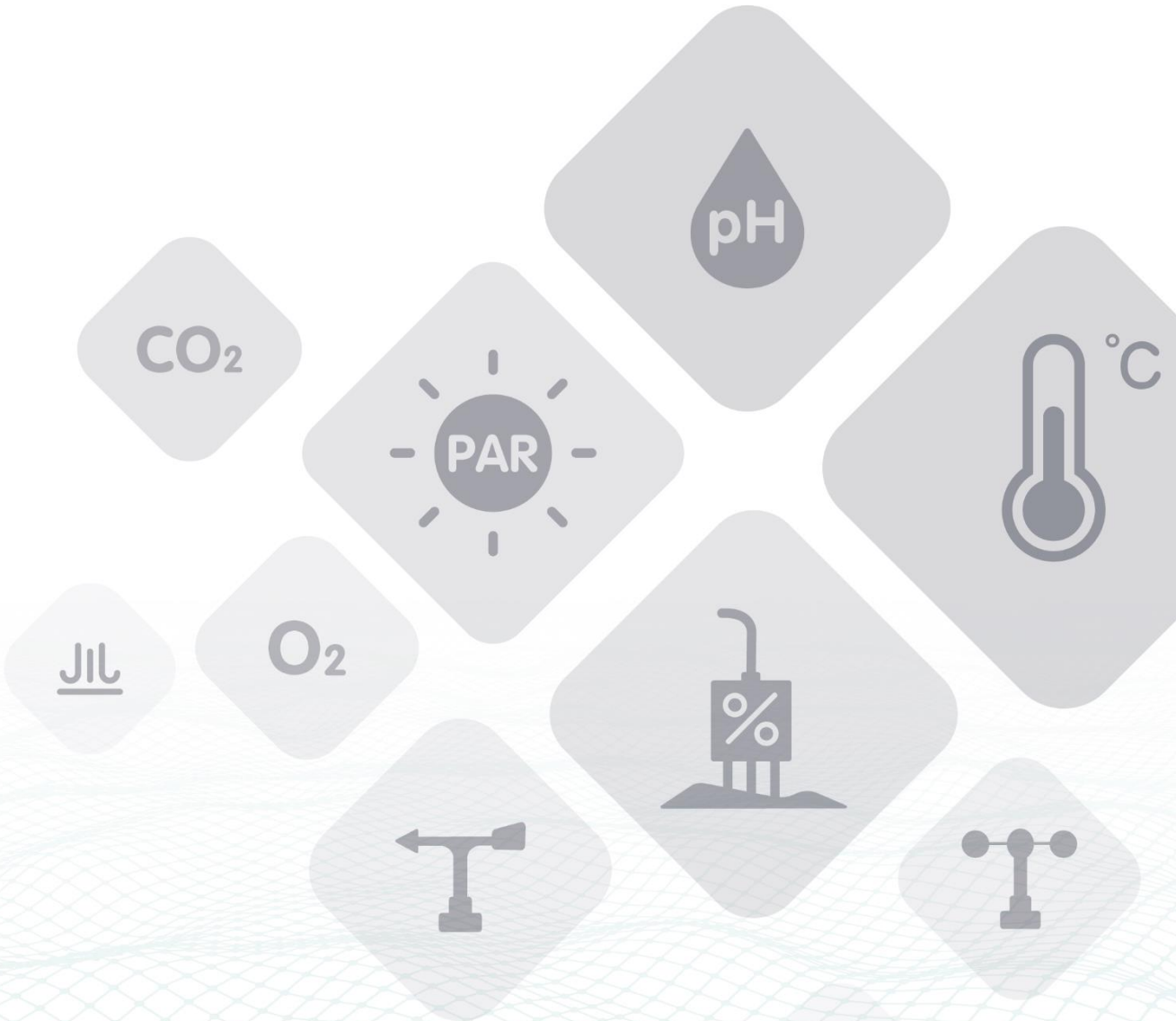
SENSECAP

# SenseCAP ORCH S4 User Guide

**Version:** V1.0



# 1 Product Introduction

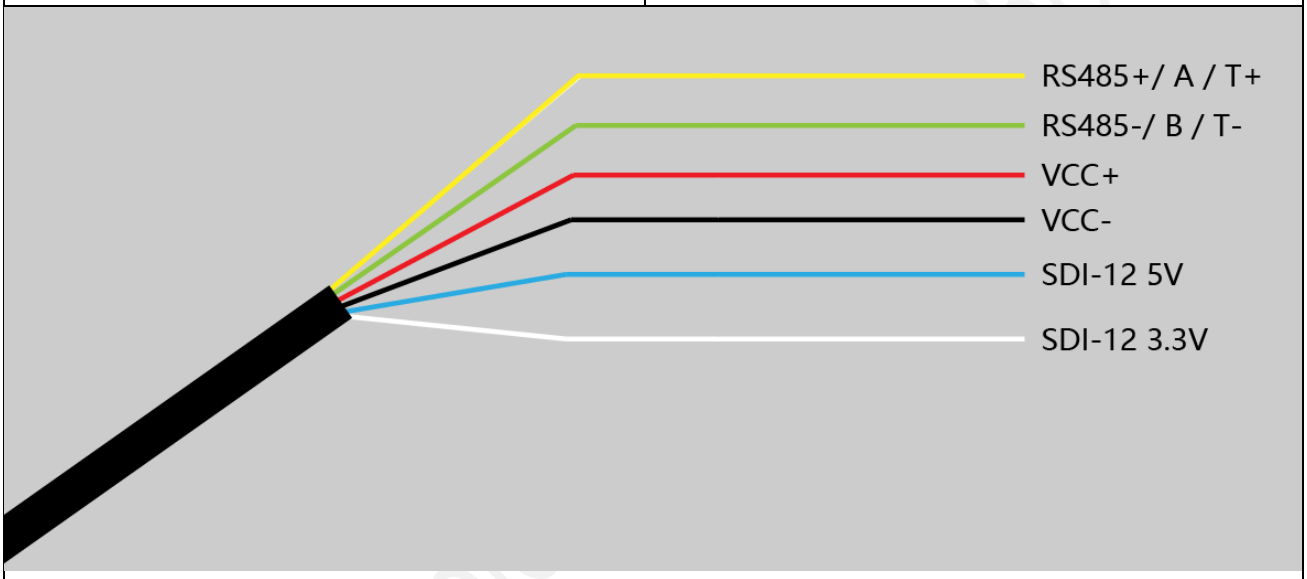
The SenseCAP ORCH S4 is designed for the professional environmental monitoring application. It measures and provides reliable data for the customers, even in extreme environmental conditions. The SenseCAP ORCH S4 provides four of the most important environmental parameters, which are air temperature, relative humidity, barometric pressure, and light intensity through a flexible combination.

It supports two standard protocols: MODBUS (MODBUS-RTU / MODBUS-ASCLL) and SDI-12, with low power consumption and wide range power voltage from 3.6V to 16V. Built-in data pre-processing, therefore customers can quickly complete system development and integration.

The Solar Radiation Shield can effectively reduce the impact of solar radiation. which improve the accuracy of detection. The enclosure can protect the device from rain and sunlight exposure, etc.

# 2 Wiring

| Sensor wiring | |
|---|---|
| Yellow | RS485+ / A /T+ |
| Green | RS485- / B /T- |
| Red | VCC+ |
| Black | VCC- (GND) |
| Blue | SDI-12 5V |
| White | SDI-12 3.3V |



# 3 Specifications

| **General Parameters** | | | | | |
|---|---|---|---|---|---|
| Product Model | SenseCAP ORCH S4 | | | | |
| Power Supply | 3.6 ~ 16V | | | | |
| Protocol | MODBUS-RTU RS485/ MODBUS-ASCII RS485/ SDI-12 (v1.4) | | | | |
| IP Rating | IP66 (water-proof box)；IPX5 (solar radiation shield) | | | | |
| Operating Temperature | -40 ℃ ~ +65 ℃ | | | | |
| Operating Humidity | 0 ~ 100 %RH (non-condensing) | | | | |
