Abstract

Modern HVAC engineering technology has made numerous improvements since the invention of the first thermostat. Current HVAC systems utilizing thermostats are often overlooked, as they operate mostly independently. Despite this, the project detailed within this report shows that this technology is often changing and improving. A system was modeled to emulate this type of technology on an elementary level and is included in this report. The software and hardware needed to simulate a basic thermostat temperature-sensing system is laid out for replication purposes. This report also contains details of how the hardware and software interact, the theory and research behind the project, possible improvements of future iterations, fully commented and comprehensible code, and finally the conclusions made as a result.

Table of Contents

Background Information

The information in the following sections may be used to inform readers of what is meant by heating, ventilation and air conditioning (HVAC) engineering. The following content also explains the problems that were solved through this project, as well as the significance of the project itself.

1.0 Introduction

Study.com defines HVAC engineering as "[a branch of engineering which is concerned with the design, installation, maintenance, and repair of heating, ventilation, air conditioning, cooling, and refrigeration systems [1]. In the case of this project, HVAC engineering is more concerned with the interaction between electronic devices such thermostats and their external surroundings, in order to create an effective air conditioning (AC) / heating system. This definition is more fit to the project described within this report.

The project, based on a thermostat, contains basic thermostat heating & cooling system. A set temperature is established in degrees Celsius and is to be maintained by the system. If the external temperature of the system, read using a LM35Dz temperature sensor, differs from the set temperature by 5 degrees fahrenheit, the proper AC or heating system is turned on and the intensity is adjusted using fuzzy logic control. The system also utilizes a blue and red LED to show whether the AC or heat is on. The analog-to-digital converter (ADC) system is to be sampled once every second using a periodic interrupt. Through the use of two pushbutton switches, or through the use of an app, the set temperature is able to be controlled remotely, raising or lowering it by 1°C. Every time the temperature was sampled, the set temperature along with the external sampled temperature is communicated and displayed through the use of a raspberry pi interface. The temperatures can also be viewed on a UART dummy terminal for easy testing and debugging. The code was programmed for use with a Tiva TM4C123GH6PM launchpad, which was used throughout the project. This board gave the necessary general-purpose input/output (GPIO) pins to operate our system. The inputs necessary are used to read data from the temperature sensor, as well as to read the status of the two switches. The outputs on the board were utilized after processing external information and send appropriate signals to the two LEDs, raspberry pi, cooling system, heating system, and exhaust.

1.1 Problem Statement

In order to properly simulate a working thermostat system, it is important to analyze the system as a whole and define the problems needed to be solved. This is true in the case of HVAC engineering, where small mistakes may result in malfunctioning systems used in hospitals or marine environments, where safety and health are important factors [2].

One likely problem which needed to be avoided was improper interaction between the LEDs and the external temperature readings. The logic to turn on the AC and heating units could easily be swapped if not careful. In a real life scenario, this is the equivalent of the AC turning on when the external temperature is below the desired temperature, or the same as the heating system turning on when the external temperature is above the desired temperature. This would cause the AC or heat to stay on until a possible error handler could shut the system off. If no error handling is in place, this rising or falling temperature could go on until the system overheats. This could be very dangerous.

Another possible issue which may arise is the failure of the set temperature to change despite user input. In an in-home scenario, this would mean turning the temperature dial and not have the heat or air conditioning turn on. This is similar to the issue raised previously, as the system would malfunction; however in this case the system would continue to maintain the original set temperature even if the user wanted to increase or decrease it away from this temperature.

To get around this issue, software goes through many testing phases, such as the simulation performed in this project, in order to confirm the system is acting safely. In modern heating systems, safety concerns are very real. It is stated that, "...the most dangerous source of a burning smell is an electrical problem in the system. Issues with wiring can cause a burning smell and could eventually lead to a fire in the home." [3]

Before going in more detail about project itself, it is both important and interesting to understand the history of HVAC technology engineering.

Theory

The research within the following sections in this document are not meant to cover every major technological breakthrough in HVAC engineering technology, because the scope of HVAC engineering far exceeds that of this paper. The following research is instead focused on the basics of heating and air conditioning, and will be focused more on how the systems were developed since the beginning, as well as how they are utilized in modern day. The sections ahead are used as an attempt to inform readers on only the crucial HVAC engineering related breakthroughs, which will all connections to the thermostat project. This will provide necessary information on how and why these systems are so important and why there is an entire career field focused solely on these systems.

2.0 History of Thermostats

Before the modern thermostats of today, the technology was quite limited. Up until the late 19th century, heating systems were manually operated. Prior to the thermostats we have now, heating systems required frequent physical adjustments of dampers, drafts, valves to maintain heat. Not only was the manual control of these systems a nuisance, but it was especially difficult for companies who needed to maintain specific temperatures, such as textile mills [4]. Because of these troubles, numerous inventors have tried to improve the means of regulating temperature.

In 1883, when a college professor by the name of Warren S. Johnson, "was annoyed that his classroom was never at the temperature he wanted" sought out to invent the first electrical thermostat [4]. The invention of this model is seen as a great milestone in the world of thermostat development although it still required direct interaction between human and thermostat. The most important attribute this invention brought was the bypassing of the user-furnace interaction process. This model simply included a bell which would notify a janitor or maintenance person to operate the furnace.

To further improve on thermostat technology, and to accomplish what it's predecessors couldn't- the total operation of furnaces without human interaction, Albert Butz invented the "damper-flapper" in 1885. Equip with an automatic pulley system, this thermostat was able to regulate furnaces by opening and closing the furnace door (see Figure 1, next page). This inventor's fame continued to rise, as his company, the Butz Thermoelectric Regulator Company, eventually merged in 1927 to create the most iconic thermostat manufacturer in history - Honeywell International.

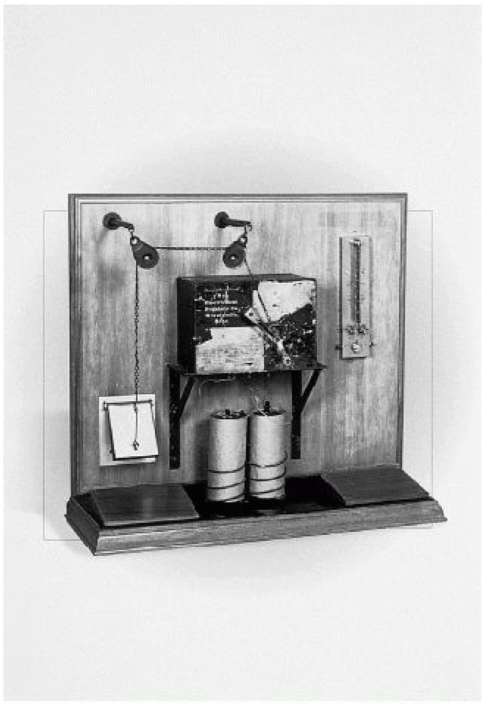


Figure 1: Damper-Flapper
Reprinted from
https://www.power-eng.com/articles/print/volume-100/issue-4/features/todays-control-systems-evolved-from-early-pioneers-dreams.html.

2.1 Modern Day Thermostats

The thermostats of today come in a variety of styles. The circular, non-programmable, "iconic" thermostat (Figure 2, below) made famous by Honeywell International, is the most common type. This is also known as the "round" or "classic" model.

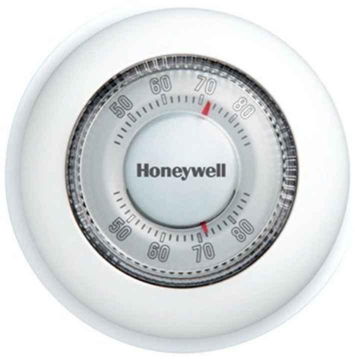


Figure 2: Honeywell Classic Thermostat
Reprinted from https://www.honeywellstore.com/store/products/honeywell-round-manual-thermostat-ct87n1001.htm

There aren't many more features to be added to thermostats, as the demand for more innovative styles hasn't been high. However, in this digitally connected world, thermostats, like many other everyday appliances, are becoming more and more connected. "Smart" thermostats have only become popular within the last decade, however it seems that they're around to stay. The features on these devices include remote control of the thermostat using Wi-Fi, communication and monitorization of different temperature "zones" in one's home, and even detectors that are conservative, able to detect if anyone's home and adjust the heat accordingly. These devices also bring a more sleek design, removing the mechanical appearance of the traditional thermostat. An example of one of these smart thermostats is Ecobee's Ecobee3 Lite (see Figure 3, below).

Figure 3: Ecobee's Ecobee3 Lite Smart Thermostat
Reprinted from https://www.cnet.com/news/smart-thermostat-roundup/.

2.2 Future of Thermostats and HVAC Engineering

Smart thermostats are doing well, and the idea of them is universally praised, although there are a few issues preventing these capable devices from becoming the home standard. Two of the main issues faced by modern smart thermostats are cost and connectedness to devices [6].

Many consumers see the price of these smart thermostats up front, ranging from $100 to upwards of $300, compared to the $30-$50 price tag on classic thermostats. While conservative features are available on smart thermostats, which make the overall long-term cost of these less than the classic thermostats, customers still tend to purchase the cheaper upfront price of the traditional thermostat. As technology improves, the upfront cost of smart thermostats are decreasing in cost.

The second factor preventing success of smart thermostats is the lack of connectedness with other devices. Some thermostats are only available to communicate with Apple devices exclusively. Other thermostats are only available to communicate with Android devices exclusively. As these smart devices develop further, is it expected that they will be compatible with multiple types of popular operating systems.

Design of AC / Heating System

When implementing this AC / heating system, both hardware and software design were thought through before implementing the two together. The following sections are used to outline both hardware and software design separately. The thought behind these designs will be supported by diagrams with detailed descriptions.

3.1 Hardware Design

Before showing the diagrams of how the hardware was connected, it's important to list out the components of the circuit in case readers want to replicate the simulation. The more complex hardware components are labeled with reference numbers for further familiarization.

