Design Engineering Report

Course: Computer Information Technology
Code: ICS2O, ICS3U, ICS4U
Author: Joseph Vretenar
Date: Friday, November 17, 2023

Table of Contents

Project 1.1. Voltage Division ***1***

Purpose **1**

Reference **1**

Procedure **1**

Media **2**

Reflection **2**

Project 1.2. Capacitor Visualizer ***3***

Purpose **3**

Reference **3**

Procedure **3**

Media **4**

Reflection **5**

Project 1.3. The Analog Oscillator ***7***

Purpose **7**

Reference **7**

Procedure **7**

Theory **7**

Media **8**

Reflection **8**

Project 1.4. The Counting Circuit ***9***

Overall Theory **9**

Reference **9**

Part A. The Analog Input **9**

Purpose 9
Procedure 9
Media 9

Part B. NAND Gate Oscillator **10**

Purpose 10
Reference 10
Procedure 10
Media 11

Part C. Decade Counter **12**

Purpose 12
Reference 13
Media 13

Part D. Decimal Counting Binary Up/Down Counter **14**
Purpose 14
Reference 14
Media 14

Part E. Binary Counting Decimal Decoder **15**
Purpose 15
Reference 15

Part F. Seven Segment Display **16**
Purpose 16
Reference 16
Procedure 16
Media 16
Reflection 17

Project 1.5. Binary Clock (ISP) ***19***

Purpose **19**

Procedure **19**

Media **20**

Reflection **22**

ICS3U ***23***

Project 2.1. Traffic Light ***25***

Purpose **25**

Reference **25**

Procedure **25**

Reflection **25**

Media **26**

Code **27**

Project 2.2. Binary Button Echo ***29***

Purpose **29**

Reference **29**

Procedure **29**

Reflection **30**

Media **30**

Code **31**

Project 2.3. PoV Word ***33***

Purpose **33**

Reference **33**

Procedure **33**

Reflection **34**
Media **34**
Code **34**
Project 2.4.1. Breadboard ATmega328P ***39***
Purpose **39**
Reference **39**
Procedure **39**
Reflection **40**
Media **41**
Code **43**
Project 2.4.2. Altoids Arduino ***45***
Purpose **45**
Reference **45**
Procedure **45**
Reflection **45**
Media **46**
Project 2.5 MatrixMadeEZ ***47***
Purpose **47**
Reference **47**
Procedure **47**
Reflection **48**
Media **48**
Code **49**
Project 2.6. Legacy PCB/Appliance ***53***
Reference **53**
Inspiration **53**
Procedure **53**
Procedure Final 54
Media **54**
Reflection **56**
Project 2.7. Medium ISP ***57***
Purpose **57**
Reference **57**

Procedure **57**
Reflection **58**
Media **59**
Code **60**
Project 2.8. DC Motor Tachometry ***63***
Purpose **63**
Reference **63**
Procedure **63**
Reflection **63**
Media **63**
Code **65**
Project 2.9. DC Motor Control ***66***
Media **66**
Project 2.10. Time-Sensitive Mechanics ***67***
Purpose **67**
Reference **67**
Procedure **67**
Reflection **67**
Media **68**
Code **68**
Project 2.11. Long ISP ***71***
Purpose **71**
Reference **71**
Procedure **71**
Reflection **72**
Media **73**
Code **74**
ICS4U ***85***
Project 3.1. GB Machine ***87***
Purpose **87**
Reference **87**
Procedure **87**
Reflection **87**

Media **88**

Project 3.2. CHUMP ***89***

Overall Theory **89**

Reference **89**

Project 3.2.1. Code **89**
Purpose 89
Procedure 89
Code 89

Project 3.2.2. Clock **90**
Purpose 90
Procedure 90
Media 90

Project 3.2.3. Counter **91**
Purpose 91
Procedure 91
Media 92

Project 3.2.4. Program EEPROM **93**
Purpose 93
Procedure 93
Media 94

Project 3.2.5. Control EEPROM **95**
Purpose 95
Procedure 95
Media 96

Project 3.2.6. Multiplexer **97**
Purpose 97
Procedure 97
Media 98

Project 3.2.7. Accumulator **99**
Purpose 99
Procedure 99
Media 99

Project 3.2.8. Address **100**
Purpose 100
Procedure 100
Media 100

Project 3.2.9. ALU **102**
Purpose 102
Procedure 102
Media 103

Project 3.2.10. RAM **104**
Purpose 104
Procedure 104
Media 104

Project 3.2.11. Code (Revisited) **106**
Purpose 106
Procedure 106
Code 106

Project 3.2.12. CHUMP Final **107**
Reflection 107
Media 107

Project 3.3. Short ISP ***108***

Purpose **108**

Reference **108**

Background **108**

Procedure **108**

Reflection **109**

Media **110**

Project 3.4. Keypad Matrix Echo ***112***

Purpose **112**

Reference **112**

Procedure **112**

Reflection **114**

Code **114**

Media **115**

Project 3.5. Pin Change Interrupt ***116***

Purpose **116**

Reference **116**

Procedure **116**

Reflection **117**

Code **118**

Media **121**

Project 3.6. Medium ISP ***122***

Purpose **122**

Reference **122**

Procedure **122**

Reflection **123**

Media **123**

Project 3.7. DDP: Legacy Shield ***124***

Purpose **124**
Reference **124**
Procedure **124**
Reflection **124**
Media **124**
Project 3.8. Long ISP ***126***
Purpose **126**
Reference **126**
Procedure **126**
Reflection **127**
Media **127**
Code **129**

Project 1.1. Voltage Division

Purpose

The purpose of this project is to learn about circuits, and how they work. In this project, the circuit uses a push button to change the color of a bicolor LED. The color of the LED will either be red or green based on which route the power is coming from. With this knowledge of circuits, more complex circuits will be easier to create as the course progresses.

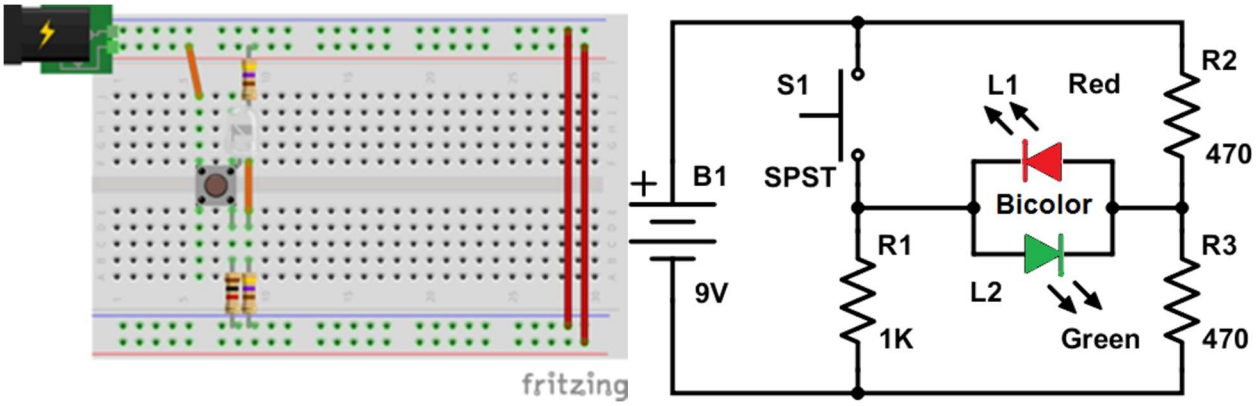
Reference

2018 ICS2O Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEL3M/1819/TasksFall.html#VoltageDivision

Procedure

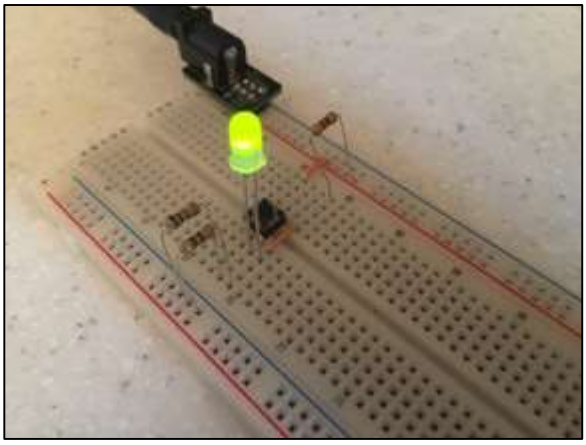
Parts List	
Component	Quantity
470Ω	2
1KΩ	1
Bicolor LED	1
Push Button	1
Assorted Wires	4
Breadboard	1
Power Plug	1

The parts needed for this project are listed to the right in the parts table. The circuit is quite simple in design.
On a breadboard, two wires connect the positive and ground rails on both sides. The rails are connected to a power plug which connects to a 9V battery. The button is connected to the positive rail on the left side. The bicolor LED's positive leg is connected to the other side of the button. The positive leg is also connected to a 1KΩ resistor that connects to the ground rail. The bicolor LED's negative leg is connected to a 470Ω resistor that comes from the positive rail. The negative lead of the bicolor LED is also connected to another 470Ω resistor that connects to the ground rail. When the button is not pushed down, the LED will be green. This happens because the power goes through one 470Ω resistor, then it goes to the bicolor LED's ground leg. The power then splits off and goes through the second 470Ω resistor and the 1KΩ resistor. When the button is pushed down, the power will go through the button, to the positive leg on the LED. This will make the LED red. The power will then split off and go through the second 470Ω resistor and the 1KΩ resistor.

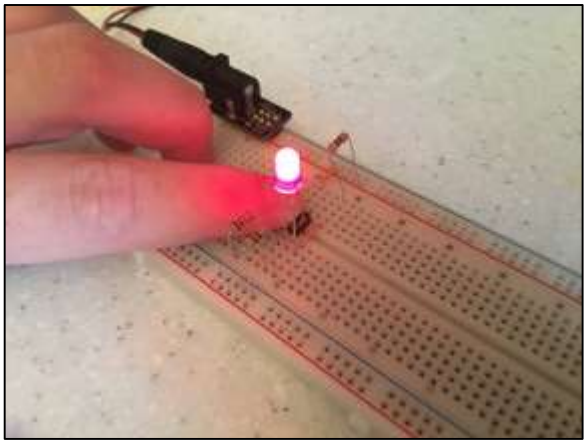


Media

https://youtu.be/fFN6bWEIMxE



Circuit with Button not pressed down



Circuit with Button pressed down

Reflection

During this project, I learned how a bicolor LED works, I also learned how switches and buttons work. At the start, I had some trouble creating the circuit because the schematic showed 2 different LEDs. This was confusing because LEDs are polarized, so when you line up two LEDs going different ways, one positive leg cannot get power because it is covered up by the ground leg. Once I found out that the 2 LEDs were actually one, this made the circuit so much easier. I enjoyed how the circuit worked once I finally wired it up. I feel like I can utilize the bicolor LED in future projects.

Project 1.2. Capacitor Visualizer

Purpose

The purpose of the capacitor visualizer is to demonstrate how capacitors work. Capacitors store a certain amount of voltage and dispense it over a period of time. This circuit uses a two pushbuttons, one bicolor LED and one yellow LED to visualize how a capacitor works. To visualize this we use two LEDs to show charging and discharging. When the capacitor is charging the bicolor LED will be red and the yellow one will be on to show power is moving through the circuit, and when the capacitor is discharging, the bicolor LED will be green, and the yellow LED will fade out.

Reference

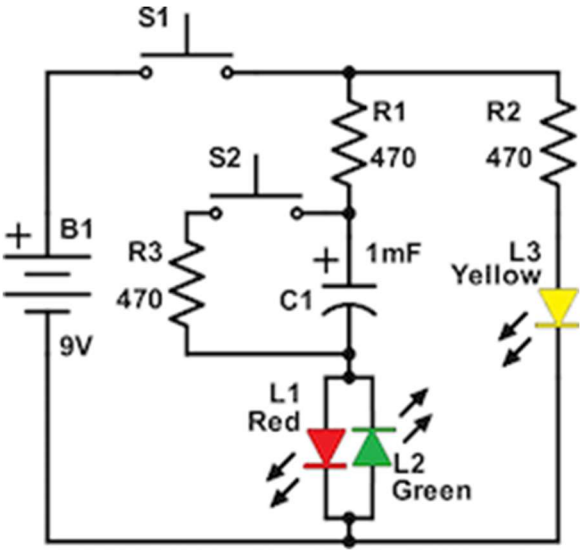
2018 ICS2O Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEL3M/1819/TasksFall.html#capacitor

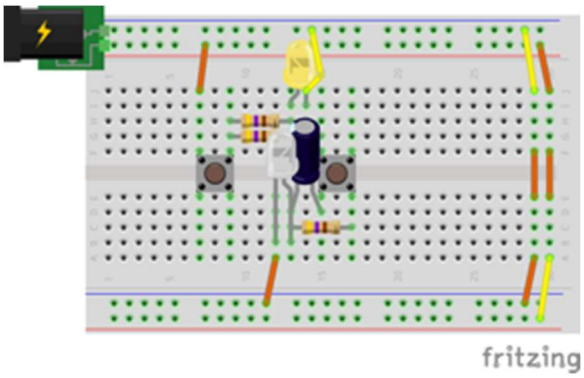
Procedure

The parts for this project are listed in the parts table on the right. This circuit is simple to wire up and does not use a lot of parts.

Parts List	
Component	Quantity
470Ω	3
Yellow LED	1
Bicolor LED	1
Push Button	2
Assorted Wires	3
Breadboard	1
Power Plug	1
1000µF Capacitor	1

On a breadboard, two wires connect the positive and ground rails to each other. These rails are connected to a 9V battery with a power plug. A wire connects the positive rail to one side of a button. On the other side of the button, there are two 470Ω resistors in parallel. One 470Ω resistor leads to a yellow LED which then goes to the ground rail. The other 470Ω resistor goes to one side of a second button. On that same side of the button, a 1000µF capacitor is placed. On the other side of the button, there will be a third 470Ω resistor that leads to the same row that the ground lead of the capacitor is in. This row is also connected to the positive lead of a bicolor LED. This LED connects to the ground rail. When the first button is pushed, the yellow LED will turn on, and the bicolor LED will turn red, and start fading until it is almost off. When the button is let go, the bicolor LED turns green and both the bicolor, and yellow LED fade until they are off. The second button discharges the capacitor and turns everything off.



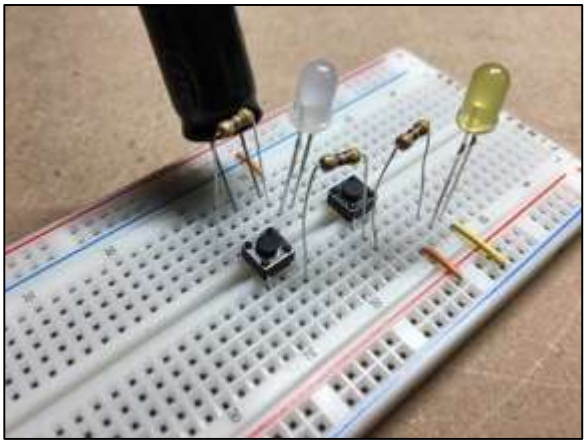


Time to charge

Resistance (Ω)	Capacitance (µF)	Theoretical 5τ (s)	Observed 5τ (s)
1000000	1	5.0	0.0
100000	10	5.0	0.0
100000	20	10.0	0.1
10000	100	5.0	2.3
6600	100	3.3	2.4
470	1000	2.4	3.6

Media

https://youtu.be/y5xOPFdmEX8



Capacitor Visualizer Breadboarded

Capacitor visualizer soldered

Reflection

This project wasn't too hard to wire up. With my knowledge of how to wire up circuits from the previous project, I was able to wire it up successfully on the first try. I felt pretty good about the circuit and filled out the "Time to charge" table. The time was a bit hard to record when the resistance was very high and the capacitance was low, because the digital multi meter (DMM) was not giving me the right reading on the full capacitance of the capacitor. Instead I had to use a stopwatch to estimate the time it took to charge. Since I went in early to get the table done, I was able to do the soldering done quickly. I had previous experience soldering with my Dad, so this was very easy and didn't take me too long. When I first tried the buttons, it didn't work. I looked at all my soldering and found that the yellow LED's lead was not fully connected to the PCB. I then went back to my soldering iron and made the connection between the PCB and the lead. The LEDs lit up and the circuit worked perfectly.

Project 1.3. The Analog Oscillator

Purpose

The purpose of the Analog Oscillator is to figure out how transistors work, and use them to create a circuit. Transistors are like switches except that they are not mechanical. This means that there is no bounce or time difference in the circuit when the transistor is turned on. This allows for instant switching between the different modes or paths the circuit is going.

Reference

2018 ICS2O Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEL3M/1819/TasksFall.html#AnalogOscillator

Procedure

The parts needed for this project are listed to the right in the parts table. This circuit is a bit fussy in the sense that you have to try different resistor and capacitor combinations to get the right result.

Parts List	
Component	Quantity
1KΩ Resistor	2
47KΩ Resistor	2
5mm Red LED	2
NPN Transistor	2
10µF Capacitor	2
Assorted Wires	4
Full+ Breadboard	1
Power Plug	1
9V Battery	1

On a bread board, two 1KΩ resistors and two 47KΩ resistors are placed on the positive rail, right beside each other with the 1KΩ resistors on the outside and the 47KΩ resistors on the inside. Each of the 1KΩ resistors are connected to the positive leads of two different red LEDs, the negative lead of the LEDs go into the collector pin on the transistors and also into the positive lead of the 10µF capacitors. The negative lead of the capacitors is then connected to the middle pin, or base pin of the opposite LEDs transistor. The negative lead of the capacitors also go to the 47KΩ resistors that are in the positive rail. The emitter lead of the transistors is then connected to the ground rail on the breadboard. When this circuit is plugged in, the LEDs should oscillate, or take turns going on and off. This rate at which these LEDs turn on and off, can be adjusted if the 47KΩ resistor is changed to a lower or higher resistance. When the resistance is lower, the lights will flash faster, and when the resistance is higher, the lights will flash at a slower speed.

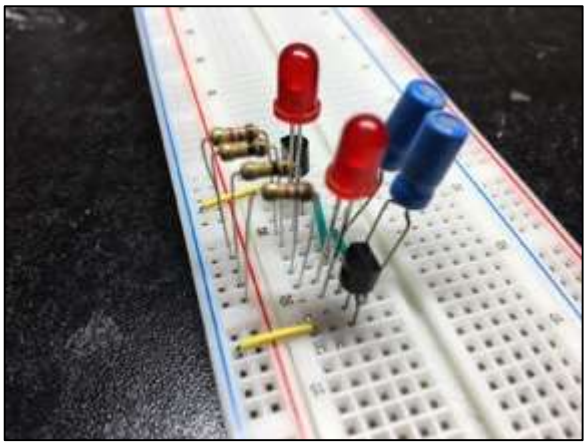
Theory

The oscillating effect is created when the capacitor charges and discharges. When one capacitor charges, the other capacitor is discharging. When the capacitor discharges, it sends voltage to the base pin of one transistor. This then turns one of the LEDs on because when the base pin is given power, it allows electricity to flow through the transistor, and turn on the LED. Since each capacitor takes turns charging and discharging, it will create the oscillating effect where you see the LEDs flashing.

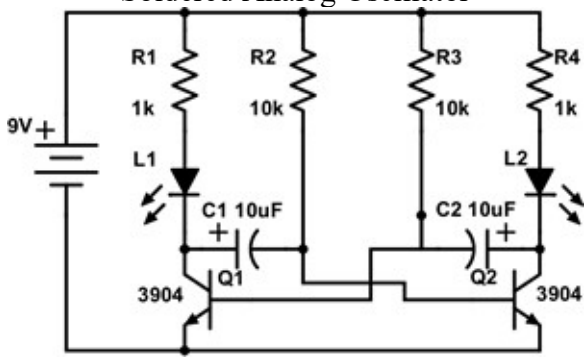
Media

https://youtu.be/cf8yYabTgYw

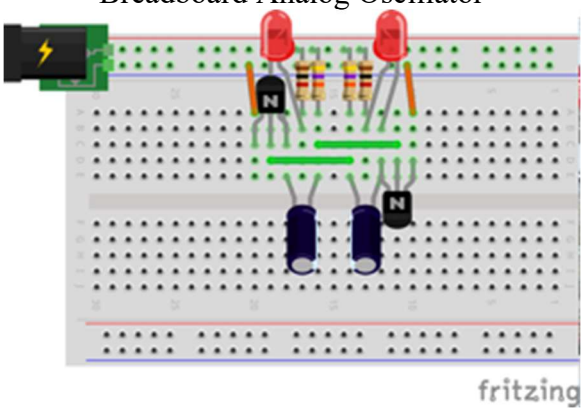
Soldered Analog Oscillator



Breadboard Analog Oscillator



Analog Oscillator Schematic



Analog Oscillator Fritzing diagram

Reflection

I found that this circuit was a bit hard to create at first. I found that I had to use a lot of wires in the circuit. Once I got it working I went back and modified it so that it looked better, was more compact, and used fewer wires. I found that playing around with the resistance was also hard, because you needed to see that the circuit had a on and off to the LEDs. Once I figured out what I had to change, I experimented with it a little more, just to see how it would look. I then got all my parts and soldered the components to the PCB. This was easy because we already did one soldering project before and I have gotten good at soldering.

Project 1.4. The Counting Circuit

Overall Theory

This circuit uses an integrated circuit to create an oscillating effect. That effect powers some counting integrated chips that count from zero to nine on a seven segment display.

Reference

2018 ICS2O Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEL3M/1819/TasksFall.html#counting

Part A. The Analog Input

Purpose

The purpose of the input is to create input for the circuit. This is done by connecting a resistor and a button from V+ to ground.

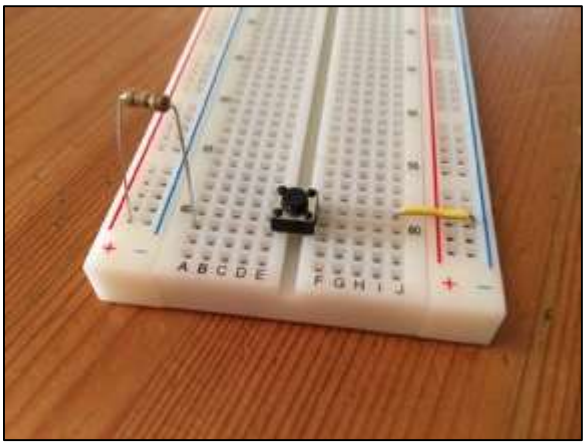
Procedure

To create the Analog input, there are only a couple of parts. These parts are in the parts table to the right.

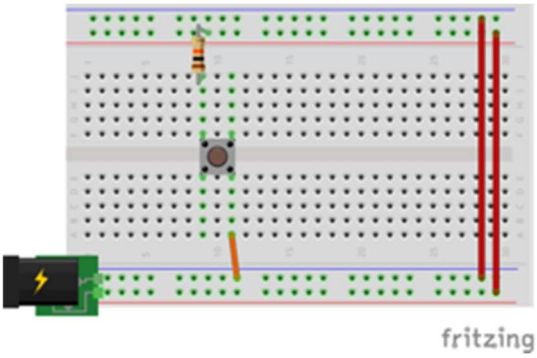
This circuit is not too challenging to create. It uses a push button that connects a 10KΩ resistor to ground when the button is pressed. This creates a pull up resistor effect and allows a high signal to reach the base pins from the chip when the button is not pressed, and allows a low signal to reach the pins when the chip is pressed.

Component	Quantity
10KΩ Resistor	1
Push Button	1
Wires	1
Power Plug	1
9V Battery	1

Media



Pull up resistor configuration

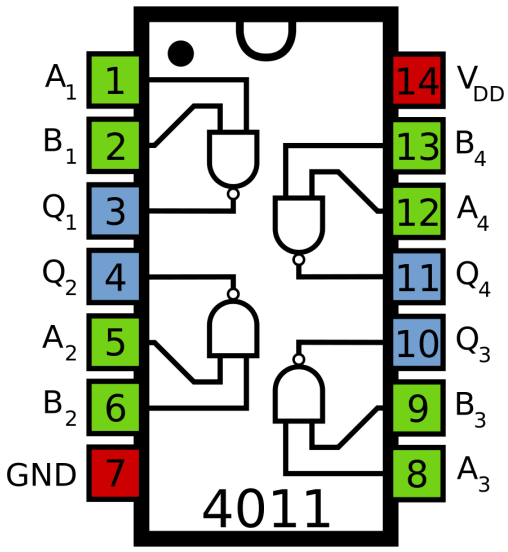


Pull up resistor fritzing diagram

Part B. NAND Gate Oscillator

Purpose

The purpose of the NAND gate oscillator is to create an oscillating effect for the rest of the circuit. This will allow our decade counter to count up or down. This oscillating effect is a square wave, which means that it will go on and off instantly. The chip that is used is a 4011 chip. This chip has 14 pins and is in a DIP configuration. DIP stands for dual in-line package, this means that there are pins on both sides of the chip and it needs two rows to sit in. This chips normally sit across the middle of the breadboard. The chip is automatically connected to power and ground through pin 14 and pin 7. There are a total of four NAND gates in this chip. The input pins are colored green and the output pins are blue. In the 3rd schematic in the media section, there are two different highlighted parts RC1 in green, and RC2 in red. These are two pretty special parts: RC1 controls the power and how long the oscillating effect will go on for; where RC2 controls the oscillating effect itself. RC1 has a capacitor that gets charged when the button is pressed down. When the button is up, the voltage that was stored in the capacitor will then go to pin 2 on the chip. This will give that pin a high signal and create the output as high. The resistor will slowly drain the capacitor which turns the circuit off. If that resistor is removed, the circuit will run until the battery drains because the base pins do not actually take any power. RC2 controls the oscillating effect by allowing voltage to send a signal back through the 470Ω resistor into pin 1 on the chip. This will make the output on pin 3 low, which then gets sent back to pin 1 which makes the output high, and this repeats until the capacitor is drained.



Reference

2018 ICS2O Engineering Tasks:
http://mail.rsgc.in.ca//~cdarcy/Datasheets/CD4011.pdf

Procedure

This circuit was challenging to wire up and it takes some time to get it right. It uses a lot of different parts to create the oscillating effect.

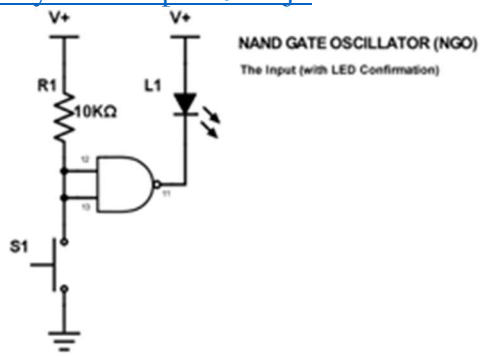
Parts List	
Component	Quantity
1MΩ Resistor	2
470Ω Resistor	1
10µF Capacitor	1
0.1µF Capacitor	1
Red LED	1
Signal Diode	1
4011 NAND Gate CMOS IC	1
Wires	15
Section A parts	5

The 4011 chip should have the 14th pin connected to the power supply and the 7th pin connected to ground. This circuit connects to a pull up resistor configuration from the last section, connecting to the 12th and 13th pins on the 4011 chip. These then lead to a signal diode. Which goes to a 10µF capacitor and a 1MΩ resistor that are parallel. The wire then connects to the 2nd pin on the chip. This then meets up with a 1MΩ resistor and the 4th and 5th pin on the chip. The 1MΩ resistor leads off of the circuit and breaks off into two different parts. One of those is a 470Ω resistor that travels to the 1st pin on the chip. The second part is a 0.1µF capacitor which meets up with the 4th pin on the chip. The 4th pin and the 0.1µF capacitor then travel to the

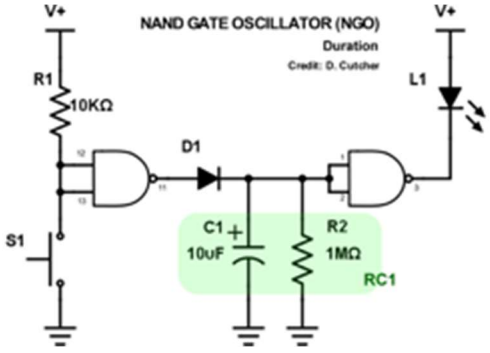
8th and 9th pin on the chip. The 10th pin on the chip then leads to the cathode or ground lead of an LED. The LEDs anode or positive lead is then connected to the power supply.

Media

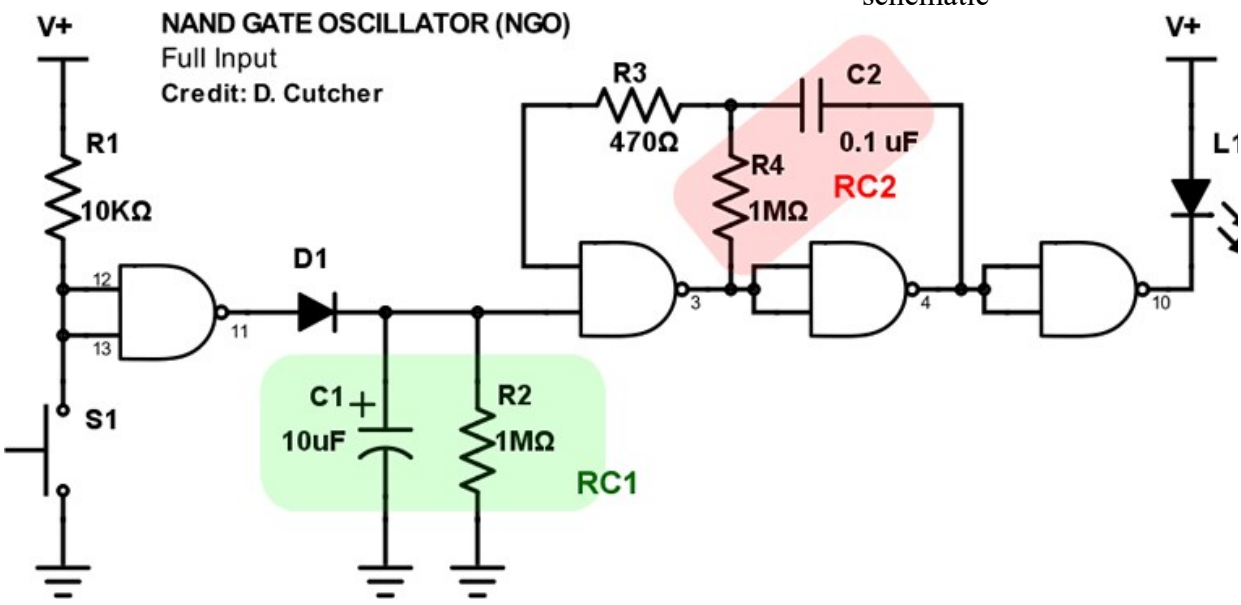
https://youtu.be/tp-Ek9zIRjo



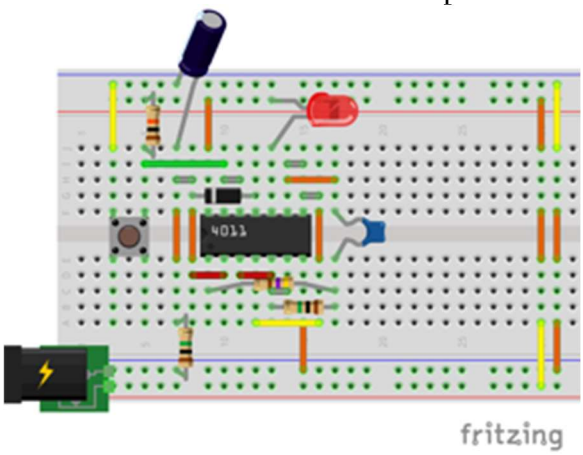
Input for the NAND gate oscillator schematic



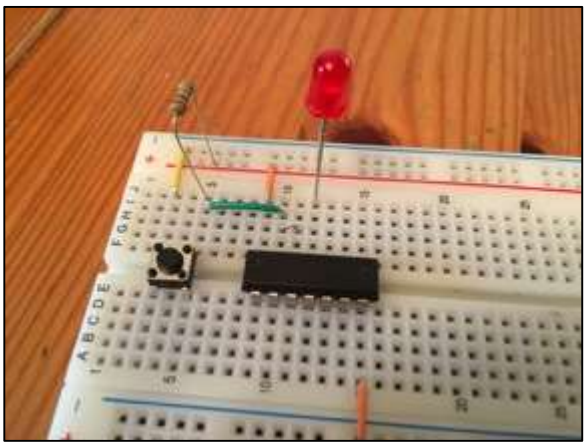
Duration section for the NAND gate oscillator schematic



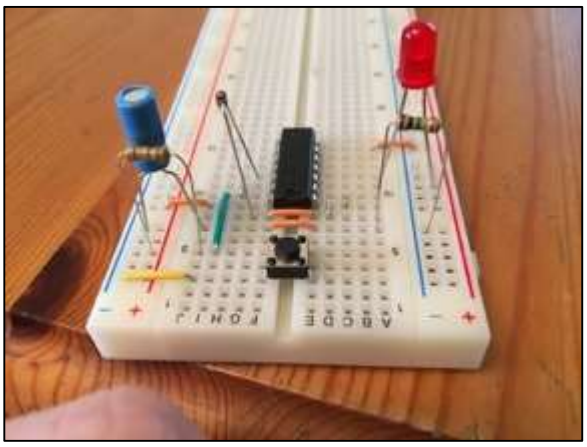
Full input for the final circuit schematic



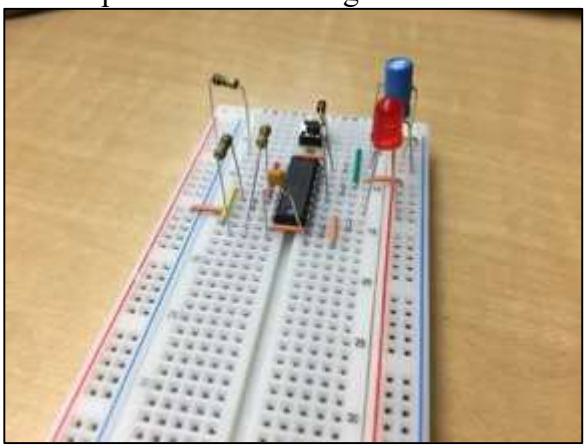
Fritzing diagram for the finished circuit



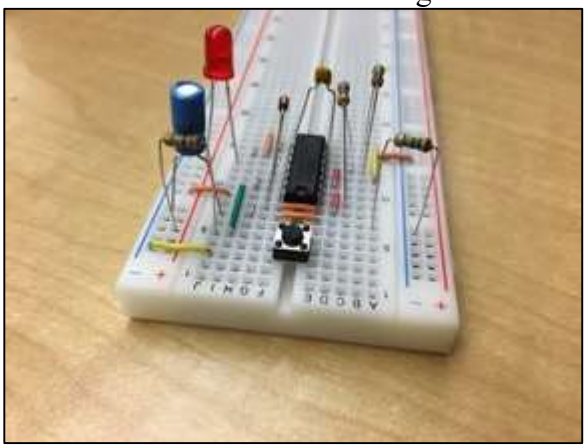
Input for the NAND gate oscillator



Duration section for the NAND gate oscillator



Right view of full input

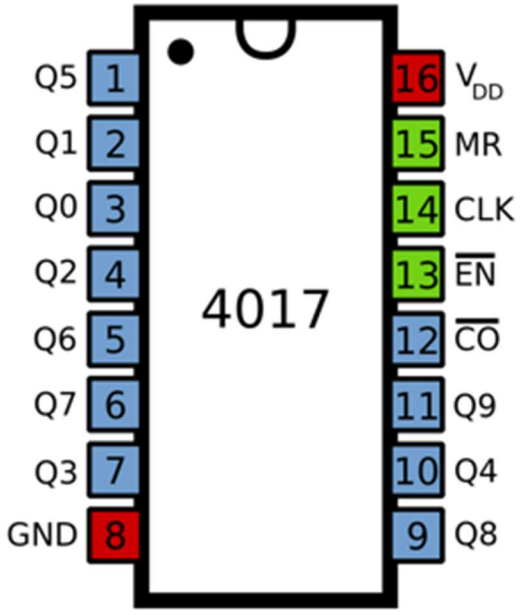


Left view of full input

Part C. Decade Counter

Purpose

The purpose of the 4017 chip is to turn the NAND gate oscillator into a counter that counts in decimal which is how we normally count. It will then display these numbers counting up from 1 to 10 on 10 different LEDs. The input of this chip goes to pin 14. This is the click pin and is what makes it count. This pin is connected to the output of the NAND gate oscillator. Pin 16 is connected to power and pin 8 is connect to ground. Pin 13 is the enable output pin and has to be grounded for the circuit to work. Pin 15 is the reset input pin and will be grounded for normal operation. If it is connected to power, then the input will restart at 0. Pin 12 is the divide by 10 pin and will be connected to an 11th LED. This LED will only be on for the first 5 LEDs and will turn off for the rest. Pins 1-7 and 9-11 will be the output pins for

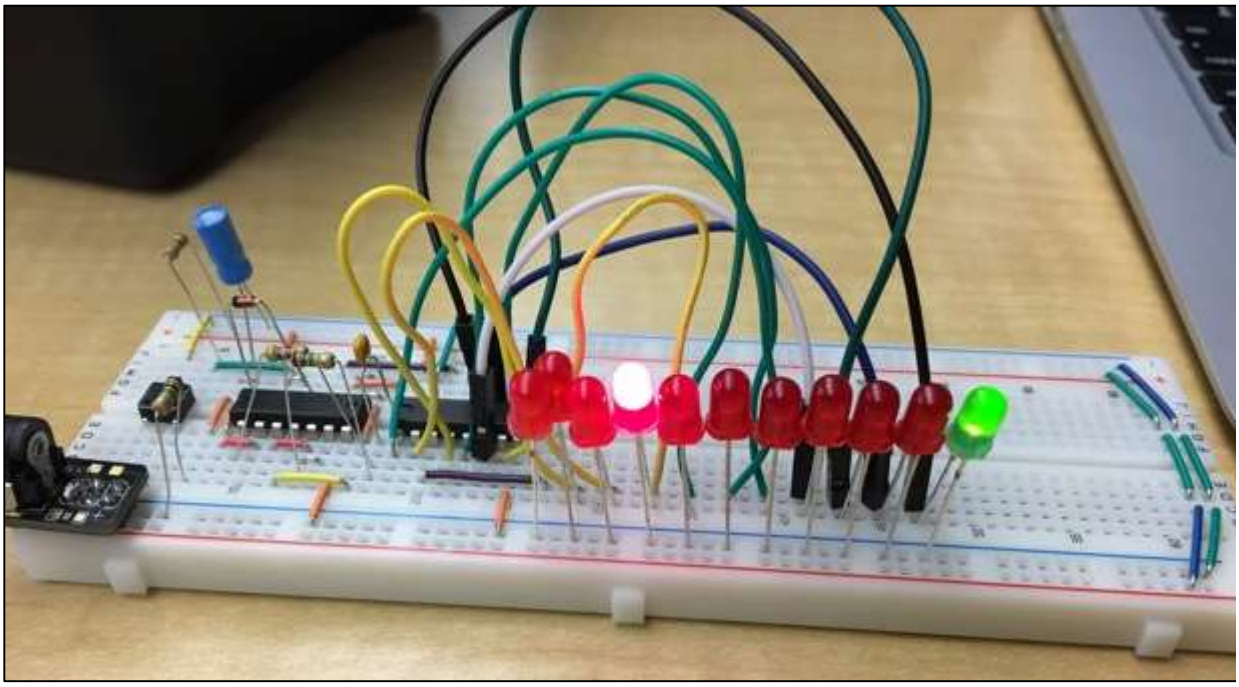


the other LEDs. The LEDs will start at pin 3 and follow the order from 1-10.

