DAWSON COLLEGE - Electronics Engineering Technology

Semester: Fall 2023

Course: Embedded System Hardware

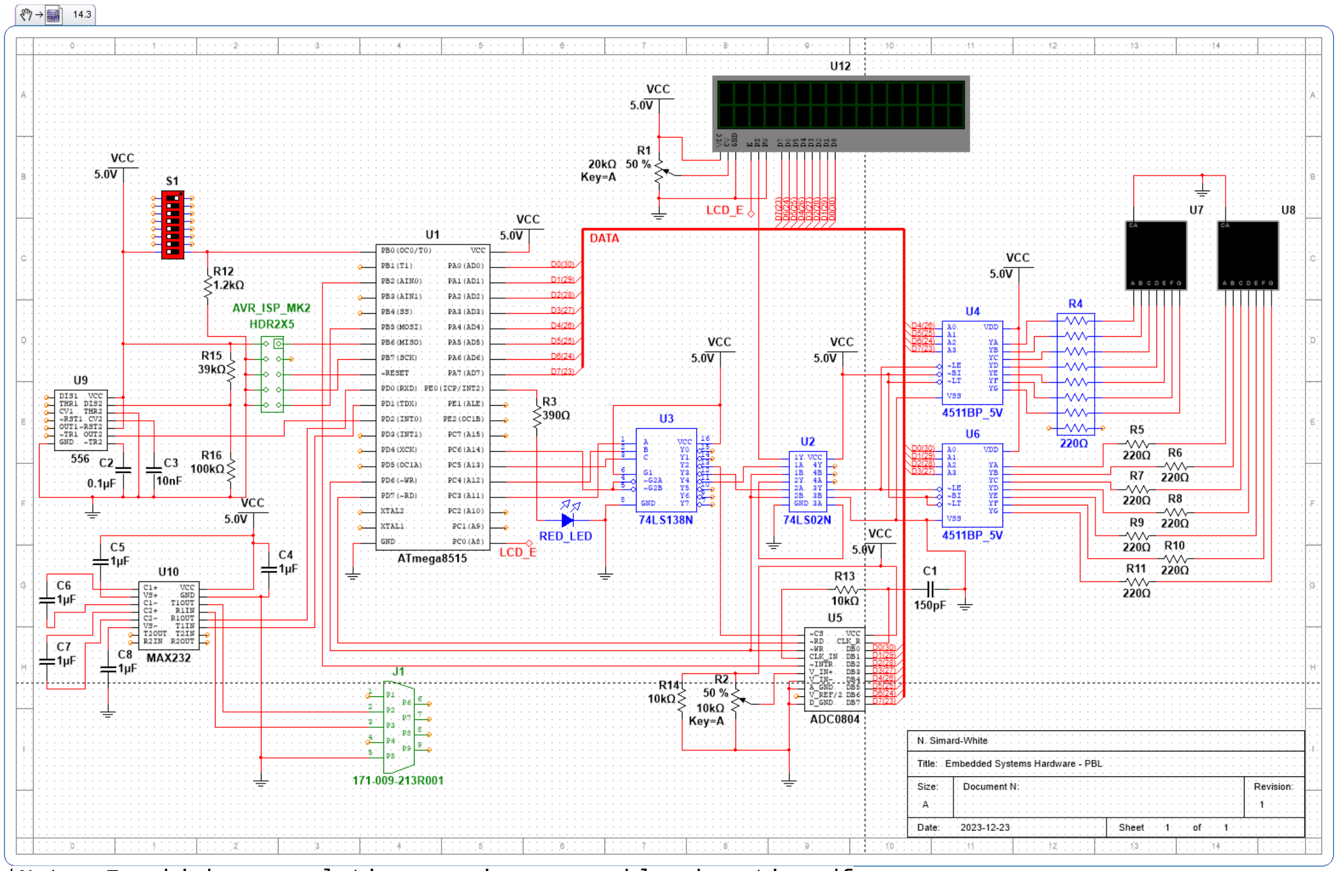
Teacher: Nick Markou

PBL

By: Nathanael Simard-White

Due by: 2023-12-24

1 – Circuit Schematic:



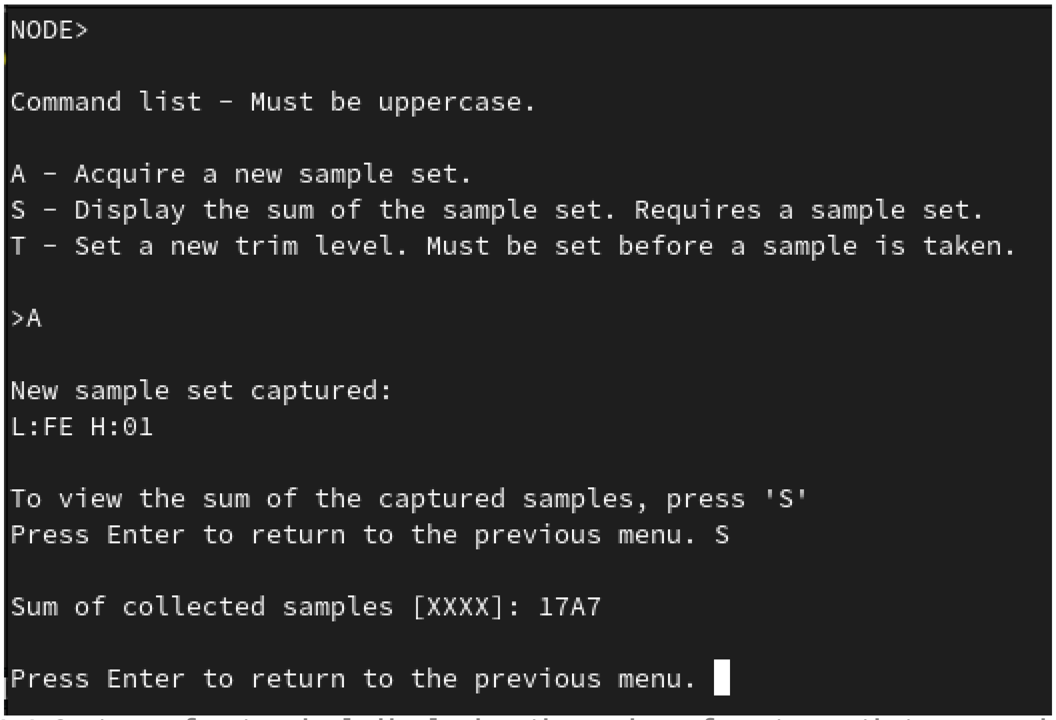
*Note: For higher resolution version, see pbl_schematic.pdf

Figure 1.1.1 Schematic of circuit made during PBL.

