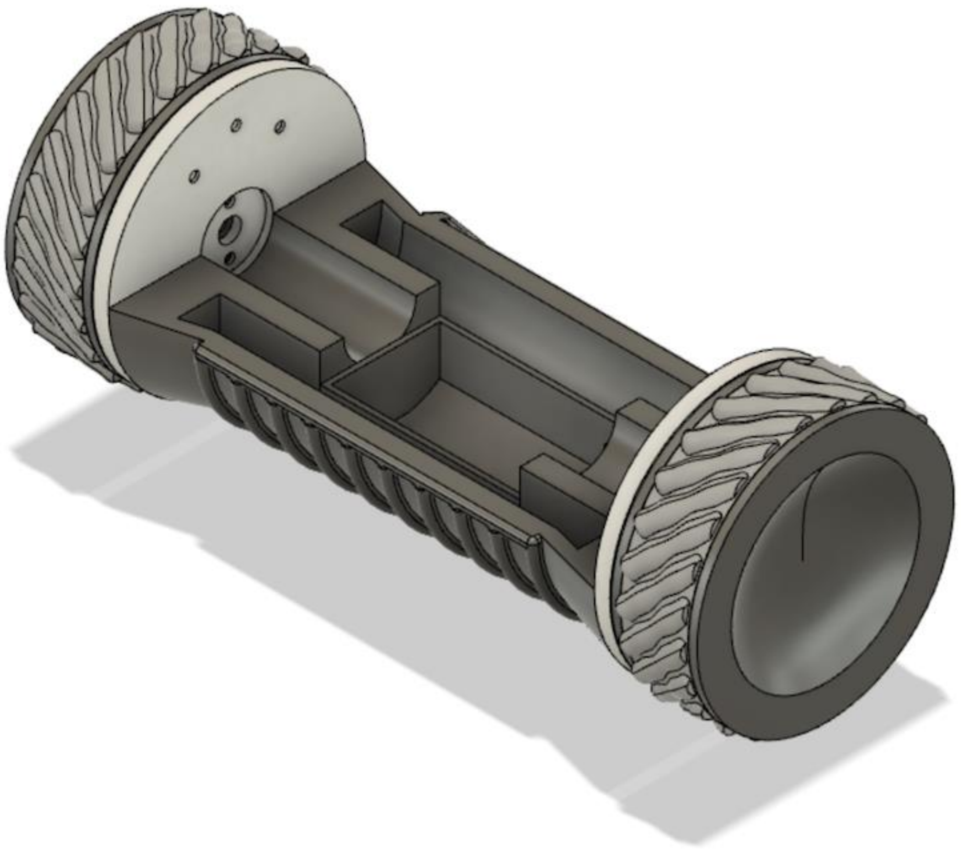
Design & Engineering Report



Course:	Computer Information Technology
Code:	TEL3M, ICS3U, ICS4U
Author:	Josh Dolgin

TABLE OF CONTENTS

Project 1. Voltage Dividers

Reference

http://darcy.rsgc.on.ca/ACES/TEL3M/1718/TasksFall.html - VoltageDividers

Theory

The theory behind this project is to use voltage dividers and variable resistors to change the flow of electrons through a bi-color LED, therefor changing its color. Firstly, what is a voltage divider? According to Kirchhoff's voltage law, the sum of the current into a component is equal to the sum of the current out of that component, and the sum of the voltage through a closed loop is equal to zero. As well as Ohm's Law, I = V/R or Current = Voltage/Resistance. Using these two separate laws, voltage dividers can be established as two or more resistors in series that each take half of the remaining voltage. The first crucial component of the circuit is the breadboard, a breadboard is an easy way to lay out a circuit without soldering the parts together. By using vertically connected rows and easy to access power supply, it is a crucial piece of equipment for prototyping circuits. Next, the variable resistor, also known as a potentiometer. The variable resistor is a resistor of a certain value that can be changed simply by turning the dial on top. Next, the bi-color LED, the way that a bi-color LED is different from a regular LED is that it allows current to flow in both directions as opposed to regular diodes, in one direction the light is green and the other direction the light turns red. After learning about these separate components, it is time to talk about how they all come together in the circuit. The theory behind this circuit is that with the turn of the potentiometer, current will have two paths to go through, choosing the path which offers less resistance to the flow of electrons. Voltage dividers prove useful because they split the current in half when the potentiometer dial is at half way, so the LED appeared off because the voltage is split directly in half, but as the dial was turned, the resistance changed, and there for the bi-color LED would light up one of two colors.

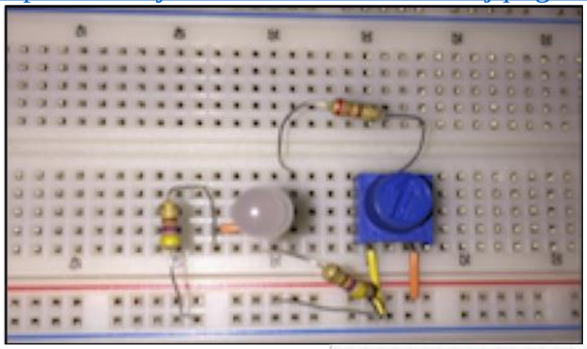
Procedure

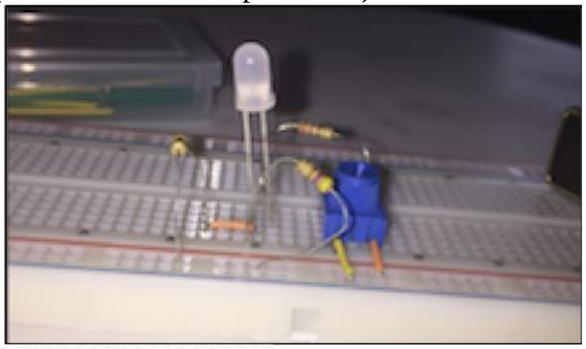
Parts Table	
Component	Quantity
9v battery	1
Potentiometer	1
470Ω resistor	2
220Ω resistor	1
Bi- color LED	1
Breadboard	1

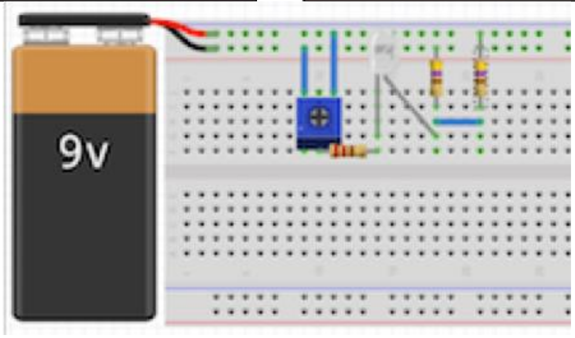
After learning about all the separate parts as well as how voltage dividers work, it is time to put them all together into a circuit and prototype a breadboard. however, there is some trial and error in figuring out what parts go where. Firstly, the circuit diagram must be altered in order to add the potentiometer to the circuit. Since the circuit diagram was created using a switch instead of a potentiometer it was difficult to tell which leg to input the current into and where the output would lead. However, after further examination of the circuit diagram it became clear that the junction with the switch formed three separate directions, just like how a potentiometer has three separate legs. After discovering this junction, it became very simple to see how a potentiometer could be added to the circuit, as well as how turning the dial on the potentiometer would cause a change in the current flow, to which of the two paths offered less resistance. What is the difference between a variable and fixed resistor? Firstly, the variable resistor, also known as a potentiometer, is a resistor of a certain value that can be changed simply by turning the dial on top. This is possible because the potentiometer has three separate legs, as opposed to the fixed resistors with two legs and a fixed resistance. As the dial is turned, the amount of voltage from A leg to B leg is changed and the rest of the voltage is sent to C leg, this offers a certain amount of resistance to the two separate paths connected to both C leg and B leg, dividing the voltage input to A leg. On the other hand, a fixed resistor is a resistor that had a certain Ω rating, and cannot be changed once being input to the circuit. It is easy to tell the total resistance on a resistor by reading the colored band code, referred to on page 74 in the book. Both of these resistors are crucial for the development of this circuit because the fixed resistors acted as voltage dividers to prevent the bi-color LED from being blown as well as the variable resistor which changes the path offering the least resistance for the current to flow through, and which leg of the LED the current travels through, therefor changing the color of the LED. Finally, the circuit diagram must be represented and tested out on a breadboard. This can be done by connecting the 9v battery to the positive and negative terminals on the breadboard. The circuit diagram shows which components are connected to the negative and positive terminals of the battery and which of the potentiometers legs connect to positive and negative respectively. After that it is a simple matter of completing the circuit by adding components between the positive and negative leads already mentioned. Once the prototyping is finished, with the twist of the dial on top of the potentiometer, the bi-color LED should change colors.

Media

https://www.youtube.com/watch?v=Bljqfeg6064(use subtitles for explanation)







Conclusion

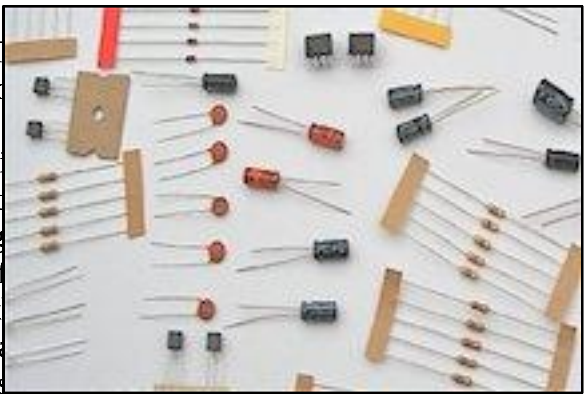
Over the course of the project I encountered many problems but after I focused and completed what I was working towards and through these challenges, I became better at many things. I became better at reading circuit diagrams and incorporating separate components, presenting the circuit diagram on a breadboard, as well as how to format an Engineering Report on Word and many useful tricks. At the beginning of this course, my only prior knowledge of reading circuit diagrams was last year during my science project, and even then, we weren't using the complex symbols seen on the circuit diagrams we look at in class. Through the first several classes I found myself wondering about the importance of learning these complex symbols, but now I realize truly how important they were. After taking the time to study diagrams I can draw them myself, I can find which parts go where as well as incorporating new components to the diagram already given. It would be difficult to build a circuit simply from a diagram, and that's why we use breadboards to prototype circuits. It's easy to take components out and try a new order, which makes it very simple to constantly attempt new circuits without losing parts. After using all of these new skills I have acquired, it was time to explain them in writing, also known as an Engineering Report (ER). ER's will store all my past and future projects, so it is important to format them in a legible and organized manner, although this is my first report, I found that my skills in Microsoft Word have increased dramatically. In conclusion, this course is a great way to merge my creative and scientific mind, and know that nothing is impossible as long as you work hard through trial and error to create something, it is truly a great experience.

Project 2. Analog Oscillator

Reference

http://darcy.rsgc.on.ca/ACES/TEL3M/1718/TasksFall.html - AnalogOscillator

Theory



The theory of this project is to use capacitors and transistors to cause LED's to flash back and forth. T circuit has two unstable states, changing back and fo But how does this work? Capacitors usually consist two electric plates, with a dielectric material or liqu between the plates. When voltage is applied, an elec field is created. One plate will collect positive charg and negative on the other. At first, the capacitors of little to no resistance, but as they charge at 63% of t remaining space, then 63% of what is left, as the ch increases, the resistance increases as well, letting les electrons in at each time interval. Transistors are 3 pronged components internally consisting of diodes and resistors. These 3 prongs are the base, collector and emitter pins. Similar to the two at rest states of push buttons, PBNO meaning push button normally open, meaning in its at rest state, no current can pass through the button, and PBNC or push button normally closed. The transistors used in this project are PBNO, meaning they only allow current to flow from collector to emitter when current is applied to the base pin. After dry fitting the circuit, it's time to dry fit then solder the components to a PCB board. This is useful because soldering is a very difficult skill to master, and you can't apply to much heat the internals of your parts.

Procedure

Parts table	
Component	Quantity
100µF Capacitor	2
2N3904 Transistors	2
10mm LED	2
1KΩ fixed resistor	2
10KΩ fixed resistor	2
9v battery	1
Slide switch	1
Printed circuit board	1

After learning about these separate parts, it is time to prototype, and finally solder the finished board. The first step is to take out all the components from the toolbox. The schematic could be found on page 8 in the book, the schematic can also be found on the project page listed above. Connect the 9v power supply to the breadboard. Begin by connecting the four separate resistors to the positive terminal and the capacitors in the polarity shown by the schematic.
Connect the negative terminal of each capacitor to the base of the opposing transistor. Connect the LED's to the positive terminal of the capacitor, then to ground. Connect the positive terminal of the capacitor to the
collector of the transistor and the emitter to ground. Test your circuit by connecting the power to the breadboard. The rate of flashing of the LED's can be changed by altering the resistance, or the rating of farads on the capacitor. Once the breadboard works,it is time to retrieve the PCB board for dry fit and soldering. Once studying the board and components,it is clear which components fit where and how to fit the parts in. Once all the parts are put into the PCB board, make sure that

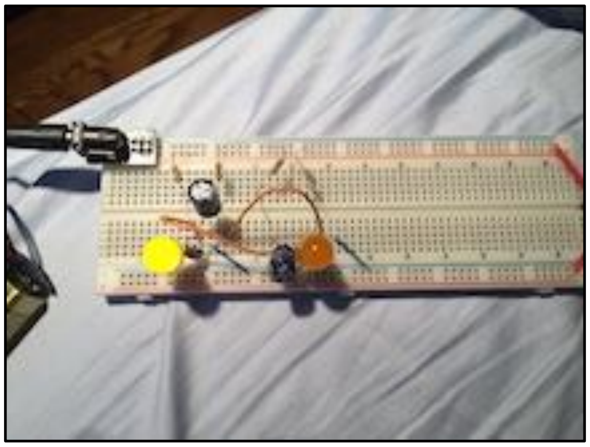
the parts are connected to the metal pads at the bottom of the hole, add the battery jack and power, and see if the circuit works, if not it can easily be troubleshot by looking at the example on the website. If it still doesn't work, use the DMM to test each of the parts induvially. Once the dry fit is complete and approved, the parts must be soldered. The parts that run less of a risk of being damaged due to heat should be soldered first, this includes the resistors, the switch, the battery adaptor, and the LED prongs. To ensure that the parts fit tightly to the PCB board, the parts can be held by hand while soldering or, can be held tightly using the reverse tweezers. The helping hands tools can hold the entire board stable while soldering. Using the side cutters, snip the excess wire off the bottom the PCB board. After soldering, use the DMM again to assure that the parts are still in working order. Finally, return the more fragile parts to the PCB board, making sure the transistors and capacitors are facing in the right direction. Remember, direct heat to these components for 4s or more run the risk of damaging the parts. Once the soldering is complete, use a magnifying glass to assure that all the joints are strong. Add the LED's to the prongs, in the right polarity. Connect your power, and the Analog oscillator should work. Remember to use the DMM to recheck all the parts and a magnifying glass to test the joints.

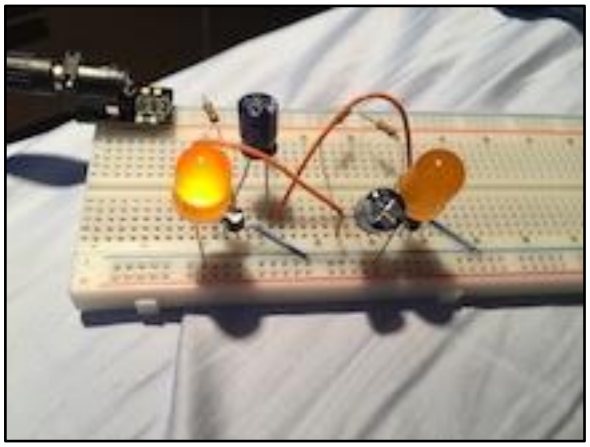
Purpose

The purpose of this project to teach students how to: both use transistors and capacitors, and how to solder components to a permanent PCB as opposed to keeping them on a breadboard. Capacitors and transistors are key components to circuit board, and it is important to know how they work. As well being able to solder components is a crucial skill in computer engineering, and fitting components to PCB boards.

Media

https://www.youtube.com/watch?v=rOSOMul5F-Q





Conclusion

Over the course of this project, I encountered many problems, but I became better at many important techniques used in the course. I became better at reading circuit diagrams and presenting them on a breadboard, and I became better at soldering components to a PCB. However, I did not become better at these techniques simply by practice, I had to learn from my mistakes. Firstly, when adding my components to a breadboard, I did not know the orientation in which to place the transistors, as not only do they have 3 legs, each of the legs serves a different purpose. This was easily solved however, as we learned in class, the orientation of the legs, when read from left to right, and facing the flat side are EBC or Enjoy Basic Computers. The capacitors, have the same orientation of the LED's.

Finally, I encountered a problem while soldering components to the PCB. Once I finished soldering, and added power, my circuit was still not working, I then went through the labor-intensive process of checking each of the components using a DMM. But, the single thing that was wrong with my circuit board was a faulty joint on the LED female header pin. This caused current not to flow into the LED, therefor it didn't flash. Overall, throughout this project I learned how to further read circuit boards, I learned about new components, and finally, I learned to test all my solder joints when soldering to PCB's.

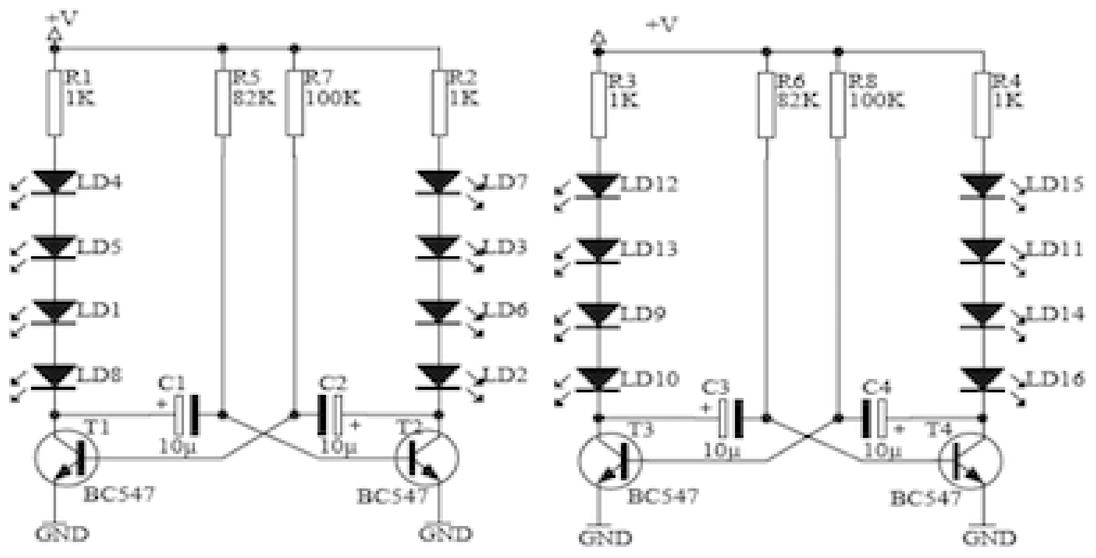
Project 3. The 3D Christmas Tree

Reference

http://darcy.rsgc.on.ca/ACES/TEL3M/1718/TasksFall.html - Tree

Purpose

The purpose of this project is to use two separate analog oscillator circuits to drive LED's on Printed Circuit Boards created to mimic the shape of a Christmas tree. There are 4 sets of 4 LED's being driven by two Analog oscillators. An Analog oscillator uses a series of components to cause LED's to flash back and forth. The first component used in an analog oscillator is two BC547 transistor's (3904's work as well), the transistor only allows current to pass through when current is applied to the base pin. The transistors function in the circuit is to allow current to flow though the LED's when current is applied to the base pin from the discharge of capacitors. Capacitors usually consist of 2 plates with an electrolytic liquid between them, they build up and save charge as current is applied to them, and eventually discharge the current in a single burst. Resistors also play a large role in the analog oscillator circuit, resistance is used to reduce current as stated in ohms law current=voltage/resistance. By limiting current, the rate at which the capacitors charge is also changed. The final component in the project are 3mm LED's (light emitting diodes) they are polarized devices that emit energy as light when the right amount of current is applied. Another purpose of the Christmas tree is to teach how to become better at solder as well as debugging circuits, as some of the copper pads are close together and difficult to solder without connecting, and since most of the components are polarized it is easy to accidentally switch the direction. For the 3D Christmas Tree project, it is extremely helpful to be patient when dry fitting and soldering components.



Procedure

Parts Table	
Component	Quantity
BC547 Transistor	4
10 μF Capacitor	4
3mm LED	16
Spring	2
1k Ω Resistor	4
100k Ω Resistor	2
82k Ω Resistor	2
Jumper wire	2
PCB	2
9v Battery	1

Once receiving the kit with all the components, separate the resistors, transistors, LED's, springs, PCB's and jumper cables. Firstly, solder the components that are the easiest and have the least chance of being compromised by the heat, such as the resistors. Since the resistors are non-polarized, the orientation on the PCB does not matter. The correct resistors can be put in place my checking the circuit diagram and matching the resistor number to the number printed on the PCB. When soldering, use the reverse tweezers to keep the components tight to the board, and prevent the component from moving while soldering. Once all the resistors are soldered in place, find all the capacitors and put them in place. Capacitors are polarized, remember that the anode is the positive leg. When soldering the capacitors, leave some room between the capacitor and PCB so that they can be bent down to rest on the board. The next component to be soldered to the PCB is the transistors, it is crucial not to apply to much heat to the transistors as they very easy to be compromised, it is best to solder one leg of each transistor at a time as to not apply all the heat at once. Next, solder the LED's into their places on the PCB's, remember that LED's are polarized as well, in the same orientation as the capacitors. Also, assure that the LED's can be bent to the notches along the sides of the PCB. The final step of the 3D Christmas tree is to solder the springs and jumper wires, bend the springs into designated holes and solder along the copper padding, making sure that they are a straight as possible. Slide the first PCB onto the second and connect J1-J1 and J2-J2 via jumper cables. Connect the springs to a 9V battery and watch the LED's flicker in a seemingly random pattern. If the tree does not work, check the solder connections and if that fails, check the components with a DMM.

Media

https://www.youtube.com/watch?v=h3dKRV4eAAA

Conclusion

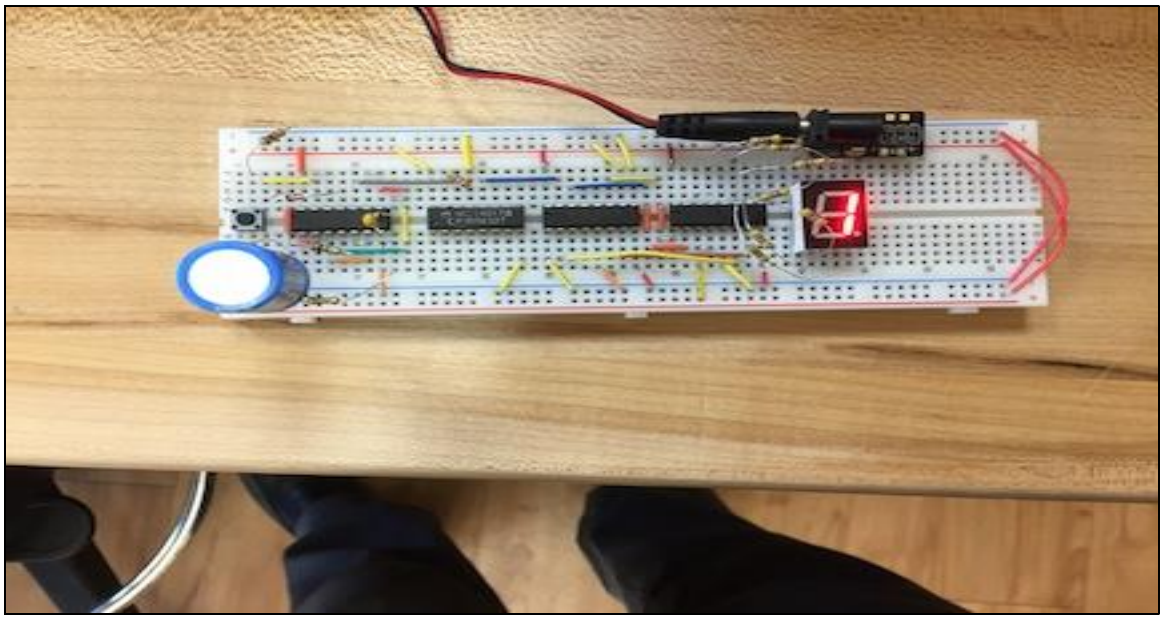
In conclusion, this project was very fun to make and is very practical, as well as how it relates to the previous project which is an analog oscillator. The project was enjoyable to build because it was not that much more difficult than last project, it was just on a larger scale than the single analog oscillator that was created in the previous project. The project is practical because it is a holiday bush that I created right around the holidays, it can be used as a gift, or as decoration. This project very clearly is related to the previous project because it uses two separate analog oscillators, using 8 LED's per oscillator, however all the LED's aren't along one side, they are mixed throughout the PCB. For those reasons, I believe that this was a great project for the grade 10 hardware class.

Project 4. A Counting Circuit

Theory

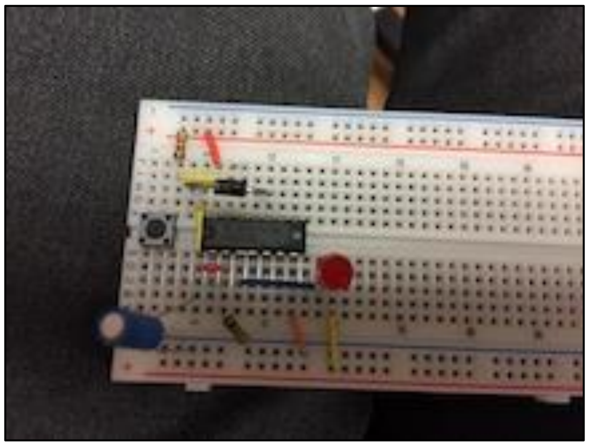
Parts table	
Component	Quantity
Breadboard	1
DC battery connector	1
Wire kit	1
0.1μF capacitor	1
100μF capacitor	1
220 Ω resistor	1
470 Ω resistor	7
Seven-segment display	1
4011 IC	1
4017 IC	1
4510 IC	1
4511 IC	1
9V battery	1
PBNO	1
LED	7

The theory behind this project is to use a series of specialized IC's and components to drive a seven-segment display and allow it to count 0-9. The first part of this circuit is to use a 4011 NAND chip and a PBNO to demonstrate NAND gates and how inputs work. Firstly, the input to this entire circuit, the push button. The push button (PBNO) allows current to flow when the button is pressed and cuts off flow when released. With the addition of a capacitor and resistor, the LED will stay on momentarily after the button is pressed. To increase the time the LED remains on, remember time = capacitance/resistance. After the analog input is complete, it is time to create a NAND gate oscillator, using the addition of a 0.1μF capacitor and some extra wires, the LED will oscillate once the button is pressed. This oscillation however, can be used as a clock pulse to operate the next chip, a clock pulse means that there is an output of a certain duration followed by no output of a certain duration, this output versus no input is always a constant. The 4017 specialized IC known as a divide by 10 chip and uses a clock pulse and sends each pulse to the next sequential output, the amount of outputs can be limited by connecting the last output to the clock reset pin. Once the divide by 10 pin is connected to the clock input of the following 4516 chip, which takes the clock pulse and uses 4 output pins which act as a binary counter and decoder, the counting goes to 15 and resets back to 0. The final IC used in this circuit is a 4511 chip which takes 4 inputs and displays them on a seven-segment display. Since the 4516 IC has 4 outputs of binary numbers, and the 4511 has 4 inputs. The 4511 will take the binary counting input and show it on the seven-segment display. However, the seven-segment can only display the numbers from 0-9 so there will be a break as the numbers range from 10-16 and then reset to 0. By using an analog input and a series of specialized IC's, it is possible to create a counting circuit.



A. Analog Input

Purpose

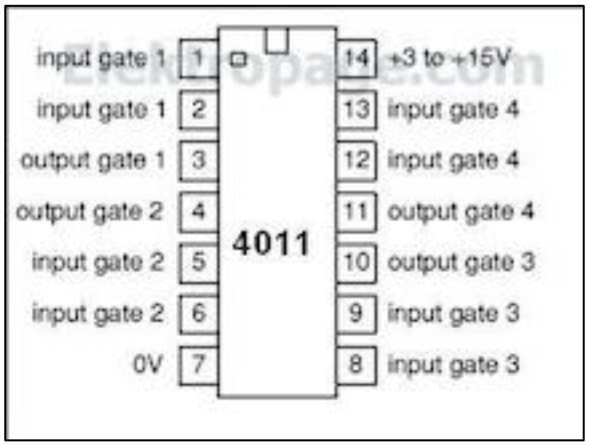


The purpose of developing this first circuit is to demonstrate a NAND gate and using a pull up resistor configuration to allow current to flow until the PBNO is pressed. Once the push button is pressed, the electrons take the path of least resistance and flow there instead of through the NAND gate. At rest, power flows through both of the inputs to the NAND gate and therefore the LED is off and remains off until the button is pressed. The input of a PBNO only allows one input to the NAND gate circuit. Since the push button is difficult to hold it is time to increase the time the LED stays on. The addition of a capacitor and resistor allow current to flow back out of the capacitor once the button is released keeping the LED on.

B. NAND Gate Oscillator (4011)

Reference http://mail.rsgc.in.ca//~cdarcy/Datasheets/CD4011.pdf

Purpose

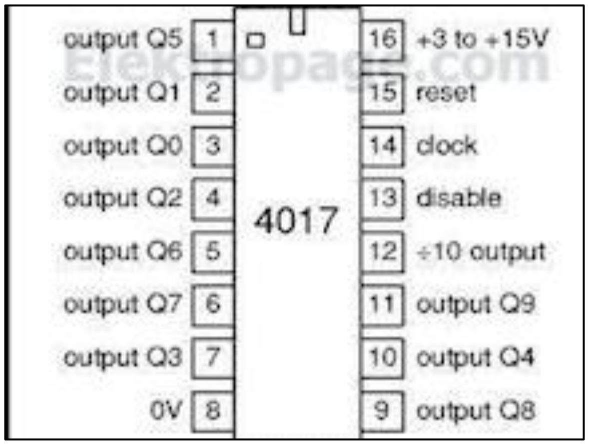


The 4011 chip, also known as the NAND gate chip is the primary and most significant chip in this circuit. NAND refers to a logic gate with two inputs, the gate has an output that is normally at logic 1 and only goes low to logic 0 when all its inputs are at logic level 1. Using a 0.1μF capacitor that is constantly filling and releasing, the output of the circuit causes the LED to flash at a constant rate. The input of this circuit (as previously discussed) is a PBNO which charges a capacitor that later releases into the resistor capacitor configuration. The analog input (pressing the PBNO) of this circuit then later feeds it through a specialized, digital IC. Each of the CMOS specialized IC's have a certain pin configuration shown in the pin out diagram above. To activate the chip, connect pin 7 to the negative rail of a breadboard and pin 14 to the positive rail. The chip has 4 separate NAND gates, with two inputs and 1 output each, 2 gates per each side of the chip. Using a combination of these gates as well as capacitors and resistors, once the PBNO is pressed, the LED will oscillate. The oscillation also shows that a clock pulse has effectively been generated. A clock signal is a signal which oscillates between a high and a low state and acts like a metronome and operates at a constant. Also found in this circuit, are two RC (resistor-capacitor) configurations label RC1 and RC2. The purpose of RC1 is to calculate the time that the circuit will continue to run after the PBNO is released and this can be calculated using the formula time (s) = Resistance (Ω) / Capacitance (F). The second RC configuration is to determine the rate at which the circuit oscillates, or how fast the LED will flash.

C. Decade counter (4017)

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

Purpose



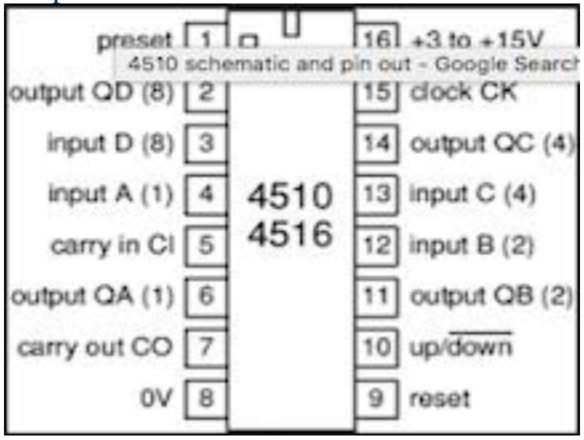
The third part of the circuit, is to input the clock pulse into a specialized IC, the 4017 chip, or also now as the divide by 10 chip. The 4017 IC takes a clock input and on each output, the IC sends the current to the next sequential output. After connecting the correct pins to positive and negative rails respectively, connect the clock output of the analog oscillator to the clock input of the 4017. Connect each of the next sequential outputs to 10 separate LED's. After the 10 outputs, 2 connections to power and ground, and finally the clock input, you are left with 3 pins.

Firstly, is the divide by 10 output, which outputs current for the last 5 outputs of the chip, this pin also works as a slow clock pulse. The next two pins are disable clock input and reset input. It is important to pay attention to pin configurations and be sure to ground pins that you do not want to be used, if the pin is neither grounded nor connected to VSS then it sits in a ghost state with some unpredictable outcomes. The reset input resets the outputs once it reaches 9, and if not grounded then the outputs will stop at 9 and remain stationary. The disable clock input disables the clock input and leaves the output stationary.

D. Decimal Counting Binary Up/Down Counter (4510)

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

Purpose



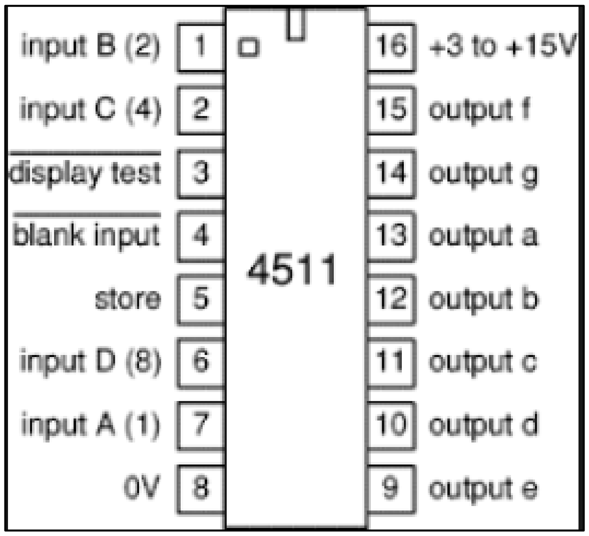
The 4510 BCD specialized IC, which stands for binary coded decimal, is a specialized IC that takes a
clock pulse and transfers it into Binary numbers through 4 separate outputs labeled A, B, C, D. The binary decoded into base 10 numbers can be found in the table to the right. Taking the output of the divide by 10 pin on the 4017 and connecting it to pin 15, connect pin 16 and 0 to negative and positive rails respectively. Connect the carry in clock and preset rails to ground. If the circuit is counting too slowly then instead of connecting the 4510 to the divide by 10 pin of the 4017 then connect it to the original clock pulse from the NAND gate oscillator.

Binary	Base 10
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

E. Binary Counting Decimal Counter (4511)

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

Purpose

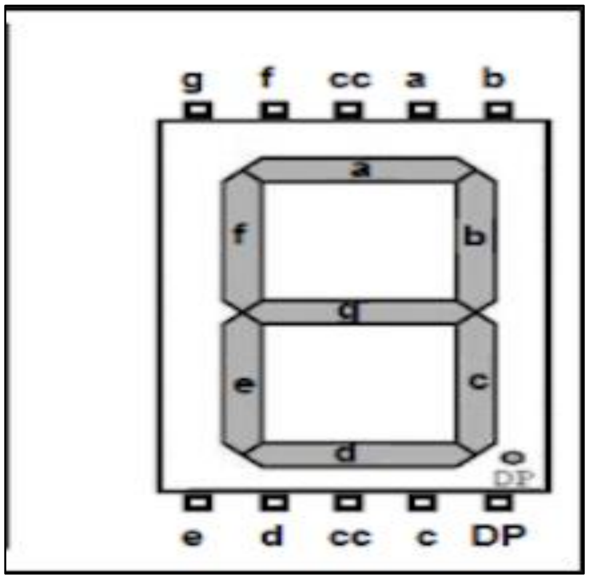


The 4511 specialized IC is a binary decoder which turns a binary input and outputs it to display on a seven-segment display. There are four input pins on the 4511-labeled input A-D respectively. These can easily be connected to the outputs labeled A-D on the 4510 IC. After receiving the binary counting output from the 4510 the 4511 processes the input and outputs it through seven separate pins, therefor driving a seven-segment display. After factoring out the output, input, voltage, and grounding pins, there are three pins left; store, blank input and display test. Display test will turn on all of the segments on the seven-segment display to demonstrate it is working well. Blank input pin will stop the number where it is an blank any input being sent to the four input pins. Finally store (also known as latch enable) when the enable input is on, the signal propagates directly through the circuit, from the input to the output.

F. Seven-Segment Display

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

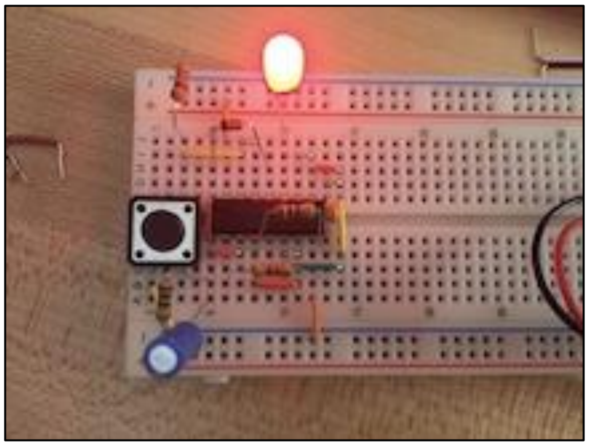
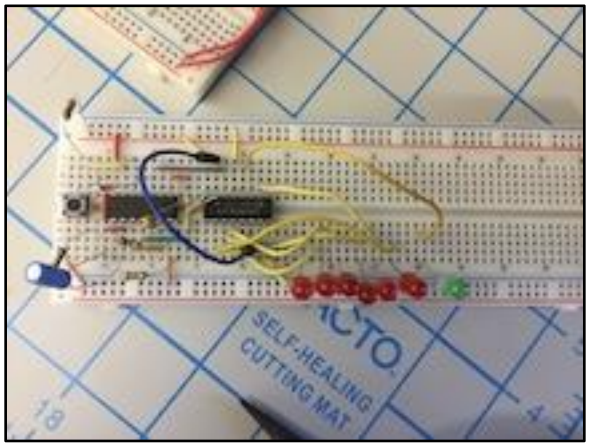
Purpose

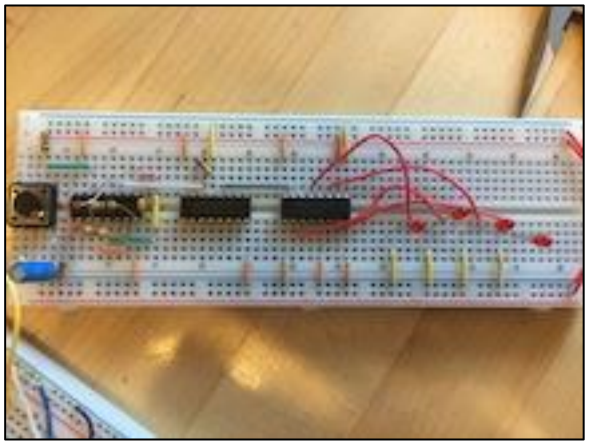
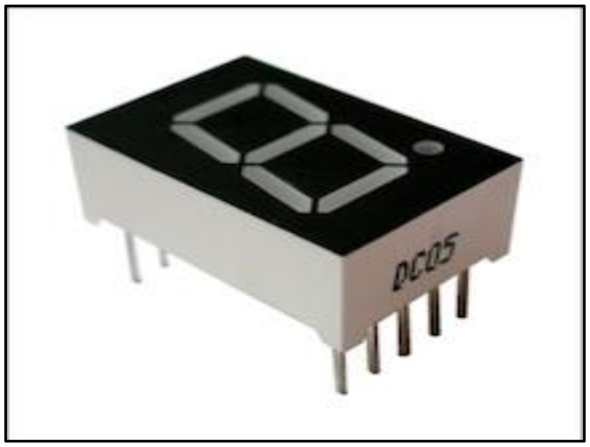


A seven-segment display is uses 7 LED's to light up 7 separate rectangular displays, using the orientation of the bars, it is possible to display the numbers 0-9. As well to the separate segments there is a decimal point as well. The segment we used was a common anode meaning that the center pin on both sides, when connected to the positive allow the segments to turn on when the respective pins are grounded. Seven-segment displays are available in both common anode and common cathode. There are seven output pins on the 4511-labeled a-g can be easily connected to pins a-g on the seven-segment display. However, each pin requires a separate resistor because they are each separate LED's and if only one has a resistor attached, it would be dimmer than the rest of the segments.

Media

https://www.youtube.com/watch?v=_aNvdUCtt6Q

Conclusion

The counting circuit project is the final project of the year (except for independent Study project) and was quite difficult to build and understand. This project gave me better knowledge of inputs and outputs, through using an analog input and predicting the outcome or output of the circuit. I became better at conserving space on a breadboard and wiring more carefully, knowing that this would be a large circuit I needed to keep my wiring tight and conserve as much space as possible. I gained knowledge of four new specialized IC's and became better at reading and understanding pin out diagrams. The final and most important skill I have acquired during this project is time management, both time management in building and completing the circuits through each step, and time management in writing my ER's. Especially with a report as long as this one it is important to not procrastinate and to finish the work in advance. In conclusion, this project has given me experience in many important aspects of engineering.

Project 5. LED Cylinder

Theory

The theory behind this project is to use both a 555 timer IC and a 4017 divide by 10 IC to create a rotating effect on a LED cylinder. The first component in the project is a perma-proto half breadboard, this component uses a breadboard style arrangement of holes, just like a breadboard. The difference is that the perma-proto board uses large holes in order to be solder friendly and a more permanent way to store and build products. The first part of this circuit uses a 555 timer to create a clock pulse, a clock pulse is a signal that oscillates between a high and a low state and is utilized similar to a metronome. The length between these pulses can be determined on a 555 timer by changing capacitance between pins 1 and 2 or changing resistance between both pins 8 and 7, as well as 7 to 6. If the resistor between pins 7 and 6 is replaced with a potentiometer, then the speed of the pulse can be changed by altering the resistance on the potentiometer. The clock pulse then feeds into the 4017 divide by 10 specialized IC. Which takes in the clock pulse and converts it into 11 separate outputs, 10 sequential outputs, and one divide by 10 output. The 10 sequential outputs are labeled 0-9, and with each pulse entering the 4017 the next sequential output will output VSS. The LED cylinder is of comprised 3 layers of 6 LED's each layer with a corresponding common cathode for the entire layer, each column of LED's are connected to a common anode. Therefor single LED's can be controlled by grounding a certain layer and applying VSS to a column. After applying the first 6 outputs of the 4017 IC to 6 corresponding columns of LED's the cylinder will dance using the certain outputs designated.

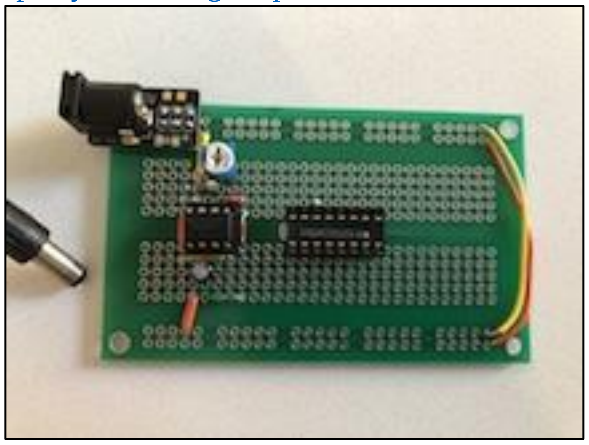
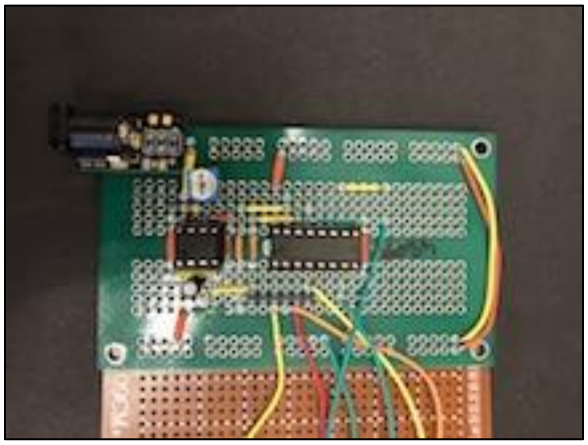
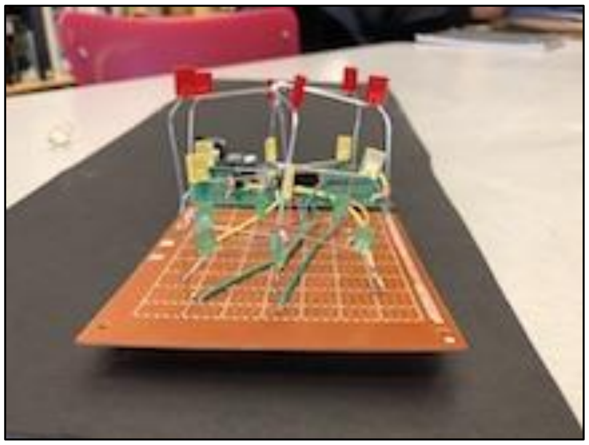
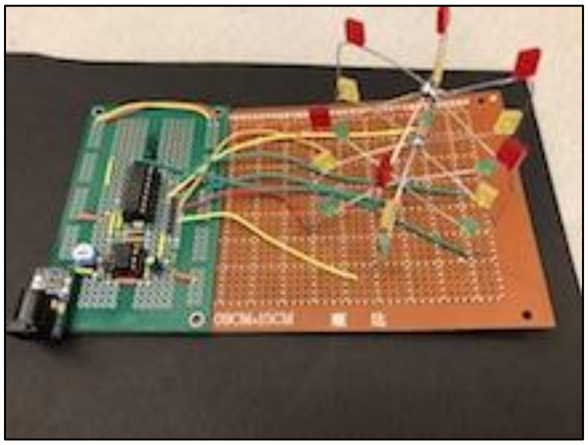
Procedure

Parts List	
Component	Quantity
Square LED's	18
Perma-Proto half breadboard	1
555 Timer	1
4017 IC	1
Chip seat	2
Wires	28
DC battery adapter	1
1µF capacitor	1
100kΩ potentiometer	1
220Ω fized resistor	2
9V battery	1
Block of wood	1

Once acquiring all the components required lay them out in an organized format. Add the 555 timer and 4017 IC to a breadboard 4 rows apart. Connect the IC's to power and ground rails respectively. Connect pin 8 to pin 7 of the 555 timer using the 220Ω fixed resistor. Connect pin 7 of the 555 timer to pin 6 using the A and C pins of the 100kΩ potentiometer. Connect pin 4 of the 555 timer to pin 8. Connect pin 2 of the 555 timer to pin 6. Connect pin 1 of the 555 timer to pin 2 with the 1µF capacitor, the positive leg connected to pin 2 and negative leg to pin 1. Connect the output (pin 3) of the 555 timer to pin 14 of the 4017 divide by 10 counter. Connect outputs 0-5 of the 4017 IC to 6 separate LED's. Connect output 6 of the 4017 to the reset pin (pin 15). Connect the battery and debug any problems found in the breadboard prototype. Solder the chip seats to the perma proto board in the same orientation as the breadboard prototype. Copy the breadboard wiring on the 555 timer. Once done wiring the 555 timer connect a LED to the output to make sure that the 555 timer is creating a clock pulse, the LED should oscillate. Once it is confirmed that a clock pulse is created, connect the output of the 555 timer to pin 14 of the 4017 IC and repeat the breadboard wiring as well, except for the outputs. Draw a circle on the block of wood and mark 6 spots on the circle, equidistant from each other. Using a 7/16" drill bit, drill holes where the marks on the circle are. Place a LED in each of the holes with the cathodes bent to the center of the circle and the anodes bent outwards. Solder all the cathodes to each other, and remove the layer from the template. Repeat for as many layers as required. Once all layers are complete, stack the layers using 2" of foam between each layer to keep balanced. Bend the anodes of the LED's to create 6 vertical columns. Solder each of the columns and carefully remove the foam spacers. Connect all of the common cathodes for each layer to one central wire. Connect the central wire to the negative rail of the perma proto breadboard and connect each of the sequential outputs (0-5) to the respective columns on the cylinder.

Media

https://youtu.be/9gKfnp_fJFw

Conclusion

In conclusion, I encountered many problems throughout the project and had some successes. My first challenge was in choosing one of my multiple ideas for the independent study project (ISP), as throughout the year I found many interests. The LED cylinder idea came into my mind while watching a LED cube instructable, I liked the idea of a 3D controllable object because it appealed to both my liking of structures and structural engineering as well as controlling it using specialized IC's and other basic electronic components, as opposed to a micro controller or programmable device. The next problem was encountered in designing the cylinder, at that point I already knew I wanted to control my structure using a 4017 IC using the clock pulse from a 555 timer. Next was to decide the structure to build, either a cylinder or a rectangular prism were my two good ideas, as well as how high my structure would be built. I did not encounter any problems in breadboard prototyping my circuit or acquiring all my components. The next problem was in deciding which LED's would be used in the cylinder. 3mm and 10mm LED's did not fit it the template I made, so when I tried using them, the layers could not stack properly, and with the 5mm LED's the cylinder was uneven and LED's became angled. Multiple times throughout the build process I forgot that the LED's were wired in parallel not series and only required a maximum of 3V, and accidentally applied 9V to test them. Eventually I built my final LED cylinder design, using block LED's 3 layers high. I also had multiple successes while building this project, I succeeded in designing and prototyping the circuit on a breadboard with tight wiring and minimal space taken up. My soldering has improved drastically as illustrated by the good connections and tight wiring on the perma proto half breadboard. I succeeded in fulfilling the description of my product in my ISP proposal, with a robust and clean design. Overall, I think this project was a great way to demonstrate the skills and knowledge in this half course, as well as pushing myself slightly out of my comfort zone design wise.

ICS3U

AVR Foundations

Project 6. Traffic Light Assembly and Testing

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Tasks.html - TrafficLightAssembly

Purpose

The purpose of this project is to learn the basics of coding and using Arduino, as well as helping regain our critical hardware thinking skills after a long summer. The project is to drive 3 LED's using only 3 pins on the Arduino. Which is no easy task because there are 2 legs per LED, making 6 pins total. Figuring out the arrangement of the pins requires some critical thinking skills, because in some orientations multiple LED's would come on at the same time. Although the hardware aspect of this project was not as intensive or thought provoking as previous projects, the real challenge was in the code behind it. The Arduino Uno uses a DIP ATmega328P MCU and outputs it to 28 pins, consisting of mostly power, ground, analog input, and digital output pins. The P in ATMega328P stands for pico power, meaning the chip can run on anywhere from 1.8-5 volts. The Arduino Uno uses a code similar to C, and is powered by the USB slot of a computer, or a 5v input. The purpose for learning how to code an Arduino it is close to C code, a code that all engineers need to know. As well, learning how to program the digital input and output pins opens up a lot of projects and ideas that were difficult to control simply using hardware. Such as creating a square wave and being able to change the frequency and duty cycle without having to use a series of resistor capacitor circuits connected to a 555 timer.