| **Measurement Parameters** | | | | | |
| | Range | Accuracy | | Resolution | Drift |
| Air Temperature | -40~+85 ℃ | ±0.2 ℃ | | 0.1 ℃ | < 0.3 ℃/year |
| Air Humidity | 0~100 %RH | ±1.5 %RH | | 1 %RH | < 0.25% RH/year |
| Barometric Pressure | 300~1100 hPa | Relative Accuracy (700~900 hPa/ 25~40℃): ±0.12 hPa<br>Absolute Accuracy (300~1100 hPa/ -20~0℃): ±1.7 hPa<br>Absolute Accuracy (300~1100 hPa/ 0~65 ℃): ±1.0 hPa | | 1 Pa | ±1.0 hPa/year |
| Light Intensity | 0~188000 Lux | ± 5% | | 0.045 Lux | — |
| **Current Consumption** | | | | | |
| | Power Supply (V) | 16 | 12 | 5 | 3.6 |
| RS-485 | working current (mA) | 3.60 | 4.70 | 11.00 | 14.50 |
| SDI-12 | working current (mA) | 3.04 | 3.98 | 8.85 | 11.80 |
| | Sleep current (μA) | 40.00 | 45.30 | 77.60 | 2.24 (mA) |
| **Program Parameters** | | | | | |
| | Parameter | Min | Typical | Max | Unit |
| RS-485 | Warm-up Time[1] | 300 | 300[2] | 1000 | millisecond |
| | Scan Interval [3] | — | 1000 | — | millisecond |
| | Poll Rate[4] | — | 1 | — | Hz |
| | Response Time [5] | 1 | — | 4 | millisecond |
| SDI-12 | Warm-up Time [6] | — | 15 | — | millisecond |
| | Scan Interval [7] | 350 | 350[8] | 1200 | millisecond |

[1] The time from when the sensor is powered on to when the data is read. Note the parameter when the sensor is powered on.

[2] When the light intensity is greater than 150 Lux, the warm-up time of the value (@typical 300ms) can ensure 100% reading of the correct data. When the light intensity is less than 150 Lux, the warm-up time of the value (@max 1000ms) can ensure 100% reading of the correct data.

