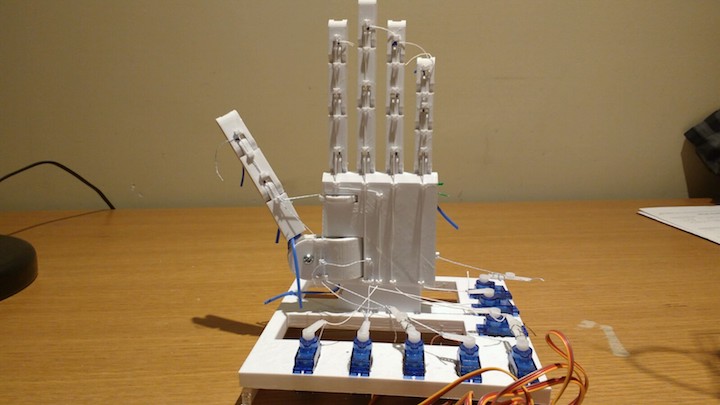
Design Engineering Report



Course: Computer Engineering Technology

Code: TEL3M, TEI3M, ICS4U

Instructor: C. D’Arcy

Author: James Corley

Date: Thursday, December 31, 2020

Table of Contents

[Project 1. LED Brightness 2](#_Toc10064799)

[Purpose 2](#_Toc10064800)

[Reference 2](#_Toc10064801)

[Theory 2](#_Toc10064802)

[Procedure 2](#_Toc10064803)

[Media 3](#_Toc10064804)

[Reflection 4](#_Toc10064805)

[Project 2. The Automatic Night Light 6](#_Toc10064806)

[Purpose 6](#_Toc10064807)

[Reference 6](#_Toc10064808)

[Theory 6](#_Toc10064809)

[Procedure 7](#_Toc10064810)

[Media 8](#_Toc10064811)

[Reflection 9](#_Toc10064812)

[Project 3. The 3D Christmas Tree 10](#_Toc10064813)

[Purpose 10](#_Toc10064814)

[Reference 10](#_Toc10064815)

[Theory 10](#_Toc10064816)

[Procedure 10](#_Toc10064817)

[Media 11](#_Toc10064818)

[Reflection 12](#_Toc10064819)

[Project 4. Digital (Transistor-Based) Logic Gates 14](#_Toc10064820)

[Project 5. 12 Hour Clock 16](#_Toc10064821)

[Purpose 16](#_Toc10064822)

[Reference 16](#_Toc10064823)

[Theory 16](#_Toc10064824)

[Procedure 17](#_Toc10064825)

[Failures 18](#_Toc10064826)

[Media 19](#_Toc10064827)

[Reflection 21](#_Toc10064828)

[TEI3M-Arduino 22](#_Toc10064829)

[Project 6. Traffic Light Assembly and Testing 24](#_Toc10064830)

[Purpose 24](#_Toc10064831)

[Reference 24](#_Toc10064832)

[Procedure 24](#_Toc10064833)

[Code 25](#_Toc10064834)

[Media 25](#_Toc10064835)

[Reflection 26](#_Toc10064836)

[Project 7. Digital and Analog: Reading and Writing 28](#_Toc10064837)

[Purpose 28](#_Toc10064838)

[Reference 28](#_Toc10064839)

[Theory 28](#_Toc10064840)

[Software 28](#_Toc10064841)

[Hardware 28](#_Toc10064842)

[Procedure 29](#_Toc10064843)

[Code 30](#_Toc10064844)

[Media 32](#_Toc10064845)

[Reflection 32](#_Toc10064846)

[Project 8. Design Challenge 1 34](#_Toc10064847)

[Purpose 34](#_Toc10064848)

[Reference 34](#_Toc10064849)

[Acknowledgments 34](#_Toc10064850)

[Procedure 34](#_Toc10064851)

[Evolution 35](#_Toc10064852)

[Media 36](#_Toc10064853)

[Reflection 37](#_Toc10064854)

[Project 9. POV: Dual Seven Segment Display Application 38](#_Toc10064855)

[Purpose 38](#_Toc10064856)

[Reference 38](#_Toc10064857)

[Theory 38](#_Toc10064858)

[Procedure 39](#_Toc10064859)

[Code 40](#_Toc10064860)

[Media 41](#_Toc10064861)

[Reflection 41](#_Toc10064862)

[Project 10. DIY Bicolor LED Display 42](#_Toc10064863)

[Purpose 42](#_Toc10064864)

[Reference 42](#_Toc10064865)

[Theory 42](#_Toc10064866)

[Procedure 43](#_Toc10064867)

[Code 44](#_Toc10064868)

[Theme 46](#_Toc10064869)

[Media 47](#_Toc10064870)

[Reflection 47](#_Toc10064871)

[Project 11. Frequency Spectrum Analyzer 48](#_Toc10064872)

[Purpose 48](#_Toc10064873)

[Reference 48](#_Toc10064874)

[Theory 48](#_Toc10064875)

[Procedure 50](#_Toc10064876)

[Code 51](#_Toc10064877)

[Media 52](#_Toc10064878)

[Reflection 52](#_Toc10064879)

[Project 12. Robotic Hand 54](#_Toc10064880)

[Purpose 54](#_Toc10064881)

[Reference 54](#_Toc10064882)

[Acknowledgments 54](#_Toc10064883)

[Theory 54](#_Toc10064884)

[Procedure 56](#_Toc10064885)

[Issues 58](#_Toc10064886)

[Code 60](#_Toc10064887)

[Media 67](#_Toc10064888)

[Reflection 70](#_Toc10064889)

[Project 13. Printed Circuit Board: ATtiny84 Breakout Board 72](#_Toc10064890)

[Purpose 72](#_Toc10064891)

[Reference 72](#_Toc10064892)

[Theory 72](#_Toc10064893)

[Procedure 73](#_Toc10064894)

[Media 74](#_Toc10064895)

[Reflection 75](#_Toc10064896)

[Project 14. I2C Data Logger 76](#_Toc10064897)

[Purpose 76](#_Toc10064898)

[Reference 76](#_Toc10064899)

[Theory 76](#_Toc10064900)

[Procedure 78](#_Toc10064901)

[Code 79](#_Toc10064902)

[Media 83](#_Toc10064903)

[Reflection 84](#_Toc10064904)

[Project 15. Textwriter 86](#_Toc10064905)

[Purpose 86](#_Toc10064906)

[Reference 86](#_Toc10064907)

[Acknowledgments 86](#_Toc10064908)

[Theory 86](#_Toc10064909)

[Procedure 88](#_Toc10064910)

[Important Functions 90](#_Toc10064911)

[Variables and Arrays 91](#_Toc10064912)

[Code 92](#_Toc10064913)

[Media 103](#_Toc10064914)

[Reflection 105](#_Toc10064915)

[ICS4U-AVR Optimization 106](#_Toc10064916)

[Project 16. CHUMP 108](#_Toc10064917)

[Code 108](#_Toc10064918)

[Purpose 108](#_Toc10064919)

[Reference 108](#_Toc10064920)

[Theory 108](#_Toc10064921)

[Code 109](#_Toc10064922)

[Reflection 109](#_Toc10064923)

[Clock 110](#_Toc10064924)

[Purpose 110](#_Toc10064925)

[Reference 110](#_Toc10064926)

[Theory 110](#_Toc10064927)

[Procedure 112](#_Toc10064928)

[Media 113](#_Toc10064929)

[Reflection 113](#_Toc10064930)

[Arithmetic and Logic Unit 114](#_Toc10064931)

[Purpose 114](#_Toc10064932)

[Reference 114](#_Toc10064933)

[Theory 114](#_Toc10064934)

[Procedure 116](#_Toc10064935)

[Media 116](#_Toc10064936)

[Reflection 116](#_Toc10064937)

[Processor 118](#_Toc10064938)

[Purpose 118](#_Toc10064939)

[Reference 118](#_Toc10064940)

[Theory 118](#_Toc10064941)

[Procedure 119](#_Toc10064942)

[Media 120](#_Toc10064943)

[Reflection 121](#_Toc10064944)

[Completed Processor 122](#_Toc10064945)

[Purpose 122](#_Toc10064946)

[Reference 122](#_Toc10064947)

[Theory 122](#_Toc10064948)

[Procedure 124](#_Toc10064949)

[Issues 126](#_Toc10064950)

[Media 126](#_Toc10064951)

[Reflection 127](#_Toc10064952)

[Project 17. Elevator 128](#_Toc10064953)

[Purpose 128](#_Toc10064954)

[Reference 128](#_Toc10064955)

[Acknowledgments 128](#_Toc10064956)

[Theory 128](#_Toc10064957)

[Hardware 128](#_Toc10064958)

[Software 130](#_Toc10064959)

[Procedure 130](#_Toc10064960)

[Code 132](#_Toc10064961)

[Media 135](#_Toc10064962)

[Reflection 135](#_Toc10064963)

[Project 18. Telephone Keypad 136](#_Toc10064964)

[Purpose 136](#_Toc10064965)

[Reference 136](#_Toc10064966)

[Theory 136](#_Toc10064967)

[Procedure 138](#_Toc10064968)

[Issues 138](#_Toc10064969)

[Code 139](#_Toc10064970)

[Media 143](#_Toc10064971)

[Reflection 143](#_Toc10064972)

[Project 19. Rotary BCD Switch 144](#_Toc10064973)

[Purpose 144](#_Toc10064974)

[Reference 144](#_Toc10064975)

[Theory 144](#_Toc10064976)

[Procedure 145](#_Toc10064977)

[Code 146](#_Toc10064978)

[Media 147](#_Toc10064979)

[Reflection 147](#_Toc10064980)

[Project 20. Knight Rider 148](#_Toc10064981)

[Purpose 148](#_Toc10064982)

[Reference 148](#_Toc10064983)

[Theory 148](#_Toc10064984)

[Procedure 149](#_Toc10064985)

[Code 150](#_Toc10064986)

[Media 152](#_Toc10064987)

[Reflection 152](#_Toc10064988)

[Project 21. The 4-Wire DC Fan 154](#_Toc10064989)

[Part A. 154](#_Toc10064990)

[Purpose 154](#_Toc10064991)

[Reference 154](#_Toc10064992)

[Theory 154](#_Toc10064993)

[Media 156](#_Toc10064994)

[Reflection 156](#_Toc10064995)

[Project 22. ACEQuest 2.0 158](#_Toc10064996)

[Purpose 158](#_Toc10064997)

[Reference 158](#_Toc10064998)

[Theory 158](#_Toc10064999)

[Daisy-chaining 158](#_Toc10065000)

[Speed in Arduino 158](#_Toc10065001)

[Timer Interrupts 159](#_Toc10065002)

[Procedure 160](#_Toc10065003)

[Hardware Function 160](#_Toc10065004)

[Game Function 160](#_Toc10065005)

[Code Function 161](#_Toc10065006)

[Code 163](#_Toc10065007)

[Media 170](#_Toc10065008)

[Reflection 171](#_Toc10065009)

[Project 23. Flex Page 172](#_Toc10065010)

[Purpose 172](#_Toc10065011)

[Reference 172](#_Toc10065012)

[Theory 172](#_Toc10065013)

[PCB Part Mounting 172](#_Toc10065014)

[Surfacemount(SMD) 172](#_Toc10065015)

[Soldering Surfacemount Parts 172](#_Toc10065016)

[Pin Change Interrupts 173](#_Toc10065017)

[Procedure 173](#_Toc10065018)

[Code 174](#_Toc10065019)

[Media 176](#_Toc10065020)

[Reflection 177](#_Toc10065021)

[Project 24. Long ISP 178](#_Toc10065022)

[Purpose 178](#_Toc10065023)

[Reference 178](#_Toc10065024)

[Theory 178](#_Toc10065025)

[Procedure 178](#_Toc10065026)

[Programmer 178](#_Toc10065027)

[Case 179](#_Toc10065028)

[Adding a command to the CHUMP 180](#_Toc10065029)

[Media 181](#_Toc10065030)

[Reflection 182](#_Toc10065031)

# Project 1. LED Brightness

## Purpose

To determine the effect of a potentiometer as a variable resistor on a Light Emitting Diode (LED).

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

1.7 LED Brightness Project Description

[http://darcy.rsgc.on.ca/ACES/TEL3M/1617/TasksFall.html - LEDBrightness](http://darcy.rsgc.on.ca/ACES/TEL3M/1617/TasksFall.html#LEDBrightness)

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

## Theory

A potentiometer increases and decreases resistance depending on the location of a windshield wiper like arm. If the arm is turned far away from the leg that has current flowing in, then it has high resistance because the current has to flow a long way through resistant material. As the arm is turned closer to the leg with current entering it, the electrons do not have to fight through the material for as long, decreasing the resistance. As the potentiometer’s resistance is increased the LED will dim because the electrons wastes more power getting through the potentiometer, therefore it has less left over to be used in the LED. As the resistance decreases, the electrons have to use less power to get past the potentiometer, and have more to be used at the LED, making it shine brighter.

## Procedure

Path of the current through Circuit 1.7.

|  |
| --- |
| Parts List |
| Yellow LED |
| 470Ω Resistor |
| Potentiometer |
| Power Diode |
| 9V Battery |
| Connecting wire |
| 9V to Barrel Jack Adaptor |
| DC Barrel Jack Adaptor(breadboard compatible) |

First the current flows out of the 9V battery onto the power bar of the breadboard. A diode to protect the current from flowing backwards, is connected to the positive power bar, directing the current to a 470Ω resistor which is large enough to protect the LED from burning out. From there the current flows into the A leg of the potentiometer losing a variable amount of power, then exiting through the C leg into the LED. This LED is then emits light depending on the remaining power, and the current exits into the negative channel of the power bar. A wire connecting the two negative power bars return the current to the anode on the power supply.

Questions and Answers

1. Explain the circuit above? *This circuit uses a potentiometer as a variable resistor to change the brightness of an LED.*

2. What is the purpose of R1?

*The purpose of R1(470Ω resistor) is to have a base amount of resistance to protect the LED. If it was not there and the potentiometer’s resistance was lowered to much then the LED could burn out.*

1. Prototype the circuit on your breadboard and adjust it. Did the LED ever go completely out? Should it have?

*No, the LED never went out completely, the resistance was not high enough to eliminate all power in the circuit, even at the potentiometers highest resistance.*

## Media

|  |  |
| --- | --- |
| 1.7 Circuit Schematic(Fritzing) | 1.7 Circuit Breadboard Diagram(Fritzing) |
|  |  |
| Link to YouTube Video  <https://www.youtube.com/watch?v=celPGEG3qWY> |  |
|  | 1.7 Circuit Built on a Breadboard |

## 

## Reflection

In conclusion, turning the potentiometers wheel changes its resistance, through adjusting the amount of resistant material the current must go through. In turn this changes the brightness of the LED by altering power reaching it.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Location of Wheel (with current entering on A leg) | Potentiometer Resistance | Power Used in Potentiometer | Power Left Over to be Used in LED | Brightness of LED |
| Turned to the right | High | High | Low | Dim |
| Turned to the left | Low | Low | High | Bright |

# Project 2. The Automatic Night Light

## Purpose

To apply and observe the effects of a transistor, a light dependent resistor, and a potentiometer to create an Automatic Night Light.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

The Automatic Night Light Project Description

<http://darcy.rsgc.on.ca/ACES/TEL3M/1617/TasksFall.html>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

## Theory

|  |
| --- |
|  |
| Transistor (NPN) |

In Project 2. The Automatic Night Light two variable resistors (light dependent resistor and a potentiometer) and a transistor are used to create a night light. The light dependent resistor, changes its resistance according to the amount of ambient light hitting it. As more light hits the resistor the resistance will decrease and as less hits it the resistance increases. The transistor is more complicated, it is essentially a switch that is activated by electrical current. Because of this it has no moving parts and cannot be activated manually. The only way to turn the switch on or off is by putting current through the Base(B) leg. Depending on the amount of current and voltage hitting that leg that will open up the Collector(C) leg to send current through to the Emitter(E) leg.

In building this, two new tools were used, solder and hot glue. Though the hot glue is relatively

|  |
| --- |
|  |
| Basic Soldering |

simple, the solder required the application of a hot pen and several other tools. Essentially, to solder the hot pen must be applied to the joint to heat the area, then the solder in wire form can be applied. The solder will melt and when the hot pen is removed it will harden forming a joint.

## Procedure

|  |
| --- |
| Parts List |
| 2 Yellow LEDs |
| Potentiometer |
| 470Ω Resistor |
| Light Dependent Resistor |
| 22kΩ Resistor |
| NPN Transistor |
| Small Storage Case |
| Hot glue |
| Solder |
| DC to AC Adaptor |
| Switch |
| 11 Connecting Wires |
| Printed Circuit Board (PCB) |

Path of the current through Circuit 4.1  
To begin with there is a switch which decides whether or not the current will flow. After this (if the switch is closed) the current will flow through a potentiometer and then to a Light Dependent Resistor(LDR) and to ground. Depending on the resistance of this LDR the current may decide to flow into a 22k Ω Resistor (the current travels the path of least resistance) this resistor leads to the B leg of a transistor. Once the B leg is activated the C and E legs connect and that opens up current to travel through them. So instead of traveling to the potentiometer alone it will split in parallel and travel to the transistor. So now current can flow through the transistor into a 470Ω Resistor (to protect the LED’s) and split in parallel traveling to two different LED’s lighting them up, to then return to ground. In this circuit two new tools were applied, solder and hot glue. When building the final product solder was used to attach connecting wires to certain parts, and to connect the final parts and wires to my PCB. And hot glue was simply used to secure certain parts to the final project box

**Questions and Answers**

|  |  |
| --- | --- |
| Results(V) | |
| \*look on schematic to see test point  \*a)LDR not covered  \*b)LDR covered | |
| a)0Ω resistance | 8.35 |
| b)0Ω resistance | 8.60 |
| c)a)50kΩ resistance | 1.39 |
| c)b)50kΩ resistance | 7.46 |
| d)a)100kΩ resistance | 0.66 |
| d)b)100kΩ resistance | 6.65 |

1. Describe the circuit both at rest and the influence adjusting the potentiometer has on the outcome. *At rest the current ether goes through the LDR to ground(in high light) or is pushed into the base leg of transistor(if in low light). If this happens that activates the transistor and current flows through it lighting up the LEDs. However, if the Potentiometer greatly changes things. As its resistance is increased the voltage getting through is significantly reduced. More so when the LDR is covered, the voltage increases, this happens because now rather than a low around 5kΩ resistance to get through ether a 22kΩ resistor or a near 5MΩ LDR. This obviously takes far more effort, so more voltage is needed.*

## Media

|  |  |
| --- | --- |
|  |  |
| Circuit 4.1 Schematic | Circuit 4.1 Breadboard Prototype |
|  |  |
| Soldered Circuit 4.1 (Bottom) | Soldered Circuit 4.1 (Top) |
|  |  |
| Automatic Night Light (Front) | Automatic Night Light (Top) |
|  | Link to YouTube Video:  <https://youtu.be/SdqNmFTCYu0> |
| Dry fitted PCB |  |

## Reflection

To conclude this circuit uses a transistor and a LDR to create the effects of a night light. I would change very little about my project in the end. The only things I would change would be to add heat shrink to the solder joint connected the LEDs to the wire, and to do a more refined job with the hot glue, maybe using Epoxy instead. However, with that said overall I was pleased with the outcome, because it does work as a product that has real life application.

# Project 3. The 3D Christmas Tree

## Purpose

To create an oscillating function and use it to make the effect of flashing lights, then to use the kit provided to make the 3D Christmas tree based off of the prototype.

## Reference

Project 3. Description

[http://darcy.rsgc.on.ca/ACES/TEL3M/1617/TasksFall.html - Tree](http://darcy.rsgc.on.ca/ACES/TEL3M/1617/TasksFall.html#Tree)

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Path of Current Through an Analog Oscillator

<http://www.falstad.com/circuit/e-multivib-a.html>

## Theory

|  |
| --- |
|  |

In this project no new parts were used, however the previously used parts were used to create a new function. This function is called an oscillation, it uses two transistors, two capacitors and several resistors to send current one way for a short time then, switch and send current a different way. When paired with LED’s as the loads this can be used to create twinkling Christmas lights.

## Procedure

|  |
| --- |
| Parts List |
| 16 LED’s (4 Yellow, 4 Green, 8 Red) |
| 2 NPN Transistors |
| 2 82k Ω Resistors |
| 2 100k Ω Resistors |
| 2 1k Ω Resistors |
| 2 10 μF Capacitors |
| 9V Battery |
| 2 Connected Wires |
| 2 Springs |
| Solder |
| 2 Circuit Boards |

Path of current through Project 4.1

To begin with the current splits to parallel going through a 1kΩ resistor, through 4 LED’s (lighting them) and into a transistor as a charged capacitor releases its charge to open it. The second path it goes is charging a second capacitor, once this happens and its resistance increases enough and the other capacitor drains enough, the circuit oscillates. Once that happens it simply repeats, doing the exact same thing, on the other side of the circuit. However, this is how the circuit works once the cycle has started, in the beginning the current jumps back and forth with little bursts, increasing how much they charge the capacitors each time. To see an animation for this, look above.

Building this circuit was very different from previous. In order to reach the final product, it required a kit (picture bellow). After the circuit was dry fitted according to the instructions and markings on the circuit boards (paying attention to the polarities), the parts could be carefully soldered into place. The difference with this circuit is that once the parts have been soldered on to both boards the boards have to be fitted together. After being slid into place they are soldered together with connecting wires, to transfer the current to both boards. Adding the battery should result in the Christmas Tree to start twinkling.

## Media

|  |  |
| --- | --- |
|  | |
| Schematic of Circuit | |
|  |  |
| 0Dry Fit Circuit | 3D Christmas Tree Kit |
|  |  |
| Prototype in Mid Oscillation | Prototype in Mid Oscillation |
|  | Explanatory YouTube Video  <https://youtu.be/z-wPfH29NdQ>  Theatrical YouTube Video  <https://youtu.be/LiHkdE2fK8I> |
| Finished 3D Christmas Tree |  |

## Reflection

To conclude, this circuit used transistors, capacitors, and resistors, to create an oscillator, and in this case, Christmas lights. It was a very good example of not only a way to use a oscillator, but a more interestingly a way to create a oscillator, a generally digital function, out of analog parts. I was very happy with this circuit, though stressful to solder, the result is well worth it. And it has created a product that I show pride for and will undoubtedly use.

## Project 4. Digital (Transistor-Based) Logic Gates

|  |  |  |
| --- | --- | --- |
| **TRANSISTOR** | **SERIES** | **PARALLEL** |
| **NPN**  **BJT_symbol_NPN** | BinaryANDLogic.gif  Name: **AND** Symbol: and-gate-hi.png     |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | 0 | 0 | 0 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 1 | | Name: **OR** Symbol:   |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 1 | |
| **PNP**  **BJT_symbol_PNP** | Name: **NOR**  Symbol:   |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | 0 | 0 | 1 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 0 | | Name: **NAND** Symbol:   |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | 0 | 0 | 1 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 0 | |

# Project 5. 12 Hour Clock

## Purpose

To create a 12-hour alarm clock, staying as close to analog as possible.

## Reference

Aces Home Page

<http://darcy.rsgc.on.ca/>

4017 Decade Counter Explanation

<http://www.doctronics.co.uk/4017.htm>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

|  |
| --- |
|  |
| 4017 Decade Counter IC |
|  |
| Function of 4017 IC |

## Theory

This project used several new parts such the 4017 Decade Counter (Integrated Circuit). Also a previously used function was applied, the analog oscillator. The oscillator would send a signal to the 4017 IC’s CLOCK leg. (If the RESET and ENABLE legs were tied to ground and the chip is tied to ground in leg 8 and tied to power in leg 16) then the signal that entered the CLOCK leg would cause HIGH to be put out from the output 0 for the full wavelength, then once the oscillator oscillates and repeats the previous signal it will cause output 1 to go HIGH for the wavelength. And as before once the wavelength finishes output 2 will go HIGH and so on up until 9. Output ten is different, it is HIGH for 0-4 outputs, but turns off as soon as output 5 goes HIGH. Essentially this IC takes in a sine wave, for example one with a one second wavelength, and puts that out through ten outputs(excluding output 10), one after another, giving the illusion that its counting from 0 through to 9.

Another ability of this chip is that it doesn’t have to count all the way to 9. If current gives HIGH on the RESET leg (leg15) then the IC will restart and will change from putting current out from whatever leg it is and put current out of output 0, “resetting” the IC. For example to count 0-6 just tie output 7 to the reset input.

## Procedure

|  |
| --- |
| Parts List |
| 18 NPN Transistors |
| Connecting Wire |
| 32 LED’s |
| 5 4017 Decade Counter ICs |
| 1 Button |
| 2 10μF |
| 2 100μF |
| 4 1000μF |
| 4 2200μF |
| 2 Switch |
| 2 Diodes |
| 10 100 000Ω Resistors |
| 14 1000Ω Resistors |
| 5 330 000 Ω Resistors |
| 1 Buzzer |

As discussed in the theory the circuit uses a combination of an oscillator and the 4017 IC, the combination of which will be referred to as cells from now on. However, that simply gives the function of counting 0 to 9. To apply this to create a clock function 6 different cells must be created. The first puts out a signal every one second, the second puts out a signal every ten seconds. The third every minute, the fourth every ten minutes, and the fifth and sixth every hour and tens of hours. Though they are not connected at all when they run together it appears that they are. For example, after ten seconds it looks like they add up to ten and create a new place value, when really the second cycle has just finished one oscillation and the first cycle has completed ten, looping back to restart at 0.

However not all time units are 0-9, such as tens of seconds, tens of minutes, and tens of hours.

|  |  |
| --- | --- |
| LED Colour | Time |
| Yellow | Seconds |
| Tens of Seconds |
| Orange | Minutes |
| Tens of Minutes |
| Red | Hours |
| Tens of Hours |
| Green | AM |
| Blue | PM |

As discussed in the Theory the RESET leg can be used to accomplish this, for tens of seconds and tens of minutes they both have output seven connected to their RESET legs, so they only loop 0 – 6. For tens of hours however its different

The hardest part is making the oscillations perfectly timed to what they are supposed to be. For example getting exactly one second, or one hour is very hard in the analog oscillator. To do this the equation RC=T (Resistance x Capacitance = Time) must be applied. Even though this is useful to get a ballpark answer it is not at all perfect in getting the exact answer. So to hone down on the exact answer trial and error had to be used.

This circuit didn’t only function as a clock, it also had built in alarm. It would have inputs into the base legs of several transistors in series which would be plugged into the required time. For example if 7:35 AM was the time the alarm needs to go off, the wires would be plugged into AM, 7 hours, 3 in tens of minutes, and 5 in minutes. So when all of those went HIGH the alarm would go off. The initial design used a AND gate, and then an AND and NAND gate. This for no comprehendible reason didn’t work. So instead of the logic gates, NPNs were used to recreate the AND gate, which worked very effectively.

Another issue encountered was that it appeared that each cell skipped the first value, and this proved puzzling. However, it was due to the fact that this was powered by an analog pulse, not digital, meaning that the value isn’t ether HIGH, or LOW, it can be anything in between. So when the circuit goes HIGH, the analog oscillator isn’t immediately HIGH or LOW, it has to be somewhere in between for the first oscillation or so to get it running. For example, this is like a car engine starting, once it gets going it runs in perfect unison firing, refilling, compressing, firing, however, it needs a spark plug to get that loop going. One attempt to counter this was too run the circuit off of two switches. The first powers the circuit, and the second uses transistors to prevent the 4017 IC’s from receiving that power. So if the first switch is flipped that sends current to the oscillators starting them, then once they have gotten past the analog glitch in the beginning the second switch can be flipped giving power to the 4017 IC’s and starting the clock. This method was attempted on blocking the power, blocking the ground, and blocking the CLOCK leg from receiving a pulse, all of which failed. The more effective route was to attach a switch that when pressed went HIGH, and put that too all the 4017’s RESET legs. The same method used by the AM/PM, to reset the circuit.

## Failures

The initial idea was to not use an oscillator, but instead use almost just one side of one. Relying

|  |
| --- |
|  |
| Seven Segment Display (SSD) Common Cathode |

on one capacitor to fill and discharge to open a NPN, without the aid of an opposite side. This didn’t end up working despite many different attempts.

From then the 4017 IC was used. However, though it does a very good job of counting with LED’s it is very difficult to use with a Seven Segment display. Despite getting close to figuring it out, it was incredibly costly, complicated, and took up massive amounts of space. It worked through sending the signal to a “decoding” section of the breadboard that would split that current up and send out signals to however many outputs to the required legs of the SSD. However with all these wires being connected into the same seven segment display legs, the current would go back through the other wires instead of into the display and end up lighting all seven pieces of the display. To solve this issue NPN’s were applied, but to no useful extent. However diodes could solve the issue, and more so, using PNP’s could help too, because in the digits 0-9 more lines are being lite up than being left dark. So attaching wires to the legs that needed darkness was far easier and more effective than attaching wires to the legs that need light. Despite this, after significant trouble with the diodes and blowing all four PNP’s, and even if the circuit had worked, it required massive amounts of wire, diodes, transistors and space. So a simpler route was taken, this idea used the 4510 4511, these, when put together could easily turn a pulse into a signal that the SSD could easily understand and display. After several attempts the plan was not used because the circuit and write-up so far had been done around the 4017, and, the 4017 plus the LEDs stayed more true to the original idea of a analog clock, so that option was chosen.

## Media

|  |  |
| --- | --- |
|  |  |
| Alarm Using AND and NAND Gates Prototype | First Clock Prototype |
|  |  |
| Final 12 Hour Clock | Early Analog Clock Schematic |
|  |  |
| Early Analog Clock Schematic | Early Analog Clock Schematic |
|  | |
| Final Schematic | |
| YouTube Video:  <https://youtu.be/rFLpLIy3VX4> | |

## Reflection

Overall this was a very interesting and difficult project. Because I made my own schematics it could be quiet frustrating putting hours of research and building into a failed idea. However, at the same time it made it so much more important because this was my design, the only help I got was the suggestion of the 4017 Decade Counter, a piece of hardware I didn’t know of. Even though it is disappointing that some of the timings are off, and it undergoes glitches, it is very nice to have built something like this completely on my own.

# TEI3M-Arduino

2017-2018

# Project 6. Traffic Light Assembly and Testing

## Purpose

To build a traffic light and apply our knowledge of the Arduino create functioning code for it.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Project Description

[http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html - TrafficLightAssembly](http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#TrafficLightAssembly)

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

## Procedure

|  |
| --- |
| Parts List |
| 3 LEDs (1 Red, 1 Green, 1 Yellow |
| Custom PCB |
| 220 Ω Resistor |
| Arduino |
| Pin Header |
| Solder |
| USB A to USB B Cable |
| Power Supply(Laptop) |

Hardware

The hardware of this project is rather simple. The basic principle is three different inputs lead to three separate LEDs, after which all the input pins converge into a 220 Ω Resistor and GND. Whenever one of the input pins goes HIGH, the LED on that givin line will light up. When this devices inputs are put into three digital outputs(in this case 11,12,13) and on GND of an Arduino, code can be created to make this device work like a Traffic Light.

Software

Just like the hardware, the software is very simple. The idea of this device uses a blinking light, the basic code on the Arduino. Without any adjustment, the Arduino puts out HIGH on output 13 every other second, for a second. To make the Traffic Light, two more statements were put in affecting outputs 12 and 11. The result being, output 13(connected to the green LED) goes HIGH for one second, then LOW. Upon going LOW output 12 (connected to the yellow LED) goes HIGH, for one second, then LOW, and this repeats with output 11(connected to the red LED). Then once the sequence is over, because its in a LOOP, it will repeat starting from output 13.

Failures

One failure encountered was trying to condense the code using a single HIGH LOW command for all three LEDs, instead of each LED having an individual one. The same effect of three different commands could be achived by one command with a changing output. Every LOOP was supposed to decrease the output number. It would start at 13, then decrease to 12, etc. Once the the HIGH LOW command was sent to output 11(connected to the red LED), there was an IF statement that would return it to the output 13, therefore starting the sequence over. In the end lack of experience and knowledge with the language made debugging very slow, so the idea was put down.

