SPECIFICATION

HCRH-Modbus-Ka-HQ, HCRH-Modbus-V-Ka-HQ

„Humidity and temperature transmitter RH&T ► MODBUS RTU”

Prepared by:

HOTCOLD s.c.

2017-11-29

1. Introduction

The subject of the present paper is a functionality characteristic of a *humidity and temperature transmitter based on Sensirion SHT series sensor, with an interface RS-485 and a built-in MODBUS RTU protocol, with an optional RH analogue output in 0-10V standard*

NOTE: Read the text in this paper before running the module.

1.1. Device Features

- **relative humidity** measurement
- optional analogue voltage output 0-10 [V] (in the range 0-100% RH)
- **temperature** measurement
- calculation of the dew point
- LED operation indication
- RS-485 serial interface (reading of measurement values, configuration of work parameters)
    - MODBUS RTU protocol
    - communication in HALF DUPLEX mode
    - hardware configurable address (1-127)
    - hardware configurable baud rate (9600, 19200, 38400, 57600, 115200)

1.2. Device Characteristics

The basic function of the HCRH-Modbus-Ka transmitter is the determination of instantaneous temperature-compensated relative humidity values and instantaneous temperature values in parallel. The values measured by integrated SHT sensor by Sensirion, then recalculated and averaged in microcontroller, are available in its memory (in holding registers), in accordance with MODBUS standard. Registers are read by means of MODBUS protocol functions sent via RS-485 serial interface. Signalling the lack/error of a sensor is realized by means of a status register. Optional relative humidity value is also presented in an analogue form on a voltage output in standard 0-10 [V].

2. Technical Data

2.1. Transmitter General Parameters

Power supply	
- DC voltage	DC 24V (20...30V)
- AC voltage	AC 24V (20...27.6V)
Current consumption	
- typical [1]	<13.0 mA
- maximum [2]	<23.0 mA
LED signaling	0.2 Hz
Installation connector	mount screw 5.00mm (≤ 2.5mm^2)
Dimensions	80 x 80 x 25 (L x H x W)
Weight / Degree of protection	65 g / IP65
Mounting [3]	wall-mounted
Working environment	dust-free, air, neutral gases
Working temperature	-40°C ÷ 80°C

1) Average current consumption of the device in the following conditions: transmission 10 requests per second; transmission baud rate 9600 b/s; simultaneous reading of 3 registers; bus terminating resistors 2 x 120Ω; power supply 24V DC, voltage output loaded with 10k resistance;

2) Maximum current consumption of the device in conditions as in item 1) + voltage output with 1k resistance;

3) The device should be installed by qualified personnel; Vertical orientation according to the UP-DOWN marking;

2.2. Humidity Measurement Parameters

Sensor type	SHT series
Measuring range	0 ÷ 100 %RH
Resolution	12 bits (0.04 %RH)
Accuracy for T=25°C	
- in the range 20 ÷ 80 %RH	±1 %RH
- within the remaining range	±(2 ÷ 3) %RH
Hysteresis	±1 %RH
Stability of measurement	<0,5%RH/year
Sampling frequency	1 Hz
Response time [1]	8 s

1) The condition to obtain the given response times is an air flow of > 1m/s at 25°C; the indicated response time is equal to one time constant corresponding to 63% of the set value;

2.3. Temperature Measurement Parameters

Sensor type	SHT series
Resolution	14 bits (0.01 °C)
Measuring range	-40°C ÷ 123.8°C
Accuracy	
- within the range 10 ÷ 50°C	±0.1°C
- within the range 0 ÷ 60°C	±0.4°C
- within the remaining range	±(0.4 ÷ 1.0) °C
Sampling frequency	1 Hz

Response time [1)]	5 ÷ 30 s

1) The condition to obtain the given response times is an air flow of > 1m/s; the indicated response time is equal to one time constant corresponding to 63% of the set value;

2.4. Analog output parameters (option)

Output type	voltage
Output range	10 V
Resolution	12 bits (5 mV)
Capacitance	R_L > 1 kΩ
Refresh rate	1 Hz

2.5. Serial Interface Parameters

Physical layer	RS-485
Communication protocol	MODBUS RTU
Connection configurations [1)]	HALF DUPLEX
Transmission baud rates	9600 / 19200 / 38400 / 57600 / 115200 b/s

1) HALF DUPLEX - bi-directional communication with one pair of wires;

3. Installation

3.1. Safety

- The device should be installed by qualified personnel!
- All connections must be made in accordance with the wiring diagrams provided in this specification!
- Check all electrical connections before commissioning!