1x - Texas Instruments Tiva™ TM4C123GH6PM Microcontroller [6]

1x - Breadboard to connect components

1x - LM35Dz Temperature Sensor [7]

2x - LED of separate colors (preferably 1 blue & 1 red)

2x - Push-button switches

30x (approx.) - Wires, approx. 20 cm in length

1x - Raspberry Pi [8]

1x - Enclosure 10x10x5 inches

3x - 5-12V DC Fans [9]

2x - .1 μF Capacitors

3x - 2n2222 Transistors [10]

1x LM324 Op-Amp [11]

1x - LM741 Op-amp [12]

Various Resistors

The first part of the hardware assembly was to assembly the physical circuit on the breadboard. The first components added were push buttons, separated on opposite ends of the breadboard to isolate the AC and heating systems. The push buttons fit snugly across the center of the board, so that pins A and B (seen in Figure 4, below) share a column. Pins C and D also share a column on the other side of the separation on the breadboard.

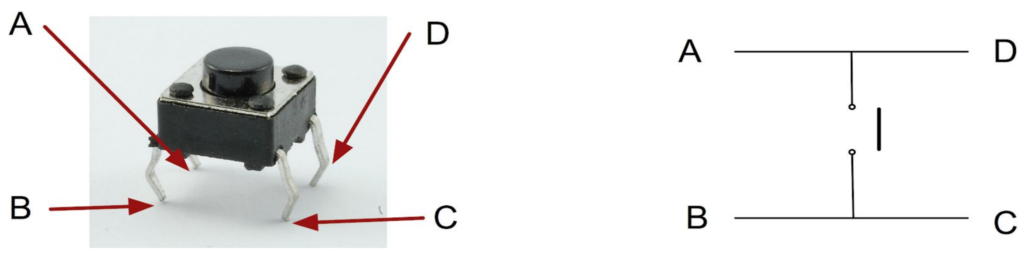


Figure 4: Push button switch pinout
Reprinted from https://learn.adafruit.com/adafruit-arduino-lesson-6-digital-inputs/push-switches.

After the push buttons were laid into the breadboard, the LEDs were placed down. The two LEDs were placed all in a similar fashion, with the longer, positive legs in their own rows, and the shorter, negative legs being brought directly to the column dedicated for a common ground. The directions of the LEDs matter because they are simply diodes, which only allow current flow in one direction. An LED is depicted in Figure 5, below.

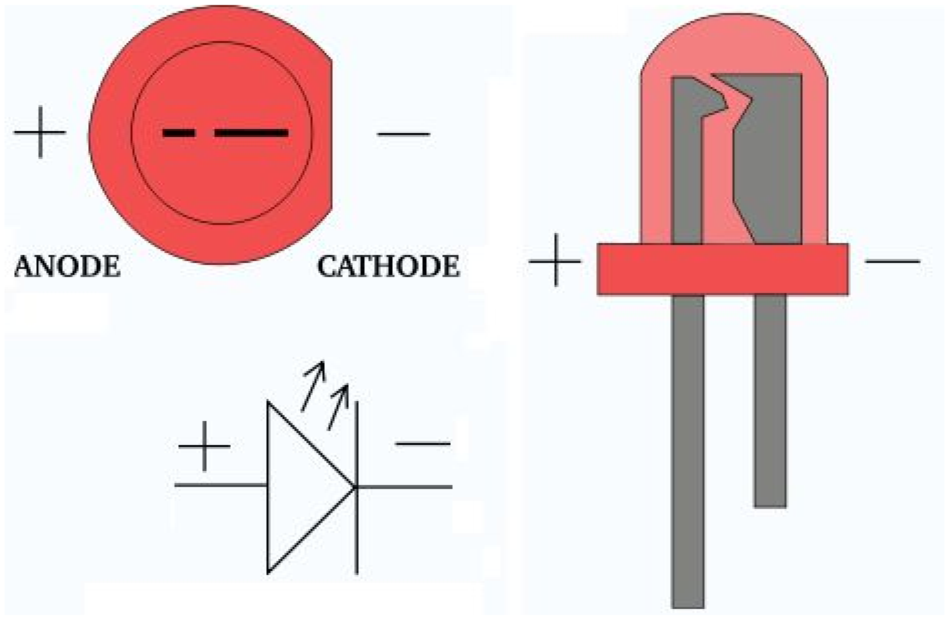


Figure 5: Diagram of LED
Reprinted from https://www.societyofrobots.com/electronics_led_tutorial.shtml.

A 5kΩ resistor was placed down connecting each button ("A" or "B" on the pinout) to the ground column on the breadboard. This helped limit current in that node, insuring that the current wouldn't be high enough to burn out the LED's (also connected to this node).

Wires were then added to connect the breadboard circuit to the Tiva launchpad. Each push button switch was fed to the Vcc column, where voltage was supplied by the Tiva's board 3.3V pin.

To establish an air conditioning section of the circuit, a wire was added from the switch to port PA5 on the launchpad. This wire was responsible for informing the launchpad whether or not the switch is being pressed. A wire was added to join the positive end of the blue LED to port PB5. This sent the appropriate signal from the controller to the launchpad after processing whether or not the LED should be on. This LED represented the AC being turned on. The breadboard circuit for this portion of the hardware is shown in Figure 6, below.

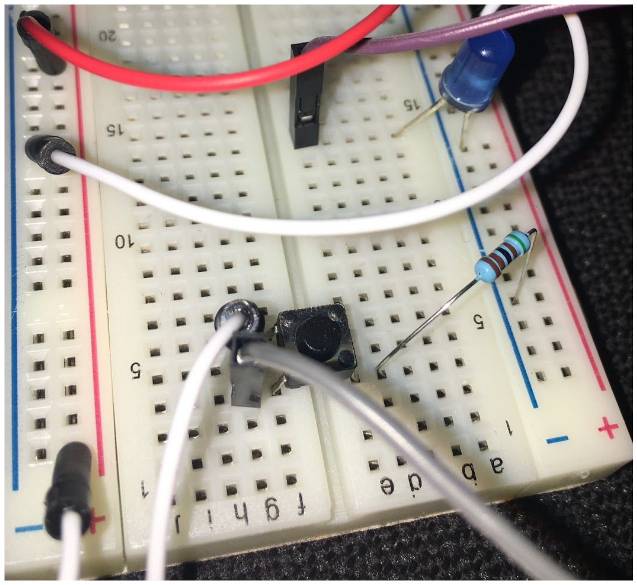


Figure 6: “Air Conditioning” Section Circuitry

For the “heating” section of the circuit, similar connections were made. However, the wire to connect the switch to the launchpad was connected to pin PA4 and the wire connecting the launchpad to the positive terminal of the red LED was from pin PB6.

The last component added to the circuit is the LM35Dz Temperature Sensor. The LM35 series are temperature sensing devices with three pins. One dedicated to power (Vcc), ground (GND) and an output voltage which is linearly proportional to the Centigrade temperature, where 1V output voltage corresponds to 100°C. This output voltage is determined with the internal circuitry (resistance) of the device. This output voltage is connected to port PE3 on the launchpad using a wire. The Vcc and GND pins were connected to the power and ground columns on the breadboard, respectively. The completed circuit up to this point can be seen in Figure 7, below.

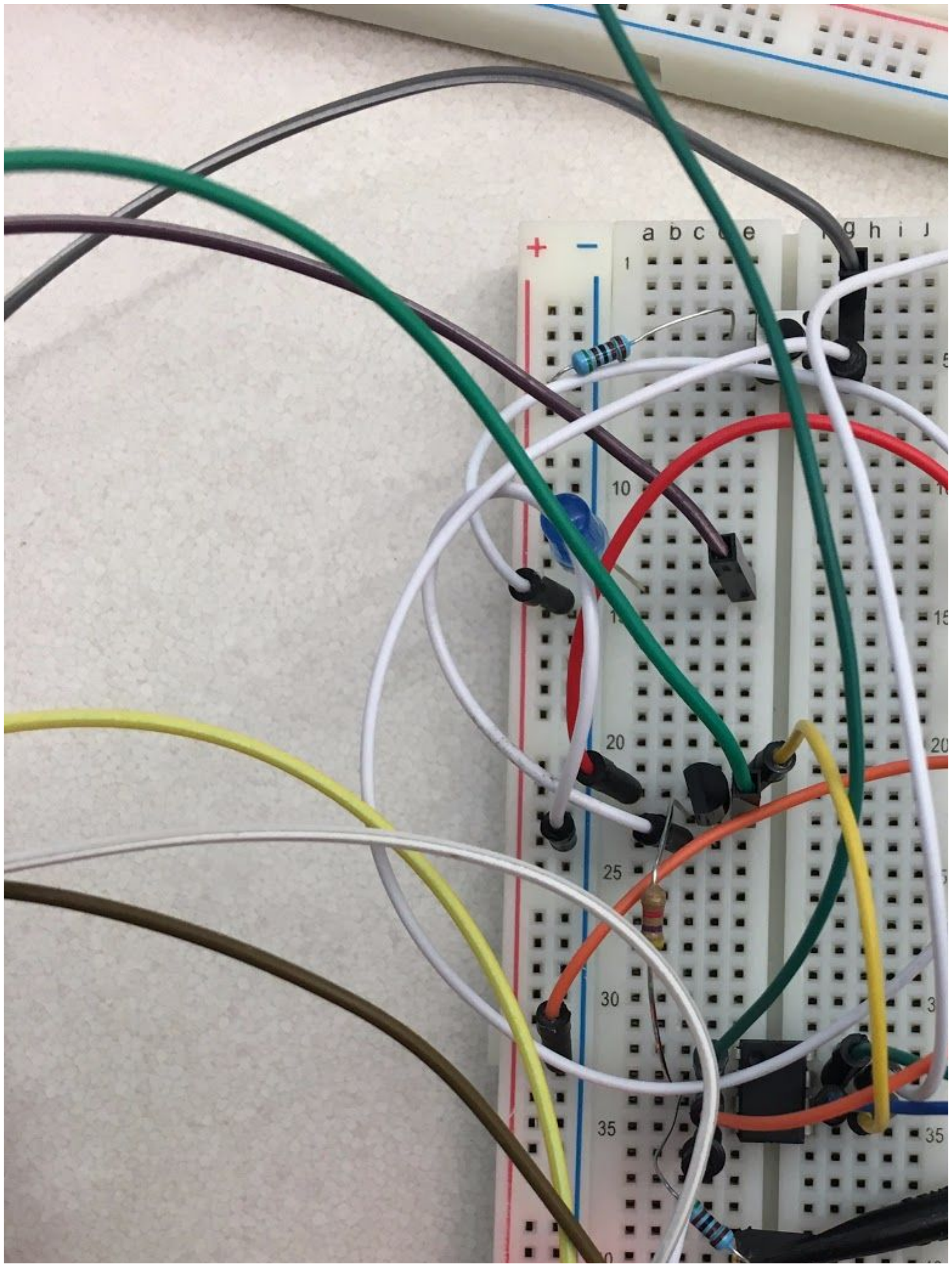


Figure 7: Full Hardware Design on Breadboard

A closer look of the launchpads connections can be seen in Figure 8, below.

