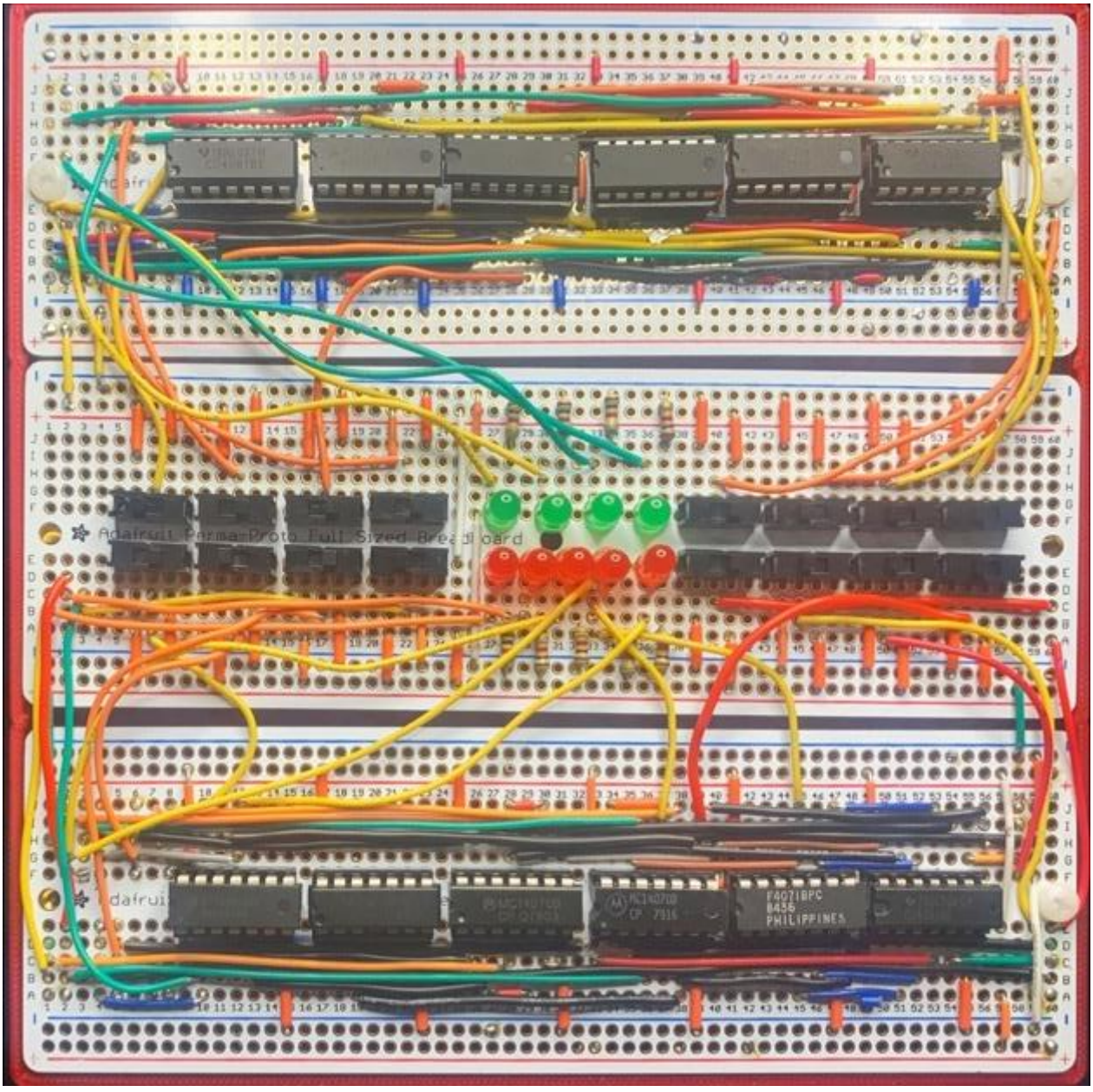
Design Engineering Report



Course: Introduction to Computer Studies
Code: ICS2O-E, ICS3U-E, ICS4U-E
Author: Max Zischka
Date: November 13, 2026

Table of Contents

PROJECT 1.1 VOLTAGE H-BRIDGE 1

PURPOSE 1
REFERENCE 1
PROCEDURE 1
MEDIA 2
REFLECTION 2

PROJECT 1.2 JOULE THIEF 3

PURPOSE 3
REFERENCE 3
PROCEDURE 3
MEDIA 5
REFLECTION 5

PROJECT 1.3 1-BIT MAGNITUDE COMPARATOR 6

PURPOSE 6
REFERENCE 6
THEORY 6
PROCEDURE 6
MEDIA 8
REFLECTION 10

PROJECT 1.4. A COUNTING CIRCUIT 11

THEORY 11
A. ANALOG INPUT 11
Purpose 11
Reference 11
Procedure 11
Media 11
B. NAND GATE OSCILLATOR 12
Purpose 12
Reference 12
Procedure 12
Media 13
C. DECADE COUNTER 14
Purpose 14
Reference 14
Procedure 14
Media 15
D. DECIMAL COUNTING UP/DOWN COUNTER 15
Purpose 15
Reference 15
Procedure 15
Media 16
E. BINARY COUNTING DECIMAL DECODER 17
Purpose 17
Reference 17
Procedure 17
Media 18
F. Seven-Segment Display 19

Purpose ... *19*
Reference ... *19*
Procedure ... *19*
Media ... *20*
G. A Counting Circuit PCB ... 21
Purpose ... *21*
Reference ... *21*
Procedure ... *21*
Media ... *22*
H. A Counting Circuit Case ... 22
Purpose ... *22*
Reference ... *22*
Procedure ... *22*
Media ... *23*
I. A Two Digit Counting Circuit Display ... 24
Purpose ... *24*
Reference ... *24*
Procedure ... *24*
Media ... *25*
Media ... 25
Reflection ... 25

ICS3U-E ... 26

PROJECT 2.1 DIGITAL-TO-ANALOG CONVERSION ... 27

Purpose ... 27
Reference ... 27
Theory ... 27
Procedure ... 27
Media ... 29
Reflection ... 29

PROJECT 2.2 THE 555 TIME MACHINE ... 30

Purpose ... 30
Reference ... 30
Theory ... 30
Procedure ... 32
Media ... 34
Reflection ... 34

PROJECT 2.3 MULTI-BIT MEMORY (REGISTER) ... 35

Purpose ... 35
Reference ... 35
Theory ... 35
Procedure ... 36
Media ... 38
Reflection ... 39

PROJECT 2.4 BINARY GAME PART 1 ... 40

Purpose ... 40
Reference ... 40
Theory ... 40
Procedure ... 41

CODE 42
MEDIA 44
REFLECTION 44

PROJECT 2.5.3 SHORT ISP: THE BINARY CALCULATOR 45

PURPOSE 45
REFERENCE 45
THEORY 45
PROCEDURE 46
MEDIA 48
REFLECTION 50

PROJECT 2.6. A TINY CLOCK 51

PROJECT 2.6.1 INTER-INTEGRATED CIRCUIT(I2C) 51
Purpose 51
Reference 51
Theory 51
Procedure 52
Code 53
Media 53
Reflection 54
PROJECT 2.6.2 REAL TIME CLOCK (RTC) PROTOTYPE 55
Purpose 55
Reference 55
Theory 55
Procedure 56
Code 58
Media 61
Reflection 62

2.7.3 MEDIUM ISP THE MAGIC CHESSBOARD 63

PURPOSE 63
REFERENCE 63
THEORY 63
PROCEDURE 64
CODE 67
Arduino Program 67
Pi Program 72
MEDIA 73
REFLECTION 75

ICS4U-E 76

PROJECT 3.1 CHARLIECLOCK 77

PURPOSE 77
REFERENCE 77
THEORY 77
PROCEDURE 78
CODE 80
Atmega328p 80
Attiny85 83
MEDIA 84
REFLECTION 86

PROJECT 3.2 CHUMP **87**

PROJECT 3.2.1 CHUMP: CLOCK AND COUNTER 87
Purpose *87*
Reference *87*
Theory *87*
Procedure *88*
Media *91*
Reflection *91*
PROJECT 3.2.2 CHUMP: EEPROM 92
Purpose *92*
Reference *92*
Theory *92*
Procedure *93*
Code *95*
Media *97*
Reflection *98*
3.2.3 CHUMP: ALU EXPLORER 99
Purpose *99*
Reference *99*
Theory *99*
Procedure *100*
Media *101*
Reflection *102*
3.2.4 CHUMP: FINAL BUILD 103
Purpose *103*
Reference *103*
Theory *103*
Procedure *103*
Code *107*
Media *109*
Add-ons *112*
Reflection *112*

LONG INDEPENDENT STUDY PROJECT: AUDIO SPECTRUM ANALYZER **113**

PURPOSE 113
REFERENCE 113
THEORY 113
PROCEDURE 115
MEDIA 118
REFLECTION 120

3.3 AVR ASSEMBLY: TRAFFIC LIGHT **121**

PURPOSE 121
REFERENCE 121
THEORY 121
PROCEDURE 121
CODE 122
MEDIA 123
REFLECTION 123

SUCCESSIVE APPROXIMATION REGISTER ANALOG-TO-DIGITAL-CONVERTER **124**

SAR ADC: OVERVIEW AND CLOCK 124

Purpose *124*
Reference *124*
Theory *124*
Procedure *125*
Media *126*
Reflection *126*
SAR ADC: R/2R Ladder DAC 127
Purpose *127*
Reference *127*
Theory *127*
Procedure *128*
Media *129*
Reflection *129*
SAR ADC: Completed 130
Purpose *130*
Reference *130*
Theory *130*
Procedure *131*
Media *132*
Reflection *134*

MEDIUM ISP: ACES DC-DC CONVERTERS 135

Purpose 135
Reference 135
Theory 135
Procedure: Boost Converter 140
Procedure: SEPIC Buck-Boost Converter 142
Code 143
Media 144
Reflection 145

AVR ASSEMBLY: DOUBLE DABBLE 146

Purpose 146
Reference 146
Theory 146
Procedure 146
Media 148
Code 149
Reflection 149

PROJECT 3.7 4517 GRAY CODE 150

Purpose 150
Reference 150
Theory 150
Procedure 151
Media 153
Reflection 154

Project 1.1 Voltage H-Bridge

Purpose

The purpose of this project is to develop a deeper understanding of current flow in a circuit using a potentiometer and light-emitting diodes.

Reference

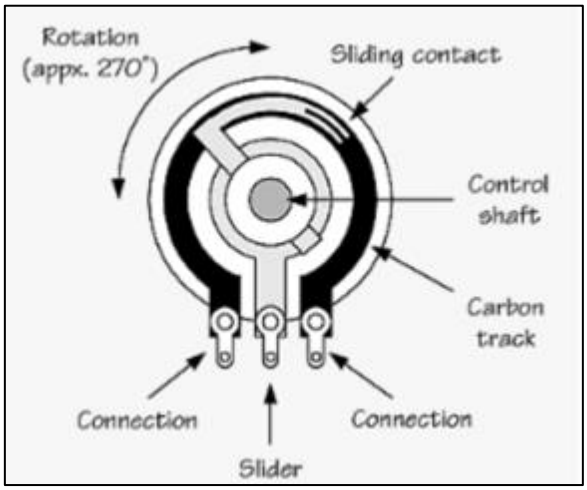
Project Description: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#hBridge
Falstad Circuit Simulator: https://www.falstad.com/circuit/

Procedure

The procedure for this project starts with the parts used. The parts used are displayed on the parts table shown. Each part helps to complete the purpose of this project. A potentiometer configured as a voltage divider establishes a variable fractional battery voltage at the first point B of the circuit. Two 470 Ω resistors establish a fixed voltage drop that is either greater, equal to or less than the fractional battery voltage at a second point D of the circuit. The 5mm Bicolor LED is an indicator of the direction in which the current is flowing between points B and D.

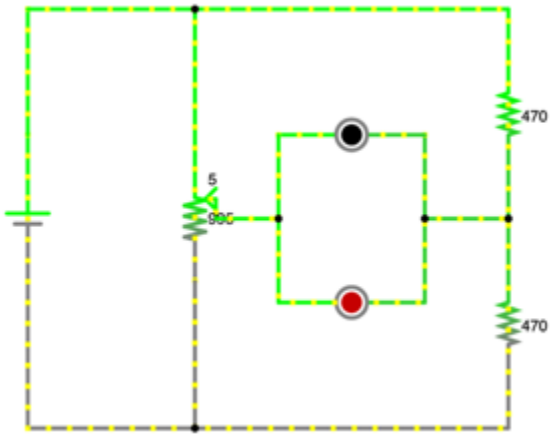
Parts Table	
Quantity	**Description**
1	10 kΩ Potentiometer
2	470 Ω Fixed Resistor
1	5 mm Bicolor LED
1	9V Power Supply

The potentiometer has 3 contacts those being A, B and C. The resistance between A and C is fixed, but the resistance between A and B, and B and C varies with the position of the potentiometer. The two 470 fixed resistors will each always take up 4.5V. Current always flows from higher to lower voltage and the greater resistance a resistor has the greater the voltage drop. These statements of the resistors taking up this amount of voltage are true based on Kirchhoff's voltage law that all voltage must be used up in a circuit.

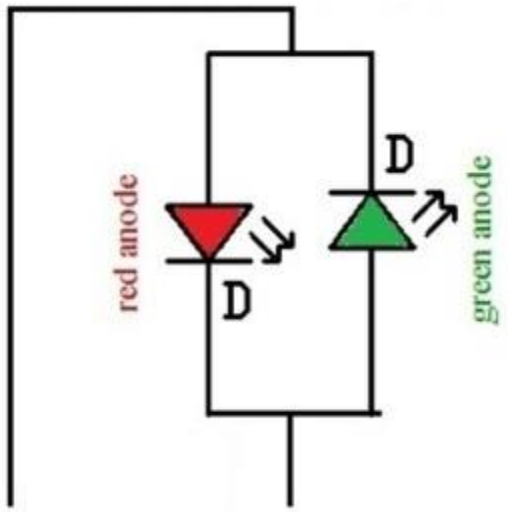


The procedure of the circuit to meet the purpose relates to voltage division. In this circuit, there are two types of voltage division fixed and variable. Fixed voltage resistance is done with the two

The variable division is done based on the formula $(R2 \div (R1+R2)) \cdot V$. Based on the information that $R1+R2=10\ k\Omega$. The potentiometer points A-B can be represented as R1 being all the resistance between those two points. Points C-B can be represented as R2 also. If R1 is greater than R2 then the voltage at point B will be less than at point D between the fixed resistors. If R1 is less, then the voltage at point B will be higher than point D and if R1 equals R2 then the voltage at point B will be the same as at point D.



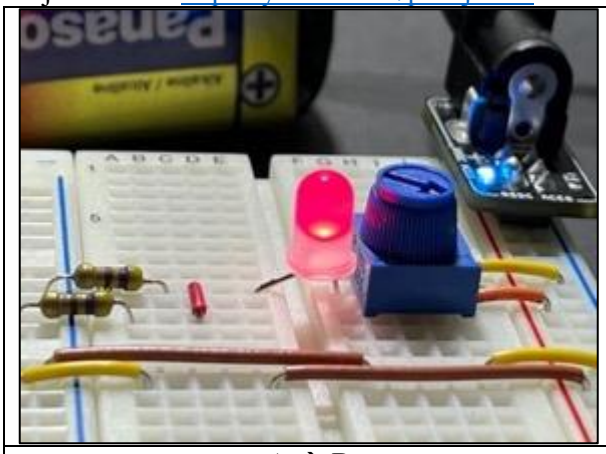
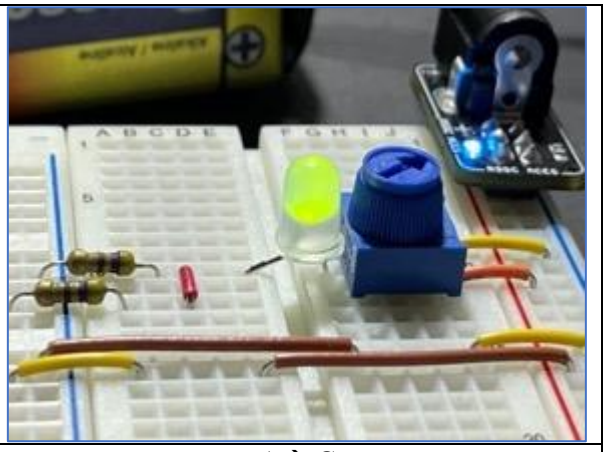
Fixed voltage division is done with the two 470 Ω resistors. Once again using that formula we can state that R1 represents one of the 470 resistors while R2 represents the other. The formula (R2 ÷ R1 + R2) · V at point D will always equal 0.5V battery. The fixed resistors still play a vital role in the circuit as the voltage drop at point B of the potentiometer either meets, exceeds or has less. Without the two fixed resistors voltage division would occur in 2 places and the voltage drop would not vary turning on both colours of the LED.



The 5 mm Bicolor LED plays a large role in the purpose of displaying the current flow. Instead of a normal LED that allows current to flow one way and stops currently in the other, the bicolor LED has 2 LEDs inside on bulb. The reason a Bicolor LED is perfect for this project is that with a normal LED, the only factor it shows is if current flows or not, with this it shows that factor along with the direction of current flow.

Media

Project Video: https://youtu.be/U7p2S-jukNs

	
A → B	A→ C

Reflection

One thing that this project did differently than other subjects was let me figure out the problems and the solutions; it put me in the driver's seat. I had to figure out the timeline, and I had to invest time after school in the DES. I think the idea of putting the student in the driver's seat is one of the school issues, it is what makes creativity so hard when you get into the real world and what makes this course so interesting and important.

Project 1.2 Joule Thief

Purpose

The purpose of the Joule Thief is to show how an inductor can store electrical energy that is powered by a 0.5V to 1.5V battery, can work through a 3904 transistor and is indicated through a white light emitting diode.

Reference

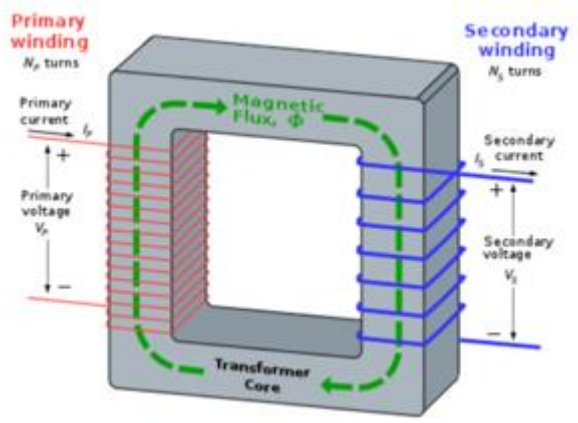
Project Description: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#joule
Joule Thief Explanation: https://www.youtube.com/watch?v=0GVLnyTdqkg

Procedure

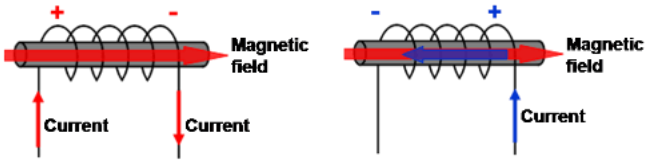
The procedure for the Joule Thief begins with the parts needed to create this circuit. There are several parts needed to create this circuit. The *ACES' Wound Toroid Ferrite Ring* with two opposing wires that power the circuit but also collapse the circuit with the magnetic field it creates between the two wires. The 3904 Transistor acts as a gate that works when the base is powered, if not powered it collapses the circuit powering the LED. The switch is connected to the ground which provides a ground connection when on. The 2.8V White LED is an indicator of when the transistor is open or closed. The 1 kΩ resistor takes up some of the current to not fry the transistor. Finally, the 0.5V to 1.5V AA Battery creates a circuit that usually with a spent battery wouldn't power the LED but the circuit makes it able for this spent battery to power the 2.8V LED.

Parts Table	
Quantity	**Description**
1	ACES Wound Toroid
1	3904 Transistor
1	Switch
1	2.8 V White LED
1	1 kΩ Resistor
1	0.5V to 1.5V AA Battery

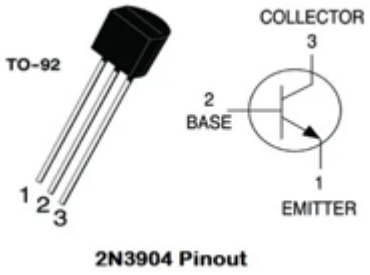
In 1820 Ørsted's law discovered that electric current flowing through a wire creates a magnetic field. This law is how the Joule Thief can work because the Toroid ferrite ring is wrapped with two opposing wires that create magnetic fields that interact, those magnetic fields induce the current to make the Joule Thief possible. The way this circuit functions is by following Faraday's law of induction. Faraday's law says that when the two opposing wires start to build up a magnetic field it creates a magnetic flux. In this circuit, the magnetic field is larger on the primary winding (wire connected to the collector) because of the way the wires are wound around the ferrite ring.



The image to the right shows how the current functions when these two magnetic fields are created. This affects the current being flowed through the secondary winding. The primary winding on the circuit starts to create an induced current adding current to the secondary winding. Based on Lenz's law induced current always flows in the opposite direction in which it was produced. The image on the right shows how induced current is created and flows.

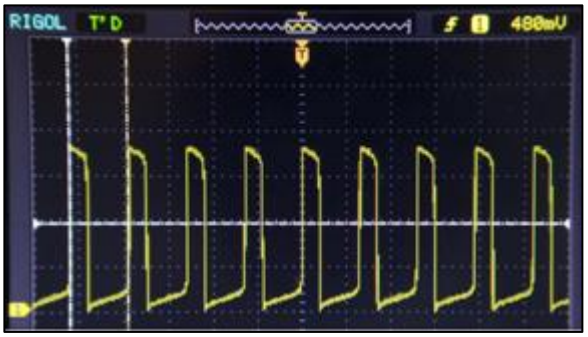


Faraday and Lenz's laws are the underlying factors that interrelate magnetic field and current. When the transistor and inductor interact, an increase in current induces an increased voltage in the wire connected to the base which oscillates the transistor on and off. This then creates a feedback loop, increasing the voltage on the collector and turning the LED on. The transistor used in this circuit is a 3904 Transistor. This transistor has three leads as shown on the right. Pin 1 is the emitter, pin 2 is the base and pin 3 is the collector.

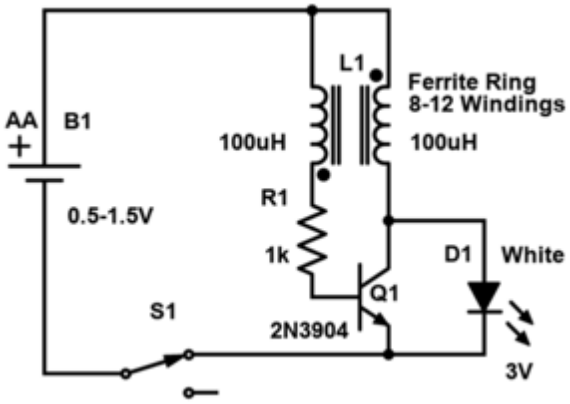


2N3904 Pinout

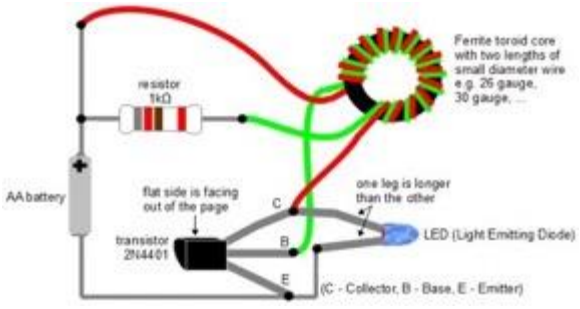
The reason this transistor is crucial is that the 3904 transistor allows current to flow through two leads, the base and emitter. The current will only flow through the collector if the base has current being flowed to it, the feedback loop continues rapidly, increasing the voltage on the lead of the LED. Understanding how the transistor operates in the circuits helps with the understanding of how the circuit collapses and generates as seen in the image on the right.



Developing an understanding of how the ACES' Wound Toroid Ferrite ring works and the transistor will show how the AA battery can power the LED. On the right is the Joule Thief schematic. Looking into it the two 100 micro-henry inductors play the role of the Toroid ferrite ring. Current begins to flow from the AA battery's positive lead and flows through the secondary coil connected to the 1 kΩ resistor and then into the base lead of the transistor. With the current at the base transistor, the collector opens, allowing current to flow through the collector which starts to induce magnetic fields in both coils. Because the primary is intertwined with the secondary coil, this creates an induced current through current and secondary content perpetuates the feedback loop. When the feedback loop has generated enough current on the base, the transistor opens completely, connecting the collector and emitter pins.



Once the magnetic field has grown as much as it possibly can there is no more magnetic flux resulting in no more induced current. When the collector receives more current than the base the transistor turns off. This leads the magnetic field to collapse and pushes an extra amount of current to the 2.8V LED turning it on. Once that occurs the transistor receiving no current through the collector or base can turn on again restarting the circuit. The image to the right shows the circuit in real life with the red wire being the primary wire and the green wire being the secondary wire.

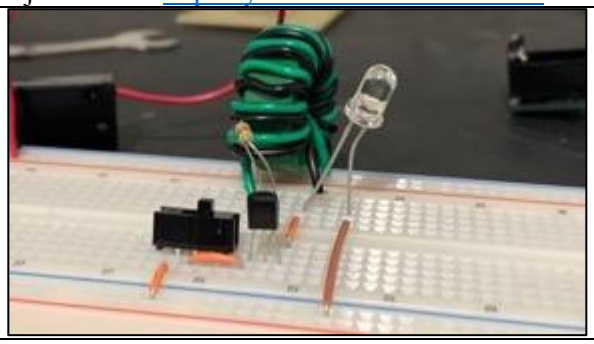


After understanding how the Joule Thief works on a breadboard the circuit, this be implemented onto a *printed circuit board* (PCB). The image shows the parts table shows the parts used. The PCB circuit uses 2 new parts and replaces the Ferrite ring with two 100 µH fixed inductors. These inductors feature a ferrite core and copper coils, this creates that magnetic field. The solder used is a metal alloy made of tin, lead and gold that when heated up to 300°C liquefies allowing it to flow to the PCB and connect the parts to the PCB.

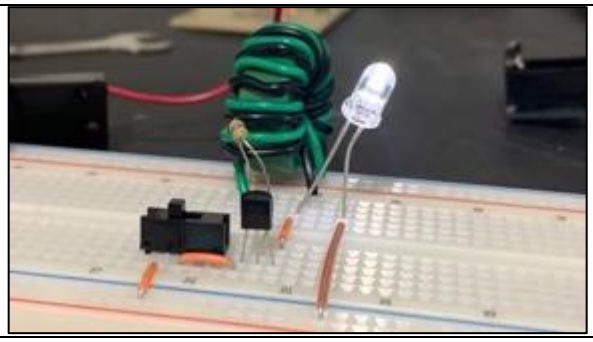
Parts Table	
Quantity	**Description**
1	ACES Joule Thief PCB
2	100 µH Fixed Inductor
1	2.8V White LED
1	AA Battery Holder
1	1 kΩ Resistor
1	3904 Transistor
1	Switch
1	0.5V to 1.5V AA Battery

Media

Project Video: https://youtu.be/DwkUtAMsvuk



Breadboard Joule Thief with Switch Off



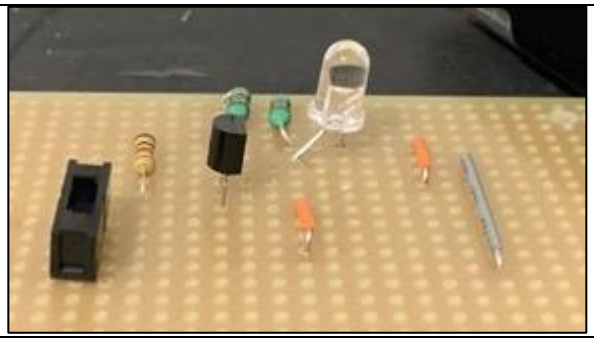
Breadboard Joule Thief with Switch On



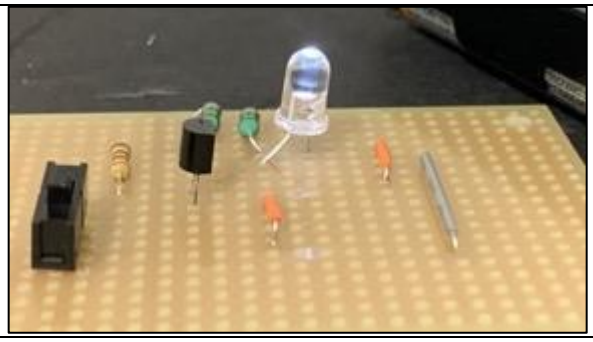
PCB Joule Thief with Switch Off



PCB Joule Thief with Switch On



Stripboard Joule Thief with Switch Off



Stripboard Joule Thief with Switch On

Reflection

This project was a step up from the last project which was something I liked. The project allowed me to put in lots of hours to something interesting yet hard and, in the end, looking at the DER I am very proud of the experience I had. Overall, the experience taught me how to solder, taught me about inductors, magnetic fields and so much more.

Project 1.3 1-Bit Magnitude Comparator

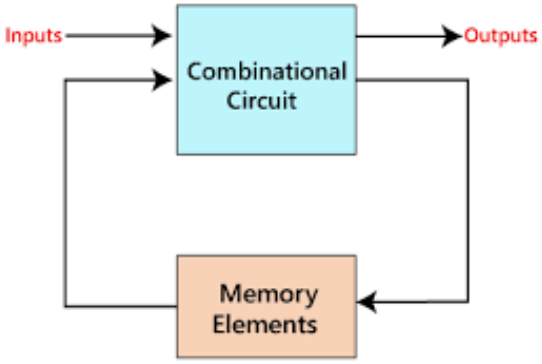
Purpose

The purpose of the 1-Bit Magnitude Comparator is to demonstrate how logic gates are configured to compare binary inputs.

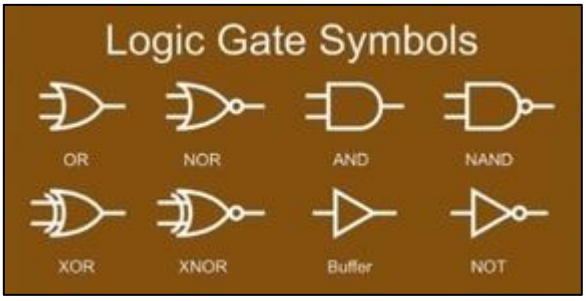
Reference

Project Description: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#magnitude
1-Bit Magnitude Comparator Explanation: https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/

Theory



Before understanding how the 1-Bit Magnitude Comparator works understanding the difference between a combinational circuit and a sequential circuit is essential. A combinational circuit uses the inputs that are given at the moment the circuit is functioning to produce an output while a sequential circuit is using the inputs present in the circuit while also using inputs that have been inputted in the past to add on to the output. An example of a combinational circuit is the half-adder circuit which uses the inputs given in a moment to indicate what LED will turn on. An example of a sequential circuit is a digital clock that uses the inputs given at the moment as well as adding onto the inputs that the clock has had in the past to give the time. The image on the right shows how a combinational circuit becomes sequential.



Procedure

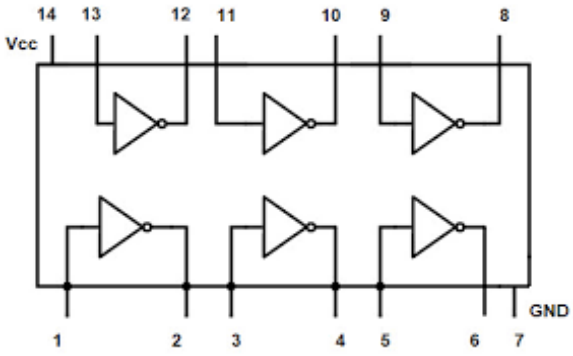
The 1-Bit magnitude comparator focuses on how digital computers use binary to produce the correct outputs. In the real world when working with numbers math uses the base ten system because of the number of fingers on both of our hands yet the binary system is in base two so the only two numbers used are zero and one to represent on and off. In math, numbers are configured using addition, subtraction, multiplication and division yet in the world of binary logic gates are what configure the inputs. The image on the right shows eight logic gates that differently configure inputs.

The procedure for the 1-Bit Magnitude Comparator begins with the parts used to assemble it. The switch used in this circuit is a single pole double throw which has two positions where current flows. The two positions are the power supply through a resistor to the ground (logic number 0) or from the power supply through the circuit (logic number 1). The 1 kΩ resistor makes sure that if the switch takes the nine volts right to the ground that the resistor uses the voltage. The RGB LED is used in the circuit to 3 leads that take current from the AND chip coming from the A power supply, the output of the NOR chip and the AND chip coming from the B power supply. The circuit with the logic chips is designed so that only one colour is on the LED at once.

Parts Table	
Quantity	**Description**
1	4069(NOT) Logic Chip
1	4081(AND) Logic Chip
1	4001(NOR) Logic Chip
2	SPDT Switch
3	1 kΩ Resistors
1	RGB 4 Pin LED

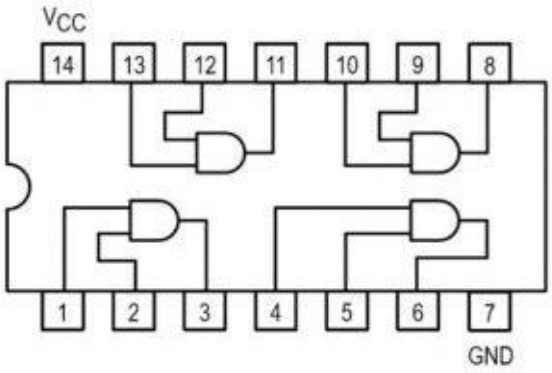
Input	Output
A	Y
1	0
0	1

Another main component of this circuit is the logic gates that make it possible for the binary inputs to compare and turn on specific colours on the LED based on inputs. For this circuit, there are two different types of logic gates. The first logic gate is labeled 4069 on the chip which states that it is a NOT logic chip. The reason it has the name NOT is because of what the chip does based on the input. The truth table on the right shows what happens to the input when it goes through a NOT logic chip. As the table shows, when you give an input of 1 the output of the gate will have an output of zero and if the input is zero the gate will produce an input of one through the power source on gate fourteen. The image on the right shows that the NOT logic chip is a one-input gate that only evaluates one input and changes it to the opposite. The logic chip has two inputs for power and ground which are how the chip works: when the chip receives an input of zero through input gate one the chip receives the input zero then the power source on gate fourteen will produce a current that is output through output gate one.



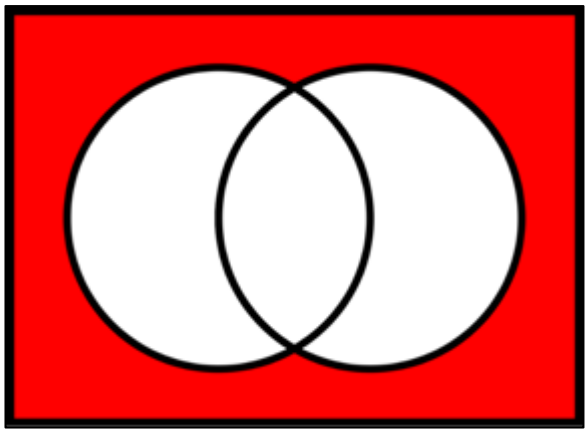
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The second logic chip used in this circuit is the 4081 chip also known as the AND chip. This chip provides the function of only allowing current to flow through if both inputs have current flowing through them. The chip used here is comparing two inputs and only allows the input one and one to produce an output of one. The truth table on the right shows how this will function. As the image shows when anything except for the two last inputs is pressed then the chip will not send an output. The AND chips have a very useful function in the 1-Bit magnitude comparator because without that chip correlating with the NOT chip it wouldn't allow the LED that is created from the 1-Bit magnitude comparator to produce just the red, blue and green colour outputs. The photo on the right shows how this logic chip has four gates inside of it and that the chip needs two inputs of 1 to produce the output of one.

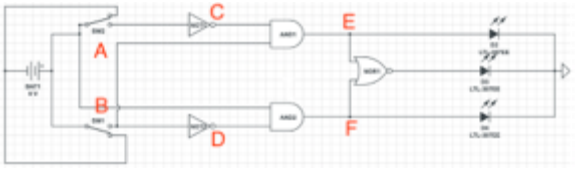


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

The final chip used in this circuit is the 4001 chip which is known as the NOR chip. The NOR chip has the function of being a combination of the OR chip and the NOT chip. The OR gate works to allow every input to have an output of one except if both input gates have an input of zero then no output will be produced along with the NOT gate having the function of inverting the output that the OR produces. In the truth table of the NOR gate, the inputs are completely reversed from the truth table of the OR gate because of the inverse the NOT part gives you in the NOR chip. This chip is very important to the circuit as it is the only chip that only allows the inputs zero and zero to produce one while having no other inputs work.



Understanding how the switch and logic gates operate in a binary circuit helps with the understanding of when an input is put into the 1-Bit magnitude comparator. The input starts at the single-pull-double-throw switch when has two inputs of one or zero. When the input of one is put into both switches the inputs are moved through the NOT gates to turn the input of one to zero. After moving through the NOT gate the inputs are put through the AND gate and because the two inputs are one and zero the AND gate produces an output of zero through both AND gates. The NOR has only one output of one if both inputs are zero the two inputs in this scenario output a value of one through the LED in the middle connected to NOR turning the colour green. This flow of inputs just shown works for the inputs zero and zero along with the inputs one and one. Now if the inputs are changed to one out of A and B has a value of zero then B with go through the NOT gate and turn to one yet this time because A has a value of one then the AND gate at F will produce a value of one make the NOR unable to function and only creating one pathway to the blue LED. The procedure is just done with A being one and B being zero will produce the same output with a different LED turning on if A is zero and B is one. Now looking at this from a truth table perspective all different inputs give different outputs based on the chips used. The Microsoft Excel formulas show how the inputs would flow in a circuit. With the image on the right, the idea of combination circuitry is shown with the N – bit comparator not needing any memory.



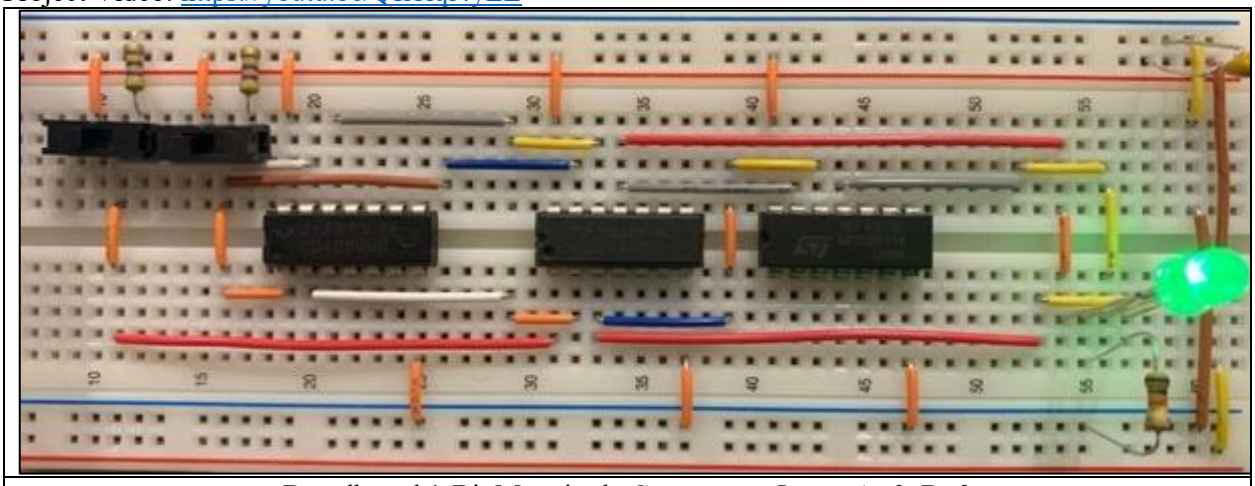


NOT				AND			NOR
A	B	C	D	E	F		G
0	0	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
0	1	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
1	0	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
1	1	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

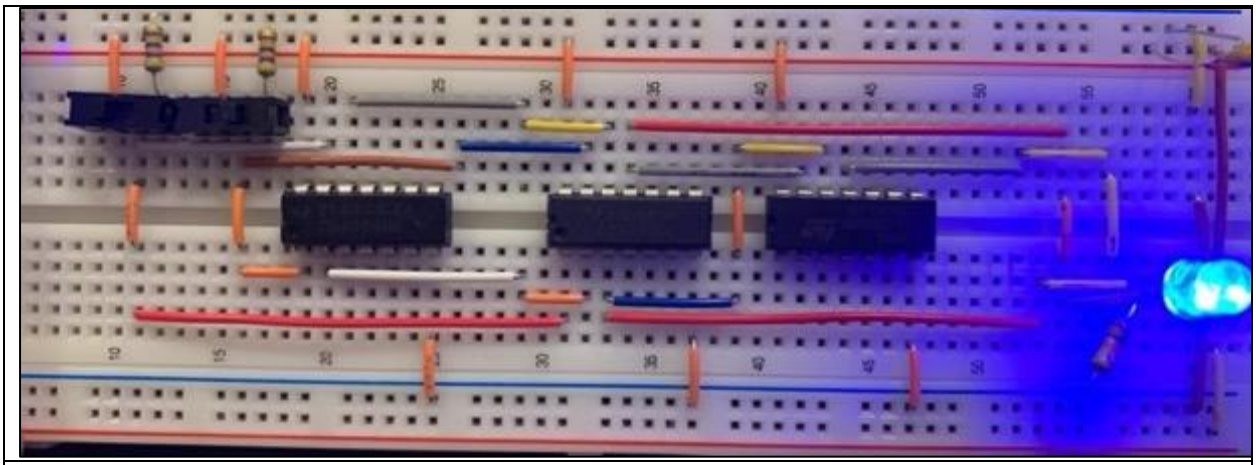


Media

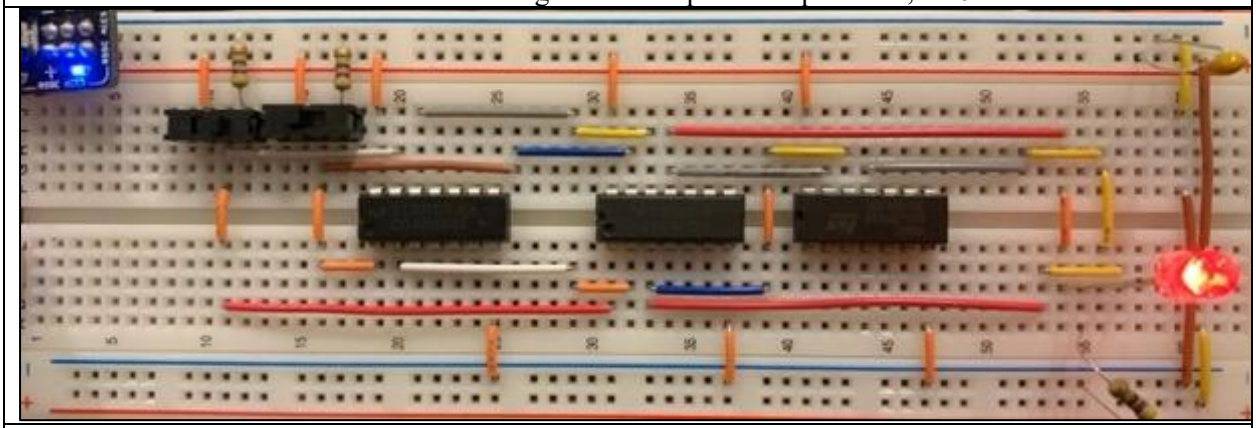
Project Video: https://youtu.be/Qfi1lql4yLE



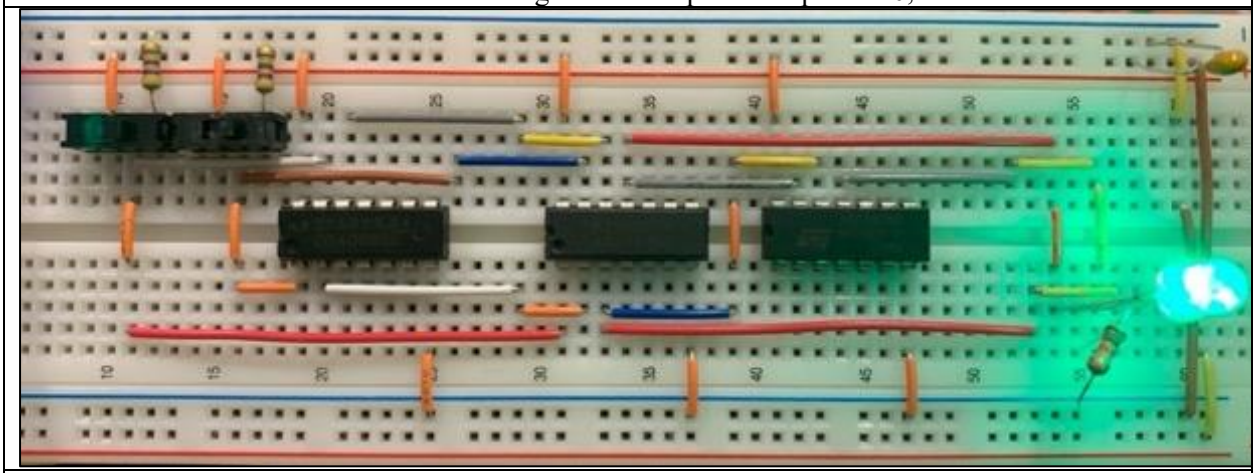
Breadboard 1-Bit Magnitude Comparator Input: A=0, B=0



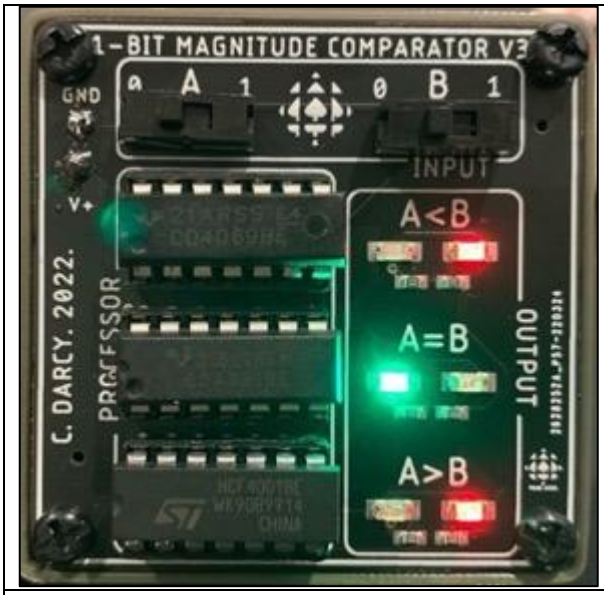
Breadboard 1-Bit Magnitude Comparator Input: A=1, B=0



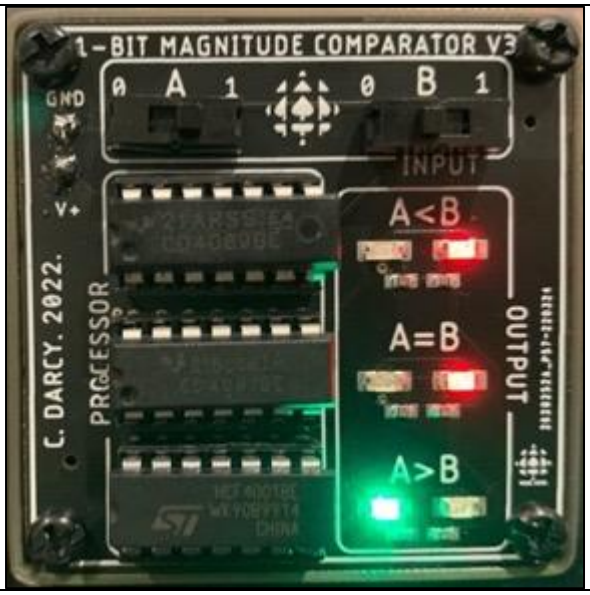
Breadboard 1-Bit Magnitude Comparator Input: A=0, B=1



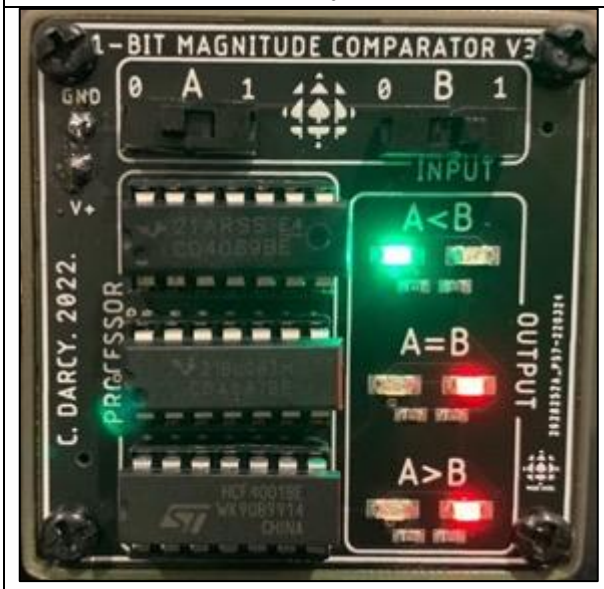
Breadboard 1-Bit Magnitude Comparator Input: A=1, B=1



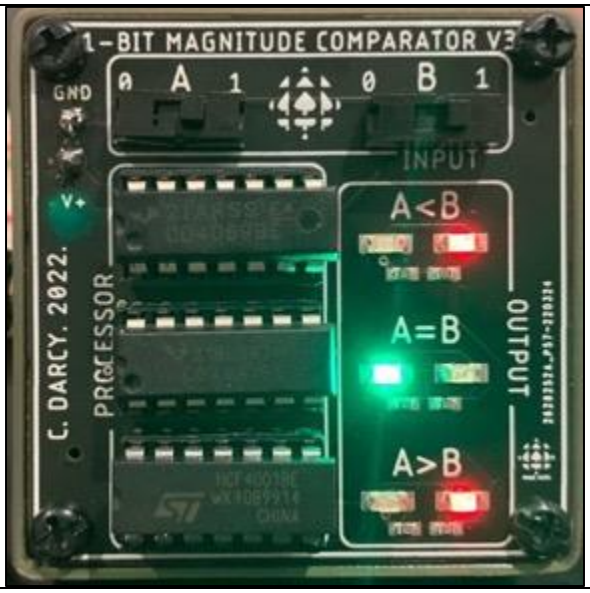
PCB 1-Bit Magnitude Comparator Input: A=0, B=0



PCB 1-Bit Magnitude Comparator Input: A=1, B=0



PCB 1-Bit Magnitude Comparator Input: A=0, B=1



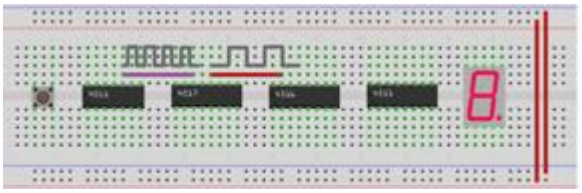
PCB 1-Bit Magnitude Comparator Input: A=1, B=1

Reflection

This project was a big change from the last with the change from analog to digital but I think it was a great eye-opener on what I hope to take of this course, understanding computers. Binary is such a large part of computers and it was great to understand something I have seen so much in my life. It was a hard project with time management but I feel in the end the classes planned out the learning very well to meet the needs of writing the DER. I was a little upset when Rohan and I tried the 2-bit magnitude comparator for three hours yet failed.

Project 1.4. A Counting Circuit

Theory



The counting circuit is an example of sequential digital logic circuitry. The electronic counter uses oscillation to create two states in a circuit that send two continuously different signals through the circuit and count the clock pulses. The clock pulses produce a rising and falling edge along with a momentary period that has an output of one. This pulse is then able to send the rising edge input through a series of IC Logic Chips to produce a display that sequentially goes up as time does.

A. Analog Input

Purpose

The purpose of the analog input is to explore how a momentary input creates an output of a fixed duration through the PBNO's momentary input and the known states provided by the pull-down resistor.

Reference

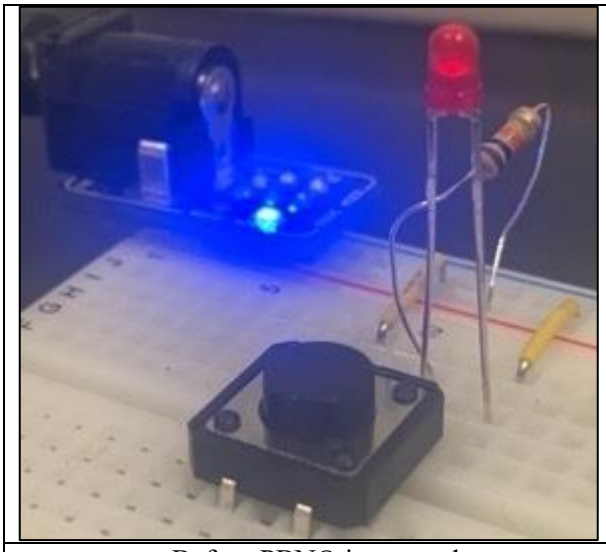
Project Description: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#counting

Procedure

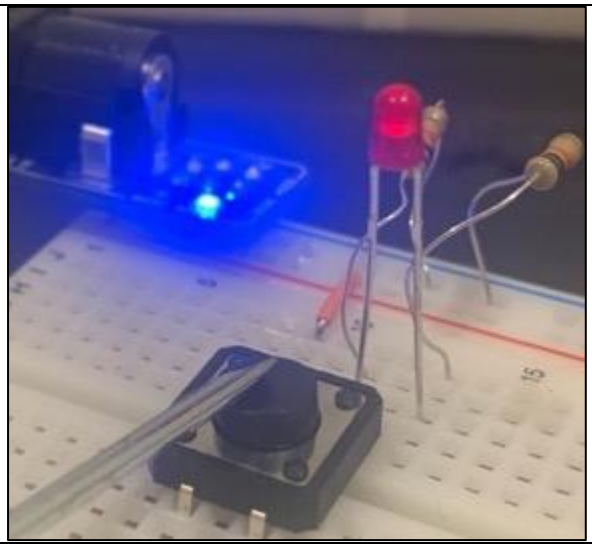
The procedure for the analog input is that it provides two know states through the input given by the PBNO and the pull-up resistor. The pull-up resistor is able tell the state of the circuit as either zero or one as the resistor will take the 9V if ground is presented and if not than the voltage will flow throw the circuit. The LED acts as the indicator. When using a pull-down resistor an input in needed before any output has voltage and the opposite is for the pull up resistor.

Parts Table	
Quantity	**Description**
1	10 kΩ Resistors
1	PBNO
1	Red LED

Media



Before PBNO is pressed



After PBNO is pressed for a momentary time

B. NAND Gate Oscillator

Purpose

The purpose of the NAND Gate Oscillator is to display how two capacitors, one controlling the time the entire circuit is on and the other can create repeated rising and falling edges. The rises and falls begin to create the decade counter needed for the counting circuit.

Reference

Project Description:

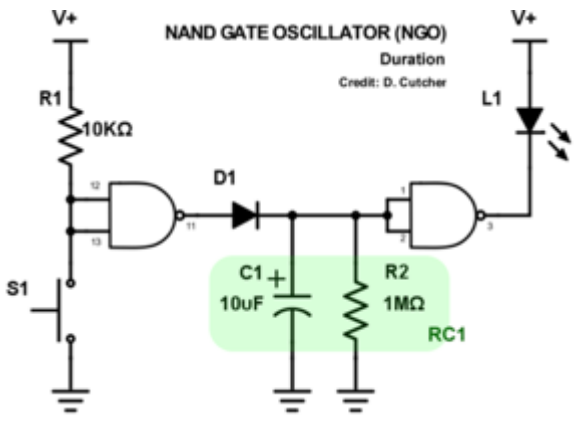
Procedure

The procedure for the NAND gate oscillator a is a very important part of the counting circuit, the advancement made during this part of the circuit is that the NAND gate oscillator takes the fixed duration in which the circuit is on and adds several rises and falls to that fixed duration. The way the circuit can have several rises and falls during that fixed duration is based on the parts used to create the circuit.

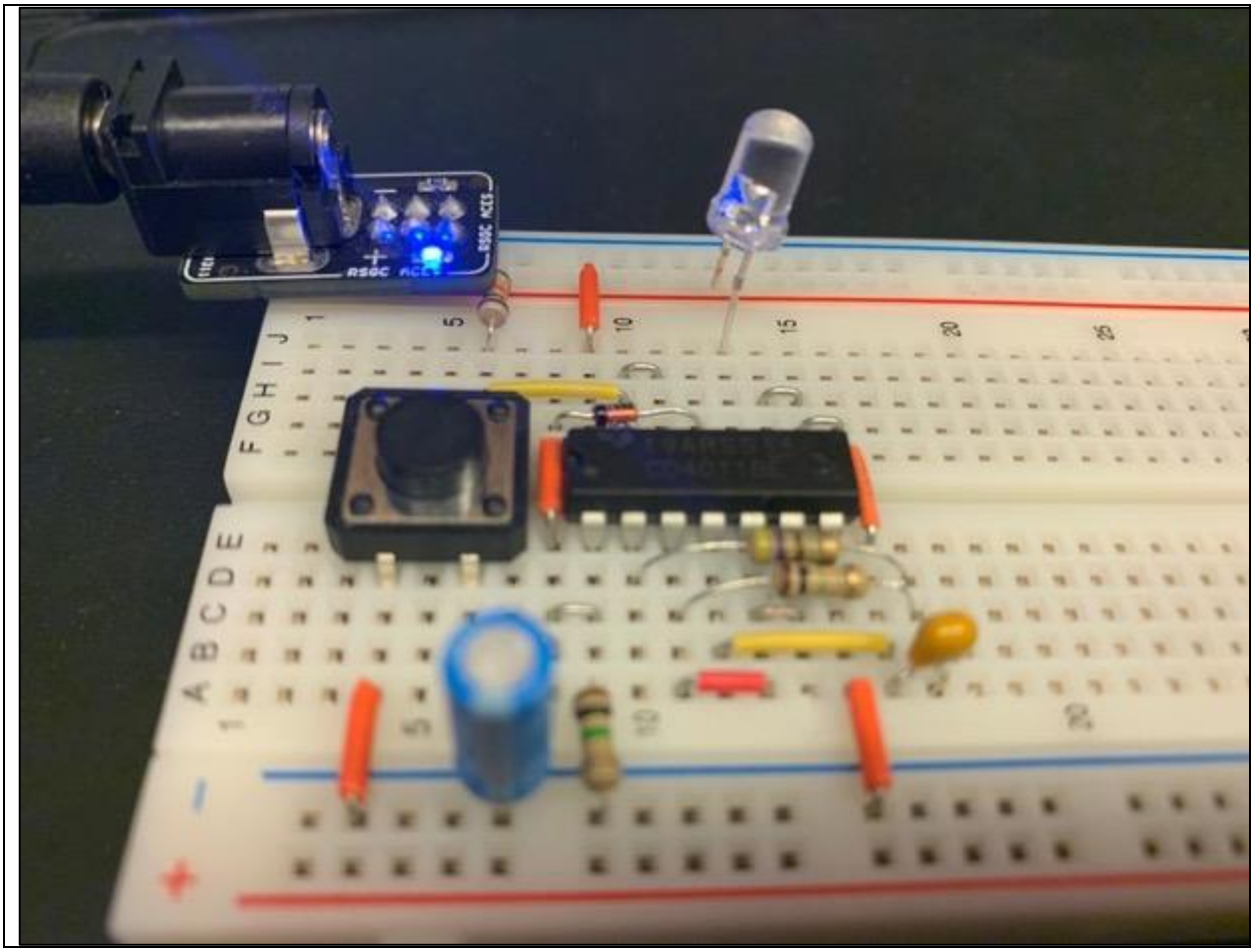
Parts Table	
Quantity	**Description**
2	1 MΩ Resistor
1	4011 CMOS IC Logic Chip
1	Diode
1	10µF Capacitor
1	10 kΩ Resistors
1	PBNO
1	Blue LED
1	470 Ω Resistor
1	0.1µF Capacitor

The analog input starts with the parts used to create this circuit. Firstly, the analog input is shown through the blue LED to indicate when the PBNO has input and when the capacitor has current flowing through the circuit. The NAND chip is what makes the input of the circuit possible to create the output needed in an analog input. The NAND gates make it so that when the pull-down resistor takes all 9V the capacitor can charge while not turning on the LED and when the pull-down resistor doesn't take all 9V the opposite occurs for a momentary amount of time. The diode plays the role of only allowing the current to flow to the ground. The 10µF capacitor outputs current after the PBNO is realized through the 1 MΩ resistor if the capacitor's farad level was increased that would increase the time the LED stayed on after the PBNO was turned off. The 10 kΩ resistor is a pull-down resistor that allows the state of the circuit to be known when the PBNO is not pressed, this state is 0V. The PBNO gives the circuit two inputs one when pressed and zero when the pull-down resistor takes the 9V. Finally, the 1 MΩ resistor acts as a path to ground when the capacitor outputs the 8V it has then as it slowly goes through the 1 MΩ resistor it also gives the NAND gate a high input until the NAND gate is no longer receives more than 4.5V.

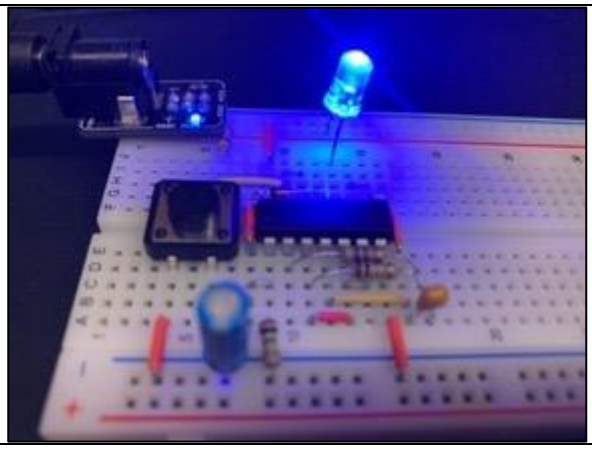
The parts used to create the circuit include all parts from the analog input to give the circuit a fixed duration. The additional parts add to the rises and falls. Firstly, the 4011 Chip uses an additional three gates in this circuit. The three gates give the 0.1µF capacitor the function of continuously charging and discharging to give the NAND gate oscillator the function of a flashing LED. The 1 MΩ resistor makes it possible for the charges and discharges by the 0.1µF capacitor able to be seen by the human eye. The 0.1µF as already described plays the role of continuous charging and discharging as seen in the image. The image on the right also shows how this constant charge and discharge with rises and falls is a clock signal. The clock signal has the oscillation between each rise and fall so when units are added to this circuit every number displayed will be displayed for an equal amount of time.

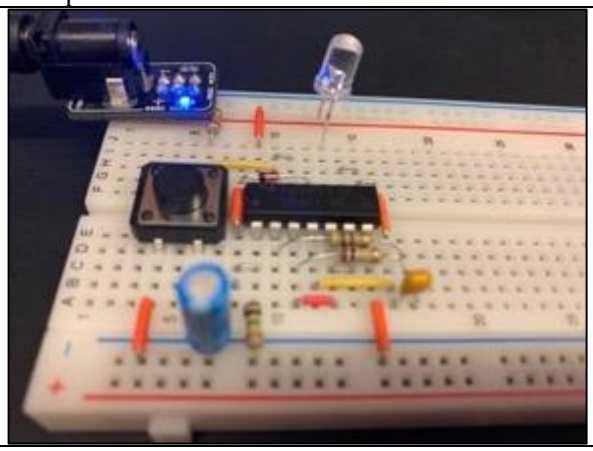


Media



Before PBNO input





After PBNO

C. Decade Counter

Purpose

The purpose of the decade counter is to count the rising edges of the NAND Gate Oscillator through the 4017-logic gate and displayed through the 10 Segment Bar Graph Light Display.

Reference

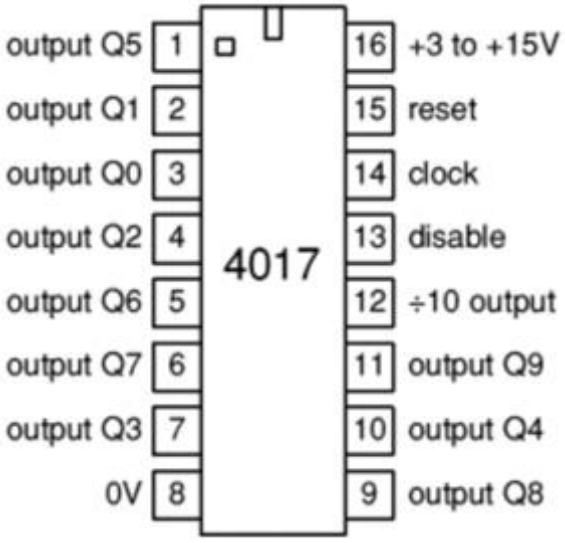
Project Description: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#counting

Procedure

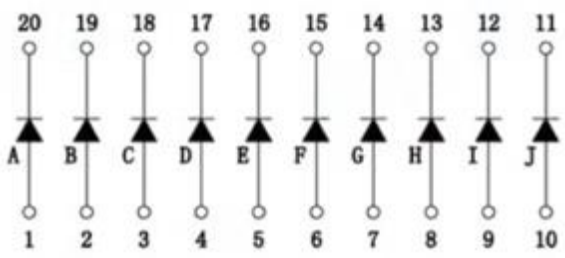
Parts Table	
Quantity	**Description**
2	1 MΩ Resistor
1	Diode
1	10µF Capacitor
1	10 kΩ Resistors
1	PBNO
1	0.1µF Capacitor
1	470 Ω Resistor
1	4011 CMOS IC Logic Chip

The decade counter takes the NAND gate and begins to add the sequential circuit added to the circuit. The way this can be done with the addition of two new parts. One is a logic gate that counts the rises from the NAND gate. Two the Bar Graph Light Display is shown to display these rising edges continuously increasing.

The first thing to deeply look into is the 4017 Logic Chip. The 4017 Logic Chip is a logic chip designed around the clock input needed for this circuit to function. There are 16 pins for the 4017 chip and each plays a significant role in completing the clock function. The pins used to have the 4017-chip act as a decade counter with the main function of counting the clock pulses.

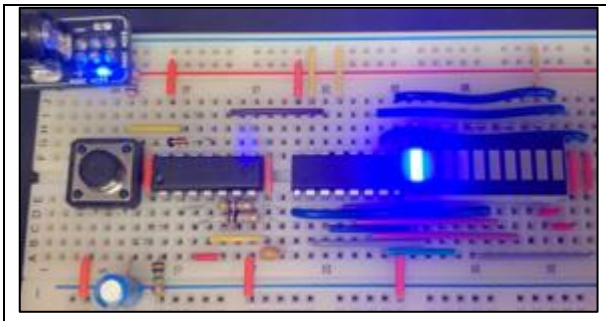


As seen on the pinout diagram on the right the 4017 chip has pins 16 and 8 to power the chip to complete all functions needed. Next, the pin used to make the entire circuit function correctly is pin 14 which is connected to the NAND gate oscillator. The way the clock is working is by counting every rising edge the NAND gate oscillator produces. Pin 14 has the function of counting all of those rising edges and producing the correct outputs. The reset pin, pin 15, in this circuit when the objective is to count to 9 before resetting by connecting pin 15 to the ground. The disable pin is connected to pin 13 and is only relevant in the circuit if the pin goes to a high value, it disables the clock from receiving new clock pulses. Lastly, the 4017 chip has 10 output pins that function in a sequential form. When the clock pin has received zero clock pulses the output Q0 is output and as more pulses are added the output pins change to higher values.

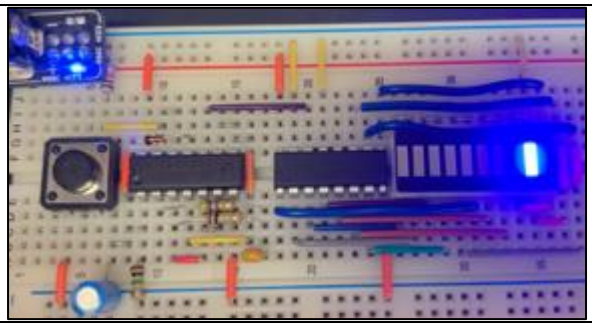


The other new component used in the decade counter is the display used to show the function of the 4017-logic chip. The Bar Graph Display is 10 LEDs that display what outputs are coming out of the 4017 gates. When building the Decade Counter seen in the images below the wire connections function so that Q0 is connected to pin one on the bar graph display, Q1 is connected to pin 2 and so forth. When the button input is pressed and the bar graph continuously increases this shows that the 4017 functions in a clock method and that the decade counter was wired correctly.

Media



4017 chip receiving zero pulses



4017 chip receiving nine pulses

D. Decimal Counting Up/Down Counter

Purpose

The purpose of the Decimal Counting Up/Down Counter is to display how a 4516 Logic Chip can take the outputs from the 4017 Logic gate to count 4-4 bit shown through four LEDs.

Reference

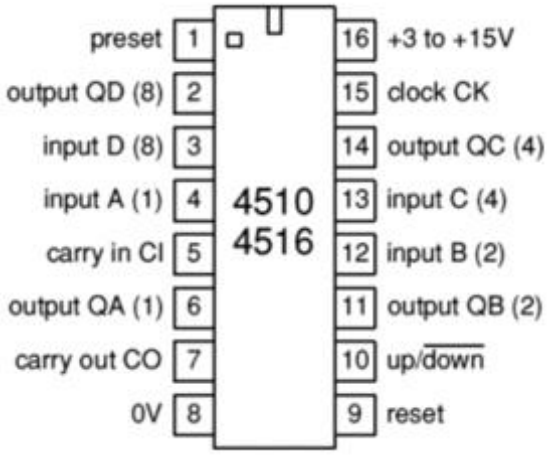
Project Description: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#counting

Procedure

The procedure for the Decimal Counting Up/Down Counter begins with understanding how the 4516 takes the outputs from the decade counter and creates a 4-bit counter from the 4516 Logic Chip. The two new parts used in the Decimal Counting Up/Down Counter are the four LEDs and the 4516/4510 Logic Chip. The major change between the decade counter and the decimal counting up/down counter is the addition of the 4516 Logic Chip. allowing the configuration to count 4 bits in this circumstance. The 4510/4516 plays the major role of counting the pulses in a base of 4-bits, or 15 in base 10, and the value 10. Without this chip the counter has no maximum when operating with only one digit making this chip essential to the operation.

Parts Table	
Quantity	**Description**
4	Blue LEDs
2	1 MΩ Resistor
1	10µF Capacitor
1	10 kΩ Resistors
1	PBNO
1	Diode
1	470 Ω Resistor
1	0.1µF Capacitor
1	4011 CMOS IC Logic Chip
1	4510 CMOS IC Logic Chip
1	4017 CMOS IC Logic Chip
1	SPDT Switch

Firstly, as the pinout on the right shows, two pins are connected to power and ground to give power to the circuit. The circuit pin 15 received input from pin 12 on the 4017 chip which sends a high input after every 10 rising edges through the 4017 chip. The tenth pin "up/down" is used to show if the circuit should be counting up or down. When it receives a high input, the circuit counts up and does the opposite for a low input. The present, reset and carry-in pins all have functions that for this circuit all needed to be powered directly to the ground. The output pins are the most important part of the 4516 and 4510 Chips. When working with the 4516 the four outputs allow the chip to give a maximum of 4-bit binary value when all outputs are high the chip outputs the value of 15 and turns on all four LEDs.

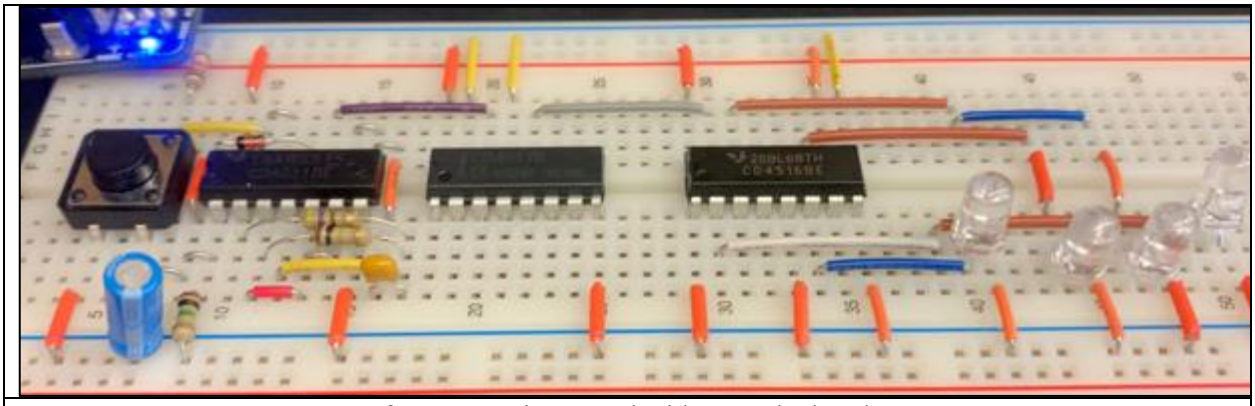


Going deeper into the chip outputs the output from the chip consists of four binary outputs, labelled Q0, Q1, Q2, and Q3. These outputs represent the current count value of the counter, which can be either in a high state, logic level 1, or a low state, logic level 0. The counter sections in the 4510 chip are capable of counting from 0 to 9 in the BCD format, and the Q0, Q1, Q2, and Q3 outputs reflect the binary representation of the count value. The 4510 chip has additional control inputs, such as clock inputs, enable inputs, and a direction control input, which determines the behaviour and operation of the counter, allowing it to be controlled externally.

