Wuxi Super Laser Technology Co.,Ltd

Handheld welding communication protocol

V1.5

Modify the record:

Version Number	Major Changes	Draft/modify date	Notes
V1.0	First Release	2021/7/7	
V1.1	Add some examples	2021/7/10	
V1.2	Supplementary Notes	2021/11/01	
V1.3	Add hardware interface definitions Update the document format	2021/11/27	
V1.4	Added function code 0x06 for writing to a single register; Change the CRC to have the lower byte first and the higher byte second; Add definitions for the address of the feeder-related registers.	2022/9/1	This protocol supports software versions 557 and above
V1.5	Add description of hardware interface definitions for the V6.1 version of the main control board; Add description of hardware interface definitions for V6.2 and later versions of the main control board.	2023/11/01	This protocol supports software versions 557 and above. The new full protocol is adopted for v825 and is also compatible with this protocol

Contents

Ⅰ.Hardware interface definitions

The external communication interface of our handheld soldering products is RS485 interface, as shown in Figures 1.1 and 1.2, which are the hardware interface diagrams corresponding to hardware versions V6.1 and V6.2 respectively. Tables 1.1, 1.2 and 1.3 list the interface definitions of control boxes for different hardware versions respectively:

Table 1.1 Interface Definitions for Hardware versions V5.81 and below

Interface for V5.81 and below versions : DB9 female socket		
Pin numbers	Signal	Function Description
1	RS485_B	RS485 Signal B
2	RS232_TXD	For internal use, it is recommended to leave it suspended
3	RS232_RXD	For internal use, it is recommended to hang
4	NC	Not used
5	GND	to
6	RS485_A	RS485 signal A
7	NC	Not used
8	NC	Not used
9	+5V	Not used

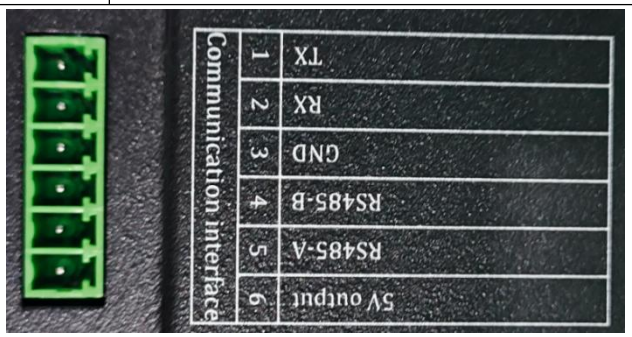

Figure 1.1 Diagram of external communication interface for hardware version V6.1

Table 1.2 Interface Definitions for Hardware Version V6.1

V6.1 Interface : 15EDGV-3.81mm-6Pin straight pin socket		
Pin numbers	Signal	Function Description
1	TX	For internal use, it is recommended to hang
2	RX	For internal use, it is recommended to hang
3	GND	to
4	RS485_B	RS485 Signal B
5	RS485_A	RS485 signal A
6	5V output	Reserve. It is recommended to leave it hanging

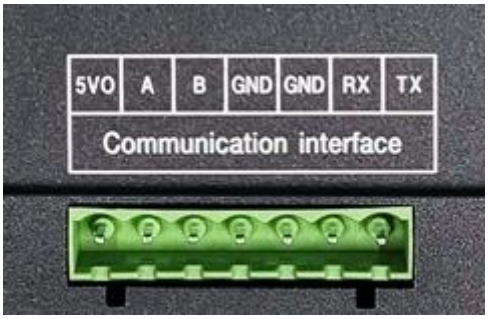

Figure 1.2 Diagram of the external communication interface for hardware version V6.2 and above

Table 1.3 Interface Definitions for Hardware versions V6.2 and above

Interface for version 6.2 and above:KF2EDGVC-5.08mm - 7-pin straight pin socket		
Pin numbers	Signal	Function Description
1	5VO	Reserve. It is recommended to leave it hanging
2	A	RS485 signal A
3	B	RS485 signal B
4	GND	GND
5	GND	GND
6	RX	For internal use, it is recommended to hang
7	TX	For internal use, it is recommended to hang

Note: Wire according to the corresponding interface definition table based on the hardware version.