Procedure

Parts List	
Component	Quantity
Stripboard	1
Light Emitting Diode (red, green, amber)	3
Arduino Uno	1
ATMega328P MCU	1
Breadboard	1
28 gauge wires	3
Three-pin right-angle header	1

After receiving the bag with the stripboard and 3 pronged male header pin, solder the header pin to the stripboard with the curved part facing away from the board. Think as to the orientation of the LED's on the stripboard and how to use the 3 pins most effectively. Test the LED's on a LED tester to make sure they all function, Arrange the LED's on a breadboard and change the orientation of them until they are in an position that will work. This orientation is to have the anode of the green LED on the far right pin and its cathode on the center pin. Then solder the anode of the amber LED to the center pin and the cathode to the left pin. Finally, solder the anode of the red LED to the left pin and the cathode to the center pin. Check to make sure all the solder connections are strong, and that the LED's are flush against the stripboard. Connect the right pin to digital pin 11, center pin to digital pin 12, and the left pin to digital pin 13 on the Arduino board. The next step is to write the code. Open a new .ino file and add the project title, author, date, status, and reference. Create 3 unsigned 8 bit integers to digital pins 13, 12, 11, and name the green, amber, and red respectively. Create an unsigned 16 bit integer for duration and make it 3000 milliseconds (3 seconds), this is how long the green and red lights remain on for. In setup, set the pin mode for pin 11, 12, and 13 as output. Under loop, write the green pin to high for a delay of 3000 milliseconds (duration), then set it back to low. Then, write the amber pin to high for a duration of 1000 milliseconds (duration/3), and write it back to low. Finally, write the red pin to high for a duration of 3000 milliseconds (duration), and

write it back to low. Compile the code, and if it compiles completely upload it to the Arduino. Connect the 3 male header pins to the female
header pins for digital pin 11, 12, and 13. The traffic light should be working, with the green LED coming on for 3 seconds, then the amber LED for 1 second, and finally the red LED for 3 seconds.

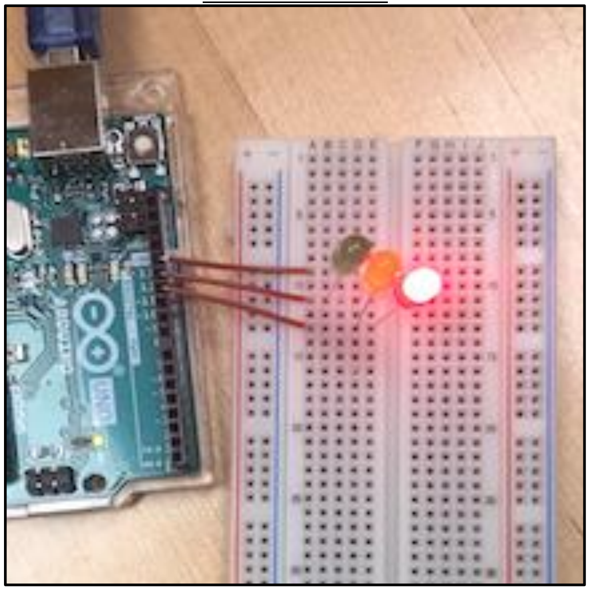
Code

```
//Project : Traffic Light
//Author : Joshua Dolgin
//Date   : 2018/09/18
//Status : Working
//Reference :http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Tasks.html

uint8_t green = 13;
uint8_t amber = 12;
uint8_t red = 11;
/*uses an unsigned 8 bit integer to rename
each pin to the corresponding color*/

uint16_t duration = 3000;
/* uses an unsigned 16 bit integer to create
a variable called "duration that lasts 3000
milliseconds*/

void setup() {
  // put your setup code here, to run once:

pinMode(green, OUTPUT);
pinMode(amber, OUTPUT);
pinMode(red, OUTPUT);
/*sets the 3 pins to output*/

}

void loop() {
  // put your main code here, to run repeatedly:
 digitalWrite(green, HIGH);
 /*sets the pins output to high,
 effectively turning on the LED*/
 delay(duration);
 /*delays for the variable "duration"
 which was previously defined as 3000 milliseconds*/
 digitalWrite(green, LOW);
 /*sets the pins output to low,
effectively turning off the LED*/
 digitalWrite(amber, HIGH);
 delay(duration/3);
 digitalWrite(amber, LOW);
 digitalWrite(red, HIGH);
 delay(duration);
 digitalWrite(red, LOW);

}
```

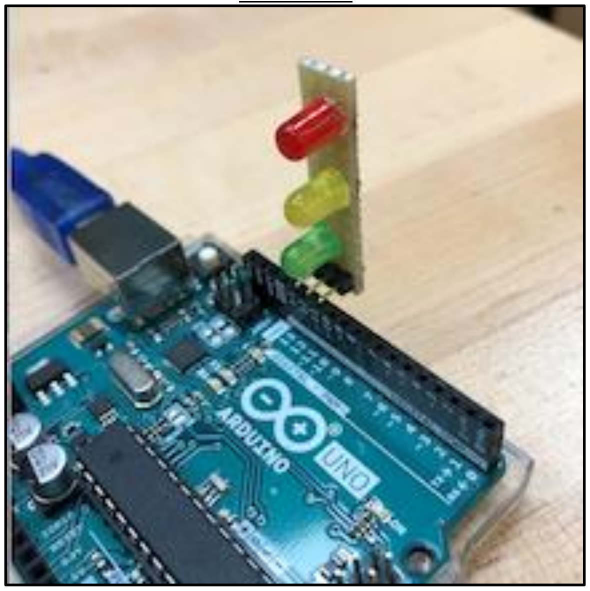
Media

https://www.youtube.com/watch?v=FlCa0TD4fB0

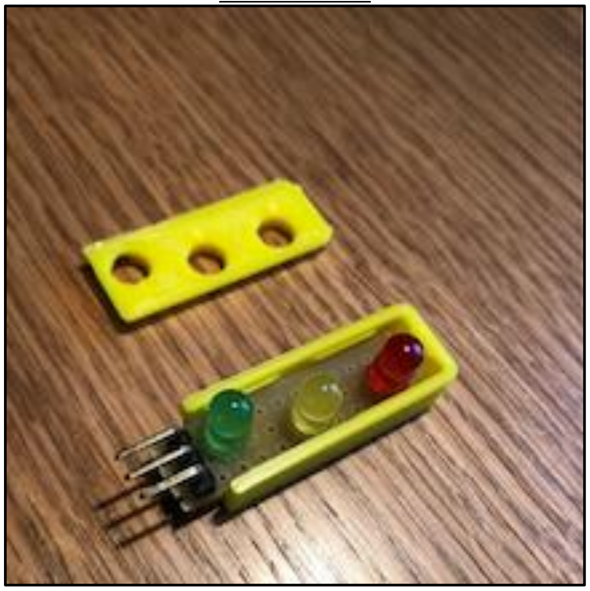
Breadboard test



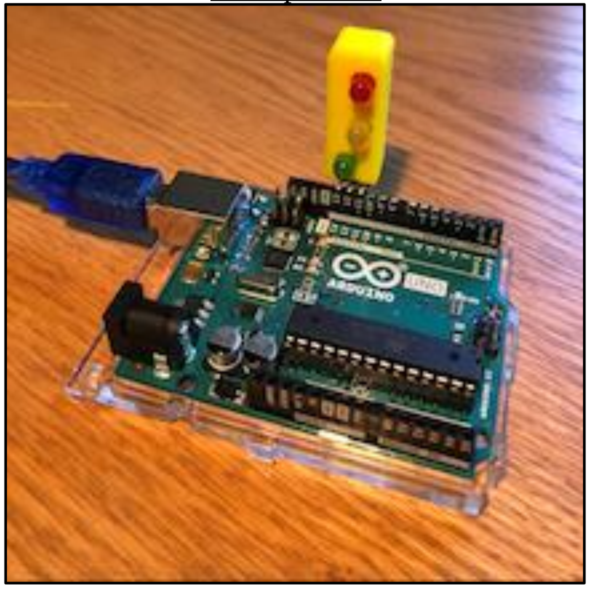
Soldered



Case test fit



Final product



Reflection

I really enjoyed the first engineering report of my grade 11 year. I learned how to write some code in Arduino, I got to work on my critical thinking skills, and incorporate something I'm really passionate about. Knowing how to code will help me in the future as an engineer and also opens up a wide array of projects that would otherwise be hard with mechanical components. I got to incorporate something that I'm passionate about, 3D design and printing. I spent a lot of time in the DES over the last two weeks working on my ER and building the new Prusa i3 MK3, and I enjoyed finding a way to combine both my school work and my passion. After 3 iterations of the traffic light holder, I finally had a simple design that fit well, it uses 5 snap fit connections to ensure that the lid stays tight to the case, and also requires no other mechanical components.

Project 7. ASCII & Buttons

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Tasks.html - ASCII

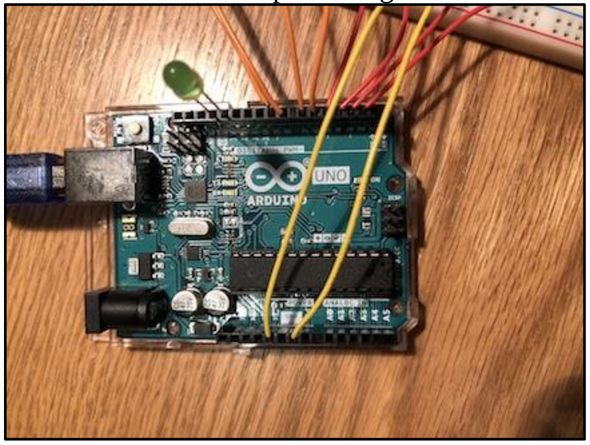
Theory

The theory behind this project is to use the Serial Monitor on the Arduino Uno as well as its digital input pins to create a tactile game to work on binary skills. The Serial Monitor on an Arduino is essentially a tether between the computer and the ATMega328P UC. The Serial Monitor operates in terms of Baud (bits per second), and can display characters and numbers. This project consists of writing a code that generates a random ASCII character between the integers 33 and 126 (ASCII characters ! to ~), and displaying the value as both an integer and an ASCII character. The other part is to use the digital pins on the Arduino Uno to take 8 input values, each switch is intended to represent a single bit ranging from 2^0-2^7. By using 8 slide switches being pulled down to ground; each of the bits are read as a 0 (off), until switched to the on (1) position. If the random integer value is correctly represented in binary across the 8 slide switches, power is applied to the anode of a green LED, resulting in the LED turning on.

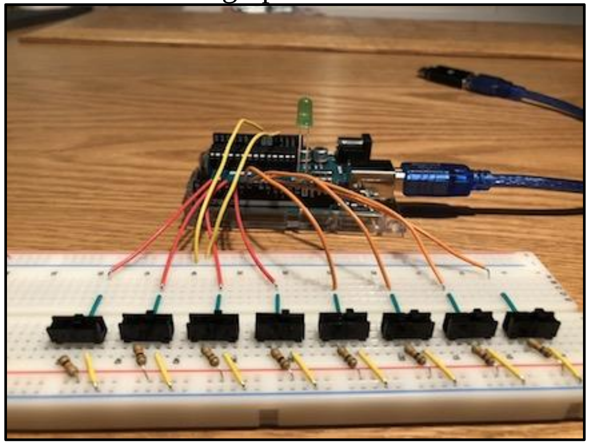
Media

https://www.youtube.com/watch?v=-PGrJ90f6KU

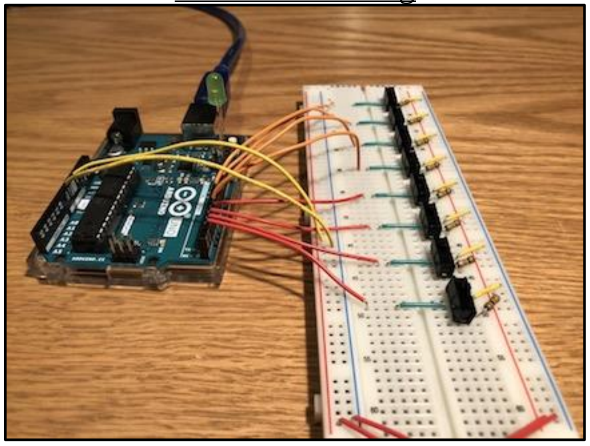
Initial pin wiring



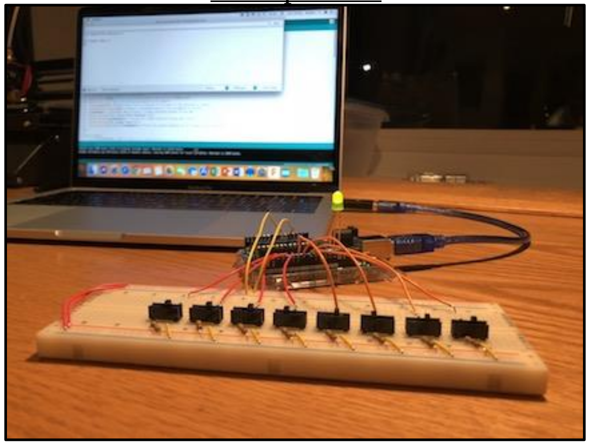
Setting up the switches



Breadboard wiring



Final product



Code

```
//Project : To generate a random ASCII character and have a participant represent
the character in binary.
//Author : Joshua Dolgin
//Date   : 2018/10/9
//Status : Working


uint8_t bin1 = 2;/*Uses an unsigned 8 bit integer to rename pins 2-9 to their binary
values*/
uint8_t bin2 = 3;
uint8_t bin4 = 4;
uint8_t bin8 = 5;
uint8_t bin16 = 6;
uint8_t bin32 = 7;
uint8_t bin64 = 8;
uint8_t bin128 = 9;
uint8_t greenLED = 13;
uint8_t randomNumber; /*Creates an unsigned 8 bit integer named randomNumber*/
uint8_t binaryInput = 0;/*Creates an unsigned 8 bit integer equal to 0*/

void setup() {
  Serial.begin(9600);
  pinMode(greenLED, OUTPUT);/*Sets the LED pin to output*/
  while (!Serial); /*waiting for serial monitor to start*/
  randomSeed(analogRead(0));/*Uses the analog read on pin 0 to generate a
randomseed*/
  randomNumber = random(33, 127);/*Creates a random integer between 33 and 126*/
  Serial.println("Your Random ASCII character is");
  Serial.println((char) randomNumber);/*Prints the random integer as a character*/
  Serial.println("The integer value is");
  Serial.println(randomNumber); /*Prints it as an integer*/
}


void loop() {
  if (digitalRead(bin1))/*If there is power going to pin bin1, it adds 1 to the
binaryInput value*/
    binaryInput++;

  if (digitalRead(bin2))
    binaryInput += 2;
  if (digitalRead(bin4))
    binaryInput += 4;

  if (digitalRead(bin8))
    binaryInput += 8;

  if (digitalRead(bin16))
    binaryInput += 16;

  if (digitalRead(bin32))
    binaryInput += 32;

  if (digitalRead(bin64))
    binaryInput += 64;

  if (digitalRead(bin128))
    binaryInput += 128;

  if (binaryInput == randomNumber) { /*If the binaryInput is equal to randomNumber,
then it turns on the greenLED*/
    digitalWrite(greenLED, HIGH);
  } else
  {/*If binaryInput is not equal to randomNumber, the LED is turned off and
binaryInput is set back to 0*/
    digitalWrite(greenLED, LOW);
    binaryInput == 0;
  }
}
```

Procedure

Parts Table	
Component	Quantity
Arduino UNO	1
ATMega328P	1
Wires	26
Slide Switch	8
Green LED	1
Breadboard	1
10KΩ fixed resistor	8

Gather 8 slide switches and wire them to the breadboard using pull down wires on one side, and the other side connected to the power rail via a 10KΩ fixed resistor. Test the green LED to make sure it functions properly and will emit light when current is applied to it. Attach the anode of the green LED to digital pin 13 on the Arduino board and the cathode to the ground pin. Connect switches 1-8 to pins 2-9 respectively. Connect the 5v pin on the Arduino to the power rail and the ground pin to the ground rail. Connect the Arduino board to a computer; then open the Arduino IDE and create a new sketch. Use 8 separate unsigned 8 bit integers to rename pins 2-9 to the binary values they will represent. Create 2 different unsigned 8 bit integer to represent the random number and the binary input value. Create a random integer between 33 and 127 using a random seed based off of the analogRead from pin 0. Display the integer and its character value in the Serial Monitor. Create separate "if" statements for each switch that adds the binary value of the switch it represents to the binary input value. Create an "if" statement that turns on the LED if the binary input value equals the random integer, and if it is not equal the LED will be turned off and the value of binary input will be set to 0.

Reflection

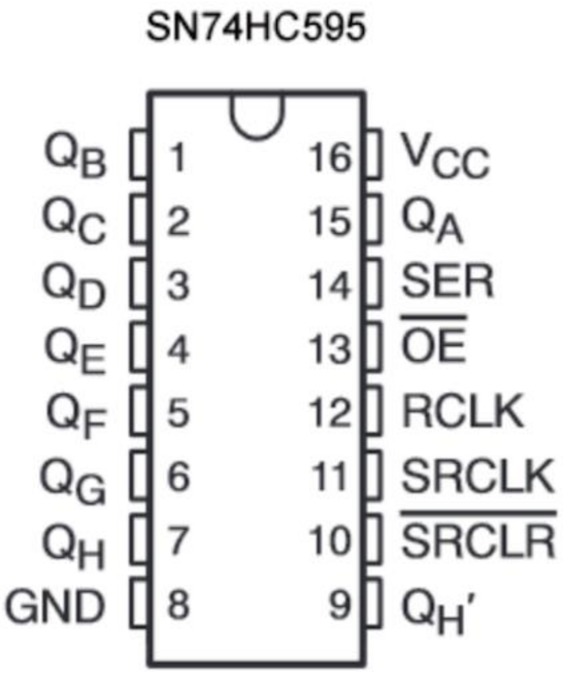
When I first read the email on the forum discussing ASCII and buttons, my mind immediately starting running through ideas, both too advanced or not advanced enough to what we had been learning in class. After a few more classes in discussing the `randomSeed` and `digitalRead` functions; I narrowed my options to what I believed would be an entertaining project to work on that was not beyond my skill level, but would further advance my understanding of the concepts themselves. When I first wrote the code I was not thinking on how to reduce how much code was used, but simply how to get it to function. After some time thinking, I thought of a different way to write the code so it would apply to all the ASCII values rather than just some. Although the code for this ER still exceeds what is recommended in terms of space, I believe I used what I learned last year to the best of my ability and look forward to rewriting it using much less space later this year.

Project 8. Shift Register – Bargraph

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Submissions.html

Purpose



The purpose of this project is to find an unfamiliar sensor, and display its readings on a bargraph using the SN74HC595 Shift Register IC. There are many sensors available that measure a variety of things, so finding one that is interesting and different is not too difficult a task. The most important component however, is the 595 Shift Register. The 595 takes a clock pulse as an input and internally maps it in 8 bits, each either being high or low, once power is applied to the latch pin, the 8 values are pushed through the 8 outputs (QA-QH) as either high or low. Through using the `shiftOut` command in the Arduino IDE, the Arduino does all the dirty work and outputs the clock pulse and the serial data to the necessary pins. In this project, the 8 outputs of the 595 will be connected to 8 of the 10 LED's on a LED bargraph. By taking an analog reading of the output voltage the sensor creates, and mapping it to a value between 0 and 8. The data can be sent to the 595 via 3 digital output pins, and later displayed on the bargraph.

Procedure

Parts Table	
Component	Quantity
Sharp GP2Y0A41SK0F Analog Distance Sensor	1
Breadboard	1
SN74HC595 Shift Register IC	1
10 LED Bargraph	1
Tim Morlands PCB	1
5 pin right angle header	1
Connector Wires	5
Arduino UNO	1
ATMega328p	1
330Ω bussed SIP resistor	1
16 pin chip seat	1

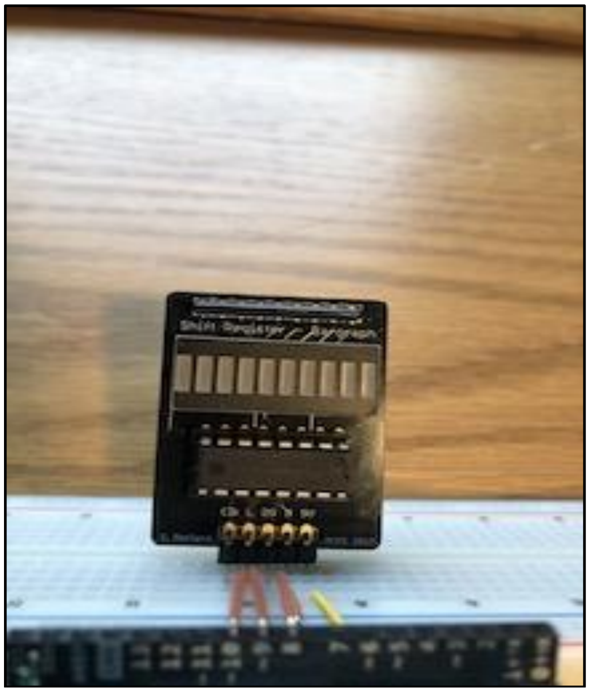
Gather the Sharp analog distance sensor and the Arduino UNO. Connect the 5v and ground wires of the distance sensor to their respective pins on the Arduino. Connect the data output wire to one of the analog input pins on the Arduino. Create a new sketch in the Arduino IDE and include the `Sharp.IR` library. Create an unsigned 8 bit integer that reads the distance via the analog input and prints the value on the Serial Monitor. Gather all the components to be soldered to the PCB and lay them out. Test the bargraph to make sure all the LED's are functioning properly. Solder each of the components to the PCB, careful of their orientation. Test each of the LED's to make sure they still function correctly. Open up the previous code, and create an `array` in binary going from all 8 bits as zeros, to all 8 bits as ones. Using the

Arduino `map()` function, map the distance read from between 4-30, to 0-8. Write a `for loop` that loops itself the amount of times as the newly mapped value. Within the `for loop` write the latch pin to low. When the `for loop` is finished write the latch pin to high, releasing the value through the 8 output pins of the 595. Finally, write a `shiftOut` command that outputs the array of binary values to the Shift Register.

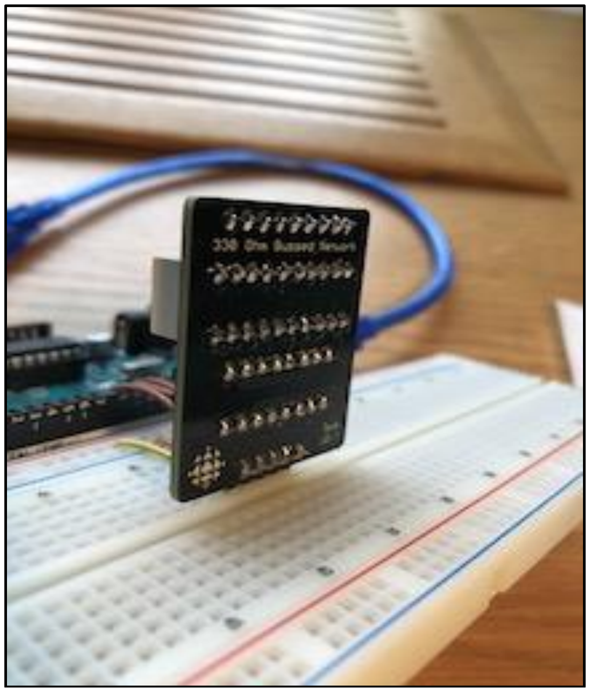
Media

https://youtu.be/UJ3j36QZLZI

PCB test fit

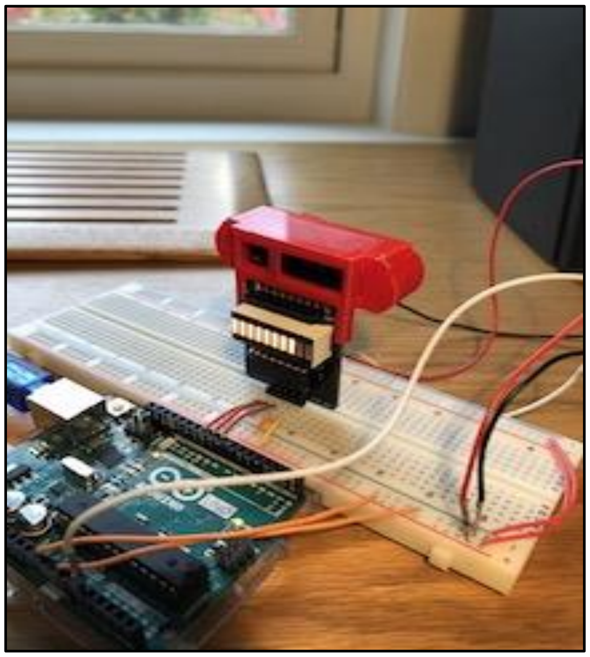


Solder Connections



Case final design

Final Product



Code

```
//Project : To measure the distance using an IR motion sensor and displaying
//it on a bargraph using a 595 shift register IC
//Author : Joshua Dolgin
//Date  : 2018/10/27
#include <SharpIR.h>//Includes the Sharp infared sensor library
#define model 430 //Defines the model of infared sensor
uint8_t latchPin = 9; //Pin connected to ST_CP (latch) of 74HC595
uint8_t clockPin = 10; //Pin connected to SH_CP (clock) of 74HC595
uint8_t dataPin = 8; //Pin connected to DS (data) of 74HC595
uint8_t infaredSensor = A0; //Uses 8 bit unsigned integer to rename pin A0
SharpIR SharpIR(infaredSensor, 430); //Measures the distance away from the sensor
//Creates an array of binary value
uint8_t digitMap[] = {0b00000001, 0b00000011, 0b00000111, 0b000001111,
0b00011111, 0b00111111, 0b01111111, 0b11111111};//
void setup() {
  Serial.begin(9600);
  while (!Serial);
  pinMode(dataPin, OUTPUT); //Sets dataPin to OUTPUT
  pinMode(clockPin, OUTPUT);//Sets clockPin to OUTPUT
  pinMode(latchPin, OUTPUT);//Sets latchPin to OUTPUT
}
void loop() {
  uint8_t dis = SharpIR.distance(); // this returns the distance to the object
you're measuring
  dis = constrain(dis, 4, 30); //Constrains the value between 4 and 30
  Serial.println("Average distance in centimetres");
  Serial.println(dis); //Prints the average distance of centimetres
  dis = map(dis, 4, 30, 1, 8);//Maps the distance to an integer from 0-8
  Serial.println("Amount of LED's");
  Serial.println(dis); //Prints the amount of LEDs that should be lit up
  for (uint8_t j = 0; j < dis; j++) {
    digitalWrite(latchPin, LOW);  //ground latchPin and hold low for as long as you
are transmitting
    shiftOut(dataPin, clockPin, MSBFIRST, digitMap[j]);
  }
  digitalWrite(latchPin, HIGH);//Returns the latch pin to high
  delay(1000);
}
```

Reflection

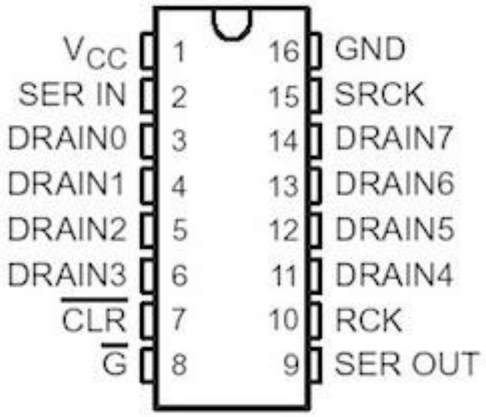
When I first saw Project 3. Shift Register – Bargraph appear on the aces page, I immediately began my research into the 595 shift register and its function. Although I understood the `for loops` and `arrays`, I couldn't seem to grasp what the 595 actually did. Once we started to use the bargraph and 595 in class, I began to use the 595 in tandem with `for loops` and `arrays` in order to further understand this integrated circuit. Then came the creative portion of this product. It was time to choose a sensor to collect the data for my project: I tore through my toolbox looking for something that would be different, but also interests me. I stumbled across the Sharp IR motion sensor I scrapped after attempting to you use it last year with no success and decided to give it a second try. Through further research of the motion sensor and its capabilities, I discovered how to successfully integrate it into my circuit. I enjoyed watching how much I have improved in both my software and hardware capabilities since last year and look forward to improving even more.

Project 9. MatrixMadeEZ

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Tasks.html - MatrixMadeEZ

Purpose



The purpose of this project is to use two different 595 Shift Register IC's to control either the rows or columns of a 8x8 LED matrix. A LED matrix uses 16 pins (8 Cathods and 8 Anodes) to control 64 LED's in a 8x8 square configuration. By controlling where power and ground is applied to the 16 pins, specific LED's can be controlled to create complex animations. By using two Shift Registers, only 3 pins need to be tied to the Arduino's digital input pins rather than 16 in order to control the matrix. The first Shift Register is the SN74HC595, which takes in data as a clock pulse an internally maps it to 8 bits. Once the latch pin is set to high, the 8 bits are applied to the 8 outputs. The other Shift Register is a TPIC6C595 power logic 8-bit Shift Register. The TPIC6C595 functions off of the same general concept of the SN74HC595, using clock, data and latch pins; but, the TPIC6C595 can source much more voltage on a single pin then the SN74HC595, allowing it to ground the current passing through multiple LED's. The final component being used is an Auto Gain Control electret condenser mic; the AGC electret condenser mic takes in sound from 0 to 60 decibles, and outputs 2Vpp on a 1.25V bias. Through using the `shiftout` command in the Arduino IDE, a complex animation can be displayed based on the microphones output.

Procedure

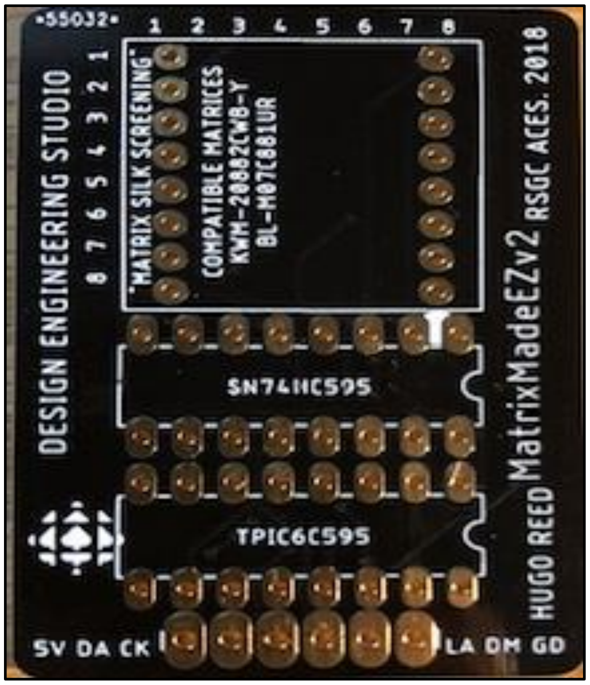
Parts Table	
Component	Quantity
MatrixMadeEZ PCB	1
8x8 LED Matrix	1
16 pin chip seat	2
SN74HC595 Shift Register IC	1
TPIC6C595 Shift Register IC	1
6 pin right angle male header	1
AGC Electret Condenser Mic	1
Arduino Uno	1
ATMega328P-PU	1
Breadboard Connector wires	14
Push Button (normally open)	2
1KΩ fixed resistor	2

Open the Arduino IDE and flash the blink sketch on to the onboard Arduino using the sparkfun pocket programmer and confirm that the LED attached to pin 19 of the ATMega328P-PU is blinking with a 50% duty cycle at a 0.5HZ frequency. Gather the MatrixMadeEZ PCB (designed by Hugo Reed) as well as the 16 pin chip seats, 8x8 LED matrix, and the 6 pin right angle male header. Solder each of the components to the PCB, careful of their orientation. Check all solder connections to make sure they coat the copper pad, and no connections are bridged. Connect the 5V and ground pin of both the ACG electret condenser mic and MatrixMadeEZ PCB to the power and ground rails on the breadboard. Connect the output of the condenser mic to analog input pin A0 on the Arduino board. Connect the data, clock, and latch pins of the PCB to digital input pins 11, 10, and 9 on the Arduino respectively. Wire 2 PBNO's with pull up resistors and connect them to analog input pin A5 and A4. Open the Arduino IDE and create a new sketch. Create 3 unsigned 8 bit integers to rename the digital input pins 11, 10, and 9 to data, clock, or latch. Use 3 unsigned 8 bit integers to create 3 arrays, each representing a different animation. In the setup function, declare the data, latch, and clock pins as `OUTPUT`. Create a sample window of 50ms to measure the `analogRead()` on pin A0 and return an average value between 0 and 1023. Using the Arduino `map()` function, map the value from 0 to 1023, to an integer between 1 and 5. Create a series of `if-else` ladders that monitor the state of the PBNO's and shift out the corresponding animation. Upload the sketch to the Arduino board and confirm that everything is functioning correctly.

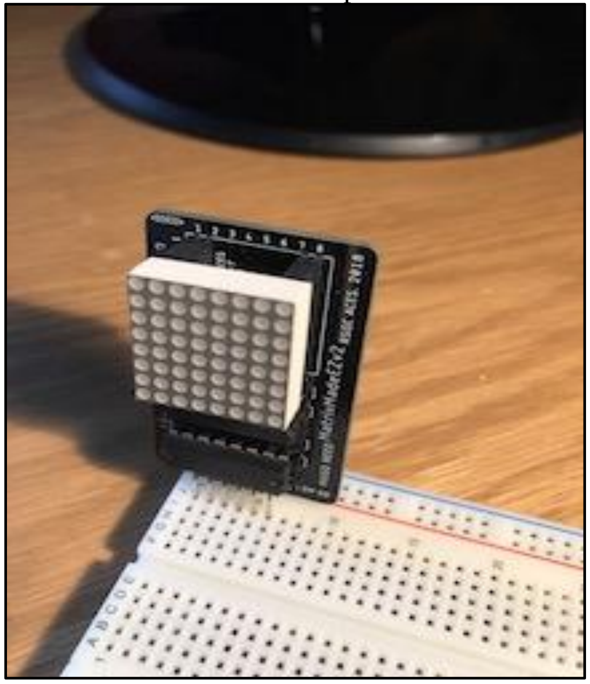
Media

https://www.youtube.com/watch?v=_cc5dbJQ5ZQ

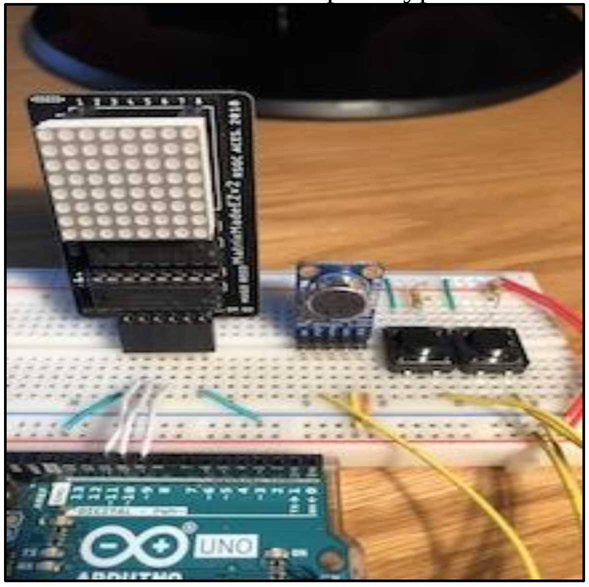
MatrixMadeEZ PCB



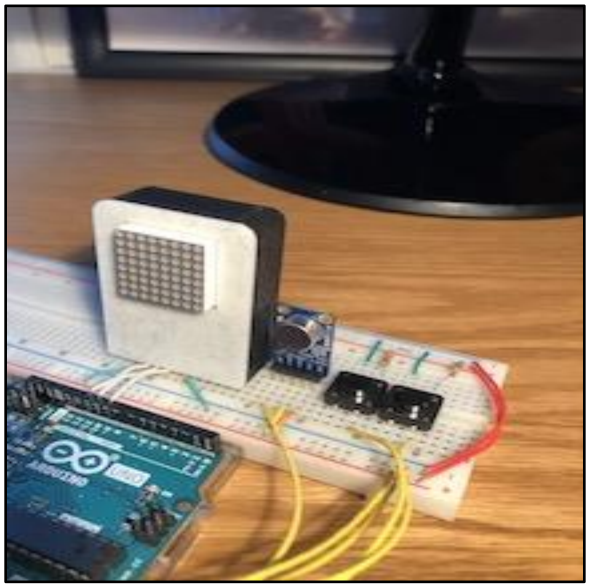
Soldered Component



Breadboarded prototype



Final Product



Code

```
//Project : To create seperate animations on an LED matrix based off of the
//readings from an electret condensor mic.
//Author : Joshua Dolgin
//Date  : 2018/11/17
//Status :Working
const int sampleTime = 50; // Sample window width in mS
uint16_t soundSample;
uint8_t data = 11;        // DA: SER IN (Serial Input)
uint8_t clk = 10;         // CK: SRCK (Shift Register cloCK)
uint8_t latch = 9;       // LA: RCK (Register Clock)
uint8_t cornerData [] = {0x80, 0xC0, 0xE0, 0xF0, 0xF8, 0xFC, 0xFE, 0xFF};
// Creates an array to represent the first animation
uint8_t cornersData [] = {0x00, 0x81, 0xC3, 0xE7, 0xFF};
// Creates an array to represent the second animation
uint8_t boxData [] = {0x00, 0x18, 0x3C, 0x7E, 0xFF};
// Creates an array to represent the third animation

void setup() {
  pinMode(data, OUTPUT);  //declares the data pin for OUTPUT
  pinMode(clk, OUTPUT);  //declares the clock pin for OUTPUT
  pinMode(latch, OUTPUT);//declares the latch pin for OUTPUT
}

void loop() {
  // put your main code here, to run repeatedly:
  uint32_t startMillis = millis(); // Start of sample window
  uint16_t soundReading = 0;   //Sets the soundLevel to 0
  uint16_t signalMaximum = 0; // establsihes a signal maximum
  uint16_t signalMinimum = 1024; // establishes a signal minimum

  while (millis() - startMillis < sampleTime)  // collect data for 50 mS
  {
    soundSample = analogRead(A0);//Read volts on analog pin 0
    if (soundSample < 1024)  //If sound sample < 1024
    {
      if (soundSample > signalMaximum) //If sound sample is > signal maximum
      {
        signalMaximum = soundSample;  //signal maximum equals sample
      }
      else if (soundSample < signalMinimum) //if the sound level is greater than the
minimum value
      {
        signalMinimum = soundSample;  //signal minimum equals sample
      }
    }
  }
  soundReading = signalMaximum - signalMinimum; // Max value- min value = soundLevel
  float voltReading = (soundReading * 5.0) / 1024;  // Convert to volts
  voltReading = voltReading * 10; //Multiplies the volt reading by 10 in order to
use integer values.
  voltReading = constrain(voltReading, 2, 24);//Constrain volt values between 2 - 24

  if (!analogRead(A5)) { // If there is low voltage on A5
    voltReading = map(voltReading, 2, 24, 1, 5);//Map the volt reading to a value
between 1 and 5
    for (uint8_t j = 0; j < voltReading; j++) {
      digitalWrite(latch, LOW); // Writes the latch pin to low
      shiftOut(data, clk, LSBFIRST, cornersData[j]); // Shifts out the corners
animation
      shiftOut(data, clk, LSBFIRST, cornersData[j]);
    }
    digitalWrite(latch, HIGH); // Writes the latch pin to high
  } else {
    if (analogRead(A4)) { //If there is a low voltage on A4
      voltReading = map(voltReading, 2, 24, 1, 8);//Map the volt reading to a value
between 1 and 5
      for (uint8_t j = 0; j < voltReading; j++) {
        digitalWrite(latch, LOW);
        shiftOut(data, clk, LSBFIRST, cornerData[j]); // Shifts out the box
animation
        shiftOut(data, clk, LSBFIRST, cornerData[j]);
      }
      digitalWrite(latch, HIGH);// Writes the latch pin to high
    } else {
      voltReading = map(voltReading, 2, 24, 1, 5);;//Map the volt reading to a value
between 1 and 8
      for (uint8_t j = 0; j < voltReading; j++) {
        digitalWrite(latch, LOW);
   shiftOut(data, clk, LSBFIRST, boxData[j]); //Shifts the box animation
   shiftOut(data, clk, LSBFIRST, boxData[j]);
      }
  digitalWrite(latch, HIGH);// Writes the latch pin to high
    }
  }
}
```

Reflection

I first came across the project.? MatrixMadeEZ page when the details of the page were still listed as TBA. I constantly checked and refreshed the page to see when the details and subsequent instructions for this project would be posted so I could begin thinking of possible ideas. Not more than a couple days after I submitted my Engineering Report for Project 8. Shift Register – Bargraph, the MatrixMadeEZ project was announced. But, in order to get the PCB and begin my project, I first had to get my onboard Arduino working, and in that began my initial struggles with this project. I ran through each component multiple times, checked to make sure the 5V regulator was indeed outputting 5V to the power rail; yet the statement “Invalid Device Signature” still flashed on my screen. I decided to take apart my third iteration of the onboard Arduino and attempt to rewire it one more time. All my frustrations disappeared when the LED began to flash the blink sketch. Once gathering and soldering my components I spent multiple hours playing with the `shiftout` command and turnary statements to play with and discover the animations possible on an LED matrix. Upon finishing my code, I believe that my project was a complete success. I found a sensor that was interesting in both its mechanics and practical use, I created animations to properly display those readings, and I never felt like I was doing work: I felt I was doing what I loved.

Project 10. Design Sessions

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Tasks.html - Design

EAGLE

Purpose

The purpose of the three EAGLE sessions is to learn how to use the basic tools and principles in designing a Printed Circuit Board (PCB). EAGLE is a computer application designed by Autodesk to help create and prepare PCB's to be manufactured. Knowing how to design and send PCB's off to be manufactured is a necessary component of being able create functioning and easy to understand prototypes and products. In these design sessions, not only are these tools and their subsequent functions taught, but also how to apply them to an actual PCB, in this case; a Seven-Segment Display tester. The Seven-Segment Display tester functions off of the same general principle of an LED tester, supplying female header pins for the Seven-Segment to plug into the board. The power is immediately supplied to all the necessary pins of the Seven-Segment, with the Common Cathode pins being tied to ground via a PBNO. When the PBNO is pushed, the Common Cathode pins become grounded, lighting all the separate LED segments (including the decimal point) and indicating which of the Segments function correctly.

Procedure

EAGLE Parts Table		
Component	Code	#
DC Jack	DCJ0202	1
Omron PBNO	40-XX	1
470Ω resistor	R-US_0207	1
CC 7-Segment Display	HD-H101	1

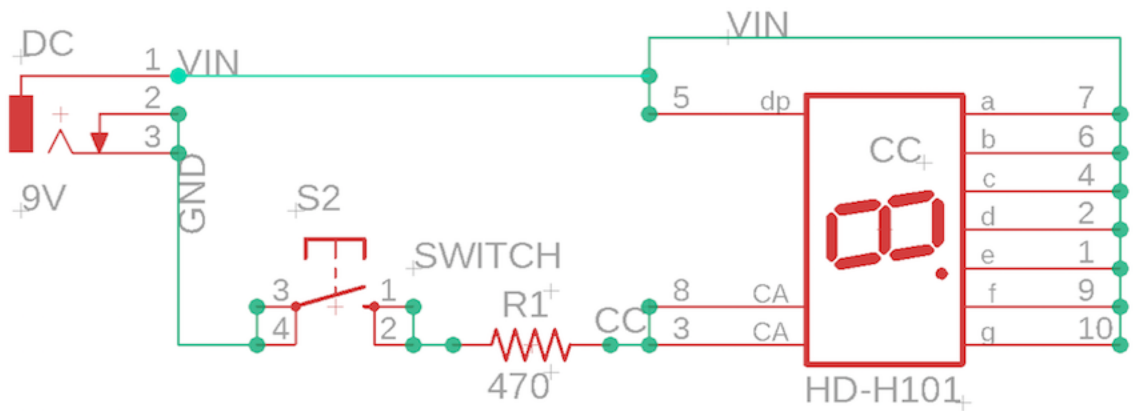
Open the EAGLE software and create a new project entitled "SevenSegmentTester". Create a project schematic by right clicking on the project and clicking "New Schematic" in the drop down menu. Download the SparkFun and Adafruit libraries from github and save them to a folder labeled "EAGLE Libraries". Within EAGLE, select "Options"-"Directories" and route the Library bar to the folder holding the previously downloaded libraries. Open the previously generated schematic and select "Add Part". Add each of the components listed in the EAGLE Parts Table by typing the code listed into the search bar. Move and rotate each of the components until they are well organized and fit within a reasonable space. Add junctions to wherever two components will be connected. Connect the power pin of the DC jack to pins dp, a, b, c, d, e, f, and g on the Seven-Segment Display. Connect both of the Common Cathode pins to one side of the PBNO via a 470Ω resistor. Connect the other side of the PBNO to the ground pins of the DC jack. Use the "name" and "value" tools to name each of the parts and display their values. Use the "label" tool to name the "GND" line, the "VIN" line, the "CC" line, and the "Switch" line. Once everything is given a name or value, all the connections are junctioned, and all the connections are correct, click on the "switch to board" button in the upper left corner of the screen. Once the board diagram is generated, group all the parts together and move them into the board area. Rotate and place all the parts within the board area so they fit well and are in the desired location. Using the "info" tool, change the values of each of the lines surrounding the parts so that the board area is now a 40mm x 40mm square. Select "Tools" then Design Rules Check ("DRC") and set the default size of the copper trace to 40mil. On the top layer(1), select the "Route" tool and connect all of the components powered by the DC jack; in this case, connect all the pins on the Seven-Segment Display that require power to the VCC pad of the DC jack. Switch to the bottom layer(16), connect the rest of the parts that require a ground connection(Switch, CC pins, resistor) to the ground pads of the DC jack. Create 4 3.04mm diameter holes and place them at a 3mm offset in each corner of the board. Use the rounded "miter" tool and mitre the edges of the board until the radius of the miter roughly matches the shape of the hole. Create a new layer labelled "tSilk" (layer 112) and select that layer in the drop down menu. Using the "line" tool, closely outline the outsides of the board in whichever color was chosen for the layer. Use the "text" tool to write creators name, and whatever else is necessary. Bring in the ACES logo as BitMap image profile (BMP) and arrange it on the board. Now the board is ready to be manufactured. Select "Cam Processor" and in the load job drop down menu, select the SparkFun Cam Processor, and a location to save the file. Load the DirtyPCB's website and drag the zip file into the "choose file" tab. Select preferred board settings and load into the board preview to see the finished product.

Reflection

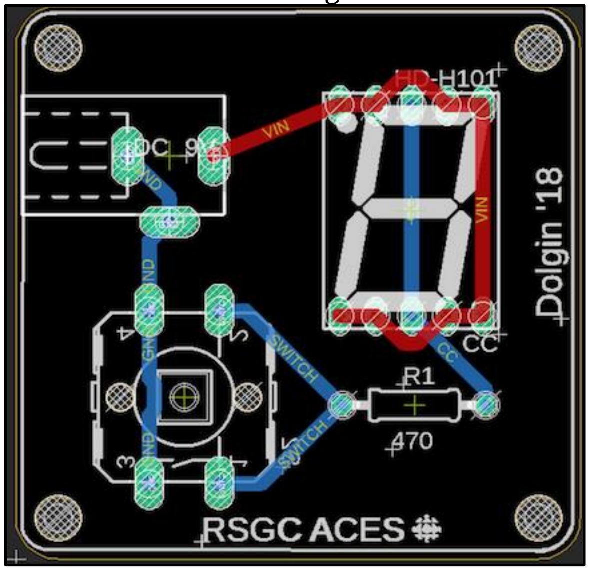
I have been interested in designing and manufacturing PCB's since I was first thinking of my ISP in grade 10. I looked into separate software design programs and tutorials but they all detailed extremely complex circuits and tools I wasn't able to understand. Upon learning the basics of EAGLE, I realized that designing was not as complex as I had imagined previously. Of course, every Computer Aided Drawing software is different and some excel at certain aspects more than others, but in my opinion EAGLE is more friendly to beginners and hobbyist, but can also support intermediate and advanced users. As well, I learned about exporting and importing DXF files to allow seamless transition between different Autodesk programs, which will help progress my 3D designing. In conclusion, in learning EAGLE, I picked up a skill necessary and useful in progressing as an Engineer, that opens up a world of possibilities for future projects.

Media

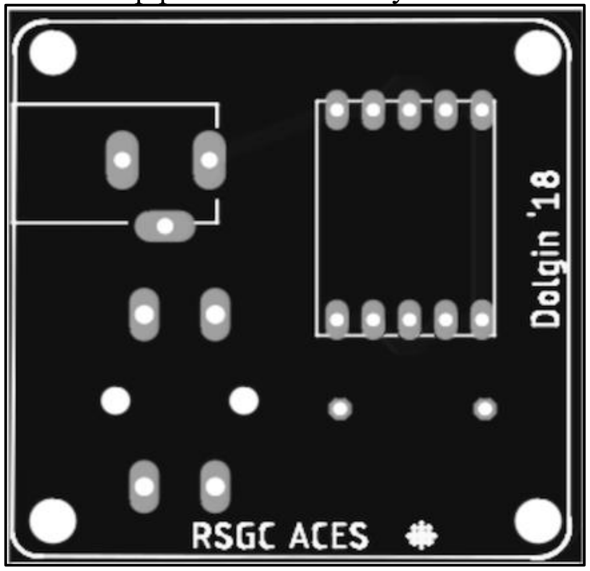
Schematic diagram



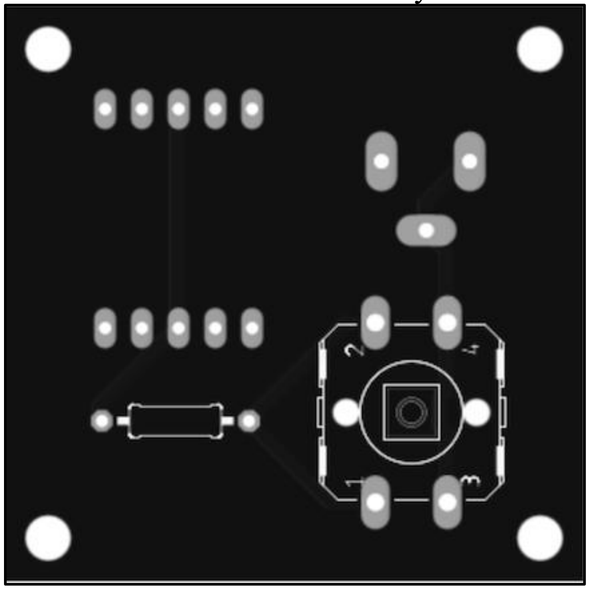
Board diagram



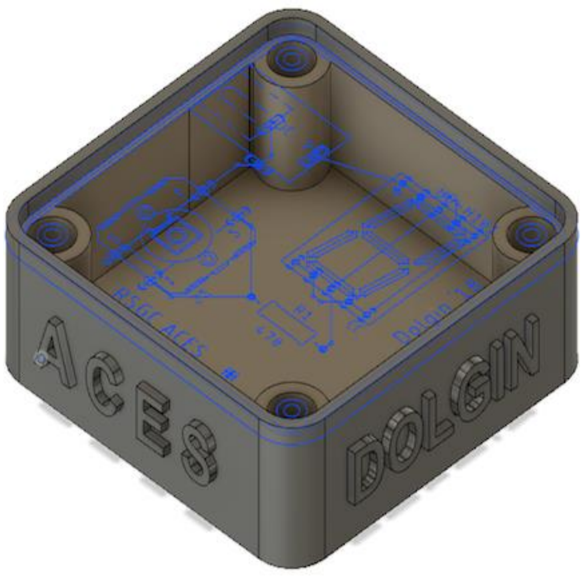
Top preview from DirtyPCB's



Bottom Preview from DirtyPCB's



Case designed in Fusion 360

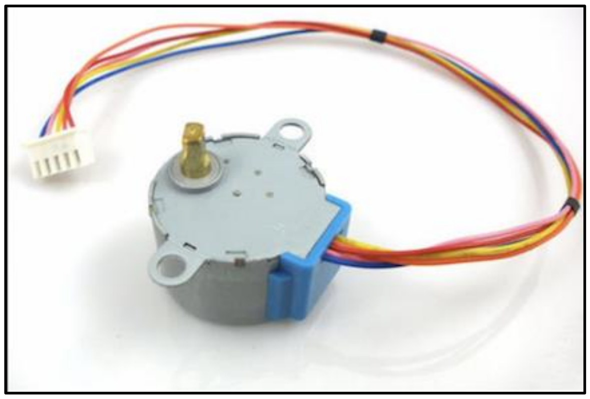


ViaCAD

Purpose

The purpose of the three ViaCAD sessions is to learn the tools and ideas when it comes to 3D designing. ViaCAD is another Autodesk Computer Aided Drawing software; but, unlike EAGLE, ViaCAD is used as a tool for 2D and 3D design rather than PCB design. The goal for these three sessions is to design a mount for the 28-BYJ stepper motor that will secure it underneath the chassis of an autonomous vehicle. The case is designed to screw or bolt underneath the vehicle chassis and hold the motor in an optimal position for the wheels to spin. 3D and 2D designing, although complicated, is yet another necessary tool for progressing the development and testing of prototypes and products. In using 3D design, complex forms can be created and tested in rapid succession. With the help of the 4 ACES 3D printers found in the back of the DES, designs can be formed and tested within hours of being created.

