SPECIFICATIONS

Duct Air Quality Transmitter - Humidity and Temperature, CO_2, Volatile Organic Compounds (VOC)

Prepared by:

Hotcold

01 March 2023

1. Introduction

This document describes performance of the universal Air Quality Transmitter - humidity and temperature, CO2 concentration, VOC content, with RS-485 interface with a built-in MODBUS RTU protocol.

CAUTION: Read this document before starting up your device.

1.1. Device functions

- carbon dioxide concentration measurement[1]
- temperature and humidity measurement[2]
- volatile organic compounds (VOC) measurement[3]
- OLED graphic display with 256x64 resolution (optional)
- RS-485 (reading of measured values, setting of operating parameters)
  - MODBUS RTU protocol
  - HALF DUPLEX communication
  - with configurable address by hardware (1-127) or software (1-247)
  - hardware or software configurable interface baud rate

1.2. Characteristics

Primary function of this device is to indicate instantaneous values of the measured air quality parameters (according to the version purchased). Versions available:

- CO2 measurement: **HCEM-02K**
- volatile organic compounds (VOC) measurement: **HCEM-03K**
- air temperature and humidity measurement: **HCEM-04K**
- CO2 and VOC measurement: **HCEM-05K**
- CO2, air humidity and temperature measurement: **HCEM-06K**
- VOC, humidity and temperature measurement: **HCEM-07K**
- CO2, VOC, humidity and temperature measurement: **HCEM-08K**

The values measured by the inside sensors are then calculated in the microcontroller and made available in its memory (in the HOLDING REGISTERS) in accordance with the MODBUS RTU standard, and they are optionally indicated on the unit display. Registers are read using MODBUS protocol functions transmitted via the serial RS-485 interface. No sensor or faulty sensor output is indicated in the status register.

[1] if CO2 transducer is installed

[2] if SHT4 transducer is installed

[3] if VOC transducer is installed

2. Technical data

2.1. General performance

Power supply	
- **DC**	24V (20...30V)
- **AC**	24V (20...27.6V)
Current consumption	
- **typical**[1)]	<15.0 mA
- **max**[2)]	<80.0 mA
LED signals	0.2 Hz
Installation	screw in 5.00mm grid (≤ 2.5mm$_2$)
connection	115 x 65 x 50 (L x H x W)
Dimensions	
Weight -	
Installation[3)]	-
Operating environment	Air
Operating temperature	-20°C ÷ 50°C

1) Average current consumption of the device in the following conditions: transmission rate of 10 requests per second; baud rate 9600 b/s; simultaneous reading of 3 registers; bus terminating resistors 2 x 120Ω; power supply 24V DC; mounted sensors CO2, VOC, RHT
2) current consumption during CO2 processing + conditions from 1)
3) Installation of the unit must be carried out by qualified personnel; Vertical orientation as marked UP, DOWN; Do not obstruct the device ventilation openings.

2.2. Carbon dioxide measurement parameters.

Sensor type	Infrared
Measuring range	400 ÷ 2000 ppm
Resolution	1 ppm
Accuracy 25°C ±5°C	± 50ppm+5% of reading
Sampling frequency	2s
Response time	T_{90} < 120s

2.3. Temperature and humidity measurement parameters

Sensor type	SHT series
Measuring range	-40 ÷ +125°C
Resolution	0.1°C
Accuracy	
- in range 0÷60°C	±0.2°C
- in range -40÷120°C	±0.6°C
Sampling frequency	1.5s
Measurement time	<1s
Humidity measuring range	0 ÷ 100%
Resolution	0.1%
Accuracy	
- in range 30÷70%	±1.8%
- in range 0÷100%	±3%
Sampling frequency	5s
Measurement time < 1s	

2.4. VOC measurement parameters

Sensor type	MEMS
Measuring range	0 ÷ 5 ppm
Resolution	1 ppm
Sampling frequency	2s
Response time	T_{90} < 60s

2.5. Serial interface parameters

Physical layer	RS-485
Communication protocol	MODBUS RTU
Connection configuration [1]	HALF DUPLEX
Baud rate	2400 4800 9600 14400 19200 28800 38400 57600 115200 b/s

