Modbus RTU 协议手册

V1.2

目录

前言

Modbus 是一种串行通信协议，是 Modicon 于 1979 年，为使用可编程逻辑控制器（PLC）而发表的。Modbus 是工业领域通信协议的业界标准，并且现在是工业电子设备之间相当常用的连接方式。Modbus 比其他通信协议使用的更广泛的主要原因有：

（1）公开发表并且无版税要求

（2）相对容易的工业网络部署

（3）对供应商来说，修改移动原生的位元或字节限制较少

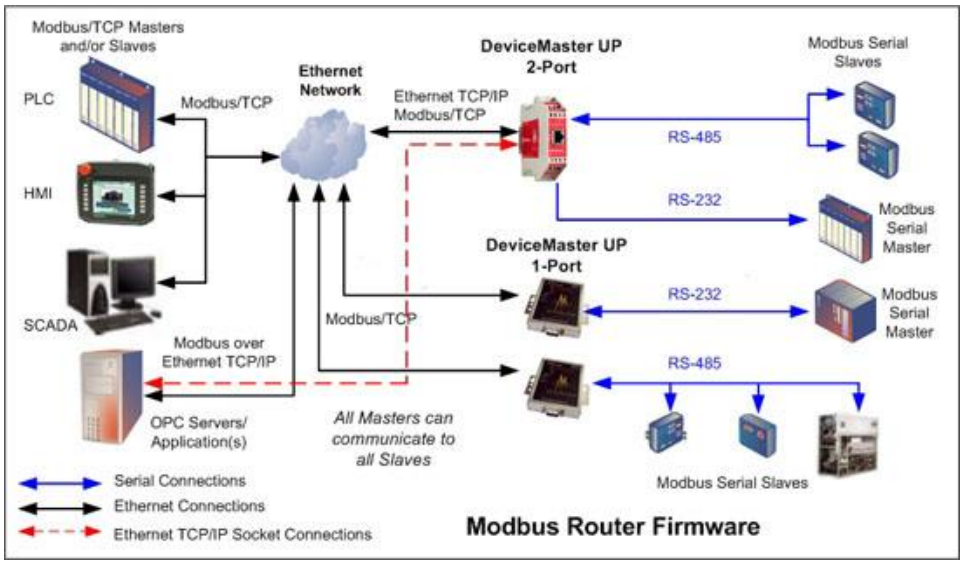

图 1 Modbus 网络示意图

注：带前缀 0x 或后缀 H 的数据为十六进制。

1 Modbus RTU 功能码简述

Modbus RTU 常用功能码如表 1.1 所示。

表 1.1 Modbus RTU 常用功能码

MODBUS 常用功能码				
功能码	**功能**	**寄存器 PLC 地址**	**位操作/字操作**	**操作数量**
01H	读线圈状态	00001-09999	位操作(1 字节)	单个或多个
02H	读离散输入状态	10001-19999	位操作(1 字节)	单个或多个
03H	读保持寄存器	40001-49999	字操作(2 字节)	单个或多个
04H	读输入寄存器	30001-39999	字操作(2 字节)	单个或多个
05H	写单个线圈	00001-09999	位操作(1 字节)	单个
06H	写单个保持寄存器	40001-49999	字操作(2 字节)	单个
0FH	写多个线圈	00001-09999	位操作(1 字节)	多个
10H	写多个保持寄存器	40001-49999	字操作(2 字节)	多个

1.1 功能码说明

功能码可以分为位操作和字操作两类。位操作的最小单位为 BIT，字操作的最小单位为两个字节。

（1）位操作指令

读线圈状态 01H，读（离散）输入状态 02H，写单个线圈 05H 和写多个线圈 0FH。

（2）字操作指令

读保持寄存器 03H，写单个寄存器 06H，写多个保持寄存器 10H。

1.2 寄存器地址分配

表 1.2 Modbus RTU 寄存器地址分配

MODBUS 寄存器地址分配				
寄存器 PLC 地址	**寄存器协议地址**	**功能码**	**位操作/字操作**	**操作数量**
00001-09999	0000H-FFFFH	01H、05H、0FH	线圈状态	可读可写
10001-19999	0000H-FFFFH	02H	离散输入状态	可读
30001-39999	0000H-FFFFH	04H	输入寄存器	可读
40001-49999	0000H-FFFFH	03H、06H、10H	保持寄存器	可读可写

