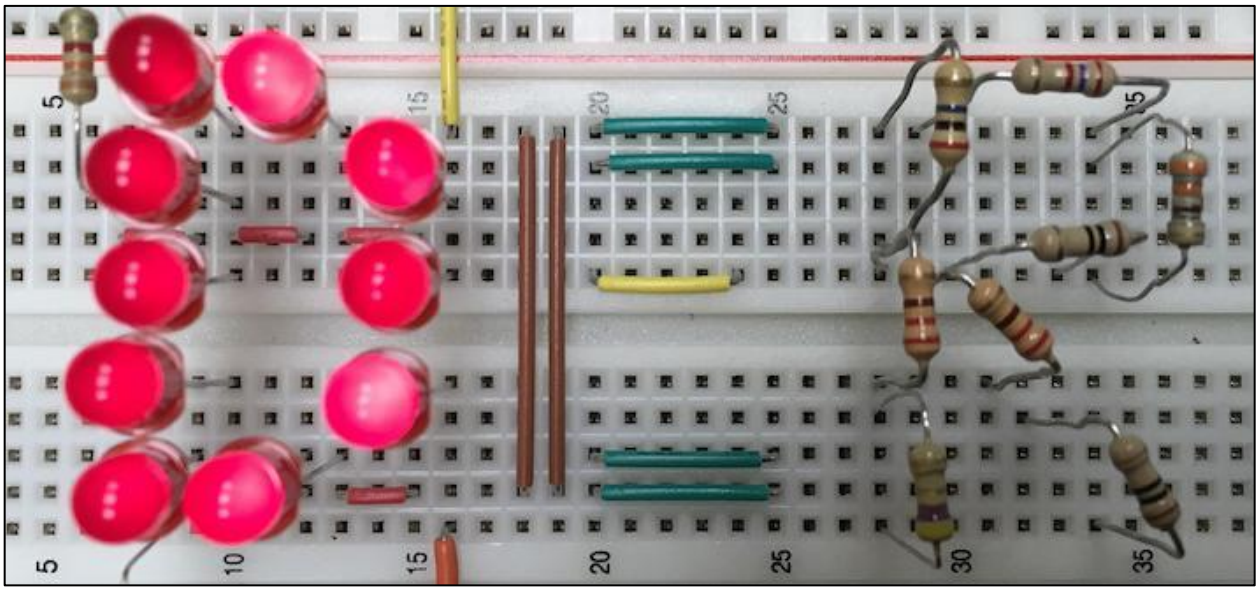
Design Engineering Report



Course:	Computer Engineering
Code:	ICS2O, ICS3U, ICS4U
Student:	Xander Chin
Date:	November 26, 2022

Table of Contents

ICS2O

DC Circuits

Project 1.1: Voltage H-Bridge

Purpose

The purpose of this project is to develop the concept of using a potentiometer to divide voltage, therefore changing the direction of current.

Reference

https://learn.ssparkfun.com/tutorials/voltage-dividers/all
http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#VoltageHBridge

Procedure

The concept of the voltage divider used in the circuit is simple to understand. It creates resistance on one side, thus restricting current to flow in only one direction because of Kirchhoff's law. In other uses, the voltage divider can be used to control voltage using the formula:

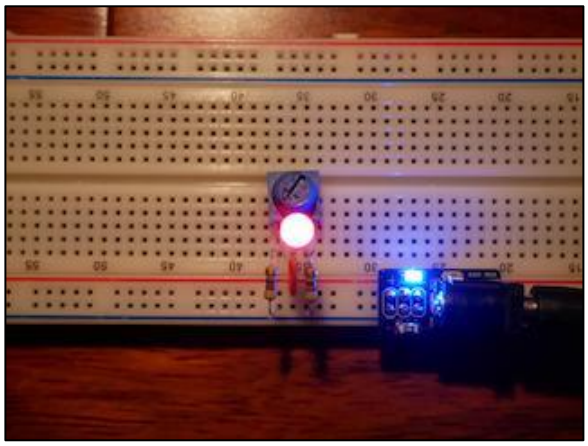
$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
1	10 kΩ potentiometer
2	470 Ω fixed resistor
1	5 mm bicolor LED
1	Breadboard+

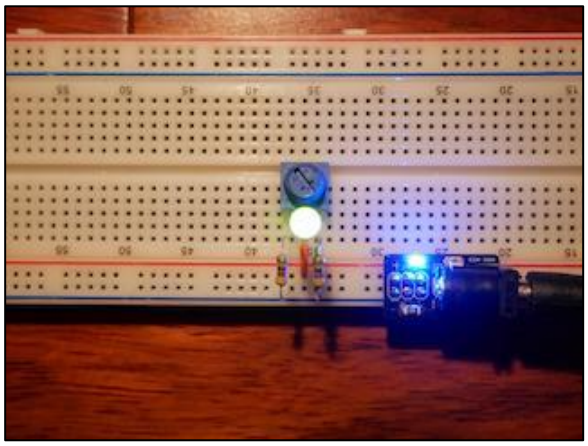
There are two main components of this circuit, the potentiometer and the bicolor LED. The potentiometer has three leads, the A, B and C lead. The A and B leads are located on the edges while the C lead is in the middle. In the formula above, R_2 represents the total resistance between the A and B lead and R_1 represents the total resistance of the A to C lead on the potentiometer. Using only the A and B leads in a circuit transforms the potentiometer into a fixed resistor, while incorporating the C lead would turn it into a variable resistor which controls the resistance between A and C as well as B and C. When current travels through the C lead, it can flow from A to B or from B to A. Since Kirchhoff's voltage law states that current flows from a high to low voltage, it will flow through the high voltage lead (the one with less resistance) to the low voltage lead (the one with higher resistance). When the potentiometer is positioned in the middle, current cannot flow because both leads have the same voltage and therefore cannot travel from a high to low potential. The bicolor LED is the other main component. It contains two LEDs arranged in opposite directions so that current can only flow though one LED. This way, the direction of current is shown by the color of the LED.

In the voltage H-bridge, the potentiometer is used as variable resistor since the C lead is being utilized. It can now control the direction of current to light up the red or green LED. When the red LED is on, current is moving from the left to the right of the LED. When the green LED is on, current is moving from the right to the left of the LED.

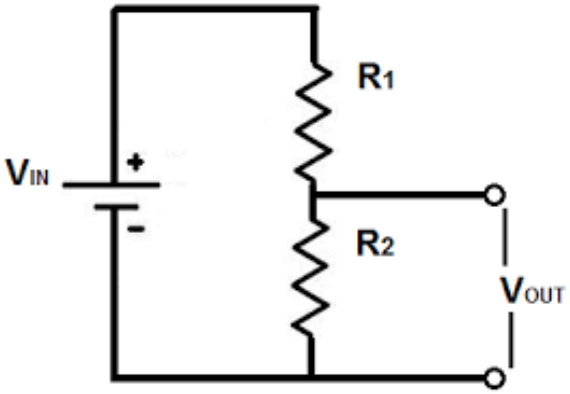
Media



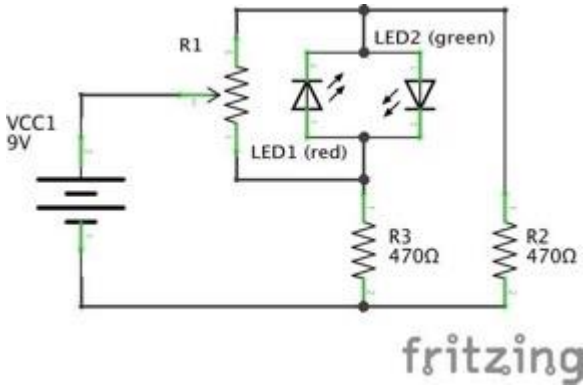
Current flows from left to right of the LED, turning on the red side



Current flows from right to left of the LED, turning on the green side



Schematic of a voltage divider



Schematic of the voltage H-bridge

YouTube video link: https://youtu.be/9rcGfo8x2Zw

Reflection

This concept of constructing a voltage H-bridge on a breadboard was very hard because it was difficult to understand. I had trouble building it and frequently asked for help from my peers. Eventually, I built the circuit correctly but still did not fully understand it. This impacted the time management of my DER as I had to rewrite it several times and replace diagrams with new and improved versions. However, I finally grasped the correct concept and used it to create my first project. It took a long time, mainly because I had not done it before. However, this will change in the future because I will strive to improve and plan for every DER that I will make in the history of this course.

Project 1.2: The Capacitor Visualizer

Purpose

The purpose of this project is to demonstrate the use of capacitors as timers by discharging and charging them.

Reference

http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#capacitor

Procedure

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
2	Momentary PBNO
1	1000 µF capacitor
3	470 Ω fixed resistor
1	5 mm yellow LED
1	5 mm bicolor LED
1	Breadboard+
1	Capacitor visualizer PCB
1	Female barrel jack
1	Capacitor visualizer case

The Capacitor Visualizer is a circuit that charges a capacitor and releases its energy for a set amount of time. Basically, the capacitor visualizer can be used as a timer and a short-term power source.

The push button and the capacitor are the two new parts introduced in this circuit. The push button can come in two forms. One can be a push button normally open (PBNO) and the other can be a push button normally closed (PBNC). When input is applied to the PBNO, the circuit closes and current flows. However, when input is applied to the PBNC, the circuit opens and current cannot flow. In the capacitor visualizer, PBNO's are used. After the push buttons comes the capacitor. To put it simply, capacitors have two plates that induce a charge on one another until the first plate is mostly charged up with electrons. Once this happens, resistance increases and current slows down. Capacitors also introduce two new units called the farad (F) and the tau (τ). The farad is the capacitance or how much charge a capacitor can hold and the tau is a measurement of time used to determine the charging time of capacitors. One tau is equal to the time it takes for a capacitor to charge to 63% of the source voltage and two taus is equal to the time it takes for a capacitor to charge to 63% of its remaining source voltage after the first tau. After 5 taus, the capacitor is essentially filled. In order to find a single tau of a capacitor, this equation is used (τ is in seconds, R is in Ω, and C is in F):

$$\tau = R \bullet C$$

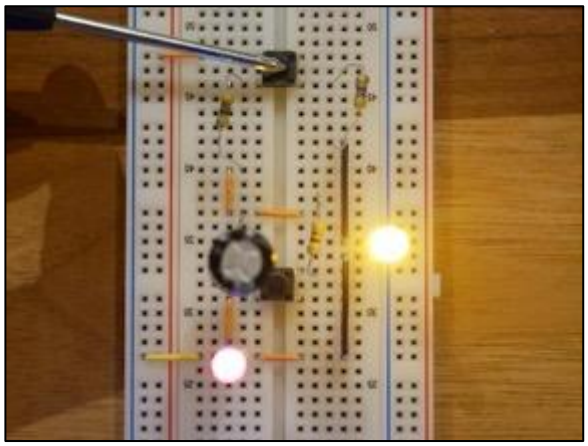
To acquire 5 taus or the total charging time of the capacitor, simply multiply the answer by 5. As a means of proving the equation accurate, a test was conducted with different resistor-capacitor pairs and the results somewhat proved to line up with the equation (the battery source voltage measured 8.71V while doing the tests).

Resistance (Ω)	Capacitance (μF)	Theoretical 5τ (s)	Observed 5τ (s)
1000000	100	500	540 (to 6V)
100000	100	50	120 (to 7V)
470	1000	2.35	12 (to 7.65V)

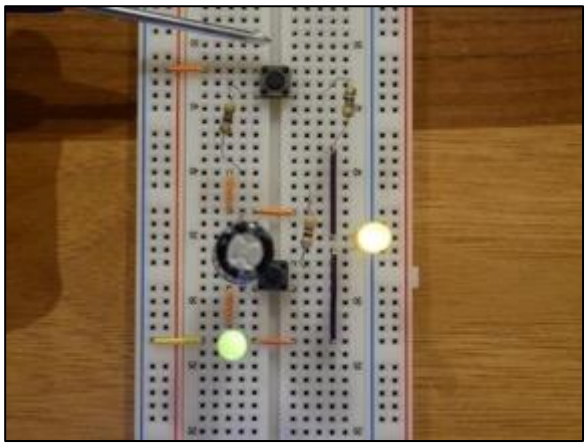
The circuit works by charging and discharging the capacitor using buttons to light up the bicolor LED. When input is applied to button 1, current flows clockwise turning on the red and yellow LED as the output. The red LED then dims slowly because the capacitor is filling up. Once input is released from button 1, current flows counter clockwise, releasing the built-up charge in the capacitor and turning on the green and yellow LED as the output for a brief moment. When input is applied to button 2, the capacitor drains quickly through the resistor, turning off the green and yellow LED as the output. If input is given to both buttons, current flows clockwise and the capacitor fills up halfway. Then it travels through button 2 and the red and yellow LEDs turn on without dimming.

The capacitor visualizer was completed in three sections. First, the breadboard prototype was built using the parts listed in the first table. Once the prototype worked, a custom-made PCB was handed out to solder on the components. After everything was soldered on and it functioned properly, 3D printed cases were given out to wrap up the project in a neat and great-looking way.

Media



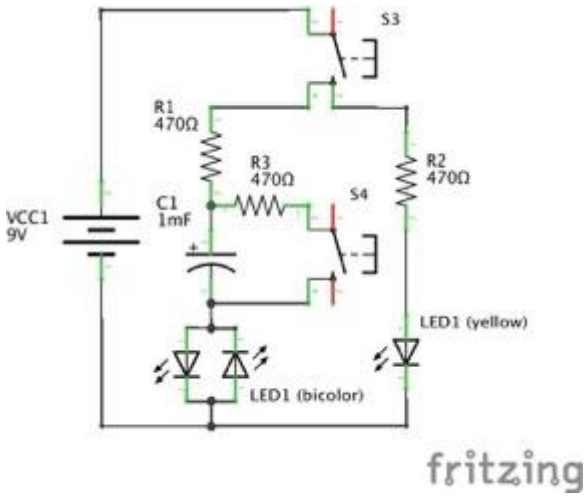
Current flows from positive to negative from the capacitor turning on the red side (breadboard)



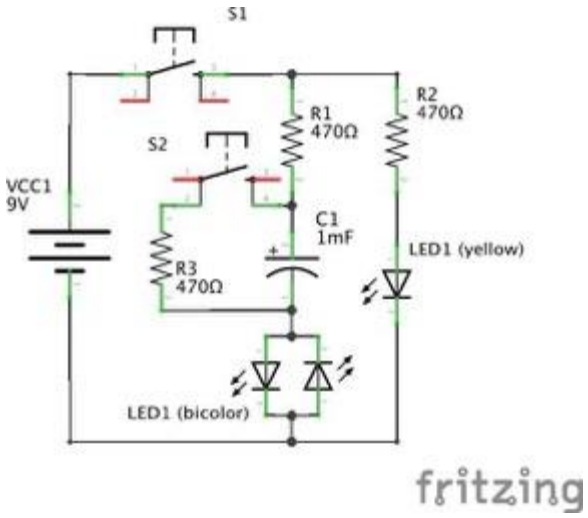
Current flows from negative to positive from the capacitor turning on the green side (breadboard)

Current flows from positive to negative filling up the capacitor and dimming the red LED (PCB)

The capacitor acts like a battery and drains its charge thorough the green LED (PCB)



Schematic of the capacitor visualizer breadboard



Schematic of the capacitor visualizer PCB

YouTube video link: https://youtu.be/Y5R257CdEeA

Reflection

Compared to the voltage H-bridge, the capacitor visualizer was a lot easier to understand. I was off to a much better start and I was given a lot more time. In addition, I had a little more experience of how hardware class works from the voltage H-bridge and so I knew how to plan out my time effectively and finish it early. All went very well until I had to make the final product. I ran into a multitude of problems from assembling it the wrong way to the screw holes in the case not lining up. Because of this, I had to desolder my wires twice and then move the screw holes in the case using a soldering iron. Luckily, through hard work and patience, my project was saved. Overall, this whole turn of events taught me many lessons. Firstly, it taught me to plan, visualize and think about every step carefully. Secondly, it taught me the grit and perseverance to see a project through even when times are difficult. Lastly, it taught me that failure is experience and its ok to fail because you will always learn. For example, because of the whole fiasco, I have become an expert at desoldering and solder sucking, which will help me or others in the future when they mess up something in their projects. All in all, this was a great second project that I will learn from in the oncoming days of hardware.

Project 1.3: The Analog Oscillator (aka the Astable Multivibrator)

Purpose

The purpose of this circuit is to demonstrate the use of resistor-capacitor pairs in order to automate transistors for blinking LEDs.

Reference

http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#AnalogOscillator
http://tinyurl.com/yf4ajl88
http://tinyurl.com/yjxotwmn

Theory

In order for the analog oscillator to work, resistor-capacitor pairs are arranged in series that connect to the base pin of a transistor. The capacitor-resistor pairs take time to charge, allowing the transistor to be unsaturated after a certain time period because the base pin of a transistor has a certain voltage threshold between saturation and unsaturation. The capacitor then discharges and the pattern start over again. This produces a uniform wave or a duty cycle that repeats infinitely.

Procedure

The analog oscillator is a circuit that emits alternating light from a pair of LEDs. Basically, the transistor is used as an automated switch when current alternates in a set amount of time from the resistors and the capacitors. The transistor is the new part implemented into this circuit. It plays a crucial role in the automation of the circuit by using voltage to close or open it. The transistor has three leads. The emitter lead, the base lead, and the collector lead. When input is applied to the base pin in the form of voltage, the collector and emitter lead close or open, depending on the type of transistor which can come in two forms. A PNP transistor opens the collector and emitter lead when current flows and an NPN transistor closes them when current flows. These types can also be compared to the two types of push buttons, just with different inputs. A push button requires manual pressure to activate, while a transistor requires electricity and thus does not rely on human input; this changed the history of electronics forever. A PNP is like a push button normally closed (PBNC) since input opens the circuit and an NPN is like a push button normally open (PBNO) because input closes it.

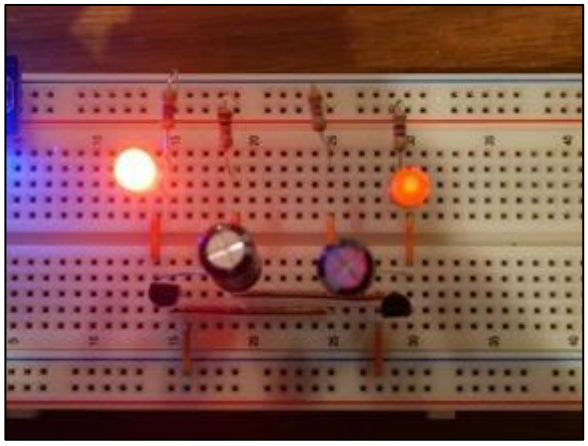
Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
2	100 µF capacitor
2	470 Ω fixed resistor
2	10 kΩ fixed resistor
2	5 mm red LED
2	5 mm blue LED
2	2N3904 NPN transistor
1	Breadboard+
1	Analog oscillator PCB
1	Female barrel jack
1	Analog oscillator case

The other two crucial but familiar components are the capacitor and the resistor. Their ability to be used as timers as shown in the capacitor visualizer make them responsible for the length duration of the oscillation. When the resistance or capacitance is high, charging the capacitor takes longer and therefore, the rate of oscillation is slower. When the resistance or capacitance is low, charging the capacitor is shorter, speeding up the oscillation.

The analog oscillator circuit is a complicated circuit to understand. It functions by reversing current flow from the capacitor C2 discharging to the base pin of transistor Q1, reducing the potential difference below 0.65V which is the base-emitter cut-off voltage for both transistors. Once that happens, Q1 unsaturates and current from the collector to the emitter stops, turning off LED1. This causes current to charge through C1 and into the base of Q2, allowing current to flow from collector to emitter, turning on LED2 and allowing C2 to discharge. Once C1 charges completely, current reverses and it discharges through Q1, creating negative voltage across Q2, switching it to its cut off state. Now, C2 begins to charge, saturating Q1 and allowing LED1 to turn on and C1 to discharge. Once C2 fully charges, it begins to discharge and the pattern repeats indefinitely until the 9V battery is unplugged. If the rate of oscillation is too fast or slow for standards, resistance from R2 and R3 or capacitance from C1 and C2 would increase or decrease to slow down or speed up the rate of oscillation respectively.

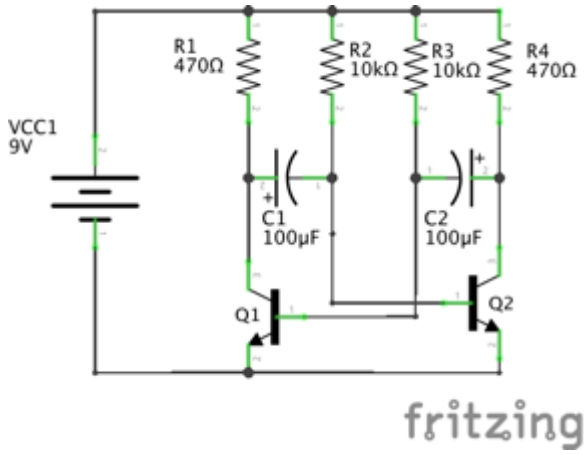
Theoretically, if the capacitors and resistors were exactly uniform, both transistors would saturate at the same time and no oscillation would occur but since there are biases to the components, only one will saturate at a time. Also, the schematic of the analog oscillator produces similar waves to a capacitor charging with a sharp cut-off instead of square waves. This is because current slows down when either C1 or C2 charges, leading to a fade out of LED1 or LED2 as shown in the capacitor visualizer. To produce a true digital square wave, both LEDs have to be connected to ground parallel to the capacitors so that the capacitors do not charge through the LEDs.

Media

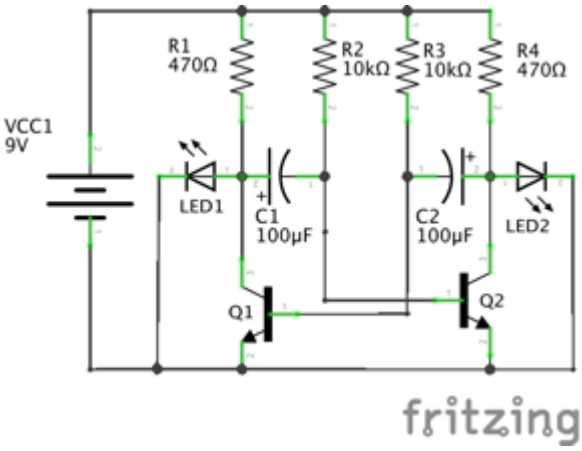


The capacitor charges with the LED, producing blinking and fading lights in a curved wave

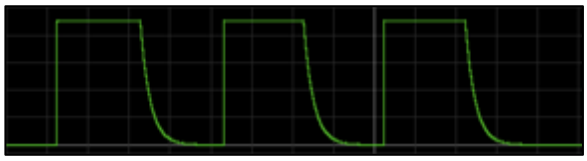
The capacitor charges without the LED, producing non-fading and blinking lights in a square wave



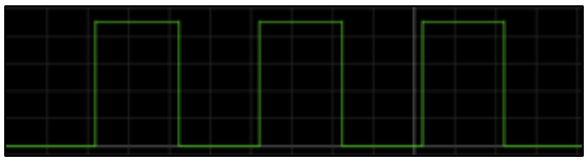
Schematic of the analog oscillator with fading LEDs (breadboard)



Schematic of the analog oscillator with non-fading LEDs (PCB)



The oscillation wave of an LED for the analog oscillator with fading LEDs (breadboard)



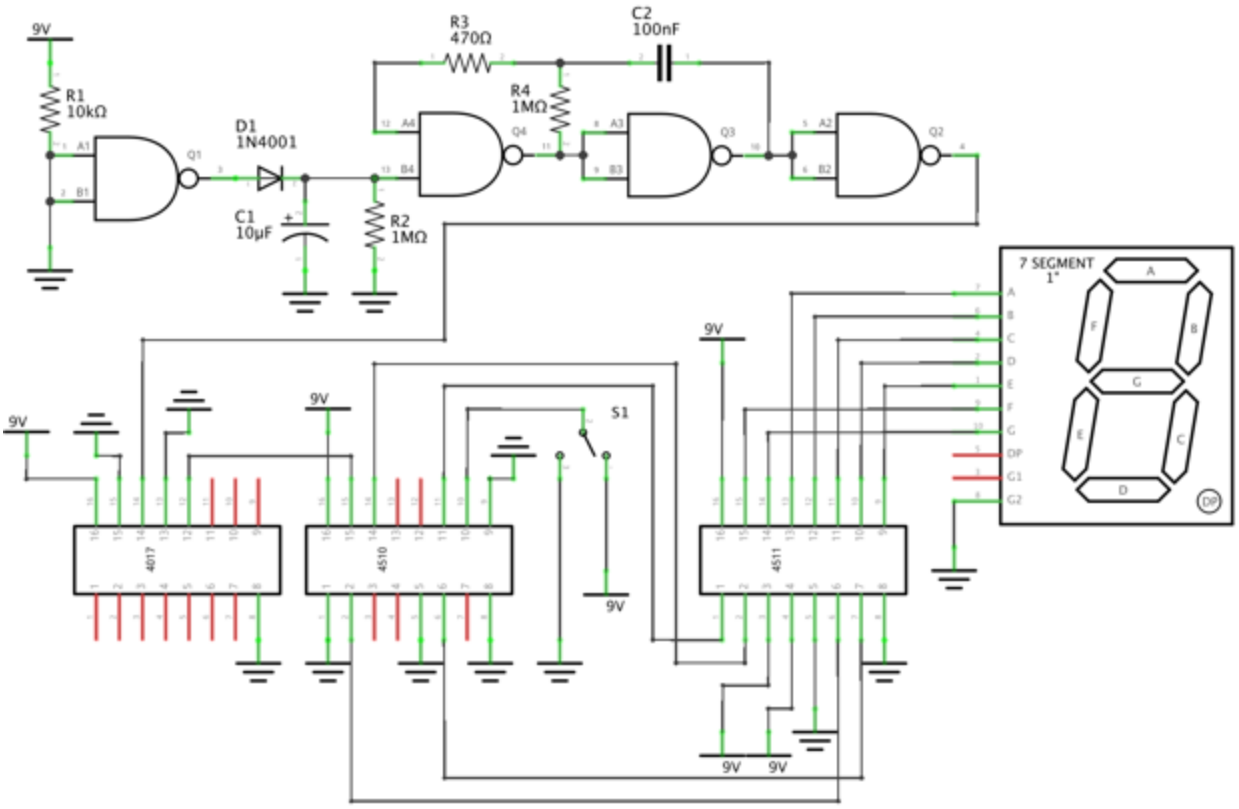
The oscillation wave of an LED for the analog oscillator with non-fading LEDs (PCB)

YouTube video link: https://youtu.be/o5JJ0ilo2Eo

Reflection

Overall, this circuit was a hard one for me to fully comprehend. I was not entirely sure exactly how it works but I now believe I understand the fundamentals of it. On a positive note, I managed my time quite well. I started the project early and was able to solder my PCB and attach it to the case without trouble. However, figuring out the circuit was the hard and time-consuming part. I had to change and revise my theories and explanations whenever something did not make sense in the Falstad circuit demonstration. Even when I did, I kept getting more confused but after a while, I finally understood the main concepts of the analog oscillator. In a way, this was beneficial to me because it taught me to think very deeply and logically about what is correct and what is wrong.

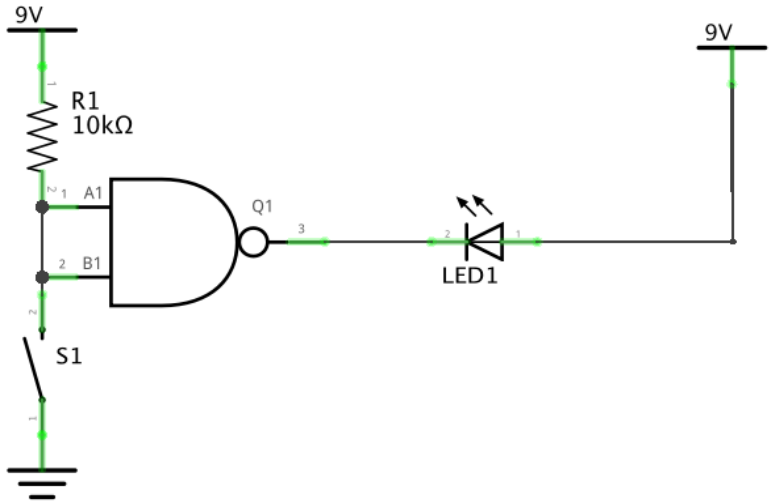
Project 1.4: The Counting Circuit



Theory

The counting circuit is one of the most basic fundamentals of digital electronics. In parts A and B, NAND logic is being used to take an analog signal and output it into a digital one. The output is a digital clock signal from an LED, resembling a square wave at a 50% duty cycle with its frequency and duration determined by the analog resistor-capacitor pairs. In the second part of the circuit, a plethora of integrated circuit or ICs for short are utilized to get the input of clock signals to be displayed on a 7-segment display. Firstly, the 4017 decade counter is used to slow down the clock signal from the NAND gate oscillator to $1/10^{th}$ of its frequency. This clock signal from the square wave is inputted into a 4510 binary up/down counter that counts from zero to nine in binary. After that, the binary signals are fed into a 4511 binary decimal decoder which displays its count on a 7-segment display. Altogether, these parts form the final counting circuit.

Part A: Analog Input



Purpose

The purpose of this circuit is to get an introduction with the basic fundamentals of NAND logic using the NAND gate and a pull-up resistor configuration with a visible output.

Reference

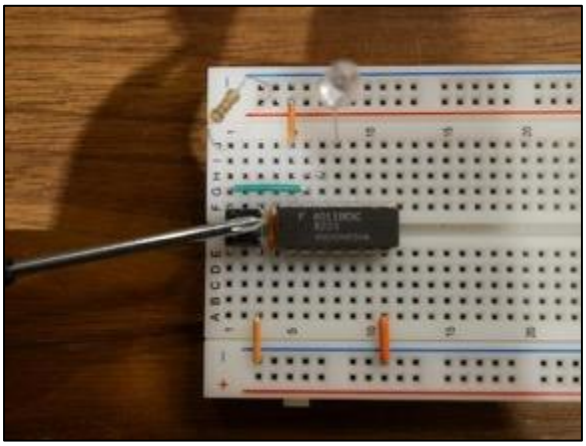
http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#counting

Procedure

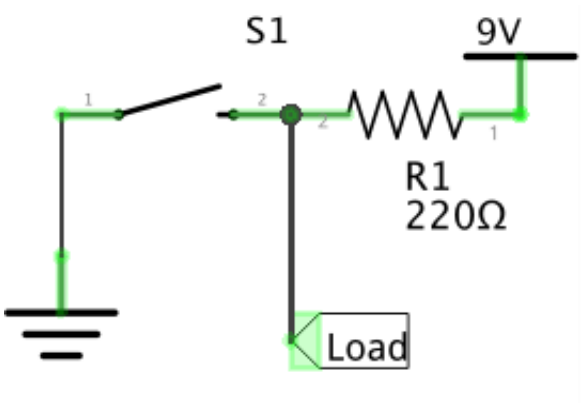
Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
1	10 kΩ fixed resistor
1	Momentary PBNO
1	5 mm red LED
1	4011 NAND gate IC
1	Breadboard+

This main part in the circuit is the pull-up resistor configuration with its input feeding into a NAND gate. An LED is present with its cathode lead positioned at the output of the NAND gate and its anode lead positioned in the 9V power supply. At rest, the NAND gate outputs a low from two highs from the pull-up resistor. This allows current to flow from the anode to the cathode, turning the LED on. When input is applied to the PBNO, current is grounded because of the pull up resistor configuration and it produces two lows, outputting a high out of the NAND gate and into the LED. When an LED experiences a high on both leads, current cannot flow from the cathode to the anode and the LED turns off. At rest in a pull-up resistor configuration, current flows can only flow one way from the 10kΩ resistor into the load. When the PBNO is pressed, current is grounded because that direct path to ground offers the least resistance.

Media

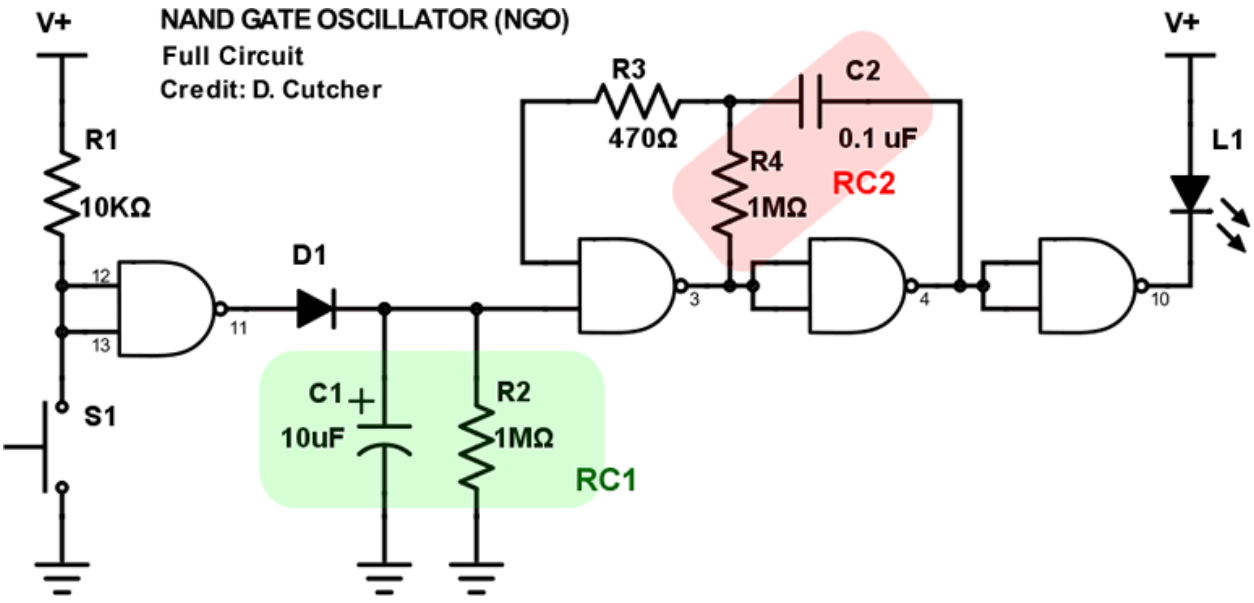


Current is grounded, inputting two lows then outputting a high



Pull-up resistor configuration

Part B: NAND Gate Oscillator (4011)



Purpose

The next stage of the counting circuit is the NAND gate oscillator, featuring the output of an oscillating LED at a certain frequency and duration using NAND logic. With the use of inputs from resistor-capacitor pairs and the forced digital outputs from NAND gates, a true square wave can be formed.

Reference

http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4011.pdf
https://www.semiconductorstore.com/blog/2018/What-Are-Clock-Signals-in-Digital-Circuits-and-How-Are-They-Produced-Symmetry-Blog/3322/
http://tinyurl.com/s6tfnop
http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#counting

Procedure

The basic functionality of a NAND gate consists of two inputs and its processing. Its inputs consist of a high or low, otherwise known as power or no power, and together with the processing, it can produce one output also of a high or low. The processing is explained fully in the truth table located to the right where 0 is low and 1 is high. Basically, if there is a low on either input lead then the output will be high and if both input leads experience a certain a high, the gate will output a low.

NAND Gate Logic		
Inputs		Output
0	0	1
0	1	1
1	0	1
1	1	0

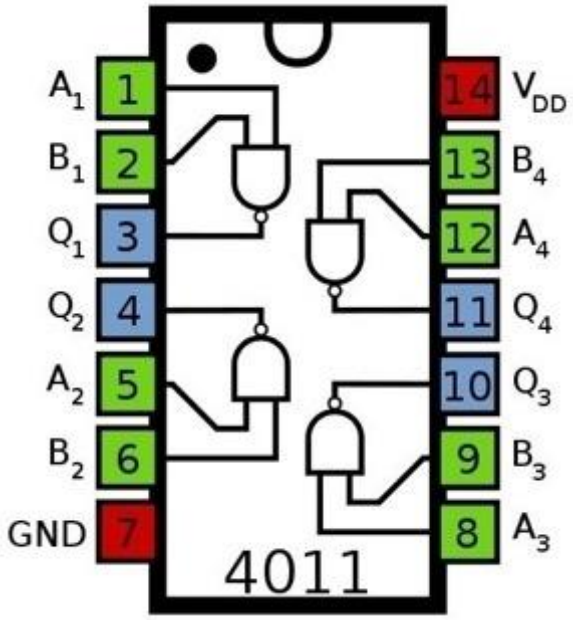
NAND logic is a type of digital logic that takes in two analog signals, processing them as digital, and outputting one digital signal of high or low. For a NAND gate to convert an analog signal into a digital one, it has to measure the strength of the analog input signal. If the analog input signal is 0 volts to less than ½ of the supply voltage, the NAND gate will process that as a low and if it is more than ½ of the source voltage, the NAND gate will process that as a high. This forms the concept of a NAND gate converting an analog input into a digital one.

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
1	10 kΩ fixed resistor
2	1 MΩ fixed resistor
1	470 Ω fixed resistor
1	Momentary PBNO
1	5 mm red LED
1	4011 NAND gate IC
1	Diode
1	10 µF capacitor
1	100 nF capacitor
1	Breadboard+

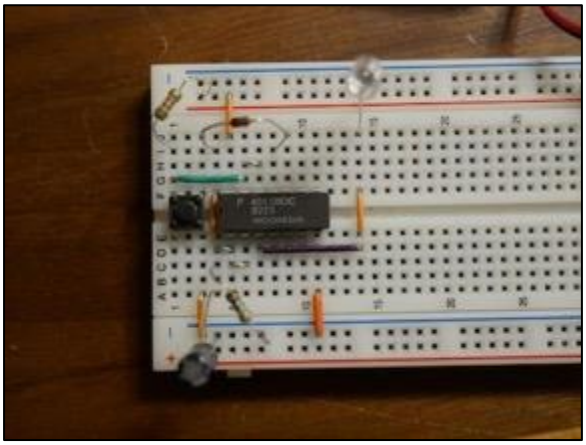
The signal that oscillates between high and low, also known as the clock signal, is the other crucial part of the circuit design. The clock signal can be compared to the tempo of a metronome, keeping the whole circuit in time. The clock signal can have different characteristics such as the time interval between each high and low, also known as frequency, and its duty cycle which controls the ratio of high and low. In the NAND gate oscillator, there are 2 resistor-capacitor pairs that are used as timers to control certain characteristics of the clock signal. RC1 controls the duration of the whole circuit by filling up the capacitor and using its charge to power the circuit for a finite amount of time. A diode is positioned before the capacitor to direct current towards the NAND gate instead of ground when the capacitor charges. RC2 controls the frequency, charging and discharging the capacitor as

shown before in the analog oscillator. The higher the capacitance or resistance in RC1, the longer the LED keeps on oscillating and the higher the capacitance or resistance in RC2, the lower the frequency of the oscillation as shown in the LED.

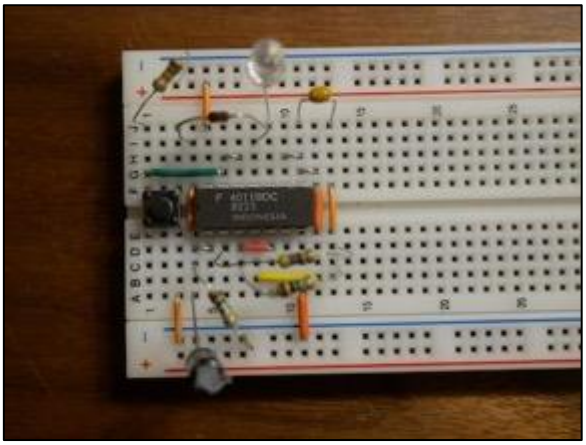
The main part featured in the circuit is the 4011 NAND gate IC. The IC contains four NAND gates arranged in a two by two formation with their outputs facing each other. This allows for the use of all four NAND gates as shown in the NAND gate oscillator. A detailed pinout diagram is shown to the right. In order to power the NAND gates, pin 14 must be connected to power and pin 7 to ground. This is how the chip powers its outputs.



Media



The NAND gate oscillator without the oscillation



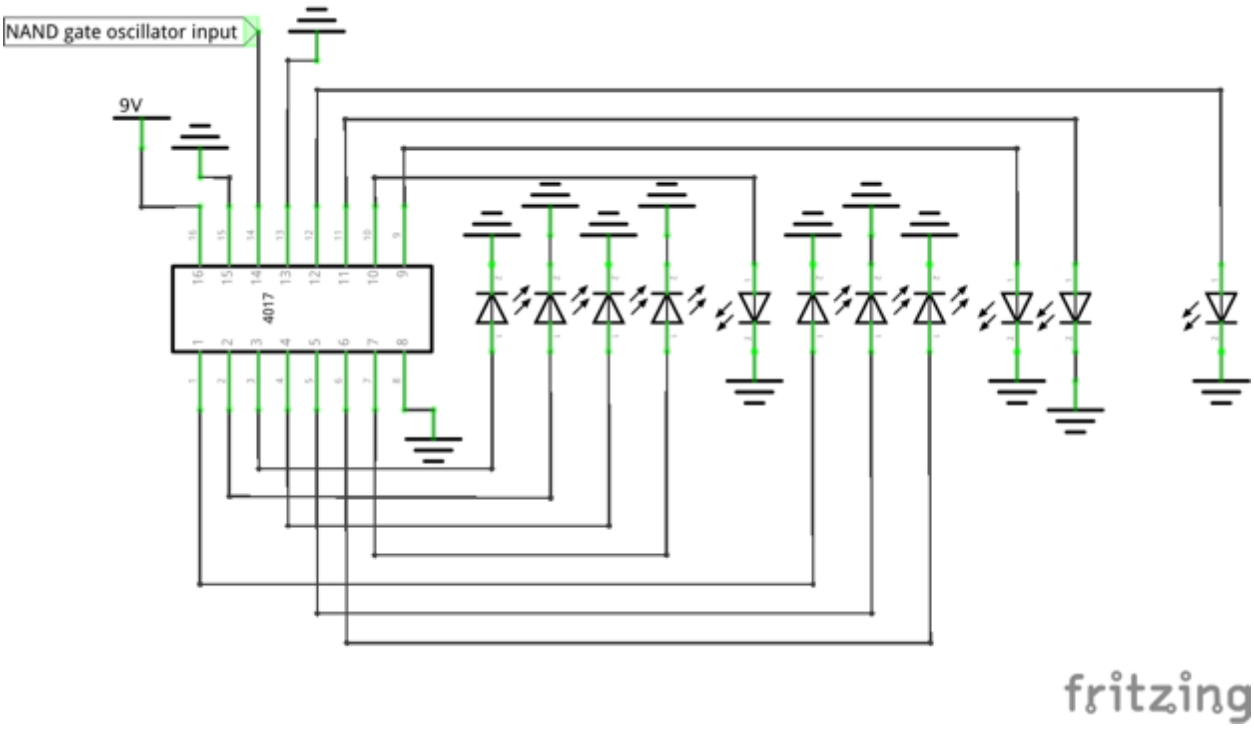
Full setup of the NAND gate oscillator

YouTube video link (parts A and B): https://youtu.be/Wj4ZiwTUSnk

Reflection

This circuit was one of the most challenging but rewarding to understand and build. As the course slowly introduces more and more content, I was forced to think harder and deeper which is one of the most important experiences I have ever had. On my DER however, I was pressed for time because of a robotics competition on Saturday that ate up most of my filming day. This led to completing and filming the video on Friday and finishing everything on the counting circuit on Wednesday and Thursday. However, because of my limited time, I learnt how to focus very hard and complete things efficiently. Overall, a very busy and stressful but rewarding week.

Part C: Decade Counter (4017)



Purpose

Part C includes a circuit that systematically lights up LEDs for a certain duration and frequency using the NAND gate oscillator and the 4017 decade counter. This effect makes it seem as though the circuit is counting.

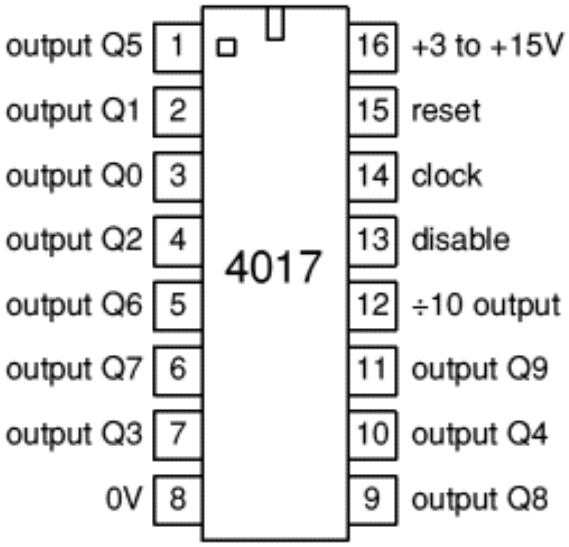
Reference

http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4017.pdf
https://electronicsclub.info/cmos.htm#4017
http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#counting

Procedure

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
1	NAND gate oscillator
1	5 mm green LED
10	5 mm red LED
1	4017 decade counter IC
1	Breadboard+

The decade counter is a sixteen pin IC featuring eleven output pins labeled 0 to 9 for the first ten outputs and a divide by 10 pin for the last output. The whole pinout diagram can be viewed on the right. Output pins 0 to 9 sequentially emit a high each time a the high of a clock signal from an oscillator is inputted into the clock input pin. For example, if the NAND gate oscillator outputs three clock signals, output 0 will emit a high for the duration of the first clock signal, output 1 will emit a high for the duration of the second clock signal, and output 2 will emit a high for the duration of the third clock signal. If the oscillator produces more than ten clock signals, the decade counter will keep cycling through outputs 0 to 9 until it stops.

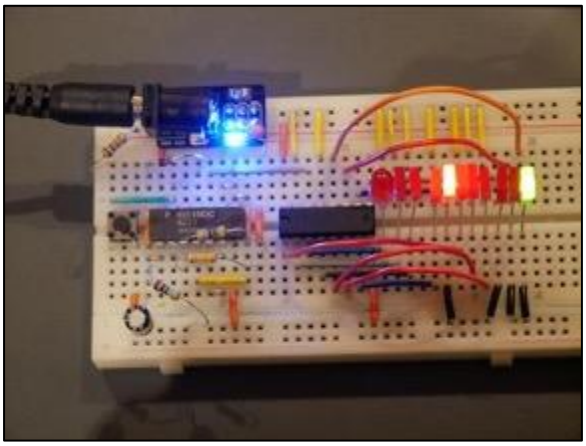


The last output pin, the divide by 10, works the same way as the other output pins but with a lower frequency. When either output pins of 0 to 4 are high, the divide by 10 pin will output high and when either output pins of 5 to 9 are high, the divide by 10 pin will output low. This equates to outputting a frequency $1/10^{th}$ of the original clock signal's frequency. In order to condition the input pins correctly so that no glitches occur, tie the disable and reset pin to ground for normal function. If the disable or reset pin is high, the IC will ignore incoming clock pulses and the count will stay constant.

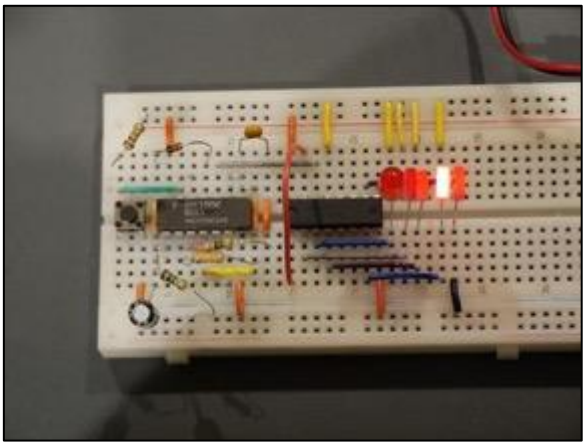
The decade counter doesn't just count to ten. By utilizing the reset pin which resets the count to zero, the counting can be shortened. To utilize it, wire a certain output pin to the reset pin and now it will only count to the number at the output pin. For example, if output pin 7 is wired to the reset pin, only output pins 0 to 7 will sequentially emit a high because when pin 7 is high, it triggers the reset pin to start the count all over at 0.

In the decade counter circuit, the clock signal from the NAND gate oscillator is fed into the input of the decade counter instead of an LED. Its output pins lead to 11 LED's to produce an LED chaser where the 10 LED's will light up one at a time to make it seem like it is counting. The last LED will light up when the half of the LED's light up and turn off when the second half of the LED's light up. The length and frequency of the clock signal can be adjusted by changing RC1 and RC2 respectively as shown in the NAND gate oscillator. The frequency changes how fast the LED's count while the duration affects the amount of time they count.

Media

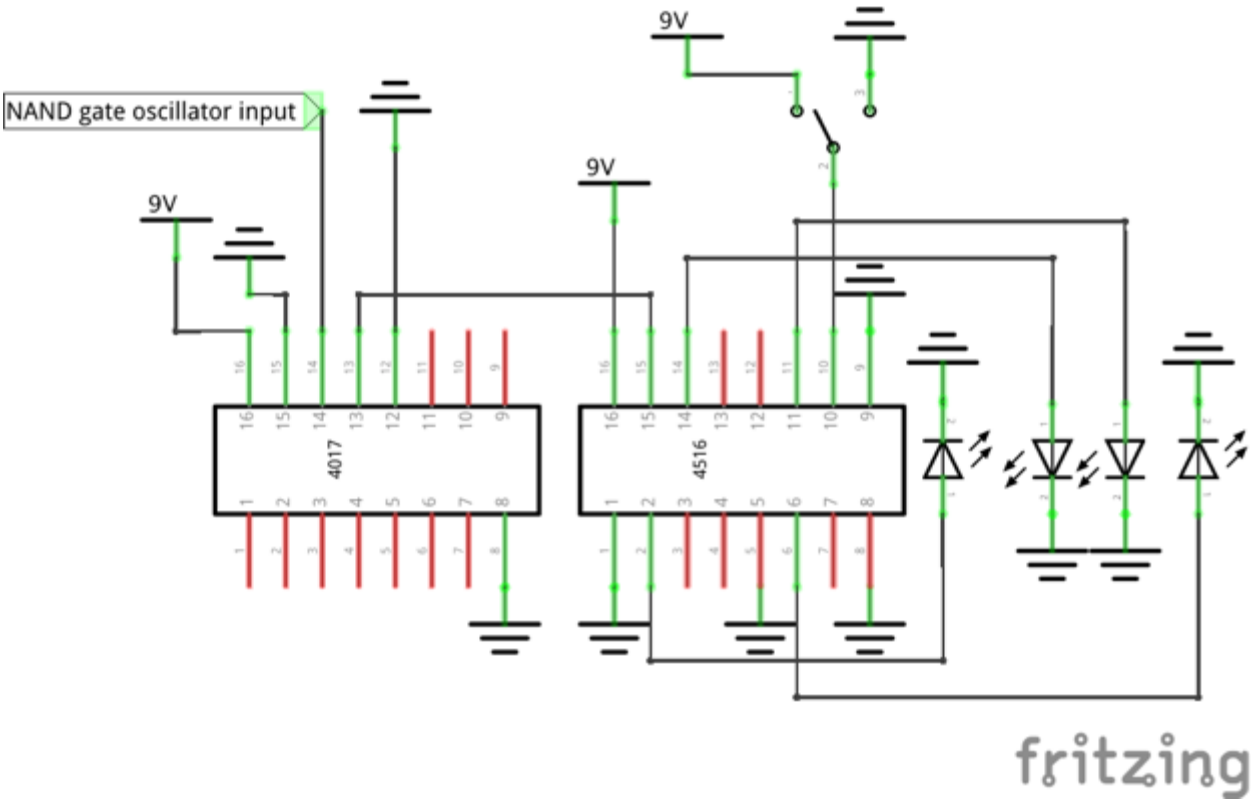


The decade counter counting from zero to ten



The decade counter only counting from zero to five

Part D: Decimal Counting Binary Up/Down Counter (4510/4516)



Purpose

Part D of the counting circuit is a direct continuation of Part C with a 4516 decade up/down binary counter IC and a few changes. These changes along with the new IC produce a number in binary that counts up or down.

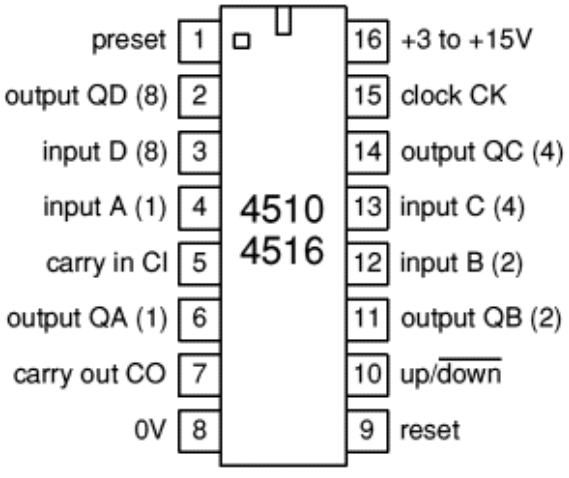
Reference

http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4510.pdf
https://electronicsclub.info/cmos.htm#4510
http://www.edutek.ltd.uk/Tutorial_Pages/BCD_Counter_4510.html
http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#counting

Procedure

The 4516 functions like the 4017 decade counter by counting up to a sixteen instead of ten. However,
unlike the 4017, the 4516 converts a clock signal into binary signals with the choice for it to count up or down. The 4516 also features a pre-set pin which allows it to count from any number up to sixteen or to count down from any number to zero. There are variants of the 4516 as well such as the 4510 which behaves exactly the same way but it only counts up to ten in binary.

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack (PB machine)
1	NAND gate oscillator
4	5 mm red LED
1	4017 decade counter IC
1	4516 binary up/down counter IC
1	Breadboard+

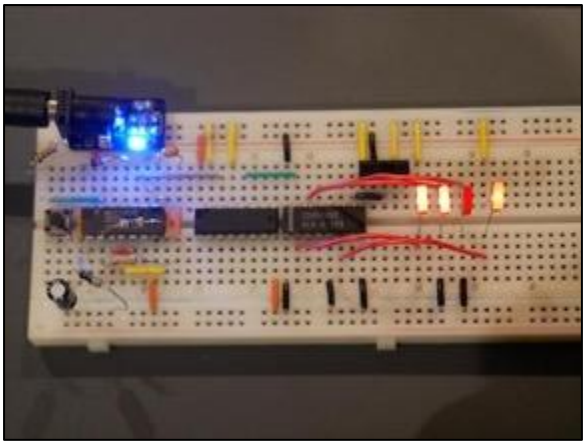


Altogether, the 4516 features 16 pins as shown on the right each with different instructions. Firstly, there are four output pins, sending out a 4-bit binary encoded signal using highs and lows each time a high from a clock signal is fed into the clock pin. Secondly, the present pin, wired to the four input pins, sets the IC to start at any number. For example, if input A and C are wired to the present pin, the IC would replace zero with five in binary as its base number. Thirdly, when the IC starts to count, it can either count up or down from its base number depending on the state of the up/down pin. If the up/down pin is low, the IC will count down and vice versa. Lastly, there are the carry out and carry in pins used for cascading multiple 4516's.

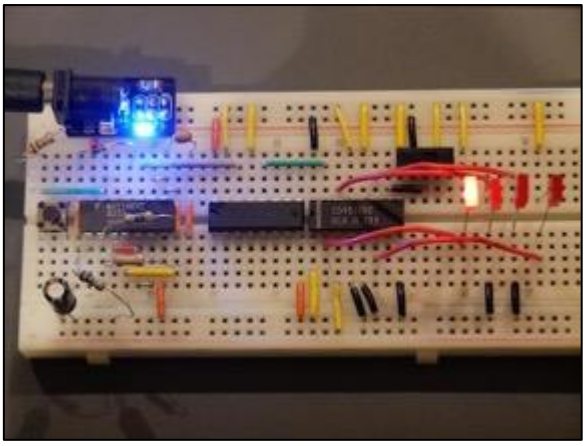
In the binary up/down counter circuit, a 4516 is used, which is a 4-bit counter counting to sixteen, instead of the 4510, which only count to ten. Four LED's are connected to the four outputs to represent each bit. When the LED is lit, that represents a 1 while when the LED is off, it represents a 0. Below is a conversion table is provided to translate binary into decimal as well as other numeric systems.

Numerical System Translator Table			
Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Media



The 4516 binary up/down counter



The 4516 binary up/down counter with a pre-set of eight

Part E: Binary Counting Decimal Decoder (4511)

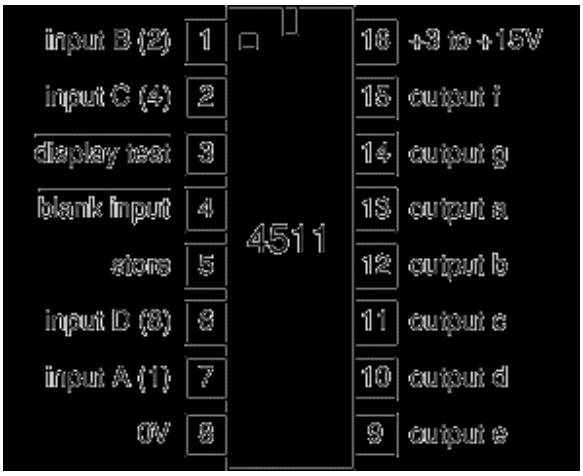
Purpose

In order to convert binary outputs from the 4510 to a readable decimal number, 4511 or a binary counting decoder is needed.

Reference

http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4511.pdf
https://electronicsclub.info/cmos.htm#4511
http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#counting

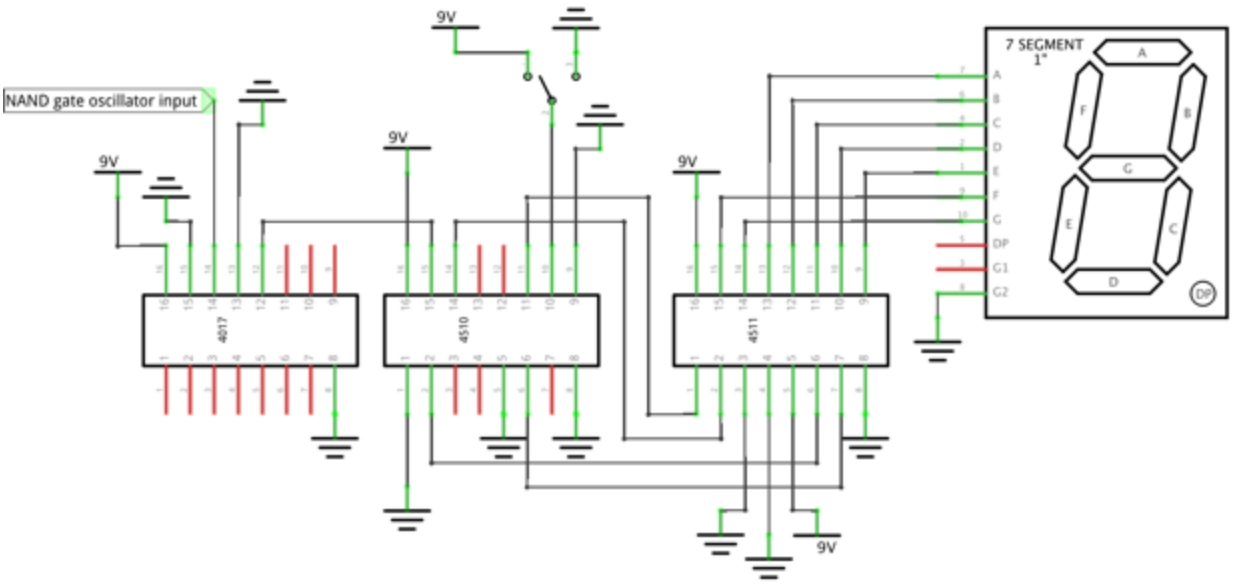
Procedure



The 4511 converts a binary number into outputs that correspond to lighting up the numbers on a 7-segment display. These binary outputs are wired to inputs A, B, C and D of the 4511. Each binary number input produces a pattern of highs from outputs a to f that form numbers zero to nine. For example, if input A is high while all the other inputs are low, that would be a 0001 in binary and outputs b and c will be high which lights up 1 on the 7-segment display. The three other pins are necessary for the function of the 7-segment display. The display test lights up all of the LED's on the 7-segment display when grounded which ensures that the 7-segment display is fully functioning. The blank input turns off all of the LED's when grounded and the store pins stores a number on the 4511 so that the binary input is ignored. The blank input pin should be high and the store pin should be low for normal operation. The last pin called the store pin stores a number on the display when the pin is high. When this IC is used in a circuit, a 4510 or similar IC that counts up to nine is needed to ensure proper counting. If a binary number higher than 9 is fed into the inputs, the output pins will produce a low until the count reaches a number from zero to nine.

4511 Output Table							
7-segment display number	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	0	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	0	0	1	1

Part F: Seven-Segment Display



Purpose

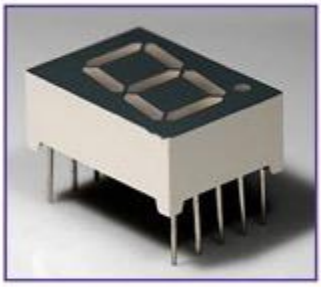
With the edition of the 7-segment display, the counting circuit is complete by providing a readable number from the 4511 decoder.

Reference

http://mail.rsgc.on.ca//~cdarcy/Datasheets/7SegmentDisplay.pdf
http://darcy.rsgc.on.ca/ACES/TEL3M/1920/TasksFall.html#counting

Procedure

The 7-segment display contains 7 LEDs labeled a to f arranged in a figure eight and one more as a decimal point. This labelling makes wiring the 7-segment display with the 4511 easy since wires just connect to their respective letters. Each LED is arranged in parallel to produce uniform brightness when the LEDs form different numbers. With this figure eight arrangement, numbers 0 to 9 can be formed by lighting up certain sections. For example, to form a 1, sections b and c should be lit up.



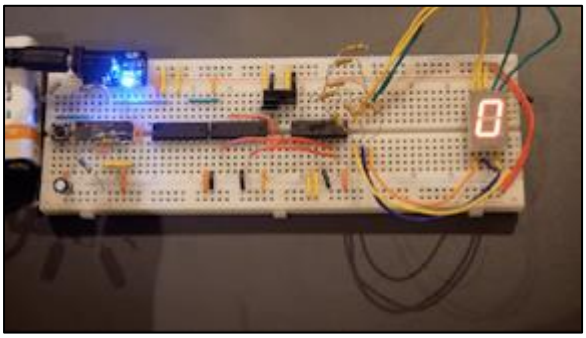
There are two types of 7-segment displays. The common anode and the common cathode. The common anode means power comes in from one pin and each LED is grounded while common cathode takes in power from every LED and is grounded through one pin. When using 7-segment displays, separate resistors are needed for each LED segment to produce uniform brightness for different numbers. If separate resistors are not used, a number with more high LEDs would be dimmer than a number with less.

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
1	NAND gate oscillator
1	4017 decade counter IC
1	4510 binary up/down counter IC
1	4511 binary decimal decoder IC
1	CC 7-segment display
7	330Ω fixed resistor
1	Breadboard+

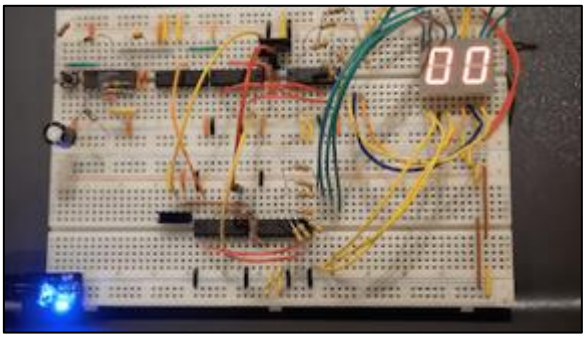
In the final part of the counting circuit, a common cathode 7-segment display is used with resistors feeding from LEDs a to f to the corresponding output pins of the 4511. The 4511 produces outputs that light up certain LED segments to form numbers which corresponds to each clock pulse. Also, the 4516 was replaced with the 4510 since the 7-segment display can only count from 0 to 9.

For further use of the counting circuit, more 7-segments displays paired with 4511s and 4510s can be used to form double- or triple-digit counting circuits by chaining the carry out pin of one 4510 to the carry in pin of the other 4510. This way, the clock signal is $1/10^{th}$ less than the previous clock signal for every chained 4510.

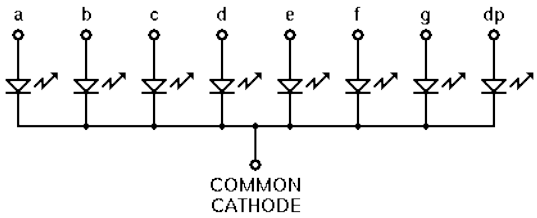
Media



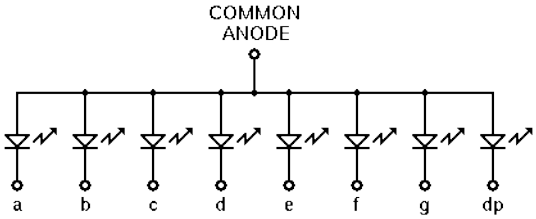
The full counting circuit at rest



A double-digit counting circuit at rest



Common cathode configuration



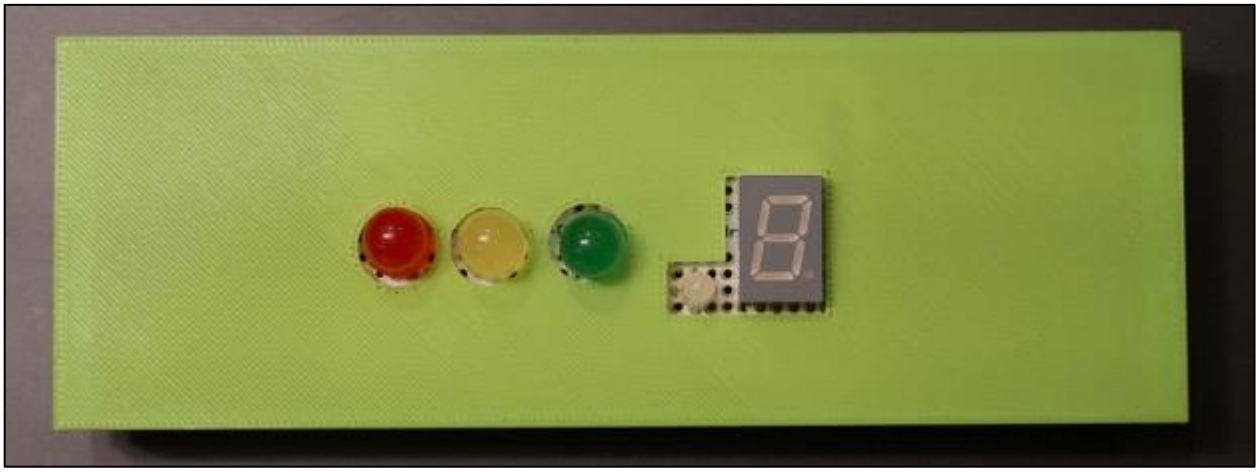
Common anode configuration

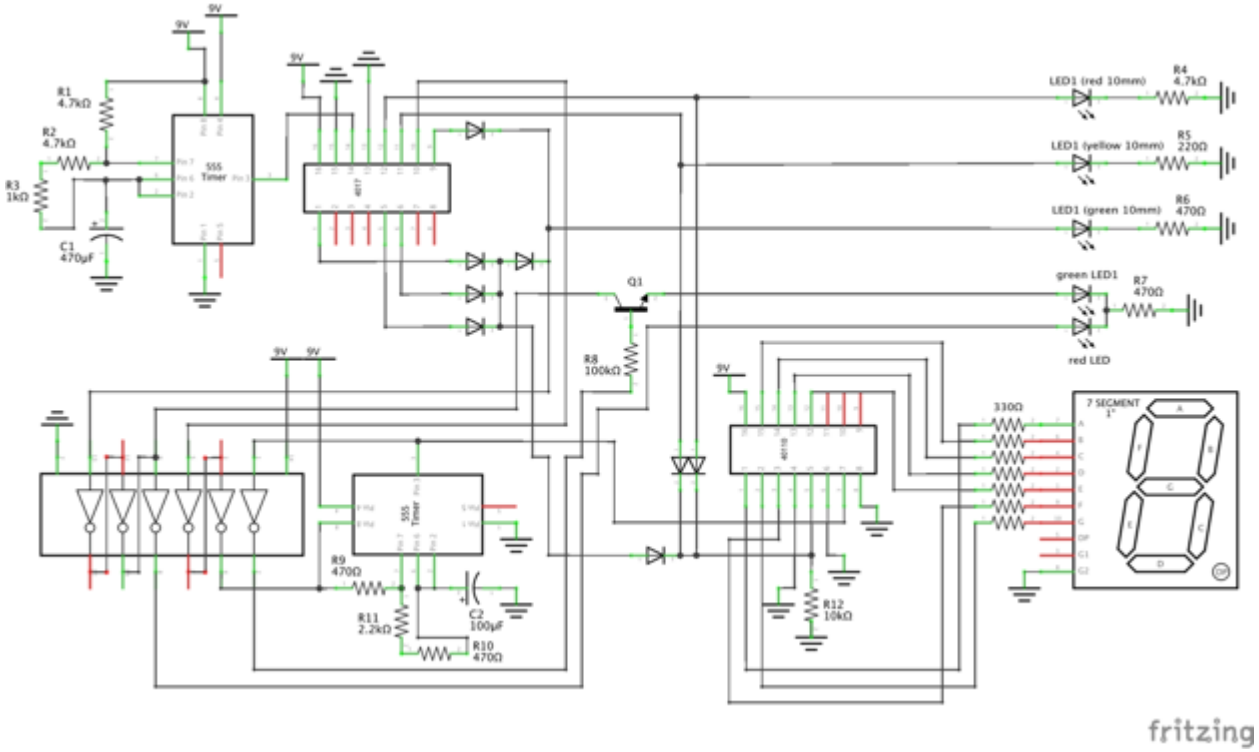
YouTube video link: https://youtu.be/r2jyUD1W5HM

Reflection

This circuit was a fun one to build. It certainly challenged me in following instructions in pinout diagrams of all the integrated circuits. However, my time management improved and I was able to complete the full circuit without any stress. As a challenge and a way to knock off some spare time, I built a double counting circuit which was a very special moment for me. I was also able to understand all of the components, how they all work together and how to wire them correctly. One problem that perplexed me for a while was when my 7-segment display pins were very dim. For a long time, I kept rewiring things and making sure my components were functioning properly until I realized that my battery was low on power. This taught me to always check your battery level with a voltmeter. Overall, this circuit was a wonderful circuit to build and I had a lot of fun with it.

Project 1.5 (ISP): The Analog Traffic Light





The Analog Traffic Light Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	9V DC power jack
1	Adafruit full-sized perma protoboard
1	Adafruit half-sized perma protoboard
1	3D printed analog traffic light case
1	4017 decade counter IC
2	555 timer IC
1	4049 hex NOT gate IC
1	40110 up/down counter and LED driver IC
1	CC 7-segment display
1	10 mm red LED
1	10 mm yellow LED
1	10 mm green LED
1	5 mm bicolor LED (triple pin)
8	Rectifier diode
1	220 Ω fixed resistor
7	330 Ω fixed resistor
3	470 Ω fixed resistor
1	1 kΩ fixed resistor
3	4.7 kΩ resistor
1	10 kΩ fixed resistor
1	100 kΩ fixed resistor
1	470 µF capacitor
1	100 µF capacitor
1	2N3904 NPN transistor

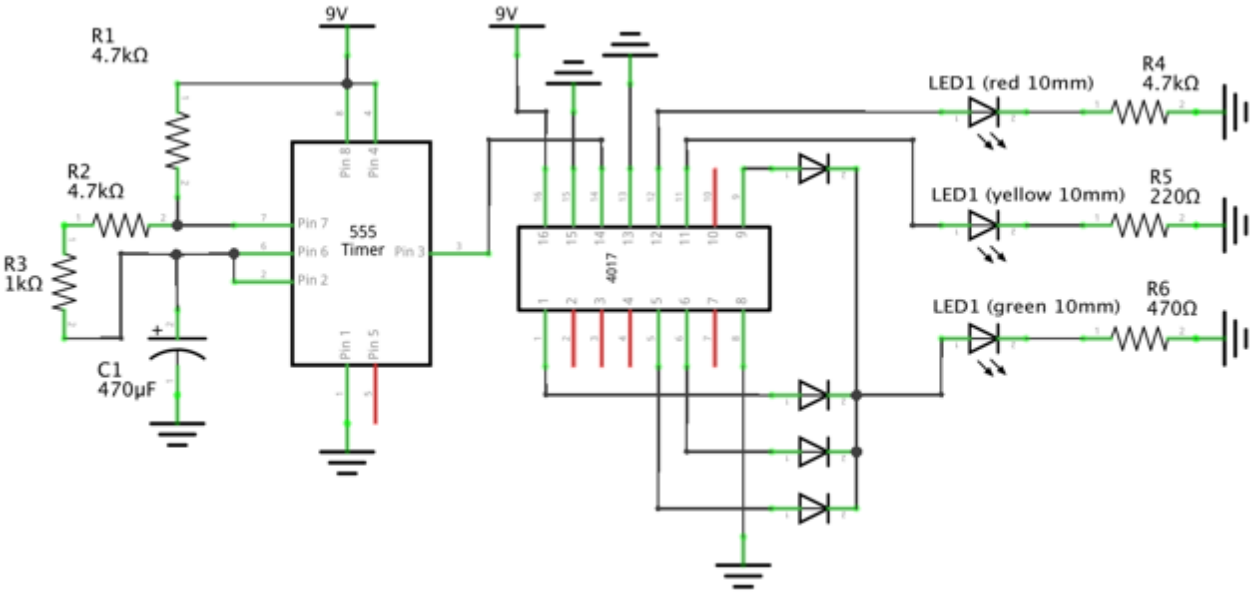
Purpose

The purpose of this final project is to utilize and combine all of the skills learnt in this course to achieve the goal of designing a completely analog traffic light with a pedestrian signal and a countdown timer.

Theory

The analog traffic light, unlike conventional ones, is one that is run with no programming involved. To achieve this, a plethora of IC's, resistors, capacitors, and transistors work in tandem with each other to replicate a fully functioning traffic light. The 4017 decade counter, drives the main red, green, and yellow lights, the 4049 hex NOT gate IC drives the bi-color LED and provides maximum power efficiency, a 555 timer oscillates the bi-color LED, and an up/down counter LED display driver 40110 IC powers the countdown timer.

Part A: The Base Traffic Light



Purpose

The purpose of this circuit it to designate a timing period to red, yellow, and green LED's to form the basis of a traffic light.

Reference

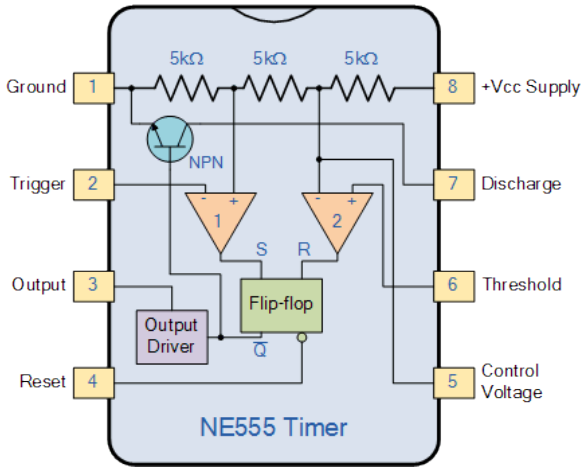
https://www.youtube.com/watch?v=i0SNb__dkYI

Procedure

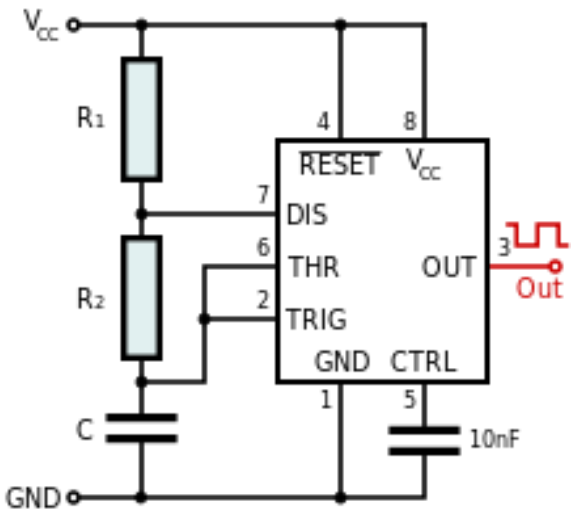
Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack (PB machine)
1	4017 decade counter IC
1	555 timer IC
1	10 mm red LED
1	10 mm yellow LED
1	10 mm green LED
1	220 Ω fixed resistor
1	470 Ω fixed resistor
1	1 kΩ fixed resistor
3	4.7 kΩ resistor
4	Rectifier diode
1	470 µF capacitor
1	Breadboard+

In order to obtain a repeated pattern of red, green, and yellow LED's, a 4017-decade counter is utilized. However, this repeated pattern has a uniform time interval for each output pin, which would result in the red and green light having the same on time as the yellow light. To solve this issue, multiple output pins of the 4017 are connected to each LED to produce a flash sequence. Basically, each output pin connected to an LED lengthens the duration of it being on. In the analog traffic light, this sequence is set to five output pins for the red LED, four output pins to the green LED, and the remaining one for the yellow LED to mimic the real-life version of a traffic light. But if wires are connected to each LED, current from the output will travel through one of the output pins to ground so to prevent that, each output pin connected to each other needs to have a diode. The divide by ten pin, which is equivalent to five output pins, is connected to the red LED to save on diodes.

The 555 timer is comprised of a voltage divider with three five kiloohm resistors, hence the name 555. In addition to that, there are two comparators, a flip flop, a discharge transistor, and an output as shown in the corresponding diagram. At first, the output of the two comparators are 0 and 1 from the top down. The flip flop will flip these outputs to be 1 and 0 and the output stage will invert the bottom flip flop output. In this case, the bottom output from the flip flop is 0 so the output stage will become a 1.



The 555 timer is used to input a square wave into the decade counter. This timer can be easily configured to produce a square wave with varying frequency and duty cycle using resistor capacitor pairs in the astable mode configuration to the right. To increase the frequency of the square wave, decrease the capacitance of C and the resistance of R1 and to increase the duty cycle, increase the resistance of R2 which will also decrease the frequency by a touch.

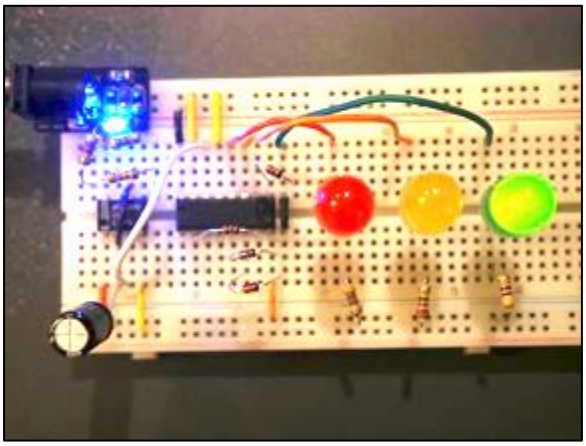


When voltage is applied to the power pin, the capacitor charges up through R1 and R2 and voltage is fed through the trigger and threshold pins to the comparators. The more the capacitor charges up, the more voltage is experienced on the two comparators. When a voltage higher than 1/3 of the source voltage is on the trigger pin, the bottom comparator switches its output to 0 but it does not change the output of the flip flop so no change is made to the output. When a voltage higher than 2/3 of the source voltage is on the threshold pin, the comparator switches its output to 1. Now the flip flop has an input of 1 and 0 from top to bottom and because of its logic, these outputs get flipped and a 1 is outputted to the output section which is then inverted to 0. There is also a discharge NPN transistor that is connected to the output of the bottom flip flop. When that bottom output becomes high, the transistor saturates and the capacitor begins to drain through it.

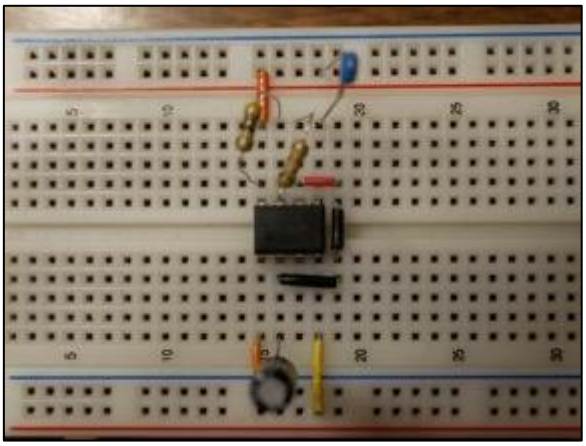
Once the capacitor starts to drain through R2, voltage on the threshold pin will dip below 2/3 of the source voltage and the top comparator will output a 0 which will not change the output of the flip flop as shown previously. It is only when voltage across the capacitor is 1/3 less than the source voltage that the bottom comparator changes its output to a 1 and so the flip flop outputs a 0. This cuts off the NPN transistor, allowing the capacitor to charge again and the output stages outputs a 1. This cycle of the 555 repeats indefinitely until power is turned off.

In the analog traffic light, a resistance of 4.7 kiloohms as R1, a resistance of 5.7 kiloohms R2, and a capacitance of 470 microfarads is used to produce a square wave of around five seconds in length which is quite close to the duration of a yellow light in a real traffic light. This timing would make a red light 25 seconds, a green light is 20 seconds and a yellow light is five seconds.

Media

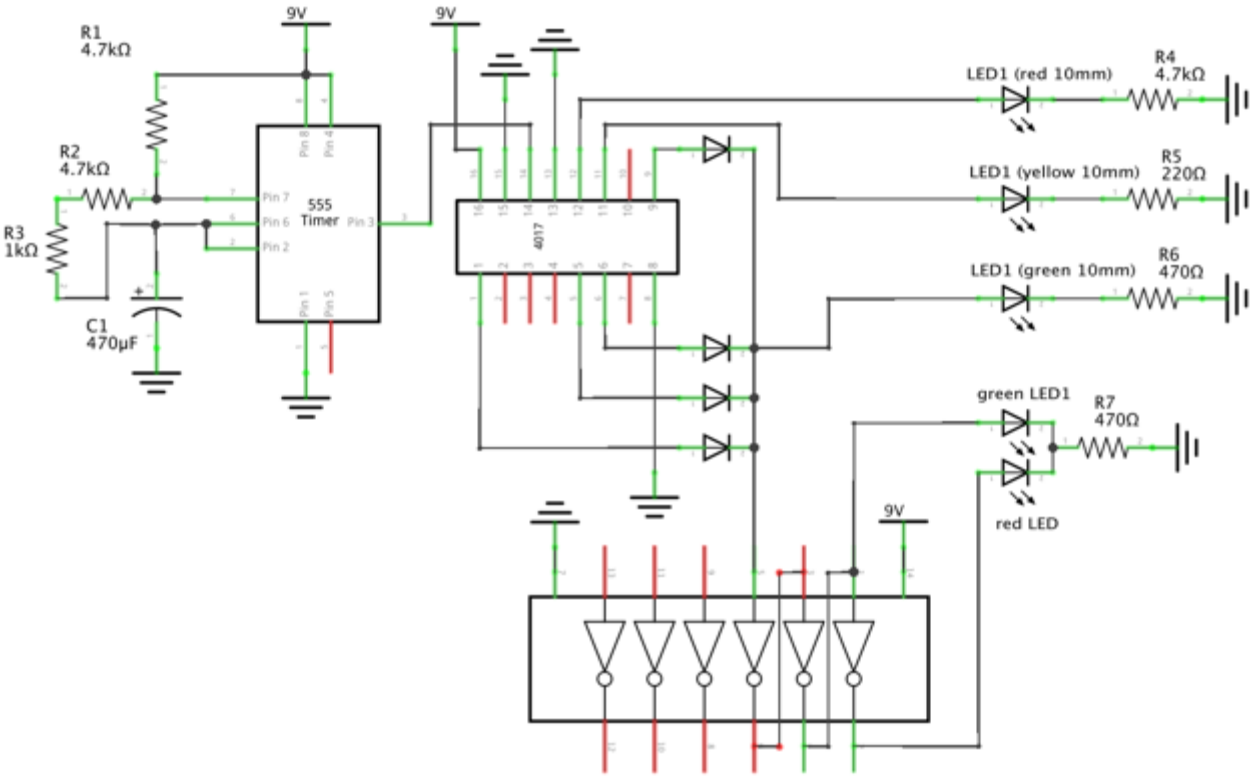


The base traffic light on a breadboard



The setup of a 555 timer

Part B: The Pedestrian Signal



Purpose

The purpose of this circuit is to implement a pedestrian signal so the traffic light is useful for both vehicles and people.

Reference

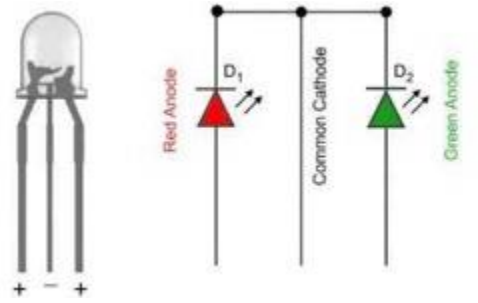
http://tinyurl.com/sjdd3ld
http://lednique.com/gpio-tricks/1-gpio-dual-led-common-cathode/
https://electronicsclub.info/cmos.htm#4049

Procedure

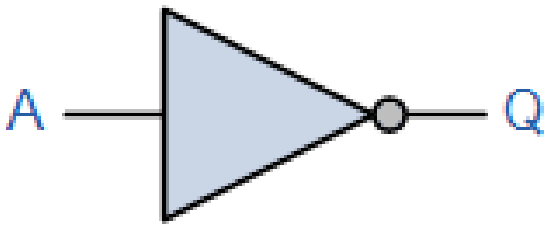
A pedestrian signal from a traffic light has only two modes which are stop and go. Using a bicolor LED, a simple pedestrian signal can be created where red is stop and green is go. In a real-life traffic light, the pedestrian signal tells the pedestrians to stop when there is a red or yellow light and go when there is a green light. In order to do that, a three-pin bi color LED with a common cathode and an inverter in between the two LED's is needed.

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack (PB machine)
1	Base traffic light
1	4049 hex NOT gate IC
1	5 mm bicolor LED (triple pin)
1	470 Ω fixed resistor
1	Breadboard+
1	Half breadboard

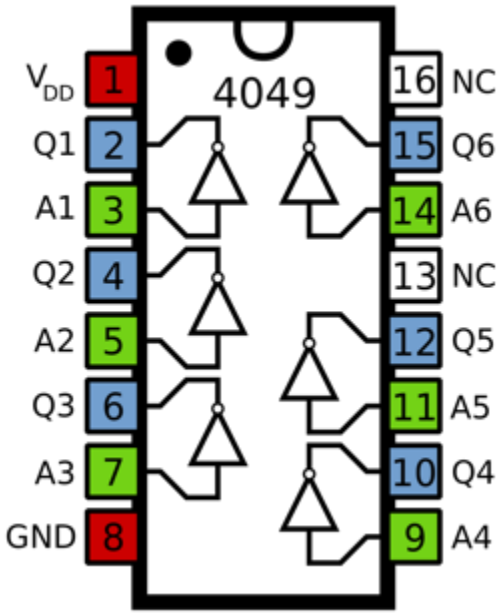
A three-pin bi color LED is a little different from the usual two pin bi color LED. Its schematic is shown on the right. In order for it to work, the middle pin must be grounded and input on one pin will turn on its respective LED. If both pins are equally high, the LED will appear orange. A resistor can be put between the middle pin and ground for voltages below 5 volts but for high voltages, each LED should have its own resistor to prevent reverse biasing the LEDs.



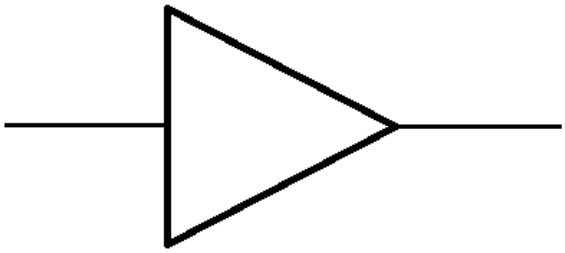
An inverter or a NOT gate is a type of logic where the input is the opposite of the output. For example, if the input is 0 the output is 1 and vice versa. The symbol for an inverter is shown on the right. The components of an inverter are very simple. All that is needed is a pull-up resistor configuration with an NPN transistor replacing the push button. Now the base of the NPN transistor with a current limiting resistor becomes the input while the output of the pull-up resistor becomes the inverter output.



