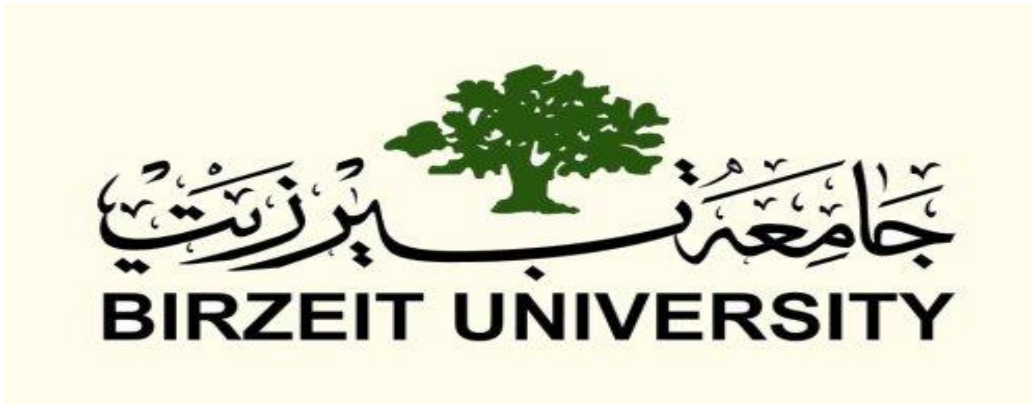
بسم الله الرحمن الرحيم



Faculty of Engineering & Information Technology

Interfacing Laboratory - ENCS 412

Experiments 1-3 Report

Arduino, I2C, Multiplexed Display & LabVIEW

Instructor: Dr. Ahmad Afaneh

Teacher Assistant: Eng. Mohammed Modallal

Name: Hussein Dahir, **ID:** 1131138

Partner's Name: Yazeed Obaid, **ID:** 1130036

Partner's Name: Maher Saleem, **ID:** 1130258

Section: 2

Date: 10 March 2017

Abstract:

These three experiments aim to introduce the students to Arduino UNO and some sensors that we can use with it, like photocells and LM75B temperature sensor. They also aim to introduce the students to the I2C interface and to the idea of display multiplexing. In experiment 3, LabVIEW is introduced too, which allows students to make simulation for many systems with no need for real sensors or hardware, which is reliable and can be used if one have just a computer.

Contents

Theory 5
- Arduino 5
  - What is Arduino? 5
  - Why Arduino? 5
- Photocells 5
  - What is photocell? 5
  - How to measure light using a photocell? 5
- Interrupts 6
  - What is an interrupt? 6
  - Hardware interrupts 6
  - Software interrupts 6
- I2C 8
  - What is I2C? 8
  - I2C Library 8
- LM75B Temperature Sensor 8
- Seven Segment Display 9
- Multiplexed Display 10
- LabVIEW 10

Procedure & Discussion 11
- Experiment 1 Tasks 11
  - Photocells 11
  - Hardware Interrupts 13
  - Software Interrupts 14
  - To Do 16
- Experiment 2 Tasks 17
  - LM75B Temperature Sensor 17
  - Seven Segment Display 19
  - To Do 21
- Experiment 3 Tasks 25
  - Simple Alarm System 25
  - Simple Liquid Store System 26
  - Lowpass and Highpass filters in time domain 27
  - To Do 29

Conclusion 30
References 31

Theory:

Arduino [1]:

What is Arduino?
Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Why Arduino?
Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics.

Photocells:

What is photocell? [2]

A photocell (or light-dependent resistor, LDR, or photoresistor) is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibits photoconductivity. A photoresistor can be applied in light-sensitive detector circuits, and light- and dark-activated switching circuits. (see figure 1).

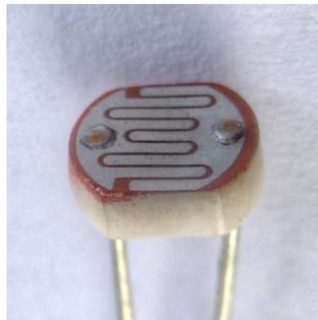


Figure 1: Photocell

How to measure light using a photocell? [3]

The easiest way to measure a resistive sensor is to connect one end to Power and the other to a pull-down resistor to ground. Then the point between the fixed pulldown resistor and the variable photocell resistor is connected to the analog input of a microcontroller such as an Arduino (see figure 2).

Using Voltage Divider Rule:

Vo = Vcc (R / (R + Photocell))

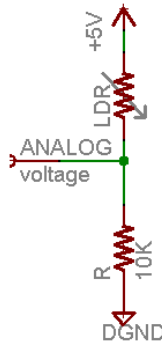


Figure 2: How to connect a photocell

That is, the voltage is proportional to the inverse of the photocell resistance which is, in turn, inversely proportional to light levels.

Interrupts [3]:

What is an interrupt?

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a small program called an interrupt handler (or interrupt service routine, ISR) to deal with the event. This interruption is temporary, and after the interrupt handler finishes, the processor resumes execution of the previous thread. There are two types of interrupts: hardware interrupts and software interrupts.

Hardware interrupts:

A hardware interrupt is an electronic alerting signal sent to the processor from an external device. Arduino UNO board has two external interrupts: int. 0 (on digital pin 2) and int. 1 (on digital pin 3). When the triggering event is captured at these pins, the corresponding interrupt service routine is executed.

The attachInterrupt() API is used to specify a function to call when an external interrupt occurs. Its syntax is as follows:

attachInterrupt (interruptNumber, ISR_name, mode_constant)

which attaches the interrupt with the ISR called “ISR_name” to the interrupt “interruptNumber” in mode “mode_constant”.

Software interrupts:

A software interrupt is caused either by an exceptional condition in the processor itself, or a special instruction in the instruction set which causes an interrupt when it is executed. We will use Timer1 only here, Timer1 is 16-bit hardware timer on the ATmega168/328. The timer can be programmed by some special registers, which you can use to configure the pre-scaler for the timer, Timer1's clock speed is defined by setting this pre-scaler, this pre-scaler can be set to 1, 8, 64, 256 or 1024.

You can change the Timer behavior through the timer registers. The most important timer registers are:

TCCRx - Timer/Counter Control Register. The pre-scaler can be configured here.

TCNTx - Timer/Counter Register. The actual timer value is stored here.

TIMSKx - Timer/Counter Interrupt Mask Register. To enable/disable timer interrupts.

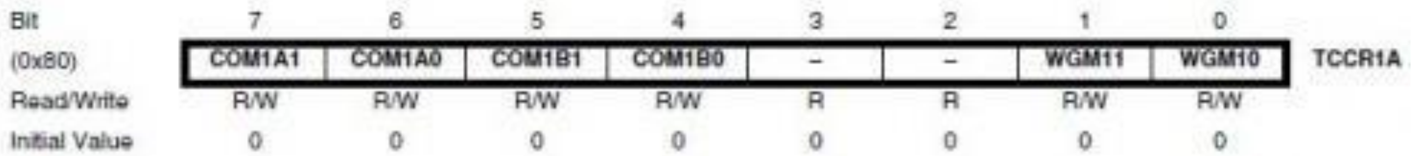


Figure 3: TCCR1A Register

Figure 4: TCCR1B Register

Table 1 shows the values for registers CS12, CS11 and CS10 needed for the corresponding pre-scalers and other settings.

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Table 1: Register Values for Clock Selection

I2C:

What is I2C? [5]

The I2C (I-two-C) protocol was invented by Philips. In I2C the serial data transmission is done in asynchronous mode. This protocol uses only two wires for communicating between two or more ICs. The two bidirectional open drain lines named SDA (Serial Data) and SCL (Serial Clock) with pull up resistors are used for data transfer between devices.

