Modules Modbus RTU Protocol

1. Introduction

This document describes the Modbus RTU protocol, which is suitable for our company's sensor module communication needs to detect gas concentration and usage environment

2. MODBUS Protocol Description

2.1 Modbus address rules

Modbus is a master-slave communication mode, where the communication is initiated by the host and respond by slave with the corresponding address.

The host has no address, and the slave address range is: 1~247; 0 is the broadcast address. The slave address is unique on the Modbus serial bus.

2.2 Modbus protocol frame

The Modbus protocol frame consists of an address field, a function code, a data field, and a check code.

Table 2.1.1 General Modbus frame

Address field	function code	Data Field	Check code

The protocol frame is divided into two formats: RTU transmission mode and ASCII transmission mode.

2.3 RTU transmission mode

2.3.1 Byte Format

The byte contain: 1 start bit, 8 data bits (least significant bit first), no parity bit, 1 stop bit, a total of 10 bits.

Each character or byte is sent in the following order (from left to right): from LSB to MSB.

Table 2.3.1 Byte order in RTU transmission mode

Start bit	Data bits								Stop bits	Stop bits
1	2	3	4	5	6	7	8	9	10	11

2.3.2 RTU message frame

The RTU message frame includes: slave address, function code, data field, and CRC check.

The maximum length of an RTU message frame is 256 bytes, of which the maximum length of the data field is 252 bytes.

Table 2.3.2a RTU message frame

Format	Slave Address	Function Code	Data	CRC Check	
Number of bytes	1 byte	1 byte	0~252 bytes	2 bytes	
				Low Byte	High Byte

In RTU mode, idle intervals of at least 3.5 character times separate message frames.

The entire message frame must be sent as a continuous stream of characters.

If the idle interval between two characters is greater than 1.5 character times, the message frame is considered incomplete and the receiving station should discard the message frame.

Start	Slave Address	Function Code	Data	CRC Check	End
≥3.5 char	8-bit	8-bit	N*8 bits	16-bit	≥3.5 char

Table 2.3.2b RTU message frame transmission order

2.3.3 CRC Check

The CRC consists of two 8-bit bytes forming a 16-bit value.

The CRC field is appended to the message as the last field of the message. When appending,

the low-order byte of the field is appended first, and then the high-order byte of the field is appended. The CRC high-order byte is the last byte sent in the message.

Calculation of CRC:

The CRC calculation is started by preloading a 16-bit register with all 1s. Then, the 8-bit bytes in the subsequent message are calculated with the contents of the current register. Only the 8 data bits in each character are used to generate the CRC. The start bit, stop bit, and check bit are not used in the CRC calculation.

In CRC processing, each 8-bit character is XOR with the value in the register. Then, this result is shifted in the direction of the least significant bit (LSB), while the most significant bit (MSB) is felled with zero. The LSB is extracted and checked. If the LSB is 1, the value in the register is XOR with a fixed preset value; if the LSB is 0, no XOR operation is performed.

This process is repeated until 8 shifts have been completed. After the last (8th) shift is completed, the next 8-bit byte is XOR with the current value of the register, and the process is repeated 8 times as described above. After all bytes in the message have been calculated, the final value of the register is the CRC.

2. 4 Function Code

Table 2.5.1 only lists the function codes used in this protocol

Table 2.5.1 Function code list

Serial Number	Function Code	Illustrate
1	03H	Read Holding Registers
2	04H	Read multiple input registers
3	06H	Writing a Single Holding Register

2. 5 Error Codes

Table 2.4.1 Exception code details

Code	Illustrate	Remark
01H	Illegal function code	The requested function code is not supported by slave
02H	Illegal data address	The requested data address is not valid
03H	Illegal data value	The data value provided in the request is not valid or allowed by slave
04H	Slave device failure	Slave device encountered an internal error while processing the request
05H	Acknowledge	It has received the request but needs additional time to process it
06H	Slave device busy	Slave device to indicate that it is busy executing some other command
07H	Negative Response	The slave can not execute the program function specified in the request
08H	Storage parity error	The slave detected a parity error when reading the extended memory

2.6 Storage Area Identification

This protocol groups the storage area addresses into the following types of symbols: 3XXXX, 4XXXX. The grouping rules are shown in Table 2.5.1.

