无锡超强伟业科技有限公司

手持焊通信协议

V1.0

修改记录:

版本号	主要变更内容	拟制/修改日期	备注
V1.0	首次发布，基于手持焊接协议 v1.4 修改	2024/01/18	

目录

1. 硬件接口定义

我司手持焊产品对外通信接口为 RS485 接口，采用 DB9 母头插座。该 DB9 插座接口定义见表 1.1。

表 1.1 DB9 接口定义（手持焊端）

引脚序号	信号	功能说明
1	RS485_B	RS485 信号 B
2	RS232_TXD	内部使用，建议悬空
3	RS232_RXD	内部使用，建议悬空
4	NC	不使用
5	GND	信号地
6	RS485_A	RS485 信号 A
7	NC	不使用
8	NC	不使用
9	+5V	不使用

2. 通信格式

本协议兼容 Modbus RTU 规范。

2.1 通信模块基本参数

通信模块基本参数见表 2.1：

表 2.1 通信模块基本参数

编码	8 位二进制
数据位	8 位
奇偶校验位	无
停止位	1 位
波特率	19200 bit/s

2.2 寄存器地址定义

寄存器地址定义见表 2.2：

表 2.2 手持焊寄存器地址定义表

NO.	功能名称	数据长度(Bytes)	数据类型	数据范围	寄存器地址	R/W 属性
手持清洗相关参数						
1	扫描频率	2	无符号	30~100 Hz	0x0000	R/W
2	扫描宽度	2	无符号 扩大 10 倍传输，即 1 位小数	0~150/0~225/0-300mm,根据屏幕系统“设置”页不同的聚焦镜选择不同	0x0001	R/W
3	峰值功率	2	无符号	1~3000W（实际不能超过激光器功率)	0x0002	R/W
4	占空比	2	无符号	0~100 %	0x0003	R/W
5	脉冲频率 31~16 位	2	无符号	5~100000 Hz	0x0004	R/W
6	脉冲频率 15~0 位	2	无符号		0x0005	R/W
7	端点优化	2	有符号	-30-30	0x0006	R/W
8	工艺切换	2	无符号	0-2(工艺 0~工艺 2)	0x0030	W
9	保存当前手持焊数据	2	无符号	0 或 1	0x003F	W

注：R/W—表示该参数能读能写；R—表示该参数只能读；W—表示该参数只能写

数据更新后，屏幕首页数据会更新，但是工艺页面不会刷新。特别要注意的是端点优化，在首页没有显示，工艺页也不会刷新，但是实际会生效。

各寄存器功能定义如表 2.3 所示：

表 2.3 各寄存器功能定义

NO.	功能名称	功能定义
手持焊相关参数		
1	扫描频率	30Hz~100Hz,宽度采用满范围时，推荐频率采用不大于 70Hz。
2	扫描宽度	0~150/0~225/0-300mm,根据屏幕系统“设置”页不同的聚焦镜选择不同
3	峰值功率	峰值功率需小于等于参数页激光器功率。
4	占空比	占空比范围：0～100%。

5	脉冲频率	脉冲频率范围：5～100000Hz，建议 5～5000Hz。
6	端点优化	-30~30，消除扫描轨迹两端火花
8	保存当前手持焊数据	该指令将保存手持焊当前所有工艺数据。建议：非必要不保存，以免影响 flash 寿命。

2.3 数据帧格式

（1）发送、回执（成功响应）基本数据格式

发送与回执基本数据格式见表 2.4：

表 2.4 发送与回执基本数据格式

从机地址（1 Byte）	功能码（1 Byte）	数据区（n Bytes）			错误校验码(低字节)	错误校验码(高字节)
0x09	0x03/0x06/0x10	0xXX	……	0xXX	0xXX	0xXX

从机地址：手持焊默认地址为 0x09，可通过显示屏修改地址，修改方法如下：

① 进入显示屏监测页的授权密码输入页面；

② 输入 fffffdd001～fffffdd247（后 3 位表示从机地址），可将从机地址修改为 1～247。如图 2-1 所示为将手持焊地址修改为 1：

图 2-1 将手持焊地址修改为 1

功能码：见表 2.5。

0x03 读 1 个或多个寄存器，对应的异常回执功能码为 0x83；

0x06 写单个寄存器，对应的异常回执功能码为 0x86。

0x10 写多个寄存器，对应的异常回执功能码为 0x90。

数据区：n 个字节，双字节数据时，高字节在前，低字节在后。

错误校验码：校验方式采用 CRC16(Modbus)，高字节在前，低字节在后。CRC 计算包含从“从机地址”开始，直到 CRC 校验码前的所有数据，计算方法参见附录 A。