1.3 寄存器种类说明

表 1.3 Modbus RTU 寄存器种类说明

MODBUS 寄存器种类说明			
寄存器种类	**说明**	**PLC 类比**	**举例说明**
线圈状态	输出端口。可设定端口的输出状态，也可以读取该位的输出状态	DO 数字量输出	继电器输出，MOSFET（晶体管）输出等
离散输入状态	输入端口。通过外部设定改变输入状态，可读但不可写	DI 数字量输入	按钮开关，光电开关等
保持寄存器	输出参数或保持参数。控制器运行时被设定的某些参数。可读可写	AO 模拟量输出	模拟量输出设定值，PID 运行参数，变量阀输出大小，传感器报警上限下限
输入寄存器	输入参数。控制器运行时从外部设备获得的参数。可读但不可写	AI 模拟量输入	模拟量输入

1.4 PLC 地址和协议地址区别

PLC 地址可以理解为协议地址的变种，在触摸屏和 PLC 编程中应用较为广泛。

1.4.1 寄存器 PLC 地址

寄存器 PLC 地址指存放于控制器中的地址，这些控制器可以是 PLC，也可以是触摸屏，或是文本显示器。PLC 地址一般采用 10 进制描述，共有 5 位，其中第一位代码寄存器类型。第一位数字和寄存器类型的对应关系如表 1 所示。PLC 地址例如 40001、30002 等。

1.4.2 寄存器协议地址

寄存器协议地址指指通信时使用的寄存器地址，例如 PLC 地址 40001 对应寻址地址 0000H，40002 对应寻址地址 0001H，寄存器寻址地址一般使用 16 进制描述。再如，PLC 寄存器地址 40003 对应协议地址 0002，PLC 寄存器地址 30003 对应协议地址 0002，虽然两个 PLC 寄存器寄存器通信时使用相同的地址，但是需要使用不同的命令访问，所以访问时不存在冲突。

2 Modbus RTU 指令说明

2.1 读线圈寄存器 01H

（1）描述

读线圈寄存器当前状态。

（2）查询：

例如从机地址为 01H，线圈寄存器的起始地址为 0013H，结束地址为 0037H。该次查询总共访问 37 个线圈寄存器。

表 2.1.1 读线圈寄存器—查询

读线圈寄存器 01H-主机发送		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	01
3	寄存器起始地址高字节	00
4	寄存器起始地址低字节	13
5	寄存器数量高字节	00
6	寄存器数量低字节	25
7	CRC 校验低字节	0C
8	CRC 校验高字节	14

（3）响应

响应负载中的各线圈状态与数据内容每位相对应。1 代表 ON，0 代表 OFF。若返回的线圈数不为 8 的倍数，则在最后数据字节未尾使用 0 代替。

表 2.1.2 读线圈寄存器—响应

读线圈寄存器 01H-模块返回		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	01

3	返回字节数	05
4	数据 1(线圈 0013H-线圈 001AH)	CD
5	数据 2(线圈 001BH-线圈 0022H)	6B
6	数据 3(线圈 0023H-线圈 002AH)	B2
7	数据 4(线圈 0032H-线圈 002BH)	0E
8	数据 5(线圈 0037H-线圈 0033H)	1B
9	CRC 校验低字节	44
10	CRC 校验高字节	EA

线圈 0013H 到线圈 001AH 的状态为 CDH，二进制值为 11001101，该字节的最高位为线圈 001AH，最低位为线圈 0013H。线圈 001AH 到线圈 0013H 的状态分别为 ON-ON-OFF-OFF-ON-ON-OFF-ON。

表 2.1.3 线圈 0013H 到 001A 状态

001AH	0019H	0018H	0017H	0016H	0015H	0014H	0013H
1	1	0	0	1	1	0	1

最后一个数据字节中，线圈 0033H 到线圈 0037 的状态为 1BH（二进制 00011011），线圈 0037H 是左数第 4 位，线圈 0033H 为该字节的最低位，线圈 0037H 至线圈 0033H 的状态分别为 ON-ON-OFF-ON-ON，剩余 3 位使用 0 填充。

表 2.1.4 线圈 0033H 到线圈 0037 状态

003AH	0039H	0038H	0037H	0036H	0035H	0034H	0033H
0	0	0	1	1	0	1	1

2.2 读离散输入寄存器 02H

（1）说明

读离散输入寄存器状态。

（2）查询

从机地址为 01H，离散输入寄存器的起始地址为 00C4H，结束寄存器地址为 00D9H。

总共访问 22 个离散输入寄存器。

表 2.2.1 读离散输入寄存器—查询

读离散输入寄存器 02H-主机发送		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	02
3	寄存器起始地址高字节	00
4	寄存器起始地址低字节	C4
5	寄存器数量高字节	00
6	寄存器数量低字节	16
7	CRC 校验低字节	B8
8	CRC 校验高字节	39

