CE1100 Introduction to Electrical and Computer Engineering

Lab Report #4: Temperature Sensor System with Buzzer
(Semester Project)

Laboratory Team Members: Mahir Rahman and Lecture Section #001,
Instructor: James M. Florence

Date: December 4, 2020

Abstract

In this report, we describe our design of an Arduino controlled system that turns different LEDs on to indicate different ranges of temperature on a temperature sensor (TMP36), and uses a buzzer to notify user of extremely low or high temperatures. The program for controlling the system was developed in TinkerCad. The system was successfully designed and when simulated, produced desired results.

Introduction

The system designed involves a temperature sensor (TMP36) that allows us to change the temperature from -40 °C to 125 °C. The Arduino program detects the corresponding voltage for the current temperature displayed in the sensor, converts the values from analogue to digital and lights up different LEDs (red, green, and yellow) in different patterns to inform the user of which temperature range the selected temperature is in. It also buzzes a buzzer for very high or low temperatures. A high frequency sound is played for temperatures above 100 °C, and a low frequency sound for temperatures below 0 °C. Temperature sensors measure the degree of coldness of a system. Temperature sensor systems are very important in laboratories, greenhouses, or any other facility that requires constant surveillance of the temperature of the surroundings. They enable us to record temperatures, and notify us if there is a sudden change in temperature in a controlled environment. Moreover, many commercial products like ammonia and sulfuric acid need their reactions to go at an optimum temperature. In systems such as this, temperature changes can lead to poor yield of desired product. There has been a lot of effort throughout history to measure temperature in a standard manner. The earliest evidence of it dates back to 176 A.D , when physician Claudius Galensus combined equal amount of ice and boiling water to create a “neutral” temperature standard. However, noticeable progress was not made until early 18th century when Gabriel Fahrenheit created the first thermometer that incorporated mercury and a scale to measure its calibrated height. The Fahrenheit scale is still in use today besides the Kelvin and Celsius scales. From that point on, different types of sensors developed. Out of those different types of sensors, TMP36, rose into prominence. The TMP36 sensor is a low voltage, precision centi-grade temperature sensor. This allows it to provide changes in voltage that are linear to changes in temperature. It usually records small changes in temperature (-40 °C to 125 °C) which means it is quite sensitive and can give accurate readings. These temperature sensors are mainly used in environmental control systems, thermal protection, industrial process control, fire alarms, power system monitors, and CPU thermal management. They can be connected to an Arduino to give analog readings which can be converted to digital value. The TMP-36 has three pins (Vout, Vin, and Ground). They use solid-state techniques and have no moving parts which accounts for its durability. The buzzer is a piezo-electric buzzer, and these types of buzzers were invented during the 1970s by the collaboration of different Japanese companies. They formed a group called the Barium Titanate Application Research Committee to accelerate the invention of these types of buzzers. These buzzers contain a piezo crystal that can deform its shape when voltage is applied across it. The crystal’s changes push a diaphragm which acts like a speaker cone to produce sound waves that can be perceived by human ears. An amplifier is usually connected to generate louder sounds or amplify voltage across the piezo-electric buzzer.

Procedure and Design

As mentioned previously, the program informs the user of the temperature range a selected temperature reading is in by measuring the analog voltage, calculating ADC values and turning on different LEDs in different ways. For instance, if the temperature sensor displays a value of 56 °C, the temperature range it will fall in is 40 to 59 °C. In this temperature range, the program will light up two green LEDs and one yellow LED. A temperature of 0 °C is in the range of 0 to 19 °C. Therefore, a green LED turns on only. Along with these changes in LED colors, a buzzer is attached that gives off a sound of low frequency for temperatures below 0 °C and a sound of high frequency for temperatures above 100 °C. The logic implemented here is that an analogue pin is set up as an input to receive voltage from TMP36. The temperature sensor produces different values of voltage depending on selected temperature. TMP36 produces and amplifies an analog output voltage that shares a proportional relationship with temperature. The readings taken in by input are converted from analogue to digital values. The ADC value is stored in a variable, and then checked against different conditions to see which voltage range it falls in. Depending on the voltage range, the Arduino program commands different LEDs to turn on. The hardware setup requires the Arduino Uno R3, breadboard, connecting wires, six LEDs (two green, two yellow, and two red), resistors of 220 Ω, a piezo-electric buzzer, and a multimeter to record voltage across temperature sensor. The diagram below shows how to set up everything up.