[3] Measure data update time interval. The sensor periodically updates the readings after the power-on warm-up time, if the power supply continues.

[4] Modbus master poll rate.

[5] When the delay response register 0x0020 is set to 0, the time from the sensor receiving the read instruction to the start of sending data.

[6] The time from when the sensor is powered on to when the data is read. Note the parameter when the sensor is powered on.

[7] It refers to the time between the data logger sending the ‘aM!’ and the sensor responding to the service request. Note the parameter when the data logger does not wait for the service request but delays for a period of time and directly send ‘aD0!’.

[8] When the light intensity is greater than 150 Lux, the warm-up time of the value (@typical 350ms) can ensure 100% reading of the correct data. When the light intensity is less than 150 Lux, the warm-up time of the value (@max 1200ms) can ensure 100% reading of the correct data.

# 4 How to work on RS-485

## 4.1 Introduction to Modbus-RTU RS-485

Modbus protocol is a common protocol used in electronic devices. Through this protocol, the devices communicate with each other. It has become a common industry standard, widely used in data logger, sensor equipment and so on. Based on this protocol, devices from different vendors can communicate with each other for system integration.

Modbus protocol is a master-slave architecture protocol. One node is the Master and the other nodes that use Modbus protocol to participate in the communication are Slaves. Each slave device has a unique address. The SenseCAP ORCH S4 has RS485 interface and supports Modbus-RTU and Modbus-RTU ASCLL. Modbus commands can be used to obtain sensor data or modify communication parameters.

> **Note:**
>
> Default communication parameters: baud rate 9600bps, one start bit, eight data bits, no check, one stop bit.

## 4.2 Modbus Register

| Parameter | Register Address | Type | Function Code | Range and Descript | Default |
|---|---|---|---|---|---|
| | | | | | |
| **Read-only Register** | | | | | |
| Air Temperature | 0x0000 | int16 (complement), read-only | 3、4 | Value=Register Value * 0.01<br>Unit: ℃ | N/A |
| Air Humidity | 0x0001 | uint1, read-only | 3、4 | Value= Register Value * 0.01<br>Unit: %RH | N/A |
| Barometric Pressure MSB | 0x0002 | uint16, read-only | 3、4 | Value=(uint32)(Register 0x0002 << 16 + Register 0x0003)<br>Unit: Pa | N/A |
| Barometric Pressure LSB | 0x0003 | uint16, read-only | 3、4 | | N/A |
| Light Intensity MSB | 0x0004 | uint16, read-only | 3、4 | Value =((uint32)( Register 0x0004 << 16 + Register 0x0005)) * 0.001<br>Unit: Lux | N/A |
| Light Intensity LSB | 0x0005 | uint16, read-only | 3、4 | | N/A |
| Device Type | 0x0006 | uint16, read-only | 3、4 | 4：SenseCAP ORCH S4<br>3：SenseCAP ORCH S3<br>1：Light Intensity Sensor | N/A |
| Version | 0x0007 | uint16, read-only | 3、4 | High Byte: Hardware Version<br>Low Byte: Software Version | N/A |
| | | | | | |
| **Protocol Configuration Register** | | | | | |
| Modbus Slave Address | 0x0010 | uint16, read-only | 3、6 | 1 ~ 247 | 1 |
| Baud Rate | 0x0011 | uint16, read-only | 3、6 | 0 ~ 7<br>0：1200bps<br>1：2400bps<br>2：4800bps<br>3：9600bps<br>4：19200bps<br>5：38400bps<br>6：57600bps<br>7：115200bps | 3 |
| Parity | 0x0012 | uint16, | 3、6 | 0 ~ 2 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | read-only | | 0：None<br>1：Odd<br>2：Even | |
| Stop Bit | 0x0013 | uint16,<br>read-write | 3、6 | 0 ~ 1<br>0：1 Stop Bit<br>1：2 Stop Bit | 0 |
| Modbus protocol | 0x0014 | uint16,<br>read-write | 3、6 | 0-1<br>0：Modbus-RTU<br>1：Modbus-ASCII | 0 |
| | | | | | |
| **Function Configuration Register** | | | | | |
| Delay Response | 0x0020 | uint16,<br>read-write | 3、6 | 0 ~ 65535<br>Unit: millisecond | 10 |
| Reserved | 0x0021 | | | | |

- **Error Code**

If an error occurs from the slave, an error response is returned.

| Address | Error Function Code | Error Code | CRC Check |
|---|---|---|---|
| 1 byte | 1 byte | 1 byte | 2 byte |
| | Request Code+0x80 | 0x1 ： Function code error<br>0x2 ： Register address error<br>0x3 ： Write or read data out of range<br>0x4 ： Slave internal error | |