（3）响应

响应各离散输入寄存器状态，分别对应数据区中的每位值，1 代表 ON；0 代表 OFF。第一个数据字节的 LSB（最低位）为查询的寻址地址寄存器值，其他输入口按顺序在该字节中由低位向高位节排列，直到填充满 8 位。下一个字节中的 8 个输入位也是从低字节到高字节排列。若返回的输入位数不是 8 的倍数，则在最后的数据字节中的剩余位至该字节的最高位使用 0 填充。

表 2.2.2 读输入寄存器—响应

读离散输入寄存器 02H-模块返回		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	02
3	返回字节数	03
4	数据 1(00C4H-00CBH)	AC
5	数据 2(00CCH-00D3H)	DB
6	数据 3(00D4H-00D9H)	35
7	CRC 校验低字节	22

8	CRC 校验高字节	88

离散输入寄存器 00D4H 到 00D9H 的状态为 35H（二进制 00110101）。输入寄存器 00D9H 为左数第 3 位，输入寄存器 00D4 为最低位，输入寄存器 00D9H 到 00D4H 的状态分别为 ON-ON-OFF-ON-OFF-ON。00DBH 寄存器和 00DAH 寄存器被 0 填充。

表 2.2.3 离散输入寄存器 00C4H 到 00DBH 状态

00CBH	00CAH	00C9H	00C8H	00C7H	00C6H	00C5H	00C4H
0	0	1	1	0	1	0	1
00D3H	**00D2H**	**00D1H**	**00D0H**	**00CFH**	**00CEH**	**00CDH**	**00CCH**
1	1	1	0	1	0	1	1
00DBH	**00DAH**	**00D9H**	**00D8H**	**00D7H**	**00D6H**	**00D5H**	**00D4H**
0	0	1	1	0	1	0	1

2.3 读保持寄存器 03H

（1）说明

读保持寄存器。可读取单个或多个保持寄存器。

（2）查询

从机地址为 01H。保持寄存器的起始地址为 006BH，结束地址为 006DH。该次查询总共访问 3 个保持寄存器。

表 2.3.1 读保持寄存器—查询

读保持寄存器 03H-主机发送		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	03
3	寄存器起始地址高字节	00
4	寄存器起始地址低字节	6B
5	寄存器数量高字节	00
6	寄存器数量低字节	03

7	CRC 校验低字节	74
8	CRC 校验高字节	17

（3）响应

保持寄存器的长度为 2 个字节。对于单个保持寄存器而言，寄存器高字节数据先被传输，低字节数据后被传输。保持寄存器之间，低地址寄存器先被传输，高地址寄存器后被传输。

表 2.3.2 读保持寄存器—响应

读保持寄存器 03H-模块返回		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	03
3	返回字节数	06
4	数据 1 高字节(006BH)	00
5	数据 1 低字节(006BH)	6B
6	数据 2 高字节(006CH)	00
7	数据 2 低字节(006CH)	13
8	数据 3 高字节(006DH)	00
9	数据 3 低字节(006DH)	00
10	CRC 校验低字节	F5
11	CRC 校验高字节	79

表 2.3.3 保持寄存器 006BH 到 006DH 结果

006BH 高字节	006BH 低字节	006CH 高字节	006CH 低字节	006DH 高字节	006DH 低字节
00	6B	00	13	00	00

2.4 读输入寄存器 04H

（1）说明

读输入寄存器命令。该命令支持单个寄存器访问也支持多个寄存器访问。

（2）查询

从机地址为 01H。输入寄存器的起始地址为 0008H，寄存器的结束地址为 0009H。本次访问访问 2 个输入寄存器。

表 2.4.1 读输入寄存器—查询

读输入寄存器 04H-主机发送		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	04
3	寄存器起始地址高字节	00
4	寄存器起始地址低字节	08
5	寄存器数量高字节	00
6	寄存器数量低字节	02
7	CRC 校验低字节	F0
8	CRC 校验高字节	09

（3）响应

输入寄存器长度为 2 个字节。对于单个输入寄存器而言，寄存器高字节数据先被传输，低字节数据后被传输。输入寄存器之间，低地址寄存器先被传输，高地址寄存器后被传输。

表 2.4.2 读输入寄存器—响应

读输入寄存器 04H-模块返回		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	04
3	返回字节数	04
4	数据 1 高字节(006BH)	00
5	数据 1 低字节(006BH)	0A
6	数据 2 高字节(006CH)	00
7	数据 2 低字节(006CH)	0B

