COMMUNICATION PROTOCOL

🕮 INTERFACE

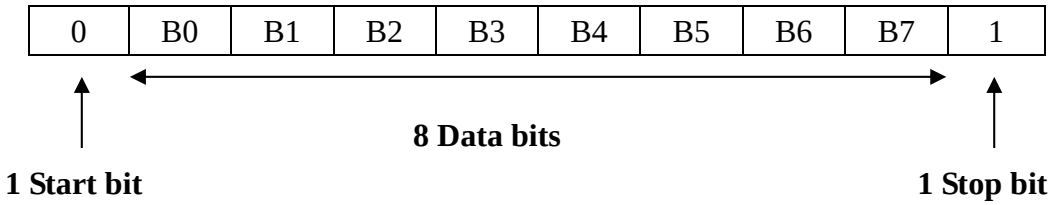
RS-485, RS-232

🕮 BAUD RATE

2400, 4800, 9600, 19200, 38400 bps

🕮 DATA FRAME

Data Bits = 8, Parity = None, Start bit = 1, Stop bit = 1



🕮 DATA FORMAT

ModBus Protocol RTU Mode

RTU Request：Read command

0	1	2	3	4	5	6	7
	0x03						
Station Number	**Func-tion**	**Address (MSB LSB)**		**Count (MSB LSB)**		**CRC16 (LSB MSB)**	

Station Number: 00H~1FH

Address: 0000H~0100H

Count: Number of Data

CRC16: Cyclical Redundancy Check

RTU Response：Read command

0	1	2	3	4	5	6	7	8
	0x03							
Station Number	**Func-tion**	**Byte Count**	**Data1 (MSB LSB)**		**Data2.. (MSB LSB)**		**CRC16 (LSB MSB)**	

Station Number: 00H~1FH

Address: 0000H~0100H

Byte Count: Number of Data Bytes

CRC16: Cyclical Redundancy Check

RTU Request：Write command

0	1	2	3	4	5	6	7
	0x06						
Station Number	**Func-tion**	**Address (MSB LSB)**		**Data (MSB LSB)**		**CRC16 (LSB MSB)**	

Station Number: 00H~1FH

Address: 0000H~00B6H

CRC16: Cyclical Redundancy Check

RTU Response：Write command

0	1	2	3	4	5	6	7
	0x06						
Station Number	**Func-tion**	**Address (MSB LSB)**		**Data (MSB LSB)**		**CRC16 (LSB MSB)**	

Station Number: 00H~1FH

Address: 0000H~00B6H

CRC16: Cyclical Redundancy Check

🕮 COMMUNICATION EXAMPLES

RTU Request：Read command

🗁 Read PV from station 1

- ☞ Station number: 01H
- ☞ Function: 03H
- ☞ Address MSB: 01H
- ☞ Address LSB: 00H
- ☞ Count MSB: 00H
- ☞ Count LSB: 01H
- ☞ CRC16 LSB: 85H
- ☞ CRC16 MSB: F6H

0	1	2	3	4	5	6	7
0x01	0x03	0x01	0x00	0x00	0x01	0x85	0xF6
Station Number	**Func-tion**	**Address (MSB LSB)**		**Count (MSB LSB)**		**CRC16 (LSB MSB)**	

RTU Response：Read command

- Response PV 100.0℃ from D-Series station 1
  - Station number: 01H
  - Function: 03H
  - Byte count: 02H
  - Data MSB: 03H
  - Data LSB: E8H
  - CRC16 LSB: B8H
  - CRC16 MSB: FAH

0	1	2	3	4	5	6
0x01	0x03	0x02	0x03	0xE8	0xB8	0xFA
Station Number	**Func-tion**	**Byte Count**	**Data1 (MSB LSB)**		**CRC16 (LSB MSB)**	

RTU Request：Write command

- Write SV 100.0℃ to D-Series station 1
  - Station number: 01H
  - Function: 06H
  - Address MSB: 00H
  - Address LSB: 02H
  - Data MSB: 03H
  - Data LSB: E8H
  - CRC16 LSB: 28H
  - CRC16 MSB: B4H

0	1	2	3	4	5	6	7
0x01	0x06	0x00	0x02	0x03	0xE8	0x28	0xB4
Station Number	**Func-tion**	**Address (MSB LSB)**		**Data (MSB LSB)**		**CRC16 (LSB MSB)**	