## 4.3 Modbus Register Detail Description

| **Air Temperature** | | |
|---|---|---|
| Range | -4000 ~ 8500 corresponding -40 ~ 85 ℃ | Register Address: 0x0000 |
| For example:<br>If the return value is 0x0AEE(HEX) = 2798 (DEC), then the temperature measurement is 2798/100=27.98 ℃.<br>If the value returned is FF05H (hexadecimal, complement), then the original code is - (FA+1) (HEX) = -251 (DEC), then the temperature measurement is -251/100 = -2.51℃ | | |

| **Air Humidity** | | |
|---|---|---|
| Range | 0 ~ 1000 corresponding 0 ~ 100 %RH | Register Address: 0x0001 |
| For example:<br>If the return value is 0x1AB2 (HEX), then 1AB2 (HEX) = 6834 (DEC), then the temperature measurement is 6834/100=68.34%RH | | |

| **Barometric Pressure** | | |
|---|---|---|
| Range | 30000 ~ 110000 corresponding 30000 ~ 110000 Pa | Register Address: 0x0002; 0x0003 |
| For Example:<br>If the return value is 0x0001 (MSB), 0x87D5 (LSB), it is actually (0x0001<<16) + 0x87D5 = 0x000187D5=100309 (DEC), so the measured barometric pressure is 100309Pa | | |

| **Light Intensity** | | |
|---|---|---|
| Range | 0 ~ 188000000 corresponding 0 ~ 188000 Lux | Register Address: 0x0004; 0x0005 |
| For Example:<br>If the return value is 0x0005 (MSB), 0xFA00 (LSB), then it is actually (0x0005<<16) + 0xFA00 = 0x0005FA00=391680 (DEC), so the measured atmospheric pressure is 391680/1000=391.680Lux | | |

| **Device Type** | | |
|---|---|---|
| Range | 1、3、4 (default: 4) | Register Address: 0x0006 |
| If the return value is 0x0004, the device type is the SenseCAP ORCH S4 | | |

| **Version** | | |
|---|---|---|
| Range | High byte: hardware version; Low byte: software version | Register Address: 0x0007 |
| If the return value is 0x0B0A, the hardware version = 0x0B / 10 =1.1; software version = 0x0A / 10 = 1.0 | | |

| **Modbus Slave Address** | | |
|---|---|---|
| Range | 1 ~ 247 (default: 1) | Register Address: 0x0010 |
| The device needs to be restarted to take effect after setup. | | |

| **Baud Rate** | | |
|---|---|---|
| Range | 0 ~ 7 (default: 3) | Register Address: 0x0011 |
| 0: 1200bps | | |

| |
|---|
| 1：2400bps<br>2：4800bps<br>3：9600bps<br>4：19200bps<br>5：38400bps<br>6：57600bps<br>7：115200bps<br>The device needs to be restarted to take effect after setup. |

| **Parity Bits** | | |
|---|---|---|
| Range | 0 ~ 2 (default: 0) | Register Address: 0x0012 |
| 0：None<br>1：Odd<br>2：Even<br>**Note:** when parity is enabled, the Master should set the data bit to 7 and the secure communication protocol to Modbus-ASCII.<br>The device needs to be restarted to take effect after setup. | | |

| **Stop Bits** | | |
|---|---|---|
| Range | 0 、1 (default: 0) | Register Address: 0x0013 |
| 0：1 Stop bit<br>1：2 Stop bits<br>The device needs to be restarted to take effect after setup. | | |

| **Modbus Protocol** | | |
|---|---|---|
| Range | 0 、1 (default: 0) | Register Address: 0x0014 |
| 0：Modbus-RTU<br>1：Modbus-ASCII<br>The device needs to be restarted to take effect after setup. | | |

| **Delay Response** | | |
|---|---|---|
| Range | 0 ~ 65535 millisecond (default: 10) | Register Address: 0x0020 |
| After receiving the request from the Master, the sensor will be sampled, and the response will be given after the completion of the sampling. This command is mainly used in the situation where the speed of the Master is slow when switching from RS485 sending state to receiving state. When set to 0, there is no extra delay.<br>The device needs to be restarted to take effect after setup. | | |

## 4.4 Example

In the following instructions, data beginning with 0x or ending with H is hexadecimal. The Modbus protocol has two common register types:

(1) keep the register, the storage data is not lost power, is readable and writable. Normally read with function number 3 (0x03) and write with function number 6 (0x06) or 16 (0x10).

(2) input register, used to store some read-only physical quantities, such as temperature value, is read-only. Normally read with function number 4 (0x04).

### 4.4.1 Example of Function Code 3

Request command: AA 03 RRRR NNNN CCCC

| AA | 03 | RRRR | NNNN | CCCC |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Bytes | 2 Bytes | 2 Bytes |
| Slave address, range 0-247 | Function Code is 3 | Start register address, high byte first | The number of registers to read, high byte first | CRC check |

Response: AA 03 MM VV0 VV1 VV2 VV3… CCCC

| AA | 03 | MM | VV0 | VV2 | … | CCCC |
|---|---|---|---|---|---|---|
| 1 Byte | 1 Byte | 1 Bytes | 2 Bytes | 2 Bytes | … | 2 Bytes |
| Slave address, range 0-247 | Function Code is 3 | Returns the number of data bytes of the register value | The first register value returned | The second register value returned | The N register value returned (N=MM/2) | CRC check |

**For example:** Read registers 0x0000 ~ 0x0007 to read the measured values of air temperature and humidity, barometric pressure and light intensity.

