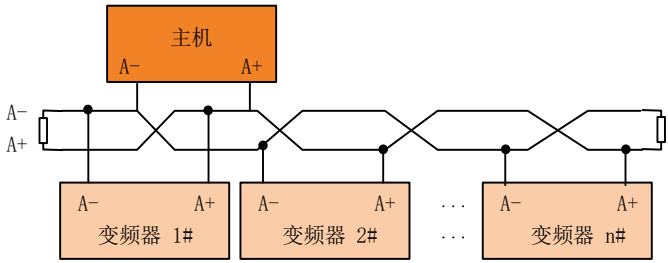
Chapter 55 MODBUS Communication Protocol

55.1 Applicable Scope

1. Applicable series: EM700 series
2. Applicable network: Support the “single-master multi-slave” communication network with MODBUS-RTU protocol and RS-485 bus.



55.2 Interface Mode

RS-485 asynchronous half-duplex communication mode, with the least significant bit sent first;

RS-485 network address: 1-247; 0 is the broadcast address;

Default data format of RS-485 terminal: 1-8-N-1[1] (options: 1-8-E-1, 1-8-O-1, 1-8-N-2, 1-8-E-2 and 1-8-O-2);

Default baud rate of RS-485 terminal: 9600bps (options: 4800bps, 19200bps, 38400bps, 57600bps and 115200bps)

It is recommended to use twisted-pair shielded cable as the communication cable to reduce the impact of external interference on communication.

[2]： *1-8-N-1, meaning 1 start bit - 8 characters per byte of data - no parity - 1 stop bit. E: even parity. O: odd parity.*

55.3 Protocol Format

55.3.1 Message format

As shown in Fig. 12-18, a standard MODBUS message includes a start tag, RTU (Remote Terminal Unit) message, and end tag.

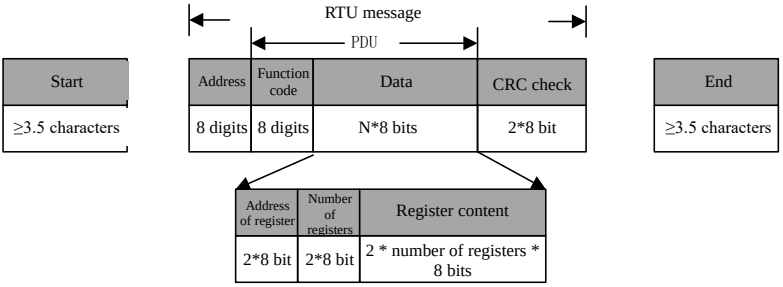


Fig. 12-18 Schematic Diagram of Message Frame in RTU Mode

The RTU message includes the address code, PDU (Protocol Data Unit) and CRC[3] check. The PDU includes the function code and data part (mainly including the register address, number of registers, register content and the like; the detailed definitions of function codes are different, as shown in 55.3.3**Function Code**).

[3]: *the low byte of CRC check is in front of the high byte.*

55.3.2 Address code

Address Range	Purpose
1-247	Slave
0	Broadcast

55.3.3 Function code

The classification of MODBUS function codes is shown in Fig. 12-19.

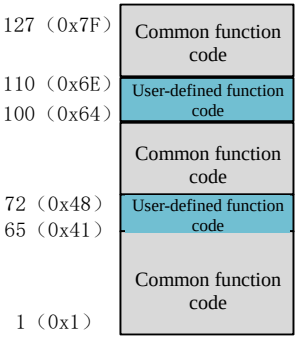


Fig. 12-19 Classification of MODBUS Function Codes

As shown in Table 12-26, EM700 series products mainly involve **common function codes**. As shown in , , 0x10: function code used to write multiple registers or commands

and 0x08: function code for diagnosis.

In addition, for some specific functions, such as register writing (RAM) without EEPROM storage, the **user-defined function codes** include, 0x41: function code used to write a single register or command (without saving) and 0x42: function code used to write multiple registers or commands (without saving).

55.3.4 When the abnormal valid data is received from a device, a related abnormality message will be returned (see 55.3.8 **Exception Response**). The abnormality function code is defined to distinguish the abnormal data from normal communication data. Corresponding to the normal request function code, the **abnormality function code = request function code + 0x80**.

Table 12-26 Function Code Definitions of EM700 series Product

Function code	Abnormality function code	Function
03	83	This function code is used to read multiple registers or status words.
41	C1	This function code is used to write a single register or command without saving.
42	C2	This function code is used to write multiple registers or commands without saving.
08	88	This function code is used for diagnosis.
06	86	This function code is used to write a single register or command.
10	90	This function code is used to write multiple registers or commands.

PDU parts are detailed in the following sections, depending on various functions.

1.1.1.1 0x03: function code used to read multiple registers or status words

In the remote terminal unit, this function code is used to read the content in the continuous block of the holding register. The request PDU describes the starting register address and the number of registers.

The register data in the response message is divided into two bytes in each register. The first byte of each register includes high-order bits and the second byte includes low-order bits.

