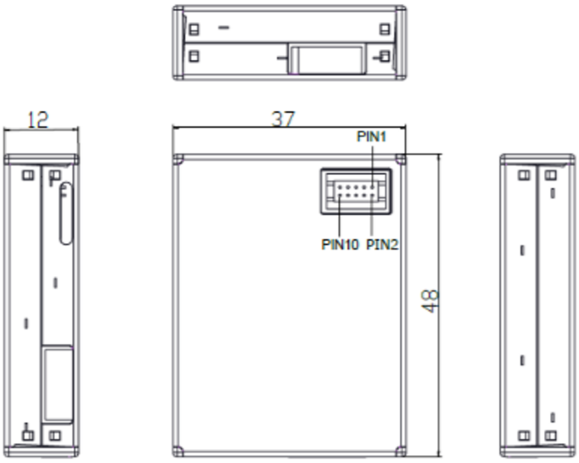
Plantower

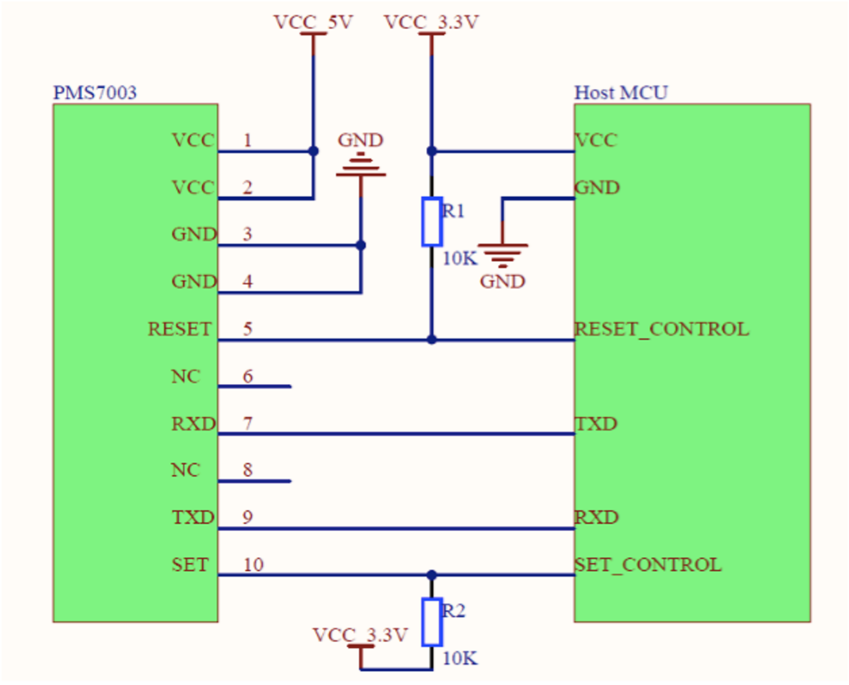
PMS-7003

Air Quality Sensor



Pin1	VCC
Pin2	VCC
Pin3	GND
Pin4	GND
Pin5	Reset
Pin6	N/C
Pin7	RX
Pin8	N/C
Pin9	TX
Pin10	Set

Measures:
0.3
0.5
1.0
2.5
5.0
10.0



PIN4	GND	Negative power supply
PIN5	RESET	Module reset signal / TTL level @ 3.3V, low reset
PIN6	NC	
PIN7	RX	Serial Receive Pin / TTL Level @ 3.3V
PIN8	NC	
PIN9	TX	Serial port pin / TTL level @ 3.3V
PIN10	SET	Set pin / TTL level @ 3.3V, high or floating for Normal working state, low level is dormant state

Typical circuit connection

Figure 3 Typical circuit connection diagram

Circuit design should be noted

1. PMS7003 requires 5V power supply, this is because the fan needs 5V drive. But other data communication and control Pins require 3.3V as a high level. So the host board with which the communication is connected should be powered by 3.3V. If the motherboard MCU is 5V power supply, then the communication line (RXD, TXD) and control line (SET, RESET) Should be added to the level conversion chip or circuit.

