Saturday, December 8, 2018 10:57 AM

logbook...

50.002 1D Logbook

Ho Jin Kind (1002846)
December 13, 2018

My 1D project logbook for 50.002 - Computation Structures.
Group:
Ho Jin Kind, Myo Oo, Joanna Saw, Sean Liew.

Page	Log Date	Task	Details
2	Week 1	Mini Hardware Project	Project Scoping/ Planning
4	Week 2	Mini Hardware Project	Soldering
5	Week 3	Mini Hardware Project	Working with FPGA
6	Week 6	1D Project	Building 16 bit ALU
7	Week 7	1D Project	Creating Automatic Testing
8	Week 8	1D Game Ideation	Coming up with few ideas
9	Week 9	1D game prototype	Materials required.
10	Week 10	Hardware configuration	Figuring out the hardware
11	Week 12	Software configuration	Implementing the Mojo
12	Week 13	Reflection	Reflection

Mini Hardware Project 1

Project Scoping/ Planning

The first project that our group was assigned was the Mini Hardware Project (MHP). We were tasked with creating a 1-bit full adder on a stripboard. As we were all new to soldering and using stripboard, it originally seemed like a daunting task.

On our first meeting, we planned on the different task we had to accomplish during the project:

1. Design circuit/schematics for the 1-bit full adder .
2. test circuit on a breadboard so as to ensure it works.
3. Design the placement of different wires on the stripboard.
4. Prepare wires and cutting them of various length for stripboard.
5. Solder components onto a stripboard.
6. (BONUS) Program the Mojo FPGA to automatically test our circuit.
7. Produce a poster and record video of our working prototype.

We met up to work on the 1-bit adder. My groupmates had already designed the truth table for the adder, and we decided to go with using just 2 integrated circuits (ICs) instead of 3. We realised that we just needed a XOR gate and an AND gate for the 1-bit full adder to function correctly.

After some research. The function of a full-adder circuit is to input three one-bit numbers

After some research, the function of a full adder circuit is to input three one-bit numbers and output two one-bit numbers, also known as the Sum and Carry out.

By constructing the Truth Table as seen in Figure 1, the Boolean expression for Carry out and Sum can be formed and logic circuit can be form.

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 1 Truth Table for 1-bit Full Adder

As we continued, we realised that there were alternate design for full adders.For example, a ripple carry circuit (Arithmetic Logic Unit) and can be made up of two half-adder circuit(Fig 2). This would require 3 ICs. Whereas if used the design on (Fig 3), we would only need a XOR gate and an AND gate to function correctly.

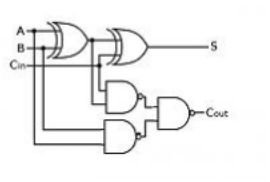


Figure 2: Circuit with 3 ICs

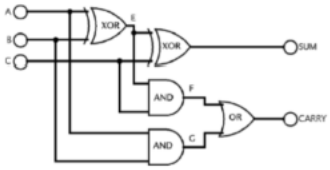


Figure 3: Circuit with 2 ICs

We then implemented the 1-bit adder design on a breadboard and tested it to ensure it worked.

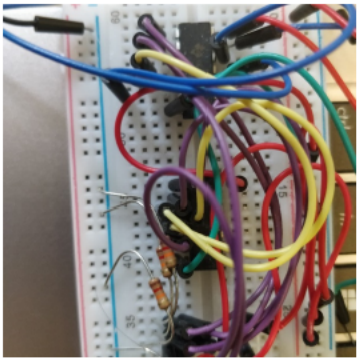


Figure 4: Breadboard prototype

We then moved on to designing the stripboard. We designed the schematic of our design digitally using excel at first, as we were not familiar with the tools available. Ultimately, we were able to use sophisticated tools like Fritzing to design the circuit and ended the meeting.

Summary:

On a side note, it was during this process where we realised a lot of the wires could be shortened by tapping on certain columns, such as 5 volt, which many required. This may seem trivial, but could save a lot of wiring in the long term. Furthermore, the circuitry looks a lot neater when we do not have a red wire pulling from original 5 volt column every time.

Also, although we were using rudimentary software such as excel, it allowed us to avoid common errors faced by many groups, and having to redo the stripboard portion due to messy wires criss-crossing everywhere.

