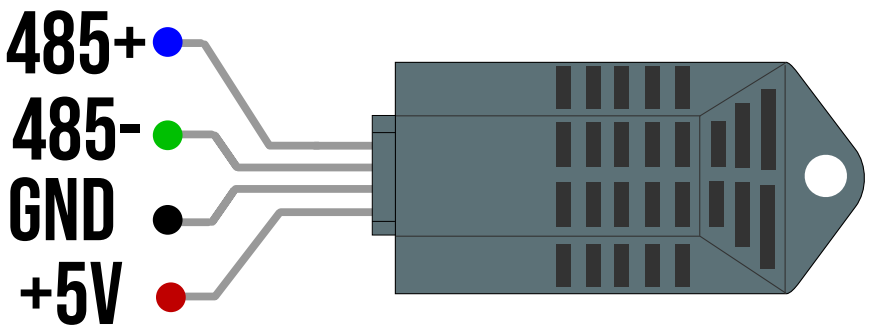
MODBUS Protocol（THM）

V2.5

Part Number HRTM-D20E, THM-D20E



1.Overview

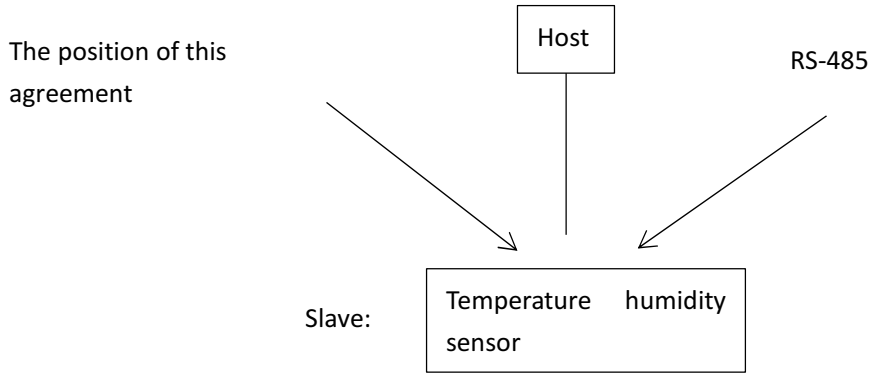
The communication protocol describes in detail the input and output commands, information and data of the temperature and humidity sensor for use and development by third parties.

1.1 Function of communication protocol

Enable information and data to be transmitted effectively between the upper computer (master station) and the temperature and humidity sensor, allowing access to all measurements of the temperature and humidity sensor.

The temperature and humidity sensor can collect the value of temperature and humidity in the field in real time, and has an rs-485 (rs-232) communication port, which can meet the requirements of small temperature and humidity monitoring system. The function and technical index of temperature and humidity sensor refer to the product specification.

The communication protocol of temperature and humidity sensor adopts MODBUS RTU protocol. This protocol specifies the communication protocol between the host and temperature and humidity sensor in the application layer. Its position in the application system is shown as follows:



1.2 Physical interface:

The main communication port of the upper computer is connected with the standard serial rs-485 (rs-232).

Information transmission is asynchronous, starting bit 1, data bit 8, stop bit 1, no check.

The default rate of data transmission is 9600b/s

2.MODBU RTU communication protocol Description

2. Basic rules of the agreement

The following rules determine the communication rules for devices in the loop controller and other serial communication loops.

1) all loops shall be communicated in the master/slave mode. In this way, information and data are transmitted between a single master station and an slave station (monitoring equipment).

2) the master station will initialize and control all information transmitted on the communication loop.

3) communication cannot start from a slave station at all costs.

4) all communication on the loop takes place in a "packing" manner. A package is a simple string (8 bits per string), containing up to 255 bytes in a package. The bytes that make up this package constitute the standard asynchronous serial data and are passed by means of 8-bit data bits, 1-bit stop bits, and no check bits. The serial data flow is generated by devices similar to those used in rs-232.

5) transmission on all return routes is divided into two packaging modes:

A) master/slave transfer

B) slave/master transfer

6) if the master station or any slave station receives a package containing unknown orders, the package will be ignored and the receiving station will not respond.

