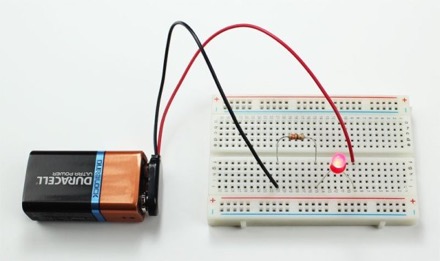
Design Engineering Report



Course: Computer Engineering

Code: ICS2O, ICS3U

Name: William Tessier

Date: May. 30, 20

Table of Contents

[Project 1. The 111 Voltage Divider 1](#_Toc41677506)

[Reference 1](#_Toc41677507)

[Purpose 1](#_Toc41677508)

[Procedure 1](#_Toc41677509)

[Media 1](#_Toc41677510)

[Reflection 2](#_Toc41677511)

[Project 2. The Capacitor Visualizer 3](#_Toc41677512)

[Reference 3](#_Toc41677513)

[Purpose 3](#_Toc41677514)

[Procedure 3](#_Toc41677515)

[Media 4](#_Toc41677516)

[Reflection 5](#_Toc41677517)

[Project 3. The Astable Multivibrator 6](#_Toc41677518)

[Reference 6](#_Toc41677519)

[Purpose 6](#_Toc41677520)

[Procedure 6](#_Toc41677521)

[Media 7](#_Toc41677522)

[Reflection 7](#_Toc41677523)

[Project 4. A Counting Circuit 8](#_Toc41677524)

[Theory 8](#_Toc41677525)

[Procedure 8](#_Toc41677526)

[A. Analog Input 8](#_Toc41677527)

[B. NAND Gate Oscillator (4011) 8](#_Toc41677528)

[Reference: http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4011.pdf 8](#_Toc41677529)

[C. Decade Counter (4017) 9](#_Toc41677530)

[Reference: http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4017.pdf 9](#_Toc41677531)

[D. Decimal Counting Binary Up/Down Counter (4510) 9](#_Toc41677532)

[Reference: http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4510.pdf 9](#_Toc41677533)

[E. Binary Counting Decimal Decoder (4511) 10](#_Toc41677534)

[Reference: http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4511.pdf 10](#_Toc41677535)

[F. Seven Segment Display 11](#_Toc41677536)

[Reference: http://mail.rsgc.on.ca//~cdarcy/Datasheets/7SegmentDisplay.pdf 11](#_Toc41677537)

[G. A Counting Circuit PCB 11](#_Toc41677538)

[H. A Counting Circuit PCB Case 11](#_Toc41677539)

[Media 12](#_Toc41677540)

[Reflection 13](#_Toc41677541)

[Project 5. 2-Digit Scoreboard 14](#_Toc41677542)

[Reference 14](#_Toc41677543)

[Purpose 14](#_Toc41677544)

[Procedure 14](#_Toc41677545)

[Media 15](#_Toc41677546)

[Reflection 16](#_Toc41677547)

[ICS3U 17](#_Toc41677548)

[Project 6. Traffic Light 18](#_Toc41677549)

[Purpose 18](#_Toc41677550)

[Reference 18](#_Toc41677551)

[Procedure 18](#_Toc41677552)

[Code 19](#_Toc41677553)

[Media 19](#_Toc41677554)

[Reflection 20](#_Toc41677555)

[Project 7. Binary Button Echo 21](#_Toc41677556)

[Purpose 21](#_Toc41677557)

[Reference 21](#_Toc41677558)

[Procedure 21](#_Toc41677559)

[Code 22](#_Toc41677560)

[Media 22](#_Toc41677561)

[Reflection 23](#_Toc41677562)

[Project 8. PoV Word 24](#_Toc41677563)

[Purpose 24](#_Toc41677564)

[Reference 24](#_Toc41677565)

[Procedure 24](#_Toc41677566)

[Code 25](#_Toc41677567)

[Media 28](#_Toc41677568)

[Reflection 29](#_Toc41677569)

[Project 9.1 Breadboard ATmega328P 30](#_Toc41677570)

[Purpose 30](#_Toc41677571)

[Reference 30](#_Toc41677572)

[Procedure 30](#_Toc41677573)

[Code 31](#_Toc41677574)

[Media 31](#_Toc41677575)

[Reflection 32](#_Toc41677576)

[Project 9.2 Altoids Arduino 33](#_Toc41677577)

[Purpose 33](#_Toc41677578)

[Reference 33](#_Toc41677579)

[Procedure 33](#_Toc41677580)

[Code 34](#_Toc41677581)

[Media 34](#_Toc41677582)

[Reflection 35](#_Toc41677583)

[Project 10. MatrixMadeEZ 36](#_Toc41677584)

[Purpose 36](#_Toc41677585)

[Reference 36](#_Toc41677586)

[Procedure 36](#_Toc41677587)

[Code 38](#_Toc41677588)

[Media 39](#_Toc41677589)

[Reflection 39](#_Toc41677590)

[Project 11. Legacy PCB/Appliance 40](#_Toc41677591)

[Reference 40](#_Toc41677592)

[Inspiration 40](#_Toc41677593)

[Procedure 40](#_Toc41677594)

[Media 40](#_Toc41677595)

[Reflection 41](#_Toc41677596)

[Project 12. ATtiny85 Binary Game 42](#_Toc41677597)

[Purpose 42](#_Toc41677598)

[Reference 42](#_Toc41677599)

[Theory 42](#_Toc41677600)

[Procedure 43](#_Toc41677601)

[Code 44](#_Toc41677602)

[Media 46](#_Toc41677603)

[Reflection 47](#_Toc41677604)

[Project 13. DC Motor Tachometry 48](#_Toc41677605)

[Purpose 48](#_Toc41677606)

[Reference 48](#_Toc41677607)

[Theory 48](#_Toc41677608)

[Procedure 48](#_Toc41677609)

[Code 49](#_Toc41677610)

[Media 50](#_Toc41677611)

[Results 51](#_Toc41677612)

[Reflection 51](#_Toc41677613)

[Project 14. Time-Sensitive Mechanics 52](#_Toc41677614)

[Purpose 52](#_Toc41677615)

[Reference 52](#_Toc41677616)

[Stepper Motors 52](#_Toc41677617)

[Procedure 52](#_Toc41677618)

[Code 54](#_Toc41677619)

[Media 55](#_Toc41677620)

[Reflection 55](#_Toc41677621)

[Project 15. Game Buzzers 56](#_Toc41677622)

[Purpose 56](#_Toc41677623)

[Reference 56](#_Toc41677624)

[Theory 56](#_Toc41677625)

[Procedure 56](#_Toc41677626)

[Code 57](#_Toc41677627)

[Media 58](#_Toc41677628)

[Reflection 58](#_Toc41677629)

# Project 1. The 111 Voltage Divider

## Reference

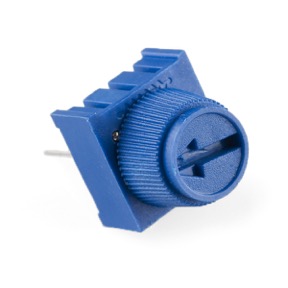
Project Description: <http://darcy.rsgc.on.ca/ACES/TEL3M/1819/TasksSpring.html#111>

## Purpose

The purpose of this project is to build a working voltage divider and to understand how it works, thus advancing collective knowledge on the subject of voltage, resistance, and voltage division.

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| Breadboard | 1 |
| 1kΩ Resistor (Watt) | 3 |
| 10k Potentiometer | 1 |
| Yellow LED | 2 |

To build this circuit you must first acquire all the components in the parts list located to the right as well as some extra wires. Start by removing the breadboard and joining both outer sides of the breadboard thus, joining power on each side. After completing this step as efficiently as possible (use of wires), take the variable resistor (potentiometer), and place it on the top of the board. Next, connect wires so that the outer legs of the potentiometer are directly connected to positive and negative on the breadboard. Plug in the 2 yellow LEDs so that the longer lead (positive) is on the same row as the middle lead of the potentiometer. Next, take the other LED and plug in the shorter lead (negative) onto the same row as the previous positive lead and middle potentiometer lead. Now, find the 3 1kΩ resistors and plug one of the resistors leads into positive on the breadboard and the other lead into the first LED mentioner’s negative lead. Take another 1kΩ resistor and plug one of the leads into the same row as the previous step and the other lead into the secondly mentioned LEDs positive lead, thus running the resistors in series with the LEDs. Finally take the last 1kΩ resistor and plug it into ground on the breadboard. Then take the other lead and plug it into the same row as the second mentioned LEDs positive lead and other resistors row. Now that the circuit is wired, plug in a 9V battery to the breadboard. Next, twist the potentiometer to see if it works. When twisting the knob, you are dialing a specific voltage between 9V and 0V. As you twist the knob dialing 9V and 0V. one light should turn on, turn off, and the other should turn on as you approach the other side of the knob.

## Media

<https://youtu.be/E_gVxP1gkl4> - A link to a video on this project.

|  |  |
| --- | --- |
| Front View | Rear View |
|  |  |

## Reflection

After completing my first DER, it seems as though hardware will be a very different course than normal. This project, the first of many was very nerve racking as I am not sure if what I am writing is acceptable or ‘correct’. Many aspects of this project worry me such as the fact of how little my media section is whether the reference section I have is enough or I am missing something. I am at comforted by the clear fact that in time, and through comments that I will understand and learn how to write these projects. I learned a lot from this project ranging from the uses of resistors to what a potentiometer is. Hopefully this process will get easier in time.

# Project 2. The Capacitor Visualizer

## Reference

Project Description: <http://darcy.rsgc.on.ca/ACES/TEL3M/1819/TasksSpring.html#capacitor>

## Purpose

The purpose of this project is to deepen/cement our understanding of what a capacitor is, how it works, its storage capabilities and a simple application of it through our prototype and finished product.

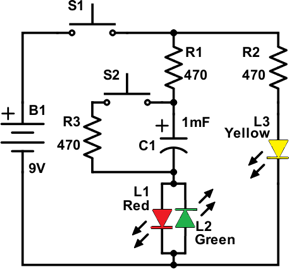
## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| Breadboard | 1 |
| 470Ω Resistor (Watt) | 3 |
| 1000μF Capacitor | 1 |
| Yellow LED | 1 |
| Bicolor LED (Green/Red) | 1 |
| Momentary Push Button Switch | 3 |

Before you begin assembling the “Capacitor Visualizer” prototype, you must first understand how the circuit you are building works and what it is.

A capacitor is equivalent to a glass of water. When it is connected to the battery and being supplied with power, it acts as a glass and fills up with ‘water’ (charge). When the battery stops filling the glass, the glass is poured out and acts as a battery to the rest of this circuit. While emptying, it empties at a specific rate: τ (Tao). The length of τ (time constant) can be determined by the formula: . Where R is the resistance of the resistor wired in series in front of the capacitor and C is the size of the capacitor. Every τ fills up approximately 63% of the capacitor. At 5τ we reach approximately 99%. Below is a chart representing the relationship explained above. This formula works for direct current (DC), capacitors are also useful in alternating current (AC) as they act like a window, letting sound through but not rocks; allowing voltage to pass when full but not current when completely full.

|  |  |  |  |
| --- | --- | --- | --- |
| **Resistance (Ω)** | **Capacitance (μF)** | **Theoretical 5τ (s)** | **Observed 5τ (s)** |
| 1000000 | 100 | 500(s) | 523.2(s) |
| 100000 | 100 | 50(s) | 57.1(s) |
| 470 | 1000 | 2.35(s) | 3.9(s) |



Now that you understand how a capacitor works, begin by procuring all of the parts listed in the parts table above. Place a wire connecting positive power on your breadboard to a button, this will act as your master switch to the rest of the circuit. Next, connect two 470Ω resistors (R1 and R2 in the diagram to the right) to your master switch. Continue by adding a yellow LED (Light Emitting Diode) with the positive lead (longer stem) connected to one of the resistors. Next, plug your 1000μF Capacitor into your breadboard with the positive lead on the same line as the other 470Ω resistor. Add your bicolor LED in, connected to the capacitor and have the other lead, in a separate row on the breadboard, connected to the other LEDs lead. Then connect the bicolor LED to ground on the battery from the same row on the breadboard that the yellow LED was connected to. Now that we have a direct connection through the capacitor and yellow LED, you can click the master switch button and see the yellow LED turn on with the red coming on the bicolor LED. Release the button and watch the red turn green and the yellow slowly fade. As this is taking a while, we must add a release to drain our capacitor and empty our glass of water. Start by connecting the positive lead on the capacitor to a button and finish by connecting a new 470Ω resistor to the button and to the negative lead on the capacitor. To test, press the master switch, release and click the other button, everything should (hopefully) turn off.

## Media

A link to a video on the project - <https://youtu.be/1pODyEbooKU>

|  |  |
| --- | --- |
| Front View | Rear View |
|  |  |
|  |  |
|  |  |

## Reflection

This project proved to be useful as we learned what and how capacitors work. I am glad we learned about basic uses such as the elementary changing color on a bicolor LED and how they store energy and discharge it. In this project we also learned about τ, and how it applies to capacitors as a time constant. I enjoyed this project and it had very few challenges. One challenge that required time and concentration was how to initially wire it to be able to access both switches in the prototype.