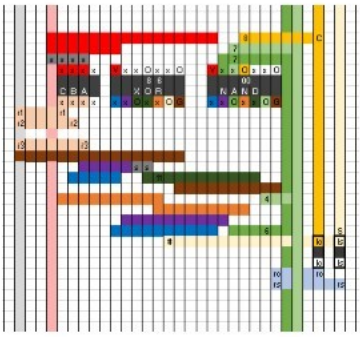


Figure 5: initial design stripboard using Excel

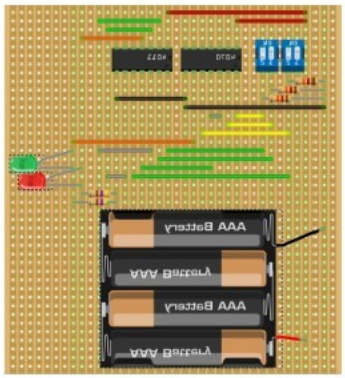
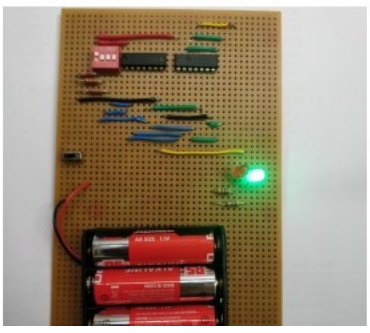


Figure 6: Design using Fritzing

Soldering

On our next meeting on week 2, we focused on soldering. It was unfortunate that despite the well planning, we were amateur at soldiering. This resulted in the first stripboard being wasted as there were too many short circuit. Our next plan of approach was that we were going to use the wasted board for practicing. After we felt that we were ready, we moved on to the next board. Although the soldering job could be a bit neater, it worked as expected.



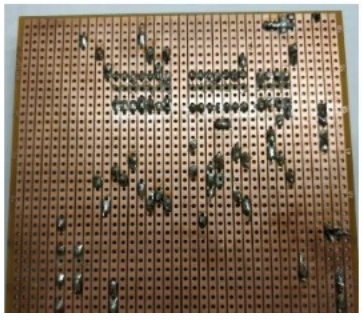


Figure 7 Front of full adder

Figure 8 back of full adder

Working with FPGA

We proceeded on to designing the FPGA tester. We started with trying the starter projects. We then started with the automated tester. The tester would have to iterate through 8 possible inputs (all possible combinations of A, B, and Ci). We would require a finite state machine (FSM) with 8 states. We programmed the Mojo to output the respective values to the output pins. It would then check if the inputs from sum and carry were as expected.

We made use of the LED strips of the Mojo as follows:

- The first three LED of the Mojo will display to output at the current moment.
- The next LED will turn on should there be an error in the input.

(Code uploaded onto GitHub.)

In the process of soldering the holder, I made the mistake of soldering the holder on the wrong side of the carry and sum LEDs. This resulted in me getting erros on my Mojo on all combinations except when A, B ,c was 0. As I originally had the misconceptions with LED having a low resistance, I presumed that it was only after the resistors that the voltage will be grounded. After looking back at my physics notes, I understood and switched it to the other side.

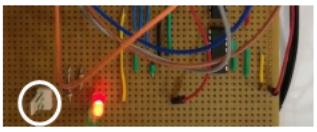


Figure 9 Circle where placeholder was soldered wrongly across LED will be close to 3.3v

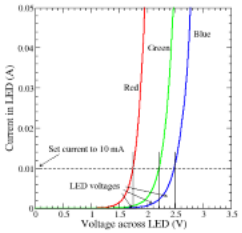


Figure 10 Voltage

After fixing it, our tester worked. For the manual part, we created additional states, and made used of the io_dip switches as our A, B and Cin.

The programming of the FPGA was very different from conventional programming languages.
Understanding the syntax of the Mojo IDE was rather difficult, especially understanding how the clock works. It was a relieve however, when we were able to fall back certain concepts such as FSMs which really allowed me to built upon the foundation of understanding the working of FPGA, and putting the pieces together.

Summary:
Working with stripboards and FPGAs was a new experience. From this, I learnt on the importance of soldering and being prepared. This could prevent us from wasting stripboards and other parts unnecessarily. Hence, proper soldering techniques are essential when doing any such projects. Working with the FPGA through the mini hardware project was also a good way to ease me into using it, experimenting before we have to do the harder projects.