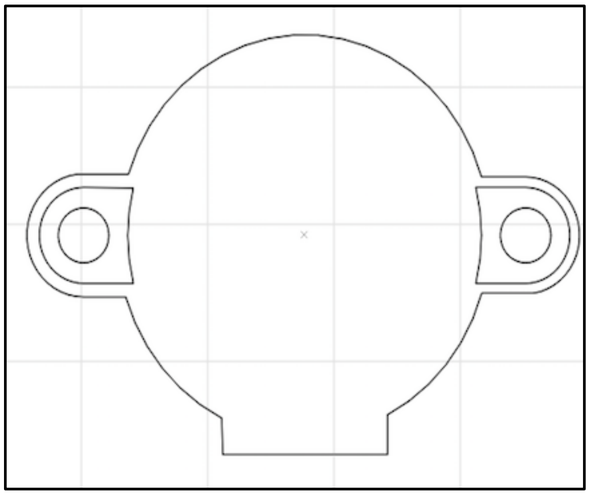
Procedure



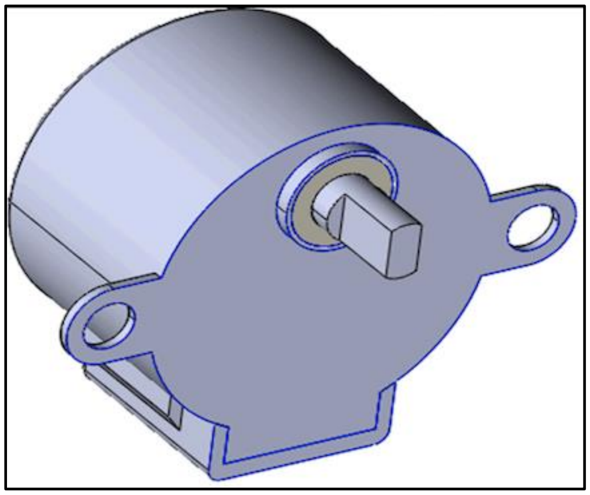
Open ViaCAD and sign in using the key provided in class. Create a new sketch entitled "StepperMotorCase" and save it to an easy to access folder. Begin a sketch and select to start on the Z plane, to do this, select "View" and then "Top" from the drop down menu. Open the document provided by Mr.Elia and follow the instructions to create a 2 dimensional sketch of the top view of the motor. Create lines 1.2mm long extending from all the joints or arc bases outwards. Connect all of these points using lines and 3 point arcs to make a shell of the existing diagram. Use the "show/hide" tool to hide the construction lines that are no longer necessary. Select all the lines, arcs, and circles that created the original motor diagram and group them together using the "group" tool. Select "View" then "Isometric" from the drop down menu. Using the "Push/Pull" tool, create a downward extrusion 2mm long, this will become the base of the motor mount. Now, using the "Push/Pull" tool, create a 19.5mm upwards extrusion. Using the "shell" tool, shell the previous 19.5mm extrusion to a depth of 1mm, this will leave a 0.2mm clearance for the motor to fit properly. Create an extrusion for the "wings" of the motor to rest on by extruding the identical shape from the original sketch, this will allow screws to be put through the 3D printed plastic to hold the motor snuggly in place. Using the "box" tool, create a rectangular extrusion on top of the motor housing that extend 20mm in both directions. Using the "cylinder" tool, create 2 3.04mm holes at each end of the rectangle, this will allow the housing to mount to the vehicle chassis.

Media

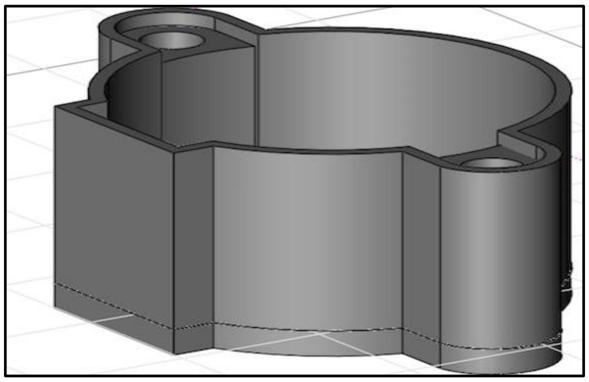
2D sketch



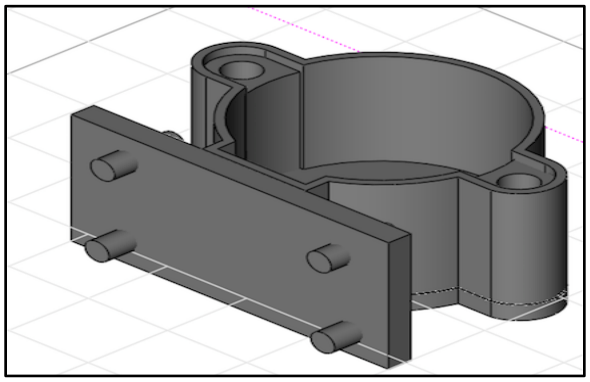
3D sketch



Shell to house motor



Mount to attach to chassis



Reflection

I have been thinking about 3D design and printing since I witnessed the case design competition in the grade 11 ACES program last year. I became instantly interested in the world of 3D printing, so I did my research and eventually bought myself a 3D printer in June of last year. I began designing and printing using rudimentary design platforms, tinkercad, OnShape, etc: but I never really felt I could design to my full potential. At the beginning of this year, I began designing in Fusion 360, and it truly felt like a landscape where I could design and produce whatever I imagined, as long as I was aware of the tools and their functions. Upon being introduced to ViaCAD, I became frustrated and pessimistic as to my abilities in using this new program: I was unaware of the keystrokes, functions, and icons. This immediate flood of frustration made me want to return to Fusion 360 for designing and prototyping the stepper motor mount. Through the help of Mr.Elia, I realized that the programs were not completely incomparable, but some simple adjustments had to be made. Since, I am still much more adept and comfortable in Fusion 360, I will most likely use it for future designs; although, I believe I am skilled enough to learn and use ViaCAD in order to help my classmates, and further my knowledge of 3D design.

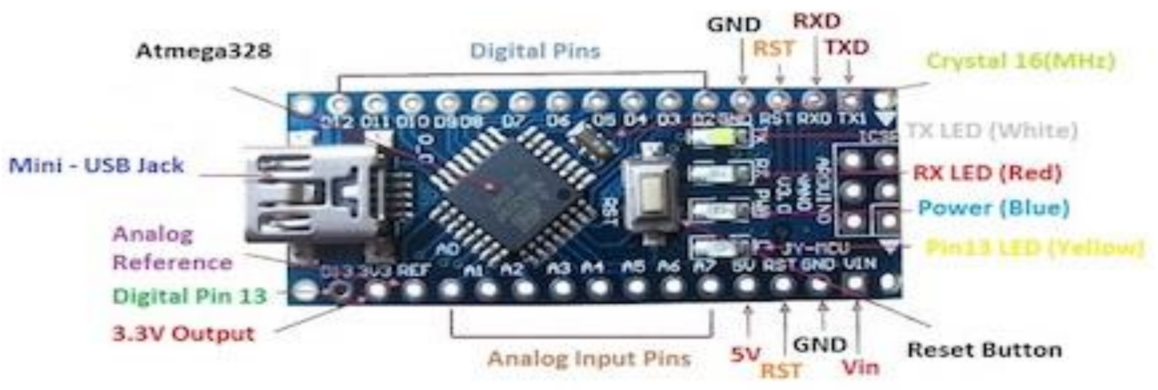
Project 11. Smart Trash Can

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/ISPs.html

Purpose

The purpose of this project is to dive deeper into concepts and components introduced up to this point, and create a unique and interesting project. Specifically, this projects purpose is to use a custom PCB, HC-SR04 ultrasonic distance sensor, a sharp IR distance sensor, a micro servo, an Arduino Nano microcontroller, code written in the Arduino IDE, and 3D design and printing. Through using these components, code, and CAD design, a trashcan can be created that serves two purposes. Firstly, an HC-SR04 will be placed to scan the front of the bin, scanning for any movement or something getting closer towards it. The HC-SR04 is an ultrasonic distance sensor, using one side to emit ultrasonic sound, that then bounces off an object and is received by the echo side. Through using a mathematical formula, this time difference can be converted to centimeter distance. If the HC-SR04 scans an object within a hardcoded threshold (5cm), then it triggers the mirco servo to lift its arm 90°, lifting the lid and placing the contents into the bin. A microservo motor is controlled by sending a pulse width modulation (PWM) signal through the control wire. There is a minimum pulse, a maximum pulse, and a repetition rate. Once something new has been placed in the tray, the sharp IR (infared) distance sensor checks the depth of the internal reservoir, and displays how full the trash can on a 10 LED bargraph custom PCB that uses two Shift Registers to allow complete control of the bargraph. As well as increasing my knowledge in the components included and coding, this project also serves the purpose of helping to further progress 3D design and CAD skills learned previously in this course.



Code

```
//Project : Smart trash can
//Author : Joshua Dolgin
//Date  : 2019/02/21
//Status : Working
#include <Servo.h> //includes servo library
#include <SharpIR.h>//Includes the Sharp infared sensor library
#define model 430 //Defines the model of infared sensor
#define ENDROTATION 0 //defines a variable to end rotation
#define STARTROTATION 90//defines variable at which to start rotation
Servo myservo;
uint8_t latchPin = 4; //Pin connected to ST_CP (latch) of 74HC595
uint8_t clockPin = 5; //Pin connected to SH_CP (clock) of 74HC595
uint8_t dataPin = 3; //Pin connected to DS (data) of 74HC595
uint8_t pos = 0; // position of the servo
uint8_t trigPin = 7; // trigger pin on HC-SR04
uint8_t echoPin = 8; // echo pin on HC-SR04
uint8_t infaredSensor = A0; //Uses 8 bit unsigned integer to rename pin A0
SharpIR SharpIR(infaredSensor, 430); //Measures the distance away
long duration; // creates a variable to measure duration
uint16_t distance; // creates an unsigned 16 bit integer to measure distance
uint8_t digitMap[] = {0b11111111, 0b00111111, 0b000001111, 0b00000011}; // creates
an array of binary values
void setup() {
  Serial.begin(9600);
  while (!Serial);
  myservo.attach(9); // attach servo to pin 9
  pinMode(trigPin, OUTPUT); // sets trigger pin to OUTPUT
  pinMode(echoPin, INPUT); // sets echo pin to INPUT
  pinMode(dataPin, OUTPUT); //Sets dataPin to OUTPUT
  pinMode(clockPin, OUTPUT);//Sets clockPin to OUTPUT
  pinMode(latchPin, OUTPUT);//Sets latchPin to OUTPUT}
void lift() {
  delay(2000); // delay 2 seconds
  myservo.write(ENDROTATION); // write the servo to the end of rotation
  delay(1000); // delay 1 second
  myservo.write(STARTROTATION); // write the servo to start position
}
void checkDepth() {
  uint8_t dis = SharpIR.distance(); // this returns the distance to the object
you're measuring
  dis = constrain(dis, 4, 8); //Constrains the value between 4 and 30
  Serial.println("Internal Distance");
  Serial.println(dis); //Prints the average distance of centimetres
  dis = map(dis, 4, 8, 1, 4); //Maps the value between 4 and 30 to an integer
between 0 and 8
  for (uint8_t j = 0; j < dis; j++) {
    digitalWrite(latchPin, LOW);  //ground latchPin and hold low for as long as you
are transmitting
    shiftOut(dataPin, clockPin, MSBFIRST, digitMap[j]); // shifts out the value in
the array at cell "j"
  }
  digitalWrite(latchPin, HIGH);//Returns the latch pin to high
  delay(1000); //delay 1 second
}
void scanTray() {
  myservo.write(STARTROTATION); //puts the servo to start of rotation
  digitalWrite(trigPin, LOW); // writes the trigger pin low
  delayMicroseconds(2); //delay 2 microsecond
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH); //writes the trig pin high
  delayMicroseconds(10); // delay 10 microseconds
  digitalWrite(trigPin, LOW); // write trigger pin low
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2;
  // Prints the distance on the Serial Monitor
  distance = constrain(distance, 0, 30);
  Serial.print(" External Distance ");
  Serial.println(distance);
  if (distance <= 5) {
    lift(); //execute lift function
    checkDepth(); // check depth
}
}
void loop() {
  scanTray(); // execute the scan tray function
  checkDepth(); // execute the check depth function
}
```

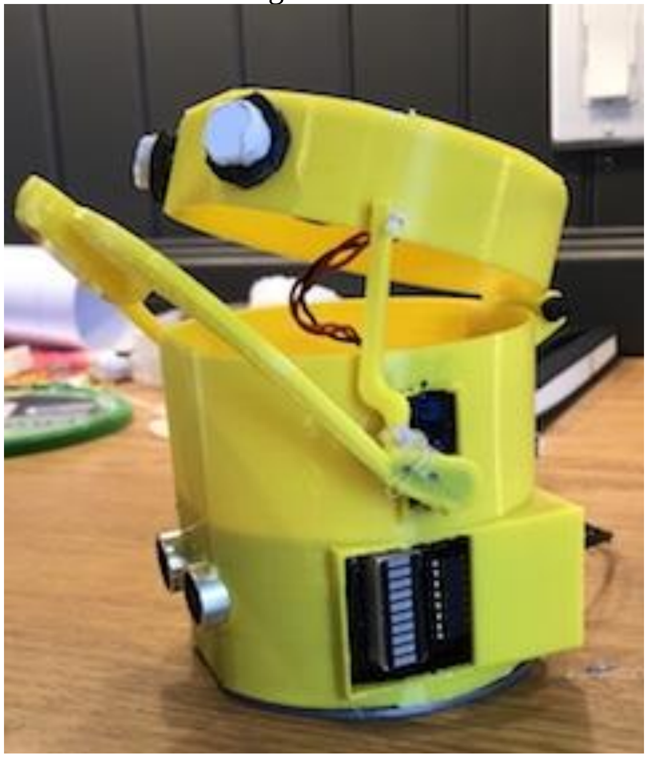
CAD

There are two elements of Computer Assisted Drawing software used in this project. Firstly, EAGLE is implemented to design and prepare the PCB for this project. The goal of the PCB is to reimagine Tim Moorland's Shift Register bargraph, and implement a second 595 Shift Register IC to light up all 10 LED's rather than 8. To do this, bring all the components required (11 pin bussed 330Ω resistor network, 10 LED bargraph, 2 SN74HC595 Shift Register ICs, and a 5 pin header) into the schematic diagram and attach each of the VCC and GND pins to power and ground respectively. Tie the overflow pin of the first Shift Register (pin 9) to the data signal pin of the second Shift Register. Attach the first 10 output pins to pins 1 through 10 on the LED bargraph. Convert the schematic to a board diagram and arrange the pieces so they fit well and are aestetically pleasing. Route the connections using the top and bottom layers of the PCB, until everything is connected to what it should be. Add silk screening to the header pins identifying their purpose. Round the edges of the board using the mitre tool. Using the sparkfun CAM processor, export the CAM files as a ZIP file containing everything required. Load the files into a PCB website and select board thickness, color, coating type, etc. The second form of CAD software employed in this project is Autodesk Fusion 360, a 3D design software. Open Fusion and create a new sketch, then select a name when prompted. Begin a sketch on the top plane. Draw a circle of diameter 90mm and a line roughly at the ¾ mark of the circle. Offset this shape by 0.2mm, 1.4mm, and 2.6mm. Extrude the piece offset by 1.4mm 50mm upwards, then round off the top using the fillet tool, so that an enclosed object is created, this will house the Arduino and some hardware components. Along the flat edge, draw 2 16mm diameter holes 31mm apart, this will house the HC-SR04 sensor. Create a tangential plane along the side of the cylinder, perpendicular to the flat edge. Along this plane, sketch a rectangle that is 23.4mm tall, and 54.2mm wide, extrude this piece into the cylinder and use the cutting tool to create a place to house the PCB. Above the PCB slot, create a rectangle with the dimensions of the servo motor used, and offset that by 0.2mm. Extrude that inwards to create a cut out for the servo motor. Create a lid using the same dimensions as the cylinder, and create a hinge with a 1.2mm hole in the middle so the lid and the body can be joined. Create a rotating arm to be placed on the servo with a semisphere shaped placement tray resting in front of the ultrasonic sensor. Create a bar extending from the servo arm to the lid, held in place by two holes, this will lift the lid when the servo arm is raised. Finally, use chamfers and fillets to add aesthetics and structural integrity. Print each piece seperately, no infill and some supports is required.

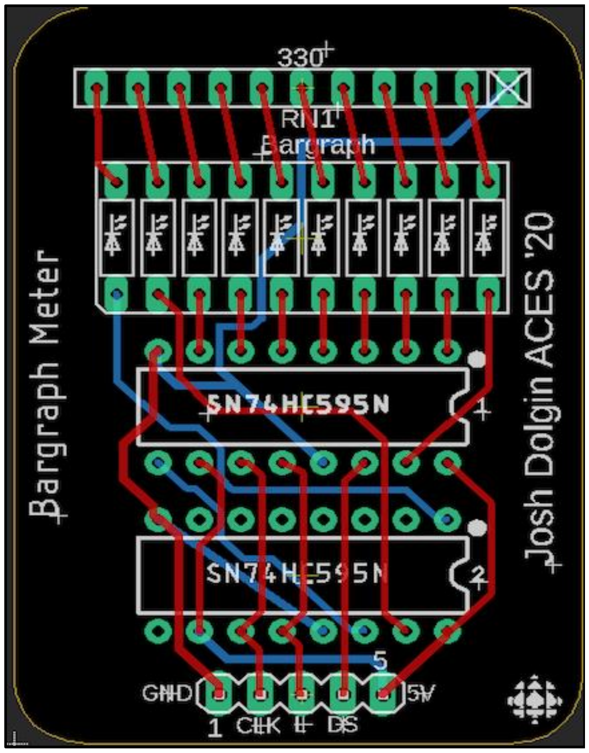
Media

https://www.youtube.com/watch?v=oCvCiEcJl0o

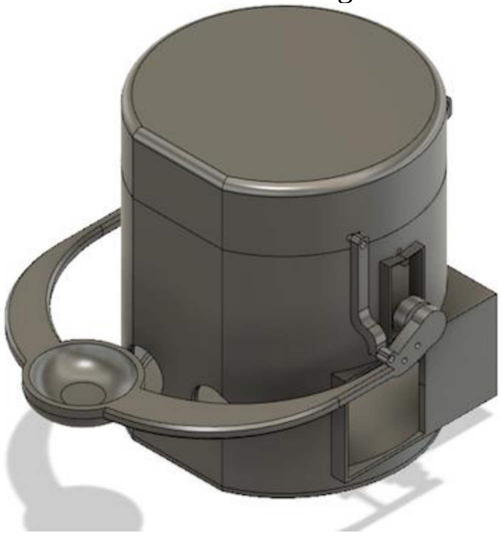
Final Design with arm lifted



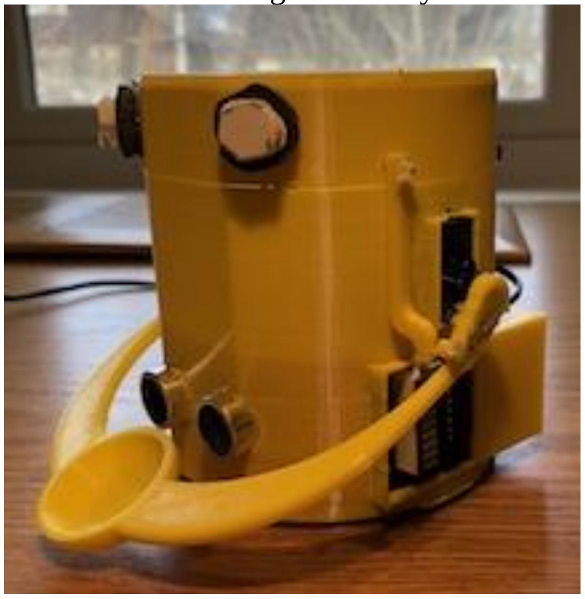
PCB in EAGLE



Fusion 360 design



Final Design stationary



Procedure

Parts Table	
Component	#
HC-SR04	1
Sharp IR distance sensor	1
Micro Servo motor	1
Custom PCB	1
11 pin bussed 330Ω Resistor Network	1
10 LED bargraph	1
SN74HC595 Shift Register	1
5 Pin male header	1
Arduino Nano	1
½ Breadboard Perma-Proto	1
3D printed pieces	6
Assorted Femal-Male wires	15
5V DC power supply	1
DC power jack	1

Gather all the components listed in the parts table. Solder the male header pins into the Arduino Nano, and place it into a ½ size breadboard. Connect the female headers to the male headers of the other components. Connect the input wire of the servo motor to pin 9 of the Arduino. Connect the trigger and echo pins of the HC-SR04 to pins 7 and 8 respectively. Connect the output wire of the Sharp IR distance sensor to analog input pin 0. Connect the latch pin of the PCB to pin 4, the data pin to pin 3, an the clock pin to pin 5. Connect all the power and ground wires to the power and ground rails of the breadboard. On a small square of through hole solder board, solder 2 wires to the power and ground pins of the DC jack, this will supply the power to the Arduino and components. Attach the Vin pin of the Arduino to the power rail and the GND pin to the ground rail. Open the Arduino IDE and create a sketch entitled "ISP_MEDIUM". Include the `servo.h` and `sharp IR.h` libraries. Use the `#define` function to define the Sharp IR model number, as well as the start and end rotation positions of the servo. Use an unsigned 8 bit integer to rename the pin numbers to which components are attached to the names or functions of those pins. Create 2 unsigned 16 bit integers labelled "distance" and "duration". Create a 4 cell `array` that includes 4 binary values (in decimal 255, 63, 15, and 3). In the setup function, declare the trigPin, dataPin, clockPin, and latchPin for `OUTPUT`, and the echoPin for `INPUT`. Create a `lift()` function that delays for 2 seconds, lifts the servo to the end of rotation, delays for another second, then returns the sevo arm back to the beginning of rotation position. Create a `scanTray()` function that will scan using the ultrasonic sensor, and once an onject is perceived within a 5cm distance, employs the `lift()` function. Create a `checkDepth()` function that checks the distance on the inside of the trash can using the sharp IR distance sensor, the shifts out that value as a cell in the `array`. Upload the code to the Arduino and confirm everything is working as intended. Once everything is functioning correctly, solder the circuit in the same orientation on a ½ breadboard perma proto. 3D print all the pieces required and arrange them in order of inner to outermost application. Press the components into their designated spot in the 3D printed pieces, use some hot glue to secure them if necessary. Connect the rest of the printed pieces using string and zip ties, and confirm everything is rotating correctly. Plug in the circuit and confirm it still works. Press the bottom plate into place and secure it with a little more hot glue to close off the design.

Reflection

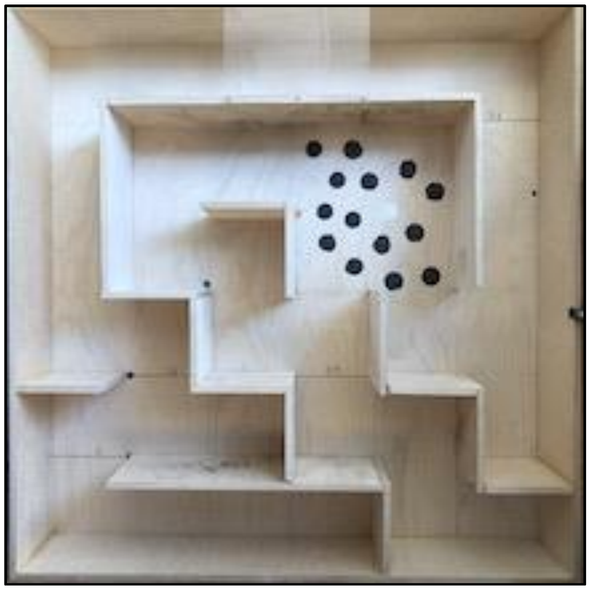
In my opinion this project was one of the best ways to challenge myself enough so that I could succeed, while feeling immense joy and being proud of myself upon completing it. I chose to do this project so I could further enhance my 3D design skills and writing modular code. Since the hardware in this project is not too intensive, I said to myself that the design and the code had to be impressive. This was one of the first times I've ever written modular code, which seemed weird at first, but made it entirely more easy to understand and find mistakes. At the end of this project, I believe that my code was written to the best of my ability and serves its purpose extremely well. Most of my bumps in the road when creating this project came in the CAD aspects, some of my components were hard to design around, so it was difficult to create an easy to print structure that could hosue everything required, and it took me multiple attempts. As well, due to the unfortunate circumstances regarding Chinese new year, my PCB has not yet arrived, yet my case is designed to fit it perfectly. In conclusion, even though I found this project hard to continue and envision a working solution at times, I believe I used my skill sets in each of the domains to the best of my ability and I am extremely proud in the product I was able to produce.

Project 12. The ACES Rover Project

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Tasks.html - Rover

Purpose



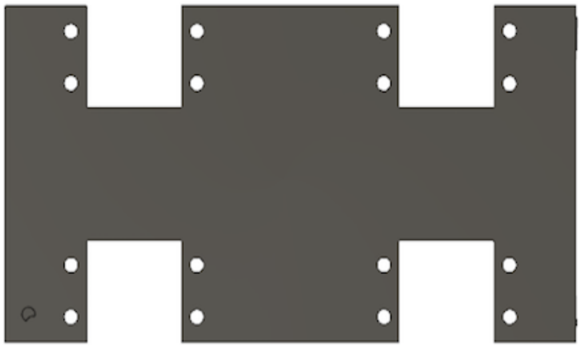
The purpose of this project is to work together as a team with two other ACES to create an autonomous
rover that will maneuver itself through an obstacle course. There are very little restrictions on the components used, as long as the only thing the rover communicates with is itself. As well, there is a restriction limiting the dimensions of the vehicle, it must be equal to or less than 16cm in the X, Y, and Z directions. Through being given this task ACES are forced to adapt their mindset to work with others, taking responsibility for either their own procrastination, or dealing with the procrastination of group members. This project is designed for each team member to serve a specific purpose in the domain which they chose to be the one they are most passionate about; whether it be Design, Hardware, or Software. By doing this, each group member can further their skill in the domain they feel most passionate about, as well as learn key information about the other two domains from their group members.

Procedure

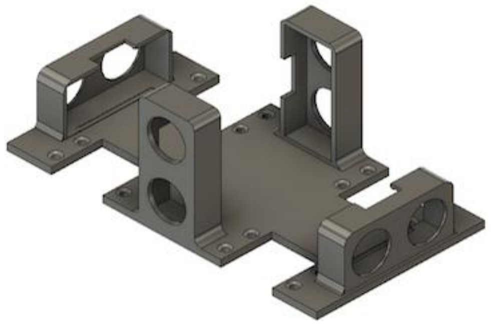
Gather all the hardware components required and lay them out. Following the hardware procedure, connect all of the necessary components to the Arduino Nano VIA a ¼ perma proto board. Connect all of the external components to a common power and ground. Finally, connect all of the motors to the motor drivers. Using the Arduino IDE, flash the code onto the Arduino Nano board and ensure that everything is working correctly as intended. 3D print each of the necessary pieces according to the settings and programs outlined in the Design section. Once all the pieces are printed, lay them out and ensure everything is there. Gather 24 bolts and hex nuts, these will secure the motors to the chassis and motor mounts. Slide each motor mount onto the motor, so that the bolt holes on the design and the motor line up. Put a bolt through each hole and secure them in place using a hex nut. Slide each motor into place on the chassis and make sure that each of the holes in the chassis line up to the holes on the motor mounts. Using the final 16 bolts and nuts, secure each motor to the chassis. Gather the 4 HC-SR04 ultrasonic sensors and press them into place, with the right angle leads lining up to the rectangular cutouts in the mounts. 3D print a shell of the body, using 5 perimetres and 25% infill to make sure that it will retain its shape throughout the vacuum forming process. Using the mayku formbox, create a vacuum form of the body. Trim the edges of the plastic using a sharp exacto knife and make any size adjustments if necessary. Secure the body to the chassis using hot glue. Plug in the two 9V batteries to supply power to the rover. Finally, test that the rover is functioning correctly by doing a test run of the course, change any software or hardware if necessary.

Design

Chassis below



Rover Chassis above



Open Fusion 360 and create a new file entitled "RoverDesign". Create a sketch on the top plane. In the sketch, create a 18.6mm by 22.5mm rectangle and offset it by 0.2mm and by 1.8mm. Fillet the edges of the rectangle to a radius of 0.2mm. Extrude the outermost shell to a distance of 10.6mm. Create a sketch on one of the side of the extrusion. Within the sketch, create 2 holes with a diameter of 3.04mm, one on the left side of the rectangle and one on the right side. Dimension the centre of each circle so that they are 5.3mm from the bottom and top edge and 2.00mm from their respective side. Using the "Cut" tool, create a cylindrical cutout that creates 4 total holes, 2 on each side of the mount, this is where the mount will be bolted into the motor. On the top face of the rectangle, create a sketch consisting of 4 3.3mm wide and 6.6mm long rectangles, one on each corner of the face. In the middle of each rectangle (dimensioned 3.3mm from the right side and dimensioned 1.65 mm from the top edge) create a centre point circle 3.02mm in diameter. Repeat this process on the other 3 rectangles. Extrude each of the rectangles (excluding the circles) 5mm downwards, this will allow the motor mount to be attached to the chassis. Fillet the edge of each hole what screws will be going through and chamfer the outside edges of the mount for ease of printing. Using the "Cut" tool in the middle of one of the edges of the rectangle (perpendicular to the motor holes), create a 2mm wide cutout in the centre of the edge to allow some flexibility when putting the motor mount onto the motor. Create an offset plane, offset 35mm from the flat side of the motor mount. Using the "mirror" tool, mirror the body just created, making a second motor mount 70mm away from the first; this will allow for space between the motors for electronics, wiring, and stability. Create a second offset plane, perpendicular to the other plane, and offset this plane by 55mm. Using the "Mirror" again, mirror the two motor mounts, creating 4 total, spaced 70mm in the X and 110mm in the Y. Create a center point rectangle that begins where the two perpendicular planes meet and has dimensions of 140mm by 70mm. Create rectangular and circular cutouts for the motor mounts and bolt holes, allowing the motors to slide into place and be bolted to the chassis. Extrude this rectangle to a height of 3.5mm to increase the strength of the chassis. On the front of the chassis, create a sketch on the front face. Within the sketch create a rectangle that is 45.40mm wide and 20.51mm tall. Offset this rectangle by 0.2mm and 1.8mm. Within the rectangle, Create 2 vertically aligned circles with a diameter of 17mm, dimensioned to be 1.70mm away from their respective side. Extrude this shape to a distance of 10mm, this will house the HC-SR04 ultrasonic sensor. Create a midplane between the front and the back of the rover and mirror the ultrasonic sensor mount so now there is one on the back of the rover as well. Using the same dimensions and steps, create vertical ultrasonic sensor mounts centered on the right and left side of the chassis, using a new midplane between the left and right sides of the rover. Finally, using the "form" tool, create a freeform body to encase the electronics of the rover, ensure that the body will avoid the motors and each of the HC-SR04 sensors. Save each individual body as an STL file and bring them individually into Ultimaker Cura. Print each body at 50% infill and 4 perimeters to ensure rigidity and strength.

Hardware

Parts Table	
Component	Quantity
Arduino Nano	1
DC motors	4
9 volt battery	2
Assorted wires	30
1/4 Perma proto board	1
DC motor driver	1
Rubber wheels	4
HC-SR04 sensors	4
Female DC power jack with wires	2

The hardware component of the rover is relatively simple. It consists of four DC motors, four HC-SR04 ultra-sonic sensors, a four terminal DC motor driver, and an Arduino Nano. Because each motor has two terminals, making eight wires that need a terminal on the four terminal driver, some changes had to be made to the wiring component. Two motors on each side were daisy chained, so that the positives and the negatives of each motor were soldered and heat shrunk. Daisy chaining the motors to the other on the same side, that reduced the amount of terminals need from eight to four, perfect for the DC motor driver. The ultra-sonic sensors have four male pins, all of which were connected to the on board Arduino. The same problem occurred with the power and ground wires of the sensors as with the motors, there are not enough pins and terminals to power all of them. There are only two ground pins and two power pins on the Arduino, and the sensors need four of each. To solve this problem, eight wires (the four power wires and four ground wires) were joined to two central wires, creating and central power and ground pins. In doing so, the need for four VCC and four GND pins was reduced to one of each, eliminating that problem. Two wires, ground and power, were soldered to the Nano's VIN and GND pins to supply voltage to the ultra-sonic sensors. The DC motor driver was connected to four analog pins to enable the motors speed to be controlled. Motor A1 and A2 terminals control one sides motor, and motor B1 and B2 terminals control the other. Having each side controlled separately allows the rover to turn similar to a tank. More about the motor control will be touched upon in the software section.

Software

The code used to control the rover consists of reading from multiple ultrasonic sensors to know direction, moving forward and adjusting speed to stay straight, and turning based off distance values. These elements allow for the rover to be programmed to navigate any course with a different layout than the course that was navigated in the competition. To read distance from the ultrasonic sensors, the `readDistance()` function was made to simply manipulate the specified trigg pin depending on which direction needed to be sensed. This function is called in `readDirection()` where one distance reading is made of 30 averaged sensor readings. In `readDirection()`, the sensor readings are then constrained and mapped to create the final readings used for course navigation. The `constrain()` call determines the sensitivity of the sensor and the `map()` call puts the readings in a 1-10 range for simplicity. The `turnOrStraighten()` function is dual-purposed as it allows the rover to turn or keep straight when the rover is moving forward. These two features can be manipulated with the distance conditions located in the if statements used for right and left distances. The `forward()` function activates the pin pairs for the right and left sides of the rover using `analogWrite()` to manipulate speed while making the other pin low for each side.

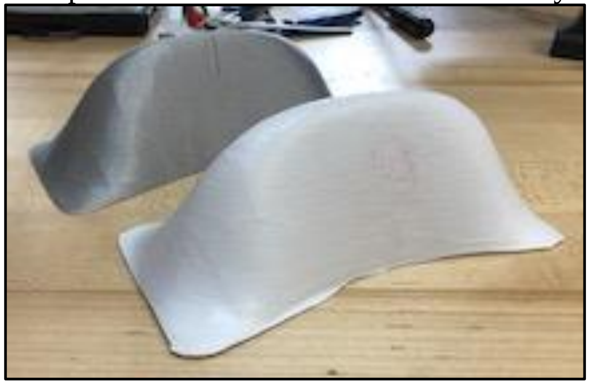
Code

```
// NAME          :Simon Peterson
// INSTRUCTOR    :Mr. D'Arcy
// COURSE        :ICS3U
// PURPOSE       :Semi-Autonomous ACES Rover Code that provides all functions needed
to adjust for any course layout
// DATE          :2019/03/9

uint8_t speed = 150; // 0-255 range for speed
#define goTime 1000 //sets forward duration

//front
#define triggF 5
#define echoF 6
//right
#define triggR 10
#define echoR 11
//left
#define triggL 3
#define echoL 2


uint8_t frontDistance;
uint8_t leftDistance;
uint8_t rightDistance;
uint64_t constrainedSensor;
long echoDuration; //long due to too much variation to use unsigned integers

void setup() {

  Serial.begin(9600);
  while (!Serial);
  //right side motors
  DDRD = (1 << PD3) | (1 << PD4);
  //left side motors
  DDRB = (1 << PB1) | (1 << PB4);
}
void loop() {
  /*Distance sensing, turning, straightening, and going forward functions
    can all be called in a certain order depending on the course layout
  */
}

void readDirection() { //change distance variable for each direction when calling

  echoDuration = 0;
  for (uint8_t i = 0; i < 30; i++) {
    readDistance();
  }
  echoDuration /= 30; //divide by 30 to average and smooth readings
  constrainedSensor = constrain(echoDuration, 0, 2500); //upp bound can be changed
to make more/less sensitive
  frontDistance = map(constrainedSensor, 0, 2500, 1, 10); //maps to new range for
course
  Serial.println(frontDistance); //print sensor values
}

void turnOrStraighten() {
  if (rightDistance == 1) {  //change distance tolerances to either turn or
straighten
    PORTD = 0 << PD4;
    analogWrite(3, speed);
    speed = speed + 10;
  } else {
    speed = 150;
  }
  if (leftDistance == 1) {
    PORTB = 0 << PB4;
    analogWrite(9, speed);
    speed = speed + 10;
  } else {
    speed = 150;
  }
}

void forward() {
  analogWrite(3, speed);
  analogWrite(9, speed);
  PORTD = 0 << PD4;
  PORTB = 0 << PB4;
  delay(goTime);
}
void readDistance() {
  digitalWrite(triggF, HIGH); //trigg & echo can be changed to specify direction
  delayMicroseconds(10);  //pulse length
  digitalWrite(triggF, LOW);
  echoDuration += pulseIn(echoF, HIGH); //30 readins are added up when called in
readDirection()
}
```

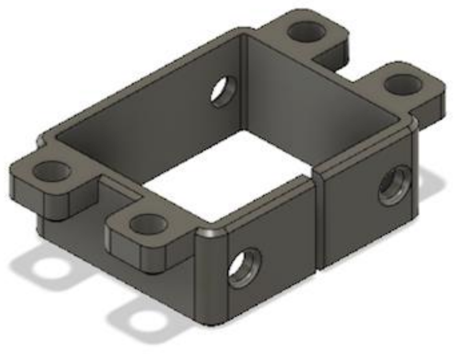
Media

https://www.youtube.com/watch?v=BiKr3W78RNk

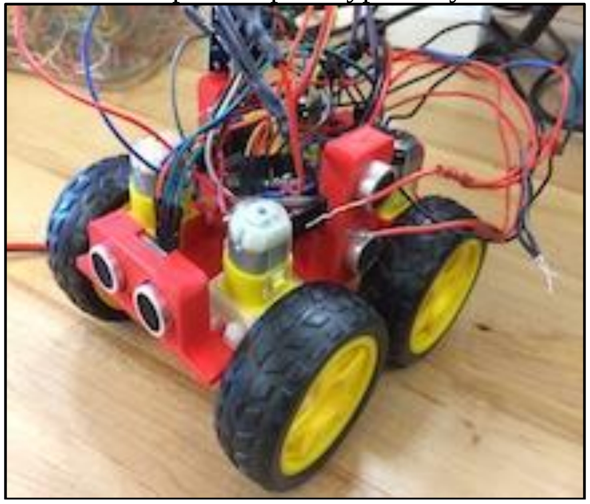
3D printed and vacuum formed rover body



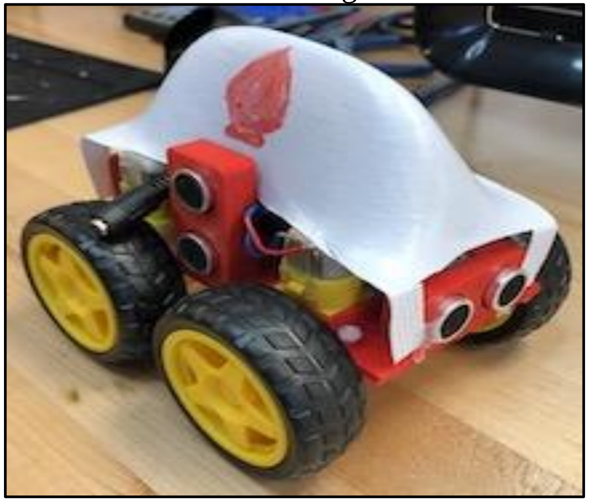
Motor mount



3D printed prototype body



Final design



Reflection

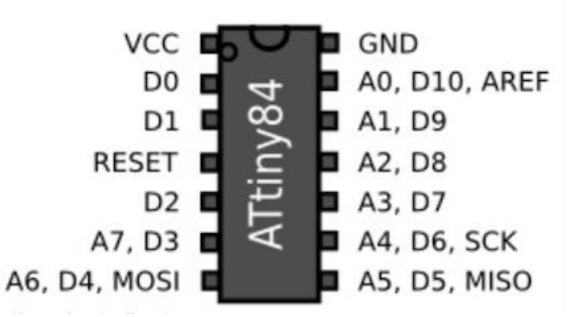
When reflecting upon this project it is hard to find ways in which I consider my teams project a success. I consider whether the design I created for the rover was well done, both in terms of overall design, and my ability to incorporate my group members requests for places to fit parts and shape of the vehicle. However, I believe that the reason our groups rover did not do as well was due in part to my procrastination. I went through multiple iterations of the design, and in my opinion, I finished with enough time for my other group members to do the rest. It was easy to confront hardware and software issues by simply stating "I am just the design guy", and upon reflecting I realize that I definitely should have been more involved in the other domians of this project rather than just the one I was assigned. I normally like to work by myself on projects, simply relying on my own skills and work ethic to either finish or not, which I believe is the conditions in which I work the best. Although I prefer working alone, a group hardware project was a new and exciting project that I was happy to take a part in, it helped me learn more about choosing people to work with, and actually having people rely that you will hand them a polished product. As well, when writing a group DER I found it especially challenging to incorporate my teammates writing into my own report, as I couldn't get everything to be properly spaced, so unfortunately some gaps are left at the bottom of my pages. On top of working on the rover with my group mates, I also had to maintain the 4 3D printers working in the DES. Unfortunately one of the printers stopped working near the beginning of this project, so only the 3 other printers were in commission. It was interesting to see every groups design process and final designs, I learned alternative ways of thinking about problems from a design perspective which was really great. So in conclusion, although my group wasn't as successful as planned, I learned from my mistakes, and hope to further progress my skills in all 3 domains for the rest of the year.

Project 13. Legacy PCB/Appliance: ATtiny Arduino

Reference

https://github.com/damellis/attiny
https://42bots.com/tutorials/programming-ATtiny84-attiny44-with-arduino-uno/
http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Tasks.html#LegacyPCB
https://learn.sparkfun.com/tutorials/pocket-avr-programmer-hookup-guide/all
https://dirtypcbs.com/store/pcbs

Inspiration

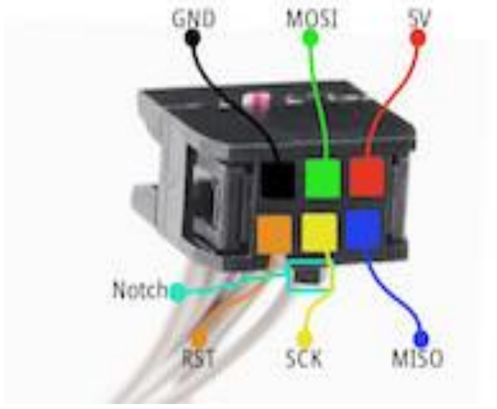


The inspiration for this project came from me looking at a multitude of shrunken boards and projects online and on the ACES home page. I quickly realized that the ATiny is a very useful IC, capable of basic Arduino programming and control. It then came to mind that this board could be used to make a micro-Arduino of sorts, useful for small projects where the size or cost of an Arduino or ATMega328P-PU would be unecessary. I then thought it would be a good idea to add female headers so that the board could function similar to an Arduino, an ISP header so the IC can be programmed on board, and finally an on board 5V regulator with DC jack input. Using these 3 tools, a simple yet effective PCB can be created that can serve the purpose of a programmer for the ATtiny24/44/84, or a smaller Arduino board to test the capacbilities of your circuit with the ATtiny before going off-board.

Procedure

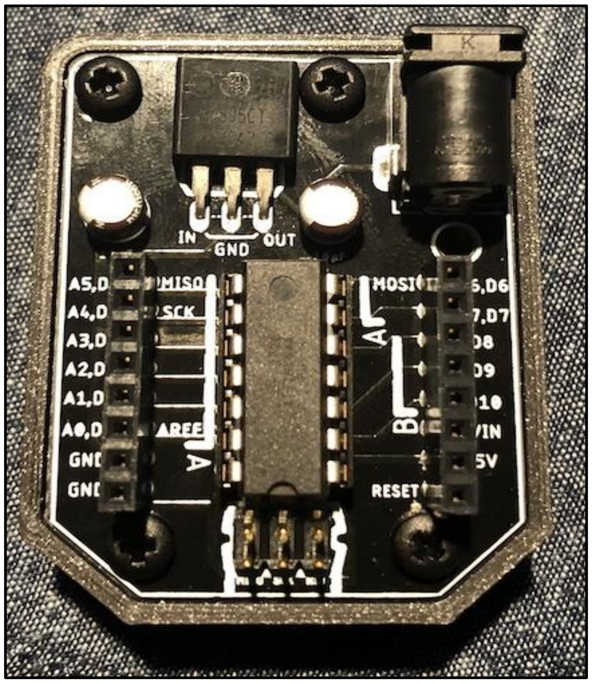
Parts Table		
Component	EAGLE Code	#
5V regulator	78LXX	1
ATtiny24/44/84 IC	DIL14	1
8 Pin header	FE08-1	2
10μF Capacitor	CPOL-US	2
6x2 ISP header	MA03-2	1
DC Jack	DCJ0202	1

Gather an ATtiny84 IC, some assorted wires, and a breadboard. Using the sparkfun pocket programmer, and the ACES AVR ISP breakout board, connect the MISO, MOSI, RESET, SCK, 5V, and GND pins to their respective pins on the ATtiny. Open the Arduino IDE and open the blink sketch. Select the "Sketch" option from the toolbar and then select the "manage libraries" option from the drop down menu. Include the damellis ATtiny library, and ensure it is installed. Within the sketch, change the `LED_BUILTIN` to an unsigned 8 bit integer defining pin 10 as "LED". Within the IDE, select "board"-"ATtiny84"-"1Mz internal oscillator". Upload the sketch and connect a 5mm LED to pin D10 of the ATtiny VIA a 330Ω resistor, and connect the cathode leg of the LED to ground. Once the LED blinks on a 50% duty cycle at a ½Hz rate, change the delay values within the sketch to make sure that everything is working properly. Open EAGLE and create a new schematic diagram, include all the parts listed in the parts table above. Arrange all the components so they are easily discernable and can be connected to other components with ease. Connect the GND pins of DC jack, 6x2 ISP header, ATtiny, 10μF Capacitors, and 5V regulator. Label one of the pins on the 8x2 headers as "VIN", and connect that to the VCC pin of the DC jack, the "IN" pin of the 5V regulator, and the VCC pin of one of the capacitors. Connect the "OUT" pin of the 5V regulator to the VCC pin of the other capacitors, as well as the 5V pin of the ATtiny, and the 6x2 ISP header. Connect the MISO, MOSI, VCC, GND, SCK, and RESET pins of the 6x2 ISP header to their respective pins on the ATtiny. Connect each pin on the ATtiny to a single pin on one of the 8x2 headers, this is so the IC can be used to test programming while on the PCB. Now, select the "generate board" option in the toolbar. Drag each component into the board view and using the "info" tool, create a 5cmx5cm square. Arrange the components in an aestically pleasing order, while ensuring it will be easy to connect all the traces. Once all the parts are organized, shape the board while ensuring it remains less than or equal to 5cmx5cm. Using the DRC, set the default trace size 20mil. Use the "Auto Router" tool to route the connections. Change any connections or traces you see necessary, ensuring that the connections are not too close to the copper pads or edge of the board. Use the "name" and "value" tool to name and value each components, and drag the writing into a visible and legible place. Add any other silk screening necessary, including pin names, ACES logo, board outline, designers name, etc. Open the cam processor, and using the "ACESLegacyPCB.Cam" select all necessary layers to export, such as; "tsilk", "200 BMP", "bsilk", "dimension", etc. Drag the exported ZIP file into the DirtyPCB's website, and select necessary board color, trace material, and board width.



Procedure Phase 2

Once the boards are received, remove them from their vaccum sealed packaging and ensure the board matches the gerber file previewed on the dirtyPCB's website. Lay out the components required to solder two boards, this preparation to test a second board should the first board tested not fuction. Firstly, solder the 9V DC jack, 5V regulator, and the 2 10μF capacitors, ensuring the correct orientation of the parts. Also, the 5V regulator and 10μF capacitors can be bent down to lie against the board, reducing the overall profile of the board. Once those components are soldered in, connect a 9V battery to the DC jack and using a DMM, ensure the output is 5V by probing the 5V pad and the ground pad on the board. After the power supply and regulation parts are soldered, solder the chip seat and 2 1x8 female headers into their respective position, making sure the notch on the chip seat aligns with the notch on the silk screening. Finally, solder the 2x6 female header into position. Insert an ATtiny84 into the chip seat, aliging the notch of the IC with notch of the chip seat. Gather the MatrixMadeEZ board and electret condensor mic breakout board used in the MatrixMadeEZ project (see project 9). Insert the MatrixMadeEz into female header pins such that the ground pin is connected to the ground header, and the other 5 pins span from A0-A4. Also, insert the microphone as an



appliance spanning pins A5-D10. Open the sketch previously used in the MatrixMadeEZ project and change the pinouts to coincide with the new appliance. Program the board by using the sparkfun AVR pocket programmer, attaching the ISP header to the 6x2 male header at the bottom of the board, the direction of the cable is marked by the two silk screening lines coming from the bottom of the header. Upload the sketch, using the ATtiny84 at 8MHz internal oscillator settings. Plug a 9V battery into the DC jack and confirm everything is working as intended.

Reflection

I believe the most difficult part of this project for me was coming up with an idea to begin, I wanted something useful for me, but also future ACES (a legacy). I then realized that many of our projects and some of my plans for future projects involved using an ATtiny 14 pin micro controller, whether it be the ATtiny84, 44, or 24. There my idea came to fruition, I quickly looked into ATtiny Arduino libraries, how to program them, as well as their usefulness in hardware projects. Although the ATtiny 84 is only capable of `digital` and `analog reads` and `writes`, `shiftout()` commands, and `delay()`, there is a lot of possibilities using only those tools. So I thought that this would be an awesome project, just an ATtiny24/44/84 programmer. I then quickly realized it would be very useful to have female header pins connected to the outputs of the IC, allowing prototyping to be done on board. I became slightly concerned when Mr.D'Arcy spoke to me about the previous attempt at the same PCB, and I was told to "get it right", and that I believe I did. I provided (in my opinion) all the necessary information for high and low level coding with this IC, and most importantly I am happy with the result. I am extremely excited to receive my protopack and see if this PCB could become one of hopefully many legacys I leave in the DES and ACES program.