# Project 3. The Astable Multivibrator

## Reference

Project Description: <http://darcy.rsgc.on.ca/ACES/TEL3M/1819/TasksSpring.html#astable>

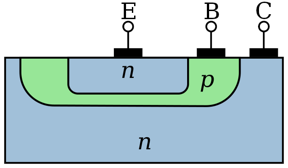
## Purpose

The purpose of this project is to understand how a transistor works and to be able to apply our knowledge to build an analog oscillator circuit with 2 LEDs.

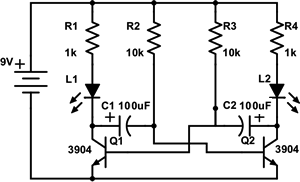
## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| Breadboard | 1 |
| 10kΩ Resistor (Watt) | 2 |
| 1kΩ Resistor (Watt) | 2 |
| 100μF Capacitor | 2 |
| LEDs | 2 |
| 2N3904 Transistor (NPN) | 2 |

Before beginning the construction of your Astable Multivabrator AKA your Analog Oscillator, you first need to be able to understand how everything works.

Before 1947 the only way to turn a circuit on temporarily was with a button. Noticing issues in bounce; the metal bouncing very quickly but causing long term wear and issues with fine sensors, William Shockley, John Bardeen and Walter Brattain invented a not mechanical switch known as a transistor. The transistor works by receiving a small amount of current in the base leg (middle leg) and converting an insulator to a conductor connecting the 2 outside legs.

To the right is a visualization of the inside of a NPN (3904) transistor, the one used in this project. The p section of it clearly divides the other n sections keeping your other pins; emitter and collector, separate from each other. This arrangement with p in the middle is referred to as an NPN. In this project the transistor is used as an electronic switch but it can also be used as an amplifier.

Begin construction of your prototype analog oscillator by collecting all the parts mentioned in the parts list above. Start by grabbing your breadboard and connecting a 1KΩ from positive to the positive lead on one of your LEDs. Have that lead connect to the positive lead one of your 100μF capacitors and the C leg (On an NPN, the C leg is the one on the right if the flat part of your NPN is closest to you.) to one of your transistors (Q1). Connect the capacitors negative lead to a 10KΩ resistor (R2) and have that resistor go back to positive. Connect Q1s E leg to ground on your breadboard. Now, do the exact same thing with some distance to your current circuit such as it shows through symmetry on the circuit diagram above. Next connect C1s negative lead to Q2s (Your other transistor) base pin (B leg), and connect C2s (Your other capacitors) negative lead to Q1s base pin. Now that you are done, add power and watch your LEDs oscillate. The oscillating happens due to how capacitors allow less and less current through as they fill up and when one fills up (C2), an electrical signal is stops being sent (to Q1) allowing the other LED (L1) to turn on. When C1 fills up the same thing happens resulting in an oscillating current (The time of each LED being on is determined by the capacitors and R1 + R4, read Project 2 if you do not understand).

## Media

A link to a video on this project: <https://youtu.be/qicmHTSRUAw>

|  |  |
| --- | --- |
| Front View | Rear View |
|  |  |
|  |  |
|  |  |

## Reflection

This flashy project was really enjoyable to build and then turn a prototype into the final product shown above. Understanding why it oscillated was difficult to understand but easy to build and set up. It was fun playing around with different resistors and capacitors to augment the speed of this flashy project! The biggest challenge I encountered when completing this project was by screwing the screws into my Josh Dolgin printed case. To solve this problem, I had to stick a soldering iron at (220°) into the inserts already in the case and move them around such that I could actually screw my screws in! I enjoyed this project and might possible use the oscillating current in my future ISP.

# Project 4. A Counting Circuit

## Theory

The idea behind the counting circuit is an analog input (button click) triggering one capacitor to fill up and discharge as a different capacitor repeatedly charges and discharges creating a square wave. The square wave then gets divided by ten and inputted and counted in binary format. This binary format is sent to a decoder which converts binary format to base ten decimal format and is displayed on a seven segment display. And with the initial click of a button, we get a base ten number increasing or decreasing on a display.

## Procedure

### Analog Input

|  |
| --- |
| **Analog Input Visual** |
|  |

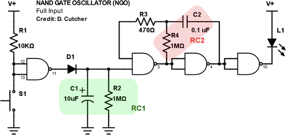
To create an analog input we can use in this circuit, we need it to be on at rest and off with input. To do this you will need a button, a wire, and a large resistor. We will put this in pull up resistor configuration. This means that we have a 10KΩ resistor plugged into power running directly to an LED which is connected to ground. The button, when clicked, connects ground to this resistor providing new negatives to our LED leaving it with 0V and turning off.

#### Purpose

The purpose of this aspect of the circuit is to provide a manual input to our counting circuit. The pull up resistor configuration is perfect for creating a NAND Gate Oscillator (NGO).

### NAND Gate Oscillator (4011)

### Reference: <http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4011.pdf>

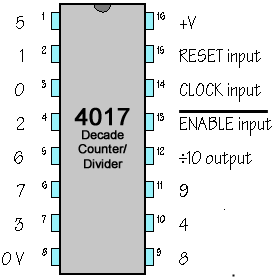
To create the second part of this circuit, we require logic gates. To create an oscillating signal, we will use the NAND 4011 IC (Integrated Circuit), the NAND logic gate only sends a low signal when its two inputs are high, meaning that when it is attached to our analog input, sends out a low signal at rest. Now that we have plugged our IC in, we need to attach two more pieces to this circuit to get an oscillating signal. RC1, comes off of a diode (D1 shown to the right) coming from the output of the NAND gate previously used. RC1 is made by using one 10μF capacitor with the positive lead attached to the end of the diode and a resistor in parallel with the capacitor with one end in ground and the other connected to the capacitor’s positive lead. With RC1 completed, connect the positive lead of the capacitor to a different input of a NAND gate. For RC2, connect a 470Ω resistor to the other input. Run this resistor to a 1MΩ resistor and a 0.1μF capacitor. Connect the 1MΩ resistor to 2 inputs of a NAND gate, having both inputs of the gate connected will change a NAND gate to a NOT gate. Hook up the output of the gate to the other lead of the 0.1μF capacitor and bring this output to 2 inputs of another NAND gate. Finally connect the oscillating output of this NAND gate to an LED to have a visualization of the square wave. The square wave produced has a duration relative to the size of RC1s capacitor and a frequency relative to the other capacitor and the other 1MΩ used in RC2. The duty cycle of this square wave is 50% as it is giving a high signal for the same time period as a low signal. To build this circuit efficiently, it is recommended to use the pinouts above.

#### Purpose

The purpose of creating this oscillating signal is to be able to change our number at a given rate. Without this clock signal, our circuit will not be able to count.

### Decade Counter (4017)

### Reference: <http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4017.pdf>

The 4017 Decade Counter IC acts as a processor for a square wave. The IC has the external input of a square wave and 11 different outputs. This chip counts ten pulses and spreads these counts out, meaning the first of 10 pulses given will be outputted on one lead of the IC then shut off until the other 9 leads have finished their pulses. This can be best shown if you were to line up ten pencils (parallel) and roll a ball past them. The outputs of the ICs leads can be shown as when the ball is directly in front of a pencil. The part of this IC we will use is the ÷10 output on pin 12. This pin is used to only send a rising edge at the same time output 0 would start and the high lasts till output 5’s rising edge. For normal use of this IC ensure that pins 8, 13 and 15 are all grounded and pin 16 is positive. To add this to the counting circuit, wire the IC as indicated and connect the square wave output of the NGO to the clock input at pin 14 on the 4017 IC. The ÷10 output is what will be used in this counting circuit.

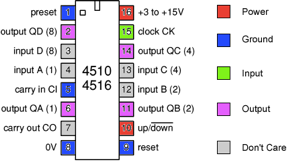
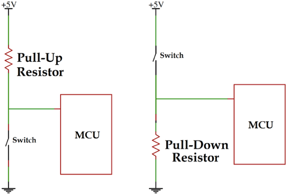
#### Purpose

The 4017 IC is used to slow down our square wave input for the rest of the counting circuit.

### Decimal Counting Binary Up/Down Counter (4510)

### Reference: <http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4510.pdf>

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

The 4510 IC also known as the Binary Counter chip takes a clock input and counts it in binary as shown above. As it counts these pulses, it outputs them in binary coded decimal format (BCD format). This means that on the first counted pulse, it sends a high on pin 6. On the second count, a high on pin 11 (Output 2) and a low on pin 6 (Output 1). On the 7th count, it sends a rising edge on pins 14, 11, and 6. Resulting in 0111 binary count in the chip. The 4510 chip only counts from 0-9, then it goes back to 0 and starts again. Should you which to count from 0-15, you can use the 4516 IC, it has the same pinouts but counts higher. For normal use with either chip, keep pins 1, ,5, and 9 ground, should you require a reset button, it is recommended to use a pull down resistor configuration with pin 9, this means keeping a resistor running from ground to pin 9 and having a button connect pin 9 and positive. Make sure pins 8 and 16 are in ground and positive respectively. For pin 10, you should keep it positive to count upwards or negative for downwards counting. To control this easily, a switch is recommended connecting pin 10 to either ground or positive.

In relation to the counting circuit we are building, the final product will have both a reset button and an up/down switch. To incorporate this IC ensure that all the pinouts are properly accounted for and pin 12 of the 4017 IC is directly connected to pin 15 of the 4510 IC thus transmitting our outputted clock signal to our clock input on this IC.

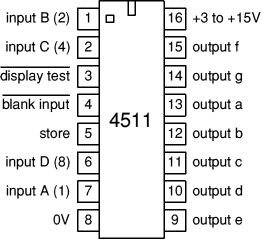
#### Purpose

The purpose of this IC is to be able to actually count the clock signals we produce so that will be able to display them on a 7 segment display.

### Binary Counting Decimal Decoder (4511)

### Reference: <http://mail.rsgc.on.ca//~cdarcy/Datasheets/CD4511.pdf>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Pins to**  **Attach** | **4510 IC** | 6 | 11 | 14 | 2 |
| **4511 IC** | 7 | 1 | 2 | 6 |



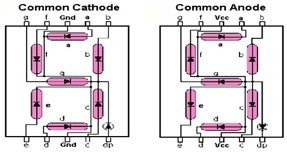
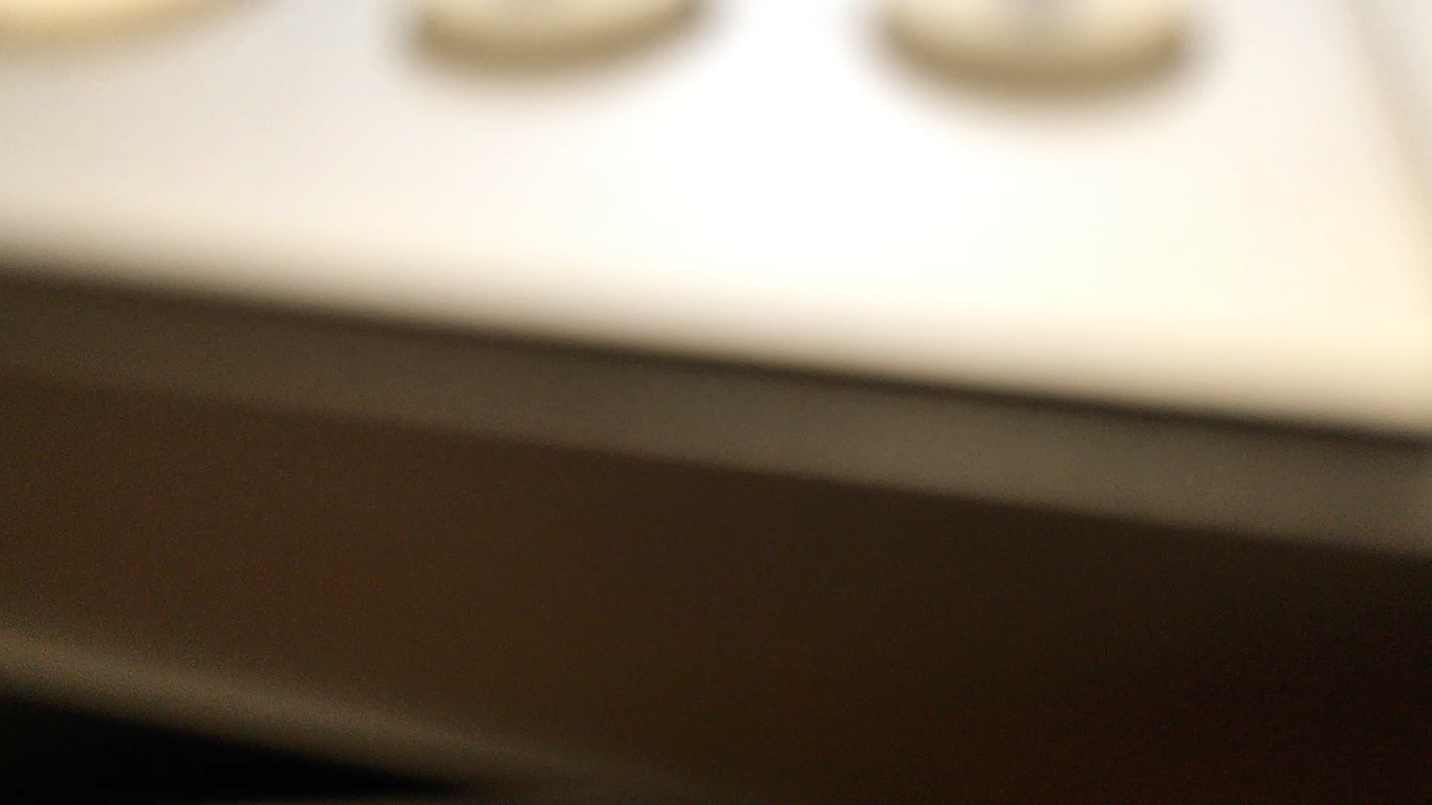
The 4511 IC also known as the decoder, decodes BCD format and converts it to base 10 and the outputs it to a seven segment display visual. For the purpose of our counting circuit we need to connect the outputs of the 4510 IC to the inputs of the 4511 IC shown above. For normal use, run positive to pins 3,4 and 16, and ground to pins 5 and 8.