● Request PDU

Function code	1 byte	**0x03**
Starting Address	2 bytes	0x0000 - 0xFFFF
Number of registers	2 bytes	1 - 16

- Response PDU

Function code	1 byte	**0x03**
Number of bytes	1 byte	2×N*
Register value	N*×2 bytes	

N* = number of registers

- Error PDU

Error code	1 byte	**0x83**
Exception code	1 byte	01, 02, 03 or 04

Below is an example of a request to read the registers F19.00 to F19.05 (relevant information about the last protection):

Request		Respond			
Domain name	(0x)	Domain name (normal)	(0x)	Domain name (abnormal)	(0x)
Function code	03	Function code	03	Function	83
Starting address Hi	13	Number of bytes	0C	Exception code	03 (example, the same below)
Starting address Lo	00	Register value Hi (F19.00)	00		
Number (Hi) of registers	00	Register value Lo (F19.00)	11		
Number (Lo) of registers	06	Register value Hi (F19.01)	00		
		Register value Lo (F19.01)	00		
		Register value Hi (F19.02)	00		
		Register value Lo (F19.02)	00		
		Register value Hi (F19.03)	01		
		Register value Lo (F19.03)	2C		
		Register value Hi (F19.04)	00		
		Register value Lo (F19.04)	00		
		Register value Hi (F19.05)	00		
		Register value Lo (F19.05)	00		

According to the returned data, the "17 (0011H): temperature sensor abnormality protection" of the inverter is enabled, in which the output frequency is 0.00Hz, the output current is 0.00A, the bus voltage is 300V (012CH), the acceleration and deceleration status is "standby", and the working time is 0 hour.

★: At present, the function code 0x03 of MODBUS protocol supports the reading of multiple function codes across groups. However, it is recommended not to read them across groups in the case of no special requirements, so the customer's software does not need to be upgraded after our products are upgraded.

1.1.1.2 0x41: function code used to write a single register or command (without saving)

In the remote terminal unit, this function code is used to write a single non-holding register.

The request PDU describes the address to be written to the register.

The normal response is the response made to the request, which is returned after the register content is written.

● Request PDU

Function code	1 byte	**0x41**
Address of register	2 bytes	0x0000 - 0xFFFF
Register value	2 bytes	0x0000 - 0xFFFF

● Response PDU

Function code	1 byte	**0x41**
Address of register	2 bytes	0x0000 - 0xFFFF
Register value	2 bytes	0x0000 - 0xFFFF

● Error PDU

Error code	1 byte	**0xC1**
Exception code	1 byte	See Table 22-29

Below is an example of a request to change the main frequency source A (7001H) to "-50.00%":

Request		Respond			
Domain name	(0x)	Domain name (normal)	(0x)	Domain name (abnormal)	(0x)
Function	41	Function	41	Function	C1

Register address Hi	70	Register address Hi	70	Exception code	03
Register address Lo	01	Register address Lo	01		
Register value Hi	EC	Register value Hi	EC		
Register value Lo	78	Register value Lo	78		

★ This function code cannot be used to change the parameters of the attribute "○" (it cannot be changed during operation). That is, only the parameters of the attribute "●" (it can be changed during operation) can be changed. Otherwise, the error code 1 will be returned.

1.1.1.3 0x42: function code used to write multiple registers or commands (without saving)

In the remote terminal unit, this function code is used to write consecutive non-holding register blocks (1 to 16 registers).

The value requested to be written is described in the request data field. The data of each register is divided into two bytes.

In the normal response, the function code, starting address and number of registers written will be returned.

● Request PDU

Function code	1 byte	**0x42**
Starting Address	2 bytes	0x0000-0xFFFF
Number of registers	2 bytes	1-16
Number of bytes	1 byte	2×N*
Register value	N*×2 bytes	

N* = number of registers

● Response PDU

Function code	1 byte	**0x42**
Starting Address	2 bytes	0x0000-0xFFFF
Number of registers	2 bytes	1-16

● Error PDU

Error code	1 byte	**0xC2**
Exception code	1 byte	See Table 22-29

Below is an example of a request to set the acceleration time 1 (F00.14) to 5.00 and

deceleration time 1 (F00.15) to 6.00:

Request		Respond			
Domain name	(0x)	Domain name (normal)	(0x)	Domain name (abnormal)	(0x)
Function	42	Function	42	Function	C2
Starting address Hi	00	Starting address Hi	00	Exception code	03
Starting address Lo	0E	Starting address Lo	0E		
Number (Hi) of registers	00	Number (Hi) of registers	00		
Number (Lo) of registers	02	Number (Lo) of registers	02		
Number of bytes	04				
Register value Hi (F00.14)	01				
Register value Lo (F00.14)	F4				
Register value Hi (F00.15)	02				
Register value Lo (F00.15)	58				

★ This function code cannot be used to change the parameters of the attribute "○" (it cannot be changed during operation). That is, only the parameters of the attribute "●" (it can be changed during operation) can be changed. Otherwise, the error code 1 will be returned.

1.1.1.4 0x08: function code for diagnosis

The Modbus function code 08 involves a series of tests to check the communication system between the client (master station) and server (slave station), or internal error statuses of the server.

The test to be executed is defined by the sub-function code fields of two bytes in the request. The server makes responses properly.

Copy the function codes and sub-function codes. Some diagnoses will enable the remote terminal unit to return the corresponding data through the data field in normal response.

Under normal circumstances, when the diagnosis function is sent to the remote terminal unit, the user program in this remote terminal unit will not be affected. Diagnosis can't access user logic such as discrete magnitude and the register. The error counter in the remote terminal unit can be remotely reset by applying some functions.

The main diagnosis function used by our company is line diagnosis (0000), which

is used to test the normal communication between the host and slave. The normal response to a request to return query data is to return the same data. At the same time, the function codes and sub-function codes are also copied.