One of the two devices, which controls the entire process, is known as Master and the other which responds to the queries of master is known as Slave device. The ACK (acknowledgement) signal is sent/received from both the sides after every transfer and hence reduces the error. SCL is the clock line bus used for synchronization and is controlled by the master. SDA is known as the data transfer bus.

I2C Library [6]:

This library allows you to communicate with I2C devices. On the Arduino boards with the R3 layout, the SDA (data line) and SCL (clock line) are A4 (SDA), A5 (SCL).

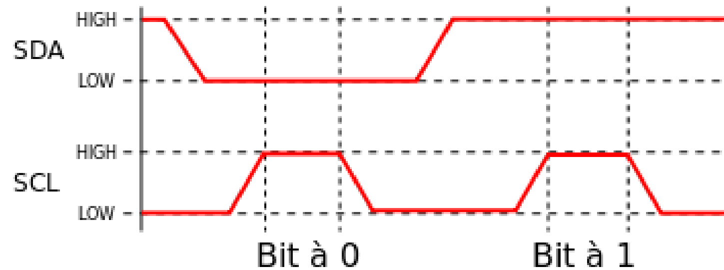


Figure 5: One Clock Pulse Per Data Bit

LM75B Temperature Sensor [6]:

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.

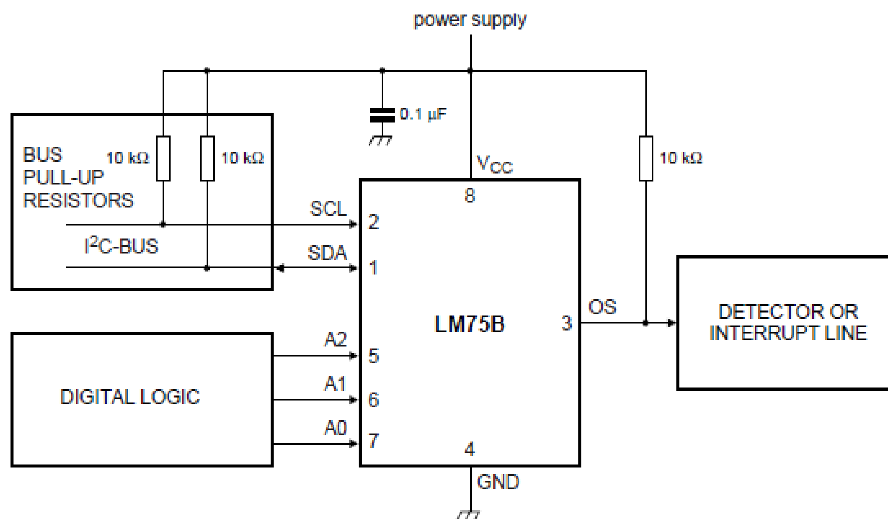


Figure 6: LM75B Schematic

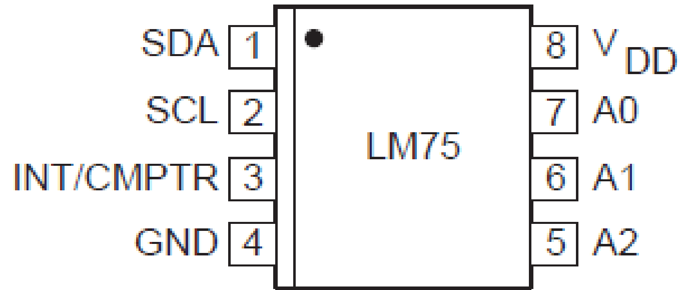


Figure 7: LM75B Pins

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

Table 2: LM75B Pin Description

Note: 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. In this lab, (A2, A1, A0) are connected to the ground, so that they are zeros, and so the address is 48h.

1	0	0	1	A2	A1	A0
MSB						LSB

Figure 8: LM75B Address

Seven Segment Display [6]:

A seven-segment display, or seven-segment indicator, is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot-matrix displays. Seven-segment displays are widely used in digital clocks, electronic meters, and other electronic devices for displaying numerical information.

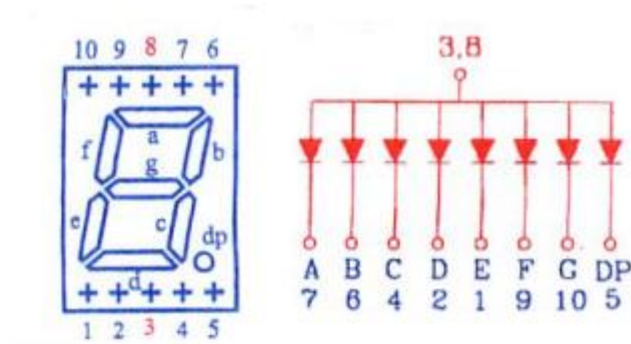


Figure 9: Pin Mappings of Common Anode Seven Segment Display

Multiplexed Display [6]:

Multiplexed display are electronic displays where the entire display is not driven at one time. Instead, sub-units of the display (typically, rows or columns for a dot matrix display or individual characters for a character orientated display, occasionally individual display elements) are multiplexed, that is, driven one at a time, but the electronics and the persistence of vision combine to make the viewer believe the entire display is continuously active.

LabVIEW [7]:

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical programming environment. So it relies on graphical components to implement different systems and different programing scripts. There are many applications use the Lab-View in different fields: science, education and industry. In this Experiment, the students will write programs to become familiar with some of the basics of LabVIEW.

Procedure & Discussion:

Experiment 1 Tasks:

Photocells:

First, we tested the LED to make sure that it is working properly, so, we connected the circuit as shown in figure 10.

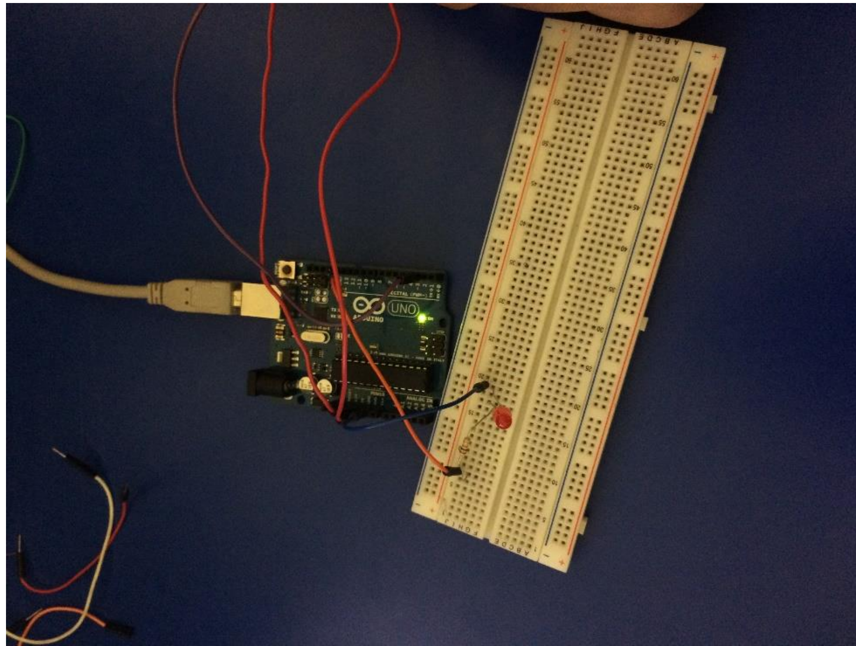


Figure 10: LED Connected for testing purposes

Then, we connected the LDR with the LED, as shown in figures 11 & 12.