RTU Response：Write command

- Response request command from D-Series station 1
  - Station number: 01H
  - Function: 06H
  - Address MSB: 00H
  - Address LSB: 02H
  - Data MSB: 03H
  - Data LSB: E8H
  - CRC16 LSB: 28H
  - CRC16 MSB: B4H

0	1	2	3	4	5	6	7
0x01	0x06	0x00	0x02	0x03	0xE8	0x28	0xB4
Station Number	**Func-tion**	**Address (MSB LSB)**		**Data (MSB LSB)**		**CRC16 (LSB MSB)**	

🕮 ADDRESS INDEX

PARA	ADDR	PARA	ADDR	PARA	ADDR
Pv	0100H	**Ar**	001FH	**O1HS**	003FH
LEvL	0000H	**P2**	0020H	**AO**	0040H
LoCK	0001H	**i2**	0021H	**O2LS**	0041H
Sv	0002H	**d2**	0022H	**O2HS**	0042H
OutL	0003H	**Ct2**	0023H	**t1SS**	0043H
At	0004H	**HSt2**	0024H	**t1On**	0044H
mAn	0005H	**db**	0025H	**t1ES**	0045H
AL1S	0006H	**SSv**	0026H	**t1oF**	0046H
AL1L	0007H	**Sout**	0027H	**t2SS**	0047H
AL1U	0008H	**StmE**	0028H	**t2On**	0048H
AL2S	0009H	**rUCy**	0029H	**t2ES**	0049H
AL2L	000AH	**rPt**	002AH	**t2oF**	004AH
AL2U	000BH	**StAt**	002BH	**inP1**	004BH
AL3S	000CH	**PvSt**	002CH	**LoSP**	004CH
AL3L	000DH	**wAit**	002DH	**HiSP**	004DH
AL3U	000EH	**Pid**	002EH	**LoAn**	004EH
SOAK	000FH	**EndP**	002FH	**HiAn**	004FH
rAmP	0010H	**AL1F**	0030H	**A1LS**	0050H
PvoF	0011H	**AL1H**	0031H	**A1HS**	0051H
Pvrr	0012H	**AL1t**	0032H	**unit**	0052H
SvoF	0013H	**AL1m**	0033H	**dP**	0053H
Ct	0014H	**AL2F**	0034H	**FiLt**	0054H
HbA	0015H	**AL2H**	0035H	**inP2**	0055H
LbA	0016H	**AL2t**	0036H	**A2LS**	0056H
Lbd	0017H	**AL2m**	0037H	**A2HS**	0057H
rPtm	0018H	**AL3F**	0038H	**bAud**	0059H
P1	0019H	**AL3H**	0039H	**Addr**	005AH
i1	001AH	**AL3t**	003AH	**LEv1**	005BH
d1	001BH	**AL3m**	003BH	**LEv2**	005CH
Ct1	001CH	**Act**	003CH	**LEv3**	005DH
HSt1	001DH	**Outm**	003DH	**LvSL**	005EH
AtoF	001EH	**O1LS**	003EH	**L1P1**	005FH