#### Purpose

The purpose of this IC is to convert BCD format from our 4510 IC into a seven segment display visual. This way we can have our counting circuit function properly.

### Seven Segment Display

### Reference: <http://mail.rsgc.on.ca//~cdarcy/Datasheets/7SegmentDisplay.pdf>

The seven segment display is considered one of the more basic visualizers for a real number instead of an LED. The seven segment display is made up 8 different LEDs each with their own pin, and 2 pins that are either grounded or put to positive. A common cathode display means that all of the LEDs share a ground and are given positive, common anode is the reverse. The seven different LEDs mainly shown, are named A-G. The 8th LED is the decimal point.

The outputs of the 4511 IC A-G should be connected directly to the outputs on the common cathode seven segment display we are using. Pins 3 or 8 of the display should be connected to ground for proper use.

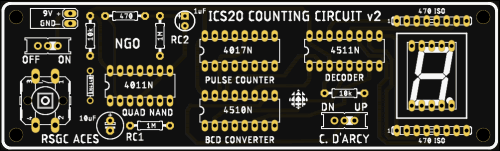
#### Purpose

The purpose of this display is to have a visual for our counting circuit.

### A Counting Circuit PCB

#### Purpose

The PCB shown below designed by the RSGC ACES community is meant to securely hold all the parts mentioned above in a secure, compact, and elegant form.



### A Counting Circuit PCB Case

#### Purpose

The 3D printed Josh Dolgin case shown below is solely for the purpose of guarding our PCB as elegantly yet compactly as possible. The case also provides housing for an ‘inhouse’ battery.



## Media

A link to a video on the different visualizers for steps on this project: <https://youtu.be/1EIJhoN9iUw>

|  |  |
| --- | --- |
| **NAND Gate Oscillator Visual** | **4511 IC Visual** |
|  |  |
| **4510 IC Visual** | **Complete Counting Circuit** |
|  |  |
| **Completed Counting Circuit on PCB with Dolgin Case** | |
|  | |

## Reflection

This has been one of the most enjoyable school projects I have ever had. I started this project as soon as I heard about it and had time to refine my design to a compact, working breadboard prototype. ICs were a wonderful aspect of electrical engineering I had never used or known about. It was really fun learning about how they work and the many different ICs used in this project. Pull up resistors were my biggest difficulty in this project and the steps established by Mr. Darcy, my hardware teacher, to complete the NGO were greatly appreciated.

# Project 5. 2-Digit Scoreboard

## Reference

4026 IC: <https://www.ti.com/lit/ds/symlink/cd4026b.pdf>

LB-602 A / K2: <https://mail.rsgc.on.ca/~cdarcy/Datasheets/lb-602ak2.pdf>

## Purpose

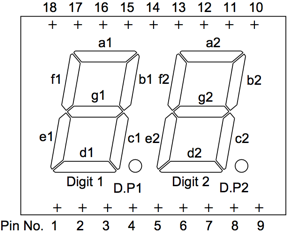
The purpose of this project is for me to expand my own knowledge on DC circuits and venture out of my own comfort zone. My specific ISP came from the idea to be able to record a number via analog input. This project was achieved due to myself constantly looking o improve my own project.

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| Breadboard | 1 |
| 470Ω Resistor (Watt) | 14 |
| 10kΩ Resistor (Watt) | 2 |
| PBNO | 2 |
| CD4026B IC | 2 |
| LB-602 A / K2 | 1 |

Before beginning the procedure on how this project works, you should get the parts listed to the right and learn about how the theory works.

The 4026 IC (Integrated Circuit) is a very versatile IC, it counts clock pulses, divides them by ten, and emits the count in an A-G, 7 segment display pinout. Pin 8 and 16 should be grounded and put to positive respectively to provide potential positive and negative outputs. Pins 3,4 should be put to positive. Pins 2 and 15 should be put to ground with a 10kΩ resistor to act as pull down resistor configuration for a future reset button. The second digit in our scoreboard will come from one of the 4026 ICs pin 5 (÷10) output being the input for the other 4026 ICs clock input. This way when our analog input sends one signal to one of the 4026 ICs, it will output the ones digit visually while the ÷10 input will function as the tens digit.

To begin construction on this scoreboard circuit, begin by hooking up a pull up resistor configuration to our ones digit IC. Connect our pull up resistor configuration to the clock input of one of the 4026 ICs. Wire the 4026 ICs as mentioned earlier leaving pin 14 afloat. Bring the ÷10 pin of the ones digit IC to the clock input of the tens digit IC. After wiring the ICs carefully and correctly, we will add our reset button. To add our reset button to this circuit, add a pull down resistor configuration to both of the ICs pin 15 (reset). Now that our IC’s are hooked up, we can plug in our LB-602 green dual seven segment display. The pinout diagram to the right shows the respective titles for the LEDs in the display. The ones digit IC is connected to A2-G2 and tens digit IC to A1-G1 pins. The complete pinout is listed in the reference section above. The K2 of the display refers to the display being a common cathode display meaning that the LEDs share ground leads. To complete the circuit, ground pin 14 and 13.

## Media

A link to a video on this project: https://youtu.be/6aOY2FsG8lY

|  |  |
| --- | --- |
| **Breadboard Prototype** | |
|  | |
| **Final Product With Case** | **Final Product Without Case** |
|  |  |

## Reflection

My first ISP has been very tedious yet enjoyable. While I have made few mistakes in my ISP, my biggest one and only notable mistake was forgetting to solder in the connections of my final product’s heat shrunk resistor connections from ICs to the display. While I have struggled in this project, I loved doing this project. Playing with logic is really fun and in the process for building and preparing for his IC I experimented with many different chips and maybe got off track a few times… Throughout this project, the luckiest moment was when I was the 1 in 10 person to be selected to present 2 days later than when I was planning on presenting. In those two days I designed and printed the case you see in the media section. This was extremely fortunate and I feel lucky to have had that opportunity.

# ICS3U

# Project 6. Traffic Light

## Purpose

This simple Traffic Light project is meant as an intro to software on Arduino. Through this project a basic understanding of the Arduino coding language is gained.

## Reference

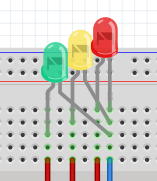
Project Description: <http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#TrafficLight>

Description of PCB: <http://darcy.rsgc.on.ca/ACES/PCBs/index.html#TrafficLight>

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| Arduino Uno | 1 |
| Schaffer Traffic Light PCB | 1 |
| Red LED (10mm) | 1 |
| Green LED (10mm) | 1 |
| Yellow LED (10mm) | 1 |

Before you begin construction on your Traffic Light you must aquire the parts listed in the table to the right. Now that you have the necessary tools, you may wish to breadboard this before saudering your Schaffer Traffic Light PCB. To start first setup 3 LEDs which represent your red, green, and yellow LED of your completed project. Place them so that they are in a common cathode formation; this means all LEDs share a ground pin. Now that your bread board is hooked up, draw wires from your Arduino Uno outputs to your 3 LED positive pins and ground pin. With this complete, all that is left is writing your code.

In your Arduino Sketch, you will first want to define some variables. Create an unsigned integer that only occupies 8 bits of RAM by typing: *uint8\_t*. Set each colour of your traffic light and your ground output to an output pin that is run to your breadboard. Now, we must create an interval. This duration will be the space in between the different LEDs being on and off. For this use the same function as before except give it 16 bits of RAM for numbers up to 65535. After defining your interval we must type define what is being used as what, this code goes in the setup area. We must set our pins that are being used as outputs, to do this type: *pinMode(red, OUTPUT);*. For each variable excluding your interval variable type this. Now we must set our ground pin as a constant low: *digitalWrite(gnd, LOW);* this goes in our setup area as we only need to do it once. For our main body of looping code we must turn each light on, then off for the given interval while making our yellow light 4 times slower. To turn on pins we need a *HIGH* output from one of our female headers on the Arduino, and we use the *digitalWrite();* function mentioned above. To complete turn on and off each (excluding ground/interval) variable with the *delay(interval);* statement in between. For an example of the completed code, see below in ‘Code’ section.

## Code

// Title.  : Traffic Light

// Course   : ICS3U

// Author   : William Tessier

// MCU      : 328P9

// Date     : 2019 09 26

// Status   : Working

uint8\_t red = 11;         //Using 8 bits of RAM to remember variable

uint8\_t yellow = 10;      //'red' (or other) is set to the given value: 11.

uint8\_t green = 9;

uint8\_t gnd = 8;

uint16\_t interval = 2000; //Using 2 bytes of RAM to remember variable

                         //'interval' is given the value of 2000 millis.

void setup() {

 pinMode(red, OUTPUT);   //Defining our variables as output pins

 pinMode(gnd, OUTPUT);

 pinMode(yellow, OUTPUT);

 pinMode(green, OUTPUT);

 digitalWrite(gnd, LOW); //Setting our gnd pin as a ground output

}

void loop() {

 digitalWrite(green,HIGH); //'green' on for duration of 'interval'

 delay(interval);

 digitalWrite(green,LOW);  //'green' off, 'yellow' on for a quarter

 digitalWrite(yellow,HIGH);//the duration of 'interval'.

 delay(interval/4);

 digitalWrite(yellow,LOW); //'yellow' off, 'red' on for the duration

 digitalWrite(red,HIGH);   //'interval'.

 delay(interval);

 digitalWrite(red,LOW);    //'red' off and program loops to 'green' on.

}

## Media

A link to a video on this project: <https://youtu.be/vGNz0VDxAPY>

|  |  |
| --- | --- |
| **Arduino and Schaffer Traffic Light PCB** | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Reflection

I think this straightforward project was a good choice to introduce the basics of Arduino coding to us. Arduino seems like a good way to introduce code to students everywhere and this project was a good first step in our new journey to write good code. Learning about how the basic int function is innefecient was fun and how different languages use different storage capacities for the function. This project was an overall good introduction to Arduino and I look forward to the next one project.

# Project 7. Binary Button Echo

## Purpose

The purpose of completing the Binary Button Echo is to create a program that monitors the state of 8 different buttons, then it echoes the state of the buttons to an LED bargraph through the 595 Shift Register IC.

## Reference

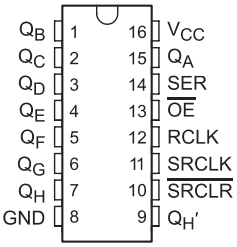
Project Description: <http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#BinaryButtonEcho>

Description of PCB: <http://darcy.rsgc.on.ca/ACES/PCBs/index.html#ShiftBarTHTV3>

SN74HC595 IC Data Sheet: <http://mail.rsgc.on.ca/~cdarcy/Datasheets/sn74hc595.pdf>

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| Arduino Uno | 1 |
| Morland Bargraph PCB | 1 |
| Momentary Tactile Button | 8 |
| 470 Ω Resistor (Watt) | 8 |
| 330 Ω Resistor Network | 1 |
| 10-LED Blue Bargraph | 1 |
| SN74HC595 IC | 1 |

To begin work on this project, first complete all the hardware components before moving on to the software components. To start, wire up 8 buttons coming off of 8 female ports from your Arduino Uno in a pull up resistor fashion. To achieve pull up resistor configuration, have a resistor connect ground, to a button, to your Arduino so that when you push the button, you are connecting a high voltage, to your low attached to the button. Following this setup you will need to properly solder all of the parts left, to your Morland Bargraph PCB. On the PCB are marks indicating orientation for each part, be careful as if you put in say, the bargraph, in the wrong way, your output will not be working.

The Morland PCB holds the 595 IC, the Shift Register. This IC (featured to the right), takes Serial in, then outputs it all as parallel out when a high is sent to the latch pin. To set up the IC for normal use, you must attach pins 8, 13, and 10 to ground. Connect pin 16 to a high, and have all your Q outputs attached to your designated output. Once you complete your soldering of the PCB, all that is left will be the code.

