Interfacing and Embedded laboratory

Two wires interface I2C

Table of Contents

Objectives **2**

Introduction **3**

Part 1.1: I2C protocol overview **3**

Part 1.2: I2C data transfer and timing **4**

Part 1.3: LM75B temperature sensor **8**

Procedure **10**

Part 2.1: Writing the LCD to Arduino and I2C scanning **10**

Part 2.2: Display static text on the LCD **14**

Part 2.3: I2C communication between 2 Arduinos **16**

Part 2.4: I2C communication between 2 Arduinos and LCD **18**

Part 2.5: I2C communication between Arduino and Temperature sensor **18**

Objectives

- Study the serial communication between devices.
- Study I2C and understand the protocol in depth.
- Apply address scanning on I2C protocol.
- Connect LCD with the Arduino using I2C
- Identify different functions for displaying on LCD
- Connect multiple device to communicate using I2C

Introduction

Part 1.1: I2C protocol overview

A typical embedded system consists of one or more microcontrollers and peripheral devices like memories, converters, I/O expanders, LCD drivers, sensors, matrix switches, etc. The complexity and the cost of connecting all those devices together must be kept to a minimum. The system must be designed in such a way that slower devices can communicate with the system without slowing down faster ones.

To satisfy these requirements a serial bus is needed. A bus means specification for the connections, protocol, formats, addresses and procedures that define the rules on the bus. This is exactly what I2C bus specifications define.

I2C combines the best features of other serial communication protocols like SPI and UARTs. With I2C, you can connect multiple slaves to a single master and you can have multiple masters controlling single, or multiple slaves. This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD.

Like UART communication, I2C only uses two wires to transmit data between devices (see Figure 1).

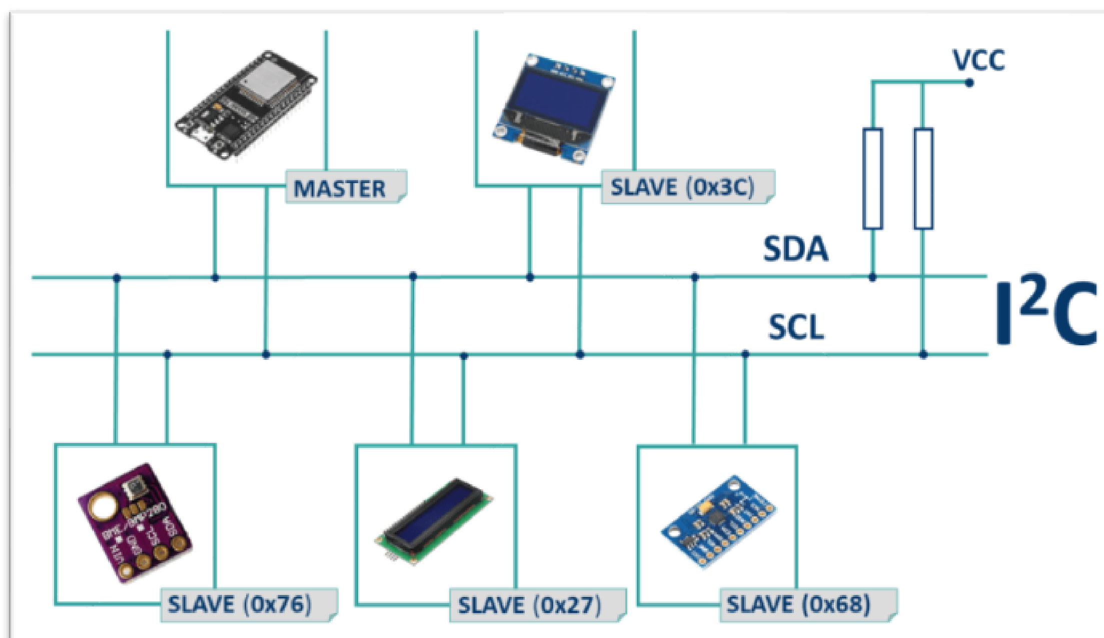


Figure 1: I2C connection

The wires used are:

SDA (Serial Data) – The line for the master and slave to send and receive data.

SCL (Serial Clock) – The line that carries the clock signal.

Both signals (SCL and SDA) are bidirectional. They are connected via resistors to a positive power supply voltage. This means that when the bus is free, both lines are high.

I2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line).

I2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.

Moreover, here are some of the I2c terminologies:

Transmitter
This is the device that transmits data to the bus

Receiver
This is the device that receives data from the bus

Master
This is the device that generates clock, starts communication, sends I2C commands and stops communication

Slave
This is the device that listens to the bus and is addressed by the master

Multi-master
I2C can have more than one master and each can send commands

Arbitration
A process to determine which of the masters on the bus can use it when more masters need to use the bus

Synchronization
A process to synchronize clocks of two or more devices

Part 1.2: I2C data transfer and timing

1. Bit Transfer:

For each clock pulse one bit of data is transferred. The SDA signal can only change when the SCL signal is low – when the clock is high the data should be stable. (see figure 2)

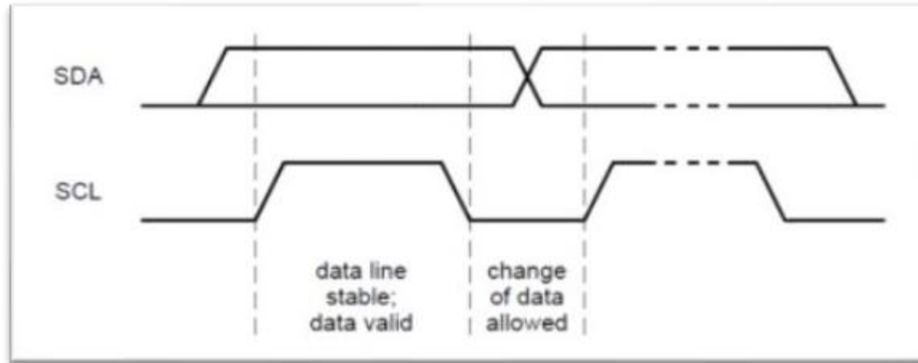


Figure 2: I2C bit transfer

2. Start and stop conditions:

Each I2C command initiated by master device starts with a START condition and ends with a STOP condition. For both conditions SCL has to be high. A high to low transition of SDA is considered as STARTand a low to high transition as STOP. (see figure 3)

