森岛变频器 SD680 通信协议

第一章：通信协议

SD680 采用 ModBus 通信协议

该协议定义了 ModBus 总线 MASTER（主站）与 SLAVE（从站）之间的通讯报文格式，对于主站来说，ModBus 协议是联系 PLC/PC 机的接口，而且所有的通讯都是“透明的”。 ModBus 协议是工业最常用的 PLC 通信协议， 包含 RTU 及 ASCII 两种格式，两种格式的报文各字段解释是相同的。他们之间最大的差别是他们执行错误核对的方式与字符的格式。主机可对各从机寻址，发出广播信息，从机返回信息作为对查询的响应。从机对于主机的广播查询，无响应则返回 ModBus 协议报据设备地址，请求功能代码，发送数据，错误校验码。建立了主机查询格式，从机的响应信息也用 ModBus 协议组织，它包括确认动作的代码，返回数据和错误校验码。若在接收信息时出现一个错误或从机不能执行要求的动作时，从机会组织一个错误信息，并向主机发送作为响应。关于该协议详细信息请参考 ModBus 通信协议。

SD680 通信模式

RTU 模式

控制器以 RTU 模式在 Modbus 总线上进行通讯时，信息中的每 8 位字节分成 2 个 4 位 16 进制的字符，该模式的主要优点是在相同波特率下其传输的字符的密度高于 ASCII 模式，每个信息必须连续传输。

RTU 模式中每个字节的格式：

编码系统：8 位二进制，十六进制 0-9，A-F
数据位： 1 起始位，8 位数据位，低位先送
奇/偶校验时 1 位；无奇偶校验时 0 位
带校验时 1 位停止位；无校验时 2 位停止位
错误校验区：循环冗余校验(CRC)

SD680Modbus 信息帧

RTU 帧

RTU 模式中，信息开始至少需要有 3.5 个字符的静止时间，依据使用的波特率，很容易计算这个静止的时间。接着，第一个区的数据为设备地址。各个区允许发送的字符均为 16 进制的 0-9,A-F。 网络上的设备连续监测网络上的信息，包括静止时间。当接收第一个地址数据时，每台设备立即对它解码，以决定是否是自己的地址。发送完最后一个字符号后，也有一个 3.5 个字符的静止时

间，然后才能发送一个新的信息。 整个信息必须连续发送。如果在发送帧信息期间，出现大于 1.5 个字符的静止时间时，则接收设备刷新不完整的信息，并假设下一个地址数据。同样一个信息完成后,立即发送一个新的信息,(若无 3.5 个字符的静止时间）这将会产生一个错误，是因为合并信息的 CRC 校验码无效而产生的错误。

典型的信息帧见下表：

开始	地址	功能	数据	校验	终止
3.5 个字符时间	8bit	8bit	N×8bit	16bit	3.5 个字符时间

SD680 地址设置

信息地址包括 8 位(RTU)，有效的从机设备地址范围 0-247,(十进制)，各从机设备的寻址范围为 1-247。主机把从机地址放入信息帧的地址区，并向从机寻址。从机响应时，把自己的地址放入响应信息的地址区，让主机识别已作出响应的从机地址。当系统使用 RS-485 串联通讯介面控制或监控时，每一台驱动器必须设定其通讯地址且每一个连结网中每个地址均为“唯一”不可重复。地址出厂设置值：01。

广播发送地址：0

从机地址：1～247（0x01～0xF7）

SD680 功能码设置

信息帧功能代码包括 8 位(RTU)。有效码范围 1-225(十进制)，功能代码可说明从机正常响应或出现错误(即不正常响应)，正常响应时，从机简单返回原始功能代码，不正常响应时，从机返回与原始代码相等效的一个码，并把最高有效位设定为“1”。

如，主机要求从机读主频率源数字频率时，则发送信息的功能码为：

0000 0011 (十六进制 03)

若从机正确接收请求的动作信息后，则返回相同的代码值作为正常响应。发现错时，则返回一个不正常响信息：

1000 0011(十六进制 83)

从机对功能代码作了修改,此外,还把一个特殊码放入响应信息的数据区中,告诉主机出现的错误类型和不正常响应的原因。主机设备的应用程序负责处理不正常响应,典型处理过程是主机把对信息的测试和诊断送给从机,并通知操作者。