Ⅱ.Communication format

This protocol is compatible with the Modbus RTU specification.

2.1 Basic parameters of the communication module

The basic parameters of the communication module are shown in Table 2.1:

Table 2.1 Basic Parameters of Communication Modules

Encoding	8-bit binary
Data bits	8 bits
Parity bits	no
Stop position	1 bit
Baud rate	19200 bit/s

2.2 Register address definition

Register address definitions are shown in Table 2.2:

Table 2.2 Handheld Solder Register Address Definition Table

NO.	Function name	Data Length (Bytes)	Data type	Data Range	Register address	R/W Properties
Handheld weld-related parameters						
1	Scan speed	2	Unsigned Enlarge the transfer by 10 times, that is, one decimal place	2~6000 mm/s	0x0000	R/W
2	Scan width	2	Unsigned Enlarge the transfer by 10 times, that is, one decimal place	0~6 mm	0x0001	R/W
3	Peak power	2	Unsigned	1 to 3000W (do not actually exceed the laser power)	0x0002	R/W
4	Duty cycle	2	Unsigned	0 to 100 percent	0x0003	R/W
5	Pulse frequency 31 to 16 bits	2	Unsigned	0~100000 Hz	0x0004	R/W
6	Pulse frequency 15 to 0 bits	2	Unsigned		0x0005	R/W
7	Save the current handheld welding data	2	Unsigned	0 or 1	0x003F	W
Wire feeder-related parameters						
1	Automatic wire feeding speed	2	Unsigned	15~600 cm/min	0x0100	R/W
2	Manual wire feeding speed	2	Unsigned	15~600 cm/min	0x0101	R/W
3	Manually pull back the speed	2	Unsigned	15~600 cm/min	0x0102	R/W
4	Startup delay	2	Unsigned	0~2000 ms	0x0103	R/W
5	Retraction	2	Unsigned	0~100 mm	0x0104	R/W

	length					
6	Patch length	2	Unsigned	0~100 mm	0x0105	R/W
7	Wire filling delay	2	Unsigned	0~2000 ms	0x0106	R/W
8	Pulse period	2	Unsigned	100~1000 ms	0x0107	R/W
9	Smoothness	2	Unsigned	25 percent to 80 percent	0x0108	R/W
10	Average speed	2	Unsigned	15~150 cm/min	0x0109	R/W
11	Language selection	2	Unsigned	0～8	0x010A	R/W
12	Reserved	2	Unsigned		0x010B	R/W
13	Mode Settings	2	Unsigned	0: Continuous mode 1: Pulse mode	0x010C	R/W
14	Reserved	2	Unsigned		0x010D	R/W
15	Reserved	2	Unsigned		0x010E	R/W
16	Reserved	2	Unsigned		0x010F	R/W
17	Reserved	2	Unsigned		0x0110	R/W
18	Reserved	2	Unsigned		0x0111	R/W
19	Overcurrent alert	2	Unsigned	1: Alarm 2: Normal	0x0112	R
20	Hardware version	2	Unsigned		0x0113	R
21	Software version	2	Unsigned		0x0114	R
22	Operating status	2	Unsigned	0: Stop 1: Running	0x0115	R
23	Save the current data of the Wire feed	2	Unsigned	XX (any value)	0x013F	W

Note: R/W - indicates that the parameter can read and write; R - indicates that the parameter can only be read; W - indicates that the parameter can only be written

Starting from the V1.4 version of the protocol, it supports querying and setting relevant parameters of the Wire feed through handheld welding.