Request: 01 03 0000 0008 440C

| 01 | 03 | 0000 | 0008 | 440C |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Bytes | 2 Bytes | 2 Bytes |
| Slave address 0x01 | Function Code is 3 | Start register address: 0x00 | Read 8 registers | CRC check |

Response: 01 03 10 0AE7 1CF1 0001 882F 0000 5FA0 0004 0A0A 8B07

| 01 | 03 | 10 | 0AE7 | 1CF1 | 0001 | 882F | 0000 | 5FA0 | 0004 | 0A0A | 8B07 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 Byte | 1 Byte | 1 Bytes | 2 Bytes | 2 Bytes | 2 Bytes | 2 Bytes | 2 Bytes | 2 Bytes | 2 Bytes | 2 Bytes | 2 Bytes |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Slave address 0x01 | Function Code is 3 | Returns 16 bytes of register data | Returns the value of register 0x0000 | Returns the value of register 0x0001 | Returns the value of register 0x0002 | Returns the value of register 0x0003 | Returns the value of register 0x0004 | Returns the value of register 0x0005 | Returns the value of register 0x0006 | Returns the value of register 0x0007 | CRC check |

### 4.4.2 Example of Function Code 6

Request: AA 06 RRRR NNNN CCCC

| AA | 06 | RRRR | NNNN | CCCC |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Bytes | 2 Bytes | 2 Bytes |
| Slave address, range 0-247 | Function Code is 6 | Write to the register address, high byte first | Write the value of the register, high byte first | CRC check |

Response: AA 06 RRRR VVVV CCCC

| AA | 06 | RRRR | VVVV | CCCC |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Bytes | 2 Bytes | 2 Bytes |
| Slave address, range 0-247 | Function Code is 6 | Write to the register address, high byte first | Write the value of the register, high byte first | CRC check |

For example: Write register 0x0010 and modify the slave address of the device to 0x02.

Request: 01 06 0010 0002 09CE

| 01 | 06 | 0010 | 0002 | 09CE |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Bytes | 2 Bytes | 2 Bytes |
| Slave address, range 0-247 | Function Code is 6 | Write to the register address, high byte first | Write the value of the register, high byte first | CRC check |

Response: 01 06 0010 0002 09CE

| 01 | 06 | 0010 | 0002 | 09CE |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Bytes | 2 Bytes | 2 Bytes |
| Slave address, range 0-247 | Function Code is 6 | Write to the register address, high byte first | Write the value of the register, high byte first | CRC check |

### 4.4.3 CRC Check algorithm

```
static const unsigned char aucCRCHi[] = {
    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
};

static const unsigned char aucCRCLo[] = {
    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
};

unsigned short usCRC16( unsigned char * pucFrame, unsigned short usLen )
{
    unsigned char    ucCRCHi = 0xFF;
    unsigned char    ucCRCLo = 0xFF;
    int              iIndex;

    while( usLen-- )
    {
        iIndex = ucCRCLo ^ *( pucFrame++ );
        ucCRCLo = ( UCHAR )( ucCRCHi ^ aucCRCHi[iIndex] );
        ucCRCHi = aucCRCLo[iIndex];
    }
    return ( unsigned short )( ucCRCHi << 8 | ucCRCLo );
}
```

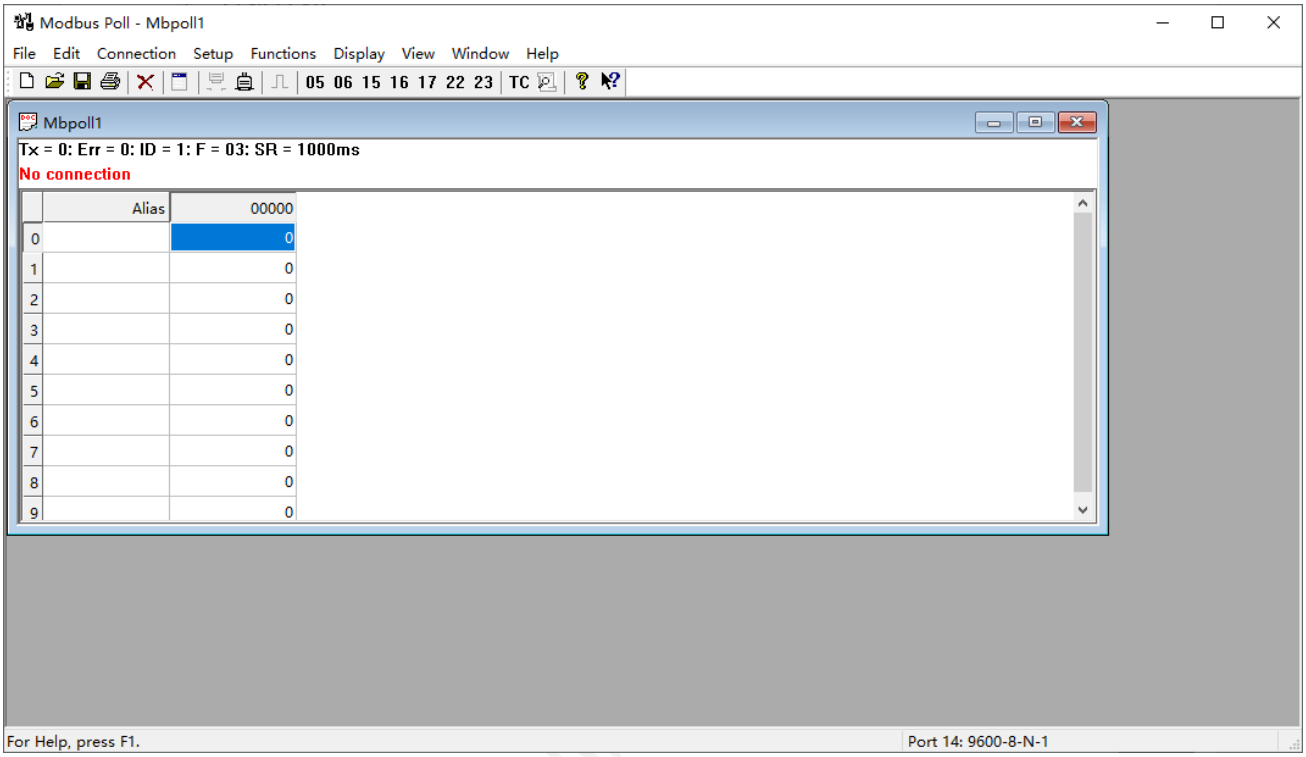
The CRC generated by this function has exchanged high and low bytes and can be sent directly into a message.