SD680 数据区的内容

数据区有 n 个 16 进制的数据位，数据范围为 00-FF(16 进制)，根据网络串行传输的方式，数据区可由一个 RTU 字符组成。 主机向从机设备发送的信息数据中包含了从机执行主机功能代码中规定的请求动作，如离散量寄存器地址，处理对象的数目，以及实际的数据字节数等。

举例说明：若主机请求从机读一组寄存器（功能代码 03），该数据规定了寄存器的起始地址，以及寄存器的数量。又如，主机要在一从机中写一组寄存器，（则功能代码为 10H），该数据区规定了要写入寄存区的起始地址，寄存器的数量，数据的字节数，以及要写入到寄存器的数据。

若无错误出现，从机向主机的响应信息中包含了请求数据，若有错误出现，则数据中有一个不正常代码，使主机能判断并作出下一步的动作。

SD680 信息帧错误校验

RTU

使用 RTU 方式时，错误校验码为一个 16 位的值，2 个 8 位字节。错误校验值是对信息内容执行 CRC 校验结果。CRC 校验信息帧是最后的一个数据，得到的校验码先送低位字节，后送高位字节，所以 CRC 码的高位字节是最后被传送的信息。

SD680 串行传送信息

在标准的 Modbus 上传送的信息中，每个字符或字节，按由左向右的次序传送：

最低有效位：（LSB）最高有效位：（MSB）

RTU 数据帧位序:

带奇偶校验

Start	1	2	3	4	5	6	7	8	Par	Stop

无奇偶校验

Star	1	2	3	4	5	6	7	8/	Stop	Stop

SD680 错误校验方法

标准的 Modbus 串行通讯网络采用两种错误校验方法，奇偶校验(奇或偶)可用于校验每一个字符，信息帧校验(LRC 或 CRC)适用整个信息的校验，字符校验和信息帧校验均由主机设备产生，并在传送前加到信息中去。从机设备在接收信息过程中校验每个字符和整个信息。

主机可由用户设置的一个预定时间间隔，确定是否放弃传送信息。该间隔应有足够的时间来满足从机的正常响应。若主机检测到传输错误时，则传输的信息无效。从机不再向主机返回响应信息。此时，主机会产生一个超时信息，并允许主机程序处理该错误信号。注意：主机向实际并未存在的从机发送信息时也会引起超时出错信号。

在 MAP 或 Modbus+等其它网络上使用时，采用比 Modbus 更高一级的数据帧校验方法。在这些网络中，不再运用 Modbus 中的 LRC 或 CRC 校验方法。当出现发送错误时，网络中的通讯协议通知发送设备有错误出现，并允许根据设置的情况，重试或放弃信息发送。若信息已发送，但从机设备未作响应，则主机通过程序检查后发出一个超时错误。

奇偶校验

用户可设置奇偶校验或无校验，以此决定每个字符发送时的奇偶校验位的状态。无论是奇或偶校验，它均会计算每个字符数据中值为“1”的位数，ASCII 方式为位数据；RTU 方式为 8 位数据。并根据“1”的位数值(奇数或偶数)来设定为“0”或“1”

如一个 RTU 数据帧中 8 位数据位为：

1100 0101

在该帧中，值为“1”的总位数为 4，即偶数。如采用奇校验方式时，则 “1”的总位数为奇数，即 5。 发送信息时，计算奇偶位，并加到数据帧中，接收设备统计位值为“1”的数量，若与该设备要求的不一致时产生一个错误。在 Modbus 总线上的所有设备必须采用相同的奇偶校验方式。

注意：奇偶校验只能检测到数据帧在传输过程中丢失奇数“位”时才产生的错误。如采用奇数校验方式时，一个包含 3 个“1”位的数据丢失 2 个“1”位时，其结果仍然是奇数。若无奇偶校验方式时，传输中不作实际的校验，应附加一个停止位。

CRC 校验

RTU 方式时，采用 CRC 方法计算错误校验码， CRC 校验传送的全部数据。它忽略信息中单个字符数据的奇偶校验方法。

CRC 码为 2 个字节，16 位的二进制值。由发送设备计算 CRC 值，并把它附到信息中去。接收设备在接收信息过程中再次计算 CRC 值并与 CRC 的实际值进行比较，若二者不一致，亦产生一个错误，校验开始时，把 16 位寄存器的各位都置为“1”，然后把信息中的相邻 2 个 8 位字节数据放到当前寄存器中处理，只有每个字符的 8 位数据用于 CRC 处理。起始位，停止位和校验位不参与 CRC 计算。 CRC 校验时，每个 8 位数据与该寄存器的内容进行异或运算，然后向最低有效位(LSB)方向移位，用零填入最高有效位(MSB)后，再对 LSB 检查，若 LSB=1，则寄存器与预置的固定值异或，若 LSB=0，不作异或运算。