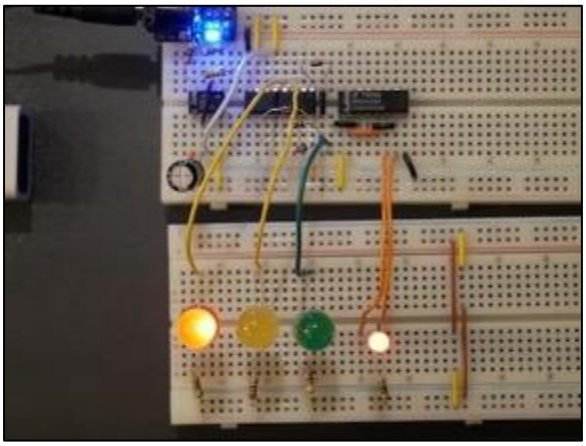
In the analog traffic light, a 4049 IC chip is used which houses six inverters. When there is a high voltage on an inverter, the green LED will turn on and when there is no voltage at all, the red LED will turn on. The green LED should be on when a green light is showing and the red LED should be on otherwise. In order to do that, the input signal of the green light is wired to the input of the inverter. That way, when the green light is on, the inverter will output a low, turning off the red LED while the green LED is turned on from the green light input signal. When a red or yellow light is showing, there is no input signal from the green light which turns off the green LED and outputs a high from the inverter, turning on the red LED.



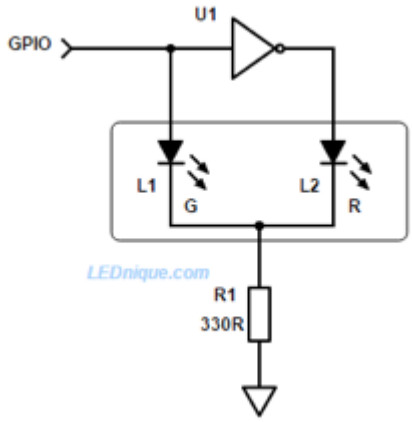
When testing the circuit, the green light of the pedestrian signal would be quite dim. To ensure maximum brightness for it, the input of the green light signal is wired through two inverters forming a buffer gate. A buffer gate, as shown on the right, simply outputs the signal inputted into it. For example, if the input is 1, the output is 1. However, an analog high voltage such as five volts can be put through the buffer gate and it would register that input as high which will output a high but at nine volts instead of five. This would solve the minor diode voltage drop problem. It would also solve another problem. Since gates only require voltage, all output current can flow through the green LED traffic light ensuring its maximum brightness. The green LED pedestrian signal would also get maximum brightness since all current from the buffer gate would travel through it. If the buffer gate was not there, the output current from the decade counter would have to be divided among the two green lights, dimming them both.



Media

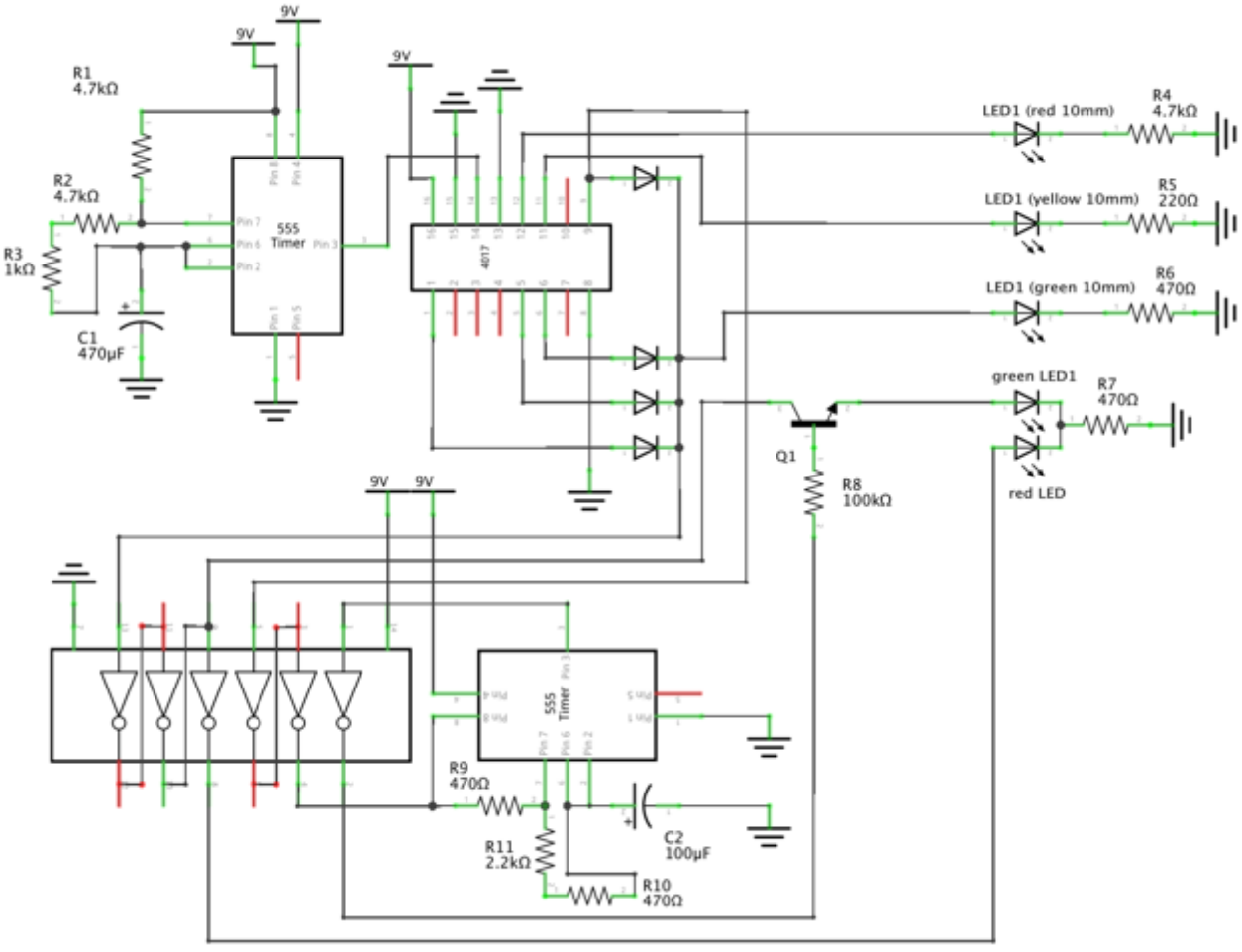


The base traffic light with the pedestrian signal



The schematic of driving a three-pin bi-color LED

Part C: The Oscillating Pedestrian Signal



Purpose

The purpose of this circuit is replicate the flashing go sign before a yellow light to alert pedestrians to finish or stop crossing the road

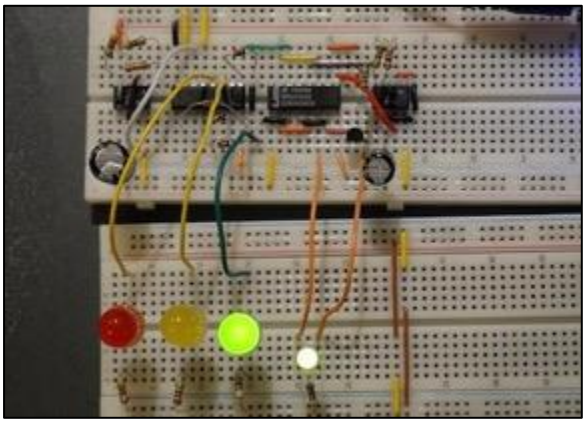
Procedure

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
1	Base traffic light with pedestrian signal
1	555 timer IC
1	2N3904 NPN transistor
2	470 Ω fixed resistor
1	2.2 kΩ fixed resistor
1	100 kΩ fixed resistor
1	100 μF capacitor
1	Breadboard+
1	Half breadboard

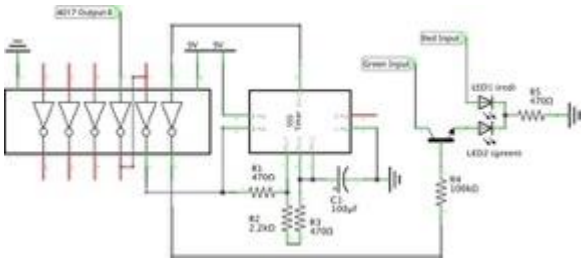
To replicate the flash, the green part of the bi-color LED will oscillate on and off before a yellow light. To create the flashing sequence, another 555 timer in astable mode is needed to form the square wave. This square wave will then turn on and off the green light of the bicolor LED so to do that, the square wave is fed into the base of an NPN transistor with a current limiting resistor. The NPN transistor is then put in between the output of the inverter and the input of the green light of the bicolor LED. Now, the transistor acts like a solid-state switch; however, the green light of the pedestrian signal will not be on until the square wave is fed into the base of the transistor. To fix that, an inverter is used from the 4049 IC to turn the NPN into a solid state PBNC which will only open when current from the square wave is on the base pin.

Now, power must be fed into the 555 timer but if power is directly applied, the 555 timer will keep producing a square wave and therefore keep the LED flashing. To solve this problem, power is provided from output 8 of the decade counter which is the output right before the yellow light output. This way, the LED oscillation will start right before a yellow light. However, the green light would always become dim when the oscillation started because the output of the decade counter has a max output current. Therefore, two inverters are needed to form another buffer gate so that all output current from the decade counter can flow into the green LED.

Media

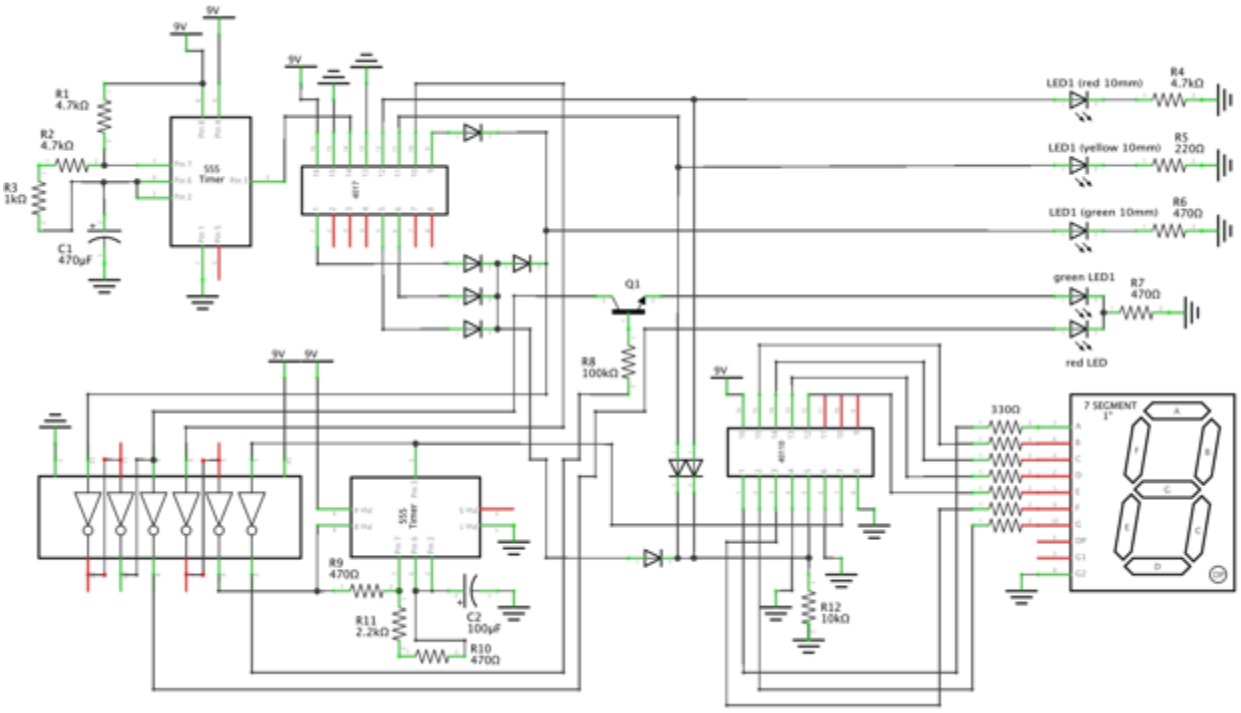


The traffic light with an oscillating pedestrian signal



The schematic of driving an oscillating pedestrian signal

Part D: The Countdown Timer



Purpose

The purpose of the final circuit addition is to improve on the oscillating pedestrian signal to let pedestrians and vehicles know exactly when a yellow light is about to happen.

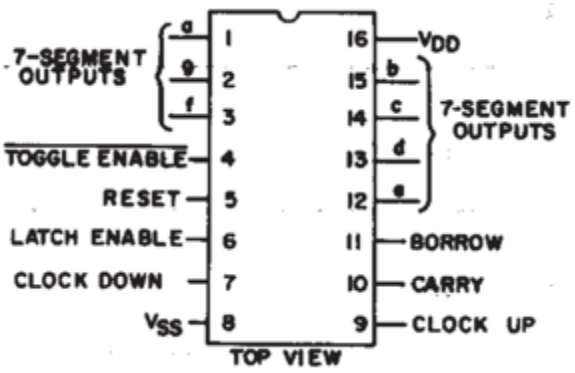
Reference

http://www.ti.com/lit/ds/symlink/cd40110b.pdf

Procedure

For the analog traffic light, a single digit number counting down from nine was chosen to simplify the circuit instead of the normal double-digits. To achieve the countdown, a 40110 IC is used to drive a 7-segment display. This handy IC is an LED driver with an up/down counter feature so when a clock signal is inputted into the IC, it will begin to send out a combination of outputs from a to g that form numbers counting up, if the clock signal is inputted into the clock up pin, or down, if the clock signal is inputted into the clock down pin, on a 7-segment display. The chip is similar to a 4510 paired with a 4511; however, no pre-set is included but since the analog traffic light counts down from 9, no pre-set is needed.

Parts Table	
Quantity	Description
1	9V DC alkaline battery
1	ACES power jack
1	Base traffic light with the oscillating pedestrian signal
1	40110 up/down counter and LED driver IC
1	CC 7-segment display
4	Rectifier diode
7	330 Ω fixed resistor
1	10 kΩ fixed resistor
1	Breadboard+
1	Half breadboard



There are three other pins that play a crucial role in the proper function of the chip. Firstly, there is the reset pin. When a high is present on that pin, the 7-segment display and the internal counter of the IC will reset to zero. Secondly, there is the latch enable pin. This pin will freeze the number displayed on the 7-segment display but the internal counter will still keep counting. Lastly, there is a toggle enable pin which needs to be grounded in order for the internal counting circuit to work. For normal functioning, all three pins should be grounded to avoid the pins picking up any stray pulses. The two remaining pins are toggle and carry and they are used to cascade multiple counters for double or triple digit displays. The full pinout of the 40110 IC is shown on the right and an output table as well as a truth table is shown below.

40110 Output Table							
7-segment display number	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

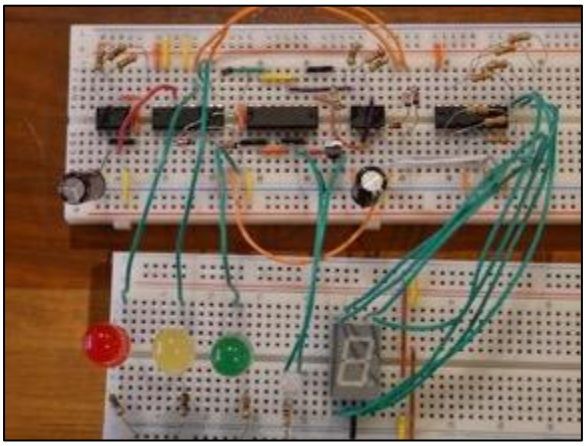
CLOCK UP *	CLOCK DOWN *	LATCH ENABLE	$\overline{\text{TOGGLE}}$ ENABLE	RESET	COUNTER	DISPLAY
⎍	X	0	0	0	Increments by 1	Follows Counter
X	⎍	0	0	0	Decrements by 1	Follows Counter
⎎	⎎	X	X	0	No Change	No Change
X	X	1	X	1	Goes to 00000	Remains Fixed
X	X	0	X	1	Goes to 00000	Follows Counter (Display = ⎍)
X	X	X	1	0	Inhibited	Remains Fixed
⎍	X	1	0	0	Increments by 1	Remains Fixed
X	⎍	1	0	0	Decrements by 1	Remains Fixed

X = Don't Care 1 = High State 0 = Low State

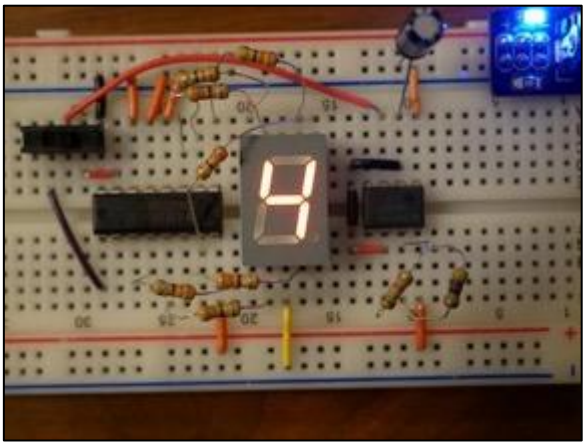
In the analog traffic light, the clock signal from the second 555 timer is inputted into the clock down pin. This way, the count will only start when the pedestrian signal starts oscillating which is right before a yellow light. Also, since the clock signal is inputted into the clock down pin, the counter will start to count down from 9. However, the length of the square wave has to stop when the counter reaches 0 otherwise the count will count from 9 again. To prevent that, a capacitance of 100 microfarads, a resistance of 470 ohms for R1 and a resistance of 2670 ohms for R2 is needed for the second 555 timer.

To prevent any potential glitches, the reset pin should be tied high when it is not counting to be sure the counter stays at zero until it needs to count. To do that, four more diodes are needed and a ten kiloohm resistor. The resistor ties the reset pin to ground and it prevents a short when current is in the reset pin. The four diodes are for preventing current from travelling through the LED's and into ground, keeping them on. One diode is for the red light, one diode is for the yellow light, and two are for the green light. The reason a second diode is needed is to prevent the reset from staying high when the countdown is supposed to happen. In order to do this, outputs five six and seven need to travel through one more diode to light the green LED and the reset pin is tied right before the second diode so that only outputs five six and seven can render it high. This prevents current from output eight, the output that initiates the countdown, travelling to the reset pin.

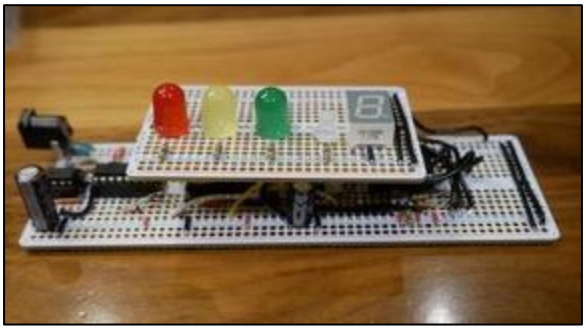
Media



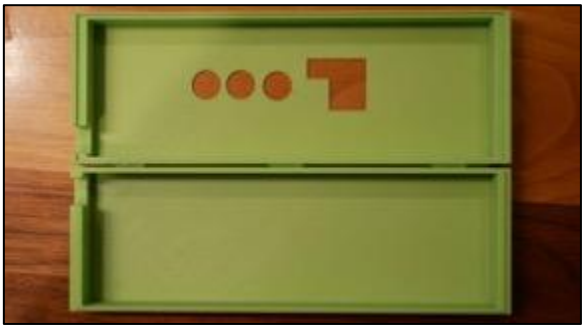
The traffic light with the addition of the countdown timer



A simple circuit that counts up or down using the 40110 IC



The soldered version of the analog traffic light



The 3D printed case for the analog traffic light

YouTube video link: https://youtu.be/O-rvWUiZsPw

Reflection

This half year was one of the most life-changing experiences that I have had. I feel that I have come a long way from a proud gamer, in the beginning of the year, to a proud junior ACE that knows what it feels like to really push yourself and do the best you can. Honestly, when the first DER submission came around, I was really nervous because I didn't want famously get "roasted" so I quit gaming for a while and worked non-stop Friday and Saturday which is what led me to a perfect submission. That first perfect submission inspired me to continue on working even harder for the rest of the reports and to prioritize work over procrastination. This has led me through some very stressful paths (such as completing this final project) but it paid off and I owe most of it to you for pushing everyone to be the best version of themselves. Thank you for a great year.

ICS3U

AVR Foundations

Project 2.1: The Traffic Light

Purpose

The purpose of this project is to combine hardware and software to create a traffic light and to obtain a basic understanding of the Arduino and its functions in order to lay the foundation of efficient coding to use in the rest of this year.

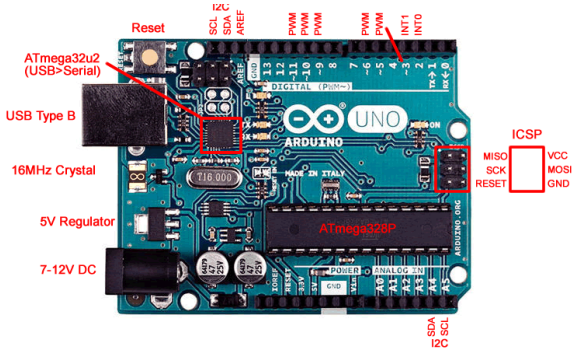
Reference

https://learn.sparkfun.com/tutorials/what-is-an-arduino/all
https://www.arduino.cc/en/Tutorial/Foundations/Memory
https://mail.rsgc.on.ca/~cdarcy/Datasheets/ATmega328P.pdf
https://crashcourseforarduino.libsyn.com/the-arduino-development-toolchain-how-it-all-gets-done

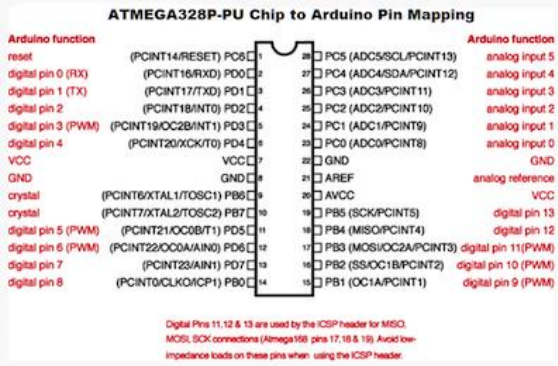
Procedure

The Arduino platform is a fundamental introduction to the application of software to hardware. It allows the user to replicate the complex circuits that would take up several breadboards and bags of components only using a few lines of code. It simplifies the process of engineering and allows more complex creation feats, but, to achieve those, learning about the foundations, functions, and commands of Arduino is key. To do this, a simple working traffic light is programmed using very basic code.

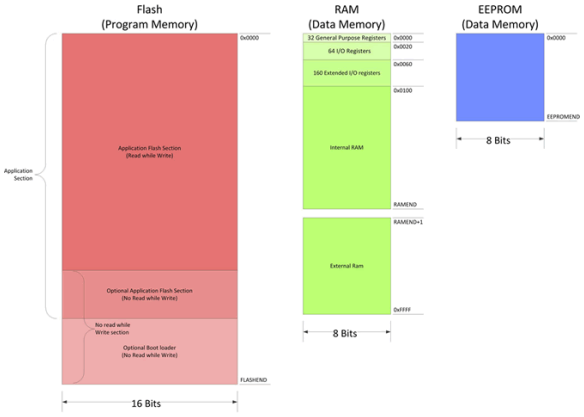
An Arduino board is the combination of a programmable microcontroller with a plethora of functions and an integrated development environment (IDE) where instructions are fabricated to preform those functions. It is essentially a mix of hardware and software. Commands and code are written in the IDE and sent out in machine code, telling the microcontroller what functions to perform. For this project, the Arduino Uno board is utilized.



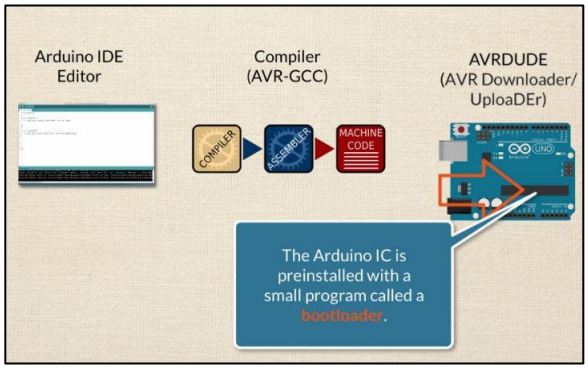
The hardware aspect of the Arduino Uno board is shown in the form of the 8-bit ATmega328P microcontroller. This microcontroller is a 28 to 32 pin IC featuring numerous general purpose I/O pins used for a variety of purposes such as PWM (pulse width modulation), counting and comparing paired with memory to store the commands inputted through software.



The memory is divided up into three parts: flash memory, SRAM, and EEPROM. Flash memory, containing 32KB, stores the whole file or sketch of code, SRAM, containing 2KB, stores variables and finally, EEPROM, containing 1KB, is where long-term information regarding the inputted code is stored. Flash memory and EEPROM is non-volatile, signifying that the information written to it are not lost when power is disconnected while SRAM is volatile meaning information is lost upon power loss. This is mainly because SRAM is made for speed in order for the code to run quickly.



On the flip side, the software aspect comes in the form of the Arduino IDE editor, a development software application where code is written. This development software employs the use of the C++ language to create tasks and functions that the Arduino board carries out. When those tasks and functions have been fabricated in the form of code, it uploaded through a series of steps and processes called the Arduino toolchain, consisting of the Arduino IDE editor, the compiler, AVRDUDE (AVR Downloader and Uploader) and the bootloader. The Arduino IDE editor uploads the C++ code with the push of the upload button to the complier called AVR-GCC, which checks for errors and assembles it into machine code in the form of a .hex file. This .hex file is transported to the board through AVRDUDE and the bootloader writes the code onto the MCU.



The Schaffer Traffic Light PCB is a simple circuit consisting of a red, yellow and green LED with a resistor near ground to limit the brightness. The LED's have their own male header pin to turn them on which all link up to the resistor and ground pin. These pins are used in conjunction with the Arduino's digital female headers to individually turn the LEDs on, simulating a traffic light.

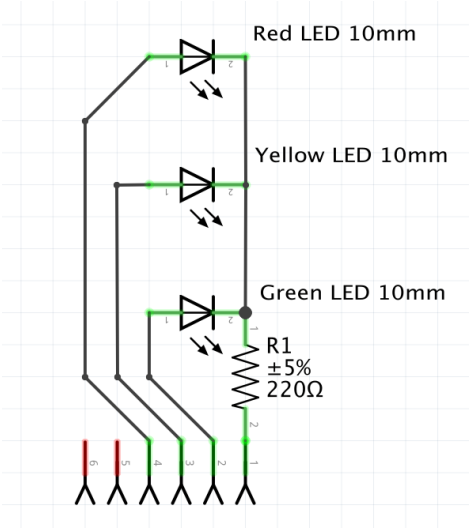
Parts Table	
Quantity	Description
1	Computer
1	USB type A to USB type B cable
1	Arduino UNO
1	Schaffer Traffic Light PCB
1	10 mm red LED
1	10 mm yellow LED
1	10 mm green LED
1	220 Ω fixed resistor
1	4 right angle male header pins

The code used emulate the traffic light is simple but crucial to future coding practices. Unsigned integers, meaning positive whole numbers, of specific sizes such as 8-bit and 16-bit were used to increase the codes efficiency and reduce the amount of space taken up in the Arduino's memory. The same goes for bit shifting which saved lots of processing power as opposed to simply dividing. As a final touch, and to make the code more user friendly, the red LED pin was linked to all the others so that the user only needs to change one variable to move the traffic light around.

```
uint8_t redPin = 11;
uint8_t yellowPin = redPin - 1;
uint8_t greenPin = redPin - 2;
uint8_t groundPin = redPin - 3;
```

Media

The Schaffer Traffic Light PCB in effect with the Arduino UNO



The schematic of the Schaffer Traffic Light PCB

YouTube video link: https://youtu.be/fSRU5WQWoZU

Code

```
// PROJECT    : Traffic Light
// PURPOSE    : To create code that controls the Schaffer Traffic Light PCB
// DATE       : Sep 23, 2020
// MCU        : 328p/84/85
// STATUS     : Working
// NOTES      : None
// REFERENCES : http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html

uint8_t redPin = 11;              //red LED pin
uint8_t yellowPin = redPin - 1;   //yellow LED pin
uint8_t greenPin = redPin - 2;    //green LED pin
uint8_t groundPin = redPin - 3;   //ground pin

uint16_t redGreenDelay = 5000;    //time regarding pauses of red and green

void setup() {    //function only runs once

  //pins are set to output
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(groundPin, OUTPUT);

  digitalWrite(groundPin, LOW);   //pin set to 0V (ground)
}

void loop() {   //function loops infinitely

  //traffic light sequence:
  lightSequence(redPin, redGreenDelay);
  lightSequence(greenPin, redGreenDelay);
  lightSequence(yellowPin, redGreenDelay >> 2);    //On for 1/4 of red and green
}

void lightSequence(uint8_t led, uint16_t delayTime) {
  //function with parameters for each traffic light LED

  digitalWrite(led, HIGH);      //traffic light LED turns on
  delay(delayTime);             //pause
  digitalWrite(led, LOW);       //traffic light LED turns off
}
```

Reflection

Overall, the traffic light was a simple but crucial first project to get started with the Arduino. Without this introductory submission, a lot of the aspects regarding efficient coding would have been missed and overlooked. It was definitely hard getting back into the swing of things; I went from a relaxing summer where I would game all night long to working hard on my DER all afternoon and late into the night. However, I know things will get a lot easier and less stressful as it did in my grade 10 year, though, things will be a lot different with the circumstances of COVID-19. I guess we just have to wait and see what lies ahead of us later in the year.

Project 2.2: Persistence of Vision

Purpose

The purpose of this project is to use software, hardware, and the persistence of vision concept to effectively and efficiently light up numbers and characters on a dual 14 pin alphanumeric display

Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#PoV
https://lastminuteengineers.com/74hc595-shift-register-arduino-tutorial/
https://www.arduino.cc/reference/en/language/functions/advanced-io/shiftout/
https://www.futurelearn.com/courses/explore-animation/0/steps/12222

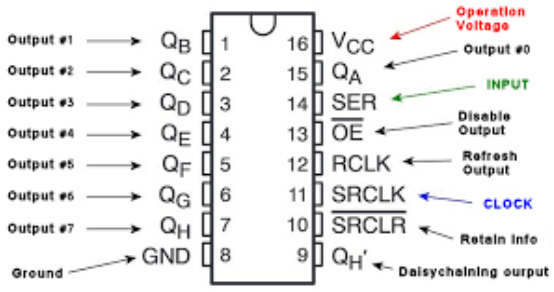
Procedure

The persistence of vision project contains many new concepts introduced in the course. It combines, shift registers, the dual 14 pin alphanumeric display, the persistence of vision effect and circuit, the use of EEPROM, and more efficient coding techniques.

Parts Table	
Quantity	Description
1	Computer
1	USB type A to USB type B cable
1	Arduino UNO
1	Common cathode alphanumeric display
4	330 Ω isolated resistor network
2	74HC595N shift register
1	3904 NPN transistor
1	3906 PNP transistor
1	10 kΩ fixed resistor
2	1 kΩ fixed resistor
1	Rectifier diode

The ICs used in the function of the alphanumeric display is the newly introduced shift register. It's job is to expand the number of I/O (input/output) pins available for use. For example, to control a 20 LED bar graph, you would normally need 20 output pins on the Arduino to control each individual LED. This setup is inefficient because most Arduino boards do not have this many pins, and even if they did, 20 I/O pins would be wasted on some simple LEDs. This inefficient use and lack of I/O pins is solved through the employment of shift registers, expanding the I/O pins to your needs while only taking up 3 I/O pins on an Arduino board.

The specific and very ubiquitous shift register utilized in the persistence of vision is the 74HC595N, commonly known as the 595 shift register. This IC expands the number of the Arduino's output pins through a chain of eight D flip flops tied to a shared clock pin, equating to 8 outputs to 1 595 shift register.



This clock pin is the first of three important pins used in the IC. It determines when the outputs are shown, so when a rising edge is detected on the clock pin, the data in the form of highs and lows stored on the D flip flops will be shifted out to the right by one bit. This is where the IC gets its name from. For example, if the current data reads 00000010 and a high is sent to the clock pin, the current outputs will read 00000001.

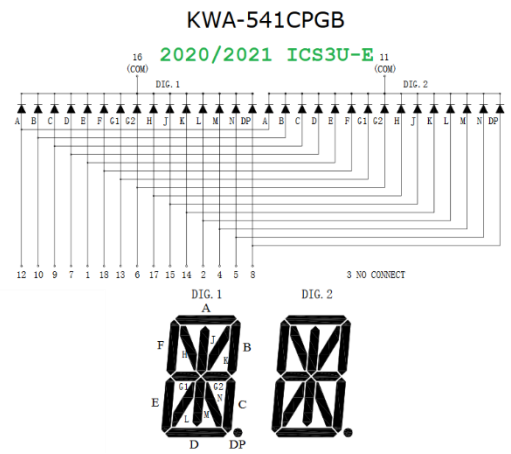
The second pin is the data pin which tells what binary value to store on the flip flops through a high for 1 and a low for 0 on the pin. If a high is presented on the data pin when a rising edge is on the clock pin, the data will be shifted to the right with the addition of a 1 stored on the first flip flop or most significant bit. To put things visually, the current data, 00000010, will be transformed into 10000001 when a high is present on the data pin. Otherwise, when the a low is present, the data, 00000010 will equate to 00000001 when the clock pin detects a high. The combination of highs and lows on the data pin and a constant square wave on the clock pin can then control which outputs pins are high or low.

To send the data on D flip flops onto the output pins, the final significant pin called the latch pin copies the data on the flip flops into the storage/latch register, overwriting any data that was previously programmed. The data in the storage/latch register is then displayed when the output enable pin (OE) is grounded. Through the utilization of these three significant pins, three I/O pins on the Arduino can be exchanged for eight I/O pins. Other pins of the shift register play a more simple and minor role. The shift register clear pin (SRCLR) clears the data from the flip flops and storage/latch while the $Q_{H'}$ pin connects to the data pin of another shift register to daisy chain them and create more output pins. These output pins are represented as Q_A to Q_H with Q_A representing the least significant bit and Q_H signifying the most significant bit.

To integrate the 595 shift register with the Arduino IDE, the shiftOut function is used to program a shift register with ease. The syntax of the function is shown here:

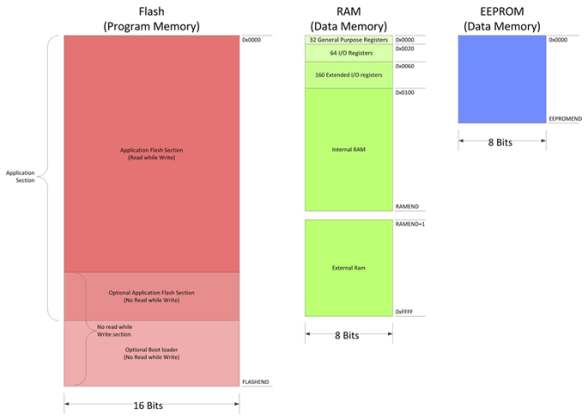
```
shiftOut(dataPin, clockPin, bitOrder, value);
```

The first two parameters mentioned above are the data pin and clock pin. They are defined I/O pin numbers on the Arduino board that are wired to the data pin and clock pin on the 595. The third parameter, bitOrder, tells the shift register what order to shift out the bits. MSBFIRST shifts out the bits to the right, starting from the most significant bit and ending with the least. LSBFIRST switches the order of bits from least significant to most, basically, mirroring the former. The last parameter, value, determines the state of the output pins. Accepted values are 8-bit unsigned integers that can be displayed in binary, octal, decimal, or hex. The accepted values are processed into binary and presented on the output pins when the latch is set high. For example, defining the value as 240, using MSBFIRST, will present the state of the output pins as 1110000 (240 in binary) where Q_H is the most significant and Q_A is the least. Applying the same value but switching out MSBFIRST for LSBFIRST mirrors the output to 0000111.



To display part of this project comes in the form of the KWA-541CPGB common cathode dual 14 pin alphanumeric display. 14 leds for each display allows all upper and lower case characters as well as most symbols to be presented. As shown in the pinout, input pins of both displays are wired together while ground pins are separate. This creates the problem of one display emulating the other since input pins are wired together. However, through a phenomenon known as persistence of vision, different characters on separate displays can be presented even through they share a common input.

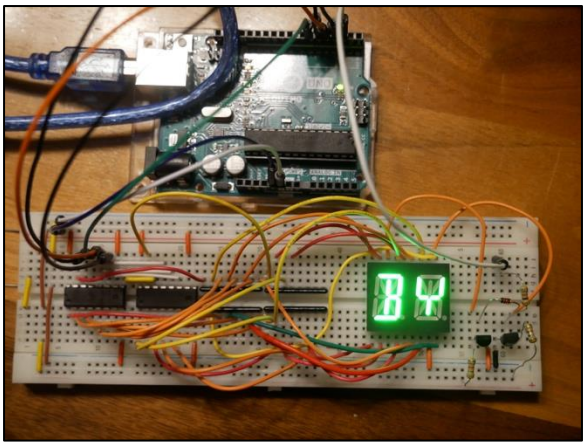
Persistence of vision is an omnipresent optical illusion that the human eye experiences when watching animations. Its an omnipresent phenomenon extending back in history to the phenakitoscope to the modern day screens that humans spend most their day looking at. Human eyes can only process 10 to 12 images per second. Therefore, when flipping through or displaying an animation that cycles images faster than 10 to 12 images per second, the human eye will perceive a series of still images as one continuous moving image. The same concept is applied to the dual 14 pin alphanumeric display where the separate ground pins of each display alternate between sinking current which rapidly switches the state between the two displays. This rate of on and off is extended beyond 12 switches per second to seem like both displays are on. Doing this allows different characters to be displayed; when switching between ground pins, inputs would switch as well to light up a different section of LED's, alternating between characters of the two displays. To quickly sink current from both ground pins, an oscillating circuit resembling the analog oscillator is used with a square wave input from an Arduino.



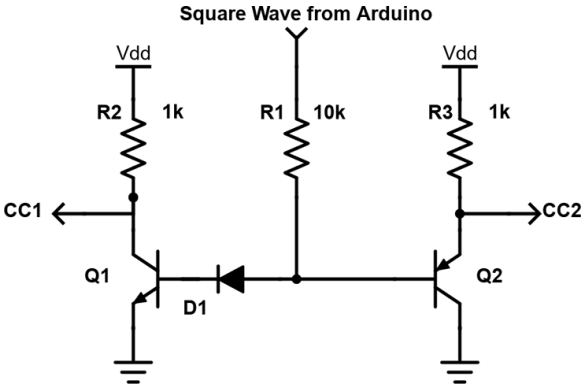
EEPROM is used in this project to store character segment values in binary to send to the shift registers which then light up letters and numbers on the alphanumeric display. These segment values of all letters and numbers are stored in addresses of the EEPROM which can be read from. This read function works in tandem with the shiftOut function to produce characters on the alphanumeric display. As mentioned before, EEPROM is non-volitle; it only needs to be programmed once and it will remember the data forever unless it is overwritten with new data. This trait is extremely useful as it can save loads of processing power and only needs to be run once. The full use of EEPROM with the segment maps can be viewed in the code section below.

To extend one's knowledge and interest of the persistence of vision, two additional features extending the creativity was included. The first, after taking inspiration from Jacob Buchan's own POV project, was the use of scrambling letters to achieve a level of flair. This was achieved by iterating through all letters at a rapid pace to make the code appear as if it scrambles. The second feature included was the programming of a hexadecimal to decimal game to better understand it. This was accomplished through presenting random hex values in an array, converting that hex value into a decimal number for the user to guess and then creating a score. Also a game duration was included by including the `millis()` function. More details of the code are explained in the code section.

Media



The Persistence of Vision project prototyped on a breadboard



The schematic of the oscillation circuit

YouTube video link: https://youtu.be/4KbFu0teXwE

Code

EEPROM

```
// PROJECT  :Write14SegUpperCaseCharacters
// PURPOSE  :Writes a lookup table of 14 segments (word) for the uppercase ASCII
//          :characters to EEPROM
// COURSE   :ICS3U
// AUTHOR   :Xander Chin
// DATE     :2020 10 23
// MCU      :328p
// STATUS   :Working
// REFERENCE:http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#PoV

#include <EEPROM.h>       //requires the support of this Arduino library

// Define the 14-Segment Uppercase Letter LookUp Table...
uint16_t letterSegmentMap[] = {
  //ABCDEFGGHJKLMN__ :Segment order
  0b1110111100000000,  //A
  0b1111000101001000,  //B
  0b1001110000000000,  //C
  0b1111000001001000,  //D
  0b1001111000000000,  //E
  0b1000111000000000,  //F
  0b1011110100000000,  //G
  0b0110111100000000,  //H
  0b1001000001001000,  //I
  0b0111100000000000,  //J
  0b0000111000100100,  //K
  0b0001110000000000,  //L
  0b0110110010100000,  //M
  0b0110110010000100,  //N
  0b1111110000000000,  //O
  0b1100111100000000,  //P
  0b1111110000000100,  //Q
  0b1100111100000100,  //R
  0b1011000110000000,  //S
  0b1000000001001000,  //T
  0b0111110000000000,  //U
  0b0000110000110000,  //V
  0b0110110000010100,  //W
  0b0000000010110100,  //X
  0b0100011100001000,  //Y
  0b1001000000110000   //Z
};

uint16_t numberSegmentMap[] = {
  //ABCDEFGGHJKLMN__ :Segment order
  0b1111110000110000,   //0
  0b0110000000100000,   //1
  0b1101000100010000,   //2
  0b1111000100000000,   //3
  0b0110011100000000,   //4
  0b1011011100000000,   //5
  0b1011111100000000,   //6
  0b1110000000000000,   //7
  0b1111111100000000,   //8
  0b1111011100000000    //9
};

uint8_t letterSizeMap = sizeof(letterSegmentMap) >> 1;  //number of entries in the LUT letters
uint8_t numberSizeMap = sizeof(numberSegmentMap) >> 1;  //number of entries in the LUT numbers
```

```
void setup() {
  Serial.begin(9600);       //requires the support of the Serial Monitor
  while (!Serial);          //pause while the Serial monitor initializes itself

  Serial.println("Note: Highest accessible EEPROM Address: " + String(E2END, DEC));

  //flash the LUT to EEPROM...(once, since EEPROM is non-volatile)
  //Note: The ASCII value is used as the EEPROM address for efficiency
  for (uint8_t x = 0; x <= letterSizeMap; x++) {  //iterate through the LUT array for letters

    EEPROM.write('A' + x << 1, lowByte(letterSegmentMap[x])); //   segment data to EEPROM
    EEPROM.write(('A' + x << 1) + 1, highByte(letterSegmentMap[x]));
  }

  for (uint8_t y = 0; y <= numberSizeMap; y++) {  //iterate through the LUT array for numbers

    EEPROM.write('0' + y << 1, lowByte(numberSegmentMap[y])); //   segment data to EEPROM
    EEPROM.write(('0' + y << 1) + 1, highByte(numberSegmentMap[y]));
  }

  //Echo the first few to the Serial Monitor for confirmation...
  for (uint8_t ch = '0'; ch <= '4'; ch++) {
    Serial.print(EEPROM.read((ch << 1) + 1), BIN);
    Serial.println(EEPROM.read(ch << 1), BIN);
  }

}

void loop() {
}
```

Main

```
// PROJECT    : Persistence of Vision (with hexadecimal game)
// PURPOSE    : To exploit POV to present characters on an alphanumeric display
// DATE       : Oct 24, 2020
// MCU        : 328p/84/85
// STATUS     : Working
// NOTES      : Includes a hexadecimal to decimal game
// REFERENCES : http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#PoV

#include <EEPROM.h>               //includes segment map of characters

//regular code variables
#define clockPin 3
#define dataPin 5
#define latchPin 4
#define squareWave 2
#define enablePin 6
#define scrambleDuriation 15      //how long each letter is shown (ms)

uint8_t ch1;                      //left character
uint8_t ch2;                      //right character
uint32_t split;                   //general purpose split
uint32_t gameSplit;               //split for hex game

String entry;                     //for entering two character
bool hex = false;                 //keeps hex game separate


//HEX game variables
#define countdownDuriation 750    //duriation of countdown for hex game (ms)
#define gameDuration 60000        //how long the game lasts for (ms)
#define ggDuration 7000           //duriation of GG after hex game (ms)

uint16_t hexNumbers[] =           //characters allowed in hex game
 {'0', '1', '2', '3',
  '4', '5', '6', '7',
  '8', '9', 'A', 'B',
  'C', 'D', 'E', 'F'};

uint8_t r1;                       //1st array container
uint8_t r2;                       //2nd array container
uint8_t guess;                    //guess of hex game in decimal
uint8_t score = 0;                //score of hex game

void setup() {

  Serial.begin(74880);     //begin serial (needs to be fast for smoothness)
  Serial.setTimeout(1);    //speeds up serial response time
  while (!Serial);         //waits for serial to initalize

  //initialize pins for output
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(squareWave, OUTPUT);
  pinMode(enablePin, OUTPUT);

  //enables outputs
  digitalWrite(enablePin, HIGH);

  //initializes to certain characters, not random characters
  ch1 = 'H';
  ch2 = 'I';
```

```
  //serial instructions:
  Serial.println("Enter any two numbers or letters");
  Serial.println("Type 'play' to practice your hexadecimal skills");
}

void loop() {

  while (Serial.available() && hex == false) {    //while there is something in
serial

    entry = Serial.readString();    //entry = inputted string
    entry.toUpperCase();            //converts letters to upper
case

    ch1 = entry.charAt(0);          //sets variable to first
character
    ch2 = entry.charAt(1);          //sets variable to second character

    //serial confirmation
    Serial.println("\n" + String("you typed: ") + entry + "\n");

    if(entry == "PLAY") {

      Serial.println("Starting hex game!");   //serial confirmation
      hex = true;                             //blocks sections of code when hex game = on
      startHexGame();
    }

    //serial instructions:
    Serial.println("Enter any two numbers or letters");
    Serial.println("Type 'play' to practice your hexadecimal skills");
  }

  while (!Serial.available() && hex == false) {   //while there is nothing in serial

    displayCharacters(ch1, ch2);
  }

  //if there is no hex game
  if(hex == false) {

    scrambleEffect();
  }


}

//starts countdown for hex game
void startHexGame() {

  //countdown until start of hex game
  for(ch2 = '3'; ch2 >= '1'; ch2--) {   //EEPROM value decreases to show ajacent characters

    split = millis();
    while(millis() - split < countdownDuriation) {    //displays characters for duriation

      ch1 = '0';
      displayCharacters(ch1, ch2);
    }
  }
```

```
  split = millis();
  while(millis() - split < countdownDuriation) {      //displays characters for duriation

    ch1 = 'G'; ch2 = 'O';

    displayCharacters(ch1, ch2);
  }

  scrambleEffect();
  hexGame();
}

//code for hex game
void hexGame() {

  randomSeed(analogRead(A0));   //random seed from a floating pin

  r1 = random(15);              //calls number from seed (0 to 15)
  r2 = random(15);              //calls number from seed (0 to 15)
  ch1 = hexNumbers[r1];         //character equals allowed hex vaule
  ch2 = hexNumbers[r2];         //character equlas allowed hex vaule

  gameSplit = millis();
  while(millis() - gameSplit < gameDuriation) {   //while duration of game

    while(Serial.available()) {         //while there is something in serial
      guess = Serial.parseInt();

      if(guess == (r1 << 4) + r2) {     //if guess equals hex number in decimal
        Serial.println("correct!");     //serial confirmation

        scrambleEffect();
        score++;                        //increase score by 1

        r1 = random(15);                //calls number from seed (0 to 15)
        r2 = random(15);                //calls number from seed (0 to 15)
        ch1 = hexNumbers[r1];           //character equals allowed hex vaule
        ch2 = hexNumbers[r2];           //character equlas allowed hex vaule
      }

      else {                            //if guess is not correct
        Serial.println("incorrect!");   //serial confirmation
      }
    }

    //while there is nothing in serial and game duriation is active
    while(!Serial.available() && (millis() - gameSplit < gameDuriation)) {
      displayCharacters(ch1, ch2);
    }
  }

  //serial confirmation
  Serial.println("\n" + String("game over!"));
  Serial.println("score: " + String(score) + "\n");

  split = millis();
  while(millis() - split < ggDuriation) {   //duration of GG

    ch1 = 'G'; ch2 = 'G';                   //sets characters to GG (good game)
    displayCharacters(ch1, ch2);
  }

  scrambleEffect();
  ch1 = 'H'; ch2 = 'I';             //displays characters after hex game
  hex = false;                      //allows access to regular code

  while(Serial.available()) {       //clears characters entered in serial
       Serial.read();
  }
}
```

Reflection

This was, by far, my most stressful, my most time consuming, and my most invested project that I have done so far. I have been working on this since Thursday night, non-stop until now. I spent most of my time coding this because I wanted to gain a deeper understanding of this project and a produce something creative, which I have successfully accomplished through a scramble effect and a hexadecimal game. Though, the perfectionist part of me felt that it fell short. I found that the quality of my DER and video, the two things I prided myself the most in, suffered; my submission is not where I want it to be. I did not have time to edit my video to further clarify the points I made and I feel like my DER is missing a few key aspects. Right now its 15 miniutes till midnight. From when this project first got introduced, I wanted everything about it to be perfect, similar to the rest of my projects. But, through this submission, I have learnt that humans aren't perfect and we make mistakes, however, we can improve. I am definetly going to reflect on this project and build off from it from this point forward, always trying to learn from past mistakes. I am truly sorry that this DER was not up to standards.

Project 2.3: Ask UNO

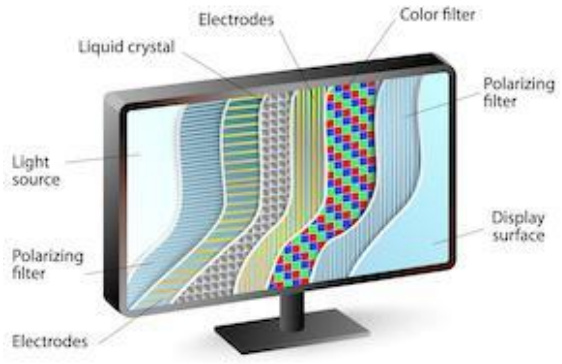
Purpose

The purpose of this project is to demonstrate hardware wired communication and processing between two Arduinos to emulate a wired smarthome device

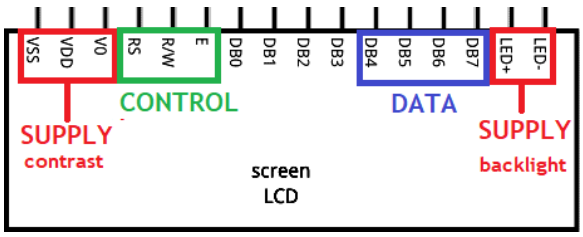
Reference

http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#AskUNO
https://www.explainthatstuff.com/lcdtv.html
https://www.allaboutcircuits.com/technical-articles/back-to-basics-the-universal-asynchronous-receiver-transmitter-uart/

Procedure

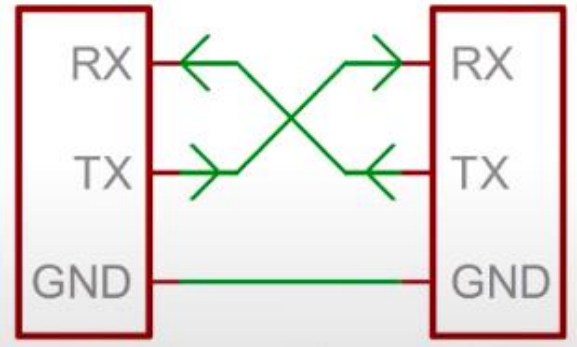


The main part of this project is demonstrated using the very common liquid crystal display or LCD for short used in TV's, computer monitors, calculators, and digital watches. LCD's work a lot differently than traditional LED displays. In LCD screens, there are individual pixels that can block or let light through from a backlight, allowing characters to be formed on the screen. These pixels have polarized filters, allowing light to pass through when rotated one way but blocking all light when rotated another way. To rotate these pixels, liquid crystals are used, hence the name liquid crystal display. These liquid crystals have disctinct forms pertaining to electricity being applied. If no current is applied, the crystal assumes a twisted structure, blocking the back light and when current is applied, the crystal straightens out and lets light pass through. This is how letters, numbers and characters are formed on an LCD display.



The specific LCD display implemented in the project is a 16x2 LCD screen, with each block containing a 5x8 rectangle of pixels. There are a total of 16 pins to utilize all of its functions however, only 6 will be controlled through a microcontroller to display information in 4-bit mode. 4-bit mode basically shifts out data 4-bits at a time using 4 of the DB pins while 8-bit mode, requiring all DB pins, shifts out data 8-bits at a time, speeding up transmission. A pinout of the device is shown above. On the software side, an LCD library containing fuctions to drive the LCD screen is used. This library simplifies and shortens code, allowing easy programming of the LCD display using straightforward functions.

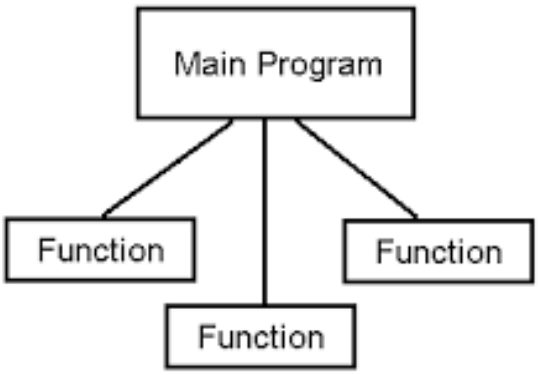
Communication is essential, especially nowadays. It is what allowed us to understand each other and to work together to contruct the civilization that humans live in today and now it plays a role in keeping individuals and societies connected through the communication protocols of technology. There are two main ways of communication in devices: wired communication, involving physical links for communication and wirelss communication, involving non-physical ways.



In this project, data is sent using wired communication in the form of universal asynchronous reception and transmission (UART) to other serial devices like computers or other MCUs. In many Arduinos like the NANO and UNO there is only one built in UART channel which are pins 0 for Rx (receiving) and 1 for Tx (transmitting), reserved for reading to and writing from the serial monitor. Unfortunatley, this leaves no space for more UART connections between other MCUs as one UART channel can only support communication between one serial device. This is fixed by putting the software serial library into action, which converts specified digital I/O pins into Rx and Tx communicator pins for more communication. In order to establish a link between two MCUs, the Rx pins connects with the Tx pins of opposite MCUs and a common ground is shared.

Combining these new concepts leads to the Ask UNO project where two Arduino devices, the NANO and UNO, use UART communication with each other to solve arithmetic inputted by the user in the serial monitor. The answer to the specified arithmetic expression is then displayed on LCD screens. The NANO acts as the host, taking in data from the serial monitor and relaying it to the UNO assistant, which processes the data and displays an answer on the LCD screen. This is efficiently accomplished through the use of modular programming.

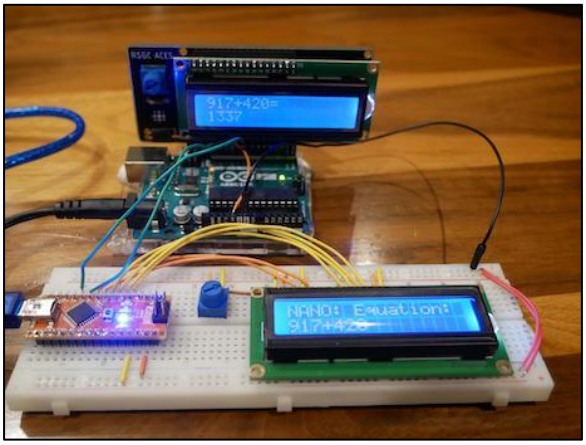
Parts Table	
Quantity	Description
1	Computer
1	9V power source
1	USB A to USB B cable
1	USB A to USB C cable
1	Arduino UNO
1	Arduino NANO
2	16 × 2 LCD display
1	LCD Appliance PCB
1	330 Ω fixed resistor
2	10 kΩ potentiometers
1	2 right angle male header pins
1	6 right angle male header pins
1	2 right angle female headers
1	16 × 1 straight female header



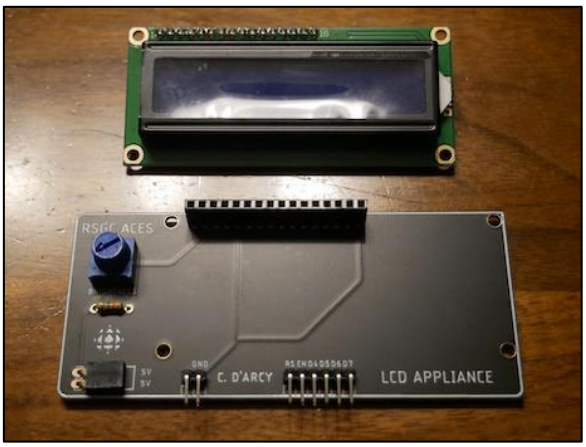
Modular programming is described as code that is separated into parts which can be used interchangeably to preform something specific. It breaks down complex code into simpler more manageable chunks called functions in order to increase organization, flexibility, and variety in use. Functions take in a value, process it, and outputs another value, so breaking down code into functions allows the programmer to select certain functions for use. For example, the LCD and software serial library use functions and modular programming so that certain aspects of the library can be used in an organized way. The same idea is implemented in the project code sketches which can be viewed in the code section.

The basic task of this project involves inputting expressions consisting of `a?b` where `a` and `b` are single digit operands and `?` is the operator consisting of addition, subtraction, multiplication, division and modulus. The output is then expressed. To extend this basic task, the code was modified to allow multi-digit operands, a safety net was introduced to defend against undefined answers such as division by 0, malformed expressions are not sent to the UNO, and more operators such as the bitwise operators and random functions were made available to use.

Media



Setup of the NANO host and the UNO assistant



The LCD appliance PCB providing easy and organized usage

YouTube video link: https://youtu.be/IRZPm464c-M

Code

NANO

```
// PROJECT    : Ask UNO - NANO Host
// PURPOSE    : LCD displays and UART Communication
// DATE       : November 21, 2020
// MCU        : 328p/84/85
// STATUS     : Working
// NOTES      : None
// REFERENCES : http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#AskUNO

#include <SoftwareSerial.h>   //Software serial library
SoftwareSerial chat(8, 9);    // RX, TX

#include <LiquidCrystal.h>    //LCD Library
#define LCD_COLUMNS 16        //Number of columns in Character LCD screen
#define LCD_ROWS    2         //Number of rows on LCD screen

//LCD setup
uint8_t RS = 7, EN = 6, D4 = 5, D5 = 4, D6 = 3, D7 = 2;
LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);

uint32_t timeElapsed;
#define timeOutTime 5000    //in (ms)

void setup() {
  lcd.begin(LCD_COLUMNS, LCD_ROWS);   //initialize LCD screen

  displayData("NANO: Equation?", "Waiting...");

  chat.begin(9600);         //initialize chat
  chat.setTimeout(10);      //speed up chat
  while (!chat);

  Serial.begin(9600);       //initialize serial
  Serial.setTimeout(10);    //speed up serial
  while (!Serial);

  tutorial();
}

void loop() {
  while(Serial.available() > 0) {   //while there is something in serial
    timeElapsed = millis();

    String equation = Serial.readString();

    //find where the operator position is
    uint8_t pos = 0;
    while(pos < equation.length()) {
      if(isOperator(equation.charAt(pos))) break;
      pos++;
    }

    //separate operands
    String stringA = equation.substring(0, pos);
    String stringB = equation.substring(pos+1, equation.length());
    uint32_t intA = equation.substring(0, pos).toInt();
    uint32_t intB = equation.substring(pos+1, equation.length()).toInt();
```

```
    //defend against malformed expressions:
    //note: operands cannot be negative numbers
    if(!stringA.equals((String)intA) || !stringB.equals((String)intB)) {
      displayData("Malformed:", equation);
      break;
    }

    //defend against min > max in random function
    if(equation.charAt(pos) == 'r' && intA > intB) {
      displayData("Malformed:", "a<b when using r");
      break;
    }

    chat.print(equation);                       //sends data to the UNO
    displayData("NANO: Equation:", equation);   //confirmation of equation
  }

  //displays after a certain amount of time has passed
  if(millis() - timeElapsed > timeOutTime) {
    displayData("NANO: Equation?", "Waiting...");
  }
}

//function to display data on LCD:
void displayData(String s0, String s1) {

  lcd.setCursor(0, 0);                    //cursor at first row and column
  lcd.print(s0 + "                ");     //clears first row and prints s0
  lcd.setCursor(0, 1);                    //cursor at second row first column
  lcd.print(s1 + "                ");     //clears second row and prints s1
}

//if character is an operator
boolean isOperator(char ch) {

  String operators = "+-*/%&<>^|r";    //allowed operators
  return operators.indexOf(ch) >= 0;    //returns true or false
}

//how to operate ask UNO
void tutorial() {

  //F macro used to save RAM
  Serial.println(F("Welcome to Ask UNO! Type an equation in the form: a?b"));
  Serial.println(F("The answer will be displayed on the UNO LCD \n"));
  Serial.println(F("a = any positive number"));
  Serial.println(F("b = any positive number \n"));
  Serial.println(F("Allowed equations:"));
  Serial.println(F("a+b \t addition"));
  Serial.println(F("a-b \t subtraction"));
  Serial.println(F("a*b \t multiplecation"));
  Serial.println(F("a/b \t division"));
  Serial.println(F("a%b \t modulus"));
  Serial.println(F("a<b \t bitshift left"));
  Serial.println(F("a>b \t bitshift right"));
  Serial.println(F("a&b \t bitwise AND"));
  Serial.println(F("a^b \t bitwise XOR"));
  Serial.println(F("a|b \t bitwise OR"));
  Serial.print(F("arb \t random number between a and b. "));
  Serial.println(F("a needs to be less than or equal to b \n"));
  Serial.print(F("Arduino operators: "));
  Serial.println(F("https://www.arduino.cc/reference/en/#structure"));
}
```

UNO

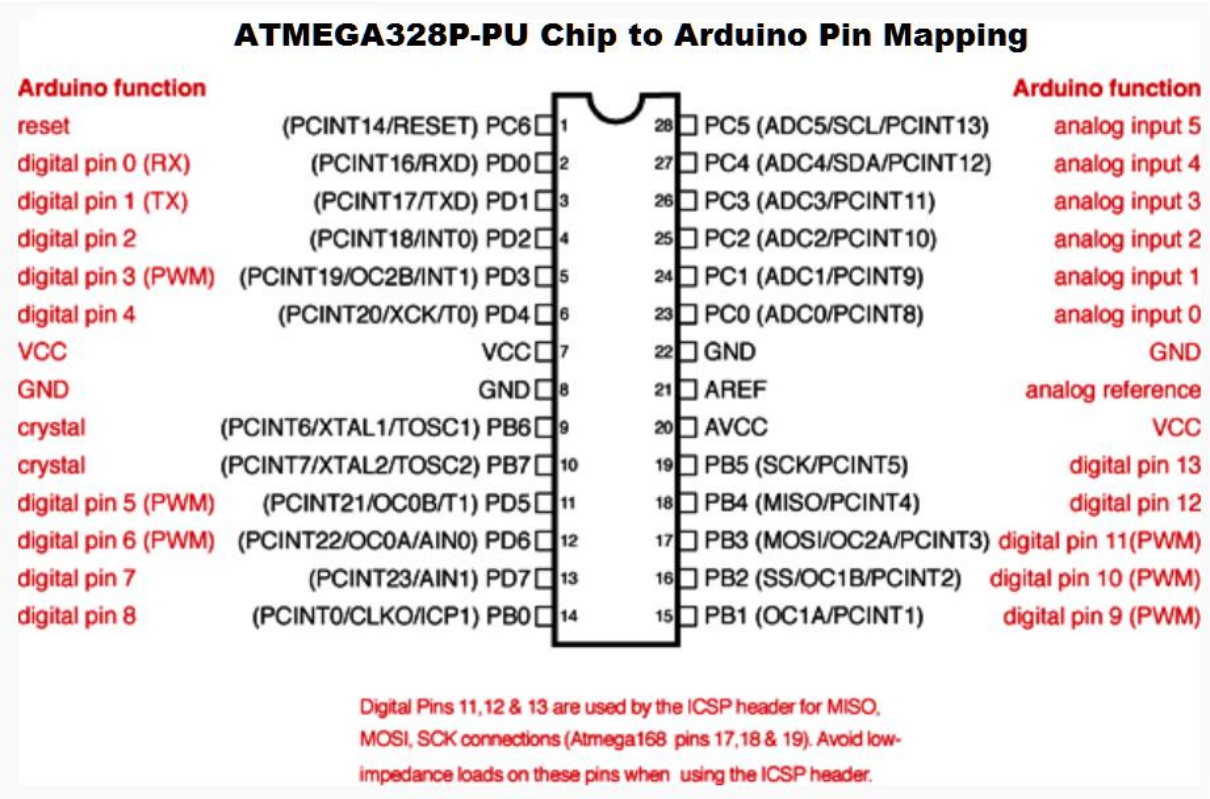
```
// PROJECT    : Ask UNO - UNO Assistant
// PURPOSE    : LCD displays and UART Communication
// DATE       : November 21, 2020
// MCU        : 328p/84/85
// STATUS     : Working
// NOTES      : None
// REFERENCES : http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#AskUNO

#include <SoftwareSerial.h>    //Software serial library
SoftwareSerial chat(8, 9);     //Rx Tx

#include <LiquidCrystal.h>     //LCD Library
#define LCD_COLUMNS 16         //Number of columns in Character LCD screen
#define LCD_ROWS    2          //Number of rows on LCD screen

//LCD setup
uint8_t RS = 7, EN = 6, D4 = 5, D5 = 4, D6 = 3, D7 = 2;
LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);

char mathOp;    //needed in different functions

uint32_t timeElapsed;
#define timeOutTime 5000    //in (ms)

void setup() {
  //initialize LCD
  lcd.begin(LCD_COLUMNS, LCD_ROWS);
  displayData("UNO Assistant", "Waiting...");

  chat.begin(9600);             //initialize chat
  chat.setTimeout(10);          //speed up chat
  while(!chat);
}

void loop() {
  while(chat.available()) {   //while there is something in chat UART
    timeElapsed = millis();

    String equation = chat.readString();

    //find where operator position is
    uint8_t pos = 0;
    while(pos < equation.length()) {
      if(isOperator(equation.charAt(pos))) {
        mathOp = equation.charAt(pos);
        break;
      }
      pos++;
    }

    //separate operands
    uint32_t operandA = (equation.substring(0, pos)).toInt();
    uint32_t operandB = (equation.substring(pos+1, equation.length())).toInt();

    //defend against division by 0
    if(mathOp == '/' && operandB == 0) {
      displayData("UNO Assistant", "ERROR");
      break;
    }

    //convert to string then display
    String answer = (String)solve(operandA, mathOp, operandB);
    displayData(equation + "=", answer);
  }
```

```
  //displays after a certain amount of time has passed
  if(millis() - timeElapsed > timeOutTime) {
    displayData("UNO Assistant", "Waiting...");
  }
}

boolean isOperator(char ch) {

  String operators = "+-*/%&<>^|r";    //allowed operators
  return operators.indexOf(ch) >= 0;    //returns true or false
}

//function to display data on LCD
void displayData(String s0, String s1) {
  lcd.setCursor(0, 0);                  //cursor at first row and column
  lcd.print(s0 + "                ");   //clears first row and prints s0
  lcd.setCursor(0, 1);                  //cursor at first row and second column
  lcd.print(s1 + "                ");   //clears second row and prints s1
}

//returns number value a?b
int32_t solve(uint32_t a, char mathOperator, uint32_t b) {

  randomSeed(analogRead(A0));   //for random function

  //case values = ASCII values
  switch(mathOperator) {
    case 37: return a%b; break;
    case 38: return a&b; break;
    case 42: return a*b; break;
    case 43: return a+b; break;
    case 45: return a-b; break;
    case 47: return a/b; break;
    case 60: return a<<b; break;
    case 62: return a>>b; break;
    case 94: return a^b; break;
    case 114: return random(a, b+1); break;
    case 124: return a|b; break;

    default: break;
  }
}
```

Reflection

Compared to the persistence of vision project, the ask UNO project felt a lot easier and less stressful. I knew how to organize my time better and I learned from my mistakes, which allowed me to fully look over and finish the project. However, there were some stressful moments during this weekend which is normal, as this course is very challenging and takes a toll on the brain, especially if you have been working non-stop since Friday morning. Though, I do believe as time goes on, these stressful moments will disappear as I get used to the curriculum. Overall, I really enjoyed programming, learning, and playing with the LCD screens as they are so ubiquitous in everyday life. It was also cool to learn the parts of an LCD screen and how the science behind it worked and communication between Arduino's was a great thing to experience as it is implemented in most everyday devices and it opens up the doors to many types of new and creative projects.

Project 2.4.1: Breadboard ATmega328P



Purpose

The purpose of this project is to develop an in-system programmable device to extend the versatility and use of creating circuits and projects.

References

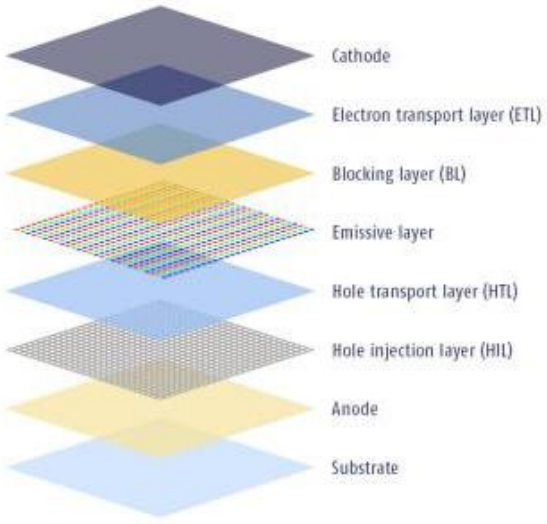
http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#standalone
https://www.oled-info.com/oled-introduction
https://www.oled-info.com/oled-technology
https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/
https://www.electronics-tutorials.ws/oscillator/crystal.html
https://www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoISP
https://www.arduino.cc/en/reference/SPI

Procedure

The in-system programmable device is a simple 555 timer frequency reader that displays information on an oscillating signal from the timer. This information shown are the high, low and total time of the wave, the duty cycle, and the frequency which is relayed to an OLED display using I2C communication. Also, a real time LED displays when the wave is low or high. This is very useful because the resistor capacitor (RC) pairs can be easily mixed and matched to get the desired frequency, duty cycle, and total time. Also, this device gives very accurate readings compared to an online 555 timer calculator since resistors and capacitors can vary in their values, therefore differentiating the output wave from the calculated one.

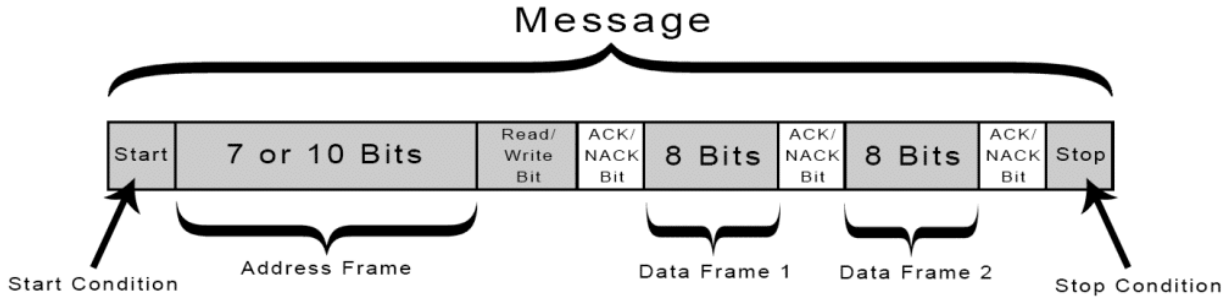
Parts Table	
Quantity	Description
1	Computer
1	AVR pocket programmer
1	AVR breakout board
1	USB A to USB C cable
1	ATmega328P
1	16 MHz quartz crystal
2	20 pF ceramic capacitors
1	10 kΩ fixed resistor
1	Momentary PBNO
1	DC power jack
1	7805CT 5V regulator
1	9V battery
1	555 Timer IC
1	3 mm red LED
1	680 Ω fixed resistor
1	128 × 64 OLED display

The 555 timer is a basic all round IC that outputs a varying square wave using RC pairs. It was extensively used in the ICS2O course hence why this device was created. More details on the 555 timer can be viewed in earlier sections of the DER.



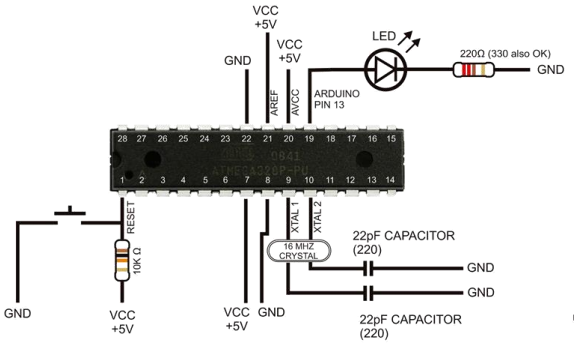
The OLED (Organic Light Emitting Diodes) display is a new and widely used display device similar but vastly superior to the LCD screen. It is thinner, more efficient, bendable, provides better image quality, and does not require a backlight. This is why it is used in the screens of most modern technology companies such as Apple and Samsung. An OLED is comprised of carbon-based material and does not contain any harmful metals hence the organic label. To put it simply, the material emits light when electricity is passed through. However, there are more important layers to protect the material such as the substrate and backpane, and the encapsulation layer proctecting the frontpane containing the carbon-based material and electrodes from oxygen. As time passed, more layers were applied to make them more efficient and durable.

The OLED display used in this project is a 128 by 64 screen using I2C communication. Like UART, I2C communication is a very simple only using two wires; one for the SCL clock signal and the other for the SDA data signal. Therefore, most devices only require four pins to operate, the other two being power and ground. To send data, I2C uses messages comprised of frames as shown below:



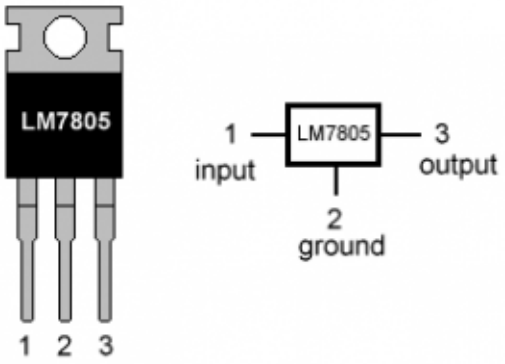
The start frame is initialized by the SDA switching from high to low before the SCL does the same. The address frame tells the data the specific device address to go to or receive from depending on the next frame called the read/write bit. After that comes the actual data in the form of data frames 1 and 2. The frames that surround these data frames are confirmation frames called ACK/NACK (acknowledge/no acknowledge) bits to verify that the data was send/received properly. Finally, the stop frame closes the message by the SDA line switching from low to high after the SCL line does the same.

To create an in system programmable device, the ATmega328P needs to be implemented into a breadboard. However, this MCU needs some basic external peripherals in order to function properly. These include a crystal and capacitors for the clock signal and a pull up resistor to reset the MCU. The crystal capacitor formation creates an oscillation circuit producing a square wave from the quartz crystal at a frequency of 16 MHz. This frequency governs the flow of code and all timing aspects that is used in the ATmega328P. The basic setup it of a breadboard is shown on the right with the inclusion of a blink LED.

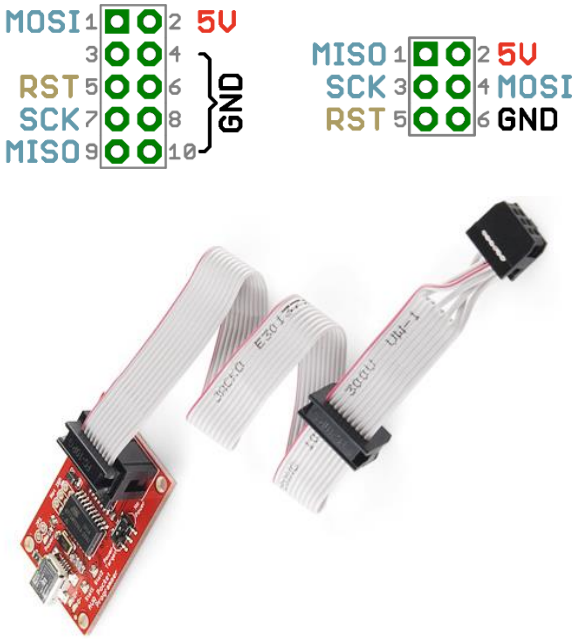


An important thing to note is that the ATmega328P only accepts 5V power so a 5V regulator is needed to turn 9V from a battery into 5V. The 5V regulator used in the project is the 7805 which has 3 pins. The leftmost one is the input voltage, the middle is groud and the rightmost pin is the output voltage of 5V.

LM7805 PINOUT DIAGRAM



The ATmega328P is not like any other IC where it performs a specific function right out of the factory. The user needs to implement the bootloader, which is the communication bridge between the Arduino IDE and the ATmega328P, along with some code in order to tell it what to do. There are a few methods to do this but the method used in this project is from the use of an AVR Programmer shown on the right. This handy board with the help of an AVR breakout board attaches the nessecary pins to the chip shown to the right. These pins are the MISO (master in slave out), clock, reset, power, MOSI (master out slave in), and ground. Once connected, the bootloader can be burned to the MCU and the code can be uploaded using the programmer option in the Arduino IDE.



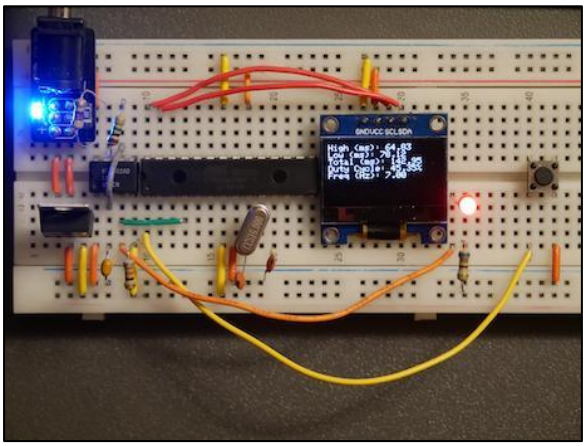
This communication between software and hardware of the IDE and ATmega328P follows the standard Serial Peripheral Interface (SPI) by using the MISO, MOSI and clock line. The MISO pin sends data from the slave to the master, the MOSI pin sends data to the peripheral devices, and the clock line synchronizes the data transmission. This form of hardwired communication is used for fast transmission over short distances such as reading and writing from an SD card.

The code for the project is relatively simple with no new major concepts introduced. The only new features are the Adafruit OLED display library implemented for easy use of the OLED screen and a new function called `pulseIn` which times the high and low states of a pin in microseconds. Here are the parameters of the function:

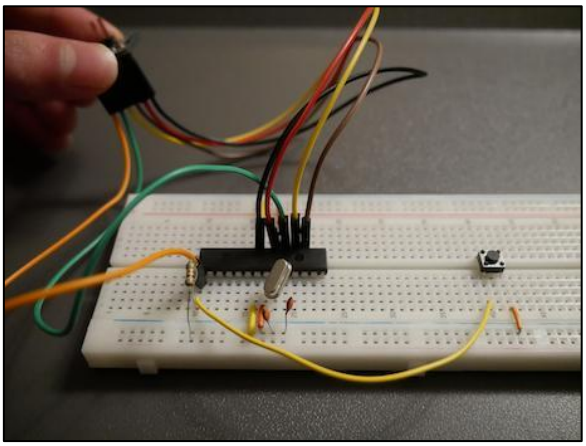
```
pulseIn(pin, value, timeout);
```

The `pin` parameter defines the pin of the ATmega328P to read from, `value` is defined as HIGH or LOW which tells the function what value to time, and `timeout` is an optional parameter that tells the function when to stop timing after the defined number of miliseconds. The default is 1 second. There is an alternative function called `pulseInLong` which contains the same parameters but is better at taking longer pulses. The new Adafruit OLED library has many functions to easily use the OLED display but all of the functions that are utilized are self explanatory so there is no need for detail.

Media



The ATMega Breadboard ISP device setup



The setup for uploading code using male to female wires

YouTube video link: https://youtu.be/4NGWaetlcpA

Code

```
// PROJECT    : ATmega328P Breadboard 555 Timer Reader
// PURPOSE    : Extending the versatility of the ATmega328P
// DATE       : December 18, 2020
// MCU        : 328p/84/85
// STATUS     : Working
// NOTES      : Requires 1 external capacitor and 2 external resistors
// REFERENCES : http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#standalone

//I2C library
#include <Wire.h>

//OLED libraries
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define displayWidth 128    // OLED display displayWidth, in pixels
#define displayHeight 64    // OLED display displayHeight, in pixels
#define objectWidth 23      //object display displayHeight in pixels
#define objectHeight 16     //object display displayHeight in pixels

#define OLED_RESET 4  //required reset declaration

//display and I2C communication setup
Adafruit_SSD1306 display(displayWidth, displayHeight, &Wire, OLED_RESET);

//information variables
uint32_t offTime;
uint32_t onTime;
uint32_t totalTime;
float frequency;
float dutyCycle;
```

```
#define readPin 2          //input pin from the 555 timer
#define timeout 90000000   //timeout limit in microseconds

//bitmap of the object in hex
const uint8_t PROGMEM timer555[] = {
  0x39, 0xe7, 0x9c, 0xff, 0xff, 0xfe, 0x80, 0x00,
  0x02, 0x80, 0x00, 0x02, 0x80, 0x00, 0x02, 0x8f,
  0xbe, 0xfa, 0xc8, 0x20, 0x82, 0xaf, 0x3c, 0xf2,
  0xa0, 0x82, 0x0a, 0xc0, 0x82, 0x0a, 0x8f, 0x3c,
  0xf2, 0x80, 0x00, 0x02, 0x80, 0x00, 0x02, 0x80,
  0x00, 0x02, 0xff, 0xff, 0xfe, 0x39, 0xe7, 0x9c
};

//animation variables
uint8_t xSpeed = 1;
uint8_t ySpeed = 1;
uint8_t x;
uint8_t y;

void setup() {
  pinMode(readPin, INPUT);                      //input pin is defined
  Wire.begin();                                 //begin I2C transmission
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);    //begin display

  setupText(2, 0, 0);                  //setup text
  display.println(F("Waiting..."));    //print into memory
  display.display();                   //display on OLED
  getData();                           //get 555 timer information
}

void loop() {
  setupText(1, 0, 0);
  display.print(F("High (ms): "));
  display.println(onTime * 0.001);      //display is in milliseconds
  display.print(F("Low (ms): "));
  display.println(offTime * 0.001);     //display is in milliseconds

  display.print(F("Total (ms): "));
  display.println(totalTime * 0.001);   //display is in milliseconds

  display.print(F("Duty Cycle: "));
  display.print(dutyCycle);
  display.println(F("%"));

  display.print(F("Freq (Hz): "));
  display.println(frequency);

  display.display();
  getData();
}

void getData() {
  uint64_t currentTime = micros();

  onTime = pulseInLong(readPin, HIGH, timeout);   //number of microseconds when HIGH
  offTime = pulseInLong(readPin, LOW, timeout);   //number of microseconds when LOW

  if(micros() - currentTime > timeout)  timeoutAnimation();

  totalTime = onTime + offTime;               //
  dutyCycle = float(onTime)/totalTime*100;    //
  frequency = 1000000.0/totalTime;            //frequency in Hz
}
```

```
void setupText(uint8_t s, uint8_t x, uint8_t y) {
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(s);
  display.setCursor(x, y);
}

void timeoutAnimation() {
  randomSeed(analogRead(A0));
  x = random(displayWidth - objectWidth);      //random y position
  y = random(displayHeight - objectHeight);    //random x position

  //infinite loop (each iteration is a frame)
  while(true) {
    display.clearDisplay();

    //display text in the center
    setupText(1, 44, 28);
    display.println(F("Timeout"));

    //creates an object
    display.drawBitmap(x, y, timer555, objectWidth, objectHeight, SSD1306_WHITE);

    x = x + xSpeed;   //move by xSpeed
    y = y + ySpeed;   //move by ySpeed

    //checks if the object hits the edge of the screen
    if(x + objectWidth >= displayWidth || x <= 0) xSpeed = -xSpeed;
    if(y + objectHeight >= displayHeight || y <= 0) ySpeed = -ySpeed;

    display.display();
  }
}
```

Reflection

I felt like this project was the tamest one compared to the others as ample was available to design and program the device. Also, there were not many complications or new and confusing things to learn. Because of this, I decided to try implementing some advanced code such as implementing the parody of the DVD bouncing logo when the square wave of the 555 timer is too long and when the device is first calculating the time of the pulses. However, after many tries I couldn't get the animation to run when loading the data in when using Timer1 interrupts. This left me disappointed and wondering why it wouldn't function. So I researched this and found that something about I2C communication conflicted with the interrupts so I left it at that. However, over the Christmas holidays I will try and find some workaround to do this. After creating this device on a breadboard, I would love to have the opportunity to put this on a perma-proto board to prepare for the upcoming ISPs. I just hope that I won't die of stress when Project 2.4.2 Perma-Proto ATmega328P rolls around.

Project 2.4.2: Perma-Proto ATmega328P

Purpose

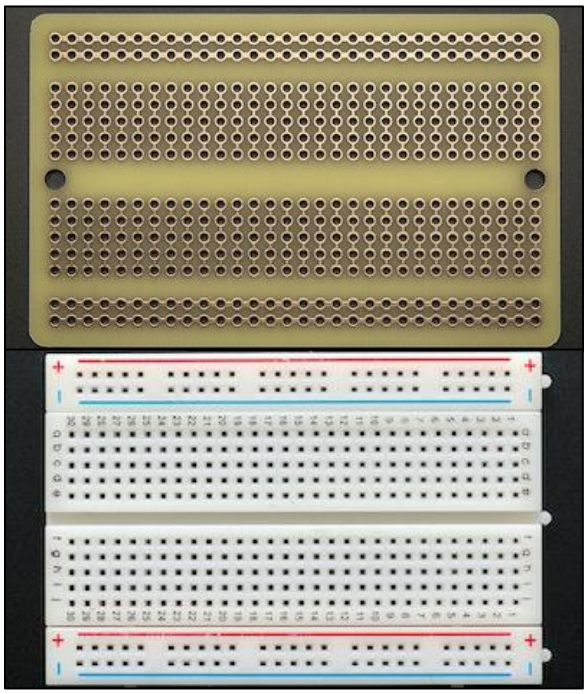
The purpose of this project is to transfer the breadboard standalone circuit to a perma-proto board to create a permanent circuit and to practice soldering skills as well as our planning, carefulness and attention to detail.

References

http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#permaproto

Procedure

The project goal was to convert the temporary prototype 555 timer reader on the breadboard to a permanent circuit on a 30 column perma-proto board whose size allows it to fit into a small metal tin. This presented the specific challenge of fitting in the circuit onto a reduced set size of 30 columns instead of 60. This perma protoboard I used is shown on the right which is also juxtaposed with a half breadboard to compare the differences. As shown, the hole sets and wiring are nearly identical the only difference being more holes in the power rails of the perma-proto board and a shortened distance between them and the main holes. Also, when working with the board, there is much more flexibility as the connections can be disconnected using a box cutter to strip the copper wiring. Wires can also be routed on both sides.

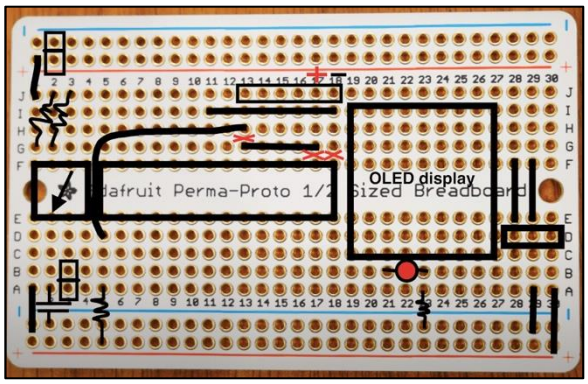


Luckily, I went to the effort of trying to fit the breadboard prototype into 30 columns in the last part of the project, however, I still had to make a few changes, mainly to accommodate the potential metal tin case. First, I removed the reset button and used a big mountable PBNO to save space. Then, I decided to rearrange the circuit by moving the 5 V regulator to the right and putting the squarewave signal LED under the OLED display to further compact the circuit.

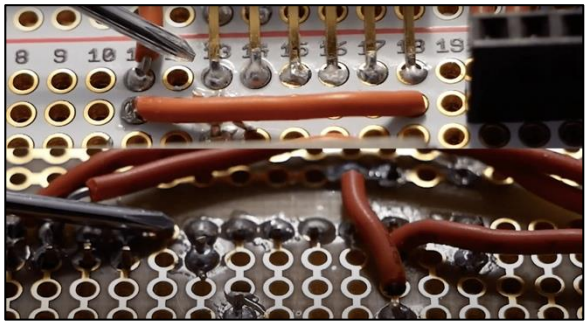
I also made a few more part additions to the already full perma-proto board to increase performance and ease of use as soldering parts can sometimes have adverse effects if not careful. Firstly, I added a 1 MΩ resistor across the crystal pins which would achieve a more consistent square wave from the crystal when powered on. Next I added a 10 µF capacitor across the power rail, making sure to position it as close to the source of power as possible in order to maximize the smoothing out of power spikes. A similar approach was made on the 5 volt regulator by also adding a 10 µF capacitor across the 9 volt input pin to ground and a 0.1 µF capacitor across the 5 volt output pin also to ground. Finally, breakable female headers were put in place of the 555 timer resistors and capacitors to easily mix and match for a desired frequency and a 1 × 2 headers were added for outsourcing the square wave and power pins.With this rearrangement I put it together on the breadboard to make sure it worked and then planned the circuit on the perma-proto board.

