SPECIFICATION

Air Velocity Transmitter
HCV-V, HCV-M, HCV-M-V

Compiled by:

HOTCOLD s.c.

19 April 2023

1. Introduction

The subject of this document is the characterisation of the functionality of Air Velocity Transmitter based on a Honeywell ABP series differential pressure sensor, with RS- 485 interface with built-in MODBUS RTU protocol and 0 - 10V analogue output.

NOTE: Please, read this document before starting to use your unit.

1.1. Devicefunctions

- air velocity measurement
- measurement range configuration
- LED indication of correct operation
- analogue output in the 0-10V standard
- RS-485 serial interface (reading measured values, configuration of operating parameters)
  - MODBUS RTU protocol
  - HALF DUPLEX communication
  - hardware configurable address (1-31)

1.2. Devicecharacteristics

Primary function of the HCV-M, HCV-V, HCV-M-V transmitter is to measure the air velocity in the duct. The values measured via the integrated Honeywell ABP series sensor, then recalculated and averaged in the microcontroller, are available in its memory (in HOLDING REGISTERS) according to the MODBUS standard. Registers are read using MODBUS protocol functions transmitted via the serial RS-485 interface. The registers also contain information about the currently set (configurable) measuring range, the time constant (also configurable) and the percentage of air velocity related to the measuring range. Indication of missing sensor, overrange states, transmitter occupancy in the event of an offset calibration being performed are also realised by status registers.

1.3. Air velocity transmitter versions

HCV-V Air Velocity transmitter with 0-10V output (6 measurement ranges)
HCV-M Air Velocity transmitter with RS485 output with MODBUS RTU protocol
HCV-M-V Air Velocity Transmitter with RS485 output with MODBUS RTU protocol and 0-10V analogue output in parallel.

All versions of the transmitter come with 200, 300 and 400 mm measuring lances.

2. Technical data

2.1. General parameters

Power supply	
- DC	24V (20...30V)
- AC	24V (21,5...26,5V)
Current consumption	
- minimum [1]	9.0mA
- typical	11.0mA
- maximum [3]	22.0mA
LED signalling	0.2Hz
Installation connection	screw in 5.00mm grid (≤ 2.5mm^2)
Dimensions	130 x 70 x 40 (L x H x W)
Weight	approx. 150 g
Mounting [4)]	in duct
Degree of protection	IP65
Operational environment	dust-free, air, neutral gases
Operating temperature	0°C ÷ 50°C
Storage conditions	
- temperature	-40°C ÷ 85°C
- relative humidity	20 ÷ 60 %RH

1. Average current consumption of the device under conditions: no transmission; analogue output unloaded; 24V DC power supply;
2. Average current consumption of the device under the following conditions: transmission of 10 queries per second; transmission speed 9600 b/s; simultaneous reading of 20 registers; bus terminating resistors 2 x 120Ω; analogue output set to 10V and loaded with 10kΩ resistance; 24V DC power supply;
3. Maximum instantaneous current consumption under the following conditions: analogue output loaded with 1kΩ resistance; indicating LED permanently on; other conditions as in 2);
4. Installation of the unit must be carried out by qualified personnel;

2.2. Air velocity measurement parameters

Sensor type	ABP
Max measuring range	40m/s
Resolution	12 bits
Accuracy in range 0 ÷ 50°C :	
for <5 m/s	≤ ± 5%
for <10 m/s	≤ ± 2%
for <20 m/s	≤ ± 1%
for <30 m/s	≤ ± 1%
for <40 m/s	≤ ± 1%
Sampling frequency	100Hz
Response time [1]	0.8s / 4s [2]

1) response time given is equal to one-time constant corresponding to 63% of the set value;
2) default value is shorter response time;

2.3. Serial interfaceparameters

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

1. HALF DUPLEX - two-way communication over one pair of wires;

3. Installation

3.1. Safety

- Installation of the unit must be carried out by qualified personnel!
- All connections must be made in accordance with the wiring diagrams shown in this specification!
- Check all electrical connections before commissioning!