重复上述处理过程，直至移位 8 次，最后一次(第 8 次)移位后，下一个 8 位字节数据与寄存器的当前值异或，再重复上述过程。全部处理完信息中的数据字节后，最终得到的寄存器值为 CRC 值。

CRC 值附加到信息时，低位在先，高位在后。在梯形图中，CKSM 函数计算信息中的 CRC 值。用于主计算机时，可查阅附录中的一个实例，它详细说明了 CRC 的校验。

第二章：功能说明

数字值表达

若无特殊说明在此节文中用＋进制值表示，图中的数据区则用十六进制表示。

SD680 支持的功能代码

功能码	功能说明
03	读多个寄存器
06	写单个寄存器
10	连续写多个寄存器
13	读单个参数(带最大最小值)

SD680 功能代码格式

功能码 03：读寄存器内容

主机询问：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x03
寄存器地址	2	0x0000～0x0FFF
寄存器数目	2	0x0000～0x0008
校验	CRC	

从机应答：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x03
读取字节数	1	2*寄存器数目
寄存器数据	2*寄存器数目	
校验	CRC	

寄存器地址及寄存器数目说明：

	寄存器地址	寄存器数目：
状态	0xA000	0x0001～0x0001
监控参数	0xD000～0xD027	0x0001～0x0008
控制参数	0x0000～0x0FFF	0x0001～0x0008
错误码	0xE000	0x0001～0x0001
报警码	0xE001	0x0001～0x0001
MODBUS 设定频率	0x2000	0x0001～0x0001

功能码 06：写寄存器内容

主机询问：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x06
寄存器地址	2	0x0000～0x0FFF
寄存器数据	2	0x0000～0x0008
校验	CRC	

从机应答：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x06
寄存器地址	2	0x0000～0x0FFF
寄存器数据	2	见操作说明
校验	CRC	

寄存器地址说明：

控制参数	0x0000～0x0FFF
控制命令	0x2000
MODBUS 设定频率	0x2001

功能码 0x13：读参数（包括属性、最小值、最大值）

主机询问：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x13
寄存器地址	2	0x0000～0x0FFF
寄存器数目	2	0x0001～0x0004
校验	CRC	

从机应答：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x13
读取字节数	1	2*寄存器数目
寄存器数据	2*寄存器数目	
校验	CRC	

寄存器地址及寄存器数目说明：

	寄存器地址	寄存器数目：
监控参数	0xD000～0xD027	0x0001～0x0002
控制参数	0x0000～0x 0FFF	0x0001～0x0004
MODBUS 设定频率	0x2001	0x0001～0x0004

功能码 0x10：连续写参数

主机询问：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x10
寄存器地址	2	0x0000～0x0FFF
寄存器数目	2	0x0001～0x0008
写寄存器内容	2*寄存器数目	
校验	CRC	

从机应答：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x10
寄存器地址	2	0x0000～0x0FFF
寄存器数目	2	0x0001～0x0008
写寄存器内容	2*寄存器数目	
校验	CRC	

寄存器地址及寄存器数目说明：

	寄存器地址	寄存器数目：
控制命令	0x2000	0x0001～0x0002
控制参数	0x0000～0x0FFF	0x0001～0x0008
MODBUS 设定频率	0x2001	0x0001～0x0001

发生错误时，回应格式为：

	数据字节长度	数据范围
地址	1	0～247
功能码	1	0x80+功能码
错误代码	1	
校验	CRC	

错误代码：

0x01	非法功能码	0x06	参数运行中不可修改
0x02	非法地址	0x07	参数不可修改
0x03	非法数据	0x08	上位机控制命令无效
0x04	非法寄存器长度	0x09	参数受密码保护
0x05	CRC 校验错误	0x0A	密码错误

控制命令字格式：

位	含义
Bit7~Bit5	保留
Bit4	0:无动作 1:复位
Bit3	0:正转 1:反转
Bit2~Bit0	100:自由停机 011:停机 010:点动运行 001:运行