1D Project (16-bit)

Project Scoping/ Planning

Our first meeting was on Monday of week 6. Having done the software lab, and were familiar with the how to programme it.The difficulty were to figure out the states required in our FSM. We spent some time looking around for how to design a 16 bit adder, and thought of how we were going to organize our code. 1 issue we faced was to allocate the switches, as we needed more switches than 24, due to input A an B. For code organising, we agreed in unison that to modularize our code was the best approach. Started with programming the adder, comparator, boolean, and shifter units, which was relatively simple. Each unit was its own .luc file, and so was the ALU, in order for us to work with them conveniently.

For the inputs, we realised we needed 16 for A, 16 for B, and 6 for ALU. Joanna was in charge for the encoding of different ALUFN values in the lucid file.

Operation	*ALUFN[5:0]*	*hex*
ADD	000000	0x00
SUB	000001	0x01
AND	011000	0x18
OR	011110	0x1E
XOR	010110	0x16
“A” (LDR)	011010	0x1A
SHL	100000	0x20
SHR	100001	0x21
SRA	100011	0x23

CMPEQ	110011	0x33
CMPLT	110101	0x35
CMPLE	110111	0x37

Figure 11:ALUFN values corresponding to different operation

As for the lack of switches, the set up we came up with was for an initial state called listenA, listen, and process.

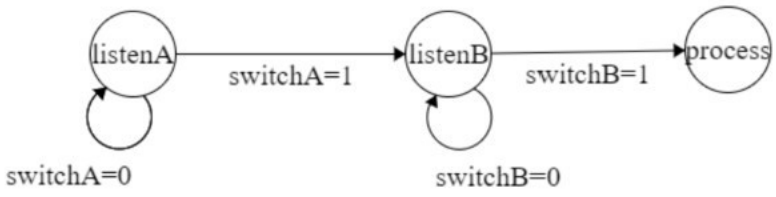


Figure 12 Manual

Manual:

Figure 13: state A

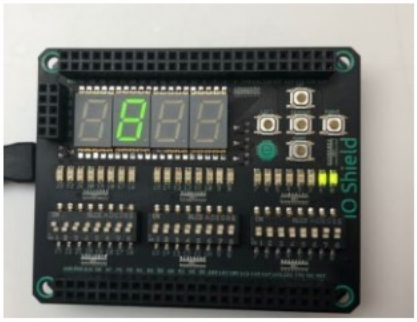


Figure 14:state B

How it works: The first 16 switches will be used to take in the value of A, when the toggle switch for A switches ON, this will save the value of the inputs of the 16 bits to var A, and move on to state B. After state B switch is also toggled to ON, the input at that instant will be save to var B.

```
v = a[15]&&b[15]&&!adderModule.result[15]||!a[15]&&b[15]&&adderModule.result[15];
    case(alufn) {
       Inst.CMPEQ:             // Compare Equal To
           result[0] = a == b;         //z
       Inst.CMPLT:             // Compare Less Than
           result[0] = adderModule.result[15] ^ v;       //n XOR v
       Inst.CMPLE:             // Compare Less Than Equal To
           result[0] = (a==b)||(adderModule.result[15] ^ v);  //z OR (n XOR v)
```

Figure 15: Need for values Z, V and N in compare module

One issue we faced was with the compare less than. Although values of Z, V and N were not required in the adder, to account for overflow situation, it was needed to get the correct value when

comparing less than.(for example, negative number A and positive number B will give an erroneous value as A<B in lucid will assume it to be unsigned, and show that A is larger). Hence, we included the value for overflow within the compare module, along with Z and N. This allowed us to catch those cases.

