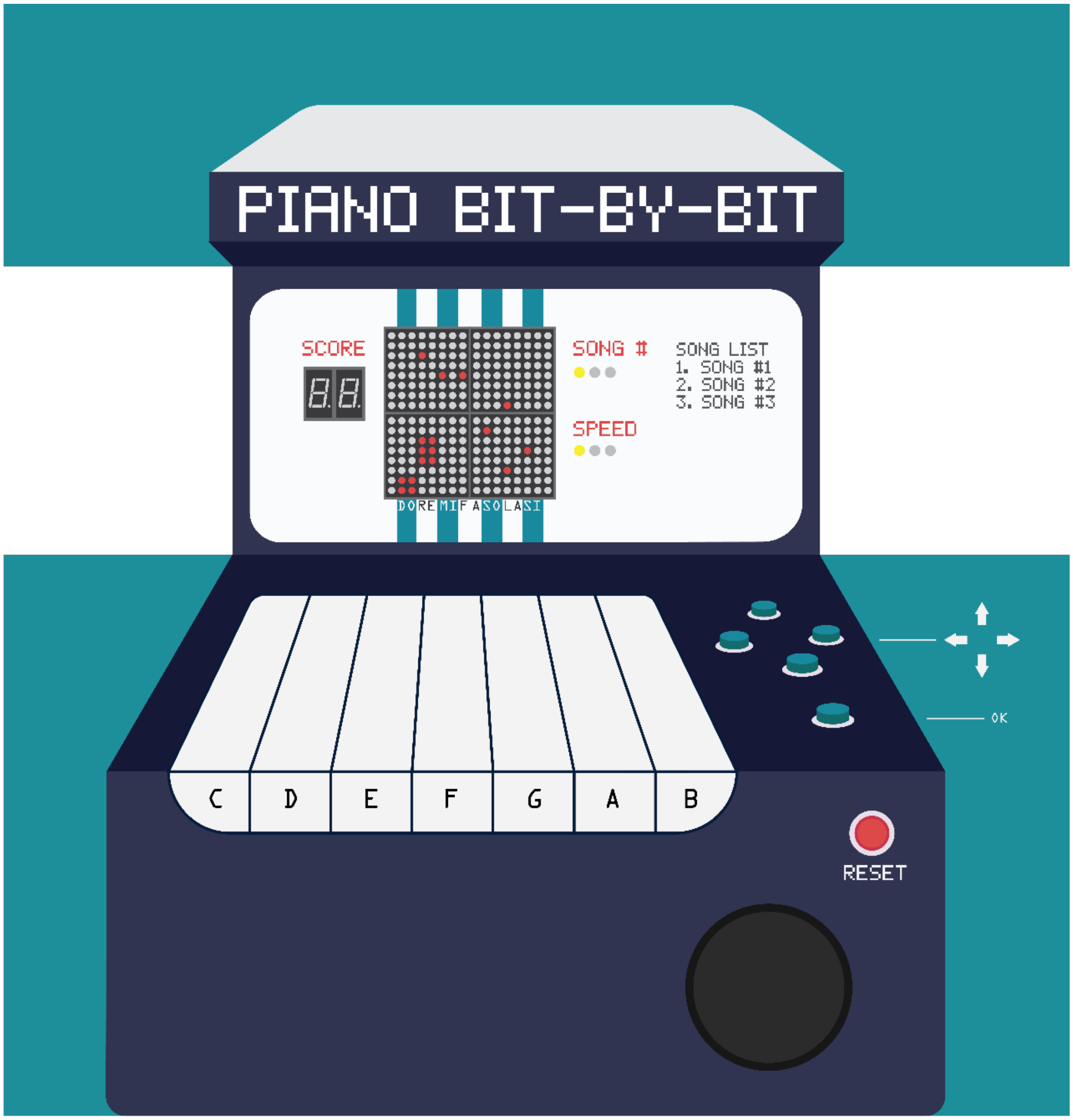


PIANO BIT-BY-BIT

50.002 COMPUTATIONAL STRUCTURES
1D REPORT

Angelia Lau	1002417
Kimberlyn Loh	1002221
Li Xingxuan	1002189
Zhao Gualun	1001429

Table of Contents

1. Introduction

The company Digi-alpha built and designed a 32-bit ALU (arithmetic and logic unit) named Alpha. A certain chip maker company is showing interest in buying Alpha and request for a working gaming prototype that will be donated to orphans as part of their corporate social responsibility program. The prototype will be design and built around a singular 8-bit ALU using an FPGA (Field Programmable Gate Array) and educative.

This project aims to build a piano game that utilize an octave of a piano keys for the gameplay. Players must press corresponding piano key of the 'falling' notes, which is displayed in the screen, in the correct sequences and timing.

2. Game Description

The game is based on an alternate version of whack a mole, where the player must follow the musical notes in the correct sequence. This aims to teach the player about how to briefly play the piano, be more alert and train on their hand-eye coordination.

2.1. GAME DESIGN

This game offers dynamic gameplay that hooks in the player by offering the following:

1) Different Speed
   This aims to make the game challenging depending on the choice of the player to play the song at usual speed or slower than the usual speed. The scoring system is the same so as a measurement of whether they have improve from choosing a different speed.

2) Keyboard and Housing Fitting
   The user interface of the game depicts the exact replica of an octave of the keyboard and the housing is shaped slightly like the frame of the piano. This is to allow the player to have a feel of what it will be like to be actually playing on the piano.

