SKU:SEN0352

Introduction

DFRobot URM13 is an ultrasonic ranging sensor with open single probe. This sensor supports TRIG Pulse-triggered ranging(SR04 compatible), UART and I2C, which brings more possibilities for actual using scenarios. With a small and compact body, the sensor works well with 3.3V or 5V mainboards like Arduino, Raspberry Pi, easy to use and integrate into various applications. Besides, the UART mode employs standard Modbus-RTU protocol and integrates receive/send control output to allow users to easily expand RS485 interface using external RS485 transceiver.

URM13 is positioned to be a professional and advanced ultrasonic sensor. The integration of new precision circuit design and smart detection algorithms on the sensor makes it much more smaller and lighter while still excellent in sensitivity. At the same time, URM13 sensor is able to automatically detect the environment and electrical noise, and complete the dynamic adjustment and calibration of sensor parameters in real-time, which ensures that it can keep stable performance in various complex application scenarios.

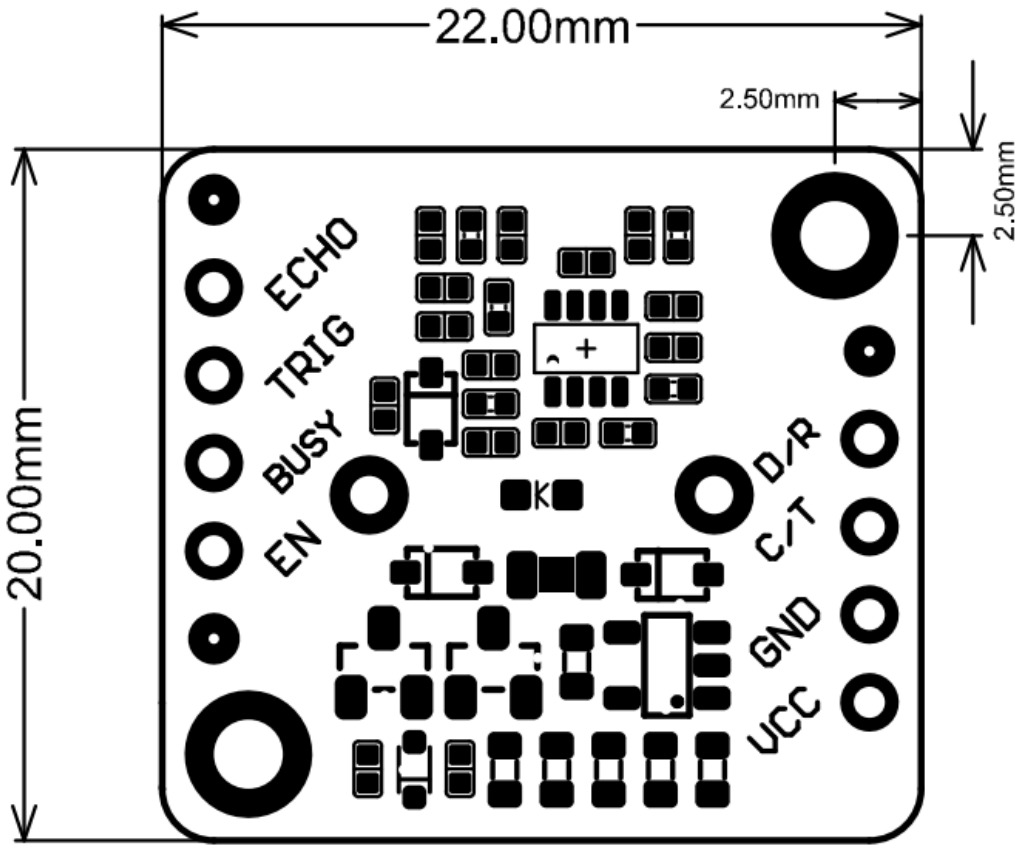
URM13 provides users with two built-in measuring ranges for meeting different application requirements：

1. 15~150cm small range with up to 50Hz detecting frequency, suitbale for indoor robot obstacle avoidance, etc.
2. 40~900cm large range with 10Hz frequency and high sensitivity, applicable for open field scenarios or projects that requires high accuracy and long ranging distance.
   The two detecting ranges can be triggered repeatedly in actual use to realize the measurement of whole range.

Specification

- Power Supply: 3.3~5.5V DC
- Max Instantaneous Work Current: 350mA
- Effective Measuring Range: 15cm～900cm
- Distance Resolution: 1cm
- Distance Error: ±1%
- Temperature Resolution: 0.1℃
- Temperature Error: ±1℃
- Measuring Frequency: 10Hz
- Operating Temperature: -10℃～＋70℃
- Operating Humidity: RH<75%
- Sensor Acoustic Frequency: 40±1KHz
- Sensor Direction Angle: 60°(-6dB)
- Communication Interface: I2C&TRIG/UART(Modbus-RTU)

Board Overview



- Pin Description

Users can switch the URM13 communication interface between I2C and UART according to actual use. The selected interface will be automatically saved in the sensor and can be directly used next time.

Pin	**Default function (UART)**	**Reuse Function (I2C)**	**Description**
VCC	+	+	3.3~5.5V
GND	-	-	Ground
C/T	UART-TX	I2C-SCL	Sensor default to UART, please refer to Mode Switch if you need to use I2C. (The pin is in open-drain, internally pull up 6.2K resistance to VCC)
D/R	UART-RX	I2C-SDA	Sensor default to UART, please refer to Mode Switch if you need to use I2C.(The pin is in open-drain, internally pull up 6.2K resistance to VCC)
EN	Power Enable	Power Enable	Power enable pin, internal pull-up; pull pin EN down to turn off sensor power, leave it floating when this funciton is not used.
BUSY	RS485 Receive/Send control Port	BUSY(Open drain output)	- Under UART mode, this pin can be used as RS485 Receive/Send control port. You can expand a RS485 interface by connecting an external RS485 driver IC. - Under I2C mode, BUSY pin will be pulled down in ranging. If pin BUSY outputs high, it means the sensor is in idle state or the last ranging is done; Connect this pin to Master's interrupt port, then the sensor data can be read in real-time when pin BUSY outputs level rising edge.
TRIG	NC	TRIG	- Not used in UART mode - In I2C mode, used as external port ranging trigger pin, rising edge trigger.