CRC 校验过程：发送方计算待发送数据的 CRC 后，将计算出的 CRC 值加入发送帧。接收方接收到数据帧后，重新计算接收到的数据的 CRC 值，并与接收到的 CRC 值

比较，如果两个 CRC 值相同，认为正常，否则异常。

表 2.5 功能码列表

功能码(发送)	功能	回执功能码(正常)	回执功能码(异常)
0x03	读 1 个或多个寄存器	0x03	0x83
0x06	写单个寄存器	0x06	0x86
0x10	写多个寄存器	0x10	0x90

（2）错误回执基本数据格式：

错误回执基本数据格式见表 2.6：

表 2.6 错误回执基本数据格式

从机地址（1 Byte）	功能码（1 Byte）	错误代码（1 Byte）	CRC16(低字节)	CRC16(高字节)
0x09	0x83/0x86/0x90	0x01/0x02/0x03	0XX	0xXX

异常功能码对应回执关系：

0x03—0x83 当功能值异常时，回执功能码高位置 1，即：0x83。

0x06—0x86 当功能值异常时，回执功能码高位置 1，即：0x86。

0x10—0x90 当功能值异常时，回执功能码高位置 1，即：0x90。

错误代码：

0x01—非法功能；

0x02—非法寄存器地址；

0x03—非法寄存器值。

注：从机地址错误、CRC 校验错误、功能码错误时，手持焊不响应。

CRC 默认为小端模式（低字节在前，高字节在后），可通过显示屏修改 CRC 高低字节位顺序，修改方法如下：

① 进入显示屏监测页的授权密码输入页面；

② 输入 fffffee111 可将 CRC 改为大端模式(高字节在前,低字节在后),输入 fffffee222 可将 CRC 改为小端模式(低字节在前,高字节在后)。图 2-2 所示为将 CRC 设为大端模式：

图 2-2 将 CRC 设为大端模式

2.4 读寄存器指令—功能码 0x03

2.4.1 功能码 0x03 指令结构介绍

（1）功能码 0x03 可同时读取 1 个或多个寄存器的值，其指令结构见表 2.7：

表 2.7 功能码 0x03 指令结构

读寄存器 (主机请求)								
数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	数据区				错误校验码	
主机→从机			寄存器起始地址 (2 Bytes)		寄存器数量 (2 Bytes)		CRC16 (2 Bytes)	
			寄存器起始地址 (高字节)	寄存器起始地址 (低字节)	寄存器数量 (高字节)	寄存器数量 (低字节)	CRC16 (低字节)	CRC16 (高字节)
	0x09	0x03	0xXX	0xXX	N		0xXX	0xXX

注：当读取多个寄存器时，它的含义是读取以寄存器起始地址开始的连续 N 个寄存器，这里的连续是指寄存器地址连续。

（2）功能码 0x03 成功响应，指令结构见表 2.8：

表 2.8 功能码 0x03 成功响应指令结构

读寄存器 (从机应答)										
数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	数据区						错误校验码	
从机→主机 (成功响应)			返回字节数 (1 Byte)	Data1 (2 Bytes)		……	DataN (2 Bytes)		CRC16 (2 Bytes)	
				Data1 (高字节)	Data1 (低字节)	……	DataN (高字节)	DataN (低字节)	CRC16 (低字节)	CRC16 (高字节)
	0x09	0x03	2*N	0xXX	0xXX		0xXX	0xXX	0xXX	0xXX

注：

① “返回字节数”表示“Data1”～“DataN”总的字节数量。

② Data1 为起始地址对应的寄存器的值，Data2 为起始地址+1 对应的寄存器的值，如此依次递增。

（3）错误响应指令结构见表 2.6。

2.4.2 功能码 0x03 指令示例

（1）读“峰值功率”和“占空比”指令示例见表 2.9：

表 2.9 读“峰值功率”和“占空比”示例

数据方向	指令内容
主机→从机	09 03 00 02 00 02 64 83
从机→主机	09 03 04 00 64 00 32 B3 F9

示例说明见表 2.10：

表 2.10 读“峰值功率”和“占空比”示例说明

数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	寄存器地址 (2 Bytes)	寄存器数量 (2 Bytes)	CRC16 (2 Bytes)	
主机→从机	0x09	0x03	0x0002	0x0002	0x6483	
数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	返回字节数 (1 Byte)	Data1 (2 Bytes)	Data2 (2 Bytes)	CRC16 (2 Bytes)
从机→主机	0x09	0x03	0x04	0x0064	0x0032	0xB3F9

