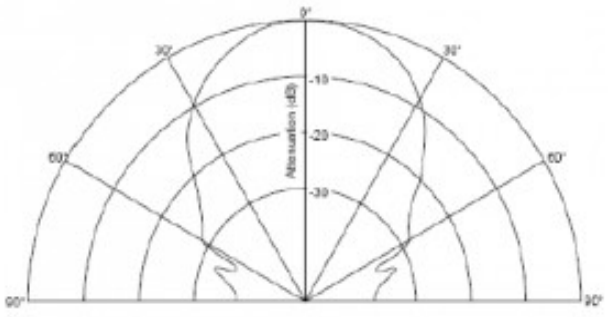
RB-Dfr-12
DFRobot URM04 v2.0 Ultrasonic Sensor

URM04 is developed based upon our popular URM37 ultrasonic sensor. The RS485 interface allows a number of sensors working together. Up to 32 URM04 may be connected together in a network.

Specifications

- Power: +5V
- Current: <20mA
- Working temperature: -10℃～+70℃
- Detecting range: 4cm-500cm
- Resolution: 1cm
- Frequency: 40KHz
- Interface: RS485
- Units: Range reported in cm
- Temperature sensor: 12 bits reading from serial port
- Size: 34mm × 51 mm
- Weight: 30g
- Default Address:0x11
- **D**efault Baudrate:19200

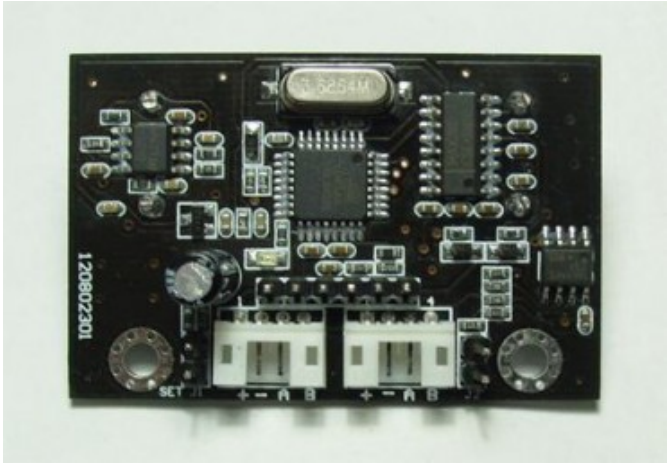


URM04 Beam Width 60 degree

Dimension and Pin definition

URM4 V2 front view and dimension

URM4 V2 Back view



RS485 Interface: Two connectors, + ： +5V DC Power +5V - ： GND Ground A ： A RS485 A(+) B ： B RS485 B(-) ISP Pin：For factory firmware uploading

Communication LED: As the device is powered up, this LED will flash four times which indicates that the sensor is working properly. This LED will also flash when it is communcating with other devices. Jumper A：Not in use Jumper B：When the sensor is working under a network, only the Jumper B for the first Device and the last Device need to be bridged.

Communication Protocols

- The device is fixed at 19200 bps Baud Rate,8/N/1.
- Note:The previous version has 115200 bps Baud Rate,8/N/1.

Set Device Address

Command:

Header		Address	Length	Cmd	Set Address	SUM
55	aa	AB	1	55	ADD	SUM

Return Value：

Header		Address	Length	Cmd	Flag	SUM
55	aa	ADD	1	55	S	SUM

PS: The address of each device can be changed when multiple devices are connected. The new Address must be between 0x11 and 0x80. If the address is set sucessfully, the flag will be set to 0x01 in the return data. If unsucessful, there is no return data. (The default address for the sensor is 0x12)

Note: The previous default address is 0x11.

