



## RS485 Sensor Node V1.0 (SKU:DFR0233)

---



### Contents

- [1 Introduction](#)
- [2 Application](#)
- [3 Specification](#)
- [4 Pin Out](#)
  - [4.1 More details](#)
- [5 Product Directive](#)
- [6 Connecting Diagram](#)
- [7 Sample Code](#)

### Introduction

This RS-485 Sensor Node module can be used in various applications such as intelligent agriculture, environment monitoring, home automation...etc. It is a Internet of things device. This RS-485 Sensor node provides 6 channel analog input and 1 SHT1x Humidity & Temperature digital input. The RS485 protocol supports up to 254 nodes at 1200m distance between each node. This allows a wide range cover for monitoring the environment. A screw free design allows easy connection of cables without scarifice reliable & stable.

RS-485 standard is used effectively over long distances and in electrically noisy environments. Multiple receivers may be connected to such a network in a linear, multi-drop configuration. These characteristics make such networks useful in industrial environments and similar applications. RS-485 enables the configuration of inexpensive local networks and multidrop communications links. It offers data transmission speeds of 35 Mbit/s up to 10 m and 100 kbit/s at 1200 m.

RS-485 bus is the most popular communication method in industry. Compared with RS-232 bus, it is able to transfer information in further distance with lower cost. Establishing an "Internet of Things" by integrating RS-485 with Ethernet, which is the widely available in the world, in the hope that this method will coordinate all devices with low cost and high efficiency.

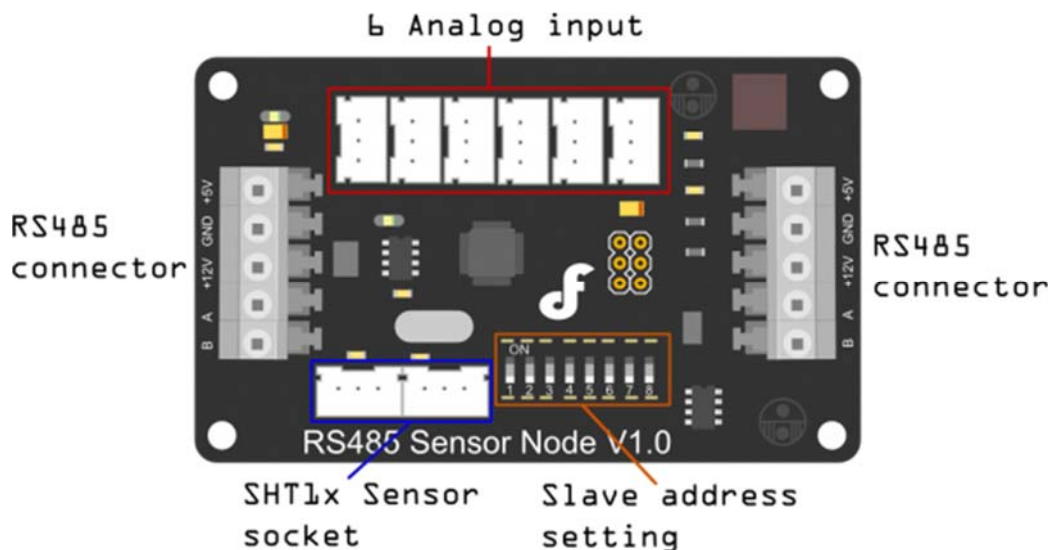
## Application

- Intelligent agriculture
- Public safety
- Environment monitoring
- Individual health
- Home automation