Pin	Default function (UART)	Reuse Function (I2C)	Description
ECHO	NC	ECHO	- Not used in UART mode - In I2C mode, when the sensor ranging function is triggered, this pin outputs a high pulse width that represents ultrasonic transmit time (unit: us)

Onboard LED State Indication

- When powered on, the sensor LED flashes once, which means that it is in UART mode now, and the LED will flash once every time the sensor receives correct command from Master.
- When powered on, the sensor LED flashes twice, which means that it is in I2C mode now, and the LED will flash once when the sensor starts ranging.

Mode Switch (UART and I2C)

The URM13 sensor is set to UART mode by default, and users can switch it between UART and I2C by the short-circuit of different pins before powering on.

- Short circuit pin TRIG and ECHO, then power on, the LED will falsh twice, represeting that the sensor is in I2C mode.
- Short circuit pin TRIG and BUSY, then power on, the LED will flash once, representing that the sensor is in UART mode.

After the mode is successfully switched, users can disconnect the short circuit. The switched mode will be recorded by the sensor permanently.

UART Register Description

In UART mode, the sensor is regarded as a Modbus slave, the Master needs to communicate with the sensor by Modbus protocol.

Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
0x00	1	Module PID	R	0x0000 - 0xFFF	0x0003	Product check (Detect module type)

Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
		Register		F		
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	0x000D	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 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

Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
						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 1mm
0x06	1	Onboard temperature data register	R	0x0000 - 0xFFFF	0x0000	The temperature value 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)
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)

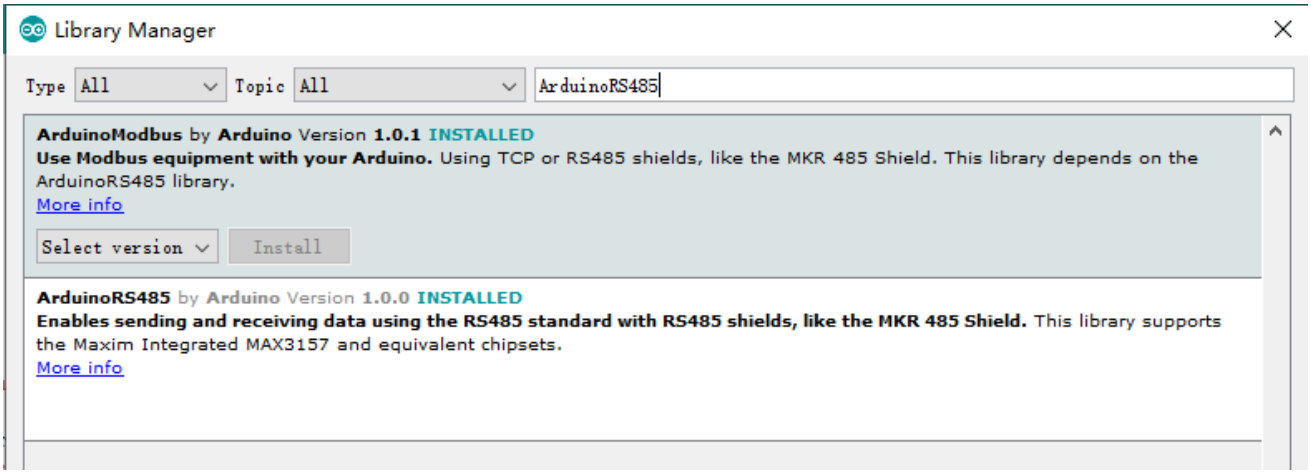
Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
						bit1: 0-enable temperature compensation function 1-disable temperature compensation function bit2: 0-auto detection 1-passive detection bit3: In passive detection mode, write 1 to this bit, then it will measure distance once. The distance value can be read from distance register about 100ms later. This bit is reserved in passive detection mode. This bit will be auto cleared when set to 1 bit4: 0-Large measuring range(40-900cm) 1-Small measuring range(15-150cm) **Save when powered off, take effect after restarting**
0x09	1	Electrical noise degree register	R	0x0000 -0x0A	0x0000	0x0000-0x000A corresponds to noise degree 0 to 10. This parameter can reflect the influence of power supply and environment on the sensor. The smaller the noise level, the more accurate the distance value

Register Address	Number	Name	Read/Write	Data Range	Default Value	Data Description
						detected by the sensor.
0x0A	1	Ranging Sensitivity Setting register	R/W	0x0000 -0x0A	0x0000	0x0000-0x000A corresponds to sensitivity level 0 to 10. Set the ranging sensitivity in large measuring range. The smaller the value, the higher the sensitivity. **Save when powered off, take effect after restarting**