## Code

void setup() {

 byte green = 13;

pinMode(green,OUTPUT);

 byte yellow = 12;

pinMode(yellow,OUTPUT);

 byte red = 11;

pinMode(red,OUTPUT);

}

void loop() {

byte green = 13;

 digitalWrite(green,HIGH);

 delay(1000);

 digitalWrite(green,LOW);

byte yellow = 12;

 digitalWrite(yellow,HIGH);

 delay(1000);

 digitalWrite(yellow,LOW);

byte red = 11;

 digitalWrite(red,HIGH);

 delay(1000);

   digitalWrite(red,LOW);

}

## Media

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
| PCB | Arduino | | Finished PCB |
|  | |  | |
| Completed Traffic Light(in Arduino) | | Traffic Light Schematic | |
|  | |  | |

YouTube Video

<https://www.youtube.com/watch?v=pafKTivASXA&t=13s>

## Reflection

This project was a very usefull introduction to the Arduino. Though both the hardware and software were realitivly simple, they introduced and reintroduced important skills that can be applied to much more complex devices. More than that the creation of the hardware of this project can be applied and used to learn more Arduino. Overall an excellent beginning to the new year and the new topics.

# Project 7. Digital and Analog: Reading and Writing

## Purpose

To produce a effective software, that shows understanding of Digital and Analog: Reading and Writing statements. This will be used in conjuction with hardware to create the effect of an intersection.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Project Description

[http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html - ReadingandWriting](http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#ReadingandWriting)

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Arduino Language Reference

<https://www.arduino.cc/en/Reference/HomePage>

## Theory

### Software

Digital Write – Digital Write puts out a HIGH or LOW on a defined Output from the Arduino.

|  |
| --- |
| Pulse Width Modulation(PWM) |
|  |
| Pulse Width Modulation is a method of appearing to give out an analog voltage by rapidly switching between HIGH and LOW. |

Analog Write – Analog Write puts out a variable voltage(PWM) on a defined Output from the Arduino.

Digital Read – Digital Read returns either HIGH or LOW from an defined input.(voltages of under 2.5 will generally be rounded to LOW and Voltages of above 2.5 will generally be rounded to HIGH)

Analog Read – Analog Read returns a integer value between 0 and 1023. The ratio of this integer is the ratio of voltage between 0V and 5V.

### Hardware

|  |
| --- |
| The LM386 Sound Detector |
|  |

LM386 Sound Detector – The LM386 Sound Detector is a circuit board that uses a microphone and LM386 amplifier to give out a signal when it detects noise above a certain level. This level can be set by a potentiometer and when reached it outputs LOW out of the “OUT” leg, a leg normally outputting HIGH.

## Procedure

|  |  |  |
| --- | --- | --- |
| Features | Hardware | Software |
| Traffic Light | A PCB that has 3 inputs and one output to GND. The inputs lead to a Red, Yellow and Green LED. After leading to GND is a 220Ω Resistor. | Digital Write will be used to activiate the Red and Green lights, and Analog Write will be used to make a brightening and dimming effect from the Yellow light. |
| Car Sensor | Uses an Light Dependant Resistor(LDR), in conjuction with a voltage divider to send a volage signal(between 0 and 5V) to a Analog In input on the Arduino. If the voltage is below a level it shortens the length of the Red Light. | The Analog Read takes the voltage input and turns the ratio of Voltage to that ratio between 1023. If this Integer is a bellow a certain limit it means a car has pulled up over the LDR. This will trigger an IF statement to reduce the time at the red light. |
| Siren Sensor | The LM386 Sound Detector which normally outputs HIGH, will output LOW if it detects a siren. When this happens the Traffic Light will start flashing a yellow hazard.(If this was a complete intersection it would also send a signal out to the perpendicular Lights to turn Red) | If Digital Read detects HIGH it will continue on with the usual light sequence, however when it detects LOW, it will trigger a WHILE statement. This causes an Analog Write signal that increases and decreases in brightness to be sent to the Yellow Light. |

|  |
| --- |
| Parts List |
| Arduino Uno |
| Connecting Wire |
| LM386 Sound Detector |
| 1000Ω Resistor |
| Light Dependant Resistor |
| USB A to USB B Cable |
| |  | | --- | | Traffic Light | | 3 LEDs (1 Red, 1 Green, 1 Yellow | | 220Ω Resistor | | Custom PCB | | Power Supply(Laptop) | |

Issues

The only real issue in this assignemnet was the sound sensor. This device was incredibly problematic as it rarely actually outputted a solid or reliable signal. Even if the sensitivity was turned to the max(before it started picking up minut noises that set it off constantly) no amount of either naturel or electronic noises could set it off. Several different Hertz were tried(including the recommended from the microphone description).

## Code

// Purpose   : To create a traffic Light that reacts to light, and sound.

// Reference : http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#ReadingandWriting

// Author    : James Corley

// Date      : Saturday October 21 2017

// Status    : Working

//output(Traffic Light) pins

uint8\_t greenpin = 13; //green light pin

uint8\_t yellowpin = 10; //yellow light pin

uint8\_t redpin = 11; //red light pin

uint16\_t basetime = 1000;//a base time for all Traffic Light colors

//analog write variables

uint16\_t flashtime = 100; //the duration of the

uint8\_t brightness = 0; //the value of the analogWrite

uint8\_t increase = 10;//the increase in the analogWrite

//sound detector varables

uint8\_t sirenpin = 5; //sound sensor input pin

boolean sirenstate;//Determines if the sound sensor is HIGH or LOW

//LDR variables

uint8\_t sensorpin = A0;//the pin the LDR(car detector is connected too)

uint16\_t sensorvalue; //the voltage value converted by analogRead

uint16\_t greaterlesssensor = 700;//variable sensorvalue will be compared too

//lighton function for standard traffic light

void lighton (byte pin, int duration) {

 digitalWrite(pin , HIGH);

 delay(duration);

 digitalWrite(pin , LOW);

 sirenstate = digitalRead(sirenpin);

**Serial**.println("Button State: " + String(sirenstate));

}

//declaring pins that will be used

void setup() {

**Serial**.begin(9600);

 pinMode(greenpin, OUTPUT);

 pinMode(yellowpin, OUTPUT);

 pinMode(redpin, OUTPUT);

 pinMode(sirenpin, INPUT);

 pinMode(sensorpin, INPUT);

}

void loop() {

sirenstate = digitalRead(sirenpin);//checking the state of the sound detector

while (sirenstate == LOW) {

   for (int x = 0; x < flashtime; x ++)//FOR LOOP for the hazard light

   {

   //send red signal to perpendicular lane of traffic

   analogWrite(yellowpin, brightness);

   delay(50);

   brightness += increase;

   if (brightness == 250) brightness == 0; // returning brightness to 0

       sirenstate = digitalRead(sirenpin);//checks sound detector state

   }

 }

 while (sirenstate == HIGH) { // the normal traffic light configuration

   lighton(greenpin, basetime); //sets off greenlight

   lighton(yellowpin, basetime);//sets off yellowlight

   lighton(redpin, basetime);//sets off redlight

   sensorvalue = analogRead(sensorpin);//reads voltage from the “car sensor”

   delay(1);

   if (sensorvalue > greaterlesssensor) {//determines IF a car has pulled up

     lighton(redpin, basetime); // if not the Red light will repeat itself

   }

   sirenstate = digitalRead(sirenpin);//checks sound detector state

 }

}

## Media

|  |  |
| --- | --- |
|  |  |
| Top View Modified Traffic Light | Front View Modified Traffic Light |
|  | Youtube Video  <https://www.youtube.com/watch?v=XLsGsZvzXTw&t=37s> |
| Schematic of Modified Traffic Light |  |

## Reflection

This project, though it had the appearance of simplicity ended up being a new and unique software/hardware puzzle to solve. Even though there were hiccups along the way and the sound sensor has persisted to confuse, the result of this project is ecxiting because it is applicable in the world today. Though the methods I used are definitely not the most technologically advanced, they are an answer. In essense it ended up being a lesson in Arduino language, a tool that when used with all the tools used last year could make a very interesting ISP.

# Project 8. Design Challenge 1

## Purpose

The purpose of this challenge is to create a unique project box designed to house the AdaFruit Perma-Proto board.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Project Description

[http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html - DesignChallenge1ER](http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#DesignChallenge1ER)

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Adafruit Perma-Proto

<https://www.adafruit.com/?q=perma-proto>

ViaCad(Project was designed in ViaCad)

<http://www.punchcad.com/p-43-Punch-ViaCAD-2D3D-v10>

## Acknowledgments

Mr. Paul Elia (<http://www.eliasculpture.com/index.html>) – Design Instruction and Support

Kreher Fiset-Algarvio & Darius Dadyburjor – Printing Project Boxes

Sawdust&Noise (<http://www.wecut4u.com/)> - Cutting and Etching acrylic

## Procedure

From the start of this assignment the main goal was versatility, which to me translated as modularity. Having a box that could be transformed to the creators needs would be the main goal. For example, the first design was supposed to be able to combine with a another box of the same type for extra space, and/or a smaller box holding a a 9V battery. Whatever the creator needed could be added too. This idea changed, to give the creator a box that lacked the modular nature but could be cut and altered to perform the desired function.

**Design Descions**(This is the summary used to explain the box to the grade 10’s and 12’s) “*The primary focus of my box is to be versatile and cheap. As oppose to having a screwable on top and bottom, the yellow pins are used as a compression screw, pulling the top and bottom together(cutting the number of required screws and screw heat inserts by half). By using these yellow pins it reduces cost while still allowing access to the bottom of the box. On the bottom plate there is a small housing for a 9V battery, but if that is not required it can be removed depending on your needs. For example if the box will solely use external power there is a hole cut out for the barrel jack on the side. There is also a rectangular cut-out for an Arduino reprogrammer, obviously not everyone will need that but it can easily be used for something else, or simply ignored. This is because behind these cuts there is a small piece of removable acrylic. This piece allows for a perfect seal while the box is in transit, or if the precut holes are not going to be used. Keep in mind if ever you need to cut your own opening you can easily do that. This box can also be extended, if more space is required than one box, it is 79possible to place another separate box on top of the other(using one top plate and one bottom plate) and with modified yellow compression screws pull it all together. To finish important to remember for all the boxes that measurements may not be perfect and fit awkwardly, but if chosen they will be fixed. Also this case can be adapted into an extended form for a full board. The cost of this case (including perma proto half board) would be around $13.20.”*

## Evolution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Core Idea | Power | Modularity | Price |
| 1) | End panel removed to allow for acrylic slide in | -with separate small box to house a 9V battery which can be slid into the slots for the acrylic | -allow two boxes to fit acrylic face to acrylic face with acrylic removed). | 6 Screws and Heat Inserts |
| 2) | Screw on top and bottom |  | -boxes can be screwed top too bottom | 10 Screws and Heat Inserts |
| 3) | End panel with holes cut out (plus acrylic slide in)  +  Screw on top and bottom | -Allows for a power cable to be plugged in  -Any other cables can be plugged in through cut out holes | -boxes can be screwed top too bottom | 10 Screws and Heat Inserts |
| 4) | End panel with holes cut out (plus acrylic slide in)  +  Plastic compression screw design | -Allows for a power cable to be plugged in  -Any other cables can be plugged in through cut out holes | -Two boxes can be put on top of one another with modified compression screws | 6 Screws and Heat Inserts |
| 5) | End panel with holes cut out (plus acrylic slide in)  +  Plastic compression screw design(Bottom plate has 9V battery holder) | -Allows for a power cable to be plugged in  -Any other cables can be plugged in through cut out holes  -9V battery can be held internally | -Two boxes can be put on top of one another with modified compression screws | 6 Screws and Heat Inserts |

This project went through many stages, with Three main factors in mind: Power, Modularity, and Price(the following chart shows the evolution of design ideas with 1) being initial 5) being final).

## Media

|  |  |
| --- | --- |
| /Users/student/Downloads/IMG_20171121_130312001.jpg | /Users/student/Downloads/IMG_20171121_130347632.jpg |
| Acrylic Side | Acrylic Top |
| /Users/student/Downloads/IMG_20171121_131037927.jpg | /Users/student/Downloads/IMG_20171121_131013078.jpg |
| Finished Project | Finished Project(end view) |
| /Users/student/Downloads/IMG_20171121_131024667.jpg | /Users/student/Downloads/IMG_20171121_130248934.jpg |
| Finished Project(top view) | Base Plate(with battery) |
| /Users/student/Downloads/IMG_20171121_130154370.jpg | /Users/student/Downloads/IMG_20171121_130238576.jpg |
| Base Plate(with pins & battery) | Pins(compression screws) |
| Explanation Video:  <https://www.youtube.com/watch?v=GnBt6PaX3AA> | |
|  |  |

## Reflection

I found this project to be very intriguing. The notion that an idea can turn into a real thing I can hold is incredible. More than that the fact that it can also be turned into a product that other ACES, or the wider public, can use inspired a deep commitment to making a truly useful product. The result of this, I believe, is the creation of a unique project box that is applicable in the real world.

# Project 9. POV: Dual Seven Segment Display Application

## Purpose

To show the output of a sensor(in this case a temperature sensor) using Persistance of Vision(POV) on Duel Seven Segment Display.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Project Description

[http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html - POV1](http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#POV1)

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

LM35(Temperature Sensor) Data Sheet

<https://www.techshopbd.com/uploads/product_document/LM35%20datasheet.pdf>

## Theory

This project has two new concepts Persistance of Vision, and the new part, the LM35 Temperature Sensor.

Persistance of Vision

|  |
| --- |
| Definition |
| “the retention of a visual image for a short period of time after the removal of the stimulus” |

This concept is applied in the project with the double seven segment display(SSD). As is

|  |
| --- |
|  |
| This is the small circuit that produces the POV signal. It takes a square wave from the Arduino, and splits into two opposite signals(CC1 amd CC2) which lead to the SSD’s GND’s. |

apparent in the picture, only the first digit has wires directly from the Arduino UNO. Despite this both digits light up, this was accomplished through POV. Underneath the SSD there is wireing connecting the segment of the first to that of the second. So when a signal is sent out of the

Arduino it is presented to both displays, which seems problematic as they would be the same. The way to make them different is to only have one on at a time(in this case it’s a common cathode SSD so only one of the displays is tied to ground at a time). So when one is on it is shown the desired digit and when they switch the other one shows its desired digit. Logically it would seem like that would cause the SSD to flicker, but this is where POV comes in. Since the eye can only detect light so quickly when something turns on and off fast enough the human eye simply interprets it as constantly on.

|  |
| --- |
|  |

LM35 Temperature Sensor

The LM35 is a intuitive Integrated Circiut(IC) that has a Voltage pin, GND pin, and its Output

pin. This pin sends out an analog signal according to the temperature it is in. To decode this signal into a Degrees Celsius reading multiply it by 0.48828125 (0.500mV/1024).

## Procedure

|  |
| --- |
| Parts List |
| Arduino Uno |
| Jumper Wires |
| LM35 |
| Double Seven Segment Display |
| NPN Transistor |
| PNP Transistor |
| 1000Ω Resistor |
| 2 330Ω Isolated Resistor Networks |

Hardware

This circuit, as seen in the parts list, is particually small, and seemingly uncomplex. Pins 1 to 7 on the Arduino are sent to isolated resistor networks which in turn lead to the a1 to g1 on the double SSD. Underneath the SSD as discussed above the segments lead to their equivalent on the second display(ex. a1 leads to a2). Pin 12 sends out a square wave too a small circuit(pictured above) that splits that square wave by sending it to a PNP and a NPN. Whenever one is HIGH the other is LOW, so using this logic if it is connected to the SSD’s GND’s it can create the POV signal. Pin 13 is simply set HIGH and it leads to the LM35’s Voltage pin. This is important because it runs at 40 mA which is far closer to the 60mA limit of the sensor than the 500mA-1A of the 5V pin. A0 is the other pin that interacts with the LM35, it however is set as the analog input for the sensors OUT leg. Other than the 5V and GND pin going the the power bar there are no other Arduino pins being used.

Software

Though at face value the hardware seems basic, the ideas behind it and the software are where this circuits complexity arises. The principle of the circuit is that pin twelve goes HIGH(*PORTB = 1 << 4 | 1 << 5;*) and LOW(*PORTB &= ~(1<<4);)* making the square wave(discussed above). When HIGH it produces the tens digit(*PORTD = display[celsius / 10];* ) and when LOW it produces the ones digit (*PORTD = display[celsius % 10];* ). *celsius* is the raw analog input value mulitiplied by the temperature decoder(0.48828125), it is the value that is displayed through the SSD. The / (division) and % (remainder) operators are then used to separate *celsius* into its ones and tens digit. Once the value is determined that needs to be displayed through the SSD’s respective digit, it is used to acsess its corresponding SSD value, which is stored in the array *display[]*.(ex *display[1]* is the value (*0b00110000*) which turns on the two right segments of the digit on making a 1. After each value is displayed it waits a short time(*delay(10);*) and then clears the seven segment( *PORTD = 0;*). Once it cycles thorugh the HIGH and LOW of the square wave, it briefly checks *celsius* value again and tehn repleats.

## Code

// Purpose   : To display the value of a sensor on a Double

//             Seven Segment Display using POV

// Reference : http://darcy.rsgc.on.ca

// Author    : James Corley

// Date      : Jan 20 2018

// Status    : Working

uint8\_t sqrwave = 5;        //value to bitshift in PORTD for pin 13

uint8\_t sensorpower = 4;    //value to bitshift in PORTD for pin 12

uint8\_t celsius;            //the temperature in celsius

float rawtocelsius = 0.48828125;//this value\*analog input = celsius

uint8\_t display[10] = {     //array holding SSD decoding value

 0b11111100,//0

 0b00110000,//1

 0b11011010,//2

 0b01111010,//3

 0b00110110,//4

 0b01101110,//5

 0b11101110,//6

 0b00111000,//7

 0b11111110,//8

 0b00111110};//9

void setup() {

 DDRD = 0b11111111;                     //turns pins 0-7 to OUTPUT

 DDRB = 1 << sqrwave | 1 << sensorpower;//turns pins 12-13 to OUTPUT

 PORTB = 1 << sensorpower;              //pin 13 HIGH

}

void loop() {

 celsius = (analogRead(A0) \* rawtocelsius);//produce celsius value

 //start of square wave

 PORTB = 1 << sqrwave | 1 << sensorpower;//pin 13(and 12) HIGH

 PORTD = display[celsius / 10];          //display tens digit

 delay(5);

 PORTD = 0;                              //pins 0-7 LOW(clear SSD)

 PORTB &= ~(1 << sqrwave);               //pin 13 LOW

 PORTD = display[celsius % 10];          //display ones digit

 delay(5);

 PORTD = 0;                             //pins 0-7 LOW(clear SSD)

 //end of the square wave

}

## Media

|  |  |
| --- | --- |
|  |  |
| Full Circiut | Wiring Underneath the Seven Segment |
|  |  |
| Arduino | Close Up Circiut |
| YouTube Explination Video  <https://www.youtube.com/watch?v=Z4RDNmle0ug&t=1s> | |

## Reflection

This project was actually one of the more rewarding projects I have done so far. Even though all of the projects so far had their own function, projects like this, the night light, and to an extent my clock were particually rewarding because they are a direct product. I can plug in my nightlight and it will work, similarly I can plug this in and get the temperature. As well, learning and applying the medium level techniques(however tedious they can be) makes the coding far more interesting because its cutting away the buffer and middleman that stands between us and the computer. Either way this project was a large conjunction of new ideas, and to have them all come together finally was nice to see.

# Project 10. DIY Bicolor LED Display

## Purpose

To first solder a 4x4 Bicolour LED Matrix, and then use our code to produce a unique display.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Project Description

[http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html - DIY](http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#DIY)

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

## Theory

LED Matrix

|  |
| --- |
|  |
| Bicolour LED(Ignore R, this version was NOT internally resisted). |

The most obvious new piece behind this assignment is the LED Matrix. There are many commercial versions of this product, ranging in size and number of LED’s. However, in this project the Matrix was created through soldering 16 Bicolor LED’s together, in a 4x4 layout.

Bicolour LED

The Bicolor LED is the other new tool applied in this project. In the version used here there are only two pins, as oppose some versions that contain three. Essentially instead of being cathode and anode pins, they can be either. If current is pushed through one way(A-B) then one color(Red) will light up. But if the current flows the other way(B-A) then the other colour(Green) will light up. This is because LED stands for Light Emitting Diode, the Diode part of that means that it will only let current pass through one way. Check the digram to see this.

|  |
| --- |
|  |
| 595 Shift Register |

595 Shift Register

The final part I used was the 595 Shift Register. This Integrated Circiut’s(IC) takes in data at one input pin(SER), and outputs it in parallel(QA-QH). The data is presented to the input pin one bit at a time, and every clock cycle(tie a square wave to clock pin(SRCLK) to produce this cycle) that bit is pushed into the output line, shifting any data in front of it down one. Once 8 bits of data have been pushed into this output line, trigger the latch(RCLK) to send out the byte of information in parallel through Outputs 0-8(QA-QH).

|  |
| --- |
|  |
| Pin layout of DIY Matrix |

## Procedure

|  |  |
| --- | --- |
| Shift Register | Matrix |
| 14(QA) | 0 |
| 1(QB) | 1 |
| 2(QC) | 2 |
| 3(QD) | 3 |
| 4(QE) | 4 |
| 5(QF) | 5 |
| 6(QG) | 6 |
| 7(QH) | 7 |

To start the most complex section was creating the correct bytes of data to send to the shift register. Because this is a Bicolour LED Matrix the data to be sent has to be determined in a very unique way. To find the correct patterns, many BitWise operators were applied. Once the correct signal is determined it is sent through pin 13 out, too pin 14 of the shift register (SER). After 8 clock cycles, the full byte will be loaded into the shift resigter, so the latch pin can be put HIGH and the byte released in parallel. These parrellel connections are tied to their respective LED matrix counterparts(see table on the left) For the pin mapping on the matrix refer to the diagram on the left. Then, according to the signal sent the appropriate LED’s will turn on. If the current if flowing from 0-3 to 4-7 then the LED will be green. If it flows the other way, it will be red. This circuit is extended with a switch and a button. The switch, depending on the position will either send HIGH or LOW to pin 9. From there a digitalRead statement will read in the position of the switch and allow the display to be either automatic, or manually changed. If it is the latter, then the manual change will occur with the assistance of a button. The button works similarly to the switch, it is tied to pin 8, where a digitalRead statement will determine when its pressed. Upon pressing, it will switch between displays, with the help of a switch case statement. Each time the button is pressed it adds one to a currentdisplay variable. Then, depending on the value of currentdisplay, the correct case will be determined and the function for the desired display called. Once currentdisplay reaches 4, it will activate the default case, which returns its value to 0.

Failure

|  |
| --- |
| Parts List |
| Arduino Uno |
| Jumper Wires |
| 595 Shift Register |
| 16 BiColor LED’s |
| Button |
| Switch |

After several attempts with the Bicolor matrix, it becomes evitident that it is impossible to light only a single point up. This became promlematic because that means that making a custom design is essentially impossible. In an attempt to fix this I altered my design to use two 595 shift registers and NPN transitors. The goal was to create a state that was neither HIGH or LOW, still using the shift registers. One shift register would be used per side. The low nibble of each shift register would then be tied to its respiective base pin. So essentially the low nibble would descide which of the matrix pins would either be HIGH/LOW or neutral. Then the high nibble of that transitor would send the actual desired signal(to the Collecter pin). However, transistors amplify current, so when I was sending LOW to Collecter pins and HIGH to the Base pin, the result on the Emitter was 3.5V. This happened because LOW isn’t nessisarly 0V there is often a very small amount of voltage even if a LOW signal is sent. This is often to little to be an issue, but when it is sent thorugh a transistor is amplifies this small amount, into a a very problematic amount. Because of this everything was essentially HIGH, which meant no LED’s could turn on.

Code

// Reference: http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#DIY

// Author   : James Corley

// Date     : Feb 10, 2018

// Status   : WORKING

uint8\_t serialpin = 13;//setting pin values

uint8\_t clockpin = 12;

uint8\_t latchpin = 11;

uint8\_t powerpin = 10;

uint8\_t switchpin = 9;

uint8\_t buttonpin = 8;

uint8\_t currentdisplay = 0;

uint8\_t bival[] = {(0b00001111),//array with desired binary display values

                  (0b11110000),

                  (0b11111111),

                  (0b01010101),

                  (0b00001010),

                  (0b10100000),

                  (0b01101001),

                  (0b10010110),

                  (0b10011001),

                  (0b01100110)

                 };

//function completely deal with shift register.

void shiftout(uint8\_t shiftval, uint8\_t waitfor)

{

 delay(waitfor);//a delay that can be set when the function is called

 digitalWrite(latchpin, LOW);//latchpin low to push in information

 shiftOut (serialpin, clockpin, LSBFIRST, shiftval);//push in information

 digitalWrite(latchpin, HIGH);//latchpin high to release information

}

void raindisplay() {

 for (uint8\_t i = 0; i < 16; i++) {//creates the rain display

   if (i < 4) {

     shiftout((bival[3] << i) & (bival[1]) | (bival[4]), 1000);//rain

   } else if (i < 8) {

     shiftout((~bival[3] << i - 4) & (bival[0]) | (bival[5]), 1000);//rain

   } else if (i < 12) {

     shiftout(bival[5] << i - 8, 1000); //shift green lines down

   } else if (i < 16) {

     shiftout(bival[1] >> i - 12, 1000); //full green

   }

 }

}

void squaredisplay() {

 for (uint8\_t i = 0; i < 16; i++) {//diagonal square display

   if (i < 4) {

     shiftout(bival[0] << i, 1000);

   }//bottomright too topleft square

   else if (i < 8) {

     shiftout(bival[1] >> i - 4, 1000);

   }//topleft too bottomright square

   else if (i < 12) {

     shiftout(((bival[0] << i - 8) & (bival[1])) | (bival[0] >> i - 8 & bival[0]), 1000);

   }//topright too bottomleft square

   else if (i < 16) {

     shiftout((bival[1] >> i - 12) & (bival[1]) | ~(bival[2] << i - 12), 1000);

   }//bottomleft too topright square

 }

}

void explosiondisplay() {

 for (uint8\_t i = 0; i < 5; i++) {

   for (uint16\_t x = 0; x < 1500; x++) {

     shiftout(bival[1] | bival[0] >> i, 0); //green door closing(l to r)

     shiftout(~(bival[2] << i), 0); //red door closing(r to l)

   }

 }

 for (uint8\_t i = 0; i < 3; i++) {//explosion display

   for (uint16\_t x = 0; x < 1500; x++) {

     shiftout(bival[6], 0);

     shiftout(bival[7], 0);

   }//orange inner square and four corners lit

   for (uint16\_t x = 0; x < 1500; x++) {

     shiftout(bival[8], 0);

     shiftout(bival[9], 0);

   }//outer led's lit(without corners)

 }

}

void setup() {

 DDRB = 0b11111100;//setting required pins to output(10,11,12,13)

 digitalWrite(powerpin, HIGH);//creating resisted power supply

}

void loop() {

 if (digitalRead(switchpin) == HIGH)

 {//if the switch is high, it will cycle through the three sections automaticallly

   raindisplay();

   squaredisplay();

   explosiondisplay();

} else {

   if (digitalRead(buttonpin) == HIGH) {

     currentdisplay ++;

   }//if the switch is low it will switch displays only if the button is pressed

   switch (currentdisplay) {

     case 1: //if the button was pressed once, it will do the raindisplay

       raindisplay();

       break;

     case 2://if the button is pressed again it will do the square display

       squaredisplay();

       break;

     case 3:// if the button is pressed a third time it will do the expolsion display

       explosiondisplay();

       break;

     default://if the button is pressed again it will just reset it back to zero

       currentdisplay = 0;

       break;

   }

 }

}

## Theme

As I made the different displays, I tried to make a small movie with the tools I had. I ended up making a small small story from the three main display sections. The following is the little story that my display tries to convey.

Creation(the rain falling to the green ground, growing a world), existence(the cubes shifting between back and forth between green and red(good and bad)) and destruction(an impact between the green and red, which ended both of them in the orange explosion) but then in the end, the orange destruction ends up raining back down green and red, making a new world.

## Media

|  |  |
| --- | --- |
|  |  |
| Finished Circiut(Top) | Finsihed Circiut(Front) |
|  | [Explanation Video](https://www.youtube.com/watch?v=pEdSmSiM4gw)  [Display Video](https://www.youtube.com/watch?v=4WTXBBCAiEw) |
| Adapted Version  (failed) |  |

## Reflection

This project offered a great amount of freedom, which is a pleasant change form day to day work, and I am sure will provide many unique and entertaining results. However this amount of freedom also means that it is difficult to know when enough is enough. Even though the display I created(and the logic I created it with) are fun and interesting, to me. There is undoubtly a few students who will hit it out of the park and create a result that makes a simple display, like mine, look quite insignificant. Despite this, I did enjoy making the result, it took time, but mastering and applying the bitwise operators could be quite entertaining. In the end I was proud of the little display I was able to create, and story I was able to tell.

# Project 11. Frequency Spectrum Analyzer

## Purpose

To read in audio signals either through a audio jack or a microphone and display the result on an LED Matrix.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Project Description

<http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#FSA>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

MSGEQ7 IC Overview Blog

<http://nuewire.com/info-archive/msgeq7-by-j-skoba/>

Audio Cable Overview Blog

<http://www.cablechick.com.au/blog/understanding-trrs-and-audio-jacks/>

ATTiny84 Datasheet

<http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8495-8-bit-AVR-Microcontrollers-ATtiny441-ATtiny841_Datasheet.pdf>

ATTiny84 Guide

<http://highlowtech.org/?p=1695>

## Theory

This project bears some similarities to Project 10. DIY Bicolor LED Display so for an understanding of the 595 Shift Register refer to the above project. However, as opposed to the previous assignment relying on precoded signals to be displayed out through the matrix, this project has active inputs that determine the display.

|  |
| --- |
|  |

Microphone

This piece is not merely just a microphone. The microphone is mounted in a circuitboard that is primarily driven by the MAX9814 IC. Essentially the overarching effect is an audio device that can interpret ambient noise, and outputs a voltage to the ratio of that noise. However, due to the MAX9814, it also interally boosts the sound of quitefar away noises and reduces the sound of very loud noises. This helps because it stops one loud noise drowning out the rest, it makes it able to display a far greater range of noise, and, in doing so, create a more representative and realistic effect. So, in terms of wiring, GND and Vdd to the power rail. And then OUT is just the output pin. Both AR and Gain can be ignored for this specific project.

|  |  |
| --- | --- |
|  |  |

Audio Jack

The second audio is far less complex. It is simply an audio jack input device on a breakout board.

But, to use this device properly, the user has to understand the audio jack itself, because all the chip does is break out the audio jack to make it breadboard compatible. The first diagram on the right is the proper naming of the audio jack, and similarly will be the names on the chip. The second digram is what is coming from each of these components.(Note: these are not the only types, this specific jack is only for stereo audio, there are also TRRS which contains an microphone output and TS which is just mono audio. TRRS and TS are not compatible in the jack used in this project, and will short if used.

MSGEQ7

This IC takes in an audio singal from another input device, such as the two discussed above, and

|  |
| --- |
|  |
| Input and Output Signals form the MSGEQ7  (The graphic continues through to 16kHz) |

splits the spectrum into seven displayable bands, 63Hz, 160Hz, 400Hz, 1kHz, 2.5kHz, 6.25kHz and 16kHz. To achieve this the chip has one output, one strobe input (essentially a clock pin), and a reset pin (Note: this doesn’t go to reset on the ATtiny, it is just its name. The pin acts more as a latch pin from the 595 Shift Register. At the start of the loop it should be put HIGH for a short period and then switched LOW, this signals the start of the output at 63Hz). So once this HIGH-LOW sequence finished and STROBE goes LOW the 63Hz signal will be sent, the strobe goes HIGH and LOW again and the 160Hz signal will be sent. This continues through to the 16kHz signal, and then the RESET should be put to HIGH-LOW again.

|  |
| --- |
|  |

ATTiny84

The final new piece of hardware is the ATtiny84, this is a microcontroller similar to the ATmega that is in the Arduino UNO. It is a very complex IC that can be coded using the Arduino C++ language. Because of this it can be used in place of an Arduino. For interaction with this device the Sparkfun Programmer was used. It takes a USB input from the computer and with the help of a ACES circiutboard breaks it out into six pins MOSI, MISO, RESET, 5V and GND. In terms of hardware simply connect 5V and GND to the power rail, and the rest to their counterpart on the ATtiny84. Software is a bit more complex, see the Reference section for a guide on how to format the Arduino software. As well, for a more conclusive description in general, see this ICs datasheet, which is also linked in the Reference section.

## Procedure

First to preface, as this is offboard, this circuit does not rely on an Arduino. The ATtiny84 works in place of the Arduino, and once this the correct has been uploaded, the Sparkfun Programmer can be unplugged, and the circuit can simply powered by a battery.