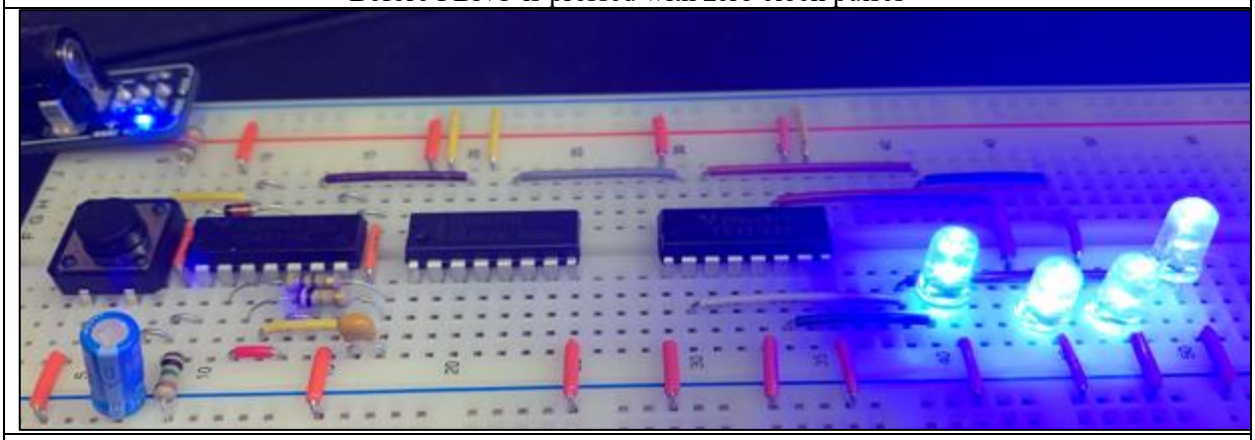
The four LEDs in the circuit play the role of displaying how the 4516 can function. The difference between the 4510 and the 4516 is that one is based around functioning for a 4-bit counter to 15 while the other is counting from 0-9 for a seven-segment counter. The three LEDs display that there are 4 bits with the least value being zero with all LEDs off and when all are on the 4 binary values add up to 15.

Binary	Decimal
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

Media



Before PBNO is pressed with zero clock pulses



After PBNO is pressed with nine clock pulses

E. Binary Counting Decimal Decoder

Purpose

The purpose of the Binary Counting Decimal Decoder is to take the four outputs for the 4510 Logic chip and create seven simple outputs.

Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#counting

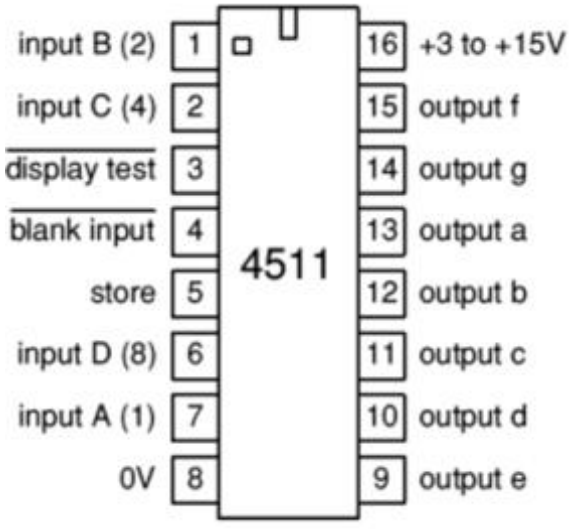
Procedure

The procedure for the Binary Counting Decimal Decoder begins with the addition of one new part. This new part plays a major role in making it possible for the counting of the 4510 Logic Chip to be displayed through seven outputs from the 4511 chip. The way this function is by adding to the count of this circuit is with the 4011 chip's ability to continuously rising edges to the 4017 chip's ability to count these rising edges and output them to the 4510 chip when those outputs are counted in base ten to finally the 4511 being able to take those outputs and control them into just seven.

Parts Table	
Quantity	**Description**
4	Blue LEDs
1	Diode
1	10µF Capacitor
1	10 kΩ Resistors
1	PBNO
1	1 MΩ Resistor
1	470 Ω Resistor
1	0.1µF Capacitor
1	4011 CMOS Logic IC Chip
1	4510 CMOS IC Logic Chip
1	4017 CMOS IC Logic Chip
1	4511 CMOS IC Logic Chip
1	SPDT Switch

The function of the 4511 is very complicated with several different pins playing different roles in the function. The pinout shown consists of six sections that all play roles in the circuit's function.

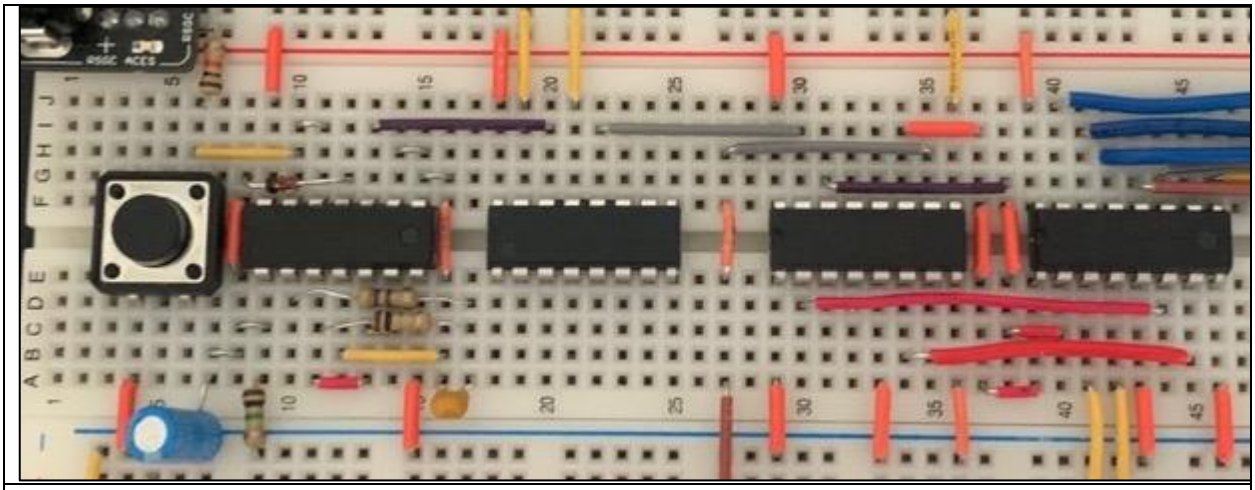
The sections include the power and ground rails, input pins, the store pin, blank input and display test and finally outputs. Firstly, looking at pins 16 and 8 it is shown that these pins power how all other pins function. Secondly, the input pins are designed to work with a BCD (Binary Coded Decimal) the four inputs can receive 4-bits but when working with the seven-segment display the max input will be the fourth bit and the first bit adding up to a value of nine. Next looking at pin 5 which is the store pin, in the case of making the counting circuit should be connected to the ground giving a low input. If the store pin is connected to power, it stores the current output and keeps it constant no matter the changes made to the inputs. The blank input and display test are active low which means it only turns on when the input is low. When the input is high on the blank input all outputs go to a low and the display will go blank. When the display test is high all the outputs will go to high no matter the inputs. Lastly, the outputs are designed to take the four inputs and create a display that works with the seven-segment display.



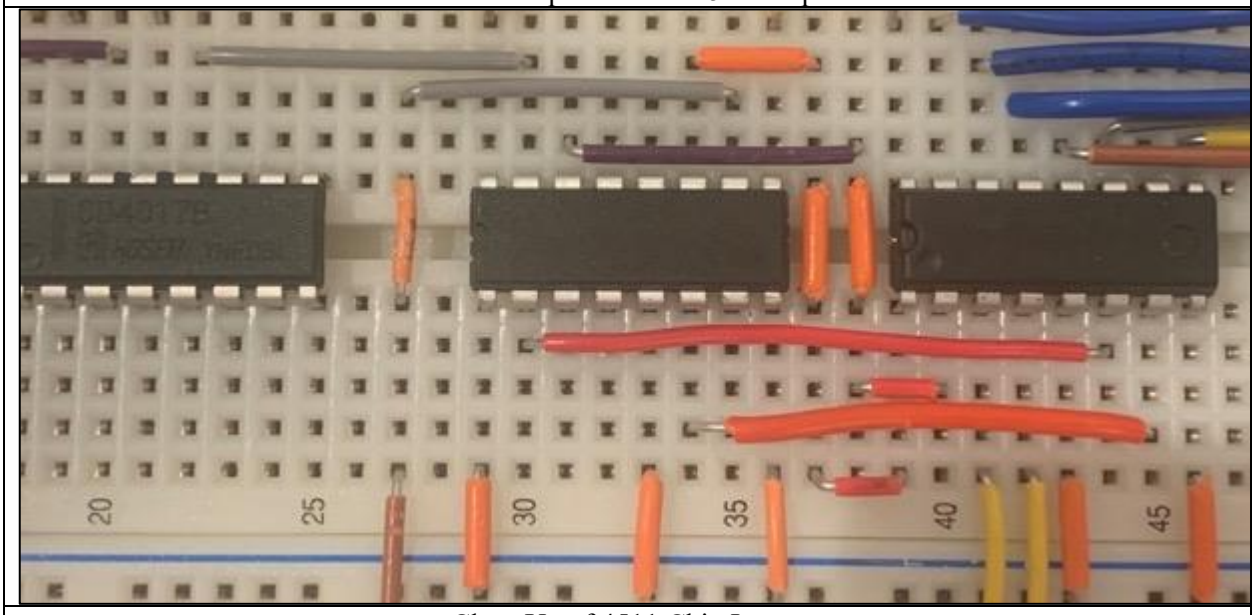
In a BCD system, each decimal digit from 0 to 9 is represented by its four-bit binary code. The four bits in BCD can take on values from 0000 to 1001. To represent a decimal number in BCD, each decimal digit is converted into its corresponding four-bit binary code and arranged in sequence. BCD is a direct representation of decimal digits, making it easy to convert between BCD and decimal.

BCD	Decimal	BCD	Decimal
0000	0	0101	5
0001	1	0110	6
0010	2	0111	7
0011	3	1000	8
0100	4	1001	9

Media



Circuit Updated with 4511 Chip



Close Up of 4511 Chip Inputs

F. Seven-Segment Display

Purpose

The purpose of the Seven Segment display is to display the outputs provided by the 4511 Logic chip in a zero to nine counter.

Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#counting
Electronics Tutorial: https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html

Procedure

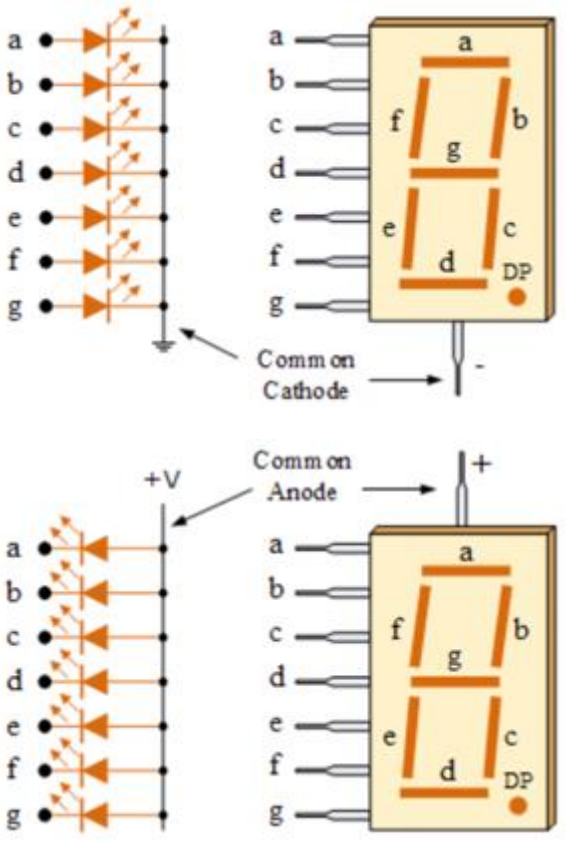
The additional component used in this stage of the counting circuit is the seven-segment display this is built on receiving 7 inputs to display numbers 0-9. Elaborating, the function of the seven-segment display is to receive the outputs from the 4511-chip ranging from a to g. These outputs go through the display to specific LEDs on the display and show the inputs the chip has been given to display a number from 0 to 9.

Parts Table	
Quantity	**Description**
7	630 Ω Resistor
2	1 MΩ Resistor
1	10 µF Capacitor
1	10 kΩ Resistors
1	PBNO
1	Diode
1	470 Ω Resistor
1	0.1µF Capacitor
1	4011(NAND) Chip
1	4516/4510 Logic Chip
1	4017 Logic Chip
1	4511 Logic Chip
1	Seven-Segment Display
1	SPDT Switch

The seven-segment display can be configured in two different ways. The first is when the "com" pin is connected to ground providing a common cathode for all LEDs in the display. The other is when the "com" pin is connected to power providing a common anode to all LEDs.

Firstly, look at the configuration on the common cathode display. The configuration has all LEDs connecting to a common cathode that has the value of 0 or "low". The way the counting circuit function when the seven-segment display has the common cathode display the series will function with the 4011 creating the rising edges and the 4017 counting the rising edges to the 4510 making to counts into base 10 counting from 0-9 to finally going through the 4511 which in this case will send inputs to each of the LEDs which has voltage drop through the common cathode.

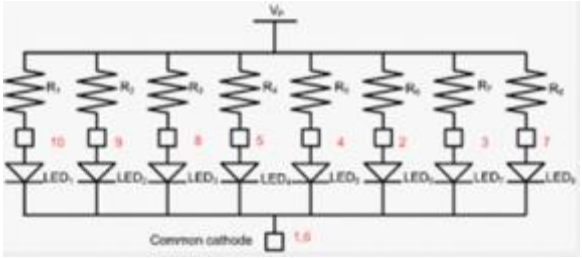
The second of configuration that can be used by the seven-segment display is the common anode configuration. This configuration has the "com" pin connected to power with the LEDs using the common anode as the high voltage and waiting for a voltage drop instead of the common anode where the LEDs have a voltage drop and are waiting for a voltage high. When working with the counting circuit the common Anode display is not used based this is based on the fact that when using the common anode, the seven-segment display is looking for a low signal for a LED to light up



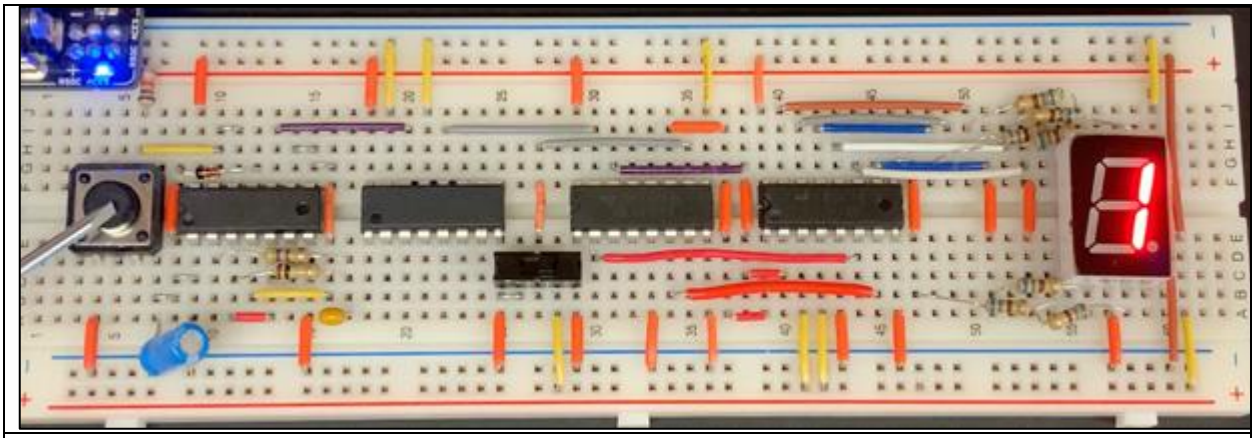
The configuration of the seven-segment display is built around the configuration of the 4511-logic chip making it simple to wire each output of the chip to the display as the outputs are labelled with the same pins in the seven-segment display ranging from a to g. The table on the right show how the inputs function when connecting the 4511-logic chip to the seven-segment display. The table shows what outputs of the chip need to be high and what outputs need to be low to display decimal numbers

s	BCD	G	F	E	D	C	B	A
0	0000	0	1	1	1	1	1	1
1	0001	0	0	0	0	1	1	0
2	0010	1	0	1	1	0	1	1
3	0011	1	0	0	1	1	1	1
4	0100	1	1	0	0	1	1	0
5	0101	1	1	0	1	1	0	1
6	0110	1	1	1	1	1	0	1
7	0111	0	0	0	0	1	1	1
8	1000	1	1	1	1	1	1	1
9	1001	1	1	0	1	1	1	1

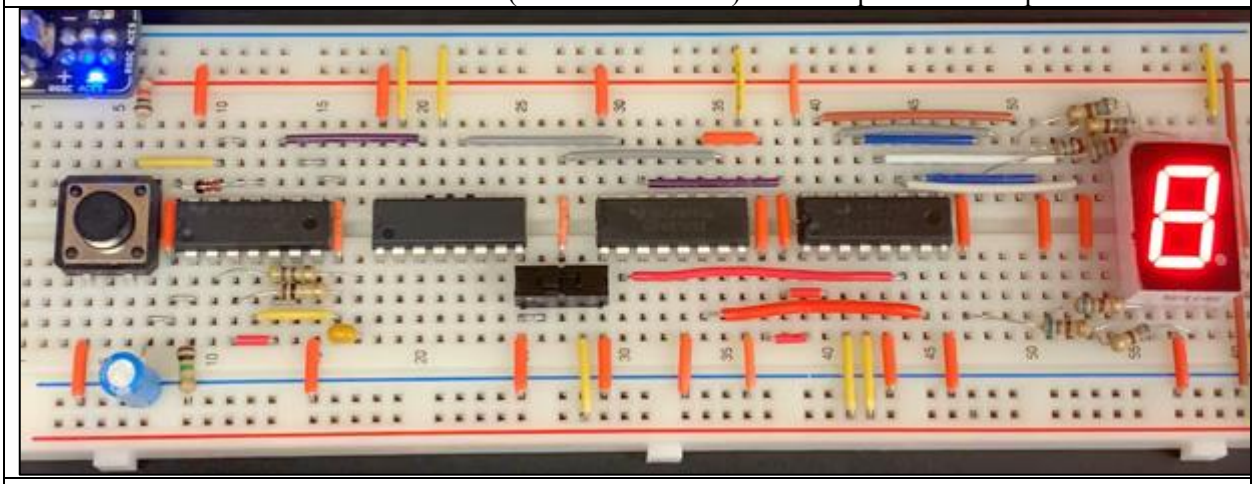
The image on the right shows how the seven-segment display looks as a schematic. When looking at the schematic it shows how each LED function on the seven-segment display each receive its own high input. With the resistor the LED would take the entire high input from the 4511 chip. The 4511 chip is connected to a 9V power rail meaning it would take all 9Vs when it can only take 2V so with a resistor it can take only 2 and the resistor will happily use the other 7V.



Media



Value of 1 "com" connected to CC (Common Cathode) SPDT to power from Up/Down Pin



Value of 1 "com" connected to CC (Common cathode) SPDT to ground from Up/Down Pin

G. A Counting Circuit PCB

Purpose

The purpose of the A Counting Circuit PCB is to integrate the circuity of the breadboard Counting circuit into a permanent form soldered to the board.

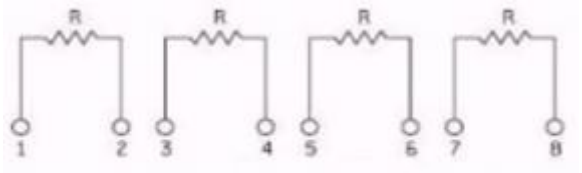
Reference

Project Description: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#counting

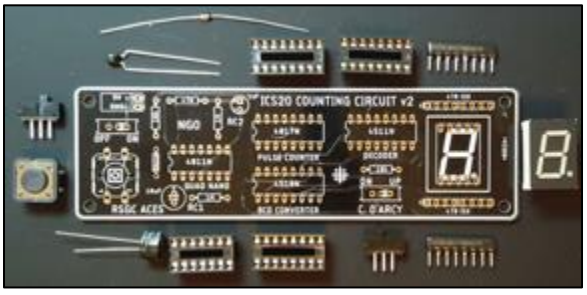
Procedure

The procedure of A Counting Circuit PCB begins with the introduction to several new parts. the parts table shows the additional pieces and the chips that are placed in the PCB. The circuit is built with several different parts and inputs. The first piece being the 2-pos 2.54 mm Term Block that provides power to the entire circuit. Next is the first of two SPDT switches that acts as a gate for power to be provided to the circuit or not. Then, the PBNO when pressed begins the momentary period where the 100 µF Electrolytic Capacitor is charged. Going through the 14-pin IC Socket holding the 4011 NAND chip through the 0.1 µF Disk Capacitor created continuous rising edges. These pulses are sent to a 16-pin IC Socket holding the 4017 Chip that counts the pulse. Those pulses are sent to another 16-pin IC Socket holding the 4510 Chip and connected to another SPDT switch controlling if the counter goes up or two. The pulses are configured into base ten and sent to the 4511 chip where the seven outputs each go through 1 kΩ Isolated resistors network to the seven segment display creating the a counting circuit on PCB.

Parts Table	
Quantity	**Description**
2	16-pin Chip Seat
2	1 MΩ Resistor
2	1 kΩ Isolated Network Resistor
2	10 kΩ Resistor
2	1 MΩ Resistor
2	(SPDT) Switch
1	4011 CMOS Logic Chip
1	4511 CMOS Logic Chip
1	4510 CMOS Logic Chip
1	4017 CMOS Logic Chip
1	14-pin Chip Seat
1	470 Ω Resistor
1	PBNO Button
1	Seven Segment Display
1	0.1 µF Disk Capacitor
1	100 µF Electrolytic Capacitor
1	Custom Counting Circuit PCB



One significant change made when using the counting circuit on the PCB compared to the breadboard is the 1 kΩ Isolated Resistor Network. These resistors allow for the PCB to be simpler and more compact. The way this resistor functions, when it is an isolated version, is having several resistors in one piece with each 1 kΩ resistor having an input from the 4511and output to the seven-segment display.



When soldering all pieces to the PCB the primary pieces to begin with include the tight to board pieces like diodes and resistors, then pieces larger that are still tight to the board like PBNO's and the chip seats which has the function of allowing a pathway of current to flow from the PCB to all IC Chips without burning the chips upon soldering to the board. After that with all practice to other pieces on the PCB solder the SPDT switch, 1 kΩ Isolated Resistor Networks and the seven-segment display, doing these last is best considering the have the highest chance of shorting unless done correctly and the seven-segment display has an easy chance of overheating.

Media



A Counting Circuit SPDT Switch going to PCB



A Counting Circuit SPDT Switch going to Ground

H. A Counting Circuit Case

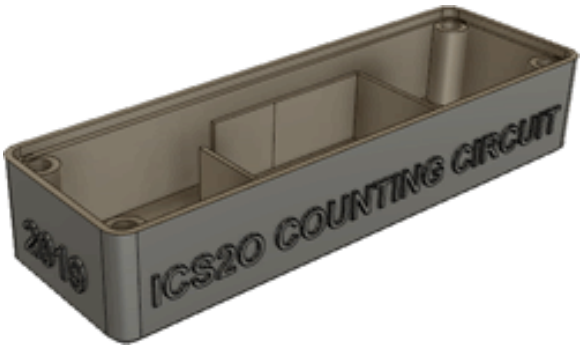
Purpose

The purpose of A Counting Circuit Case is to provide a battery holder to for the PCB along with a protector from any future damages.

Reference

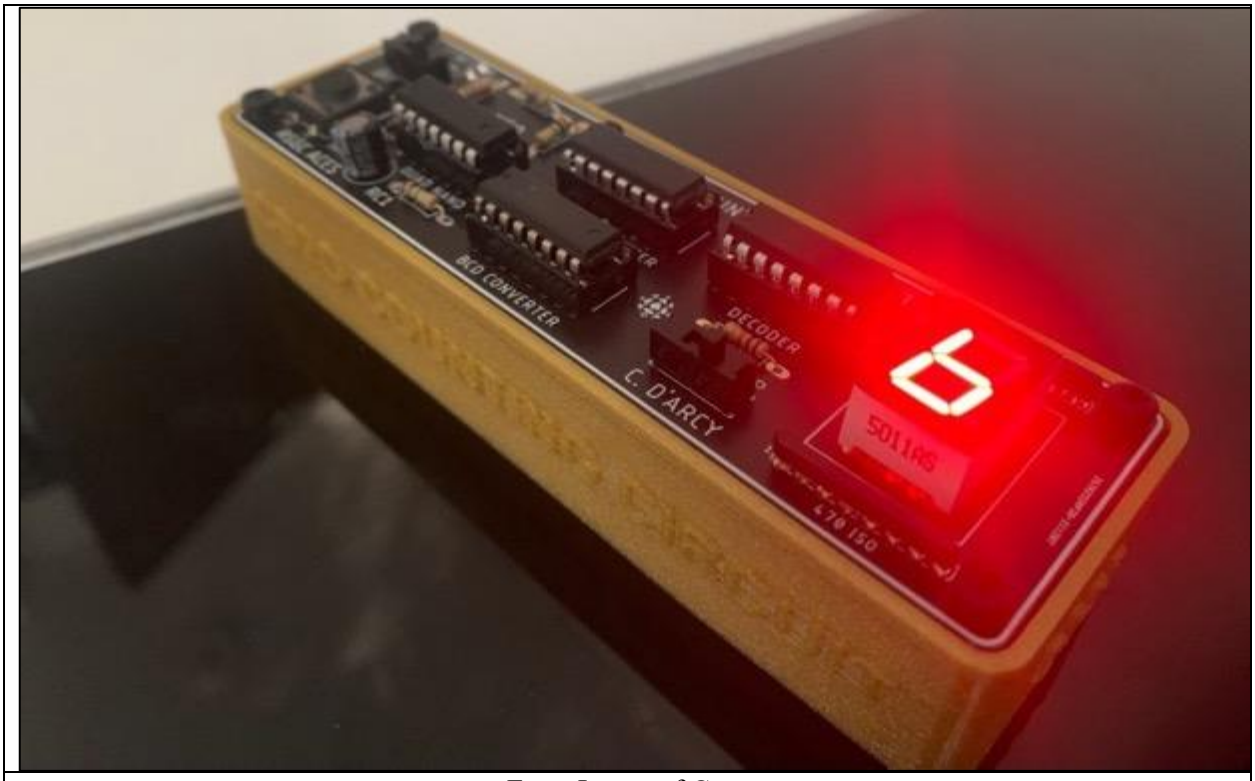
Project Reference: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#counting

Procedure



The case is a piece given when the PCB is completed and successfully functions by having the display count forward and backwards. The case is reward for making it to the final part of the counting circuit marathon. The case provided was designed and printed by 2020 ACES graduate J. Dolgin having four screw sockets to hold the PCB by screws and a battery holder than is mounted down by a piece of electrical tape. This electrical tape has the function of not shorting the circuit

Media



From Image of Case



A Counting Circuit Case Bottom



A Counting Circuit Case Top

I. A Two Digit Counting Circuit Display

Purpose

The purpose of A Two Digit Counting Circuit Display is to incorporate the function of a circuit that can count pulses to another seven-segment display done through the addition of a 4511 IC Chip and 4510 IC Chip.

Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEL3M/Tasks.html#counting

Procedure

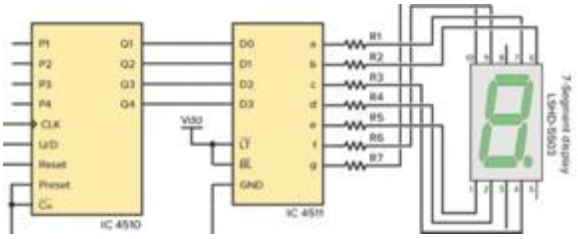
The procedure for A Two Digit Counting Display begins with the addition parts and how they function. The two additional parts are displayed on the bread board are the new 4510 IC Logic Chip and the 4511 IC Logic IC Logic Chip. The addition of the two new chips backpacks of the function of the original one gate.

Parts Table	
Quantity	**Description**
14	630 Ω Resistor
2	1 MΩ Resistor
2	4511 Logic Chip
2	Seven-Segment Display
2	SPDT Switch
2	4510 CMOS Logic Chip
1	470 Ω Resistor
1	0.1µF Capacitor
1	4011(NAND) Chip
1	4017 Logic Chip
1	10µF Capacitor
1	10 kΩ Resistors
1	PBNO
1	Diode

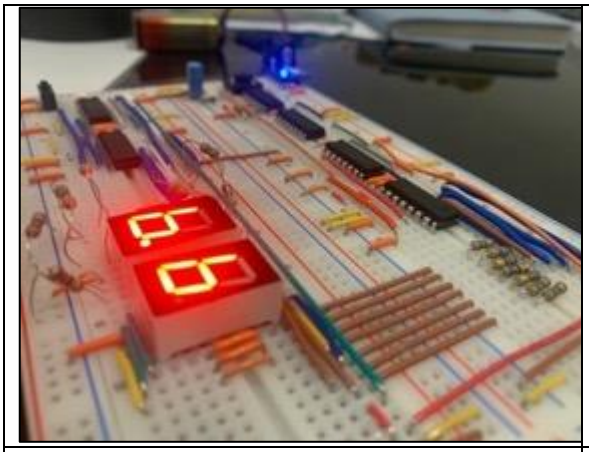
Firstly, looking back at the theory of the counting circuit it begins with the idea that the NAND gate oscillator is producing constant rising pulses. These rising pulses produce the inputs that are counted by the 4017 and then outputted by a divided by 10 output.

The different between the one- and two-digit counting circuit display is that the second is function off of every other output that isn't the divided by 10 output. This is known has the carry out and can be connected to a carry in on another chip to make it function. When the carry ins are plugged in it allows both 4510's to correctly transfer the pulses.

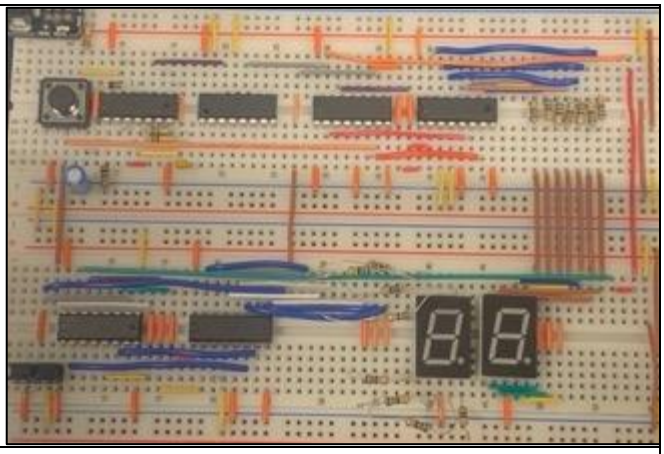
After that all is left is setting up the wiring from the 4510 to 4511 to another seven-segment display. To make the counter look better as a 2-digit counter shifting the first seven-segment looks best. After doing that all that is left is to wire everything correctly.



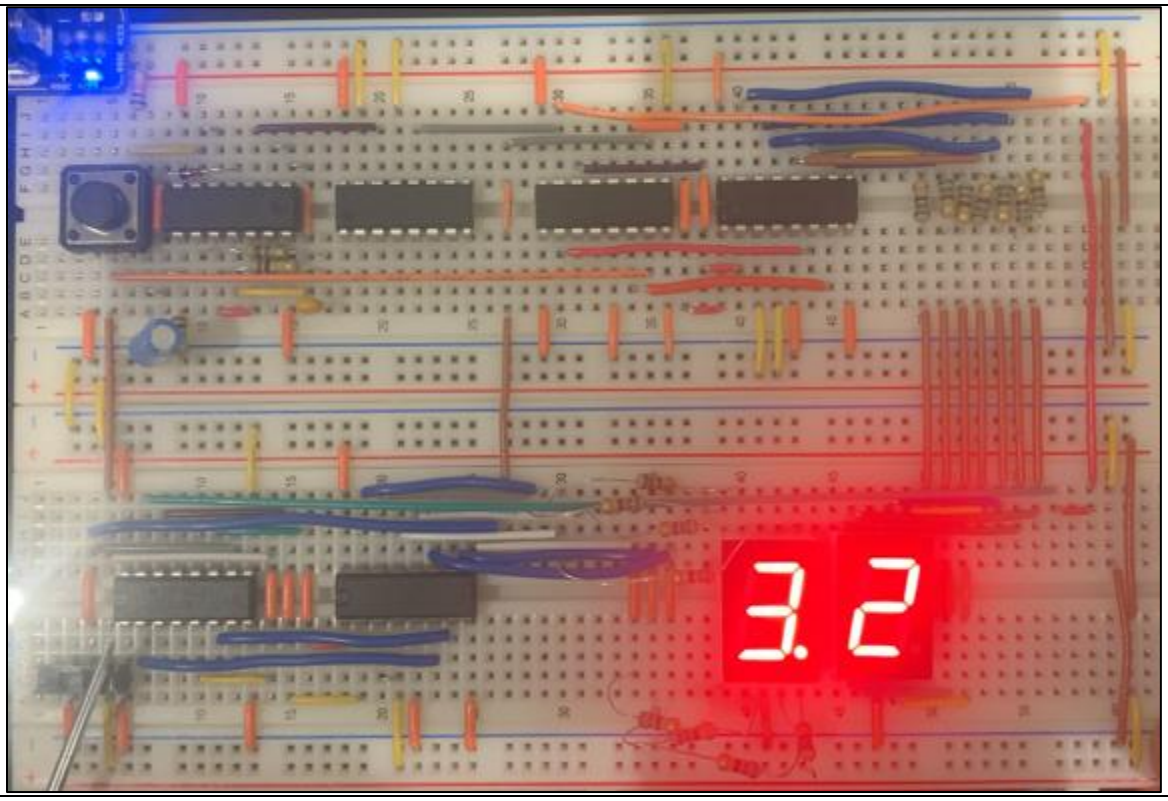
Media



Front Eye View Image



Bird Eye View Image Off



Bird Eye View Image On

Media

Project Video: https://youtu.be/UQEjwV1EUXA

Reflection

After just finishing the counting circuit DER and video I have a lot of emotions from the exhaustion of hours of research on every part of this circuit to the excitement that I can look back on this DER document and see my hard work looking amazing. As I said in my first DER, I think one thing this course teaches as much if not more than engineering is independence and exposure to the real world.

ICS3U-E

Project 2.1 Digital-to-Analog Conversion

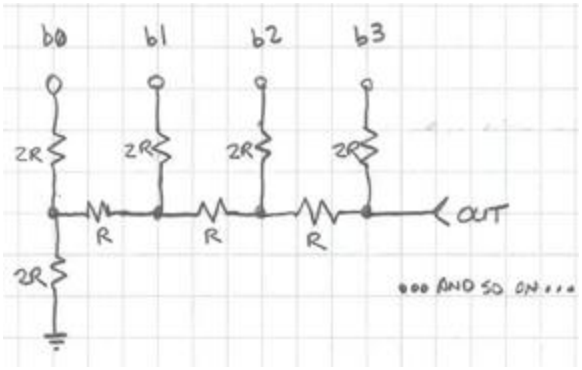
Purpose

The purpose of the Digital-to-Analog Conversion is to show how a power source and a resistor configuration can convert digital inputs to analog along with displaying *Thevenin's equivalent circuits* and *Superpositioning*.

Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/Tasks.html#R2R
Video Explanation: http://www.youtube.com/embed/AulX1OM7RwE

Theory



The theory behind the DAC (Digital-to-Analog Conversion) follows two principles **Thevenin's Equivalent Circuits** and **Superposition**. Thevenin's Equivalent Circuits states that if a circuit is built of only linear electrical devices, voltage sources, resistance sources and current sources, then it can all be simplified to one power source and one resistor. This allows the R/2R circuit to follow the rule that the output resistance from the circuit is always equal to "R" no matter how many bits are added. Superposition is the principle used when calculating how the bits each have a different weighted voltage. The way superpositioning is done in the DAC is by shorting all voltage sources/bits except the bit in which voltage is being calculated once all bits are calculated the answer will show a continuous binary increase in the fraction of voltage used by each weighted bit.

Procedure

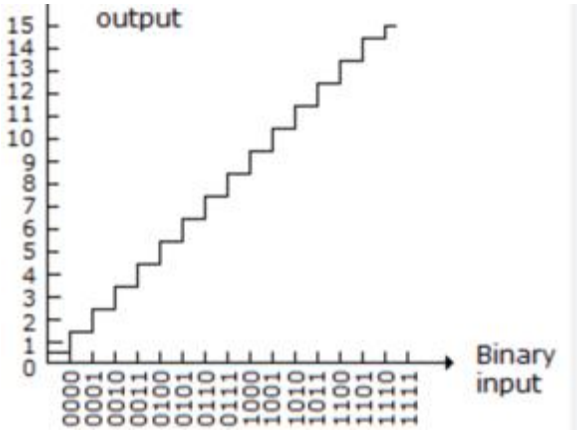
Parts Table	
Quantity	Description
9	10.2k Ω Resistor
7	5.1k Ω Resistor
8	Slide Switches
1	5V DC Power Supply

The Digital to Analog conversion has a specific configuration that makes it possible to turn digital inputs into analog outputs. This can be done by slide switches each having an input that goes through a different resistor configuration for each input. The different resistor configurations convert the inputs to analog. This configuration can convert digital to analog using Thevenin's Equivalent Circuits and the Superposition principle, as spoken about in the theory section.

Bit	Contrib
0	1/16
1	2/16
2	4/16
3	8/16

The first component to look into is the resistors used to make it possible for this circuit to turn digital inputs analog is the resistors. Firstly, these resistors have a premium that allows them to have a resistance within a 1% accuracy of what is labelled unlike usual resistors used in this course that offer only a 5% accuracy. For this circuit, the only values are R and 2R which is the 5.1k Ω and 10.2k Ω. The resistors have a key role in this circuit because the resistors create the weighted binary inputs needed for a DAC. The circuit is constructed so that bits further away from V_{out}, the location where the voltage is measured, have more impedance the further away the switch is from V_{out}. The image on the right displays how the weighted binary voltage would work with a 4-bit digital-to-analog conversion.

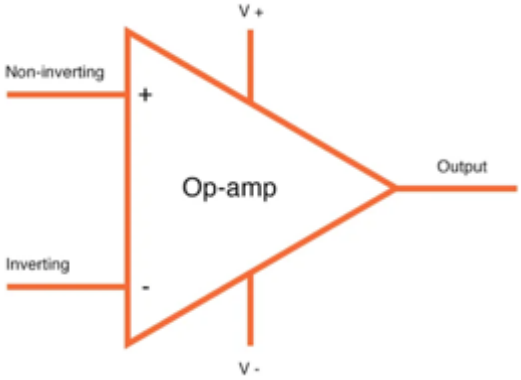
The other aspect of the circuit is the digital aspect of the slide switch. The slide switch is designed to have two inputs to one path. In this case, the switches create the two paths of 5V (Voltage Reference) or ground to give the binary inputs of 0 and 1 that give the digital aspect of this circuit. The key thing to note about the DAC is that the more binary inputs in the circuit the less significant the least significant bit will get. What this means is that the more bits in the circuit the more voltage levels there are between 0 and 1. This is a 4-bit DAC programmed into an Arduino to give all 16 different voltage levels achievable with a 4-bit DAC.



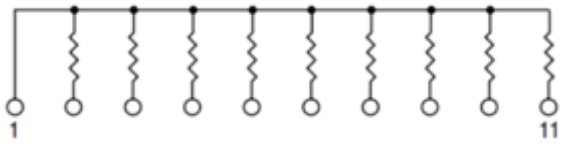
The DAC is a circuit that is built on Theorems that can simplify the circuit when understood. Additional steps can be taken to make the DAC a little more advanced and as well as gain information on some other parts that can be helpful in future DERs. These parts just like the DAC are confusing at first look but with gained knowledge become simple parts that can have extremely helpful aspects to the DAC.

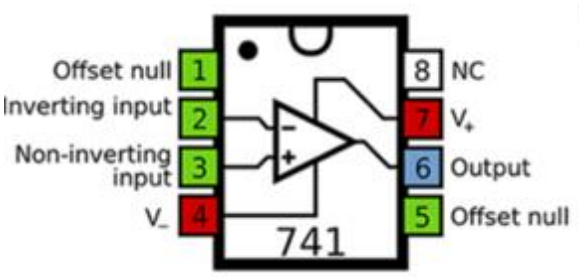
Additional Parts Table	
Quantity	Description
1	9-Pin Bussed Resistor Network
1	R/2R Series Network
1	Switch Rocker DIP SPDT
1	LM741 Op Amp Chip

One additional key electrical piece used in the DAC is the *Operational Amplifier*. The Operational Amplifier or Op Amp for short is for this circuit a logic chip following two rules. The first rule being followed by the Op Amp is that no current flows in or out of the inputs shown in the image on the right. What this means is that because of the large impedance, the input's only function is to monitor the voltage based on the electrical signal being inputted. In the case of the DAC one of the rules is that the output resistance is always equal to R and for this circuit R = 5.1k Ω meaning without an Op Amp the 0V to 5V going through a 5.1k Ω resistor results in a low current. For the addition of an LED display to see all the states that the DAC could create the LED needs a lot less resistance to show values of 0.3125V on an LED. This is where the Op Amp comes in, the Op Amp monitors the voltage and through the power source, the chip is directly connected to output the voltage from input without any resistance.



Another configuration of this circuit can be done with the switch rocker DIP SPDT, the R/2R series network and the 9-pin *bussed* resistor network. The photo in the media below shows how this configuration would work in a circuit. As seen the configuration has a 10-pin R/2R series network with one pin going to ground and the other going to an analog output that converted the 8 bits connected through the rocker switch. The actual resistance is coming from a 9-pin bussed resistor network meaning that there are 8 resistors each of 10k Ω that are all connected to one source of ground. The schematic for an example of an 11-pin bussed resistor is shown on the right.

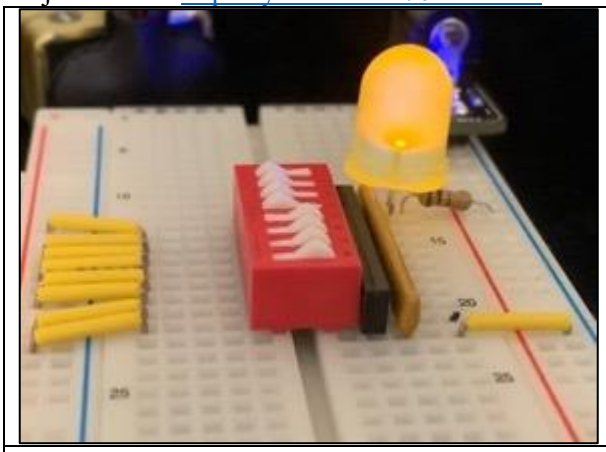




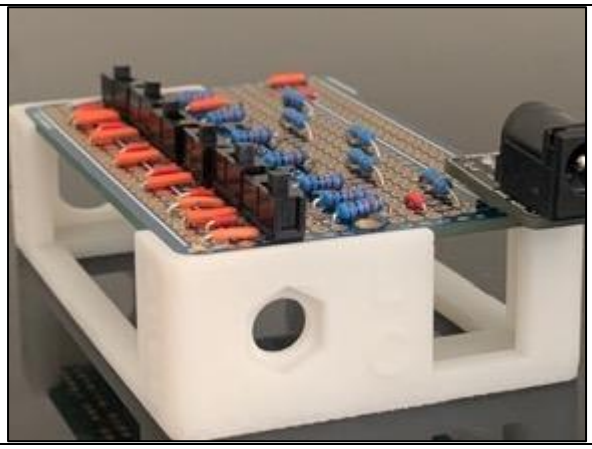
The LM741 does not have a rail-to-rail range from V+ to V- it has a range with about 1V less than the exact amount. A fixture to this issue is to make the range greater than 0V to 5V. How to do this is to make negative voltage and another plug of 9V from another battery. This can be done when the voltage has to flow in the opposite direction. As known voltage flows from high to low but if the voltage has to flow in the opposite direction, then the voltage becomes negative. The configuration on the right shows the layout of the 741 chip, pin 4 has -9V and pin 7 has +9V.

Media

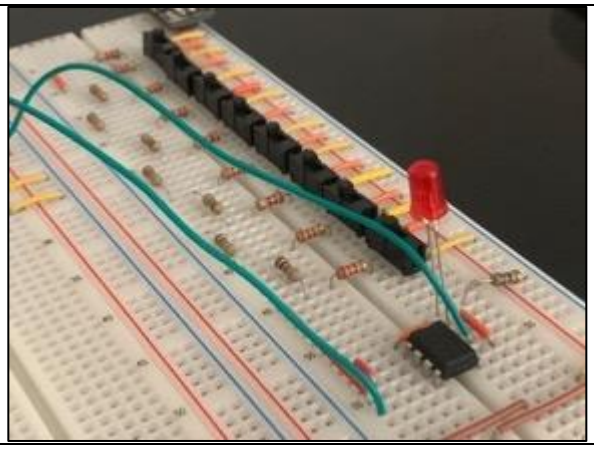
Project Video: https://youtu.be/atf70sxOCwY



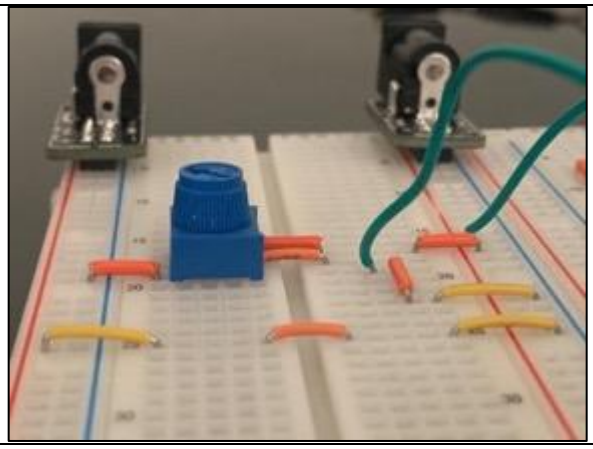
8-bit Switch Rocker R/2R Resistor Network



6-bit Perma-Proto R/2R Ladder 1% Resistors



8-bit Breadboard Resistor Network



Negative Voltage Input Breadboard Close-Up

Reflection

This DER saw a big new step for me and the rest of my class. It was our first DER of grade 11. One of my favourite parts about it was being surrounded by people who loved engineering as much as I did. Having great resources from everyone in my class was extremely helpful. The actual DER has been my second favourite of all my DERs I loved understanding what at first looked like an extremely confusing due to the simplicity. It has also inspired me to think of some ISP project around the idea of a speaker which uses the DAC principles.

Project 2.2 The 555 Time Machine

Purpose

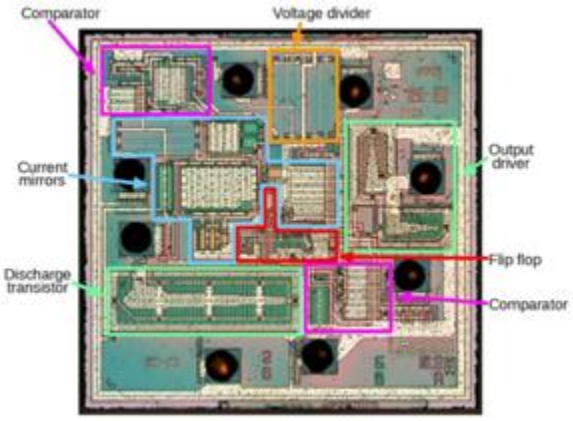
The purpose of the 555 Time Machine is to show how the 555 Chip functions and most importantly in astable mode where it can output square waves while additionally showing how an SR latch can function.

Reference

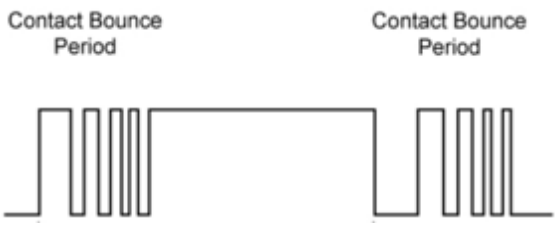
Project Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/Tasks.html#555
555 Timer Explanation: https://youtu.be/kRlSFm519Bo?si=q2mIdTBFjcxUWDZT
SR Latch Explanation: https://youtu.be/KM0DdEaY5sY?si=iZhv_v2ZET4HA-Ru

Theory

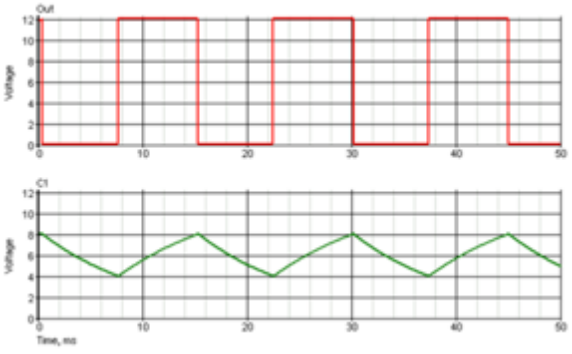
The 555 Timer has various functions. The chip has three noteworthy modes: monostable mode, astable mode, and oscillation mode. This chip can at first look complicated but after further inspection, it can be understood what is going on inside the circuit and how the external parts function as well. The pinout shown in the procedure shows how specific pins need specific external circuitry. Two states the 555 can provide are monostable and astable modes. As further discussed in procedure the 555 can be adapted. This adaptability is what differs from the three modes discussed. This image on the right shows a cross-section of the IC and the subsections of the chip.

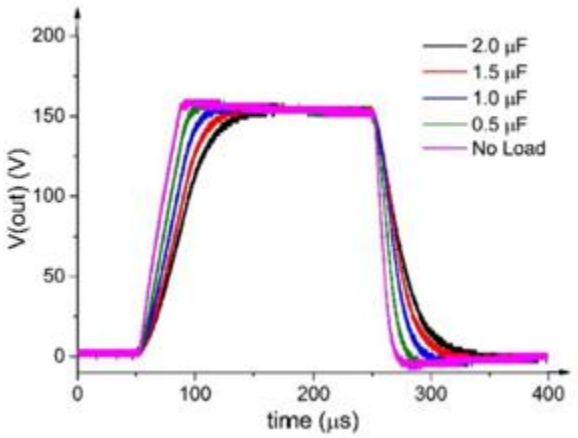


The first state to be discussed is the monostable state. The monostable mode is used to provide a one-shot pulse meaning it produces a single high signal only when a manual input is inputted. The monostable mode is an extremely useful mode used in circuits like time-delay circuits and pulse shaping. One other application that is very significant is button inputs with no bounce. At its core, a button is two metal contacts connecting. When these pieces connect, there are almost always small intervals where the current begins to flow and then stops flowing. This means a one click can produce many smaller pulses. This can become a big flaw that can be solved when you include a 555. The 555's circuitry allows for that to happen but the output because going through the Op-Amps allows for an output of one without and bounce.

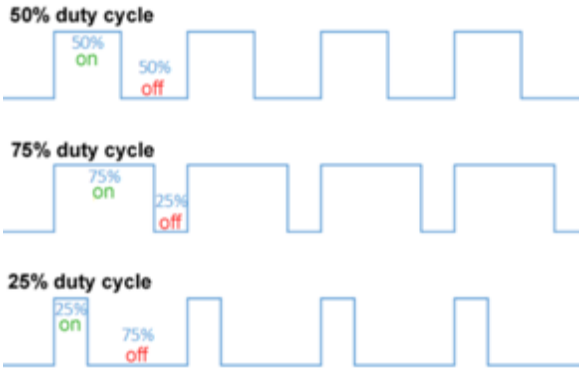


The other mode worth noting is the oscillation mode. The oscillation mode produces constant square waves that output a constant 0 and 1. This chip compared to other oscillators is extremely easy to use. This is the most common use of the 555 because of its non-complicated wiring, and simplicity. This can be used to create LED flashers, and square waves with different frequencies and duty cycles. The 555 is used for a lot of circuits for its ease of use, stability and deep research into the chip.

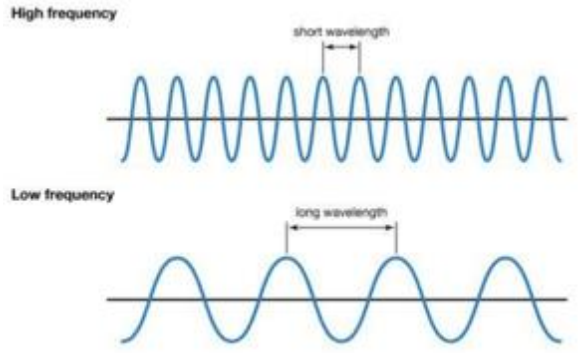




The first two mathematical concepts that need to be discussed when understanding the math behind the 555 Timer are the math behind the capacitance and the math behind the length of the rise and the length of the fall. The reason for understanding the math behind the capacitance has to do with the importance of the capacitor with the frequency. The rise and fall must be understood before understanding the duty cycle and frequency concepts. The capacitor has a variable that is very important because of its time to charge and discharge. The rise and fall play important roles in the mathematical equations for the time of each part of the wave. This is especially important in the duty cycle. The rise equation can be analysed as the time when the capacitor is charging through R2 giving it the equation $Rise \approx 0.693 \times R1 \times R2 \times C$. Looking at the fall equation this looks at the discharging of the capacitor being $Fall \approx 0.693 \times R2 \times C$.



A mathematical concept that finds itself present in the circuit is the duty cycle. In definition, the duty cycle is the percentage in any period where a signal is active. In the 555-time machine, an active signal can be defined as $V_{out} = V_{CC}$. For a mathematical analysis of the duty cycle, an equation must be created to understand how the cycle works. The way to figure this out is by calculating how much time the 555 is in a rise state and how long the 555 is in a fall state. This can become an equation that says DC (duty cycle)$=\frac{rise}{rise + fall}$. This can further become the equation DC$=\frac{R1}{2\times(R1+2R2)} +$ ½. This can be understood to use a resistor based on the information in the procedure on the function of the resistor in the circuit.



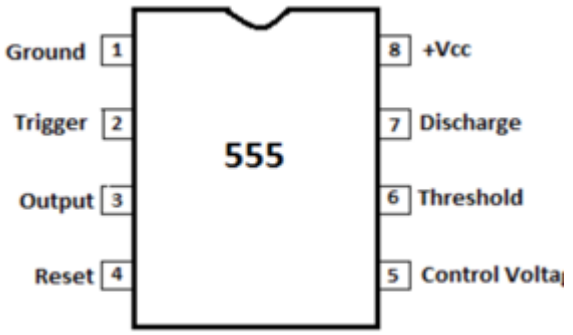
The duty cycle is one of two factors in consideration for how the 555 square wave functions. The other factor in mind here is the frequency function. The duty cycle can be looked at as a ratio for how long a rise and fall are but the frequency looks at the wavelength of both rises and fall being shorter or longer. The frequency in oscillation can be seen in an equation as firstly two equations being the time to rise and time to fall. After those two equations are found as they have been in the paragraph above next variable C must be introduced because in this circuit the capacitor and how long it takes to drain has a large correlation to the length of the wave. The other variable that needs to be defined is LN which stands for line neutral meaning it is the neutral line in all these wavelengths. This information can lead to the equation that when the 555 has frequency in astable mode the equation is$\frac{1}{C\times\ln(2)(R_1+2R_2)}$.

Procedure

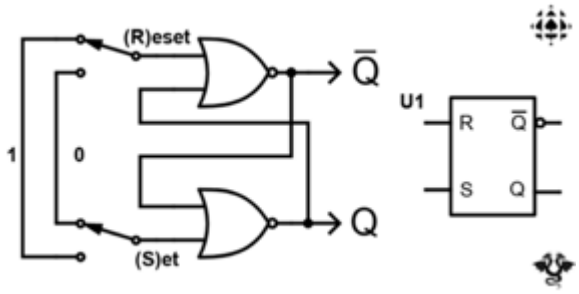
The first thing to be discussed is the less significant parts used in the circuit. First are the resistors, for this project resistors are significantly used at two points; the first is the three 5.1 kΩ used to divide the voltage to two divided points for the Op-Amps being 1/3 and 2/3 of our battery level. The second use of resistors correlates to the resistance of how long it takes to drain and fill the capacitor. The more time it takes to drain the capacitor, the more time the Op-Amps measure the draining capacitor. In the enhanced 555 timer a resistor can become a potentiometer to display what different resistances do to the output.

Parts Table	
Quantity	Description
1	3904 Transistor
1	LM358 Dual Op Amp
1	4001 CMOS Logic Chip
1	Light Emitting Diode
1	1 µF Capacitor
1	220 Ω Resistor
3	5.1 kΩ Resistor
1	1 MΩ Potentiometer
2	1 kΩ Resistor

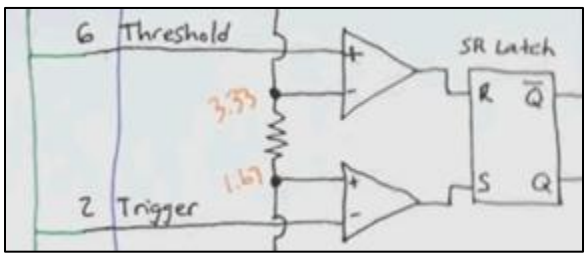
The pinout for the 555 is extremely interesting and will be further examined when examining how the 555 works on the inside. The first pin is voltage which when looking further has a greater significance than just powering the chip. The next two extremely significant pins are discharge and threshold. Threshold plays its most significant role when it is high from a capacitor discharge. The control voltage is used as a controller for the voltage levels when a 0.01 µF capacitor is used with it. Trigger plays its most significant role when low. The discharge pin is part of the connection to the external capacitor that is connected through a resistor that plays a role in the timing of the circuit. The Reset pin is used to restart the timing in the 555 but is actively low for normal operations. Lastly ground provides the voltage drop to allow it to travel.

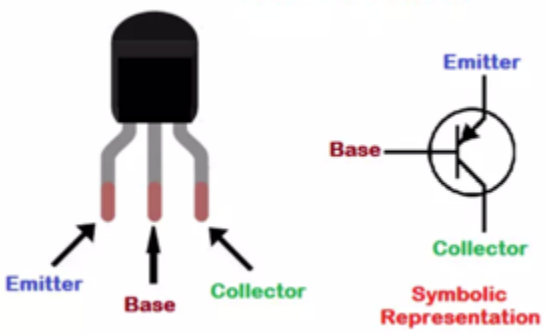


The first key part used in the 555 chip is the SR Latch. The SR Latch is a latch that has two inputs being set and reset. The output of the S part is Q and the output from the R is $\bar{Q}$. An SR Latch acts as a one-bit memory cell. This means if S or R isn't being pressed the latch will hold its previous state. The image on the right shows the SR Latch that is used in our circuit being two NOR gates receiving inputs from both A inputs while having both B inputs wired to the outputs.

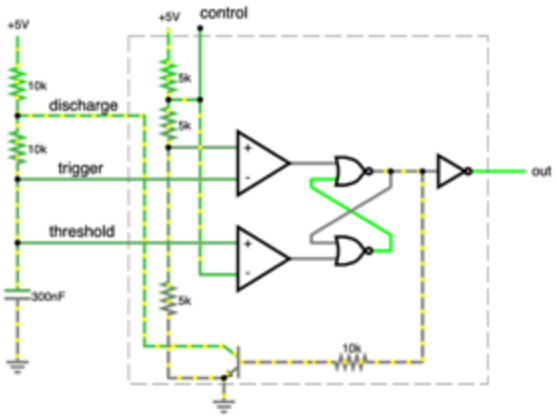


The SR Latch was an essential addition to the Op-Amp used in the previous project. The image to the right exemplifies this. The Op-Amp follows two rules being that Op-Amps have not current flow in or out and only observe voltage and secondly the Op-Amp tries to keep the inputs the same. In the case of the 555, 2 Op-Amps are wired from one constant and one changing voltage level. This is what makes the Op-Amp known as a voltage comparator, they only follow voltage signals and never use it.

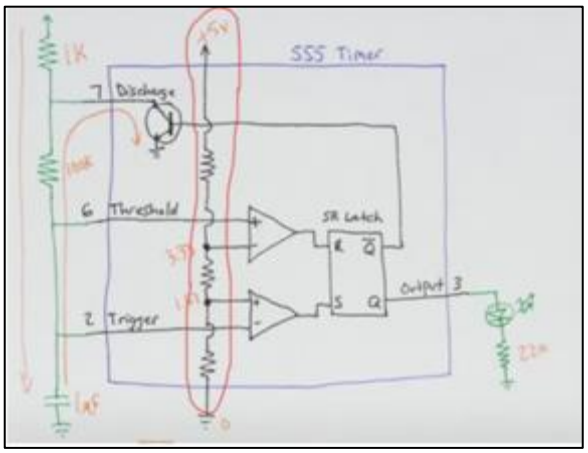




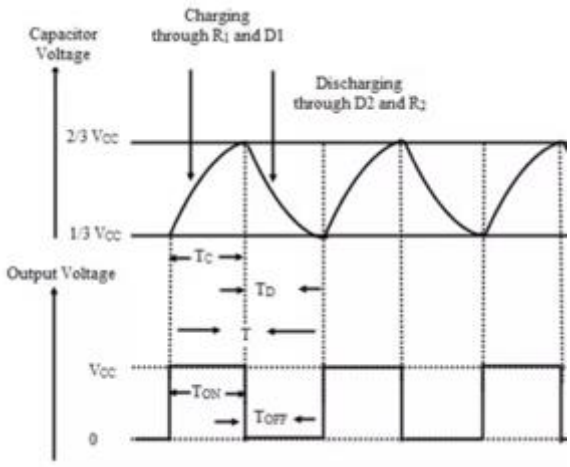
The next key part that plays a vital role in the circuit is the 3904-transistor role in the circuit. The transistor is considered the most important invention ever for its use as a gate. The pinout of a 3904 transistor is made up of three parts the emitter, base and collector. The key principle with the NPN transistor is that for a path from the emitter to the collector, there needs to be an input to the base. The transistor plays a major role in this circuit as it does in most oscillation circuits. In this circuit, the transistor can provide a path to ground when the base is connected which only happens when ground isn't presented and the voltage is powering the R latch. This connection between the base and reset latch is what plays a major role in the oscillation.



The 555 IC can be opened up and looked at how it can achieve all three states discussed in the procedure. First, we have to look at the 555 when it's opened up. All components used in this circuit have previously been discussed. With that knowledge, the schematic on the right shows all parts of the inside of the 555. This schematic shows a few key things discussed in correlation with the rest of the circuit. The first key part discussed is the 3906 transistors shown having one connection from the discharge pin having the main function of discharging the capacitor and one pin coming from the reset latch to power the base of the transistor. This enables the flow of current from the discharge pin to the ground. The last thing that this schematic shows that the schematic below doesn't is having the NOT gate present. This is used to produce a stronger output.



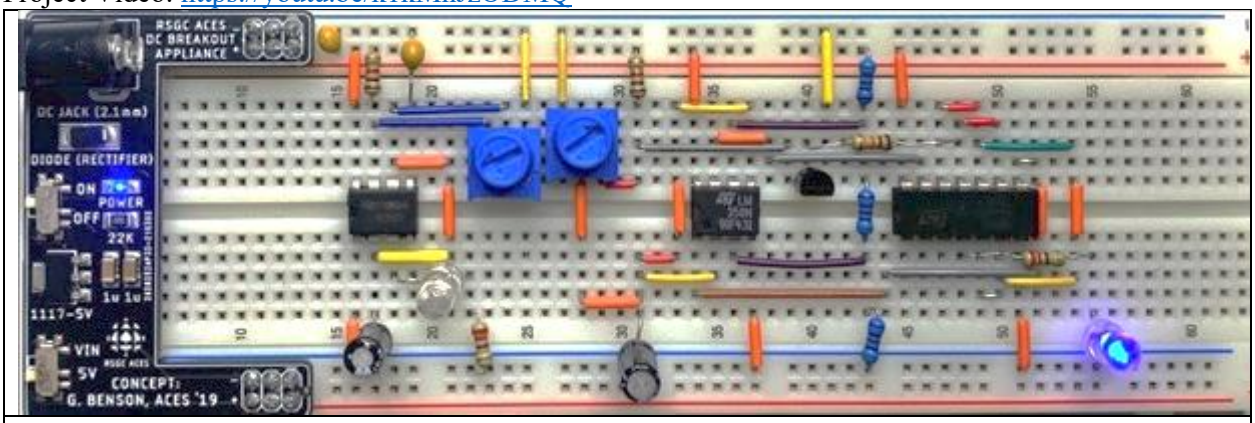
First the two power sources in the circuit. The first is the 5V that is powering the discharge, threshold and trigger. When 5V is sent through the terminal it starts to fill the capacitor. Once the capacitor is full it begins draining, enabling Op-Amps to function. The Op-Amps are always trying to make the voltage levels equal but the two power sources. The two Op-Amps will be named the trigger Op-Amp (the lower Op-Amp) and the threshold Op-Amp (the upper Op-Amp). The threshold Op-Amp has the threshold powered to the non-inverted pin meaning when its voltage exceeds the inverted pin voltage level it will send a power signal. The trigger however does the opposite because of is connected to the inverted pin meaning when its voltage exceeds the voltage of the other connected pin it turns off sending a 0 signal.



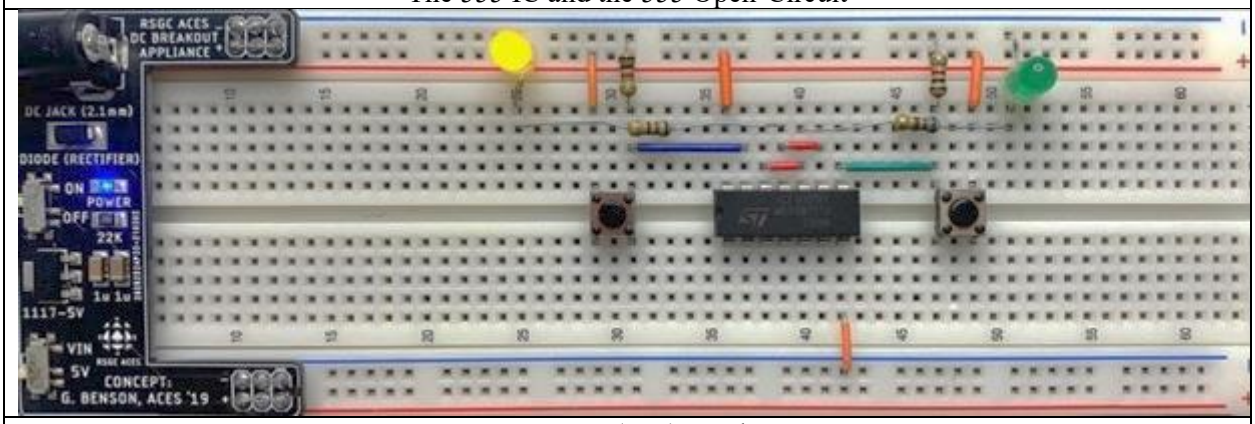
When current is sent through the two sources the initial state is that the trigger is below the 1.67, the other Op-Amp pins voltage, so it sends a positive signal to the set latch and turns on the LED. As this is happening the capacitor is filling up which can take a bit of time as it has to flow through a 100 kΩ resistor. Once the capacitor is filled up the voltage no longer has anywhere to flow and starts the draining of the capacitor when this happens the trigger and threshold Op-Amp pins observe the increase in voltage through the draining capacitor as it flows to discharge. When this happens, first, the trigger pin begins to charge past 1.67V and stops sending a signal to the set latch but as a latch, it keeps the same signal. The change happens when the threshold pin measures a charge greater than 3.3 it sends a high signal to the reset latch taking the high signal from the LED to the base of the 3906 transistor and providing a direct source to ground that drains the capacitor and restarts the entire process.

Media

Project Video: https://youtu.be/k1kMnJzODMQ



The 555 IC and the 555 Open-Circuit



Set-Reset(S-R) Latch

Reflection

I thought this project was a really good project from the time taken to make the circuit to the time spent understanding it. I spent a lot of time on this project and think the end product shows that. The DES was a place where I spent countless hours and began to truly appreciate the space.

Project 2.3 Multi-Bit Memory (Register)

Purpose

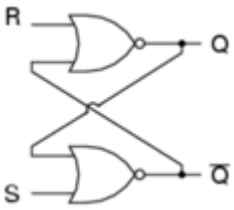
The purpose of the Multi-Bit Memory (Register) is to display memory through a configuration of memory holders and the monostable mode as well as helping to gain an understanding of Data Latches and Flip-Flops.

Reference

Project Description: http://darcy.rsgc.on.ca/ACES/TEI3M/Tasks.html#DFlipFlop
Eater's D-Latch Description: https://youtu.be/peCh_859q7Q?si=C58DrFmJTejHRVGv

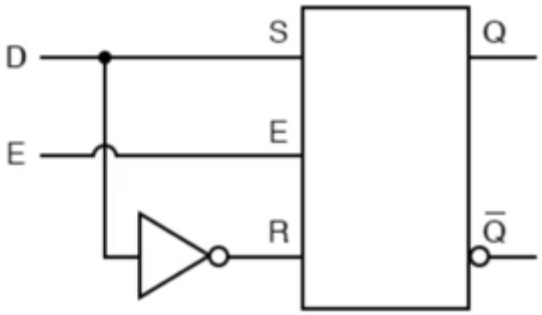
Theory

The theory behind how the multi-bit memory register functions builds on the learning made with the 555-time machine. The first aspect to look into is the gated S-R Latch used in the 555. This piece of electronics is a latch, meaning that if an input is sent to the set latch, for example, it will not lose that input until the reset pin is pressed. This has a useful function but also has a few flaws. Firstly, the first input to either set or reset is unknown when first powered which is not good in electronics. As well, another issue arises when both inputs are pressed simultaneously, this causes the problem of not knowing which input should go on leading to another unknown.

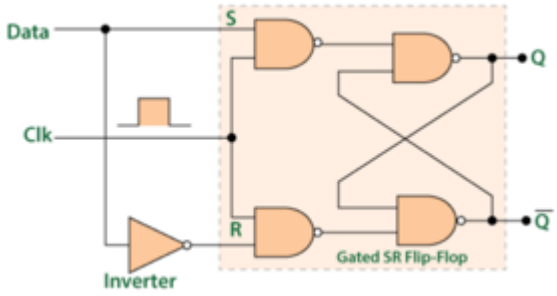


S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

These problems can be solved by using a D-Latch. The D-Latch is an enhanced S-R Latch by taking the original configuration and adding how the inputs get to that configuration. Instead of two pins the D-Latch only has one that is always inverted for the R latch. The D-Latch functions off of two inputs being the data input and the clock input. The data input takes in data when the clock input is enabled by a rising edge. The data latch takes in inputs provided when a 1 is sent through the data pin a 0 goes to the reset pin because of the inverted gate. This data will be inputted to the S-R latch only if the clock latch is also on allowing the AND to function. In summary, if the clock pin is inputted the data latch functions just like an S-R latch, without errors but if the clock pin is off the data latches to its current input.



The understanding of the D-Latch helps to understand the device used in this circuit; a D Flip Flop. There are a few differences between the D-Latch and D Flip Flop. The first has to do with the clock input. In the D Flip Flop in the multi-bit memory register the dependent factor is the rising edge received from the clock input in a positive D Flip Flop. There are also negative D Flip Flops that are dependent on the falling edge. The latch is trigger sensitive so as long as the enable pin is pressed the data pin functions like an S-R Latch. In summary, the D Latch is a triggering device and the D Flip Flop is an edge trigger device.

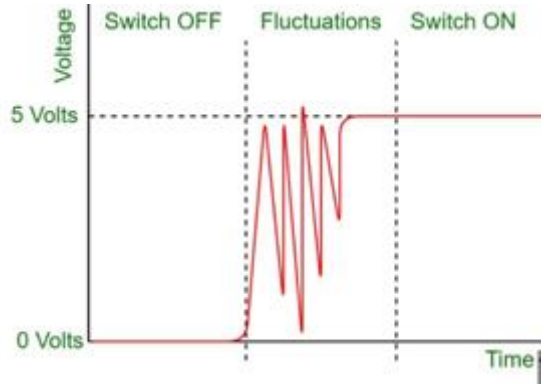


Procedure

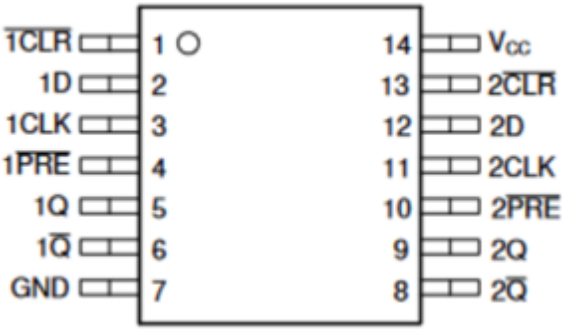
Parts Table	
Quantity	Description
8	Light Emitting Diodes
4	SN74HC74 IC
1	NE555 IC
3	SPST Button
1	10 nF Capacitor

The first device that is present in the multi-bit memory is the NE555 IC being put to use as a monostable input device. In the last DER, the 555 was used to create oscillation for an LED but the 555 has more than 1 mode. One of the other modes is the monostable mode. The reason this mode can be useful is because of something called bounce. Bounce is something that a normal single-pole single-throw button creates when the two metal pieces are moved together to connect sometimes the metal bounces off and provides an unstable input for a moment, in electronics this is not helpful.

