3.18 MPU6050 get data

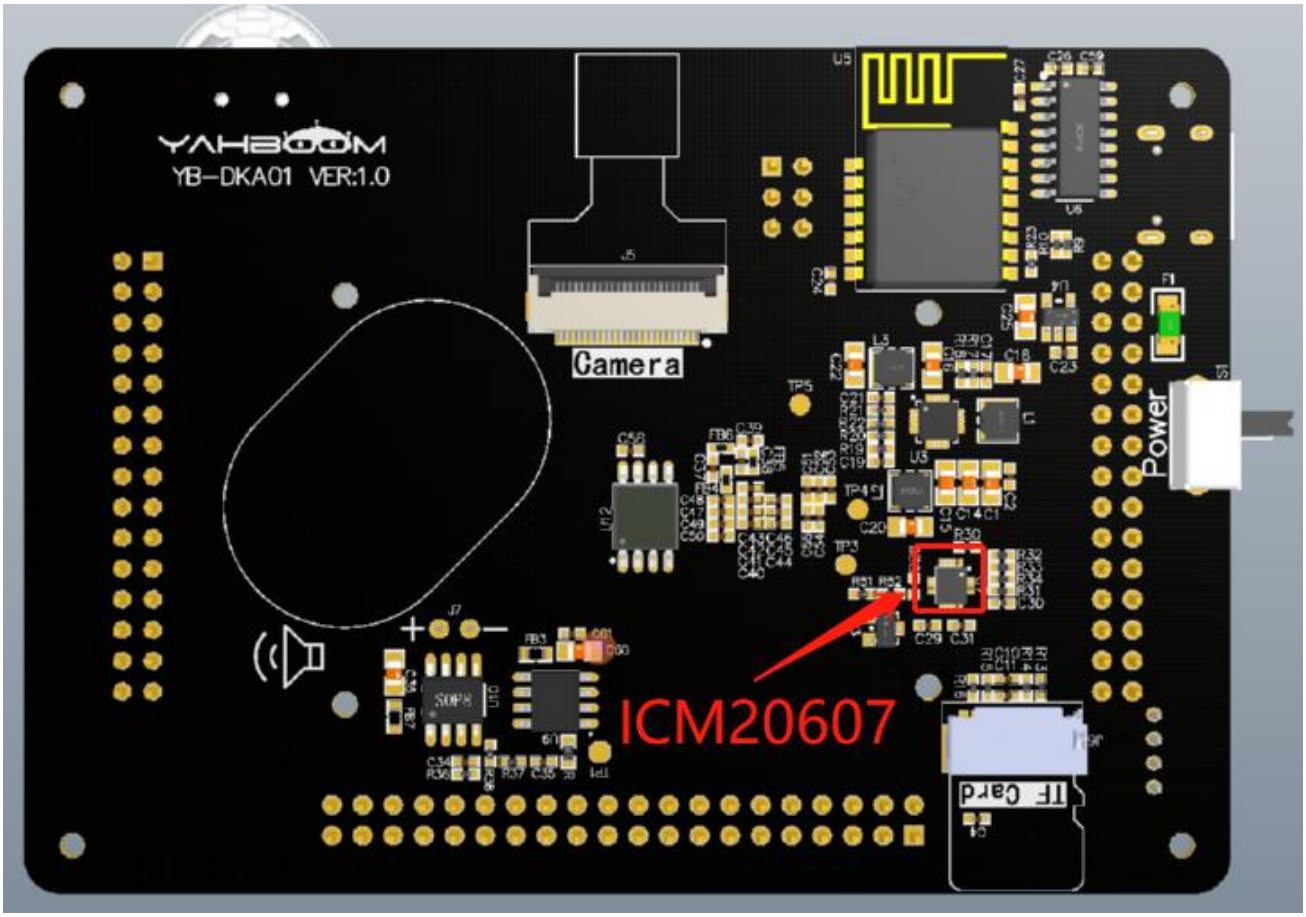
1. Experiment purpose

In this lesson, we mainly learn k210 read X/Y/Z axis raw data of ICM20607 chip.