3.2. Design

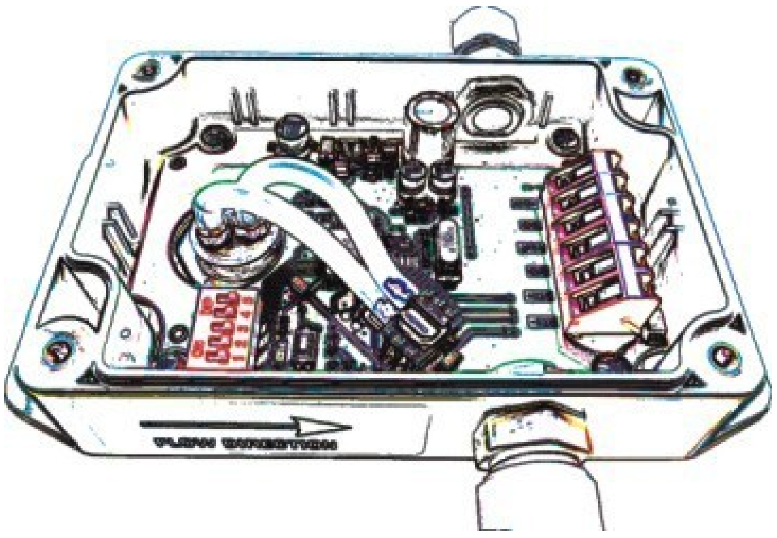


Figure 1. Overview of the device.

3.3. Terminals

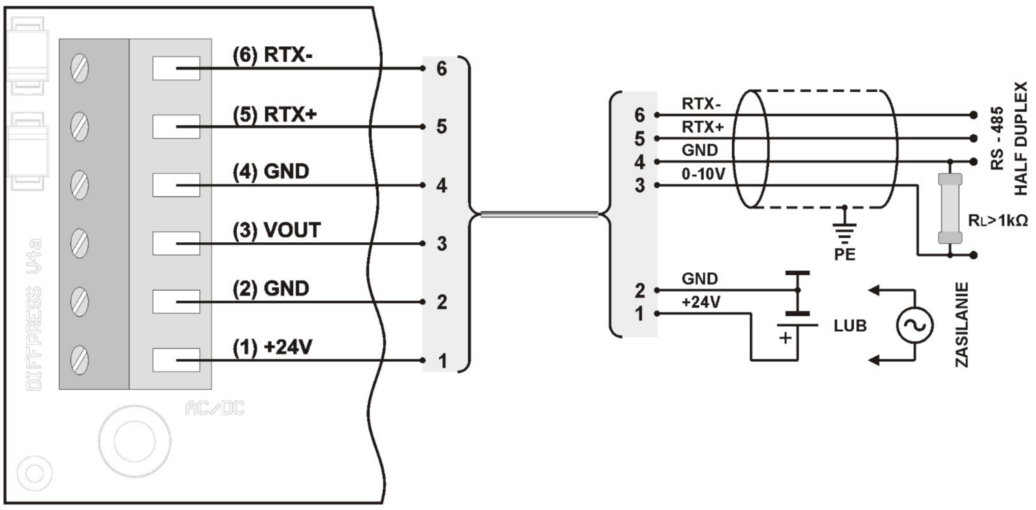


Figure 2. Transmitter wiring diagram.

3.4. Address configuration

The device is equipped with a 5-position switch for hardware address setting (from "1" to "31"). Setting the address "0" on the switch will use the address stored in the device via the MODBUS protocol (default "1").

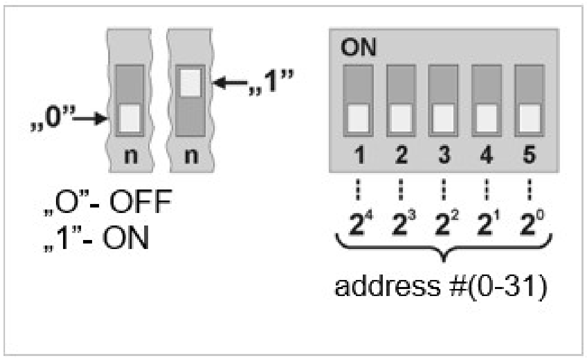


Figure 2. Transmitter addressing.

3.5. Offset reset

Reset by command:

Before calibrating (zeroing) the offset, the output range must first be set, and the measuring lance must be placed at equal pressure (it can be removed from the duct). Resetting process takes place when the offset calibration command is sent. Calibration takes approximately 7s. Once correctly calibrated, the unit should indicate zero air speed.