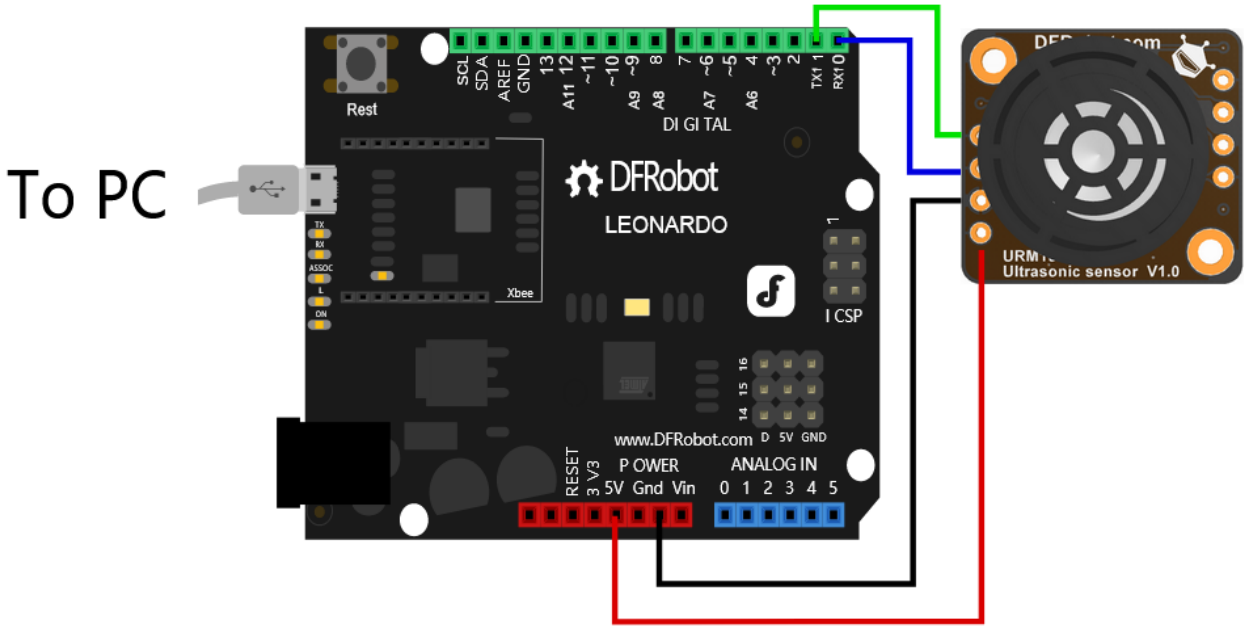
Arduino Tutorial in UART Mode

Requirements

- **Hardware**
  - 1 x Arduino Leonardo (Since printing information needs a serial port, it is recommended to use device with two(or above) serial ports. Because Arduino Modbust takes up a lot of memory, it is suggested to use Arduino Mega2560 controller.)
  - USB Data Cable x 1
- **Software**
  - Arduino IDE
  - Open Library Manager(Ctrl+Shift+I) in Arduino IDE, find and install ArduinoModbus and ArduinoRS485 Libraries.



Connection Diagram



Sample Code(Based on Arduino Modbus Library)

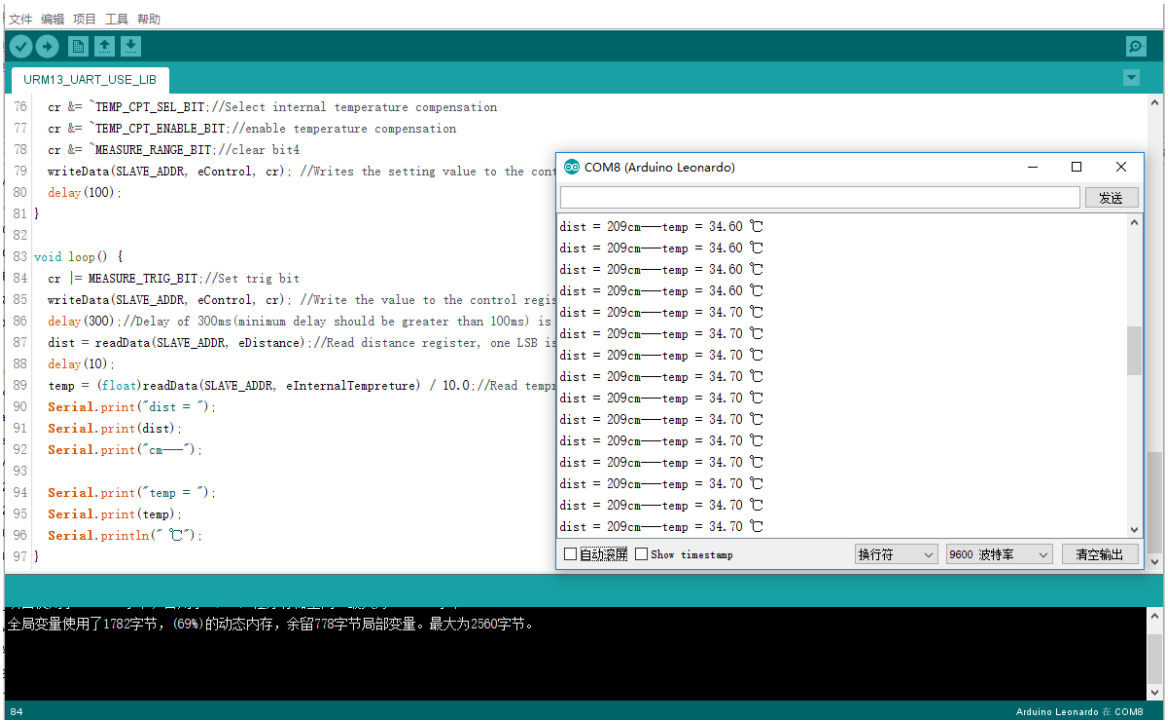
```
/*************************************************************************************
**************************
     This code tests the range finder function of the URM13 ultrasonic sensor
     @ author : roker.wang@dfrobot.com
     @ data   : 21.08.2020
     @ version: 1.0
*************************************************************************************
**************************/
#include <ArduinoModbus.h>
#include <ArduinoRS485.h>
```

```
#define   SLAVE_ADDR                ((uint16_t)0x0D)

#define   TEMP_CPT_SEL_BIT          ((uint16_t)0x01 << 0)
#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)
#define   MEASURE_RANGE_BIT         ((uint16_t)0x01 << 4)

typedef enum {
  ePid,
  eVid,
  eAddr,
  eComBaudrate,
  eComParityStop,
  eDistance,
  eInternalTempreture,
  eExternTempreture,
  eControl,
  eNoise,
  eSensitivity
} 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;
}

int16_t  dist;float temp;
volatile uint16_t cr = 0;
void setup() {
  ModbusRTUClient.begin(19200);
  Serial.begin(9600);
  cr &= ~TEMP_CPT_SEL_BIT;//clear bit0, select internal temperature compensation
  //cr |= TEMP_CPT_SEL_BIT;//set bit0,select external temperature compensation
  cr &= ~TEMP_CPT_ENABLE_BIT;//clear bit1,enable temperature compensation
  //cr |= TEMP_CPT_ENABLE_BIT; //set bit1,disable temperature compensation
  cr |= MEASURE_MODE_BIT;//set bit2 , set to trigger mode
  //cr &= ~MEASURE_MODE_BIT;//clear bit2 , set to Automatic ranging mode
  cr &= ~MEASURE_RANGE_BIT;//clear bit4,long-range ranging mode
  //cr |= MEASURE_RANGE_BIT; //set bit4,short-range ranging mode
  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 100ms) is to wait
for the completion of ranging
  dist = readData(SLAVE_ADDR, eDistance);//Read distance register, one LSB is 1cm
  delay(10);
  temp = (float)readData(SLAVE_ADDR, eInternalTempreture) / 10.0;//Read tempreture
register, one LSB is 0.1 ℃
  Serial.print("dist = ");
  Serial.print(dist);
  Serial.print("cm---");

  Serial.print("temp = ");
  Serial.print(temp);
  Serial.println(" ℃");
}
```