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

3. Installation

3.1. Safety

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

3.2. Design

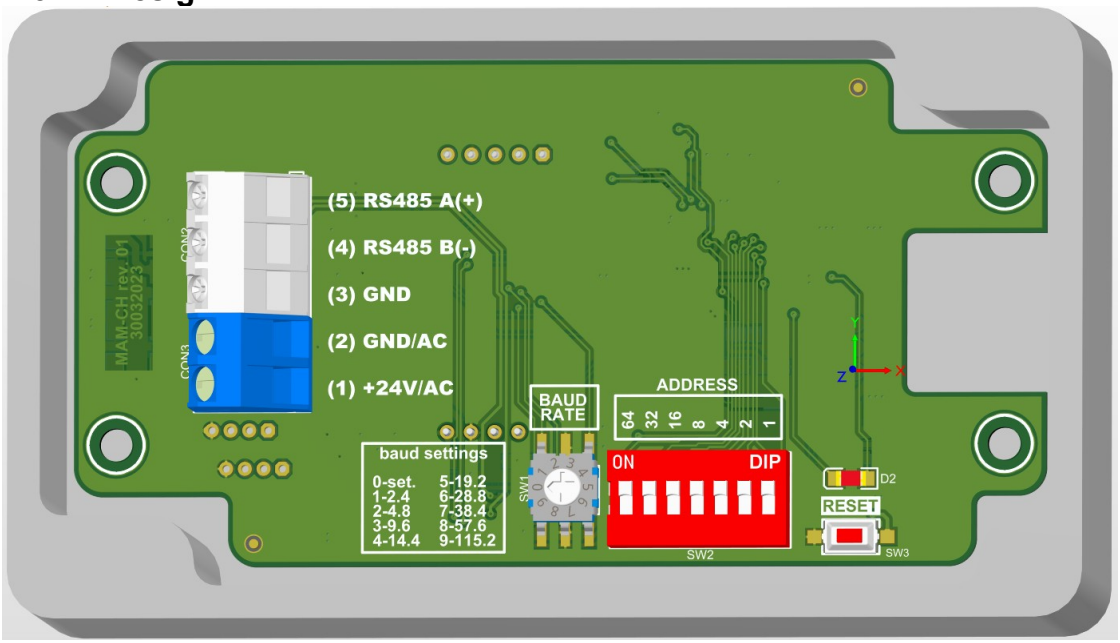


Figure 1. View of the transmitter PCB.

3.3. Connection diagram

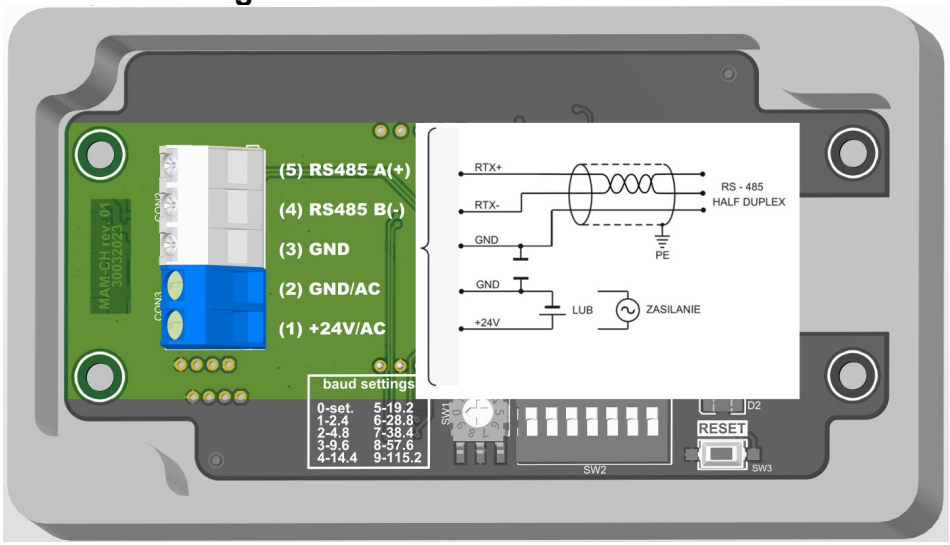


Figure 2. Transmitter connection diagram

3.4. Address setup

The device features a 7-position switch for hardware address setup (from '1' to '128'). When you set the address "0" on the switch, the unit will use the address stored in it via the MODBUSprotocol(default"1").