3.2. Device Design

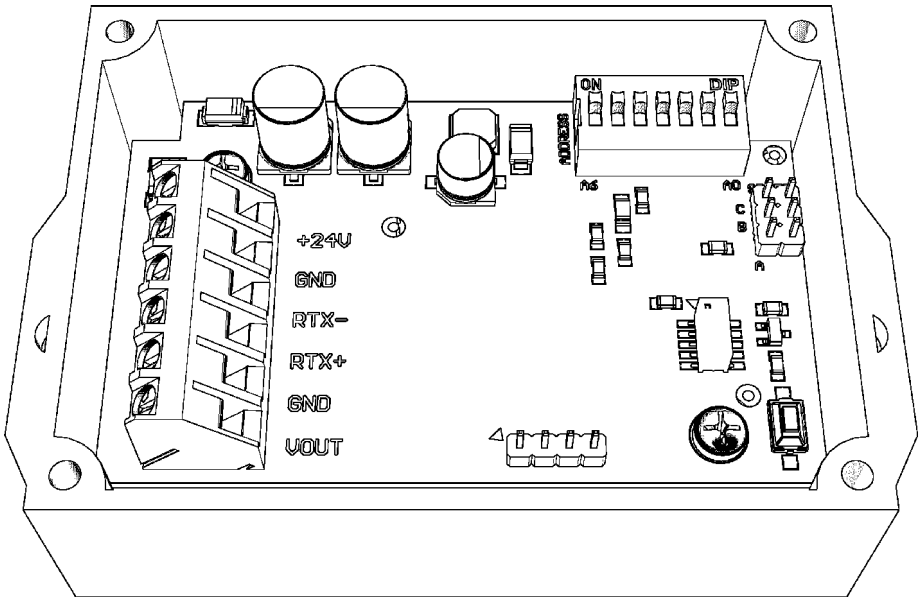

Figure 1. View of the printed circuit of the wall-mounted version of the transmitter.

3.3. Output Description

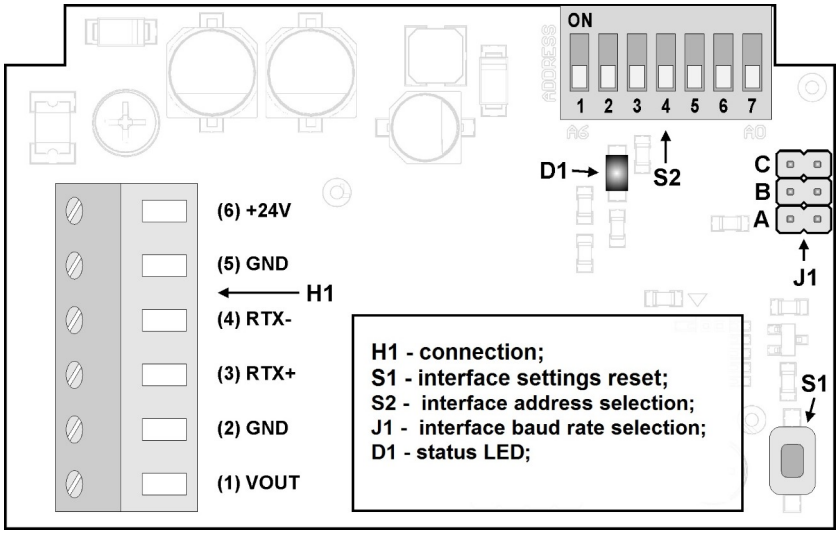

Figure 2. Description of outputs of the wall-mounted version of the transmitter.