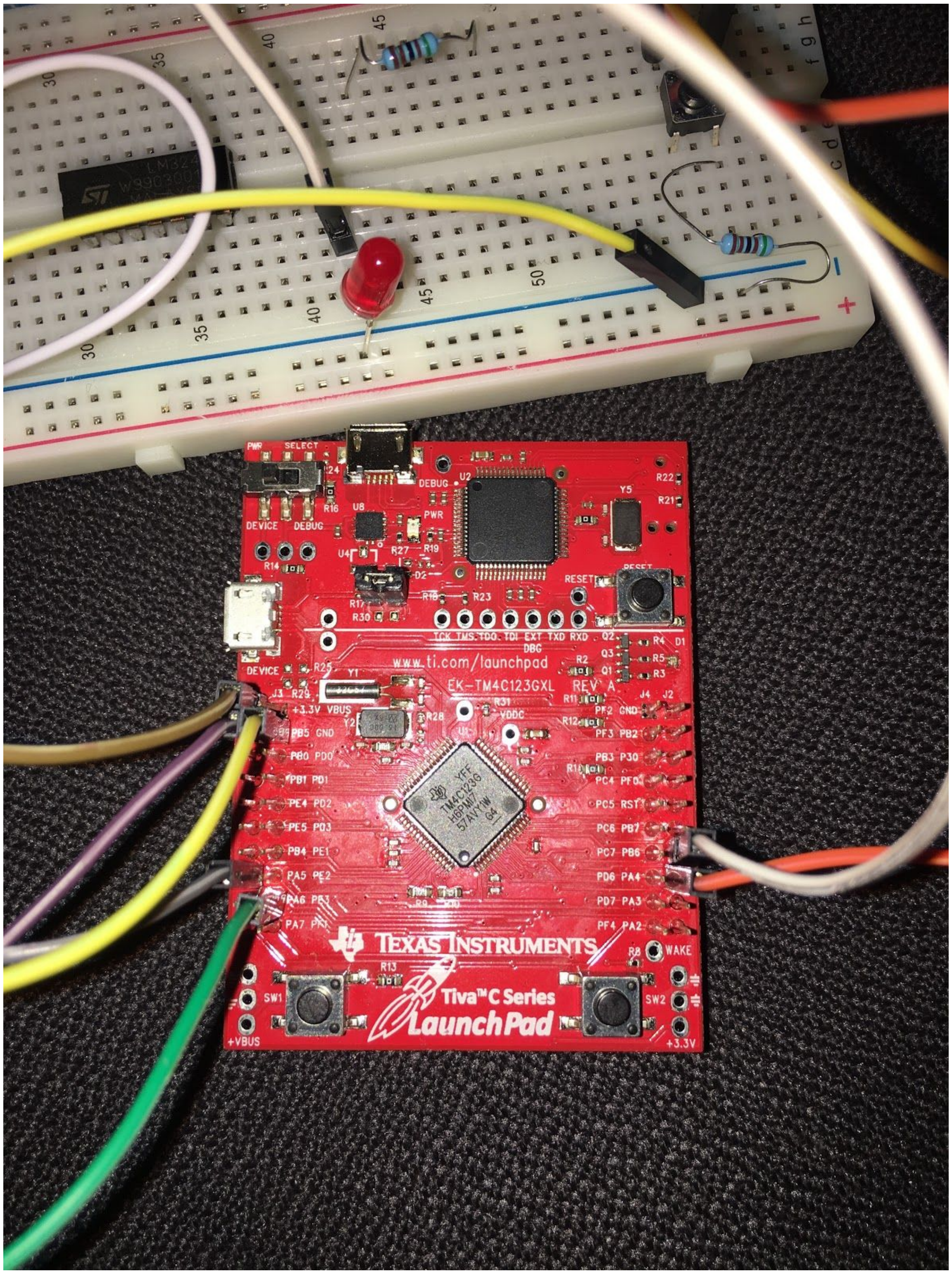


Figure 8: Close-up View of Microcontroller Board & Ports

The LM741 op-amp (seen in “ 3.1 Hardware Design”) was then added for functionality in the non-inverting mode in order to maintain highest resolution. The signal of the temperature sensor was amplified by factors of 3.3 to get full resolution. The op-amp’s inverting input was connected to 4.7kΩ resistor totaling, then to GND.

After these connections were made and verified, the capacitors and transistors were wired to the LM324. This wiring was then connected to the input of the DC fans. This was done for both the AC and heating sides. An example of this connection can be seen in Figure 9. Resistors were picked to give a gain of 3V this is because the Tiva outputs 3.3 volts and the max DC voltage for the fan is 12V.

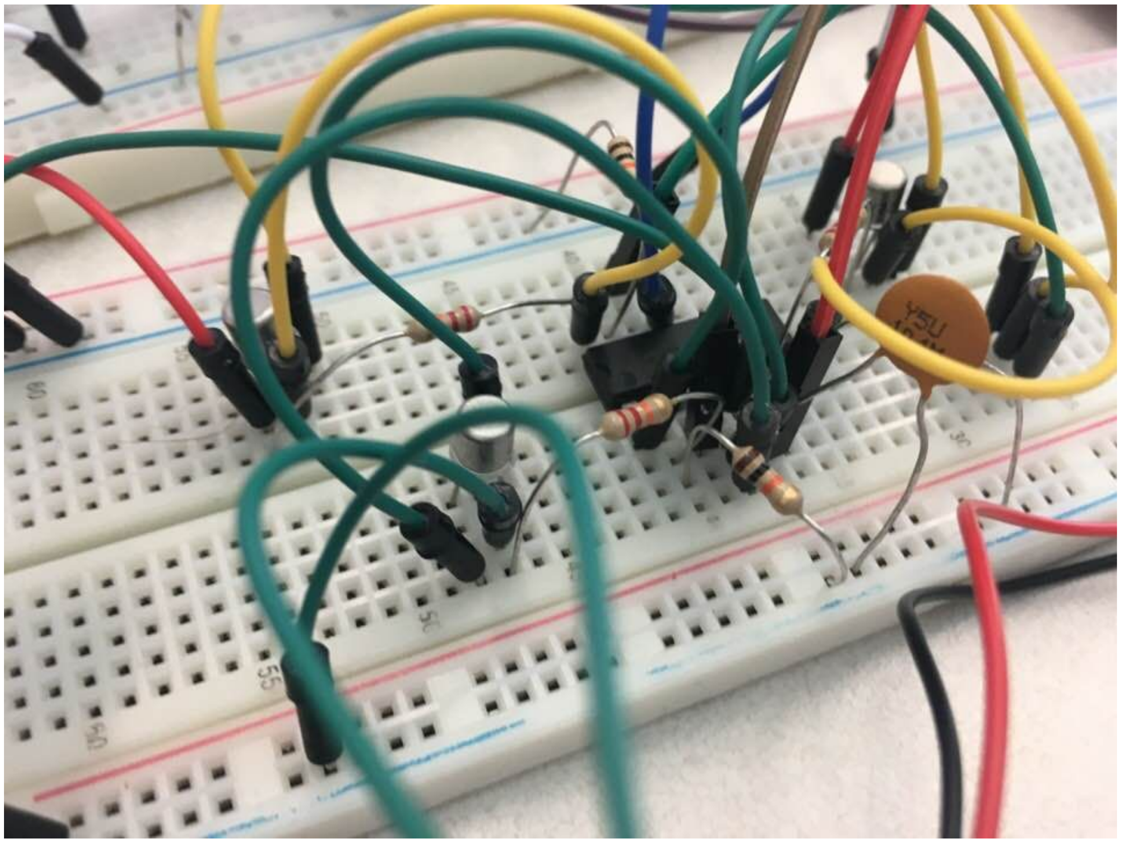


Figure 9: LM324 Connections for DC Fan Power

Lastly, the heating, ac and exhaust fans were all wired to the op-amps and were tested. These connections were tested with a multimeter. Fuzzy logic was incorporated to speed up or slow down the fans depending on how close the actual temperature was to the desired temperature. This fuzzy logic was established using equations of lines, which can be seen in the bibliography. The completed circuitry can be seen in Figure 10.