Begin by setting variables for each of the female ports the PCB will use according to the earlier explanation. Create an array of variable for each of female ports your buttons will occupy. Moving on to the setup area of our Arduino program. For this section assign all the ports that the PCB uses as an OUTPUT and set the ones that will not change throughout the code (such as a ground pin). INPUT is the default selection so you will not need to set your array. Inside your main loop statement, create a variable that will be the number that you display on your bargraph. Then set your latch variable to a low. Now create a for loop that will run until n reaches the size of the array, in this case: 8. Because we will be sending the total size of each button pressed added together to be displayed on the bargraph, we will need to take the variable we created earlier, then add 2 to the power of the count of the for loop we are on, and multiply it by a 1(true) or 0(false) which we get by checking whether the button has been pressed. Perform this check by typing digitalRead(buttonInput[n]), it will return a 1 or 0. After the for loop has completed it’s cycle, we can now send this variable which has the culmination of every button to the bargraph. To do this, we use the shiftOut function as such: shiftOut(dataPin, clockPin, MSBFIRST, value). Then to complete the program set latch high to send the data to be outputted.

## Code

// Title  : Binary Button Echo

// Course : ICS3U

// Author : William Tessier

// MCU    : 328P9

// Date   : 2019 10 19

// Status : Working

uint8\_t gnd = A5;

uint8\_t clockPin = A4;

uint8\_t constHigh = A3;

uint8\_t data = A2;

uint8\_t latch = A1;

uint8\_t gnd2 = A0;

uint8\_t constHigh2 = 13;

uint8\_t buttonInput[] = {9, 8, 7, 6, 5, 4, 3, 2};

// Creating an array of variables for each button input

void setup() {

**Serial**.begin(9600);

 pinMode(gnd, OUTPUT);       // Setting up all pins that

 digitalWrite(gnd, LOW);     // will be used.

 pinMode(clockPin, OUTPUT);

 pinMode(constHigh, OUTPUT);

 digitalWrite(constHigh, HIGH);

 pinMode(data, OUTPUT);

 pinMode(latch, OUTPUT);

 pinMode(gnd2, OUTPUT);

 digitalWrite(gnd2, LOW);

 pinMode(constHigh2, OUTPUT);

 digitalWrite(constHigh2, HIGH);

 //You don't need to define the buttonInput[] as input is the default

}

void loop() {

 uint8\_t sumOfButtons = 0; //Creating a variable and setting it to 0

 digitalWrite(latch, LOW); //Resets IC

 for (uint8\_t n = 0; n < 8; n++) { //runs 8 times

   sumOfButtons = sumOfButtons + (1<<n) \* digitalRead(buttonInput[n]); }

 shiftOut(data, clockPin, MSBFIRST, sumOfButtons);

 digitalWrite(latch, HIGH); //Tells IC to display the byte to the bargraph

}

## Media

A link to a video on this project: <https://youtu.be/REu-_MdKZMU>

|  |  |
| --- | --- |
| **Binary Button Echo Project** | |
|  |  |
|  |  |
|  |  |

## Reflection

I really struggled with this project. I would like to say it was fun, but it was frustrating until I finally finished. Then the satisfaction of completing something difficult sunk in as I thought, why didn’t I use the bit shift operand to start with. I began by using the pow(base, exponent) function but for some reason everytime my value exceeded 2 I would start getting 1 less then the desired outcome. While this project seemed straightforward at first, I am glad I got to experience struggling with my code in high school before I eventually head off to university.

# Project 8. PoV Word

## Purpose

The purpose of this project is to learn to use female ports on an Arduino more efficiently. Through using a display of the 14-segment display, two 595 shift registers, and 2 transistors we can gain valuable experience in how to use the Arduino more efficiently.

## Reference

Project Description: <http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#PoV>

SN74HC595 IC Data Sheet: <http://mail.rsgc.on.ca/~cdarcy/Datasheets/sn74hc595.pdf>

LTP-3784G (14-Segment Display): <https://mail.rsgc.on.ca/~cdarcy/Datasheets/LTP-3784G.pdf>

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| Arduino Uno | 1 |
| LTP-3784G (14-Seg Display) | 1 |
| SN74HC595 IC | 2 |
| 10 KΩ Resistor (Watt) | 2 |
| 2N3904 Transistor (NPN) | 1 |
| 2N3906 Transistor (PNP) | 1 |

The idea of this circuit is that two inputted character inputs are used to find segment maps that describe how to display these letters. Then these segment maps are outputted to two 595 shift register IC to then be displayed on a 14-Segment

Display. To display two different characters on the display, a persistence of vision effect needs to be created. This visual phenomena is caused by LEDs flashing on and off at a speed high enough that the human eye perceives them as both being on at the same time. Thus we can send the character ‘A’ to the shift register to be outputted, turn off the half of the display it is being displayed on, then output the other character ‘B’, display it, and through the repetition of this action, it will appear as two different characters displayed at the same time.



To begin assembly on this circuit, first wire the 595 ICs with the necessary pins grounded, and put to a high. Next, wire the carry pin (9) of one of your 595 ICs to the data pin (14) of the other IC, this ensures that if we send this IC 16 bits, the 8 it will not be able to hold will be sent to the other IC. Wire IC’s latch and clock pins to the other IC’s latch and clock pins respectively. Next, connect the output pins of your 595 ICs to your 14-Segment Display. Finally, wire your transistors as shown to the right. It does not matter which one is which as long as they are different, if they are not different a persistence of vision effect will not be achieved. Connect wires that touch the 10KΩ resistors and connect these to each of the common cathode pins (16, 11) on the display.

Before writing more code, write a segment map that contains exactly which segments of the display to be turned on to look like a desired letter. Next write them to EEPROM so that it will be easier to call them in the future. To allow your Arduino to access EEPROM remember to type: #include <EEPROM.h>, and include the library. In a separate sketch, to write the code, start by defining the optimal outputs for your Arduino to shift register. These would include defining the clockPin as 13, and data and latch pins as 11 and 10 respectively, remember to define them as OUTPUT in your setup phase. Create two char integers, these will become your inputs later on, and a uint8\_t variable called ‘x’. This variable will serve as a way to count even and odd for when you need to set two inputs with 1 going to the first char and the second input to the second char. Tell the Arduino to wait until there are inputs in the Serial monitor by typing while (!Serial.available()); once there are inputs available in the Serial monitor, we will need to read them. First lets look at what is available without getting rid of it by using the Serial.peek() function. If the input is between ‘A’ and ‘Z’ or ‘a’ and ‘z’, set it to one of your character variables, then increase x by 1. Use an if/else statement to only make it a variable if it is the first or second variable, this way we are left with 2 different variables, each being one of the inputs. To display them we first set a switchPin to low so that only half of the display is on, then shiftOut() the segment map (from EEPROM) of the char we are using, remember to call it twice, for each segment map of the desired letter. Next, set the switchPin to high and repeat but this tme with the other char variable. Put these in a loop to run indefinitely and we will achieve the back and forth, PoV effect we have been seeking.

## Code

// Title  : PoV Word

// Course : ICS3U

// Author : William Tessier

// MCU    : 328P9

// Date   : 2019 11 08

// Status : Working

#include <**EEPROM**.h>

uint8\_t clockPin = 13;  //Defining optimal pins

uint8\_t switchPin = 12;

uint8\_t dataPin = 11;

uint8\_t latchPin = 10;

void setup() {

**Serial**.begin(9600); //Run the Serial connection at 9600 baud...

 while (!**Serial**); //Waits for the Serial monitor to be ready...

 pinMode(clockPin, OUTPUT); //Conditioning pins.

 pinMode(dataPin, OUTPUT);

 pinMode(latchPin, OUTPUT);

 pinMode(switchPin, OUTPUT);

}

//Establishing variables and setting them to zero for later...

char ch1 = 0;

char ch2 = 0;

uint8\_t x = 0;

void loop() {

**Serial**.println ("Input any two letters... "); //Give two inputs...

   while (!**Serial**.available());

   if ('A' <= **Serial**.peek() &&  **Serial**.peek() <= 'Z') {

     if (x % 2 == 0) { //Checks if its even, this is to alternate

       ch1 = **Serial**.read();  // characters to different variables.

     } else {

       ch2 = **Serial**.read();

     }

     x = x + 1;

   }

   if ('a' <= **Serial**.peek() &&  **Serial**.peek() <= 'z') {

     if (x % 2 == 0) {

       ch1 = **Serial**.read() - 'a' + 'A';

     } else {

       ch2 = **Serial**.read() - 'a' + 'A';

     }

     x = x + 1;

   }

   if (x == 2) {

**Serial**.println ("Thanks for the inputs... ");

     while (true) { //Now output the letters we have inputed.

       digitalWrite (switchPin, LOW); //First letter on, Second letter off

       digitalWrite (latchPin, LOW);  //Resets IC

       shiftOut (dataPin, clockPin, LSBFIRST, **EEPROM**.read(ch1));

       shiftOut (dataPin, clockPin, LSBFIRST, **EEPROM**.read(ch1 + 26));

       digitalWrite (latchPin, HIGH); //Outputs bits/byte in IC

       //Above sends the first letter to be displayed, below is the second

       digitalWrite (switchPin, HIGH);

//Second letter on, First letter off

       digitalWrite (latchPin, LOW);

       shiftOut (dataPin, clockPin, LSBFIRST, **EEPROM**.read(ch2));

       shiftOut (dataPin, clockPin, LSBFIRST, **EEPROM**.read(ch2 + 26));

       digitalWrite (latchPin, HIGH);

     }

   }

}

// Title  : Write14SegASCIIEEPROM

// Course : ICS3U

// Author : William Tessier

// MCU    : 328P9

// Date   : 2019 11 09

// Status : Working

#include <**EEPROM**.h>

uint8\_t segmentMap[] = {

 0b01110110, //A1 (0) 'A'

 0b01111001, //B1 (1)

 0b01001110, //C1 (2)

 0b01111001, //D1 (3)

 0b01001110, //E1 (4)

 0b01000110, //F1 (5)

 0b01011110, //G1 (6)

 0b00110110, //H1 (7)

 0b01001001, //I1 (8)

 0b00111100, //J1 (9)

 0b00000110, //K1 (10)

 0b00001110, //L1 (11)

 0b00110110, //M1 (12)

 0b00110110, //N1 (13)

 0b01111110, //O1 (14)

 0b01100110, //P1 (15)

 0b01111110, //Q1 (16)

 0b01100110, //R1 (17)

 0b01011000, //S1 (18)

 0b01000001, //T1 (19)

 0b00111110, //U1 (20)

 0b00000110, //V1 (21)

 0b00110110, //W1 (22)

 0b00000000, //X1 (23)

 0b00000000, //Y1 (24)

 0b01001000, //Z1 (25)

 0b00100010, //A2 (26  ['A' + 26])

 0b00101000, //B2 (27)

 0b00000000, //C2 (28)

 0b00001000, //D2 (29)

 0b00100010, //E2 (30)

 0b00100010, //F2 (31)

 0b00100000, //G2 (32)

 0b00100010, //H2 (33)

 0b00001000, //I2 (34)

 0b00000000, //J2 (35)

 0b01010010, //K2 (36)

 0b00000000, //L2 (37)

 0b01000001, //M2 (38)

 0b00010001, //N2 (39)

 0b00000000, //O2 (40)

 0b00100010, //P2 (41)

 0b00010000, //Q2 (42)

 0b00110010, //R2 (43)

 0b00100001, //S2 (44)

 0b00001000, //T2 (45)

 0b00000000, //U2 (46)

 0b01000100, //V2 (47)

 0b00010100, //W2 (48)

 0b01010101, //X2 (49)

 0b01001001, //Y2 (50)

 0b01000100, //Z2 (51)

};

void setup() {

**Serial**.begin(9600);

 while (!**Serial**);

 for (uint16\_t i = 0; i < sizeof(segmentMap); i++) {

**EEPROM**.write(i + 'A', segmentMap[i]);

 }

}

void loop() {

}

 0b00100001, //S2 (44)

 0b00001000, //T2 (45)

 0b00000000, //U2 (46)

 0b01000100, //V2 (47)

 0b00010100, //W2 (48)

 0b01010101, //X2 (49)

 0b01001001, //Y2 (50)

 0b01000100, //Z2 (51)

};

void setup() {

**Serial**.begin(9600);

 while (!**Serial**);

 for (uint16\_t i = 0; i < sizeof(segmentMap); i++) {

**EEPROM**.write(i + 'A', segmentMap[i]);

 } //Writes each variable in array to EEPROM

}

void loop() {

}

## Media

|  |  |
| --- | --- |
| **PoV Word Project** | |
|  |  |
|  |  |
|  |  |

A link to a video on this project: <https://youtu.be/yGkmSg3EBbs>

## Reflection

This is without doubt, the hardest project so far. It scares me that it is only rated at 3 diamonds. After many long hours and reflecting on past projects did I finally realize how to solve the most challenging problem of the project, two inputs. Taking 2 inputs from Serial and classifying them as each being their own variable took up too much of my Friday which was supposed to not be a school day. The challenge was fun however I do not feel like I have succeeded as everytime you want to input and display two characters, you need to reupload the code onto the Arduino. I am curious to see how my classmates approached this problem and look forward to the feedback email that will be put out.

# Project 9.1 Breadboard ATmega328P

## Purpose

The purpose of this project is to gain experience in taking the ATmega328P off of the Arduino to learn to make more compact designs that will be vital to making small cases in the future. This specific project is merely a test in using the ATmega328P in a breadboard environment.

