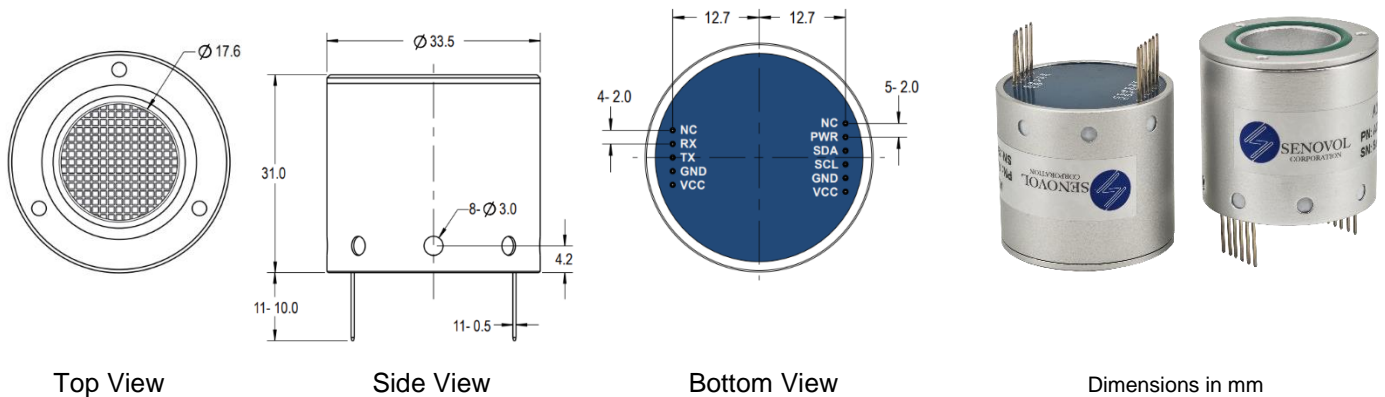


This Digital CO sensor is part of Senovol's Air Quality Monitoring (AQM) sensor series, utilizing optimized electrochemical technology to enhance sensor sensitivity, accuracy, and long-term stability. This design enables the sensor to detect fluctuations in Carbon Monoxide (CO) levels with parts per billion (ppb) precision.

The Digital CO sensor shares the same platform with Digital PID, SO₂, NO, NO₂, O₃, H₂S, and NH₃ sensors. They are all interchangeable using the same digital communication protocol.

Product Dimensions

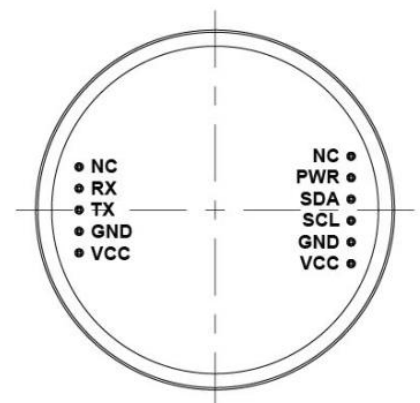


Specifications

| | | | |
|-------------------------------|--|--------------------------------|--|
| Target gas | Carbon Monoxide (CO) | Operating temperature | -40°C ~ +55°C |
| Detection range | 0 ~ 20 ppm Maximum load 100 ppm | Operating humidity | 15% ~ 90% RH Non-condensing |
| Resolution | 15 ppb | Operating pressure | 1 atm ±10% |
| Response time T90 | < 35 seconds | Expected operating life | 3 years in clean air |
| Output mode | UART (3.3V TTL) I ² C (3.3V TTL) | Storage life | 1 year in original package and a mild environment |
| Operating voltage | 3.2 ~ 5.5 VDC | Enclosure material | Aluminum alloy |
| Operating current | ≤ 5mA @5V | Weight | 30 g |
| Long term output drift | <10% signal/year | Warranty | 2 years |

Pinout

| Pin | Function | Description |
|-----|----------|---------------------------|
| 1 | NC | Reserved pins (suspended) |
| 2 | RX | Serial port receiving |
| 3 | TX | Serial port sending |
| 4 | GND | 0V - Ground |
| 5 | VCC | 3.2 ~ 5.5 VDC |
| 6 | NC | Reserved pins (suspended) |



| | | |
|----|-----|--|
| 7 | PWR | Module power enable (low level turns off, high level turns on, built-in pull-up resistor). The default pull-up level is high |
| 8 | SDA | I2C signal SDA, default high level |
| 9 | SCL | I2C signal SCL, default high level |
| 10 | GND | 0V - Ground |
| 11 | VCC | 3.2~5.5 VDC |

* Note: Two VCCs are connected internally.

UART Communication Protocol

1. Settings

Start bit – 1 Data bit – 8 Stop bit – 1 Check bit – None Baud rate –115,200 bps

Without special instructions, the response time is less than 100 ms (please refer to the specific instructions for special circumstances). System cannot respond to other commands until the current command is answered.