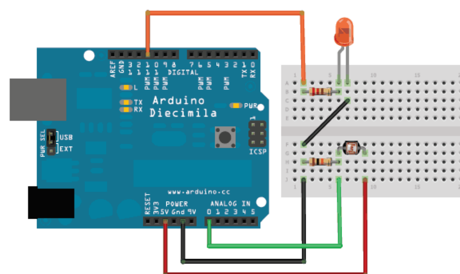


Figure 11: LDR connected to Arduino UNO with a LED

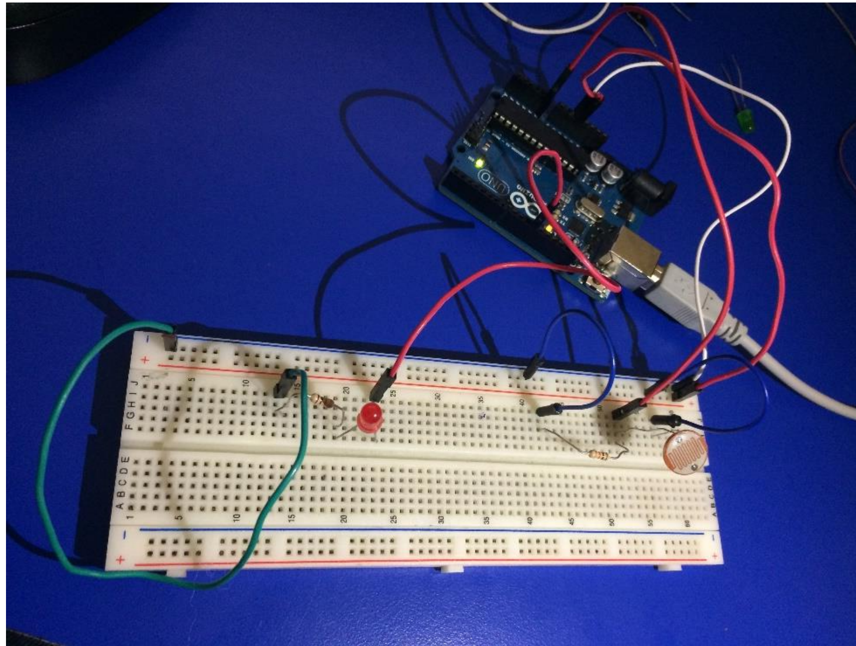


Figure 12: LDR connected to Arduino UNO with a LED

Code Used:

```
int photocellPin = 0;     // the cell and 10K pulldown are connected to a0. Analog
bin NO. 0
int photocellReading;     // the analog reading from the sensor divider
int LEDpin = 11;          // connect Red LED to pin 11 (PWM pin)
int LEDbrightness;        // the brightness of the LED

void setup(void) {
  Serial.begin(9600);
}

void loop(void) {
  photocellReading = analogRead(photocellPin);
  Serial.print("Analog reading = ");
  Serial.print(photocellReading);     // the raw analog reading
  if (photocellReading < 500)         // Analog read is 10 bits, so rang is 0-1023.
We choose 500.
    Serial.println(" - Dark");
    else
      Serial.println(" - bright");

  // LED gets brighter the darker it is at the sensor
  // that means we have to -invert- the reading from 0-1023 back to 1023-0
  photocellReading = 1023 - photocellReading;


  //now we have to map 0-1023 to 0-255 since that's the range analogWrite uses
  LEDbrightness = map(photocellReading, 0, 1023, 0, 255);
  analogWrite(LEDpin, LEDbrightness);
  delay(10000);                       // Delay 10 seconds
}
```

Code Explanation:

This code reads the analog voltage from the LDR (range 0-1023), shows it on the serial monitor, and decides if it is dark or bright, depending on a threshold. The threshold here was set to 500, if the reading is less than 500, then it is dark, otherwise, it is bright. The LED brightness is the inverse of the LDR reading, so that the brightness increases as it is darker, and vice versa. We had to map the LDR reading from (0-1023) to (0-255), since analog write uses this range (i.e. 8 bits), and finally we just wrote this brightness value (see figure 13).

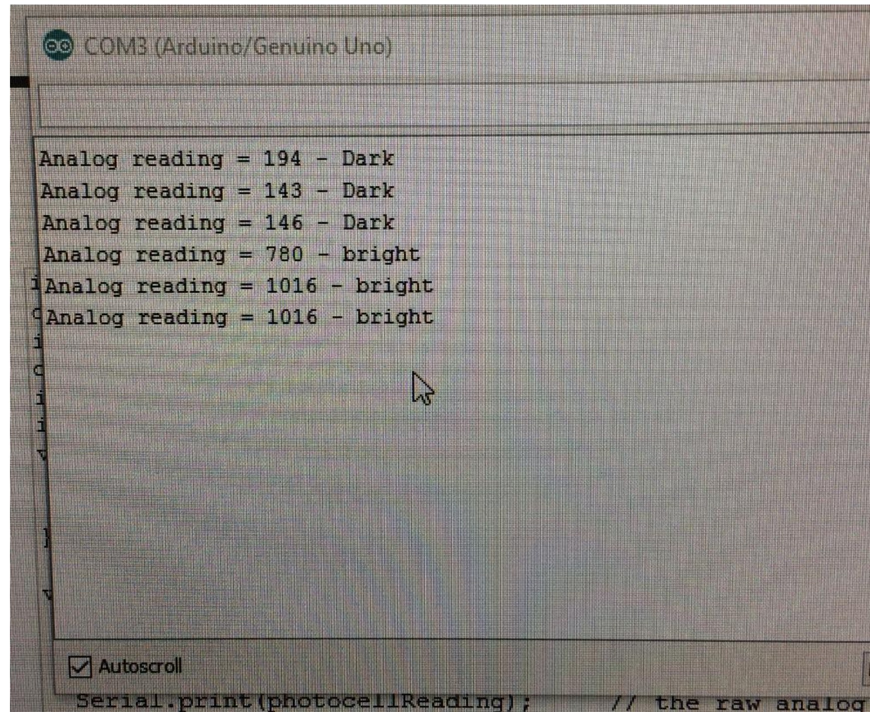


Figure 13: Readings from photocell

Hardware Interrupts:

In this part, a push button was used to trigger the hardware interrupt, i.e. give a signal to the needed pin to inform the Arduino that an interrupt has occurred. It was connected as shown in figure 14.

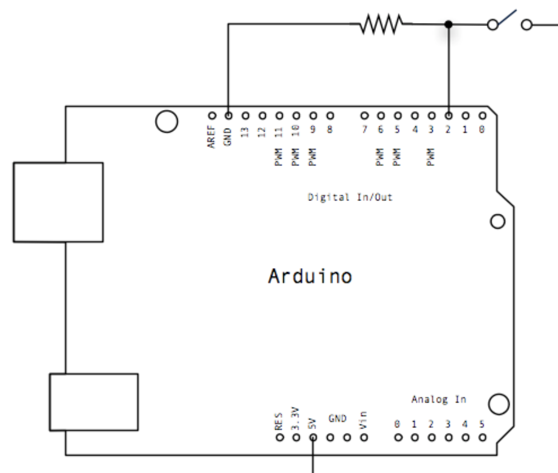


Figure 14: Connection of push button [4]

The value at pin 2 is zero by default, if the push button has pressed, then 5 volts will be connected to the resistor, and to pin 2 too, making a change from 0 to 5 volts, which causes a hardware interrupt, and so the ISR attached to interrupt 0 (which corresponds to pin 2) will be executed.