Copy

Expected Result 1



Sample Code (Send command from Arduino serial)

```
/*!
 * @file URM13_UART_USE_LIB.ino
 * @brief This code tests the range finder function of the URM13 ultrasonic sensor
 * @author : roker.wang@dfrobot.com
 * @data   : 21.08.2020
 * @version: 1.0
 */
#define   SLAVE_ADDR                ((uint16_t)0x0D)

#define   TEMP_CPT_SEL_BIT          ((uint16_t)0x01 << 0)
#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)
#define   MEASURE_RANGE_BIT         ((uint16_t)0x01 << 4)

#define   MB_OP_WRITE_SINGLE_HOLDING_REG  ((uint8_t)0x06)
#define   MB_OP_READ_HOLDING_REGS         ((uint8_t)0x03)

typedef enum {
  ePid,
  eVid,
  eAddr,
  eComBaudrate,
```

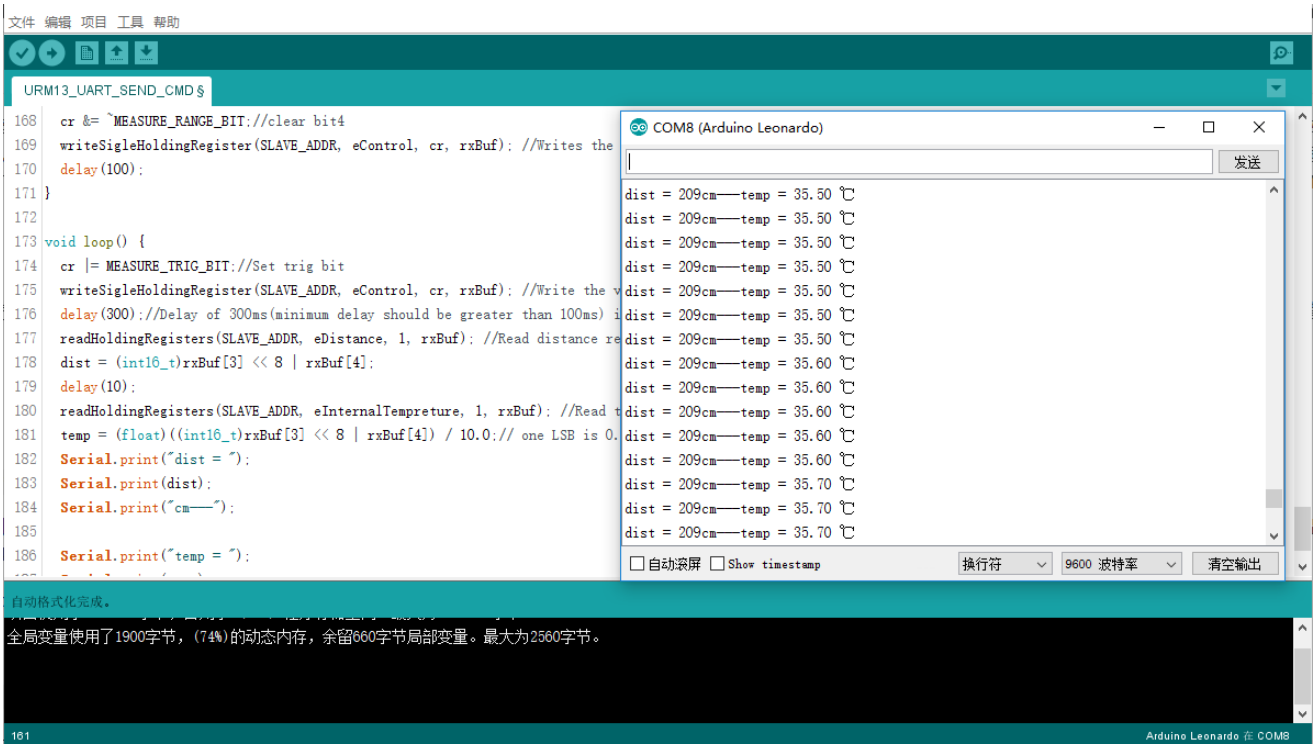
```
  eComParityStop,
  eDistance,
  eInternalTempreture,
  eExternTempreture,
  eControl,
  eNoise,
  eSensitivity
} eRegIndex_t; //Sensor register index

static const uint8_t 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 uint8_t 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
};

static uint16_t mbCrcCalculated(uint8_t  * pCmd, uint8_t  usLen )
{
  uint8_t ucCRCHi = 0xFF;
  uint8_t ucCRCLo = 0xFF;
  int16_t iIndex;

  while ( usLen-- )
  {
    iIndex = ucCRCLo ^ *( pCmd++ );
    ucCRCLo = ( uint8_t )( ucCRCHi ^ aucCRCHi[iIndex] );
    ucCRCHi = aucCRCLo[iIndex];
  }
  return ( uint16_t )( (uint16_t)ucCRCHi << 8 | ucCRCLo );
}
/*
  @brief Read data from holding register of client

  @param addr ： Address of Client
  @param reg: Reg index
  @param regNum: The number of registers to read, The register is 16 bits wide
  @param pBuf:Points to the receive data buffer
*/
void readHoldingRegisters(uint16_t addr, eRegIndex_t regIndex, uint16_t regNum,
uint8_t *pBuf)
{
  uint8_t pCmdBuf[8], i;
  uint16_t crc;

  pCmdBuf[0] = addr;
  pCmdBuf[1] = MB_OP_READ_HOLDING_REGS;
  pCmdBuf[2] = regIndex >> 8;
  pCmdBuf[3] = (uint8_t)regIndex;
  pCmdBuf[4] = regNum >> 8;
  pCmdBuf[5] = (uint8_t)regNum;

  crc = mbCrcCalculated(pCmdBuf, 6);
  pCmdBuf[6] = (uint8_t)crc;
  pCmdBuf[7] = crc >> 8;

  for (i = 0; i < 8; i++) {
    Serial1.write( pCmdBuf[i]);
  }
  delay(150);
  i = 0;
  while (Serial1.available()) {
    pBuf[i++] = (Serial1.read());
  }
}
/*
  @brief Write data to holding register of client
```

```
  @param addr  :  Address of Client
  @param reg: Reg index
  @param data: The data to be sent
  @param pBuf:Points to the receive data buffer
*/
void writeSigleHoldingRegister(uint16_t addr, eRegIndex_t regIndex, uint16_t data,
uint8_t *pBuf)
{
  uint8_t pCmdBuf[8], i;
  uint16_t crc;

  pCmdBuf[0] = addr;
  pCmdBuf[1] = MB_OP_WRITE_SINGLE_HOLDING_REG;
  pCmdBuf[2] = regIndex >> 8;
  pCmdBuf[3] = (uint8_t)regIndex;
  pCmdBuf[4] = data >> 8;
  pCmdBuf[5] = (uint8_t)data;

  crc = mbCrcCalculated(pCmdBuf, 6);
  pCmdBuf[6] = (uint8_t)crc;
  pCmdBuf[7] = crc >> 8;

  for (i = 0; i < 8; i++) {
    Serial1.write( pCmdBuf[i]);
  }
  delay(150);
  i = 0;
  while (Serial1.available()) {
    pBuf[i++] = (Serial1.read());
  }
}

uint8_t rxBuf[100];
int16_t  dist; float temp;

volatile uint16_t cr = 0;
void setup() {
  Serial1.begin(19200);
  Serial.begin(9600);
  cr &= ~TEMP_CPT_SEL_BIT;//clear bit0, select internal temperature compensation
  //cr |= TEMP_CPT_SEL_BIT;//set bit0,select external temperature compensation
  cr &= ~TEMP_CPT_ENABLE_BIT;//clear bit1,enable temperature compensation
  //cr |= TEMP_CPT_ENABLE_BIT; //set bit1,disable temperature compensation
  cr |= MEASURE_MODE_BIT;//set bit2 , set to trigger mode
  //cr &= ~MEASURE_MODE_BIT;//clear bit2 , set to Automatic ranging mode
  cr &= ~MEASURE_RANGE_BIT;//clear bit4,long-range ranging mode
  //cr |= MEASURE_RANGE_BIT; //set bit4,short-range ranging mode
  writeSigleHoldingRegister(SLAVE_ADDR, eControl, cr, rxBuf); //Writes the setting
value to the control register
  delay(100);
}

void loop() {
  cr |= MEASURE_TRIG_BIT;//Set trig bit
```

```
  writeSigleHoldingRegister(SLAVE_ADDR, eControl, cr, rxBuf); //Write the value to
the control register and trigger a ranging
  delay(300);//Delay of 300ms(minimum delay should be greater than 100ms) is to wait
for the completion of ranging
  readHoldingRegisters(SLAVE_ADDR, eDistance, 1, rxBuf); //Read distance register,
one LSB is 1cm
  dist = (int16_t)rxBuf[3] << 8 | rxBuf[4];
  delay(10);
  readHoldingRegisters(SLAVE_ADDR, eInternalTempreture, 1, rxBuf); //Read tempreture
register
  temp = (float)((int16_t)rxBuf[3] << 8 | rxBuf[4]) / 10.0;// one LSB is 0.1 ℃
  Serial.print("dist = ");
  Serial.print(dist);
  Serial.print("cm---");

  Serial.print("temp = ");
  Serial.print(temp);
  Serial.println(" ℃");
}
```