2. 2 description of data frame structure

Each data frame is composed as follows :(RTU mode)

Address function code data number data 1... Data n CRC16 bit check

3.Transmission format

(1) command message format

The host sends the temperature and humidity data command:

Address	Function code	Data start address high	Data start address low	data number high	data number low	CRC 16 bit check
xx	03	00	02	00	02	xxxx Low before high

Temperature and humidity data value returned from the slave sensor:

Address	Function code	Data start address high	Data start address low	data number high	data number low	CRC 16 bit check
00	03	00	00	00	01	xxxx Low before high

The slave sensor return address value:

Address	Function code	Bite length	Address high	Address low	CRC 16 bit check
00	03	02	00	xx	xxxx Low before high

The host sends address setup command:

Address	Function code	Write address high	Write address low	Operand high	Operand low	Bite length	Write content high	Write content low	CRC 16 bit check
00	10	00	00	00	01	02	00	xx	xxxx Low before high

The slave sensor return response value

Address	Function code	Write address high	Write address low	Operand high	Operand low	CRC 16 bit check
00	10	00	00	00	01	xxxx Low before high

(2) Frame format(10 bits)

Start bit	D0	D1	D2	D3	D4	D5	D6	D7	Stop bit

4.Basic Configuration

When reading the temperature and humidity sensor data, the upper computer should read the data at an interval of not less than 500ms and the recommended value is 1s.

5. Examples of commands:

Serial port Settings: asynchronous communication, starting bit 1, data bit 8, no validation, stop bit 1

The default data transmission rate is: 9600b/s

When the address is 1, read the temperature and humidity value :(CRC calibration low is before)

Upper computer send: 001 03 0002 0002 65 CB (read two analog quantities starting from the data starting address of 0002H)

Transmitter return: 001 03 04, temperature H, temperature L, humidity H, humidity L, CRC_L, CRC_H

Read address code value :(CRC check low position before)

Sending by upper computer: 0003, 0000, 0001, 85db (read one analog quantity from the data start address of 0000H)

Transmitter return: 0003 02 00, address L, CRC_L, CRC_H

Set new address "2" for the transmitter: (CRC calibration low is before) (address setting range: 1 ~ 254)

Upper computer send: 00 10 00 00 00 01 02 00 02 2A 01

Transmitter return: 00 10 00 00 00 00 01 00 18

Note:

The temperature and humidity data returned from the machine are respectively represented by two bytes, the high is before and the low is after.

The return data range is -32768 ~ 32767, and the actual temperature and humidity data need to divide the return value by 10.

The CRC16 bit verification is represented by two bytes, with the low position first and the high position second.

Address setting range: 1 ~ 254

For example:

Return humidity hexadecimal data: 0x0311 corresponds to decimal 785, indicating humidity of 78.5%RH

Return hexadecimal data: temperature 0 x00ff, corresponding decimal number 255, said the temperature is 25.5 ℃

Return hexadecimal data: temperature 0 x8064, highest for 1 negative, corresponding decimal - 100, said the temperature is 10.0 ℃

Appendix:

CRC high byte value table:

static 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, 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};

Calculation method of CRC function:

1. Preset 1 16-bit register to hex FFFF (i.e., all 1); Call this register the CRC register;

2. Lower the first 8-bit binary data (the first byte of the communication frame) with the 16-bit CRC register

8 bit different or, put the results in the CRC register;

3. Move the contents of the CRC register one bit to the right (low position) and fill the highest position with 0, and check the shift out after right move.

4. If the shift is 0: repeat step 3 (move one more bit to the right);

If the shift is 1: CRC register is different or different from polynomial A001 (1010, 0000, 0001);

5. Repeat steps 3 and 4 until you move to the right 8 times, so that all 8 bits of data are processed.

6. Repeat steps 2 to 5 to process the next byte of communication information frame.

7. After all the bytes of the communication information frame are calculated according to the above steps, the high and low 16-bit CRC register is obtained

Exchange of bytes;

8. The final CRC register content is: CRC code.

CRC function routine:

```
//*pushMsg is the array pointer variables to check， usDataLen is the number of data variables to checkvoid
CRC16(char *pushMsg,unsigned short usDataLen)
{
  char uchCRCHi=0xFF;                  // high CRC byte initialization
  char uchCRCLo=0xFF;                  // low CRC byte initialization
  unsigned int uIndex;                 // the index in CRC loops
  while(usDataLen--)                   //CRC table check function
    {
      uIndex    =uchCRCHi^*pushMsg++;//Calculate CRC
      uchCRCHi=uchCRCLo^auchCRCHi[uIndex];
      uchCRCLo=auchCRCLo[uIndex];
    }
  *pushMsg++=uchCRCHi;                 // Check data high after low
  *pushMsg=uchCRCLo;                   // Check data low before high
}
```