Automatic Testing:

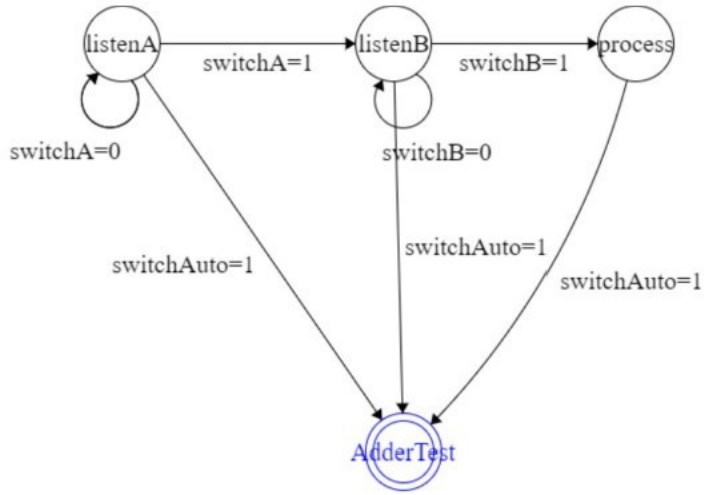


Figure 16: To be able to jump to automatic test from any state,

For automatic testing, we include another switch, we when pressed, as it is at the end of each state, would override and set next state to AdderTest1.

```
        storeb=c{io_dip[1],io_dip[0]};
      if(io_button[4] ==1){
        state.d=state.ADDERTEST1;
      }
```

Figure 17: Jumps to automatic test from manual.

This will trigger the automatic tester, which will go through all the test cases that we designed.

There were a total of 6 adder test, 3 multiply, 5 boolean, 5 shifter test cases and 8 Compare test cases.

For the test cases, as it test case was a state itself, we displayed on the seven segment the test case number. The last test case would be a fail case, where the hard coded output is a different value. This would result in an error, which would go to our respective error cases for each operation.

```
state.ADDERERRORTEST://Simulated ADDER error test case: 5+7=12 (+1)
alu.alufn = 6b0;
alu.a = 16b101;
alu.b = 16b111;
seg.values={4d10,4d4,4d4,4d5};
//ignore the line to display A bits
//ignore the line to display B bits
//ignore the line to show "A" on seven_seg
if (counter.q[DELAYTIME] ==1&&alu.result+1==16b1100){
    counter.d = 0;
    state.d = state.BOOLTEST1;
}
else if (counter.q[DELAYTIME] == 1 && alu.result!= 16b1100){
    counter.d=0;
    state.d=state.ADDERERROR;}
else{
    counter.d = counter.q+1;
}
```

Figure 18 test case which will lead to error(alu.result +1)

```
  state.ADDERERROR: //ADDER ERROR
  //display something on seven_seg to show error


  seg.values={4d9,4d14,4d0,4d9};// display "no"
  if (counter.q[DELAYTIME]==1){ //once counter reaches defined value
      counter.d = 0;
      state.d = state.MULTIPLYTEST1;} //move to test the next unit
  else{
      counter.d = counter.q+1;
  }
```

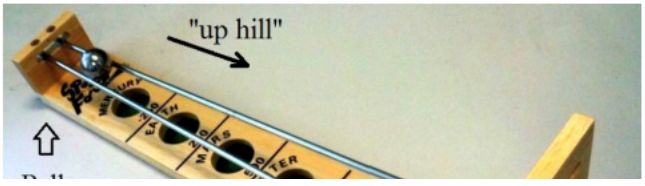
Figure 19:AdderError state

Our code was thus able to go through the entire test cases, and loop back to listenA state at the end. Passing all the testcase, and going to Error test case properly when it needed to.

1D Project meeting 1:

Game Ideation

Game 1: shoot the moon



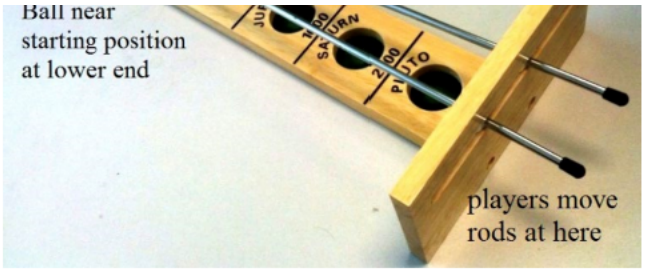
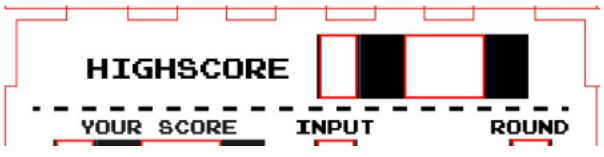


Figure 20 our game with electronics embedded into vintage game

A single player game which does an arithmetic operation every round, depending on which pot you fall into. This operation is done between *Your Score* & *Given Input*. Each hole in the board equipped with a crash sensor connected to the MOJO top. Each of it represents a specific ALU Operation to be performed between *Your Score* & *Given Input*. Since the **hole highest up** the board is the **hardest to achieve**, it will trigger the multiplication operation, followed by division, addition and subtraction. (illustrated in picture).