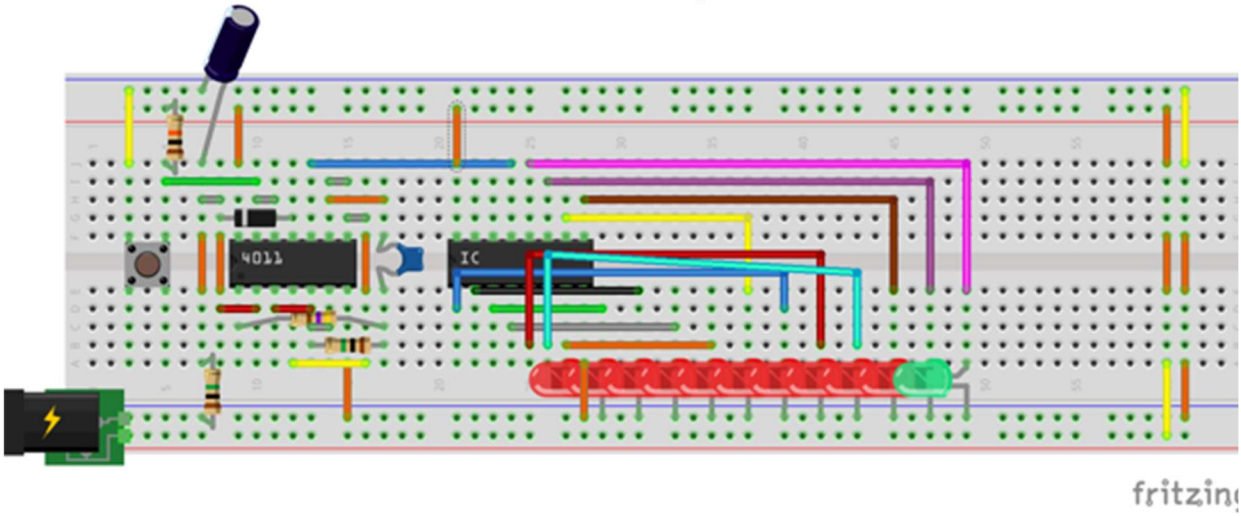
Reference

http://mail.rsgc.on.ca/~cdarcy/Datasheets/CD4017.pdf

Media



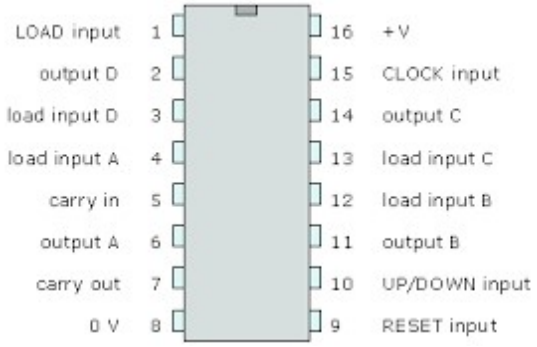
Part C of the Counting Circuit



Part C of the Counting Circuit Fritzing Diagram

Part D. Decimal Counting Binary Up/Down Counter

Purpose

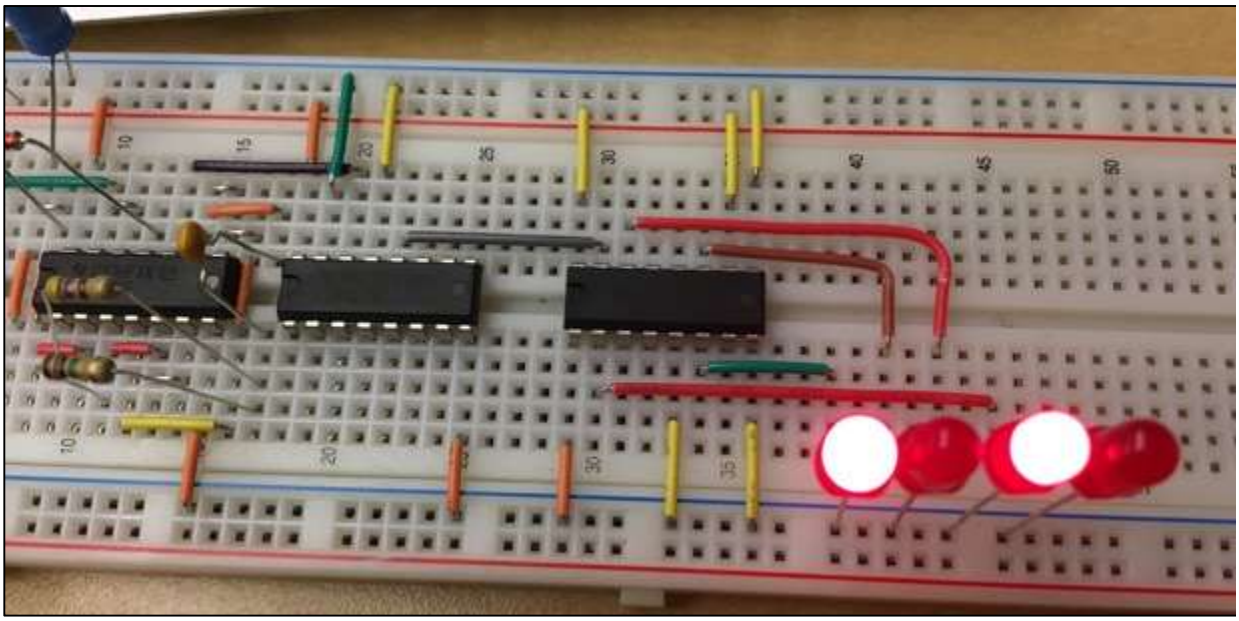


The purpose of the 4510 chip is to turn decimal counting to binary. It changes the numbers 1 to 10 to binary that shows up as 4 different LEDs on the breadboard. The LEDs are the numbers 1, 2, 4, and 8. These numbers are 2 to the power of 0, 1, 2, and 3. Pin 16 is connected to power and pin 8 to ground. This pin is powered from the divide by 10 pin or the 12th pin on the 4017 chip. This will go into pin 15 which is the clock input. Pin 3, 4, 12, and 13 are not needed because they are used for binary input but we are using a decimal input. Pin 1 and pin 5 are grounded for normal operation. For pin 10 it has two different meanings. If you want to count up, the pin has to be high. If you want to count down, the pin has to be low. The reset input will be set to low, if it is set to high, it will reset the counting from 0.

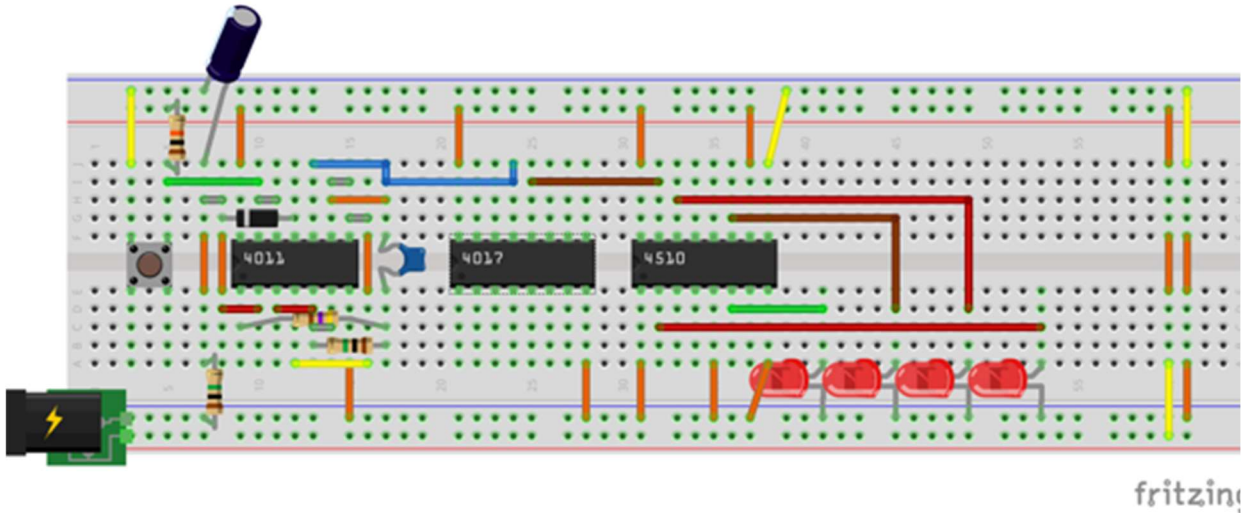
Reference

http://mail.rsgc.on.ca/~cdarcy/Datasheets/CD4510.pdf

Media



Part D of the Counting Circuit

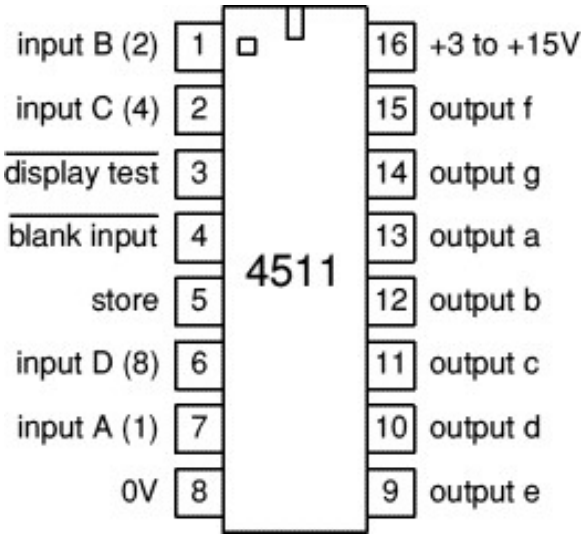


Part D of the Counting Circuit Fritzing Diagram

Part E. Binary Counting Decimal Decoder

Purpose

The purpose of the 4511 chip is to change the binary output from the 4510 to decimal numbers that can be shown on a seven segment display. Pin 16 is connected to power and pin 8 is connected to ground. Pins 1, 2, 6, and 7 will be connected to their respective outputs from the 4510 chip. Pin 5 will be grounded. If it is high, it will store a number and start from that number. When display test is grounded, it will turn on all the lights on the seven segment display. If it is set to high, it will not do anything. If blank input is grounded then the seven segment display will show nothing. It will be high for normal operation. Pins 9 to 15 will be the outputs and will be what is actually connected to the seven segment display. The seven segment display has 7 different pins that are labelled from A to F. When this chip gets the binary coded decimal from the 4510 chip, it will send a signal through the different pins to the seven segment display to turn on different numbers.



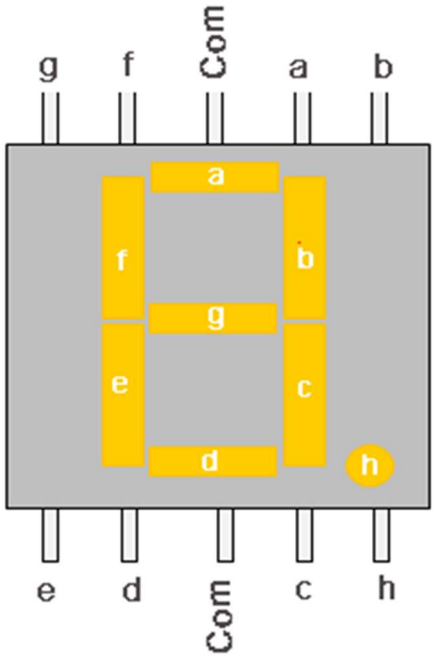
Reference

http://mail.rsgc.on.ca/~cdarcy/Datasheets/CD4511.pdf

Part F. Seven Segment Display

Purpose

The purpose of the seven segment display is to show the binary coded decimal as a seven segment number. There are 10 pins in total on the seven segment display. There are two types of 7-segment displays: common cathode and common anode. The difference between the two displays is the common cathode has all the cathodes of the 7-segments connected directly together and the common anode has all the anodes of the 7-segments connected together. There are 8 other pins, we only need 7. The pins A to F will be connected to the 4511 chip outputs that are labeled A to F. The pins from the 4511 chip will have resistors connected to them. They need resistors because the display is just 8 LEDs and they will burn out if the resistor is not limiting the current. The seven segment display will count from 0 to 9 and then restart. It will run until the capacitor in the NAND gate oscillator runs out.



Reference

http://mail.rsgc.on.ca/~cdarcy/Datasheets/7SegmentDisplay.pdf

Procedure

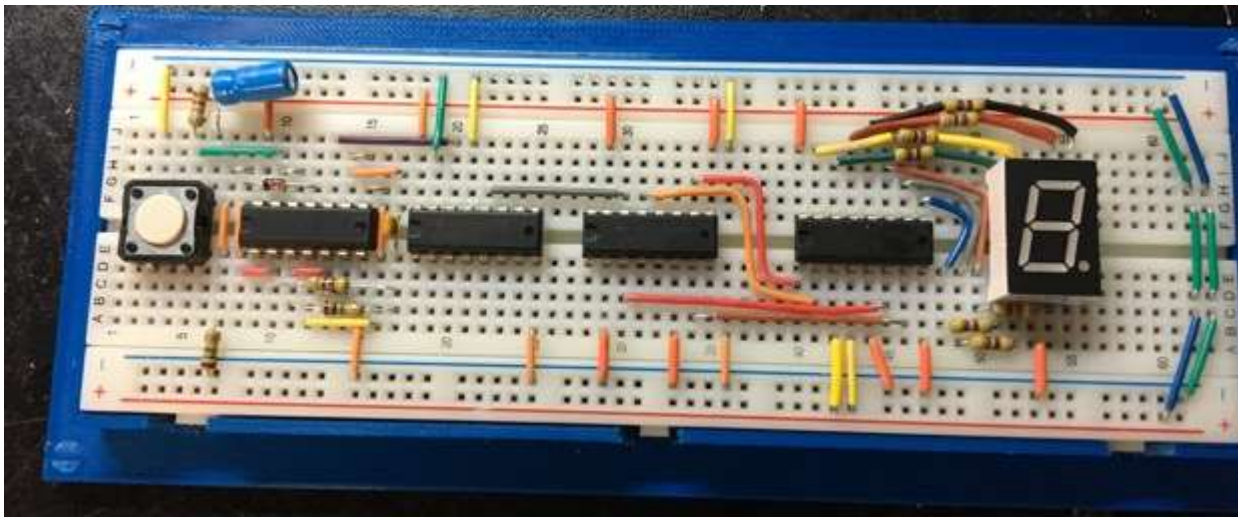
The wiring from the chips is pretty straight forward. The divide by 10 pin from the 4017
connects to the input of the 4510. The outputs of the 4510 are then connected to the respective inputs on the 4511. The 4511 outputs are then connected to their respective inputs on the seven segment displays.

Parts List	
Component	Quantity
470Ω Resistor	7
Seven Segment Display	1
4511 IC	1
4510 IC	1
4017 IC	1
Wires	21
Section A and B parts	28

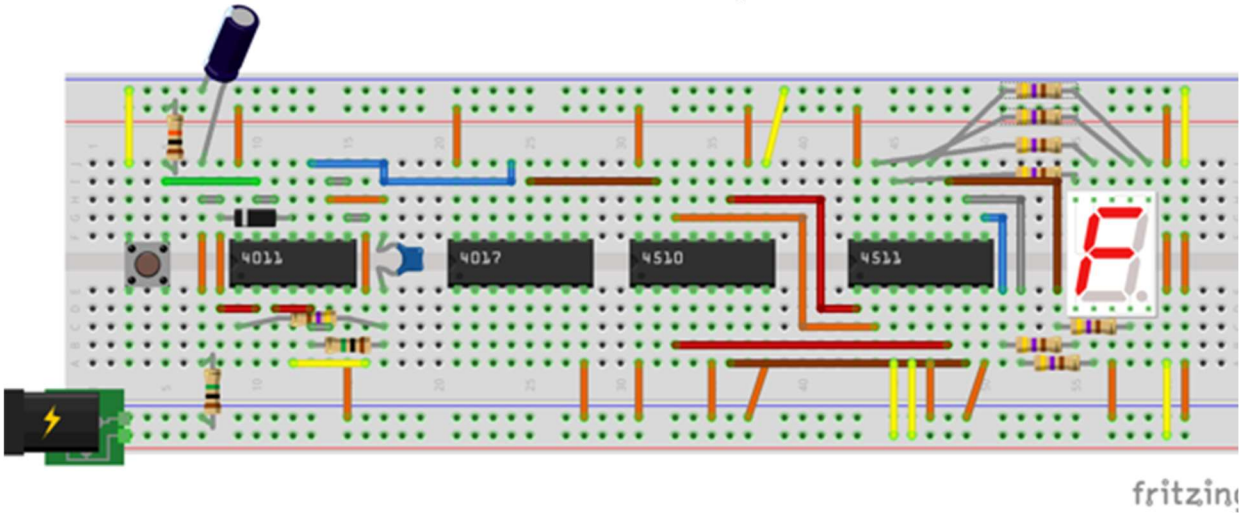
The one hard part is to make sure the correct pins are connected to ground and power on the different chips. On all three chips, pin 8 is connected to ground and pin 16 is connected to power. Other than that, the chips have different power and ground connections. On the 4017 pins 15 and 13 are also connected to ground. On the 4510 pins 1, 5 and 9 are also connected to ground and pin 10 is connected to power. On the 4511 pin 5 is connected to ground and pins 3 and 4 are connected to power.

Media

https://youtu.be/Xx0piAinhjs



Part E and F of the Counting Circuit



Part E and F of the Counting Circuit Fritzing Diagram

Reflection

The circuit was more difficult than previous circuits but I was able to get it done pretty easily. I had some trouble with the seven segment display the first time I put it in my circuit. I had some difficulty understanding where some of the pins on the 4511 chip should go, whether it was ground or power. I fiddled with some of the pins and finally got the result I wanted. Then another problem came up, some of the numbers on the seven segment display were not correct when it was counting. This was because some the resistors that were connected to the pins weren't fully in or were connected to a different pin. Once I fixed the resistors, I was able to get the result I wanted.

Project 1.5. Binary Clock (ISP)

Purpose

The purpose of the ISP is to take what I have learned over the semester and put it into a project that I want to create. It took a while to find a good project to do for my ISP but I finally settled on a Binary Clock. This integrates a lot of components we have learned about, such as ICs, LEDs, resistors, capacitors, buttons, signal diodes and soldering.

Procedure

The clock will require a 1 Hz pulse, binary counters, something to switch the minutes, and hours.

Parts List	
Component	Quantity
74HC393 IC	3
74HC08 IC	2
74HC14	1
Red LEDs	7
Orange LEDs	6
Yellow LEDs	5
330Ω Resistor	17
47KΩ Resistor	5
10KΩ Resistor	6
220Ω Resistor	1
22KΩ Resistor	1
4.7KΩ Resistor	1
555 Timer IC	1
100nf capacitor	3
10μf capacitor	1
1N4148	4
Wires	LOTS!!!!

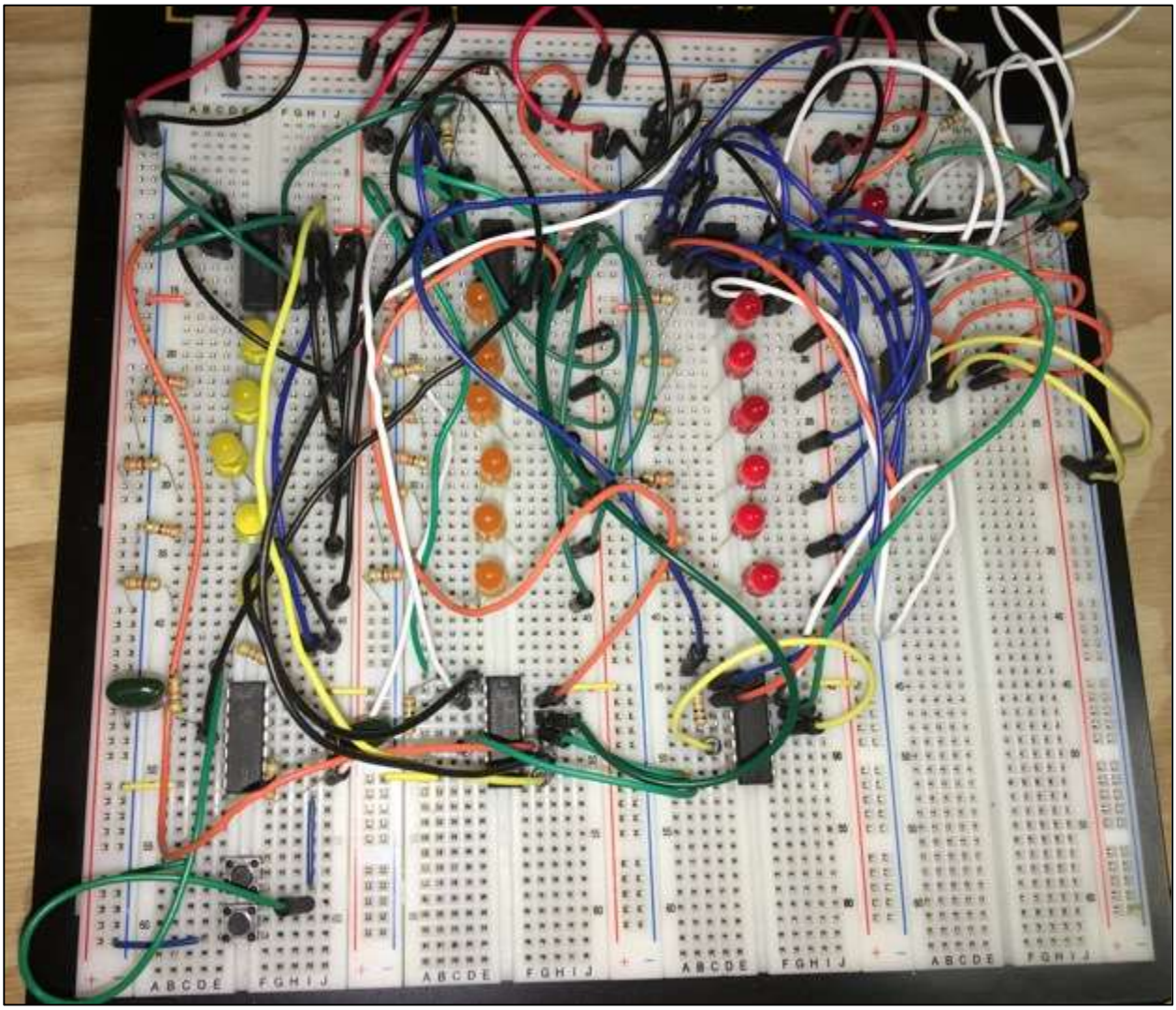
For the project 4 breadboards are required. One breadboard has a 555 with pin 8 and 4 to power and pin 1 to ground. Pins 4 and 7 are connected with a 10KΩ resistor. Pins 7 and 2 are connected with one 47KΩ resistor, a 22KΩ resistor, a 10KΩ, and a 4.7KΩ resistor which will add to a total resistance of 83.7KΩ. This is what controls the 1 Hz output that comes on pin 3. Pins 2 and 6 are connected with a wire, and pin 6 has a 10μf capacitor connected to ground. Pin 5 has a 100nf capacitor connected to ground. Pin 3 is the output and splits off in two different paths. One path has a 220Ω resistor connected to an LED that goes to ground, and the other path has the output that we will feed into the clock.

The second part that has to be set up is the "set hours" and "set minutes". This uses a inverter gate chip. Pin 14 is connected to power and pin 7 is connected to ground. One 47KΩ resistor comes from power and breaks off into two different paths. One path has another 47KΩ resistor connected to the 9th pin and it also has a 100nf capacitor going to ground. The other path has a button that is connected to ground. Pin 8 is one of the outputs, this is the "set minutes" output and goes through two 1N4148 signal diodes into the minutes and seconds binary counters. Another two 47KΩ resistors are set up on the other side the only difference is that the second 47KΩ resistor connects to pin 3. The output on the 4th pin is the "set hours" output. This also goes through two 1N4148 signal diodes, but it goes to the hours and minutes binary counters.

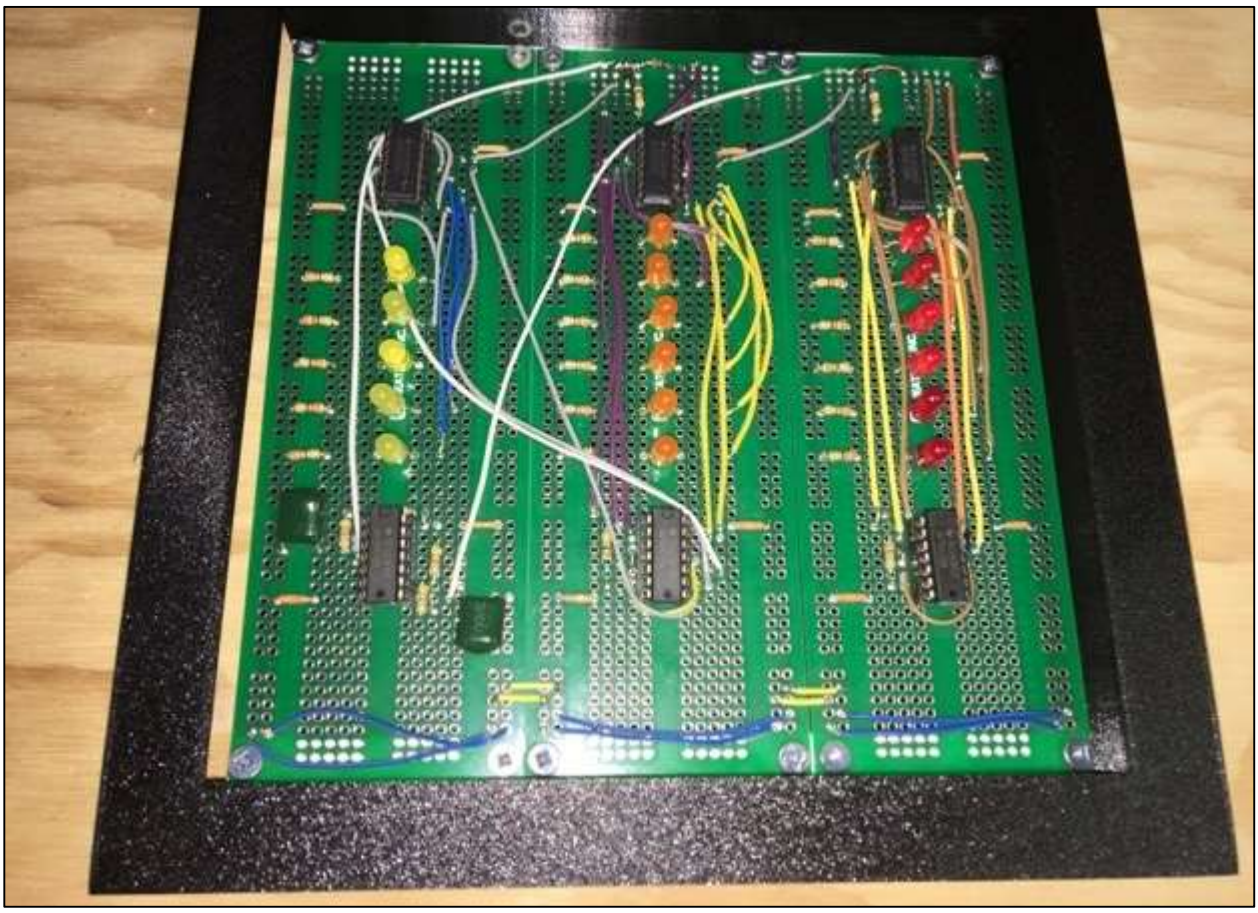
The next step is to connect the binary counters. For all three pin 14 is connected to power and pin 7 is connected to ground. Each binary counter has two CLK pins and two RES pins. The CLK pin is the input for the clock signal to power the counters. The RES pins are used to reset the counting to zero. The RES pins are connected to each other so that when one is triggered, the other one will reset as well. The CLK pins are not connected together. The CLK2 pin is connected to the 1Hz square wave from the 555 timer for the seconds. The CLK2 pins on the other ones are connected to one of the 1N4148 signal diodes that is attached to the 4th pin

through a 10KΩ resistor. The CLK1 pins are all connected to the 8th pin which will make the signal start on the other side once the 4th LED turns off. There are 4 outputs on either side of the pins. This means that this chip is able to have 8 total outputs. The output is set up so that when the LEDs representing 32, 16, 8 and 4 are about to go on, it will reset the counting. This happens because there is an AND gate that takes the output from those LEDs that will all add up and reset that counter and add another LED onto the next counter.

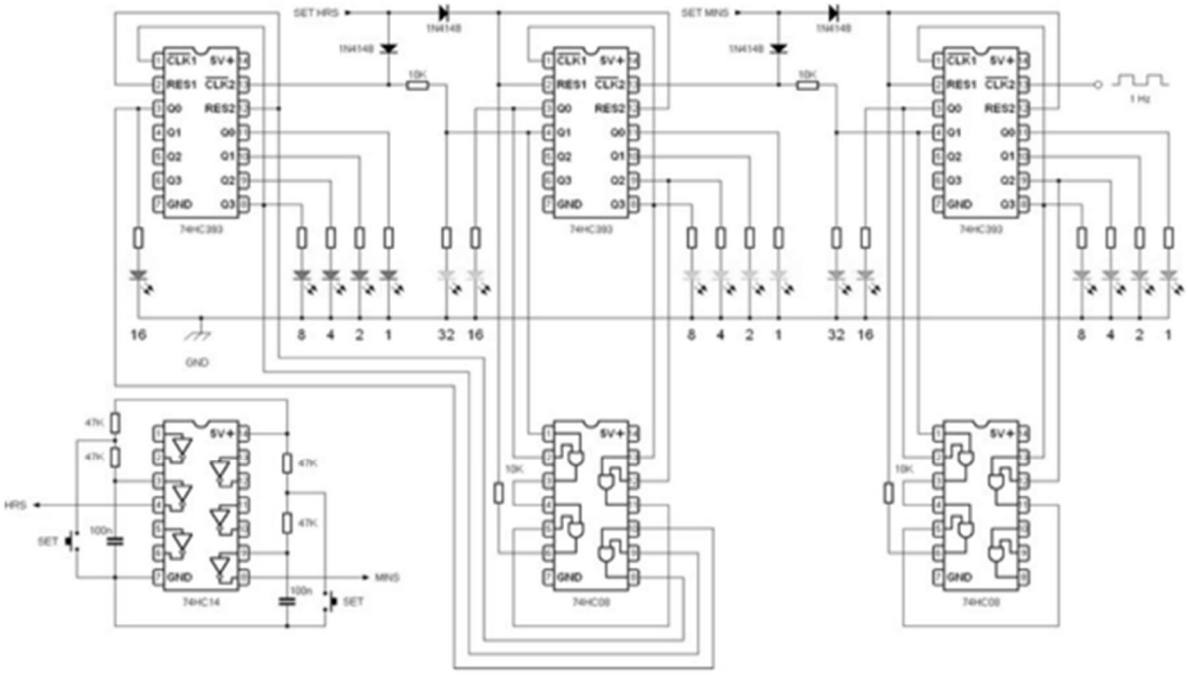
Media
https://youtu.be/8hzn7ZrmAFQ



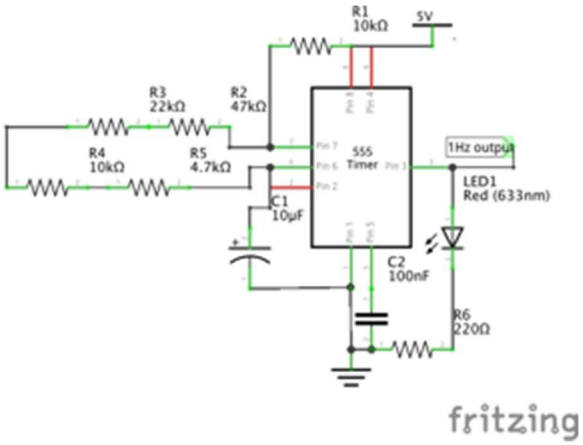
Binary Clock Prototype



3D printed Binary Clock Case and Fully Soldered Clock



Binary Clock Schematic



1Hz Output Schematic

Reflection

I found this project very interesting to do and it was very enjoyable. I had to use all the skills I had learned throughout this course in this project. I had the most fun wiring up, trouble shooting, and even coming up with the ideas for the project. I did run into one problem when I first wired up the circuit. I found that my wiring was wrong. I then took out all my wires and restarted. The second time it worked. I was very happy with the result of my prototype and I wanted to solder it. I did run into a big problem the first time I soldered the 555. I had put the 555 and the voltage regulator on the same breadboard so that the regulator wouldn't have to be on the front. The problem was that I had made the mistake of soldering the regulator in the wrong way. This meant that I had fried my 555 timer. So since I used half a breadboard, I had another half to work with. Luckily, one of my friends wasn't using their 555 and let me use it. I then soldered that up and it worked perfectly. I finished the rest of my soldering and am 3D printing a case for it. I think this is the perfect project to finish the course off with. I have really enjoyed this course and learned a lot in the process.

ICS3U

Project 2.1. Traffic Light

Purpose

The purpose of the Traffic light is to learn how to code a basic program for the Arduino. It also helps to develop our understanding of the Arduino IDE and the ATmega328p chip on the Arduino.

Reference

2019/20 ICS3U Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#TrafficLight

Procedure

This project requires a PCB that we use to solder the parts on, a row of five right header pins to connect the PCB to the Arduino, one 470Ω resistor but you can just use a wire in its place, one red LED for the red light, one yellow LED for the yellow light, one green LED for the green light, and one green LED for the left turn signal, and a small bit of copper stripboard for the red LED on top.

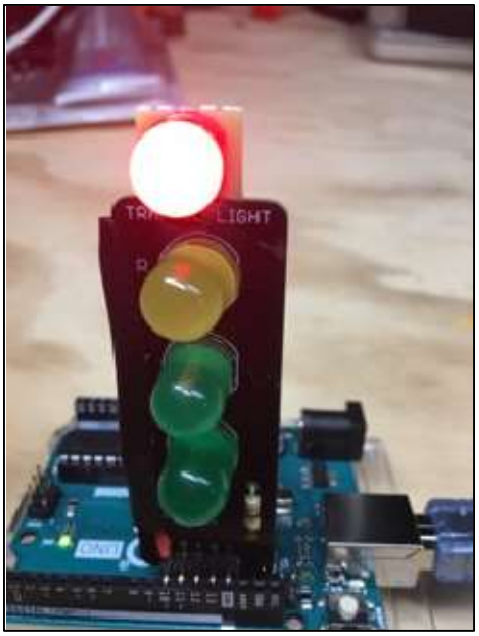
Parts List	
Component	Quantity
Red LED	1
Yellow LED	1
Green LEDs	2
PCB	1
Right header pins	5
470Ω resistor	1
Copper stripboard	1

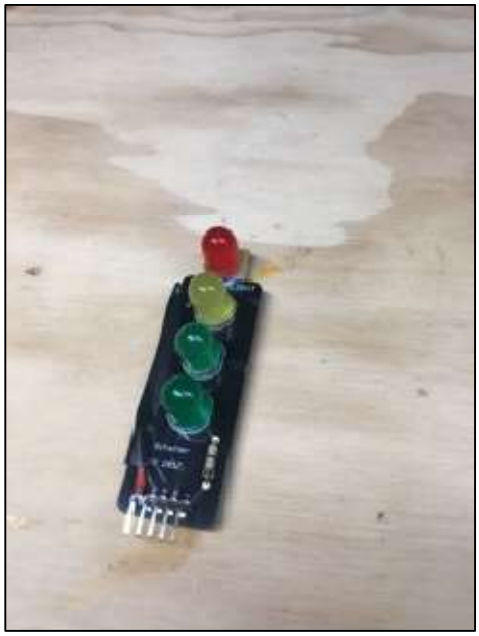
On the PCB, the LEDs need to be soldered in the correct spots. The right header pins are soldered at the bottom to connect the PCB in a vertical position to the Arduino. The 470Ω resistor is needed to connect the negative leads of the LED to the ground pin so they can complete the circuit. The resistor is not required it will just make the LEDs less bright.

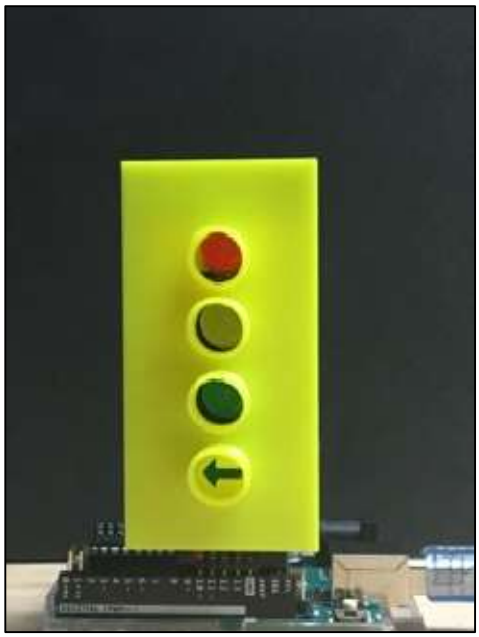
Reflection

This project was a pretty good introduction into programming. I added an extra LED into my project, except the only problem was adding another space for an LED on the PCB. I added some copper stripboard that I got from the DES on top of my PCB. This was an idea that was given to me by my guitar teacher when I was talking to him about the ACES course. This meant that I had to add extra code, but with a couple tutorials and some help from my Dad I learned how to add a function into the program and was able to create a loop function that loops a certain amount of times. I enjoyed this project and found it very interesting to learn new ways to improve my code.

Media







YouTube video: https://www.youtube.com/watch?v=7ZqMaltRnxc

Code

```
// Project: Traffic Light
// Purpose: Creating a traffic light using code.
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2019 09 28
// Status : Working
// Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#TrafficLight

uint8_t aGreen = 13; //used to create a unsigned integer to represent the pin number for the different LEDs
uint8_t green = 12;
uint8_t yellow = 11;
uint8_t red = 10;
uint16_t durG = 4000; //used to create a unsigned integer to represent the duration the LEDs are on
uint16_t durY = 2000;
uint16_t durR = 6000;
uint16_t durAG = 250;

void setup() {
  pinMode(red, OUTPUT); //sets the different pins from the unsigned integers to Outputs so the Arduino knows to only use them
  pinMode(yellow, OUTPUT);
  pinMode(green, OUTPUT);
  pinMode(aGreen, OUTPUT);
}

void loop() {
  digitalWrite(green, 1); //turns on the green LED
  advanceGreen(10); //starts to run the advanced green function 10 times
  delay(durG); //keeps the green LED on after the advanced green turns off
  lightSwitch(green, yellow, durY); //turns the green LED off and the yellow one on
  lightSwitch(yellow, red, durR); //turns the yellow LED off and the red one on
  digitalWrite(red, 0); //turns the red LED off and then repeats the sequence
}

void lightSwitch(int off, int on, int dur) { //creates an function to turn the different lights off and on in one line
  digitalWrite(off, 0);
  digitalWrite(on, 1);
  delay(dur);
}
```

Project 2.2. Binary Button Echo

Purpose

The purpose of the Binary Button Echo is to learn how the 74HC595 shift register works through the use of eight buttons and a bar graph. This will help us when we do the next project where we have to breadboard the 595 chip and control 16 segment displays with it.