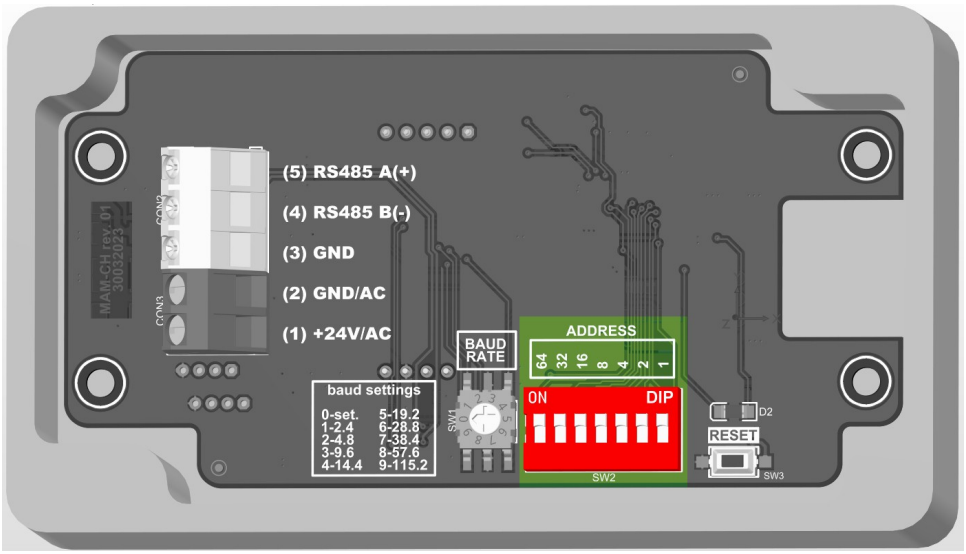
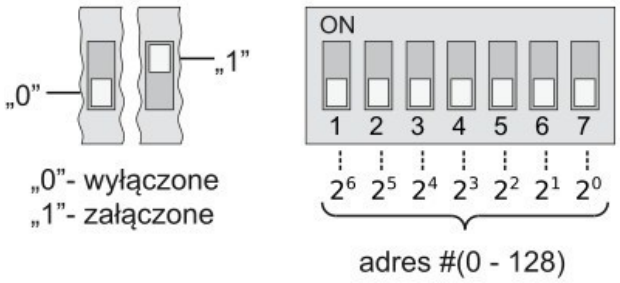


Figure 3. Transmitter addressing.



3.5. Baud rate setup

The unit is equipped with a rotary switch to set the hardware speed of the RS-485 interface (according to the table on the cover). When you set "0", the unit will use the rate stored in it via the MODBUS protocol (default "9600 b/s").

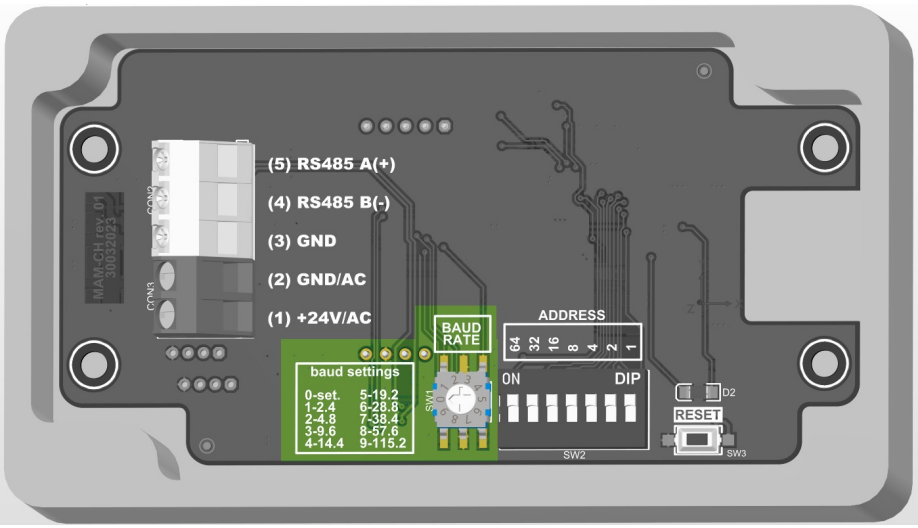


Figure 4. RS-485 interface speed setup.