Essentially, the circuit starts when an audio signal is picked up on either input. It will then send

|  |
| --- |
| Parts List |
| 2 595 Shift Register |
| ATTiny84 |
| MSGEQ7 |
| Switch |
| Audio Jack Breakout Board |
| Microphone(with amplifier) |
| 2 100μF |
| 10μF |
| 4 33pF |
| 4.7μF |
| 200kΩ Resistor |
| Jumper Wire |
| Sparkfun Programmer |
| ACES Sparkfun Programmer Breakout Board |
| 8x8 LED Matrix |

this signal from either its OUT or its TIP/RNG pins. However, both signals reach opposite sides of a switch, depending on which way the switch is flipped, its respective signal will be let throught toward the MSFEQ7. From there the MSFEQ7 will split this signal into seven different Hz values, and send then out one at a time according to the STROBE pins HIGH-LOW signal. From here each signal will be sent into the ATTiny84’s analog input pin, where an AnalogRead statement will turn this analog value into a quantifiable piece of data. This data will then be converted into something that can be displayed on the matrix. Initally I tried mapping the value from 0-1024 to 0-7, then I coud just raise 2 to the power of this value. Even though this sort of worked, and was responsive, it didn’t work in the desired fashion so a new approach was attempted. This by this by similarly converting the 0-1024 value to a value between 0-7. Then this data will be used to find the position in a array called displayVal. This array contains preset values that when put on a matrix will create columns of different sizes. For example at location displayVal[7], a full column will light up, but at location displayVal[4], then 5 LED’s in the column will light up. From here this converted data will be inputed into a new array called spectrum, with 7 elements(one for each Hz value). Since this is all in a for-LOOP it will iterate 7 times to fill the spectrum array. Then once the array is full, the code continues to another for-LOOP, this one however uses the shiftOut function to send out each element of the spectrum array. Unfortuantly there also needs to be a second shiftOut function, this to choose which column the data is being displayed on. This command shiftOuts all HIGH’s except for one LOW which is where the data will be being displayed. Not only that but both shiftOut’s are sent to the same input on one s

hift register, this is possible by linking shift registers together. Refer to the diagram of the 595 Shift Register in Project 10 to see that there is a pin called QH’, once one shift register is full, this pin can be connected to the input pin of another shift register and another byte can be pushed into the first, pushing the first byte into the linked shift register.

## Code

/ Author: James Corley

// Project: Frequency Spectrum Analyzer

// Date: Saturday Feb 24

// Status: Working

uint8\_t clck = 2;         //setting clock pin

uint8\_t latch = 1;        //setting latch pin

uint8\_t data = 0;         //setting data pin

uint8\_t resetPin = 10;    //setting reset pin

uint8\_t strobe = 9;       //setting strobe pin

uint8\_t signalin = 7;     //setting analog input pin

uint8\_t spectrum[7] = {}; //array to hold converted Hz values

//array holding predefined column data

uint8\_t displayVal[8] = {0x01, 0x03, 0x07, 0x0E, 0x1F, 0x3F, 0x7F, 0xFF};

void setup() {

 //setting desired pins to OUTPUT or INPUT

 pinMode(clck, OUTPUT);

 pinMode(latch, OUTPUT);

 pinMode(data, OUTPUT);

 pinMode(strobe, OUTPUT);

 pinMode(resetPin, OUTPUT);

 pinMode(signalin, INPUT);

 digitalWrite(resetPin, LOW);//setting reset to start LOW

 digitalWrite(strobe, HIGH); //setting strobe to start HIGH

}

void loop() {

 //reset data to start again from 63Hz

 digitalWrite(resetPin, HIGH);

 digitalWrite(resetPin, LOW);

 for (uint8\_t i = 0; i < 7; i++) {

   digitalWrite(strobe, LOW);       //clock cycle start

   delayMicroseconds(30);           // to allow the output to settle

//reading in and converting data to correct value, storing it in spectrum

   spectrum[i] = displayVal[analogRead(signalin) >> 7];

   digitalWrite(strobe, HIGH);      //clock cycle end

 }

 for (uint8\_t i = 0; i < 7; i++) {

   digitalWrite(latch, LOW);                      //close latch

   shiftOut (data, clck, LSBFIRST, spectrum[i]); //fill in Column Size data

   shiftOut (data, clck, LSBFIRST, ~(0x80 >> i));//fill in GND Column data

   digitalWrite(latch, HIGH);                     //release stored data

 }

}

## Media

|  |  |
| --- | --- |
|  |  |
| Entire Board(Top view) | Analog Input Circiut(Right)  ATTiny84 Programmer Circiut(Left)  (Top View) |
|  | YouTube Video:  <https://www.youtube.com/watch?v=f-bxhXxFsbs> |
| Analog Input Circiut(Right)  ATTiny84 Programmer Circiut(Left)  (Front View) |  |

## Reflection

This project reaffirmed something I knew in theory, but apparently failed to apply in practice. It is important for the worker, whoever they are to understand how they operate. And if a job needs to be done, not just sitting down and pouring hours into it. Instead they should understand what makes them work more effectively, whether that be location, music, other people, comfort, etc. For example, if I am hungry, I can accomplish most tasks, inefficiently, but it can be done. However, to accomplish hardware problem solving it takes a calm, collected, and patient mind. When hungry, I am the exact opposite of this. So it is absolutely pointless to try to work this way and instead, I, and anyone else in this situation, should work to put myself in an environment and state of being that allows work to be done efficiently. As I worked on this project, I grew so close to the correct answer, but I didn’t have the patience to look through each possible spot of error and debug efficiently. Instead, I just got frusterated, and my time was consumed on an impossible task. Eventually, with some assistance, and an attentive mind debugging was possible. And the issue that was found was a wire plugged into the wrong pin on the ATTiny84.

# Project 12. Robotic Hand

## Purpose

To apply our CAD skills and our experience with servo motors, to produce a humanlike robotic hand. It should be able to mimic almost all movements of the real human hand, as prepared functions.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

ViaCad(Project was designed in ViaCad)

<http://www.punchcad.com/p-43-Punch-ViaCAD-2D3D-v10>

Micro Servo Datasheet

<http://akizukidenshi.com/download/ds/towerpro/SG90.pdf>

## Acknowledgments

Mr. Paul Elia (<http://www.eliasculpture.com/index.html>) – Design Instruction and Support

Kreher Fiset-Algarvio – 3-D Printing the Design

## Theory

|  |
| --- |
| Servo Pin Color |
| V++ |
| GND |
| Signal |

In principle this project is simple. Essentially there is a network of string stretched out between the fingers and thumb, where the muscles would be in a human hand. The end of these lines leads down to the bottom of the hand and out to servo motors(discussed below), which can pull on the string and create the effect of muscles.

Servo Motor

|  |
| --- |
|  |
| PWM to Servo Rotation Conversion |

A servo motor is a rotary or linear actuator that generally allows control over, angle of rotation or linear position, as well as in more advanced cases its velocity and acceleration. In the case used in this project it’s a rotary servo motor, that has a variable angle of rotation but does not have variable velocity or acceleration. To control this device there are three wires presented out of the servo. The red wire is V++, the black is GND and the orange is the Signal pin. This pin interprets a PWM signal in this case, from an Arduino UNO. As the graphic on the right states and the datasheet for the specific servo confirms. A duty cycle of 1ms puts the servo at 0 degrees, 1.5ms at 90º and 2ms at 180º. These are not the only positions however, if a PWM signal of 1.25ms was sent the Arduino would move to 45º, etc.

Servo Application

Now for the application of the servo. Instead of most motors having a single rotary lead extending out, the servo has whats called a servo horn that can be attached to the end of its lead. The easiest way to understand this is thinking about a Ferris wheel. The only force is the rotation in the center of the wheel. But with this simple rotational force, it is able to generate horizontal and vertical displacement for the persion riding it. That is essentially what the horns do, transfering a rotational force, into something more usable like it this case a linear tension force in the string. This is the force which is then used to pull on the string and in turn act as the “muscle(compression)” part of the muscles in the human hand.

Driving Voltage Demanding Parts

This project used 8 micro servos, each one with a supply range of 4.8V-5V. This puts massive amounts of strain on the power line. Initially when using servos it seems like 5V pin from the Arduino is sufficient. And for driving a single servo this pin may be enough. However, once more servos are added, the Arduino’s 5V simply can’t do it. The problem is that the 5V pin isn’t just a raw 5V waiting to be tapped into, its already being used to power the Arduino, and the ATmega at its heart. So when too great a load is demanded from the pin the actual IC powering the Arduino fails to receive enough power, and begins to send out random signals. So an external power source is needed. There are two problems here. Firstly the micro servos require a very strict voltage, any more than 5V can damage them, but any less than 4.8V and they simply won’t function. So when using something like a voltage divider it can be hard to get the exact value. The second issue, is that parts like this drain batteries very quickly, so something like a 9V Battery is simply not usable. To answer the issue of high consumption, the only option is a wall jack. Batteries technically work but in terms of cost its incredibly wasteful. And for the second issue of exact voltage, the easiest answer is called a Voltage Regulator.

Voltage Regualtor

This is a small device that takes in a wide range of voltages and releases a fixed stable one. So

|  |
| --- |
|  |
| 7805 in TO-220 package |

for example, in this project 5V were needed, but the only power sources present gave out upwards of 9V. So the 7805 was used, which can convert 7V-35V to 5V. This part is very useful, but it has its drawbacks. The main of these being how it regulates this voltage. It essentially just turns all extra current into heat, which in low amounts is bearable, but if to high a voltage is used, it can blow the part. Now some regualtors can take more than others, but in general heat is not a good thing, and should be avoided. To fix this the easiest answer is called a heat sink. (Its also worth to note that sometimes these devices require capacitors to clean out any noise, in my project they weren’t applied because they was no issue but be aware they may be needed)

Heat Sink

|  |
| --- |
|  |
| TO-220 with Heat Sink |

A heat sink is just a part with very conductive part with high surface area. They are in a variety of electronic products due to their ability to dissipate thermal energy. For example in this project, they have the ability to be able to be screwed to the back of the 7805, since its in a TO-220 package.

3D Printing

|  |
| --- |
|  |
|  |

3D printing is a method of creating an object designed in CAD software. There are several ways of doing this but with most commercial products (such as the Ultimaker used in this poject) it essentially just feeds a length of plastic wire into a nozzle called an extruder. This extruder takes in the hard pastic and heats it up past its very low melting point. Once it has been heated into its liquid form it is sent out through the bottom of the extruder and applied to where the object is created. This extruder can be moved in the horizontal axis to make any sort of design. Its worth to note that this plastic has a very low melting point, and things like heat guns, and solding irons may inadvertatly melt the product.

## Procedure

|  |
| --- |
| Parts List |
| 8 Micro Servo |
| 7805 Voltage Regulator |
| TO-220 Compatible Heat Sink |
| Barrel Jack Input |
| Jumper Wire |
| Arduino UNO |
| String |
| Elastics(Silicone Vessel Loops) |
| Twist Tie |
| Power Source(with barrel output) |

The principle of this project to produce a 3D printed hand can simulate natural human motion by a network of strings stretched between the joints that change in length according to the angle of servo motors. The result being a artificial hand, that can mimic a human hand in almost every way.

Easily the largest part of this is the design. A design which was started before the ISP was even technically assigned. However, before the design could even be started it took lots of planning to see which direction to go. First the muscles and tendons in the hand had to be drawn out and understood. Then an ideal for how to mimic these with string was drawn out. The hardest part of this was easily the thumb joint which went through many iterations.

From these drawings measurements had to be applied to them. The internet was almost useless for this because even if there was somewhere with information, it didn’t have all the specifics needed. Because of this my brothers hand was used as the primary reference. Using digital calipers all sorts of measurements were taken and applied into the drawings.

Now the drawn design could be applied into ViaCAD. This process was slow, not only because simply making the objects, but because every action had to be thought out in depth before it was taken. For example, when ViaCAD started crashing(see below) it caused whole hour and a half periods of work to be wiped out. But to do that same work to catch up probably took 20 minutes. This because all the numbers and ideas were already fresh in mind, so to simply use the software was easy. The hard part was the thinking of sizes, location, shape, its movement, how real hands move, what is possible with the printer and margin of error while printing. So even though it could be frusterating at times, especially with the crashing, the design was overall a very enjoyable task.

|  |
| --- |
|  |
| Faulty Design(There should be a pillar by the yellow line) |
|  |
| Adapted Wooden Center Pilliar inside the Removed Joint |

From here was the printing of the hand itself. Of which it took three prints before the version used for the actual project. The first of which was printer failure. The second was simply an issue of size. All of the joints were too small so even though some of the hand moved, it was very stiff. Also after that print it was obvious the thumb had to be printed separately. This because it was on a 45º angle and the printer just couldn’t print it like that. The final print in many respects worked. All the fingers and the thumb worked very well. Unfortuantly the thumb joint itself wasn’t perfect. In an effort to give more space in the joints for easy movement, the pillar that was used for the “pincer” motion(bringing the thumb parallel to the hand) was deleted. This meant the joint just slid out. To fix this a small piece of wood was sanded into a circle and inserted into the center of the joint(where the pillar was). This took time but eventually it fit well so a hole was drilled through the entire piece and a bolt was put in to hold it all together. From here the base was printed without issue and the entire hand was done.

Finally the servos could be attached. They can be inserted into the base, and though in the design there were small holes for the screws they were to small and were virtually none existant in the final product. This didn’t turn out to be an issue though because basically all the servos are friction fit in. One of the more frusterating parts of this assignment was setting up the string and elastics. This because it had to be done several times over to find the right length of string, and right amount of stretching in the elastics.

From this point all that had to be done was the code. It is admittedly not overly complex, but its relation to the physical world is very much so. Finding the right values to .write() was a painstaking process because if it was the wrong value and the servos over stressed themselves, it makes a brutal sound and can permanently damage them. As well there are many factors to watch out for, like when trying to bend the thumb, it ends up rotating the entire thumb joint inwards. Despite this in the end the code is rather basic. It ends up essentially just nine prepared functions, that can be called through the serial monitor. These functions use other smaller ones and the .write() statement to move the servos, and in turn the hand, to a desired position. See the chart below for the functions and their discriptions, and see the videos in the media section which illustrate these.

|  |  |
| --- | --- |
| Funcitons | Action |
| Rest() | Hand straight up |
| Fist() | Hand completely compressed |
| Hold() | Hand moves slowly down and thumb moves slowly up to meet it |
| Calculator() | Fist()  Asks the user for an input between 1-31  Displays this input in binary through the fingers |
| Wave() | Moves fingers down and up slowly |
| Salute() | Keeps fingers straight up, but brings thumb bent flat to the palm of the hand |
| Count() | Counts from 1-5 through the fingers |
| Pincer() | Brings the thumb up towards the hand in a pincer motion and back |
| RangeofMotion() | Brings the thumb up towards the hand in a pincer motion and back  Turns the thumb around towards the front of the hand and back  Bends the thumb completely and brings it back to rest |
| Touch() | Touches the thumb and pointer finger together |

## Issues

Issues with ViaCAD

Nearing the end of the project ViaCAD began to crash very often. It was discovered this happened when using the push/pull tool on the thumb. This was frusterating, but once the actual cause of the problem surfaced it was easy to move around it. It was a lesson in using “save as” instead of just *“save”* when using programs like this were crashing is not out of the picture.

Issues with Extension Motions

Finding the right way to extend the fingers was one of the more challenging parts of this project. Initially the idea was to simply have a second set of servos on the back of the hand to do this extension motion. There are two issues with this, firstly it is difficult to have two servos working in perfect tandum and they may end of fighting each other which can be very hard on the servos. The second issue is simply price, by using this method it is essentially doubling the amount of servos needed and in doing so doubling the price.

From here several materials were tested see the chart below.

|  |  |  |
| --- | --- | --- |
| Pros and Cons of Extension Materials | | |
|  | Pros | Cons |
| Elastic   * Elastic Band * Silicone Vessel Loops | * Variable strength * Large range of motion | * Non uniform motion |
| Metal   * Hacksaw blade * Model airplane wire | * Uniform bending * Range of motion | * Tough to bend |
| Wood   * Balsa wood | * Easy to bend * Uniform bending | * Low range of motion(breaks easily) |

|  |
| --- |
|  |

\*Uniform bending = (Both joints in the finger bend equally as oppose to one bending fully and then the other)

\*Silicone Vessel Loops = Short blue elastic bands, “Intended to occlude, retract and idenfity

arteries, veins, tendons and nerves in surgical procedures”

\*From here on the Silicone Vessel Loops will be refered to as elasitcs for simplicity

In the end the elastics were the only real option. The metal was had the potential to be perfect, if only there was one which was easier to bend it would have been perfect.

Issues with Servos

When using the servos two main issues came up. Firstly, the servos don’t always have enough power to pull against the elastics on the back of the hand. Especially since the servos being used are microservos the torque they generate is not very high. The second problem with the servos is the displacement they can create. For example, in my situation the horn is around 2cm is length. So the total displacement that can be produced is around 4cm. However the way the joints work they need 4.5-5cm worth of displacement is the string to compress the finger fully. The unfortunate result of this is that the fingers simply don’t compress completely. This can be fixed to an extent by extending the horn. The problem with this is that since now the horn is longer there is less force being exerted at its end, so even though the displacement problem was fixed they still don’t work because of lack of force.

## Code

// Author: James Corley

// Project: Micro Servo Controlled Robotic Hand(Short ISP)

// Date: March 9 2018

// Status: Working

#include <**Servo**.h> //including servo libarary

//declaring pins to their respective variables

uint8\_t pointerPIN = 12;

uint8\_t middlePIN = 11;

uint8\_t indexPIN = 13;

uint8\_t pinkeyPIN = 10;

uint8\_t T\_INpillarPIN = 7;

uint8\_t T\_OUTpillarPIN = 6;

uint8\_t T\_pincerPIN = 5;

uint8\_t T\_bendPIN = 4;

//CREATING SERVO OBJECTS

//servo objects for the fingers

**Servo** pointerservo; //creating pointer finger object

**Servo** middleservo;  //creating middle finger object

**Servo** indexservo;   //creating index finger object

**Servo** pinkeyservo;  //creating pinkey finger object

//servo objects for the thumb

**Servo** T\_INpillarservo;//thumb pillar rotation(inwards)

**Servo** T\_OUTpillarservo;//thumb pillar rotation(outwards)

**Servo** T\_pincerservo;//pulling thumb towards hand

**Servo** T\_bendservo;//bending thumb

//NON-DIRECT FUNCTIONS

//Moves all fingers the by the given value

void moveallfingers(uint8\_t x)

{

 pointerservo.write(x);

 middleservo.write(x - 40);//since the displacement for this servo 8cm it has to turn less

 indexservo.write(x);

 pinkeyservo.write(x);

}

//contracts thumb fully

void thumbfullcontract()

{

 T\_bendservo.write(130);

 delay(100);

 T\_INpillarservo.write(60);

 T\_OUTpillarservo.write(0);

 T\_pincerservo.write(45);

}

//contracts the thumb halfway

void thumbhalfcontract()

{

 T\_INpillarservo.write(90);

 delay(100);

 T\_OUTpillarservo.write(90);

 T\_pincerservo.write(25);

 T\_bendservo.write(70);

}

//thumb extended in the rest position

void thumbrest()

{

 T\_INpillarservo.write(0);

 T\_OUTpillarservo.write(180);

 T\_pincerservo.write(0);

 T\_bendservo.write(0);

}

//DIRECT FUNCTIONS

//Bends pointer finger and thumb to meet each other in the "ok" sign

void touch() {

 T\_INpillarservo.write(30);

 T\_OUTpillarservo.write(180);

 T\_pincerservo.write(25);

 T\_bendservo.write(105);

 pointerservo.write(180);

}

//counts 1-5 through the fingers

void count()

{

 fist();

 delay(1000);

 pointerservo.write(0);

 delay(1000);

 middleservo.write(0);

 delay(1000);

 indexservo.write(0);

 delay(1000);

 pinkeyservo.write(0);

 delay(1000);

 T\_INpillarservo.write(0);

 T\_OUTpillarservo.write(180);

 T\_pincerservo.write(60);

 T\_bendservo.write(0);

}

//moves thumb up parrallel to the hand and back

//turns thumb joint around to the front of the hand and back

//bends and then rests thumb

void rangeofmotion()

{

 //for loop to cycle through which movement is being done

 for (uint8\_t count = 1; count < 8; count++) {

   for (uint8\_t i = 0; i < 180; i++) {//for loop to control degree rotation

     //if loop to choose which movement is being shown

     if (count == 1) {

       T\_pincerservo.write(i);

     } else if (count == 2) {

       T\_INpillarservo.write(i);

       T\_OUTpillarservo.write(0);

     } else if (count == 3) {

       T\_bendservo.write(i);

     }

     delay(10);//rest for a short time to allow time to admire

   }

   for (uint8\_t i = 180; i > 0; i--) {//for loop to control degree rotation

     //if loop to choose which movement is being shown

     if (count == 1) {

       T\_pincerservo.write(i);

     } else if (count == 2) {

       T\_INpillarservo.write(0);

       T\_OUTpillarservo.write(180 - i);

     } else if (count == 3) {

       T\_bendservo.write(i);

     }

     delay(10);//rest for a short time to allow time to admire

   }

 }

}

void claw()//brings thumb parrallel to hand and back to rest position

{

 //bring thumb parrallel

 for (uint8\_t i = 0; i < 180; i++) {

   T\_pincerservo.write(i);

   delay(10);

 }

 //returns thumb to rest

 for (uint8\_t i = 180; i > 0; i--) {

   T\_pincerservo.write(i);

   delay(10);

 }

}

//moves thumb up parrallel to the hand and back

//turns thumb joint around to the front of the hand and back

//bends and then rests thumb

void rangeofmotion()

{

 //for loop to cycle through which movement is being done

 for (uint8\_t count = 1; count < 8; count++) {

   for (uint8\_t i = 0; i < 180; i++) {//for loop to control degree rotation

     //if loop to choose which movement is being shown

     if (count == 1) {

       T\_pincerservo.write(i);

     } else if (count == 2) {

       T\_INpillarservo.write(i);

       T\_OUTpillarservo.write(0);

     } else if (count == 3) {

       T\_bendservo.write(i);

     }

     delay(10);//rest for a short time to allow time to admire

   }

   for (uint8\_t i = 180; i > 0; i--) {//for loop to control degree rotation

     //if loop to choose which movement is being shown

     if (count == 1) {

       T\_pincerservo.write(i);

     } else if (count == 2) {

       T\_INpillarservo.write(0);

       T\_OUTpillarservo.write(180 - i);

     } else if (count == 3) {

       T\_bendservo.write(i);

     }

     delay(10);//rest for a short time to allow time to admire

   }

 }

}

void claw()//brings thumb parrallel to hand and back to rest position

{

 //bring thumb parrallel

 for (uint8\_t i = 0; i < 180; i++) {

   T\_pincerservo.write(i);

   delay(10);

 }

 //returns thumb to rest

 for (uint8\_t i = 180; i > 0; i--) {

   T\_pincerservo.write(i);

   delay(10);

 }

}

//counts 1-5 through the fingers

void count()

{

 fist();

 delay(1000);

 pointerservo.write(0);

 delay(1000);

 middleservo.write(0);

 delay(1000);

 indexservo.write(0);

 delay(1000);

 pinkeyservo.write(0);

 delay(1000);

 T\_INpillarservo.write(0);

 T\_OUTpillarservo.write(180);

 T\_pincerservo.write(60);

 T\_bendservo.write(0);

}

//Keeps fingers straight up, but brings thumb bent flat to the palm of the hand

void salute()

{

 moveallfingers(0);

 T\_INpillarservo.write(90);

 T\_OUTpillarservo.write(90);

 T\_pincerservo.write(180);

 T\_bendservo.write(160);

}

//contracts all fingers

void fist()

{

 thumbfullcontract();

 delay(700);

 moveallfingers(180);

}

//Moves hand slowly in a pincer motion

void hold()

{

 for (uint8\_t i = 0; i < 45; i++) {

   moveallfingers(i);

   delay(300);

   thumbhalfcontract();

 }

}

//Moves hand to fully extended or "rest"

void rest()

{

 moveallfingers(0);

 thumbrest();

}

//Turns a inputed decimal value to a binary value to be displayed by the hand

void calculator()

{

 fist();

 uint8\_t count = 0;//varibable to confirm if there was serial input

**Serial**.println("Enter a number between 1-31");//asks for a number(1-31 displayable range)

 do {

   if (**Serial**.available() > 0) {          //checks if there is serial input

     count = 1;                           //turns to do-while loop off

     uint8\_t numinput = **Serial**.parseInt();//inputs int from serial monitor

     //for{if{if - convets decimal to binary

     for (uint8\_t i = 0; i < 5; i++) {

       if ((numinput & 0b00000001 << i) > 0) {

         if (i == 0) {

           T\_bendservo.write(0);

           T\_INpillarservo.write(0);

           T\_OUTpillarservo.write(180);

**Serial**.println("1");

         } else if (i == 1) {

           pointerservo.write(0);

**Serial**.println("2");

         } else if (i == 2) {

           middleservo.write(0);

**Serial**.println("4");

         } else if (i == 3) {

           indexservo.write(0);

**Serial**.println("8");

         } else if (i == 4) {

           pinkeyservo.write(0);

**Serial**.println("16");

         }

       }

     }

   }

 } while (count < 1);

}

//Moves all fingers up and down to "wave"

void wave()

{

 for (uint8\_t i = 0; i < 180; i += 5) {

   moveallfingers(i);

   delay(100);

 }

 for (uint8\_t i = 150; i > 1; i -= 5) {

   moveallfingers(i);

   delay(100);

 }

}

void setup() {

**Serial**.begin(9600);

 //ATTACHING SERVOS TO PINS

 pointerservo.attach(pointerPIN);     //pointer finger

 middleservo.attach(middlePIN);       //middle finger

 indexservo.attach(indexPIN);         //index finger

 pinkeyservo.attach(pinkeyPIN);       //pinkey finger

 T\_INpillarservo.attach(T\_INpillarPIN); //thumb pillar inwards

 T\_OUTpillarservo.attach(T\_OUTpillarPIN); //thumb pillar outwards

 T\_pincerservo.attach(T\_pincerPIN);       //thumb to hand

 T\_bendservo.attach(T\_bendPIN);           //bending the thumb

 pinMode(pointerPIN, OUTPUT);

 pinMode(middlePIN, OUTPUT);

 pinMode(indexPIN, OUTPUT);

 pinMode(pinkeyPIN, OUTPUT);

 pinMode(T\_INpillarPIN, OUTPUT);

 pinMode(T\_OUTpillarPIN, OUTPUT);

 pinMode(T\_pincerPIN, OUTPUT);

 pinMode(T\_bendPIN, OUTPUT);

 //COMMANDS

**Serial**.println("Possible commands");

**Serial**.println("1) Rest");

**Serial**.println("2) Fist");

**Serial**.println("3) Hold");

**Serial**.println("4) Calculator");

**Serial**.println("5 ) Wave");

**Serial**.println("6 ) Salute");

**Serial**.println("7 ) Count");

**Serial**.println("8 ) Pincer");

**Serial**.println("8 ) Rangeofmotion");

**Serial**.println("9 ) Touch");

 //to start the hand will always be at rest

 rest();

}

void loop() {

 //INPUT LOOP

 //waits for an input to the serial monitor

 if (**Serial**.available() > 0) {

   //reads string so it can be compared

   String stringinput = **Serial**.readString();

   //if else statment to compare input to preset commands

   if (stringinput.equals("rest")) {

     rest();

   }

   else if (stringinput.equals("fist")) {

     fist();

   }

   else if (stringinput.equals("hold")) {

     hold();

   }

   else if (stringinput.equals("calculator")) {

     calculator();

   }

   else if (stringinput.equals("wave")) {

     wave();

   }

   else if (stringinput.equals("salute")) {

     salute();

   }

   else if (stringinput.equals("count")) {

     count();

   }

   else if (stringinput.equals("pincer")) {

     claw();

   }

   else if (stringinput.equals("rangeofmotion")) {

     rangeofmotion();

   }

   else if (stringinput.equals("touch")) {

     touch();

   }

 }

}

## Media

|  |  |
| --- | --- |
|  |  |
| Outline of Hand with rough Measurements | Development of How to Control the Thumb |
|  |  |
| Thumb Joint Idea | Early Joint Idea |
|  |  |
| Final Thumb Idea | Early Plan to Make Arm with Hand |
|  |  |
| ViaCAD Layout File | Final Hand |
|  |  |
| Final Thumb (Front and Back) | Final Base |
|  |  |
| Printing Prototype #2 | Printing Prototype #2 |
|  |  |
| Printing Final copy | Final Thumb Joint(Rotated to 90°) |
|  |  |
| Final Thumb Joint(Back) | Printing Base |
|  |  |
| Finished Full Hand (Front) | Finished Palm |
|  |  |
| Full Hand(Front) | Full Hand(Back) |
|  |  |
| Full Circuit |  |
| YouTube:  Explanation Video  <https://www.youtube.com/watch?v=2MHkMgNY19U&t=47s>  touch();  <https://www.youtube.com/watch?v=U4K11KeuzU0&index=1&list=PLHzmd7IrM-uOdQXpaqVw7N_7fNR6V9XJZ&t=0s>  count();  <https://www.youtube.com/watch?v=Ab-2lSVonkk&list=PLHzmd7IrM-uOdQXpaqVw7N_7fNR6V9XJZ&index=2>  rangeofmotion();  <https://www.youtube.com/watch?v=xHdriZfR5cI&list=PLHzmd7IrM-uOdQXpaqVw7N_7fNR6V9XJZ&index=3>  calculator();  <https://www.youtube.com/watch?v=sNst4CFDs8M&list=PLHzmd7IrM-uOdQXpaqVw7N_7fNR6V9XJZ&index=4> | |

## Reflection

This was definitely one of my favorite projects done so far. And even though there were some major issues, for the most part, fixing them wasn’t a too frustrating because I could see the end result shaping up. And even though in the end I am disappointed in some respects, overall I am happy with how it turned out. I have always had a passion for art and drawing, and as much as I liked it in the end they were just pictures. No matter how many I made or how much detail was put in them. They could never be real. But to actually have an idea and be able to render it not only in three dimesions but also so that it actually works and moves is a mind-boggling concept and a major part of why I am passionate about this course. In the end this project is one of those things that as a kid would seem like its impossible, so to actually make it is truly amazing.

# Project 13. Printed Circuit Board: ATtiny84 Breakout Board

## Purpose

To use Eagle CAD to produce a small PCB that can be used to teach other ACES about a specific part, in my case this is the ATtiny84.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Description

[http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html - PCB1](http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html#PCB1)

## Theory

The idea of this project was to assist with offboarding. Initially, I was very bad at offboarding and avoided it at all costs. But after using the ATtiny84 for a few projects, it really began to grow on me. Now the ATtiny84 is one of my favourite IC’s simply due to its size and versatility.

For many younger ACES and hobbyists, it is easy to become complacent with all the easy benefits the Arduino provides. However, it is important to maintain the idea that the Arduino is a tool to assist interaction with a certain microcontroller, as oppose being an individual product. So to break this idea, the ACEduino Mini provides a different option in terms of ICs, with the goal of expanding the user's horizon. Instead of being locked into one microcontroller, with this ACES are opened up to the option of using an alternate, and perhaps more efficient method. However, this isn’t an end game. It is meant to assist, not dominate an ACES experience with the ATtiny84, so that after some time with this tool they will be able to offboard with no problem. And if they ever have an issue they can come back and use this to debug.

SPI Communication

|  |
| --- |
|  |

Now to briefly summarize offboard communication. The communication method that almost all microcontrollers use is something called SPI or (Serial Peripheral Interface). It uses four pins called SCLK: Serial Clock, MISO: Master In Slave Out, MOSI: Master Out Slave In, SS: Slave Select(SS is also commonly called RESET). SCLK just outputs a square wave and is used for timing. MOSI is for sending signals out from the master to the slave, and MISO for sending input signals from the slave to the master. SS is just used to choose which slave is being interacted with, it can also be called RESET. This can be used to control many different devices at once, or as used in the project, to just interact with one device. As well it's not just restricted to coding microcontrollers, SPI can interact with many different devices.

Resistor Networks

|  |
| --- |
|  |

Resistor networks are actually very simple parts. They are essentially just many resistors

combined into one to reduce space used. There are two main types, bussed and isolated. Isolated means that it is just several resistors stuck together. There are one input and one output, hence being isolated. Bussed look the same, but instead of having one input and one output, they have several inputs and one output. So for example, if one is a 10 pin, 470, Bussed Resistor network it just means that it has 9 leads with a 470 value, that lead to a single output pin. If that same one was Isolated, however, it would mean that it is essentially just 5 resistors.