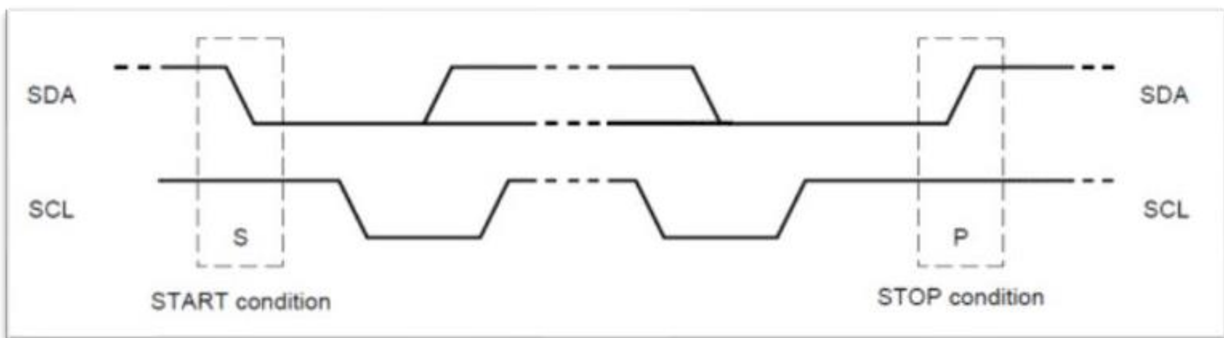


Figure 3: I2C Start and stop bits transfer

After the Start condition the bus is considered as busy and can be used by another master only after a Stop condition is detected. After the Start condition the master can generate a repeated Start. This is equivalent to a normal Start and is usually followed by the slave I2C address.

Microcontrollers that have dedicated I2C hardware can easily detect bus changes and behave also as I2C slave devices. However, if the I2C communication is implemented in software, the bus signals must be sampled at least two times per clock cycle in order to detect necessary changes.

3. I2C data transfer:

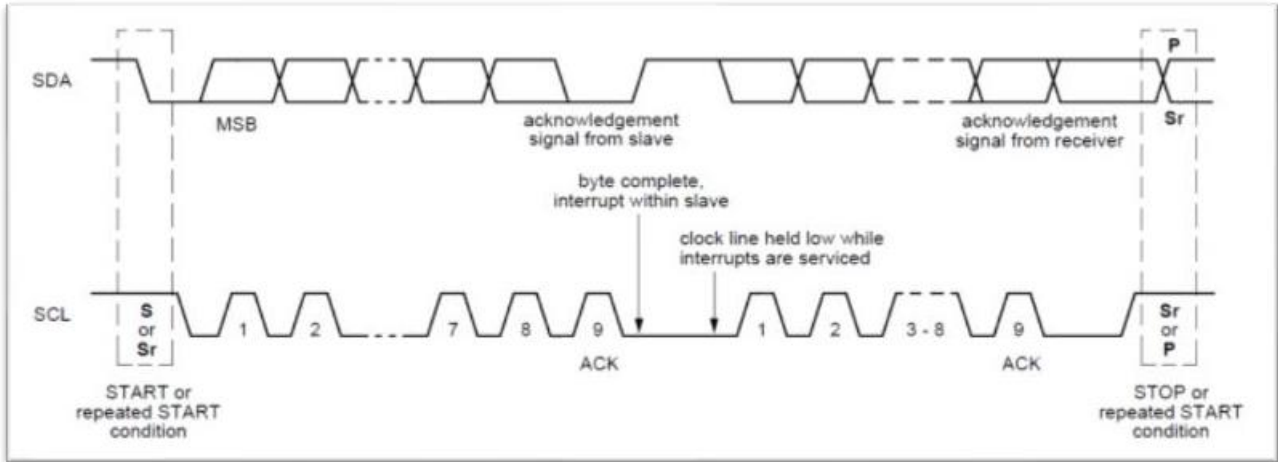


Figure 4: I2C data transfer

Figure 4 shows the data flow for the I2C on both SDA and SCL. Data on the I2C bus is transferred in 8-bit packets (bytes). There is no limitation on the number of bytes, however, each byte must be followed by an Acknowledge bit. This bit signals whether the device is ready to proceed with the next byte. For all data bits including the Acknowledge bit, the master must generate clock pulses. If the slave device does not acknowledge transfer this means that there is no more data or the device is not ready for the transfer yet. The master device must either generate Stop or Repeated Start condition. (see figure 5 for the timing of the acknowledgment bit).