- Request PDU

Function code	1 byte	**0x08**
Sub-function code	2 bytes	0x0000 - 0xFFFF
Data	2 bytes	0x0000 - 0xFFFF

- Response PDU

Function code	1 byte	**0x08**
Sub-function code	2 bytes	0x0000 - 0xFFFF
Data	2 bytes	0x0000 - 0xFFFF

- Error PDU

Error code	1 byte	**0x88**
Exception code	1 byte	See Table 22-29

- Sub-function code

Sub-function	Meaning	Data field (request)	Data field (response)
0000	Return query data	Any	Copy request data
…			

0000: return the data transferred in the request data field in the response. All messages should be consistent with the request message.

The following table is an example of requesting the remote terminal unit to return query data. The sub-function code 0000 is used. The returned data is sent in the two-byte data field (0xA537).

Request		Respond			
Domain name	(0x)	Domain name (normal)	(0x)	Domain name (abnormal)	(0x)
Function	08	Function	08	Function	88
Sub-function code Hi	00	Sub-function code Hi	00	Exception code	03
Sub-function code Lo	00	Sub-function code Lo	00		
Data Hi	A5	Data Hi	A5		
Data Lo	37	Data Lo	37		

1.1.1.5 0x06: function code used to write a single register or command

In the remote terminal unit, this function code is used to write a single holding register.

The request PDU describes the address to be written to the register.

The normal response is the response made to the request, which is returned after the register content is written.

● Request PDU

Function code	1 byte	**0x06**
Address of register	2 bytes	0x0000 - 0xFFFF
Register value	2 bytes	0x0000 - 0xFFFF

● Response PDU

Function code	1 byte	**0x06**
Address of register	2 bytes	0x0000 - 0xFFFF
Register value	2 bytes	0x0000 - 0xFFFF

● Error PDU

Error code	1 byte	**0x86**
Exception code	1 byte	See Table 22-29

★ The function code 0x06 cannot be used if modified frequently, in order to avoid damage to the inverter.

The user-defined function code 0x41 "change without saving" corresponds to the standard common function code 0x06. Its definition is the same as that of the corresponding standard function code (the same request, response and error PDU). The difference is that when the slave responds to this user-defined function code, the corresponding value of RAM is changed only and not stored in EEPROM (holding register).

For the function codes (e.g. F00.07) that are often modified, it is recommended to use the function code 0x41 (you can change the main frequency source A by directly setting 7001H, as detailed in 1.1.1.2 and 55.3.5), to avoid damage to the inverter. The specific operation is as follows.

Request		Respond	
Domain name	(0x)	Domain name (normal)	(0x)
Function	41	Function	41
Register address Hi	00	Register address Hi	00
Register address Lo	07	Register address Lo	07
Register value Hi	13	Register value Hi	13
Register value Lo	88	Register value Lo	88

Once the set frequency (F00.07) is set to 50.00Hz, the above data will be valid but not

be stored in EEPROM. That is, the inverter will run at 50.00Hz after change but at the frequency before change if powered on again.

1.1.1.6 0x10: function code used to write multiple registers or commands

In the remote terminal unit, this function code is used to write consecutive register blocks (1 to 16 registers).

The value requested to be written is described in the request data field. The data of each register is divided into two bytes.

In the normal response, the function code, starting address and number of registers written will be returned.

- Request PDU

Function code	1 byte	**0x10**
Starting Address	2 bytes	0x0000-0xFFFF
Number of registers	2 bytes	1-16
Number of bytes	1 byte	2×N*
Register value	N*×2 bytes	

N* = number of registers

- Response PDU

Function code	1 byte	**0x10**
Starting Address	2 bytes	0x0000-0xFFFF
Number of registers	2 bytes	1-16

- Error PDU

Error code	1 byte	**0x90**
Exception code	1 byte	See Table 22-29

Below is an example of a request to write 00 1 and 00 3 into two registers starting from F03.00 (i.e. setting the Y1 and Y2 output terminal function):

Request		Respond			
Domain name	(0x)	Domain name (normal)	(0x)	Domain name (abnormal)	(0x)
Function	10	Function	10	Function	90
Starting address Hi	03	Starting address Hi	03	Exception code	03
Starting address Lo	00	Starting address Lo	00		
Number (Hi) of registers	00	Number (Hi) of registers	00		

Number (Lo) of registers	02	Number (Lo) of registers	02
Number of bytes	04		
Register value Hi (F03.00)	00		
Register value Lo (F03.00)	01		
Register value Hi (F03.01)	00		
Register value Lo (F03.01)	03		

★ The function code 0x10 cannot be used if modified frequently, in order to avoid damage to the inverter, as detailed in Chapter 1.1.1.5.