The functions of each register are defined as shown in Table 2.3:

Table 2.3 Function Definitions of Registers

NO.	Function name	Function Definitions
Handheld weld-related parameters		
1	Scan speed	Scan speed: 2 to 6000mm/s, scan width: 0 to 6mm. The scanning speed is limited by the scanning width, and the limiting relationship is: 10≤ scan speed /(scan width *2)≤1000. If the limit is exceeded, it will automatically change to the limit value. When the scan width is set to 0, it will not scan (i.e., point light source).
2	Scan width	
3	Peak power	The peak power must be less than or equal to the laser power on the parameter page.
4	Duty cycle	Duty cycle range: 0 to 100%.
5	Pulse frequency	Pulse frequency range: 5 to 100,000 Hz, recommended 5 to 5000Hz.
6	Save the current handheld welding data	This instruction will save all current process data for the handheld weld. Suggestion: Do not save unless necessary to avoid affecting the flash's lifespan.
Wire feeder-related parameters		
1	Automatic wire feeding speed	Indicates the speed of automatic wire feeding, with a range of: 15 to 600cm/min.
2	Manual wire feeding speed	Indicates the speed of manual wire feeding, with a range of: 15 to 600cm/min.
3	Manual retraction speed	Indicates the speed of manual retraction, with a range of: 15 to 600cm/min.
4	Start delay	It represents the delay time between the press of the wire feed switch and the start of wire feeding, ranging from 0 to 2000ms.
5	Retraction length	Indicates the length of the wire to be pulled back after the wire feed switch is released (to help break the wire), with a range of 0 to 100mm.
6	Fill wire length	It indicates the length of the refilled wire after the refilled wire, with a range of 0 to 100mm.
7	Wire supplement delay	It represents the waiting time between retraction and filament filling, ranging from 0 to 2000ms. (Used to enhance filament breakage effect)
8	Pulse period	Used in pulse control mode, it shows the size of individual fish scale patterns, with larger values indicating larger fish scale patterns. The range is: 100 to 1000ms.
9	Smoothness	For pulse control mode, it reflects the smoothness of the fish-scale pattern. The larger the value, the smoother the pattern, and the less obvious the fish-scale effect. The range is: 25% to 80%.
10	Average speed	Used in pulse control mode, it represents the average wire feeding speed in pulse mode, with a range of: 15 to 150cm/min.

11	Language selection	The register values range from 0 to 8, corresponding in sequence to the following 9 languages: Simplified Chinese, English, Traditional Chinese, Japanese, Korean, Russian, German, French, and Latin.
12	Reserved	
13	Mode Settings	Used to set the current wire feeding mode: 0: Continuous mode 1: Pulse mode
14 ~18	Reserved	
19	Overcurrent alert	After the wire feeder generates an overcurrent alarm, it will automatically stop feeding. After checking and confirming that there are no abnormalities, restart the wire feeder to resume normal operation.
20	Hardware version	The current hardware version number of the wire feeder in use.
21	Software version	The current version number of the wire feeder main control board software in use.
22	Running status	The running status of automatic wire feeding, stop or run.
23	Save the current wire feeder data	This instruction will save the current values of all parameters of the wire feeder. Suggestion: Do not save unless necessary to avoid affecting the flash's lifespan.

2.3 Data frame format

(1)Send, Receipt (successful response) basic data format

The basic data formats for sending and receiving are shown in Table 2.4:

Table 2.4 Basic Data Formats for Sending and Receiving Receipts

Slave address (1 Byte)	Function Code (1 Byte)	Data area (n Bytes			Error check code (Low byte)	Error check code (High byte)
0x09	0x03/0x06/0x10	0xXX	……	0xXX	0xXX	0xXX

Slave address: The default address for the handheld solder is 0x09. The address can be modified via the display screen as follows:

① Enter the authorization password entry page on the monitor page of the display screen;

② Enter ffffffdd001 to ffffffdd247 (the last three digits represent the slave address) to change the slave address to 1 to 247. As shown in Figure 2-1, to modify the hand-held solder address to 1:

Figure 2-1 Modify the handheld solder address to 1

Function code: See Table 2.5.

0x03 reads one or more registers, and the corresponding exception receipt function code is 0x83.

0x06 writes to a single register, corresponding to the exception receipt function code 0x86.

0x10 writes to multiple registers, corresponding to the exception receipt function code 0x90.

Data area: n bytes. For double-byte data, the higher byte comes first and the lower byte follows.

Error check code: The check method is CRC16(Modbus), with the high byte in front and the low byte behind. CRC calculation includes all data from the "slave address" up to the CRC check code. See Appendix A for the calculation method.