Reference

2019/20 ICS3U Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#BinaryButtonEcho

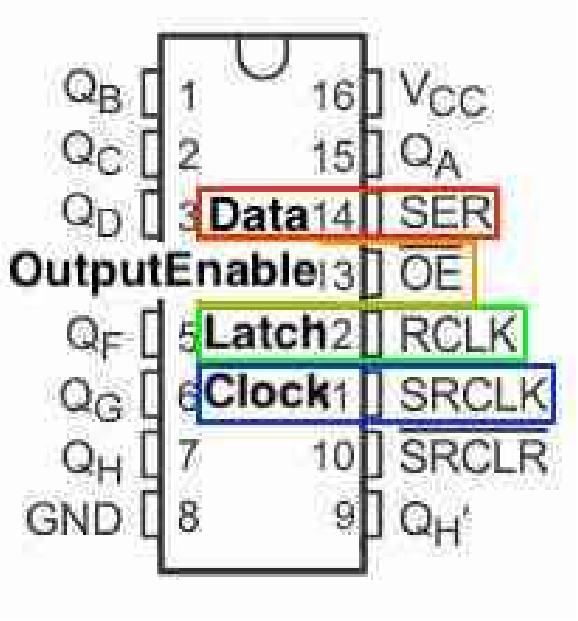
Procedure

This project uses the SN74HC595 shift register to control the LED bar graph. The bar graph's LEDs turn on base off of which button number we are holding down. If the first button is down the first LED will come on. If two buttons are down, those two respective LEDs will turn on at the bar graph.

Parts List	
Component	Quantity
Blue LED Bar Graph	1
330Ω resistor network	1
SN74HC595 chip	1
PCB	1
Right header pins	6
16 pin chip seat	1

The positive leads of the bar graph connect to the respective outputs from the 595 chip, and the negative leads from the bar graph each connect to a 330Ω resistor in a network that is then connected to ground. Eight buttons are breadboarded with pull up resistors. These buttons are then connected to which LEDs turn on and go into pins 9-2 on the Arduino board. In the code these are set as inputs and when a button is pressed the respective LED is turned on.

The shift register has 16 total pins. Two pins are the power pin and the ground pin. Eight pins are taken up for output, so we are able to have eight outputs from the chip. Four of the pins are the data pin, the clock pin, the latch pin, and the output enable pin. The data pin is what controls the outputs and tells which ones to be powered. The clock pin keeps a constant pulse. The latch pin when activated high will give an output that the data pin has set, until the latch pin is set back to low. The output enable pin the chip is grounded to show that we can output from this chip. Pin 10 is the clear pin. This pin has a line over the top of it which means it has to be grounded to function. For the code we are putting on we don't want this pin connected to ground because it would just clear everything. The last pin is pin 9, this is the pin that would be used if you want to connect two or more of these chips together to control more outputs.



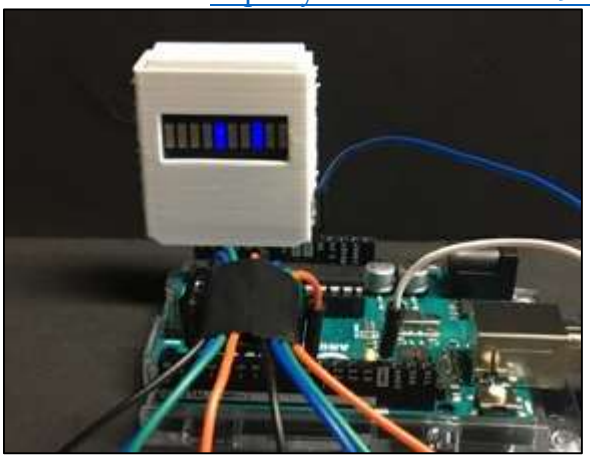
The SN74HC595 chip only has one input that controls outputs, this is the data pin. This makes it hard to control the individual outputs with multiple inputs. This meant that I had to use the `shiftOut` function for the code. The shift out function uses the data pin, clock pin, the way you want the outputs to turn on and a number which controls the output. My code uses the `MSBFIRST`, which means most significant bit first. So, when I press the first of my 8 buttons the LED on the far right of the bar graph will turn on. If I were to switch that with `LSBFIRST` or least significant bit first that would mean the LED 2 away from the left would turn on. This is because we have a 10 LED bar graph and we only have 8 outputs.

Reflection

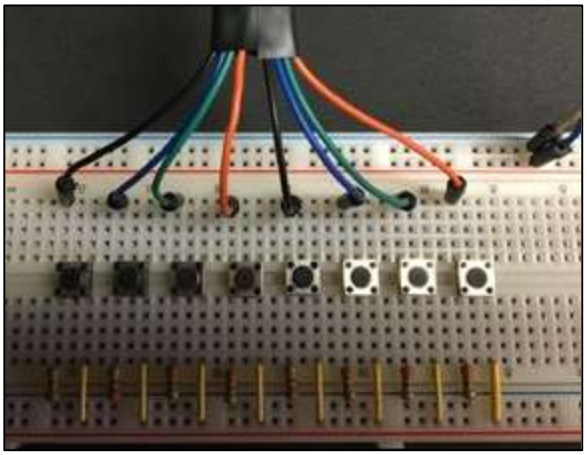
For this project I had a bit of trouble with the coding. I tried at first to use an `if-else` string but that was not at all effective. I then went back and looked over my code. I modified it to use a array for my buttons and changed the if to a for. I tried for a while to use the pow function but it wasn't working properly. I then changed the power function to a bitwise shift to the left and that fixed my problem. I polished it up a bit and wrote my comments, and then my computer died. I had to use my Dad's computer. I wrote out all of my code again and finished my DER. Overall this project was a good challenge for the coding and I enjoyed it a lot.

Media

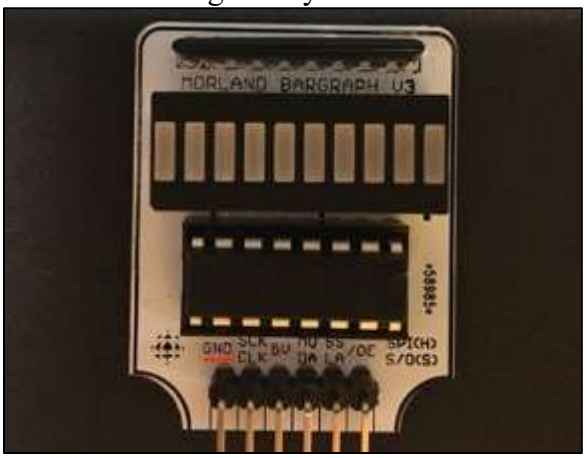
YouTube Video: https://youtu.be/AGhJJsKk9W0

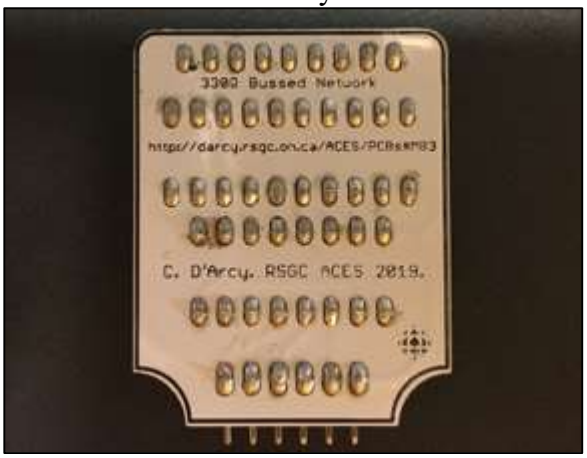


Working Binary Button Echo



Buttons for Binary Button Echo





Binary Button Echo PCB

Binary Button Echo Soldering

Code

```
// PROJECT   : Bianary Button Echo
// PURPOSE   : To learn how to code a shift register so we have a better
//             understanding of how it controls output.
// COURSE    : ICS3U
// AUTHOR    : J. Vretenar
// DATE      : October 19th, 2019
// STATUS    : Working
// REFERENCE: http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#BinaryButtonEcho

uint8_t gnd = A5;                                   //unsigned int for ground pin
uint8_t clk = A4;                                   //unsigned int for clock pin
uint8_t pwr = A3;                                   //unsigned int for power pin
uint8_t data = A2;                                  //unsigned int for data pin
uint8_t latch = A1;                                 //unsigned int for latch pin
uint8_t outputEnable = A0;                          //unsigned int for output enable
uint8_t buttonArray[] = {9, 8, 7, 6, 5, 4, 3, 2}; //unsigned int for buttons
uint8_t buttonState = 0;                            //unsigned int for button state
uint8_t binaryNumber = 0;                           //unsigned int for binary number


void setup() {
  Serial.begin(9600);               //starts serial monitor at 9600 baud
  pinMode(clk, OUTPUT);             //declares clock pin as output
  pinMode(pwr, OUTPUT);             //declares power pin as output
  pinMode(data, OUTPUT);            //declares data pin as output
  pinMode(latch, OUTPUT);           //declares latch pin as output
  digitalWrite(gnd, LOW);           //sets ground pin to low
  digitalWrite(pwr, HIGH);          //sets power pin to high
  digitalWrite(outputEnable, LOW);  //sets output enable to low
  for(uint8_t n = 0; n <=7; n++) {  //sets up the buttons as inputs
    pinMode(buttonArray[n], INPUT);
  }
}

void loop() {
  binaryNumber = 0;                               //LEDs are off
  for (uint8_t n = 0; n <= 7; n++) {
    if (digitalRead(buttonArray[n]) == HIGH) {  //reads buttons that are pressed
      binaryNumber |= (1<<n);                   //sets it as a binary number
    }
  }
  Serial.println(binaryNumber, BIN);            //prints binary to serial monitor
  digitalWrite(latch, LOW);                     //latch is set to low
  shiftOut(data, clk, MSBFIRST, binaryNumber);  //button signal is sent
  digitalWrite(latch, HIGH);                    //LEDs for buttons turn on
}
```

Project 2.3. PoV Word

Purpose

The Purpose of this project is to figure out how to use 2 shift registers to output a value on a dual 14-segment display. To do this it has to have a Persistence of Vision effect where the segments are constantly switching but fast enough so we see both of the displays on.

Reference

2019/20 ICS3U Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#PoV

Procedure

This project uses two SN74HC595 chips to display two letters on a dual 14-segment display. The different LEDs in the 14-segment displays turn on based off of a segment map that is on EEPROM. The persistence of vision is what makes both 14-segment displays turn on at once. This is controlled by a NPN transistor and a PNP transistor set up so they ground one segment when a pin is high, and ground the other when the pin is low.

Parts List	
Component	Quantity
Dual 14-segment display	1
3904 NPN transistor	1
3906 PNP transistor	
SN74HC595 chip	2
1kΩ resistors	3

The first thing that is set up are the 595 ICs. The 1st and 16th pin are set to V++ and ground respectively. The data pin, clock pin, and latch pin are all connected to the Arduino in different pins. The master reset is connected to V++ so it doesn't reset. The output enable pin is set to ground which enables output. The overflow pin of the first shift register goes into the data pin on the second shift register. The clock and latch pin from the first shift register connect to the clock and latch pin on the second one. The outputs on both shift registers are connected up so one through eight on the first shift register go to the segments A-H, and the first six outputs are connected on the second one to segments J-N, and P.

The transistors are set up so that the base pins are connected together and then a 1kΩ resistor is connected to the switch pin on the Arduino. The emitter pins on both transistors are connected to ground and the collector pins are a resisted high with 1kΩ resistors. The common cathode of the first 14-segment display is connected to the 3904 on the collector pin. The common cathode of the second 14-segment display is connected to the 3906 on the collector pin.

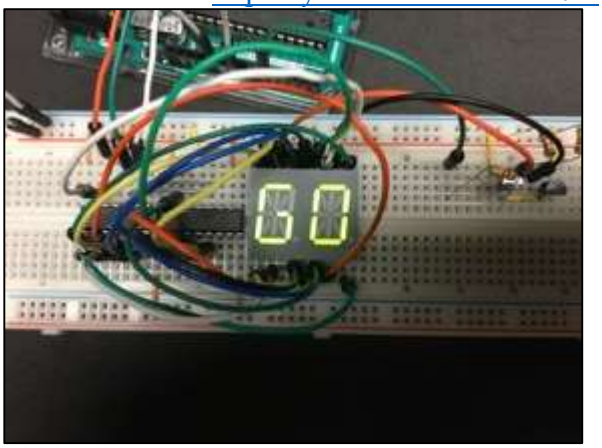
The code is split into two parts. One part writes the segment map in two different bytes per letter and then writes the segment map to EEPROM to be stored for later. The second part takes the segment map values from the EEPROM and uses the `shiftOut` function to show it on the dual 14-segment display. In between all the shifting that the IC is doing, it is changing the switch pin from HIGH to LOW and back again to create the persistence of vision effect that shows up on the 14-segment display for us as two letters.

Reflection

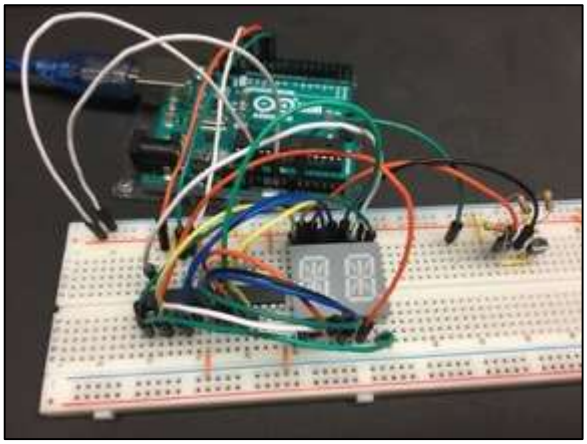
I found this project very hard. I was having a lot of trouble with what to do with the writing. At first, I found that my segment map was messed up and had to re write it. I than figured out that my wiring was messed up and re wired my dual 14-segment map. I also had some trouble with writing the code and making sure that the 14-segment actually had both displays on and not just one. I did some research about different functions on the Arduino website and found this one person using the `Serial.available` where it had a value also equal to false. I used this and added some extra things to allow me to only stop the loop when there was a new value inputted into the Arduino. I then had some trouble with how the baud was allowing for too much of a break and didn't allow me to put in a second value too fast or it would mess up. Switching my baud to 19200 fixed this problem for most of the time and made it easier to input values faster.

Media

YouTube Video: https://youtu.be/BnbNDa-e7fs



PoV Word displaying GO



PoV Word wiring

Code

EEPROM writing code

```
// Project: POVWord EEPROM
// Purpose: Writes a lookup table of 14-segment
//        :  Uppercase Letter Segment Maps to EEPROM
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2019 11 09
// Status : Working
// Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#PoV

#include <EEPROM.h>

uint8_t segmentMap14[] = {
  0b11101100, //A
  0b11110010, //B
  0b10011100, //C
  0b11110010, //D
  0b10011100, //E
  0b10001100, //F
  0b10111100, //G
  0b01101100, //H
```

```
  0b10010010, //I
  0b01111000, //J
  0b00001101, //K
  0b00011100, //L
  0b01101101, //M
  0b01101100, //N
  0b11111100, //O
  0b11001100, //P
  0b11111100, //Q
  0b11001100, //R
  0b10110100, //S
  0b10000010, //T
  0b01111100, //U
  0b00001101, //V
  0b01101100, //W
  0b00000001, //X
  0b00000001, //Y
  0b10010001, //Z
  0b10001000, //A2
  0b10100000, //B2
  0b00000000, //C2
  0b00100000, //D2
  0b10001000, //E2
  0b10001000, //F2
  0b10000000, //G2
  0b10001000, //H2
  0b00100000, //I2
  0b00000000, //J2
  0b01001000, //K2
  0b00000000, //L2
  0b00000100, //M2
  0b01000100, //N2
  0b00000000, //O2
  0b10001000, //P2
  0b01000000, //Q2
  0b11001000, //R2
  0b10001000, //S2
  0b00100000, //T2
  0b00000000, //U2
  0b00010000, //V2
  0b01010000, //W2
  0b01010100, //X2
  0b00100100, //Y2
  0b00010000, //Z2
};

uint8_t sizeMap = sizeof(segmentMap14);

void setup() {
  Serial.begin(9600);
  for (uint8_t i = 0; i <= sizeMap; i++)
    EEPROM.write('A' + i, segmentMap14[i]);
  Serial.println("EEPROM set up");
}

void loop() {
}
```

PoV Word code

```
// Project: POVWord
// Purpose: Reads a LUT of 14-segment Uppercase
//        :   letters segment maps on EEPROM and
//        :   displays it on a dual 14-segment display.
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2019 11 09
// Status : Working
// Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#PoV

#include <EEPROM.h>     //includes EEPROM library
uint8_t clockPin = 13;  //creates unsigned ints for
uint8_t dataPin = 11;   //clock, data, latch and switch
uint8_t latchPin = 10;
uint8_t switchPin = 14;
char buff[2];           //creates character buff

void setup() {
  Serial.begin(19200);        //starts Serial monitor at 19200 baud
  pinMode(clockPin, OUTPUT);  //  set clock, data, latch, and switch
  pinMode(dataPin, OUTPUT);   //  unsigned ints to OUTPUTs
  pinMode(latchPin, OUTPUT);
  pinMode(switchPin, OUTPUT);
  Serial.print("Input 2 letters uppercase or lowercase");
}

void loop() {
  boolean terminated = false; //terminated is false
  uint8_t ndx = 0;            //creates unsigned int index
  uint8_t recv = 0;           //creates unsigned int recive

  // read from keyboard
  while (Serial.available() && terminated == false) {
    recv = Serial.read();
    if (recv == '\n') {
      terminated = true;      // keep reading until the enter key is
pressed
    } else {
      // check for lower case
      if (recv >= 'a' && recv <= 'z')
        recv = recv - 32;     // make it upper case
      buff[ndx] = recv;
      ndx++;
      if (ndx == 2)
        ndx--;                // keeps writing to the last array item
    }
  }
  ndx = 0;
  terminated = false;

  //writes the latch pin LOW, shifts out two different bytes for the first
  //  14-segment display, latch pin is HIGH, switch pin is HIGH
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, EEPROM.read(buff[0] + 26));
  shiftOut(dataPin, clockPin, LSBFIRST, EEPROM.read(buff[0]));
  digitalWrite(latchPin, HIGH);
```

```
  digitalWrite(switchPin, HIGH);
  delay(1);

  //latch pin LOW, shifts out two different bytes for the 2nd 14-segment
  //  display, latch pin is HIGH, switch pin goes back to LOW
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, EEPROM.read(buff[1] + 26));
  shiftOut(dataPin, clockPin, LSBFIRST, EEPROM.read(buff[1]));
  digitalWrite(latchPin, HIGH);
  digitalWrite(switchPin, LOW);
  delay(1);
}
```

Project 2.4.1. Breadboard ATmega328P

Purpose

The purpose of the Breadboarded ATmega328P is to expand our knowledge of how the Arduino works. When it is breadboarded, we are able to see what pins we need, and how the outputs on the Arduino and extra features are different from the chip.

Reference

2019/20 ICS3U Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#standalone

Procedure

The main part is this project is the ATmega328P IC. This is identical to the IC that is in the Arduino. This IC has 28 pins. Pins 21, 20 and 7 have to all be connected to VCC. Pins 22 and 8 are connected to ground. Pin 1 is the reset pin on the Arduino. This is connected to a push button that when pressed will reset the code that is running on the IC. Pins 9 and 10 are connected together with the 16 MHz crystal. They are also both connected to ground through 20 pF capacitors.

Pins 2-6 and 11-19 are the digital output pins 0-13 on the Arduino. These pins are all used to output something from the IC.

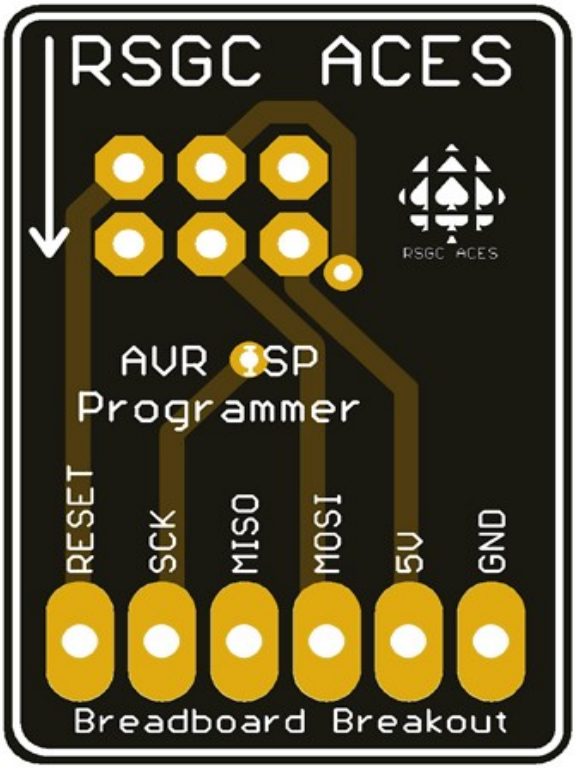
Pins 23-28 are the analog input pins A0-A6, but can also be pins 14-19.

Parts List	
Component	Quantity
ATmega328P chip	1
7 segment display	1
Red LEDs	4
5V voltage regulator	1
SN74HC595 chip	2
1 kΩ resistors	4
16 MHz crystal	1
0.1 uF capacitor	2
20 pF capacitor	2
10 kΩ resistor	1
Push button	1

The circuit that we create is controlled by the AVR programmer that is in our kit. Normally on the Arduino it would connect to six male header pins on the opposite side of the type-b usb connector. We used a small PCB that turns the same male header pins into 6 straight male pins. This is then plugged into the breadboard and connected to the pins on the ATmega328P

The 6 pins that we get are Reset, Clock, MISO, MOSI, 5V and Ground. The reset pin connects to the reset pin on the chip, this allows the programmer to reset the program that was currently running. The Clock pin is connected to pin 19 on the ATmega328P chip. This is also digital pin 13. This allows the programmer to set a clock for the program it is running. The MISO pin, which is Master In Slave Out, is connected to pin 18 on the ATmega328P. This controls the data going into the chip from our computer. The opposite is the Master Out Slave In or MOSI pin, This controls the data that is going from the chip to our computer. Both of these pins are the pins that allow our computer to communicate with the ATmega328P IC. The last two pins are 5V and

ground. They are connected to the rails of our breadboard and provide the whole circuit with a VCC and ground line.



To show how the breadboarded ATmega328P worked, there are two shift registers used to count up to 15 in binary and also count to 15 in hex. The binary is displayed through 4 red LEDs. The hex is displayed through a 7-segment display beside the red LEDs. The circuit is counting up every second until it hits 15, then it rolls back to zero and starts again.

The two shift registers are connected through VCC in pin 16 and ground on pin 8. The first shift register has data connected to pin 16 on the ATmega328P chip. The latch pin is connected to pin 15 and the clock pin is connected to pin 14. The data of the second shift register is connected to the overflow pin from the first shift register. The latch and clock pins are connected to the latch and clock pins of the first shift register. The master reset on both shift registers is connected to VCC and the output enable on both of them is connected to ground. The first shift register is connected up to the 7-segment display through the 8 output pins. The second shift register uses 4 output pins that are connected up to the 4 LEDs.

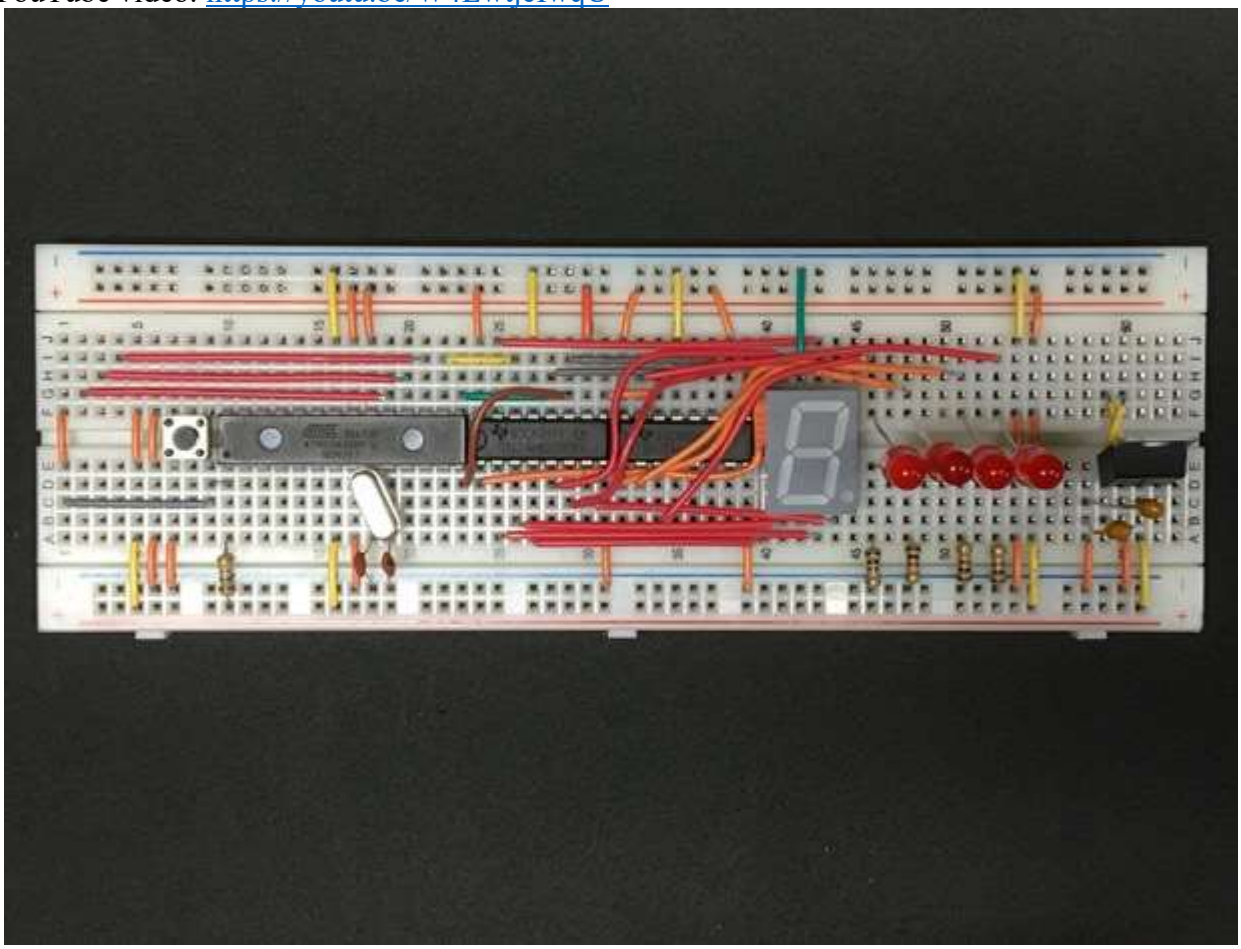
The code sets the latch pin low, it then shifts out two different values that are defined in an array earlier. The LED binary value gets shifted out first and then the value for the 7-segment display LEDs is shifted out next. The latch pin is then set high and a delay of 1000 milliseconds allows the number to stay on for 1 second before moving up to the next number.

Reflection

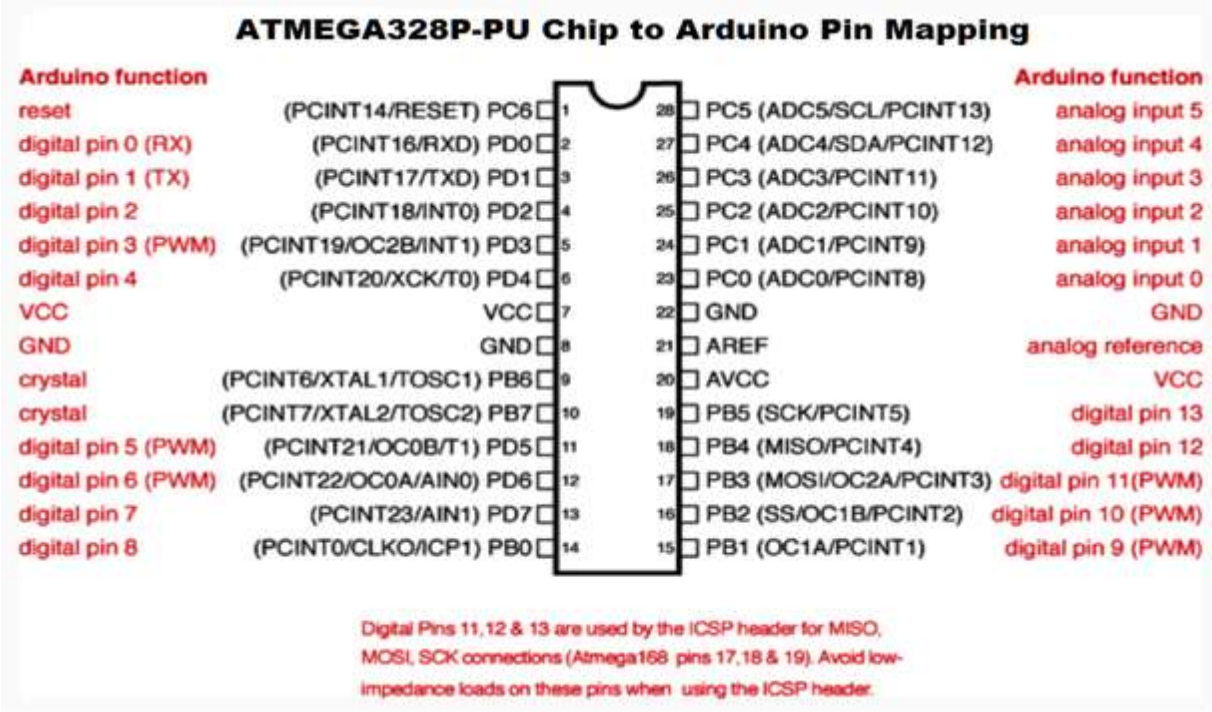
This was a pretty interesting project to do. I found getting the breadboarded ATmega328P wasn't too hard at first. To figure out what to do was the hard part. I chose to go with something that I could use my skills to create. I then went through all the parts in my kit and built the circuit. It didn't take me too long to wire up, the only problem I had was that my LEDs were in the reversed for counting so I just had to move some wires and then it worked. I really enjoyed this project and found it very informative about a general way of how the Arduino is wired up. I could have made my code shorter by using a chip from the counting circuit but I realized that a bit too late.

Media

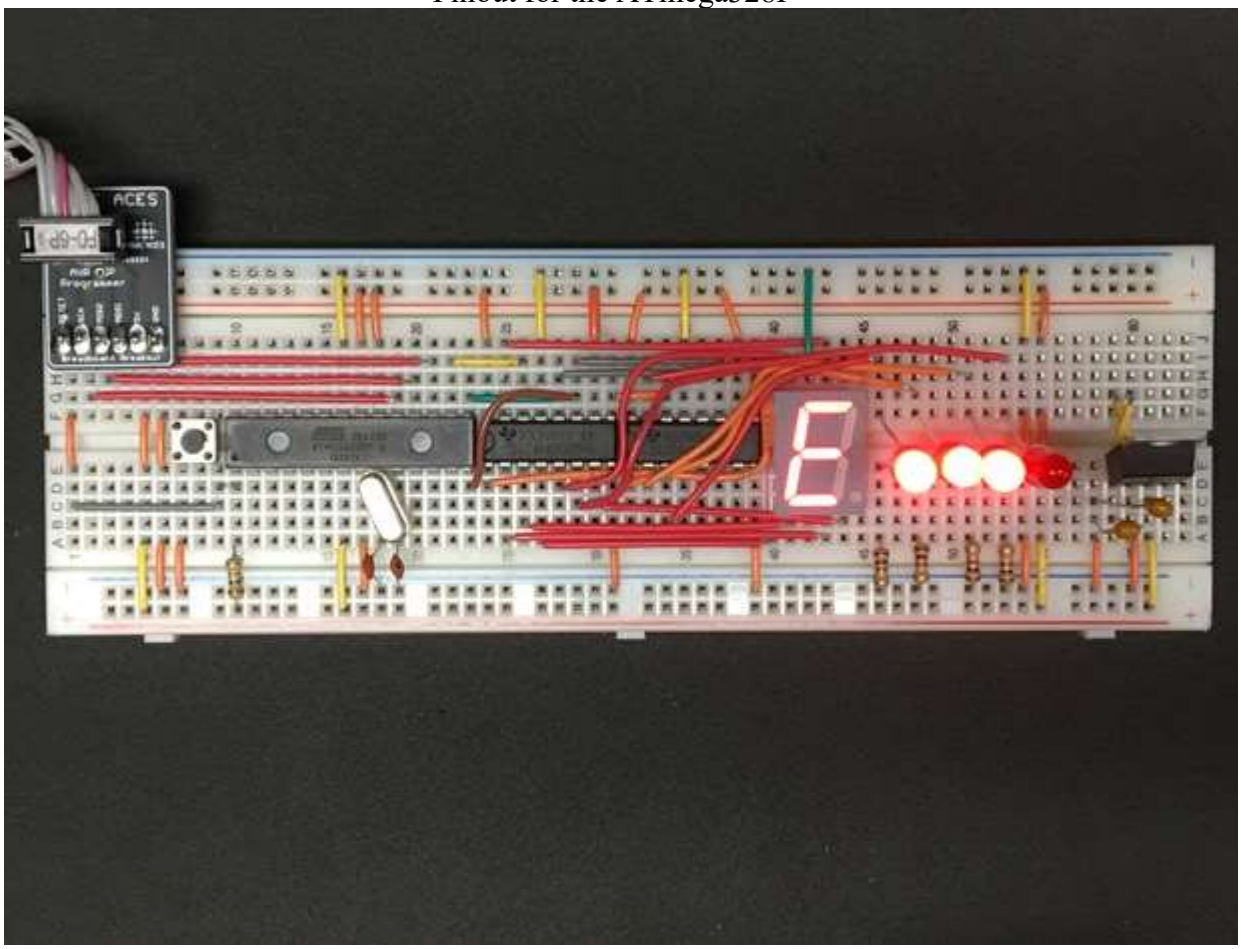
YouTube video: https://youtu.be/W4LwtjeIwqU



Wired Circuit



Pinout for the ATmega328P



Working circuit

Code

```
// Project: Breadboard ATmega328P counting up to 15
// Purpose: To breadboard an arduino and program something on it
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2019 11 23
// Status : Working
// Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#standalone

uint8_t mapping[] {
  0b11111100, //0
  0b01100000, //1
  0b11011010, //2
  0b11110010, //3
  0b01100110, //4
  0b10110110, //5
  0b10111110, //6
  0b11100000, //7
  0b11111110, //8
  0b11100110, //9
  0b11101110, //A
  0b00111110, //B
  0b10011100, //C
  0b01111010, //D
  0b10011110, //E
  0b10001110  //F
};

uint8_t dataPin = 10; //values for pins
uint8_t latchPin = 9;
uint8_t clockPin = 8;

void setup() {
  Serial.begin(9600);
  pinMode(dataPin, OUTPUT); //sets these values as outputs
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
}

void loop() {
  for (uint8_t i = 0; i <= 15; i++) {
    digitalWrite(latchPin, LOW);                          //latch low
    shiftOut(dataPin, clockPin, LSBFIRST, i);    //shift binary
    shiftOut(dataPin, clockPin, LSBFIRST, mapping[i+16]); //shift hex
    digitalWrite(latchPin, HIGH);                         //latch high
    delay(1000);                                          //wait one second
  }
}
```

Project 2.4.2. Altoids Arduino

Purpose

The purpose of the Altoids Arduino is to put the previous projects circuit into an Altioids tin. To put the circuit from the previous project into the Altoids tin, it requires a lot of planning and designing to get everything to look nice.

Reference

2019/20 ICS3U Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#AltoidsArduino

Procedure

Parts List	
Component	Quantity
28 pin chip seat	1
16 pin chip seat	2
1×5 female headers	2
1×6 female headers	1
Adafruit Perma Proto	1
Power jack	1

This project is the exact same project as last time, except it is an Altoids tin. The circuit needs to be wired onto a prema-proto board. There is a Altoids tin shaped board that is made by Adafruit. All the pieces from the previous project are soldered onto this board. Some connections had to be cut so there would be no shorting. The chip seats are used to replace where the ICs would go so the ICs are not soldered directly onto the perma-proto board. 1x5 headers are lined up and soldered onto the board so that they line up with a 7-segment display. The 1x6 header is used from programming the circuit through the MOSI, MISO, and Reset pins that were discussed in the last project.

The Altoids tin needs some modifications to fit everything that was in the previous project into it. There was a hole that needed to be cut out to fit the power jack, a hole needed to be cut out for the 7-segment display and the LEDs, and my name had to be on the outside of the tin. For the holes for the display and LEDs a Dremel with a disk head was used to cut out square holes that fit the row of LEDs and the 7-segment display through. For the power jack, a drill was used to drill a hole in the side that the power jack could fit through. My initials were then 3D printed and glued to the side of the Altoids tin.

Reflection

I had a lot of problems with this project. First off, I had to fit a full sized breadboard circuit into a half size perma-proto board. This meant that I had to use an Exacto blade and cut connections so everything could fit. I quickly learned after about an hour of soldering that one connection was still connected and ended up shorting the circuit, this meant that I had fried an ATmega328P. I then went ahead and got another ATmega328P. I figured out the problem with my circuit and fixed it. I continued to solder every part, checking each connection for the right reading. I then made my case look good. I took a Dremel at home and cut out a few holes for my 7-segment display and my LEDs. I then took a drill and cut out a hole for the power jack. The only problem was that my circuit wasn't working, I tested 4 different ATmega328Ps including the one on my Arduino. Every single one said that there was no signature detecting. So I tried to trouble shoot. I was testing the connections on my circuit for a good hour but it still would not work.

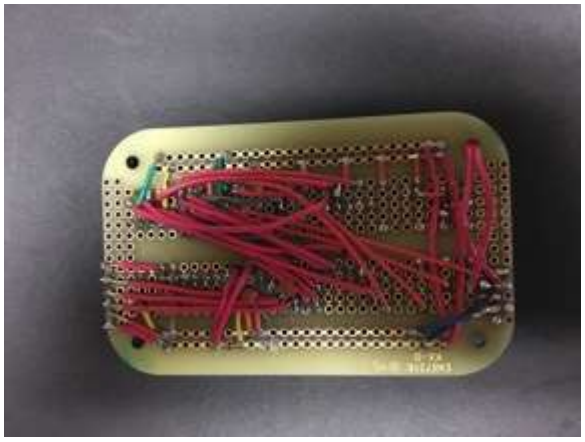
Media

YouTube video: https://youtu.be/NdlzzAH9kbM

Circuit in Altoids Tin with lid on

Circuit in Altoids tin with lid off

Initials on side of case



Underside of PCB

Project 2.5 MatrixMadeEZ

Purpose

The purpose of the MatrixMadeEZ was to gain understanding of the TPIC6C595 power shift register and how to use it with the SN74HC595 shift register to program a 8×8 LED matrix.