3.6. Restoring factory settings

Factory reset function only applies to the transmission parameters of the RS-485 interface (including address and baud rate). To restore the settings, press and hold the S1 button for approximately 2 seconds (protection against accidental pressing). When the D1 LED starts blinking, release the button. The unit will start operating with the new settings automatically.

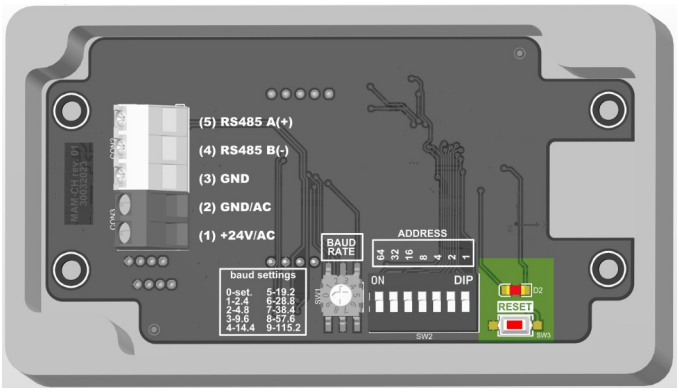


Figure 5. Interface reset.

3.7. Guidelines

- Use shielded cables when operating the device in a high interference environment.
- Connect the cable shield to the nearest PE point on the power supply side.