2 – Results:

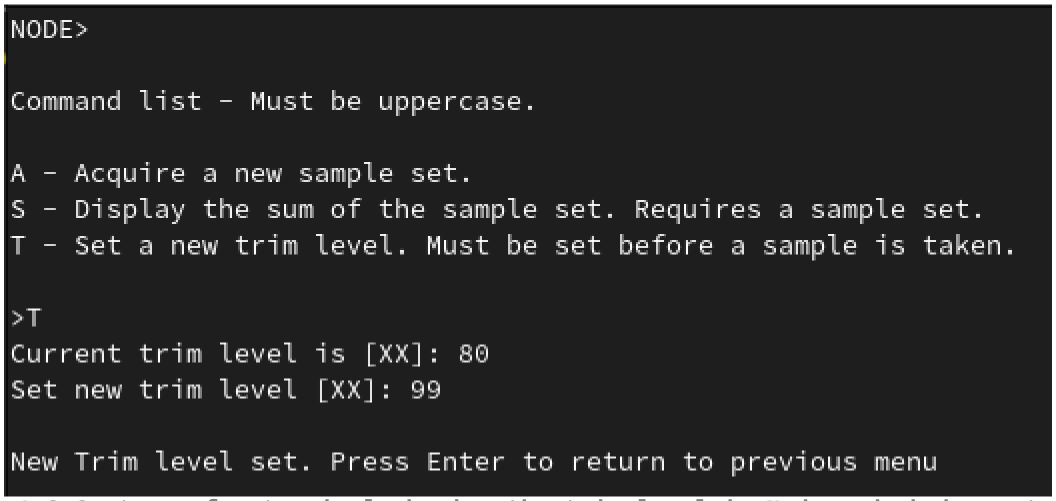
ADC acquire:

Terminal:



```
NODE>

Command list - Must be uppercase.

A - Acquire a new sample set.
S - Display the sum of the sample set. Requires a sample set.
T - Set a new trim level. Must be set before a sample is taken.

>A

New sample set captured:
L:FE H:01

To view the sum of the captured samples, press 'S'
Press Enter to return to the previous menu. S

Sum of collected samples [XXXX]: 17A7

Press Enter to return to the previous menu.
```

Figure 2.1.1 Capture of a terminal displaying the number of captures that were under or over the trim level and the sum of said captures.



```
NODE>

Command list - Must be uppercase.

A - Acquire a new sample set.
S - Display the sum of the sample set. Requires a sample set.
T - Set a new trim level. Must be set before a sample is taken.

>T
Current trim level is [XX]: 80
Set new trim level [XX]: 99

New Trim level set. Press Enter to return to previous menu
```

Figure 2.1.2 Capture of a terminal showing the trim level in Node mode being set to a new value.

LCD:

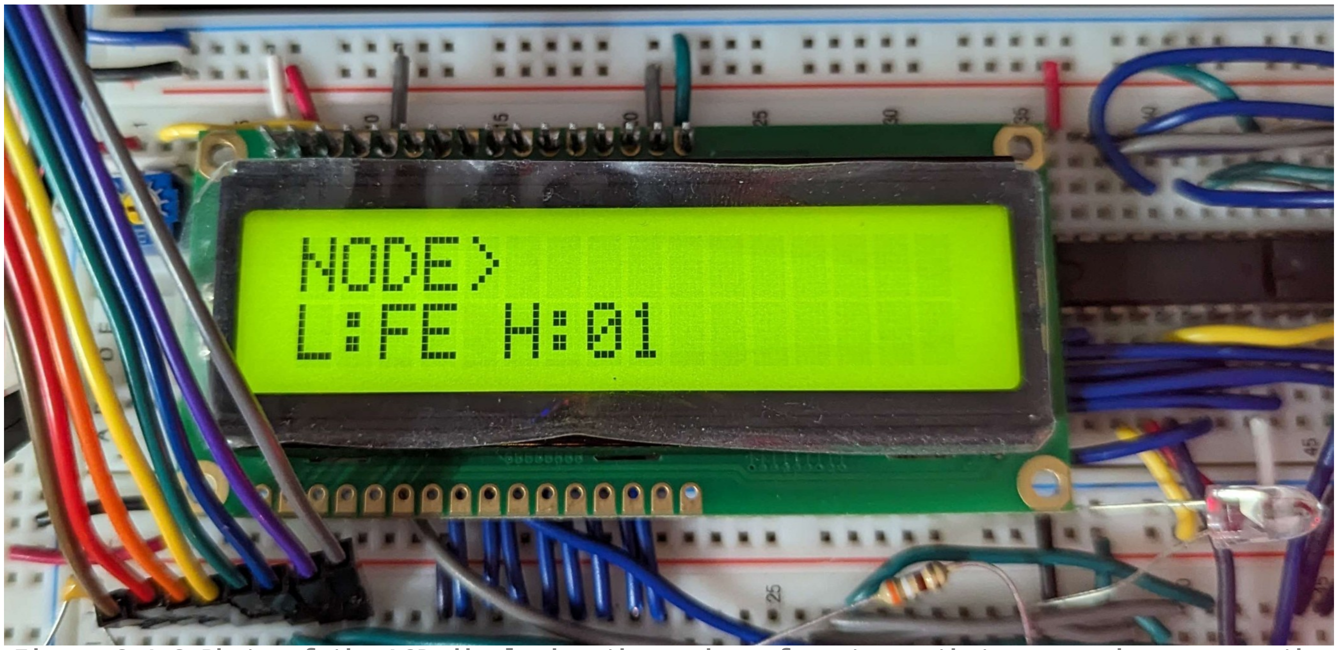


Figure 2.1.3 Photo of the LCD displaying the number of captures that were under or over the trim level while in node mode.