55.3.5 Register address distribution

Table 12-27 Detailed Definition of Register Address of MODBUS Protocol

Address Space		Description	
Function code 0000H-6F63H		For the function code FXX.YY, the high order is hexadecimal of XX and the low order is hexadecimal of YY. For example, the address of F00.14 is 000EH (00D=00H, 14D=0EH).	
Function code (not saved after power-down) 8000H-EF63H		When the parameters are set with the function code 0x06 or 0x10, the function that "the settings are valid immediately and not saved after power-down" can be realized in the form of "original address +8000H". For example, the corresponding address of F00.14 is 800EH (=000EH+8000H).	
Control command (**write only**) 7000H -71FFH	7000H control word	0000H	Invalid command
		0001H	Forward running
		0002H	Reverse running
		0003H	JOG forward
		0004H	JOG reverse
		0005H	Deceleration to stop
		0006H	Stop the controller quickly
		0007H	Free stop
		0008H	Reset protection
		0009H	+/- input switching
		000BH	JOG stop
		Others to 00FFH	Reserved
	7001H	Communication percentage setting of main channel frequency A	-100.00%-100.00% (100% = maximum frequency)
	7002H	Communication percentage setting of auxiliary	-100.00%-100.00% (100% = maximum frequency)

		channel frequency B		
	7004H	Communication setting of process PID setting	-100.00%-100.00%	
	7005H	Communication setting of process PID feedback	-100.00%-100.00%	
	7006H	Voltage setting of VF separation mode	0.00%-100.00% (digital setting reference)	
	7007H-7009H	Reserved		
	700AH	Communication percentage setting of upper frequency limit	0.00%-200.00% (digital setting reference)	
	700CH	Linear speed input for inertia compensation	0.00% -100.00% (digital setting reference)	
	700DH-700EH	Reserved		
	700FH	Master-slave communication setting	-100.00%-100.00% (maximum reference)	
	7010H-7013H	Reserved		
	7014H	External protection	Protection input of external device (including option card)	
	7015H	Communication setting of main channel frequency A	0.00 to maximum frequency	
	7016H	Communication setting of auxiliary channel frequency B	0.00 to maximum frequency	
	7017H	Communication setting of upper frequency limit	0.00 to maximum frequency	
	7019H	Communication setting of upper torque limit of speed control	0.0-250.0% (based on 100.0% or direct sending)	
	701AH	Reserved		
	701CH-71FFH	Reserved		
Working status	7200H status word 1	Bit7-0 running status	00H	Parameter setting
			01H	Slave running

7200H -73FFH			02H	JOG running
			03H	Self-learning running
			04H	Slave stop
			05H	JOG stop
			06H	Protection status
			07H	Factory self-inspection
			08H-0FFH	Reserved
		Bit15-8 protection information	00H	Normal running of inverter
			xxH	Inverter protection status, where “xx” is the protection code
	7201H status word 2	Bit0 setting direction	1	- setting is valid
			0	+ setting is valid
		Bit1 running direction	1	Reverse frequency output
			0	Forward frequency output
		Bit3-2 running mode	00	Speed control mode
			10	Reserved
			11	Reserved
		Bit4 parameter protection	1	Valid parameter protection
			0	Invalid parameter protection
		Bit6-5	Reserved	
		Bit8-7 setting mode	00	Keyboard control
			01	Terminal control
			10	Communication control
			11	Reserved
		Bit9	Reserved	
		Bit10 warning	0	No warning
			1	Warning status (see 7230H for details)
		Bit15-10	Reserved	
	7202H monitoring frequency +/- status word 1 (1: -; 0: +)	Bit0	Output frequency	
		Bit1	Input frequency	
		Bit2	Synchronization frequency	
		Bit3	Reserved	
		Bit4	Estimate feedback frequency	
		Bit5	Estimated slip frequency	
		Bit6	Load rate	
		Bit15-7	Reserved	
	7203H	Output frequency		

	7204H	Output voltage								
	7205H	Output power								
	7206H	Running speed								
	7207H	Bus voltage								
	7208H	Output torque								
	7209H	Digital input 1	15	14	13	12	11	10	9	8
			*	*	*	*	*	*	*	*
			7	6	5	4	3	2	1	0
			*	*	*	*	X4	X3	X2	X1
	720AH	Digital input 2	15	14	13	12	11	10	9	8
			VX8	VX7	VX6	VX5	VX4	VX3	VX2	VX1
			7	6	5	4	3	2	1	0
			*	*	*	*	*	*	Reserved	AI1
	720BH	Digital output 1	15	14	13	12	11	10	9	8
			*	*	*	*	*	*	*	*
			7	6	5	4	3	2	1	0
			*	*	*	*	*	Y1	*	R1
	720CH	Digital output 2	15	14	13	12	11	10	9	8
			VY8	VY7	VY6	VY5	VY4	VY3	VY2	VY1
			7	6	5	4	3	2	1	0
			*	*	*	*	*	*	*	*
	720DH	Previous two protections								
	720EH	Previous three protections								
	720FH	Last protection								
	7210H	Output frequency of the last protection								
	7211H	Output current of the last protection								
	7212H	Bus voltage of the last protection								
	7213H	Running status of the last protection								
	7214H	Working time of the last protection								
	7215H	Set acceleration time								
	7216H	Set deceleration time								
	7217H	Cumulative length								
	7218H	Reserved								
	7219H	UP/DOWN offset frequency symbol (0/1: +/-)								
	7224H	Output current								
	7225H	Set frequency								
	7228H	Cumulative power-on time								
	722FH	Fault No.								
	7230H	Warning number	0: no warning; others: current warning sign							
	Other-73FFH	Reserved								
Product information	7500H	Performance software S/N 1	Corresponding to the function code F12.22							
	7501H	Performance software	Corresponding to the function code							

7500H ~ 75FFH		S/N2	F12.23
	7502H	Functional software S/N 1	Corresponding to the function code F12.24
	7503H	Functional software S/N 2	Corresponding to the function code F12.25
	7504H	Keyboard software serial number 1	Corresponding to the function code F12.26
	7505H	Keyboard software serial number 2	Corresponding to the function code F12.27
	7506H	Serial No. 1	Corresponding to the function code F12.28
	7507H	Serial No. 2	Corresponding to the function code F12.29
	7508H	Serial No. 3	Corresponding to the function code F12.30
	7509H-75FFH	Reserved	
Others	Reserved		

55.3.6 Definition of frame data length

The PDU part of the RTU frame of the MODBUS message is able to read/write 1-16 registers. For different function codes, the actual length of the RTU frame varies, as detailed in Table 12-28.