Expected Result 2



I2C Register Description

Register Address	Name	Read/Write	Data Range	Default Value	Data Description
0x00	Sensor Address Register	R	0x01-0xF7	0x12	I2C Slave address **Save when powered off, take effect after restarting**
0x01	Sensor PID Register	R	0x00-0xFF	0x02	Product check (detect sensor type)
0x02	Sensor VID Register	R	0x00-0xFF	0x10	Firmware Version: 0x10 for V1.0
0x03 0x04	Distance value register high bit Distance value register low bit	R R	0x00-0xFF 0x00-0xFF	0xFF 0xFF	LSB is 1cm, 0x0064 = 100cm
0x05 0x06	Onboard temperature value register high bit Onboard temperature value register low bit	R R	0x00-0xFF 0x00-0xFF	0xFF 0xFF	LSB is 0.1℃(with unit symbol). For example, TEMP_H = 0x00, TEMP_L = 0xfe, the actual measured temperature 0x00fe / 10 = 25.4℃
0x07 0x08	External temperature compensation data value register high bit External temperature compensation data register	R/W R/W	0x00-0xFF 0x00-0xFF	0x00 0x00	Write ambient temperature data to this register for external temperature compensation, LSB is 0.1℃(with unit symbol)

Register Address	Name	Read/Write	Data Range	Default Value	Data Description
	low bit				
0x09	Configuration register	R/W	0x00-0xFF	0x04	bit5-bit7: reserved bit4(Max measuring distance set bit): 0: large measuring range (40 - 900cm) 1: small measuring range(15-150cm) bit 3: resvered: bit2: 0:0-auto detection, sensor do measurement all the time and the distance in register is constantly updated 1:passive detection, send the ranging command once, the sensor do the measurement once and store the distance value in distance register bit1: 0: enable temperature compensation 1: distable temperature compensation bit0: 0-use onboard temperature compensation function 1-use external temperature compensation function. **Save when powered off, take effect after restarting**
0x0A	Command register	R/W	0x00-0xFF	0x00	bit7-bit6: reserved bit0: Write 1 to this bit to trigger one measurement. 0 will be ignored
0x0B	Electrical noise degree	R	0x00-0x0A	0x00	0x00-0x0A corresponds to noise degree 0 to 10.

Register Address	Name	Read/Write	Data Range	Default Value	Data Description
	register				This parameter can reflect the influence of power supply and environment on the sensor. The smaller the noise level, the more accurate the distance value detected by the sensor.
0x0C	Ranging Sensitivity Setting register	R/W	0x00-0x0A	0x00	0x00-0x0A corresponds to sensitivity level 0 to 10. Set the ranging sensitivity in large measuring range(40-900cm). The smaller the value, the higher the sensitivity. **Save when powered off, take effect after restarting**