The below table shows how the game could go down.

A (Your Score)	B (Given Input)	Chosen Operation
0	1	Add
1	2	Add
3	3	Multiply
9	4	Multiply
…	…	…



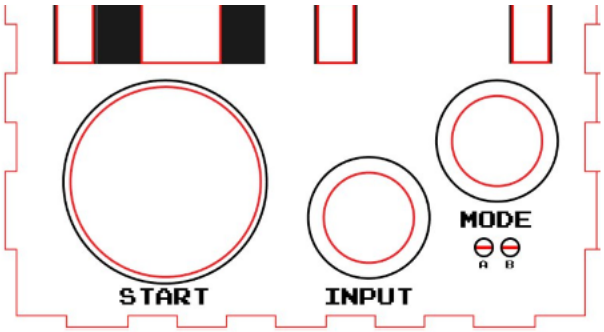


Figure 21: User interface of our game

This is a mock up of the UI. As you can see, it consist of the score, next input, highscore, led which marks which mode you are on, and which round you are at.

Mode:

For the second mode, it is basically an extension to the previous idea. Based on the risk-taking nature of the player, the user can choose the next input value, ranging from 1-5. This can allow the user to hit a potentially much higher score(multiply on positive), or much lower (multiply on negative).

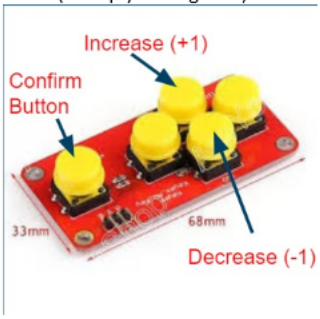


Figure 22: Increase or decrease next input value strategically before round starts.

Review:

This game although interesting in the sense it mixes in mechanical aspect of the game to the electronics, requires a lot more wiring in terms of multiple seven segment. I like this idea however the existence of motivation for the player, in terms of this high score element. As long as high score is not reached, there is always incentive to try for players.

Game 2: Maze runner

In this game, Users will be given a question, an unsolved equation and arithmetic operations at the 4 corners. Users will start from the centre of the maze, and navigate to collect the operations to fit into the equation. The order is of course paramount, to fill in each one of the operations.