2. SET and RESET internal pull-up resistor, if not used, it should be vacant.

3. PIN6 and PIN8 for the program internal debugging, the application circuit should be vacant.

4. When applying the sleep function, note that the fan stops working when you sleep and the fan restart requires at least 30 Sec settling time, so to obtain accurate data, the sleep wake-up after the sensor working time should not be low In 30 seconds.

Typical output characteristics

Asymmetric unit: μ g / m^3 (PM2.5 mass concentration standard value, Appendix A data 2) abscissa unit: times

Annex **A: PMS7003** Transfer Protocol

Default baud rate: 9600bps Parity: None Stop bit: 1 bit
Total length of the protocol: 32 bytes

Starting character	0x42 (fixed)	
Start character 2	0x4d (fixed)	
Frame length is high octet	..	Frame length = 2x13 + 2 (data + check digit)
The frame length is eight bits long	..	
Data 1 high octet	...	* Data 1 indicates PM1.0 concentration (CF = 1, standard particles)
Data 1 low octet	...	Unit µ g / m3
Data 2 high octet	...	Data 2 indicates PM2.5 concentration (CF = 1, standard particulate matter)
Data 2 low octet	...	Unit µ g / m3
Data 3 high octet	...	Data 3 indicates PM10 concentration (CF = 1, standard particulate matter)
Data 3 low eight bits	...	Unit µ g / m3
Data 4 high octet	...	* Data 4 indicates PM1.0 concentration (in atmospheric environment)
Data 4 low octets	...	Unit µ g / m3
Data 5 high octet	...	Data 5 indicates PM2.5 concentration (in atmospheric environment)
Data 5 low octets	...	Unit µ g / m3
Data 6 high octet	...	Data 6 indicates PM10 concentration (in atmospheric environment)
Data 6 is low octet	...	Unit µ g / m3
Data 7 high octet	...	Data 7 indicates that 0.1 liter of air has a diameter above 0.3um
Data 7 is low octet	...	The number of particles
Data 8 high octet	...	Data 8 indicates that 0.1 liter of air has a diameter of 0.5um or more
Data 8 is low	...	The number of particles
Data 9 high octet	...	Data 9 indicates that 0.1 liter of air has a diameter of 1.0um or more
Data 9 is low octet	...	The number of particles
Data 10 high octet	...	Data 10 indicates that the diameter of 0.1 liter of air is above 2.5um
Data 10 low octets	...	The number of particles
Data 11 High octet	...	Data 11 indicates that 0.1 liter of air has a diameter of 5.0um or more
Data 11 is low octet	...	The number of particles

Data 12 high octet ...	Data 12 indicates that 0.1 liter of air has a diameter above 10um
Data 12 is low octet ...	The number of particles
Data 13 high octet ...	version number
Data 13 low octets ...	error code
Data and check high eight ...	Check code = start character 1 + start character 2 + ... + data 13 low
Data and check low eight ...	Eight

Note: The standard particle mass concentration value refers to the use of industrial metal particles as equivalent particles for density conversion

To the mass concentration value, suitable for industrial production workshop and other environments.
The mass concentration of atmospheric particulate matter is empty
The main pollutants in the gas are equivalent particles for density conversion, suitable for ordinary indoor and outdoor atmosphere.

B: Sensor Slave Extended Instruction Protocol

1. Host communication protocol format

Feature Byte 1	Feature Byte 2	Instruction Byte	Status byte 1	Status byte 2	Check byte 1	Check byte 2
0x42	0x4d	CMD	DATAH	DATAL	LRCH	LRCL

2. Instruction and feature byte definition

CMD	DATAH	DATAL	Description
0xe2	X	X	Passive reading
0xe1	X	00H- Passive 01H- active	State switch
0xe4	X	00H standby mode 01H normal mode	Standby control

3. Command response:

0xe2: Acknowledgment 32 bytes, with the sensor specification protocol.

