6502 Project

Isaac Preyser

Over the past month I've been working on building a working breadboard computer, based on the W65C02 IC. In this report, I seek to inform the involved parties of what I've learned, the challenges I've had to overcome, and the conclusions I've drawn from this project.

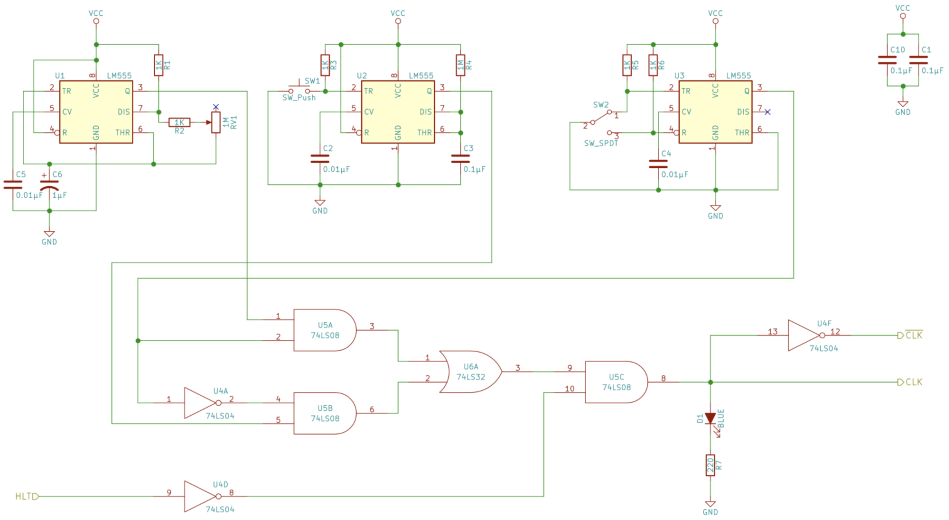
Ultimately, the project has proven too difficult to complete at the moment given the problems that have arisen, and the time left in the semester. This project is still possible, but the scope of understanding needed to fully debug and repair some of the parts is outside of my realm of knowledge at the moment. I'm currently working on expanding that knowledge at the moment.

Goals of the Project

The goal of this project was to build a functioning computer out of integrated circuits, capable of driving a simple LCD display. In order to do this, I needed to find a way to allow for a CPU to be able to both read and write to memory, interact with I/O devices, and be able to detect when a program is finished to be able to halt the system clock. By following along with video lessons published by Ben Eater, I was able to learn as I built, and also look into other areas that he had suggested in his videos.

Pre-Build

Being able to satisfy all of these goals turned out to be a monumental task. Building the clock even proved to be a challenge. In order to begin building the circuits, I first needed to understand how to properly interpret circuit diagrams, such as this one:



In addition to this skill, I needed some practical know-how as well, to build the circuit. Some of these skills include identifying the resistance of a resistor based on it's color bands, being able to effectively use tools such as wire strippers and wire cutters. During this phase I also learned how to identify what side of the IC was the top. (as shown by the divot or dot at the top near PIN1.)