Bus signals:

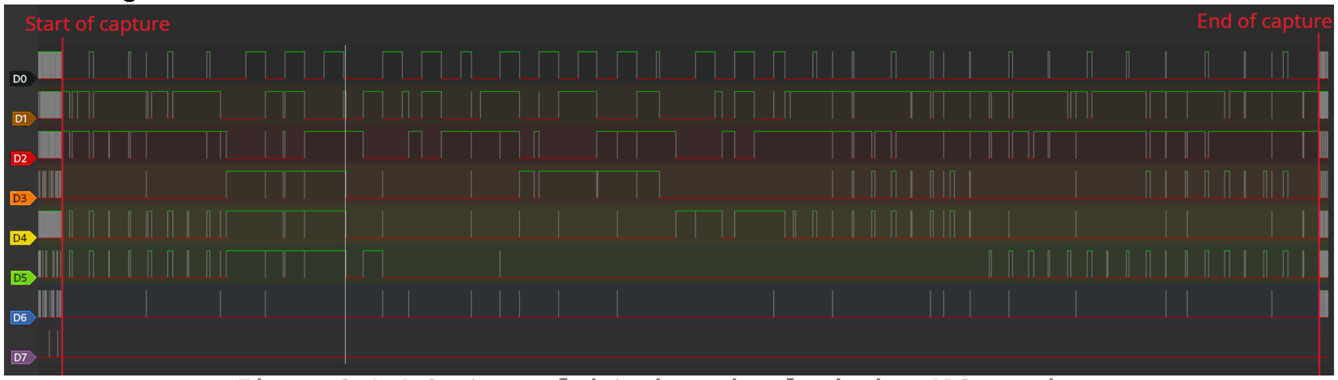


Figure 2.1.4 Capture of data bus signals during ADC acquire.

Memory decoder signals:

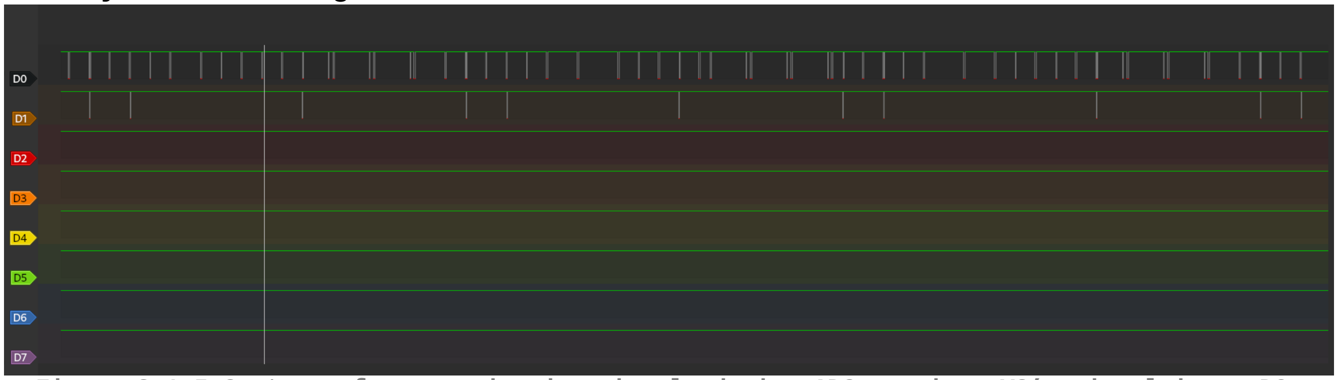
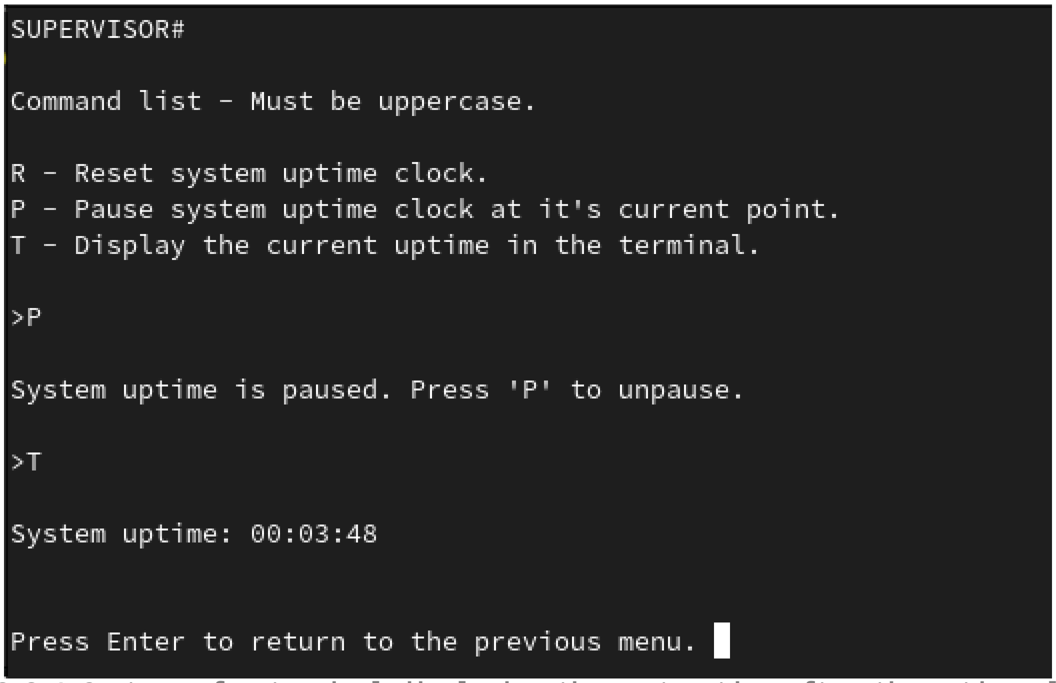


Figure 2.1.5 Capture of memory decoder signals during ADC acquire. Y2's signal is on D0.