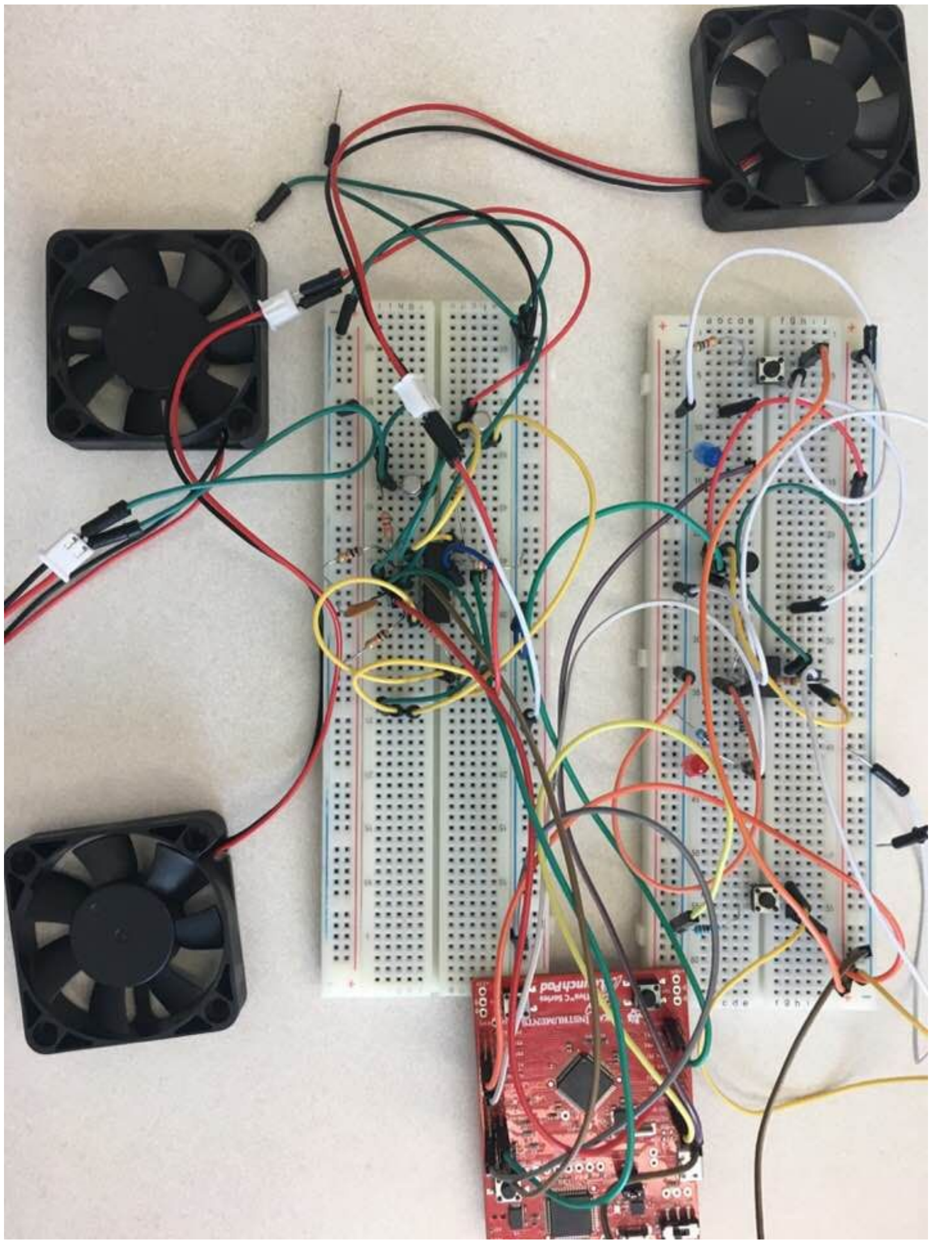


Figure 10: Completed Circuitry

After the inputs and output ports were connected to the physical hardware properly, the simulation was ready to be compiled and run. Alternate views of the hardware can be seen in Appendix A - Detailed View of Hardware.

3.2 Software Design

Similarly to the hardware, before describing of how to operate the software, it's important to list out the software required and the steps in order to the temperature sensor to work in case readers want to replicate the simulation.

-Keil uVision4 [13]

-TeraTerm [14] or another dummy terminal software, such as PuTTY

-Code written in C

-Code written in Python

The first step of getting the temperature to work is to write code that would read temperature from the temperature sensor and output it to the terminal software. Then use two switches as interrupts to increase or decrease the temperature setpoint. The hardest part of getting the two switches to work was making them work individually. After getting the switches to work, the next part was to get the LEDs to work to represent AC and heat. The blue led turns on when the temperature is higher than the setpoint to indicate that the AC is on and the red led turns on when the temperature is below the setpoint to indicate that the heat is on. Plus, create a buffer for a +/-2 degree from the setpoint to indicate that both LEDs are turn off when the temperature is getting close to the setpoint. After making the LEDs work, the hardest part is to use periodic interrupts to capture ADC values and update display. The final part of the code is to implement the LM741 op-amp to receive a larger resolution for the temperature reading.

Code samples from microdigitaled.com were implemented to avoid reinventing the wheel. These three segments are from the “Interfacing LM34 Temperature Sensor” program [15], the “UART0 Transmit” program [16] and the Testing Nested Interrupts program [17].

After the software was tested for close-up use, the software was developed for remote use utilizing the raspberry pi and wifi communication. A simple raspberry pi script was used to poll the temperatures and the output was wired to the switches. This way, if the raspberry pi was fed an input, in the form of a char value, the pi could send a 1 to the right switch, thus incrementing or decrementing the temperature as desired.

Obstacles

Despite the project eventually becoming a complete system, many obstacles were encountered before the final design was reached. A few of these obstacles are lack of enclosure and unfamiliarity with communication specifics.

One of the biggest obstacles encountered by the group when designing the project was finding a proper enclosure for the system. The enclosure needed to be as air tight as possible so that it was run at maximum efficiency. Initially, the enclosure was going to be created using simple plastic containers, as shown in Figure 11.

Figure 11: Initial Enclosure

The reason we didn't use this enclosure was because we wanted to save on our costs. Our running receipt totalled to about $120, not counting the raspberry pi (which was also bought for personal use). Instead of this enclosure, we decided to use Solidworks to design and 3d print an enclosure. This would allow for easy customizability if the system were to be redesigned. Unfortunately, a week before finalizing the design, the group realized that the design wouldn't work with the printer, due to various pipes and bends being featured in the design. As seen in "Hardware Design" a Lego enclosure was used in the final design.

Another issue faced was that some of the group had never used any type of communication with the Raspberry Pi. A lot of the information needed to be researched before moving forward.

Possible Improvements

Although this project simulates the workings of a temperature-sensing thermostat system on a basic level, improvements could be made in order to increase the accuracy of the system. New features could also be added in future iterations. Some possible improvements would be printing timestamps along with the temperature readings or to use a LCD to display the temperature instead of the terminal.

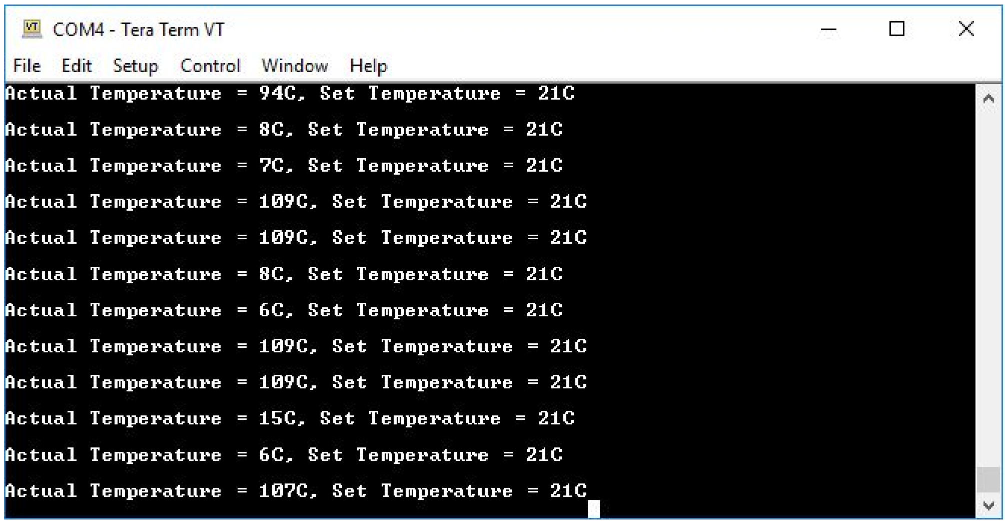
Utilizing serial peripheral interfacing, a synchronous serial protocol (SPI) and master-slave communication, or inter-integrated circuit bus (I^2C) processing, future iterations of this system could be used to print timestamps generated by internal clocks.

Results & Concluding Thoughts

This section analyses the results of the experiment. This experiment went as expected. With the use of two external switches, the room 'SetTemp' can be increased or decreased. The SetTemp is whatever temperature the user wants their home to maintain. When the temperature sensor gets a reading of ± 2℃ of the SetTemp, either the heat or the air conditioning will turn on. Our group wired a blue LED to indicate cooling was turned on and a red LED to indicate heating. The group also used DC fans to heat and cool down the system. The op-amp being used was LM741 to increase resolution in order to receive a more precise temperature output.

There were a two issues while working on the temperature experiment. The first issue was the switches. Both switches would either output the same result, or one of the switches would work and the other one would crash the program or was unresponsive.

The second issue was connecting the op-amp with the temperature sensor. Both op-amps were not giving an accurate output despite the algorithm formula multiplier being changed from 330 to 110. Figure 9 shows how the temperature lacks a trend, giving seemingly random values from zero to over 100. The op-amp was tested using the DC and the output was the right result but when connecting to the temperature that when the temperature did not output an accurate temperature as shown in Figure 10. The group fixed the issue using LM741 op-amp by checking the connection again and looking at the pinout of the op-amp and noticing that we missed a 4.7k resistor from input to output and some connection were connected wrong. Overall, our group created a functional home temperature controller as shown in Figure 11 and Figure 12.



```
Actual Temperature = 94C, Set Temperature = 21C
Actual Temperature = 8C, Set Temperature = 21C
Actual Temperature = 7C, Set Temperature = 21C
Actual Temperature = 109C, Set Temperature = 21C
Actual Temperature = 109C, Set Temperature = 21C
Actual Temperature = 8C, Set Temperature = 21C
Actual Temperature = 6C, Set Temperature = 21C
Actual Temperature = 109C, Set Temperature = 21C
Actual Temperature = 109C, Set Temperature = 21C
Actual Temperature = 15C, Set Temperature = 21C
Actual Temperature = 6C, Set Temperature = 21C
Actual Temperature = 107C, Set Temperature = 21C
```