Reference

2019/20 ICS3U Engineering Tasks:
http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#MatrixMadeEZ

Procedure

This project uses a TPIC6C595 power shift register, and a SN74HC595 shift register to control a 8×8 LED matrix. These three parts are soldered onto a PCB with 1×6 right angle header pins that are used to program the shift registers.

Parts List	
Component	Quantity
MatrixMadeEZ PCB	1
8×8 LED matrix	1
SN74HC595 shift register	1
TPIC6C595 power shift register	1
1×6 right angle header pins	1

The SN74HC595 shift register controls the columns on the matrix and the TPIC6C595 shift register controls the rows. For this to work, the project requires persistence of vision. This is so if the matrix is trying to display a diagonal it will display the diagonal and not a square. This means that the `shiftOut` function has to be called twice to turn on the column through the SN74HC595 shift register and ground the rows through the TPIC6C595 power shift register. If these functions alternate back and for while using `bitShifting`, it will allow the matrix to display letters as well as animations.

To display a scrolling message, the letters have to be defined in EEPROM or earlier in the code through an `array`. The message has to have a buffer before the first letter to allow the first letter to move into the matrix. The code then has to `bitShift` the letter so it can display the next column. This gets repeated until the letter gets to the end, then the next letter does the same thing, except the buffer for the next letter is the last letter. Animations on the matrix work in a very similar way to the scrolling message, except instead of `bitShifting` the letters across, the animation `array` is `bitshifted` until the animation is complete.

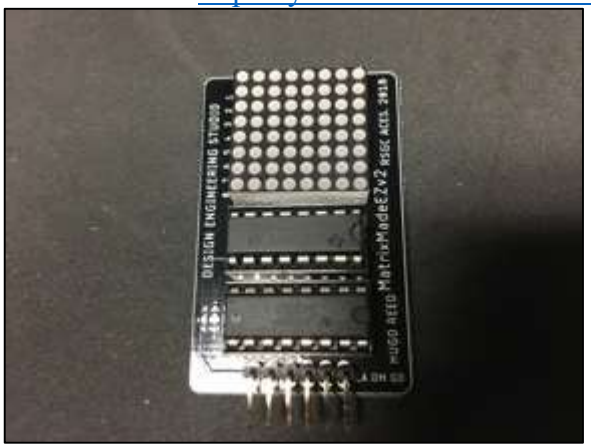
The code for this project required knowledge of how to do both animation and scrolling message coding. The matrix displays a firework going up, exploding, and then falling down. After the animation is done, the word "BOOM" scrolls across the matrix. This whole animation and scrolling message repeats. This code requires two `arrays`, one for the animation and one for the scrolling message. The animation is displayed one part at a time, `bitshifting` changes what is going to be the `shiftOut` numbering so it will display the next part. The code then jumps to the scrolling message, this is displayed with a buffer in front of the first letter so the "B" will scroll into view. Every time the two `shiftOut` functions are done, the letter gets `bitShifted` one to the left and displays the next part. The second letter gets `bitShifted` into view once the first part of the first letter is out of the matrix.

Reflection

This project was very fun to program. Coming up with an idea to display on the matrix allowed me to create an animation of a firework blowing up and then a scrolling message saying "BOOM". I ended up being able to include both EEPROM and `timerOne` in my code. I had some trouble with some of the code though. When I first started, I didn't include `timerOne` just to see how it would display. This gave me some issues with blinking animations and letters, but once I included `timerOne`, all the blinking went away. I also had some trouble writing and reading from EEPROM. When I first wrote to EEPROM, I forgot to call the function that I created to write EEPROM to a value in my code. I ended up fixing that problem, but then I ran into a problem where it was displaying the animation wrong. This was because I had written the "BOOM" `array` over the animation `array`. I fixed that problem and it worked perfectly. I really loved learning about the matrix and how it worked. This might make a fine addition to my medium ISP.

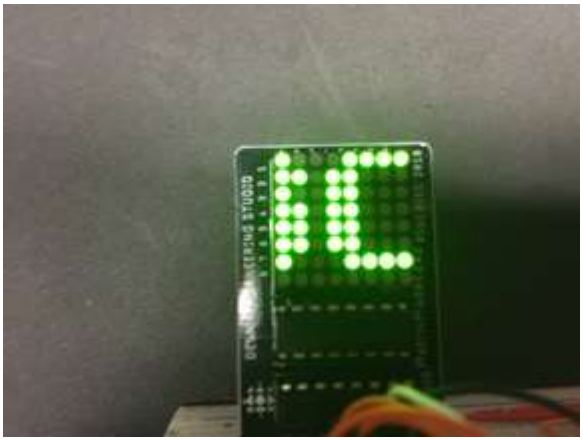
Media

YouTube video: https://youtu.be/KK-UU2cJc1w



Front of MatrixMadeEZ PCB

Back of MatrixMadeEZ PCB



Scrolling on MatrixMadeEZ

Code

```
// Project: MatrixMadeEZ EEPROM
// Purpose: To write EEPROM animations for MatrixMadeEZ
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2019 12 14
// Status : Not Working
// Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#MatrixMadeEZ

#include <EEPROM.h>
#define fireworkLocation 1
#define boomLocation 129

uint8_t firework[][8] {           //firework arrays
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 8},
  {0, 0, 0, 0, 0, 0, 8, 8},
  {0, 0, 0, 0, 0, 8, 8, 0},
  {0, 0, 0, 0, 8, 8, 0, 0},
  {0, 0, 0, 8, 8, 0, 0, 0},
  {0, 0, 8, 8, 0, 0, 0, 0},
  {0, 0, 8, 0, 0, 0, 0, 0},
  {0, 0, 8, 0, 0, 0, 0, 0},
  {0, 12, 24, 20, 0, 0, 0, 0},
  {10, 12, 48, 20, 34, 0, 0, 0},
  {19, 0, 32, 64, 34, 34, 0, 0},
  {16, 17, 1, 64, 64, 34, 34, 0},
  {0, 16, 17, 1, 64, 64, 34, 34},
  {0, 0, 0, 16, 1, 0, 64, 0},
  {0, 0, 0, 0, 0, 0, 0, 0},
};

uint16_t boom[][8] {                //boom arrays
  {126, 51, 62, 63, 51, 51, 126, 0},
  {62, 99, 99, 99, 99, 99, 62, 0},
  {99, 119, 127, 107, 99, 99, 99, 0}
};

void setup() {
  for (uint8_t i = 0; i < 16; i++) {  //writes firework to EEPROM
    for (uint8_t j = 0; j < 8; j++)
    {
      EEPROM.write(fireworkLocation + j + (i * 8), firework[i][j]);
    }
  }
  for (uint8_t k = 0; k < 3; k++) {   //writes boom to EEPROM
    for (uint8_t l = 0; l < 8; l++)
    {
      EEPROM.write(boomLocation + l + (k * 8), boom[k][l]);
    }
  }
}

void loop() {

}
```

```
// Project: MatrixMadeEZ
// Purpose: To use the EEPROM LUT to display on the MatrixMadeEZ
// Course : ICS3U
// Author : J. Vretenar
// Date   : 20?? ?? ??
// Status : Not Working
// Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#MatrixMadeEZ

#include <EEPROM.h>
#include <TimerOne.h>

#define fireworkLocation 1  //sets fireworkLocation to 1
#define boomLocation 129    //sets boomLoaction to 129
#define SPEED 70000         //sets speed
#define DMLEVEL 0           //sets dimming level
uint8_t dimmer = 3;         //dimmer pin to 3
uint8_t data = 11;          //data pin to 11
uint8_t latch = 12;         //latch pin to 12
uint8_t clk = 13;           //clock pin to 13
uint8_t i = 0;              //sets variables to 0 for later
uint8_t j = 0;
uint8_t m = 0;
bool fireworkComplete = false;  //bool value for loop
uint8_t letterCount = 0;    //sets letter count to 0
uint8_t firework[16][8];    //creates firework array value
uint16_t boom[3][8];        //creates boom array value

void setup() {
  pinMode(dimmer, OUTPUT);  //sets outputs
  pinMode(data, OUTPUT);
  pinMode(clk, OUTPUT);
  pinMode(latch, OUTPUT);
  Serial.begin(9600);
  Timer1.initialize(SPEED); //sets speed of timer
  Timer1.attachInterrupt(advanceFirework);  //sets interrupt for timer
  loadFirework();           //loads firework to the earlier value
  loadBoom();               //loads boom to the earlier value
}

void advanceFirework() { //advance firework animation function
  if (i == 15) {
    fireworkComplete = true;
    Timer1.attachInterrupt(advanceBoom);
  }
  else
    i++;
}

void advanceBoom() { //advance boom scrolling function
  if (j == 7) {
    letterCount++;
    j = 0;
  }
  else {
    j++;
  }
}
```

```
void loadFirework() { //load firework to value
  for (uint8_t w = 0; w < 16; w++) {
    for (uint8_t x = 0; x < 8; x++) {
      firework[w][x] = EEPROM.read(fireworkLocation + x + (w * 8));
      Serial.println(firework[w][x]);
    }
  }
}

void loadBoom() { //load boom to value
  for (uint8_t w = 0; w < 3; w++) {
    for (uint8_t x = 0; x < 8; x++) {
      boom[w][x] = EEPROM.read(boomLocation + x + (w * 8));
      Serial.println(firework[w][x]);
    }
  }
}

void loop() {
  if (!fireworkComplete) {  //firework animation shiftOut
    for (uint8_t n = 0; n <= 7; n++) {
      shiftOut(data, clk, MSBFIRST, firework[i][n]);
      shiftOut(data, clk, LSBFIRST, 1 << n);
      digitalWrite(latch, LOW);
      digitalWrite(latch, HIGH);
    }
  }
  else
  {
    if (letterCount == 0) { //B scrolling shiftOut
      for (uint8_t m = 0; m <= 7; m++) {
        digitalWrite(latch, LOW);
        shiftOut(data, clk, MSBFIRST, boom[0][m] >> 8 - j);
        shiftOut(data, clk, LSBFIRST, 1 << m);
        digitalWrite(latch, HIGH);
      }
    }
    if (letterCount == 1) { //O scrolling shiftOut
      for (uint8_t m = 0; m <= 7; m++) {
        digitalWrite(latch, LOW);
        shiftOut(data, clk, MSBFIRST, (boom[0][m] << j) + (boom[1][m] >> 8 - j));
        shiftOut(data, clk, LSBFIRST, 1 << m);
        digitalWrite(latch, HIGH);
      }
    }
    if (letterCount == 2) { //O2 scrolling shiftOut
      for (uint8_t m = 0; m <= 7; m++) {
        shiftOut(data, clk, MSBFIRST, (boom[1][m] << j) + (boom[1][m] >> 8 - j));
        shiftOut(data, clk, LSBFIRST, 1 << m);
        digitalWrite(latch, LOW);
        digitalWrite(latch, HIGH);
      }
    }
    if (letterCount == 3) { //M scrolling shiftOut
      for (uint8_t m = 0; m <= 7; m++) {
        shiftOut(data, clk, MSBFIRST, (boom[1][m] << j) + (boom[2][m] >> 8 - j));
        shiftOut(data, clk, LSBFIRST, 1 << m);
        digitalWrite(latch, LOW);
        digitalWrite(latch, HIGH);
      }
    }
```

```
    if (letterCount == 4) { //end scrolling shiftOut
       for (uint8_t m = 0; m <= 7; m++) {
         shiftOut(data, clk, MSBFIRST, boom[2][m] << j + 1);
         shiftOut(data, clk, LSBFIRST, 1 << m);
         digitalWrite(latch, LOW);
         digitalWrite(latch, HIGH);
       }
     }
     if (letterCount == 5)
     {
       Timer1.detachInterrupt(); //restart animation
       delay(500);
       letterCount = 0;
       j = 0;
       i = 0;
       fireworkComplete = false;
       Timer1.attachInterrupt(advanceFirework);
     }
   }
 }
```

Project 2.6. Legacy PCB/Appliance

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#PCBfiles

Inspiration

The PCB is going to be used for my ISP which is a CN Tower message board. The inspiration for this device was at the Ultimaker training session when Mr. D'Arcy suggested to make a CN Tower. The PCB allows me to control LEDs on the CN Tower with an ATtiny84.

Procedure

To create the PCB, there is a program called EAGLE that sets up and builds PCBs. This project required planning done before hand so the parts needed are added to EAGLE. When EAGLE is opened, there is a projects folder that has all previous projects in there. The first step to create a PCB is to make a new project, and then make a schematic. When that schematic is opened, there should be a blank white canvas ready to have parts put in. The next step is to add all of the parts needed for the PCB to the schematic page. These parts should be moved so that they are in the general area of where they go on the PCB. Once they are setup near each other the leads of parts must be connected. There are two ways of connecting parts. There is a line tool that can be used to draw a green line between the leads that should be connected. This will connect these two leads together. The other way is to use the junction tool, these junctions can be placed on the leads that are to be connected. Once they are placed there is a info tool that can be used to rename the junction, if these junctions are named the same, they will be connected to each other. Once everything is wired up correctly, there is a button in the top left that will turn the schematic into a basic PCB. When the button is pressed, there will be a black rectangle with gold lines around it and the parts for the PCB should be on the left of it. These parts should then be dragged onto the PCB board and put in the general area near connections. These parts should be moved and possibly rotated so that the wires will be easy to route. There are two different ways to route the wires on the PCB, the wires can be manully routed or they can be auto routed. The auto router is really easy to use and gives multiple options for routing the wires, the only problem is the design rule check for some PCB manufactuers can lead to errors with some spillage. To avoid this problem, before autorouting at the top under the tools tab, the design rules check (DRC) tab will allow you to add a set of design rules that are specific to what you are doing. For this project and manufacturer, we are using a thicker wire line and a bigger clearance between wires and soldering pads. This makes sure we do not have any bad connections that could destroy parts or mess up the final product. Once all the wires are routed, the finishing touches can be added such as a name, logo, holes for mounting, or rounded corners. The name and logo have to be changed to layer 112 named `tsilk`, if this step is not done, they will not show up on the top of the board.

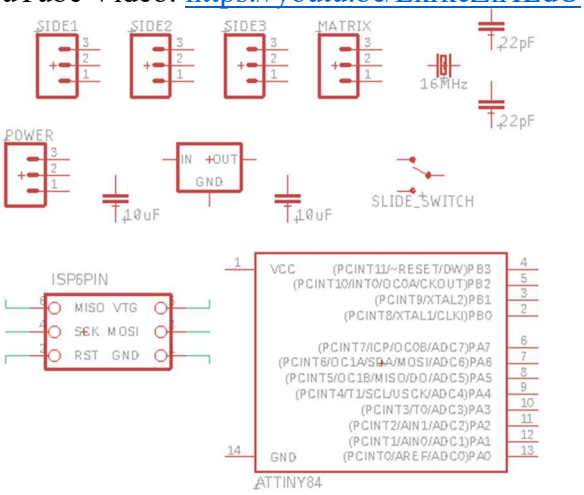
Parts List	
Component	Quantity
1×3 female header	2
1×3 male header	4
ATtiny84	1
5V voltage regulator	1
10uF capacitor	2
22uF capacitor	2
16 MHz crystal	1

Procedure Final

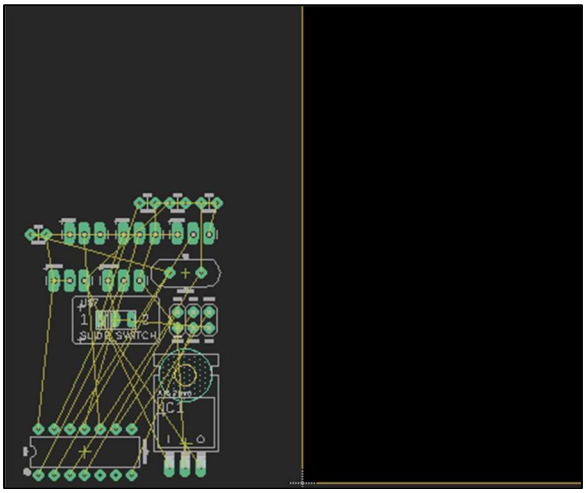
The PCB arrived a couple weeks after the request was sent in. Once it arrived, the headers, chipseat, voltage regulator, crystal, and capacitors were soldered on. I used the dimensions of the PCB to design a hole for the PCB to sit in for my CN Tower. To program the ATtiny84, it requires the use of an ISP header. On the Arduino IDE, another board has to be downloaded that has the required bootloader and settings for the ATtiny84. Once these are installed, the code can be uploaded through the programmer.

Media

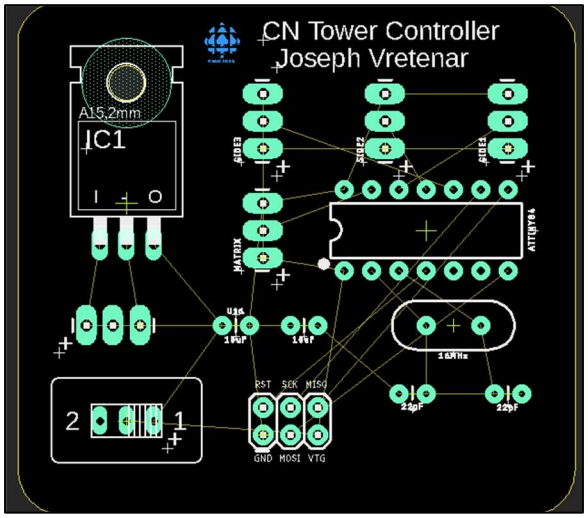
YouTube Video: https://youtu.be/LkrkeZiXLdU



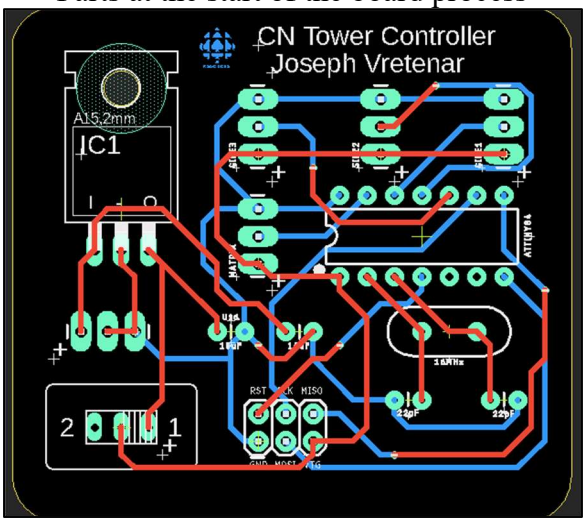
Parts laid out for schematic



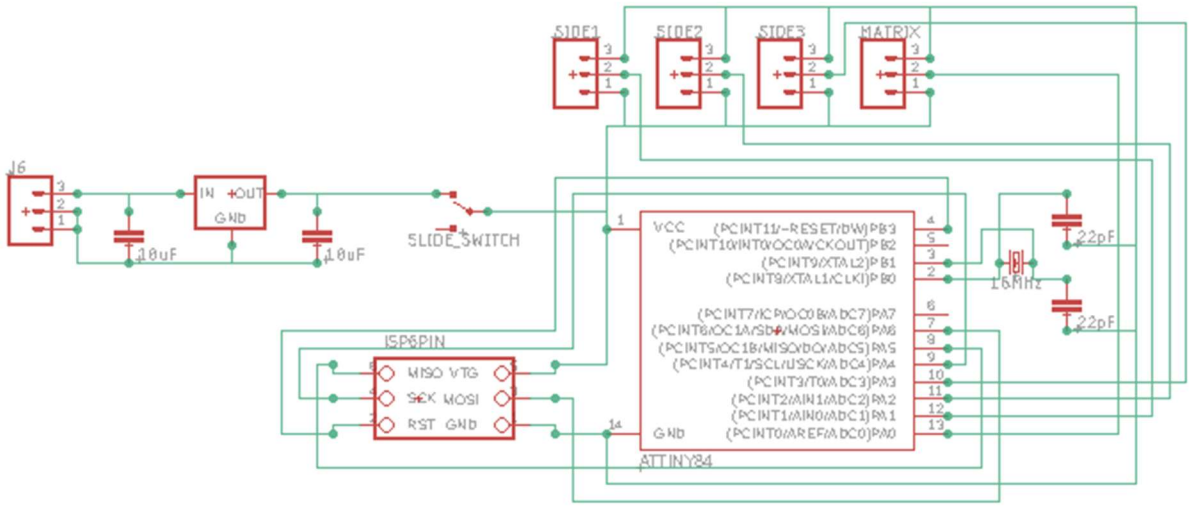
Parts at the start of the board process



Parts laid out on the board

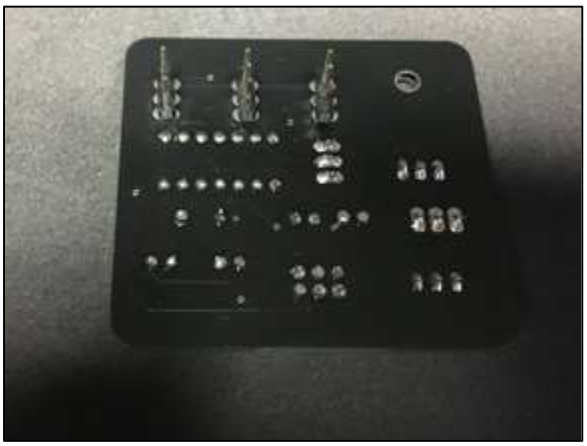


Parts all connected with naming on the board.

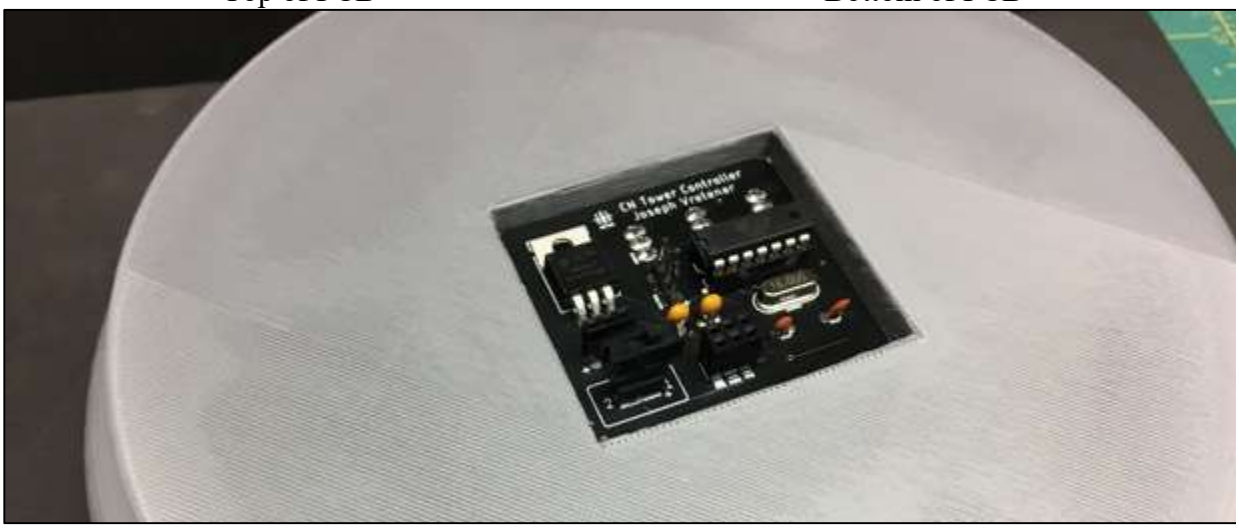


Parts all connected in schematic view

Top of PCB



Bottom of PCB



Case for PCB

Reflection

This project was a good learning process. It took a lot of time for me to think about what I should do for the PCB and how it will be used. In the end I went with a PCB design for my Medium ISP. I found that EAGLE was very easy to use and this helped me in the process. The only problem was that I can't test if my PCB works because the LEDs haven't arrived yet. I think that EAGLE will be a very helpful tool to use through out the hardware course and I will probably use it for my Long ISP aswell.

When I got my PCB, I was pretty confident that it was going to work. Once I got the ATtiny84 board downloaded, I got to work on programming the circuit. There was a problem with what LEDs I am trying to use. When I tried to upload code to the ATtiny84 that would program the matrix, it would give a long strand of error messages where it does not recognize any of the declared information in one of the libraries I am using. It ended up being a problem with the `Adafruit_GFX` library so the matrix that I am going to use will be programmed with an ATtiny85 while the LED strips can still be programmed through the ATtiny84. Though this is a big problem in my circuit, all of my connections and routes work properly.

Project 2.7. Medium ISP

Purpose

The purpose of this ISP was to dive deeper into something that we are interested in. I chose to design a CN tower and add a scrolling message board and elevator lights.

Reference

NeoPixel Library: https://github.com/adafruit/Adafruit_NeoPixel
NeoMatrix Library: https://github.com/adafruit/Adafruit_NeoMatrix

Procedure

To start this project, I had to make a design for the CN Tower. I started by doing some research for different dimensions. I could not find any specific dimensions while researching so I decided to base my design off of pictures.

Parts List	
Component	Quantity
Arduino Nano	2
Wire in m	10
NeoPixel Matrix	1
NeoPixel 60 LED strips	3
Spools of Grey Filament	4
Power Switch	1
Power Jack	1
5V AC to DC power plug	1

I used Fusion 360 to design my CN Tower. I started designing the tall base. I had to find a couple of photos and designed it with ratios. I planned to make the CN Tower 2 meters and 8 centimeters. Once I finished designing one side of the base, I had to design another side that would have a power switch and a power jack. Once that was done, I started to design the restaurant. I had to come up with a design that would allow the matrix to fit on it and the wires to go through. I ended up cutting part of the restaurant out so the matrix would be flat on the restaurant. Once the restaurant was finished. I had to design the top part of the CN Tower. This part was by far the easiest piece to design. It was just a couple of cylinders put together with cone shapes that combine them.

Once all the parts were designed, it was time for printing. I started by printing the restaurant. These parts printed very nicely and had no problems. After these, I printed the 3 base sides. This was were the problems started, it was a 31 hour print per side and that allows for a lot of error. The first side that I printed ended up failing pretty early. The problem with this was I was now out the filament that I was using. I went ahead and purchased some more filament, it was a different color put I feel like it looks more like concreate. I started printing the side again and it printed very nicely. I went ahead to print the second part and in the last hour or so it failed because of a filament tangle. When I reprinted this part, it worked out really well and I also printed the 3rd side. The final part to print was the top. I had to print this in a couple of different parts as I didn't have a full spool of filament left.

All the parts were now printed and had to be assembled. The CN Tower had to be printed in 26 different parts so it was a lot of assembly. I used super glue on all of the side pieces and the top. I ended up using epoxy to put all the sides together. I added velcro to the top of the sides and the bottom of the restaurant so if I wanted to adjust some of the wires then I can do that easily. In the restaurant, I added a little platform that raises the two Nanos to program the lights.

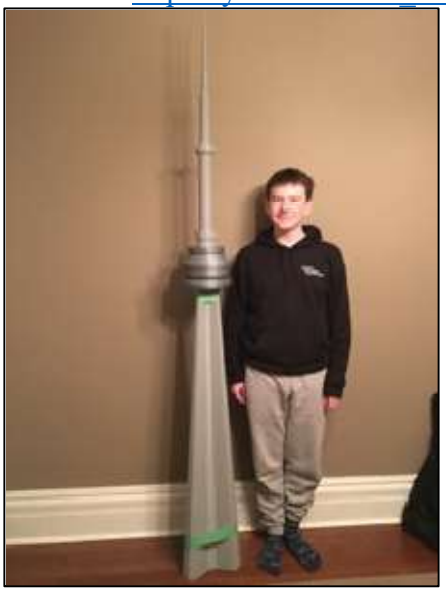
I put the LED strips as elevators on and put the matrix on. Once these were all on, I programed the strips and matrix. I started with some basic test examples that were provided with the libraries that I needed. The programming had to be done on two different Nanos because if I did it on one, it would not scroll the message at the same time as the LEDs. Once I made sure that the LEDs and matrix worked with some code I started to write my own. Since these libraries had some pretty useful bits added into them I used them to write my code. I started by programming the LEDs. I figured out that you can't actually control multiple different strips on one Nano, so I put them all together connecting them in the CN Tower. A problem that happened was that I wired the LED strips up backwards so the original strand test code would make the LEDs travel down. This was an easy fix as I just started at the end and made it subtract every time. The main thing with programming them was getting them to all go up at the same time. If I ran the example code that I modified, it would just do one side at a time. I had to end up creating code that uses the total number of LEDs divided by 3 and then multiplied by 1, 2, and 3 to start at the end of each strip. This allowed the LEDs to scroll up all at once. The next part was the matrix. I loaded up one of the examples that scrolls "howdy" across the matrix. When I uploaded the code, it was set up for a 5 by 8 matrix. I figured out this problem and adjusted the width, I ran the code again. This time it was doing some weird thing with the letters after every column it would flip. This happened to be a problem with the initialization. My matrix was wired up in a Zig-Zag pattern, but the way the code was programmed was in the Column pattern. I changed this and uploaded the code again, this time it worked. When I tried to change the "howdy" to a longer string, it ended up not showing the whole string. In the code, they had a buffer for each letter and I found this and changed it. It seemed to be that the buffer was 6 for every character and then an extra 6 to allow the final letter to scroll.

Reflection

This project was probably a bit too ambitious of a project but I somehow managed to pull it off. I started it really early as I thought it would take a long time. In total, it amounted to about 15 hours of 3D design, 15 hours of assembly, 200 hours of printing, over 1 kilometer of filament, and about $250 dollars. There were many little struggles along the way and many little changes that if I were to do it all again I would change. Some things are as little as changing the design to have some holes or reversing the LED strips.

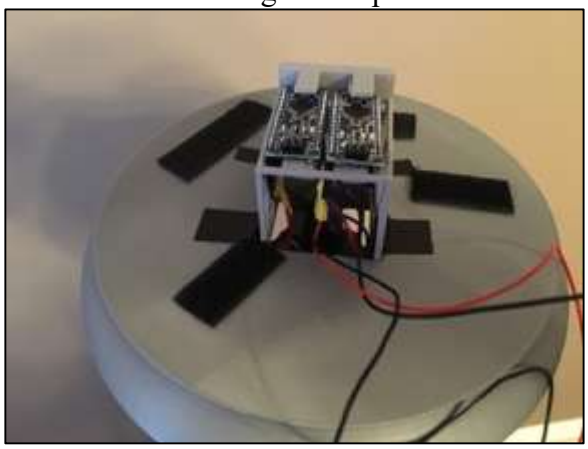
Media

YouTube Video: https://youtu.be/IskZ_uK-Kw8



CN Tower partially assembled with me next to it for Height Comparrison

CN Tower fully assembled with lights off



Arduino in the restaurant

CN Tower fully assembled with lights on

Code

```
// Project: CNTowerLEDStrips
// Purpose: To make LED strips scroll up on my Medium ISP
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2020 03 29
// Status : Working
// Reference: https://github.com/adafruit/Adafruit_NeoPixel

#include <Adafruit_NeoPixel.h>

#define PIN 6         //pin on the Nano that is being used
#define LED_COUNT 180 //Total number of LEDS on the CN Tower

Adafruit_NeoPixel strip(LED_COUNT, PIN, NEO_GRB + NEO_KHZ800);  //initialize the
                                                                //strip

#define LEDS (strip.numPixels()/3)  //Total LEDS on one side

void setup() {
  strip.begin();           //Start the strip and have it ready
  strip.show();            //Show what has been loaded on the strip
  strip.setBrightness(40);  //Set the brightness of the strip
}

void towerTwoColors(uint32_t color, uint32_t color2, uint16_t wait) {
  for (uint8_t i = LEDS; i != 0; i--) {              //Scrolls up on the LEDS from
    strip.setPixelColor(i - 1, color);               //one color to another
    strip.setPixelColor((i * 2 - 1), color);
    strip.setPixelColor((i * 3 - 1), color);
    strip.show();
    delay(wait);
  }
  for (uint8_t j = LEDS; j != 0; j--) {
    strip.setPixelColor(j - 1, color2);
    strip.setPixelColor((j * 2 - 1), color2);
    strip.setPixelColor((j * 3 - 1), color2);
    strip.show();
    delay(wait);
  }
}

void towerThreeColors(uint32_t color, uint32_t color2, uint32_t color3, uint16_t
wait) {
  for (uint8_t i = LEDS; i > 0; i--) {           //Scrolls up on the LEDs from
    strip.setPixelColor(i - 1, color);           //the first color to the
    strip.setPixelColor((i * 2 - 1), color);     //second color to the third color
    strip.setPixelColor((i * 3 - 1), color);
    strip.show();                              //Shows what is programmed before
    delay(wait);                               //The wait between the LEDs scrolling
  }                                            //lower is faster, higher is slower
  for (uint8_t j = LEDS; j > 0; j--) {
    strip.setPixelColor(j - 1, color2);
    strip.setPixelColor((j * 2 - 1), color2);
    strip.setPixelColor((j * 3 - 1), color2);
    strip.show();
    delay(wait);
  }
```

```
 for (uint8_t k = LEDS; k > 0; k--) {
    strip.setPixelColor(k - 1, color3);
    strip.setPixelColor((k * 2 - 1), color3);
    strip.setPixelColor((k * 3 - 1), color3);
    strip.show();
    delay(wait);
  }
}

void loop() {
  towerThreeColors(strip.Color(255, 0, 0), strip.Color(0, 255, 0), strip.Color(0,
  0, 255), 50);
  towerTwoColors(strip.Color(255, 247, 0), sript.Color(212, 0, 255), 50);
}
```

```
// Project: CNTowerMatrix
// Purpose: To scroll different messages on the CNTower
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2020 03 29
// Status : Working
// Reference: https://github.com/adafruit/Adafruit_NeoMatrix

#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>

#define PIN 6   //pin in use

//This is how the Matrix is initialized, the important parts to this
//are the things in the brackets, it starts by going with the width,
//then the height, then the pin in use. The next part determines what
//type of matrix you are using and what orientation it is in.
//For this matrix, it is scrolling from left to right so the matrix has
//to be initialized in the top left as the right point. The next part of
//this is how the matrix got wired up, there are a couple of different
//ways to do this as it can be by row or by column and then the LEDs
//can be in different orintations, the matrix that I got was wired up in
//the zigzag. GRB defines what type of product it is as the other ones
//are GRBW and RGB but they are for different products. The KHZ800 also
//is used for these WS2812b LED strips.
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(32, 8, PIN,
                            NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
                            NEO_MATRIX_COLUMNS + NEO_MATRIX_ZIGZAG,
                            NEO_GRB            + NEO_KHZ800);

const uint16_t colors[] = {
  matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0, 255)
};

void setup() {
  matrix.begin();
  matrix.setTextWrap(false);
  matrix.setBrightness(40);
  matrix.setTextColor(colors[0]);
}
```

```
uint8_t x = matrix.width();
uint8_t pass = 0;

void loop() {
  matrix.fillScreen(0);
  matrix.setCursor(x, 0);
  matrix.print(F("RSGC ACES 2020"));
  if (--x < -90) {
    x = matrix.width();
    if (++pass >= 3) pass = 0;
    matrix.setTextColor(colors[pass]);
  }
  matrix.show();
  delay(80);
}
```

Project 2.8. DC Motor Tachometry

Purpose

The purpose of this project is to test the rpm of our DC hobby motor. It also is a way to start with communication by working with IR.

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#tach

Procedure

Parts List	
Component	Quantity
DC Motor	1
10kΩ potentiometer	1
1MΩ resistor	1
220Ω resistor	1
10kΩ resistor	1
IRF520N Mosfet	1
IR Emitter	1
IR Receiver	1

The first thing to wire up in this project is the IR receiver and emmiter. The emmiter pins are connected straight to v++ and gnd. There is a 220Ω resistor between the ground lead on the emmiter and the ground rail. The receiver LED has the positive lead connected to the ground rail with a 10kΩ resistor between. The negative lead of the receiver connects to the positive rail. There is a wire connecting the positive lead of the receiver LED to pin 2 of the Arduino. This is used to take the information of the sensor. If there is nothing blocking the sensor, then it will read a 1, but when there is something blocking it, it will read 0. In our code we will use this to read the rpm. The code will read the falling edge of pin two and add up the count of the rpm.

To control the motor, we are using a potentiometer as a voltage divider that leads into a mosfet. The mosfet has the gate pin going to ground through a 1MΩ resistor and going to the middle lead on the potentiometer. The drain pin on the mosfet goes to the negative lead on the motor and the source pin goes to the ground rail. When we turn the potentiometer it will either turn up the speed of the motor or turn it down.

The code will use a `interrupt` to count the rpm whenever the beam is broken this uses a `falling` edge. Every second, it will give a reading for the rpm. This reading will be 8 times as fast as it should because we have 8 teeth on the encoder. To counter this, we `bitshift` the number by 3 because 8 is 2 to the power of 3. We then multiply this by 60 to allow us to figure out the rpm of the motor.

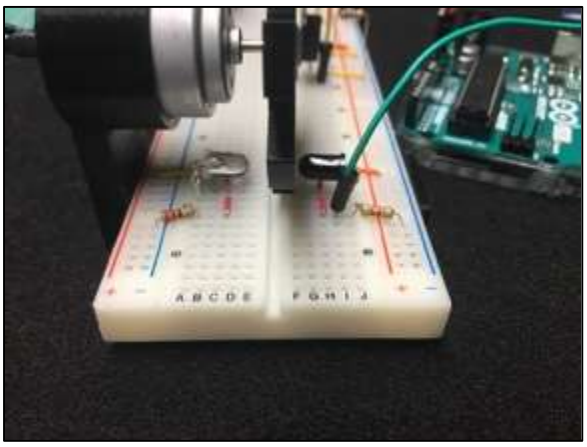
Reflection

I am very happy to be back working on another hardware project. This project was very fun and and made me interested in motors. I enjoyed making the 3D models for this project and it seemed to have worked pretty well. I hope this situation is going to be over soon so we can get back to the original hardware classes in the DES.