如表 2.10 所示：

主机→从机：读寄存器功能码为 0x03，“峰值功率”地址为 0x0002，“占空比”地址为 0x0003，它们地址连续，因此直接读取以地址 0x0002 为起始地址的 2 个寄存器的值，即可读取到“峰值功率”和“占空比”的值。

从机→主机：读寄存器成功响应，功能码不变，为 0x03；返回 2 个寄存器的值，返回字节数=2*2=4；Data1 为 0x0064，对应的 10 进制形式为 100，表示当前峰值功率为 100W；Data2 为 0x0032，对应的 10 进制形式为 50，表示当前占空比为 50%。

（2）错误响应示例：

如表 2.11 所示，当主机向从机发送读一个不存在的寄存器地址 0x0010 时，从机回复错误功能码 0x83，错误代码 0x02（表示非法寄存器地址）。

表 2.11 读寄存器错误响应示例

数据方向	指令内容
主机→从机	09 03 00 10 00 01 84 87
从机→主机	09 83 02 41 33

2.5 写寄存器指令—功能码 0x06 和功能码 0x10

2.5.1 功能码 0x06 指令结构介绍

（1）功能码 0x06 可写单个寄存器，其指令结构见表 2.12：

表 2.12 功能码 0x06 指令结构

写单个寄存器（主机请求）								
数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	数据区				错误校验码	
主机→从机			寄存器地址 (2 Bytes)		寄存器值 (2 Bytes)		CRC16 (2 Bytes)	
			寄存器地址 (高字节)	寄存器地址 (低字节)	寄存器值 (高字节)	寄存器值 (低字节)	CRC16 (低字节)	CRC16 (高字节)
	0x09	0x06	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX

如表 2.12 所示，通过功能码 0x06 可向单个寄存器地址中写入数据。

（2）功能码 0x06 成功响应，指令结构见表 2.13：

表 2.13 功能码 0x06 成功响应指令结构

写单个寄存器（从机应答）								
数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	数据区				错误校验码	
从机→主机 (成功响应)			寄存器地址 (2 Bytes)		寄存器值 (2 Bytes)		CRC16 (2 Bytes)	
			寄存器地址 (高字节)	寄存器地址 (低字节)	寄存器值 (高字节)	寄存器值 (低字节)	CRC16 (低字节)	CRC16 （高字节）
	0x0A	0x06	0xXX	0xXX	N		0xXX	0xXX

（3）错误响应指令结构见表 2.6。

2.5.2 功能码 0x06 指令示例

写“峰值功率”示例见表 2.14：

表 2.14 写“峰值功率”示例

数据方向	指令内容
主机→从机	09 06 00 02 00 64 28 A9
从机→主机	09 06 00 02 00 64 28 A9

示例说明见表 2.15：

表 2.15 写“峰值功率”示例说明

数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	寄存器地址 (2 Bytes)	DATA1 (2 Bytes)	CRC16 (2 Bytes)
主机→从机	0x09	0x06	0x0002	0x0064	0x28A9
数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	寄存器地址 (2 Bytes)	寄存器数量 (2 Bytes)	CRC16 (2 Bytes)
从机→主机	0x09	0x06	0x0002	0x0064	0x28A9

如表 2.15 所示，“峰值功率”寄存器地址为 0x0002，示例中向地址 0x0002 中写入数值 0x0064（对应的 10 进制数为 100），表示将“峰值功率”设置为 100W。

2.5.3 功能码 0x10 指令结构介绍

写寄存器（主机请求）														
数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	数据区											错误校验码
			寄存器起始地址		寄存器数量		数据字节数 (1 Byte)	Data1		.....	DataN		CRC16	
主机→从机			寄存器起始地址 (高字节)	寄存器起始地址 (低字节)	寄存器数量 (高字节)	寄存器数量 (低字节)		Data1 (高字节)	Data1 (低字节)	.....	DataN (高字节)	DataN (低字节)	CRC16 (低字节)	CRC16 (高字节)
	0x09	0x10	0xXX	0xXX	N		2*N	0xXX	0xXX		0xXX	0xXX	0xXX	0xXX

（1）功能码 0x10 可同时写多个寄存器，其指令结构见表 2.16：

表 2.16 功能码 0x10 指令结构

如表 2.16 所示，功能码 0x10 可同时向以“寄存器起始地址”开始的连续 N 个寄存器写入数据。“数据字节数”表示“Data1”～“DataN”总的字节数量。