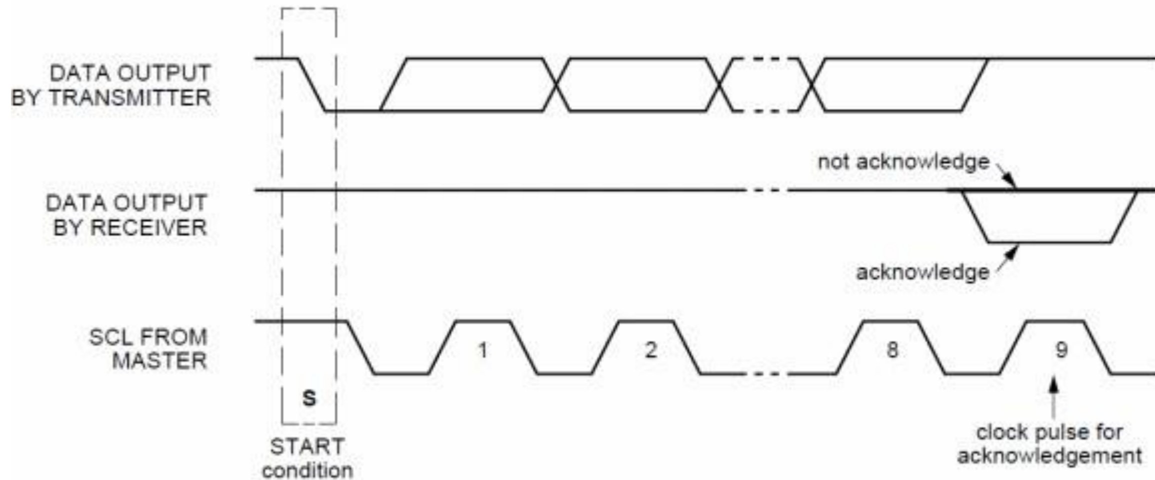


Figure 5: I2C ACK diagram

4. I2C SYNCHRONIZATION:

Each master must generate its own clock signal and the data can change only when the clock is low. For successful bus arbitration a synchronized clock is needed. Once a master pulls the clock

low it stays low until all masters put the clock into high state. Similarly, the clock is in the high state until the first master pulls it low. This way by observing the SCL signal, master devices can synchronize their clocks.

5. ARBITRATION:

For normal data transfer on the I2C bus only one master can be active. If for some reason two masters initiate I2C command at the same time, the arbitration procedure determines which master wins and can continue with the command. Arbitration is performed on the SDA signal while the SCL signal is high. Each master checks if the SDA signal on the bus corresponds to the generated SDA signal. If the SDA signal on the bus is low but it should be high, then this master has lost arbitration. Master I2C device that has lost arbitration can generate SCL pulses until the byte ends and must then release the bus and go into slave mode. The arbitration procedure can continue until all the data is transferred. This means that in multi-master system each I2C master must monitor the I2C bus for collisions and act accordingly.

6. COMMUNICATION WITH 7-BIT I2C ADDRESSES

Each slave device on the bus should have a unique 7-bit address. The communication starts with the Start condition, followed by the 7-bit slave address and the data direction bit. If this bit is 0 then the master will write to the slave device. Otherwise, if the data direction bit is 1, the master will read from slave device. After the slave address and the data direction is sent, the master can continue with reading or writing. The communication is ended with the Stop condition which also signals that the I2C bus is free. If the master needs to communicate with other slaves it can generate a repeated start with another slave address without generation Stop condition. All the bytes are transferred with the MSB bit shifted first. Figure 6 shows the full timing for the I2C protocol signals including the start and stop bits.

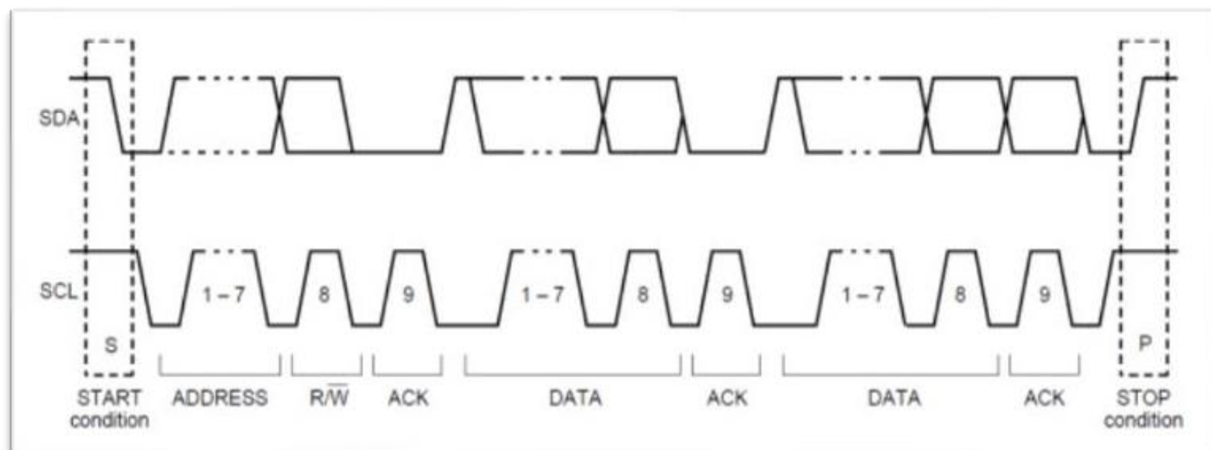


Figure 6: Full timing for the I2C data flow

If the master only writes to the slave device then the data transfer direction is not changed. (see figure 7).

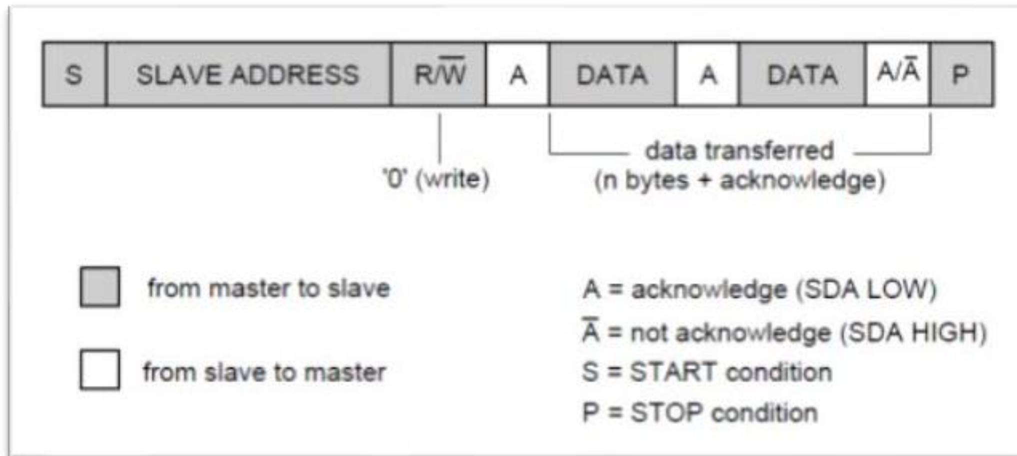


Figure 7: Master writes on slave

If the master only needs to read from the slave device then it simply sends the I2C address with the R/W bit set to read. After this the master device starts reading the data. (see figure 8).