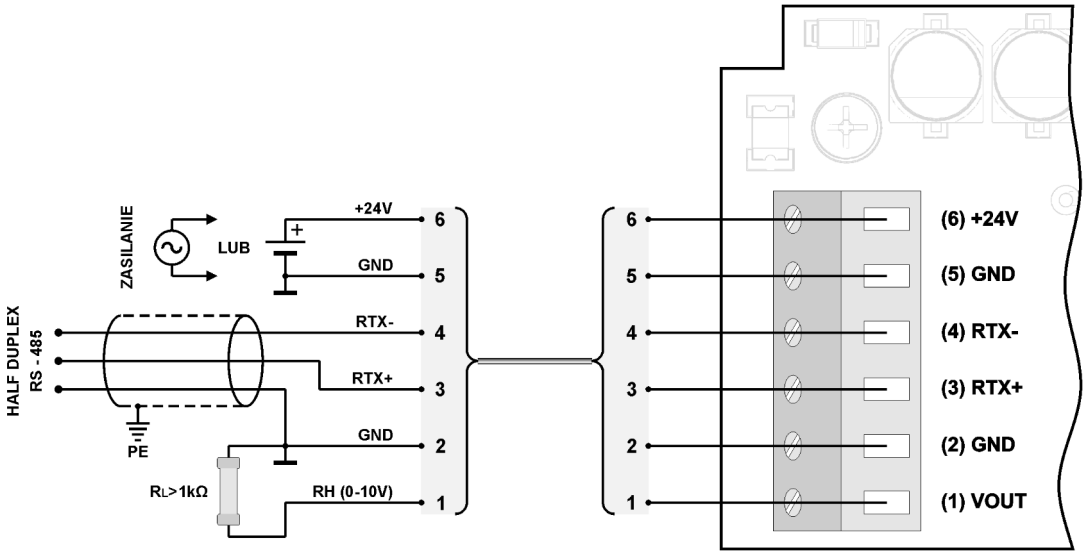

Figure 3. Wiring diagram of the transmitter in the wall-mounted version.

3.4. Address Configuration

The device is equipped with a DIP-SWITCH (5, 6, or 7 positions) for hardware address setting (from "1" to max "127"). Setting the address "0" on the switch will use the address stored in the device via MODBUS protocol (default "1").

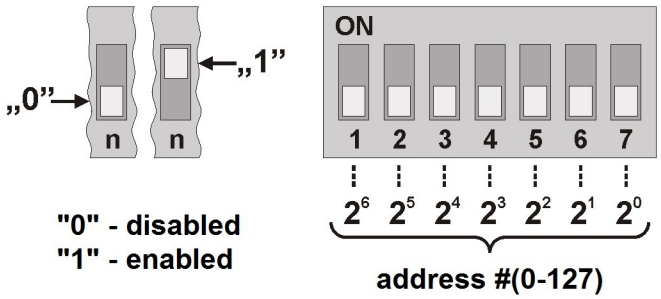

Figure 4. Addressing of the transmitter.

3.5. Baud Rate Configuration

The device is equipped with a 3 jumper circuit for hardware baud rate setting of the RS-485 interface (as shown in the table below). No jumpers will use the baud rate value stored in the device via MODBUS protocol (default "9600 b/s").

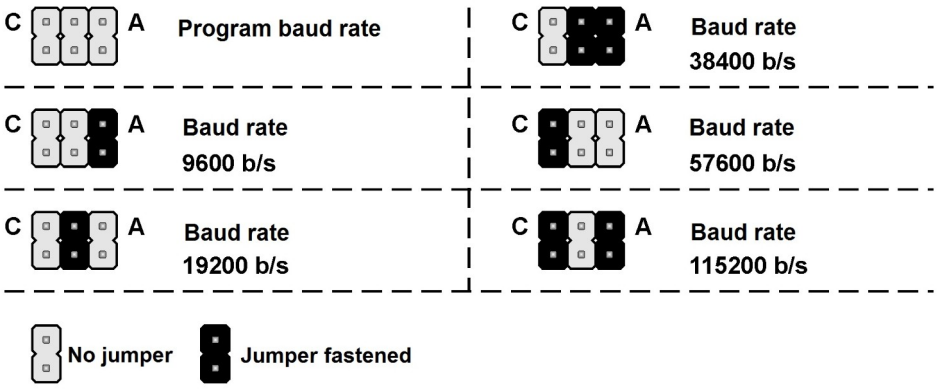

Figure 5. RS-485 interface baud rate configuration.

3.6. Restoring Factory Settings

The factory restoring function applies only to RS-485 interface transmission parameters (including address and baud rate). To restore the settings, press and hold the S1 button for about 2 seconds (protection against accidental pressing). When D1 LED flashes, release the button. The device will operate with the new settings automatically.