第三章：操作说明：

主工作方式

变频器工作在主方式(广播发送),变频器地址设为 00：

地址	功能码	寄存器地址	寄存器数目	数据	校验码
00	10	20 00	00 02	B3~B0	XX XX

数据含义：

字节	位	含义
Byte3	Bit7~Bit0	保留
Byte2	Bit7~Bit5	保留
	Bit4	0:无动作 1:复位
	Bit3	0:正转 1:反转
	Bit2~Bit0	100:自由停机 011:停机 010:点动运行 001:运行
Byte1	Bit7~Bit0	最终频率高 8 位
Byte0	Bit7~Bit0	最终频率低 8 位

从工作方式（地址默认 01)：

读功能

03H 读多个（含一个）寄存器

1)读状态寄存器

主机询问：

地址	功能码	寄存器地址	寄存器数目	校验码
01	03	A0 00	00 01	A6 0A

从机应答：

地址	功能码	字节总数	数据	校验码
01	03	02	B1 B0	XX XX

应答数据含义：

字节	位	含义
Byte1	Bit7	0:无动作 1：OVERLOAD_ALARM
	Bit6~Bit5	0:INV_220V 1:INV_380V 2:INV_660V 3:INV_1140V
	Bit4	0:NO_ACTION 1:PWR_OFF_SAVE
	Bit3	0:NO_ACTION 1:PWR_ON_FIRST
	Bit2~Bit1	0:NO_ACTION 1:TUNE_STATIC 2:TUNE_ALL
	Bit0	0:KEY 1:Terminal 2:Modbus 3:Extern card
Byte0	Bit7	
	Bit6	0:NO_ACTION 1:DC_BUILD
	Bit5	0:NO_ACTION 1:DC_LOWER
	Bit4	0:NO_ACTION 1:JOG
	Bit3	0:FOR 1:REV
	Bit2~Bit1	1:SPEED_ACC 2:SPEED_DEC 3:SPEED_NO_CHANGE
	Bit0	0:停机状态 1:运行状态

2)读监控参数

主机询问：

地址	功能码	寄存器地址	寄存器数目	校验码
01	03	XX XX	00 0X	XX XX

从机应答：

地址	功能码	字节总数	数据	校验码
01	03	2×寄存器数目	Bn~B0	XX XX

寄存器地址：D0 00~D0 27
寄存器数目：00 01~00 08
数据:n 为 2×寄存器数目-1

应答数据含义：

地址	功能说明	地址	功能说明
D000	输出频率(Hz)	D020	输出端子状态
D001	设定频率(Hz)	D021	模块温度 1(℃)
D002	输入电压(V)	D022	模块温度 2(℃)
D003	输出电压(V)	D023	当前计数值
D004	母线电压(V)	D024	设定计数值
D005	设定电压	D025	当前定时值
D006	输出电流(A)	D026	设定定时值
D007	输出转矩(%)	D027	设定长度
D008	电机转速(RPM/min)	D028	当前长度
D009	实测转速(RPM/min)	D029	保留
D010	运行线速度(m/s)	D030	第三次故障代码
D011	设定线速度(m/s)	D031	第二次故障代码
D012	保留	D032	最近一次故障代码
D013	PID 设定值	D033	最近一次故障时的变频器运行状态
D014	PID 反馈值	D034	最近一次故障时输出频率(Hz)
D015	模拟输入 AI1(V)	D035	最近一次故障时设定频率(Hz)
D016	模拟输入 AI2(V)	D036	最近一次故障时输出电流(A)
D017	脉冲输入频率(KHz)	D037	最近一次故障时输出电压(V)
D018	脉冲输出频率(KHz)	D038	最近一次故障时母线电压(V)
D019	输入端子状态	D039	最近一次故障时模块温度(℃)

