧拉角 Euler Angles

    printf("start\r\n");

    // DMP 初始化 DMP initialization
    while (mpu_dmp_init())
    {
        printf("DMP error\r\n");
        delay_ms(200);
    }

    printf("Initialization Data Succeed \r\n");

    while (1)
    {
        // 获取欧拉角 Get Euler angles
        if (mpu_dmp_get_data(&pitch, &roll, &yaw) == 0)
        {
            // 格式化数据并发送 Format data and send
            sprintf(buf, "pitch = %d, roll = %d, yaw = %d\n", (int)pitch,
(int)roll, (int)yaw);
            uart0_send_string(buf);  // 串口发送 Serial port sending
        }
        else
        {
            // 获取数据失败 Failed to obtain data
            printf("Data get error\r\n");
        }

        delay_ms(200);  // 延时，根据实际采样率调整 Delay, adjusted according to the
actual sampling rate
    }

}
```

6. Compile



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

4. Program Analysis

- inv_mpu.c

```c
//得到dmp处理后的数据(注意,本函数需要比较多堆栈,局部变量有点多)
//pitch:俯仰角 精度:0.1°   范围:-90.0° <---> +90.0°
//roll:横滚角  精度:0.1°   范围:-180.0°<---> +180.0°
//yaw:航向角   精度:0.1°   范围:-180.0°<---> +180.0°
//返回值:0,正常
//    其他,失败
//Get the data after dmp processing (note that this function requires a lot of stacks and a lot of local variables)
//pitch: pitch angle    accuracy: 0.1°  range: -90.0° <---> +90.0°
//roll: roll angle      accuracy: 0.1°   range: -180.0° <---> +180.0°
//yaw: heading angle    accuracy: 0.1°   range: -180.0° <---> +180.0°
//Return value: 0, normal
//              Others, failed
u8 mpu_dmp_get_data(float *pitch,float *roll,float *yaw)
{
    float q0=1.0f,q1=0.0f,q2=0.0f,q3=0.0f;
    unsigned long sensor_timestamp;
    short gyro[3], accel[3], sensors;
    unsigned char more;
    long quat[4];
    if(dmp_read_fifo(gyro, accel, quat, &sensor_timestamp, &sensors,&more))return 1;
    /* Gyro and accel data are written to the FIFO by the DMP in chip frame and hardware units.
     * This behavior is convenient because it keeps the gyro and accel outputs of dmp_read_fifo and mpu_read_fifo consistent.
    **/
    /*if (sensors & INV_XYZ_GYRO )
    send_packet(PACKET_TYPE_GYRO, gyro);
    if (sensors & INV_XYZ_ACCEL)
    send_packet(PACKET_TYPE_ACCEL, accel); */
    /* Unlike gyro and accel, quaternions are written to the FIFO in the body frame, q30.
     * The orientation is set by the scalar passed to dmp_set_orientation during initialization.
    **/
    if(sensors&INV_WXYZ_QUAT)
    {
        q0 = quat[0] / q30; //q30格式转换为浮点数 Convert q30 format to floating point number
        q1 = quat[1] / q30;
        q2 = quat[2] / q30;
        q3 = quat[3] / q30;
        //计算得到俯仰角/横滚角/航向角
        // Calculate the pitch angle/roll angle/heading angle
        *pitch = asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; // pitch
        *roll  = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)* 57.3; // roll
        *yaw   = atan2(2*(q1*q2 + q0*q3),q0*q0+q1*q1-q2*q2-q3*q3) * 57.3;   //yaw
    }else return 2;
    return 0;
}
```

This function reads the sensor data (including gyroscope, accelerometer and quaternion) processed by DMP from the FIFO of MPU6050, calculates the pitch angle (pitch), roll angle (roll) and heading angle (yaw) by parsing the quaternion, and returns the result.

- empty.c

```c
char buf[100];  // 定义缓冲区 Defining buffers

int main(void)
{
    // 开发板初始化 Development board initialization
    board_init();

    // 初始化 MPU6050 Initialize MPU6050
    MPU6050_Init();

    uint8_t ret = 1;
    float pitch = 0, roll = 0, yaw = 0;  // 欧拉角 Euler Angles

    printf("start\r\n");

    // DMP 初始化 DMP initialization
    while (mpu_dmp_init())
    {
        printf("DMP error\r\n");
        delay_ms(200);
    }

    printf("Initialization Data Succeed \r\n");

    while (1)
    {
        // 获取欧拉角 Get Euler angles
        if (mpu_dmp_get_data(&pitch, &roll, &yaw) == 0)
        {
            // 格式化数据并发送 Format data and send
            sprintf(buf, "pitch = %d, roll = %d, yaw = %d\n", (int)pitch, (int)roll, (int)yaw);
            uart0_send_string(buf);  // 串口发送 Serial port sending
        }
        else
        {
            // 获取数据失败 Failed to obtain data
            printf("Data get error\r\n");
        }

        delay_ms(200);  // 延时，根据实际采样率调整 Delay, adjusted according to the actual sampling rate
    }