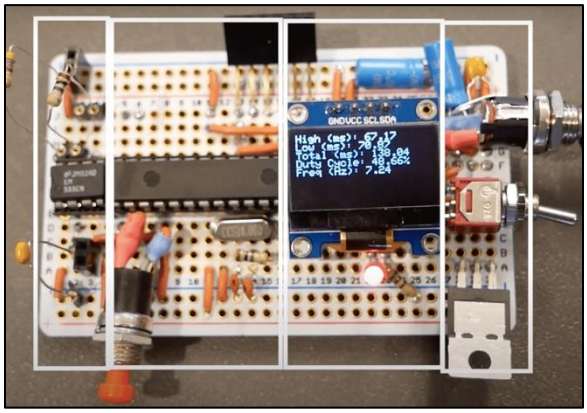
Parts Table	
Quantity	Description
1	Previous project components
1	Half sized perma-proto
1	1 × 6 female header
1	1 × 4 female header
2	1 × 2 female headers
1	8-pin chip seat
1	28-pin chip seat
2	10 µf capacitors
1	0.1 µf capacitor
1	1 MΩ fixed resistor
1	Mountable PBNO
1	Mountable DC power jack
1	Mountable toggle switch

To start planning, I took a picture of the board to roughly draw out the components and arrangements. I also had to decide where to make cuts in the connections of the proto board to be able to upload code directly with the SAPP using a 1 × 6 bendable female header instead of individual female wires to simplify the process of flashing ocde. Once a rough plan was in place, I soldered everything over a few days, changing up minor details of the plan after some reconsiderations between soldering sessions. The rough drawing of the plan can be seen on the right.



The soldering process was tough but not as hard as soldering my grade 10 ISP, the analog traffic light. I learned a lot of lessons and skills from that culminating project which helped me to improve my work and the result for this project. For example, I learned that cutting your own wire was a lot neater and cleaner than using breadboard wires. I had also learned to solder up both sides of the perma-proto board where wires poked through to beef up the physical strength and connectivity which can be seen on the right.



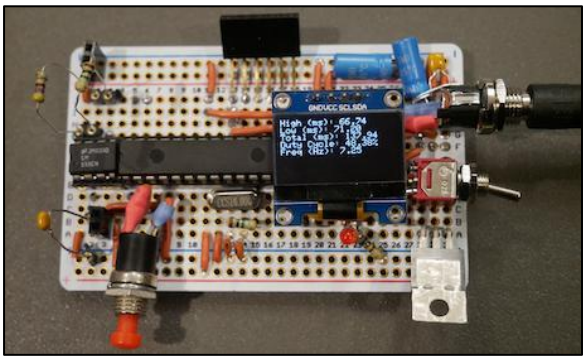


To ease up the process, I divided the circuit into sections shown on the right and soldered the sections individually while taking breaks in between. I also made sure to have my digital multimeter handy to check for connectivity to components to catch any errors after soldering each component. Testing the different stages of the circuit was crucial to locating any problems and fixing them. One such example was with a test where the 555 timer stoped functioning. Since the majority of components were not soldered on, I knew the problem lay in the timer itself or the power rails. I checked the connectivity of the bottom power rails and sure enough I found the connectivity to be dodgy so I simply added more solder on the bottom as well as the top of the wires connecting the two power lines together.

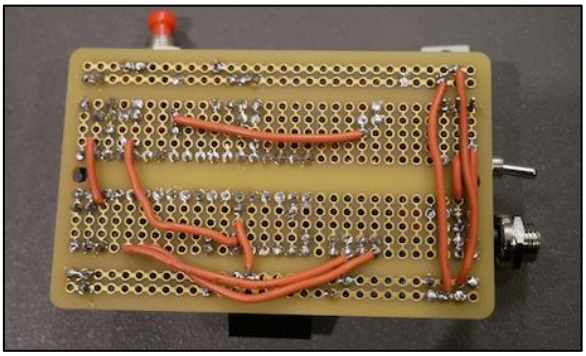
After the soldering process the device worked almost flawlessly. Everything worked as before though this time, the device can switch on an off using the mounted toggle switch, the square wave can be utilized from the output, and the resistors and capacitors of the 555 timer can be swapped. Uploading code became much simpler by sticking the AVR breakout board in the 1×6 female header displayed on the picture instead of using the loopy female jumper wires of the last project. Also, the chip seats and female headers allow any of the major components such as the ICs or OLED display to be replaced if they ever get damaged. No changes to the software were made in this project.

In the end, there were a few minor issues. Firstly, there is some flickering of power when moving the on/off switch around on which suggests a soldering issue. As I get more practice soldering, these kinds of issues will start to disappear. Also, the OLED display can glitch out when handling the the board from the underside as this is where I made a poor decision to put the data and clock lines. In the future, these lines will be in a more robust cable and be located somewhere where there is minimal contact as these lines are extremely sensitive to the slightest disruption of electrical signals.

Media



The full working device soldered and fit onto a half perma-proto board



The underside of the device

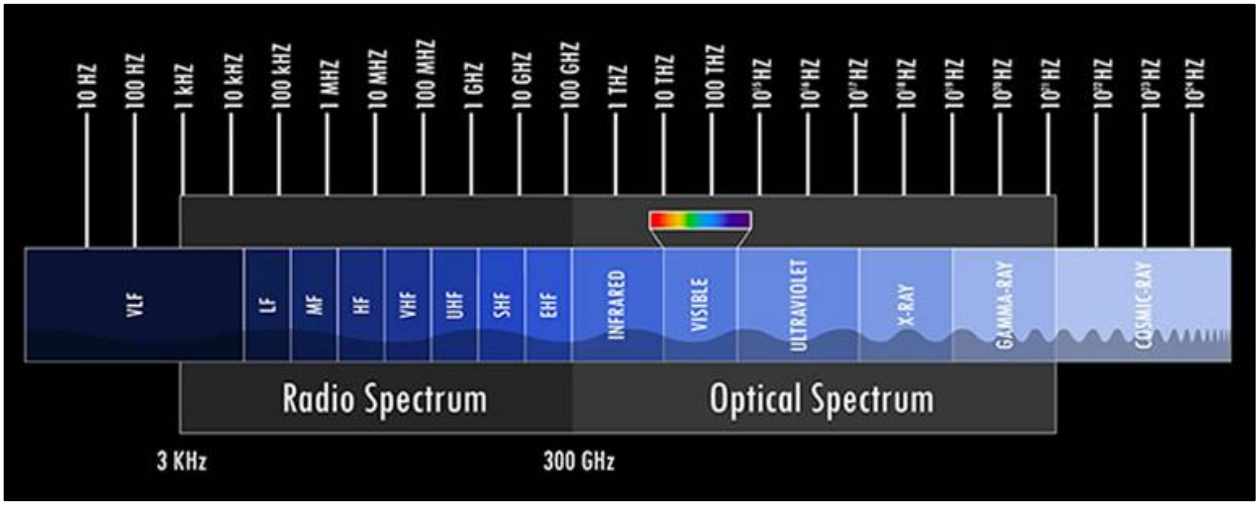
YouTube video link: https://youtu.be/SkQtCr-GAQM

Reflection

Transferring circuit onto a perma-proto board was better than I expected it to be; I was lucky to have some soldering experience from my Grade 10 ISP, the analog traffic light where I learned a lot of valuable lessons. When first introduced to the project, I knew I had to take my time with it in order to get it 100 percent working as I was not yet an expert at soldering but had lots of time over the Christmas break. When I finally got to that stage, I saw the benefit of previous soldering projects paying off as it felt a lot easier than last year. I knew how to avoid problems that I ran into and how to fix them as well. Overall, it was a nice change to challenge my design, planning, and carefulness of familiar components instead of learning and experimenting with new ones.

After deep consideration of my time, other courses, and energy, I have decided to try my best at putting the perma-protoboard in a metal mint tin. I was truly on the fence about whether or not to push this project to the finish line as the workload from other classes has ramped up; there were major culminating projects approaching their deadline as well. From the very start of this project, I kept pondering the tradeoffs of boths sides, but in the end, after the completion of this DER writeup and a hot shower, I realized I had come this far and my hesitation turned into certainty. I decided to keep on going as I felt it was worth a final effort. I realized over the years of many DER writeups, videos, and painstaking projects, that there are many lessons and skills to be learned when taking the long, hard, and unfamiliar road to the finish line, even if you do not completely cross it.

Project 2.5: Wireless Communication (Infrared)



Purpose

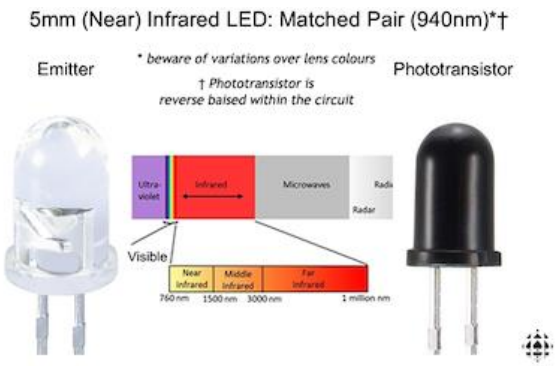
The purpose of this project is to explore and work with the common forms of wireless communctiation. This specific project explored the use of Infrared control.

References

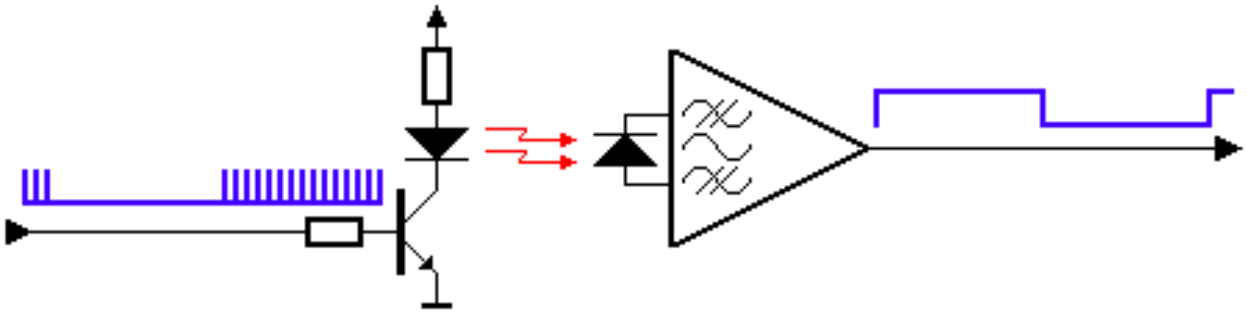
http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html#Wireless
https://learn.sparkfun.com/tutorials/ir-communication/all
https://www.sbprojects.net/knowledge/ir/index.php

Procedure

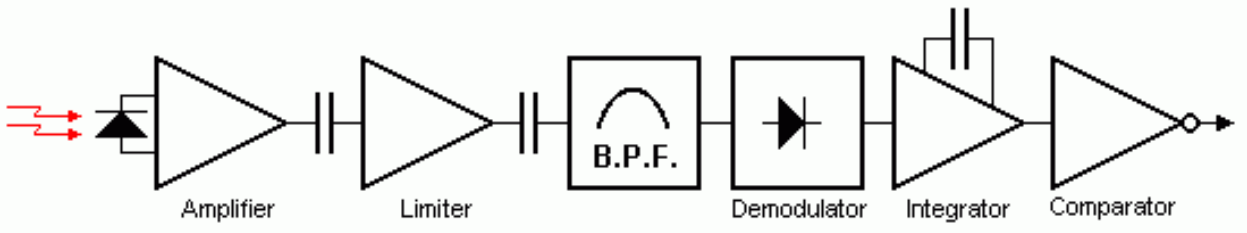
The basis of the project utilizes the Infrared wavelength to transmit and receive data. Infrared, or IR for short, is a wavelength with a lower frequency than visible light, hence, humans cannot perceive it. Cameras can visually detect IR in near proximity as shown to the right in the form of purple light since the silicon based sensor absorbs that wavelength. Other animals such as snakes, mosquitos, and fish also dectect infrared. They evolved to see IR to locate prey and navigate in the dark as IR is radiated through sources of heat. Therefore, for a receiver diode to properly receive an infrared signal, it must be modulated or flashed at a certain frequency so that the receiver can distinguish between ambient IR and the actual transmitter LED. Usually, these frequencies are set at 38 kHz however depending on the remote and protocol, the frequency may differ.



As shown on the right, the transmitter is comprised of a transparent infrared emitter LED while the receiver diode is black or dark blue to absorb the most light. Current running through the transmitter LED controls how far the signal travels; the more current runs through, the farther the signal can be detected, but as more current flows through, the higher the heat production in the LED and energy used in the battery. These side effects can be reduced by turning down the pulse/pause ratio. The rest of the circuit usually include buttons, a microcontroller and a quartz crystar or ceramic resonator to modulate the signal. Usually, the microcontrollers feature a low power sleep mode where it "wakes up" and transmits a command once a button is pressed to save battery life.



The command is sent and received through the receiver LED and its circuit. Displayed above, the picture substitudes a microcontroller with a simple transistor circuit to relay the IR pulses. The wavelength is picked up by the receiver diode and travels through the components in the block diagram shown at the bottom. The amplifier amplifies the signal while the limiter acts as an automatic gain control (AGC) circuit to output a constant pulse strength no matter its input strength. The signal is sent to the band pass filter, tuned to listen and accept a certain modulation frequency while rejecting others. That is why TV's use specific remotes so that other remotes do not interfere. Next, the demodulator converts the modulated wavelength into a regular high low square wave while the last two components further configure the signal to produce a low when a carrier frequency is detected. The microcontroller then reads the incoming data as a serial bit stream.



At first, IR control was attemped to the two dimensional 8 × 8 LED matrix using Hugo Reed's MatrixMadeEZ PCB. But, after three days of failure from the IR codes not reading properly and my limited timeframe, I decided to step down to the simpler one dimensional Morland Bargraph PCB. Here, a remote can control the bits state, shift the bits left or right, increase or decrease the displayed binary value by one, create a scrolling animation with play/pause functionality, reset the value to 0, and use the on/off button to turn the display on and off.

The remote used for this project features a 3 × 7 button arrangement containing icons of standard remote functions as shown. This remote sends IR frequencies to a three pin receiver sharp GP1UX511QS IR sensor which accepts a range of frequencies and sends its output through one pin while the other two are used for five volts of power and ground. The Morland Bargraph PCB consists of a 74HC595N shift register controlling a ten LED bargraph. Because of the dilemma of a maximum of eight outputs on the shift register, only eight LEDs of the bargraph can be controlled. Current is then fed into a nine pin bussed resistor network to avoid blowing the LEDs. Bussed resistor networks contain a number of inputs containing resistors for a number of LEDs or other devices that need limited current and routes them to one output pin usually leading to ground. Therefore, the bussed resistor network containing nine pins features only eight resistors.

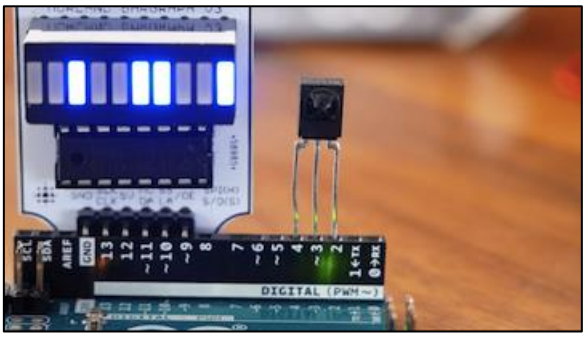
Parts Table	
Quantity	Description
1	Arduino UNO
	USB A to C cable
1	IR remote
1	Sharp GP1UX511QS sensor
1	Morland Bargraph PCB
1	1 × 6 female header
1	74HC595N shift register
1	10 LED bargraph
1	9 pin 330Ω bussed resistor

Instead of using the software shift out function to manipulate the shift register, the more efficient and faster hardware SPI can be used in exchange for limited flexibility as the shift registers serial input pin, clock and latch pin must be connected to the ATmega328P's hardcoded master out slave in (MOSI), clock, and slave select pins respectfully. On the Arduino UNO the SPI peripherals are located in digital pins 13 for the clock, 11 for MISO, and 9 for slave select. Digital pin 12 contains the master in slave out (MISO), however, since the bargraph does not return any data to the microcontroller, it is not used. Through meticulous planning and design of the PCB from T. Morland, this device can be used as a breadboard appliance on the UNO while using SPI. Therefore, no breadboard is needed for this project as both the IR receiver and PCB can fit on the UNO.

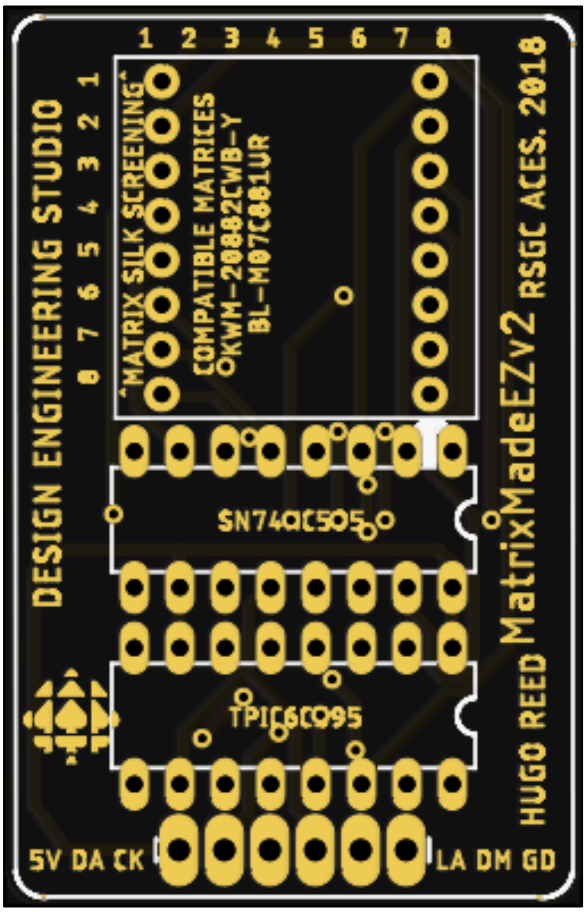
The code in the Arduino IDE to form specific codes from reading the incoming data stream of the IR sensor uses Ken Sherrifs library to decode the signal. It contains a plethora of protocols and frequencies from different remotes used to identify and send signals that emulate the original IR remote signal. Once the code is detected, a series of if statements compare predetermined codes that correspond to the buttons on the remotes to see if it was pressed. If so, an 8-bit value is altered in some way using bitmath and the value is then relayed to the shift register and displayed in binary. More information is available in the code section.

Media

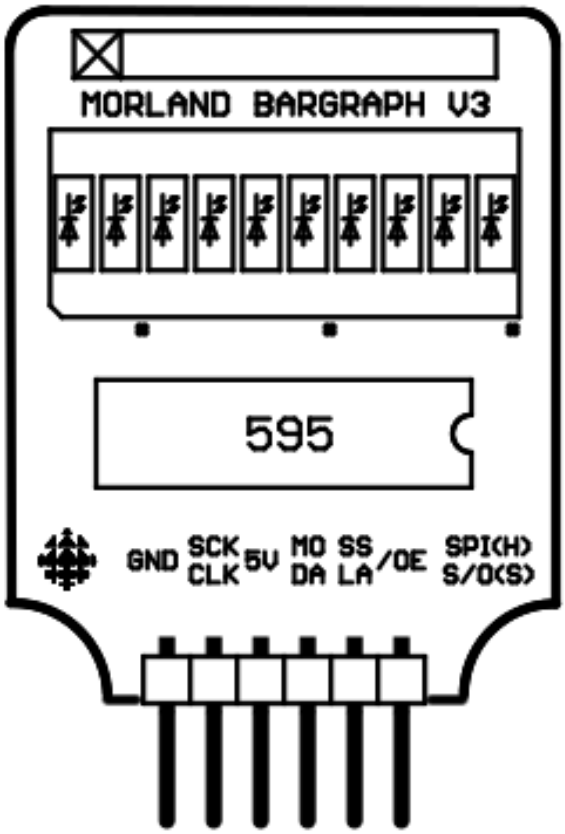
The project setup controlled through IR by a remote



The Morland Bargraph configured in SPI mode on the Arduino UNO



The MatrixMadeEZ PCB designed by Hugo Reed



The Morland Bargraph PCB designed by Tim Morland

YouTube video link: https://youtu.be/41g1aDzq_po

Code

```
// PROJECT  :SPIShiftOut
// PURPOSE  :To highlight the overlap between the speed of hardware SPI and the flexibility
//          :of software shiftOut by transmitting a byte to a 595 Shift Register
// DEVICE   :Arduino UNO + Morland Bargraph V3 (insert appliance as shown)
// AUTHOR   :C. D'Arcy (SPI Morland Bargraph) + Xander Chin (IR control)
// DATE     :Feburary 27, 2021
// uC       :328p
// COURSE   :ICS3U/ICS4U
// STATUS   :Working
// REFERENCE:http://darcy.rsgc.on.ca/ACES/TEI3M/images/SPIConcept.png
//          :http://darcy.rsgc.on.ca/ACES/TEI3M/SPICommunication/images/SPIvsShiftOut.png
// NOTES    :Hardware SPI (fast) and software shiftOut (flexible) are not identical but
//          :strikingly similar in their digital waveform behaviour.
//          :This code highlights and contrasts the differences between their usage
//          :on familiar hardware. Explore which one serves each of your applications better.

#include <SPI.h>              //required SPI library
#define OENABLE 9             //MBV3 pin for Output Enable

#include <IRremote.h>
#define receivePin 4
#define GND 3
#define VCC 2
IRrecv irrecv(receivePin);
decode_results results;

uint32_t code = 0;         //code value
uint32_t prevCode = 0;     //for repeat codes
uint32_t numberCodes[] =
{16738455, 16724175, 16718055, 16743045,
 16716015, 16726215, 16734885, 16728765};
//--bit0--|--bit1--|---bit2---|--bit3---|
//--bit4--|--bit5--|---bit6---|--bit7---|

uint8_t value = 0;            //value shifted out to the shift register
bool play = false;            //scroll or static
bool left = false;            //scroll left or right
bool state = false;           //controls state of output enable pin
uint32_t lastShift;           //

bool showValue = true;        //print value in the serial monitor
#define scrollSpeed = 100;    //how fast the bits scroll in milliseconds

void setup() {
  Serial.begin(9600);
  pinMode(OENABLE, OUTPUT);   //595's output enable pin must be pulled
  pinMode(GND, OUTPUT);
  pinMode(VCC, OUTPUT);
  digitalWrite(GND, LOW);
  digitalWrite(VCC, HIGH);
  digitalWrite(OENABLE, state); //LOW to display its storage registers
  //pinMode(MISO, OUTPUT);      //No need for MISO(12) in this example

  Serial.println("Enabling IRin");
  irrecv.enableIRIn(); // Start the receiver
  Serial.println("Enabled IRin");
}
```

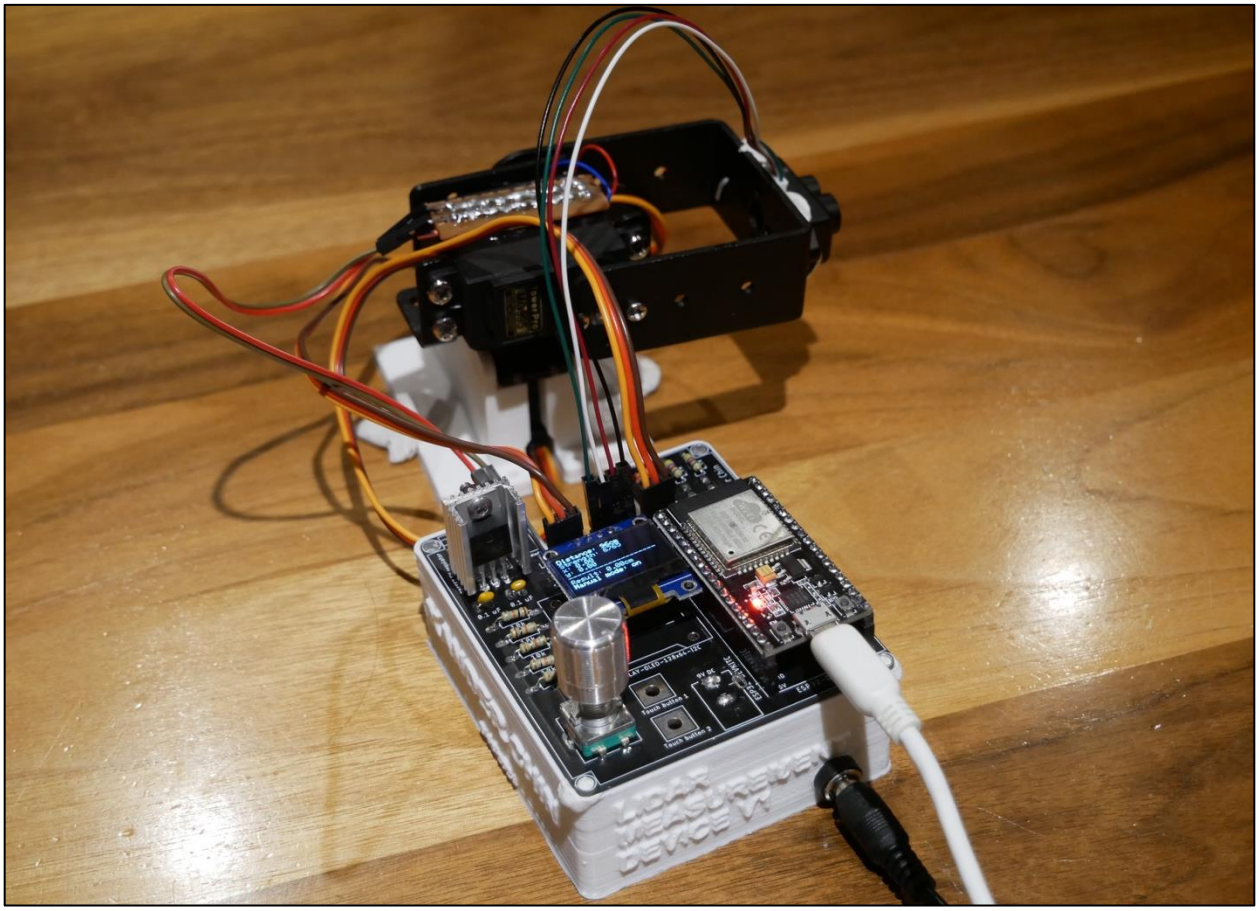
```
void hardwareShiftOut(uint8_t value) {
  //Initializes the SPI bus setting SCK, MOSI, and SS to outputs,
  SPI.begin();                //pulls SCK and MOSI low, and SS high. Default: MSBFIRST
  //digitalWrite(SS, LOW);    //true SPI requires Slave Select LOW to identify specfic target
  SPI.transfer(value);        //invert the output for ease of interpretation
  digitalWrite(SS, LOW);      //pull Slave Select LOW to identify target device
  digitalWrite(SS, HIGH);     //release target device
  SPI.end();                  //disables SPI Bus (leaving pin modes unchanged)
  digitalWrite(OENABLE, state);
}

void loop() {
  checkIR();

  //cycle through number codes
  for(uint8_t x = 0; x < 8; x++) {
    if(code == numberCodes[x]) {
      if((value >> x) % 2 == 1) value = value - (1 << x); //if bit is on turn off bit
        else value = value + (1 << x);                    //else turn off bit
    }
  }

  //codes for fast forward/backward arrow keys
  if(code == 16720605 || code == 16761405) {
    if(code == 16720605)
      left = true;                //shift left
    else
      left = false;               //shift right

    if(!play) keepInBits(left);   //if no scrolling shift bits
  }

  //more codes
  if(code == 16769565) value = 0;         //reset button
  if(code == 16712445) play = !play;      //play/pause button
  if(code == 16753245) state = !state;    //power button
  if(code == 16748655) value++;           //up button
  if(code == 16769055) value--;           //down button

  //scrolling animation
  if(millis() - lastShift > scrollSpeed && play) {
    keepInBits(left);       //scroll bits
    lastShift = millis();   //reset timer
  }

  //helps repeat codes
  if(code != 4294967295 && code != 0) prevCode = code;

  //repeat codes
  if(code == 4294967295) {
    if(prevCode == 16748655) value++;   //up button
    if(prevCode == 16769055) value--;   //down button
  }

  hardwareShiftOut(value);                //display value on bargraph
  code = 0;                               //reset code
  if(showValue) Serial.println(value);    //print value in decimal
}

void checkIR() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value);
    code = results.value;
    irrecv.resume(); // Receive the next value
  }
}
```

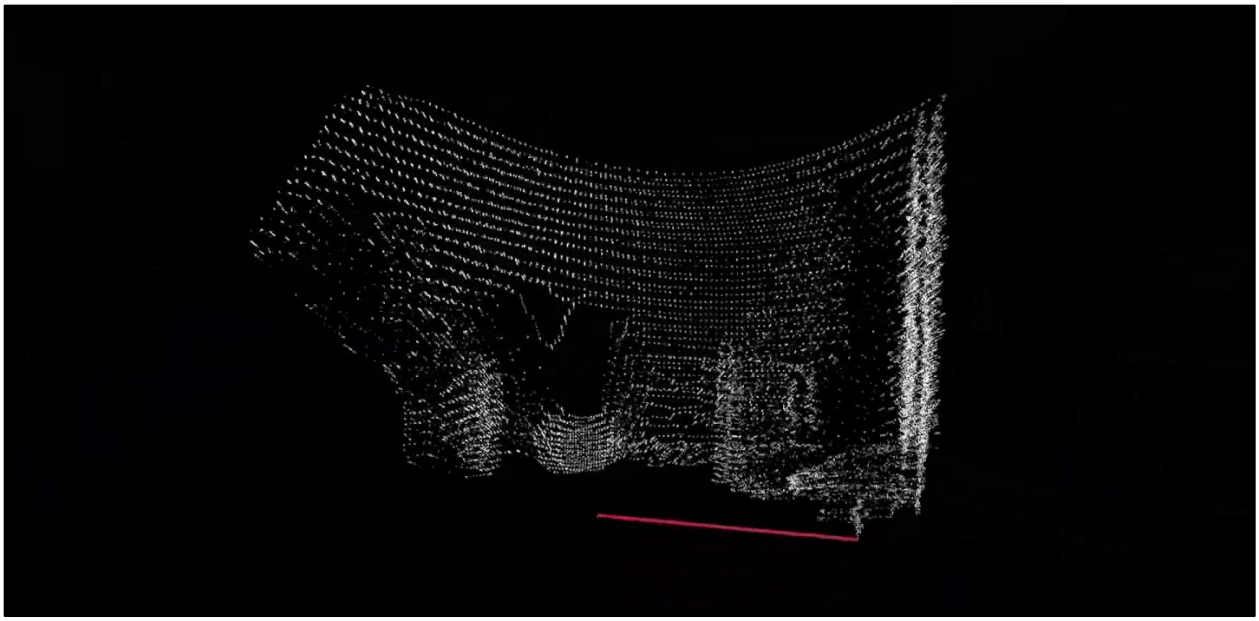
```
void keepInBits(bool left) {
  if(left) {
    //if bit 7 is not on:
    if((value >> 7) % 2 == 0) value <<= 1;     //shift up by 1
    else {                                     //
      value <<= 1;                             //shift up by 1
      value += 1;                              //turn on bit 0
    }
  }

  else {
    //if bit 0 is not on:
    if(value % 2 == 0) value >>= 1;            //shift down by 1
    else {                                     //
      value >>= 1;                             //shift down by 1
      value += 128;                            //turn on bit 7
    }
  }
}
```

Reflection

This optional project was fun but frustrating to do, especially when trying to figure out and failing to control the MatrixMadeEZ with IR. I realized that a lot of time was spent trying to fix it, where it could have been used for more important things such as focusing on my ISP. However, it was from this experience that I learned a lot about the matrix, interrupts, infrared control and code improvements. When I finally decided to step down to the bargraph, I did not have a lot of time to complete the project, but with my previous experience of working with the MatrixMadeEZ, configuring the Morland Bargraph was a breeze. Though, I do believe that my code could have been refined more as I did use a lot of if statements, but with limited time, this was the best I could do. Along the way of completing the project, I experimented with the blocking the IR signal in various ways and was very fascinated and surprised at the sensitivity and range. Overall, IR was fun to play around with and because of my acquired knowledge and usage of this specific form of wireless communication, I am currently planning to use it in my future projects.

Project 2.6: (ISP - Medium): The LiDAR Measurement Device





The LiDAR Measurement Device Parts Table	
Quantity	Description
1	ESP32
2	MG995 servo motors
1	TFmini S LiDAR sensor
1	Laser pointer
1	MG995 servo motor 3D printed case
1	LiDAR Measurement Device 3D printed case
1	LiDAR Measurement Device PCB
2	0.1 µF capacitor
5	10 kΩ fixed resistor
2	4.7 kΩ fixed resistor
2	3.3 kΩ fixed resistor
1	EC-11 rotary encoder
1	128 × 64 OLED display
1	7805CT 5V regulator
1	7805CT 5V regulator heatsink
1	USB C to microUSB
1	5V 2A DC power adapter
8	Rectifier diode
*****	Male headers
*****	Female headers

Purpose

The purpose of this project was to explore our interests in engineering by creating a project that encompasses the three branches: Hardware, software, and design. For my project I decided to create a device that can measure the shortest distance between any two objects.

Theory

To achive this result, trigonometry is needed as without it, the only way to measure distance between two objects would be to direcly measure with a ruler. Ideally, the setup forms some sort of triangle with some known angles and side lengths. The two objects can act as two verticies and of the triangle with one other point as the last vertex. This last point is where the device is can be positioned, forming a triangle with the distance between the two objects opposite where the device sits. This setup is perfect for implementing the Law of Cosines where distance between the objects and lidar sensors as well as the angle between the two objects relative to the device is known. The angle between the two objects are gathered from servo motors, which are used to set their position and attached is a a distance sensor is used to get the lengths from it to both objects. This fulfills the requirements for the Law of Cosines, therefore, distance can be measured from any two objects so long as the servo motors have the mobility to do so.

$$c = \sqrt{a^2 + b^2 - 2ab\cos\Theta}$$

Here, the project will be divided into four parts: Hardware, software, mathematics, and design.

Part A: Hardware

Purpose

The hardware section discusses the detailed workings of components implemented into the project as well as how they interact together.

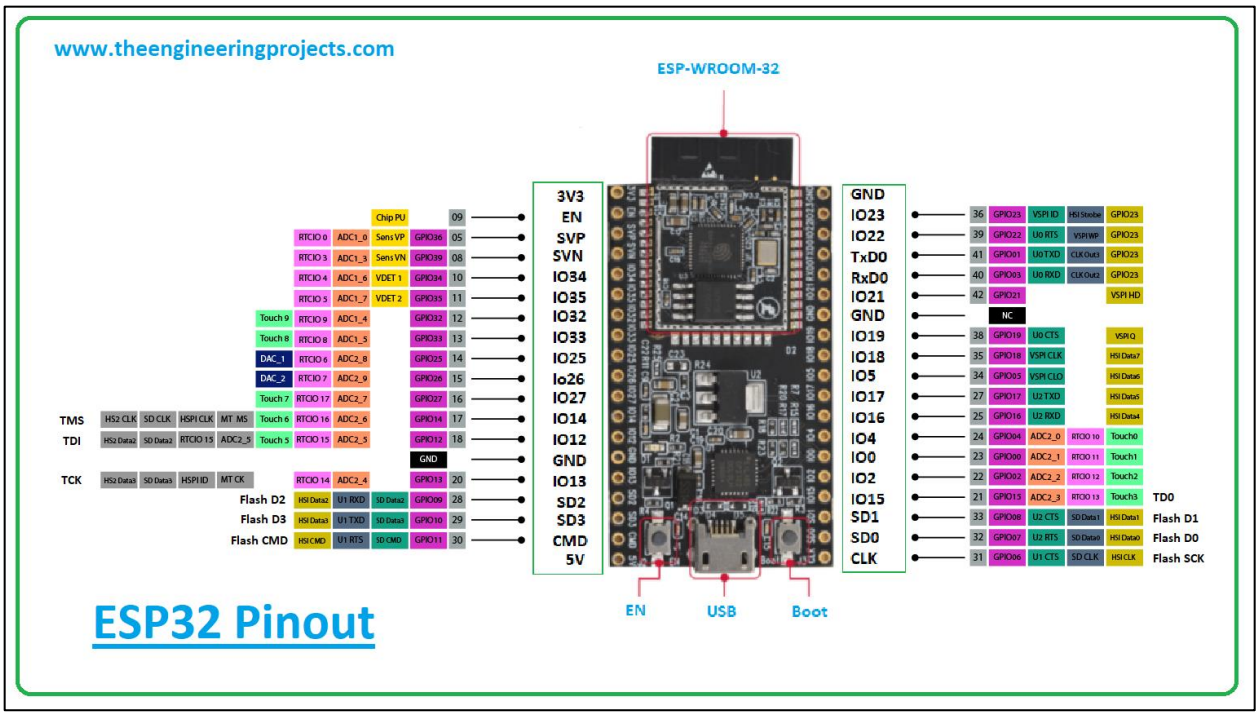
References

http://www.technoblogy.com/show?1YHJ
https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf
https://circuitdigest.com/microcontroller-projects/using-classic-bluetooth-in-esp32-and-toogle-an-led
https://www.youtube.com/watch?v=5bHPKU4ybHY&t=455s
https://www.makeblock.com/project/mg995-standard-servo
https://en.wikipedia.org/wiki/Lidar
https://www.gotronic.fr/pj2-sj-pm-tfmini-s-a00-product-mannual-en-2155.pdf

Procedure

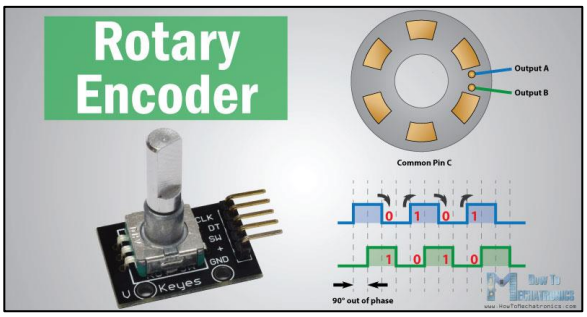
The basic rundown of the hardware consists of a distance sensor mounted on two servo motors, one controlling the yaw while the other controls the pitch. These servos are usually controlled through Bluetooth, but if not connected, the device will revert to manual control. This consists of a rotary encoder paired with touch buttons to manually control the servos while an OLED display presents the current information of the device such as the current distance, signal strength, pitch and yaw angles with I2C communication. A laser pointer is also mounted on the top for better visualization of where the sensor is pointing at. Pushing the button on the rotary encoder saves a current point and by moving the servos around, the OLED screen displays the distance between the saved point and the current point the sensor is looking at. All these components are driven by a microcontroller powered through a microUSB cable.

This microcontroller is a new Arduino compatable board called the ESP32, similar to the familiar ATmega328P but with more peripherals and faster processing. It comes from a line of other ESP boards. This specific ESP32-WROOM board offeres lots of extra features including in-built Bluetooth, Wi-Fi, a hall effect sensor and capacitative touch pins just to name a few. The processor supports a clock frequency of up to 240 MHz with an onboard SPI flash size of 4 MB and 34 programmable I/O pins. These pins have a myrad of functions with 8-bit digital to analog converters (DAC) and 12-bit analog to digital converters (ADC), along side SPI, I2C, UART, PWM and I2S pin support. The whole list of features for each pin can be found in the datasheet provided in the references and in the pinout below.



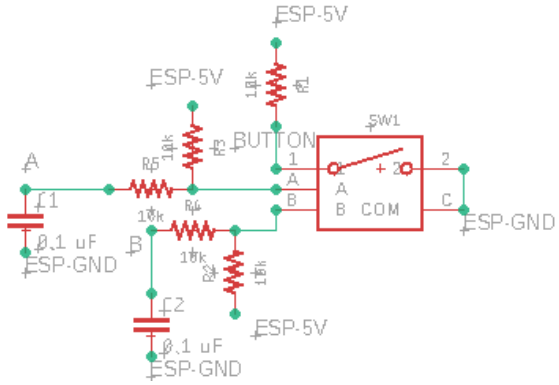
The main feature implemented in the project was Bluetooth for wireless control and data transmission. Bluetooth on the ESP32 offers many implementations such as Bluetooth radio and Bluetooth interfaces however, the Bluetooth needed for this project is quite simple. It uses Classic Bluetooth, which is the ones everyday smartphones and computers use, to send and receive data in the form of the Serial Port Protocol (SPP). The ESP32 opens up a serial port where another device using the standard protocol can easily connect by going to the Bluetooth connection terminal. This makes connecting, sending and receiving very straightforward like any device.

Other features such as the capacitative touch pins were implemented to control the device. These touch pins read a value and when anything that can hold an electrical charge is detected such as a finger, the value drops. This effect will work when the pin is connected other conductive material. Compared to a standard button, these touch sensing pins are more effective as there is no bounce, there are is need for any external components and they are directly implemented into the board for easy control such as integrating them with interrupts. Here, two touch buttons are setup to control the pitch of the servos.

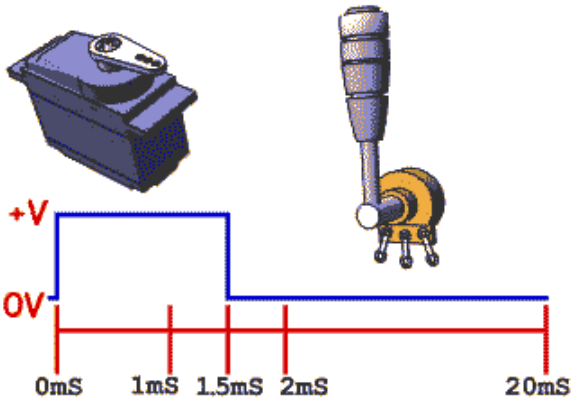


Along side these buttons is the EC-11 rotary encoder. By turning it one way or the other, the microcontroller reads it and can detect the rotation direction and the amount. To do this, the rotary encoder has two juxtaposed output pins labeled A and B. Inside the encoder are metal sheets that when turned, emit an electrical signal to the pins. With the way they are setup, one rotation will enable a high pin A before pin B while the other rotation does the opposite, therefore, any microcontroller can read these two pins and detect which direction it turns and how many times it does by tallying each pins change in state. This is especially useful for fine control over electrical components.

Rotary encoders are still susceptible to bounce so a low pass filter or other hardware/software implementation is needed to reduce this. Here, resistors and capacitors are used to reduce noise using this circuit shown. The EC-11 also includes a built in button which is used for saving distance points. By turning the encoder once, the servo yaw moves by one degree either left or right depending on the direction of rotation.



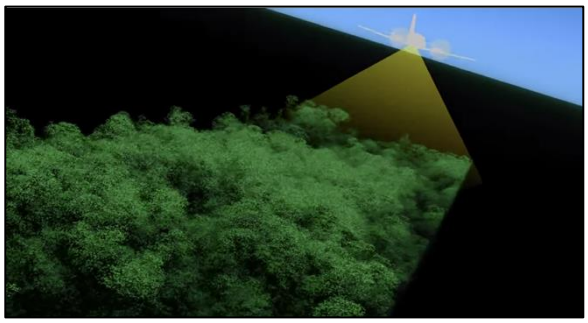
With the touch buttons and rotary encoder, the signals are read by the ESP32 and sent out to the servo motors. Servo Motors have a certain range of motion usually from 0° to 180° controlled by a pulse width modulation (PWM) signal through one pin. Differnet PWM signals correlate to different positions of the servo. Attaching any PWM pin to one of the I/O pins of the ESP32 will properly control the motor as all pins support PWM. Servo motors tend to require a lot of current especially when under a load so they are usually powered through a separate power supply since most microcontrollers output a maximum of 500 mA.



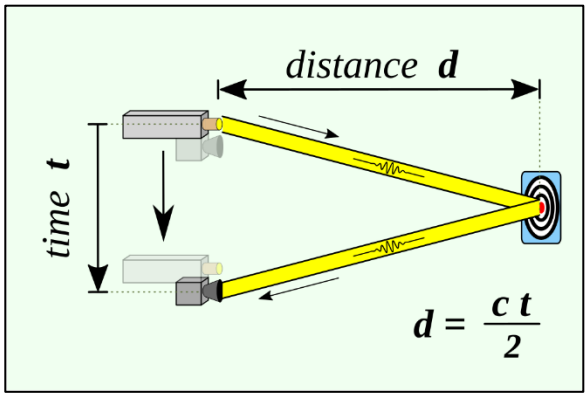
In this project, common MG995s servo motors are used with a range of 180°, an operating voltage of 5 volts and a current draw of 350 mA each without any load. When stalling, the current needed peaks to around 1500 mA. Hence, the minimum current draw is 700 mA and the maximum is 3 amps, therefore, a separate DC power supply is needed. Originally 9 volts DC was provided and scaled down to 5 volts using a linear regulator and a heat sink to dissipate heat better, but this proved too dangerous as the regulator rapidly became too hot to touch because of the amount of wattage being dissipated through heat. Wattage (W) can be calculated using this formula where V is voltage and A is amps:

$$W = V \bullet A$$

When plugging in the values of the wasted voltage and amount of current flow, the amount of watts turns out to be $4V \bullet 0.7A = 2.8W$ minimum and $4V \bullet 3A = 12W$ maximum! This amount of watts emanated through heat was most likely enough to overcome the regulators maximum operating temperature so 9 volts was replaced with a 5 volt 2 amp power supply.



The LiDAR sensor is the last and arguably the most important piece of hardware equipment that measures the distance between the object and itself. LiDAR, LIDAR, or lidar is an acryonym for light detection and ranging or laser imaging, detection, and ranging. As the name suggests, this type of technology is used in 3D scanning, imaging and rangefinding. One prime example is self-driving cars where a spinning LiDAR sensor mounted ontop scans its surroundings. Another is equipping it onto a flying vehicle to scan cities or forests.



In essence, LiDAR sensors works by emitting ultraviolet, visible, or near infared rays where it bounces off an object and is received by the sensor. In spite of this, the more reflective the object is the higher the accuracy will be. The time it takes for the beam to bounce back is recorded and is implemented as t in the speed equation where c is the speed of light. This is called the time of flight principle

The LiDAR sensor used in this project is the TFmini-S which is one of the more affordable ones to buy. The sensor uses the ToF principle by sending out a near-infrared beam and communicates with a microcontroller using UART at a standard baud rate of 115200 or I2C. It operates at 5 volts but uses 3.3 volt logic levels, therefore, a logic converter or simple voltage divider is needed for the communication pins. For simplicity sake, a voltage divider is used with a 3.3 kΩ resistor connected to 5 volts and another resistor of 4.7 kΩ connected to ground, stepping the voltage down by around 3/5.

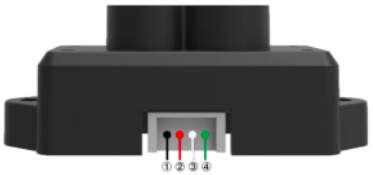


Table5: The Function and Connection Description of each pin

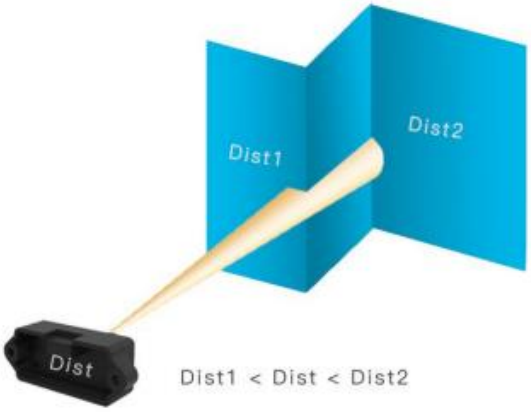
No.	Color	Function	Comment
①	Black	GND	Ground
②	Red	+5V	Power supply
③	White	RXD/SDA	Receiving/Data
④	Green	TXD/SCL	Transmitting/Clock

Since this LiDAR sensor is one of the cheapest, it comes with certain limitations but is still very suitable for a prototype. Characterization of the sensor is shown on the table taken from the data sheet. Here, the range and accuracy can vary depending on reflectivity. The data sheet states that a black object with 10% reflectivity will only accurately measure from 0.1 to 5 meters while a white object with 90% reflectivity will accureately measure from 0.1 to 12 meters with a standard deviation from 0.5cm to 2.5cm respectively. Also, the field of view of the beam will play a role in effective detection of an object. The farther the object is, the wider it has to be for the increasing width of beam to effectly make contact. This minimum diameter of the object can be calculated using this formula where D is the distance between and β is the FOV but halved:

Description	Parameter value
Operating range	0.1m~12m①
Accuracy	±6cm@ (0.1-6m) ②
	±1%@ (6m-12m)
Measurement unit	cm
Range resolution	1cm
FOV	2°③
Frame rate	1~1000Hz (adjustable) ④

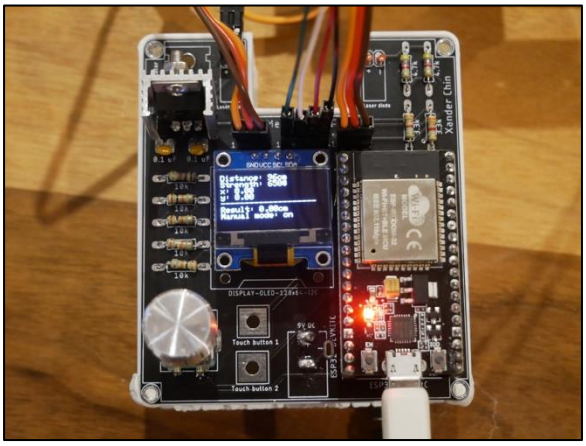
$$d = 2 \bullet D \bullet \tan\beta$$

Therefore, using this equation for an object 1 meter away gives a minimum diameter of 3.5cm while detecting an object 12 meters away unfortunately gives a rather large minimum diameter of 42cm. This effect also creates another downside where the distance will be between the actual distance of two object if the light spot hits both as shown on the right. This is especially prevalent the farther the sensor detects as the diameter of the light spot gets larger according to the formula.

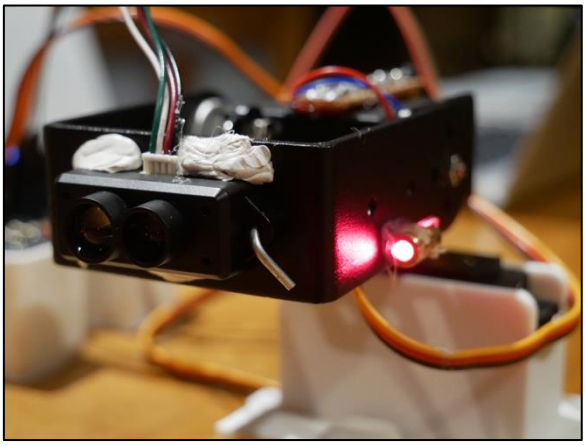


With all these components, the hardware aspect is complete. Software takes over from here by giving instructions to the hardware from the ESP32 and other applications.

Media



The LiDAR Measurement Device reverting to manual control



The laser pointer that aids with the direction at which the sensor is pointing at

Part B: Software

Purpose

The software examines the programs and sketches designed to work with the hardware components and to implement engaging visualization and user control.

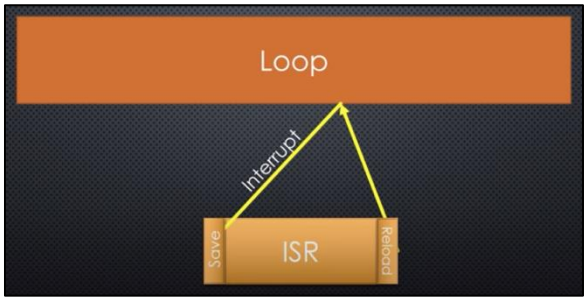
References

https://learn.sparkfun.com/tutorials/processor-interrupts-with-arduino/all
https://www.youtube.com/watch?v=CJhWlfkf-5M
https://processing.org/

Procedure

The project uses two coding platforms. One is the familiar Arudino IDE where a single sketch coded in the C++ language is loaded onto the ESP32 while the other is Processing which uses the Java language to implement graphic display and user interactivity on an every day computer. These two platforms are connected by Bluetooth where code from the ESP32 wirelessly sends over data to the two processing sketches depending on which one is open.

On the ESP32, the sketch checks to see if Bluetooth is connected through an in-built callback function in the Bluetooth library. If Bluetooth is not connected the ESP reverts to reading the encoder and touchbuttons and displaying information to the OLED display.



The sketch makes use of the many interrupts on the ESP32, specifically touch interrupts. Usually, a microcontroller can only run one piece of code at a time. In the event where a change is made when it is not running the if statement that checks that code, that change will be missed. Interrupts, however, detect some sort of change no matter what the microcontroller is doing at the moment. This triggers a section of code called the interrupt service routine (ISR) to run, allowing the microcontroller to "multitask" somewhat. Interrupts do come with some limitations such as no delays and print statements as well as simple as possible ISRs in order for them to function efficiently and properly. Hardware interrupts respond to a change in a pin while software intterups respond to a change in software instruction. Here, the touch interrupt is a type of hardware interrupt that senses when a finger is touching the pin and reacts accordingly, allowing the OLED screen to display information while at the same time detecting touch contact. More information can be found in the code posted in the code section.

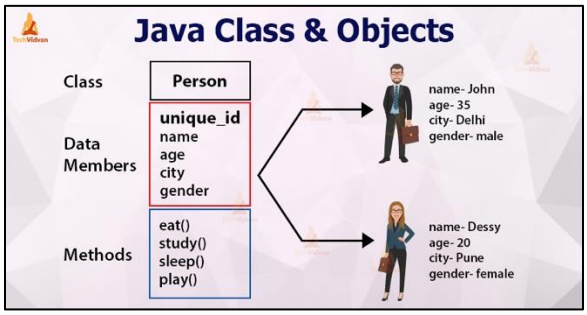
The rotary encoder at first was also read using hardware interrupts for the two pins. Hardware interrupts can be configured in many ways such as changing it to detect rising edges, falling edges, changing edges, and low and high signals. In this situation, these hardware interrupts were configured to trigger an ISR on falling edges of the two pins, which allowed accurate rotations and direction of the encoder. This proved to be ineffective as it took a while to figure out that the ESP32 for whatever reason dectects both falling and rising edges when configured to falling. As of now, the encoder is read through simple polling of the loop function as the problem was too time-consuming to solve which unfortunately leads to some missed readings. The OLED display also needs to be paused in order for the ESP to properly read the encoder as the display function of the Adafruit OLED library takes a long amount of time to complete.

In tandem with the touch interrupts are the timer interrupts that control how fast the servos pitch changes. Timer interrupts are a little different than traditional interrupts; they run their ISR when a certain amount of time has passed. Together, it increses or decreases the servo pitch by one degree depending on which touch button is pressed through the mentioned touch interrupts. All code with relevant comments can be found in the code section for a better understanding.

The other code includes the two sketches formulated in Processing for user control and visualization of the measruements. In short, Processing is a software program relvolving around the manipulation of visual arts and graphics using a computer which makes it the perfect application to use for professional looking viusals and controls for electronic devices. As mentioned before the program uses Java as opposed to the Arduino IDE's C++. Comparing the languages at a fundamental level used in the sketches, there is not much notable differences. But, when adding in the extra classes and methods that make up the Arduino IDE for electronic control and Processing for visuals, both present a challenge in learning and getting used to.

The basic setup of a processing sketch includes a setup function that runs once and a draw function that runs for an indefinite amount of time just like the Arduino IDE. To acutally communicate with a microcontroller, the sketch implements the serial library to detect, send and receive data through the Bluetooth serial port setup by the ESP32. Whenever serial data is sent over to processing, the sketch runs the code in the serial event function like an ISR. This goes for other built-in functions such as keyPressed or mousePressed which runs code when a certain key or mouse is pressed. Using these functions allows for suitable user control.

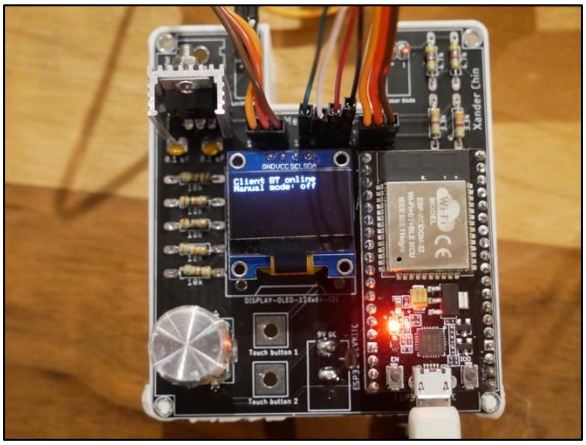
The first processing sketch demonstrates some basics of processing. A 2D canvas is setup for text and other shapes to be displayed upon. The servos are controlled through the WASD keys while receiving data from the device. Certain points can then be saved to the sketch and are then used to measure distance between selected points. The sketch updates and presents this information in text on the interface. The YouTube video in the media section of this project demonstrates this more clearly.



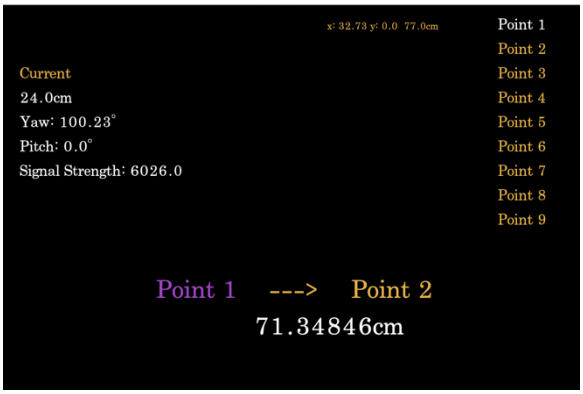
A notable code function used widely in this project is the custom class implementation. Classes are often used in Java, as it is an object-oriented language. They consist of custom made objects which have their own specified attributes and functions. Names for these fucntions are most commonly uppercase to distinguish it from other varibles. In this sketch, a custom class called Text is created for displaying the information gathered from the ESP32 in text on the processing window. This class allows for highlighting, selection, and a change of words for the text object so that different points can be selected and interchanged among one other.

A custom class is also used in the second processing sketch which controls the servos to systematically scan the room around the device, presenting a 3D visualization of it called a point cloud. Here, a 3D canvas with an x y and z axis is used for displaying points, lines and other shapes located at different positions in 3D for a real life representation of the room. With this, distance measurements between points can be made and displayed on the point cloud while a 2D heads up display (HUD) lay out the necessary information. It also offers a lot of customization as shown in the YouTube video demonstration and media section. Code for the ESP32 and the two processing sketches can be found in the section titled code.

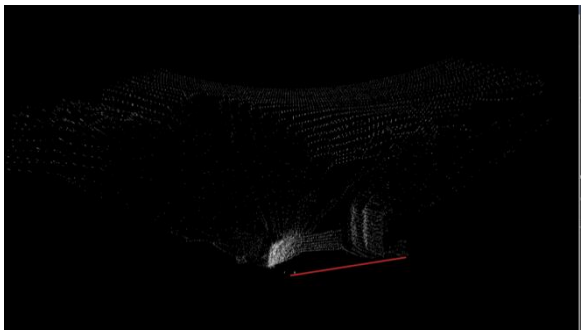
Media



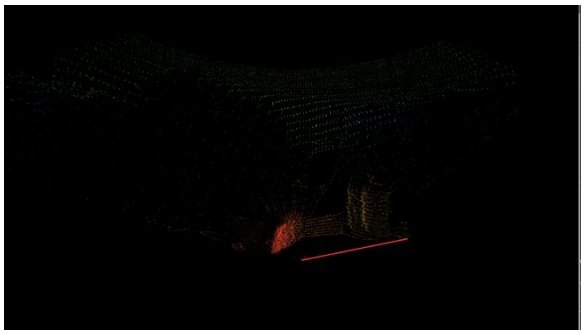
The LiDAR Measurement Device controlled over Bluetooth



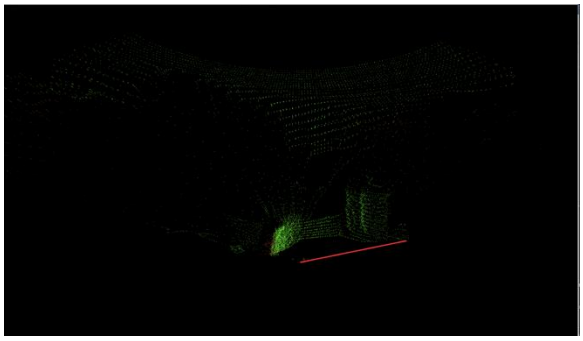
The first processing sketch showing off highlighting and selection features of text



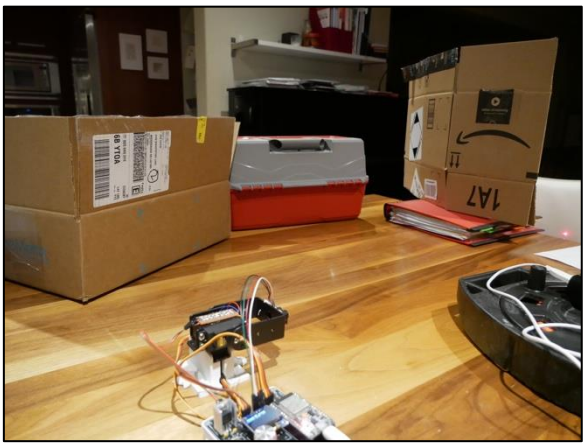
A point cloud created by a scan of the surroundings



A point cloud with a distance gradient created by a scan of the surroundings



A point cloud with a strength gradient created by a scan of the surroundings



The real life surroundings of the LiDAR Measurment Device when scanning

Part C: Mathematics

Purpose

The mathemiatics section thoroughly explains the mathematics and use of formulas behind calculating distance measurements between points and displaying them in a 3D environment.

References

https://mathinsight.org/definition/magnitude_vector
https://www.khanacademy.org/science/physics/magnetic-forces-and-magnetic-fields/electric-motors/v/the-dot-product
https://stackoverflow.com/questions/11819833/finding-3d-coordinates-from-point-with-known-xyz-angles-radius-and-origin/29602921
https://onlinemschool.com/math/library/vector/angl/

Procedure

As explained in the theory section, the LiDAR measurement device relies on the Law of Cosines to calculate distance from two points using the two distances from the sensor to each of the two point as well as the angle inbetween. When moving only yaw or pitch, the angle can be taken directly from the difference in their yaw or pitch positions. However, when yaw and pitch are both altered in some way from one point to another, the angle calculation must deal with angles in 3D space. To make this solving the angles simple, the LiDAR sensor is based around the origin where as different objects around the room are centered around the scanner. When calculating distance between two objects, vectors can be drawn between the origin and the objects.

In summary, vectors are values or objects that have a direction and magnitude which is important in a 3D world because direction gives information on where an object is from the origin and magnitude gives information on how far the object is in relation to the start of the vector. In this case, all vectors from the LiDAR scanner start at the origin.

Here, 3D vectors math equations are needed to help calculate the angle between two points in 3D space. The equation to calculate the angle between two vectors, using the aforementioned direction and magnitude, is written here:

$$\gamma = \cos^{-1}\left(\frac{\overrightarrow{d_1} \bullet \overrightarrow{d_2}}{||\overrightarrow{d_1}|| \times ||\overrightarrow{d_2}||}\right)$$

The top part of the fraction represents the dot product of the two vectors written as $\overrightarrow{d_1} \bullet \overrightarrow{d_2}$ and their magnitudes written as $||\overrightarrow{d_1}|| \bullet ||\overrightarrow{d_2}||$. Magnitude of the vectors is simply the length of the vectors and the equation to calculate this can be calculated using their ending coordinates in 3D space when starting at the origin:

$$\left|\left|\overrightarrow{d_1}\right|\right| = \sqrt{(x_1)^2 + (y_1)^2 + (z_1)^2}$$

$$\left|\left|\overrightarrow{d_2}\right|\right| = \sqrt{(x_2)^2 + (y_2)^2 + (z_2)^2}$$

This however is not needed as the magnitudes can be taken from the LiDAR scanner calculating the distance between itself and the object. Now, the magnitude will be refered to as just d. Dot products give a scalar value of how much one vector is travelling in the same direction to another. There are two equations to calculate this:

$$\overrightarrow{d_1} \bullet \overrightarrow{d_2} = d_1 \bullet d_2 \bullet \cos\gamma$$

$$\overrightarrow{d_1} \bullet \overrightarrow{d_2} = x_1 \times x_2 + y_1 \times y_2 + z_1 \times z_2$$

Since the first one requires the angle between the two vectors, the only option is to use 3D coordinates which are also used in the equation for magnitude. Calculating these 3D coordinates when the LiDAR scanner is based around the origin are shown here where d is the magnitude and yaw and pitch are the angles of the motors in the x and y axis:

$$x = \cos(yaw) \bullet \cos(pitch) \bullet d$$
$$y = \sin(yaw) \bullet \cos(pitch) \bullet d$$
$$z = \sin(pitch) \bullet d$$

This is essensially how applications map out objects in a 3D space and how the second processing sketch does exactly that. Now that the coordinates have been figured out, the angle can be calucuated using the very long formula seen in the beginning by replacing the dot products with 3D coordinates:

$$\gamma = \cos^{-1}\left(\frac{\cos(yaw_1) \times \cos(pitch_1) \times d_1 \times \cos(yaw_2) \times \cos(pitch_2) \times d_2 + \sin(yaw_1) \times \cos(pitch_1) \times d_1 + \sin(yaw_2) \times \cos(pitch_2) \times d_2 + \sin(pitch_1) \times d_1 \times \sin(pitch_2) \times d_2}{d_1 \times d_2}\right)$$

Here, all that is needed is magnitudes of the two vectors d, and the yaw and pitch of the servos that point to the two different objects which then correctly calculates the angle. Luckily, this equation can be simplified further by taking out the magnitudes of the vectors through simple cancelling. Here is the resulting equation:

$$\gamma = \cos^{-1}(\cos(yaw_1) \times \cos(pitch_1) \times \cos(yaw_2) \times \cos(pitch_2) + \sin(yaw_1) \times \cos(pitch_1) \times \sin(yaw_2) \times \cos(pitch_2) + \sin(pitch_1) \times \sin(pitch_2))$$

The equation is still quite long so further simplifying can still be done. Factoring out $\cos pitch_1$ and $\cos pitch_2$ converts the previous equation to this:

$$\cos\gamma = \cos(pitch_1) \times \cos(pitch_2) \times (\cos yaw_1 \times \cos yaw_2 + \sin yaw_1 \times \sin yaw_2) + \sin pitch_1 \times \sin pitch_2$$

And through testing, I figured out that $\cos yaw_1 \bullet \cos yaw_2 + \sin yaw_1 \bullet \sin yaw_2$ is equal to $\cos(yaw_1 - yaw_2)$ so at the current moment, the equation implemented in the software to figure out the angle between two vectors knowing their pitch and yaw angles is as follows (after some post-project research, I discovered this simplification is due to the angle difference identity):

$$\gamma = \cos^{-1}(\cos(pitch_1) \times \cos(pitch_2) \times \cos(yaw_1 - yaw_2) + \sin(pitch_1) \times \sin(pitch_2))$$

I believe this equation can be simplified even further, however, with the time frame given for this project, this is all I could simplify that monster equation at the beginning to. Even so, simplifying the equation does not change the program as they are the same equation. But, simplifying it does make the code neater and helps further understand the basis of trigonometry at a deeper level.

Part D: Design

Purpose

The design section goes over the design aspects of the circuit and housing cases that elevates the device's ease of use and appearance.

References

https://www.thingiverse.com/thing:4483512

Procedure

Once prototyping of the project was complete on the breadboard, a PCB to house the hardware components as well as two 3D-printed cases were designed on EAGLE and Fusion360 respectively. The PCB was designed with all components mounted on top with the touch buttons designed as pads leading back to the designated touch pin of the ESP32. A notable design feature is the cutout at the front which was designated for the servo motors to fit in between. Right under this cutout are the The servo and LiDAR wires that connect to the PCB to provide more mobility of the servos and sensor. Also the copper traces of the board were increased from to 28 milli-inches (around 0.7 millimeters) to allow large current flow of the servo motors. In preparation of a case, M3 screw cutouts were made at the corners for mounting.
After sending off the file to JLCPCBs, a case for the PCB and motors were designed in Fusion360, the brother of EAGLE. By importing the board file into Fusion, the outline of the PCB served as the shape. Through some extrusions and shellings, a suitable case was made for the PCB with M3 screw holes to mount the PCB onto the case. To touch it up, my name, the project name and the angle equation and law of cosines were printed on the side of the case. It should be noted that after thorough testing, the the angle equation printed on the side of this case is not correct and differs from the one implemented in the Arduino and processing sketches. The equation is as printed:

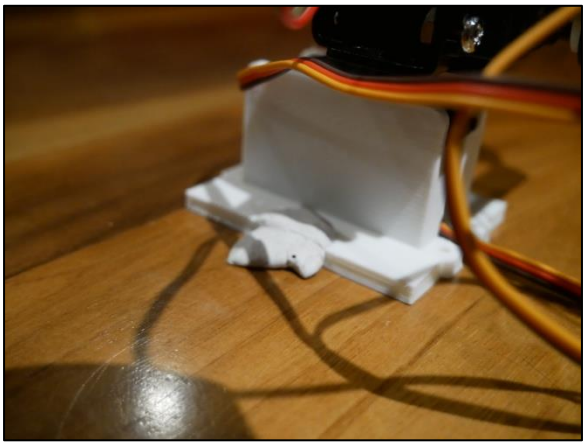
$$\gamma = \cos^{-1}(\cos(yaw_1 - yaw_2) \times \cos(pitch_1 - pitch_2))$$

This only works when the pitch angle begins at 0. If the pitch angle began higher, the more it reaches 90°, the less accurate the angle will be. I therefore had to figure out another way of how to calculate it.

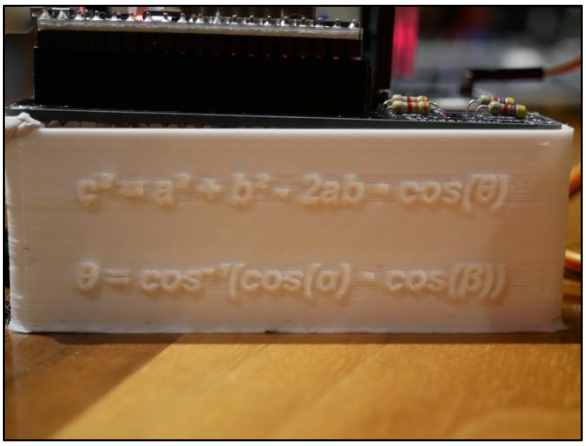
The case for the motors was based around a premade case. Since the premade one was designed for only one motor, a rectangular base was made to keep the two servos stable.

The results of the PCB was perfect and everything functioned as planned. When soldering, female and male headers were placed in case components need to be switched out. The cases, after generously being printed by a peer, were not so successful but still adequate. The PCB did not completey fit in the case and the screw holes were way too small. The servo case fit the servos perfectly but, the base was too narrow which lead the motors to move around when scanning. These problems were fixed through the use of sticky tack. Even with these issues, the cases and PCB managed to elevate the projects appearance and in the future, they will be improved in V2.

Media



Using sticky tack to hold down th motor case



Incorrect angle formula on the case (the bottom formula)

YouTube video link: https://www.youtube.com/watch?v=frzUmAhQT3E

Code

Arduino IDE

```
// PROJECT  : The LiDAR Measurement Device
// PURPOSE  : Measures the distance between two objects
// COURSE   : ICS3U
// AUTHOR   : Xander Chin
// DATE     : March 27, 2021
// MCU      : ESP32
// STATUS   : Working
// REFERENCE:
// FIXES    :

//bluetooth library
#include "BluetoothSerial.h"
BluetoothSerial SerialBT;

#define touchPin1 32
#define touchPin2 33
#define threshold 30    //touch pin threshold

bool statusClient;
bool offlineMessage = true;
bool onlineMessage = false;

//-----------------------------//

#include <HardwareSerial.h>   //UART connection to LiDAR
HardwareSerial MySerial2(2);
#include "TFMini.h"

#define RXD2 16
#define TXD2 17

TFMini tfmini;

//-----------------------------//

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>   //I2C connection for OLED

#define displayWidth 128
#define displayHeight 64
#define OLED_RESET -1

Adafruit_SSD1306 display(displayWidth, displayHeight, &Wire, OLED_RESET);

//-------------------------------//
#include <ESP32Servo.h>

Servo myServoX;
Servo myServoY;
#define servoPinX 5
#define servoPinY 18

#define encoderDT 27
#define encoderCLK 26
#define encoderButton 25
```

```
#define offsetDistance 6    //offset distance from the center of the servo
#define minServoX 1
#define maxServoX 177
#define minServoY 5
#define maxServoY 180

bool stateDT;
bool stateCLK;
bool lastStateCLK;
bool lastStateDT;

uint16_t distance;
uint8_t posX;
uint8_t posY;
float realPosX;
float realPosY;
volatile bool movement;
bool up;
bool down;

uint32_t lastTurn;

//--------------------------------//
float savedPosX;
float savedPosY;
uint16_t savedDistance;
float result;
bool buttonPressed = false;

hw_timer_t * timer = NULL;
portMUX_TYPE synch = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

//interrupt
void IRAM_ATTR isr() {
  portENTER_CRITICAL(&synch);
  portEXIT_CRITICAL(&synch);
}

//timer interrupt
void IRAM_ATTR onTimer() {
  portENTER_CRITICAL_ISR(&timerMux);
  movement = true;
  portEXIT_CRITICAL_ISR(&timerMux);
}

void setup() {
  //serial monitor connection
  //Serial.begin(115200);

  //TFmini LiDAR serial connection
  MySerial2.begin(115200);
  MySerial2.setTimeout(1);
  tfmini.begin(&MySerial2);

  //OLED display I2C connection
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

  //bluetooth serial connection
  SerialBT.register_callback(callback);
  SerialBT.begin("ESP32-LiDAR");
  SerialBT.setTimeout(1);
```

```
  //servo setup
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);
  myServoX.setPeriodHertz(50);              //50hz servo
  myServoY.setPeriodHertz(50);              //50hz servo
  myServoX.attach(servoPinX, 200, 5000);    //PWM range
  myServoY.attach(servoPinY, 200, 5000);    //PWM range

  pinMode(encoderCLK, INPUT_PULLUP);
  pinMode(encoderDT, INPUT_PULLUP);
  pinMode(encoderButton, INPUT_PULLUP);

  touchAttachInterrupt(T8, increment, threshold);   //pin 33
  touchAttachInterrupt(T9, decrement, threshold);   //pin 32

  //setup interrupts
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &onTimer, true);
  timerAlarmWrite(timer, 50000, true);

  //text setup for OLED
  setupText(1, 0, 0);
  display.println("LiDAR Measurement Device");
  display.display();
  display.clearDisplay();
}

void loop() {
  distance = tfmini.getDistance() + offsetDistance;
  uint16_t strength = tfmini.getRecentSignalStrength();
  MySerial2.flush();    //clear buffer for no lag

  //receiving data from bluetooth client
  if (SerialBT.available() > 0 && statusClient) {
    String serialData = SerialBT.readStringUntil('.');
    if(serialData.charAt(0) == 'x') {
      posX = serialData.substring(1).toInt();
    }
    if(serialData.charAt(0) == 'y') {
      posY = serialData.substring(1).toInt();
    }
    SerialBT.flush();   //clear buffer for no lag
  }

  //servos overturn a bit
  posX = constrain(posX, minServoX, maxServoX);
  posY = constrain(posY, minServoY, maxServoY);
  realPosX = mapFloat(posX, minServoX, maxServoX, 0, 180);    //actual angle of servos (servos
overturn)
  realPosY = mapFloat(posY, minServoY, maxServoY, 0, 180);    //actual angle of servos (servos
overturn)

  myServoX.write(posX);
  myServoY.write(posY);

  //send data over bluetooth to the client
  if(statusClient) {
    setupText(1, 0, 0);
    display.println("Client BT online");
    display.println("Manual mode: off");
    if(onlineMessage) {
      onlineMessage = false;
      display.display();
    }
    SerialBT.flush();   //prevent lag
    SerialBT.println(String(distance)+","+String(realPosX)+","+String(realPosY)+","+
String(strength));
  }
```

```
  //manual control if the client is offline
  if(!statusClient) {
    if(offlineMessage) {
      offlineMessage = false;
      setupText(1, 0, 0);
      display.println("Client BT offline");
      display.println("Manual mode: on");
      display.display();
      delay(3000);      //pause program and display message for three seconds
    }

    //read encoder and increase/decrease yaw
    stateCLK = digitalRead(encoderCLK);
    stateDT = digitalRead(encoderDT);
    if(lastStateCLK == 1 && lastStateDT == 1) {
      if(stateCLK == 0 || stateDT == 0) {
        stateCLK == 0 ? posX++ : posX--;
        lastTurn = millis();
      }
    }
    lastStateCLK = stateCLK;
    lastStateDT = stateDT;

    //read touch buttons and increase/decrease pitch
    if(movement) {
      if(up) posY++;
      if(down) posY--;
      up = false;
      down = false;
      movement = false;
    }

    //save a point
    if(digitalRead(encoderButton) == 0) {
      buttonPressed = true;
      savedPosX = realPosX;
      savedPosY = realPosY;
      savedDistance = distance;
    } else if(!buttonPressed) {
      savedPosX = realPosX;
      savedPosY = realPosY;
      savedDistance = distance;
    }

    //do calculations
    float x = degToRad(abs(realPosX - savedPosX));
    float y1 = degToRad(savedPosY);
    float y2 = degToRad(realPosY);
    result = cosineLaw(distance, savedDistance, calculateAngle(x, y1, y2));

    //display on OLED
    if(millis() - lastTurn > 1000) {
      setupText(1, 0, 0);
      display.print("Distance: ");
      display.print(distance);
      display.println("cm");
      display.print("Strength: ");
      display.println(strength);
      display.println("x: " + String(realPosX));
      display.println("y: " + String(realPosY));
      display.println("--------------------");
      display.print("Result: ");
      display.print(result);
      display.println("cm");
      display.println("Manual mode: on");
      display.display();
    }
  }
  timerAlarmEnable(timer);    //enable the interrupt
}
```

```
void setupText(uint8_t s, uint8_t x, uint8_t y) {
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(s);
  display.setCursor(x, y);
}

//dectects if bluetooth connection is offline or online
void callback(esp_spp_cb_event_t event, esp_spp_cb_param_t *param) {
  if(event == ESP_SPP_SRV_OPEN_EVT) {
    statusClient = true;
    onlineMessage = true;
  } else if(event == ESP_SPP_CLOSE_EVT) {
    statusClient = false;
    offlineMessage = true;
  }
}

float mapFloat(uint16_t x, uint16_t in_min, uint16_t in_max, uint16_t out_min, float out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

void increment() {
  up = true;
}

void decrement() {
  down = true;
}

//-------------math functions----------------//

float calculateAngle(float x, float y1, float y2) {
  return acos(cos(y1)*cos(y2)*cos(x)+sin(y1)*sin(y2));
}

float cosineLaw(float a, float b, float angle) {
  return sqrt(pow(a, 2)+pow(b, 2)-2*a*b*cos(angle));
}

float degToRad(float deg) {
  return deg*(PI/180);
}

float radToDeg(float rad) {
  return rad*(180/PI);
}
```

Processing – LiDAR Measurement Sketch

```
/ PROJECT  : The LiDAR Measurement Device Bluetooth Control
// PURPOSE  : Creates a point cloud where you can measure distances using bluetooth
// COURSE   : ICS3U
// AUTHOR   : Xander Chin
// DATE     : March 27, 2021
// MCU      : ESP32
// STATUS   : Working
// REFERENCE:
// FIXES    :

//import serial library to communicate with arduino
import processing.serial.*;
Serial myPort;

//set max and min servo positions
static byte minServoX = 1;
static int maxServoX = 177;
static byte minServoY = 5;
static int maxServoY = 180;

float xPos = 0;
float yPos = 0;
float xPosIncrement = 0;
float yPosIncrement = 0;
float distance;
float yaw;
float pitch;
float strength;
float lineDistance;
String serialData = "";
String[] components;
boolean change;
PFont font;

//number of points to measure
int numberOfPoints = 10;

byte dataPoint1 = 0;
byte dataPoint2 = 0;

float[][] data = new float[numberOfPoints][3];

Text[] points = new Text[numberOfPoints];
Text[] pointsInfo = new Text[numberOfPoints];
Text current;
Text selectedPoint1;
Text selectedPoint2;

void setup() {
  size(1300, 800);    //setup 2D canvas

  //serial setup
  String portName = Serial.list()[1];
  myPort = new Serial(this, portName, 115200);
  myPort.bufferUntil('\n');

  //load font
  font = loadFont("AppleMyungjo-50.vlw");
  textFont(font, 50);
```

```
  //initialize array to avoid null pointer exceptions
  points[0] = new Text("Current", 70, 150, 30);
  pointsInfo[0] = new Text("", 70, 100, 30);
  for(byte x = 1; x < numberOfPoints; x++) {
    points[x] = new Text("Point " + x, 1050, 50*x, 30);
    pointsInfo[x] = new Text("x: null  y: null  nullcm", 700, 50*x, 20);
  }

  //current = new Text("Current", 70, 100, 30);
  selectedPoint1 = new Text("Current", 350, 600, 50);
  selectedPoint2 = new Text("Current", 750, 600, 50);
}

void draw() {
  background(0);

  //current distance servo yaw pitch angles
  textSize(30);
  fill(255);
  text(distance + "cm", 70, 200);
  text("Yaw: " + yaw + "°", 70, 250);
  text("Pitch: " + pitch + "°", 70, 300);
  text("Signal Strength: " + strength, 70, 350);

  textSize(50);
  fill(237, 180, 34);
  text("--->", 580, 600);

  lineDistance = calculateDistance();
  fill(255);
  textSize(50);
  text(lineDistance + "cm", 550, 675);

  //display configurable points
  for(byte x = 0; x < numberOfPoints; x++) {
    if(points[x].collision(mouseX, mouseY)) {
      points[x].highlight = true;
      if(x != 0) pointsInfo[x].display();
    } else {
      points[x].highlight = false;
    }
    points[x].display();
  }

  //if hovering over point 1
  if(selectedPoint1.collision(mouseX, mouseY)) {
    selectedPoint1.highlight = true;
  } else {
    selectedPoint1.highlight = false;
  }

  //if hovering over point 2
  if(selectedPoint2.collision(mouseX, mouseY)) {
    selectedPoint2.highlight = true;
  } else {
    selectedPoint2.highlight = false;
  }

  xPos += xPosIncrement;
  yPos += yPosIncrement;
  xPos = constrain(xPos, minServoX, maxServoX);
  yPos = constrain(yPos, minServoY, maxServoY);
```

```
  //send out data to arduino
  if(!change) {
    myPort.write("x" + xPos + ".");
  }
  if(change) {
    myPort.write("y" + yPos + ".");
  }
  change = !change;

  //display text
  selectedPoint1.display();
  selectedPoint2.display();
}

void configurePointInfo(byte x) {
  data[x][0] = distance;
  data[x][1] = yaw;
  data[x][2] = pitch;
  pointsInfo[x].updateTextTo("x: " + data[x][1] + " y: " + data[x][2] + "  " + data[x][0] +
"cm");
}

//happens when serial data is received
void serialEvent(Serial myPort) {
   serialData = myPort.readStringUntil('\n');
   serialData = serialData.substring(0, serialData.length() - 1);
   components = split(serialData, ',');
   distance = float(components[0]);
   yaw = float(components[1]);
   pitch = float(components[2]);
   strength = float(components[3]);

   data[0][0] = distance;
   data[0][1] = yaw;
   data[0][2] = pitch;
}

//class object Text for selecting, changing font and highlighting
public class Text {
  String text;
  int textPosX;
  int textPosY;
  float textWidth;
  int textSize;
  boolean highlight = false;
  boolean selected = false;

  Text(String string, int x, int y, int z) {
    text = string;
    textPosX = x;
    textPosY = y;
    textSize = z;
    textSize(textSize);    //initialize text size before calculating text width
    textWidth = textWidth(string);
  }

  void display() {
    if(highlight) {
      fill(255);
    } else {
      fill(237, 180, 34);
    }

    if(selected) fill(173, 23, 203);

    textSize(textSize);
    text(text, textPosX, textPosY, textWidth);
  }
```

```
  void updateTextTo(String string) {
    text = string;
  }

  boolean collision(float mouseX, float mouseY) {
    if(mouseX >= textPosX && mouseX <= textPosX + textWidth) {
      if(mouseY >= textPosY - textSize && mouseY <= textPosY) {
        return true;
      }
    }
    return false;
  }
}

void keyPressed() {
  if(key == 'a') xPosIncrement = 0.5;
  if(key == 'A') xPosIncrement = 0.1;
  if(key == 'd') xPosIncrement = -0.5;
  if(key == 'D') xPosIncrement = -0.1;
  if(key == 'w') yPosIncrement = 0.5;
  if(key == 'W') yPosIncrement = 0.1;
  if(key == 's') yPosIncrement = -0.5;
  if(key == 'S') yPosIncrement = -0.1;
  if(key == ' ') {
    for(byte x = 0; x < numberOfPoints; x++) {
      if(points[x].highlight) {
        xPos = map(data[x][1], 0, 180, minServoX, maxServoX);
        yPos = map(data[x][2], 0, 180, minServoY, maxServoY);
      }
    }
  }
}

void keyReleased() {
  if(key == 'a') xPosIncrement = 0;
  if(key == 'A') xPosIncrement = 0;
  if(key == 'd') xPosIncrement = 0;
  if(key == 'D') xPosIncrement = 0;
  if(key == 'w') yPosIncrement = 0;
  if(key == 'W') yPosIncrement = 0;
  if(key == 's') yPosIncrement = 0;
  if(key == 'S') yPosIncrement = 0;
}
```

```
void mousePressed() {
  for(byte x = 0; x < numberOfPoints; x++) {
    if(points[x].collision(mouseX, mouseY)) {
      if(!selectedPoint1.selected && !selectedPoint2.selected) {
        configurePointInfo(x);
      }

      if(selectedPoint1.selected) {
        if(x == 0) {
          selectedPoint1.updateTextTo("Current");
        } else {
          selectedPoint1.updateTextTo("Point " + x);
        }
        selectedPoint1.selected = false;
        dataPoint1 = x;
      }

      if(selectedPoint2.selected) {
        if(x == 0) {
          selectedPoint2.updateTextTo("Current");
        } else {
          selectedPoint2.updateTextTo("Point " + x);
        }
        selectedPoint2.selected = false;
        dataPoint2 = x;
      }
    }
  }

  if(selectedPoint1.collision(mouseX, mouseY)) selectedPoint1.selected =
!selectedPoint1.selected;
  if(selectedPoint2.collision(mouseX, mouseY)) selectedPoint2.selected =
!selectedPoint2.selected;
}

void mouseReleased() {
  if(selectedPoint1.selected && selectedPoint2.selected) {
    selectedPoint1.selected = false;
    selectedPoint2.selected = false;
  }
}

//---------------math functions-----------------//
float calculateDistance() {
  float x = degToRad(abs(data[dataPoint1][1] - data[dataPoint2][1]));
  float y1 = degToRad(data[dataPoint1][2]);
  float y2 = degToRad(data[dataPoint2][2]);
  return cosineLaw(data[dataPoint1][0], data[dataPoint2][0], calculateAngle(x, y1, y2));
}

float calculateAngle(float x, float y1, float y2) {
  return acos(cos(y1)*cos(y2)*cos(x)+sin(y1)*sin(y2));
}

float cosineLaw(float a, float b, float angle) {
  return sqrt(pow(a, 2)+pow(b, 2)-2*a*b*cos(angle));
}

float degToRad(float deg) {
  return deg*(PI/180);
}

float radToDeg(float rad) {
  return rad*(180/PI);
}
```

