

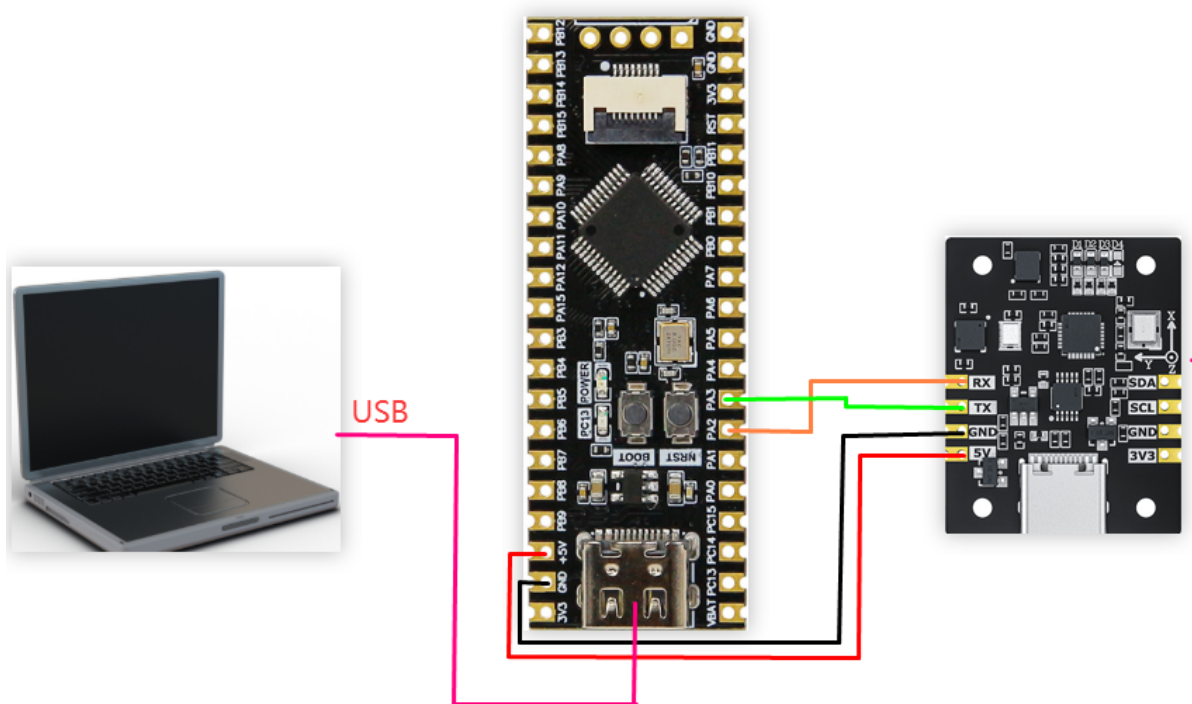
STM32 Serial Port Data Reading

STM32 Serial Port Data Reading

1. Connecting the Device
2. Key Code Analysis
3. Reading IMU Data

This example uses the STM32F103C8T6 core board (Yabo), a Windows computer, several DuPont wires, and an IMU attitude sensor.

1. Connecting the Device



IMU Attitude Sensor	STM32F103C8T6 Core Board (Yahboom)
RX	PA2
TX	PA3
GND	GND
5V	5V