TMP36 with different colored LEDs

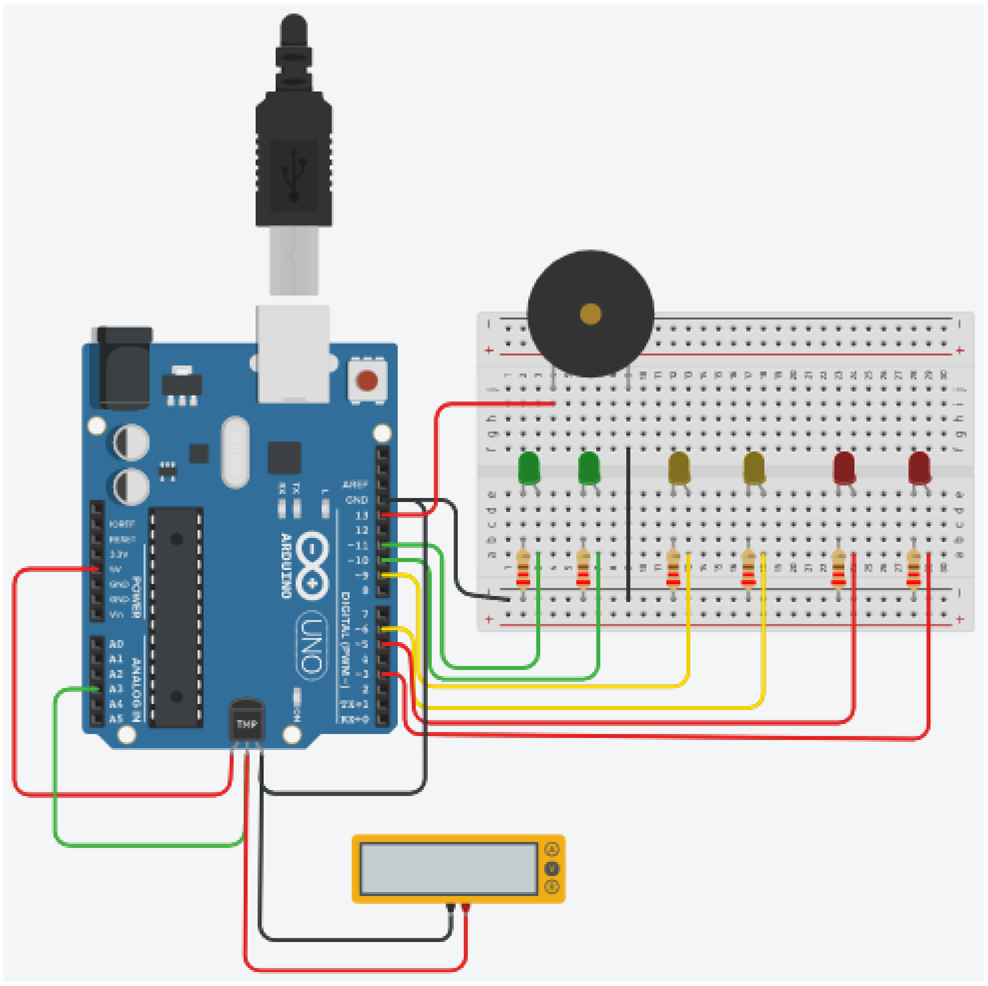
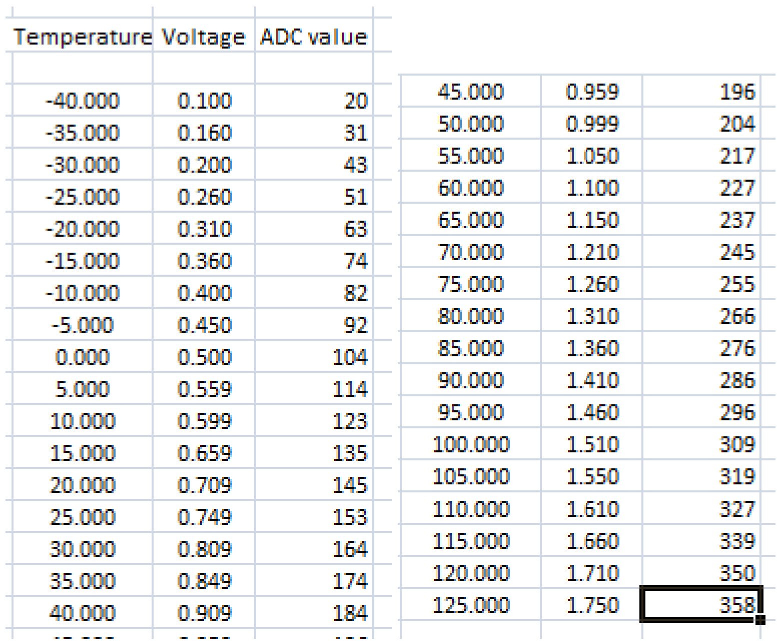