Table 12-28 Correspondence between RTU Frame Length and Function Code

Function code (0x)	RTU frame length (bytes)			Maximum length (Byte)
	Request	Normal response	Exception response	
03	8	$5+2N_r$[4]	5	37
41 (06)	8	8	5	8
08	8	8	5	8
42 (10)	$9+2N_w$[5]	8	5	41

[4]： $N_r \leq 16$, indicating the number of requests to read registers;

[5]： $N_w \leq 16$, indicating the number of requests to write registers.

[6]： $N_w+N_r \leq 16$;

55.3.7 CRC check

The low byte of CRC check is in front of the high byte.

The transmitter first calculates the CRC value, which is included in the sent message. Upon receiving the message, the receiver will recalculate the CRC value and compare the calculated value with the received CRC value. If the two values are not equal, it means that there is an error in the sending process.

Calculation process of CRC check:

(1) Define a CRC register and assign an initial value, FFFFH.

(2) Perform the XOR calculation with the first byte of the transmitted message and the value of the CRC register, and store the result in the CRC register. Starting from the address code, the start bit and stop bit are not involved in calculation.

(3) Extract and check the LSB (the least significant bit of the CRC register).

(4) If the LSB is 1, each bit of the CRC register is shifted to the right by one bit, and the most significant bit is supplemented by 0. Perform the XOR calculation of the value of the CRC register and A001H, and store the result in the CRC register.

(5) If the LSB is 0, each bit of the CRC register is shifted to the right by one bit, and the most significant bit is supplemented by 0.

(6) Repeat the steps 3, 4, and 5 until 8 shifts are completed.

(7) Repeat the steps 2, 3, 4, 5 and 6 to process next byte of the transmitted message, until all bytes of the transmitted message are processed. until all bytes of information are processed and transmitted.

(8) After the calculation, the content of the CRC register is the value of CRC check.

(9) In a system with limited time resources, it is recommended to perform CRC check by the table lookup method.

The simple function of CRC is as follows (programmed in C language):

```
unsigned int CRC_Cal_Value(unsigned char *Data, unsigned char Length)
{
    unsigned int crc_value = 0xFFFF;
    int i = 0;
    while(Length--)
    {
   crc_value ^= *Data++;
   for(i=0;i<8;i++)
   {
      if(crc_value & 0x0001)
      {
         crc_value = (crc_value>>1)^ 0xa001;
```

```
                }
        else
        {
            crc_value = crc_value>>1;
                }
      }
   }
   return(crc_value);
}
```

This only describes the theory of CRC check and requires a long execution time. Especially when the check data is long, the calculation time will be too long. Thus, the following two table lookup methods are applied for 16-bit and 8-bit controllers, respectively.

- CRC16 lookup table for the 8-bit processor: (The high byte in the final result of this program is in front. Please reverse it during sending.)

```
const Uint8 crc_l_tab[256] = {
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
```

```
};
constUint8 crc_h_tab[256] = {
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
};
Uint16CRC(Uint8 * buffer, Uint8 crc_len)
{
   Uint8  crc_i,crc_lsb,crc_msb;
   Uint16 crc;
   crc_msb = 0xFF;
   crc_lsb = 0xFF;
   while(crc_len--)
   {
      crc_i = crc_lsb ^ *buffer;
      buffer ++;
      crc_lsb = crc_msb ^ crc_l_tab[crc_i];
      crc_msb = crc_h_tab[crc_i];
```

```
    }
    crc = crc_msb;
    crc = (crc << 8) + crc_lsb;
    return crc;
}
```