Register Number	Name	Format	Type	Storage unit address	function code
3XXXX	Input Register	Char	Read	30001 ~ 3XXXX	04H
4XXXX	Output Holding Registers	Char	Read/Write	40001 ~ 4XXXX	06H

Table 2.5.1 Storage area address identification grouping rules

2. 7 Register Function

2.7.1 Read holding register (function code: 0x03)

Read Holding Register Request

Function Code	1 byte	0x03
Initial Address	2 bytes	0x0000~0xFFFF
Number of registers	2 bytes	1~125(0x7D)

Read Holding Register Response

Function Code	1 byte	0x03
Byte Count	1 byte	2×N*
Register Value	N * × 2 bytes	
* N = number of registers		

Read Holding Register Error Response

Abnormal function code	1 byte	0x83
Exception code	1 byte	01 or 02 or 03 or 04

Example: Request to read holding registers [108~110].

Read Holding Register Example

Request		Reply	
Field Name	hexadecimal	Field Name	hexadecimal
Function code	03	function code	03
Starting address Hi	00	Byte Count	06
Starting address Lo	6B	Register [108] Hi	02
Number of registers Hi	00	Register [108] Lo	2B
Number of registers Lo	03	Register [109] Hi	00
		Register [109] Lo	00

		Register [110] Hi	00
		Register [110] Lo	64

Note:

1. Register [1] corresponds to address 0x0000;

2. Register [108] corresponds to address 0x006B.

2.7.2 Read input register (function code: 0x04)

Read Input Register Request

Function code	1 byte	0x04
Initial address	2 bytes	0x0000~0xFFFF
Number of registers	2 bytes	1~125(0x7D)

Read Input Register Response

Function code	1 byte	0x04
Byte Count	1 byte	2×N*
Register Value	N * × 2 bytes	
* N = number of registers		

Read Input Register Error Response

Abnormal function code	1 byte	0x84
Exception code	1 byte	01 or 02 or 03 or 04

Example: Request to read input register 9.

Read Input Register Example

Request		Reply	
Field Name	hexadecimal	Field Name	hexadecimal
Function	04	Function	04

Starting address Hi	00	Byte Count	02
Starting address Lo	08	Register [09] Hi	00
Number of registers Hi	00	Register [09] Lo	0A
Number of register Lo	01		

Note:

1. Address 0x0000 corresponds to register [1];

2. Address 0x0008 corresponds to register [9].

2.7.3 Write a single holding register (function code: 0x06)

Write Single Holding Register Request

Function code	1 byte	0x06
Holding register address	2 bytes	0x0000~0xFFFF
Holding register values	2 bytes	0x0000~0xFFFF

Write Single Holding Register Response

Function code	1 byte	0x06
Holding register address	2 bytes	0x0000~0xFFFF
Holding register values	2 bytes	0x0000~0xFFFF

Write Single Holding Register Error Response

Abnormal function code	1 byte	0x86
Abnormal code	1 byte	01 or 02 or 03 or 04

Example: Request to write 0x0003 to holding register [2].

Request		Reply	
Field Name	hex	Field Name	hex
Function code	06	Function	06

Holding register address Hi	00	Holding register address Hi	00
Holding register address Lo	01	Holding register address Lo	01
Holding register value Hi	00	Holding register value Hi	00
Holding register value Lo	03	Holding register value Lo	03

2.8 Protocol Application

The device can use RS485 as the physical interface.

Baud rate: 4800 bps, 9600 bps, 115200 bps; default 9600bps.

Function code application comparison table

Application Content	Type	Function code	Operate	Register address
Sensor analog signal	Character	04H	Read	3XXXX
Device Address	Character	03H	Read	4XXXX
		06H	Write	

2.9 Input register (address: 3XXXX) data content

Table 3.4.1 Equipment data

Input register address	Content	Size	Scope	Unit	Remark
3000 6	PM2.5	2Byte	0~999	ug/m^3	PM2.5 concentration
3000 7	TVOC	2Byte	0~ 50000	PPM*1000	TVOC concentration
3000 8	Formaldehyde	2Byte	0~ 3000	PPM*1000	HCHO concentration
3000 9	CO2	2Byte	2000-5000	PPM	CO2 concentration
300 12	Temperature	2Byte	-30-70	Celsius	temperature
300 13	Humidity	2Byte	0-99	%	humidity
300 36	PM2.5 level	2Byte	0~ 5	NA	0=Excellent 1=Good 2=Mild 3=Moderate