（2）功能码 0x10 成功响应，指令结构见表 2.17：

表 2.17 功能码 0x10 成功响应指令结构

写寄存器 (从机应答)								
数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	数据区				错误校验码	
			寄存器起始地址 (2 Bytes)		寄存器数量 (2 Bytes)		CRC16 (2 Bytes)	
从机→主机 (成功响应)			寄存器起始地址 (高字节)	寄存器起始地址 (低字节)	寄存器数量 (高字节)	寄存器数量 (低字节)	CRC16 (低字节)	CRC16 （高字节）
	0x09	0x10	0xXX	0xXX	N		0xXX	0xXX

（3）错误响应指令结构见表 2.6。

2.5.4 功能码 0x10 指令示例

写“脉冲频率”示例见表 2.18：

表 2.18 将“脉冲频率”设置为 2000Hz 示例

数据方向	指令内容
主机→从机	09 10 00 04 00 02 04 00 00 07 D0 DB 90
从机→主机	09 10 00 04 00 02 01 41

示例说明见表 2.19：

表 2.19 写“脉冲频率”示例说明

数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	寄存器地址 (2 Bytes)	寄存器数量 (2 Bytes)	字节数量 (1 Bytes)	DATA1 (2 Bytes)	DATA2 (2 Bytes)	CRC16 (2 Bytes)
主机→从机	0x09	0x10	0x0004	0x0002	0x04	0x0000	0x07D0	0xDB90

数据方向	从机地址 (1 Byte)	功能码 (1 Byte)	寄存器地址 (2 Bytes)	寄存器数量 (2 Bytes)	CRC16 (2 Bytes)
从机→主机	0x09	0x10	0x0004	0x0002	0x0141

如表 2.19 所示，“脉冲频率”寄存器地址为 0x0004（高 16 位）、0x0005（低 16 位），寄存器数量为 2 个，字节数量=2 个寄存器*2 个字节=4 字节。2000 对应的 16 进制数为 0x000007D0。

2.6 使用说明

（1）**指令间隔**：波特率默认为 19200bps，指令发送最小时间间隔为 20ms。

（2）**参数生效**：通过 Modbus 协议更新完工艺参数后，屏幕界面会立即更新，若设备正处在焊接工作状态，参数会立即生效。

（3）**参数保存**：数据都是作为临时参数，掉电不保存，高频参数变化时，不推荐进行参数实时保存，容易影响数据发送效率和 flash 寿命。若要保存，发送保存指令，保存后，屏幕工艺界面数据会更新。

2.7 常用指令示例

（1）单独设置“扫描宽度”指令示例如表 2.20 所示：

表 2.20 扫描宽度设置指令示例

设置扫描宽度	
设置值（mm）	指令
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）单独设置“峰值功率”指令示例如表 2.21 所示：

表 2.21 峰值功率设置指令示例

设置扫描宽度	
设置值（W）	指令
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）单独设置“占空比”指令示例如表 2.22 所示：

表 2.22 占空比设置指令示例

设置占空比	
设置值（%）	指令
50	09 06 00 03 00 32 F9 57
100	09 06 00 03 00 64 79 69

（4）单独设置“脉冲频率”指令示例如表 2.23 所示：

表 2.23 脉冲频率设置指令示例

设置占空比	
设置值（Hz）	指令
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）同时写多个举例

同时写多个						
频率	宽度	功率	占空比	Pwm 频率高字节	Pwm 频率低字节	端点优化
50	100.0	500	100	0	200	20
发送：09 10 00 00 00 07 0E 00 32 03 E8 01 F4 00 64 00 00 07 D0 00 14 F1 77						
70	300.0	500	100	0	200	20
发送：09 10 00 00 00 07 0E 00 46 0B B8 01 F4 00 64 00 00 07 D0 00 14 A0 1D						
50	200.0	1500	100	0	2000	15
发送：09 10 00 00 00 07 0E 00 32 07 D0 05 DC 00 64 00 00 07 D0 00 0F 2A 07						

附录：CRC 算法

（1）查表法：

```
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];
    }
    //将 CRC 高低字节交换
    wCRCResult = wCRCWord>>8;
    wCRCResult = wCRCResult | (wCRCWord << 8);
    return wCRCResult;
}
```

（2）直接计算：

```
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;
            }
        }
    }
    //将 CRC 高低字节交换
    wCRCResult = wCRCWord>>8;
    wCRCResult = wCRCResult | (wCRCWord << 8);
    return wCRCResult;
}
```