● CRC16 lookup table for the 16-bit processor: (The high byte in the final result of this program is in front. Please reverse it during sending.)

```
const Uint16 crc_table[256] = {
0x0000,0xC1C0,0x81C1,0x4001,0x01C3,0xC003,0x8002,0x41C2,0x01C6,0xC006
,0x8007,0x41C7,0x0005,0xC1C5,0x81C4,0x4004,0x01CC,0xC00C,0x800D,0x41CD
,0x000F,0xC1CF,0x81CE,0x400E,0x000A,0xC1CA,0x81CB,0x400B,0x01C9,0xC009
,0x8008,0x41C8,0x01D8,0xC018,0x8019,0x41D9,0x001B,0xC1DB,0x81DA,0x401A
,0x001E,0xC1DE,0x81DF,0x401F,0x01DD,0xC01D,0x801C,0x41DC,0x0014,0xC1D4
,0x81D5,0x4015,0x01D7,0xC017,0x8016,0x41D6,0x01D2,0xC012,0x8013,0x41D3
,0x0011,0xC1D1,0x81D0,0x4010,0x01F0,0xC030,0x8031,0x41F1,0x0033,0xC1F3
,0x81F2,0x4032,0x0036,0xC1F6,0x81F7,0x4037,0x01F5,0xC035,0x8034,0x41F4
,0x003C,0xC1FC,0x81FD,0x403D,0x01FF,0xC03F,0x803E,0x41FE,0x01FA,0xC03A
,0x803B,0x41FB,0x0039,0xC1F9,0x81F8,0x4038,0x0028,0xC1E8,0x81E9,0x4029
,0x01EB,0xC02B,0x802A,0x41EA,0x01EE,0xC02E,0x802F,0x41EF,0x002D,0xC1ED
,0x81EC,0x402C,0x01E4,0xC024,0x8025,0x41E5,0x0027,0xC1E7,0x81E6,0x4026
,0x0022,0xC1E2,0x81E3,0x4023,0x01E1,0xC021,0x8020,0x41E0,0x01A0,0xC060
,0x8061,0x41A1,0x0063,0xC1A3,0x81A2,0x4062,0x0066,0xC1A6,0x81A7,0x4067
,0x01A5,0xC065,0x8064,0x41A4,0x006C,0xC1AC,0x81AD,0x406D,0x01AF,0xC06F
,0x806E,0x41AE,0x01AA,0xC06A,0x806B,0x41AB,0x0069,0xC1A9,0x81A8,0x4068
,0x0078,0xC1B8,0x81B9,0x4079,0x01BB,0xC07B,0x807A,0x41BA,0x01BE,0xC07E
,0x807F,0x41BF,0x007D,0xC1BD,0x81BC,0x407C,0x01B4,0xC074,0x8075,0x41B5
,0x0077,0xC1B7,0x81B6,0x4076,0x0072,0xC1B2,0x81B3,0x4073,0x01B1,0xC071
,0x8070,0x41B0,0x0050,0xC190,0x8191,0x4051,0x0193,0xC053,0x8052,0x4192
,0x0196,0xC056,0x8057,0x4197,0x0055,0xC195,0x8194,0x4054,0x019C,0xC05C
,0x805D,0x419D,0x005F,0xC19F,0x819E,0x405E,0x005A,0xC19A,0x819B,0x405B
,0x0199,0xC059,0x8058,0x4198,0x0188,0xC048,0x8049,0x4189,0x004B,0xC18B
,0x818A,0x404A,0x004E,0xC18E,0x818F,0x404F,0x018D,0xC04D,0x804C,0x418C
```

```
,0x0044,0xC184,0x8185,0x4045,0x0187,0xC047,0x8046,0x4186,0x0182,0xC042
,0x8043,0x4183,0x0041,0xC181,0x8180,0x4040};
Uint16 CRC16(Uint16 *msg , Uint16 len){
   Uint16 crcL = 0xFF , crcH = 0xFF;
   Uint16 index;
   while(len--){
      index = crcL ^ *msg++;
      crcL = ((crc_table[index] & 0xFF00) >> 8) ^ (crcH);
      crcH = crc_table[index] & 0xFF;
   }
   return (crcH<<8) | (crcL);
}
```

55.3.8 Exception response

When the master station sends a request to the slave station, the master station expects a normal response. Query of the master station may result in one of the following four events:

- If a request without communication error is received from the slave station and can be processed properly, a normal response will be returned by the slave station.
- If the slave station does not receive a request due to communication errors, no message will be returned. This will be regarded as a timeout by the slave station.
- If the slave station receives a request but detects a communication error (parity, address, frame error, etc.), no response will be returned. This will be regarded as a timeout by the slave station.
- If the slave station receives a request without communication error but cannot process the request (e.g. a request to read the non-existent register), the slave station will return an exception response and the master station will be informed of the actual error.

The exception response message has two fields different from those of the normal response:

- Function code field: In the normal response, the slave station copies the function code of the original request in the corresponding function code field. The MSB values

of all function codes are 0. In the exception response, the MSB of the function code is set to 1 by the slave station. That is, the exception response function code = normal response function code + 0x80.

- Data field: The slave station can return the data from the data field in the normal response and exception code in the exception response. The defined exception codes are detailed in the Table 22-29 Definitions of Exception Codes Definitions of Exception Codes.

Table 22-29 Definitions of Exception Codes

Exception code	Item	Meaning
01H	Illegal function	The function code received by the slave station (inverter) is beyond the configured range (see).
02H	Illegal data address	The data address received by the slave station (inverter) is not allowed. In particular, the combination of the start address of the register and the transmission length is invalid (see).
03H	Illegal data frame	The slave station (inverter) has detected the incorrect query data frame length or CRC check.
04H	Slave protection	When the slave station (inverter) tries to execute a requested operation, an unrecoverable error occurs. This may be caused by the logic error, failure to write to the EEPROM, etc.
05H	Data over-range	The data received by the slave station (inverter) is not between the minimum and maximum values of the corresponding register.
06H	Parameter read-only	The current register is read-only and cannot be written.
07H	Unchangeable parameter in running	When the inverter is in the running status, the current register cannot be written. If necessary, please shut down the inverter.
08H	Parameter protection by password	The current register is protected by a password.

55.4 Protocol Description

55.4.1 Definition of inter-frame and intra-frame time interval

A complete MODBUS message contains not only the necessary data units, but also the starting and ending tags. Thus, as shown in Fig. 12-18 or Fig. 22-20, the idle level with a transmission time of 3.5 characters or more is defined as the starting and ending tag. If there is an idle level with a transmission time of more than 1.5 characters during message transmission, the transmission will be deemed exceptional.