					4=Severe 5=Extreme
30 037	TVOC level	2Byte	0~ 5	NA	0=Excellent 1=Good 2=Mild 3=Moderate 4=Severe 5=Extreme
30 038	HCHO level	2Byte	0~ 5	NA	0=Excellent 1=Good 2=Mild 3=Moderate 4=Severe 5=Extreme
30 039	CO2 level	2Byte	0~ 5	NA	0=Excellent 1=Good 2=Mild 3=Moderate 4=Severe 5=Extreme
30106	PM2.5 sensor failure	2Byte	0~ 2	NA	0=No fault 1=Communication fault 2=Sensor abnormal fault
30107	TVOC sensor failure	2Byte	0~ 2	NA	0=No fault 1=Communication fault 2=Sensor abnormal fault
30108	HCHO sensor failure	2Byte	0~ 2	NA	0=No fault 1: Communication fault 2: Sensor abnormal fault
30109	CO2 sensor failure	2Byte	0~ 2	NA	0=No fault 1=Communication fault 2=Sensor abnormal fault
30111	temperature and humidity fault	2Byte	0~ 2	NA	0=No fault 1=Communication fault 2=Sensor abnormal fault

3.1 Holding Register (Address: 4XXXX)

Table 3.5.1 User setting quantity (40001)

Holding register address	content	size	scope	unit
40001	Device slave address code	2Byte	1-247	NA

The communication protocol is explained as follows:

Read the slave address code of the device (function code: 0x03)

Request frame (hexadecimal)

Address code	Function code	Initial address	Data length	Check high bit	Check low
0x01	0x03	0x9C 0x41	0x00 0x01	0xFA	0x4E

Response frame (hexadecimal) (e.g. read from the device)

Address code	Function code	Returns valid bytes	Address code	Check bit high	Check bit low
0x01	0x03	0x02	0x00 0x01	0x79	0x84

Write the slave address code of the device (function code: 0x06)

Address	Function	Write register address	Register value	Check bit high	Check bit low
0x01	0x06	0x9C 0x41	0x00 0x01	0x36	0x4E

Inquiry frame (hexadecimal)

Response frame (hexadecimal) (e.g. read from the device)

Address	Function	Byte Counter	Register value	Check bit high	Check bit low
0x01	0x06	0x02	0x00 0x01	0x79	0x48

Read device sensor data (function code: 0x04)

Address	Function	Register start address	Number of registers	Check bit high	Check bit low

0x01	0x04	0x75 0x36	0x00 0x05	0xCA	0x0B

Inquiry frame (hexadecimal)

Response frame (hexadecimal) (e.g. data read from the device)

Address	Function	Byte Counter	Register Value	Check high bit	Check bit low
0x01	0x04	0x0A	PM2.5: 0x00 0x19 TVOC: 0x00 0x32 HCHO: 0x00 0xFA CO2: 0x13 0x88 temperature: 0x00 0x00	0x39	0x94

Read the device sensor quality level (function code: 0x04)

Address	Function	Register start address	Number of registers	Check bit high	Check bit low
0x01	0x04	0x75 0x54	0x00 0x04	0xAA	0x15

Inquiry frame (hexadecimal)

Response frame (hexadecimal) (e.g., read the alarm level of the device)

Address	Function	Byte Count	Register Value	Check bit high	Check bit low
0x01	0x04	0x06	PM2.5 :0x00 0x01 TVOC :0x00 0x01 HCHO :0x00 0x02 CO2 :0x00 0x00	0xE4	0xAD

You can check the verification code at the following website

https://www.23bei.com/tool-59.html

The C language CRC16 check code is as follows:

static const uint16_t crc_tab16[256] = {

0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,

0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,

0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,

0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,

0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,

0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,

0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,

0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,

0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,

0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,

0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,

0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,

0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,

0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,

0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,

0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,

0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,

0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,

0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,

0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,

0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,

0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,

0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,

0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,

0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,

0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,

0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,

0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,

```
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040
};

uint16_t updateCrc16(uint16_t crc, uint8_t c)
{
uint16_t tmp, short_c;

short_c = 0x00FF & (uint16_t)c;
tmp = crc ^ short_c;
crc = (crc >> 8) ^ crc_tab16[tmp & 0xFF];
return crc;
}

uint16_t calcBuffCrc16(uint8_t *data, uint16_t len)
{
uint16_t i, crc16 = 0xFFFF;
for(i = 0; i < len; ++i) {
crc16 = updateCrc16(crc16, *(data+i));
}
return crc16;
}
```