Processing – LiDAR Point Cloud

```
// PROJECT  : The LiDAR Point Cloud
// PURPOSE  : Creates a point cloud where you can measure distances
// COURSE   : ICS3U
// AUTHOR   : Xander Chin
// DATE     : March 27, 2021
// MCU      : ESP32
// STATUS   : Working
// REFERENCE:
// FIXES    :

//serial library to communicate with arduino
import processing.serial.*;
Serial myPort;

//camera controlled with mouse
import peasy.*;
PeasyCam cam;
PeasyDragHandler PanDragHandler;

//font
PFont font;

//sets max servo positions
static byte minServoX = 1;
static int maxServoX = 177;
static byte minServoY = 5;
static int maxServoY = 180;

String serialData;
String[] components;
float distance;
float yaw;
float pitch;
float strength;
float xPos;
float yPos;
float zPos;
ArrayList<PVector> vectors;

//camera offset
float angle = -0.3;
float xOffset = 3;
float yOffset = 152;
float scale = 2.5;
float angleIncrement = 0;
float xOffsetIncrement = 0;
float yOffsetIncrement = 0;
float scaleIncrement = 0;

//customizable elements of the point cloud
boolean change;
boolean play;
boolean writePoints = true;
boolean displayPoints = true;
boolean displayBoundaries;
boolean coordinateAxis = true;
boolean displayHUD = true;
```

```
int lastMove;
float servoX = minServoX;
float servoY = minServoY;
float servoXIncrement = 0;
float servoYIncrement = 0;
float scanInterval = 25;
float scanIntervalInc;
float minServoScanX = minServoX;
float maxServoScanX = maxServoX;
float minServoScanY = minServoY;
float maxServoScanY = map(90, 0, 180, minServoY, maxServoY);
float minServoScanXInc;
float maxServoScanXInc;
float minServoScanYInc;
float maxServoScanYInc;
int xSpeed = 1;

//info colours and number of points
byte numberOfPoints = 9;
boolean[] pressed = new boolean[numberOfPoints];
float[][][] data = new float[numberOfPoints][2][3];
float[] distancePoint = new float[numberOfPoints];
color[] colour = {color(255, 0, 0), color(255, 165, 0), color(255, 255, 0),
                  color(165, 225, 0), color(0, 255, 0), color(0, 255, 255),
                  color(0, 0, 255), color(165, 0, 255), color(255, 0, 255) };

Info[] info = new Info[numberOfPoints];
IntList distanceColours;
IntList strengthColours;
byte mode = 0;
String textMode;
int maxColourDistance = 500;
int maxColDistanceInc = 0;
int maxDistance = 1200;
int maxDistanceInc = 0;
int maxStrength = 2000;

void setup() {
  size(1400, 800, P3D);                                    //setup canvas
  perspective(PI/3.0, (float)width/height, 1, 100000);     //prevent clipping
  vectors = new ArrayList<PVector>();                      //list to store data from arduino
  distanceColours = new IntList();                         //list to store data from arduino
  strengthColours = new IntList();                         //list to store data from arduino

  //camera for movement
  cam = new PeasyCam(this, 0, 0, 0, 1200);
  cam.setMinimumDistance(0.01);
  cam.setMaximumDistance(10000);
  PanDragHandler = cam.getPanDragHandler();
  cam.setLeftDragHandler(PanDragHandler);

  //set up bluetooth serial connection
  String portName = Serial.list()[1];
  myPort = new Serial(this, portName, 115200);
  myPort.bufferUntil('\n');

  font = loadFont("AppleMyungjo-50.vlw");    //load font into sketch
  textFont(font, 50);

  //initialize customizable point info
  for(byte x = 0; x < numberOfPoints; x++) {
    info[x] = new Info((x+1) + ":", distancePoint[x], 1050, 230+(30*x), 20, colour[x]);
  }
}
```

```
void draw() {
  background(0);                          //set background to black
  rotateY(angle);                         //rotate everything by angle

  //current position of lidar sensor
  stroke(255, 0, 0);
  strokeWeight(3);
  line(xOffset, yOffset, 0, xPos * scale + xOffset, -zPos * scale + yOffset, -yPos * scale);

  for(int index = 0; index < vectors.size(); index++) {
    PVector v = vectors.get(index);
    color pointColour;
    float magnitude = v.mag();

    //colour modes
    switch(mode) {
      case 0:
        pointColour = color(255);
        textMode = "Regular";
        break;
      case 1:
        pointColour = distanceColours.get(index);
        textMode = "Distance";
        break;
      case 2:
        pointColour = strengthColours.get(index);
        textMode = "Strength";
        break;
      default:
        pointColour = color(255);
        textMode = "Regular";
        break;
    }

    //display points in 3d space
    stroke(pointColour);
    strokeWeight(1);
    if(displayPoints && magnitude <= maxDistance) {
      point(v.x * scale + xOffset, -v.z * scale + yOffset, -v.y * scale);
    }
  }

  //create lines from selected points in the scan
  for(byte i = 0; i < numberOfPoints; i++) {
    float distance1 = data[i][0][0];
    float distance2 = data[i][1][0];
    float yaw1 = data[i][0][1];
    float yaw2 = data[i][1][1];
    float pitch1 = data[i][0][2];
    float pitch2 = data[i][1][2];

    if(pressed[i]) {
      distance2 = distance;
      yaw2 = yaw;
      pitch2 = pitch;
    }
    float azimuth = degToRad(abs(yaw1 - yaw2));
    float elevation1 = degToRad(pitch1);
    float elevation2 = degToRad(pitch2);
    distancePoint[i] = cosineLaw(distance1, distance2, calculateAngle(azimuth, elevation1, ele
vation2));

    float x1 = coordinateX(distance1, yaw1, pitch1);
    float y1 = coordinateY(distance1, yaw1, pitch1);
    float z1 = coordinateZ(distance1, pitch1);

    float x2 = coordinateX(distance2, yaw2, pitch2);
    float y2 = coordinateY(distance2, yaw2, pitch2);
    float z2 = coordinateZ(distance2, pitch2);
```

```
    info[i].createLine(x1*scale+xOffset, -z1*scale+yOffset, -y1*scale, x2*scale+xOffset, -
z2*scale+yOffset, -y2*scale);
  }

  //display scan boundaries
  if(displayBoundaries) {
    float minX = map(minServoScanX, minServoX, maxServoX, 0, 180);
    float maxX = map(maxServoScanX, minServoX, maxServoX, 0, 180);
    float minY = map(minServoScanY, minServoY, maxServoY, 0, 180);
    float maxY = map(maxServoScanY, minServoY, maxServoY, 0, 180);

    float x1 = coordinateX(2000, minX, minY);
    float y1 = coordinateY(2000, minX, minY);
    float z1 = coordinateZ(2000, minY);

    float x2 = coordinateX(2000, minX, maxY);
    float y2 = coordinateY(2000, minX, maxY);
    float z2 = coordinateZ(2000, maxY);

    float x3 = coordinateX(2000, maxX, maxY);
    float y3 = coordinateY(2000, maxX, maxY);
    float z3 = coordinateZ(2000, maxY);

    float x4 = coordinateX(2000, maxX, minY);
    float y4 = coordinateY(2000, maxX, minY);
    float z4 = coordinateZ(2000, minY);

    stroke(255, 0, 0);
    strokeWeight(1);
    line(xOffset, yOffset, 0, x1*scale+xOffset, -z1*scale+yOffset, -y1*scale);
    line(xOffset, yOffset, 0, x2*scale+xOffset, -z2*scale+yOffset, -y2*scale);
    line(xOffset, yOffset, 0, x3*scale+xOffset, -z3*scale+yOffset, -y3*scale);
    line(xOffset, yOffset, 0, x4*scale+xOffset, -z4*scale+yOffset, -y4*scale);
  }

  //display coordinate axis
  if(coordinateAxis) {
    stroke(255);
    strokeWeight(1);
    line(xOffset, yOffset, 0, 300*scale+xOffset, yOffset, 0);
    line(xOffset, yOffset, 0, -300*scale+xOffset, yOffset, 0);
    line(xOffset, yOffset, 0, xOffset, yOffset, -300*scale);
    line(xOffset, yOffset, 0, xOffset, yOffset, 300*scale);
    line(xOffset, yOffset, 0, xOffset, -100, 0);
    fill(255);
    textSize(50);
    text("0°", 300*scale+xOffset, yOffset, 0);
    text("90°", xOffset, yOffset, -300*scale);
    text("180°", -300*scale+xOffset, yOffset, 0);
  }
```

```
  //2D text creation showing info
  if(displayHUD) {
    cam.beginHUD();
    textSize(20);
    fill(255);
    String state1 = displayPoints ? "ON" : "OFF";
    String state2 = writePoints ? "ON" : "OFF";
    String state3 = displayBoundaries ? "ON" : "OFF";
    String state4 = coordinateAxis ? "ON" : "OFF";
    float realMinDomainX = map(minServoScanX, minServoX, maxServoX, 0, 180);
    float realMaxDomainX = map(maxServoScanX, minServoX, maxServoX, 0, 180);
    float realMinRangeY = map(minServoScanY, minServoY, maxServoY, 0, 180);
    float realMaxRangeY = map(maxServoScanY, minServoY, maxServoY, 0, 180);
    text("Display Points: " + state1, 1050, 50);
    text("Create Points: " + state2, 1050, 70);
    text("Display Scan Boundaries: " + state3, 1050, 90);
    text("Axis: " + state4, 1050, 110);

    text(distance + "cm", 50, 50);
    text("Signal Strength: " + strength, 50, 70);
    text("Yaw: " + yaw + "°", 50, 90);
    text("Pitch: " + pitch + "°", 50, 110);

    text("Max Distance: " + maxDistance + "cm", 1050, 570);
    text("Max Colour Distance: " + maxColourDistance + "cm", 1050, 590);
    text("Scan Interval: " + scanInterval + "ms", 1050, 610);
    text("Domain of Scan: (" + nf(realMinDomainX, 0, 1) + ", " + nf(realMaxDomainX, 0, 1) +
")", 1050, 650);
    text("Range of Scan: (" + nf(realMinRangeY, 0, 1) + ", " + nf(realMaxRangeY, 0, 1) + ")",
1050, 670);
    text("Mode: " + textMode, 1050, 720);

    for(byte x = 0; x < numberOfPoints; x++) {
      String selected = pressed[x] ? "•" : " ";
      info[x].updateInfoTo(selected + (x+1) + ":", distancePoint[x]);
      info[x].display();
    }
    cam.endHUD();
  }

  //servo scanning
  if(play) {
    if(millis() - scanInterval > lastMove) {
      servoX = servoX + xSpeed;
      if(servoX > maxServoScanX) {
        servoX = maxServoScanX;
        xSpeed = -1;
        servoY++;
      } else if(servoX < minServoScanX) {
        servoX = minServoScanX;
        xSpeed = 1;
        servoY++;
      }
      if(servoY > maxServoScanY) {
        writePoints = false;
        servoX = minServoScanX;
        servoY = minServoScanY;
        play = false;
      }
      lastMove = millis();
    }
  }
```

```
  servoX += servoXIncrement;
  servoY += servoYIncrement;
  minServoScanX += minServoScanXInc;
  maxServoScanX += maxServoScanXInc;
  minServoScanY += minServoScanYInc;
  maxServoScanY += maxServoScanYInc;
  minServoScanX = constrain(minServoScanX, minServoX, maxServoScanX + 5);
  maxServoScanX = constrain(maxServoScanX, minServoScanX + 5, maxServoX);
  minServoScanY = constrain(minServoScanY, minServoY, maxServoScanY + 5);
  maxServoScanY = constrain(maxServoScanY, minServoScanY + 5, maxServoY);
  servoX = constrain(servoX, minServoX, maxServoX);
  servoY = constrain(servoY, minServoY, maxServoY);

  //send data to arduino
  if(!change) {
    myPort.write("x" + servoX + ".");
  }
  if(change) {
    myPort.write("y" + servoY + ".");
  }
  change = !change;

  angle += angleIncrement;
  xOffset += xOffsetIncrement;
  yOffset += yOffsetIncrement;
  scale += scaleIncrement;
  maxDistance += maxDistanceInc;
  maxColourDistance += maxColDistanceInc;
  scanInterval += scanIntervalInc;
  maxDistance = constrain(maxDistance, 0, 2000);
  maxColourDistance = constrain(maxColourDistance, 0, 2000);
  scanInterval = constrain(scanInterval, 2, 100);
}

//if data is sent over from the arudino
void serialEvent(Serial myPort) {
   serialData = myPort.readStringUntil('\n');
   serialData = serialData.substring(0, serialData.length() - 1);
   components = split(serialData, ',');
   distance = float(components[0]);
   yaw = float(components[1]);
   pitch = float(components[2]);
   strength = float(components[3]);

   xPos = coordinateX(distance, yaw, pitch);
   yPos = coordinateY(distance, yaw, pitch);
   zPos = coordinateZ(distance, pitch);

   color colourDistance = color(255);
   float q1 = maxColourDistance >> 2;
   float mid = maxColourDistance >> 1;
   float q3 = (maxColourDistance >> 1) + (maxColourDistance >> 2);
```

```
    //color gradient of distance
    if(distance <= q1) {
      colourDistance = lerpColor(color(255, 0, 0), color(255, 255, 0), norm(distance, 0, q1));
    } else if(distance >= q1 && distance <= mid) {
      colourDistance = lerpColor(color(255, 255, 0), color(0, 255, 0), norm(distance, q1,
mid));
    } else if(distance >= mid && distance <= q3) {
      colourDistance = lerpColor(color(0, 255, 0), color(0, 255, 255), norm(distance, mid,
q3));
    } else if(distance >= q3) {
      colourDistance = lerpColor(color(0, 255, 255), color(0, 0, 255), norm(distance, q3,
maxColourDistance));
    } else {
      colourDistance = color(255);
    }

    //color gradient of strength
    float i = norm(strength, 0, maxStrength);
    color colourStrength = lerpColor(color(255, 0, 0), color(0, 255, 0), i);

    //add to array list of vectors and colors
    if(writePoints) {
      distanceColours.append(color(colourDistance));
      strengthColours.append(color(colourStrength));
      vectors.add(new PVector(xPos, yPos, zPos));
    }
}

public class Info {
  String text;
  float measurement;
  int textPosX;
  int textPosY;
  int textSize;
  color colour;

  Info(String string, float m, int x, int y, int z, color c) {
    text = string;
    measurement = m;
    textPosX = x;
    textPosY = y;
    textSize = z;
    colour = c;
  }

  void display() {
    fill(colour);
    textSize(textSize);
    text(text, textPosX, textPosY);
    text(nf(measurement, 0, 2) + "cm", textPosX + 100, textPosY);
  }

  void updateInfoTo(String string, float m) {
    text = string;
    measurement = m;
  }

  void createLine(float x1, float y1, float z1, float x2, float y2, float z2) {
    stroke(colour);
    strokeWeight(2);
    line(x1, y1, z1, x2, y2, z2);
  }
}
```

```
void keyPressed() {
  if(key == 'w' && !play) servoYIncrement = 0.5;
  if(key == 'W' && !play) servoYIncrement = 0.1;
  if(key == 's' && !play) servoYIncrement = -0.5;
  if(key == 'S' && !play) servoYIncrement = -0.1;
  if(key == 'a' && !play) servoXIncrement = 0.5;
  if(key == 'A' && !play) servoXIncrement = 0.1;
  if(key == 'd' && !play) servoXIncrement = -0.5;
  if(key == 'D' && !play) servoXIncrement = -0.1;
  if(key == 'f') xOffsetIncrement = 1f;
  if(key == 'h') xOffsetIncrement = -1f;
  if(key == 't') yOffsetIncrement = 1f;
  if(key == 'g') yOffsetIncrement = -1f;
  if(key == ' ') play = !play;
  if(key == 'c') writePoints = !writePoints;
  if(key == 'z') displayPoints = !displayPoints;
  if(key == '-') maxDistanceInc = -3;
  if(key == '=') maxDistanceInc = 3;
  if(key == '_') maxColDistanceInc = -3;
  if(key == '+') maxColDistanceInc = 3;
  if(key == '[') scanIntervalInc = -0.5;
  if(key == ']') scanIntervalInc = 0.5;
  if(key == 'L' && !play) maxServoScanXInc = 0.5;
  if(key == '"' && !play) maxServoScanXInc = -0.5;
  if(key == 'l' && !play) minServoScanXInc = 0.5;
  if(key == 39 && !play) minServoScanXInc = -0.5;    // ' key
  if(key == 'p' && !play) maxServoScanYInc = 0.5;
  if(key == ';' && !play) maxServoScanYInc = -0.5;
  if(key == 'P' && !play) minServoScanYInc = 0.5;
  if(key == ':' && !play) minServoScanYInc = -0.5;
  if(key == ',') coordinateAxis = !coordinateAxis;
  if(key == '.') displayBoundaries = !displayBoundaries;
  if(key == '/') displayHUD = !displayHUD;
  if (key =='x') {    // erase all points
    vectors.clear();
    distanceColours.clear();
    strengthColours.clear();
  }
  if(key == 'm') {
    if(mode < 2) mode++;
      else if(mode >= 2) mode = 0;
  }
  if (key == 'r' && !play) {
    writePoints = false;
    servoX = minServoScanX;
    servoY = minServoScanY;
  }
  if (key == CODED) {
    if (keyCode == LEFT) angleIncrement = 0.015f;    // rotate left
    if (keyCode == RIGHT) angleIncrement = -0.015f;  // rotate right
    if (keyCode == UP) scaleIncrement = 0.02f;       //zoom in
    if (keyCode == DOWN) scaleIncrement = -0.02f;    //zoom out
  }
  for(byte x = 0; x < numberOfPoints; x++) {
    if(key == x + 49) {
      data[x][int(pressed[x])][0] = distance;
      data[x][int(pressed[x])][1] = yaw;
      data[x][int(pressed[x])][2] = pitch;
      pressed[x] = !pressed[x];
    }
  }
}
```

```
void keyReleased() {
  if(key == 'w' && !play) servoYIncrement = 0;
  if(key == 'W' && !play) servoYIncrement = 0;
  if(key == 's' && !play) servoYIncrement = 0;
  if(key == 'S' && !play) servoYIncrement = 0;
  if(key == 'a' && !play) servoXIncrement = 0;
  if(key == 'A' && !play) servoXIncrement = 0;
  if(key == 'd' && !play) servoXIncrement = 0;
  if(key == 'D' && !play) servoXIncrement = 0;
  if(key == 'f') xOffsetIncrement = 0f;
  if(key == 'h') xOffsetIncrement = 0f;
  if(key == 't') yOffsetIncrement = 0f;
  if(key == 'g') yOffsetIncrement = 0f;
  if(key == '-') maxDistanceInc = 0;
  if(key == '=') maxDistanceInc = 0;
  if(key == '_') maxColDistanceInc = 0;
  if(key == '+') maxColDistanceInc = 0;
  if(key == '[') scanIntervalInc = 0;
  if(key == ']') scanIntervalInc = 0;
  if(key == 'L' && !play) maxServoScanXInc = 0;
  if(key == '"' && !play) maxServoScanXInc = 0;
  if(key == 'l' && !play) minServoScanXInc = 0;
  if(key == 39 && !play) minServoScanXInc = 0;
  if(key == 'p' && !play) maxServoScanYInc = 0;
  if(key == ';' && !play) maxServoScanYInc = 0;
  if(key == 'P' && !play) minServoScanYInc = 0;
  if(key == ':' && !play) minServoScanYInc = 0;
  if (key == CODED) {
    if (keyCode == LEFT) angleIncrement = 0f;
    if (keyCode == RIGHT) angleIncrement = 0f;
    if (keyCode == UP) scaleIncrement = 0f;
    if (keyCode == DOWN) scaleIncrement = 0f;
  }
}

//------------math functions-------------//
/*float calculateDistance() {
  float x = degToRad(abs(data[dataPoint1][1] - data[dataPoint2][1]));
  float y = degToRad(abs(data[dataPoint1][2] - data[dataPoint2][2]));
  return cosineLaw(data[dataPoint1][0], data[dataPoint2][0], calculateAngle(x, y));
}*/

float coordinateX(float d, float x, float y) {
  return d*cos(degToRad(x))*cos(degToRad(y));
}

float coordinateY(float d, float x, float y) {
  return d*sin(degToRad(x))*cos(degToRad(y));
}

float coordinateZ(float d, float y) {
  return d*sin(degToRad(y));
}

float calculateAngle(float x, float y1, float y2) {
  return acos(cos(y1)*cos(y2)*cos(x)+sin(y1)*sin(y2));
}

float cosineLaw(float a, float b, float angle) {
  return sqrt(pow(a, 2)+pow(b, 2)-2*a*b*cos(angle));
}

float degToRad(float deg) {
  return deg*(PI/180);
}

float radToDeg(float rad) {
  return rad*(180/PI);
}
```

Reflection

I am tired. Very, very tired and exhausted from working on this project, writing this report and making the video. So tired in fact that I really just want to go to sleep. There are some things that I do want to fix such as the math equation sizes, inserting some more pictures, scaling down the image sizes, likely spelling and grammar mistakes, and the list goes on but I really do not have the energy right now to do so (it is 3am). I will definitely fix up this report first thing in the morning however.

Even with the amount of work I put in and life sucked out of me, I loved making it from prototyping it, to designing my first PCB in EAGLE, to developing the processing sketches and seeing them scan the room, to sitting down and trying to solve the required math equations and the list goes on an on. I love how I got to incorporate math as well as the three domains of an engineer which are hardware, software, and design. Throughout the process, I had to problem solve in every domain which helped me grow as a person, however, I believe I tried to solve to many problems and incorporate too many ideas as I ran out of time and am submitting my report after the deadline. This is also why I am currently exhausted. Therefore, the experience has taught me to focus more on actually completeing the report and video instead of just focusing on building because building is fun while writing… not so much. But, even with this setback, this was definitely my favorite project to complete as I really care and had a lot of fun doing it.

Project 2.7: Mechanical

Purpose

The purpose of this project is to explore and make use of the different types of motors available which are DC hobby motors, servo motors, and stepper motors.

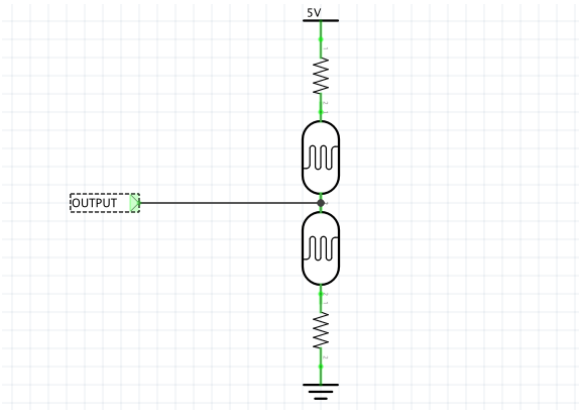
References

http://darcy.rsgc.on.ca/ACES/TEI3M/2021/Tasks.html
https://www.makecrate.club/how-does-a-photoresistor-work/71182/
https://www.gadgetronicx.com/attiny85-timer-tutorial-generating-time-delay-interrupts/
https://www.gadgetronicx.com/attiny85-compare-match-tutorial-interrupts/
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet-Summary.pdf

Procedure

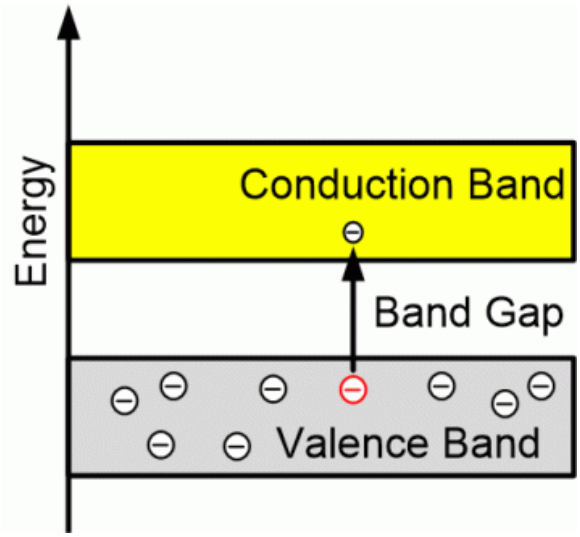
Parts Table	
Quantity	Description
1	ATtiny85
1	Mini breadboard
1	MG995 servo motor
1	5mm red LED
2	10 kΩ LDR
2	4.7 kΩ fixed resistor

For this project, a simple electronic sunflower was made that followed a source of light or avoided darkness which adheres to the myth that sunflowers always face the sun. It also provides more efficiency if a solar cell is mounted onto it as it will always receive the most amount of sunlight that it can. To begin, a servo motor as well as an extra pan and tilt kit came in handy to create the base that turns left and right. The servo motor, an MG995, was the same one used in the LiDAR measurement device. As discussed before, it uses three pins, VCC, GND, and the PWM pin controlling the servos position from 0 to 180 degrees. At first it was decided that two servo motors were used for a full range of motion, however, this proved too complicated and so only one was used. The microcontroller driving the device is an ATtiny85.



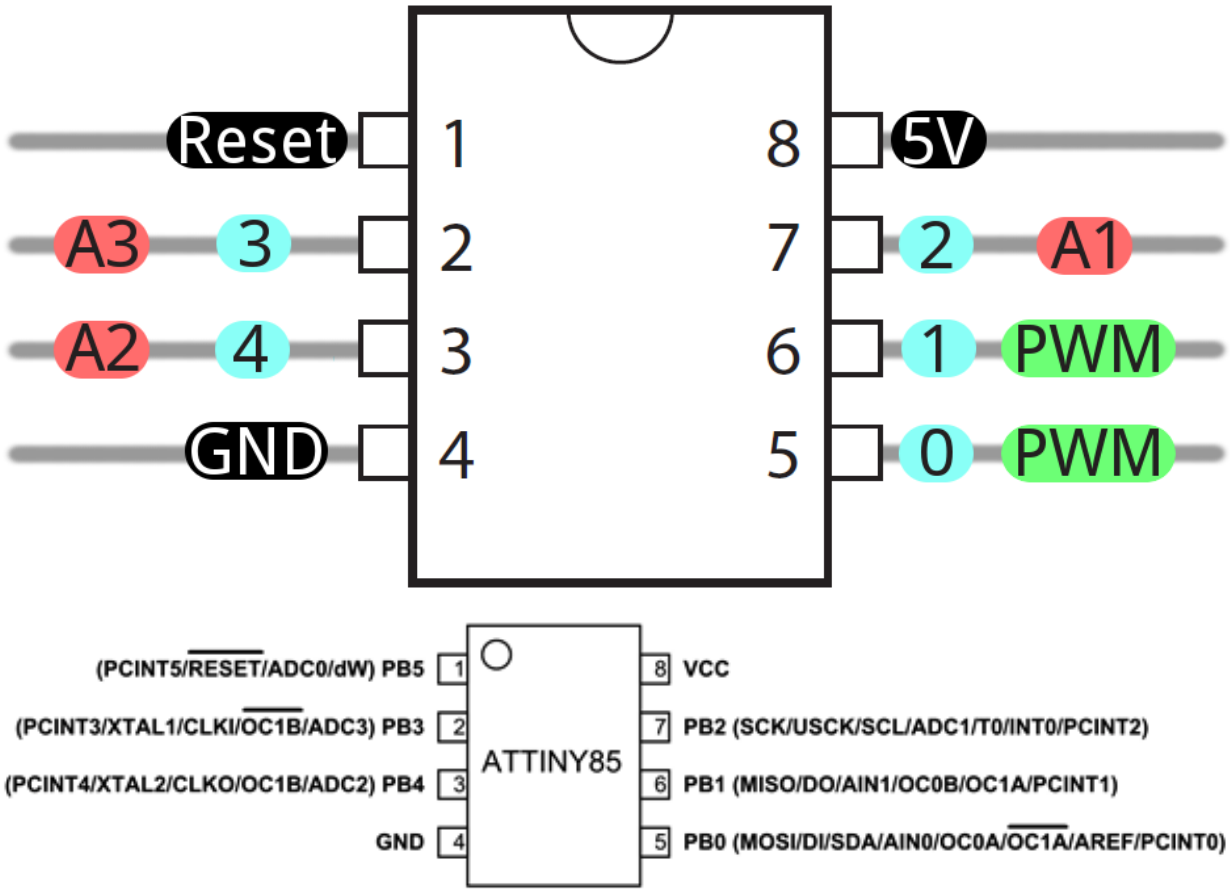
With the base complete, the light sensing mechanism was needed to move the device around accordingly. Here, two equally resistant photoresistors or light depended resistors (LDR) were used to detect which side of the device had greater amount of light. To do this, the photoresistors were placed in a classic voltage divider formation shown on the right. Since these sensors creates a variable resistance in between them according to the light level, with light decreasing the resistance and darkness increasing it, the output voltage of the circuit varies according to the comparison of light levels on both sensors. With a higher light level on the first LDR, resistance drops creating a voltage higher than half of the input voltage on the output pin and when a higher light level is present on the second LDR, the voltage drops below half of the input voltage. This simple and effective design only uses up one analog pin on the NANO to read the incoming voltage attained from the circuit using the revisited Kirchhoff's voltage law:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$



Photoresistors are quite a common variable resistor used as a sensor in many analog and digital circuits with its iconic look pictured on the right. It varies the amount of current flow through its construction material of a semi-conductor, similar to a transistor. The procedure works almost identically to one. Inside, it has three bands with electrons arranged in them so that they have similar energy levels to one another. These bands are the valance band, where electrons cannot move the least freely and the conduction band, where electrons can move around almost normally. Separating these two bands is a gap called the energy gap. This layout of bands and gaps, again are similar to an NPN or PNP layout. Electrons are restricted in their flow depending on the state of the valance band and when the LDR is saturated with light, the electrons in that band become excited, allowing electrons to flow across the energy gap and to the conduction band, thereby lowering resistance and allowing more current to flow. Manufacturing of these variable resistors vary in voltage, some reaching a few MOhms in complete darkness while others.

With the simple hardware out of the way comes the also bone simple software. There are a few differences as shown in the code section that will confuse many Arduino enthusiasts as some low-level coding instead of the usual high-level techniques were introduced to control the ATtiny85. This involves directly manipulating ports instead of using the in-built libraries and functions that hide the complexity of low-level coding. Uploading code to the ATtiny85 is the same as uploading code to an ATmega328P.



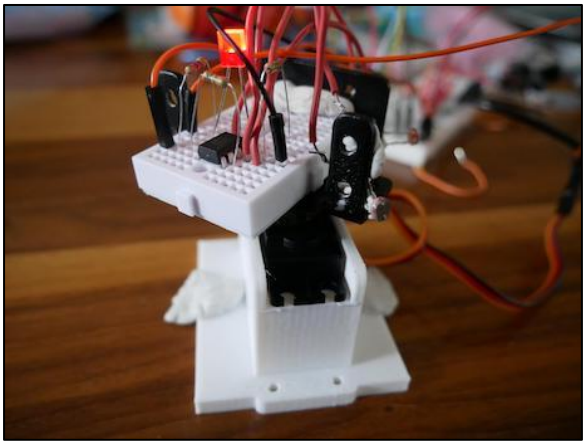
Here pictured above are detailed diagrams of the ATtiny85 with its GPIO pins and port names. These port names are crucial to manipulating the desired pins in the Arduino IDE as each pin is assigned a port name. On the ATtiny85, there is only one port type which is defined by the letter B proceeded by the letter P for port. Examples of manipulating these pins using port names can be viewed in the commented code used for this project located in the code section.

Along with pins and ports come the timers in the ATtiny85 which include two 8-bit timers called timer0 and timer1. Since these are 8-bit timers, the popular Servo library cannot be used as it requires a 16-bit timer to control PWM on the pins. Therefore, a library was not used and instead, direct port and timer manipulation was implemented to create a PWM signal. Please note that the information provides is a brief summary and may be incorrect as port and timer manipulation are new and advanced concepts. More information can be found in the references section.

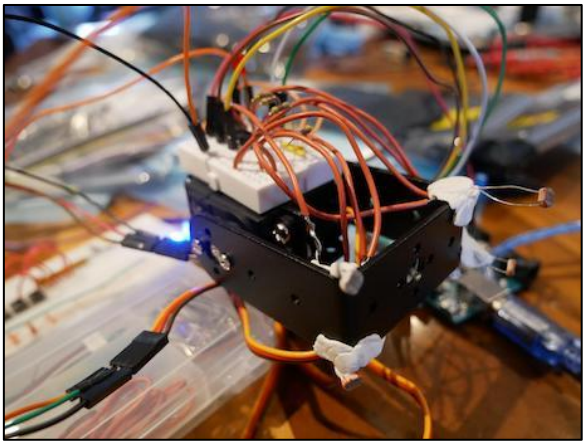
The timer on the ATtiny85 that is being used is timer0 controlled by accessing certain registers. It is setup in normal mode through the TCCROA register and then the bits WGM0, WGM01 and WGM02 of the same register are set to a binary value of 010 respectfully in order to set the timer in CTC mode. This mode counts up in equal time intervals in a timer register called TCNT0 from 0 to a pre-set value stored in an 8-bit timer register called OCR0A, therefore a max pre-set value of 255 is allowed. When the count in the TCNT0 register matches the pre-set value, it sets a bit called OCF0A high in the TIFR flag register. Monitoring the bit in the register allows one to see how much time has passed, acting as a delay or millis function, if one sets the time interval in between increments of the TCNT0. These time intervals between increments are set through prescaled values of the original microcontroller speed of the ATtiny85. These prescaler values divide the speed, which is 8 MHz as the ATtiny85 can only support up to 10 MHz, by certain powers of two, allowing for longer delay times. This combined with the pre-set bits in the OCR0A register allow for measuring time with decent accuracy.

In this project, a microsecond timer similar to the delayMicroseconds function is created by setting the time increment is to 8 MHz by applying no prescalers through setting the TCCR0B to a binary value of 0. This means that it takes 8 increments of the TCNT0 register for one microsecond to pass, therefore, by setting the OCR0A register to 8, the OCF0A bit goes high in the TFIR flag register every microsecond. Thereby, through monitoring this bit, a PWM signal can be generated to control the servo by making a while loop, akin to a delay function, that exits once a certain count of loops that are each one microsecond long have passed. To fully understand the code, take a look at the commented code section below.

Media



The ATtiny85 sits on top of the device



The very messy previous version using four LDR's for omnidirectional movement

YouTube video link: https://youtu.be/K975ZXLuv1w

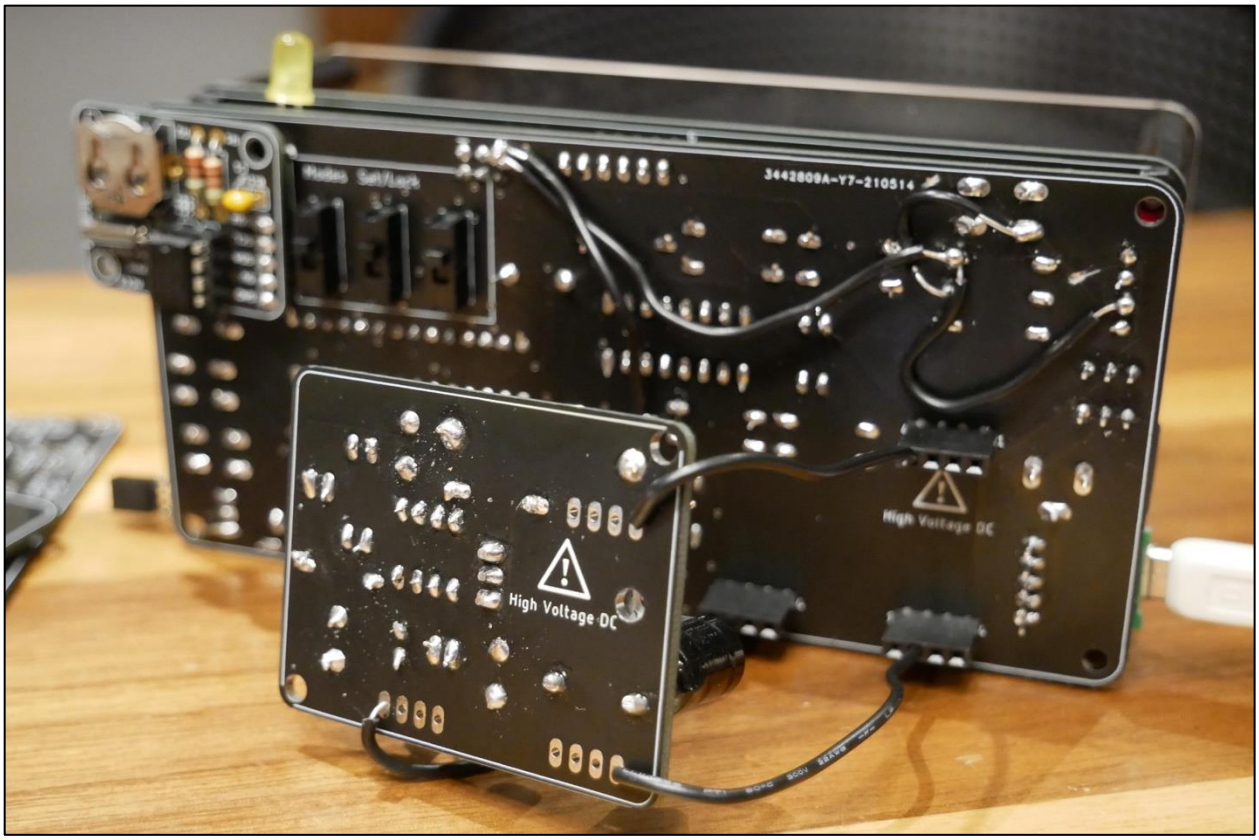
Code

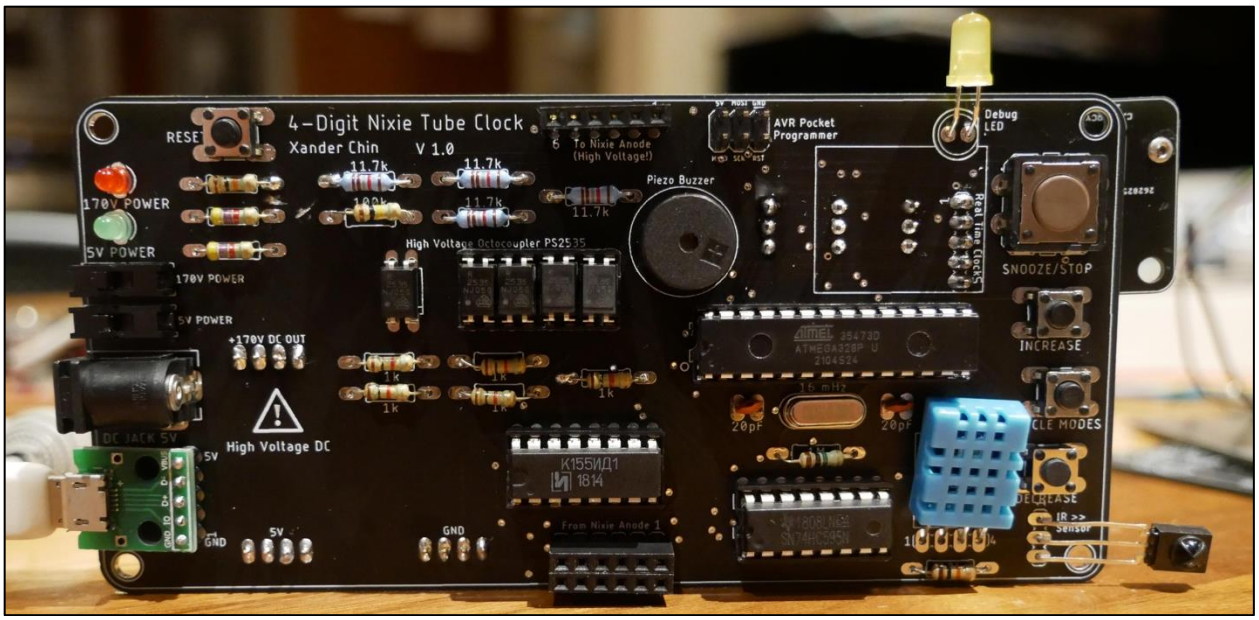
```
// PROJECT  : The Electronic Sunflower
// PURPOSE  : Using mechanical motors to create a project
// COURSE   : ICS3U
// AUTHOR   : Xander Chin
// DATE     : May 7, 2021
// MCU      : ATtiny85
// STATUS   : Working
// REFERENCE:

uint8_t pos = 0;

void setup() {
  DDRB |= (1 << PB0) | (1 << PB1);  // or B11;, set PB0 and PB1 to output
  TCCR0A = 0x00;          //set to normal mode of timer0
  TCCR0B = 0x00;          //
  TCCR0B |= B001;         //or (1 << CS00); toggle bit CS00 on, prescaling with 1 (divides
clock MHz by 1)
  TCCR0A |= (1 << WGM01); //toggle mode and compare match  mode
  OCR0A = 7;//compare value TCNT0. Should be 8 (8 MHz * 8 = 1 microsecond). Errors of time = 7
  TCNT0 = 0;          //increments up by one to value in OCR0A every clock MHz/prescaler value
}

void loop() {
  if(analogRead(A2) < 450)
    pos++;
    PORTB &= ~(1 << PB1);       //indicator LED off
  else if(analogRead(A2) > 550)
    pos--;
    PORTB &= ~(1 << PB1);       //indicator LED off
  else
    PORTB |= (1 << PB1);        //indicator LED on

  pos = constrain(pos, 1, 180);
  servoWrite(pos);
}

void servoWrite(uint8_t pos) {
  uint16_t pulse = (pos << 3) + (pos << 1) + 500;   //converts into microsecond pulses
  PORTB |= (1 << PB0);                              //binary 1
  timer(pulse);                                     //sets the position of servo (PWM)
  PORTB &= ~(1 << PB0);                             //binary 1
  timer(50000);                                     //50 milliseconds (for 50 Hz motors)
}

void timer(uint32_t microSeconds) {       //similar to delayMicroseconds()
  uint32_t i = 0;
  while (i <= microSeconds >> 1) {        //should not be >> 1; again errors in timing
    while ((TIFR & (1 << OCF0A) ) == 0);  //waiting to OCF0A flag bit high
    TIFR |= (1 << OCF0A);                 //toggling OCF0A
    i++;                                  //each loop is approx one microsecond
  }
}
```

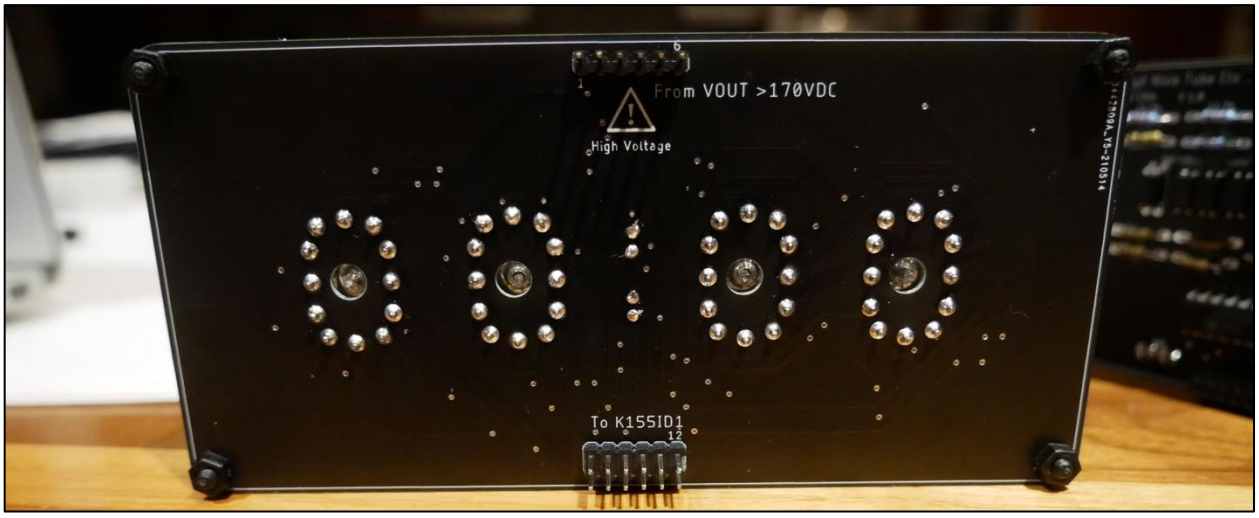
Reflection

I saw this DER submission as an easier one where I tried to get back into the routine of submitting one after a long break of doing so since the medium ISP submission. I made my project simple due to a couple of reasons. One, I myself am not too interested at the moment in mechanics and motors as I do not have advanced 3D printing design skills that enhance and are almost required for any interesting mechanical project. Two, I was quite busy over the past week as summative after summative kept pouring in from other classes and currently, my main priority is getting my long ISP working and stable. That is to say though that I did have to experiment around and try quite a bit. At first I tried to implement a full range of motion for the electronic sunflower which proved to be very messy from the amount of wires and sticky tack present. When resorting to a one dimensional range of motion, I decided to try and make the project work with an ATtiny85 which proved a lot of work as their timers do not support the popular servo library and many others I tried using. So I decided to try and control the servos using low-level coding which I found to be frustrating but eventually I began to understand how the port and timer registers worked in the ATtiny85 which will definitely prove useful in other projects.

Project 2.8 (ISP – Long): The IR Nixie Gecko




4-Digit Nixie Tube Clock
Xander Chin
V 1.0
RESET
170V POWER
5V POWER
High Voltage Octocoupler PS2535
Piezo Buzzer
AVR Pocket Programmer
Debug LED
SNOOZE/STOP
INCREASE
DECREASE
IR >> Sensor
+170V DC OUT
High Voltage DC
DC JACK 5V
GND
5V
To Nixie Anode (High Voltage!)
From Nixie Anode 1
К155ИД1
1814


From VOUT >170VDC
High Voltage
To K155ID1


MAX1771
DC Boost Converter
VOUT
0.1uF
Xander Chin
1364 Taylor HV Power Supply Breakout
45k
5V
GND

Base Clock Parts Table	
Quantity	Description
1	Base clock PCB
1	ATmega328P
1	5V female USB micro
1	5V DC jack
5	SPDT switches
5	Momentary PBNO
1	3mm red LED
1	5mm yellow LED
1	3mm green LED
2	0.1 μF capacitor
2	10 kΩ fixed resistor
1	100 kΩ fixed resistor
1	1 MΩ fixed resistor
2	4.7 kΩ fixed resistor
4	11.7 kΩ ½ W fixed resistor
5	1 kΩ fixed resistor
1	DS1307 PCB + parts
5	PS2535 high voltage optocouplers
1	595 shift register
1	K155ID1
1	Piezo buzzer
1	Sharp GP1UX511QS sensor
1	DHT11 sensor
*****	Male headers
*****	Female headers

Digit Display Parts Table	
Quantity	Description
1	Digit Display PCB
4	IN-12A nixie tubes
2	Nixie bulbs
1	2D custom acrylic board
*****	Male headers
*****	M3 Standoffs + screws + nuts

Boost Converters Parts Table	
Quantity	Description
1	MAX1771 boost converter PCB
2	MAX1771 IC
1	230 uH inductor
1	EPG-10 1A diode
1	IRF740 MOSFET
1	100 uF 25V capacitor
1	10 uF 25V capacitor
2	0.1 uF capacitor
1	4.7 uF 400V capacitor
1	0.1 uF 250V capacitor
1	680 pF 250V capacitor
1	11.7 kΩ fixed resistor
1	1 MΩ fixed resistor
1	330 kΩ fixed resistor
1	HV Taylor 1364 breakout PCB
1	45 kΩ fixed resistor

IR remote	
Quantity	Description
1	Arduino NANO
1	Full sized breadboard
1	3mm Infrared LED
1	680 Ω fixed resistor
5	Momentary PBNO
1	DAC R2R ladder
5	Pulldown resistors

Purpose

The purpose of this project was to explore our interests in engineering by creating a project that encompasses the three branches: hardware, software, and design. For my project I decided to create a four-digit nixie clock that can be controlled through infrared as I am a big fan of the warm glow that illuminates their digits. The circuit and design will be similar to the ACES 2014-2015 Gecko clock.

Theory

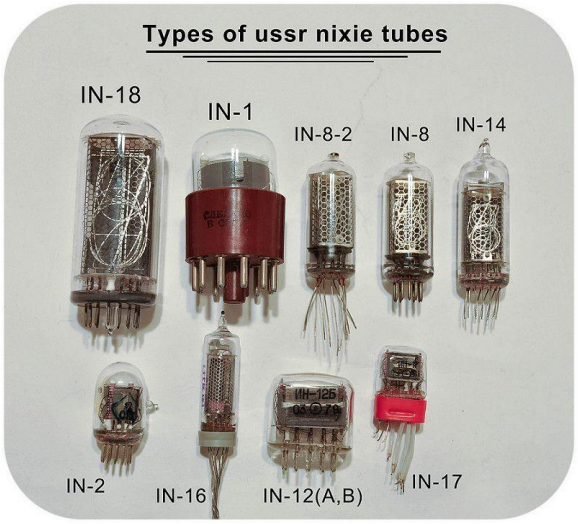
The IR nixie gecko preforms exactly like the original Gecko in that it shows the time, date, year and temperature. It also looks like the gecko which involves designing a 2D acrylic plate and using four nixie tubes to replicate the four digits of the 7 segment display. A 5V USB cable powers the clock just like the gecko and a real time clock (RTC) module as well as a temperature sensor will record data of the time. As an added challenge and better accessibility, an IR remote sensor like the one used in the Wireless IR project will receive commands from a remote to change the clock functions.

Essentially, the circuit surrounds the ATmega328P microcontroller that controls all the important components. These components include an RTC, a temperature sensor, an IR receiver and the four nixie tubes that are all powered through USB inserted into a micro USB female jack. In addition, all parts are through hole so that anyone with a soldering iron can easily put this clock together. This circuit setup is similar the Gecko but when delving into the specifics, there are a few key differences present in my clock. Firstly, a high voltage of up to 170 VDC is needed to power the nixie tubes. Secondly, to control the digits that are on, the individual cathodes on each nixie tube need to be connected to ground to illuminate different numbers so I2C communication used by the seven segment display on the gecko is not an option. Otherwise, much of it is the same.

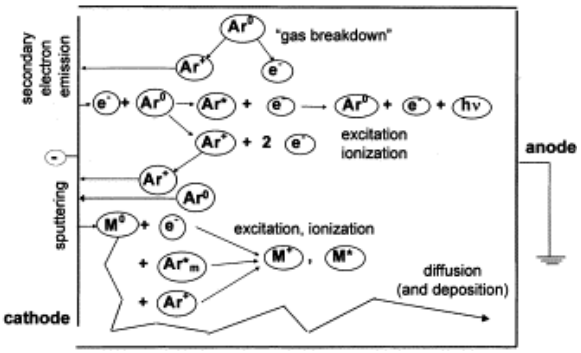
References

https://spectrum.ieee.org/tech-history/dawn-of-electronics/the-nixie-tube-story-the-neon-display-tech-that-engineers-cant-quit
https://www.youtube.com/watch?v=vmNpsofY4-U
https://desmith.net/NMdS/Electronics/NixiePSU.html
https://datasheets.maximintegrated.com/en/ds/MAX1771.pdf
https://desmith.net/NMdS/Electronics/NixiePSU.html
https://electronics.stackexchange.com/questions/331220/anode-driver-for-nixie-clock
https://www.jameco.com/Jameco/workshop/Howitworks/what-is-an-optocoupler-and-how-it-works.html
https://www.reddit.com/r/AskElectronics/comments/4awg9d/nixie_tube_display_ghosting_issue/
https://datasheets.maximintegrated.com/en/ds/DS1307.pdf
https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
https://www.hackster.io/mrelia100/particle-photon-nixie-clock-c8389b

Procedure



The main feature of the project are the nixie tubes. They were invented in the 1920's but only gained traction throughout the mid 1900's. During this period, displaying numbers were especially important at that time because computers were starting to come to life so scientists and engineers needed a way to display output signals. So thereafter, these tubes started appearing in many devices and instances such as nuclear power plants, Wall Street stock prices, and even played a role in displaying data for NASA's moon landing. This technology started to die off in the late 1900's as LED's took over many displays as they were cheaper to manufacture and consumed less power and in the 1990's the last nixie tubes were produced by the Soviet Union. However, nixie tubes have made a comeback and small scale companies have started manufacturing again because of the demand from hobbyists and fans of retro technology. For now, most nixie tubes that people buy come from the unused stocks in Russia particularly from the IN series. Below is a picture of the different IN tubes with the IN-12 and IN-14 nixies being the cheapest and most popular.



Nixie tubes work through a phenomenon known as *gas discharge*. This is when electrically charged particles like electrons move through a gas at high speeds. These particles collide with gas molecules, ionizing them and elevating them to a higher energy level which. Excess energy is emitted as photons or light which is how nixie tubes display their digits, and the color of the light is dependent on the type of gas used. Most nixie tubes contain mainly neon which gives it their reddish-orange glow.

To produce a digit glow, electrons are passed through a cathode wire shaped in the form of a character to ionize the gas around it and illuminate it. In the most common nixie tubes, the metal wires are shaped in the numbers 0 to 9 and are stacked on top of each other. In the case of most of the IN series, a common anode is used and different cathodes control which digit passes electrons through and lights up. A problem exists called cathode poisoning where one single digit is left on for too long; when this happens, a buildup of material is deposited onto adjacent digits so when they next light up, some light is blocked and can dim digits. This is most notable in the picture to the right. Luckily, there is a way solve this issue which is by cycling through all the digits once in a while to keep them active not only preventing cathode poisoning but adding flair as well.

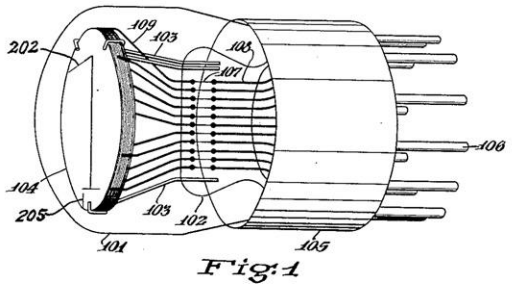


Fig: 1

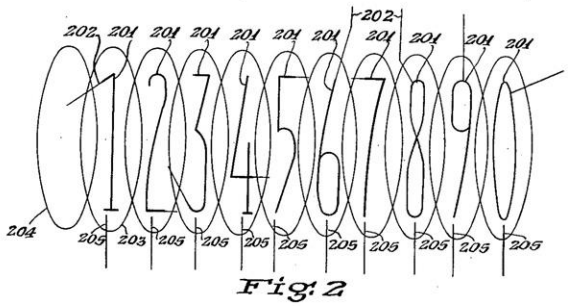
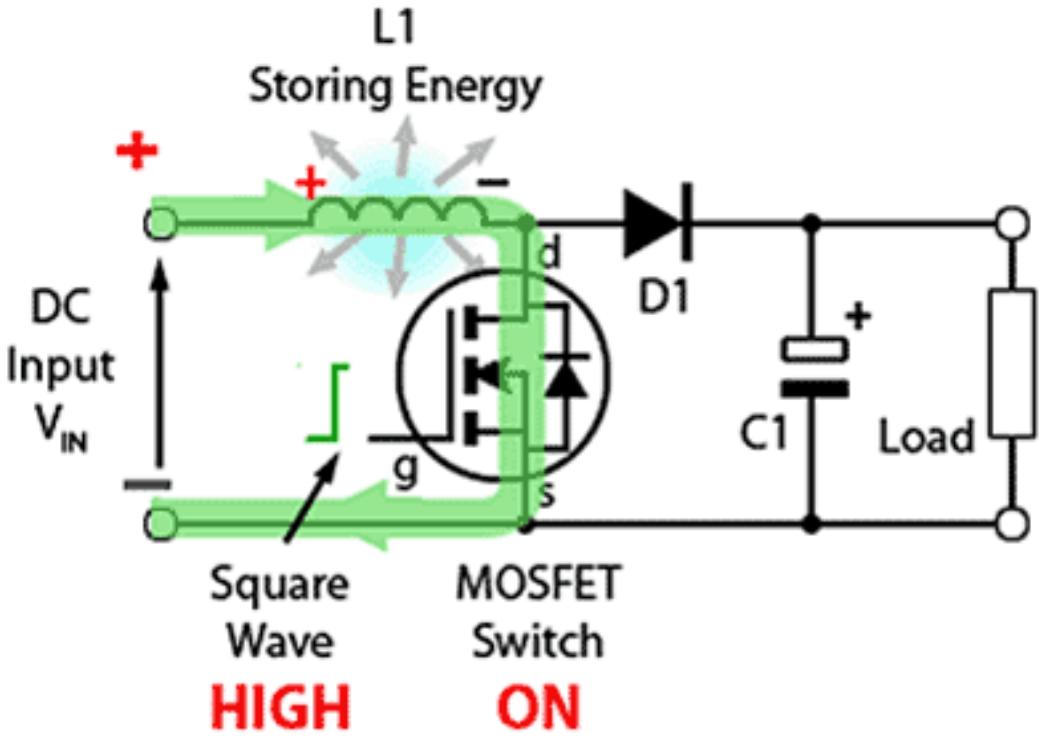


Fig: 2

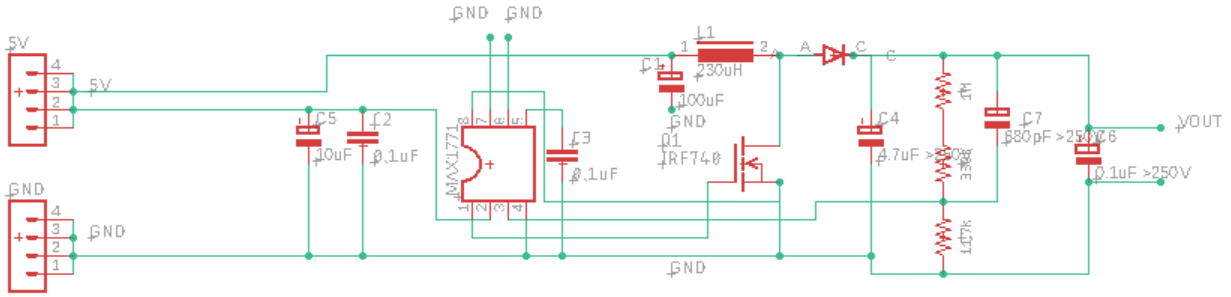
To ionize the gas properly, a nixie tube need much more than five, nine, or twelve volts. In fact, the first product that used gas discharge dubbed Geissler tubes named after the inventor Heinrich Geissler, used a few thousand volts to ionize the gas! His invention eventually lead up to the nixie tube which now only requires 170 volts to ignite which is still very high and potentially lethal. However, nixie tubes only need a small amount of current; around 2.5 mA so a current limiting resistor must always be added to not damage or shorten the lifetime of the nixie. Now to actually achieve such voltages, there are a few ways such as using transformers or boost converters also known as DC-DC step up converters which step up the common low DC voltages to higher numbers. In this project, a boost converter was built and used.

Boost converters work on the premise of exchanging low voltage and high current for high voltage and low current. So, no extra energy is being created as the wattage would equate to about the same, through this is always not the case because of power loss through heat and other factors. Therefore, all boost converters have an efficiency rating. Those with an efficiently rating of over 85% are considered efficient and only loose under 15% of input power to heat. The circuit shown below is the most basic and essential form of a boost converter.

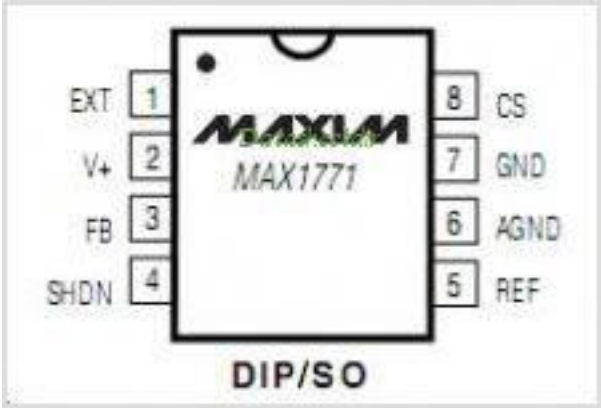


Here a new part is introduced, which is an inductor. Inductors are simply wires wrapped around to form a coil with an optional ferric material inside. This part works on the fact that when energy is passed through a wire, a small magnetic field is generated. By coiling up wires, this magnetic field is magnified and when voltage changes in the inductor, the built up magnetic field collapses and induces current in the coil, creating a large voltage spike which is how the voltage "steps up". In the circuit above, the MOSFET, a type of transistor, switches on and off through a PWM signal. When the signal is high, current runs through the inductor and its magnetic field grows to its maximum strength which is its saturation state. Here, current slows down as the inductor reaches saturation. Once the signal goes low, the MOSFET turns off, cutting off the previous route. The magnetic field collapses inducing extra current into the capacitor which adds on voltage. The diode then prevents the capacitor from discharging immediately. With this, voltage can be controlled through the PWM frequency by controlling how much output voltage there is. However, if the current in the load changes, the voltage will not remain stable if the PWM frequency is kept the same.

To solve this issue, people made and use dedicated ICs to maintaining voltage in a boost converter. Once such IC is called the MAX1771 made by the chip manufacturer Maximum Integrated that was used in the boost converter made using through hole parts. The schematic for this is shown below that echoes the basic boost converter circuit with a few extra parts such as capacitors required for voltage stability as well as the chip itself.



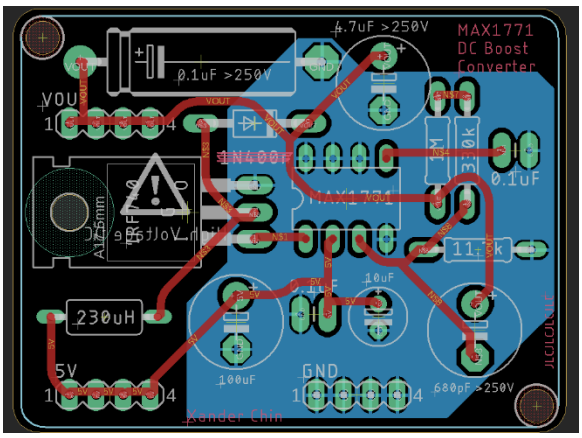
The MAX1771 is an eight pin IC that produces a PWM signal on a MOSFET to keep a pre-set voltage constant with changing current loads by taking feedback on the feedback (FB) pin. The voltage can be set to 12V by grounding the FB pin and can also be set to any output voltage is by connecting two external resistors in a voltage divider formation. The resistance of each resistor tells the MAX1771 what output voltage to produce using the formula below where Vref is 1.5:



$$R2 = (R1)\left(\frac{V_{out}}{V_{ref}} - 1\right)$$

To get a voltage of around 170, a 1 MΩ and 330 kΩ resistor was combined for R1 and a 11.7kΩ resistor was set as R2. When plugging these values in, the result is a voltage of 175. The other pins play roles in power, outputting the PWM signal and enabling the chip. However, the CS pin acts as an output current control that requires a current-sense resistor of below 0.1Ω. Higher resistance equates to less current and through testing I found that there was a drop in voltage too. These current-sense must be able to handle high wattage as large amounts of current and voltage a passing through them. This makes quite large in THT packages, though I managed to find some that were quite small but were only available in SMD packages. Therefore, I skipped out on the resistor which didn't seem to have much of an effect on performance, though if I make a surface mount version of a power supply as most commercial ones do, I will include that resistor.

Since boost converter circuits have a power efficiency rating of how little power is lost to heat when boosting voltage, the component selection for the MAX1771 matters greatly. If component selection is poor, more power will be lost. For example if the MOSFET or diode cannot switch fast enough, there will be some voltage loss. If the inductor is not large enough or has too much resistance, the boost converter will have trouble maintaining the specified voltage. Obviously, anything that has high voltage passing through should be rated for a high power rating so many of the components can handle up to 400VDC Likewise, circuit configuration plays a crucial role as well in efficiency. Essentially, everything should be connected as short as possible to prevent any small amount of resistance that can affect the IC and all ground signals should be connected together in a star ground configuration to reduce signal noise.

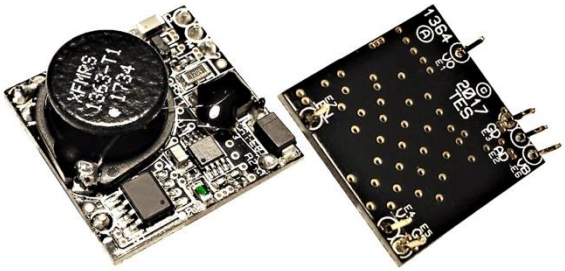


After breadboarding the MAX1771 step-up converter, which worked but had some trouble maintaining voltage (it would drop from 170V to 150V), a PCB was designed. Unlike regular PCBs, design of traces and component placement was crucial to how effective the boost converter works. Wider traces and curved corners where implemented to reduce resistance and enable better current flow and high current traces were kept away from signal traces to prevent any noise. Also, a ground plane was added to effectively create a star ground configuration. Component placement included placing the inductor off the ground plane and off to the side so as to not induce any unwanted current on communication traces and parts and the gate of the MOSFET was placed very close to the EXT pin of the MAX1771 where the PWM signal is outputted.

When powering a nixie tube with it, the circuit on the PCB immediately shot up to 170 volts and maintained its voltage when powering the nixie tube. This was a good sign as it was suitable for powering the clock. To explore efficiency when powering the nixies, the input and output current and voltage was measured and put into the following formula:

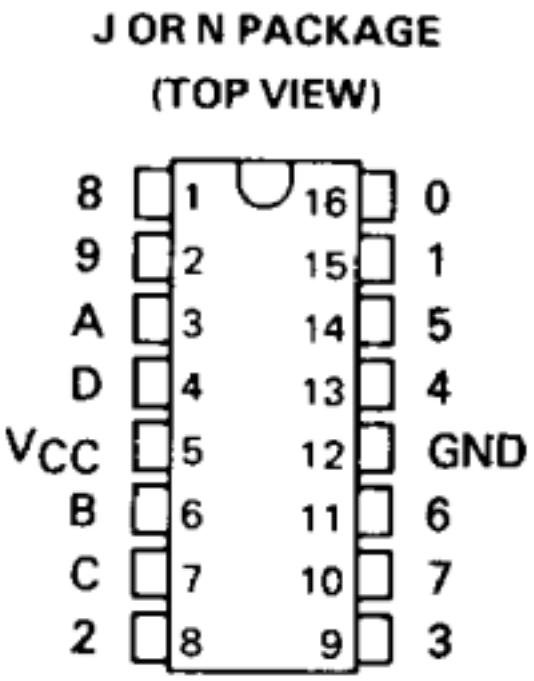
$$\frac{V_{out} \bullet A_{out}}{V_{in} \bullet A_{in}} \bullet 100 = efficiency\ \%$$

$$\frac{170V \bullet 2.03mA}{5V \bullet 80mA} \bullet 100 = 86.275\%$$



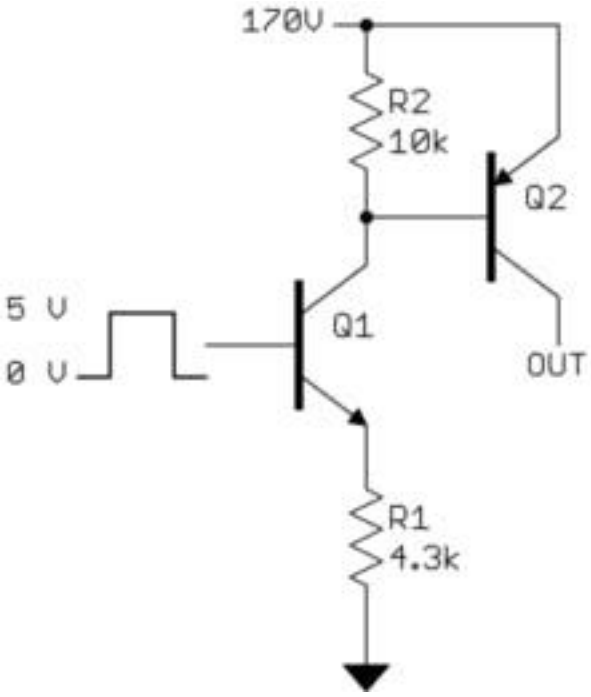
Overall, when using the boost converter in my clock, it works well and is viable to use. Although it still has a bit of trouble maintaining voltage and current, as shown when some of the digits fade a bit when they flash in edit mode, it gets the job done. To improve on it, I would create one that is not entirely through hole soldering and instead integrate some SMD components to reduce PCB space and trace widths and therefore increase performance, similar to the Taylor 1364 nixie power supply pictured below and that I bought just in case the one I built did not function properly. Compared to mine, it is much smaller and also maintains voltage perfectly.

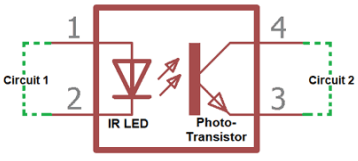
Now that the nixie tubes have power, the different digits need to be controlled to show time, date, temperature, and other important information. To control which digit is on, the digits cathode has to be pulled to ground as each nixie tube has a common anode and separate cathodes. Luckily since these tubes were widely used, a dedicated IC was made to sink high voltages from the nixie tubes. This IC is the Russian made K155ID1. It takes in a four-bit binary input and allows current to pass through the equivalent decimal number assigned to a pin. So by connecting the digit cathodes of the nixie tube to the respective digit grounds on the IC, different digits can be illuminated.



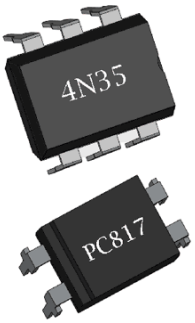
Since one K155ID1 controls one nixie tube, four is needed to make a four digit clock. However, these chips only come in a DIP package which take up a large amount of space. Because only through hole parts were allows to make soldering easy, POV was implemented. This means that only one K155ID1 was used as only one digit will be on at a given time. Also, only one anode of a tube must be on as well; if other anodes of different tubes are on, they would all display the same digits so each tubes anode must have the ability to turn on and off. By doing this, the first tube would turn on and the K155ID1 would sink the digit cathode to illuminate it. Then the first anode would turn off and the next tube anode repeats this cycle.

To turn off and on high voltage anodes, an high power NPN transistor must be used where it can be controlled by a microcontroller, but the catch is that one transistor for each anode is not enough. If only one is used, it will switch on but the voltage at the emitter will be smaller than the voltage applied to the base because it is in an "emitter follower" configuration. Therefore, if controlling the transistor at the base with a microcontroller, the max output voltage produced will be less than 5V no matter the voltage at the collector, too little to turn on a nixie. To solve this, a configuration like the one on the right involving an NPN and PNP is used which prevents an emitter follower configuration.



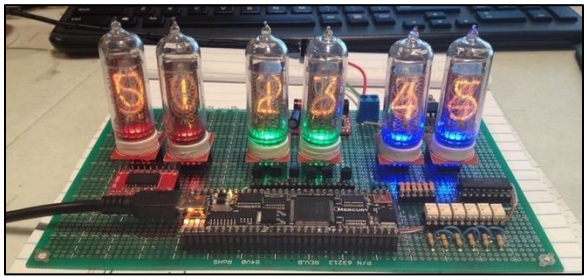


Opto-coupler



When employing this configuration, the anode could not be switched off. This led me to believe that the leakage current of the transistors I bought was too great and was keeping the PNP transistor constantly saturated. These transistors were different from the recommended MPSA42 NPN and MPSA92 PNP as I could not find any through hole version in stock. As a workaround, I discovered optocouplers, which are similar to transistors. They simply consist of an LED and a photosensor inside a DIP package with four pins; two for the LED and two for the photoresistor. When the LED turns on by connecting power and ground to the respective, the photoresistor conducts and allow current to pass through both output pins, connecting the circuit. Luckily, some high voltage optocouplers were available so I setup four of them for each digit. This setup was also a lot neater and worked very well compared to the transistor layout.

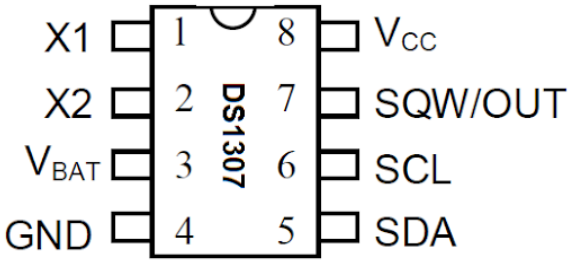
To recap on controlling the clock, four inputs need to control each digit cathode and an extra four inputs are required to control the optocouplers for each digit. This equates to eight total inputs which is the perfect amount to be driven by a 595 shift register; the four low bytes control the cathodes through the K155ID1 and the four high bytes control the anodes through the optocouplers. This saves I/O pins for the ATmega328P as only three need to control the shift register. This presents a small problem of speed as the software shift out function can require some time especially since it needs to be called every millisecond or so to enable the POV effect. But, using dedicated SPI pins on the ATmega328P, a strategy used in the Wireless project, the result was a much faster shift out time.

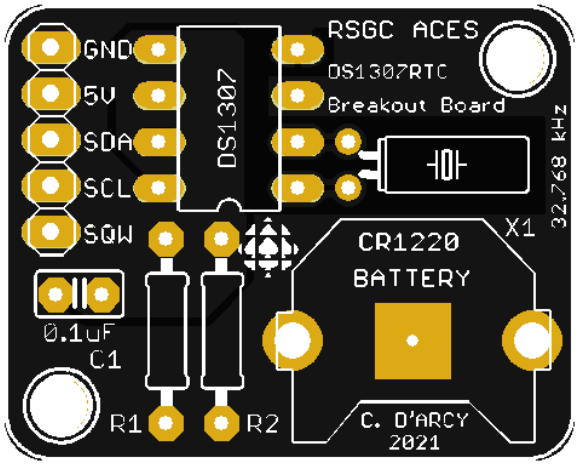


In the first persistence of vision project, the strategy of shifting out bits to form alphanumerical patterns applies to the nixie clock as well. Unlike the shift registers in POV project, the shift register on the clock must be given specific instructions to prevent a common issue called ghosting. This is a common issue in multiplexing nixie tubes when an illuminated digit on one tube is dimly lit in the other tubes. To solve this, a nixie tubes anode must be turned off before proceeding to turn the next one on.

Now that the tubes can be powered and individual digits can be illuminated, all that's left is getting data of the time and temperature. For time, another dedicated IC called the DS1307RTC records time with the help of some other external components and for the temperature, a DHT11 sensor records not only temperature but humidity as well.

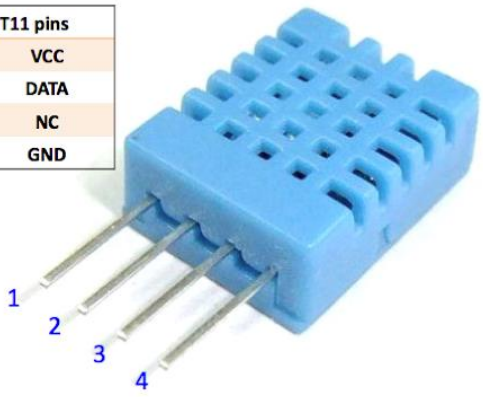
Let's start with the DS1307. This eight pin IC sends data back and forth through I2C and keeps track of time even when the clock is not powered through the help of a 3V lithium coin battery. For better accuracy, an optional 32.768 kHz crystal can be attached to the X1 and X2 pins. Luckily, my teacher C. D'Arcy designed a breakout board for the DS1307 for easy setup so that was used. This breakout board includes some supporting resistors and capacitors to facilitate smoother power and data connections



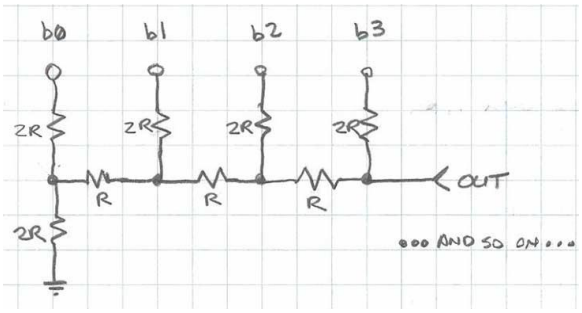


The DHT11 is a basic weather sensor containing a temperature and humidity sensor inside calibrated to measure between 20-90% humidity and 0-50 °C with a medium amount of accuracy. To communicate with an MCU, the device uses serial interface which involves one wire and two-way travel where the MCU sends a stream of bits to the DHT11 to start up and the DHT11 sends another stream of bits to the MCU reporting the temperature and humidity. When recording data, it takes approximately one second to complete. More information can be found on the datasheet in the references.

DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND

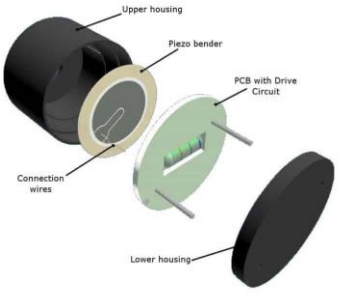


For better controllability, a sharp GP1UX511QS sensor takes in IR signals from a custom remote. This remote consists of several buttons that all connect to one analog pin through an 8-bit digital to analog (DAC) R2R ladder. This is similar to a resistor array except that the resistors are arranged in a different configuration shown below. It converts a digital binary input from highs or lows on the input pins and converts them into an analog voltage.



Here, power is connected to the power pin and buttons in a pull-up resistor configuration connect to the input pins of the R2R ladder. The output pin emits a varying voltage depending on which button is pressed. This output voltage can then be calculated as a binary percentage of the input voltage. For example, if 5 volts is placed on the input pin and the most significant pin of the R2R ladder receives a high from pressing the button, the output voltage will be 5 over 128, the decimal value of the most significant bit, and 255, the binary to decimal value when all 8-bits are high. This voltage will then be around 2.5V. Since the Arduino NANO, the microcontroller that is used to read the R2R ladder, has a 10-bit analog to digital (ADC) converter, 5 volts will equate to 1023 and 2.5 volts will either be 512 or 511. By using an if else ladder of ADC values from reading the output, the press of different buttons can be sensed only using one pin and different IR signals can be sent out, which manipulates the clock in different ways. This is achieved through connecting an IR LED to a PWM pin and a current limiting resistor. The IR LED sends out a signal of a certain remote protocol and it is read by the GP1UX511QS on the clock.

These are all the parts and implementations that are included for the clock. To recap, a DHT temperature sensor and an RTC breakout board send data to the ATmega328P. The MCU then controls a 595 shift register through SPI which then controls the K155ID1 and four optocouplers to show the collected data with persistence of vision. The optocouplers help turn on and off each anode of the tubes and the K155ID1 sinks current of a specific digit cathode on the tubes. The tubes receive high voltage power through a boost converter. Buttons and switches connected to the spare pins of the ATmega328 act as inputs for someone to control the clock.

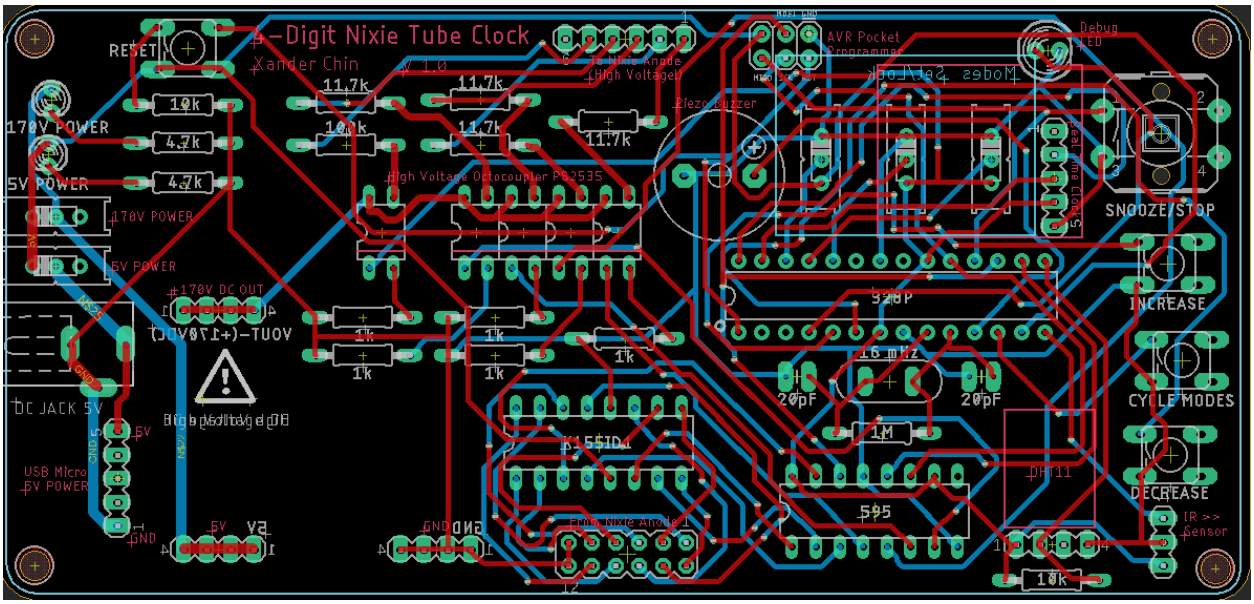


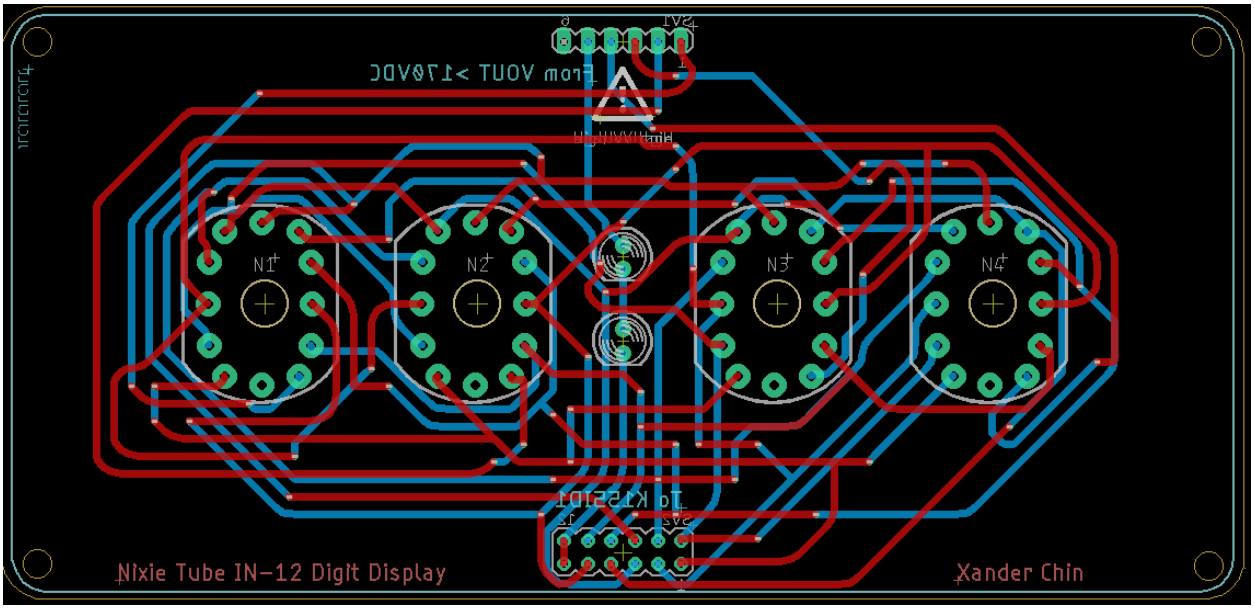
Piezo Buzzer Construction

Also, a piezo buzzer is attached to act as an alarm. It works by using a piezo crystal that changes shape when a voltage is applied which then pushes against a diaphragm in the buzzer and produces a pressure wave that is picked up as sound. In case the clock wants to be changed from far away, a custom remote with an IR LED with buttons and a R2R ladder to control what signal is sent controls the clock. These IR signals are picked up the sharp GP1UX511QS attached to a pin on the MCU. As an added feature, two small neon bulbs that also require 170 volts were added to form the colon that separates the hours from the minutes. A 100 kΩ resistor was attached to them to lower brightness and to prevent all the maximum current draw of the power supply from going through the bulbs, dimming the illuminated digits.

Once prototyped, PCB designs were fabricated for the clock and power supply. Unfortunately, there was not enough time to produce a PCB for the remote so it remained on the breadboard. Unlike regular PCBs, these were dealing with high voltage and had to have certain requirements to prevent degradation and to protect the whole circuit. Firstly, PCB traces that handled power were increased in thickness for the amount of current consumed by the power supply and more importantly, clearance between pads, vias and traces was extended to prevent any arcing and short circuiting. As an extra precaution, high voltage and ground traces were kept far away from each other and most corners were rounded for better current flow. Secondly, parts were placed so that the distance between traces were minimal and the overlap of high voltage to signal traces were minimized to prevent any induced current in the data lines.

Once sorting through the precautions, looks and ergonomics came into play. I decided that the clock should have a display PCB where the nixie tubes were soldered on and it would go on top of the base PCB where most of the parts would reside. This was accomplished through male and female header pins so that the top could be easily taken off. The top contained the anodes of the nixie tubes and colon bulbs and the bottom bunched the cathodes to the K155ID1on the base PCB. Speaking of the base PCB, buttons were placed on the side for better accessibility and switches were mounted on the back. The boost converter PCB would be separate from the base PCB just in case something went wrong with the boost converter. To top it all off, a 2D acrylic plate with holes for M3 screws and standoffs was made and attached in front of the digit PCB. Essentially, you can imagine the clock like a good looking sandwich of PCBs.



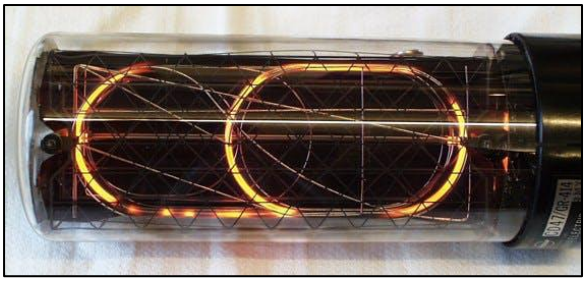


After soldering everything up, all worked well. There were of course a few issues with clock. Firstly, the reset button for the ATmega328 did not work as I reversed the ground and power traces. This was fixed by cutting the respective traces and rerouting them using external wires. Secondly, the cathode inputs of the display to the base PCB were mixed up so binary input of the K155ID1 did not match the illumiated decimal digit equivalent. Luckily, this was an easy fix by remapping the mixed up digits with the correct ones in the code. Some other minor issues included the difficulty of accessing the power switches and buttons, misspelling optocoupler as "octocoupler" on the silkscreening, and not being able to fit the RTC breakout board on the front so it was simply moved to the back. For the boost converter, I planned to use male pins to attach to the base but they were too short and were blocked by the components so a wire was used instead. To fix this in V2, the components will be soldered on the other side. Other than that, everything works perfectly. To view a demonstration and to learn aobut the different functions it has, take a look at the YouTube link in the media section.