The 555 in monostable solves this issue because of its configuration. The 555 in oscillation mode will receive constant power and provide the changing output. In monostable mode, the input provided is by a button meaning it won't receive constant power. This leads to the button immediately filling the capacitor and draining to the Op-Amps providing the button input without debouncing when going through the capacitor and Op-Amps. The image on the right shows the bouncing that the button can create. In this circuit that can cause the circuit to have an unknown conclusion because of the rising and edges created by this button that the clock input records.



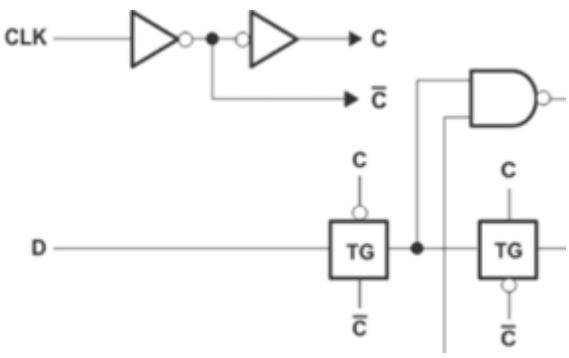
The next key part used in this circuit is the SN74HC74. For the standard 8-bit device 4 of these chips are needed. The pinout has several inputs each playing roles in this circuit. This chip is an integrated circuit of the D Flip Flop. This means that it will need two pins per D Flip Flop being the clock input and data input. A few other important pins used in this circuit are voltage and ground to provide clear 5V outputs. The pre-set input for the multi-bit memory operation can be set to low because the only inputs for this operation are coming from the clock and data. Lastly, this chip has outputs from Q and $\overline{Q}$ just as a standard D Flip Flop produces.



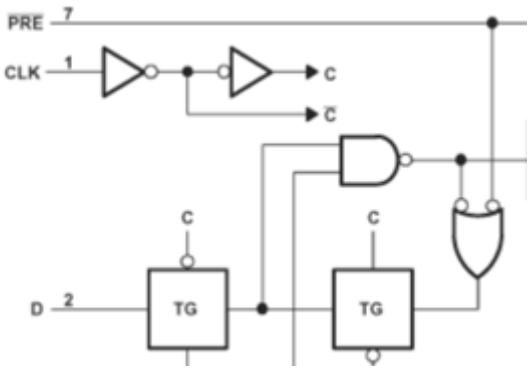
The first feature to understand in this circuit is how the inputs are received from the clock input. This input is being received from the clock input. The clock input plays a role in producing two outputs to Q and $\overline{Q}$ meaning it has to enable the output from Q when receiving an input as well as making sure the $\overline{Q}$ will not turn on. This can be done through two NOT gates one inverted the pin for the $\overline{Q}$ output and the other to keep the input the same for the Q output.



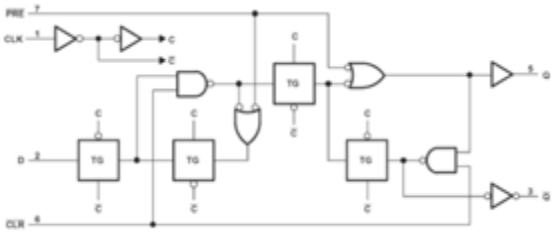
The next input to the SN74HC74 is the data input. This input is coming from a standard button with no need to debounce. Taking the input from the clock decides what happens to the data input when going through the following gates. The clock inputs start allowing the data to function in the chip when the clock reads a rising edge. The data goes through the TG which varies the output based on the clock output. This process is present at three points in the circuit these three points decide what inputs go into several gates through the IC to provide either a 1 signal to Q or $\overline{Q}$.



The next pin that plays a significant role in this circuit is the clear pin. For this pin in normal function, it should be set to active low. When the clear is set to high the circuit is reset regardless of any other inputs. This has to do with the IC operating off of NAND gates which when another high signal is set allows all other inputs not to become an input of zero. The other pin that should be set to 0 for normal operation is the pre-set pin. What makes this pin different is that it has the job of varying how the chip functions by pre-setting. The difference between this pin and the clear pin is that this pin is connected to NOR gates allowing for function rather than just clearing all operations. The image on the right shows this pre-set pin in the integrated circuit being wired to a NOR gate.



After understanding all essential pins, it can now be understood how every input functions in this IC. Firstly, the data input will not go through any of the logic gates unless a rising edge is recorded through the clock pin. The data input when positive and allowed through with the clock pin will then go through the NAND gate which will produce a 1 output unless the clear input is on which produces a 0 output. This output then goes through a NOR gate which goes through a buffer gate. A few other situations can lead to different outputs. When the first clock input does not go through leading to more gates all being stoppages of the input. The pre-set pin can lead to no input through the NAND gate and lead an output through the $\overline{Q}$.



This digital circuit stores binary data using the SN74HC74, a dual D-type flip-flop integrated circuit. Applications involving data latching and sequential logic operations are benefitted most from it. The clock input is an essential part of many digital systems since it allows you to regulate when data is sampled and stored. Unlike the D Latch, the image on the right shows the Flip Flop response using the edges.

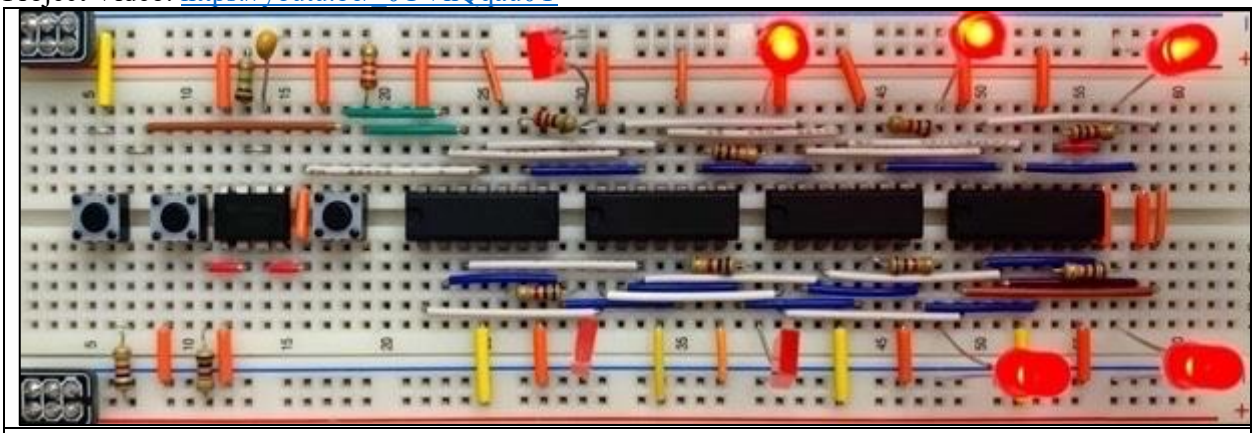


Parts Table	
Quantity	Description
2	Light Emitting Diodes
1	4001 CMOS Chip
1	4081 CMOS Chip
2	SPST Button
2	1 kΩ Resistor
2	680 Ω Resistor

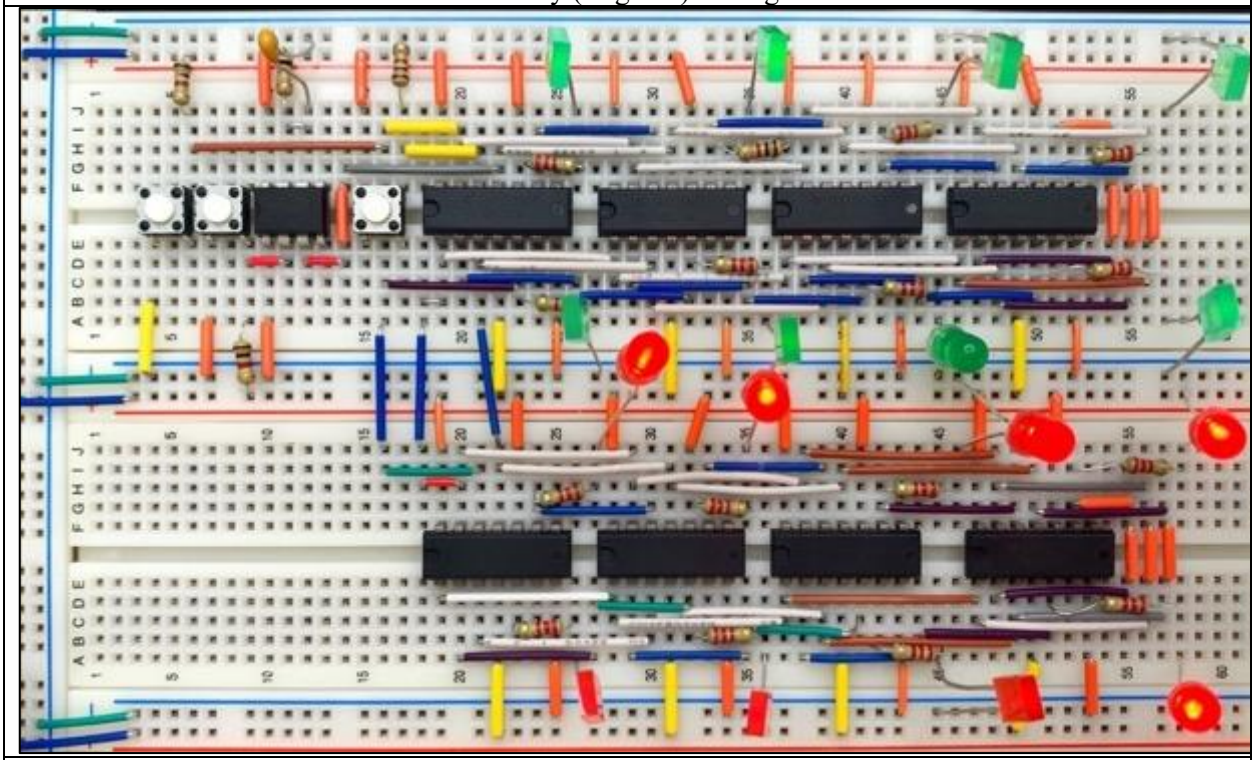
The Multi-Bit Memory (Register) can be enhanced for deeper learning on a device used. This device that can have the learning deepened is the D-Latch opened up. As already discussed, the D-Latch is a level-sensitive device. This can be displayed by the 2 pull-up buttons for the enable and data inputs. The data input has one input inverted conservatively this can be done by wiring one input through two inputs on a NOR gate. This output along with one directly from the button can go through an AND gate with the key function of the enable pin being used. Lastly, two NOR gates are wired together creating the D-Latch.

Media

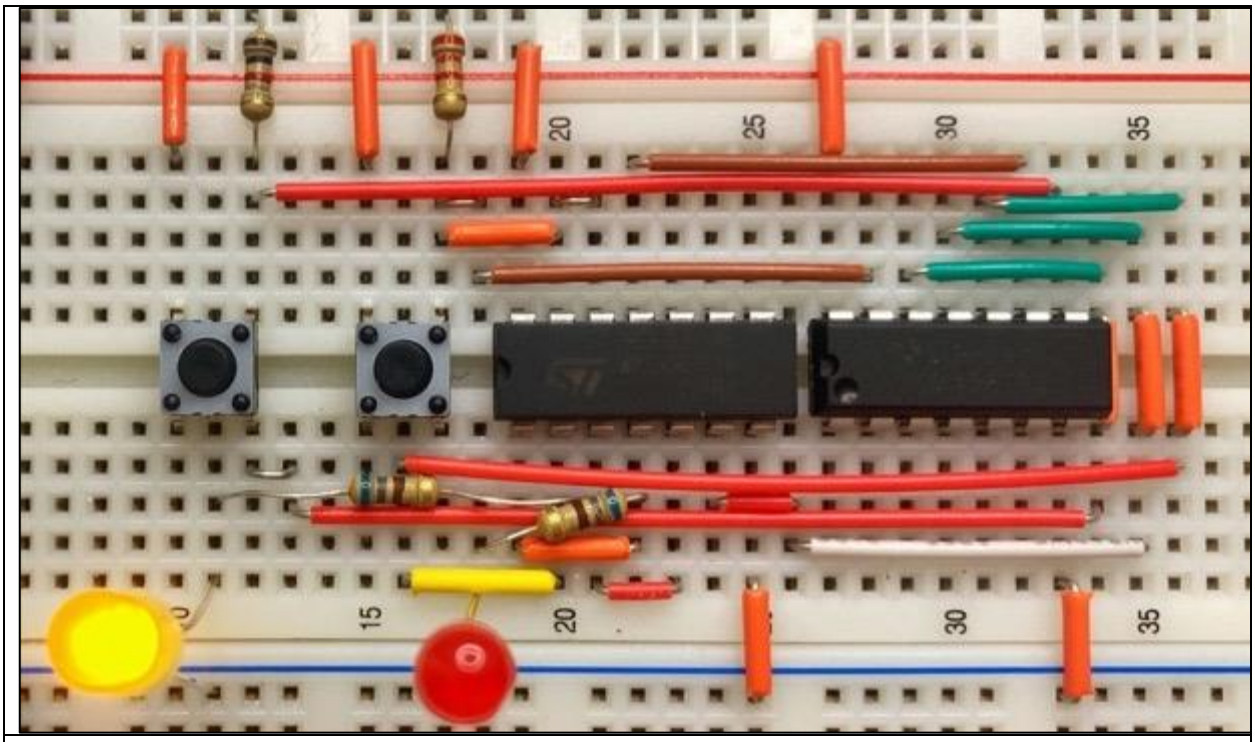
Project Video: https://youtu.be/_0UVhQqau0Y



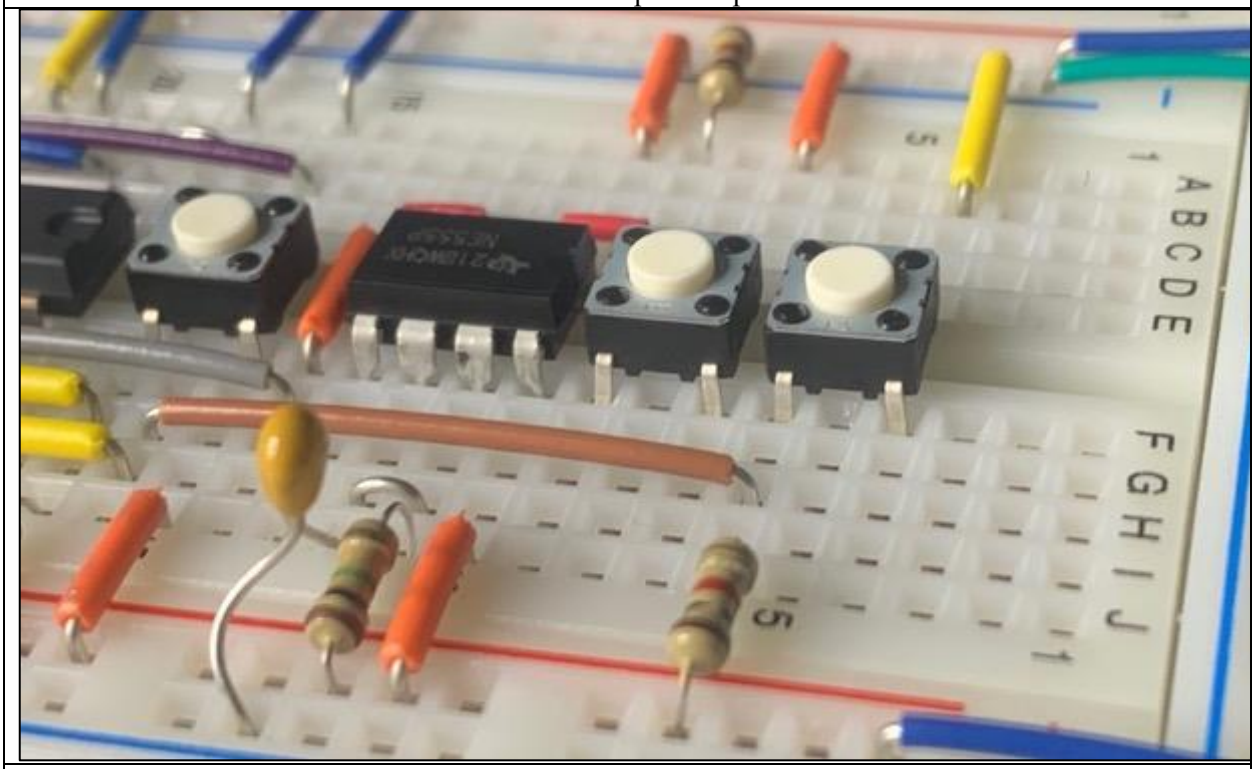
8-Bit Memory (Register) storing 6-Bits



16-Bit Memory (Register) Storing 16-Bits



D-Latch Opened Up



Close Up of Three Inputs(Clear, Clock and Data)

Reflection

One thing I had to understand about this project was how it was related to the real world engineering I hear about. This has to do with balancing the ISP and this project along with other subjects. It gave the chance to experience a lot of hard work and challenges to overcome.

Project 2.4 Binary Game Part 1

Purpose

The purpose of the Binary Game Part 1 is to create an 8-bit display while gaining an understanding of key parts used such as the Morland Bar graph. This also allows an introduction to C through the functioning of the Arduino Nano.

Reference

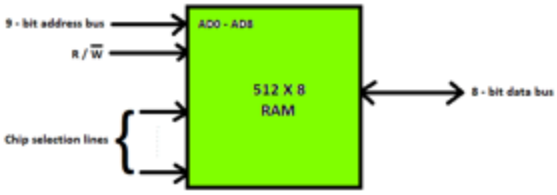
Project Description: http://darcy.rsgc.on.ca/ACES/TEI3M/Tasks.html#BinaryGame1
C Operator Precedence: https://en.cppreference.com/w/c/language/operator_precedence

Theory

The use of the Arduino brings the introduction to C code as said in the purpose. C provides fast and reliable code with a few rules and short cuts that can be understood deeper. The first rule that needs to be understood is the precedence of operators. In C the precedence functions very similarly to the precedence of arithmetic known as BEDMAS (Brackets, Exponents, Division, Multiplication, Addition and Subtraction). The one precedence that can be discussed further is the logical operators which have a precedence after BEDMAS.

Operators	Type
!	Logical NOT
* / %	Arithmetic
+ -	Arithmetic
< > <= >=	Relational
== !=	Relational
&&	Logical AND
||	Logical OR
=	Assignment

The second concept to be understood is memory with the Nano. When statements are defined the machinery needs to remember these statements and remembering takes up RAM. RAM stand for Random-Access-Memory. This is a computers short term memory. RAM stores all data that is needed for the computer to run. RAM has the ability to organize data but with more data the organization gets slower and slower.



The most important operator used in the Binary game are the logical operators. One thing to point out is the difference between logical and bitwise operators. Logical operators have the function of compare bits presented and can produce multiple results while bitwise operators take an individual bit and perform the logical operations with only one output. For this project because 8-bits are being compared and one output is presented the operator being used is bitwise. The logical and bitwise operators function just as logic gate ICs. In the binary game 7 OR-gate comparisons are being made. Comparison is made in left to right order with the two most left binary value inputs being compared producing one output that is compared to the next most left bit and having this process continue. The image on the right displays bitwise math.

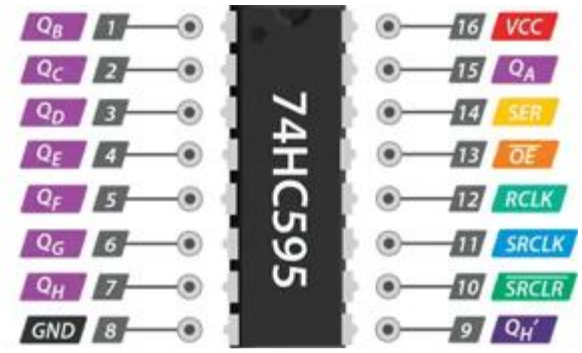
	Decimal value	Binary value
Input 1	5	00000101
Input 2	3	00000011
Bitwise And	1	00000001
Bitwise Or	7	00000111
Bitwise XOr	6	00000110

Procedure

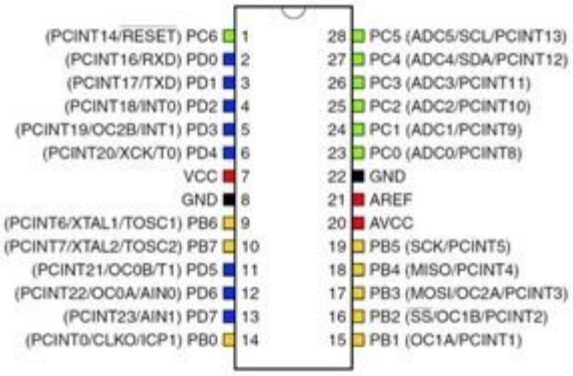
The Binary Game takes the first look at code being applied to circuitry. For this DER the code is the middle-man between the Binary inputs from the rocker switch and the display created from the Morland Bargraph. The Rocker Switch works with 8 pull-up resistors to provide the value 0 and 1 to the Arduino Nano. The three significant parts to this project are the Morland Bargraph, Rocker Switch and Arduino Nano which will all be discussed further in the DER.

Parts Table	
Quantity	Description
1	Arduino Nano
1	Rocker Switch
1	Morland Bargraph V3
1	1 kΩ Bussed Resistor
1	Push Button

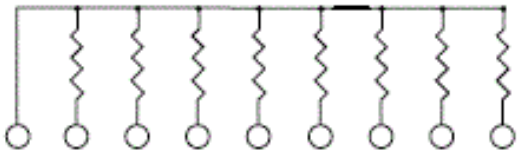
The first key part that can be examined deeper is the Morland Bargraph. The Morland Bargraph is made of four significant pieces being 6 pins for the 6 inputs necessary, the 595 IC, the 10-segment bargraph and one 11 segment 10 kΩ bussed resistor network. The most important part when understanding the Binary game is understanding the 595. This integrated circuit can be described as a shift register. As learned is the past DER a combination of SN74 ICs created one 595. What this means is that the 595 will use two of the inputs from the memory register being the Data and Clock along with the new addition of the Latch input. The clock and data are both influenced by the binary input connected through the Arduino while the latch is manually told when to go high and low.



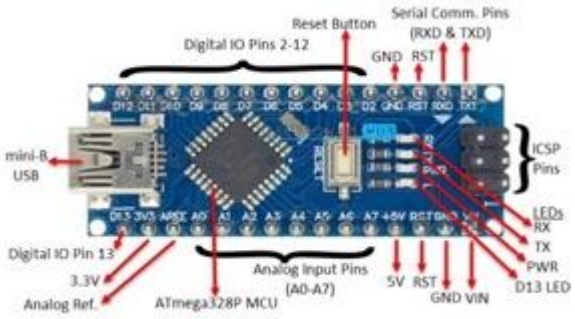
The Arduino Nano is a microcontroller that works with breadboards extremely well. The microcontroller functions in the same manner as the microchip ATmega328P as it has one embedded in the surface. The first section being the power and ground. The difference however is that unlike ICs the Nano can receive power from a USB and the power and ground power the breadboard. The one key thing to point out is there are three different power rails being the 5V, VIN and +3V3. The VIN can produce 7-21 volts so when working with the 595 it's very important to use 5V. The most important piece however is the ATmega328P. This microcontroller provides the best application with the Arduino Nano for its low cost and power supply. The key thing that this microcontroller can do is have a wide range of operations it is able to complete for the Arduino Nano. These abilities include analog-to-digital converters, universal asynchronous receiver-transmitter communication, timers/counters, and general-purpose input/output pins.



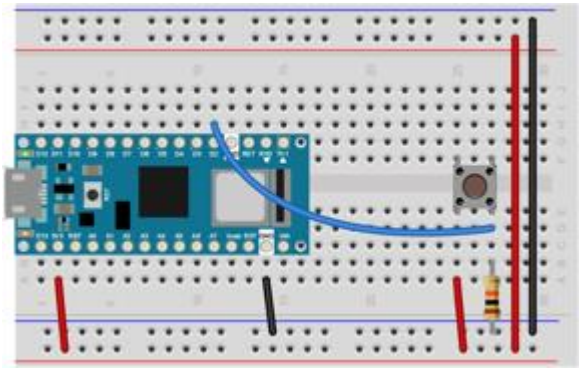
The other key part to be discussed in this DER is the 9-pin bussed resistor network. This resistor network holds 8 l kΩ resistor networks with 1 common cathode. The diagram on the right shows a schematic of the exact resistor network used in this circuit.



The next section to be understood is the pins labelled from D0 to D21. These pins play a significant role in the functioning of receiving and outputting bits that may be alerted through the code of the NANO. For D0 and D1 these pins may also be used for receiving and transmitting. The pins D12 and D11 are data channels that stand for master-in-slave-out (MISO) and master-out-slave-in (MOSI). AREF provides an Analog Reference. AREF can provide a voltage reference when an external power supply is used.



In summary, the rocker switch is wired to a 9-pin bussed resistor and sends 8 binary values that have each value shifted one less than the other to the left. These binary values are compared through a bitwise OR, a concept covered in the theory section. From this point, the Nano takes the 8-bit binary value and programs the 595 to allow a shift register to happen with the 8-bit provided. This happens through the clock and data allowing 8-bits to be presented before the latch pin presents all 8-bits in parallel to the output. So, the Nano will provide the rising edge for the clock and then provide the outputted data to the 595 data pin. Once these two things have happened the Arduino Nano presents a high to the latch pin presenting this binary value to the Morland Bargraph. With all of this happening in a loop statement this process continues forever creating the binary game.



Code

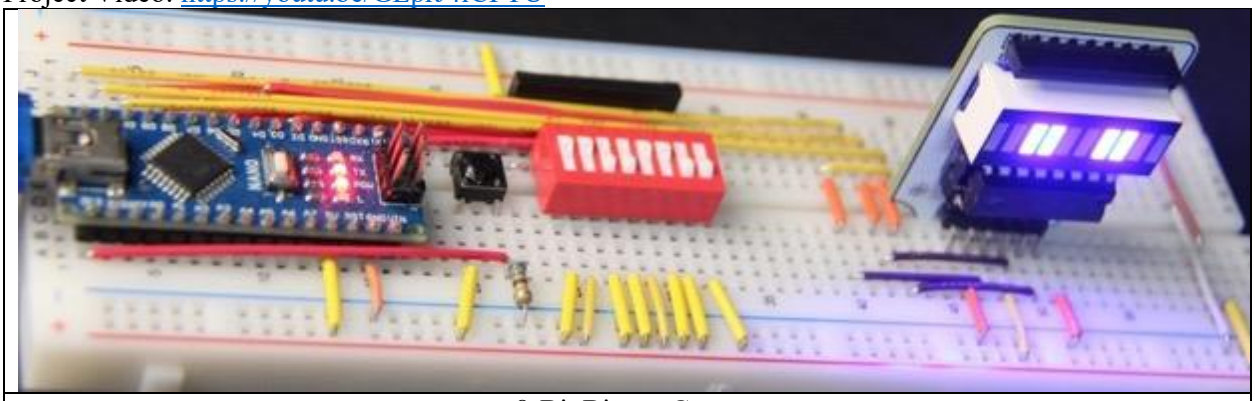
```
// PROJECT :Binary Game
// PURPOSE :Display binary value created by rocker switch through Morland Bargraph
// Author  :M. Zischka
// COURSE  :ICS3U-E
// DATE    :15 11 2023
// MCU     :328 P
// STATUS  :Working
//Reference:http://darcy.rsgc.on.ca/ACES/TEI3M/Tasks.html#BinaryGame1
// NOTES   :I used 2.1 Arduino IDE with an Arduino Nano as my microcontroller
#define DATA 12   //pin 12 data
#define CLOCK 10  //pin 10 clock
#define LATCH 11  //pin 11 latch
#define MSB 2     //input pin most significant bit
#define SMSB 3    //input pin second most significant bit
#define TMSB 4    //input pin third most significant bit
#define FMSB 5    //input pin fourth most significant bit
#define FLSB 6    //input pin fourth least significant bit
#define TLSB 7    //input pin 42hird least significant bit
#define SLSB 8    //input pin second least significant bit
#define LSB 9     //input pin least significant bit
#define octalToBinaryPin 13
uint8_t mSBS;  //all digital read value for input pins
uint8_t sMSBS;
uint8_t tMSBS;
uint8_t fMSBS;
uint8_t fLSBS;
uint8_t tLSBS;
uint8_t sLSBS;
uint8_t lSBS;
```

```
uint8_t octalToBinaryRead;
void setup() {
  // put your setup code here, to run once:
  pinMode(DATA, OUTPUT);
  pinMode(CLOCK, OUTPUT);
  pinMode(LATCH, OUTPUT);
  Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
  octalToBinaryRead = digitalRead(octalToBinaryPin);
//Reads button constanly
  if (octalToBinaryRead == 1) {
//provides option for input of 1
    digitalWrite(LATCH, LOW);
//allows new binary inputs
    mSBS = digitalRead(MSB);
//reads switch pin 1
    sMSBS = digitalRead(SMSB);
//reads switch pin 2
    tMSBS = digitalRead(TMSB);
//reads switch pin 3
    fMSBS = digitalRead(FMSB);
//reads switch pin 4
    fLSBS = digitalRead(FLSB);
//reads switch pin 5
    tLSBS = digitalRead(TLSB);
//reads switch pin 6
    sLSBS = digitalRead(SLSB);
//reads switch pin 7
    lSBS = digitalRead(LSB);
//reads switch pin 8
    uint8_t value = (MSBS << 7 | SMSBS << 6 | TMSBS << 5 | FMSBS << 4 | FLSBS << 3 |
TLSBS << 2 | SLSBS << 1 | LSBS);  //compiles all binary values together giving one
8-bit binary value
    Serial.print("Octal Value:");
//displays binary definition beside
    Serial.println(value, OCT);
    shiftOut(DATA, CLOCK, MSBFIRST, value);
//outputs the binary value compiled
    digitalWrite(LATCH, HIGH);
//displays output provided
  } else {
//Provide counter to 1 value
    digitalWrite(LATCH, LOW);
//allows new binary inputs
    mSBS = digitalRead(MSB);
//reads switch pin 1
    sMSBS = digitalRead(SMSB);
//reads switch pin 2
    tMSBS = digitalRead(TMSB);
//reads switch pin 3
    fMSBS = digitalRead(FMSB);
//reads switch pin 4
    fLSBS = digitalRead(FLSB);
//reads switch pin 5
    tLSBS = digitalRead(TLSB);
//reads switch pin 6
    sLSBS = digitalRead(SLSB);
//reads switch pin 7
    lSBS = digitalRead(LSB);
//reads switch pin 8
```

```
    uint8_t value = (MSBS << 7 | SMSBS << 6 | TMSBS << 5 | FMSBS << 4 | FLSBS << 3 |
TLSBS << 2 | SLSBS << 1 | LSBS);  //compiles all binary values together giving one
8-bit binary value
    Serial.print("Binary Value:");
//displays binary definition beside
    Serial.println(value, BIN);
//Provide Binary Value
    shiftOut(DATA, CLOCK, MSBFIRST, value);
//outputs the binary value compiled
    digitalWrite(LATCH, HIGH);
//displays output provided
  }
}
```

Media

Project Video: https://youtu.be/GEprJ4iCFTU



8-Bit Binary Game



Close Up of Binary Inputs

Reflection

This project saw an introduction to coding which was a big step for me who had never seen code before but I was excited because I knew software was as present in the future as it has been in the past 50 years. The code process was difficult but had me excited and this DER provided those same emotions.

Project 2.5.3 Short ISP: The Binary Calculator

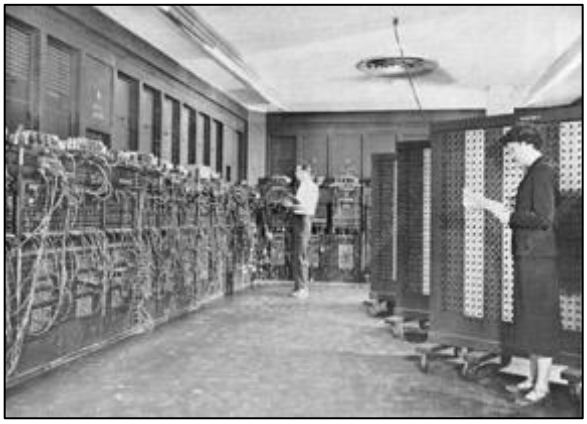
Purpose

The purpose of the ACES Binary Calculator is to conduct binary addition and subtraction using the learning of Logic gates and Binary inputs displayed through light-emitting diodes.

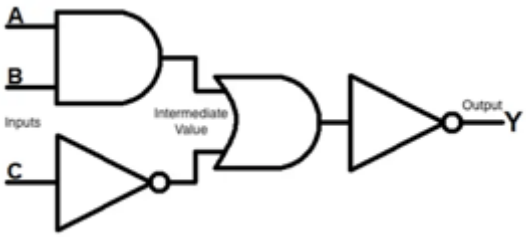
Reference

2-Bit Adder Description: http://darcy.rsgc.on.ca/ACES/TEL3M/2223/TasksFall.html#AdderTask
4000 series CMOS Information: https://electronicsclub.info/cmos.htm

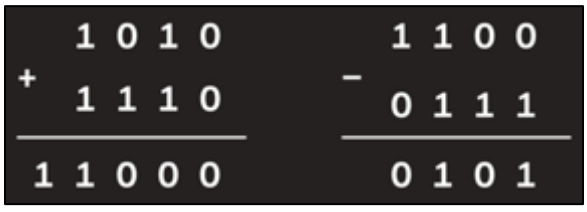
Theory



Firstly, the concept of the ACES binary calculator is to have two sides of switches, these being the addends. The inputs are provided to a configuration of logic gates learned last year to produce two outputs the sum and the carry. The sum is displayed through an LED while the carry is inputted into another configuration for another bit. This process can go on forever each time doubling the number of numbers available and the sums created. The binary calculator will conduct binary math with only logic gates being used. The image on the right shows a calculator from the 1940s displaying part of the inspiration behind this project. Calculators being part of the understanding and development in technologies history make this so significant.



The first theoretical idea to understand about the binary calculator is the logic gate concept. Logic gates are transistor configurations that provide specific outputs based on the inputs. For the case of the binary calculator because two different binary values are being added the logic gates necessary are two-input-one-output logic gates. The binary adder and subtractor use separate logic gate configurations to reach the separate goals. The concepts of these goals will be further talked in the procedure. The image on the right shows and example of a logic gate configuration with the three parts: inputs, intermediate values and outputs.



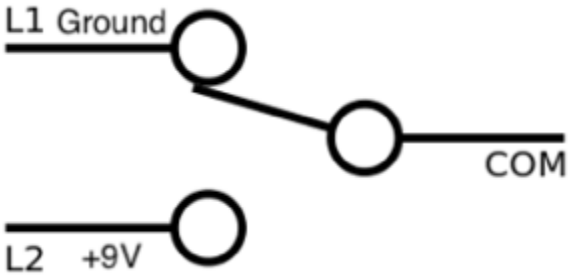
The second principle, building on the first, to be understood about the binary calculator is the binary math used. In standard arithmetic base 10 is used. This entails that a number counts up from 0 to 9 before carrying over to the next value (ones to tens to hundreds…) but when making a *binary* calculator different math is used. Binary is made in base 2 meaning a number can count up from 0 to 1 before, just like base 10, carrying over. The binary calculator takes on 4-bits which are weighted differently. The bit values increase as the switches distance further from the LED as seen in the media section. The bit values range from 1-8 all being exponential values of 2. The image on the right displays this binary math with carries of the value one both adding and subtracting to the final value.

Procedure

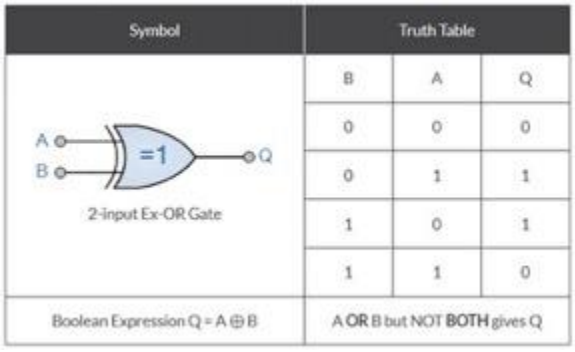
Parts Table	
Quantity	Description
9	Light Emitting Diodes
8	SPDT Slide Switch
4	4081 CMOS Chip
1	4069 CMOS Chip
4	4070 CMOS Chip
2	4071 CMOS Chip

The binary calculator as noted in the theory is made up of two logic gate configurations. The images found in media display how the calculator is split down the middle into the inputs, configuration and outputs for both the adder and subtractor. The inputs are the slide switches, the configuration is the logic gates and the outputs are displayed through the light emitting diodes with red for addition and green for subtraction. This circuit can be powered with a voltage range of 5V to 13V for satisfaction with the chips. The concept of addition and subtraction needs to be displayed in a calculator. When making a 4-bit adder this allows for a minimum value of 1 and a maximum value of 30 meaning that 5 bits are needed to display the value 01111 while the 4-bit subtractor max value is 16 meaning only 4 LEDs are necessary to display 1111.

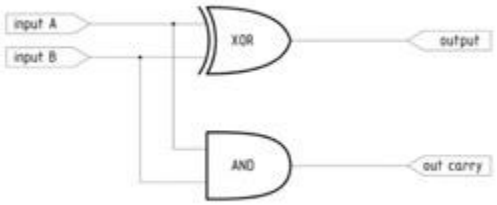
The other simple part to understand incorporated in this circuit to learn before beginning about the logic gate configurations is the slide switches. Slide Switches are the best type of device for the goal trying to be achieved with the binary calculator an argument can be made for the SPST button yet if wanting the binary value to *stay* inputted it makes the most sense to use slide switches. The slide switches are wired to power and ground to provide binary values 0 and 1 to each logic gate necessary.



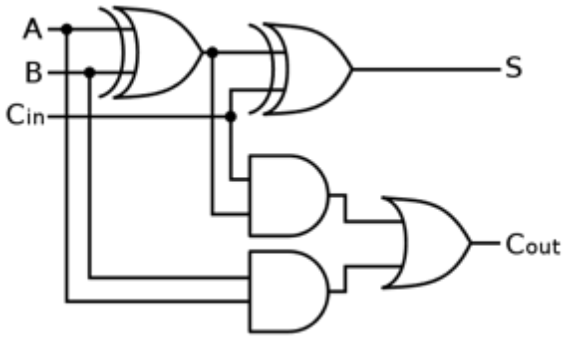
Now with an understanding of how the inputs are created and the outputs are displayed it is now time to see how it is possible to get from point A to B. The first concept to be looked at is the adder when understanding how an adder will work the first concept to go back to is the XOR logic. A concept spoken about last year has the possible outputs wanted being a value of zero and a value of 1. As the image shows the truth table for this logic gate it begins the most primary part of the adder's logic gate configuration.



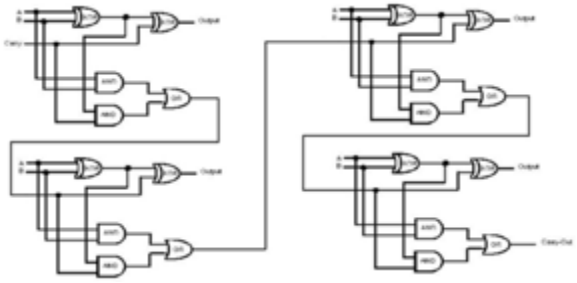
The flaw that comes with the XOR gate as already spoken about in the theory section is that the outputs it provides would not show correct addition. The first three statements are correct yet the flaw appears when 1 and 1 provide an output of zero. Now this is not exactly wrong because when working with base 2 when 1 and 1 are added the sum should be a carry. The best way to do this operation is to include an AND gate providing a second output known as a carry. Here is an image showing the second addition made to the 4-bit adder.



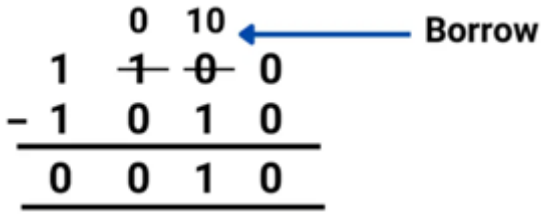
Now that concept discussed above works with just one bit being added. Problems however occur when taking this carry to another piece of bit addition it knows requires a new configuration. When taking this carry-out a new input is created called the binary input. The binary input almost acts like memory as it keeps the previous sum in the circuit as it continues to go through the configuration. This provides the last addition to the simple look at how each bit for the 4-bit adder should look before finally wiring all 4 together with carry-out being the tying factor.



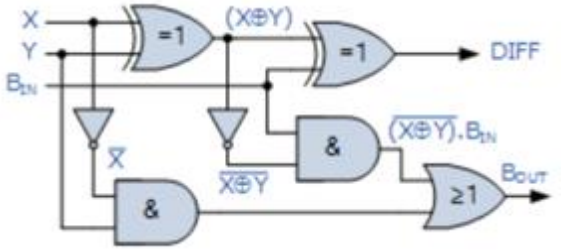
Here is a look at how this logic gate configuration is when there are 4-bits. Each A and B shows a different slide switch these A's and B's can also be classified as $A_0 - A_3$ and B_0 to B_3. A_0 and B_0 represent the first configuration without a binary input and each configuration that has more binary inputs before displays a larger sum. The two XOR gates allow for the configuration to decide if an output show goes through the carry-out or through to the sum/output. The two AND gates decide how the carry-in is utilized in the configuration of the next bit. Lastly, the OR gate makes sure that the only way a binary carry-out occurs is if both inputs are 1 or if the carry-in and the output from that singular bit's addition both have an input to the OR gate as 1.



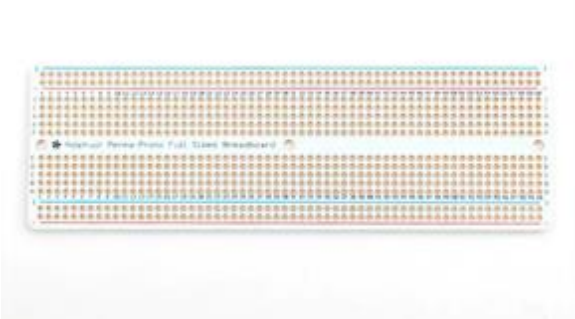
The 4-bit Subtractor begins to build on the idea of the 4-bit adder. The only difference being worked with is that a 4-bit subtractor will have one set of addends that represent negative numbers. This is a concept that will be touched on further in the procedure. At the moment the first task is understanding how a 4-bit adder can be created but with one set of negative numbers. This means the building of the subtractor will look extremely similar to that of the adder with just a few tweaks. This also means the starting gate to work around is once again the XOR gate.



The XOR gate will work better with a subtractor because every statement, other than $0 - 1$, provides the correct outputs as looked at in the truth table above. The one thing that requires more logic gates is the carry. Conceptually speaking what this means is that if one side inputs the number 5 and the other inputs -3 there needs to be a way for the negative or positive number to carry into the next bit addition. In the end, the circuit will have the same look as the 4-bit adder yet the Y-values will go through a 4069 NOT gate to invert the value inputted. The configuration allows this inverted input to represent a negative number being added with the positive X value following the same rules as the 4-bit adder above.



With a concept that was made successfully on a breadboard, as seen in the media section, the next step came with making a permanent version of this case. The first step was finding out what materials to use for this. On a hardware side, the same parts would need to be used. From a choice of what permanent material to use the best choice was the perma-proto as perf-board and stripboards didn't make sense for a solder job of this size. In the end, the materials included all hardware parts seen in the parts table as well as 3 Adafruit perma-protos.

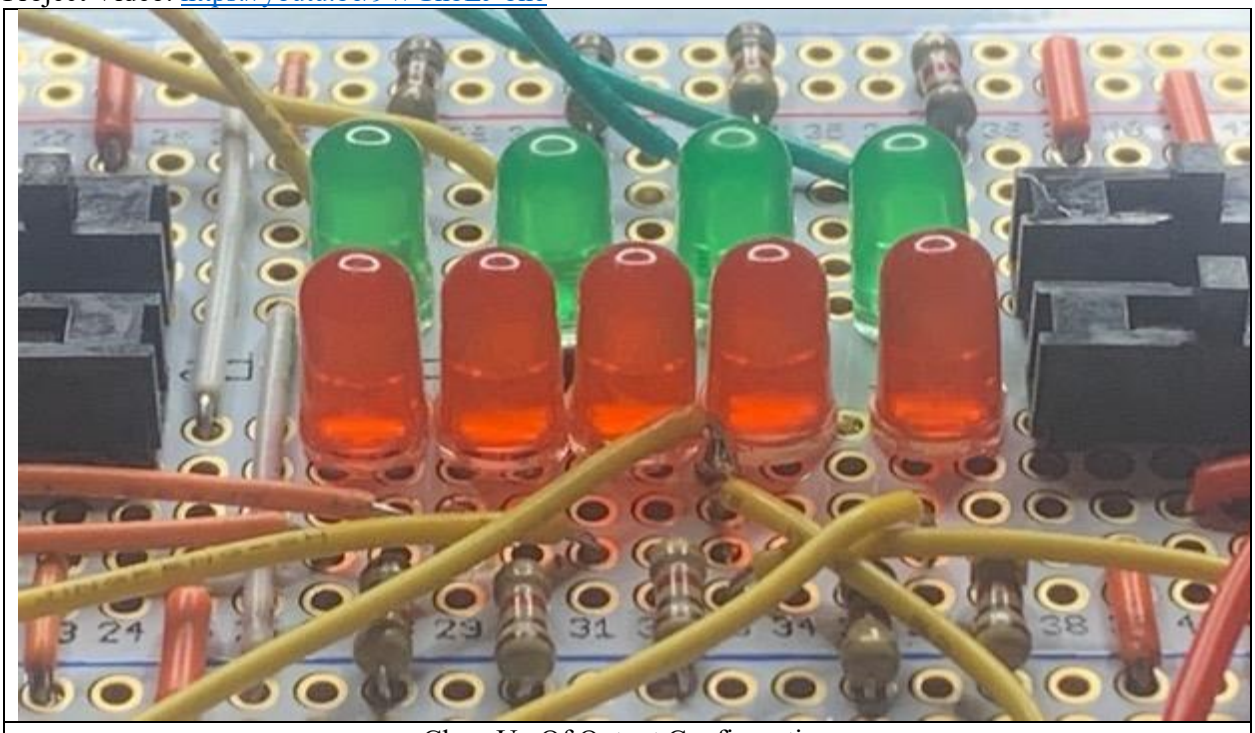


The last step in designing this calculator is the 3D printing case. The first idea behind the case was a single perma-proto case that would have three perma-protos stacked. The flaw behind this design idea is looping the inputs and outputs as well as the ground and power rails. This idea was then refined by creating a design that allowed three perma-proto cases to be stacked horizontally. The image on the right shows the design used for the calculator. In total three print jobs were done to complete this objective.

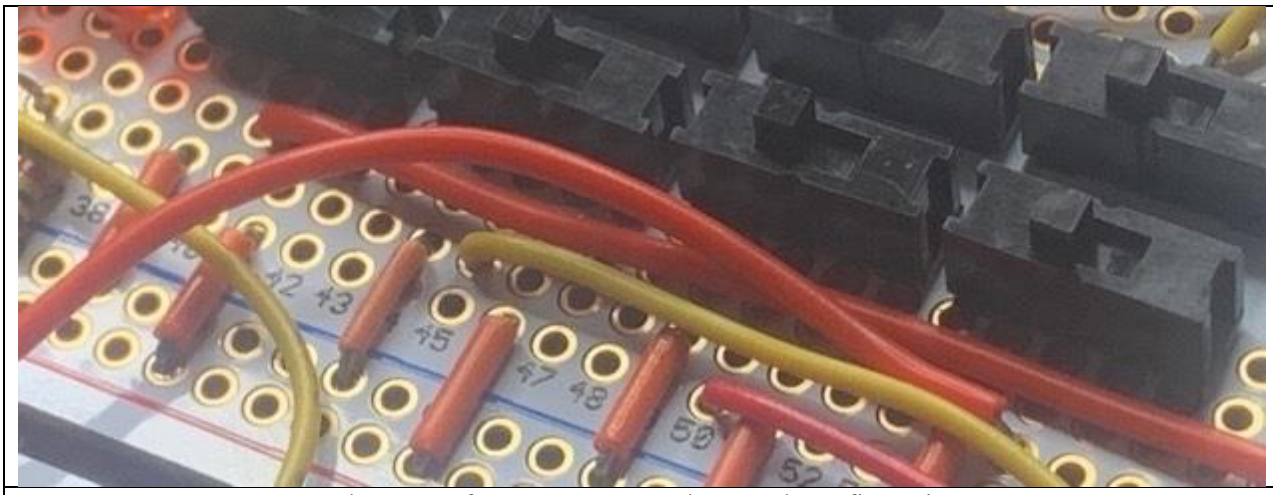


Media

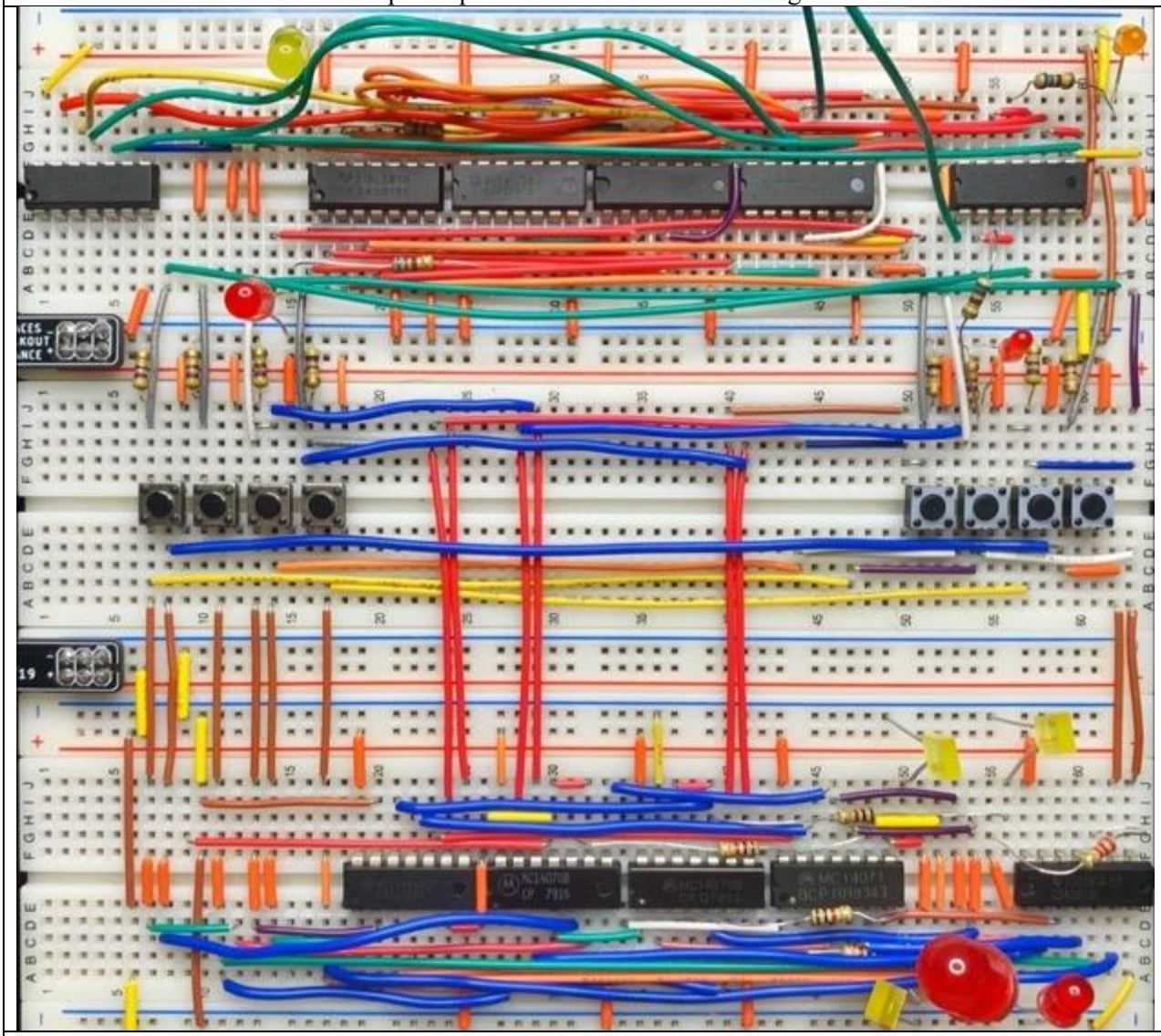
Project Video: https://youtu.be/9WGnbLJ-bnc



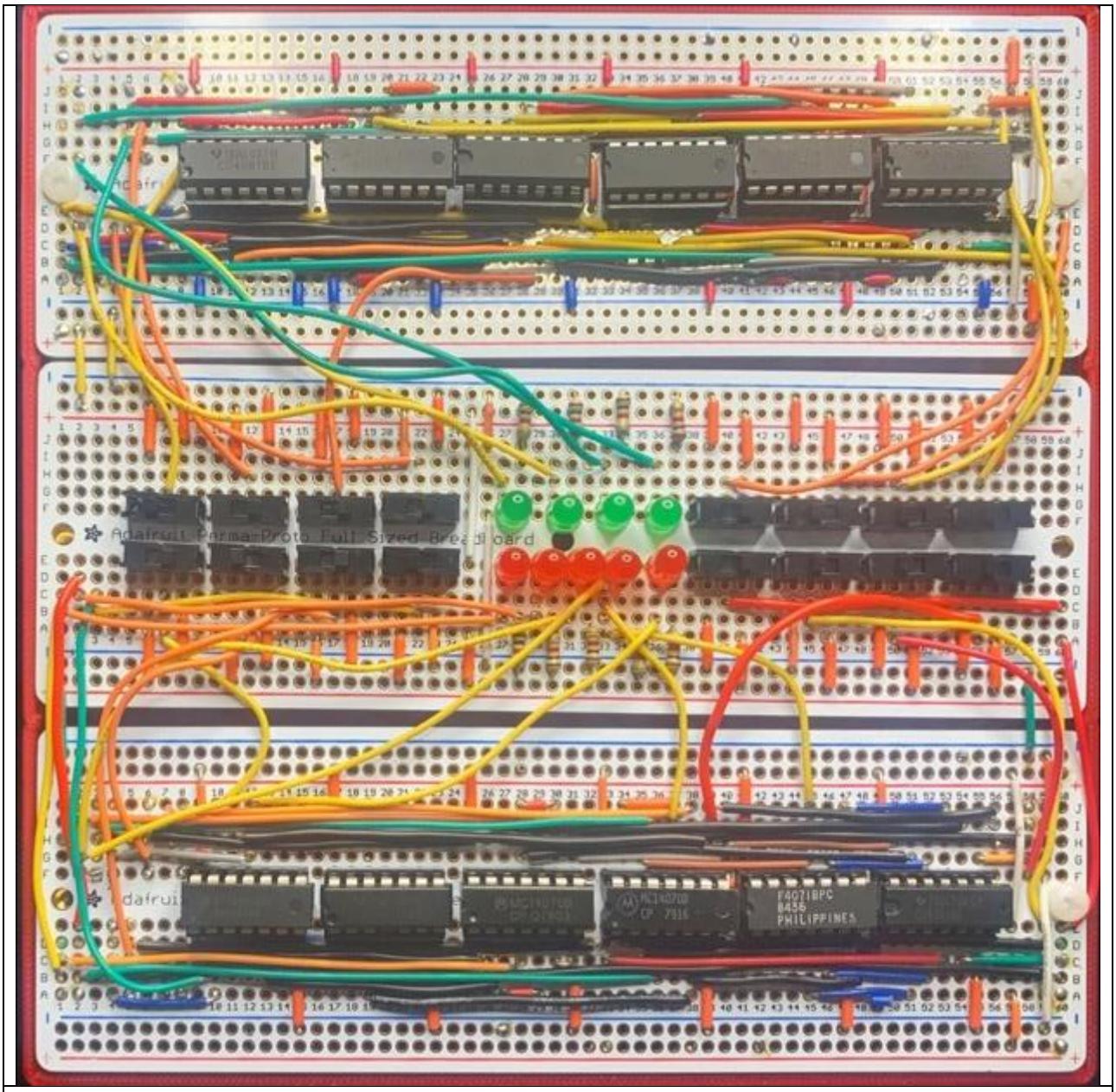
Close Up Of Output Configuration



Close-Up of Inputs: Power and Ground Configuration



Binary Calculator Breadboard Prototype



Binary Calculator Perma Proto Final Design

Reflection

The ISP for me was something I had been thinking about since I decided to pursue grade 11 Hardware. For me, it was such a great learning experience to set my OWN objective and follow through on my OWN project. I emphasize the word own because it taught me that I won't always get a simple project laid out for me and not have to worry about something being too ambitious or easy. I think for the real world and especially engineering it provides a great perspective for what is doable in a correct period. For me, this project was doable but came with sacrifices and a lot of them, sleep especially☺. The end product however is what makes me love this course with something I am super proud to have committed time to and created.

Project 2.6. A Tiny Clock

Project 2.6.1 Inter-Integrated Circuit(I2C)

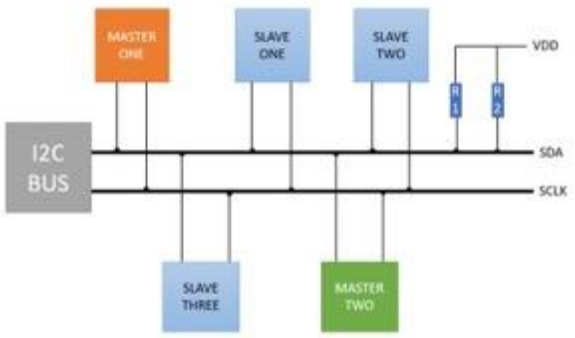
Purpose

The purpose of the I2C Bus is to gain an understanding of how Inter-Integrated Circuit communication functions. The project additionally has the purpose of being the first building block in the ATtiny Clock project.

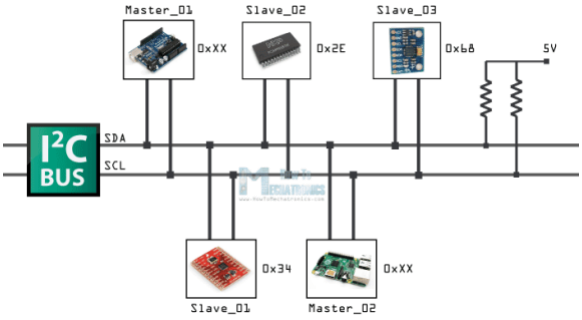
Reference

Project Description: http://darcy.rsgc.on.ca/ACES/TEI3M/Tasks.html#TinyClock
Video on Topic: https://www.youtube.com/watch?v=qeJN_80CiMU

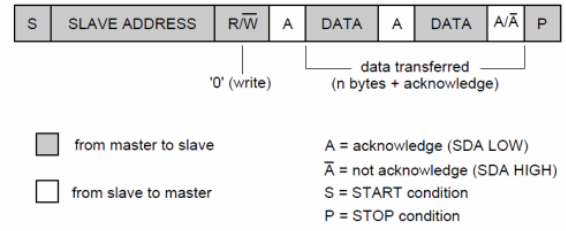
Theory



Inter-integrated communication is a serial communicator that has the capability of controlling 127 slaves with only one master needed. An I2C has two rails, one for SDA the Serial Data Line and the other from SCLK, the serial clock line. The two rails are bi-directional high signal meaning the signal can be transmitted both to and from microcontrollers. The signal is always high so when connecting to a microcontroller the pull-up resistors make it to pull *down* the voltage. The image on displayed the concept of how an I2C works with two rails containing multiple masters and slaves that are all controlled through the communication from the SDA and SCLK.



The I2C function is based on a relationship between the Master and the Slave. The slave's function is based on the address communication provided by the master. The SDA communication, which will be spoken about further in the next paragraph, sends signals through the rails which *every* device reads but only the signal with a match to the address will respond. The master-to-slave communication flows both ways with addresses flowing to slaves while the slaves send and acknowledge. This communication is how the master can distinguish which device the signals are sent to.

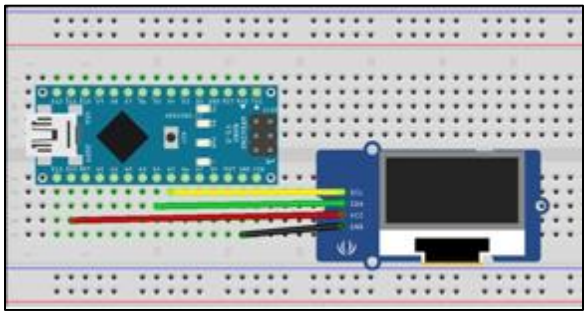


I2C communication begins with a dive into each device used, carrying its identification specific to the device to make it possible to communicate. The wires communication has the two wires function in different ways, the SCLK is synchronizing the data transfers through clock signals while the SDA carries the data. The way the data communicates is by first providing a start signal to let devices know that operations have begun. Next, an 8-bit signal is sent to depict which device is used through an 8-bit Slave address before sending an acknowledge signal to know that the device has completed what it was told to do. Next comes another internal register address which is sent for communication with the internal registers of the device. Lastly, the data is sent through another 8-bit sequence before stopping. Between every sequence, an acknowledge signal is sent

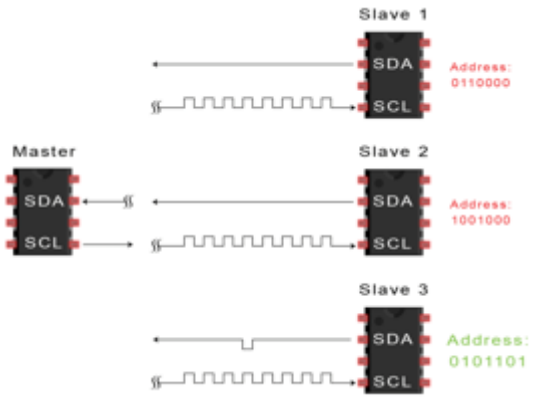
Procedure

Parts Table	
Quantity	Description
1	Arduino Nano
1	TC74 Temperature Sensor
1	DS1307 IC
1	32.768 kHz Oscillator
1	0.01 µF Capacitor
1	CR1220 Lithium Battery
1	CR1220 Retainer

In understanding how the ATtiny clock function the first step is looking at the interesting parts used in the project. The first part is the Arduino Nano which plays the role of being the master in the I2C rails. The temperature sensor is an I2C-compatible device that can access the temperature digitally. The DS1307 IC is the real-time clock used in the step of building the ATtiny clock. This IC is also I2C-compatible. The quartz oscillator is an additional piece to the RTC. When the quartz oscillates 32768 pulses a second the RTC interprets it as 1 second through an electric pulse. The capacitor is used like any capacitor between power and ground rail. Lastly, the CR1220 battery allows the clock to continue keeping the correct time while the power is plugged to the circuit. The three key parts spoken about in this DER are the Arduino's communication with the TC74 and the DS1307. This communication is the I2C communication spoken about in the theory section. The master is the Arduino Nano with pins being the master with the two slaves. This I2C communication is the foundation of how each device works together in the ATtiny clock. The key thing to take away from this communication is that it is both ways as seen in the image. This idea of two-way communication is what makes the code described later in the report to work.



The first bit of the code displayed is the inclusion of the library saying `#include <Wire.h>`. This library has the function of allowing I2C communication. The first step in doing so is setting pins A4 and A5 to the serial data line (SDA) and serial clock line (SCLK). The next step is the functions this library brings in to be used. The functions concerned in this code include `Wire.begin();`, `Wire.beginTransmission();` and `Wire.endTransmission();`. The image on the right shows A4(SDA) and A5(SCLK) being used as an I2C master with an OLED.



The code starts with the begin function which initializes the library into the microcontroller as an I2C controller while also initializing the Serial Monitor this all happens in `void setup()`. Two definitions are made for the number of devices which creates the variable that counts how many I2C devices are in the circuit with a 5000-millisecond interval defined for the hot swap. Next, the statement `for (uint8_t address = 0; address < 128; address++)`. The for statement is used because the maximum value is known being 127 as the maximum address value is $2^7 - 1$. This then goes into the function `Wire.beginTransmission(address);`. This function will send an address provided by the master based on the address given in the parentheses. Once the address matches and is a success a Boolean statement of zero will be provided leading to which serial prints the address in hexadecimal. If this function does not return a result of 0 because there is no I2C then it will serial print an explanation before the interval occurs.

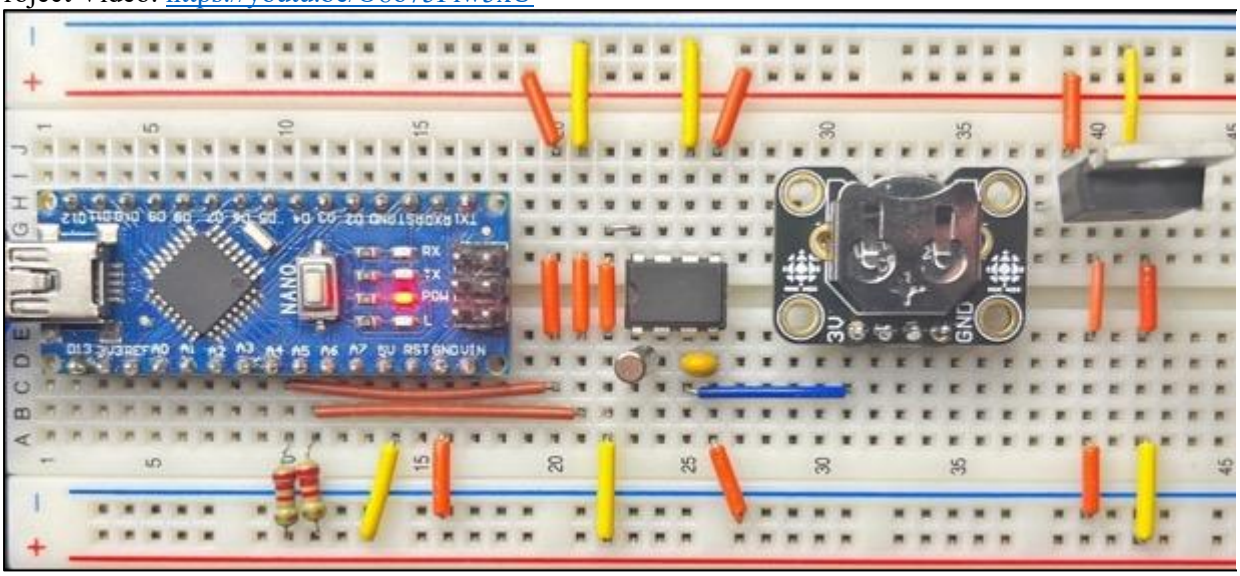
Code

```
// PROJECT :I2CScanner
// PURPOSE :Introduce (gently) 11s to I2C
// Author  :M. Zischka
// COURSE  :ICS3U-E
// DATE    :2024 01 25
// MCU     :328 P
// STATUS  :Working
// NOTES   : The I2C slaves used to respond are the TC74 and the DS1307
#include <Wire.h>      //Core Arduino Library Used for Project
#define INTERVAL 5000  //Delay for Hot swapping
uint8_t numDevices;    //This is the count of the number of devices
void setup() {  //Inout is the default setting
  // put your setup code here, to run once:
  Wire.begin();
  Serial.begin(9600);
  while (!Serial);
}
void loop() {
  numDevices = 0;
  Serial.println("\nScanning our I2C Bus");
  for (uint8_t address = 0; address < 128; address++) {  //checks every value
    Wire.beginTransmission(address);                     //begins checking
    uint8_t result = Wire.endTransmission();             // set 53oolean value
    if (result == 0) {
      Serial.print("I2C Device found at address 0x");
      if (address < 16) //to include a zero infront of values less that 16
        Serial.print(0);
      Serial.println(address, HEX);
      numDevices++;
    }
  }
  Serial.println("There are " + String(numDevices) + " devices on the bus.");
  delay(INTERVAL);  //Hot swap

}
```

Media

Project Video: https://youtu.be/O8o75Plw5xU



View of Breadboard configuration with I2C rails

```
Scanning our I2C Bus
I2C Device found at address 0x4D
I2C Device found at address 0x68
There are 2 devices on the bus.

Scanning our I2C Bus
I2C Device found at address 0x4D
I2C Device found at address 0x68
There are 2 devices on the bus.
```

Serial Monitor Output

Reflection

This project was my second DER with code involved. Since the first project in class and out-of-class activities I have appreciated why software is an important skill, unlike when working on the binary game, I am now very interested in the topic and like this simple DER. What I liked about this idea of the second semester is that it involves the core ideas of Design, Hardware and Software all together and focuses on skill development. I am excited to see where this semester takes me and how I can develop my clock via my skills in this course.

Project 2.6.2 Real Time Clock (RTC) Prototype

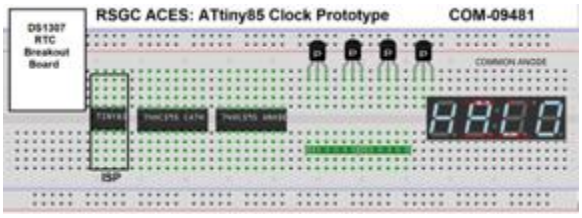
Purpose

The purpose of the Real Time Clock prototype is to develop an understanding of hardware components like the HC595, DS1307, 4-Digit-Seven-Segment Display and ATtiny85. The prototype additionally has the purpose of developing supporting code for the hardware components.

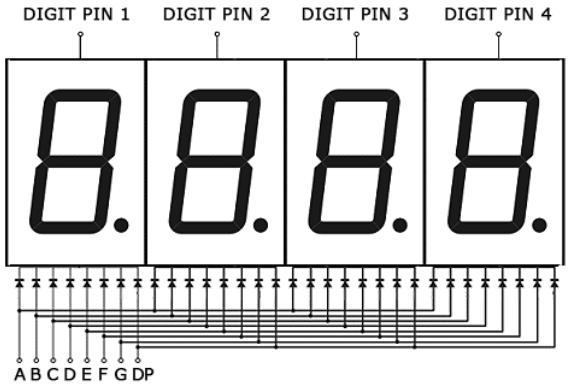
Reference

Project Description: http://darcy.rsgc.on.ca/ACES/TEI3M/Tasks.html#Prototype
4 Digit 7 Segment Display Data Sheet: https://www.sparkfun.com/datasheets/Components/LED/7-Segment/YSD-439AB4B-35.pdf
DS1307 Data Sheet: https://www.analog.com/media/en/technical-documentation/data-sheets/ds1307.pdf

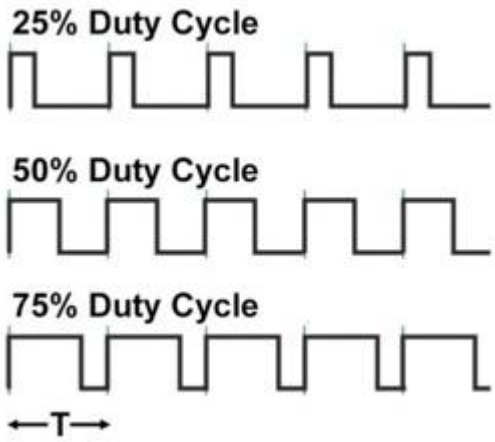
Theory



The theoretical idea behind the real-time clock Prototype is the idea of how it is possible to display values on the 4-digit-seven segment display. The problem behind a display like this is that it has two components; common anodes and common cathodes. This is a problem because not many values can be displayed with the correct segments only have common anodes and cathodes. This is a similar problem to displaying a diagonal line on an LED matrix. The solution when using devices like this is to use persistence of vision.



Persistence of vision is based on the idea that the human brain can only process around 10 to 12 images per second. After this time frames begin to form together as if it is only one image and not a blinking image. The fixture to this problem is to have each segment shift from segment to segment. If done fast enough, which should be achievable with the ATtiny85 it should be able to solve the problem. The problem being solved is to have the display show all segments in an extremely short time so the display looks like it is fully on not blinking. The approach to completing this goal will be explained in the procedure below.



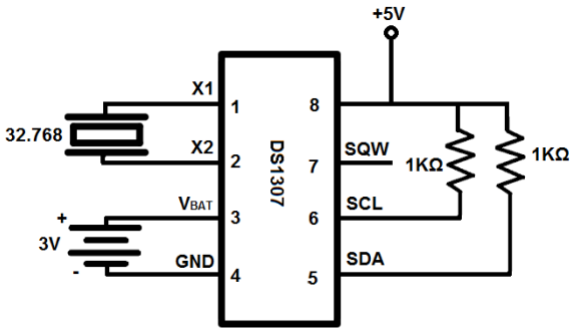
The other sub-concept is understanding how setting brightness is achievable. The duty cycle can be described as the mathematical equation for a square wave that states $Duty\ Cycle = Time\ On \div Total\ Time$. This can be seen in the image on the right as well. This is done because of PWM (pulse width modulation). This says that the width of the pulse alters how bright the display is. The display at 100% duty cycle is too bright for a clock. The fixture to this is using the shift registers to load the display and then load a blank display in equal time yet still produces more than 12 images per second to retain a complete value while having a lower brightness due to PWM.

Procedure

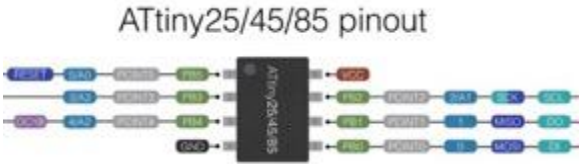
As already spoken about, the goal of the real-time clock prototype is to display real-time through the seven-segment display. What it takes to complete this goal is three components starting at the DS1307 then moving through the ATtiny before being shifted to the 7-segment display. Some less important parts to be discussed include the 0.01µF which acts as a supporting component for the quartz oscillator. The CR1220 is a 3V coin cell battery that powers the DS1307 and additional components while they are not being powered by the ATtiny's power. Lastly, the 10 kΩ resistors are paired with 3906 transistors connected to the base. This is done not to exceed the "safe limit" of current allowed to supply the base pin.