4. Check word generation

All bytes are summed from the feature word

```
//------------------------------------------------------------------
------------
//  PM sensor PMS7003 (fine dust)

/*

Copyright 2017 Scapeler

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
See the License for the specific language governing permissions and
limitations under the License.

*/

//------------------------------------------------------------------
------------

#include <SoftwareSerial.h>

SoftwareSerial Serial1(10, 11); // serial ports RX, TX

// input byte variables
int inputHigh = 0;
int inputLow = 0;
// variable to caclulate checksum input variables
uint16_t inputChecksum = 0;
// sensor variables
uint16_t concPM1_0_CF1;
uint16_t concPM2_5_CF1;
uint16_t concPM10_0_CF1;
uint16_t concPM1_0_amb;
uint16_t concPM2_5_amb;
uint16_t concPM10_0_amb;
uint16_t rawGt0_3um;
uint16_t rawGt0_5um;
uint16_t rawGt1_0um;
uint16_t rawGt2_5um;
uint16_t rawGt5_0um;
uint16_t rawGt10_0um;
uint8_t  version;
uint8_t  errorCode;
uint16_t checksum;

void setup() {
    Serial.begin(9600);
    while (!Serial) {
    }
    Serial.println("Serial port ready");
    Serial1.begin(9600);
    while (!Serial1) {
    }
    while (Serial1.read()!=-1) {};  //clear buffer
    Serial.println("Sensor port ready");
}

bool pms7003ReadData() {

//    while (Serial1.read()!=-1) {};  //clear buffer

    if (Serial1.available() < 32) {
      if (Serial1.available() == 0) {
        delay(150);
        return;
      };
      if (Serial1.available() > 16) {
        delay(10);
        return;
      };
      if (Serial1.available() > 0) {
        delay(30);
        return;
      };
      delay(100);
      return;
    }
    if (Serial1.read() != 0x42) return;
    if (Serial1.read() != 0x4D) return;

    inputChecksum = 0x42 + 0x4D;

    inputHigh = Serial1.read();
    inputLow = Serial1.read();
    inputChecksum += inputHigh + inputLow;
    if (inputHigh != 0x00) return;
    if (inputLow != 0x1c) return;

    inputHigh = Serial1.read();
    inputLow = Serial1.read();
    inputChecksum += inputHigh + inputLow;
    concPM1_0_CF1 = inputLow+(inputHigh<<8);

    inputHigh = Serial1.read();
    inputLow = Serial1.read();
    inputChecksum += inputHigh + inputLow;
    concPM2_5_CF1 = inputLow+(inputHigh<<8);

    inputHigh = Serial1.read();
    inputLow = Serial1.read();
```

```
inputChecksum += inputHigh + inputLow;
concPM10_0_CF1 = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
concPM1_0_amb = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
concPM2_5_amb = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
concPM10_0_amb = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
rawGt0_3um = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
rawGt0_5um = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
rawGt1_0um = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
rawGt2_5um = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
rawGt5_0um = inputLow+(inputHigh<<8);

inputHigh = Serial1.read();
inputLow = Serial1.read();
inputChecksum += inputHigh + inputLow;
rawGt10_0um = inputLow+(inputHigh<<8);

inputLow = Serial1.read();
inputChecksum += inputLow;
version = inputLow;

inputLow = Serial1.read();
inputChecksum += inputLow;
```

```
    }

    errorCode = inputLow;

    Serial.print("PMS7003;");
    Serial.print(concPM1_0_CF1);
    Serial.print(';');
    Serial.print(concPM2_5_CF1);
    Serial.print(';');
    Serial.print(concPM10_0_CF1);
    Serial.print(';');
    Serial.print(concPM1_0_amb);
    Serial.print(';');
    Serial.print(concPM2_5_amb);
    Serial.print(';');
    Serial.print(concPM10_0_amb);
    Serial.print(';');
    Serial.print(rawGt0_3um);
    Serial.print(';');
    Serial.print(rawGt0_5um);
    Serial.print(';');
    Serial.print(rawGt1_0um);
    Serial.print(';');
    Serial.print(rawGt2_5um);
    Serial.print(';');
    Serial.print(rawGt5_0um);
    Serial.print(';');
    Serial.print(rawGt10_0um);
    Serial.print(';');
    Serial.print(version);
    Serial.print(';');
    Serial.print(errorCode);

    inputHigh = Serial1.read();
    inputLow = Serial1.read();
    checksum = inputLow+(inputHigh<<8);
    if (checksum != inputChecksum) {
      Serial.print(';');
      Serial.print(checksum);
      Serial.print(';');
      Serial.print(inputChecksum);
    }
    Serial.print('\n');

    delay(700);  // higher will get you checksum errors

    return;
}


void loop () {
    pms7003ReadData();
}
```