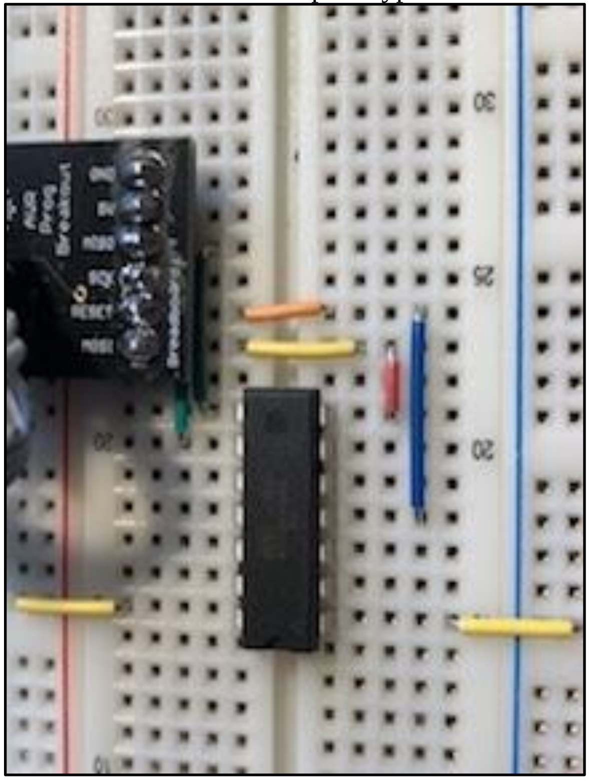
Reflection Phase 2

Upon receiving my board I was extremely excited to solder it. I had already printed my case, as well as prepared a little bag housing enough of each component to solder two boards. I firstly soldered all of the voltage regulation circuit and upon checking with a DMM, a relieving 5V output appeared on the screen. I soldered the rest of components into place, having a series of mini heart attacks hoping components were spaced enough. Once my board was soldered I realized something and my heart sank, I thought that the 6x2 male ISP header was too close to the IC, and that the programmer would not fit. I was extremely relieved when the header fit almost too perfectly, with roughly a 0.5mm gap between the IC and the header. After testing my board for the previous few days I am extremely content with the board I created. It fits the purpose I created it to fulfill, it looks (in my opinion) really well designed and overall pretty cool. Although there are some things I would change if I were to order a v2 of this board, I am nonetheless proud of my idea and execution. Although this board will most likely challenge my peers and I next year in the grade 12 course, I will be happy to have my name mentioned in their DER, and hopefully continue my legacy.

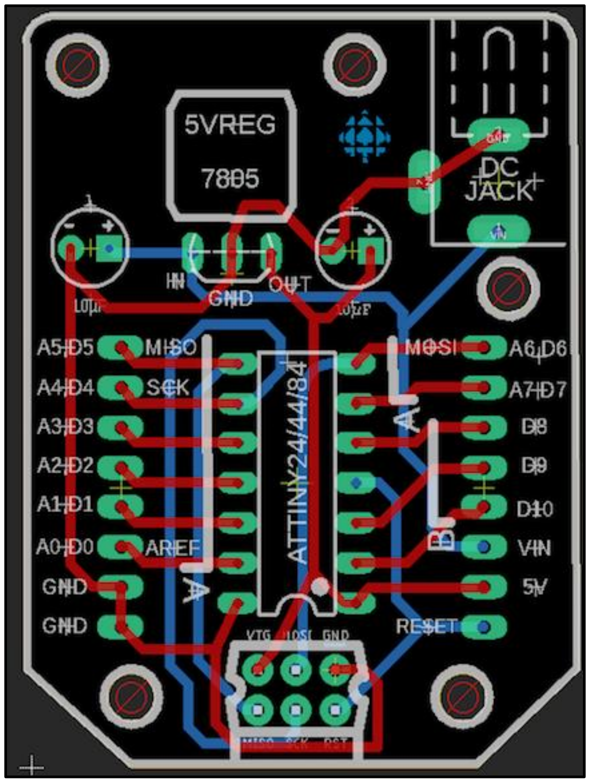
Media

https://www.youtube.com/watch?v=kUmnxdx_YRE

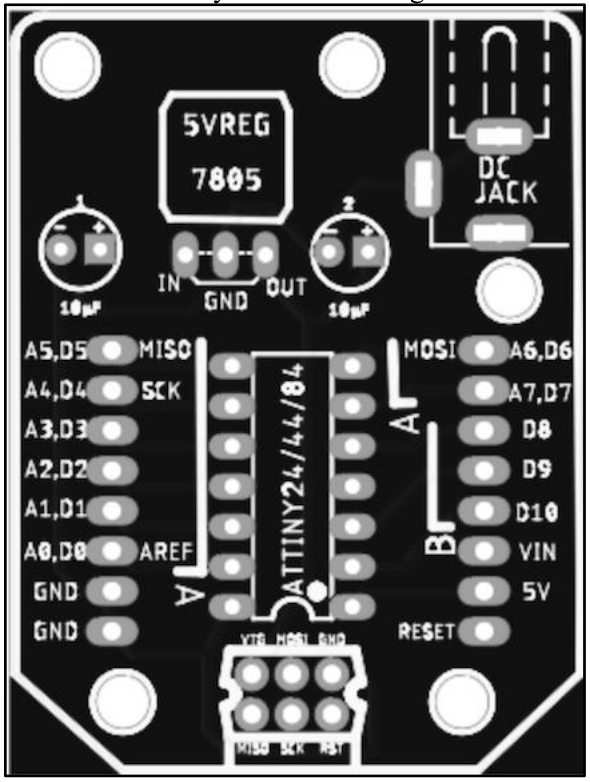
Breadboard prototype



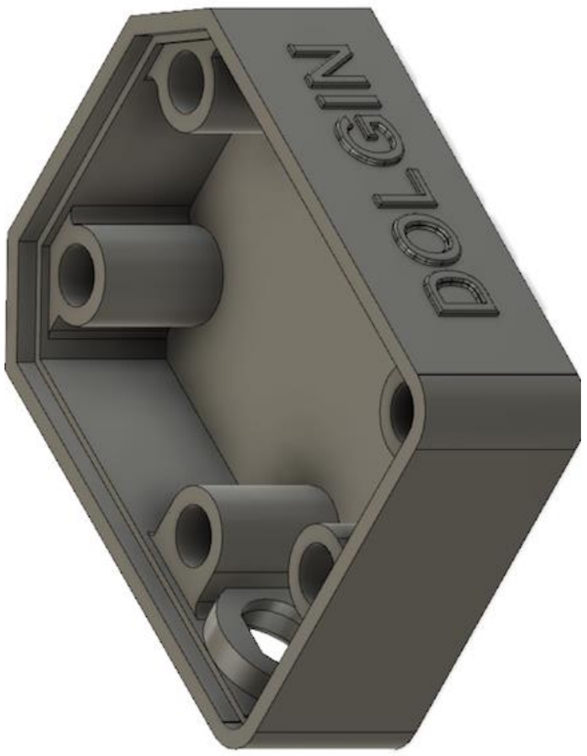
EAGLE Board



DirtyPCB's rendering



Fusion 360 Case

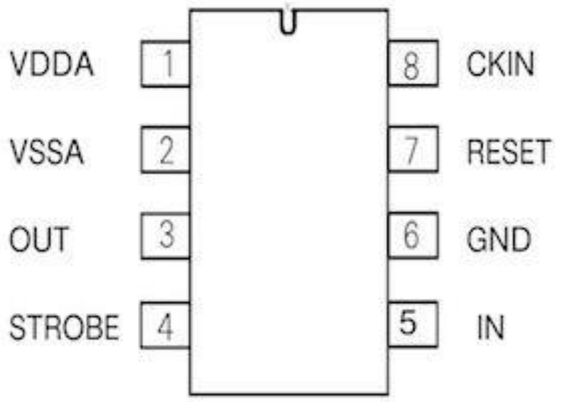


Project 14. ACES Choice: Matrix Equalizer Stick

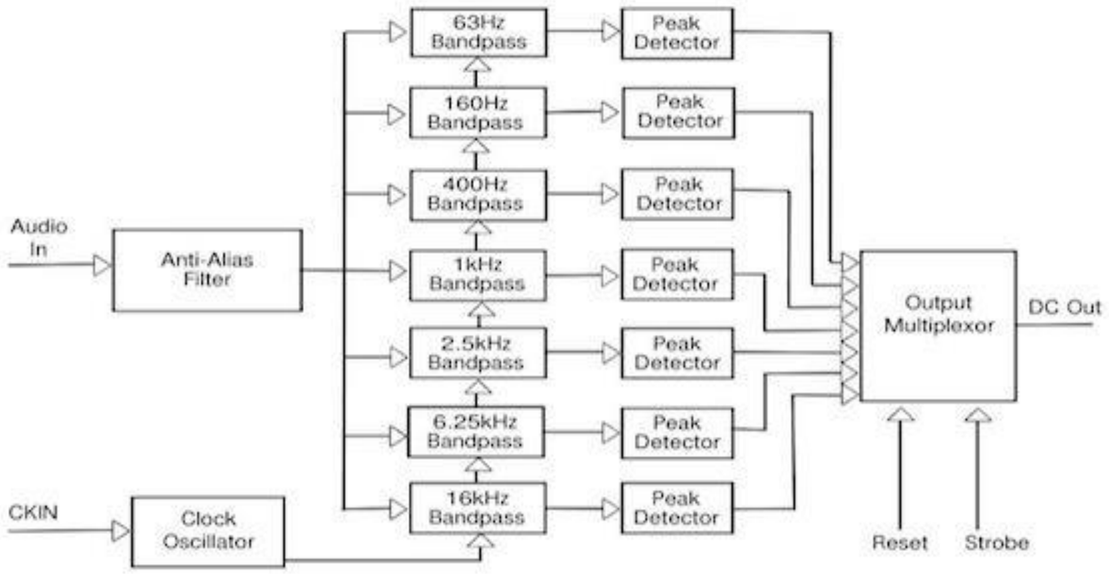
Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/Tasks.html
http://nuewire.com/info-archive/msgeq7-by-j-skoba/

Purpose



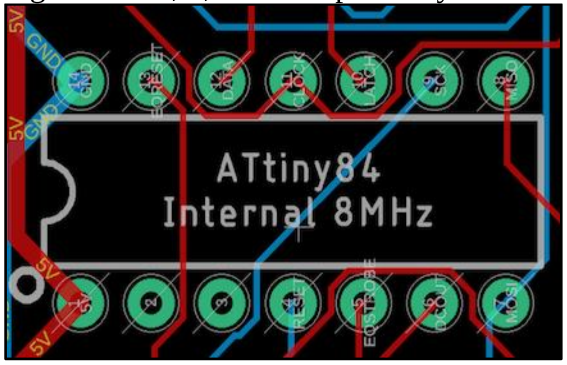
The purpose of this project is to use the specialized MSGEQ7 IC, a 3.5mm audio jack input, an ATtiny84, 2 Shift Registers and a 8x8 LED matrix to create a device that analyzes, reads, and displays audio spectrum values. The MSQEQ7 IC takes a single audio input, and through using the strobe and reset pins, outputs a string of sequential values corresponding to certain ranges of frequencies, from 63Hz to 16000Hz. At start up the chip needs to be reset, accomplished by bringing the reset line high and cycling the strobe line. After the reset line is brought low, the strobe line can then be clocked to retrieve the 7 frequencies sequentially. After all 7 frequencies have been output, the chip starts back over at the first frequency. Then, through using the ATtiny84 to read the output of the IC, and simple bar animation can be created on a 8x8 LED matrix, with each of the 7 bars corresponding to a range of frequencies. The other purpose of this project is to offer ACES one final project at the end of the year to develop their collaboration and coding skills to the best of their current ability. By allowing students to collaborate on code, this project also serves to help ACES work on communication and team skills with their peers.



Procedure

Parts Table		
Component	EAGLE Code	#
6 pin Header	MA06-1	1
3.5mm jack	AUDIO_JACK	1
0.1µF Capacitor	C025	2
.01µF Capacitor	C025	1
33pF Capacitor	C050	1
200kΩ Resistor	0207/10	1
ATtiny84	DIL14	1
74HC595	DIL16	1
TPIC595	DIL16	1
8x8 Matrix	M07C881UR	1
8 pin header	MA08-1	2

Gather the PCB and the components required to solder. Solder the chip seats, ensuring the notch of the seat alignes with the silk screening notch. Solder each of the capacitors in place, they are non polarized, so simply ensure that the capacitance is correct. Solder the audio jack and 6 pin right angle header. Solder two 8 pin female headers into place rather than the Matrix, so that the Matrix sits above the TPIC595 IC. Once soldered, configure the Arduino IDE settings to upload to the ATtiny84 (See Project 13). Download the JSkoba MSQEQ7 code, and create a new project entitled "MatrixEqualizer". Delete all unneccesary components of the Skoba code, including all `Serial.Print` lines. Create 3 unsigned 8 bit integers declaring the data, clock, and latch pins of the Shift Registers as 1, 2, and 3 respectively. Create an unsigned 8 bit integer named "colData" with a value of 0x01, this value will control which column is being lit. Create an array with 7 cell's named "rowData", beginning at 0x01, and ending at 0xFF. Within the `for loop` that reads and places the `analogRead` values into the "spectrumValue" `array` prepare to write to the Shift Registers by writing the latch pin low. Then, shift out the "colData<<i" this cycles the columns from the 1st to the 7th column,



depending on which frequency is being read. Shift out the "rowData[spectrumValue[i]>>7]" this reduces the 10 bit integer (0-1023), to a 3 bit integer (0-7), this effectively lights up the corresponding amount of rows depending on how much of each frequency is detected. Finally, write the latch pin to high, so that the next values can be loaded into the Shift Registers. This code uses Persistence of Vision to create a seemless animation of each individual bar, when in reality, the code is lighting each bar at a time and scrolling through at a rapid rate. Connect the sparkFun pocket programmer to the PCB using the ACES ISP breakout board. Upload the code and ensuring everything is working correctly.

Reflection

This is the last project assigned to us in our ACES grade 11 course, and for some ACES, the last project ever. Although the main focus in the past couple weeks has been my long ISP, this was a final challenge I was willing to accept. Earlier in the year, fellow ACE James Lank asked for help on his code, he was attempting to create an animation on the H.Reed MatrixMadeEZ of the 7 audio spectrum outputs from the MSGEQ7 IC. My *not working* solution for him then consisted of too many `map(), constrain() and if statements` to count, and despite our efforts, no working solution was reached. This project for me is like a round two to show what I have learned since that project earlier this year. Once given the board and components I immediately went to solder, careful of the parts orientations. Once soldered, I had to base my code off of the

EAGLE board files provided to us. With my moderate knowledge of the pinout as well as coding and uploading to the ATtiny84, I was already ahead of my classmates. Imagine my surprise when I realized the dozens of lines of code I wrote previously could be replaced by two lines. All that needs to be done is move the columns along with the spectrum values, accomplished simply by the statement "`colData<<i`". Instead of having to map and constrain the audio values at each cell in the array from 0-1023(2^{10}) to 0-7(2^3), the values simply have to be bitshifted right 7 times. Despite my frustration with my past self, I am extremely happy that I figured it out, and there comes a great feeling with knowing your code is as consise and efficient as you can make it. I took the extra work period as time to redevelop the board, using an Electret Condenser Mic, so that there is an easier correlation between the display, and hearing the actual sound. As well, since the matrices used on this board are pretty well obsolete, I chose to use the more current pinout used on the MatrixMadeEZ. In conclusion, there comes a mix of happiness and sadness in finishing the last non independent project of the year, which truly makes one reflect on how they've grown. Upon reflecting onto my performance as a whole this year on these projects, I see a reduction in size of my code and an increase in complexity, I see a case made for every project I could, and most importantly I can look back upon my work and see how much I have benefit from this course.

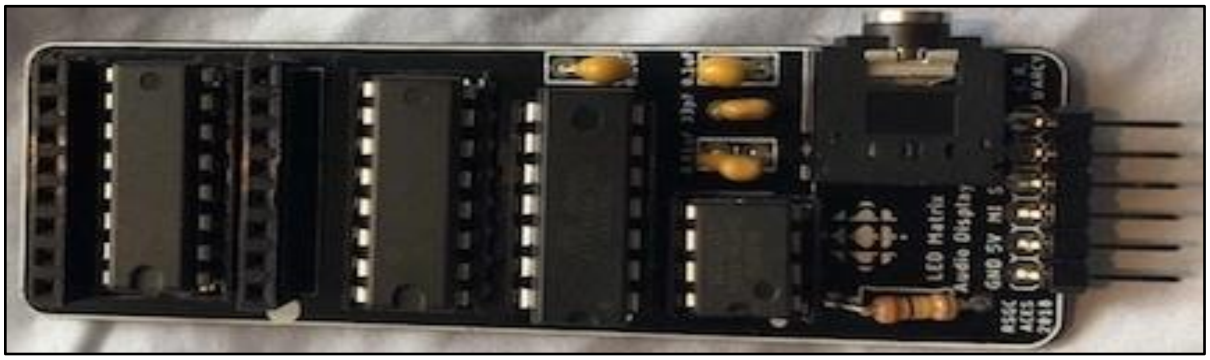
Code

```
uint8_t analogPin = 7; // read multiplexer using analog input 0
uint8_t strobePin = 8; // strobe
uint8_t resetPin = 0; // reset
int spectrumValue[7]; // to hold values
uint8_t data = 1;         // DA: SER IN (Serial Input)
uint8_t clk = 2;          // CK: SRCK (Shift Register cloCK)
uint8_t latch = 3;        // LA: RCK (Register Clock)
uint8_t colData = 0x01;   //which column is lit
uint8_t rowData [] = {0x01, 0x03, 0x07, 0x0F, 0x1F, 0x3F, 0x7F, 0xFF}; // How many
LEDs are lit
void setup(){
  pinMode(analogPin, INPUT);
  pinMode(strobePin, OUTPUT);
  pinMode(resetPin, OUTPUT);
  analogReference(DEFAULT); //Analog reference to 5V
  pinMode(data, OUTPUT);
  pinMode(clk, OUTPUT);
  pinMode(latch, OUTPUT);
  digitalWrite(resetPin, LOW);
  digitalWrite(strobePin, HIGH);
}void loop(){
  digitalWrite(resetPin, HIGH);
  digitalWrite(resetPin, LOW); //Reset the MSGEQ7
  for (uint8_t i = 0; i < 7; i++)
  {
    digitalWrite(strobePin, LOW);
    delayMicroseconds(30); // to allow the output to settle
    spectrumValue[i] = analogRead(analogPin); //Reading the analog value
    digitalWrite(latch, LOW);
    shiftOut(data, clk, LSBFIRST, colData<<i); //Move over i columns
    shiftOut(data, clk, LSBFIRST, rowData[spectrumValue[i] >> 7]);
//Determines amount of LEDs lit based off of the analog read of the frequency value
    digitalWrite(latch, HIGH);
    digitalWrite(strobePin, HIGH);
  }
}
```

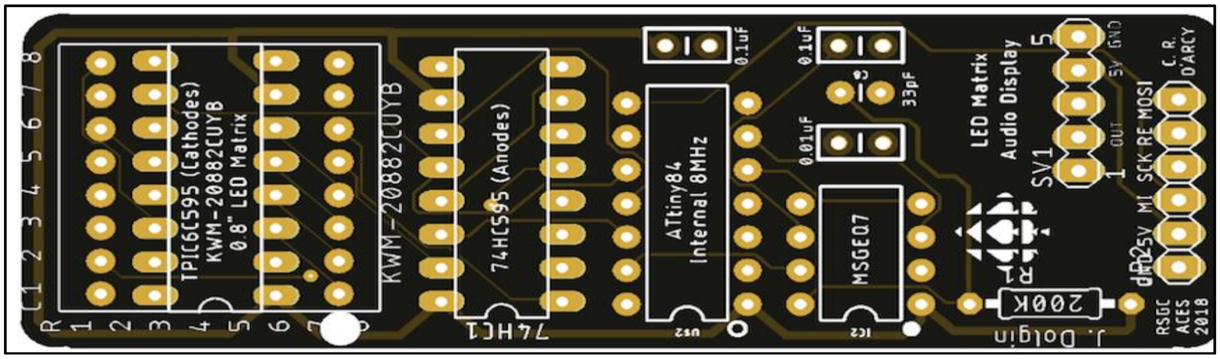
Media

https://youtu.be/5aCXA6JaEWs

Soldered PCB



Edited PCB to include new Matrix and Electret Condensor mic



Board Testing

Final product encased

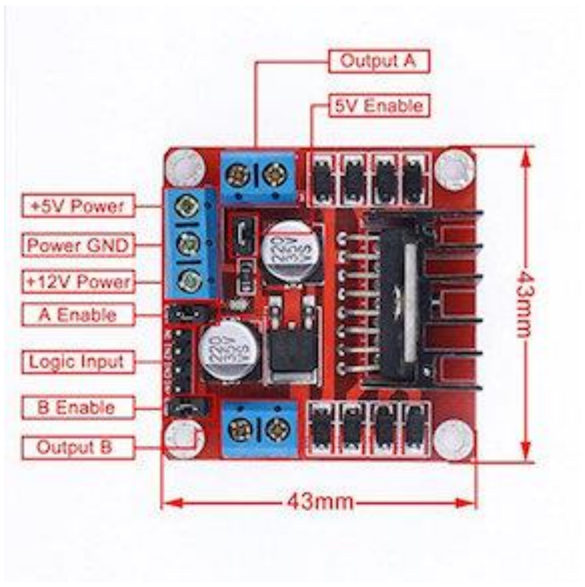
Project 15. The Bi-wheeled Rover

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/1819/ISPs.html

Purpose

L298 Motor driver



NRF24l01 module

The purpose of this long ISP project is to create a two wheeled rover with radio frequency communication, that can be controlled via a joystick input. This project will use the L298 dual motor driver with 2A max channel current to drive the two 6V 100rpm DC motors with 1.5A max stall current. The L298 (pictured to the right) takes either a direct 5V input, or a 5-12V input which then outputs a regulated 5V to the 5V power pin, which can apply power to the Arduino nano board via the VIN pin. The direction of the motors can be controlled using the 4 logic input pins, two for each motor. The voltage applied to the motors can be controlled by the Output A and B pins, which when tied to a PWM signal, vary the voltage powering the motors. The motors used have a maximum torque of 3.6kg/cm, providing more than enough torque for the 1.2 kg rover. For the radio frequency communication protocol, two NRF24l01 tranceiver (transmitter and receiver) modules are used to send the RGB color values and joystick data from the remote to the rover electronics. These NRF modules connect to the Arduino via a Serial Peripheral Interphase (SPI), consuming GND, 3.3V, MOSI, MISO, and SCK pins, while the Chip Enable (CE), CSN (SS), and IRQ pins can be connected to any other analog/digital pins available. The NRF24l01 module has an operation frequency of 2.4-2.525GHz, allowing 125 possible channels for communication. The final new component of this project is the sparkfun analog joystick, which uses two potentiometers for the X and Y directions, and outputs two corresponding values between 0 and 5V. Finally, this project serves the purpose of giving ACES the creative freedom to pursue a creative and original project that they are interested to learn more about, and hopefully by the end emerge with a deeper appreciation and knowledge of the components and processes they chose to pursue.The long ISP stands as a test of student's commitment and creativity, truly showing if they are able to use the devices and critical thinking skills taught in this course to their own advantage.

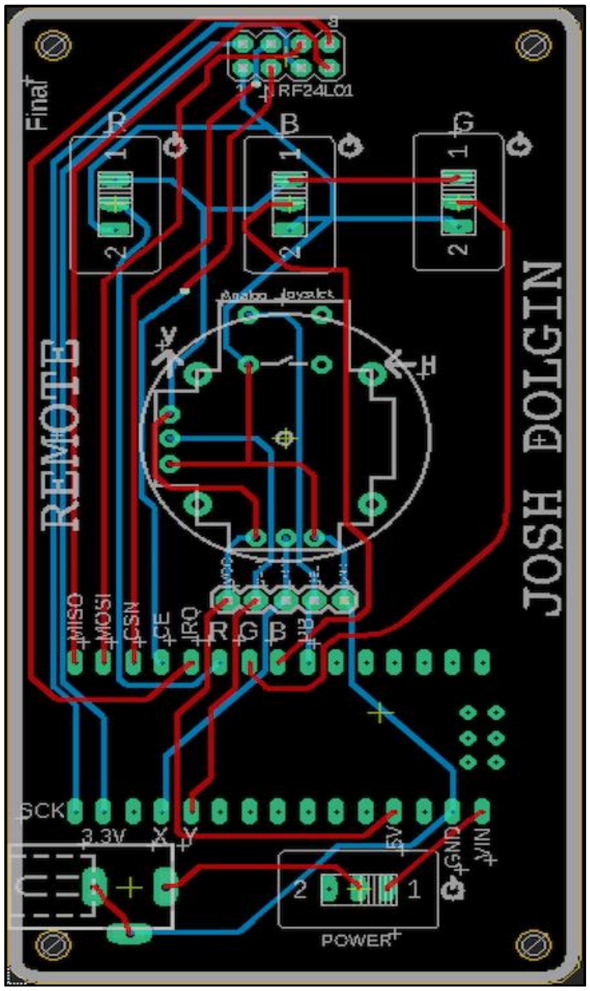
Procedure

Hardware

Collect each of the components required for the remote PCB and lay them out. Solder each component into place, careful of their orientation. Solder the Arduino Nano onto the board, or connect it using 1x10 female headers. Ensure there is no short circuits on the board by connection power and making sure that the Arduino Nano is on and running the blink sketch. Upload the transmitter code to the Nano and ensure that the values are being read correctly and transmitting by checking the Serial monitor. Cut the end of 8 male to female wires, and solder the cut ends into the 3.3V pin, the GND pin, and pins 13-8 of the second Arduino Nano. Connect each of these wires to their respective pin on the NRF24l01 module, in which pin 13-SCK, 12-MISO, 11-MOSI, 10-CE, 9-CSN, and 8-IRQ. Upload the receiver code to the second Nano, and confirm that the Arduino is receiving the 2 joystick values and the 3 RGB values from the remote. Solder 4 male-female wires into pins 3, 4, 5, and 6 of the Arduino. Connect the 4 female ends of these wires to the logic input of the L298 motor driver. Solder 2 JST-F connectors in series for the Lithium-Polymer batterys, to double the voltage but keep current constant. Have the two remaining positive and GND leads connect to a JST-M connector. Solder red and black wires to their repective positive and negative ends of both DC motors. Connect these wires to the Motor-A and Motor-B terminals of the motor driver. Connect a JST-F connector to the 12V input and GND terminals of the L298 driver. Connect the 5V output and GND of the driver to the VIN and GND pins of the Arduino. Solder the positive lead of the RGB led strip to the VIN pin as well, and solder the red, green, and blue leads to pins A0-A2 of the Arduino. Once again, confirm that the code receives the values, and that the motors spin in the correct corresponding direction to the motion of the joystick.

Parts Table	
Component	#
Arduino Nano	2
Sparkfun Joystick	1
6V 100rpm DC motor	2
NRF24l01 module	2
L298 motor driver	1
RGB led strip	1
3.4V Li-Po battery	2
SPDT slide switch	3
DC jack	1

Remote PCB

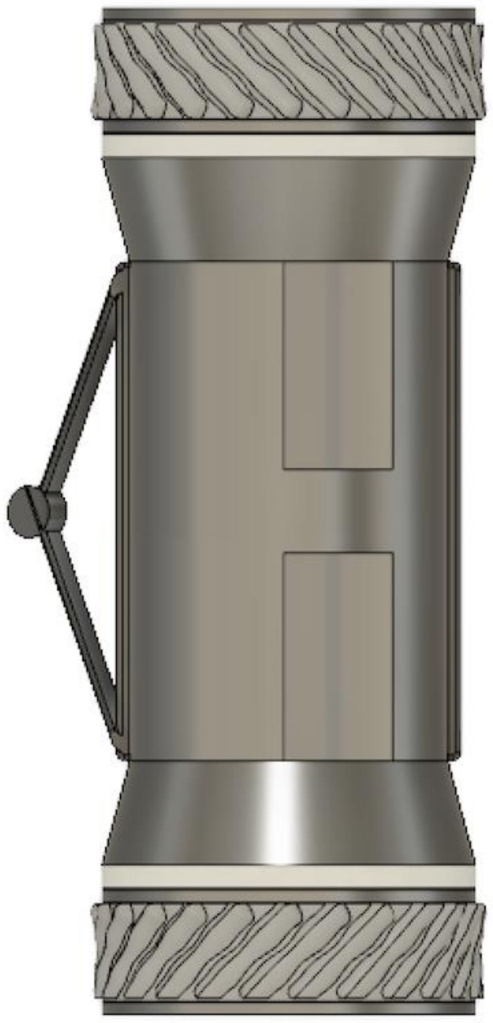


Software

Include the radio library and define the CE and CSN pins attached to the radio as pins 9 and 10 on the Arduino. Create 5 unsigned 8 bit integers to declare the names for the 2 joystick pins and the 3 RGB pins. Create a data structure titled "package" housing 2 unsigned 8 bit integers for the joystick readings, and 3 boolean variables for the switch readings. Set up all of the necessary functions for RF communication, including the operating channel and addresses, as well as if the module will be receiving or transmitting data. Load values into the data structure one at a time, starting with the joystick values and then the switch values. Since the RF modules can only send 32 bits at a time, it is necessary to house the joystick values (0-1023, or 2^{10}) as unsigned 8 bit integers (0-255, or 2^{8}) rather than 16. This is accomplished by bitshifting the joystick values 2 to the left. Print each of the values in the Serial monitor, then send the data structure using the `radio.write` command. In the receiver code, use the same setup as the transmitter, with the exception of adding 4 extra pin declarations for the motor logic, as well as setting the module to receive. Create 5 functions detailing the 5 motions of the rover; `stationary()`, `goForward()` `goBackward()`, `turnLeft()`, and `turnRight()`. Once there is a radio signal to be received, use the `radio.read` command to read the data structure, and then print out the read values to confirm they are correct. Create and `if() else()` ladder that uses the joystick values to correspond to a direction of the rover. Finally, write the inversion (~) of the RGB, so that the lead is grounded when the switch is turned high.

CAD

Top View of Design



Create a sketch on the Y plane of a rectangle 45mm tall and 30mm across, and using the spline, create an arc across one side of the rectangle. Revolve the rectangle around the bottom axis in a complete circle. Along the flat side of the body, create a 8mm deep hole to fit the motor shaft. Along the top of the wheel, revolve a second strip, and create a freeform cylindrical extrusion along the top, this will be the tire of the rover. Create a midplane 65mm from the flat side of the wheel, and mirror the wheel and tire over that plane. Create a second 90mm disc, with a motor housing in the center complete with two screws to hold the motor in place. Create 6 holes 5mm in diameter, 3 along the top and 3 along the bottom of the flat face of the disc. Create a hollow, chamfered extrusion 30mm long at a 15° angle. Mirror this extrusion as well, and create a cylindrical extrusion connecting the two pieces. Using the "split body" tool, split the cylindrical body symmetrically into a top half and a bottom half. Create 3.04mm screw holes coinciding with the holes in the disk on the top and bottom half of the rover, allowing the top and bottom half to be secured together by the disk. Create a ribbed design along the bottom half of the rover (purely for looks) and pattern the design across the entirety of the body. Create a tail sticking off the back of the rover, in order to stabilize movement. Create two openings on the top of the rover, to house the RGB LED strips. Finally, within the bottom of the motor body, create a well to hold the 2 Li-Po batteries, as well as a cylindrical cutout in both the top and bottom half of the rover to house the motors securely in place.

Code

Transmitter

```
//Project: Transmitter Code for Bi-Whelled rover
//Author: Josh Dolgin
//Date: 2019/05/26
//Status: Working
#include <RF24.h>
RF24 radio(9, 10);              //CE (Chip Enable/Disable), CSN(SS)
byte addresses[][6] = {"1Node", "2Node"};
#define CHANNEL 14 //2.414GHz
uint8_t joyX = A0;
uint8_t joyY = A1;
uint8_t rLed = 7;
uint8_t bLed = 5;
uint8_t gLed = 6;


struct package //Create data structure
{
  uint8_t xVal = 0;
  uint8_t yVal = 0;
  boolean R = 0;
  boolean G = 0;
  boolean B = 0;
};
typedef struct package Package;
Package data;

void setup() {
  Serial.begin(9600);
  radio.begin();                          //invoke the radio object
  radio.setPALevel(RF24_PA_MIN);          //close range so minimum power sufficient
  radio.setChannel(CHANNEL);              //Tx and Rx communication on same channel
  //https://tmrh20.github.io/RF24/classRF24.html#af2e409e62d49a23e372a70b904ae30e1
  radio.openWritingPipe(addresses[0]);    //Transmit assumes these pipes
  radio.openReadingPipe(1, addresses[1]);
  radio.stopListening(); //Transmit rather than receive
}
void loop() {
  data.xVal = analogRead(joyX) >> 2; //creates 8 bit integer rather than 10
  data.yVal = analogRead(joyY) >> 2; //creates 8 bit integer rather than 10
  data.R = digitalRead(rLed); //Receives R value
  data.B = digitalRead(bLed); //Receives b value
  data.G = digitalRead(gLed); //Receives g value
  radio.write(&data, sizeof(data)); //Sends the data structure
  Serial.print(data.xVal);
  Serial.print("  ");
  Serial.print(data.yVal);
  Serial.print("  ");
  Serial.print(data.R);
  Serial.print("  ");
  Serial.print(data.B);
  Serial.print("  ");
  Serial.println(data.G); //Prints all of the data in the serial monitor
  delay(100);
}
```

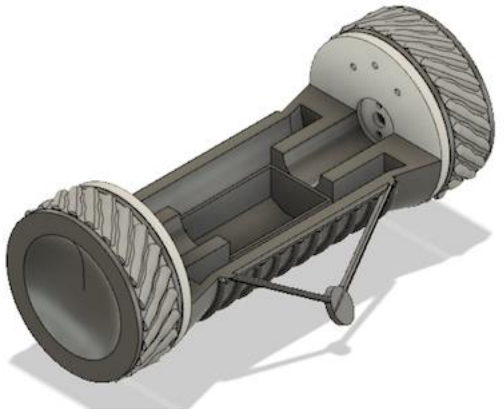
Receiver

```
//Project: Receiver code for Bi-Wheeled rover
//Author: Josh Dolgin
//Date: 2019/06/26
//Status: Working
#include <RF24.h>
RF24 radio(9, 10);              //CE (Chip Enable/Disable), CSN(SS)
byte addresses[][6] = {"1Node", "2Node"};
#define CHANNEL 14 // 2.414 GHz
uint8_t in1 = 6;
uint8_t in2 = 5;
uint8_t in3 = 4;
uint8_t in4 = 3; // Motor control pins
uint8_t rLed = A0;
uint8_t bLed = A1;
uint8_t gLed = A2; // LED pins

struct package // Data structure to receive
{
  uint8_t xVal = 0;
  uint8_t yVal = 0;
  boolean R = 0;
  boolean G = 0;
  boolean B = 0;
};
typedef struct package Package;
Package data;

void setup() {
  Serial.begin(9600);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  radio.begin();                          //invoke the radio object
  radio.setPALevel(RF24_PA_MIN);          //close range so minimum power sufficient
  radio.setChannel(CHANNEL);              //Tx and Rx communication on same
channel  //https://tmrh20.github.io/RF24/classRF24.html#af2e409e62d49a23e372a70b904a
e30e1
  radio.openWritingPipe(addresses[1]);    //select ONE addresses to write to
  radio.openReadingPipe(1, addresses[0]); //receiver and transmit are reversed
  radio.startListening(); // Receiver
}
void stationary() { //Stay still
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
}

void goForward() {
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}

void goBackward() {
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}

void turnLeft() {
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}

void turnRight() {
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}
void loop() {
  if (radio.available())
  {
    while (radio.available())
    {
      radio.read(&data, sizeof(data) ); // Read the data
    }
    Serial.print(data.xVal);
    Serial.print("  ");
    Serial.print(data.yVal);
    Serial.print("  ");
    Serial.print(data.R);
    Serial.print("  ");
    Serial.print(data.B);
    Serial.print("  ");
    Serial.println(data.G); // Print the data on the Serial monitor
  }
  if (data.xVal < 150 && data.xVal > 100 && data.yVal < 150 && data.yVal > 100) {
// if the joystick is in the middle
    Serial.println("Nothing");
    stationary();
  }
  if (data.xVal < 150 && data.xVal > 100) {
    if (data.yVal > 151) {
      Serial.println("FORWARD");
      goForward();
    }
    if (data.yVal < 99) {
      Serial.println("BACKWARD");
      goBackward();
    }
  }
  if (data.xVal > 151) {
    Serial.println("LEFT");
    turnRight();
  } else if (data.xVal < 99) {
    Serial.println("RIGHT");
    turnLeft();
  }
  digitalWrite(rLed, ~data.R); //inversion of the Boolean value
  digitalWrite(bLed, ~data.B);
  digitalWrite(gLed, ~data.G);
  delay(100);
}
```

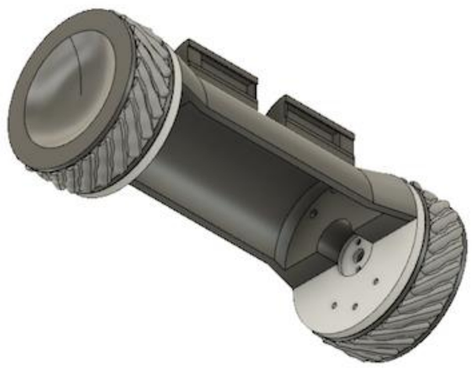
Media

https://www.youtube.com/watch?v=M1K6hx5Crig

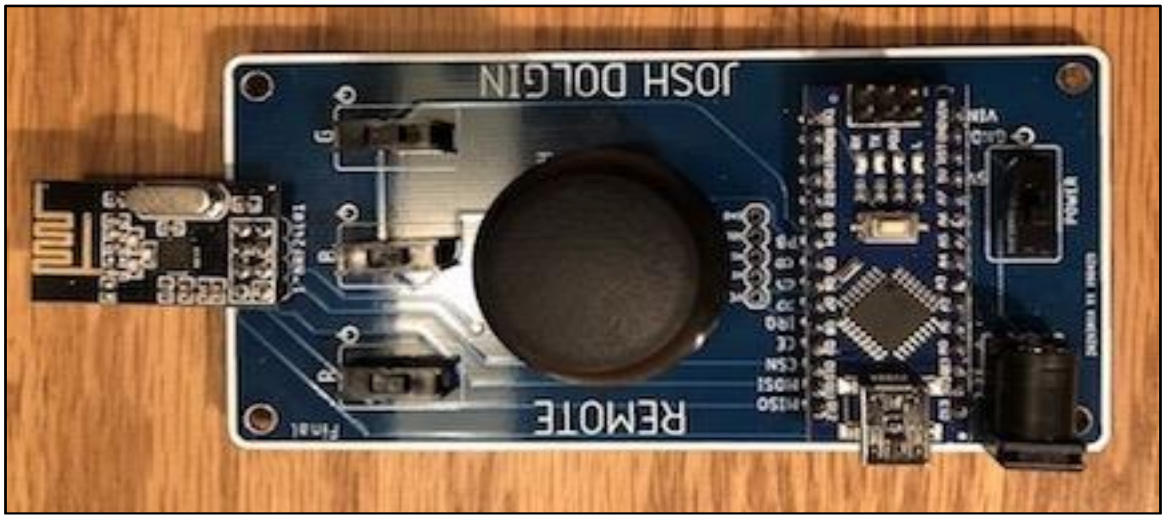
Bottom Half of the Design



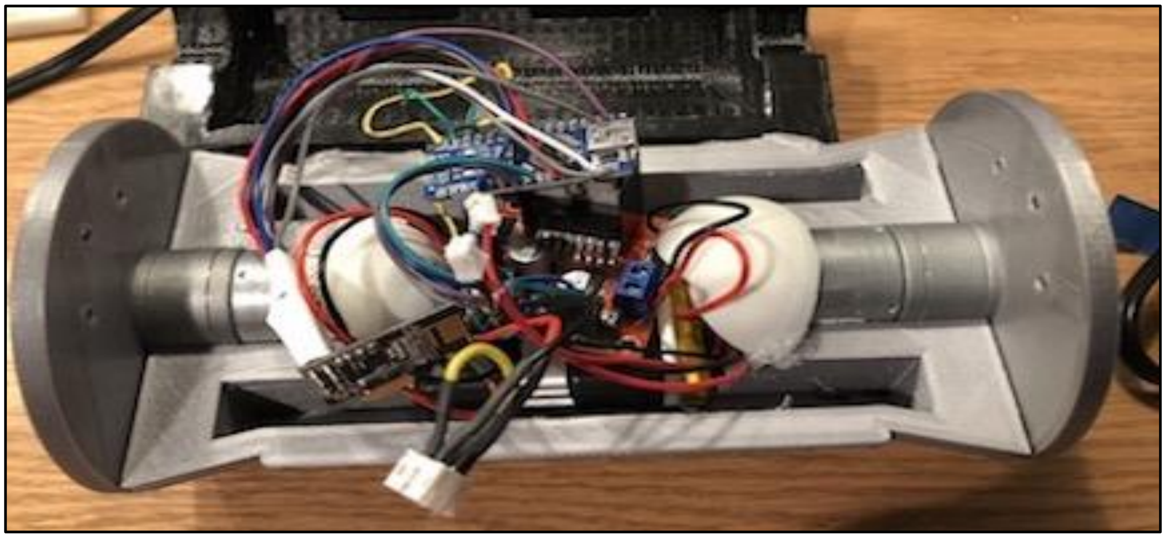
Top half of the Design



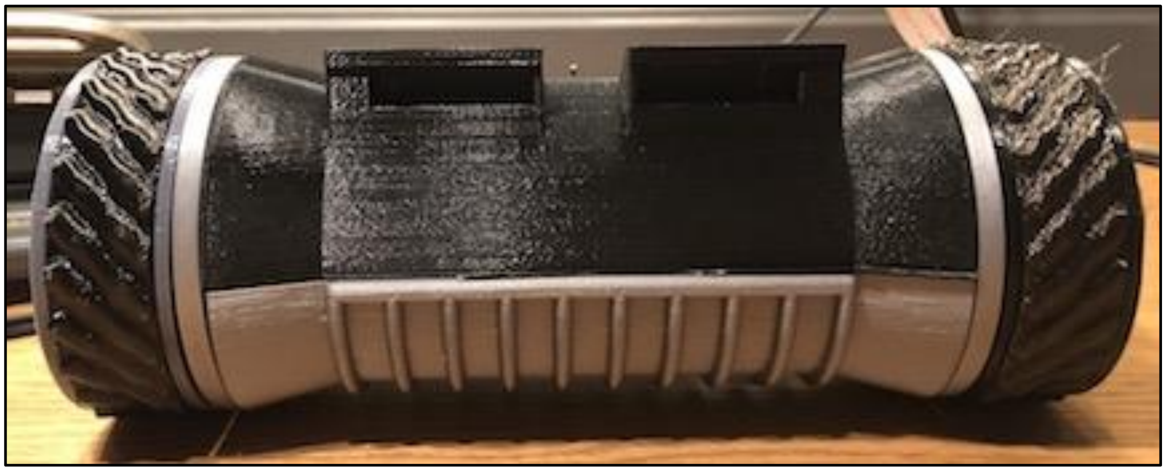
Soldered Remote



Hardware inside rover



Final Product



Reflection

This project was certainly the most rewarding one that I have done all year. When I began to write ideas for my long ISP, I was comsumed with the what I interpreted to be the failure of my groups rover during the term 2 ACES Rover Project. With the addition of the newly learned communication protocols, an interesting combination of the two came to mind. I would set out to create a two wheeled, RF controlled rover (two wheels looks cooler) with some controllable RGB strip lights (because why not). My project began with choosing components that I could design code and CAD around, so I set out to learn from mistakes on the ACE of Spades rover. I began by choosing a motor with a much higher torque, in order to support the weight of the Bi-Wheeled rover. This higher torque, however, comes at a price. The max stall current for the new motors is 1.5A, much greater than the 200mA consumed by the geared down DC motors. This higher current means that the motor driver used previously which has a maximum current flow of 0.8A won't work. I learned this the hard way, when two of those motor drivers began to smoke and heat up when the motors were running. With two burnt drivers down, I found the issue and bought a driver capable of handling this current. Other than that issue, I had very little problems wiring the hardware for this project, but the software is an entirely different story. By using Mr.D'Arcy's example code as well as some tutorials on the internet, I was quickly able to send two different pieces of data over RF, the X and Y values of the joystick. Once I tried to add the color values, everything went downhill. Once the addition of the radio write and read statements were added, all values went off the rails. I was getting 0's for all of my data then all of a sudden my X value was 1 and my red LED value was 554. As I later found out after trying countless other methods of sending, arrays, characters, etc. I discovered that I could simply make a data structure. A normal NRF transceiver can send 32 bits at a time, making two 16 bit integers and 3 boolean variables a no-go. Once all the hardware and software was finished, I simply had to design the rest. The first PCB I sent to china did everything I wanted it to, and didn't fry my Nano like I expected it to. The 3D design aspect of this project was tricky, but not impossible. I created an extremely modular design, so that each piece can be printed, rapidly fixed and re-printed if necessary. I found a sense of completion when I looked at my fully printed design and smiled. The flexible filament treds for the tires is a really cool idea, that works much better than anticipated. The split main body held together by the end disks, which are both in turn attached to the motor work well and hold the body together, allowing for a balance between rigidity and the ability to quickly open the design to fix or plug in electronics. Then, on the day before initial presentations, everything that could go wrong, did. The leads on one of my motors broke off, the NRF module burnt out, and the batteries no longer supplied enough current. After a quick trip to trusty Creatron, and 4 hours of anxiety and sweat fueled soldering in my basement, a more compact, efficient, and trustworthy wiring setup emerged. In conclusion, although this project has had many, many, many issues, pushing me to my wits end, I created a product through struggle, that shows my resiliency and capability of working on a project at the end of my grade 11 year. Looking back, I would do it all again, and that’s all that matters. Right?

ICS4U

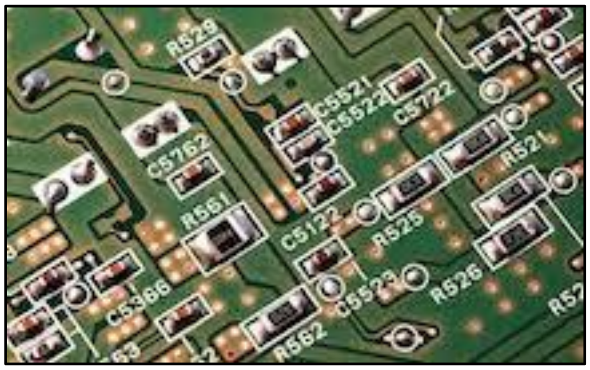
AVR Optimization

Project 16. The GB machine

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/1920/Tasks.html - GBMachine

Purpose



The purpose of this project is to solder an extremely useful PCB for breadboard prototyping, as well as provide an introduction to Surface Mount Devices (SMD). The purpose of the GB machine is to power both sets of power-ground rails on a breadboard, with some extra extremely helpful additions to the breadboarding process. The PCB uses two separate SPDT slide switched to turn the power on and off, and to choose what voltage to apply to the rails, either input voltage or 5 volt regulated voltage. Most of the components used on the PCB are Through Hole Technology, but with the addition of a single SMD LED and resistor, this PCB allows for a quick but important introduction to surface mount soldering. Surface mount soldering is quickly becoming the norm, and with less and less through hole parts being offered, the limitations of THT are becoming very apparent. In order to combat this issue, it is important to teach the skill of surface mount soldering as early as possible.

Procedure

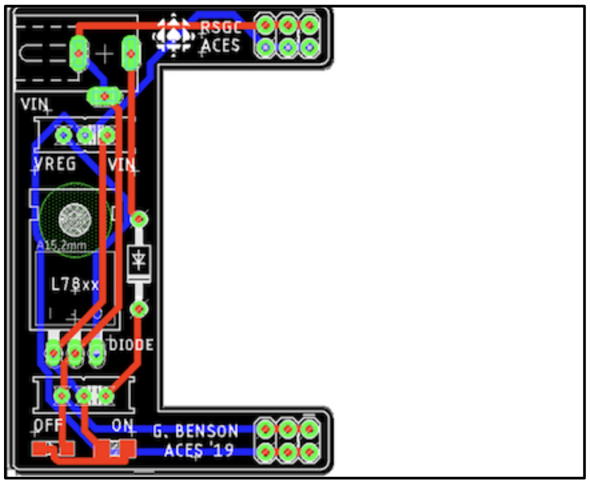
Parts Table	
Component	#
GB Machine PCB	1
SPDT slide switch	2
L7805 5V regulator	1
diode	1
6x2 Male header pins	2
DC jack	1
9V battery	1
1204 SMD LED	1
1KΩ SMD resistor	1

Gather all the components listed in the parts table and lay them out. In order not to overheat the THT components, it is important to solder the SMD components first. Take the solder paste out of the fridge and press a dab of the paste onto the tip of a toothpick. Clean the tip of the solder paste syringe and return the paste to the fridge. Use the toothpick to place a dollop of solder paste onto each SMD pad. Using the tweezers, roughly position the components onto the solder pads, keeping in mind the polarity of the LED. Set the hot air gun to 225° C and wait until it heats up. Hold the hot air gun perpendicular to the board and make gentle circular motions in order not to heat one spot for too long. Continue to slowly move the hot air gun towards the board until the solder melts and pulls the surface mount components into place. Solder the rest of the THT components into place, carefully checking the orientation and connections of each part. Test to make sure the SMD components work by connecting 9V to the DC jack and turning the device on. The LED should light up when the SPDT switch is in the "on" position. Connect the GB machine to a breadboard and test both the input voltage and regulated voltage values with a DMM. Build a simple circuit consisting of an ATtiny84 and 8 LED's to show the 5V regulated voltage capabilities.

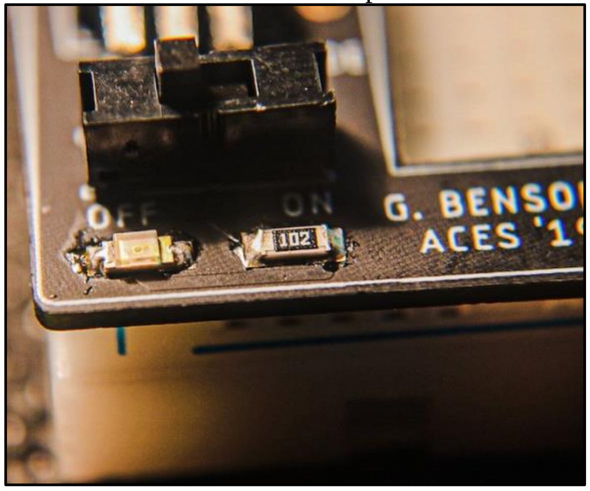
Media

https://www.youtube.com/watch?v=9XFJa6iUPqI

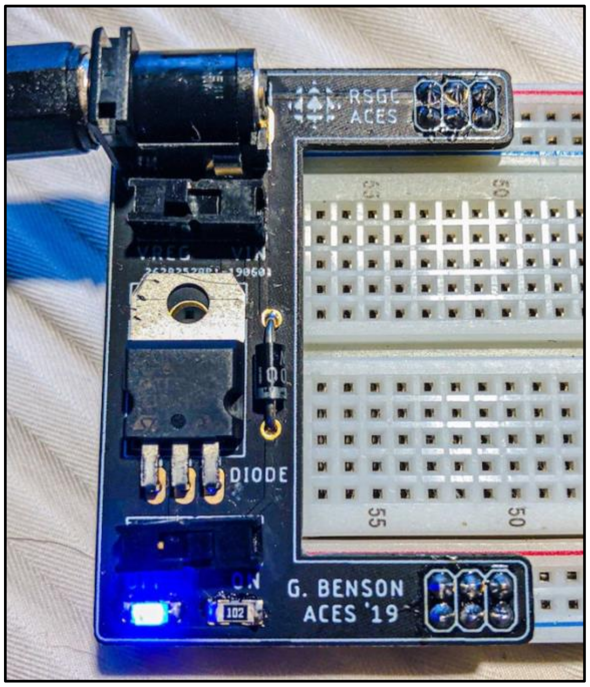
EAGLE PCB



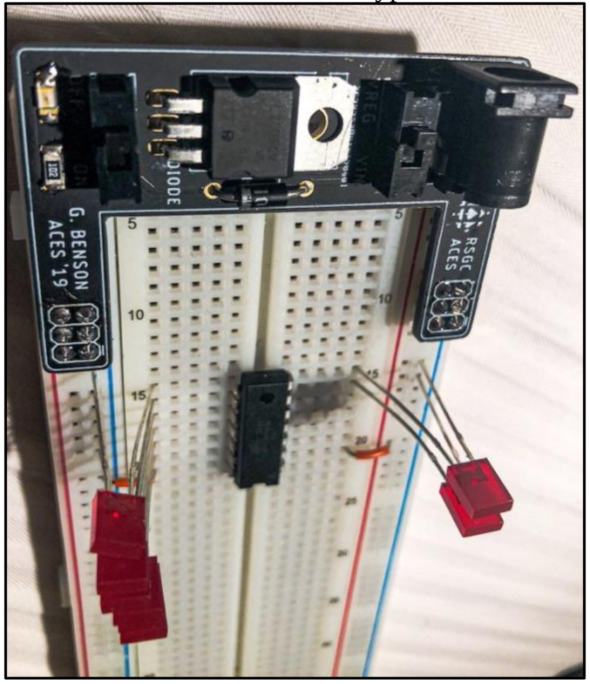
Surface mount components



Soldered GB machine



Breaboard Prototype



Reflection

Overall I believe that this was a very great first project to start the grade 12 year. Although the new concept of surface mount soldering was introduced, I could still rely and build upon on my though hole soldering skills to overcome that challenge. I believe that my first introduction to SMD was extremely successful and I look forward to implementing it within my future projects. In conclusion, this project was the perfect introduction and ease into what hopes to be a thoroughly rewarding and challenging grade 12 year.