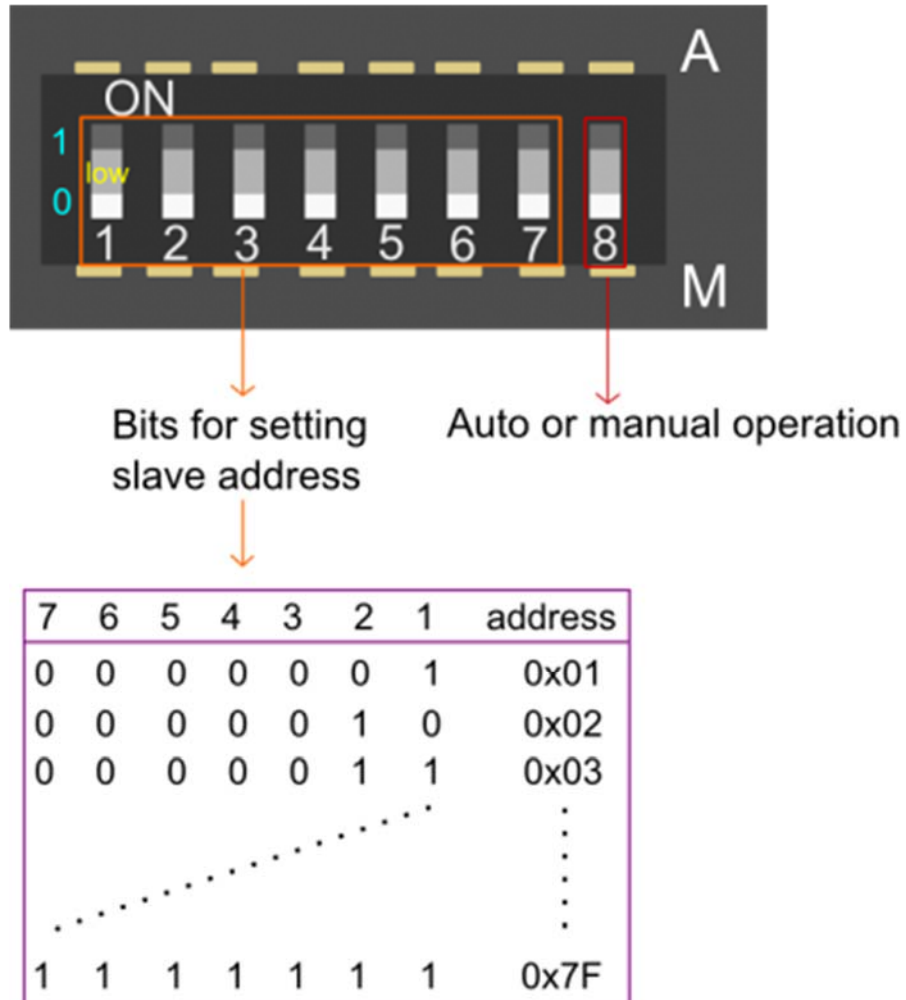
## Specification

- MCU:Atmega8
- Input Voltage:12V
- Baud rate:9600
- Slave address: 0x01 - 0x7F
- RS-485 long distance, reliable & stable communication
- Multiple connection (up to 127 modules)
- Press to connect cables, quick and easy
- 1 SHT1x Humidity and Temperature Sensor interface
- 6 channel analog sensor interface
- 8 small switches for setting slave address directly
- Humidity:0-100%RH( $\pm 4.5\%$ RH)
- Temperature:-40-128.8°C( $\pm 0.5^\circ\text{C}$ )
- Size:82x50mm

## Pin Out



More details



Details for slave address setting

**Auto or manual operation:** This bit is for setting the addresses of slave devices by software or hardware.

- A: set slave address via software, when it is at the A side, the Bits for setting slave address will be invalid.
- M: set slave address via hardware, when it is at the M side means you can set the slave address via the 7 small switches. Success to set after 30 seconds.

**Bits for setting address of slave:** 0x01~0x7F, just be effective to M side

Product Directive

- Check all the current real-time data directive--0x21

**Command:**

|           |      |                |              |              |          |
|-----------|------|----------------|--------------|--------------|----------|
| Word Head |      | Device Address | Frame Length | Command Word | Checksum |
| 0x55      | 0xAA | 0x11           | 0x00         | 0x21         | SUM      |

This command will check all data of current device. There are 10 register data totally. A register data is 16 digit, consisting of 8 high digit and 8 low digit.

**Return the following:**

| Content Order | Register Illustration                     | Register Data | Register Range                |
|---------------|---|---------------|-------------------------------|
| 1             | manually/automatically set address status | 0x00 0x01     | 1 for automatic; 0 for manual |
| 2             | Humidity Measurement                      | 0x00 0x02     | 0.0 to 100.0%(RH)             |
| 3             | Temperature Measurement                   | 0x00 0x03     | -40.0 to 128.0(°C)            |
| 4             | SHT1X error status                        | 0x00 0x04     | 1 for error; 0 for normal     |
| 5             | Analog Measurement 1                      | 0x00 0x05     | 0 to 1023                     |
| 6             | Analog Measurement 2                      | 0x00 0x06     | 0 to 1023                     |
| 7             | Analog Measurement 3                      | 0x00 0x07     | 0 to 1023                     |
| 8             | Analog Measurement 4                      | 0x00 0x08     | 0 to 1023                     |
| 9             | Analog Measurement 5                      | 0x00 0x09     | 0 to 1023                     |
| 10            | Analog Measurement 6                      | 0x00 0x10     | 0 to 1023                     |