3.7. Guidelines

- When operating in a high noise environment, use shielded cables.
- Connect the cable shield to the nearest PE point on the power supply side.

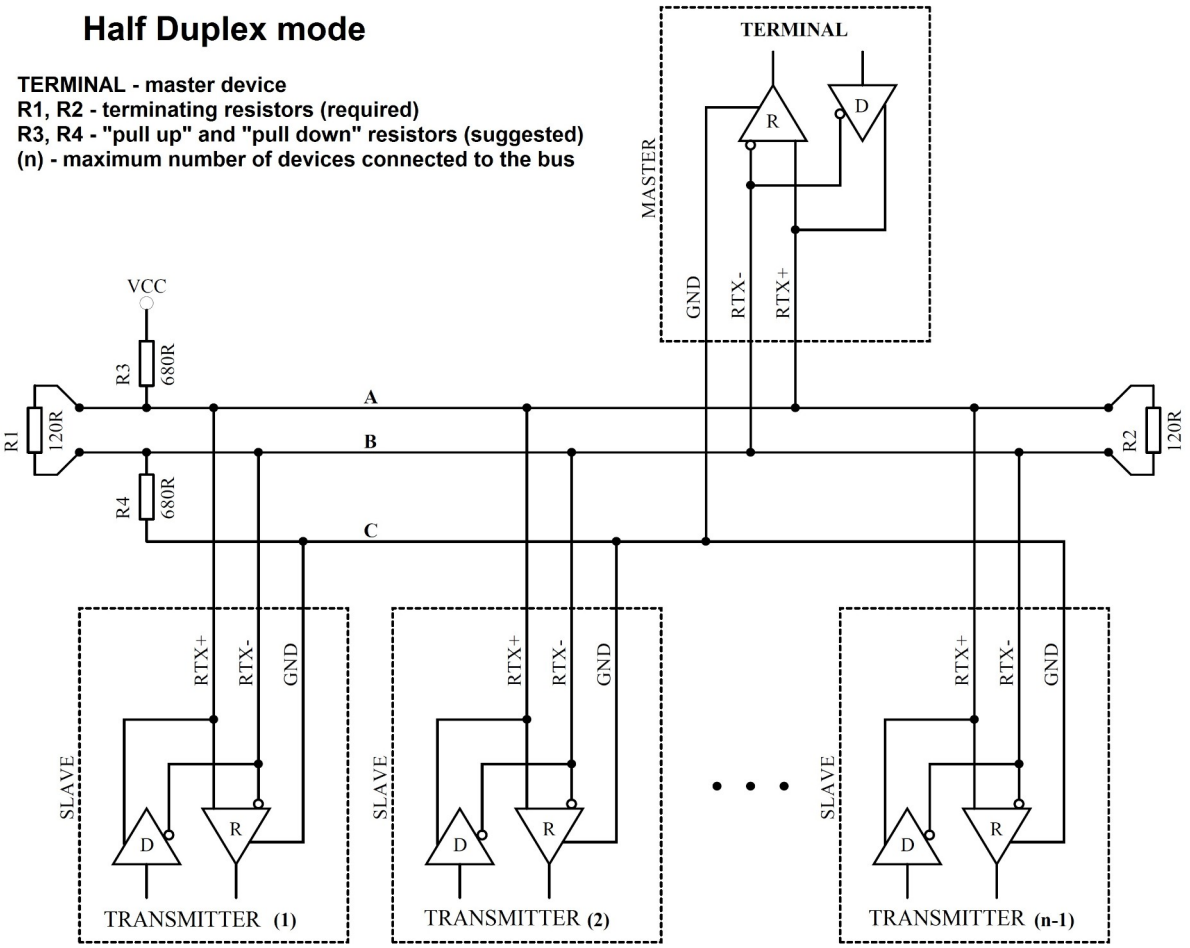

Figure 6. Connection of the transmitter to the RS-485 bus operating in HALF DUPLEX mode.