Project 17. 3D Printing and Forming

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/1920/ISPs.html
https://manual.prusa3d.com/c/Original_Prusa_i3_MK3S_to_MMU2S

Purpose

Prusa MMU2s

The purpose of this project is to build, troubleshoot and test all of the capabilities of the Prusa MK3s Multi Material Upgrade 2.0, which allows the printer to print in 5 colors or materials, as well as the capabilities of the Mayku formbox vacuum former. The Prusa MK3s MMU2s upgrade uses 3 additional stepper motors attached to the top of the printer, as well as a series of 3D printed parts and electrical components to switch between each of the 5 filaments loaded to the unit. The first motor spins all 5 of the drive gears underneath the filament lines. The second motor rotates a cylinder with angled ball bearings, pressing one filament onto the drive gear at a time, and therefor driving the filament towards the filament selector. The filament selector motor turns a threaded rod which in turn moves the filament sensor and bowden tube leading to the hotend to line up with one of the 5 filament openings. Through the combination of design and software, the MMU2s is able to simplify the process of printing in multiple colors, using a single extruder nozzle. The upgrade can be built and attached within 6 hours, and printing in 7.

Mayku Formbox

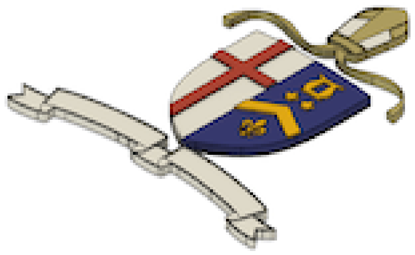
The Mayku formbox is a desktop vacuum former produced with the hobbyist in mind. The former uses two forms of plastic sheets, each require different temperatures and timing to produce the desired vacuum sealed affect. The sheet is sandwiched between two metal extrusions, and lifted to the top of the machine to the heater grate. Once the plastic is heated to the point it droops 1cm below the metal extrusions, the vacuum connected to the back of the machine is turned on. The vacuum sucks air through the metal grate along the bottom of the formbox. This takes all the air from under the plastic sheet, creating a perfect mold of the model placed on the metal grate.

Prusa MK3s MMU2s

Procedure

Parts Table
Programs
Fusion 360
Prusa Slic3r
Parts
Prusa Mk3s
MMU2s upgrade kit
Assorted Prusa filaments

RSGC crest in Fusion 360

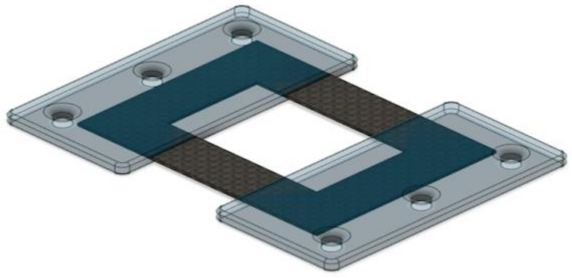


Unbox the MMU2s upgrade kit and lay the bags out in ascending order based on the label number. Open the Prusa online manual and begin with instruction set 1 and continue until the upgrade is completed. Attach the upgrade to the printer via the snaps located at the back of the unit. Flash both the printer and the upgrade with the most recent firmware additions and check to ensure everything is properly assembled. Begin the troubleshooting process and ensure everything is up to the standard listed in the manual. Begin the first test print and confirm everything is working as intended. Open Fusion 360 and create a new design. To test the multi-color capabilities of the printer, import an svg image of the RSGC school crest. Selecting a single shape of the crest at a time, extrude each piece to create a new body at the desired height. Export each of the bodies one at a time to a single folder. Open the Prusa slic3r and import each of the bodies within the the folder. Assign the extruder number tied to the color you would like to each of the bodies in the file. Change any other settings to ensure print quality and reduce printing time. To test the multi material capabilities of the upgrade, create a sketch of two 50mm by 80mm rectangles and extrude them to height of 1.2mm. On top of the new extrusions create two bridges between the rectangles that are 40mm long. Connect the bridges so that a circle is formed in the middle of the two rectangles. Export each body as an stl file into a designated folder and drag the contents of the folder into the Prusa slic3r. Slice the model and watch the first couple layers of the print to ensure everything runs smoothly. Take the hinge and attach each side of it to a 2" by 4" rectangle of scrap wood using 3 ¾" wood screws. Sand and trim any pointed or unwanted strands of filament attached to the print.

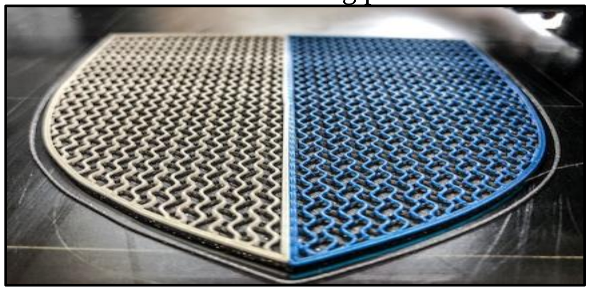
Media

https://www.youtube.com/watch?v=YuwwHZVtlIk

Hinge designed in Fusion 360



RSGC crest being printed



ACES RSGC crests

MMU2s sampler tray

Mayku Formbox

Procedure

Parts Table	
Part	Quantity
Mayku Formbox	1
Formbox cast sheets	1
3D printed RSGC crests	12
Assorted Multi-color prints	7

Turn on the Mayku formbox and attach the vacuum hose to the back of the machine. Turn the "timer" dial to 1 minute 20 seconds. Turn the "tepurature" dial to 5. Wait until the blinking amber light beside the temperature dial turns green, this means that the machine has reached the desired temperature. Lift the yellow lever on both sides of the top metal extrusion and lift the plate all the way to the top. Place the plastic sheet in the open slot left on the lower metal plate. Lower the top metal plate back down and snap the levers into place. Raise the metal and plastic sheets to the top of the guide rods. Click the timer button on the left side of the machine and wait until the light changes from red to green. Once the button turns green, place the models to be formed along the metal grate in the preferred arrangement and turn on the vacuum. Slowly lower the plastic sheet, giving the vacuum enough time to suck out all the air from underneath the plastic. Leave the vacuum on for an extra 10 seconds to cool the plastic. Turn off the vacuum and lift the upper metal sheet to the top of the guide rods. Remove the plastic sheet from the slot, and pop out the model from the vacuum seal. Following the previous steps, create a single vacuum mold of the 12 3D printed RSGC crests. Remove the 3D prints from the plastic sheet and set them aside. Slowly melt 2 cups of semisweet chocolate, and remove the pan from the heat. Using a clean paintbrush, paint a thin layer of chocolate in each individual mold, ensuring the casted result retains the detail of the model. Using a spoon, add and smooth the chocolate in each mold until the mold is full. Place the sheet into the freezer and wait 30 minutes before removing the chocolate from the mold and placing them into a Tupperware container. Place the container into the fridge until ready for consumption.

Media

Vacuum formed RSGC crest

Vacuum shell

Vacuum form of complex geometries

RSGC chocolate crests

Reflection

n late August I sent Mr. D'Arcy an email, reminding him of the Prusa Multi Material Upgrade, something we had discussed before the summer and agreed would make a great addition to the DES. Within the first week of my grade 12 year, the familiar but always exciting walk down to pick up a package for the DES enticed me yet again, what could this package be? Lo and behold it was the upgrade kit I had been thinking about for the past couple of weeks. Even better than ripping the plastic wrapping off of the cardboard box like a toddler on Christmas morning was checking the ISP page and seeing that a new project had appeared, under the title of Prusa Multi-Material. Over the next week I continued to chip away at the build process, off in my own head about how I would love to do that project. When it came to selecting our ISPs in class, I selected my project and sat there anxiously hoping no one would write the same words on their sheet of paper. After 15 minutes of the ISP yankee swap, I remained with my initial choice. This project hasn't taught me anything new; however, it helped me build upon the ideas already ingrained from my previous hardware experience. I spent the usual 3 hours in the DES after school tinkering away with Allen keys and dials, rather than prodding with a DMM. I spent a couple of minutes looking directly at the problem before identifying that it was, in fact ,the problem. Throughout the 5 weeks of working on this project I have gained extremely valuable knowledge and skills on two of the most important and complex prototyping devices in the DES; this information is not only useful to me, but to my classmates, as the final extension of my project is to share my expertise with whoever would like to listen.

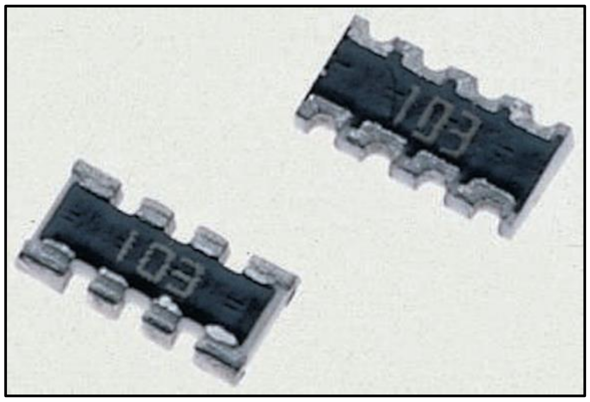
Project 18. CharlieStick

Part 1

Reference

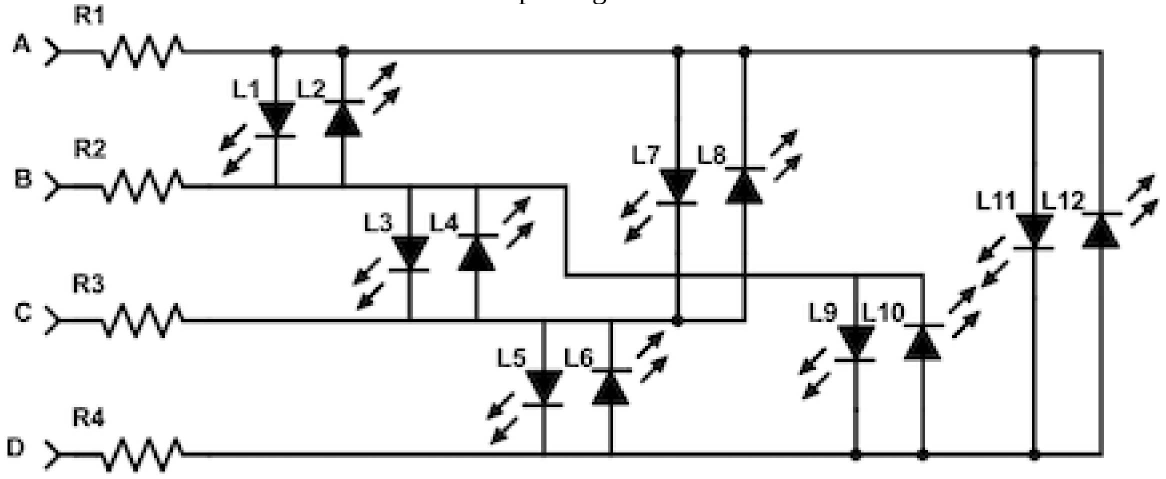
http://darcy.rsgc.on.ca/ACES/TEI4M/1920/Tasks.html - CharlieStick

Purpose



The purpose of this project is to refresh and further refine surface mount soldering skills, as well as educate ourselves in the software technique of *charlieplexing*. The year began with a brief introduction to the soldering of the 1206 LED and resistor packages using the AOYUE hot air gun. This project aims to build upon those fundamental skills by soldering 12 1206 LED's, and an extremely small 3216m resistor network. Charlieplexing is the idea of efficiently using as few digital IO pins on the Arduino to control as many LED's as possible. Since LED's are polarized, and only allow current to flow through in a single direction, LED's can be arranged and connected in separate permutations rather than combinations. Through using this technique, 12 LED's can be driven by 4 pins on the Arduino. In the diagram below, there are 12 permutations of pins; AB, BA, AC, CA, AD, DA, BC, CB, BD, DB, CD, and DC, each of these permutations corresponds to one of the 12 LED's. Since there are resistors on pins A, B, C and D, it is important to keep in mind that the resistance on each LED is the sum of the resistors attached to the cathode and anode of the LED. Through changing the configuration and states on the 4 pins, 1 LED can be controlled at a time to create complex animations.

Charlieplexing Schematic



Procedure

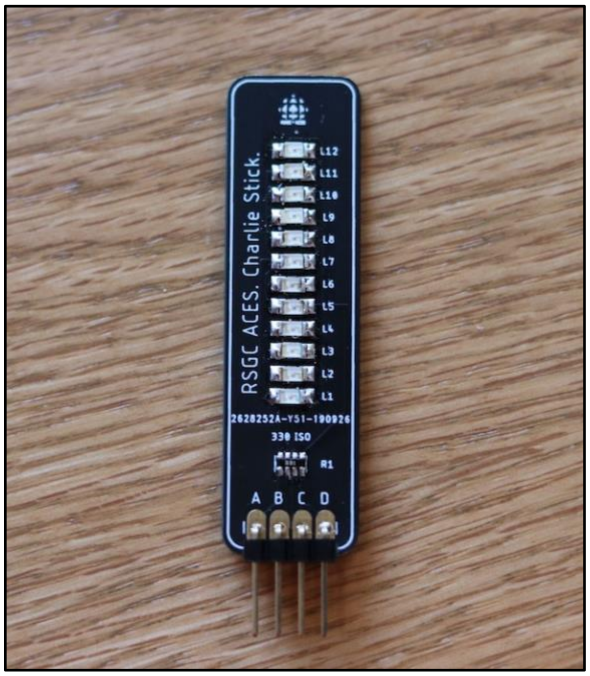
Parts Table	
Component	Quantity
CharlieStick PCB	1
1206 LED	12
1206 100Ω Resistor Network	1
4-pin right angle header	1
AOYUE hot air station	1

Gather all the surface mount components and lay them out on the silicon sheet. Remove the solder paste from the fridge and eject some paste onto a tissue. Using a toothpick, place a dab of the solder paste on all the pads for surface mount components. Arrange all the components onto the PCB using the small tweezers, mindful of the orientation of the LED's. Turn on the AOYUE solder station and preheat the SMD rework hot air gun to 250 °C. Begin high above the board, and gradually lower the hot air gun, making small circular movements. As the solder paste melts, each component will be pulled into the correct position. Using a solder pen, solder the 4-pin right angle header onto the PCB. Using 2.4V, touch the header pins in every permutation, ensuring all LED's are working correctly. Using the dimensions of the board, design a simple case in fusion 360 that slides over top of the PCB, showing the LED's but covering and protecting the rest of the PCB. Open the Arduino IDE and create a new sketch labelled "CharlieStick". Create an array housing all the pin numbers used for the CharlieStick PCB. Define the amount of LED's (12) as well as the number of pins (4). Create a marix, with 12 rows and two columns, with each row corresponding to an LED, while the columns refer to the pin states and pin configurations respectively. Within the loop function, create a `for loop` that runs as many time the number of LED's. Within the `for loop`, create a second `for loop` that runs as many times as the amount of pins used. Within the second `for loop`, assign each of the pins their corresponding configuration and state values. Plug the CharlieStick into the Arduino as an appliance and test the code, ensuring everything is working properly.

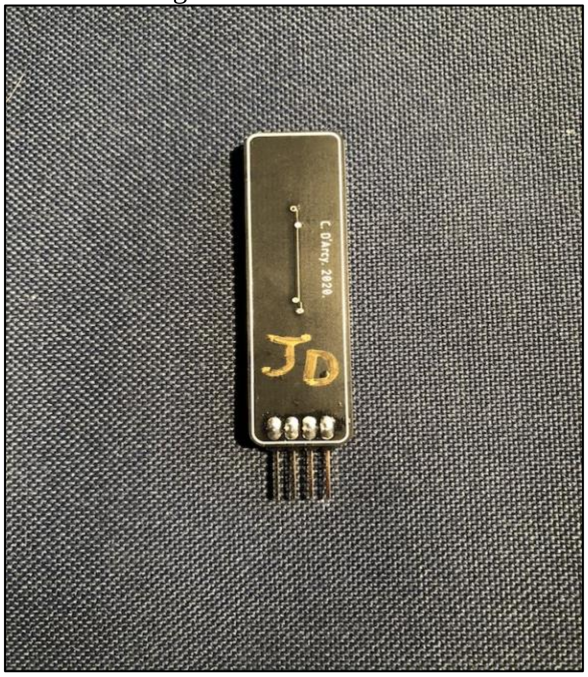
Media

https://www.youtube.com/watch?v=WzJIQHfZoLY

Soldered PCB

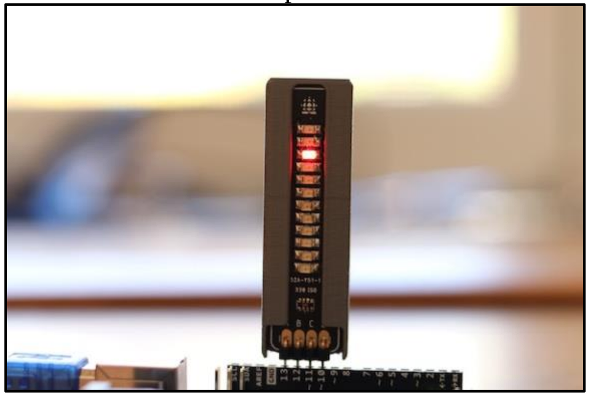


Signed back of the board



Case designed in Fusion 360

Final product



Code

```
//Project: Create an animation for the Charlie stick
//Author: Josh Dolgin
//Date: 2019/10/25
//Status: Working
#define PinCon 0
#define PinState 1
#define LED_COUNT 12
#define numPin 4 //How many pins are used for the stick
uint8_t stick[] = {10, 11, 12, 13}; //What pins are being used
uint8_t configuration[LED_COUNT][2][4] = {
  { { INPUT, INPUT, OUTPUT, OUTPUT }, { LOW, LOW, LOW, HIGH } }, // 1
  { { INPUT, INPUT, OUTPUT, OUTPUT }, { LOW, LOW, HIGH, LOW } }, // 2
  { { INPUT, OUTPUT, OUTPUT, INPUT }, { LOW, LOW, HIGH, LOW } }, // 3
  { { INPUT, OUTPUT, OUTPUT, INPUT }, { LOW, HIGH, LOW, LOW } }, // 4
  { { OUTPUT, OUTPUT, INPUT, INPUT }, { LOW, HIGH, LOW, LOW } }, // 5
  { { OUTPUT, OUTPUT, INPUT, INPUT }, { HIGH, LOW, LOW, LOW } }, // 6
  { { INPUT, OUTPUT, INPUT, OUTPUT }, { LOW, LOW, LOW, HIGH } }, // 7
  { { INPUT, OUTPUT, INPUT, OUTPUT }, { LOW, HIGH, LOW, LOW } }, // 8
  { { OUTPUT, INPUT, OUTPUT, INPUT }, { LOW, LOW, HIGH, LOW } }, // 9
  { { OUTPUT, INPUT, OUTPUT, INPUT }, { HIGH, LOW, LOW, LOW } }, // 10
  { { OUTPUT, INPUT, INPUT, OUTPUT }, { LOW, LOW, LOW, HIGH } }, // 11
  { { OUTPUT, INPUT, INPUT, OUTPUT }, { HIGH, LOW, LOW, LOW } }  // 12
};

void setup() {
}

void loop() {
  for (uint8_t l = 0; l < LED_COUNT; l++) { //Runs as many times as there are LED's
    for (uint8_t i = 0; i < numPin; i++) { //Runs as many times as the pins on the
stick
      pinMode(stick[i], configuration[l][PinCon][i]); //Sets the pin configuration
      digitalWrite(stick[i], configuration[l][PinState][i]); //Sets the pin state
    }
    delay(100);
  }
  for (uint8_t l = LEDCOUNT-1; l > 0; l--) { //Counts back down
    for (uint8_t i = 0; i < numPin; i++) {
      pinMode(stick[i], configuration[l][PinCon][i]); //Sets pin configuration
      digitalWrite(stick[i], configuration[l][PinState][i]); //Sets pin state
    }
    delay(100);
  }
}
```

Reflection

I was sitting in my little corner of the DES during a period 2 spare when Mr. D'Arcy introduced this project to me. I was handed a CharlieStick PCB and 12 surface mount LED's and given the option to solder them during my free time. Along with the LED's came an unfamiliarly tiny component, a 3216m resistor network. I was blown away by how small each of the solder pads were. After soldering 3 CharlieStick PCB's, each with a different value of resistor network, I can confidently say my comfortability with surface mount components has increased drastically. I was able to reduce the time to solder the PCB from 40 minutes to 8 minutes, all from practice. After soldering my PCB, I was met with a certain amount of relief; that was all the work I had to do for this DER. That relief was slowly overcome by my unease, I may not be able to code it. In an effort to get ahead, I did some research into charlieplexing, and the software behind it. To my surprise, within an hour or two, I had a somewhat concise piece of code that did exactly what I wanted it to do. In conclusion, part 1 of the CharlieStick process was a complete success for me, I bettered my surface mount soldering and coding skills, two things that were in need of improvement.

Project 19. CHUMP

Part 1: Code

Reference

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

Code

High Level	Machine Level		CHUMP (Assembly) Level	Comment
	Address	Instruction		
n=0	0000	**0000** 0000	**LOAD** 0	accum = 0; pc++;
	0001	**0110** 0010	**STORETO** 2	mem[2]<-accum; pc++;
for(x=5; x>0; x--)	0010	**0000** 0101	**LOAD** 5	accum = 5; pc++;
	0011	**0110** 0001	**STORETO** 1	mem[1]<-accum; pc++;
	0100	**1000** 0010	**READ** 2	addr<-2; pc++;
	0101	**0001** 0000	**LOAD IT**	accum = mem[2]; pc++;
	0110	**1000** 0001	**READ** 1	addr<-1; pc++;
n = n+x	0111	**0011** 0000	**ADD IT**	accum+mem[1]; pc++;
	1000	**0110** 0010	**STORETO** 2	mem[2]<-accum; pc++;
	1001	**1000** 0001	**READ** 1	addr<-1; pc++;
	1010	**0001** 0000	**LOAD IT**	accum = mem[1]; pc++;
	1011	**0100** 0001	**SUBTRACT** 1	accum--; pc++
	1100	**1100** 1110	**IFZERO** 14	accum==0?pc=14; pc++;
x!=0	1101	**1010** 0011	**GOTO 3**	pc<-0011
x=0	1110	**1100** 1110	**GOTO** 14	pc<-1110

Explanation

This code is my attempt at making an Arduino C `for loop` in chumpanese. First a constant 5 is loaded into the accumulator, and then 1 is subtracted from the 5. The subsequent result is stored in memory address 1. Then the constant in memory address 2 is loaded into the accumulator, in which the constant in memory address 1 is added. This result is stored back into memory address 2. Then the value in address 1 is put back into the accumulator. If the number located in the accumulator is 0, then the code goes to line 12 and the program ends, if the number in the accumulator is not 0, the program counter goes back to line 1. This program will run 5 times and each time it will add 5, 4, 3, 2, or 1, depending on how many times it has looped.

Part 2: Clock

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html - clock
https://youtube.com/watch?v=kRlSFm519Bo
https://www.youtube.com/watch?v=81BgFhm2vz8
https://www.youtube.com/watch?v=WCwJNnx36Rk
https://www.youtube.com/watch?v=SmQ5K7UQPMM

Purpose

The purpose of this project is to demonstrate the three possible states of a 555 timer IC; monostable, astable, and bistable. Through the use of these states and an additional 3 logic IC's, a clock logic circuit can be developed, which can switch between continuous astable oscillation output, and a manual debounced output. Within the 555 timer is an SR (Set Reset) latch, with two outputs, a Q output on pin 3, and an inverted Q output. Attached to the S or R pin of the SR latch is a comparator, which will compare the two analog signals, and output either a high or low signal depending on whether the voltage on the "–" pin is less than the voltage on the "+" pin. When the output is reset, the inverted output flows to the base pin of a NPN transistor, allowing current to flow from pin 7 to pin 1.

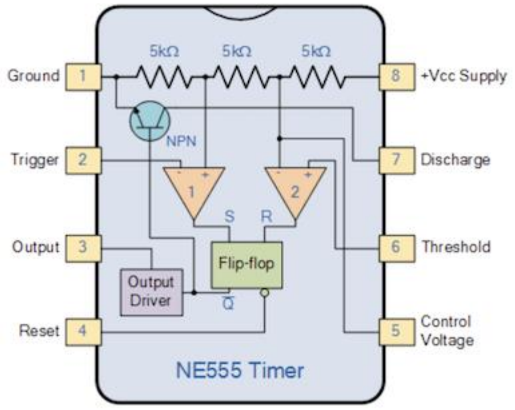
Monostable

The purpose of the monostable 555 timer within this circuit is to debounce the manual input from a PBNO. When a PBNO is pushed and therefore closed, the metal tabs that connect the two leads often touch multiple times, rather than once. When viewed on an oscillioscope, it is clear that the PBNO often bounces multiple time per input. When using a manual input to advance the Program Counter of the CHUMP, it is important that the program advances by 1 line each time the button is pressed, rather than multiple times. When the PBNO is pushed, the trigger pin (pin 2) is pulled low, causing the comparator to be turned on, activating the set pin of the SR latch within the 555 timer; therefor activating the output. As well, when the PBNO is closed, the capacitor begins to charge, and once the charge of the capacitor becomes greater than the threshold voltage of the second comparator, the latch is reset, and the capitor discharges.

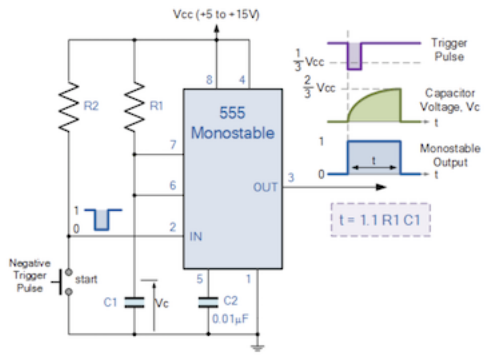
Astable

The purpose of the astable 555 timer within this circuit is to create an oscillatting clock pulse with a frequency or duty cycle that can be controlled by using a potentiometer. This

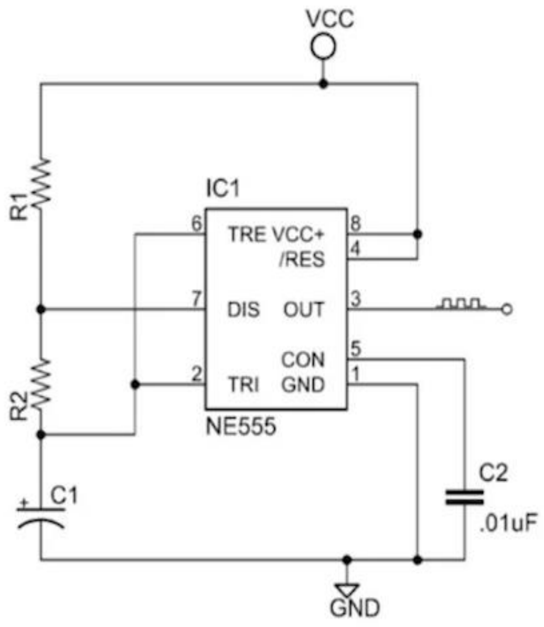
555 Timer



MonoStable 555 timer



Astable 555 timer



circuit is designed to create a square wave output from pin 3, which the frequency or duty cycle can be controlled by changing the values of resistance or capacitance of R1, R2, and C1. As the capacitor C1 begins to charge, the voltage will be less than that on the trigger pin, activating the output of the SR latch. As the capacitor continues to charge, at some point the charge of the capacitor will overcome the 3.3V on the "-" side of the second comparator, turning off the output and discharging the capacitor, which will in turn begin to charge the capacitor again until the voltage is below the trigger voltage, which will then turn on the output again. The frequency, period, and duty cycle can be calculated using the following formulae:

$$Period = 0.693(R1 + 2R2)C1$$

$$Frequency = \frac{1}{Period} = \frac{1.44}{(R1 + 2R2)C1}$$

$$Duty\ Cycle = \frac{R2}{R1 + 2R2}$$

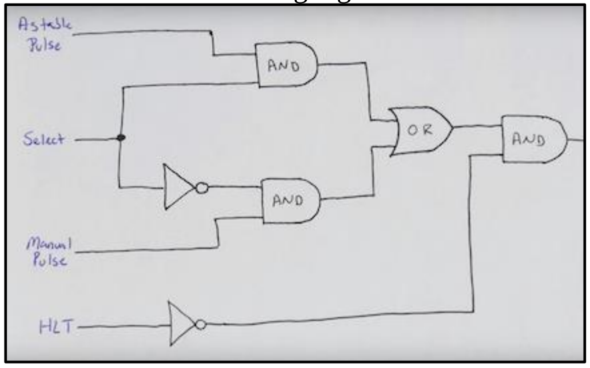
By adding a variable resistor instead of a fixed R2 resistor, the period, frequency, and duty cycle can all be changed by changing the resistance value.

Bistable

The purpose of the bistable 555 timer is to act as a flip-flop circuit, which alternates between two stable states, a high output and a low output. This circuit works by attaching the middle pin of a SPDT switch to ground, and attaching one of the poles to the trigger pin (pin 2) and the other pole to the reset pin (pin 4). When the trigger pin is tied to ground, the first comparator will activate the set pin of the SR latch, outputting a high voltage. When the reset pin is tied to ground, the voltage runs through an inverter, and then activates the reset pin of the SR latch, turning off the output.

Clock Logic

Clock logic gates



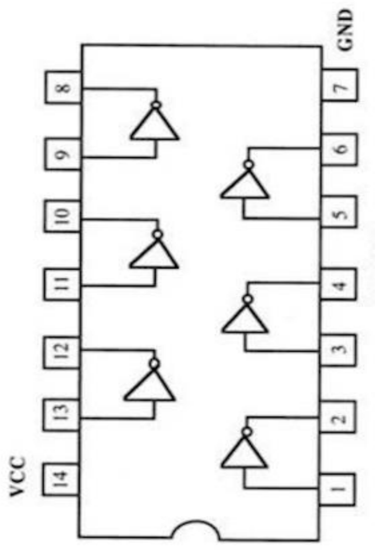
The goal of the clock logic circuit is to output either the astable pulse from the astable 555 circuit, or the manual pulse from the monostable 555 circuit, depending on the state of the bistable 555 circuit. There are 3 logic IC's used in the clock logic circuit, the 74LS04, which contains 6 inverters, the 74LS08, which contains 4 AND gates, and the 74LS32, which contains 4 OR gates If the select pin is high, then the AND gate with the astable pulse will be turned on and oscillate with the astable pulse, while the AND gate tied to the manual pulse will be 0, since the signal from the select pin goes through an inverter. If the select pin is low, then the AND gate connected to the manual pulse will go high, since the low signal is inverted, and the AND gate tied to the astable pulse will go low. Both of these AND gates is tied to an OR gate, and since only 1 of the AND gates outputs a signal at a time, either the manual pulse or the astable pulse is output from the circuit. Finally, there is a halt signal, which runs through an inverter and into an AND gate with the either astable or manual pulse. This is so that the program can be halted at any moment by applying a high signal to the halt pin, setting the output low.

Procedure

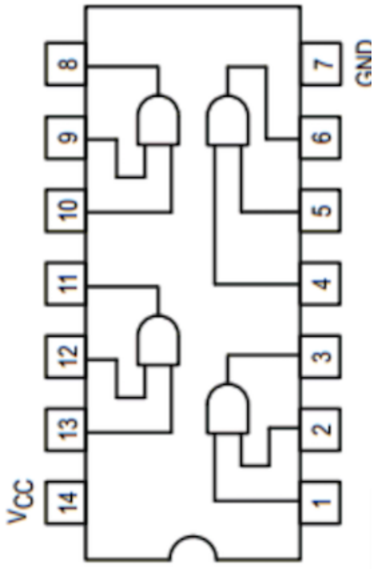
Parts Table	
Component	Quantity
555 timer	3
0.1μF capacitor	3
PBNO	1
SPDT slide switch	1
500kΩ potentiometer	1
Assorted fixed resistors	8
Assorted LEDs	4
1μF capacitor	1
74LS04 inverter IC	1
74LS08 AND IC	1
74LS32 OR IC	1

Open Ben Eater's 4-part video series of constructing the clock logic circuit (found in reference section). Gather all of the components listed in the parts table and lay them out. Place the first 555 timer into the breadboard and connect pin 8 and pin 1 to VCC and GND respectively. Connect pin 5 to ground via a 0.01μF capacitor. Connect pin 4 to VCC. Connect the center leg of the potentiometer to pin 6 of the IC, and tie pin 6 to pin 2. Connect pin 2 to ground via a 0.1μF capacitor. Connect pin 7 to VCC via a 10KΩ resistor. Connect pin 7 to the outer leg of the potentiometer via a 100KΩ resistor. Connect the anode of a LED to pin 3 and tie the cathode to ground via a 220Ω resistor. Place the PBNO into the breadboard and tie the first lead to GND. Connect the other lead of the PBNO to VCC via a 1KΩ resistor, as well as to pin 2 of the second 555 timer. Connect pin 8 and 1 of the second 555 timer to VCC and GND
respectively. Connect pins 6 and 7 via a jumper wire. Connect pin 6 to VCC via a 1MΩ resistor and connect pin 7 to GND via a 1μF capacitor. Connect the anode of another LED to pin 3 and connect the cathode to ground via a 220Ω resistor. Place the third 555 timer IC into the breadboard and tie pins 8 and 1 to VCC and GND respectively. Insert a SPDT slide switch into the breadboard and connect the center pin to GND. Connect one side of the switch to pin 4, and the other to pin 2. Connect pins 2 & 3 to VCC via a 1KΩ resistor. Connect pin 6 to GND and connect pin 5 to GND via a 0.01μF capacitor. Connect the anode of a LED to pin 3 and connect the cathode to GND via a 220Ω resistor. Insert the 3 remaining IC's into the breadboard. Connect pins 7 and 14 of the IC's to VCC and GND respectively. Connect pin 3 of the bistable 555 to pin 1 of the 74LS04 IC, and pin 1 of the 74LS08 IC. Connect pin 2 of the 74LS04 IC to pin 4 of the 74LS08 IC. Connect pin 9 of the 74LS04 IC to GND. Connect pin 8 of the 74LS04 IC to pin 10 of the 74LS08 IC. Connect the output of the astable 555 timer to pin 2 of the 74LS08 IC. Connect the output of the monostable 555 timer to pin 5 of the 74LS08 IC. Connect pins 1 and 2 of the 74LS32 IC to pins 6 and 3 of the 74LS08 IC respectively. Connect pin 3 of the 74LS32 IC to pin 8 of the 74LS08 IC. Connect pin 7 of the 74LS08 IC to the anode of an LED, and connect the cathode to GND. Connect 5V to the circuit and ensure each portion is working correctly, troubleshoot if needed.

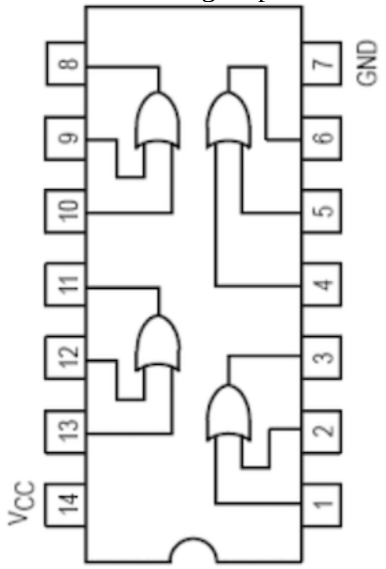
78LS04 NOT gate pinout



78LS08 AND gate pinout



78LS32 OR gate pinout



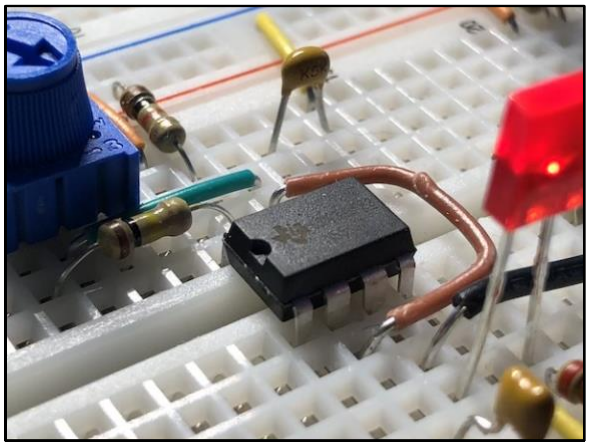
Reflection

Upon reflection, this was not an extremely difficult project from a physical perspective, but I had some issues with conceptualizing how each state of the 555 timer works. After a little bit of research and rewatching of Ben Eater's videos, I conceptually understood each part of this circuit. Surprisingly, there was nothing to debug in my circuit, each part worked the first time I tried it. Unfortunately I likely will not be as lucky during the rest of this project, as there will be plenty more to debug in the coming weeks.

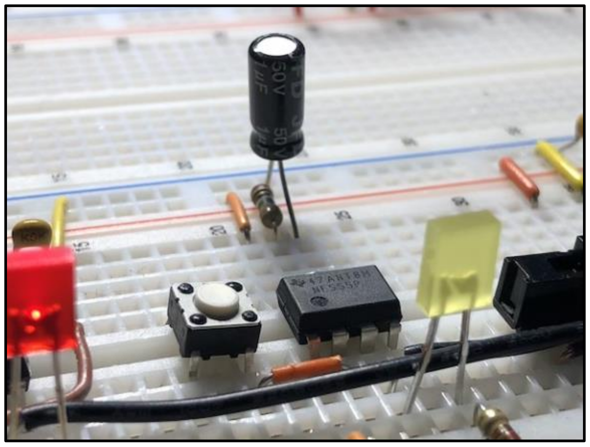
Media

https://www.youtube.com/watch?v=3kM5tD6Ou0M&feature=youtu.be

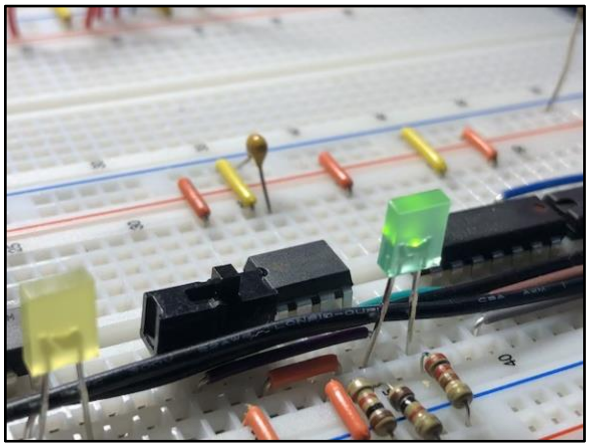
Astable 555 timer



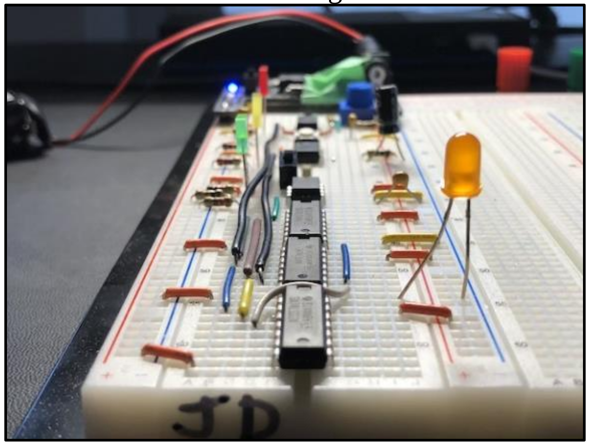
Monostable 555 timer



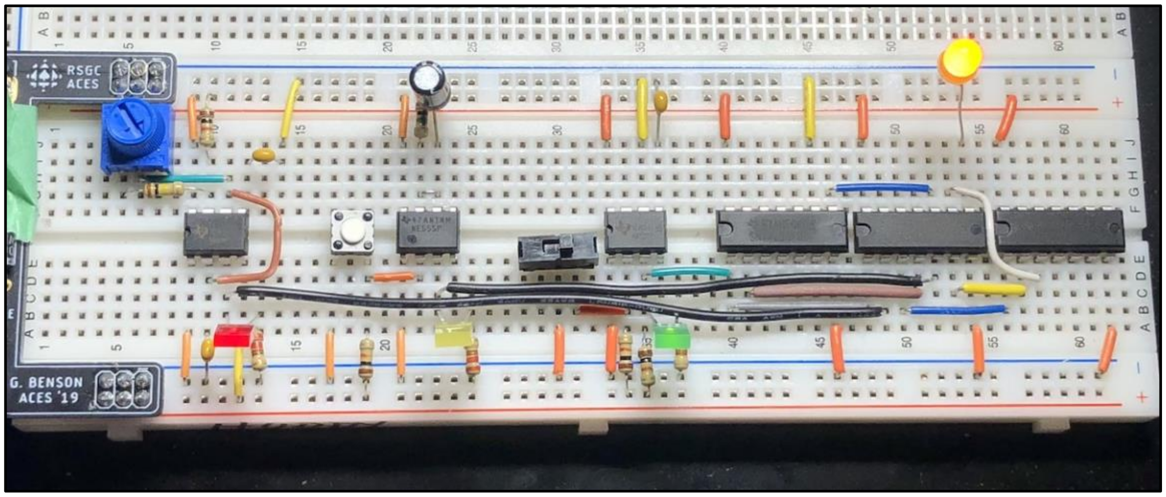
Bistable 555 timer



Clock logic



Final circuit



Part 3: Arithmetic and Logic Unit

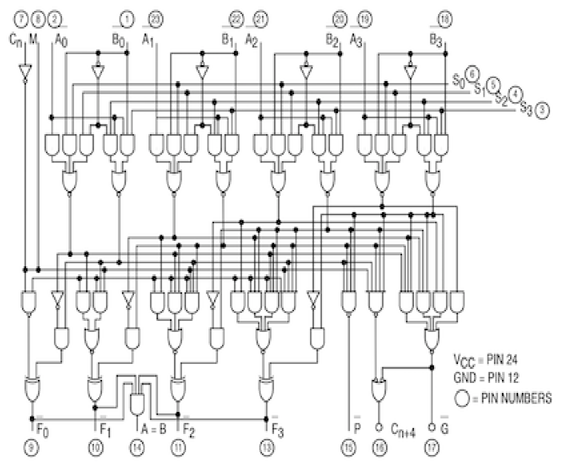
Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html - ALU
https://eater.net/8bit/alu
http://www.righto.com/2017/03/inside-vintage-74181-alu-chip-how-it.html

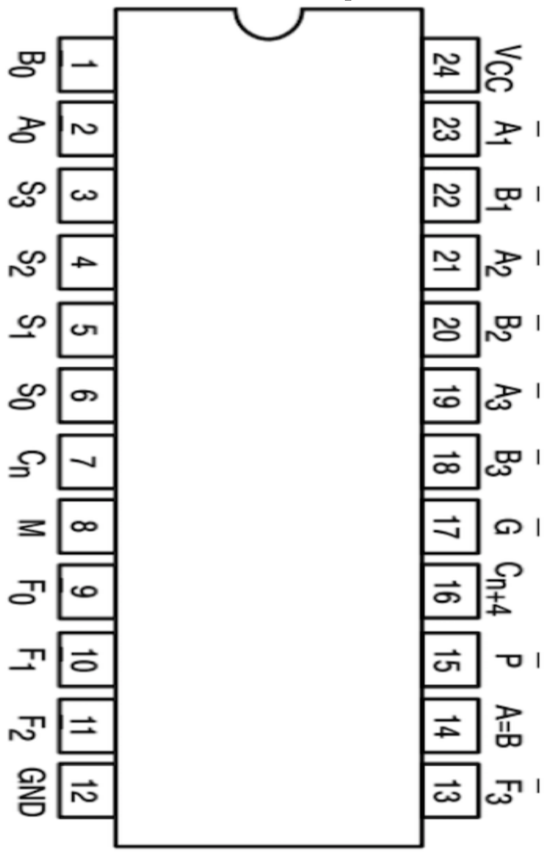
Purpose

The purpose of this project is to test the SN74LS181 Arithmetic Logic Unit (ALU); a 24 pin TTL IC commonly used in computers in the 1970s onward. The arithmetic unit takes two 4 bit input values (A_0-A_3 and B_0-B_3) and either performs arithmetic or bitwise logic functions depending on the state of the mode pin (M). The function which the ALU performs upon the two 4 bit integers is dependent on the states of the function select pins (S_0-S_3). Since there is 16 possible function select states (binary 0-15), there is 32 possible functions, 16 arithmetic and 16 logic. These functions are performed across a series of AND, OR, NOR, and NAND gates, computing the 4 outputs (F_0-F_3) and outputting a high or low signal on the outputs corresponding to the state of each computed bit. The C_n pin of the ALU is only used when performing arithmetic functions, and will add 1 to the output. Whereas the C_{n+4} will output a value if there is a set bit in the next most significant bit than F_3. These pins are useful when chaining multiple 181 ALU's together, as connecting the carry in pin (C_n) to the carry out pin (C_{n+4}) results in a ripple carry capable of computing more than 4 bits. Carry lookahead (C_{n+4}) uses "Generate" and "Propagate" signals (pins G and P) to determine if each bit position will always generate a carry bit or potentially could generate a carry out bit. For example, if you add 0+0+C (where C is a single binary bit), regardless of whether C is set or not, there will be no carry. If you add 1+0+C or 0+1+C, whether or not there is a carry is dependent on the C value. Finally in the case of 1+1+C, there will always be a carry generated. The ultimate goal of this stage in the CHUMP process is to wire the 181 ALU on a breadboard and manipulate both arithmetic and logic functions using a DIP switch, developing a deeper understanding of the IC in the process.

Logic gates within the ALU



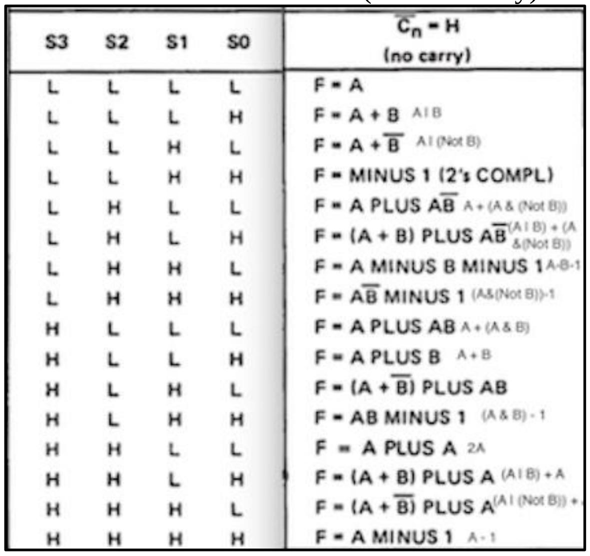
SN74LS181 ALU pinout



Arithmetic

In order to perform arithmetic functions, the mode select pin is set low, and the carry in pin can be set high or low, but since multiple ALU's are not beign tied together carry in will remain tied high to signify no carry in digit. Arithmetic functions differ from logic functions by allowing bits to be carried over to the next significant bit position. For instance, if the function was set to binary 1001, the arithmetic A plus B function would be performed. If A and B both had a set bit in the same significant bit position, the output would carry the bit to the next most significant bit position. Within the arithmetic functions list, there is certain specific logic functions, and combined logic and arithmetic functions.

Arithmetic Function table (without carry)

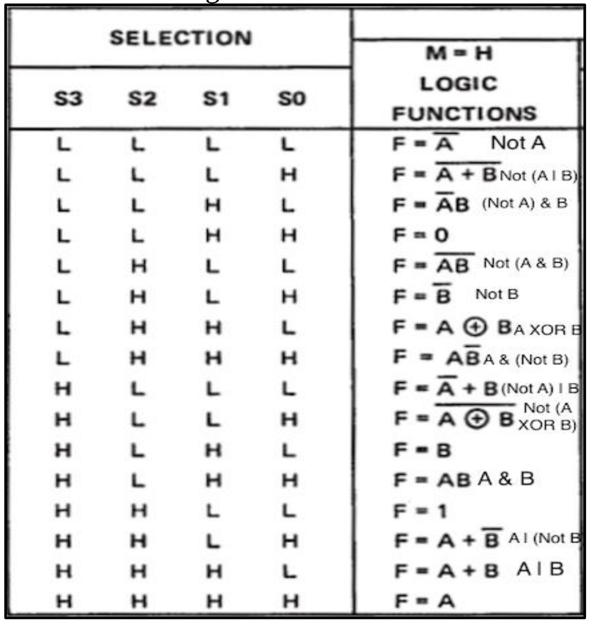


S3	S2	S1	S0	$\overline{C_n}$ = H (no carry)
L	L	L	L	F = A
L	L	L	H	F = A + B A | B
L	L	H	L	F = A + $\overline{B}$ A | (Not B)
L	L	H	H	F = MINUS 1 (2's COMPL)
L	H	L	L	F = A PLUS A$\overline{B}$ A + (A & (Not B))
L	H	L	H	F = (A + B) PLUS A$\overline{B}$ (A | B) + (A &(Not B))
L	H	H	L	F = A MINUS B MINUS 1 A-B-1
L	H	H	H	F = A$\overline{B}$ MINUS 1 (A&(Not B))-1
H	L	L	L	F = A PLUS AB A + (A & B)
H	L	L	H	F = A PLUS B A + B
H	L	H	L	F = (A + $\overline{B}$) PLUS AB
H	L	H	H	F = AB MINUS 1 (A & B) - 1
H	H	L	L	F = A PLUS A 2A
H	H	L	H	F = (A + B) PLUS A (A | B) + A
H	H	H	L	F = (A + $\overline{B}$) PLUS A (A | (Not B)) +
H	H	H	H	F = A MINUS 1 A - 1

Logic

The logic functions perform strictly bitwise logic, meaning no carries are needed, so the carry in bit can be neglected, C_n can be connected to VCC or GND. Bitwise logic means that each numbered bit of the output is dependent on the logic function acted on the two bits of the same significance in input A or input B. An interesting property to note about the function table is that the inverse of the bits within a function selection results in a inverse operation. Such as function 0001, which is NOT(A OR B), the inverse of this function is 1110, which is (A OR B). Although some functions seem quite complex and unneeded, they could be quite useful in practice.