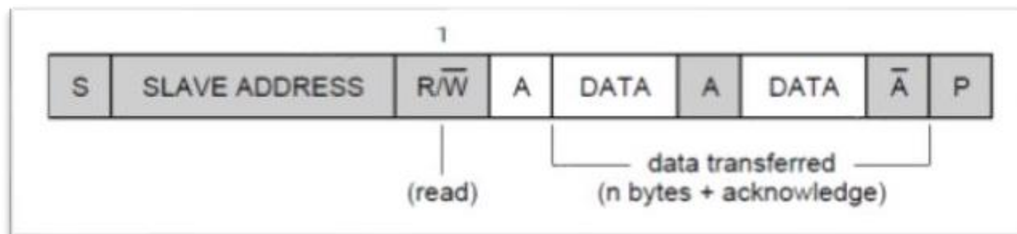


Figure 8: Master reads from slave

Sometimes the master needs to write some data and then read from the slave device. In such cases it must first write to the slave device, change the data transfer direction and then read the device. This means sending the I2C address with the R/W bit set to write and then sending some additional data like register address. After writing is finished the master device generates repeated start condition and sends the I2C address with the R/W bit set to read. After this the data transfer direction is changed and the master device starts reading the data. (See figure 9).

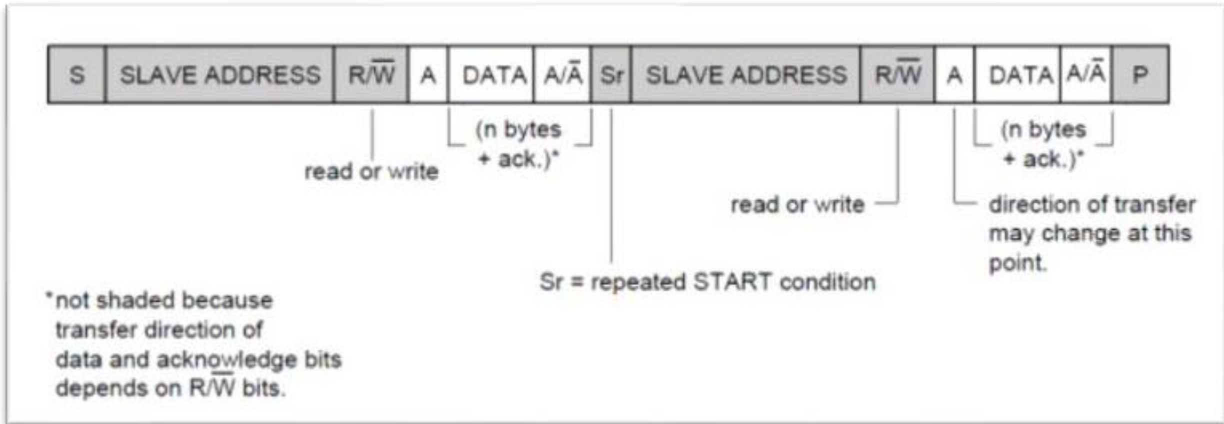
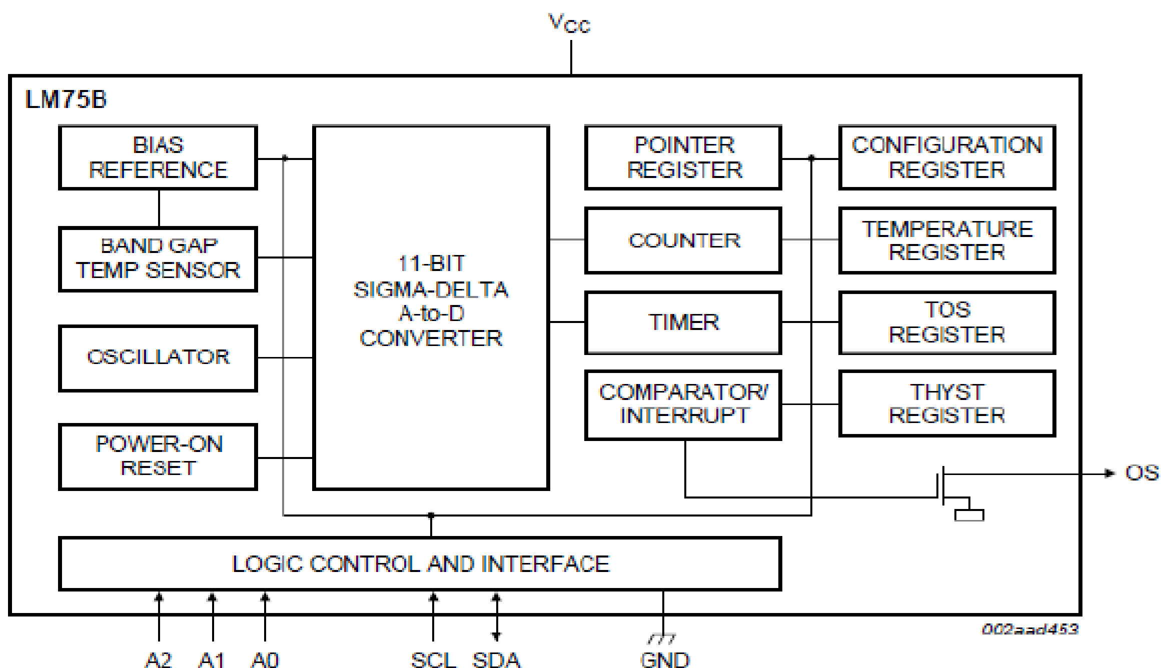