2. Key Code Analysis

Please refer to the source code in the documentation for specific code.

```
//Process RX ring buffer, parse frames and update internal cache  
  
void IMU_UART_Process(void)
```

```

{
    enum {
        RX_STATE_EXPECT_HEAD1 = 0,
        RX_STATE_EXPECT_HEAD2,
        RX_STATE_EXPECT_LENGTH,
        RX_STATE_EXPECT_FUNCTION,
        RX_STATE_COLLECT_DATA
    };

    static uint8_t rx_state = RX_STATE_EXPECT_HEAD1;
    static uint8_t frame_length = 0;
    static uint8_t frame_function = 0;
    static uint8_t frame_buffer[64]; /* data section + checksum */
    static uint16_t frame_index = 0;

    uint8_t current_byte = 0;

    while (_rxbuf_pop(&current_byte) == 0) {
        switch (rx_state) {
            case RX_STATE_EXPECT_HEAD1:
                rx_state = (current_byte == FRAME_HEAD1) ? RX_STATE_EXPECT_HEAD2 :
RX_STATE_EXPECT_HEAD1;
                break;

            case RX_STATE_EXPECT_HEAD2:
                rx_state = (current_byte == FRAME_HEAD2) ? RX_STATE_EXPECT_LENGTH :
RX_STATE_EXPECT_HEAD1;
                break;

            case RX_STATE_EXPECT_LENGTH:
                frame_length = current_byte;
                rx_state = RX_STATE_EXPECT_FUNCTION;
                break;

            case RX_STATE_EXPECT_FUNCTION:
                frame_function = current_byte;
                frame_index = 0;
                rx_state = RX_STATE_COLLECT_DATA;
                break;

            case RX_STATE_COLLECT_DATA: {
                uint16_t data_length = (frame_length >= 4) ? (uint16_t)(frame_length
- 4) : 0;
                if (data_length == 0 || data_length > sizeof(frame_buffer)) {
                    rx_state = RX_STATE_EXPECT_HEAD1;
                    break;
                }

                frame_buffer[frame_index++] = current_byte;
                if (frame_index >= data_length) {
                    uint8_t calculated_checksum = (uint8_t)(FRAME_HEAD1 +
FRAME_HEAD2 + frame_length + frame_function);
                    for (uint16_t i = 0; i < data_length - 1; ++i) {
                        calculated_checksum = (uint8_t)(calculated_checksum +
frame_buffer[i]);
                    }

                    uint8_t received_checksum = frame_buffer[data_length - 1];

