Recorder Communication Protocol Description

## 1. Overview

The instrument uses the standard Modbus protocol.

## 2. Modbus transmission mode

Modbus standard network two transmission modes: ASCII or RTU. The instrument uses RTU mode, does not support ASCII mode.

RTU mode

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Address | Code | Data | Data 1 | ... | Data n | CRC High Word | CRC Low Word |

When the controller is set to communicate in RTU (Remote Terminal Unit) mode on a Modbus network, each 8Bit byte in the message contains two 4Bit hexadecimal characters. The main advantage of this approach is that at the same baud rate, more data can be transferred than in ASCII mode.

Code system

- 8-bit binary, hexadecimal 0 ... 9, A ... F
- Each 8-bit field in the message is composed of two hexadecimal characters

Bit per byte

- 1 start bit
- 8 data bits, the least significant first sent
- 1 parity bit, none if no parity
- 1 stop bit (with calibration), 2 bits (without calibration)

Error detection field

- CRC (Cyclic Length Detection)

## 3. Modbus function code

The following table is the Modbus supported function code:

| Code | Name | Effect |
|---|---|---|
| 01 | Read coil status | Get the current state of a set of logic coils (ON / OFF) |
| 02 | Read input status | Get the current status of a group of switch inputs (ON / OFF) |
| 03 | Read Holding Register | Get the current binary value in one or more holding registers |
| 04 | Read Input Register | Get the current binary value in one or more input registers |
| 05 | strong single coil | Strong one logic coil on-off state |
| 06 | Preset Single Register | Loads a specific binary value into a holding register |
| 07 | Read the abnormal state | Obtain the on-off state of 8 internal coils, the addresses of these 8 coils are decided by the controller |
| 08 | Loopback Diagnostics Checksum | sends a diagnostic check message to the slave to |

evaluate the communication process

09 Programming (only for 484) Enables the host analog programmer to modify the PC slave logic

10 Inquiry (only for 484) The master can communicate with a slave executing a long program task to inquire whether the slave has completed its operation. Only after the message containing the function code 9 is sent, the function code Only send

11 Reading the event count allows the host to issue a single query and immediately determine if the operation was successful, especially if a communication error occurs on the command or other response

12 Reading Communication Event Logs the master retrieves the ModBus transaction event record for each slave. If a transaction is completed, the record gives the error

13 Programming (184/384 484 584) Enables the host analog programmer function to modify the PC slave logic

14 Interrogation (184/384 484 584) The master can be used to communicate with the slave executing a task and periodically check whether the slave has completed its programmed operations. This function code is available only after a message containing function 13 has been sent Send

15 strong multi-coil strong series of continuous logic coil on-off

16 Preset Multi-Registers Load specific binary values into a succession of holding registers

17 Report Slave ID Enables the host to determine the type of addressed slave and the status of the slave run indicator

18 (884 and MICRO 84) allows the host to simulate programming functions and modify PC state logic

19 Reset communication link After a non-modifiable error, the slave resets to a known state, resetting the sequential bytes

20 Read General Parameters (584L) Displays the data information in the extended memory file

21 Write General Parameters (584L) Write general parameters to the extended memory file or modify

22 to 64 Reserved for extended function backup

65 to 72 Reserved Extended codes for user functions Reserved for standby function

73 ~ 119 illegal function

120 ~ 127 Reserve Reserved for internal use

128 ~ 255 Reserve Reserved for abnormal response

The instrument only supports 03 command (read holding register)

**4. Modbus message frame format**

Unless otherwise specified, any data greater than 1 byte are high byte first, low byte last. The maximum frame length must be limited to 256 bytes. In addition to the broadcast command, any host needs to slave to machine access to respond. CRC check error does not need to return.

Host send format

| Name | Device Address | Function Code | Data Area | CRC |
|---|---|---|---|---|
| Byte | 1 | 1 | N | 2 |

Host send format

| Name | Device Address | Function Code | Data Area | CRC |
|---|---|---|---|---|
| Byte | 1 | 1 | N | 2 |

**4.1 03H Read Holding Register**

Inquiry frame

| Name | Device Address | 0x03 | Start Address | Data Number | CRC |
|---|---|---|---|---|---|
| Number of bytes | 1 | 1 | 2 | 2 | 2 |

Return frame

| Name | Device Address | 0x03 | Number of bytes returned N | Return data | CRC |
|---|---|---|---|---|---|
| Number of bytes | 1 | 1 | 1 | N × 1 | 2 |

Description: The starting address and CRC are high byte first.