Example:

Command: 0x55 0xaa 0x11 0x01 0x55 0x12 0x79 (Set Address to 0x12)
Return: 0x55 0xaa 0x12 0x01 0x55 0x01 0x69 (Address set sucessfully)

Read tempeature

Command:

Header		Address	Length	Cmd	SUM
55	aa	ADD	0	03	SUM

Header		Address	Length	Cmd	High Byte	Low Byte	SUM
55	aa	ADD	2	03	H	L	SUM

PS: The command will return the temperature reading. The return temperature reading is using Celsius scale. If the temperature is above 0 Celsius, the first four bits of High will be all 0. If the temperature is below 0 Celsius, the first four bits of High will be all 1. The last 4 bits of High together with the Low bits stands for 12bits temperature. The resolution is 0.1. When the reading is invalid, it returns 0xFF 0xFF

Example:
Command:

0x55 0xaa 0x11 0x00 0x03 0x13(SUM)

Return:

0x55 0xaa 0x11 0x02 0x03 0xF0 0x0A 0x11 (+1 Celsius Degree)
0x55 0xaa 0x11 0x02 0x03 0x00 0x0A 0x20 (-1 Celsius Degree)
0x55 0xaa 0x11 0x02 0x03 0xFF 0xFF 0x20 (Out of Range)

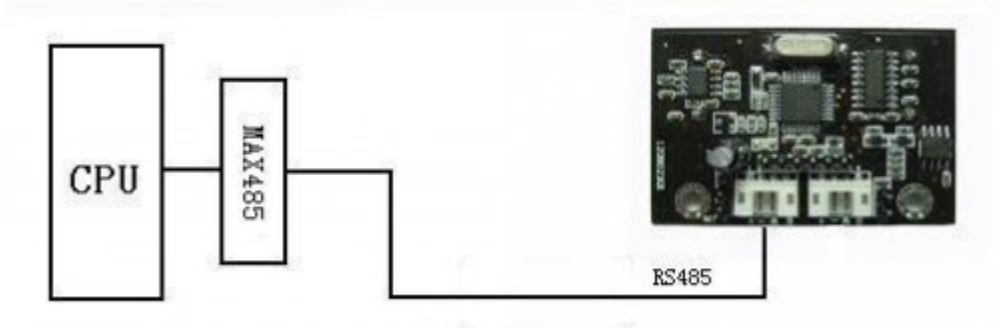
Function to calculate the temperature:

```
IF(HightByte>=0xF0)
{
Temperature= ((HightByte-0xF0)*256-LowByte)/10
}
Else
{
Temperature= ((HightByte)*256-LowByte)/10
}
```

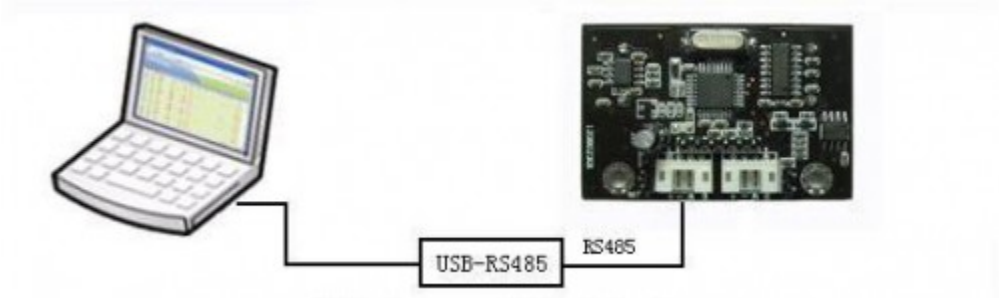
Sensor Connection Diagram

As the sensor uses RS485 interface which can not be connected directly to the MCU, a MAX485 chip will bridge the TTL interface to RS485.