-For an explination of the ATtiny84, refer to Project 11.

-For an explination of the 7803 Voltage Regulator, refer to Project 12.

|  |
| --- |
| Parts List |
| ATtiny84 |
| 2 470Ω Resistor Isolated Network |
| Female Header Pins |
| Barrel Jack Input |
| 7805 Voltage Regulator |
| Arduino UNO |
| 470Ω Resistor |
| PCB |
| Sparkfun Reprogrammer |

## Procedure

As said above this PCB is essentially just an Arduino, but for the ATtiny84. It simplifies interaction using three main tools. Firstly, it provides power to the ATtiny84. Either through the reprogrammer(which will be discussed later) or the barrel jack. This barrel jack is connected to a 5V Voltage Regulator (7805) which then supplies power to the ATtiny84. This means that the user has to be careful with what parts they are trying to drive with this project as if too much power is drawn by external parts the ATtiny84 may not run properly. Secondly, it resists all of the I/O pins. Using two isolated resistor networks and one standalone resistor all eleven I/O pins of the ATtiny84 are resisted so the user can use small parts like LEDs without fear of burning them out. And the last function of the this is too have a reprogramming doc. This is one of my favourite features as it streamlines the coding process dramatically. Now, all it requires is plugging in the SparkFun Reprogrammer and the ATtiny84, and coding becomes as easy as with the Arduino.

|  |  |  |
| --- | --- | --- |
| ACEduino Mini Pinout | | |
| GND |  |  |
| 5V |  |  |
| MISO |  |  |
| SCK |  | 5V |
| RESET |  | ?V |
| MOSI |  | GND |
|  |  |  |
| 10 |  | 0 |
| 9 |  | 1 |
| RESET |  | 2 |
| 8 |  | 3 |
| 7 |  | 4 + SCK |
| MOSI + 6 |  | 5 + MISO |

|  |
| --- |
| SparkFun Reprogrammer Pins |

|  |
| --- |
| Power/GND Pins |

|  |
| --- |
| Resisted I/O Pins |

\*Notes:

-All pins with the same name, are tied together, so be more careful when using those pins.

-?V is the raw voltage from the barrel jack. If a 9V battery is plugged in, this voltage will be 9V.

## Media

|  |  |
| --- | --- |
|  |  |
| Side View | Top View |
|  |  |
| DirtyPCB Rendering | Board View |
|  | |
| Schematic View | |

YouTube Video

<https://www.youtube.com/watch?v=aHof0TPoIUM>

## Reflection

This project helped to make me appreciate the full opportunity that I have in the world today. It is amazing that I can produce something in a computer program in Canada. Which will then cause some factory on the other side of the world to buzz to life to build my PCB. And then a week or so later I will have it in my hand. The idea that I have so many resources at my fingertips is mind-boggling. With the 3D printer from the last project, and now this service, the possibility in terms of ISPs and other projects are almost endless. A very exciting fact because no matter how crazy an ISP idea I have, it is almost always possible.

# Project 14. I2C Data Logger

## Purpose

To use I2C communication to interact with a sensor of our choice and store that data on to an external EEPROM IC.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI3M/1718/Tasks.html>

## Theory

I2C Communication

|  |
| --- |
|  |

I2C is a method of communication between one controlling IC, which is referred to as the master,

and any number of tertiary IC’s referred to as slaves. As long as they are I2C parts, one master can control any number of slaves. They can do this because no matter the number of slaves, only two pins are ever needed to allow interaction. Tie these two pins to each IC and the master can interact with any of them. The two pins needed are called SDA and SCL. While SCL simply outputs a clock signal SDA is far more complex. First, the signal through this line contains the desired slave address to first initiate the connection. Generally, each slave has a unique factory-set address, but in some cases, they can have addresses that can be set by the user by presenting different logic signals on the actual IC’s pins. From then the master and slave are connected, so any data can be sent back or forth. Then once the user has finished interacting with the device an end bit is sent out severing the connection.

**Note:** As can be seen in the diagram to the right, there are two pull up resistors from the power source. To have any interaction using I2C, these two resistors are needed. No matter the number of slaves whether it's 1 or 100 they only need these two resistors. This is because the master doesn’t actually output HIGH, it just puts out LOW and these resistors bring that signal HIGH.

|  |  |
| --- | --- |
|  | Structure of SDA and SCL signals |

24LC256

This IC is a external EEPROM(Electronically Erasable Programable Read-Only Memory) chip.

|  |
| --- |
|  |

While the Arduino itself contains some EEPROM, around 1024 bytes, for the purposes of detailed data logging that is not enough. So instead we use the 24LC256 which contains around 32k bytes. To interact through I2C with this device it first requires the address of the chip to be sent out. This address has four hardcoded bits, but there are three that can be set by the user by plugging different logic(HIGH/LOW) into pins A0, A1, A2 of the chip itself. Then a Read/Write bit is sent out to determine if data will be stored or accessed. Now that master has formed its communication with the 24LC256 it can start to actually interact with it. To do this imagine the 24LC256 as an array(which is essentially what it is). First, we need to know the index of where to read or write on the array, this index is made up of two bytes and contains a value between 0-32k(the size of the chip). Then once the location to be written to or read from has been established, 1 byte of data can be stored there, or read from there. Finally, the connection will be ended and the I2C SDA line will be freed up for other communication.

|  |
| --- |
|  |
| 24LC256 Address Where to write on the 24LC256 Data to write |

TC74

The TC74 is an I2C controlled temperature sensor. This part is far more manageable than the

|  |
| --- |
|  |
| TC74 in the TO-220 package |

24LC256. All it does is return a raw temperature value. To access this value as before we first must send out the address of the TC74, unlike the 24LC256 the address is fixed, and cannot be changed by external pin logic. Once the connection is established all that has to be done is to ask for the data in register 0, by simply sending out a 0, and ending the connection. Then we request a value from the sensor and wait to read it. The value that the sensor sends back is the actual temperature at that time.

|  |
| --- |
|  |
| TC74 Address Where to read from TC74 Address Data to be read |

Timer Interupts

A interrupt is a method of timing in C that does not require static code. The classical method of amateur code is to run through lines sequentially until all the lines of code have been completed. So if the user wanted to do timing their only option was to just have a line of code called delay(); that would just do nothing for a given period of time. Interrupts run outside of this classical style. All that they require is a time, at what interval to occur, and a function to be called when they do occur. So lets say a interrupt is supposed to occur every second, and the function it calls simply prints out “Hello”. This interrupt is in code that just does complex mathematical equations in a loop forever. If this code is run the usual part of the code doing the equations will start. However, every second these equations will pause, save wherever they are, and the interrupt will run its code, in this case printing out “Hello”. Once it is finished it will return to exactly where it was before in the equations and continue to do them, until the next second when the next interrupt happens. An important point though, do not make the code in the interrupts function very long or else an interrupt may occur in another interrupt. Instead have some sort of variable that can be triggered to run more complex code outside of the interrupt.

## Procedure

|  |
| --- |
| Parts List |
| Arduino UNO |
| 24LC256 |
| TC74 |
| RTC |
| |  | | --- | | Traffic Light | | 3 LEDs (1 Red, 1 Green, 1 Yellow | | Custom PCB | | 220 Ω Resistor | |
| Connecting Wire |

This code is slighty complex as it runs off of three different files. So this section will be split into one for each of them.

EEPROM\_Store\_Data

This file writes the temperature data from the TC74 on to the EEPROM chip. The code starts declaring the timer interrupt in the setup()function. Then it launches straight into the loop() function, however the loop can only be run if the interrupt becomes active, so the code just loops until the first second when the interrupt goes off. When it goes off the pulse() method is called. Since I only want to record a value every minute, there is a count variable that adds one each time pulse is called. Once this value reaches 60 that means that 60 interrupts have occurred and since each interrupt is 1 second that means 60 seconds or a full minute had passed. Once this happens a boolean variable will be set to high and pin 13 will be toggled. Now because the boolean was finally triggered, the loop can actually run its code. Now the first thing to do is to make sure the code hasn’t run more than 120 times(120 minutes makes 2 hours, the desired run time for the code) , if it’s bellow that add one to the timer variable, and return the boolean trigger value back to false. Then the code interacts with the TC74(theory section for detail) and returns the temperature value. Then a connection is opened with the 24LC256(theory section for detail). The timer variable that is keeping track of how many times this code has run, and how long it will run for becomes the location to write to on the EEPROM chip. And the temperature value that has just been recorded is the data that will be written. Because of the timer variable, this code will repeat these steps for the allotted period of time.

External\_EEPROM\_READ

This sketch is very simple. It just runs through a loop a given number of times(according to the number of inputs in the EEPROM) each time it reads the value from the 24LC256(theory section for detail) and outputs it on to the serial monitor.

EEPROM\_Tea\_temp

Similarly to EPROM\_Store\_Data this starts off by setting up the timer interrupt. As well the loop is the same, it will only be activated if triggered by the pulse function so for the actual code to start we have to wait for the first interrupt. This is where the code separates from the previous sketch, instead of running once a minute now it runs once a second so the count variable can be removed. Because of this every time pulse is called the code runs. So upon calling the trigger variable is set to true and the loop function is accessed. The loop code starts by recording the temperature from the TC74. If this value is 10 degrees or above the perfect value the red LED will turn on. If its just between 10 degrees and the perfect value the yellow LED will turn on. And it it’s the perfect value, or 15 degrees below the perfect value the green LED will turn on. And if it's below 15 degrees below the perfect value all the LEDs will turn on. Moreover, it also compares every value inputted by the TC74, to all the data on the 24LC256 EEPROM chip. If it finds the same value inputted there it takes the time that that value has read(by checking the current index) and makes that into a variable called currentime. Then the location of the perfect drinking temperature is found and its time is recorded under the perfecttime variable. Then these two values are subtracted from each other to find out the time until the cup of tea will be the perfect temperature. This loop has no timer, and will run indefinitely.

## Code

//////////////////////EEPROM\_Store\_Data//////////////////////////////

// Author: James Corley

// Project: I2C Data Logger

// Date: May 12 2018

// Status: Working

//defining addresses and values+including libraries

#include <Wire.h>

#define RTCADDRESS 0x68

#define TC74\_ADDRESS 0x48

#define EEPROM\_ADDRESS 0x50

#define SQWE 4

#define CONTROLEREG 7

#define INTO 2

#define RS0 0

#define RS1 1

#define Hz1 0

#define Hz4 1<<RS0

#define Hx8 1<<RS1

#define Hz32 (1<<RS1)|(1<<RS0)

volatile uint16\_t count = 0;

volatile bool triggered = false;

uint8\_t seconds;

uint16\_t timer = 0;

void setup() {

**Serial**.begin(9600);

 Wire.begin();

//setting up RTC for Interupts

 Wire.beginTransmission(RTCADDRESS);

 Wire.write(CONTROLEREG);

 Wire.write((1 << SQWE) | Hz1);

 Wire.endTransmission(RTCADDRESS);

 DDRD = 0;

 PORTD |= (1 << PD2); //enable pullup resistor(20k)

 DDRB |= (1 << PB5);

 PORTB |= (1 << PB5);

//attaching interupt

 attachInterrupt(digitalPinToInterrupt(INTO), pulse, FALLING);

}

void pulse() {

 count++;

 if (count == 60) {

   count = 0;

   triggered = true;

   PORTB ^= (1 << PB5);

 }

}

void loop() {

 if (triggered && timer < 120) {

   triggered = false;

   timer++;

   Wire.beginTransmission(TC74\_ADDRESS); //Which I2C device

   Wire.write(0); //Where to read from

   Wire.endTransmission();

   Wire.requestFrom(TC74\_ADDRESS, 1); //Requesting Data

   while (!Wire.available()); //Waiting for Data

   uint8\_t temp = Wire.read(); //Resiving Data

**Serial**.println(temp);

   Wire.beginTransmission(EEPROM\_ADDRESS); //Which I2C device

   Wire.write((int)(timer >> 8));

   Wire.write((int)(timer & 0xFF));

   Wire.write(temp);

   Wire.endTransmission();

 }

}

///////////////////// EEPROM\_Tea\_temp //////////////////////////

// Author: James Corley

// Project: I2C Data Logger

// Date: May 12 2018

// Status: Working

#include <Wire.h>

#define RTCADDRESS 0x68

#define TC74\_ADDRESS 0x48

#define EEPROM\_ADDRESS 0x50

#define SQWE 4

#define CONTROLEREG 7

#define INTO 2

#define RS0 0

#define RS1 1

#define Hz1 0

#define Hz4 1<<RS0

#define Hx8 1<<RS1

#define Hz32 (1<<RS1)|(1<<RS0)

volatile uint16\_t count = 0;

volatile bool triggered = false;

uint16\_t timer = 0;

uint8\_t redpin = 11;

uint8\_t yellowpin = 12;

uint8\_t greenpin = 13;

uint8\_t perfecttemp = 58;

uint8\_t perfecttime;

uint8\_t currenttime;

void setup() {

**Serial**.begin(9600);

 Wire.begin();

 Wire.beginTransmission(RTCADDRESS); //Which I2C device

 Wire.write(CONTROLEREG);

 Wire.write((1 << SQWE) | Hz1);

 Wire.endTransmission(RTCADDRESS);

 DDRD = 0;

 PORTD |= (1 << PD2); //enable pullup resistor(20k)

 DDRB |= (1 << PB5);

 PORTB |= (1 << PB5);

 attachInterrupt(digitalPinToInterrupt(INTO), pulse, FALLING);

}

void pulse() {

 count = 0;

 triggered = true;

}

void loop() {

 if (triggered) {

   triggered = false;

   Wire.beginTransmission(TC74\_ADDRESS); //Which I2C device

   Wire.write(0);

   Wire.endTransmission();

   Wire.requestFrom(TC74\_ADDRESS, 1); //requesting temperature data

   while (!Wire.available()); //waiting for data

   uint8\_t temp = Wire.read(); //reading temperature data

**Serial**.print("Temperature: ");

**Serial**.println(temp);

   PORTB = 0; //reseting LEDs

//setting LEDs to respective colour according to temperature

   if (temp > perfecttemp + 10) PORTB = (1 << PB3);

   else if (temp > perfecttemp)PORTB = (1 << PB4);

   else if (temp <= perfecttemp && temp > perfecttemp-15)PORTB=(1 << PB5);

   else if (temp < perfecttemp - 15) {

     PORTB = (1 << PB5);

     PORTB = (1 << PB4);

     PORTB = (1 << PB3);

   }

   for (int i = 0; i < 120; i++) {

     Wire.beginTransmission(EEPROM\_ADDRESS); //Which I2C device

     Wire.write((int)(i >> 8)); //Where to read from on EEPROM

     Wire.write((int)(i & 0xFF)); //Where to read from on EEPROM

     Wire.endTransmission();

     Wire.requestFrom(EEPROM\_ADDRESS, 1); //requesting data

     uint8\_t EEPROMdata;

     while (!Wire.available());

     EEPROMdata = Wire.read(); //reading data

     if (temp == EEPROMdata) {

       currenttime = i;

     }

     if (perfecttemp == EEPROMdata) {

       perfecttime = i;

     }

   }

**Serial**.print("You Beverage will be ready in:  ");

**Serial**.println(perfecttime - currenttime);

 }

}

///////////////////////// External\_EEPROM\_READ//////////////////////////

// Author: James Corley

// Project: I2C Data Logger

// Date: May 12 2018

// Status: Working

#include <Wire.h>

#define EEPROM\_ADDRESS 0x50

volatile bool triggered = false; //variable to react with interupts

uint16\_t eepromlocation = 0; //location in the EEPROM chip

void setup() {

**Serial**.begin(9600);

 Wire.begin();

}

void loop() {

 if (eepromlocation < 117) {

   eepromlocation++;

   Wire.beginTransmission(EEPROM\_ADDRESS); //Which I2C device

   Wire.write((int)(eepromlocation >> 8)); //Where to read from on EEPROM

   Wire.write((int)(eepromlocation & 0xFF)); //Where to read from on EEPROM

   Wire.endTransmission();

   Wire.requestFrom(EEPROM\_ADDRESS, 1); //Requesting Data

   int EEPROMdata;

   while(!Wire.available()); //waiting for data

EEPROMdata = Wire.read(); //Reading Data

**Serial**.print("location: ");

**Serial**.println(eepromlocation);

**Serial**.print("temp: ");

**Serial**.println(EEPROMdata);

 }

}

## Media

|  |  |
| --- | --- |
|  |  |
| Overall Circiut | Wiring + underneath RTC |
|  |  |
| Circiut Working | Close Up |
|  | |
| Serial Plotter Data | |

YouTube Video:

<https://www.youtube.com/watch?v=igkZtWcGLm0&t=3s>

## Reflection

Overall, though this is a busy period of time this ended up being a fun project. Because believe it or not hot beverages going cold is actually a problem for me. About once a week a cup of tea or some other beverage gets forgotten and goes cold. So to make something that was related to, and could hopefully stop this trend was quite entertaining. And though it took some thinking, and writing, with all the new concepts introduced overall they worked well. By the end of the project, I was happy with the result. I had a goal and I completed it fully, something that doesn’t always happen.

# Project 15. Textwriter

## Purpose

To create a small textwriting machine. It should have a keyboard input system, and it should us a SD card to store and transfer files.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI3M/1718/ISPs.html>

## Acknowledgments

Kreher Fiset-Algarvio – 3-D Printing the Box and Top

## Theory

Voltage Divider

A voltage divider is a set of two resistors in series, in the order PWR, resistor 1(R1), resistor

|  |
| --- |
| /Users/student/Downloads/VoltageDivider.png |

2(R2), GND. The idea is that in between resistor 1 and 2 the voltage will be split depending on the values of R1 and R2. As an example R1 = R2 then the voltage output in the middle will be exactly half the voltage input. Now, this can be used in many ways but one of the most common will be to analogRead the middle value, to use the voltage divider as a form of input to the Arduino. Tools like potentiometers are very useful for this.

Keyboard

The keyboard was designed specifically for this project. It requires only two Arduino analog input pins and should be able, with the assistance of some code, to be able to display every symbol on the common keyboard. It does this through a system of buttons(keys) tied each to their own individual resistor. This keyboard works by providing a voltage divider between the resistor unique to the key(R1) and a base resistor(R2) of 1kΩ. Then a analogread pin from the Arduino is tied in the middle of this voltage divider. The result being, whenever a key is pressed a value unique to that key(and its respective resistor) will be read in by the machine. Then all the code has to do use this inputted value to find the respective symbol which is hardcoded in an 2D array(this key is only able to move through in one dimension through the 2D array). However, there is a second digital pin called shiftVal. . If shift is not pressed and LOW is presented, then the keyVal pin will look through the 2D array in the first dimension. However, if shift is HIGH then it will switch and look through the second dimension which just contains the same letters as the first but the capital version(or for symbols whatever is the second symbol).

SD Card.

Now raw interaction is complex, as it is almost always done by computers. SD cards were not

|  |
| --- |
| /Users/student/Downloads/254-03.jpg |

designed with prototyping in mind, but with the assistance of the SD library, and an Adafruit SPI SD breakout board, communication actual becomes relatively simple. The way data is stored is primarily through a two-layer system of >directory>file>data. Though for for more complex purposes directories can be used, for the simple function of text documents that layer of complexity can be bypassed

Note: The maximum length of a filename is 8 characters long. If it’s a text file, this name must end in .txt(the .txt is not part of the 8)

Note: For some reason, the use of SD or File functions CAN NOT be done outside of a single method. For example, having one method called begin(){SD.begin(10)} and then calling begin(); and then running any other SD or File functions like file.open(); won’t work. Interaction with the SD has to be done in a single method and ended in that method.

LCD

To start there are two main types of LCD screens. Text-based and graphics based. And the

|  |
| --- |
| /Users/student/Downloads/LCD-Display-Tutorial.png |

purposes for each are rather self-explanatory. Graphics base displays allow specific pixel manipulation, the result of which is the creation of pictures, or if the user is determined enough, text as well. However, the more simple option for text is called a text-based LCD. These screens, unlike the graphic displays, do not allow pixel manipulation of the entire screen. Instead, there are 5x8 sections of “pixels” spaced out across the screen. These segments are the location that each individual character can be written out to on the screen. When a character is written into a location on the LCD the cursor then moves to the right by one space to write another character.

Strings in Arduino C

Strings are one of the less developed parts of Arduino C. A string, in this language, is just an array of chars that ends in the null char. So an important note of this, a string consisting of a single character, “a” for example, is not the same as ‘a’. This is because “a” is actually string[‘a’,’null’]. There are a few tools that we can use however to help us when using strings.

Char Array to String

Functions: StringVar.charAt(location); StringVar.setCharAt(location,character);

are two of the most useful tools. This allows the user to know what char is at a given location. And then if need be to change the char at that location. This is the only way to turn an array of chars into a string. All that’s needed is a buffer string(a string that will be converted into an actually useful string) and an array of chars. Then a for-loop to go through the array and the buffer string, at each location it buffer.setCharAt(i,charArray[i]);.

String to Char Array

Functions: StringVar.toCharArray(charArray[],length of charArray)

Now, this is far easier to use because there is an actual function already specifically designed to accomplish this task.

Storage

There are three levels of storage in this project, Micro SD, EEPROM(internal), and Program Memory. Each with their own benefits and drawbacks. Program memory is the easiest to interact with, however, it is easily filled up, and once the program ends it will not be saved(unless it’s a hardcoded value). EEPROM is slightly more difficult to interact with, as each EEPROM location can only have one byte written to it, so saving things like strings are more difficult. However, this storage is “non-volatile” memory, meaning even after the program ends and the Arduino is disconnected from power it stays stored. Even though overall this has less storage than the program memory because none of it is used by other processes it often ends up to have more storage than the Program memory(external EEPROM chips can be added if need be). The last storage source is the Micro SD card. It has the largest storage by a long shot, and can easily interact with external devices. However, when it comes to actually interacting with the Arduino it is by far the most difficult. It should only be used for mass data storage or for data that is needed by other high-level devices(phone, computer, etc).

|  |
| --- |
| Parts List |
| 2x16 LCD |
| 4x20 LCD |
| Adafruit SD breakout Board |
| Micro SD Card |
| Jumper Wire |
| Arduino UNO |
| Case and Top(3D printed) |
| 2 10k Ω Potentiometers |
| 7805 Voltage Regulator |
| 2 220 Ω Resistors |

## Procedure

The new parts used in this project, despite being temperamental, are not overly complex to interact with.

However the system that is used to turn all of those individual parts very much is. So to start a password is asked of the user using the userinput method. Once the user has inputted a password and hit enter it will compare the inputted password to a saved internal password. If the password is correct then a menu screen will be outputted to the main LCD. Three options OpenDoc, MakeDoc, and DeleteDoc. To interface with all the screens presented to the user the arrow keys and the enter button are used. To choose any of these options, simply move the cursor around the screen to that option and hit enter.There are several main ideas that could be in the theory section, but since they are unique to this project they will be discussed here.

Basic Code Structure

This codes actual running capabilities rely on a single switch-case statement inside the Arduino's loop(). This statement runs off of a variable called marker, which corresponds to whichever screen is open at that time. So for example marker = 0 corresponds to having the menu open, 1 to having a document open, 2 to making a document and 3 to deleting a document. Now within this switch-case are all the calls to the desired functions that actually make switch-case do anything. The marker variable is changed internally during its specific case so that by the time the function loops back it will enter a different segment of the code. As an example when the menu function is run, meaning marker = 0, and the three options are provided to the user if the user chooses to make a document the marker will be set to 2, so that by the time the code loops and comes back to the beginning of the switch-case, instead of entering and executing the menu function again it will go into case 2:, to create a document.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Password(0) | |  |
|  | Checks if password is correct.  If its not repeat, if it is move on to Menu | |  |
|  | Menu(0) | |  |
| User hits ENTER on a chosen function. | | | |
| MakeDoc(2) | OpenDoc(1) | | Delete Doc(3) |
| Asks for user to name document | Lists Documents | | Asks for user to name the document to delete. |
| Saves Document | Hit ENTER on desired document to open it | User hits x |
| Opens Document | |  | Deletes Document |
| User interacts with document | |  |  |
| User hits x | | |  |
| Return to Menu(0) | | | |

|  |  |
| --- | --- |
| passwordRun() function (marker = 0) | openDoc() function (marker = 1)   * listFiles() function(docOpen = false) * scroll() function(docOpen = true) |
| deleteDoc() function (marker = 3) |
| makeDoc() function (marker = 2) |
| menu() function(marker = 0) |  |

Moving the cursor

Moving the cursor was a major part of this assignment and a job that the LCD to not make particularly easy. There is a lcd.setCursor(xaxis,yaxis) function that can be used to set the location of the LCD’s cursor on the screen, however, there is no way to know the location of the cursor. As well when characters are inputted across the screen and it reaches the end of the screen, it does not automatically move to the next line. Instead, it goes to the 1st, 3rd, 2nd, 4th(the way the data is stored on the LCD) lines. To fix these problems I created a new method called moveCursor(xaxis,yaxis,lcd). This method essentially is the same as the lcd.setCursor(xaxis,yaxis), but instead, it saves the new location of the cursor in x and y variables. Moreover, these two variables are used to store all the data in a 2D array. Whenever a character is written to the screen, it also writes it(using its current x and y locations) to a 2D array which can then be written to an SD card.

### Important Functions

readkey();

readkey(); easily one of the most complex yet useful functions of the code. It is solely responsible for interpreting signals from the keyboard and saving the values into the writetext[][] 2D array. This function first reads in the analogRead value from the keyboard and converts that raw input into the variable letter. If this value is smaller than the charArray’s(the2D array holding every character on the keyboard) size then it simply uses that variable to find its corresponding character in the array( charArray[letter][shift]). It then sets that to the writetext[x][y] array and prints it out to the LCD. However, if letter is larger than charArray[][]’s size, then it is one of five possible commands, cursor up, cursor down, cursor left, cursor right, or ENTER. It determines this by using another switch-case statement, with letter as the variable. For the directions it is relatively simple, just adding one to either the x or y variables and then updating the cursor's location. The ENTER button is far more complex though. ENTER is so problematic because it means different things at different locations on the screen, and on different screens(example, on the makeDoc function it means the name is finished and to save a document under the name just inputted, but on the openDoc function it can mean, scroll up, down, close the document or even just move to the next line down.)

userinput(String phrase);

this function is designed to output a phrase to the user which will prompt the user to input an eight character string. This function reads the characters into the writetext[][] array while also converting the inputted characters into a string using the Char array to String method discussed in the theory section. It then clears the screen and returns the inputted string.

eepromreadbyte(uint8\_t line);

this function is used to read a byte(of bytes) from eeprom's internal storage. Similar to userinput(); it takes this array of chars and converts them into a string, which it returns. The line parameter is used to determine which byte(of bytes) will be read.

scroll(uint16\_t pagenum);

Scroll is the function in charge of actually reading from the SD card and displaying that to the user. Its secondary function is to also list the files as part of the viewdocs() function. If the marker is set to 1 it will perform the former, and a marker value of 2 will execute the latter.

writetodoc();

Uses readkey(); to read in up to 76 characters and then prints them into the current file open on the SD card.

opendoc();

Initially this function was supposed to physically open a document on the SD card. However, since the documents must be opened and closed in a single function that plan failed. So now this function merely confirms a connection as well as setting up the LCDs to be written to, giving the illusion of opening a document. It sets up the titlelcd and the side option bar.

movecursor(uint8\_t xloc, uint8\_t yloc, String whichlcd);

As discussed above this function simply moves the cursor and saves its new location in the x and y variables.

passwordRun();

A simple function which asks the user for a password, if the user enters the password correctly it will pass through the function without a problem. If the user fails to enter the password incorrectly though it will loop infinitely until they do. It does this by using the userinput(); function to get a string value from the user and then comparing that using .equals()to a hardcoded password value.

