2.1 Low-level Parsing at the Slave End (Arduino Version)

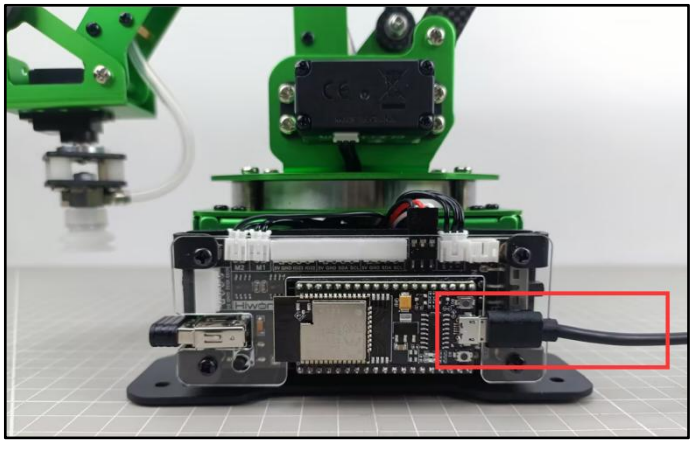
This lesson will explain the low-level program of MaxArm's slave-side communication control functions, analyzing MaxArm's reception of data from other devices, parsing the data to control MaxArm, and implementing the functionality to send data to other devices.

1. Communication Connection

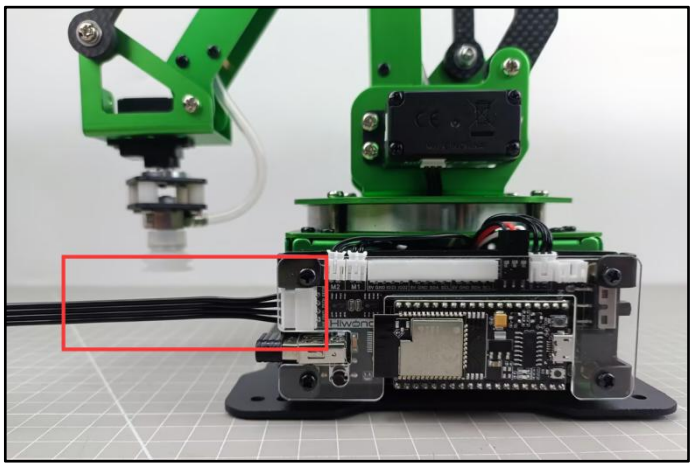
1.1 Hardware Connection

MaxArm's serial communication supports two interfaces, one is the 4-pin interface, suitable for devices with pin interfaces for UART communication, commonly used with Arduino development boards and STM32 development board. The other interface is the micro-USB interface, suitable for devices with USB master interfaces, such as Raspberry Pi, Jetson Nano and others.

1) The communication connection for the micro-USB interface is as follow:



2) The communication connection for 4Pin interfaces is as follow:



1.2 Communication Protocol

The communication protocol for both the master and slave devices in the MaxArm communication routines follows the following format:

Frame header	Function code	Data length	Data information	Check bit
0xAA 0x55	func	len	data	check

The annotations of each part of the protocol are as follows:

Frame header: if 0xAA and 0x55 are received sequentially, it indicates that there is data to be received, consisting of a fixed 2 bytes.

Function Code: Used to indicate the purpose of an information frame, consists of 1 byte.

Data Length: Indicates the number of data bits carried by the data frame.

Check Bit: Verifies the correctness of the data frame. If correct, the corresponding function is called; otherwise, the data frame is skipped.

The calculation method for the check bit is: calculate the sum of the function code, data length, and data, then take the complement, and finally, take the low byte, which serves as the checksum.

1.3 Functions and Corresponding Function Code Instructions

Function Name	Instruction	Function Code
FUNC_SET_ANGLE	Set servo angle	0x01
FUNC_SET_XYZ	Set the robotic arm coordinate position	0x03
FUNC_SET_PWMSERVO	Control PWM servo for nozzle	0x05
FUNC_SET_SUCTIONNOZZLE	Control the air pump state	0x07
FUNC_READ_ANGLE	Control servo angle	0x11
FUNC_READ_XYZ	Read robotic arm coordinate position	0x13

2. Program Interface Parsing

The routines analyzed in this document are located in **“Communication Routine Low-Level Files (Arduino Version)/ MaxArm_Arduino_microUSB”** in the same directory. The routines for communication using the micro-USB interface are provided, and similar routines for communication using the 4-pin interface are also available, with the difference being the use of different IO ports for communication. This section primarily analyzes the files "**PC_rec.h**" and "**PC_rec.c**", which are the implementation files for the slave-level communication of MaxArm.