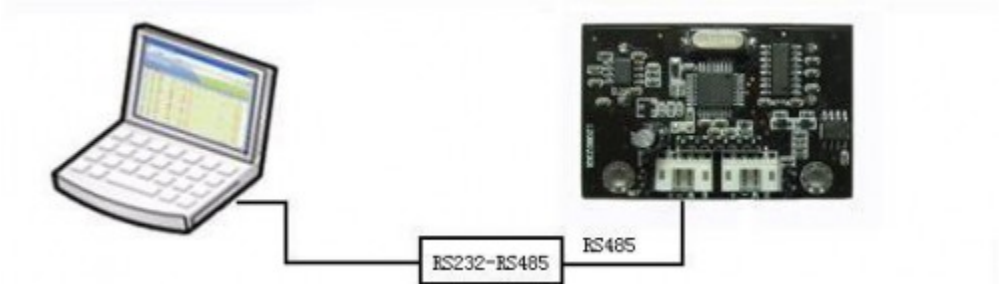
For PC users, either a USB-RS485 or RS232-RS485 converter will bridge the gap.



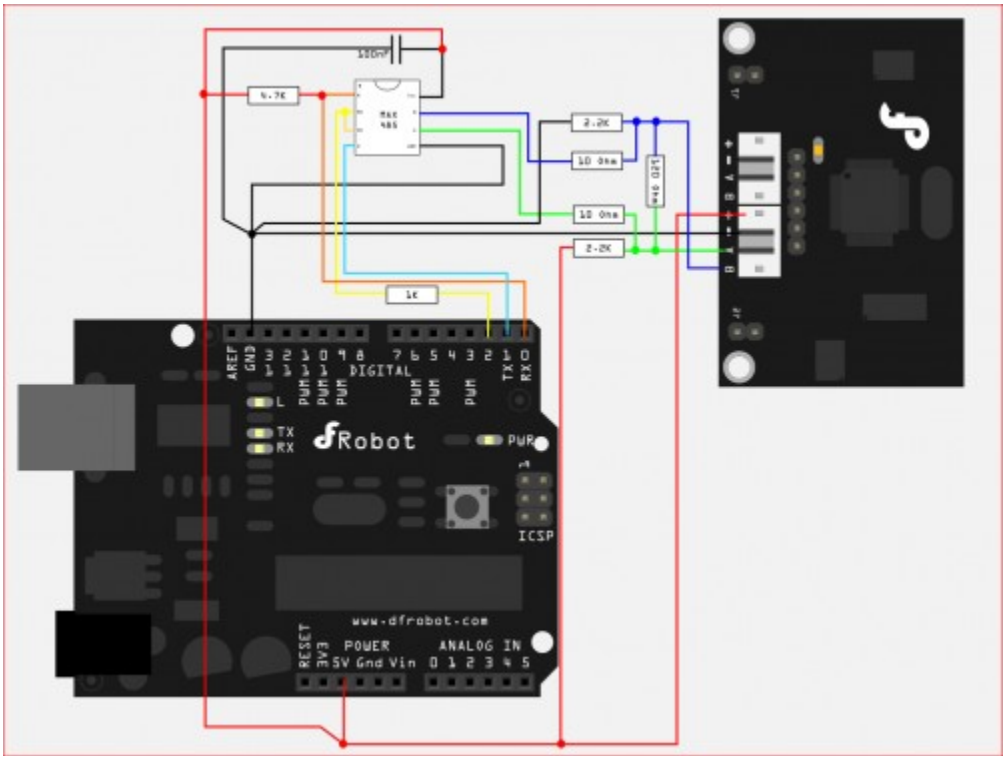
Connect Sensor to MCU via MAX485



Connect Sensor to PC via USB-RS485 converter



Connect Sensor to PC via RS232-RS485 Converter



Connect Sensor to Arduino Via MAX485 IC

Sensor Networking

Upto 32 URM04 sensors are able to join a network. Simply serially connect the sensors uses twisted pair cables. A diagram is illustrated:

URM04 Networking

Arduino sketch for driving one URM04 sensor

The sketch code:

```
/*
# The Sample code for driving single URM04 measuring distance function

# Editor : Lauren
# Date   : 2012.2.8
# Ver    : 0.3
# Product: URM04 Ultrasonic sensor

# Specification
    * Detecting range: 4cm-500cm
    * Resolution     : 1cm
    * Interface      : RS485
    * Units: Range reported in cm
    * Temperature sensor: 12 bits reading from serial port


# Description:

# finish driving single URM function
# if use the IO expansion shield to drive the urm sensors, the measuring rate may be 20Hz or slower[if you want].
# The sample code is compatible with the Arduino IDE 1.0 and also the earlier version.

*/

#include "Urm4parser.h"

void setup(){
  urmInit();      // Init the URM04 sensor
}

void loop(){
```

```
  static unsigned long timePoint = 0;

  runUrm4();      // Drive URM04 Sensor and transmit the protocol to the sensor via RS485 interface
                  // (IO Expansion shield V5 for arduino)
  decodeURM4();   // Read and get the distance value from the sensor
  if(millis() - timePoint > 100){
    PrintData();  // print the data
    timePoint = millis();
  }
//  PrintData();
//  delay(100);

}

void PrintData(){

  Serial.print("Distance value:  ");
  for(int i = 0; i < urmAccount; i++){
    Serial.print(urmData[i]);
    Serial.print(" ");
  }
/*
  for(int i = 0; i < urmAccount;i ++){
    Serial.print(urmID[i],HEX);
    Serial.print(" ");
  }
*/
  Serial.println();
}
```