**For example:** The CRC16 of a certain frame is calculated by this function to be equal to 0x4112, then the message is placed as follows:

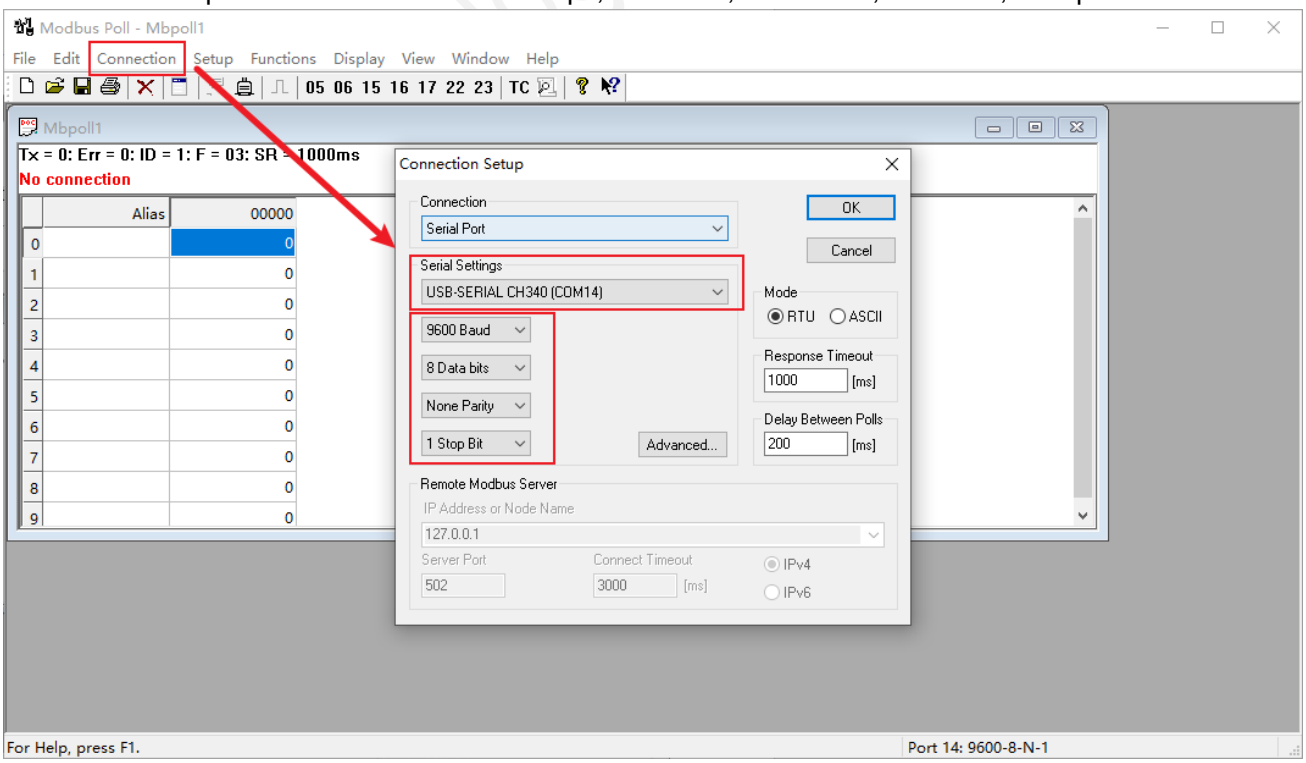
| AA | 03 | RRRR | NNNN | CC | CC |
|---|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Bytes | 2 Bytes | 1 Byte | 1 Byte |
| Slave address, range 0-247 | Function Code is 3 | Start register address, high byte first | The number of registers to read, high byte first | CRC check, low byte 0x41 | CRC check, high byte 0x12 |