In the "**MaxArm_Arduino_microUSB.ino**" file, which serves as the entry point for the Arduino program, the program flow is as follows:

2.1 Create a Serial Communication Control Object

Create a serial communication object globally, as shown in the following figure:

```
PC_REC pc_rec;
```

The constructor of the object is invoked during its creation to initialize the serial port baud rate to 9600. This is implemented in the "PC_rec.c" file as shown below:

```
PC_REC::PC_REC(void)
{
  Serial.begin(9600);
}
```

2.2 Initialize MaxArm Control Interface

In the void setup() function, the buzzer, bus servo interface, end effector interface, PWM servo interface, etc., are initialized, and the robotic arm is reset, as shown in the following figure:

```
void setup() {
  // 初始化
  Buzzer_init(); //蜂鸣器
  ESPMax_init(); //总线舵机
  Nozzle_init(); //吸嘴
  PWMServo_init(); //PWM舵机
  Valve_on();
  go_home(2000);
```

```
  Valve_off();
  delay(100);
  SetPWMServo(1, 1500, 1000);
  Serial.println("start...");
}
```

2.3 Receiving, Parsing, and Controlling

In the void loop() function, continuously call the rec_data() function in a loop to wait for serial port information reception and parse the information to control

the MaxArm robotic arm.

1）The **void rec_data(void)** function is implemented in the "PC_rec.c" file. Its function is to receive serial port information, parse the serial port information, and control MaxArm or return the read information based on the information. The function flow is as follows:

Check if data is received on the serial port. If so, copy the data to the pk_ctl.data[] array.

```
//读取数据
uint32_t len = Serial.available();
while(len--)
{
  int rd = Serial.read();
  pk_ctl.data[pk_ctl.index_tail] = (char)rd;
  pk_ctl.index_tail++;
  if(BUFFER_SIZE <= pk_ctl.index_tail)
  {
    pk_ctl.index_tail = 0;
  }
  if(pk_ctl.index_tail == pk_ctl.index_head)
  {
    pk_ctl.index_head++;
    if(BUFFER_SIZE <= pk_ctl.index_head)
    {
      pk_ctl.index_head = 0;
    }
  }else{
    pk_ctl.len++;
  }
}
```

Iterate through the bytes of data in pk_ctl.data[], parse the data according to the communication protocol, and save the correctly received function code, data, etc., in the pk_ctl.frame structure variable. Part of the program is as follows:

```
uint8_t crc = 0;
//解析数据
while(pk_ctl.len > 0)
{
  switch(pk_ctl.state)
  {
    case STATE_STARTBYTE1: /* 处理帧头标记1 */
      pk_ctl.state = CONST_STARTBYTE1 ==
                     pk_ctl.data[pk_ctl.index_head] ?
                     STATE_STARTBYTE2 : STATE_STARTBYTE1;
      break;
    case STATE_STARTBYTE2: /* 处理帧头标记2 */
      pk_ctl.state = CONST_STARTBYTE2 ==
                     pk_ctl.data[pk_ctl.index_head] ?
                     STATE_FUNCTION : STATE_STARTBYTE1;
      break;
    case STATE_FUNCTION: /* 处理帧功能号 */
      pk_ctl.state = STATE_LENGTH;
      if(FUNC_SET_ANGLE != pk_ctl.data[pk_ctl.index_head])
        if(FUNC_SET_XYZ != pk_ctl.data[pk_ctl.index_head])
          if(FUNC_SET_PWMSERVO != pk_ctl.data[pk_ctl.index_head])
            if(FUNC_SET_SUCTIONNOZZLE != pk_ctl.data[pk_ctl.index_head])
              if(FUNC_READ_ANGLE != pk_ctl.data[pk_ctl.index_head])
                if(FUNC_READ_XYZ != pk_ctl.data[pk_ctl.index_head])
                {
                  pk_ctl.state = STATE_STARTBYTE1;
                }
```

```
    if(STATE_LENGTH == pk_ctl.state) {
        pk_ctl.frame.function = pk_ctl.data[pk_ctl.index_head];
    }
    break;
  case STATE_LENGTH: /* 处理帧数据长度 */
    if(pk_ctl.data[pk_ctl.index_head] >= DATA_SIZE)
    {
      pk_ctl.state = STATE_STARTBYTE1;
      continue;
    }else{
      pk_ctl.frame.data_length = pk_ctl.data[pk_ctl.index_head];
      pk_ctl.state = (0 == pk_ctl.frame.data_length) ?
                     STATE_CHECKSUM : STATE_DATA;
      pk_ctl.data_index = 0;
      break;
    }
  case STATE_DATA: /* 处理帧数据 */
    pk_ctl.frame.data[pk_ctl.data_index] = pk_ctl.data[pk_ctl.index_head];
    ++pk_ctl.data_index;
    if(pk_ctl.data_index >= pk_ctl.frame.data_length) {
        pk_ctl.state = STATE_CHECKSUM;
        pk_ctl.frame.data[pk_ctl.data_index] = '\0';
    }
    break;
```