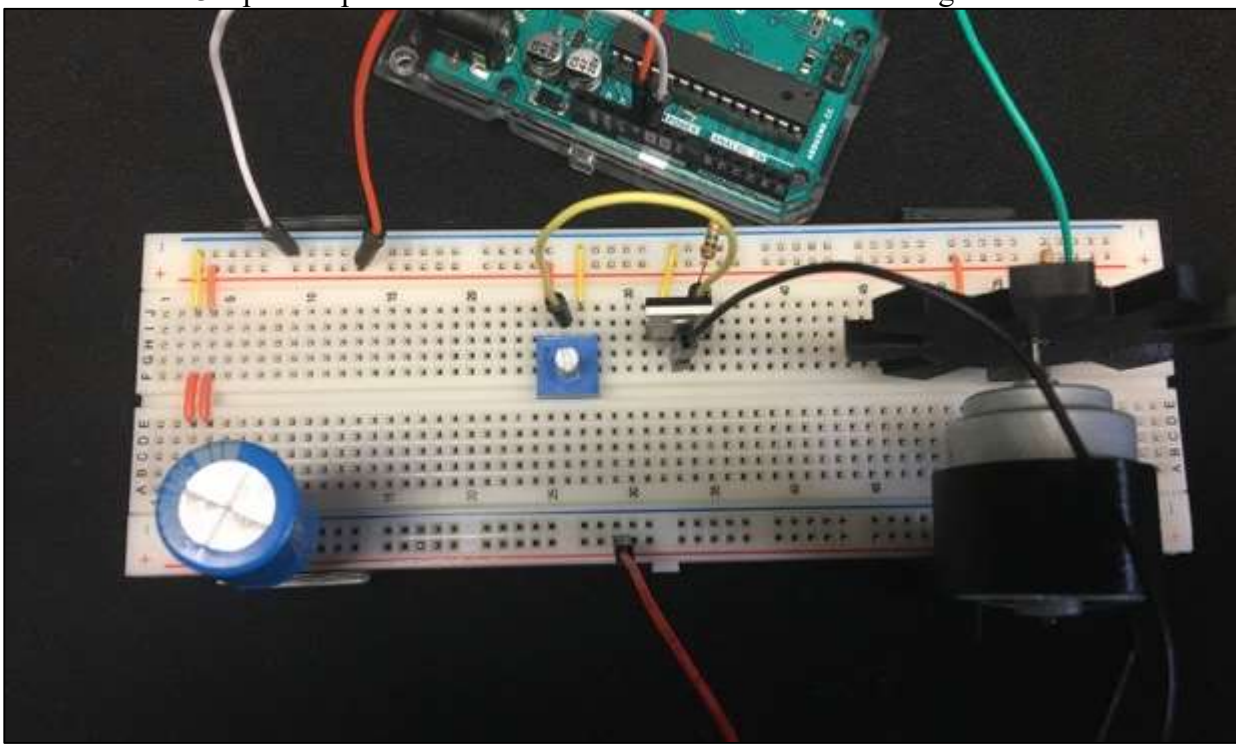
Media

YouTube Video: https://youtu.be/eL3VnPQ83Lo

3D printed parts



Encoder breaking the IR beam



Full Circuit

Code

```
// Project: DC Motor Tachometry
// Purpose: To figure out the RPM of our Motor
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2020 04 11
// Status : Working

#include <TimerOne.h> //TimerOne library

uint16_t count;         //count of rotations
uint16_t copy;          //copy of count to reset quickly
uint16_t RPM;           //final rpm reading
uint16_t interrupt = 0; //sets the if statment to 0

#define TEETH 3         //2^3 teeth for bitshifting
#define TIME 1000000    //timer interval between each printing

void setup() {
  Serial.begin(9600);
  Timer1.initialize(TIME);          //sets interrupt
  Timer1.attachInterrupt(printIt);  //sets the interrupt function
  attachInterrupt(digitalPinToInterrupt(2), addRPM, FALLING);
}                                   //sets pin 2 to read the falling edge

void addRPM() {
  count++;        //adds up each tooth that passes
}

void printIt() {
  interrupt = 1;  //sets interrupt to 1 so it wil print
}

void loop() {
  if (interrupt) {                          //if statement for interrupt
  copy = count;                             //copy the count to reset faster
  count = 0;                                //reset count to not miss any rpm
  interrupt = 0;                            //sets the interrupt timer to 0
  RPM = (copy >> TEETH) * 60;               //calculates the actual RPM
  Serial.println("RPM = " + String(RPM)); //prints what the RPM is
  }
}
```

Project 2.9. DC Motor Control

Media

Youtube Video: https://youtu.be/qaxaw16fo8Y

Project 2.10. Time-Sensitive Mechanics

Purpose

The purpose this project is to help us delve deeper into the understanding of motors, displays, and clocks. This project is using a real time clock(RTC), a servo, stepper, or DC motor, and a display such as a LCD or a row of 7-segment displays.

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#TimeSensitiveMechanics

Procedure

This project is going to be a shaker table that measure how long the table is shaking. This will be used for another class later to help explain building resonance. For this project, we are using the DC motor to make the table shake, a LCD will be displaying the time that the table is shaking. The RTC will tell us how long the table was shaking.

Parts List	
Component	Quantity
DC Motor	1
LCD display	1
DS1307RTC	1
Ball Bearing	2

The first thing for this project is a table that will move horizontally with an input that is rotational. The way this is done is through a car engine piston setup where there is an offset point that is on the motor that connects to the end of the table. The table has rows under it where a mushroom shaped part can hold it straight. This required a couple of 3D models that were printed and 2 ball bearings.

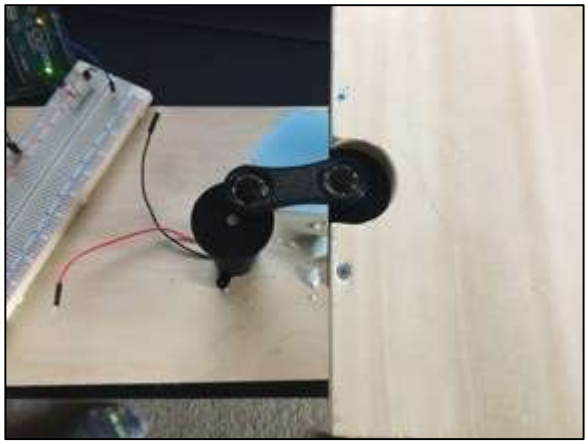
Once the table is moving horizontally, there needed to be something to turn the motor on and off. The solution for this was the RTC. It allowed the motor to turn on and off when it reached a certain time. These times are then recorded onto the LCD screen and it will change once it reached the end.

Reflection

This project was a very hard project for me, I ran into multiple problems with my design and was very frustrating. I had some trouble with my first gear design and ended up having to make it smaller for it to work. Another problem was the struts underneath the table that moved it forwards and backwards. Those were too circular in my first design and made it a bit too rotational in the movement. I still had fun with this project and was a good learning experience for me.

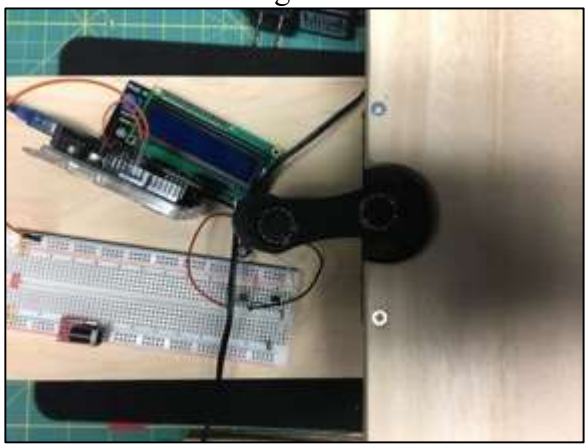
Media

Youtube Video: https://youtu.be/9JyMNL9qARE



First Design of Shaker

Underside of First Design



Final design of Shaker

Underside of Final Design

Code

```
// Project: Time-Sensitive Mechanics
// Purpose: To explore more with motors, RTC's, and LCD
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2020 05 09
// Status : Working

#include <Wire.h>
#include <TimeLib.h>
#include <DS1307RTC.h>
#include <LiquidCrystal.h>

const int rs = 7, en = 6, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

uint8_t x;
uint8_t y;
uint8_t z;
uint8_t a;
```

```
uint8_t b;
uint8_t c;

uint8_t motorPin = 9;

uint8_t shakeTime = 1;

void setup() {
  pinMode(motorPin, OUTPUT);
  pinMode(A2, OUTPUT);
  pinMode(A3, OUTPUT);
  digitalWrite(A2, LOW);
  digitalWrite(A3, HIGH);
  Serial.begin(9600);
  lcd.begin(16, 2);
  setScreen();
  analogWrite(motorPin, 255);
}

void print2digits(int number) {
  if (number >= 0 && number < 10) {
    lcd.print('0');
  }
  lcd.print(number);
}

void setScreen() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Time Start");
  lcd.setCursor(0, 1);
  tmElements_t tm;
  if (RTC.read(tm)) {
    x = tm.Hour;
    y = tm.Minute;
    z = tm.Second;
    print2digits(x);
    lcd.print(":");
    print2digits(y);
    lcd.print(":");
    print2digits(z);
  }
}

void motorOff() {
  tmElements_t tm;
  if (RTC.read(tm)) {
    a = tm.Hour;
    b = tm.Minute;
    c = tm.Second;
  }
  if (b == (y + 1)) {
    analogWrite(motorPin, 0);
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Time End");
    lcd.setCursor(0, 1);
```

```
    tmElements_t tm;
     print2digits(a);
     lcd.print(":");
     print2digits(b);
     lcd.print(":");
     print2digits(c);
   }
   else {
     motorOff();
   }
}

void loop() {

}
```

Project 2.11. Long ISP

Purpose

The purpose of the long ISP is to allow us to delve deeper into something that is interesting to us. This project uses a couple of different topics tought throughout this year. It combines these topics and some other stuff that was picked up along the way to create a 7-segment clock that will display the time, and school day or weekday.

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1920/ISPs.html

Procedure

The main part of this project is the actual 7-segment displays. The first thing that has to be done is to print off the 7-segmetns for this project. These are complety modular and can be rearranged to fit different needs. The 7-segments are fairly big to help it be seen from far away and help with the diffusion.

Parts List	
Component	Quantity
Arduino Nano	2
9V power jack	1
DS1307RTC	2
NeoPixel 60LED/m LEDs	1

The next part of the project is to connect the different back plates together. These are where the wires and LEDs will be. The LEDs can be placed in the middle of the trench in the back plate or for ease of wiring can be placed next to one of the edges. The LEDs are easier to program when they are connected in the same way for each segment. Since there is only going to be one LED per segment, the LED strip will have to be cut and soldered back together. The best way to solder the LED strips is to place the solder on before connecting the wires, and then remelting the solder and placing the wire in the solder. This is the fastest way to solder them back together and can save a lot of time.

Once all the LEDs are re soldered together, they have to be connected to the Arduino Nano and the RTC. The NeoPixel LEDs use 3 leads, power, data, and ground. These will be connected to 5V, digital pin 6, and ground respectively. Once these pins are connected, the RTC has to be connected to the Arduino Nano. The RTC has 4 pins that have to be connected, ground, power, SDA, and SCL. These pins will be connected to analog pin 2, analog pin 3, analog pin 4, and analog pin 5 respectivly. The SDA and SCL pins have to be connected to these exact pins on the Arduino because these are the pins needed for I2C communication. Once these are all connected up, the Vin on the two Arduino Nanos have to be hooked up to an external power source. To do this, we are going to use a 9V power jack. This can be put anywere where there is a hole big enough to fit the power jack. Once this is hooked up, we can break it into 2 different wires so it can go to both the 1st 7-segment clock and to the 2nd 7-segment clock. This can be done inside of the 7-segment clock on the bottom to leave minimal wires outside of the clocks. To connect the power to the second one, there will have to be a hole that lines up with the bottom of the top 7-segment on the top of the bottom 7-segment. Wires can then be run through these holes and go to Vin and ground on the second Arduino Nano.

The next part of this project is to figure out the code and how to display what is wanted. For both of the 7-segment clocks, we have to have different segment maps. The way that is easiest is to use `arrays` to map out what the segment should look like. For the clock, we will need the numbers 0 – 9, and for the school day clock, we will need 1-8, and most of the letters in the alphabet.

The main clock is pretty easy to display. We have to read from the RTC, and then take the `tens` and `ones` digit of both the hours and minutes. Once we have these numbers, we are able to display the segments easily if each segment was wired up in the same pattern. The way to do this is to go through the `array` for each number used, and set the pixel to turn on. Since the dots inbetween the hours and minutes is the start of the LED strip, the first segment has to have all the segments offset by an extra two. Each segment after will have to be 9, 16, and 23 for ones of hours, tens of minutes, and ones of minutes respectively.

The day of school week is a bit harder to code. There are a couple more `arrays` that we have to have that will allow us to write letters. After that, we need an `array` for what each school day it is in the year. This will have to be changed every year, but for testing purposes, it has gone as far as is visable so far. Once we input each day manualy with the holidays or weekends being 0, we have to display the day of the week. To do this we have gone the same route of using a segment map with the clock except there are no dots on this clock so we don't have to offset the first segment and can set it to 0. We are then able to display the 3 letters for day, and at the end we have to ask the RTC what month and day of month it is. Once we have these, we have another `array` that will tell us how many days are in a year for each month. We can then take the month we are on and subtract one and then create a `for` statement that will keep adding the numbers in our array of days in each month until it reaches the month we are in where it will not add that and then it will add the day that we are on. This will allow us to figure out what day of the year it is and go back into our original `array` and pull out the day of the school week it is. For each day that isn't a holiday it is pretty straight forware where it will just display the day that is in the `array`, but for the days that are holidays or weekends, we are going to have to either display a letter that says it is not a school day or do something different with the display. For this part it can be nice to know what day it is, when you are on breaks, it can feel like you don't know what day it is. To do this, we will go through all of the tasks to find what school day it is and then we can see if it is zero. If this ends up being zero, we can just read from the RTC again and ask for what day of the week it is. This will give us a number from 1-7 where 1 is Sunday. We can use this number to figure out which day of the week it is and then display the three digit letters for that day of the week.

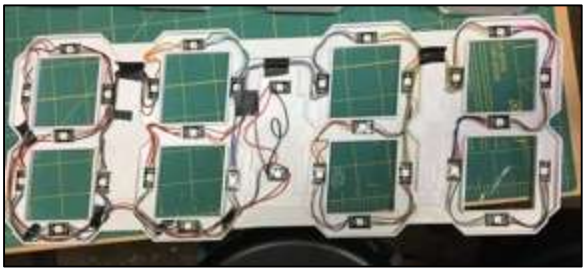
Reflection

This project was very difficult and probably the one that took me the longest to code. I found this a good expansion for my software experience vs hardware. It took me a long time to figure out the code and how to display everything I wanter. There definitely could be a much more effiecient way to do it but this was the best for me. I was really happy with the design of this project and the final product. It has been a great long full year of projects and learning and can't wait to do it all again next year even if it is long distance.

Media

Youtube Video: https://youtu.be/yVo8gQn_hA4

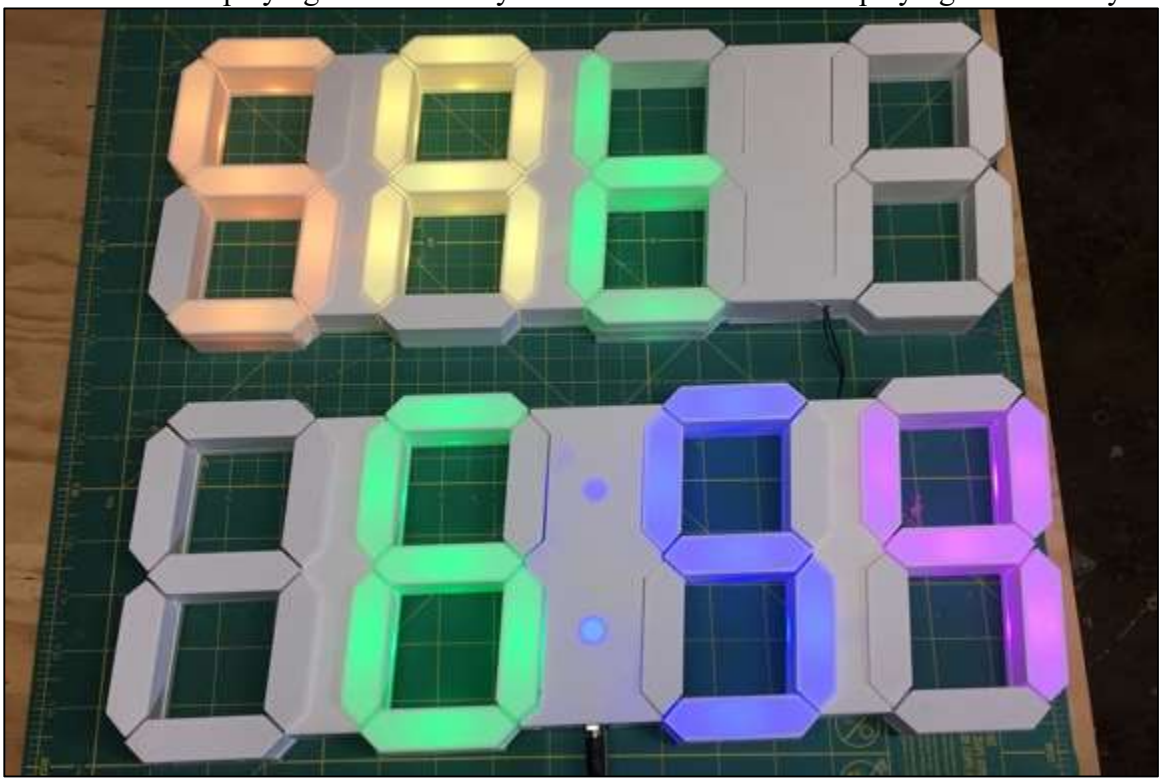
Clock before any wiring with just LEDs



Clock all wired up with LEDs

Clock displaying the school day

Clock displaying the weekday



Clock fully assembled and working

Code

```
// Project: Long ISP Day
// Purpose: To read an RTC and display the day of school
// Course : ISC3U
// Author : J. Vretenar
// Date   : 2020 05 30
// Status : Working
// Reference:

#include <Adafruit_NeoPixel.h>

#include <Wire.h>
#include <TimeLib.h>
#include <DS1307RTC.h>

static int digitOne[7] = { 0, 0, 1, 0, 0, 0, 1};
static int digitTwo[7] = { 0, 1, 1, 1, 1, 1, 0};
static int digitThree[7] = { 0, 1, 1, 1, 0, 1, 1};
static int digitFour[7] = { 1, 0, 1, 1, 0, 0, 1};
static int digitFive[7] = { 1, 1, 0, 1, 0, 1, 1};
static int digitSix[7] = { 1, 1, 0, 1, 1, 1, 1};
static int digitSeven[7] = { 0, 1, 1, 0, 0, 0, 1};
static int digitEight[7] = { 1, 1, 1, 1, 1, 1, 1};
static int digitNine[7] = { 1, 1, 1, 1, 0, 0, 1};
static int digitZero[7] = { 1, 1, 1, 0, 1, 1, 1};
static int digitS[7] = { 1, 1, 0, 1, 0, 1, 1};
static int digitA[7] = { 1, 1, 1, 1, 1, 0, 1};
static int digitT[7] = { 1, 0, 0, 1, 1, 1, 0};
static int digitM[7] = { 0, 1, 0, 0, 1, 0, 1};
static int digitO[7] = { 0, 0, 0, 1, 1, 1, 1};
static int digitN[7] = { 0, 0, 0, 1, 1, 0, 1};
static int digitU[7] = { 0, 0, 0, 0, 1, 1, 1};
static int digitE[7] = { 1, 1, 0, 1, 1, 1, 0};
static int digitW[7] = { 1, 0, 1, 0, 0, 1, 0};
static int digitD[7] = { 0 , 0, 1, 1, 1, 1, 1};
static int digitH[7] = { 1, 0, 0, 1, 1, 0, 1};
static int digitF[7] = { 1, 1, 0, 1, 1, 0, 0};
static int digitR[7] = { 0, 0, 0, 1, 1, 0, 0};
static int digitI[7] = { 0, 0, 0, 0, 0, 0, 1};
static int digitY[7] = { 1, 0, 1, 1, 0, 1, 1};

int year20[366] = {
  0, 0, 0, 0,
  0, 0, 8, 1, 2, 3, 0,
  0, 4, 5, 6, 7, 8, 0,
  0, 1, 2, 3, 4, 5, 0,
  0, 6, 7, 8, 1, 2, 0,
  0, 3, 4, 5, 6, 7, 0,
  0, 8, 1, 2, 3, 0, 0,
  0, 0, 4, 5, 6, 7, 0,
  0, 8, 1, 2, 3, 4, 0,
  0, 5, 6, 7, 8, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 1, 2, 3, 4, 0,
  0, 5, 6, 7, 8, 1, 0,
```

```
  0, 2, 3, 4, 5, 0, 0,
   0, 0, 6, 7, 8, 1, 0,
   0, 2, 3, 4, 5, 6, 0,
   0, 7, 8, 1, 2, 3, 0,
   0, 4, 5, 6, 7, 8, 0,
   0, 1, 2, 3, 4, 5, 0,
   0, 0, 6, 7, 8, 1, 0,
   0, 2, 3, 4, 5, 6, 0,
   0, 7, 8, 1, 2, 3, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 1, 2, 0,
   0, 3, 4, 5, 6, 7, 0,
   0, 8, 1, 2, 3, 0, 0,
   0, 4, 5, 6, 7, 8, 0,
   0, 1, 2, 3, 4, 5, 0,
   0, 0, 6, 7, 8, 1, 0,
   0, 2, 3, 4, 5, 6, 0,
   0, 7, 8, 1, 2, 3, 0,
   0, 4, 5, 6, 7, 8, 0,
   0, 1, 2, 3, 4, 0, 0,
   0, 0, 5, 6, 7, 0, 0,
   0, 1, 2, 3, 4, 5, 0,
   0, 6, 7, 8, 1, 2, 0,
   0, 3, 4, 5, 6, 7, 0,
   0, 8, 1, 2, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0
};

int year21[365] = {
  0, 0,
  0, 0, 3, 4, 5, 6, 0,
  0, 7, 8, 1, 2, 3, 0,
  0, 4, 5, 6, 7, 8, 0,
  0, 1, 2, 3, 4, 5, 0,
  0, 6, 7, 8, 1, 2, 0,
  0, 3, 4, 5, 6, 0, 0,
  0, 0, 7, 8, 1, 2, 0,
  0, 3, 4, 5, 6, 7, 0,
  0, 8, 1, 2, 3, 4, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 5, 6, 7, 8, 1, 0,
```

```
  0, 2, 3, 4, 5, 0, 0,
  0, 0, 6, 7, 8, 1, 0,
  0, 2, 3, 4, 5, 6, 0,
  0, 7, 8, 1, 2, 3, 0,
  0, 4, 5, 6, 7, 8, 0,
  0, 1, 2, 3, 4, 5, 0,
  0, 6, 7, 8, 1, 2, 0,
  0, 3, 4, 5, 6, 7, 0,
  0, 0, 8, 1, 2, 3, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, //29,30,31,1,2,3,4 - sep
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0
};

uint32_t color1 = 0;
uint32_t color2 = 0;
uint32_t color3 = 0;
uint32_t red = 0;
uint32_t yellow = 0;
uint32_t orange = 0;
uint32_t blue = 0;
uint32_t indigo = 0;
uint32_t violet = 0;
uint32_t green = 0;
uint32_t rainbow[7];

#define LED_PIN    6
```

```
#define LED_COUNT 30

Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);

void setup() {

//RAINBOW
  red = strip.Color(255, 0, 0);
  orange = strip.Color(255,127,0);
  yellow = strip.Color(255,255,0);
  green = strip.Color(0,255,0);
  blue = strip.Color(0,0,255);
  indigo = strip.Color( 63, 0, 108);
  violet = strip.Color(148,0,211);

/*  // BLUE PALETTE
  red = strip.Color(66, 165, 245);
  orange = strip.Color(33, 150, 243);
  yellow = strip.Color(25,118,210);
  green = strip.Color(13,71,161);
  blue = strip.Color(21,101,192);
  indigo = strip.Color( 25,118,210);
  violet = strip.Color(33, 150, 243);
*/

  // OCEAN PALETTE
  /*red = strip.Color(139,217,199);
  orange = strip.Color(72,191,245);
  yellow = strip.Color(0,118,190);
  green = strip.Color(10,143,189);
  blue = strip.Color(64,166,205);
  indigo = strip.Color(85,191,212);
  violet = strip.Color(128,211,221);*/

  rainbow[0] = red;
  rainbow[1] = orange;
  rainbow[2] = yellow;
  rainbow[3] = green;
  rainbow[4] = blue;
  rainbow[5] = indigo;
  rainbow[6] = violet;
  color1 = 0;
  color2 = 1;
  color3 = 2;

  pinMode(A2, OUTPUT);
  pinMode(A3, OUTPUT);
  digitalWrite(A2, LOW);
  digitalWrite(A3, HIGH);
  strip.begin();
  strip.show();
  strip.setBrightness(255);
}

void showSchoolDay(uint8_t schoolDay) {
```

```
 showDAY(schoolDay);
  showDigit(getDigitMap(schoolDay), 21, strip.Color( 0, 0, 255));
}

void showWeekday(uint8_t wday, uint32_t color1, uint32_t color2, uint32_t
color3) {
  if (wday == 1) {
    showDigit(digitS, 0, color1);
    showDigit(digitU, 7, color2);
    showDigit(digitN, 14, color3);
  }
  else if (wday == 2) {
    showDigit(digitM, 0, color1);
    showDigit(digitO, 7, color2);
    showDigit(digitN, 14, color3);
  }
  else if (wday == 3) {
    showDigit(digitT, 0, color1);
    showDigit(digitU, 7, color2);
    showDigit(digitE, 14, color3);
  }
  else if (wday == 4) {
    showDigit(digitW, 0, color1);
    showDigit(digitE, 7, color2);
    showDigit(digitD, 14, color3);
  }
  else if (wday == 5) {
    showDigit(digitT, 0, color1);
    showDigit(digitH, 7, color2);
    showDigit(digitU, 14, color3);
  }
  else if (wday == 6) {
    showDigit(digitF, 0, color1);
    showDigit(digitR, 7, color2);
    showDigit(digitI, 14, color3);
  }
  else {
    showDigit(digitS, 0, color1);
    showDigit(digitA, 7, color2);
    showDigit(digitT, 14, color3);
  }
}

void showDAY(uint8_t schoolDay) {
  showDigit(digitD, 0, strip.Color( 255, 0, 0));
  showDigit(digitA, 7, strip.Color( 255, 128, 0));
  showDigit(digitY, 14, strip.Color( 0, 255, 0));
}

void showDigit(int digitMap[], int offset, uint32_t color) {
  for (int i = 0; i < 7; i++)
  {
    if (digitMap[i] == 1) {
      strip.setPixelColor(offset + i, color);
    }
```

```
  }
}

int getDigitMap(int num)
{

  if (num == 1)
    return digitOne;
  else if (num == 2)
    return digitTwo;
  else if (num == 3)
    return digitThree;
  else if (num == 4)
    return digitFour;
  else if (num == 5)
    return digitFive;
  else if (num == 6)
    return digitSix;
  else if (num == 7)
    return digitSeven;
  else if (num == 8)
    return digitEight;
  else if (num == 9)
    return digitNine;
  else
    return digitZero;

}

void loop() {
  strip.clear();
  tmElements_t tm;
  RTC.read(tm);
  uint16_t currentDayOfYear = 0;
  uint16_t currentYear = tm.Year;
  uint8_t schoolDay = 0;
  if (currentYear == 2020) {
    currentDayOfYear = leapYearDay(tm.Day, tm.Month);
    schoolDay = year20[currentDayOfYear];
  }
  else {
    currentDayOfYear = yearDay(tm.Day, tm.Month);
    schoolDay = year21[currentDayOfYear];
  }

  if (schoolDay == 0)
    showWeekday(tm.Wday, rainbow[color1], rainbow[color2],rainbow[color3]);
  else
    showSchoolDay(schoolDay);

  strip.show();

  color1 = (color1+1)%7;
  color2 = (color2+1)%7;
```

```
 color3 = (color3+1)%7;
  delay(250);

}

uint16_t leapYearDay(uint16_t d, uint16_t m) {
  int daysInLeapMonth[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31};
  uint16_t daysInYear = 0;
  for (uint16_t i = 0; i < 12; i++)
  {
    if (i == m - 1)
      break;
    daysInYear += daysInLeapMonth[i];

  }
  daysInYear += d;
  return daysInYear;
}

uint16_t yearDay(uint16_t d, uint16_t m) {
  int daysInMonth[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  uint16_t daysInYear = 0;
  for (uint16_t i = 0; i < 12; i++)
  {
    if (i == m - 1)
      break;
    daysInYear += daysInMonth[i];

  }
  daysInYear += d;
  return daysInYear;
}
```

```
// Project: Long ISP clock
// Purpose: To read an RTC and display the time
// Course : ICS3U
// Author : J. Vretenar
// Date   : 2020 05 30
// Status : Working
// Reference:

#include <Adafruit_NeoPixel.h>

#include <Wire.h>
#include <TimeLib.h>
#include <DS1307RTC.h>

static int digitOne[7] = { 0, 0, 1, 0, 0, 0, 1};
static int digitTwo[7] = { 0, 1, 1, 1, 1, 1, 0};
static int digitThree[7] = { 0, 1, 1, 1, 0, 1, 1};
static int digitFour[7] = { 1, 0, 1, 1, 0, 0, 1};
static int digitFive[7] = { 1, 1, 0, 1, 0, 1, 1};
static int digitSix[7] = { 1, 1, 0, 1, 1, 1, 1};
static int digitSeven[7] = { 0, 1, 1, 0, 0, 0, 1};
```

```
static int digitEight[7] = { 1, 1, 1, 1, 1, 1, 1};
static int digitNine[7] = { 1, 1, 1, 1, 0, 0, 1};
static int digitZero[7] = { 1, 1, 1, 0, 1, 1, 1};

#define LED_PIN    6

#define LED_COUNT 30

uint32_t color1 = 0;
uint32_t color2 = 0;
uint32_t color3 = 0;
uint32_t color4 = 0;
uint32_t red = 0;
uint32_t yellow = 0;
uint32_t orange = 0;
uint32_t blue = 0;
uint32_t indigo = 0;
uint32_t violet = 0;
uint32_t green = 0;
uint32_t rainbow[7];

uint8_t dots = 0;

Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  //RAINBOW
  red = strip.Color(255, 0, 0);
  orange = strip.Color(255, 127, 0);
  yellow = strip.Color(255, 255, 0);
  green = strip.Color(0, 255, 0);
  blue = strip.Color(0, 0, 255);
  indigo = strip.Color( 63, 0, 108);
  violet = strip.Color(148, 0, 211);

  /*  // BLUE PALETTE
    red = strip.Color(66, 165, 245);
    orange = strip.Color(33, 150, 243);
    yellow = strip.Color(25,118,210);
    green = strip.Color(13,71,161);
    blue = strip.Color(21,101,192);
    indigo = strip.Color( 25,118,210);
    violet = strip.Color(33, 150, 243);
  */

  //OCEAN PALETTE
  /*red = strip.Color(139,217,199);
    orange = strip.Color(72,191,245);
    yellow = strip.Color(0,118,190);
    green = strip.Color(10,143,189);
    blue = strip.Color(64,166,205);
    indigo = strip.Color(85,191,212);
    violet = strip.Color(128,211,221);*/

  rainbow[0] = red;
  rainbow[1] = orange;
```

```
rainbow[2] = yellow;
  rainbow[3] = green;
  rainbow[4] = blue;
  rainbow[5] = indigo;
  rainbow[6] = violet;
  color1 = 0;
  color2 = 1;
  color3 = 2;
  color4 = 4;

  pinMode(A2, OUTPUT);
  pinMode(A3, OUTPUT);
  digitalWrite(A2, LOW);
  digitalWrite(A3, HIGH);
  strip.begin();
  strip.show();
  strip.setBrightness(255);
}


void loop() {

  // get the time

  // show the time
  tmElements_t tm;
  RTC.read(tm);
  int h = tm.Hour;
  if (h > 12)
    h -= 12;
  int m = tm.Minute;
  strip.clear();
  showTime(h, m, rainbow[color1], rainbow[color2], rainbow[color3],
rainbow[color4]);

  if (dots == 1) {
    strip.setPixelColor(0, rainbow[color3]);
    strip.setPixelColor(1, rainbow[color3]);
  }
  strip.show();

  color1 = (color1 + 1) % 7;
  color2 = (color2 + 1) % 7;
  color3 = (color3 + 1) % 7;
  color4 = (color4 + 1) % 7;
  dots = (dots + 1) % 2;
  delay(1000);
}

int getDigitMap(int num)
{

  if (num == 1)
    return digitOne;
  else if (num == 2)
    return digitTwo;
```

```
else if (num == 3)
    return digitThree;
  else if (num == 4)
    return digitFour;
  else if (num == 5)
    return digitFive;
  else if (num == 6)
    return digitSix;
  else if (num == 7)
    return digitSeven;
  else if (num == 8)
    return digitEight;
  else if (num == 9)
    return digitNine;
  else
    return digitZero;

}

void showTime(int h, int m, uint32_t color1, uint32_t color2, uint32_t
color3, uint32_t color4)
{
  showHour(h, color1, color2);
  showMinute(m, color3, color4);
}

void showHour(int h, uint32_t color1, uint32_t color2)
{
  int tens;
  int ones;
  tens = h / 10;
  ones = h % 10;


  if (tens == 1) {
    showDigit(getDigitMap(tens), 2, color1);
  }
  showDigit(getDigitMap(ones), 9, color2);


}

void showMinute(int m, uint32_t color1, uint32_t color2)
{
  int tens = m / 10;
  int ones = m % 10;

  showDigit(getDigitMap(tens), 16, color1);
  showDigit(getDigitMap(ones), 23, color2);

}


void showDigit(int digitMap[], int offset, uint32_t color) {
  for (int i = 0; i < 7; i++)
  {
```

```
    if (digitMap[i] == 1) {
       strip.setPixelColor(offset + i, color);
     }
   }
 }
```

ICS4U

Project 3.1. GB Machine

Purpose

The purpose of the GB Machine is to help introduce us into surface mount soldering, as well as giving us a device that will provide 5v without taking up space on a breadboard for voltage regulation.

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/2021/Tasks.html#GBMachine

Procedure

The Compnents for this project are common parts that we have used before. The new parts are the surface mount LED, and resistor.

The surface mount compnents can be done in a couple of ways, they can be soldered with a small tip and thin solder, they can be soldered with hot air and solder paste or a stencil, or they can be soldered with a custom soldering oven and solder paste. The resistor like normal resistors can be rotated any way, but the LED has a positive side and a negative side that is indicated by a green markings.

Parts List	
Component	Quantity
DC Jack	1
SPDT Switch	2
7805 Voltage Regulator	1
2×3 Male Headers	2
1kΩ Surface Mount Resistor	1
Blue 1206 LED	1
Power Diode	1

Once the surface mount parts are soldered on, the through hole parts have to be soldered on. These are more normal and easier to solder on. After these parts are on, the GB machine is finished.

To build the circuit after, it is just a wire running from positive to the positive lead of 5 different LEDs, and from the negative lead of those 5 LEDs, there are 5 220Ω resistors connecting them to ground.

Reflection

This project was fairly simple and was a good introduction to surface mount soldering. I had done some surface mount practice in the summer with the hot air soldering technique so I thought I would try the soldering pen. This went very well and was simple. The open ended side of this project with no given function for the GB machine allowed me to try something new with some plexiglass and my CNC machine. Overall I liked how my project turned out and can't wait to continue in my final year of Hardware.

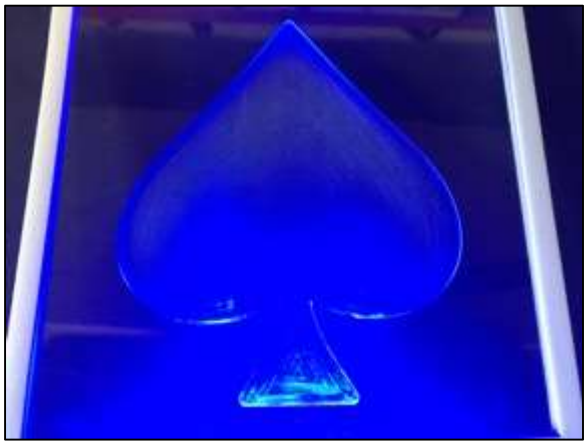
Media

Youtube Video: https://youtu.be/fF7hYvBlpLw

GB machine top

GB machine bottom

GB machine plugged in with the switch on



GB machine powering LEDs lighting up the plexiglass Ace

Project 3.2. CHUMP

Overall Theory

This project will help to expand the knowledge of lower level programming as well as creating a fully programmable 4-bit computer.

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html

Project 3.2.1. Code

Purpose

The purpose of this part of the CHUMP process is to act as a introduction into the code that will be used on the full build of the CHUMP. This will help to transition from the normal high level coding done in Arduino, and switch to a lower level using either binary or hex to execute commands.

Procedure

This code is supposed to mimic a vending machine. When money is inserted into a vending machine, the total required to buy a pop is reduced. The cost of a pop is 15 cents, and only 5 cents can be inserted at a time. This code will loop through adding 5 cents until it reaches 15 cents, once the machine has 15 cents inserted, a pop will be dispensed. This means that the stock of pop that is in the machine must be kept count of. When all 15 cents are inserted, then 1 pop will be removed from the total amount of 10 in the machine. When the machine reaches 0 total pop, it repeates an infinite while loop where it won't insert pop until reset.

Code

High Level	Address	Instruction	CHUMP level	Comment
`s = 10;`	`0000`	`0000 1010`	`LOAD 10`	`accum = 10; pc++`
	`0001`	`0110 0000`	`STORETO s`	`mem[0] = accum; pc++`
`c = 15;`	`0010`	`0000 1111`	`LOAD 15`	`accum = 15; pc++`
	`0011`	`0110 0001`	`STORETO c`	`mem[1] = accum; pc++`
`while c != 0;`	`0100`	`1110 0111`	`IFZERO 7`	`accum==0?pc=7: pc++`
`c = c - 5;`	`0101`	`0100 0101`	`SUBTRACT 5`	`accum -= 5; pc++`
	`0110`	`1100 0011`	`GOTO 3`	`pc = 3; pc++`
	`0111`	`1000 0000`	`READ s`	`addr = 0; pc++`
`while s != 0;`	`1000`	`1110 1100`	`IFZERO 12`	`accum==?pc=12: pc++`
`s--;`	`1001`	`0100 0001`	`SUBTRACT 1`	`accum -= 1; pc++`
	`1010`	`0110 0000`	`STORETO s`	`mem[0] = accum; pc++`
	`1011`	`1100 0010`	`GOTO 2`	`pc = 2; pc++`
`pause;`	`1100`	`1100 1101`	`GOTO 12`	`pc = 12; pc++`

Project 3.2.2. Clock

Purpose

The purpose of the clock build in the CHUMP is to create a device that can switch between automatic counting and manual counting to allow the program that is written on our computer to advance. This clock will also be used for many different parts of the CHUMP to keep them in time and synced.

Procedure

The main components needed for the clock part of the CHUMP are the three 555 ICs, the SN74LS04N AND gate IC, the SN74LS08N INVERTER gate IC, and the SN74LS32N OR gate IC.

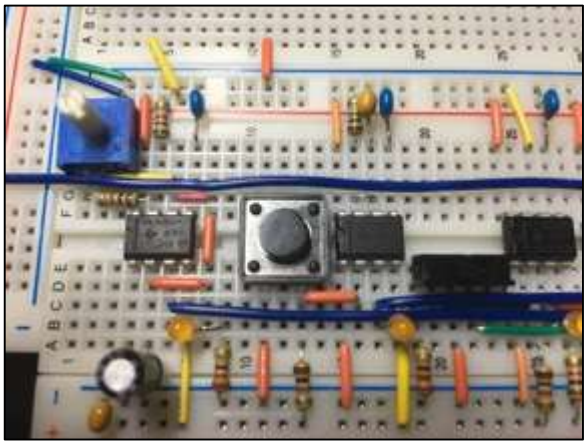
Parts List	
Component	Quantity
555 IC	3
SN74LS04N	1
SN74LS08N	1
SN74LS32N	1

The first 555 will be used to provide a constant square wave that has adjustable speed also known as astable. The second 555 will be used as a button debouncer to allow for a manual pulse to advance the counter. The final 555 will use a switch that is debounced with the 555 that can switch a LED on or off we'll call it select.