# 4.5 Modbus Poll Tool

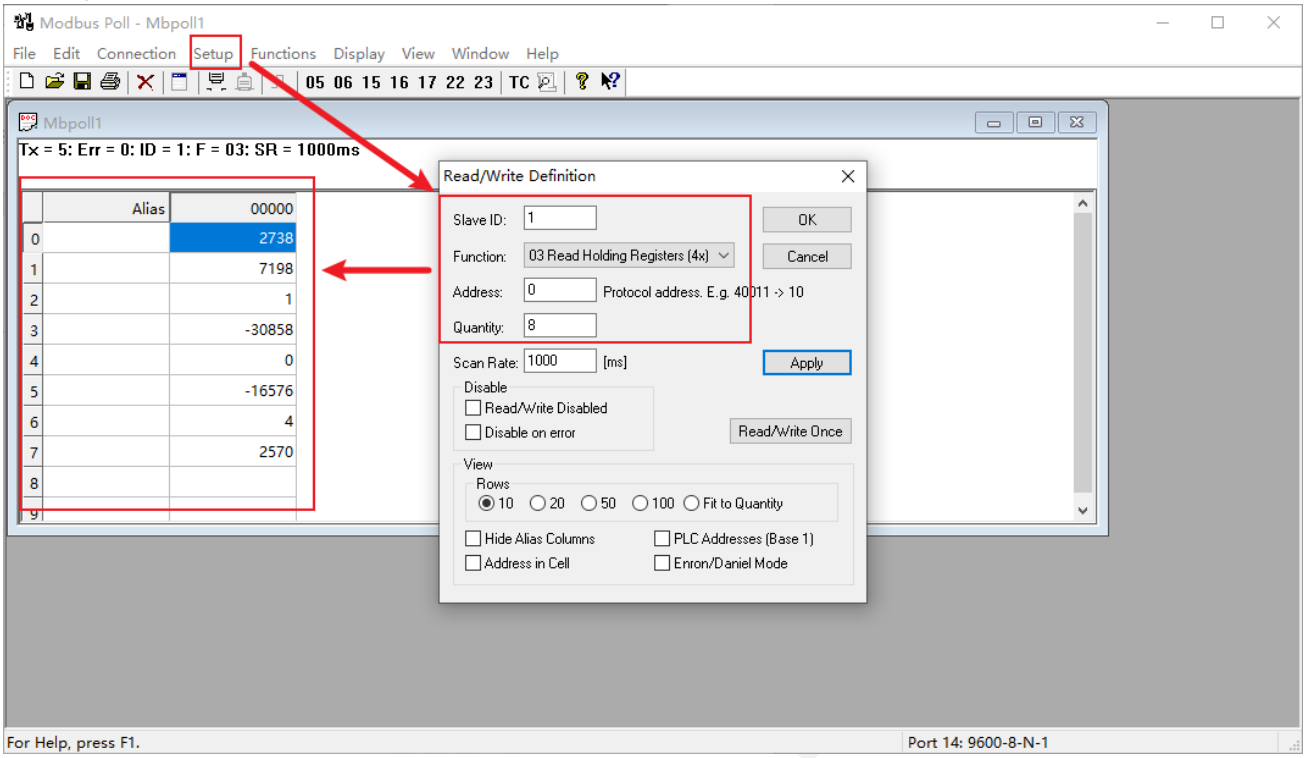
Take the Modbus Poll tool as an example.
Download: https://www.modbustools.com/download.html




Communication parameters: baud rate 9600bps, 1 start bit, 8 data bits, no check, 1 stop bit.



Configure the parameters of read registers 0x00~0x07: the slave address defaults to 1, function code 03, starting address 0, and number 8.



# 5 SDI-12

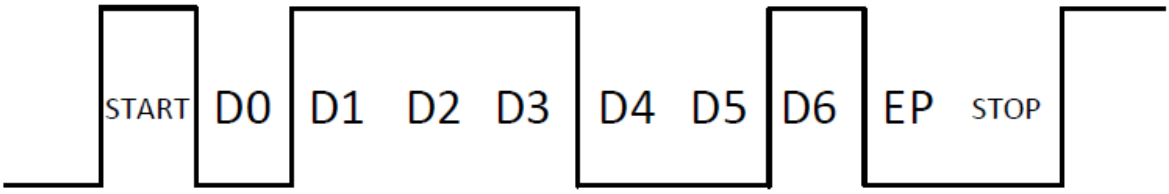
## 5.1 Introduction of SDI-12

SDI-12 communication adopts three wires, two of which are sensor power supply wires and the other is SDI-12 signal wire.

Each sensor on the SDI-12 bus has a unique address, which can be set to '0', '1' ~ '9', 'A' ~ 'Z', 'A' ~ 'Z'. The SDI-12 address of the SenseCAP ORCH S4 defaults to '0'. The instructions supported by this sensor are shown in the next chapter, where each instruction conforms to the SDI-12 v1.4.