Code Used:

```
// Hardware interrupt example
int pin = 13;
volatile int state = LOW;

void setup() {
      pinMode(pin, OUTPUT);
      attachInterrupt(0, blink, CHANGE);
}

void loop() {
      digitalWrite(pin, state);
}

void blink() {
      state = !state;
}
```

Code Explanation:

Initially, state is LOW, that is the LED connected to pin 13 is off, function “blink” is attached to interrupt 0 (pin 2), with mode set to “CHANGE”, that is, the function “blink”, will be executed each time the value at pin 2 is changed, from 0 to 1, or from 1 to 0 (1 corresponds to 5 volts). So, when the push button is pressed (and held pressed), the value at pin 2 will change from 0 to 1, triggering the function “blink” which in turn will invert the state from LOW to HIGH, making the LED emit, if the button released, then the value at pin 2 will change again from 1 to 0, triggering the function “blink” again, i.e. inverting the state again, and so, turning off the LED.

Software Interrupts:

In this part, a software interrupt was programmed to occur with frequency of 2 Hertz (i.e. every 0.5 second), and that’s using Timer1.

To get a frequency of 2 Hertz using Timer1, we need to substitute in the timer equation as follows:

$$needed\ value = 2^{16} - \frac{16\ MHz}{prescaler \times frequency}$$

Substituting frequency = 2 Hz, and trying pre-scaler = 256, with give us a value of 34286, which is acceptable, since it is less than 65536 (the max value for a 16-bit timer). If the value was not acceptable, then what we had to do was just trying a larger pre-scaler.

When the timer counts from 34286 to 65536, it will overflow causing a software interrupt to occur.

Code Used:

```
// Timer1 overflow interrupt example
#define ledPin 13

void setup() {
  // Defining status of LED pin to be output
  pinMode(ledPin, OUTPUT);


  // initialize timer1
  noInterrupts();           // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;


  TCNT1 = 34286;            // preload timer 65536-16MHz/256/2Hz
  TCCR1B |= (1 << CS12);    // 256 pre-scaler
  TIMSK1 |= (1 << TOIE1);   // enable timer overflow interrupt
  interrupts();             // enable all interrupts
}


// interrupt service routine that wraps a user defined function supplied by
attachInterrupt

ISR(TIMER1_OVF_vect) {
  TCNT1 = 34286;            // preload timer
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}


void loop() {
  // your program here...
}
```

Code Explanation:

A LED is connected to the Arduino UNO using pin 13, the code starts with configuring the timer to make an interrupt every 0.5 second. In setup() function we can see that first interrupts are disabled using noInterrupts() function until we configure our timer, then TCCR1A & TCCR1B registers are set to zero, TCNT1 register holds the value was found using equation above, 34286, TCCR1B is then set by OR operation with (1 << CS12), i.e. shifting left value 1 for CS12 times, so that we set the bit that corresponds to CS12 (as mentioned in table 1). Similar operation is done with TIMSK1 register to enable timer overflow interrupt by setting the bit that corresponds to TOIE1 (Timer Overflow Interrupt Enable 1). Finally, interrupts are enabled again using interrupts() function. Now, when the timer counter reaches 65536, which will cause an overflow that leads an interrupt to occur, the ISR will be executed, which will preload the timer with value 34286 again, so that the process keeps repeating (if we didn’t preload it, the interrupt will occur once only). The ISR will also take a reading from pin 13 (LED reading), negate it, and write it back to the pin, which makes the LED turns on and off each 0.5 second.

To Do:

In this part, we need to generate a square wave with frequency 1 KHz. To do so, we just need to use exactly the previous code but with different value for the timer, since we have a different frequency here, substituting 256 for pre-scaler and 1KHz for frequency in the timer equation, will give us the value 65474 (which will be put in TCNT1), and it is acceptable since it is less than 65536. To see the square wave on the oscilloscope, we just need to connect the oscilloscope to pin 13 of the Arduino.

Experiment 2 Tasks:

LM75B Temperature Sensor:

In this part, the LM75B sensor is being tested with a simple program that reads the temperature and print it on the serial monitor in both Celsius and Fahrenheit.

Code Used:

```
//Simple code for the LM75B, simply prints temperature via serial

// Importing Wire library for I2C wire
#include <Wire.h>

// Address of sensor, A2A1A0 = 000, since they are connected to GND, so address is
1001000 in binary.

int LM75BAddress = 0x48;


void setup(){
  Serial.begin(9600);
  Wire.begin();                               //communication start
}

void loop(){
  // Reading temperature

  float celsius = getTemperature();
  Serial.print("Celsius: ");
  Serial.println(celsius);

  // Converting to Ferhenhit!
  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(){

  // Sending address of sensor to I2C bus and sending address.
  // The function requestFrom(), send bit bit data, it sends first most significant
bit first, then next to
  // most and so no.

  Wire.requestFrom(LM75BAddress,2);

  // The function, read() read byte byte.
  byte MSB = Wire.read();
  byte LSB = Wire.read();
```

```
  // it's a 12bit int, using two's compliment for negative.
  // integer in Arduino is 12 bits. So first we shift MSB to 8-bits. Then we OR with
  // LSB to get full 16-bit number. Since integer is 12-bits, we need to convert 16-
bits
  // to 12-bits. So, we remove least significant four bits.

  int TemperatureSum = ((MSB << 8) | LSB) >> 4;

  float celsius = TemperatureSum*0.0625;
  return celsius;
}
```

Code Explanation:

As we mentioned earlier, A2, A1 and A0 pins for the LM75B sensors used in this lab are connected to the ground, so they are zeros and the address for the sensor is 1001000 in binary (48h). A simple function called "getTemperature" was used to get the temperature reading from the sensor and return it in Celsius. In this function, we first specify the address that will be used for the sensor with I2C (to read data from), using the sentence Wire.requestFrom(LM75BAddress,2), and we also specify that the data will come in two segments (two bytes), most significant byte then least significant byte. So, these two bytes are read and stored in MSB and LSB variables, respectively. The two data segments are then merged into one data value, but since integer in Arduino is only 12 bits, we need to get rid of four bits, which will be the least significant four bits using shift and OR operation. Then the temperature is multiplied by a constant and returned to the main loop function, which in turn prints it in Celsius and Fahrenheit to the serial monitor.