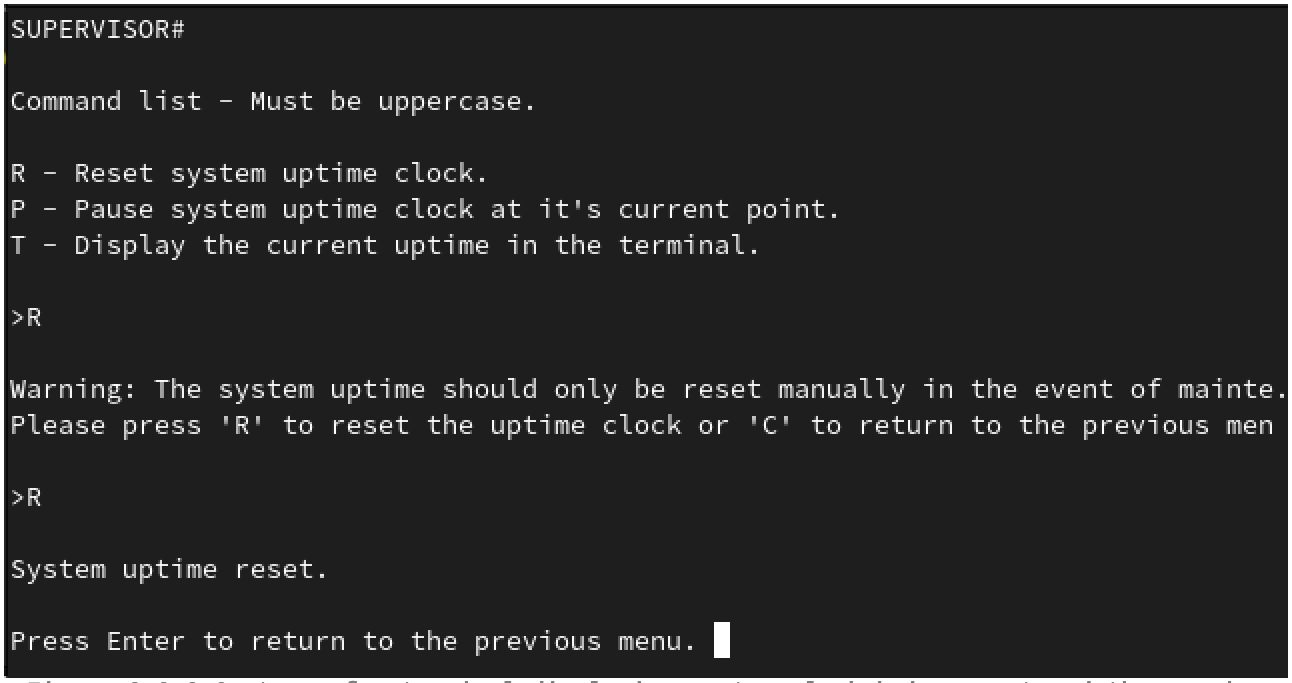
Supervisor mode:

Terminal:



```
SUPERVISOR#

Command list - Must be uppercase.

R - Reset system uptime clock.
P - Pause system uptime clock at it's current point.
T - Display the current uptime in the terminal.

>P

System uptime is paused. Press 'P' to unpause.

>T

System uptime: 00:03:48


Press Enter to return to the previous menu.
```

Figure 2.2.1 Capture of a terminal displaying the system time after the uptime clock was paused.



```
SUPERVISOR#

Command list - Must be uppercase.

R - Reset system uptime clock.
P - Pause system uptime clock at it's current point.
T - Display the current uptime in the terminal.

>R

Warning: The system uptime should only be reset manually in the event of mainte.
Please press 'R' to reset the uptime clock or 'C' to return to the previous men

>R

System uptime reset.

Press Enter to return to the previous menu.
```

Figure 2.2.2 Capture of a terminal displaying system clock being reset and the warning message asking to confirm the operation.

LCD:

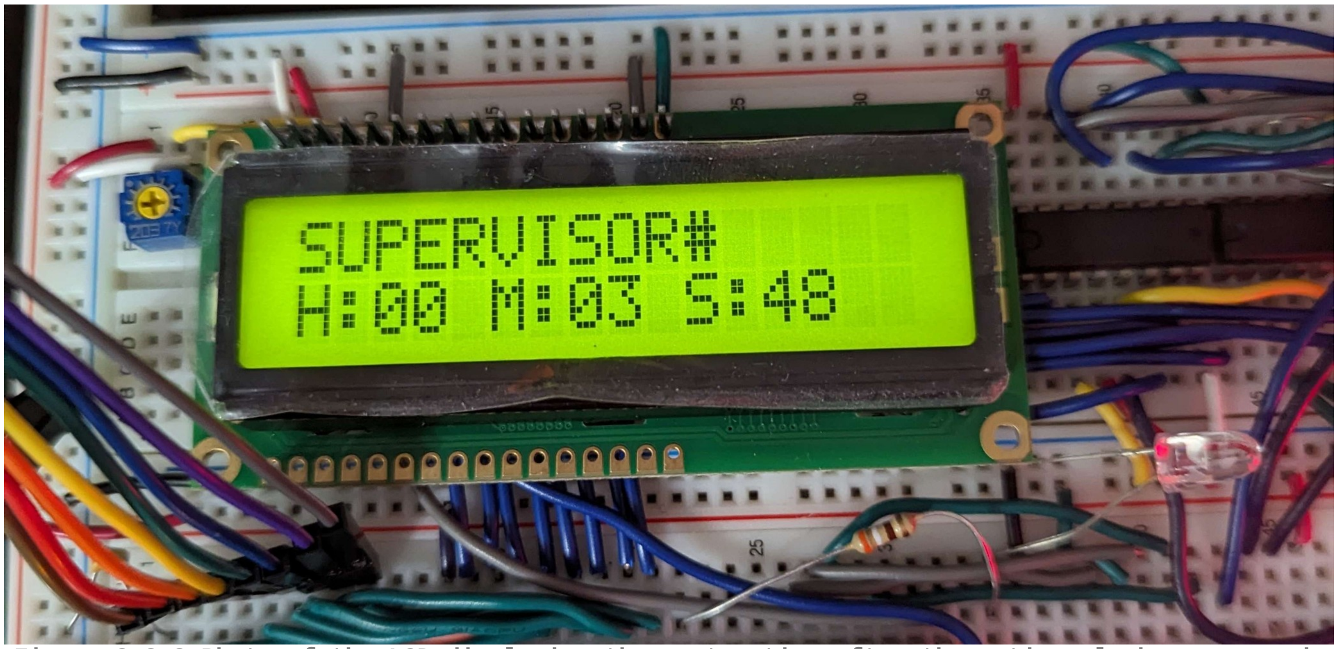


Figure 2.2.3 Photo of the LCD displaying the system time after the uptime clock was paused. Photo was taken at the same time as figure 2.2.1.