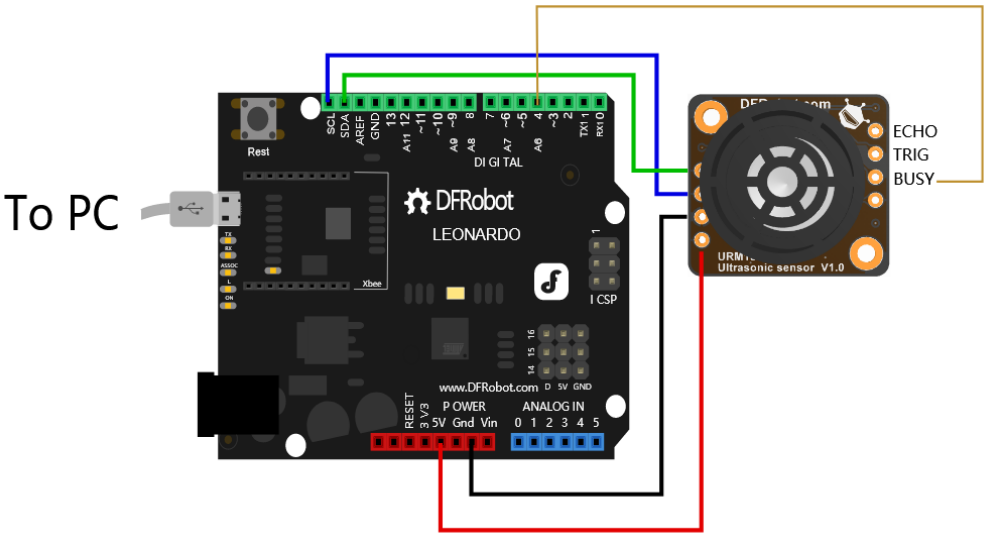
I2C Mode Arduino Tutorial

Requirements

- **Hardware**
  - 1 x Arduino Leonardo
  - 1 x USB Data Cable (Connect Arduino board to a PC with USB cable)

- **Software**
  - Arduino IDE

Connection Diagram



Sample Code

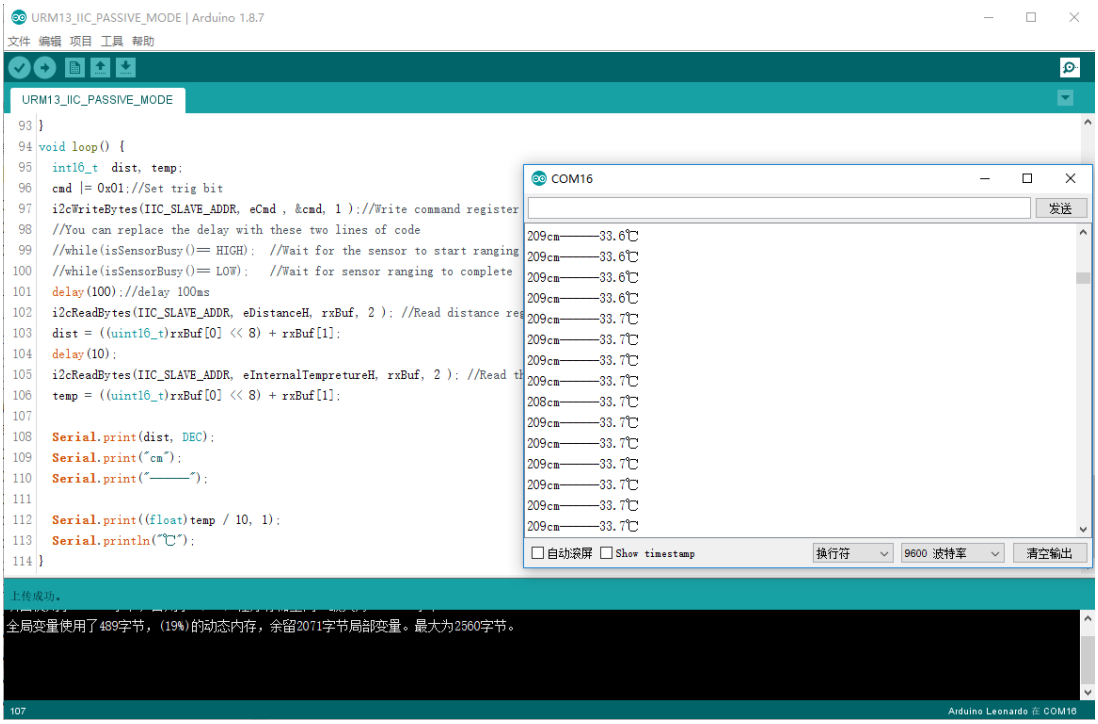
```
/*!
       Download this demo to test config to URM13, connect sensor through IIC
interface
       Data will print on your serial monitor

       This example is the ultrasonic passive measurement distance and the
temperature of the module.

       Copyright   [DFRobot](http://www.dfrobot.com), 2018
       Copyright   GNU Lesser General Public License

       version  V1.0
       date  21/08/2020
*/
#include <Wire.h>
typedef enum {
  eAddr = 0,
  ePid,
  eVid,
  eDistanceH ,
  eDistanceL,
  eInternalTempretureH,
  eInternalTempretureL,
  eExternalTempretureH,
  eExternalTempretureL,
  eConfig,
  eCmd,
  eNoise,
  eSensitivity,
```

```
  eRegNum
} regindexTypedef;

#define    MEASURE_RANGE_BIT        ((uint8_t)0x01 << 4)
#define    MEASURE_MODE_BIT         ((uint8_t)0x01 << 2)
#define    TEMP_CPT_ENABLE_BIT      ((uint8_t)0x01 << 1)
#define    TEMP_CPT_SEL_BIT         ((uint8_t)0x01 << 0)

#define    IIC_SLAVE_ADDR           ((uint8_t)0x12)
#define    isSensorBusy()           (digitalRead(busyPin))

int16_t    busyPin = 4;
/*
  @brief Write data to register of client

  @param addr : Address of Client
  @param regIndex: Reg index
  @param pDataBuf: point to data buffer
  @param dataLen: data length
*/
void i2cWriteBytes(uint8_t addr, regindexTypedef regIndex , uint8_t *pDataBuf,
uint8_t dataLen )
{
  Wire.beginTransmission(addr); // transmit to device
  Wire.write(regIndex);              // sends one byte
  for (uint8_t i = 0; i < dataLen; i++) {
    Wire.write(*pDataBuf);
    pDataBuf++;
  }
  Wire.endTransmission();    // stop transmitting
}
/*
  @brief Read data from register of client

  @param addr : Address of Client
  @param regIndex: Reg index
  @param pDataBuf: point to data buffer
  @param dataLen: data length
*/
void i2cReadBytes(uint8_t addr, regindexTypedef regIndex , uint8_t *pDataBuf, uint8_t
dataLen )
{
  unsigned char i = 0;
  Wire.beginTransmission(addr); // transmit to device #8
  Wire.write(regIndex);              // sends one byte
  Wire.endTransmission();    // stop transmitting
  Wire.requestFrom(addr, dataLen);
  while (Wire.available()) {  // slave may send less than requested
    pDataBuf[i] = Wire.read();
    i++;
  }
}

uint8_t cfg = 0, cmd = 0;
uint8_t rxBuf[100] = {0};
```

```
void setup() {
  Wire.begin(); // join i2c bus (address optional for master)
  Serial.begin(9600); // join i2c bus (address optional for master)
  pinMode(busyPin, INPUT);
  cfg &= ~MEASURE_RANGE_BIT;//clear bit4,long-range ranging mode
  //cfg |= MEASURE_RANGE_BIT;//set bit4,short-range ranging mode
  cfg |=  MEASURE_MODE_BIT;//Set bit2, i2c passive mode
  //cfg &= ~MEASURE_MODE_BIT;//clear bit2 , set to Automatic ranging mode
  cfg &= ~TEMP_CPT_ENABLE_BIT;//clear bit1,enable temperature compensation
  //cfg |= TEMP_CPT_ENABLE_BIT;//set bit1,disable temperature compensation
  cfg &= ~TEMP_CPT_SEL_BIT;//clear bit0,select internal temperature compensation
  //cfg |= TEMP_CPT_SEL_BIT;//set bit0,select external temperature compensation
  i2cWriteBytes(IIC_SLAVE_ADDR, eConfig , &cfg, 1 );
  delay(100);
}
void loop() {
  int16_t  dist, temp;
  cmd |= 0x01;//Set trig bit
  i2cWriteBytes(IIC_SLAVE_ADDR, eCmd , &cmd, 1 );//Write command register
  //You can replace the delay with these two lines of code
  //while(isSensorBusy()== HIGH);  //Wait for the sensor to start ranging
  //while(isSensorBusy()== LOW);   //Wait for sensor ranging to complete
  delay(100);//delay 100ms
  i2cReadBytes(IIC_SLAVE_ADDR, eDistanceH, rxBuf, 2 ); //Read distance register
  dist = ((uint16_t)rxBuf[0] << 8) + rxBuf[1];
  delay(10);
  i2cReadBytes(IIC_SLAVE_ADDR, eInternalTempretureH, rxBuf, 2 ); //Read the onboard
temperature register
  temp = ((uint16_t)rxBuf[0] << 8) + rxBuf[1];

  Serial.print(dist, DEC);
  Serial.print("cm");
  Serial.print("------");

  Serial.print((float)temp / 10, 1);
  Serial.println("℃");
}
```