Parts Table	
Quantity	Description
1	ATtiny85
2	HC595 Shift Register
4	3906 PNP Transistor
1	DS1307 IC
1	CR1220 Lithium Battery
1	0.01µF Capacitor
1	32.768 kHz Oscillator
1	4 Digit 7 Segment Display
4	10 kΩ Resistor

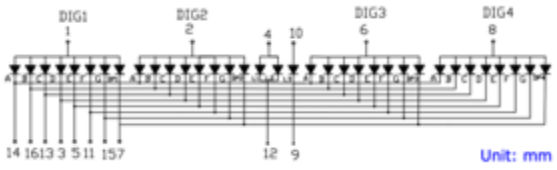
One major component of the real-time clock is the DS1307. This IC is built on several components including 8 pins that are all used. The power and ground components use one ground source but two power supplies one from the ATtiny and one from the coin cell battery. The next component used to support the DS1307 is the 32.768 kHz quartz oscillator. This hardware component when powered by voltage has a quartz crystal vibrate 32,768 times per second or 2^{15}. With these pulses being inputted the DS1307 internally divides this value down to 1 Hz per second. This makes it possible for the IC to keep track of the time. The square wave pin is a confirmation pin that the DS1307 is keeping track of the time in receiving the 1 Hz which is used in the real-time clock as the colon flasher.

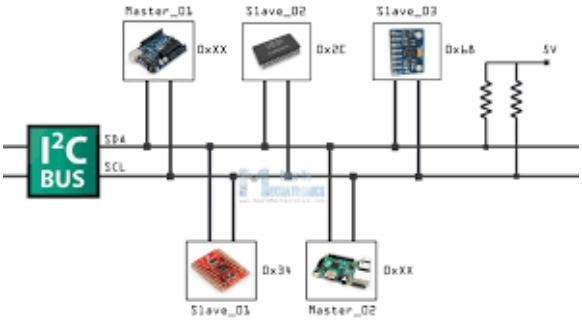


The next major hardware component of the real-time clock is the ATtiny85. This hardware component is in substitution for the Arduino Nano which holds a 328P as the microcontroller. For the prototype, the pins necessary are serial data, serial clock, clock pin, data pin and latch pin. More efficiently the data pin and serial clock pin can be done in one. This means only 7 pins are necessary, which makes the ATtiny85 the perfect microcontroller. The one disadvantage of the ATtiny is that the serial monitors and USB connection are no longer accessible. The fixture to this is to use the Spark Fun programmer with a connection through ISP pins.

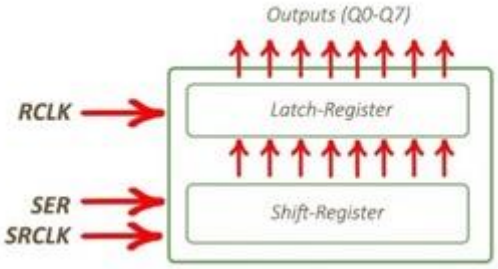


The last major hardware component is the 4-digit 7-segment display. The configuration as already spoken about in the theory is a common anode configuration. The common cathodes are wired to each segment while each anode is wired to each digit. The configuration is achievable by using two HC595s. One 595 is wired for the cathodes and one is for the anodes. This is achievable by wiring 595s to the ATtiny with the clock pin and latch pin in parallel and the data pin wired through pin 9 on the 595.

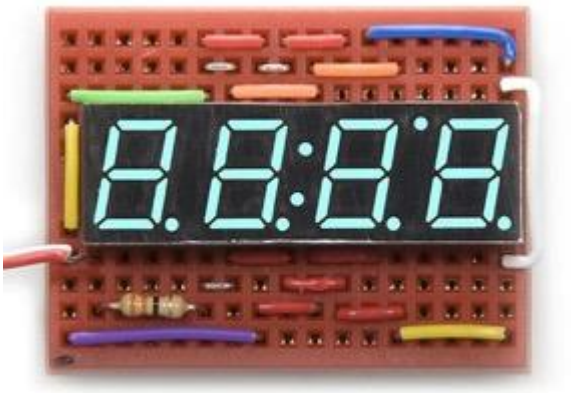




Now with an understanding of how the hardware components work it is time to understand how the software configures each hardware piece to reach the goal of displaying the real-time. The first step is using I2C communication to do this. As spoken about in the DER above I2C uses two rails serial clock and serial data. For the real-time prototype, the only I2C communication necessary is setting the real-time and then retrieving it from DS1307. This is all done through two simple wires between the SDA and SCL on both components. On the software side, a few lines make this possible first enabling the library; `#include <TinyWireM.h>`. Next the import line is `TinyWireM.RequestFrom(RTCADDRESS,2);`. This communicates with DS1307 telling it how many bits of information it needs, in this case, it needs the information on the minutes and hours. In addition, I2C is used to retrieve the temperature every 10 seconds. This is done with an extremely similar line `TinyWireM.RequestFrom(TC47ADDRESS,1);`. The only change is what device the I2C is communicating with and how many bits it is retrieving.

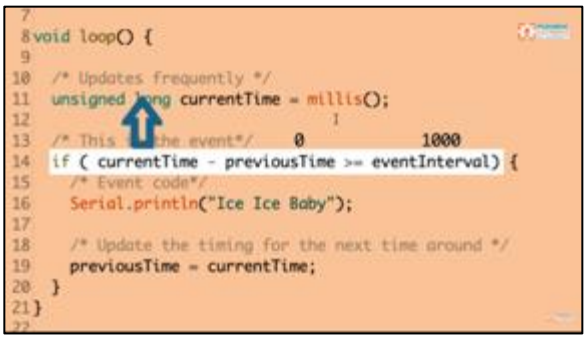


Now with information retrieved the next step involves displaying the information to the 4-digit 7-segment display. As explained in the procedure the ATtiny is connected to two 595s with outputs connected to the common cathodes and common anodes. With this in mind, it is possible to create a goal of what the hardware components should do. In this case, the 595s need to be pulled low before being loaded with one 595 out loading the correct configuration of segments for one of the 4 digits and repeating for all four. Have this repeat at a rate fast enough for the persistence of vision to occur.



The first step in writing out the code is a necessary for statement. A for statement is used because the stamen will be run 4 times for each common anode, if the number of times the statement was being run is unknown then a while statement would be used. The next step is the code written in this for statement. After pulling the latch low the line `shiftOut(DATA, CLOCK, LSBFIRST, ~(1 << digit));` is written. This is done to set one of the 4 common anodes. This is followed by the line `shiftOut(DATA, CLOCK, LSBFIRST, ~buffer[digit]);`. This sets the segments for each segment on each digit. When this command is run in a loop function persistence of vision occurs and the time is set. The next step involves adjusting the pulse width to lower the brightness of the display. The way to this involves adjusting the duty cycle, a concept discussed in the theory. To do this intervals need to be created like `delay(TIMEON);`. This sets a specific amount of time for the display to be on and then off. In this situation, it is on equal time it is off providing a 50% duty cycle.

The software component knows as `millis` is used to swap my displays from display to display. The `millis()` function is used to return the number of milliseconds since the Arduino board began running the current program. It is often used for timing and creating delays without blocking the execution of other code. In this situation the `millis()` function is run every 10 seconds before displaying the time then displayed the date after another 100 seconds and does this in loop.



Code

```
// PROJECT  :4Digit7SegmentDisplayRTC
// PURPOSE  : Display time,temperate and date
// COURSE   :ICS3U
// AUTHOR   :In Class Code
// DATE     :2024 02 22
// MCU      :'85
// STATUS   :Working
// REFERENCE: http://darcy.rsgc.on.ca/ACES/TEI3M/Tasks.html#Prototype
#include <TinyWireM.h>
#define RTCADDRESS 0x68  //DS1307I2C address
#define TC74ADDRESS 0x4D
#define DATA 0                      //  SDA:0, SCL:2
#define LATCH 3                     //
#define CLOCK 4                     //
#define INTERVAL 4                  // former delay duration
#define TIMEON 1                    //  The TIMEON/TIMEOFF ratio changes the duty
cycle
#define TIMEOFF 1                   //  of the display, hence the brightness.
#define SQWREGISTER 7               //square wave register
#define SQWE 4                      //ENABLE
#define RS0 0                       //
#define RS1 1                       //
#define HZ1 0 << RS1 | 0 << RS0     //1   Hz
#define HZ4K 0 << RS1 | 1 << RS0    //4k  Hz
#define HZ8K 1 << RS1 | 0 << RS0    //8k  Hz
#define HZ32K 1 << RS1 | 1 << RS0   //32k Hz
uint8_t segments[] = {
  //7-segment digit map
  //B0abcdefg
  B01111110,  //0
  B00110000,  //1
  B01101101,  //2
  B01111001,  //3
  B00110011,  //4
  B01011011,  //5
  B01011111,  //6
  B01110000,  //7
  B01111111,  //8
  B01110011,  //9
  B01100011,  //Degrees
  B01001110,  //Celcuis
};
uint8_t Date1segments[]{
  //first letter of date
  B01011000,  //j
  B01000111,  //f
  B01010101,  //m/n
  B01110111,  //a
  B01110110,  //m
```

```
  B01011000,  //j
  B01011000,  //j
  B01110111,  //a
  B01011011,  //s
  B01111110,  //o
  B01110110,  //n
  B00111101,  //D
};
uint8_t Date2segments[]{
  //second letter of date
  B01110111,  //a
  B01001111,  //e
  B01110111,  //a
  B01100111,  //p
  B01110111,  //a
  B01111110,  //u
  B00001110,  //l
  B01111110,  //u
  B01001111,  //e
  B01001110,  //o
  B01001110,  //o
  B01001111,  //e
};
uint64_t millisTime;  //allows millisTime to equal millis
uint8_t seconds = 0;  //hard code a time and date for simplicity
uint8_t minutes = 12;
uint8_t hours = 10;
uint8_t day = 4;  //1-Monday...7-Sunday
uint8_t date = 22;
uint8_t month = 2;
uint8_t year = 24;
uint8_t buffer[] = { 0, 0, 0, 0 };  //intermediate data holding array
void setup() {
  pinMode(DATA, OUTPUT);
  pinMode(CLOCK, OUTPUT);
  pinMode(LATCH, OUTPUT);
  TinyWireM.begin();
  setTimeDate(seconds, minutes, hours, day, date, month, year);
}
void loop() {
  millisTime = millis();
  while (millis() - millisTime < 10000) {
    getTimeDate();  // update the (time) buffer
    showTime();     // display the time
  }
  millisTime = millis();
  while (millis() - millisTime < 1000) {
    displayTemp();
  }
  millisTime = millis();
  while (millis() - millisTime < 100000) {
    getTimeDate();  // update the (time) buffer
    showTime();     // display the time
  }
  millisTime = millis();
  while (millis() - millisTime < 1000) {
    getAndShowDate();
  }
}
// Function to write time and date data to the RTC
void setTimeDate(uint8_t s,
                 uint8_t m,
                 uint8_t h,
```

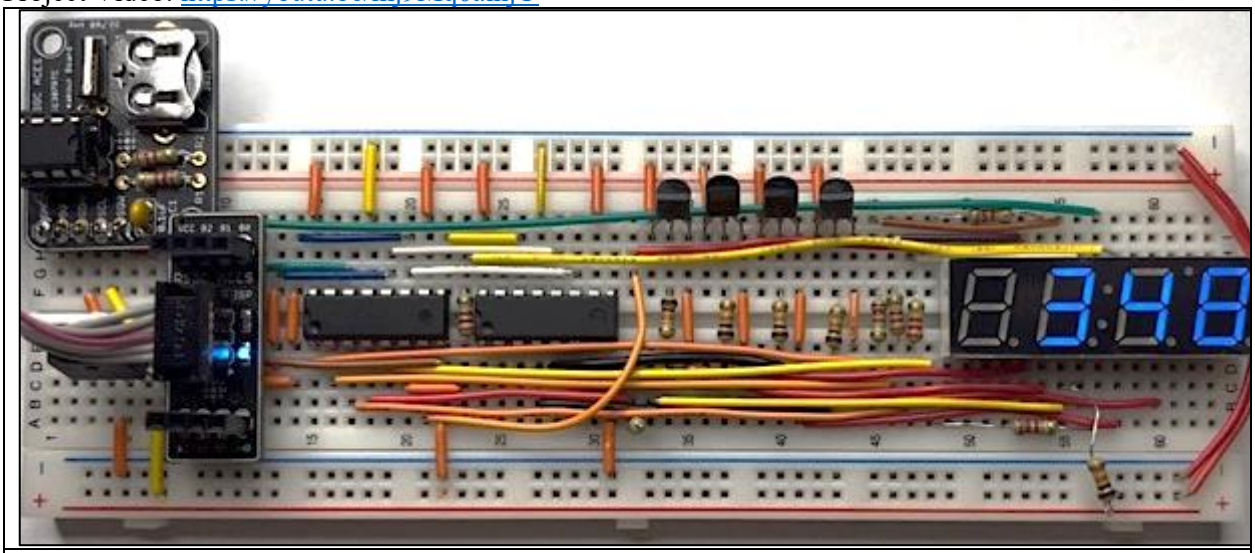
```
                  uint8_t day,
                  uint8_t date,
                  uint8_t mon,
                  uint8_t y) {
  uint16_t hrs, mins, secs, dat, yr;                 //variables must be 16 bits for
sscanf function
  sscanf(__TIME__, "%d:%d:%d", &hrs, &mins, &secs);  //variables must be uint16_t
  char Month[4];
  sscanf(__DATE__, "%s %d %d", Month, &date, &yr);  //scans real time
  TinyWireM.beginTransmission(RTCADDRESS);
  TinyWireM.write(0);  //start writing at first preferred Register
  TinyWireM.write(DEC2BCD(/*s*/ secs));
  TinyWireM.write(DEC2BCD(/*m*/ mins));
  TinyWireM.write(DEC2BCD(/*h*/ hrs));
  TinyWireM.write(DEC2BCD(day));
  TinyWireM.write(DEC2BCD(dat));
  TinyWireM.write(DEC2BCD(/*mon*/ month));
  TinyWireM.write(DEC2BCD(/*y*/ yr));
  TinyWireM.endTransmission();
}
// Function to confirm the time and date were written successfully
void getTimeDate() {
  TinyWireM.beginTransmission(RTCADDRESS);
  TinyWireM.write(1);  //start reading from the first required Register
  TinyWireM.endTransmission();
  TinyWireM.requestFrom(RTCADDRESS, 2);  //tell the device how much data is required
  while (TinyWireM.available() == 0);//wait for the device to place in read buffer
  minutes = BCD2DEC(TinyWireM.read());
  hours = BCD2DEC(TinyWireM.read());
  hours = hours % 12;
  if (hours == 0) hours = 12;
  if (hours > 9) buffer[0] = segments[hours / 10];  //10 dig of hrs
  else buffer[0] = 0;                               //if a zero nothing displays
  buffer[1] = segments[hours % 10];                 //1s dig of hrs
  buffer[2] = segments[minutes / 10];               //10s dig of mins
  buffer[3] = segments[minutes % 10];               //1s dig of mins
}
void showTime() {
  for (uint8_t digit = 0; digit < 4; digit++) {
    //   for (uint8_t i = 0; i < 8; i++) {
    digitalWrite(LATCH, LOW);
    shiftOut(DATA, CLOCK, LSBFIRST, ~(1 << digit));
    shiftOut(DATA, CLOCK, LSBFIRST, ~buffer[digit]);
    digitalWrite(LATCH, HIGH);
    delay(TIMEON);  //duty cycle
    digitalWrite(LATCH, LOW);
    shiftOut(DATA, CLOCK, LSBFIRST, 0x0F);
    shiftOut(DATA, CLOCK, LSBFIRST, 0xFF);
    digitalWrite(LATCH, HIGH);
    delay(TIMEOFF);  //duty cycle
  }
}
void setSQW(uint8_t rate) {  //set 1Hz sqw
  TinyWireM.begin();
  TinyWireM.beginTransmission(RTCADDRESS);
  TinyWireM.write(SQWREGISTER);
  TinyWireM.write(1 << SQWE | rate);
  TinyWireM.endTransmission();
}
uint8_t DEC2BCD(uint8_t value) {
  return (value / 10 << 4) + (value % 10);  //bracket for clarity
}
uint8_t BCD2DEC(uint8_t value) {
```

```
  return (value >> 4) * 10 + (value & 0x0F);
}

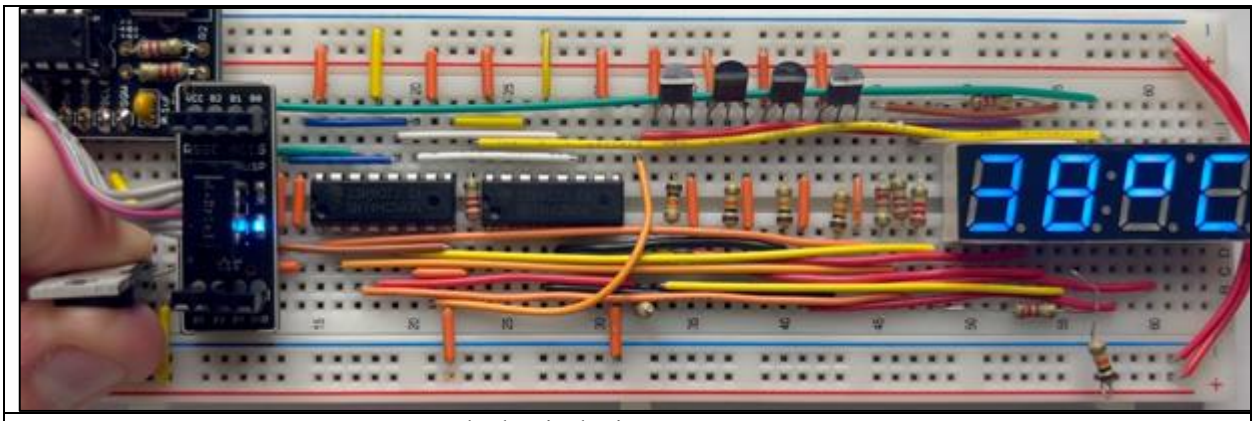
void displayTemp() {                              //retrieve and displays temp
  TinyWireM.beginTransmission(TC74ADDRESS);  //Start a conversation with the sensor
  TinyWireM.write(0);                             //Interested in reading from register 0
  TinyWireM.endTransmission();                //That's it for the moment
  TinyWireM.requestFrom(TC74ADDRESS, 1);     //Interested in 1 byte from register 0
  while (TinyWireM.available() == 0)
    ;                                          //Wait for data to appear in the buffer
  int8_t celsius = TinyWireM.read();    //Read it
  buffer[0] = segments[celsius / 10];   //display 10s dig
  buffer[1] = segments[celsius % 10];   //display 1s dig
  buffer[2] = segments[10];
  buffer[3] = segments[11];
  for (uint8_t digit = 0; digit < 4; digit++) {
    digitalWrite(LATCH, LOW);
    shiftOut(DATA, CLOCK, LSBFIRST, ~(1 << digit));
    shiftOut(DATA, CLOCK, LSBFIRST, ~buffer[digit]);
    digitalWrite(LATCH, HIGH);
  }
}
void getAndShowDate() {  //retrieves mon and dat then displays
  TinyWireM.beginTransmission(RTCADDRESS);
  TinyWireM.write(4);  //start reading from the fourth required Register
  TinyWireM.endTransmission();
  TinyWireM.requestFrom(RTCADDRESS, 2);  //tell the device how much data is required
  while (TinyWireM.available() == 0)
    ;  //wait for the device to place the data in the read buffer
  date = BCD2DEC(TinyWireM.read());
  month = BCD2DEC(TinyWireM.read());
  buffer[0] = Date1segments[month];  //displays first letter
  buffer[1] = Date2segments[month];  //displays second letter
  buffer[2] = segments[date / 10];
  buffer[3] = segments[date % 10];
}
```

Media

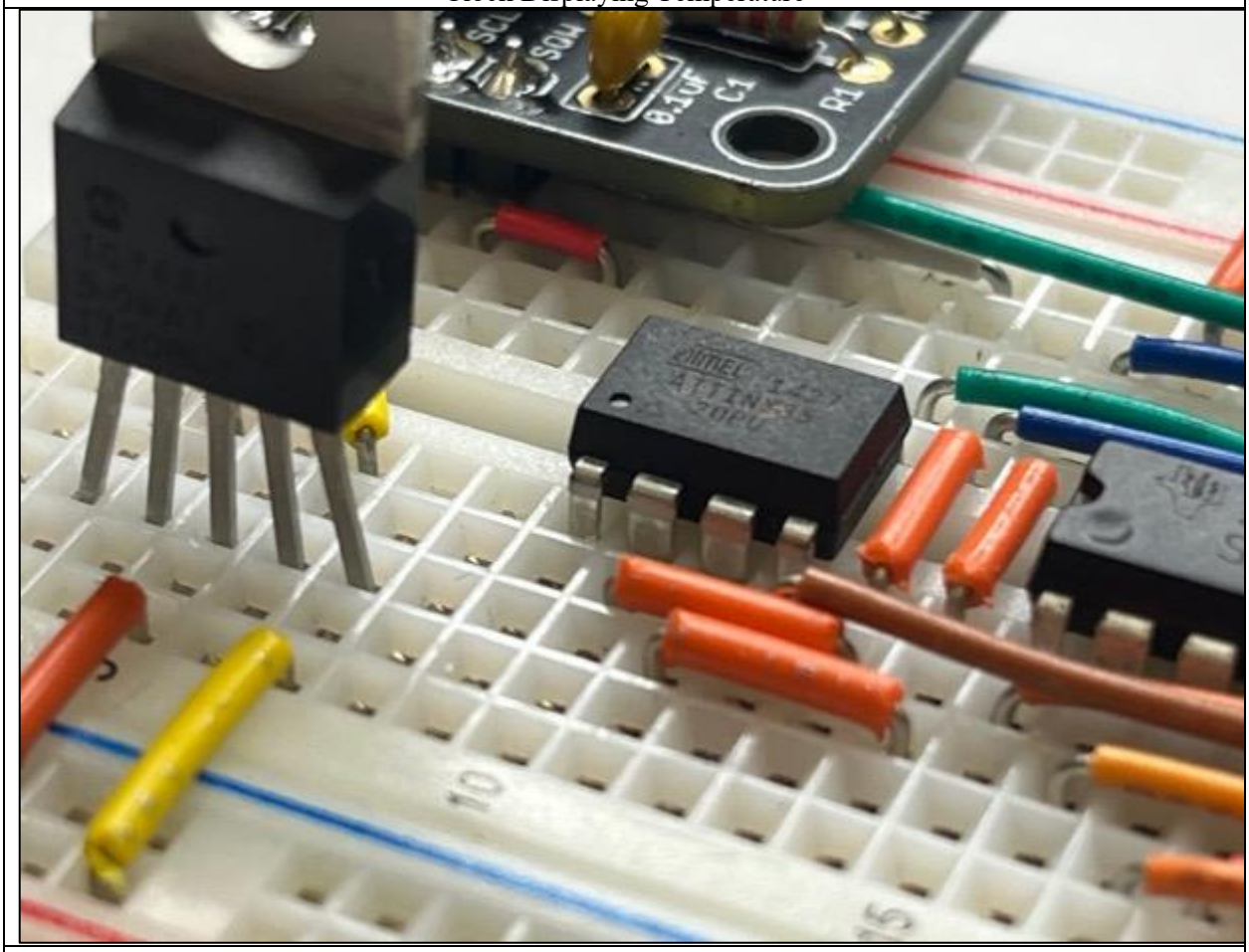
Project Video: https://youtu.be/mj9Mq0alhjY



Clock Displaying Time



Clock Displaying Temperature



ATtiny85 Close-Up

Reflection

This DER saw a lot of stress between ISP work, DER work and everything else life has to offer. Which meant the greatest skill I was able to work on was time management. The actual DER was an extremely fun project between the software involvement, which I love and the end when I received a clock was really just a fun experience. Going above and beyond on this DER was a goal I had set out when the due date was extended but after setting the goal the work was hard but the lessons learned were amazing.

2.7.3 Medium ISP The Magic Chessboard

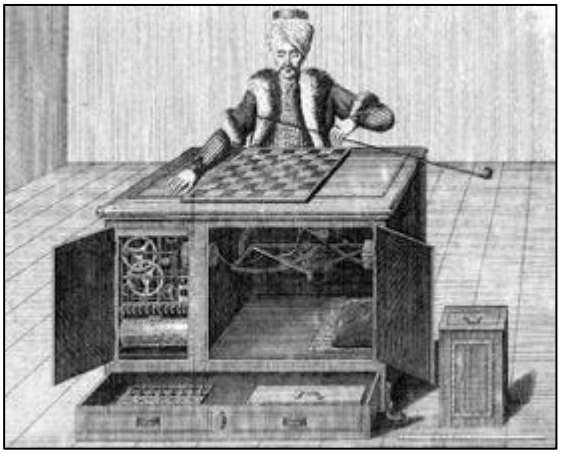
Purpose

The purpose of the magic chessboard is for a person to be able to play against a chess engine. The move is made with the help of an electromagnet. This purpose is achieved by move tracking through multiplexers, chess evaluation from a Raspberry Pi and magnet movement through stepper motors.

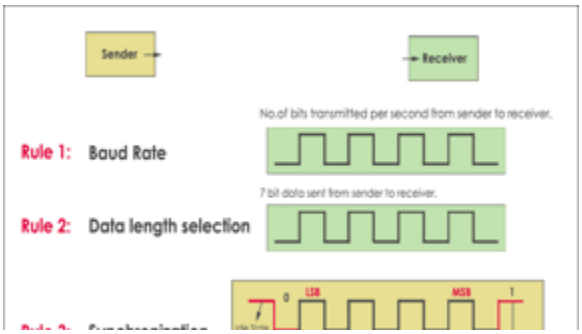
Reference

Inspiration: https://www.instructables.com/DIY-Super-Smart-Chessboard-Play-Online-or-Against-/
CoreXY Explanation: https://3ddistributed.com/corexy-3d-printer/corexy-belt-routing/
A4988 Explanation: https://www.youtube.com/watch?v=5CmjB4WF5XA&t=271s
3-D Print and Tools Assistance: https://www.instructables.com/Automated-Chessboard/
Multiplexer Explanation: https://www.youtube.com/watch?v=j_ZP_zd0WHI
Chess Engine Information: https://www.raspberry-pi-geek.com/Archive/2016/20/PyChess-A-chess-program-for-your-Rasp-Pi
Serial Communication Explanation: https://www.youtube.com/watch?v=OJtpA_qTNL0
Raspberry Pi Setup: https://www.tomshardware.com/how-to/set-up-raspberry-pi
Stepper Motor Explanation: https://www.monolithicpower.com/stepper-motors-basics-types-uses#:~:text=The%20basic%20working%20principle%20of,rotor%20aligns%20with%20this%20field.
How to Run Program on Boot: https://www.youtube.com/watch?v=5ujwF4dB4Fk&t=1s

Theory



The theory behind a complex project like a self-moving chessboard involves several theoretical ideas with the first involving how the board can move from retrieving the move made by white to getting the best move the black pieces and then making it. The theoretical idea involves reed switches providing either the value of 1 or 0 made possible by 10 kΩ pull-down resistors. The coordinate made by the Arduino is sent to the Raspberry Pi via Serial communication between the USB port on the Arduino to the USB port on the Pi. After going through the code which is presented later in the DER. Lastly, through several if statements the electromagnet makes the correct displacement before turning on the electromagnet moving the piece. The image on the right shows how this ambitious project based around an old game like chess has been a pursued engineering project for some time.



The next theoretical idea to understand is how serial communication works. Serial communication can be wired in two ways being hardwired to the pins on microcontrollers like Arduino and on computers like the Raspberry Pi however serial communication is how code can be uploaded so it is possible to use the serial ports for serial communication. Every serial communication needs two things a TX which is the device that is transmitting the information and then an RX for receiving. The image on the right illustrates how this can be done. Matching baud rates between the two components will have them receiving and transmitting at the same rate before the binary information is sent.

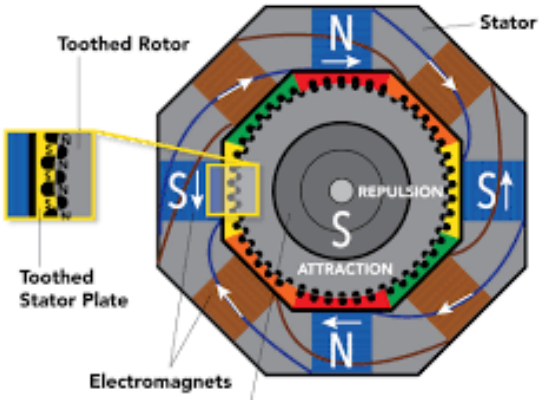
Procedure

Parts Table	
Quantity	Description
1	Arduino Nano
2	Nema 17 Stepper Motor
2	A4988 Motor Driver
1	Raspberry Pi 4
4	16 Channel Multiplexer
1	12V Electromagnet
1	TIP Power Transistor
64	Reed Switch
1	PBNO Button
5	Aluminum Rails
8	V-Slot Wheel

The parts used in the creation of the magic chessboard involve the major pieces displayed on the parts table with the minor pieces not documented. The minor pieces include things like all the additional parts necessary for creating a coreXY motion. One of the most used pieces was Tee Nuts which are nuts that can be slotted into the rails making it possible to attach 3-D prints, the electromagnet and corner pieces. Other pieces include spacers and washers that provide an essential role in the functioning of the circuit. This includes the use of eccentric spacers a piece that allows an adjustable amount of friction provided to the aluminum rails from the wheels for clean motion. Washers are another essential piece in this build for not allowing friction to obstruct the motion necessary for the chessboard. This is especially useful at the points when the teethed belt is turning. To allow for no motion obstruction washers are used.

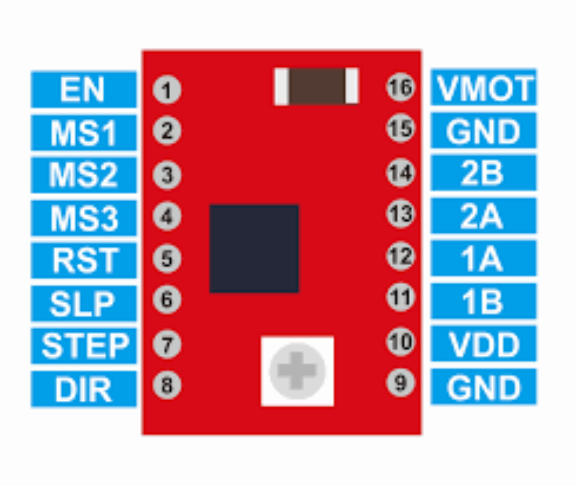


The first major component of the magic chessboard to be discussed is the 16 Channel Multiplexer. The multiplexer is used to allow for each of the 64-reed switches to be read without using too many Arduino nano pins. This is achievable by wiring the 4 switch pins (S0, S1, S2 and S3) to the Arduino along with the SIG pin. The way the multiplexer works is by digitally inputting high or low signal to the 4 switch pins to create a binary value. This binary value in decimal is the channel that the sig pin can read in that moment. So, if S0, S1 and S2 all have high value but S3 has a low value then the sig pin can read the state of channel 7 in that moment. For further pin conservation because the code is doing the same output for all 4 mux boards, the switch pins can be wired in parallel and instead use the enable pin to see which mux is being read.

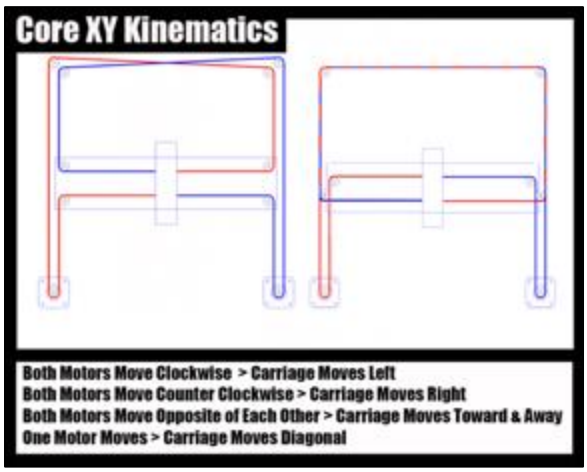


The next idea to be discussed is how the stepper motor functions. The image on the right displays how the stepper motor can rotate. Coils wired to electromagnets are continuously receiving high and low inputs. These high inputs are continuously cycling clockwise. The permanent magnet, with the rotator attached, follows these high inputs to the magnet making it rotate. The A4988 makes these correctly timed electromagnet pulses possible by having the 4 coils wired to the IC and allowing the Arduino to control the steps made. With the Nema 17 stepper motor each electromagnet pulse moves the rotator 1.8 degrees. This precise motion is what makes the stepper motor a perfect component for precise motion of each chess square. The motor driver and software behind this component will be discussed later in the report.

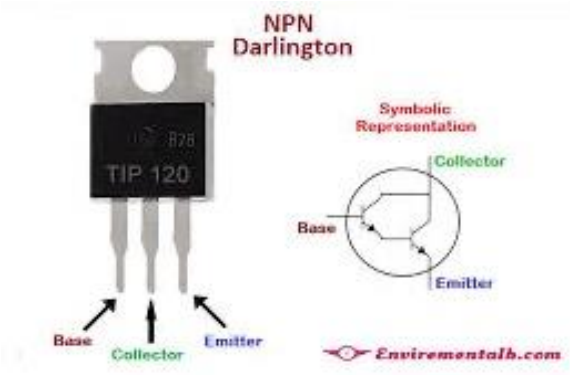
The pairing component to the Nema 17 stepper motor is the A4988 motor driver. A motor driver is the only way to get a stepper motor to move. The pinout of this driver is displayed on the right. The important pins include the coil pins, the two power supplies, the step pin and the direction pin. The other pins are not important for the function of the magic chessboard. The MS pins are used if more precision is necessary and can provide up to 1/16 of a step if configured correctly. For normal function, the reset and sleep pins are wired together. The step pin and direction pin are wired to the microcontroller to enable which direction is high for clockwise and low for counter-clockwise. The step pin provides the PWM (pulse width modulation) which when provided a high frequency will have a faster motion. However, the duty in this case has to stay at 50% meaning equal microsecond delays between the high signal and low signal.



The next key concept to be discussed is the CoreXY configuration that the electromagnet moves with. The kinematics of this system involve putting two stepper motors at the bottom right and bottom left of the system. Next, the leads of the belt need to be connected to opposing points on the electromagnet cart. This means if one of the leads is connected to the bottom right of the cart then the other lead is connected to the top left. This configuration allows for horizontal motion if both motors are moving in the same direction, vertical motion if both motors are moving in opposing directions and diagonal motion if only one motor is moving. Stepper motors are the best motor for this motion as the moves from square to square only require about 31mm of motion. Furthermore, the stepper needs to be calibrated and can sometimes pull more in one direction during a long motion. To fix this every single motion moves one square before a half-second delay so that one motor doesn't pull more than the other.



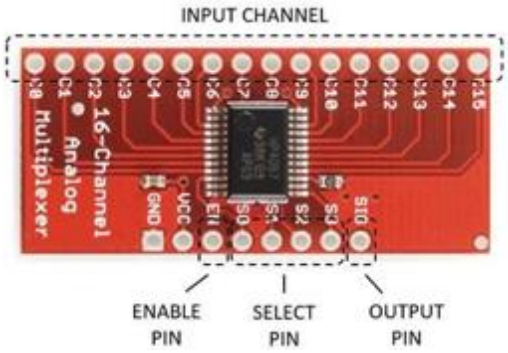
The electromagnet under the board is the centerpiece of this project being the main objective of moving the pieces. However, between moves, the electromagnet needs to move across the board to get from the end position to the next starting position. To do this a power transistor is put into use to control when the electromagnet receives the power it needs. The transistor used is the TIP 120. This is a power transistor which can handle 12V with ease. In this case, the 12V power supply is powered to the collector with the electromagnet lead connected to the emitter and the 5V Arduino supply is connected to the base allowing when the electromagnet should be powered and when not.



With an understanding of all the devices it is now possible to begin understanding how software can complete the purpose of this project. The first part of the code that needs to be discussed is the global variables. The global variables include pin definitions and a few arrays. The pin definitions are put in arrays with the goal of cycling through pin variable as some alter due to the for loops making them. An example of this is found in the `muxOutputPins[]` this variable is continuously changing in the `for` loops as it cycles though each multiplexer.

```
uint8_t muxPins[] = { 3, 4, 5, 6 };           // multiplexer switch pins
uint8_t enablePins[] = { 7, 8, 9, 10 };       // multiplexer enable pins
uint8_t muxOutputPins[] = { A0, A1, A2, A3 }; // multiplexer sig pins
volatile bool triggered = false;              // boolean statement for attach interrupt
uint32_t moveTime = 5000;                     // time for pins to scan
uint32_t previousStatesArray[4][16] = { 0 };  // array to track changes
uint32_t lastTimeChangeArray[4][16] = { 0 };  // array to track time changes
uint32_t startTime;                           // variable for millis
#define magnet A5
#define dirPin 12  // all A4988 pin definitions
#define stepPin 11
#define dirPin2 A4
#define stepPin2 13
```

The next key part of the code to discuss is the `for` statement used to scan each channel on the multiplexer. The first `for` statement calls the function `void selectChannel(int channel)` this function can digitally write a high signal in binary to the specific pins. This can be done through the `bitRead(x,n);` function which takes the specific value being the x value and the n which is the specific bit it takes so `bitRead(11,3);` will output a 0 because because 11 in binary is 1011 and `bitRead` is reading that third bit. The other key statements involve `if (state != previousStatesArray[mux][channel]) {` this is the way that the mux detects if the values have changed from the previous array. When the mux is first turned on the 32 values go through a state change which updates the array but doesn't link moves. Lastly, the code iterates through two statements that say being a `for` and an `if` statement. These have the role of scanning through the array to find the values that detected change before `Serial.print(chessArray[mux * 16 + channel]);` and `Serial.println(chessArray[mux * 16 + linkedChannel]);` which can be reversed based on move timing.

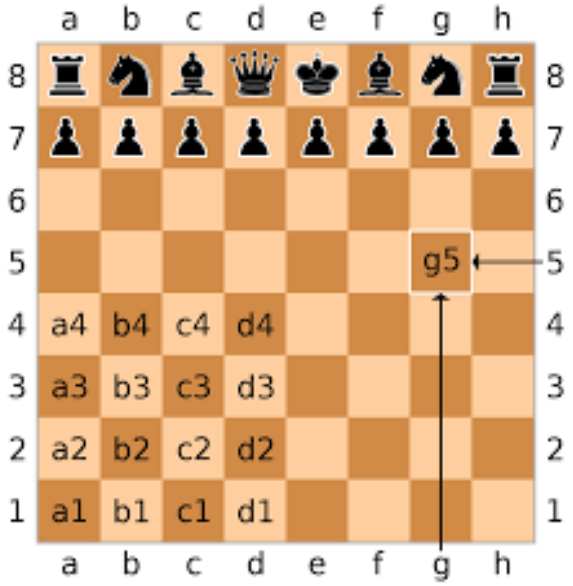


The next part of the code is what happens with the Raspberry Pi. The code illustrated is taking a new line print through the serial monitor and evaluating with a stockfish chess engine. The way it is able to read a new line printed by the Arduino is by the python instruction `if ser.in_waiting > 0: line = ser.readline().decode().strip()`. This is reading the newest line before evaluating it with the chess engine `result = engine.play(board, chess.engine.Limit(time=0.1))`. What this line does is set a specific time for the chess engine to make its move and also what board it is evaluating. This board is constantly updating. Finally the best move is then sent back to the Arduino with the line `ser.write(str(best_move).encode() + b'\n')`.



Once this move is sent back to the arduino the arduino must break it down into numbers that it can do addition and subtraction with to decide how many squares to move. This can be done by subtracting the character letters by 97 or 'a' and the character numbers by 48. Example: `uint8_t first column = move[0] - 'a' +1`

```
uint8_t firstColumn = move[0] - 'a' + 1;  // breaking up the move
uint8_t firstRow = move[1] - 48;
uint8_t secondColumn = move[2] - 'a' + 1;
uint8_t secondRow = move[3] - 48;
```



Finally, the motor must know how to move. This is done by firstly deciphering what type of move is being made. A vertical move will only have a change in the number of rows, a horizontal change will only have a change in the columns and a diagonal or L-shaped move will have both changes in the rows and columns. The way to decipher which type of move it is by creating if statements that match the type of moves described earlier. An example of this would be if a vertical move like e2e4 is made then it would match the statement. `if (firstColumn == secondColumn && firstRow != secondRow)` once matched with the correct if statement it moves to the starting position by subtracting the starting position from d8 with the function `void moveToStart(uint8_t firstColumn, uint8_t firstRow) {` . This is when the magnet turns to high and it horizontally moves the amount of displacement between the starting column and ending column in this case. Lastly, the magnet is turned low and an opposing function returns the electromagnet to d8.

Code

Arduino Program

```
// Project : Magic Chessboard
// Purpose : To retrieve the move made by human, send to rpi and move motor with
response
// Author  : Max Zischka
// Course  : ICS3U-E
// Date    : 04/12/24
// MCU     : 328 P
// Status  : Working
// Reference: https://www.youtube.com/watch?v=Dco6jo9xgAo&t=526s
// Notes   : The code works with hardware components but could be optimized if I try
a version two(castling and horse moves still need work along with move capturing)
uint8_t muxPins[] = { 3, 4, 5, 6 };             // multiplexer switch pins
uint8_t enablePins[] = { 7, 8, 9, 10 };         // multiplexer enable pins
uint8_t muxOutputPins[] = { A0, A1, A2, A3 };   // multiplexer sig pins
volatile bool triggered = false;                // boolean statement for attach
interrupt
uint32_t moveTime = 5000;                       // time for pins to scan
uint32_t previousStatesArray[4][16] = { 0 };    // array to track changes
uint32_t lastTimeChangeArray[4][16] = { 0 };    // array to track time changes
uint32_t startTime;                             // variable for millis
#define interruptPin 2
#define magnet A5
#define dirPin 12  // all A4988 pin definitions
#define stepPin 11
#define dirPin2 A4
#define stepPin2 13
char* chessArray[64] = {
  "a1", "a2", "a3", "a4", "a5", "a6", "a7", "a8",
  "b1", "b2", "b3", "b4", "b5", "b6", "b7", "b8",
  "c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8",
  "d1", "d2", "d3", "d4", "d5", "d6", "d7", "d8",
  "e1", "e2", "e3", "e4", "e5", "e6", "e7", "e8",
  "f1", "f2", "f3", "f4", "f5", "f6", "f7", "f8",
  "g1", "g2", "g3", "g4", "g5", "g6", "g7", "g8",
  "h1", "h2", "h3", "h4", "h5", "h6", "h7", "h8"
```

```
};  // chessboard array
void setup() {

  for (uint8_t i = 0; i < 4; i++) {  // setting up pins for multiplexer and motors
    pinMode(muxPins[i], OUTPUT);
    pinMode(enablePins[i], OUTPUT);
    digitalWrite(enablePins[i], HIGH);
  }
  pinMode(interruptPin,INPUT_PULLUP);                    // setting interrupt pin
  attachInterrupt(digitalPinToInterrupt(2), ISR_Chess, FALLING);// attach interrupt
  Serial.begin(9600);                          // initializing serial communication
}
void loop() {
  // handling triggered state
  if (triggered) {
    startTime = millis();  // recording start time
    uint32_t currentTime = millis();
    while (millis() - startTime < moveTime) {  // scanning for specified time
      for (uint8_t mux = 0; mux < 4; mux++) {
        digitalWrite(enablePins[mux], LOW);  // enabling single mux
        // scanning each channel of mux
        for (uint8_t channel = 0; channel < 16; channel++) {
          selectChannel(channel);                            // selecting channel
          uint32_t state = digitalRead(muxOutputPins[mux]);  // reading state

          // tracking changes
          if (state != previousStatesArray[mux][channel]) {
            lastTimeChangeArray[mux][channel] = currentTime;// setting specific time
            previousStatesArray[mux][channel] = state;        // updating array
          }
        }

        digitalWrite(enablePins[mux], HIGH);  // turning off mux
      }


      for (uint8_t mux = 0; mux < 4; mux++) {  // scanning changes within each mux
        for (uint8_t channel = 0; channel < 16; channel++) {
          for (uint8_t linkedChannel = 0; linkedChannel < 16; linkedChannel++) {
            if (linkedChannel != channel) {
              if (lastTimeChangeArray[mux][channel] != 0 &&
lastTimeChangeArray[mux][linkedChannel] != 0 &&
abs(lastTimeChangeArray[mux][channel] - lastTimeChangeArray[mux][linkedChannel]) <
moveTime) {

                if (previousStatesArray[mux][channel] <
previousStatesArray[mux][linkedChannel]) {  // checking for correct move and
printing
                  Serial.print(chessArray[mux * 16 + channel]);
                  Serial.println(chessArray[mux * 16 + linkedChannel]);
                } else {
                  Serial.print(chessArray[mux * 16 + linkedChannel]);
                  Serial.println(chessArray[mux * 16 + channel]);
                }
                lastTimeChangeArray[mux][channel] = 0;
                lastTimeChangeArray[mux][linkedChannel] = 0;
              }
            }
          }
        }

        for (uint8_t otherMux = 0; otherMux < 4; otherMux++) {  // scanning changes
          for (uint8_t channel = 0; channel < 16; channel++) {
```

```
            for (uint8_t otherChannel = 0; otherChannel < 16; otherChannel++) {
              if (otherMux != mux) {
                if (lastTimeChangeArray[mux][channel] != 0 &&
lastTimeChangeArray[otherMux][otherChannel] != 0 &&
abs(lastTimeChangeArray[mux][channel] - lastTimeChangeArray[otherMux][otherChannel])
< moveTime) {

                  if (previousStatesArray[mux][channel] <
previousStatesArray[otherMux][otherChannel]) {  // checking for correct move
                    Serial.print(chessArray[mux * 16 + channel]);
                    Serial.println(chessArray[otherMux * 16 + otherChannel]);
                  } else {
                    Serial.print(chessArray[otherMux * 16 + otherChannel]);
                    Serial.println(chessArray[mux * 16 + channel]);
                  }
                  lastTimeChangeArray[mux][channel] = 0;
                  lastTimeChangeArray[otherMux][channel] = 0;
                  lastTimeChangeArray[otherMux][otherChannel] = 0;
                }
              }
            }
          }
        }
      }
    }
    triggered = false;  // resetting triggered flag
  }
  if (Serial.available() > 0) {  // receiving move from RPi
    String move = Serial.readStringUntil('\n');
    uint8_t firstColumn = move[0] - 'a' + 1;  // breaking up the move
    uint8_t firstRow = move[1] - 48;
    uint8_t secondColumn = move[2] - 'a' + 1;
    uint8_t secondRow = move[3] - 48;

    if (firstColumn == secondColumn && firstRow != secondRow) {  // moving
vertically
      moveToStart(firstColumn, firstRow);               // setting start position
      digitalWrite(magnet, HIGH);                       // turning on magnet
      uint8_t amountSteps = firstRow - secondRow;
      if (amountSteps > 0)
        moveDown(abs(amountSteps));
      else
        moveUp(abs(amountSteps));
      digitalWrite(magnet, LOW);
      moveToEnd(secondColumn, secondRow);  // setting end position
    }
    if (firstColumn != secondColumn && firstRow == secondRow) {// moving horx
      uint8_t horzSteps = firstRow - secondRow;
      if (horzSteps > 0)
        moveRight(abs(horzSteps));
      else
        moveLeft(abs(horzSteps));
    }
    if (firstColumn != secondColumn && firstRow != secondRow) {// moving diag
      uint8_t horzDis = firstColumn - secondColumn;
      uint8_t vertDis = firstRow - secondRow;

      if (abs(horzDis) == abs(vertDis)) {  // diagonal move
        moveToStart(firstColumn, firstRow);
        digitalWrite(magnet, HIGH);
        if (horzDis > 0 && vertDis > 0)
          moveSW(abs(horzDis));
        if (horzDis > 0 && vertDis < 0)
```

```
          moveNW(abs(horzDis));
        if (horzDis < 0 && vertDis > 0)
          moveSE(abs(horzDis));
        if (horzDis < 0 && vertDis < 0)
          moveNE(abs(horzDis));
        digitalWrite(magnet, LOW);
        moveToEnd(secondColumn, secondRow);
      }
      if (abs(horzDis) != abs(vertDis) && move != "e8g8h8f8" && move != "e8c8a8d8")
{  // horse move
        moveToStart(firstColumn, firstRow);
        digitalWrite(magnet, HIGH);
        if (vertDis > 0)
          moveDown(abs(vertDis));
        if (vertDis > 0)
          moveUp(abs(vertDis));
        if (horzDis > 0)
          moveLeft(abs(horzDis));
        if (horzDis < 0)
          moveRight(abs(horzDis));
        digitalWrite(magnet, LOW);
        moveToEnd(secondColumn, secondRow);
      }
    }
  }
}
void ISR_Chess() {  // interrupt service routine for chess scan
  triggered = true;
  delay(100);  // debouncing delay
}
void selectChannel(int channel) {  // selecting channel for mux
  for (uint8_t x; x < 4; x++)
    digitalWrite(muxPins[x], bitRead(channel, 0));
}
void moveUp(uint8_t val) {  // function for moving up (both motors move in opposite
directions)
  for (uint8_t stepTimesN; stepTimesN < val; stepTimesN++) {
    digitalWrite(dirPin, HIGH);
    digitalWrite(dirPin2, LOW);
    for (uint8_t x; x++; x < 80) { //one board move
      digitalWrite(stepPin, HIGH);
      digitalWrite(stepPin2, HIGH);
      delayMicroseconds(5000);
      digitalWrite(stepPin, LOW);
      digitalWrite(stepPin2, LOW);
      delayMicroseconds(5000);
    }
    delay(500);
  }
}
void moveRight(uint8_t val) {  // function for moving right (both motors move in
same directions)
  digitalWrite(dirPin, LOW);
  digitalWrite(dirPin2, LOW);
  for (uint8_t stepTimesE; stepTimesE < val; stepTimesE++) {
    for (uint8_t x; x++; x < 80) { //one board move
      digitalWrite(stepPin, HIGH);
      digitalWrite(stepPin2, HIGH);
      delayMicroseconds(5000);
      digitalWrite(stepPin, LOW);
      digitalWrite(stepPin2, LOW);
      delayMicroseconds(5000);
    }
```

```
    delay(500);
  }
}

void moveLeft(uint8_t val) {  // function for moving left (both motors move in same
directions)
  for (uint8_t stepTimesW; stepTimesW < val; stepTimesW++) {
    digitalWrite(dirPin, HIGH);
    digitalWrite(dirPin2, HIGH);
    for (uint8_t x; x++; x < 80) { //one board move
      digitalWrite(stepPin, HIGH);
      digitalWrite(stepPin2, HIGH);
      delayMicroseconds(5000);
      digitalWrite(stepPin, LOW);
      digitalWrite(stepPin2, LOW);
      delayMicroseconds(5000);
    }
    delay(500);
  }
}
void moveDown(uint8_t val) {  // function for moving down (both motors move in
opposite directions)
  for (uint8_t stepTimesS; stepTimesS < val; stepTimesS++) {
    digitalWrite(dirPin, LOW);
    digitalWrite(dirPin2, HIGH);
    for (uint8_t x; x++; x < 80) {//one board move
      digitalWrite(stepPin, HIGH);
      digitalWrite(stepPin2, HIGH);
      delayMicroseconds(5000);
      digitalWrite(stepPin, LOW);
      digitalWrite(stepPin2, LOW);
      delayMicroseconds(5000);
    }
    delay(500);
  }
}
void moveNW(uint8_t val) {  // function for moving diagonally NW(only one motor
necessary)
  for (uint8_t stepTimesNW; stepTimesNW < val; stepTimesNW++) {
    digitalWrite(dirPin, HIGH);
    for (uint8_t x; x++; x < 80) {//one board move
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(5000);
      digitalWrite(stepPin, LOW);
      delayMicroseconds(5000);
    }
    delay(500);
  }
}
void moveNE(uint8_t val) {  // function for moving diagonally NE(only one motor
necessary)
  for (uint8_t stepTimesNE; stepTimesNE < val; stepTimesNE++) {
    digitalWrite(dirPin2, HIGH);
    for (uint8_t x; x++; x < 80) {//one board move
      digitalWrite(stepPin2, HIGH);
      delayMicroseconds(5000);
      digitalWrite(stepPin2, LOW);
      delayMicroseconds(5000);
    }
    delay(500);
  }
}
```

```
void moveSW(uint8_t val) {  // function for moving diagonally SW(only one motor
necessary)
  for (uint8_t stepTimesSW; stepTimesSW < val; stepTimesSW++) {
    digitalWrite(dirPin2, LOW);
    for (uint8_t x; x++; x < 80) {//one board move
      digitalWrite(stepPin2, HIGH);
      delayMicroseconds(5000);
      digitalWrite(stepPin2, LOW);
      delayMicroseconds(5000);
    }
    delay(500);
  }
}
void moveSE(uint8_t val) {  // function for moving diagonally SE(only one motor
necessary)
  for (uint8_t stepTimesSE; stepTimesSE < val; stepTimesSE++) {
    digitalWrite(dirPin, LOW);
    for (uint8_t x; x++; x < 80) { //one board move
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(5000);
      digitalWrite(stepPin, LOW);
      delayMicroseconds(5000);
    }
    delay(500);
  }
}
void moveToStart(uint8_t firstColumn, uint8_t firstRow) {  // setting start position
  uint8_t horzStartingPos = 8;
  uint8_t vertStartingPos = 1;
  uint8_t startDishorz = horzStartingPos - firstColumn;
  uint8_t startverthorz = horzStartingPos - firstRow;
  moveLeft(startDishorz);
  moveUp(startverthorz);
}
void moveToEnd(uint8_t secondColumn, uint8_t secondRow) {  // setting end position
  uint8_t horzEndingPos = 8;
  uint8_t vertEndingPos = 1;
  uint8_t endDishorz = horzEndingPos - secondColumn;
  uint8_t endverthorz = vertEndingPos - secondRow;
  moveRight(endDishorz);
  moveDown(endverthorz);
}
```

Pi Program

```
// Project : Magic Chessboard
// Purpose : To retrieve the move made by human, send to rpi and move motor with
response
// Author  : Max Zischka
// Course  : ICS3U-E
// Date    : 04/12/24
// MCU     : 328 P
// Status  : Working
// Reference: https://www.instructables.com/Raspberry-Pi-Arduino-Serial-
Communication/
// Notes   : The code works with hardware components but could be optimized if I try
a version two(castling and horse moves still need work along with move capturing)
import serial
import chess.engine

ser = serial.Serial('/dev/ttyUSB1', 9600)#matching baud rate
engine = chess.engine.SimpleEngine.popen_uci("/usr/games/stockfish")
last_line = ""  # basic storage varible
line_number = 0  # basic storage varible
```

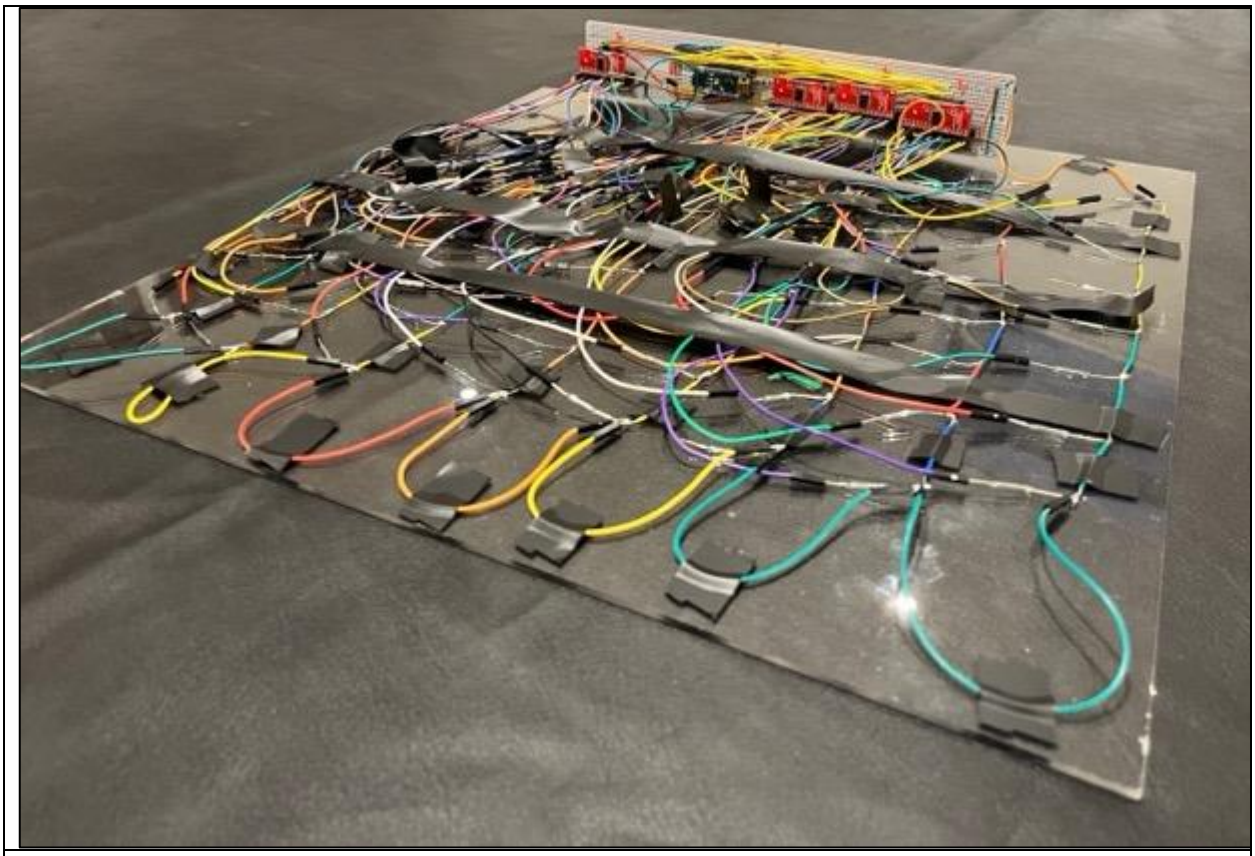
```
while True:
    if ser.in_waiting > 0:
        line = ser.readline().decode().strip()
        line_number += 1  # keep track of line #
        if line_number % 2 != 0:  # check if the line number is odd(white move)
            if line != last_line:  # check for change
                last_line = line  # Update the last line
                print("Received:", line)
                board = chess.Board()
                try:
                    board.push_san(line)  # Attempt to push the received move
                    result = engine.play(board, chess.engine.Limit(time=0.1))
                    best_move = result.move
                    print("Best move for black:", best_move)
                    ser.write(str(best_move).encode() + b'\n')
                except ValueError:
                    print("Invalid move:", line)
```

Media

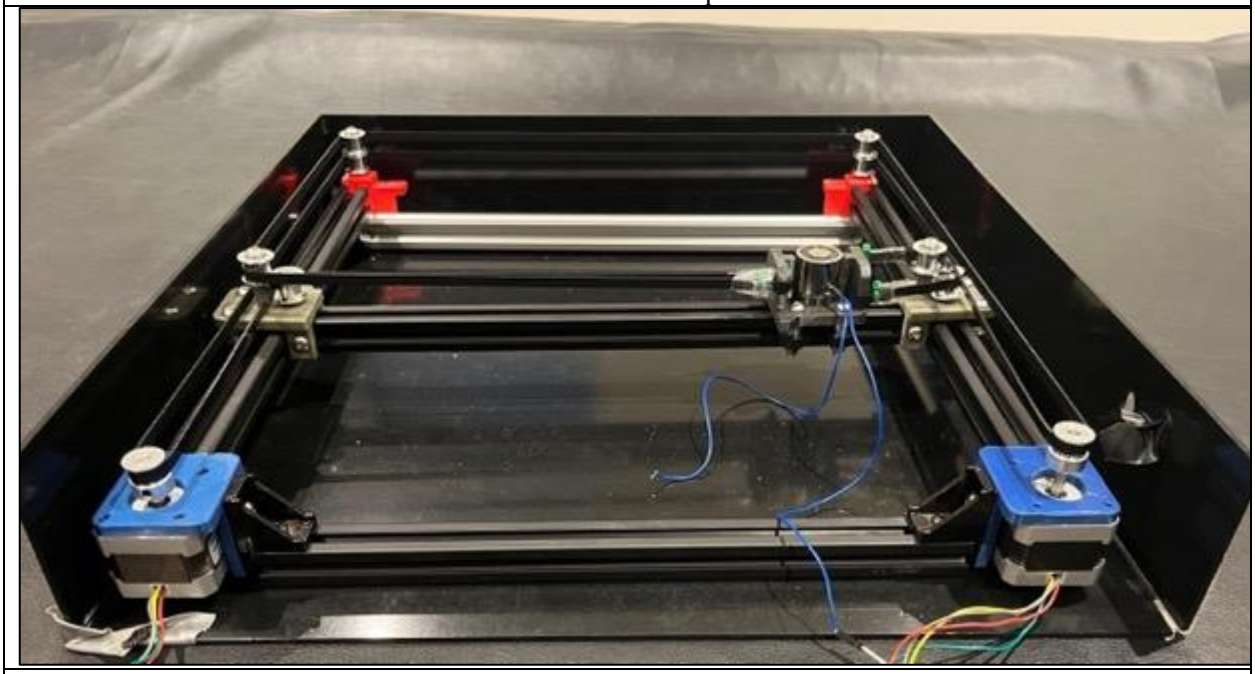
Project Video: https://youtu.be/Vd35VLUMOtQ



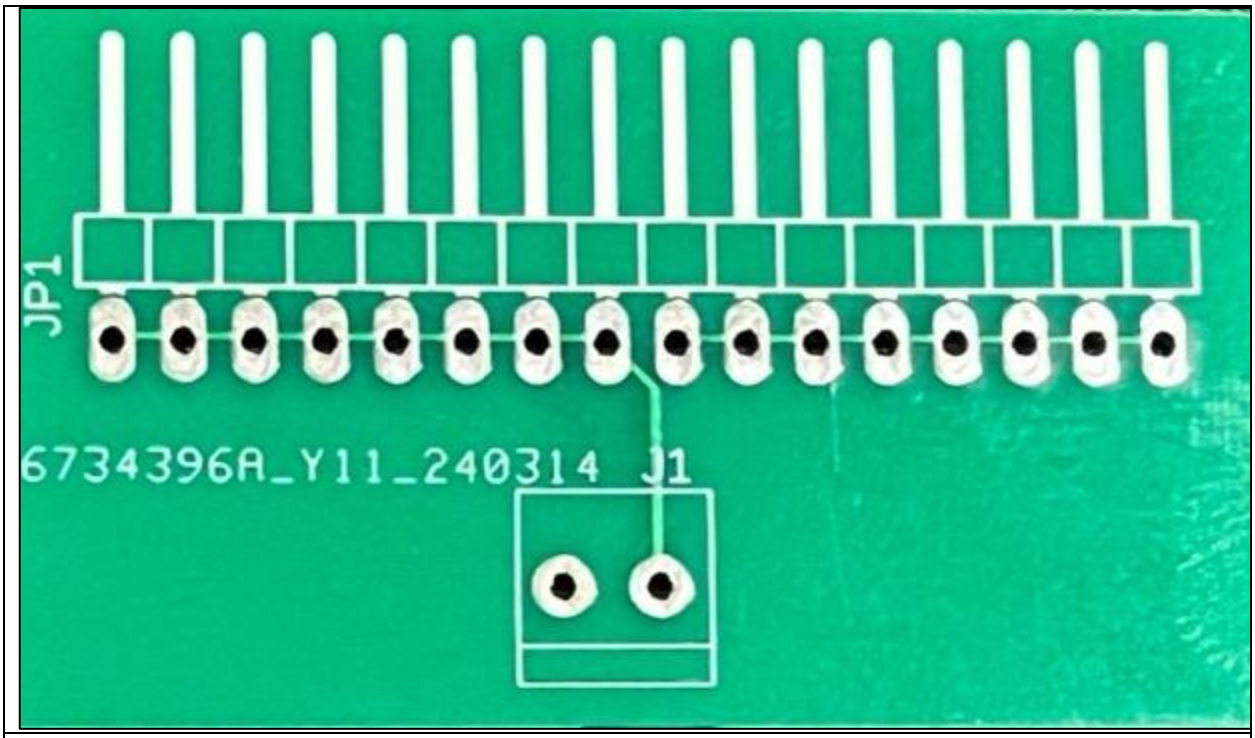
The Magic Chessboard



View of the top



View of inside



PCB Photo(Other tracing on back)

Reflection

Wow, to think that this ISP is over is one of the craziest most bittersweet feelings I have ever had. For such a hard project to be done is a great feeling but it is also sad because this is probably the project, I have worked on the most in my entire life clocking in about 250 hours I would say. The project is an experience I will always remember from staying up late to finding different approaches to problems having hit a dead end and turning around. I really did love the work that went into a project of this difficulty and I do not think this will be the last time I put work into this idea. When thinking about how ambitious it was, I will definitely keep this in mind in the future. I did love the project but the work was A LOT and, in the future, I want to keep all the things I have learned in mind. This project has taught me so much about being an engineer and is how I understand why ISPs are a gift to the ACES.

This ISP definitely made me think that I was a passionate engineer. Two years ago, before the ACES program looking at my brother and father, I told myself I did not want to become an engineer but after this program and all it has taught me over the year and half with the ISP being a big part of that learning experience, I realized that why being an engineer is such an enriching experience. Staying up late solving problems, hearing about the other projects going on. This medium ISP was such a fun experience and I am so grateful I got to have it.

ICS4U-E

Project 3.1 CharlieClock

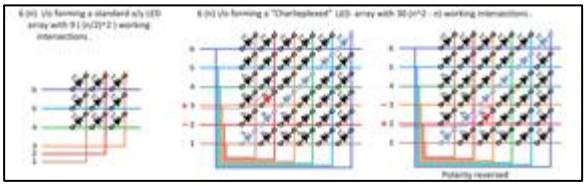
Purpose

The purpose of the CharlieClock is to display the time in hours, minutes and seconds through 132 LEDs controlled by 12 microcontroller pins. Additionally, components such as the rotary encoder, ATtiny85, and 7-Segment display are included to be able alter the time displayed and display the date and month.

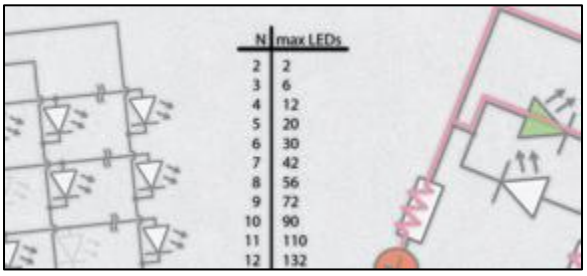
Reference

Project Description: http://darcy.rsgc.on.ca/ACES/TEI4M/Tasks.html#CharlieClock
Charlieplexing Explanation: https://en.wikipedia.org/wiki/Charlieplexing
Permutations: https://www.calculatorsoup.com/calculators/discretemathematics/permutations.php
Reflow Oven Demo: https://whizoo.com/pages/buildguide#Step-1
STL Viewer: https://www.viewstl.com/

Theory

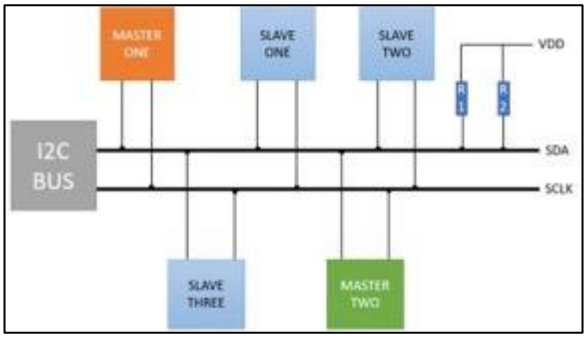


The method by which the CharlieClock displays the time relies on a technique known as CharliePlexing. This approach allows control over a larger number of LEDs using far fewer control pins, as long as impedance can be added or removed from the circuit as needed. The key to CharliePlexing is the way the components are arranged so that when impedance is removed from two specific pins—one serving as the cathode and the other as the anode—only a single LED light up. This selective control is what makes it possible to manage many LEDs with minimal pins. In the schematic shown to the right, the concept is demonstrated with 20 LEDs being controlled by just five high-impedance pins, showcasing the efficiency of CharliePlexing in reducing the number of necessary connections.



N	max LEDs
2	2
3	6
4	12
5	20
6	30
7	42
8	56
9	72
10	90
11	110
12	132

The relationship between the number of LEDs and the pins required for a microcontroller can be effectively illustrated using the equation $L(P) = P(P - 1)$, which defines how the number of LEDs varies as a function of time. For instance, if you have 5 microcontroller pins, substituting this value into the equation gives $L(5) = 5(4)$. This indicates that with just 5 microcontroller pins, you can control a total of 20 individual LEDs. The quadratic nature of this equation signifies that the number of LEDs that can be controlled increases exponentially as more pins are added, which is by the equation $L(P) = P^2 - P$. This expanded form highlights the exponential growth characteristic of the equation, and the accompanying sketch serves to visually demonstrate this exponential relationship, illustrating how the number of controlled LEDs rapidly escalates with an increase in microcontroller pins.



The last theoretical idea brought up is inter-integrated communication. I2C communication happens between two buses that are pulled high and communicate through a clock line sending the start and stop signal with the data lines sending the 8-bit instructions. I2C is a type of communication used to two devices in this project which it can easily communicate between because of the slave addresses making sure every communication goes to the correct component.

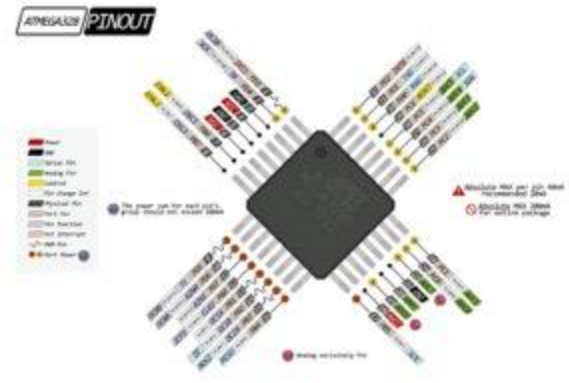
Procedure

The procedure for creating the CharlieClock begins with stating the goal behind the project. The goal being to control 132 LEDs with 12 control pins while being able to display the time. The exact approach we took was to use an ATmega328P as a controller. An ATtiny85 used as a slave device for more control pins to display the day and month through the components of a seven-segment-display and two shift register to output the correct values. The DS1307 is a real-time clock that can hold the real time through the component of a 32.768kHz crystal that scales down the time to one sec. Lastly there is the addition of the rotary encoder is included for the functions of adjusting the time. This is done with the rotary encoders push ability and the scrolling ability.

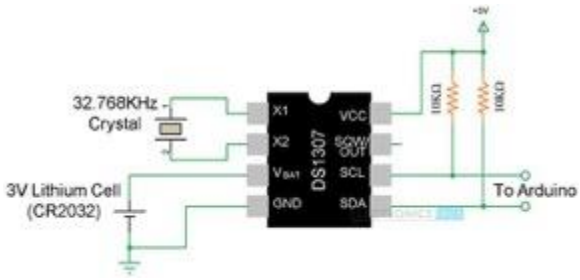
Parts Table	
Quantity	Description
1	ATmega328p
1	ATtiny85
1	7-Segment Display
1	CharlieClock PCB
1	DS1307
1	32.768kHz Crystal
1	S0024 Rotary Encoder
2	595 Shift Register
8	1206 10k Ω Resistor
2	0.1µ F



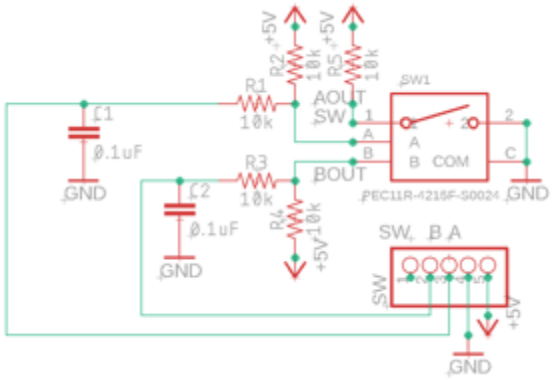
The first major component of this project is the CharlieClock PCB. This is a 132 LED configuration set to 12 header pins that connect to a microcontroller. Using the idea learned in the theory if P(12) is inputted the output would be 12(11) which equals 132. These LEDs are set up so that if only one pin is set to be an anode and one to cathode that only LED will turn on. The next problem that arises in the process of trying to get this PCB to tell the time is navigating which of the 132 LEDs is which. This can be done when analyzing the schematic provided to the right. This schematic shows the wiring and helps to navigate which pins emit which sections of LEDs. As the schematic shows there are three sections for the three values of time, hours, minutes and seconds. Color coded LEDs are used to show which value represent which with hours being blue, minutes green and seconds red as displayed through the LED diffuser.