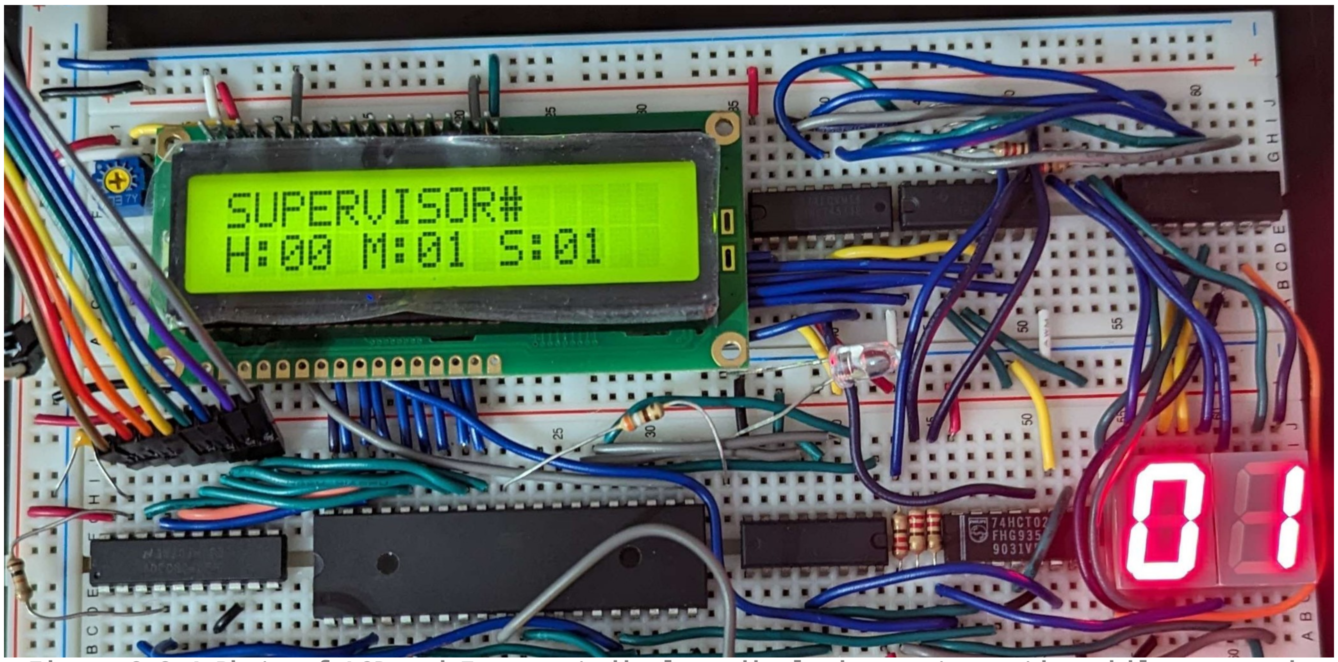


Figure 2.2.4 Photo of LCD and 7 segment displays displaying system uptime while unpaused.

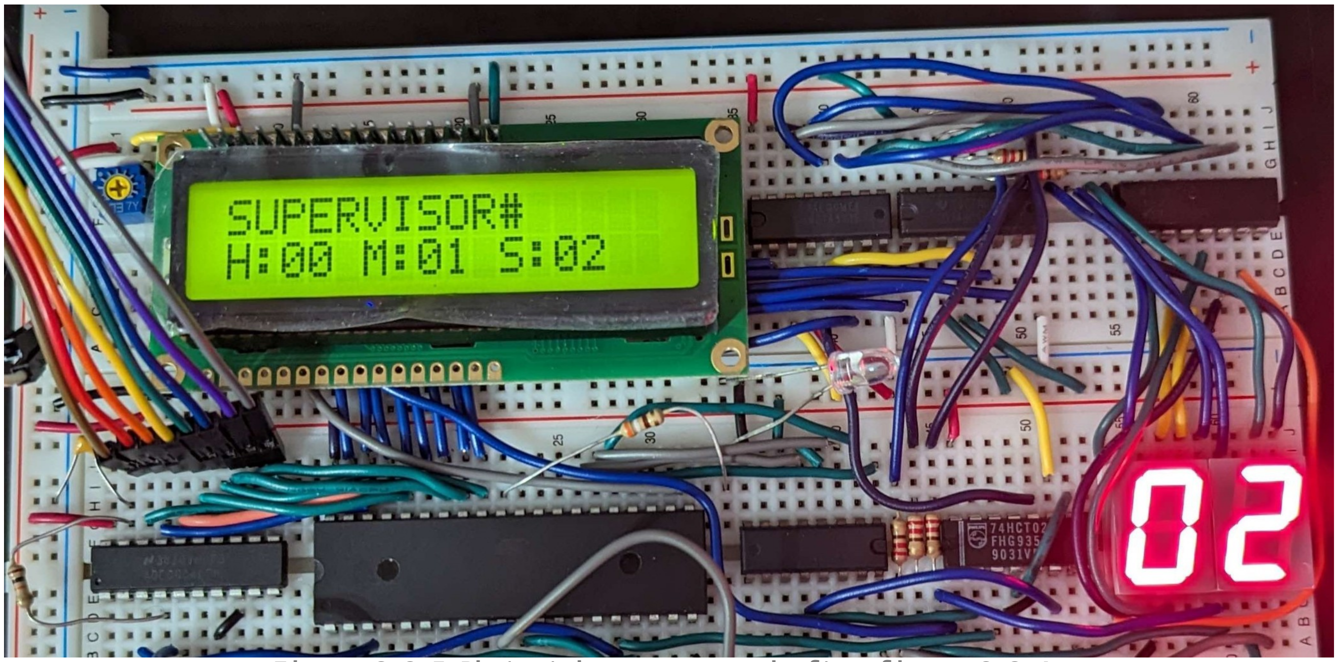
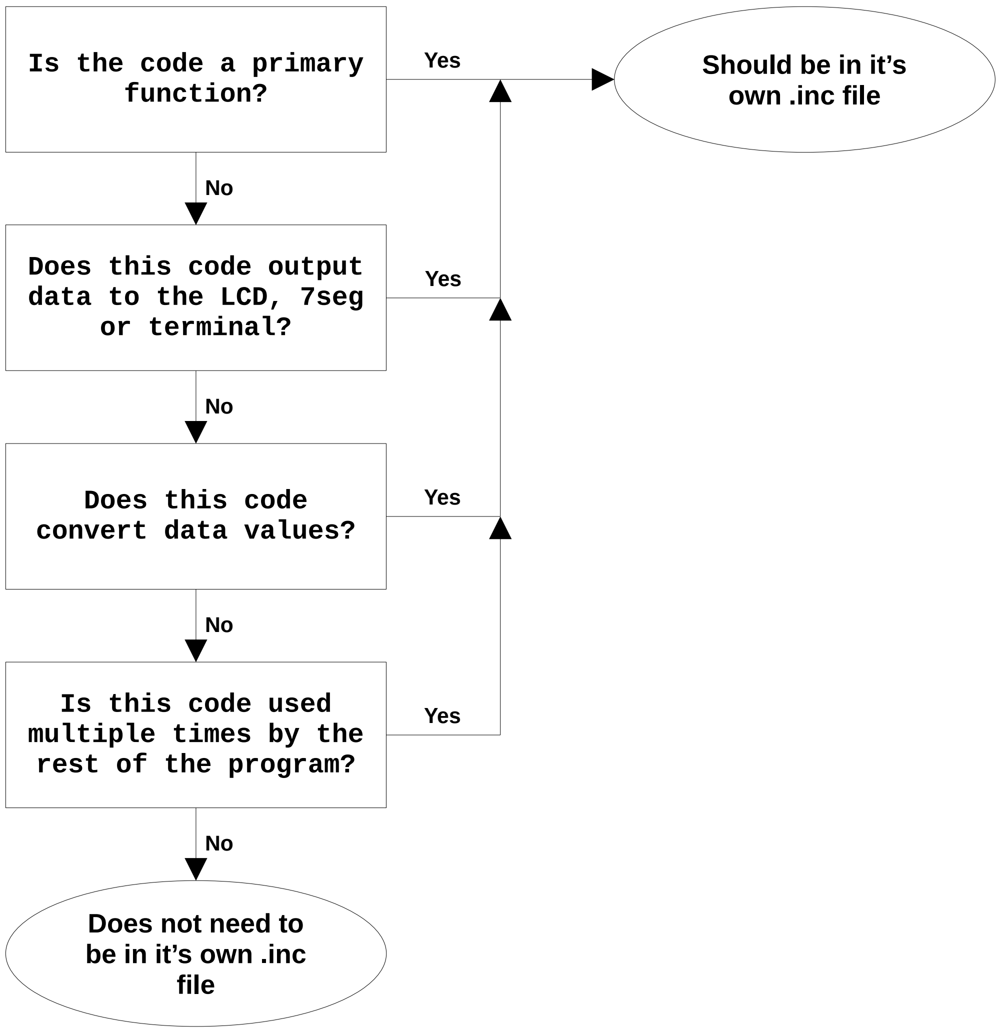