**Instruction Description:** content value consists of 2 byte; 0.0 to 100.0 degree stand for 0 to 1000; -40.0 to 128.0 degree stand for -400 to 1280

**Return Value:**

|           |      |                |              |              |         |   |          |
|-----------|------|----------------|--------------|--------------|---------|---|----------|
| Word Head |      | Device Address | Frame Length | Command Word | Content |   | Checksum |
| 0x55      | 0xAA | 0x11           | 0x14         | 0x21         | H       | T | SUM      |

**Sample:**

Send instruction:

| Word Head |      | Device Address | Frame Length | Command Word | Checksum |
|-----------|------|----------------|--------------|--------------|----------|
| 0x55      | 0xAA | 0x11           | 0x00         | 0x21         | 0x55     |

Return instruction:

| Word Head |           | Device Address | Frame Length | Command Word | Content1  | Content2  | Content3  |
|-----------|-----------|----------------|--------------|--------------|-----------|-----------|-----------|
| 0x55      | 0xAA      | 0x11           | 0x14         | 0x21         | 0x00 0x00 | 0x00 0x00 | 0x00 0x00 |
| Content4  | Content5  | Content6       | Content7     | Content8     | Content9  | Content10 | Checksum  |
| 0x00 0x01 | 0x00 0x66 | 0x00 0x99      | 0x00 0x66    | 0x00 0x99    | 0x00 0x66 | 0x00 0x99 | 0x46      |

- **Set address for the model --0x55**

**Command:**

| Word Head |      | Device Address | Frame Length | Command Word | Content | Checksum |
|-----------|------|----------------|--------------|--------------|---------|----------|
| 0x55      | 0xAA | 0xAB           | 0x01         | 0x55         | 0x22    | SUM      |

**Return Value:**

| Word Head |      | Device Address | Frame Length | Command Word | Content | Checksum |
|-----------|------|----------------|--------------|--------------|---------|----------|
| 0x55      | 0xAA | 0xAB           | 0x01         | 0x55         | 0x22    | SUM      |

**Instruction Description: 0xAB is broadcast address,that is, it is shared address of all models. Send 0x55 to address 0xAB in order to set model address in the uncertain model status**

According to new device address,model will return 0x55 after address set successfully; In manual status,sending 0x55 can't set current device address,if the product can set device address manually and automatically.Then, the return value is 0xFE that illustrates product in manual setting address status.

**Return Value:**

| Word Head |      | Device Address | Frame Length | Command Word | Content | Checksum |
|-----------|------|----------------|--------------|--------------|---------|----------|
| 0x55      | 0xAA | 0xAB           | 0x01         | 0x55         | 0xFF    | SUM      |

**Sample:**

Send instruction:

| Word Head |      | Device Address | Frame Length | Command Word | Content | Checksum |
|-----------|------|----------------|--------------|--------------|---------|----------|
| 0x55      | 0xAA | 0xAB           | 0x01         | 0x55         | 0x11    | 0x11     |

this sample is used for setting device address as 0x11.