8	CRC 校验低字节	9A
9	CRC 校验高字节	41

表 2.4.3 输入寄存器 0008H 到 0009H 结果

006BH 高字节	006BH 低字节	006CH 高字节	006CH 低字节
00	0A	00	0B

2.5 写单个线圈寄存器 05H

（1）说明

写单个线圈寄存器。FF00H 值请求线圈处于 ON 状态，0000H 值请求线圈处于 OFF 状态。05H 指令设置单个线圈的状态，15H 指令可以设置多个线圈的状态，两个指令虽然都设定线圈的 ON/OFF 状态，但是 ON/OFF 的表达方式却不同。

（2）请求

从机地址为 01H，线圈寄存器的地址为 00ACH。使 00ACH 线圈处于 ON 状态，即数据内容为 FF00H。

表 2.5.1 写单个线圈—请求

写单个线圈寄存器 05H-主机发送		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	05
3	寄存器地址高字节	00
4	寄存器地址低字节	AC
5	数据 1 高字节	FF
6	数据 1 低字节	00
7	CRC 校验低字节	4C
8	CRC 校验高字节	1B

（3）响应

2.5.2 写单个线圈—响应

写单个线圈寄存器 05H-模块返回		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	05
3	寄存器地址高字节	00
4	寄存器地址低字节	AC
5	寄存器 1 高字节	FF
6	寄存器 1 低字节	00
7	CRC 校验低字节	4C
8	CRC 校验高字节	1B

2.6 写单个保持寄存器 06H

（1）说明

写保持寄存器。注意 06H 指令只能操作单个保持寄存器，10H 指令可以设置单个或多个保持寄存器。

（2）请求

从机地址为 01H。保持寄存器地址为 0000H。寄存器内容为 0001H。

表 2.6.1 写单个保持寄存器—请求

写单个保持寄存器 06H-主机发送		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	06
3	寄存器地址高字节	00
4	寄存器地址低字节	00
5	数据高字节	00
6	数据低字节	01

7	CRC 校验低字节	48
8	CRC 校验高字节	0A

（3）响应

表 2.6.2 写单个保持寄存器—响应

写单个保持寄存器 06H-模块返回		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	06
3	寄存器地址高字节	00
4	寄存器地址低字节	00
5	寄存器数据高字节	00
6	寄存器数据低字节	01
7	CRC 校验低字节	48
8	CRC 校验高字节	0A

2.7 写多个线圈寄存器 0FH

（1）说明

写多个线圈寄存器。若数据区的某位值为“1”表示被请求的相应线圈状态为 ON，若某位值为“0”，则为状态为 OFF。

（2）请求

从机地址为 01H，线圈寄存器的起始地址为 0013H，线圈寄存器的结束地址为 001CH。总共访问 10 个寄存器。寄存器内容如下表所示。

表 2.7.1 线圈寄存器 0013H 到 001CH

001AH	0019H	0018H	0017H	0016H	0015H	0014H	0013H
1	1	0	0	1	1	0	1
0022H	**0021H**	**0020H**	**001FH**	**001EH**	**001DH**	**001CH**	**001BH**
0	0	0	0	0	0	0	1

传输的第一个字节 CDH 对应线圈为 0013H 到 001AH，LSB（最低位）对应线圈 0013H，传输第二个字节为 01H，对应的线圈为 001BH 到 001CH，LSB（最低位）对应线圈 001BH，其余未使用位使用 0 填充。

表 2.7.2 写多个线圈寄存器—请求

写多个线圈寄存器 0FH-主机发送		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	0F
3	寄存器起始地址高字节	00
4	寄存器起始地址低字节	13
5	寄存器数量高字节	00
6	寄存器数量低字节	0A
7	字节数	02
8	数据 1(0013H-001AH)	CD
9	数据 2(001BH-001CH)	01
10	CRC 校验低字节	72
11	CRC 校验高字节	CB

（3）响应

表 2.7.3 写多个线圈寄存器—响应

写多个线圈寄存器 0FH-模块返回		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	0F
3	寄存器起始地址高字节	00
4	寄存器起始地址低字节	13
5	寄存器数量高字节	00
6	寄存器数量低字节	0A