Figure 9: Temperature readings using LM741 Op-Amp

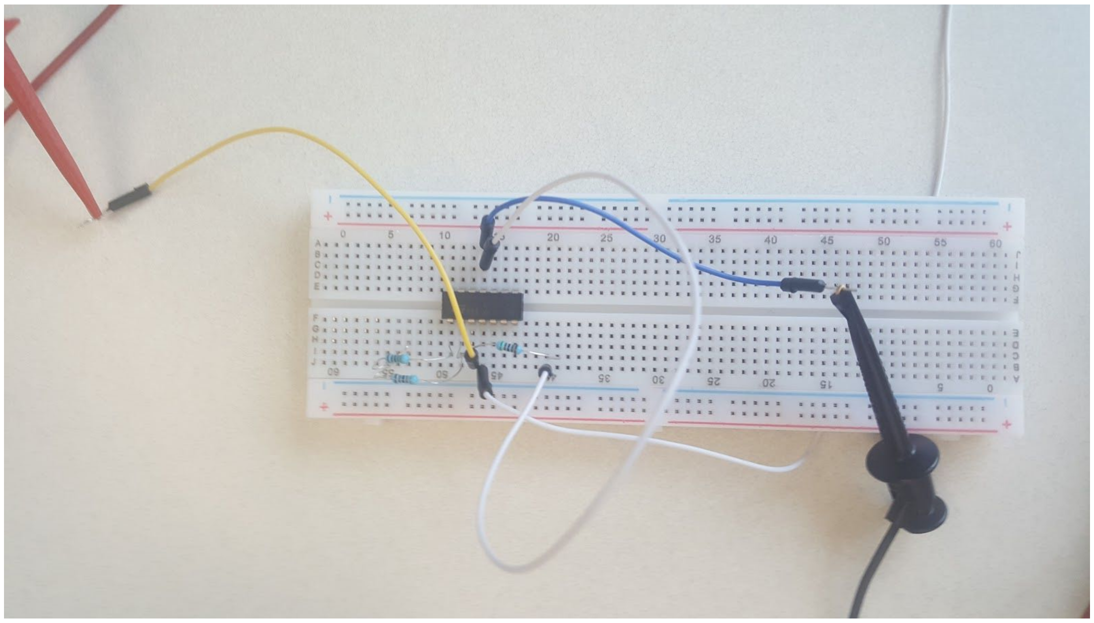
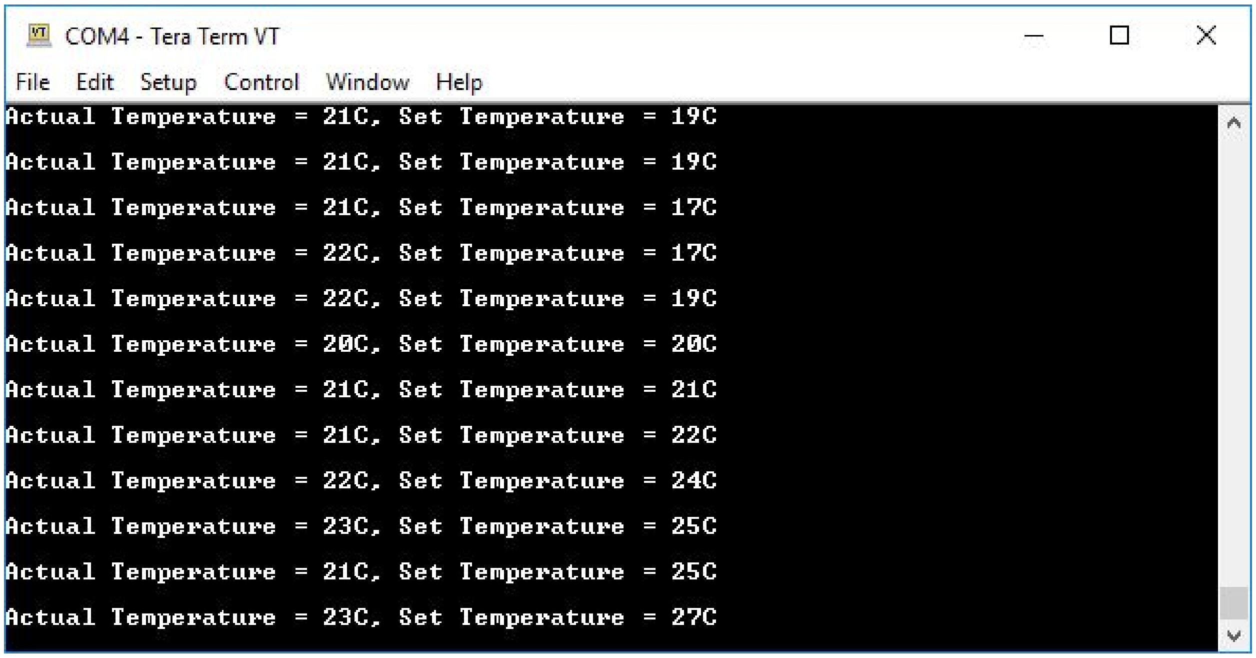


Figure 10: LM324 Op-Amp Tested using DC



```
Actual Temperature = 21C, Set Temperature = 19C
Actual Temperature = 21C, Set Temperature = 19C
Actual Temperature = 21C, Set Temperature = 17C
Actual Temperature = 22C, Set Temperature = 17C
Actual Temperature = 22C, Set Temperature = 19C
Actual Temperature = 20C, Set Temperature = 20C
Actual Temperature = 21C, Set Temperature = 21C
Actual Temperature = 21C, Set Temperature = 22C
Actual Temperature = 22C, Set Temperature = 24C
Actual Temperature = 23C, Set Temperature = 25C
Actual Temperature = 21C, Set Temperature = 25C
Actual Temperature = 23C, Set Temperature = 27C
```