Media

The ACES Gecko

The IR Nixie Gecko



Cathode poisoning on a nixie tube

Cycling through the digits to prevent cathode poisoning

YouTube video link: https://youtu.be/nEOyhyJHx2M

Code

Nixie Clock

```
// PROJECT  : The IR Nixie Gecko
// PURPOSE  : Long ISP
// COURSE   : ICS3U
// AUTHOR   : Xander Chin
// DATE     : May 29, 2021
// MCU      : 328P
// STATUS   : Working. Could use some improvements (very messy)
// REFERENCE:

#include <EEPROM.h>   //was going to be used for alarm

#include <IRremote.h>
int RECV_PIN = 8;
IRrecv irrecv(RECV_PIN);
decode_results results;

#include <SPI.h>
#define OENABLE 9

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

#include <NewTone.h>

#include <DHT.h>
#define DHTpin 9
DHT dht(DHTpin, DHT11);

uint32_t code;
uint32_t lastSwitch;
#define pov 5000
bool state = true;
uint8_t digits[4] = {1, 2, 3, 4};

//for mixed up digits
uint8_t trueNumbers[10] = {9, 2, 1, 4, 3, 6, 5, 8, 7, 0};

uint8_t output = 0;
uint8_t x = 0;

uint8_t hours;
uint8_t minutes;
uint8_t months;
uint8_t days;
uint16_t years;
//uint8_t timerMinutes;
//uint16_t timerSeconds;
//uint16_t stopwatch = 0000;    //unimplemented

uint8_t alarmHour = 7;
uint8_t alarmMinute = 0;

bool edit;
bool digitOn;
bool alarm;
bool alarmOn;
bool snooze;
uint32_t lastSnooze;
bool timeMode;
```

```
uint8_t mode = 0;
uint8_t editMode = 0;
uint32_t lastCycle;
#define cycle 15

uint32_t lastPress;
#define debounce 250

uint32_t lastFlash;
uint32_t lastOn;

bool tempMode;
uint16_t t;
uint8_t h;

uint32_t lastTimeCycle;
uint8_t flash;

tmElements_t tm;

#define snoozeDuration 10000

void setup() {
  irrecv.enableIRIn(); // Start the receiver
  pinMode(2, OUTPUT);
  pinMode(3, INPUT_PULLUP);
  pinMode(4, INPUT_PULLUP);
  pinMode(5, OUTPUT);
  pinMode(6, INPUT_PULLUP);
  pinMode(7, OUTPUT);
  pinMode(12, INPUT_PULLUP);

  dht.begin();
  t = dht.readTemperature()*100;
  h = dht.readHumidity();
}

void loop() {
  alarm = digitalRead(A2) ? true : false;
  timeMode = digitalRead(A3) ? true : false;
  if(!alarm) alarmOn = false;
  if(alarm && alarmHour == tm.Hour && alarmMinute == tm.Minute && tm.Second == 0) {
    alarmOn = true;
    snooze = false;
  }
  if(alarm && alarmOn) {
    NewTone(5, 330);
    if(!digitalRead(12)) {
      alarmOn = false;
      snooze = true;
    }
  } else noNewTone();

  if(alarm && snooze) {
    if(millis() - lastSnooze > snoozeDuration) {
      alarmOn = true;
      snooze = false;
      lastSnooze = millis();
    }
  }

  //refresh variables
  hours = tm.Hour;
  minutes = tm.Minute;
  months = tm.Month;
  days = tm.Day;
  years = tm.Year;
```

```
  edit = digitalRead(A1) ? true : false;
  if(edit) {
    switch(mode) {
      case 0: editTime(); break;
      case 1: editDate(); break;
      case 2: editYear(); break;
      case 3: editAlarm(); break;
      //case 4: editTimer(); break;   //unimplemented
      default: break;
    }
  } else {
    flash = B1100;
    switch(mode) {
      case 0: showTime(); break;
      case 1: showDate(); break;
      case 2: showYear(); break;
      case 3: showTemp(); break;
      //case 4: showTimer(); break;   //unimplemented
      default: break;
    }
  }

  //IR codes
  if(irrecv.decode(&results)) {
    code = results.value;
    irrecv.resume();
  }

  if(code == 100) {
    mode++;
    code = 0;
    scramble();
  }

  if(code == 200) {
    mode--;
    code = 0;
    scramble();
  }

  if(!edit) {
    if(!digitalRead(6) && millis() - lastPress > debounce) {
      scramble();
      mode++;
      lastPress = millis();
    }
  }

  //cycle through digits
  if(millis() - lastTimeCycle > 60000) {
    scramble();
    lastTimeCycle = millis();
  }

  //blink colon bulbs
  state = tm.Second % 2 ? 1 : 0;
  digitalWrite(7, state);

  if(!edit) displayNumbers();   //instead flash digits

  //cycles to the first mode after the last
  mode = mode % 4;
}
```

```
//
void displayNumbers() {
  if(micros() - lastSwitch > pov) {
    bitWrite(output, 4+x, LOW);
    hardwareShiftOut(output);
    x = (x+1) % 4;
    output = digits[x];
    hardwareShiftOut(output);
    bitWrite(output, 4+(3-x), HIGH);
    lastSwitch = micros();
  }
  hardwareShiftOut(output);
}

void digitFlash(uint8_t number) {
  if(millis() - lastOn > 500) {
    digitOn = !digitOn;
    lastOn = millis();
  }
  if(micros() - lastFlash > pov) {
    bitWrite(output, 4+x, LOW);
    hardwareShiftOut(output);
    x = (x+1) % 4;
    output = digits[x];
    hardwareShiftOut(output);
    if(digitOn) {
      bitWrite(output, 4+(3-x), HIGH);
    } else {
      if(bitRead(number, 3-x)) bitWrite(output, 4+(3-x), HIGH);
    }
    lastFlash = micros();
  }
  hardwareShiftOut(output);
}

void showTime() {
  if(RTC.read(tm)) {
    if(timeMode) {
      if(tm.Hour == 0 || tm.Hour == 12)
        setDigits(12*100 + tm.Minute);
      else setDigits((tm.Hour % 12)*100 + tm.Minute);
    }
    else setDigits(tm.Hour*100 + tm.Minute);
  }
  if(timeMode) {
    if(tm.Hour > 11) digitalWrite(2, HIGH);
    else digitalWrite(2, LOW);
  } else digitalWrite(2, LOW);
}

void showDate() {
  if(RTC.read(tm)) {
    setDigits(tm.Month*100 + tm.Day);
  }
}

void showYear() {
  if(RTC.read(tm)) {
    setDigits(tmYearToCalendar(tm.Year));
  }
}
```

```
void showTemp() {
  if(RTC.read(tm));   //keeps updating the time
  if(millis() - lastCycle > 3000) {
    tempMode = !tempMode;
    if(!tempMode) scramble();
    t = dht.readTemperature()*100;
    h = dht.readHumidity();
    lastCycle = millis();
  }
  tempMode ? setDigits(h) : setDigits(t);
}

void showTimer() {
  //unimplemented
}

void showAlarm() {
  setDigits(alarmHour*100 + alarmMinute);
}

void editTime() {
  //configure time
  if(!digitalRead(3) && millis() - lastPress > debounce) {
    if(flash == B1100) {
      hours = (hours + 1) % 24;
      tm.Hour = hours;
      RTC.write(tm);
    } else if(flash == B0011) {
      minutes = (minutes + 1) % 60;
      tm.Minute = minutes;
      RTC.write(tm);
    }
    lastPress = millis();
  } else if(!digitalRead(4) && millis() - lastPress > debounce) {
    if(flash == B1100) {
      hours == 0 ? hours = 23 : hours--;
      tm.Hour = hours;
      RTC.write(tm);
    } else if(flash == B0011) {
      minutes == 0 ? minutes = 59 : minutes--;
      tm.Minute = minutes;
      RTC.write(tm);
    }
    lastPress = millis();
  }
  showTime();
  flashNumbers();
}
```

```
void editDate() {
  //configure date
  if(!digitalRead(3) && millis() - lastPress > debounce) {
    if(flash == B1100) {
      months = (months + 1) % 13;
      tm.Month = months;
      RTC.write(tm);
    } else if(flash == B0011) {
      days = (days + 1) % 60;
      tm.Day = days;
      RTC.write(tm);
    }
    lastPress = millis();
  } else if(!digitalRead(4) && millis() - lastPress > debounce) {
    if(flash == B1100) {
      months == 0 ? months = 23 : months--;
      tm.Month = months;
      RTC.write(tm);
    } else if(flash == B0011) {
      days == 0 ? days = 59 : days--;
      tm.Day = days;
      RTC.write(tm);
    }
    lastPress = millis();
  }
  showDate();
  flashNumbers();
}

void editYear() {
  if(!digitalRead(3) && millis() - lastPress > debounce) {
    years = (years + 1) % 10000;
    tm.Year = years;
    RTC.write(tm);
    lastPress = millis();
  } else if(!digitalRead(4) && millis() - lastPress > debounce) {
    years == 0 ? years = 9999 : years--;
    tm.Year = years;
    RTC.write(tm);
    lastPress = millis();
  }
  showYear();
  digitFlash(0);
  if(!digitalRead(6) && millis() - lastPress > debounce) {
    mode++;
  } else if(!digitalRead(12) && millis() - lastPress > debounce) {
    mode--;
  }
}

void editAlarm() {
  if(!digitalRead(3) && millis() - lastPress > debounce) {
    if(flash == B1100) {
      alarmHour = (alarmHour + 1) % 24;
    } else if(flash == B0011) {
      alarmMinute = (alarmMinute + 1) % 60;
    }
    lastPress = millis();
  } else if(!digitalRead(4) && millis() - lastPress > debounce) {
    if(flash == B1100) {
      alarmHour == 0 ? alarmHour = 23 : alarmHour--;
    } else if(flash == B0011) {
      alarmMinute == 0 ? alarmMinute = 59 : alarmMinute--;
    }
    lastPress = millis();
  }
  showAlarm();
  flashNumbers();
}
```

```
void editTimer() {
  //not implemented yet
}

void hardwareShiftOut(uint8_t value) {
  //Initializes the SPI bus setting SCK, MOSI, and SS to outputs,
  SPI.begin();                //pulls SCK and MOSI low, and SS high. Default: MSBFIRST
  SPI.transfer(value);        //invert the output for ease of interpretation
  digitalWrite(SS, LOW);      //pull Slave Select LOW to identify target device
  digitalWrite(SS, HIGH);     //release target device
  SPI.end();                  //disables SPI Bus (leaving pin modes unchanged)
  digitalWrite(OENABLE, state);
}

void scramble() {   //iterates through all digits
  for(uint8_t x = 0; x < 30; x++) {
    for(uint8_t y = 0; y < 4; y++) {
      digits[y] = (digits[y] + 1) % 10;
    }
    lastCycle = millis();
    while(millis() - lastCycle < cycle) {
      displayNumbers();
    }
  }
}

void setDigits(uint16_t number) {   //set a 4-digit number to display
  uint8_t n = 0;
  for(uint8_t x = 0; x < sizeof(digits); x++) {
    n = number % 10;
    number /= 10;
    digits[x] = trueNumbers[n];
  }
}

void flashNumbers() {   //flash/blink specific digits
  if(!digitalRead(6) && millis() - lastPress > debounce) {
    flash >>= 2;
    if(flash == 0) {
      mode++;
      flash = B1100;
    }
    lastPress = millis();
  } else if(!digitalRead(12) && millis() - lastPress > debounce) {
    flash <<= 2;
    if(flash > 16) {
      mode--;
      flash = B0011;
    }
    lastPress = millis();
  }
  digitFlash(flash);
}
```

IR Remote

```
// PROJECT  : IR remote for IR Nixie Gecko
// PURPOSE  : IR control
// COURSE   : ICS3U
// AUTHOR   : Xander Chin
// DATE     : May 29, 2021
// MCU      : 328P
// STATUS   : Working
// REFERENCE:

#include <IRremote.h>
IRsend irsend;
#define READ A5

void setup() {
  Serial.begin(9600);
}

void loop() {
  uint16_t button = analogRead(READ);
  Serial.println(button);
  if(button == 0) {
    //nothing
  } else if(button < 100) {
    irsend.sendSony(100, 12);
  } else if(button < 200) {
    irsend.sendSony(200, 12);
  } else if(button < 300) {
    //unimplemented
  } else if(button < 500) {
    //unimplemented
  } else if(button < 600) {
    //unimplemented
  }
}
```

Reflection

Here, at the end of another long year, I felt like I've accomplished so much in every engineering domain of hardware, software, design and communication. It has truly been a pleasure even with COVID-19 distance learning and I am extremely proud of what I accomplished this year. Unfortunately, like my other ISP, this one is late as I spent most of my time debugging and completing other assessments from other classes. This also had an effect on the performance of the code; I wanted to implement port manipulation techniques or at least clean it up a little but I simply ran out of time. But, even though this submission is late, I am still really proud making a fully functional clock powered by a DIY power supply that worked with a few tweaks on the first version and I plan on using it as a bedside/work clock. I also plan on continuing to add more features and improve on it further for it to be on par with commercial ones as I was running low on time. Before going to bed, I want to thank former ACE Mariano Elia for providing me the HV Taylor 1364 power supply, extra K155ID1s and his tips and tricks that he learned over the years while developing his version of a nixie clock.

ICS4U

AVR Optimization

Project 3.1: PB Machine

Purpose

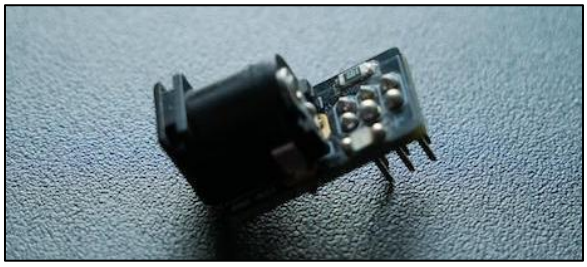
The purpose of this project is to practice surface mount technology (SMT) soldering skills and to refresh the procedures of submitting a project by assembling a PB machine and using it to power a circuit.

References

https://youtu.be/d58GzhXKKG8

Procedure

SMT refers to components that are very small in size that allows hobbyists and engineers to fit more components into a smaller PCB. This helps transition a bulky device to a sleeker more compact device such as an iPhone. The PB machine, although very simple, prepares oneself to solder SMT components.

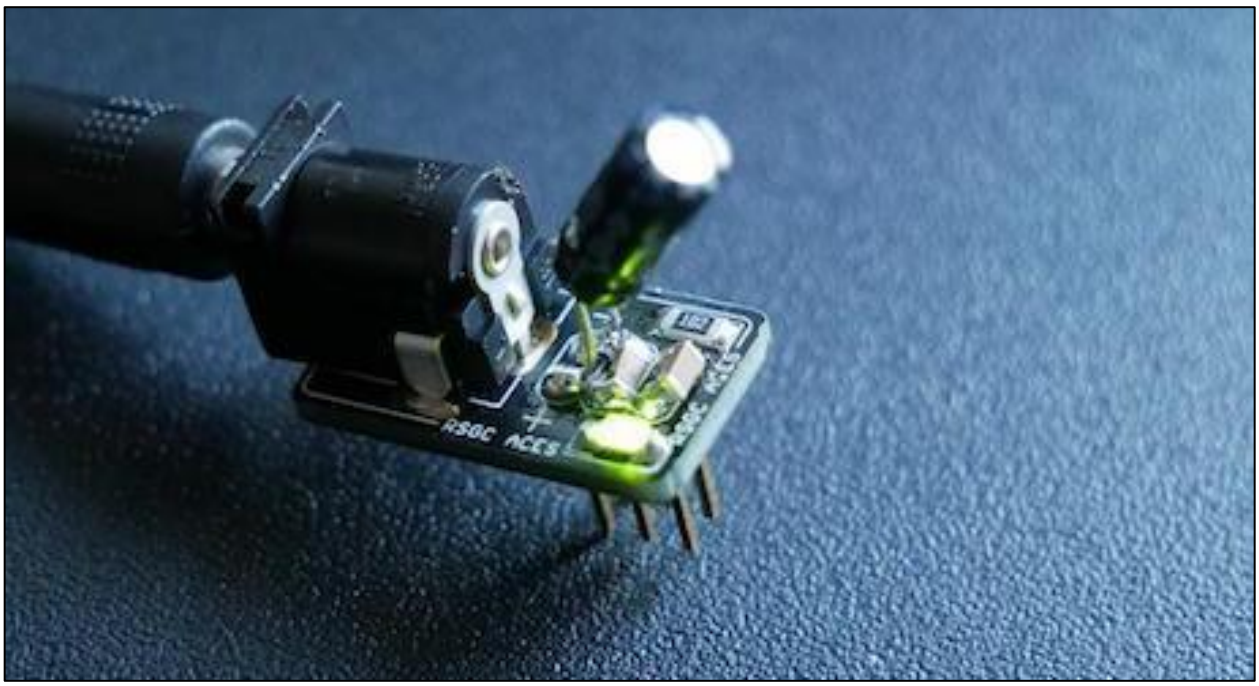


The PB machine is a handy device used to supply power to a breadboard with a DC barrel jack input via male header pins. It also includes a small surface mount device (SMD) LED and resistor to indicate whether power is being supplied. This turns the LED on whereas no power or a short circuit will turn off the LED which makes it a very useful addition. The PB machine has been the most extensively used device throughout this course as it was always used in the ICS2O section. This time, the SMD LED and resistor did not come presoldered and instead, the task was to solder on these components using a variety of SMT soldering techniques.

To solder on the SMD LED, a normal soldering iron was used. The technique for this is to put solder on one pad, place the LED on the pads, reheat the solder so that the LED wicks to it, and finally solder it to the remaining pad. This technique can also be used for resistors and capacitors, however, SMT IC's are a little harder to do.

To solder on the resistor, I opted for a hot air gun and soldering paste to get familiar with this second technique. First, a small amount of solder paste was added to both pads and the resistor was placed on the pads. The hot air gun was then set to 300 °C with a small amount of airflow. To solder the resistor, the hot air gun should be above the resistor and then slowly decend downward. After a while, the solder paste will turn into melted solder and the resistor should wick into place on the pads. It is important to put a small amount of solder paste or else the solder will bridge between the pads. This technique is more precise and easier to do than manually using a regular soldering iron, however, to solder IC's and components with small pads on a PCB, a stencil must be used to correctly apply solder paste to the small pad footprints.

To practice even more, three capacitors of varying capacitance were added on which helps filter out sudden voltage spikes from the power supplied in the PB machine. Two of the capacitors were SMD while the third was a normal polarized capacitor where all were soldered between the supply and ground rails of the 2 × 3 male pin header.



The circuit that the PB machine powers converts a square wave produced by a 555 timer into a sine wave using resistor capacitor pairs. Three of these RC pairs are needed to successfully produce a decent looking sine wave. The first pair turns the square wave into one that shows the charging and discharging of a capacitor over time as shown in Project 1.2 The Capacitor Visualizer. The second RC pair turns that wave into a triangle wave and the final RC pair converts it into a sine wave. It is also noted that these triangle and sine waves show up with a gap which is currently unknown why they are there. Since resistors and capacitors are fixed in their values, certain pairs of them can only handle a range of frequencies. For example, a 15 kΩ resistor paired with a 10 μF capacitor can convert a signal of around 1 Hz. This frequency and the RC values were found using this formula where f is the frequency, R is the resistance in Ω, and C is the capacitance in farads:

Parts Table	
Quantity	Description
1	PB machine
1	9V power source
1	1206 SMT LED
1	1kΩ 1206 SMT resistor
1	0.1 μF SMT capacitor
1	1 μF SMT capacitor
4	10 μF capacitor
1	555 timer IC
1	5mm red LED
1	100 μF capacitor
1	10 kΩ potentiometer
1	100 Ω fixed resistor
1	2.2 kΩ fixed resistor
1	470 Ω fixed resistor
1	330 Ω fixed resistor
3	10 kΩ fixed resistor
1	4.7 kΩ fixed resistor
1	Oscilloscope + probes

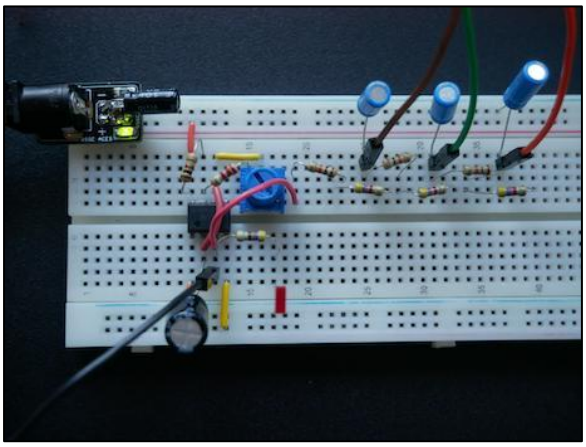
$$f = \frac{1}{2\pi RC}$$

Also, as the wave is filtered through RC pairs, the amplitude decreases because the capacitor does not have enough time to charge all the way to its full voltage and therefore must discharge when the square wave is low. So, as the frequency increases the magnitude decreases and vice versa.

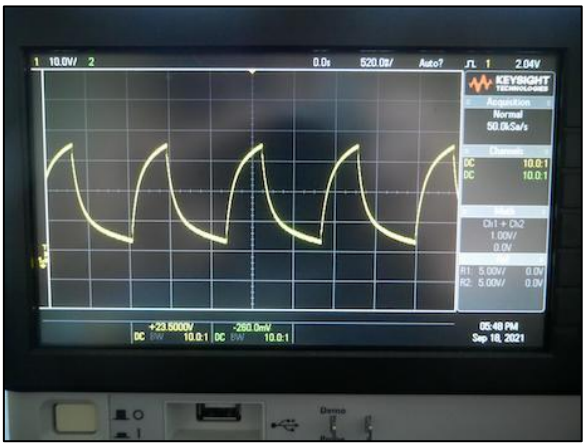
To capture the waveforms, a handy device was used called an oscilloscope. This device graphs voltage, current, and other values over time, hence forth, it is very useful in debugging and understanding what goes on in a circuit. Most oscilloscopes are quite expensive and their price increases depending on three main factors. First is the amount of channels an oscillioscope has; the more channels the more expensive it will be. The number of channels available means that the oscillopscope can measure up to that number of locations. Second is the sample rate which describes how many measurements the scope can take in a second, therefore limiting the frequency of a signal that it can measure. The standard sample rate values are usually 1 or 2 gigasample per second (GSa/s) which means they can measure most MHz signals with good resolution. The third factor is the bandwidth which further limits the frequency since when you get close to the bandwidth frequency, the signal output starts to become altered. Most good scopes have bandwidths within the MHz range.

For a basic use of the scope, one must connect a probe to a channel, connect the ground to the circuit ground, and finally the acutal probe to a position in the citcuit to take a look at the waveforms. Afterward, the waveform must be positioned and shown properly through corresponding buttons. More advanced uses include math functions for calculating or predicting values of a circuit.

Media



The custom circuit powered by the PB machine



The waveform passed through one RC pair

The waveform passed through two RC pairs

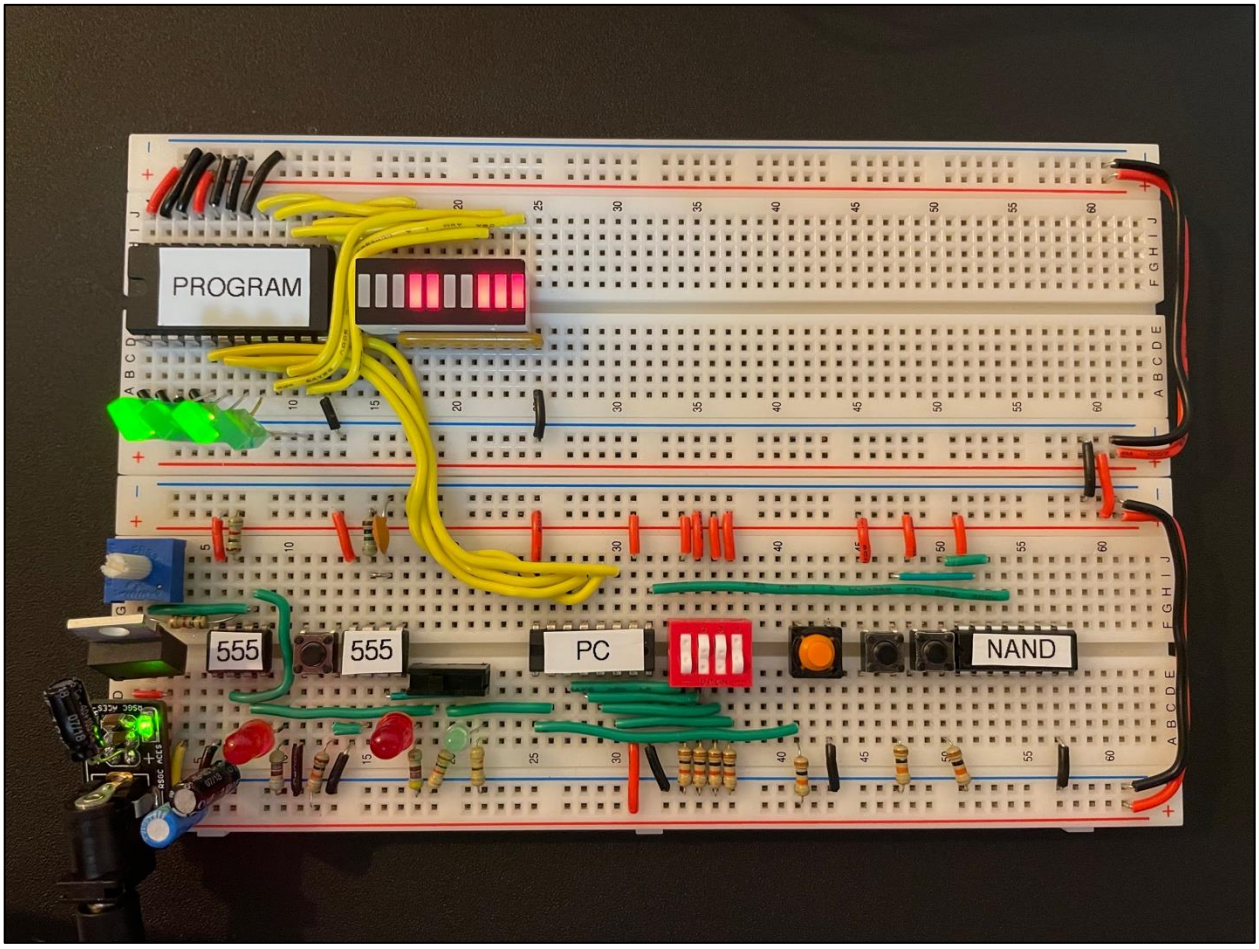
The waveform passed through three RC pairs

YouTube video link: https://youtu.be/GJCeeS7HzAk

Reflection

Even though this project was a simple one, I still felt a little stressed as it is the first project of the year. However, it was simple enough to allow me to focus on getting back into the procedure of writing, filming, and brushing up my DER for this year as well as having the time to add extra capacitors the PB machine. I enjoyed exploring the soldering options for SMT components and will defninely be using more of those as their size saves a lot of space on a PCB and an enclosure. Also getting to explore the functions of the oscilloscope was fun since they are essential to debugging complex circuits in the future. All in all, it's good to be back and I'm really excited to explore and complete other projects that lie ahead.

Project 3.2: CHUMP Code, Clock, Counter



Purpose

The purpose of this project is to lay a foundation for the Cheap Homebrew Understandable Minimal Processor (CHUMP) 4-bit computer built on a breadboard by designing its clock and program counter functions as well as creating a unique code to run on it. This allows the rest of the parts to be added on easily and tested.

Theory

The CHUMP computer is a simple as possible and easy to understand 5 volt logic 4-bit computer created by Dave Feinberg to expose his AP software students to the basic hardware and software of a computer. A 4-bit design means that it can only hold 16 line numbers that the CHUMP can use in its processing of data values, addresses or line numbers, and output, making the build very restrictive in computing power but easier to comprehend and assemble. Even so, the CHUMP is still a complex circuit and is divided up into sub circuits. These subcircuits include the CHUMP clock, the program counter, the program and control EEPROM, the multiplexer, the ALU, the accumulator, the RAM, and the address. Power must be 5 volts as most of the IC's used are TTL chips that require 5 volts specifically. Color coded wires help with identifying the different functions of the wires for a more understandable, organized, and easily debuggable build and a 7805 voltage regulator with filter caps allow for an input of more than 5 volts. So far, green wires connect and transmit clock signals while yellow ones are for address lines. Red and black wires are used for Vcc and ground respectively.

Part A: The Clock

Purpose

The purpose of the clock circuit is to provide a universal time constant that synchronizes all CHUMP components to properly execute in instructions in the correct order similar to how time on Earth does the same to humans. Wires corresponding to the working of the clock are coloured green.

References

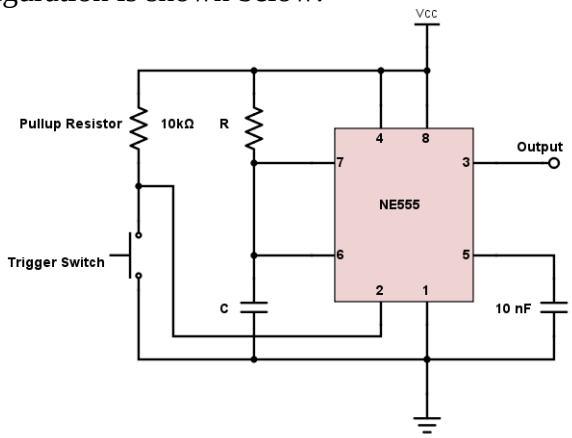
http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html#tasks
http://mail.rsgc.on.ca/~cdarcy/Datasheets/A%20Simple%20and%20Affordable%20TTL%20Processor%20for%20the%20Classroom.pdf
https://youtu.be/kRlSFm519Bo
https://youtu.be/81BgFhm2vz8

Procedure

Parts Table	
Quantity	Description
1	PB machine
2	555 timer IC
1	7805 5V regulator
1	10 kΩ potentiometer
1	PBNO button
2	10 µF capacitor
1	SPDT switch
2	5mm red LED
1	0.1 µF capacitor
1	3mm green LED
3	470 Ω fixed resistor
3	1 kΩ fixed resistor
1	10 kΩ fixed resistor
1	1 MΩ fixed resistor

The most important part of a computer is its clock. This clock determines the computer's speed and is the heartbeat to executing code stored on the CHUMP where once clock cycle equates to both fetching and executing an instruction. This clock is comprised of a rising and falling edge automatically generated by a 555 timer in astable mode or manually generated by a simple pull up resistor and push button, allowing the user to fully understand what the computer is doing before moving on to the next clock cycle and therefore, next instruction.

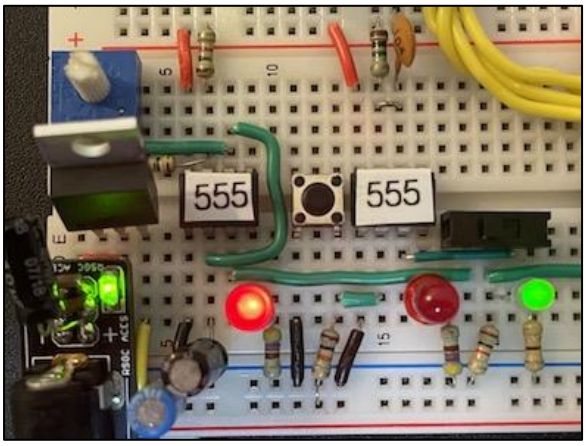
The speed of the astable 555 timer can be adjusted with a potentiometer, increasing the RC charge and discharge time of its supporting external components with a higher resistance and vice versa. For the manual clock signal, another 555 timer is also used but is put into monostable mode. This means that the 555 timer is stable on the low state and becomes high for a set amount of time when a high signal, in this case the push of the push button, is fed into it. When the button is released, the output stays high for the set amount of time no matter the input, effectively removing button bounce by the timer blocking any spikes, and then becomes low once the time runs out. The monostable circuit configuration is shown below.



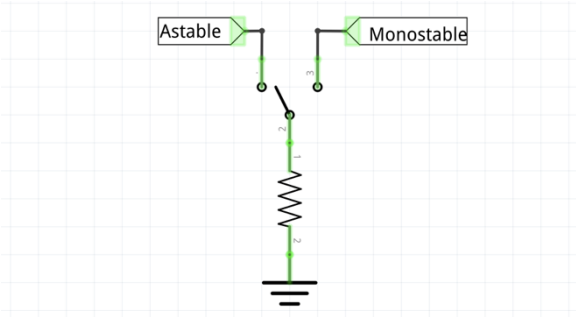
To switch between a manual or automatic clock pulse, a simple SPDT switch is controls whether the output pin of the switch is connected to the manual or automatic signal. At first, when directly wiring the two signals to the inputs and then wiring the output to the program counter, the next part of the CHUMP build, the output would glitch when switching modes. This was solved by putting a pull down resistor on the output to prevent the output pin from floating when in the middle of pulling the switch. The problem with the floating state of the pin is that it is susceptible to noise which produces extra unwanted clock signals.

This configuration takes up less space than the clock circuit built by Ben Eater while still providing the around the same functionality excluding the halt function which can be forgone by switching the clock to manual mode and not pressing the button. This overall configuration was chosen for conserving breadboard space in case of any extra extensions that may be added to the CHUMP.

Media



The clock circuit section



The output pin is temporarily floating when switching. Solved using a pull down resistor

Part B: The Program Counter

Purpose

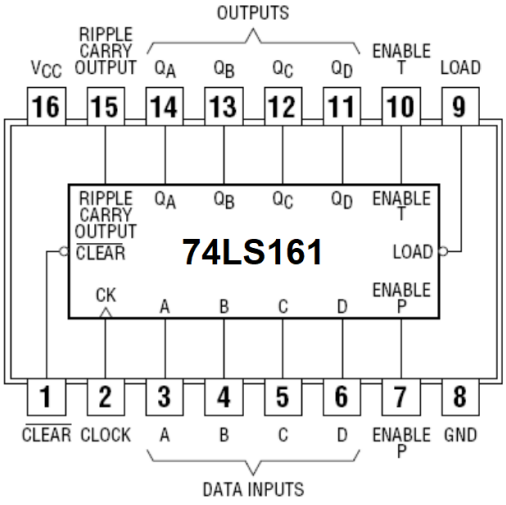
The purpose of the program counter is to track and output which line number/instruction the CHUMP is executing.

References

http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html#tasks
http://mail.rsgc.on.ca/~cdarcy/Datasheets/74LS161.pdf

Procedure

The output of the clock signal is fed into the SN74LS161 program counter which outputs a 4-bit binary number that increments by 1 every rising edge, exactly like the 4516 binary up down counter explored in Project 1.4: The Counting Circuit. Like the 4516, it has a ripple carry pin for cascading multiple program counters for extending the count to 8 bits. Unlike the 4516 however, the program counter has a few more functions such as the clear and load functions.

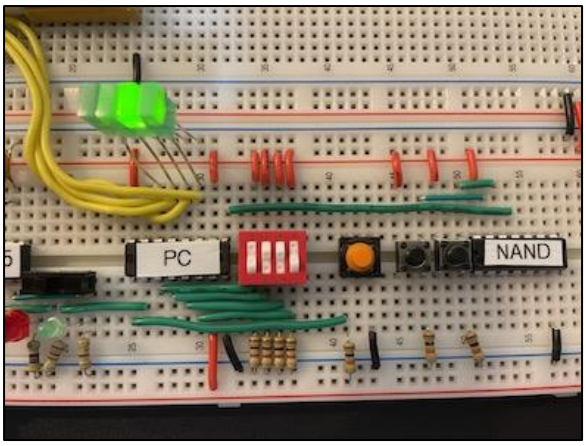


The clear and load function are both active low pins which means when the clear pin is pulled low, the count resets to 0 and when the load pin is pulled low and the clock rises the count jumps to the binary number on the input pins. Clearing the line number is useful for resetting the computer and loading a line number is crucial to jump to a certain line number to execute a specific instruction.

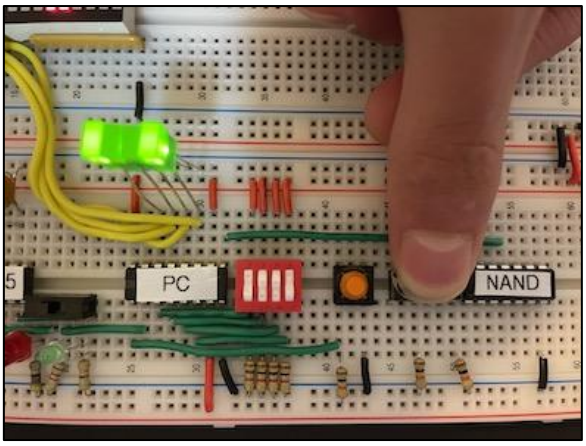
Parts Table	
Quantity	Description
1	SN74LS161
1	SN74LS00 NAND
1	4-bit DIP switch
3	PBNO buttons
1	3mm green LED
3	470 Ω fixed resistor
4	1 kΩ fixed resistor
3	10 kΩ fixed resistor
4	5mm green LEDs

When configured to work with the CHUMP, the pins are wired to manual inputs such as a pull up resistor configured push button connected to the clear pin and a 4-pin DIP switch, a NAND gate and two push buttons this time with pull down configurations controlling the load function. The clear pin configuration is pretty self-explanatory; the button is pushed, sending a low signal to the clear pin which sets the count to 0 while the load pin configuration is a bit more complicated. A NAND gate first takes in two inputs and outputs a its signal to the load pin. The output is low, which set the line number to the input pin presets on the next rising edge of the clock, when both inputs are high and for the rest of the input configurations, the NAND gate outputs a high, keeping the program counter counting normally. To set the inputs, a 4-pin DIP switch configured with 1 kΩ pull down resistors (10 kΩ resistors will not deliver enough current) assigns in binary what number the program counter will jump to when the load pin is low from the NAND gate with two push buttons and the clock signal rises. It should be noted that the manual controls such as the push buttons and the dip switch for the load function will be replaced later on with outputs from other IC's in the later stage of the CHUMP build. For now, they are used for demonstration purposes of the program counter load function.

Media



LED's display the current line number (leftmost is LSB and rightmost is MSB



Orange clears while the two black buttons load. DIP sets input to load 9

Part C: Program EEPROM

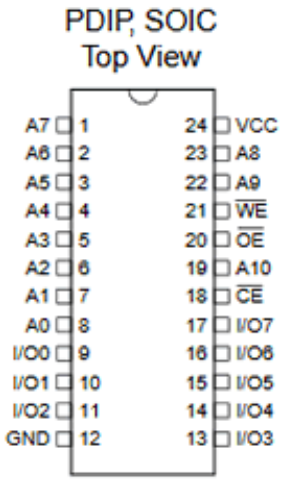
Purpose

The purpose of the program EEPROM is to store the a custom CHUMP program consisting of code made up of 8-bits for each line burned into the EEPROM addresses. This is similar to creating an Arduino sketch with code and uploading it to the processor.

References

https://youtu.be/BA12Z7gQ4P0
https://github.com/rsgcaces/AVROptimization/tree/master/AT28C16EEPROMShieldVersion

Procedure



This next IC, the AT28C16 program EEPROM was an optional endeavor in this week's submission. It can store 2KB of addresses each holding 8-bits. This means there are 11 address pins (2^{11} = 2048) and 8 I/O pins that output/input the 8-bit number when a certain address is selected through binary inputs when reading/writing respectively. To read the data of an address on the EEPROM or to write to an address on the EERPOM, a write enable pin is used. This pin is active low, meaning a low signal will write the states present on the 8 I/O pins to the address that is selected through binary highs and lows. According to the data sheet, this specific low time for proper writing is around 680 nanoseconds. To display the contents, the output enable should be high and the address pins are selected with a binary input. One could upload custom code by hand as shown in Ben Eater's video on replacing combinational logic with EEPROM but for time and effort sake, an EEPROM burner utilizing the ATmega328P with the Arduino IDE was made and used to make writing as easy as possible. The Arduino code used to upload custom code can be found in the references section.

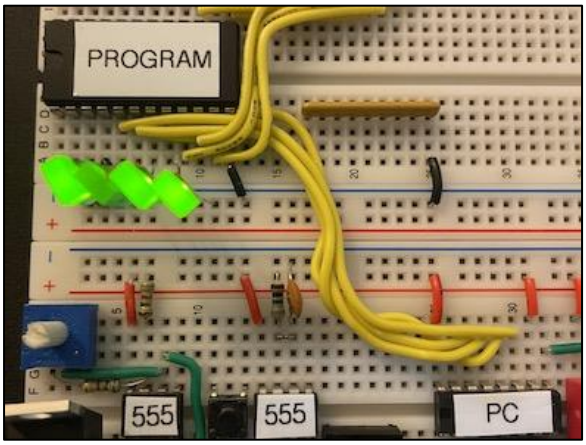
Since the code is limited to a maximum of 16 lines in the CHUMP, address A0 to A3 will only be used so the rest will be grounded. Therefore, when building the EEPROM programmer circuit, only those addresses need to be connected to the microcontroller which saves on wiring.

Parts Table	
Quantity	Description
1	AT28C16
1	Arduino NANO
1	220 Ω resistor network
1	10 LED bar graph
1	USB C to B cable

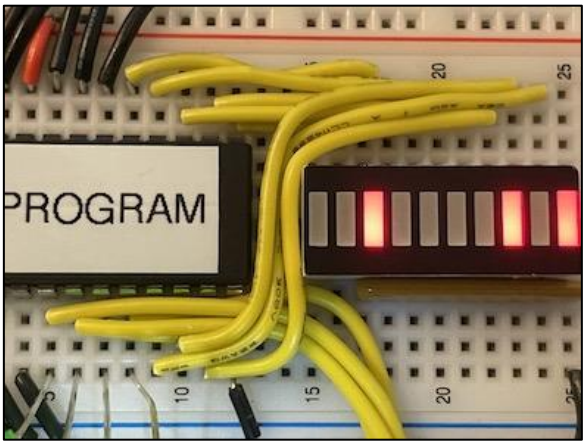
These four addresses are selected by the program counter and the 8-bit code programmed onto it is outputted from the selected address onto its output pins which will lead to the Control EEPROM later on. For now, the output is shown on a bar graph with a current limiting resistor network.

A possibility of extending the CHUMP build is to access the rest of the 7 unused program EEPROM addresses for writing 2^7 or 128 different sketches! Here, the high 7 bits control the sketch/program number while the low 4 are still used for the line numbers. These sketches can be toggled on and off using DIP switches or something similar depending on which one the user wants to run. This extends the use of the CHUMP a lot further as it can hold and output multiple programs without the EEPROM needing to be overwritten.

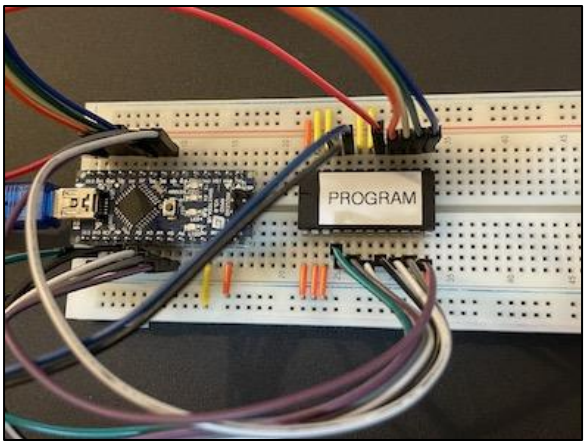
Media



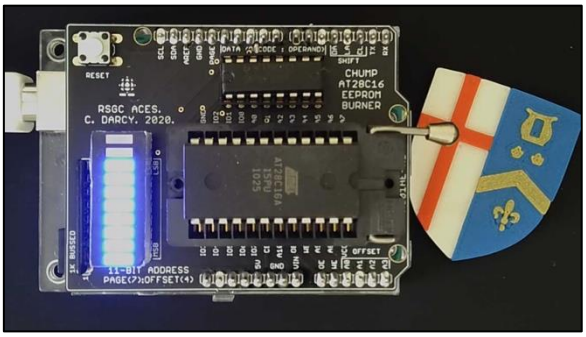
Program counter output is wired to pins A0 to A3. LEDs moved to show correct binary output



Output assembly code stored in addresses selected by the program counter is shown on the bargraph



Custom EEPROM burner using an Arduino NANO on a breadboard



An EEPROM burner shield that fits on an Arduino UNO

Code Structure

Purpose

The purpose of this section is to design custom binary (machine) code and for writing programs that drive the CHUMP processor and output useful information.

References

https://codepen.io/davefdavef/full/WNxRpMR

Procedure

The beauty of designing a 4-bit computer from scratch is the ability to design your own coding language. This coding language, dubbed the Chumpanese language by its creator, David Feinberg, is the lowest of low-level coding as each bit and byte can and will be controlled allowing unprecedented freedom to a level way beyond the mid-level coding of ports on the ATTiny and mega series or the severely restrictive high-level language of the Arduino IDE. As explained in the program EEPROM section, each line will contain an 8-bit instruction or machine code that will perform a specific function. The basic rundown of the way Feinberg designed the Chumpanese language is for the 8-bits to be split into a high-nybble and a low-nybble. The high-nybble is the op-code, indicating which operation the processor should perform and the low-nybble is the operand value limited to 4-bits, that is used as a data value, a RAM address, or a program line number. A table of all the operations is shown below:

OpCode (Machine)	Operand (const/mem)	Mnemonic (Assembly)	High-level Summary	
0000	const	LOAD	`accum = const;`	`pc++`
0001	mem		`accum = mem[addr];`	`pc++`
0010	const	ADD	`accum += const;`	`pc++`
0011	mem		`accum += mem[addr];`	`pc++`
0100	const	SUBTRACT	`accum -= const;`	`pc++`
0101	mem		`accum -= mem[addr];`	`pc++`
0110	const	STORETO	`mem[const] = accum`	`pc++`
0111	mem		`mem[addr] = accum`	`pc++`
1000	const	READ	`addr = const;`	`pc++`
1001	mem		`addr = mem[addr];`	`pc++`
1010	const	?	`?`	`pc++`
1011	mem		`?`	`pc++`
1100	const	GOTO	`pc = const;`	`pc++`
1101	mem		`pc = mem[addr];`	`pc++`
1110	const	IFZERO	`accum == 0 ? pc = const : pc++`	
1111	mem		`accum == 0 ? pc = mem[addr] : pc++`	

To break down this complicated table, the terms will be explained first. The op-code is the high-nybble of the 8-bit program code. Here, it is divided into two parts: the high three bits control what operation is being carried out while the least significant bit decides whether the operand is interpreted as a constant value defined by the low-nybble of the 8-bit program code or a RAM address, in which case the low-nybble of the program code is ignored. The accumulator (addressed as accum in the table) serves as the 4-bit output to the CHUMP where displayed values over 15 will roll over to 0 while pc stands for the program counter line number. Mem stands for the RAM IC while addr is defined as the address number located in mem or RAM, similar to how EEPROM stores its data in address numbers. The question marks are undefined functions which will later be defined as unique operators later in the final CHUMP build.

Here is a brief summary of what all the defined operators in the table do:

- LOAD (const): Sets the output of the accumulator to the constant defined in the low-nybble of the program code. Program counter increments by 1.
- LOAD (mem): Sets the output of the accumulator to the value in a specified RAM address. To specify the address, this function should be preceeded by a READ (const) to place a 4-bit value into the address register. Program counter increments by 1.
- ADD (const): Increases the accumulator by the defined constant value. Program counter increments by 1.
- ADD (mem): Increases the accumulator by the value in the defined address of RAM. Should be preceded by a READ (const) function as with all memory functions. Program counter increments by 1.
- SUBTRACT (const): Decreases the accumulator by the defined constant value. Program counter increments by 1.
- SUBTRACT (mem): Decreases the accumulator by the value in the defined address of RAM. Program counter increments by 1.
- STORETO (const): Stores the value in the accumulator to an address number in RAM specified by the const value. Program counter increments by 1.
- STORETO (mem): Stores the value in the accumulator to an address number in RAM specified by the value in the defined address of RAM. Program counter increments by 1.
- READ (const): Sets the address value to the const value (low-nybble in the 8-bit in the program code). Program counter increments by 1.
- READ (mem): Sets the address value to the value in the defined address of RAM. Program counter increments by 1.
- GOTO (const): Sets the program counter (aka line number) to the const value. Program counter increments by 1.
- GOTO (mem): Sets the program counter to the to the value in the defined address of RAM. Program counter increments by 1.
- IFZERO (const): If the accumulator output is 0, the program counter is set to the const value, else the program counter advances by 1.
- IFZERO (mem): If the accumulator output is 0, the program counter is set to the value in the defined address of RAM, else the program counter advances by 1.

Here is an example CHUMP program written in Chumpanese that will run on the final CHUMP build. It simply takes the value in the accumulator, processes it, and infinitely loops on the squared value that was put into the accumulator. In the example code, an input value of 3 was put in so a output value presented on the accumulator will be 9. This code was made and tested with the help of Feinberg's online CHUMP simulation.

High Level	Machine Level		CHUMP (Assembly)	Comment
	Address	Instruction		
`x = 3;`	0000 (0)	0000 0011	LOAD 3	Accum←3, pc++
	0001 (1)	0110 0101	STORETO 5	Accum←[5], pc++
`y = 3;`	0010 (2)	0110 0110	STORETO 6	Accum←[6], pc++
`while(x != 0){`	0011 (3)	1110 1110	IFZERO 14	accum==0 ? pc←14 : pc++
`x--;`	0100 (4)	0010 0001	SUBTRACT 1	Accum-=1, pc++
	0101 (5)	0110 0101	STORETO 5	Accum←[5], pc++
	0110 (6)	1000 0111	READ 7	Addr←7, pc++
`z = x`	0111 (7)	0001 0000	LOAD IT	accum←[7], pc++
	1000 (8)	1000 0110	READ 6	addr←6, pc++
`z += y`	1001 (9)	0011 0000	ADD IT	accum←accum+[6], pc++
	1010 (10)	0110 0111	STORETO 7	Accum←[7], pc++
	1011 (11)	1000 0101	READ 5	addr←5, pc++
`z = x`	1100 (12)	0001 0000	LOAD IT	accum←[5], pc++
`}`	1101 (13)	1100 0011	GOTO 3	pc←3
`x = z;`	1110 (14)	1000 0111	READ 7	addr←7, pc++
	1111 (15)	0001 0000	LOAD IT	accum←[7], pc++

There are still a few problems with it such as the need set address 7 to 0 by cutting power off to erase the RAM for the program to function properly after 1 iteration (assuming that cutting power sets all addresses to 0). This is because of the 16 line restriction of the program counter so it is unable to support the few extra lines that erase the specified address of RAM. The way to solve this would be cutting down the number of lines used by eliminating and shortening verbose lines of code, which can be done. However, the writer of this report spent an absurd amount of time burning an obscene amount of calories from brain usage of first trying to figure out how to put it into Chumpanese, then trying to shorten it and then finally giving up to work on a modulus function (that isn't working yet…) which further emphasized the pain and voices in his head. So, here it is so far, but rest assured that this code will be improved in the next CHUMP report. A more detailed explanation of the code and how each line works is covered in the YouTube video.

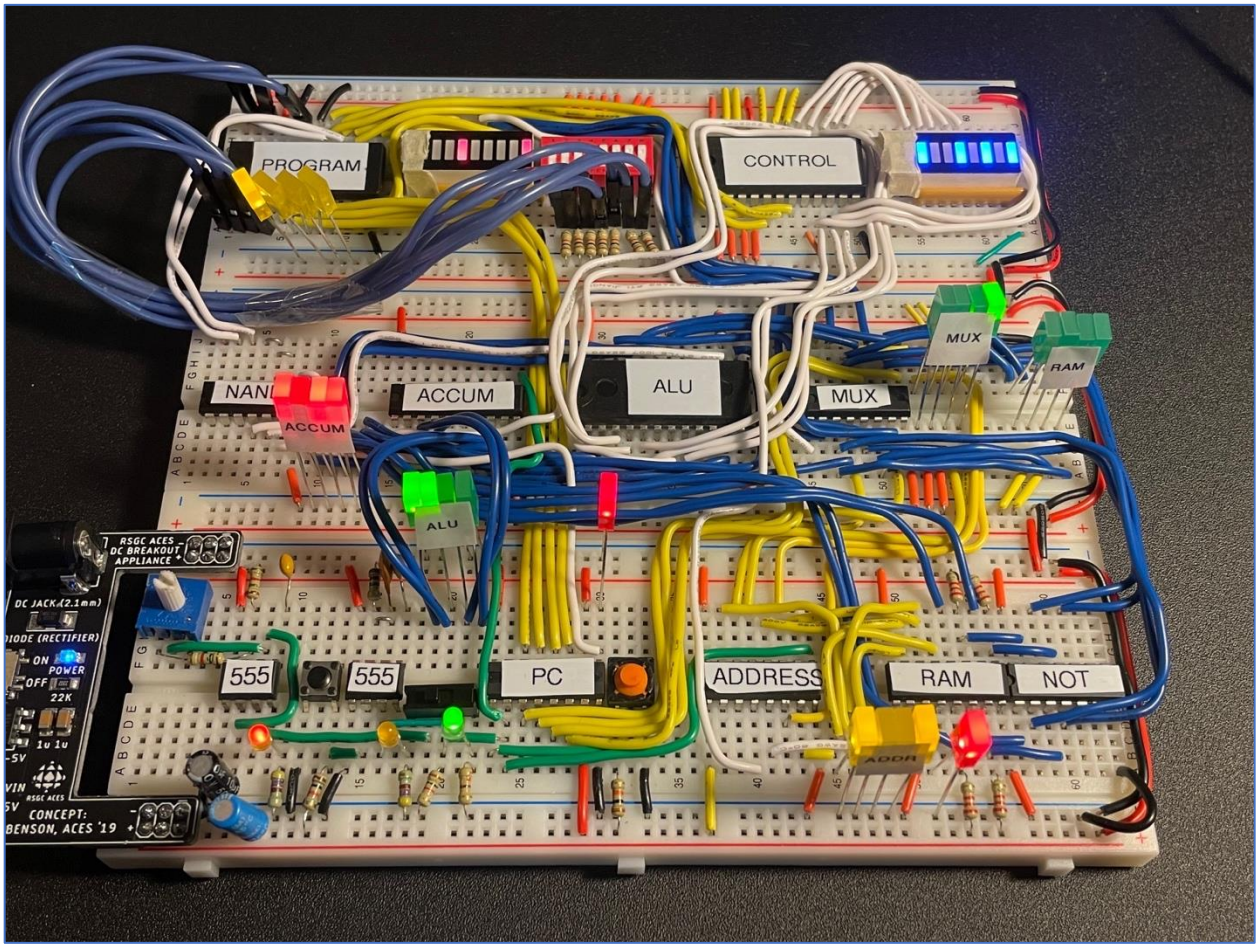
Media

YouTube video link: https://youtu.be/b5qDwCN9Q2c

Reflection

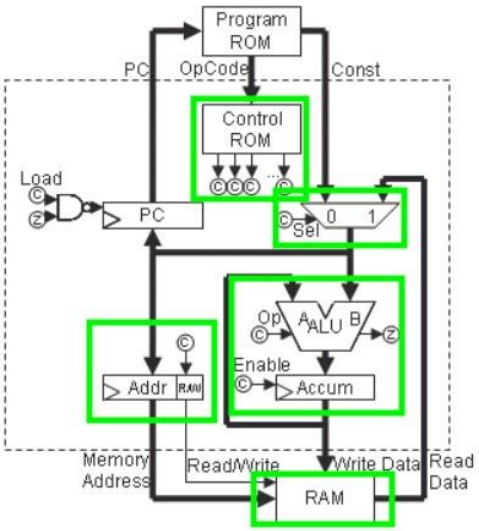
This circuit is definitely akin to first building the counting circuit in grade 10. New parts were introduced, there was a lot of breadboarding, but most importantly, it was challenging and time consuming with a lot of room for errors. The part that was the most frustrating and had me the most stuck was figuring out how to create my custom made squaring code and then trying in vain to shorten it. I then gave up and tried to create a modulus function CHUMP program, which also ended up not working. Since I wasn't making any progress with them, I tried harder and harder, getting more and more tired and angry as I couldn't bear not fixing it. Eventually, after a long 5 or 6 hours, I managed to let go and moved on to the more pressing DER report. Thinking back, I really wish I could have stopped earlier as I suffered a price for spending all my time and energy on the code (the price being staying up very very late at night). Another experience that was similar was my stubbornness of wanting to include animations in my video from a program called Manim. These animations are neat and look wonderful as showcased by its creator 3Blue1Brown (https://www.youtube.com/c/3blue1brown) and I was really looking forward to including them in my video as its theme fits in well to explaining the Chumpanese code language. However, like my custom code, I had trouble with figuring out the python code required to make animations as it was my first time using it, but I was stubborn to make it work. So a lot of time went into trying in vain instead of doing the more important things such as my DER or actual content of my video. These experiences has therefore taught me that it is ok to let go of things you can't fix right away and move on so as not to waste time, and then eventually come back to it with a fresh mind. So before the final CHUMP build project starts, I will take my time to improve my code and learn how to use Manim so that it can be included in my final CHUMP video as the animations that you can create, even ones with simple numbers and shapes, are simply too beautiful and perfect for explaining CHUMP.

Project 3.3: CHUMP Final



Purpose

The purpose of this project is to build and understand the rest of the components that make up the 4-bit CHUMP computer. Pictured below are the rest of the components needed to complete the CHUMP build. The functionality of the CHUMP includes custom code that can perform basic arithmetic functions, conditional statements, and memory storage to keep data for later use.



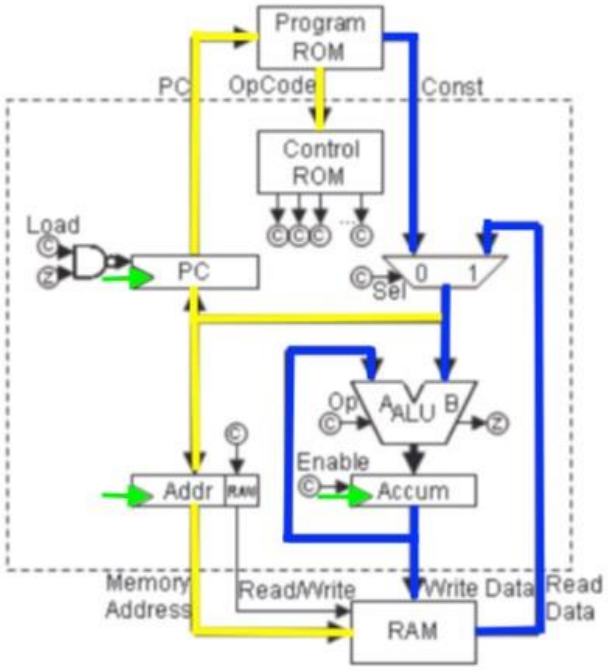
Theory

With the CHUMP complete, here is an overall rundown. First, 555 timers with variable frequency and modes provide a clock signal to synchronize all components. The clock is connected to the program counter and accumulator and address flip-flops. Next, the program counter takes in this clock signal and increments its 4-bit binary counter on the rising edge. This counter determines the line number with an 8-bit piece of code that the CHUMP executes. To do this the outputs of the counter are routed to the program EEPROM where the code written by a user is located. The outputs then select the 8-bit line of code to execute from the program EEPROM. This 8-bit code is the custom chumpanese language containing an op-code in the high byte and a constant value in the low byte. The op-code determines the function that the CHUMP performs and the constant is the numerical value attached to that function. All this was done in the first part of the CHUMP build.

Since the op-code is in chumpanese, it must be translated into usable machine code where each bit controls a specific component. To do this the op-code is passed through another EEPROM designated as the control EEPROM outputting the machine code, referred to as control bits, from the inputs of the op-code. The constant does not need to be translated into machine code as it is in the native binary that all chips can understand. This constant is fed into a multiplexer which chooses with a control bit between the constant in the program EEPROM of a value read from RAM. The output of the multiplexer then is fed into the ALU, program counter and address flip-flops as the constant can represent an operand, a line number for the program counter to jump to, or a location to store to or read values from RAM. The ALU computes two 4-bit operand values with an operator chosen from the control bits. The result is presented on the accumulator flip-flops whose output can be enabled or disabled using a control bit. The output of the accumulator is then fed into one of the operand inputs of the ALU with the other operand coming from the multiplexer. The RAM stores the output of the accumulator into a selected address and the RAM outputs are then fed into the input of the multiplexer. A control bit tells the RAM whether to read a value, where it reads data from the selected address, or writes a value where it stores the accumulator value in the selected address. This is overall the basic rundown of CHUMP.

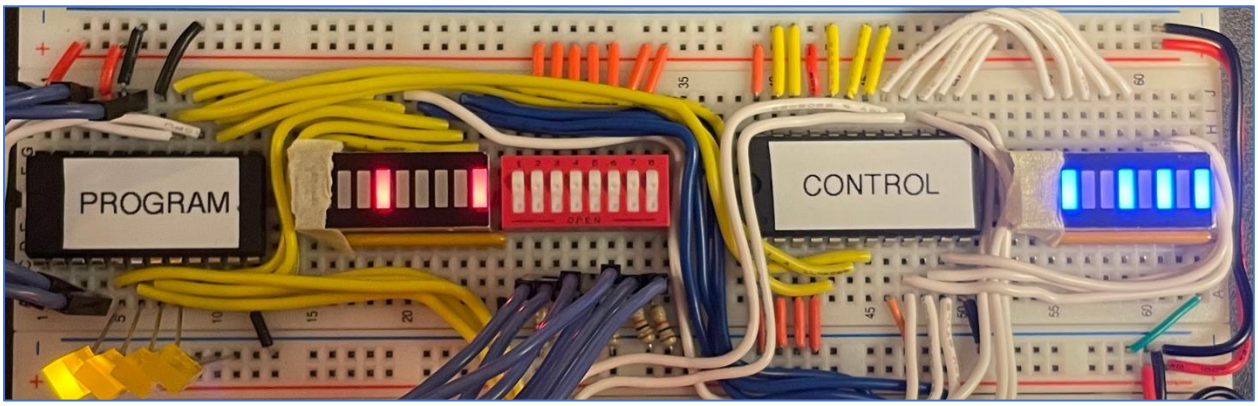
Since the CHUMP is built on breadboards, a layout of all the IC's is crucial to a neat and understandable build. I placed components in such a way to balance organization and size which proved to be an arduous task of planning and visualizing. Eventually, a decent layout was made with a compact size of three breadboards as opposed to four without sacrificing too much organization. The layout could be even more orderly by rearranging and shortening wire lengths but I felt that the time and effort could be better spent elsewhere.

An easy way of improving visual understandability without having to rewire and reorganize was to use coloured wire corresponding to the type of data it carries. In my CHUMP build, there are four main types of colors: green, yellow, blue, and white. Green was used for wires transmitting or generating clock signals. Yellow was defined as address wires that serve to read data from or direct data to specific addresses. Blue lines carry data values such as the operands of the ALU, outputs of RAM, and any inputs or outputs with a 4-bit binary value that does not correspond to an address value. Finally, white wires symbolize the control bits of the machine code. With this colour scheme, it is easier to debug and understand the CHUMP and provides visual appeal. The block diagram with the data lines as blue, the address lines as yellow, and green arrows as clock connections is shown below.



Also, labels were used for each IC and the LED outputs corresponding to the value that the IC is handling were also labeled, further improving the readability.

Control EEPROM



The binary chumpanese code in red vs the binary control code in blue for the function ADD 1

Purpose

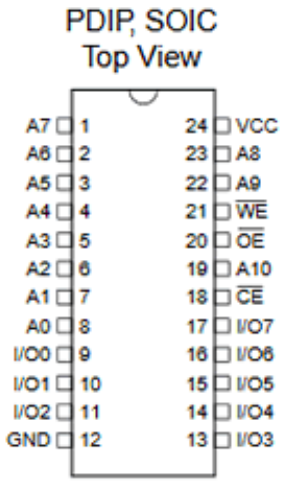
The first part of the final CHUMP build is wiring up the control electronically erasable program read-only memory (EEPROM) which converts the chumpanese language code outputted by the program EEPROM into machine code that acts as logic inputs to certain components to influence their output and CHUMP behavior.

References

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

Procedure

The control EEPROM architecture is the same as the program EEPROM as they both use the same AT2816C chip, however, they are both loaded with different programs. The program EEPROM takes in a custom code written in the Chumpanese language while the control EEPROM converts this language into usable high and low outputs that directly control the rest of the components such as the ALU, the RAM, and the accumulator. These outputs are called machine code which is the lowest level and most intricate type of coding as it takes a complete understanding of the computer architecture and the subsequent components to understand what each bit is doing and hence understand the code.

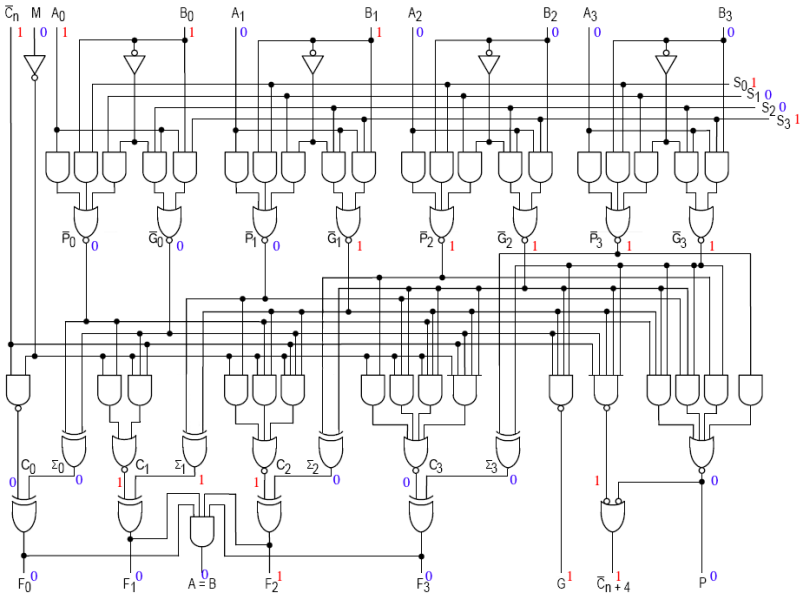


The control EEPROM is wired similar to the program EEPROM except that the address select pins of the control are wired to the I/O pins of the program which output the chumpanese codes. Since the chumpanese codes are split into a high byte consisting of an op-code function and a low byte corresponding to the constant portion, only the op-code will be routed to address pins A0 to A3 of the control EEPROM. The constant byte is already at its simplest binary form so it does not need to be converted. All other address pins are tied to ground to avoid floating pins which affect what addresses the EEPROM reads and data it outputs. A0 can be further tied to ground since the lowest bit of the op-code corresponds to selecting between the read and write function of the RAM. In the chumpanese language, this is identical to selecting between a constant or an IT. This bit from the program EEPROM can be directly wired to the select pin of the multiplexer which is explained in the multiplexer section. So in total, the control EEPROM only takes in 3 bits from the program EEPROM which can represent numbers 0 to 7. These numbers each represent one of the functions defined in the chumpanese code so each function will have its own unique control code.

Like the chumpanese code, the control codes consist of a high byte and low byte split into separate functions. The high byte contains the select lines that chooses the arithmetic function that the ALU performs. The low byte comprises a mix between a mode and carry input function connected to the ALU and an accumulator enable bit and read/write bit. These two bits are wired to the accumulator and RAM respectively. The table shown below links each function with the corresponding control codes.

Instruction	Sel	(ALU)	S3	S2	S1	S0	M	Cn	Accum	R/W
LOAD const	0	B	1	0	1	0	1	X	0	1
LOAD IT	1	B	1	0	1	0	1	X	0	1
ADD const	0	A plus B	1	0	0	1	0	1	0	1
ADD IT	1	A plus B	1	0	0	1	0	1	0	1
SUBTRACT const	0	A minus B	0	1	1	0	0	0	0	1
SUBTRACT IT	1	A minus B	0	1	1	0	0	0	0	1
STORETO const	0	X	X	X	X	X	X	X	1	0
STORETO IT	1	X	X	X	X	X	X	X	1	0
READ const	0	X	X	X	X	X	X	X	1	1
READ IT	1	X	X	X	X	X	X	X	1	1
USER const	0									
USER IT	1									
GOTO const	0	Logic 1	1	1	0	0	1	X	1	1
GOTO IT	1	Logic 1	1	1	0	0	1	X	1	1
IFZERO const	0	Not A	0	0	0	0	1	X	1	1
IFZERO IT	1	Not A	0	0	0	0	1	X	1	1

Arithmetic Logic Unit (ALU)



Purpose

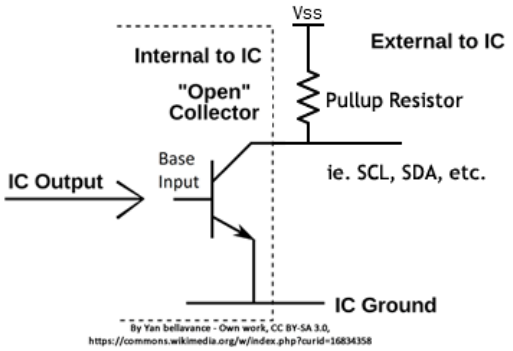
The 74LS181 Arithmetic Logic Unit (ALU) acts as the brain and CPU of the CHUMP by providing and computing a plethora of arithmetic and logical functions.

References

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

Procedure

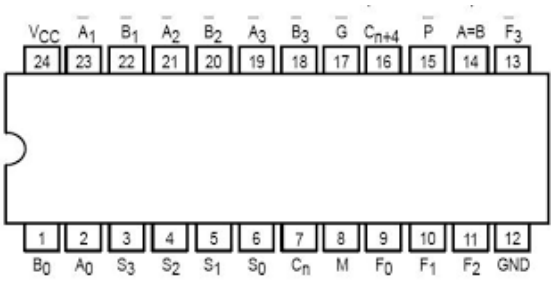
The 74LS181 ALU is a 24-pin chip that computes 4-bit arithmetic and logic functions and can be chained together to increase the computation bit size. It played a key role as the CPU in early computers. The previous ALUs were built out of simple logic gates such as full adder circuits chained together but the 74LS181 produced by Texas Instruments revolutionized the ALU by optimizing for speed and high performance with an unconventional complex implementation of logic gates housed in a TTL chip.



The chip's arithmetic functions are based on 16 logic functions selected by the S0 to S3 pins. The binary inputs of these functions can only be 4 bits in length and are labeled as A0 to A3 as the one input and B0 to B3 as the second. The output is presented on the function output pins F0 to F3. When all of these function output pins are high, the A=B pin output pin will become high which will be crucial to performing the GOTO function. This is an open collector pin, meaning a pull-up resistor is needed to stop the pin from floating when a low is not present. By combining the addition of input A with these logic functions, the arithmetic functions can be created. To switch between logic functions to an arithmetic one, the mode pin and carry pin alter the function depending on its state. The mode adds A to the resulting output of one of the 16 logic functions and the carry pin simply adds one the overall value. A table consisting of all the functions and how to select each one is shown below.

SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	$\overline{C_n}$ = H (no carry)	$\overline{C_n}$ = L (with carry)
L	L	L	L	F = $\overline{A}$	F = A	F = A PLUS 1
L	L	L	H	F = $\overline{A+B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	F = $\overline{A}$B	F = A + $\overline{B}$	F = (A + $\overline{B}$) PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	F = $\overline{AB}$	F = A PLUS A$\overline{B}$	F = A PLUS A$\overline{B}$ PLUS 1
L	H	L	H	F = $\overline{B}$	F = (A + B) PLUS A$\overline{B}$	F = (A + B) PLUS A$\overline{B}$ PLUS 1
L	H	H	L	F = A ⊕ B	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	F = A$\overline{B}$	F = A$\overline{B}$ MINUS 1	F = A$\overline{B}$
H	L	L	L	F = $\overline{A}$ + B	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	F = $\overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	F = (A + $\overline{B}$) PLUS AB	F = (A + $\overline{B}$) PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A PLUS A	F = A PLUS A PLUS 1
H	H	L	H	F = A + $\overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	F = (A + $\overline{B}$) PLUS A	F = (A + $\overline{B}$) PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

As an example, the arithmetic operation of addition consists of the logic function F = B combined with a high present on the mode pin and a low carry pin. Since arithmetic functions are created from adding the A input to the output of the logic function of the A and B inputs, the outputs presented on the F0 to F3 pins will be A added to B. This is done by making use of three other internal 4 bit variables in the ALU which are the carry-lookahead (C), the propagate (P) and generate (G) bits.



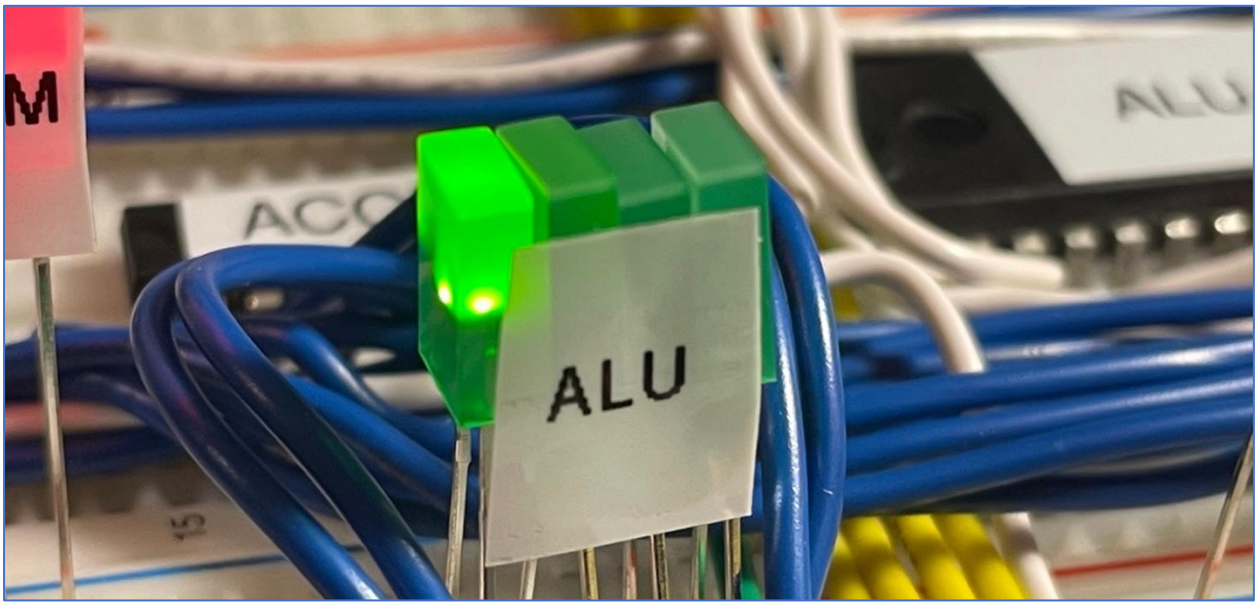
The carry lookahead removes the ripple carry effect of carrying each bit to the next highest bit, similar to doing long addition by hand. Adding 1 to 9999 means that the 1 must be carried through each column to the next which is computationally slow. To avoid this serial way of adding, the ALU computes all the carries before doing the addition and then adds these carry bits to the input in a column-wise fashion. Coming back to the example of adding 1 to 9999, this method would calculate the carry bits as 11111, as there are 5 carries in the operation, and then add each 9 to the carry bit to produce each digit of the result. This involves four 9+1 operations and a 0+1 the resultant of which is 10000. How the ALU achieves this is through the P and G signals calculating whether a carry can be generated or not in a certain bit position (the 1s bit, the 2s bit, the 4s bit, etc). For example, when A and B inputs of a certain bit position are both low, no carry can be generated regardless of the carry pin adding 1 or 0. When A and B are both high, a carry will always be generated. This is a generated case and the G bit will be set to high. When A or B is high and A or B is low, a carry depends on the carry pin. This is a propagate case and the P bit will be set high. Putting these cases together, a carry bit can be calculated from putting the carry pin, P, and G bits through the following logic: P AND (Cn OR G).

This only applies for the first carry bit. Other higher-order bits have more combinations that generate or don't generate a carry bit. The logic can get very complicated so it will not be explained here. For more detail check out Ken Shirrifs blog where he explains the chip's logic and structure in detail and even provides an interactive and detailed simulation of the ALU.

While all arithmetic functions use the carry bits in some way, logic functions do not. Setting the mode pin high sets all carry bits high, disabling its function. This yields an overall output of NOT(A XOR f) for each bit of the output where f are the select pin inputs. For arithmetic outputs, Fn, they are each formed through Cn XOR Pn XOR Gn where Cn are the carry bits and n is the bit position.

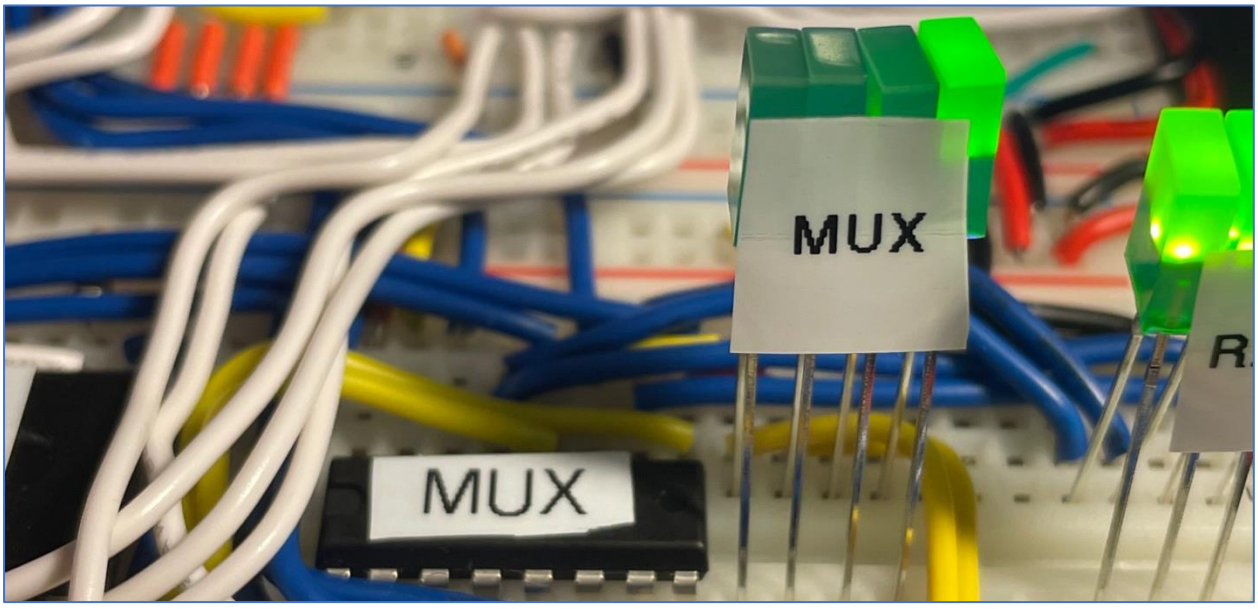
To use the ALU, one only needs to understand how each pin changes the function of the ALU as described in the ALU function table, however, an internal understanding of the chip always helps. In the case of the CHUMP build, the rest of the components only interact with the external inputs and have nothing to do with the internal logic that goes on so it is not explicitly required to understand it to build the CHUMP. These external inputs come from the control EEPROM outputs as mentioned in the control EEPROM section. Now that an understanding of the ALU has been established, the control codes start to make sense. For the subtraction function, the high byte 0110 of the control code corresponds to selecting the row of the table in which the subtract function is located and the two high bits on the low byte select the column. Here, both mode and carry are low to select it.

Media



LEDs show ALU F0 to F3 outputs

Multiplexer



Output of the multiplexer

Purpose

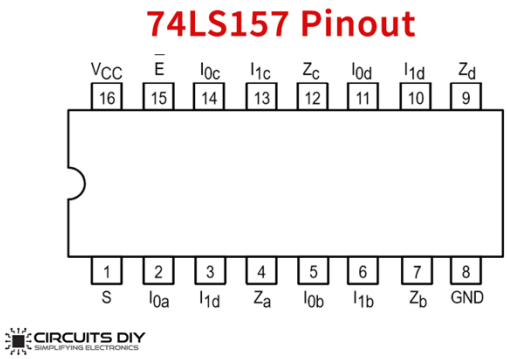
The multiplexer decides whether the input to the ALU comes from the 4-bit constant in the program EEPROM or a 4-bit value read from the RAM. In the context of transportation, it acts as a rail selector to direct trains to certain locations.

References

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

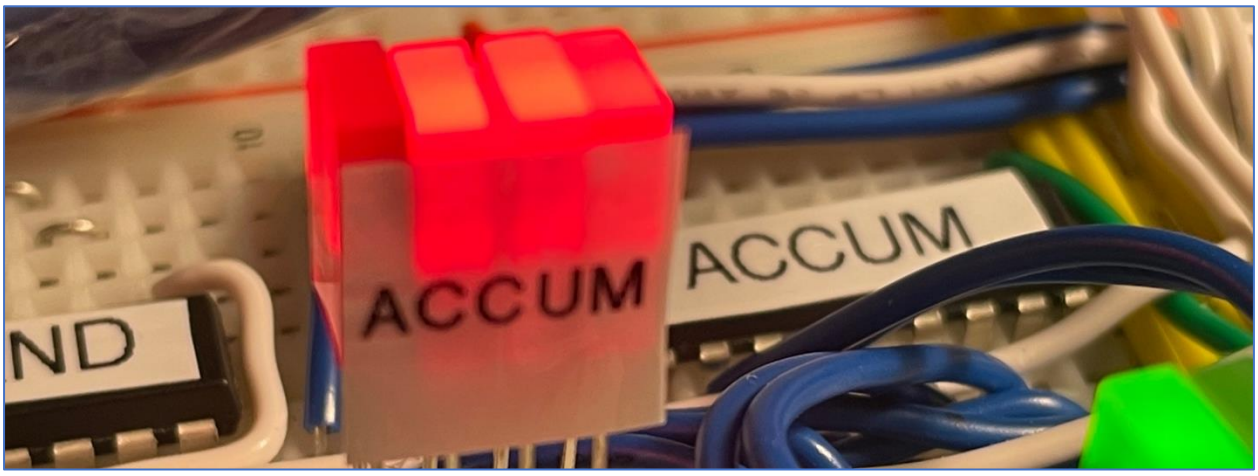
Procedure

The SN74LS157 Quad 2-line to 1-line data selector/multiplexer is a simple chip that takes in two 4-bit inputs and selects which one is presented on its output pins through the usage of a select pin. The input pins are A1 to A4 and B1 to B4 while the output pins are Y1 to Y2. The select line pin sets the outputs equal to the A inputs if it is low or the B inputs if it is high while the strobe when high sets all outputs to low.



In the context of CHUMP, the A input line comes from the 4-bit constant of the 8-bit chumpanese code in the program EEPROM which is the low byte. The B inputs come from the RAM outputs. The select pin is wired to the lowest bit of the op-code portion of the program EEPROM output. As a reminder, this bit controls whether the function selects the constant shown or a value from RAM (denoted as IT). The addition of the multiplexer allows this operation to function. The outputs whether from the constant or RAM are then routed to the B inputs of the ALU. They are also connected to the address D flip-flops for RAM storage and the program counter inputs to select a certain value to present on the program counter outputs. The reason for this is discussed in the other sections.

Accumulator



The binary chumpanese code in red vs the binary control code in blue for the function ADD 1

Purpose

The purpose of the accumulator is to present and temporarily store and use the outputs from the ALU. The outputs of this chip can be considered the final result of the CHUMP code that the user receives.

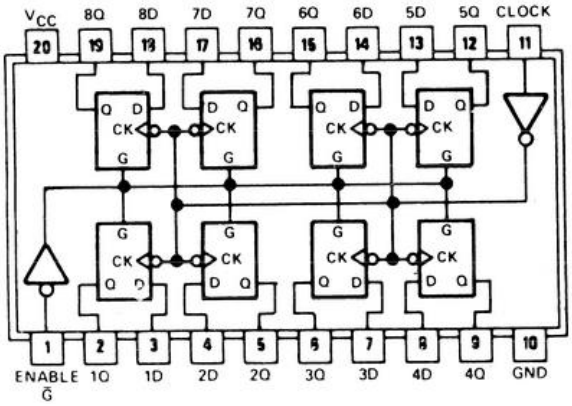
References

https://www.electronics-tutorials.ws/sequential/conversion-of-flip-flops.html

Procedure

The 74LS377 accumulator register is a 20-pin IC that houses a bank of 8 D-type flip-flops. Hence, it is not formally called the accumulator but is instead referred to as an octal quad d-type flip-flop IC as there are 8 in total and 4 on either side. These are denoted by the Q and D letters where D is the input and Q is the output. The enable (E) pin controls whether the outputs are presented and the clock pin takes in a clock signal.

The D-type flip flop or a data type flip flop is a flip-flop/SR latch with some extra gates to take in a data bit and a clock signal. As a quick recap/rundown, a flip-flop takes in two inputs: a set and a reset pin and outputs Q and Q_ where Q_ is the NOT or opposite of Q. The set pin when changing states switches the outputs states hence the name flip-flop and when the set pin switches states, the outputs do not change hence its other name as an SR (set-reset) latch. Only when the reset pin is triggered does the output revert back to its original state where it can be latched again.

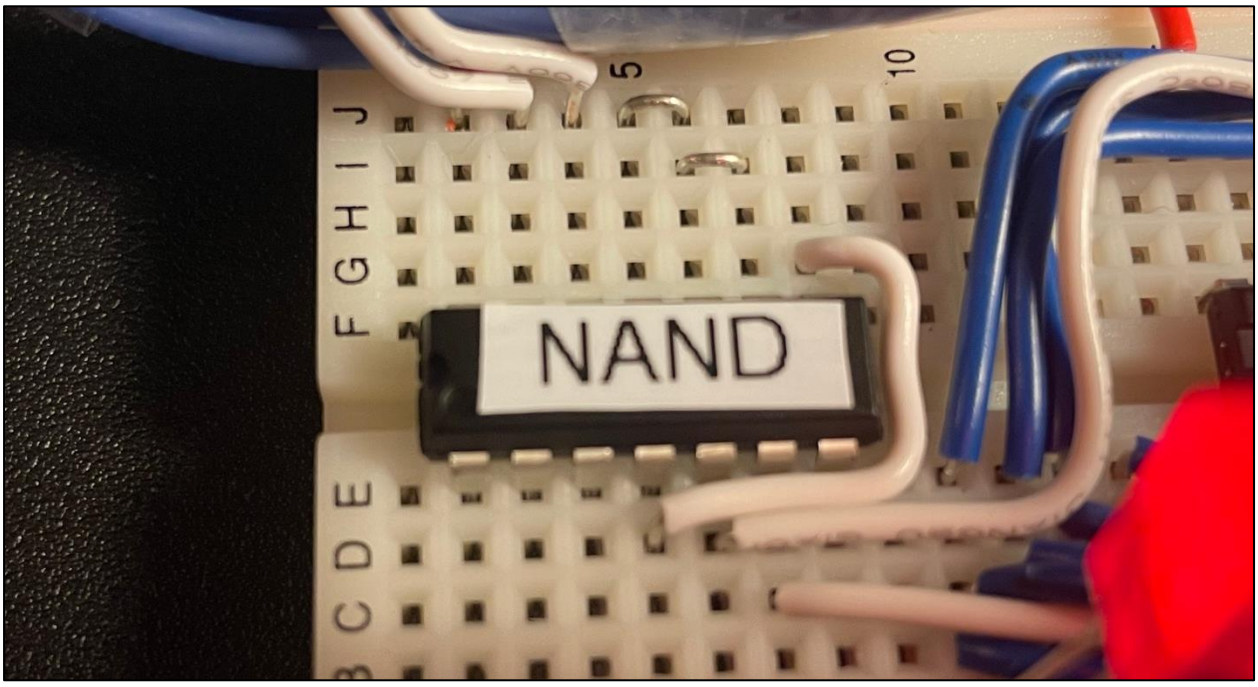


The D-type flip-flop contains the regular flip-flop along with two extra NAND gates and an optional inverter which is there to make sure that the S and R pins are complements of each other. It can be omitted by connecting the output of the NAND gate to one of the inputs on the other NAND gate, saving a gate. Essentially, this type of flip flop makes the output Q follow the D input on the rising edge of the clock signal.

In the context of the CHUMP, the inputs to the D inputs are from each bit of the F0 to F3 output pins of the ALU. LEDs connected to the Q outputs of the flip-flops present the output to the user. These outputs are also connected back to the A inputs of the ALU to facilitate adding, subtracting, and other functions of the output with the B inputs of the ALU. So the code ADD 3 adds the output shown on the accumulator (representing the A input) with the value 3 (representing the B input) that comes from the multiplexer. The clock pin of the accumulator originates from the clock signal generated by the 555 timers, latching the outputs shown and allowing temporary storage. It also prevents the ALU from constantly changing its outputs as fast as it can by synchronizing this change with the clock signal. Therefore, the ALU output will always be one step ahead of the accumulator outputs. The enable pin is attached to one of the control bits wired to the control EEPROM. By doing so, the CHUMP program can control the output shown from the inputs of the F0 to F3 pins on the ALU. A GOTO function for example will temporarily change all F0 to F3 pins high. By setting the enable pin high to disable the change in the outputs, the true arithmetic output is preserved and not altered to 1111. The accumulator outputs also connect to the data in pins of RAM to store the output data. This is discussed in the RAM section.

With the chips set up so far (555 clock, program counter, program, and control EEPROM, multiplexer, ALU, and accumulator) some chumpanese operations can be performed such as the ADD, SUBTRACT, and LOAD as these functions only involve these components. The other four functions require changes to the previous circuitry or setup of the remaining address flip-flops and RAM ICs.

Other Changes



New implementation of the NAND gate

Purpose

Before moving on to implementing the rest of the ICs, it is important to note the changes that have occurred to the program counter section as it was necessary to do so to add certain functions. These functions are the GOTO and IFZERO functions that set the program counter to a certain value to execute the specified instruction located on the program EEPROM.

References

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

Procedure

The changes that have been made are the inputs to the load pin and the load input pins. In the code, clock, and counter circuit, these were originally controlled by two push buttons and a 4-pin DIP switch respectively. Now, they are wired electronically to implement the GOTO and IFZERO functions. To do so, the input load pins were wired to the multiplexer output as the number constant data from either the program EEPROM or RAM are passed through, allowing the program counter to jump to the number in the constant data specified from those chips. To set the program counter to the binary number present on these load input pins, the two push buttons connected to the inputs of a NAND gate are replaced by wires connecting a control bit and a Z flag bit. The Z flag bit comes from the A=B open-collector pin on the ALU which turns high once all output pins are high. This is why the op-code in the control EEPROM specifies a Logic 1 for GOTO and Not A instruction for IFZERO as all output bits will turn on, triggering the A=B pin, if logic 1 is called or if Not A is true. This will load the program counter to the number in the constant or data read from RAM. To do this the A=B pin is simply wired to the input of the NAND gate whose output connects to the load pin on the program counter.

Unfortunately, the control bit to trigger GOTO and IFZERO is not included in the control EEPROM as there are not enough bits. It is also not an option to forgo this bit since if all outputs are high on the ALU for any reason, the program counter will always jump to the number presented on the data coming from the multiplexer whether it was intentional or not. The control bit is therefore required to only trigger this once a GOTO or IFZERO command has been called. To add this crucial bit, there must be a way to detect one of these commands to trigger this bit. Luckily, the chumpanese code functions were strategically ordered to save a control bit on the control EEPROM. Taking a look at these commands reveals that the two highest bits, bit 7 and bit 8 or the op2 and op3 op-code bits are simultaneously high in both GOTO and IFZERO. So, by wiring these two bits into an unused NAND gate on the 74ls00 IC it outputs a 0 only when one of these commands is called. Before it is wired to the input of the other NAND gate, the bit must be inverted. This is done on the CHUMP by wiring it to both inputs of a NAND gate, successfully flipping the bit.

With these changes in place, the GOTO and IFZERO functions now work along with the ADD, SUBTRACT, and LOAD. All that is left to do is add in the READ, LOADIT, and STORETO functions by configuring the RAM and flip-flop address IC.