Figure 2.2.5 Photo taken one second after figure 2.2.4.

3 – Code design:

When writing my code I wanted to make each of the primary functions of the program self contained as possible. That being said, there were also certain functions that were performed by each component of the program, so it made the most sense to use RCALLs and store them in separate documents. My thought process for this was as follows:


Is the code a primary function?
Yes
Should be in it's own .inc file
No
Does this code output data to the LCD, 7seg or terminal?
Yes
No
Does this code convert data values?
Yes
No
Is this code used multiple times by the rest of the program?
Yes
No
Does not need to be in it's own .inc file

The very first functions that I put into .inc files were the delays, as they are used heavily in the rest of the program and are conveniently entirely self contained. The 3 primary functions, startup, supervisor mode and node mode all in their own .inc files however they regularly use RCALLs to perform tasks such as converting values into usable data, outputting data to displays, performing delays and displaying help menus. While it was possible to create a file which would have handled all of the character checking for Supervisor and Node modes, I felt that doing so could potentially cause problems and that it was easier to simply keep each of them self-contained.

When it came to displaying text, most data was stored as strings to make it easy to load from memory and output to the LCD or terminal, however when a small number of characters were being used, such as the capture results in Node mode and the system time in Supervisor mode, I would output each ASCII value individually rather than trying to store the obtained results into strings.

4 – Hardware design:

After doing an initial test with the startup program, I decided to rewire my entire breadboard as certain components, especially the 8515, were not in an ideal location.

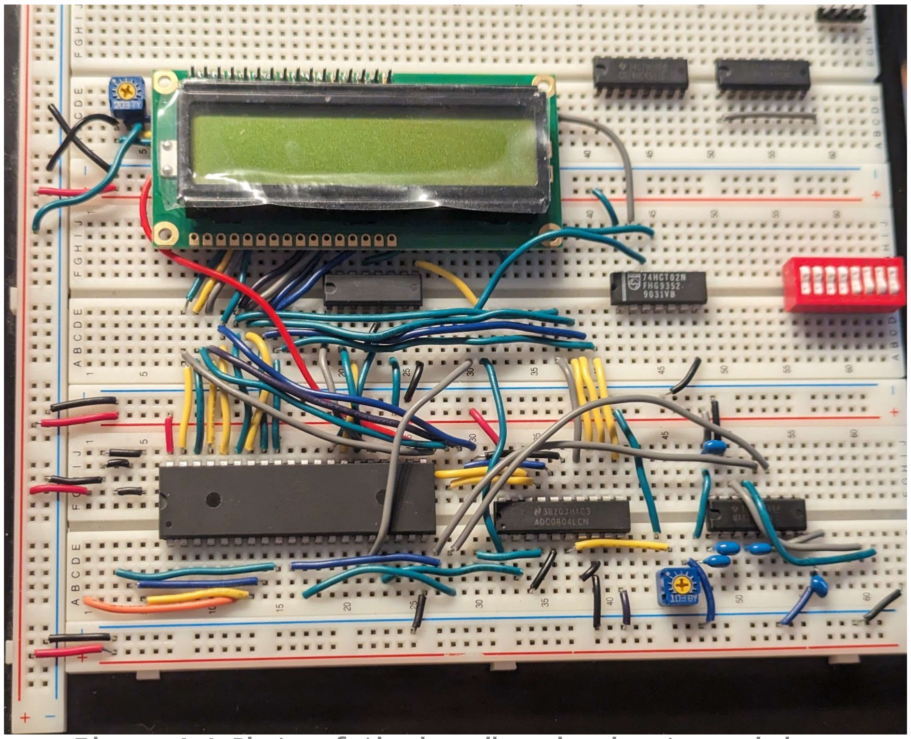


Figure 4.1 Photo of the breadboard prior to rewiring.

My initial design choices were as followed: The microcontroller would be placed in the centre of the breadboard with all output signals being placed above it and the input signals would be below while memory signals were placed on the middle track in order to shorten the connections between it and the 8515. I decided to use a 556 chip for the interrupt signal as I did not have any 555 ICs on hand, so it was more effective use of time to use the chip I already had. The memory addressing was configured to use 16k of memory where the I/O devices would be assigned spaces between 0x1000-0x27FF, the following calculations were made to determine the wiring of the 3-8 decoder:

log(16*1024)/log(2) = 14 address lines

Memory Addresses							Memory Decoder			
		A15-A12	**A11-A8**	**A7-A4**	**A3-A0**	**HEX**	**C**	**B**	**A**	**~Y**
ADC	**Start**	0001	0000	0000	0000	0x1000	L	H	L	2
	End	0001	0111	1111	1111	0x17FF				
7SEG	**Start**	0001	1000	0000	0000	0x1700	L	H	H	3
	End	0001	1111	1111	1111	0x1FFF				
LCD	**Start**	0010	0000	0000	0000	0x2000	H	L	L	4
	End	0010	0111	1111	1111	0x27FF				

Figure 4.2 Table containing the memory addresses of each of the hardware components and the associated outputs on the 3-to-8 decoder.