### Variables and Arrays

|  |  |
| --- | --- |
| Array Name | Function |
| displayarray[a][b]  a=[lcdwidth-1]  b=[lcdheight] | This array is purely for outputting to the LCD and saving to the SD. It has all its values set by the writetext[][] array, so it doesn’t worry about anything but printing its values to the LCD. All it does is act as a middleman between the writetext[][] array and the LCD and SD. |
| writetext[a][b]  a=[lcdwidth-1]  b=[lcdheight] | This is used to store the data on the screen(it uses the x and y location variables when it stores data. So the character at x and y on the LCD is the same as the character at x and y in the writetext[][] array). It never actually interacts with the user, however, it only writes its own values to the displayarray[][]. |
| charArray[a][b]  a=number of character keys on the keyboard  b=2(if the shift key is being pressed or not) | This array simply holds the characters for each key. Whenever a value is read in from the keyboard it(after a mathematical conversion) is used to as the index to find get its corresponding character. The second layer of this array holds the secondary key value(if the user is pressing shift it will go to this layer). |

|  |  |  |
| --- | --- | --- |
| Variable Name | Type | Variable Function |
| docOpen | Boolean | To determine if there is a document open at the time. HIGH if there is LOW if there is not |
| hitenter | Boolean | To determine if the user has hit the enter key or not. HIGH if it has, LOW if it hasn’t |
| marker | uint8\_t | Used to determine which screen is currently open. 0-menu, 1-document or listing files, 2-making document, 3-deleteing document |
| password | String | A string holding the password |
| pagenum | uint8\_t | A very important variable holding the vertical location in a file. For example, when listing the documents, only 4 files can be listed at a time. Any more must be stored on the next “page”. So this variable keeps track of which “page” the user is on. |
| currentfile | String | The name of file currently open(without .txt) |
| x | uint8\_t | The x location on the LCD |
| y | uint8\_t | The y location on the LCD |

## Code

// Author : James Corley

// Project: Textwriter(Medium ISP)

// Date: May 26 2018

// Status: Working

#include <**EEPROM**.h>

#include <**LiquidCrystal**.h>

#include <**SD**.h>

File myFile;

const int rss = 7, enn = 6, dd4 = 5, dd5 = 4, dd6 = 3, dd7 = 2;

**LiquidCrystal** titlelcd(rss, enn, dd4, dd5, dd6, dd7);

const int rs = 9, en = 8, d4 = A2, d5 = A3, d6 = A4, d7 = A5;

**LiquidCrystal** lcd(rs, en, d4, d5, d6, d7);

#define arraysize (sizeof(charArray)/2)

#define shiftpin  A1

#define keyvalpin  A0

#define lcdwidth  20

#define lcdheight  4

#define titlelcdwidth 16

#define titlelcdheight  2

boolean docOpen = false;

uint8\_t marker = 0;

\*

  marker

  0 menu screen

  1 doc is open

  2 listing documents

  3 delete document

  4 user input

\*/

boolean hitenter = false;

String password = "aaa";

uint8\_t pagenum = 0;

String currentfile;

uint8\_t x = 0;

uint8\_t y = 0;

char writetext[lcdwidth - 1][lcdheight];

char displayarray[lcdwidth - 1][lcdheight];

char charArray[][2] = {

 {'a', 'A'}, {'b', 'B'}, {'c', 'C'}, {'d', 'D'}, {'e', 'E'}, {'f', 'F'},

 {'g', 'G'}, {'h', 'H'}, {'i', 'I'}, {'j', 'J'}, {'k', 'K'}, {'l', 'L'},

 {'m', 'M'}, {'n', 'N'}, {'o', 'O'}, {'p', 'P'}, {'q', 'Q'}, {'r', 'R'},

 {'s', 'S'}, {'t', 'T'}, {'u', 'U'}, {'v', 'V'}, {'w', 'W'}, {'x', 'X'},

 {'y', 'Y'}, {'z', 'Z'},

 {'1', '!'}, {'2', '@'}, {'3', '#'}, {'4', '$'}, {'5', '%'}, {'6', '^'},

 {'7', '&'}, {'8', '\*'}, {'9', '('}, {'0', ')'}, {'-', '\_'}, {'=', '+'},

 {'[', '{'}, {']', '}'}, {(char)92, '|'}, {';', ':'}, {(char)44, '"'},

 {',', '<'}, {'.', '>'}, {'/', '?'}, {'`', '~'}, {' '}

};

//CUSTOM CHARACTERS

byte xclose[8] = {

 B11111,

 B00000,

 B10001,

 B01010,

 B00100,

 B01010,

 B10001,

};

byte full[8] = {

 B11111,

 B11111,

 B11111,

 B11111,

 B11111,

 B11111,

 B11111,

};

byte up[8] = {

 B00100,

 B01110,

 B11111,

 B11111,

 B11111,

 B11111,

 B11111,

};

byte down[8] = {

 B11111,

 B11111,

 B11111,

 B11111,

 B11111,

 B01110,

 B00100,

};

//MODIFIED COMMANDS

void setlcd(String todisplay, String choselcd)//a modified function to clear the lcd and print a desired subject on it

{

 movecursor(0, 0, choselcd);

 if (choselcd == "lcd")

 {

   lcd.clear();

   lcd.print(todisplay);

 } else if (choselcd == "titlelcd")

 {

   titlelcd.clear();

   titlelcd.print(todisplay);

 }

}

//a modified function for printing to the lcd

//it accounts for reaching the end of lcd screen, and moving the cursor after something is printed

void lcdprint(char subject) {

 lcd.print(subject);

 if (x == (lcdwidth - 1)) {

   x = 0;

   y++;

   //movecursor(x, y, "lcd");

 } else {

   x++;

 }

 movecursor(x, y, "lcd");

}

//modified function to track cursor location in x and y variables

void movecursor(uint8\_t xloc, uint8\_t yloc, String whichlcd)

{

 x = xloc;

 y = yloc;

 char placeholder;

 lcd.cursor();

 if (whichlcd == "titlelcd")

   titlelcd.setCursor(x, y);

 if (whichlcd == "lcd")

   lcd.setCursor(x, y);

}

//DISPLAY

//asks the user to enter a password, if it wrong, function loops, if not the user passes through

void passwordRun()

{

 setlcd("Password", "titlelcd");

 do {

   lcd.clear();

 } while (!password.equals(userinput("Password")));

 marker = 0;

}

//prints out the menu options to the user

void menu()

{

 setlcd("MENU", "titlelcd");

 movecursor(0, 0, "lcd");

 lcd.print("1)OpenDoc");

 movecursor(0, 1, "lcd");

 lcd.print("2)MakeDoc");

 movecursor(0, 2, "lcd");

 lcd.print("3)DeleteDoc");

 movecursor(0, 0, "lcd");

 do {

   readkey();

 } while (marker == 0);

}

//sets the scroll buttons and the close button on the left of the screen

void setscreen() {

 movecursor(0, 0, "lcd");

 lcd.write(byte(0));

 movecursor(0, 1, "lcd");

 lcd.write(byte(2));

 movecursor(0, 2, "lcd");

 lcd.write(byte(1));

 movecursor(0, 3, "lcd");

 lcd.write(byte(3));

 movecursor(1, 0, "lcd");

}

//DOC COMMANDS

//asks the user to input 8 characters, and turns it into a string

String userinput(String phrase) {

 marker = 4;

 char inputarray[10];

 String convertedchar = "          ";

 movecursor(0, 0, "lcd");

 lcd.print(phrase);

 movecursor(0, 1, "lcd");

 lcd.print("\_\_\_\_\_\_\_\_");

 movecursor(0, 1, "lcd");

 for (int i = 0; i < 8; i++)

 {

   readkey();

   inputarray[i] = writetext[x][y];

   convertedchar.setCharAt(i, inputarray[i]);

   if (hitenter) {

     convertedchar = convertedchar.substring(0, i--);

     i = 10;

   }

 }

 hitenter = false;

 lcd.clear();

 return convertedchar;

}

void makedoc()

{

 currentfile = userinput("Make Document");

 for (int i = 1; i <= 8; i++) {                //saves file name to eeprom

**EEPROM**[i + (**EEPROM**[0] \* 8) ] = int(currentfile.charAt(i - 1));

 }

 docOpen = true;

 marker = 1;

**EEPROM**[0]++;

}

String eepromreadbyte(uint8\_t line)

{

 String convertedchar = "        ";

 for (int i = 0; i < 8; i++)

 {

   if (**EEPROM**[i + (8 \* line) + 1] > 32) {

     convertedchar.setCharAt(i, char(**EEPROM**[i + (8 \* line) + 1]));

   } else {

     convertedchar = convertedchar.substring(0, i);

     i = 8;

   }

 }

 return convertedchar;

}

//prints the document names out to the user and waits until the

//user chooses one

void viewdocs()

{

 uint8\_t currentpage = pagenum;

 setlcd("Listing Docs...", "titlelcd");

 marker = 2;

 setscreen();

 scroll(pagenum);

 do {

   readkey();

 } while (marker == 2 && currentpage == pagenum);

 //pagenum = 0;

 if (currentpage == pagenum) {

   docOpen = true;

 }

 marker = 1;

 movecursor(1, 0, "lcd");

}

//opens a document and prepares it to be written to or read from

void opendoc()

{

 docOpen = true;

**SD**.begin(10);

 myFile = **SD**.open(currentfile + ".txt", FILE\_WRITE);

 if (myFile) {

   //Serial.print("here2");

   setlcd(currentfile + " " + myFile.size() + "B", "titlelcd");

   //titlelcd.print(currentfile + " " + myFile.size() + "B");

   movecursor(0, 1, "titlelcd");

   movecursor(0, 0, "lcd");

   setscreen();//prepares left option bar

   myFile.close();

 }

}

/sets openDoc to close, clears the screen and returns to menu

void closedoc()

{

 lcd.clear();

 marker = 0;

 docOpen = false;

}

//deletes a document from the SD

void deletedoc(String filename)

{

**SD**.remove(filename);

}

//reads from a document the inputted page

void scroll(uint16\_t pagenum) {

 myFile.close();

 if (marker == 1) {

**Serial**.print("currentfile before scroll:");

**Serial**.println(currentfile);

**SD**.begin(10);

   myFile = **SD**.open(currentfile + ".txt", FILE\_WRITE);

**Serial**.println(myFile.name());

//76 character is one page, so to find the correct location

   myFile.seek(76 \* pagenum);

   for (uint8\_t t = 0; t < lcdheight; t++) {

     for (uint8\_t i = 1; i < lcdwidth; i++) {

       if (myFile) {

         displayarray[i][t] = myFile.read();

         lcd.print(displayarray[i][t]);

**Serial**.println(displayarray[i][t]);

       } else {

         i = lcdwidth;

         t = lcdheight;

       }

     }

     movecursor(1, y + 1, "lcd");

   }

   movecursor(1, 0, "lcd");

   myFile.close();

 } else if (marker == 2) {

   //if the user is making a document

   movecursor(1, 0, "lcd");

   for (uint8\_t i = 0; i < **EEPROM**[0]; i++) {

     lcd.print(eepromreadbyte(i));

     y++;

     movecursor(1, y, "lcd");

   }

 }

}

//reads from a document the inputted page

void scroll(uint16\_t pagenum) {

 myFile.close();

 if (marker == 1) {

**Serial**.print("currentfile before scroll:");

**Serial**.println(currentfile);

**SD**.begin(10);

   myFile = **SD**.open(currentfile + ".txt", FILE\_WRITE);

**Serial**.println(myFile.name());

   myFile.seek(76 \* pagenum);//76 character is one page, so to find the correct location

                             //to start reading from just use 76 \* pagenum

   for (uint8\_t t = 0; t < lcdheight; t++) {

     for (uint8\_t i = 1; i < lcdwidth; i++) {

       if (myFile) {

         displayarray[i][t] = myFile.read();

         lcd.print(displayarray[i][t]);

**Serial**.println(displayarray[i][t]);

       } else {

         i = lcdwidth;

         t = lcdheight;

       }

     }

     movecursor(1, y + 1, "lcd");

   }

   movecursor(1, 0, "lcd");

   myFile.close();

 } else if (marker == 2) {

   //if the user is making a document

   movecursor(1, 0, "lcd");

   for (uint8\_t i = 0; i < **EEPROM**[0]; i++) {

     lcd.print(eepromreadbyte(i));

     y++;

     movecursor(1, y, "lcd");

   }

 }

}

//reads in a input from the keyboard and interprets it.

void readkey()

{

 delay(400);

 while (analogRead(keyvalpin) == 0);

 uint8\_t shift = 0;

 uint16\_t shiftinput = analogRead(shiftpin);

 uint16\_t raw = analogRead(keyvalpin);     //read in keyVal

 uint16\_t letter = raw / 10;               //turn raw in usable value letter

**Serial**.print("The Letter Value is: ");

**Serial**.println((charArray[letter][shift]));

**Serial**.print("letter=");

**Serial**.println(letter);

 if (letter < arraysize) {

   if (shiftinput > 0) {

     shift = 1;//if shift is pressed set shift to 1

   }

   //print inputted letter on to the lcd and into writetext[][] to store it

   lcdprint(charArray[letter][shift]);

   writetext[x][y] = (charArray[letter][shift]);

**Serial**.print("x = ");

**Serial**.println(x);

   //chosen2Darray[x][y] = (charArray[letter][shift]);

 } else {

   switch (letter) {

     case arraysize :

     case arraysize+1 :

       //up

       y--;

**Serial**.println("up");

       break;

     case arraysize+2 :

     case arraysize+3 :

       //down

       y++;

**Serial**.println("down");

       break;

     case arraysize+4 :

     case arraysize+5 :

       //left

       x--;

**Serial**.println("left");

       break;

     case arraysize+6 :

     case arraysize+7 :

       //right

       x++;

**Serial**.println("right");

       break;

case arraysize+8 :

     case arraysize+9 :

     case arraysize+10 :

     case arraysize+11 :

     case arraysize+12 :

       //ENTER

**Serial**.println("ENTERPRESS");

       switch (marker) {

         case 0:

           //sets the marker to whicher menu option the user choose

           marker = y + 1;

**Serial**.println(marker);

**Serial**.print("x");

**Serial**.println(x);

**Serial**.print("y");

**Serial**.println(y);

           break;

         case 2:

           if (x > 0) {

             marker = 1;

             currentfile = eepromreadbyte(y + (pagenum \* 4));

             //set currentfile to whichever file the user hit enter on

             break;

           }

         case 1:

           //basic document movement commands

           if (y == 0) {

             closedoc();

           } else if (y == 1) {

             pagenum--;

           } else if (y == 3) {

             pagenum++;

           }

           break;

         case 4:

           hitenter = true;

           break;

       }

       break;

     default:

       lcd.print("!");//if the input was out of bouns output error

       break;

   }

**Serial**.print("x");

**Serial**.println(x);

**Serial**.print("y");

**Serial**.println(y);

 }

 movecursor(x, y, "lcd");//update cursor location

}

void setup() {

 //CUSTOM CHARACTERS

 lcd.createChar(0, xclose);//top left x

 lcd.createChar(1, full);//a fill 8x4 pixel layout

 lcd.createChar(2, up);//up arrow

 lcd.createChar(3, down);//down arrow

 //setup

**SD**.begin(10);

 lcd.begin(lcdwidth, lcdheight);

 titlelcd.begin(titlelcdwidth, titlelcdheight);

**Serial**.begin(9600);

 pinMode(10, OUTPUT);

 //sets all varaibles to starting value

 marker = 0;

 docOpen = false;

 passwordRun();//ask for password, only once

}

void loop() {

 //switch case to determine what function the user wants to perform

 switch (marker) {

   case 0://Menu

     lcd.clear();

     menu();

     lcd.clear();

     break;

   case 1://openDoc

     if (docOpen == true)

     {

       opendoc();

**Serial**.println("DOCOPEN");

       scroll(pagenum);

       writetodoc();

     } else {

       viewdocs();

       lcd.clear();

     }

     break;

   case 2://makeDoc

     makedoc();

**Serial**.print(currentfile);

     opendoc();

     break;

   case 3://deleteDoc

     deletedoc(userinput("Delete?"));

     break;

 }

}

## Media

|  |  |
| --- | --- |
| /Users/student/Desktop/Screen Shot 2018-05-23 at 6.16.00 PM.png | |
| Eagle Board | |
| /Users/student/Desktop/Screen Shot 2018-05-26 at 5.00.14 PM.png | |
| Eagle Schematic | |
| /Users/student/Downloads/IMG_20180526_211631624.jpg | /Users/student/Downloads/IMG_20180526_170956198.jpg |
| Full Textwriter(close) | Full Textwriter(open) |
| /Users/student/Downloads/IMG_20180526_153715967.jpg | /Users/student/Downloads/IMG_20180526_153730888.jpg |
| Inside of Box | Display |
| /Users/student/Downloads/IMG_20180526_172930336.jpg | /Users/student/Downloads/IMG_20180526_153647139_HDR.jpg |
| Keyboard PCB(before soldering) | Keyboard |
| YouTube Video  <https://www.youtube.com/watch?v=xkaIgpPPelo>  Apologies for the video quality. Part way through the video the voltage regulators output changed slightly which threw the arrowkeys off. I tried to rewire the resistors but it ended up being a waste of time. The voltage regulators output keeps changing, I don’t know if it is the regulator or the power source but it makes finding the “right” resistor values very difficult. Either way I should have have not used analog parts like these, or made a more robust voltage divider, or even just found a more exact regulator, either way in the end it is on me. | |

## Reflection

Overall this was one of the hardest assignments I have done so far. Working on this code was one of the most complex and frustrating parts of this course so far. It provided many new obstacles that I was not prepared to solve. The result of this being lots of backtracking and modifying code. As well the keyboard and its resistors provided an excellent lesson in being patient. Now this is no different from my other ISPs, I end up trying to tackle a beast far more complex than I can comprehend, and I end up with a product that works, ish. For the second time, I have tried to use analog parts in my ISP and it simply ends in frustration. Just when it seems like everything will be okay those parts, in my case the keyboard, always jump in and make it go wrong. I need to learn to not just find a way for something to work, but find a way for it to work perfectly. I get carried away with cool ideas and often fail to analyze them objectively. As with all my ISPs there is an element of disappointment. But an element of pride. Because in the end, it worked. Yes it was very jumpy and only worked under select circumstances but it did. And I think something I need to learn for future is to recognize that more. When working on projects like this it is easy to get sucked into them and tune out everything around you. Not only do you become desensitized with the project but with every other aspect of your environment as well. So, to sit back and to look objectively at what you have just created, and to look out the window at the beautiful spring weather is important. It keeps you grounded and keeps everything your working on in perspective. Overall, with all its excitement and frustration this project and the TEI3M course is over. And I am ready for the break. Ready to just relax and not have to worry about much for a time. But you can be sure that once that break is over I will be all ready for yet another year of creating.

# ICS4U-AVR Optimization

2018-2019

# Project 16. CHUMP

## Code

### Purpose

To write a piece of code that will run on the “CHUMP” 4-bit computer.

### Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html#tasks>

### Theory

Code

Since the CHUMP is a 4-bit computer its capabilities in regard to commands are quite simple. It can only ever deal with the numbers 0-15 in every respect, meaning its computation ability and command set are quite limited, the commands below are the entire set(the ? can be defined by the user).

Each line of code is 1 byte in size, the high nibble being one of the commands below, and the low nibble being a number provided by the user(0-15) to be used in conjunction with the command. If the command(high nibble) ends with a 0, that means the number(low nibble) will be used as a constant for whatever the command needs. However if the command ends with a 1, then the following number will be used to find an address in memory, and whichever number is at that address will then be used by the command.

|  |
| --- |
|  |

### Code

Purpose: this code should store 15 at address 15 and address 0 in storage. Then it should count down in both of these addresses until they equal 0. When they do it should loop back and store 15 in both locations again, continuing the process.

0000: 00001111 Load 15 ; accum<-15

0001: 01101111 StoreTo 15 ; mem[15]<-accum

0002: 10001111 Read 15 ; addr<-15

0003: 00010000 Load IT ; accum<-mem[addr]

0004: 01000001 Subtract 1 ; accum--

0005: 01100000 StoreTo 0 ; mem[0]<-accum

0006: 01101111 StoreTo 15 ; mem[15]<-accum

0007: 11000000 IfZero 0 ; accum==0?pc<-0

0008: 10100010 GoTo 2 ; pc<-2

### Reflection

So far an interesting start to the project. I’ve definitely appreciated writing code in 1’s and 0’s, the low level approach has been fun to conceptualize and work with. Even though initially I found it quite difficult, as I became more used to how the machine works, the coding become dramtically easier. Overall a good start, I am ecxited to see what this will become.

## Clock

### Purpose

To wire a small circuit that will provide the clock signal for the CHUMP.

### Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

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

### Theory

LM555

|  |
| --- |
|  |
| LM555 Internal Diagram |

The primary driver of this small circuit is the LM555 Timer. Though this IC can be used for many functions. Its primary use is as a square wave generator or timer. Essentially the chip uses a voltage divider to feed distinct voltages to comparators(see below), the other inputs to these comparators run to pins trigger(2) and threshold(6). As the voltage in these pins increases eventually they will pass the voltages provided by the internal divider. If 5V is being provided at Vcc then, 1.67V will be at the Trigger comparator and 3.33 will be provided at the Threshold comparator, so once the ambient external voltage connected to Trigger passes 1.67V, It’s output will switch from HIGH to LOW. Then once the ambient external voltage connected to Threshold passes 3.33V, that comparator will switch from outputting LOW to outputting HIGH. Which will reset the SR Latch(see below), putting Q(output) LOW. Since Q is now LOW, that means the inverse Q will be HIGH, this will trigger the base pin of an internal NPN transistor, thereby allowing current to flow directly from discharge(7) to GND. There is also a direct Reset(4) pin, which will reset the SR latch, and return it to its base state.

Comparator

A comparator is a device that takes in two inputs, labelled as positive and negative. If the voltage value presented at the negative input greater than the value presented at the positive, the comparator’s output will be LOW. However, if the value presented at the positive input is greater than the value at the negative input, the comparator will output HIGH.

|  |
| --- |
|  |
| SR(Set-Reset) Latch |

SR Latch(Flip Flop)

An SR or Set-Reset Latch is an advanced application of the NOR logic gate. It contains two NOR gates, with R being an input to one, S being the input to the other, and both of their outputs being tied to the others second input. If both S and R are LOW, that will cause Q to go HIGH. By doing so they would also supply HIGH to the input of the S NOR gate, thereby immediately putting Inverse Q LOW.

From then on out, the only way to change the circuit is to input HIGH to R, which would switch around the roles and put Q LOW(If HIGH

|  |  |  |
| --- | --- | --- |
| NOR Logic | | |
| A | B | OUT |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |

was put into S, since its a NOR and it already has one HIGH input no logic would change). Then to switch again, input HIGH to S, which would revert the circuit to its initial state. This is essentially the function of an SR latch. Send HIGH to S to send out HIGH from Q. Then rest and Q will continuously output HIGH until HIGH is put on R, which will put LOW to Q until S is put HIGH again. And so on.

Debouncing

With any sort of switch or button there is a chance that when they are pressed, the two metal

|  |
| --- |
|  |
| Switch Bouncing |

leads that make the connection internally will not connect perfectly. Sometimes the two leads bounce off of each other a few times before actually setting in together. This is problematic because instead of the output current being symbolic of a single press, it will seem like it was pressed on and off several times quickly. This phenomena is called bouncing. And unsurprisingly the way to fix it is refered to as debouncing. It can be debounced in a variety of ways. But the idea is that once the button or switch gives HIGH, there is a delay before the switches state is checked again, this will allow the bounce to cycle through, eliminating the noise. With the 555 we simply used the switch to toggle the internal SR latch’s S input on one side, and on the other toggling the R input. If S is HIGH, Q will output HIGH, if R is HIGH, Q will output LOW.

### Procedure

Essentailly this circuit functions as a timer for a the larger CHUMP computer. It has two different modes, Astable(automatic square wave), and Monostable(manual pulse). A debounced switch and several logic gate IC’s give the user the option to choose which mode the board outputs.

Astable

|  |
| --- |
| Parts List |
| 3x LM555 |
| SN74LS04 |
| SN74LS08 |
| SN74LS32 |
| 4x LED |
| 2x switches |
| 5x 0.01μF Ceramic Capacitors |
| 0.1μF Electrolytic Capacitor |
| 4x 1kΩ Resistors |
| 3x 220Ω Resistors |
| 1MΩ Resistor |
| 10kΩ Resistor |
| Push Button |

As discussed above the 555’s primary function, though there are many, is to output a square wave from it’s

output pin. To do this two resistors and a capacitor needs to be added. A 1kΩ from 5V to discharge(7) then a 100kΩ from discharge to threshold(6) and trigger(2), which should be tied together. Then finally both threshold and trigger should lead to a 1μ capacitor and then GND. This combination of resistors and capacitor will be used to make the actual raw timing for the 555. Essentially, when the circuit starts running current will immediately flow into the capacitor, filling it up. As it fills up it will raise the ambient voltage attached to the threshold and trigger pins. Once the ambient voltage reaches 3.33V it will overcome the thresholds comparator, and switch the SR latches output from HIGH, to LOW. This action will trigger the SR latches inverse output to go from LOW to HIGH, which will cause an internal NPN to connect discharge to GND. This will immediately cause the capacitor to drain it’s power through the discharge pin, reducing the ambient voltage back to 0V. By doing that both of the comparators would return to their original states, causing the SR latches output to go HIGH again, and its inverse LOW turning the discharge NPN off. Then the circuit would be as it was initially and the cycle would start over again.

Monostable

The second use of the 555 is as a single pulse generator. It requires a 1MΩ resistor from 5V to discharge and threshold and then another 1μ capacitor to GND. Then a push button from GND to trigger, with a 1kΩ pull up resistor. This circuits rest configuration has it outputting LOW, meaning that its inverse connected to the base of the internal NPN resistor is outputting HIGH. Because of this, when the circuit is at its rest state, current is flowing in from 5V through the 1MΩ resistor, into the discharge pin, through the activated NPN and into GND. As well, with the button at the rest state, a constant 5V will be presented at the trigger pin causing the comparator to stay outputting LOW. However, once the button is pressed and released, the 5V would flow into GND instead of the trigger pin. This would cause the comparator to go HIGH “setting” the SR latch’s output HIGH. This would cause the inverse output to go LOW, shutting of the NPN and discharge pin. Because of this the capacitor would start to charge up, and using similar logic to the previous application, once it charges past 3.33V it would activate it’s comparator it go HIGH, resetting the SR latch to LOW. Returning the circuit to it’s orginal resting state. And since this circuit has no feedback, that is where it would end. With a simple debounced single cycle square wave output.

Output Switch

The third way the 555 was used was simply to debounce a switch using its internal SR latch.

|  |
| --- |
|  |
| Gate Logic Diagram |

Attach the switch’s middle pin to GND, and then one side to reset and the other to the trigger pin, which should also have a 1kΩ pull-up resistor attached to it. This is attached to an SN74LS04, SN74LS08, and the SN74LS32, and it uses the gates they provide to supply the logic that allows the switch to output either astable or monostable square wave output. There is also a halt(HLT) switch. While the other switch let’s the user choose their desired output, this switch is used to turn on or off the output entirely

### Media

|  |  |
| --- | --- |
|  |  |
| Full Circuit | Full Circuit Side |
|  |  |
| Full Circuit Left Side | Full Circuit Right Side |
| Video:  <https://www.youtube.com/watch?v=ieHuFc3vR2k> | |

### Reflection

Unlike most projects, the hardest part of this assignment was easily the write-up. I found conceptualizing the circuit and specifically the 555 on its own difficult let alone trying to communicate it to someone else. However, Ben Eater’s videos provided constant support and clarity and without his deep explanations, it would have been a far more difficult to achieve my own understanding of how this IC and the circuit as a whole, functions. It was, however, a very nice end product. I am quite happy with how the circuit turned out.

## Arithmetic and Logic Unit

### Purpose

To wire the a small circuit that shows the SN74LS181N’s arithmetic and logic capabilities.

### Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html#ALU>

### Theory

|  |
| --- |
|  |
| ALU Diagram |

ALU

An ALU or Arithmetic and Logic Unit is the computing power of a computer. Handling both arithmetic calculations and low-level logic gates. It takes in 2 inputs, labelled as “Integer Operand”, these are the actual numbers or logic signals to be computed. The “Opcode” input determines an actual function that will be computed. And the “Status”(carry) input/outputs, are for additional data. Such as if the ALU generates a number larger than it can output, it can output the extra bit that it couldn’t traditional communicate.

|  |
| --- |
|  |

SN74LS181N

The SN74LS181N is a 4 bit ALU. With a fairly simple logical structure, but a much more complex nature in reality. It takes a 4-bit input from pins A0-A3 and B0-B3, which function as the ALU’s “Integer Operands”. The next important pin is the M or Mode pin. Depending on the logic presented to this pin the ALU will switch between its Logic mode and Arithmetic Mode. Then, the user can choose which specific command they would like to use with the select, or “Opcode” pins, S0-S3. After that the result from the calculation or logic gate will be outputted through the F0-F3 pins. The board also contains an input and output “carry”. This output will go HIGH if the value generated by the ALU is larger than it has the capacity to output, essentially adding another bit of accuracy. The input is the opposite, if it is HIGH it essentially just adds 1 to any arithmetic calculation being completed.

|  |
| --- |
|  |
| SN74LS181N Internal Logic |

There are several important caveats to this IC though. First off, it has two overall “modes” of its own, Active HIGH in one state, Active LOW in the other. Interestingly the circuit is not actually set to one mode or another, the desired state is achieved by simply using that logic. As long as it is standardized across the entire chip Active HIGH or Active LOW can be used and the chip will interpret it correctly. The user has to be careful to use the correct graphs on the datasheet, for example, the select pins have a different table of their results depending on which logic state is being used.

As well on the Operation Select Table below wherever there is a “+” sign, it represents the OR logic gate, not the addition symbol. Any addition will be denotated by “PLUS” or “MINUS in the case of subtraction. As well when two numbers are touching such as “AB” it does not ream multiplication, it instead means the AND logic gate.

|  |
| --- |
|  |
| Operation Select Table(Active HIGH) |

Active HIGH – Active LOW

Even though it is commonly accepted that 1 = HIGH = True. That is not necessarily a given. TRUE can either be caused by HIGH or LOW depending on the logic the system uses. With this, we can derive the following results.

Active HIGH: 1 = True, 0 = False

Active LOW: 1 = False, 0 = True

On some ICs, a pin can have a horizontal line above its name, this means that that particular pins function is activated when that pin is put LOW or Active LOW.

### Procedure

|  |
| --- |
| Parts List |
| LED Bar Graph |
| Bussed Resistor Network |
| SN74LS181N |
| Jumper Wire |

Unlike the previous entry for the CHUMP, this circuit is a proof of concept as opposed to an actually applicable tool. All of the A, B, S, and M inputs are connected with jumper wires leading to either GND or 5V. This allows the user to manually select both A and B operand inputs, as well as what arithmetic or logic operator will be used to combine A and B. The F outputs from this circuit lead to a LED bar graph and bused resistor pack to display the result from the operation.

### Media

|  |  |
| --- | --- |
|  |  |
| Full Circuit Top View | Circuit Jumper Wires Removed Top View |
| YouTube Video  <https://www.youtube.com/watch?v=m4X01Vwcdcw> | |

### Reflection

This circuit has been quite nice and easy to wire, a refreshing change from many of my previous circuits. As well the large-scale logic was relatively simple, which was quite nice compared to the previous project. The real challenge arrived in actually trying to physically execute the logic. Initially playing around with it I encountered many strange results that seemed completely random. Eventually, these problems were largely explained and by the end of this project the function of this IC makes sense. It has also been a cool introduction into what an ALU can do, and the power that they have.

## Processor

### Purpose

To wire the program counter, program ROM and control ROM of the CHUMP, and successfully see the code stored in the program execute.

### Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

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

### Theory

|  |
| --- |
|  |
| 74LS161 Pinout |

74LS161

This IC is a 4-bit counter. Meaning that it takes in a square wave clock input, and whenever that

the wave goes from LOW to HIGH(or LOW to HIGH depending on the chip), it adds one to an internal counter. This counter is then displayed through 4 pins, QA-QD. Once it reaches its maximum output(15), it will toggle the Ripple Carry Output(RCO) HIGH then back to LOW, and reset the counter back to 0. This counter value can also be set at any time by triggering the active-LOW load pin and inputting the desired counter value into the A-D input pins. The reset pin is an active-LOW pin used to reset the chip from 0. Finally, Enable P and Enable T can be set HIGH for the natural functioning of the IC.

28C17

The 28C17 is a 16k bit EEPROM(Electronically Erasable Programmable Read-Only Memory) IC. The 16k of bits is arranged in an array 2048\*8 in size. Meaning the device has 2048 possible addresses, with one byte of storage at each one.

|  |  |
| --- | --- |
| 74LS161 Pinout Description | |
| RDY/BUSY | outputs the current state of the IC(LOW if busy, HIGH if ready) |
| NC(No Connection) |  |
| A0-A10 (Address pins) | Used to find the place to either read too or write from in storage |
| I/O0-I/O7(Input/Output) | Used to input the number to store, or output the number stored at the address presented by the address pins. |
| WE(Write Enable) | Enables to chip to be written too(LOW to enable) |
| OE(Output Enable) | Enables the chip to output data(LOW to enable) |
| CE(Chip Enable) | Enables the chip to be interacted with(LOW to enable) |

|  |
| --- |
|  |
| 28C17 Pinout |

-Read

To read from pre-stored memory from the 28C17, put CE and OE LOW, and WE HIGH. As well present the desired address to the address pins. If this is done the input/output pins will be outputting the value stored at the desired address.

-Write

Writing is similar to reading, it starts by putting OE HIGH and WE and CE LOW. Then the user presents the address to write too through the address pins, and whatever value is being inputted into the input/output pins will be written to that location. As an aside, when a location is written too, it automatically erases whatever is stored there.

### Procedure

|  |
| --- |
|  |
| Block Diagram of CHUMP |

The circuit starts by first burning our program code and control code onto the program ROM and control ROM. This is done externally from our CHUMP which a specialized circuit.

Once the desired code is burned onto the program ROM and control ROM, they can be plugged into the CHUMP circuit.

This circuit starts with the Clock, discussed in the previous section. This clock sends its square wave into the 74LS16 4-bit counter. It takes this square wave, and with each rising edge(LOW to HIGH) the IC adds one to an internal counter. As discussed above this counter outputs its value through pinsQA-QD, which are tied to the A0-A3 on the program ROM(28C17). This allows the current value of the counter to be the location to read code from on the program ROM. So as the clock pulse enters the counter, it goes through each line of code stored in the ROM and sends it out through its output pins, I/O0-I/O7.

|  |
| --- |
| Parts List |
| 2x 28C17 |
| 74LS161 |
| 8x LED |
| Clock Circuit(see previous section) |
| Connecting Wire |

The HIGH-nibble of this code(I/O0-I/O3) then runs to another IC, the control ROM, which is another 28C17. The signals enter the control ROM through its address pins A0-A3, being used to find a particular address location containing the control signals for the presented piece of code. It will then output these values through its I/O pins. These control signals go all over the CHUMP and are represented in the diagram above by a “C” with a circle around it.

Unconnected Pins

For now, the 74LS161 A-D inputs, LOAD, RESET, and RipCarry Output are left unconnected. As well the program ROM’s I/O4-I/O7, are just tied to LEDs, and all of the control ROM outputs are tied to LEDs as well. When the CHUMP is completed, all of these pins will be connected.

### Media

|  |  |
| --- | --- |
|  |  |
| Program ROM (Jumper Wires removed) | Control ROM |
|  | | |
| Full Circuit | |
| YouTube Video  <https://www.youtube.com/watch?v=6LoEplz2p04> | |

### Reflection

Initially, the circuit was quite intimidating. However, once I researched both IC’s in-depth, very quickly everything began to make sense. Because even though a machine like a 4-bit computer is definitely not simple, if the user understands each IC and each IC’s function and attacks the problem module by module it breaks down the daunting nature of the task into something very much doable. As well I am getting a taste for manually cutting and bending my own custom length wires, which is admittedly a test in patience, but the result is very much worth it.

## Completed Processor

### Purpose

To completely finish the CHUMP and to see the previously written code execute through LEDs on it.

### Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/4BitComputer/index.html#Individual>

### Theory

74LS174(Address Flip-Flop)

This IC is a Hex D Flip-Flip, meaning that it contains 6 Flip-Flops(SR latches)(see the

|  |
| --- |
|  |
| 74LS174 Pinout |

CHUMP: Clock section for an explanation), each with one input is given to the user as a pin, and the other tied to a Clock Pulse(CP) pin. This allows the 74LS174 to function as a temporary storage unit, taking in an input(HIGH or LOW) and holding on to it for the duration of a single clock cycle, while constantly outputting it. Then once that single clock cycle is over, the Flip-Flop is reset and a new value can be stored. Its pinout is rather simple, there are 6 inputs, Q0-Q5, and 6 outputs D0-D5, one of each for each SR latch. Then there is the Clock input labelled at CP, and a master reset(MR) to reset all of SR latch at any point.