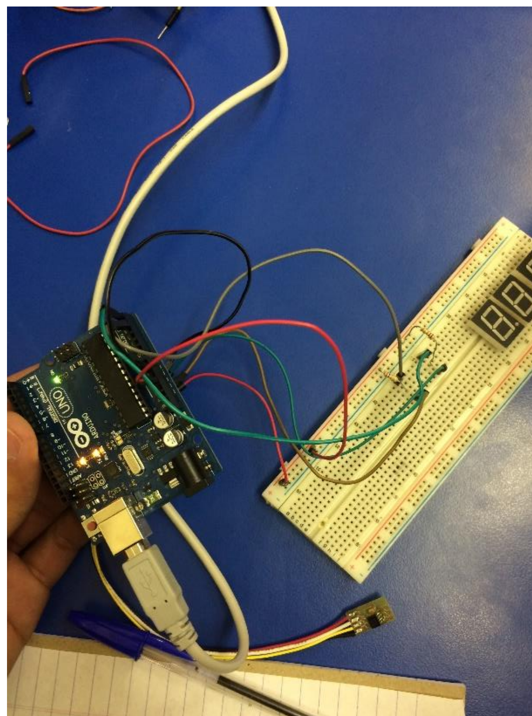


Figure 15: LM75B Sensor connected to the Arduino UNO

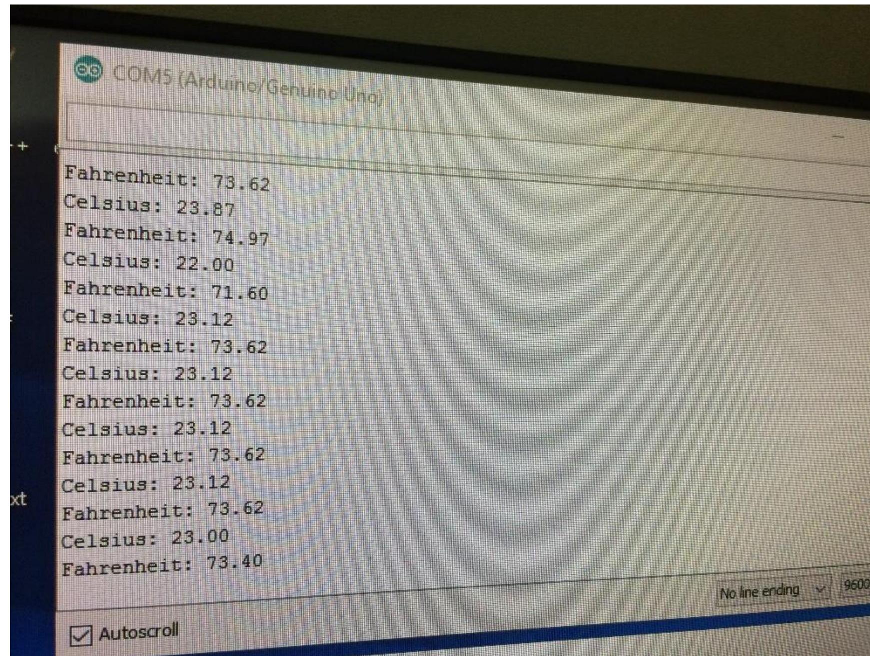


Figure 16: Temperature Readings on Serial Monitor

Seven Segment Display:

In this part, the seven segment display is being tested with a simple counter program that counts from nine to zero.

<u>Code Used:</u>

```
// Define the LED digit patters, from 0 - 9
// Note that these patterns are for common anode displays, common anode is active
low. Since we connect all pins to high
// so other pin we connect it to GND to enable it.
// Arduino pin: 2,3,4,5,6,7,8
                                    //A,B,C,D,E,F,G
byte seven_seg_digits[10][7] = {    { 0,0,0,0,0,0,1 },  // = 0
                                    { 1,0,0,1,1,1,1 },  // = 1
                                    { 0,0,1,0,0,1,0 },  // = 2
                                    { 0,0,0,0,1,1,0 },  // = 3
                                    { 1,0,0,1,1,0,0 },  // = 4
                                    { 0,1,0,0,1,0,0 },  // = 5
                                    { 0,1,0,0,0,0,0 },  // = 6
                                    { 0,0,0,1,1,1,1 },  // = 7
                                    { 0,0,0,0,0,0,0 },  // = 8
                                    { 0,0,0,0,1,0,0 }   // = 9
                               };
```

```
byte count=0;

void setup() {
  // Digital pins 2,...,9. Other pins are connected to GND, two pins, and other is
connected to 3.3volts
  // 5 volts didn't output a correct result, so we used 3.3 volts.
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  writeDot(1);             // start with the "dot" off
}

// Function to enable or disable dot.
void writeDot(byte dot) {
  digitalWrite(9, dot);
}

// Writing to seven segment.
void sevenSegWrite(byte digit) {
  byte pin = 2;
  for (byte segCount = 0; segCount < 7; ++segCount) {
    digitalWrite(pin, seven_seg_digits[digit][segCount]);
    ++pin;
  }
}


// Main function
void loop() {
      for (byte count = 10; count > 0; --count) {
            delay(1000);
            sevenSegWrite(count - 1);
      }
      delay(1000);
}
```

Code Explanation:

seven_seg_digits is an matrix with a row for each digit (each row contain seven values for the seven segments). writeDot is just a function to enable or disable the dot in the seven segment display, sevenSegWrite function is used to write a digit (given as parameter) to the seven segment, bit by bit, pin by pin. In the main loop function, we are just writing the digits from nine down to zero, one by one, with one second delay between any two consecutive digits.

To Do:

In this part, we take a reading from the LM75B temperature sensor and display it using three 7-segment displays, in the form dd.d.

<u>Code Used:</u>

```
// Importing Wire library for I2C wire
#include <Wire.h>

// Address of sensor, A2A1A0 = 000, by defult they are connected to GND, so address
is 1001000
int LM75BAddress = 0x48;

// Define the LED digit patters, from 0 - 9
// Note that these patterns are for common anode displays, common anode is active
low. Since we connect all pins to high
// so other pin we connect it to GND to enable it.
// Arduino pin: 2,3,4,5,6,7,8
                                      //A,B,C,D,E,F,G
byte seven_seg_digits[10][7] = {      { 0,0,0,0,0,0,1 },  // = 0
                                      { 1,0,0,1,1,1,1 },  // = 1
                                      { 0,0,1,0,0,1,0 },  // = 2
                                      { 0,0,0,0,1,1,0 },  // = 3
                                      { 1,0,0,1,1,0,0 },  // = 4
                                      { 0,1,0,0,1,0,0 },  // = 5
                                      { 0,1,0,0,0,0,0 },  // = 6
                                      { 0,0,0,1,1,1,1 },  // = 7
                                      { 0,0,0,0,0,0,0 },  // = 8
                                      { 0,0,0,0,1,0,0 }   // = 9
                               };

byte count=0;
byte dig1,dig2,dig3;

void setup() {

   Serial.begin(9600);
    Wire.begin();                               //communication start

  // Digital pins 2,...,9. Other pins are connected to GND, two pins, and other is
connected to 3.3volts
  // 5 volts didn't output a correct result, so we used 3.3 volts.
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  writeDot(1);                          // start with the "dot" off
}
```

```
// Function to enable or disable dot.
void writeDot(byte dot) {
  digitalWrite(9, dot);
}

// Writing to seven segment.
void sevenSegWrite(byte digit) {
  byte pin = 2;
  for (byte segCount = 0; segCount < 7; ++segCount) {
    digitalWrite(pin, seven_seg_digits[digit][segCount]);
    ++pin;
  }
}

float getTemperature(){

  // Sending address of sensor to I2C bus and sending address.
  // The function requestFrom(), send bit data, it sends first most significant bit
first, then next to
  // most and so no.
  Wire.requestFrom(LM75BAddress,2);

  // The function, read() read byte.
  byte MSB = Wire.read();
  byte LSB = Wire.read();

  //it's a 12bit int, using two's compliment for negative.
  // integer in Arduino is 12 bits. So first we shift MSB to 8-bits. Then we OR with
  // LSB to get full 16-bit number. Since integer is 12-bits, we need to convert 16-
bits
  // to 12-bits. So we remove least significant four bits.
  int TemperatureSum = ((MSB << 8) | LSB) >> 4;

  // Constant = 0.0625
  float celsius = TemperatureSum*0.0625;
  return celsius;
}
```

```
void loop() {
  int value = 5;

  // We used 11, 12, & 13 for the enables of the three seven segments. We connect Vcc
of seven segments
  // to these pins.
  for(int i = 0; i < 3; ++i) {

    // Reading temperature
    float celsius = getTemperature();
    Serial.print("Celsius: ");
    Serial.println(celsius);
    int x = celsius * 10;
    int d0,d1,d2;

    d0 = x % 10;
    d1 = x / 10 % 10;
    d2 = x / 100;


    digitalWrite(11, LOW);
    digitalWrite(12, LOW);
    digitalWrite(13, LOW);
    if (i==0){
        digitalWrite(11, HIGH);
        sevenSegWrite(d2);
        writeDot(1);
        if (!temp){
          temp = 10;}
        else{temp--; }
        //Serial.println(temp);
        delay(value);


    }
    else if (i==1){
        digitalWrite(12, HIGH);

        sevenSegWrite(d1);
        writeDot(0);
        if (!temp){
          temp = 10;}
        else{temp--; }
        delay(value);
    }
    else {
        digitalWrite(13, HIGH);
        sevenSegWrite(d0);
        writeDot(1);
        if (!temp){
          temp = 10;}
        else{temp--; }
        delay(value);
    }
    //delay(100);
  }
}
```

Code Explanation:

In this code, we just used the codes from the previous two parts, but we used display multiplexing, that is we connected the three 7-segment displays together with the Arduino UNO, so that any data sent from Arduino will reach the three 7-segment displays, but one of them only will be enabled and will display the received data. This process will be repeated for the three digits periodically with proper delay setting so that the viewer won't notice that the displays values are changing.