4. MODBUS Protocol

4.1. Map of Registries

Registers table:

Register No.	Values	Description
1	1 – 1000	Relative humidity (1 = 0.1%; 1000 = 100%)
2	-4000 – 12380	Temperature [ºC] (1 = 0.01 ºC) with a sign
3	-4000 – 12380	Dew point [ºC] (1 = 0.01 ºC) with a sign
4	1234	Password register
5	1 / 2 / 3	Command register
6	according to command table	Parameter register
7	0-65535	Counter of valid frames
8	0-65535	Exception counter
9	0-65535	Counter of erroneous CRC
10	0-65535	Counter of erroneous bytes
11	-	not used
12	0 / 1 / 2	Status register (0: “NO SENSOR”, 1: “SENSOR OK”, 2: “ERROR” **(*)**
13	1000 (0x03E8)	Test value - for verifying correctness of registers reading

(*) "NO SENSOR" - lack of sensor; "SENSOR OK" - correct sensor operation; "ERROR" - sensor operation error;

Command table:

Command No.	Function	Parameters
1	Set device address	1 - 247 (1 is the default value)
2	Set baud rate	96 - 9600 b/s (1 is the default value) 192 – 19200 b/s 384 – 38400 b/s 576 – 57600 b/s 1152 – 115200 b/s
3	Set parity bits	0 – NO PARITY; no parity bit 1 - EVEN PARITY; (default value) 2 – ODD PARITY,
4	Set the bits	1 - 1 x STOP; 1 stop bit (default value) 2 - 2 x STOP; 2 stop bits
5	Device reset	1 - software device reset

Notes:

- Specifying an incorrect or out-of-range parameter value results in 0xEEEE value being written to the command register.
- Each command invocation must be accompanied by entering the password (1234 decimal).

- Each command invocation through individual entries in registers must be followed by entering the password.

4.2. Protocol functions

The following MODBUS standard functions are implemented in the transmitter:

CODE	*SIGNIFICANCE*
03 (0x03)	*Reading N x 16-bit registers*
16 (0x10)	*Write N x 16-bit registers*

4.2.1. Reading the contents of the output registers group (0x03)

Request format:

Description	*Size*	*Values*
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x03***
Data block address	*2 byte*	*0x0000 – 0xFFFF*
Number of registers (N)	*2 byte*	*1 – 125 (0x7D)*
CRC checksum	*2 byte*	*by calculation*

Response format:

Description	*Size*	*Values*
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x03***
Byte counter	*1 byte*	*2 x N*
Register values	*N x 2 bytes*	*by map of registries*
CRC checksum	*2 byte*	*by calculation*

Error format:

Description	*Size*	*Values*
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x83***
Error code	*1 byte*	*0x01 / 0x02 / 0x03 / 0x04*
CRC checksum	*2 byte*	*by calculation*

4.2.2. Writing to output register group (0x10)

Request format:

Description	*Size*	*Values*
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x10***
Data block address	*2 byte*	*0x0000 – 0xFFFF*
Number of registers (N)	*2 byte*	*1 – 123 (0x7B)*
Byte counter	*1 byte*	*2 x N*
Values	*N x 2 bytes*	*of the user*
CRC checksum	*2 byte*	*by calculation*

Response format:

Description	*Size*	*Values*
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x10***
Data block address	*2 byte*	*0x0000 – 0xFFFF*
Number of registers (N)	*2 byte*	*1 – 123 (0x7B)*
CRC checksum	*2 byte*	*by calculation*

Error format:

Description	*Size*	*Values*
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x90***
Error code	*1 byte*	*0x01 / 0x02 / 0x03 / 0x04*
CRC checksum	*2 byte*	*by calculation*

4.3. Data format

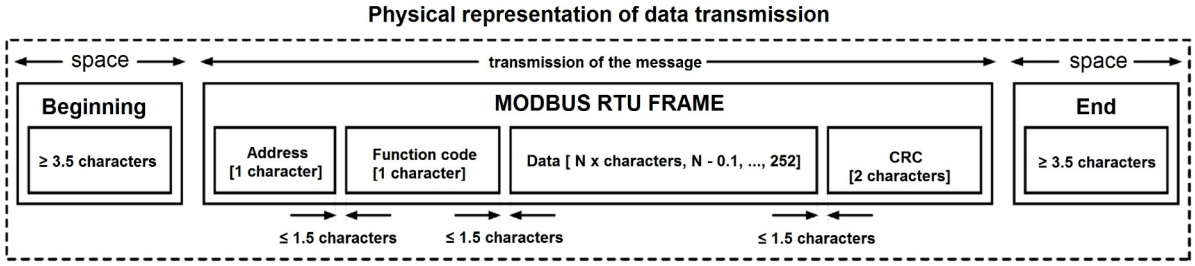

Figure 7. MODBUS RTU standard data transfer implemented in the transmitter.