74LS377(Accumulator)

This IC is very similar to the 74LS174 in function, but with some slight advantages and

|  |
| --- |
|  |
| 74LS377 Pinout |

disadvantages. First off, the function is the same, it is a temporary register built out of consecutive SR latches, in this case, 8. All these latches have one of their inputs available to the user by pin, and the other tied to the clock pin(CLK). However, unlike the previous IC, it has replaced its master reset(MR) pin with an active-LOW CLK Enable(G) pin. This allows the user to either allow or deny the clock cycle to reset the SR latch. Giving the user the option of storing a value for more than one clock cycle. Again it’s pinout is similar to the 74LS174. It has 8 inputs Q1-Q8, and 8 outputs D1-D8, one for each SR latch. Then it has its Clock input(CLK) and its clock enable pin(G).

|  |
| --- |
|  |

Multiplexer

A multiplexer, or data selector, is a hardware component that functions like an electronically activated multistate switch. It takes in several data inputs, and use a single selector pin to decide which of these inputs it will carry through an output.

|  |
| --- |
|  |
| 74LS157 Pinout |

74LS157(Selector)

The 74LS157 is a Quad 2-Line to 1-Line Multiplexer, meaning that it contains four internal multiplexers each one with 2 inputs(A1-A4&B1-B4) and 1 output(Y1-Y4). Each of these devices has a common select pin(S), and a strobe or clock(G) pin. This strobe pin is more of a peripheral component to the multiplexer. It allows the logic of the output to only ever change between clock cycles.

74LS189(RAM)

The 74LS189 is a 64 bit RAM IC. Its data is stored in a 4x16 array. This means that at each

|  |
| --- |
|  |
| 74LS189 Pinout |

location in memory a 4-bit value can be stored. To accomplish this, there are 4 address pins, A0-A3, and 4 input pins D1-D4. There are also 4 inverted outputs pins, and 2 utility active-LOW pins CS(Chip Select) WE(Write Enable), allowing the user to choose a desired function, input or output.

-Input: to input a value to this RAM IC, CS must be presented with HIGH, and WE must be held LOW. Then the user can present their value into pins D1-D4, and it will be stored at whatever address is being presented to A0-A3.

-Output: to output a value from this IC, CS can be LOW and WE can be HIGH, as we don’t need to write any values. Then, this IC will output whatever value is stored at the given address from pins A0-A3.

### Procedure

As discussed in the “Processor” section, the Clock is being used to drive the program counter, which itself counts through each line of code contained on the program ROM. This code is then sent out in two separate signals, the Opcode and a 4-bit constant number. The Opcode goes into the control ROM which translates it into control signals. These signals are what prepares all the other parts of the CHUMP to execute whatever command the user would like.

The control ROMs outputs are as follows,

|  |  |  |
| --- | --- | --- |
| Control Pin | Connection  Location | Function |
| I/O0 | Addr(D4)  Addr(Q4)-  RAM(WE) | R/W- To tell the RAM to prepare to ouput a value from memory or to input a value to memory |
| I/O1 | Accum(E) | Enable- To enable or disable the accumulator. |
| I/O2 | ALU(Cn) | Carry- To input a carry bit into the accumulator or to leave it at rest |
| I/O3 | ALU(M) | Mode- To tell the ALU to interpret logic or arithmetic calculations. |
| I/O4-7 | ALU(S0-3) | Function Select- A 4 bit data path to determine the function that the user would like to execute. |

Unique Control Pins

There are two unique control pins that are not sourced from the control ROM’s outputs.

* 1)The first is rather simple, it is simply tied from the program ROM’s I/O4 straight too the the Selector, and is used to determine wither the provided value from the user will be used as the constant used in calculations itself, or if that provided number will be used to find an address in RAM, and that number will be used as the constant.
* 2)The second control pin is sourced with far more complexity. It starts with three separate pins from the control ROM I/O5, I/O6, and I/O7. I/O5 and I/O6 are first connected to the two inputs to the spare OR gate on Clock board. Then the output from that OR gate is connected to one input of the spare AND gate, also on the Clock board. The second input to that AND gate is pin I/O7. The output from that AND gate is technically the final control pin. This pin is combined with the “z” pin in a NAND gate, with its output attached to the program counters active-LOW LOAD pin. This pin if put LOW allows the user to manually set the program counters internal counter. With the goal in relation to the CHUMP of choosing the line of code to read next.

So once the control signals have been sent out, the 4-bit constant from the program ROM can be interpreted and computed as the user dictated. To do this the number runs down to the A1-A4 pins of the Selector. Then depending on the Control pin tied to the Selector’s Select(S) pin, it will decide to forward through the A input(the constant number from the program ROM) or it will forward through its B input(a value from RAM). The number sent through then connects to three separate locations.

|  |
| --- |
| Parts List |
| 74LS189 |
| 74LS157 |
| 74LS174 |
| 74LS377 |
| 2x 74LSC17 |
| 74LS161 |
| CD4069 |
| CD4011BE |
| LEDs |
| Clock Circuit(see clock section) |
| 5x 2.2k Ω Resistors |
| Connecting Wire |

Firstly this number is sent to the input pins(A-D) of the program counter. Most of the time this

will mean nothing, as the inputs will be disabled. However, if the current OpCode is either the GOTO or the IFZERO function it will be very different. This is because the active-LOW input enable pin, referred to as the LOAD, is connected to a NAND gate with two inputs. One of these inputs is from the A=B pin of the ALU which is simply set HIGH is the IFZERO command is true(notated as “z” on the diagram), while the other input is the second unique control pin. So if the GOTO or IFZERO codes are used, then the 4-bit constant may be used to select the next line of code to read from.

The second location is to the Address Flip-Flop and its job is rather simple. The address will simply store this 4-bit constant for the duration of a single clock cycle, constantly outputting it. Its outputs are then connected the address pins of the RAM. This allows the user to interact with specific addresses in RAM.

|  |
| --- |
|  |
| Block Diagram of CHUMP |

The third location is arguably the most vital, and it is the connection to the B(B0-B3) inputs of the ALU. From here the 4-bit number will be computed according to 6 control pins(I/O2-7) and the current value being presented at the A(A0-A3) inputs(which is being sourced from the Accumulators outputs).

Now if the Accumulators Enable control pin is HIGH the new computed number will be directly inputted into it for storage. The output of this stored value is then sent to two locations. As brought up above it is sent to the A input of the ALU allowing the user to use it in more calculations in later clock cycles, and to the RAM inputs(D1-D4).

So if the R/W control pin on the RAM is currently set to Write, this computed value from the accumulator will be stored at whatever address is currently store the address register. If the RAM is set to Read then the address presented at its data inputs are irrelevant, the only value that matters to it is the value in the Address Flip-Flop, because the RAM is using that value to determine which location in storage to output. Those output pins(O1-O4) are then connected to the Selectors B input. This allows the user instead of manually supplying their 4-bit constant to be used itself in calculations, it can instead be used to find an address in RAM, and that value will be used as the new 4-bit constant number in whatever calculations or other functions the user needs.

### Issues

Throughout this project, there have been more than a few issues that have come up.

One of the most frustrating of these has to do with the A=B pin of the ALU. Contrary to this pins name, it’s output pin has absolutely nothing to do comparing the A and the B pin. Instead, it’s output will only go HIGH if all of the ALU’s outputs(F0-F3) are HIGH.

The second complication that came up had to do with the RAM IC. Even though it seems very simple, as the datasheet states, their outputs are the “complement” of their inputs. This essentially means that when a value at an address is outputted it will be inverted from what it was stored as. Feinberg recommends inverting the data before storing it so that the data outputted can be used straight from the output pins. Initally to do this I simply used 4 PNPs, tieing the emitter to 5V, the base pin to the output pin from the accumulator(that would normally be connected straight to the RAM’s input pin) and finally the collector to the RAM input pin. And even though logically this seesm like it should work, it didn’t. So to remedy this I instead used a simple NOT gate to invert the signal which ended up working quite well.

### Media

|  |  |
| --- | --- |
|  |  |
| CHUMP Top Board | CHUMP Middle Board |
|  | YouTube Video  <https://www.youtube.com/watch?v=H0CE4R-k4Os> |
| CHUMP Bottom Board(Clock) |  |
|  | |
| CHUMP Full | |

### Reflection

Finally, the CHUMP comes to a close. Despite all of this project's complications and long hours it has been remarkable to finish. Admittedly near the end of this project, I began to fear that I would not be able to finish. However, with the help of Daniel and a fair sum of patience, I was able to get through all the bugs and errors that plagued the machine. And finally finishing this project has been one of the nicest payoffs so far. Seeing it finally work literally hours before the due date is quite the feeling. Overall a very fun yet tiering project. And it has peaked my interest in the computing world.

# Project 17. Elevator

## Purpose

To design a modular elevator that is capable of lifting a relatively large mass in relation to its own size.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/1819/ISPs.html>

## Acknowledgments

Kreher Fiset-Algarvio – 3-D Printing the entire elevator

## Theory

### Hardware

Torque

|  |
| --- |
| Definition: Torque |
| The measure of a force that can cause an object to rotate around an axis |

As the definition states, torque is the measure of a force, around an axis. There are many units for this, such as (kg)(m^2)/(s^2). And though these units are accurate, intuitively they are difficult to conceptualize. So simply for comprehension purposes, we can use kg\*cms. This makes it dramatically more simple, for example, the motor used in the elevator is 18kg\*cm. This means that at 1cm away from the centre of rotation(the motors output shaft) the motor can lift 18kg. Or at 2cm it could lift 9kg.

|  |
| --- |
|  |

DC Motors

A DC motor is a rotary actuator, that converts electrically current into mechanical force. There are two offshoots of the DC motor the Brushed and Brushless. Each with their own pros and cons.

-Brushed DC motors

Brushed DC motors are generally the most common and easy to use. This because at their most basic level they have two simple inputs. Which can be connected straight to a power source. This is due to the internal functioning of these motors. Essentially, each input pin is connected to a part called an armature which is then connected to the output shaft. So to rotate this armature and as an extension the output shaft two parts must be applied. Firstly, since the armature is connected to a power source and thereby current is flowing through it, it will generate a magnetic field around itself. This is magnified by the fact that the armature will have wire coiled around it, which enhances magnetic fields. The direction of rotation is determined by which way the current is flowing through the armature(which way the power source is connected). Alone this is useless, however, generally, this armature will be surrounded(not connected too) by two magnets, with opposite polarities. These magnets have their own strong magnetic fields which will interact with that of the armature and causes it to spin. Now intuitively it seems like there should be a problem since the power source is connected straight to the rotating armature. However, this is not exactly true in reality. Instead, the two input wires are connected to “Brushes” which like the wire is static in its movement. These brushes are then connected to the armature through a part called a communicator, which essentially just transfers the current from the brushes into the armature. Because of this physical connection over time, these brushes will wear out.

-Brushless DC Motors

|  |
| --- |
|  |

Brushless are simply the opposite of brushed motors. Instead of having fixed magnets on the outside in static motion they have the magnets on the armature. By doing this the current from the user is sent to coils that surround the middle armature, thereby removing the need to have brushes, since no current has to be communicated from a fixed to a rotary position. From here they function in similar ways. Varying charges presented at the static coils create magnetic fields. These fields then interact with the magnets fields causing the armature to rotate.

|  |  |  |
| --- | --- | --- |
|  | Pros | Cons |
| Brushed | -Cheap  -Simple to interact with | -Inefficent(power wasted from brushes friction)75-80%  -Brushes ware out and must be replaced |
| Brushless | -Efficent 90-85%  -Very long lifespan(no brushes that can be worn out) | -Expensive(has to be driven with an encoder) |

\*Efficency percentage is the percentage of electrical power that is translated into mechanical energy, typically energy will be lost as heat

Gearboxes

A gearbox is a peripheral hardware component that can be added to a motor to adjust its RPM to torque ratio. They function by using a combination of gears to act as a middleman between a motors output shaft and a new modified output shaft. Using gears of varying diameters, the user can increase RPM or torque to any desired value. The drawback is that these devices don’t draw energy from anywhere, whenever they increase one value, the other value must decrease proportionally.

SN754410

The SN754410 is a quad half H-Bridge IC. This means that it contains four different half H

|  |
| --- |
|  |
| SN754410 Pinout |

bridges. So to power this IC it requires two separate sources, VCC1, which is just a constant 5V for the IC’s logic. And VCC2 which can have between 4.5 and 36V presented to it, whatever voltage this is it will be used a the raw power the IC outputs. So whenever an input pin A1-A4 has HIGH presented to it, its corresponding output pin Y1-Y4 will have the VCC2 voltage put through it. 1,2EN and 3,4EN are simply put HIGH if the user needs any either 1 and 2, and/or 3 and 4 input and output pins. And finally Heat Sink and Ground Pins can all be put to GND if the voltage and current going through the IC is relatively small, however, if it large at least one of those pins can be should to a heat sink to reduce the chance this IC will overheat.

### Software

Interrupts

A timer interrupt essentially allows the user to at any point stop whatever code is happening, run a predefined function, and then return to that point in the code again and continue as before. To do this it needs to be declared or “attached

attachInterrupt(digitalPinToInterrupt(interruptpin), function, STATE);

Where interruptpin is the pin(on an UNO must be 2 or 3) that when triggered causes an interrupt, function is the name of the function to call, and STATE is when that the pin is triggered(RISING, FALLING, HIGH, or LOW).

From here all that has to be done is to create the function to call. It must be short and must not return any values. As well its name MUST match that of the name inside the attach.

## Procedure

The goal with this project, as with many of my projects, was not to simply make a elevator. But

|  |
| --- |
| Parts List |
| 131:1 37D Metal Gear Motor |
| SN754410 |
| 4x Push Buttons |
| 4x 22k Ω Resistors |
| Bistate Switch |
| Arduino |
| Fishing Line |
| 3x Diodes |

instead to make a product that actually works, and that would actually have a place in the world. And even though obviously this is small scale and printed from a 3-D printer, the ideas that it contains should genuinely work when applied in reality.

With this in mind, the first problem to overcome was how the elevator would function. Initially, the idea was to have the elevator look like a coiled spring, with the elevator car running up and down the centre of the spring. The way it would do this would be with a track that connected the elevator car to the elevator body(spring). This track would run all the way up the spring shaped body on the inside. This would allow a string to be fed down inside the spring and connected to the elevator car through the track. Then when the string was pulled up it would pull the car up the track, thereby lifting and rotating the car. While this was an interesting idea it simply wasn’t plausible. The spring shape of the elevator body would be very weak due to its size and the fact it would have to be made from 3-D printing plastic. This would cause the elevator to crack if it ever tried to carry any real load. So even though it could have been cool, as of yet it was a bit of a reach.

So from there, a new approach had to be taken that would still be unique but could be applied and rendered in the real world. This time instead of maximizing appearance, functionality was made paramount. The new design was intended to a modular, easily transportable, easily constructible, manufacturable, and load bearing. By modular, it was meant to be constructed to any height the user required. In freestanding elevators this is a necessity as required heights will change with each project. Easily transportable is essentially just the fact that it was intended to avoid bulk. For example, instead of printing a full floor which would take up lots of space, having each wall printed separately should allow it to be moved in stacks and set up at the site. Easily constructible was intended so that it could be set up anywhere, without the need of too many actual tools and machinery to do so. Manufacturability ties into the previous two ideas. The goal by making it manufacturable was to limit unique parts. So, for example, all of the walls of the elevator are the same piece, they can all function as doors or just as walls. Reducing the amount of complex manufacturing to go into it. And finally load bearing is rather simple, it should be able to support a relatively large load to its own weight. So it is intended to be as lightweight as it can be while keeping its strength as high as possible.

Once my key ideas had been developed it could physically be designed. In order to accomplish

|  |
| --- |
|  |
| Two walls held together with a peg |

the ideas discussed above many design tools were applied. The first and most key idea has to do the implementation of the pegs. Initially, each floor and each wall on each floor would be connected using a different method. However, in an effort to reduce complexity and parts needed, the pegs were used. They allow the user to first connect each wall of each floor together using the bottom half of the peg, then, the top half of the peg is used to connect each floor together. As well, doing this it would create four “pillars” on each corner of the square elevator increasing load bearing strength. Even though this may help, the walls of the elevator would inevitably have to carry a load as well. To strengthen them while keeping them generally lighter an arch was used. Even though a network of triangles is stronger than a single arch, since each wall was also supposed to function as a door, a group of triangles would not work out.

From here the motor can be introduced. Since this motor is supposed to be able to lift a relatively large mass is must have a very high torque and a low RPM. To do this a motor with a built-in gearbox was used. This allowed the torque to mass to be set at 80RPM and 18kg\*cm of force. This gives the elevator more than enough RPM and more than enough torque to be able to lift any mass that it could fit in its car. This motor will be mounted to the bottom of the elevator in a custom case to hold it in place. Then its output shaft will be put into a 3-D printed spool. This spool will be where the fishing line where is connected and coiled up. From here the line will run up to a pully at the top of the elevator, and be curved and sent straight down the centre of the elevator and attach to the car itself. So when the motor turns it coils up the line, pulling the car up the shaft. And in turn, if the car is on the third floor and it needs to be let down, the motor simply turns the opposite way releasing line and letting the car move down the shaft.

On the software end of this project, the circuit starts doing absolutely nothing. It will wait until the first input by the user is given. This input comes in the form of 3 buttons, one for each possible floor on the elevator. Once one of them is pressed, it will send a pulse to pin 2 which is the Interupt pin dictated in the code, and to a unique analog input pin. The pulse to pin 2 will cause the Interrupt to activate and call the interrupt() function. This function is a middleman to then activate the checkbuttons() function. This function checks all the analog inputs and sets the next open cell(nextfloorlocation) in the floors[]array to whichever button was pressed(Button 1-> floors[nextfloorlocation] = 1 etc). As well this function adds one to the nextfloorlocation variable so when the next button is pressed it will be stored at the next address. And finally it sets trigger to HIGH so the movetofloor() can finally run.

When it runs it first enters through a for loop that encompasses the entire function and iterates ten times through the variable x. Then immediately it hits and if statement that checks if floors[x] is equal to zero. If it is equal to zero then that means the elevator has gone to all of the required floors for now, so it will clear the floors[]array, reset the nextfloorlocation to zero and return the trigger to LOW. Thereby returning the program to its original resting state of simply waiting for a button input. However, if floors[x] was not zero that means there is a floor to go to. So it will transfer that value from the floors[]’s array into the nextfloor variable and then use that to subtract from the current floor and find the floors needed to travel. Then it will determine if it needs to go up or down and move the desired number of floors in that direction using the upfloor() and downfloor() functions.

## Code

// Author : James Corley

// Project: Elevator(Short ISP)

// Date: Nov 3 2018

// Status: Working

#define uppin 11

#define downpin 10

#define button1pin A0

#define button2pin A1

#define button3pin A2

#define interruptpin 2

uint8\_t currentfloor = 1;      //the floor the user is currently on

int8\_t floorstomove = 0;       //the absolute value of currentfloor-nextfloor

uint16\_t timetoturn = 1500;    //the time(in milliseconds) to move one floor

uint8\_t floors[10];            //an array of floors to travel to next

uint8\_t nextfloor = 1;         //the next floor in the array

uint8\_t nextfloorlocation = 0; //the location of the next floor in the array

bool trigger = LOW;            //to determine if there is valid input

void setup() {

 //Setting Serial Monitor

**Serial**.begin(9600);

 //Setting up pins

 pinMode(uppin, OUTPUT);

 pinMode(downpin, OUTPUT);

 pinMode(button1pin, INPUT);

 pinMode(button2pin, INPUT);

 pinMode(button3pin, INPUT);

 attachInterrupt(digitalPinToInterrupt(interruptpin), interrupt, RISING);

}

void loop() {

 if (trigger)            //if there is a command ready activate movetofloor

 {

   movetofloor();

 }

}

//checks buttons current states and updates desiredfloor if any buttons were pressed

void checkbuttons()

{

 if (analogRead(button1pin) > 1000)               //first floor pressed

 {

   floors[nextfloorlocation] = 1;

 } else if (analogRead(button2pin) > 1000)        //second floor pressed

 {

   floors[nextfloorlocation] = 2;

 } else if (analogRead(button3pin) > 1000)        //third floor pressed

 {

   floors[nextfloorlocation] = 3;

 }

 nextfloorlocation++;

 trigger = HIGH;

}

void interrupt()                                   //function to be called when interrupt goes HIGH

{

 checkbuttons();

}

void movetofloor()

{

 for (uint8\_t x = 0; x < 10; x++)

 {

   if (floors[x] == 0) {                         //if there are no new floors, wipe data

     for (uint8\_t y = 0; y < 10; y++)            //clearing array

     {

       floors[y] = 0;

     }

     nextfloorlocation = 0;                      //reset array locator

     trigger = LOW;                              //return trigger to LOW

**Serial**.print("CLEAR ARRAY");

   } else {

     nextfloor = floors[x];

     floorstomove = abs(nextfloor - currentfloor); //find floorstomove

     if (currentfloor < nextfloor)            //if going up, use upfloor

     {

       for (uint8\_t x = 0; x < floorstomove; x++)

       {

         upfloor();

       }

       currentfloor = nextfloor;              //update currentfloor

     } else if (currentfloor > nextfloor)     //if going down downfloor

     {

       for (uint8\_t x = 0; x < floorstomove; x++)

       {

         downfloor();

       }

       currentfloor = nextfloor;              //update currentfloor

     }

     else {

     }

   }

**Serial**.print("NextFloor =");

**Serial**.println(nextfloor);

**Serial**.print("Currentfloor =");

**Serial**.println(currentfloor);

 }

}

void upfloor()                                  //move up a single floor

{

 digitalWrite(uppin, HIGH);

 delay(timetoturn);

 digitalWrite(uppin, LOW);

}

void downfloor()                                 //move down a single floor

{

 digitalWrite(downpin, HIGH);

 delay(timetoturn);

 digitalWrite(downpin, LOW);

}

## Media

|  |  |
| --- | --- |
|  |  |
| Full Elevator | Mounted Motor |
|  |  |
| Elevator Top | Full Circuit |
|  |  |
| Button Circuit(Interrupt, F1, F2, F3) | Motor Driver Circuit |
| YouTube Functioning Video  <https://www.youtube.com/watch?v=LfUt0U_6uDQ>  YouTube Explanation Video  <https://www.youtube.com/watch?v=rXln3PRBl9w> | |

## Reflection

Overall this project has been a pleasant change of pace for me. I was able to get my designing done and printed and everything bought weeks before it was due. So even though the CHUMP took up time in the weeks before the due date I have still been able to finish my project on time. And even though it has been a rough return to Arduino coding, it has been nice to refresh myself it in. As well, since it was a project from a list, the scope was far more realistic than many of my projects up until this point which was nice to deal with. Overall I’m proud of the more balanced approach I took to this project, a trend I can hope I can continue in future.

# Project 18. Telephone Keypad

## Purpose

To use “Telephone Keypad” as an input system for a data encryption device. The device should take this input and output it as an encypted value.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/1819/Tasks.html#TelephoneKeypad>

## Theory

Encryption

|  |
| --- |
| Definition: Encryption |
| Encoding a message or information so that only authorized parties can view it. |

There are many methods of encrypting data. These can range from simple physical encryption tables and keys, such as most people are familiar, to hypercomplex computer algorithms. And while physical encryption keys work to encrypt data, they are not actually applicable in any real complex or large-scale capacity. So to truly encrypt data in a useful way, algorithms have to be used. There are many different algorithms that can be used such as AES, 3DES, Twofish, and the focus of this assignment, RSA.

RSA

RSA is a unique encryption method due to the fact that the method to encrypt data can be made public(the public key), while keeping the method to decrypt it can remain private(the private key). The algorithm is as follows

Finding the Public Key

First two relatively large prime numbers need to be made up notated as P and Q.

These can then be multiplied to together to make n.

p = any prime

q = any prime

n=p\*q

Again use p and q to calculate Φ(n) given the equation

Φ(n) = (p-1)(q-1)

Then a small integer named e between 1<e<Φ(n) that is NOT a prime of n has to be made up. This new integer e and n will make up the public key with the equation.

E = Encrypted data

A = Actual data

E = Ae mod n

Finding the Private Key

This key uses many of the values above, and a new integer k in the below equation

k = any integer

d = (k\*Φ(n)+1)/e

Now this new d can be applied to make the private key

E = Encrypted data

A = Actual data

A = Ed mod n

Debouncing

|  |
| --- |
|  |
|  |

This topic was discussed in the above project, Project 16. CHUMP Clock. That project provided a solution to the issue using the 555 IC. And while that works, this project takes a different, more software-based approach to deboucning. Upon first receiving a signal it will wait a predefined very short amount of time and then check the signal again. If the input is still high after the time period it registers as a valid button press. This means that if there are any bumps or noise on the input, the waiting period pass over them without registering them as button presses.

Keypad

The “Telephone keypad” is a 3 by 4 matrix of buttons. Because of this on the pinout, there are 3 column output pins and 4-row output pins. This means that whenever a button is pressed it will connect the respective row and column pin for that button together. As well, the “common” pin is connected to every button, so whenever any button is pressed the common will also be connected to the respective row and column pins.

Floating Point Math

Due to the fact that the Arduino has limited space and processing power, it is forced to use a unique type of arithmetic called floating point arithmetic. Floating point arithmetic is based on the idea that a computer does not have infinite accuracy. And the closer they try to get to perfect accuracy the longer and more wasteful it gets. So to fix this, they have an admitted amount of inaccuracy. Attempting to find the perfect ratio of speed too to accurate numbers. While its algorithms can be quite complex in some situations, floating point is similar to the idea of scientific notation, where numbers are expressed as (a\*(bc)), where a is the computers admitted accuracy, called the significand. So even though it may be able to represent very large or very small numbers, it will not be able to do it accurately.

## Procedure

This project starts when the first button is pressed. This connects the buttons rows, columns, and

|  |
| --- |
| Parts List |
| 3x4 Telephone Keypad |
| Bistate Switch |
| Jumper Wire |

the common pins together. The code reacts to this change on the common pin and waits a short period to confirm that it was a genuine button press. After this time, if the signal is still present it will move to interpret the row and column pin signals to see which button was pressed. First, it will find the col, then it will prob all the rows and constantly check the entire previously found col. Since this all happens very quickly the button will still be pressed down, so the connection between the row and col will still be open. Once the previously found col pin goes high, it means the correct row was probed and both the row and col values can be saved. These values will then be used to find the location in a matrix that holds its respective buttons value. This value between 0-9 will be saved in a small 3 address temporary array since the largest ASCII character has 3 decimal digits. Once the user has entered the desired digits they can hit the ‘\*’ sign(or once they have entered 3 digits) it will convert the three numbers in the array into a single 3 digit(or fewer) number. This will then be saved in the data array. Once the user has put is as many values as they would like, they can hit the ‘#’ symbol to either encrypt or decrypt them, depending on the state of an external switch. Once this button is hit, it will overwrite the old data in the data[] array, outputting the new encrypted or decrypted values to the serial monitor.

## Issues

The most complex and unexpected error arose from the application of complex math in the Arduino IDE. This comes from several spots.

Firstly, since the largest data type on an Arduino is essentially a uint32\_t, the largest possible number that can be stored is 4 294 967 296. This makes application of the RSA algorithm difficult since it relies on values being large so computers, like this one, have a tough time decoding it.

More than that the application of these numbers is not made particularly easy either. Firstly, since most of the more complex math functions in math.h require double values that automatically constrains our max value too 2 147 483 648 since the highest bit is used for the sign. More than that the actual functions themselves are not reliable, due to the fact that it uses floating point math. For example, the pow(base, exponent) function is supposed to raise the given ‘base’, to the given ‘exponent’. And while it generally does this, for larger numbers it losses accuracy, such as giving 9 765 624.95 from the equation 255. And while this seems like a small issue, these variations throw off the algorithm and give off simply incorrect information. To fix this, it is easier to design the users own exponent and modulo functions, bypassing the strange nature of floating point math.

## Code

//  Author: James Corley

// Project: Telephone Keypad Encryptor

//    Date: Sat Dec 1 2018

//  Status: Working

#include<math.h>

#define INTERVAL 10L

#define COMMON 2

//Data Input Variables

uint8\_t keys[4][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 0, 11}};

uint8\_t colPins[] = {6, 5, 4};

uint8\_t numCols = sizeof(colPins);

uint8\_t rowPins[] = {11, 10, 9, 8};

uint8\_t numRows = sizeof(rowPins);

uint8\_t row, col;

unsigned long detect;

boolean debug = false;

boolean rowFound;

uint8\_t input;

//Encryption Variables

double p = 3;                           //any large prime

double q = 7;                           //any large prime

double n = p \* q;

double phi = ((p - 1) \* (q - 1));

double e = 2;                           //the exponent

double k = 2;                           //any integer

double d = (((k \* phi) + 1) / e);

//Data Array Variables

uint8\_t sChar[3];                        //temporary array

uint8\_t sCharCounter = 0;                //temporary array counter

uint8\_t data[10];                        //data array

uint8\_t dataCounter = 0;                 //data array counter

//State Switch Pins

#define encryptpin A0

#define decryptpin A2

#define supplypin A1

void setup() {

**Serial**.begin(9600);

 pinMode(COMMON, INPUT\_PULLUP);

 for (col = 0; col < numCols; col++)

   pinMode(colPins[col], INPUT\_PULLUP);

for (row = 0; row < numRows; row++)

 {

   pinMode(rowPins[row], OUTPUT);

   digitalWrite(rowPins[row], LOW);

 }

 pinMode(encryptpin, INPUT);

 pinMode(supplypin, OUTPUT);

 pinMode(decryptpin, INPUT);

 digitalWrite(supplypin, HIGH);

}

void loop() {

 while (e < phi)             //find e so its coprime to phi

 {

   if (gcd(e, phi) == 1)

     break;

   else

     e++;

 }

 d = (((k \* phi) + 1) / e);

 while (bitRead(PIND, COMMON));

 if (debug)**Serial**.println("Pressed");

 detect = millis();

 while ((millis() - detect) < INTERVAL);

 input = PIND;

 if (getTelKey(input) == 11) switchstate();   //checks for encrypt command

//checks for end of char

 if (getTelKey(input) == 10 || sCharCounter == 3) arrayToVal();

  if (getTelKey(input) < 10) {                 //saves button press

   sChar[sCharCounter] = getTelKey(input);

**Serial**.println(sChar[sCharCounter]);

   sCharCounter++;

 }

 while (!bitRead(PIND, COMMON));

 if (debug)**Serial**.println("Released");

 detect = millis();

 while ((millis() - detect) < INTERVAL << 1);

}

void arrayToVal()

{

 for (uint8\_t x = 0; x < 3; x++)

 {

   data[dataCounter] += sChar[x] \* pow(10, x); //digit to val

   sChar[x] = 0;                               //resets sChar[x]

   sCharCounter = 0;                           //resets sCharCounter[x]

 }

**Serial**.print("data Value at " + String(dataCounter) + " is :");

**Serial**.println(data[dataCounter]);

 dataCounter++;

}

void switchstate()

{

 if (digitalRead(encryptpin))                   //encryption mode

 {

   for (uint8\_t x = 0; x < 10; x++)

   {

**Serial**.print("Storage Space: ");

**Serial**.println(x);

**Serial**.print("RAW DATA: ");

**Serial**.println(data[x]);

     data[x] = encrypt(data[x]);               //replace original data with encrypted data

**Serial**.print(F("Encrypted Values: "));

**Serial**.println(data[x]);

   }

 }

 if (digitalRead(decryptpin))                   //decryption mode

 {

   for (uint8\_t x = 0; x < 10; x++)

   {

**Serial**.print("Storage Space: ");

**Serial**.println(x);

**Serial**.print("RAW DATA: ");

**Serial**.println(data[x]);

     data[x] = decrypt(data[x]);               //replace original data with decrypted data

**Serial**.print("Decrypted Values: ");

**Serial**.println(data[x]);

   }

 }

}

char getTelKey(uint8\_t input) {

 switch (input & 0b01110000) {

   case 0b00110000: col = 0; break;

   case 0b01010000: col = 1; break;

   default: col = 2;

 }

 if (debug) **Serial**.println("Column: " + String(col));

 for (row = 0; row < numRows; row++) {

   digitalWrite(rowPins[row], HIGH);

   if (digitalRead(colPins[col])) {

     digitalWrite(rowPins[row], LOW);

     return keys[row][col];

   }

   digitalWrite(rowPins[row], LOW);

 }

 return 'X';

}

double encrypt(double a)

{

 if (debug)**Serial**.print("e: ");

 if (debug) **Serial**.println(e);

 if (debug) **Serial**.print("d: ");

 if (debug) **Serial**.println(d);

 if (debug) **Serial**.print("a:  ");

 if (debug) **Serial**.println(a);

 if (debug) **Serial**.print("exponent(a, e);: ");

 if (debug)**Serial**.println(exponent(a, e));

 if (debug) **Serial**.print("n: ");

 if (debug) **Serial**.println(n);

 double encrypteddata = 0;                   //resetting encrypted data

 encrypteddata = exponent(a, e);             //public key

 encrypteddata = modulo(encrypteddata, n);   //public key

 return encrypteddata;                       //returning encrypted data

}

double decrypt(double a)

{

 double decrypteddata = 0;                   //reseting data

 if (debug) **Serial**.print("exponent(a, d): ");

 if (debug) **Serial**.println(exponent(a, d));

 decrypteddata = exponent(a, d);              //private key

 decrypteddata = modulo(decrypteddata, n);    //private key

 return decrypteddata;                        //returning actual data

}

int gcd(int a, int h)                       //function to find gcd

{

 int temp;

 while (1)

 {

   temp = a % h;

   if (temp == 0)

     return h;

   a = h;

   h = temp;

 }

}

double exponent(double base, double power)      //function to find power values

{

 double returnval = 1;

 for (uint8\_t a = 0; a < power; a++)

 {

   returnval = returnval \* base;

 }

 return returnval;

}

double modulo(double a, double b)            //function to find modulo values

{

 uint8\_t turnBacka;

 while (a > 0)

 {

   turnBacka = a;

   a = a - b;

 }

 return turnBacka;

}

## Media

|  |  |
| --- | --- |
|  |  |
| Full View | Arduino |
|  | YouTube Video  [https://www.youtube.com/watch?v=kZJav](https://www.youtube.com/watch?v=kZJavgYOzpg&t=37s)  [gYOzpg&t=37s](https://www.youtube.com/watch?v=kZJavgYOzpg&t=37s) |
| Keypad |  |

## Reflection

As a more code-based and theoretical project, this has been a nice departure from my traditional projects. Researching the convoluted world of data encryption has been surprisingly interesting. As well doing research into the Arduino’s arithmetic capabilities has been very useful, as it is something that I have almost never had to encounter. And though the project isn’t the most flashy, I’m still very happy with the result I have now.