3)读控制参数

主机询问：

地址	功能码	寄存器地址	寄存器数目	校验码
01	03	XX XX	00 0X	XX XX

从机应答：

地址	功能码	字节总数	数据	校验码
01	03	2×寄存器数目	Bn~B0	XX XX

寄存器地址：00 00~0F FF,详见参数表
寄存器数目：00 01~00 08
数据:n 为 2×寄存器数目-1

4)读 ModBus 设定频率

主机询问：

地址	功能码	寄存器地址	寄存器数目	校验码
01	03	XX XX	00 0X	XX XX

从机应答：

地址	功能码	字节总数	数据	校验码
01	03	2×寄存器数目	Bn~B0	XX XX

寄存器地址：20 00
寄存器数目：00 01
数据:n 为 2×寄存器数目-1

5)读错误或警告

主机询问：

地址	功能码	寄存器地址	寄存器数目	校验码
01	03	XX XX	00 0X	XX XX

从机应答：

地址	功能码	字节总数	数据	校验码
01	03	2×寄存器数目	Bn~B0	XX XX

寄存器地址：E0 00~E0 01
寄存器数目：00 01~00 01
数据:n 为 2×寄存器数目-1
注意：寄存器地址+寄存器数目≤E0 01

13H 读参数（包括属性、最小值、最大值）

1)读控制参数（包括属性、最小值、最大值）

主机询问：

地址	功能码	寄存器地址	寄存器数目	校验码
01	13	XX XX	00 0X	XX XX

从机应答：

地址	功能码	字节总数	数据	校验码
01	13	2×寄存器数目	Bn~B0	XX XX

寄存器地址：00 00~0F FF，详见参数表
寄存器数目：00 01~00 04
00 01：参数值
00 02：参数值+属性

00 03：参数值+属性+最小值
00 04：参数值+属性+最小值+最大值
数据:n 为 2×寄存器数目-1

属性含义：

位	含义		
Bit15	保留		
Bit14	菜单		
Bit13	进制		
Bit12	恢复出厂值覆盖		
Bit11	EEPROM		
Bit10~Bit9	″○″:01 ″×″:10 ″◆″:11 ″◇″:00		
Bit8	符号		
Bit7~Bit3	1:00000 V:00001 A:00010 rpm:00011 HZ:00100 %:00110 S:01000	KHZ:01100 KW:01010 om:01110 ms:01001 MA:01011 KM:01101 CM:01111	us:10001 HZ/S:10000 mh:10010 C:10011 m/s:10100 H:10101 KWH:10110
Bit2~Bit0	小数点		

2)读 ModBus 设定频率（包括额定值、极值）

主机询问：

地址	功能码	寄存器地址	寄存器数目	校验码
01	13	20 01	00 0X	XX XX

从机应答：

地址	功能码	字节总数	数据	校验码
01	13	2×寄存器数目	Bn~B0	XX XX

寄存器数目：00 01~00 04
00 01：参数值
00 02：参数值+额定值
00 03：参数值+额定值+极小值
00 04：参数值+额定值+极小值+极大值
数据:n 为 2×寄存器数目-1

3)读监控参数（包括属性）

主机询问：

地址	功能码	寄存器地址	寄存器数目	校验码
01	13	XX XX	00 0X	XX XX

从机应答：

地址	功能码	字节总数	数据	校验码
01	13	2×寄存器数目	Bn~B0	XX XX

寄存器地址：D0 00~D0 27

寄存器数目：00 01~00 02

数据:n 为 2×寄存器数目-1

应答数据含义见 03H 中读监控参数

写功能

06H 写单个参数数据不保存

1) 写控制命令

主机询问：

地址	功能码	寄存器地址	数据	校验码
01	06	20 00	B1 B0	XX XX

从机应答：

地址	功能码	寄存器地址	数据	校验码
01	06	20 00	B1 B0	XX XX

数据含义：

字节	位	含义
Byte1	Bit7~Bit0	保留
Byte0	Bit7~Bit5	保留
	Bit4	0:无动作 1:复位
	Bit3	0:正转 1:反转
	Bit2~Bit0	100:自由停机 011:停机 010:点动运行 001:运行