2.Experiment preparation

2.1 components

MPU605 sensor GD25LQ128C



2.2 Component characteristics

ICM20607 is a six-axis motion tracking device, which combines a 3-axis gyroscope and a 3-axis accelerometer, with a 1K byte FIFO.

The programmable range of ICM20607's gyroscope is ±250, ±500, ±1000 and ±2000 degrees/second, and the accelerometer's full range is ±2g, ±4g, ±8g and ±16g.

ICM20607 includes on-chip 16-bit ADC, programmable digital filter, embedded temperature sensor and programmable interrupt, supports I2C and SPI communication. VDD operating range is 1.71V~3.45V.

The 3-axis MEMS gyroscope has the following characteristics in ICM-20607:

• Digital output X, Y, Z axis angular velocity sensor (gyro), user-programmable full scale range of ±250, ±500, ±1000 and ±2000°/sec, integrated 16-bit adc

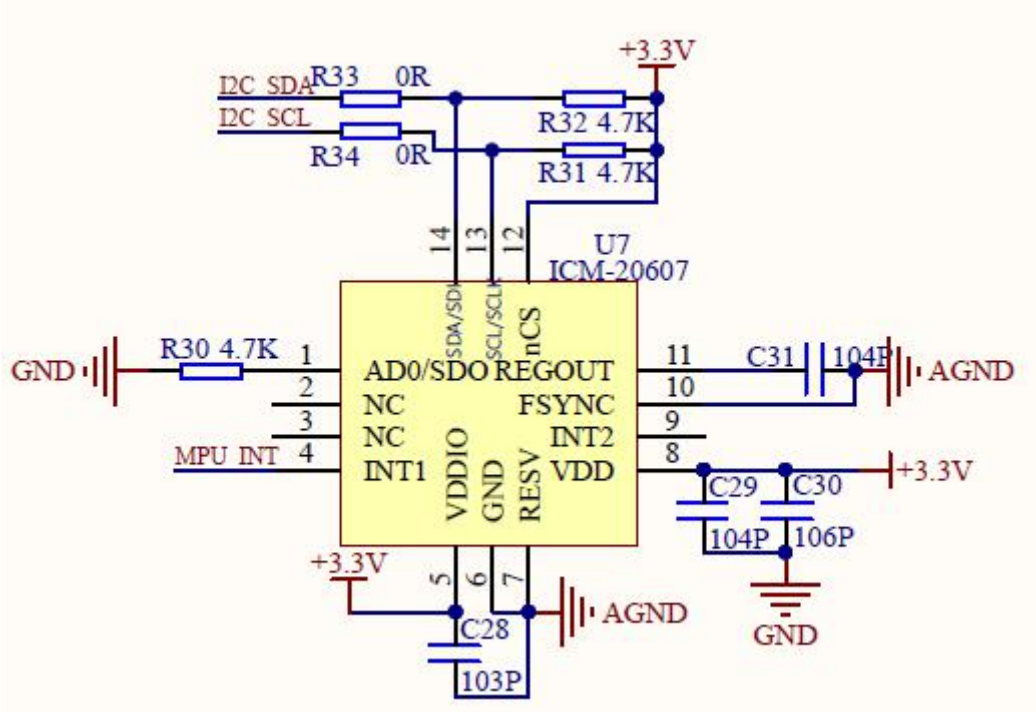
• Digital programmable low pass filter

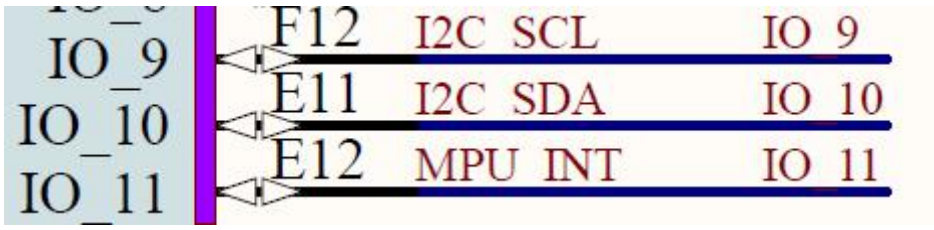
• Low-power gyroscope operation

• Factory-calibrated sensitivity scale factor
• Self-test

The 3-axis MEMS accelerometer includes the following features in ICM-20607:
•Digital output X, Y, Z-axis accelerometer, programmable full-scale range of ±2g, ±4g, ±8g, ±16g, integrated 16-bit adc
• Programmable interrupt
• Wake-on-motion
• Self-test