Reset by push button:

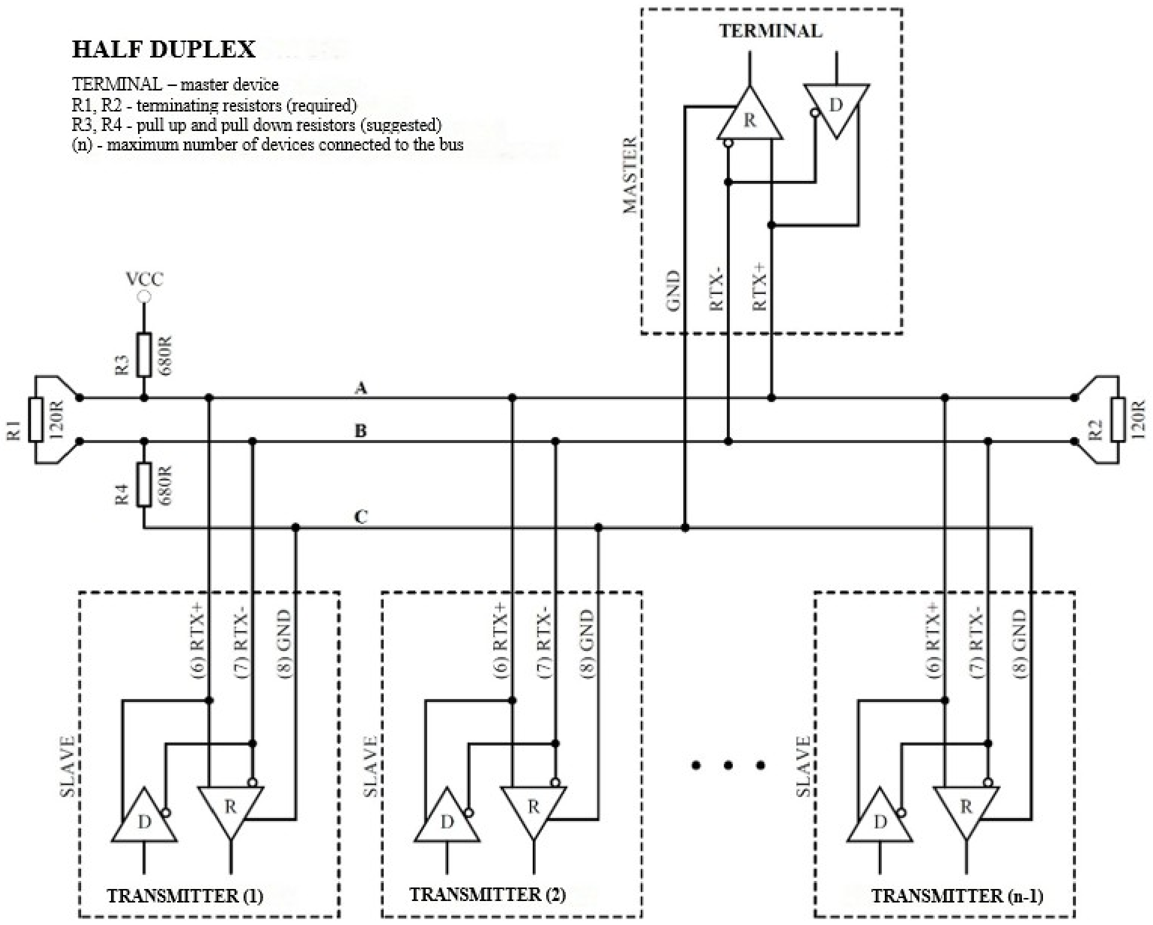
Before calibrating (zeroing) the offset, the measuring lance must first be placed at equal pressure (it can be removed from the duct). To trigger the zeroing process, press and hold the S1 button for approximately 3 seconds until the D1 LED starts blinking. Time of calibration is counted from the moment the button is released and it is approximately 7 seconds. The calibration process is signalled by blinking D1 LED. Once correctly calibrated, the unit should indicate zero air speed.