Figure 1 A TinkerCad layout including TMP36 and different LEDs

From Figure 1, we observe that the TMP36 temperature sensor is present at the bottom of the Arduino Uno R3. It is connected to a 5 V supply, and the voltage across it is used as input by the analog pin A3. The LEDs are placed on a breadboard, their cathode connected to the resistor and their anode connected to pins on the Arduino. Starting from the left in Figure 1, green LED 1, green LED 2, yellow LED 1, yellow LED 2, red LED 1, and red LED 2 are connected to pins 11, 10, 9, 6, 5 and 3 respectively. The temperature sensor has a voltmeter across its Vout and GND pin while the 5 V is connected to the power pin. The positive terminal of buzzer is attached to pin 13, and the negative terminal has been connected to ground.

Now that we are clear about the hardware set-up, we need to know the analog voltages and their corresponding ADC values that accompany each 5 °C change in temperature to program our Arduino. The table below shows us the collection of data.

A table displaying the data required for developing sensor system



Temperature	Voltage	ADC value
-40.000	0.100	20
-35.000	0.160	31
-30.000	0.200	43
-25.000	0.260	51
-20.000	0.310	63
-15.000	0.360	74
-10.000	0.400	82
-5.000	0.450	92
0.000	0.500	104
5.000	0.559	114
10.000	0.599	123
15.000	0.659	135
20.000	0.709	145
25.000	0.749	153
30.000	0.809	164
35.000	0.849	174
40.000	0.909	184
45.000	0.959	196
50.000	0.999	204
55.000	1.050	217
60.000	1.100	227
65.000	1.150	237
70.000	1.210	245
75.000	1.260	255
80.000	1.310	266
85.000	1.360	276
90.000	1.410	286
95.000	1.460	296
100.000	1.510	309
105.000	1.550	319
110.000	1.610	327
115.000	1.660	339
120.000	1.710	350
125.000	1.750	358

Table 1 Voltage, ADC, and Temperature values for TMP36

From Table 1, we observe the data collected to program our sensor system. It shows us the ADC values and analog voltage values from -40 °C to 125 °C for every 5 °C interval. This information helps us construct the graphs below. The relationship between analog voltage from TMP36, the ADC values and Temperature in °C is an important aspect of our system. The diagrams below help us understand the relationship.