## Reference

Project Description: <http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#standalone>

ATmega328P Description: <https://mail.rsgc.on.ca/~cdarcy/Datasheets/ATmega328P.pdf>

SN74HC595 IC Data Sheet: <http://mail.rsgc.on.ca/~cdarcy/Datasheets/sn74hc595.pdf>

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| ATmega328P | 1 |
| LED (any colours) | 8 |
| SN74HC595 IC | 1 |
| 10 KΩ Resistor (Watt) | 2 |
| Momentary Tactile Button | 1 |

This the base idea behind this project being fairly vague, the actual project being constructed is a binary LED counter. With LEDs arranged in a row it becomes possible to display 256 different arrangements. The goal of the circuit is to have 8 LEDs display counting from 0 to 255 then back, using LEDs positioned in a binary sense.

To begin contruction on our circuit we first need to properly wire the ATmega328P chip. Wire pins 7, 20, and 21 to high and pins 8, and 22 to low. Wire pin one with a pull down resistor configuration to achieve a proper reset button.

Next, wire up the 595 IC with the proper wiring meaning that pins 16, and 10 are high and pins 8, and 13 are low. Then connect data, latch, and clock pins from your 595 IC to optimal pins coming from the ATmega328P chip. Ideal pin setups are pin 19 for clock (11), pin 11 for data (14), and pin 10 for latch (12). Finally to complete the circuit, connect LEDs from each output of the 595 IC to ground so that when given a high, they will turn on.

For the code, simply define and set the pins for data, latch, and clock as outputs. Create one more variable and make sure it is signed so that it can be negative, then set it to one. Finally, write a for loop that starts at zero and increases by the variable defined earlier. In the loop, shiftOut() the iteration number the loop is on. After the for loop finishes, just set the variable to a negative version of itself. When this is inside the main loop area of an Arduino sketch, it will run indefinetly counting from 0 to 255 then back to 0.

## Code

// Title  : LED Byte Counter

// Purpose: To create code to show a byte increasing 1 by 1

// Author : William Tessier

// Date   : 2019 11 23

// Status : Working

typedef uint8\_t u8;

typedef int8\_t i8;

u8 clockPin = 13;

u8 dataPin = 11;

u8 latchPin = 10;

i8 x = 1; // Rate of counting

uint16\_t delayRate = 500;  // Delay between different numbers

void setup() {

 pinMode(clockPin, OUTPUT);  // Defining optimal pins

 pinMode(dataPin, OUTPUT);

 pinMode(latchPin, OUTPUT);

}

u8 n = 0;

void loop() {

 for (n; 0 <= n < 255; n += x) { // Increasing by x

   digitalWrite(latchPin, LOW);

   shiftOut(dataPin, clockPin, MSBFIRST, n);

   digitalWrite(latchPin, HIGH);

   delay(delayRate);

 }

 x = -x; // Counts up, then counts down, then back up...

 n = n + x; // Allows repetition

}

## Media

A link to a video on this project: <https://youtu.be/871YezNccoE>

|  |  |
| --- | --- |
| **Binary LED Byte Counter** | |
|  |  |
|  |  |
|  |  |
|  |  |
| **For Uploading directly to the chip** | **9 Volt to 5 Volt Adapter** |
|  |  |

## Reflection

I did not enjoy this project as much as I should have. Most people who know me will understand my dislike for more creative projects. While I enjoy other creative aspects in life such as music, figuring out what my project will be then being graded against a rubric I have never seen before is terrifying to me and quickly diminishes my enthusiasm. While I view it as a necessary project in learning to use the ATmega328P chip, I regret not planning more so I could try a harder task.

# Project 9.2 Altoids Arduino

## Purpose

The purpose of this project is to gain experience in making a case and soldering circuits. For this project the circuit that will be soldered is the circuit made in Project 9.1 Breadboard ATmega328P.

## Reference

Project Description: <http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#AltoidsArduino>

ATmega328P Description: <https://mail.rsgc.on.ca/~cdarcy/Datasheets/ATmega328P.pdf>

SN74HC595 IC Data Sheet: <http://mail.rsgc.on.ca/~cdarcy/Datasheets/sn74hc595.pdf>

Voltage Regulator Data Sheet: <https://mail.rsgc.on.ca/~cdarcy/Datasheets/LM7805.pdf>

PCB Reference: <https://www.adafruit.com/product/723>

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| ATmega328P | 1 |
| Volatage Regulator LM7805 | 1 |
| SN74HC595 IC | 1 |
| 10 KΩ Resistor (Watt) | 2 |
| Momentary Tactile Button | 1 |
| 10-LED Blue Bargraph | 1 |

After the previous project has been completed and the circuit has been completed at the breadboard level, the process for soldering a the circuit should begin.

First ensure to use chip seeds for each of the chips that will be used (ATmega328P, 595 shift register). Solder connections between these two chip seeds should be done carefully. Make sure that you have remembered to solder the necessary pins such that normal operation can be achieved. Now, align the bargraph properly such that the cathode leads of the bargraph are put directly into ground. After careful soldering, begin construction on a voltage adapter using the 5V voltage adapter. Wire the outpus of the 5V voltage adapter to high and low on the PCB. Remember to connect the high and low strips of the PCB to each side to ensure power across the board.

Next, to be able to new code into the ATmega328P after it has been placed in the circuit, you will need to add some breakout headers. 2 by 3 right angle breakout headers allow the headers to comeout the side of the Altoids tin and allow use of a spark fun programmer to flash code into the chip. Properly wire and solder, reset, ground, high, MOSI, MISO, and SCK (clock) from the ATmega328P chip seed to these breakout headers. Cut holes in the Altoids tin for the headers, power source, and display to be naturally visble. Finally, flash the previous code then line the tin with electrical tape or use stilts to prevent shorting of your PCB in the Altoids tin.

## Code

// Title  : LED Byte Counter

// Purpose: To create code to show a byte increasing 1 by 1

// Author : William Tessier

// Date   : 2019 11 30

// Status : Working

typedef uint8\_t u8;

typedef int8\_t i8;

u8 clockPin = 13;

u8 dataPin = 11;

u8 latchPin = 10;

u8 outputEnable = 9;

u8 n = 254;

i8 x = -1;                // Rate of counting

uint16\_t delayRate = 15;  // Delay between different numbers

void setup() {

 pinMode(clockPin, OUTPUT);  // Defining optimal pins

 pinMode(dataPin, OUTPUT);

 pinMode(latchPin, OUTPUT);

 pinMode(outputEnable, OUTPUT);

 digitalWrite (outputEnable, LOW);

}

void loop() {

 for (n; 0 < n && n < 255; n += x) { // Increasing by x

   digitalWrite(latchPin, LOW);

   shiftOut(dataPin, clockPin, MSBFIRST, n);

   digitalWrite(latchPin, HIGH);

   delay(delayRate);

 }

 x = -x; // Counts up, then counts down, then back up...

 n = n + x; // Allows repitition of loop

}

## Media

A link to a video on this project: <https://youtu.be/NPoE_yxxY38>

|  |  |
| --- | --- |
| **Altoids Arduino – Binary Counter** | |
|  |  |
|  |  |
| **Altoids Arduino Adafruit Mint-Sized PCB Backside** | |
|  |  |

## Reflection

I believe that this projects difficulty was largely based on the difficulty of the previous project. I myself had very little issues in this project. While I made the obvious mistake of putting, and soldering my LED bargraph in backwards, I was still able to correct this in order to achieve a working product. This projects ‘4 diamond’ rating was proven in the difficulty to solder the outer rim connections for the 2 by 3 headers. Having to solder something into a hole then something else to that in a cramped environment proved to be extremely difficult. The only other issue I encountered was when, in my code, I tried to use a PWM pin to limit the brightness of outputs from the 595 shift register however, I was never able to achieve the desired result of dimming. There is some proof of this present in my code, however there is not that much. Overall, this was an enjoyable, difficult project to attempt, and succeed.

# Project 10. MatrixMadeEZ

## Purpose

The purpose of this project is to gain valuable experience in applying images onto LED bargraphs. With experience gained from learning to scroll messages or numbers, insights can be gained into learning how to replicate ‘Jumbotrons’.

## Reference

Project Description: <http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#MatrixMadeEZ>

SN74HC595 IC Data Sheet: <http://mail.rsgc.on.ca/~cdarcy/Datasheets/sn74hc595.pdf>

TPIC6C595 IC Data Sheet: <http://www.ti.com/lit/ds/symlink/tpic6c595.pdf>

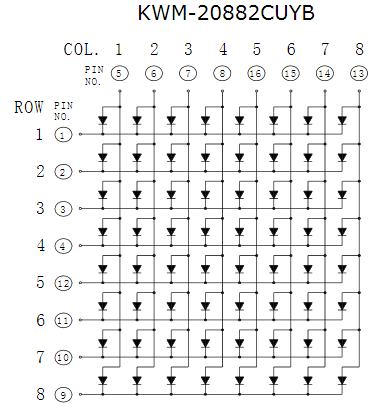
KWM-20882CUYB DataSheet <https://cdn-shop.adafruit.com/datasheets/860datasheet.pdf>

Hugo Reed PCB: <http://darcy.rsgc.on.ca/ACES/PCBs/index.html#MatrixMadeEZ>

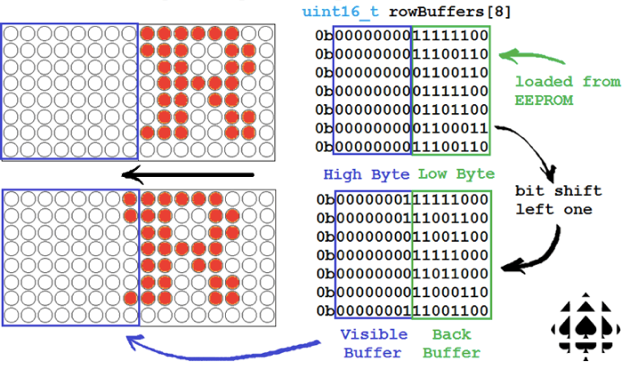
ASCII font written to EEPROM linked below: <https://github.com/rsgcaces/AVRFoundations/blob/master/B_BASIC_SKILLS/EEPROMWriteChars/EEPROMWriteChars.ino>

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| Arduino Uno | 1 |
| LuckyLight 8x8 LED Matrix | 1 |
| SN74HC595 IC | 1 |
| TPIC6C595 IC | 1 |
| Hugo Reed PCB | 1 |

Before starting this project in both hardware or software aspects, collect the required parts listed in the table to the right, then solder the parts into the PCB with proper orientation.

Understand that while the SN74HC595, and TPIC6C595 ICs are both shift registers, they will be used in different ways. To put something to ground with the new, TPIC 595 IC, you will need to output a 1 instead of a 0. This way, if you output two 1s, 1 on each IC, with the SN 595 IC being attached to the anode pin of an LED and the TPIC 595 IC being attached to the cathode pin on the LED, it will turn on. The LuckyLight 8x8 LED Matrix operates as shown to the right. The PCB used in this project connect the 8 outputs of the SN 595 IC to control (x-axis) which columns are on while the TPIC 595 IC will control (y-axis) which rows are on. To display an image on the matrix, a PoV (persistence of vision) effect is needed. This means that we will need to output one row at a time with each row being slightly different allowing, at a high speed, an image to be perceived.

The image that will be displayed will be the ones digit within the Fibonacci sequence. To manage the rows and the image, we write, then pull values out of EEPROM. Listed in the reference is a direct link to a font for each ASCII character. To output an image we will access 8 different numbers in an array and shift one out, then switch to the next row and output the next number in the array. If this loops, then rappid succession of row outputs will appear as a solid image.

To scroll, a slightly different approach is needed. Typically to scroll an image you would just need to shift bits right by one each time, thus showing a moving light. To scroll a message or characters, you would inevitably have the previous image still on the screen as the next image enters the frame. To achieve this, you will need to create a 16 bit array of 8 numbers, output the image created by these numbers, then add on each number in the character font to it’s respective number in the array. After this we need to only show the upper part of the 16 bits (only the high 8 bits) so we will output the array with the highByte(array16bit[x]); (displayed above) function. After 8 shifts to the right, add the next image with the bitwise OR operation and keep scrolling the image.

Finally, to control how fast we scroll, we will inclue the Timer1 library and set it to scroll at every given interval (it is counted in nano seconds). With this new timer, all we have to do in the created function is scroll displayed outputs, add on the next outputs by counting how many time have been scrolled, and run a Fibonacci sequence. To run the sequence create 3 variables and only set one to 1 and the other 2 to 0. Remember to use the modulo operation by 10 everytime such that only the first digit will be shown and scrolled. Then add them, set them, and set them with each other, run this every 8 scrolls and you can continuously load, display, and scroll the Fibonacci sequence.