# Project 19. Rotary BCD Switch

## Purpose

To use assembly code to display the location of a 16 position rotary BCD switch through a seven segment display.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/1819/Tasks.html#Rotary>

## Theory

Rotary BCD Switch

The rotary BCD switch is a 16 location rotary switch with hexadecimal characters for each 0-16

|  |
| --- |
|  |
| Rotary BCD Switch Pinout + Logic Table |

location. Its function is rather simple, it has 4 output pins all connected to two common pins. Whatever signal is at the common will be presented through the output pins depending on the switches location.(to use this signal, simply put pull down resistors on each output pin to pull current through it). Since there are 16 locations and 4 pins, the pins use binary as an output method, so for example at location 0 no output pins will be connected to common, at location 1 the first pin, at location 2 the second pin, location 3 the first and second, and so on but until a max of 16 or F in hexadecimal.

Assembly Language

Assembly is a very low-level coding language. It essentially is one layer of abstraction from pure machine code. Instead of having complex compilers like many other common languages like C or Java, the assembly code is converted to machine code through something called an assembler. Which is essentially a very minimalistic compiler that all computers contain. Because of assemblies low level and close relation to machine code its commands are not particularly simple to interact with. Each one is an abbreviation or acronym for a larger command and is generally around 2-4(very rarely 5) characters long. Followed by any register addresses or constant values required for the command. And these instructions will be very limited in what they do, so a single line in typical coding languages can take several assembly commands to execute. However, because of all of this, assembly is dramatically easier to run for the machine. Because it is almost machine code it doesn’t waste space or computing power on the fluff that comes along with most languages. As well, many higher level languages commands are overly complex and can mess around with many unexpected areas of an Arduino beneath the surface. But when using assembly the machine does purely what it is told, avoiding any background mishaps.

Port Manipulation

On every Arduino, there is a method of interaction called “Port Manipulation”. And this essentially means instead of using high-level commands like digitalWrite, the user interacts with each I/O pin on a register level. The first note to understand is that each pin is part of a larger port, which is just a bank of 8 pins. Each port is denotated with a letter, so the, for example, has ports B, C and D at their disposal. To interact with each unique port’s register simply put the register name, (i.e. DDR) followed by the port’s letter(i.e. B, the result being DDRB). The second idea is that each port has three registers, they are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Name | Function | High Level |
| DDR | Data Direction Register | Used to determine the port’s pins for either input or output | pinMode() use |
| PORT | Data Register | Used to set certain pins on the port to HIGH or LOW | digitalWrite() use |
| PIN | Input Pins Register | Used to read in any values presented to the port’s pins. | digitalRead() use |

Since each of these registers contains 8 pins, and since each one is of a size byte, the binary value presented in each register mirrors the 8 pins. For example, if PORTD = 10100110 that would be equivalent to setting pins 0,2,5, and 6 of that port to HIGH.

Arrays in Assembly

Managing arrays is assembly is a far more complex process than in typical languages. Essentially, they function off of a three value system, X, Y, and Z. X is the location in storage of the start of the array, Y is the location in storage of the end of the array, and Z is a pointer that can move between X and Y to interact with the data. Each of these values is made up of two bytes, having the high and low bytes denoted by an H or L following the values letter. These values are all stored in registers, specifically, XL, XH, YL, YH, and ZL ZH are equivalent to registers r26, r27, r28, r29, and r30 r31 respectively. So when an array is declared a label is placed at the beginning to show where the start is(X) and a label is placed at the end to show where it is(Y). Then to interact with the array, set Z to X to place it at the start of the array, and then count through it to whichever location is desired. Once Z is at that intended location the lpm ‘Register’, Z Z can be used to store the value in memory at Z to a given register.

## Procedure

This circuit starts with the rotary switch. Whichever position the switch is at it will send out

|  |
| --- |
| Parts List |
| Rotary BCD Switch |
| Seven Segment Display |
| 4x 10k Ω Resistors |
| Arduino |

that value in binary to pins 8-12 of port B. This 4-bit value corresponds to the location that the switch is currently at, so it can be used directly to source the character to display on the seven segment display. However, before this can happen the input signal needs to be decoded into something the seven segment display can actually display. This is done using an array which holds the seven segment display signals, at the respective numerical addresses. So when a value is inputted in through the switch, it is used to find the location in the array holding its decoded signals. These signals are then sent out through pins 0-7 on port D and displayed through the seven segments.

## Code

;  Author: James Corley

; Project: RotaryBCDSwitch

;    Date: Wed, Dec 19 2018

;  Status: WORKING

#include <avr/io.h>

ddrS =  DDRB - 0x20   ;setting port

pinS = PINB - 0x20    ;setting port

ddrD =  DDRD - 0x20   ;setting port

portD = PORTD - 0x20  ;setting port

displayPins = 0xFF    ;setting display pins to output

switchPins = 0x00     ;setting switch pins to input

seqStart:

.byte 0b01110111   ;0

.byte 0b00010100   ;1

.byte 0b10110011   ;2

.byte 0b10110110   ;3

.byte 0b11010100   ;4

.byte 0b11100110   ;5

.byte 0b11100111   ;6

.byte 0b00110100   ;7

.byte 0b11110111   ;8

.byte 0b11110100   ;9

.byte 0b11110101   ;A

.byte 0b11000111   ;B

.byte 0b01100011   ;C

.byte 0b10010111   ;D

.byte 0b11100011   ;E

.byte 0b11100001   ;F

seqEnd:

.global setup

setup:

;array info

ldi XH, hi8(seqStart) ;load the arrays start location(High byte) to XH

ldi XL, lo8(seqStart) ;load the arrays start location(Low byte) to XL

ldi YH, hi8(seqEnd)   ;load the arrays end location(High byte) to YH

ldi YL, lo8(seqEnd)   ;load the arrays end location(Low byte) to YL

ldi r16, displayPins  ;load displayPins into a register

ldi r17, switchPins   ;load switchPins into a register

out ddrD, r16         ;set display ddr to output

out ddrS, r17         ;set switch ddr to input

ret

.global loop

loop:

in r18, pinS          ;read switch input and save it to a register

andi r18, 0b00001111  ;mask out the upper nibble of input

movw Z, X             ;set Z marker to start of array

add ZL, r18           ;move to desired array location

lpm r19, Z            ;load desired array value into a register

out portD, r19        ;output register value

ret

## Media

|  |  |
| --- | --- |
|  |  |
| Full Circuit | Close View |
| YouTube Video:  <https://www.youtube.com/watch?v=DUw4DiCl6g0> | |

## Reflection

Overall this project has been a unique problem to solve. While it is rather simple itself, the complex part was looking up all the assembly commands and understanding what they do. This process took time and patience, but the rewarding part is that it will get easier as I write more in the language. And since I enjoy the orderly world of software it was nice to focus more heavily on that aspect than I have in previous projects. In the end, it was a solid introduction to assembly and while I have some trepidation about the new year I am excited to see what we will do next.

# Project 20. Knight Rider

## Purpose

To use assembly code to produce an interesting display from the Morland ShiftBar.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/1819/Tasks.html#KnightRider>

Microchip – AVR Assembler Instructions

<https://www.microchip.com/webdoc/avrassembler/avrassembler.wb_instruction_list.html>

## Theory

Atmel Studio 7

Atmel Studio 7 is an integrated development platform (IDP) designed for writing and debugging code for AVR or SAM microcontrollers. The environment is specifically suited for low-level coding and has a wide variety of options for specifically debugging microcontrollers, such as having direct access to their ports(discussed in the previous project).

To interact with devices like the Arduino an Atmel ICE is required as a middleman between the computer itself, and the Arduino, separate from the traditional USB A to USB B cable.

\*Notes: -Not available on Mac OS

-Power to the Arduino must be supplied separately.

ShiftOut High Level

In Ardunio C, there is the command shiftOut(). This command is used to interact with shift registers, like the 595(explained in Project 10) and takes 4 paramters:

dataPin: The pin to output the serial data

clockPin: The pin to output the clock signal

bitOrder: The order to send out the data, MSBFIRST or LSBFIRST

value: The byte to shift out

This command automatically takes these four inputs, along with a manual flip of the 595’s latch pin before and after the shiftout, and together has a very simple high level method of interaction with shift registers. However, this can also be achiveved with low level manual interaction as will be discussed in the procedure.

Morland Shift Register Bargraph

This device is a specially designed IC containing a 595 Shift Register, LED Bar graph, and a bused resistor network. It combined these three parts so there are only 5 pins, clock, latch, data, ground, and power. The outputs of the shift register are directly tied to the bargraph so that whatever input is sent in can be displayed out through LEDs.

\*See media section for picture

## Procedure

The code starts with a relativily basic main function, called reset, this simply sets up all the registers, and calls on any actually useful functions. The most important of these is the shiftout function.

ShiftOut Low Level(assembly)

|  |
| --- |
| Parts List |
| Morland Shift Register Bargraph |
| Atmel ICE |
| Arduino |

As with the high level, the first thing that needs to be done is to set the latch pin LOW with a cbi, thereby allowing data to be inputted serially on its data pin. Then the main interaction can start, first by entering a loop and pulling the clock pin LOW again using cbi. As well the loop counter needs to be incremented with inc. Now that the register is ready for its first bit, that bit has to isolated from its full byte, called displayData. This can be achieved by first mirroring the displayData into another register with the mov command, this allows the user to manipulate the copy, called displayDataC, without damaging the original value. Once displayDataC has been created it will undergo a logical and with the mask, pow2. pow2 should initially contain a value of one, so when it first ands it removes all of displayDataC other than the first bit, which will stay whatever it originally was. Now if displayDataC is greater than zero, it must mean the masked bit was one, and therefore a one should be put out through the data pin. If not it means masked bit was zero, and the data pin should be put LOW. This is accomplished by using a breq immediately after the and, so if displayDataC is zero it can jump a few lines downwards to the label zero, and execute a cbi on the data pin. If not it will pass through the breq and execute a sbi on the data pin, then a rjmp is needed to jump over the zero label and its contents. Now that the first bit has been presented, the clock can be set HIGH, “saving” it in place and the code can prepare for the second bit. This is achieved by multiplying the mask by 2(or bitshift left 1). The mul command in assembly takes two register inputs, multiplies their values and stores the product in r0:r1(low byte: high byte). So in the code this can be used to multiply pow2 and just2(a register holding 2). Then the mov command can be used a second time to save the product, currently at r0, to pow2. With that the code is ready to loop over again, if it needs to, so it will enter a cpi command, comparing to see if counter, and eight are equal. If they are not, the brne statement following it will return the code to the start of the loop and start this process over. If counter does equal eight however, the function simply needs to put the latch high, reset its registers and return.

The third is the basic function. This is used to output the basic counting display shown on the ACES website. This starts by loading a one into displayData with the ldi command and immediately enters the loop. After a short delay counter2 is incremented with inc and the data is shifted out with the previously defined function. Then, as before displayData, is multiplied by 2 with the mul command, so the product is stored in r0. So in order to display the “filling” functionality instead of just a single LED being on at a time, r0 will or with displayData, the result replacing displayData. And the function is finished off with a cpi comparing the counter to see if its reached eight, if not it loops back to the beginning and repeats now with two LEDs on, up until all eight are on. If it has finished however it clr’s counter2 and returns from the function.

The final function, called newfunc, starts with two ldi statements putting one into lbit and 128 into hbit. Then as the basic function it enters the loop and hits the rcall to delay. It then clears the utility register and loads full 255 into displayData. Then using the mov command, lbit is copied into util, which is then or’s hbit. displayData and util then and each other, setting all of displayData LOW other than the values in util. rcall shiftout sends the data out. Then lsl is used to bitshift lbit one to the left, and lsr is used to shift hbit one to the right. The counter is then incremented and compared to 8, as in basic. If counter2 hasn’t reached 8 yet it will loop back and continue the animation, if it has, it will clear counter2 and return.

## Code

\*Written in assembly language

; Date : February 2nd 2018

; Author : James Corley

; Project: Knight Rider

; Status : Working

#define clock PD7 ;location of clock in PORT\_

#define latch PD6 ;location of latch in PORT\_

#define data PD5 ;location of data in PORT\_

#define ground PD4 ;location of ground in PORT\_

#define power PD3 ;location of power in PORT\_

.def displayData = r17 ;data to be shifted out

.def counter = r18 ;shiftout loop counter

.def pow2 = r19 ;holds a power of 2

.def just2 = r20 ;2 in a register

.def displayDataC = r21 ;a copy of displayDataC for manipulation

.def util = r22 ;utility register

.def counter2 = r23 ;a function loop counter

.def hbit = r24 ;high bit (newfunc register)

.def lbit = r25 ;low bit (newfunc register)

;registers 28-30 use in delay function

#define ddr DDRD ;desried DDR\_

#define port PORTD ;desried PORT\_

reset:

ldi r16, 0b11111000 ;clk,latch,data,gnd,pwr - output

out ddr, r16 ;push out

sbi port, power ;pwr - HIGH

cbi port, ground ;gnd - LOW

ldi just2, 2 ;just2 = 2

ldi pow2, 1 ;pwr = 1

rcall basic ;calling basic function

rcall newfunc ;calling new function

rjmp reset ;loop back

shiftOut:

cbi port, latch ;latch - LOW

loop: ;enter loop

inc counter ;add 1 to counter

cbi port, clock ;clk - LOW

mov displayDataC,displayData ;copy data to displayDataC

and displayDataC, pow2 ;and out all other data

breq zero ;if displayDataC is not 0

sbi port, data ;set data pin HIGH

rjmp f1 ;jump over zero option

zero: ;if displayDataC is 0

cbi port, data ;set data pin LOW

f1:

sbi port, clock ;set clock pin HIGH

mul pow2,just2 ;multipy up by powers of 2

mov pow2,r0 ;store product back into pow2

cpi counter, 8 ;compare counter to 8

brne loop ;loop back to start

sbi port, latch ;latch = HIGH(shift out data)

clr counter ;clear counter

clr displayDataC ;reset displayDataC

ldi pow2, 0b00000001 ;reset pow2

ret

basic:

ldi displayData, 0b00000001 ;displayData = 1

loop2: ;enter loop2

rcall delay1s ;wait

inc counter2 ;add 1 to counter2

rcall shiftOut ;shiftout displayData

mul displayData, just2 ;r0:r1 = displayData\*2

or displayData, r0 ;displayData = displayData|r0

cpi counter2, 8 ;check if counter2 = 8

brne loop2 ;if ^ is false, loop, else pass over

clr counter2 ;reset counter2

ret

newfunc:

ldi lbit, 0b00000001 ;lbit = 0b00000001

ldi hbit, 0b10000000 ;hbit = 0b10000000

loop3: ;enter loop3

rcall delay1s ;wait

clr util ;clear utiliy register

ldi displayData, 0b11111111 ;displayData = 0b11111111

mov util, lbit ;copy lbit into util

or util, hbit ;util = util|hbit

and displayData, util ;displayData = displayData&util

rcall shiftout ;shiftout displayData

lsl lbit ;shift left lbit

lsr hbit ;shift right hbit

inc counter2 ;add 1 to counter2

cpi counter2, 8 ;check if counter2 = 8

brne loop3 ;if ^ is false, loop, else pass over

clr counter2 ;reset counter2

ret

delay1s:

; Generated by delay loop calculator

; at <http://www.bretmulvey.com/avrdelay.html>

; Delay 16 000 000 cycles

; 1s at 16.0 MHz

ldi r28, 82

ldi r29, 43

ldi r30, 0

L1: dec r30

brne L1

dec r29

brne L1

dec r28

brne L1

lpm

nop

ret

## Media

|  |  |
| --- | --- |
|  |  |
| Full Setup | Morland ShiftBar |
| YouTube Video:  <https://www.youtube.com/watch?v=JPQv7IsJBKo> | |

## Reflection

Overall this project was quite daunting initially. The relatively new language and new environment made starting off the project and getting comfortable difficult, and I spent the majority of the time on the Assembler Instruction page, looking up all the commands and their requirements. I ended up spending two hours coding one evening and only writing a total of 3 lines, or 12 words. However, as I became used to the instructions and tools it became dramatically easier. I worked out many of the issues I had in the environment and ended up coding relatively confidently near the end of the project. As with all the projects it had its fair share of frustration but in the end, I’m proud of the things I’ve learned and the code I’ve written.

# Project 21. The 4-Wire DC Fan

## Part A.

### Purpose

To research and synthsize information about the Sunon 4 wire fan.

### Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/1819/AssemblyTasks.html>

Sunon DataSheet

<http://mail.rsgc.on.ca/~cdarcy/PDFs/Sunon12VDCFanSpec.pdf>

Intel Paper on 4 Wire Fans

<http://mail.rsgc.on.ca/~cdarcy/PDFs/4WirePWMFans.pdf>

### Theory

#### Basic Interaction

|  |  |
| --- | --- |
| Colour | Function |
| Black | GND |
| Red | PWR |
| Yellow | Sense |
| Blue | Control |

The fan has 4 pins given to the user. These pins follow the colour scheme Black = GND, Red = PWR, Yellow=Sense, Blue=Control. The PWR pin requires 12V±1.2V of around 2 Amps (these values will vary upon startup).

#### Control Pin

|  |
| --- |
|  |
| \*Note – If no signal is presented, the fan will operate at the maximum rpm |

The control pin is a PWM input pin for the fan, that is used to control its speed. Seeing as it runs on PWM it has some important parameters, 0V-0.4V = LOW and 2.8V-6V = HIGH (recommended 5V) with an absolute maximum current draw of around 5mA. As well the frequency of the square wave used can be from 22 kHz – 28kHz however, 25kHz is recommended. To see how the fan interacts with this PWM signal, see the chart to the right.

#### analogWrite() Limitations

The primary issue with this function is its seemingly fixed PWM frequency. Because on the Arduino reference page it is rated to provide a signal with a frequency of 490Hz from all pins other than 5 and 6 which can have a frequency of 980Hz. These values are obviously far below the desired 25kHz of the fan. Ultimately these limitations are set in place because of the Arduino timers, the prescalars placed upon them, and the length of each of the cycles. For example, the length of a PWM cycle on the Arduino UNO is 510 clock pulses, so the 16MHz will be divided by 510. Then another arbitrary scalar will scale the new 31372.55 down by a factor of 64 to around 490, which is the frequency available. The same is true for pins 5 and 6 but the cycle length is only 256.

#### Achieving 22 kHz – 28kHz PWM on an Arduino

So, while the basic high-level commands will not work, there are still things that can be changed to fix the PWM output frequency. All that needs to be done is some low-level timer register manipulation to change the prescalars so that the values are closer to the desired range. The important caveat to this is that by changing these values, any function that relies on the timers, such as delay() or millis() will be changed as well, and will not function as they originally did since the timers have been fundamentally changed.

#### Sense Pin

|  |
| --- |
|  |

The sense pin is an output pin from the fan that gives data about the fans speed. This pin in open collector, which means in order to read the correct signal a pull-up resistor has to be used, as shown on the diagram to the right. The signal itself is rather simple, every two pulses from the pin translate to one full rotation of the fan.(Frequency:Rotations 2:1) This can be given by the equation:

Where *ppm* = pulses per minute

*rpm* = rotations per minute

#### Baisc Function

There are some important notes about the physical functioning of the fan to keep in mind. The fans maximum rpm is 4800 rpm ±10% if it is being given the recommended 12V. The direction of rotation of the blade, as well as the direction of air flow, is shown on the top of the fan. In essence, when looking at the fan from the front(at the green label) the blade will rotate clockwise, and the flow of air will be straight at you(See the media for a picture of this). As well the fan must be mounted properly or it wil cause vibrations and a resulting noise.

### Media

|  |  |
| --- | --- |
|  |  |
| Front View | Full View(Red = Air Flow Information) |
|  |  |
| Block Diagram | Top View(Green arrow = green label) |

### Reflection

The two documents were fairly conclusive and I feel relatively confident working with the fan from now on. As well I tried to make this document as easy to read and comprehend because it will probably be the main resource I return to when I get stuck throughout the project. Overall a nice change of pace from the typical submissions.

# Project 22. ACEQuest 2.0

## Purpose

To create a small game in a 3D printed case that is driven with a completely SMD PCB.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/1819/ISPs.html>

Bill Grundmann’s Blog

<https://billgrundmann.wordpress.com/2009/03/03/to-use-or-not-use-writedigital/>

## Theory

### Daisy-chaining

Daisy-chaining is a method of controlling several shift registers through a single data pin. This is achieved by using the shift registers output pin. If the output of one is tied into the input of another, it essentially makes it into one two byte shift register. Now, after the first byte has been sent into the data pin of the first register, instead of pulling the latch low and shifting that data out, another full byte can be sent into the same pin. When this is done it shifts the first byte through its output pin into the second register. It is important to note, while the clocks can be tied together, the latch pin of the second register still has to be interacted with separately, only once the first byte has been entered can it be pulled LOW for data to be sent in. If both latches are pulled LOW at the same time it will cause the data sent in on each clock cycle to be random, since there is no defined data there.

### Speed in Arduino

High-level Arduino C code is a very powerful tool and in many cases works perfectly. However, one area that this language is rather weak, is speed. Commands such as digitalWrite() or shiftOut() though they work, take far longer to execute than they necessarily need too. For example in Bill Grundmann’s Blog(see reference), he found that given,

digitalWrite(pin, HIGH)

digitalWrite(pin, LOW)

It caused the given pin, to be HIGH for 53 clock cycles and then LOW for 55 cycles. Whereas a lower level approach that uses port manipulation, as shown

PORT\_ |= pin

PORT\_ &= ~pin

Is HIGH for around 2 cycles, and LOW for around 4 cycles. This is obviously a dramatic difference, roughly a factor of 26 times slower and it is due to the fact that the low-level commands can be directly translated by the GNU compiler to assembly commands SBI and CBI. Whereas the high-level command has way more to it to make it more versatile and portable. While normally timing like this isn’t an issue, if the user is trying to achieve POV(see Project 9 explanation) these commands will be too slow and the display will not work properly.

### Timer Interrupts

|  |  |  |
| --- | --- | --- |
| Timer | Size | Used by |
| 0 | 8bit | delay(), millis(), micro(), PWM pins 5&6 |
| 1 | 16bit | servo(), PWM pins 9&10 |
| 2 | 8bit | tone(), PWM pins 11&13 |

A timer interrupt is essentially an internal interrupt that will alert the system at a given interval and execute a small function. They run off of one of three timers, appropriately called timer0, timer1 and timer2, see a summary of these timers to the right. As well the “Used by” section shows which functions, libraries or PWM pins use each timer. So when changing a timer, be aware of the ramifications it will have on these areas. These timers take a pulse from the crystal, each time incrementing by one. Once they have reached the max size of the timer they will overflow and reset to zero. Before this pulse can be used several registers need to be set.

TCCR\_A is used to set the mode. There are 16 different modes but it will generally be set to

|  |
| --- |
|  |
| Block Diagram |

normal mode by putting the whole register LOW.

TCCR\_B can be used to scale the counting to get a desired timing for the interrupts. The prescalers.h file can generally help with as it provides aliases for each of the scaling values.

Finally the TIMSK\_(Timer Interrupt Mask) register enables and disables interrupts. While there are three different interrupts that can be set, the focus will be on the TOIE\_(Timer Interrupt Overflow Enable). This interrupt simply triggers when the timer reaches its max size and overflows. So at default when this happens it will call ISR(TIMER\_\_OVF\_vect). As with all interrupt service routines, this function must be kept short and concise. If it is too long and another interrupt is called while it is inside this one it can cause some complications with the code.

Other notes

With this, there are a few more things to keep in mind. Firstly when configuring interrupts in the setup it is safe to use

cli();

\*\*configure interrupt registers\*\*

sei();

This will disable the global interrupt system while the registers are being changed so that no interrupts occur until they have been configured properly. Secondly, if additional scaling is required there are two methods of doing this. To increase the speed of the interrupts simply give the timer a starting value after it overflows so it has a shorter distance to count and thereby a shorter time to do so. Do this by making TCNT\_ = a, where a > 0 after each interrupt. The second method is a way of slowing the interrupts down. This method is rather basic but has a high degree of flexibility, more than the prescalars. Simply have a statement like

overflowCount++;

if (overflowCount = a){

\*\*interrupt code\*\*

}

This allows the user to have interrupts happening often, but only actually acting on them some of the time.

## Procedure

### Hardware Function

|  |
| --- |
| Parts List |
| 64xRGB LED |
| 9x Red LED |
| 16x Green LED |
| 16 MHz Crystal |
| 2x 22pF |
| ATmega328p |
| 6x 595 Shift Register |
| 3x 330Ω Isolated Resistor Network |
| 10000Ω Isolated Resistor Network |
| USB Mini B Input |
| FFC Cable Input |
| 4x Push Button |

The heart of this circuit is an ATmega328p. From this point, there are four main important

systems. The first is the primary RGB matrix. This is driven through four daisy-chained 595 shift registers. This microcontroller is then completely used to control four 595 shift registers which drive the primary 8x8 RGB matrix. The first register of the four runs through an isolated resistor array and provides the power to the matrix since each of the LEDs is common cathode. Following in order at the red green and blue shift registers. In order to activate any of the specific LED colours, simply put that bit in the register LOW, and it will ground that LED lighting it up. The second is the Mini-map matrix. This is a 4x4 monocolour matrix driven through a single 595 shift register. It uses half of a resistor array for its cathode pins. And since only a single LED needs to be on at a time it does not need POV. The third is the health array which shows the users current health. It is an 8 LED array driven by a single 595 and resistor array. And finally is the input system, the four directional buttons and the single utility button all tied on one side to 5V and then on the other to ATmega328p pins and then through a resistor to GND.

### Game Function

#### Basic game

There are four main colours that follow the table to the right. Each colour has something it represents, which will change on how the player interacts with it. For example, blue LEDs signify water and will prevent the player from moving on to them, whereas green LEDs simply mean walkable land. The system gets more complex when RED is introduced, the enemy. While the player can walk on to an enemy's location using his directional buttons, it will cause a transition to a minigame(see section). Finally, if the user moves to any of the borders on the current map and tries to move farther, it will transition to the next map in that direction if it exists. If it doesn’t, nothing will happen. The location of which map the user currently on is shown by the minimap to the right.

#### Mini Game

In order to “defeat” an enemy, the user has survived for one minute in the mini-game. Every second from a random border row or column a “missile” will spawn and all currently existing missiles will move one space. Each missile has a direction associated with it and it will follow that direction until it moves off of the map and disappears. The goal of the player is to move their character in a way to avoid all of the missiles. If they can do this for the full sixty seconds, the game will end and they will return to the main game with the enemy gone. If they fail to avoid the missiles and one hits them, combat will end and they will lose a health point. Their health points are shown to the upper right of the console as a red LED array. If all ten health points are lost the game is over.

### Code Function

|  |  |  |
| --- | --- | --- |
| Basic mstrMatrix Key | | |
| Value | Colour | Meaning |
| 0 | Blue | Water |
| 1 | Green | Land |
| 2 | Multi | Character |
| 4 | Red | Enemy |

#### mstrMatrix

The code is structured around one main 8x136 matrix, in this case, called mstrMatrix[][].  
mstrMatrix[][] is split up into 16 unique 8x8 maps, as well as one blank one which is used for the minigame. The currentMap variable is used to show which. It has a value from 0-16 and is used to scale as shown mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)]. When a map is currently being used, the values in each of its cells correspond to a particular meaning shown in the chart to the right. These values, determine how the user will interact with that particular space if they try to move there.

#### Output

updateDisplay()

In order to actually display the mstrMatrix[][] to the user, it has to go through a middleman function, called updateDisplay(). This function essentially reads through each cell in mstrMatrix[][] and converts the values into three arrays redArray[], greenArray[], and blueArray[]. These arrays are all 8 bytes long, so they can be thought of as matrixes of bits. And that is how they are used in the code, each of these bytes corresponds to the ground pins of a row of the matrix since these are the bytes that are loaded into the shift registers. So by putting a bit low in one of these arrays, it will light up that particular LED on the actual RGB LED matrix.

displayRGB()

However, updateDisplay() simply manipulates the data in software. In order to actually light up the matrix, it requires one more function, displayRGB(). This function essentially just runs through each of the colour arrays and loads their values into the shift registers. And since each 595 is daisy-chained it means that all four bytes(including the power 595 byte) can be loaded in at once and all latched out at the same time.

#### Input

buttonPress()

This function is one of the most vital in the code and is the sole method of input to the game. It is called when an external interrupt is triggered. Once this happens it checks all the button inputs to see which button was actually pressed and if the space in that direction is not water. If both are true then it moves there. However, it also checks if the player has reached the border of their current map. If they have, and if the button they pressed was in that direction it will then change maps and move to the map in that direction, jumping through the mstrMatrix[][]. The last thing it checks is if it is currently on an enemy. If so it calls the fightEnemy() function and combat begins.

#### Combat

|  |  |
| --- | --- |
| gameMode = | Game |
| 0 | Default |
| 1 | Mini-Game(combat) |

fightEnemy()

fightEnemy() does four important things. It sets gameMode to one, saves the currentMap into returnMap and then sets currentMap to 16, the blank map. And saves the players return location into returnPlyrLocY and returnPlyrLocX.

Now that the gameMode has been updated, the loop will work slightly differently. Now there are two if statements that will be checked each loop, firstly if a timer interrupt has occurred, and secondly if the player has been hit. Before this, a timer interrupt has been occurring but it has just been ignored and triggerTimer variable has been set to zero. However, now if a timer interrupt occurs it means one second has passed and it will trigger several things. Firstly it will create a new missile(see missile section) on a random border location with a random velocity. And secondly, it will call moveMoveMissiles(). All this does is move through the mstrMatrix[][] and move any missiles it finds in their desired directions. After 60 missiles have been created, all of which the player has avoided, the mini-game will end and the player has “won”, and returnFromFight() will be called. If the player was hit, they will call returnFromFight() all the same but the player will lose one health point. A fact which will be shown on their health bar.

Missiles

|  |  |  |
| --- | --- | --- |
| Missile mstrMatrix[][] key | | |
| Value | Direction | Alias |
| 16 | Up | missile\_U |
| 32 | Down | missile\_D |
| 48 | Left | missile\_L |
| 64 | Right | missile\_R |
| 80 | Up/Left | missile\_UL |
| 96 | Up/Right | missile\_UR |
| 112 | Down/Left | missile\_DL |
| 128 | Down/Right | missile\_DR |

Missiles are new objects in the mstrMatrix[][]. And similar to all the other objects they have a unique number associated with them. Unlike all the other objects, however, there are many different types of missiles(shown to the right). This is because their value is associated with their direction of travel.

returnFromFight()

This is essentially the inverse of the fightEnemy(). It will return the player to their original map and location, set gameMode to zero and remove the enemy.