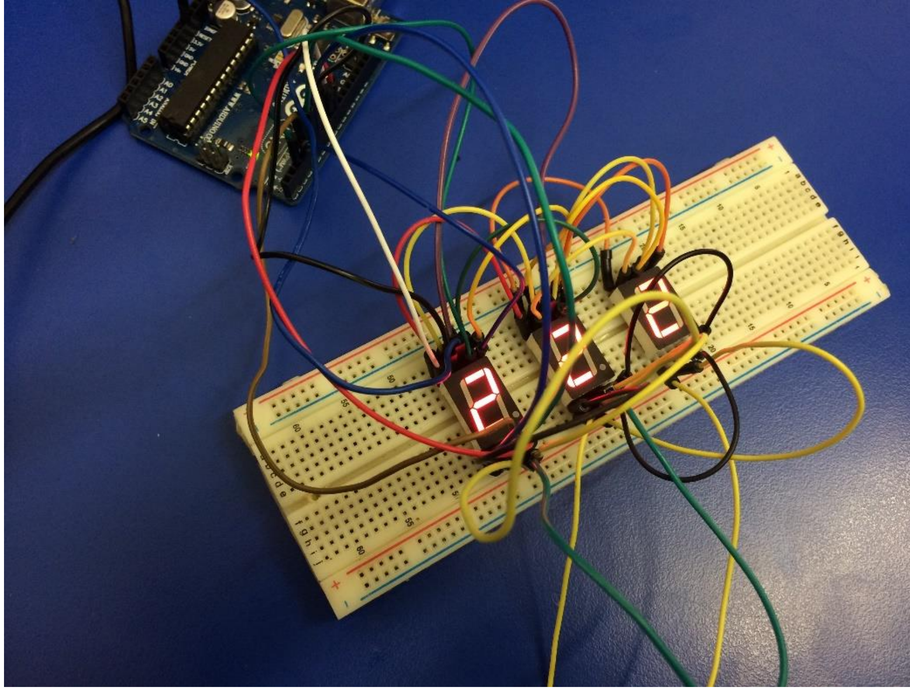


Figure 17: Three 7-Segment Displays Connected

Experiment 3 Tasks:

Simple Alarm System:

We need to design a control system with the following specifications:

1- The system has two inputs: pressure and temperature.
2- There is an alarm when the temperature exceeds 100° C or the pressure exceeds 15KPa.
3- The transfer function of the temperature and pressure transducers are 3.2mV/ °C, 0.3V/KPa respectively.

Figure 18: Simple Alarm System VI Front Panel

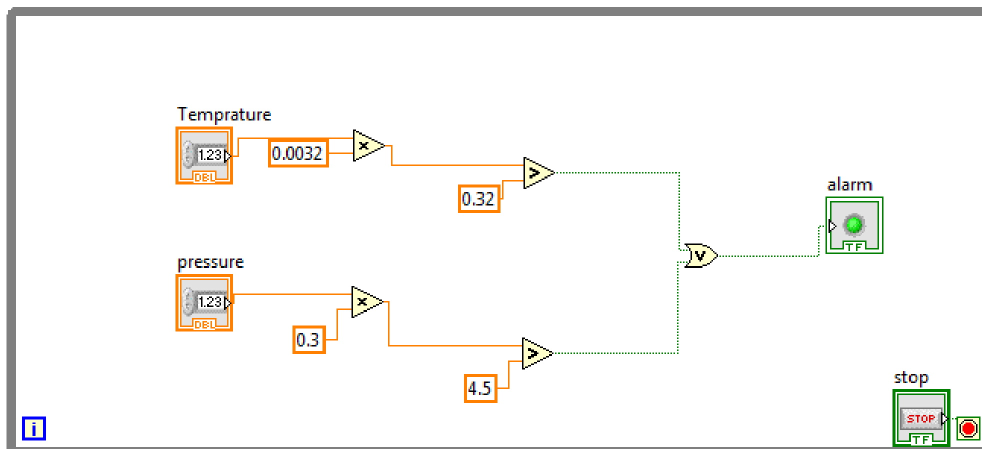


Figure 19: Simple Alarm System VI Block Diagram

Explanation:

This system is very simple, we just need to compare the value of the temperature and the pressure, if any of them exceeds the threshold, the alarm will have turned on. We did that by first multiplying temperature by 0.0032, and then comparing it with 0.32, note that if the temperature is 100 degrees, then 100*0.0032 = 0.32, with is the threshold we are using here. Same method used with pressure. An OR operation was used between the two comparisons, so that if only one of the two measures exceeds the threshold, then the alarm is on.

Simple Liquid Store System:

Design a simple liquid store system in the LABVIEW with the following specifications:

1- The system has four inputs (volume of the liquid, temperature of the liquid and two enables) and two outputs (led and screen).
2- When the volume of the liquid exceeds 6 liters and led enable is on, the led will turn into Red color.
3- When the temperature of the liquid exceeds 60° C and temperature enable is on, the message “The temperature of the liquid is high” should be displayed on the screen.