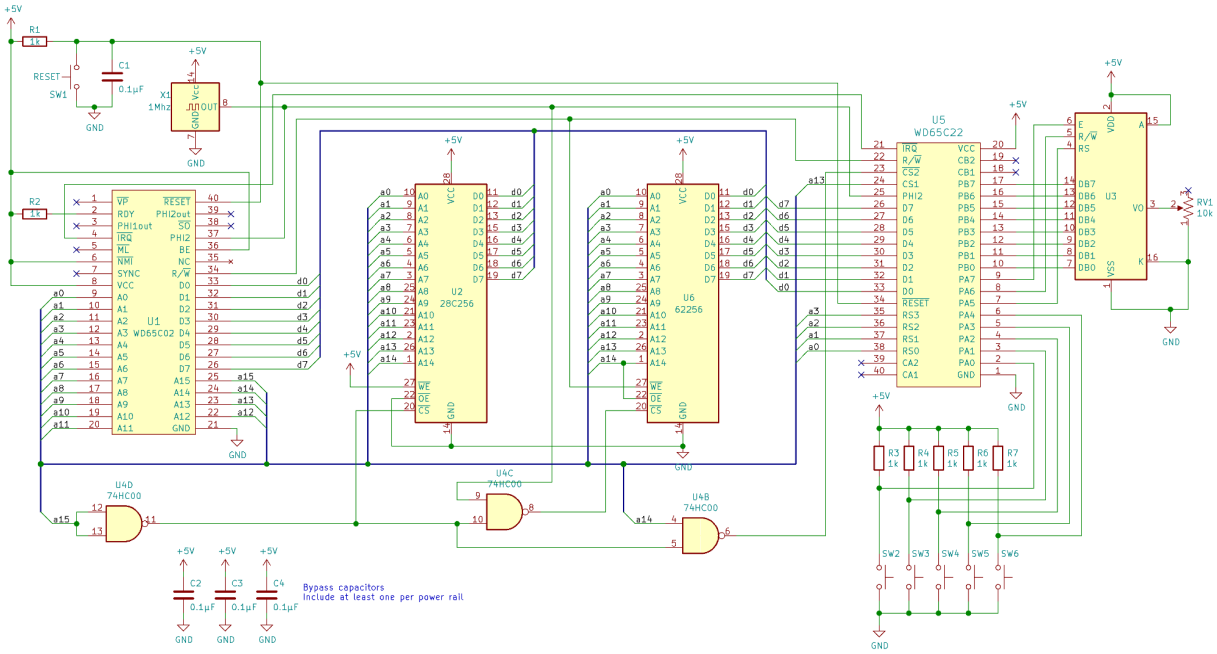
Clock Module

After attaining the knowledge needed to begin building, I started to work on the clock module. I first began by utilizing the LM555 IC to create an astable circuit that would pulse continuously. This works by having the output of the circuit directly affect the input, thereby causing an eternal imbalance with no state where the circuit will be at rest. This would form the basis of the system clock. The timings are controlled by a variable resistor, which increases the amount of resistance needed for the discharge capacitor to fill and subsequently discharge, and reset the trigger pin. (This constitutes a clock cycle.) Increasing the resistance of the resistor would limit the amount of voltage reaching the capacitor, thereby increasing the period. I also placed an LED on the line that the capacitor discharged on, to visualize the clock cycle easily. To finish, I took the output of this, and put it into an AND gate, controlled by a lever. This allows me to select between this astable mode, and the bistable mode I would set up next.

I next began setting up a monostable mode, which would allow me to single step instructions on the processor, which became extremely valuable once the processor began running into issues later on. A monostable circuit will have one stable state, and in the case of attaching it to the button I'd debounced earlier, would push a single clock cycle to the CPU. This was achieved by using another LM555 IC, the same as the astable clock before. However this time, the trigger pin is no longer controlled by the output of itself, but is now controlled by the push button. Like the previous part, I then fed the output of this into an AND gate, to allow for the toggling between two modes.

Following this, I set up a bistable mode, which allowed me to select between the two modes fed into the AND gates, and have a single clock output. This single clock output is then wired to the PHI2 pin on the 6502.

6502 Module



Following the build of the Clock Module, I began wiring up the W65C02 IC to the breadboard's power rails and the output from the clock module to PHI2. At this point I wired up some LEDs to the data bus to confirm that the CPU was indeed doing *something* and was properly stepping in correct time with my system clock I'd made. At this point everything seemed to be running normally. I continued on from here and began to hook up the AT28C256 EEPROM, which had been intended to store the program data. At this point I needed to learn how to interface with the EEPROM and program it safely and efficiently, which proved to be more difficult than intended. One of the main difficulties I found was the included software was very poorly translated to english, and with it had poor documentation. Everything done at this point was by trial and error, and it took almost an hour to get the software to acknowledge that I had the EEPROM plugged in to the flash programmer.

At this point I needed to learn more about how to create a valid binary that the 6502 could interpret and run. This proved to be fairly simple after watching Ben Eater's videos on the topic, and how assembly compares to machine code. Coming right off of learning C++ and also React at the time, picking up the new "language" felt fairly simple. However, programming in assembly proved to be much different to other languages, because assembly lacks many simple features used in programming that are standard in other languages. Most of this is actually found in the syntax- as the conventions used when setting up a loop or an if statement is wildly different, as assembly forgoes the curly braces and semicolons I'm accustomed to in the C family of languages. Using the mnemonics to program instead of looking up the op codes to program is really nice, as remembering that `LDA` is "Load A" and `STA` is "Store A", instead of having to remember `0xa9` is LDA (in the correct operating mode) and `0x8D` is STA.

An important note here- since the EEPROM requires a chip enable signal to allow it to be read from, A15 goes through a NAND gate to enable EEPROM to write, so the first location where memory can be read appears to the CPU as `$8000`.