Figure 9: Master reads and writes to the slave device

Part 1.3: LM75B temperature sensor

LM75B will be an example of the communication between Arduino and a sensor by using I2C.
The LM75B is a temperature-to-digital converter using an on-chip band gap temperature sensor and Sigma-Delta A-to-D conversion technique. The result of conversion is available via I2C serial connection. The principle of the sensor is that the forward voltage VBE of a silicon diode is temperature-dependent. If VBE is measured for 2 different values IC1 and IC2 of the current, the variation DVBE is related to the temperature by:

$$\Delta V_{BE} = \frac{KT}{q} \cdot \ln\left(\frac{I_{C1}}{I_{C2}}\right)$$



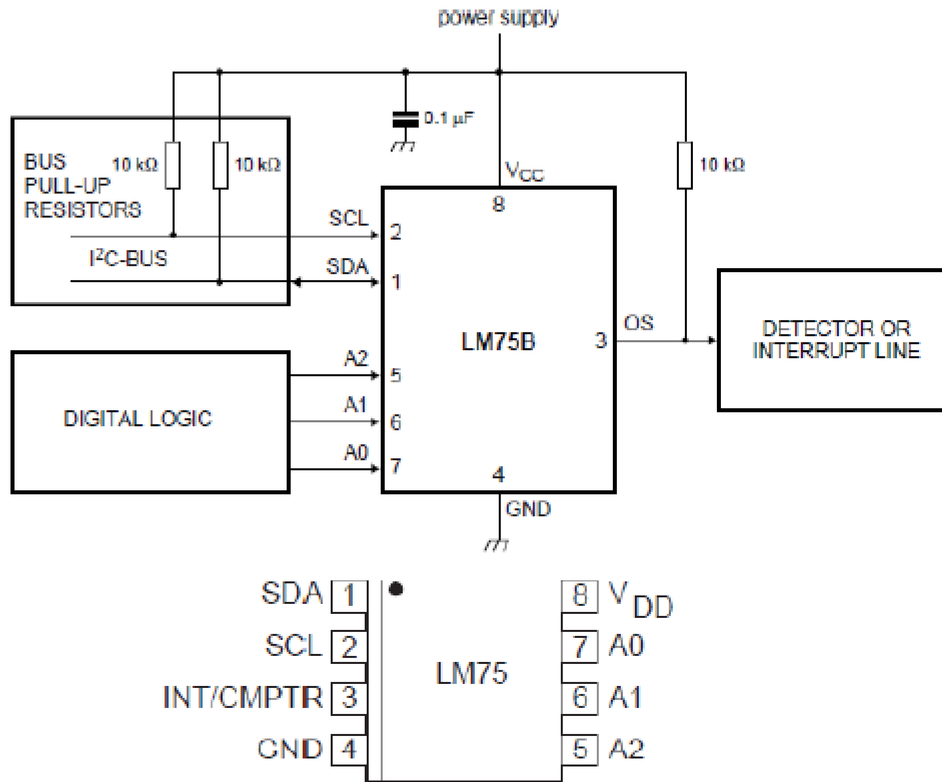


Figure 10: LM75B Config

Pin No.	Symbol	Description
1	SDA	Bidirectional Serial Data
2	SCL	Serial Data Clock Input
3	INT/CMPTR	Interrupt or Comparator Output
4	GND	System Ground
5	A_2	Address Select Pin (MSB)
6	A_1	Address Select Pin
7	A_0	Address Select Pin (LSB)
8	V_{DD}	Power Supply Input

1	0	0	1	A2	A1	A0
MSB						LSB

Address (A2, A1, A0) Inputs. Sets the three least significant bits of the LM758–bit address. A **match between the LM75's address and the address specified in the serial bit stream** must be made to initiate communication with the LM75.

Procedure

Part 2.1: Writing the LCD to Arduino and I2C scanning

This display uses I2C communication, which makes wiring really simple.