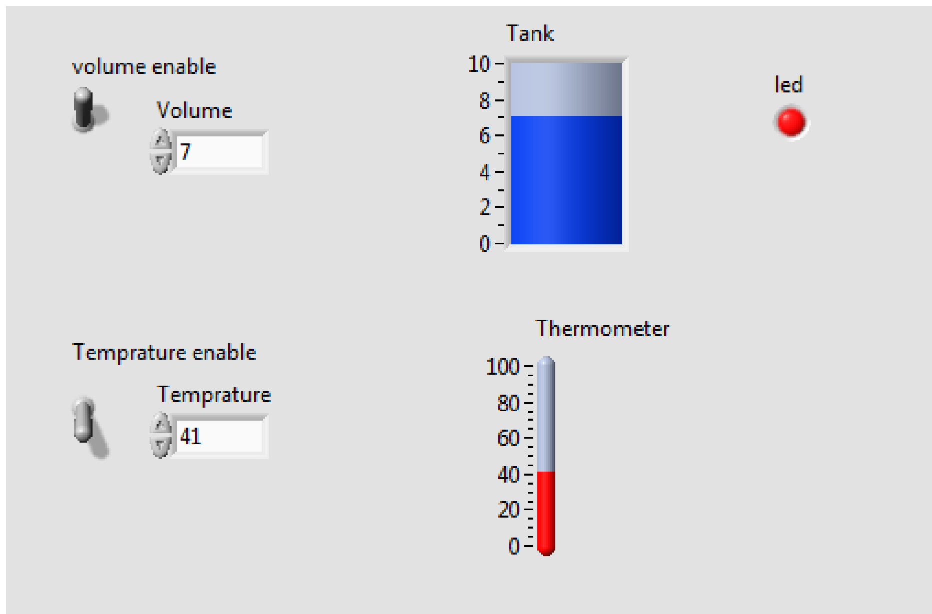


Figure 18: Volume Exceeds 6 Liters, LED is On

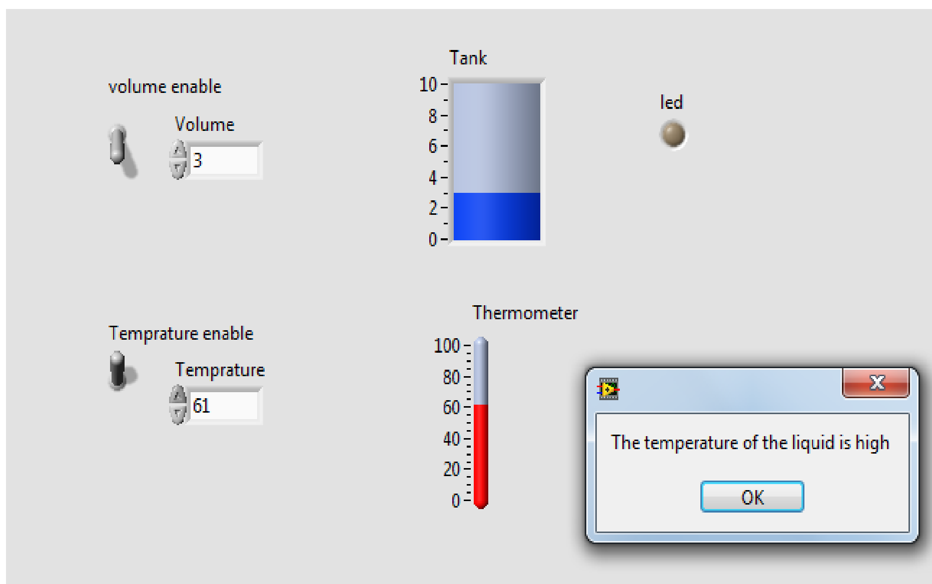


Figure 19: Temperature Exceeds 60 Degrees, Alert Message Shown

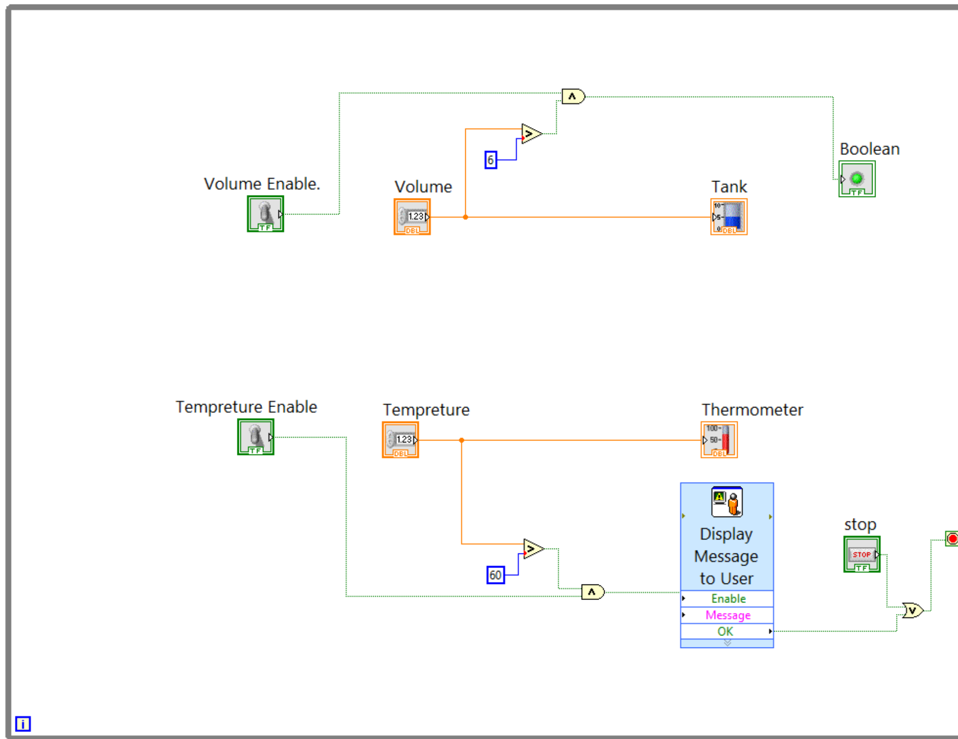


Figure 20: Simple Liquid Store System VI Block Diagram

Explanation:

The idea used here is similar to the previous part. We used a comparator for the volume to compare it with 6 liters, connected to an AND gate with the volume enable, so that the LED won't turn on unless the volume exceeds 6 liters and the volume enable is switched is enabled. Same thing was done with the thermos-meter, but with different threshold, and with different alert type, which was a pop-up window displaying an alert to the user.

Note: We have connected the display message to and OR gate with the stop button, so that if the window was showing, then the program will terminate. We had to do this, because if we didn't, then the alert window will keep displaying.

Lowpass and Highpass filters in time domain:

1- Build Lowpass and Highpass filters VI, then explain the result.
2- For lowpass filter try to make the cutoff frequency less than the signal frequency and explain the result in your report.
3- For highpass filter try to make the cutoff frequency greater than the signal frequency and explain the result in your report.