For this, I used "vasm" as my assembler of choice. It was the one used in the videos, and worked directly with VS-Code in the terminal. I also got syntax highlighting to work in my IDE! Making the rom file for the 6502 was fairly simple, as Ben Eater walked through the creation of these files step by step, and provided code available for download. He used vasm to write his assembly first, which then outputted an assembly program that could be fed into a python program he'd set up to first write the program to a 32 kilobyte rom file, which then filled with "No Op" commands. This then outputs a binary, which is used to flash the EEPROM. After learning how to do this, he then taught the "JuMP or jmp" mnemonic, used to make the program loop, which made this python program not necessary to write a valid binary.

At this point, I had written a program that would tell the computer to start loading operations at a specific address. I excitedly flashed the program, and got to work setting up the serial monitor to confirm the ICs were behaving as intended.

Serial Monitor

In order to verify that everything was working as intended, I needed to know exactly what the CPU was doing. Ben originally used a dedicated serial monitor/oscilloscope, but later instructed how to create one out of an Arduino Mega. Since we only need to monitor if a pin is low or high, the digital pins on the Mega are perfect for this use case. Here, the address and data pins are connected to the Mega's ample digital pins, and are monitored and printed as binary on the serial monitor's output. In addition to this, the read/write bus is monitored, and outputted to see if the CPU is reading or writing during the clock cycle. Finally, the PHI2 pin is monitored, so the Mega only takes a snapshot of the current clock cycle when a new one actually happens. This helps to limit the strain on the Mega, as well as helping to reduce the amount of output to sort through in the serial monitor's output.

I hooked everything up, and quickly was alarmed by what I was seeing. Firstly, the arduino was capturing multiple snapshots in one clock cycle, when it was only supposed to capture one. (This was proven by the actual clock only putting out one cycle, as I used my monostable mode to do this. I further proved this by placing an LED on the PHI2 pin, and it only pulsed once.) Furthermore, the snapshots returned had impossible operation codes according to the operation matrix in the data sheets. Therefore, I knew something strange must have been occurring.

At this point I began troubleshooting.

Troubleshooting

I first tried to validate that all the connections were good by using a multimeter and it's continuity checking functions. All of my address and data lines seemed to be intact, and transmitting data to the

right places. I was at loss as to why I was getting such strange readings from my serial monitor. I'd triple checked the code, being sure to carefully place the jumper cables in the proper pins on both the Arduino and the breadboard.

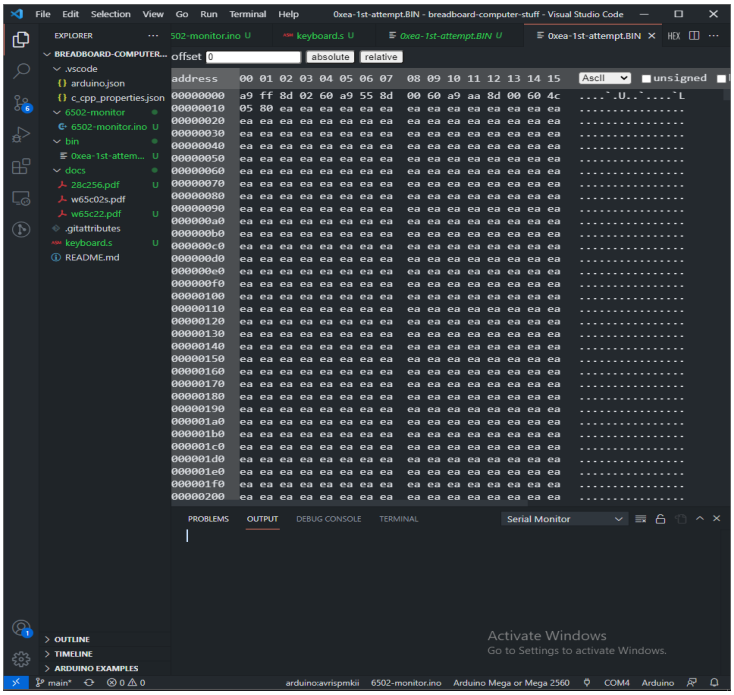
At this point the clock pulse signals appeared to be dying out whenever I let the board sit for too long without disturbing it. This proved to be a fascinating phenomena and I showed it to my dad. This is when we deduced it to likely be electromagnetic interference. In the context of such small and precise electronics, this is very bad.

[This video showcases the phenomena, as both my dad and I go over it.]