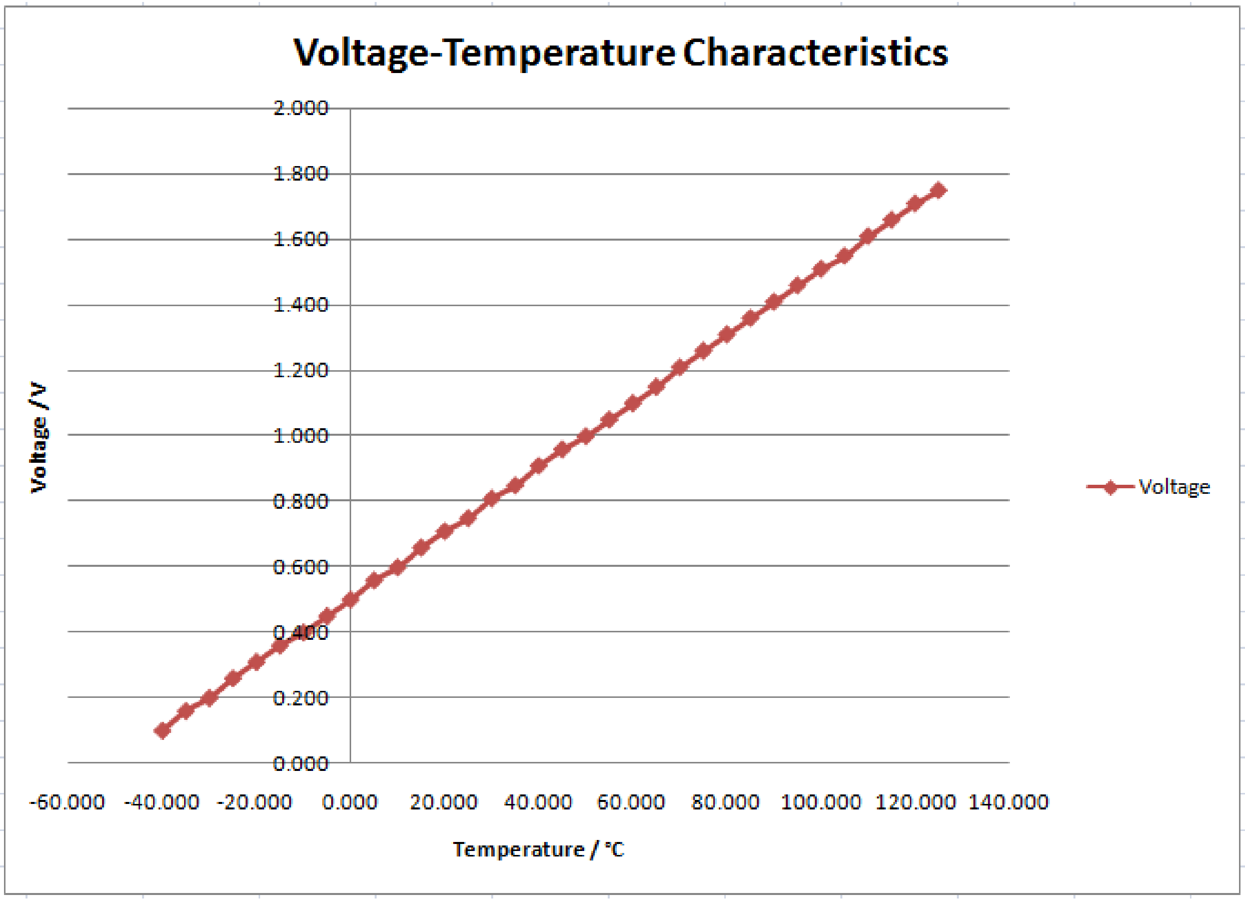


Figure 2 Voltage-Temperature Characteristics Graph

From Figure 2, we see a graph of voltage (in V) against temperature (in °C). The graph is a straight line. The TMP36's range is limited to – 40 to 125 °C. However, in that range the analog voltage displays a linear relationship with temperature. As they have such a relationship, we can use the analog voltage to make a temperature sensor system. Each temperature gives a different voltage reading while remaining proportional. This makes it easier to write a program that performs fixed commands when within a certain range.