}
```

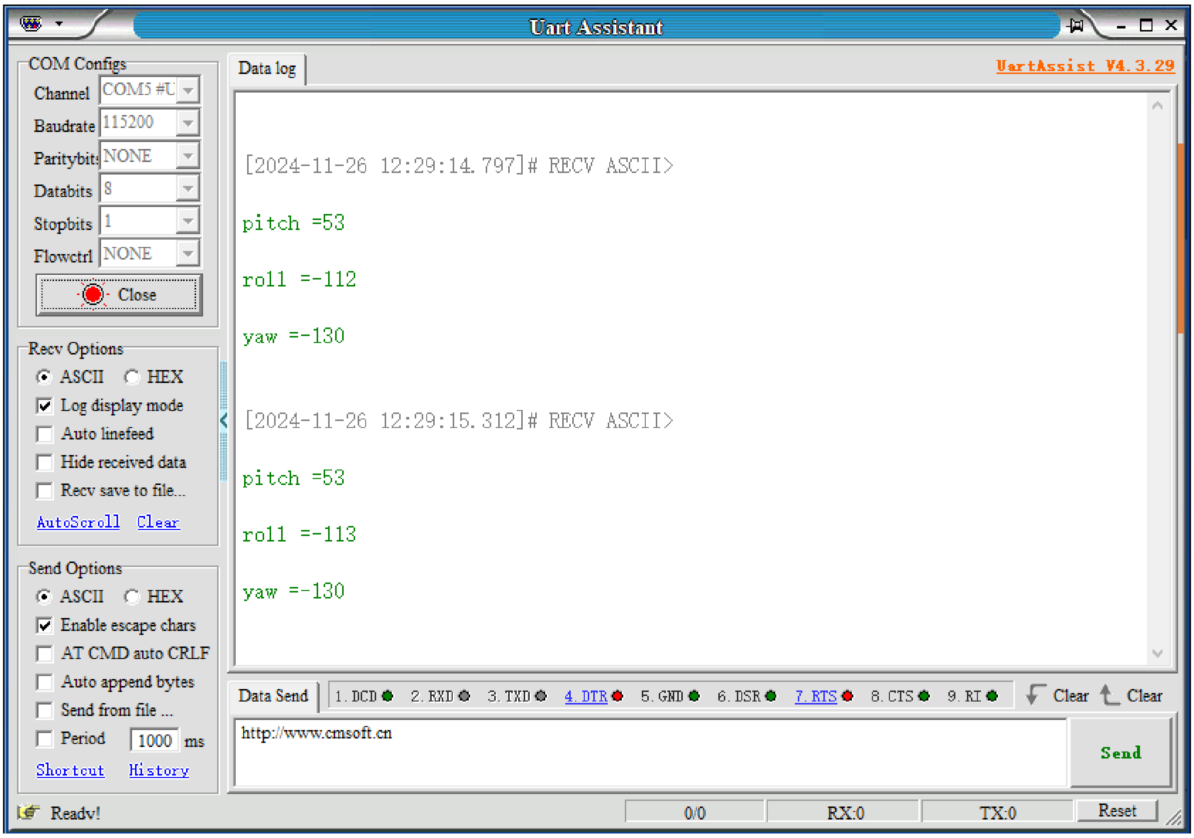
This program obtains posture data (`pitch`, `roll`, `yaw`) through the **MPU6050** sensor and uses the **DMP** (digital motion processor) to calculate the Euler angle.

After initializing the development board and the **MPU6050** sensor, try to initialize the DMP module. If the initialization fails, try again.

In the main loop, the Euler angle data of the sensor is continuously obtained, formatted and sent through the serial port. If the data acquisition fails, an error message is output. Data is acquired every 200 milliseconds to ensure an appropriate sampling frequency.

V. Experimental phenomenon

After the program is downloaded, configure the serial port assistant as shown in the figure below, open the serial port, and you can read the real-time data of the MPU6050 module.


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
Ready!
Data log
UartAssist V4.3.29
[2024-11-26 12:29:14.797]# RECV ASCII>
pitch =53
roll =-112
yaw =-130
[2024-11-26 12:29:15.312]# RECV ASCII>
pitch =53
roll =-113
yaw =-130
Data Send
1.DCD 2.RXD 3.TXD 4.DTR 5.GND 6.DSR 7.RTS 8.CTS 9.RI
Clear Clear
http://www.cmsoft.cn
Send
0/0
RX:0
TX:0
Reset