We found that whenever someone bumped the halt cable, the light would come back on. We weren't too sure why this was happening, but after straightening the cable, the problem seemed to be remedied. However, the serial monitor was still outputting garbage data, and pulsing more often than on each clock cycle. Something strange was still going on. Somehow current was getting into the HALT line, and stopping the CLOCK line from transmitting a clean signal.

At this point I wanted to make sure that the EEPROM's program data wasn't compromised. So, I took out the chip programmer, and monitored the EEPROM's data through my computer. Here, I had to stumble around the software, trying to figure out how to read what was on the chip. I got the chip to dump it's contents into a binary, so I began to decode what was on it with a hex editor:

(Yes, I know I hadn't pushed the changes in my repository from me adding all the files at the time of the screenshot, this is remedied now)



To my surprise, the contents of the chip were unchanged. Although the serial monitor had been outputting that the breadboard was rewriting the chip, nothing had happened. At this point I was beginning to not trust the serial monitor, however I needed it to work in order to continue on with the project.

I also suspected the power supply might be causing these issues, as insufficient power would definitely cause issues with the 6502. I began studying the data sheets of both the EEPROM and the 6502. I learned that if the voltage supplied through the PHI2 clock input was insufficient, weird things could occur. Thus, I used my voltmeter, and analyzed the voltage associated with the clock pulse coming through. All appeared to be reading normally, and I eliminated this variable.

At this point I began to review the supplied serial monitor code to make sure that it all was working correctly. The code itself seemed to work, and I didn't see any major flaws that would cause the serial

monitor to give such inconsistent outputs. I believe this to be EMI as well, due to the disarray of the jumper cables to the digital pins on the Arduino.

I also learned that labeling cables, and having clean and efficient runs when dealing with a breadboard-based project is imperative to success. Compared to using the pre-cut wires with the clock kit, the custom cut runs I made on the 6502 module were much easier to follow, and thus less of a pain to debug.

Additionally, as I'm writing this, my LED monitoring the output of the CLOCK line has died. I'm not sure why this happened, as I had it connected to ground through a current limiting resistor, but it shows no sign of life anymore. Strangely replacing the LED has made no change, unless I replaced it with yet another dead LED.

Due to the numerous troubleshooting issues I have right now, I've been forced to place this project on hold. I have a couple ideas left that I could try, but right now they are far too time consuming to try before writing this report, which needs to be done by the end of the semester. Next semester I have *Computer Explorations 11* with Mr. Lustch, where I suspect I'll have more time on my hands from time to time to take a look at this at school.

Conclusions and Recommendations

After struggling with this computer for nearly a month now, I would not recommend using these kits to teach electrical engineering in a high school environment. Even with experience already in this area, I faced many difficulties, as parts had been misbehaving. One does need a fair amount of technical know-how, and knowledge in electronics, physics, and computer science before tackling a project of this size. Additionally, when things go wrong, tools such as soldering irons may be needed to repair pins, as I needed to on the EEPROM IC. (The pins were extremely fragile coming out of the flash programmer.)

Doing this sort of thing on one's own is difficult, and I'd recommend learning this with an actual instructor providing guidance. This way, more interaction is achieved and greater amounts of information can be custom-fit to be synthesized by students. For example, if I was to run into a specific problem, I could ask for help directly, instead of receiving assorted answers online that may not exactly pertain to my use case, situation or the tools at my disposal.

Furthermore there is little documentation with the kit itself, as it's expected you learn everything from the videos. This makes it very difficult to reference something as one is building, as it requires constant stopping and starting of the video to follow along intently. There is some support online in Ben Eater's online communities, however without users online being able to remotely diagnose issues with physical hardware the remedies usually offered to problems is to start again from scratch.

Also, I needed to possess knowledge in a variety of concepts and skills before beginning this project. Below I will dive into some of these skills, and elaborate a small amount on them.

Things I Learned:

Software

- Advanced VS-Code operation

  Configuring the gcc compiler to work nicely with the Arduino extension in VS-Code was quite a pain. VS-Code relies on it's settings.json file heavily, which I needed to learn what entries I needed to edit to properly set up the software to look for the right libraries and the AVR compiler.

  I'd say from this experience VS-Code is a better IDE for modern languages, particularly ones that target web platforms. While the IDE may be flexible enough to handle all kinds of files, for my use case I needed to install many extensions to make my workflow at least somewhat efficient. I needed a hex editor to view my binaries that I was creating, as well as automatic formatting for my assembly files. Both of these things needed dodgy extensions in order to work in the software.