PARA	ADDR	PARA	ADDR	PARA	ADDR
L1i1	0060H	**tS1**	0080H	**1-12**	00A0H
L1d1	0061H	**Sv2**	0081H	**1-13**	00A1H
L1Ar	0062H	**tP2**	0082H	**1-14**	00A2H
L1P2	0063H	**tS2**	0083H	**1-15**	00A3H
L1i2	0064H	**Sv3**	0084H	**1-16**	00A4H
L1d2	0065H	**tP3**	0085H	**1-17**	00A5H
L2P1	0066H	**tS3**	0086H	**1-18**	00A6H
L2i1	0067H	**Sv4**	0087H	**1-19**	00A7H
L2d1	0068H	**tP4**	0088H	**1-20**	00A8H
L2Ar	0069H	**tS4**	0089H	**1-21**	00A9H
L2P2	006AH	**Sv5**	008AH	**1-22**	00AAH
L2i2	006BH	**tP5**	008BH	**2-14**	00ABH
L2d2	006CH	**tS5**	008CH	**2-15**	00ACH
L3P1	006DH	**Sv6**	008DH	**2-16**	00ADH
L3i1	006EH	**tP6**	008EH	**2-17**	00AEH
L3d1	006FH	**tS6**	008FH	**3-20**	00AFH
L3Ar	0070H	**Sv7**	0090H	**3-21**	00B0H
L3P2	0071H	**tP7**	0091H	**3-22**	00B1H
L3i2	0072H	**tS7**	0092H	**3-23**	00B2H
L3d2	0073H	**Sv8**	0093H	**3-24**	00B3H
L4P1	0074H	**tP8**	0094H	**3-25**	00B4H
L4i1	0075H	**tS8**	0095H	**3-26**	00B5H
L4d1	0076H	**1-2**	0096H	**3-27**	00B6H
L4Ar	0077H	**1-3**	0097H		
L4P2	0078H	**1-4**	0098H		
L4i2	0079H	**1-5**	0099H		
L4d2	007AH	**1-6**	009AH		
SEG	007BH	**1-7**	009BH		
TimE	007CH	**1-8**	009CH		
EndS	007DH	**1-9**	009DH		
Sv1	007EH	**1-10**	009EH		
tP1	007FH	**1-11**	009FH		

🕮 CRC Generation

The Cyclical Redundancy Check (CRC) field is two bytes, containing a 16–bit binary value. The CRC value is calculated by the transmitting device, which appends the CRC to the message. The receiving device recalculates a CRC during receipt of the message, and compares the calculated value to the actual value it received in the CRC field. If the two values are not equal, an error results.

The CRC is started by first preloading a 16–bit register to all 1's. Then a process begins of applying successive 8–bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC. Start and stop bits, and the parity bit, do not apply to the CRC.

During generation of the CRC, each 8–bit character is exclusive ORed with the register contents. Then the result is shifted in the direction of the least significant bit (LSB), with a zero filled into the most significant bit (MSB) position. The LSB is extracted and examined. If the LSB was a 1, the register is then exclusive ORed with a preset, fixed value. If the LSB was a 0, no exclusive OR takes place.

This process is repeated until eight shifts have been performed. After the last (eighth) shift, the next 8–bit character is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above. The final content of the register, after all the characters of the message have been applied, is the CRC value.

A procedure for generating a CRC is:

- ☞ Load a 16–bit register with FFFF hex (all 1's). Call this the CRC register.
- ☞ Exclusive OR the first 8–bit byte of the message with the low–order byte of the 16–bit CRC register, putting the result in the CRC register.
- ☞ Shift the CRC register one bit to the right (toward the LSB), zero–filling the MSB. Extract and examine the LSB.
- ☞ (If the LSB was 0): Repeat Step 3 (another shift).(If the LSB was 1): Exclusive OR the CRC register with the polynomial value A001 hex (1010 0000 0000 0001).
- ☞ Repeat Steps 3 and 4 until 8 shifts have been performed. When this is done, a complete 8–bit byte will have been processed.
- ☞ Repeat Steps 2 through 5 for the next 8–bit byte of the message. Continue doing this until all bytes have been processed.
- ☞ The final content of the CRC register is the CRC value.
- ☞ When the CRC is placed into the message, its upper and lower bytes must be swapped as described below.