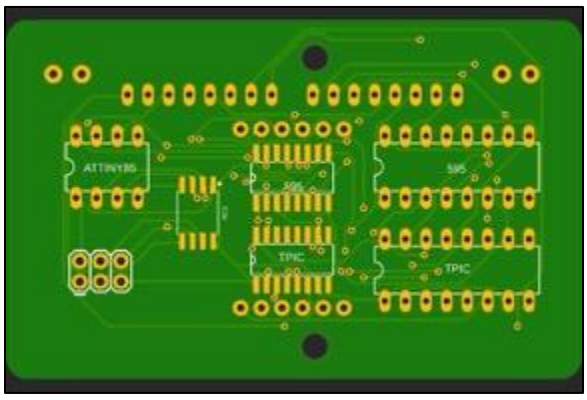
The Atmega328p finds itself at the center of this project as the microcontroller. The tasks under this microcontroller include controller the 12 pins that charlieplex the 132 LEDs along with configuring the additional pieces; the rotary encoder which moves through the Atmega328p to alter the time variables and to set and read the DS1307 to have the real-time and then every time power is applied again it will pull this time value again. What makes the Atmega328p a perfect component for this project is the fact that the microcontroller pins have two important pin modes: input and output. When a pin is on output it can act as a cathode sending a low signal and act as an anode sending a high signal. When the pins are set to input the pins is only reading and not taking in any voltage our current. The high, low and impeading pins are all essential in charlieplexing 12 pins making the 328p the perfect component for this task.



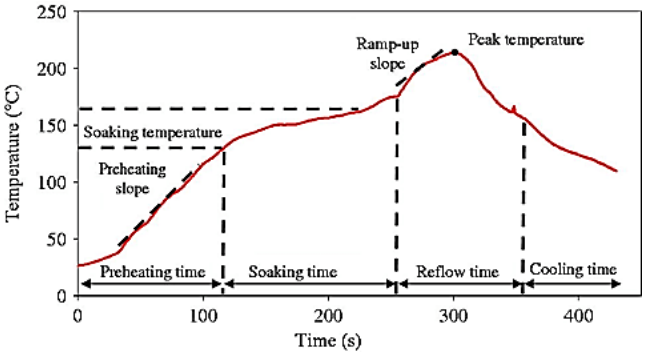
The next integrated circuit used in this version of the CharlieClock is the DS1307 real-time clock. This component connected to 32.768kHz crystal has the function of keeping the real time before storing the value in the IC which will be pulled through I2C communication. The first part to understand about the DS1307 is how it is able to keep track of the real-time. This comes from 32.768kHz crystal sitting with the chip that does binary division of the crystal oscillating 32,768 times a second to only count the value one when the oscillations are divided by 2^{15} this division actually happens in 3 steps from 32.768 to 16.384 to 8.192 before the 1Hz signal is used. This data that is being kept while powered and while powered off due to the 3.3V battery retainer kept with the chip. It is then able to extracted by calling the I2C device and requesting the correct bits from the chip as seen to be done in the code.



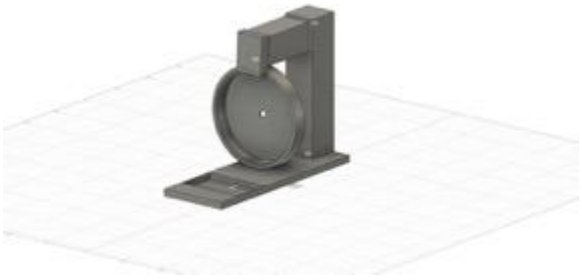
The next important component is the piece for telling the time on the CharlieClock, the rotary encoder. The rotary encoder has the ability to send a square wave pulse through 1 pin, and have alternating binary outputs that are of different orientation based on the way you rotate it. What this allows is for this single component to be able to change the variable time values and switch between which time value the user wants to alter. This pulls the load of several buttons and potentiometers all in one component. The code below displays exactly how the Atmega was able read counter-clockwise to clockwise and a button press. One example from the code is the `attachInterrupt(digitalPinToInterrupt(SW), ISR_SWPRESSED, RISING);` this shows the normal operation being interrupted to alter a variable that decides which time variable can be change by the rotary encoder. The schematic shows the S0024 encoder with the passive debouncing components used.



The last component is the ATtiny85 PCB used to display the month and date from the DS1307. This process is done using a PCB that my partner, Rohan Jamal, created for his Medium ISP project last year. The configuration involves the Attiny85 acting as slave device for the ATmega as the master in I2C communication. This also allows for more microcontroller pins that can interface with two HC595 shift registers. Shift register function off three pins controlled by microcontroller which are all controlled by the Attiny85. It will continuously output the variable unless which can be changed via I2C by the Atmega. These values are shifted to a common cathode 7-segment display that has been a part of several projects before. The display is showing the month and date values. The image on the right displays the PCB.



When all components are put together, they are put onto a PCB made by JLCPCB and designed is EASYEDA. The parts were all configured to be above the sizing 1206 to make the next step of this process easier. For cost efficiency the approach taken was to use a PCB stencil instead of JLCPCBs option of PCBA. This meant to surface mount products were assembled by my partner and I using a reflow oven. The concept behind the reflow oven is that all the parts are put into place with the correct amount of solder past in the correct placements this is then all put into the reflow oven. In the oven the temperatures are gradually increasing before reaching the reflow stage for a moment and then cooling. This allows for an even airflow and no issues around burning components. The outputs received from the CharlieClock PCB were very successful even with the Atmega328p with 48 extremely close pads as displayed in media.



From a design aspect the components have the functions of holding a 7-segment display, CharlieClock PCB and supplement board to the PCB. In the end it is broken up into 4 pieces that are friction fit together to create the structure shown to the right. Additional pieces include an LED diffuser and a rotary encoder piece. The LED diffuser is segmented into 133 parts for the 132-time values and the large circle in the center. The process of making involves individually shelling the bodies into the segments before finally at a text along an additional circle arc. The additional design featured are displayed in the media section below.

Code

Atmega328p

```
// PROJECT    : CharlieClock
// AUTHOR     : R. Jamal and M. Zischka
// PURPOSE    : To display time on CharlieClock
// COURSE     : ICS4U-E
// DATE       : 04 10 2024
// MCU        : ATMEGA328P
// STATUS     : Working
// NOTES      : Written in 1.8.19
#include  <Wire.h>
#define P1 17
#define P2 16
#define P3 15
#define P4 14
#define P5 13
#define P6 12
#define P7 11
#define P8 8
#define P9 7
#define P10 6
#define P11 5
#define P12 4
#define DS1307RTCADDRESS  0x68      //RTC I2C Address
#define DISPLAYADDRESS    0x30      //Display I2C Address
#define SW 2                        //Rotary encoder button pin
#define PINA 3                      //Rotary encoder pin 'A'
```

```
#define PINB 10                       //Rotary encoder pin 'B'
uint8_t pins[] = {P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12};
uint8_t numPins = sizeof(pins);       //Number of charlieplexed pins
uint8_t preA, date, mon;               //Last state of RE 'A' pin
uint8_t count = 0;                    //Which attribute encoder affects
volatile uint8_t div2 = 0;            //Variable to divide RTC PCINT by 2
volatile uint8_t pin_A, pin_B;        //Current states of pins A and B
volatile bool changed = false;        //Boolean value for rotary encoder changes
volatile int16_t time[] = {0, 0, 0};//Array to store the current time
int64_t lastTrans = -10000;
struct LED {                          //Struct to hold LED pin maps
  uint16_t cathode;                   //Cathode
  uint8_t anode;                      //Anode
};
LED hours[] = {                       //Hours pin map
  {P1, P9}, {P1, P10}, {P1, P11}, {P1, P12}, {P11, P12}, {P1, P5}, {P1, P4},
  {P1, P3}, {P1, P2}, {P1, P6}, {P1, P7}, {P1, P8}
};
LED minutes[] = {                     //Minutes pin map
  {P2, P8}, {P2, P9}, {P2, P10}, {P2, P11}, {P2, P12},
  {P12, P1}, {P12, P2}, {P12, P3}, {P12, P4}, {P12, P5},
  {P12, P6}, {P12, P7}, {P12, P8}, {P12, P9}, {P12, P10}, {P12, P11},
  {P11, P10}, {P11, P9}, {P11, P8}, {P11, P7}, {P11, P6}, {P11, P5},
  {P11, P4}, {P11, P3}, {P11, P2}, {P11, P1}, {P10, P12}, {P10, P11},
  {P10, P9}, {P10, P8}, {P10, P7}, {P10, P6}, {P10, P5},
  {P10, P4}, {P10, P3}, {P10, P2}, {P10, P1}, {P4, P7}, {P4, P6}, {P4, P5},
  {P4, P3}, {P4, P2}, {P4, P1}, {P3, P1}, {P3, P2}, {P3, P4}, {P3, P5},
  {P3, P6}, {P3, P7}, {P3, P8}, {P3, P9}, {P3, P10}, {P3, P11}, {P3, P12},
  {P2, P1}, {P2, P3}, {P2, P4}, {P2, P5}, {P2, P6}, {P2, P7}
};
LED seconds[] = {                     //Seconds pin map
  {P5, P1}, {P5, P2}, {P5, P3}, {P5, P4}, {P5, P6}, {P5, P7}, {P5, P8},
  {P5, P9}, {P5, P10}, {P5, P11}, {P5, P12}, {P4, P8}, {P4, P9}, {P4, P10},
  {P4, P11}, {P4, P12}, {P9, P12}, {P9, P11}, {P9, P10}, {P9, P8}, {P9, P7},
  {P9, P6}, {P9, P5}, {P9, P4}, {P9, P3}, {P9, P2}, {P9, P1}, {P8, P12},
  {P8, P11}, {P8, P10}, {P8, P9}, {P8, P7}, {P8, P6}, {P8, P5}, {P8, P4},
  {P8, P3}, {P8, P2}, {P8, P1}, {P7, P12}, {P7, P11}, {P7, P10}, {P7, P9},
  {P7, P8}, {P7, P6}, {P7, P5}, {P7, P4}, {P7, P3}, {P7, P2}, {P7, P1}, {P6, P1},
  {P6, P2}, {P6, P3}, {P6, P4}, {P6, P5}, {P6, P7}, {P6, P8}, {P6, P9},
  {P6, P10}, {P6, P11}, {P6, P12}
};
uint8_t numHours = sizeof(hours) / sizeof(LED);     //Total number of hours LEDs
uint8_t numMins = sizeof(minutes) / sizeof(LED);    //Total number of minutes LEDs
uint8_t numSecs = sizeof(seconds) / sizeof(LED);    //Total number of seconds LEDs

void setup() {
  attachInterrupt(digitalPinToInterrupt(SW), ISR_SWPRESSED, RISING);  //Encoder INT
  cli();                                    //Clear interrupts/noInterrupts();
  PCICR |= (1 << PCIE0);                    //Enable PCINT for pins D8-D13
  PCMSK0 |= (1 << PCINT1);                  //Mask PCINT for pin 9
  if (count != 0)
    PCMSK1 &= ~(1 << PCINT1);
  sei();                                    //Set interrupts/Interrupts();
  Wire.begin();                             //Initialize I2C
  Wire.beginTransmission(DS1307RTCADDRESS);  //Start a transmission to RTC
  Wire.write(0);                            //Start reading from Register 0
  Wire.endTransmission();                   //End I2C transmission
  Wire.requestFrom(DS1307RTCADDRESS, 6);     //Specify how much data required
  while (!Wire.available());                //Wait data to enter read buffer
  time[0] = BCD2DEC(Wire.read());           //Store first byte to sec buffer
  time[1] = BCD2DEC(Wire.read());           //Store second byte to min buffer
  time[2] = BCD2DEC(Wire.read());           //Store third byte to hours buffer
  date = Wire.read();
```

```
  date = BCD2DEC(Wire.read());
  mon = BCD2DEC(Wire.read());
}

void loop() {
  if ((millis() - lastTrans) > 9999) {
    Wire.beginTransmission(DISPLAYADDRESS);
    Wire.write(mon);
    Wire.write(date);
    Wire.write(3);
    Wire.endTransmission();
  }
  if (count)                                          //When count != 0 (set mode)
    attachInterrupt(digitalPinToInterrupt(PINA), ISR_ROTARYCHANGE, CHANGE); //Attach
  else detachInterrupt(digitalPinToInterrupt(PINA));  //If count = 0 detatch
  if (changed) {                                      //When change flag has tripped
    changed = false;                                  //Reset change flag
    if (pin_A != preA) {                              //Determine direction
      preA = pin_A;                                   //Store current state of pin_A
      if (pin_A == pin_B) time[count - 1]--;          //Decrement time (direction A)
      else time[count - 1]++;                         //Increment time (direction B)
    }
  }
  if (time[0] == 60) {  //Seconds wraparound (positive)
    time[1]++;
    time[0] = 0;
  }
  if (time[0] < 0) {    //Seconds wraparound (negative)
    time[1]--;
    time[0] = 59;
  }
  if (time[1] == 60) {  //Minutes wraparound (positive)
    time[2]++;
    time[1] = 0;
  }
  if (time[1] < 0) {    //Minutes wraparound (negative)
    time[2]--;
    time[1] = 59;
  }

  for (uint8_t pin = 0; pin < numPins; pin++)       //Set all pins to high-impedance
    pinMode(pins[pin], INPUT);                      //Set all pins to high-impedance
  pinMode(seconds[time[0]].anode, OUTPUT);          //Set seconds anode to output
  pinMode(seconds[time[0]].cathode, OUTPUT);        //Set seconds cathode to output
  digitalWrite(seconds[time[0]].anode, HIGH);       //Set seconds anode high
  digitalWrite(seconds[time[0]].cathode, LOW);      //Set seconds cathode low

  for (uint8_t pin = 0; pin < numPins; pin++)       //Set all pins to high-impedance
    pinMode(pins[pin], INPUT);                      //Set all pins to high-impedance
  pinMode(minutes[time[1]].anode, OUTPUT);          //Set minutes anode to output
  pinMode(minutes[time[1]].cathode, OUTPUT);        //Set minutes cathode to output
  digitalWrite(minutes[time[1]].anode, HIGH);       //Set minutes anode high
  digitalWrite(minutes[time[1]].cathode, LOW);      //Set minutes cathode low

  for (uint8_t pin = 0; pin < numPins; pin++)       //Set all pins to high-impedance
    pinMode(pins[pin], INPUT);                      //Set all pins to high-impedance
  pinMode(hours[time[2] % 12].anode, OUTPUT);       //Set hours anode to output
  pinMode(hours[time[2] % 12].cathode, OUTPUT);     //Set hours cathode to output
  digitalWrite(hours[time[2] % 12].anode, HIGH);    //Set hours anode high
  digitalWrite(hours[time[2] % 12].cathode, LOW);   //Set hours cathode low
}
ISR(PCINT0_vect) {                                  //PCINT attached to RTC SQW pin
  div2++;                                           //Divide frequency to 1 Hz
```

```
  if (div2 % 2) time[0]++;                          //Increment seconds on falling edge
}
void ISR_SWPRESSED() {                              //ISR for when button is pressed
  count++;                                          //Increment count
  if (count == 4) count = 0;                        //Count wraparound
}
void ISR_ROTARYCHANGE() {                           //Rotary encoder ISR
  changed = true;                                   //Activate 'changed' flag
  pin_A = digitalRead(PINA);                        //Store state of pin 'A'
  pin_B = digitalRead(PINB);                        //Store state of pin 'B'
}
uint8_t BCD2DEC(uint8_t value) {                    //Function to convert BCD to decimal
  return (value >> 4) * 10  + (value & 0x0F);
}
```

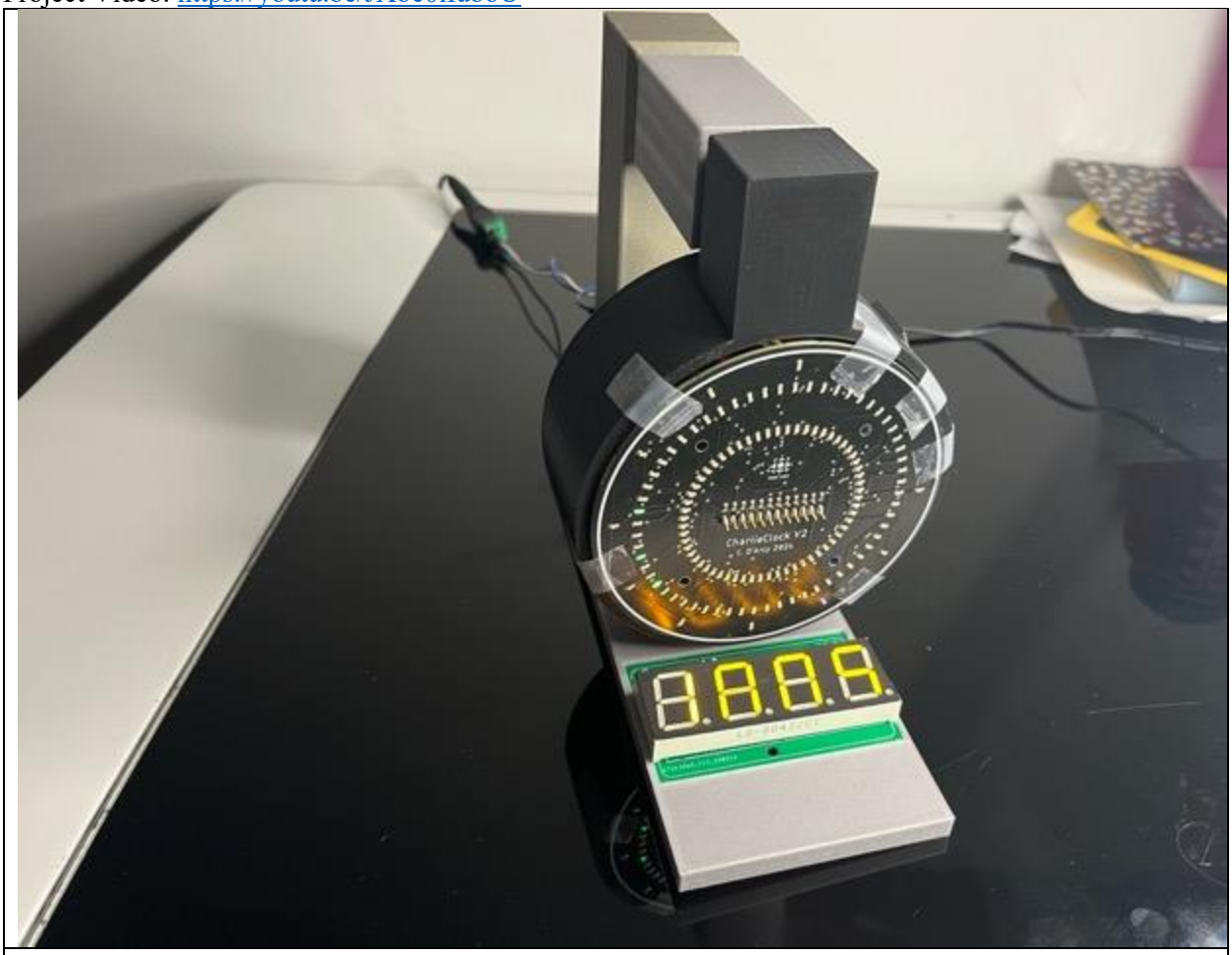
Attiny85

```
// PROJECT    : Display device slave (I2C)
// AUTHOR     : R. Jamal
// PURPOSE    : To display value received over I2C on display
// COURSE     : ICS4U-E
// DATE       : 04 10 2024
// MCU        : TINY85
// STATUS     : Working
#include <TinyWireS.h>                  //I2C slave library
#include <TinyWireM.h>                  //I2C library for compatibility
#define clockPin 4                      //Clock pin shift register
#define latchPin 3                      //Latch pin shift register
#define dataPin 1                       //Data pin shift register
#define ADDRESS 0x30                    //I2C address
Uint8_t rec1, rec2;                     //Receieved data buffer
uint8_t dec = 0;                        //Decimal data buffer
bool de;                                //Decimal enabled bool
uint8_t numbers[] = {                   //Start segment map
  B11111100,                            //0 (a,b,c,d,e,f)
  B01100000,                            //1 (b,c)
  B11011010,                            //2 (a,b,d,e,g)
  B11110010,                            //3 (a,b,c,d,g)
  B01100110,                            //4 (b,c,f,g)
  B10110110,                            //5 (a,c,d,f,g)
  B10111110,                            //6 (a,c,d,e,f,g)
  B11100000,                            //7 (a,b,c)
  B11111110,                            //8 (a,b,c,d,e,f,g)
  B11110110,                            //9 (a,b,c,d,f,g)
};                                      //End segment map
uint8_t buffer[] = {4, 3, 2, 1};        //Intermediate data holder
void setup() {                          //Setup function, runs once
  TinyWireS.begin(ADDRESS);             //Define I2C address
  pinMode(dataPin, OUTPUT);             //Set dataPin as output (DDR)
  pinMode(latchPin, OUTPUT);            //Set latchPin as output (DDR)
  pinMode(clockPin, OUTPUT);            //Set clockPin as output (DDR)
}
void loop() {                                   //Loop function, runs continuously
  writeBuffer();                                //Store received number to buffer
  TinyWireS_stop_check();                       //Detect stop condition
  for (uint8_t dig = 4; dig < 8; dig++) {       //Light digits, set decimal
    if ((dig - dec) == 3) de = 1;               //Decide when to enable decimal
    else de = 0;                                //Logic for decimal point
    lightDisp(1 << dig, numbers[buffer[dig - 4]] | de); //Shiftout segments
  }
}
void lightDisp(uint8_t anode, uint8_t cathode) {  //Function to shiftout segments
  PORTB &= ~(1 << latchPin);                      //Pull latch low
  shiftOut(dataPin, clockPin, LSBFIRST, anode);   //Shift out anodes
```

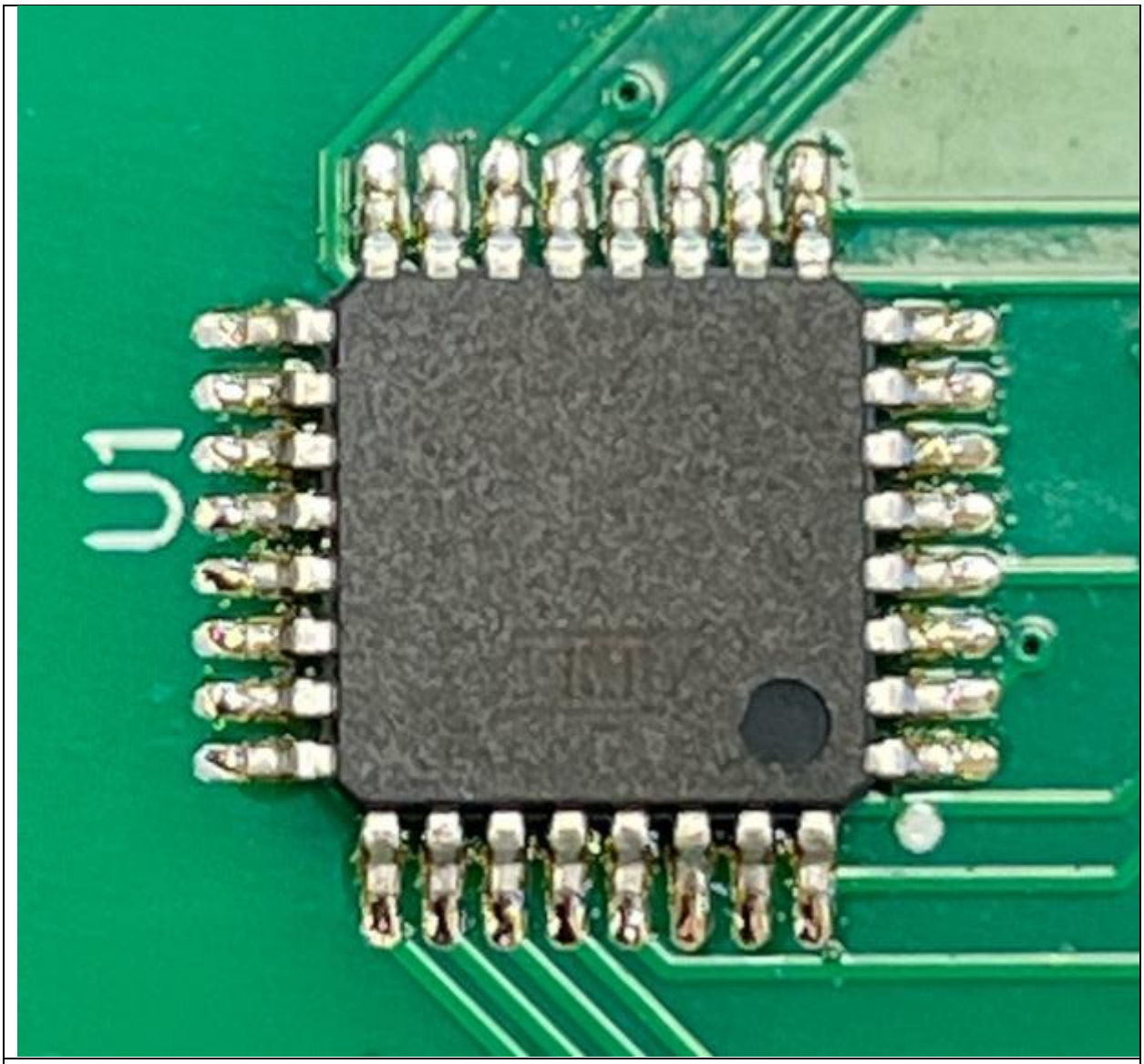
```
  shiftOut(dataPin, clockPin, LSBFIRST, cathode); //Shift out cathodes
  PORTB |= 1 << latchPin;                         //Set latch high
}
void writeBuffer() {                    //Function, stores received value to buffer
  if (TinyWireS.available()) {          //Waits until there is data being sent
    rec1 = TinyWireS.receive();         //Receives first byte
    rec2 = TinyWireS.receive();         //Receives second byte
    dec = TinyWireS.receive();          //Receives decimal byte
  }
  buffer[3] = rec1 / 10;                //Gets thousands digit
  buffer[2] = rec1 % 10;                //Gets hundreds digit
  buffer[1] = rec2 / 10;                //Gets tens digit
  buffer[0] = rec2 % 10;                //Gets ones digit
}
```

Media

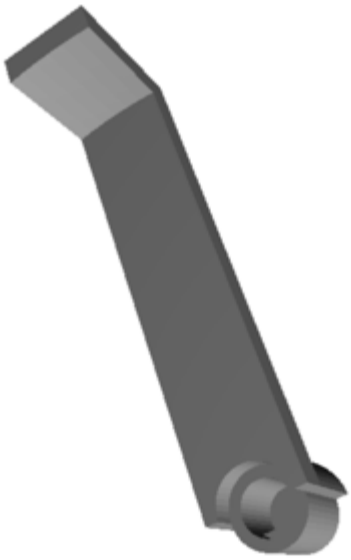
Project Video: https://youtu.be/JXbe0iIuboU



Final Design



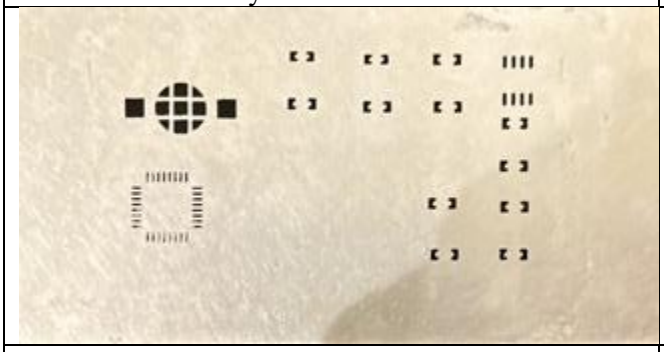
Reflow Oven Solder Job on Atmega



Rotary Encoder Addition



LED Diffuser



PCB Stencil



PCB Version 2

Reflection

This project was definitely a fun way to get the grade 12 ACES program running. Having a project that started in May already set the tone for how big this project was going to be. Rohan and I during the summer decided to try this summer work one night which turned into working on the entire project together. The group project saw the two of us ambitiously plan how we wanted to succeed at this project. The first flaws came when PCB prototype version failed due to design flaws around some components. Then deciding to go around for version two in the span of 12 hours to beat the holidays in China. This quickly saw more problems when the PCB didn't arrive until 2 days before the due date. Then in the span of 36 hours after trying to figure out the flaw in our PCB and nothing coming up, we made quick turn around and spent the next 15 hours hard at work. In the span of a day, we tried to solder a whole new charlieplex supplement board which also meant a new print job saw a whole new mountain of problems. Even in the end when the we came up short to our extremely ambitious goals, I am happy about the learning experiences I had while working on the CharlieClock. One of the most important things I learned during this project was that the amount of time you put in always benefits the outcome because even when you hit slumps like failed PCBs you are able to learning things like how to solder using a reflow oven, remembering to wire a crystal to an Arduino and so many more small learning experiences that I hope will make me a better engineer for the next project.

Project 3.2 CHUMP

Project 3.2.1 CHUMP: Clock and Counter

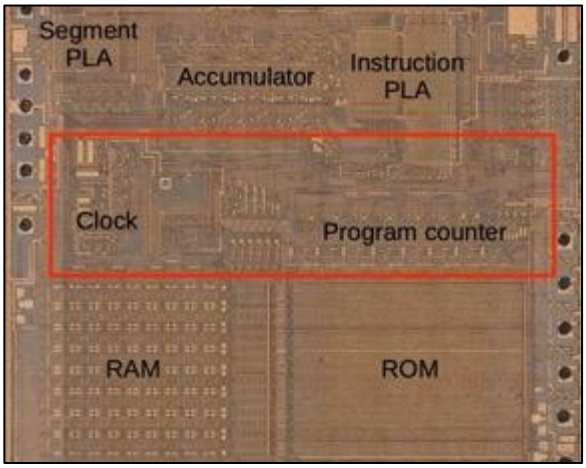
Purpose

The purpose of CHUMP: Clock and Counter is to create the first piece of the homemade 4-bit computer known as CHUMP. The piece has the purpose of counting the program instructions giving to the computer with the clock pulses of a astable and monostable 555 timer.

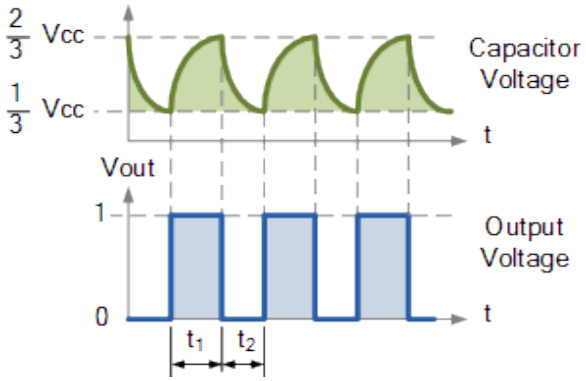
Reference

Project Description
Astable 555 Description: https://www.youtube.com/watch?v=kRlSFm519Bo&t=440s
Monostable 555 Description: https://www.youtube.com/watch?v=81BgFhm2vz8&t=239s
Bistable 555 Description: https://www.youtube.com/watch?v=WCwJNnx36Rk&t=524s

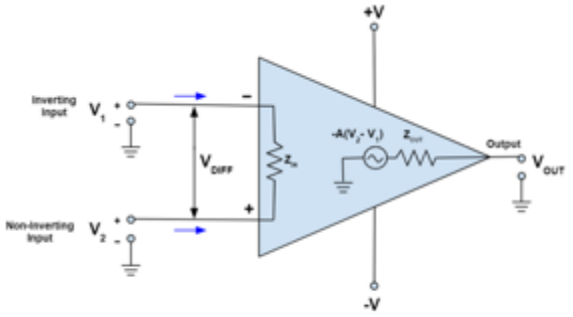
Theory



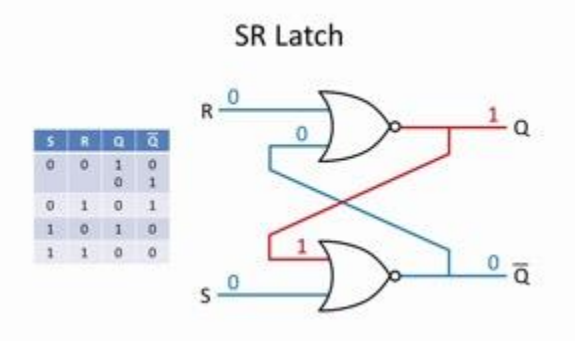
As the photo of the TMS1000, a 4-bit computer IC, on the right displays the two pieces discussed in this DER are critically important to several computing devices including the 4-bit computer known as CHUMP. These two pieces being the program counter and the clock. Theoretical factors that are important to the construction of the counter include capacitance and resistance's relationship and pulses. The main component which will be spoken about in the procedure is the 555 timer IC. The IC on the inside is built of an Op-Amp, transistor and resistor configuration that allows for the ability to use simple passive external components to adjust the frequency and pulses outputted from in integrated chip. A swell the theory behind SR Latch's and Op-Amps will be discussed.



The first theoretical idea resistors and capacitors effect on square waves. One way that a square wave can be manipulated is by manipulating the duty cycle, this is known is the ratio between time on and time off or mathematically expressed as $\frac{Time\ On}{Time\ On+Time\ Off} \times 100$. In the theoretical situation the duty cycle lies at 50% with rising pulse or falling pulse happening every half period. When working with the 555 timer the charge and discharge out of the capacitors will always have to stay between 1/3 and 2/3 of the VCC because of the arrangement of the three voltage dividers inside the 555 timer. The equation for charge time can be represented by the equation $t_{charge} = .693(R_1 + R_2)C$. This equation can be found when the exponential charge of the capacitor charging to two thirds of the voltage supply can be writing as $V_{CC}(1 - e^{-\frac{t_{charge}}{(R_1+R_2)C}})$ which when broken down equals $\ln(\frac{1}{3})$ when the supply is two thirds of the voltage which equals the constant shown above. This shows the relationship between the time for the capacitor to charge, altering the 0 signal to 1 signal, and the resistor and capacitor values being able to alter frequency and square wave outputs.



Inside the main component used in this circuit is the 555 timer the components of Op-Amps are used. Op-Amps are a versatile component that can be found in several circuits for several uses. In the case of the 555 timer the Op-Amps are used as comparators or a way to convert the analog capacitor charging and discharging signals into the square wave digital signals. An Op-Amp is a very simple component have three leads the inverting pin, non-inverting pin and the output pin. The component follows a simple rule that the output is equal to $A\ (V\ (+)\ -\ V(-))$ where the V(+) is the non-inverting pin, V(-) is the inverting pin and A is a number close to infinity. What this means when dealing with an Op-Amps comparator is that if V(-) which acts as the reference voltage is greater than V(+) the Op-Amp open-loop equation will be $A\big(-V(-)\big)$ providing a value that will be equal to the -V rail providing a signal of 0 digitally. If the V(+) is greater than the V(-) value the equation will be $A(V(+))$ which is a value equivalent to the +V rail providing a 1 signal.



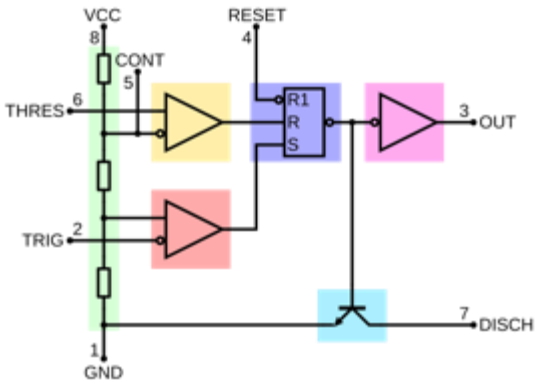
The other significant component is the SR Latch. This component stands for set and reset for the two pins function in the NAND gate configuration. If the S value receives a high value, then that triggers both the R pin and the inverted Q pin to be set to zero making Q a high value. Now the only way that the Q value can be turned back to a low value is with the input from the R value. In the state when Q is high the S value has no influence on an altering signal on either pin. This is because the NAND gates will only alter a signal when both pins are low and because the output of Q is wired to the other input from the S pin no matter what input is applied it cannot change. The only change occurs when the R value goes high making Q low and if S is still low the inverted Q becomes high. This creates a configuration that allows for a change in state to only occur when the other pin is altered.

Procedure

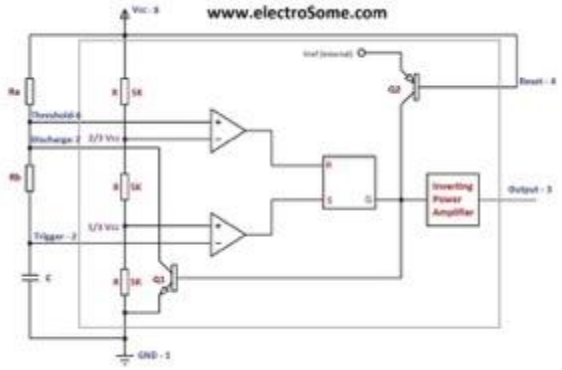
The clock is built of the three 555 timer circuits all constructed differently to create the three outputs necessary to have various square wave outputs that can reach the program counter. The three outputs created are the astable which outputs a square wave which a frequency and duty cycle based on the resistors and capacitors as explained above in the theory. The second being the monostable mode that has a specific duty cycle as well but only runs when charged by a push button press. The final being the bistable mode which only turns on when charged by a voltage signal and will turn of if no signal is provided making it a debounced button. All three of these signals play a role in the NAND gate logic configuration which adds a haul signal if stopping the counting is ever necessary.

Parts Table	
Quantity	Description
3	NE555 IC
2	SN74LS00N(NAND)
1	SN74LS262AN(PC)
2	1 mF Capacitor
1	1 nF Capacitor
1	100 kΩ Resistor
1	1 mΩ Potentiometer
2	SPST Push Button
1	SPDT Slide Switch
1	4 pin DIP Switches

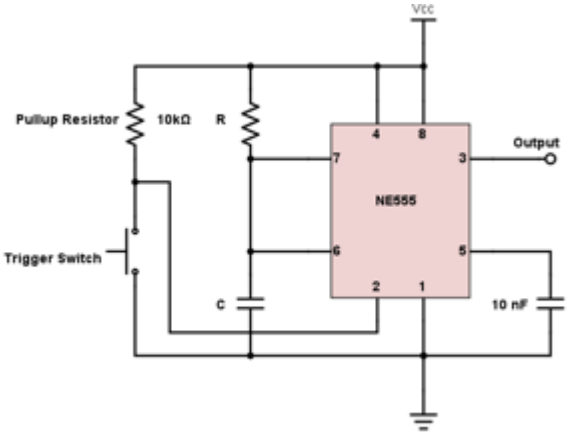
The first output to be examined is the astable 555 timer circuit. This circuit involves a potentiometer, two resistors and a capacitor. The goal behind the configuration is to have an oscillating square wave output. To understand how the 555 is able to do this the image to right displays the inside of the 555 timer. In the configuration of the astable mode the capacitor is place on the trigger pin and wired to ground, while the potentiometer is placed between the discharge pin and the threshold pin while the smaller 1 kΩ is then placed between the VCC and the discharge pin. The circuit diagram then deals with the internal components of two Op-Amps acting as comparators, an SR Latch and a transistor.

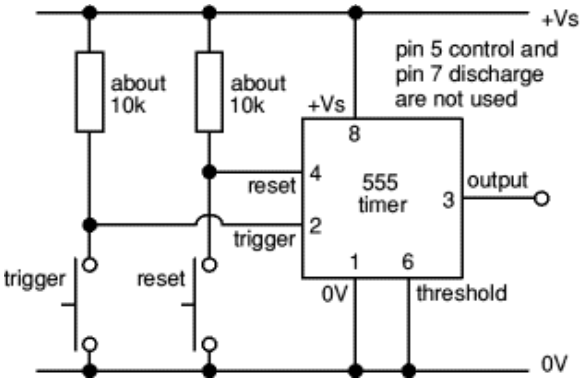


When voltage is applied to the circuit the first thing that occurs is it flows through R1 and R2 and begins to charge the capacitor. The Op-Amps which act as comparators, as shown in the theory, will begin to have an increasing voltage value. The reference voltage for the first Op-Amp is 2/3 of the voltage supply. When the reference voltage is exceeded the non-inverting pin a greater value which outputs a value of 1. The second Op-Amp has a reference voltage of 1/3 which is connected to the non-inverting pin meaning when the voltage exceeds the reference voltage the Op-Amps on signal will turn off. The speed at which the capacitor discharges the voltage to Op-Amps and how quickly it flows with the slowing down of resistors is all based on the passive component's configuration. The Op-Amp outputs are sent to an SR Latch which as learned will only change states when the other value changes so after the second Op-Amp exceeds the 1/3 VCC reference voltage no change will occur until the 2/3 reference voltage on the first Op-Amp is exceeded. This is how the 555 Timer IC is able to operate in a stable mode.

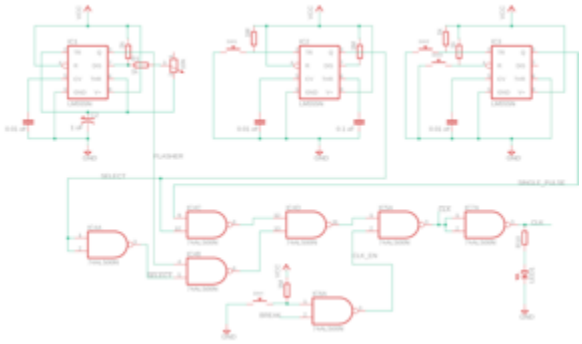


The second mode to be discussed is the monostable mode. A very frequent complication is the difference between the monostable 555 and the bistable 555. The difference being if the push button is applied once based on the RC (resistor capacitor) configurator the LED will stay on for a set amount of time. The bistable only turns on when a signal is applied. The configuration for the monostable 555 can be very similar to that of the astable 555 with the only changes being no resistor between pin 6 and 7 and instead of voltage charging into pin 2 it is an immediate high signal provided by a button. When the button is applied a signal of zero is brought through pin 2 which brings a high signal through pin 2 bring a high value to the output. After the button is let go the capacitor is discharged through a transistor found in the center which will discharge at a set amount of time due to the RC configuration.

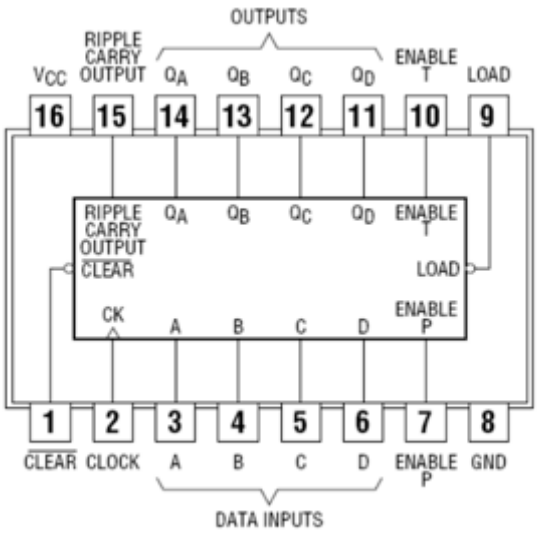




The astable and monostable modes on the 555 timer are both necessary for feeding the PC but the ability to swap between the modes is achievable by using a bistable 555 timer configuration. This circuit involves two inputs from buttons that can be substituted for a slide switch with a base pin of ground. The bistable 555 takes on the same configuration for pin 2 which is an instant low signal triggering the set latch. Then a reset pin with an inverting gate imbedded inside provides a reset latch that can turn off the output. This provides two pins that with a ground signal can either provide a signal of one or a signal of zero. These values are coming from the set and reset pins allowing for no bouncing from the slide switch.



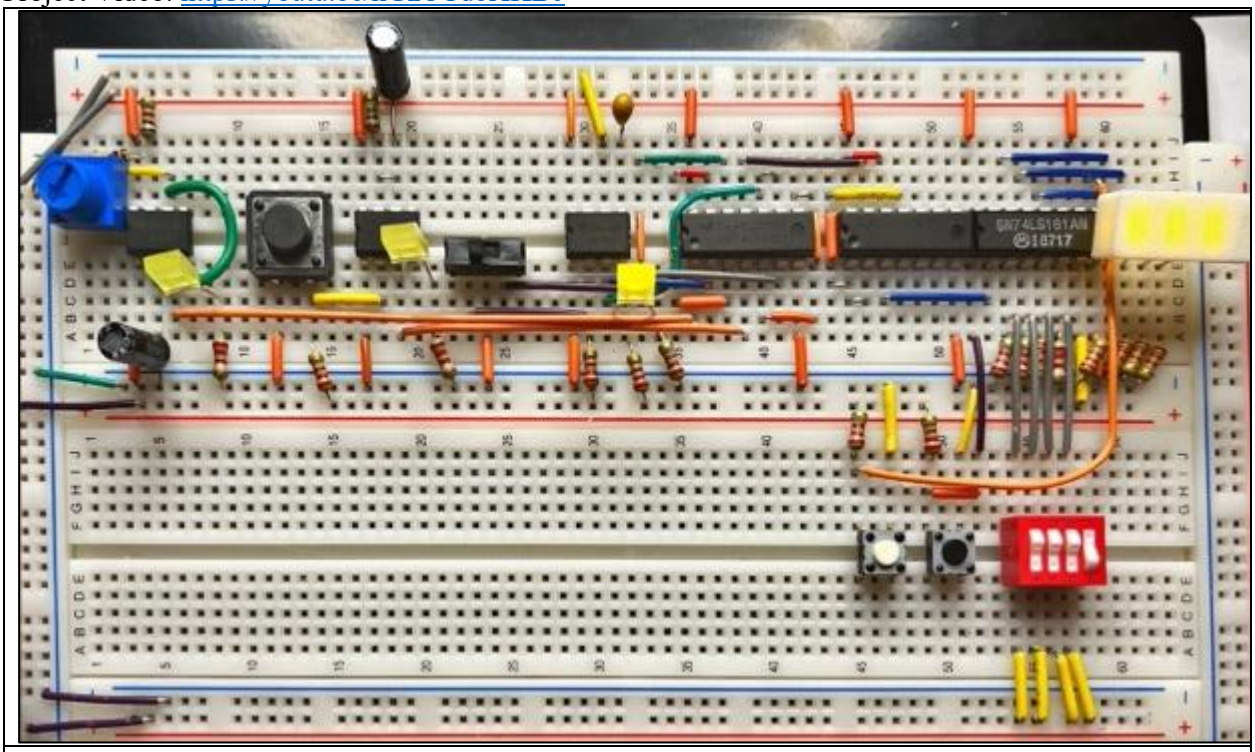
With all three configurations of the 555 a simple logic circuit can now be created to have a way to swap from monostable mode to astable mode in a debounced form as well as an additional haul pin for the ability to stop the counting. The logic is a seven NAND gate configuration that completes the function described above. As displayed in the schematic to the right it begins with a select pin from the bistable 555 sending the high signal to either the monostable or astable mode. Based on which output is selected it is sent through a NAND gate with a pin pulled high meaning no high outputs can be created without the low value from the halt value. This is how the selection and haul pin are included in the clock.



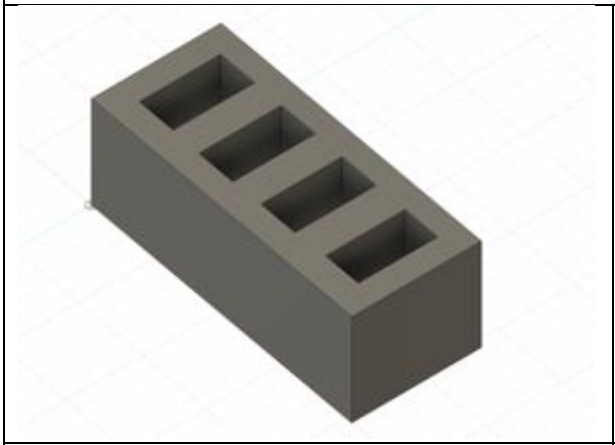
The output from the NAND gate configuration is then sent to the IC 74LS161. This IC is a program counter with several pins with additional functions helpful for program counting. The first important part is pins 2, 11, 12, 13 and 14. These are the clock pin and output pins part of this counter. The pulses are sent to the clock pin which are outputted through a 4-bit counter. The counter will continuously count from 0-15 in a time range based on the RC configurations of the monostable and bistable 555s. When in this state the pins load, clear, enable T and enable P are all set to high for normal operation. In this configuration the pins clear and load are set with a pull-up resistor button configuration. This then has the input pins wired to 4 DIP switches that are all pulled low unless the voltage signal is applied. This means that if the button is pressed after the next clock signal is applied instead of incrementing by one the output value is equivalent to the input value. For the clear pin when the button is pressed the output value goes to zero. Enable T is a pin that enable the counter to increment by 1 if it is set to zero the condition is not met and counting stops. Enable P is more of global enable pin that allows for the chips function or not.

Media

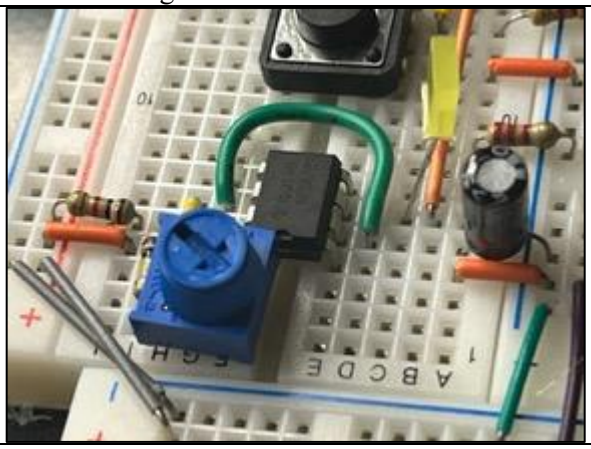
Project Video: https://youtu.be/kGzOTdbXKBc



CHUMP Clock and Counter Configuration



Bar graph LED Holder



Close-Up of Astable 555 Configuration

Reflection

The building of this first part of the CHUMP project has gotten me slightly excited. Although I have loved our software-heavy projects, the CHUMP interests me for the return to essential electronics components and their ability to make super exciting machines. I liked the return of the 555 in my report this time because, with my research for my long ISP and my work over the entire grade 11 year, I can see my progress in knowledge of electronics components. Overall, I am excited for the next few DERs centred around CHUMP; I am already looking ahead at what's coming by watching some past ACES videos and thinking about ways to get creative with such an exciting project.

Project 3.2.2 CHUMP: EEPROM

Purpose

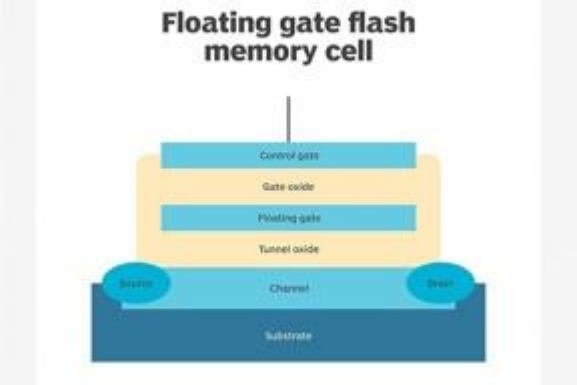
The purpose of the CHUMP: EEPROM is to create a program Read-Only Memory (ROM) read chumanease software. In this section, the electrically erasable read-only memory (EEPROM) has the purpose of read the five lines code by David Feinberg and personally written code. Additionally, components of an Arduino Nano, shift registers and 7-segment displays make this possible.

Reference

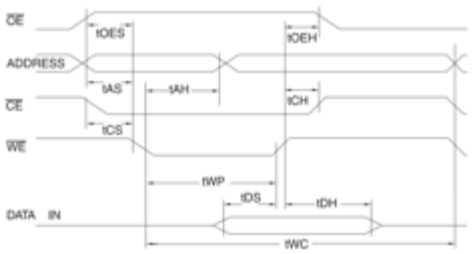
Project Reference:http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html#PROGRAMEEPROM
AT28C16 data sheet: http://cva.stanford.edu/classes/cs99s/datasheets/at28c16.pdf
EEPROM Description: https://www.youtube.com/watch?v=BA12Z7gQ4P0&t=1184s
EEPROM Burner Description: https://www.youtube.com/watch?v=K88pgWhEb1M
Inspiration for code: https://github.com/beneater/eeprom-programmer/tree/master/eeprom-programmer

Theory

The theoretical side behind EEPROM is understanding how this chip can function. The base of all EEPROMs is built on floating-gate transistors or FGTs, as shown on the right. These FGTs work with a metal-oxide transistor to hold the memory in the FGT and erase the memory when a sufficient amount of voltage is applied. This works because of the basic construction behind a transistor. The rule behind transistors is that voltage cannot pass from the source to the drain unless a current flows from the control gate to the drain. The other fact to note is that all transistors have a part called the depletion region acting as the gate to prevent electrons from passing through. Only when a sufficient voltage drop occurs can the region be passed through. This means that in this FGT, electrons move through the floating gate only when a sufficient voltage is applied to the control pin and a large enough voltage drops into the drain. When the gate leaves, electrons will still be in the floating gate, providing the binary state of zero. When 32,768 transistors, two per byte, are configured correctly, the creation of the AT28C16 is possible.



The other theoretical aspect of the EEPROM is how it is able to write out a new set of data of to the chip. The image on the right has been pulled from the data and shows how write enable works with all other pins. This provides the concrete idea of how the EEPROM will work with the Arduino. The first line to look at is the setup line, values output enables, address setup and chip enable need 10 ns to have a stable binary value that does not interfere this represents the time before write cycle time occurs. The write enable then goes through a 1000 ns time period that informs the chip of initiation of writing. After this time frame the write enable goes high and the data after another moment becoming stable will send the have the data interpreted. Less important signals include tOEH, tCH and tDH which is the amount of time after write enable goes high the control enable, data input and output enable must stay high. Configurations of these signals and their time increments are detailed further the in-code portion.

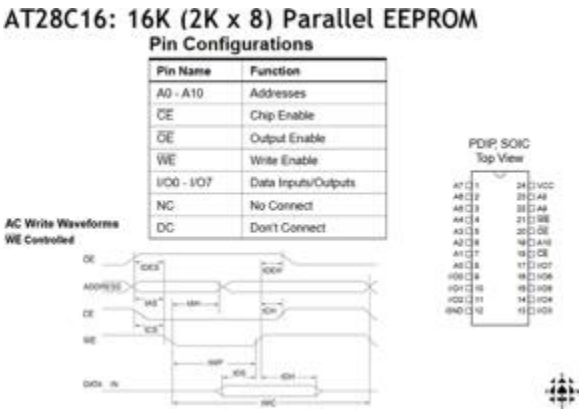


Procedure

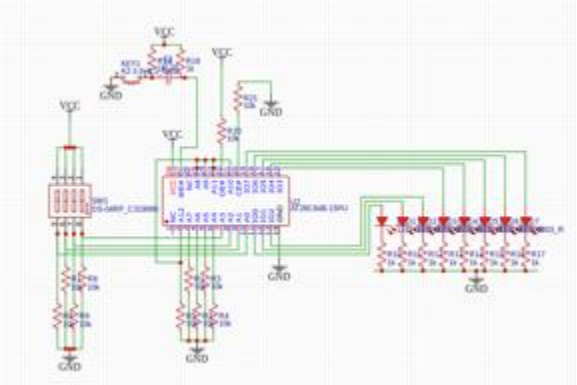
The EEPROM portions of the CHUMP uses the previous 4-bit counter creation to make the it possible for the EEPROM to read line by line. For refreshment, the configurations of astable and monostable 555 timers are sending clock signals to the SN74LS262AN which configures this counting into 4-bit counting. The 4 bits 1,2,4, and 8 are then sent to address lines 0,1,2, and 3 on the EEPROM with all other address lines set to low. How this configuration makes it possible to read code line by line will be discussed later in the procedure. The additional circuit built for this section of the is a the EEPROM burner. This does not play a role in the CHUMP final configuration but is the most efficient way to load 8-bit values into each address of the device. The only other pursuit taken is to manually switch addresses, load values and signal write enable to go high.

Parts Table	
Quantity	Description
3	NE555 IC
2	SN74LS00N(NAND)
1	SN74LS262AN(PC)
2	1 mF Capacitor
1	1 nF Capacitor
1	100 kΩ Resistor
1	1 mΩ Potentiometer
1	4 pin DIP Switches
3	AT28C16 EEPROM
2	Seven-Segment Displays
1	Arduino Nano
2	HC595 Shift Register

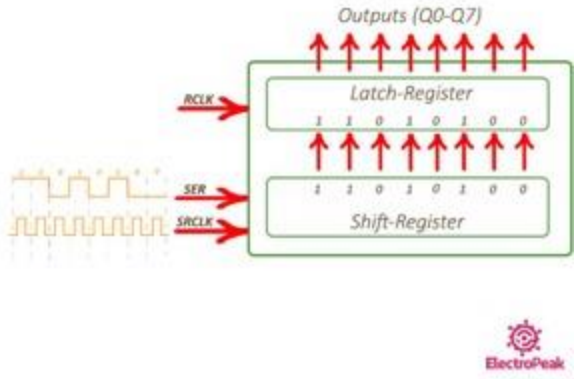
The most important chip to speak about in the procedure is the AT28C16. An EEPROM with 11 address lines to hold 2^{11} or 2048 8-bit values. The pinout includes these 11 address pins along with 8 I/O pins which display the 8-bit value in each address or, when configured differently, load an 8-bit value into the EEPROM. The other important pins discussed in the theory sections include the chip enable which should be set to active low for normal operations to have the enabled. The output enables which is set to high when low when writing to the EEPROM and high when displaying its values. This is set to high in the chump at all times because writing is going on in another circuit configuration.



The first circuit configuration used to understand how an EEPROM works is the manual burner. This uses 11 DIP switch with a resistor network along with 8 LEDs to indicate the states of all value in the addresses through the I/O pins. Write enable is hooked up to a reset capacitor configuration that will charge in the required timeframe of 1000 nanoseconds. The output enables state is held with a jumper wire that is connected to ground when writing a value to the I/O pins. When the configuration is wired to 5V the LED will output the 8-bit configuration of whatever is in the address selected by the DIP switches. The writing portion includes after setting OE low and setting what 8-bit value wanted in selected address to send high values to the I/O pins. The circuitry created is displayed in the media section.



A more efficient approach to burning values into the EEPROM is to use an Arduino Nano microcontroller. The one problem that comes with using the Arduino is lack of pins. Using only digital pins there are too many pins needing control on the EEPROM for the nano to be able to write and read from the chip. To solve this problem the use of 2 HC595 shift registers assist to use only 3 pins select from all 10 addresses and decide if output enable is high or low. For a refreshment the 595 register works by having a clock pin continuously pulsing to let the IC know to interpret the data being sent. Once 8 values are sent through the registers the latch is pulled high and this 8-bit value is sent through.



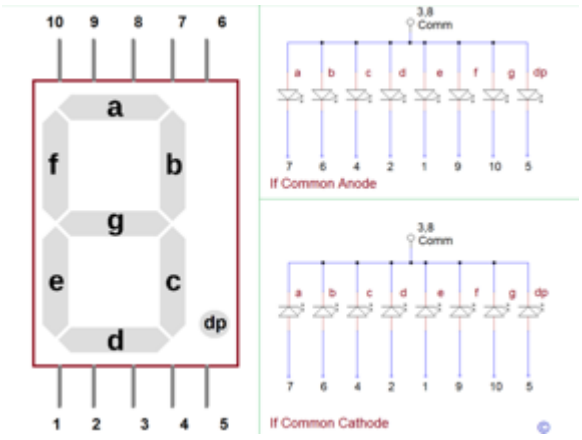
The code below illustrates how the nano is able to make this happen. For a better description the `shiftOut()` is rewritten to be able to be examined further behind the curtain. The function simply runs through a for loop 8 times with the line `for(uint8_t i=0; i<8; i++)` this is then followed by a logical statement to retrieve one value from the 8-bit value sent. Finally, the clock pin is turned on then off to create a clock pulse for the register to recognize. This function provides a detailed explanation for how the shift register actually functions.

Only three pins being used for all address values on the Arduino nano leaves 9 pins for the 8 I/O pins and the write enable pin. As spoke on in the manual section the write enable is a high value that is sent when wanting to set the address to 8 specific values the high value sets those values in. This is done using the concept of a higher amount of voltage to break the depletion region on the floating gate. The code behind this involves first taking the 8-bit value from the array `data[]` and from pins 5 to 12 running a for loop that has the 8 bit value start at pin 5 and taking the first of the 8 values using a logical "AND" statement.

```
void writeEEPROM(int address, byte data) {
  setAddress(address, /*outputEnable*/ false);
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
    pinMode(pin, OUTPUT);
  }

  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
    digitalWrite(pin, data & 1);
    data = data >> 1;
  }
  digitalWrite(WRITE_EN, LOW);
  delayMicroseconds(1);
  digitalWrite(WRITE_EN, HIGH);
  delay(10);
}
```

For an additional part of the display two more EEPROMs are used to be decoders of hexadecimal values into values that comply with a seven-segment display. The displays are built with each of the 7 lines that build up the number being individual LEDs. The LEDs are connected to either a common anode or common cathode and then signals are individually sent to each LED to turn on that line. If the value 8 is wanted to be turned on the value sent to the display must be all 0 if in common anode configuration. To have a way to turn the value in the address to displayable values two more EEPROMs are used to convert the values. The EEPROMs acts as a type of array in hardware. This means if the hexadecimal value 8 is sent to an EEPROM the output should be all zeros on address 8. Meaning if values from bytes 0-7 are sent to one EEPROM and bytes 8-15 are sent to another EEPROM burned the same way the values are able to be displayed on seven segment displays. The reason behind the addresses is to separate the upper and lower nibble.



Code

```
// PROJECT : 3.2.2 CHUMP:EEPROM
// PURPOSE : To burn the hexadecimal 8-bit values into EEPROM addresses
// Author  : M. Zischka
// COURSE  :ICS4U-E
// DATE    : 10/27/24
// MCU     :328 P
// STATUS  :Working
//Reference: Ben Eater: https://github.com/beneater/eeprom-
programmer/tree/master/eeprom-programmer
// NOTES   :

#define SHIFT_DATA 2
#define SHIFT_CLK 4
#define SHIFT_LATCH 3
#define EEPROM_D0 5
#define EEPROM_D7 12
#define WRITE_EN 13
 * Output the address bits and outputEnable signal using shift registers.
 */
void setAddress(int address, bool outputEnable) {
  shiftOutRegister(SHIFT_DATA, SHIFT_CLK, MSBFIRST, (address >> 8) | (outputEnable ?
0x00 : 0x80));
  shiftOutRegister(SHIFT_DATA, SHIFT_CLK, MSBFIRST, address);

  digitalWrite(SHIFT_LATCH, LOW);
  digitalWrite(SHIFT_LATCH, HIGH);
  digitalWrite(SHIFT_LATCH, LOW);
}
/*
 * Read a byte from the EEPROM at the specified address.
 */
byte readEEPROM(int address) {
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
    pinMode(pin, INPUT);
  }
  setAddress(address, /*outputEnable*/ true);

  byte data = 0;
  for (int pin = EEPROM_D7; pin >= EEPROM_D0; pin -= 1) {
    data = (data << 1) + digitalRead(pin);
  }
  return data;
}
/*
 * Write a byte to the EEPROM at the specified address.
 */
void writeEEPROM(int address, byte data) {
  setAddress(address, /*outputEnable*/ false);
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
    pinMode(pin, OUTPUT);
  }

  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
    digitalWrite(pin, data & 1);
    data = data >> 1;
  }
  digitalWrite(WRITE_EN, LOW);
  delayMicroseconds(1);
  digitalWrite(WRITE_EN, HIGH);
  delay(10);
}
/*
```

```
 * Read the contents of the EEPROM and print them to the serial monitor.
 */
void printContents() {
  for (int base = 0; base <= 255; base += 16) {
    byte data[16];
    for (int offset = 0; offset <= 15; offset += 1) {
      data[offset] = readEEPROM(base + offset);
    }

    char buf[80];
    sprintf(buf, "%03x:  %02x %02x %02x %02x %02x %02x %02x %02x   %02x %02x %02x
%02x %02x %02x %02x %02x",
            base, data[0], data[1], data[2], data[3], data[4], data[5], data[6],
data[7],
            data[8], data[9], data[10], data[11], data[12], data[13], data[14],
data[15]);

    Serial.println(buf);
  }
}


// byte data[] = { 0x82, 0x10, 0x21, 0x62, 0xa0, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff,0xff, 0xff, 0xff,
//         0x08, 0x62, 0x07, 0x66, 0x82, 0x50, 0x60, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff}; Chumpanese code

//byte data[] = { 0x3f, 0x30, 0x5b, 0x4f, 0x33, 0x7b, 0x7d, 0x6f, 0x7f, 0xdf, 0x77,
0x1f, 0x4e, 0x3d, 0x79, 0x71 }; hex decoder

void setup() {
  // put your setup code here, to run once:
  pinMode(SHIFT_DATA, OUTPUT);
  pinMode(SHIFT_CLK, OUTPUT);
  pinMode(SHIFT_LATCH, OUTPUT);
  digitalWrite(WRITE_EN, HIGH);
  pinMode(WRITE_EN, OUTPUT);
  Serial.begin(57600);

  // Erase entire EEPROM
  Serial.print("Erasing EEPROM");
  for (int address = 0; address <= 2047; address += 1) {
    writeEEPROM(address, 0xff);

    if (address % 64 == 0) {
      Serial.print(".");
    }
  }
  Serial.println(" done");


  // Program data bytes
  Serial.print("Programming EEPROM");
  for (int address = 0; address < sizeof(data); address += 1) {
    writeEEPROM(address, data[address]);

    if (address % 64 == 0) {
      Serial.print(".");
    }
  }
  Serial.println(" done");
```

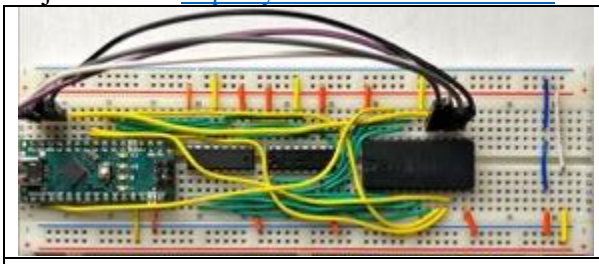
```
  // Read and print out the contents of the EERPROM
  Serial.println("Reading EEPROM");
  printContents();
}
void shiftOutRegister(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t
val) {
    DDRD |= (1 << dataPin) | (1 << clockPin);

    for (uint8_t i = 0; i < 8; i++) {
            if (val & (1 << (7 - i))) {
                PORTD |= (1 << dataPin);
            } else {
                PORTD &= ~(1 << dataPin);
            }
        }

        // Toggle clock pin
        PORTD |= (1 << clockPin);
        PORTD &= ~(1 << clockPin);
    }
}
void loop() {
}
```

Media

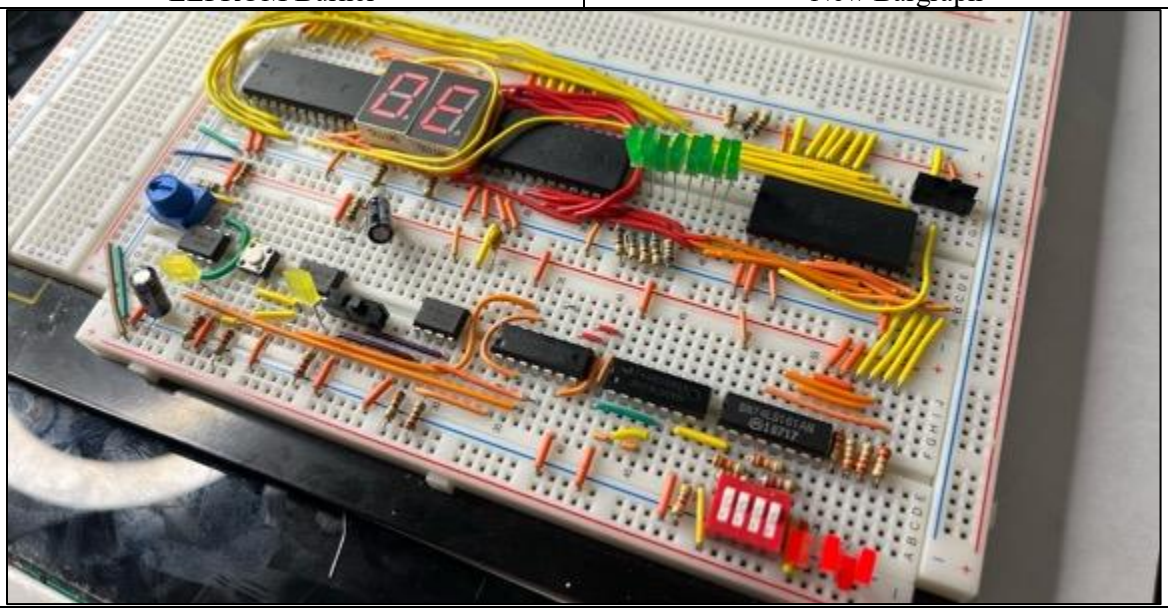
Project Video: https://youtu.be/rRGPcxRw9L4



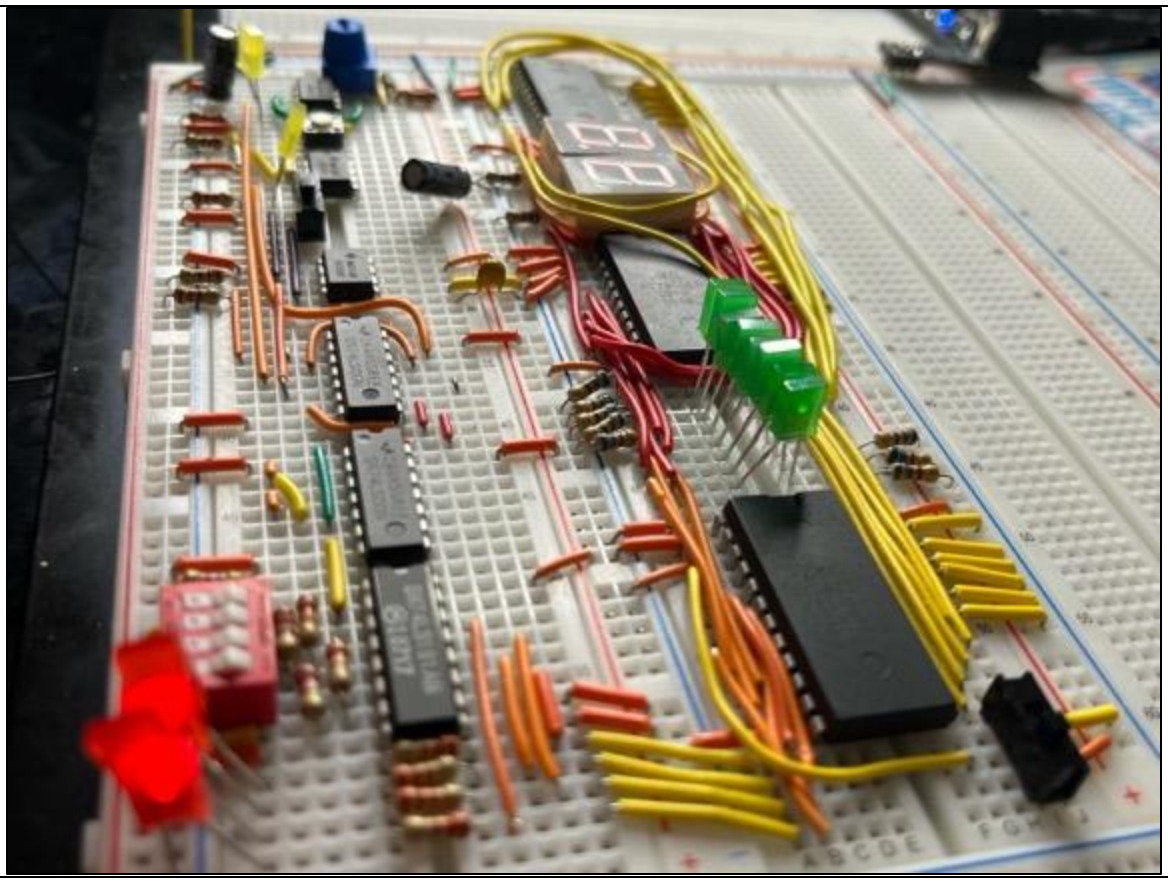
EEPROM Burner



New Bargraph



CHUMP: Front View



Chump; Side View

Reflection

I am really starting to like the direction this project is going in. The amount of work I put in this week after thinking because I had only one summative remaining for Thursday, Friday and Saturday, being the DER, I put in over 15 hours. Complicated wiring took time and learning how each piece worked. My favourite part of the project was the creation of Chumpanese code which really deepened my understanding on the topic. I am happy with the strides I made to get the EEPROM working with the 7-segment display.

Although I am pursuing electrical engineering, the beauty of a project like CHUMP is something else. Understanding the components at such a concrete level and with the time put in makes the piece created something amazing. Only having the program and counter without the very interesting things like understanding how buses work or creating the control ROM and RAM are things I am excited for with the future of this project. Additionally, I have started the schematic for a CHUMP:PCB and build on it as every project furthers.

I am excited for what comes next.

3.2.3 CHUMP: ALU Explorer

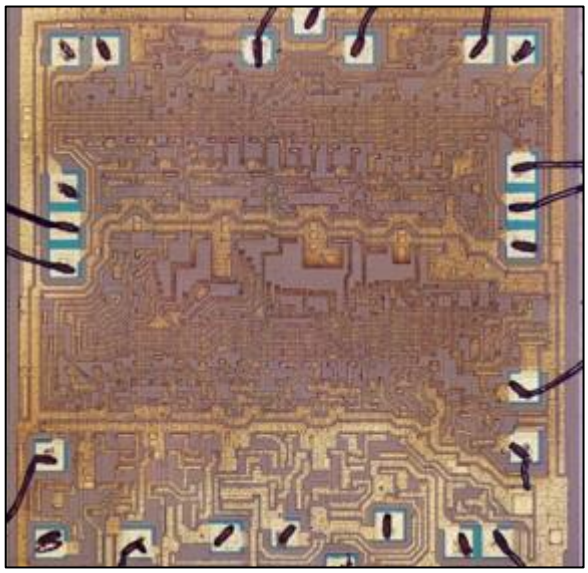
Purpose

The purpose of the ALU explorer is to explore the intersection between logical and arithmetic operations through the SN74LS181N. The contents of the ALU are displayed using the additional component of a LED bargraph.