Specific starting/ending and exception intervals are related to the baud rate, as detailed in Table 22-30. If the baud rate is 9600bps and the sampling period is 1ms, the starting and ending time interval is the idle level of 4ms or more (3.5×10/9600=3.64≈4), and the exceptional data interval is the idle level in which the interval of data bits of one frame is greater than or equal to 2ms (1.5×10/9600=1.56≈2) and less than 4m (the idle level of normal data bits is less than or equal to 1ms).

Table 22-30 Correspondence between Time Interval and Baud Rate (t_{adjust}=1ms)

Baud Rate (bps)	Starting and ending time interval $T_{interval}$ (t_{adjust})	Exception interval $T_{exception}$ (t_{adjust})	Remarks
4800	8	4	The idle level of 3ms or less is allowed for a normal frame. When the idle level is 8ms or greater, it indicates the end of a frame of data.
9600	4	2	The idle level of 1ms or less is allowed for a normal frame. When the idle level is 4ms or greater, it indicates the end of a frame of data.
19200	2	1	The idle level of less than 1ms is allowed for a normal frame. When the idle level is 2ms or greater, it indicates the end of a frame of data.
Higher	**1**	**1**	**When an idle level of 1ms appears, it indicates the end of a frame.**

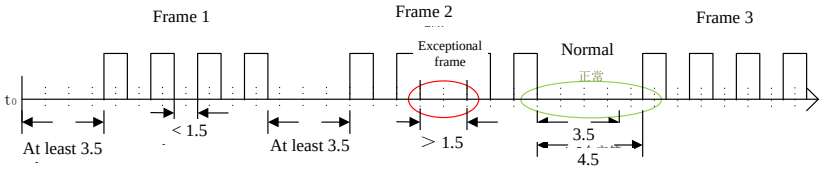


Fig. 22-20 Schematic Diagram of Normal and Exceptional Data Frames

55.4.2 Data frame processing

Upon receiving a frame data, the system will first perform preprocessing to determine whether it is a legal frame sent to this machine and check whether the data is correct, followed by final processing. If the received frame is not legal, the data will not be sent back. If the received frame is legal but incorrect, the corresponding exceptional message frame will be sent back.

Legal frame: Meet the address (local or broadcast) and length (not less than 3) requirements.

Correct frame: It is a legal frame with a correct memory address. The memory content

is within the defined range and can be processed at present.

55.4.3 Response delay

The response delay (depending on the function code F10.04) is defined as the time interval from the reception of valid data frame[1] (data in the RS-485 network, different from the command sent by the keyboard) to data parsing and return. Since the starting and ending characters are defined in the standard protocol, it is impossible to avoid response delay, at least "3.5-character time interval + 1 ms (chip stabilization time of 485 protocol, t_{wait2})". The specific minimum time interval is related to the baud rate. If the baud rate is 9600bps, the minimum response delay is 5ms (3.5×10/9600+1=4.64≈5).

If the communication data involves EEPROM operation, the time interval will be longer.

[7]: Valid data frame: Sent by the external master station (not keyboard) to this machine. The function code, length and CRC of the data are correct.

Fig. 22-21 shows the data sending segment (t_{send}), sending end segment (t_{wait1}), 75176-to-sending wait segment (t_{wait2}), data return segment (t_{return}), and 75176-to-receiving wait segment (t_{wait3}).

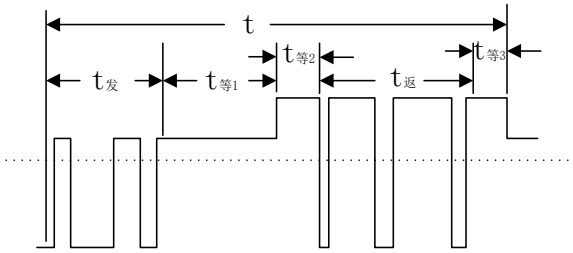


Fig. 22-21 Timing Parse Diagram of Complete Data Frame

55.4.4 Communication timeout

The communication time interval △t is defined as the period from the previous reception of valid data frames by the slave station (inverter) to next reception of valid data frames. If △t is greater than the set time (depending on the function code F10.03; this function is invalid if set to 0), it will be regarded communication timeout.

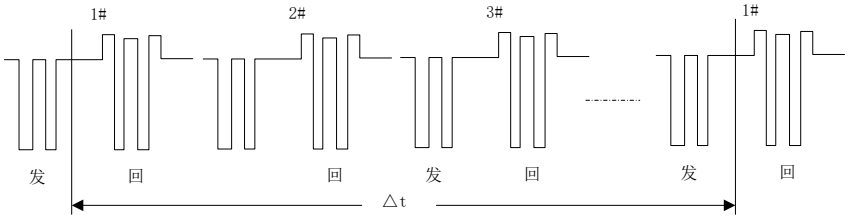


Fig. 22-22 Schematic Diagram of 485 Network Link Data

55.5 Examples

1) Forward running of inverter

Send: **01 41 70 0000 01 E6 C5**

Return: 01 41 70 0000 01 E6 C5 (normal)

Return: 01 C1 04 70 53 (exception, assuming a slave protection)