Hex D Flip-Flops

Address outputs wired to RAM addresses with the red LED read/write state

Purpose

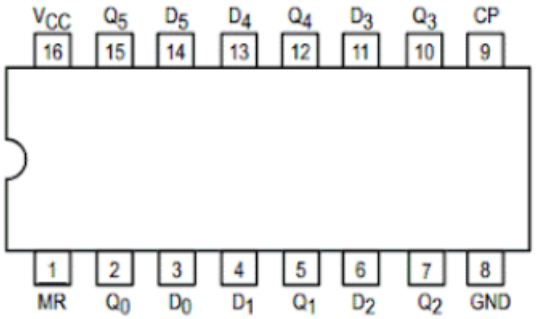
The purpose of this flip-flop IC is to assist in transmitting the address line value that the RAM is reading or writing from by syncing it up with the clock of the CHUMP.

References

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

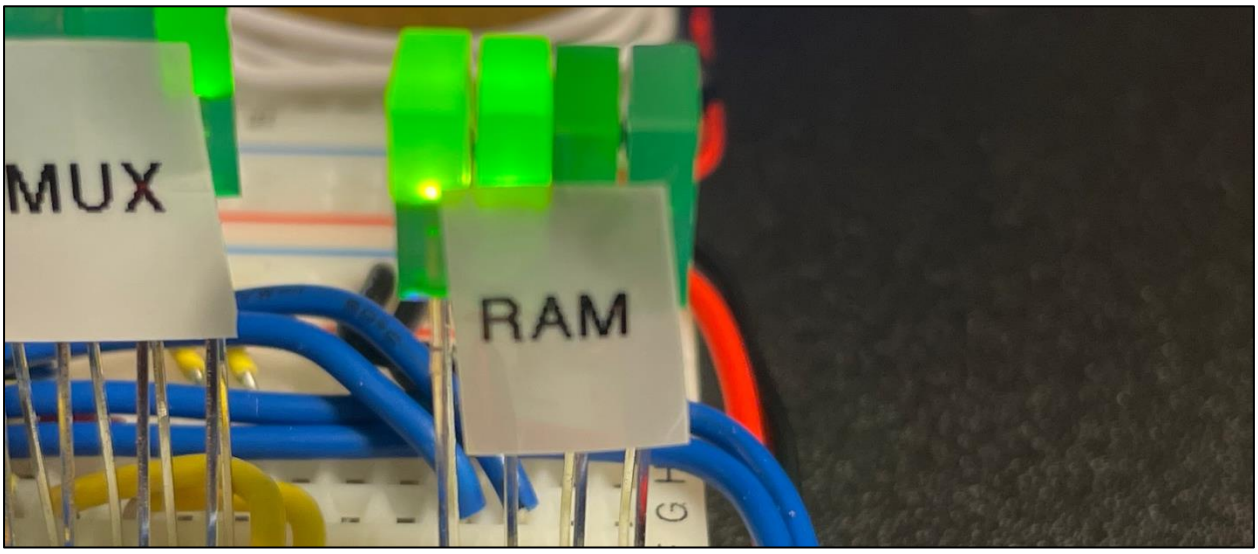
Procedure

The 74LS174 hex flip-flop IC is a 16 pin DIP that is functionally the same as the 74LS337 accumulator but contains 6 flip-flops instead of 8. It also does not have an enable but like the 74LS337 it has a clear and clock pin which is positively edge-triggered along with the regular D inputs and Q outputs.



In the CHUMP build four inputs are connected to the multiplexer output as the constant portion from the program EEPROM or RAM will specify the address in RAM to write or read for the STORETO and READ functions. As a reminder, these functions store data from the accumulator to a specified address of RAM or read data from the specified address of RAM. This is why this IC is referred to as the address IC and why its inputs are connected to the output of the multiplexer. The Q outputs are then connected to the address lines of RAM to determine where the data is read or written to. Along with these four address lines, a read/write bit is needed to be synchronized as well to decide between the two. It is fed from the lowest bit from the control EEPROM to a D input and the Q output travels to the read/write pin on RAM. The explanation of this bit and other pins on the RAM will be discussed in the RAM section.

Random Access Memory (RAM)



RAM outputs read from a selected address

Purpose

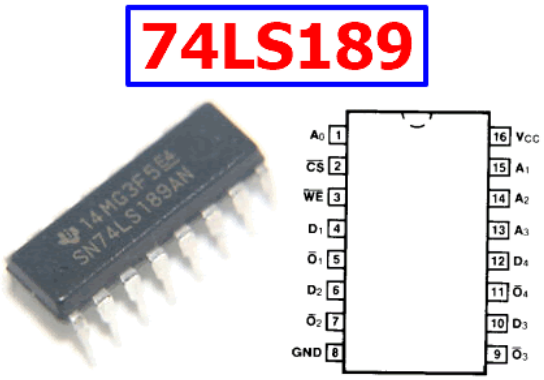
The RAM IC allows variables to be stored for later use in the program code. This is like using local and global variables in a code sketch which is necessary for more useful code.

References

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

Procedure

The 74LS189 RAM is a 16 DIP IC that houses 4-bit value in its 16 address registers. As a reminder, RAM stands for random access memory and unlike EEPROM it does not retain its contents after power is disconnected, therefore it is said to be volatile. This does not matter for the purpose it serves since the RAM is just a storage space for values that are used later in the chumpanese code and is not used for storing important permanent code. This specific 74LS189 chip is designed for high-speed reading and writing.



The reading and writing functionality is what the READ and STORE TO functions are based on. Reading a value requires an address to be selected in RAM through its address pins A0 to A3 and the data is presented as its complement on outputs O1 to O4. The complement of the data is simply the inverse where 1010 corresponds to 0101 on the output. Therefore, when reading a value from this RAM IC, the outputs must be passed through an inverter chip. Keeping with the TTL theme, a 74LS04 inverter was used which contains six inverters, four of which were wired to the outputs of RAM. The outputs of RAM are also open collector which means that a pull-up resistor is required on all outputs of RAM. Without it, the output pins will be left floating. These outputs once configured with pull-up resistors and after passing through the inverters are ready to be used by the ALU. But first, it must pass through the multiplexer inputs of B1 to B4 to output the value from RAM or the constant portion of the program EEPROM. That is why the multiplexer is there along with the IT variants of all the functions since it determines if the value is coming from RAM or the code constant. However, every IT variant of a function must be preceded by a READ function to load the value from the selected address. Hence, a READ function must contain a constant value afterward to select the address of RAM to read from. So, a READ 5 function will output the value of address 5 of RAM and present it to the multiplexer and ALU. Then using an IT variant of any of the functions, the ALU will use the value from address 5 as one of its variables instead of the constant defined in the code.

Before reading a value, data must be written to the addresses of RAM first. Since RAM is volatile, there must be a function in the chumpanese code that allows the storage of a value. This is the STORETO function. This utilizes the data input pins D1 to D4 and address select pins where data presented on these pins are stored to the address selected in binary using the address select pins. This is how STORETO works. It selects an address to store a value from the constant portion of STORETO. So STORETO 3 stores a value to address 3, presenting a binary value of 0011 on the address register and then on the RAM address select pins once the clock reaches a rising edge. As a reminder, all constant values are routed through the multiplexer but originate from either RAM or a constant in the program. The value being stored in the selected address is the value that is presented on the accumulator. Therefore the outputs are wired to the data inputs of RAM. Once a STORETO command is called, the value is written with the help of a read/write bit connected to the write enable pin producing a low.

Updated Chumpanese Code

Purpose

This serves to update and improved Chumpanese code to replace the previous version. It also covers uploading the EEPROM data from an Arduino

References

https://github.com/rsgcaces/AVROptimization/blob/master/AT28C16EEPROMShieldVersion/AT28C16EEPROMShieldVersion.ino

Procedure

The custom chumpanese code created in the first stages of CHUMP was discarded as it was inefficient and worked only when power was interrupted after the code was complete to erase RAM. Instead, a function that outputs the number of subtractions of operand B from operand A so that their differences 0 was created. This output is then infinitely looped on the accumulator. Calculating the function involves repeated subtraction of B from A and where A is updated with that value. A counter then counts the number of subtractions that have been performed. Eventually, the output will reach 0, at which point it will indefinitely loop with the accumulator presenting the number of subtractions it took, or the difference between A and B will become negative. Since the ALU is not set up to handle negative numbers, an output of -1 will loop back to 15 and the subtraction continues from 15. Eventually, the two operand differences will equal 0 or the pattern will loop indefinitely. An indefinite loop example would be 7 and 2. If 2 is constantly subtracted from 7 where -1 loops back to 15, the result will never equal 0 as the output will always show odd numbers. However, for 15 and 7, constantly subtracting 7 from 15 will yield 0 after nine subtractions. Calculating it by hand proves this true. This chumpanese code for this program is shown below with the example of 15 as the A operand and 7 as the B operand.

High Level	Machine Level		CHUMP (Assembly)	Comment
	Address	Instruction		
	0000 (0)	0000 0001	LOAD 1	accum←1, pc++
n = 1;	0001 (1)	0110 1010	STORETO 10	[10]←accum, pc++
X = 15	0010 (2)	0000 1111	LOAD 15	accum←15, pc++
while(x != 0){	0011 (3)	0100 0111	SUBTRACT 7	Accum-=7, pc++
x = x-7	0100 (4)	0110 0000	STORETO 0	[0]←accum, pc++
	0101 (5)	1110 1101	IFZERO 13	accum==0 ? pc←13 : pc++
	0110 (6)	1000 1010	READ 10	addr←10, pc++
	0111 (7)	0001 0000	LOAD IT	accum←[10], pc++
	1000 (8)	0010 0110	ADD 1	Accum+=1, pc++
n++	1001 (9)	0110 1010	STORETO 10	[10]←accum, pc++
	1010 (10)	1000 0000	READ 0	addr←0, pc++
	1011 (11)	0001 0000	LOAD IT	accum← [0], pc++
}	1100 (12)	1100 0011	GOTO 3	pc←3
	1101 (13)	1000 1010	READ 10	Addr←10
println(n)	1110 (14)	0001 0000	LOAD IT	addr← [10], pc++
while(true)	1111 (15)	1100 1111	GOTO 15	pc←15

To select the different A and B operands an 8-pin DIP switch was wired to the A4 to A10 address pins of the control EEPROM, taking advantage of the large storage size of the AT2816C. Since these are the remaining seven address pins, seven pins on the DIP switch are used where the highest four bits select A and the lowest three select B. Therefore, a maximum value of 15 for A and 7 for B can be computed. To compute different A and B operands, the DIP switches are selected and the reset button is pressed. This functionality of choosing combinations of numbers required writing an additional 128 sketches with different load values corresponding to the binary value present on the DIP switch bank. So by selecting an A value and a B value to compute, the program EEPROM selects a specific 4-bit program out of the 128 that loads the chosen A operand and subtracts B from the chosen A operand. Here are some examples of outputs from different A and B inputs. Double-checking the math by hand proves the outputs to be true:

Operand A	Operand B (subtract operand)	Output
10	2	5
14	13	6
6	9	6
4	3	12

Programming the EEPROM with all 128 sketches required creating an Arduino sketch to do so. However, it does not yet work as the program EEPROM would not flash correctly. If time permits in the future, it will be fixed as it would be useful to take advantage of the large EEPROM space. The current state of the sketch is shown below:

Code (Arduino C)

```
// PROJECT  :
// PURPOSE  :
// COURSE   :
// AUTHOR   : Xander Chin
// DATE     :
// MCU      :
// STATUS   : Not Working
// REFERENCE:

#define DFLT  0xFF     //default byte contents for EEPROM write buffer
byte codeWrite[16] = { DFLT, DFLT, DFLT, DFLT, //storage for code written
                       DFLT, DFLT, DFLT, DFLT,
                       DFLT, DFLT, DFLT, DFLT,
                       DFLT, DFLT, DFLT, DFLT
                     };

byte codeRead[16] = { 0, 0, 0, 0, //storage for EEPROM read buffer
                      0, 0, 0, 0,
                      0, 0, 0, 0,
                      0, 0, 0, 0
                    };

#define IO0          2
#define IO1          3
#define IO2          4
#define IO3          5
#define IO4          6
#define IO5          7
#define IO6          8
#define IO7          9

#define EEPROM_OE   10    // AT28C16 Output Enable
#define EEPROM_WE   11    // AT28C16 Write Enable

#define SER         16
#define RCLK        17
#define SRCLK       18

#define PROG_SIZE sizeof(code)

uint8_t operandA = 0;
uint8_t operandB = 0;

bool codeDIP = false;
byte code[] = {
  B00000001,    //load 1
  B01101010,    //storeto 10
  B00000000,    //load A
  B01000000,    //subtract B
  B01100000,    //storeto 0
  B11101101,    //ifzero 13
  B10001010,    //read 10
  B00010000,    //load it
  B00100001,    //add 1
  B01101010,    //storeto 10
  B10000000,    //read 0
  B00010000,    //load it
  B11000011,    //goto 3
  B10001010,    //read 10
  B00010000,    //load it
  B11001111,    //goto 15
};
```

```
void setup() {
  Serial.begin(9600);

  pinMode(SER, OUTPUT);
  pinMode(RCLK, OUTPUT);
  pinMode(SRCLK, OUTPUT);

  pinMode(IO0, OUTPUT);
  pinMode(IO1, OUTPUT);
  pinMode(IO2, OUTPUT);
  pinMode(IO3, OUTPUT);
  pinMode(IO4, OUTPUT);
  pinMode(IO5, OUTPUT);
  pinMode(IO6, OUTPUT);
  pinMode(IO7, OUTPUT);

  pinMode(EEPROM_WE, OUTPUT);
  pinMode(EEPROM_OE, OUTPUT);
  digitalWrite(EEPROM_WE, HIGH);
  digitalWrite(EEPROM_OE, HIGH);

  if(codeDIP) {
    for(uint8_t a = 0; a < 16; a++) {
      code[2] = a + (0b0000 << 4);       //LOAD A
      for(uint8_t b = 0; b < 8; b++) {
        code[3] = b + (0b0100 << 4);     //SUBTRACT B

        for(uint8_t addr = 0; addr < PROG_SIZE; addr++) {
          codeWrite[addr] = code[addr];
        }
        for(uint8_t addr = 0; addr < 16; addr++) {
          writeEEPROM(addr + (b<<4) + (a<<7), codeWrite[addr]);
          Serial.print(addr + (b<<4) + (a<<7));
          Serial.print("             ");
          Serial.println(codeWrite[addr], BIN);
        }
      }
    }
  }

  for(uint8_t addr = 0; addr < PROG_SIZE; addr++) {
    codeWrite[addr] = code[addr];
  }
  for(uint8_t addr = 0; addr < 16; addr++) {
    writeEEPROM(addr + (operandB<<4) + (operandA<<7), codeWrite[addr]);
    //Serial.print(addr + (operandB<<4) + (operandA<<7));
    //Serial.print("             ");
    //Serial.println(codeWrite[addr], BIN);
  }

  Serial.println("Reading EEPROM");
  printContents();
}

void writeEEPROM(uint16_t address, uint8_t data) {
  digitalWrite(EEPROM_OE, LOW);
  digitalWrite(EEPROM_WE, HIGH);
  setAddress(address);  //Set the address
  digitalWrite(EEPROM_OE, HIGH);
```

```
  for (int pin = IO0; pin <= IO7; pin++) {. //prepare to write the data...
    pinMode(pin, OUTPUT);
  }
  delay(10);
  for (int pin = IO7; pin >= IO0; pin--) {  //write the data...
    digitalWrite(pin, data & 0x80);
    //    if (pin == EEPROM_D7 - 4) Serial.print(' ');
    Serial.print(data & 0x80 ? HIGH : LOW);
    data <<= 1;     //destructive....
  }
  Serial.println("");
  delay(10);

  digitalWrite(EEPROM_WE, LOW);
  delayMicroseconds(100);
  digitalWrite(EEPROM_WE, HIGH);
  delay(10);
}

void setAddress(uint16_t address) {
  digitalWrite(RCLK, LOW);
  shiftOut(SER, SRCLK, MSBFIRST, address>>8);
  shiftOut(SER, SRCLK, MSBFIRST, address);
  digitalWrite(RCLK, HIGH);
  delay(10);
}

byte readEEPROM(int address) {
  digitalWrite(EEPROM_WE, HIGH);
  for (int pin = IO7; pin >= IO0; pin--) {
    pinMode(pin, INPUT);
  }
  byte value = 0;
  setAddress(address);
  digitalWrite(EEPROM_OE, LOW);
  delayMicroseconds(1);
  digitalWrite(EEPROM_OE, HIGH);
  for (int pin = IO7; pin >= IO0; pin--) {
    // if (debug) Serial.print(digitalRead(pin));
    value = (value << 1) + digitalRead(pin);
  }
  //  if (debug) Serial.print(" ");
  return value;
}

void printContents() {
  // Zero out the target for the reading of EEPROM just to be sure...
  for (int address = 0; address < 16; address++)
    codeRead[address] = 0;
  for (int address = 0; address < 16; address++){ //Separate the reading from thedisplaying...

    codeRead[address] = readEEPROM(address);
    Serial.print(codeRead[address], HEX);
    Serial.print(' ');
  }
}

void updateShiftTo(uint16_t data) {
  digitalWrite(RCLK, LOW);
  shiftOut(SER, SRCLK, MSBFIRST, data);
  shiftOut(SER, SRCLK, MSBFIRST, data);
  digitalWrite(RCLK, HIGH);
}

void loop() {}
```

To upload the control EEPROM codes, a sketch from my teacher Mr. D'Arcy was used. It is linked in the references. The problem with that code was that the control code for load constant was written as A8 rather than A9. This would have the effect of writing a value to the addresses shown on the mux to RAM. This messed with my custom code and took a bit of thinking to figure out the problem. Once changing it back to A8, the code worked well.

Media

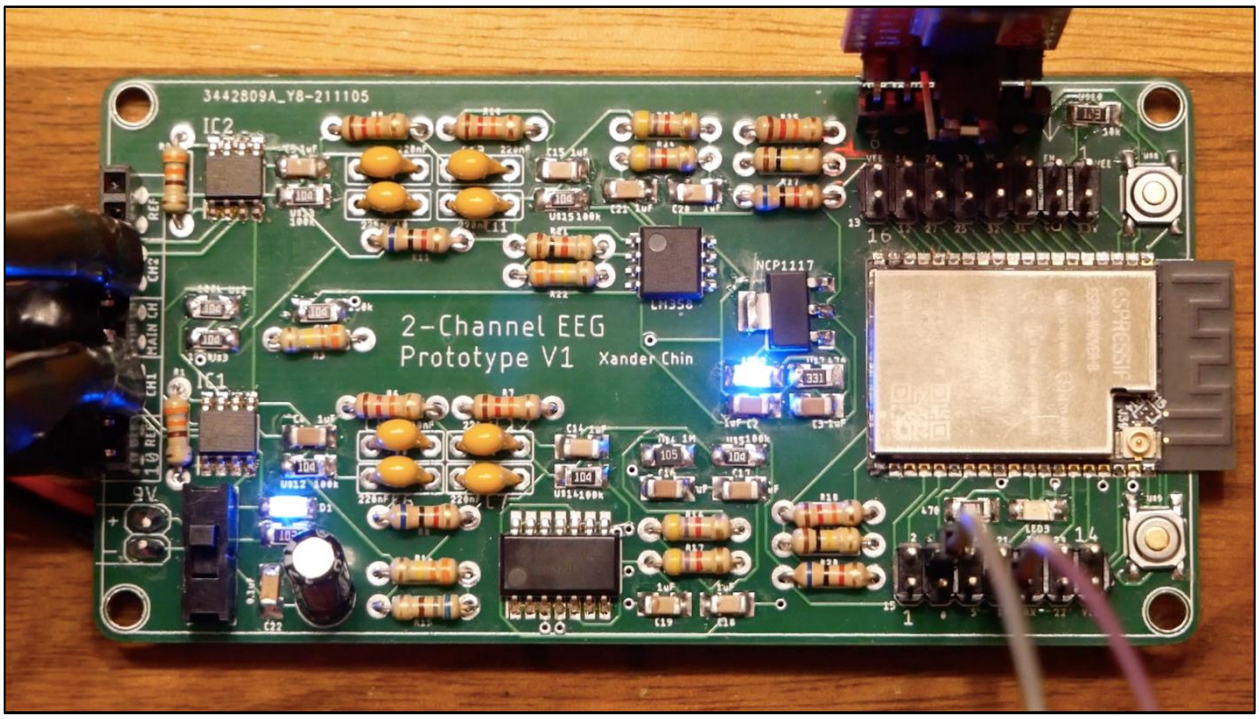
YouTube video link: https://youtu.be/fg3poC2dT2o

Reflection

The CHUMP was a harrowing, painstaking and drawn out process combining software, hardware, planning and patience to the deepest levels of understanding and implementation. Throughout the build, I faced countless problems. For example, I faced a problem with the clock signal where one of the 555 timers was malfunctioning. After double-checking the whole circuit, the problem was found and the timer was replaced. Another was not properly checking the control EEPROM codes where I scratched my head for at least two hours as to why the RAM was not storing values correctly. Also there was the trouble of creating new Chumpanese code as the one in my last submission was not great. Finally, there was wiring. To make a long story short, I will make sure the wires are all the way in the breadboard. There were definitely some other smaller technical problems as well but I have forgotten most of them. All I know is that time certainly flies when trying to fix a problem. And, it doesn't help when I get frustrated, especially with a looming deadline approaching plus other summatives from other classes plus the short ISP due next week. This leads into the non-technical problems I experienced. Before discussing those, I said in my last submission that I wanted to try and create manim animations. Now, I realize it was such an unimportant thing to do and with the work that has piled up, I never once thought about it. Maybe next time. Over the weekend though, I had to tell myself when to stop and to focus on the priorities. Once I immerse myself in a problem, even if it is not important or there are other workarounds available, I cannot let go of it until it is solved. This week has been trying to let go of that perfectionist thinking. An example would be trying to select different programs using the DIP switch. After working on it for a couple of hours, I painfully forced myself to stop to continue with the more important priority of making my video. And I do realize that this video was not the best one I've made and I hope that it is ok. The reason was because I focused more on my DER and CHUMP build. Also, I have trouble making unscripted videos since I usually make videos with a script with additional editing to the footage.

The other problem I faced was not being diligent and careful enough. For example, not taking notes and understanding the CHUMP when the teacher talked about it in class. This had a huge consequence as I had to take additional time to understand it when building leading to rewiring my CHUMP several times. On the positive side, I realized my mistake and sought out to start my DER early and wire the CHUMP a week before the deadline. With it, I wired up the chump systematically and in steps while testing it every so often. This method worked but even so, I am still scrambling to get my DER submitted on a Saturday evening. This goes to show that the CHUMP is no joke and really is the hardest project submission so far. Even though I put in a lot of work, more work than any regular project, it fell short of my expectations and there are some aspects of the submission to be improved. But, it was a good experience to learn about the complex architecture that goes into creating a simple 4-bit computer. However, next time, I think I wish I could have put in less work as I have neglected a lot of other priorities. Also I find it funny is that it is harder for me to stop working on any problem than it is to abandon it. I hope the severity of this will change over time. Anyway, It's been a hectic week and will become even more as the ISPs are due next. I am not too sure what will happen.

Project 3.4 (ISP – Short): EEG and EMG Mind Control Headset



Purpose

The purpose of this chosen ISP was to explore the biomedical and neurological field of engineering through the creation of an electroencephalography (EEG) reader. This would also enable the user to "mind control" devices with the output of the reader.

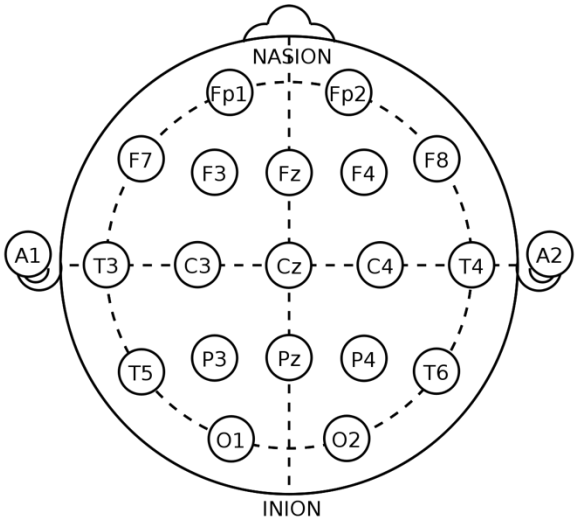
Theory

The theory and basic rundown of the project are as follows. An EEG reader would pick up brain signals corresponding to certain activities and using those waves, certain devices can be controlled. The EEG reader consists of electrodes connected to the scalp and forehead, picking up signals, which would then be filtered and amplified through a variety of resistor-capacitor circuits, yielding a clean EEG signal. A microcontroller would then read these signals and transmit them wirelessly to a computer where a graph of the data would be displayed. To make sense of this data, a Fast Fourier Transform (FFT) algorithm is employed to convert a position-time domain signal into a frequency domain, separating the different sine waves that correspond to brain activity. Using this data, the computer can send a signal to a device, controlling it through the data gathered from EEG. Since EEG is somewhat controllable, this would enable a form of limited mind control.

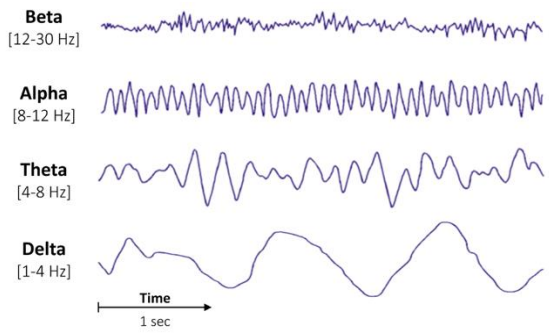
The reality of this project turned out to be not as impressive. Currently, four electrodes are attached to the user's head. It then transmits mainly Electromyography (EMG) data which is muscle activity and a little bit of EEG data. This data is then filtered and amplified through a PCB circuit and is outputted through a wired connection to a computer. The computer graphs this data and runs it through an FFT to scan for EEG signals. Certain waves like alpha waves can be easily detected and the EMG is noticeable and controllable through blinks, eye movement, eyebrow raises, and other muscle activities so it was planned to implement machine learning to classify each EMG reading so that the user can control devices using EMG. But, due to time constraints, this was not implemented.

EEG and EMG

The basis of the project lies in EEG which are waves that transmit brain activity. It is used in hospitals for anesthesia, detecting brain abnormalities, and assessing sleep. The best way to acquire brain activity is through invasive techniques where electrodes are placed directly on the brain, producing a very clear and detailed signal. Unfortunately, this is not feasible in many situations as invasive techniques require surgery and anesthetics so instead a non-invasive was implemented. This method comprises electrodes placed in certain locations around the brain, allowing an easy non-invasive alternative at the cost of a dampened and weak brain signal as they have to travel through the skull and skin. The locations of where to put the electrodes are based on the 10-20 system which is an internationally recognized method for placing electrodes to gather EEG data. It is called the 10-20 system due to the percentages of space that are left between each electrode.



Each electrode on this system corresponds to a certain part of the brain responsible for a certain activity. They are labelled with letters identifying the parts of the brain along with a number of the letter z where odd numbers occupy the left side, even numbers occupy the right and the z letter is for the midline of the skull. For example, the pre-frontal cortex labelled Fp1 and Fp2 are responsible for personality, speech control and planning. The occipital lobe labelled O is primarily used for visual processing. These two areas are used in the EEG acquisition device.

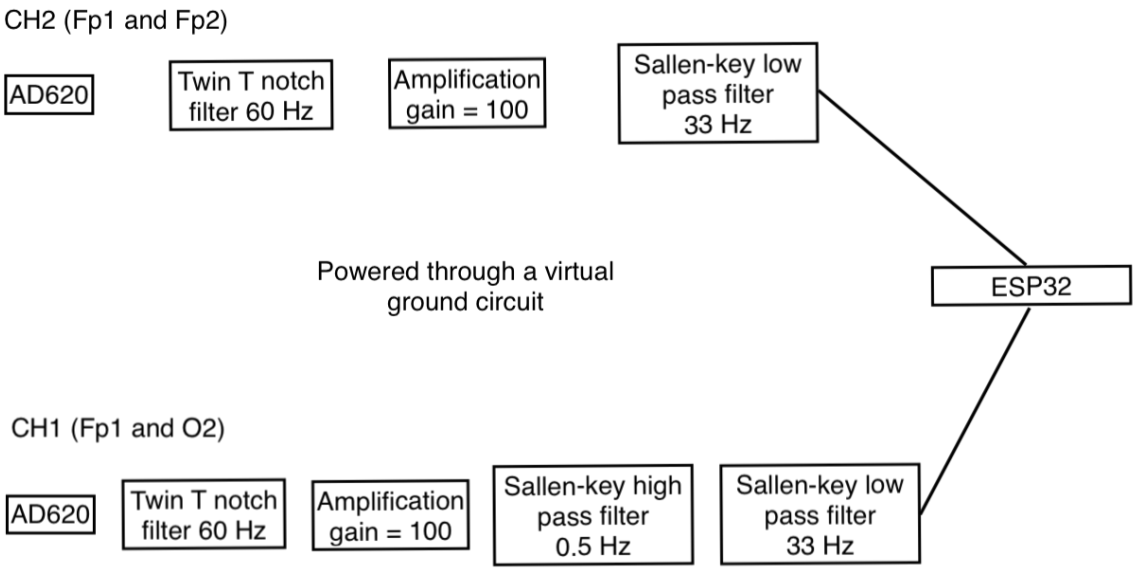


The EEG gathered is in the form of sine waves and each part of the brain produces different magnitudes and frequencies of waves that can be detected albeit the magnitudes are in the orders of microvolts as the EEG must travel through the skull and skin. The most common types picked up in other EEG acquisition projects are beta, alpha, theta, delta, mu and gamma waves. Beta waves are characteristic of alertness or focus and are evident by 13-30 Hz waves while alpha waves are linked to relaxation and occur in the 8-14 Hz range. Theta and delta waves are similar since they measure drowsiness and have the most magnitude and occur in the 1-8 Hz range. The last wave, mu, measures motor neurons at rest where mu suppression suppresses these waves that also occur at the 8-13 Hz range. Each wave has more magnitude at certain locations in the 10-20 system. For example, mu waves are the strongest at the central sections of the 10-20 system whereas alpha waves are detected near the occipital lobe. Since the occipital lobe deals with vision, closing one's eyes leads to more prominent brain waves. Aside from different wave frequencies, there are other specific patterns in the waves that can show up such as pattern recognition triggering and intense visual processing.

With the acquisition of EEG, there is EMG from facial muscles that corrupts EEG data called artifacts. Prominent examples include blinking, forehead, mouth and eye movement. These show up as deviations and dips in the signal. In a medical situation, these artifacts are problematic as they interfere with the delicate EEG signal and are usually filtered out through software. However, for hobbyists trying to use these signals to control devices, these can prove useful as they can easily be identified, controlled, and require fewer components for acquisition. This is why such signals are not filtered out in my EEG device. In spite of that, the project is both an EEG and EMG acquisition tool. Examples of each EMG artifact are shown in the media section.

In the project, electrodes were only placed on the forehead where the two pre-frontal sections are and one placed on O2 which is the occipital region. With this setup, alpha waves can be noticeably detected which would have allowed a user to control a device using these waves. There is also a reference electrode placed behind the ear that reduces the noise generated by the body. Only four electrodes were placed because it dramatically reduces the EEG acquisition circuitry, software, and cost as putting all 12 electrodes in correspondence with the 10-20 system is not feasible.

Hardware



References

https://tinyurl.com/y24tdkpt
https://tinyurl.com/y2db44h7

Procedure

Electrodes are one of the key components in getting EEG data since these brain signals are already heavily dampened from the skull. They help reduce the impedance from the skin to the circuit to keep the distortion to a minimum. There are two types of electrodes used, dry and wet. Dry electrodes are fitted with materials like clean metals that offer the least impedance but are still prone to faulty contact with the scalp. Such contact increases resistance that the signal must travel through which leads to a really messy and inaccurate recording. This is why good electrodes and electrode placement are crucial. Wet electrodes offer less impedance but are not reusable or need gel to help improve the signal.

For this project, disposable wet electrodes were chosen because they were readily available and are known to produce better results. Since they are also sticky, the glue was taped over when attaching an electrode to the occipital region of the head as the vicinity contains hair. This hair does, unfortunately, hinder the connection so the reading is not the clearest.

The electrodes can then be snapped onto electrode cables. The cables that were used were shielded to absorb and prevent the signal from the corruption of electrical activity through, unfortunately, these cables do not prevent signals generated from cable movement. At the end of the cable, the wire is routed into a DIN connector and since none were available or able to interface with my circuit, the cable had to be stripped and soldered to male pin headers. In total, four cables were needed, one for each electrode.

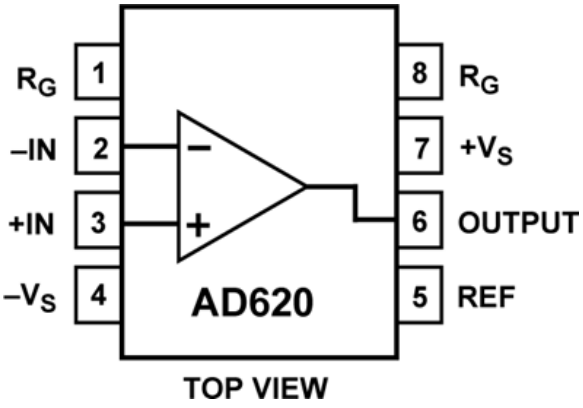
These cables then plug into the amplifier and filter circuit. This circuit is absolutely necessary to achieve a good reading as it amplifies the EEG signals from microvolts to volts for a computer to pick up and also filters noise generated by components and other devices. This is especially important because the signal is being amplified.

The circuit design that was chosen was based on general guidelines, forums, project websites, and my own experiments. In the end, the circuit consists of an instrumentation amplifier, a notch filter, an amplification stage, a sallen-key high and low pass filter, and then through a resistor divider circuit so that the ESP-32 microcontroller can read the values. The order written is the order of circuits that the signal passes through. Along these stages, there are also some simple RC high pass filters sprinkled around that only consist of a capacitor and resistor connected along the signal pathway and ground. All this is powered through a 9V battery split into a dual-rail power supply.

The circuit involves a lot of op-amps or operational amplifiers so here is the basic working of them. Op-amps work as amplifiers of two signals or can output the sum or difference through external analog components of capacitors, inductors and resistors. They require positive and negative voltages to be supplied and they also have a few key properties. More information is provided in the references section.

The first part of the circuit design is the 8-pin AD620 instrumentation amplifier which is basically a more professional differential amplifier. In essence, the instrumentation amplifier takes in two signals and outputs and amplifies the difference between them. With the AD620, a lot of the problems that occur with the three op-amp differential amplifier setup are reduced dramatically. So, its high performance regarding a high common-mode rejection ratio and low offset voltage and noise combined with a somewhat low cost makes this chip popular in both medical and hobbyist EEG devices. This performance is also due to the reference pin which takes in a reference electrode signal to get rid of the voltage offset produced by the body. More data is provided in the AD620 datasheet. The gain of the difference can also be set with resistors. In the project, there is a gain of around 150 set by a 330-ohm resistor spanning the gain pins. The equation for this gain is shown here:

$$G = \frac{49.4\ k\Omega}{R} + 1$$



In the project, two AD620s are used, therefore, the EEG device can measure two channels. It takes in four inputs, two of which are from the same electrode and the other two are from different electrodes. The same electrode is placed on the Fp1 region in the 10-20 system and the two different electrodes are placed on the O2 and Fp2 regions. Therefore, the first channel measures the brainwave difference between Fp1 and O2 for the acquisition of alpha waves and the second channel measures the signal difference between Fp1 and Fp2 mainly for EMG recordings. Right vs left thinking also have the potential to be detected with this electrode placement. Each AD620s reference pin is then connected to the single reference electrode placed behind the ear.

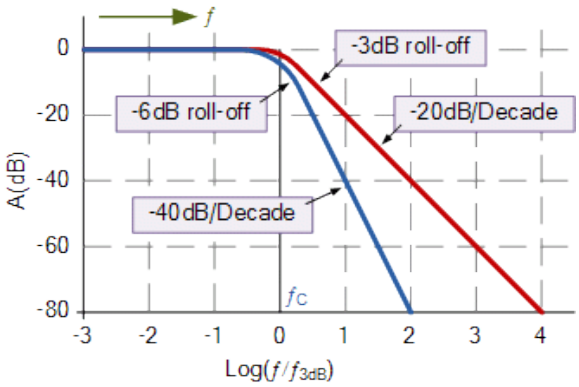
Next is the notch filter that only filters out 60 Hz electrical signals, meaning sine waves corresponding to a frequency of 60 Hz are eliminated. This 60 Hz noise is prevalent everywhere with electronics and is especially frequent in hospitals where a lot of specialized electronic equipment is always on, so it is pivotal to include some sort of method for getting rid of it as it interferes heavily with the data. In really expensive EEG devices, the circuitry is complex and specialized software is used to further tone down the noise, but for this project, a simple notch filter does the job. The notch filter used is called a twin T notch filter and it consists of capacitors and resistors in RC pair formations.

For a chosen frequency of 60 Hz, an R value of 6 Kohm and C value of 220 nF were chosen. In the circuit, 6 kΩ and 12 kΩ resistors were used along with a ceramic capacitor with the chosen C value of 220 nF. To get the value of 2C or 440 nF, two 220 nF capacitors were put in parallel to double the 220 nF. A simulation demonstration of the twin t notch filter filtering 60 Hz noise is shown as a link to a Falstad circuit.

Other EEG devices typically use other types of notch filters, as none were found that used this circuit. So, it was decided upon to try it out due to its simplicity and test its effectiveness. Overall, it seemed to work well and caused no issues.

The next major part of the circuit is the amplification stage used to amplify the brain signals. This next stage was chosen to be placed after the notch circuit so that the 60 Hz noise will not be amplified and the other noise present can be more effectively toned down with the high and low pass filters ahead. Plus, the notch filter reduces and varies the amplitude of the signal so placing the amplification circuit after keeps the signal at a constant level. The amplification stage is a standard op-amp amplifier that uses external resistors for a voltage divider. This divider is responsible for the amount of amplification so the resistor values give the formula for gain.

A gain of 330 and 485 was chosen for the first and second channel respectively. The difference of gain values was chosen because the EMG recorded on the second channel was too low in amplitude so the gain was adjusted for that channel only. To set the gain of 330, resistor values of 330 kohm for R1 and 1 kohm for R2 was used and to set the gain of 485, 330 kohm was also used for R2 but R2 was instead 680 ohms. It should also be noted that all op-amps used come from the LM324 14 pin quad op amp and the LM358 dual 8 pin op amp.



Once amplified the signal passes through a high pass and low pass filter set to frequencies that capture the common EEG signal frequencies. The high pass filter was set to pass frequencies above 0.5 Hz. Also, with the high pass filter, the signal apparently stays relatively close to the midpoint rather than vary its equation of axis which is important for analyzing EEG. The low pass filter was set to pass signals below 34 Hz. With both these filters, values set between 0.5 Hz and 34 Hz, which is the optimal range for EEG signals, were kept intact while those outside that range are attenuated and fall off to 0. Because of this, some other EEG projects forego the notch filter as the high and low pass filters mostly take care of the noise. But, it was decided to keep it in to further the chances of producing a clear signal. The types of high and low pass filters used are called Sallen-Key filters which are 2nd order filters based around an op-amp. 2nd order means that the attenuation cut-off slope is higher than a simple RC 1st order filter, decreasing the noise present more effectively.

Values chosen for the filters are 1 uF non-polar capacitors and 1 megaohm and 100 kohm resistors for the high pass and 1 uF non-polar capacitors and 4.7 kohm resistors for the low pass. The equation to determine these frequencies of Sallen-key filters can be calculated using this formula:

$$f = \frac{1}{2\pi * R1 * C1 * R2 * C2}$$

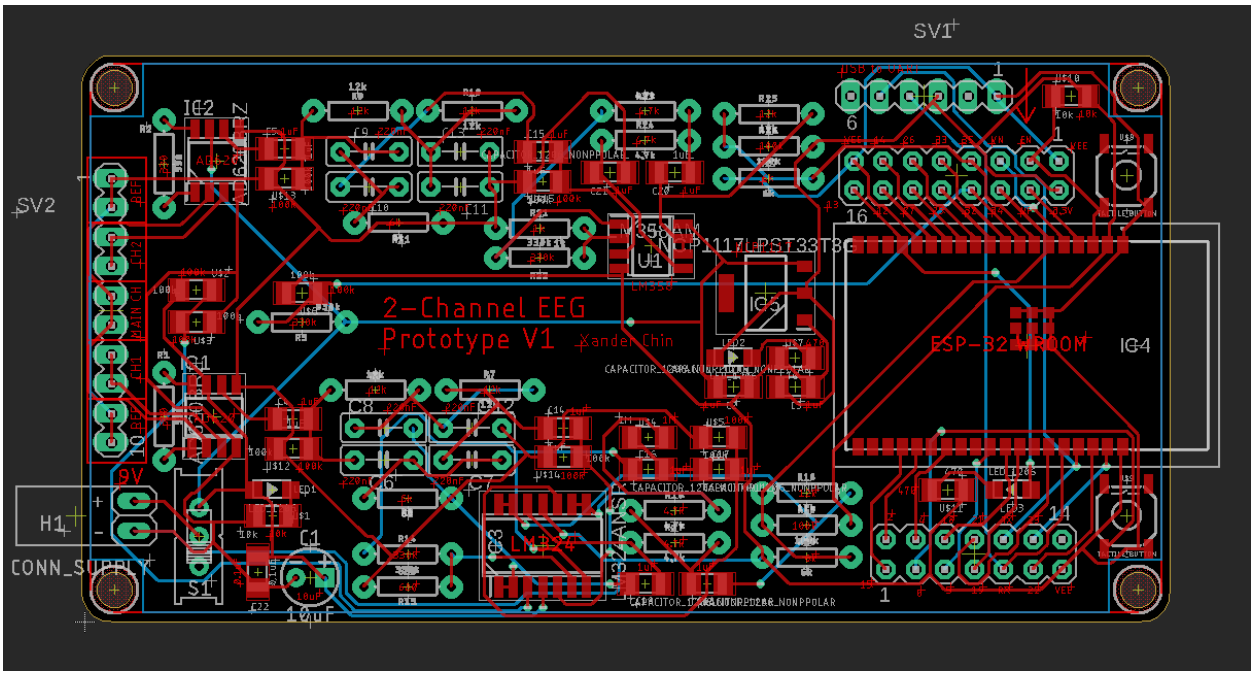
In the project, the first channel contains a Sallen-key high and low pass filter while the second channel only contains the low pass filter.. This hardware choice was implemented because the high pass filter was not as important for acquiring data, especially if the second channel is mostly reserved for detecting EMG. Also, there was little room left to fit all the components on the PCB. To ompensate, both channels have passive RC high pass filters along the way set to pass frequencies above 0.15 Hz through resistor values of 1 megaohm and capacitance values of 1 uF.

Finally, a voltage divider type configuration was used to scale down the voltage to a readable value under 3.3 volts so that the ESP32 can read it. This voltage configuration was found in a blog post and was experimented with and configured to reduce peaks of 9 volts to a maximum value of 3.3 volts with respect to 0 volts as the ground. A Falstad simulation of this is linked in the references section.

As mentioned before, the whole circuit is powered by a 9 volt battery. These 9 volts are split into 9 volts, 4.5 volts and 0 through a voltage divider formed from two 100 kohm resistors and the 4.5 volts is passed through an op-amp with a gain of 1. This is called a dual rail power supply because the 4.5 volts now acts as the ground so that the 9 volts and 0 volts are transformed to 4.5 and -4.5 volts respectively. Therefore, all circuits that are connected ground are actually connected to 4.5 volts and the pins designated to VCC are connected to 9 volts and those designated to VEE are connected to 0 volts. This is one way of how negative voltage can be supplied to op-amps simply by using 4.5 volts as ground.

Unfortunately, the 4.5 volts is not enough to power the ESP32 using the 5V pin as input and is too much to directly bypass the 3.3 voltage regulator. So, instead of using the 4.5 volts as ground and powering with the 9 volts, supplying a total of 4.5 volts, the ground for the ESP32 was chosen to be 0 volts and the power as 9 volts. In reference to the op-amp parts of the circuit, the ESP32 is grounded to a –4.5 volt signal and powered with a 4.5 signal. This brings in two different grounds and so there was a lot of caution involved in making sure that no reverse voltage or overvoltage was present on the ESP32. With those considerations in mind, the signals from the two channels are now ready to be read through software.

Design

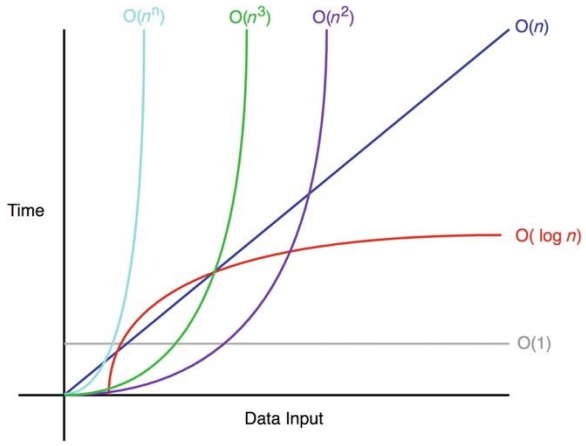


The design aspect of the project was mainly about the PCB. The PCB was split into the two channels where the circuits of one channel were kept on the bottom while the other circuits for the second channel were kept on top. This reduced signal traces from crossing over from one channel to the other which may interfere with the delicate signal of EEG. A mix of SMT components and through hole were used for simplicity purposes as many specific resistors were needed. Since some were available already in through hole components, it was decided to just use them in the PCB.

To solder the SMT components, a mix of hand soldering for the 1206 components and baking for the more detailed components like the ICs and the ESP32. Baking was a new experience and it involved putting solder paste on the pads either by hand or with a stencil and putting them in an oven like appliance that heats up the board and melts the solder. This method is quite simple, fail-safe and effective.

Although the gathering of EEG and EMG is messy, complicated and still prone to inaccuracies, it is still impressive that a small degree of brain activity can be read and deciphered, especially since the brain is locked up behind high impedance skin and a thick skull. It is also a step towards more sophisticated brain computer interfaces for those who cannot communicate verbally or those who are paralyzed.

Mathematics



The mathematics behind EEG acquisition is very complex and involves some knowledge on calculus. The math mainly consists of the Discrete Fourier Transform (DFT) or the Fast Fourier Transform (FFT) which transforms a signal made up of sine waves of different magnitudes and frequencies into its frequency components. The FFT is widely used and is one of the most ingenious algorithms and has many applications in image, sound, and video processing along with others. The difference between the two is that the DFT is much slower than the FFT in terms of the time to complete the number of operations. The DFT has a quadratic running time function where as the number of inputs increases, the number of operations required increases quadratically. This is denoted by big $O(n^2)$. The FFT on the other hand has $O(n \times \log n)$, so as n gets very large, the scaling will almost be linear which is a huge improvement from a quadratic scaling. This is due to recursions of the function that reduce the matrix multiplication involved down to simpler matrices. In fact, in the processing sketches provided in the software sections, there is a DFT sketch and an FFT sketch both transforming the same signal in processing. However, as n increases, there is a noticeable increase in time difference between the two sketches.

To facilitate more understanding of the DFT and FFT, complex numbers need to be understood as these algorithms deal with them. Also, some knowledge on integrals helps. In the references section, there are a multitude of resources that helped me learn about the DFT and FFT.

Media

YouTube video link: https://youtu.be/Dr2lxEIa05U

Software

Here are all the sketches used in the project. The code on them is currently quite inefficient as readability and understanding was prioritized as that was needed when learning and implementing the FFT.

ESP32 Serial

```
// PROJECT  : ESP32 EEG Headset
// PURPOSE  : Short ISP
// COURSE   : ICS4U
// AUTHOR   : Xander Chin
// DATE     : Dec 4, 2021
// MCU      : ESP32
// STATUS   : Working
// REFERENCE:

#define EEGPIN1 34
#define EEGPIN2 35

String command;
bool state;

void setup() {
  Serial.begin(115200);
  Serial.setTimeout(1);
  pinMode(4, OUTPUT);
  pinMode(19, OUTPUT);
}

void loop() {
  uint16_t sensorValue1 = analogRead(EEGPIN1);
  uint16_t sensorValue2 = analogRead(EEGPIN2);
  Serial.println(String(sensorValue1) + "," + String(sensorValue2));

  if(Serial.available()) {
    command = Serial.readString();
    if(command.equals("blink")) state = !state;
  }

  digitalWrite(4, state);
  digitalWrite(19, !state);
}
```

Graphing EEG (Processing)

```
import processing.serial.*;
Serial myPort;

byte rWidth = 5;
String serialData = "";
int y1;
int y2;

int x = 0;
int lastShift;
int lastSample;

int[] data1;
int[] data2;
float[] scaleData1;
float[] realResult;
float[] pixHeight;

float[] window;
float[] windowNone;         //array for no window
float[] windowFlat;         //array for flat top window
float a0 = 0.21557895;      //data for flat top window
float a1 = 0.41663158;
float a2 = 0.277263158;
float a3 = 0.083578947;
float a4 = 0.006947368;
float[] windowHann;         //array for hanning window
String windowLabel;

int N = 512;        //number of samples of the FFT
int sampleN = 0;
boolean computeFFT = false;

float[] frequencyBins;      //frequency outputs of the FFT
Complex[] output;           //

boolean blink = false;      //detect blinks
int blinkCounter = 0;

//can be swapped for 2D vector class
//created for understandability
class Complex {
    float real;   // the real part
    float img;    // the imaginary part

    public Complex(float real, float img) {
        this.real = real;
        this.img = img;
    }

    public float real() {
      return this.real();
    }

    public float img() {
      return this.img();
    }

    void describe(){
      println( "" + this.real + " + " + this.img + "i" );
    }
}
```

```
void setup() {

  String portName = Serial.list()[5];    //port corrsponding to esp32 serial port
  //println(Serial.list());
  myPort = new Serial(this, portName, 115200);
  myPort.bufferUntil('\n');

  size(1300, 800, P2D);
  data1 = new int[width];      //define arrays
  data2 = new int[width];
  scaleData1 = new float[N];
  realResult = new float[N];
  pixHeight = new float[scaleData1.length];

  frequencyBins = new float[10];

  window = new float[N];
  windowNone = new float[N];
  windowFlat = new float[N];
  windowHann = new float[N];
  for(int x = 0; x < N; x++) {    //calculate windows
    windowNone[x] = 1;
    windowFlat[x] = a0-a1*cos(2*PI*x/N)+a2*cos(4*PI*x/N)-a3*cos(6*PI*x/N)+a4*cos(8*PI*x/N);
    windowHann[x] = 0.5*(1-cos(2*PI*x/N));
  }
  window = windowNone;
  windowLabel = "No window";

  rectMode(CORNERS);
}

void draw() {
  background(0);

  stroke(255);
  //graph channel 1 data
  for (int i = 1; i < data1.length; i++) {
    line(width-i, height-data1[i], width-i+1, height-data1[i-1]);
  }
  //graph channel 2 data
  for (int i = 1; i < data2.length; i++) {
    line(width-i, height-data2[i], width-i+1, height-data2[i-1]);
  }
  //graph FFT of channel 1
  for (int x = 0; x < pixHeight.length; x++) {
    rect(x*10, height-pixHeight[x], (x+1)*10, height);
  }
  textSize(30);

  //labels for current window and blink counter
  text(windowLabel, 100, 50);
  text("blink counter: " + str(blinkCounter), 100, 90);

  //FFT calculations
  if(computeFFT) {
    Complex[] scaleSignalC = new Complex[scaleData1.length];
    for(int x = 0; x < scaleData1.length; x++) {
      scaleSignalC[x] = new Complex(scaleData1[x]*window[x], 0);    //multiply by window
    }
```

```
    Complex[] result = fft(scaleSignalC);    //calculate fft
    for(int x = 0; x < scaleData1.length; x++) {
      realResult[x] = sqrt(result[x].real*result[x].real + result[x].img*result[x].img);
//turn complex number into real number (magnitude)
      pixHeight[x] = map(realResult[x], min(realResult), max(realResult), 0, 5000);
//determine rectangle heights
      //println(str(x) + ": " + realResult[x]);
    }
    computeFFT = false;
  }

  //detects blinks
  if(data1[0] > 400 && blink) {
    blink = false;
    myPort.write("blink");
    blinkCounter++;
  } else if (data1[0] < 400) {
    blink = true;
  }
}

void serialEvent(Serial myPort) {
  //This section needs to be first so that serialEvent is not disabled
  serialData = myPort.readStringUntil('\n');
  serialData = serialData.substring(0, serialData.length() - 2);
  int[] splitData = int(split(serialData, ","));

  //shift data every one millisecond
  if(millis() - lastShift > 1) {
    for(int i = data1.length-1; i > 0; i--) {
      data1[i] = data1[i-1];
    }
    for(int i = data2.length-1; i > 0; i--) {
      data2[i] = data2[i-1];
    }
    lastShift = millis();
  }

  //map data to pixel heights for plottting
  data1[0] = int(map(float(splitData[0]), 0, 4095, 50, 600));
  data2[0] = int(map(float(splitData[1]), 0, 4095, 300, 900));

  //calculate FFT every 1 ms
  if(millis() - lastSample >= 1) {
    scaleData1[sampleN] = map(float(splitData[0]), 0, 4095, 0, 10);
    sampleN = (sampleN + 1) % N;
    if(sampleN == N - 1) computeFFT = true;

    lastSample = millis();
  }

  //println(splitData[1]);
  //println(serialData);
}

//use custom complex number class. returns complex number
Complex[] fft(Complex[] data) {
  int n = data.length;
  if(n == 1) return data;

  //divide into even and odd indexes
  int halfN = n/2;
  Complex[] even = new Complex[halfN];
  Complex[] odd = new Complex[halfN];
  for(int i = 0; i < halfN; i++) {
    even[i] = data[i*2];
    odd[i] = data[i*2+1];
  }
```

```
  //recursive
  even = fft(even);
  odd = fft(odd);

  output = new Complex[n];    //initialize output

  //after recursion - combine
  for(int k = 0; k < halfN; k++) {
    output[k] = new Complex(even[k].real + (odd[k].real*cos(2*PI*k/n) -
odd[k].img*sin(2*PI*k/n)),
                        even[k].img + (odd[k].real*sin(2*PI*k/n) + odd[k].img*cos(2*PI*k/n)));

    output[k+n/2] = new Complex(even[k].real - (odd[k].real*cos(2*PI*k/n) -
odd[k].img*sin(2*PI*k/n)),
                           even[k].img - (odd[k].real*sin(2*PI*k/n) +
odd[k].img*cos(2*PI*k/n)));
  }
  return output;
}

//unused but simpler than FFT
//takes in data array data[] - data is an array vector
//takes in N number of samples in data[]
//will return frequency components array vector
float[] dft(float[] data) {
  int N = data.length;
  int K = N/2;

  for(int k = 0; k<K; k++) {
    float realSum = 0;
    float imaginarySum = 0;
    for(int n = 0; n<N; n++) {
      realSum += data[n] * cos((2*PI/N)*k*n);
      imaginarySum += data[n] * sin((2*PI/N)*k*n);
    }
    //realSum = realSum*(1/N);
    frequencyBins[k] = sqrt(realSum*realSum + imaginarySum*imaginarySum);
  }
  return frequencyBins;
}

void keyPressed() {
  if(key == '1') {
    window = windowHann;
    windowLabel = "Hann window";
  } else if(key == '2') {
    window = windowFlat;
    windowLabel = "Flatop window";
  } else if(key == '0') {
    window = windowNone;
    windowLabel = "No window";
  } else if(key == 'C') {
    blinkCounter = 0;
  }
}
```

Neural Network Sketch (Processing)

```
//--------------for saving to files
import java.io.BufferedWriter;
import java.io.FileWriter;

FileWriter fw;
BufferedWriter bw;

String fileName = "signalData.txt";

//--------------getting data from serial
import processing.serial.*;

Serial myPort;
String serialData = "";

//-----------------other
int[] data1;        //channel 1 data
int[] data2;        //channel 2 data
int lastShift;

int mode = 0;       //0 = normal graphing
                    //1 = put red line
                    //2 = sort data
int marker1 = 0;    //marker 1 to 2: buffer
int marker2 = 0;    //marker 2 to 3: action
int marker3 = 0;    //marker 3 to end: buffer
int start;          //start of data to be saved
int end;            //end of data to be saved
String label = "nothing";       //classification of data
int labelNum = 1;               //number classification (goes into training file)

String[] savedData;
String fileText;

/*
object classes

network class

  attributes
  neuron - input, hidden, output layers

  methods
  respond()
  train()

neuron class

  learning rate (for all neurons)

  attributes

  neuron inputs - place to store outputs from previous layer
  output
  weights
  error (difference in desired and output)
*/
```

```
void setup() {
  //setup network (100 neurons, 50 hidden layers, 4 outputs)
  //setup sigmoid data
  //

  size(1300, 800, P2D);

  data1 = new int[width];
  data2 = new int[width];
  savedData = new String[101];    //extra 1 for classification name

  String portName = Serial.list()[5];
  //println(Serial.list());
  myPort = new Serial(this, portName, 115200);
  myPort.bufferUntil('\n');

   delay(1000);

}

void draw() {
  /*
  gather data
    graph eeg signals onto screen
    record button (or key)

    if press record button
      red line is marked on signal
      perform blink/action
      when record is pushed again, eeg signals stop
      lines show up on eeg signals that mark 0 to 50 on each signal (ch1 and ch2)  //10ms for
each point. 500ms in total
      classify each highlighted area as one of the outputs (nothing, blink left, blink right,
etc) use numpad
      can delete recording if not good
      can check through data collection by using arrow keys and seeing what segments are class
ified as
      save to a training file
  */

  //graph data
    background(0);

    stroke(255);

    //graph channel 1
    for (int i = 1; i < data1.length; i++) {
      line(width-i, height-(data1[i]+50), width-i+1, height-(data1[i-1]+50));
    }
    //graph channel 2
    for (int i = 1; i < data2.length; i++) {
      line(width-i, height-(data2[i]+400), width-i+1, height-(data2[i-1]+400));
    }
```

```
    if(mode != 0) {
      stroke(255, 255, 0);
      line(width-marker1, 100, width-marker1, 800);
      stroke(0, 255, 0);
      line(width-marker2, 100, width-marker2, 800);
      stroke(255, 0, 0);
      line(width-marker3, 100, width-marker3, 800);

      if(mode == 2) {
        stroke(255, 0, 255);
        line(width-start, 100, width-start, 800);
        line(width-end, 100, width-end, 800);

        //text
        textSize(50);
        text(str(labelNum) + ": " + label, 30, 120);
        fill(255);
      }
    }
}

void keyPressed() {
  if(keyCode == ' ') {
    mode = (mode + 1) % 3;
    if(mode != 2) {
      marker1 = 0;
      marker2 = 0;
      marker3 = 0;
    }
    //place red line in recording
  } else if(keyCode == 't') {
    //train model
  }

  if(mode == 2) {
    if      (keyCode == '1') {label = "nothing"; labelNum = 1;}
    else if (keyCode == '2') {label = "right blink"; labelNum = 2;}
    else if (keyCode == '3') {label = "left blink"; labelNum = 3;}
    else if (keyCode == '4') {label = "eye move right"; labelNum = 4;}
    else if (keyCode == '5') {label = "eye move left"; labelNum = 5;}
    else if (key == ENTER) {
      savedData[100] = str(labelNum);
      for(int x = 0; x < 50; x++) {
        savedData[x] = str(data1[start-x*5]);
        savedData[x+50] = str(data2[start-x*5]);
      }
      fileText = join(savedData, ",");
      appendTextToFile(fileName, fileText);

      start = start-5;
      end = end-5;
    }
    else if(key == CODED) {
      if(keyCode == LEFT) {
        start = start+5;
        end = end+5;
      }
      else if(keyCode == RIGHT) {
        start = start-5;
        end = end-5;
      }
    }

    else if(keyCode == 'T') {
      label = "training...";
      trainNetwork();
      label = "nothing";
      labelNum = 1;
    }
  }
```

```
  //if key pressed:
    //load training data
    //go train model
  /*
  training phase - train neural network with new data
    load training file
    train network with data (repeat for all data segments chosen in random order)
      select random data segment
      network responds to the data segment
      network trains according to the data segment
       modify and save weights of each neuron in the current network

    other stuff
      display how many are classified and how many in each classification
      add reset button to reset all the neurons to random weights (reset to untrained network)
  */

  /*
  test data with incoming eeg signals (no back propagation)

  network will respond to each 100 eeg segments for each of the channels (classifies type of s
ignal)
  integrate with rest of eeg headset code

  */
}

void serialEvent(Serial myPort) {
  //This section needs to be first so that serialEvent is not disabled
  serialData = myPort.readStringUntil('\n');
  serialData = serialData.substring(0, serialData.length() - 2);
  int[] splitData = int(split(serialData, ","));

  if(millis() - lastShift > 2) {
    if(mode == 0 || mode == 1) {
      for(int i = data1.length-1; i > 0; i--) {
        data1[i] = data1[i-1];
      }
      for(int i = data2.length-1; i > 0; i--) {
        data2[i] = data2[i-1];
      }
      lastShift = millis();
    }
    if(mode == 1) {
      marker1++;
      if(marker1 >= 250) marker2++;
      if(marker1 >= 500) marker3++;
      if(marker1 >= 750) {
        mode = 2;
        start = marker1;
        end = marker2;
      }
    }
  }

  //println(splitData[1]);

  data1[0] = int(map(float(splitData[0]), 0, 4095, 0, 400));
  data2[0] = int(map(float(splitData[1]), 0, 4095, 0, 400));

  println(serialData);
}
```

```
void appendTextToFile(String filename, String text) {
  File file = new File("/Users/student/Documents/Processing/NeuralNetwork/" + filename);
  try {
    FileWriter fw = new FileWriter(file, true);///true = append
    BufferedWriter bw = new BufferedWriter(fw);
    PrintWriter pw = new PrintWriter(bw);

    pw.println(text);
    pw.close();
  } catch (IOException e) {
      e.printStackTrace();
  }
}

void trainNetwork() {

}
```

DFT Example

```
float[] frequencyBins;
float[] signal;    //over 1 second

float[] testSignal = {1, 0, -1, 0, 1, 0, -1, 0};

int N = 2048;
float freq1 = 41;
float freq2 = 33;
float sigPhase1 = 0;
float sigPhase2 = 0;

void setup() {
  frequencyBins = new float[N];
  signal = new float[N];
  for(int x = 0; x < signal.length; x++) {
    signal[x] = cos((2*PI/N)*freq1*x + sigPhase1) + cos((2*PI/N)*freq2*x + sigPhase2);
  }
}

void draw() {
  int start = millis();
  float[] result = dft(signal);
  println(str(millis() - start) + " ms");    //print time it takes to compute

  for(int x = 0; x < signal.length; x++) {
    //println(str(x) + ": " + str(result[x]));
  }

  while(true);
}

//takes in data array data[] - data is an array vector
//takes in N number of samples in data[]
//will return frequency components array vector
float[] dft(float[] data) {
  int N = data.length;
  int K = N/2;

  for(int k = 0; k<N; k++) {
    float realSum = 0;
    float imaginarySum = 0;
    for(int n = 0; n<N; n++) {
      realSum += data[n] * cos((2*PI/N)*k*n);
      imaginarySum += data[n] * sin((2*PI/N)*k*n);
    }
    //realSum = realSum*(1/N);
    frequencyBins[k] = sqrt(realSum*realSum + imaginarySum*imaginarySum);    //combine complex
number to real number
  }
  return frequencyBins;
}
```

FFT Example

```
//WORK ON ACCEPTING INPUTS THAT ARE NOT A POWER OF 2

Complex[] output;

float sampleTime = 0.1;    //in seconds
float[] signal;
Complex[] signalC;

float[] realResult;
float[] pixHeight;

float[] testSignalF = {0, 10, 0, -10, 0, 10, 0, -10, 0, 10, 0, -10, 0, 10, 0, -10};  //signal
gathered over sampleTime seconds
Complex[] testSignalC;

int N = 128;            //N must be a power of 2
float freq1 = 41;
float freq2 = 33;
float sigPhase1 = 0;
float sigPhase2 = 0;

void setup() {
  signal = new float[N];
  signalC = new Complex[N];
  for(int x = 0; x < signal.length; x++) {
    signal[x] = cos((2*PI/N)*freq1*x + sigPhase1) + cos((2*PI/N)*freq2*x + sigPhase2);
    signalC[x] = new Complex(signal[x], 0);
  }

  float[] scaleSignal = new float[testSignalF.length];
  testSignalC = new Complex[testSignalF.length];
  for(int x = 0; x < testSignalF.length; x++) {
    scaleSignal[x] = map(testSignalF[x], min(testSignalF), max(testSignalF), 0, 10);
//scale to a factor of 10
    testSignalC[x] = new Complex(scaleSignal[x], 0);
  }

  realResult = new float[N];
  pixHeight = new float[realResult.length];

  size(1300, 800, P2D);
  rectMode(CORNERS);
}

void draw() {
   background(0);
   fill(255);

  int start = millis();
  Complex[] result = fft(signalC);
  println(str(millis() - start) + " ms");

  for(int x = 0; x < signalC.length; x++) {
    //result[x].describe();
    realResult[x] = sqrt(result[x].real*result[x].real + result[x].img*result[x].img);
    pixHeight[x] = map(realResult[x], min(realResult), max(realResult), 0, 300);
    println(str(x/sampleTime) + " Hz: " + (1.0/testSignalF.length)*realResult[x]);
  }

  //graph result
  for(int x = 0; x < pixHeight.length; x++) {
    rect(x*10, height-pixHeight[x], (x+1)*10, height);
    //println(pixHeight[x]);
  }
```

```
  //while(true);
}

//use custom complex number class
Complex[] fft(Complex[] data) {
  int n = data.length;
  if(n == 1) return data;

  //divide into even and odd
  int halfN = n/2;
  Complex[] even = new Complex[halfN];
  Complex[] odd = new Complex[halfN];
  for(int i = 0; i < halfN; i++) {
    even[i] = data[i*2];
    odd[i] = data[i*2+1];
  }

  //recursive
  even = fft(even);
  odd = fft(odd);

  output = new Complex[n];    //initialize output

  //after recursion - combine
  for(int k = 0; k < halfN; k++) {
    output[k] = new Complex(even[k].real + (odd[k].real*cos(2*PI*k/n) -
odd[k].img*sin(2*PI*k/n)),       //complex number multiplication
                        even[k].img + (odd[k].real*sin(2*PI*k/n) + odd[k].img*cos(2*PI*k/n)));

    output[k+n/2] = new Complex(even[k].real - (odd[k].real*cos(2*PI*k/n) -
odd[k].img*sin(2*PI*k/n)),   //complex number multiplication
                            even[k].img - (odd[k].real*sin(2*PI*k/n) +
odd[k].img*cos(2*PI*k/n)));
  }
  return output;
}

//can be swapped for 2D vector class
class Complex {
    float real;   // the real part
    float img;    // the imaginary part

    public Complex(float real, float img) {
        this.real = real;
        this.img = img;
    }

    public float real() {
      return this.real();
    }

    public float img() {
      return this.img();
    }

    void describe(){
      println( "" + this.real + " + " + this.img + "i" );
    }
}
```

FFT With Windowing

```
//WORK ON ACCEPTING INPUTS THAT ARE NOT A POWER OF 2
//currently only accepts number of samples that are a power of two

Complex[] output;

float sampleTime = 0.1;    //in seconds
float[] signal;
float[] signalWindowed;
float[] windowF;            //flat top window
float[] windowHann;         //hanning window

//values needed for flat top window
float a0 = 0.21557895;
float a1 = 0.41663158;
float a2 = 0.277263158;
float a3 = 0.083578947;
float a4 = 0.006947368;

Complex[] signalC;

float[] realResult;
float[] pixHeight;

float[] testSignalF = {0, 10, 0, -10, 0, 10, 0, -10, 0, 10, 0, -10, 0, 10, 0, -10};  //signal
gathered over sampleTime seconds
Complex[] testSignalC;

int N = 256;            //N must be a power of 2
float freq1 = 100;
float freq2 = 33;
float sigPhase1 = 0;
float sigPhase2 = 0;

void setup() {
  signal = new float[N];
  signalWindowed = new float[N];
  windowF = new float[N];
  windowHann = new float[N];
  signalC = new Complex[N];
  for(int x = 0; x < signal.length; x++) {
    signal[x] = cos((2*PI/(N+1))*freq1*x + sigPhase1) + cos((2*PI/(N+1))*freq2*x + sigPhase2);
    windowF[x] = a0-a1*cos(2*PI*x/N)+a2*cos(4*PI*x/N)-a3*cos(6*PI*x/N)+a4*cos(8*PI*x/N);
    windowHann[x] = 0.5*(1-cos(2*PI*x/N));
    //window[x] = 1;
    signalWindowed[x] = signal[x]*windowHann[x];
    signalC[x] = new Complex(signalWindowed[x], 0);
  }

  float[] scaleSignal = new float[testSignalF.length];
  testSignalC = new Complex[testSignalF.length];
  for(int x = 0; x < testSignalF.length; x++) {
    scaleSignal[x] = map(testSignalF[x], min(testSignalF), max(testSignalF), 0, 10);
//scale to a factor of 10
    testSignalC[x] = new Complex(scaleSignal[x], 0);
  }

  realResult = new float[N];
  pixHeight = new float[realResult.length];

  size(1300, 800, P2D);
  rectMode(CORNERS);
}
```

```
void draw() {
   background(0);
   fill(255);

  int start = millis();
  Complex[] result = fft(signalC);        //calculate fft
  println(str(millis() - start) + " ms");

  for(int x = 0; x < signalC.length; x++) {
    //result[x].describe();
    realResult[x] = sqrt(result[x].real*result[x].real + result[x].img*result[x].img);
    pixHeight[x] = map(realResult[x], min(realResult), max(realResult), 0, 300);
    println(str(x/sampleTime) + " Hz: " + (1.0/testSignalF.length)*realResult[x]);
  }

  for(int x = 0; x < pixHeight.length; x++) {
    rect(x*10, height-pixHeight[x], (x+1)*10, height);
    //println(pixHeight[x]);
  }




  //while(true);
}

//use custom complex number class
Complex[] fft(Complex[] data) {
  int n = data.length;
  if(n == 1) return data;

  //divide into even and odd
  int halfN = n/2;
  Complex[] even = new Complex[halfN];
  Complex[] odd = new Complex[halfN];
  for(int i = 0; i < halfN; i++) {
    even[i] = data[i*2];
    odd[i] = data[i*2+1];
  }

  //recursive
  even = fft(even);
  odd = fft(odd);

  output = new Complex[n];    //initialize output

  //after recursion - combine
  for(int k = 0; k < halfN; k++) {
    output[k] = new Complex(even[k].real + (odd[k].real*cos(2*PI*k/n) -
odd[k].img*sin(2*PI*k/n)),       //complex number multiplication
                        even[k].img + (odd[k].real*sin(2*PI*k/n) + odd[k].img*cos(2*PI*k/n)));

    output[k+n/2] = new Complex(even[k].real - (odd[k].real*cos(2*PI*k/n) -
odd[k].img*sin(2*PI*k/n)),   //complex number multiplication
                            even[k].img - (odd[k].real*sin(2*PI*k/n) +
odd[k].img*cos(2*PI*k/n)));
  }
  return output;
}
```

```
//can be swapped for 2D vector class
class Complex {
    float real;   // the real part
    float img;    // the imaginary part

    public Complex(float real, float img) {
        this.real = real;
        this.img = img;
    }

    public float real() {
      return this.real();
    }

    public float img() {
      return this.img();
    }

    void describe(){
      println( "" + this.real + " + " + this.img + "i" );
    }
}
```

Reflection

There are many lessons I learned these past months. Firstly, write the most important parts of the DER first and then add detail later. I mistakenly added too much detail in the beginning, got burned out, and now I am skipping over detail on some of the sections such as the mathematics and design part. Secondly, do not be too ambitious. Unfortunately, this was the case when I first started and when I was halfway through and I paid a price for that as my presentation of my device did not go well and not much worked. However, with some miracles, I managed to get the ISP working over the weekend. The price for this though was staying up until 4 am trying to write the rest of my DER and film my video which I am currently doing. Since it is this late (or early), I am going to go to submit what I have and go to bed, but I really do want to write more on the mathematics and design aspects as well as add in more pictures so I will do so tomorrow when I have more energy. I knew that this was going to be a late submission and I am sorry about that, so I decided to space things out and take breaks when needed so as not to get too stressed. Though writing this now I really do wish I could have finished earlier as I am really tired and do not have the mental energy to format everything properly. But overall, I am really pleased and grateful that this ISP managed to work out in the end as all the hard work I put over the last 3 months finally paid off and it lifted my mood from the presentation that did not go well. I will save my ambitions for the future when I have time to work more on this project.

Project 3.5: Pin Change Interrupt

Purpose

The purpose of this circuit is to exploit the low level inbuilt external pin change interrupts of the ATmega328P to free up the CPU for other tasks. This is demonstrated through a wiring of a simple 3-digit numerical combination lock.

References

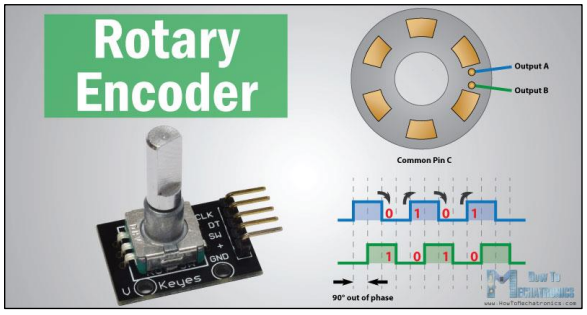
http://darcy.rsgc.on.ca/ACES/TEI4M/RegisterLevelOptimization/Interrupts.html#PinChange
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

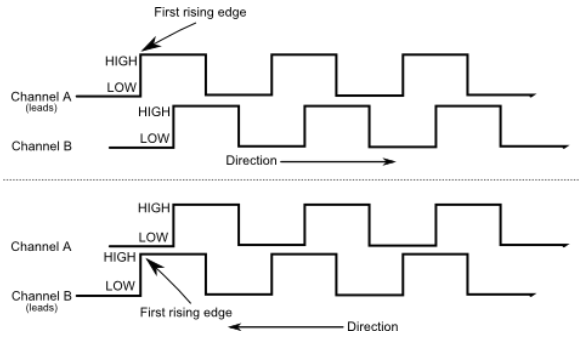
Procedure

Parts Table	
Quantity	Description
1	Arduino NANO
1	Rotary encoder + PCB
1	Debounced Button PCB
1	10 kΩ fixed resistor
1	4511 binary decimal decoder
1	Common cathode 7-segment

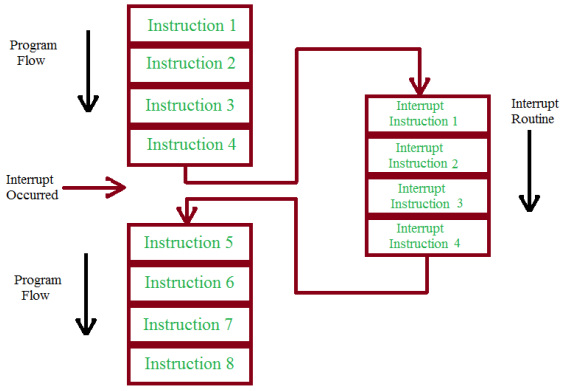
To create the combination lock, a rotary encoder connects to an Arduino NANO which controls certain pin outputs to drive a 7-segment display through the help of the 4511 binary to 7-segment display IC used in the ICS2O Counting Circuit project. A bi-color LED indicates the state of the lock; red for closed and green for open. The 7-segment display presents the number to select while a clockwise and counterclockwise turn increments and decrements the digit by 1 with a rollover to 0 and 9 respectively. To input the digit shown, the debounced rotary encoder button is pressed and after three digits have been selected, the bi-color turns green if the three numbers are correct and in the correct order, otherwise the bi-color led flashes red. To reset the digit selection, the other debounced button is pressed.

All significant parts used have had extensive use in other projects, so a quick recap will be provided. The rotary encoder, last used in the ICS3U Medium ISP, the LiDAR measurement device, works through greycode counting, a system of counting used for mechanical devices. The layout is featured to the right where one turn results in the two output pins presenting both highs or lows.





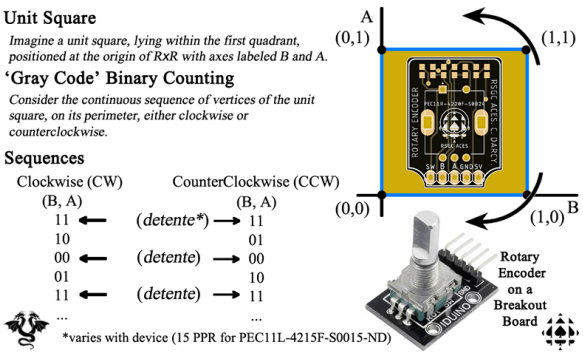
When turning the rotary encoder, the outputs will change state one after the other, forming 90 degree phase shifted square waves. The orientation of the phase shift depends on the direction of the turn, therefore, by reading the two pins, the direction and number of turns can be calculated. Also, a push of the rotary encoder takes the digit shown as one of the digit inputs and an external push button resets all the input digits to 0.



In order to sense and execute all these actions, the Arduino NANO uses interrupts, which is defined as a signal emitted by hardware or software to perform a process or set of instructions requiring immediate attention. The ATmega328P on the Arduino NANO comes with timer and external interrupts, but for this project only external interrupts are needed as the different software actions are triggered by external state changes. When these interrupts are called, these "actions", known as interrupt service routines (ISRs) are carried out. This is where the incrementing, resetting and storing actions are called, however, ISRs are recommended to run as fast as possible, therefore no complex functions should be called directly in an ISR. Instead, a common strategy used in this project is to simply set a Boolean value in the ISR to let the CPU know through an if statement that a specific interrupt has been triggered.

To trigger an external interrupt on the ATmega328P, a change of state on certain pins needs to be made. For more specific external interrupts such as INT0 and INT1 located on digital pins 2 and 3, the interrupt can be set to trigger on falling, rising and changing edges. However, for regular pin change interrupts, specified as PCINT on each pin, they are triggered whenever a change in state occurs. On the ATmega328P, there are only 3 pin change interrupts linked to all the available I/O pins, one for each I/O port. In this project, PCIE2 is used which triggers when interrupt enabled pins on port D (digital pins 0 to 7) experience a state change. Therefore to read rotary encoder rotations, pin change interrupts are enabled on pins 4 and 5 which link to output pins A and B on the rotary encoder. This leaves pins 2 and 3 for INT0 and INT1 interrupts which are set to trigger on a rising edge from the two push buttons. Setting interrupts and ports require manipulation of bits inside certain ATmega328 registers which is shown in the code and datasheet.

In this project, a turn increases or decreases by 1 the binary value sent from the Arduino NANO pins into the 4511 where the 4511 converts binary into the corresponding number presented on the 7-segment display. It does this by writing a value to port C of the ATmega328P. Since the port is a byte where each bit controls the state of a pin, writing a value to the port produces a binary value which can be read by the 4511. To set the pins for output, the DDR registers must be set where a 1 sets the is setting the corresponding pin to output and a 0 sets it to input. If set to input, the states present on the pin are read by reading the port through the PIN macro. This is used to read the states of the rotary encoder by equating a value to PIND where D stands for port D. If set to output, the pin can be low or high by setting the bits on the PORT macro to 0s or 1s. To output a binary value to the 4511, the low nibble in DDRC is set and PORTC is set to the digit value selected. Also, the two highest pins of port D are set to output through setting bits 6 and 7 on DDRD high, enabling PORTD output control to toggle between red and green of the bi-color LED, representing locked and unlocked respectively.



To detect the direction of rotation of the rotary encoder, one can imagine a unit square with coordinates (0, 0) (0, 1) (1, 1) and (1, 0). These coordinates correspond to the output pin states of the rotary encoder when rotated a certain direction. Turning clockwise results in coordinates that correspond with traversing the unit square clockwise producing 01 and 11 with one turn and 10 and 00 in the other turn. Turning counterclockwise would produce a 10 and 11 for one of the turns and 01 and 00 for the other turn. By combining the coordinates into a nybble, unique numbers are generated for rotation directions, allowing it to be calculated. The high bits of the nibble represent the current pin states while the lower two are the previous states. So 1110 and 0001 mark a counterclockwise turn and 1101 and 0010 mark a clockwise turn. If statements then check the numbers and execute a digit increase or decrease, though to maximize efficiency, a combination of bit manipulation and masking would be preferable.

An emphasis on low level coding techniques through direct manipulation bits in a port increases speed and efficiency compared to relying on the high level Arduino functions. This is where high-level software distills down into the depths of low-level control of ATmega328P hardware. The code in this project uses no pinMode, digitalWrite, digitalRead and attachInterrupt functions. The code is also flexible, in that parameters can be changed in the #define statements with minimal rewriting for the program to work on different ports, pins , and even different AVR MCUs. For example, to change the passcode, one only needs to modify the array that stores the passcode.

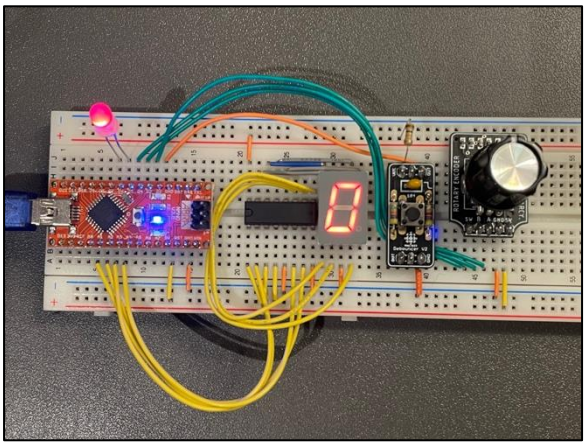
Code

```
// PROJECT  : Combination lock with rotary encoder
// PURPOSE  : Exploit external and pin change interrupts
// COURSE   : ICS4U
// AUTHOR   : Xander Chin
// DATE     : February 3, 2022
// MCU      : ATmega328P
// STATUS   : Working
// REFERENCE:
http://darcy.rsgc.on.ca/ACES/TEI4M/RegisterLevelOptimization/Interrupts.html#PinChange
// MY VIDEO : youtube link...

#include <Mega328P.h>

#define EI0LOW       (0<<ISC01) | (0<<ISC00)    // INT0 trigger on LOW state
#define EI0CHANGE    (0<<ISC01) | (1<<ISC00)    // INT0 trigger on state change
#define EI0FALLING   (1<<ISC01) | (0<<ISC00)    // INT0 trigger on falling edge
#define EI0RISING    (1<<ISC01) | (1<<ISC00)    // INT0 trigger on rising edge

#define EI1LOW       (0<<ISC11) | (0<<ISC10)    // INT1 trigger on LOW state
#define EI1CHANGE    (0<<ISC11) | (1<<ISC10)    // INT1 trigger on LOW state
#define EI1FALLING   (1<<ISC11) | (0<<ISC10)    // INT1 trigger on LOW state
#define EI1RISING    (1<<ISC11) | (1<<ISC10)    // INT1 trigger on LOW state

#define PC2ENABLE (1<<PCIE2)      // pin change interrupts on Port D
#define AENABLE   (1<<PCINT20)    // pin change interrupt on PD4
#define BENABLE   (1<<PCINT21)    // pin change interrput on PD5
#define PCFLAG    (1<<PCIF2)      // pin change flag

#define READENC   PIND
#define A         (1<<PD4)        // pin A rotary encoder
#define B         (1<<PD5)        // pin B rotary encoder

#define DDRDIGIT  DDRC            // 4511 ddr
#define PORTDIGIT PORTC           // 4511 port

#define DDRLED    DDRD            // bicolor led ddr
#define PORTLED   PORTD           // bicolor led port
#define RED       (1<<PD6)        // bicolor red led pin
#define GREEN     (1<<PD7)        // bicolor green led pin
#define LEDPINS   RED | GREEN     // bicolor led outputs

#define NUMDIGITS 10                // max number size for 4511
uint8_t digit = 0;                  // current digit shown on 7-segment
volatile uint8_t digitPos = 0;      // current digit position of user input

volatile bool extInt0 = false;      // external button custom flag
volatile bool extInt1 = false;      // rotary encoder button custom flag
volatile bool changeInt = false;    // rotary encoder custom flag

volatile uint8_t state = 0;         // rotary encoder current and previous states
bool wrong;                         // incorrect code flag

const uint8_t code[] = {3,1,4};     // correct password. edit to change password
uint8_t codeInput[sizeof(code)];    // user input code

void pinChangeSetup() {
  PCICR |= PC2ENABLE;             // enable pin change interrupts on port D
  PCMSK2 |= AENABLE | BENABLE;    // activate pin change interrupt on these pins
  PCIFR |= PCFLAG;                // clear flag by setting it to 1
}

void extIntSetup() {
  EICRA |= EI0FALLING | EI1FALLING;   // select the preferred 'edge sense'
  EIMSK |= (1<<INT0) | (1<<INT1);     // enable the two external interrupts
  EIFR |= (1<<INTF1) | (1<<INTF0);    // clear the two interrupt flags
}
```

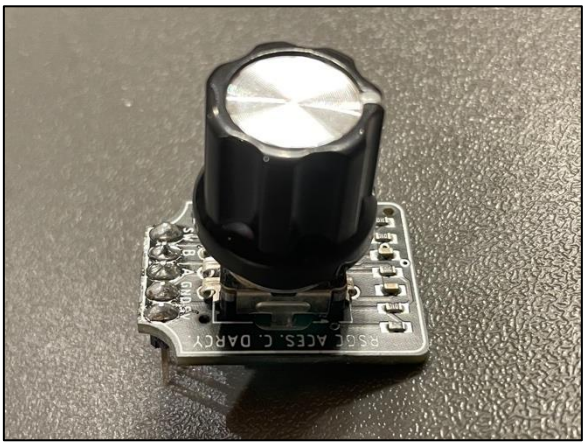
```
void displaySetup() {
  DDRDIGIT |= 0x0F;   // set 4511 pins to output
}

void bicolorSetup() {
  DDRLED |= LEDPINS;    // set led pins to output
  PORTLED |= RED;       // turn on red
  PORTLED &= ~GREEN;    // turn off green
}

void setup() {

  SREG &= ~(1<<7);    //cli(); disable global interrupts

  pinChangeSetup();   // prepare to read rotary encoder
  extIntSetup();      // prepare INT0 and INT1 for use
  displaySetup();     // setup ports for 4511 and 7-segment
  bicolorSetup();     // setup ports for bicolor led
  resetCodeInput();   // set code input to 0s
  showDigit(digit);   // present a value to start

  SREG |= (1<<7);     //sei(); enable global interrupts
}

void loop() {
  if(changeInt) {
    changeInt = false;                    //clear pin change flag
    if(state == B1110 || state == B0001) {
      digit--;                            //decrement current digit
      digit = digit % (256 - NUMDIGITS);  //rollover at 0
    } else if(state == B1101 || state == B0010) {
      digit = (digit + 1) % NUMDIGITS;    //increment with rollover
    }

    //same as above but with XORs and modulus
    //commented out due to inefficiency of modulus
    /*
    if(!((state^B1110) % 15)) {
      digit--;
      digit = digit % (256 - numDigits);
    }
    if(!((state^B1101) % 15)) {
      digit = (digit + 1) % numDigits;
    }
    */
  }

  if(extInt1) {
    extInt1 = false;    //clear INT1 flag
    blinkRed();         //notify user of their input

    codeInput[digitPos] = digit;  //log digit in password input array
    digitPos++;                   //increase password input index array

    if(digitPos == sizeof(code)) {                  //if at max index
      for(uint8_t x = 0; x < sizeof(code); x++) {   //loop through digits
        if(codeInput[x] != code[x]) {               //check each digit
          wrong = true;                             //set wrong flag
        }
        codeInput[x] = 0;   //clear password input in the process
      }
      digitPos = 0;         //reset password input index array for next use

      wrong ? flashRed() : correct();   //check wrong flag
      wrong = false;                    //clear wrong flag
      digit = 0;                        //reset current digit to 0
    }
  }
```

```
  if(extInt0) {
    extInt0 = false;    //set false
    resetCodeInput();   //reset password input
    scramble();         //7-segment scramble effect - notifies user of reset
    digit = 0;          //set current digit to 0
  }

  showDigit(digit); //display digit
}

ISR(INT0_vect) {
  extInt0 = true;   //set INT0 flag
}


ISR(INT1_vect) {
  extInt1 = true;   //set INT1 flag
}

ISR(PCINT2_vect) {
  state = (READENC & (A | B)) >> 2 | state >> 2;   //immediately read in case of change
  changeInt = true;                                //set pin change flag
}

void showDigit(uint8_t d) {
  PORTDIGIT = d;    //output binary on low nybble of port to 4511
}

void flashRed() {
  for(uint8_t x = 0; x < 8; x++) {
    PORTLED ^= RED;   //toggle between led states
    delay(100);       //hold delay (use timers for non-blocking)
  }
}

void blinkRed() {
  PORTLED &= ~RED;   //led off
  delay(100);        //hold
  PORTLED |= RED;    //led on
}

void correct() {
  PORTLED ^= LEDPINS;   //switch colors (to green)
  delay(3000);          //hold for 3 seconds
  PORTLED ^= LEDPINS;   //switch colors back (to red)
}


void scramble() {
  for(uint8_t x = 0; x < 15; x++) {   //iterate through digits
    digit = (digit + 1) % NUMDIGITS;  //increase + rollover
    showDigit(digit);                 //display digit
    delay(30);                        //short hold
  }
}

void resetCodeInput() {
  for(uint8_t x = 0; x < sizeof(code); x++) {   //run through all elements
    codeInput[x] = 0;                           //set to 0
  }
  digitPos = 0;                                 //set password index to 0
}
```

Media



The combination lock circuit



The rotary encoder custom PCB with a debounce circuit designed by C. D'Arcy

YouTube video link: https://youtu.be/nX0koJXdlao

Reflection

Although simple, I loved this project because it allowed me to delve deeper into the details, as opposed to others like my independent study projects that left me no time to do so. Details included replacing all high-level code with low-level code, modularizing it, and finding the best and most efficient way to read the rotary encoder. Although the last part was somewhat unsuccessful, I'm glad I took the time and effort to try as I gained insights into the binary outputs of rotary encoder pins and also enjoyed the process. This simple project also allowed me to put the necessary time into my video and DER while balancing workload from my other courses. Learning how to configure interrupts and normalizing low level port and register programming pays huge dividends as they will form the basis of my code from now on to greatly boost performance. It also opens up the door to more ATmega328P peripherals to maximize what the MCU can do in future projects.

