SKU:SEN0358 (https://www.dfrobot.com/product-2173.html)

(https://www.dfrobot.com/product-2173.html)

Introduction

This is 200KHz high-frequency ultrasonic ranging sensor with IP65 protection grade. The sensor uses the RS485 interface that follows the standard Modbus-RTU communication protocol, featuring reliable communication. The sensor's slave address and serial port parameter can be revised according to the actual use, so it can be conveniently used with all kinds of industrial controlling machines. In addition, the sensor comes with flexible temperature compensation function that allows users to use the sensor built-in temperature compensation or external temperature compensation according to their needs, which makes it applicable to various complicated application scenarios.

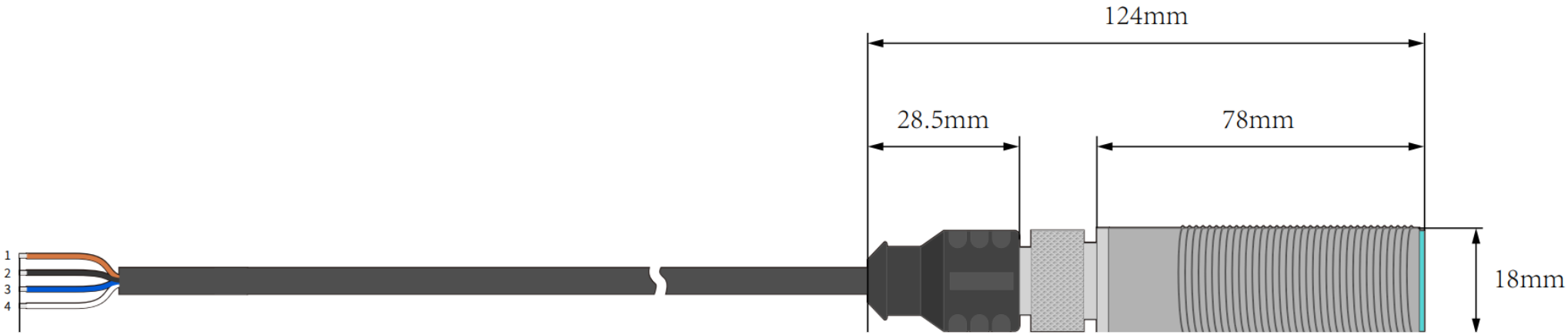
Besides that, the URM14 adopts high-requency and stable ultrasonic transducer that offers 100mm~1500mm effective measuring range for flat reflecting wall, 1mm accuracy, and ±1% error. Compared with the general ultrasonic ranging sensor working at 40 KHZ acoustic frequency, this ranging sensor has higher measurement accuracy and shorter measurement period, which makes it very suitable for high precision and fast response ranging applications. In addition, the sensor has built-in noise level evaluation function, which can realize adaptive ranging in complex scenes. Users can also obtain noise level to optimize the working conditions of the sensor, so as to obtain the highest precision measurement data.

Specification

- Operating Voltage: DC 7-15V
- Max Instantaneous Current: 350mA
- Effective Measuring Range: 100-1500mm

- Effective Measuring Range: 100~1500mm
- Distance Resolution: 0.1mm
- Measuring Accuracy: 1mm
- Distance Error: ±0.1%
- Temperature Resolution: 0.1℃
- Temperature Error: ±1℃
- Measuring Frequency: 30HZ
- Operating Temperature: -20℃ ~ + 80℃
- Sensor Acoustic Frequency: 200KHz±4%
- Directional Angle: 12°±2°(-6dB)
- Protection Grade: IP65
- Communication interface: RS485

Board Overview



Module Wire Connecting Order:

- Orange-----VCC
- Black------GND
- Blue------RS485-B
- White-----RS485-A

Register

Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
0x00	1	Module PID Register	R	0x0000-0xFFFF	0x0002	Product check (Detect module type)
0x01	1	Module VID Register	R	0x0000-0xFFFF	0x0010	Version check (0x0010 represents V0.0.1.0)
0x02	1	Module Address Register	R/W	0x0001-0x00F7	0x000C	When the sensor address is unknown, write to the register through the broadcast address 0x00, at this time, the sensor will not have data output **Save when powered off, take effect after restarting**
0x03	1	Serial parameter control register 1	R/W	0x0000-0xFFFF	0x0005	Module Baudrate: 0x0001---2400 0x0003---9600 0x0004---14400 0x0005---19200 0x0006---38400 0x0007---57600 0x0008---115200

Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
						Other----115200 **Save when powered off, take effect after restarting**
0x04	1	Serial parameter control register 2	R/W	0x0000-0xFFFF	0x0001	Module check bit H: Stop bit L: 0x00---None 0x00---0.5byte 0x01---Even 0x01---1byte 0x02---Odd 0x02---1.5byte other---none 0x03---2byte Other---1byte **Save when powered off, take effect after restarting**
0x05	1	Distance register	R	0x0000-0xFFFF	0xFFFF	The distance value LSB measured by the module represents 0.1mm
0x06	1	Onboard temperature data register	R	0x0000-0xFFFF	0x0000	The temperature value LSB measured by the onboard temperature sensor represents 0.1℃ (with unit symbol)
0x07	1	External temperature compensation data register	R/W	0x0000-0xFFFF	0x0000	Write ambient temperature data to this register for external temperature compensation, LSB represents 0.1℃ (with unit symbol)

Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
0x08	1	Control register	R/W	0x0000-0xFFFF	0x0004	bit0: 0-use onboard temperature compensation function 1-use external temperature compensation function(Users need to write temperature data to external temperature compensation data register) bit1: 0-enable temperature compensation function 1-disable temperature compensation function bit2: 0-auto detection 1-passive detection bit3: In passive detection mode, write 1 to this bit, then it will measure distance once. The distance value can be read from distance register about 30ms later. This bit is reserved in passive detection mode. This bit will be auto cleared when set to 1 **Save when powered off, take effect after restarting**
0x09	1	Electrical noise level register	R	0x0000-0x0A	0x0000	0x0000-0x000A corresponds to noise level 0 to 10. This parameter can show the effect of power supply and environment on the sensor. The smaller the noise level, the more accurate the distance value detected by the

Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
						sensor.

SEN0358 Register Read/Write Tutorial

Requirements

- **Hardware**
  - Arduino Leonardo (https://www.dfrobot.com/product-832.html) x 1 (RS485 to TTL needs to occupy one serial port, so we recommend you to use device with more than 2 serial ports. Since Arduino Modbust takes up a lot of memory, it is suggested to use Arduino Mega2560 controller.)
  - RS485 Shield for Arduino (dfrobot.com/product-1024.html) x1
  - DC Power Supply x1
  - USB Data Cable (Connect the Arduino board to a computer via the USB cable)
- **Software**
  - Arduino IDE (https://www.arduino.cc/en/Main/Software)
  - Open Library Manager(Ctrl+Shift+I) in Arduino IDE, find and install ArduinoModbus and ArduinoRS485 Libraries.

Connection Diagram

Read the detected distance

```
/*****************************************************************************************************************
     This code tests the range finder function of the URM14 ultrasonic sensor
     @ author : roker.wang@dfrobot.com
     @ data   : 11.08.2020
     @ version: 1.0
*****************************************************************************************************************/
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>

#define   SLAVE_ADDR               ((uint16_t)0x0C)

#define   TEMP_CPT_SEL_BIT         ((uint16_t)0x01)
#define   TEMP_CPT_ENABLE_BIT      ((uint16_t)0x01 << 1)
#define   MEASURE_MODE_BIT         ((uint16_t)0x01 << 2)
#define   MEASURE_TRIG_BIT         ((uint16_t)0x01 << 3)


typedef enum{
  ePid,
  eVid,
  eAddr,
  eComBaudrate,
  eComParityStop,
  eDistance,
  eInternalTempreture,
  eExternTempreture,
  eControl,
  eNoise
}eRegIndex_t;//Sensor register index

/*
 *@brief Read data from holding register of client
```

```
 *@brief Read data from holding register of client
 *
 *@param addr :  Address of Client
 *@param reg: Reg index
 *@return data if execute successfully, false oxffff.

 */
uint16_t readData(uint16_t addr, eRegIndex_t reg)
{
  uint16_t data;
  if (!ModbusRTUClient.requestFrom(addr, HOLDING_REGISTERS, reg, 1)){
    Serial.print("failed to read registers! ");
    Serial.println(ModbusRTUClient.lastError());
    data = 0xffff;
  }else{
    data =  ModbusRTUClient.read();
  }
  return data;
}

/*
 *@brief write data to holding register of client
 *
 *@param addr :  Address of Client
 *@param reg: Reg index
 *@param data: The data to be written
 *@return 1 if execute successfully, false 0.
 */
uint16_t writeData(uint16_t addr, eRegIndex_t reg, uint16_t data)
{
  if (!ModbusRTUClient.holdingRegisterWrite(addr, reg, data)){
    Serial.print("Failed to write coil! ");
    Serial.println(ModbusRTUClient.lastError());
    return 0;
  }else
    return 1;
}
```

```
float  dist;
volatile uint16_t cr = 0;
void setup() {
  ModbusRTUClient.begin(19200);
  Serial.begin(9600);

  cr |= MEASURE_MODE_BIT;//Set bit2 , Set to trigger mode
  cr &= ~(uint16_t)TEMP_CPT_SEL_BIT;//Select internal temperature compensation
  cr &= ~(uint16_t)TEMP_CPT_ENABLE_BIT;//enable temperature compensation
  writeData(SLAVE_ADDR, eControl, cr); //Writes the setting value to the control register
  delay(100);
}

void loop() {
  cr |= MEASURE_TRIG_BIT;//Set trig bit
  writeData(SLAVE_ADDR, eControl, cr); //Write the value to the control register and trigger a ranging
  delay(300);//Delay of 300ms(minimum delay should be greater than 30ms) is to wait for the completion of ranging
  dist = (float)readData(SLAVE_ADDR, eDistance) / 10;//Read distance register, one LSB is 0.1mm

  Serial.print("distance = ");
  Serial.print(dist, 1);
  Serial.println("mm");
}
```