Copy

Expected Result 3



TRIG Pulse-triggered Sample

- The way to drive the module is the same to SR04. Input a pulse level in TRIG port to trigger distance measurement. Pin ECHO will output a pulse level when ranging is done. The width of the output pulse level is equal to the round-trip time of ultrasonic wave between the detected object and the sensor.

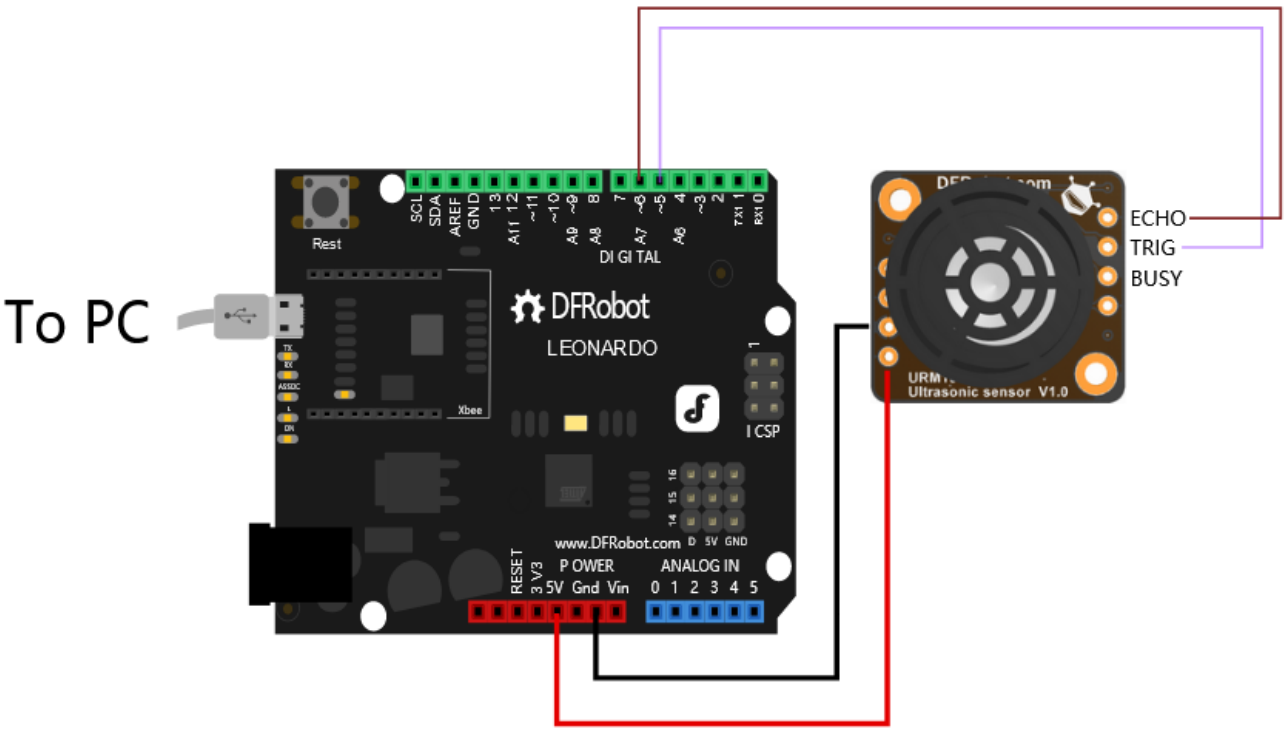
Special Instruction:
1.Measuring Range cannot be switched in TRIG mode, you have to use I2C interface to configure or switch.
2.The output pulse width in TRIG mode is not compensated by temperature.

```
- Please make sure that the sensor is in I2C mode and passive ranging mode(I2C Config registerbit 2=1) before using.
```