Project 3.6: (ISP – Medium): Giant RGBW LED Matrix

Purpose

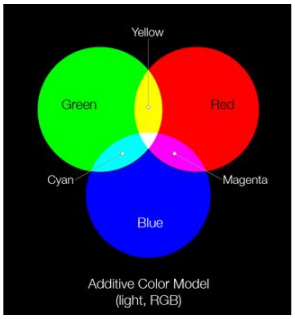
The purpose of this Independent Study Project is to explore interest in hardware, software and design concepts to create a working device. For my ISP, I chose to create a large 32 × 24 RGB LED matrix using 768 SK6812 Neopixel LEDs and ping pong balls as diffusers. This was inspired by Bitluni's own V3 RGB LED wall.

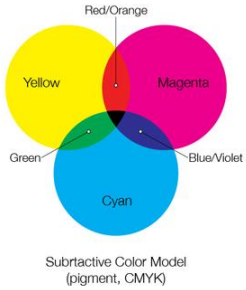
References

Bitluni Ping Pong LED wall V3: https://youtu.be/EZEMK-C-nSo
http://www.normandled.com/upload/201805/SK6812RGBX-XX%20Datasheet.pdf

Procedure

The project's most defining part are the red green and blue (RGB) LEDs, specifically 768 of them to make a 24 ´ 36 matrix. By varying the brightness of these three fundamental colours, all colors of the visible light spectrum can be displayed. Creating colors using light is very different from creating colors using paint. Mixing light adds the two wavelengths together which is why white is formed with red green and blue leds come together while mixing paint subtracts the color wavelengths, producing a darker color when adding in more colors.





For pixels, they display light and therefore abide by the laws of additive wavelengths. This is where the name RGB comes from, an acronym for the three fundamental colors of light: red, green and blue. Another popular format is HSV or hue, saturation and value. Hue is quantitatively represented from 0 to 360 and includes the different colors of the visible spectrum. Saturation determines how "white" the color appears with values from 0 to 100 while value is the brightness level or how "black" the color looks which also has a range from 0 to 100. In this project, RGB values are the main format.

Parts Table	
Quantity	Description
1	Arduino NANO
1	Lincoiah 5V 60A pwr supply
768	SK6812 RGBW LED
96	LED PCB strips
24	Right PCB connectors
24	Left PCB connectors
72	Middle PCB connectors
*	Male breakable headers
*	Female breakable headers
*	Shielded cable (for data lines)
*	Power cables + wires

To produce different colors, one has to vary intensity or brightness of red green and blue, the most obvious way is to control the current or PWM duty cycle of each color component. However, as the number of LEDs increases, the amount of I/Os with PWM needed to control increases, therefore, addressable RGB LEDs were invented which drastically reduces the amount of control pins. Such LEDs have a tiny microcontroller built in to take in data signals and translate them into To write to these LEDs, common modern day communication protocols such as SPI are used, therefore needing a clock and data pin, but one of the most popular methods is the one-wire communication used in Adafruits neopixels. These RGB LEDs only use one pin and can be chained together by connecting their data out of the previous LED to the data in to the next LED to form large displays while the first LED in the chain connects to a microcontroller I/O. Popular Neopixels include WS2812, APA106 RGB LEDs and SK6812 RGBW LEDs which include an extra white LED.

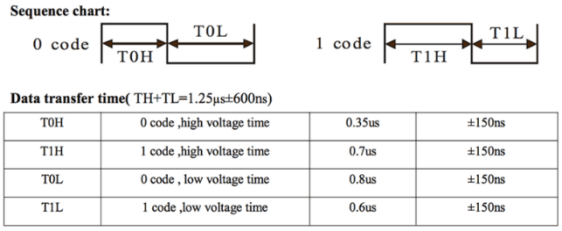
13. The data structure of 32bit:

R7	R6	R5	R4	R3	R2	R1	R0	G7	G6	G5	G4
G3	G2	G1	G0	B7	B6	B5	B4	B3	B2	B1	B0
W7	W6	W5	W4	W3	W2	W1	W0				

Note: high starting, in order to send data (R7 - R6 - ....W0)

The microcontroller inside varies each LED brightness through 8-bit control, therefore one RGB LED requires 24 bits of information while an RGBW variant requires 32 bits. Sending these bits though one wire involves specific timings and since each are chained together, the bits are shifted down as they are generated. For each pixel, the 32 bits are separated into a green byte, red byte, blue byte and white byte from most significant to least, therefore, the highest significant green bit must be sent first and the lowest significant bit of the white byte is sent last. Sending low or high bits involves setting the data pin high then low for a certain amount of time. This sends the bits out but does not turn on the LEDs. To turn on these LED's, a latch is needed which is generated through a low state for a certain amount of time. Each Neopixel variant has slightly different timings, but all are on the scale of nanoseconds. For example here are the timings to send out a high and low bit and a latch time for the SK6812 RGBW LEDs:

T0H	0 code, high level time	0.3μs	±0.15μs
T0L	0 code, low level time	0.9μs	±0.15μs
T1H	1 code, high level time	0.6μs	±0.15μs
T1L	1 code, low level time	0.6μs	±0.15μs
Trst	Reset code，low level time	80μs	

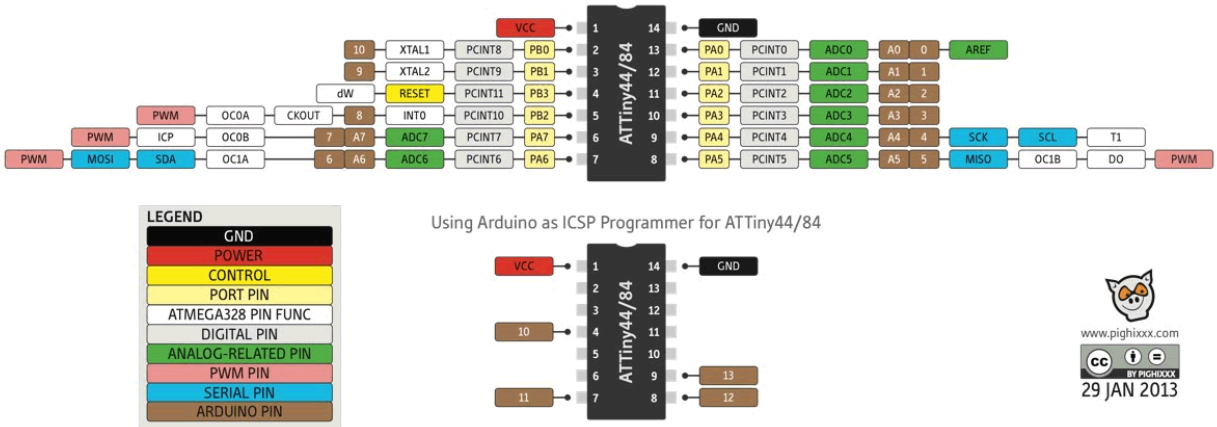


Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.35us	±150ns
T1H	1 code ,high voltage time	0.7us	±150ns
T0L	0 code , low voltage time	0.8us	±150ns
T1L	1 code ,low voltage time	0.6us	±150ns

However, many neopixel bit timings work with some general approximation as the values listed above are quite flexible. A popular method for timing approximation that works with most neopixels is to split up the timings of a single bit, which is around 1050 nanoseconds to send a single bit, into three stages of equal time, each being around 350 nanoseconds. In the first stage, the data pin is always high, no matter the bit, in the second stage, the bit being sent corresponds to the state of the the pin, and in the third stage, the data pin is low no matter the bit. This set of timing can therefore approximate bit timings of many types of neopixels. With different microcontrollers with different clock speeds, it is near impossible to achieve an exact 350 nanoseconds for each stage, but again, neopixels are quite flexible and timing can vary a little as long as it follows the three stage protocol mentioned above. For an ATmega328P running at 16 MHz, each clock cycle is 62.5 nanoseconds, therefore a good approximation of 350 nanoseconds is 6 clock cycles where $6 \times 62.5 = 375$ nanoseconds. Testing further reveals that 8 clock cycles or 500 nanoseconds does the trick as well, though any higher than that would turn low bits into high bits. On an ATtiny84 running on its 8 MHz internal clock, each clock cycle is 125 nanoseconds, so 3 or 4 clock cycles would be ideal. Between bits, the signal can surprisingly stay low for up to 80 nanoseconds, where afterward it latches and displays the written bits. 80 microseconds is around 1600 cycles for a 16 MHz or 800 cycles on a 8 MHz latches them. It should be noted that any low state for more than this amount latches the LEDs, therefore, the maximum time between sending bits is less than 80 microseconds, which limits the computations that can be made between sending bits.

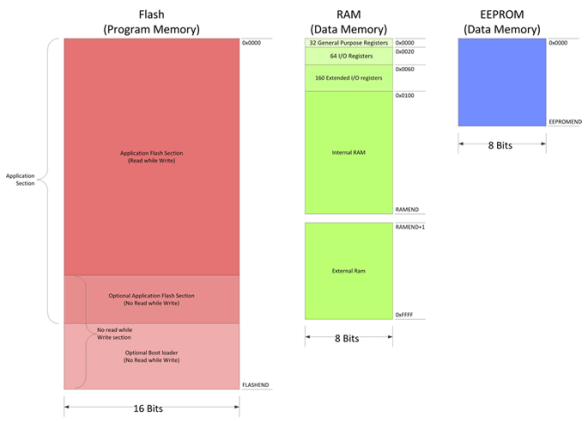
With these nanoscale timing requirements, the number of clock cycles of each instruction must be taken into account. Low level port manipulation was used where setting and clearing bits each takes two clock cycles. To set and clear specific bits, bit manipulation is involved which takes another two clock cycles, therefore, a delay of 2 clock cycles is needed to complete the 6 cycle stage. In between bits, others can be calculated and subsequently sent out, however as mentioned above, the computations must be done under 80 nanoseconds in order to avoid the LEDs from latching. It is also important to turn off global interrupts by clearing the highest bit in the SREG register since they disturb the crucial timings of the LEDs.



With the ATmega and ATtiny chips, most ports contain a maximum of 8 pins and all 8 can be individually set or cleared through the aforementioned low level port command. Therefore, 8 outputs can be controlled in parallel which reduces the amount of time it takes to send the signal by a factor of 8, greatly boosting performance with large setups. So, any Arduino with at least one full sized port can write bits to neopixels in parallel, for example the attiny84. In theory, the delays needed for timing can be replaced with other port commands which can allow for 16 or 24 line parallel control if there were enough pins. This would be a great application for the Arduino Mega.

Instead of delays, timers were also experimented with to control neopixels using ISRs. However, through testing, ISRs when called take a good amount of clock cycles to execute, therefore, they cannot be used to control neopixels. Instead, the pins connected to timers can be used to output the data, but this would forego the parallel output from a port.

To extend use beyond the AVR microcontrollers, the ESP32 was looked at. It is much more powerful but does not have port registers. Instead it has many different peripherals, two in particular, the I2S sound and RMT remote control, are perfect for sending customized signals since they deal with sending and receiving different kinds of sound and IR signals. These two peripherals also include a parallel mode where signals can be sent out simultaneously. Plus, sending and buffering these signals is handled by a Direct Memory Access (DMA), requiring no CPU control. These features are used in the FastLED library. Since setting them up to write neopixels required a lot of complex manipulation, they were not used.

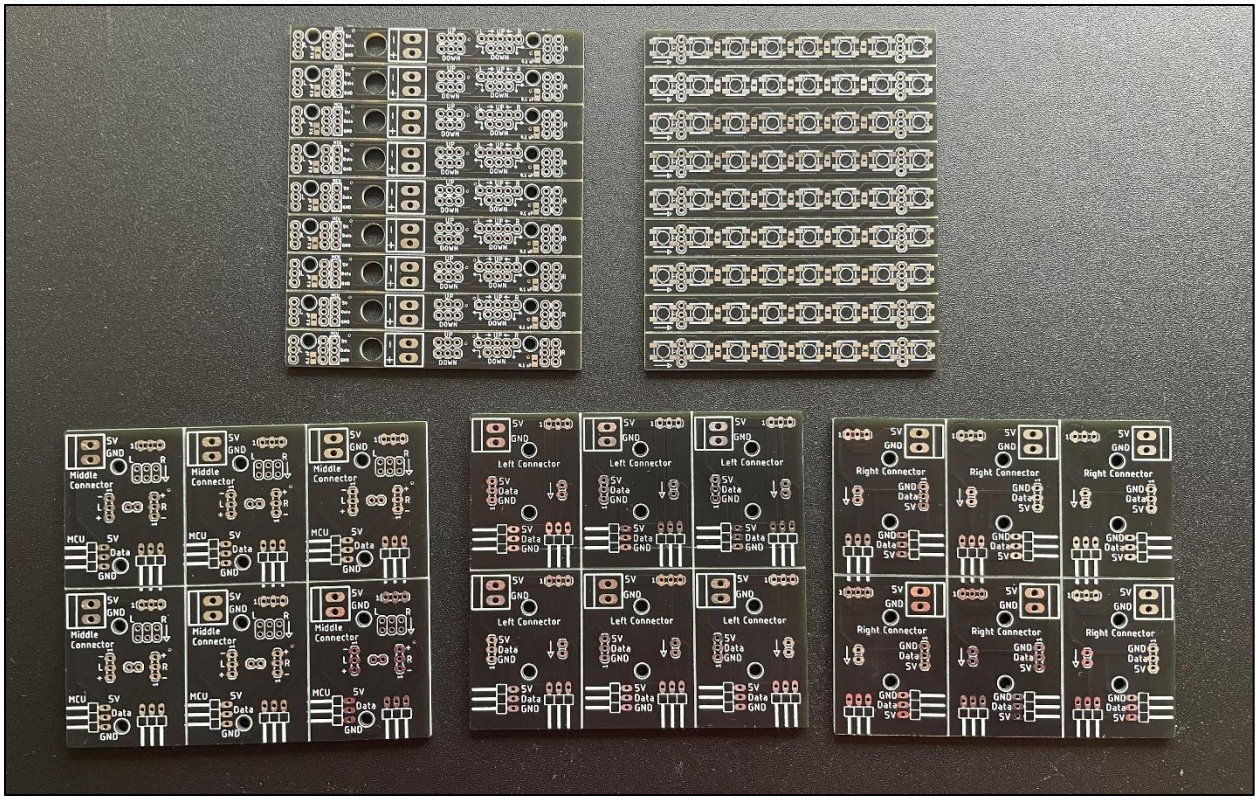


To display images, a buffer containing all the color bits of each pixel is needed. Since there are 768 pixels and each pixel contains 4 bytes of red, green, blue and white, a total of 3072 bytes are needed to display a still image. This is more than the available SRAM in an ATmega328P so flash or program memory containing 32 kilobytes of storage is used to keep these images. Unfortunately, storage in flash is read-only, data can only be written about 10,000 times, and certain in built functions are required to properly read bytes stored in program memory, but it is non-volatile so powering off the MCU does not erase its contents. If one wants to change data, SRAM must be used to create a buffer which limits the amount of pixels. This can be resolved by moving to an Arduino Mega or by using external memory such as an EEPROM or SD card.

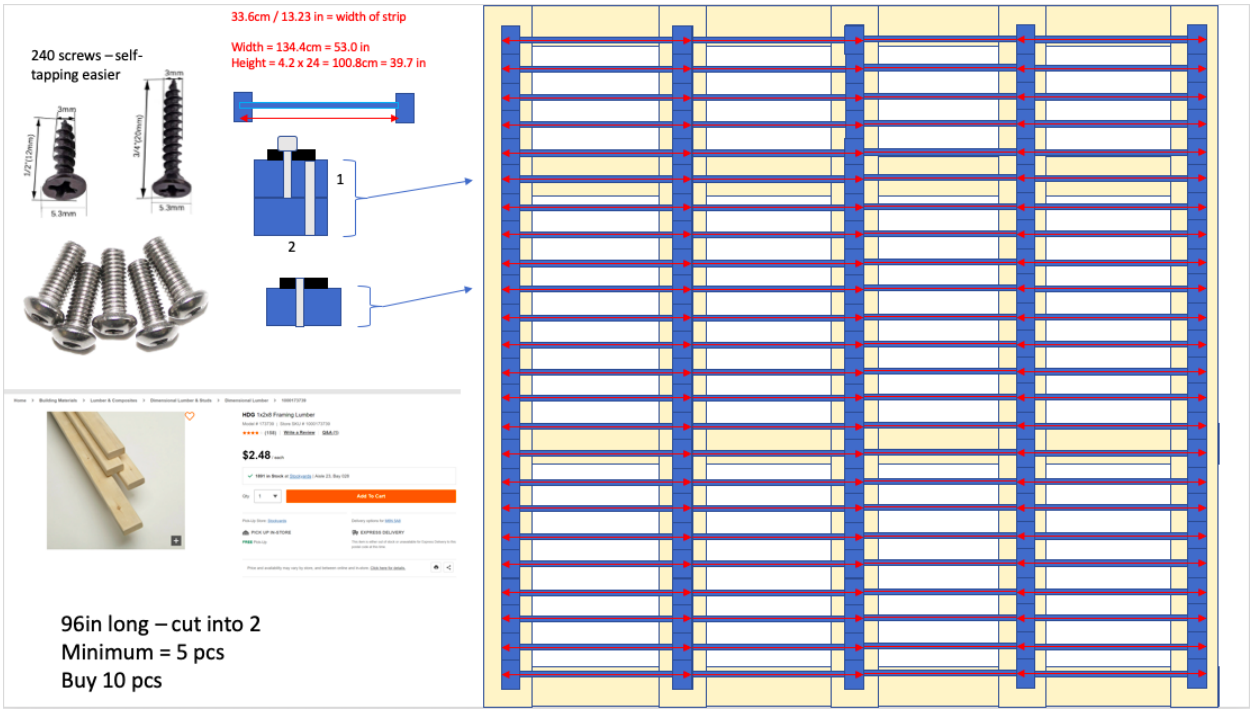
The LED buffer is stored in an 8-bit 2D array where each row represents a set of 8 pixels connected to the port pins, each column is the color bit value in order of the correct output with green first and white last, and each byte element is the bit written to the port and subsequently its 8 pins. This array is then read and sent out sequentially. To change a specific single pixel value with an RGB configuration to the configuration mentioned above, some complex bit writing is executed which unfortunately takes a few thousand clock cycles just to change one pixel to a specified color value. So, updating all 768 LEDs with this command would result in low performance which is under 30 frames. Many attempts were made to make this conversion more efficient but to no avail.

A third option of controlling the LEDs is to calculate and send the bits, saving a lot of SRAM space as no buffer is needed. The downside is that it limits the matrix to only display patterns formed with for loops and math, so no elaborate pictures can be shown. Instead, this is a good way to generate all sorts of gradients such as a rainbow pattern and other custom ones. Also, parallel output control does not work as the extra bit math needed to translate rgb values into values to write to the port takes longer than 80 microseconds, which as mentioned previously would latch the LEDs and prevent any more data from being sent. So, only a serial output works (though the rest of the port pins can be connected to repeat the data sent out).

To create a modular build, the PCBs were designed to fit with female to male connectors for easy configuration. Custom sizes can be made with LEDs arranged in strips of 8 with enough space for pingpong balls and connector boards that can route data signals in a variety of directions using jumper connectors. These connector boards were made for different parts of the wall. Large traces and ground planes are needed to facilitate large current draw needed by the LEDs. 96 strips were manufactured for a total of 768 LEDs and they were arranged horizontally in a 24 down by 4 across configuration. Each connector board holds the strips with female headers so 120 of them were produced for 5 columns of 24 with each holding a strip. Boards were connected to each other through removable wires in female headers to connect power, ground and data signals. Such features allow for a user to create any sized matrix. The same principle of modularity was applied to smaller PCB strips to complement the large LED wall. These strips have a single base connector and a strip on top which like the other larger strips can be rotated to send data from the left or right. A case and diffuser with a grid for the smaller matrix was designed in fusion which fits an 8 by 8 array of pixels.



PCBs were created in EAGLE and panelized with the SparkFun ULP where they were sent to JLCPCB. The benefit of panelization is the reduced cost for more PCBs, but the panel has to be under 100 mm by 100 mm in order to pay the lowest amount.



The PCBs of the large wall did not hold well by themselves and needed to be mounted to a support. A large wooden skeleton frame was planned and built. Each vertical strip contained a column of connectors and underneath, horizontal supporting boards were screwed together for better stability. A frame rather than a solid board made carrying and building the wall easier and cheaper though mounting the power supply was out of the question. To make sure that the frame was measured properly, a row of LEDs and connectors were soldered and assembled to check for squareness. Drilling the holes and then screwing the wood screws in to connect the wood supports was the ideal choice for no wood splits and easier securement. After building the frame, All 120 connectors were soldered and screwed onto the five vertical strips. The LED strips were plugged in as the connectors were mounted to assure that everything fit. Thick power lines were wired to terminal blocks and long data lines from the microcontroller to the connector board JST connector needed a shielded cable connection to prevent interference which messes up the delicate neopixel timing.

Code

Arduino Nano

```
// PROJECT  : Control SK6812s with Arduino NANO
// PURPOSE  : Medium ISP
// COURSE   : ICS4U
// AUTHOR   : Xander Chin
// DATE     : Feburary 20, 2022
// MCU      : ATmega328P
// STATUS   : Working
// REFERENCE:

#define NUM_STRIPS 8
#define NUM_LEDS_PER_STRIP 2
#define NUM_LEDS (NUM_STRIPS*NUM_LEDS_PER_STRIP)

#define PIXEL_PORT    PORTD
#define PIXEL_DDR     DDRD
#define PIXEL_PINS    255   //all 8 pins

#define LATCH         1600
#define T_BLOCK       7       //5 clock cycles jittery

//columns -> bit number (g,r,b,w)(32 color bits per row = 32 real bytes)
//storage problem

uint8_t pixelData[60][32] = {
//G7 G6 G5 G4 G3 G2 G1 G0 R7 R6 R5 R4 R3 R2 R1 R0 B7 B6 B5 B4 B3 B2 B1 B0 W7 W6 W5 W4 W3 W2 W1
W0
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //1st pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //2nd pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //3rd pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //4th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //5th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //6th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //7th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //8th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //9th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //10th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //11th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //12th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //13th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //14th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  },   //15th pixel
  { 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0  }    //16th pixel
};

const uint8_t nyan1[96][32] PROGMEM = {
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,0,0,0,0,0,0,0,0},
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,0,0,0,0,0,0,0,0},
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,16},
{44,9,8,12,45,40,8,12,14,78,78,78,14,78,78,78,97,96,96,97,64,97,96,96,16,0,0,16,16,0,0,16},
{12,9,8,12,13,8,8,12,14,14,14,14,14,14,14,14,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0},
{0,1,0,0,1,0,0,0,2,2,2,2,2,2,2,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0},
{0,1,60,60,61,60,60,0,0,60,60,60,0,0,0,0,1,0,0,1,0,1,0,0,60,0,0,0,60,60,60,60},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,1,66,66,67,66,66,0,4,126,126,126,60,4,60,4,5,56,56,61,56,5,56,4,122,0,0,0,66,122,66,122},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
```

```
{0,129,66,66,195,66,66,0,32,126,126,126,60,32,60,32,161,28,28,189,28,161,28,32,94,0,0,0,66,94,
66,94},
{0,129,66,66,195,66,66,0,8,126,126,126,60,8,60,8,137,52,52,189,52,137,52,8,118,0,0,0,66,118,66
,118},
{0,129,66,66,195,66,66,0,16,126,126,126,60,16,60,16,145,44,44,189,44,145,44,16,110,0,0,0,66,11
0,66,110},
{0,129,66,66,195,66,66,0,0,78,78,78,12,0,12,0,129,12,12,141,12,129,12,0,78,0,0,0,66,78,66,78},
{0,129,66,66,195,66,66,0,0,70,70,70,4,0,4,0,129,4,4,133,4,129,4,0,118,0,0,48,114,70,66,118},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,62,0,0,56,26,6,2,62},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,126,0,0,120,90,6,2,126},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,126,0,0,120,122,6,2,126},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,118,0,0,112,114,6,2,118},
{0,1,2,2,3,2,2,0,0,14,14,14,12,0,12,0,1,12,12,13,12,1,12,0,126,0,0,112,114,14,2,126},
{0,129,2,2,131,2,2,0,0,14,14,14,12,0,12,0,129,12,12,141,12,129,12,0,126,0,0,112,114,14,2,126},
{0,129,2,2,131,2,2,0,0,14,14,14,12,0,12,0,129,12,12,141,12,129,12,0,126,0,0,112,114,14,2,126},
{0,1,12,12,13,12,12,0,0,12,12,12,0,0,0,0,1,0,0,1,0,1,0,0,124,0,0,112,124,12,12,124},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,112,0,0,112,112,0,0,112},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,32,32,0,32,32,32,0,135,0,0,135,0,135,32,0,56,0,0,56,24,0,0,56},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,195,0,0,195,0,0,0,0,32,32,0,32,32,32,0,195,0,0,195,0,195,32,0,56,0,0,56,24,0,0,56},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,40,0,0,40,40,0,0,40},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,24,16,16,24,24,16,16,24},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,56,0,0,56,56,0,0,56},
{0,129,6,6,135,6,6,0,0,6,6,6,0,0,0,0,129,0,0,129,0,129,0,0,54,0,0,48,54,6,6,54},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,22,0,0,16,18,6,2,22},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,0,0,129,0,0,0,4,6,6,6,6,4,6,4,133,2,2,135,2,133,2,4,50,0,0,48,48,2,0,50},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,14,0,0,8,8,6,0,14},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,62,16,16,56,56,22,16,62},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,54,0,0,48,48,6,0,54},
{0,129,0,0,129,0,0,0,0,14,14,14,14,0,14,0,129,14,14,143,14,129,14,0,14,0,0,0,0,14,0,14},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,32,62,62,62,62,32,62,32,161,30,30,191,30,161,30,32,30,0,0,0,0,30,0,30},
{0,129,2,2,131,2,2,0,16,62,62,62,60,16,60,16,145,44,44,189,44,145,44,16,46,0,0,0,2,46,2,46},
{0,129,34,34,163,34,34,0,0,62,62,62,28,0,28,0,129,28,28,157,28,129,28,0,62,0,0,0,34,62,34,62},
{0,129,62,62,191,62,62,0,0,62,62,62,0,0,0,0,129,0,0,129,0,129,0,0,126,0,0,64,126,62,62,126},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,64,0,0,64,64,0,0,64},
{44,137,8,12,173,40,8,12,14,14,14,14,14,14,14,14,161,32,32,161,0,161,32,32,80,0,0,80,80,0,0,80
},
{36,129,0,4,165,32,0,4,6,6,6,6,6,6,6,6,161,32,32,161,0,161,32,32,16,0,0,16,16,0,0,16},
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,0,0,0,0,0,0,0,0},
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,8,0,0,8,8,0,0,8},
{52,145,16,20,181,48,16,20,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,8,0,0,8,8,0,0,8},
{22,196,4,6,214,20,4,6,7,39,39,39,7,39,39,39,240,48,48,240,32,240,48,48,0,0,0,0,0,0,0,0},
{22,196,4,6,214,20,4,6,7,39,39,39,7,39,39,39,240,48,48,240,32,240,48,48,8,0,0,8,8,0,0,8},
{6,196,4,6,198,4,4,6,7,39,39,39,7,39,39,39,224,32,32,224,32,224,32,32,8,0,0,8,8,0,0,8},
{6,132,4,6,134,4,4,6,7,39,39,39,7,39,39,39,160,32,32,160,32,160,32,32,0,0,0,0,0,0,0,0},
{36,129,0,4,165,32,0,4,6,6,6,6,6,6,6,6,161,32,32,161,0,161,32,32,64,0,0,64,64,0,0,64},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,64,0,0,64,64,0,0,64},
{0,129,62,62,191,62,62,0,0,62,62,62,0,0,0,0,129,0,0,129,0,129,0,0,62,0,0,0,62,62,62,62},
{0,128,34,34,162,34,34,0,0,62,62,62,28,0,28,0,128,28,28,156,28,128,28,0,62,0,0,0,34,62,34,62},
{0,128,32,32,160,32,32,0,0,126,126,126,30,64,94,64,192,94,94,222,94,192,94,64,62,0,0,0,32,62,3
2,62},
{0,128,0,0,128,0,0,0,0,62,62,62,62,0,62,0,128,62,62,190,62,128,62,0,62,0,0,0,0,62,0,62},
{0,128,0,0,128,0,0,0,8,62,62,62,62,8,62,8,136,54,54,190,54,136,54,8,118,0,0,64,64,54,0,118},
{0,128,0,0,128,0,0,0,0,62,62,62,62,0,62,0,128,62,62,190,62,128,62,0,126,0,0,64,64,62,0,126},
```

```
{0,128,0,0,128,0,0,0,0,62,62,62,62,0,62,0,128,62,62,190,62,128,62,0,62,0,0,0,0,62,0,62},
{0,192,0,0,192,0,0,0,0,62,62,62,62,0,62,0,192,62,62,254,62,192,62,0,62,0,0,0,0,62,0,62},
{0,192,0,0,192,0,0,0,0,2,62,62,62,62,2,62,2,194,60,60,254,60,194,60,2,60,0,0,0,0,60,0,60},
{0,192,0,0,192,0,0,0,0,0,46,46,46,46,0,46,0,192,46,46,238,46,192,46,0,46,0,0,0,0,46,0,46},
{0,192,0,0,192,0,0,0,0,0,2,2,2,2,0,2,0,192,2,2,194,2,192,2,0,18,0,0,16,16,2,0,18},
{0,192,0,0,192,0,0,0,0,2,2,2,2,2,2,2,2,194,0,0,194,0,194,0,2,60,0,0,60,60,0,0,60},
{0,128,0,0,128,0,0,0,0,0,2,2,2,2,0,2,0,128,2,2,130,2,128,2,0,62,0,0,60,60,2,0,62},
{0,128,0,0,128,0,0,0,0,0,2,2,2,2,0,2,0,128,2,2,130,2,128,2,0,106,0,0,104,104,2,0,106},
{0,128,0,0,128,0,0,0,0,0,6,6,6,6,0,6,0,128,6,6,134,6,128,6,0,78,0,0,72,72,6,0,78},
{0,128,0,0,128,0,0,0,0,0,6,6,6,6,0,6,0,128,6,6,134,6,128,6,0,30,0,0,0,24,24,6,0,30},
{0,192,0,0,192,0,0,0,0,0,6,6,6,6,0,6,0,192,6,6,198,6,192,6,0,30,0,0,0,24,24,6,0,30},
{0,128,2,2,130,2,2,0,0,0,6,6,6,6,4,0,4,0,128,4,4,132,4,128,4,0,30,0,0,0,24,24,26,6,2,30},
{0,129,6,6,135,6,6,0,0,0,6,6,6,6,0,0,0,0,129,0,0,129,0,129,0,0,78,0,0,0,72,78,6,6,78},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,88,0,0,0,88,88,0,0,88},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,8,0,0,0,8,8,0,0,8},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,44,0,0,0,44,44,0,0,44},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,60,0,0,0,60,60,0,0,60},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,16,0,0,0,16,16,0,0,16},
{0,239,0,0,239,0,0,0,0,0,0,0,0,0,0,0,0,0,239,0,0,239,0,239,0,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0,0}
};

const uint8_t nyan2[96][32] PROGMEM {
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,0,0,0,0,0,0,0,0},
{12,137,8,12,141,8,8,12,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0},
{12,9,8,12,13,8,8,12,14,78,78,78,14,78,78,78,65,64,64,65,64,65,64,64,0,0,0,0,0,0,0,0},
{0,1,0,0,1,0,0,0,2,2,2,2,2,2,2,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0},
{0,1,60,60,61,60,60,0,0,60,60,60,0,0,0,0,1,0,0,1,0,1,0,0,60,0,0,0,60,60,60,60},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,66,66,195,66,66,0,4,126,126,126,60,4,60,4,133,56,56,189,56,133,56,4,122,0,0,0,66,122,66
,122},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,1,66,66,67,66,66,0,32,126,126,126,60,32,60,32,33,28,28,61,28,33,28,32,94,0,0,0,66,94,66,94}
,
{0,129,66,66,195,66,66,0,8,126,126,126,60,8,60,8,137,52,52,189,52,137,52,8,118,0,0,0,66,118,66
,118},
{0,129,66,66,195,66,66,0,16,126,126,126,60,16,60,16,145,44,44,189,44,145,44,16,110,0,0,0,66,11
0,66,110},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,66,66,195,66,66,0,0,78,78,78,12,0,12,0,129,12,12,141,12,129,12,0,78,0,0,0,66,78,66,78},
{0,129,66,66,195,66,66,0,0,70,70,70,4,0,4,0,129,4,4,133,4,129,4,0,118,0,0,48,114,70,66,118},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,62,0,0,56,26,6,2,62},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,126,0,0,120,90,6,2,126},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,126,0,0,120,122,6,2,126},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,118,0,0,112,114,6,2,118},
{0,1,2,2,3,2,2,0,0,14,14,14,12,0,12,0,1,12,12,13,12,1,12,0,126,0,0,112,114,14,2,126},
{0,129,2,2,131,2,2,0,0,14,14,14,12,0,12,0,129,12,12,141,12,129,12,0,126,0,0,112,114,14,2,126},
{0,129,12,12,141,12,12,0,0,12,12,12,0,0,0,0,129,0,0,129,0,129,0,0,124,0,0,112,124,12,12,124},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,112,0,0,112,112,0,0,112},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,112,0,0,112,112,0,0,112},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,0,32,32,0,32,32,32,0,135,0,0,135,0,135,32,0,56,0,0,56,24,0,0,56},
{0,199,0,0,199,0,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,195,0,0,195,0,0,0,0,32,32,0,32,32,32,0,195,0,0,195,0,195,32,0,56,0,0,56,24,0,0,56},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,40,0,0,40,40,0,0,40},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,24,16,16,24,24,16,16,24},
```

```
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,56,0,0,56,56,0,0,56},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,48,0,0,48,48,0,0,48},
{0,129,6,6,135,6,6,0,0,6,6,6,0,0,0,0,129,0,0,129,0,129,0,0,22,0,0,16,22,6,6,22},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,0,0,129,0,0,0,4,6,6,6,6,4,6,4,133,2,2,135,2,133,2,4,10,0,0,8,8,2,0,10},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,62,16,16,56,56,22,16,62},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,54,0,0,48,48,6,0,54},
{0,129,0,0,129,0,0,0,0,14,14,14,14,0,14,0,129,14,14,143,14,129,14,0,14,0,0,0,0,14,0,14},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,32,62,62,62,62,32,62,32,161,30,30,191,30,161,30,32,30,0,0,0,0,30,0,30},
{0,129,2,2,131,2,2,0,16,62,62,62,60,16,60,16,145,44,44,189,44,145,44,16,46,0,0,0,2,46,2,46},
{0,129,34,34,163,34,34,0,0,62,62,62,28,0,28,0,129,28,28,157,28,129,28,0,62,0,0,0,34,62,34,62},
{0,129,62,62,191,62,62,0,0,62,62,62,0,0,0,0,129,0,0,129,0,129,0,0,126,0,0,64,126,62,62,126},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,64,0,0,64,64,0,0,64},
{44,137,8,12,173,40,8,12,14,14,14,14,14,14,14,14,161,32,32,161,0,161,32,32,16,0,0,16,16,0,0,16
},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,16},
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,0,0,0,0,0,0,0,0},
{22,196,4,6,214,20,4,6,7,39,39,39,7,39,39,39,240,48,48,240,32,240,48,48,8,0,0,8,8,0,0,8},
{6,196,4,6,198,4,4,6,7,39,39,39,7,39,39,39,224,32,32,224,32,224,32,32,8,0,0,8,8,0,0,8},
{6,196,4,6,198,4,4,6,7,39,39,39,7,39,39,39,224,32,32,224,32,224,32,32,0,0,0,0,0,0,0,0},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,137,8,12,173,40,8,12,14,14,14,14,14,14,14,14,161,32,32,161,0,161,32,32,16,0,0,16,16,0,0,16
},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,64,0,0,64,64,0,0,64},
{0,129,62,62,191,62,62,0,0,62,62,62,0,0,0,0,129,0,0,129,0,129,0,0,126,0,0,64,126,62,62,126},
{0,128,34,34,162,34,34,0,0,62,62,62,28,0,28,0,128,28,28,156,28,128,28,0,62,0,0,0,34,62,34,62},
{0,192,32,32,224,32,32,0,0,62,62,62,30,0,30,0,192,30,30,222,30,192,30,0,62,0,0,0,32,62,32,62},
{0,128,0,0,128,0,0,0,0,62,62,62,62,0,62,0,128,62,62,190,62,128,62,0,62,0,0,0,0,62,0,62},
{0,128,0,0,128,0,0,0,8,62,62,62,62,8,62,8,136,54,54,190,54,136,54,8,118,0,0,64,64,54,0,118},
{0,128,0,0,128,0,0,0,0,62,62,62,62,0,62,0,128,62,62,190,62,128,62,0,126,0,0,64,64,62,0,126},
{0,128,0,0,128,0,0,0,0,62,62,62,62,0,62,0,128,62,62,190,62,128,62,0,62,0,0,0,0,62,0,62},
{0,192,0,0,192,0,0,0,0,62,62,62,62,0,62,0,192,62,62,254,62,192,62,0,62,0,0,0,0,62,0,62},
{0,192,0,0,192,0,0,0,2,62,62,62,62,2,62,2,194,60,60,254,60,194,60,2,60,0,0,0,0,60,0,60},
{0,192,0,0,192,0,0,0,0,62,62,62,62,0,62,0,192,62,62,254,62,192,62,0,62,0,0,0,0,62,0,62},
{0,192,0,0,192,0,0,0,0,46,46,46,46,0,46,0,192,46,46,238,46,192,46,0,46,0,0,0,0,46,0,46},
{0,192,0,0,192,0,0,0,2,2,2,2,2,2,2,2,194,0,0,194,0,194,0,2,16,0,0,16,16,0,0,16},
{0,192,0,0,192,0,0,0,0,2,2,2,2,0,2,0,192,2,2,194,2,192,2,0,62,0,0,60,60,2,0,62},
{0,128,0,0,128,0,0,0,0,2,2,2,2,0,2,0,128,2,2,130,2,128,2,0,62,0,0,60,60,2,0,62},
{0,128,0,0,128,0,0,0,0,2,2,2,2,0,2,0,128,2,2,130,2,128,2,0,106,0,0,104,104,2,0,106},
{0,128,0,0,128,0,0,0,0,6,6,6,6,0,6,0,128,6,6,134,6,128,6,0,78,0,0,72,72,6,0,78},
{0,128,0,0,128,0,0,0,0,6,6,6,6,0,6,0,128,6,6,134,6,128,6,0,30,0,0,24,24,6,0,30},
{0,192,2,2,194,2,2,0,0,6,6,6,4,0,4,0,192,4,4,196,4,192,4,0,30,0,0,24,26,6,2,30},
{0,129,6,6,135,6,6,0,0,6,6,6,0,0,0,0,129,0,0,129,0,129,0,0,30,0,0,24,30,6,6,30},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,72,0,0,72,72,0,0,72},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,88,0,0,88,88,0,0,88},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,8,0,0,8,8,0,0,8},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,44,0,0,44,44,0,0,44},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,60,0,0,60,60,0,0,60},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,16,0,0,16,16,0,0,16},
{0,239,0,0,239,0,0,0,0,0,0,0,0,0,0,0,239,0,0,239,0,239,0,0,0,0,0,0,0,0,0,0,}
};

const uint8_t nyan3[96][32] PROGMEM = {
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0}
,
```

```
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{0,1,0,0,1,0,0,0,2,2,2,2,2,2,2,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0},
{0,1,124,124,125,124,124,0,2,126,126,126,2,2,2,2,1,0,0,1,0,1,0,0,252,0,0,128,252,124,124,252},
{0,1,68,68,69,68,68,0,0,124,124,124,56,0,56,0,1,56,56,57,56,1,56,0,252,0,0,128,196,124,68,252}
,
{0,1,64,64,65,64,64,0,0,124,124,124,60,0,60,0,1,60,60,61,60,1,60,0,124,0,0,0,64,124,64,124},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,1,0,0,1,0,0,0,16,124,124,124,124,16,124,16,17,108,108,125,108,17,108,16,108,0,0,0,0,108,0,1
08},
{0,1,0,0,1,0,0,0,0,124,124,124,124,0,124,0,1,124,124,125,124,1,124,0,252,0,0,128,128,124,0,252
},
{0,1,0,0,1,0,0,0,0,124,124,124,124,0,124,0,1,124,124,125,124,1,124,0,252,0,0,128,128,124,0,252
},
{0,1,0,0,1,0,0,0,0,124,124,124,124,0,124,0,1,124,124,125,124,1,124,0,124,0,0,0,0,124,0,124},
{0,129,0,0,129,0,0,0,4,124,124,124,124,4,124,4,133,120,120,253,120,133,120,4,120,0,0,0,0,120,0
,120},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,0,92,92,92,92,0,92,0,129,92,92,221,92,129,92,0,92,0,0,0,0,92,0,92},
{0,129,0,0,129,0,0,0,4,4,4,4,4,4,4,4,133,0,0,133,0,133,0,4,32,0,0,32,32,0,0,32},
{0,129,0,0,129,0,0,0,0,4,4,4,4,0,4,0,129,4,4,133,4,129,4,0,124,0,0,120,120,4,0,124},
{0,129,0,0,129,0,0,0,0,4,4,4,4,0,4,0,129,4,4,133,4,129,4,0,124,0,0,120,120,4,0,124},
{0,1,0,0,1,0,0,0,0,4,4,4,4,0,4,0,1,4,4,5,4,1,4,0,84,0,0,80,80,4,0,84},
{0,1,0,0,1,0,0,0,0,12,12,12,12,0,12,0,1,12,12,13,12,1,12,0,156,0,0,144,144,12,0,156},
{0,1,0,0,1,0,0,0,0,12,12,12,12,0,12,0,1,12,12,13,12,1,12,0,188,0,0,176,176,12,0,188},
{0,1,4,4,5,4,4,0,0,12,12,12,8,0,8,0,1,8,8,9,8,1,8,0,60,0,0,48,52,12,4,60},
{0,131,12,12,143,12,12,0,0,12,12,12,0,0,0,0,131,0,0,131,0,131,0,0,60,0,0,48,60,12,12,60},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,16,0,0,16,16,0,0,16},
{0,15,0,0,15,0,0,0,0,0,0,0,0,0,0,15,0,0,15,0,15,0,0,176,0,0,176,176,0,0,176},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,144,0,0,144,144,0,0,144},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,88,0,0,88,88,0,0,88},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,32,0,0,32,32,0,0,32},
{0,223,0,0,223,0,0,0,0,0,0,0,0,0,0,223,0,0,223,0,223,0,0,0,0,0,0,0,0,0,0},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,135,0,0,135,0,0,0,0,32,32,0,32,32,32,0,135,0,0,135,0,135,32,0,56,0,0,56,24,0,0,56},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,120,0,0,120,120,0,0,120},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,120,0,0,120,120,0,0,120},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,112,0,0,112,112,0,0,112},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,112,0,0,112,112,0,0,112},
{0,129,12,12,141,12,12,0,0,12,12,12,0,0,0,0,129,0,0,129,0,129,0,0,124,0,0,112,124,12,12,124},
{0,1,2,2,3,2,2,0,0,14,14,14,12,0,12,0,1,12,12,13,12,1,12,0,126,0,0,112,114,14,2,126},
{0,1,2,2,3,2,2,0,0,14,14,14,12,0,12,0,1,12,12,13,12,1,12,0,126,0,0,112,114,14,2,126},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,118,0,0,112,114,6,2,118},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,126,0,0,120,122,6,2,126},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,126,0,0,120,90,6,2,126},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,62,0,0,56,26,6,2,62},
{0,129,66,66,195,66,66,0,0,70,70,70,4,0,4,0,129,4,4,133,4,129,4,0,118,0,0,48,114,70,66,118},
{0,129,66,66,195,66,66,0,0,78,78,78,12,0,12,0,129,12,12,141,12,129,12,0,78,0,0,0,66,78,66,78},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,66,66,195,66,66,0,16,126,126,126,60,16,60,16,145,44,44,189,44,145,44,16,110,0,0,0,66,11
0,66,110},
{0,1,66,66,67,66,66,0,8,126,126,126,60,8,60,8,9,52,52,61,52,9,52,8,118,0,0,0,66,118,66,118},
{0,1,66,66,67,66,66,0,32,126,126,126,60,32,60,32,33,28,28,61,28,33,28,32,94,0,0,0,66,94,66,94}
,
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
```

```
{0,129,66,66,195,66,66,0,4,126,126,126,60,4,60,4,133,56,56,189,56,133,56,4,122,0,0,0,66,122,66
,122},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,1,60,60,61,60,60,0,0,60,60,60,0,0,0,0,1,0,0,1,0,1,0,0,60,0,0,0,60,60,60,60},
{0,1,0,0,1,0,0,0,2,2,2,2,2,2,2,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0},
{12,137,8,12,141,8,8,12,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0}
,
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0}
,
{12,137,8,12,141,8,8,12,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0},
{12,137,8,12,141,8,8,12,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0},
{12,137,8,12,141,8,8,12,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0},
{14,204,12,14,206,12,12,14,7,39,39,39,7,39,39,39,224,32,32,224,32,224,32,32,0,0,0,0,0,0,0,0},
{0,128,0,0,128,0,0,0,1,1,1,1,1,1,1,128,0,0,128,0,128,0,0,0,0,0,0,0,0,0},
{0,128,62,62,190,62,62,0,1,63,63,63,1,1,1,1,128,0,0,128,0,128,0,0,126,0,0,64,126,62,62,126},
{0,128,34,34,162,34,34,0,1,63,63,63,29,1,29,1,128,28,28,156,28,128,28,0,62,0,0,0,34,62,34,62},
{0,128,2,2,130,2,2,0,17,63,63,63,61,17,61,17,144,44,44,188,44,144,44,16,46,0,0,0,2,46,2,46},
{0,128,0,0,128,0,0,0,33,63,63,63,63,33,63,33,160,30,30,190,30,160,30,32,30,0,0,0,0,30,0,30},
{0,128,0,0,128,0,0,0,1,63,63,63,63,1,63,1,128,62,62,190,62,128,62,0,62,0,0,0,0,62,0,62},
{0,128,0,0,128,0,0,0,1,63,63,63,63,1,63,1,128,62,62,190,62,128,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,14,14,14,14,0,14,0,129,14,14,143,14,129,14,0,14,0,0,0,0,14,0,14},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,54,0,0,48,48,6,0,54},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,62,16,16,56,56,22,16,62},
{0,129,0,0,129,0,0,0,4,6,6,6,6,4,6,4,133,2,2,135,2,133,2,4,10,0,0,8,8,2,0,10},
{0,129,2,2,131,2,2,0,0,6,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,2,2,131,2,2,0,0,6,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,6,6,135,6,6,0,0,6,6,6,6,0,0,0,0,129,0,0,129,0,129,0,0,22,0,0,16,22,6,6,22},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,48,0,0,48,48,0,0,48},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,56,0,0,56,56,0,0,56},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,24,16,16,24,24,16,16,24},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,40,0,0,40,40,0,0,40},
{0,195,0,0,195,0,0,0,0,32,32,0,32,32,32,0,195,0,0,195,0,195,32,0,56,0,0,56,24,0,0,56},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0}
};

const uint8_t nyan4[96][32] PROGMEM = {
{28,153,24,28,157,24,24,28,14,14,14,14,14,14,14,14,129,0,0,129,0,129,0,0,0,0,0,0,0,0,0,0},
{28,153,24,28,157,24,24,28,14,14,14,14,14,14,14,14,129,0,0,129,0,129,0,0,32,0,0,32,32,0,0,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,25,24,28,29,24,24,28,14,78,78,78,14,78,78,78,65,64,64,65,64,65,64,64,32,0,0,32,32,0,0,32},
{0,1,0,0,1,0,0,0,2,2,2,2,2,2,2,1,0,0,1,0,1,0,0,128,0,0,128,128,0,0,128},
{0,1,124,124,125,124,124,0,2,126,126,126,2,2,2,2,1,0,0,1,0,1,0,0,252,0,0,128,252,124,124,252},
{0,1,68,68,69,68,68,0,0,124,124,124,56,0,56,0,1,56,56,57,56,1,56,0,124,0,0,0,68,124,68,124},
{0,129,64,64,193,64,64,0,0,124,124,124,60,0,60,0,129,60,60,189,60,129,60,0,124,0,0,0,64,124,64
,124},
{0,1,0,0,1,0,0,0,0,124,124,124,124,0,124,0,1,124,124,125,124,1,124,0,124,0,0,0,0,124,0,124},
```

```
{0,1,0,0,1,0,0,0,16,124,124,124,124,16,124,16,17,108,108,125,108,17,108,16,236,0,0,128,128,108
,0,236},
{0,1,0,0,1,0,0,0,0,124,124,124,124,0,124,0,1,124,124,125,124,1,124,0,252,0,0,128,128,124,0,252
},
{0,1,0,0,1,0,0,0,0,124,124,124,124,0,124,0,1,124,124,125,124,1,124,0,124,0,0,0,0,124,0,124},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,4,124,124,124,124,4,124,4,133,120,120,253,120,133,120,4,120,0,0,0,0,120,0
,120},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,0,92,92,92,92,0,92,0,129,92,92,221,92,129,92,0,92,0,0,0,0,92,0,92},
{0,129,0,0,129,0,0,0,4,4,4,4,4,4,4,4,133,0,0,133,0,133,0,4,32,0,0,32,32,0,0,32},
{0,129,0,0,129,0,0,0,0,4,4,4,4,0,4,0,129,4,4,133,4,129,4,0,124,0,0,120,120,4,0,124},
{0,1,0,0,1,0,0,0,0,4,4,4,4,0,4,0,1,4,4,5,4,1,4,0,124,0,0,120,120,4,0,124},
{0,1,0,0,1,0,0,0,0,4,4,4,4,0,4,0,1,4,4,5,4,1,4,0,212,0,0,208,208,4,0,212},
{0,1,0,0,1,0,0,0,0,12,12,12,12,0,12,0,1,12,12,13,12,1,12,0,156,0,0,144,144,12,0,156},
{0,1,0,0,1,0,0,0,0,12,12,12,12,0,12,0,1,12,12,13,12,1,12,0,60,0,0,48,48,12,0,60},
{0,129,4,4,133,4,4,0,0,12,12,12,8,0,8,0,129,8,8,137,8,129,8,0,60,0,0,48,52,12,4,60},
{0,3,12,12,15,12,12,0,0,12,12,12,0,0,0,0,3,0,0,3,0,3,0,0,60,0,0,48,60,12,12,60},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,144,0,0,144,144,0,0,144},
{0,15,0,0,15,0,0,0,0,0,0,0,0,0,0,15,0,0,15,0,15,0,0,176,0,0,176,176,0,0,176},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,16,0,0,16,16,0,0,16},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,88,0,0,88,88,0,0,88},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,32,0,0,32,32,0,0,32},
{0,223,0,0,223,0,0,0,0,0,0,0,0,0,0,223,0,0,223,0,223,0,0,0,0,0,0,0,0,0,0},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,135,0,0,135,0,0,0,0,32,32,0,32,32,32,0,135,0,0,135,0,135,32,0,56,0,0,56,24,0,0,56},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,120,0,0,120,120,0,0,120},
{0,7,0,0,7,0,0,0,0,0,0,0,0,0,0,7,0,0,7,0,7,0,0,112,0,0,112,112,0,0,112},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,112,0,0,112,112,0,0,112},
{0,129,12,12,141,12,12,0,0,12,12,12,0,0,0,0,129,0,0,129,0,129,0,0,124,0,0,112,124,12,12,124},
{0,129,2,2,131,2,2,0,0,14,14,14,12,0,12,0,129,12,12,141,12,129,12,0,126,0,0,112,114,14,2,126},
{0,1,2,2,3,2,2,0,0,14,14,14,12,0,12,0,1,12,12,13,12,1,12,0,126,0,0,112,114,14,2,126},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,118,0,0,112,114,6,2,118},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,126,0,0,120,122,6,2,126},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,126,0,0,120,90,6,2,126},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,62,0,0,56,26,6,2,62},
{0,129,66,66,195,66,66,0,0,70,70,70,4,0,4,0,129,4,4,133,4,129,4,0,118,0,0,48,114,70,66,118},
{0,129,66,66,195,66,66,0,0,78,78,78,12,0,12,0,129,12,12,141,12,129,12,0,78,0,0,0,66,78,66,78},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,66,66,195,66,66,0,16,126,126,126,60,16,60,16,145,44,44,189,44,145,44,16,110,0,0,0,66,11
0,66,110},
{0,129,66,66,195,66,66,0,8,126,126,126,60,8,60,8,137,52,52,189,52,137,52,8,118,0,0,0,66,118,66
,118},
{0,1,66,66,67,66,66,0,32,126,126,126,60,32,60,32,33,28,28,61,28,33,28,32,94,0,0,0,66,94,66,94}
,
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,129,66,66,195,66,66,0,4,126,126,126,60,4,60,4,133,56,56,189,56,133,56,4,122,0,0,0,66,122,66
,122},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,1,60,60,61,60,60,0,0,60,60,60,0,0,0,0,1,0,0,1,0,1,0,0,60,0,0,0,60,60,60,60},
{0,1,0,0,1,0,0,0,2,2,2,2,2,2,2,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0},
{12,9,8,12,13,8,8,12,14,78,78,78,14,78,78,78,65,64,64,65,64,65,64,64,0,0,0,0,0,0,0,0},
{12,137,8,12,141,8,8,12,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0}
,
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
```

```
{28,153,24,28,157,24,24,28,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0
,32},
{12,137,8,12,141,8,8,12,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,32,0,0,32,32,0,0,32
},
{12,137,8,12,141,8,8,12,14,78,78,78,14,78,78,78,193,64,64,193,64,193,64,64,0,0,0,0,0,0,0,0},
{14,204,12,14,206,12,12,14,7,39,39,39,7,39,39,39,224,32,32,224,32,224,32,32,16,0,0,16,16,0,0,1
6},
{14,140,12,14,142,12,12,14,7,39,39,39,7,39,39,39,160,32,32,160,32,160,32,32,16,0,0,16,16,0,0,1
6},
{0,128,0,0,128,0,0,0,1,1,1,1,1,1,1,128,0,0,128,0,128,0,0,64,0,0,64,64,0,0,64},
{0,128,62,62,190,62,62,0,1,63,63,63,1,1,1,1,128,0,0,128,0,128,0,0,126,0,0,64,126,62,62,126},
{0,128,34,34,162,34,34,0,1,63,63,63,29,1,29,1,128,28,28,156,28,128,28,0,62,0,0,0,34,62,34,62},
{0,128,2,2,130,2,2,0,17,63,63,63,61,17,61,17,144,44,44,188,44,144,44,16,46,0,0,0,2,46,2,46},
{0,128,0,0,128,0,0,0,33,63,63,63,63,33,63,33,160,30,30,190,30,160,30,32,30,0,0,0,0,30,0,30},
{0,128,0,0,128,0,0,0,1,63,63,63,63,1,63,1,128,62,62,190,62,128,62,0,62,0,0,0,0,62,0,62},
{0,128,0,0,128,0,0,0,1,63,63,63,63,1,63,1,128,62,62,190,62,128,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,14,14,14,14,0,14,0,129,14,14,143,14,129,14,0,14,0,0,0,0,14,0,14},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,54,0,0,48,48,6,0,54},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,62,16,16,56,56,22,16,62},
{0,129,0,0,129,0,0,0,4,6,6,6,6,4,6,4,133,2,2,135,2,133,2,4,10,0,0,8,8,2,0,10},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,6,6,135,6,6,0,0,6,6,6,0,0,0,0,129,0,0,129,0,129,0,0,22,0,0,16,22,6,6,22},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,48,0,0,48,48,0,0,48},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,56,0,0,56,56,0,0,56},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,24,16,16,24,24,16,16,24},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,40,0,0,40,40,0,0,40},
{0,195,0,0,195,0,0,0,0,32,32,0,32,32,32,0,195,0,0,195,0,195,32,0,56,0,0,56,24,0,0,56},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0}
};

const uint8_t nyan5[96][32] PROGMEM = {
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,9,8,12,45,40,8,12,14,78,78,78,14,78,78,78,97,96,96,97,64,97,96,96,0,0,0,0,0,0,0,0},
{12,9,8,12,13,8,8,12,14,78,78,78,14,78,78,78,65,64,64,65,64,65,64,64,128,0,0,128,128,0,0,128},
{12,9,8,12,13,8,8,12,14,14,14,14,14,14,14,14,1,0,0,1,0,1,0,0,128,0,0,128,128,0,0,128},
{0,1,0,0,1,0,0,0,2,2,2,2,2,2,2,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0},
{0,129,124,124,253,124,124,0,2,126,126,126,2,2,2,2,129,0,0,129,0,129,0,0,124,0,0,0,124,124,124
,124},
{0,1,68,68,69,68,68,0,0,124,124,124,56,0,56,0,1,56,56,57,56,1,56,0,124,0,0,0,68,124,68,124},
{0,1,64,64,65,64,64,0,0,124,124,124,60,0,60,0,1,60,60,61,60,1,60,0,252,0,0,128,192,124,64,252}
,
{0,1,0,0,1,0,0,0,0,124,124,124,124,0,124,0,1,124,124,125,124,1,124,0,252,0,0,128,128,124,0,252
},
{0,1,0,0,1,0,0,0,16,124,124,124,124,16,124,16,17,108,108,125,108,17,108,16,108,0,0,0,0,108,0,1
08},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,4,124,124,124,124,4,124,4,133,120,120,253,120,133,120,4,120,0,0,0,0,120,0
,120},
{0,129,0,0,129,0,0,0,0,92,92,92,92,0,92,0,129,92,92,221,92,129,92,0,92,0,0,0,0,92,0,92},
{0,129,0,0,129,0,0,0,0,4,4,4,4,0,4,0,129,4,4,133,4,129,4,0,36,0,0,32,32,4,0,36},
{0,1,0,0,1,0,0,0,4,4,4,4,4,4,4,5,0,0,5,0,5,0,4,120,0,0,120,120,0,0,120},
{0,1,0,0,1,0,0,0,0,4,4,4,4,0,4,0,1,4,4,5,4,1,4,0,252,0,0,248,248,4,0,252},
{0,1,0,0,1,0,0,0,0,4,4,4,4,0,4,0,1,4,4,5,4,1,4,0,212,0,0,208,208,4,0,212},
```

```
{0,1,0,0,1,0,0,0,0,12,12,12,12,0,12,0,1,12,12,13,12,1,12,0,28,0,0,16,16,12,0,28},
{0,129,0,0,129,0,0,0,0,12,12,12,12,0,12,0,129,12,12,141,12,129,12,0,60,0,0,48,48,12,0,60},
{0,1,0,0,1,0,0,0,0,12,12,12,12,0,12,0,1,12,12,13,12,1,12,0,60,0,0,48,48,12,0,60},
{0,1,4,4,5,4,4,0,0,12,12,12,8,0,8,0,1,8,8,9,8,1,8,0,188,0,0,176,180,12,4,188},
{0,3,12,12,15,12,12,0,0,12,12,12,0,0,0,0,3,0,0,3,0,3,0,0,156,0,0,144,156,12,12,156},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,48,0,0,48,48,0,0,48},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,16,0,0,16,16,0,0,16},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,88,0,0,88,88,0,0,88},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,32,0,0,32,32,0,0,32},
{0,223,0,0,223,0,0,0,0,0,0,0,0,0,0,0,223,0,0,223,0,223,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,135,0,0,135,0,0,0,0,32,32,0,32,32,32,0,135,0,0,135,0,135,32,0,56,0,0,56,24,0,0,56},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,112,0,0,112,112,0,0,112},
{0,1,12,12,13,12,12,0,0,12,12,12,0,0,0,0,1,0,0,1,0,1,0,0,124,0,0,112,124,12,12,124},
{0,1,2,2,3,2,2,0,0,14,14,14,12,0,12,0,1,12,12,13,12,1,12,0,126,0,0,112,114,14,2,126},
{0,129,2,2,131,2,2,0,0,14,14,14,12,0,12,0,129,12,12,141,12,129,12,0,126,0,0,112,114,14,2,126},
{0,129,2,2,131,2,2,0,0,14,14,14,12,0,12,0,129,12,12,141,12,129,12,0,126,0,0,112,114,14,2,126},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,118,0,0,112,114,6,2,118},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,126,0,0,120,122,6,2,126},
{0,1,2,2,3,2,2,0,0,38,38,6,36,32,36,0,1,4,4,5,4,1,36,0,126,0,0,120,90,6,2,126},
{0,129,2,2,131,2,2,0,0,38,38,6,36,32,36,0,129,4,4,133,4,129,36,0,62,0,0,56,26,6,2,62},
{0,129,66,66,195,66,66,0,0,70,70,70,4,0,4,0,129,4,4,133,4,129,4,0,118,0,0,48,114,70,66,118},
{0,129,66,66,195,66,66,0,0,78,78,78,12,0,12,0,129,12,12,141,12,129,12,0,78,0,0,0,66,78,66,78},
{0,129,66,66,195,66,66,0,16,126,126,126,60,16,60,16,145,44,44,189,44,145,44,16,110,0,0,0,66,11
0,66,110},
{0,129,66,66,195,66,66,0,8,126,126,126,60,8,60,8,137,52,52,189,52,137,52,8,118,0,0,0,66,118,66
,118},
{0,129,66,66,195,66,66,0,32,126,126,126,60,32,60,32,161,28,28,189,28,161,28,32,94,0,0,0,66,94,
66,94},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,1,66,66,67,66,66,0,4,126,126,126,60,4,60,4,5,56,56,61,56,5,56,4,122,0,0,0,66,122,66,122},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,60,60,189,60,60,0,0,60,60,60,0,0,0,0,129,0,0,129,0,129,0,0,60,0,0,0,60,60,60,60},
{0,129,0,0,129,0,0,0,2,2,2,2,2,2,2,2,129,0,0,129,0,129,0,0,0,0,0,0,0,0,0,0},
{44,9,8,12,45,40,8,12,14,14,14,14,14,14,14,14,33,32,32,33,0,33,32,32,0,0,0,0,0,0,0,0},
{44,9,8,12,45,40,8,12,14,14,14,14,14,14,14,14,33,32,32,33,0,33,32,32,16,0,0,16,16,0,0,16},
{44,9,8,12,45,40,8,12,14,78,78,78,14,78,78,78,97,96,96,97,64,97,96,96,16,0,0,16,16,0,0,16},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{6,196,4,6,198,4,4,6,7,39,39,39,7,39,39,39,224,32,32,224,32,224,32,32,0,0,0,0,0,0,0,0},
{6,196,4,6,198,4,4,6,7,39,39,39,7,39,39,39,224,32,32,224,32,224,32,32,0,0,0,0,0,0,0,0},
{6,132,4,6,134,4,4,6,7,39,39,39,7,39,39,39,160,32,32,160,32,160,32,32,0,0,0,0,0,0,0,0},
{14,140,12,14,142,12,12,14,7,39,39,39,7,39,39,39,160,32,32,160,32,160,32,32,64,0,0,64,64,0,0,6
4},
{44,137,8,12,173,40,8,12,14,14,14,14,14,14,14,14,161,32,32,161,0,161,32,32,80,0,0,80,80,0,0,80
},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,64,0,0,64,64,0,0,64},
{0,129,62,62,191,62,62,0,0,62,62,62,0,0,0,0,129,0,0,129,0,129,0,0,62,0,0,0,62,62,62,62},
{0,129,34,34,163,34,34,0,0,62,62,62,28,0,28,0,129,28,28,157,28,129,28,0,62,0,0,0,34,62,34,62},
{0,129,2,2,131,2,2,0,16,62,62,62,60,16,60,16,145,44,44,189,44,145,44,16,46,0,0,0,2,46,2,46},
{0,129,0,0,129,0,0,0,32,62,62,62,62,32,62,32,161,30,30,191,30,161,30,32,30,0,0,0,0,30,0,30},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,14,14,14,14,0,14,0,129,14,14,143,14,129,14,0,14,0,0,0,0,14,0,14},
```

```
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,54,0,0,48,48,6,0,54},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,34,34,2,34,32,34,0,129,2,2,131,2,129,34,0,58,0,0,56,24,2,0,58},
{0,129,0,0,129,0,0,0,0,6,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,62,16,16,56,56,22,16,62},
{0,129,0,0,129,0,0,0,0,6,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,14,0,0,8,8,6,0,14},
{0,129,0,0,129,0,0,0,4,6,6,6,6,4,6,4,133,2,2,135,2,133,2,4,50,0,0,48,48,2,0,50},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,22,0,0,16,18,6,2,22},
{0,129,6,6,135,6,6,0,0,6,6,6,0,0,0,0,129,0,0,129,0,129,0,0,54,0,0,48,54,6,6,54},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,56,0,0,56,56,0,0,56},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,24,16,16,24,24,16,16,24},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,40,0,0,40,40,0,0,40},
{0,195,0,0,195,0,0,0,0,32,32,0,32,32,32,0,195,0,0,195,0,195,32,0,56,0,0,56,24,0,0,56},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0}
};

const uint8_t nyan6[96][32] PROGMEM = {
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,137,8,12,173,40,8,12,14,78,78,78,14,78,78,78,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,1
6},
{44,9,8,12,45,40,8,12,14,78,78,78,14,78,78,78,97,96,96,97,64,97,96,96,0,0,0,0,0,0,0,0},
{12,9,8,12,13,8,8,12,14,78,78,78,14,78,78,78,65,64,64,65,64,65,64,64,128,0,0,128,128,0,0,128},
{12,9,8,12,13,8,8,12,14,14,14,14,14,14,14,14,14,1,0,0,1,0,1,0,0,128,0,0,128,128,0,0,128},
{0,1,0,0,1,0,0,0,2,2,2,2,2,2,2,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0},
{0,129,124,124,253,124,124,0,2,126,126,126,2,2,2,2,129,0,0,129,0,129,0,0,124,0,0,0,124,124,124
,124},
{0,1,68,68,69,68,68,0,0,124,124,124,56,0,56,0,1,56,56,57,56,1,56,0,124,0,0,0,68,124,68,124},
{0,1,64,64,65,64,64,0,0,124,124,124,60,0,60,0,1,60,60,61,60,1,60,0,252,0,0,128,192,124,64,252}
,
{0,1,0,0,1,0,0,0,0,124,124,124,124,0,124,0,1,124,124,125,124,1,124,0,252,0,0,128,128,124,0,252
},
{0,1,0,0,1,0,0,0,16,124,124,124,124,16,124,16,17,108,108,125,108,17,108,16,108,0,0,0,0,108,0,1
08},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,0,124,124,124,124,0,124,0,129,124,124,253,124,129,124,0,124,0,0,0,0,124,0
,124},
{0,129,0,0,129,0,0,0,4,124,124,124,124,4,124,4,133,120,120,253,120,133,120,4,120,0,0,0,0,120,0
,120},
{0,129,0,0,129,0,0,0,0,76,76,76,76,0,76,0,129,76,76,205,76,129,76,0,76,0,0,0,0,76,0,76},
{0,129,0,0,129,0,0,0,0,68,68,68,68,0,68,0,129,68,68,197,68,129,68,0,116,0,0,48,48,68,0,116},
{0,1,0,0,1,0,0,0,4,36,36,4,36,36,36,4,5,0,0,5,0,5,32,4,56,0,0,56,24,0,0,56},
{0,1,0,0,1,0,0,0,0,36,36,4,36,32,36,0,1,4,4,5,4,1,36,0,252,0,0,248,216,4,0,252},
{0,1,0,0,1,0,0,0,0,4,4,4,4,0,4,0,1,4,4,5,4,1,4,0,252,0,0,248,248,4,0,252},
{0,1,0,0,1,0,0,0,0,4,4,4,4,0,4,0,1,4,4,5,4,1,4,0,116,0,0,112,112,4,0,116},
{0,129,0,0,129,0,0,0,0,12,12,12,12,0,12,0,129,12,12,141,12,129,12,0,124,0,0,112,112,12,0,124},
{0,1,0,0,1,0,0,0,0,12,12,12,12,0,12,0,1,12,12,13,12,1,12,0,124,0,0,112,112,12,0,124},
{0,1,4,4,5,4,4,0,0,12,12,12,8,0,8,0,1,8,8,9,8,1,8,0,252,0,0,240,244,12,4,252},
{0,3,12,12,15,12,12,0,0,12,12,12,0,0,0,0,3,0,0,3,0,3,0,0,252,0,0,240,252,12,12,252},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,112,0,0,112,112,0,0,112},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,120,0,0,120,120,0,0,120},
{0,135,0,0,135,0,0,0,0,32,32,0,32,32,32,0,135,0,0,135,0,135,32,0,56,0,0,56,24,0,0,56},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0},
{0,207,0,0,207,0,0,0,0,0,0,0,0,0,0,207,0,0,207,0,207,0,0,0,0,0,0,0,0,0,0},
{0,199,0,0,199,0,0,0,0,32,32,0,32,32,32,0,199,0,0,199,0,199,32,0,48,0,0,48,16,0,0,48},
{0,195,0,0,195,0,0,0,0,32,32,0,32,32,32,0,195,0,0,195,0,195,32,0,56,0,0,56,24,0,0,56},
{0,131,0,0,131,0,0,0,0,0,0,0,0,0,0,131,0,0,131,0,131,0,0,40,0,0,40,40,0,0,40},
```

```
{0,135,0,0,135,0,0,0,0,0,0,0,0,0,0,0,135,0,0,135,0,135,0,0,24,16,16,24,24,16,16,24},
{0,3,0,0,3,0,0,0,0,0,0,0,0,0,0,0,3,0,0,3,0,3,0,0,56,0,0,56,56,0,0,56},
{0,1,4,4,5,4,4,0,0,4,4,4,0,0,0,0,1,0,0,1,0,1,0,0,52,0,0,48,52,4,4,52},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,22,0,0,16,18,6,2,22},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,54,0,0,48,50,6,2,54},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,14,0,0,8,10,6,2,14},
{0,1,2,2,3,2,2,0,0,6,6,6,4,0,4,0,1,4,4,5,4,1,4,0,62,16,16,56,58,22,18,62},
{0,1,2,2,3,2,2,0,0,34,34,2,32,32,32,0,1,0,0,1,0,1,32,0,58,0,0,56,26,2,2,58},
{0,1,66,66,67,66,66,0,0,98,98,66,32,32,32,0,1,0,0,1,0,1,32,0,122,0,0,56,90,66,66,122},
{0,129,66,66,195,66,66,0,0,70,70,70,4,0,4,0,129,4,4,133,4,129,4,0,118,0,0,48,114,70,66,118},
{0,129,66,66,195,66,66,0,0,78,78,78,12,0,12,0,129,12,12,141,12,129,12,0,78,0,0,0,66,78,66,78},
{0,129,66,66,195,66,66,0,16,126,126,126,60,16,60,16,145,44,44,189,44,145,44,16,110,0,0,0,66,11
0,66,110},
{0,129,66,66,195,66,66,0,8,126,126,126,60,8,60,8,137,52,52,189,52,137,52,8,118,0,0,0,66,118,66
,118},
{0,129,66,66,195,66,66,0,32,126,126,126,60,32,60,32,161,28,28,189,28,161,28,32,94,0,0,0,66,94,
66,94},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,129,66,66,195,66,66,0,0,126,126,126,60,0,60,0,129,60,60,189,60,129,60,0,126,0,0,0,66,126,66
,126},
{0,1,66,66,67,66,66,0,4,126,126,126,60,4,60,4,5,56,56,61,56,5,56,4,122,0,0,0,66,122,66,122},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,1,66,66,67,66,66,0,0,126,126,126,60,0,60,0,1,60,60,61,60,1,60,0,126,0,0,0,66,126,66,126},
{0,129,60,60,189,60,60,0,0,60,60,60,0,0,0,0,129,0,0,129,0,129,0,0,60,0,0,0,60,60,60,60},
{0,129,0,0,129,0,0,0,2,2,2,2,2,2,2,2,129,0,0,129,0,129,0,0,0,0,0,0,0,0,0,0},
{44,9,8,12,45,40,8,12,14,14,14,14,14,14,14,14,33,32,32,33,0,33,32,32,80,0,0,80,80,0,0,80},
{44,9,8,12,45,40,8,12,14,14,14,14,14,14,14,14,33,32,32,33,0,33,32,32,16,0,0,16,16,0,0,16},
{44,9,8,12,45,40,8,12,14,78,78,78,14,78,78,78,97,96,96,97,64,97,96,96,16,0,0,16,16,0,0,16},
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,16,0,0,16,16,0,0,16},
{36,129,0,4,165,32,0,4,6,70,70,70,6,70,70,70,225,96,96,225,64,225,96,96,0,0,0,0,0,0,0,0},
{22,196,4,6,214,20,4,6,7,39,39,39,7,39,39,39,240,48,48,240,32,240,48,48,8,0,0,8,8,0,0,8},
{6,196,4,6,198,4,4,6,7,39,39,39,7,39,39,39,224,32,32,224,32,224,32,32,8,0,0,8,8,0,0,8},
{6,132,4,6,134,4,4,6,7,39,39,39,7,39,39,39,160,32,32,160,32,160,32,32,0,0,0,0,0,0,0,0},
{44,137,8,12,173,40,8,12,14,14,14,14,14,14,14,14,161,32,32,161,0,161,32,32,80,0,0,80,80,0,0,80
},
{44,137,8,12,173,40,8,12,14,14,14,14,14,14,14,14,161,32,32,161,0,161,32,32,80,0,0,80,80,0,0,80
},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,64,0,0,64,64,0,0,64},
{0,129,62,62,191,62,62,0,0,62,62,62,0,0,0,0,129,0,0,129,0,129,0,0,62,0,0,0,62,62,62,62},
{0,129,34,34,163,34,34,0,0,62,62,62,28,0,28,0,129,28,28,157,28,129,28,0,62,0,0,0,34,62,34,62},
{0,129,2,2,131,2,2,0,16,62,62,62,60,16,60,16,145,44,44,189,44,145,44,16,46,0,0,0,2,46,2,46},
{0,129,0,0,129,0,0,0,32,62,62,62,62,32,62,32,161,30,30,191,30,161,30,32,30,0,0,0,0,30,0,30},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,62,62,62,62,0,62,0,129,62,62,191,62,129,62,0,62,0,0,0,0,62,0,62},
{0,129,0,0,129,0,0,0,0,46,46,46,46,0,46,0,129,46,46,175,46,129,46,0,46,0,0,0,0,46,0,46},
{0,129,0,0,129,0,0,0,0,2,2,2,2,0,2,0,129,2,2,131,2,129,2,0,18,0,0,16,16,2,0,18},
{0,129,0,0,129,0,0,0,0,2,2,2,2,0,2,0,129,2,2,131,2,129,2,0,62,0,0,60,60,2,0,62},
{0,129,0,0,129,0,0,0,0,2,2,2,2,0,2,0,129,2,2,131,2,129,2,0,62,0,0,60,60,2,0,62},
{0,129,0,0,129,0,0,0,0,2,2,2,2,0,2,0,129,2,2,131,2,129,2,0,42,0,0,40,40,2,0,42},
{0,129,0,0,129,0,0,0,0,6,6,6,6,0,6,0,129,6,6,135,6,129,6,0,14,0,0,8,8,6,0,14},
{0,129,0,0,129,0,0,0,4,6,6,6,6,4,6,4,133,2,2,135,2,133,2,4,26,0,0,24,24,2,0,26},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,30,0,0,24,26,6,2,30},
{0,129,2,2,131,2,2,0,0,6,6,6,4,0,4,0,129,4,4,133,4,129,4,0,30,0,0,24,26,6,2,30},
{0,129,6,6,135,6,6,0,0,6,6,6,0,0,0,0,129,0,0,129,0,129,0,0,78,0,0,72,78,6,6,78},
{0,129,0,0,129,0,0,0,0,0,0,0,0,0,0,0,129,0,0,129,0,129,0,0,24,0,0,24,24,0,0,24},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,8,0,0,8,8,0,0,8},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,44,0,0,44,44,0,0,44},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,60,0,0,60,60,0,0,60},
{0,195,0,0,195,0,0,0,0,0,0,0,0,0,0,0,195,0,0,195,0,195,0,0,16,0,0,16,16,0,0,16},
{0,239,0,0,239,0,0,0,0,0,0,0,0,0,0,0,239,0,0,239,0,239,0,0,0,0,0,0,0,0,0,0},
{0,255,0,0,255,0,0,0,0,0,0,0,0,0,0,0,255,0,0,255,0,255,0,0,0,0,0,0,0,0,0,0}
};

const uint8_t rsgcLogo[96][32] PROGMEM = {
```

```
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,127,127,127,255,255,127,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,127,127,255,255,255,255,127},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,127,119},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,127,255,255,255,127,247,119,119},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,0,0,0,0,0,0,0,16,255,255,127,127,127,127,111,127},
{0,0,0,0,0,0,0,0,0,0,0,0,0,16,16,0,0,0,0,0,0,0,16,0,255,255,127,127,207,255,111,111},
{0,0,0,0,0,40,40,40,0,0,0,0,16,0,16,16,0,0,0,0,0,40,56,56,255,255,127,255,111,71,23,83},
{0,0,0,0,64,0,0,0,56,0,0,0,40,0,16,170,0,0,0,0,120,184,2,82,199,135,7,63,21,239,239,127},
{0,134,0,0,2,0,130,132,0,134,0,2,4,150,20,130,0,0,0,0,0,0,16,0,121,253,123,251,105,57,239,107}
,
{0,6,34,32,34,22,32,54,0,6,34,34,4,32,4,100,0,88,88,0,0,16,80,0,129,167,217,167,219,253,111,17
5},
{0,130,130,18,18,52,164,130,0,130,146,2,134,18,218,26,0,104,96,8,104,96,64,40,5,131,125,199,18
9,179,135,73},
{0,0,40,0,32,72,120,224,0,0,40,0,40,96,40,32,0,16,0,16,16,0,144,144,199,7,255,231,175,215,245,
163},
{0,0,16,16,0,0,48,48,0,0,16,16,16,0,24,0,0,40,32,8,32,32,0,40,199,199,111,119,31,87,199,23},
{0,0,0,0,32,0,56,40,0,0,0,0,32,32,0,36,0,24,24,0,24,24,20,16,199,199,127,119,111,79,123,251},
{0,0,0,0,0,0,20,20,0,0,0,0,0,0,16,8,0,0,0,0,0,0,8,4,255,127,255,127,103,251,239,195},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,127,255,127,127,255,251,119},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,207,207,207,207,79,207,79,71},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,127,127,255,127,127,255,127},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,207,207,207,207,207,207,207,207},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,239,239,239,239,239,239,239,239},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,223,223,223,223,223,223,223,223},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,207,207,205,205,207,207,205,205},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,223,223,223,223,223,223,223,223},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,207,207,207,207,207,207,207,207},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,247,247,247,247,247,247,247,247},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,247,247,247,247,247,247,247,247},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,199,199,199,199,199,199,199,199},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,247,247,255,255,247,247},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,215,215,215,215,215,215,215,215},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,247,247,215,215,247,247,215,215},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,199,199,199,199,199,199,199,199},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,247,247,255,255,247,247},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,215,215,215,215,215,215,215,215},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,247,247,247,247,247,247,247,247},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,231,231,231,231,231,231,231,231},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,207,207,143,143,207,207,143,143},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,247,247,247,247,247,247,247,247},
{0,0,128,128,128,128,4,0,0,0,128,128,128,128,132,0,0,0,0,0,0,0,0,71,199,71,71,71,199,195,199
},
{0,4,4,4,0,0,4,4,0,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,251,255,251,255,255,255,63,251},
{0,0,0,0,4,0,4,4,0,0,0,0,0,0,4,0,0,0,0,0,0,0,0,0,255,255,251,255,243,235,239,55},
{0,0,0,0,0,0,16,24,12,0,0,0,0,0,0,0,4,0,24,24,16,8,8,0,32,231,231,219,227,223,219,223,63},
{0,0,4,0,4,4,0,0,0,0,4,4,0,24,60,20,0,48,40,32,24,0,48,32,195,195,239,211,231,247,147,83},
{0,48,0,56,44,0,4,36,0,48,0,56,60,60,56,20,0,0,0,0,0,0,0,0,199,243,215,227,203,249,47,43},
{0,8,0,0,136,8,6,70,0,8,0,0,8,20,178,244,0,32,48,16,0,16,16,16,199,79,163,211,43,251,209,229},
{0,2,0,0,4,4,0,4,0,2,0,4,0,0,4,63,0,56,56,0,64,64,120,65,65,135,187,1,249,253,185,130},
{0,4,0,0,0,6,132,4,0,4,0,0,4,4,4,0,0,0,0,0,0,0,2,0,251,127,123,191,63,125,187,255},
{0,0,0,0,0,0,0,134,134,56,0,0,0,56,0,134,190,0,0,0,0,0,56,56,0,0,199,199,135,191,133,123,251,191},
{0,0,0,4,36,28,24,60,0,0,4,0,0,0,0,0,0,0,0,0,0,0,32,24,24,56,187,251,191,187,223,167,231,199},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,0,4,4,0,0,191,255,187,223,187,255,255,239},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,191,187,187,191,191,55},
{0,4,0,4,0,0,0,0,0,4,0,4,0,4,0,4,0,0,0,0,0,0,0,0,0,0,251,255,255,255,123,127,123,63},
{0,0,0,128,0,128,128,0,0,0,0,128,128,0,128,128,0,0,0,0,0,0,0,0,255,127,255,127,127,255,251,191
},
{0,0,0,0,0,128,0,128,0,0,0,0,0,128,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,127,127,127},
```

```
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,127,255,255,255,127,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,127,127,191,127,191},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,127,127,255,255,63,255},
{0,0,0,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,255,255,255,127,127,255,59,127},
{0,0,0,0,4,0,0,4,0,0,0,0,0,4,4,0,0,0,0,0,0,0,0,0,0,255,251,251,251,251,255,251,63},
{0,0,0,0,0,0,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,255,255,191,255,255,191,179,243},
{0,0,0,0,0,0,0,0,0,8,0,0,8,8,8,0,0,0,0,0,0,0,0,0,0,183,243,255,251,247,251,163,255},
{0,0,0,0,0,16,16,48,0,8,0,8,8,32,12,36,0,0,0,0,8,16,20,20,247,215,223,215,159,175,227,163},
{0,0,0,0,0,0,0,0,28,32,40,32,56,44,40,28,0,0,0,12,24,52,32,0,195,195,195,247,231,179,159,252},
{0,0,0,0,0,0,1,0,0,8,0,0,9,9,8,2,0,0,0,0,0,0,0,0,246,247,191,190,182,254,247,116},
{0,1,3,9,0,9,3,1,0,1,3,1,19,5,18,53,0,60,60,0,8,8,56,0,192,193,254,195,245,105,178,223},
{0,16,16,0,0,2,48,51,0,16,16,16,0,22,8,24,0,44,44,0,36,32,12,9,195,209,253,207,225,243,236,110
},
{0,0,0,0,0,0,0,0,0,0,0,0,20,24,0,0,44,28,48,44,32,28,32,195,195,175,151,219,223,235,251},
{0,16,0,16,16,48,0,48,0,16,16,0,16,8,8,0,0,12,8,4,8,0,32,8,227,211,255,195,219,255,251,143},
{0,0,0,0,0,0,24,12,0,0,0,0,0,0,0,0,24,28,0,8,12,20,4,227,227,255,231,255,251,143,223},
{0,0,0,0,0,4,0,0,0,0,0,0,0,4,0,24,0,0,0,0,0,0,8,24,255,255,255,255,247,171,183,191},
{0,0,4,0,0,4,0,0,0,0,4,0,0,4,4,4,0,0,0,0,0,0,0,0,251,255,251,251,255,251,187,187},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,0,0,0,0,0,4,4,199,199,199,199,67,135,3,195},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,175,111,47,239,239,239,239,47},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,215,215,151,151,215,215,151,151},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,199,199,199,199,199,199,199,199},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,207,207,207,207,207,207,207,207},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,207,207,207,207,207,207,207,207},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,199,199,199,199,199,199,199,199},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,223,223,223,223,223,223,223,223},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,223,223,223,223,223,223,223,223},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,199,199,199,199,199,199,199,199},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,223,223,223,223,223,223,223,223},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,223,223,223,223,223,223,223,223},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255}
};


const uint8_t acesLogo[96][32] PROGMEM = {
{48,0,32,0,48,32,16,48,48,32,16,0,48,48,0,32,0,0,0,0,0,0,0,0,0,0,16,0,0,48,0},
{112,0,48,64,112,112,64,48,112,112,0,0,48,112,64,112,0,0,0,0,0,0,0,0,0,0,0,0,64,0,48,0},
{112,8,104,24,120,120,0,112,120,96,16,24,112,112,8,120,0,0,0,0,0,0,0,0,0,0,0,8,24,8,8,120,0},
{96,24,112,24,120,120,16,96,112,104,8,16,104,120,16,112,0,0,0,0,0,0,0,0,0,24,8,8,0,0,104,8},
{96,16,120,8,104,96,16,120,96,120,16,16,120,112,8,96,0,0,0,0,0,0,0,0,8,16,24,0,0,8,120,16},
{96,20,96,24,104,96,16,108,96,116,24,0,100,100,20,96,0,0,0,0,0,0,0,0,24,12,4,8,0,28,108,24},
{96,20,96,0,96,96,20,96,96,116,0,20,96,96,20,116,0,0,0,0,0,0,0,0,28,8,8,28,28,28,104,28},
{98,0,116,22,118,98,2,118,98,116,2,2,118,98,22,98,0,0,0,0,0,0,0,0,28,8,28,8,8,30,126,10},
{226,20,162,64,226,226,84,226,226,182,64,84,226,226,84,246,0,0,0,0,0,0,0,0,28,8,8,28,28,92,234
,92},
{227,20,226,29,235,227,21,235,227,246,29,1,227,227,21,227,0,0,0,0,0,0,0,0,28,8,0,8,0,29,235,29
},
{227,20,254,9,235,227,21,255,227,254,21,21,255,247,9,227,0,0,0,0,0,0,0,0,8,20,28,0,0,9,255,21}
,
{226,28,182,92,254,254,84,226,246,170,72,84,234,254,84,246,0,0,0,0,0,0,0,0,0,28,8,8,0,64,234,7
2},
{118,8,104,30,126,126,10,118,126,96,22,30,118,126,10,126,0,0,0,0,0,0,0,0,0,8,0,20,0,2,118,2},
{124,0,124,0,124,124,0,124,124,124,0,0,124,124,0,124,0,0,0,0,0,0,0,0,0,0,0,0,0,0,124,0},
{124,0,124,0,124,124,0,124,124,124,0,0,124,124,0,124,0,0,0,0,0,0,0,0,0,0,0,0,0,0,124,0},
{120,0,120,0,120,120,0,120,120,120,0,0,120,120,0,124,0,0,0,0,0,0,4,4,0,0,0,0,0,0,0,120,0},
{120,0,120,0,120,120,0,120,120,120,0,0,120,120,0,120,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,122,2},
{120,0,120,0,120,120,0,120,120,120,0,0,120,120,0,120,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,122,2},
{112,0,48,64,112,112,64,48,112,112,0,0,48,112,64,112,0,0,0,0,0,0,0,0,0,2,2,2,2,66,2,50,2},
{48,0,32,0,48,32,16,48,48,32,16,0,48,48,0,32,0,0,0,0,0,0,0,0,0,0,0,0,0,16,0,0,48,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,24,24,24,24,24,24,24},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,8,8,8,8,8,8,8},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,24,8,8,24,24,8,8},
```

```
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,8,8,8,8,8,8,8},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,8,8,8,8,8,8,8},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,24,16,16,24,24,16,16},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,16,16,16,16,16,16,16},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,17,17,17,17,17,17,17,17,17},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,27,27,27,27,27,27,27,27},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,16,16,16,16,16,16,16},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,17,17,17,17,17,17,17,17},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3},
{48,0,48,0,0,48,48,48,48,48,0,0,0,48,48,48,0,0,0,0,0,0,0,0,0,0,0,0,48,0,0,0},
{56,0,48,8,56,56,8,56,56,48,8,8,56,56,8,56,0,0,0,0,0,0,0,0,2,2,2,2,2,10,58,10},
{56,0,120,64,56,120,0,120,56,120,0,0,120,56,0,120,0,0,0,0,0,0,0,0,1,1,65,1,1,1,57,1},
{120,0,56,64,120,120,64,120,120,56,64,64,120,120,68,124,0,0,0,0,4,0,4,4,2,2,2,2,6,66,122,70},
{124,0,60,64,124,124,64,124,124,60,64,64,124,124,64,124,0,0,0,0,0,0,0,0,0,0,0,0,0,64,124,64},
{124,0,60,0,60,60,64,124,124,60,64,0,60,124,0,124,0,0,0,0,0,0,0,0,0,0,0,0,64,0,0,124,0},
{60,0,62,66,60,126,0,62,60,62,0,64,126,124,64,62,0,0,0,0,0,0,0,0,0,0,0,66,0,0,0,124,64},
{62,128,178,4,178,58,12,58,62,178,140,132,178,178,132,178,0,0,0,0,0,0,0,0,0,0,0,12,136,128,12,58
,12},
{178,12,175,17,166,239,16,246,182,171,24,24,242,238,84,242,0,0,0,0,0,0,0,0,0,0,0,12,8,85,93,81,235
,76},
{227,28,241,22,243,243,28,227,243,237,6,26,227,243,28,251,0,0,0,0,0,0,0,0,0,0,0,12,16,0,10,8,12,227,
12},
{227,16,253,6,241,241,2,249,243,229,10,2,233,241,18,229,0,0,0,0,0,0,0,0,0,0,0,12,24,6,16,28,14,237,3
0},
{162,16,180,1,169,179,2,167,162,176,23,80,173,177,1,224,0,0,0,0,0,0,0,0,0,0,0,12,92,27,10,84,14,190,
31},
{32,146,162,22,54,50,2,36,34,176,148,130,176,178,130,162,0,0,0,0,0,0,0,0,0,0,0,12,30,24,156,136,12,6
0,28},
{32,16,116,64,42,112,2,100,32,112,20,18,110,48,0,98,0,0,0,0,0,0,0,0,0,0,0,12,28,90,8,22,14,62,30},
{96,16,60,68,48,112,64,56,112,36,72,64,40,112,80,100,0,0,0,0,0,0,0,0,0,0,0,12,24,4,16,28,76,108,92},
{96,28,48,80,112,112,88,100,112,44,64,88,100,116,92,120,0,0,0,0,0,0,0,0,0,0,0,8,20,4,8,8,76,100,72},
{112,8,40,80,96,104,80,112,112,40,88,88,112,104,84,112,0,0,0,0,0,4,4,0,0,0,0,8,8,16,24,80,108,76},
{56,0,48,64,112,120,8,120,56,48,72,0,112,112,64,48,0,0,0,0,0,0,0,0,0,0,0,0,0,72,8,0,8,56,72},
{56,0,48,8,48,56,8,48,56,48,8,8,56,56,8,56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,56,8},
{48,0,48,0,0,48,32,32,48,48,0,0,16,48,32,48,0,0,0,0,0,0,0,0,0,0,0,0,0,32,0,16,0},
{16,32,48,0,16,16,0,16,16,48,32,32,48,48,32,56,0,0,0,0,0,0,8,8,0,0,0,0,0,32,32,0,16,0},
{56,0,56,0,56,56,0,56,56,56,0,0,56,56,0,56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,56,0},
{56,0,48,8,48,48,8,48,56,48,8,8,48,48,12,48,0,0,0,0,4,4,0,4,0,0,8,0,4,8,48,12},
{48,12,60,0,56,60,0,56,56,52,4,4,48,60,8,48,0,0,0,0,0,0,0,0,0,0,12,4,8,12,0,52,12},
{48,12,32,24,48,48,12,48,48,44,24,20,48,48,12,52,0,0,0,0,0,0,0,0,0,0,12,0,0,20,4,12,48,12},
{48,0,44,24,32,32,16,36,48,40,20,16,36,34,16,40,0,0,0,0,0,2,2,0,2,12,4,24,0,14,28,46,30},
{48,2,42,0,36,50,16,56,48,34,26,2,44,34,0,32,0,0,0,0,0,0,0,0,0,12,14,22,20,10,28,62,30},
{160,18,176,26,186,178,16,168,176,162,10,16,162,178,16,176,0,0,0,0,0,0,0,0,0,12,30,6,14,4,12,174
,14},
{241,2,171,64,229,243,80,249,241,163,90,66,237,227,64,225,0,0,0,0,0,0,0,0,0,12,14,22,20,10,92,25
5,94},
{241,2,239,24,227,227,16,231,243,233,20,16,229,227,18,233,0,0,0,0,0,0,0,0,0,12,6,24,2,14,28,237,
30},
{241,14,227,26,243,243,14,241,243,237,24,22,241,243,14,247,0,0,0,0,0,0,0,0,0,12,2,0,20,4,12,241,
12},
{243,12,189,66,249,253,66,251,251,181,70,70,243,253,74,243,0,0,0,0,0,0,0,0,0,12,4,10,14,66,245
,76},
{190,0,178,8,178,182,12,182,190,178,12,8,178,178,8,178,0,0,0,0,0,0,0,0,1,0,0,12,4,0,12,182,12},
{62,0,62,0,62,62,0,62,62,62,0,0,62,62,0,62,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,62,0},
{60,0,60,0,60,60,0,60,60,60,0,0,60,62,0,60,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,60,0},
{60,0,60,0,60,60,0,60,60,60,0,0,0,60,60,0,60,0,0,0,0,0,0,0,0,0,0,0,0,0,0,60,0},
{60,0,60,0,60,60,0,60,60,60,0,0,0,60,60,0,60,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,63,3},
{56,0,56,0,56,56,0,56,56,56,0,0,60,56,0,56,0,0,0,0,0,4,0,4,0,0,0,4,4,0,56,4},
{56,0,56,0,56,56,0,56,56,56,0,0,56,56,0,56,0,0,0,0,0,0,0,0,3,3,3,3,3,3,59,3},
{16,32,48,0,48,48,0,16,48,16,0,0,48,16,32,56,0,0,0,0,0,8,8,0,0,0,0,32,32,0,16,32},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3},
```

```
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,8,8,8,8,8,8,8},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,8,0,0,8,8,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,3,3,3,3,3,3},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4,4,4},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
};

const uint8_t darcy[96][32] PROGMEM = {
{0,0,0,0,0,127,255,0,0,0,0,0,127,0,127,255,0,0,0,0,0,0,0,127,255,255,127,128,128,0,127},
{0,0,0,0,0,63,127,0,0,0,0,0,63,0,63,63,0,0,0,0,0,0,0,64,63,63,63,63,0,0,64,63},
{0,0,0,0,0,63,31,32,0,0,0,0,31,0,31,63,0,0,0,0,0,0,0,31,63,63,31,32,32,0,95},
{0,0,0,0,0,31,31,32,0,0,0,0,31,0,31,31,0,0,0,0,0,0,0,64,31,31,31,31,0,0,0,127},
{0,0,0,0,16,15,31,0,0,0,0,0,31,0,31,31,0,0,0,0,0,0,0,31,31,31,15,16,16,48,79},
{0,0,0,0,0,31,31,16,0,0,0,0,15,0,15,11,0,0,0,0,0,0,0,16,15,15,15,15,0,16,20,75},
{0,0,0,0,0,31,11,4,0,0,0,0,11,4,15,11,0,0,0,0,0,0,16,16,15,15,15,15,0,20,4,91},
{0,0,0,0,4,31,25,4,0,0,4,0,9,4,11,11,0,0,0,0,0,16,16,0,9,13,9,13,6,18,2,15},
{0,0,0,12,0,16,48,13,0,12,0,0,44,8,60,20,0,0,0,0,0,0,0,0,12,0,4,8,36,52,49,24},
{0,0,0,0,12,20,17,21,0,5,24,56,40,49,13,32,0,0,0,0,0,0,0,12,0,44,20,56,9,9,63},
{0,0,0,0,68,64,10,18,0,88,4,70,70,91,44,49,0,0,0,0,0,1,0,32,12,0,20,92,80,80,103,15},
{0,0,0,0,19,78,37,67,0,64,87,30,109,18,91,10,0,0,0,0,0,0,0,0,4,0,76,64,20,96,62,153},
{0,0,0,67,13,19,7,65,0,75,68,66,5,116,102,132,0,0,0,0,0,0,0,0,4,72,72,74,25,125,69,188},
{0,0,0,64,198,111,200,163,64,0,134,168,33,42,45,137,0,0,0,0,0,0,0,0,15,65,239,11,8,72,184,253}
,
{0,0,64,0,8,23,55,32,64,0,0,32,15,0,255,7,0,0,0,0,0,0,0,0,15,79,103,79,32,48,8,39},
{0,0,0,0,0,15,95,16,0,0,0,128,111,128,175,79,0,0,0,0,128,0,144,144,79,79,111,47,64,160,64,239}
,
{0,0,0,32,0,111,127,16,0,32,96,32,207,208,79,159,0,0,0,0,0,0,128,0,79,79,47,111,144,48,0,159},
{0,0,0,0,32,29,237,0,0,32,64,0,221,48,141,13,0,0,0,0,0,0,0,0,45,141,77,205,96,160,64,13},
{0,0,0,32,32,31,79,0,0,32,16,160,47,240,255,191,0,0,0,0,0,0,0,0,15,47,31,175,144,176,208,223},
{0,0,0,0,96,61,157,144,0,32,0,128,253,32,157,189,0,0,0,0,0,0,0,0,253,93,221,93,0,0,128,125},
{0,0,0,0,0,255,239,144,0,0,128,0,127,32,79,111,0,0,0,0,0,0,0,0,127,255,95,95,160,128,16,239},
{0,0,0,0,0,245,245,0,0,0,0,0,245,0,245,245,0,0,0,0,0,0,0,0,245,245,245,245,0,0,0,213},
{0,0,0,0,0,253,93,128,0,0,0,0,93,32,61,253,0,0,0,0,0,0,0,0,253,221,125,29,224,64,0,93},
{0,0,0,0,0,253,253,0,0,0,128,128,125,128,253,253,0,0,0,0,0,0,0,0,253,125,125,253,0,0,0,125},
{0,0,0,0,0,255,127,128,0,0,64,64,63,64,127,255,0,0,0,0,0,0,0,0,255,191,63,127,128,0,0,63},
{0,0,0,0,0,253,253,0,0,0,0,0,253,0,253,253,0,0,0,0,0,0,0,0,253,253,253,253,0,0,0,253},
{0,0,0,0,0,245,245,0,0,0,0,0,245,0,245,245,0,0,0,0,0,0,0,0,245,245,245,245,0,0,0,245},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,249,249,0,0,0,0,0,249,0,249,249,0,0,0,0,0,0,0,0,249,249,249,249,0,0,0,249},
{0,0,0,0,0,246,246,0,0,0,0,0,246,0,246,246,0,0,0,0,0,0,0,0,246,246,246,246,0,0,0,246},
{0,0,0,0,0,248,248,0,0,0,0,0,248,0,248,248,0,0,0,0,0,0,0,0,248,248,248,248,0,0,0,248},
{0,0,0,64,0,191,223,0,0,0,96,64,159,96,159,255,0,0,0,0,0,0,0,0,159,223,159,255,32,32,64,255},
{0,0,0,0,0,248,120,0,0,0,160,32,88,160,120,248,0,0,0,0,0,0,0,0,120,88,88,248,128,128,0,216},
{0,0,0,0,64,58,186,64,0,0,32,160,218,160,58,250,0,0,0,0,0,0,0,0,122,90,218,122,0,128,128,90},
{0,0,0,128,0,96,224,0,0,0,160,160,80,160,64,208,0,0,0,0,0,0,16,16,80,240,80,208,32,48,176,192}
,
{0,0,0,0,192,47,175,16,0,0,144,16,239,16,15,239,0,0,0,0,0,0,0,0,127,111,255,255,128,16,176,47}
,
{0,0,16,0,32,109,157,16,0,16,144,32,253,32,141,61,0,0,0,0,0,0,0,0,77,93,205,221,176,160,240,29
},
{0,0,0,128,32,30,30,144,0,16,224,176,206,144,62,14,0,0,0,0,64,0,64,0,14,254,30,206,112,208,112
,206},
{0,0,0,0,0,12,140,128,0,0,64,128,236,0,28,188,0,0,0,0,0,0,0,0,44,108,44,204,48,32,112,44},
{0,0,0,32,64,111,111,96,0,96,64,192,15,32,255,207,0,0,0,0,0,0,0,128,111,15,47,47,32,192,0,31},
{0,0,0,0,8,69,77,200,0,64,32,40,5,64,125,125,0,0,0,0,0,32,2,16,45,45,7,77,104,74,80,173},
```

```
{0,0,0,104,64,69,93,32,64,32,136,8,109,192,71,229,0,0,0,0,0,2,0,18,165,5,237,109,128,0,8,229},
{0,0,0,98,163,72,29,70,0,98,229,8,18,15,232,142,0,0,0,0,0,0,0,0,37,0,73,178,17,175,137,98},
{0,0,34,8,4,38,4,19,32,2,15,13,27,61,46,45,0,0,0,0,0,0,128,128,4,40,12,34,48,41,6,2},
{0,0,0,42,34,14,34,75,32,10,2,5,47,98,30,109,0,0,0,0,0,16,0,0,8,0,36,42,32,34,111,76},
{0,0,0,2,3,76,2,0,0,2,3,77,13,97,79,108,0,0,0,0,0,0,32,32,0,0,76,34,96,33,92,110},
{0,0,1,4,1,31,0,4,0,7,1,25,13,49,11,29,0,0,0,0,0,0,0,0,4,0,8,2,53,4,24,46},
{0,0,0,0,0,12,3,0,0,0,0,12,44,32,44,44,0,0,0,0,0,32,2,2,0,0,12,0,32,0,12,28},
{0,0,0,0,4,9,9,0,0,0,0,0,13,4,15,15,0,0,0,0,0,0,0,0,13,13,9,9,2,6,6,25},
{0,0,0,0,0,15,15,0,0,0,0,0,15,0,15,15,0,0,0,0,0,0,0,0,15,15,15,15,0,0,0,31},
{0,0,0,0,0,15,15,0,0,0,0,0,15,0,15,15,0,0,0,0,0,0,0,0,15,15,15,15,0,0,0,47},
{0,0,0,0,0,31,15,16,0,0,0,0,15,0,15,31,0,0,0,0,0,0,0,0,15,15,15,15,0,16,16,127},
{0,0,0,0,16,31,31,16,0,0,0,0,15,0,15,15,0,0,0,0,0,0,16,0,16,15,15,15,15,16,16,16,31},
{0,0,0,0,0,31,31,0,0,0,0,0,31,16,15,31,0,0,0,0,0,0,0,0,32,31,31,31,31,0,0,0,47},
{0,0,0,0,0,63,191,128,0,0,0,0,63,0,63,63,0,0,0,0,0,0,0,0,63,63,63,63,0,0,128,63},
{0,0,0,0,0,255,191,64,0,0,0,0,191,0,191,63,0,0,0,0,0,64,0,0,191,191,191,191,64,0,0,255},
{0,0,0,0,0,255,191,0,0,0,0,0,191,0,191,255,0,0,0,0,0,0,0,0,191,191,191,191,0,64,64,191},
{0,0,0,0,0,191,191,32,0,0,0,0,159,0,159,159,0,0,0,0,0,0,0,0,32,159,191,191,159,0,32,32,191},
{0,0,0,0,0,31,159,128,0,0,0,0,15,144,15,31,0,0,0,0,0,0,0,0,32,15,159,159,15,144,16,32,175},
{0,0,0,0,0,15,31,0,0,0,0,0,15,0,31,15,0,0,0,0,0,0,0,0,15,15,15,15,0,0,0,95},
{0,0,0,0,0,15,13,0,0,0,0,0,15,32,13,15,0,0,0,0,0,0,0,0,13,13,15,15,2,32,34,95},
{0,0,0,0,0,13,45,32,0,0,0,0,45,34,13,45,0,0,0,0,0,0,0,0,2,13,13,13,13,0,32,0,47},
{0,0,0,0,0,13,13,0,0,0,0,0,13,0,13,13,0,0,0,0,0,0,0,0,13,13,13,13,0,0,0,29},
{0,0,0,0,5,8,8,4,0,0,0,0,13,4,9,12,0,0,0,0,0,0,0,0,13,13,13,9,4,4,4,9},
{0,0,0,6,2,12,10,6,0,6,0,0,14,10,14,8,0,0,0,0,0,0,0,1,0,4,0,8,14,12,2,8,26},
{0,0,0,9,1,29,7,0,0,9,5,4,31,9,3,19,0,0,0,0,0,0,0,0,12,4,0,8,25,9,20,55},
{0,0,1,0,44,4,4,42,0,45,1,0,40,81,112,5,0,0,0,0,0,0,0,64,18,12,0,44,32,40,53,88,61},
{0,0,0,33,44,52,32,0,32,13,0,33,18,22,43,123,0,0,0,0,0,0,0,0,12,32,4,12,48,15,1,120},
{0,0,0,33,37,48,20,138,32,1,14,29,20,30,25,25,0,0,0,0,0,0,0,0,4,40,40,22,53,9,18,167},
{0,0,0,97,166,34,72,80,0,96,163,118,82,27,9,46,0,0,0,0,0,0,0,0,37,6,81,128,227,24,212,37},
{0,0,0,32,8,37,45,168,0,32,136,32,165,10,71,45,0,0,0,0,0,2,2,64,37,133,37,13,8,136,160,221},
{0,0,0,72,64,79,103,88,0,64,72,40,103,40,15,47,0,0,0,0,0,0,0,0,39,103,39,15,0,72,0,15},
{0,0,0,0,0,15,15,16,0,0,0,64,71,0,207,143,0,0,0,0,0,64,0,192,71,71,7,79,8,8,192,143},
{0,0,0,0,32,108,12,96,0,16,96,48,44,48,12,76,0,0,0,0,0,0,0,0,16,12,28,28,76,96,80,112,60},
{0,0,0,0,16,45,29,48,0,0,112,16,109,64,77,45,0,0,0,0,0,0,0,0,64,13,77,13,61,64,96,16,61},
{0,0,0,0,64,174,174,176,0,0,160,128,206,176,46,62,0,0,0,0,0,0,0,0,110,206,206,78,160,128,16,46
},
{0,0,0,0,80,47,111,64,0,0,0,192,175,144,127,63,0,0,0,0,128,0,128,0,255,191,255,127,0,64,0,111}
,
{0,0,0,0,0,112,96,0,0,0,144,0,96,16,224,240,0,0,0,0,128,0,0,0,224,224,96,240,16,16,0,240},
{0,0,0,0,208,7,7,208,0,0,144,0,87,128,87,247,0,0,0,0,0,0,0,0,215,87,71,215,128,0,128,55},
{0,0,0,0,0,244,244,0,0,0,32,32,212,32,244,244,0,0,0,0,0,0,0,0,244,212,212,244,0,0,0,212},
{0,0,0,64,0,191,223,32,0,0,64,64,159,64,159,255,0,0,0,0,0,0,0,0,191,255,159,223,32,0,64,223},
{0,0,0,0,0,248,248,0,0,0,0,0,248,0,248,248,0,0,0,0,0,0,0,0,248,248,248,248,0,0,0,248},
{0,0,0,0,0,247,247,0,0,0,0,0,247,0,247,247,0,0,0,0,0,0,0,0,247,247,247,247,0,0,0,247},
{0,0,0,0,0,248,248,0,0,0,0,0,248,0,248,248,0,0,0,0,0,0,0,0,248,248,248,248,0,0,0,248},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255},
{0,0,0,0,0,255,255,0,0,0,0,0,255,0,255,255,0,0,0,0,0,0,0,0,255,255,255,255,0,0,0,255}
};


uint8_t minimum(uint8_t a, uint8_t b) {
  return a < b ? a : b;
}

void skLoadPixel(uint8_t row, uint8_t c, uint8_t r, uint8_t g, uint8_t b) {
  uint8_t w = minimum(r,minimum(g,b));
  skLoadPixel(row, c, r-w, g-w, b-w, w);
}

void skLoadPixel(uint8_t row, uint8_t c, uint8_t r, uint8_t g, uint8_t b, uint8_t w) {
  for(int8_t n = 7; n >= 0; n--) {
    writeBit(7-n, c, row, (g>>n) & 1);
    writeBit(15-n, c, row, (r>>n) & 1);
    writeBit(23-n, c, row, (b>>n) & 1);
    writeBit(31-n, c, row, (w>>n) & 1);
  }
}
```

```
void skLoadPixel(uint8_t row, uint8_t column, uint32_t hex) {
  skLoadPixel(row, column, hex>>24, hex>>16, hex>>8, hex);
}

void skSendPixels() {
  SREG &= ~(1<<7);    //interrupts off
  for(uint8_t column = 0; column < 60; column++) {
    for(uint8_t n = 0; n < 32; n++) {
      writePixelBit(pixelData[column][n]);
    }
  }
  SREG |= 1<<7;       //interrupts on
}

void writePixelBit(uint8_t data) {
  PIXEL_PORT = 255;   //controls whole port in 2 clock cycles
  //PIXEL_PORT |= LED_PINS; extra 2 clock cycles
  __builtin_avr_delay_cycles(T_BLOCK - 2);
  PIXEL_PORT = data;  //controls whole port in 2 clock cycles
  //PIXEL_PORT &= ~data; extra 2 clock cycles
  __builtin_avr_delay_cycles(T_BLOCK - 2);
  PIXEL_PORT = 0;     //controls whole port in 2 clock cycles
  //PIXEL_PORT &= ~LED_PINS; extra 2 clock cycles
  __builtin_avr_delay_cycles(T_BLOCK - 2);
}

void showLEDs() {
  PIXEL_PORT = 0;
  __builtin_avr_delay_cycles(LATCH);
}

void setup() {
  //Serial.begin(9600);
  PIXEL_DDR |= 255;

  for(uint8_t x = 0; x < 60; x++) {
    for(uint8_t y = 0; y < 32; y++) {
      pixelData[x][y] = 0;
    }
  }
  delay(5000);
}

void loop() {

  SREG &= ~(1<<7);
  for(uint8_t column = 0; column < 96; column++) {
    for(uint8_t n = 0; n < 32; n++) {
      writePixelBit(pgm_read_byte(&rsgcLogo[column][n]));  //goes to address of data in
PROGMEM and reads
    }
  }
  SREG |= 1<<7;
  showLEDs();
  delay(3000);

  SREG &= ~(1<<7);
  for(uint8_t column = 0; column < 96; column++) {
    for(uint8_t n = 0; n < 32; n++) {
      writePixelBit(pgm_read_byte(&acesLogo[column][n]));  //goes to address of data in
PROGMEM and reads
    }
  }
  SREG |= 1<<7;
  showLEDs();
  delay(3000);

  SREG &= ~(1<<7);
```

```
  for(uint8_t column = 0; column < 96; column++) {
    for(uint8_t n = 0; n < 32; n++) {
      writePixelBit(pgm_read_byte(&darcy[column][n]));  //goes to address of data in PROGMEM
and reads
    }
  }
  SREG |= 1<<7;
  showLEDs();
  delay(3000);


  for(uint8_t x = 0; x < 12; x++) {

    //---------1
    SREG &= ~(1<<7);
    for(uint8_t column = 0; column < 96; column++) {
      for(uint8_t n = 0; n < 32; n++) {
        writePixelBit(pgm_read_byte(&nyan1[column][n]));  //goes to address of data in PROGMEM
and reads
      }
    }
    SREG |= 1<<7;
    showLEDs();

    delay(50);

    //------------2
    SREG &= ~(1<<7);
    for(uint8_t column = 0; column < 96; column++) {
      for(uint8_t n = 0; n < 32; n++) {
        writePixelBit(pgm_read_byte(&nyan2[column][n]));
      }
    }
    SREG |= 1<<7;
    showLEDs();

    delay(50);

    //-----------3
    SREG &= ~(1<<7);
    for(uint8_t column = 0; column < 96; column++) {
      for(uint8_t n = 0; n < 32; n++) {
        writePixelBit(pgm_read_byte(&nyan3[column][n]));
      }
    }
    SREG |= 1<<7;
    showLEDs();

    delay(50);

    //---------------4
    SREG &= ~(1<<7);
    for(uint8_t column = 0; column < 96; column++) {
      for(uint8_t n = 0; n < 32; n++) {
        writePixelBit(pgm_read_byte(&nyan4[column][n]));
      }
    }
    SREG |= 1<<7;
    showLEDs();

    delay(50);

    //--------------5
    SREG &= ~(1<<7);
    for(uint8_t column = 0; column < 96; column++) {
      for(uint8_t n = 0; n < 32; n++) {
        writePixelBit(pgm_read_byte(&nyan5[column][n]));
      }
    }
```

```
    SREG |= 1<<7;
    showLEDs();

    delay(50);

    //----------------6
    SREG &= ~(1<<7);
    for(uint8_t column = 0; column < 96; column++) {
      for(uint8_t n = 0; n < 32; n++) {
        writePixelBit(pgm_read_byte(&nyan6[column][n]));
      }
    }
    SREG |= 1<<7;
    showLEDs();

    delay(50);
  }

  SREG &= ~(1<<7);
  for(uint8_t n = 0; n < 10; n++) {
    for(uint16_t x = 0; x < 384; x++) {
      writePixel(255, 100, 0);
      writePixel(0, 0, 0);
    }
    showLEDs();
    delay(30);
    for(uint16_t x = 0; x < 384; x++) {
      writePixel(0, 0, 0);
      writePixel(255, 100, 0);
    }
    showLEDs();
    delay(30);
  }
  //SREG |= 1<<7;

  skSendPixels();
  showLEDs();
  delay(100);

  for(uint16_t x = 0; x < 1000; x++) {
    skLoadPixel(random(0, 60), random(0, 32), random(0, 255), random(0, 255), random(0, 255));
    skSendPixels();
    showLEDs();
  }
}

void writePixel(uint8_t r, uint8_t g, uint8_t b) {
  uint8_t w = minimum(r,minimum(g,b));
  sendByte(g - w);
  sendByte(r - w);
  sendByte(b - w);
  sendByte(w);
}

void sendByte(uint8_t data) {
  for(int8_t x = 7; x >= 0; x--) {
    sendBit(0 - ((data>>x) & 1));
  }
}

void sendBit(uint8_t data) {
  PIXEL_PORT = 255;   //controls whole port in 2 clock cycles
  //PIXEL_PORT |= LED_PINS; extra 2 clock cycles
  __builtin_avr_delay_cycles(T_BLOCK - 2);
  PIXEL_PORT = data;  //controls whole port in 2 clock cycles
  //PIXEL_PORT &= ~data; extra 2 clock cycles
  __builtin_avr_delay_cycles(T_BLOCK - 2);
  PIXEL_PORT = 0;     //controls whole port in 2 clock cycles
  //PIXEL_PORT &= ~LED_PINS; extra 2 clock cycles
```

```
  __builtin_avr_delay_cycles(T_BLOCK - 2);
}

uint8_t writeBit(uint8_t n, uint8_t c, uint8_t r, uint8_t b) {

  //other methods:
  /*
  if(b) {
    pixelData[c][n] |= 1<<r;
  } else  {
    pixelData[c][n] &= ~(1<<r);
  }*/

  //pixelData[c][n] ^= (-b ^ pixelData[c][n]) & (1 << r);
  //*(*(pixelData+c)+n) ^= (-b ^ *(*(pixelData + c) + n)) & (1 << r);
  //pixelData[c][n] ^= (-b ^ *(*(pixelData + c) + n)) & (1 << r);
  //pixelData[c][n] = (pixelData[c][n] & ~(1 << r)) | (-b & (1 << r));

  pixelData[c][n] &= ~(1 << r);   //clear bit
  pixelData[c][n] |= b << r;      //write a 0 or 1, depends on b
}

void white() {
  for(uint8_t x = 0; x < 8; x++) {
    for(uint8_t y = 0; y < 32; y++) {
      skLoadPixel(x, y, 0, 0, 0, 255);
    }
  }
}

void black() {
  for(uint8_t x = 0; x < 8; x++) {
    for(uint8_t y = 0; y < 32; y++) {
      skLoadPixel(x, y, 0, 0, 0, 0);
    }
  }
}
```