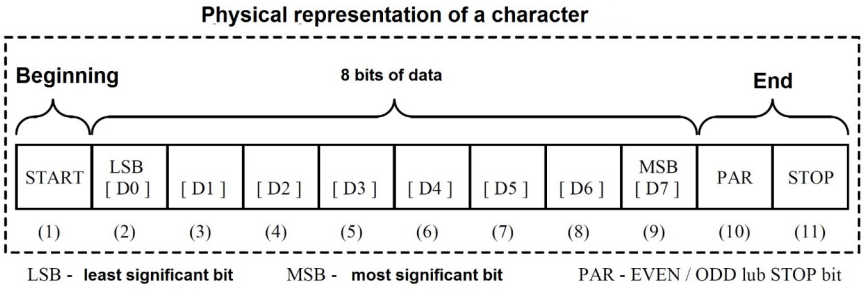

Figure 8. MODBUS RTU standard character format implemented in the transmitter.

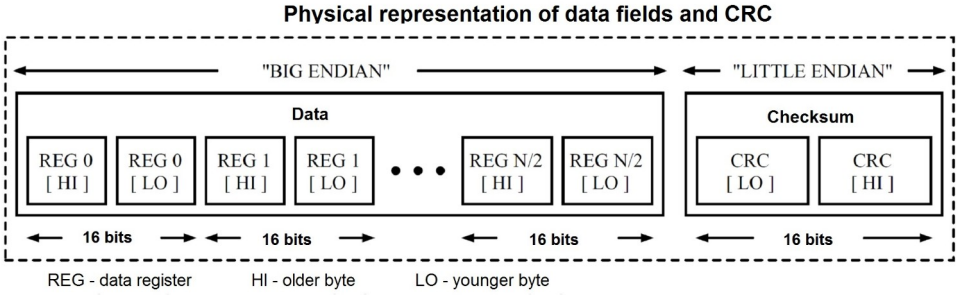

Figure 9. Format of data fields and CRC in MODBUS RTU standard as implemented in the transmitter.

4.4. CRC checksum

According to the MODBUS standard, a polynomial was used to calculate the CRC checksum:
X16 + X15 + X2 + 1.

4.4.1. Bitwise CRC calculation algorithm:

Procedure for determining the CRC checksum using the bitwise method:

a) loading the value 0xFFFF into the 16-bit CRC register;
b) taking the first byte from the data block and performing an EX-OR operation with the younger byte of the CRC register, placing the result in the register;
c) shifting the CRC register contents to the right by one bit towards the least significant bit (LSB), clearing the most significant bit (MSB);
d) checking the state of the youngest bit (LSB) in the CRC register, if its state is 0, then return to the point c; if 1, then the EX-OR operation of the CRC register with the constant 0xA001 is performed;
e) repeat points c and d up to eight times, which corresponds to processing the entire byte;
f) repetition of the sequence b, c, d, e for the next byte of the message, continuation of this process until all bytes of the message have been processed;
g) the content of the CRC register after performing the mentioned operations is the wanted value of the CRC checksum;
h) adding CRC checksum to the MODBUS RTU frame must be preceded by swapping places of the older and the younger byte of the CRC register.

4.4.2. Tabular CRC calculation algorithm:

Example of an implementation of the CRC checksum determination procedure using the array method:

```
/* The function returns the CRC as a unsigned short type */
unsigned short CRC16 ( puchMsg, usDataLen )
/* message to calculate CRC upon */
unsigned char *puchMsg ;
/* quantity of bytes in message */
unsigned short usDataLen ;

{
        /* high byte of CRC initialized */
        unsigned char uchCRCHi = 0xFF ;
        /* low byte of CRC initialized */
        unsigned char uchCRCLo = 0xFF ;
        /* will index into CRC lookup table */
        unsigned uIndex ;

        /* pass through message buffer */
        while (usDataLen--)
        {
                /* calculate the CRC */
                uIndex = uchCRCLo ^ *puchMsg++ ;
                uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex] ;
                uchCRCHi = auchCRCLo[uIndex] ;
        }
return (uchCRCHi << 8 | uchCRCLo) ;
}
```

```
/* 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
} ;

/* 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
};
```