2) 写控制参数

主机询问：

地址	功能码	寄存器地址	数据	校验码
01	06	XX XX	B1 B0	XX XX

从机应答：

地址	功能码	寄存器地址	数据	校验码
01	06	XX XX	B1 B0	XX XX

寄存器地址：00 00~0F FF,详见参数表

3）写 ModBus 频率设定

主机询问：

地址	功能码	寄存器地址	数据	校验码
01	06	20 01	B1 B0	XX XX

从机应答：

地址	功能码	寄存器地址	数据	校验码
01	06	20 01	B1 B0	XX XX

10H 写多个寄存器数据不保存

1）写控制参数

主机询问：

地址	功能码	寄存器地址	寄存器数目	数据	校验码
01	10	20 00	00 0X	Bn~B0	XX XX

从机应答：

地址	功能码	寄存器地址	寄存器数目	数据	校验码
01	10	20 00	00 0X	Bn~B0	XX XX

寄存器数目：00 01~00 02

00 01：控制命令

00 02：控制命令+ModBus 设定频率

2)写控制参数

主机询问：

地址	功能码	寄存器地址	寄存器数目	数据	校验码
01	10	XX XX	00 0X	Bn~B0	XX XX

从机应答：

地址	功能码	寄存器地址	寄存器数目	数据	校验码
01	10	XX XX	00 0X	Bn~B0	XX XX

寄存器地址：00 00~0F FF，详见参数表

寄存器数目：00 01~00 08

附录

CRC 生成

CRC 循环冗余校验

循环冗余校验 CRC 区为 2 字节，含一个 16 位二进制数据。由发送设备计算 CRC 值，并把计算值附在信息中，接收设备在接收信息时，重新计算 CRC 值，并把计算值与接收的在 CRC 区中实际值进行比较,若两者不相同,则产生一个错误。

CRC 开始时先把寄存器的 16 位全部置成“1”,然后把相邻 2 个 8 位字节的数据放入当前寄存器中，只有每个字符的 8 位数据用作产生 CRC，起始位，停止位和奇偶校验位不加到 CRC 中。

产生 CRC 期间，每 8 位数据与寄存器中值进行异或运算，其结果向右移一位(向 LSB 方向)，并用“0”填入 MSB，检测 LSB，若 LSB 为“1”则与预置的固定值异或，若 LSB 为“0”则不作异或运算。

重复上述处过程，直至移位 8 次，完成第 8 次移位后，下一个 8 位数据,与该寄存器的当前值异或，在所有信息处理完后，寄存器中的最终值为 CRC 值。

产生 CRC 的过程：

1. 把 16 位 CRC 寄存器置成 FFFFH.
2. 第一个 8 位数据与 CRC 寄存器低 8 位进行异或运算，把结果放入 CRC 寄存器。
3. CRC 寄存器向右移一位，MSB 填零，检查 LSB.
4. (若 LSB 为 0):重复 3，再右移一位。

   (若 LSB 为 1):CRC 寄存器与 Aool H 进行异或运算
5. 重复 3 和 4 直至完成 8 次移位，完成 8 位字节的处理。
6. 重复 2 至 5 步，处理下一个 8 位数据，直至全部字节处理完毕。
7. CRC 寄存器的最终值为 CRC 值。
8. 把 CRC 值放入信息时，高 8 位和低 8 位应分开放置。

把 CRC 值放入信息中

发送信息中的 16 位 CRC 值时，先送低 8 位，后送高 8 位。

若 CRC 值为 1241(0001 0010 0100 0001):

Addr	Func	Data Count	Data	Data	Data	Data	CRC Hi	CRC Lo
							41	12

例：

各种可能的 CRC 值，按两列装入，一列在 16 位 CRC 的高 8 位区，为(0-256 的)CRC 值，另一类为低 8 位区，为 CRC 的低位值。

用这种方法得到的 CRC 其执行速度快于计算缓冲器中每个新字符得到一个新 CRC 值的方法。

Ú 注意：该功能内部交换 CRC 中的高/低字节，返回的 CRC 值中，其字节已交换。

因此，由功能码返回的 CRC 值，能直接放在信息中传送。

CRC 生成

功能函数

```
u16 CRC_16BIT_ChkSum( u8 *chkbuf, u8 len )
{
   u8  uIndex;
   u8  uchCRCHi = 0xff;
   u8  uchCRCLo = 0xff;
   u16 chk_sum;

   while (len--){
      uIndex =  uchCRCHi ^ *chkbuf++;
      uchCRCHi = uchCRCLo ^ auchCRCHi_exp[uIndex];
      uchCRCLo = auchCRCLo_exp[uIndex];
   }

   chk_sum = uchCRCHi;
   chk_sum = ((u16)chk_sum<<8) | (uchCRCLo & 0xff);

   return chk_sum;

}
```

高位字节表

```
/* Table of CRC values for high-order byte */

const u8 auchCRCHi_exp[] = {
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
    } ;
```

低位字节表

```
    /* Table of CRC values for low-order byte */

    const u8 auchCRCLo_exp[] = {
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
    } ;
```