In the status of setting address manually

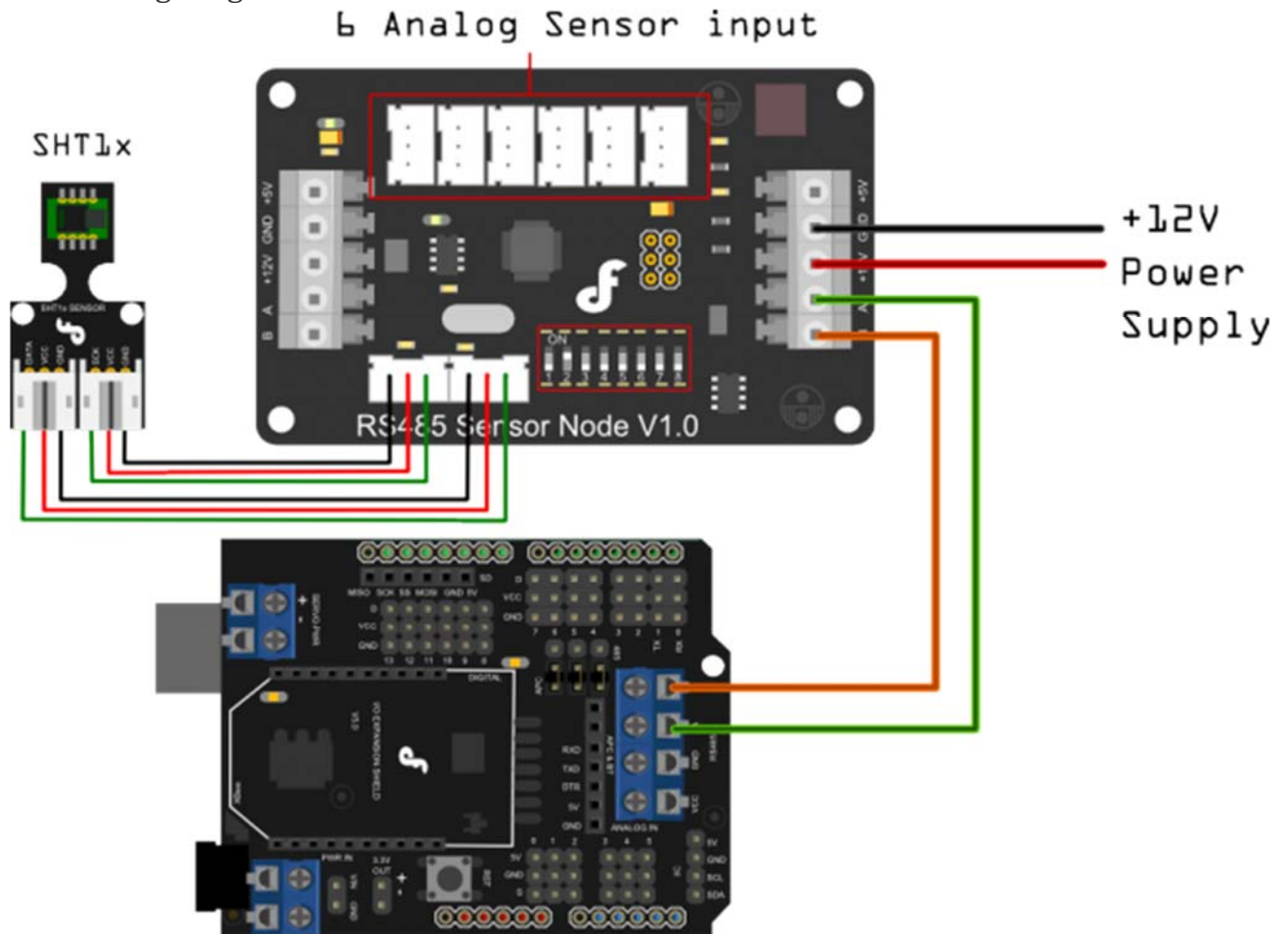
Send instruction:

| Word Head |      | Device Address | Frame Length | Command Word | Content | Checksum |
|-----------|------|----------------|--------------|--------------|---------|----------|
| 0x55      | 0xAA | 0xAB           | 0x01         | 0x55         | 0x11    | 0x11     |

Return instruction:

| Word Head |      | Device Address | Frame Length | Command Word | Content | Checksum |
|-----------|------|----------------|--------------|--------------|---------|----------|
| 0x55      | 0xAA | 0xAB           | 0x01         | 0x55         | 0xFF    | 0xFF     |

## Connecting Diagram



RS485 Sensor Node V1.0 connecting diagram

## Sample Code

```
/*  
# The Sample code for test the data of Analogue_Test and SHT1X Module  
  
# Editor : Lisper  
# Date   : 2013.12.9  
# Ver    : 1.3  
# Product: Analogue_Test and SHT1X Module  
# SKU    : DFR0233
```

```

# Description:
# Read the Analog value and the data of humidity & temperature

# Hardwares:
1. Arduino UNO
2. IO Expansion Shield V5
3. Analogue_Test and SHT1X Module

# Interface: RS485

# Note: Connect the Analogue_Test and SHT1X Module with IO Expansion Shield
V5 through RS485

Set the address of the module in manual,range from 0x02 to 0x7F,take effect
after 30 seconds
*/

#define uint    unsigned    int
#define uchar   unsigned    char
#define ulong   unsigned    long
#define addr    0x02        //set address of the device for 0x02
uchar cmd[50];
uchar receive_ACK[100];
int EN = 2;

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#define printByte(args) Serial.write(args)
#define printlnByte(args)  Serial.write(args),Serial.println()
#else
#include "WProgram.h"
#define printByte(args) Serial.print(args,BYTE)
#define printlnByte(args)  Serial.println(args,BYTE)
#endif

void setup() {