Logic Function table



SELECTION				M = H LOGIC FUNCTIONS
S3	S2	S1	S0	
L	L	L	L	F = $\overline{A}$ Not A
L	L	L	H	F = $\overline{A + B}$ Not (A | B)
L	L	H	L	F = $\overline{A}$B (Not A) & B
L	L	H	H	F = 0
L	H	L	L	F = $\overline{AB}$ Not (A & B)
L	H	L	H	F = $\overline{B}$ Not B
L	H	H	L	F = A ⊕ B A XOR B
L	H	H	H	F = A$\overline{B}$ A & (Not B)
H	L	L	L	F = $\overline{A}$ + B (Not A) | B
H	L	L	H	F = $\overline{A \oplus B}$ Not (A XOR B)
H	L	H	L	F = B
H	L	H	H	F = AB A & B
H	H	L	L	F = 1
H	H	L	H	F = A + $\overline{B}$ A | (Not B
H	H	H	L	F = A + B A | B
H	H	H	H	F = A

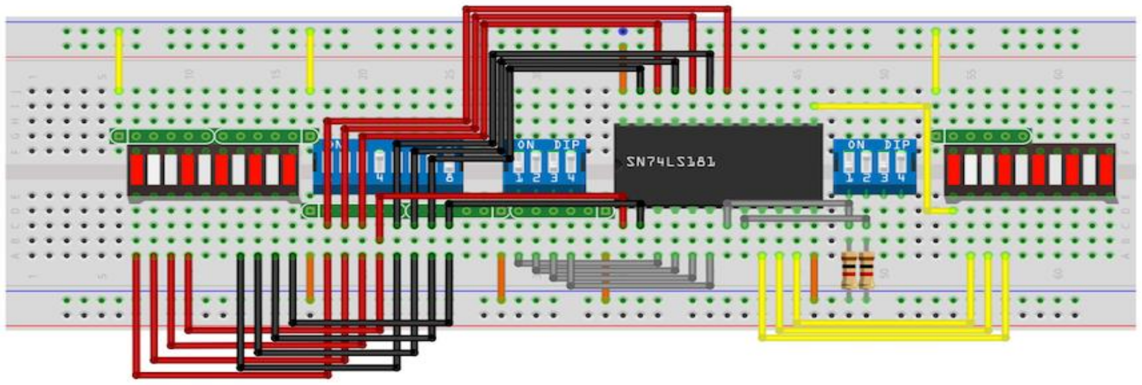
Procedure

Gather all the components listed in the parts table and lay them out. Insert the first LED pargraph into a breadboard, followed by the 16 pin DIP switch, the 8 pin DIP switch, the 181 ALU IC, the second DIP switch, and finally the second bargraph, ensuring they are each separated by at least a single row on the breadboard. Connect pins 10-14 and 17-20 of the first bargraph to GND, each set via a 10KΩ 5 pin bussed resistor network. Do the same for pins 7-10 of the second bargraph. Use the 1KΩ 8 bin bussed resistor network to create pull down

Parts Table	
Component	Quantity
SN74LS181 ALU IC	1
16 pin DIP switch	1
8 pin DIP switch	2
10 LED bargraph	2
1 KΩ fixed resistor	7
1 KΩ 8 bin resistor network	1
10 KΩ 5 pin resistor network	3
Wires	Assorted*

resistors for the 8 pin DIP switch, and use the 1KΩ fixed resistors to create pull down resistors to the remaining DIP switch pins. Connect the remaining leads of the 3 DIP switches to VCC. Connect pin 12 and pin 24 of the ALU to VCC and GND respectively. Connect pins 1-4 of the first DIP switch to pins 1-4 of the first bargraph, these are the A inputs. Connect pins 1-4 of the DIP first DIP switch to pins A_0-A_3 of the ALU, where pin 1 corresponds to the most significant bit (A_3). Connect pin 5-8 of the input DIP swtich to pins 7-10 of the first bargraph, and in a similar fashion, connect the outputs to inputs B_0-B_3 of the ALU. Connect pins 1-4 of the second DIP switch to the 4 function control pins, where pin 1 corresponds to S_3 and pin 4 coresponds to S_0. Connect pins 1 and 2 of the third DIP switch to the carry in and mode select pin of the ALU respectively. Finally, connect pins F_0-F_3 to pins 1-4 of the second bargraph.

Fritzing diagram



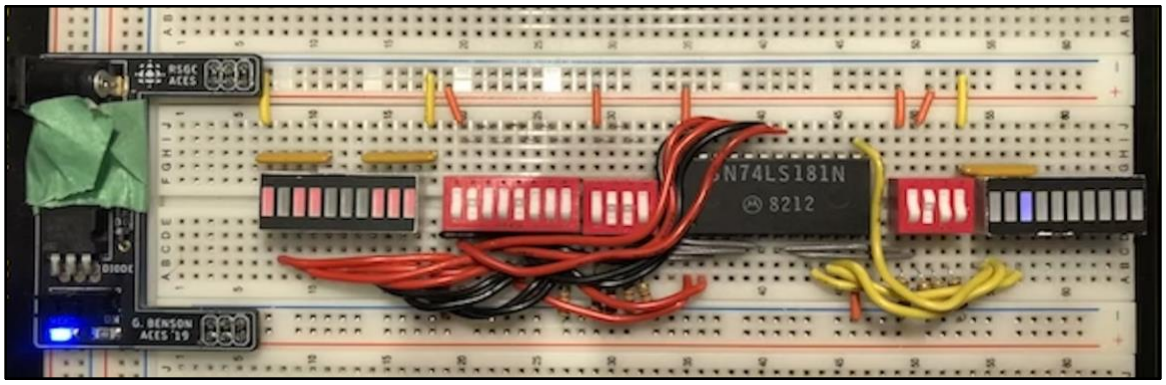
Reflection

When the class was handed our assignments for what IC we were to present on, I learned that I was responsible for the ALU, one of the more complex IC's used in the CHUMP. Mr.D'Arcy handed me some IC's to tinker with on the weekend before my presentation, so I began to test and learn about the IC. After multiple more hours of research and probing the Arithmetic Logic Unit, I had finally begun to understandall the capabilities of the IC. I introduced the IC to my classmates, and offered as much help and resources as I could so they would finish their project as well. Although I encountered some issues wiring the final circuit, and had to troubleshoot for a while, I solved the problem and continued to work away on my DER.

Media

https://www.youtube.com/watch?v=IkSzYp3VTkY

Wired Circuit



Part 4: EEPROM

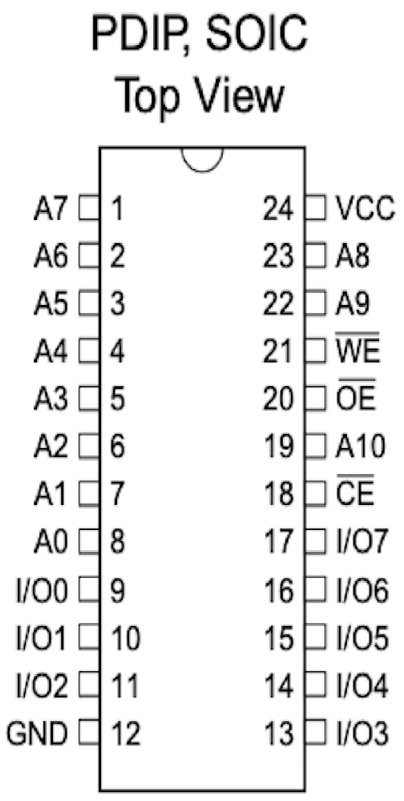
Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html - EEPROM
https://www.youtube.com/watch?v=BA12Z7gQ4P0
https://www.youtube.com/watch?v=K88pgWhEb1M&t=305s

Purpose

The purpose of this project is to wire a simple circuit and develop code to write and read 8 bit values from the first 16 EEPROM addresses. EEPROM stands for Electrically Eraseable Programmable Read-Only Memory, this version of Programmable Read-Only Memory is used in this project, so that the values in each address can be erased and changed using digital pins from an Arduino Nano. The AT28C16 EEPROM offers a 16K memory organized as 2048 words by 8 bits. In order to provide 2048 separate addresses (0-2047) 11 address pins are used to designate which specific address will be used. Since the CHUMP is a 4-bit computer, only 4 address pins are needed, and the remaining 7 can be tied to GND. As well within the 24 pin DIP package is 8 data Input Output (I/O) pins, depending on the state of the output enable pin and write enable pins, the 8 I/O pins can either act as an ouput pin showing what is in the selected address, or as an input pin allowing the user to write to the specified address. It is important to note that the enable pins are active low pins, so they must be connected to GND in order for them to activate. Through using 14 pins on the Arduino Nano to control 4 address pins, 8 I/O pins, as well as the /WE and /OE pins, a code will be developed to write the previously written CHUMP: code into respective addresses within the EEPROM and then read those values, viewing the entire process in the serial monitor.

AT28C16 EEPROM pinout



Procedure

Parts Table	
Component	Quantity
AT28C16 EEPROM	1
Wires	Assorted*
Arduino Nano	1

Place the AT28C16 EEPROM and the Arduino Nano into a breadboard. Connect the 5V pin from the Arduino to the power rail and the GND pin to the GND rail. Connect pins 12 and 24 of the EEPROM to VCC and GND respectively. Connect pins A4-A11 to GND. Connect the /CE pin to GND. Connect pins A0-A3 of the EEPROM to pins A0-A3 of the Arduino. Connect I/O pins 0-7 to pins 2-9 on the Arduino.

Connect the /CE and /WE pins to pins 10 and 11 of the Arduino. Open the Arduino IDE and create a new sketch entitled "WriteEEPROM". Within an array, place the binary machine instructions from the previously written CHUMP: Code. Define each of the address pins, I/O pins, and enable pins. Set the address pins to output. Print the hex values that are to be written to EEPROM. To write to EEPROM, write the /OE pin low, and the /WE pin high. Create a `for loop` that declares all of the I/O pins for output. Create a bit mask that AND's bit 8 with a 1, outputs that value to the corresponding I/O pin within the `for loop,` then shift the data left 1 position. This means that the outptus of the I/O pins will reflect the binary value. To read EEPROM, this process is basically inverted, the /WE pin is written high and the /OE pin is written low. Write a `for loop` that moves throughout the I/O pins and shifts the value of that pin to its corresponding bit position, then writing that value to the serial monitor.

Code

```
// PROJECT  :AT28C16Write
// PURPOSE  :Writes data to the AT28C16 (2Kx8) EEPROM IC
// COURSE   :ICS4U
// AUTHOR   :B. Eater. adapted for ACES CHUMP use by C. D'Arcy
// DATE     :2019 11 13-16
// MCU      :Nano/328
// STATUS   :Working
// REFERENCE:B. Eater Github...
//          :https://github.com/beneater/eeprom-
programmer/blob/master/eeprom-
//           programmer/eeprom-programmer.ino
// REFERENCE:B. Eater Videos
//          1. Using an EEPROM to replace combinational logic
//          https://www.youtube.com/watch?v=BA12Z7gQ4P0&feature=emb_logo
//          2. Build an Arduino EEPROM programmer
//          https://www.youtube.com/watch?v=K88pgWhEb1M&feature=emb_logo
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// CHUMP Basic Program Example
//00000101 LOAD 5     ;accum = 5; pc++;
//01100001 STORETO 1  ;mem[1]<-accum; pc++;
//10000010 READ 2     ;addr<-2; pc++;
//00010000 LOAD IT    ;accum = mem[2]; pc++;
//10000001 READ 1     ;addr<-1; pc++;
//00110000 ADD IT     ;accum+mem[1]; pc++;
//01100010 STORETO.   ;2 mem[2]<-accum; pc++;
//10000001 READ 1     ;addr<-1; pc++;
//00010000 LOAD IT    ;accum = mem[1]; pc++;
//01000001 SUBTRACT 1 ;accum--; pc++
//11001100 IFZERO 12  ;accum==0?pc=12; pc++;
//10100001 GOTO 1     ;pc<-0001
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// populate array below with machine instructions above
byte code [] = {
  0b00000101,
  0b01100001,
  0b10000010,
  0b00010000,
  0b10000001,
  0b00110000,
  0b01100010,
  0b10000001,
  0b00010000,
```

```
  0b01000001,
  0b11001100,
  0b10100001
};
#define PROG_SIZE sizeof(code)
#define EEPROM_D0 2
#define EEPROM_D7 9
#define EEPROM_OE 10
#define EEPROM_WE 11
#define EEPROM_A0 14
#define EEPROM_A3 17

void setup() {
for (uint8_t pin = EEPROM_A3; pin >= EEPROM_A0; pin--) {
  pinMode(pin, OUTPUT);
  }
  digitalWrite(EEPROM_WE, HIGH);
  pinMode(EEPROM_WE, OUTPUT);
  digitalWrite(EEPROM_OE, HIGH);
  pinMode(EEPROM_OE, OUTPUT);
  Serial.begin(9600);
  Serial.println("Here's the data to be flashed to EEPROM...");
  for (int address = 0; address < PROG_SIZE; address++) {
    Serial.print(code[address], HEX);
    Serial.print("  ");
  }
  Serial.println();
  // Write the code to EEPROM...
  Serial.println("Writing " + String(PROG_SIZE) + " bytes of code to 
EEPROM...");
  for (int address = 0; address < PROG_SIZE; address++) {
    writeEEPROM(address, code[address]);
  }
  Serial.println("Done");
 // Confirm the write by reading the code in the serial monitor
  Serial.println("Reading EEPROM");
  printContents();
}

// Read the contents of the EEPROM and print them to the serial monitor.
void printContents() {
  for (int address = 0; address < PROG_SIZE; address++) {
    Serial.println(readEEPROM(address), HEX);
  }
}

// Output the address bits and outputEnable signal using shift registers.
void setAddress(int address) {
  for (int pin = EEPROM_A3; pin >= EEPROM_A0; pin--) {
    digitalWrite(pin, address & 0x08);
    address <<= 1;     //destructive
  }
}
byte readEEPROM(int address) {
  digitalWrite(EEPROM_WE, HIGH);
  for (int pin = EEPROM_D7; pin >= EEPROM_D0; pin--) {
    pinMode(pin, INPUT);
  }
  byte value = 0;
  setAddress(address);
  digitalWrite(EEPROM_OE, LOW);
  delayMicroseconds(1);
  digitalWrite(EEPROM_OE, HIGH);
  for (int pin = EEPROM_D7; pin >= EEPROM_D0; pin--) {
    Serial.print(digitalRead(pin));
    value = (value << 1) + digitalRead(pin);
  }
  Serial.print(" ");
  return value;
}
//  Write a byte to the EEPROM at the specified address.
void writeEEPROM(int address, byte data) {
  digitalWrite(EEPROM_OE, LOW);
  digitalWrite(EEPROM_WE, HIGH);
  Serial.println("[" + String(address) + "]" + String(code[address], HEX));
 setAddress(address);
  digitalWrite(EEPROM_OE, HIGH);
  //prepare to write the data...
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin++) {
    pinMode(pin, OUTPUT);
  }
  //write the data...
  for (int pin = EEPROM_D7; pin >= EEPROM_D0; pin--) {
    digitalWrite(pin, data & 0x80);
    data <<= 1;   //destructive....
  }
  digitalWrite(EEPROM_WE, LOW);
  delayMicroseconds(1);
  digitalWrite(EEPROM_WE, HIGH);
  delay(10);
}
void loop() {
  // nothing left to do...
}
```

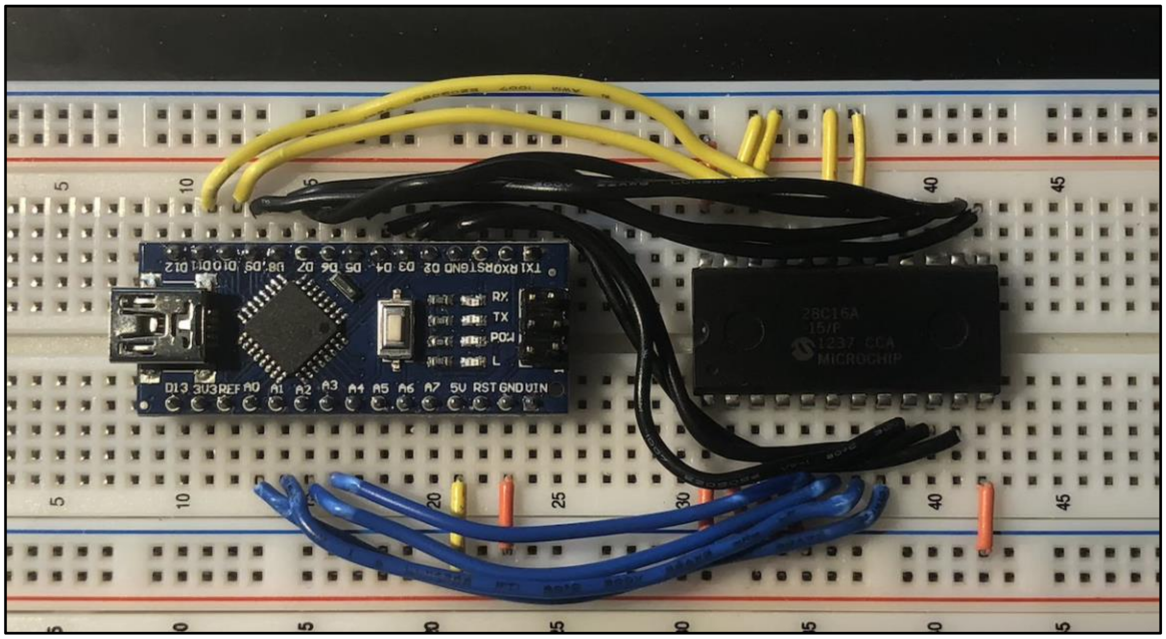
Reflection

This was not an extremely difficult project to complete. We were given the code in class, and only needed to make simple adjustments to create a functioning circuit. The wiring was intuitive and simple, all the connections coming out of the AT28C16 EEPROM IC either went to a rail or to the Arduino. The IC itself is quite interesting, how it is able to store that much data, and how specific the timing must be to utilize the chip.

Media

https://www.youtube.com/watch?v=rTyUNsrv6sM

Wired circuit



Part 5: Program Counter

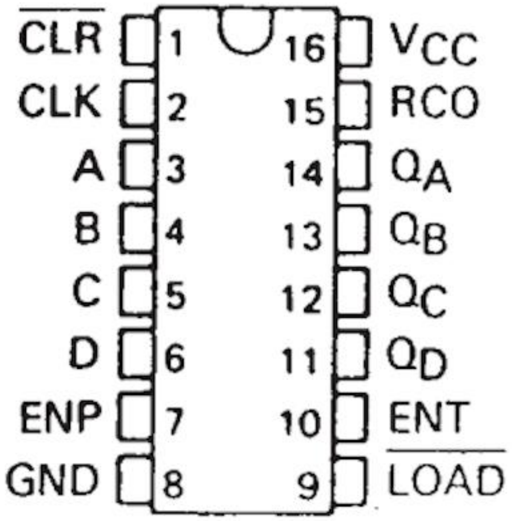
Reference

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

Purpose

The purpose of this section of the CHUMP build to wire the clock pulse from the Eater clock circuit into the SN74LS161 IC, creating a 4-bit counter to run through the previously written CHUMP program. The SN74LS161 synchronus counter is a 4-bit counter that increments by 1 with each clock pulse, outputting the 4-bit value across outputs Q_A-Q_D. Once confirming that the program counter portion of the CHUMP build works by placing 4 LED's across outputs Q_A-Q_D, the outputs will be wired to address pins A_0-A_3 of the program EEPROM, which was flashed with the CHUMP code in the previous section. This will create a circuit to run through the first 16 addresses of the AT28C16 EEPROM IC.

SN74LS161 Pinout



Procedure

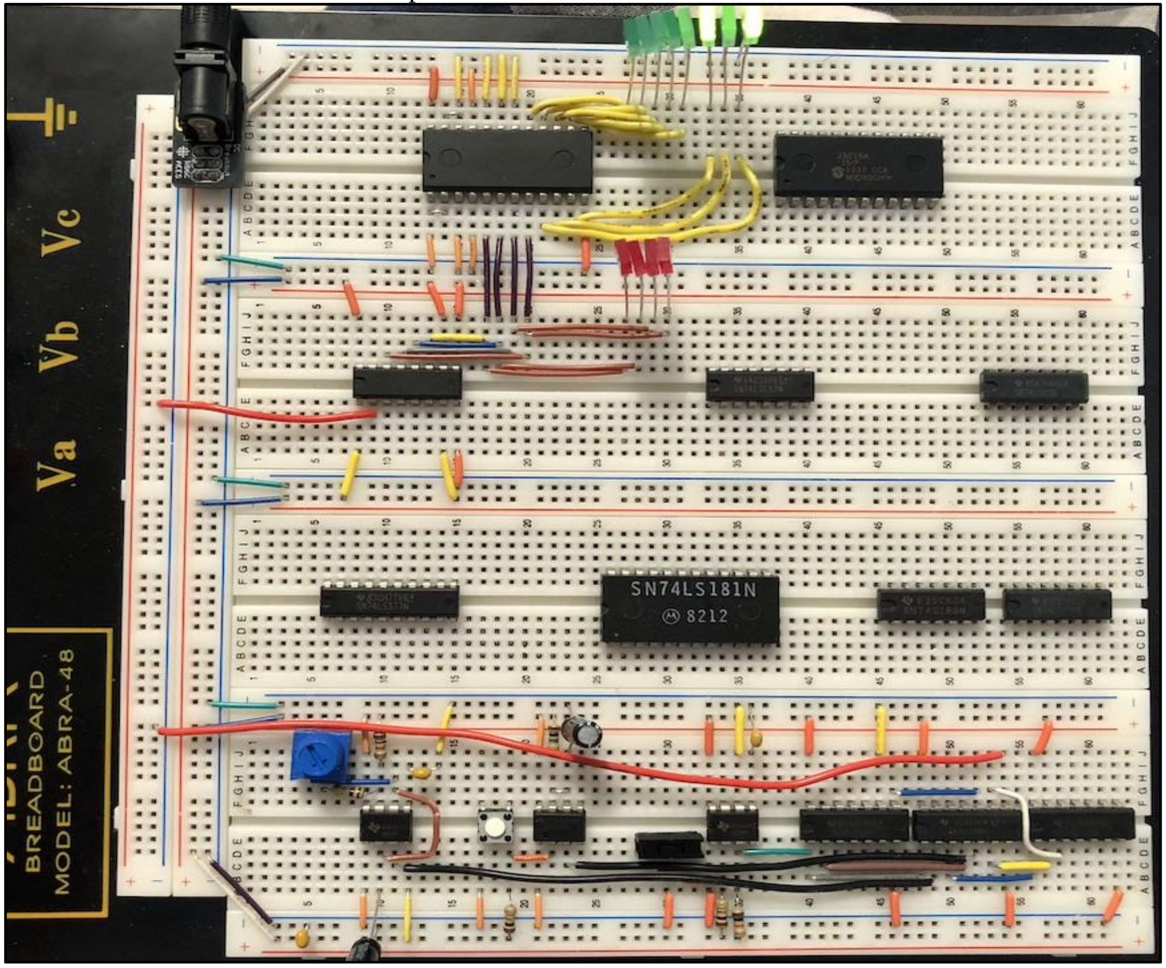
Parts Table	
Component	Quantity
SN74LS161 4-bit counter IC	1
AT28C16 EEPROM IC	1
5mm rectangular LED	12
Wires	Assorted*

Place the SN74LS161 synchronus 4-bit counter IC into the breadboard. Connect the final clock output from the Eater clock build to the unconnected ground rail at the top of the breadboard, and connect this clock pulse to the CLK pin of the IC. Open the ACES TTL processor instruction word document and scroll to the section on the SN74LS161 IC. Since it is not required to clear the output of the IC, the clear (/CLR) pin must be tied high. The enable P and enable T pins, are also to be wired high. Connect pins 8 and 16 to VCC and GND respectively. The /LOAD pin of the IC, when wired low, will output the values placed on inputs A-D to pins Q_A-Q_D; therefor, the load pin will be tied high to select synchronous counting. Also, there is the ripple carry out pin, which is not needed in this project, this pin is used to connect multiple synchronous counter IC's together in order to achieve counting with more than 4 bits. On the EEPROM IC, tie addresses 4-11 to GND, tie the /WE pin high, the /OE pin low, and the /CE pin low.

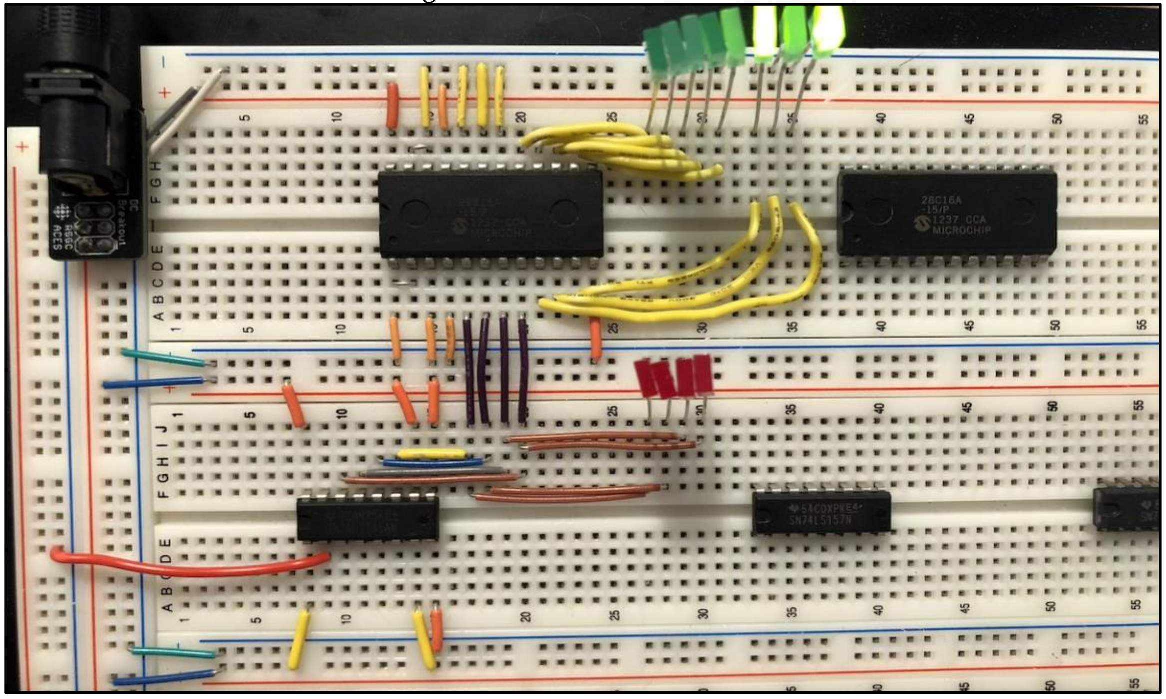
Media

https://www.youtube.com/watch?v=uF9kMlv5h88

Complete circuit with Eater clock build



Program counter EEPROM circuit



Reflection

Upon reflection, this iteration of the CHUMP build was not particularly more difficult than the previous ones. I understood the concept of the circuit immediately once it was explained, and see how this new IC fits into the CHUMP circuit. I completed the wiring for this project in one of my spares during the week, and after a minor inconvenience; getting the bit orders of the outputs and inputs on the counter and EEPROM IC a little confused, the build was complete. In conclusion, overall this has been a rewarding process conceptually, but too many concepts from other courses mixed in with this project make it quite hard to retain a work ethic and focus.

Part 6: Processor

Reference

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

Purpose

The purpose of this stage in the CHUMP build is to utilize the knowledge and experience the class has in each persons respective IC's, as well as share the knowledge gained in the previous steps of the build to hopefully arrive at a working CHUMP. The purpose of the CHUMP is to take the code from the previously flashed program EEPROM, and use those program codes to dictate what control codes (stemming from the op-codes) will be sent across the build to perform specific functions. Ultimately the CHUMP processor should be able to run the code written in CHUMP part 1, while the outputs of certain necessary pieces within the build can be visualized using 3mm LED's. Once confirming that the build works with the code already written, it is possible to reflash the program EEPROM with more recent iterations of code written in chumpanese, to test the full capabilities of the processor within the 4-bit constraint. This is a very general overview of the CHUMP final build, the detailed purpose of each essential IC within the build will be listed below.

Multiplexor

The 74LS157 multiplexor(or selector) IC acts as a way to choose whether a memory or constant is loaded into the CHUMP. There is 4 sets of inputs to the multiplexor (A_1-B_1, A_2-B_2, etc), and each of these input combinations is fed to a single output (Y_1-Y_4). Either the A or B inputs are fed to the outputs depending on the state of the select pin (pin 1). If pin 1 is set high, then the Y outputs mirror the B inputs, whereas if the select pin is low, the Y outputs mirror the A inputs. By feeding the lower nibble of the program EEPROM IO pins into input A, the inverted ram ouptuts into input B, and the select pin to program EEPROM IO_4, the multiplexor is able to select between a constant or memory output depending on the state of bit 4 of the program code.

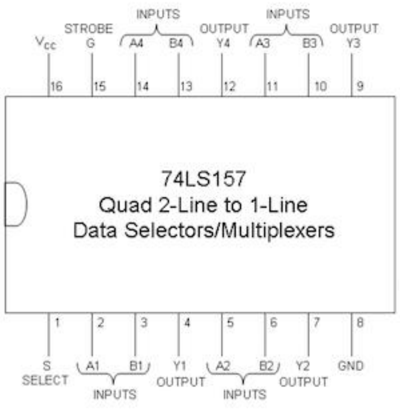
ALU

The purpose of the ALU within the CHUMP processor is to act as the mathematical heart of the build. The ALU gathers it's B inputs from the output of the multiplexor(selector), and performs the specified operation, whether arithmetic or logic. The outputs from the ALU are fed through the accumulator and into the Ram IC. The ALU is used to perform primarily addition, and subtraction, as well as the output F = B function to load a value from the multiplexor into the accumulator. At the core the ALU is ultimately responsible for a large proportion of the CHUMP's functionality, which is furthered by the fact that 6 of the 8 control code bits are used to dictate the function select, mode select and carry in pin of the ALU.

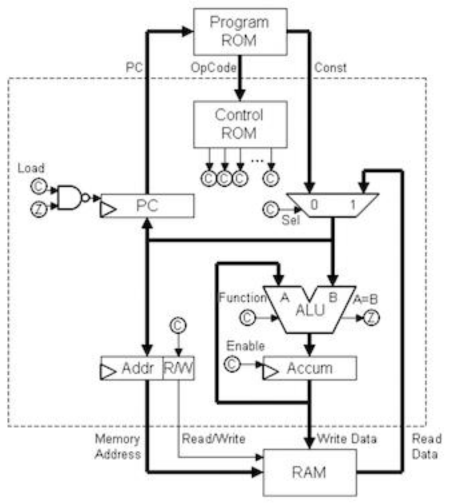
RAM

The LS189 Random Access Memory IC serves to hold the memory values for the CHUMP. The ram holds 16 addresses of 4 bit values, that can be written to or read from depending on the state of the WE pin of the IC. The address pins (A_0-A_3) and WE pin are dictated by the outputs of the address register, and the ram inputs(D_1-D_4) are controlled by the accumulator outputs. The inverted ram outputs are finally fed into the B inputs of the multiplexor.

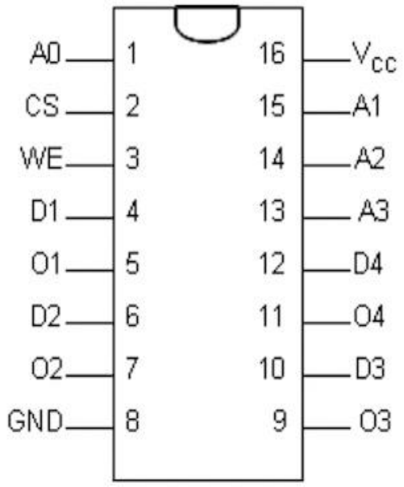
74LS157 Multiplexor pinout



CHUMP datapath



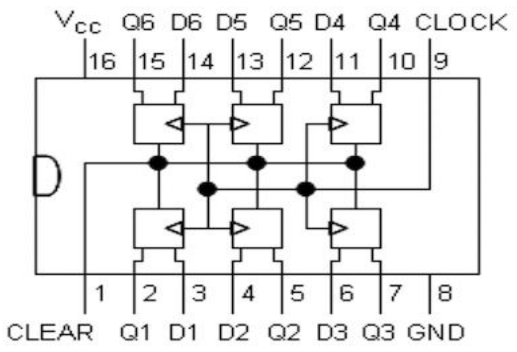
74LS189 RAM pinout



Address register

The purpose of the address register in the CHUMP build is to control what addresses of RAM are being accessed, as well as the state of the R/W pin of the RAM. The address register takes its inputs from the multiplexor outputs, and on the next clock pulse pushes these values to the RAM, dictating what address is being read from/written to.

74LS174 Hex Flip-Flop pinout



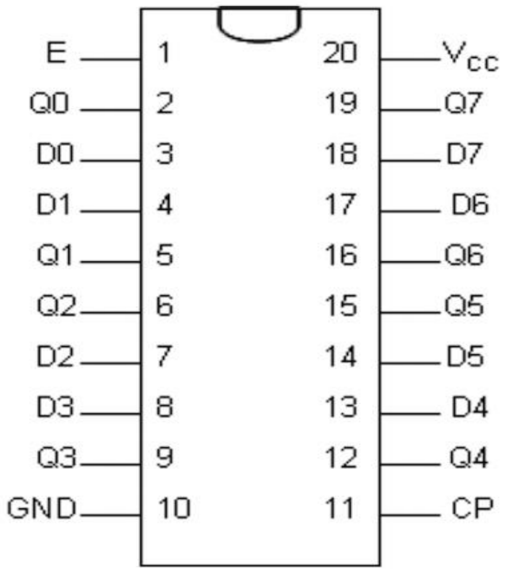
Program/Control EEPROM

The first 15 RAM addresses of the program EEPROM house the previously written and flashed CHUMP code. The written code is flashed into each individual address, incrementing the address by 1 each time per line of code. Within the build, the program EEPROMs input comes from the program counter outputs. The upper nibble of the IO output pins from the program EEPROM are fed into the address pins of the control EEPROM, as each op-code will correspond to a specific address in control EEPROM to perform that specified function. The lower nibble output is fed into the A inputs of the multiplexor, and finally, IO_4 is connected to the select pin of the multiplexor, selecting the lower nibble output if the bit is a 0 (constant) and selecting a memory command if the bit it high. The IO pins of the control EEPROM are spread across the entirety of the CHUMP build, and each 8 bit value housed in a EEPROM address will perform a very specific function on the remaining CHUMP components. The control lines are responsible for dictating the ALU functions (S_0-S_3), the ALU mode select (M) and carry in pins (CN), the accumulator enable (E) pin and finally the RAM WE pin. The control and program EEPROM are responsible for controlling the entire datapath of the CHUMP, and extremely crucial in the final steps of the build.

Accumulator

The purpose of the accumulator is to accumulate the outputs from the ALU (F_0-F_3) and on the next clock cycle, these inputs will be transferred to the outputs, similar to the address register. The D pins are the input pins to the accumulator, while the Q pins are the output pins. The enable input pin (E) is connected to control EEPROM through one of the control wires and the CP pin connects to the clock pulse.

SN74LS377 8-bit register pinout



Program Counter

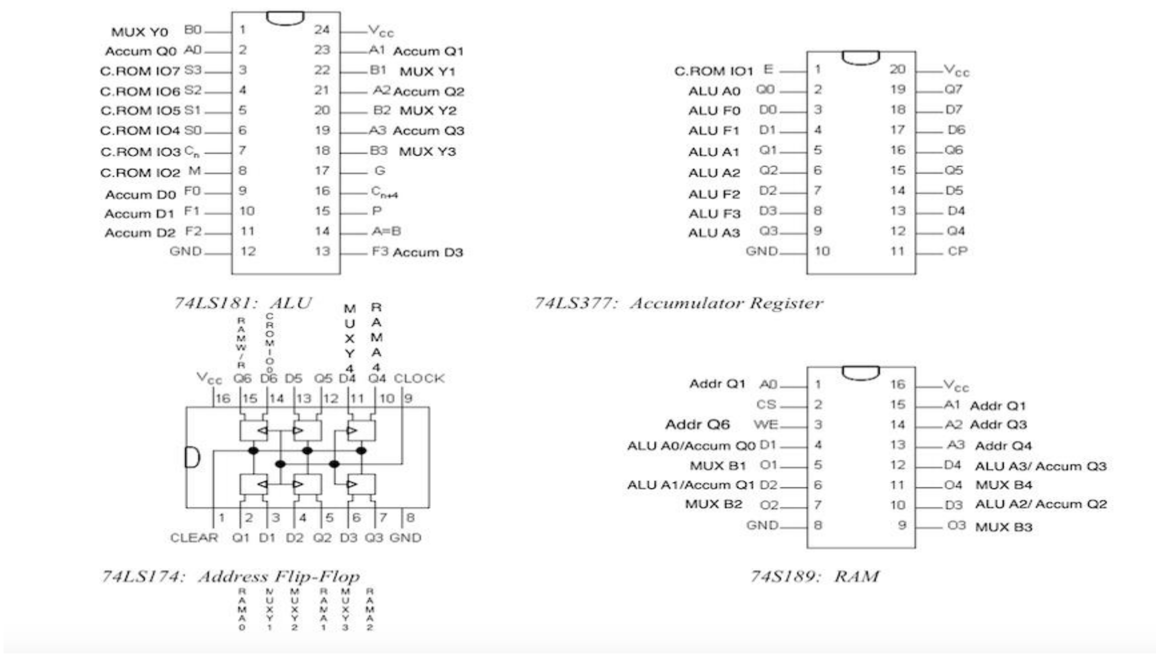
The program counter functions to increment it's output after each clock cycle; this is useful to increment the code in program EEPROM line by line. The only exeption to the incrementing output is with a GOTO or IFZERO command.

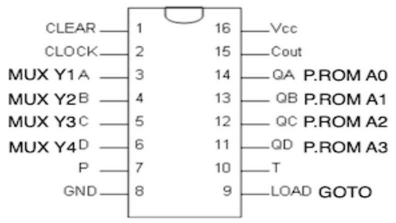
Procedure

Parts Table	
Component	Quantity
Ben Eater clock circuit	1
AT28C16 EEPROM IC	2
74LS157 Multiplexor	1
74LS174 Hex Flip-Flop	1
SN74LS377 8-bit register	1
74LS189 RAM	1
SN74LS181 ALU	1
SN74LS161 4-bit counter	1
SN74LS00N NAND IC	1
SN74LS04 NOT IC	1
3mm LED	Assorted*
Wires	Assorted*

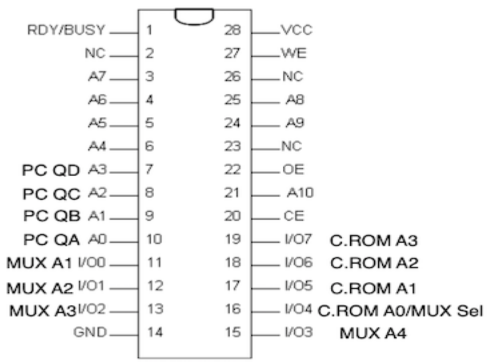
In order to maintain some clarity in wiring and to ease debugging, a certain wire color will be chosen to pertain to certain datapaths. In this CHUMP build the colors are as follows, Blue: Multiplexor and ALU outputs, Yellow: Multiplexor/Address register inputs, Black: Control EEPROM address inputs and IO outputs, RED: Clock input, Grey: RAM inputs and address register outputs, Green: bitwise logic for jump and Z bit. The CHUMP wiring is fairly complex, but as long as each connection is marked properly and easy to recognize, debugging should (hopefully) not be as difficult. With the exeption of ground and VCC connections, all important connections are marked on the pinout diagrams below. Important things to note are: include pull up resistors on the RAM outputs and A=B pin of the ALU of resistance 1KΩ or greater. The RAM outputs must pass through the SN74LS04 inverter IC to the B inputs of the multiplexor, or else the read memory values will be inverted. The remaining address pins on both the EEPROMs must be tied to GND. The /CS pins on all the required IC's must be tied to GND. The /OE pin of the EEPROMs are to be Grounded, and the /WE pin must be tied high. In order to create the bitwise logic for the GOTO and IFZERO commands, NAND the IO_7 and IO_6 outputs of the program EEPROM, then feed the output into the input of a new NAND gate. Connect the input's of the second NAND gate together to create a makeshift AND gate. Connect the output of the makeshift AND gate to the input of a new NAND gate, with the other input being the A=B pin of the ALU. Finally tie this output to the load pin of the program counter.

Connections

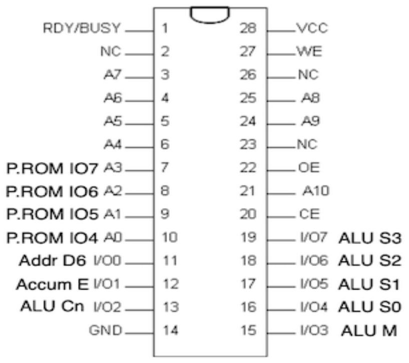




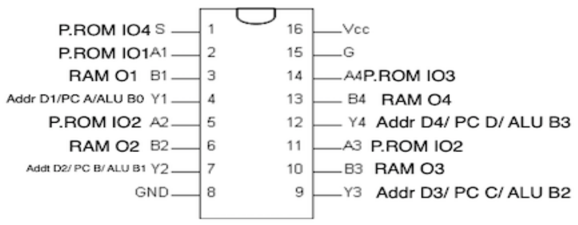
74LS161: Program Counter



28C17: Program ROM



28C17: Control ROM



74LS157: Selector

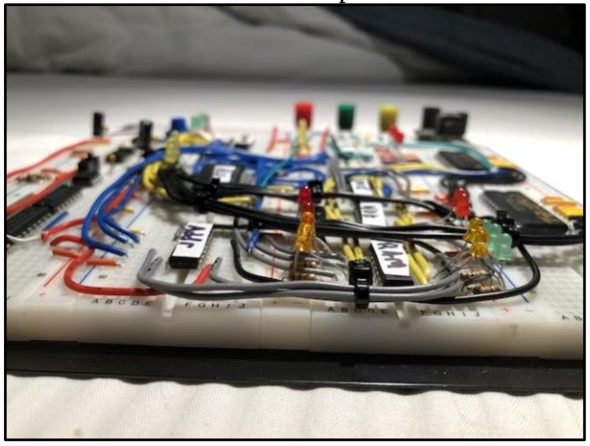
Reflection

This was one of the most rewarding projects of my ACES, or even high school career. Just after the short ISP presentations, and before we started investigating the CHUMP, Mr. D'Arcy came up to me; telling me that he knows I am more of a "design guy", but it is crucial that I stay involved in the CHUMP process despite my interest in domains other than hardware. Although I can attest to be sort of a "design guy", this project was a great way to further my hardware skill. Typically with design or software, you get sort of an instant gratification with each step of the process working. For me the CHUMP was extremely binary (*pun intended*), either it works and I can feel content with myself, or it doesn't and I'll spend the next weeks beating myself up over what I consider to be a failure. Needless to say, I poured hours of time into finalizing my CHUMP build. I spent period 1 and period 4 spares, free Saturdays, late nights in the DES trying to understand each section of the CHUMP: and I learned a considerable amount. I spent enough time on my CHUMP that I became sort of a CHUMP TA, answering questions from the entirety of the grade 12 ACES class at every opportunity they could ask. Overall, I couldn't feel happier or more rewarded by the outcome of this project. I went from the design kid who was anticipated to spend the CHUMP periods on his computer designing in Fusion 360, to the kid so involved in his CHUMP he would rather come to the last day of the school year at 7:00 even with a period 1 spare, just to see the multitude of effort he put in pay off.

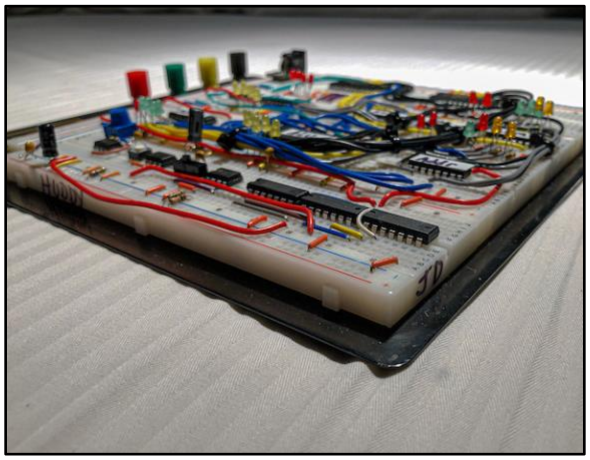
Media

https://www.youtube.com/watch?v=gL__OgbNLjg

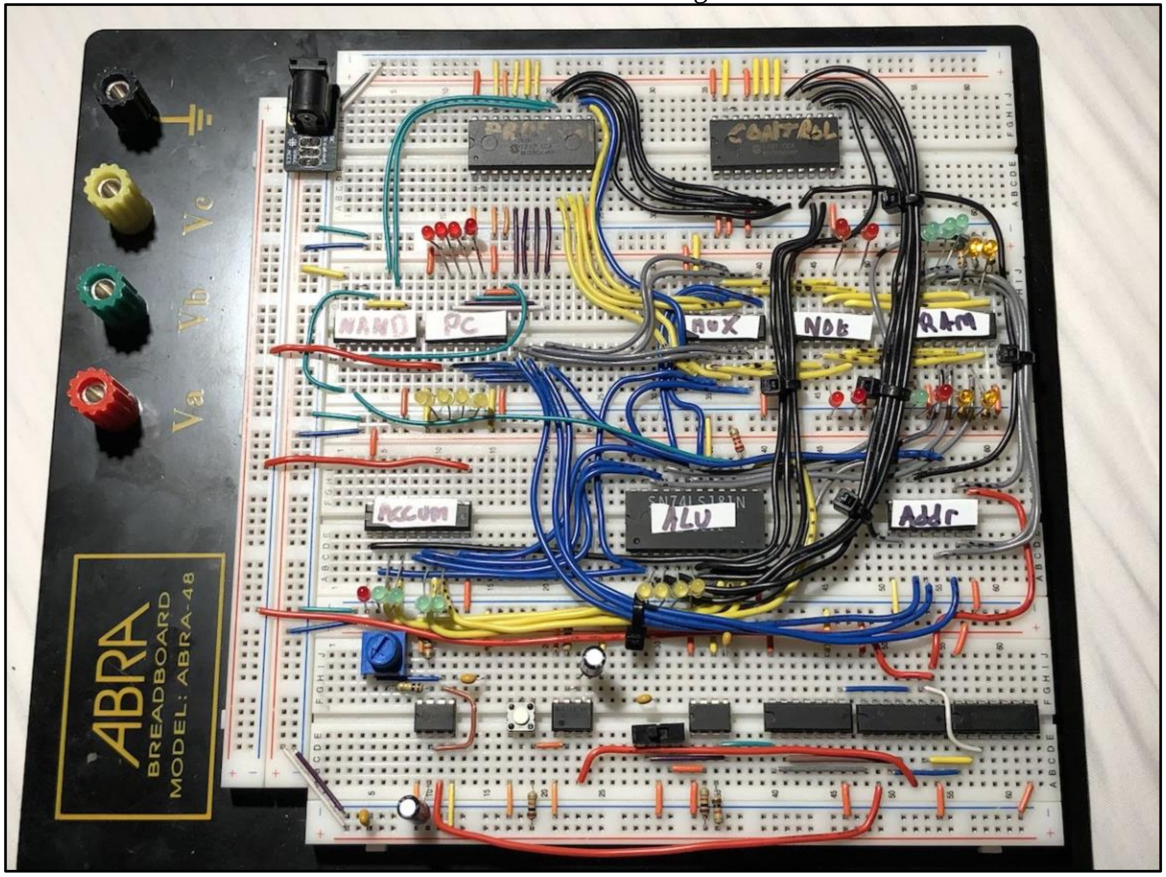
CHUMP side profile



CHUMP front view



CHUMP final wiring



Project 20. Dolgin Development Platform

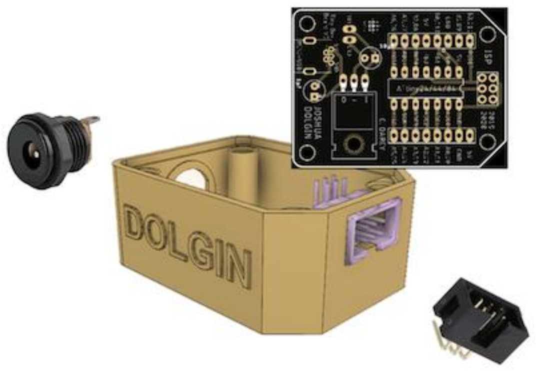
Part 1: Assembly

Reference

Project 13. Legacy PCB/Appliance
http://darcy.rsgc.on.ca/ACES/TEI4M/DolginDevPlatform/index.html#assembly

Purpose

DDP PCB & Case



The purpose of this project is to solder and begin testing a refined version of the previously created Dolgin Development Board, now named the Dolgin Development Platform. The core of the board remains the same (see project 13 for details), with 2 8 pin female headers for appliance connections, and a 2x3 ISP header (now shrouded in the case), as well as the 5V regulation circuit and AtTiny24/44/84 DIP-14 solder pads. Most of the DDP redesign has come in the form of the power supply to the board. Since the ATMEL ICE does not supply voltage to the board, multiple options are available for the user. Either the sparkfun pocket programmer can be used, a 5V micro usb cable can be used to supply power once code is flashed, or the board can run using a regulated 5V input from a DC barrel jack.

Procedure

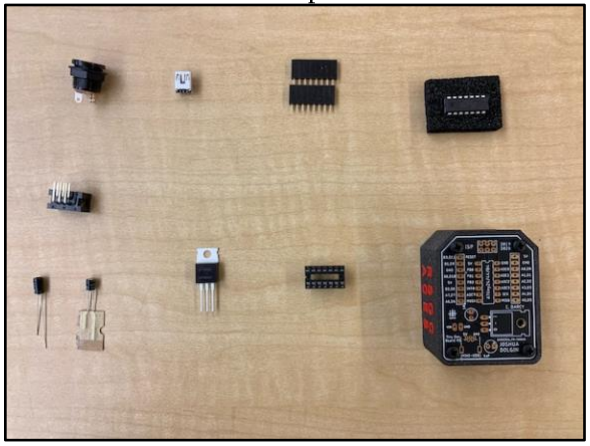
Parts Table	
Component	Quantity
ATtiny84	1
14 pin IC socket	1
USB type B connector	1
5V regulator	1
1x8 female header	2
10 µF capacitor	1
1 µF capacitor	1
Power jack	1
Red & black hookup wire	2
2x3 shrouded ISP header	1
DDP v5 PCB	1
M3 screws	4
DDP case	1

Open the ziplock bag containing the components for the DDP board and case, and lay them out along a silicon pad. Solder the components laying directly on the board first; beginning with the 14 pin IC socket, then continuing to the USB connector, 5V regulator and filtering capacitors, and the two 1x8 female headers. Solder a red wire to the positive terminal of the power jack and black wire to the negative terminal, being sure to heat shrink the connections afterwards. Insert the jack into the DDP case, then solder the red and black leads to VIN and GND pins respectively. Insert the 2x3 female ISP header into the designated slot in the DDP case, then place the PCB over top to solder the connections. Finish by screwing the board into the case using the given mounting screws. Follow the same process as previously listed in project 13 to upload a new sketch to the device and confirm its functionality.