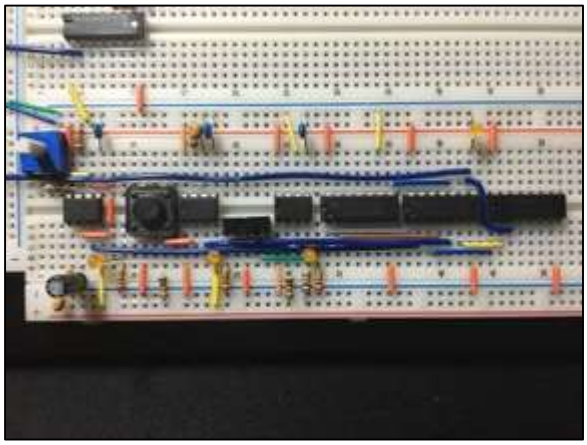
The AND, INVERTER, and OR gates are used to connect these 555s together for one output. The astable pulse will go into one input on an AND gate and the other input will be one of the select lines. The other select line goes into the INVERTER and then goes into a second AND gate, the other input on the second AND gate is the manual pulse. The outputs of both the AND gates go into an OR gate and the output of the OR gate is the final clock pulse. The way these gates and inputs are set up will allow us to switch between the manual and astable pulse by switching the switch that is connected to the third 555.

Media

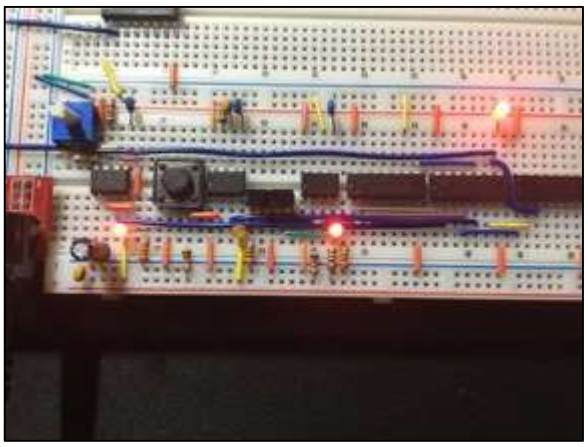
Youtube Video: https://youtu.be/zfDGYWAn-cw



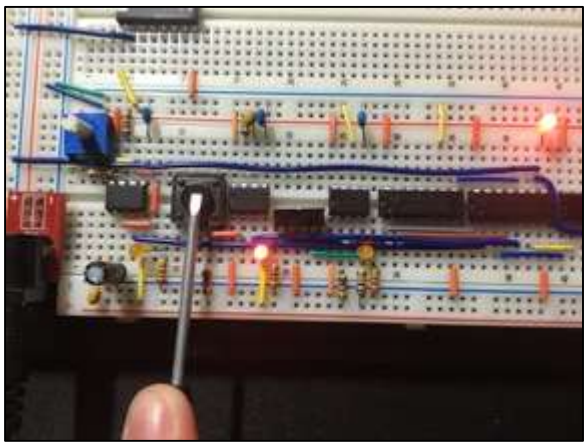
Close up of the 555 setup



Full Clock module build



Clock module on counting with astable
(output is top right)



Clock module on counting with manual
(output is top right)

Project 3.2.3. Counter

Purpose

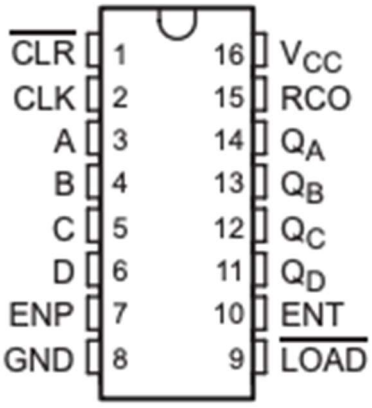
The purpose of the counter or program counter part of the CHUMP is to keep track of what program out of the 16 that can be coded into our EEPROM we are on. Whenever a command is executed, the program counter will increase by one and move onto the next command in the EEPROM.

Procedure

The 74LS161A is a 4 bit binary counting chip, very similar to the chip that was used back in Project 1.4. The Counting Circuit. This chip as used in this context will take a clock pulse on input and count from zero to 15 and then rollover. The difference between this chip and the one used in The Counting Circuit is this has a few extra features.

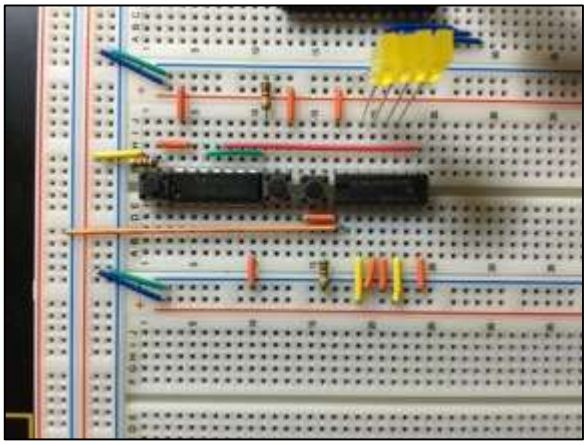
Parts List	
Component	Quantity
SN74LS161A	1
Yellow LEDs	4
SN74LS00	4

The chip has the normal pin 8 GND and pin 16 Vcc, the clock input goes into pin 2, and the 4 outputs are Q_A - Q_D respectively. RCO is the ripple clock output, this is an output that can be enabled to show spikes in the counting. This pin is enabled when ENP and ENT are high. Pin 1 is the CLR pin and will reset the outputs when set low. The final and one of the more important pins for the CHUMP are pins 3 - 6 as inputs A – D respectively, and pin 9 which is the LOAD pin. When the LOAD pin is set to low, it will enable it and make the ouputs mimic the inputs. This is useful as it allows the program counter to switch between any command line as long as the input it that command line and the LOAD pin is low.

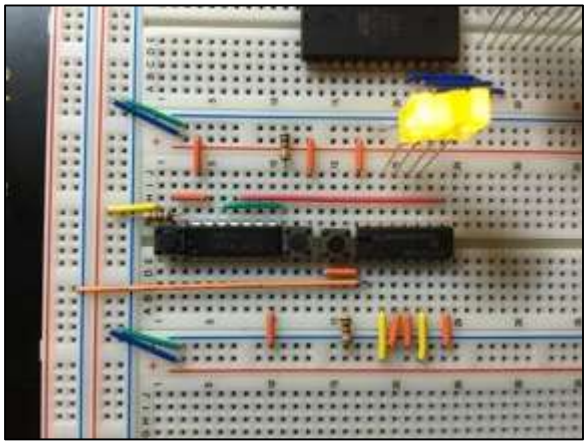


When the CHUMP is fully assembled, there will be a NAND gate that goes into the LOAD pin. This will allow the CHUMP to execute `GOTO` commands. When the LOAD pin is low, the CHUMP can set input pins high to allow it to go to a different line in the code. To simulate that computer input, 2 push buttons and attached to the inputs of a NAND gate and when activated will set the output to low, which results in the LOAD pin being low and the pre wired highs and lows for the input will display on the LEDs, in this case `1001`.

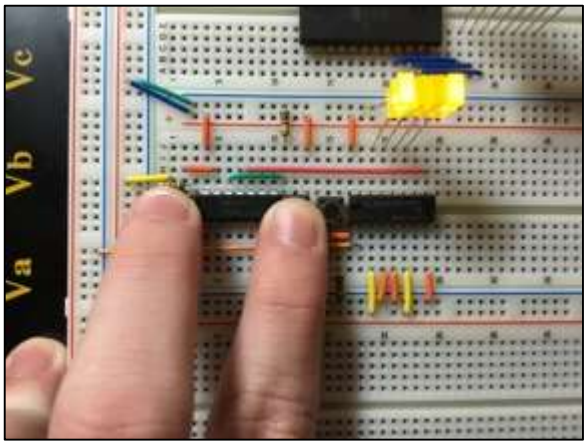
Media
Youtube Video: https://youtu.be/zfDGYWAn-cw



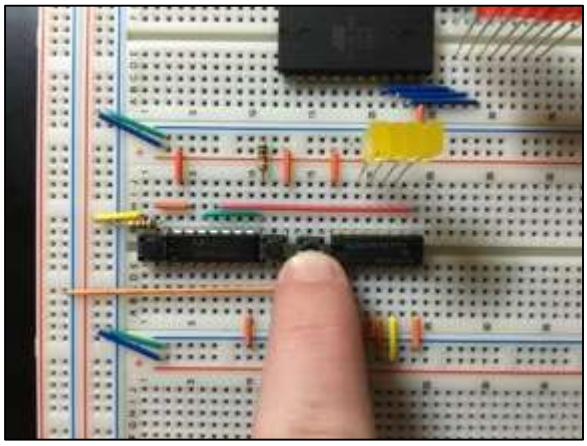
Counter off



Counter On



Counter with Load pin active



Counter with clear pin active

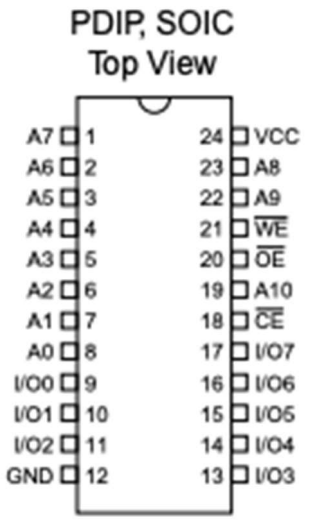
Project 3.2.4. Program EEPROM

Purpose

The purpose of the Program EEPROM is to hold the program for the CHUMP. This program will be displayed on eight output LEDs wired up to the EEPROM.

Procedure

The Program EEPROM, or the AT28C16A IC, is wired up to ground and power just like any other chip. Once this is done there are a few other pins that can be wired to ground and power. These other pins are pin 21, $\overline{WE}$, pin 20 $\overline{OE}$,, and pin 18 $\overline{CE}$,. These stand for write enable, output enable, and chip enable. These pins have a line above the two letters so that means that they are active when low. Since we are using this chip, we always want the chip to be enabled so that can be wired to ground. The write enable and output enable are only grounded when one of those functions is needed. The write enable is grounded when the EEPROM is being programmed, and the output enable is grounded when the EEPROM program is being read from.

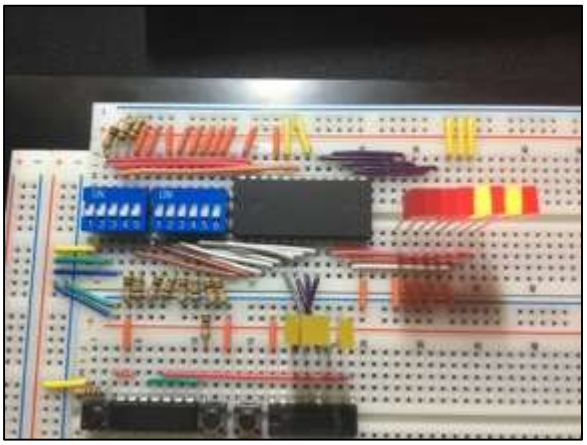


The rest of the pins on the EEPROM are addressed with letters followed by numbers. The I/O(0-7) pins are input or output pins. For this part of the CHUMP, they will be used as output for LEDs. The A(0-10) pins are the address pins. If the output from the program counter is not set in the EEPROM, these pins can be set to either high or low and that will result in the EEPROM showing what is at that value on the EEPROM.

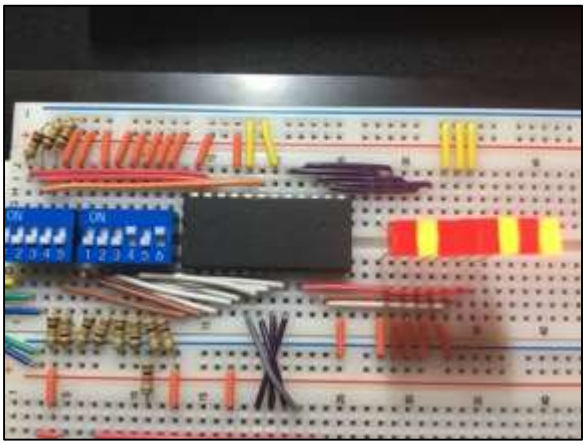
For this part in the CHUMP build, the 4 output pins on the program counter will connect to pins A0, A1, A2, and A3 respectivly. When the EEPROM is programmed and the clock is progressing the counter, the 8 LEDs that are wired out of the I/O pins will display that program that is on the EEPROM in binary.

When the EEPROM is first plugged in, it will be programmed to display all 8 of the LEDs as high. The way this EEPROM can be programmed is either done with a universal programmer, or breadboarded and connected an Arduino. In the universal programmer app, it will show up with the address on top, and two HEX inputs below. If we take the Code from earlier and turn the instruction values into HEX and then input them into the app. Once the EEPROM gets re programmed and set back into the CHUMP breadboard. The address that got changed should display the HEX instruction as binary again. If we take `0000 1010` which was the first line in the CHUMP code, it can be turned into the HEX value of `0A`. This can then be inputed into the first address of the app and uploaded. When the EEPROM is inserted back into place on the breadboard, and the counter is running. When the counters LEDs are all off, the LEDs on the EEPROM should show `0000 1010` which is the first line of the code.

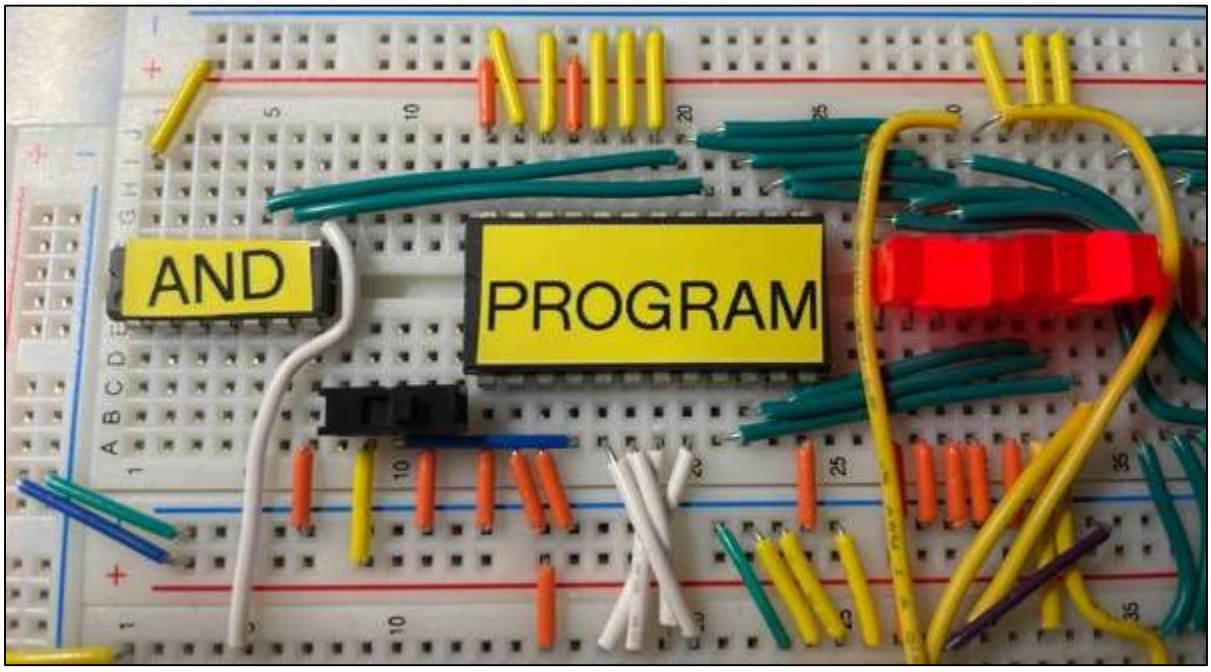
Media



EEPROM at Address 0



EEPROM at Address 5



Full and Final Program EEPROM

Project 3.2.5. Control EEPROM

Purpose

The purpose of the Control EEPROM is to hold the control codes that can be accessed by the Program EEPROM to execute the command.

Procedure

The Control EEPROM is the same AT28C16 chip as used in the Program EEPROM. It is wired in a fairly similar way to the Program, with the Gnd, OE, and CE grounded, and the Vcc, and WE pins connected to power.

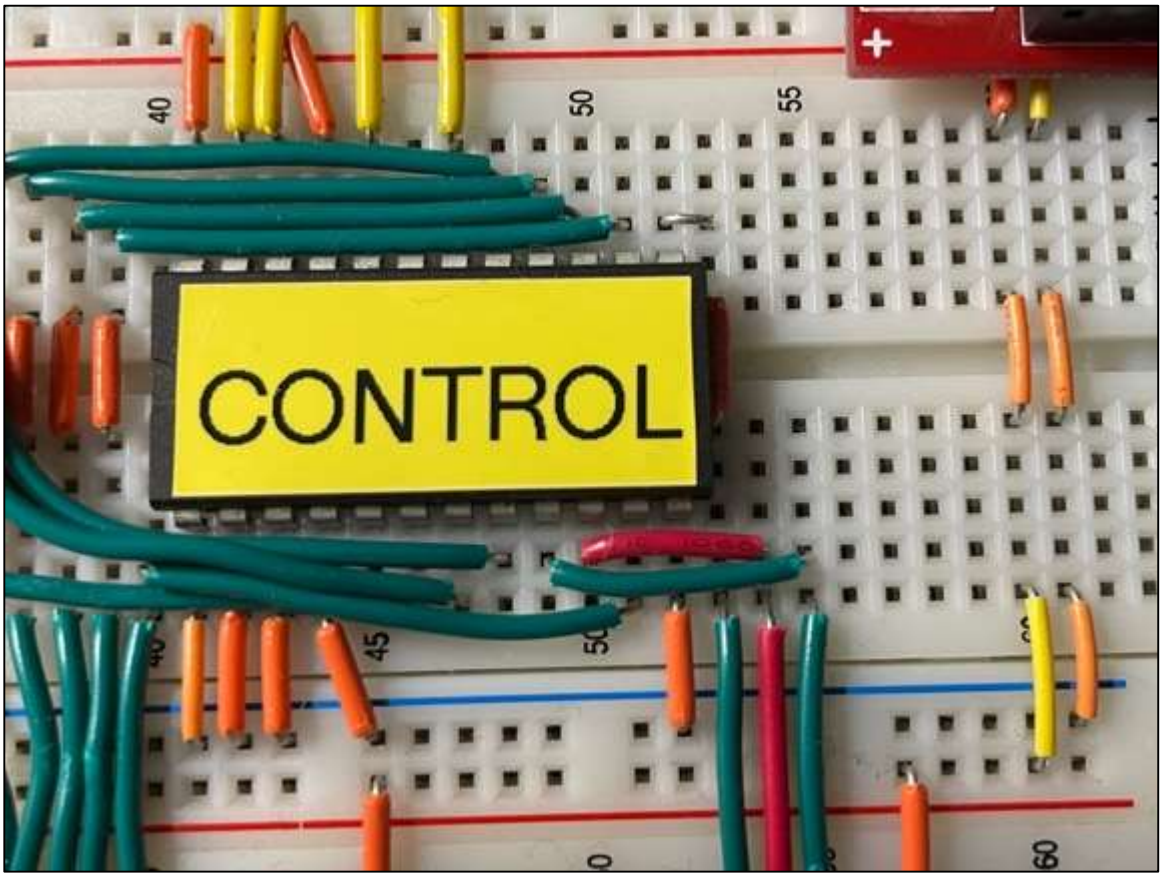
This time, the A0 to A3 pins are connected to the I/O-4 to I/O-7 pins from the Program EEPROM. The rest of the A pins are connected to ground. The I/O-4 to I/O-7 pins on the Control EEPROM will be connected to the ALU select.

The lower four outputs from the Control EEPROM will be wired into Ram, Accumulator, and the ALU. I/O-1 will go into the Chip Select on the Ram chip. I/O-2 will go into pin 1 on the Accumulator. I/O-3 and I/O-4 will go into the ALU for the Carry Input, and the Mode Control input respectively.

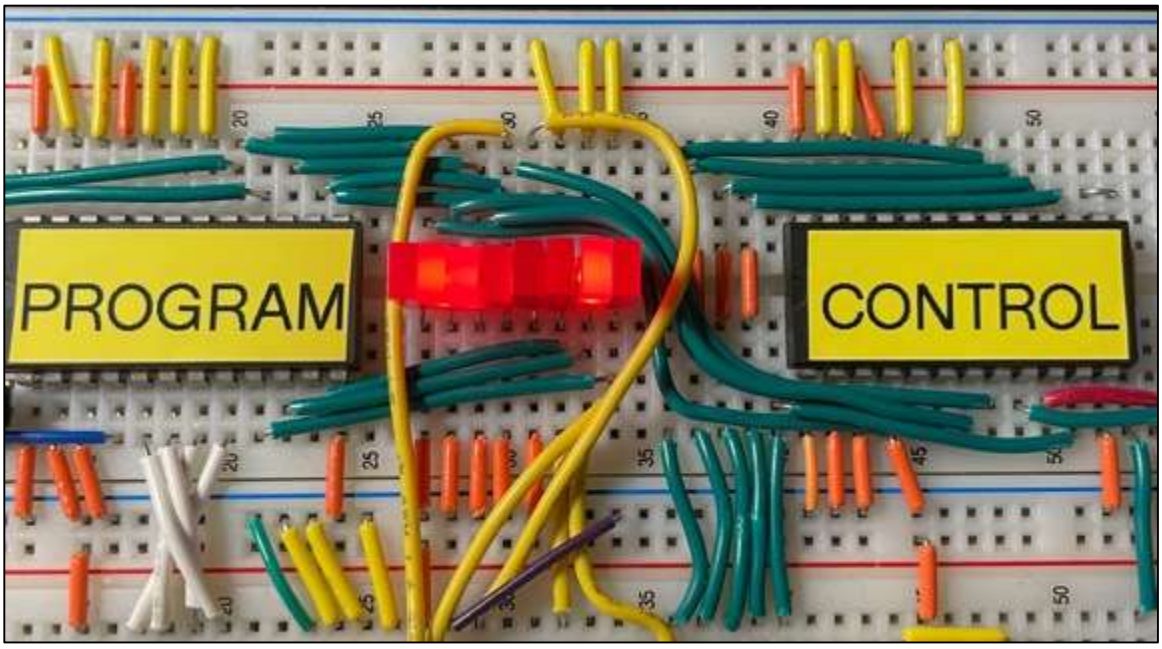
The code below is what would be programed onto the Control EEPROM chip. It is programmed onto the chip through the TL866 universal program in Hex. Once this is programmed onto the chip, it can be referenced on the CHUMP at any point. The main code OpCode has 3 different values that can be 0 or 1 and then the final value that can also be 0 or 1. The final value when it is 0, means that the second half of the main code should be used for a value. When the final value is 0, it means that the CHUMP will read from memeory and not use the second half.

Mnemonic	OpCode	Operand	Comment	Hex
LOAD	0000	Const	accum = const; pc++	0xA8
LOAD	0001	Mem	accum = mem[addr]; pc++	0xA9
ADD	0010	Const	accum += const; pc++	0x95
ADD	0011	Mem	accum += mem[addr]; pc++	0x95
SUBTRACT	0100	Const	accum -= const; pc++	0x61
SUBTRACT	0101	Mem	accum -= mem[addr]; pc++	0x61
STORETO	0110	Const	mem[const] = accum; pc++	0x02
STORETO	0111	Mem	mem[addr] = accum; pc++	0x02
READ	1000	Const	Accum = mem[0]; pc++	0x03
READ	1001	Mem	accum==?pc=12: pc++	0x03
	1010	Const		0x00
	1011	Mem		0x00
GOTO	1100	Const	pc = const; pc++	0xCB
GOTO	1101	Mem	pc = mem[addr]; pc++	0xCB
IFZERO	1110	Const	accum==0?pc=const: pc++	0x0B
IFZERO	1111	Mem	accum==0?pc=mem[addr]: pc++	0x0B

Media



Control EEPROM



Program and Control EEPROMs

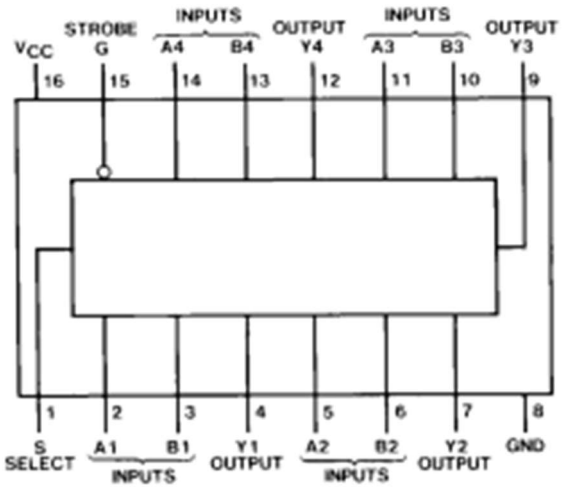
Project 3.2.6. Multiplexer

Purpose

The purpose of the Multiplexer is to choose between the outputs of the Control EEPROM, and the outputs of the Ram chip.

Procedure

The 74LS157 is a Quad 2-line to 1-line data selector, also known as a Multiplexer. This will take 4 groups of 2 inputs as the chips suggest and convert it to 4 groups of 1 outputs. This chip will take inputs on pins A1 to A4 from the Program EEPROMs lower 4 outputs. The B1 to B4 inputs will come from the output of the Ram chip. The G pin is able to be grounded along with the GND pin. Vcc is set to high, and the Select pin is connected to I/O-4 on the Program EEPROM. This Select pin when set low will send the inputs from the Program EEPROM through the outputs. While the Select pin is high though, it will send the inputs from Ram through the outputs. These final output values will go to the B pins on the ALU.

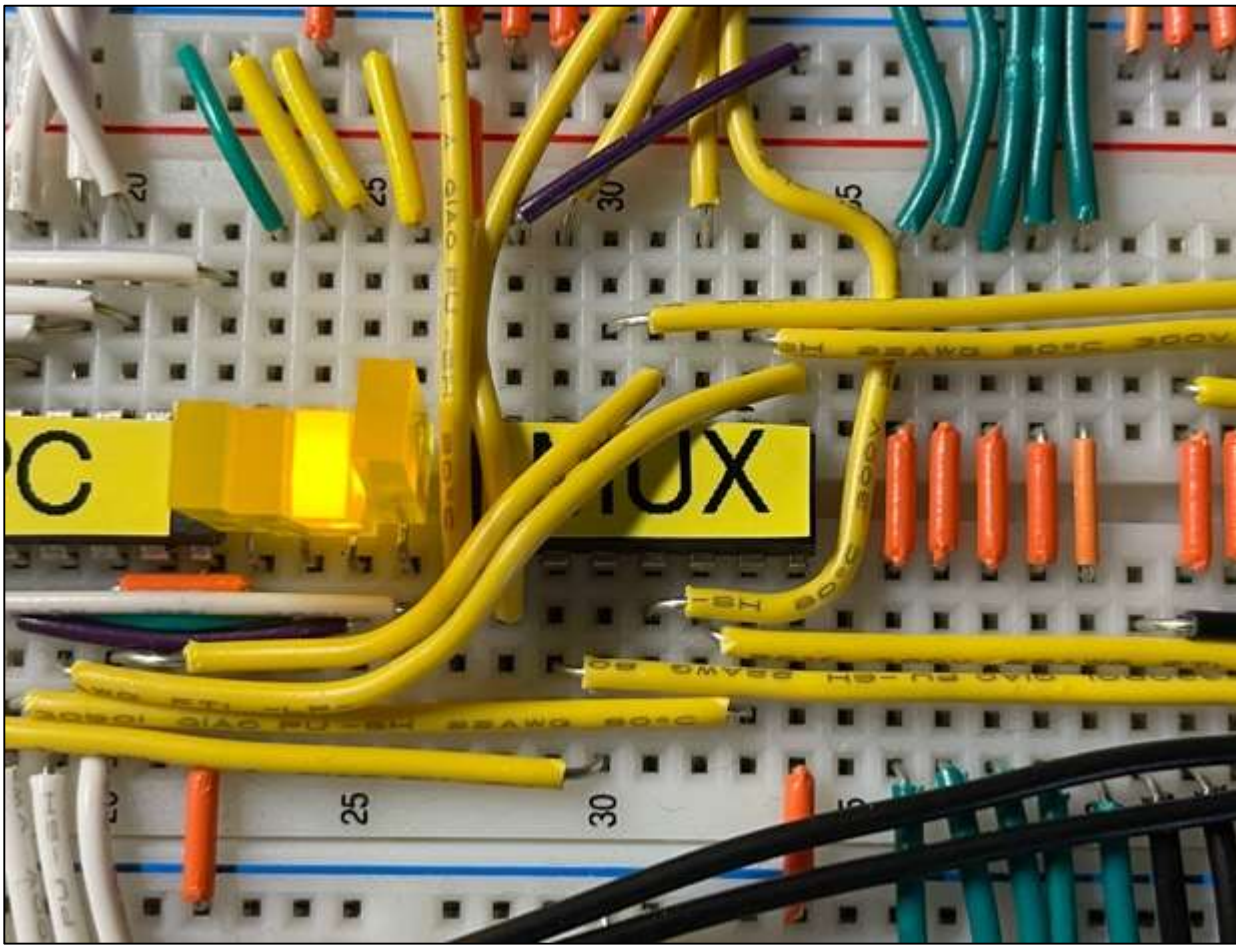


The output pins from the Multiplexer connects to the Program Counter. This will be used for the `GOTO` command to set the Program Counter to a specific value. This will allow it to switch between lines of code to be used like a `while` loop.

The select pin is triggered by the last bit on the OpCode which tells the CHUMP whether to display a const value or a value from memory.

Mnemonic	OpCode	Operand	Comment	Hex
LOAD	0000	Const	accum = const; pc++	0xA8
LOAD	0001	Mem	accum = mem[addr]; pc++	0xA9

Media



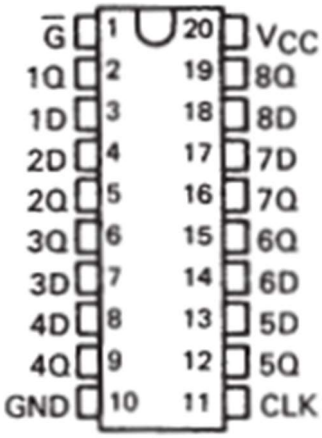
Multiplexer Wiring

Project 3.2.7. Accumulator

Purpose

The Accumulator, or Sn74LS377, is a flip-flop with an enable pin that connects the ALU to the RAM.

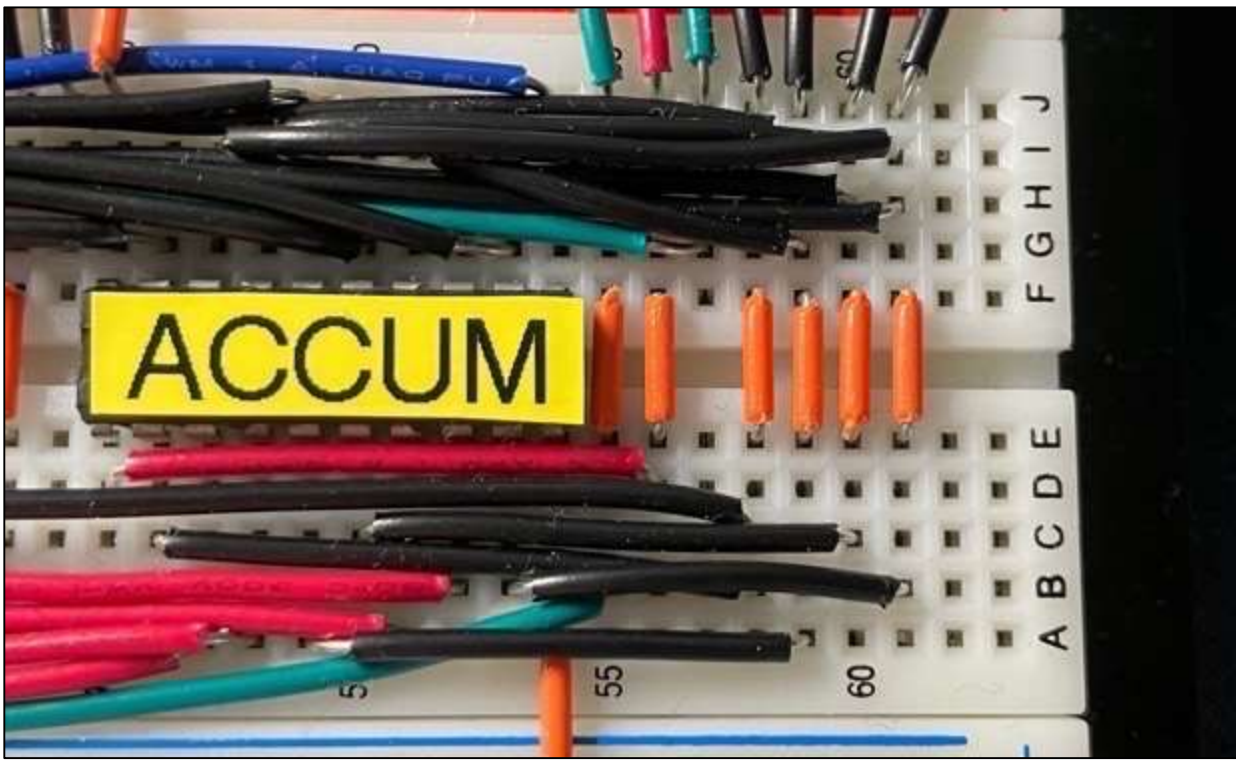
Procedure



The Accumulator is a Octal flip-flop that has the enable pin G. Pins 1 and 10 and be connected to VCC and GND respectively and pin 11 will connect to the clock output. Pin G is connected to the I/O-1 pin on the Control EEPROM. The Accumulator has the inputs D and outputs Q. For the CHUMP, only one side of the Accumulator is needed because there are 4 inputs and outputs.

The inputs from 1D to 4D are taken from the F pins on the ALU. These are put into the respective pins on the Accumulator and the ouputs from 1Q to 4Q are fed into a inverter that will go into the Ram chip.This chip will only trigger when the clock pulse is a rising, or positive-edge, and won't trigger on the falling, or negative-edge. While the clock is High or Low, it will not send a signal through the output. This will prevent a false clock signal that could break the system.

Media



Accumulator Wiring

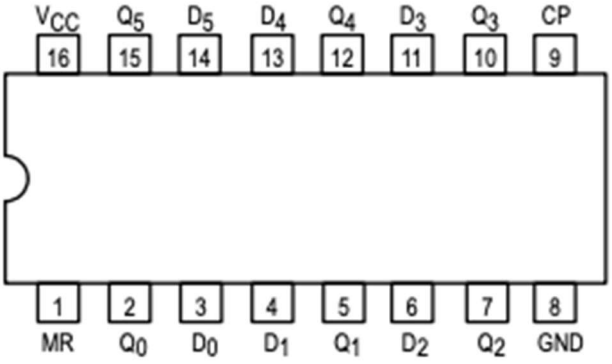
Project 3.2.8. Address

Purpose

The purpose of the Address is to go to the next R/W address for Ram.

Procedure

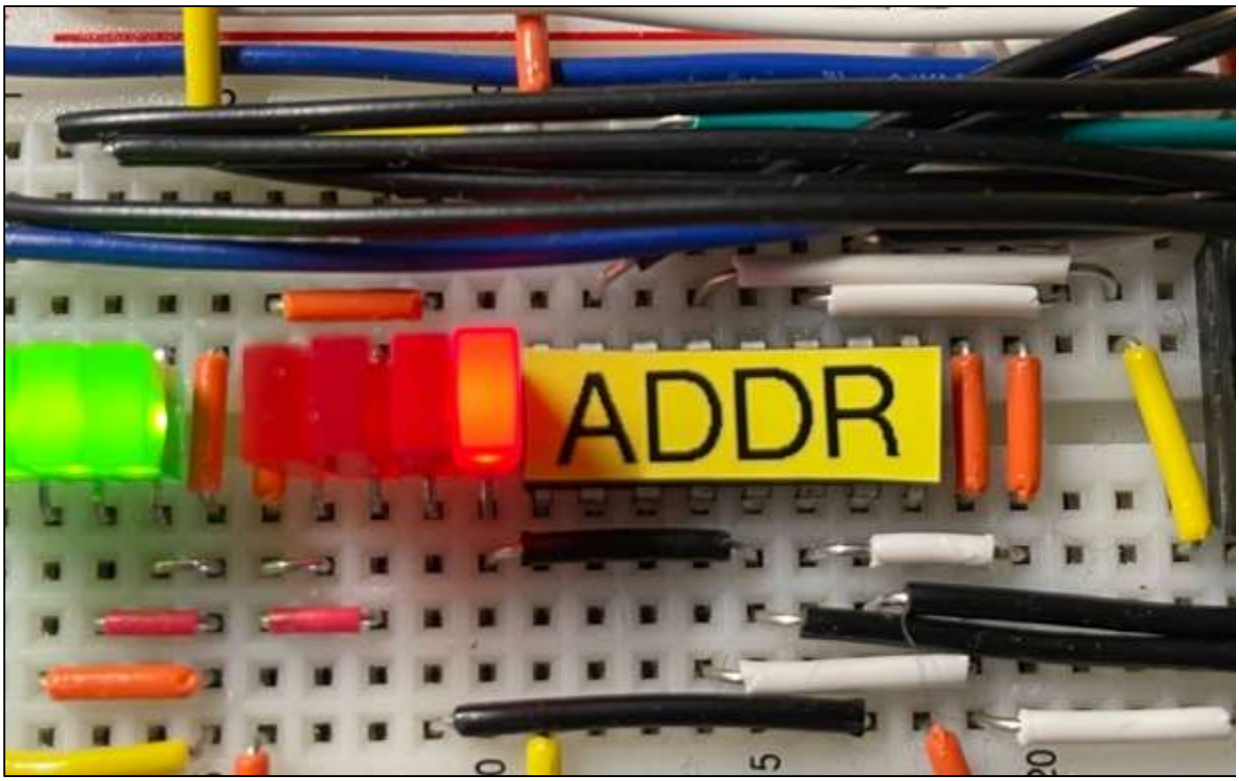
The SN74LS174 is fairly similar to the Accumulator in the fact that it is also a flip-flop. The only big differences are that it is a hex flip-flop instead of an octal, and there is no enable pin.



Pin 16 is connected to Power, and pin 8 is connected to GND. Pin 9 is a clock input pin, and pin 1 can be connected to power because master reset is not needed.

Inputs D0 to D3 are connected to the multiplexer output which is also connected to the Program Counter inputs. Input D4 is connected to the I/O-0 pin on the Control EEPROM. The Ouputs Q0 to Q3 are connected to the A0 to A4 on the Ram chip. Output Pin Q5 will connect to the write enable pin on the Ram chip. This will tell the ram chip if it is being written to, or being read from.