The sensor is powered by a DC power supply of 3.6~16V. After the sensor is powered on, it will go into sleep mode immediately and wait for the data acquisition equipment to give instructions. SDI-12 uses baud rate 9600bps, 1 start bit (high level), 7 data bits (high 0 and low 1, anti-logic), 1 even parity bit and 1 stop bit.

The sequence of each byte sent is shown in the following figure:



## 5.2 SDI-12 Commands

| **Query the device address** | | |
|---|---|---|
| Request | ?! | ? - Address wildcard<br>! - Instruction terminator |
| Response | a<CR><LF> | a - Address<br><CR><LF> - End of the response |
| Example | Request：?!<br>Response：0<CR><LF> | The sensor at address '0' responded to the query |

| **Query the device status** | | |
|---|---|---|
| Request | a! | a - Address<br>! - Instruction terminator |
| Response | a<CR><LF> | a - Address<br><CR><LF> - End of the response |
| Example | Request：0!<br>Response：0<CR><LF> | Address '0' of device online |

| **Query the device information** | | |
|---|---|---|
| Request | aI! | a - Address<br>I – Identification<br>! - Instruction terminator |
| Response | allccccccccmmmmmmvvvxxx . . . xxx<CR><LF> | a - Address<br>II - 2 characters, SDI-12 protocol version, for example 14 represents v1.4<br>cccccccc - 8 characters, company name or product name<br>mmmmmm - 6 characters, sensor type<br>vvv - 3 characters, software version<br>xxx . . . xx - Optional, up to 13 characters, can be used to send serial numbers, or other information<br><CR><LF> - End of the response |
| Example | Request：0I!<br>Response：014SENSECAPORCHSx1.0004A0040THPL | 0I! - Command<br>0 - Address<br>14 – Hardware version 1.4<br>SENSECAP – Product brand<br>ORCHSx -Product model<br>1.0 – Software version 1.0 |

| | | |
|---|---|---|
| | | 004A0040 - 8 characters, serial number<br>THPL - Output measurement list, T temperature, H humidity, P atmospheric pressure, L light |
| **Modify device address** | | |
| Request | aAb! | a –The current address<br>A - Address Change<br>b – The new address<br>! - Instruction terminator |
| Response | b<CR><LF> | b –The new address<br><CR><LF> - End of the response |
| Example | Request：0A1!<br>Response：1<CR><LF> | The address 0 was changed to 1. The response received was modified successfully |

| **Start measuring** | | |
|---|---|---|
| Request | aM!<br>或 aMC! | a - Address<br>M - Measure<br>C - CRC<br>! - Instruction terminator |
| Response | atttn<CR><LF> | a - Address<br>ttt - 3 characters, measurement end time, unit: second<br>n - 1 character, the number of measurements to be output<br><CR><LF> - End of the response<br>When the data logger sends ‘aMC!’, the response returns data with a CRC. |
| Example | Request：0M!<br>Response：00024<CR><LF> | 0M! - Command<br>0 - Address<br>002 - The detection is completed after a maximum of 2 seconds<br>4 – Four measurements will be output |
| Service request | a<CR><LF> | a - Address<br><br>When the data logger sends a instruction, the sensor immediately responds to ' atttn<CR><LF>', and when the measurement is finished, the service request will be replied |

| | | |
|---|---|---|
| | | to inform the data logger that the data can be collected.<br>The data collector should not request another sensor between the time the aM! is sent and the time it receives service request, unless the measurement is interrupted with a break (the measurement value will not be updated). |

| **Read measured value** | | |
|---|---|---|
| Request | aD0! | a - Address<br>D0 - Data（D1 . . . D9 and so on）<br>! - Instruction terminator |
| Response | a<values><CR><LF> or a<values><CRC><CR><LF> | a - Address<br>values - as follows<br><CRC> - 3 characters, CRC, response to aMC! carry<br><CR><LF> - End of the response values：<br><Temperature><Humidity><Barometric Pressure><Light Intensity>，The order and number are shown in the ‘aI!’, in which the measured value is composed of the following parts:<br><symbol><integer>[.<decimal>] |
| Example | Request：0D0!<br>Response：0+25.6+88.9+111111+33333.33 | 0D0! - Command<br>0 - Address<br>+25.6 – Temperature, ℃<br>+88.9 – Humidity, %RH<br>+111111 – Barometric, Pa<br>+33333.33 - Light Intensity, Lux |

## 5.3 Precautions for the use of SDI-12

(1) multiple sensors can be mounted on the sdi-12 bus, but the state maintenance of sensors should be paid attention to and the failed sensors should be detected in time, because the failure of one sensor may affect the normal work of the whole bus, even if other sensors are normal.

(2) Under the power supply of 3.3V, there can only be one device on the SDI-12 bus, with no influence under other voltages.

(3) when the data collector operates the sensor, retry should be included in the logic, otherwise there will be a certain probability that the data cannot be read due to cable interference, baud rate deviation and other reasons.