Media

https://www.youtube.com/watch?v=vR8WSHAPsVg

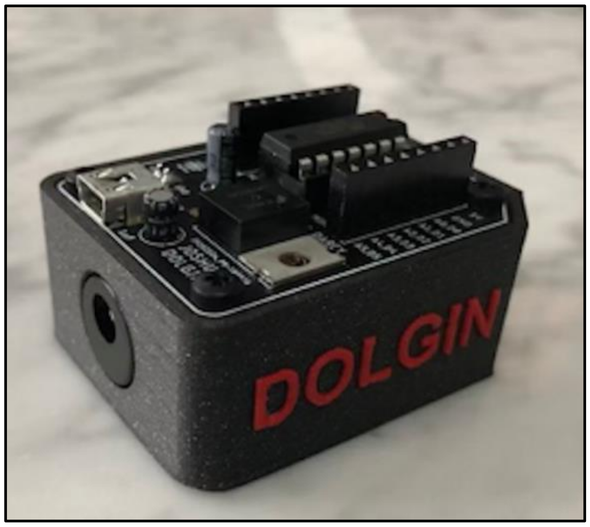
DDP components



Soldered DDP board

Reflection

DDP board encased



So here we are, turns out my legacy PCB would not only leave my legacy in the DES, but would also come back for me and my peers in our Grade 12 year. I have to admit, there is a sense of accomplishment seeing your name mentioned frequently across the ACES website, not to mention a series of projects all beginning with your last name, it is truly incredible. Upon reflection to the assembly of this board, nothing new really happened. I have soldered 4 variations of this board multiple times, so there were no issues in understanding orientations or the purpose of any aspect of the board, the only thing that differed this assemply from the previous is that my peers are completing it along side me. Being able to offer my knowledge towards another ACES project proves again to me that even though my interests are heavily design related, I am still capable of helping with software and hardware

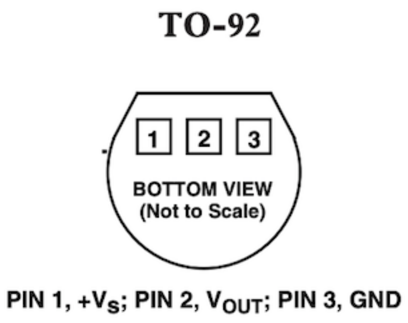
Part 2: Testing

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/DolginDevPlatform/index.html - testing

Purpose

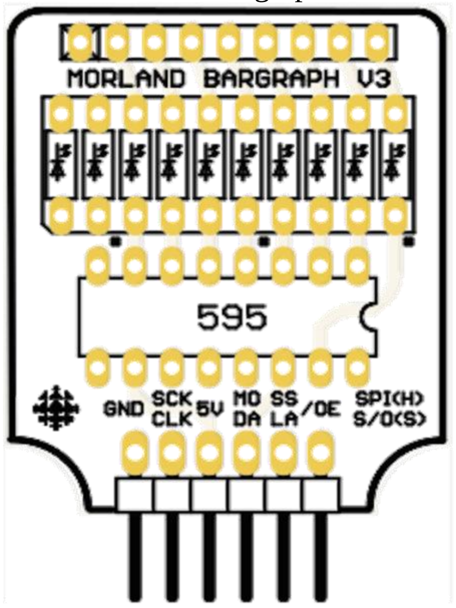
TMP36 Temperature sensor pinout



The purpose of this project is to refresh some forgotten high-level coding skills through various media such as the shaffer traffic light, morland bargraph, and various sensors to generate data that can be displayed. After multiple months of CHUMPANESE and hardware focus, it is important to remember some high-level techniques before starting some more assembly. The multiple coding goals of this project entail: demonstrating the blink sketch, traffic light, a simple bargraph animation, a fading effect using the OE pin of the morland bargraph, an animation to represent analog input data from a TMP36 temperature sensor, finalized by a display for a sensor of ones choosing. To ease with software development, the pinout of the DDP has been carfully thought of. Beginning with the breakout pins on the left side of the board, which break GND and 5V pins in such a way that the bargraph can plug in as an appliance, without having to source or sync voltage from a digital pin on the board, freeing the remaining digital pins to perfrom other tasks. On the other side of the board is another 5V and GND pin, as well as 6 analog/digital pins of ascending order(0-5).

Reflection

Morland Bargraph v3



In reflection, this was a fairly difficult and stressful project to complete. I had a lot of other things to work on this week, and I just procrastinated by coding and working on hardware. I am upset that I couldn't get the joystick or temperature sensor to work perfectly, but believe I should be proud of the effort I put in, and my reluctance to choose an easier sensor rather than try and finish what I started. The frustration came from the fact that the code worked on my Arduino board, but not my DDP. Overall I would still consider this section of the DDP a success, because I got some of my code to work well, and I wrote it in concise and modular sections to the best of my ability. I am slowly becoming more acclimated to the software world.

Procedure

Plug the traffic light into 3 digital pins of the DDP as an appliance. Develop updated code that uses an array housing the 3 used pins, and a `for loop` that changes which LED is on based off the output from the 3 cells in the array. Plug the morland bargraph into the DDP, so that the 5V and GND pins of both boards match. Code must be developed to create animations for: a simple animation on the LED bargraph, use OE pins to create a breathing affect on the bargraph, to display input from a joystick on the bargraph, and finally to display the input data from a TMP36 temperature sensor on the bargraph. The entire left side of the PCB is used for the morland bargraph (it blocks the remaining two pins), so the left side is the only available for sensor input data. Unfortunately due to the limitated pin numbers on the DDP, only the joystick and traffic light will be used for the combined code, and the TMP36 will have a separate block of code. Plug the traffic light into pins 3, 4, and 5, and the joystick into GND, VCC, and pins 0-2. Combine the code written for each individual module into a single sketch, changing the animation from either button or joystick input.

Parts Table	
Part	Quantity
DDP	1
Morland Bargraph	1
Shaffer traffic light	1
Sparkfun joystick	1
TMP36 temperature sensor	1

Code

Code for traffic light, joystick, and bargraph

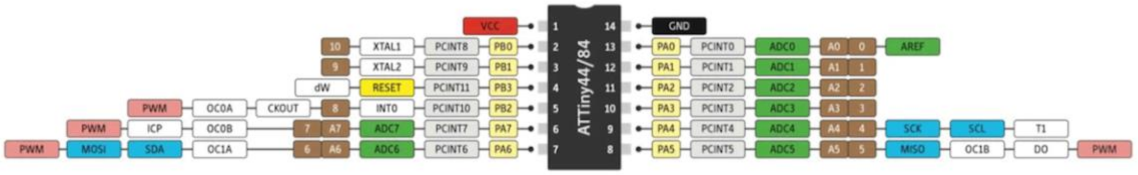
```
#define clk 10
#define data 8
#define latch 7
#define OE 6
#define SPEED 5
#define yVal 1
#define xVal 0
#define BUTTON 3
#define COUNT 2000
uint8_t LED[] = { 3, 4, 5 };
uint8_t numLED = sizeof(LED);
uint8_t value [] = { 0x01, 0x03, 0x07, 0x0F, 0x1F, 0x3F, 0x7F, 0xFF};
uint8_t animation = 0;
uint8_t light = 1;
uint8_t delta = 1;
uint16_t pos = 0;
uint16_t brightness = 0;
uint8_t increment = 0;
uint8_t whatAnimation = 0;

void setup() {
  for (uint8_t i = clk; i > OE; i--) {
    pinMode(i, OUTPUT);
  }
  for (uint8_t x = 0; x < numLED; x++) {
    pinMode(LED[x], OUTPUT);
  }
}
void animation1() { //Breathing bargraph
  delay(SPEED);
  digitalWrite(latch, LOW);
  shiftOut(data, clk, MSBFIRST, value[animation]);
  digitalWrite(latch, HIGH);
  analogWrite(OE, light);
  light = light + delta;
  if (light == 255 || light == 0)
  {
    delta = -delta;
  }
  if (light == 0) {
    animation == 7 ? animation = 0 : animation++;
    brightness++;
  }
}

void adjustBrightness() {
  brightness = analogRead(yVal);
  analogWrite(OE, brightness);
}

void animation2() { //traffic light
  for (uint8_t i = 0; i < numLED; i++) {
    digitalWrite(LED[i], HIGH);
    delay(COUNT);
    digitalWrite(LED[i], LOW);
  }
}

void animation3() { //Joystick animation
  pos = analogRead(xVal);
  while (pos > 900) {
    for (uint8_t x = increment; x < 7; x++) {
      digitalWrite(latch, LOW);
      shiftOut(data, clk, LSBFIRST, value[x]);
      digitalWrite(latch, HIGH);
      increment = x;
      adjustBrightness();
    }
  } while (pos < 300) {
    for (uint8_t x = increment; x > 0; x--) {
      digitalWrite(latch, LOW);
      shiftOut(data, clk, LSBFIRST, value[x]);
      digitalWrite(latch, HIGH);
      increment = x;
      adjustBrightness();
    }
  }
}
void loop() {
  if (digitalRead(BUTTON)) {
    whatAnimation++;
  }
  switch (whatAnimation % 3) {
    case 0:
      animation1();
      break;
    case 1:
      animation2();
      break;
    case 2:
      animation3();
      break;
  }
}
```

```
                         Code for temperature sensor
#define clk 10
#define data 8
#define latch 7
#define OE 6
#define SPEED 5
uint16_t temp = 0;
uint8_t value [] = { 0x01, 0x03, 0x07, 0x0F, 0x1F, 0x3F, 0x7F, 0xFF};
void setup() {
  for (uint8_t i = clk; i > OE; i--) {
    pinMode(i, OUTPUT);
  }
}

void loop() {
 temp = analogRead(A0);
 temp = constrain(temp, 140, 170); //≈ 20 - 30 ˙C
 digitalWrite(latch, LOW);
 shiftOut(data, clk, LSBFIRST, temp);
 digitalWrite(latch, HIGH);
 delay(50);
}
```

Media

https://www.youtube.com/watch?v=nLkLmkLTCUQ

ATtiny pinout



DDP with morland bargraph

DDP with traffic light

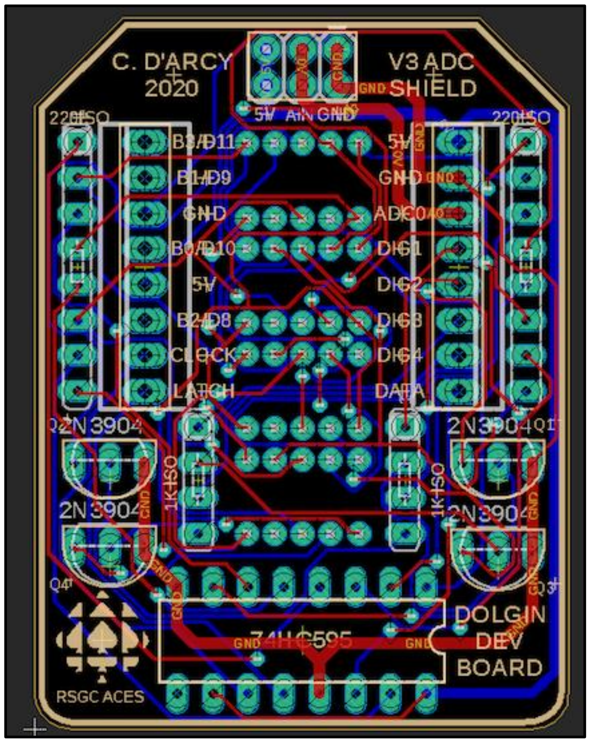
Project 21. ADC shield

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/DolginDevPlatform/index.html - adc

Purpose

ADC shield EAGLE board



The purpose of this project is to solder and develop code for the Analog to Digital Conversion (ADC) shield for the DDP. The board uses 4 NPN transistors to ground the CC pin of the now-obselete LA-301VL, a 2 mm pitch seven-segment display. Through the use of a shift register to control the segment pins of each display, and the transistors to only allow a single one to ground at a time, a 4 digit analog-read value can be displayed (0-1023). The analog input comes from the 2×3 female header found at the top of the shield, which offers 2 5V, Analog in, and GND headers respectively. This arrangement allows for the insertion of various devices such as a trimpot, TMP36 sensor, or even a voltage divider using a LDR and a fixed resistor.

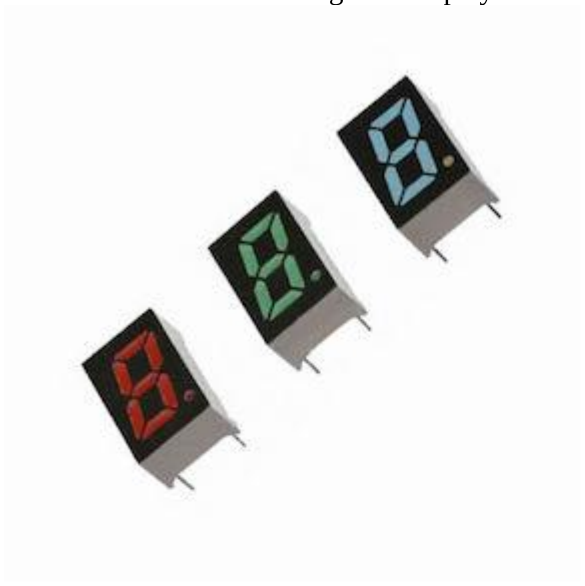
In order to create a POV scrolling effect across each display to show the analog read value, some (hopefully register level) code has to be developed to smooth the input data and display the ADC value on the displays, while inhibiting the display of leading zeros within the 4 digit number. In conclusion this project serves to refresh some through-hole soldering skills and develop further understanding of register level coding, and as an added bonus helping Mr.D'Arcy stock up on soldered shields for next year.

Procedure

Parts Table	
Component	Quantity
DDP	1
SN74HC595 IC	1
LA-301VL CC display	4
3904 NPN transistor	4
4 pin 1 KΩ ISO RN	2
8 pin 220 Ω ISO RN	2
2×3 Female header	1

Open the attached component bag and neatly lay out the parts. Start by inserting a single LA-301VL display, confirming its orientation using the provided EAGLE board sketch. Hold the display in and solder the leads on the side opposite to the next display to be added. Insert the second display and solder the remaining side of the first display. Repeat until all seven-segment displays are soldered in place. Solder the remaining components in place, beginning with the isolated

LA-301VL seven segment display



resistor networks (orientation does not matter), and continuing to the femal header, transistors (orientation does matter), 16 pin chip seat and finally the 1x8 male headers. Create a blank Arduino sketch entitiled "ADC_Shield" and save it. Using the EAGLE schematic and board, create pin definitions matching the hardware on the board to the pinout of the DDP. Create an array housing a series of 8 bit values that will create the numbers 0-9 on the displays. Begin by using high level code to display a single 0 onto the display, gradually working up into creating a POV 2 digit number. Using the previous high level code, develop register level code that employs a `getAverage()` function to average the previous 100 readings in order to smooth the data, a register level `shiftOut()` command that uses register level manipulation of the clock, data, and latch bits, and finally a `for loop` that cuts off the displayed leading zeros.

Code

```
//Project: ADC Shield code to inhibit the display leading zeros, using a variety of
register level commands.
//Author: Josh Dolgin
//Date: 20/02/12
//Status: Working
#define NUMDIGITS 4 //The number of digits of the number to be displayed
#define CLK  1<<PORTA7 //Shift register clock pin
#define LATCH 1<<PORTA6 //Shift register latch pin
#define DATA 1<<PORTA5 //Shift register data pin
#define THOUSANDS 1<<PORTA1 //Pin to turn on thousands digit
#define HUNDREDS 1<<PORTA2 //Pin to turn on hundreds digit
#define TENS 1<<PORTA3 //Pin to turn on tend digit
#define ONES 1<<PORTA4//Pin to turn on ones digit
#define AIN A0 //Analog input pin
#define NUMELEMENTS 100 //decide on number of averaged values
uint16_t history[NUMELEMENTS];
uint8_t numbers[] = {0b11111100, 0b01100000, 0b11011010, 0b11110010, 0b01100110,
0b10110110, 0b10111110, 0b11100000, 0b11111110, 0b11110110}; //QA of SR is DP, QB-QH
is GSEG - ASEG
uint8_t pins[] = {ONES, TENS, HUNDREDS, THOUSANDS}; //Array of pins to display the
digits

void setup() {
  DDRA = CLK | LATCH | DATA | ONES | TENS | HUNDREDS | THOUSANDS; //Declaring pins
for output
  PORTA = 0; //Making sure all pins are off
}

void shiftout(uint8_t order, uint8_t value) {
  uint8_t mask = order ? 0x80 : 0x01; // if non-zero (1-255), the mask changes based
off the order
  for (uint8_t i = 0; i < 8; i++) { // 8 times, setting each bit of data
    PORTA &= ~CLK; //Clock pin low
    if (value & mask)
      PORTA |= DATA; //If that bit is set, data pin high.
    else
      PORTA &= ~DATA;
    PORTA |= CLK; //Data gets shifted in on rising edge
    value = order ? value << 1 : value>>1; //Moving value into the mask
  }
}

uint16_t getAverage(uint16_t value) {
  uint32_t total = 0;
 for (uint8_t i = 0; i < NUMELEMENTS - 1; i++) {
    history[i + 1] = history[i];
    total += history[i];
  }
  history[0] = value;

  return (total + value) / NUMELEMENTS;
}

void loop() {
  uint16_t reading = getAverage(analogRead(AIN)); //Get a value
  for (uint8_t x = 0; x < NUMDIGITS; x++) {
    PORTA &= ~LATCH; //Latch low
    shiftout(1, numbers[reading % 10]); //1 is MSBFIRST for shiftOut
    PORTA |= LATCH; //Latch high
    PORTA |= pins[x]; //Ones bit first, counting up to thousands
    delay(2);
    PORTA &= ~pins[x]; //Turn off the bit
    Reading /= 10; //Moving to the next digit
    if (reading == 0)
      x = NUMDIGITS; //If there is nothing left to display,
  }
}
```

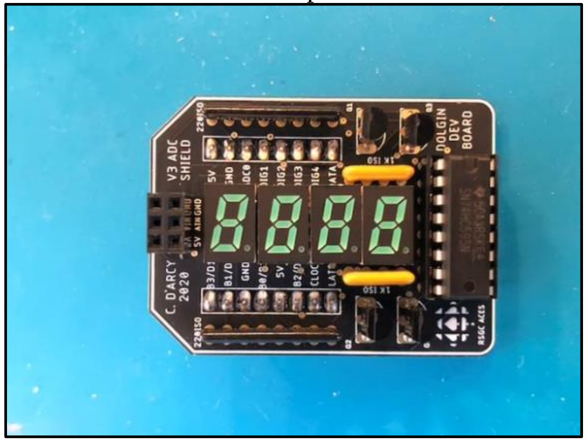
Reflection

In reflection I believe that this was a highly successful project for me. Once I was done soldering my board I immediately went into the EAGLE schematic and board to ensure that all my pin declarations were correct, because without the correct pins there was no chance my code would work. After beginning my ADC coding in high level code, I felt unfulfilled seeing my 100 lines of arithmetic and if statements to display the correct value. Instead I began to modularize and simplify my code to a compact register level display of what I have learned so far. Although I struggled initially with setting and clearing bits, eventually I got the hang of manipulating individual bits in the data direction and port registers. The end result is a piece of code I consider to be one of the favourite pieces I have written to date, a display of my code progression in second term. Overall it has been extremely interesting to see the effect of how pushing myself to understand influences my domain preferences. I started the year in the design corner of the triangle, and gradually have been making my way to the center through advanced hardware development in the CHUMP, and new software techniques for the DDP.

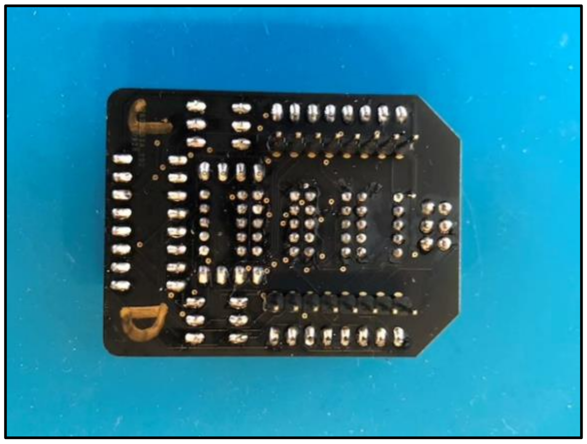
Media

https://www.youtube.com/watch?v=G9IsNfSlmPk

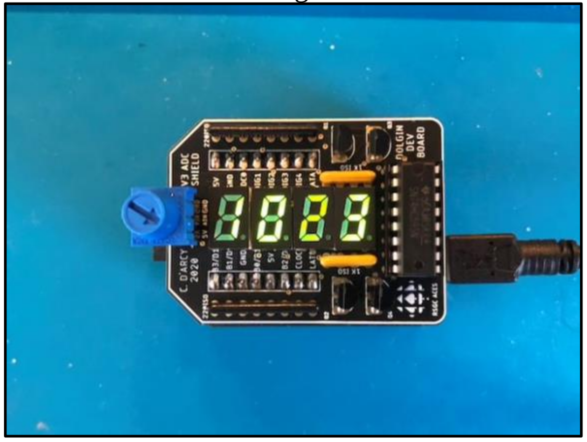
Soldered top of PCB



Soldered bottom of PCB



Working board



Project 22. Intersection Shield

Purpose

The purpose of the intersection shield is to mimic a city intersection, utilizing the same LA-301VL 2 mm pitch seven-segment displays as the previous ADC shield project to display the countdown at pedestrian crossings, and 5 mm flat top LEDs to show the traffic light signals. The project also serves as a reminder to an extremely useful 4xxx series through hole IC, the 4511 BCD seven-segment display driver.

The soldering and coding of the intersection shield helps ACES with their endeavours in beginning to implement register level commands to their high level code, as practice is helpful in completely understanding bitwise register manipulation to control the LEDs, NPN transistors, and 4511 IC.

Cylindrical LED

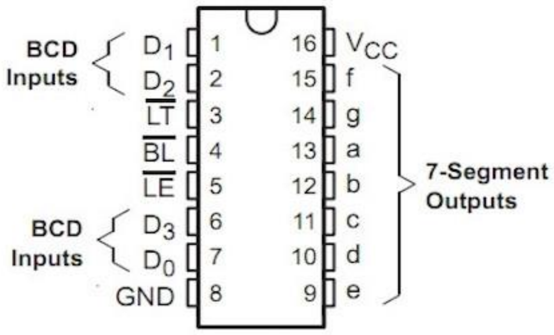
Digikey Part# 754-1304-ND

Theory

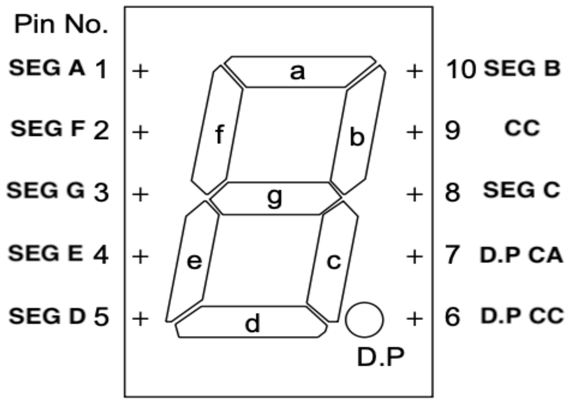
4511 seven-segment driver

The 4511 seven-segment driver is a CMOS logic IC which takes a Binary Coded Decimal input and outputs the decimal value of the input to a seven-segment display. Binary Coded decimal is the typical way in which a computer stores a decimal number to be displayed; and through the use of a 4-bit nibble (D_0-D_3) a number ranging from 0-15 can be placed on the input of the 4511. This value is output by the segment pins in the necessary order to display the decimal value(0-9) of the BCD input. For instance, if the BCD input was 0010 (decimal 2), the 4511 would output voltage to segments a, b, g, e, and d. A value of 1001 (decimal 9) would output voltage to the a, b, c, g, and f pins. Therefor, by using the NPN transistors to control which display is active, and counting down the BCD input from 7 to 0, the pedestrian countdown aspect of the project becomes fairly simple to code.

4511 pinout

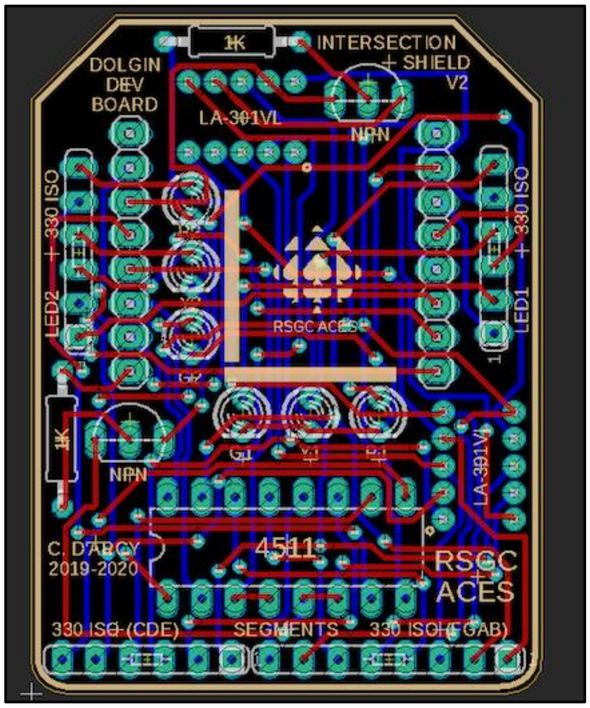


LA-301VL pinout



Intersection shield

Intersection shield PCB



To begin understanding the theory of the intersection shield, one must first open the provided EAGLE schematic and board to see how the pin layout corresponds to the DDP pin layout. In order to show the pedestrian countdown timer for a respective intersection, the CC pin of the seven-segment display must be tied to ground. The active display can be altered by applying power to the base pin to one of the two 3904 NPN transistors on the board, allowing the output of the 4511 to be displayed on the seven-segments. 3 of the 4 BCD inputs to the 4511 are connected to bits 0-2 of the DDP PORTB register (PB0-PB2), meaning the decimal numbers 0-7 can be shown.

Since the first 3 bits of the PORTB register are connected to the first 3 bits of input to the 4511, simply echoing whatever number one wants to display onto PORTB will output said number to the seven-segment display. The remaining 6 LEDs are placed into the PORTA register at bits 0-5, these LED's can be controlled simply by turning on the corresponding bit in the PORTA register. Ultimately, the goal is to code the following sequence of events; turn on the green LED for intersection 1 and the red LED of intersection 2, after a small delay begin counting down from 7 to 0 with 1 second increments on intersection 1's display, turn off intersection 1's green LED and turn on the yellow LED, after a short delay turn off the yellow LED and turn on the red LED of intersection 1, turn on the green light for intersection 2 and repeat the whole process, switching the intersection in which the commands are issued.

Procedure

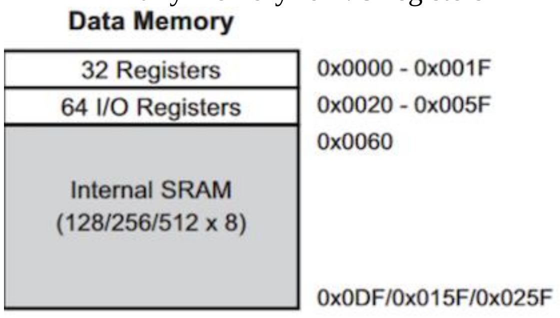
Parts Table	
Component	Quantity
DDP	1
4511 BCD decoder IC	1
LA-301VL CC display	2
3904 NPN transistor	2
6 pin 330 Ω ISO RN	3
8 pin 330 Ω ISO RN	1
1 KΩ fixed resistor	2
5mm flat top LED's	2 R, 2 G, 2 Y
1×8 male header	2

Open the bag containing all required parts including the PCB and lay them out neatly on the silicon soldering mat. Begin by soldering the tricky 2 mm pitch seven-segment displays, double checking their orientation against the provided image and EAGLE board. Solder the resistor networks and fixed resistors in place, all of which are non-polarized components. Solder in the NPN transistors, ensuring they are placed correctly by comparing the orientation of the part to the silk screening. Solder the 16 pin chip seat for the 4511 BCD decoder, matching the notch of the chip seat to the silk screening. Solder the flat top LED's into place, matching the flat side of the silk screening to the cathode leg of the LED. Finish by soldering the 1×8 male headers into place.

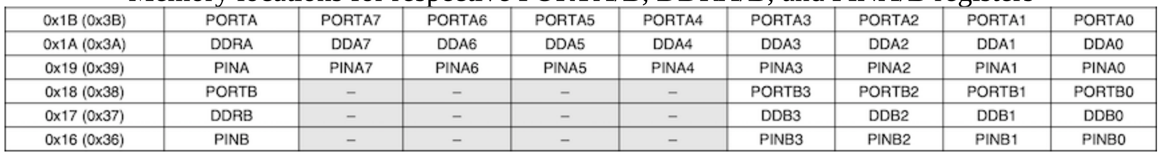
Operator	Operation
&	Bitwise AND
|	Bitwise OR
~	Ones compliment
<<	Bitshift left
>>	Bitshift Right
^	Exclusive OR (XOR)

Once the board is soldered, open the Arduino IDE and create a sketch entitled "Intersection.ino". Begin by stating the pin declarations, starting at PA0, going all the way to PA7, and PB0 to PB2. For ease of coding later, declare each value as 1<<PA_, as this simplifies the bitwise manipulation of the PORT registers later on. Continuing, there is two ways to declare a set of pins for output; the more complex way being using the sbi command (Set Bit in I/O register) to set all 11 bits in the 2 Data Direction Registers, but the more simple way being to use the |= (OR-equals compound assignment) to OR the current state of the Data Direction Register with the pins that will be declared for output. In this case the Data Direction Registers are not used for any other commands so the simple equals command would suffice, but it is good practice to only manipulate the necessary bits rather than carpet bomb the registers current bits. When manipulating Data Direction Register or PORT register bits, only really two compound assignments and one operator are used, those being |= , &=, and ~. As previously stated, OR-ing the current register value with whatever pin must be toggled will result in only a change of that respective pin. In order to clear a bit in the PORT or Data Direction register, the &= compound assignment ANDs the current register with the value passed to it. If the value being passed to the &= command is the inversion (~) of the bit to be turned off, all the other bits will be set to 1, resulting in the original register being mirrored, with the exception that the targeted bit is now off. Using this method of manipulating individual register bits, develop register level code that matches the provided reference video, playing with combining high level, regester level, and inline assembly commands into a single piece of code.

ATtiny memory for I/O registers



Memory locations for respective PORTA/B, DDRA/B, and PINA/B registers



0x1B (0x3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
0x1A (0x3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
0x19 (0x39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
0x18 (0x38)	PORTB	–	–	–	–	PORTB3	PORTB2	PORTB1	PORTB0
0x17 (0x37)	DDRB	–	–	–	–	DDB3	DDB2	DDB1	DDB0
0x16 (0x36)	PINB	–	–	–	–	PINB3	PINB2	PINB1	PINB0

Reference

https://mail.rsgc.on.ca/~cdarcy/Datasheets/ATtiny84Summary.pdf

http://darcy.rsgc.on.ca/ACES/TEI4M/DolginDevPlatform/index.html#intersection

Code

```
//Project: DDP Intersection shield
//Author: Josh Dolgin
//Date: 20/02/18
//Status: Working
#define B0 1<<PB0 //Binary Input 0 to the 4511
#define B1 1<<PB1 //Binary Input 1 to the 4511
#define B2 1<<PB2 //Binary Input 2 to the 4511

#define COUNT2 1<<PA7 //Base pin for segment 2
#define COUNT1 1<<PA6 //Base pin for segment 1
#define COUNTDOWN 7 //Starting number for countdown
#define NUMINTERSECTIONS 2 //Number of intersections

#define R2 1<<PA5 //Red for intersection 2
#define Y2 1<<PA4 //Yellow for intersection 2
#define G2 1<<PA3 //Green for intersection 2
#define R1 1<<PA2 //Red for intersection 1
#define Y1 1<<PA1 //Yellow for intersection 2
#define G1 1<<PA0 //Green for intersection 2
#define pause 1000 //Interval between changing the LED's

uint8_t segment[] = {COUNT1, COUNT2};

uint8_t pins[][NUMINTERSECTIONS] = {
  {G1, G2},
  {Y1, Y2},
  {R1, R2},
  {R2, R1},
};

void setup() {
  asm volatile  (
    "ser r16         \n" //Sets all bits in register 16
    "out 0x1A, r16   \n" //Sets I/O direction of PortA
    "in r16, 0x17    \n" //Read the current state of PORTB pins
    "ori r16, 0x07   \n" //Mask some for high
    "out 0x17, r16   \n" //Set new I/O states of PORTB
 ); //DDRA |= 0xFF, DDRB |= 0x07
}


void countDown() {
  for (int8_t i = COUNTDOWN; i > -1; i--) {
    PORTB |= i; // PB0 -> PB2 being used
    delay(1000); //always a 1 second delay each countdown
    PORTB &= ~i; //reverts PORTB
  }
}

void clearIntersection() {
  asm volatile  (
    "cbi 0x1B, 0    \n"
    "cbi 0x1B, 1    \n"
    "cbi 0x1B, 2    \n"
    "cbi 0x1B, 3    \n"
    "cbi 0x1B, 4    \n"
    "cbi 0x1B, 5    \n"
    "cbi 0x1B, 6    \n"
    "cbi 0x1B, 7    \n"
  ); // Clears PORTA, turns off all the LEDs
}

void loop() {
  for (uint8_t i = 0; i < NUMINTERSECTIONS; i++) {
    PORTA |= pins[3][i] | pins[0][i]; //Turns red on for opposite 
intersection and green on for respective intersection
    delay(pause * 2);
    PORTA |= segment[i]; // Turns on corresponding segment
    countDown();
    PORTA &= ~segment[i]; // turn off display
    delay(pause);
    PORTA &= ~pins[0][i]; //Turn off green light
    PORTA |= pins[1][i]; //Yellow
    delay(pause * 3);
    PORTA &= ~pins[1][i];
    PORTA |= pins[2][i]; //Red
    delay(pause * 3);
    clearIntersection();
  }
}
```

Reflection

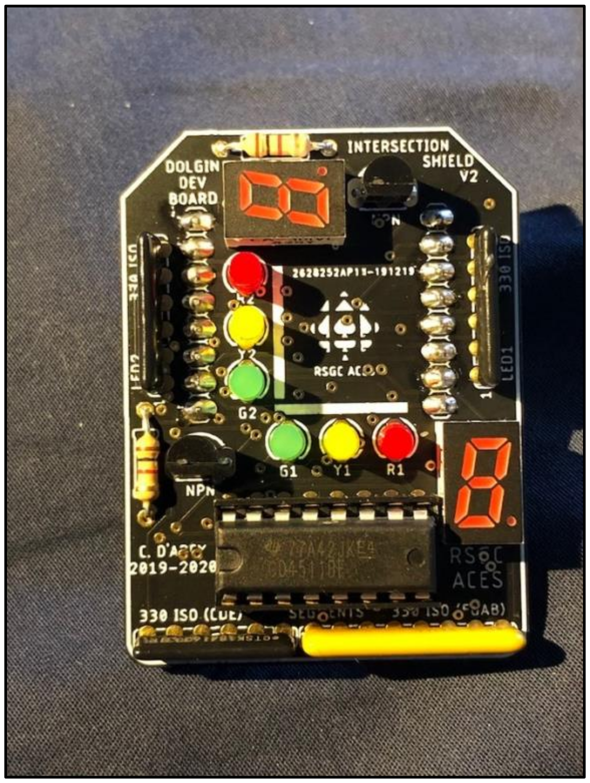
This was another highly successful project for me and my code development in register level, with the addition of basic inline assembly level commands. I began coding this project before handing in my ADC shield DER, while waiting in a coffee shop for my mom and sister to finish their gym class. Despite the bewildered expressions of those who paused to look at my computer screen, most likely pondering what form of gibberish I was writing to control what looks like a traffic light, I continued in my endeavours to arrive at a piece of code I am proud to have written. After some looking further into some more inline commands, I tried to experiment with the clear register (clr) and set register commands (ser), rather than setting/clearing a single bit at a time, however there maybe an issue with my syntax or use of the function because the new functions did not work the way I had hoped they would. Regardless, it has overall been a joy to code this shield, I had been planning it out in my head since Mr. D'Arcy showed me V1 at the end of last year, and I was able to showcase again how far I have progressed from high-level code similar to the if-else monstrosity that is ASCII & Buttons. My experience with these two shields has taught me that in order to greatly simplify the code, the pinouts of my own Legacy shield must be well thought out in a similar manner.

Media

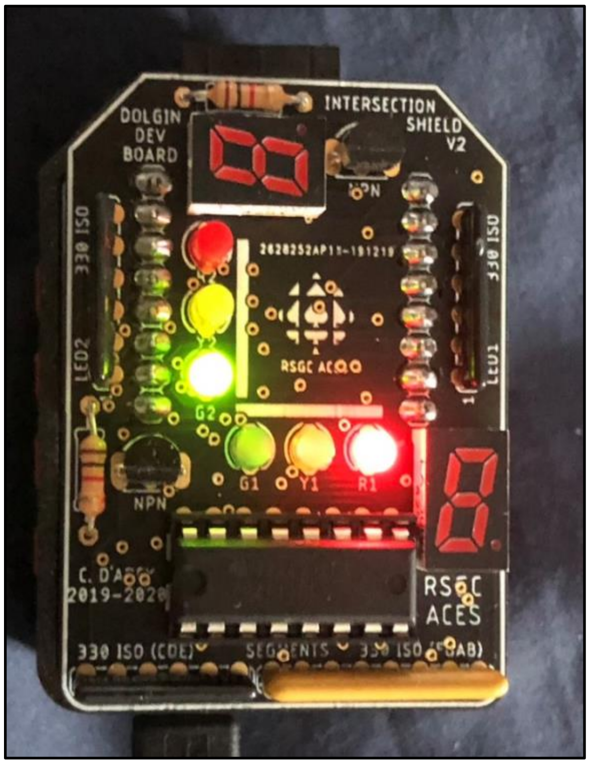
https://youtu.be/wIf4XwSSVHY

Soldered back of the board

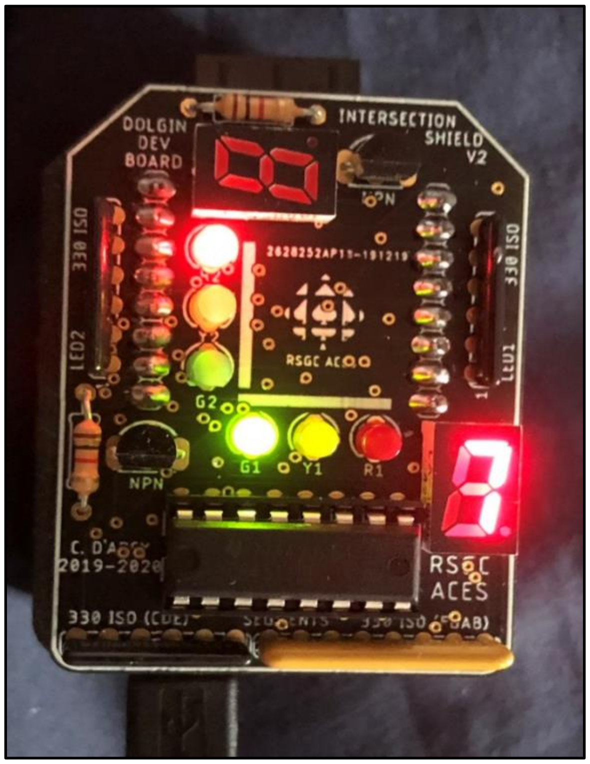
Soldered front of the board



Intersection 1



Intersection 2



Project 23. The SMD Desktop RTC-Equalizer

Purpose

Desktop Equalizer inspiration

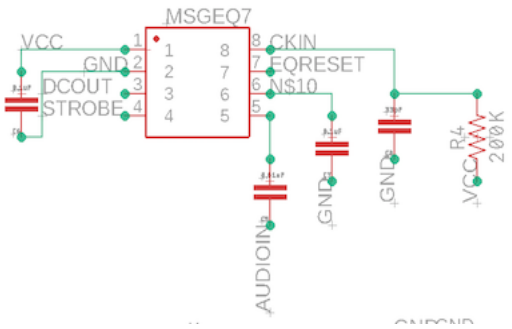
There is multiple purposes of the Dolgin RTC-Equalizer board, the primary of which being a full introduction to the surface mount soldering world, creating a thin and light board using as many SMD components as possible. Hopefully, the Dolgin Desktop RTC-Equalizer will become a piece of nostalgia for one to leave on their desk for future years and reminisce of the good times spent in the DES.

The board is filled with a multitude of components, beginning with the ATtiny84 mircocontroller, which one has become quite familiar with through a series of DDP projects. This microcontroller is responsible for taking input from two devices, a DS1307 RTC and a MSGEQ7 audio spectrum analyzer. The DS1307 Real Time Clock IC is used in the circuit for the purpose of providing the time to the viewer, displaying the new time for 10-15s every time a minute passes. The purpose of the MSGEQ7 in the circuit is to take audio input from the Adafruit microphone breakout board and pass these spectrum values through the ATtiny84 to the MAX7219 LED display driver, which will; in turn, display these values across the SMD 8×14 LED matrix. In totality, the PCB should display the 7 spectrum values in bars of 2 LED width, showing the new time after each minute passes.

Theory

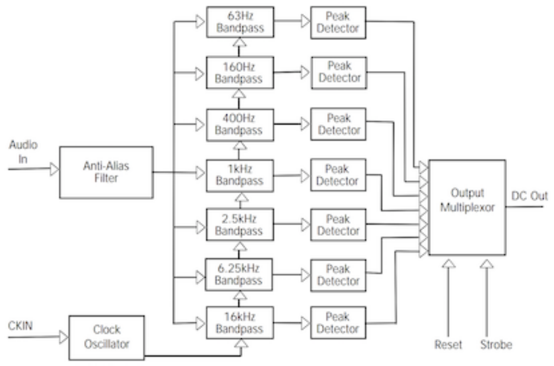
MSGEQ7

MSGEQ7 Schematic



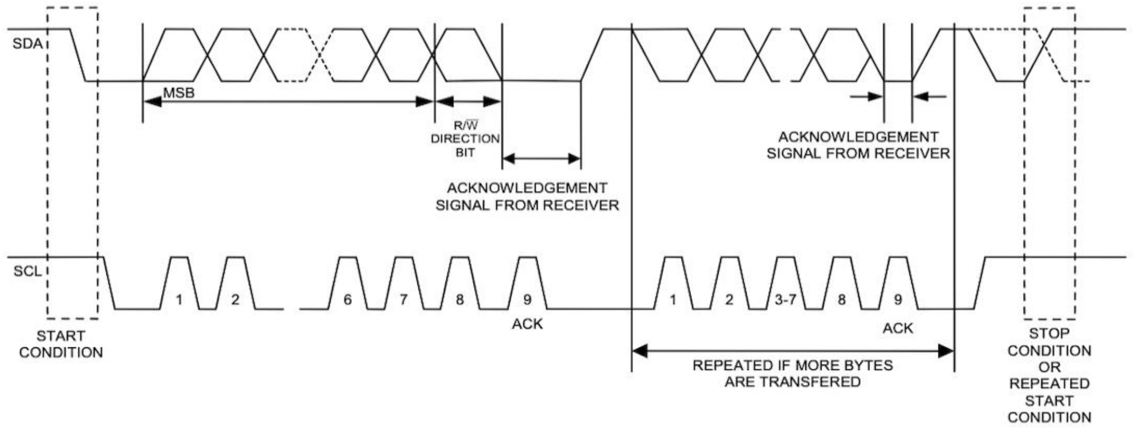
The MSGEQ7 is a seven-band spectrum equalizer IC, which takes a single audio in line, and through the manipulation of the reset and strobe pins, can output 7 analog values corresponding to the amount of a certain frequency there is. Firstly, the audio in line passes through an anti-alias filter into a set of seven band passes being fed by a clock oscillator (CKIN). The band filters range from values of 63 Hz to 16 KHz; for reference, humans typically hear sound in the 20 Hz to 20 KHz range. The DC voltage output is determined by which of the inputs to the multiplexor output are selected, which changes by toggling the reset pin high then low to reset to the 63Hz output, and with each pulse of the strobe pin the DC output is incremented by 1 to the next highest frequency. Therefore, the theory behind the use of the MSGEQ7 in this project is to take in audio from the Adafruit microphone breakout, and display a bargraph animation of each respective frequency from 63 Hz to 16 KHz.

MSGEQ7 internal circuitry



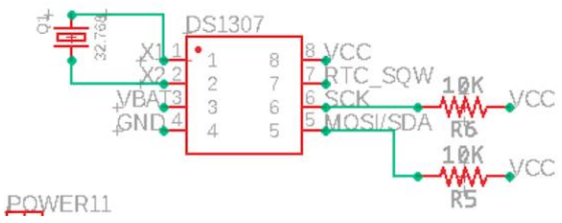
DS1307 RTC

I^{2}C Communication timing

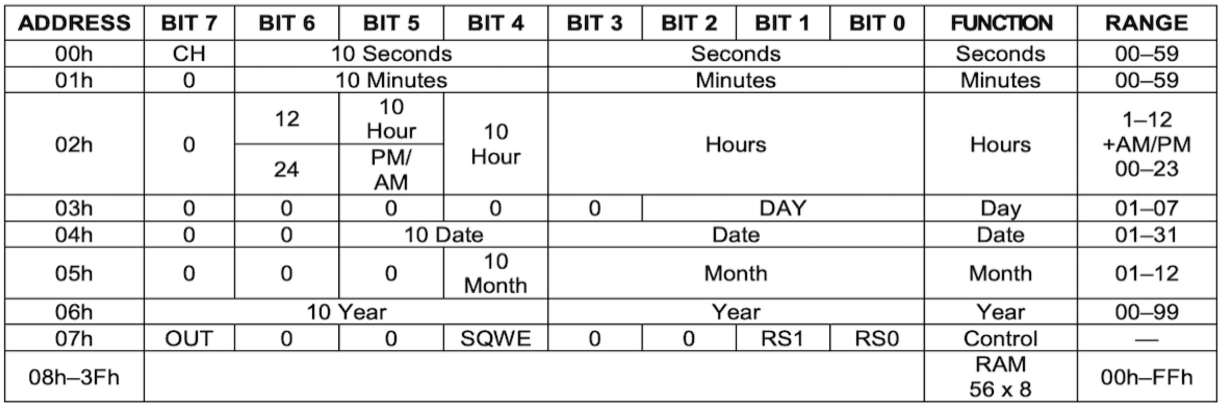

SDA
SCL
MSB
R/W̄ DIRECTION BIT
ACKNOWLEDGEMENT SIGNAL FROM RECEIVER
ACKNOWLEDGEMENT SIGNAL FROM RECEIVER
1
2
6
7
8
9
ACK
1
2
3-7
8
9
ACK
START CONDITION
REPEATED IF MORE BYTES ARE TRANSFERED
STOP CONDITION OR REPEATED START CONDITION

The DS1307 RTC is a 8 pin IC intended to keep the time in a circuit, providing year, month, week, hour, minute, or second data through an I^{2}C bus. I^{2}C is the method of two-wire communication, in which the SCL and SDA pins are used to communicate data between a designated device and the microcontroller. This form of communication is extremely useful, as over 100 devices can be connected to a microcontroller through only two lines, allowing for a project to include as many components as necessary. The I^{2}C bus works by the microcontroller sending an "address" value across the Serial Clock (SCK) line, awaiting for a single component which possesses that address to respond. Once there, one can send commands over the SCL lines, and receive the output data from the SDA line. The DS1307 serves the purpose of keeping the time within the circuit, letting the microcontroller know when a minute has passed, so that the new time can be displayed. The data arriving through the SDA lines must be parsed into usable decimal digits, since they arrive as Binary Coded Decimal (BCD). This BCD must be turned into individual nibbles, so that each nibble represents either a ones digit or tens digit of the requested time.

RTC Schematic



RTC Data registers

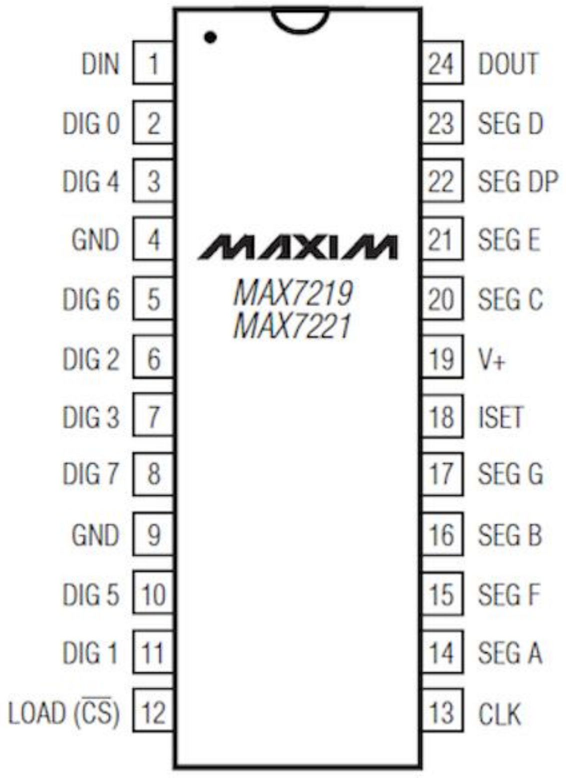


ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM
		24	PM/ AM							00–23
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

MAX7219

The MAX7219 is a specialized IC designed to drive a CC LED matrix, or multiple seven-segment displays at a single time. This is done by the current syncing digit pins, which will tie the CC of either a single digit or row of LED’s to ground at a time, allowing for a POV effect to be created. Once a single digit is grounded, the current sourcing segment pins will light up whatever necessary segments/LEDs to show the desired number/animation. The digit and segment pins are controlled by 3 input pins, LOAD, CLK and DIN.

MAX7219 Pinout



If more than 8 rows of LEDs must be lit, than the DOUT of the first MAX7219 must be connected to the DIN of the second; multiple IC’s can be connected together through this method, allowing for a display consisting of multiple matrices. In order to limit the current being synced through the digit pins, a resistor of value 10 KΩ - 64 KΩ (depending on maximum LED current) must be placed between the V+ and ISET pins. In order to control the 8×14 matrix (8 horizontal rows and 14 vertical columns), 2 MAX7219 IC’s are used in this project. Since 7 analog read values are stored in the MSGEQ7 array, each bar to display said value will be of width 2 and height 8, with 8 LED’s lit representing an Analog reading of around 900-1023. Unforunately the 8×14 matrix does pose some issues for the display of the time, only allowing numbers of 3 columns width to be displayed, unless the time is scrolled across the screen.