Compare the received checksum value with the calculated checksum value to determine whether the received data frame is correct. If received correctly, pass the parsed content to the deal_command() function, as shown in the figure:

```
case STATE_CHECKSUM: /* 处理校验值 */
  pk_ctl.frame.checksum = pk_ctl.data[pk_ctl.index_head];
  crc = checksum_crc8((uint8_t*)&pk_ctl.frame.function, pk_ctl.frame.data_length + 2);
  // Serial.println(crc);
  if(crc == pk_ctl.frame.checksum) { /* 校验失败，跳过执行 */
      deal_command(&pk_ctl.frame); //处理数据
  }
  memset(&pk_ctl.frame, 0, sizeof(struct PacketRawFrame)); //清除
  pk_ctl.state = STATE_STARTBYTE1;
  break;
```

The deal_command() function selects the corresponding function to execute based on the function code of the incoming information. The parsing programs for each function are as follows:

① For the "Set Angle" function, the incoming bytes are converted into three angle values and one time value. Then, each of the three bus servos is controlled to move according to the specified angles.

```
case FUNC_SET_ANGLE: //设置角度 0x01
  {
    Angle_Ctl_Data msg;
    if(len == 8)
    {
      memcpy(&msg , ctl_com->data , sizeof(msg));
      set_servo_in_range(1,msg.pul[0],msg.time);
      delay(2);
      set_servo_in_range(2,msg.pul[1],msg.time);
      delay(2);
      set_servo_in_range(3,msg.pul[2],msg.time);
      delay(2);
    }
  }
  break;
```

② For the "Set XYZ Axis Coordinates" function, the incoming bytes are converted into spatial coordinate values and one time value. Then, the coordinate setting interface is called to set the end-point angle coordinates of MaxArm.

```
case FUNC_SET_XYZ: //设置xyz轴 0x03
  {
    XYZ_Ctl_Data msg;
    if(len == 8)
    {
      memcpy(&msg , &ctl_com->data , sizeof(msg));
      float p[3] = {msg.pos[0],msg.pos[1],msg.pos[2]};
      set_position(p , msg.time);
    }
  }
  break;
```

③ When setting the pulse width of the end-point PWM servo, the incoming byte data is converted into PWM pulse width and time values. Subsequently, the PWM servo setting function is called to set the motion of the PWM servo.

```
case FUNC_SET_PWMSERVO: //设置PWM舵机 0x05
  {
    PWM_Ctl_Data msg;
    if(len == 4)
    {
      memcpy(&msg , &ctl_com->data , sizeof(msg));
      SetPWMServo(1, msg.pul, msg.time);
    }
  }
  break;
```

④ When setting the end effector nozzle function, the incoming bytes are converted into sub-function codes, and based on the sub-function code, the state of the nozzle is set.

```
case FUNC_SET_SUCTIONNOZZLE: //设置吸嘴 0x07
  {
    SN_Ctl_Data msg;
    if(len == 1)
    {
      memcpy(&msg , &ctl_com->data , sizeof(msg));
      switch(msg.cmd)
      {
        case 1:
          Pump_on();
          break;
        case 2:
          Valve_on();
          break;
        case 3:
          Valve_off();
          break;
      }
    }
  }
  break;
```

⑤ When reading angle values, the read_angles() function is called, and the read angle values are sent to the host device via the serial port.

```
case FUNC_READ_ANGLE: //读取角度 0x11
  {
    read_Angle_Data msg;
    int16_t angles[3];
    read_angles(angles);
    uint8_t send_data[20] = {0xAA , 0x55 , 0x11 , 0x06};
    memcpy(&send_data[4] , angles , 6);
    send_data[10] = checksum_crc8(&send_data[2] , 8);
    Serial.write(send_data , 11);
  }
  break;
```

⑥ When reading XYZ values, the read_position() function is called, and the read coordinate values are sent to the host device via the serial port.

```
case FUNC_READ_XYZ: //读取xyz轴 0x13
  {
    float pos_f[3];
    read_position(pos_f);
    int16_t pos[3] = {(int16_t)pos_f[0] , (int16_t)pos_f[1] , (int16_t)pos_f[2]};
    uint8_t send_data[20] = {0xAA , 0x55 , 0x13 , 0x06};
    memcpy(&send_data[4] , pos , 6);
    send_data[10] = checksum_crc8(&send_data[2] , 8);
    Serial.write(send_data , 11);
  }
  break;
```