7	CRC 校验低字节	24
8	CRC 校验高字节	09

2.8 写多个保持寄存 10H

（1）说明

写多个保持寄存器。

（2）请求

从机地址为 01H。保持寄存器的起始地址为 0001H，寄存器的结束地址为 0002H。总共访问 2 个寄存器。保持寄存器 0001H 的内容为 000AH，保持寄存器 0002H 的内容为 0102H。

表 2.8.1 写多个保持寄存器—请求

写多个保持寄存器 10H-主机发送		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	10
3	寄存器起始地址高字节	00
4	寄存器起始地址低字节	01
5	寄存器数量高字节	00
6	寄存器数量低字节	02
7	字节数	04
8	数据 1 高字节	00
9	数据 1 低字节	0A
10	数据 2 高字节	01
11	数据 2 低字节	02
12	CRC 校验低字节	92
13	CRC 校验高字节	30

表 2.8.2 保持寄存器 0001H 到 0002H 内容

0001H	0001H	0002H	0002H

高字节	低字节	高字节	低字节
00	0A	01	12

（3）响应

表 2.8.3 写多个保持寄存器—响应

写多个保持寄存器 10H-模块返回		
字节序号	**功能**	**16 进制数据**
1	从机地址	01
2	功能码	10
3	寄存器起始地址高字节	00
4	寄存器起始地址低字节	01
5	寄存器数量高字节	00
6	寄存器数量低字节	02
7	CRC 校验低字节	10
8	CRC 校验高字节	08

3 CRC 计算

Modbus 通信协议的 CRC（循环冗余校验码）含 2 个字节，即 16 位二进制数。CRC 码由发送设备计算，放置于所发送信息帧的尾部，低字节在前，高字节在后。接收设备再重新计算所接收信息的 CRC，比较计算得到的 CRC 是否与接收到的 CRC 相符，如果两者不相符，则认为数据出错。