  I also learned how to use VS-Code in a more efficient way, making great use of keyboard shortcuts and the F1 quick menu, and I was able to push changes to repositories, open my serial monitor, and switch to hex view mode on the fly quickly. I also appreciate the tiling system that the IDE uses, as I'm able to have multiple views open, along with my docs open in one application, without having to fiddle with Windows.
- Xgpro flash programmer software

  The Xgpro software was the software that came bundled with the flash programmer, and functioned moderately well at best. There was a steep learning curve to understand what exactly was going on in the software, and bundled with poor translations to English in it's technical docs/user manuals, a lot of trial and error was needed to figure out exactly what some buttons did. Essentially the software was dodgy at best.

  In the future, I'd recommend using Minipro to interface with the programmer, but passing a COM port through Windows to WSL2 was not working for me, and I didn't want to play with core system files on my machine to make it work. Honestly for this type of project, I'd recommend any UNIX based operating system over Windows simply for ease of use. Many of the software tools I had been using for this project were originally written for Linux, and ported to Windows later with very little in the way of testing and debugging for the platform.

  I'd highly recommend having a few dual-boot capable or solely Linux-based machines in the computer lab for teaching, and for use cases like for this project, where the better tools are simply not available on Windows.
- Bash shell

  For this project I did end up moving it to my Linux machine, which uses Ubuntu Core. This means I got very comfortable with the Bash shell and CLI based applications. This also means I needed to be able to use common applications, such as the apt package manager comfortably to be able to set up the Minipro software. I already knew how to do

this, but someone new would definitely need a quick crash course before jumping into this part of the projects

- PuTTY

  For this project I needed a way to interface between my Windows based and Linux based computers. For this I used PuTTY, as I did not need any graphical output. This software is an SSH client that allows for remote access into the shell of a given computer. I mainly used this alongside a local network share I'd made earlier to use the flash programmer effectively while I was learning how to use Xgpro.

  Here I also played with Bash Pipelines for a little bit to see if I could automate my process by using the output of vasm to flash the EEPROM. This was really cool to try out, and I could see it being used for larger workflows well.

Hardware

- Soldering Skills

  For this project I needed to re-solder some pins on the EEPROM, as they were quite fragile coming out of the flash programmer. Resoldering the pins was a trivial task with the help of a third hand tool and a steady grip.

- Precise Wire Cutting and Stripping

  The second half of this project required me to custom cut wire runs and strip wires frequently for use in the Breadboard. My skill in this has noticeably improved over time, judging by the contrast in quality of the wire runs from the beginning of the project to the end.

- Understanding and use of A Multimeter

  I had to use a multimeter in various modes for this project, to rule out certain things that may have been misbehaving. I used this tool to verify that my power supply was working within its specification, and to check that wire runs were in fact continuous.

- Safe Hardware Handling Techniques

  Like with any electrical equipment, I made sure to have clean hands when working. This should be common sense to almost anyone.

General Skills

- Reading complex circuit diagrams

  For this project I needed to be able to understand, read and build complex circuit diagrams. This was the only document supplied by the kit manufacturer that actually helped me build the kit.

- How to read and interpret data sheets

  Referencing the data sheets provided by the IC manufacturers is a huge asset when trying to build a breadboard computer. They are filled to the brim with useful information, such as descriptions of operating modes and different functions, pinout diagrams, and

numerous tables that make it easy to find a certain operation by code. These documents helped immensely during both the debugging and the building of the computer.

- Reading the resistance on a given resistor

  This is something I picked up on quickly while building the computer. It's quite simple, as you read from left to right, with the metallic band always on the right. I personally always referenced a chart, just to be sure.
- Building Readable, Organized Circuits.

  I learned this when debugging my circuit. Having clean wire runs that are organized by color is extremely important, as it helps when someone returns to maintain the prototype, or debug. Using sticky notes to label things is also helpful. I think the best way to analogize this would be to compare this practice to well-commented code.

Concepts

- Learned different ways of storing numbers (really data if you think about it)
  - This includes Hex, Binary, Base 10 etc
- Learned different common electrical circuits used in many devices today
  - SR Latch- Used in flash memory
  - Various logic gates
  - Debouncing circuits
  - How to clean up electrical signals
  - What Pull Up, Pull Down, Current Limiting Resistors are
- How the CPU addresses the memory, how the CPU reads from the EEPROM

Current State of the Project

Currently, the project is on hold until I, or someone else, gets a chance to fully restart it. There is significant electrical interference within the wires, due to the nature of the pins being so close to each other, and thus better shielded wires may be needed. This is especially true in the longer connections between modules, particularly the clock and halt lines.