CRC check process: After the sender calculates the CRC of the data to be sent, the calculated CRC value is added to the sending frame. After receiving the data frame, the recipient recalculates the CRC value of the received data and compares it with the received CRC value. If the two CRC values are the same, it is considered normal; otherwise, it is abnormal.

Table 2.5 List of Function Codes

Function Codes (send)	Features	Receipt Function Code (Normal)	Receipt Function Code (Abnormal)
0x03	Read one or more registers	0x03	0x83
0x06	Write to a single register	0x06	0x86
0x10	Write multiple registers	0x10	0x90

(2) Error receipt Basic data format:

The basic data format of the error receipt is shown in Table 2.6:

Table 2.6 Basic Data Format for Error Receipts

Slave address (1 Byte)	Function Code (1 Byte)	Error code (1 Byte)	CRC16(Low byte	CRC16 (High byte)
0x09	0x83/0x86/0x90	0x01/0x02/0x03	0XX	0xXX

Abnormal function code corresponding receipt relationship:

0x03-0x83 When the function value is abnormal, the high position of the function code on the receipt is 1, that is: 0x83.

0x06-0x86 When the function value is abnormal, return receipt function code high position 1, that is: 0x86.

0x10-0x90 when the function value is abnormal, return receipt function code high position 1, that is: 0x90.

Error code:

0x01 - Illegal feature;

0x02 - Illegal register address;

0x03 - Illegal register value.

Note: The handheld solder does not respond when the slave address is incorrect, the CRC check is incorrect, or the function code is incorrect.

By default, CRC is in little-endian mode (with the low byte first and the high byte second). The order of the high and low byte bits of CRC can be modified through the display screen. The modification method is as follows:

① Enter the authorization password entry page on the monitor page of the display screen;

② Entering ffffffee111 changes the CRC to big-endian mode (high byte first, low byte last), and entering ffffffee222 changes the CRC to little-endian mode (low byte first, high byte last). Figure 2-2 shows setting CRC to big-endian mode:

Figure 2-2 Setting CRC to big-endian mode

2.4 Read register instruction - function code 0x03

2.4.1 Introduction to the structure of Function Code 0x03 instructions

(1) Function code 0x03 can read values from one or more registers simultaneously, and its instruction structure is shown in Table 2.7:

Table 2.7 Instruction Structure of Function Code 0x03

Read registers (host request)								
Data direction	Slave address (1 Byte)	Function code (1 Byte)	Data Area				Error check code	
			Register start address (2 Bytes)		Number of registers (2 Bytes)		CRC16 (2 Bytes)	
Master → Slave			Register start address (high byte)	Register start address (low byte)	Number of registers (high byte)	Number of registers (low bytes)	CRC16 (low byte)	CRC16 (high byte)
	0x09	0x03	0xXX	0xXX	N		0xXX	0xXX

Note: When reading multiple registers, it means reading N consecutive registers starting from the register address, where consecutive means the register addresses are consecutive.

(2) Function code 0x03 responded successfully. The instruction structure is shown in Table 2.8:

Read the register (slave response)										
Data direction	Slave address (1 Byte)	Function code (1 Byte)	Data Area						Error check code	
			Return Byte count (1 Byte)	Data1 (2 Bytes)		… …	DataN (2 Bytes)		CRC16 (2 Bytes)	
Slave → Master (Successful response)				Data1(high byte)	Data1 (low byte)	… …	Data N (high byte)	Data N (low byte)	CRC16 (low byte)	CRC16 (high byte)
	0x09	0x03	2*N	0xXX	0xXX		0xXX	0xXX	0xXX	0xXX

Table 2.8 Function Code 0x03 successfully responds to instruction structure

Note:

① "Return byte count" indicates the total number of bytes from "Data1" to "DataN".

② Data1 is the value of the register corresponding to the starting address, Data2 is the value of the register corresponding to the starting address plus 1, and so on.

(3) The structure of the error response instruction is shown in Table 2.6.