## Code

// Title  : MatrixMadeEZ

// Author : William Tessier

// Date   : 2019 12 13

#include <**TimerOne**.h>

#include <**EEPROM**.h>

#define SPEED 220000

#define DMLEVEL 200

uint8\_t dimmer = 3; //Defining pins

uint8\_t data = 6;

uint8\_t latch = 4;

uint8\_t clk = 5;

uint8\_t gnd = 2;

uint8\_t high = 7;

uint16\_t rowBuffers[8];

uint8\_t n0 = 0;

uint8\_t n1 = 0;

uint8\_t n2 = 1;

uint8\_t row = 0;

uint8\_t column = 0;

void setup() {

 pinMode (dimmer, OUTPUT);

 pinMode (data, OUTPUT);

 pinMode (clk, OUTPUT);

 pinMode (latch, OUTPUT);

 pinMode (gnd, OUTPUT);

 pinMode (high, OUTPUT);

 digitalWrite (high, HIGH);

 digitalWrite (gnd, LOW);

 analogWrite (dimmer, DMLEVEL);

 Timer1.initialize (SPEED);

 Timer1.attachInterrupt (scroll);

}

void scroll() {

 column++;

 for (uint8\_t i = 0; i < 8; i++) rowBuffers[i] = rowBuffers[i] << 1;

 if (column == 8) {    //Scrolls 1 position each iteration

   column = 0;         //Compound Bitwise OR: |= , adds in the next number

   for(uint8\_t x = 0; x < 8; x++)rowBuffers[x]|=**EEPROM**.read(((n2 + 48) \* 8) + x);

   n0 = n1 + n2; //Fibonacchi sequence

   n0 = n0 % 10; //Allows only 1s digit

   n1 = n2;

   n2 = n0;

 }

}

void loop() {

 shiftOut (data, clk, LSBFIRST, highByte(rowBuffers[row]));

 shiftOut (data, clk, LSBFIRST, 1 << row);

 digitalWrite (latch, LOW);

 digitalWrite (latch, HIGH);

 row++; if (row % 7 == 0) row = 0; //Ternary too slow for PoV

}

## Media

A link to a video on this project: [https://youtu.be/XygU5z6KlYw](https://youtu.be/XygU5z6KlYw" \t "_blank)

|  |  |
| --- | --- |
| **Fibonacci Sequence + MatrixMadeEZ** | |
|  |  |

## Reflection

This was a fun project to do as I was provided with the chance to learn how LED matrices work. It was interesting to learn how to scroll digits and load the previous digit to achieve a scrolling message effect. I decided to use the Fibonacci sequence as my visual display as it was the first mathematical pattern I learned. While this project was difficult conceptually it was satisfying to complete this code with my own two hands instead of relying on other people (excluding my fantastic teacher Mr. Darcy). I hope that my next experience with a matrix will be this enjoyable.

# Project 11. Legacy PCB/Appliance

## Reference

Software Origin: <https://www.autodesk.com/products/eagle/overview>

Project Description: <http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#PCB>

## Inspiration

My inspiration for this project comes from my schools science department. In the grade 9 program, kids learn about water conductivity with sodium particles and without. Currently there are no working water conductancy testers. Surprisingly, the simple device is no longer sold in a semi digital appearance, on or off. Thus, with a clear need, I hope to make a device that is reliable and will work for years of use within the science department.

## Procedure

The circuit that is required is a simple 9V input, that directly connects to an LED. The key to this circuit is merely putting a resistor between the power input and the LED. The resistor that will be used in this circuit will be the water, accessed by 2 breakout pins. While designing it in EAGLE for a better product, carefully apply the dimensions of the 9V and allow breakout headers to be able to reach the water.

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| 9 Volt Battery Clip | 1 |
| 3mm LED | 1 |
| Resistor | 1 |
| Right Angle Headers | 2 |

After receiving the completed EAGLE PCB gather the parts listed to the right to begin assembly. Begin soldering the LED and right angle headers into place. For the battery clip, you may wish to trim the wires to allow for a more comfortable grip. Lastly, while there is a place for an LED to be soldered into the PCB, the resistance should be calibrated to a water source. Tap water commonly has a variety of conductive ions including calcium, magnesium etc… Thus, should the purpose be as a tool in a science room, you may not wish for the test to be positive in normal tap water but only for a saline solution. For tap water, a 0Ω resistor may still be too little current, this is because the water will act as a resistor in itself. As the salinity increases in water the LED my sustain long term damage with an excess of current. At this point having a default resistor becomes more important.

## Media

A short video of a breadboarded version: <https://youtu.be/gCu_gwd4jBg>

A video on the PCB: <https://www.youtube.com/watch?v=tUkCJK3ha-Q&feature=youtu.be>

|  |  |
| --- | --- |
| **EAGLE Schematic + PCB** | |
|  |  |
| **PCB Unsoldered + Soldered** | |
|  |  |
|  |  |

## Reflection

While this project is still incomplete I hope I can expand on my PCB a bit more. Due to finding out about the need for this project rather late in the development phase, my PCB lacks key thoughts about which parts I should use. I added a resistor, besides the water, because with it, while dry fitting I can set it to desired resistence. This way I could make it so that the LED won’t even be on without dissolved sodium in the water. My instant regret is that I prematurely selected my power input as one of the leather tools that you stick on a battery instead of solid iron ones that can hold the battery in place. I hope to add these in in the next iteration of my PCB such that the cost comes down and the design becomes easier to use.

The day we received the PCBs I think everyone was just wondering whether or not theirs worked. I was solely thinking about what would happen if I misplaced or screwed them up… After having finally soldered and completed them, I believe that with some calibration the science department will be able to use them successfully for years to come.

# Project 12. ATtiny85 Binary Game

## Purpose

The purpose of this independent study project is to build a functioning binary game that employs an ATtiny85 microprocessor. With this game, I hope to have as little of a software side as possible and use hardware tools wherever possible.

A binary game, is basically a game in which the objective is to convert a number, ranging from 0 to 255 into its binary equivalent. This is done in the form of ticking 8 bits in a byte to either a high, 1, or a low, 0. The game will display say, a 247, and the player will have to input, on 8 buttons, the respective binary input of in order of the button states high to low, on, on, on, on, off, on, on, on (0b11110111). The binary number system operates in base two, meaning that the last bit in the byte 0b11110111 is the number 1 multiplied by 2^0, 0 being its order. So this byte is actually the sum of: 2^0, 2^1, 2^2, 2^4, 2^5, 2^6, and 2^7 which is 247.

## Reference

4077 Logic Chip Data Sheet: <https://assets.nexperia.com/documents/data-sheet/HEF4077B.pdf>

4081 Logic Chip Data Sheet: <https://mail.rsgc.on.ca/~cdarcy/Datasheets/HEF4081B.pdf>

ATtiny85 Data Sheet: <https://mail.rsgc.on.ca/~cdarcy/Datasheets/attiny85Full.pdf>

SN74HC595 IC Data Sheet: <http://mail.rsgc.on.ca/~cdarcy/Datasheets/sn74hc595.pdf>

Lumex LDT-C512RI: <https://www.lumex.com/spec/LDT-C512RI.pdf>

YSD-439AB4B-35: <https://mail.rsgc.on.ca/~cdarcy/Datasheets/YSD-439AB4B-35.pdf>

Binary Game Description: <http://darcy.rsgc.on.ca/ACES/CoverFlow/BinaryChallenge.html>

## Theory

The primitive theory behind this project is that first, the microprocessor generates a number (0-255), then displays it, then checks if it being inputted. If the input is correct, the microprocessor will increase the score, and repeat the process. For this project, the goal is to reduce from a ATmega328p to an ATtiny85, this means the number of pins used needs to be reduced to 5. To reduce this, we can use 3 pins for shift registers to control the displays using a persistence of vision effect. A dedicated shift register will also output the number to be made. The bits in this byte will then be compared to the inputs with XNOR gates.

|  |  |  |
| --- | --- | --- |
| **XNOR Gate** | | |
| **Inputs** | | **Outputs** |
| X | Y | Z |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

XNOR gates work such that only if the two inputs are the same, the outputs will be high. This way when the XNOR ICs x inputs are connected to the button inputs, and the y inputs are connected to the shift register’s outputs. The outputs of XNOR ICs will all be high if they are the same or in other words, correct. Now, AND gates are used on the outputs of the XNOR gates, such that when the AND gates are hooked up with each other and the XNOR ICs, one final high or low signal will communicate to the microprocessor whether or not the input is correct.

For the last pin available on the ATtiny85, it will be connected to the latch pin of the dedicated shift register so that the shift register can always be outputting the number to be made.

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| ATtiny85 | 1 |
| Rectangular LED | 1 |
| 5V Regulator | 1 |
| 0.1μF Capacitor | 2 |
| 2x3 ISP header pins | 1 |
| Slide Switch | 8 |
| SN74HC595 IC | 3 |
| 4077 Logic IC | 2 |
| 4081 Logic IC | 2 |
| Lumex LDT-C512RI | 1 |
| YSD-439AB4B-35 | 1 |
| Diode | 1 |

To assemble the binary game with an ATtiny85, first aquire the parts in the table to the right. Once acquired, begin by connecting the 5V regulator with the 0.1μF capacitors, then putting putting the converted 5V power through a diode. Once the power input is complete, connect the ATtiny85 to the 2x3 ISP header pins such that it can be programmed later on. Connect 3 pins, data, latch and clock to an initial shift register, and to the ATtiny 85 (these will be later called dataPin, clockPin, and latchPin). Then connect the latch to the other two, use the data carry pin (pin 9) to bridge all the shift registers. Take the last shift register in the bridge and connect a second pin from the ATtiny85 to the latch pin on this shift register, this pin will be later defined as latchPin2 in code. Connect the other pin for latch to the second shift register, then set all the other required shift register pins. Next, wire up the logic ICs. Connect each pin 14 and pin 7 to high and low respectively. Then connect the outputs of the third shift register, in order, to the Y inputs of the XNOR ICs. Connect 8 button’s outer pins to high and low, and connect the middle pin of each one, in order, to the X inputs of the XNOR ICs. Now, each gate of the XNOR ICs will have it’s output read whether or not the button input is the same as the number to be made in the game. Now, connect each output of the XNOR ICs to the inputs of AND ICs, then bungee the outputs of those gates, to inputs. Repeat that process until one output is left. Connect this output to the later defined correctInput pin on the ATtiny85. Lastly, all that is remaining in the hardware aspects is to set up the displays. Take the Lumex display (LDT-C512RI), and connect all the different segment inputs to each other, such that if all digits are enabled via the common cathode set up, and segment A is put to a high, all three A segments will be on. Now connect all segment inputs to a shift register, connect these to the same segment inputs on the other display (YSD-439AB4B-35). Connect all digit controlling pins to the second shift register such that the ATtiny85 can now control what number is outputted, and which digit it is outputted on.

In the code section, begin by defining all pin numbers as previously referred to names. Next, create a score variable and set it to 0. Then create two arrays of size 10 and set up segment maps for number 0-9. On the second array, create the inverse of the first one, this is for the YSD-439AB4B-35 display as it is common anode and not common cathode. In the setup section, after defining pins, a random seed needs to be defined. For this, type: randomSeed(analogRead(0));, now, when the random function is called, the value will be based off an analog read of pin 0. Next, create a function to display the random number on the Lumex display. Label it void as the function will not return anything, and have an 8 bit input for the function. Inside the function, there will be 3 latch cycles from high to low and 6 shift out statements. For each set of two shifts, the first shift will be sent to the second shift register, it needs to shift the proper amount for each digit we want to turn on. This is how a persistence of vision effect will be achieved. The second shift in each cycle will be the segment map of the specific digit of a number. Such that if the number is 247, and the goal is for it to be displayed, when the ones digit will be outputted, we will output the 7th number in the array, the segment map for a 7. Next, create another function to output the score. The input for this void function will be the score. Repeat the same process as the previous function except with 8 total shift cycles and 4 latch low to high cycles. The other difference in this function is that the array called on for the digit segment maps will be the one created for common anode displays. Finally, for the main loop method. Begin by defining a random number variable as random(0, 255);, this is the number that will be made by the user of the game. Then, output the number to be made to the 3rd shift register and set latchPin2 low then high. With this, merely create a while loop that runs while correctInput is not high. Inside the body of the loop, run both functions earlier defined with proper inputs. Then after the loop ends, the correct input must have been achieved, so alter the score and continue the loop.