Read Onboard Temperature

```
/*****************************************************************************************************************
     This code tests the temperature measurement function of the URM14 ultrasonic sensor
     @ author : roker.wang@dfrobot.com
     @ data   : 11.08.2020
     @ version: 1.0
     RX(TTL-RS485转接板) -> TX1/D1 （Arduino Leonardo）  TX（TTL-RS485转接板）-> RX1/D0 （Arduino Leonardo）
*****************************************************************************************************************/
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>

#define   SLAVE_ADDR               ((uint16_t)0x0C)

#define   TEMP_CPT_SEL_BIT         ((uint16_t)0x01)
#define   TEMP_CPT_ENABLE_BIT      ((uint16_t)0x01 << 1)
#define   MEASURE_MODE_BIT         ((uint16_t)0x01 << 2)
#define   MEASURE_TRIG_BIT         ((uint16_t)0x01 << 3)

typedef enum{
  ePid,
  eVid,
  eAddr,
  eComBaudrate,
  eComParityStop,
  eDistance,
  eInternalTempreture,
  eExternTempreture,
  eControl,
  eNoise
}eRegIndex_t;//Sensor register index


/*
```

```
/*
 *@brief Read data from holding register of client
 *
 *@param addr :  Address of Client
 *@param reg: Reg index

 *@return data if execute successfully, false oxffff.
 */
uint16_t readData(uint16_t addr, eRegIndex_t reg)
{
  uint16_t data;
  if (!ModbusRTUClient.requestFrom(addr, HOLDING_REGISTERS, reg, 1)){
    Serial.print("failed to read registers! ");
    Serial.println(ModbusRTUClient.lastError());
    data = 0xffff;
  }else{
    data =  ModbusRTUClient.read();
  }
  return data;
}

/*
 *@brief write data to holding register of client
 *
 *@param addr :  Address of Client
 *@param reg: Reg index
 *@param data: The data to be written
 *@return 1 if execute successfully, false 0.
 */
uint16_t writeData(uint16_t addr, eRegIndex_t reg, uint16_t data)
{
  if (!ModbusRTUClient.holdingRegisterWrite(addr, reg, data)){
    Serial.print("Failed to write coil! ");
    Serial.println(ModbusRTUClient.lastError());
    return 0;
  }else
    return 1;
}
```

```
void setup() {
  Serial.begin(9600);
  ModbusRTUClient.begin(19200);
}


volatile float temp;
void loop() {
  temp =  (float)readData(SLAVE_ADDR, eInternalTempreture) / 10.0;//Read the temperature register, one LSB is 0.1℃
  Serial.print("internal tempreture = ");
  Serial.print(temp, 1);
  Serial.println("C");
  delay(500);
}
```

Revise Module Address

```
/*****************************************************************************************************************
     This code tests the address modification function of the URM14 ultrasonic sensor
     @ author : roker.wang@dfrobot.com
     @ data   : 11.08.2020
     @ version: 1.0
     RX(TTL-RS485转接板) -> TX1/D1 （Arduino Leonardo）  TX（TTL-RS485转接板）-> RX1/D0 （Arduino Leonardo）
*****************************************************************************************************************/
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>

#define   PUBLIC_ADDR              ((uint16_t)0x00)
#define   SLAVE_ADDR               ((uint16_t)0x0C)

#define   TEMP_CPT_SEL_BIT         ((uint16_t)0x01)
#define   TEMP_CPT_ENABLE_BIT      ((uint16_t)0x01 << 1)
#define   MEASURE_MODE_BIT         ((uint16_t)0x01 << 2)
#define   MEASURE_TRIG_BIT         ((uint16_t)0x01 << 3)

typedef enum{
  ePid,
  eVid,
  eAddr,
  eComBaudrate,
  eComParityStop,
  eDistance,
  eInternalTempreture,
  eExternTempreture,
  eControl,
  eNoise
}eRegIndex_t;//Sensor register index

/*
```

```
/*
 *@brief Read data from holding register of client
 *
 *@param addr :  Address of Client
 *@param reg: Reg index

 *@return data if execute successfully, false oxffff.
 */
uint16_t readData(uint16_t addr, eRegIndex_t reg)
{
  uint16_t data;
  if (!ModbusRTUClient.requestFrom(addr, HOLDING_REGISTERS, reg, 1)){
    Serial.print("failed to read registers! ");
    Serial.println(ModbusRTUClient.lastError());
    data = 0xffff;
  }else{
    data =  ModbusRTUClient.read();
  }
  return data;
}

/*
 *@brief write data to holding register of client
 *
 *@param addr :  Address of Client
 *@param reg: Reg index
 *@param data: The data to be written
 *@return 1 if execute successfully, false 0.
 */
uint16_t writeData(uint16_t addr, eRegIndex_t reg, uint16_t data)
{
  if (!ModbusRTUClient.holdingRegisterWrite(addr, reg, data)){
    Serial.print("Failed to write coil! ");
    Serial.println(ModbusRTUClient.lastError());
    return 0;
  }else
    return 1;
}
```

```
void setup() {
  Serial.begin(9600);
  ModbusRTUClient.begin(19200);
  delay(3000);

}
volatile uint16_t newAddr, res;
void loop() {
  newAddr = 0x11;
  res = writeData(PUBLIC_ADDR, eAddr, newAddr);//Writes the new address value to the register
  Serial.print("The device address has been modified as ");

  Serial.print(newAddr);
  Serial.println(".please reset the device!");
  while (1);
}
```