```



```

Serial.begin(9600);
pinMode(2, OUTPUT); // TTL -> RS485 chip driver pin
}

void loop() {
    static unsigned long timepoint = 0;
    if (millis() - timepoint > 1000) {
        read_command();
        timepoint = millis();
    }
    if (Serial.available() > 0) data_return();

    // delay(1000);

}

/*****Send command to Analogue_Test and SHT1X Module*****/
void read_command()
{
    int i;
    char sum_cmd = 0;
    digitalWrite(EN, HIGH); // Turn the driver pin to HIGH -> Turn on code transmitting mode for the RS485 interface
    // Turn the driver pin to LOW -> Turn on reading mode for the RS485 interface
    delay(10);
    cmd[0] = 0x55;
    cmd[1] = 0xaa;
    cmd[2] = addr;
    cmd[3] = 0x00;
    cmd[4] = 0x21;
    for (i = 0; i < 5; i++) {
        sum_cmd += cmd[i];
    }
}

```

```

cmd[5] = sum_cmd;

for (i = 0; i < 6; i++) {

    printByte(cmd[i]); // command send to device
#if defined(ARDUINO) && ARDUINO >= 100
    Serial.flush(); // complete the transmission of outgoing serial data
#endif
    delay(10);
}
digitalWrite(EN, LOW);
}

/*****Feedback data of the Analog value and humidity & temperature ****
*****/

void data_return()
{
    digitalWrite(EN, LOW); // Turn the driver pin to LOW -> Turn on reading mode for the RS485 interface
    delay(10);
    int i = 0;

    unsigned long timer = millis();

    while (true) {
        if (Serial.available()) {
            receive_ACK[i] = Serial.read();
            i++;
        }
        if (millis() - timer > 100) {
            break;
            Serial.println("Finish reading!");
        }
    }
}

```

```

    }
}
print_data () ;

/*****
// Display the original data

//      for(int j = 0; j < 26; j++){
//          Serial.print(receive_ACK[j],HEX); // return command
//          Serial.print(" ");
//      }
//      Serial.println(" ");
}

void show_0x21_command(void)
{
    sht1x_data();
    Analog_test_data();
}

/*****Deal with datas from Sht1x humidity & temperature sensor*****/

void sht1x_data()
{
    uint humidity;
    uint temperature;
    humidity = receive_ACK[7] * 256 + receive_ACK[8];
    temperature = receive_ACK[9] * 256 + receive_ACK[10];
    Serial.print("H:");
    Serial.print(humidity / 10, DEC);
    Serial.print(" ");

```

```

Serial.print("T:");
Serial.println(temperature / 10, DEC);
}

/*****Deal with datas from 6 Analog Sensors*****/
void Analog_test_data()
{
  char register_addr;
  uint Analog_data;
  register_addr = 13;
  Serial.print("Analog Value:");
  for (int n = 1; n < 7; n++) {
    Analog_data = receive_ACK[register_addr] * 256 + receive_ACK[register_addr + 1];
    register_addr = register_addr + 2;
    Serial.print(Analog_data, DEC);
    Serial.print(" ");
  }
  Serial.println(" ");
  delay(1000);
}

/***** by lisper *****/
//print humidity and temperature
void print_data () {
  if (checksum ()) { // if check sum is right
    Serial.println ();
    float humidity = read_uint8_t (receive_ACK, 7) / 10.0;
    Serial.print ("humidity=");
    Serial.println (humidity, 2);

    float temperature = (read_uint8_t (receive_ACK, 9) / 10.0);
    Serial.print ("temperature=");
    Serial.println (temperature, 2);
  }
}

```

```

}
else {
    Serial.print ("\ncheck sum error! sum=");
    Serial.println (getsum_add (receive_ACK, 25), HEX);
}
}

//if check sum is ok
boolean checksum () {
    uint8_t checksum = getsum_add (receive_ACK, 25);
    if (checksum == receive_ACK[25])
        return true;
    else
        return false;
}

//read 2 byte to uint16_t
uint16_t read_uint8_t (uint8_t *buffer, uint8_t sub) {           // Big-Endian,
first byte is high byte
    return ((uint16_t)(buffer[sub]) << 8) + buffer[sub + 1];
}

//get check sum, add from 0 to length-1
uint8_t getsum_add (uint8_t *buffer, uint8_t length) {
    uint8_t sum;
    for (int i = sum = 0; i < length; i++) {
        sum += buffer[i];
    }
    return sum;
}

/*****
***/

```