2.3 Hardware connection
ICM20607 chip adopt I2C connection, I2C_SCL is connected to IO9, I2C_SDA is connected to IO10, and MPU_INT is connected to IO11.





2.4 SDK API function
The header file is i2c.h
The I2C bus is used to communicate with multiple external devices. Multiple external devices can share an I2C bus.
The K210 chip integrated circuit bus has 3 I2C bus interfaces, they can be used as I2C master (MASTER) mode or slave (SLAVE) mode.

The I2C interface supports standard mode (0 to 100kb/s), fast mode (<=400kb/s), 7-bit or 10-bit addressing mode, bulk transfer mode, interrupt or polling mode operation.

We will provide following interfaces to users.
• i2c_init: Initialize I2C, configure the slave address, register bit width and I2C rate.
• i2c_init_as_slave: Configure I²C as slave mode.
• i2c_send_data: I2C write data.
• i2c_send_data_dma: I2C writes data through DMA.
• i2c_recv_data: I2C reads data through the CPU.
• i2c_recv_data_dma: I2C reads data through the dma.
• i2c_handle_data_dma: I2C uses dma to transmit data.

3. Experimental principle

Gyroscope is a device designed to sense and maintain direction based on the theory of conservation of angular momentum.

4. Experiment procedure

4.1 According to the above hardware connection pin diagram, K210 hardware pins and software functions use FPIOA mapping relationship.

```
/*****************************HARDWARE-PIN*********************************/
// Hardware IO port, corresponding Schematic

#define PIN_ICM_SCL             (9)
#define PIN_ICM_SDA             (10)
#define PIN_ICM_INT             (11)

/*****************************SOICMWARE-GPIO*******************************/
// Software GPIO port, corresponding program
#define ICM_INT_GPIONUM         (2)

/*****************************FUNC-GPIO************************************/
// Function of GPIO port, bound to hardware IO port
#define FUNC_ICM_INT            (FUNC_GPIOHS0 + ICM_INT_GPIONUM)
#define FUNC_ICM_SCL            (FUNC_I2C0_SCLK)
#define FUNC_ICM_SDA            (FUNC_I2C0_SDA)
```

```
void hardware_init(void)
{
    /* I2C ICM20607 */
    fpioa_set_function(PIN_ICM_SCL, FUNC_ICM_SCL);
    fpioa_set_function(PIN_ICM_SDA, FUNC_ICM_SDA);
}
```

4.2 Initialize ICM20607 chip.

```c
/* Initialize icm20607 chip */
void icm20607_init(void)
{
    uint8_t val = 0x0, res = 1;
    i2c_hardware_init(ICM_ADDRESS); // Initialization
    msleep(10);

    icm_i2c_write(PWR_MGMT_1, 0x80); //Reset device
    msleep(100);
    icm20607_who_am_i();

    do
    { //Wait reset successful
        icm_i2c_read(PWR_MGMT_1, &val, 1);
    } while(0x41 != val);

    icm_i2c_write(PWR_MGMT_1, 0x01);     //clock setting
    icm_i2c_write(PWR_MGMT_2, 0x00);     //Turn on the gyroscope and accelerometer
    icm_i2c_write(CONFIG, 0x01);         //176HZ 1KHZ
    icm_i2c_write(SMPLRT_DIV, 0x07);     //Sampling rate SAMPLE_RATE = INTERNAL_SA
    icm_i2c_write(GYRO_CONFIG, 0x18);    //±2000 dps
    icm_i2c_write(ACCEL_CONFIG, 0x10);   //±8g
    icm_i2c_write(ACCEL_CONFIG_2, 0x23); //Average 8 samples   44.8HZ
    return res;
}
```