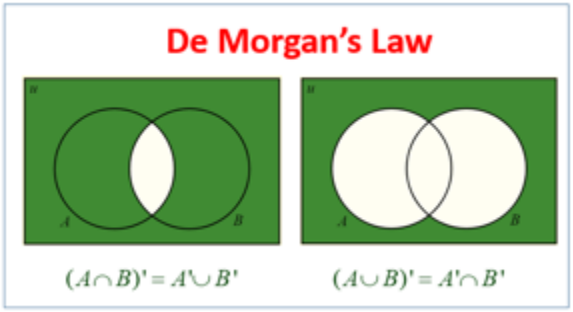
Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html#ALU
Datasheet:https://www.ti.com/lit/ds/symlink/sn54s181.pdf?ts=1730992453234&ref_url=https%253A%252F%252Fwww.google.com%252F
Deeper IC Analysis: https://www.righto.com/2017/03/inside-vintage-74181-alu-chip-how-it.html

Theory



The first theoretical aspect behind the ALU explorer to understand is the intersection between combinational logic, the way most computers operate at a core level, and math, the even more fundamental aspect of computing. The chip displays the ties between the two beautifully through the ability to complete OR, AND and NOT operations as well as several arithmetic operations. A review of this; the AND gates will provide an output of 1 only when both inputs are high. The OR gate will provide a 1 output when either of the two inputs are high. Lastly, the NOT gate acts as an inverter switching the binary state of the bit. These are computed using logical math. A topic brushed upon in the binary calculator, all 4 bits are either compared to or immediately sent to the F outputs. In logical math binary values carrying is a common occurrence and is achievable in this chip. The photo on the right shows a die photo of this SN74LS181N constructed of resistors and transistors.



The datasheet on the ALU shows the logical side and the arithmetic side. One key concept taken away from this datasheet reading is De Morgan's Law. To understand this law the first concept to comprehend is how AND and OR gates can be expressed as mathematical operations. When examining how an AND gate functions it can be seen that there is an output of 0 when any 0 is present in the value computed. A crossover can be made as in multiplication the value zero in any equation has the entire output also be zero. With OR gate any 1 being present will a zero outputting a 1 reflecting addition operation receive this arithmetic operation. With both of these arithmetic operations found in the circuit these are also found being put under the NOT operation. The equation in datasheets: $(A + B)'$ what is so interesting about this is that it does not work like normal mathematics when collapsing a bracket. When an equivalent bracket that is collapsed needs to be created in equation is would look like $A' \times B'$. This shows the workings of combinational logic with math and how AND and OR gates can work in equivalent operations.

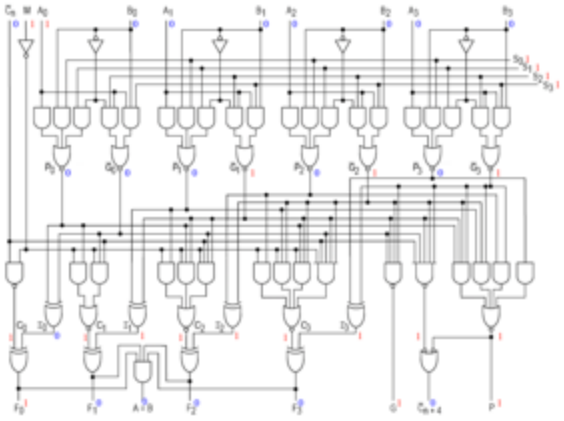
Procedure

Parts Table	
Quantity	Description
1	SN74LS181N
2	1x10 LED Bargraph
2	16 pin Rocker Switch
1	ALU Explorer PCB
1	10 kΩ resistor
4	10 kΩ bussed res-network

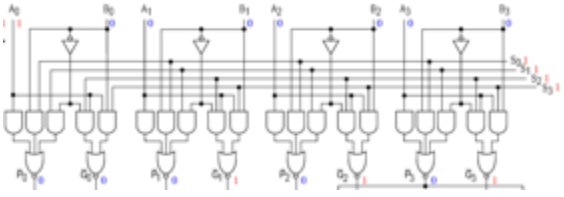
The ALU explorer is a simple circuit that has the function of displaying the F outputs from 14 inputs. The inputs can be broken down into four sections being the select pins that choose which function is being called. The A inputs, one side of the input comparisons. The B inputs which are the other side of the input comparison. Finally, the C and M pins which play a role in deciding if the functions to select from with the S pins are arithmetic functions or logical functions. The C pin is used when carrying inputs from a second ALU. It must be selected to a low to decide the arithmetic equations needed. This is how the ALU functions at a surface level.

ACTIVE LOW INPUTS & OUTPUTS		ACTIVE HIGH INPUTS & OUTPUTS	
LOGIC (M = H)	ARITHMETIC** (M = L) (C_n = L)	LOGIC (M = H)	ARITHMETIC** (M = L) (C_n = H)
$\overline{A}$	A minus 1	$\overline{A}$	A
$\overline{AB}$	AB minus 1	$\overline{A+B}$	A + B
A + B	AB minus 1	AB	A + B
Logical 1 minus 1		Logical 0 minus 1	
$\overline{A+B}$	A plus (A + B)	$\overline{AB}$	A plus AB
$\overline{B}$	AB plus (A + B)	B	(A + B) plus $\overline{AB}$

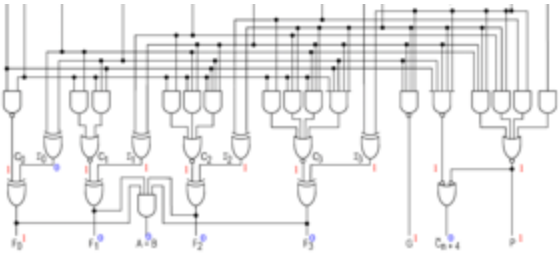
When examining the datasheet of the ALU one notable feature is the factor that the IC is both an active high and active low IC. This means that it will provide arithmetic outputs for both 0s and 1s as the signals. This makes the pin even more versatile able to compute 48 operations of inputs. The wiring used behind this the use of pull-down resistors to use active high inputs. This means that switches are commonly wired to ground unless a 5V signal is applied through and flows through a current limiting resistor. Bourns 10 kΩ are optimal for this operation for the easy wiring and little interference.



To fully understand an IC like this another circuit can be constructed. From the building of Ken Sheriff, the combinational logic configuration of the IC can be created out of nothing but AND, OR, XOR, NAND and NOR gates. This provides a much deeper explanation into the SN74LS181N. The logical configurations can be broken up into two different sections to show the steps from input to output on how this circuit works. The two being the upper region that has all twelve inputs from the A, B and S value that are put through a configuration to turn into two 4-bit value that go through a comparator. The schematic is an AND gate dominant circuit for the reason that all outputs are outputted through NOR gates.



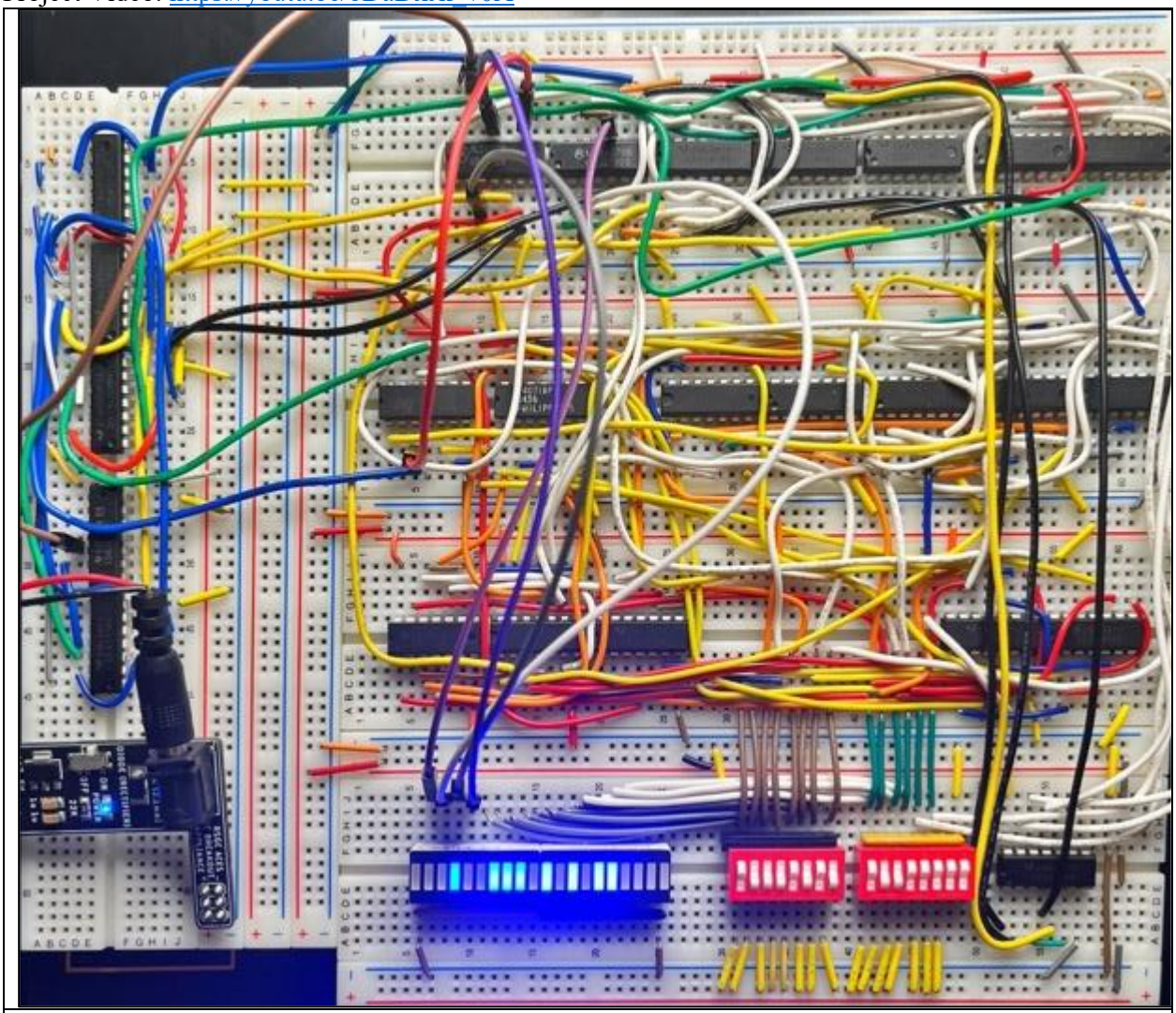
The first section known as the sorting side deals with all twelve inputs and does not take the last two C and M inputs into account this has to do with the fact that it is trying to configure the twelve inputs into 8 and does not have to do with comparing the values with a selected function yet. This is done by wiring the A inputs to two of the AND gate while wiring B and NOT B to an additional two. 4 of the 5 AND gates are being anded with and S input. The reason behind this is the logical configuration behind this circuit will depict which values are high and low by the fact that the S inputs are configured to all 4 sorting blocks. This creates a sorting way to figure out which 12 values are high and low using 8-bits.



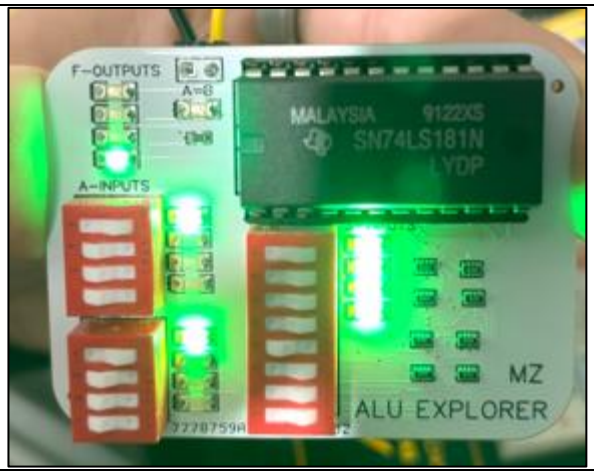
The second side is the comparison part of this circuit. When only 8 values are left to compare the values are sent through a continuously increasing amount of AND gate configurations. The reasoning behind the increasing number of AND gates have to do the values required to turn that output value high and how much information is needed. To figure out if the first value F0 is high only outputs from the A0 and B0 sorting are necessary. However, by F3 not only are those same inputs needed but every input before as some arithmetic values will increase and carrying over to the next bit. In building this combinational logic circuitry it is seen that some logic gate required three to five inputs. As a fixture to not having these types of inputs wiring 4 AND gate together can create this same type of logic gate.

Media

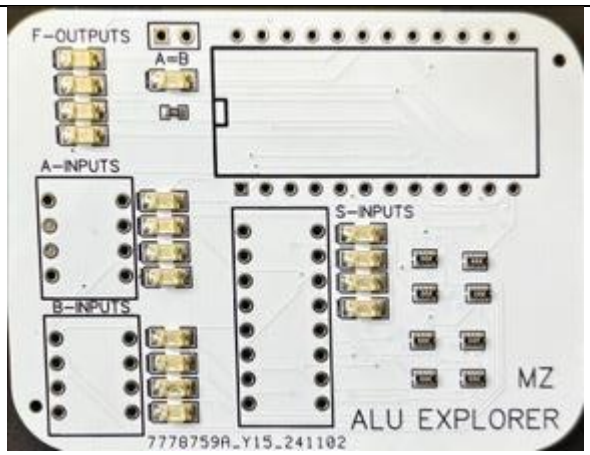
Project Video: https://youtu.be/oDaDxJk_v0A



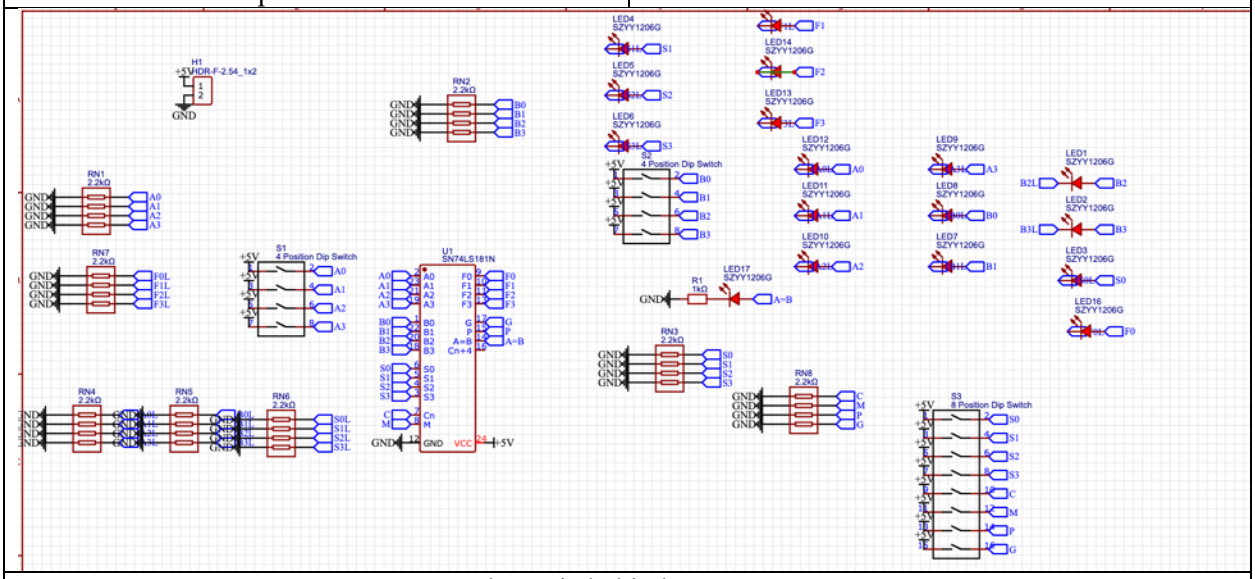
SN74LS181N combinational logic circuit showing A XOR B A=0111 B = 0101 (F Outputs Leftmost)



ALU Explorer PCB: F = A



Raw PCB



Schematic behind ALU PCB

Reflection

Something I find interesting to do in school is when taking a small project that I find interest in and turning into a much larger project. This is something I was able to do with this project because of the atmosphere of the ACES program pushing me to do more if I have time left. Coming up on Thursday I had finished the breadboard and the PCB version but still had time. Earlier in the week there had been talk about a third term project around combinational logic. This grabbed me from the interest I found in my first ISP around just using logic gates for simple arithmetic. I thought about the idea but talked myself out of it at first. Then for some reason I was too interested in the idea not to try it before the Sunday night submission. The total time took about 14 hours just for the combinational logic but the step-by-step approach I took to make sure it worked and, in the end, when wiring the final XOR gate I was so happy with my creation. Very proud of my accomplishments and hope to apply the same effort to the final CHUMP project.

3.2.4 CHUMP: Final Build

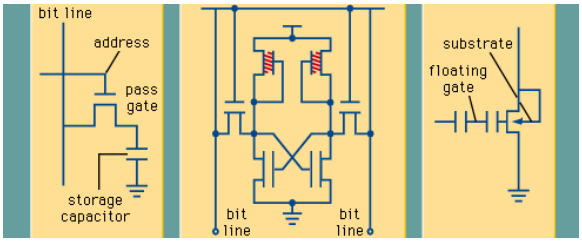
Purpose

The purpose of the CHUMP is to compute up to 16 lines of assembly language instructions using two EEPROMs, an Arithmetic Logic Unit (ALU), an accumulator, a RAM memory, an address buffer and a program counter. All the ICs together create the ability to write simple instructions that the computer undergoes.

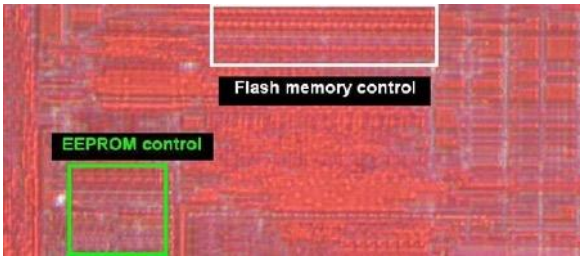
Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html#Build
ALU Information: https://www.righto.com/2017/03/inside-vintage-74181-alu-chip-how-it.html
EEPROM datasheet: http://cva.stanford.edu/classes/cs99s/datasheets/at28c16.pdf
EEPROM Burner: https://www.youtube.com/watch?v=K88pgWhEb1M&t=3048s

Theory



One theoretical concept to understand are the two types of RAM that are used being the SRAM and DRAM. DRAM is holding the memory values in the charges of capacitors and SRAM is holding the values in a transistor configuration acting like a flip-flop. Both of these types of RAM when power is turned off quickly either those the energy powering the transistors to hold energy or have the capacitors dissipate almost immediately. What makes this different from the EEPROM is that in an EEPROM the electrons are physically trapped in a memory location no matter if power is on or off. The image on the right show inner workings of a RAM IC.

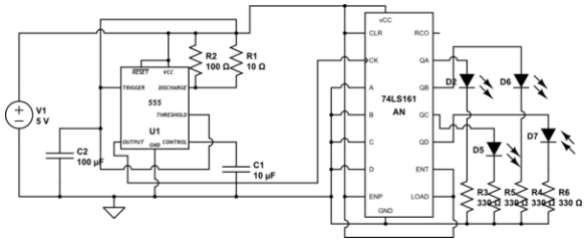


The next theoretical concept to understand is the mathematical Look Up Table or LUT that can be used to allow for more mathematical operations to be done. The EEPROM as learned in the program ROM is a look up table. If the B value is bit shifted by 4 values then the two can be combined into an 8 bit value sent to the address pins. This creates a way for logical inputs to be mapped outputs that are multiplied, divided or any other operation wanted. The image on the right shows a 4-bit processor emphasizing how versatile using EEPROMs can be.

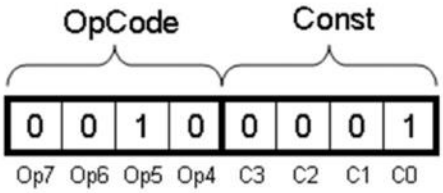
Procedure

The goal of the CHUMP, as stated above in the purpose, is to compute a 16-line instruction set. A pathway has been illustrated on how the CHUMP is able to do this but the introduction into this includes first loading the program in the EEPROM. The top 4 values are sent to the address pins of the control ROM while the lower 4 are sent to the MUX. The control ROM then decides based on the code if a value from the program ROM, the RAM or neither needs to be pulled for the instruction. When pulling a value from the RAM the address buffer sets what address the value is pulled from. If any mathematical operation is trying to be loaded then the ALU chooses from one of the thirty functions that are decided by the control ROM code. This entire process and other important incorporating bits will be understood, through a deep analysis of every IC and how they work together, in this report.

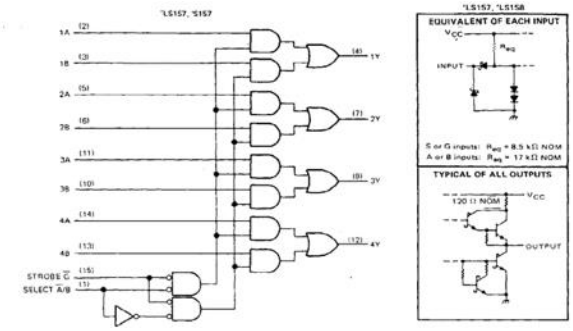
Parts Table	
Quantity	Description
1	74LS181N (ALU)
2	AT28C16 (EEPROM)
1	74LS161(PC)
1	74LS157 (MUX)
1	74LS377 (Accumulator)
1	74LS174 (Address Buffer)
1	74S189(RAM)
3	74LS04 (NAND)
3	NE555 (Timer)
1	74LS00(NOT)



The first section of the CHUMP to understand is the program counter. The full report in 3.2.1 goes into a deep analysis of this portion. A summary of this circuit starts with two 555 timer ICs set to either oscillate due to an RC configuration (astable mode) or turn of based on the charging and discharging of a single capacitor (monostable mode). Both inputs are sent through a NAND gate configuration that creates the ability to swap from either astable or monostable mode. The pulses of either 555 are then sent to the 74LS161 program counter IC. From just the clock pulses the IC acts as a standard 4-bit counter. The outputs of this 4-bit counter are sent to the address lines on the program ROM to be able to go from line to line in the code. The additions of the IC come from the clear and load pins. The ability to reset and change the program counter value is part of control ROMs ability. When the ALU is controlled to have A=B high and a value from the EEPROM both turn high then a low value is sent through the clear and load pins.

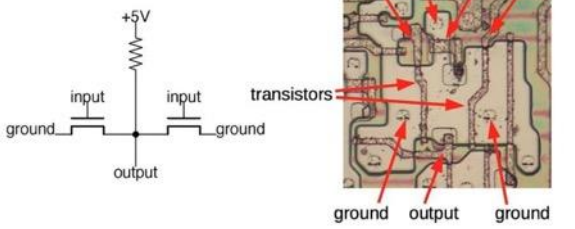


The next part of the chump, once the 4-bit counter is built, are the two EEPROMS. Possibly the most important computer building parts, the two EEPROMs make it possible to upload a program and to control every single IC to work together to satisfy the program. The first step in doing this is flashing the program ROM. This 8-bit output is split up into two 4-bit values. The high nibble is for the control ROM and the low nibble is a constant value. The EEPROM can be viewed as a look-up table. This means that an address is really just an 8-bit value stored in a single 4-bit input. EEPROMs are very useful for adding user inputs. The program is preset with whatever function is wanted. If for example if the high nibble is 0000 and the low nibble is 0111 then the function is to LOAD the value 7 into the accumulator. The exact way the 7 reaches the accumulator makes sense when the control ROM is understood. To understand how that function all other important IC's must be discussed first.



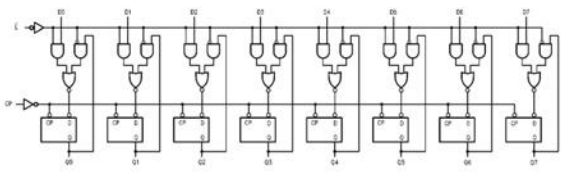
The next IC to be discussed is the SN74LS157. This IC is a 2 to 1 multiplexer that serves a very important role in the CHUMP to make it possible to choose either to a constant value or a RAM value. The multiplexer functions by taking in two inputs being the A and B inputs. A is wired to the constant value from the program ROM's low nibble and the B inputs are connected to the RAM outputs. The control code then has the duty of choosing if this multiplexer should be choosing from A or B. This is done using the select pin which will output A when select is a low value and output the B inputs when select is a high value. This is another versatile component that is seen at three points in the CHUMP. At one of the points the MUX is creating a way for the controlling to pick which pathway it should take to output. This point being if the value outputted from the ALU should flow through the multiplier EEPROM and then to the accumulator or straight to the accumulator. The other more interesting point is when using the MUX to create persistence of vision. The exact approach is spoken about in the Add-ons. The image on the right shows what the MUX truly is and shows the common select A/B line that must be high for an output.

The next component spoken about is the heart of the computer where all logical operations are achieved. This being the arithmetic logic unit. As spoken about in the ALU explorer this component is a large combinational logic circuit that is built to do 32 4-bit operations. The highlighted ones in the CHUMP are the A+B, A-B, F=1, and F=Ā. The first two operations are part of the actual functions described being adding a value to the accumulator or subtracting a value away.

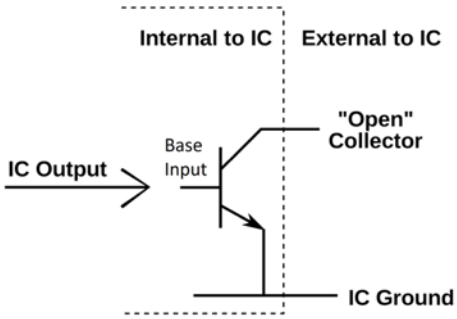


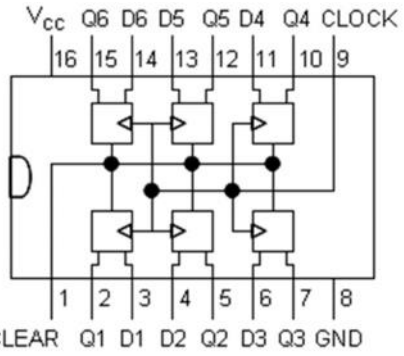
These two are done by having the accumulator wired directly to the A values on the accumulator and the MUX outputs being wires to the B values. The other two operations are useful during the functions that alter the program counter. In the case of F=1 the objective is trying to trigger the NAND gate that is connected to one control ROM line and one that is connected to the A=B pin on the ALU. For the GOTO function both must be high so this ALU selection will immediately turn it high. For the IFZERO function the A=B pin should only be triggered if the connects in the accumulator are 0. Meaning a Ā on zero would trigger this pin but anything else would not. The image on the right shows the contents of an ALU composed of transistor acting as logic gates which is all the ALU is made of. This transistor configuration is a NAND gate.

The next key IC part of this circuit is accumulator. This is the part of CHUMP where the OUTPUTs of computation can be viewed. The accumulator is an 8-bit register that act as a flip-flop when wired to a clock pin. The inputs of the accumulator are coming only from the ALU meaning if a constant value is wanted to be loaded directly to the accumulator, then the ALU must act as a simply a pathway being F = B. The outputs of this content are flowing to two important place. The outputs flowing back into the ALU are what make operations that involve altering the values in the accumulator possible. These functions include all arithmetic and GOTO functions. The other output being to the digital inputs of the RAM. The image on the right shows the accumulator at a logic gate level to allow for the enable pins use as well as the flip-flops which are connected to a clock pulse. The clock pulse in this case is connected to the same input as the PC.



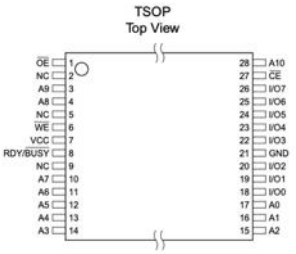
The next IC necessary for the creation of the cheap homemade computer is the memory component of the IC. The RAM stands for random access memory or volatile memory due to the fact that once powered off the values held in addresses are not kept. This component can be seen very similar to the EEPROM except that it cannot be programmed with values and then moved to another breadboard or else all values are lost as discussed in the theoretical section of the DER. In the LS189 a static RAM IC needs 8 inputs for the 4 outputs in provides to the B pins of the multiplexer. 4 of the inputs from the address buffer for what address to choose from and 4 inputs from the accumulator when writing. This IC needs several accommodations to work as a standard RAM chip in which an address can be set and pulled from. Firstly, the outputs from the RAM are inverted this problem is fixed by inverting the data inputs so that the outputs are the true accumulator values. The next issue having to do with the fact that the outputs are also open collector pins. This means that the IC's output is providing a pathway to ground meaning that the user must enable a pull-up resistor to get the true outputs.





The accessory to the RAM IC is the address buffer. This IC provides the function of giving the RAM value an address value. The IC used for this is the 74LS174 is flip-flop IC that holds the digital inputs in the IC until the next rising edge from the clock pulse and then will output the values into the RAM. The clock pulse is the same clock pulse that is hooked up to the program counter and the accumulator. The one other value that is hooked up to this hex flip-flop is the read/write enable value that is for the RAM value. The RAM acts in two ways either having a value written to the IC (WRITE) or if the IC should output what is in an address (READ). Yet because this is a flip-flop this pin the control value cannot be directly wired to the RAM it needs to be connected to same flip-flop as the inputs so when the inputs are provided to the RAM that is when the enabling of either read or write occurs.

Pin Name	Function
A0 - A10	Addresses
$\overline{CE}$	Chip Enable
$\overline{OE}$	Output Enable
$\overline{WE}$	Write Enable
I/O0 - I/O7	Data Inputs/Outputs
RDY/$\overline{BUSY}$	Ready/Busy Output
NC	No Connect



The final and most important part of the CHUMP processor is the control ROM. This EEPROM has the important task of receiving the 4-bit value from the high nibble of the program ROM and altering 10 values across all ICs just spoken about to match the function selected in the program. The way this works is by having the control ROM act as a look-up table accommodating the 13 instruction sets created by David Feinberg. For example, if LOAD which is the instruction 0000 then in address 0 and 8 bit value connected to 8 IC's will be outputted based on that 4 bit Op-Code. Because 10 enable pins need to be taken care of two of those pins must be manually wired and not go through the EEPROM. In this case, Op-code 4 can be wiring to the select pin on the MUX because and code wanting to pull from RAM will have an Op-Code with a 1 as the LSB. As well, the Cn value matches the Op-Code 5 value for the 4 instruction it is used for meaning it can also be manually wired. The table below shows what 8 values must be produced for every address. C means Constant I means It or RAM and A standing for accumulator

Instruction	Op7	Op6	Op5	Op4	Sel	ALU	S3	S2	S1	S0	M	Cn	Acc	RW	PC
Load C	0	0	0	0	0	B	1	0	1	0	1	X	0	1	0
Load I	0	0	0	1	1	B	1	0	1	0	1	X	0	1	0
Add C	0	0	1	0	0	A+B	1	0	0	1	0	1	0	1	0
Add I	0	0	1	1	1	A+B	1	0	0	1	0	1	0	1	0
Subtract C	0	1	0	0	0	A-B	0	1	1	0	0	0	0	1	0
Subtract I	0	1	0	1	1	A-B	0	1	1	0	0	0	0	1	0
StoreTo C	0	1	1	0	0	B	1	0	1	0	1	X	1	0	0
StoreTo I	0	1	1	1	1	B	1	0	1	0	1	X	1	0	0
Read C	1	0	0	0	0	B	1	0	1	0	1	X	1	1	0
Read I	1	0	0	1	1	B	1	0	1	0	1	X	1	1	0
Multiply A	1	0	1	0	0	B	1	0	1	0	1	X	1	1	0
Divide A	1	0	1	1	1	B	1	0	1	0	1	X	1	1	0
GoTo C	1	1	1	0	0	F=1	1	1	0	0	1	X	1	1	1
GoTo I	1	1	1	1	1	F=1	1	1	0	0	1	X	1	1	1
IfZero C	1	1	0	0	0	F= $\bar{A}$	0	0	0	0	1	X	1	1	1
IfZero I	1	1	0	1	1	F= $\bar{A}$	0	0	0	0	1	X	1	1	1

Code

```
// PROJECT : 3.2.2 CHUMP:EEPROM
// PURPOSE : To burn the hexadecimal 8-bit values into EEPROM addresses
// Author  : M. Zischka
// COURSE  :ICS4U-E
// DATE    : 10/27/24
// MCU     :328 P
// STATUS  :Working
//Reference: Ben Eater: https://github.com/beneater/eeprom-
programmer/tree/master/eeprom-programmer
// NOTES   :

#define SHIFT_DATA 2
#define SHIFT_CLK 4
#define SHIFT_LATCH 3
#define EEPROM_D0 5
#define EEPROM_D7 12
#define WRITE_EN 13
 * Output the address bits and outputEnable signal using shift registers.
 */
void setAddress(int address, bool outputEnable) {
  shiftOutRegister(SHIFT_DATA, SHIFT_CLK, MSBFIRST, (address >> 8) | (outputEnable ?
0x00 : 0x80));
  shiftOutRegister(SHIFT_DATA, SHIFT_CLK, MSBFIRST, address);

  digitalWrite(SHIFT_LATCH, LOW);
  digitalWrite(SHIFT_LATCH, HIGH);
  digitalWrite(SHIFT_LATCH, LOW);
}
/*
 * Read a byte from the EEPROM at the specified address.
 */
byte readEEPROM(int address) {
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
    pinMode(pin, INPUT);
  }
  setAddress(address, /*outputEnable*/ true);

  byte data = 0;
  for (int pin = EEPROM_D7; pin >= EEPROM_D0; pin -= 1) {
    data = (data << 1) + digitalRead(pin);
  }
  return data;
}
/*
 * Write a byte to the EEPROM at the specified address.
 */
void writeEEPROM(int address, byte data) {
  setAddress(address, /*outputEnable*/ false);
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
    pinMode(pin, OUTPUT);
  }

  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1) {
    digitalWrite(pin, data & 1);
    data = data >> 1;
  }
  digitalWrite(WRITE_EN, LOW);
  delayMicroseconds(1);
  digitalWrite(WRITE_EN, HIGH);
  delay(10);
}
/*
```

```
 * Read the contents of the EEPROM and print them to the serial monitor.
 */
void printContents() {
  for (int base = 0; base <= 255; base += 16) {
    byte data[16];
    for (int offset = 0; offset <= 15; offset += 1) {
      data[offset] = readEEPROM(base + offset);
    }

    char buf[80];
    sprintf(buf, "%03x:  %02x %02x %02x %02x %02x %02x %02x %02x   %02x %02x %02x
%02x %02x %02x %02x %02x",
            base, data[0], data[1], data[2], data[3], data[4], data[5], data[6],
data[7],
            data[8], data[9], data[10], data[11], data[12], data[13], data[14],
data[15]);

    Serial.println(buf);
  }
}



// byte data[] = { 0x82, 0x10, 0x21, 0x62, 0xa0, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff,0xff, 0xff, 0xff,
//         0x08, 0x62, 0x07, 0x66, 0x82, 0x50, 0x60, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff}; Chumpanese code

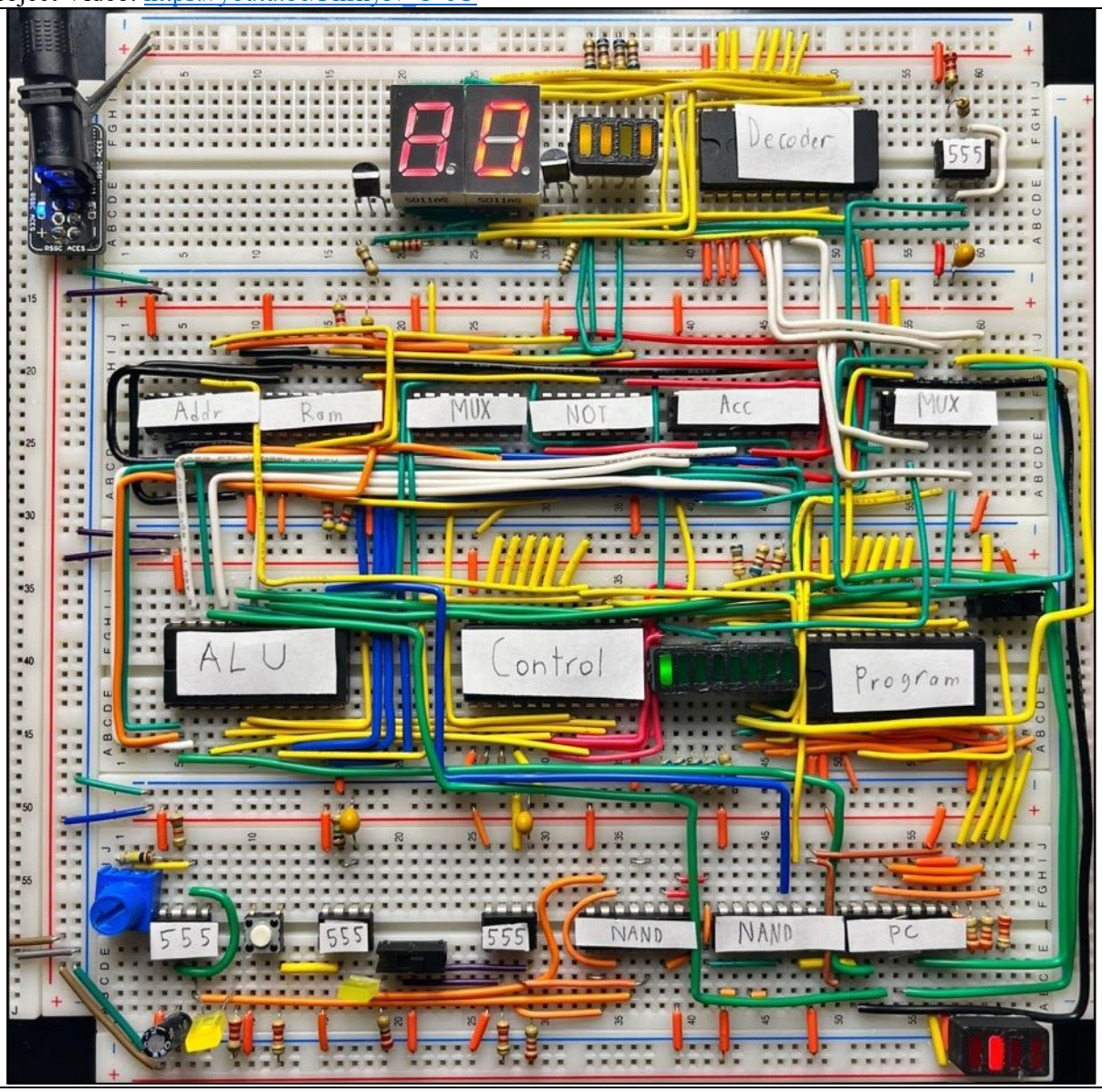
//byte data[] = { 0x3f, 0x30, 0x5b, 0x4f, 0x33, 0x7b, 0x7d, 0x6f, 0x7f, 0xdf, 0x77,
0x1f, 0x4e, 0x3d, 0x79, 0x71 }; hex decoder
void setup() {
  // put your setup code here, to run once:
  pinMode(SHIFT_DATA, OUTPUT);
  pinMode(SHIFT_CLK, OUTPUT);
  pinMode(SHIFT_LATCH, OUTPUT);
  digitalWrite(WRITE_EN, HIGH);
  pinMode(WRITE_EN, OUTPUT);
  Serial.begin(57600);
  // Erase entire EEPROM
  Serial.print("Erasing EEPROM");
  for (int address = 0; address <= 2047; address += 1) {
    writeEEPROM(address, 0xff);

    if (address % 64 == 0) {
      Serial.print(".");
    }
  }
  Serial.println(" done");
  // Program data bytes
  Serial.print("Programming EEPROM");
  for (int address = 0; address < sizeof(data); address += 1) {
    writeEEPROM(address, data[address]);
    if (address % 64 == 0) {
      Serial.print(".");
    }
  }
  Serial.println(" done");
  // Read and print out the contents of the EERPROM
  Serial.println("Reading EEPROM");
  printContents();
}
void shiftOutRegister(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t
val) {
```

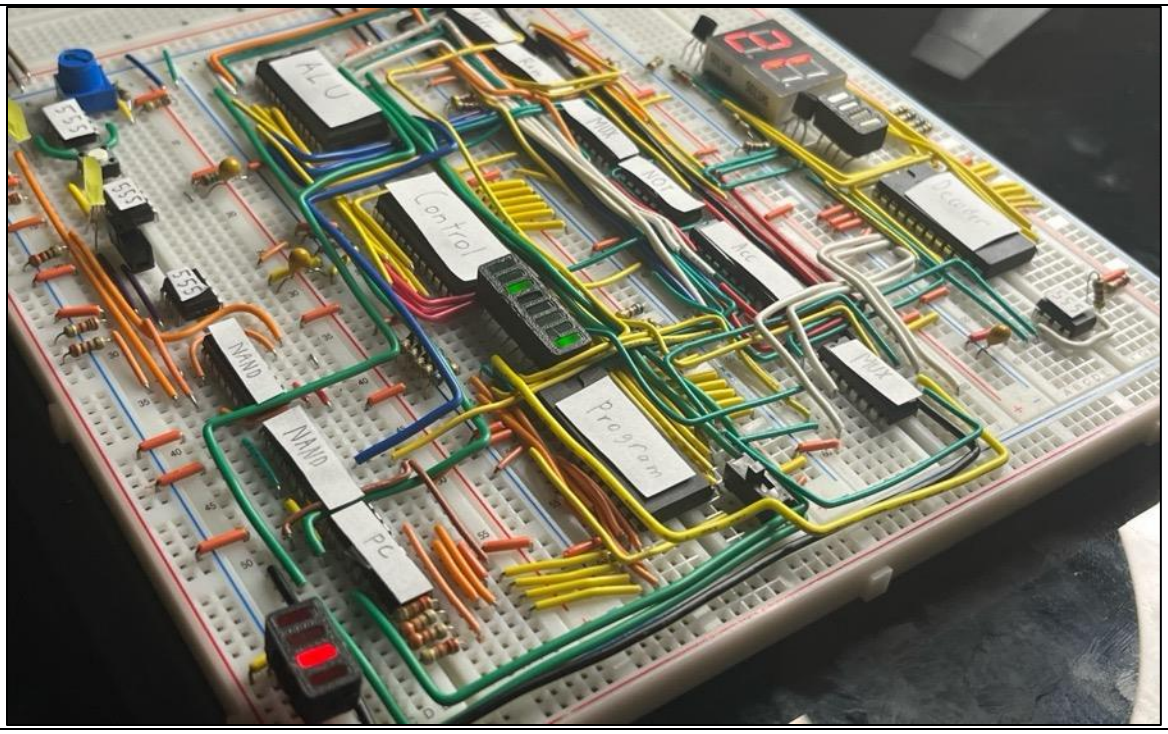
```
    DDRD |= (1 << dataPin) | (1 << clockPin);
    for (uint8_t i = 0; i < 8; i++) {
            if (val & (1 << (7 - i))) {
                PORTD |= (1 << dataPin);
            } else {
                PORTD &= ~(1 << dataPin);
            }
        }
        // Toggle clock pin
        PORTD |= (1 << clockPin);
        PORTD &= ~(1 << clockPin);
    }
}
void loop() {
}
```

Media

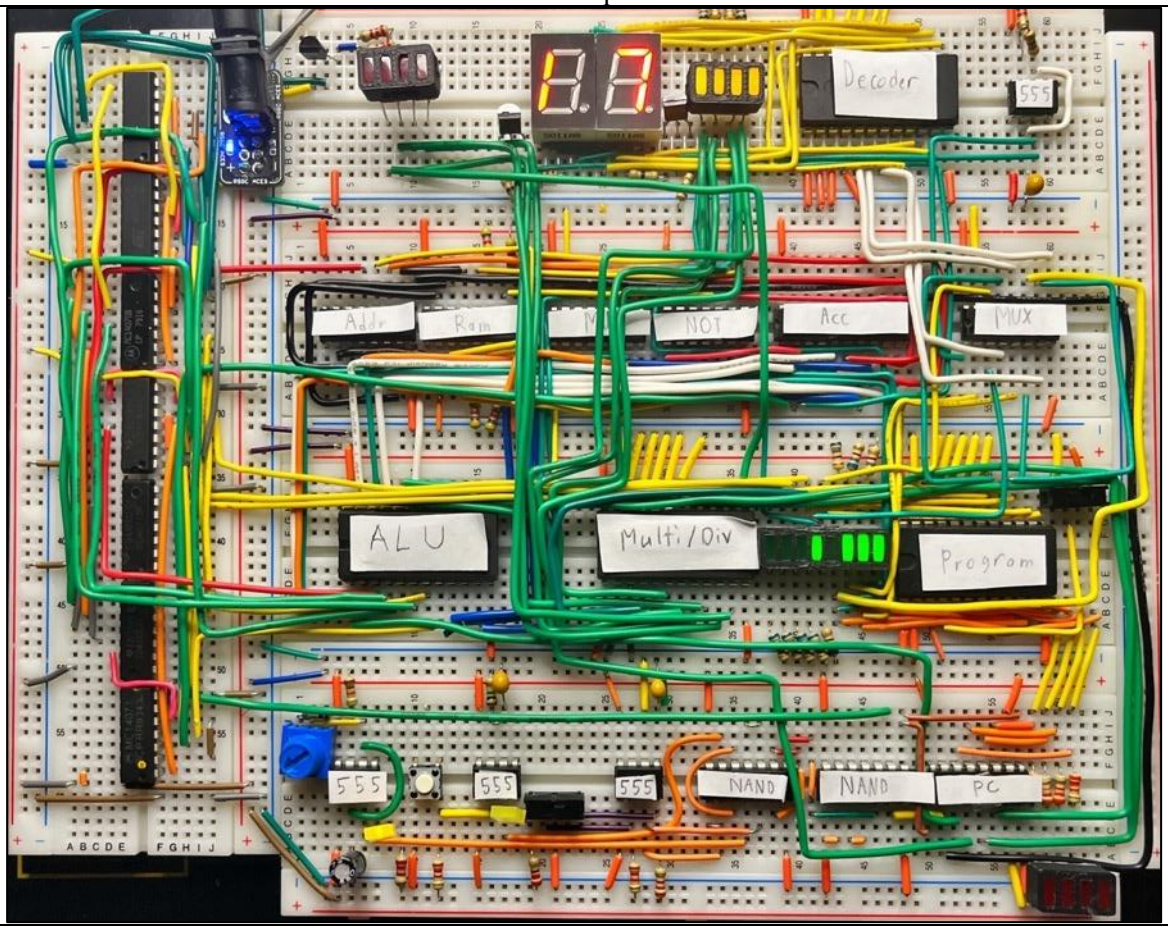
Project Video: https://youtu.be/PhskyN_O-eU



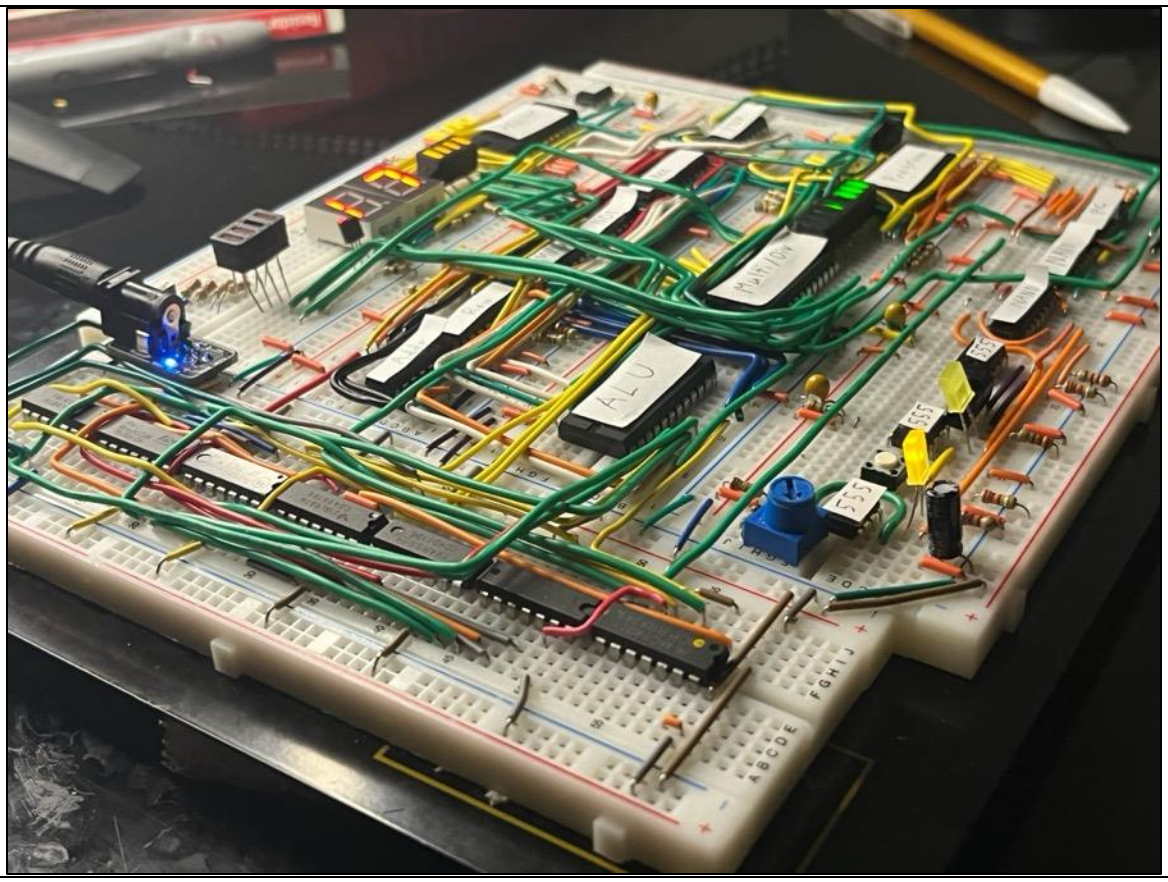
Standard Chump



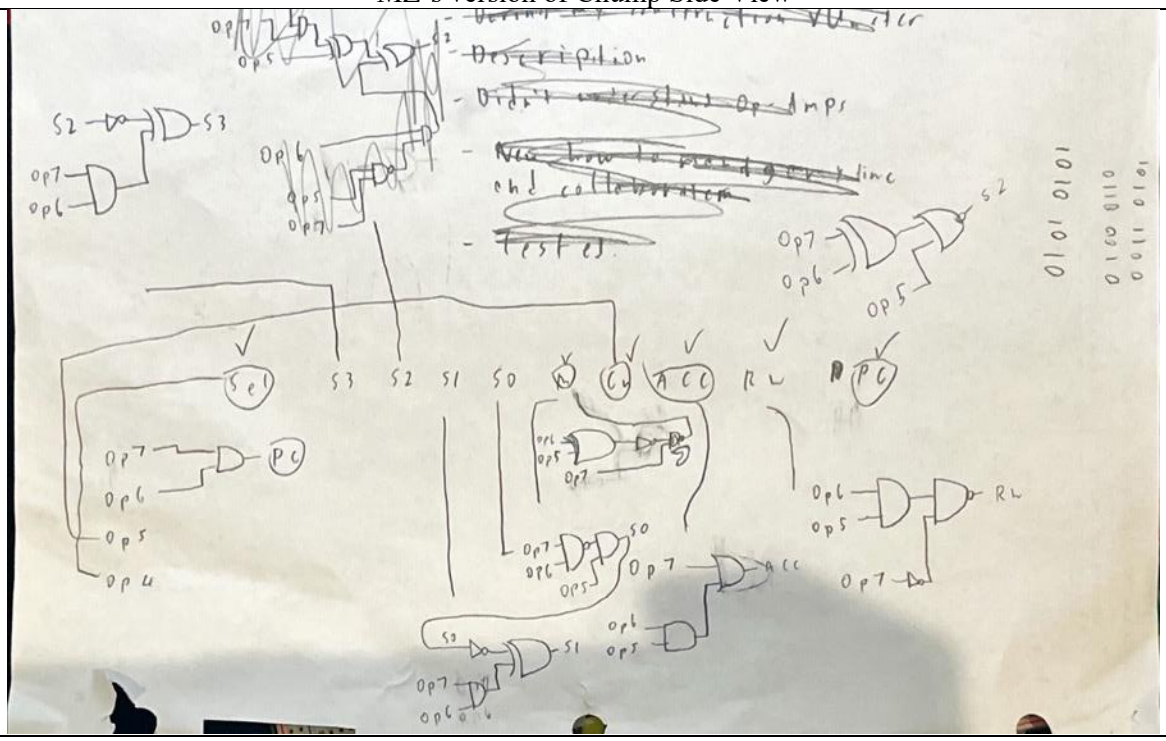
Standard Chump Side View



MZ's version of Chump

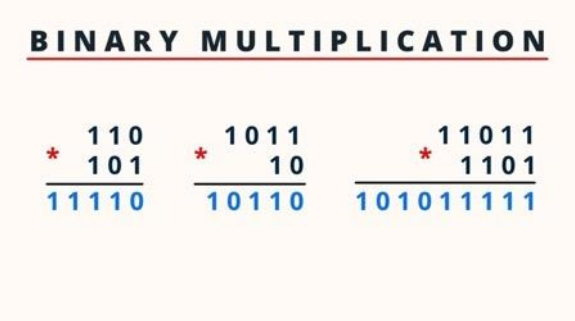


MZ's version of Chump Side View



Combinational Logic Schematic

Add-ons



For my additional parts of the CHUMP to turn it into my own project I decided to do three things to it. The first dealing with a 555, multiplexer and two seven segment displays. As tried in the program ROM DER of the CHUMP a binary to HEX seven segment display can be created. A way to do this involves taking the 4 inputs and having 7 outputs to the display. But it is possible to use one EEPROM but using persistence of vision. The way this is created is by having an EEPROM at an extremely high frequency have an output hooked up to the MUX and two transistors one NPN and one PNP this allows for the outputs of the EEPROM to display what 4 values are being fed by the MUX to only one of the two seven segment displays. The next addition was taking the control ROM and turning it into a combinational logic circuit. The approach to this involved finding patterns in the circuit that could be accommodated by the OpCode. For example, the S0 pins will only turn high for ADD function. This mean a combinational logic circuit can be built to only turn on when Op5 is 1 and all others are zero. The last add-on was to create some sort of way to include multiplication or division. The approach includes using a look-up that had mental math done on it before to know that if value A3 and A0 are high then that would be 2×1 for the inclusion of division the A4 pin on the LUT is wired to Op4 to enable the look-up table to output the mental math of division value. Note that this is integer division not exact.

Reflection

Wow. It is crazy to think that a project that spanned over three months is now over. This project truly brought out something super interesting that this course offers. What I loved about this project was how you got to understand how everything works together. By doing this having every ACES take on one IC to allow for collaboration made it an even better project. The understanding behind the IC's that allowed for the creation of a 4-bit processor is still something that mesmerises me. My personal favourite part of the project was all the combinational logic circuits that were analyzed and made. One of the things I love about hardware is going behind the curtain of electrical components you might think you know. When seeing all these circuits made of logic gates which are really two or three transistors together it makes it so much more interesting to see this whole new world. The step-by-step approach that was taken with CHUMP was definitely useful to have due to the fact that while grade 12 provides an extremely heavy course load for engineering students this project did not pose too much of a problem until the end when ambition can get in the way.

During the last week of the project, I got to work on a skill that I believe is the most important skill to have, creativity. Creativity is how anything becomes successful. The CHUMP is almost an open canvas when built that is waiting for creativity. Getting to apply the creativity of my love for math to this project along with my interest in making combinational logic circuits is another reason the CHUMP brought so much enjoyment. The workload that this project did bring involved 4 nights staying up past 3 am, about 100 phone calls to other ACES either helping or being helped and so many more great experiencing that make me love the course and love being an engineer.

Long Independent Study Project: Audio Spectrum Analyzer

Purpose

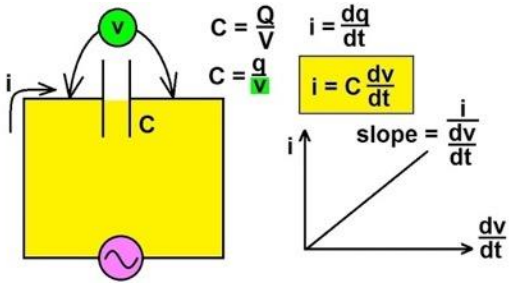
The purpose of the Audio Spectrum Analyzer is to visualize 8 bands of audio frequencies amplitudes. In doing so, an understanding of band-pass filters, capacitive reactance, complex numbers and audio amplification is created.

Reference

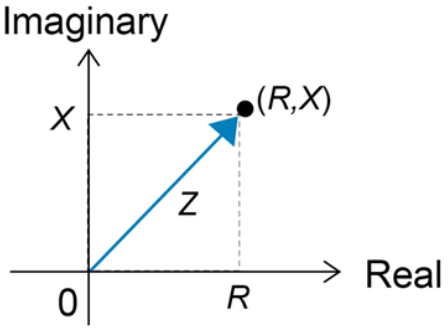
Butterworth Filter Design: https://www.electronics-tutorials.ws/filter/filter_8.html
Explanation of Capacitive Reactance Math: https://www.youtube.com/watch?v=FCNHN7B9iDM
Capacitive Reactance: https://www.electronics-tutorials.ws/filter/filter_1.html
Sallen and Key Filter Design: https://www.electronics-tutorials.ws/filter/sallen-key-filter.html
Derivation of $\frac{1}{2\pi RC}$: https://electronics.stackexchange.com/questions
Project Description: http://darcy.rsgc.on.ca/ACES/TEI4M/2425/ISPs.htmlTheory

Theory

The first part in understanding how the graphic equalizer functions is understanding how a capacitor works in an AC circuit. When the sound comes through the MAX9814 microphone it is composed of several sinusoidal waves added together. This creates an alternating current (AC) circuit. In an AC circuit when a capacitor is placed at some point in the circuit it creates a type of resistance. This however is very different from the type of resistance received by resistors. In resistors the voltage to current ratio is expressed by the equation $V = \frac{I}{R}$. Yet when working with capacitive reactance the relationship of voltage to current is $C\frac{dv}{dt} = I$. C represents the capacitance in farads which is how much voltage can be held in the capacitor. $\frac{dv}{dt}$ represents the derivate of voltage with respect to time basically measuring the instantaneous change in voltage. What this means for relation between voltage and current through a capacitor is that the greater the change in voltage at an instant the more current produced.



With this idea of how voltage relates to current through a capacitor a qualitative idea can be formed of the relationship between capacitors and frequencies. The next idea to further develop this idea is to derive the equation for capacitive reactance from the relationship between current and voltage. Firstly, the v value can be expressed as $e^{j(\omega t-\phi)}$ we e is Euler's number, j represents an imaginary number, t is time and ω is the angular velocity. This assumption can be made due to Euler's identity formula. If this is an AC waveform that is made then an assumption can be made that voltage as a function of time equal $V(t) = V_0(\omega t + \phi)$ where V_0 is the amplitude of the AC waveform, omega is the angular frequency, t is time and ϕ is the phase shift. There is also an imaginary part of this signal that is created that is $jsin(\omega t + \phi)$ now this is an imaginary part of the AC waveform but is used to help express the phase shift real cosine function has relative to current. With the equation $e^{j\theta} = \cos(\theta) + jsin(\theta)$ meaning that from the two equation one real and one imaginary voltage can be expressed elegantly as $e^{j(\omega t+\phi)}$.



$$Z = R + iX$$

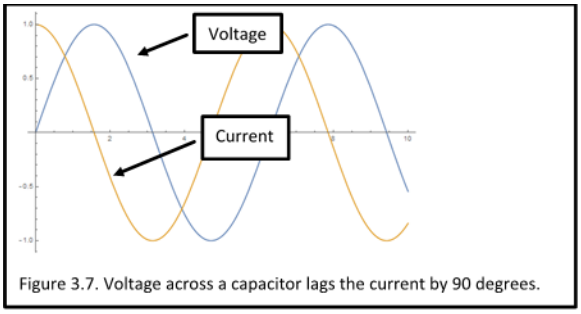
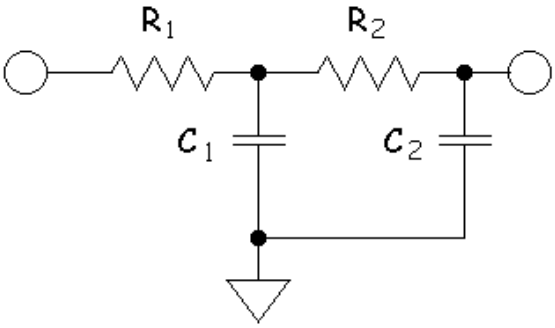
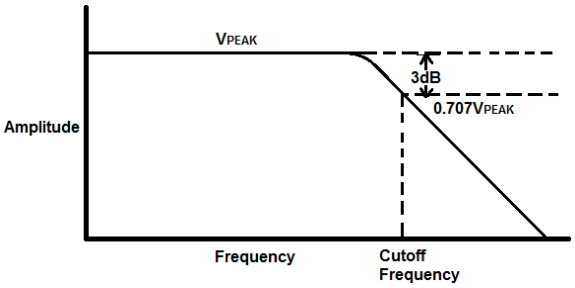


Figure 3.7. Voltage across a capacitor lags the current by 90 degrees.

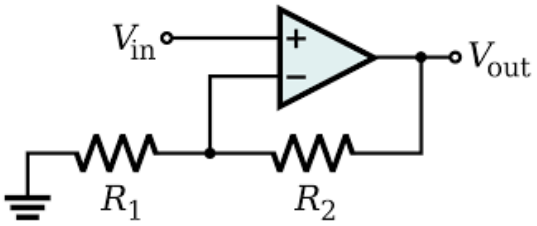
Now with the proof of how voltage relates to the $e^{j\omega t}$ when there is no phase shift the equation for capacitive reactance is now $I = C\frac{d(e^{j\omega t})}{dt}$ which when derived to equal $I = C(j\omega e^{j\omega t})$ using the chain rule which is display in the image on the right. Now from this V can be substituted back in to the equation to now have the equations be $I = Cj\omega V$ and because capacitive reactance should be a relationship of voltage over current like normal resistance. The equation can be rearranged to $\frac{V}{I} = \frac{1}{j\omega C}$. Lastly, the j can be removed when no including the phase shift occurred because capacitive reactance looks at the absolute value meaning $X_C = \frac{1}{2\pi FC}$ because $\omega = 2\pi F$. This shows the relationship between frequencies and capacitors output of voltage.



The next step in understanding how exactly the filter's function is understanding how the capacitor and resistor work together in the circuit. The image on the right shows a passive low pass that indicates what happens to a voltage signal when passing through this point. As seen, there is a voltage division between the capacitor and the resistor. The equation for any voltage division is always $V_{out} = V_{in}\frac{X_c}{R+X_c}$ where X_C is the capacitive reactance that acts like a resistor in AC circuits. Which can be further expressed as $V_{out} = V_{in}\frac{\frac{1}{j\omega C}}{R+\frac{1}{j\omega C}}$ then in the entire fraction is multiplied by $\frac{j\omega C}{j\omega C}$ then the equation simplifies to $V_{out} = V_{in}\frac{1}{1+j\omega RC}$. This provides an equation for how voltage division will occur in a resistor-capacitor circuit. The voltage division can be described as the Laplace Domain.

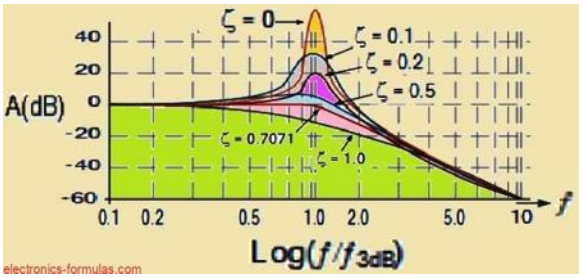


Now the next and last concept that needs to be understood to derive the equation required is to understand how gain works in these circuits. When a low-pass or high pass filter is being attenuated although in an idea filter the signal would turn completely off normal filter will continuously be attenuated more based on how displaced it is from the cutoff frequency. The attenuation of an AC waveform at the cutoff frequency, f_c, is -3dB which is which when unexpressed from a logarithmic scale it becomes $\frac{1}{\sqrt{2}}$. This means that the magnitude of $\frac{V_{out}}{V_{in}} = \frac{1}{\sqrt{2}}$ because of how the attenuation works. Which means the magnitude of the voltage division described above is equal to the attenuation meaning $\frac{1}{\sqrt{2}} = \left|\frac{1}{1+j\omega C}\right|$ where is the Laplace domain. The magnitude is $\sqrt{1+(\omega RC)^2}$ which equals the attenuation meaning $\frac{(\omega RC)^2}{1+(\omega RC)^2} = \frac{1}{2}$. Then after clearing the denominator the equation, it becomes $2(\omega RC)^2 = 1 + (\omega RC)^2$ and based on this $\omega RC = 1$ and $\omega = 2\pi F$ making the equation $f = \frac{1}{2\pi RC}$ where f is the frequency when attenuation begins.



Now deriving that equation above helps to understand how a passive filter can take in frequencies. Yet the attenuation that a passive filter provides is not useful enough to be able to track the differences between an attenuated frequency and a non-attenuated. As a result of this a operational amplifier can be placed in the circuit to add a greater amplification to non-attenuated creating a larger difference that is able to be filtered out. An Op-Amp when having a resistor in feedback acts as something that amplifies a signal. The ratio of amplification is $1 + \frac{R_2}{R_1}$ with respect to the image above. This is due to the fact that $V_{IN} = V_{OUT}\frac{R_2}{R_2+R_1}$ due to the rule of voltage division.

A specific type of filter design is known as the Butterworth filter. This chosen feedback resistors for this filter design include a R_1 and $R_1(\alpha - 1)$ where $\alpha = \sqrt{2}$. The reasoning behind that starts with the derived equation from a Laplace Domain that is $H(s) = \frac{\omega^2}{s^2+\sqrt{2}\omega s+\omega^2}$ which follows the same idea as current and voltage when damping. Therefore, it came be expressed on a complex plain as,



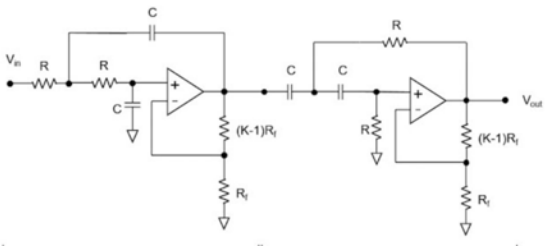
$$s = \omega(\cos\left(\frac{\pi}{4}\right)) \pm j(\sin\left(\frac{\pi}{4}\right))$$

The real part of this equation can be expressed as $\omega(\cos\left(\frac{\pi}{4}\right))$ which is representing the dampening factor therefore can also be expressed as $\zeta\omega = \frac{\omega}{\sqrt{2}}$ which when canceling out the angular frequencies leaves makes the Butterworth filter dampening at $\frac{1}{\sqrt{2}}$.

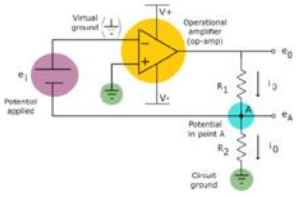
Procedure

The first part in building the Audio Spectrum Analyzer is building the filters. From the derived equation above being $f_c = \frac{1}{2\pi RC}$ the independent variable being the cut-off frequencies chosen, the dependent being the resistor values and the fixed variable being the capacitor values. The best-chosen capacitor value to fit the sound frequencies from 1Hz to 8kHz is a 100nF capacitor. The resistor is chosen as the dependent variable due to the ability to have very irregular resistor values. The function is to create 8 bandpass filter which involves 16 filters. Using these three variables. One additional note is the Op-Amps function. When an Op-Amp is in a close circuit it follows a rule that the non-inverting pin is always trying to equal the inverting pin and vice versa. This is due to the equation $V_{out} = A(V_1 - V_2)$ where the A value in an ideal Op-Amp is infinite which will reach the V positive rail on the Op-Amp when $V_1 > V_2$. This allows the Op-Amp to act as a comparator and output digital value of either the V positive rail or V negative rail based on the two pins compared.

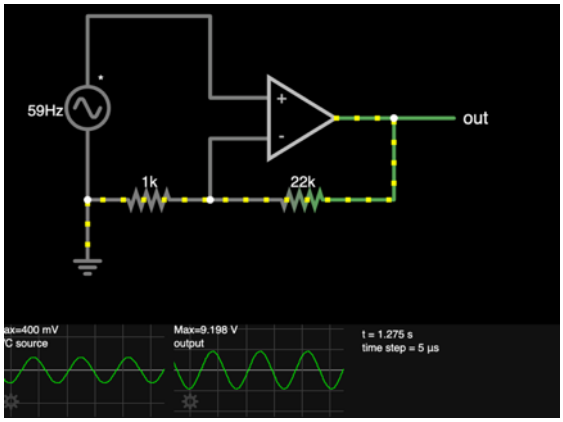
Parts Table	
Quantity	Description
1	MAX9814 Microphone
6	LM324N Quad Op-Amps
1	LM358 Dual Op-Amp
2	CD4027 Dual Flip-Flop
1	LS7408N (AND)
2	74HC4051(MUX)
1	NE555 Timer
3	8 1 kΩ Resistor Network
8	4 100nF Capacitor Array



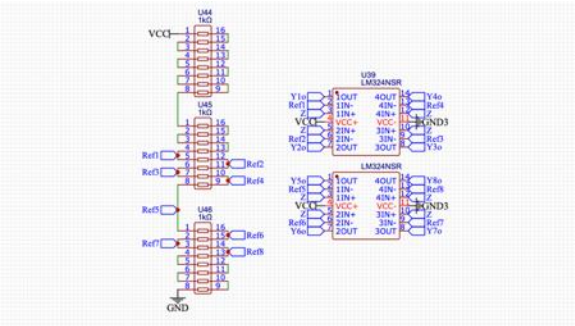
The filter design chosen above is a 2nd order Butterworth filter. Meaning each of the 16 filters must include two variable resistors based on the cut-off frequencies, two fixed resistors based on the equation above to achieve a 2nd order filter, two 100nF capacitors and a single Op-Amp. The idea behind the exact design is that the first filter is the equivalent to a passive high pass and low pass filter simply dealing with a capacitor and resistor flowing to ground. The capacitor charges and discharges in an AC circuit and the resistor decides the speed of the charges and discharge. If the charging and discharges takes too low based on the length of the soundwave or the RC circuit it will be attenuated. The 2nd order factor having a resistor and capacitor in feedback acts to attenuate the frequency further as is it fed back into the same RC configuration. Lastly, two resistors are placed between the output terminal and the inverting pin as well as the inverting pin and ground. These resistors allow for the flattened attenuation that the Butterworth filter provides.



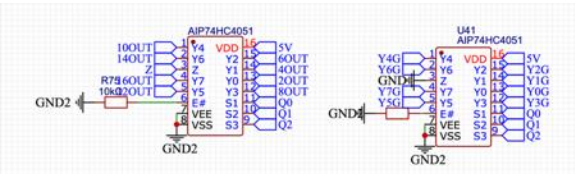
One important note to make about these filters that are not mentioned in the schematics are the need for the pull-down resistors in the Op-Amps non-inverting pin. This due to two reasons. The first reason deals with a reference ground. An input to an Op-Amp in an AC circuit configuration needs a ground reference or else noise can affect the circuit and cause the axis for the AC circuit to not be located at the ground. The other reason deals with current. In an Op-Amp one of the key rules is that an Op-Amp never takes in any of the voltage meaning it is not providing a point for the soundwave's voltage to flow. Therefore, if ground this supplied to the soundwave, then the Op-Amp can observe the voltage level at an axis of 0V. The other note is the chosen bands for the frequencies include 200Hz to 500Hz, 500Hz to 900Hz, 900 to 1400Hz, 1400Hz to 2000Hz, 2000Hz to 3700Hz, 3700Hz to 5500Hz, 5500Hz to 7400Hz and 7400Hz to 9300Hz. The decision of 200Hz being the lowest only has to do with the MAX9814 not being able to pick up low frequencies unless at very high amplitudes.



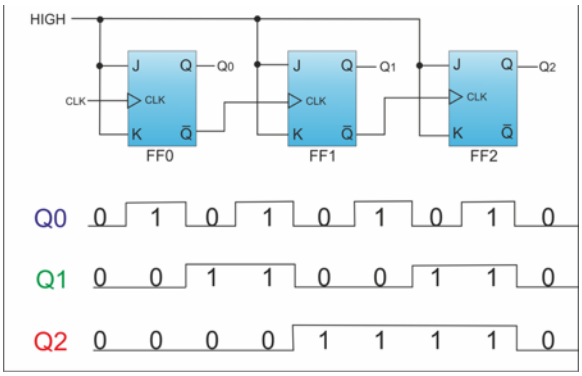
After all eight bandpass filters are created the next task is creating a way to display all the filters. The first step is creating a way to define amplitudes easily. The step to do this is to take the soundwave input and amplify it. This requires to steps. The first is to remove the DC bias the MAX9814 provides to the soundwave. DC bias occurs to allow this circuit to be created using a single power supply yet when adding an amplifier creates problems. The DC bias on the MAX9814 is at 1.5V meaning if an amplifier multiplied the voltage level by 20 the centered bias would also move by that amount multiplied. Yet if the bias is removed then the signal will remain at an axis of 0V and only amplify the peaks and troughs. The amplification to the soundwave included a 22 kΩ resistor between output and the negative terminal and a 1 kΩ resistor between the non-inverting pint and ground.