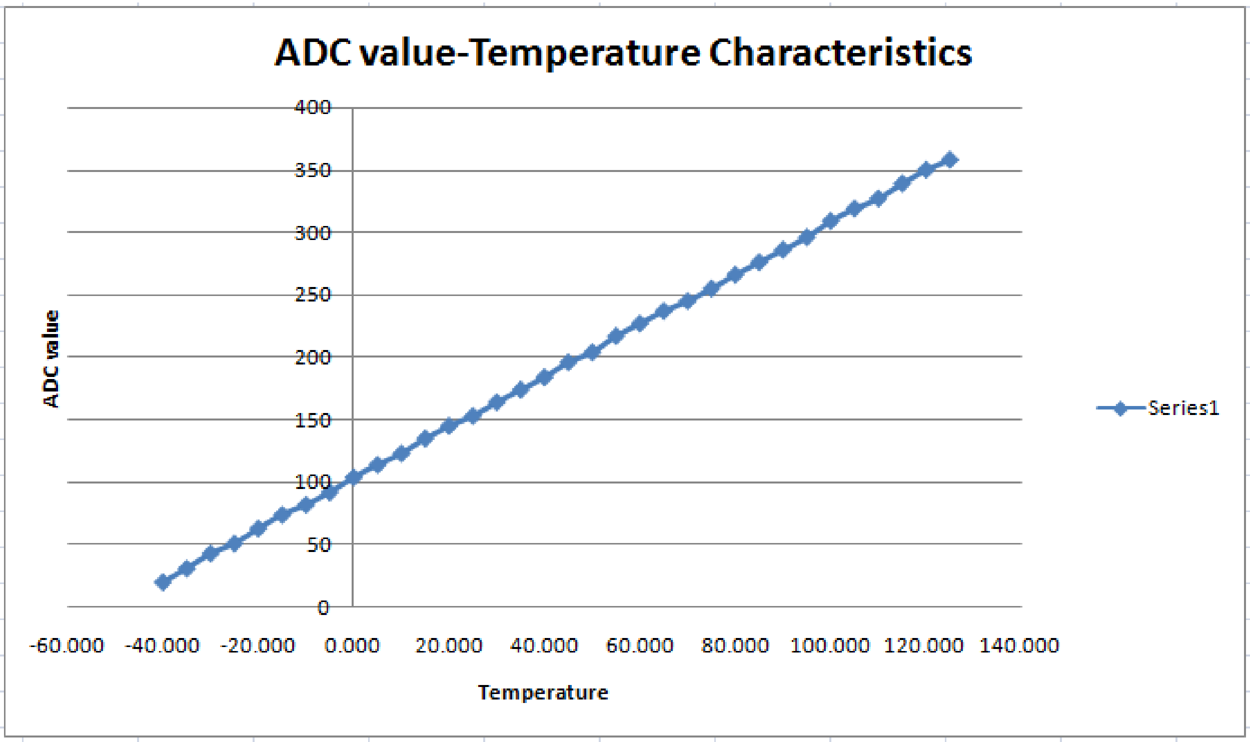
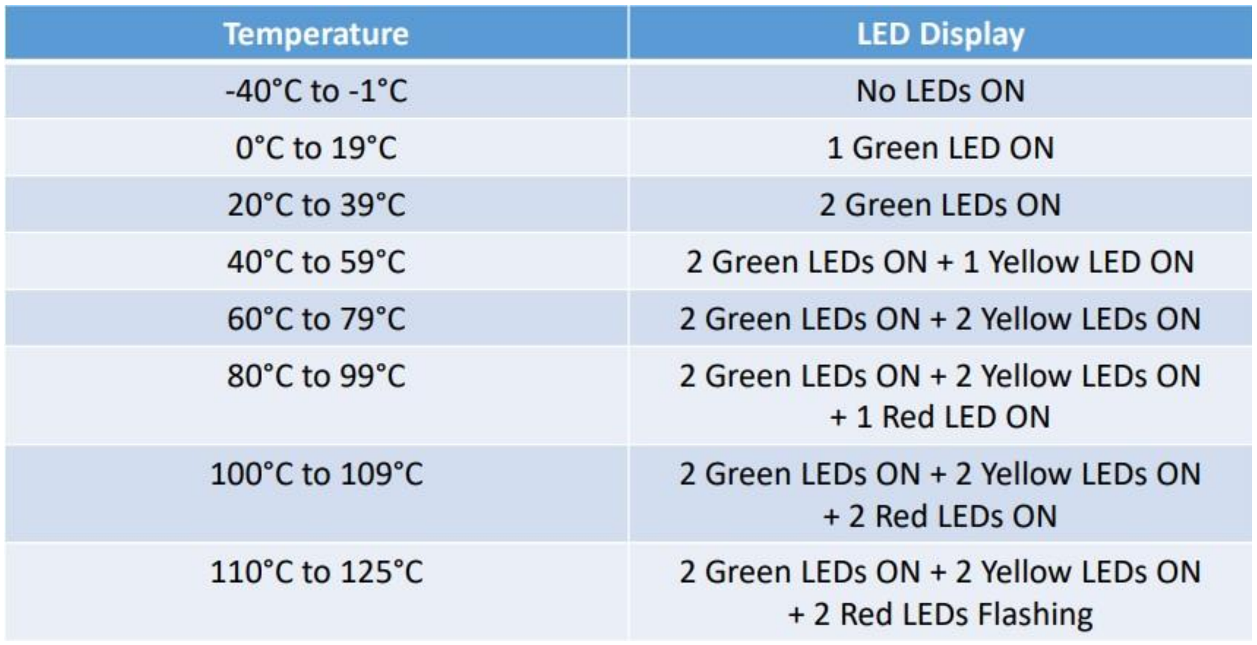