2. Frame Format (The format of each communication frame)

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check bit |
|--------|-------------|---------------|------------------|-------------|------|-----------|
| H | ID | F | A | N | D | CRC16 |

H – 1Byte, fixed as 0x3A

ID – 1Byte, defaults as 0x10 and can be customized by users

F – 1Byte, for example (0x03)

A – 2Byte, for example (0x0001)

N – 1Byte, in two bytes, for example (0x02: 4 bytes)

D – N * 2Byte, Big Endian for example (MSB LSB) is defined as signed short

CRC16 – 2Byte, using MODBUS_CRC16 checking algorithm (see Appendix 1 for details)

3. Command Description

3.1 Sensor type reading

Request from host device:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|--------|--------|
| 0x3A | 0x10 | 0x01 | 0x0000 | 0x01 | 0x0000 | 0x82B0 |

For example: 3A 10 01 00 00 01 00 00 82 B0

Module response for correct data receiving:

| Header | Device Code | Function Code | Data | Check |
|--------|-------------|---------------|----------------|-------|
| 0x3A | 0x10 | 0x01 | D (1byte data) | CRC16 |

Sensor type code: 2: CO 3: O2 7: CO2 8: O3 9: H2S 10: SO2 11: NH3 21: NO 22: NO2
23: NOX 34: CH2O 42: PID

For example: 3A 10 01 02 8D 68 (HEX 02 = DEC 2, so the sensor is CO sensor)

3.2 Sensor data reading (unit: $\mu\text{g}/\text{m}^3$)

Request from host device:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|--------|--------|
| 0x3A | 0x10 | 0x03 | 0x0000 | 0x02 | 0x0000 | 0x7352 |

For example: 3A 10 03 00 00 02 00 00 73 52

Module response for correct data receiving:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|------|-------|
| 0x3A | 0x10 | 0x03 | 0x0000 | 0x02 | D | CRC16 |

D: Received data, 4Byte, Big Endian
 For example: 3A 10 03 00 00 02 00 00 00 5E 25 35
 Sensor value ($\mu\text{g}/\text{m}^3$): 00 00 00 5E i.e. $94\mu\text{g}/\text{m}^3$

3.3 Sensor data reading (unit: ppb)

Request from host device:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|--------|--------|
| 0x3A | 0x10 | 0x03 | 0x0002 | 0x02 | 0x0000 | 0x72EA |

For example: 3A 10 03 00 02 02 00 00 72 EA

Module response for correct data receiving:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|------|-------|
| 0x3A | 0x10 | 0x03 | 0x0002 | 0x02 | D | CRC16 |

D: Received data, 4Byte, Big Endian
 For example: 3A 10 03 00 02 02 00 00 00 4C A4 DA
 Sensor value (ppb): 00 00 00 4C i.e. 76ppb

3.4 Sensor temperature data reading ($^{\circ}\text{C}$)

Request from host device:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|--------|--------|
| 0x3A | 0x10 | 0x03 | 0x0004 | 0x01 | 0x0000 | 0x8262 |

For example: 3A 10 03 00 04 01 00 00 82 62

Module response for correct data receiving:

| Header | Device Code | Function Code | Starting address | Data Length |
|--------|-------------|---------------|------------------|-------------|
| 0x3A | 0x10 | 0x03 | 0x0004 | 0x01 |

D: Received data, 2Byte, Big Endian and divided by 100 to get the temperature value.
 For example: 3A 10 03 00 04 01 0A 3D 45 13 (D = $0x0A3D = 2621$ to get the temperature of 26.21°C)

3.5 Sensor humidity data reading (%RH)

Request from host device:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|--------|--------|
| 0x3A | 0x10 | 0x03 | 0x0005 | 0x01 | 0x0000 | 0x839E |

For example: 3A 10 03 00 05 01 00 00 83 9E

Module response for correct data receiving:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|------|-------|
| 0x3A | 0x10 | 0x03 | 0x0005 | 0x01 | D | CRC16 |

D: Received data, 2Byte, Big Endian and divided by 10,000 to get the percentage humidity value
 For example: 3A 10 03 00 05 01 14 89 4D 38 ($0x14 89 = 5257$ to get the humidity of 52.57%)

3.6 Multiple parameters reading (address 0000-0005)

Request from host device:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|--------|--------|
| 0x3A | 0x10 | 0x03 | 0x0000 | 0x06 | 0x0000 | 0x3293 |

For example: 3A 10 03 00 00 06 00 00 32 93

Module response for correct data receiving:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|------|-------|
| 0x3A | 0x10 | 0x03 | 0x0000 | 0x06 | D | CRC16 |

D: Received data, 12Byte

Followed by (MSB first): sensor reading $\mu\text{g}/\text{m}^3$ 4Bytes;

Sensor reading ppb 4Bytes; Temperature 2Bytes; Humidity 2Bytes.