Modbus CRC 计算的 C 语言代码如下：

```
/**
  ******************************************************************************
  * @file        ModbusCRC16.h
  * @author      ZhongShengkeji Team
  * @version     V1.2.0
  * @date        01-June-2019
  * @brief       Calculate the CRC16 value of Modbus RTU data frame
  * @copyright   COPYRIGHT 2019 ZhongShengkeji CO.LTD
  * @taobao      https://shop205432927.taobao.com
  * @alibaba     https://shop57528a8a66139.1688.com
  * @website     www.zhongshengkeji.cn
  * @phone       0769-22331829, 138 2574 1827
  * @e-mail      zskjdg@foxmail.com
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef _MODBUS_CRC16_H
#define _MODBUS_CRC16_H

/* Exported functions ------------------------------------------------------- */

void GetModbusCRC16(unsigned char *puchMsg, unsigned short usMsgLen,
                    unsigned char *puchCRCLo, unsigned char *puchCRCHi);

#endif
```

```
/**
  ********************************************************************************
  * @file         ModbusCRC16.c
  * @author       ZhongShengkeji Team
  * @version      V1.2.0
  * @date         01-June-2019
  * @brief        Calculate the CRC16 value of Modbus RTU data frame
  * @copyright    COPYRIGHT 2019 ZhongShengkeji CO.LTD
  * @taobao       https://shop205432927.taobao.com
  * @alibaba      https://shop57528a8a66139.1688.com
  * @website      www.zhongshengkeji.cn
  * @phone        0769-22331829, 138 2574 1827
  * @e-mail       zskjdg@foxmail.com
  ********************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "ModbusCRC16.h"

/* Private variables---------------------------------------------------------*/

/* CRC low byte table */
static const unsigned char auchCRCLo[]=
{
    0x00,0xC0,0xC1,0x01,0xC3,0x03,0x02,0xC2,0xC6,0x06,
    0x07,0xC7,0x05,0xC5,0xC4,0x04,0xCC,0x0C,0x0D,0xCD,
    0x0F,0xCF,0xCE,0x0E,0x0A,0xCA,0xCB,0x0B,0xC9,0x09,
    0x08,0xC8,0xD8,0x18,0x19,0xD9,0x1B,0xDB,0xDA,0x1A,
    0x1E,0xDE,0xDF,0x1F,0xDD,0x1D,0x1C,0xDC,0x14,0xD4,
    0xD5,0x15,0xD7,0x17,0x16,0xD6,0xD2,0x12,0x13,0xD3,
    0x11,0xD1,0xD0,0x10,0xF0,0x30,0x31,0xF1,0x33,0xF3,
    0xF2,0x32,0x36,0xF6,0xF7,0x37,0xF5,0x35,0x34,0xF4,
    0x3C,0xFC,0xFD,0x3D,0xFF,0x3F,0x3E,0xFE,0xFA,0x3A,
    0x3B,0xFB,0x39,0xF9,0xF8,0x38,0x28,0xE8,0xE9,0x29,
    0xEB,0x2B,0x2A,0xEA,0xEE,0x2E,0x2F,0xEF,0x2D,0xED,
    0xEC,0x2C,0xE4,0x24,0x25,0xE5,0x27,0xE7,0xE6,0x26,
    0x22,0xE2,0xE3,0x23,0xE1,0x21,0x20,0xE0,0xA0,0x60,
    0x61,0xA1,0x63,0xA3,0xA2,0x62,0x66,0xA6,0xA7,0x67,
    0xA5,0x65,0x64,0xA4,0x6C,0xAC,0xAD,0x6D,0xAF,0x6F,
    0x6E,0xAE,0xAA,0x6A,0x6B,0xAB,0x69,0xA9,0xA8,0x68,
    0x78,0xB8,0xB9,0x79,0xBB,0x7B,0x7A,0xBA,0xBE,0x7E,
    0x7F,0xBF,0x7D,0xBD,0xBC,0x7C,0xB4,0x74,0x75,0xB5,
    0x77,0xB7,0xB6,0x76,0x72,0xB2,0xB3,0x73,0xB1,0x71,
```

```
    0x70, 0xB0, 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
    0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
    0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B,
    0x99, 0x59, 0x58, 0x98, 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B,
    0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
    0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42,
    0x43, 0x83, 0x41, 0x81, 0x80, 0x40
};

/* CRC high byte table */
static const unsigned char auchCRCHi[]=
{
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
    0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
    0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,
    0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
    0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
    0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
    0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,
    0x01, 0xC0, 0x80, 0x41, 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
};
```

```
/* Public functions ---------------------------------------------------------*/

/**
  * @brief  Calculate Modbus CRC16
  * @param  puchMsg: Data to calculate CRC
  * @param  usMsgLen: Data length
  * @param  puchCRCHi: High byte of CRC value
  * @param  puchCRCLo: Low byte of CRC value
  * @note   Send the low byte first
  * @retval None
  */
void GetModbusCRC16(unsigned char *puchMsg, unsigned short usMsgLen,
                    unsigned char *puchCRCLo, unsigned char *puchCRCHi)
{
    unsigned char uchCRCLo = 0xFF;
    unsigned char uchCRCHi = 0xFF;
    unsigned short uIndex = 0;

    while(usMsgLen--)
    {
        uIndex = (unsigned char)(uchCRCHi^*puchMsg++);
        uchCRCHi = (unsigned char)(uchCRCLo^auchCRCHi[uIndex]);
        uchCRCLo = (unsigned char)(auchCRCLo[uIndex]);
    }

    *puchCRCLo = uchCRCHi;
    *puchCRCHi = uchCRCLo;
}
```

4 公司信息

中盛科技（东莞）有限公司是一家专注于研发、生产及销售工业自动化产品和提供自动化解决方案的高新技术企业。中盛科技掌握行业领先的“检测与控制”技术，利用我们多年的经验，以及对自动化现场的深刻理解，不断满足客户对产品多样化和高品质的追求。

公司技术和研发实力雄厚，硬件电路设计、软件开发及通讯技术专家和研发人员占比40%以上，拥有 50 多项专利和软件著作成果及 10 多个产品系列。目前主要的产品系列有数字量输入输出、模拟量输入输出、温度/湿度采集、交流采集、脉冲输入输出、数码管显示屏、接口转换等系列。广泛应用于电力系统、智能交通、工业自动化、物联网、矿产能源、安防系统和智能家居等领域，积累了大量成功经验，是国内领先的工业自动化产品与解决方案提供商。

公司联系信息如下：

- 名　　称：中盛科技（东莞）有限公司
- 地　　址：广东省东莞市东城街道立新社区光大路北一街 1 号鑫鸿源产业园
- 电　　话：0769-22331829
- 联 系 人：朱盛方
- 手　　机：138 2574 1827
- 邮　　箱：zskjdg@foxmail.com
- 网　　址：www.zhongshengkeji.cn
- 淘　　宝：https://shop205432927.taobao.com
- 阿　　里：https://shop57528a8a66l39.1688.com

中盛微信　公众号　淘宝　阿里巴巴

谢谢！