The library code: please place the library file Urm4parser.h in to the sketch folder.

```
/*
# The library for the URM04 sketch

# Editor : Lauren
# Date   : 2012.2.8
# Ver    : 0.3
# Product: URM04 Ultrasonic sensor

# Specification
    * Detecting range: 4cm-500cm
    * Resolution     : 1cm
    * Interface      : RS485
    * Units: Range reported in cm
    * Temperature sensor: 12 bits reading from serial port


# Description:
```

```
# finish driving single URM function
# if use the IO expansion shield to drive the urm sensors, the measuring rate may be 20Hz or slower[if
you want].
# The sample code is compatible with the Arduino IDE 1.0 and also the earlier version.

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

#define SerialPort Serial
#define urmAccount 1   // Init the account of the URM04 sensor
#define CommMAXRetry 40
#define TriggerPin 2

/******************** Variables ***************/

byte startAddr = 0x11; // Init the URM04 485 Address -- URM04 default address = 0x11

byte readingStep;
byte cmdst[10];
int urmID[urmAccount];
unsigned int urmData[urmAccount];
unsigned long managerTimer = 20;

/******************** Functions ***************/
void urmInit();
void runUrm4();
void urmTrigger(int id);
void urmReader(int id);
void transmitCommands();
void decodeURM4();
void analyzeUrmData(byte cmd[]);

/****************** Init sensor ***************/

void urmInit(){

  pinMode(TriggerPin,OUTPUT);  // TTL -> RS485 chip driver pin
  digitalWrite(TriggerPin,LOW);// Turn the driver pin to LOW  -> Turn on reading mode for the RS485
interface
                        // Turn the drvier pin to HIGH -> Turn on code transmitting mode for the RS485
interface
  readingStep = 0;
```

```
  startAddr = 0x11;
  managerTimer = millis();
  for(int i = 0 ;i < urmAccount; i ++){  //Init the URM04 command receiving address
    urmID[i] = startAddr + i;
    urmData[i] = 0;
  }
  SerialPort.begin(19200);              // Init the RS485 interface
                                  // Also when you are driving the URM04, you could open serial monitor to
                                  // tracing the steps and data feedback from URM04
  SerialPort.println();
  SerialPort.print("The URM ID: 0x");
  for(int i = 0 ;i < urmAccount; i ++){  //Init the URM04 command receiving address
    SerialPort.print(urmID[i],HEX);
    SerialPort.print(" ");
  }
  SerialPort.println(" ");
  SerialPort.println("The default baudrate: 19200");
  SerialPort.println("Start drive the sensors!");
  for(int i = 0 ;i < 10; i++)  cmdst[i] = 0;  //init the URM04 protocol
  cmdst[0]=0x55;
  cmdst[1]=0xaa;
}

/********************* Drive URM04 and get the data code ************************/

void runUrm4(){                 // You could adjust the sensor measuring rate by changing the
managerTimer value

  static unsigned long timer = 0;
  static int num = 0;           //Set the URM04 id to be drived

  if(millis() - timer > managerTimer){
    digitalWrite(TriggerPin, HIGH);  //Turn on transmitting mode for the RS485 interface

    switch(readingStep){

    case 0:
      urmTrigger(urmID[num]);
      managerTimer = 40;            //set a interval after trigger the measuring

      break;

    case 1:
      urmReader(urmID[num]);
      managerTimer = 0;             //set a interval after transmitting the reading distance command

      break;

    case 2:
```

```
      digitalWrite(TriggerPin, LOW); //Turn on reading mode for the RS485 interface
      managerTimer = 10;
      break;

    default:
      readingStep = 0;              // Finish reading the distance and start a new measuring for the sensor
      break;
    }

    if(readingStep < 2)  readingStep++;  //step manager
    else readingStep = 0;

    timer = millis();
  }

}

/********************** Transmit Command via the RS485 interface ***************/

void urmTrigger(int id){  // The function is used to trigger the measuring
  cmdst[2] = id;
  cmdst[3] = 0x00;
  cmdst[4] = 0x01;
  transmitCommands();
  //  SerialPort.println("Trigger!");
}
void urmReader(int id){  // The function is used to read the distance
  cmdst[2] = id;
  cmdst[3]=0x00;
  cmdst[4]=0x02;
  transmitCommands();
  //  SerialPort.println("Ask for distance!");
}

void transmitCommands(){  // Send protocol via RS485 interface
  cmdst[5]=cmdst[0]+cmdst[1]+cmdst[2]+cmdst[3]+cmdst[4];
  delay(1);
  for(int j = 0; j < 6; j++){
    printByte(cmdst[j]);
//    delayMicroseconds(10);
  }
  delay(2);
}


/********************** Receive the data and get the distance value from the RS485 interface
***************/

void decodeURM4(){
```

```
  if(SerialPort.available()){

    unsigned long timerPoint = millis();

    int RetryCounter = 0;
    byte cmdrd[10];
    for(int i = 0 ;i < 10; i++)  cmdrd[i] = 0;
    int i=0;

//    SerialPort.println("OK");

    boolean flag = true;
    boolean valid = false;
    byte headerNo = 0;

    while(RetryCounter < CommMAXRetry && flag)
    {

      if(SerialPort.available()){
        cmdrd[i]= SerialPort.read();

//        printByte(cmdrd[i]);

        if(i > 7){
          flag=false;
//          printByte(0xEE);
//          printByte(0xFF);
          SerialPort.flush();
          break;
        }
        if(cmdrd[i] == 0xAA){
          headerNo = i;
          valid = true;
        }
        if(valid && i == headerNo + 6){
//          printByte(0xDD);
//          printByte(0xFF);
          flag = false;
          break;
        }
        i ++;
        RetryCounter = 0;
      }
      else{
        RetryCounter++;
        delayMicroseconds(15);
      }
    }
```

```
//    printByte(millis() - timerPoint);

    if(valid)  analyzeUrmData(cmdrd);
//    else SerialPort.println("Invalid feedback");  //Get an invalid error command

  }

}

void analyzeUrmData(byte cmd[]){

  byte sumCheck = 0;
  for(int h = 0;h < 7; h ++)  sumCheck += cmd[h];

  if(sumCheck == cmd[7] && cmd[3] == 2 && cmd[4] == 2){

    byte id = cmd[2] - startAddr;
    urmData[id] = cmd[5] * 256 + cmd[6];

//    SerialPort.print(id);
//    SerialPort.print(":");
//    SerialPort.println(urmData[id]);

  }
  else if(cmd[3] == 2 && cmd[4] == 2){
    SerialPort.print("Sum error");
  }

}
```