## Code

// Title  : ATtiny85 Binary Game

// Author : William Tessier

// Date   : 2020 02 23

typedef uint8\_t u8;

u8 clockPin = 2;     //Pin Setup

u8 dataPin = 1;

u8 latchPin = 0;

u8 latchPin2 = 4;

u8 correctInput = 3;

uint16\_t score = 0;  //For any possible score 16 bits are used

u8 digit[] = {    //Segment map for common cathode display

 0b00111111, //0

 0b00000110, //1

 0b01011011, //2

 0b01001111, //3

 0b01100110, //4

 0b01101101, //5

 0b01111101, //6

 0b00000111, //7

 0b01111111, //8

 0b01100111, //9

};

u8 digitScore[] = { //Segment map for common anode display

 0b11000000, //0     An inverse is used directly to avoid

 0b11111001, //1     having to invert bits during functions

 0b10100100, //2

 0b10110000, //3

 0b10011001, //4

 0b01101101, //5

 0b10000010, //6

 0b11111000, //7

 0b10000000, //8

 0b10011000, //9

};

// Title  : ATtiny85 Binary Game

// Author : William Tessier

// Date   : 2020 02 23

typedef uint8\_t u8;

u8 clockPin = 2;     //Pin Setup

u8 dataPin = 1;

u8 latchPin = 0;

u8 latchPin2 = 4;

u8 correctInput = 3;

uint16\_t score = 0;  //For any possible score 16 bits are used

u8 digit[] = {    //Segment map for common cathode display

 0b00111111, //0

 0b00000110, //1

 0b01011011, //2

 0b01001111, //3

 0b01100110, //4

 0b01101101, //5

 0b01111101, //6

 0b00000111, //7

 0b01111111, //8

 0b01100111, //9

};

u8 digitScore[] = { //Segment map for common anode display

 0b11000000, //0     An inverse is used directly to avoid

 0b11111001, //1     having to invert bits during functions

 0b10100100, //2

 0b10110000, //3

 0b10011001, //4

 0b01101101, //5

 0b10000010, //6

 0b11111000, //7

 0b10000000, //8

 0b10011000, //9

};

void setup() {

 randomSeed(analogRead(0));  //Setting the random function

 pinMode (clockPin, OUTPUT); //Defining active pins

 pinMode (dataPin, OUTPUT);

 pinMode (latchPin, OUTPUT);

 pinMode (latchPin2, OUTPUT);

}

void displayRandomNumber(u8 randomNum) {

 digitalWrite (latchPin, LOW); //1s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 6); //1s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digit[randomNum % 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //10s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 10); //10s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digit[((randomNum % 100) - (randomNum % 10)) / 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //100s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 12); //100s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digit[randomNum / 100]);

 digitalWrite (latchPin, HIGH);

}

void displayScore(u8 score) {

 digitalWrite (latchPin, LOW); //1s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 30); //1s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digitScore[score % 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //10s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 46); //10s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digitScore[((score % 100) - (score % 10)) / 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //100s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 78); //1000s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digitScore[((((score % 1000) - (score % 10)) / 10) - (score % 100)) / 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //100s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 142); //1000s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digitScore[score / 1000]);

 digitalWrite (latchPin, HIGH);

}

void setup() {

 randomSeed(analogRead(0));  //Setting the random function

 pinMode (clockPin, OUTPUT); //Defining active pins

 pinMode (dataPin, OUTPUT);

 pinMode (latchPin, OUTPUT);

 pinMode (latchPin2, OUTPUT);

}

void displayRandomNumber(u8 randomNum) {

 digitalWrite (latchPin, LOW); //1s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 6); //1s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digit[randomNum % 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //10s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 10); //10s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digit[((randomNum % 100) - (randomNum % 10)) / 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //100s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 12); //100s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digit[randomNum / 100]);

 digitalWrite (latchPin, HIGH);

}

void displayScore(u8 score) {

 digitalWrite (latchPin, LOW); //1s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 30); //1s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digitScore[score % 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //10s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 46); //10s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digitScore[((score % 100) - (score % 10)) / 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //100s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 78); //1000s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digitScore[((((score % 1000) - (score % 10)) / 10) - (score % 100)) / 10]);

 digitalWrite (latchPin, HIGH);

 digitalWrite (latchPin, LOW); //100s digit

 shiftOut (dataPin, clockPin, MSBFIRST, 142); //1000s digit

 shiftOut (dataPin, clockPin, MSBFIRST, digitScore[score / 1000]);

 digitalWrite (latchPin, HIGH);

}

void loop() {

 uint8\_t randomNum = random(0, 255); //Setting the input number

 digitalWrite (latchPin2, LOW);

 shiftOut (dataPin, clockPin, MSBFIRST, randomNum);

 shiftOut (dataPin, clockPin, MSBFIRST, 14);

// To send the randomNum to be checked

 shiftOut (dataPin, clockPin, MSBFIRST, 0);

// To send the randomNum to be checked

 digitalWrite (latchPin2, HIGH); //Only needs to be done once

 while (!digitalRead(correctInput)) { //Display while number is wrong

   displayRandomNumber (randomNum);

   displayScore  (score);

 }

 score++; //Increase the score

}

void loop() {

 uint8\_t randomNum = random(0, 255); //Setting the input number

 digitalWrite (latchPin2, LOW);

 shiftOut (dataPin, clockPin, MSBFIRST, randomNum);

 shiftOut (dataPin, clockPin, MSBFIRST, 14); // To send the randomNum to be checked

 shiftOut (dataPin, clockPin, MSBFIRST, 0); // To send the randomNum to be checked

 digitalWrite (latchPin2, HIGH); //Outputing the randomNum to the logic ICs only needs to be done once

 while (!digitalRead(correctInput)) { //Display while the number is wrong

   displayRandomNumber (randomNum);

   displayScore  (score);

 }

 score++; //Increase the score

}

## Media

A link to a video on this project: <https://youtu.be/9uBrsnt334Y>

|  |  |
| --- | --- |
| **Breadboard Prototype** | |
|  |  |
| **Final Product** | |
|  |  |
|  |  |

## Reflection

After completing my binary game, I only have the desire to keep going, continue refining until the design is perfect. I truly want to walk away from high school with this perfected product, this game that says “I did this, all by myself, I built this.” After the first iteration, there is still room for improvement, for example, the PCB is slightly janked as there were forgotten connections during the EAGLE process. Other places include using different, more playable switches. Originally I had this idea for a hand held controller and a later iteration of this will have that, a plug in controller that can be designed by it’s users. Looking back I realize that the current board is designed for flexibility, as it can be coded for every type of byte level game. This is a source of inefficiency as typically the more specialized something is, the better it will be at that specific task. Thus, there is still plenty of room to improve. After this project, I can say I loved working on it. While it is still not done, I will continue to improve on it until I can say “It is perfect.”

# Project 13. DC Motor Tachometry

## Purpose

The purpose of this project is to confirm the manufacturer’s statement of at 3V the motor will have about an RpM of 6600. To confirm this we will design a way to count a motors RpM.

## Reference

A link to project description: <http://darcy.rsgc.on.ca/ACES/TEI3M/1920/Tasks.html#tach>

LM358 Datasheet: <https://www.sparkfun.com/datasheets/Components/General/LM358.pdf>

2N3904 Datasheet: <https://mail.rsgc.on.ca/~cdarcy/Datasheets/2N3904.pdf>

Infared set link: <https://abra-electronics.com/opto-illumination/infrared-optics/5mm-infrared-emitter-and-sensor-set-ir-set-5mm.html> (Contains datasheet pdf for emitter and receiver)

DC Hobby Motor datasheet: <https://cdn.sparkfun.com/datasheets/Robotics/M260.pdf>

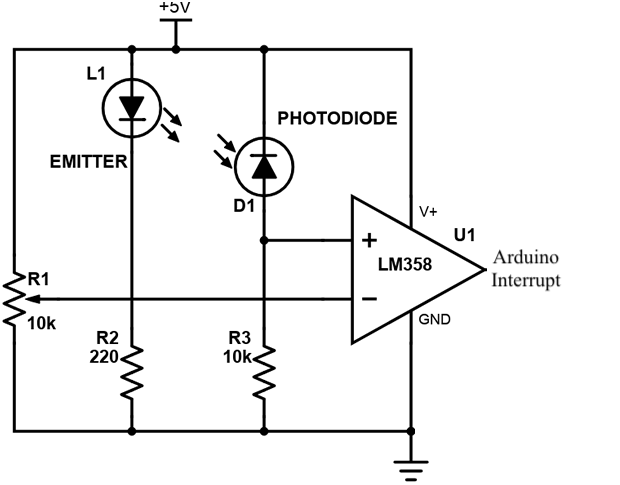
## Theory

To count this motor’s rotations, first a constant indicator that can be measured is required. For this, a propeller will be used. An infared trip wire will be set up and everytime the propeller passes through it, one rotation will have occurred. An Op-Amp circuit comparing resistance of the infared receiver and another resistance will allow a digital response to whether there is a connection or not between the trip wire. The output of the Op-Amp circuit will act as an interrupt for the Arduino which will then be counted.

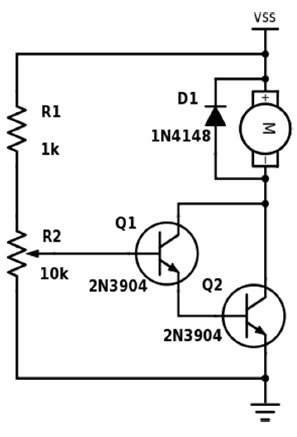
## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| 1kΩ Resistor (Watt) | 1 |
| 10kΩ Potentiometer | 2 |
| Infared Emitter/Receiver | 1 |
| 220Ω Resistor (Watt) | 1 |
| 10kΩ Resistor (Watt) | 1 |
| LM358 Op-Amp | 1 |
| 3904 BJT Transistor | 2 |

To begin this project, acquire all the parts listed in the table to the right including an Arduino UNO and DC Hobby Motor.

Begin by wiring the Op-Amp circuit that is shown below the parts table. This circuit compares a resistance dialed in by the 10kΩ potentiometer and the resistance given by the infared receiver. The infared receiver acts as a photodiode in how it outputs resistance dependant on how much infared light it gets. By comparing it with the a resistor, a sensitivity to interrupt the trip wire can be set. This helps with the overall output being either a high or low. To test this circuit, replace the Arduino interrupt pin on the LM358 with an LED.

When wiring up the LM358 connect its pin 4 and 8 to low and high respectively. Pin 1 will act as the output to the Arduino, with pins 2, and 3 acting as the potentiometer output and infared receiver output respectively.

Next, wire up Darlington pair circuit shown to the right, this will act as the motor control circuit. The Darlington pair are the two cascaded transistors. With both Q1 and Q2 transitors acting as amplifiers, Q1 amplifies the potentiometer signal and is then further amplified by the Q2 transistor to act as the input voltage for the motor. Adding a diode (D1) is recommended for this circuit. The diode acts to protect the transitors from the magnetic field releasing from the Hobby Motor.

Lastly, before beginning the coding phase, set up the trip wire so the equivalent of a propeller can pass through the trip wire and trigger a signal. Callibrate this by adjusting the pot and having an LED come out of pin 1 of the LM358. Once these steps are complete connect pin 1 of the LM358 to pin 2 of the Arduino UNO.

Now for the code, begin by enabling the library TimerOne. Next, create a variable for the rotations, and one for the input pin. In setup, enable the serial monitor and then set the input pin as an INPUT\_PULLUP so that the input is automatically in pullup resister configuration. Following, define the interrupt: attachInterrupt(), with the statement set to increase the rotations by one. This way each time the trip wire is blocked, a high signal is outputted and counted by the Arduino. Finally, set Timer1 to initialize ever second and call on a function that prints rotation count multiplied by 60 (this makes it RpM instead of RpS) and then resets the rotation count back to 0.

## Code

// Purpose: To count the RpM of a motor using an infared sensor

// Author : William Tessier

// Date   : 2020 04 05

// Status : Working

#include<**TimerOne**.h>

uint8\_t externalInteruptPin = 2; //Pin 1 of LM358

uint32\_t rotations = 0;

void setup() {

**Serial**.begin(9600)

 while(!**Serial**);

 pinMode(externalInteruptPin, INPUT\_PULLUP); //Defining pin

 attachInterrupt(digitalPinToInterrupt(externalInteruptPin), increase, FALLING);

 Timer1.initialize(1000000);   //Setting up timer to act every second

 Timer1.attachInterrupt(reset);

}

void reset() {

**Serial**.println("RpM: " + String(60\*rotations)); //For RpM

 rotations &= 0; //Resets to 0

}

void increase() {

 rotations++; //Singular increase per rotation

}

void loop() {

}

## Media

A link to a video on this project: <https://www.youtube.com/watch?v=O1gs1FVBwsA&feature=youtu.be>

|  |  |
| --- | --- |
| **Motor propeller + Infared trip-wire** | |
|  |  |

|  |  |
| --- | --- |
| **Darlington pair control circuit + Infared trip-wire circuit** | |
|  |  |

## Results

With 9V run through a Darlington pair, RpM of as low as 1,500 was measured. When the pot was turned all the way in the other direction, an RpM barely greater than 18,000 was measured. When the pot was slightly more than half, an RpM of close to 6,600 is achieved. The video in the media section features just the motor being manipulated and shows the change to RpM.

If the motor is placed without a propeller, a frequency close to 440 Hz (an “A”) can be audibly perceived. This may indicate that the Hobby Motor is capable of exceeding 25,000 when no resistance or load is added with 9V input.

## Reflection

I believe this project to be a success. If I had access to some voltage adapters I would have ran 5V and 3V through the motor to better test RpM. For me, this project has been the most frustrating and one of the most difficult projects to do. If you know me well, you know I hate doing work at home so I stay late in the DES working until my project is built and secure. For this project, that was not an option. While building it, I encountered many challenges such as holstering the break-beam for the infared sensing, working my motor (a transistor stopped working at one point, and there is no multimeter at home), and even obtaining a diode as I could only find one. None the less I did experience the satisfaction of obtaining results I believe to be accurate. I hope now on, when I have projects to do, I have a dedicated space where I can work.

# Project 14. Time-Sensitive Mechanics

## Purpose

The purpose of this project is to create a system that will create an alert every hour on a given minute. The alarm is in the form of a stepper motor. This project can be slightly adjusted to perform tasks such as opening blinds for a window, turning a light on or off, or the alarm.