Figure 3 Graph of ADC value against temperature

From Figure 3, we are observing the relationship between ADC values and temperature. It also shows us a linear relationship. Since we are using ADC values to program our system, the linear relationship allows us to calibrate our system to correspond to temperature changes. Now, we will be providing details on which temperature range LEDs are programmed to turn on and off or blink. If there was no relationship between the physical quantity (voltage) and temperature, the temperature sensor would be obsolete. Now, let us take a closer look at what LEDs the system turns on for what range in temperature.

A table displaying the blue-print for our Arduino program



Temperature	LED Display
-40°C to -1°C	No LEDs ON
0°C to 19°C	1 Green LED ON
20°C to 39°C	2 Green LEDs ON
40°C to 59°C	2 Green LEDs ON + 1 Yellow LED ON
60°C to 79°C	2 Green LEDs ON + 2 Yellow LEDs ON
80°C to 99°C	2 Green LEDs ON + 2 Yellow LEDs ON + 1 Red LED ON
100°C to 109°C	2 Green LEDs ON + 2 Yellow LEDs ON + 2 Red LEDs ON
110°C to 125°C	2 Green LEDs ON + 2 Yellow LEDs ON + 2 Red LEDs Flashing

Table 2 A table demonstrating the temperature range for which LEDs light up

From Table 2, we observe the temperature ranges and their desired LEDs to be switched on or blinked. For instance, a temperature of 83 °C falls in 80 to 99 °C range. The corresponding ADC value of 83 °C is generated and is used as criteria for turning on specific LEDs. According to table 2, the program should light up 2 green LEDs, 2 yellow LEDs, and 1 red LED. After observing how our ADC values are related to the temperature values of TMP36, the blueprint and data from table 1 helps us develop our Arduino program.

The Arduino program first declares the digital pins connected to LEDs. Starting from left-most LED in Figure 1, the first and second green LED are assigned as constant integers with names grePin1 and grePin2 respectively. The first and second yellow LEDs are given yelPin1 and yelPin2, and the first and second red LEDs have been assigned as redPin1 and redPin2 respectively. The piezo buzzer is also given a constant integer named buzzer that holds 13 in it. The value assigned to each are the pins they are connected to. Aside from that, another variable is to be declared that will hold the ADC value called tValue. Next, we move on to initialize our serial communication at 9600 bps using Serial.begin(9600). All the pins are then declared as OUTPUT using pinMode command, including the buzzer pin. Next, we move on to the loop. The ADC value is read from analog pin A3 using analogRead. The value is printed using serial.print to allow us to see and record those values in table 1. As the ADC values (tValue) is known now, we use Table 2 to identify boundary temperature values and use their corresponding ADC values inside our if conditions. For instance, the temperature range 0 °C to 19 °C inclusive corresponds to ADC values in the range 104 to 144. If the tValue is within the range, the program uses digitalWrite statements to set the grePin1 to HIGH whilst the other pins to LOW. In the temperature range, 110 ° C to 125 °C, there is the exceptional task of making the red LEDs blink. For that, a while loop is created inside the if condition. The loop allows analog input to be read over and over again until it is changed. While the ADC values lie with 327 to 358, red LEDs are set to HIGH then LOW with a delay of 500 ms. Aside from all these functions, the buzzer buzzes a high frequency of 250 Hz for ADC values above 308 and values less than 359, exclusive.