Placing the CRC into the Message

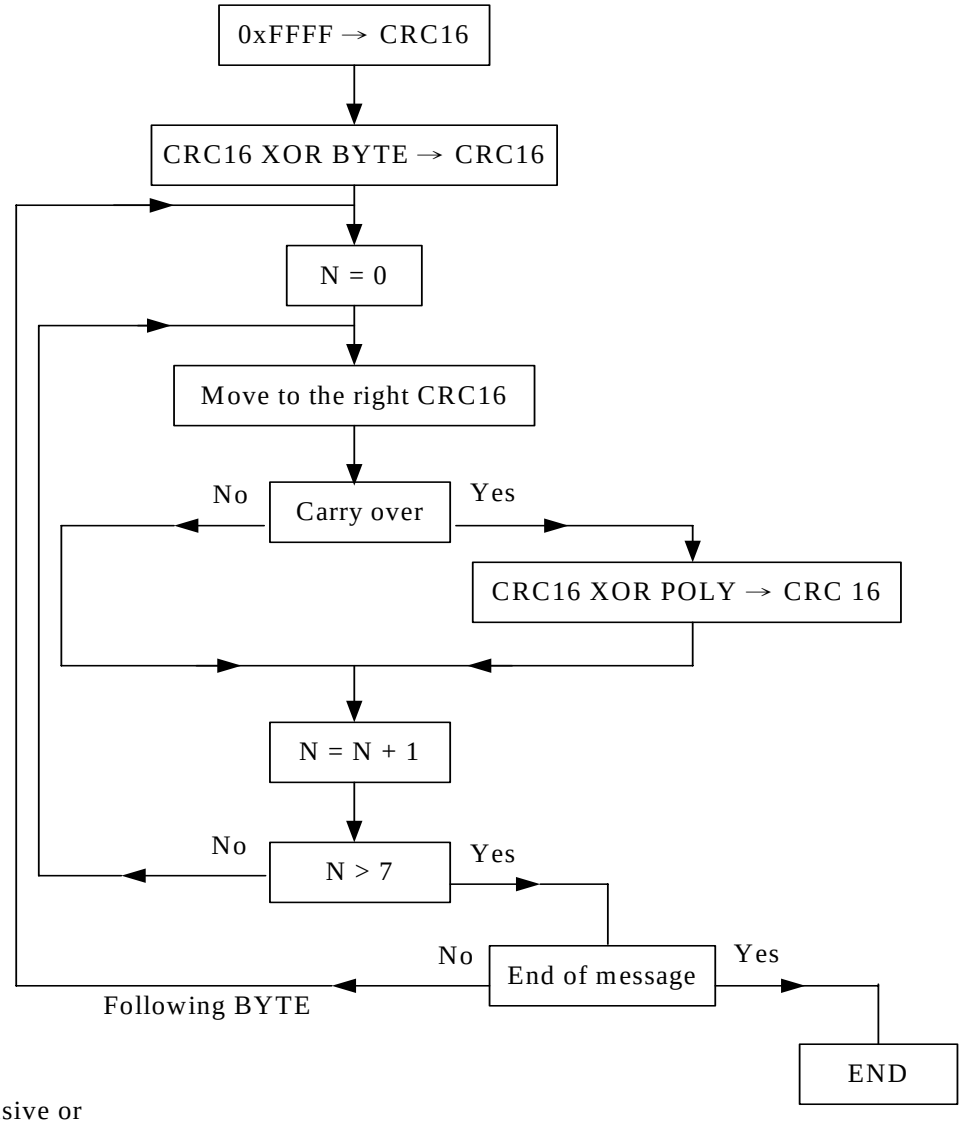
When the 16–bit CRC (two 8–bit bytes) is transmitted in the message, the low-order byte will be transmitted first, followed by the high-order byte.

For example, if the CRC value is 1241 hex (0001 0010 0100 0001):

Addr	Func	Data Count	Data	Data	Data	Data	CRC_{Lo}	CRC_{Hi}
							41	12

CRC Byte Sequence

Calculation algorithm of the CRC 16



XOR = exclusive or
N = number of information bits
POLY = calculation polynomial of the CRC 16 = 1010 0000 0000 0001
(Generating polynomial = 1 + x2 + x 15 + x 16)
In the CRC 16, the 1st byte transmitted is the least significant one.

Example of CRC calculation (frame 02 07)