## Reference

SN74HC595 IC Data Sheet: <http://mail.rsgc.on.ca/~cdarcy/Datasheets/sn74hc595.pdf>

DS3231 RTC Data Sheet: <https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>

Seven Segment Data Sheet: <http://mail.rsgc.on.ca/~cdarcy/Datasheets/7SegmentDisplay.pdf>

Motor and Driver link: <https://abra-electronics.com/electromechanical/motors/stepper-motors/mot-28byj48-stepper-motor-w-uln2003-driver.html>

RTC Library Used Title, “DS3231”, Author: Andrew Wickert.

## Stepper Motors

A stepper motor, like other motors uses magnetic fields created by coils to create a turning motion on a shaft.

Observe the image on the right. The “N”s and “S”s represent north and south poles on a magnet. Each of the coloured blocks surrounding the gear (shaft) and magnet ring represent controlled coils which when a current runs through them, they attract the northern pole on a magnet, the “N”.

When the Arduino runs current through blue, N shifts to point to it, when the Arduino runs current through purple and not blue, N shifts to point to it. Now by rapidly changing which coils receive current, you can spin the shaft to achieve a precise step each time.

Observe this video, created by Ben Strateham, for a detailed explanation of different states and the body of the specific motor used in this project.

Stateham’s video: <https://www.youtube.com/watch?v=B86nqDRskVU>.

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| DS3231 RTC | 1 |
| CC Seven Segment Display | 1 |
| Stepper Motor | 1 |
| ULN2003 Driver | 1 |
| Arduino UNO | 1 |
| SN74HC595 IC | 1 |

Before beginning assembly on this project, aquire the parts indicated on the right.

Begin by setting up the shift register. Connect outputs to the seven segment display, remember to ground the display as it is common cathode. Next, plug in the stepper motor to the ULN2003 driver and connect that to pins 8 to 11. These can be changed however these pins are PORTB 0 to 3 respectively. Access to the lower nibble of this port allows for more elegant code when port manipulation is used to drive the motor. Take an external power source which will act to drive the motor. This can range from 5V to 12V. Finally, connect this to the ULN2003 driver and connect grounds with the Arduino. Plug in the DS3231 RTC with it having mutual ground, and it’s VCC taken from Arduino’s 5V. Putting a battery in the RTC allows the time to not be reset should power be cut for some reason. Attach the SDA and SCL pins to A4 and A5 on the Arduino. These are the pins required for I2C communication.

For this RTC it is recommended to use the RTC library referenced in the code as it is the one used for control of the RTC in this project. Use the library’s examples to set up the clock.

For the code, begin by including the DS3231 and Wire librarys. Then write a segment map that will be used later for the display. Create bool variables h12 and PM for the clock. These are indicated as necessary for certain functions with the library. Create variables for the shift register. Initiate the clock by typing **DS3231** Clock and initiate Wire with Wire.begin(). Set the pins used by the shift register to OUTPUT. Next, using register level code, set only the 4 pins being used for the motor to 1 (output). Set the port being used to have only one of the pins used for the motor turned on. Create a function called spin. In this function the pin that is currently high will be shifted over, this allows for a single change in the shaft position of the stepper motor. Now, or the port used with its upper byte. This way, when the 4th bit is shifted into the fifth position (0B00001000 🡪 0B00010000) instead of the motor shutting off, it will continue (0B0001 | 0B00010000 = 0B00010001). Next, create a coast function in which the port used essentially shuts off. And the lower nibble with zeros and the upper nibble with ones, this way the motor can continue with the spin variable with no trouble. Should a coast function not be used, the motor will heat up however, it will keep its position.

In the main loop, create an if statement with how often you want your alarm and which number it is. This means if you select minutes and the number compared to is 20, every hour on the 20th minute, the statement within the if statement will be executed. Following the if statement create a while loops that runs nothing until the current minute will pass. This ensures no action is held for the entire specified time frame. To achieve the alarm, simply have the stepper spin();for an amount of time while the segments count down how long the motor will be on for. When the time desired for the motor to be on for ends, coast()so that the motor does not continue to gain heat.

## Code

// Title  : Hourly Alarm

// Author : William Tessier

// Date   : 2020 05 09

#include <**DS3231**.h>

#include <Wire.h>

#define TIME\_OF\_ALARM 36

#define DURATION 10

#define SINGLE\_STEP 0B00010001 //Simple, less torque

#define DOUBLE\_STEP 0B00110011 //More torque

#define SPEED 2

//segment map excluded

uint8\_t data = 3; uint8\_t latch = 4; uint8\_t clk = 5;

bool h12; bool PM;

**DS3231** Clock;

void setup() {

 Wire.begin();

 pinMode(data, OUTPUT);

 pinMode(latch, OUTPUT);

 pinMode(clk, OUTPUT);

 DDRB  = 0B00001111;

 PORTB = SINGLE\_STEP;

coast();

}

void spin() {

 PORTB  = PORTB << 1;   //Pushes Rotor

 PORTB |= (PORTB >> 4); //ORs high nibble

}

void coast() { //Prevents heating of stepperM

 PORTB &= 0B11110000; //Clear low nibble

}

void loop() {

 if(Clock.getMinute() == TIME\_OF\_ALARM) {

        //Changeable: Clock.getHour(h12, PM) or Clock.getSecond()

   uint8\_t secondTime = Clock.getSecond();

   while(Clock.getSecond() < (secondTime + DURATION)) {

     //Perform action for DURATION

     for(int8\_t y = 9; y > -1; y--) {

       digitalWrite(latch, LOW); //Output y

       shiftOut(data, clk, LSBFIRST, number[y]);

       digitalWrite(latch, HIGH);

       for(uint16\_t x = 0; x < (100\*DURATION/SPEED); x++) {

         spin(); delay(SPEED); //Delay influences angular velocity

   } } }

   digitalWrite(latch, LOW);

   shiftOut(data, clk, LSBFIRST, 0); //turn off display

   digitalWrite(latch, HIGH);

   coast(); //No Power to motor so it doesn't over heat

   while(Clock.getMinute() == TIME\_OF\_ALARM); // Wait till next hour

 }

}

## Media

A link to a video on this project: <https://youtu.be/UKhnAj2NYwo>

|  |  |
| --- | --- |
| **RTC + ULN2003 Configuration** | |
|  |  |
|  | |
|  | |

## Reflection

If you know me, you’ll know I am not a fan of open ended projects with specified tools. This makes me think that there is something specific in mind that could be far superior to what I am doing. Anyways, I’m glad I got to showcase some low level code on something practical like a stepper motor. If I build one myself I would hope to be able to handle a higher RPM and also adjust which way the current flows through the coils allowing a push and pull effect on the magnets inside. Creating the display for this project was frustrating. At first I was going to count seconds between minutes using a shift register and a 4511 BCD to DEC format converter, unfortunately, I did not have this part. Then I decided I would write segment maps and use two shift registers. After wiring everything and testing everything, I realized 2 things; 1. my second shift register was not working, 2. I did not have anymore shift registers. Thank you Covid. Overall this was a satisfying project to complete, and seemed like a fitting finale for this year.

# Project 15. Game Buzzers

## Purpose

The purpose of this project is to create game buzzers. This buzzer system could be used for anything involving answering questions and needing a way to determine who clicks first.

## Reference

CMOS 4071 IC Datasheet: <https://www.ti.com/lit/ds/symlink/cd4071b.pdf?ts=1590799607336>

## Theory

For this project 4 buttons will be used as individual buzzers. They can be extended with basic wires. The buzzers act in a pull down format, meaning that they are naturally grounded. When one buzzer is pressed, the Arduino UNO has to be alerted to check the button states. Thus, OR logic gates are used to check. All the outputs of the buttons are run through hardware based OR gates. This way if any of outputs of the buttons are high, the cascaded OR gates will output a single high. If all of the buttons are low, then the cascaded OR gates will output a low. With more advanced software, one can apply interrupt coding to the system.

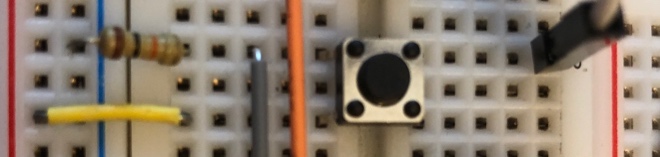
As delays are not applicable with interrupt coding, the only way to prolong an LEDs on state before turning off would be to either have a reset button or constantly monitor the interrupt pins state in a loop instead of interrupt coding. The CHANGE mode used with interrupts does not function properly to be able to activate the ISR (interrupt service routine) when the interrupt pin rises and falls.

In the loop body of the code, the state of the interrupt pin is constantly checked such that when it is high, an action may be performed. Once it is set high (a button has been clicked) the buzzer is turned on. The code then checks which of the buttons have been pressed and echoes that onto the corresponding LED. Then a slight delay with the buzzer and LED on, this way the display is more visually pleasing. Following the delay, until the contestant releases their buzzer, the LEDs will be on (buzzer off). This cycle can be repeated indefinetly throughout a game.

## Procedure

|  |  |
| --- | --- |
| **Parts Used** | **Quantity** |
| PBNO | 4 |
| 10kΩ Resistor (Watt) | 4 |
| LED | 4 |
| CMOS 4071 IC | 1 |
| Arduino UNO | 1 |
| Buzzer | 1 |

To begin this project, first gather all the parts listed in the table to the right.

Begin by wiring 4 buttons in a pull down resistor format (shown below the parts table). This means that a resisted ground is connected to one side, and the other is connected to high. Connect each button with a wire connected to the Arduino. Be sure to have the button to Arduino wires take consecutive pin on the Arduino.

Next, begin wiring up the 4071 OR gate IC. This IC is a quad-2 input logic chip, meaning it contains for separate OR gates inside. Using the reference notch as reference, power goes on pin 14 with ground on pin 7. Get power for this circuit from the Arduino UNO 5V supply. Connect all the buttons to different OR gate inputs. Pins 1 and 2, 5 and 6, 8 and 9, and 12 and 14, are the different OR gate inputs with the outputs being right next to the inner most pins. Connect the outputs of the OR gates that have been used to the inputs of another gate. Now that the 4071 OR gate IC has been cascaded, if any of the inputs are high, regardless of the others, the output will be high. This way, when any button is pressed, the Arduino will receive the equivalent of an interrupt signal.

Take 4 wires from the beginning of PORTC on the Arduino UNO (A0 to A3) and connect them to anode leads of 4 LEDs. Next, run the cathode ends of the LEDs to ground. Finally, connect a wire from an Arduino pin to the anode lead of a buzzer, and connect the cathode lead to ground.

In the code, begin by creating a variable for the pin that the Arduino from the 4071 IC. The variable does not need to be declared in setup as it is an input. Next, in the loop, simulate an interrupt by having an if statement constantly check if the 4071 IC input is high. When it is, turn the buzzer on, read each pin input from the buttons, and then echo the state onto LEDs. This can be accomplished with low level code by writing in a for loop:

PORTC |= (digitalRead(x + 8) << x); where the ‘8’ is the lowest pin that the buttons are plugged into. Set a delay to act immediately following, this way; a first button pressed will be the only one that stays on for (at least) the length of the delay. After the delay, turn off the buzzer, then create a while loop that performs nothing while a button is pressed. Finally, clear the port used, this turns everything off.

## Code

// Title  : Buzzer System

// Author : William Tessier

// Date   : 2020 05 27

#define DELAY\_TIME 0.5 //0.5 Seconds

uint8\_t clickedPin = 2; //For the outer trigger

void setup() {  //Input is default (PORTB is input)

 DDRC = 0b00011111;  //Establishing pins used as outputs

}

void loop() {

 if(digitalRead(clickedPin)) { //When a button is clicked...

   PORTC = 0b00010000; //turns buzzer on

   for(uint8\_t x = 0; x < 4; x++) PORTC |= (digitalRead(x + 8) << x);

   //8 is the digital pin of PORTC 0,   echo buttons on to the 4 LEDs

   delay(DELAY\_TIME\*1000); //Wait for a bit...

   PORTC &= 0b00001111; //Turns buzzer off, keeps LED on

   while(digitalRead(clickedPin)); //Wait with LED on...

   PORTC = 0; //turns LED off

 }

}

## Media

A link to a video on this project: <https://www.youtube.com/watch?v=YzNbfMyTq70>

|  |  |
| --- | --- |
| **Game Buzzer System** | |
|  |  |
|  |  |
|  |  |

## Reflection

I have not been to school since early March. With no school, grades that cannot change, and excess work surrounding me; my motivation has sunk. Prior to March, I was looking forward to this project, specifically to working with wireless technology. Unfortunately, when one has to make a large project, they need to test many parts individually, this means ordering parts (I have no experience with) online. After selecting parts to test, I would have to then go and look for more efficient parts, do design work, and more just to finish this project. So with low motivation, when I thought: “oh, if I’m not doing it wirelessly, I can just use some logic and build it.” This is not a good way to approach a learning environment, yet, in these times, it is the simplest approach. I cannot say: I enjoyed this project, or found it difficult, or that it was interesting, all I can say is that I am finished.

Ideas for grade 11

1. PCB for Streetlight using 4017 IC and a 3 color LED, need 10 diodes plus improvised NGO
2. TAZER
3. PCB for scoreboard (ISP)
4. PCB for stopwatch using NGO + 4510 IC, for stoppage, connect outputs to inputs then preset button

FINAL G12 ISP IDEA

BULLSEYE DARTBOARD