## Code

//  Author: James Corley

// Project: ACEQuest 2.0

//    Date: March 9th 2019

#include "prescalers.h"

#include <PinChangeInt.h>

//PORTD

uint8\_t SR\_L\_PWR = 1 << PD0;

uint8\_t SR\_L\_R = 1 << PD1;

uint8\_t SR\_L\_G = 1 << PD2;

uint8\_t SR\_L\_B = 1 << PD3;

uint8\_t SR\_L\_MP = 1 << PD4;

uint8\_t SR\_L\_HP = 1 << PD5;

uint8\_t SR\_CLK = 1 << PD6;

uint8\_t SR\_RGBData = 1 << PD7;

//PORTB

uint8\_t SR\_HPData = 1 << PB0;

uint8\_t SR\_Enable\_RGB = 1 << PB1; //output enable

//PORTC

uint8\_t B\_UP = 1 << PC0;

uint8\_t B\_DOWN = 1 << PC1;

uint8\_t B\_LEFT = 1 << PC2;

uint8\_t B\_RIGHT = 1 << PC3;

uint8\_t B\_UTIL = 1 << PC4;

uint8\_t SR\_MPData = 1 << PC5;

//Button Interrupt Variables

uint8\_t interruptPin = 9;

boolean triggerButton = 0;

//Timer Interrupt Variables

boolean triggerTimer = 0;

uint8\_t overflowCount = 0;

uint8\_t timeCounter = 0;

//Map Variables

uint8\_t returnMap = 0;

uint8\_t currentMap = 0;

//currentMap KEY:

//   12 13 14 15

//   8  9  10 11

//   4  5  6  7

//   0  1  2  3

uint8\_t miniMap = 0;

uint8\_t miniMapcounter = 0;

uint8\_t healthBar = 0;

uint8\_t utility;

uint8\_t gameMode;              //0 = default, 1 = mini-game

uint8\_t plyrLocX = 0;

uint8\_t plyrLocY = 0;

uint8\_t returnPlyrLocX = 0;

uint8\_t returnPlyrLocY = 0;

//All this data is inverted, to put an LED HIGH set it too 0

uint8\_t redArray[8] = {0b11111111,

                      0b11111111,

                      0b11111111,

                      0b11111111,

                      0b11111111,

                      0b11111111,

                      0b11111111,

                      0b11111111

                     };

uint8\_t greenArray[8] = {0b11111111,

                        0b11111111,

                        0b11111111,

                        0b11111111,

                        0b11111111,

                        0b11111111,

                        0b11111111,

                        0b11111111

                       };

uint8\_t blueArray[8] = {0b11111111,

                       0b11111111,

                       0b11111111,

                       0b11111111,

                       0b11111111,

                       0b11111111,

                       0b11111111,

                       0b11111111

                      };

#define water       0b00000000

#define land        0b00000001

#define character   0b00000010

#define enemy       0b00000100

#define missile\_U   0b00010000

#define missile\_D   0b00100000

#define missile\_L   0b00110000

#define missile\_R   0b01000000

#define missile\_UL  0b01010000

#define missile\_UR  0b01100000

#define missile\_DL  0b01110000

#define missile\_DR  0b10000000

uint8\_t mstrMatrix[8][136] =

{ {0, 0, 1, 1, 1, 1, 0, 0},

 {0, 0, 1, 1, 0, 0, 0, 0},

 {0, 1, 1, 1, 1, 1, 0, 0},

 {0, 1, 1, 0, 1, 1, 1, 0},

 {0, 1, 1, 1, 1, 1, 0, 0},

 {0, 1, 1, 0, 1, 1, 1, 0},

 {1, 1, 1, 0, 1, 1, 1, 1},

 {2, 1, 0, 0, 0, 0, 0, 0}

};

void setup() {

 //Declare DDR's For I/O

 DDRD |= (SR\_L\_PWR | SR\_L\_R | SR\_L\_G | SR\_L\_B | SR\_L\_MP | SR\_L\_HP | SR\_CLK | SR\_RGBData);

 DDRC |= SR\_MPData;

 DDRC &= ~(B\_UP | B\_DOWN | B\_LEFT | B\_RIGHT | B\_UTIL); //All buttons are Input

 DDRB |= SR\_HPData | SR\_Enable\_RGB;

//change interrupt pin and attach

**PCintPort**::attachInterrupt(interruptPin, interruptButton, HIGH);

 //Configure Timer Interrupt

 cli();                        //disable global interrupt system while configuring

 TCCR1A = 0;                   //normal mode

 TCCR1B = T1ps256;             //scale back to 0.5Hz

 TIMSK1 = 1 << TOIE1;          //enable Timer1 overflow interrupt

 sei();                        //enable glabl interrupt system when finished

}

void interruptButton()

{

 triggerButton = 1;

}

ISR(TIMER1\_OVF\_vect) {

 if (overflowCount)

 triggerTimer = 1;              //triggerTimer will go high at 1Hz

 overflowCount++;

}

void loop() {

 if (triggerButton)

 {

   triggerButton = 0;  //If a button was pressed

   buttonPress();

 }

if (!gameMode)

 {

   triggerTimer = 0;

   //updateDisplay();             //update screen

   displayRGB();                  //display screen

 } else if (gameMode) {                                    //if in Combat

   if (triggerTimer)

   {

     overflowCount = 0;                //reset overflowCount

     triggerTimer = 0;                 //reset trigger

      //Create a missile at a random border location with a random direction

     mstrMatrix[7\*random(0,2)][7\*random(0,2)+(currentMap\*8)]=(random(1, 9)<<4);

     moveMissiles();                  //update missile locations

     timeCounter++;                   //add one to counter

   }

   //If the high and low nibble of mstrMatrix[][] are HIGH, the player has been hit

   if ((mstrMatrix[plyrLocX][plyrLocY]&0xF0)& (mstrMatrix[plyrLocX][plyrLocY] & 0x0F))

   {

     healthBar >> 1;                   //subtract one health

     displayHealthBar();               //display new health

     returnFromFight();                //return to default game

     //if timeCounter = 60, return to normal gameMode

   } else if ((timeCounter & 0b00100000) & (timeCounter & 0b00010000) & (timeCounter & 0b00001000) & (timeCounter & 0b00000100)) {

     returnFromFight();

   }

 }

}

void displayRGB() {

 for (uint8\_t i = 0; i < 8; i++)

 {

   PORTB &= ~SR\_Enable\_RGB;//Enable output

   PORTD &=~SR\_L\_B;                             //blue latch LOW

   shiftOutNew(SR\_RGBData, SR\_CLK, MSBFIRST, blueArray[i], 'D');//shiftout blue data

   PORTD &= ~SR\_L\_G;                     //green latch LOW

   shiftOutNew(SR\_RGBData, SR\_CLK, MSBFIRST, greenArray[i], 'D');//shiftout green data

   PORTD &= ~SR\_L\_R;                                             //red latch LOW

   shiftOutNew(SR\_RGBData, SR\_CLK, MSBFIRST, redArray[i], 'D');  //shiftout red data

   PORTD &= ~SR\_L\_PWR;                                           //pwr latch LOW

   shiftOutNew(SR\_RGBData, SR\_CLK, MSBFIRST, 1 << i, 'D');       //shiftout pwr data

   PORTD |= (SR\_L\_PWR | SR\_L\_R | SR\_L\_G | SR\_L\_B);//latches HIGH

   PORTB |= SR\_Enable\_RGB;//Disable output

 }

}

void updateDisplay()

{

 for (uint8\_t y = 0; y < 8; y++) {

   for (uint8\_t x = 0; x < 8; x++) {

     //Default make it blank(put all LEDS HIGH)

     redArray[y] |= (0b10000000 >> x) ;//| redArray[y];

     greenArray[y] |= (0b10000000 >> x); //| greenArray[y];

     blueArray[y] |= (0b10000000 >> x) ;//| blueArray[y];

     if (!mstrMatrix[x][y + (currentMap \* 8)]) {              //if theres a 0(water)

       blueArray[y] = ~(0b10000000 >> x)&blueArray[y];          //set blue array

     } else if(mstrMatrix[x][y + (currentMap \* 8)]&land){ //if theres a 1(land)

       greenArray[y] = ~(0b10000000 >> x)&greenArray[y];    //set green array

     } else if(mstrMatrix[x][y + (currentMap \* 8)]&character){//if theres a 2(player)

       redArray[y] = ~(0b10000000 >> x)&redArray[y];           //set all arrays

       greenArray[y] = ~(0b10000000 >> x)&greenArray[y];

       blueArray[y] = ~(0b10000000 >> x)&blueArray[y];

     } else if mstrMatrix[x][y + (currentMap \* 8)]&enemy){ //if theres a 4(enemy)

       redArray[y] = ~(0b10000000 >> x)&redArray[y];             //set red array

     }

   }

 }

}

void buttonPress()

{

//If up button is pressed and there is a space above.

 if ((DDRC&B\_UP>0)&(mstrMatrix[plyrLocX][plyrLocY+1+(currentMap\*8)]!=(water))) {

   mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)] = 1;//make loc in matrix

   plyrLocY++;   //update plyrLoc

   mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)] = 2;//update matrix

 }else if((DDRC&B\_DOWN>0)&(mstrMatrix[plyrLocX][plyrLocY-1+(currentMap\*8)]!=water)){

//down pressed

   mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)] = 1;//make loc in matrix

   plyrLocY--;    //update plyrLoc

   mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)] = 2;//update matrix

 }else if((DDRC & B\_LEFT > 0) & (mstrMatrix[plyrLocX - 1][plyrLocY] != water)) {

//left pressed

   mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)] = 1; ;//make loc in matrix

plyrLocX--;   //update plyrLoc

   mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)] = 2; //update matrix

 }else if((DDRC&B\_RIGHT>0)&(mstrMatrix[plyrLocX+1][plyrLocY+(currentMap\*8)]!= water)){

//right pressed

   mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)] = 1; //make loc in matrix

   plyrLocX++;//update plyrLoc

   mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)] = 2; //update matrix

 }

 if ((plyrLocY < 0) & (currentMap < 12)) {       //If its past the top->next map

   currentMap++;                                 //updates map

   plyrLocY = 0;                                 //updates location

   displayMiniMap();

 } else if((plyrLocY > 8) & (currentMap > 3)) { //If past bottom->next map below

   currentMap--;                                 //updates map

   plyrLocY = 7;                                  //updates location

   displayMiniMap();

 } else if((plyrLocX < 0) & ~(currentMap % 4)) {//If past left->next map left

   currentMap -= 4;                              //updates map

   plyrLocX = 0;                                 //updates location

   displayMiniMap();

 } else if((plyrLocX > 8) & ~((currentMap - 3) % 4)){//If past right->next map right

   currentMap += 4;

   plyrLocX = 7;

   displayMiniMap();

 }

 if ((!gameMode) & (mstrMatrix[plyrLocX][plyrLocY + (currentMap \* 8)]&enemy)) {  //if in default mode and on an enemy, fight it.

   fightEnemy();

 }

}

void displayMiniMap()

{

 shiftOutNew(SR\_MPData, SR\_CLK, MSBFIRST, miniMap, 'C');

}

void displayHealthBar()

{

 shiftOutNew(SR\_HPData, SR\_CLK, MSBFIRST, healthBar, 'D');

}

//(dataPin,ClkPin,MSBFIRST/LSBFIRST,data,PORT)

void shiftOutNew(uint8\_t d, uint8\_t c, uint8\_t dir, uint8\_t value, char port) {

 uint8\_t mask;

 for (uint8\_t i = 0; i < 8; i++) {

   PORTD  &= ~SR\_CLK;                             //pull clock bit low

   if (dir)                                       //MSBFIRST?

     mask = value & (0x80 >> i);                  //data MSBFIRST

   else

     mask = value & (1 << i);                     //data LSBFIRST

   if (mask) {

     //check PORT of data pin

     switch (port) {

       case 'C':

         PORTC |= SR\_MPData;

         break;

       case 'D':

         PORTD |= SR\_RGBData;

         break;

       case 'B':

         PORTB |= SR\_HPData;

         break;

     }

   } else {

     //check PORT of data pin

     switch (port) {

       case 'C':

         PORTC &= ~SR\_MPData;

         break;

       case 'D':

         PORTD &= ~SR\_RGBData;

         break;

       case 'B':

         PORTB &= ~SR\_HPData;

         break;

     }

   }

   PORTD |= SR\_CLK;                             //clock in this bit

 }

}

uint8\_t fightEnemy()

{

 transition();     //display transition

 gameMode++;       //update game mode

 returnMap = currentMap;

 currentMap = 16;  //load the combat map

 returnPlyrLocX = plyrLocX;          //reset x location

 returnPlyrLocY = plyrLocY;          //reset y location

}

void moveMissiles()

{

 for (uint8\_t y = 0; y < 8; y++) {

   for (uint8\_t x = 0; x < 8; x++) {

     int8\_t moveValX = 0;

     int8\_t moveValY = 0;

     //check missile directions and move them accordingly

     if (~(mstrMatrix[y][x]&missile\_U))

     {

       moveValY = 1;

     } else if (!(mstrMatrix[y][x]&missile\_D)) {

       moveValY = -1;

     } else if (!(mstrMatrix[y][x]&missile\_L)) {

       moveValX = -1;

     } else if (!(mstrMatrix[y][x]&missile\_R)) {

       moveValX = 1;

     } else if (!(mstrMatrix[y][x]&missile\_UL)) {

       moveValY = 1;

       moveValX = -1;

     } else if (!(mstrMatrix[y][x]&missile\_UR)) {

       moveValY = 1;

       moveValX = 1;

     } else if (!(mstrMatrix[y][x]&missile\_DL)) {

       moveValY = -1;

       moveValX = -1;

     } else if (!(mstrMatrix[y][x]&missile\_DR)) {

       moveValY = -1;

       moveValX = 1;

     }

//Move value to next location

     mstrMatrix[y + moveValY][x + moveValX] = mstrMatrix[y][x];

     mstrMatrix[y][x] = 0;                         //Make previous value = 0

   }

 }

}

void returnFromFight()

{

 gameMode = 0;                                 //return to normal gameMode

 currentMap = returnMap;                       //reset map

 plyrLocX = returnPlyrLocX;          //reset x location

 plyrLocY = returnPlyrLocY;          //reset y location

 mstrMatrix[plyrLocX][plyrLocY] = 2; //replace enemy with you

}

## Media

|  |  |
| --- | --- |
|  |  |
| Board | Full Case |
|  |  |
| Side View(FFC+USB Input) | ATmega328p Pinout |
|  | |
| Full Schematic | |
| YouTube Video  <https://www.youtube.com/watch?v=7ZpW82ADNWE> | |

## Reflection

This was one of the more ‘out there’ and personal ideas and I had created. And that was reflected in the fact that a lot of the brainstorming and developing of it in Eagle and Arduino was enjoyable. It’s been a project I’ve thought about this sort of idea since Reid did his and it was nice to be able to service that. And despite the complications near the end and the fact the PCBs weren’t able to function, I’m still proud of what I was able to make. I have little doubt that the PCB would do what it was designed to do and I could format my code to fit that. There is a point when getting a project working isn’t what it was about anymore and I think this is one of those cases. Either way, it’s been a very short, yet somehow very long term and I am thoroughly ready for a break. Hopefully, the spring will bring some nice weather for our final term.

# Project 23. Flex Page

## Purpose

To create a small SMD flex PCB contained in a printed version of my DER. The PCB should have a four button password that upon being entered correctly would present, in binary, values corresponding to the page, line, and letter of a “secret message”.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/1819/Tasks.html#FP1>

## Theory

### PCB Part Mounting

There are two ways of mounting parts on PCBs, surfacemount and throughhole(THT). All projects before this, excluding ACEQuest, were mounted using throughole technology. The reason being it is far easier to use and favours simple hobby projects heavily because all the parts are usable on breadboards as well mountable on PCBs. The main idea is each part has a set of pins that stick out from it. These pins fit through holes in the PCB where they are bonded together with solder(There is a brief districption in Project 2. The Automatic Night Light, about how to solder these parts). THT technology favours the human’s ease of access over the IC’s actual requirements. So, while this technology is very powerful for prototypes and hobby projects, if a project has to be seriously compact and intricate a new approach is nessearily.

### Surfacemount(SMD)

Surfacemount mounting is the response to technologies migration towards smaller devices. It is the technology that is used in computers and phones because it is simply far more efficient in terms of space. In industrial cases the parts are so small that they are only able to be used by machines in a factory setting. However, while the parts can get remarkably small, some of them are kept slightly larger so they can still be used by humans and mounted manually.

### Soldering Surfacemount Parts

The basic idea goes as follows:

1. Apply solder paste to PCB pads
2. Place parts carefully onto their respective locations on the board
3. Cook board to liquify solderpaste
4. As boards cool solderpaste hardens

However, there are several ways of going about this. Firstly applying the solder paste manually with a toothpick or some other small instrument is possible, but it can be very hard, especially for some of the smaller parts. An easier method is to us a stencile. A stencile is simply a metal sheet with holes where each of the pads are on the board. So if it is lined up correctly the solderpaste can simply be squeegeed on. This makes it dramatically easier but it does cost more from the PCB provider. And secondly, in order to cook the boards a device called a reflow oven is very useful. This is simply a small oven that has some advanced temperature control and timing. This allows the board to be heated and cooled in exactly the right way to solidify the connections.

### Pin Change Interrupts

A pin change interrupt is an interrupt available on all none PWR/GND, pins of traditional Atmel microcontrollers such as the ATmega328p, ATtiny84 and ATtiny85. Unlike the designated external interrupt pins like int0 or int1, pin change interrupts have a single Interrupt Service Routine(ISR) shared by all the pins on a given port. So if any pin on a port is interrupted it will trigger the entire port’s ISR. As with all ISR’s they should be kept short, especially if there are several ways of triggering it, have an ISR called while its already inside it is not good.

## Procedure

The circuit starts with four buttons. Each button is tied to a resistor and then the a I/O pin of the ATtiny84. However, unlike all of the previous projects, the interrupt system is not controlled by a single designated interrupt pin. Instead, all the buttons are attached to pin change interrupts. However, since all of the buttons trigger to the same ISR, that means there has to be a second set of testing to determine which buttons was pressed. For this simply checking the PIN register to see which pin is HIGH works. More than this, the value of the pin register can be immediately loaded into the passArray[]. This is because each button is essentially mapped to a bit, in a byte. So button one is represented as 0b00000001 and button two 0b00000010, etc. And since the buttons as connected too PA0-PA3. The value PINA returns should be exactly what button was pressed in binary.

So once a button is pressed, in the ISR the global interrupt system(GIS) is disabled and a trigger is set. This then triggers an if statement in the loop, and if the correct password has not been set, PINA is immediately loaded in to passArray[index] and index is incremented, where index is initially = 0. This cycle will repeat until another if statement in the loop, that checks if the index

|  |
| --- |
| Parts List |
| 595 Shift Register |
| 330Ω Resistor Network (Isolated) |
| 8x LEDs |
| ATtiny84 |
| 4x Pushbuttons |
| 4x 10kΩ Resistors |
| ISP Programmer |

is equal to the password length, becomes true. Once this becomes true it means that the button has been pressed four times, and the inputted password should be compared to the set password. Immediately, the GIS is disabled, index is set to zero, and pass, a variable to determine if the correct password has been entered, is made true. Then using an if statement within a for loop each cell in the the inputed password(passArray[]), is compared to the set password(password[]). If any of the cells are different then pass if pulled back false, the GIS is reenabled and the code will start over. However, if all the cells are the same the code will simply move over this, reenabling the GIS and returning to the loop, this time with pass = true. Now that the correct password has been entered the “secret” binary data can be presented to the user. To stop displaying the data the user can simply hold down the “1” button, after a few seconds it will reset the board.

## Code

//  Author: James Corley

// Project: Flex PCB

//    Date: May 20 2019

//  Status: Working

uint8\_t SR\_Data = 1 << PB0;

uint8\_t SR\_Latch = 1 << PB2;

uint8\_t SR\_Clk = 1 << PB1;

uint8\_t Btn1 = 1 << PA0;

uint8\_t Btn2 = 1 << PA1;

uint8\_t Btn3 = 1 << PA2;

uint8\_t Btn4 = 1 << PA3;

uint8\_t datapin = 10;

uint8\_t clkpin = 8;

uint8\_t latchpin = 9;

uint8\_t passlength = 4;

uint8\_t passArray[4];

uint8\_t password[] = {0b00000010, 0b00000010, 0b00000100, 0b00000100};

uint8\_t index = 0;

uint8\_t count = 0;

boolean trigger = false;

boolean change = false;

boolean pass = false;

uint8\_t displayData[2][3] = {{0b00000001, 0b00000011, 0b00000111},

                            {0b10000000, 0b11000000, 0b11100000}};

void setup() {

 DDRB |= SR\_Data | SR\_Latch | SR\_Clk;

 DDRA &= ~(Btn1 | Btn2 | Btn3 | Btn4);

 cli();

 GIMSK  |= 1 << PCIE0;                                               //Enable PCINT7-0 Interrupts

 PCMSK0 |=  (1 << PCINT0 | 1 << PCINT1 | 1 << PCINT2 | 1 << PCINT3); //Enable interrupts on given pins

 sei();

}

void loop() {

 if (trigger) {

   trigger = false;

   if (!pass) {

     count++;

     if (count == 2) {       //ignore 1st interupt

       count = 0;

       passArray[index] = PINA & 0x0F; //store button into passArray[]

       flash();                          //show value through LEDs

       index++;                          //incremnt index

       sei();

     }

   } else {

     sei();

     pass = !(0b00000001 & PINA); //if button 1 is pressed exit

  digitalWrite(latchpin, LOW);

  shiftOut(datapin, clkpin, LSBFIRST, 0b00000000);

  digitalWrite(latchpin, HIGH);

   }

 }

 if (!pass) {

   if (index == passlength) {        //has a full password been entered?

     cli();                          //disable button input

     index = 0;                      //set index back to zero

     pass = true;                    //at default make the password correct

for (uint8\_t i = 0; i < passlength; i++) {

       digitalWrite(latchpin, LOW);

       shiftOut(datapin, clkpin, MSBFIRST, passArray[i] & 0x0F);

       digitalWrite(latchpin, HIGH);

       delay(1000);

       digitalWrite(latchpin, LOW);

       shiftOut(datapin, clkpin, MSBFIRST, 0);

       digitalWrite(latchpin, HIGH);

       delay(100);

       digitalWrite(latchpin, LOW);

       shiftOut(datapin, clkpin, MSBFIRST, password[i] & 0x0F);

       digitalWrite(latchpin, HIGH);

       delay(1000);

       if (passArray[i] & 0x0F != password[i] & 0x0F) {

//check if the password wrong^

         pass = false;                  //password is wrong, try again

         digitalWrite(latchpin, LOW);

         shiftOut(datapin, clkpin, MSBFIRST, 0b10101010);//Error

         digitalWrite(latchpin, HIGH);

         delay(1000);

       }

     }

     sei();

   }

 } else {

   sei();

   //output to user

   for (uint8\_t y = 0; y < 2; y++) {

     for (uint8\_t x = 0; x < 3; x++) {

       digitalWrite(latchpin, LOW);

       shiftOut(datapin, clkpin, MSBFIRST, displayData[y][x]);

       digitalWrite(latchpin, HIGH);

       delay(1000);

     }

   }

 }

}

void flash() {

 digitalWrite(latchpin, LOW);

 shiftOut(datapin, clkpin, MSBFIRST, passArray[index]);

 digitalWrite(latchpin, HIGH);

 delay(300);

 digitalWrite(latchpin, LOW);

 shiftOut(datapin, clkpin, MSBFIRST, 0);

 digitalWrite(latchpin, HIGH);

}

ISR(PCINT0\_vect) {

 trigger = true;

 cli();

}

## Media

|  |  |
| --- | --- |
|  | |
| Schematic | |
|  |  |
| Top View | Side View |
|  | YouTube Video:  <https://www.youtube.com/watch?v=gHZkC36B_3k> |
| Flex PCB |  |

## Reflection

The flex aspect of this project provided some unique project posibilites that I couldn’t have done before. As well the fact that It could be in my DER was a big drive to make it something uniquely suited to that. Unfortunatly the application of the project was not perfect. I found the surfacemount soldering dramtically more challengeing than I was expecting, similar to the last project. As well the concussions timing was particularly unideal and threw off a lot of the momentum of it. Either way I look forward to having it in my DER, even if it isn’t working properly.

# Project 24. Long ISP

## Purpose

To create an input system for the CHUMP so that it can be coded too. As well to create a PCB version of the CHUMP that can work with the input system.

## Reference

RSGC ACES Website

<http://darcy.rsgc.on.ca>

Technical Writing PDF

<http://darcy.rsgc.on.ca/ACES/technical-writing.pdf>

Project Link

<http://darcy.rsgc.on.ca/ACES/TEI4M/1819/ISPs.html>

## Theory

How Does the CHUMP Work?

See: Project 16. CHUMP – Completed Processor – Procedure

General CHUMP ICs

See: Project 16. CHUMP – Completed Processor –Theory

EEPROM(AT28C17)

See: Project 16. CHUMP - Processor - Theory

## Procedure

### Programmer

The input systems workings are rather simple. First off there are two 8 length switch banks. One

|  |
| --- |
| CHUMP Parts List |
| 74LS189 |
| 74LS157 |
| 74LS174 |
| 74LS377 |
| 2x 74LSC17 |
| 74LS161 |
| SN74LS93N |
| 2\* SN74HC595 |
| CD4069 |
| CD4011BE |
| SN74LS00N |
| Clock Circuit(See CHUMP – Clock – Procedure – Parts List) |
| 7x 2.2k Ω Resistors |
| LEDs |
| 10/2 Female Header |

of these corresponds to the address to write too, while the other corresponds to the data to write at that address. The general form is 5V → switch → bused resistor network → GND, with the signal between the switch and the resistor being attached to the address pins and I/O pins of the EEPROM. Once these bytes have been set that means the correct signals are present on the pins, and the data can be written. Do to this a simple button press will toggle the WE pin, loading in the data.

\*An important note is that the CE pin could also be toggled and it would perform the same function, as long as:

1. One of them(CE or WE) is constantly LOW.
2. The other is pulsed(CE or WE)
3. The OE is HIGH

Then, it will perform a write.

On the CHUMP board, at default the EEPROMs are automatically set to output meaning OE is

|  |
| --- |
| Programmer Parts List |
| 2x Switch Banks(8) |
| Push Button |
| 1k Ω Resistor |
| 4.7k Ω Resistors |
| 10k Ω Resistor Bused Network |
| Diode |
| 104 Capacitor |
| 10/2 Male Header |

LOW and WE is HIGH. However, when the programming board is inserted into the CHUMP it automatically pulls the WE pin LOW and the OE HIGH. So simply by connecting the programmer, the board will switch to input, so all that has to be done is to flip the switches to the desired address and data, and then press the WRITE button which will pulse the WE pin, storing the data on to the EEPROM. As well, the Program Output LEDs should be showing the current line being written in, so they can be used as a reference.

This system unfortunately in the first iteration had one problem. It stems from the fact that when the programmer is inserted it means there are two sets of pins connected to the address byte of the EEPROM, the first from the program counter, and the second from the programmer. So if one of these sends 0 and the other sends 1, it means that a LOW signal is directly being connected to a HIGH signal, i.e. 5V to GND, which causes a short circuit.

This can be fixed rather simply, all it requires is a multiplexer that takes in data from both bytes and then has a control pin to determine which should be sent to the EEPROM. The control pin could at default be set LOW, allowing the data from the PC to go to the EEPROM and then as soon as the programmer is inserted, similarly to how it automatically puts the OE pin HIGH it could also put that control pin HIGH. This relativity simple fix has been updated for the second board.

### Case

|  |  |
| --- | --- |
|  |  |
|  |  |

The best way to mount a PCB to a case is through the application of mounting screws. Add several mounting holes on to the PCB itself, and then use heat set inserts in the case to provide a base to screw the board in too and it will provide a long-lasting and secure bond.

Unfortunately, in the first draft of the CHUMP holes were overlooked. So another method had to be determined to mount the board to the case. This turned into a combination of hot glue and compression. As shown to the right there is a base to support the corner of the PCB in the case. Then on top, there is a cutout for another separate piece of the case. Once the PCB has been placed into the case on top of the pillars, the small square piece can be hot glued down on the corner in the space. This should secure the PCB in place.

However, since mounting screws are ideal they will be used for the second version of the PCB and its new case.

### Adding a command to the CHUMP

There are 14 overall base commands for the CHUMP(shown below). However, since this is a 4-bit computer, there are 16 actual possible commands, meaning the 15th and 16th are up to be defined by the user. There is a slight problem with this though. As the block diagram shows, there is a command line that leads to a AND gate and then into the program counter which when triggered loads a value into the PC. This control line is unique though because it isn’t from the control EEPROM outputs, instead, it is sourced by ORing bits 2,3 of the OpCode, and then ANDing that with bit 4. This is because this line is only needed to be HIGH for GOTO const/IT and IFZERO const/IT. The issue arises when trying to add a new command. This is because this new command would have the OpCode 1110 or 1111, which would also trigger this command line. So in the normal CHUMP, this worked fine because no final command was added. However, in order to increase versatility with this project, it should be configured to allow a final command. To do this, for the final version of the CHUMP, the GOTO and IFZERO commands will all be shifted down, to take up 1100, 1101, 1110, 1111. This means that now to derive the command line bit 4 and 3 can simply undergo an AND. It also leaves the space at 1010 and 1011 to be filled with any new instruction.

|  |
| --- |
| Notes: 1\* and X mean the signal is irrelevant |
| CHUMP Version 1 Instruction Set |

To see a detailed explanation of the CHUMPs PC extender and output system, see Daniel Raymonds DER. His video will be linked in the Media Section as well.

Overall, a large portion of this project was in the formulation and then problem-solving of the CHUMP and its peripherals. And while this takes time and effort, it is not necessarily something that can be discussed in this section.

## Media

|  |  |
| --- | --- |
|  | |
| Full CHUMP with Programmer | |
|  |  |
| Programmer Top View | Programmer Side View |
|  |  |
| Case Open Front Version 1 | Case Closed Back Version 1 |
|  |  |
| Case Version 2 |  |

Programmer Video:

<https://www.youtube.com/watch?v=C6ym7kKK5XI&t=5s>

Daniel’s Video:

<https://www.youtube.com/watch?v=vZ13xud0qBc>

## Reflection

This project was a very good final project for this year and for my overall time at RSGC. The CHUMP as a project was one of several throughout my ACES carrier that I committed myself to completely. As much as it was a school project it was a project of passion. After finishing the breadboarded version Daniel and I both wanted to continue it on for either the medium or long ISP and throughout the year we were brainstorming and talking about what form that would take. These discussions went back and forth but whatever it was, it was something I wanted to do. Unfortunately, the concussion in a pretty severe sense threw me out of the mindset though. Though I was still able to work on it, my capacity for the same fervour that I had had with it in its breadboarding and with some other projects wasn’t there for a good portion of its development. Because of this Daniel deserves credit for keeping the momentum of it going when I wasn’t completely there. In the last week primarily though, my capacity has been increasing and I have been able to get back in and try to finish this off right. Because seeing projects like this work isn’t just seeing it work, it is seeing all the hours of frustration, failures, and breakthroughs manifested into a single object of pure accomplishment.

These past two and half years have been a tremendous time. Looking back this course stands out as a defining formative experience of my life. I have never worked as hard or been as devoted to anything as I have my ISPs and work here. It taught me what I could do, and the power I had if I wholeheartedly committed myself to something. It also brought me closer to many of my peers and made some good friendships I hope to continue on throughout university. Overall I would simply like to thank you. It’s not an easy course to run and sometimes I’m sure it can seem like we are wasting it away, but it’s important to remember the impact it has on many of its students and the lessons it teaches. Without this, I may never have found my passion for software and engineering and that is something I am incredibly thankful for. I only wish that everyone could have an opportunity like this one. Despite this, it still hasn’t quite hit me that this is the end. It’s a really hard idea to truly understand. And though I may have some anxiety about the unknown years to come, I look forward to them with extreme excitement. Thank you for this course and your impact. Have a good summer, and I hope to see you next fall with news from Queens!