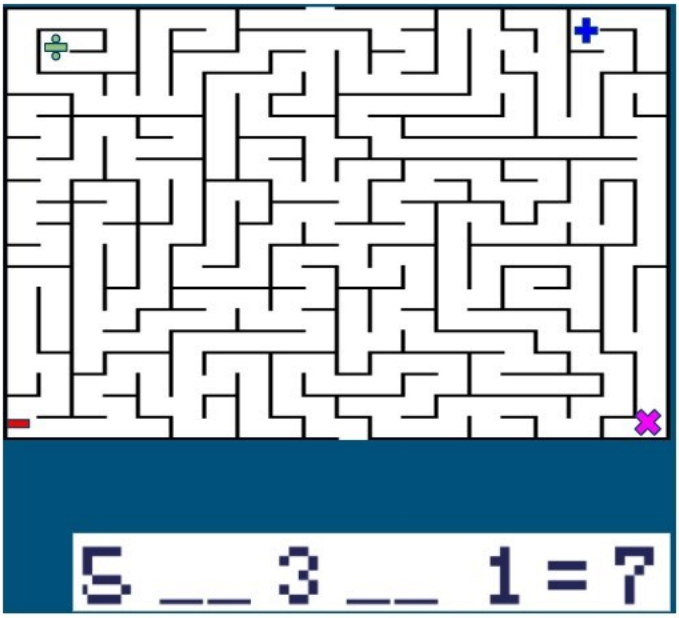


Figure 23: user interface of game

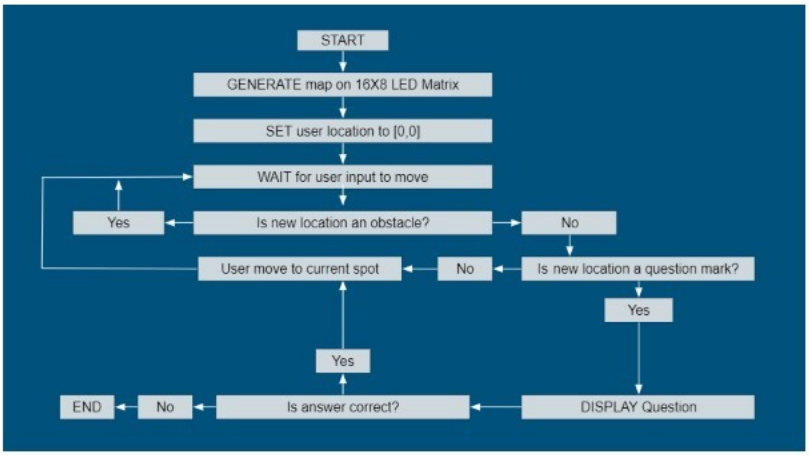


Figure 24: State machine for game 2

Modes:

1st Round: 1 operation

2nd Round: 2 operations

3rd Round: 3 operations

Review: I liked this game personally due to the endless possibilities of the equations. However, it is

Review: I liked this game personally due to the endless possibilities of the equations. However, it is very difficult to generate the map on the LED matrix.

Game 3: Boolean or not to Boolean

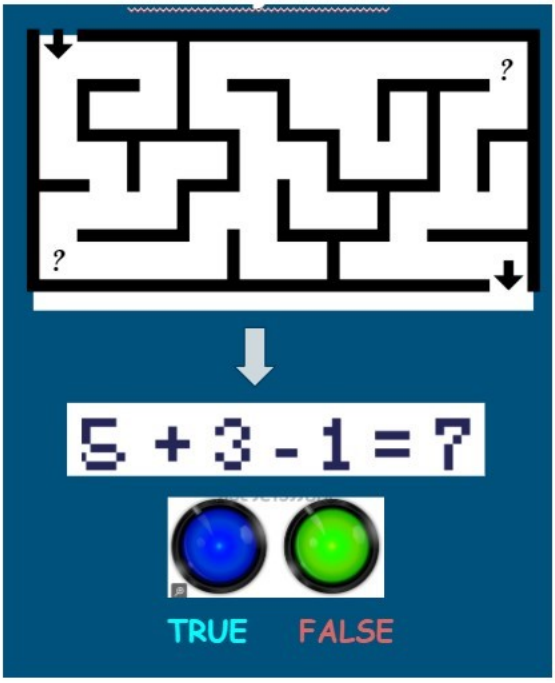


Figure 25: design of game 3

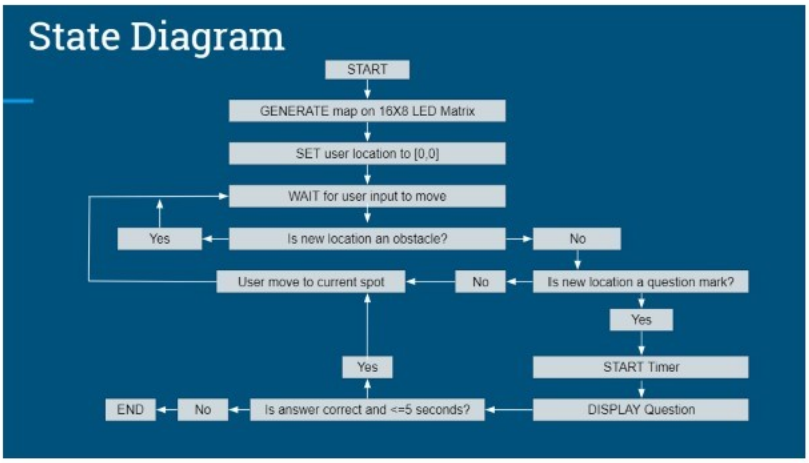


Figure 26: State diagram of game 3

Users will start at the top of the maze. Navigating through to reach the endpoint. When they encounter a question mark, they will be navigated to a different screen with an operation, where they have to pick true or false. If they pick the wrong option, they will have to restart.