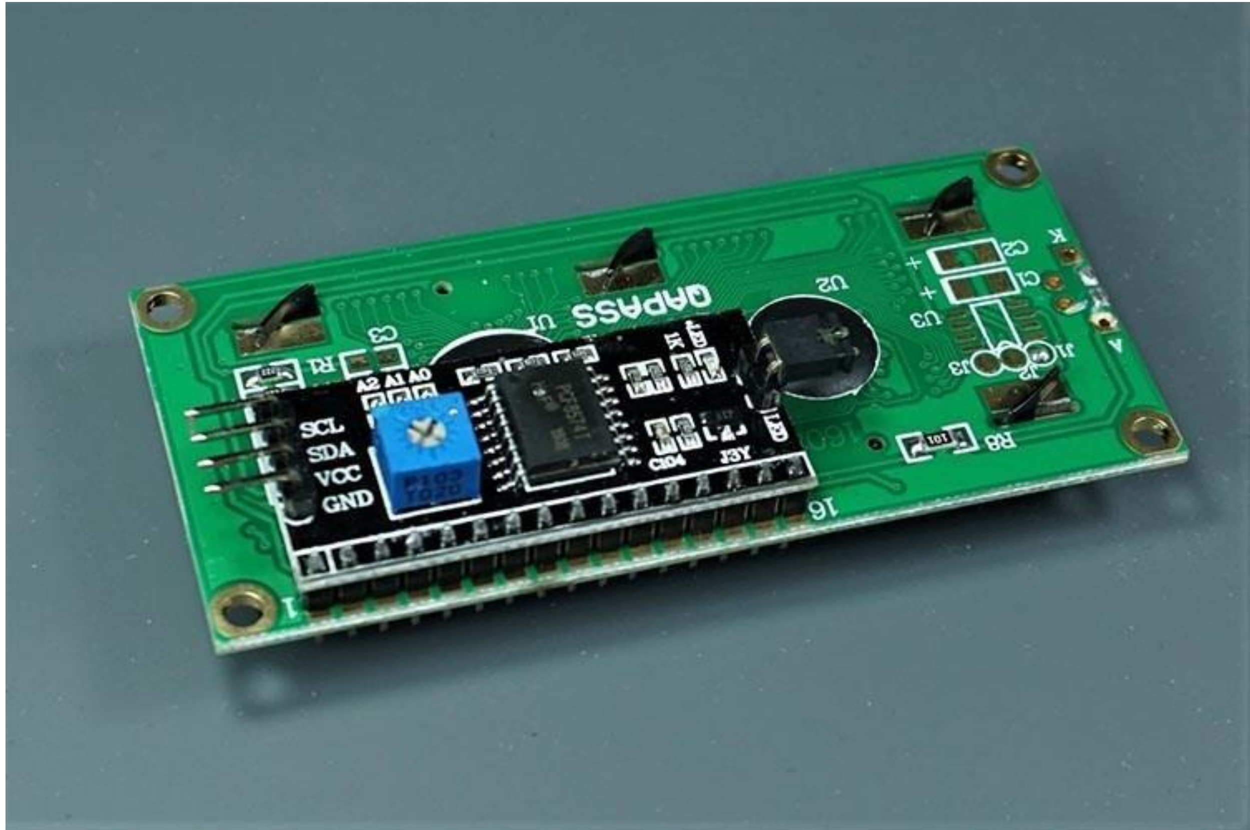


Figure 10: I2c module interface LCD pins to I2c protocol

Wire your LCD to the Arduino by following the schematic diagram in figure 11. We're using the Arduino default I2C pins (SCL and SDL).

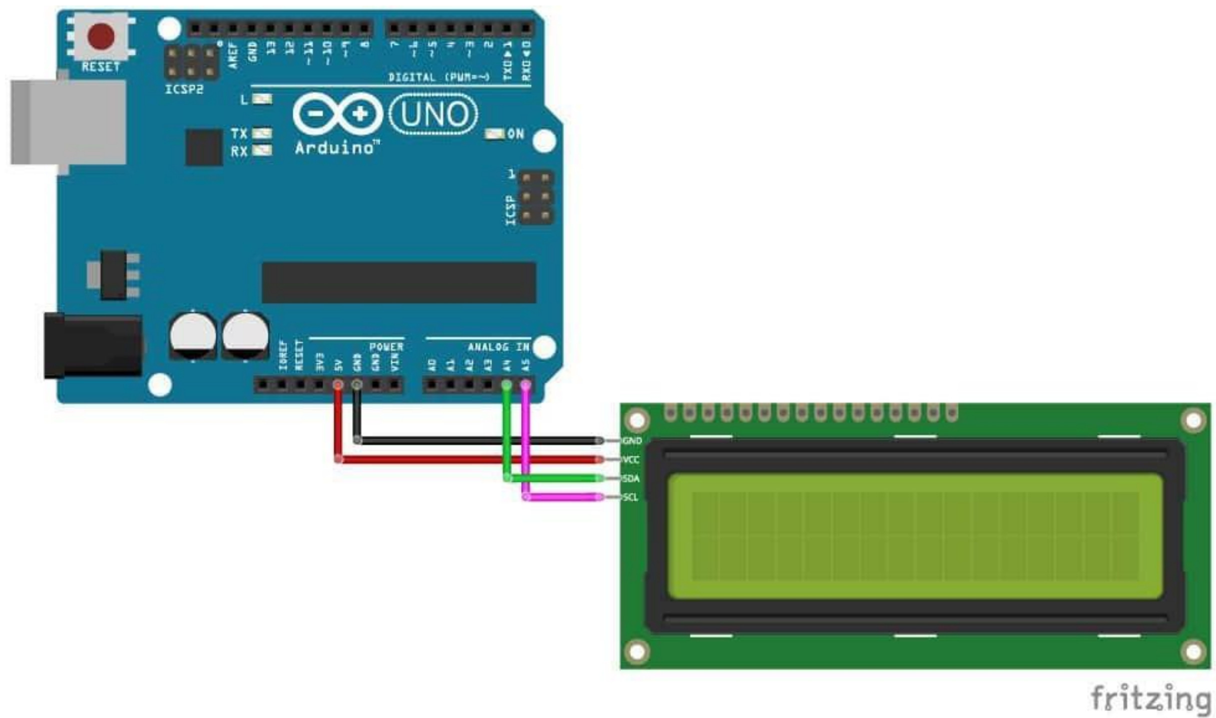


Figure 11: I2c module connection between Arduino and LCD

You can also use the following table as a reference.

I2C LCD	Arduino
GND	GND
VCC	VCC
SDA	SDA
SCL	SCL

Preparing the Arduino IDE

Before proceeding with the project, you need to install the Arduino add-on in the Arduino IDE.

Follow one of the next guides to prepare your Arduino to use the LCD functions:

Installing the LiquidCrystal_I2C Library

There are several libraries that work with the I2C LCD. We're using this library by Marco Schwartz. Follow the next steps to install the library:

1. Click here to download the LiquidCrystal_I2C library. You should have a .zip folder in your Downloads
2. Unzip the *.zip* folder and you should get LiquidCrystal_I2C-master folder
3. Rename your folder from ~~LiquidCrystal_I2C-master~~ to LiquidCrystal_I2C
4. Move the LiquidCrystal_I2C folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Getting the LCD Address (Address scanning)

Before displaying text on the LCD, you need to find the LCD I2C address. With the LCD properly wired to the Arduino, upload the following I2C Scanner sketch.

```
#include <Wire.h>

void setup() {
  Wire.begin();
  Serial.begin(9600);
  Serial.println("\nI2C Scanner");
}

void loop() {
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++ ) {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address<16) {
        Serial.print("0");
      }
      Serial.println(address,HEX);
      nDevices++;
    }
    else if (error==4) {
      Serial.print("Unknow error at address 0x");
      if (address<16) {
```

```
        Serial.print("0");
      }
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
  }
  else {
    Serial.println("done\n");
  }
  delay(5000);
}
```

After uploading the code, open the Serial Monitor at a baud rate of 9600. Press the Arduino reset button. The I2C address should be displayed in the Serial Monitor. (like in figure 12).