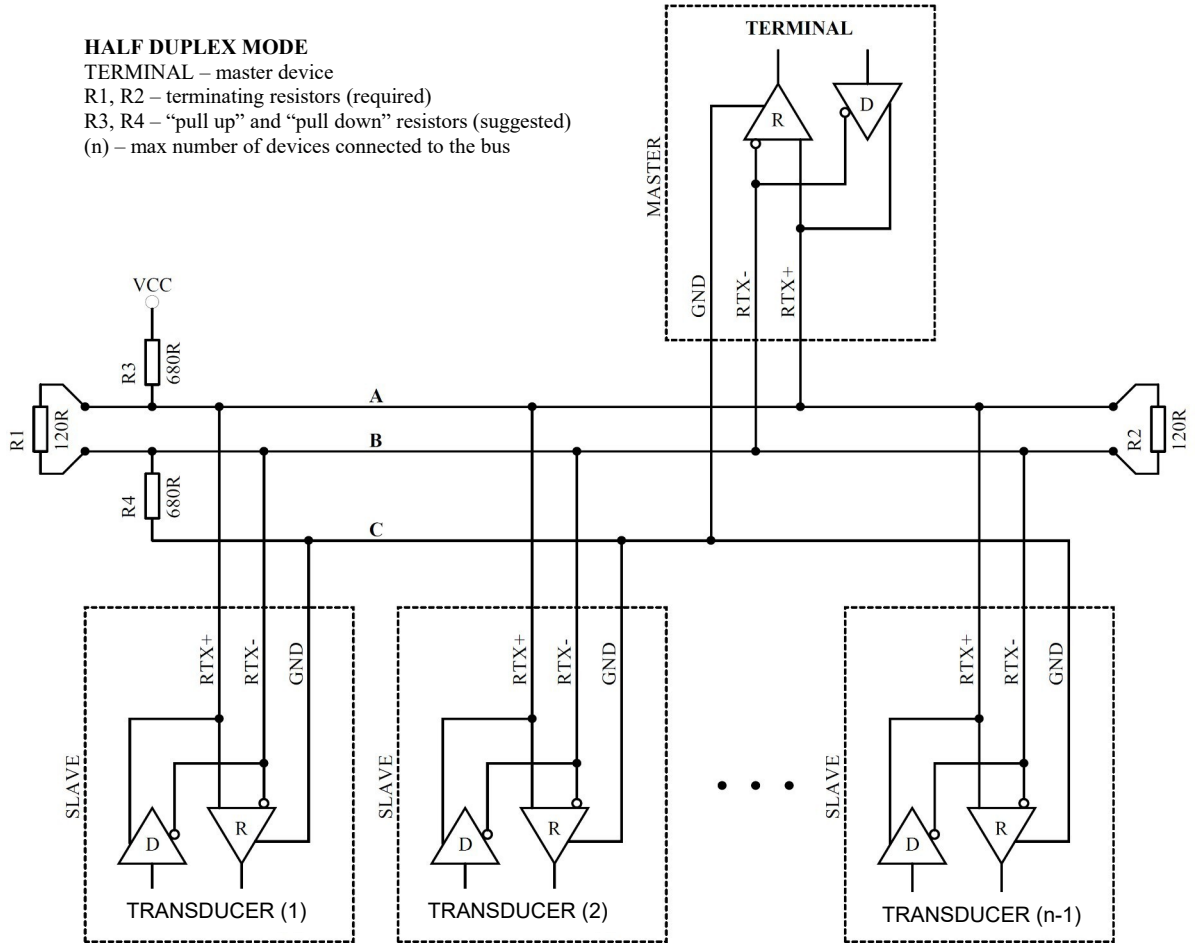


Figure 7. Method of connecting the transmitter to the RS-485 bus operating in HALF DUPLEX mode.

4. MODBUS

4.1. Map of registers

Table of registers:

Register no.	Values	Description
1	0-10000	CO2 concentration
2	0 or	Data status (CO2) 0 - not valid, 1 - OK, 2 - error
3	0-5000	VOC concentration [ppb]
4	0 or 1	Data status (VOC) 0 - not valid, 1 - OK, 2 - error
5		
6	-400 to +1250	Temperature [deciCelsius]
7	0 to 100	Relative humidity [%]
8	-400 to +1250	Dew point [deciCelsius]
9	0 or 1	Data status (temp) 0 - not valid, 1 - OK, 2 - error
10	1234	Password register
11	according to command table	Command register
12	according to command table	Parameter register
20	0-65535	Valid frames counter
21	0-65535	Exceptions counter
22	0-65535	CRC error counter
23	0-65535	Error byte counter
24	1000	Test value - to verify correct reading of registers

Command table:

Command no.	Function	Parameters
1	Set device address	1 - 247 (1 - default value)
2	Set baud rate	24 – 2400 b/s 48 – 4800 b/s 96 – 9600 b/s (default value) 144 – 14400 b/s 192 – 19200 b/s 228 – 28800 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 stop bits	1 - 1 x STOP; 1 stop bit (default value) 2 - 2 x STOP; 2 stop bits
6	Measurement time	6-180 (30 - default value) in seconds
106	Parameter 6 reading	none (read value available on next reading of parameter register)
7	Sampling frequency	60-43200 (600 - default value) in seconds
107	Parameter 7 reading	none (read value available on next reading of parameter register)
8	Reset	none, software reset of device
9	Temperature offset	-100: 100 (default value -8) [deciCelsius]
109	Parameter 9 reading	none (read value available on next reading of parameter register)
10	Humidity offset	-100: 100 (default value 0) [dec %] Note: 1= 0.1%
110	Parameter 10 reading	none (read value available on next reading of parameter register)
11	CO2 autocalibration	0 - automatic CO2 calibration OFF 1 - ON
111	Parameter 11 reading	none (read value available on next reading of parameter register)
12	CO2 calibration	1 - start manual calibration of the CO2 sensor Before starting the calibration of the zero point, ensure 20min of operation of the sensor in air with a CO2 concentration of 400ppm (time of operation 8 seconds)

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 called, it must be accompanied by the entry of a password (1234 decimal).
- Call of the command via individual registry entries must be completed with the entry of a password.Protocol functions

The following MODBUS standard functions are implemented in the transmitter:

CODE	*MEANING*
03 (0x03)	*Read N x 16-bit registers*
16 (0x10)	*Write N x 16-bit registers*
06 (0x06)	*Write single 16-bit reg.*

For all other queries, an `ILLEGAL_FUNCTION` response will be returned.