A NOR gate was used to combine some of these Y signals with the /WR signal from the 8515 in order to properly control the LCD and 4511 decoders for the 7 segment displays. The latter of these necessitated a second NOR in order to invert the signal, effectively turning it into an OR gate.

While most of the ICs on the board are in the same orientation as the microprocessor, the memory decoder was rotated 180° due to the placement of it’s input pins so that they would be aligned with the 8515’s address lines and the output pins would be aligned with the control pins of the ADC. A second change from the original planned positioning was necessary as the 7 segment displays were moved in order to place resistors between them and the 4511 decoder chips, which resulted in the displays being placed on the middle section.

The final design had the ADC, microcontroller, memory decoder, NOR chip and 7 segments on the middle row, the LCD, 4511s and resistors on the top row and finally the sample potentiometer, the connections for the AVR-ISP-mk2, the 556 chip and the serial communication chip being on the bottom row.

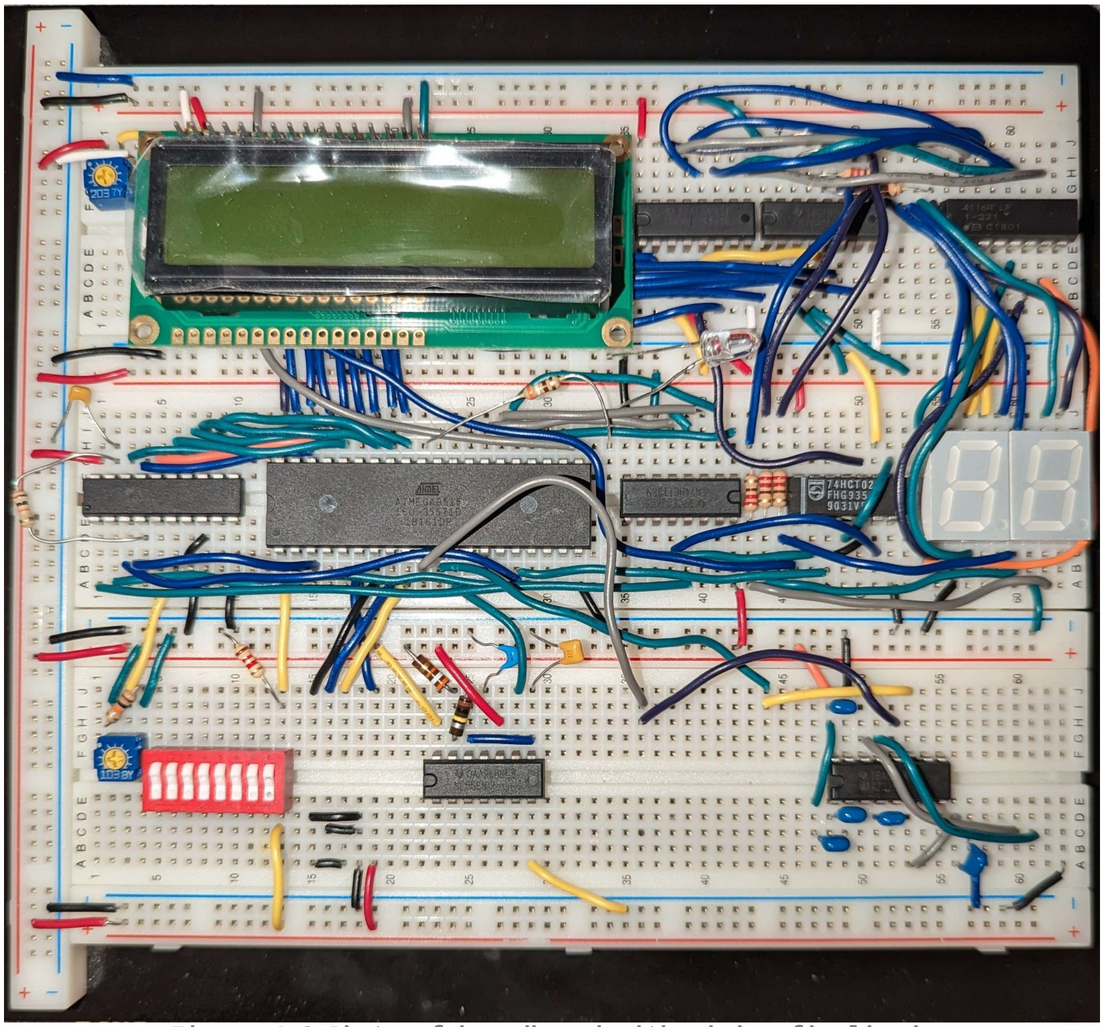
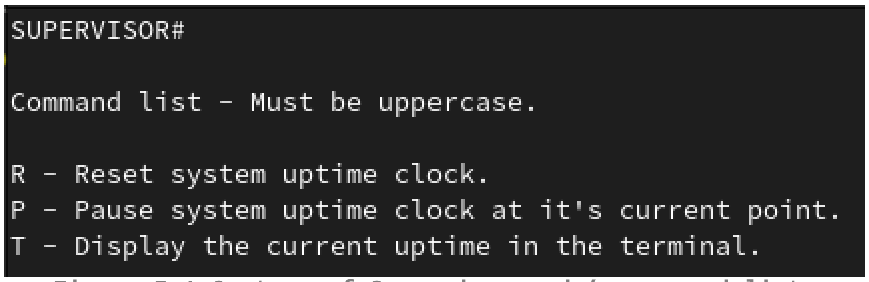


Figure 4.3 Photo of breadboard with wiring finalized.