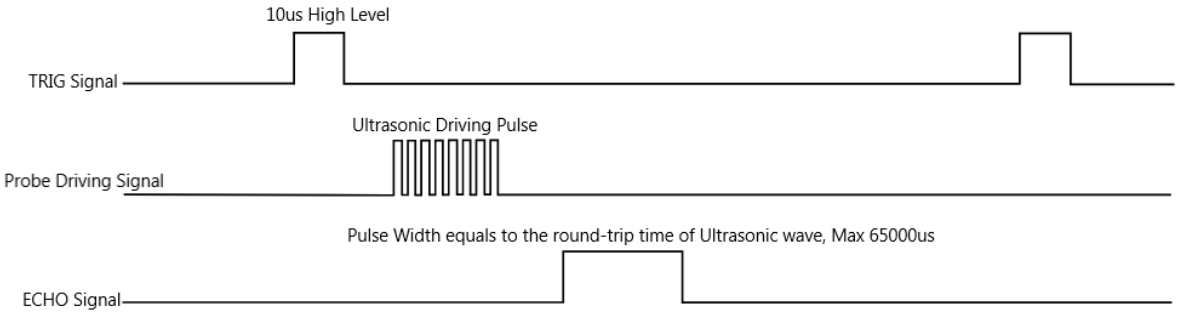
- **Hardware**
  - 1 x Arduino Leonardo
  - 1 x USB Data Cable (Connect Arduino board to a PC with USB cable)

- **Software**
  - Arduino IDE

Connection Diagram



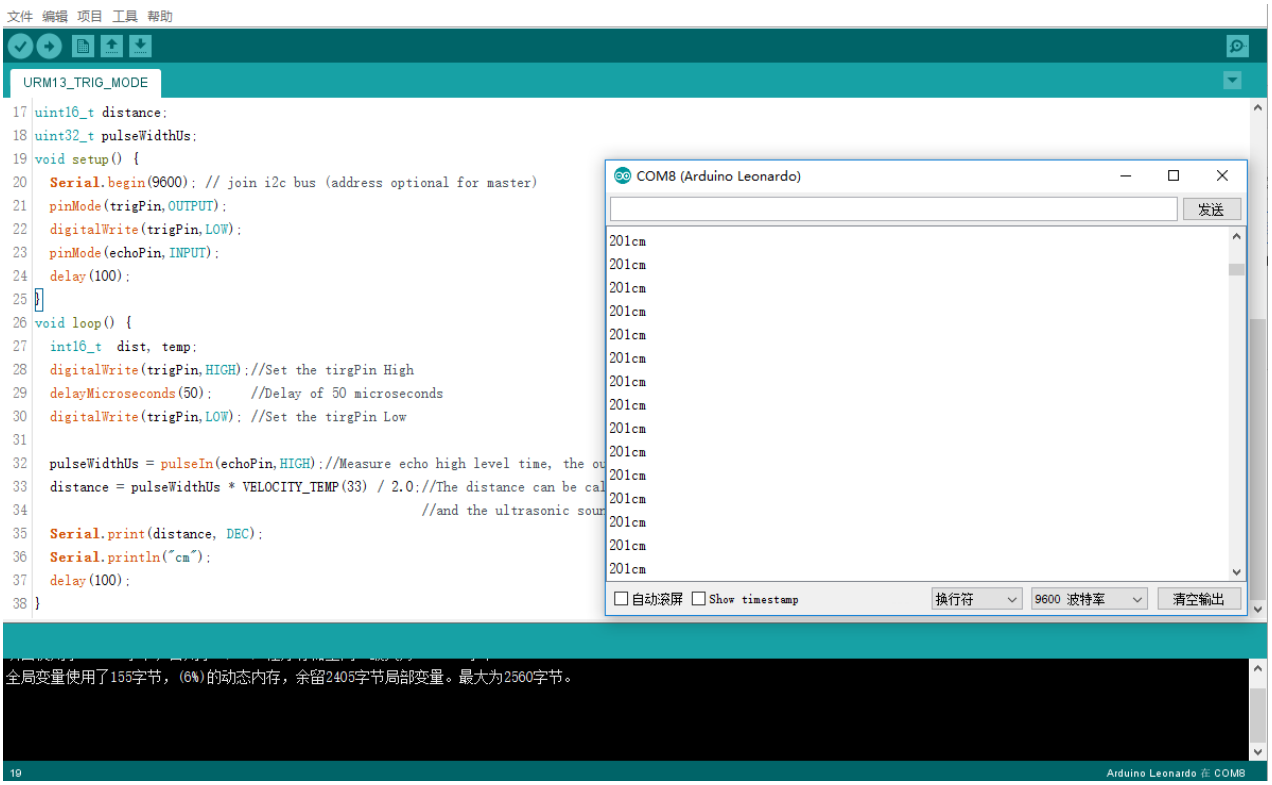
- Timing Diagram



- Sample Code

```
/*!
       This example is the ultrasonic distance measurement of the module.

       Copyright   [DFRobot](http://www.dfrobot.com), 2020
       Copyright   GNU Lesser General Public License

       version  V1.0
       date  21/08/2020
*/
#define    VELOCITY_TEMP(temp)       ( ( 331.5 + 0.6 * (float)( temp ) ) * 100 /
1000000.0 ) // The ultrasonic velocity (cm/us) compensated by temperature

int16_t trigPin = 5;
int16_t echoPin = 6;
uint16_t distance;
uint32_t pulseWidthUs;
void setup() {
  Serial.begin(9600);
  pinMode(trigPin,OUTPUT);
  digitalWrite(trigPin,LOW);
  pinMode(echoPin,INPUT);
  delay(100);
}
void loop() {
  int16_t  dist, temp;
  digitalWrite(trigPin,HIGH);//Set the tirgPin High
  delayMicroseconds(50);     //Delay of 50 microseconds
  digitalWrite(trigPin,LOW); //Set the tirgPin Low

  pulseWidthUs = pulseIn(echoPin,HIGH);//Measure echo high level time, the output
high level time represents the ultrasonic flight time (unit: us)
  distance = pulseWidthUs * VELOCITY_TEMP(33) / 2.0;//The distance can be calculated
according to the flight time of ultrasonic wave,/
                                                    //and the ultrasonic sound speed
can be compensated according to the actual ambient temperature
  Serial.print(distance, DEC);
  Serial.println("cm");
  delay(100);
}
```

Expected Result 4



FAQ

For any questions, advice or cool ideas to share, please visit the **DFRobot Forum**.

More Documents