For example: 3A 10 03 00 00 06 00 00 00 8F 00 00 00 50 0A 70 16 11 35 96

Sensor value ($\mu\text{g}/\text{m}^3$): 00 00 00 8F;

Sensor value (ppb): 00 00 00 50;

Temperature: 0A 70; Humidity: 16 11

3.7 Check error response

| Header | Device Code | Function Code | Data | Check |
|--------|-------------|---------------|------|-------|
| 0x3A | 0x10 | 0x08 | 0x00 | CRC16 |

For example: 3A 10 08 00 0A F9

3.8 Zero Calibration

Request from host device:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|--------|--------|
| 0x3A | 0x10 | 0x07 | 0x0000 | 0x01 | 0x0000 | 0x82D6 |

For example: 3A 10 07 00 00 01 00 00 82 D6

Module response for correct data receiving:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|------|-------|
| 0x3A | 0x10 | 0x07 | 0x0000 | 0x01 | D | CRC16 |

D: 2Byte data

For example: 3A 10 07 00 00 01 04 7A 01 F5

Caution: Please place the module in the pure air environment stabilizing for at least 5 minutes then send the Zero calibration command.

3.9 Span Calibration

Request from host device:

| Header | Device Code | Function Code | Starting address | Data Length | Data | Check |
|--------|-------------|---------------|------------------|-------------|--------|--------|
| 0x3A | 0x10 | 0x09 | 0x0000 | 0x01 | 0x0000 | 0x82D6 |

D: 2Byte gas concentration data, Big Endian, Unit: ppm,

For example: 3A 10 09 00 00 01 00 0A 03 FF i.e.: D=0x000A 10ppm gas is used for calibration.

Module response for correct data receiving:

| Header | Device Code | Function Code | Data | Check |
|--------|-------------|---------------|--|-------|
| 0x3A | 0x10 | 0x09 | 0x00 (Success) 0x01 (Processing) 0x02 (Fail) | CRC16 |

For example: 3A 10 09 01 CA A9 (Calibration processing)

Caution: The calibration process for environmental detection is about 300 seconds, and for industry usage is about 60 seconds, please wait for the module to respond before starting the command.

I²C Communication Protocol

1. I²C Interface

| Parameter | Definition | State | Min | Max | Unit |
|-----------------|-------------------------|------------|------|-----|------|
| f _{ck} | I2C clock frequency | Slave Mode | | 100 | kHz |
| t _{su} | Data input setup time | Slave Mode | 6.5 | | ns |
| t _h | Data input Holding Time | Slave Mode | 15.5 | | ns |

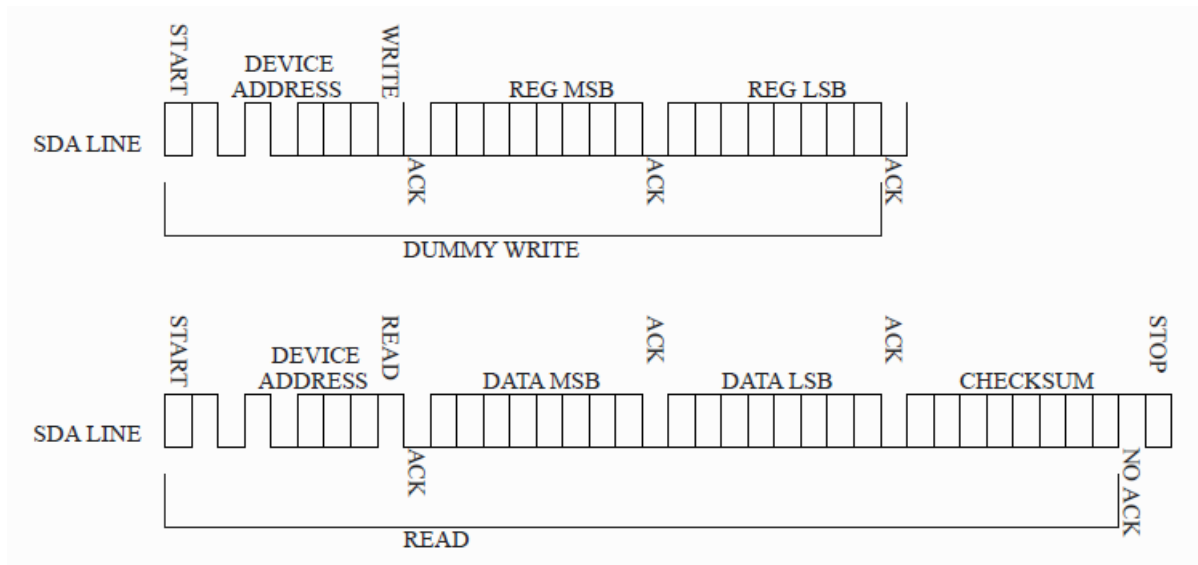
2. Slave Device Address

Slave device addresses can be customized by software tools,
Default setting

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|-----|
| CO | 0 | 0 | 0 | 0 | 0 | 0 | 1 | R/W |
| O3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | R/W |
| SO2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | R/W |
| NO2 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | R/W |