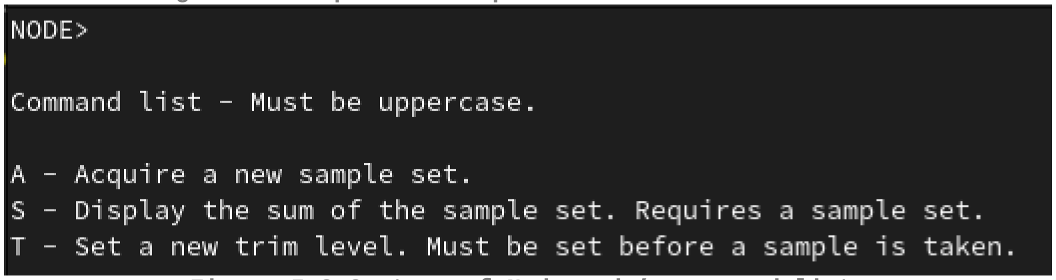
5 – Extra feature:

I had very little time to come up with a massive extra feature, so I decided to try and improve the terminal experience as much as possible. A point that is often not considered by designer is QoL and that a user may not be as technically literate as them, so it’s important to consider this and to effectively make running the program as simple as possible and explain every little detail. While I do see the irony in me writing this while not properly commenting my code, I’m just going to pull an old move from IBM in 1983 and say that this program is object-code only, so the end user shouldn't be able to know. All jokes aside, I added several strings containing the valid commands list for both modes of operation and ensured that they would be displayed every time a user entered Supervisor or Node mode. These lists indicated how to use the command and what they’re primary function was.



```
SUPERVISOR#

Command list - Must be uppercase.

R - Reset system uptime clock.
P - Pause system uptime clock at it's current point.
T - Display the current uptime in the terminal.
```

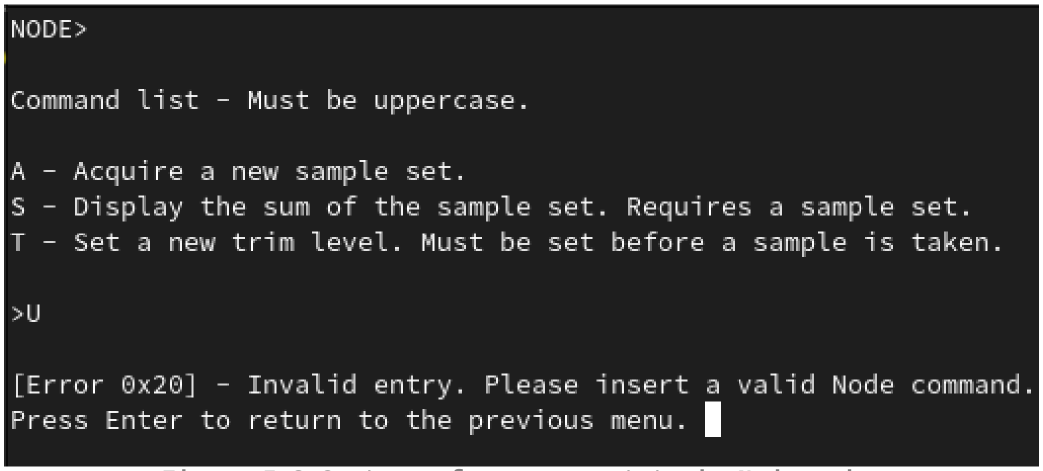
Figure 5.1 Capture of Supervisor mode’s command list.



```
NODE>

Command list - Must be uppercase.

A - Acquire a new sample set.
S - Display the sum of the sample set. Requires a sample set.
T - Set a new trim level. Must be set before a sample is taken.
```

Figure 5.2 Capture of Node mode’s command list.

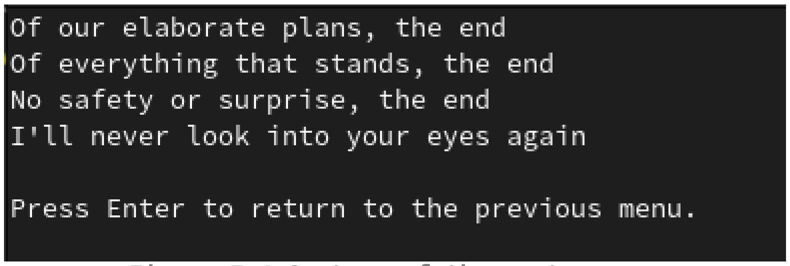
In the case of reseting the system clock in Supervisor mode, I also added warning message (see figure 2.2.2) asking the user to confirm that they wanted to reset the clock as there should only be specific circumstances where such an operation is performed. Error messages and associated codes were also added to inform the user when they are in an error state and what caused said state.



```
NODE>

Command list - Must be uppercase.

A - Acquire a new sample set.
S - Display the sum of the sample set. Requires a sample set.
T - Set a new trim level. Must be set before a sample is taken.

>U

[Error 0x20] - Invalid entry. Please insert a valid Node command.
Press Enter to return to the previous menu.
```

Figure 5.3 Capture of an error state in Node mode.

In addition to the above mentioned QoL features, I also added a reference to the song *The End* by *The Doors*, this reference has nothing to do with the program, nor does it have anything to do with my preferences in music. Rather this is a reference to the proceeding line in the song which was found commented in one of the provided .inc files for this project. To access this easter egg, hit E during the system startup before it enters Supervisor or Node mode, I would have used the END key to access this menu but unfortunately it uses scan codes rather than ASCII, and I wasn't willing to write more code for an easter egg.



```
Of our elaborate plans, the end
Of everything that stands, the end
No safety or surprise, the end
I'll never look into your eyes again

Press Enter to return to the previous menu.
```

Figure 5.4 Capture of the easter egg.

6 – Conclusion:

This PBL has been quite the experience.

I had quite a hard time at first wrapping my head around some of the tasks that were assigned to us for this assignment, so I spent a long time reading through code from old labs and datasheets. I was initially not able to see how we were supposed to create such a big program with Assembly, which lacks many of the modern comforts that most programming languages have today. People often say that you have to take a step back to see the bigger picture, and while this is somewhat true for ASM, there is a massive astrix. The bigger picture is comprised of smaller elements which are themselves comprised of even smaller details.

I began to write code, no matter how inefficient it was as long as I could get it to work properly, optimization was only thought of once a working version was made. Slowly, these elements began to take the form of the bigger picture and I had realized that by not focusing on the entire project at once, I had already completed ~60% of the entire program over the course of two days. And while much of this code was not the best way of implementing these features, the time I had saved by using this approach gave me more time to focus on improving the program piece by piece. I also saved time by reusing code as much as possible; the Supervisor mode’s terminal operations are more streamlined than those in Node mode because the latter was designed first and had more time put into debugging and improving upon it whereas the former had more QoL features implemented on top of the work which had previously been done. Ultimately, I familiarized myself thoroughly with the 8515’s instruction set and made the most out of it’s abilities to make the program in the way that I saw fit.

In the end, this entire process showed me the strengths of Assembly and how it can be used to write incredibly compact and efficient code for embedded systems. And while I believe that writing an entire program using it would probably fall under some form of insanity, being able to write libraries or functions in Assembly is an incredibly useful skillset to have.

7 – Performance:

To view the version history of this project, click the link below and check the commits history.
https://github.com/nate-sw/ESH/tree/main/pbl

Various notes taken during the R&D process of this project can be found in pbl_notes_scan.pdf