Media



Address Wiring

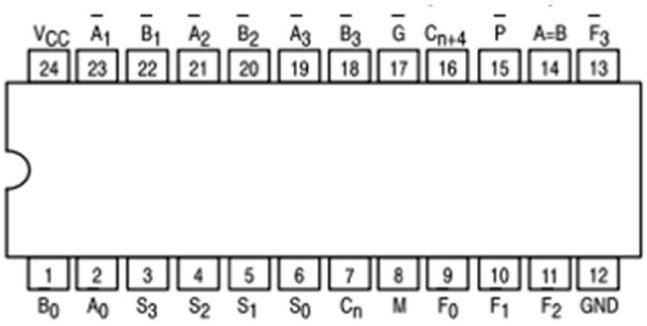
Project 3.2.9. ALU

Purpose

The ALU is the Arithmetic Logic Unit and is used as the main part of the CHUMP. This will take care of all of the calculations.

Procedure

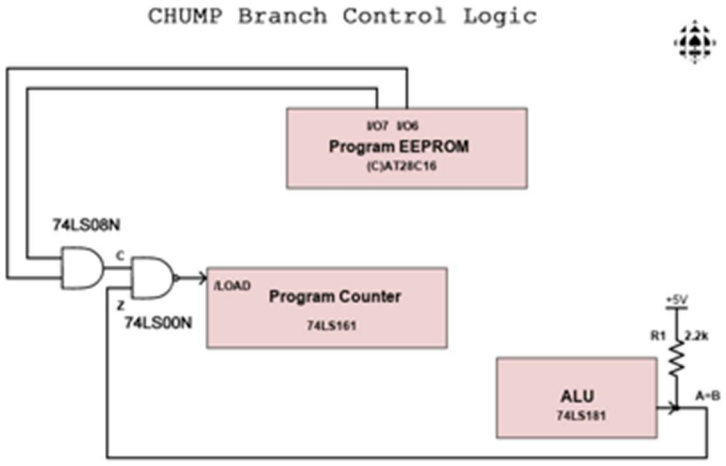
The SN74LS181 is a 24 pin 4-bit Arithmetic Logic Unit, this chip will provide all logic operations of two variable inpus.



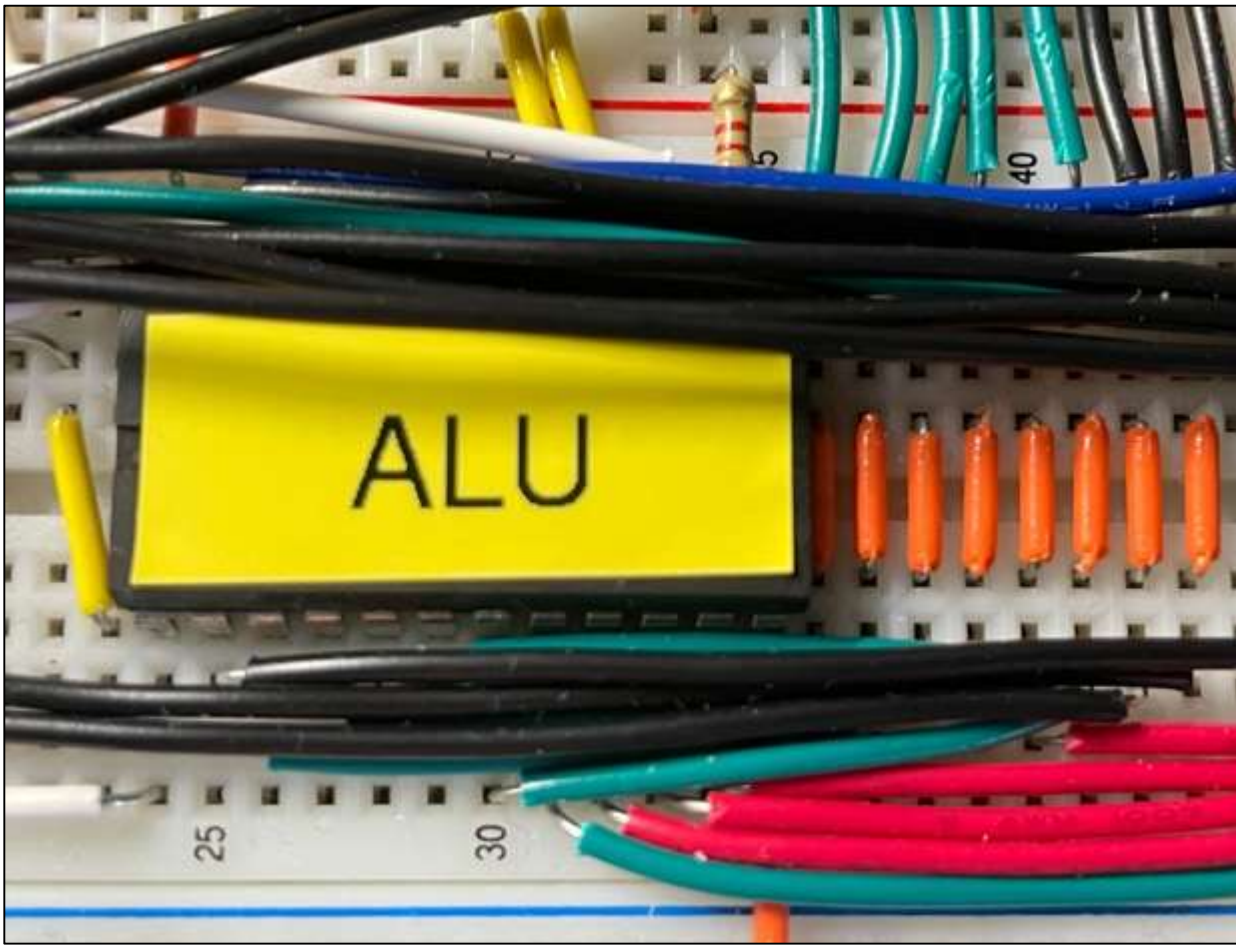
The ALU has 2 sets of 4 inputs labeled A0-3 and B0-3, 4 select inputs labeled S0-3, and 4 outputs labeled F0-3. The A inputs are connected to the outputs of the Accumulator which also goes into the inverter before Ram. The B inputs are connected to the Multiplexer outputs which also connect to the Program Counter and the Address. The 4 outputs on the ALU connect to the inputs on the Accumulator.

Pins 24 and 12 are connected as normal to Power and Ground respectively. Pin Cn also known as the Carry Input is connected to the I/O-2 pin on the Control EEPROM. Pin 8 also known as the Mode input is connected to the I/O-3 pin on the Control EEPROM. Pin 16 and 17 are both grounded.

The big pin on the ALU is the A=B pin. This pin is used as a comparator output pin. It will be connected to Vcc through a 2.2kΩ resistor. The A=B pin is used for the `GOTO` and `IFZERO` commands. This will be used along with an AND gate output from the I/O-6 and I/O-7 pins on the Program EEPROM to give a value that will only be low when both the Program EEPROM outputs are high. This will feed into the Load pin on the Program Counter. This will allow the value that feeds from the multiplexer to go through the Program Counter and change the line of code the program is on. When the two Program EEPROM outputs are both HIGH, it will produce a HIGH output on the AND gate which connects to an input on the NAND gate. Since the A=B pin is always HIGH, the HIGH of the AND gate and the HIGH from the A=B pin into the NAND gate will produce a LOW and the load pin on the Program Counter is active when low.



Media



ALU Wiring

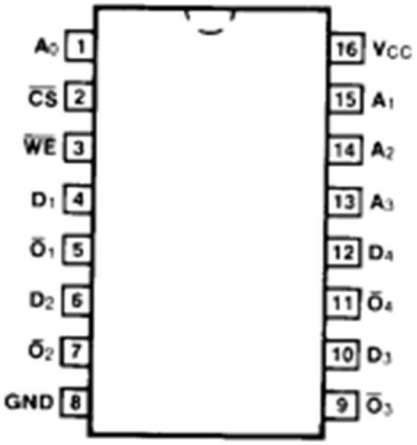
Project 3.2.10. RAM

Purpose

The purpose of the 74LS189 Ram chip is to be used as a storage system for the CHUMP. This will allow values to be stored into Ram to be used for later.

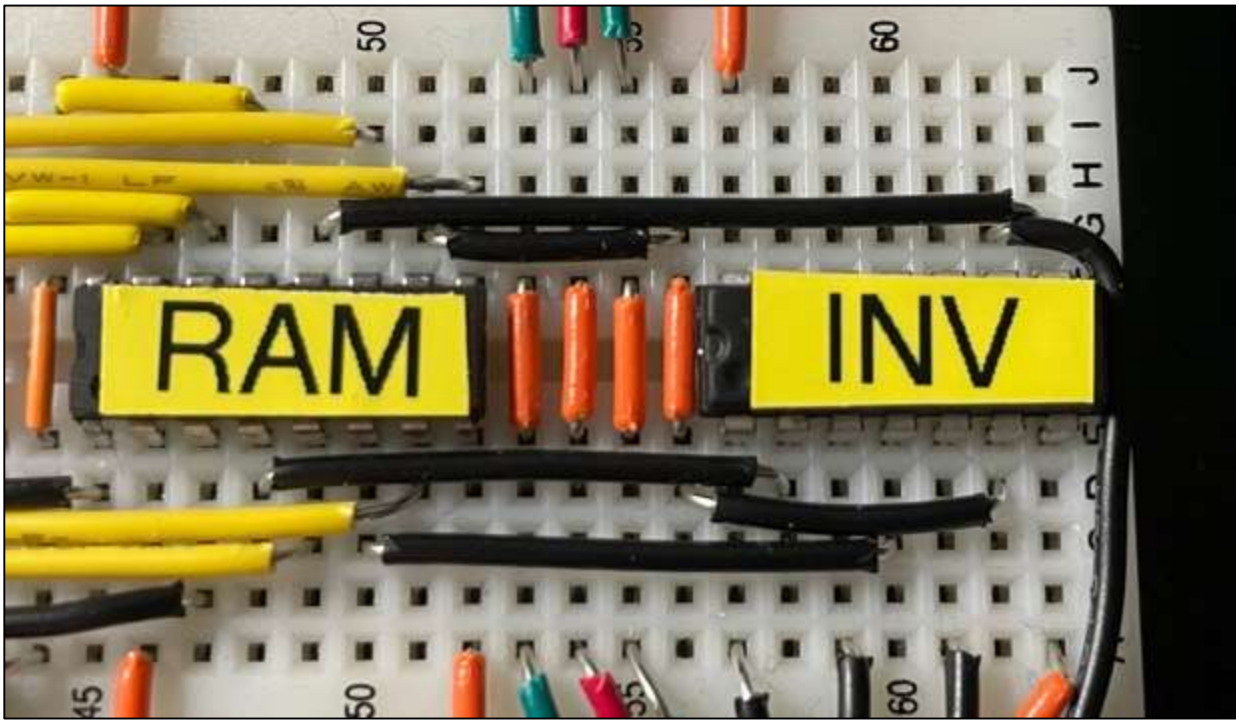
Procedure

The 74LS189 is a 64-bit Random Access Memory (Ram) chip. It is used for memory storage.



The Ram chip has the normal Vcc and GND on pins 16 and 8. This chip has 4 Address Inputs named A0-3. It also has 4 data inputs and 4 data outputs named D1-4 for the inputs and O1-4 for the outputs. The Ram chip also has a Chip Select Input on pin 2 and Write Enable on pin 3. Both are active low. The A0-3 pins will take inputs from the output of the Address chip Q1-4. The Data inputs will be coming from the Output of the Inverter that switch the Accumulator outputs. The Data outputs will go to the Multiplexer to be used as a Selection between the `Const` and `Mem` options. The Chip Select pin on the chip will be grounded because we want the chip always selected. The Write Enable pin on the Ram will be connected to the 5th output on the Address chip which will be HIGH for reading from Ram, and LOW for writing to Ram.

Media



Ram Wiring

Project 3.2.11. Code (Revisited)

Purpose

The purpose of Revisiting the Code that was written in the first stages is to check it over and make sure it is correct. Once it is correct, it can be loaded into the Program EEPROM.

Procedure

With the first version of this code, a fatal part was missing at line 8. Without this line of code, it would have not put the value of S into the Accumulator which would have not worked. Once this is added back in, it will work correctly.

Code

High Level	Address	Instruction	CHUMP level	Comment
`s = 10;`	`0000`	`0000 1010`	`LOAD 10`	`accum = 10; pc++`
	`0001`	`0110 0000`	`STORETO s`	`mem[0] = accum; pc++`
`c = 15;`	`0010`	`0000 1111`	`LOAD 15`	`accum = 15; pc++`
	`0011`	`0110 0001`	`STORETO c`	`mem[1] = accum; pc++`
`while c != 0;`	`0100`	`1110 0111`	`IFZERO 7`	`accum==0?pc=7: pc++`
`c = c - 5;`	`0101`	`0100 0101`	`SUBTRACT 5`	`accum -= 5; pc++`
	`0110`	`1100 0011`	`GOTO 3`	`pc = 3; pc++`
	`0111`	`1000 0000`	`READ s`	`addr = 0; pc++`
	`1000`	`0001 0000`	`LOAD s`	`Accum = mem[0]; pc++`
`while s != 0;`	`1001`	`1110 1100`	`IFZERO 12`	`accum==?pc=12: pc++`
`s--;`	`1010`	`0100 0001`	`SUBTRACT 1`	`accum -= 1; pc++`
	`1011`	`0110 0000`	`STORETO s`	`mem[0] = accum; pc++`
	`1100`	`1100 0010`	`GOTO 2`	`pc = 2; pc++`
`pause;`	`1101`	`1100 1101`	`GOTO 13`	`pc = 12; pc++`

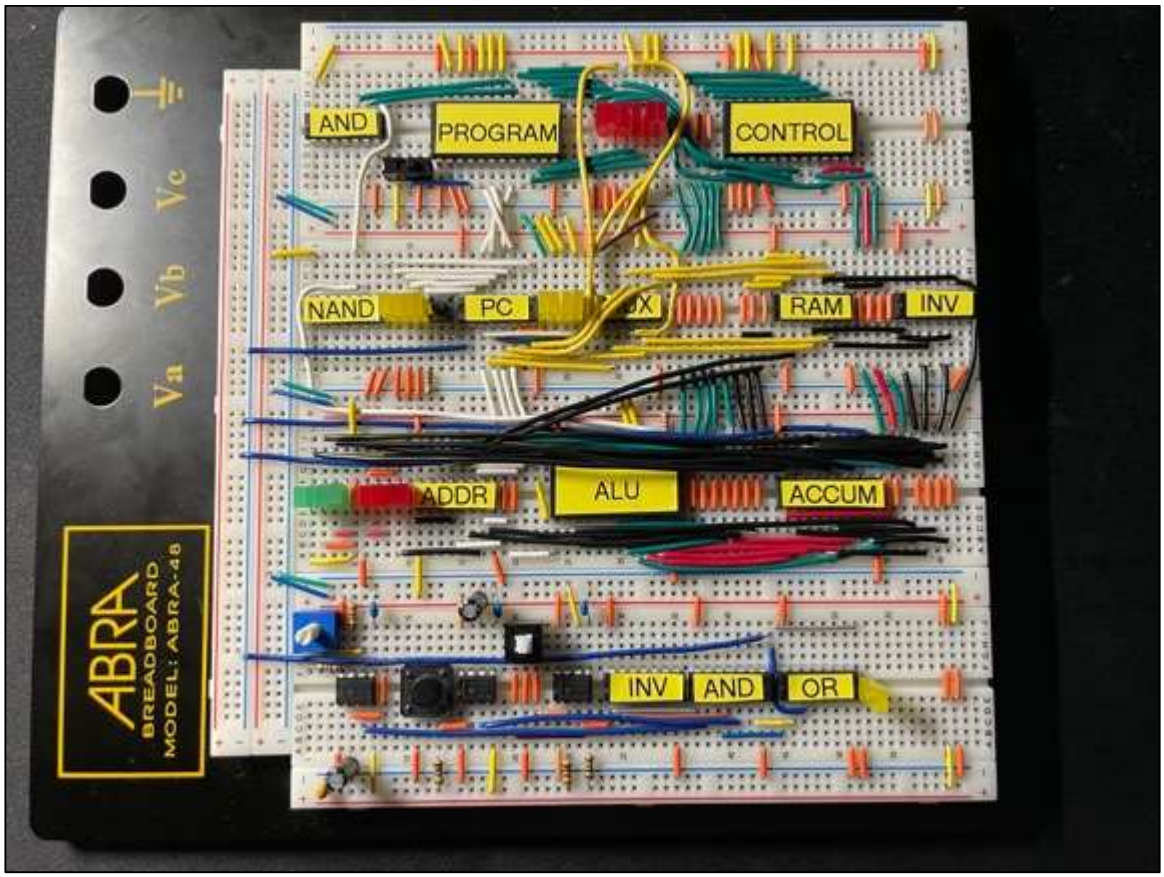
Project 3.2.12. CHUMP Final

Reflection

The CHUMP project was a very difficult project. It takes a lot of time to research and figure out how to wire up each part ot make it work. I found it very interesting to work through building the CHUMP and making sure it worked. I had one big error which stumbled me for a long time, when I tried to run any code with a GOTO command it wouldn't work. This ended up being because when the Program EEPROM has 2 output LEDs on the highest two outputs, it drops the voltage that goes into the AND gate below the value it can read which made it never produce a HIGH which didn't let the Load pin on the Program Counter activate. Once I figured this out after many hours of going through my wiring and talking with my peers, I finally figured out the problem. Once it finally worked it was amazing to see it in action. Working Flawlessly. This satisfaction of completing the CHUMP after many hours of wiring, testing, and debugging, made the whole build worth it and will make me very happy to put in my case for the Short ISP.

Media

YouTube Video: https://youtu.be/RCO_DvRfsBA



Fully Assembled CHUMP

Project 3.3. Short ISP

Purpose

The purpose of this ISP was to improve knowledge on CNC milling and design.

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/2021/ISPs.html
https://sienci.com

Background

The first part of this ISP is to understand what a CNC machine is. CNC stands for Computer Numerical Control and is the automated control of machining tools, some of the most common CNC machines are 3D printers, pasma cutters, mills, routers, and lathes. A CNC machine will take in control instructions called G-Code. A design will be created in a CAD software like Fusion 360 and then can be exported as G-Code.

For this project, a CNC router will be used to create a case for the previous CHUMP project. A CNC router is a machine that usually holds a hand held router and can be used to cut a variety of materials such as wood, aluminium, steel, plastics, foam, and acrylic. The CNC machine in this project is from Sienci Labs, a Waterloo based company, and the CNC machine is the 30"×30" model, it uses a ¼ hp Makita router, and is a 3-axis design.

Procedure

The first step in this project is the design. The final design uses a combination of wood, acrylic, and 3D printed plastic. The main part of the cases is made from Pine wood. The cover for the front of the CHUMP is made out of acrylic. Finally the 3D printed parts included are a plaque, and corners.

The wood is designed with a concave center and raised corners. On the back of the case, the DES full logo has been cut out. The front of the wood part has been cut out to leave an indented space for the whole CHUMP to sit in as well as an outer ring for the friction fit acrylic. The acrylic that sits on the front of the case is a box shape that can sit just outside of the CHUMP edges. On the side of the acrylic, there is a small whole cut out so a power jack insert can sit inside of it. This will allow power to go to the CHUMP without having to take of the acrylic. Finally on the front of the acrylic, the RSGC logo, ACES, and CHUMP logo are cut out. The final part of the case design is the 3D printed parts. On each corner of the acrylic cover, there are small 3D printed corner covers to allow the case to sit on anyside. The second 3D printed part is a small multi-material plaque with all the names of the 2020-2021 ACES class.

The next step in the project is knowing what the machine is capable of doing with the items on hand. This project uses a variety of different machining bits to create the final design. There are many different bits available and they all have their upsides and downsides. The wood backing on the case used 4 different bits. The first bit is a ½ inch ball mill. This allows the machine to take away most of the wood while keeping fairly close to the final design's concave profile. The

next bit used is a very small 1/32 inch tapered ball. This bit has a small ball at the very bottom and gradually widens to be ¼ inch. This bit allowed the machine to make the back very detailed and smooth. On the other side of the wood, a 22" surfacing bit was used to take out a small warp that had apperead after the back side was done. This bit made the front very smooth and allowed the cut that was done with a 1/8 inch bit to produce a very good holes with very little varience between the sides.

The acrylic is a bit more difficult than the wood. There are two different ways acrylic can be made, it can either be cast or extruded. The cast version produces much better and cleaner results than the extruded. The router spinning very fast as well as the movement speed on the machine produced a lot of heat on the bit and caused the extruded acrylic to melt and form on the bit ruining the first attempt. To reduce heat, the router speed was reduced from 22,000 rpm to about 15,000 rpm. The machine movement speed was also reduced from 1000mm/min to 250mm/min. Instead of working on extruded acrylic, it was switched to Cast acrylic. Fianally the bit was switched from a normal wood 1/8 inch bit to a 1/8 inch aluminum bit. This aluminum bit only has one cutting edge instead of the normal two. Once these had been changed, the final acrylic produced very nicely without any flaws.

The 3D printed parts were done on the Prusa I3 MK3S with the mmu2s attachment to allow for multiple colors to be used. The corners on the acrylic were printed in gold PLA, and the plaque was done in black and gold PLA.

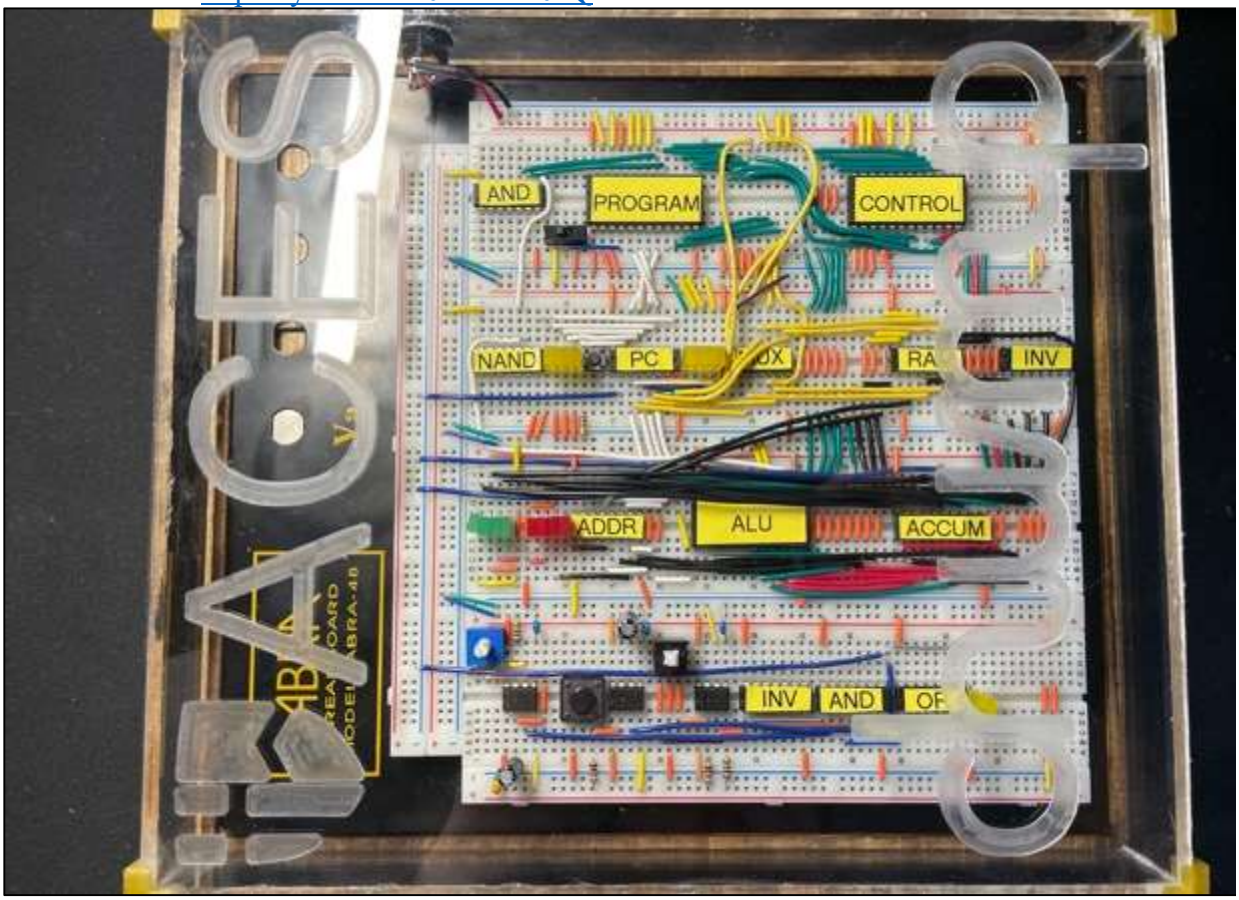
To finish the wood, a single coat of Winmax Aged Oak gel stain was applied. The letters and ACES logo was left unstained to make them pop against the much darker final stain. Behind where the CHUMP sits, a rectangular magnet was glued into place, this is to make sure when the case is on its sides, the CHUMP won't fall to the front of the acrylic. The acrylic was glued together using E6000 glue. The powerjack went into the side with a friction fit and wires were connected to it that can be plugged and unplugged from the CHUMP board. Finally the 3D printed corners and plaque were glued onto the acrylic.

Reflection

This project has allowed me to learn a lot of different skills that I can use for future CNC machining. I had to learn how to write g-code to be able to fix my program to stop it from cutting the final product. I now know how to machine using Fusion 360 which opens up a whole new variety of projects that are possible. I learned that acrylic can be a massive pain if you don't have the right settings and even the right product. This project has pushed me very far out of my comfort zone, and I have learned so much from it. I can't wait for my next big project on my CNC machine and what I can learn from it.

Media

Youtube Video: https://youtu.be/v024aCz10cQ



CHUMP Case Front

CHUMP Case Back with Plaque

Project 3.4. Keypad Matrix Echo

Purpose

The purpose of this project is to develop low level coding skills, while improving knowledge on the ATtiny84.

Reference

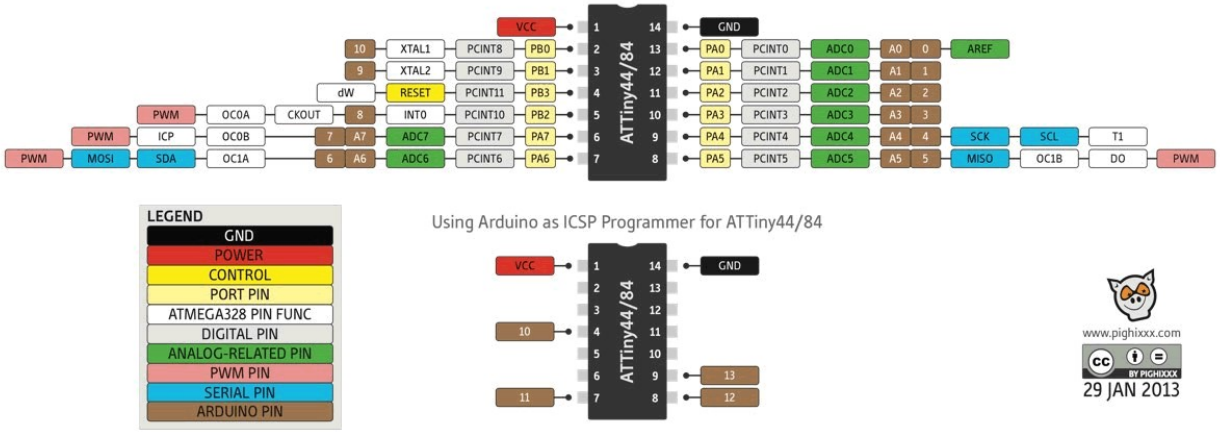
http://darcy.rsgc.on.ca/ACES/TEI4M/2021/Tasks.html#KeypadlMatrixEcho
https://github.com/SpenceKonde/ATTinyCore

Procedure

The first step in this project is to hook up the ATtiny84 to the SN74HC595 shift register, the 8×8 LED matrix, and the keypad. The keypad that is used is a 8 pin keypad. It has a common pin that can be grounded or connected to power, and it has 3 column and 4 row pins. This will be the input to the project and will be mapped out on the matrix.

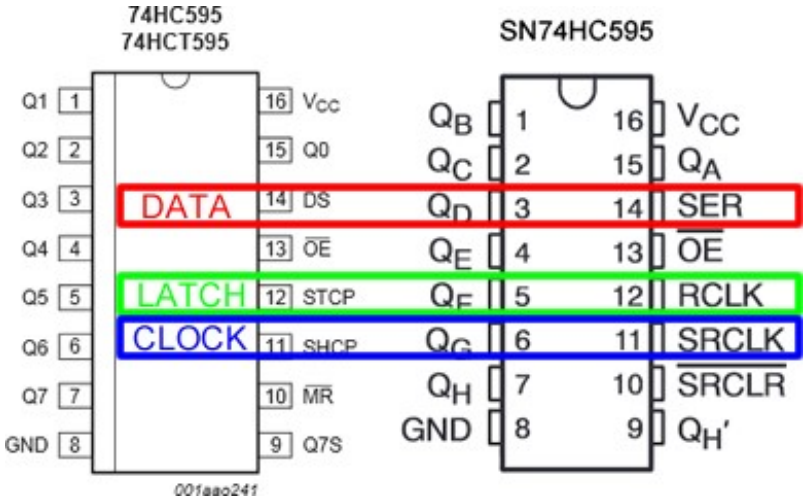
Parts List	
Component	Quantity
ATtiny84	1
SN74HC595	1
8×8 LED matrix	1
8 pin keypad	1

The ATtiny84 is a 14 pin micro controller that has two different ports, `PORTA`, and `PORTB`. `PORTA` has 8 pins that can be inputs or outputs, and `PORTB` has 4 pins that can be inputs or outputs. To program the ATtiny84, pins 4, 7, 8, and 9 can either be connected to a programmer or to the Arduino directly. For this project we will connect it to the Arduino. Pins 1 and 14 can be connected to the power rails on the side of the breadboard. Pins, 4,7,8, and 9 can be connected to pins 10, 11, 12, and 13 respectivly. To be able to program the ATtiny84, first the Arduino has to know it is a programmer. To do this, in the Arduino IDE, we can load the example sketch ArduinoISP, and then upload that code with the Arduino UNO as the board. Once that is uploaded, to use the ATtiny84, a new board called ATtiny84Core. Once that is installed, the board can be switched to ATtiny84 and the programmer has to be changed to Arduino as ISP. Now the ATtiny84 can be programmed by the Arduino.



For the wiring of the project, 7 of the `PORTA` pins will be connected to the 7 keypad pins, and 3 of the `PORTB` pins will connect to the shift register. The 3 column pins from the keypad are connected from `PA0` to `PA2` with the rightmost column on `PA0`. The 4 row pins from the keypad

are connected from `PA3` to `PA6` with the top row connected to `PA3`. The common pin on the keypad will be grounded. The first `PORTB` pins will be used to connect to the shift register, data, latch, and clock will be connected to pins `PB0`, `PB1`, and `PB2` respectivly.



On the shift register, pns 8 and 16 will are connected to ground and power respectively. Pin 13, output enable, is active low, so it will be connected to ground. Pin 10 or master reset is connected to power since it is also active low and we don't want it to be reset. Pin 9 doesn't have to be connected to anything since it is only used when there are more than one shift register. The output pins Q_0 or Q_A to Q_6 or Q_G are the only ones that will be used since there are 4 rows and 3 columns. The row pins are wired from the bottom to top going from Q_0 to Q_3. The column pins are wired from left to right going from Q_4 to Q_6.

To setup the code for the ATtiny84, the data, latch, and clock pins have to be set to output, and `PORTA` has to be set to input pullup. In register-level, output is 1, and input is 0. The default is set to 0. Setting data, latch, and clock to output, can be done through the command `DDRB |= (1<<PB0) | (1<<PB1) | (1<<PB2);`. This says that on `PORTB`, a 1 will be shifted left by, 0 or 1 or 2, which will set them to output. Input pullup can be done by just setting `PORTA` to high. The way this is done is through `PORTA = ~DDRA;`. This says that `PORTA` is the inverse of the `pinMode` of `PORTA` pins, by default the `pinMode` is 0 so inverse will make it 1, or `HIGH`. To shift out the input from the keypad to the matrix, we need a mask that changes some of the bits so they are `HIGH` instead of `LOW`. This mask in binary is `0b1111000`, and will be `XOR` with the reading from `PORTA` that can be done with `PINA`. Once that is done, it will be `AND` with a `1<<bits`, where bits is a value that goes from 0 to 7. This whole value then will be `NOT` and compared to `LOW`. If that value was `LOW`, it will set the data pin to `HIGH`, but if the value was `HIGH`, the data pin would be `LOW`. Once that happens, a clock pulse will be sent by setting the clock pin `HIGH` and then `LOW`. This whole function will be called `joeShift();`. In `void loop()`, the latch pin can be set `LOW` by using `PORTB &= ~(1<<PB1);` then the function `joeShift();` can be called. Then the latch will be set `HIGH` by using `PORTB |= (1<<PB1);` after the latch is set to LOW, a delay of 300ms can be put in with delay(300);. This will finish off the code needed and when run should turn on one LED on the matrix respective to the keypad press for 300ms and then turn it off.

Reflection

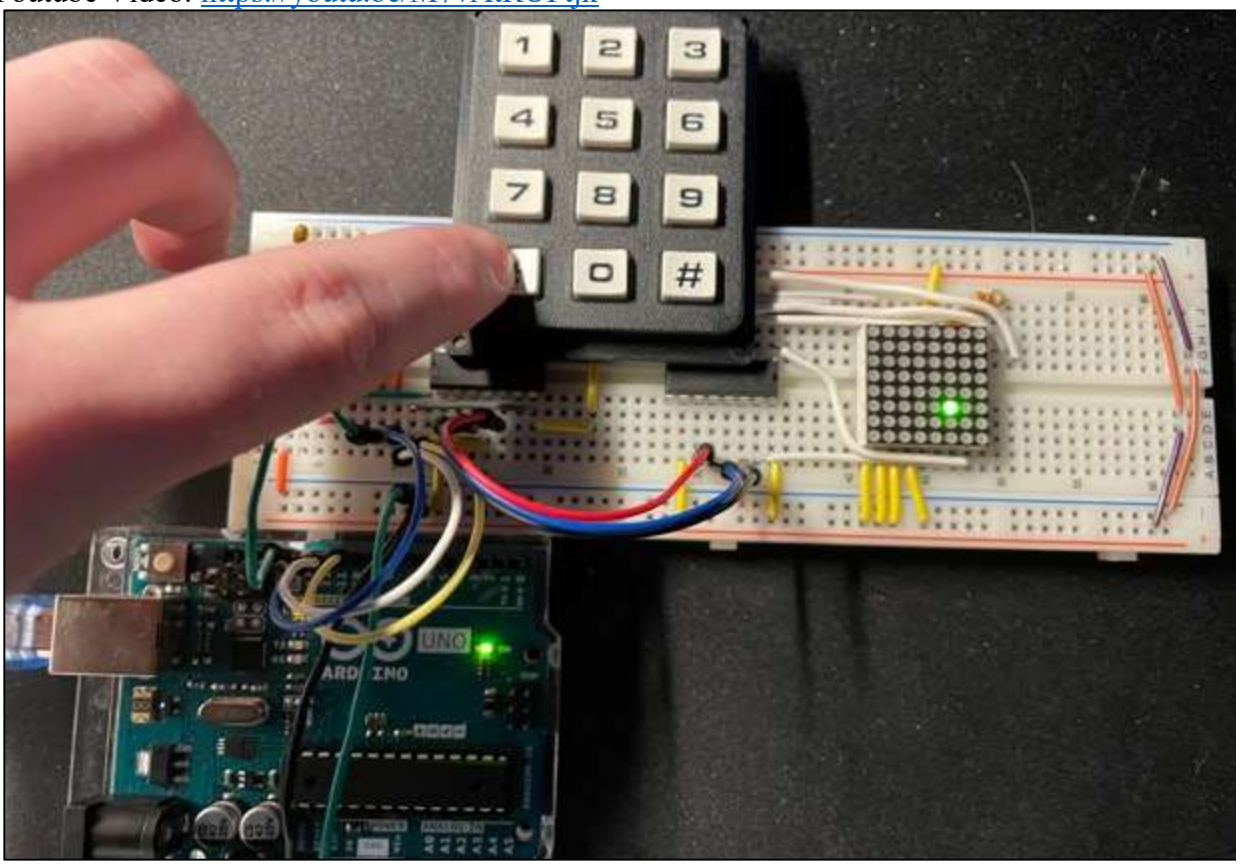
This project was a good introduction into the ATtiny84 and register level code. I had a few troubles with my Sparkfun Programmer in the fact that it was broken, but I was able to wire up my Arduino UNO and use that as my programmer. I can't wait to come back in 2021 to work on the DDP project.