ATtiny84

```
// PROJECT  : Control SK6812s with ATtiny84
// PURPOSE  : Medium ISP
// COURSE   : ICS4U
// AUTHOR   : Xander Chin
// DATE     : Feburary 20, 2022
// MCU      : ATtiny84
// STATUS   : Working
// REFERENCE:

const uint8_t attiny[96][32] PROGMEM = {
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
{0,0,0,0,195,195,0,195,195,0,195,195,195,195,195,0,0,0,0,0,0,0,0,0,0,195,195,0,195,195,0},
{0,0,0,0,183,183,0,167,167,0,167,167,167,167,167,0,0,0,0,16,0,0,16,0,0,0,183,167,0,183,183,0},
{0,0,0,0,175,175,0,175,175,0,175,175,175,175,175,0,0,0,0,0,0,0,0,0,0,175,175,0,175,175,0},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,183,183,0,183,183,0,183,183,183,183,183,0,0,0,0,0,0,0,0,0,0,8,8,191,191,8,191,191,8},
{0,0,0,0,183,183,0,183,183,0,183,183,183,183,183,0,0,0,0,0,0,0,0,0,0,8,8,191,191,8,191,191,8},
{0,0,0,0,183,183,0,183,183,0,183,183,183,183,183,0,0,0,0,0,0,0,0,0,0,8,8,191,191,8,191,191,8},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,183,183,0,183,183,0,183,183,183,183,183,0,0,0,0,0,0,0,0,0,0,8,8,191,191,8,191,191,8},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,143,143,0,143,143,0,143,143,143,143,143,0,0,0,0,0,0,0,0,0,0,48,48,191,191,48,191,191,48
},
{0,0,0,0,167,167,0,167,167,0,167,167,167,167,167,0,0,0,0,0,0,0,0,0,0,24,24,191,191,24,191,191,24
},
{0,0,0,0,143,143,0,143,143,0,143,143,143,143,143,0,0,0,0,0,0,0,0,0,0,48,48,191,191,48,191,191,48
},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,167,167,0,167,167,0,167,167,167,167,167,0,0,0,0,0,0,0,0,0,0,24,24,191,191,24,191,191,24
},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,135,135,0,135,135,0,135,135,135,135,135,0,0,0,0,0,0,0,0,0,0,56,56,191,191,56,191,191,56
},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,183,183,0,183,183,0,183,183,183,183,183,0,0,0,0,0,0,0,0,0,0,8,8,191,191,8,191,191,8},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,183,183,0,183,183,0,183,183,183,183,183,0,0,0,0,0,0,0,0,0,0,8,8,191,191,8,191,191,8},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,191,191,0,191,191,0,191,191,191,191,191,0,0,0,0,0,0,0,0,0,0,191,191,0,191,191,0},
{0,0,0,0,195,195,0,195,195,0,195,195,195,195,195,0,0,0,0,0,0,0,0,0,0,195,195,0,195,195,0},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
{0,0,0,0,195,195,0,195,195,0,195,195,195,195,195,0,0,0,0,0,0,0,0,0,0,195,195,0,195,195,0},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
{0,0,0,0,189,189,0,189,189,0,189,189,189,189,189,0,0,0,0,0,0,0,0,0,0,189,189,0,189,189,0},
{0,0,0,0,189,189,0,189,189,0,189,189,189,189,189,0,0,0,0,0,0,0,0,0,0,189,189,0,189,189,0},
{0,0,0,0,251,251,0,251,251,0,251,251,251,251,251,0,0,0,0,0,0,0,0,0,0,4,4,255,255,4,255,255,4},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
{0,0,0,0,177,177,0,177,177,0,177,177,177,177,177,0,0,0,0,0,0,0,0,0,0,12,12,189,189,12,189,189,12
},
{0,0,0,0,181,181,0,181,181,0,181,181,181,181,181,0,0,0,0,0,0,0,0,0,0,8,8,189,189,8,189,189,8},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
{0,0,0,0,247,247,0,247,247,0,247,247,247,247,247,0,0,0,0,0,0,0,0,0,0,8,8,255,255,8,255,255,8},
{0,0,0,0,189,189,0,189,189,0,189,189,189,189,189,0,0,0,0,0,0,0,0,0,0,189,189,0,189,189,0},
{0,0,0,0,133,133,0,133,133,0,133,133,133,133,133,0,0,0,0,0,0,0,0,0,0,56,56,189,189,56,189,189,56
},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
```

```
{0,0,0,0,227,227,0,227,227,0,227,227,227,227,227,0,0,0,0,0,0,0,0,28,28,255,255,28,255,255,28
},
{0,0,0,0,189,189,0,189,189,0,189,189,189,189,189,0,0,0,0,0,0,0,0,0,0,189,189,0,189,189,0},
{0,0,0,0,133,133,0,133,133,0,133,133,133,133,133,0,0,0,0,0,0,0,0,0,0,56,56,189,189,56,189,189,56
},
{0,0,0,0,219,219,0,219,219,0,219,219,219,219,219,0,0,0,0,0,0,0,0,0,0,36,36,255,255,36,255,255,36
},
{0,0,0,0,207,207,0,207,207,0,207,207,207,207,207,0,0,0,0,0,0,0,0,0,0,48,48,255,255,48,255,255,48
},
{0,0,0,0,189,189,0,189,189,0,189,189,189,189,189,0,0,0,0,0,0,0,0,0,0,0,0,189,189,0,189,189,0},
{0,0,0,0,185,185,0,185,185,0,185,185,185,185,185,0,0,0,0,0,0,0,0,0,0,4,4,189,189,4,189,189,4},
{0,0,0,0,243,243,0,243,243,0,243,243,243,243,243,0,0,0,0,0,0,0,0,0,0,12,12,255,255,12,255,255,12
},
{0,0,0,0,251,251,0,251,251,0,251,251,251,251,251,0,0,0,0,0,0,0,0,0,0,4,4,255,255,4,255,255,4},
{0,0,0,0,189,189,0,189,189,0,189,189,189,189,189,0,0,0,0,0,0,0,0,0,0,0,0,189,189,0,189,189,0},
{0,0,0,0,181,181,0,181,181,0,181,181,181,181,181,0,0,0,0,0,0,0,0,0,0,8,8,189,189,8,189,189,8},
{0,0,0,0,251,251,0,251,251,0,251,251,251,251,251,0,0,0,0,0,0,0,0,0,0,4,4,255,255,4,255,255,4},
{0,0,0,0,247,247,0,247,247,0,247,247,247,247,247,0,0,0,0,0,0,0,0,0,0,8,8,255,255,8,255,255,8},
{0,0,0,0,189,189,0,189,189,0,189,189,189,189,189,0,0,0,0,0,0,0,0,0,0,0,0,189,189,0,189,189,0},
{0,0,0,0,181,181,0,181,181,0,181,181,181,181,181,0,0,0,0,0,0,0,0,0,0,0,0,181,181,0,181,181,0},
{0,0,0,0,239,239,0,231,231,0,231,231,231,231,231,0,0,0,0,8,0,0,8,0,0,0,239,231,0,239,239,0},
{0,0,0,0,195,195,0,195,195,0,195,195,195,195,195,0,0,0,0,0,0,0,0,0,0,0,0,195,195,0,195,195,0},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0},
{0,0,0,0,195,195,0,195,195,0,195,195,195,195,195,0,0,0,0,0,0,0,0,0,0,0,0,195,195,0,195,195,0},
{0,0,0,0,253,253,0,245,245,0,245,245,245,245,245,0,0,0,0,0,8,0,0,8,0,0,0,253,245,0,253,253,0},
{0,0,0,0,245,245,0,245,245,0,245,245,245,245,245,0,0,0,0,0,0,0,0,0,0,0,0,245,245,0,245,245,0},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,249,249,0,249,249,0,249,249,249,249,249,0,0,0,0,0,0,0,0,0,0,4,4,253,253,4,253,253,4},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,249,249,0,249,249,0,249,249,249,249,249,0,0,0,0,0,0,0,0,0,0,4,4,253,253,4,253,253,4},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,249,249,0,249,249,0,249,249,249,249,249,0,0,0,0,0,0,0,0,0,0,4,4,253,253,4,253,253,4},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,233,233,0,233,233,0,233,233,233,233,233,0,0,0,0,0,0,0,0,0,0,20,20,253,253,20,253,253,20
},
{0,0,0,0,233,233,0,233,233,0,233,233,233,233,233,0,0,0,0,0,0,0,0,0,0,20,20,253,253,20,253,253,20
},
{0,0,0,0,233,233,0,233,233,0,233,233,233,233,233,0,0,0,0,0,0,0,0,0,0,20,20,253,253,20,253,253,20
},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,237,237,0,237,237,0,237,237,237,237,237,0,0,0,0,0,0,0,0,0,0,16,16,253,253,16,253,253,16
},
{0,0,0,0,237,237,0,237,237,0,237,237,237,237,237,0,0,0,0,0,0,0,0,0,0,16,16,253,253,16,253,253,16
},
{0,0,0,0,237,237,0,237,237,0,237,237,237,237,237,0,0,0,0,0,0,0,0,0,0,16,16,253,253,16,253,253,16
},
{0,0,0,0,249,249,0,249,249,0,249,249,249,249,249,0,0,0,0,0,0,0,0,0,0,4,4,253,253,4,253,253,4},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,249,249,0,249,249,0,249,249,249,249,249,0,0,0,0,0,0,0,0,0,0,4,4,253,253,4,253,253,4},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,253,253,0,253,253,0,253,253,253,253,253,0,0,0,0,0,0,0,0,0,0,0,0,253,253,0,253,253,0},
{0,0,0,0,195,195,0,195,195,0,195,195,195,195,195,0,0,0,0,0,0,0,0,0,0,0,0,195,195,0,195,195,0},
{0,0,0,0,255,255,0,255,255,0,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,255,255,0,255,255,0}
};

const uint8_t attinyfill[96][32] PROGMEM = {
{60,0,60,60,195,195,0,195,255,60,195,195,195,255,195,0,0,0,0,0,0,0,0,0,0,0,0,255,255,60,195,255,
60},
{64,0,64,64,131,131,0,131,195,64,131,131,131,195,131,0,0,0,0,0,0,0,0,0,0,0,0,195,195,64,131,195,
64},
{0,0,0,0,131,131,0,131,131,0,131,131,131,131,131,0,0,0,0,0,0,0,0,0,0,0,0,131,131,0,131,131,0},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
```

```
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{2,0,2,2,129,129,0,129,131,2,129,129,129,131,129,0,0,0,0,0,0,0,0,0,0,0,131,131,2,129,131,2},
{0,0,0,0,131,131,0,131,131,0,131,131,131,131,131,0,0,0,0,0,0,0,0,0,0,0,131,131,0,131,131,0},
{64,0,64,64,131,131,0,131,195,64,131,131,131,195,131,0,0,0,0,0,0,0,0,0,0,0,195,195,64,131,195,
64},
{60,0,60,60,195,195,0,195,255,60,195,195,195,255,195,0,0,0,0,0,0,0,0,0,0,0,255,255,60,195,255,
60},
{60,0,60,60,195,195,0,195,255,60,195,195,195,255,195,0,0,0,0,0,0,0,0,0,0,0,255,255,60,195,255,
60},
{0,0,0,0,195,195,0,195,195,0,195,195,195,195,195,0,0,0,0,0,0,0,0,0,0,0,195,195,0,195,195,0},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
```

```
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{0,0,0,0,129,129,0,129,129,0,129,129,129,129,129,0,0,0,0,0,0,0,0,0,0,0,129,129,0,129,129,0},
{66,0,66,66,129,129,0,129,195,66,129,129,129,195,129,0,0,0,0,0,0,0,0,0,0,0,195,195,66,129,195,
66},
{0,0,0,0,195,195,0,195,195,0,195,195,195,195,195,0,0,0,0,0,0,0,0,0,0,0,195,195,0,195,195,0},
{60,0,60,60,195,195,0,195,255,60,195,195,195,255,195,0,0,0,0,0,0,0,0,0,0,0,255,255,60,195,255,
60},
{60,0,60,60,195,195,0,195,255,60,195,195,195,255,195,0,0,0,0,0,0,0,0,0,0,0,255,255,60,195,255,
60},
{2,0,2,2,193,193,0,193,195,2,193,193,193,195,193,0,0,0,0,0,0,0,0,0,0,0,195,195,2,193,195,2},
{0,0,0,0,193,193,0,193,193,0,193,193,193,193,193,0,0,0,0,0,0,0,0,0,0,0,193,193,0,193,193,0},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{64,0,64,64,129,129,0,129,193,64,129,129,129,193,129,0,0,0,0,0,0,0,0,0,0,0,193,193,64,129,193,
64},
{0,0,0,0,193,193,0,193,193,0,193,193,193,193,193,0,0,0,0,0,0,0,0,0,0,0,193,193,0,193,193,0},
{2,0,2,2,193,193,0,193,195,2,193,193,193,195,193,0,0,0,0,0,0,0,0,0,0,0,195,195,2,193,195,2},
{60,0,60,60,195,195,0,195,255,60,195,195,195,255,195,0,0,0,0,0,0,0,0,0,0,0,255,255,60,195,255,
60}
```

```
};


uint64_t red = 0;
uint64_t green = 0;
uint64_t blue = 0;
uint64_t white = 0;

void setup() {
  //Serial.begin(9600);
  DDRA |= 255;
  //DDRB |= 255;

  delay(3000);
}

void loop() {

   for(uint8_t x = 0; x < 10; x++) {
     SREG &= ~(1<<7);
     for(uint8_t column = 0; column < 96; column++) {
       for(uint8_t n = 0; n < 32; n++) {
         sendBit(pgm_read_byte(&attiny[column][n]));
       }
     }
     SREG |= 1<<7;
     showLEDs();
     delay(200);

     SREG &= ~(1<<7);
     for(uint8_t column = 0; column < 96; column++) {
       for(uint8_t n = 0; n < 32; n++) {
         sendBit(pgm_read_byte(&attinyfill[column][n]));
       }
     }
     SREG |= 1<<7;
     showLEDs();
     delay(200);
   }

   SREG &= ~(1<<7);
   for(uint8_t n = 0; n < 10; n++) {
     for(uint16_t x = 0; x < 384; x++) {
       writePixel(150, 0, 255);
       writePixel(0, 0, 0);
     }
     showLEDs();
     delay(30);
     for(uint16_t x = 0; x < 384; x++) {
       writePixel(0, 0, 0);
       writePixel(150, 0, 255);
     }
     showLEDs();
     delay(30);
   }

   for(uint16_t x = 0; x < 300; x++) {
     uint16_t currentPixelHue = x;   //moves rainbow pattern
     SREG &= ~(1<<7);
     for(uint8_t n = 0; n < 97; n++) {
       uint8_t phase = currentPixelHue >> 8;
       uint8_t s = currentPixelHue & 0xff;

       //rainbow pattern:
```

```
        switch (phase) {    //calculate all pixels then write them
          case 0:
            writePixel(~s, s, 0);
            break;
          case 1:
            writePixel(0, ~s, s);
            break;
          case 2:
            writePixel(s, 0, ~s);
            break;
        }
        currentPixelHue = (currentPixelHue + 10) % (3*256);   //increase hue by each led
position
      }
      SREG |= 1<<7;
      showLEDs();
      delayMicroseconds(1);   //speed control
  }

void writePixel(uint8_t r, uint8_t g, uint8_t b) {
  uint8_t w = minimum(r,minimum(g,b));
  sendByte(g - w);
  sendByte(r - w);
  sendByte(b - w);
  sendByte(w);
}

void sendByte(uint8_t data) {
  for(int8_t x = 7; x >= 0; x--) {
    sendBit(0 - ((data>>x) & 1));
  }
}

void sendBit(uint8_t data) {
  PORTA = 255;
  //PORTB = 255;
  __builtin_avr_delay_cycles(2);
  PORTA = data;
  //PORTB = data;
  __builtin_avr_delay_cycles(2);
  PORTA = 0;
  //PORTB = 0;
  __builtin_avr_delay_cycles(2);
}

void showLEDs() {
  PORTA = 0;
  //PORTB = 0;
  __builtin_avr_delay_cycles(1000);
}

uint8_t minimum(uint8_t a, uint8_t b) {
  return a < b ? a : b;
}
```

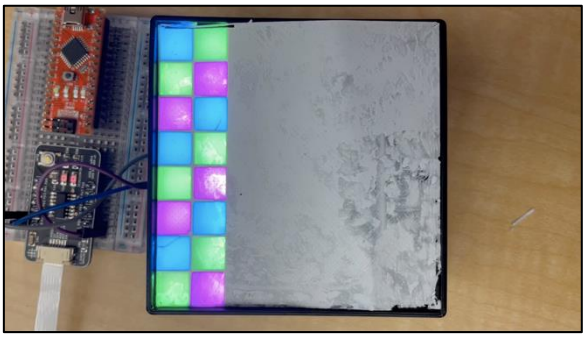
Processing

```
PrintWriter output;

int red[] = new int[24*32];
int green[] = new int[24*32];
int blue[] = new int[24*32];
int white[] = new int[24*32];

int pixelData[][] = new int[96][32];

int n = 0;

void setup() {
  size(100,100);
  PImage myImage = loadImage("attinyfill.png");
  image(myImage, 0, 0);

  output = createWriter("nyan1.txt");

  for(int y = 0; y < 24; y++) {
    for(int x = 0; x < 32; x++) {
      white[n] = min(int(red(get(x,y))), int(green(get(x,y))), int(blue(get(x,y))));

      red[n] = int(red(get(x,y))) - white[n];
      green[n] = int(green(get(x,y))) - white[n];
      blue[n] = int(blue(get(x,y))) - white[n];
      n++;
    }
  }


  //println(red[170]);
  //println(green[170]);
  //println(blue[170]);
  //println(white[170]);

  for(int x = 0; x < 96; x++) {
    for(int y = 0; y < 32; y++) {
      pixelData[x][y] = 0;
    }
  }
  for(int x = 0; x < 8; x++) {

    for(int y = 0; y < 32; y++) {
      for(int z = 0; z < 8; z++) {
        pixelData[y][z] |= ((green[y+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+8] |= ((red[y+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+16] |= ((blue[y+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+24] |= ((white[y+(x*96)] >> (7-z)) & 1) << x;
      }
    }

    for(int y = 32; y < 64; y++) {
      for(int z = 0; z < 8; z++) {
        pixelData[y][z] |= ((green[(95-y)+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+8] |= ((red[(95-y)+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+16] |= ((blue[(95-y)+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+24] |= ((white[(95-y)+(x*96)] >> (7-z)) & 1) << x;
      }
    }
    for(int y = 64; y < 96; y++) {
      for(int z = 0; z < 8; z++) {
        pixelData[y][z] |= ((green[y+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+8] |= ((red[y+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+16] |= ((blue[y+(x*96)] >> (7-z)) & 1) << x;
        pixelData[y][z+24] |= ((white[y+(x*96)] >> (7-z)) & 1) << x;
      }
```

```
    }
  }

  output.println("{");
  for(int x = 0; x < 96; x++) {
    output.print("{");
    for(int y = 0; y < 32; y++) {
      output.print(pixelData[x][y]);
      if(!(y == 31)) output.print(",");
    }
    output.print("}");
    if(!(x == 95)) output.println(",");
  }
  output.println("}");
  output.flush(); // Writes the remaining data to the file
  output.close(); // Finishes the file

  println(pixelData[0][1]);
  //println(green[96*0]);
  println(((green[96*0] >> 6) & 1) << 0);
  println(((green[96*1] >> 6) & 1) << 1);
  println(((green[96*2] >> 6) & 1) << 2);
  println(((green[96*3] >> 6) & 1) << 3);
  println(((green[96*4] >> 6) & 1) << 4);
  println(((green[96*5] >> 6) & 1) << 5);
  println(((green[96*6] >> 6) & 1) << 6);
  println(((green[96*7] >> 6) & 1) << 7);

}

void draw() {

}
```

Media

Connector board used for testing and soldering LEDs



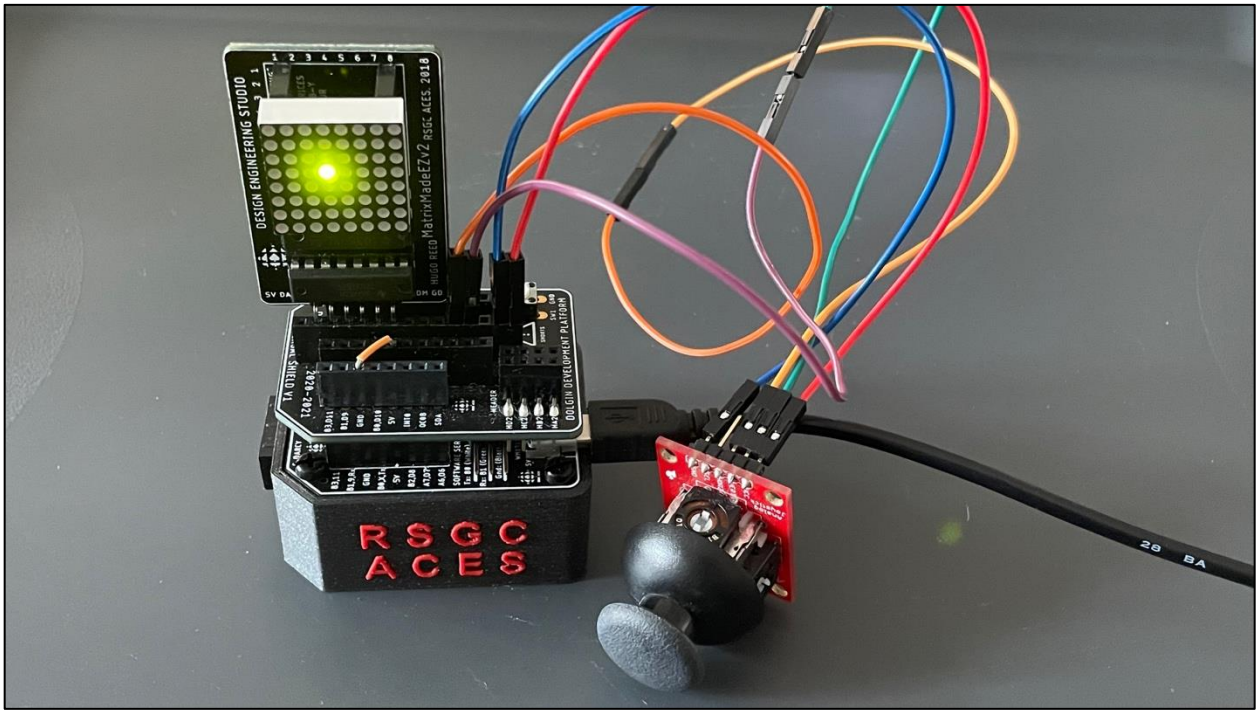
Lighting up some pixels of the small RGB matrix

YouTube link: https://youtu.be/KatHkq3PDNg

Reflection

Firstly, I apologize for submitting a day late. Though I hope to somewhat justify that with the effort put into filming and editing, I really did underestimate the amount of time I needed to put into this ISP. I got carried away with the complexity and ambitions of such a conceptually simple project which shows that I still need work on when to stop. Therefore I hope to gain a better sense of time, as I frequently found myself rushing through the soldering of this ISP. So, I will take my lessons learned in this ISP and use them to improve myself in the next and final ISP. On the positive side, I am truly happy with what I have achieved and I want to experiment more with how I can control the LEDs. A large panel opens up opportunities to interface it with anything such as a small camera monitor or data from IoT devices. Using the AVR microcontrollers and trying to push the limits of what it can do was quite fun and satisfying, especially when I got the ATtiny84 to control lots of work. A lot of the concepts recently learned in class applied to what I had to program, and such an exposure led me to come up with some ideas of some features I can later implement. For example, the mention of different communication interrupts to signal microcontrollers when data is finished sending led me to think that there can be multiple slave MCUs where their job is to send out neopixel data which would be received through a master device which could be another more powerful MCU or an interface on a phone. All in all, I hope to continue with this ISP, though it will have to be in the summer as the next ISP takes priority.

Project 3.7: TWAIN Advanced 2D



Purpose

The TWAIN (Task Without An Interesting Name) advanced 2-Dimensional version is a 100% assembly program which manipulates an LED in a 2D matrix with a joystick, emulating a joystick positioning system with an ATtiny84.

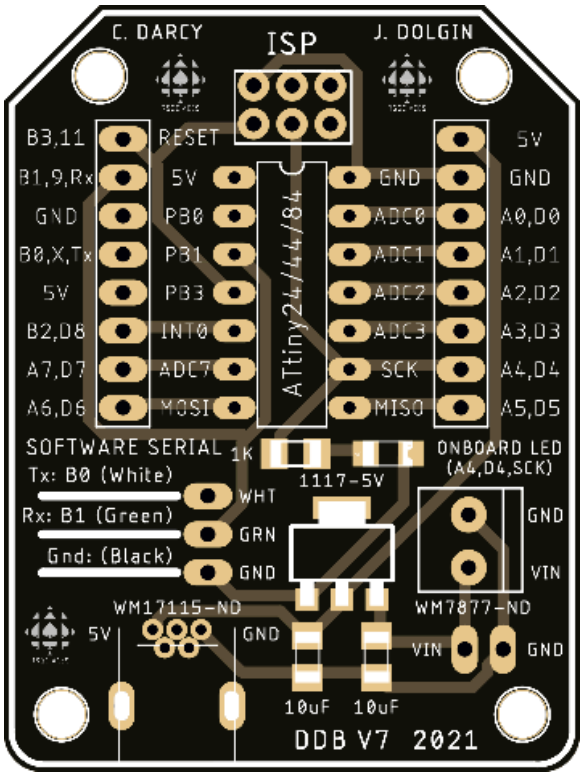
Reference

http://darcy.rsgc.on.ca/ACES/TEI4M/2122/Tasks.html
http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf

Procedure

The project is split up into input, processing and output sections, with the ACES Dolgin Development Board (DDB), created by ACES '20 Josh Dolgin, providing the foundation of the project instead of the regular UNO or NANO boards. While the Arduino development platform focuses mainly on the ATmega328P, the Dolgin Development Platform (DDP) uses the ATtiny84 MCU, the 14-pin version and younger brother of the ATMega328P.

Less ports, clock speed and other capabilities detract this MCU from the average electronic hobbyist, but suits those valuing efficiency in squeezing the most out of such a tiny chip or those whose devices don't require a lot of computational power. Nevertheless, the chip provides a great starting place in port manipulation and assembly with its minimalist features opening up new doors to other more powerful MCUs.



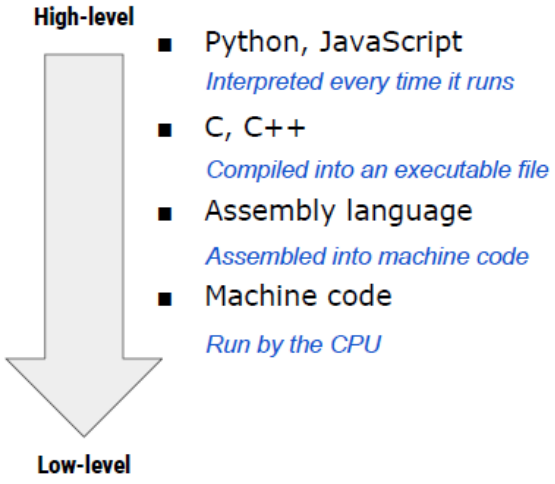
Like the ATtiny84, the DDB remains simplistic for highschool students to solder up and use. On the 7th current version, the DDBv7 has an onboard surface mount LED controlled by PA2 on the ATtiny84, a 5V voltage regulator and a terminal block and barrel jack for power. Unfortunately, there is no USB to serial converter IC, so an ISP pocket programmer is used to program the board where it connects to a 2 × 3 header. This also means there is no in-built serial communication so three external pins routed to the RX and TX pins serve as an optional serial output to an external serial to USB converter or another device. A USB type B connector is present but only serves to provide power.

The beauty of the DDBv7 is the option for ACES to create a variety of different stackable shields fitted for the DDBv7 for teaching and specific or general usage purposes, just like UNO shields greatly simplify a project by providing all the necessary components one needs. For this project, a universal shield where female headers allow for flexible wiring that houses the joystick and the matrix.

The input consists of the joystick, simply containing two potentiometers, one for the X-axis and the other one for the Y-axis as well as a button that activates when pressing down on the joystick. In total, there are five output pins, three for the two potentiometer analog outputs and the button output and the VCC and GND pins. This project does not use the button, only the two potentiometer outputs.

The ATtiny84 must then be programmed to read the analog values and convert them into digial values using the Analog to Digital Converters (ADCs) on the pins. All are situated on PORTA of the ATtiny84. In the setup, PA4 and PA5 serve as the pins reading the horizontal (x-axis) and vertical (y-axis) respectively.

There are three ways of coding the analog read. Using the high level language of the Arduino IDE, namely `analogRead()`, port manipulation of the ATtiny84 or AVR assembly language, the order of which increases in complexity and efficiency. High level languages totally mask the machine code with a compiler while port manipulation also does the same but to a lesser degree. Port manipulation is also specific to certain MCUs so it is less portable to other MCUs.

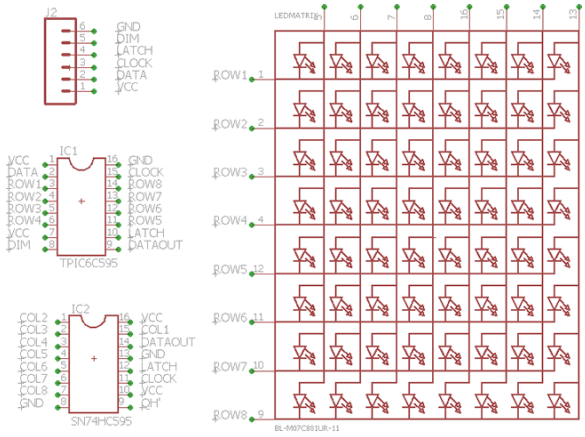


	low level *Assembler*	mid/high level *C/C++*	high level *Java/C#*	scripting *Python*
Development speed	→			
Performance	←			
Low-level optimization	++	+	–	–
Meta programming	+	++	–/+	+
Cross-platform support	individual code for each platform	compiled everywhere	compiled once, run everywhere	interpreted on many platforms
Supporting OOP	–	+	+	+
Type of linking	mostly static	static/dynamic	dynamic only	n/a

Assembly language is the ultimate form of low level code that most people write in. It is basically machine code that has some shortened words or acronyms associated with the instruction to give a description of what it does. The syntax of assembly is quite different to regular languages such as C and Java. It uses instructions quite like how high level languages use conditionals, variable assignments and math, however, the instructions are very specific. AVR assembly code line usually consists of an instruction and two operands where the instruction affects those two operands in some way. There are also other types of instructions that branch to different instructions in program memory by changing the program counter to jump to the instruction address and conditionals that affect certain bits in global registers. This is very reminiscent of the 4-bit CHUMP computer instructions. Like CHUMP, the instructions are stored sequentially in flash memory at an address near the top and but most take up eight bits of storage instead of four bits with the numerical value also taking eight bits as the ATtiny84 is an 8-bit controller. There are also other assembly languages specific to other controllers. For example, ARM controllers like the ones used by Raspberry Pi and STM32's are popular and can be coded in ARM assembly which is a more complex version of AVR featuring 32-bit commands. The full set of AVR instructions can be found in the references.

At first glance, the code seems simple enough to complete as it boils down to an ADC reading section, a processing section and an output section. These are quite easy to do in a high level language like C, using the built in `analogRead()`, and `shiftOut()` as the compiler breaks it down into (usually inefficient) machine code. But, coding these in assembly requires a lot more work and understanding of the specific MCU and its ports, registers and capabilities. In return, flexibility and efficiency is granted to those who choose to code in assembly, as each instruction is on the order of clock cycles (around 125 nanoseconds each for the internal 8 MHz ATtiny84 clock).

Essentially, the I/O ports are setup, by loading values into those ports, an ADC reading is started on the specified pin, and the value from that ADC reading is shifted right to convert to map it to a 3-bit value. This value determines the number of times to shift left in order to shiftout a single bit whose bit position corresponds to the mapped ADC value. Functions such as the ADC read and shiftout are called by jumping to an address in flash that contains the assembly code to do so. The assembly code with comments can be found in the code section, explaining the different ports and registers used on the ATtiny84 and how each line of assembly works.

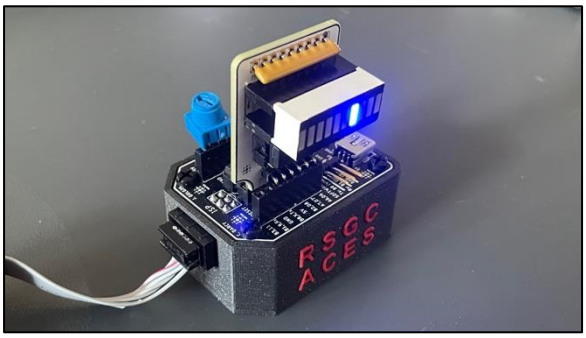


Speaking of the MatrixMadeEZ, the PCB was made by ACES '20 Hugo Reed. The layout and design is similar to the Morland bar graph but adds on a daisy chained TPIC6C595, a version of the SN74HC595 designed to sink current. Instead of a bar graph, it has an 8×8 matrix with 16 pins where they are evenly divided to control the eight rows and the eight columns. The output pins of the 74HC595 connect to the columns, providing power to the LED column anodes while the TPIC6C595 sink the current on the cathode side of the LED rows. Unfortunately, this configuration does not allow each LED to be controlled individually, so the user must implement POV where each of the rows continually cycle through being on, and only one row has a path to ground at any given time. The 74HC595 then shifts out the data corresponding to the row that is on and repeats the process eight times for the eight rows. This will create the illusion that the matrix is displaying all the data at the same time.

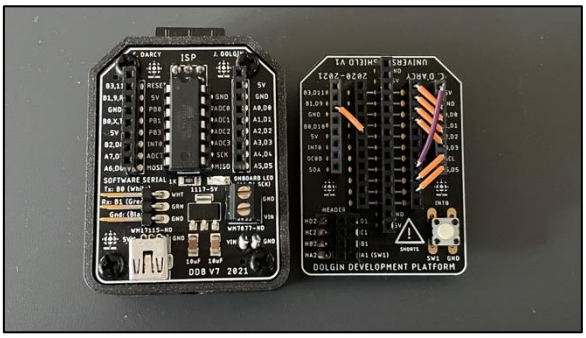
Since the project only shifts a dot around according to the joystick, POV is actually not needed. Instead, the 74HC595 shifts out the column value of the dot and the TPIC6C595 shifts out the row. So, the values presented will only have two bits on at a time.

At first, the 1 dimensional TWAIN was completed for an understanding of reading from the ADC and outputting a value to a shift register. This easier version does not need the universal DDBv7 shield and only takes in an input from one potentiometer. The ADC is read and converted from a 10-bit to a 3-bit value denoting the position of the LED on a bar graph where it is shifted out to only one 74HC595. Once the 1D version was complete, the 2D implementation simply involved another ADC read on the pin connecting the other joystick potentiometer and converting the shiftout function to a 16-bit version as the two shift registers are daisy chained. The low-byte for the horizontal ADC value is sent to the matrix columns and the high-byte for the vertical ADC is sent to the matrix rows. This systematic approach to completing the 2D TWAIN was an effective method as it allows oneself to split the project up into doable sections instead of trying to take in all of the complex steps at once.

Media



1 Dimensional TWAIN with the Morland bar graph and a potentiometer



The DDBv7 universal shield with wires alongside the DDBv7

YouTube video link: https://youtu.be/3-Q4QqpksrE

Code

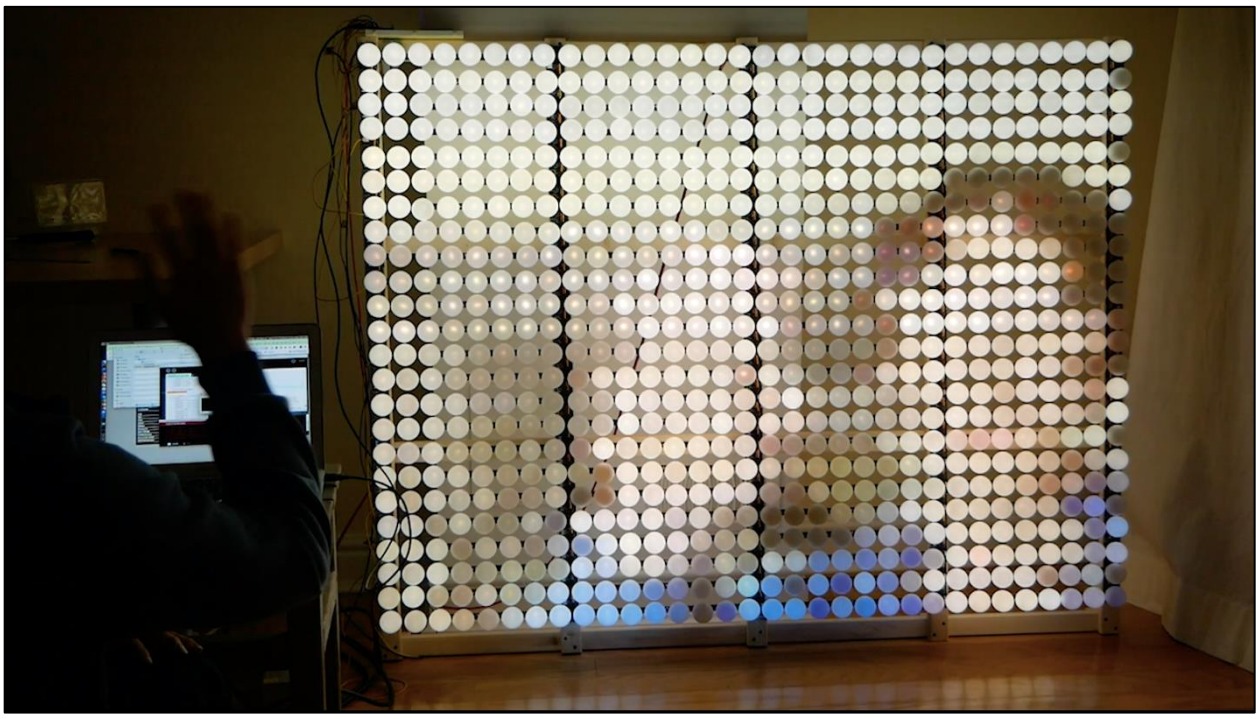
```
;PROJECT    : TWAIN 2D Advanced
;PURPOSE    : Joystick positioning system on the DDP (ATtiny84) to learn and practice assembly
;AUTHOR     : Xander Chin
;DATE       : 2022 03 30
;DEVICE     : Dolgin Development Platform. Version 7.
;MCU        : ATtiny84
;COURSE     : ICS4U
;STATUS     : Working
;REFERENCE  : http://darcy.rsgc.on.ca/ACES/Datasheets/ATtiny84.pdf
;NOTES      :
;.include   "prescalars84.inc"  ; assembly directive equivalent to compiler directive #include

.include   "prescalers84.inc"   ; assembly directive equivalent to compiler directive #include
.def        util    = r16       ; readability is enhanced through 'use' aliases for GP Registers
.equ        PIN     = PINA      ; its input register
.equ              VER               = PA5             ; vertical potentiometer
.equ              HOR               = PA4             ; horizontal potentiometer
.equ              DAT               = PA0             ; data pin for 595's
.equ              CLK               = PA1             ; clock pin for 595's
.equ              LTC               = PA2             ; latch pin for 595's
.equ              DIM               = PA3             ; output enable pin for 595's
.def              count             = r22             ; loop counter
.def              val               = r23             ; row/column value to be shifted out

.org        0x0000              ; start of Interrupt Vector Table (IVT) aka. Jump Table
    rjmp    setup               ; jump to flash address where code starts


setup:
ldi util, (1<<DAT)|(1<<CLK)|(1<<LTC)|(1<<DIM) ;setup pins for output (loads the immediate value specified into util register)
out      DDRA, util                          ; setup pins for output (puts util into the DDRA register)
cbi PORTA, DIM                               ; enable output (clear DIM bit in PORTA)
sbi PORTA, LTC                               ; set latch high (set LTC bit in PORTA)
```

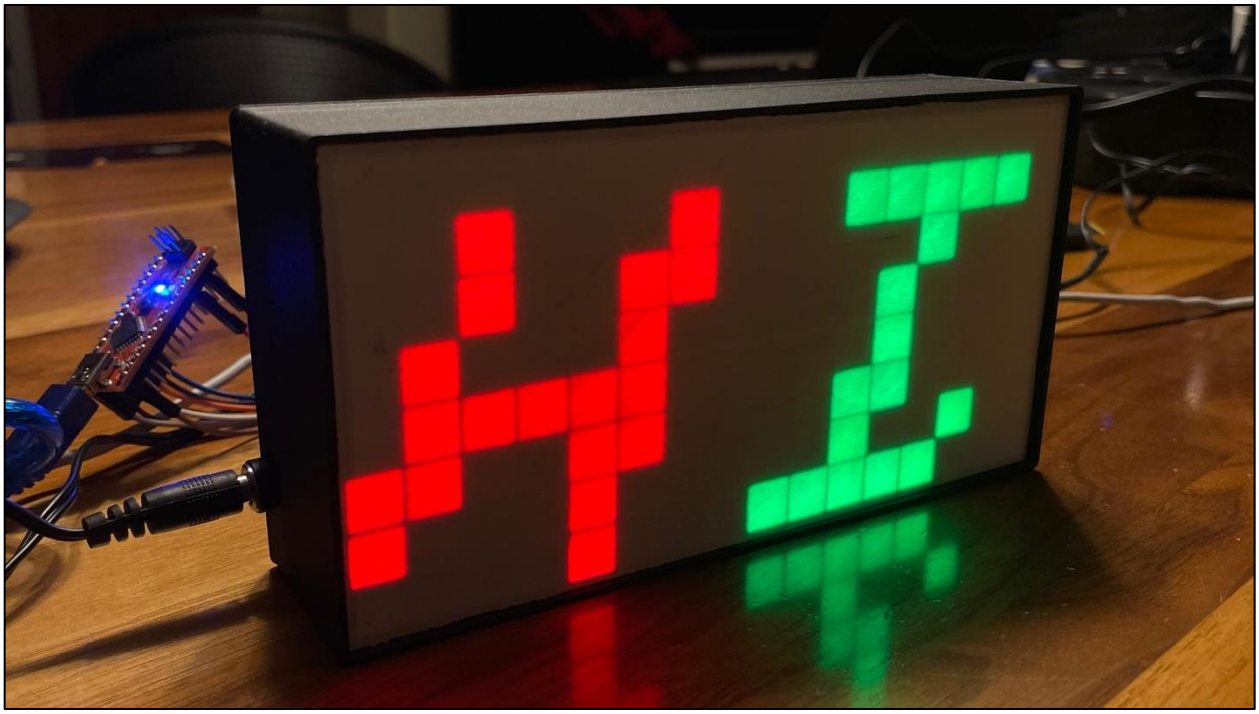
```
loop:                                   ; infinite loop
ldi util, (1<<MUX2)                     ; set PA4 to read ADC
rcall ADCReadShiftout                   ; gets ADC value from PA4, converts to col/row position, shiftout val

ldi util, (1<<MUX2)|(1<<MUX0)           ; set PA5 to read ADC
rcall ADCReadShiftout                   ; gets ADC value from PA5, converts to col/row position, shiftout val

cbi PORTA, LTC                          ; set latch low
sbi PORTA, LTC                          ; set latch high (outputs are active)
rjmp loop                               ; repeat process

ADCReadShiftout:
out      ADMUX, util                    ; set PA4 to read ADC
rcall ADCSetup                          ; setup ADC
in util, ADCH                           ; transfer from ADCH port to util
rcall ADCtoPosition                     ; convert value to column/row position
mov util, val                           ; move val into util for shiftout
rcall shiftOut                          ; shiftout
ret                                     ; return from function - pop pc from stack

ADCtoPosition:                      ; translates ADC value to row/column position on matrix
lsr util                            ; shift right, now 7-bit value (max: 127)
lsr util                            ; shift right, now 6-bit value (max: 63)
lsr util                            ; shift right, now 5-bit value (max:31)
lsr util                            ; shift right, now 4-bit value (max:15)
lsr util                            ; shift right now 3-bit value (max:7)
ldi val, 1                          ; load 1 to shiftout
shiftVal:                           ; shifting loop - shifts number of times in util
cpi util, 0                         ; if util == 0
breq doneShifting                   ; break
lsl val                             ; otherwise shift left
dec util                            ; decrease util by 1
brne shiftVal                       ; repeat again
doneShifting:                       ; done
ret                                 ; so return

shiftOut:                                   ; shift out 8-bit value
         ldi             count,8            ; so do the process 8 times...
next:
         cbi             PORTA,CLK          ; pull CLOCK low
         cbi             PORTA,DAT          ; assume BIT7 is low...
         sbrc     util,7                    ; did we guess right?
         sbi             PORTA,DAT          ; ahhh, we guessed wrong, make it high :)
         sbi             PORTA,CLK          ; pull the CLOCK HIGH!

         lsl             util               ; position the next blt for inspection...
         dec             count              ; one less to do!!!
         brne     next                      ; are we done?.
         ret                                ; return from function - pop pc from stack

ADCSetup:
ldi util, (1<<ADEN)|(1<<ADSC)       ;set adc enable, enable a conversion, set prescale value
out ADCSRA, util                    ;set adc enable, enable a conversion, set prescale value

WaitForADCComplete:                 ;wait until conversion is done
sbic ADCSRA, ADSC                   ;exit if ADSC is cleared from
rjmp WaitForADCComplete             ;otherwise keep waiting

ldi util, (1<<ADLAR)        ;shifts ADC value to the leftmost positions
out ADCSRB, util            ;do it. (ADC9 to ADC2 are now in ADCH) (ADC1 and ADC0 are the upper bits in ADCL)

ret                         ;return
```

Reflection

I am glad I got to complete a project with assembly as I really loved learning and coding with it. The precision you get is simply unparalleled, even with port manipulation in C, as you exactly know how long your code takes from the minimal clock cycle count of each instruction. This makes it great for sensitive timing control such as sending data to Neopixels. I feel like I have gone full circle in Grade 12 as the concepts in assembly relate so much to the work we did in the 4-bit CHUMP computer. This new entry opens up the doors to other more complex assembly languages, and I can't wait to try them out on the more powerful ARM and x86 processors. One more thing to note is the problem solving process and collaboration I did with another student (ACES '22 James Colraine) to challenge and push ourselves to complete the TWAIN 2D project. We came up with ideas on how to approach coding this, but ended up trying to take in too much information and code the whole sketch at once. We eventually decided to break it down into smaller pieces by first completing the easier 1D TWAIN and simply repeating some of the functions to account for the extra dimension. So not only was this project a huge milestone in my software journey but it also taught me that breaking up a complex project is key to completing it.

Project 3.8: (ISP – Long): Improving the Giant RGBW LED Matrix





Purpose

The purpose of this ISP is to add more features to the Giant RGBW Matrix by implementing assembly to really push the capabilities of the Arduino NANO. This includes live streaming serial data to the NANO, allowing for camera output, video streaming and drawing capabilities.

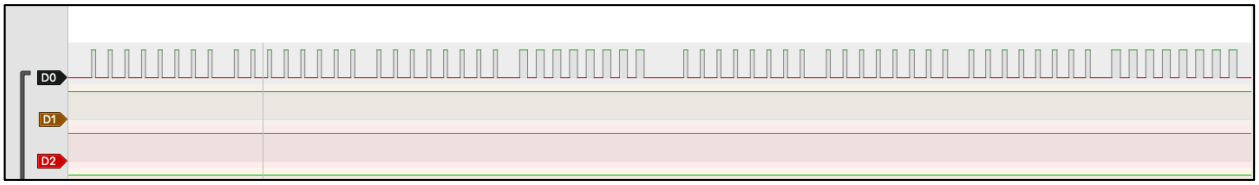
References

http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf

Procedure

Once finished building the wall and programming it in C, optimization of the software was next on the list as there were issues displaying words or patterns without using precious SRAM and read-only flash. More powerful MCUs like the ESP32 were considered but it was decided to see how much how far the trusty 8-bit 16 megahertz NANO can go. Port manipulation and high level C had reached its limit in optimization so AVR assembly was used to deal with the dirty work of transforming data and bit banging it onto the ports where clock cycle precision is required.

To help analyze the waveforms, An 8 channel logic analyzer was bought to measure and decode digital signals. It can read 8 channels simultaneously and up to 24 MHz and for its price of $13 on Amazon, it was well worth it. It is what made optimization possible as I could now visualize the subtle changes made to the code. For example, the top picture below is a waveform generated using the C code developed previously. Comparing it with the signals generated with assembly code represented as the bottom picture, it is obvious that there is a stark difference. These both produce the same effect of turning on an LED but with the logic analyzer, the difference was made clear in terms of efficiency and precision. The C code took around 95 microseconds to transmit four bytes of data while the assembly code took less than half that time at around 40 microseconds.



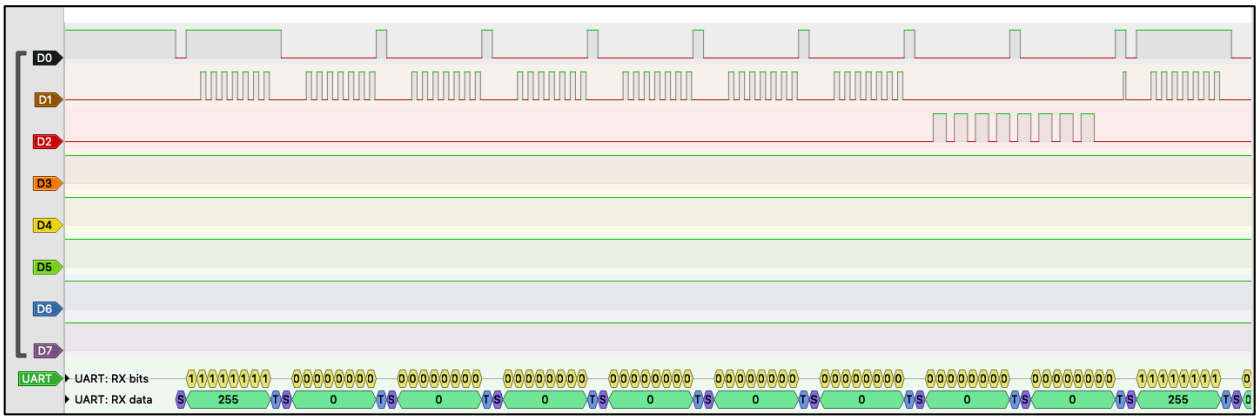
With this, numerous functions were created to output RGB data along with masking functions for scrolling messages. A mix of C, for ease of implementation and assembly, for prepping and bit banging the data, was used to get the best of both worlds. Assembly was at first done with the arduino inline assembler, which is complex, not well documented and not very pleasant to code in. Nevertheless code was still with it, but eventually discovered .s files and how they can be called from a C file, using the extern modifier, which allows for functions in different languages to be called upon.

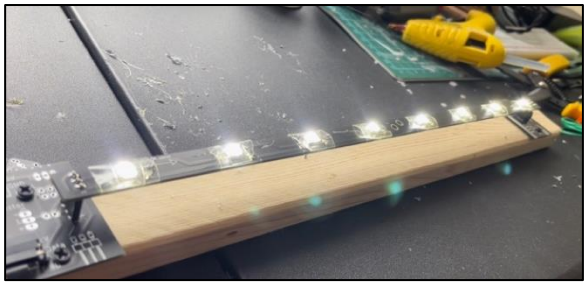
AVRAssemblySerialNeopixels § asmSerialNeo

```
10 extern "C" {
11   void asmSerialSend();
12   void test();
13 }
```

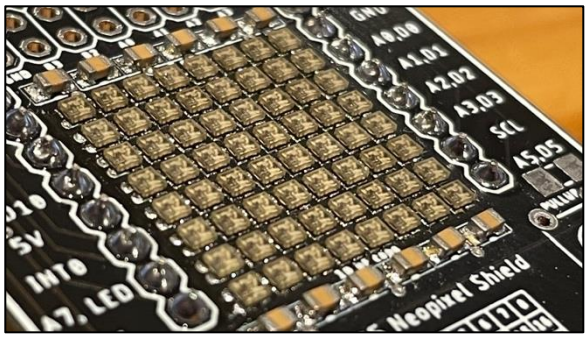
This was used in the serial live streaming sketch to call on an assembly function that gathered serial RGBW data from a computer or another MCU while simultaneously sending out the neopixel data, allowing for the aforementioned video and camera streaming without using a single byte of memory. It works by setting the serial baud rate to 2 million, the maximum baud rate an arduino NANO equipped with an FT232R USB to serial converter can handle. Other arduino development boards like the UNO or MEGA can only support up to 115200 baud rate with their USB to serial converters and therefore cannot receive data for live streaming fast enough.

The assembly code works by the NANO directly reading the RX pin on PD0 at set intervals, gathering data for 7 of the 8 bytes. On the 8th byte transmitted, the function simultaneously reads and bit bangs the 8 bits in parallel onto the whole of PORTB and the two highest bits on PORTA, as PD0 is being used to receive serial data, to stay within the 40 microsecond time frame in which the neopixels don't yet latch. A 0 bit is then required after every 8 bytes of data to prevent reading errors through accumulated error shifting in the serial communication. Reading from such high speeds with simultaneous transmission and data reorganization is only possible with assembly which is why the USART data register is not used to receive the data. Coding this took a very long time with timing experimentation and painstaking problem solving of getting to read the data properly and trying to output a clean waveform. Here is where some debug signals (channel D1), helped denote where the reading was taking place to help get the timing right.



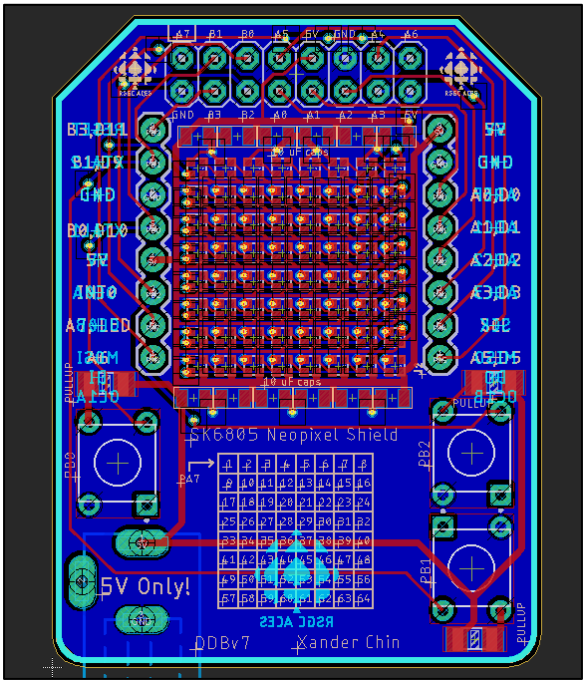


While coding, the ping pong balls were glued on in small every day increments where each one was tested afterwards. Bitluni, the creator of the original LED wall unfortunately reported that his LEDs were starting to die, probably from the glue. He mentioned using 3D printed diffusers but since a 3D printer wasn't handy and the ping pong balls were already bought, tape was put around the strips to at least physically protect the LEDs from the glue. So far, it seems to have done the trick and there have not been any problems, but if any LEDs needed replacement, the tape can be easily peeled off to expose the LED for replacement.

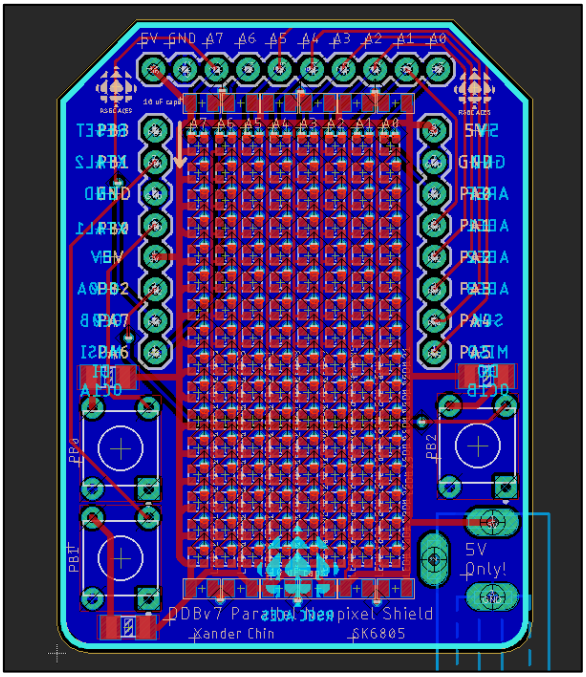


In the meantime, some prototype DDB neopixel shields were designed to practice assembly data bitbanging to some SK6805 1515 (1.5 mm by 1.5 mm) LEDs on the ATtiny84. These shields feature an 8 × 8 version controlled with PA7 and a 16 × 8 controlled in parallel with the entire attiny84 PORTA. Since the LEDs were so small, a stencil was ordered along with the PCBs and the PCB was oven baked. The LEDs still had to be placed on which was painful work, so in the future, a PCB manufacturing company will solder them on or larger ones such as the 2020 (2 mm by 2 mm) ones will be used for easier soldering.

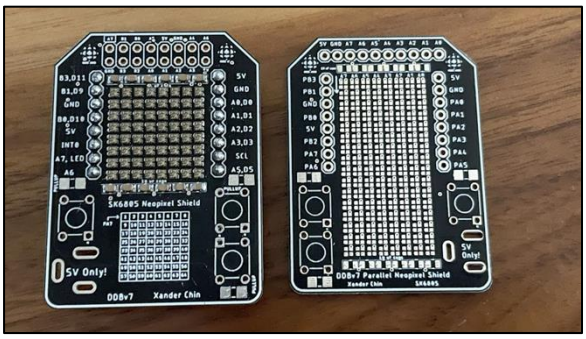
Media



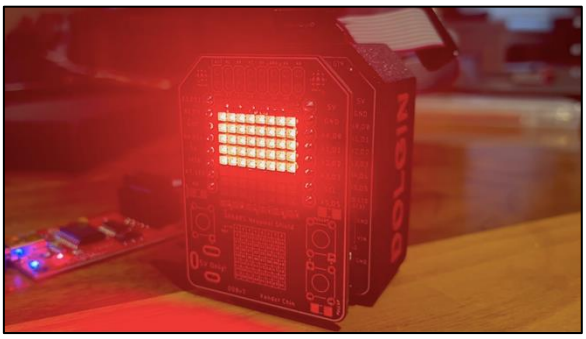
Traces of the 8 × 8 DDB serial shield



Traces of the 16 × 8 DDB parallel shield



The 8 × 8 serial shield and the 16 × 8 parallel shield



Part of the serial 8 × 8 neopixel shield working on the DDBv7

YouTube video link: https://youtu.be/1C1XGq-2Q_g

Code

Because of the many code files created, here is a link to my GitHub repository which contains all the sketches used.
https://github.com/xanderchinxyz/rgbwmatrix

Reflection

Although I didn't create a new device, I expanded a lot on my previous ISP with the work I did in assembly. I loved getting down into the details and solving a problem with the precision offered by assembly. Don't get me wrong though, there were times where I just couldn't figure out why the assembly code was behaving in strange ways. Half of the time it would be something simple like turning off global interrupts before doing any sort of time sensitive procedure. The other half would be lesser known problems where I had to tweak and test the assembler code one instruction at a time. The logic analyzer really helped with that and I have to say that it is probably the best tool in terms of bang for your buck as it just makes debugging and optimization so much easier for its cheap price. An exciting prospect about delving into complex and niche projects such as assembler usage for neopixel streaming is that you may be the first to do so, therefore cementing your role in contributing to the maker database of techniques. I don't think anyone has ever tried to or successfully "live streamed" data from a computer to an Arduino NANO so perhaps after cleaning up the code, I will write a Hackaday article on the subject. This is why I chose to delve deeper into this project, as you can contribute something truly unique.

Well here I am, still awake at 5 am and pulling my hopefully last all nighter, currently finishing off the final touches of my DER. And as the adrenaline and caffeine wears off, I must say that time flies and I can't believe that I am already at the end of Highschool and with it, my time at the ACES program. What a great three years it has been. I've learnt and done so much and experienced so many triumphs, failures and challenges. My ways of thinking have changed and through all the hard work, hardships and all-nighters pulled, I have gained some invaluable experience not just in Engineering but in communication through my reports and videos. This epic portfolio of wonderful projects and my new found my passion for engineering is what I owe to ACES program, the student support and its creator and facilitator Mr. D'Arcy. Although the ACES journey ends here, I will still continue to expand on my past projects and build new ones where they will be posted on my personal website and YouTube channel along with all the previous reports as I simply love creating. And don't worry, I will keep my DER updated as well. Thank you for coming along this journey with me.

Here is the link to my website: https://xanderchin.xyz/