4.3 Determine if it is AN ICM20607 chip.

```c
/* Determine if it is AN ICM20607 chip */
void icm20607_who_am_i(void)
{
    uint8_t val, state = 1;
    do
    {
        icm_i2c_read(WHO_AM_I, &val, 1);  // Read ICM20607 ID

        if (ICM20607_ID != val & state)  // If the ID is incorrect, it is only reported once.
        {
            printf("WHO_AM_I=0x%02x\n", val);
            state = 0;
        }
    } while(ICM20607_ID != val);
    printf("WHO_AM_I=0x%02x\n", val);
}
```

4.4 Read the original XYZ-axis data of the gyroscope.

```
/* Read the original X-axis data of the gyroscope*/
int16_t getRawGyroscopeX(void) {
    uint8_t val[2] = {0};
    icm_i2c_read(GYRO_XOUT_H, val, 2);
    return ((int16_t)val[0] << 8) + val[1];
}

/* Read the original Y-axis data of the gyroscope */
int16_t getRawGyroscopeY(void) {
    uint8_t val[2] = {0};
    icm_i2c_read(GYRO_YOUT_H, val, 2);
    return ((int16_t)val[0] << 8) + val[1];
}

/* Read the original Z-axis data of the gyroscope */
int16_t getRawGyroscopeZ(void) {
    uint8_t val[2] = {0};
    icm_i2c_read(GYRO_ZOUT_H, val, 2);
    return ((int16_t)val[0] << 8) + val[1];
}
```

4.5 Read the original XYZ-axis data of the accelerometer.

```
/* Read the original accelerometer X-axis data */
int16_t getRawAccelerationX(void) {
    uint8_t val[2] = {0};
    icm_i2c_read(ACCEL_XOUT_H, val, 2);
    return ((int16_t)val[0] << 8) + val[1];
}

/* Read the original accelerometer Y-axis data */
int16_t getRawAccelerationY(void) {
    uint8_t val[2] = {0};
    icm_i2c_read(ACCEL_YOUT_H, val, 2);
    return ((int16_t)val[0] << 8) + val[1];
}

/* Read the original accelerometer Z-axis data */
int16_t getRawAccelerationZ(void) {
    uint8_t val[2] = {0};
    icm_i2c_read(ACCEL_ZOUT_H, val, 2);
    return ((int16_t)val[0] << 8) + val[1];
}
```

4.6 Print out the corresponding data. The default is to print gyroscope data. You can change the displayed data by modifying the values of GYRO_DATA and ACC_DATA.

```
#define GYRO_DATA     1
#define ACC_DATA      0
```

```
while (1)
{
    #if GYRO_DATA
    val_gx = getRawGyroscopeX();
    val_gy = getRawGyroscopeY();
    val_gz = getRawGyroscopeZ();
    printf("gx=%d, gy=%d, gz=%d\n", val_gx, val_gy, val_gz);
    val_gx = val_gy = val_gz = 0;
    #elif ACC_DATA
    val_ax = getRawAccelerationX();
    val_ay = getRawAccelerationY();
    val_az = getRawAccelerationZ();
    printf("ax=%d, ay=%d, az=%d\n", val_ax, val_ay, val_az);
    val_ax = val_ay = val_az = 0;
    #else
    printf("Please set the GYRO_DATA or ACC_DATA to 1\n");
    #endif
    msleep(5);
}
```

4.7 Compile and debug, burn and run

Copy the gyro to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

cmake .. -DPROJ=gyro -G "MinGW Makefiles"