	Send		Normal Return		Exception Return	
*	Frame header	≥3.5 characters (idle)				
1	Address	01	Address	01	Address	01
2	Function code	41	Function code	41	Function code	C1
3	Register address Hi	70	Register address Hi	70	Exception code	04 (assumption)
4	Register address Lo	00	Register address Lo	00	CRC check Lo	70
5	Register value Hi	00	Register value Hi	00	CRC check Hi	53
6	Register value Lo	01	Register value Lo	01		
7	CRC check Lo	E	CRC check Lo	E6		
8	CRC check Hi	C	CRC check Hi	C5		
*	Tail	≥3.5 characters (idle)				

2) Free stop of inverter

Send: 01 41 70 0000 07 66 C7

Return: 01 41 70 0000 07 66 C7 (normal)

Return: 01 C1 04 70 53 (exception, assuming a slave protection)

	Send		Normal Return		Exception Return	
*	Frame header	≥3.5 characters (idle)				
1	Address	01	Address	01	Address	01

2	Function code	41	Function code	41	Function code	C1
3	Register address Hi	70	Register address Hi	70	Exception code	04 (assumption)
4	Register address Lo	00	Register address Lo	00	CRC check Lo	70
5	Register value Hi	00	Register value Hi	00	CRC check Hi	53
6	Register value Lo	07	Register value Lo	07		
7	CRC check Lo	66	CRC check Lo	66		
8	CRC check Hi	C	CRC check Hi	C7		
*	Tail	≥3.5 characters (idle)				

3) Command word for change of set frequency (e.g. 50.00Hz/1388H) (F00.04=7)

Send: 01 41 70 15 13 88 3B 97

Return: 01 41 70 15 13 88 3B 97 (normal)

Return: 01 C1 04 70 53 (exception, assuming a slave protection)

	Send		Normal Return		Exception Return	
*	Frame header	≥3.5 characters (idle)				
1	Address	01	Address	01	Address	01
2	Function code	41	Function code	41	Function code	C1
3	Register address Hi	70	Register address Hi	70	Exception code	04 (assumption)
4	Register address Lo	15	Register address Lo	15	CRC check Lo	70
5	Register value Hi	13	Register value Hi	13	CRC check Hi	53
6	Register value Lo	88	Register value Lo	88		
7	CRC check Lo	3	CRC check Lo	3B		
8	CRC check Hi	97	CRC check Hi	97		
*	Tail	≥3.5 characters (idle)				

1) Read the information of last protection (read the function codes F19.00-F19.05)

Send: 01 03 13 00 00 06 C1 4C

Return: 01 03 0C 00 11 00 00 00 00 01 2C 00 00 00 00 53 5B (normal)

Return: 01 83 04 40 F3 (exception, assuming a slave protection)

	Send		Normal Return		Exception Return	
*	Frame header	≥3.5 characters (idle)				
1	Address	01	Address	01	Address	01
2	Function code	03	Function code	03	Function	83

3	Starting address Hi	13	Number of bytes	0C	Exception	04 (assumption)
4	Starting address Lo	00	Register value Hi (F19.00)	00	CRC check	40
5	Number (Hi) of registers	00	Register value Lo (F19.00)	11	CRC check	F3
6	Number (Lo) of registers	06	Register value Hi (F19.01)	00		
7	CRC check Lo	C	Register value Lo (F19.01)	00		
8	CRC check Hi	4	Register value Hi (F19.02)	00		
9			Register value Lo (F19.02)	00		
10			Register value Hi (F19.03)	01		
11			Register value Lo (F19.03)	2C		
12			Register value Hi (F19.04)	00		
13			Register value Lo (F19.04)	00		
14			Register value Hi (F19.05)	00		
15			Register value Lo (F19.05)	00		
16			CRC check Lo	53		
17			CRC check Hi	5B		
*	Tail	≥3.5 characters (idle)				

2) Check whether the line is connected.

Send: 01 08 00 00 AA 55 5E 94

Return: 01 08 00 00 AA 55 5E 94 (normal)

Return: 01 88 04 47 C3 (exception, assuming a slave protection)

	Send		Normal Return		Exception Return	
*	Frame header	≥3.5 characters (idle)				
1	Address	01	Address	01	Address	01
2	Function	08	Function	08	Function code	88

3	Sub-function code Hi	00	Sub-function code Hi	00	Exception code	04 (assumption)
4	Sub-function code Lo	00	Sub-function code Lo	00	CRC check Lo	47
5	Data Hi	AA	Data Hi	AA	CRC check Hi	C3
6	Data Lo	55	Data Lo	55		
7	CRC check Lo	5E	CRC check Lo	5E		
8	CRC check Hi	94	CRC check Hi	94		
*	Tail	≥3.5 characters (idle)				

3) Change the carrier frequency (F00.23) to 4.0kHz. (use the function code 0x06 as such function codes are expected to be stored in EEPROM after change)

Send: 01 06 00 17 00 28 39 D0

Return: 01 06 00 17 00 28 39 D0 (normal)

Return: 01 86 04 43 A3 (exception, assuming a slave protection)

	Send		Normal Return		Exception Return	
*	Frame header	≥3.5 characters (idle)				
1	Address	01	Address	01	Address	01
2	Function code	06	Function code	06	Function code	86
3	Register address Hi	00	Register address Hi	00	Exception code	04 (assumption)
4	Register address Lo	17	Register address Lo	17	CRC check Lo	43
5	Register value Hi	00	Register value Hi	00	CRC check Hi	A3
6	Register value Lo	28	Register value Lo	28		
7	CRC check Lo	39	CRC check Lo	39		
8	CRC check Hi	D0	CRC check Hi	D0		
*	Tail	≥3.5 characters (idle)				