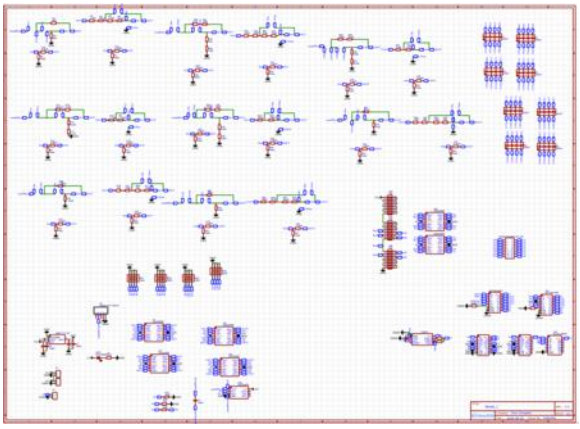
Now from the 9V signal outputted from the LM358 it is inputted to the 8 bandpass filters and the output is either the 9V wave if fitting in the required bandpass or attenuating the signal. The first step needed in driving this display is using Op-Amp comparators with specific reference voltages. For this step 24 1 kΩ resistors are required to allow for a 0.5V difference between all Op-Amp comparators. The comparators centered from 2V to 6V allowing for 8 different amplitudes to be recorded. This value causes a problem and is displaying attenuated values which it should not be. A better design would have references from 4V to 8V. All outputs from the comparators are sent directly to the anodes of the LEDs.



The comparators are able to drive the signals directly to the display due to two multiplexers functioning together. The two multiplexers are acting as a multiplexer and a demultiplexer. One is taking the signals outputted from the bandpass filters and inputting to the non-inverting pin on the comparators, demultiplexing, while the other is multiplexing a ground signal to one of the 8 common cathodes. What makes this work is the synchronized clock between the two ICs. This synchronizing means that each column will display frequency band and allows for persistence of vision to occur.



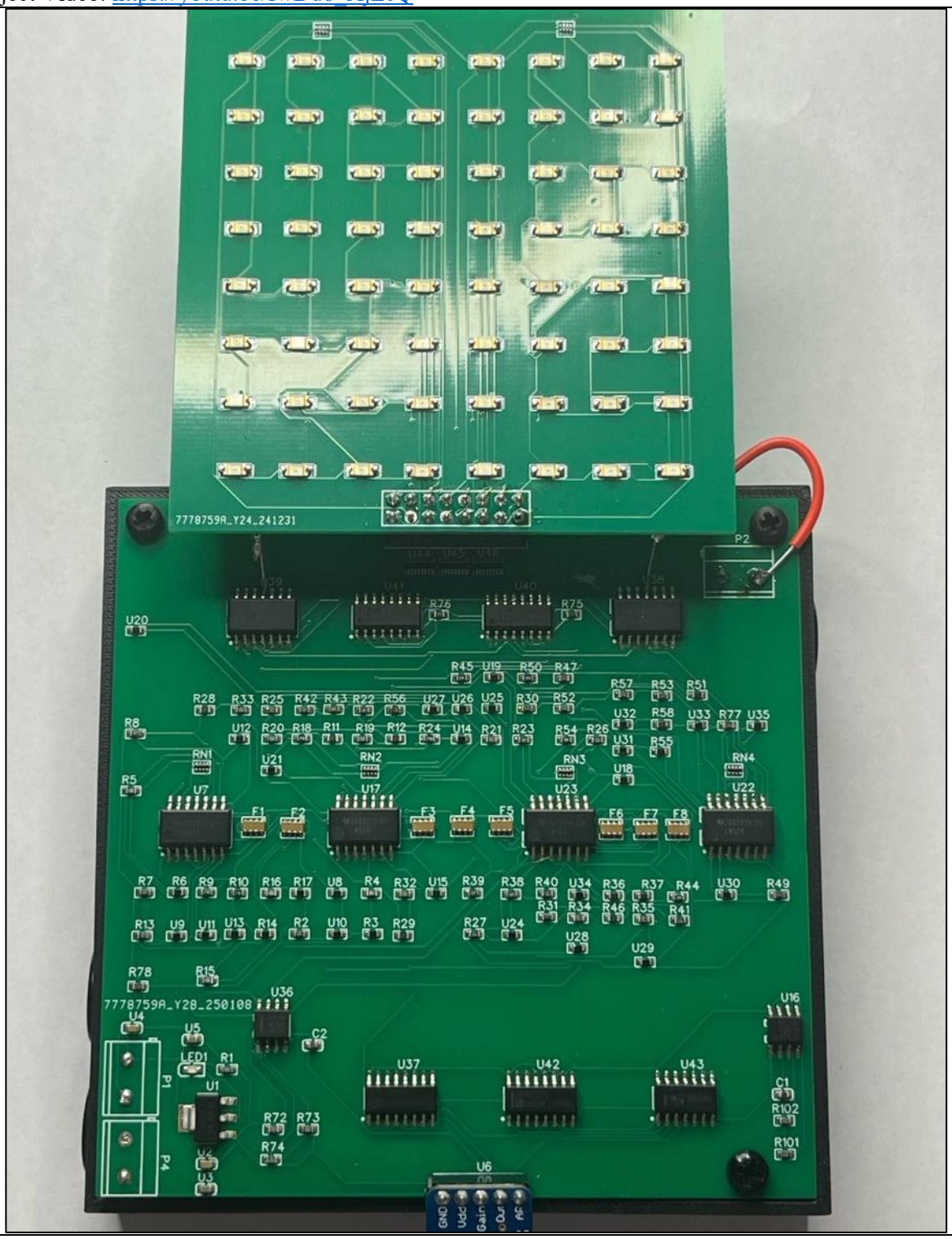
The clock created is a 3-bit binary counter. This counter can be created from JK Flip-Flops, a logical AND gate and a clock pulse. The clock pulse is driven by a 555 timer IC. For a refreshment this IC can create a square wave pulse based on the ratio of two resistor and a capacitor in an integrated circuit. For the clock pulse in this circuit the goal is to create persistence of vision there for the ratio between the two resistors a 1 to 1 ratio with a 1 uF capacitor. This creates an extremely fast clock pulse. This clock pulse is then hooked up to the clock pin of 3 JK Flip Flops. Then wiring the J and K values all to high creates frequency divider that can be used as a 3-bit counter for both of the multiplexers.



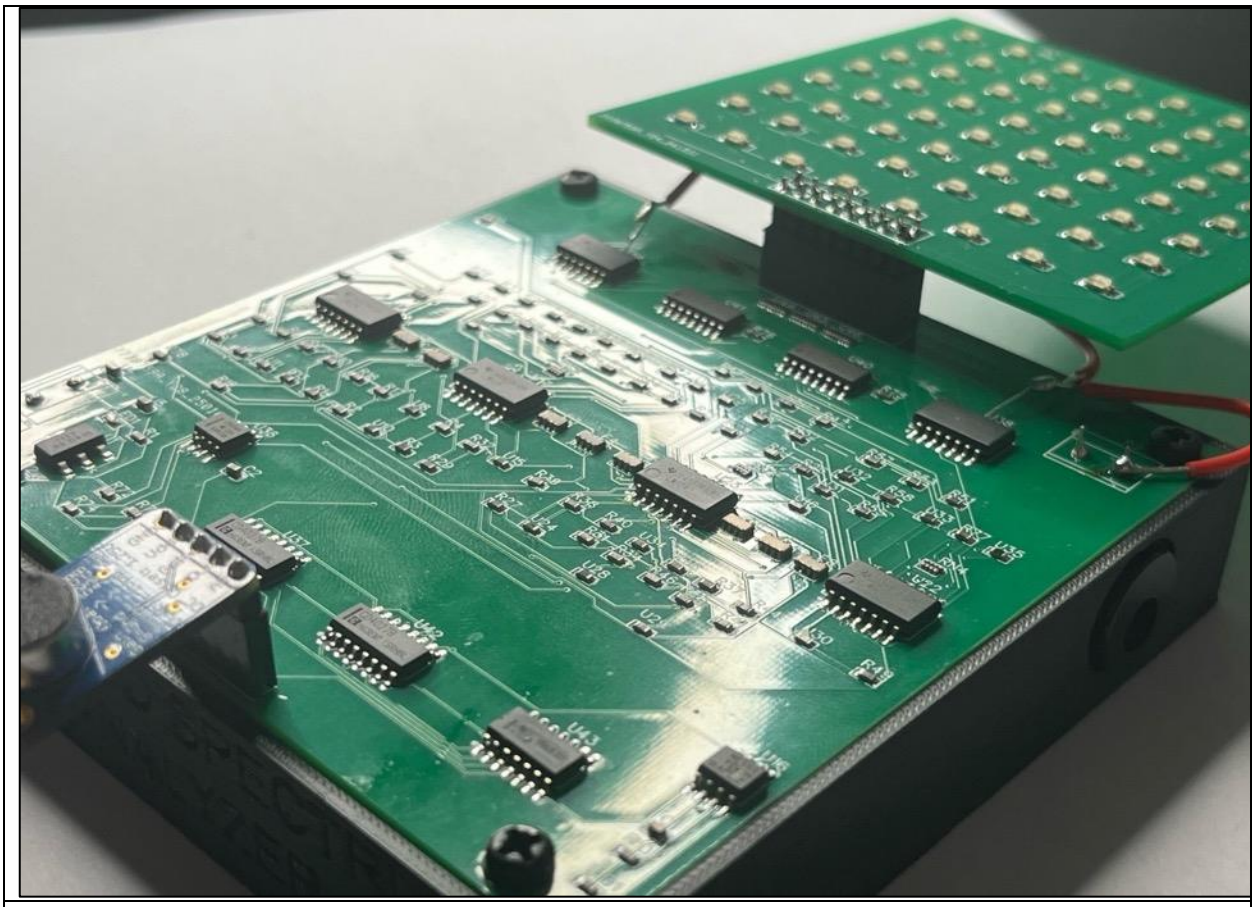
When all components work together the soundwave input is put through a capacitor. Then amplified before connected to all 8 bandpass filters. The output from the bandpass filters is then sent to 8 pins on a multiplexer that choses from one of the 8 pins based on the three-bit counter created from a JK Flip Flop. The output from the multiplexer is brought to an Op-Amp comparators non-inverting pin while it's inverting pin is hooked up to a voltage divider. The output from the Op-Amp is going directly to an individual ground pin as the clocks are synchronized. This creates the 8-bit Audio Spectrum Analyzer.

Media

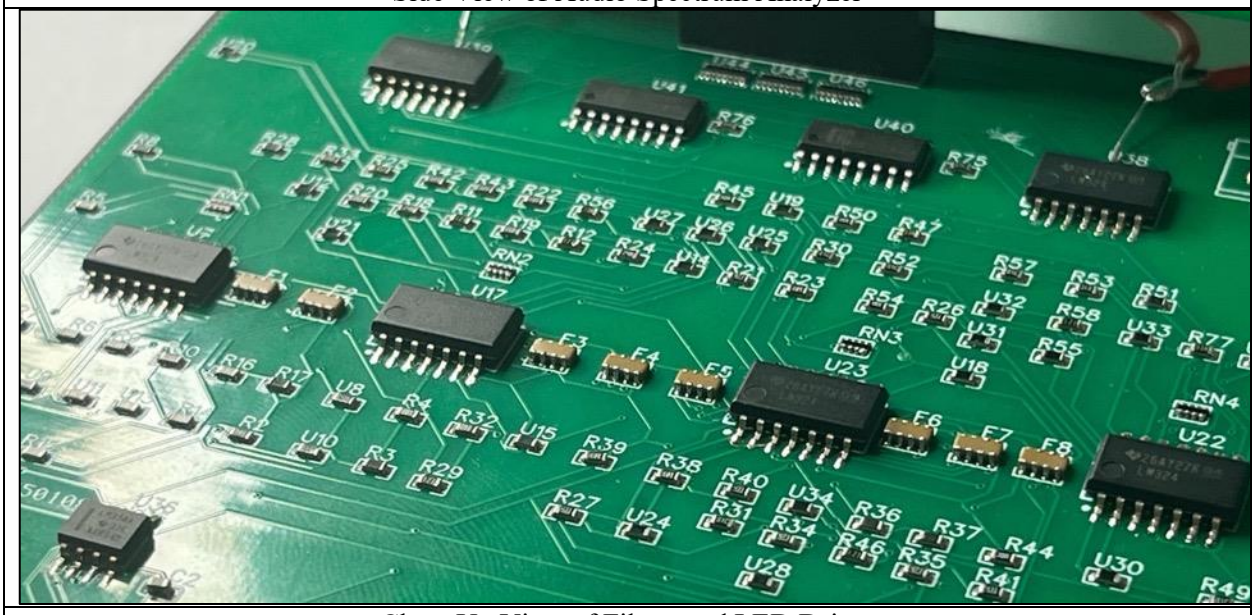
Project Video: https://youtu.be/fwDd6_AjLcQ



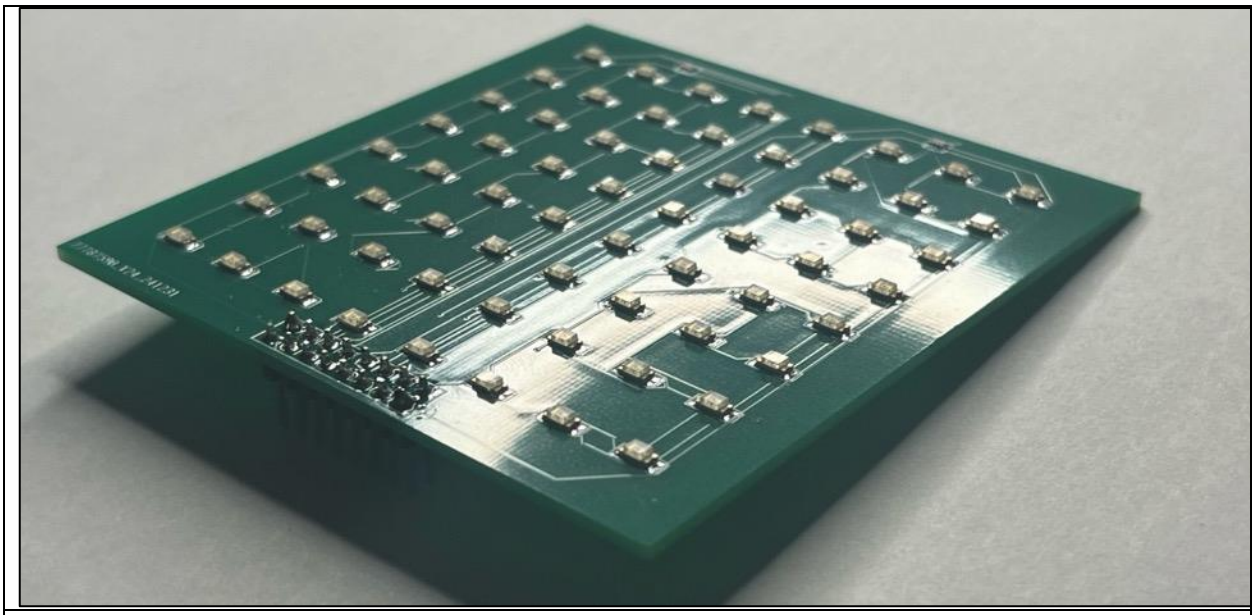
Over Top View of Audio Spectrum Analyzer



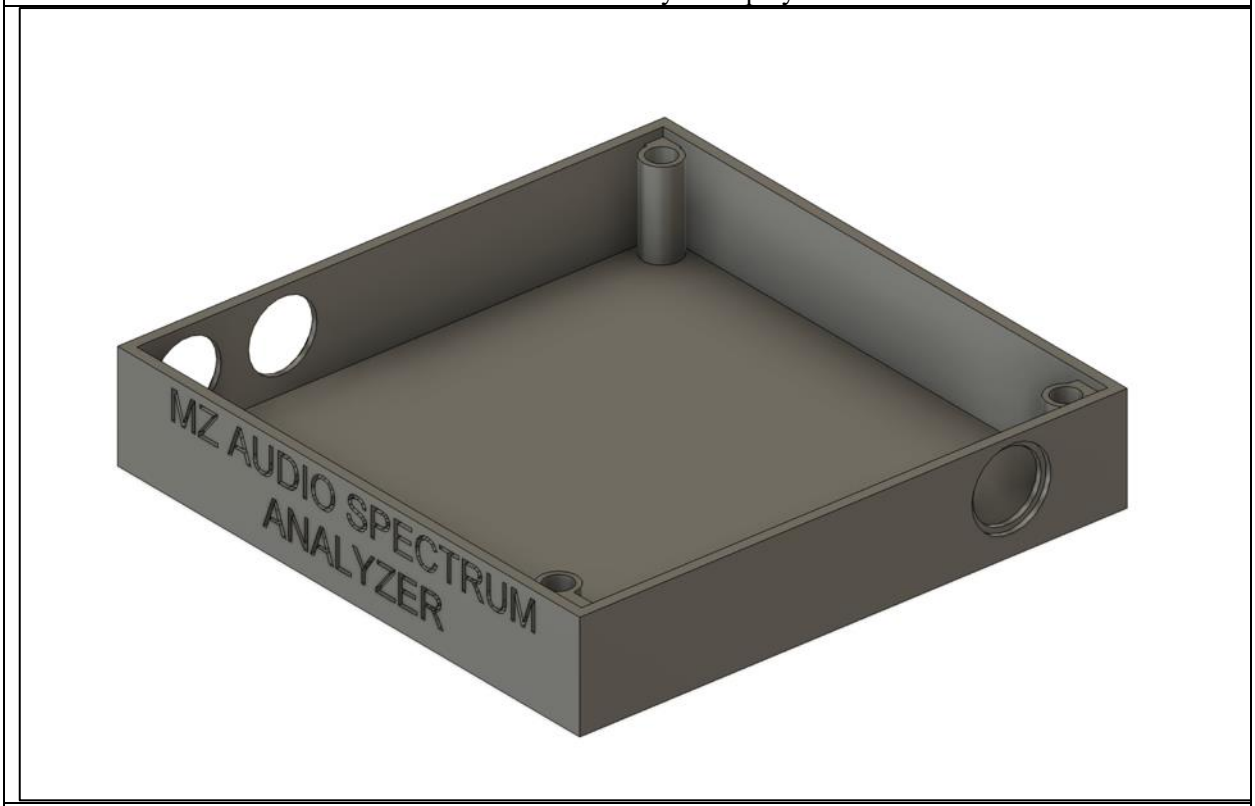
Side View of Audio Spectrum Analyzer



Close-Up View of Filters and LED Driver



Side-View of 8 by 8 Display



Encasement of Audio Spectrum Analyzer

Reflection

If there is one word, I would use to describe this three-month project it is impact. I know more now than I ever have that I love electrical engineering. A project like this was not easy and required a lot of work to complete and slightly understand what was going on and even though the display failed to work perfectly I am still proud of the output and research I put. Every minute felt more like enjoyment than work.

3.3 AVR Assembly: Traffic Light

Purpose

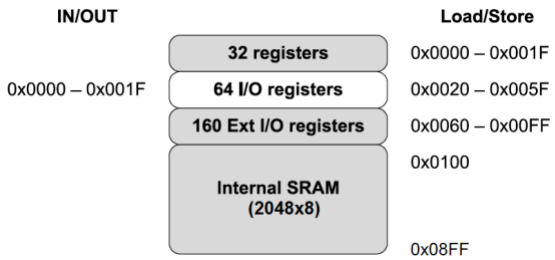
The purpose of the AVR Assembly: Traffic Light is to simulate an actual traffic lights cycle using the lowest level of software.

Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEI4M/Tasks.html#TrafficLight
Assembly Instruction Set: http://darcy.rsgc.on.ca/ACES/TEI4M/Assembly/InstructionSetSummary.pdf

Theory

The Atmega328P memory architecture is composed of several memory registers including; SRAM that holding things like temporary function calls, local variables and function parameters. The additional part of the I/O registers are the external registers split up into three parts. The most important being the first 32 registers of general-purpose working registers. These occupy 0x0000 to 0x001F of register space. The first 26 registers in this space are 8-bit registers that are used later on. The last 6 registers make up the X, Y and Z registers which are capable of being 16-bit registers.

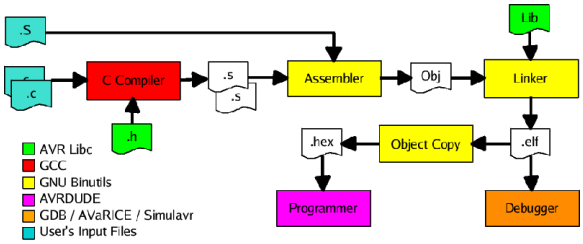


Procedure

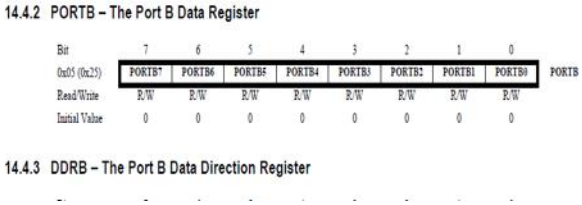
The only two parts making up this project are the traffic light PCB composed of 4 components. 3 10mm LEDs to represent the traffic lights and a current limiting resistor. The Arduino Nano is the other component and the driver behind the traffic light PCB. All the code is written in assembly language in a .S file. Made in the Arduino compiler. An assembly compiler has the task of taking all instructions and mapping to the correct ports on the piece of hardware. The ATTiny85 is used to symbolize how this code can be used on AVR board with only slight adjustments.

Parts Table	
Quantity	Description
1	Arduino Nano
1	ATTiny85
1	Traffic Light PCB

The reasoning behind the .S file in a deeper sense is symbolized in this image right here. When .S is selected is selected it tells the Arduino complier to allow for preprocessing. Preprocessing tells the computer that even it does not need to go through the same C compiling normal Arduino programs go through but instead goes straight to the assembler part which is usually translated into assembly language code but in this case does not need to be.



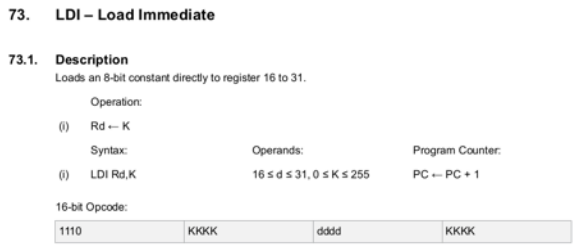
The assembly code here can be broken down into three parts: the global, the loop and setup, and the functions. Firstly, the setup is only made of four lines of code including `#include <avr/io.h>` this allows for the predefines like `DDR` and others to be turned into numbers while going through the compiler above. Next are the predefines required for this code to work on the ATMega328p. When done on the ATTiny85 the DDRB is not set relative to the memory mapping because there is only one port.



The next portion of the code is the function portion. This is spoken about previous to the loop code as it allows for the loop code to be better understood. The functions called in this code include `delay1s` and `rotate lights` because rotate lights are never ending part of the code it is included in the loop portion of this explanation. The only function called for efficiency reasons is the `delay1s` function being a very complex function. The goal of this function is to use up 16,000,000 clock cycles in an assembly way. This includes finding an amount of values that will in total when decremented by the ATMega take 16,000,000 clocks to do so and allow for each LED to be admired for 1 second. In this situation the values 0, 43 and 82 are used with the first value being zero in the `L1` function. It will subtract 1 from 0 which provides it 2's complement which is 254. When the value finally reaches zero the branch if not equal like is flagged and returns to the next part of the "stack".

```
//burns exactly 16MHz of clock cycles
delay1s:
    ldi r18, 82 //outer value
    ldi r19, 43 //second most inner value goes through the cycle 2 times
    ldi r20, 0  //inner value goes through 3 cycle
L1:
    dec r20
    brne L1
    dec r19
    brne L1
    dec r18
    brne L1
    ret
```

Finally, the main part of the code is `global` and `rotateLights` portion. The global loads specific values into r16 the external register used to make this possible. First is loads a 7 into the register before configuring that as a output for the data direction register with DDR as the port and r16 as the register. Next r16 is set to all zero to make sure all pins used on PORTB are low. Next once all LEDs are low the first pin can be set high by first loading the constant 1 into the register which will be put into PORTB once in the next function begins.



73. LDI – Load Immediate

73.1. Description

Loads an 8-bit constant directly to register 16 to 31.

	Operation:		
(i)	Rd ← K		
	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

The `rotateLights` function first takes the r16 value presented, in this case the first pin, and sets it to PORTB which is setting pin high. Next the function calls the `delay1s` explained before to view what value is one. It then shifts the r16 value to the left which turns the 0x01 value to 0x02. Then checks if the r16 value is at 0x08 because then it has exceeded the three output pins and needs to be rest. If it does not equal it starts again and will output a high on the next shifted pin creating the traffic light.

```
loop:
    ldi r16, 0x01       //loads one onto pin 8
rotate_lights:
    out PORT, r16       //sets pin 8 high
    rcall delay1s       //call delay function
    lsl r16             //shifts left to 0x02
    cpi r16, 0x08       //compares r16 with 0x08
    brne rotate_lights  //branch to top if r16 is not 0x08
    ldi r16, 0x01       //if r16 does equal 0x08 then set it back to 0x01
    rjmp rotate_lights  //start over
```

Code

```
// PROJECT  :Traffic Light (TrafficLight.S)
// PURPOSE  : Make the traffic light PCB alternate between 3 LEDs in Assembly
Language
// COURSE   :ICS4U-E
// AUTHOR   :M. Zischksa
// DATE     :2025 01 14
// MCU      :328P
// STATUS   :Working

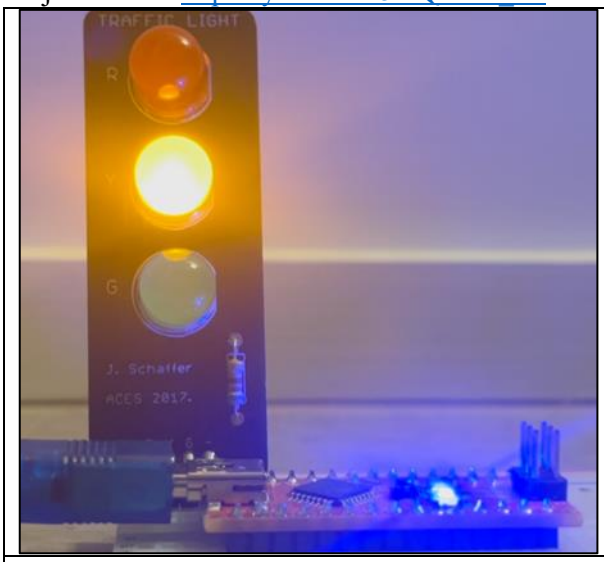
#include <avr/io.h>
#define     DDR    DDRB-0x20    //subtracts bit 6 to work with the memory mapping of
AVR
#define     PORT   PORTB-0x20   //subtracts bit 6 from the PORTB as it is relative
addressing
.global setup
setup:
    ldi r16, 0x07     //loads all ones into the last 3 bits
    out DDR, r16      //sets DDRB0, DDRB1, DDRB2 to output
```

```
    ldi r16, 0x00     //loads a 0 into the same register
    out PORT, r16     //sets all values in portB low
    ret
.global loop
loop:
    ldi r16, 0x01       //loads one onto pin 8
rotateLights:
    out PORT, r16       //sets pin 8 high
    rcall delay1s       //call delay function
    lsl r16             //shifts left
    cpi r16, 0x08       //compares r16 with 0x08
    brne rotateLights  //branch to top if r16 is not 0x08
    ldi r16, 0x01       //if r16 does equal 0x08 then set it back to 0x01
    rjmp rotateLights  //start over

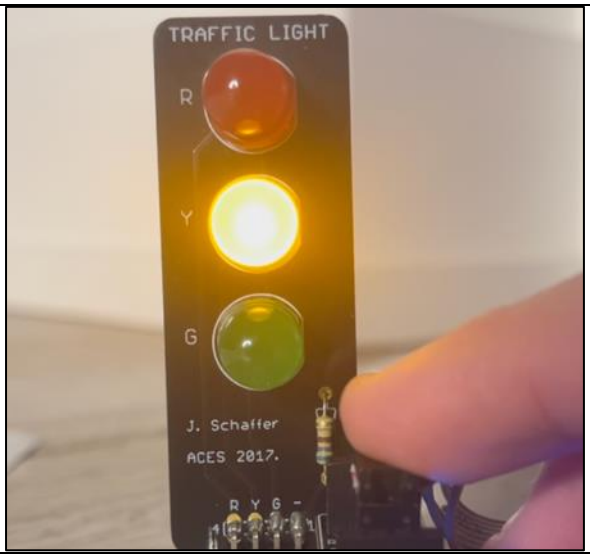
//burns exactly 16MHz of clock cycles
delay1s:
    ldi r18, 82 //outer value
    ldi r19, 43 //second most inner value goes through the cycle 2 times
    ldi r20, 0  //inner value goes through 3 cycle
L1:
    dec r20
    brne L1
    dec r19
    brne L1
    dec r18
    brne L1
    ret
```

Media

Project Video: https://youtu.be/l5f2QMT4_DI



ATMega328P Displaying Yellow



ATTiny85 Displaying Yellow

Reflection

The AVR traffic light was an interesting short project. My favourite part of it was the collaboration in learning a new language. The processes I heard from Nate, Rohan, Goran and Atticus helped me form my code and, on the last day, implement it to an ATtiny85. It was a great project.

Successive Approximation Register Analog-to-Digital-Converter

SAR ADC: Overview and Clock

Purpose

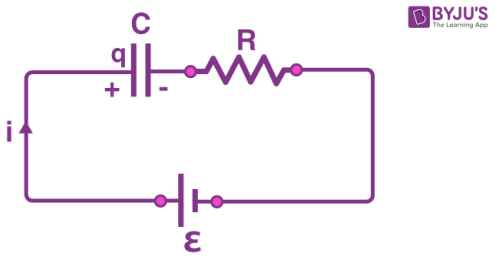
The purpose of the SAR ADC: Overview and Clock is to create the clock that will give a rising edge to the initial D Flip-flop and run the entire circuit as well as getting an overview of what makes the SAR ADC.

Reference

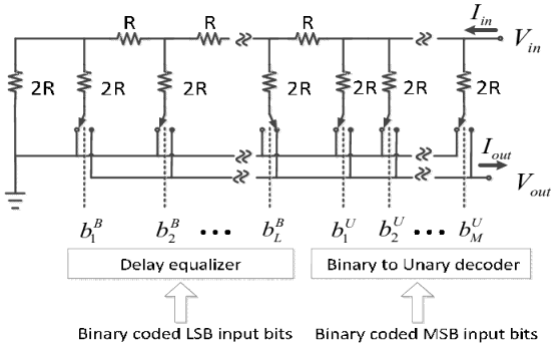
Project Reference: http://darcy.rsgc.on.ca/ACES/TEI4M/SARADC/index.html#task
SAR-ADC Description: https://hackaday.io/project/181826-homemade-successive-approximation-register-adc
Clock Simulation: https://www.falstad.com/circuit/e-inv-osc.html
R2R Ladder: https://www.youtube.com/watch?v=AulX1OM7RwE
Time Constant Description https://www.electronics-tutorials.ws/rc/time-constant.html

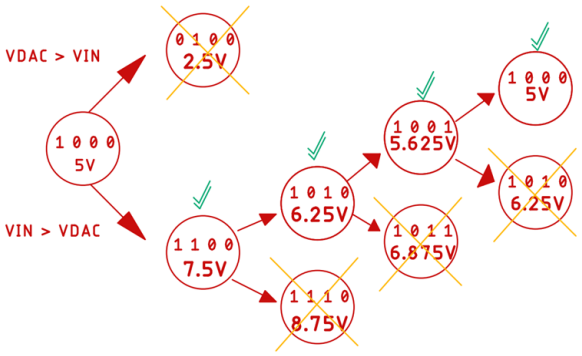
Theory

The theory behind the SAR ADC's clock begins with understanding how RC circuit's function. A capacitor can be mathematically explained with the equation Q = CV where C is the capacitance which is a constant for how an electric field can form over medium, and Q and V are proportional values. A resistor follows the standard ohm's law equation. From an understanding that a resistor controls the flow of current and a capacitor will hold an electric field based on the voltage. A relationship between a capacitors discharge when controlled by a resistor can be created. This is the time constant tau which equals the resistance in ohms (R) and the capacitance in Farad (F) making the equation $\tau = RC$. The role this time constant plays is when plugged into an exponential equation five times the time constant is the amount of time it takes for the capacitor to discharge. This plays an important role in the topology of the clock.



Looking at the overview there is a theoretical explanation behind a few of the other components in the SAR ADC. Firstly, the Op-Amp used in this circuit follows the mathematical representation from an ideal Op-Amp along the idea that $A(V_+ - V_-)$ equals the output where V is ideally an infinite number. Meaning the Op-Amp acting as a comparator will either shoot the voltage down to the minimum voltage if the inverting pin is greater or shot the voltage to the maximum if the non-inverting is greater. There is also an R2R ladder that is placed in this circuit. For refreshment, an R2R ladder functions by having a chain of resistors of resistors with the value and R and other resistors with value 2R. Based on if the switch at each logic point is turned on your decides if that much voltage is added. But as the switch have less resistance to the output the weight of the switch becomes great and follows the equation $V_{out} = V_{ref} \times \frac{DI}{2^n}$ where DI is the digital input switch and its distance from the output point and n in the number of bits.



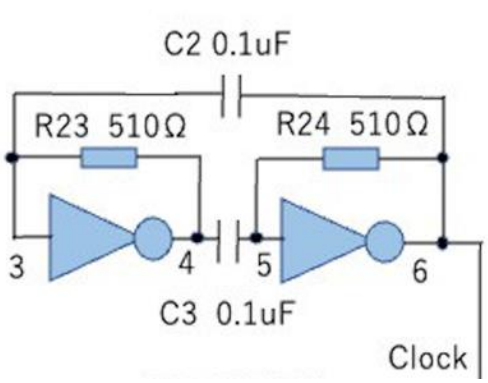


The last conceptual concept to be understood is successive approximation. The successive approximation register is how an ADC is able to be made and it involves a DAC, Op-Amp comparator and the approximator using a binary tree. The way the register works is by taking following a binary tree of digital voltage ranges on a analog signal to pin point what digital voltage is present. The binary tree on the right shows how this function. Through a process created by the construction of the SAR ADC the register will take in a voltage and run sample voltages through a comparator. The diagram is showing a 4-bit binary tree. The SAR has DAC with 10V supply that starts at exactly half of the voltage power. Then after starting from half the voltage, it runs two comparing's with half the voltage above and half the voltage below. In this case the comparator turn on for the 7.5V value from the DAC and the first bit of the DAC is the a one. This process continues to compare to approximate the voltage.

Procedure

The clock portion of the SAR ADC is constructed of basic components being an RC configuration and two NOT gates. The circuit functions by tying the output of one NOT gate to the input of the other. The speed that the gate reads the change in state in one of the not gates is based on the charging and discharging of the capacitors that are hooked up between the two components. The ADC is made up of additional components that will be added to the parts table as the project goes on. This includes an additional 10 components including shift registers to output values, Logic Gates and Op-Amp comparators. The power supplies required from this circuit include a +9/-9V for the accommodation of AC signals and a +5V supply for the ICs.

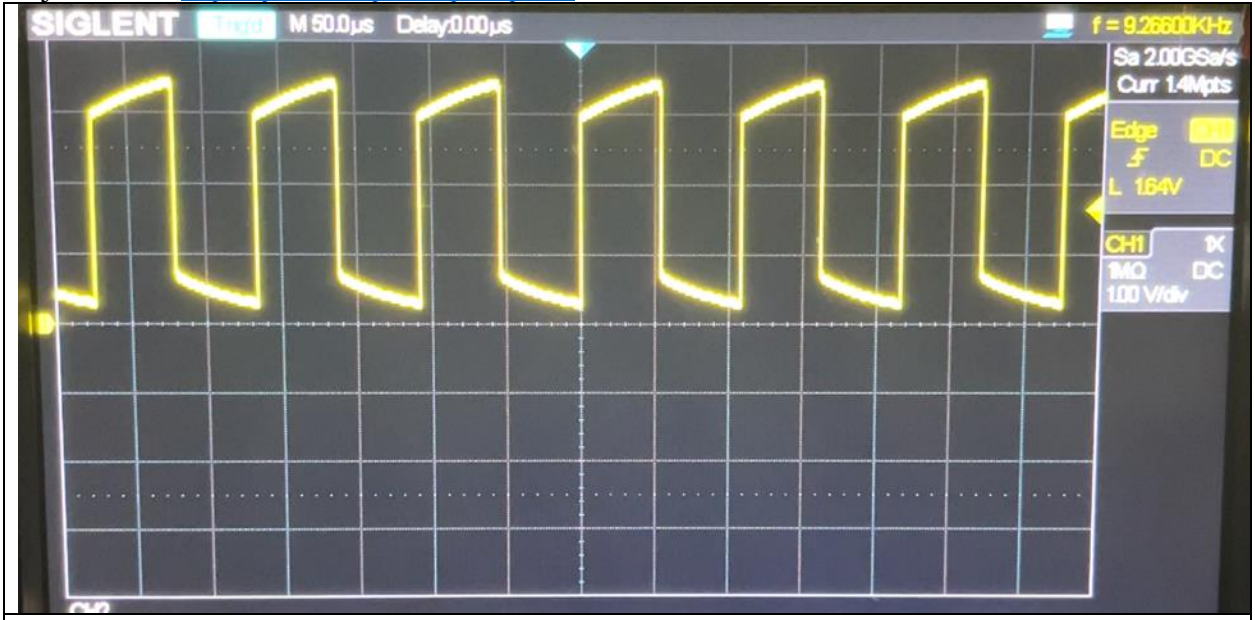
Parts Table	
Quantity	Description
1	SN74HC04N (NOT)
2	510 Ω Resistor
2	100 nF Capacitor



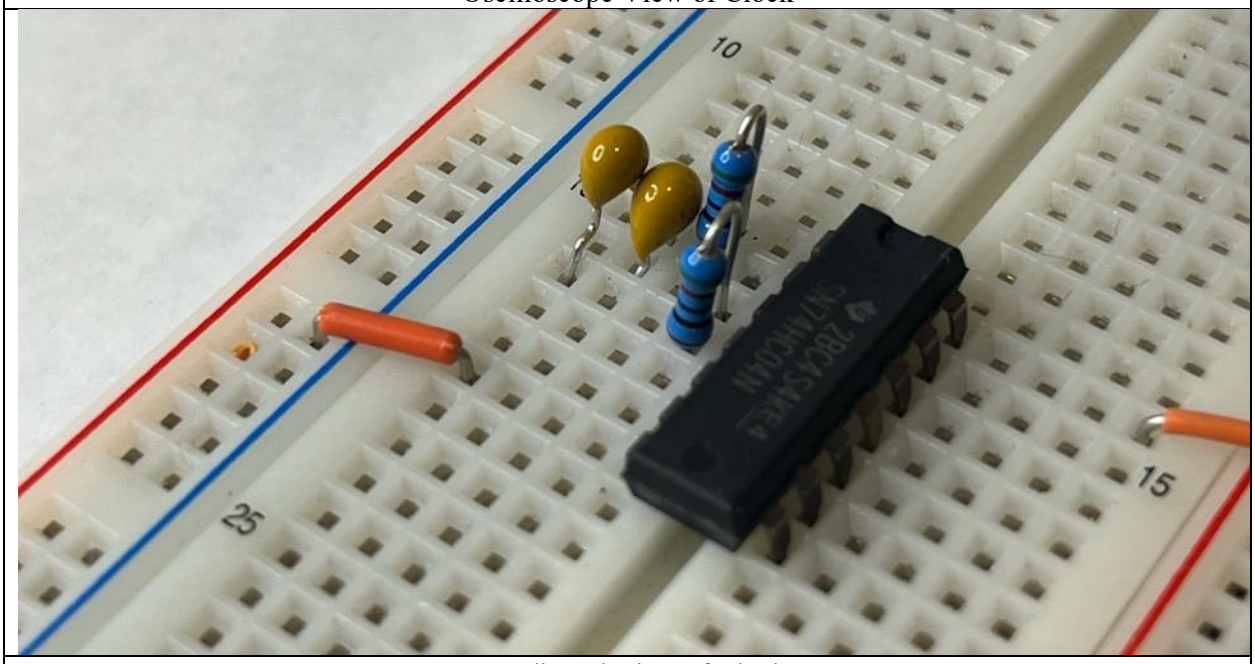
The clock functions by having two NOT gates wired with the input of each the output of each. The inputs are wired to the outputs through resistors and the output of the second NOT gate is wired to the input of the first. The first part to understand is that each circuit is acting as its own oscillator. The pins three to four are having a NOT gate wired to itself which alone will output a square wave will an extremely high period. Now the addition of the capacitor makes it so that when the when the NOT gate outputs said charge the capacitor connected will charge up said capacitor. This is easier understood when view capacitor plates connected to pin 6 and pin 4. In a situation when the NOT gate outputs a value one. So the plate connected to pin 4 is charging making the connection between 4 and 5. Once the capacitor is fully charged the voltage can no longer flow and this leads to no more outputs from the capacitor setting the logic level. This will lead to the logic gates flipping and beginning to discharge the voltage held up in the capacitor in the other direction now having pins 4 and 6 as ground sources as the capacitor begins to charge up in the other direction. This process will continue over and over creating an oscillating circuit.

Media

Project Video: https://youtu.be/pWH6pwmpbN0



Oscilloscope View of Clock



Breadboard View of Clock

Reflection

I thought this was a great start to an interesting project. I thoroughly enjoyed the Digital-to-Analog Converter from last year and after hearing that the grade twelves got to do this, I was even more excited. I think the idea of successive approximation is very interesting how it can be broken down into common ICs just like the CHUMP.

SAR ADC: R/2R Ladder DAC

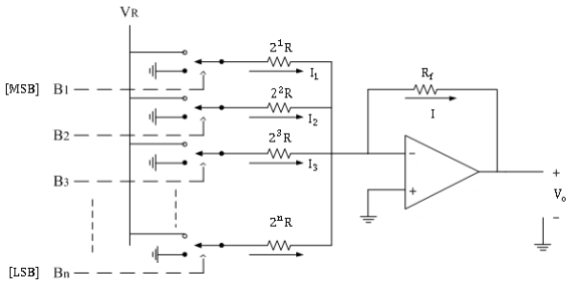
Purpose

The purpose of the R/2R Ladder DAC in the successive approximation register is to create a broad range of digital voltage signals for references for the analog input that is also held by the LF398.

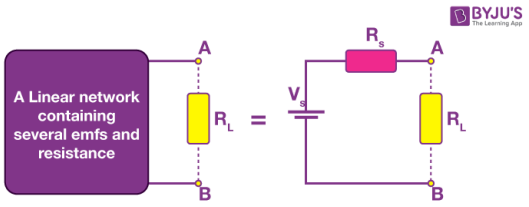
Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEI4M/SARADC/index.html#DAC
SAR-ADC Description: https://hackaday.io/project/181826-homemade-successive-approximation-register-adc
R-2R DAC Description: https://www.electronics-tutorials.ws/combination/r-2r-dac.html

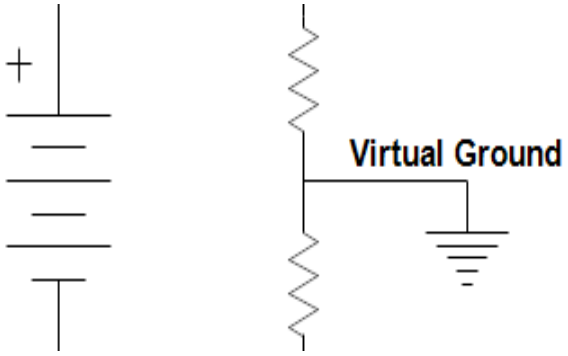
Theory



The theoretical portion of the R-2R Ladder DAC project deals with how this resistor configuration is able create an binary weighted span of voltages. The first concept that makes this configuration possible is the superposition principle. This simply states that when dealing with a linear circuit configuration like a R-2R ladder the sum of multiple independent voltage supplies contributes to the sum of total voltage with all other supplies impeded. How this applies to the DAC is that is it the core principle that allows for a digital signal to be created from the DAC. Each voltage supply is linearly increasing in its summation to the output. This is because digital signals in binary follow the powers of two meaning the next bit is double what the last bit was. This means that a digital bit can be assigned to every signal switch because the weighting of each switching being on or off is linearly increasing. An equation that helps to explain this is when thinking about 3-bit DAC $V_{out} = V_{ref}(\frac{D_0}{2} + \frac{D_1}{4} + \frac{D_2}{8})$ where each D is either a 1 or a zero contributing to the output. What the equation shows is the weighting continuously decreasing just like a 3-bit value.



The second principle to understand is Thevenin's theorem. This states that once again when working with any linear circuit like a resistor network the circuit can be simplified into one voltage source and one resistor. What this is useful for is in equivalating the circuit to find the weighting of each switch. The first switch closest to the output can have all resistors behind simplified into just a single resistor that takes half the voltage source. From using this approach each weighting can be found to equal a binary weighting of a digital signal.



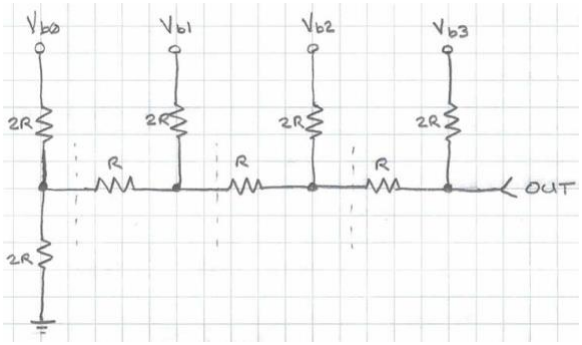
Lastly, a simple theoretical concept is how negative voltage can be created. The diagram on the side shows that when connecting the negative lead of one battery into the positive lead of another provides a -9V supply. This is due to the fact that when working with voltages everything is considered to be relative. So, in setting the positive rail of one supply to 0V the relative voltage is always 9V less with respect to the positive lead provide a -9V supply.

Procedure

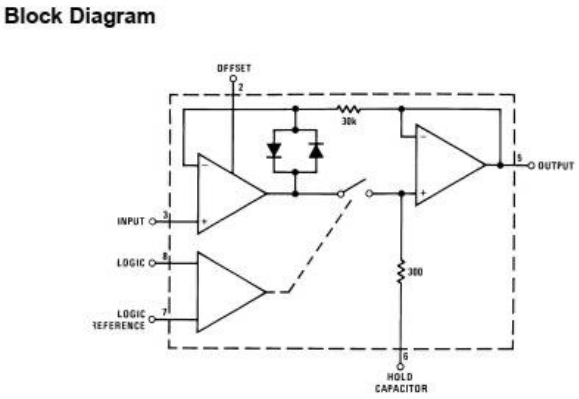
The goal with all essential parts shown to the right is to create a way for the Op-Amps comparator to compare and analog signal that is sampled and held by the LF398 and to compare it with the R-2R Ladder DAC. Some key things to note about this portion of the circuit include the Op-Amp being an open collector Op-Amp therefore requiring and extra resistor that is pulled high at the output pin for the true results. The switch bank used provides the 0 and 1 digital signals that create an analog comparing and will be replaced with automating parts in feedback.

Parts Table	
Quantity	Description
1	LM393 (Op-Amp)
1	LF398 (Sample and Hold)
1	10 kΩ Potentiometer
6	5.1 kΩ Resistors
7	SPDT Pull Switch
8	10.2 kΩ Resistors
1	1000pF Capacitor

The first key component to talk about is the R-2R ladder. As the name shows the only two things required to make this circuit possible are a selected resistor value of R that is current limiting and a second resistor that is 2R. The circuit is based on the principles discussed above and the idea that two resistors in parallel will equal half of each resistor value. What this means in this circuit is that the circuit to the right starts of at the end with the two points that have a 2R resistor in parallel configuration. From this they can be equivalized to equal R and put in series with another R to create another 2R. This process continues with 2R resistors in parallel turning into R and then becoming another 2R. What this leaves is a singular R resistor that is connected to a voltage source. The configuration acts as current divider where each time fewer current flows to ground and each time is getting two time less of an amount.

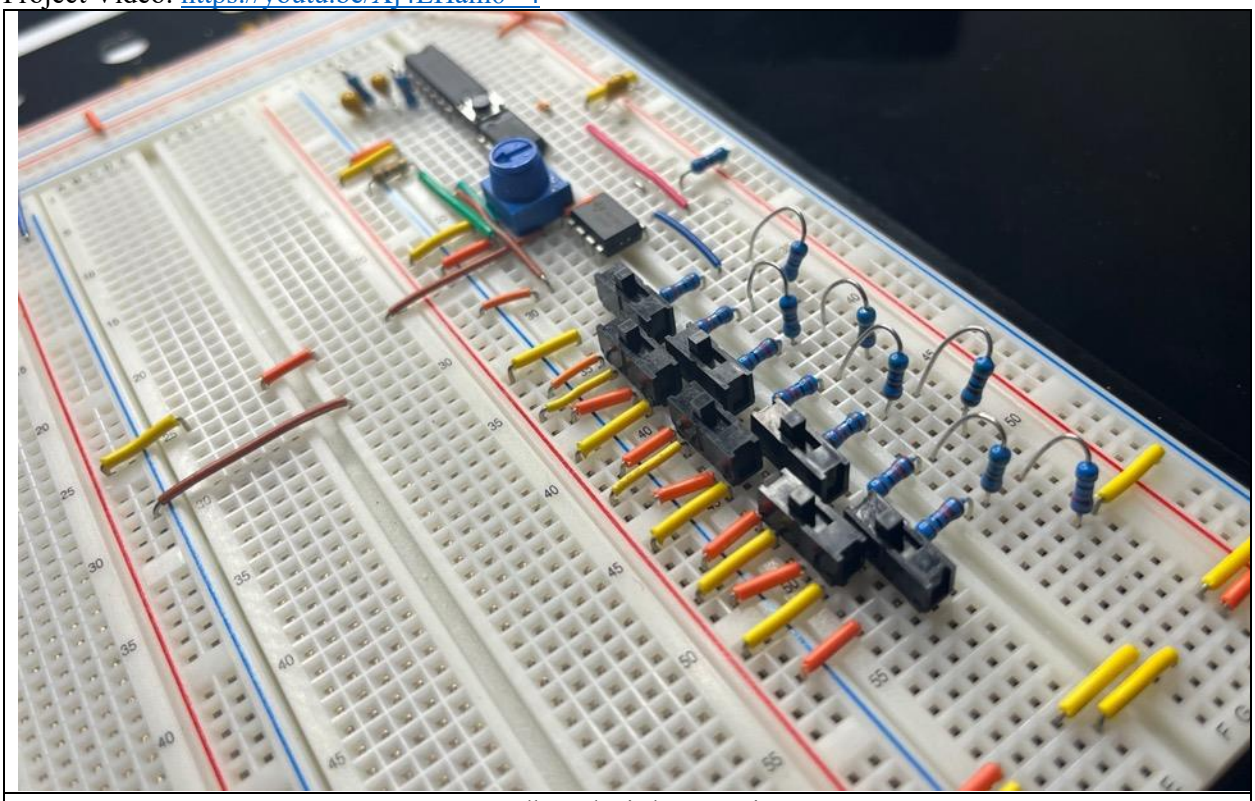


The second critical IC to discuss is the LF398 sample and hold IC. A very key part of the ADC is to be able to actually get a voltage that can be successively approximated from a sine wave. The approach done for this is through using an LF398. The procedure for using this IC starts with understanding the goal of what has to be done. When the button is pressed the IC is taking an analog signal from the analog input pin and holding it through a 1000pF capacitor which is flowed through a buffer amplifier to ensure no voltage is lost. This begins with a input buffer Op-Amp taking an input signal without and current being used as an Op-Amp impedances to make sure there are no losses. Following this the logical signal is wired to a FET or BJT to ensure a fast switch from sample to hold. The input is fed to a low leakage capacitor that holds the voltage to ensure that when the switch is lost the signal is preserved. This is once again fed through a buffer that makes sure no current is lost finally fed through the circuit once again unless a new input disrupts the operation.

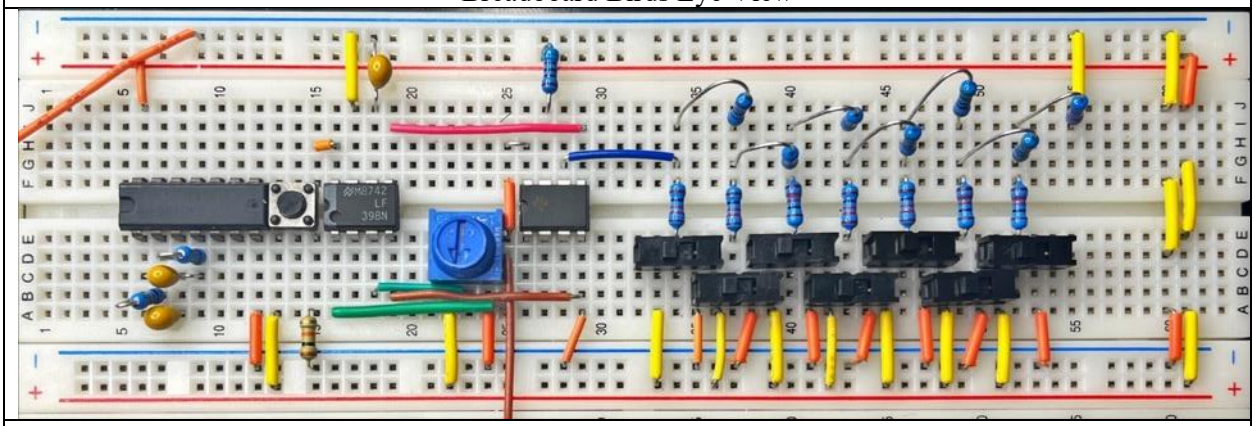


Media

Project Video: https://youtu.be/Xj4EHam6--4



Breadboard Birds Eye View



Breadboard Side View

Reflection

I thought this project was a great experience to reflect on all that I have learned over the past year and a half since the last time I worked on a project titled "R/2R Ladder." I saw all the knowledge of how the principles and voltage division worked in a new way and enjoyed the project. One interesting thing was my encounter with not having a switch bank. Doing over 20 DERs taught me how to combat minor problems like this to hand in the project on time.

SAR ADC: Completed

Purpose

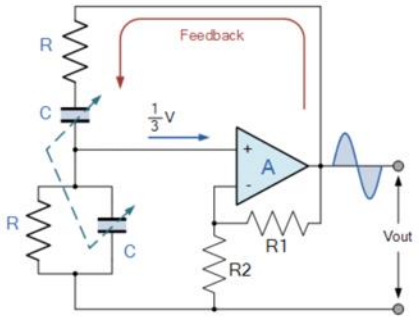
The purpose of the completed SAR ADC is to create a register that can provide a digital output for an analog input adding onto the SAR ADC clock and R/2R Ladder.

Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEI4M/SARADC/index.html#Completed
Project Inspiration: https://cdn.hackaday.io/images/3844351681867541946.jpg
Wien Bridge Oscillator Video: http://www.youtube.com/embed/geNOEzAMXKY

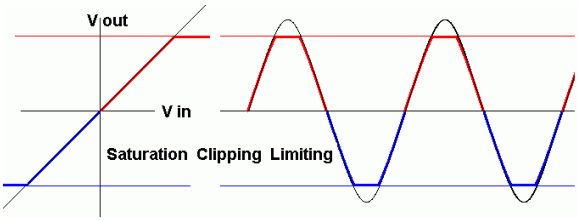
Theory

The first theoretical addition in this portion of the SAR ADC is how a Wien Bridge Oscillator can be created. The image on the right shows the three different portions of the circuit: an inverting amplifier, an RC configuration in parallel and a similar one in series. The first section of the circuit acts in a simple amplifier form, taking the output signal back through to the inverting pin with the ratio of the feedback resistor to the inverting pin resistor, deciding the gain. The specific gain of three allows for no Op-Amp saturation or oscillation dampening. Now, the two following portions work together to create oscillation.



For an even sinusoidal wave capacitor, they should be equal to each other, and the same goes for the resistors. This is because the series RC configuration acts as a high-pass filter while the parallel acts as a ground, and only at a resonating frequency is a sinusoid created. The RC configuration in series acts as a differential amplifier, creating a sinusoidal wave.

In including the Wien Bridge Oscillator, another theoretical factor is how saturation and supply voltages can affect outputs. An ideal Op-Amp has two supply rails used in several calculations to create the amplitude of sinusoid waves, playing a role in how an Op-Amp gains and what the maximum voltage possible is. This is important because although the Op-Amp allows for 9V to –9V, this is only used. After all, some voltage is lost in all the transistors configured in the Op-Amp, making it not rail to rail, leading to all input being between 0V and 5V so as not to lead to misleading readings.



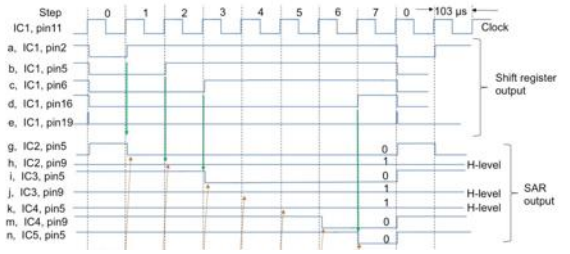
An additional theoretical factor to consider is the current sinks and sources of the AND gates in the newest section of the SAR ADC. The AND gate being used in this project is the CD74HC08. This component is used with the R/2R ladder, creating analog signals through the AND gates. This requires the AND gates to create viable ground and voltage sources. The specifics of this exact component are shown on the right, with three additional buffers for every AND gate, allowing for 52 mA of current either acting as a sink or pump. Based on previous creations of this project, it should be suitable.



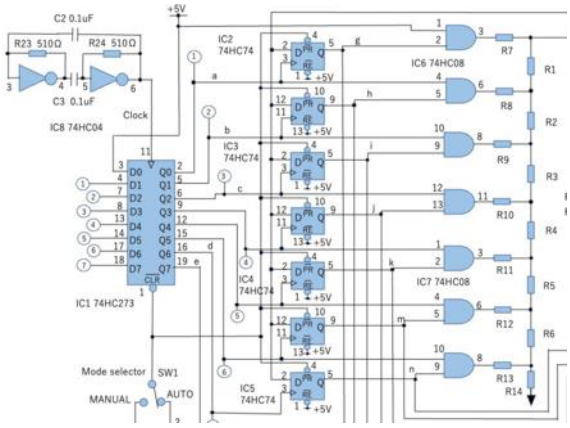
Procedure

Parts Table	
Quantity	Description
1	LM393 (Comparator)
1	LF398 (Sample and Hold)
1	10 kΩ Potentiometer
6	5.1 kΩ Resistors
8	10.2 kΩ Resistors
1	1000pF Capacitor
4	74HC74 (Flip-Flops)
2	74HC08 (AND)
2	74HC273 (8-bit Flip-Flop)
1	74HC04 (NOT)

When in function, an analog signal in the range of 0V to 5V is inputted into the LF398 sample and held in a buffer. The rails of the LF398 are +9V and -9V, requiring two additional power supplies. The analog input is held at the non-inverting pin of the comparator, whilst the inverting pin is connected to the successive approximation portion of the circuit. The first rising edges of the 10kHz clock created with the NOT gate and RC configuration are inputted to the 8-bit Flip-Flop, all loaded with high logical inputs. The first rising edge sends a rising edge to the first individual D Flip-Flop clock pin. This sends a high value to the MSB pin of the R/2R DAC. This value is sent to the comparator's other input, and based on whether it is greater or less than the input, a one or zero is sent back to the data pin of the Flip-Flop. This cycle is then repeated with the following six inputs to see the values until the falling edge is sent to all clock pins, and starts the process again.

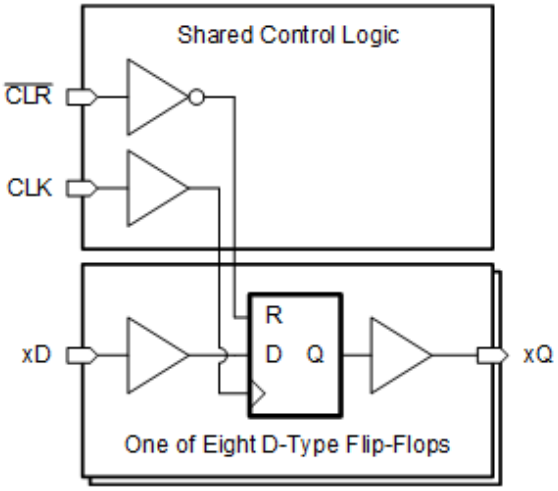


One of the most crucial parts in understanding how the integrated circuits function together is understanding the clocking of each of the Flip-Flops. The shift registers act as all others looked at in previous reports with the digital input going to the output only when a rising or in some ICs a falling edge. The image on the right is an insightful diagram of all logic levels and timings that make the SAR ADC possible. The first thing to examine is the difference in flip-flops. IC1, the 8-bit Flip-Flop will examine a new part of 7 bits after a full 103 us cycle. This means the SAR ADC takes 721 us to run through an entire 7-bit analog signal. The first rising edge from the clock sends a rising edge to the first D-Flip-Flop which checks MSB and send the Op-Amps feedback back into the output of the Flip-Flop not to be changed again. One thing to note is after the clock cycles are ran through every flip-flop is loaded with a high value.

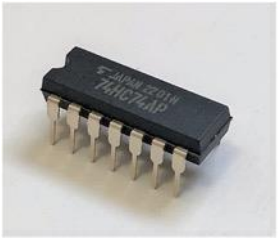


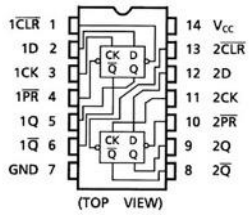
The following new circuit part involves the AND gate configuration with the R/2R ladder. For refreshment, the R/2R DAC provides incremental comparisons to the analog signal to approximate the analog signal as a digital value. This requires the R/2R ladder to change to compare the correct values continuously. The key step in this process is to have a way to hold the proper input for the DAC. This requires two inputs needing to be on. The first is the clock input. This is key because every flip-flop is preloaded with a high value at the start of successive approximation. This means if no AND gate was in place, the R/2R would output a near 5V value, yet because the AND gates have a second input from the 8-bit flip-flop, the entire DAC doesn't go high. When the first 8-bit clock cycle goes high, the R/2R is already comparing the analog value to the MSB, yet now it is sending the comparison back to the first output pin and a high value to the following AND gate. Suppose the first output was a low value. As a result, the new input to the AND gate hooked up to the MSB of the R/2R would always be low. This continues for the following pins, creating an organized way to use the R/2R ladder.

The final part to understand about the SAR ADC are the additional components that bring the build to be working. Firstly, there is a second 8-bit flip-flop used in this circuit to correctly output the 7-bit signal. The way the circuit function as seen in the timing diagram the outputs are not found all the same time with the MSB found first and the LSB found last. To make the circuit have a 7-bit output appear at once when the final clock cycle goes through the first 8-bit flip-flop the 8th bit sends a high to the second 7-bit flip-flop which because of the clear pin set to high outputs all 8-bits to provide a digital output. Another key part of the circuit is having all the preset pins wired together with the clear pin on the first 7-bit flip flop to allow for the flip-flop to be preset with a high value.



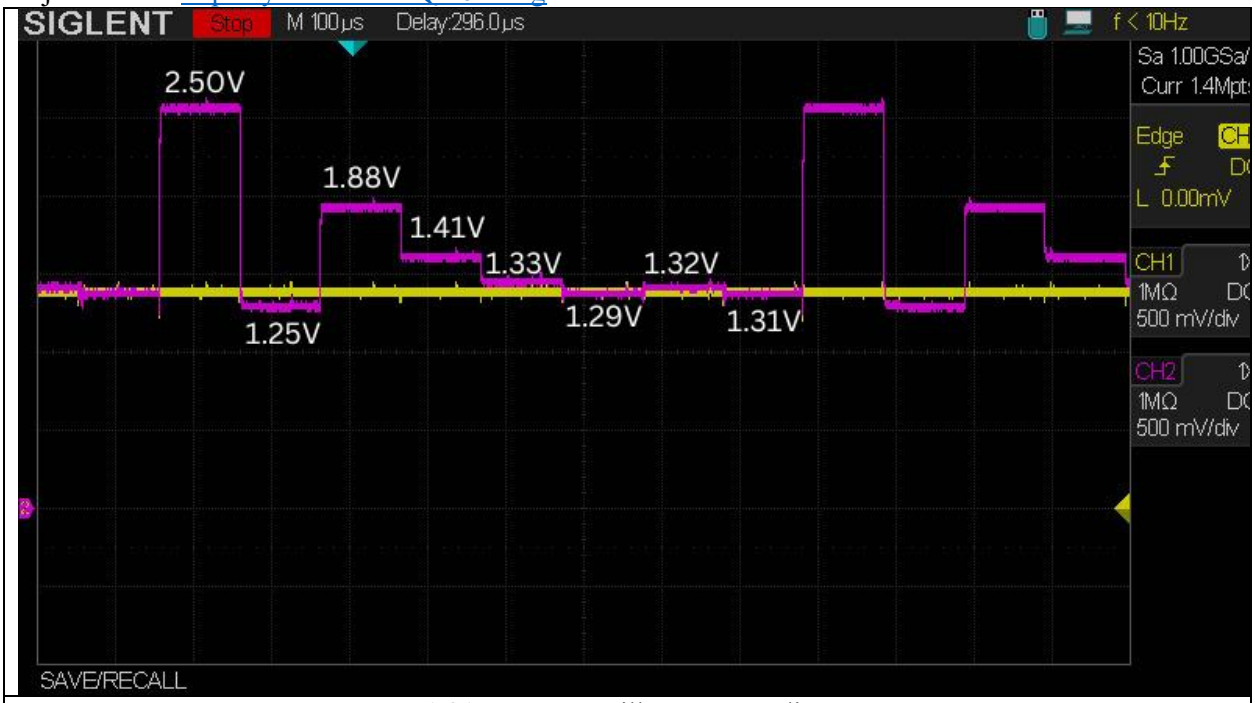
One key IC to be briefly discussed is the 74HC74. For refreshment, the IC consists of two Flip Flops and four necessary pins for this project in each register. Firstly, the data pin is wired across all seven pins, and this can be done because of the clock pin. The clock pin is hooked up individually to each AND gate and, in combination with the 8-bit shift register, creates the successive approximation. The comparison between a digital and analog value is already done before the clock cycle goes through. The op-amps feedback is all the clock records for the output.



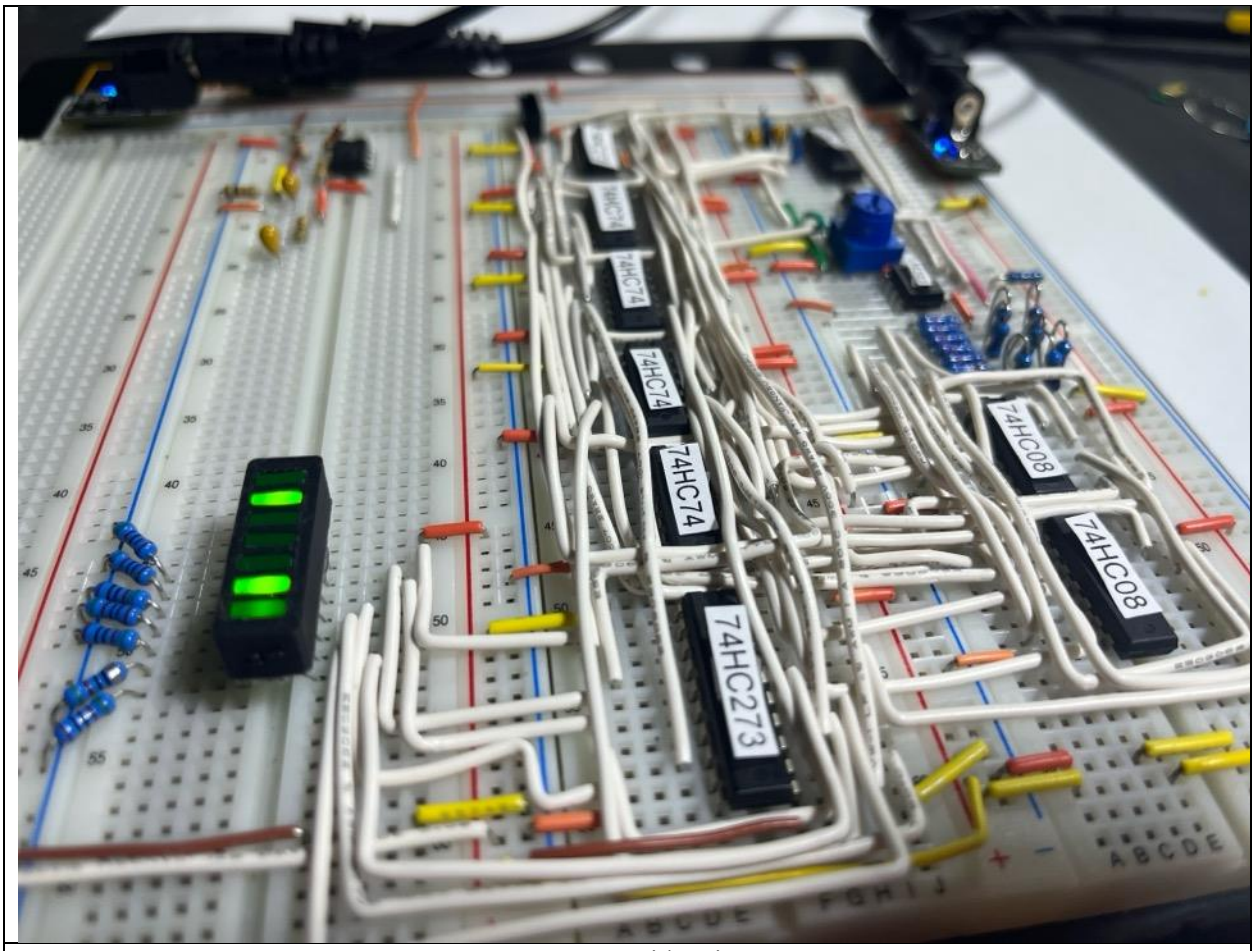


Media

Project Video: https://youtu.be/FXiQFr9A2Fg



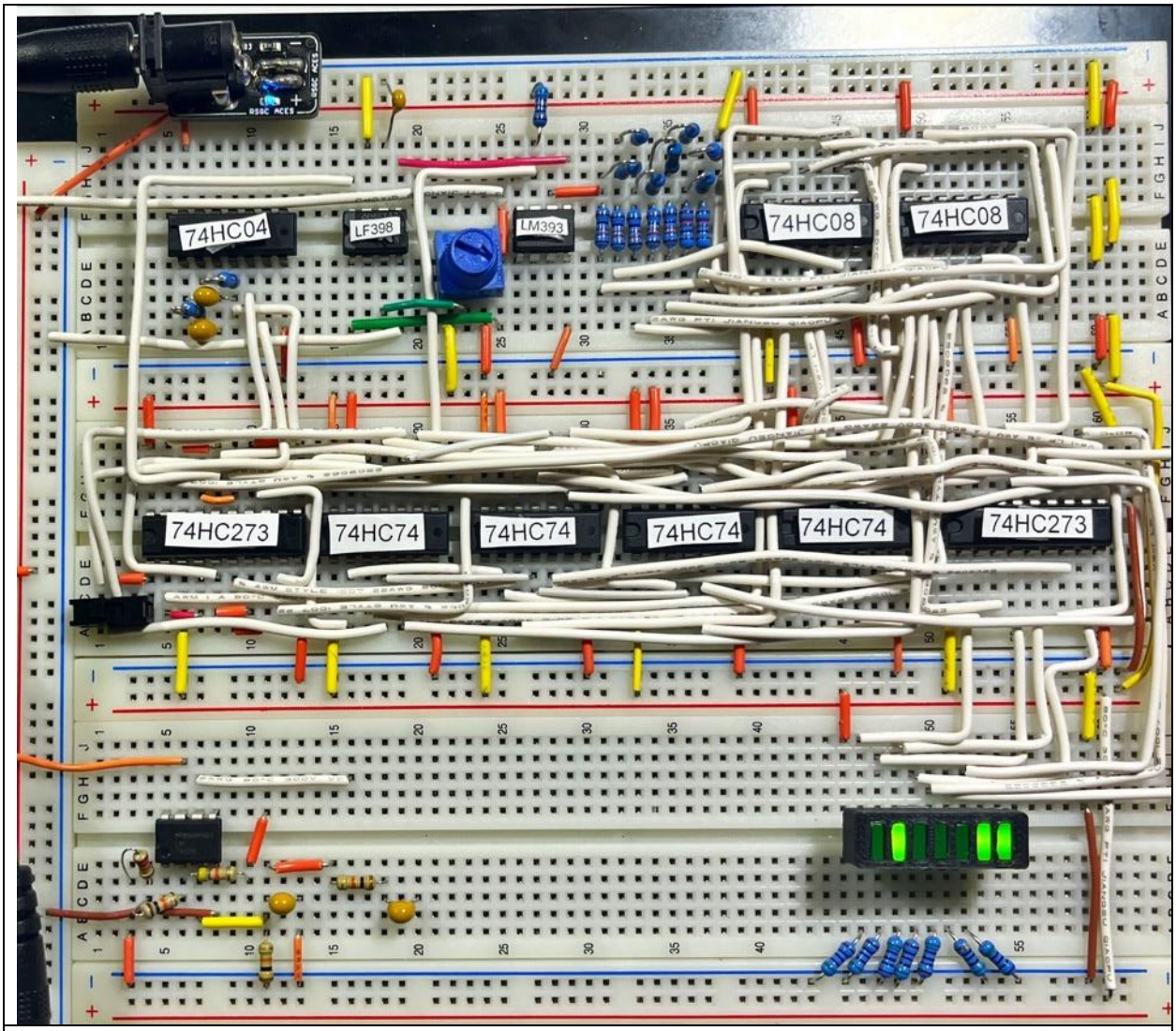
1.31V DAC Oscilloscope Reading



SAR ADC Side View



± 5V Wien Bridge Oscillator DAC Reading



SAR ADC Complete Bird's Eye View

Reflection

I thought that a project that required only 11 ICs would be extremely easy, but it proved to be a challenge. The wiring alone took me 10 hours, and while doing it, I saw how complicated and interesting this SAR circuit was. The way the preset shift registers and clock cycles function together, along with several other interesting ways to create this circuit, really helped me appreciate it. This project has become one of my favourites as I saw how cool the oscilloscope readings were. The intersection between analog and digital components has been one of the things I have loved most about this course, and finally, getting to do this project felt like the best of all those projects from before. Finding out how these standard devices work at a discrete level is one of the most interesting things you can do, which is why this project was so interesting. I want to thank Mitsuru Yamada for coming up with this great design.