Modes:

1st Round: 2 question marks

2nd Round: 3 question marks

3rd Round:5 question marks

Review: This game is simpler than the second one, requiring just a true false button. However, it is very difficult to generate the map on the LED matrix just like the second game.

Game Prototype

After settling for game 1, shoot the moon, we sat down to decide on the hardware which we would need.

This consisted of:

- 1 start button
- 1 toggle mode button
- 1 increment button
- 4 single seven segments
- 2 multisevensegments
- 2 LED bulbs
- Wood for casing
- Vintage game

We also decided on the electronic components that we would required, and started to order them so that we could work on it as soon as possible. We bought our electronic components from Simlim, and wood from BookLink.

Figure 27: Some of the hardware we got from sim lim

Hardware configurations

For the hardware, we begun by trying to make the multi seven segment work. Doing it on the mojo IO shield was easy, as we did not need to figure out how to hardware works. However, with an external seven segment, we realised the amount of wiring needed.(7 for led, 4 to select which digit). After some trial and error, we made it work. The mojo side was identical to the one IO shield, but we had to select the IO pins ourselves.

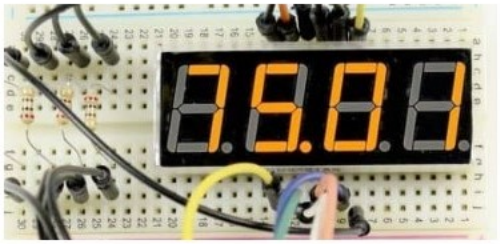


Figure 28: External seven segment

As for the laser cutting of the box, Joanna was in charge of the task.I was surprised she managed to get a laser cutting slot from the ASD students which hung out at the laser cutting room 24/7.

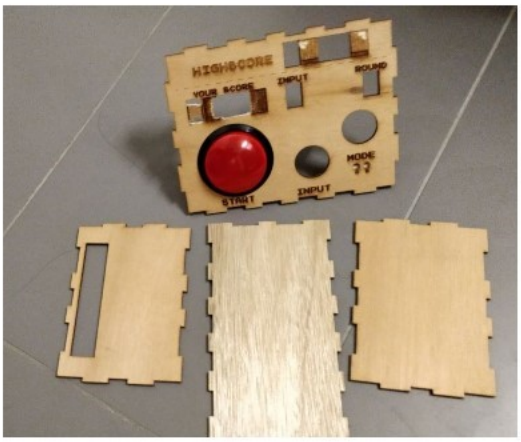
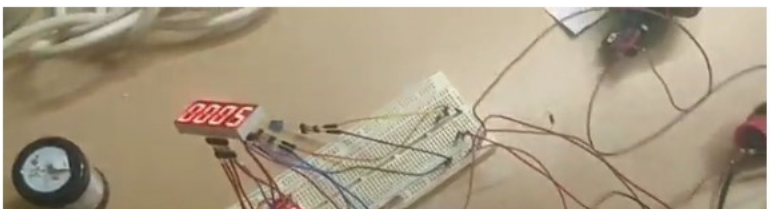


Figure 29: some of the pieces

Moving on to the soldering aspect. As we were amatuers in these aspect. We did not design the hardware to be proper. The circuit was unstable and frequently shorted. ON hindsight, we could have used a tri board to make our lives easier, but we did not think about that when we started.



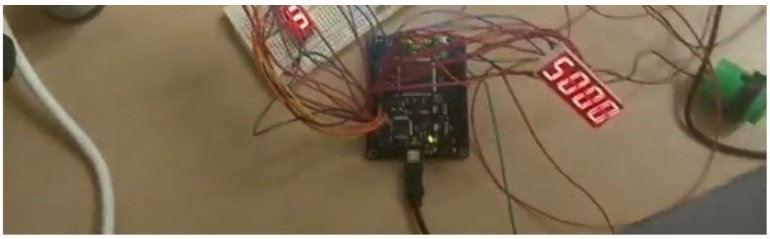
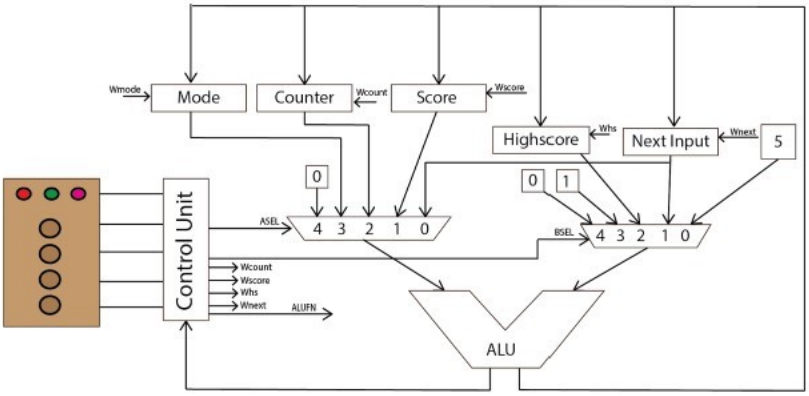