Figure 11: Temperature Readings without Op-Amp

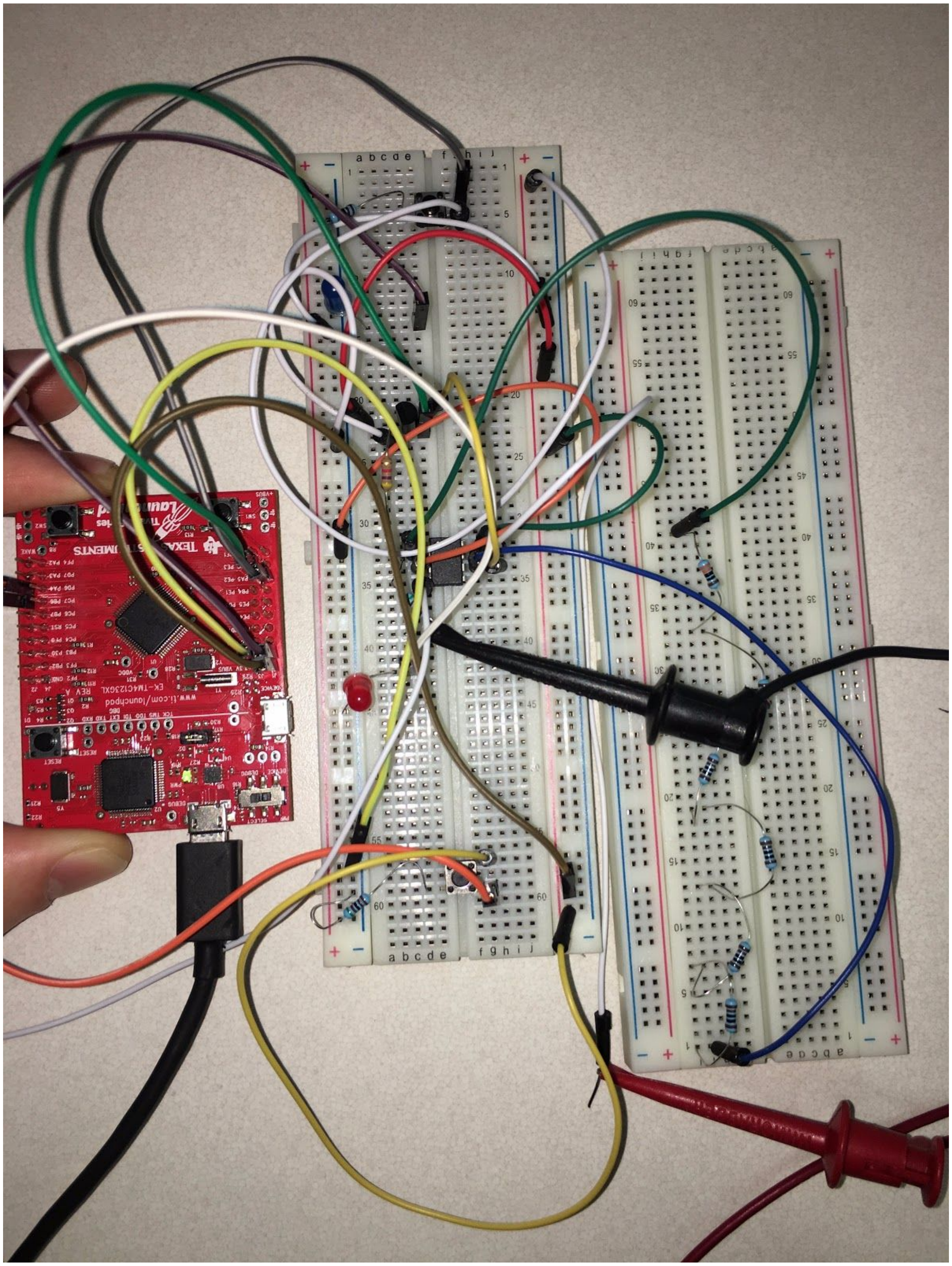


Figure 12: Temperature Sensor

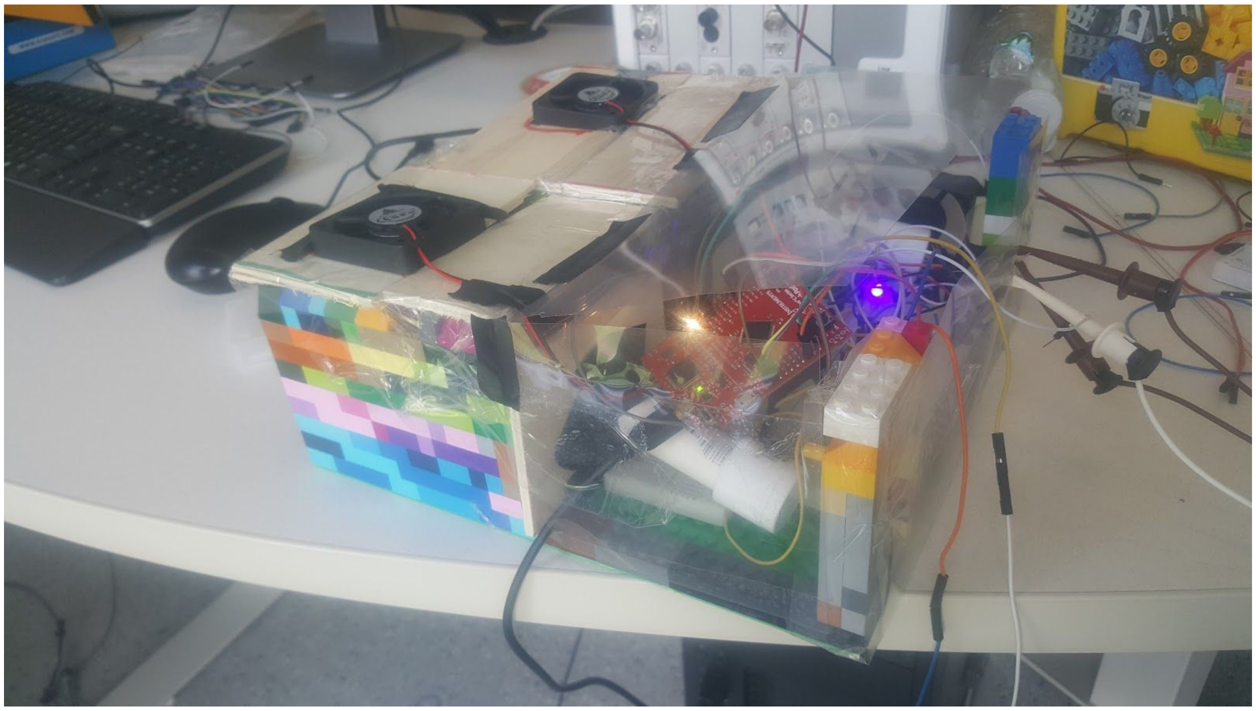


Figure 13: Full Enclosure

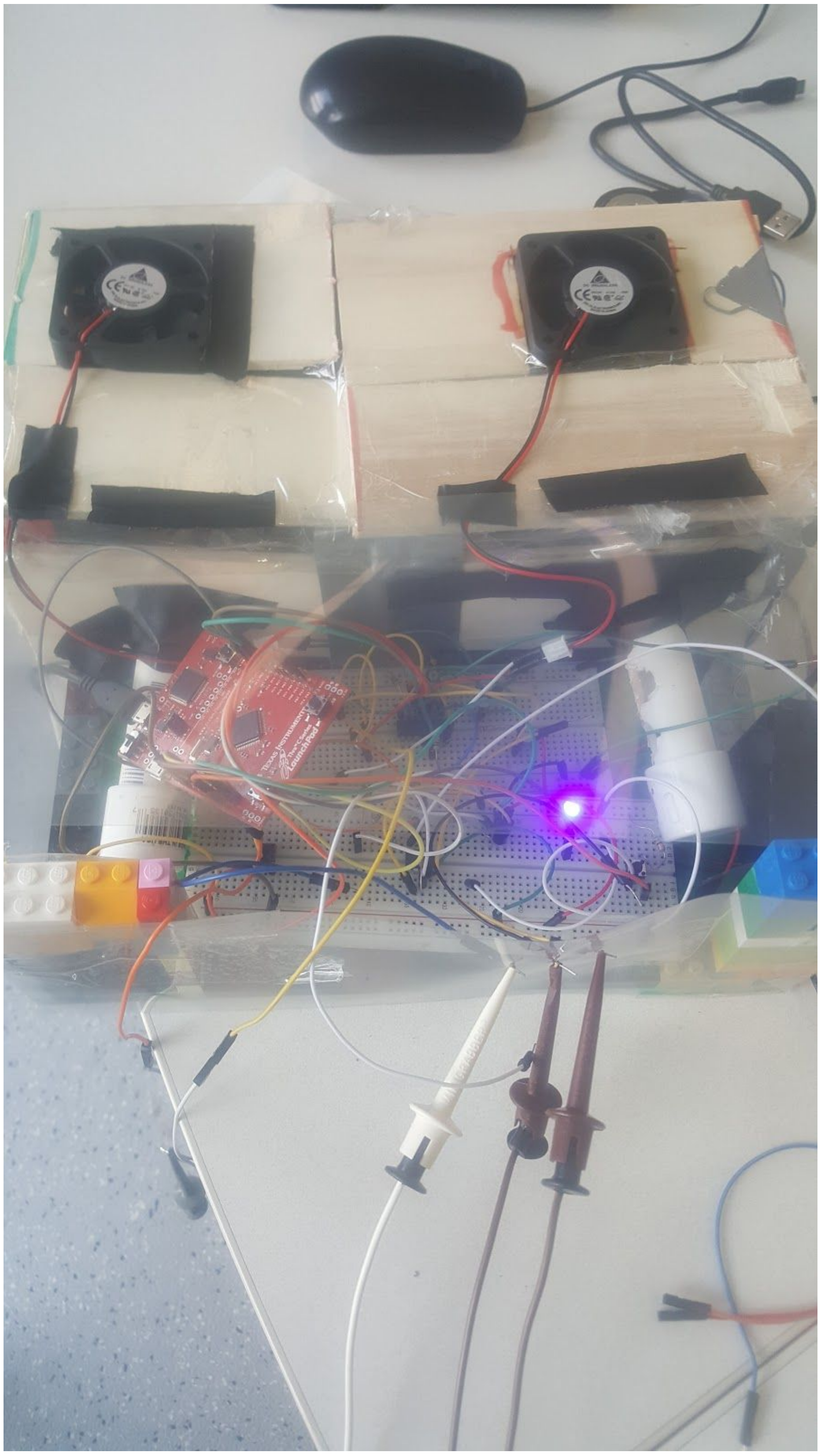


Figure 14: Full Enclosure (Top View)

Bibliography

[1] Study. “HVAC Engineer: Job Info & Career Requirements.” *Study.com*, 29 Mar. 2018, https://study.com/articles/HVAC_Engineer_Job_Info_and_Requirements_for_Students_Considering_a_Career_in_HVAC_Engineering.html

[2] Various Authors. “HVAC.” *wikipedia.org*, 29 Mar 2018, https://en.wikipedia.org/wiki/HVAC

[3] Various Authors. “Heating Repair FAQ: Troubleshooting Guide to Home Heating Systems.” *oasiscooling.com*, 29 Mar. 2018, http://www.oasiscooling.com/blog/heating-repair-troubleshooting-guide/

[4] Various Authors. “A Brief History of Thermostats: The Inventors.” *thermostat-recycle.org*, 29 Mar. 2018, https://www.thermostat-recycle.org/blog/a_brief_history_of_thermostats_the_inventors

[5] Protect America. “What’s the Future of Smart Thermostats?” *protectamerica.com*, 29 Mar. 2018, https://www.protectamerica.com/home-security-blog/tech-tips/whats-the-future-of-smart-thermostats_11329

[6] Texas Instruments. “Tiva™ TM4C123GH6PM Microcontroller Data Sheet.” *ti.com*, 29 Mar 2018, http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf

[7] Texas Instruments. “LM35Dz Sensor Datasheet.” *ti.com*, 29 Mar 2018, http://www.ti.com/lit/ds/symlink/lm35.pdf

[8] Canakit. “Raspberry Pi 3 Model b+.” canakit.com, 1 Apr 2018, https://www.canakit.com/raspberry-pi-3-model-b-plus.html?cid=usd&src=raspberrypi&src=raspberrypi

[9] Amazon. “DC Fan.” *amazon.com*, 1 Apr 2018, https://www.amazon.com/gp/product/B06XQDMMJ5/ref=oh_aui_detailpage_o01_s00?ie=UTF8&psc=1

[10] Wikipedia. “2N2222” *wikipedia.org*, 1 Apr 2018, https://en.wikipedia.org/wiki/2N2222

[11] Texas Instruments. “LM324 Datasheet.” *ti.com*, 15 Apr 2018, http://www.ti.com/lit/ds/snosc16d/snosc16d.pdf

[12] Learning about Electronics. “LM741 Pinout Connections” *learningaboutelectronics.com*, 29 Mar 2018, http://www.learningaboutelectronics.com/Articles/LM741-op-amp-pinout-connections

[13] KEIL. “MDK-ARM.” *keil.com*, 29 Mar 2018, https://www.keil.com/demo/eval/arm.htm

[14] OSDN. “Tera Term.” *osdn.net*, 29 Mar 2018, https://osdn.net/projects/ttssh2/releases/

[14] Microdigitaled. “Rewrite of the Interrupt Handler.” *microdigitaled.com*, 29 Mar. 2018, http://www.microdigitaled.com/ARM/TI_ARM/Code/Ver1/Chapter6/Program6_2.txt

[15] Microdigitaled. “Interfacing LM34 Temperature Sensor.” *microdigitaled.com*, 29 Mar. 2018 http://www.microdigitaled.com/ARM/TI_ARM/Code/Ver1/Chapter7/Program7_3.txt

[16] Microdigitaled. “UART0 Transmit.” *microdigitaled.com*, 29 Mar. 2018, http://www.microdigitaled.com/ARM/Freescale_ARM/Code/Ver1/Chapter4/Program4_1.txt

[17] Microdigitaled. “Testing Nested Interrupts.” *microdigitaled.com*, 29 Mar. 2018, http://www.microdigitaled.com/ARM/TI_ARM/Code/Ver1/Chapter6/Program6_8.txt

Various Authors. “Bang-Bang Control.” *wikipedia.org*, 29 Mar 2018, https://en.wikipedia.org/wiki/Bang–bang_control

Various Authors. “DC Motor Speed Control using PWM with PIC Microcontroller” *electrosome.com*, 1 May 2018, https://electrosome.com/dc-motor-speed-control-pic-pwm/

Various Authors. “Timer PWM mode - 100% and 0% duty cycle” *e2e.ti.com*, 1 May 2018 https://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/t/354826?Timer-PWM-mode-100-and-0-duty-cycle

Various Authors. “Pulse Width Modulation on the TM4C123GH6PM Microcontroller” *egr.msu.edu*, 1 May 2018 https://www.egr.msu.edu/classes/ece480/capstone/fall15/group09/appnotes/JoshuaLambApplicationNote.pdf

Lukas Mazzetti, George Dagis, Michael Martinez
https://docs.google.com/spreadsheets/d/1ACQ-3vjMIGmENG1bvyePi_U59TkthD6WohboPZbI2d0/edit#gid=0

Appendices

Appendix A - Detailed View of Hardware

Below are numerous pictures of the hardware setup for replication purposes.