The tone() command is used. Inside the tone() command we place the pin number of buzzer and the desired frequency of tone. For ADC values less than 104, the buzzer uses the tone() command, but plays a frequency of 50 Hz. For all other temperatures values, the buzzer is kept off using notone() command. The notone() command only nees the pin number of buzzer to stop the sound. At the end of the program, we incorporate a small delay of 2 ms to settle the ADC after the last reading. This is the main theme of the hyprogram. Upon sliding the slider on TMP36, we can produce different patterns in our LED lighting.

Results

The hyperlink to my TinkerCad Temperature Sensor System is:

https://www.tinkercad.com/things/1vzGzOAtd9Z-sensor-lab-report/editel?sharecode=R5WTE1AeBFt4fgIRBV2XHko_7giyGPPAa-U7PkaSUF8

The temperature sensor has a slider which can be moved to give different temperature readings. With different temperature readings, different color LEDs light up and buzzer will sound when conditions are met.

Discussion and Conclusions

The objectives of the lab exercise were met. The LEDs display the exact same result as displayed in Table 2 for each specified temperature range. The design was completed successfully and hardware set up was done following the design. Regarding conceptual insight, I learnt about the functionality of TMP36 in particular, its uses and structure, and practiced how to convert analog input to digital values using Arduino programs. There were no particularly significant difficulties that were faced, and fortunately no problems were faced with the finalized sensor system. If I were to repeat the sensor system again, I will try to try to make a tone for the buzzer that has more notes and timber (quality). Moreover, I would like to add gas sensor and pressure sensor to make a more full-fledged system.

References/Appendices

I worked on this lab report on my own. I used some sources to provide data on TMP36 and piezo-electric buzzer. The sources are listed below in the MLA format:

"Buzzer." *Wikipedia*, Wikimedia Foundation, 9 Nov. 2020, en.wikipedia.org/wiki/Buzzer.

DiCola, Tony. "Using Piezo Buzzers with CircuitPython & Arduino." *Adafruit Learning System*, learn.adafruit.com/using-piezo-buzzers-with-circuitpython-arduino?view=all.

"TMP36 Sensors. Datasheet Pdf. Equivalent." *DatasheetsPDF.com* datasheetspdf.com/pdf/727690/AnalogDevices/TMP36/1. Accessed 17 November 2020

"Temperature Measurement." *Wikipedia*, Wikimedia Foundation, 10 Oct. 2020, en.wikipedia.org/wiki/Temperature_measurement. Accessed 17 November 2020.

Ada, Lady. "TMP36 Temperature Sensor." *Adafruit Learning System*, learn.adafruit.com/tmp36-temperature-sensor?view=all. Accessed 17 November 2020.