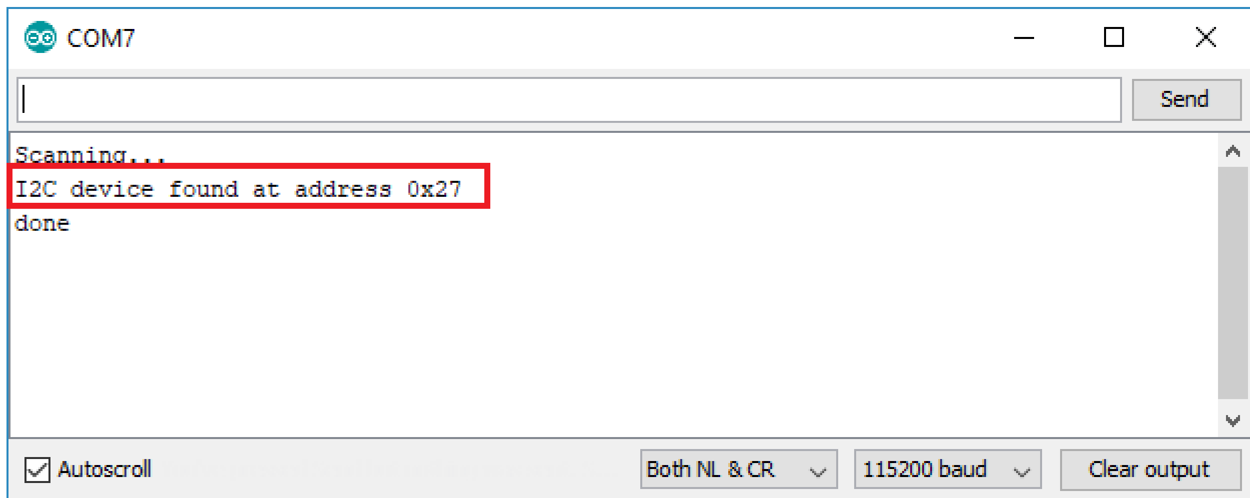


Figure 12: I2c scanner output

In this case the address is 0x27, but it could be any other address. Write down the address of your I2C module.

Part 2.2: Display static text on the LCD

Displaying static text on the LCD is very simple. All you have to do is select where you want the characters to be displayed on the screen, and then send the message to the display.

Here's a very simple sketch example that displays "`Welcome TO BZU`".

```
#include <LiquidCrystal_I2C.h>

// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;

// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

void setup(){
  // initialize LCD
  lcd.init();
  // turn on LCD backlight
  lcd.backlight();
}

void loop(){
  // set cursor to first column, first row
  lcd.setCursor(0, 0);
  // print message
  lcd.print("Welcome TO");
  delay(1000);
  // clears the display to print new message
  lcd.clear();
  // set cursor to first column, second row
  lcd.setCursor(0,1);
  lcd.print("BZU”);
  delay(1000);
  lcd.clear();
}
```

It displays the message in the first row, and then in the second row.

In this simple code we show you the most useful and important functions from the LiquidCrystal_I2C library. So, let's take a quick look at how the code works.

How the code works

First, you need to include the LiquidCrystal_I2C library.

```
#include <LiquidCrystal_I2C.h>
```

The next two lines set the number of columns and rows of your LCD display. If you're using a display with another size, you should modify those variables.

```
int lcdColumns = 16;
int lcdRows = 2;
```

Then, you need to set the display address, the number of columns and number of rows. You should use the display address you've found in the previous step.

```
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
```

In the setup(), first initialize the display with the init() method.

```
lcd.init();
```

Then, turn on the LCD backlight, so that you're able to read the characters on the display.

```
lcd.backlight();
```

To display a message on the screen, first you need to set the cursor to where you want your message to be written. The following line sets the cursor to the first column, first row.

```
lcd.setCursor(0, 0);
```

Note: 0 corresponds to the first column, 1 to the second column, and so on…

Then, you can finally print your message on the display using the print() method.

```
lcd.print("Welcome TO ");
```

Wait one second, and then clean the display with the clear() method.

```
lcd.clear();
```

After that, set the cursor to a new position: first column, second row.

```
lcd.setCursor(0,1);
```

Then, the process is repeated.

So, here's a summary of the functions to manipulate and write on the display:

- lcd.init(): initializes the display
- lcd.backlight(): turns the LCD backlight on

- lcd.setCursor(intcolumn, int row): sets the cursor to the specified column and row
- lcd.print(message): displays the message or variable on the display
- lcd.clear(): clears the display

This example works well to display static text no longer than 16 characters.

Part 2.3: I2C communication between 2 Arduinos

Figure 13 show you how to connect 2 devices to use I2C communication, in this case we set one of them as master and other as slave.