CRC register initialization		1111	1111	1111	1111	FFFF
XOR 1st character		0000	0000	0000	0010	0002
		1111	1111	1111	1101	FFFD
	Move 1	0111	1111	1111	1110|1	7FFE
		1010	0000	0000	0001	A001
Flag to 1, XOR polynomial		1101	1111	1111	1111	DFFF
	Move 2	0110	1111	1111	1111|1	6FFF
Flag to 1, XOR polynomial		1010	0000	0000	0001	A001
		1100	1111	1111	1110	CFFE
	Move 3	0110	0111	1111	1110|0	67FF
	Move 4	0011	0011	1111	1111|1	33FF
		1010	0000	0000	0001	A001
		1001	0011	1111	1110	93FE
	Move 5	0100	1001	1111	1111|0	49FF
	Move 6	0010	0100	1111	1111|1	24FF
		1010	0000	0000	0001	A001
		1000	0100	1111	1110	84FE
	Move 7	0100	0010	0111	1111|0	427F
	Move 8	0010	0001	0011	1111|1	213F
		1010	0000	0000	0001	A001
		1000	0001	0011	1110	813E
		1000	0001	0011	1110	813E
		0000	0000	0000	0111	0007
XOR 2nd character		1000	0001	0011	1001	8139
	Move 1	0100	0000	1001	1100|1	409C
		1010	0000	0000	0001	A001
		1110	0000	1001	1101	E09D
	Move 2	0111	0000	0100	1110|1	704E
		1010	0000	0000	0001	A001
		1101	0000	0100	1111	D04F
	Move 3	0110	1000	0010	0111|1	6827
		1010	0000	0000	0001	A001
		1100	1000	0010	0110	C826
	Move 4	0110	0100	0001	0011|0	6413
	Move 5	0011	0010	0000	1001|1	3209
		1010	0000	0000	0001	A001
		1001	0010	0000	1000	9208
	Move 6	0100	1001	0000	0100|0	4904
	Move 7	0010	0100	1000	0010|0	2842
	Move 8	0001	0010	0100	0001|0	1241

Most significant least significant The CRC 16 of the frame is then: 4112

An example of a C language function performing CRC generation is shown on the following pages. All of the possible CRC values are preloaded into two arrays, which are simply indexed as the function increments through the message buffer.

One array contains all of the 256 possible CRC values for the high byte of the 16–bit CRC field, and the other array contains all of the values for the low byte.

Indexing the CRC in this way provides faster execution than would be achieved by calculating a new CRC value with each new character from the message buffer.

☞ **Note** This function performs the swapping of the high/low CRC bytes internally. The bytes are already swapped in the CRC value that is returned from the function.

Therefore the CRC value returned from the function can be directly placed into the message for transmission.

The function takes two arguments:

unsigned char *puchMsg ;	A pointer to the message buffer containing binary data to be used for generating the CRC
unsigned short usDataLen ;	The quantity of bytes in the message buffer.

The function returns the CRC as a type unsigned short.

CRC Generation Function Using C Language

```
unsigned short CRC16(puchMsg, usDataLen)
unsigned char *puchMsg ; /* message to calculate CRC upon */
unsigned short usDataLen ; /* quantity of bytes in message */
{
  unsigned char uchCRCHi = 0xFF ; /* high byte of CRC initialized */
  unsigned char uchCRCLo = 0xFF ; /* low byte of CRC initialized */
  unsigned uIndex ; /* will index into CRC lookup table */
  while (usDataLen—) /* pass through message buffer */
    {
    uIndex = uchCRCHi ^ *puchMsgg++ ; /* calculate the CRC */
    uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex} ;
    uchCRCLo = auchCRCLo[uIndex] ;
    }
  return (uchCRCHi << 8 | uchCRCLo) ;
}
```

High-Order Byte Table

/* Table of CRC values for high–order byte */

```
static 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
};
```

Low-Order Byte Table

/* Table of CRC values for low–order byte */

```
static 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
};
```

Following is another C language CRC program example

```
unsigned int reg_crc = 0xffff; i = 0;
while (length--)
{
   reg_crc ^= RTUData[i];
   i ++;
   for (j = 0; j < 8; j++)
   {
      if (reg_crc & 0x01)
            reg_crc = (reg_crc >> 1) ^ 0xA001;
      else
            reg_crc = reg_crc >> 1;
   }
}
return(reg_crc);
```

CRC Generation Function Using BASIC Language

```
Public Sub CRC16(CRChigh As Byte, CRClow As Byte, ByteCount As Byte, CrcData() As Byte)
Dim CRC As Integer
      CRC = &HFFFF
      For i = 0 To ByteCount - 1
            CRC = CRC Xor CrcData(i)
            For j = 1 To 8
                  CT = CRC And &H1
                  If CRC < 0 Then CH = 1 Else CH = 0: GoTo Label1
                  CRC = CRC And &H7FFF
Label1:
                  CRC = Int(CRC / 2)
                  If CH = 1 Then CRC = CRC Or &H4000
                  If CT = 1 Then CRC = CRC Xor &HA001
            Next j
      Next i
      CRChigh = CRC And &HFF
      CRClow = Int(CRC / &H100) And &HFF
End Sub
```