The Arduino program that controls the temperature sensor system is provided below:

```
/*****************************************************
 Author:    Mahir Rahman
 Course:    CE1100.001
 Date:      11/17/2020
 Assignment: Sensor Lab Report
 Compiler:   Arduino 1.8.13

 Description:
 This program is used to detect temperature readings on
 the temperature sensor (TMP36) and indicate different
 ranges of temperature by lighting up different
 combination of LEDs. It also has a buzzer that sounds
 to alert user of very high or low temperatures.
*****************************************************/

// These constants won't change. They're used to give names to the
 pins used:

// Output pin that the LEDs are attached to
const int redPin1 = 5;
const int redPin2 = 3;
const int yelPin1 = 9;
const int yelPin2 = 6;
const int grePin1 = 11;
const int grePin2 = 10;
const int buzzer = 13;

int tValue = 0;       // value read from the sensor

void setup() {
 // initialize serial communications at 9600 bps:
 Serial.begin(9600);
 //Set up LEDs as output using pinMode
 pinMode(redPin1, OUTPUT);
 pinMode(redPin2, OUTPUT);
 pinMode(yelPin1, OUTPUT);
 pinMode(yelPin2, OUTPUT);
 pinMode(grePin1, OUTPUT);
 pinMode(grePin2, OUTPUT);
 pinMode(buzzer, OUTPUT);
}


void loop() {
 // read the analog in value:
 tValue = analogRead(A3);
 Serial.print("tValue = ");
Serial.println(tValue);
delay(500);

//The follwoing if condition checks the sensor value
//and lights up corresponding LEDs
if (tValue >= 104 && tValue < 145)
{
  //digitalWrite statements to turn ON or OFF LEDs
  noTone(buzzer);
  digitalWrite(grePin1, HIGH);
  digitalWrite(grePin2, LOW);
  digitalWrite(yelPin1, LOW);
  digitalWrite(yelPin2, LOW);
  digitalWrite(redPin1, LOW);
  digitalWrite(redPin2, LOW);
}
else if (tValue >= 145 && tValue < 184)
{
  noTone(buzzer);
  digitalWrite(grePin1, HIGH);
  digitalWrite(grePin2, HIGH);
  digitalWrite(yelPin1, LOW);
  digitalWrite(yelPin2, LOW);
  digitalWrite(redPin1, LOW);
  digitalWrite(redPin2, LOW);
}
else if (tValue >= 145 && tValue < 227)
{
  noTone(buzzer);
  digitalWrite(grePin1, HIGH);
  digitalWrite(grePin2, HIGH);
  digitalWrite(yelPin1, HIGH);
  digitalWrite(yelPin2, LOW);
  digitalWrite(redPin1, LOW);
  digitalWrite(redPin2, LOW);
}
else if (tValue >= 227 && tValue < 266)
{
  noTone(buzzer);
  digitalWrite(grePin1, HIGH);
  digitalWrite(grePin2, HIGH);
  digitalWrite(yelPin1, HIGH);
  digitalWrite(yelPin2, HIGH);
  digitalWrite(redPin1, LOW);
  digitalWrite(redPin2, LOW);
}
else if (tValue >= 266 && tValue < 309)
{
  noTone(buzzer);
  digitalWrite(grePin1, HIGH);
  digitalWrite(grePin2, HIGH);
  digitalWrite(yelPin1, HIGH);
```

```
    digitalWrite(yelPin2, HIGH);
    digitalWrite(redPin1, HIGH);
    digitalWrite(redPin2, LOW);
  }
  else if (tValue >= 309 && tValue < 327)
  {
    tone(buzzer, 250);
    digitalWrite(grePin1, HIGH);
    digitalWrite(grePin2, HIGH);
    digitalWrite(yelPin1, HIGH);
    digitalWrite(yelPin2, HIGH);
    digitalWrite(redPin1, HIGH);
    digitalWrite(redPin2, HIGH);
  }
  else if (tValue >= 327 && tValue <= 358)
  {
    tone(buzzer, 250);
    digitalWrite(grePin1, HIGH);
    digitalWrite(grePin2, HIGH);
    digitalWrite(yelPin1, HIGH);
    digitalWrite(yelPin2, HIGH);
    //This while loop allows red LEDs to blink
    while (tValue >= 327 && tValue <= 358)
    {
      tValue = analogRead(A3);
      digitalWrite(redPin1, HIGH);
      digitalWrite(redPin2, HIGH);
      delay(500);
      digitalWrite(redPin1, LOW);
      digitalWrite(redPin2, LOW);
      delay(500);
      Serial.print("tValue = ");
      Serial.println(tValue);
      delay(500);
    }
    delay(1);
  }

  else
  {
    tone(buzzer, 50);
    digitalWrite(grePin1, LOW);
    digitalWrite(grePin2, LOW);
    digitalWrite(yelPin1, LOW);
    digitalWrite(yelPin2, LOW);
    digitalWrite(redPin1, LOW);
    digitalWrite(redPin2, LOW);
  }
  // wait 2 milliseconds before the next loop for the analog-to-digital
  // converter to settle after the last reading:
  delay(2);
}
```

END