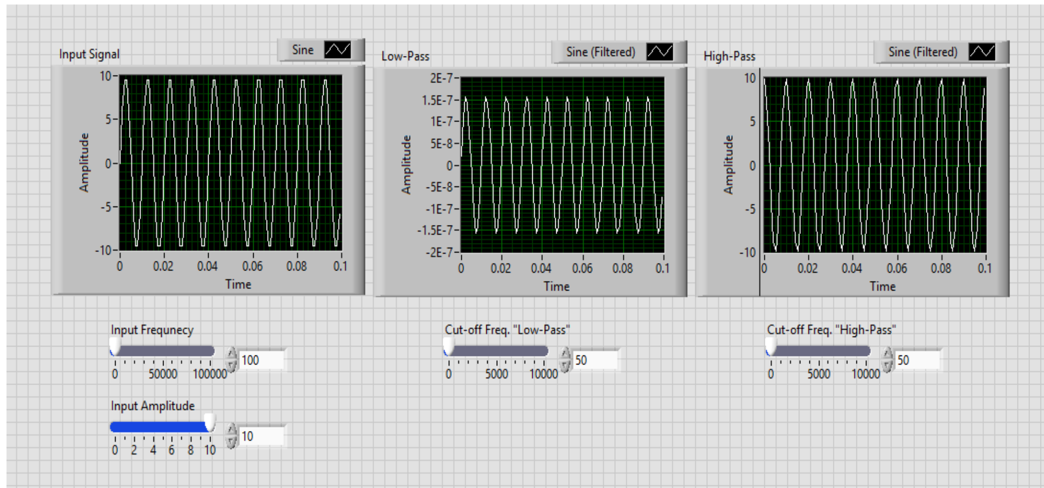


Figure 21: Lowpass & Highpass Filters VI Front Panel

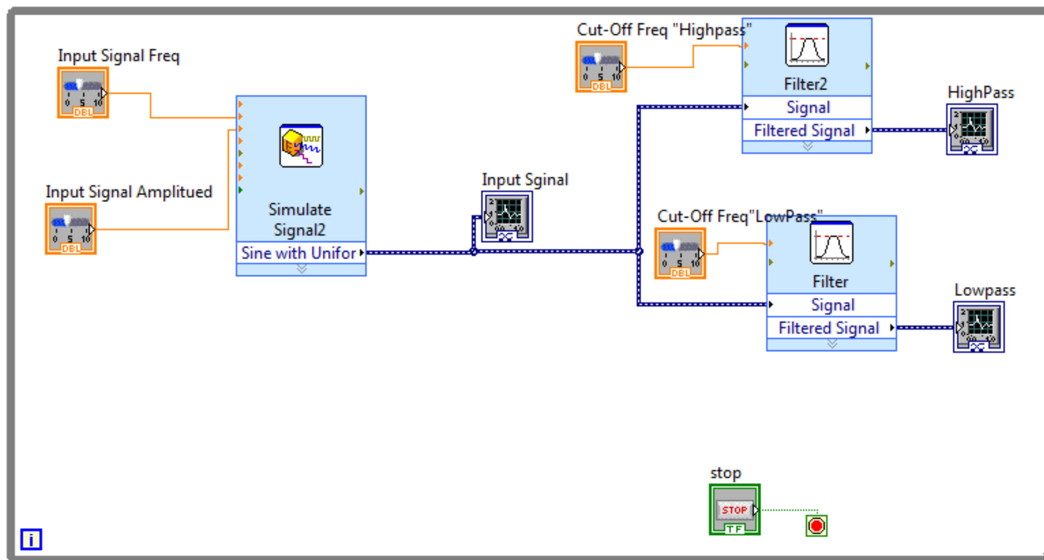


Figure 22: Lowpass & Highpass Filters VI Block Diagram

<u>Explanation:</u>

We built both lowpass and highpass filters as shown in the previous two figures. The input signal's amplitude and frequency can be changed by the user in the front panel. As shown in figure 21, we used a sine input signal of frequency of 100 and amplitude of 10, and with cutoff frequency of 50 for both filters. The result from the lowpass filter, was an attenuated sine signal (with amplitude of 1.5E-7, with is negligible), while it passed through the highpass filter without any attention, and that because the input sine signal has one single frequency which is 100, and since the lowpass filter attenuates any frequency greater than 100, it attenuates it, while the highpass filter didn't since it is in its passband region.

The opposite exactly has happened, when we set the cutoff frequency for both filters to 150, the signal was attenuated by the highpass filter, but it passed without any attenuation from the lowpass filter.

To Do:

1- Delete the simulate signal from previous block diagram and put Acquire Sound component in its place. See the wave form of your voice before and after the filter stage.
2- Connect the Play Waveform Component to a wire before filter stage to hear you voice. Then connect another instance of this component after Lowpass filter and another one after High Pass filter. What is the difference?

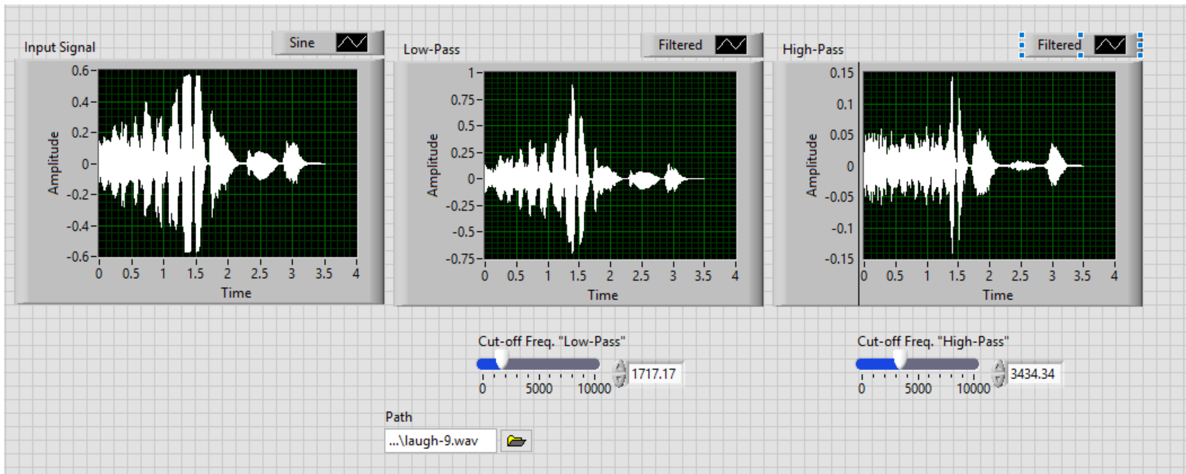


Figure 23: To Do VI Front Panel

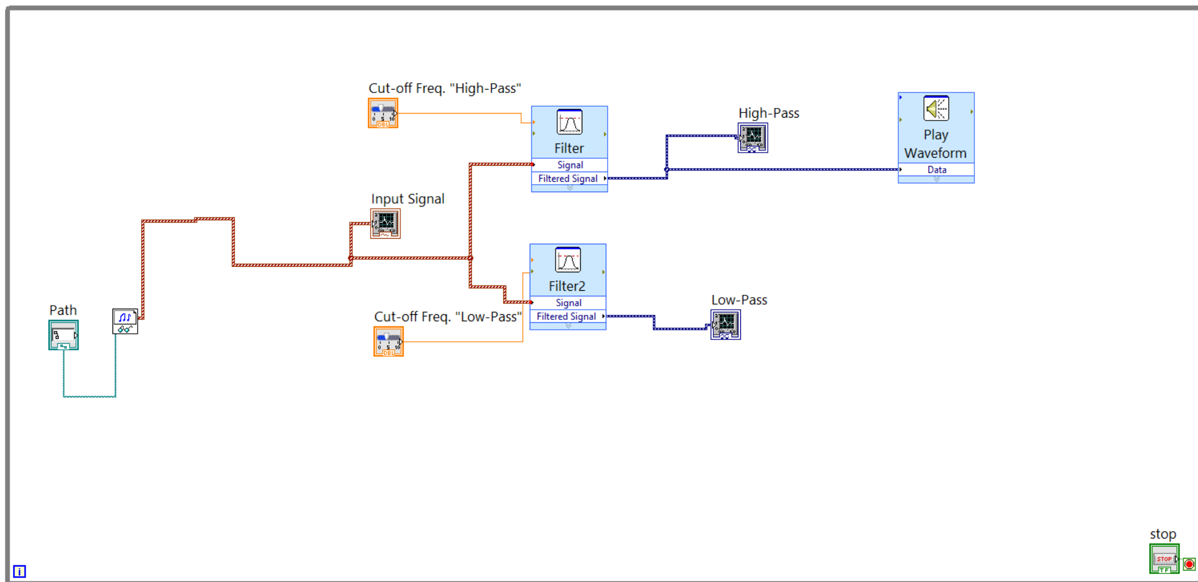


Figure 24: To Do VI Block Diagram

<u>Explanation:</u>

We noticed that most of our voice, is with low frequencies.

Conclusion:

These three experiments were a good start to the interfacing lab. Many things were introduced, including Arduino UNO, with a variety of sensors that used to build different systems. I2C interface was introduced also, which showed us that it is a very basic interface in any system, since it isn't only used on single boards, but also used to connect components which are linked via cable. Display multiplexing was also a very interesting and intelligent idea, since it uses less hardware and wires to do the same job that can be done without multiplexing. Finally, we can say that introducing LabVIEW maybe the best of all, since we can almost simulate any interfacing system using this program, with no need for any hardware.

References:

[1]: Arduino Guide – Introduction (5 March 2017).

https://www.arduino.cc/en/Guide/Introduction

[2]: Wikipedia – Photoresistor (5 March 2017).

https://en.wikipedia.org/wiki/Photoresistor

[3]: Interfacing Lab Manual - Experiment 1 (5 March 2017).

[4]: Figure Shows Push Button Connection (5 March 2017).

https://www.arduino.cc/en/Tutorial/Button

[5]: I2C Interface or TWI (Two Wire Interface) (9 March 2017).

https://www.engineersgarage.com/tutorials/twi-i2c-interface

[6]: Interfacing Lab Manual - Experiment 2 (9 March 2017).

[7]: Interfacing Lab Manual - Experiment 3 (9 March 2017).