3.6. Restore factory settings

Restoring factory settings function only applies to the transmission parameters of the RS-485 interface (including the address). To restore the settings, press and hold button S1 for approximately 10 seconds (the first 3 seconds after pressing, LED D1 lights up, the next 7 seconds it is blinking). Release the button when the D1 LED starts to be steadily on. The device will start operating with the new settings automatically.

3.7. Guidelines

- Use shielded cables when operating in a high interference environment.



- Connect the cable shield to the nearest PE point on the power supply side.

Figure 3. Method of connecting the transmitter to the RS-485 bus operating in HALF DUPLEX mode.

4. MODBUS protocol

4.1. Map of registers

Table of registers:

Register no.	Values	Desc ripti on
1	**0** – 9999	**Flow rate** (limited by measuring range) [**m/s**] (**100** = 1 **m/s**) with sign
2	0 – 1000	**Flow velocity** related to the range (1 = 0,1%; 1000 = 100%)
3	0 / 1 / 2 / 3	Status register (0: “SENSOR OK”, 1: “UNDERLOAD”, 2: “OVERLOAD”, 3: “NO SENSOR”) (*)
4	1234	Password register
5	1 / 2 / 3	Command register
6	according to command table	Parameter register
7	0 / 1	Time constant TAU (0: 0.8s; 1: 4.0s)
8	0 / 1 / 2 / 3 / 4	Measuring range (according to measuring ranges table)
9	-999 – 9999	Transmitter offset (informative) [Pa] (1 = 1 Pa) with sign
10	**0** – 9999	Lower measuring range value (informative) [Pa] (1 = 1 Pa) with sign
11	**0** – 9999	Upper measuring range value (informative) [Pa] (1 = 1 Pa) with sign
12	0 / 1	Offset calibration (zeroing) status (0: not active; 1: in progress)
13	0-65535	Count of valid frames
14	0-65535	Exception counter
15	0-65535	CRC error counter
16	0-65535	Error byte counter
17	0-65535	Incorrect address counter
18	-999 – 9999	Differential pressure (not limited to measuring range, for service purposes only) [Pa] (1 = 1 Pa) with sign

(*) "SENSOR OK" - correct sensor operation; "UNDERLOAD" - exceeding the range downward; "OVERLOAD" - exceeding the range upward; "NO SENSOR" - missing sensor;

Command table:

Cmd no.	Function	Parameters
1	Set device address	1 - 247 (1 - default value)
2	Set transmission speed	96 – 9600 b/s (default value) 192 – 19200 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 stop bits	1 - 1 x STOP; 1 stop bit (default value) 2 - 2 x STOP; 2 stop bits

5	Set time constant	0 – 0.8s;
		1 – 4.0s;

6	Set measuring range	ID according to measuring ranges table
7	Start calibration process	1 - start of offset calibration (zeroing)
8	Device reset	1 - unit software reset

Table of measurement ranges (applies to devices with 0...10 V output):

range		**Air velocity → voltage output**		
DIPSWITCH	**0 – 40 [m/s]**	**0V**	**5V**	**10V**
1	**0: 40**	**0 m/s = 0V**	**20 m/s = 5V**	**40 m/s = 10V**
2	**0: 30**	**0 m/s = 0V**	**15 m/s = 5V**	**30 m/s = 10V**
3	**0: 20**	**0 m/s = 0V**	**10 m/s = 5V**	**20 m/s = 10V**
4	**0: 10**	**0 m/s = 0V**	**5 m/s = 5V**	**10 m/s = 10V**
5	**0: 5**	**0 m/s = 0V**	**2.5 m/s = 5V**	**5 m/s = 10V**

Notes:

- Specifying an incorrect or out-of-range parameter value results in the value 0xEEEE being written to the command register.
- Each time a command is invoked, it must be accompanied by the entry of a password (1234 decimal).
- Invocation of the command via individual registry entries must be completed with the entry of a password.