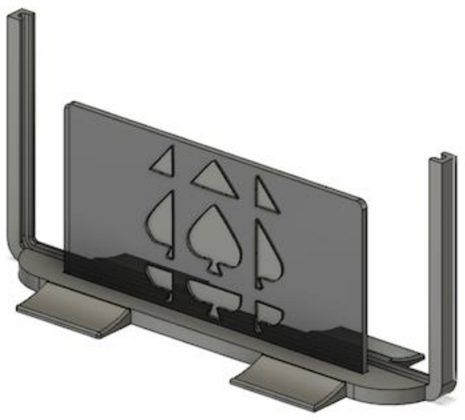
Procedure

Parts Table			
Part	Quantity	Package	Digikey Part #
2.2 KΩ Resistor	2	1206	311-2.2KERCT-ND
200 KΩ Resistor	1	1206	311-200KERCT-ND
64 KΩ Resistor	2	1206	311-60.4KFRCT-ND
10 KΩ Resistor	2	1206	311-10KERCT-ND
0.1 μF Capacitor	4	1206	399-C1206C104K5RAC7800CT-ND
0.01 μF Capacitor	1	1206	399-7174-1-ND
33 pF Capacitor	1	1206	399-1199-1-ND
10 μF Capacitor	2	1206	CAP CER 10UF 10V Y5V 1206
MAX7219 IC	2	24 SOIC	MAX7219EWG+TCT-ND
32 KHZ Crystal	1	8 mm × 3.8 mm	XC1195CT-ND
Green LED	42	1206	160-1169-1-ND
Amber LED	42	1206	160-2025-1-ND
Red LED	28	1206	160-1167-1-ND
DS1307	1	8 SOIC	DS1307ZN+T&RCT-ND
ATtiny84	1	14 SOIC	ATTINY84A-SSURCT-ND
USB AB connector	1		H125279CT-ND

CAD

Fusion 360

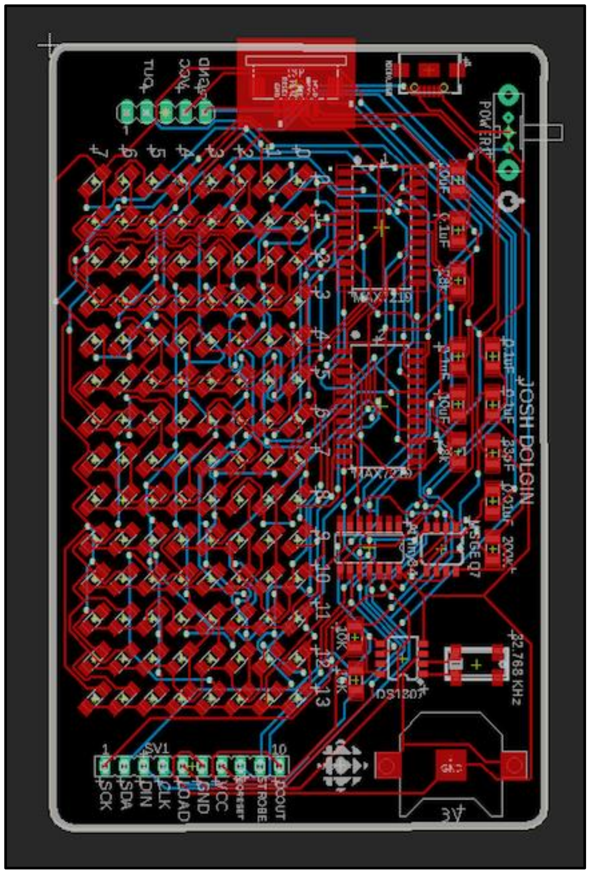
Case designed in Fusion 360



Open Fusion 360 and create a new sketch labelled "Spectrum Equalizer Case". Open the board DXF file exported from EAGLE, in order to design around the dimensions of the PCB. Begin by offsetting the outside dimension of the PCB by a value of 0.2 mm. Offset the newly generated line by 1.2 mm, allowing for a thin enough wall to be printed. Extrude this value symmetrically in both dimensions (1.2 mm walls + 0.2 mm offset + 0.8mm for board thickness). This way, walls 1.2 mm thick can be extruded to outwards to hold the board in place within the case. By following the previous steps, a simple shell to hold the board has been created, but now the case must stand up and accommodate for the LED diffuser. Extrude a rectangle 30 mm out from the lower part of the sleeve, and fillet the edges until the case looks aesteticly pleasing (around 5-10 mm). Finally, the curved legs are created by cutting part of small rectangular extrusions with a cylinder. The LED diffuser is simply a 1.0 mm thick rectangle designed to slot into the front of the case, and when printed in clear filament, diffuse the light from the LED's which allows for a smoother transition between each light. '

EAGLE

PCB Board



Open EAGLE and create a new sketch entitled "SMD_EqualizerV2". Begin by connecting the pre-labelled components (ISP header, Micro AB, etc.) to their respective busses. For example, connect each pin of the ISP header to the respective pins of the ATtiny84. Continue by adding the remaining THT and SMD parts from their respective libraries. Create an 8×14 LED matrix, in which each of the 14 columns is connected to the cathode of 8 LED's, and the 8 rows are connected to the anodes of 14 LED's. Connect the 2 MAX7219 LED drivers, feeding the DOUT of the 1st to the DIN of the second, and adding voltage smoothing capacitors on the 5V input. Add a 64 KΩ resistor between the ISET pin and the 5V pin of each MAX7219. Now connect the MAX7219's to the LED's, remembering the digit pins sync current and the segment pins source the current to the matrix. Wire the MSGEQ7 and RTC circuits, following the documentation and images in the previous sections. In order to ease the soldering aspect of this project, order a stencil along with the board. As applying solder paste for so many LED's can be tedious and cumbersome. Once the board and stencil arrive, use a solder squeegee to paste all of the necessary pads on the board. Place the components onto the board, using the silk screening for reference. Finally, place the board into the hodgson reflow oven on the reflow setting. Upon removing the board, ensure no connections are bridged.

Code

```
//Project : Surface mount desktop equalizer-RTC circuit, will show equalizer bars,
and show the time every minute
//Author  : Josh Dolgin
//Device  : Off-board Tiny84
//Date    : 20/02/29
//Status  : Not Working
//MCU     : ATtiny84
//Notes   : RTC circuit is conrfirmed to work, Equalizer circuit does not work

#include <TinyWireM.h>
#include <USI_TWI_Master.h>
#include <TinyRTClib.h> //Including necessary I2C libraries for RTC communication
RTC_DS1307 rtc;
DateTime dt = rtc.now(); //Date time structure housing all the necessary values

#include <LedControl.h> //Library to control the max7219
#define NUMDISPLAYS 2
#define DIN PA2
```

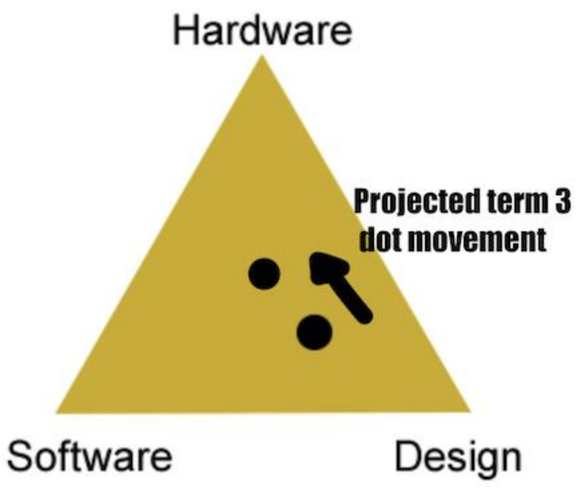
```
#define CLK PA3
#define LOAD PA1
LedControl lc = LedControl(DIN, CLK, LOAD, NUMDISPLAYS);

#define AIN PA7
#define STROBE 1<<PB2
#define RESET 1<<PB3
uint8_t spectrumValue[7]; // Holds ADC equalizer values for 7 frequencies
uint8_t colData [] = {0x01, 0x03, 0x07, 0x0F, 0x1F, 0x3F, 0x7F, 0xFF}; // How many
LEDs are lit(1-8)

uint8_t numbers[][3] { //numbers three columns wide and 6 rows tall
  {0x7C, 0x44, 0x7C}, //0
  {0x24, 0x7C, 0x04}, //1
  {0x5C, 0x54, 0x74}, //2
  {0x54, 0x54, 0x7C}, //3
  {0x70, 0x10, 0x7C}, //4
  {0x74, 0x54, 0x5C}, //5
  {0x7C, 0x54, 0x5C}, //6
  {0x40, 0x40, 0x7C}, //7
  {0x7C, 0x54, 0x7C}, //8
  {0x70, 0x50, 0x7C}, //9
};

void setup()
{
  rtc.begin();
  rtc.adjust(DateTime(2020, 2, 29, 11, 14)); //Sets the new time
  for (uint8_t i = 0; i < NUMDISPLAYS; i++) {
    lc.shutdown(i, false);
    lc.setIntensity(i, 8);
    lc.clearDisplay(0);// clear screen
  }
  DDRB |= (STROBE | RESET); //Sets strobe and reset for output
  PORTB &= ~RESET;
  PORTB |= STROBE;
}

void getValues() {
  PORTB |= RESET; //Reset high
  PORTB &= ~RESET; //Reset low, resets to the 63Hz equalizer value
  for (uint8_t i = 0; i < 7; i++)
  {
    PORTB &= ~STROBE;
    delayMicroseconds(30); // to allow the output to settle
    spectrumValue[i] = analogRead(AIN);
    PORTB |= STROBE; //Next frequency
  }
}

void displayValues() {
  getValues(); //get Equalizer values
  for (uint8_t i = 0; i < 2; i++) {
    lc.setRow(0, i, colData[(spectrumValue[0] >> 7)]);
    lc.setRow(0, i + 2, colData[(spectrumValue[1] >> 7)]);
    lc.setRow(0, i + 4, colData[(spectrumValue[2] >> 7)]);
    lc.setRow(0, i + 6, colData[(spectrumValue[3] >> 7)]);
    lc.setRow(1, i, colData[(spectrumValue[4] >> 7)]);
    lc.setRow(1, i + 2, colData[(spectrumValue[5] >> 7)]);
    lc.setRow(1, i + 4, colData[(spectrumValue[6] >> 7)]);
  }
}
```

```
void displayTime() {
  uint8_t hourValue = dt.hour();
  uint8_t minuteTens = dt.minute() / 10;
  uint8_t minuteOnes = dt.minute() % 10;
  for(uint8_t i = 0; i<3; i++){
    lc.setRow(0, i, numbers[hourValue][i]);
    lc.setRow(0, i+5, numbers[minuteTens][i]);
    lc.setRow(1, i+1, numbers[minuteOnes][i]);
  } //Unable to display double digit hour values with 8x14 matrix, currently 
developing scrolling code
}

void loop()
{
  uint8_t oldMin;
  uint8_t newMin;
  newMin = dt.minute(); //New minute value
  if (newMin == oldMin) {
    displayValues(); //If the minute value has not changed, display equalizer values
  } else {
    displayTime(); //Else display the time (the minute has changed)
  }
  oldMin = dt.minute(); //Old minute value
}
```

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/1819/ISPs.html
https://github.com/adafruit/TinyRTCLib
https://github.com/adafruit/TinyWireM

Reflection

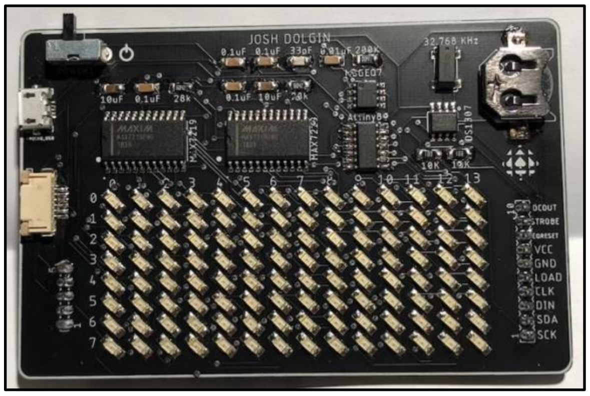


In reflection this project was not as successful as I would have liked it to be, and I am fairly frustrated with the current outcome. As much as I enjoy surface mount soldering as a concept, I felt sort of bored during the entire medium ISP process. It was practically building a prototype, sending out 2 PCB's and waiting. Beyond designing a case in fusion and implementing register level code into my original high level code, there was nothing else to do but sit and wait for the next iteration of my board to come. Although my idea was interesting and subjectively one of the better implemented designs, I may have reached a little to high on this one. Overall I found mild success in the parts of my board that did work individually, and not having to mess with the test clip to program my V2 board was a bit of a relief. Although my design expertise was not pushed very far in this project (an hour of fusion?), I am happy to have further progressed towards the center of the domain triangle, continuing hardware and software development that last year I would have thought to be impossible. Overall I am happy to have pursued as ambitious a project as I did, because I gained valuable knowledge about the SMD process that will prove valuable in the design of my flex PCB.

Media

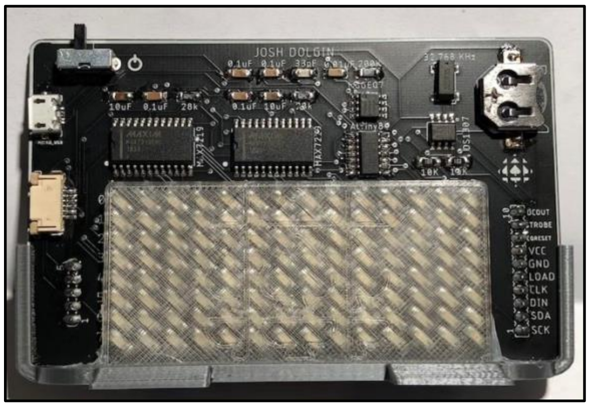
https://www.youtube.com/watch?v=eeAYpyfAd30

Soldered front of PCB

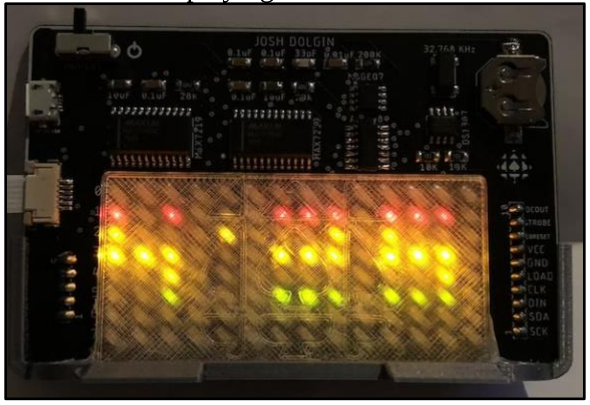


Soldered back of PCB

PCB encased



PCB displaying random time value



Project 24. Bicolor Byte

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/1920/Tasks.html - bicolorByte
https://www.microchip.com/mplab/avr-support/atmel-studio-7
https://learn.adafruit.com/usbtinyisp/drivers

Purpose

Well to start off, the purpose of this project is to re-establish some normalcy to the ACES classroom and continue developing projects during these interesting times. Although some supplies are limited due to a great deal of delays in the distribution world, ACES continue to push boundaries. This project is the first introduction to Atmel Studio 7, an AVR assembly compiler (assembler) used to program a great variety of standard microcontrollers. One huge benefit of using Atmel Studio 7 is the multiple available viewing windows while in the "debugging" state of the application. These viewing windows actively show the states of all I/O registers, Data and Storage RAM registers, and much more to analyze what is functioning in the program.

The immediate purpose of this project is to reflect a random byte bit by bit on a bicolor LED, with a green light reflecting a set bit and red lit representing a cleared bit. Each bit will be represented for 1 second, and once all bits have been displayed, there will be a 3 second pause before restarting the series over again. In order to accomplish this task, ACES must become fully acclimated to coding in only AVR assembly language, rather than just embedding some assembly into high level instructions. This means that some further research of AVR commands beyond simple `sbi` (set bit in I/O) register, `cbi` (clear bit in I/O register), or `dec` (decrement) commands must be implemented. AVR assembly language, although more complicated in synthax than standard Arduino C and other languages is extremely powerful in the amount of RAM taken up by individual instructions, as well as the total time to execute a program.

Code

```
;PROJECT    :Bicolor Byte
;PURPOSE    :Display a random byte value bit by bit on a bicolor LED
;AUTHOR     :J. Dolgin
;DATE       :2020 04 10
;DEVICE     :Dolgin Development Platform
;MCU        :ATtiny84
;COURSE     :ICS4U
;STATUS     :Working
;.include   "prescalars.inc"     ;assembly directive equivalent to compiler
directive #include
.def        mask    = r16        ;using register 16 to store the mask value
.def        byte    = r17        ;using register 17 to store the mask value
.equ        DDR     = DDRA       ;typically, we'll need the use of PortA
.equ        PORT    = PORTA      ;both its data direction and output register
; DATA Segment declarations
.dseg                            ;locate for Data Segment (SRAM) requirements
(default start at 0x0060)
data:       .BYTE 1              ;reserve one byte for a variable (the label is the
symbol)
; CODE Segment (default)
.cseg                            ;locate for Code Segment (FLASH)
; ***** START OF CODE ********************************************************
.org        0x0100               ;well clear of IVT
reset:                           ;PC jumps to here (start of code) on reset
interrupt...
setup:
             sbi DDR, 0          ;set both bits in the DDR register, to output
             sbi DDR, 1          ;
             ldi mask, 0x80      ;sets a mask in the most significant bit

loop:
             ldi byte, 0xCC      ;any random byte to be displayed
             and byte, mask      ;reflect the state of that respective bit
             breq red            ;if the result was a zero, show red
             rjmp green          ;if the bit was set, show green

green:
             cbi PORT, 0         ;set and clear respective bits to show green
             sbi PORT, 1         ;
             ldi  r18, 41        ;delay 1s
             ldi  r19, 150       ;
             ldi  r20, 128       ;
L1:          dec  r20            ;
             brne L1             ;
             dec  r19            ;
             brne L1             ;
             dec  r18            ;
             brne L1             ;
             rjmp shift          ;

red:
             cbi PORT, 1         ;clear and set respective bits to show red
             sbi PORT, 0         ;
             ldi  r18, 41        ;delay 1s
             ldi  r19, 150       ;
             ldi  r20, 128       ;
L2:          dec  r20            ;
             brne L2             ;
             dec  r19            ;
             brne L2             ;
             dec  r18            ;
```

```
            brne L2             ;
            rjmp shift          ;

shift:
            LSR mask            ;shift mask right one
            breq done           ;if mask is cleared, go to done
            rjmp loop           ;if mask isn't cleared go back

done:
            cbi PORT, 0         ;turn off LED
            cbi PORT, 1         ;
            ldi  r18, 122       ;delay 3s
            ldi  r19, 193       ;
            ldi  r20, 130       ;
L3:         dec  r20            ;
            brne L3             ;
            dec  r19            ;
            brne L3             ;
            dec  r18            ;
            brne L3             ;
            rjmp PC+1           ;
            rjmp setup          ;go set up the mask again
```

Procedure

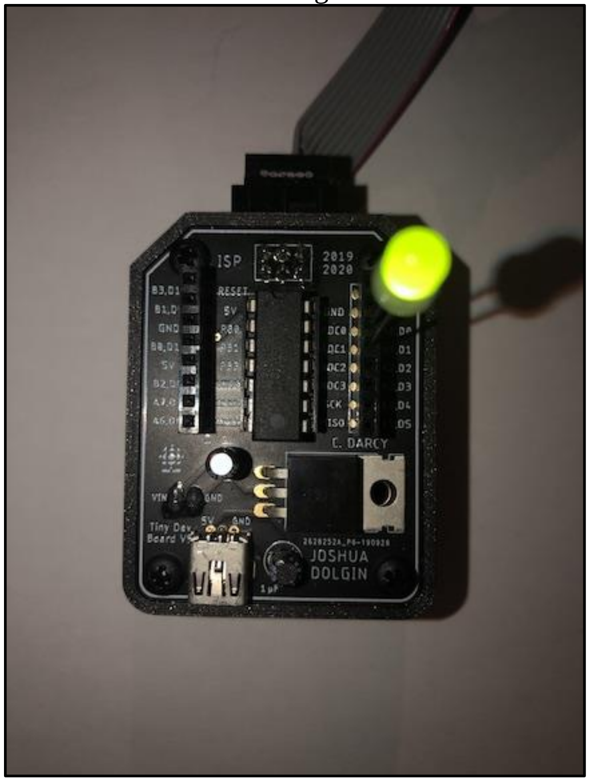
Parts Table	
Component	Quantity
DDP	1
Bicolor LED	1
Sparkfun programmer	1

Download and install Atmel Studio 7 on a windows computer. Open the previously made code shell for the ATtiny84, naming the new project "Bicolor Byte". Fill in the upper comments of the code shell, stating the new date, status, and any other required notes. Download and install the required drivers for the USBTiny pocket programmer, following the instructions posted on Adafruits website. Under the "external tools" menu, set up a tool for programming the ATtiny84, using the avrdude.exe file within the Arduino application. Begin the code by using the `sbi` command to turn on PA0 and PA1 in the DDRA register to output. Place a 0x80 in the mask register using the `ldi` command. Load the byte to be displayed into a new register entitled data. AND the byte with the mask value, mirroring the bit in the position designated by the mask. If the resulting byte is equal to 0, the bit was not set, so use the branch if equal to zero command to display a red light. If the result was not 0, the bit must be set, so display a green light. Create functions to display either the green or red LED, followed by a 1 second delay. The only real difference is which bit is set and cleared in the port register. After the function, use the relative jump command to jump to a new function, which shifts the mask to the right once. After shifting the mask using the logical shift right command, check if the result was a 0, meaning that all 8 bits have been displayed already; branch to the done function, which delays for 3 seconds. If the mask has not been fully shifted, go to the top and do the whole process again. This is essentially a for loop that runs 8 times, as the mask moves across each bit position. Finally, in the done command, after pausing for 3 seconds, go all the way to the top and reestablish the mask, beginning the process again.

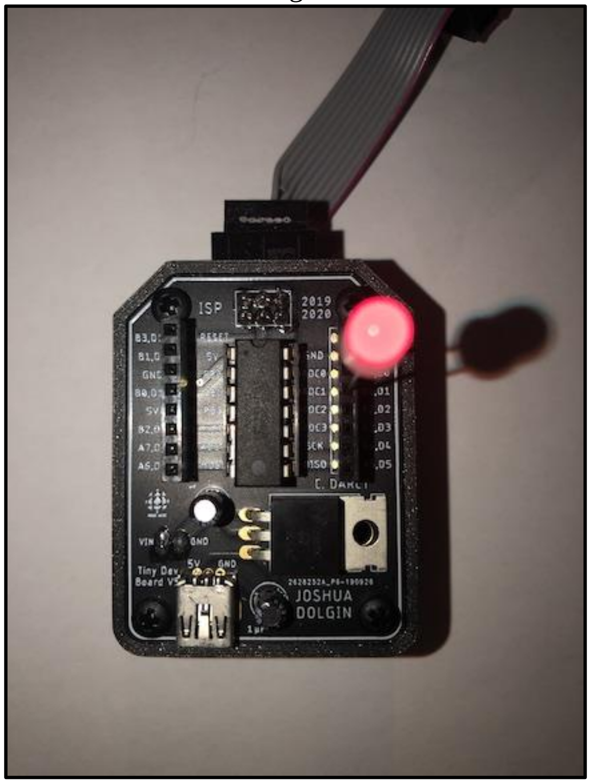
Media

https://www.youtube.com/watch?v=YnQJfR6WtqY

DDP showing a set bit



DDP showing a cleared bit



Reflection

Upon reflection I believe this was a farily successful first project while on quarantine (I'm sure there is much more to come). When Mr. D'Arcy contacted me over March Break to install Atmel Studio 7 on my laptop over the break, I was happy to oblige some of my time to helping better the instructions my peers would receive over the coming weeks. After some fuss with installing windows software onto my device, and more time struggling with installing the necessary device software to upload through the sparkfun pocket programmer, I was off to the races. In our final class before the break, Mr. D'Arcy said something really interesting which caught my attention; he told my class that if they were to work in the Arduino IDE, they should use paper to keep track of data registers. Of course AS7 provides the ability to do this through the debug software, but the idea that stuck was using paper to plan out ideas. Instead of just starting code at the top, and building the next line of code based off what came before; I wrote a plan for my code on the paper, with each function written down with a shell, so I could see what needed to happen for my code to work properly. Now, my each line was part of a greater complex of both previous instructions and the ones to come. Overall, I have reached a point where I am comfortable with setting up conditional instructions with branches, but still have issues with data value manipulation.

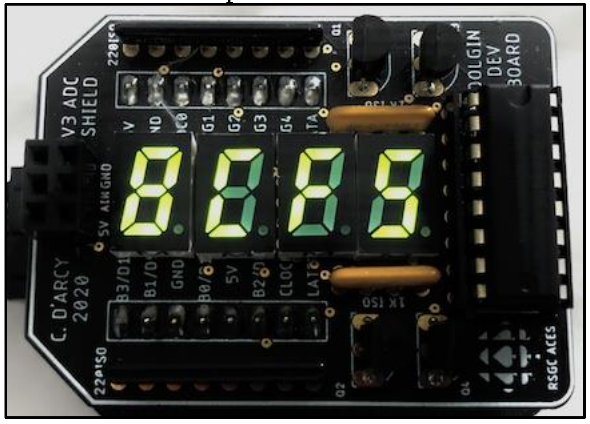
Project 25. PoV on the ADC Shield

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/DolginDevPlatform/index.html#PoV

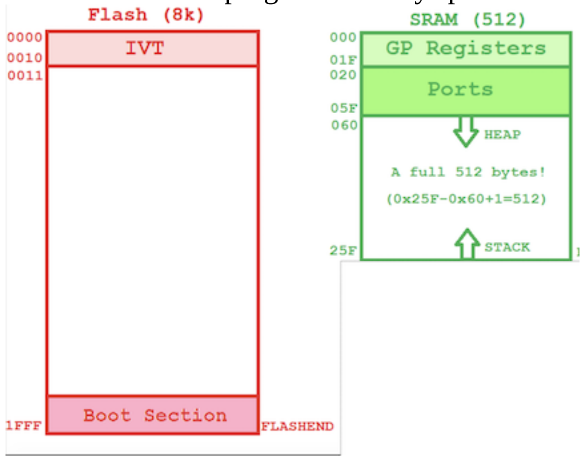
Purpose

PoV example on the ADC shield



The purpose of this project is to display a 2 byte hexadecimal word on the ADC shields 4 seven segment displays, using both shiftout and POV techniques. Although a project of similar caliber has already been completed this year (see Project 21. ADC shield), the previous iteration is coded in Arduino register-level code, while this project is coded in AVR assembly. Although it seems like this project would be one of the more difficult coding ventures of the year, if one did the incremental homework assignments of displaying the numbers 0-F on a single display, the jump to this final version is not as difficult.

SRAM and program memory space



In this project, *persistence of vision* is the act of turning on one digit at a time, shifting out the value, then turning it off and moving to the next digit. This is accomplished by grounding the base pin of a single NPN transistor at a time, allowing the CC of the seven segment display a path to ground. Also, this project serves to further develop knowledge of data storage and access in the ASM assembly environment. Both the 2 byte constant and array of shiftout values are stored in program memory, which are placed in individual addresses when the code is uploaded to the microcontroller. The contents of program memory can be accessed through the `lpm` command, which loads the contents of program memory at an address pointed to by the "Z" registers into a designated general purpose register. Once the 4 nibbles of data to be displayed are separated, since they can't be placed in program memory on upload, must be stored to SRAM instead. To store a piece of data to SRAM, the `sts` command will store the contents of a register to a specified SRAM address.

Code

```
;PROJECT    :Display 4 bit value on ADC shield
;AUTHOR     :Josh D
;DATE       :2020 05 01
;DEVICE     :Dolgin Development Platform
;MCU        :ATtiny84
;COURSE     :ICS4U
;STATUS     :working
;.include   "prescalars.inc"    ;assembly directive equivalent to compiler directive
#include

.def         util   = r16         ;general use register
.def         storage= r17         ;holds the value to be displayed
.def         mask   = r18         ;register holding the mask value
.def         use    = r19         ;general purpose register

.equ         DDR    = DDRA        ;typically, we'll need the use of PortA
.equ         PORT   = PORTA       ;both its DDR and output register
.equ         thou   = PA1         ;these are the port pins...
.equ         hund   = PA2         ;connected to the base pins of
.equ         tens   = PA3         ;each of the transistors that
.equ         units  = PA4         ;ground the respective displays
.equ         DATA   = PA5         ;595 data pin
.equ         LATCH  = PA6         ;595 latch pin
.equ         CLK    = PA7         ;595 clock pin

.dseg
digits: .byte 4                   ;reserve 4 bytes to hold digits

.cseg                             ;locate for Code Segment (FLASH)
; ***** START OF CODE ********************************************************
.org 0x0000                       ;start of Interrupt Vector Table (IVT)
      rjmp    reset                ;lowest interrupt address == highest priority!

.org        0x0011                ;well clear of IVT
constart:
.DW 0x8CF5                        ;2 byte word holding value
constend:
varStart:
.DB    0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xF6, 0xEE, 0x3E, 0x9C,
0x7A, 0x9E, 0x8E                  ;shiftout data
varEnd:

.org         0x0100
reset:
      ldi util, 0xFF              ;load 255 into utility register
      out DDR, util               ;sets all bit in DDRA register for output
      ldi ZL, low(constart<<1)    ;loads low byte of Z reg the start of the word
      ldi ZH, high(constart<<1)   ;loads high byte of Z reg with the start of word
      lpm util, Z                 ;loads first byte of the word
      ldi mask, 0x0F              ;masks off high nibble
      and util, mask              ;and together
      sts digits, util            ;store first hexvalue to be displayed
      lpm util, Z+                ;load low byte again, post incrementing Z
      rcall parse                 ;parse off high nibble
      sts digits+1, util          ;store result to second address in SRAM
      lpm util, Z                 ;load high byte of word
      and util, mask              ;mask off high nibble
      sts digits+2, util          ;store result to SRAM (3rd value)
      lpm util, Z                 ;one last time load high word
      rcall parse                 ;parse off the high nibble
      sts digits+3, util          ;store final result to SRAM
      rjmp main                  ;jump to main function to display values
parse:
      ldi storage, 0x04          ;runs 4 times
again: lsr util                  ;shift right
      dec storage                ;decrement storage register
      brne again                 ;if not equal to zero shift again
      ret
main:
      ldi use, 1<<units          ;where to start displaying
      ldi mask, 0x80             ;load mask for shiftout function
      lds storage, digits        ;load the first digit from SRAM
      rcall execute              ;do what you need to
      lds storage, digits+1      ;load the second digit from SRAM
      rcall execute              ;do what you need to
      lds storage, digits+2      ;load the third digit from SRAM
      rcall execute              ;do what you need to
      lds storage, digits+3      ;load the fourth digit from SRAM
      rcall execute              ;do what yout need to
      rjmp main                  ;go to the main function
execute:
      ldi ZL, low(varStart<<1)   ;only need to use the low part of Z reg
      add ZL, storage            ;add contents of SRAM to Z reg
      rcall shift                ;go shiftout the data
      in storage, PORT           ;load the port byte into storage reg
      add storage, use           ;add the contents of use mask to storage
      out PORT, storage          ;output this back to the PORT reg
      rcall delay                ;call the 20ms delay
      sub storage, use           ;subtract the mask
      out PORT, storage          ;change the port output
      lsr use                    ;shift right use register
      ret                        ;
shift:
      cbi  PORT, CLK             ;clear clock bit to pulse in the next data
      lpm     util, Z            ;load util the contents of address Z in prog mem
      and  util, mask            ;and the data with the mask
      breq clearBit              ;if the result was 0, clear the data bit
      rjmp setBit                ;else set the data bit
clearBit:
      cbi PORT, DATA             ;clear the data pin
      rjmp pulse                 ;pulse the clock pin
setBit:
      sbi PORT, DATA             ;set the data pin
      rjmp pulse                 ;pulse the clock pin
pulse:
      sbi PORT, CLK              ;data is shifted in on the rising edge
      LSR    mask                ;shift the mask right once
      brne shift                 ;if the mask is not equal to 0, go shift the next
      sbi PORT, LATCH            ;set the latch high to display
      cbi     PORT, LATCH        ;latch low until all 8 bits are loaded
      ldi mask, 0x80             ;reload the mask
      ret                        ;
delay:
      ldi  r21, 53               ;
L1:   dec  r21                   ;
      brne L1                    ;
      nop                        ;
      ret                        ;
```

Procedure

AVR Instructions	
Instruction	Syntax
Load immediate	`Ldi Rd, k`
Out	`Out P, Rr`
Load memory	`Lpm Rd, Z`
AND	`And Rd, Rr`
Store to SRAM	`Sts k, Rr`
Relative call	`Rcall k`
Relative jump	`Rjmp k`
Logical shift right	`Lsr Rd`
Decrement	`Dec Rd`
Branch if not equal	`Brne k`
In	`In Rd, P`
Add	`Add Rd, Rr`
Subtract	`Sub Rd, Rr`
Set bit in I/O	`Sbi P, b`
Clear bit in I/O	`Cbi P, b`

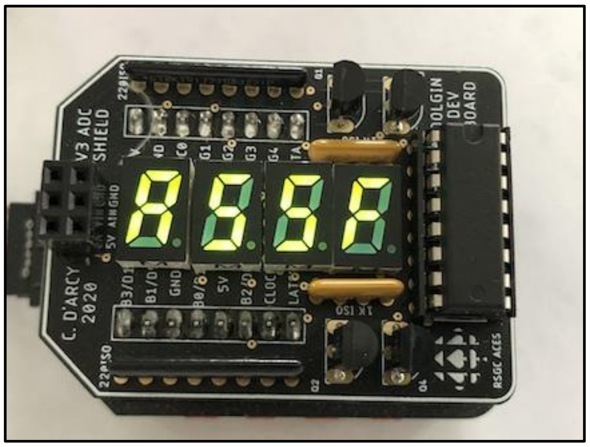
Open Atmel Studio 7 and create a new project based off the ATtiny84 template previously created. Add the necessary comments and descriptions to the document. Begin by defining 4 registers, 3 for general purpose data manipulation and 1 to house the mask values. Define labels for each of the respective PORTA values used such as Clock, Data, Latch, and the digit pins. In the data segment section, reserve 4 bytes in SRAM to hold the separated digits of the value. In the code segment, begin by creating a word constant in program memory with the `.DW` command. Create an array in program memory housing the bytes necessary to display respective numbers on the seven segment displays. Begin by setting all bits in the DDRA to output using the load immediate and out command. Load the first byte of the word from program memory using the `LPM` command. Separate the low nibble using a mask, and the high nibble by shifting the byte right 4 times. Do the same for the high byte of the word. Save each of the digits to one of the addresses saved previously in SRAM. In the main function, load the mask value into the mask register (MSBFIRST). Load the first digit to be displayed from SRAM into a general purpose register. Add the contents of this register to the low Z register, which will now point to the necessary byte in the array to show the number. Call the shiftout command previously written to display the number before turning on the digit, pausing for 20 ms and turning the digit off again. Move to the next digit and follow the same process as before.

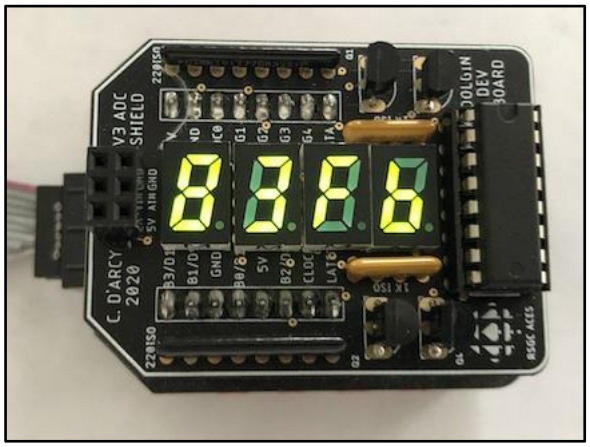
Reflection

There is not a whole lot to say in this reflection that I haven’t said already, I believe that this was another successful project in AVR Assembly. Although some of my classmates are seeming to still struggle with the concepts (4 calls today?), I was still happy to provide support despite the various amounts of resources at their disposal. I have decided that it is better to give hints rather than direct answers (I mean, I won’t always be here to help right?), which I hope will help my classmates develop the skills necessary to complete the project themselves; even without pseudocode. I stayed up to date with the homework and learning what was asked of us by Mr.D’Arcy, as it is only fair to reciprocate the effort he puts in for each class by extending myself to understand what he is teaching. I’m sure there is a more compact, less confusing way to complete the project than I have, but it works for me and I am content with the result.

Media

https://www.youtube.com/watch?v=iqntm2WK-sU





Project 26. Scrolling DER page

Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/1920/ISPs.html - logs

Purpose

Scrolling message board example

The purpose of this project is to develop a mini, flexible, SMD version of the DES scrolling message board to paste onto the front page of a Design & Engineering report, that would scroll an ACES name across the front page. Since the PCB will be on the front page of a DER, it will run off of solar power, so that whenever the laminated cover is in the sun it will scroll the information to whomever it is being shown to. Not only will this be useful and impressive to future employers, but it will serve as a piece of nostalgia for the days spent in the DES. Secondly, the project serves to finalize the progress made in both the Hardware and Software domains of Engineering.

The board is mostly populated with the 64 LED's in the 8×8 matrix, but also includes a MAX7219 LED driver with accompanying current limiting resistor and power regulation capacitors. The MCU used in the project is the all-too familiar ATtiny84, which for the most part has been used in every project of the ICS4U school year.

Code

```
//Project : Scrolling DER message board, will scroll my name across the
front page of my DER
//Author  : Josh Dolgin
//Device  : SMD ATtiny84
//Date    : 20/05/30
//Status  : Working
//Notes   : Pretty simple code, tried to get EEPROM working so message
length can increase

#include <LedControl.h> //Library for MAX7219
#define NUMDISPLAYS 1   //Number of MAX7219 displays
#define DIN PA2         //DATA pin of MAX
#define CLK PA3         //CLOCK pin of MAX
#define LOAD PA1        //LOAD pin of MAX

#define SHIFTSPEED 20   //How often the display is updated (ms)
#define BRIGHTNESS 4    //Brightness of LED's in matrix
#define DISPLAYSIZE 3   //How many columns wide each letter is
#define MATRIX      8   //Number of Columns
LedControl lc = LedControl(DIN, CLK, LOAD, NUMDISPLAYS); //Sets control for
Max7219

uint8_t toDisplay[MATRIX] {}; //Array that is being displayed
```

```
uint8_t backBuffer[MATRIX] {};//Array that shifts into what is being
displayed

String message;          //Message to be displayed
uint8_t pointer = 0;     //8 bit pointer to character in message

uint8_t font[60][DISPLAYSIZE] = { //defines each ASCII value
  { 0x00, 0x00, 0x00 }, //space
  { 0x00, 0x6F, 0x6F }, //!
  { 0x06, 0x00, 0x06 }, //"
  { 0x3E, 0x14, 0x3E }, //#
  { 0x5C, 0xD6, 0x74 }, //$
  { 0x12, 0x08, 0x24 }, //%
  { 0x76, 0x4A, 0x36 }, //&
  { 0x00, 0x06, 0x00 }, //'
  { 0x3C, 0x42, 0x00 }, //(
  { 0x42, 0x3C, 0x00 }, //)
  { 0x0A, 0x04, 0x0A }, //*
  { 0x10, 0x38, 0x10 }, //+
  { 0x60, 0x00, 0x00 }, //,
  { 0x10, 0x10, 0x10 }, //-
  { 0x60, 0x60, 0x00 }, //.
  { 0x60, 0x3C, 0x06 }, // /
  { 0x7C, 0x44, 0x7C }, //0
  { 0x68, 0x7C, 0x60 }, //1
  { 0x74, 0x54, 0x5C }, //2
  { 0x54, 0x54, 0x7C }, //3
  { 0x1C, 0x10, 0x7C }, //4
  { 0x5C, 0x54, 0x74 }, //5
  { 0x7C, 0x54, 0x74 }, //6
  { 0x04, 0x04, 0x7C }, //7
  { 0x7C, 0x54, 0x7C }, //8
  { 0x5C, 0x54, 0x7C }, //9
  { 0x6C, 0x6C, 0x00 }, //:
  { 0xB6, 0x76, 0x00 }, //;
  { 0x10, 0x28, 0x44 }, //<
  { 0x48, 0x48, 0x48 }, //=
  { 0x44, 0x28, 0x10 }, //>
  { 0x06, 0x52, 0x0E }, //?
  { 0x7C, 0x44, 0x5C }, //@
  { 0x7E, 0x12, 0x7E }, //A
  { 0x7E, 0X52, 0x6E }, //B
  { 0x3C, 0x42, 0x24 }, //C
  { 0x7E, 0x42, 0x3C }, //D
  { 0x7E, 0x5A, 0x5A }, //E
  { 0x7E, 0x12, 0x02 }, //F
  { 0x7E, 0x42, 0x72 }, //G
  { 0x7E, 0x18, 0x7E }, //H
  { 0x42, 0x7E, 0x42 }, //I
  { 0x60, 0x40, 0x7E }, //J
  { 0x7E, 0x18, 0x66 }, //K
  { 0x7E, 0x40, 0x40 }, //L
  { 0x7C, 0x08, 0x7C }, //M
  { 0x7C, 0x04, 0x7C }, //N
  { 0x3C, 0x42, 0x3C }, //O
  { 0x7E, 0x0A, 0x0E }, //P
  { 0x0E, 0x0A, 0x7C }, //Q
  { 0x7E, 0x1A, 0x6E }, //R
  { 0x4C, 0x5A, 0x32 }, //S
  { 0x02, 0x7E, 0x02 }, //T
  { 0x7E, 0x40, 0x7E }, //U
  { 0x3E, 0x40, 0x3E }, //V
  { 0x7E, 0x20, 0x7E }, //W
  { 0x66, 0x18, 0x66 }, //X
  { 0x06, 0x78, 0x06 }, //Y
  { 0x72, 0x4A, 0x46 }, //Z
};

void setup() {
  for (uint8_t i = 0; i < NUMDISPLAYS; i++) {
    lc.shutdown(i, false);    //Turns on display
    lc.setIntensity(i, BRIGHTNESS);//Sets brightness
    lc.clearDisplay(i);       //Clear screen
  }
  message = "JOSH.D RSGC ACES'20 "; //Message that is being displayed
  loadBuffer(message.charAt(pointer)); //Loads buffer with first letter
}

void loadBuffer(uint8_t value) {  //Loads ASCII value from string to back
buffer
  value = value - 32; //Subtracts the 32 *null* spaces at the beginning of
ASCII table
  uint8_t util = MATRIX - DISPLAYSIZE;   //Utility variable to load top
cells of buffer
  backBuffer[util - 1] = 0; //Loads a 1 column space between characters
  for (uint8_t i = 0; i < DISPLAYSIZE; i++) { //Loads however many columns
of character
    backBuffer[util] |= font[value][i];
    util++; //Increment util to move up a cell in the array
  }
}

void loop() {
  for (uint8_t shifts = 0; shifts < DISPLAYSIZE + 1; shifts++) { //Runs
through an entire letter + 1 space
    for (uint8_t i = 1; i < MATRIX; i++) {
      toDisplay[i - 1] = toDisplay[i]; //Shifts each cell down 1 (cell 0
gets the contents of cell 1)...
      backBuffer[i - 1] = backBuffer[i];  //Shifts each cell down 1
    }
    toDisplay[MATRIX - 1] = backBuffer[0]; //Loads upfront cell of
backBuffer to back cell of display
    backBuffer[MATRIX - 1] = 0; //Loads a 0 to back cell of back buffer
array
    for (uint8_t row = 0; row < MATRIX; row++) {
      lc.setRow(0, row, toDisplay[row]);  //Runs through and displays each
row of display
    }
    delay(SHIFTSPEED); //delay before shifting in next cell
  }
  pointer = (pointer + 1) % message.length(); //Increment pointer, pointer
=< message length
  loadBuffer(message.charAt(pointer)); //Load new Char
}
```

Procedure

Software

Parts Table	
Component	Quantity
3.6V solar film	1
1206 SMD LED	64
MAX7219	1
ATtiny84	1
60.4 kΩ 1206 resistor	1
10 µF 1206 capacitor	1
0.1 µF 1206 capacitor	1

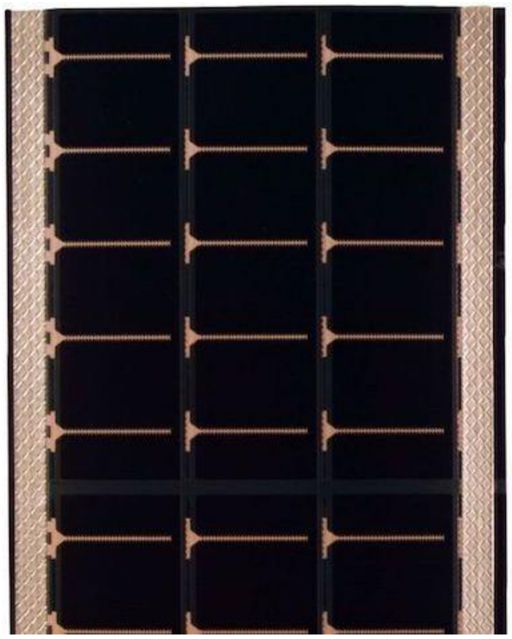
Fill in the necessary comments at the top of the page regarding the content of the code and project, including the MCU, author, date, etc. Include the "LEDcontrol" library, which is used to control the MAX7219 through a variety of commands. Define the pins connected to the DIN, CLOCK, and LOAD pins of the MAX7219 as constants for the program. Define other constants that the program will be dependent on, including the shift speed, size of display, and size of letters. Define 2 arrays, 1 that is the size of the matrix, and the other to hold the buffer values (usually just 8 cells). Create a font matrix to hold each of the letters and symbols that could be displayed, in this project due to limited memory, the ASCII values from space to capital z are used (no lowercase letters).

In the `setup` function, turn on every display to a certain brightness, and ensure they are cleared. Write the message to be displayed and load the first letter of the message into the back buffer. In the "loadBuffer" function, take the ASCII value passed to it and subtract the number 32 so the value points to the respective symbol in the array (there is 32 null characters before space in ASCII so space is technically 32). Using this cell in the array, load the empty cells of the array with the respective row values for the character. In the loop function, create a `for loop` that shifts the contents of each cell in the buffer array and display array down by 1, so that cell 0 is filled with the previous contents of cell 1, cell 1 is filled with the previous contents of cell 2, etcetera. After performing this shift, display the contents of the array on the LED's before shifting again. Once the display has shifted a complete letter + 1 extra space, then a new letter is loaded into the back buffer.

Hardware

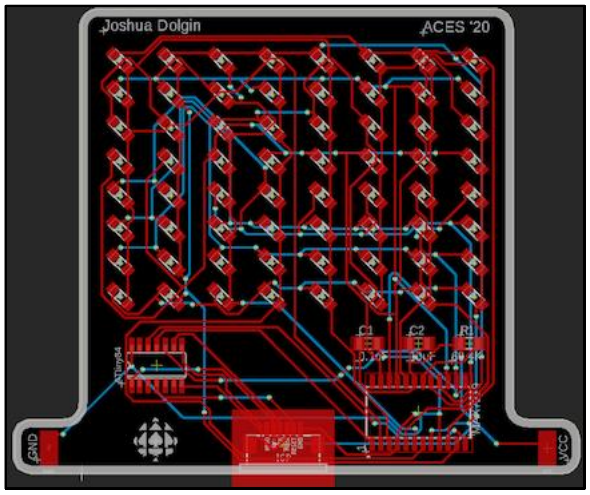
Open EAGLE and create a new schematic. Drag in the LED matrix made in Project 23. The SMD Desktop RTC-Equalizer and reduce the size to an 8×8 matrix. Connect the DIN, LOAD, and CLOCK pin of the MAX7219 to respective I/O pins on the ATtiny84. Connect the LED matrix's rows and columns to the digit and segment pins of the MAX7219 in the exact fashion as in the SMD RTC-Equalizer project linked above. Add a 60.4 kΩ SMD resistor between VCC and the ISET pin of the MAX. Add 2 power regulating capacitors between the input voltage and the VCC pin of the MAX7219. Finally, add the ISP finger connector and connect the VCC, GND, MISO, MOSI, RESET, and SCK pins from the finger to the pins of the same name on the ATtiny84.

Solar film cell



Once the PCB is delivered clean each of the pads with isopropyl alcohol and lay out the components on the silicon soldering pad. Apply small dabs of solder paste to each component pad and gently place each component on the pads (in the correct orientation). Gently swirl the hot air gun in a little circle above each component until the paste melts and pulls the component to the pad. Plug in the ISP finger and confirm that code uploads to the PCB. Finalize and upload the code to the PCB, ensuring everything works. Realize that one of the LED's was never connected in EAGLE and solder a wire connecting it to another LED in the column. Sit back and marvel in the glorious scrolling message.

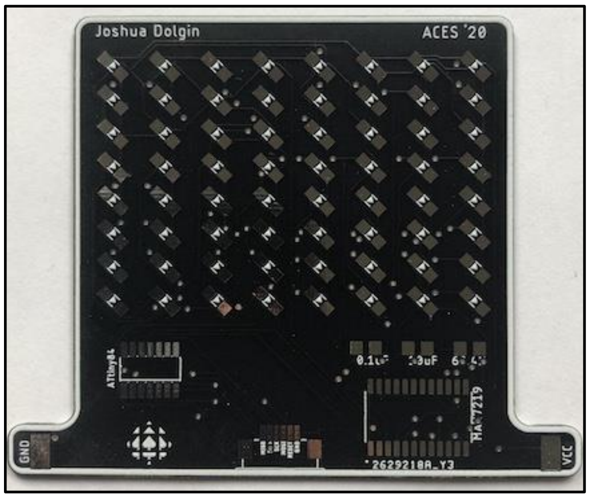
EAGLE view of PCB



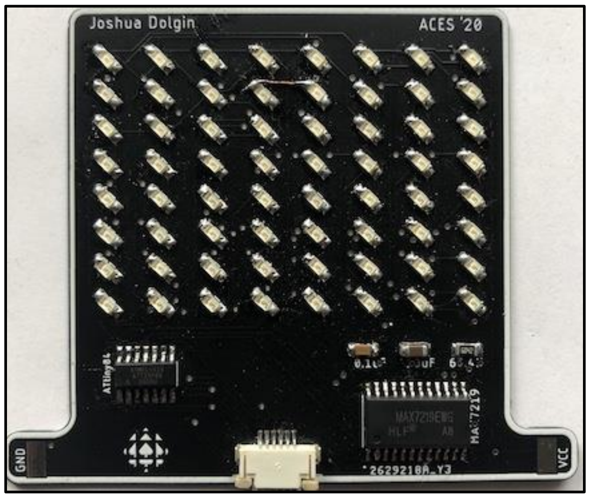
Media

https://www.youtube.com/watch?v=H8A76IpIctg

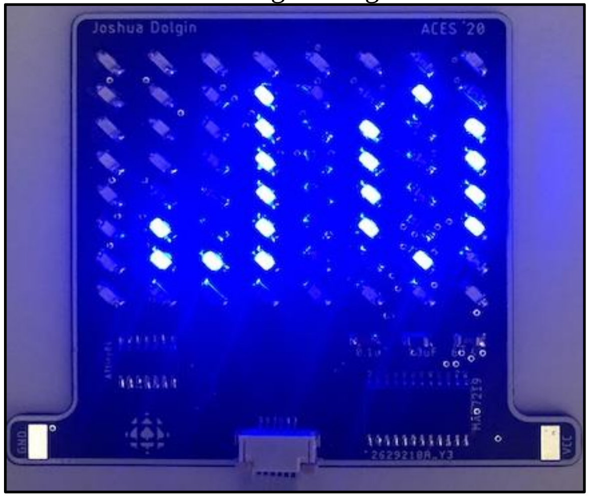
Unsoldered PCB



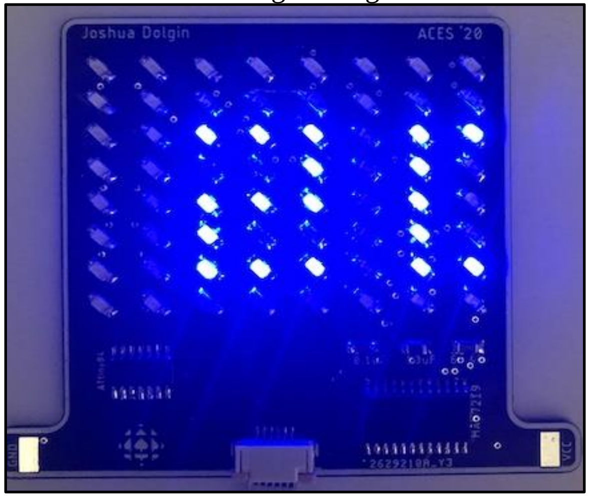
Soldered PCB



Scrolling message 1



Scrolling message 2



Reflection

Overall, this project was fairly successful from the perspective that I got it to work, which is an amazing way to finish my ACES career. Although some aspect of it feels like its missing, I didn't struggle enough with my project. In the past 26 projects, the ones I feel the most pride in writing about are the ones that take those long nights in the DES, staring at lines of code or soldered connections trying to see what one thing doesn't work. I spent more hours on CHUMP than I did on the sum of every grade 10 project, and it is because of that I'll remember the CHUMP project for years to come. I feel disappointed that this project was too easy, and I wish I was at school to add more. The concept is cool, and when I decide to execute on the flexible aspect I'm sure that I will be more content with the product, but as of right now I wish I had a better send-off.

Here we are, the end. What an amazingly stressful time the past 2 and half years have been. I remember in grade 10, I was frantically typing away on a Saturday night, praying I could get all of my ideas in order before the dreaded 11:59 deadline: As my years in the ACES program continued, I became more involved in the projects due every other weekend, investing in learning 3D design to encase as many projects as possible, even handing in my DER hours before the deadline. As I transitioned into grade 12, I had become heavily invested in 3D design, even doing my short ISP on the new multi-material 3D printer upgrade I had built in the DES. As the year continued, I found myself more invested in coding and hardware, offering to help grade 11s with their coding problems just out of sheer interest for finding a solution.

I remember grinning ear to ear as I went through my toolbox the day they were handed out, and I still get that look as I tear through old electronics, pointing out components and their function. I plan to use the MAX7219 module consisting of 4 matrices this summer to code my own mini version of the DES scrolling message board, so I can prop it on my desk wherever I end up to reminisce of where I began. I would like to extend my gratitude to my parents, for affording me this opportunity, spending hours reading through my reports out of interest and commenting on my projects. Thank you to Mr. D'Arcy, for providing such a memorable and enjoyable experience, and teaching me that it is through struggle that we find the most valuable solutions. Finally, thank you to the reader for taking interest in who I am and what I do; this is my passion, and I couldn't be prouder of who I became over the past 2 and a half years in the DES.