2.4.2 Example of Function Code 0x03 instructions

(1) Examples of instructions for reading "Peak power" and "Duty cycle" are shown in Table 2.9:

Table 2.9 Reading Examples of "Peak Power" and "Duty Cycle"

Data Direction	Instruction Content
Master → Slave	09 03 00 02 00 02 64 83
Slave → Master	09 03 04 00 64 00 32 B3 F9

Examples are shown in Table 2.10:

Table 2.10 Read the examples of "Peak Power" and "Duty Cycle"

Data Direction	Slave address (1 Byte)	Function code (1 Byte)	Register address (2 Bytes)	Number of registers (2 Bytes)	CRC16 (2 Bytes)	
Master → Slave	0x09	0x03	0x0002	0x0002	0x6483	
Data Direction	Slave address (1 Byte)	Function code (1 Byte)	Return Byte count (1 Byte)	Data1 (2 Bytes)	Data2 (2 Bytes)	CRC16 (2 Bytes)
Slave → Host	0x09	0x03	0x04	0x0064	0x0032	0xB3F9

Shown in Table 2.10:

Master → Slave: Read register function code 0x03, "Peak power" address 0x0002, "duty cycle" address 0x0003. Their addresses are consecutive, so read the values of the two registers starting with address 0x0002 directly to read the values of "peak power" and "duty cycle".

Slave → Master: Read the registers successfully and the function code remains unchanged at 0x03; Return the values of two registers, return byte count =2*2=4; Data1 is 0x0064, corresponding to the decimal form of 100, indicating the current peak power is 100W; Data2 is 0x0032, corresponding to 50 in decimal form, indicating a current duty cycle of 50%.

(2) Error response example:

As shown in Table 2.11, when the master sends a read to a non-existent register address 0x0010 to the slave, the slave replies with error function code 0x83 and error code 0x02 (indicating an illegal register address).

Table 2.11 Examples of responses to read register errors

Data direction	Instruction Content
Master → Slave	09 03 00 10 00 01 84 87
Slave → Master	09 83 02 41 33

2.5 Write register instructions - Function code 0x06 and function code 0x10

2.5.1 Introduction to the structure of Function Code 0x06 instructions

(1) Function code 0x06 can be written to a single register, and its instruction structure is shown in Table 2.12:

Table 2.12 Function Code 0x06 Instruction Structure

Write to a single register (host request)								
Data direction	Slave address (1 Byte)	Function code (1 Byte	Data Area				Error check code	
			Register address (2 Bytes)		Register values (2 Bytes)		CRC16 (2 Bytes)	
Master → Slave			Register address (high byte)	Register address (low byte)	Register value (high byte)	Register value (low byte)	CRC16 (low byte)	CRC16 (high byte)
	0x09	0x06	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX

Data can be written to a single register address via function code 0x06, as shown in Table 2.12.

(2) Function code 0x06 responds successfully. The instruction structure is shown in Table 2.13:

Table 2.13 Function Code 0x06 Successful Response Instruction Structure

Write to a single register (slave response)								
Data direction	Slave address (1 Byte)	Function code (1 Byte)	Data Area				Error check code	
			Register address (2 Bytes)		Register values (2 Bytes)		CRC16 (2 Bytes)	
Slave → Master (Successful response)			Register address (high byte)	Register address (low byte)	Register value (high byte)	Register value (low byte)	CRC16 (low byte)	CRC16 (high byte)
	0x09	0x06	0xXX	0xXX	N		0xXX	0xXX

(3) The structure of the error response instruction is shown in Table 2.6.

2.5.2 Function Code 0x06 instruction example

An example of writing "peak power" is shown in Table 2.14:

Table 2.14 Examples of Writing "Peak Power"

Data Direction	Instruction Content
Master → Slave	09 06 00 02 00 64 28 A9
Slave → master	09 06 00 02 00 64 28 A9

Examples are shown in Table 2.15:

Table 2.15 Examples of writing "Peak Power"

Data Direction	**Slave address (1 Byte)**	**Function code (1 Byte)**	**Register address (2 Bytes)**	**DATA1 (2 Bytes)**	**CRC16 (2 Bytes)**
Master → Slave	0x09	0x06	0x0002	0x0064	0x28A9
Data direction	Slave address (1 Byte)	Function code (1 Byte)	Register address (2 Bytes)	Number of registers (2 Bytes)	CRC16 (2 Bytes)
Slave → Host	0x09	0x06	0x0002	0x0064	0x28A9

As shown in Table 2.15, the "Peak Power" register address is 0x0002. In the example, writing the value 0x0064 (corresponding to 100 in decimal) to address 0x0002 indicates setting the "Peak Power" to 100W.