4.2. Protocol functions

The following MODBUS standard functions are implemented in the HCV transmitter:

CODE	***MEANING***
03 (0x03)	*Reading N x 16-bit registers*
16 (0x10)	*Writing N x 16-bit registers*

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

Request format:

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

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*	*per map of registers*
CRC checksum	*2 bytes*	*as calculated*

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 bytes*	*as calculated*

4.2.2. Writing to the 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 bytes*	*0x0000 – 0xFFFF*
Number of registers (N)	*2 bytes*	*1 – 123 (0x7B)*
Byte counter	*1 byte*	*2 x N*

Values	*N x 2 bytes*	*user*
CRC checksum	*2 byte*	*as calculated*

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*	*as calculated*

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*	*as calculated*

4.3. Data format

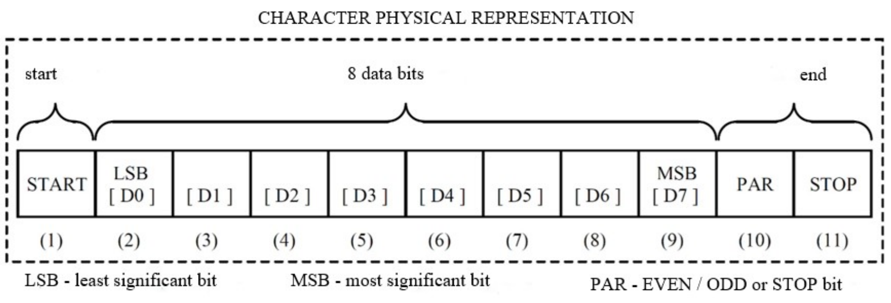


Figure 4. Data transfer in MODBUS RTU standard implemented in the transmitter.

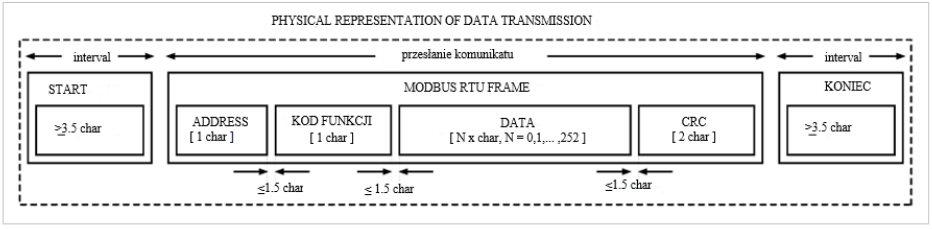


Figure 5. MODBUS RTU standard character format used in the transmitter.

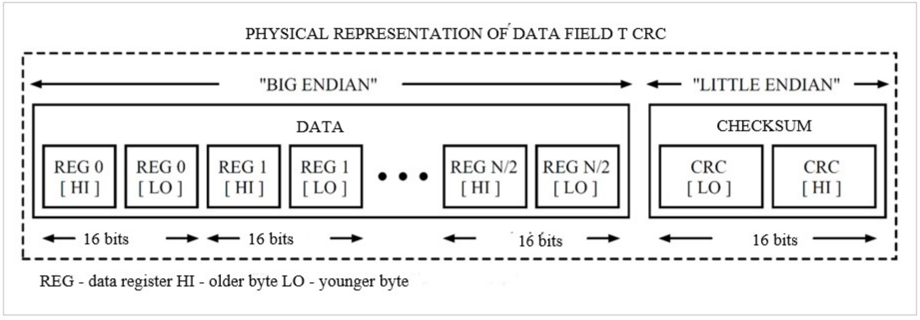


Figure 6. MODBUS RTU standard data and CRC field format used in the transmitter.

4.4. CRC checksum

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

4.4.1. Bitwise CRC counting algorithm:

Procedure for determining the CRC checksum using the bitwise method:

a. load the value 0xFFFF into the 16-bit CRC register;

b. take the first byte of the data block and performing an EX-OR operation with the younger byte of the CRC register, placing the result in the register;

c. shift the CRC register contents to the right by one bit towards the least significant bit (LSB), resetting the most significant bit (MSB);

d. check of the state of the youngest bit (LSB) in the CRC register, if its state equals 0, it is returned to point c, if 1, 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. repeat the sequence b, c, d, e for the next byte of the message, continue this process until all bytes of the message have been processed;

g. content of the CRC register after the mentioned operations is the sought CRC checksum value;

h. inserting a CRC checksum into a MODBUS RTU frame must be preceded by the swapping of places of the older and the younger byte in the CRC register.

4.4.2. Table-based CRC counting 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
};
```