```

```

        if (calculated_checksum == received_checksum) {
            _parse_frame_data(frame_function, frame_buffer);
        }
        rx_state = RX_STATE_EXPECT_HEAD1;
    }
} break;

default:
    rx_state = RX_STATE_EXPECT_HEAD1;
    break;
}
}
}

/* ----- Parse one complete frame ----- */
static void _parse_frame_data(uint8_t frame_function, const uint8_t *frame_data)
{
    if (frame_function == IMU_FUNC_RAW_ACCEL) {
        float accel_ratio = 16.0f / 32767.0f;
        s_ax = to_int16(&frame_data[0]) * accel_ratio;
        s_ay = to_int16(&frame_data[2]) * accel_ratio;
        s_az = to_int16(&frame_data[4]) * accel_ratio;

        float deg_to_rad = 3.14159265358979323846f / 180.0f;
        float gyro_ratio = (2000.0f / 32767.0f) * deg_to_rad;
        s_gx = to_int16(&frame_data[6]) * gyro_ratio;
        s_gy = to_int16(&frame_data[8]) * gyro_ratio;
        s_gz = to_int16(&frame_data[10]) * gyro_ratio;

        float mag_ratio = 800.0f / 32767.0f;
        s_mx = to_int16(&frame_data[12]) * mag_ratio;
        s_my = to_int16(&frame_data[14]) * mag_ratio;
        s_mz = to_int16(&frame_data[16]) * mag_ratio;
    } else if (frame_function == IMU_FUNC_EULER) {
        s_roll = to_float(&frame_data[0]);
        s_pitch = to_float(&frame_data[4]);
        s_yaw = to_float(&frame_data[8]);
    } else if (frame_function == IMU_FUNC_QUAT) {
        s_q0 = to_float(&frame_data[0]);
        s_q1 = to_float(&frame_data[4]);
        s_q2 = to_float(&frame_data[8]);
        s_q3 = to_float(&frame_data[12]);
    } else if (frame_function == IMU_FUNC_BARO) {
        s_height = to_float(&frame_data[0]);
        s_temperature = to_float(&frame_data[4]);
        s_pressure = to_float(&frame_data[8]);
        s_pressure_contrast = to_float(&frame_data[12]);
    } else if (frame_function == IMU_FUNC_VERSION) {
        s_version_high = frame_data[0];
        s_version_mid = frame_data[1];
        s_version_low = frame_data[2];
    } else if (frame_function == IMU_FUNC_RETURN_STATE) {
        s_last_rx_function = frame_data[0];
        s_last_rx_state = (int16_t)frame_data[1];
    }
}
}

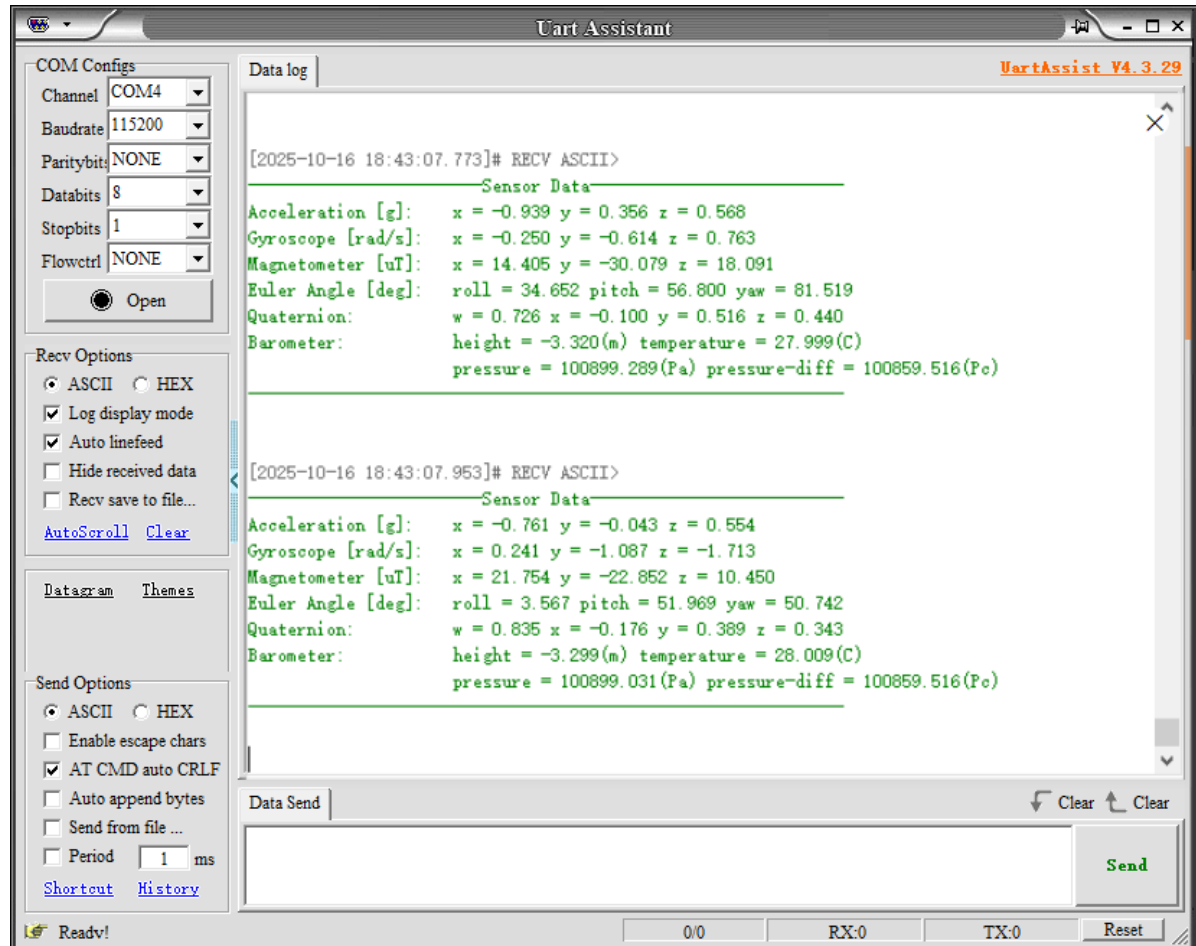
```

IMU_UART_Process(): Reads the cached data and calls _parse_frame_data to parse the data conforming to the communication protocol.

_parse_frame_data(): Parses the data.

3. Reading IMU Data

After the program is downloaded into STM32, open the serial port assistant (configuration parameters are shown in the figure below). You can see that the IMU module data is continuously printed. When we change the attitude of the IMU module, the data will change.



Note: The above is the data reading of 10-axis IMU, 6-axis without magnetometer and barometer data, and 9-axis without barometer data.