2.5.3 Description of Function Code 0x10 Instruction Structure

(1) Function code 0x10 can be written to multiple registers simultaneously, and its instruction structure is shown in Table 2.16:

Table 2.16 Function Code 0x10 Instruction Structure

As shown in Table 2.16, function code 0x10 can write data to N consecutive registers starting with the "register start address" simultaneously. "Data byte count" represents the total number of bytes from "Data1"to “ to"Data N".

Read registers (host request)													
Data direction	Slave address (1 Byte)	Function code (1 Byte)	Data Area									Error check code	
Master → Slave			Register start address (2 Bytes)		Number of registers (2 Bytes)		Number of data bytes	Data1		. . .	Data N	CRC16 (high byte)	
			Register start address (high byte)	Register start address (low byte)	Number of registers (high byte)	Number of registers (low bytes)		Data1 (high byte)	Data1 (low byte)	. . .	Data N(high byte)	Data N(low byte)	CRC16 (Low Byte	CRC16 (high byte)
	0x09	0x10	0xXX	0xXX	N		2*N	0xXX	0xXX		0xXX	0xXX	0xXX	0xXX

(2) Function code 0x10 responds successfully, instruction structure shown in Table 2.17:

Table 2.17 Function Code 0x10 Successful Response Instruction Structure

Write to the register (slave response)								
Data direction	Slave address (1 Byte)	Function code (1 Byte)	Data Area				Error check code	
Slave → Master (Successful response)			Register start address (2 Bytes)		Number of registers (2 Bytes)		CRC16 (2 Bytes)	
			Register start address (high byte)	Register start address (low byte)	Number of registers (high byte)	Number of registers (low bytes)	CRC16 (low byte)	CRC16 (high byte)
	0x09	0x10	0xXX	0xXX	N		0xXX	0xXX

(3) The structure of the error response instruction is shown in Table 2.6.

2.5.4 Function Code 0x10 instruction example

Examples of writing "pulse frequency" are shown in Table 2.18:

Table 2.18 Examples of setting the "pulse frequency" to 2000Hz

Data direction	Instruction Content
Master→Slave	09 10 00 04 00 02 04 00 00 07 D0 DB 90
Slave→Master	09 10 00 04 00 02 01 41

Examples are shown in Table 2.19:

Table 2.19 Examples of Writing "Pulse Frequency"

Data direction	**Slave address (1 Byte)**	**Function code (1 Byte)**	**Register address (2 Bytes)**	**Number of registers (2 Bytes)**	**Byte count (1 byte)**	**DATA1 (2 Bytes)**	**DATA2 (2 Bytes)**	**CRC16(2 Bytes)**
Master → Slave	0x09	0x10	0x0004	0x0002	0x04	0x0000	0x07D0	0xDB90
"Data direction"	Slave address (1 Byte)	Function code (1 Byte)	Register address (2 Bytes)	Number of registers (2 Bytes)	CRC16 (2 Bytes)			
Slave → Host	0x09	0x10	0x0004	0x0002	0x0141			

As shown in Table 2.19, the address of the "Pulse frequency" register is 0x0004 (high 16 bits), 0x0005 (low 16 bits), the number of registers is 2, and the number of bytes =2 registers *2 bytes =4 bytes. The hexadecimal number for 2000 is 0x000007D0.

2.6 Instructions for Use

(1) **Instruction interval**: The baud rate defaults to 19200 BPS, and the minimum time interval for instruction transmission is 20ms.

(2) **Parameters take effect**: After the process parameters are updated via Modbus protocol, the screen interface will update immediately. If the equipment is in the welding operation state, the parameters will take effect immediately.