I²C Sensor Type Address: CO: 0x02 O3: 0x08 H2S: 0x06 SO2: 0x0A NH3: 0x10 NO: 0x1E
NO2: 0x16 CH2O: 0x22 PID: 0x42

3. I²C Communication Protocol



CHECKSUM is cumulative and retrieve check. See Appendix 2 for details.

4. Data Analysis

The REG parameters are shown below, which is the data address in the module.

| REG | MSB | LSB |
|-----------------------------------|------|------|
| Sensor value (ug/m ³) | 0x00 | 0x00 |
| Sensor value (ppb) | 0x00 | 0x01 |
| Temperature (°C) | 0x00 | 0x02 |
| Humidity (%RH) | 0x00 | 0x03 |

Data examples:

| DATA | MSB | LSB | CHECKSUM | Actual value | Remarks |
|-----------------------------------|------|------|----------|--------------------|---|
| Sensor value (ug/m ³) | 0x00 | 0x05 | 0xFA | 5ug/m ³ | Same with USART Data Conversion Method |
| Sensor value (ppb) | 0x00 | 0x05 | 0xFA | 5ppb | |
| Temperature | 0x0A | 0x3D | 0xB8 | 26.21°C | |
| Humidity | 0x14 | 0x89 | 0x62 | 52.57% | |

Caution: To receive 5Byte data when reading sensor value, and to receive 3Byte data when reading temperature and humidity.

Warning!

- This product does not have any intrinsic safety certification or explosion proof certification. Please do NOT use this product in any hazardous locations.
- This product does not have reverse power protection and Electrostatic Discharge (ESD) protection. Please carefully verify the electrical polarity and make the ESD protection before each use or installation.
- Please use a stable DC power supply for this gas sensor module. It is highly recommended to use a power supply with the output voltage fluctuation less than 1%.

Appendix 1: MODBUS CRC16 algorithm

```
unsigned short modbus_CRC16(unsigned char *ptr, unsigned char len)
{
    unsigned short wrcr=0xFFFF; //
    int i=0, j=0;
    for (i=0; i<len; i++)
    {
        wrcr^=*ptr++;
        for ( j=0; j<8; j++)
        {
            if (wrcr&0X0001)
            {
                wrcr=wrcr>>1^0XA001;
            }
            else
            {
                wrcr>>=1;
            }
        }
    }
    return wrcr<<8| wrcr>>8; //little endian (LSB fist)
}
```

In CRC calculation, only 8 data bits, start bit and stop bit are used. If there are parity bit, including this bit, they are not involved in CRC calculation. The CRC calculation method:

1. Load a 16 bit register with the value of 0 xfff, which is CRC register.
2. The XOR of the first 8-bit binary data (the first byte of communication information frame) and the 16 bit CRC register is still stored in the CRC register.
3. Move the contents of CRC register one bit to the right, fill the highest bit with 0, and detect whether the moved out bit is 0 or 1.
4. If the move out bit is zero, repeat the third step (move one bit to the right again); If the shift out bit is 1, the CRC register XORs with 0xa001.

5. Repeat steps 3 and 4 until the right shift is 8 times, so that the entire 8-bit data is processed.
6. Repeat steps 2 and 5 to process the next byte of the communication information frame
7. After all the bytes of the communication information frame are calculated according to the above steps, the high and low bytes of the 16 bit CRC register are exchanged
8. Finally, the content of CRC register is CRC check code.
9. For example, a command 3A 10 09 00 00 01 00 32 get the wrc value of 02 2D through the above program. In this way, we get the calibration command: 3A 10 09 00 00 01 00 32 02 2D.

Appendix 2: CHECKSUM Accumulation and Verification

```

unsigned char CheckSum(unsigned char *buf, unsigned char len) //return CheckSum value
{
    unit8_t i, ret = 0;

    for (i=0; i<len; i++)
    {
        ret +=*(buf++);
    }

    ret = ~ret;
    return ret;
}

```

The check sum method:

Sender: accumulate the data to get a sum, and reverse the sum to get our check value. Then send the data with the check value to the receiver.

For example:

Sender: to send 0xA8 and 0x50, we use the unsigned char (8 bits) to save the accumulated sum, i.e. 0xf8 (0b11111000), and get the check sum of 0x07 (0b00000111). Then send these three data out.