**5. Abnormal code**

Communication abnormal response format is as follows

| Name | Device Address | Function Code | Error Code | CRC |
|---|---|---|---|---|
| Byte | 1 | 1 | 1 | 2 |

Function code: 0x80 + Function code issued by the host. If the function code sent by the host is 01, the function code of the abnormal response is 0x81. If the function code delivered by the host is 03, the function code of the abnormal response is 0x83. By analogy.

Abnormal code: Indicates the type of communication error, please refer to the table below.

| Exception code | name | explanation |
|---|---|---|
| 01H | Function code | Illegal slave receives a function code that can not be executed. |
| 02H | illegal data address | 1. The address of the data is not recognized by the slave<br>2. The data address and the number of data synthesis address is invalid |
| 03H | Data value illegal | 1. The amount of data is out of range<br>2. The data length is wrong<br>3. Data is illegal |
| 04H | Slave device error | An unrecoverable error occurred while the slave execution host requested. |
| 05H | Confirm | that the slave has received the request to process the data, but requires a longer processing time, in order to avoid the host timeout error sent the confirmation response. The host then sends one A "query process completed" undecided whether the slave has completed processing. |
| 06H | Slave Device Busy | The slave is busy processing a long-term program command, requesting the host to send a message when the slave is idle. |
| 07H | Denied | When the slave can not execute the requested program function, the code returns a "unsuccessful programming request" message to the master using a 13 or 14 decimal code. The host should request a diagnostic error message from the slave. |
| 08H | Memory Parity Error | A parity error has been detected while the data in the extended memory has been read from the machine. The host resends the data request as requested by the slave. |

**6. CRC function to achieve**

CRC simple function is as follows:

```
unsigned short CRC16 (unsigned char * puchMsg, unsigned short usDataLen)
{
unsigned char uchCRCHi = 0xFF;        / * high CRC byte initialization * /
unsigned char uchCRCLo = 0xFF; / * low CRC byte initialization * /
unsigned uIndex; / * index in CRC loop * /
while (usDataLen--) / * Transfer message buffer * /
{
uIndex = uchCRCHi ^ * puchMsgg ++; / * Calculate CRC * /
uchCRCHi = uchCRCLo ^ auchCRCHi [uIndex];
uchCRCLo = auchCRCLo [uIndex];
}
return (uchCRCHi << 8 | uchCRCLo);
}

/ * CRC high byte table * /
static unsigned char auchCRCHi [] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
```

```
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40
};

/ * CRC low byte value table * /
static char auchCRCLo [] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06,
0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD,
0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4,
0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,
0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29,
0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED,
0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60,
0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,
0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA, 0xBE, 0x7E,
0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71,
0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
0x43, 0x83, 0x41, 0x81, 0x80, 0x40 };
```

**7. register address**

| No | Parameter | Parameter Type | Register Start Address (Decimal) | The number of registers |
|---|---|---|---|---|
| 1 | 1st Analog Input | Float | 1 | 2 register |
| 2 | 2nd Analog Input | Float | 3 | 2 register |
| 3 | 3rd Analog Input | Float | 5 | 2 register |
| 4 | 4th Analog Input | Float | 7 | 2 register |
| 5 | 5th Analog Input | Float | 9 | 2 Register |
| 6 | 6th Analog Input | Float | 11 | 2 Register |
| 7 | 7th Analog Input | Float | 13 | 2 Register |
| 8 | 8th Analog Input | Float | 15 | 2 Register |
| 9 | 9th Analog Input | Float | 17 | 2 Register |

| | | | | |
|---|---|---|---|---|
| 10 | 10th Analog Input | Float | 19 | 2 Register |
| 11 | 11th analog input | Float | 21 | 2 register |
| 12 | 12th Analog Input | Float | 23 | 2 Register |
| 13 | 13th Analog Input | Float | 25 | 2 Register |
| 14 | 14th Analog Input | Float | 27 | 2 register |
| 15 | 15th Analog Input | Float | 29 | 2 Register |
| 16 | 16th Analog Input | Float | 31 | 2 register |
| 17 | 17th Analog Input | Float | 33 | 2 register |
| 18 | 18th Analog Input | Float | 35 | 2 Register |

Note: 2 register addresses for each analog channel, a total of 4 bytes. 4 bytes of adjustable order, there are 1234,2143,3412,4321 in the configuration of a total of four kinds of order optional.

**8. examples of communication**

Example: Read the real-time value of analog input 2
Send data:
06 03 00 03 00 02 35 BC
Description:
06: instrument address (configuration can be changed)
03: Modbus 03 command
00 03: register address 3
00 02: Number of registers 2
35 BC: CRC check

Return data:
06 03 04 00 00 43 48 BD F5
Description:
06: instrument address
03: Modbus 03 command
04: Return data four bytes
00 00 43 48: Float, which means 200.0
BD F5: CRC check