Medium ISP: ACES DC-DC Converters

Purpose

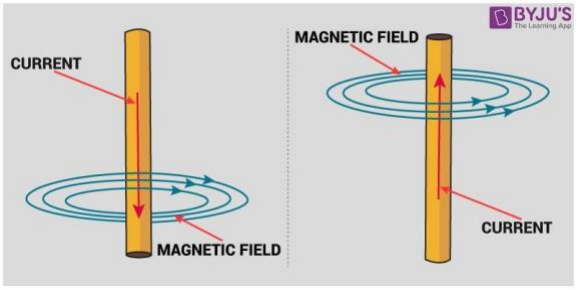
The purpose of the Medium ISP ACES DC-DC Converters is creating two circuit topologies with two different effects on an input voltage. The Asynchronous Boost converter topology uses one switch and one controller to step-up a source voltage. The SEPIC Asynchronous Buck-Boost converter takes can step-up and step-down a single voltage source. Both converters use an ATTiny85 instead of a Power Electronic controller to achieve a deeper understanding.

Reference

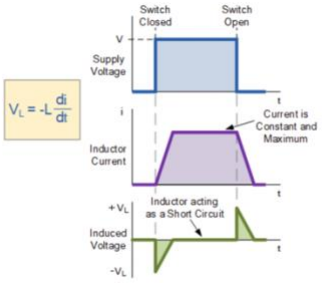
Project Description: http://darcy.rsgc.on.ca/ACES/TEI4M/2425/ISPs.html#logs
UBC Power Electronics: https://www.youtube.com/@PowerElectronicsUBC/videos
Boost Converter Calculations: https://www.ti.com/lit/an/slva372d/slva372d.pdf?ts=1743425532054
SEPIC Converter Calculation: https://www.onsemi.com/pub/collateral/and90136-d.pdf
Inductor Description: https://en.wikipedia.org/wiki/Inductor
ATTiny85 Datasheet: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2586-avr-8-bit-microcontroller-attiny25-attiny45-attiny85_datasheet.pdf
MAX17502 Datasheet: https://www.analog.com/media/en/technical-documentation/data-sheets/max17502.pdf

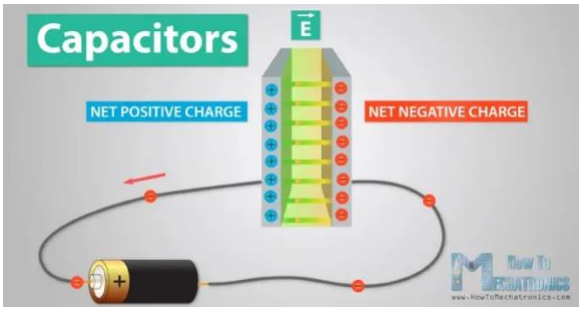
Theory

The first theoretical topic to understand behind the Power Converter is understanding how an inductor functions. With an understanding of basics of inductors, it will make more sense when implemented into a SMPS (Switch Mode Power Supply). Faraday's Laws of induction describes the relationship between magnetism and electricity. On of the laws states when current is flowing through a conductor a magnetic field is created around the wire. An equation representing this relationship is $L = \frac{\phi}{I}$ where I is the current flowing through a conductor, Phi is the magnetic flux generated by the current flow and the ratio is between flux and current known as inductance (L).

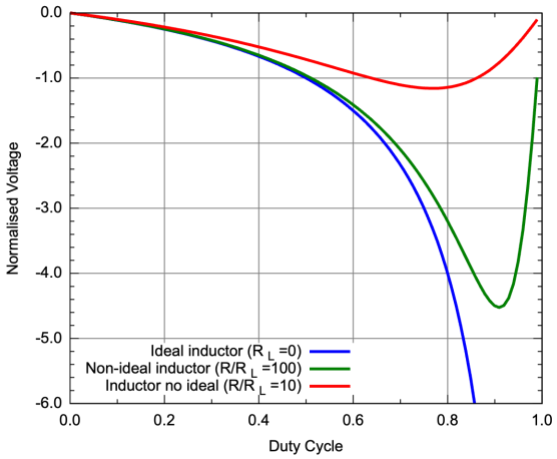


Now with this idea of changing magnetic fields due to current flowing through a conductor. When wires are wrapped around a permeable material this creates an extremely larges magnetic field. When the current flowing through inductors is changed this causes a change in the magnetic field which Emil Lenz realized is being opposed by some force. The force opposing the change in magnetic flux is an induced voltage. An equation that encapsulates the idea that a change in magnetic flux is always opposed by an induced voltage is $\mathcal{E} = -\frac{d\phi}{dt}$ where epsilon is the induced voltage. If the equation from above on the ratio between magnetic flux the equation become $\mathcal{E} = -L\frac{dI}{dt}$ which provides one of the most important equations for understanding how a boost converter functions. From this derived equation and analyzing what happens to voltage applied an inductor the reciprocal equation to the induced voltage equation can be made, $V = L\frac{dI}{dt}$.

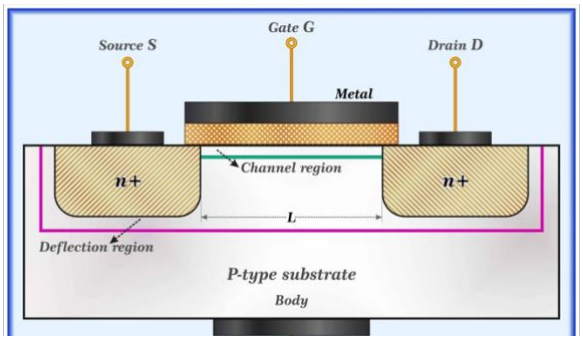




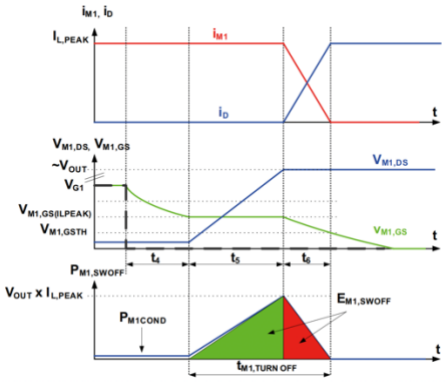
After deriving the equation for the inductor when voltage is applied. The next step is understanding the other passive component that makes this circuit possible, the capacitor. The equation for the dielectric that holds energy in the form of an electric field is Q = CV where Q is the electric charge held in the capacitor and C is the capacitance of how much voltage can be charged and discharged. The key part of this equation is that the larger a capacitor the more voltage ripple it can handle before becoming an impeding component. As a result of this when implemented into a boost converter the relationship between current and voltage can be found by differentiating the equation above and agreeing to the equation $\frac{dQ}{dt} = i_c$. Therefore, $i_c = C\frac{dv}{dt}$.



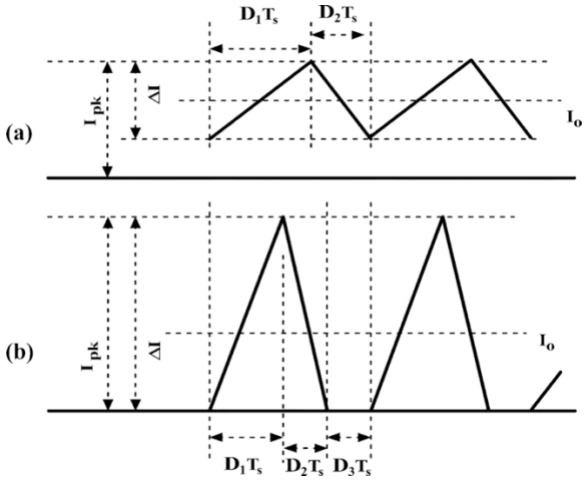
One theoretic idea to keep in mind with both capacitors and inductors in components purchased is the parasitic resistance that each component has. Parasitic resistance can stem from the materials made to create the components, the layout requiring a large amount of wiring and several other reasons. This DCR (Direct Current Resistance) effects the equations used because of the addition by Ohm's law. The important current and voltage relationship through an inductor $V_L = L\frac{di}{dt}$ which in the case of the boost converter is outputting a linear triangle wave is affected by the addition on resistance because the equation becomes $V_L = L\frac{di}{dt} + iR$. Leading to a larger resistance making the linear equation more exponential with respect to time. The more exponential, the less the mathematics work for the converter so parasitic resistance is kept to a minimum. The graph shows the effect larger resistances have on a circuit in a buck converter.



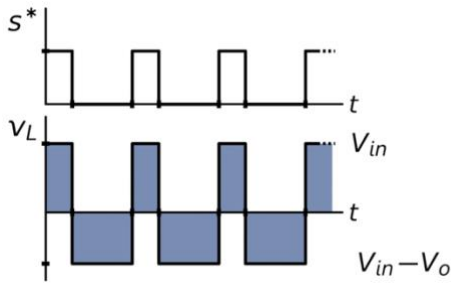
Next on the theory behind components used in the Power Electronics is the Metal Oxide Field Effect Transistor (MOSFET). The MOSFET is a 3-pin transistor that provides a connection between the source and the drain in a much different way than a BJT. In a bipolar junction transistor, there are specific spaces known as the depletion region where current and voltage cannot flow unless the flow of current is so great it jumps this region. A MOSFET, however, uses a generated electric field to create the pathway for source to flow to drain. The advantages of field effect transistors start with how the switching happens. In the case of the boost converter a high switch frequency signal is required. In the case of the ATTiny85 the output is 20mA which would not function with a BJT so a field effect transistor, which requires voltage works perfectly for this application.



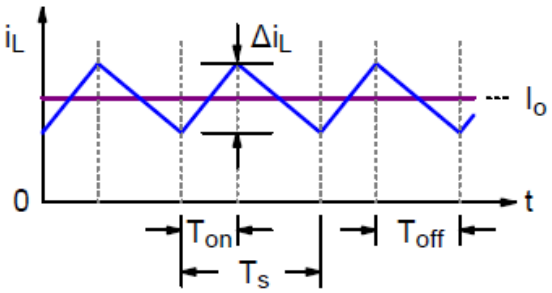
Turning a MOSFET on an off is much more complex than expected when implemented into a boost converter. One key consideration around the MOSFET are the power losses experienced in conduction and switching that dissipate into heat. Conduction losses in any MOSFET is $P_{cond} = I_{RMS}^2 \times R_{DS}$. The resistance from drain to source, a value based on MOSFET selection, is multiplied with the RMS of current squared which is based on 2 independent variables explained in the I_{RMS} derivation. When taking the RMS equation when voltage is at a minimum to get the maximum current, $I_{RMS(MAX)} = \frac{I_{out}}{1-D} \times \sqrt{(D)}$. The conduction losses are $P_{cond} = \frac{(I_{out})^2}{(1-D)^2} \times D \times R_{DS}$. Providing a relationship between current, duty cycle and the power losses during the conduction. The P losses during a switch cycle are much more significant and can be extrapolated from graphs to derive the equation. The graph on the right shows the MOSFET as voltage overcomes the gate threshold being the switching on period. For note the switching off period is the exact same triangle only dependent on duty cycle. This equation is derived by calculating the area under as $\frac{V_{out} \times I_{L,Valley} \times t_{M1ON}}{2} = E_{ON}$. So, finding power just includes multiplying the switch frequency. And if this triangle is the equivalent during the off time, then the equation for a power lost during a switching cycle becomes $P_{SW} = \frac{V_{out} \times I_{L,Valley} \times t_{M1ON} \times f_{sw}}{2} + \frac{V_{out} \times I_{L,Valley} \times t_{M1OFF} \times f_{sw}}{2}$ which is simplified to $P_{SW} = \frac{V_{out} \times I_{L,Valley} \times f_{sw}}{2}(t_{M1ON} + t_{M1OFF})$.



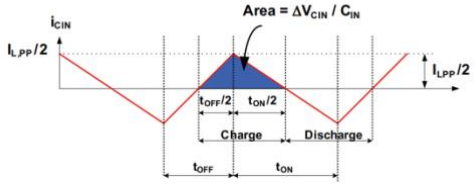
One other critical idea to note about all equations described above and below is that they follow the rule that the current flowing through the inductor is in continuous current mode (CCM). When examining the current outputted from the inductor the output can be seen to the right as a triangle wave. This triangle wave will be further examined in the theory and procedure sections but the important part of the current output is that both parts of the inductor cycle are outputted being when the inductor is short circuited (on state) and when the energy from the magnetic field is transferred to the capacitor (off state). This is a situation that only happens when the peak and trough of the inductor current are both above 0 amps. If this condition is not met then the current cycles go into discontinuous current mode (DCM) where the calculation are no longer dependent on just duty cycle between input and output voltage. To make sure this situation does not happen the boundary CCM equation is employed. This equation expresses the relationship between duty cycle, voltage, frequency and inductance. $I_{Bound} = \frac{V_{OUT}(1-D)^2}{2fL}$. And the only condition to have CCM is that the output current load has to be greater than this value. So, an approach to find if CCM is every not met is taking the minimum voltage output at the duty cycle that is and seeing if the frequency and inductance value are suitable.



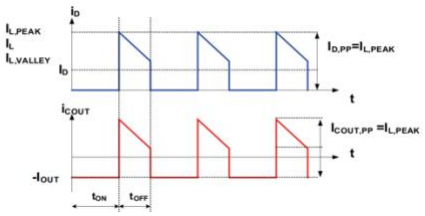
Lastly, there are several key equations around key components that when derived help understand the circuit better and each components place. The first derivation and possibly the most important is the relationship between input and output voltage. When the switch is on the voltage at the inductor is equal to the input voltage as the inductor is shorting to ground at this time $V_{Lon} = V_{in}$. When the MOSFET now turns off the inductor current is equal to the input voltage subtract to the output voltage as there is a voltage difference between the two, $V_{Loff} = V_{in} - V_{out}$, no current will flow due to the diode. Due to the volt second balance equation that state all voltage flow in an inductor must equal the voltage flowing out during a full switching cycle. Mathematically this can be expressed as $\int_0^T V_L dt = 0$ where T is the time of a full switching cycle. This equation can be expressed as $\int_0^{t_{on}} V_{Lon}\, dt + \int_{t_{on}}^{T} V_{Lon}\, dt = 0$. Expanding the equations defined before this becomes $\int_0^{t_{on}} V_{in}\, dt + \int_{t_{on}}^{T} V_{in} - V_{out}\, dt = 0$. Taking the integral provides the equation $t_{on}(V_{in}) + t_{off}(V_{in} - V_{out}) = 0$. The time values can be expressed in term of duty cycle $DT(V_{in}) + T(1-D)(V_{in} - V_{out}) = 0$. And when T is cancelled out and the equation becomes $\frac{D(V_{in})}{(V_{in}-V_{out})} = (1-D)$. This equation is flipped and the duty cycle value is brought to the right side provide a final equation of $\frac{V_{out}}{V_{in}} = \frac{1}{(1-D)}$. This is a fundamental to create a variable boost converter but only work if CCM is met.



Another important equation to learn about is the current ripple equation. Current ripple is the amount of current swing that occurs during only time cycle. This is a value measured from the peak of one cycle to the peak of another expressed with the notation ΔI_L. To understand this equation the concrete equation is enabled being $V_L = L\frac{dI}{dt}$. When dividing L by both sides and integrating with respect to time the equation becomes $\frac{V_L}{L} \times T = I$. And in measuring the gain of only the ON time has to be examined which comes out to be $\frac{V_L \times D}{Lf} = \Delta I_L$. This equation is critical for understanding how to achieve CCM and correct inductance values.



The final equations to understand are the input, output capacitors relationship and . When using a boost converter several different types of current loads can be taken from the input power supply causing voltage ripples that have to be delt with. To find out how much input voltage ripple is present during CCM is by the area under the peak-to-peak current graph which is equal to $\int_{\frac{t_{on}}{2}}^{\frac{t_{on}+t_{off}}{2}} i_{Cin} dt = \frac{1}{2} \times \frac{I_{LPP}}{2} \times \frac{t_{on}+t_{off}}{2}$ which can be solved to be $V_{CinPP} = \frac{I_{LPP}}{8 \times F_{SW} \times C_{in}}$. This provides a relationship between the current ripple and voltage ripple. As the capacitor and switching frequency increase the input voltage level increases so in the case of a power converter operating at a high frequency makes this value negligible.



Next, the output capacitor is the balancing component to store voltage and current. As this is a switching power supply one of the most crucial things is having a component that evens out the output when the voltage being supplied is dropped along with temporarily supplying the current load when the cycle is in the ON cycle. Due to this idea of what it is supplying the output load $i_{Cout} = C_{out}\frac{dV_{out}}{dt}$. And when isolating this equation for the output voltage requires integrating ($\int_0^{t_{on}} i_{Cout} = C_{out}\frac{dV_{out}}{dt}$) with respect to time and when rearranging $V_{out} = \frac{i_{Cout}}{C_{out}} \times t_{on}$ and when adjusting time for duty cycle and switching frequency $V_{out} = \frac{i_{Cout} \times D}{C_{out} \times f_{sw}}$. This provides the second capacitor equation where voltage is proportional to current and duty cycle while inversely proportional to switching frequency and capacitance. The graph on the right shows what output is sent to this capacitor.

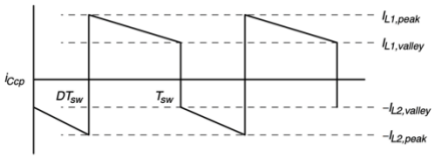


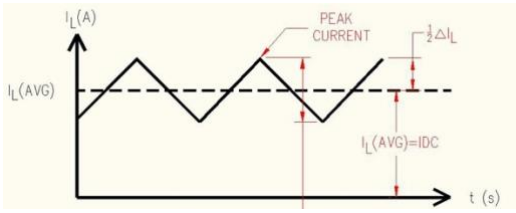
Figure 6. Capacitor Ccp, Currents

A SEPIC and Buck converter follow the same design consideration and parameters around components varying only slightly with the only large difference being the relationship between input and output voltage. The best graph to understand the SEPIC converters voltage is the current graph connecting the two inductors. If both inductors function is understood the relationship in voltage will make more sense. The image to right shows the inductor current at the connecting capacitor. The first thing that can be seen is that the peak of second inductor, the output inductor, has a peak at the peak of the ON time and a trough at $T_{sw} - T_{on}$. So, an approach to find an equation that represents this graph is the ampere second balance law. So, the area under the second inductor's current output can be expressed as -$\int_0^{DT}(I_{L2Valley} + \frac{di_{L2}}{dt}(t))dt$. This expresses the slope and initial value of the second inductor. The other half of the equation takes the same properties but now starting from the peak i_{Ccp}. $\int_{DT}^{T}(I_{L1} - \frac{di_{L1}}{dt}(t - DT))dt$. So, to find the average current both integrals are added a put over time. $\frac{1}{T_{SW}}$(-$\int_0^{DT}(I_{L2Valley} + \frac{di_{L2}}{dt}(t))dt$ + $\int_{DT}^{T}(I_{L1} - \frac{di_{L1}}{dt}(t - DT))dt) = 0$. Because of the first derived function $\frac{V_L}{L_2} = \frac{di_{L2}}{dt}$. If this rule is applied to both inductors the equation becomes. $\frac{1}{T_{SW}}$(-$\int_0^{DT}(I_{L2Valley} + \frac{V_{in}}{L_2}(t))dt$ + $\int_{DT}^{T}(I_{L1Valley} + \frac{V_{in}}{L_1}DT - \frac{D}{1-D}\frac{V_{in}}{L_1}(t - DT))dt) = 0$. The first inductors peak current was substituted for the current at the value plus the gain over an on-switch cycle. This is then all integrated becoming -$DT\left(I_{L2Valley} + \frac{I_{L2PP}}{2}\right) + (1 - D)T\left(I_{L1Valley} + \frac{I_{L1PP}}{2}\right) = 0$. Both inductor valley average brackets can be substitutes for just the inductor current values -$DT(I_{L2}) + (1 - D)T(I_{L1}) = 0$. Which can be rearranged to be $\frac{I_{L1}}{I_{L2}} = \frac{D}{1-D}$. Then assuming this is an ideal converter and based on connections $\frac{I_{L1}}{I_{L2}} = \frac{I_{in}}{I_{out}}$ and $I_{in}V_{in} = I_{out}V_{out}$ so as a result $\frac{V_{in}}{V_{out}} = \frac{D}{1-D}$ providing a relationship between input and output voltage through the duty cycle as long as the parameters for components are met along with continuous conduction mode.

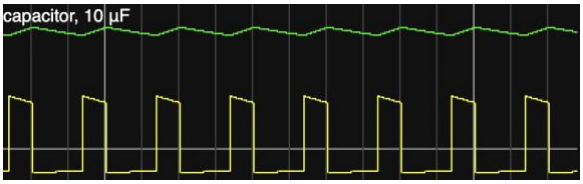
Procedure: Boost Converter

Parts Table	
Quantity	Description
1	ATTiny85
1	100 uH 49 mΩ Inductor
1	IRLZ44N MOSFET
2	47 uF Capacitor
1	100 kΩ Potentiometer
1	900mV Schotty Diode
1	200 Ω Resistor

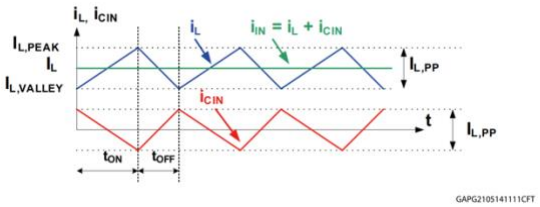
The approach to making a boost converter can be broken down into three blocks. If all three blocks are done correctly a working boost converter will be outputted. The first block includes the step-up voltage portion. This has the job of having the correct construction so when a voltage is applied to the source a higher output will be created. This is the most complex task because every single component has several parameters that have to be met. The equations derived in the theoretical section of the report will become useful for this section. The second portion is the PWM portion. This includes using a component that can output an extremely high frequency and additionally an adjustable duty cycle. Lastly, the error block has the role of finding how much of a difference there is between the intended output voltage and the actual output voltage and adjusts duty cycle to reach the intended value. Blocks 2 and 3 are in the code section of this project.



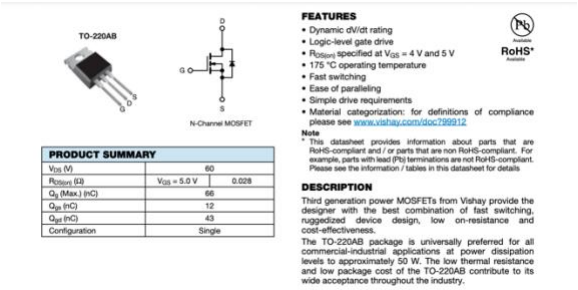
The first part is component selection. The goal is to create an adjustable power supply ranging from 5V to 25V that will have a load of 200 ohms. For starters the component for the inductor can be found. The key equations in finding the correct inductance value include the critical inductance equation and the current ripple equation. The critical inductance equation as derived above is $L_{Crit} = \frac{(1-D)^2 R}{2f_{sw}}$. Now from this equation all value ranges have already been chosen except for the switching frequency, which can go up to almost 1 MHz without effect the duty cycle. So, if an inductance value of 100 uH is desired and the max voltage wanted is I25V then $D = \frac{5}{25}$ and $R = 200$. Rearranging $f_{sw} = \frac{(1-D)^2 R}{2L_{Crit}}$. Which means $f_{sw} = \frac{(1-\frac{5}{25})^2 200}{2\times1\times10^{-4}}$ which is 640 kHz. A larger frequency making an ATTiny85 a perfect component. Secondly, the gain required by the inductor can also be found $\Delta I = \frac{V_{in(1-D)}}{L f_{sw}}$. Then plugging values from above $\Delta I = \frac{5(1-\frac{5}{25})}{(1\times10^{-4})640000}$=.0625A. Then in ratio with current from the load $I_{out} = \frac{25V}{200\Omega} = .125A$. Then find $I_{avg} = \frac{.125A}{(1-\frac{5}{25})} = .1565$ proving a ratio of $\frac{.0625}{.1565} = .4$ for current ripple to current average well under a ratio of 1 keeping the boost converter in CCM.



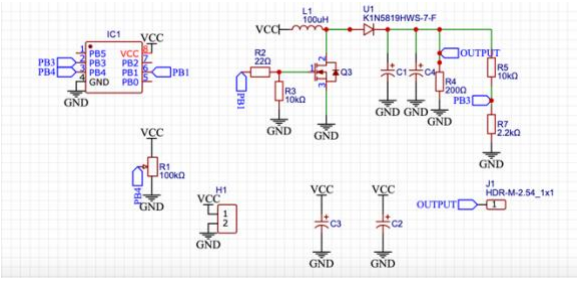
In just the last paragraph switching frequency and inductance value where found. The next step is finding the capacitance of one of the two capacitors, each receive different current flows. The one first examined is the output capacitor or discontinuous current capacitor connected directly to the MOSFET. The Falstad scope on the right shows the current in the yellow colour. From just this graphical reading it can be seen that the output capacitor is supplying the outputs current and voltage during switching cycles. And the voltage ripple is also seen between the switching cycles and needs to be flattened before reaching the output. From an equation above $C_{out} = \frac{i_{Cout}\times D}{V_{out}\times f_{sw}}$ using similar values from the equations before $C_{out} = \frac{.125\times(1-\frac{5}{25})}{25\times640000} = 6.25\times10^{-9}$. Meaning an ideal capacitor is 6.25 nF. For additional ripple when switching the load, a larger capacitor will handle larger ripples in parallel with the ideal.



Next is the other capacitor being the continuous current capacitor connected directly to the input of the inductor being the input capacitor. This capacitor is having a current waveform the right coming from the inductor in blue and handles this acting as the inverse of the capacitor. This means that as the voltage is applied to the inductor charging up and with holding the output of the inductor the capacitor is discharging a current to the load. Then as the magnetic field collapses on the inductor the capacitor charges up. This is a heavy current waveform and is explained with the equation $V_{CinPP} = \frac{I_{LPP}}{8 \times F_{SW} \times C_{in}}$. When swapping voltage for capacitance and using the peak-to-peak value found in the paragraph on calculating inductance the equation becomes $C_{in} = \frac{I_{LPP}}{8 \times F_{SW} \times V_{in}} = \frac{.0625}{8 \times 640000 \times 5} = 2.44 \times 10^{-9}$. This means at the input a 2.44 nF capacitor will be placed to quickly charge and discharge the waveforms. Additionally, because of the damaging risks in unplugging a voltage source from an inductor with a magnetic field a larger capacitor will make sure it is a steady discharge so another 47 uF capacitor will be placed in parallel.



Lastly, the final component to select is MOSFET. There are a lot of components to consider but the first is the heat dissipation. During a switch cycle some amount of heat is outputted but the switch is connected to the rest of PCB so only after a long time will real effects of the of the heat dissipation effect the performance of the converter. The first is voltage rating. The MOSFET should be able to handle the voltage going across so $V_{DSMax} > V_{out} \times 1.2$. Additionally, the resistance from drain to source has to be as low as possible because of the very high switching frequency. If the resistance is too high then the power loss during a switch a switching a cycle is too high because it has an influence on the current going to the drain. A common on the market component that fits these needs is the IRLZ44N. The resistance from drain to source is .028Ω when 5V is applied to the gate being extremely small. The power rating is 110W, much lower than the power used in this converter. The voltage from drain to source is 60V once again much less than $25 \times 1.2 = 30$. Making it a great component.

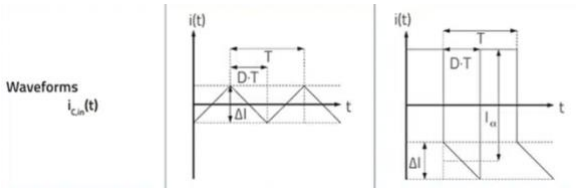


Now with all the components outlined above the ATTiny85 connects pin 1 to the MOSFET for switching. As a high signal hits the switch the inductor builds a magnet field the output capacitor is providing the load power. Then when the switch has a low signal applied the magnet field collapses and provides a higher voltage level to the capacitor which holds the voltage for a moment. This simplistic cycle works perfectly to provide a stepped-up voltage to the output. The two other components to be spoken about in this case are the potentiometer R1 and the stepped down voltage signal applied to the to PB3. Firstly, the potentiometer is selecting the voltage wanted to be outputted to create a variable supply. What every voltage is applied by the potentiometer is multiplied by $\frac{2200+10000}{2200}$. Then the duty cycle will be slightly adjusted based on the load to match the voltage perfectly using the feedback.

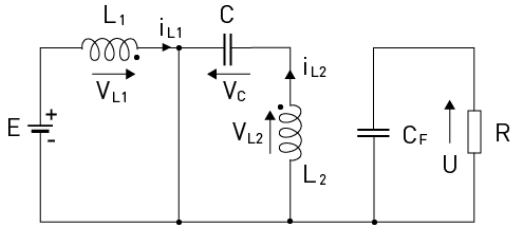
Procedure: SEPIC Buck-Boost Converter

Parts Table	
Quantity	Description
1	ATTiny85
2	22 uH 72 mΩ Inductor
1	IRLZ44N MOSFET
2	47 uF Capacitor
1	10 uF Capacitor
1	100 kΩ Potentiometer
1	900mV Schotty Diode
1	40 Ω Resistor
1	4.7 uF Capacitor

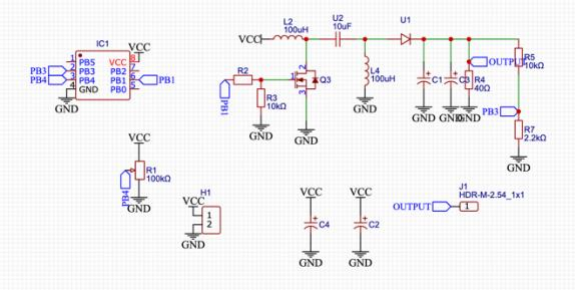
The parts table behind the SEPIC converter looks like almost an extended copy of the parts table for the Boost converter. The symmetry in the power electronic that give such different graphical readings is one of the interesting things about the designs. For starters the same potentiometer, controller and MOSFET are used in this design due to the optimization they provide to the boost converter and will provide to the SEPIC. A few additional components are added and will be spoken about but reused components follow almost the same rules as the boost converter for why selected. The parts discussed in this section are how current flowing through an entire switching cycle, the coupling capacitor, switching frequency and inductance value. The naming behind SEPIC standing for single ended primary inductor circuit. Single ended means there is a common ground between the MOSFETs drain point and the input voltage, Primary inductor means the inductor is connected directly to the input voltage.



Just like with the boost converter some values have to be outlined before deciding an inductance. In this case the goal with the SEPIC converter is to create a variable 3V – 12V DC power supple from a 5V input. Taking this information, the outputs max current can be found to be .3A if a 40 Ω resistor is used. This information provides the ripple current to be $\Delta I = I_{max}\frac{V_{max}}{V_{in}} = .3\frac{12}{5} \times .2 = .144A$. Now from this information the equation $L = \frac{V_{in}}{\Delta I \times f_{sw}} D$ and if the same frequency is wanted $L = \frac{5\times\frac{5}{12}}{.144\times 640000} = 2.2 \times 10^{-5}$. Meaning a 22 uH inductor works perfectly. The other component in the inductor to worry about is the current rating. The peak current at the first inductor is equal $I_{out}\frac{V_{max}}{V_{in}}\left(1 + \frac{Ripple\%}{2}\right) = 1.32A$. Meaning for the inductor as long it is 22 uH and has a current rating greater than 1.32A the circuit will perform around these parameters.



The next thing to discuss is the inclusion of a third capacitor in this design. In the other design waveforms for the input capacitor and output capacitor helped to make the circuit function. In this design there is a third capacitor placed and without this capacitor the waveforms that make the duty cycle equation possible would not occur. A capacitor wired how it is will completely impede any DC current flowing through the component. This means if there was a switch from one load to another the converter would never run into problems with short circuits. This is mentioned because of the capacitors most important role being an energy transfer bridge. When the MOSFET is provide a ground source the coupling capacitor has a positive side as a result. This pulls on the second inductor to create two charging inductors that both discharge when the magnetic field collapses. Without the capacitor L2 wouldn't charge and the converter wouldn't work.



Now with just this new information and using equivalent capacitors at the input and output as the boost converter because of equivalent frequency the last thing is seeing how the SEPIC functions. The first thing that should be known is that when power is initially plugged in it does not operate how it is expected to because for the correct operation the coupling capacitor must be fully charged. But once that component is fully charges and a high signal is placed on the MOSFET the input inductor L2 begins charging. The ground source at the same time is creating current flow for the coupling capacitor to discharge giving a positive charge to the capacitor side connected to ground. The other side of the capacitor create a ground supply for the second inductor L4 and begins another current flow that chargers L4. When the switch is turned off the capacitor no longer has anywhere to discharger, the first inductor is no longer shorted and as a result all energy is outputted into the output capacitors and the load creating the SEPIC converter.

The code used is very basic having the function of recognizing errors between the feedback voltage and the target voltage and fixing the errors by adjusting the duty cycle. The one thing to take note of is the prescaler used to create the fast adjustable PWM frequency. For this part `TCCR0A = (1 << COM0B1) | (1 << WGM01) | (1 << WGM00);`. This moves ones to three parts of the timer counter control register A adjusting the waveform to only rise a until a variable top. Then `TCCR0B = (1 << WGM02) | (1 << CS00);` sets the variable top to be OCR0A and the clock to have no prescalers. Lastly, for a faster clock the 8 MHz clock can be scaled up in programming by selecting PLL.

Code

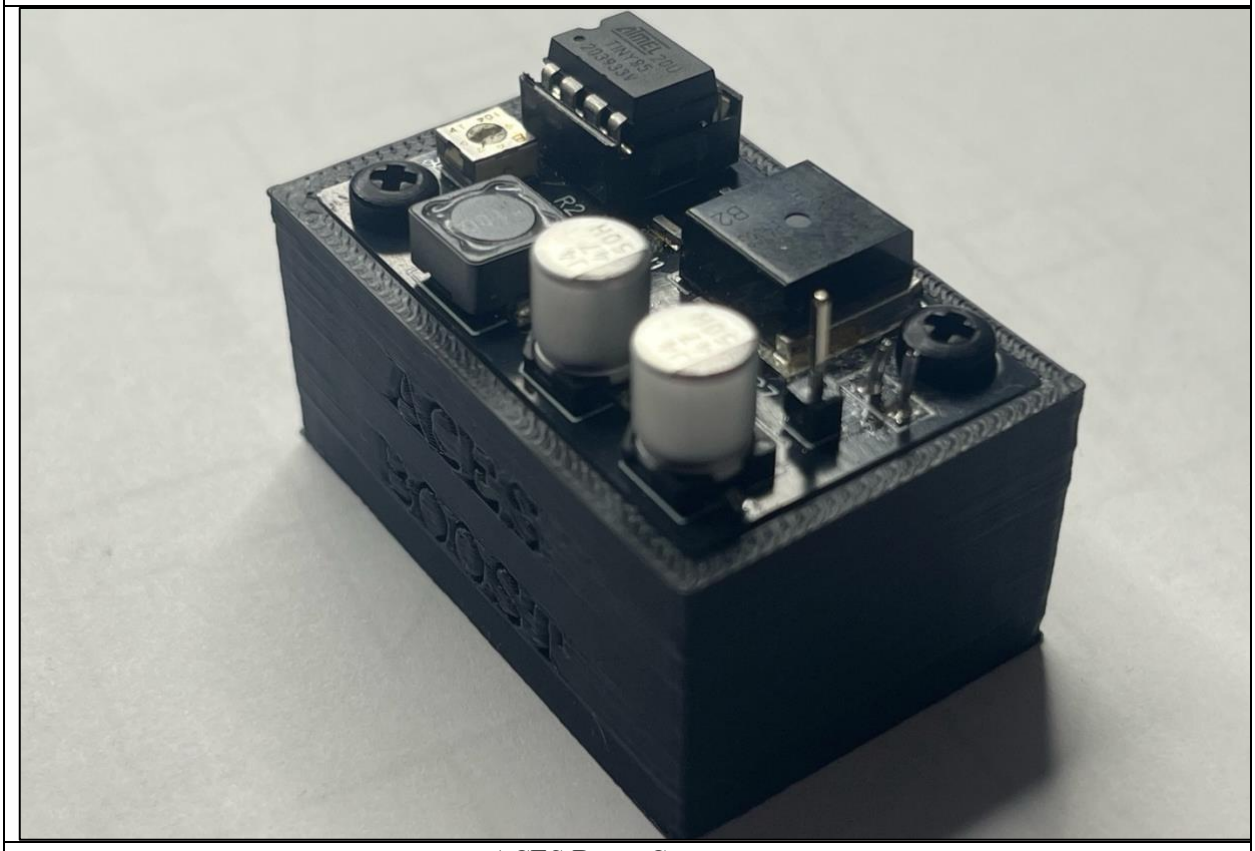
```
// PROJECT  : ACES Boost and SEPIC Converter
// PURPOSE  : Create a PWM signal and controller for the converters
// COURSE   :ICS4U
// AUTHOR   :M. Zischka
// DATE     :2025 04 05
// MCU      :'85
// STATUS   :Working
uint8_t feedbackPin = A3;
uint8_t potPin = A2;
uint8_t pwmMax = 24; //Defines max
void setup() {
  DDRB |= (1 << PB1); //Set PWM pin to output
  TCCR0A = (1 << COM0B1) | (1 << WGM01) | (1 << WGM00); //Fast PWM
  TCCR0B = (1 << WGM02) | (1 << CS00);     //TOP = OCR0A
  OCR0A = 24; //Sets frequency at 633.3kHz using a PLL
  OCR0B = 12; //Orignal Duty cycle of 50%
}
void loop() {
  uint16_t feedback = analogRead(feedbackPin);//Reads scaled down pin
  uint16_t targetADC = analogRead(A2);//sets intended voltage
  if (feedback < targetADC && OCR0B < pwmMax - 5.5)
    OCR0B = OCR0B++;  //if feedback is lower then increase duty cycle max is 75%
  else if (feedback > targetADC && 5.5 < OCR0B)
    OCR0B = OCR0B--; //if feedback is higher then decrease duty cycle max is 25%
  else if (feedback == targetADC)
    OCR0B = OCR0B;
  delay(1);
}
```

Media

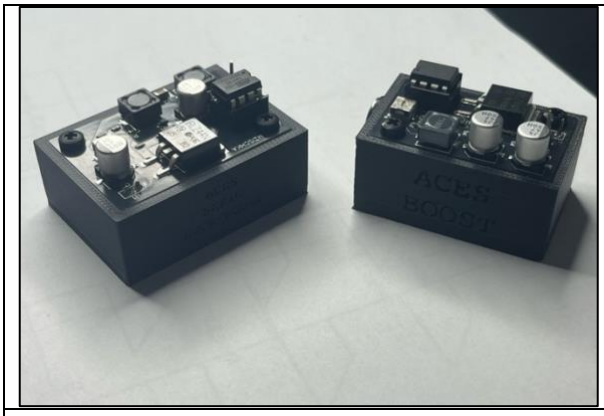
Project Video: https://youtu.be/KyaUuDN0_-o



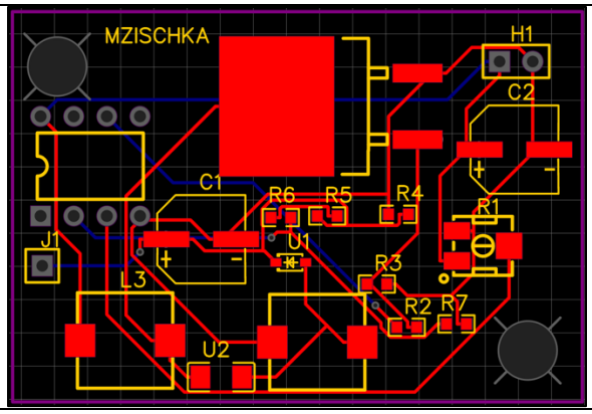
ACES SEPIC Converter



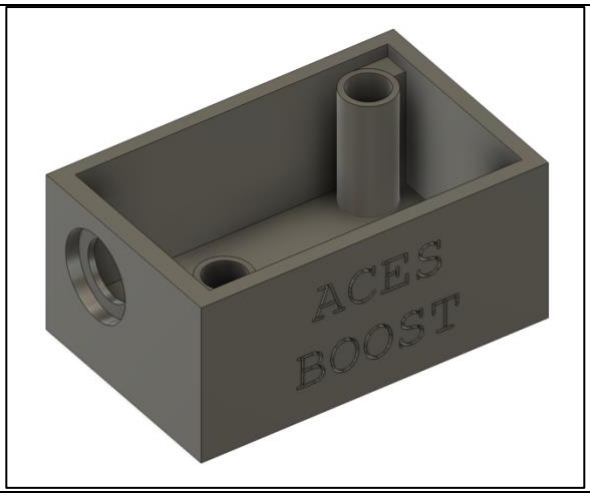
ACES Boost Converter



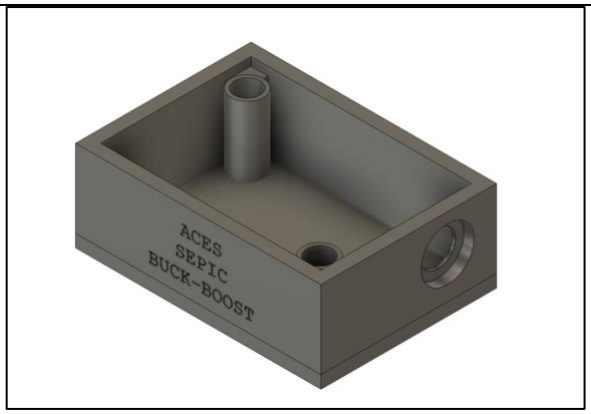
ACES Power Electronics



PCB Board View



Boost Encasement



SEPIC Encasement

Reflection

If I could choose any way to conclude two years of ISPs, this would be the way. This project encapsulated everything I was able to gain from my ISPs and hardware. I chose a difficult yet doable project that focused on my interests while teaching me about power electronics, microcontroller tinkering and much more. The project really was a great ending. A year ago, almost to the date, I was working around the clock to finish a very advanced project, the magic chessboard. The memories and lessons from that project, the 4-bit calculator and the Audio Spectrum Analyzer are worth all the hours invested into them. One lesson, even today, I remembered was that with ISPs, there is always a solution if you work hard and smart. I was tinkering with my boost converter as it only reached 60% maximum duty cycle before encountering problems. The solution was to replace 6 surface mount components, taking 2 hours to debug and replace. This lesson, along with so many others, is the only reason I was able to have a working ISP for today, and I am so grateful to have learned those lessons. Now that my high school ISP career is over, I am excited for my university ISPs and all the other projects I want to build.

AVR Assembly: Double Dabble

Purpose

The purpose of the AVR Assembly Double Dabble is to be able to routinely call an Atmega328p to read analog value and then show the value on the serial monitor all written in AVR assembly language.

Reference

Project Reference: http://darcy.rsgc.on.ca/ACES/TEI4M/Tasks.html#DoubleD
Double Dabble Summary: https://en.wikipedia.org/wiki/Double_dabble
Assembly Instructions: https://www.scribd.com/document/392128494/AVR-Instructions-Set-Summary

Theory

```
0000 0000 0000   11110011   Initialization
0000 0000 0001   11100110   Shift
0000 0000 0011   11001100   Shift
0000 0000 0111   10011000   Shift
0000 0000 1010   10011000   Add 3 to ONES, since it was 7
0000 0001 0101   00110000   Shift
0000 0001 1000   00110000   Add 3 to ONES, since it was 5
0000 0011 0000   01100000   Shift
0000 0110 0000   11000000   Shift
0000 1001 0000   11000000   Add 3 to TENS, since it was 6
0001 0010 0001   10000000   Shift
0010 0100 0011   00000000   Shift
   2    4    3
      BCD
```

The double dabble algorithm is able to take a binary value of any bit size and convert into a binary coded decimal value. This is a critical function because in trying to print the binary value the ADC produces it must be converted to ASCII. Any ASCII number is a BCD value from 0-9 with a decimal 48 added on. So, the approach to print a binary value begins with using the double dabble algorithm. It is very simple algorithm where if any of the BCD values hold a decimal value greater than 5 then three is added to the BCD value. If there is no value great than 5 the BCD value shifts and loads another binary value into the ones digit of the BCD. This algorithm iterates n times where n is the width of the binary value. The reasoning behind the shift and add three is to continuously shift out the binary values and the adjust the value to not overflow. The reasoning behind adding three after the value equals 5 or is great is because it is impossible for a value to equal more than six before the addition of three. This makes it possible to continuously shift out binary values and never overflow values. The image on the left shows a complete double dabble conversion.

```
cook@pop-os:~$ ascii -d
   0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
   1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
   2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
   3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
   4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
   5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
   6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
   7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
   8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
   9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
  10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
  11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
  12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
  13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
  14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
  15 SI     31 US     47 /    63 ?    79 O    95     111 o   127 DEL
```

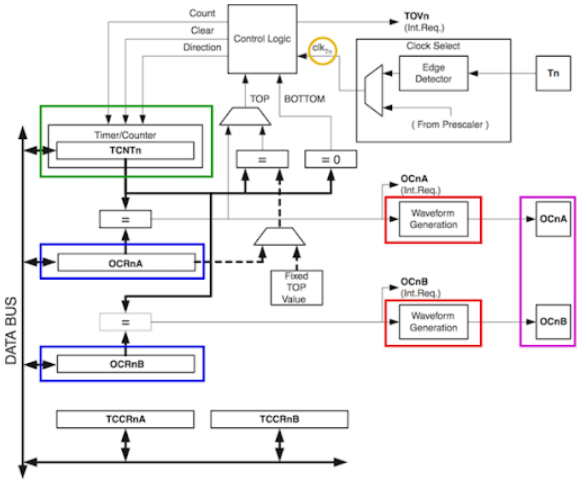
After understanding how the double algorithm function the next step is understanding where it is applied in serial printing. The American Standard Code for Information Interchange or ASCII represent the binary values used to print characters. In this case of trying to print a 4 digital decimal value requires the number characters. From decimal 48 to 57. The way to converter a BCD to ASCII is simply adding on the different between a decimal value and an ASCII value. Meaning taking a BCD converted value from the double dabble algorithm and adding on 48 will produce the ASCII digit equivalent.

Procedure

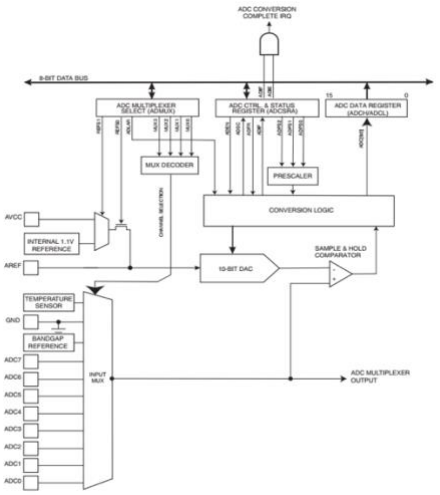
Parts Table	
Quantity	Description
1	ATmega328p
1	10 kΩ Potentiometer

The process behind the double dabble project is broken into three important steps. Firstly, the ADC converts an analog value between 0-5V to a digital value routinely through Timer1 interrupts. After, it takes the generated 8-bit analog value and converts the binary value to binary coded decimal (BCD). Finally, the BCD value is converted into an ASCII value and then uploaded into the Arduino's serial monitor. The parts needed for this operation are only a ATmega328p microcontroller and a 10 kΩ potentiometer to create an analog value. The software portion can be broken into three sections being Timer1 setup, ADC setup and UART transmission.

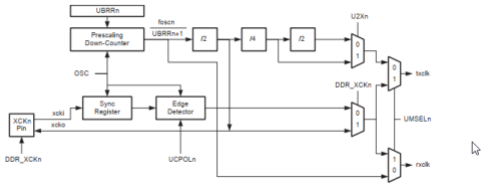
The first portion of the code to be examined is the Timer1 setup. The Atmega328P has three built in timers being Timer0 and Timer2 which are 8-bit timers and then Timer1 which is a 16-bit timer. In the case of using the Timer for an interrupt service routine the Timer1 will be used. To configure Timer1 to function with the ADC a frequency of around 125kHz is ideal. The best way to get this frequency is by setting Timer1 to normal mode and then pre scaling the timer correctly. Setting the Timer to normal mode is done by `sts TCCR1A, util` where util has the value 0x00 loaded. Next util loads the prescaler that divides the clock frequency by 64 putting it in a good range for the ADC. Next the statement `sts TCNT1H/L` sets at what point the Timer1 overflow flag is set. This creates the ideal Timer1 for an interrupt routine to convert an analog value to digital. Lastly, every time the timer overflows the vector sets the same prescales the TCNT1H/L set.



Following the Timer1 setup the most important setup code is written being the ADC setup. The first line in this code is taking the precaution of disabling all digital input pins on Port C being the analog pins with the line `sts DIDR0, util.` The next 4 lines set other important pre scalers necessary in making the ADC routinely interrupt and read A5. The following lines load pre scalers for the voltage reference `ldi util, ADCAVCC.` Next sets A5 to be the pins reading an analog value `ori util, ADC5.` Lastly, setting the output from the ADC to be left adjusted putting the lowest bits into bins 6 and 7 of ADCL which is very helpful `ori util, (1<<ADLAR).` The same loading and adding to an ADC register is done again for the enable bit ADEN, the start conversion bit ADSC and the timer pre scaler of 128 to get the ideal frequency of 125kHz. Next the ADIF flag is cleared by writing a 1 to it and then setting the ADC to be externally triggered. Next it sets Timer1 OVF to be the trigger source for the ADC.



Lastly, the UART needs to be setup to display values placed in the register through the TXCHAR function. This done by setting the Baud prescaler decided in the code. In this case it is 9600 so the UART Baud Rate register is equal to $\left(\frac{16{,}000{,}000}{3\times9600}\right) - 1 = 103$. So 103 is loaded into UBRR. Next the lines `ldi util,(1<<RXEN0) | (1<<TXEN0)` and `ldi util, ASYNCHRONOUS | CHAR_SIZE8 | PARITY_NONE | STOP_BITS1` Set receiving and transmitting enabled along with normal function for the UART respectly.



```
DoubleDabble:
  lds r16, ADCL   ;load bin6 and 7
  lds r17, ADCH   ;load all bins
  clr r18         ;clr register
  clr r19
  ldi r20, 10     ;loop counter; 10 shifts
first:
  mov r21, r19  ;Doubledabble thousands
  swap r21      ;move high nibble to low nibble
  andi r21, 0x0F  ;bit mask
  cpi r21, 5  ;check if BCD val >5
  brlo second ;if lower do not add three
  subi r21, -3  ;add three
  swap r21      ;move back to high nibble
  andi r21, 0xF0    ;make sure nothing interfers with lower
  andi r19, 0x0F
  or   r19, r21     ;add together
second:
  mov r21, r19  ;Doubledabble hundreds
  andi r21, 0x0F  ;bit mask
  cpi r21, 5      ;check if BCD val >5
  brlo third      ;if lower do not add three
  subi r19, -3    ;add three
third:
  mov r21, r18  ;Doubledabble tens
  swap r21      ;move high nibble to low nibble
  andi r21, 0x0F  ;bit mask
  cpi r21, 5      ;check if BCD val >5
  brlo fourth   ;if lower do not add three
  subi r21, -3  ;add three
  swap r21      ;move back to high nibble
  andi r21, 0xF0  ;make sure nothing interfers with lower
  andi r18, 0x0F
  or   r18, r21   ;add together
fourth:
  mov r21, r18  ;Doubledabble ones
  andi r21, 0x0F  ;bit mask
  cpi r21, 5      ;check if BCD val >5
  brlo shift      ;if lower do not add three
  subi r18, -3    ;add three
shift:
  lsl r16   ;shift entire value onces
  rol r17
  rol r18
  rol r19
  dec r20   ;decrement after 1 shift
  brne first
  ret
```

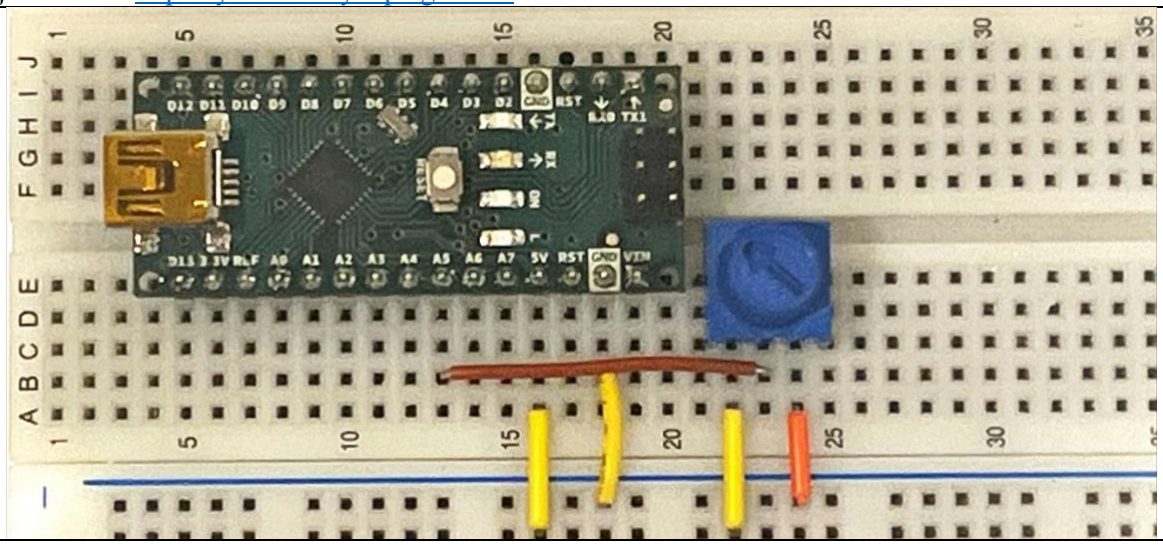
After setting up the Timer1 and ADC the final step in the AVR double project is displaying this code. The process for doing this involves taking the 10-bit analog value from ADCH and ADCL and moving it to general purpose registers. This is done at the beginning of the double double function `lds r16, ADCL` and `lds r17, ADCH.` These two lines bring the high 8-bits from ADCH and the lowest two bits from ACDL. Next the double dabble algorithm begins. The process for which is equivalent for the low nibbles of r18 and r19 along with equivalent the high nibbles. For the high nibble the register is transferred to a general register then switches places with the low nibble, `swap r21.` Next the value is bit masked and checking if it greater than 5 or not. The line `brlo second` says if the value is lower than 5 the next part of the function can be skipped but if it does not set the C flag high then the line `subi r21, -3` is excuted adding three to the function. The process is then completed three more times for the three other nibbles that make up the BCD until 10 shifts have occurred and the register loaded with the value 10; r27 goes to zero.

```
TXCHAR:
  lds   util,UCSR0A       ;determine if the transmit buffer is empty
  sbrs  util,UDRE0        ;if the flag is set, exit the loop
  rjmp  .-8               ;loop back to the previous statement
  sts   UDR0, r24         ;transmit data in r24
  ret
```

Now the BCD is in two registers and ready to be sent. The function display is the only other function part of the ADC OVF Vector. This iterates through the same 5 steps 4 times being moving the BCD value into register 24 `mov r24,r19`. Then swapping the high nibble with the low nibble (this is only done for the high nibble parts). Next a bit mask is done before the statement, `add r24,offset`, adds the difference between decimal zero and ASCII zero to convert the vale to ASCII. Finally the function TXCHAR is called display the ASCII value. The TXCHAR function checks if the transmit buffer is empty with the line `sbrs util,UDRE0`. Before loading the contents of r24 into the UDR0 the TXCHAR register.

Media

Project Video: https://youtu.be/4ylHpOgOwAk



Double Dabble pinout

Code

Github: https://github.com/MaxZischka1/AVR-Assembly-Double-Dabble

Reflection

Another bittersweet feeling as I finished my second last DER. Looking at the 150 pages of work I have completed, I see my skill set growing more and more. The assembly language projects are no exception to this. It is a real gift to be at the bedrock of coding, understanding every bit of the microcontroller I use and how to manipulate it. As I have said before, one of the reasons I took this course in grade 10 was to understand how the devices around us worked. Looking into the data sheets and seeing the schematics is one of the most interesting ways to learn how the devices work. The coding on top has been something I find more interesting the more I do it. It is very different from hardware because it is a lot less gruelling work and a lot more critical thinking.

Project 3.7 4517 Gray Code

Purpose

The purpose of the 4517 Gray code is to display a 4-bit Gray code value on a seven segment display using the minimum number of logic gates by Karnaugh mapping the inputs to the outputs.

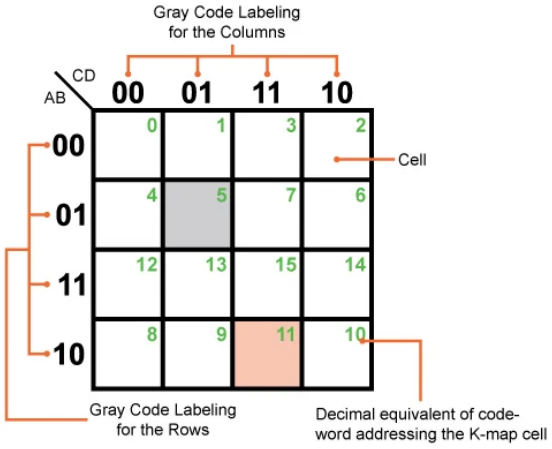
Reference

Project Description: http://darcy.rsgc.on.ca/ACES/TEI4M/Tasks.html#4517G
Digital Design: https://3rd.fcis2023.me/material/Books/DD.pdf

Theory

Law/Theorem	Law of Addition	Law of Multiplication
Identity Law	$x + 0 = x$	$x \cdot 1 = x$
Complement Law	$x + x' = 1$	$x \cdot x' = 0$
Idempotent Law	$x + x = x$	$x \cdot x = x$
Dominant Law	$x + 1 = 1$	$x \cdot 0 = 0$
Involution Law	$(x')' = x$	
Commutative Law	$x + y = y + x$	$x \cdot y = y \cdot x$
Associative Law	$x+(y+z) = (x+y)+z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Distributive Law	$x \cdot (y+z) = x \cdot y+x \cdot z$	$x+y \cdot z = (x+y) \cdot (x+z)$
Demorgan's Law	$(x+y)' = x' \cdot y'$	$(x \cdot y)' = x' + y'$
Absorption Law	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$

Boolean algebra is an approach used in computer science linking mathematics to circuit design. In this branch only two variables are used being 1's and 0's and only two expressions are used being OR and AND. OR is a value that can be compared to the addition operation for several reason. The easiest reason to understand why is because of the natural number of the OR operation. The natural number in this case is a 0 because any value ORed with a 0 stayed the same value. This is the same rule for non-Boolean algebra with additions natural number. So as a result, the operation used to express OR is +. For AND operations the natural number is 1. If you AND any variable with one is stays the same variable so 1 is its natural number. Therefore, AND operations are expressed like multiplication operations. More broadly what Boolean algebra allows people to do is use postulates and theorems on expression that express circuit to alter the circuits and usually minimize.

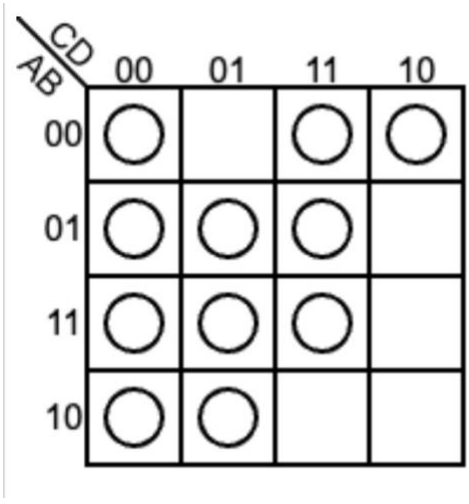


From understanding how Boolean algebra works the states of certain functions can be graphed on square layout. This layout gives each box defining the axis a different set of the variables. These variables are called the min-terms of a variable functions the min-terms will begin from the value zero where every variable is NOTed to the largest value where every variable is not NOTed. Min-terms are a useful way of showing 1's and 0's in terms of variables. Then from this layout described and shown on the side an approach to finding the most simplistic version of mapping the inputs to the outputs can be found using an approach called Karnaugh mapping. Karnaugh mapping was invented by Maurice Karnaugh and mapping each common variables to each other based on the idea that if this map shown uses a type of numbering known as grey code. When numbers are close together the number of variables differing the two values will only be one. The same rule about number of variables differing lowers are the size of the square increases. This approach will provide the most efficient circuitry in mapping inputs to outputs due the singular cell difference being detected allows for all min-term combinations to simplified. For more sense on this topic it will be discussed below in the procedure.

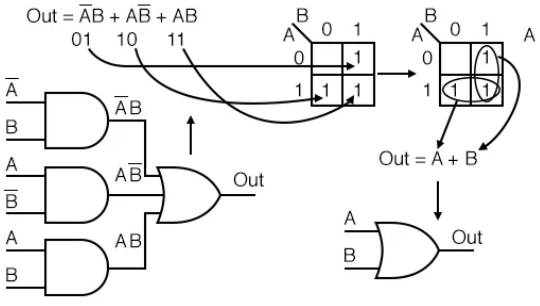
Procedure

Parts Table	
Quantity	Description
1	Gray Code Encoder
1	7-Segment Display
1	CD4069(NOT)
7	CD4081(AND
6	CD4071(OR)

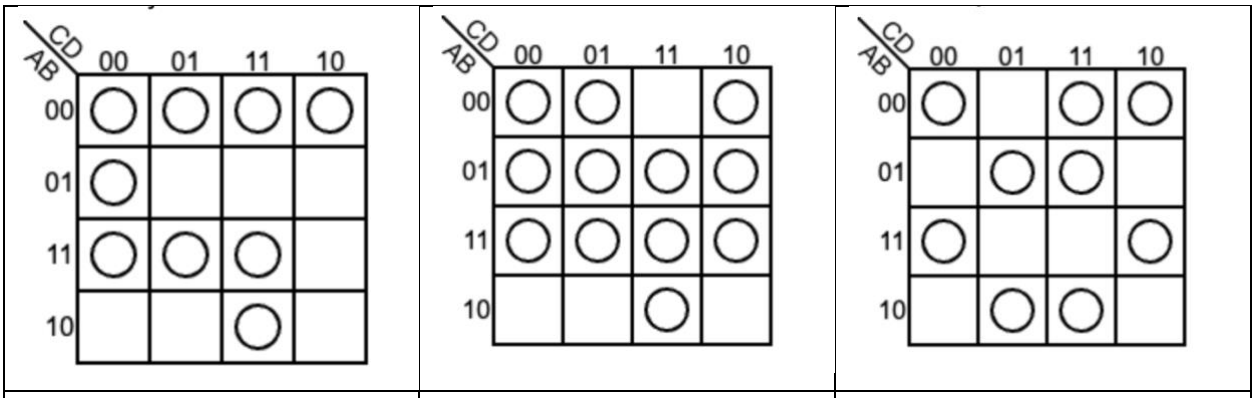
The parts used to build this circuit are only three for the mapping being the three logic gates used for Boolean expressions. This is again helpful in simplicity. The Gray Code Encoder is a device continuously counting upwards in Gray Code from 0 to 15. The approach taken to complete this task of displaying the Gray Code value on the display began with Karnaugh mapping the inputs to the outputs. From having all the Boolean expressions for the project one more step was required which was simplifying variables across the function. For example, if the same expression $AB + A'C$ was used in two different functions. Only wiring this once would save 4 logic gates. This approach makes it possible to map the inputs to outputs using only 14 logic ICs.



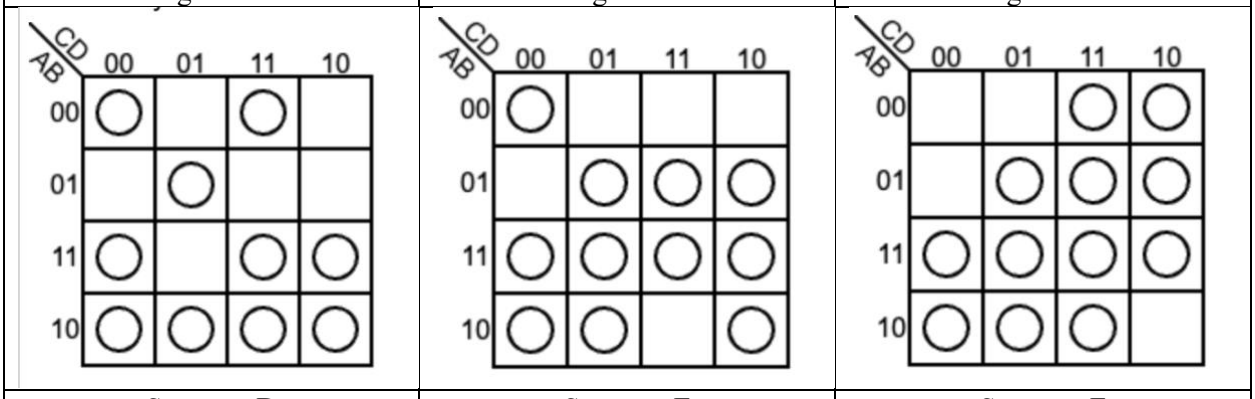
For the Karnaugh mapping tables will be used to display the Karnaugh mapping approach done and highlights will display what values matching (highlight and underline means an overlap). The first step in Karnaugh mapping is setting up the map. This starts with understanding how gray code values work. Gray code is in numbering system that never has two variables change at the same time so this means gray code will over 00, 01, 11, 10. A different numbering system than binary. Once this part is understood next is graphing the zeros and ones. A zero should be placed in the gray code cell if that gray code number when displayed on a seven-segment display does not appear for that value. Now with this simple approach down the first example can be shown which is the A segment on the seven-segment display. On the Karnaugh map shown on the right every 0 is shown as NOT for its variable respectively and the circles represent ones. The approach used for Karnaugh mapping for example here is first looking for groups of 8, a group of 8 means only one variable will be used, then groups of 4 with two variables then groups of two which use one less variable. The groups of 4 create the expressions $AC', A'CD, BD, BC'$ *and* $A'B'C'$. This approach can be used again for the next 6 segments to create the mapping for the entire circuit.



The next part after all expression have been created in minimizing the repeating functions. The first pattern to look over are the broader and most efficient patterns to catch. This includes three and 4 variable AND expressions along with patterns between to AND function to minimize OR expressions. The reasoning behind these two being done first is because how the expressions are constructed in the circuit changed based on what variable are repeated. For example, because two input AND gates are used if the expression A'BC appears and a long pattern of BC expressions appears across several patterns then it is smart to construct the BC expression first then the A' expression.



Segment B	Segment C	Segment D



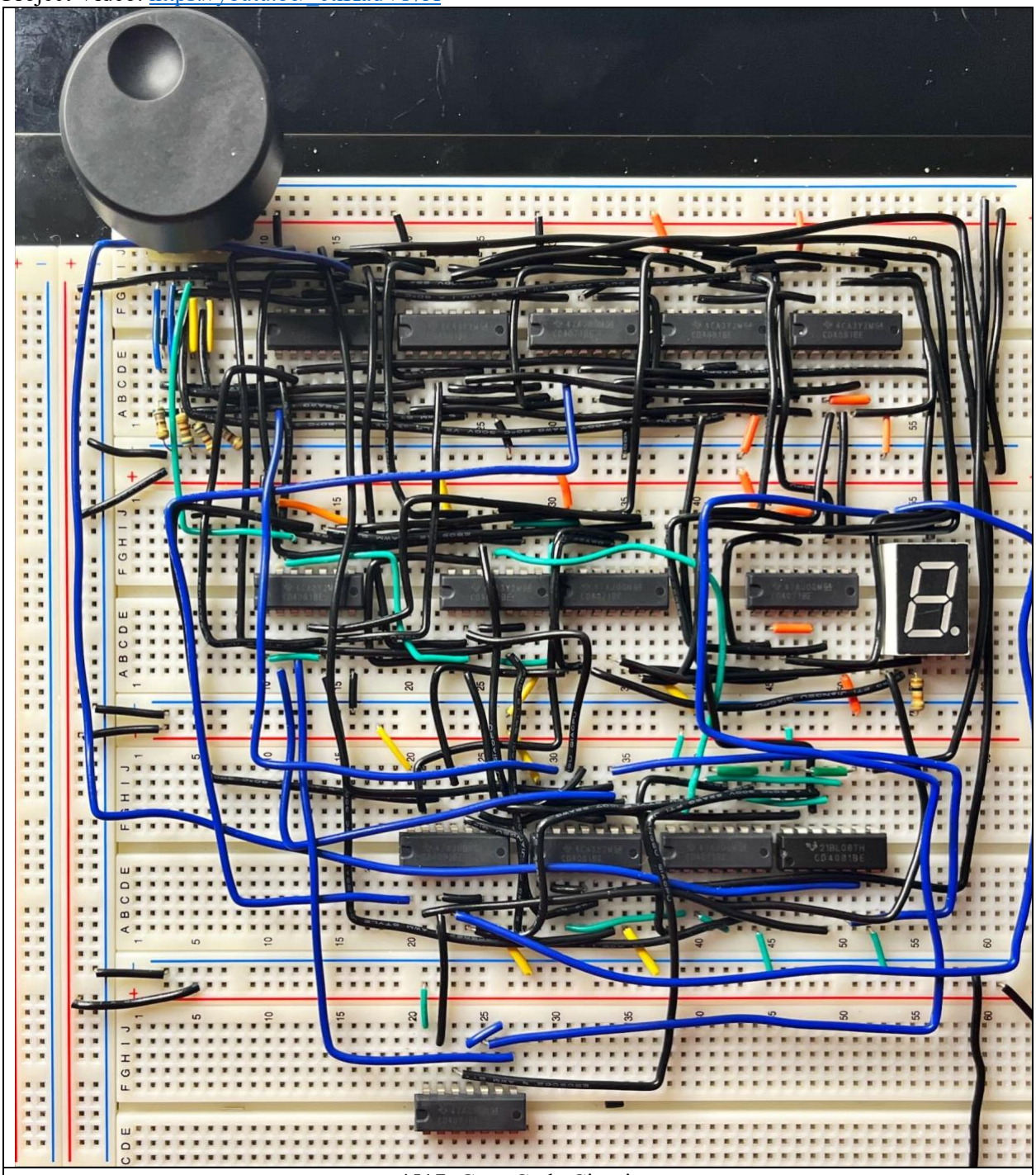
Segment D	Segment E	Segment F

The approach used for circuit minimization was putting all expressions in an excel sheet and making each cell an AND operation and each separation an OR operation. The first step was finding patterns between functions. These include the yellow, blue, navy, orange, green and grey highlights used to show the matching OR operations between functions. The patterns alone save several chips especially on segment F where instead of 16 logic chips the patterns found from other patterns minimize it to one. The next step is finding patterns between AND expressions usually two variables. These are outlined using the text colours found across the entire page. Each highlight shows how a single AND gate is minimized. Lastly, all single variables do not need to be considered because all inputs are wired to same encoder but to save gates on NOT expressions only one gate is used per variable.

A Segmet	B Segment	C Segment	D Segment	E' Segment	F Segment	G Segment
B'A'	D'C'	C	BC'	A'D	B'D	AB
A'C'	A'D'	D'A'	D'C'A'	DC'	AC	DB'
B'D	B'CD	D'B'	ACD'	BD	BC	AC
AC	ABD	ABD	DCA'	AB'CD'	A'D	BC
C'D'B			C'DA	ABC'	A'B'C'	D'B
				A'B'C'		

Media

Project Video: https://youtu.be/_ctiKadVs7A



4517: Gray Code Circuit

Reflection

When I was in grade 10, I told myself engineering was not something I wanted to pursue. I told myself there were too many engineers in my family, and I was going to do something else. It wasn't until a little after March that I knew I was going to become an engineer because of my passion and interest in the ACES course. Over the past two and a half years, however, my second life in the DES has taught me so much more than just engineering. The biggest one that I truly use every day is that you get out whatever you put into something. In the ACES program, I am lucky enough to be encouraged to push myself beyond any bounds set every term, and every term, it is however much work I put in that I get to reap the rewards of. The hardware skillset the course has been able to teach me because of the work I have put it in is one of the most priceless things to me as I embark on the university experience. I do not think I would have had the opportunity to go to the university of my choice and study something as interesting as engineering if it were not for the belief, encouragement and work ethic the ACES program and Mr. D'Arcy have taught me over these two and a half years. Next year will definitely be a challenge, but the ACES program has taught me that the hardest path to travel is always the best option. Thank you very much to Mr. D'Arcy, previous TAs and peers for the opportunity I have been granted.