make

```
[ 93%] Building C object CMakeFiles/gyro.dir/src/gyro/main.c.obj
[ 95%] Linking C executable gyro
Generating .bin file ...
[100%] Built target gyro
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

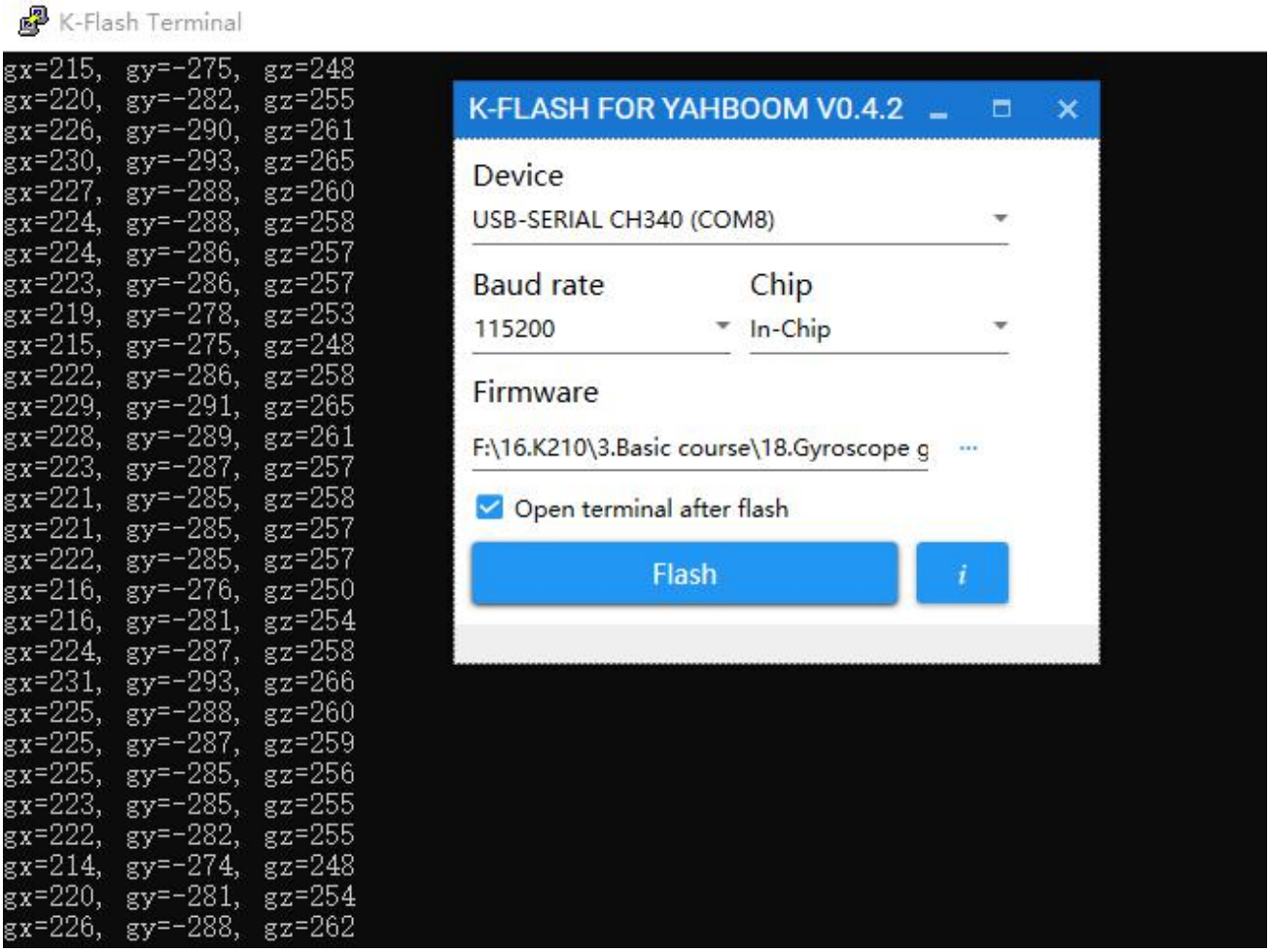
After the compilation is complete, the **gyro.bin** file will be generated in the build folder.

We need to use the type-C data cable to connect the computer and the K210 development board.

Open kflash, select the corresponding device, and then burn the **gyro.bin** file to the K210 development board.

5. Experimental phenomenon

After the firmware is write, a terminal interface will pop up. The gyroscope's data will be printed out. As shown below.

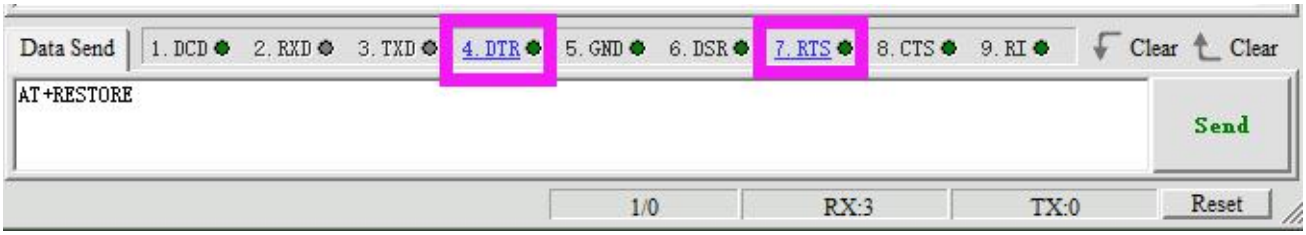


If you can't see terminal interface, please open the serial port assistant of the computer, select the corresponding serial port number of the corresponding K210 development board, set the baud rate to 115200.

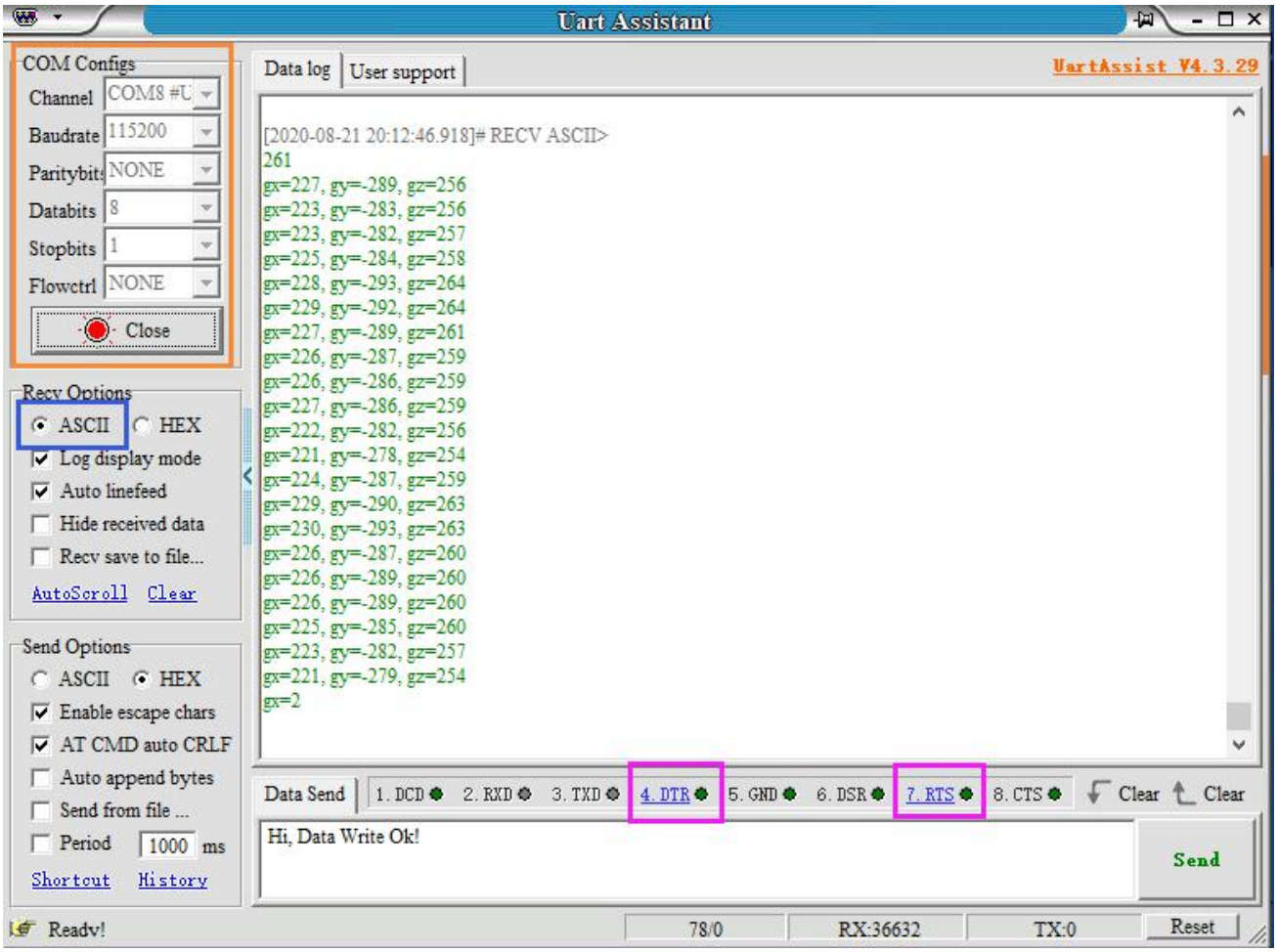
Then, click to open the serial port assistant.

!Note: you also need to set the DTR and RTS of the serial port assistant.

Click 4.DTR and 7.RTS to set them to green.



The serial port assistant will display the data printed by the current gyroscope. When we swing the development board up and down, the gyroscope data will change accordingly.



6. Experiment summary

6.1 The CM20607 chip is a 6-axis sensor chip, which contains a 3-axis gyroscope and a 3-axis accelerometer.

6.2 The gyroscope and accelerometer need to be initialized before using, otherwise they cannot be used normally.

6.3 This time icm20607 uses I2C communication to transmit data, and SPI can also be used to transmit data.

Appendix -- API

Header file is i2c.h

i2c_init

Description: Configure the I²C device slave address, register bit width, and I²C rate.