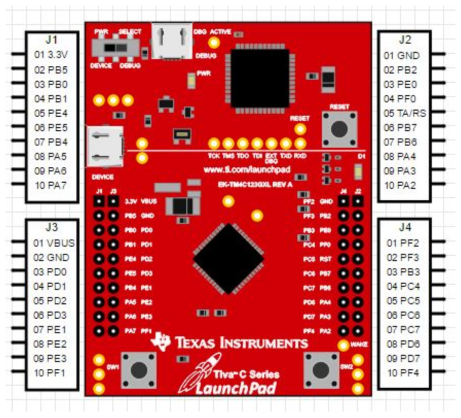


Figure A1: Texas Instruments Launchpad Microcontroller Pinout

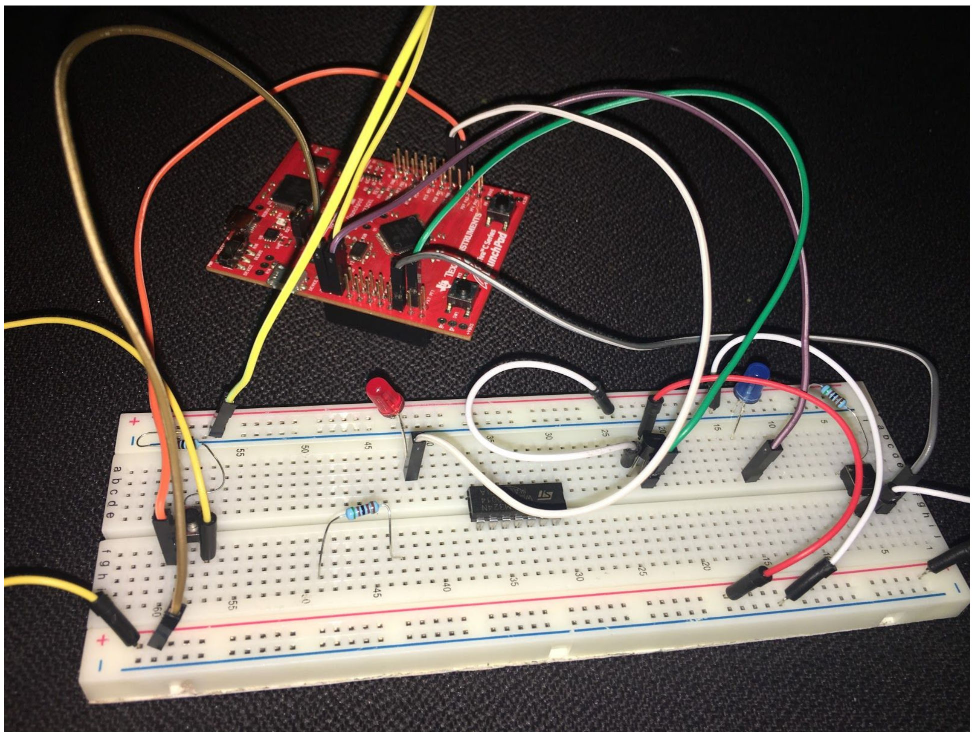


Figure A2: Full Circuit (wide perspective)

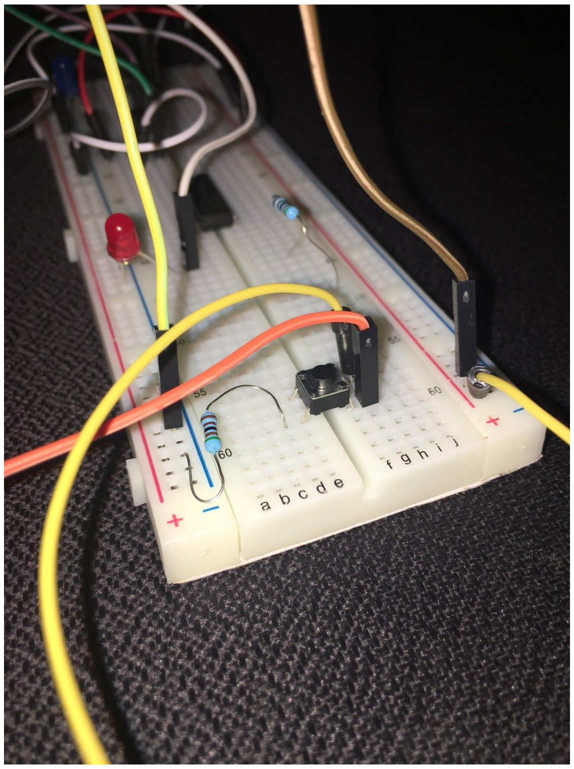


Figure A3: Demonstrated Circuit (“Heating” side)

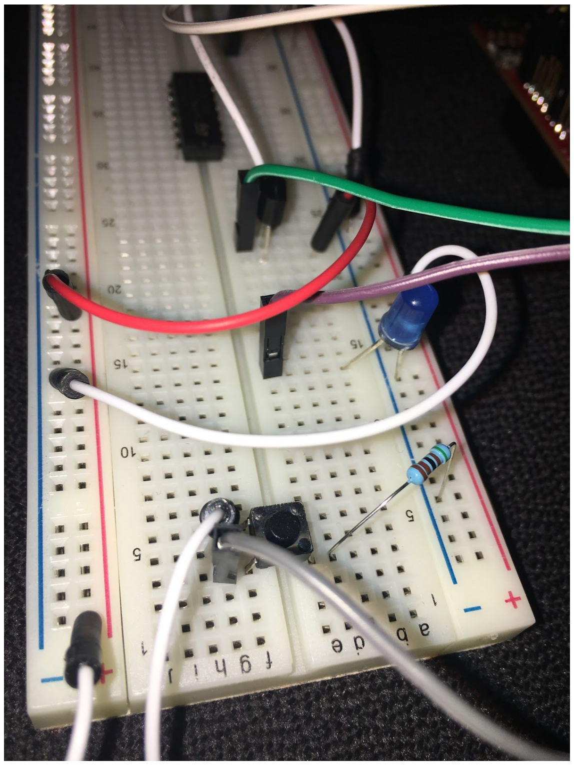


Figure A3: Demonstrated Circuit (“Air Conditioning” side)