Code

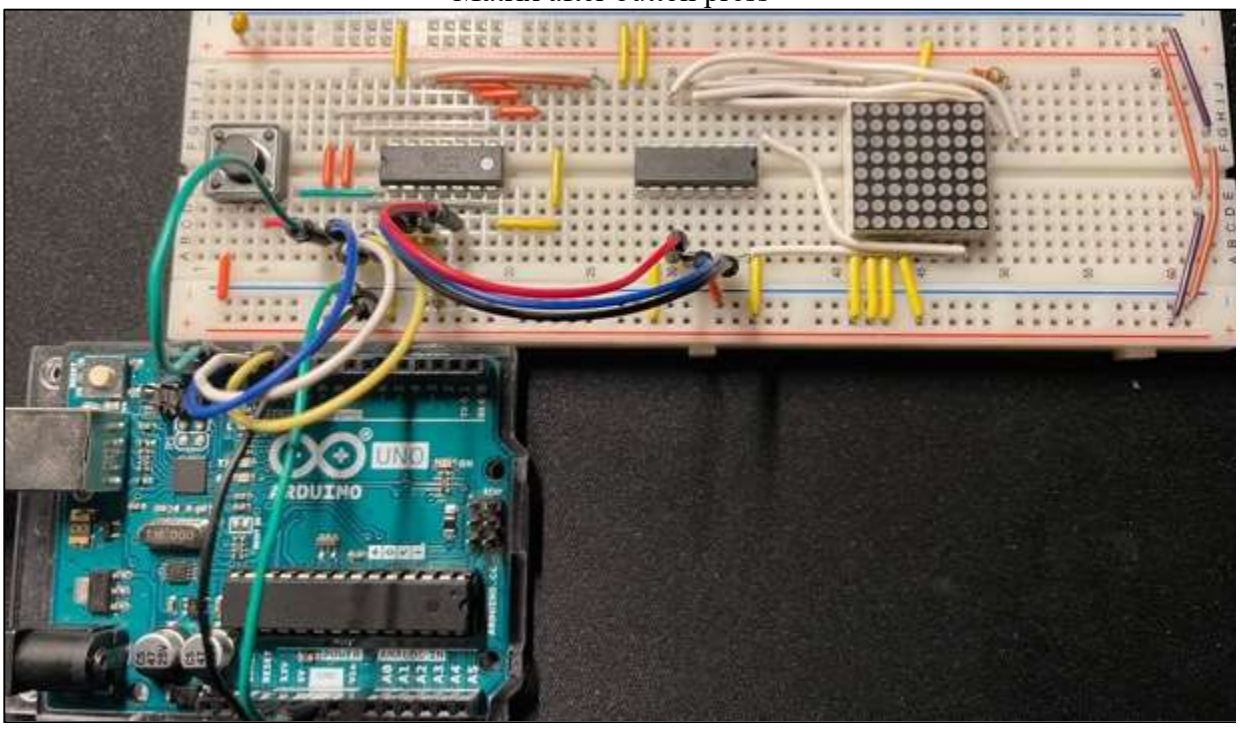
```
// Project: Kepad Matrix Echo
// Purpose: To map keypad inputs 1:1 on a matrix
// Course : ICS4U
// Author : J. Vretenar
// Date   : 2020 12 19
// Status : Working

#define DATA PB0  //data pin
#define LATCH PB1 //latch pin
#define CLK PB2   //clock pin

#define MASK 0b1111000 //mask to change input bits 3-6

void setup() {
  DDRB |= (1 << DATA) | (1 << LATCH) | (1 << CLK);  //data, clock, latch OUTPUT
  PORTA = ~DDRA;    //sets keypad pins to INPUT_PULLUP
}

void loop() {
  PORTB &= ~(1 << LATCH); //latch LOW
  joeShift();
  PORTB |= (1 << LATCH);  //latch HIGH
  delay(300);
}

void joeShift() {
  for (uint8_t bits = 0; bits < 7; bits++) {
    if (!((PINA ^ MASK) & (1 << bits)) == LOW) {
      PORTB |= (1 << DATA);   //data HIGH
    } else {
      PORTB &= ~(1 << DATA);  //data LOW
    }
    PORTB |= (1 << CLK);      //clock HIGH
    PORTB &= ~(1 << CLK);     //clock LOW
  }
}
```

Media

Youtube Video: https://youtu.be/M7vAtRUFtjk



Matrix after button press



Wiring

Project 3.5. Pin Change Interrupt

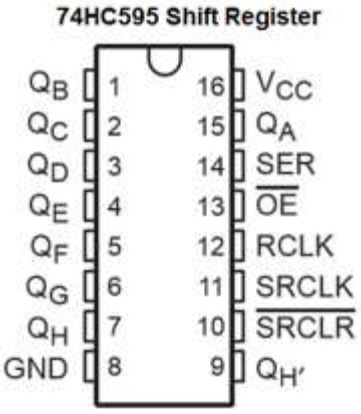
Purpose

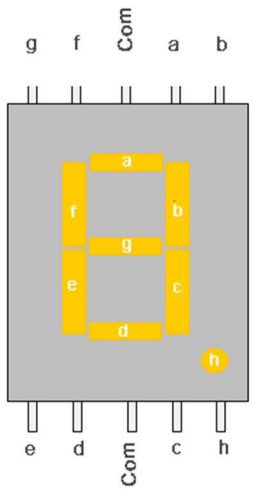
The purpose of the project is to introduce interrupt service routines into low level code. It is also the first project using the Dolgin Development Board.

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/2021/Tasks.html#PinChange
https://mail.rsgc.on.ca/~cdarcy/Datasheets/ATtiny84.pdf

Procedure





To start this project, the DDBv6 will have to be soldered and assembled. Once this is done a button debouncer should be assembled on its PCB. Then on a breadboard, a SN74HC595, 7 segment display, button debouncer, as well as a reset button. The SN74HC595 can be wired up with pins 16, and 8 to VCC and GND respectively. Data pin 14, latch pin 12, and clock pin 11 can be connected to PA5, PA4, and PA3. Output enable pin 13 is connected to GND to allow for output. The reset pin 10 will be connected to one side of the reset button which is also connected to power. The other side of the reset button will be connected to GND so that when the button is pressed it will activate the reset on the 595 and reset the count. The outputs of the shift register will connect to the 7 segment display with the first output connecting to pin G on the 7 segment and the second last output connecting to pin A.

The button debouncer will be connected to GND and VCC on their respective pins. The output on the capacitor side will connect to PB1. Pressing the button will advance the count by 1 until it reaches 15, then it will reset to 0. Enabling the data, latch, and clock as output can be done with `DDRA |= 1 << DATA | 1 << LATCH | 1 << CLK;`. To enable the button to be used as an ISR can be done with 3 lines. `PCMSK1 |= 1 << PCINT9;`, `GIMSK |= 1 << PCIE0 | 1 << PCIE1;`, and `GIFR |= 1 << PCIF1 | 1 << PCIF0;`. `PCMSKx` is setting which pin will be enabled as a pin change in the register. `GIMSK` enables interrupt on both 1 and 0, and finally `GIFR` sets the pin change interrupt flag to 1 and 0. Setting up the interrupt service routine can be done with the command `ISR (PCINT1_vect) {}`: Inside of the curly brackets, the `volatile boolean` value `triggered` which was set to `false` before setup will be set to `true`. This will allow for an `if` statement to detect when `triggered` is `true` and then execute the command to count up on the 7-segment display. This if statement will say `if (x >= 15) {x = 0;} else {x++;}` This command says that if x is equal or greater that 15, set the count to 0, but if it is not then add one to x. Then to display the numbers, the same `outShift` function that was created in the last project can be used except this time there is no mask to only select a

3×4 matrix. The data that will be feed through is a `segmentMap[]` consitsting of the binary values to display 0 to F on the 7-segment display.

The advanced part of this task is using a rotary encoder to go both up and down the binary count. The rotary encoder has 5 different pins, there are VCC, GND, A, B, and SW. VCC and GND can be connected to VCC and GND respectively. The SW pin is the connection for when the rotary encoder is pushed down. This can connect to the reset on the 595. The A and B pins are used to create a map of which position the rotary encoder is in. The encoder has two different states, a detante, and the middle state. The detante can have two different results from A and B. Both pins can either produce a 1 or a 0. The middle states will depend on which way the encoder is going, but will consist of one 0 and one 1. To read which way the encoder is going, there needs to be a comparison between the previous detante state and the middle state. The A and B pins from the encoder will be connected to PA0, and PA1.

To initiate the pins for interrupt, the line `PCMSK0 |= 1 << PCINT1 | 1 << PCINT0;`. This will say that in the A register, PA0 and PA1 are set for interrupt. To setup the ISR the command with `ISR (PCINT0_vect) {}` will be used. Inside of the curly brackets, PINA will be masked with 3 to only take the encoder pins, after that the if statement `if ((state == 0b11) || (state == 0b00) {rState = state; triggered = true;} else {mState = state}`. This statement says that if the current state of the pins is either 3 or 0, execute the command, detante state is equal to state and then make triggered true. If the state is not 3 or 0, then middle state will equal the current state. These values of state, `mState`, and `rState` will be used in a function called rotation which will add the `rState` and `mState` together. If this equals 3 or 7, that means the count should decrease. If it equals anything else, the count should increase. This will then change the x value correspondingly and execute the `outShift` function.

Reflection

This project was pretty difficult to execute. Getting the button working was pretty easy with a bit of fiddling around, but getting the encoder to work and adding the button to that was a lot more difficult. It was an interesting project because once you figure it out, it seems so simple but actually putting it into low level code proved fairly challenging.

Code

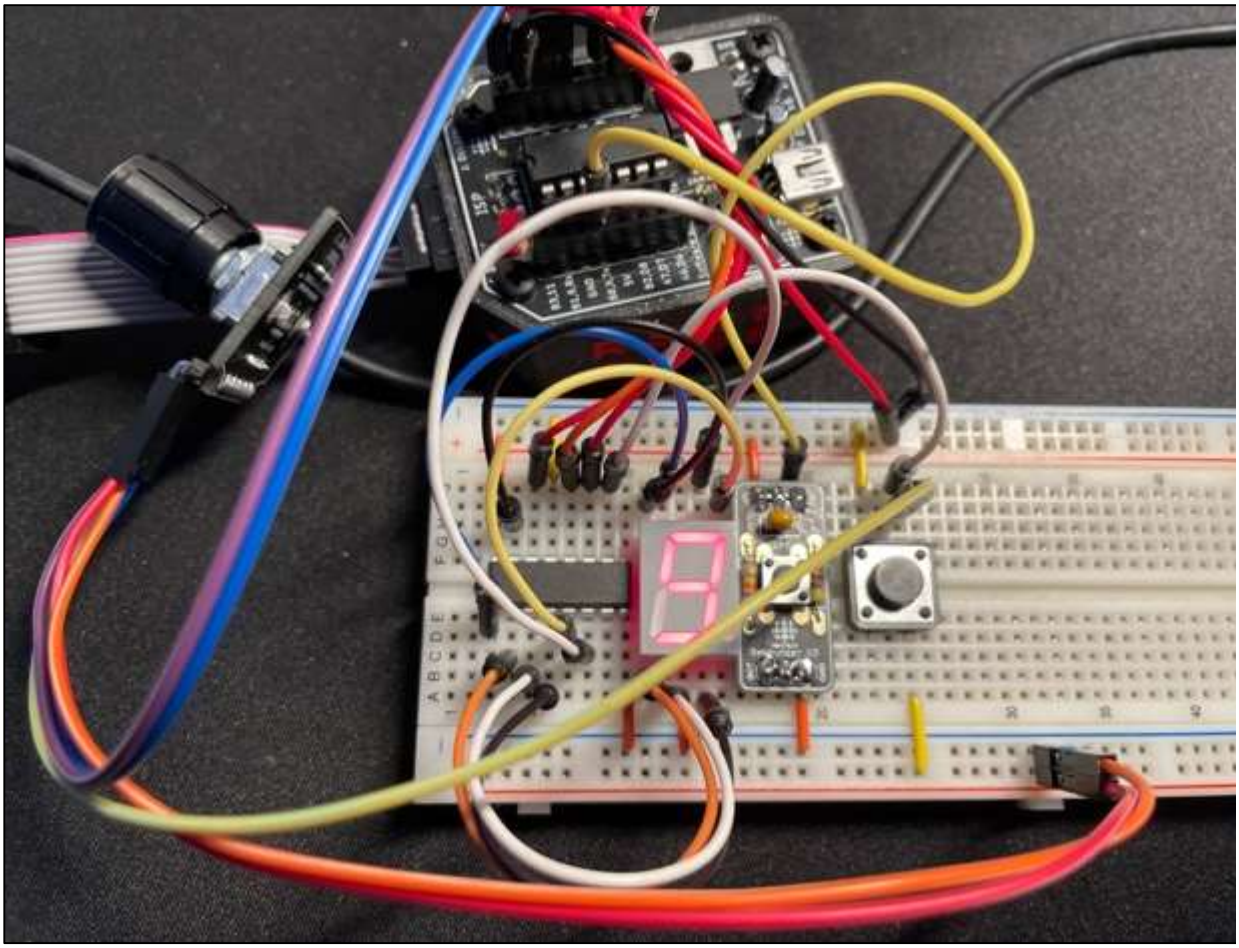
```
// Project: PinChangeInterrupt with button
// Purpose: To use a button to count up from 0 to F
// Course : ICS4U
// Author : J. Vretenar
// Date   : 2021 01 30
// Status : Working

#include "DDBv6JV.h"

volatile boolean triggered = false;  //boolean value for button
uint8_t x = 0;  //value changed by both button and encoder

#define DATA PA5  //set DATA to PA5
#define LATCH PA4 //set LATCH to PA4
#define CLK PA3   //set CLK to PA3

void setup() {
  PCMSK1 |= 1 << PCINT9;                //Masks for interrupts
  GIMSK |= 1 << PCIE0 | 1 << PCIE1;     //General Interrupt Mask
  GIFR |= 1 << PCIF1 | 1 << PCIF0;      //General Interrupt Flag

  DDRA |= (1 << DATA) | (1 << LATCH) | (1 << CLK); //set pins to OUTPUT

  PORTA &= ~(1 << LATCH);               //Latch low
  outShiftJV(DATA, CLK, segmentMap[x]); //set segment to 0
  PORTA |= (1 << LATCH);                //Latch high
}

ISR(PCINT1_vect) {  //ISR for the button
  triggered = true;
}

void button() {   //increasing x for button push
  if (x >= 15) {
    x = 0;
  } else {
    x++;
  }
}

void loop() {
  if (triggered) {  //if triggered = true execute this command
    button();
    triggered = false;
    PORTA &= ~(1 << LATCH);
    outShiftJV(DATA, CLK, segmentMap[x]);
    PORTA |= (1 << LATCH);
  }
}
```

```
// Project: PinChangeInterrupt with Encoder
// Purpose: To use a rotary encoder to go both up and down from 0 to F
// Course : ICS4U
// Author : J. Vretenar
// Date   : 2021 01 30
// Status : Working

#include "DDBv6JV.h"

volatile uint8_t state;  //the current state for the encoder
volatile uint8_t rState; //the resting state for the encoder
volatile uint8_t mState; //the middle state for the encoder
volatile boolean triggered = false;  //boolean value for button
volatile boolean triggeredE = false; //boolean value for encoder
uint8_t x = 0;  //value changed by both button and encoder

#define DATA PA5  //set DATA to PA5
#define LATCH PA4 //set LATCH to PA4
#define CLK PA3   //set CLK to PA3

void setup() {
  PCMSK0 |= 1 << PCINT1 | 1 << PCINT0;  //Masks for interrupts
  PCMSK1 |= 1 << PCINT9;
  GIMSK |= 1 << PCIE0 | 1 << PCIE1;     //General Interrupt Mask
  GIFR |= 1 << PCIF1 | 1 << PCIF0;      //General Interrupt Flag

  DDRA |= (1 << DATA) | (1 << LATCH) | (1 << CLK); //set pins to OUTPUT

  PORTA &= ~(1 << LATCH);               //Latch low
  outShiftJV(DATA, CLK, segmentMap[x]); //set segment to 0
  PORTA |= (1 << LATCH);                //Latch high
}

ISR(PCINT0_vect) {  //ISR for the Encoder
  state = (PINA & 3);
  if ((state == 0b11) || (state == 0b00)) {
    rState = state;
    triggeredE = true;
  } else {
    mState = state;
  }
}

ISR(PCINT1_vect) {  //ISR for the button
  triggered = true;
}

void rotation() {   //how to figure out the rotation
  if ((gray[rState] + gray[mState] == 7) || (gray[rState] + gray[mState]
== 3)) {
    if (x == 0) {
      x = 15;
    } else {
      x--;
    }
  } else  {
    if (x >= 15) {
```

```
      x = 0;
    } else {
      x++;
    }
  }
}

void button() {   //increasing x for button push
  if (x >= 15) {
    x = 0;
  } else {
    x++;
  }
}

void loop() {
  if (triggeredE) { //if triggeredE = true execute this command
    rotation();
    triggeredE = false;
    PORTA &= ~(1 << LATCH);
    outShiftJV(DATA, CLK, segmentMap[x]);
    PORTA |= (1 << LATCH);
  }
  if (triggered) {  //if triggered = true execute this command
    button();
    triggered = false;
    PORTA &= ~(1 << LATCH);
    outShiftJV(DATA, CLK, segmentMap[x]);
    PORTA |= (1 << LATCH);
  }
}
```

Media

YouTube Video: https://youtu.be/8JITT5LI1RI



Wired up encoder, button, and DDB

Project 3.6. Medium ISP

Purpose

The purpose of this project is to connect an Arduino to the wifi using an ESP8266 wifi module then read from an API, go through the data and display the weather on a matrix.

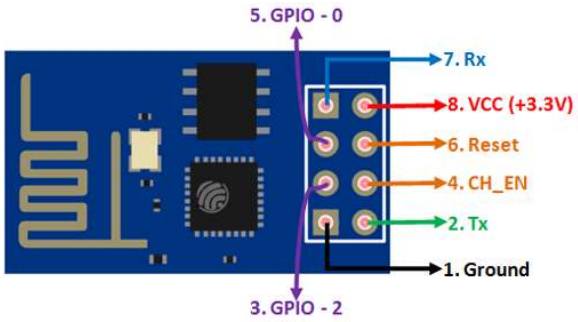
Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/2021/ISPs.html#logs
https://openweathermap.org
http://arduino.esp8266.com/stable/package_esp8266com_index.json

Procedure

This project is going to use the matrix that was on the CN Tower Project. This matrix is controlled using a single data pin and a power and ground pin. The final animation will be displayed on the matrix and will be taking data from Toronto.

To get the data that is to be displayed, an API is needed. The API used in this project is from https://openweathermap.org, it is a current weather forecast API and when used will give weather, temperature, wind speed, pressure, and humidity. This project will only be using the weather data.



To access this data, the ESP8266 module will be used. This is a small wifi module that will be programmed using a FTDI serial adapter. The ESP8266 has 8 pins. Pins 1 and 8 are connected to ground and power respectively. The reset pin can be connected to a button that when pressed will connect to ground. To program the ESP8266, the Rx and Tx pins will connect to the Tx and Rx pins on the programmer respectively. Finally when programming, the GPIO-0 pin has to connect to ground. Once all the wires are connected the ESP8266 is able to be programmed. The Arduino IDE can be used to program the ESP8266, to do this a board has to be added in the board manager using this link. Once that is done, if we go into the boards manager and search for ESP8266, there should be one that was made by the ESP8266 community and that also has to be installed. After these are installed, the example code under ESP8266 called blink can be run. Once it is uploaded the reset button can be pressed and the blue LED on the ESP8266 should be blinking.

To display the animations on the matrix, the neoPixel, neoMatrix, and neoGFX libraries are just like in the CN Tower Project. This will be controlled using one pin on the Arduino. To display the animations, the function `matrix.drawPixel();` will be used. This function allows a specific pixel to be selected and have a color set to it. Once every pixel is selected in the animation and each given a color. The matrix can display the pixels using `matrix.show();`. To finish off the animation, the pixels will be set in a different position to show the lights moving. If it is raining, the rain will be falling as the animation. If it is sunny, the sun beams will

move. Making the animations will require two functions, one for the 1st state and one for the 2nd state. Finally a 3rd function should be added to put the two together with a delay.

To read and end up display the weather outside, the API has to be called on. Calling the API for a specific place can be done through a few different ways, it can be done through a city and country code, a postal code, or the way that is done in this project, lat and long. This will give an accurate reading at the CN Tower. The website that is used for the data requires 3 parts, the base of the website is http://api.openweathermap.org/data/2.5/weather?(lat)&(long)&appid=(API key). This is calling from the website where the API key was gotten earlier, and the 3 parts that have to put in are the lat, long, and API key. The lat and long at the CN Tower according to google maps are lat = 43.6425662 long = -79.3892455. These values can be put into the lat and long parts in the main website, finally the API key from open weather map can be put into the API key place. The final link is here. This website provides a JSON file with values that have to be parsed through. The weather data is in `"weather"["id"]` and can be translated on the website to what the weather is. Once this data is gathered, the animations can be displayed with a simple `if` statement asking if the `"id"` is a certain number.

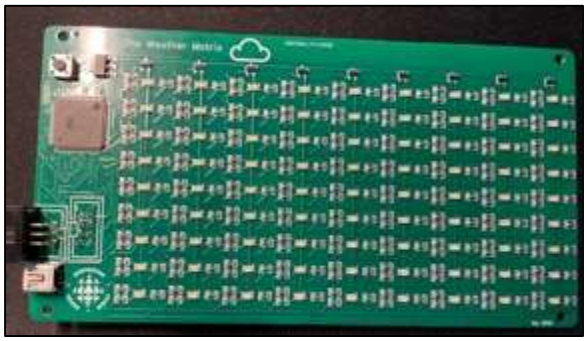
Reflection

I had quite a few problems with this project with the main one being I couldn't read from my ESP8266. The ESP was the main part of the project and having it not work means the display doesn't work unless it is changed manually. I did end up learning a lot about the ESP8266 and especially about EasyEDA during this project. When I was creating my PCB I used EasyEDA because I was getting surface mount parts from JLC and EasyEDA works well with JLC. The big problem with EasyEDA is their auto router, it created so many shorts when I tried it, which meant I had to manually route everything on my PCB. Reasearching about the ESP also showed me that there are better modules which are pretty much Arduinos with built in Wifi, if I were to do this project again I would have used an ESP like that.

Media

YouTube Video: https://youtu.be/XtSLApKJjrQ

Matrix Animation



PCB

Project 3.7. DDP: Legacy Shield

Purpose

The purpose of the Legacy shield project is to create a shield for the Dolgin Development Platform that can be used in future years of the ACES Program.

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/2021/DDPv6.html#legacy
https://github.com/NINJOE-1/LegacyShield

Procedure

The main parts of this project will be a servo motor and a SN74HC595 bargraph combination. The PCB will consist of 3 male headers for the servo motor, 3 female headers for the variable resistor, 10 DIP for the bargraph, 8 DIP for the 595 shift register, and the 2 1×8 male headers to connect to the DDP.

The servo motor uses 3 pins, VCC, GND, and PWM. The VCC and GND pins will be connected to VCC and GND respectively, and the PWM pin will connect to PB2 on the ATTiny84. The motor signal can either be adjusted through code and outputted on the B2 pin. The other way to send an output to the servo motor would be through the variable resistor headers. These headers are connected in a way that a potentiometer can be set in and will act as a voltage divider with the OUPUT going to PA0.

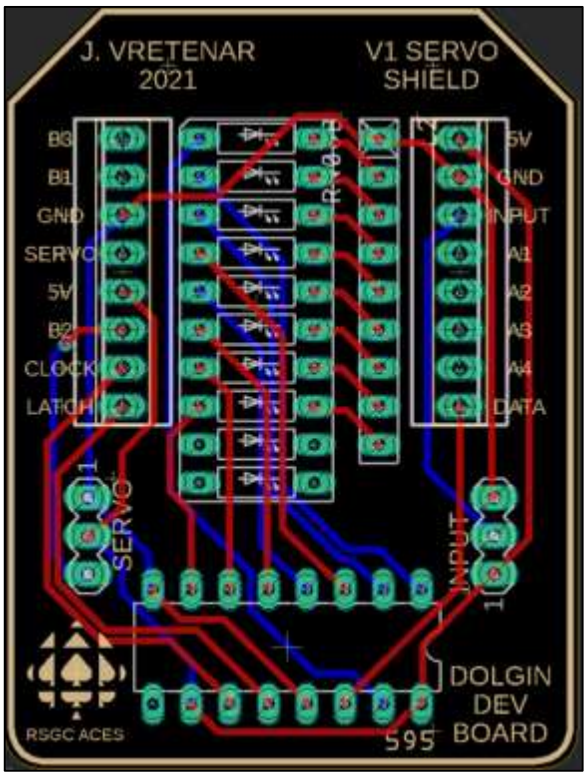
The board will allow the ATTiny84 to read the INPUT of PA0 and use analog to digital conversion to change it for the servo motor. The INPUT at PA0 will also be used for the bargraph display. This display will count up in binary the amount the servo is moving. This is controlled by the 595 shift register. The data pin will also be the analog to digital conversion from PA0. The latch pin is connected to PA6, and the clock pin is connected toe PA7. The outputs of the shift register are connected to the LED bargraph, and then connected to a resistor bus, which connects to ground.

Reflection

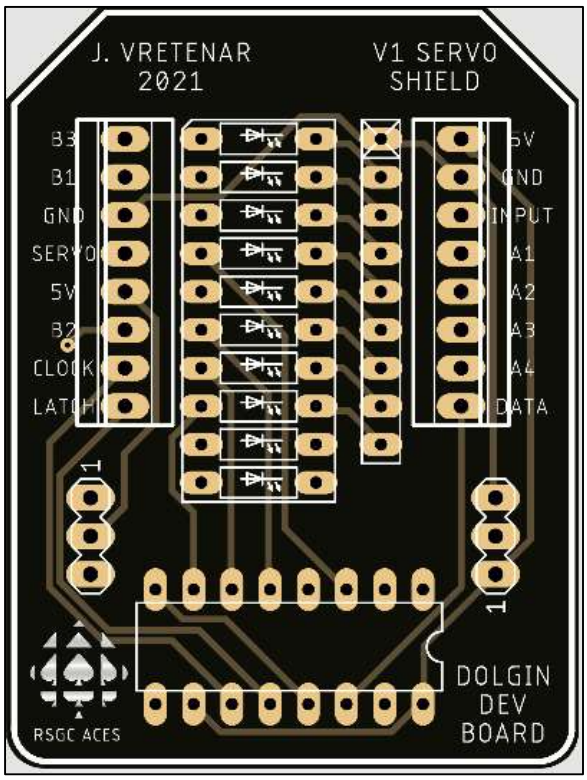
I thought for quite a while about what to do for my legacy PCB. I ended up choosing the Servo motors because I was using them in my long ISP and thought that a board which included the servo motors and an input for my resistors would be pretty cool.

Media

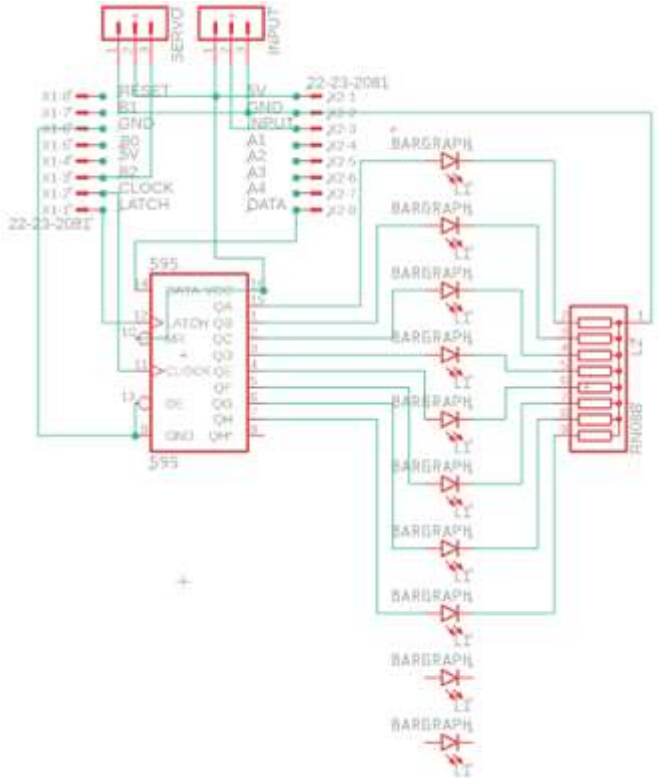
YouTube Video: https://youtu.be/w9Z9tspAXVQ



Eagle PCB



JLCPCB rendering



Schematic

Project 3.8. Long ISP

Purpose

The purpose of this project is to create a bionic hand using servo motors and flex sensors, as well as create the model of the hand from scratch.

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/2021/ISPs.html#LongISP

Procedure

This project has two main parts which are 5 SG90 servo motors, and 5 Sparkfun flex sensors. The end goal in this ISP is to take a reading from the flex sensors and translate it to rotation of the servo motors.

The flex sensor used in this project consists of 2 pins. This sensor acts just like a variable resistor. As the sensor bends it will produce a greater resistance. To read this resistance, a voltage divider can be contructed at the end of the pins to create an output. This voltage divider ends up making the flex sensor very similar to a potentieometer. This two pin configuration will end up with 3 pins through the voltage divider. One pin on the flex sensor will connect to a 10kΩ resistor so the output will never be a full 5V. This pin is also the output pin from the voltage divider. The other side of the resistor is connected to ground and the other pin on the flex sensor is connected to VCC. When plugging the voltage divider output into an Arduino and printing the analog value, the resistor ranges from 680 to 760. This will be very useful later.

The next big part of the project is the Servo motors. The SG90 servo motor used in this project contains 3 pins, VCC, GND, and PWM. The orange wire is PWM, red is VCC, and brown is GND. This can be hooked up to their respective pins on the Arduino. The servo motors can be tested with the included example in the Arduino IDE called sweep which sets the servo motor to 0° and will turn in 180° and then back again.

To make the connections between the servo motors and flex sensors easier, a PCB can be constructed. There are two different PCBs for this project, one for the servo motors and one for the flex sensor. The servo motor PCB will take all 15 pins from the servo motor and transfer it into the 5 PWM pins and then 1 VCC and 1 GND pin. This reduces the number of wires needed coming out of the servo motors and into the Arduino. The flex sensor PCBs will be individual for each flex sensor and change the 2 pin flex sensor into a 3 pin voltage divider. The flex sensor is connected using surface mount soldering pads, and the output will be 3 right angle male headers.

The bionic hand is designed in Fusion 360. The fingers are a dual color design and are printed in black and gold. The fingers have 2 hold that will be used for keeping them straight and then another hole for the servo motor string. The fingers are kept straight using stretchy shoe laces, because they are thin enough to fit through as well as not being to strong. The fingers are pulled by the servo motors with some cotton string.

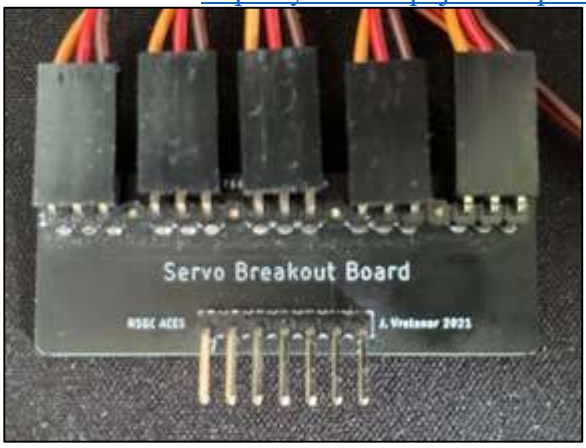
The fingers are moved based off of the analog value from the flex sensors, but the reading from the voltage divider provided a reading from 680 to 760. This is not a range of 0 to 180. The code will convert this analog reading to the angle using the `map();` command which takes 4 inputs. The inputs in the brackets are `(650, 1000, 180, 0);`. If the mapping of the servo motor was set from 680 and 760 to 0 and 180, it would rotate the servos too far and the servos also move back and forth from the slightest movement in the flex sensors. The motors can be moved using a `for` loop and `arrays` for the servo motors. This will move the servos in a loop based off of the pins they are connected to.

Reflection

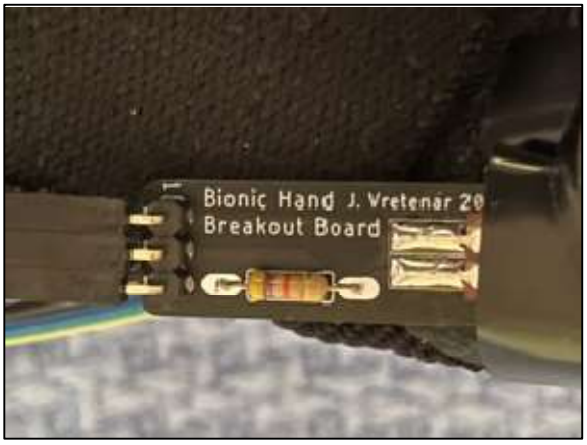
This project went very well over time I had to work on it. I am very happy with how it turned out, and think it represents my time in the ACES program well. It combines multiple aspects of the program into one final project. It is quite suprising to think that this is the end of my time at RSGC and the ACES program, though I think the work I have done in the ACES program will help me throughout University and the rest of my career.

Media

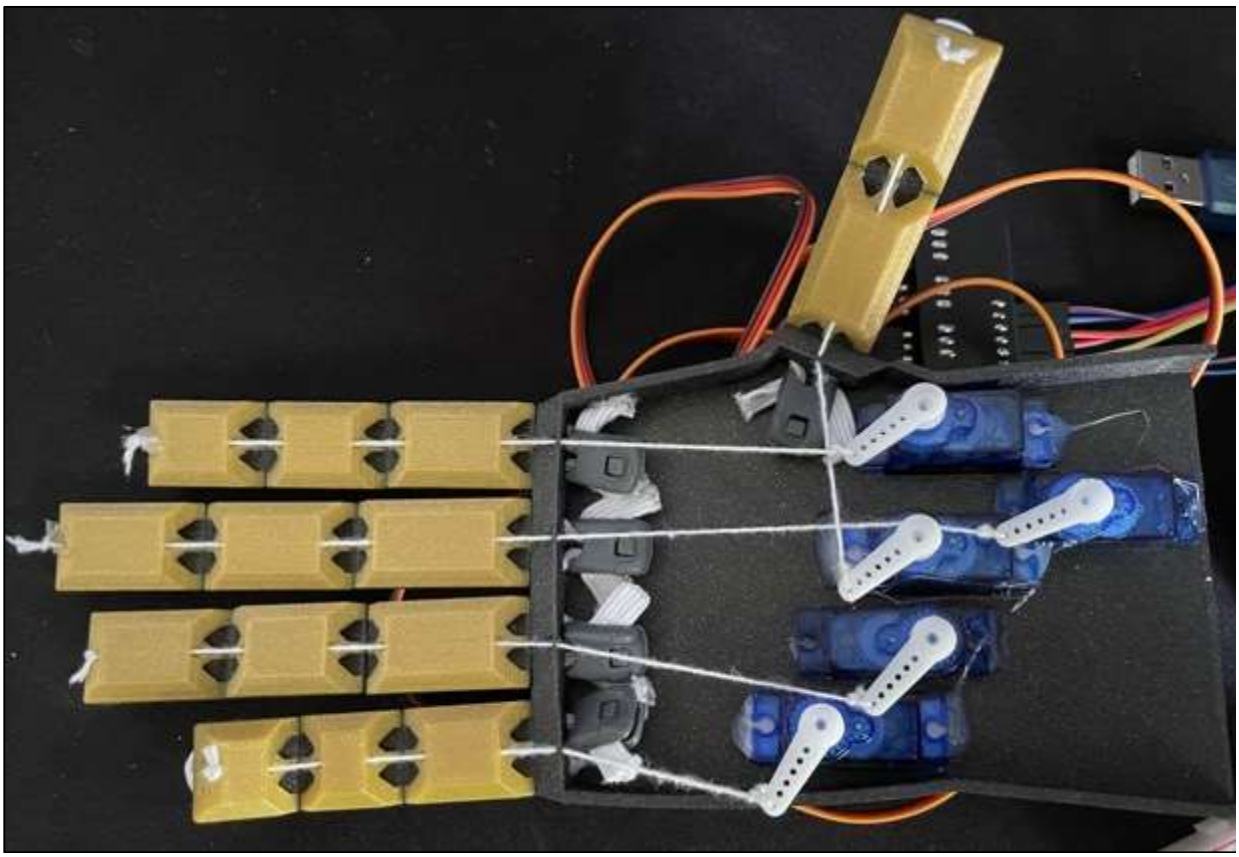
Youtube Video: https://youtu.be/phj9G0zSpMY



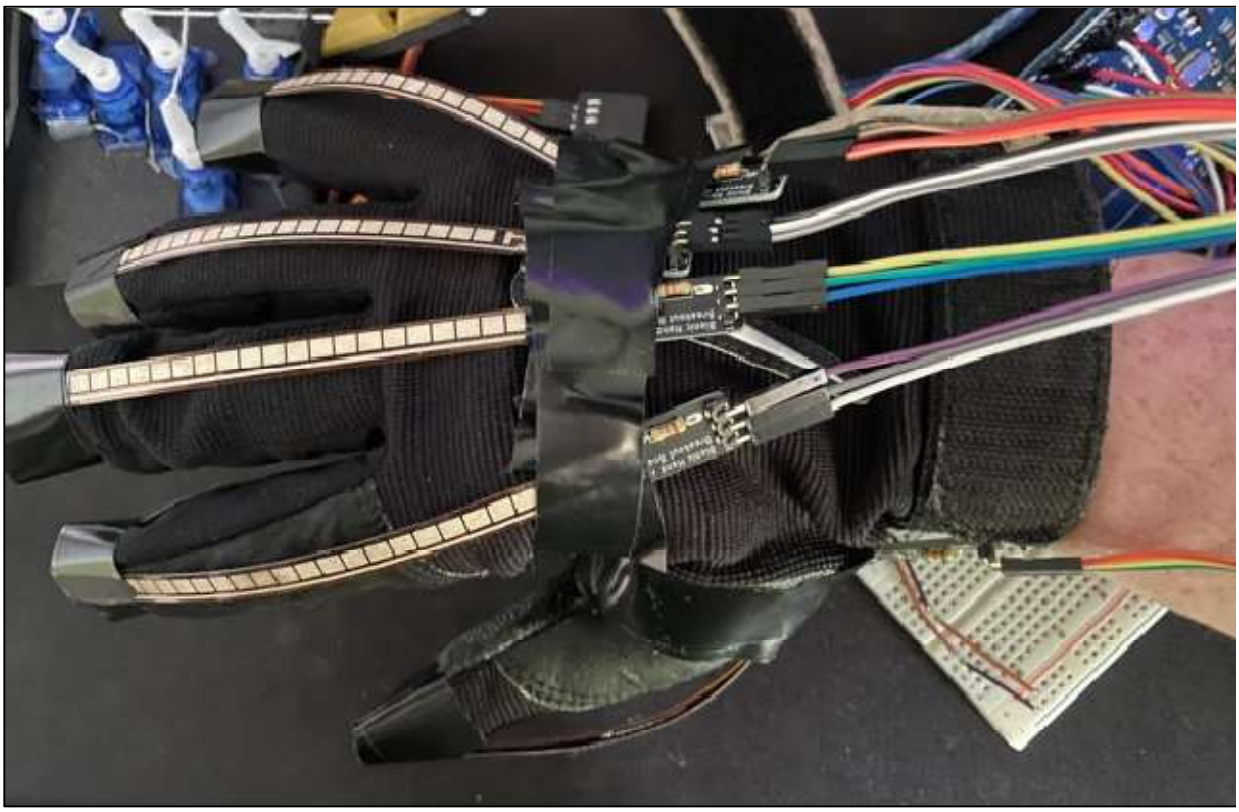
Servo Breakout Board



Flex Sensor Breakout Board



Bionic Hand



Control Glove

Code

```
// Project: Long ISP Bionic Hand
// Purpose: To create a bionic hand using servos and flex sensors
// Course : ICS4U
// Author : J. Vretenar
// Date   : 2021 05 29
// Status : Working
// Reference: http://darcy.rsgc.on.ca/ACES/TEI4M/2021/ISPs.html#LongISP

#include <Servo.h>  //Servo Library

#define SERVOS 5    //# of Servos in use

Servo myServo[SERVOS];  //Set up for servos

uint8_t servoPins[SERVOS] = {9,8,7,6,5};        //pins the servos are on
uint8_t fingerPins[SERVOS] = {A0,A1,A2,A3,A4};  //pins for flex sensors
uint16_t angleValue[SERVOS];                    //reading from flex sensors

void setup() {
  //attach all the servos to their respective pins
  for(uint8_t i = 0; i < SERVOS; i++) {
    myServo[i].attach(servoPins[i]);
  }
}

void loop() {
  //for each servo motor
  for(uint8_t i = 0; i < SERVOS; i++) {

    //read the analog value of the flex sensor
    angleValue[i] = analogRead(fingerPins[i]);

    //map that value from 650 - 1000 to 180 - 0
    angleValue[i] = map(angleValue[i], 650, 1000, 180, 0);

    //set the angle of the respective servo to the angle of the flex sensor
    myServo[i].write(angleValue[i]);

    //wait a little
    delay(15);
  }
}
```