Revise Module Baud Rate

```
/*****************************************************************************************************************
     This code tests the baudrate modification function of the URM14 ultrasonic sensor
     @ author : roker.wang@dfrobot.com
     @ data   : 11.08.2020
     @ version: 1.0
     RX(TTL-RS485转接板) -> TX1/D1 （Arduino Leonardo）  TX（TTL-RS485转接板）-> RX1/D0 （Arduino Leonardo）
*****************************************************************************************************************/
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>

#define   BAUDRATE_DEFAULT          ((uint32_t)19200)
#define   SLAVE_ADDR                ((uint16_t)0x0C)

#define   TEMP_CPT_SEL_BIT          ((uint16_t)0x01)
#define   TEMP_CPT_ENABLE_BIT       ((uint16_t)0x01 << 1)
#define   MEASURE_MODE_BIT          ((uint16_t)0x01 << 2)
#define   MEASURE_TRIG_BIT          ((uint16_t)0x01 << 3)

typedef enum{
  ePid,
  eVid,
  eAddr,
  eComBaudrate,
  eComParityStop,
  eDistance,
  eInternalTempreture,
  eExternTempreture,
  eControl,
  eNoise
}eRegIndex_t;//Sensor register index

/*
```

```
/*
 *@brief Read data from holding register of client
 *
 *@param addr :  Address of Client
 *@param reg: Reg index

 *@return data if execute successfully, false oxffff.
 */
uint16_t readData(uint16_t addr, eRegIndex_t reg)
{
  uint16_t data;
  if (!ModbusRTUClient.requestFrom(addr, HOLDING_REGISTERS, reg, 1)){
    Serial.print("failed to read registers! ");
    Serial.println(ModbusRTUClient.lastError());
    data = 0xffff;
  }else{
    data =  ModbusRTUClient.read();
  }
  return data;
}

/*
 *@brief write data to holding register of client
 *
 *@param addr :  Address of Client
 *@param reg: Reg index
 *@param data: The data to be written
 *@return 1 if execute successfully, false 0.
 */
uint16_t writeData(uint16_t addr, eRegIndex_t reg, uint16_t data)
{
  if (!ModbusRTUClient.holdingRegisterWrite(addr, reg, data)){
    Serial.print("Failed to write coil! ");
    Serial.println(ModbusRTUClient.lastError());
    return 0;
  }else
    return 1;
}
```

```
void setup() {
  Serial.begin(9600);
  ModbusRTUClient.begin(BAUDRATE_DEFAULT);
  delay(3000);

}
volatile uint16_t baudrateIndex, res;
void loop() {

  baudrateIndex = 3;       //0x0001---2400   0x0002---4800 0x0003---9600   0x0004---14400
                           //0x0005---19200  0x0006---38400 0x0007---57600 0x0008---115200 Other----115200
  res = writeData(SLAVE_ADDR, eComBaudrate, baudrateIndex);//Writes the new baud rate value to the corresponding register
  if (res)
    Serial.print("The baudrate has been modified as 9600.please reset the device!");
  while (1);
}
```

FAQ

For any questions, advice or cool ideas to share, please visit the **DFRobot Forum** (https://www.dfrobot.com/forum/).

More Documents


Get **URM14 RS485 Precision Ultrasonic Sensor** (https://www.dfrobot.com/product-2173.html) from DFRobot Store or **DFRobot Distributor**. (https://www.dfrobot.com/index.php?route=information/distributorslogo)

Turn to the Top