4.2.1. Reading the 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 bytes*	*as calculated*

Response 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)*
CRC checksum	*2 bytes*	*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 bytes*	*as calculated*

4.2.3. Writing to the single output register (0x06)

Request format:

Description	Size	Values
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x06***
Address of register	*2 bytes*	*0x0000 – 0xFFFF*
Register value	*2 bytes*	*0x0000 - 0xFFFF*
CRC checksum	*2 bytes*	*as calculated*

Response format:

Description	Size	Values
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x06***
Address of register	*2 bytes*	*0x0000 – 0xFFFF*
Register value	*2 bytes*	*0x0000 - 0xFFFF*
CRC checksum	*2 bytes*	*as calculated*

Error format:

Description	Size	Values
Device address	*1 byte*	*1 – 247 (0xF7)*
Function code	*1 byte*	***0x86***
Error code	*1 byte*	*0x01 / 0x02 / 0x03 / 0x04*
CRC checksum	*2 bytes*	*as calculated*

4.2. Data format

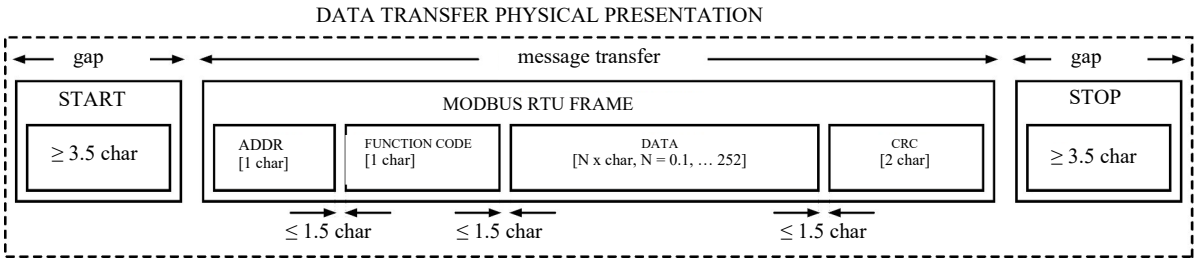


Figure 8. Data transfer in MODBUS RTU standard implemented in the transmitter.

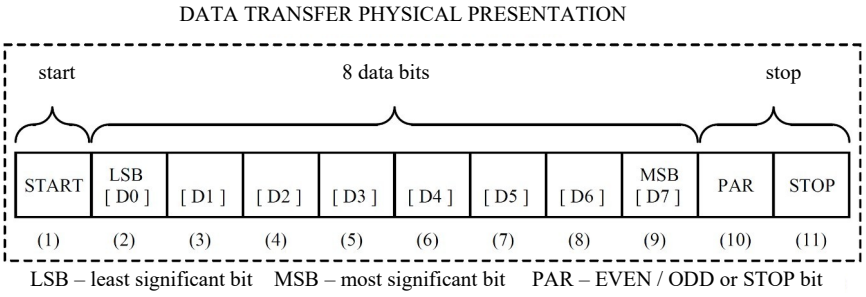


Figure 9. Character format in MODBUS RTU standard implemented in the transmitter.

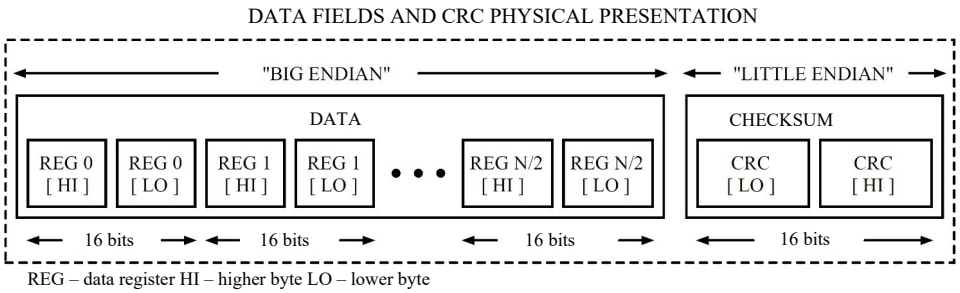


Figure 10. Data field format in MODBUS RTU standard implemented in the transmitter.

4.3. CRC checksum

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

4.4.1. CRC algorithm:

Procedure for determining the CRC checksum using the bitwise method:

a) load 0xFFFF into the 16-bit CRC register,
b) take the first byte of the data block and perform an EX-OR operation with the younger byte of the CRC register, place 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 state of the youngest bit (LSB) in the CRC register, if its state equals 0, return to point c, if 1, perform the EX-OR operation of the CRC register with the constant 0xA001;
e) repeat steps 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, continuing 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) insertion of CRC checksum into a MODBUS RTU frame must be preceded by the swapping of places of the older and the younger byte of the CRC register.

4.4.2. CRC algorithm based on array:

Example of an implementation of the CRC checksum 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
};
```