Function prototype: **void i2c_init(i2c_device_number_t i2c_num, uint32_t slave_address, uint32_t address_width, uint32_t i2c_clk)**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C number	Input
slave_address	I²C device slave address	Input
address_width	I²C device register width (7 or 10)	Input
i2c_clk	I²C rate (Hz)	Input

Return value: no

i2c_init_as_slave

Description: Configure I²C for slave mode.

Function prototype: **void i2c_init_as_slave(i2c_device_number_t i2c_num, uint32_t slave_address, uint32_t address_width, const i2c_slave_handler_t *handler)**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C number	Input
slave_address	I²C slave mode address	Input
address_width	I²C device register width (7 or 10)	Input
handler	I²C Interrupt handler function from mode	Input

Return value: no

i2c_send_data

Description: Write data

Function prototype: i**nt i2c_send_data(i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len)**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C number	Input
send_buf	Data waiting to be transmitted	Input
send_buf_len	Length of Data waiting to be transmitted	Input

Return value:

Return value	Description
0	succeed
!0	Failure

i2c_send_data_dma

Description: Write data by DMA

Function prototype: **void i2c_send_data_dma(dmac_channel_number_t dma_channel_num, i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len)**

Parameter:

Parameter name	Description	Input/Output
dma_channel_num	dma channel number	Input

Parameter name	Description	Input/Output
i2c_num	I²C number	Input
send_buf	Data waiting to be transferred	Input
send_buf_len	Length of data waiting to be transferred	Input

Return value: No

i2c_recv_data

Description: Read data by CPU