Alternatively, I could hope for the best and continue the project oblivious to the issues cropping up here, and blaming the bad data coming out to EMI in the wire runs leading to the Arduino. As I pointed out earlier, the contents of the EEPROM were preserved although A15 appeared to be enabled, signaling a write operation. This inconsistency could be the serial monitor itself and not the actual components on the breadboard. Hooking up the LCD and inputting a small program to check if that works correctly could be a way to check if something is messing up on the computer's side, or the Arduino's side.

I consider the state the project is in to be half-successful. Although I haven't produced a product, I believe the skills I learned along the way would certainly help to produce a functioning prototype for the next time I attempt this. Additionally, I created something that can be repaired and expanded upon. Given enough time to deduce what exactly is going on, the progress made on this project would not go to waste.

Considering my goals at the start of the course, I believe I have answered the fundamental questions I wanted to explore. One such question is this:

I would like to better understand how a computer physically runs the instructions we send to it. Furthermore, how does high-level code get translated into low-level code that a computer understands?

As I've traversed further and further down in programming towards machine code, I understand that compilers are essentially translators for humans to be able to describe operations to a computer. As I got closer and closer to the raw operations that take place each clock cycle, I could compare how different languages all compile down to the same target, and their binaries they output are similar.

To-Do:

If you're reading this, you likely are interested in computers, and electronics. This project is unfinished at the moment, and you could potentially finish it.

I know this seems like a lot to learn. However, I'll lay out somewhere for you to start, so maybe you can get this thing working.

6502 Breadboard Computer To-Dos:

- Continue building the computer, as per Ben Eater's video series. I left off (at least during the build phase) on part 4, because I was hesitant that what I was making wasn't working.
    - One of the main mistakes I made during this project was having a lack of confidence. **DON'T BE LIKE ME.** Many of the mistakes I made were because I was unsure of what I was doing, and lacking any true knowledge, I stopped to worry about what was going on. Continue on with the build process, and debug when it's completed. I debugged early, and thought what I was doing was wrong.
    - This is a monumental task. There's a circuit diagram of the computer on the website. I highly recommend you follow that whilst building.
    - I'd recommend going out and buying a proper ribbon cable for the Arduino Mega, or a fancy oscilloscope to debug with. My set up (the one with many stray wires) likely gave bad data due to EMF.
    - Make sure that all your cables have sheathing, all the way down to the breadboard. Cables without shielding can induce current in other wires, breaking the components. If you have any questions on why this is, ask Mr. Lilyholm in Rm.206 for details about induced currents.
    - If you have any questions, ask them! There's communities online, and locally that can help out with this stuff.
        - Some of these groups include:
        - Makerspace Victoria (get your parent to help with this, they don't really like talking to people under 18.)
        - Myself! (My name is Isaac. You can contact me at isaacpreyser@icloud.com, or my email listed at the bottom.)
        - The robotics team (Meets wednesdays in the physics lab, however the meeting place may change once I graduate)
        - And I'm sure Mr. Lutsch and Mr. Van Slyke would be happy to point you in the right direction.

        - Personally I got in touch with some 3rd year computer engineering students at UVic, who helped immensely. As a student in Victoria, you have so much help around you.
- Begin learning how 6502 Assembly works.
- Hook up the LCD Display, and make some text appear!
- You could add some inputs to the versatile interface adapter, and make a game.
    - Fun fact: this is the same CPU that the original Nintendo NES had!
        - Add a graphics bus, and rip out some parts from an old cable box, and you'd soon have a games console!
- Explore what else you could reprogram with the EEPROM Programmer. There's a lot you can do with that little thing
    - For example, we hacked my dad's 3D printer, allowing us to add extra features to it that the manufacturer would have locked us out of if we hadn't brute forced it with the programmer.
    - This programmer is compatible with almost any IC, so look up the compatibility in the software, and make sure the code you're uploading is all valid, and you should be good to go!

Honestly, this project was a lot of fun. Although it had it's quirks, it was extremely worth it, and helped solidify certain topics that are covered in other classes.

I really hope you have fun with this project, and learn a lot, because I sure did.

Get involved in all you can, and great things will surely follow. As a result of my struggles with this project, I've landed myself a seat in university level engineering classes for my grade 12 year. I'm sure if you work hard, get uncomfortable, and learn something new, you could do the same.

If you have any questions, reach out!

Now get making!

Isaac Preyser

June 6th, 2022

preyser@uvic.ca