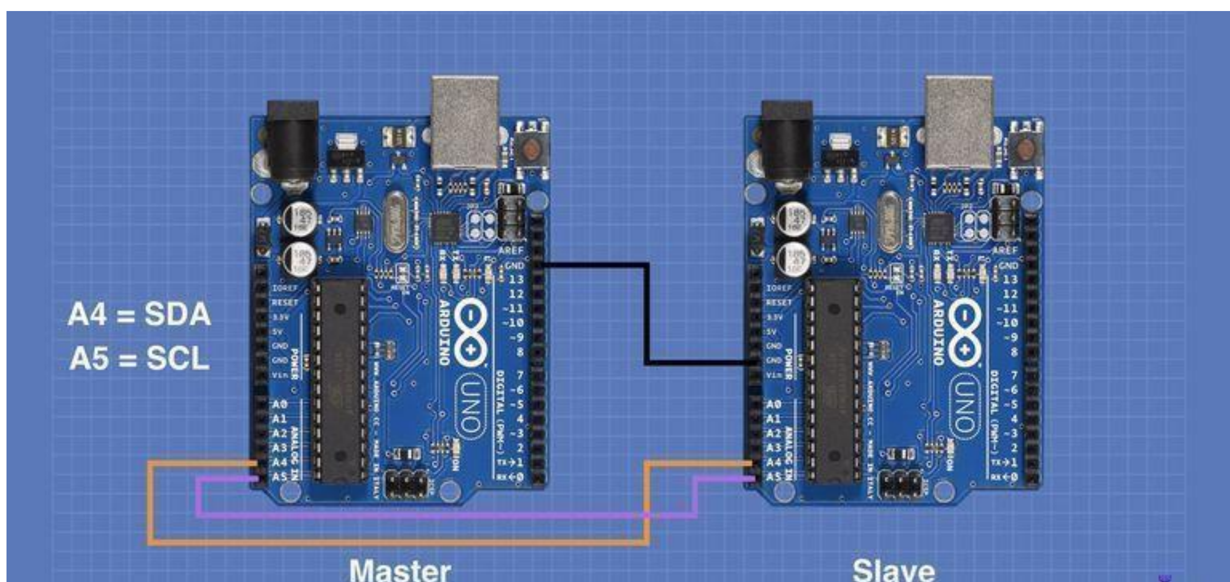


Figure 13: I2c connections

Our example will send data from slave to master. The data is characters starting from '0' and ending with 'z'.

To run the codes:

1. Do the schematic in figure 13. (make sure to make the GND common).
2. Compile and upload the master code to the first Arduino.
3. Compile and upload the slave code to the second Arduino.
4. Open the serial monitor for the master (make sure to change it since the monitor could be on the slave Arduino).

Master code:

```
// master side code
#include <Wire.h>
void setup()
{
 Wire.begin();        // join i2c bus (address optional for master)
 Serial.begin(9600);  // start serial for output
 Serial.print("master sleeping...");
 delay(2000);
 Serial.println("go");
}

void loop()
{
 Wire.requestFrom(2, 1);    // request data from slave device #2

 while(Wire.available())
 {
   char c = Wire.read(); // receive a byte as character
   Serial.print(c);         // print the character
 }

 delay(100);
}
```

Slave code:

```
// slave side code
#include <Wire.h
void setup()
{
 Wire.begin(2);                // join i2c bus with address #2
 Wire.onRequest(requestEvent); // register event
 Serial.begin(9600);           // start serial for output
}

void loop()
{
 delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void requestEvent()
{
```

```
 static char c = '0';

 Wire.write(c++);
 if (c > 'z')
   c = '0';
}
```

Part 2.4: I2C communication between 2 Arduinos and LCD

In this part we want to read analog value from slave Arduino and write the value on LCD connected to Arduino master device:

Steps:

1. Do the same schematic in figure 13 (last part).
2. To connect LCD to the master, flip the Arduino and try to find SCL and SDA written on the back of some Arduino pins! Connect them to LCD.
3. Connect a potentiometer to the slave device on A0.
4. Compile and upload the master code to the first Arduino.
5. Compile and upload the slave code to the second Arduino.
6. Open the serial monitor for the master (make sure to change it since the monitor could be on the slave Arduino).

Master code:

```
// master side code
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// set the LCD number of columns and rows
int lcdColumns = 16;
int lcdRows = 2;

// set LCD address, number of columns and rows
// if you don't know your display address, run an I2C scanner
sketch
LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);

void setup()
{
  Wire.begin();      // join i2c bus (address optional formaster)
  Serial.begin(9600);  // start serial for output
  Serial.print("master sleeping...");
  delay(2000);
  Serial.println("go");
```

```
  // initialize LCD
  lcd.init();
  // turn on LCD backlight
  lcd.backlight();
}

void loop()
{
  Wire.requestFrom(50, 2);    // request data from slave device
#2
  int res;//result from I2c reading
  byte MSB = Wire.read(); /* receive a byte MSB(NOTE: the
function is blocking, so it would not continue the code until a
byte reach)*/
  byte LSB = Wire.read(); /* receive a byte LSB(NOTE: the
function is blocking, so it would not continue the code until a
byte reach)*/
  res = ((MSB << 8) | LSB);
  Serial.print("MSB = ");
  Serial.print(MSB);
  Serial.print(" , LSB = ");
  Serial.print(LSB);
  Serial.print(" , analog value = ");
  Serial.println(res);          // print the analog value

  // set cursor to first column, first row
  lcd.setCursor(0, 0);

  // print message
  lcd.print("Reading = ");
  lcd.print(res);
  delay(1000);
  lcd.clear();

}
```

Slave code:

```
// slave side code
#include <Wire.h>
void setup()
{
 Wire.begin(50);                // join i2c bus with address #2
 Wire.onRequest(requestEvent); // register event
 Serial.begin(9600);           // start serial for output
}

void loop()
{
 delay(100);
}

// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void requestEvent()
{
  int value = analogRead(A0);
  Wire.write(value >> 8); // send the MSB
  Wire.write(value & 0x00ff); // send the LSB
}
```

Part 2.5. I2C communication between Arduino and Temperature sensor (LM75B)

In this part, we will see how we can make the communication between Arduino and **LM75B** sensor using I2C. You have to test the following example for the LM75B temperature sensor. Note that the LM75B Address not given and you have to find it using the theory information.

```
//Simple code for the LM75B, simply prints temperature via serial
#include <Wire.h>
int LM75BAddress = ??????;
void setup(){
Serial.begin(9600);
Wire.begin(); //communication start
}

void loop(){
float celsius = getTemperature();
Serial.print("Celsius: ");
Serial.println(celsius);
float fahrenheit = (1.8 * celsius) + 32;
Serial.print("Fahrenheit: ");
Serial.println(fahrenheit);
delay(200); //just here to slow down the output. You can remove this
}
float getTemperature(){
Wire.requestFrom(LM75BAddress,2);
byte MSB = Wire.read();
byte LSB = Wire.read();
//it's a 12bit int, using two's compliment for negative
int TemperatureSum = ((MSB << 8) | LSB) >> 4;
float celsius = TemperatureSum*0.0625;
return celsius;
}
```