(3) **Parameter saving**: All data are treated as temporary parameters. They will not be saved in case of power failure. When high-frequency parameters change, it is not recommended to save them in real time, as this may affect the efficiency of data transmission and the lifespan of the flash. If you want to save, send the save command, and after saving, the process interface data on the screen will be updated.

2.7 Examples of Common Commands

(1) An example of setting the "scan width" command separately is shown in Table 2.20:

Table 2.20 Scan Width Setting Instructions Example

Set scan width	
Set values (mm)	Instructions
1	09 06 00 01 00 0a 59 45
2	09 06 00 01 00 14 D9 4D
3	09 06 00 01 00 1e 59 4A
4	09 06 00 01 00 28 D9 5C
5	09 06 00 01 00 32 58 97

(2) An example of setting the "peak power" command separately is shown in Table 2.21:

Table 2.21 Examples of Peak Power Setting Instructions

Set the scan width	
Set value (W)	Instruction
100	09 06 00 02 00 64 28 A9
200	09 06 00 02 00 c8 28 D4
300	09 06 00 02 01 2c 29 0F
400	09 06 00 02 01 90 28 BE
500	09 06 00 02 01 f4 29 55
600	09 06 00 02 02 58 29 D8

(3) example of the "duty cycle" instruction set separately is shown in Table 2.22:

Table 2.22 Examples of Duty Cycle Setting Instructions

Set the duty cycle	
Set the value (%)	Instructions
50	09 06 00 03 00 32 F9 57
100	09 06 00 03 00 64 79 69

(4) An example of setting the "pulse frequency" command separately is shown in Table 2.23:

Table 2.23 Examples of Pulse Frequency Setting Instructions

Set the duty cycle	
Set the value (Hz)	Instructions
1000	09 10 00 04 00 02 04 00 00 03 e8 D8 82
2000	09 10 00 04 00 02 04 00 00 07 d0 db 90

(5)An example of setting both "scan width" and "Scan speed" commands is shown in Table 2.24:

Table 2.24 Examples of Commands for Setting Both Scan Width and Scan Speed simultaneously

Set the scan width and scan speed		
Scan width setting value (mm)	Scan speed setting value (mm/S)	Instructions
1	100	09 10 00 00 00 02 04 03 e8 00 0a d9 b8
2	200	09 10 00 00 00 02 04 07 d0 00 14 d9 4d
3	300	09 10 00 00 00 02 04 0b b8 00 1e db c6

Appendix: CRC Algorithm

(1)Table lookup method:

```
uint16_t modbus_CRC16_Table (uint8_t *nData, uint16_t wLength)
{
    static const uint16_t wCRCTable[] = {
    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 };

    uint8_t nTemp;
    uint16_t wCRCWord = 0xFFFF;
    uint16_t wCRCResult = 0;
```

```
    while (wLength--)
    {
        nTemp = *nData++ ^ wCRCWord;
        wCRCWord >>= 8;
        wCRCWord ^= wCRCTable[nTemp];
    }
    // Swap the high and low bytes of the CRC
    wCRCResult = wCRCWord>>8;
    wCRCResult = wCRCResult | (wCRCWord << 8);
    return wCRCResult;
}
```

(2)Direct calculation

```
uint16_t modbus_CRC16_Calculate(uint8_t *ndata, uint16_t wLength)
{
    uint16_t wCRCWord = 0xFFFF;
    uint16_t polynomial = 0xA001;
    uint16_t wCRCResult = 0;
    uint8_t i,j;

    for(i = 0;i < wLength;i++)
    {
        wCRCWord ^= ndata[i];
       for(j = 0;j < 8;j++)
       {
            if((wCRCWord & 0x0001) ! = 0)
            {
                wCRCWord >>= 1;
                wCRCWord ^= polynomial;
            }
            else
            {
                wCRCWord >>= 1;
            }
       }
    }
    // Swap the high and low bytes of CRC
    wCRCResult = wCRCWord>>8;
    wCRCResult = wCRCResult | (wCRCWord << 8);
    return wCRCResult;
}
```