Appendix B - Code

Similarly to Appendix A - Detailed View of Hardware, the entirety of the code used in the main.c file can be found below for replication purposes.

```
// Authors:                    George Dagis (CE),
// Professor: Michael Otis
// EGC 443 Embedded Systems
// 3-30-18
// v1.0

/* This program acts as a basic thermostat heating / cooling system. Meant to be used with a Tiva™ TM4C123GH6PM
Microcontroller*
Program converts analog output from an LM35Dz temperature sensor and convert it to temperature in Celsius
   A desired temperature is established and is to be maintained by the system
   If external temperature exceeds the desired temperature by 1-2 degrees, an AC system indicated by a blue LED is turned on
   If external temperature is less than the set temperature by 1-2 degrees, a heating system indicated by a red LED is turned on
   Using interrupts via a timer, the ADC is sampled once every second
   Both the external and desired temperature are refreshed and displayed using terminal software such as TeraTerm** or PuTTY

   *Exact Microcontroller used to work this program
   **TeraTerm 4.98 was used to work this program
*/

#include "TM4C123GH6PM.h"
#include <stdio.h>

//Program utilizes UART functionality
void UART0Tx(char c);
void UART0_init(void);
void UART0_puts(char* s);

//Program utilizes timer
void Timer1_init(void);

//Program utilizes delay
void delayMs(int n);

//Integer variable temperature. This is the temperature that is read from the LM35Dz temperature sensor
int currtemp;

//Desired temperature
int destemp = 70;

//Buffer
char buffer[16];

//Main method
int main(void)
{
  //To Run Timer
```

```
Timer1_init();

  // Initialize UART0 for output
  UART0_init();

  // Enable Clocks for GPIOE and ADC0
  SYSCTL->RCGCGPIO |= 0x10;          // Enable clock to GPIOE
  SYSCTL->RCGCADC |= 1;              // Enable clock to ADC0

  //Initialize PE3 for AIN0 input
  GPIOE->AFSEL |= 8;                 // Enable alternate function
  GPIOE->DEN &= ~8;                  // Disable digital function
  GPIOE->AMSEL |= 8;                 // Enable analog function

  // Initialize ADC0
  ADC0->ACTSS &= ~8;                 // Disable SS3 during configuration
  ADC0->EMUX &= ~0xF000;             // Software trigger conversion
  ADC0->SSMUX3 = 0;                  // Get input from channel 0
  ADC0->SSCTL3 |= 6;                 // Take one sample at a time, set flag at 1st sample
  ADC0->ACTSS |= 8;                  // Enable ADC0 sequencer 3

  //GPIOA and  Configurations
  SYSCTL->RCGCGPIO |= 0x03;          // Enable clock for Port A and Port B
                SYSCTL->RCGCGPIO |= 0x20;                        // Enable clock for Port F
  GPIOA->DIR &= ~0x30;               // PORTA5 switch
  GPIOA->PUR |= 0x30;                // Enable pull up for PORTF4, 0
  GPIOA->DEN |= 0x30;                // PORTA5 dig

  //GPIOB Configurations
  GPIOB->DIR |= 0x20;                // PORTB5 Blue LED
  GPIOB->DIR |= 0x40;                // PORTB6 Red LED

  GPIOB->DEN |= 0x20;                // PORTB5 dig
  GPIOB->DEN |= 0x40;                // PORTB6 dig


                //GPIOF Configurations
                GPIOF->AFSEL = 0x0E;             /* PF1, PF2, PF3 uses alternate function */
                GPIOF->PCTL &= ~0x0000FFF0;      /* make PF1, PF2, PF3 PWM output pin */
                GPIOF->PCTL |= 0x00005550;
                GPIOF->DIR |= 0x0E;              // Enable GPIOF Pins 3,2,1
  GPIOF->DEN |= 0x0E;                // Enable dig

  GPIOA->IS  &= ~0x30;               // Make bit 6 edge sensitive
  GPIOA->IBE &= ~0x30;               // Trigger is controlled by IEV
  GPIOA->IEV &= ~0x30;               // Falling edge trigger
  GPIOA->ICR |= 0x30;                // Clear any prior interrupt
  GPIOA->IM  |= 0x30;                // Unmask interrupt

                //Test
                GPIOF->DATA = 0x0E;
		SYSCTL->RCGCPWM |= 2;       /* enable clock to PWM1 */
    SYSCTL->RCGCGPIO |= 0x20;   /* enable clock to PORTF */
    SYSCTL->RCC &= ~0x00100000; /* no pre-divide for PWM clock */


    //PWM config.
		//Enables generator 2 and pin F1
		PWM1->_2_CTL = 0x0;         /* stop counter */
    PWM1->_2_GENB = 0x0000008C; /* M1PWM7 output set when reload, */
                             /* clear when match PWMCMPA */
    PWM1->_2_LOAD = 16000;      /* set load value for 1kHz (16MHz/16000) */
    PWM1->_2_CMPA = 10000;      /* set duty cycle to min */
    PWM1->_2_CTL = 1;

		//enables generator 3, Pins F2, F3
    PWM1->_3_CTL = 0x0;         /* stop counter */
    PWM1->_3_GENB = 0x0000008C; /* M1PWM7 output set when reload, */
                             /* clear when match PWMCMPA */
		PWM1->_3_GENA = 0x0000080C;

    PWM1->_3_LOAD =16000;       /* set load value for 1kHz (16MHz/16000) */
    PWM1->_3_CMPA = 8000;       /* set duty cycle to min */
    PWM1->_3_CTL = 1;           /* start timer */
		PWM1->_3_CMPB = 1000;       /* set duty cycle to min */
    PWM1->ENABLE = 0xE0;        /* start PWM Channels 5,6,7 */



  //Enable interrupt in NVIC and set priority to 3. Enable IRQ's
  NVIC->IP[0] = 3 << 5;             // set interrupt priority to 3
  NVIC->ISER[0] |= 0x1;             // enable IRQ0 (D0 of ISER[0])
  __enable_irq();                   // global enable IRQs

  while(1){}

  }

  //UART0
  void UART0_init(void)
  {
     //Provide & enable clocks
     SYSCTL->RCGCUART |= 1;        // Provide clock to UART0
     SYSCTL->RCGCGPIO |= 0x01;     // Enable clock to GPIOA

     // UART0 initialization
     UART0->CTL = 0;         // Disable UART0
     UART0->IBRD = 104;      // 16MHz/16=1MHz, 1MHz/104=9600 baud rate
     UART0->FBRD = 11;       // Fraction part, see Example 4-4
     UART0->CC = 0;          // Use system clock
     UART0->LCRH = 0x60;     // 8-bit, no parity, 1-stop bit, no FIFO
     UART0->CTL = 0x301;     // Enable UART0, TXE, RXE

     // UART0 TX0 and RX0 use PA0 and PA1. Set them up.
     GPIOA->DEN = 0x03;      // Make PA0 and PA1 as digital
     GPIOA->AFSEL = 0x03;    // Use PA0,PA1 alternate function
     GPIOA->PCTL = 0x11;     // Configure PA0 and PA1 for UART
  }
```

```
//Timer method for interrupts
//Method takes input from LM35Dz and prints external temperature as well as set temperature.
//Method also controls red and blue LED logic, or "AC" and "Heat"
void TIMER1A_Handler(void)
{
                        int difference = destemp-currtemp;
  int p;
                        int xc, xe, xh;
  ADC0->PSSI |= 8;        // Start a conversion sequence 3
  //Get temperature from LM35Dz, print external temperature and desired temperature. Wait 1 second and repeat.
      while((ADC0->RIS & 8) == 0) ;   // wait for conversion complete
      currtemp = (ADC0->SSFIFO3 * 594/4096) + 27;
      ADC0->ISC = 8;          // clear completion flag
      sprintf(buffer, "\r\nCurrent Temperature = %dF, Desired Temperature = %dF\n", currtemp, destemp);
      UART0_puts(buffer);
      delayMs(1000);
            ///////////////////////////////////////////////////////////////////////////////
/*
            PF1 Heating fan ->   PWM1->_2_CMPA
            PF2 Cooling fan ->   PWM1->_3_CMPB
            PF3 Exhaust fan ->   PWM1->_3_CMPA
*/
            if(difference<=2.5)
            {
                        p=1-(.133*(difference-2.5));
  xc = p*15999;
                        xh = 15000;
                        PWM1->_2_CMPA = xc;      //turn on cool
                        PWM1->_3_CMPB = xh;
  GPIOB->DATA |= 0x020;
                        GPIOB->DATA &= ~0x040; //turn off red LED
            }
            else if(-7.5>=difference>=7.5)
            {
                        p=1-(.133*(difference-7.5));
                        xe = p*15999;
                        PWM1->_3_CMPA = xe;
            }
            else
            {
                        p=(.133*(difference-2.5));
                        xh = p*15999;
                        PWM1->_3_CMPB = xh;
            //Turn on blue LED
                                    GPIOB->DATA |= 0x040;                  //Turn on red LED
                        GPIOB->DATA &= ~0x020;  //turn off blue LED

            }
}
                                    //}
            ///////////////////////////////////////////////////////////////////////////////
            /*          if(currtemp +7 < destemp) //Fan slow its so cold!!!!
                        {
                                    PWM1->_3_CMPA = x;
                                    x = 15000;
                        }
                        //If current temp is below desired temperature by 2 degrees C (too cold)
    if(currtemp < destemp)
    {
        GPIOB->DATA |= 0x040;                                    //Turn on red LED
                                        PWM1->_3_CMPA = x;
                                                x = 12000;      //decrease fan speed
                        }
                        //Else
    else
    {
                                                //Turn off red LED
        GPIOB->DATA &= ~0x040;
                                                //x=8000;
    }
                        if(currtemp > destemp + 7) //Max power its too hot!!!!
                        {
                                x = 750;
                                PWM1->_3_CMPA = x;
                        }
    //If current temp exceeds desired temperature by 2 degrees C (too warm)
    if(currtemp > destemp)
    {
                                                //Turn on blue LED
        GPIOB->DATA |= 0x020;
                                        x = 4000;
                                                PWM1->_3_CMPA = x;
                                                //Turn heating pads on here

                                                //Turn AC off here

    }

                        //Else
    else
    {
                                                //Turn off blue LED
        GPIOB->DATA &= ~0x020;
    }

                        //Match exhaust speeds here

} */

//Set up timer
void Timer1_init(void)
{
    SYSCTL->RCGCTIMER |= 2;      // Enable clock to Timer Module 1
    TIMER1->CTL = 0;             // Disable Timer1 before initialization
    TIMER1->CFG = 0x04;          // 16-bit option
    TIMER1->TAMR = 0x02;         // Periodic mode and up-counter
    TIMER1->TAPR = 250;          // 16000000 Hz / 250 = 64000 Hz
    TIMER1->TAILR = 64000;       // 64000 Hz / 64000 = 1 Hz
    TIMER1->ICR = 0x1;           // Clear the Timer1A timeout flag
    TIMER1->IMR |= 0x01;         // Enable Timer1A timeout interrupt
    TIMER1->CTL |= 0x01;         // Enable Timer1A after initialization
    NVIC->IP[21] = 4 << 5;       // Set interrupt priority to 4
    NVIC->ISER[0] |= 0x00200000;         // Enable IRQ21 (D21 of ISER[0])
```

```
}
/*
   SW1 is connected to PA4 pin, SW2 is connected to PA5. Both of them trigger PORTF interrupt
   If SW1 is pressed, decrease desired temperature by 1 degree C
   If SW2 is pressed, increase desired temperature by 1 degree C
   A delay of 1/4 second was added to avoid settemp incrementing or decrementing more than 1 time in 1 button press.
*/

//Switch Handler
void GPIOA_Handler(void)
{
   volatile int readback;

   // If SW1 (PA4) is pressed
   if (GPIOA->MIS & 0x10)
   {
   //Decrement settemp by 2
   destemp = destemp - 2;
      GPIOA->ICR |= 0x30;     // Clear the interrupt flag
        readback = GPIOA->ICR;      // A read to force clearing of interrupt flag
   delayMs(250);
    }
    // Else if SW2 (PA5) is pressed
    else if (GPIOA->MIS & 0x20)
    {
   //Increment settemp by 2
        destemp = destemp + 2;
   GPIOA->ICR |= 0x30;          // Clear the interrupt flag
          readback = GPIOA->ICR;    // A read to force clearing of interrupt flag
   delayMs(250);
     }

     // We shouldn't ever get here
     else
     {
   GPIOA->ICR = GPIOA->MIS;
   readback = GPIOA->ICR;   // A read to force clearing of interrupt flag
     }
}

void UART0Tx(char c)
{
   while((UART0->FR & 0x20) != 0); // Wait until Tx buffer not full
   UART0->DR = c;                    // Before giving it another byte
}

void UART0_puts(char* s)
{
              while (*s != 0)   // If not end of string
              UART0Tx(*s++);      // Send the character through UART0
}

// Delay n milliseconds (16 MHz CPU clock)
void delayMs(int n)
{
   int32_t i, j;
   for(i = 0 ; i < n; i++)
```

```
    for(j = 0; j < 3180; j++)
      {}  // Do nothing for 1 ms
}

void SystemInit(void)
{
  // Grant coprocessor access
  // This is required since TM4C123G has a floating point coprocessor
  SCB->CPACR |= 0x00f00000;
}
```