Function prototype: **int i2c_recv_data(i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len, uint8_t *receive_buf, size_t receive_buf_len)**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C bus number	Input
send_buf	The data waiting to be transmitted is generally the register of the i2c peripheral, if it is not set to NULL	Input
send_buf_len	Length of the data waiting to be transmitted, if not, write 0	Input
receive_buf	Receive data memory	Output
receive_buf_len	Length of receive data	Output

Return value:

Return value	Description
0	succeed
!0	Failure

i2c_recv_data_dma

Description: Read data bu dma

Function prototype: **void i2c_recv_data_dma(dmac_channel_number_t dma_send_channel_num, dmac_channel_number_t dma_receive_channel_num, i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len, uint8_t *receive_buf, size_t receive_buf_len)**

Parameter:

Parameter name	Description	Input/Output
dma_send_channel_num	DMA channel used to send data	Input
dma_receive_channel_num	DMA channel used to receive data	Input
i2c_num	I²C bus number	Input
send_buf	The data to be transmitted is generally the register of the i2c peripheral, if it is not set to NULL	Input
send_buf_len	The length of the data to be transmitted, if not, write 0	Input

Parameter name	Description	Input/Output
receive_buf	Receive data memory	Output
receive_buf_len	Length of received data	Input

Return value: no

i2c_handle_data_dma

Description: I2C uses dma to transmit data.

Function prototype: **void i2c_handle_data_dma(i2c_device_number_t i2c_num, i2c_data_t data, plic_interrupt_t *cb);**

Parameter:

Parameter name	Description	Input/Output
i2c_num	I²C bus number	Input
data	I2C data related parameters	Input
cb	DMA interrupt callback function, if it is set to NULL, it is in blocking mode, and the function exits after the transmission is completed	Input

Return value: no

Eg:

```
/* The i2c peripheral address is 0x32, 7-bit address, and the rate is 200K */
i2c_init(I2C_DEVICE_0, 0x32, 7, 200000);
uint8_t reg = 0;
uint8_t data_buf[2] = {0x00,0x01}
data_buf[0] = reg;

/* Write 0x01 to register 0*/
i2c_send_data(I2C_DEVICE_0, data_buf, 2);
i2c_send_data_dma(DMAC_CHANNEL0, I2C_DEVICE_0, data_buf, 4);

/* Read 1 byte of data from register 0*/
i2c_receive_data(I2C_DEVICE_0, &reg, 1, data_buf, 1);
i2c_receive_data_dma(DMAC_CHANNEL0, DMAC_CHANNEL1, I2C_DEVICE_0,&reg, 1, data_buf, 1);
```

Data type

The related data types and data structure are defined as follows:

- i2c_device_number_t: i2c number.
- i2c_slave_handler_t: interrupt handler handle of i2c slave mode.
- i2c_data_t: data-related parameters when using dma transmission.
- i2c_transfer_mode_t: The mode of using DMA to transfer data, sending or receiving.

i2c_device_number_t

Description: i2c number.

Define

```
typedef enum _i2c_device_number
{
    I2C_DEVICE_0,
    I2C_DEVICE_1,
    I2C_DEVICE_2,
    I2C_DEVICE_MAX,
} i2c_device_number_t;
```

i2c_slave_handler_t

Description: i2c Slave mode interrupt handler function. According to different interrupt states, perform corresponding function operations

Define

```
typedef struct _i2c_slave_handler
{
    void(*on_receive)(uint32_t data);
    uint32_t(*on_transmit)();
    void(*on_event)(i2c_event_t event);
} i2c_slave_handler_t;
```

member

Member name	Description
I2C_DEVICE_0	I2C 0
I2C_DEVICE_1	I2C 1
I2C_DEVICE_2	I2C 2

i2c_data_t

Description: data-related parameters when using dma transmission.

Define

```
typedef struct _i2c_data_t
{
    dmac_channel_number_t tx_channel;
    dmac_channel_number_t rx_channel;
    uint32_t *tx_buf;
    size_t tx_len;
    uint32_t *rx_buf;
    size_t rx_len;
    i2c_transfer_mode_t transfer_mode;
} i2c_data_t;
```

member

Member name	Description
tx_channel	DMA channel number used when sending

Member name	Description
rx_channel	DMA channel number used when receiving
tx_buf	Sent data
tx_len	Length of sent data
rx_buf	Received Data
rx_len	Received Data of received Data
transfer_mode	Transfer_mode, sent or received

i2c_transfer_mode_t

Description: The mode of using DMA to transfer data, send or receive.

Define

```
typedef enum _i2c_transfer_mode
{
    I2C_SEND,
    I2C_RECEIVE,
} i2c_transfer_mode_t;
```

member

Member name	Description
I2C_SEND	send
I2C_RECEIVE	receive