Figure 30: Exposed wiring and wires come out easily

Software configurations

On the software side, we begun by implementing the beta.



This was our beta architecture. To implement it in mojo, I created a module called emulator which acted like the beta.

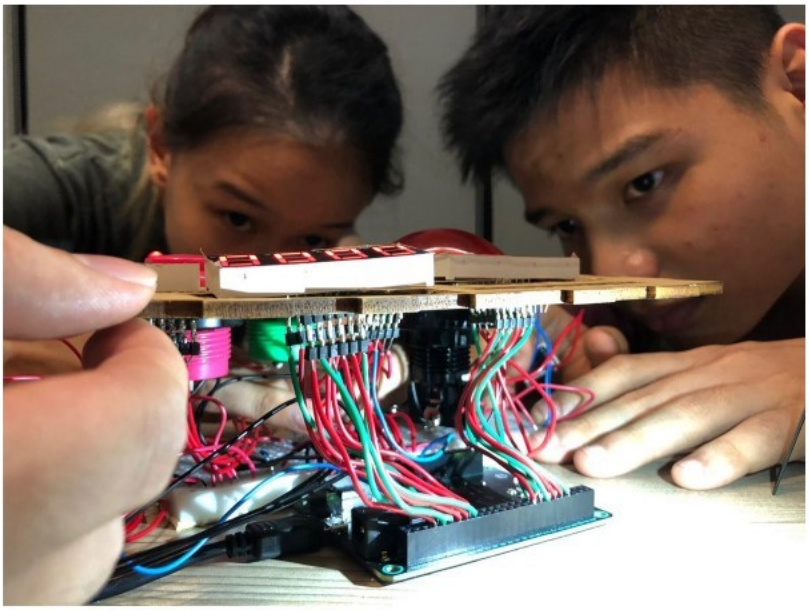
```
always {
  highscore.write=whs;
  currentscore.write=wcs;
  b.write=wb;
  mode.write=wmode;
  counter.write=wcounter;
  mux_a.sel = asel;
  mux_a.a= currentscore.out;
  mux_a.b = mode.out;
  mux_a.c = counter.out;

  mux_a.d = b.out;
  mux_a.e = 16b0;
  mux_b.sel = bsel;
  mux_b.a = 16b1;
  mux_b.b = 16d5;
  mux_b.c = highscore.out;
  mux_b.d = b.out;
  mux_b.e = 16b0;//nth
```

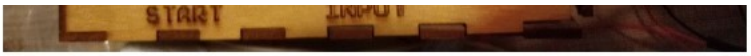
```
    alu.a = mux_a.out;
    alu.alufn = alufn;
    alu.b = mux_b.out;
    aluresult = alu.result;
    b.value=alu.result;
    counter.value=alu.result;
    highscore.value=alu.result;
```

The writing into the register will only write in when write signal is 1.

Putting it together



This was the hardest process, and after much trial and error, we got it to work as intended.



Reflection:

This was a long project. Learning how to code on Mojo allowed me to implement things we learn in class directly to an actual project, which was rare. Learning configurations and how hardware works also played a large role. However, I felt it was a pity that I was not able to get the required electronics from cheap online websites such as Aliexpress as it takes too long. Perhaps getting them in bulk through electronics lab, would be a good idea, lowering cost, and supplying us through there. This would make it more effective for students and also cost savings. Perhaps a list could be consolidated of the electronics bought, used by current batches, import a small sample size, which could be used for future batches.