3) Different Song
   With 3 different type of song, player can choose their favorite song to challenge and compete with their friends. They can also come back and practice on their favorite song or challenge another type of song.

2.2. TEST SCENARIOS

The test scenarios are as followed.

1) Song selection
    a. Testing every song
        i. Ensuring the tone of the song is correct
    b. Not choosing a song
        i. Ensuring that the default song (choice 1) is chosen

2) Speed selection
    a. Testing every speed
        i. Ensuring the tone and speed of the song is correct
    b. Not choosing a speed
        i. Ensuring that the default speed (fast) is chosen

3) Game Play
    a. Pressing the correct key
        i. Ensuring increment of score
    b. Pressing the incorrect key
        i. Ensuring decrement of score
        ii. Ensuring the score does not reduce to negative

3. User Manual

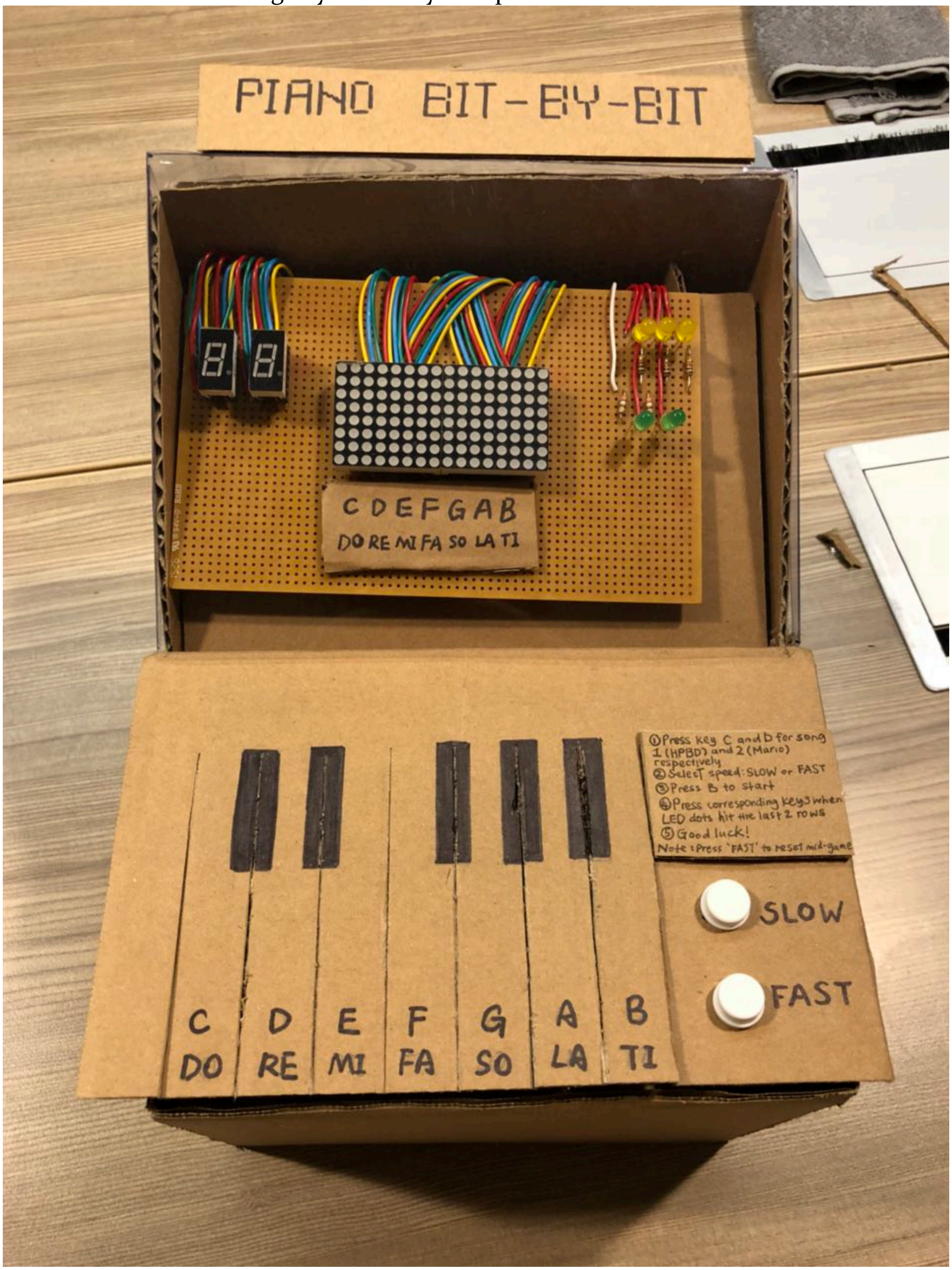
How to play:

1. Select song number using the buttons provide (Piano Key C and D)
2. Select speed level using the buttons provided (slow or fast)
3. Press B key (right most) button to begin game
4. Notes of the song will descend from the top of the LED matrix. Alternate rows in the led represent 1 key on the piano. When the note hits the last row, press the corresponding piano key.
5. Play till the end of song!
6. Press B key again

    Note: You can press FAST to reset the game midway (if you think you need another try)

How to win:

The score will increase when the correct keys are pressed and the score will decrease when the wrong key or no keys are pressed.

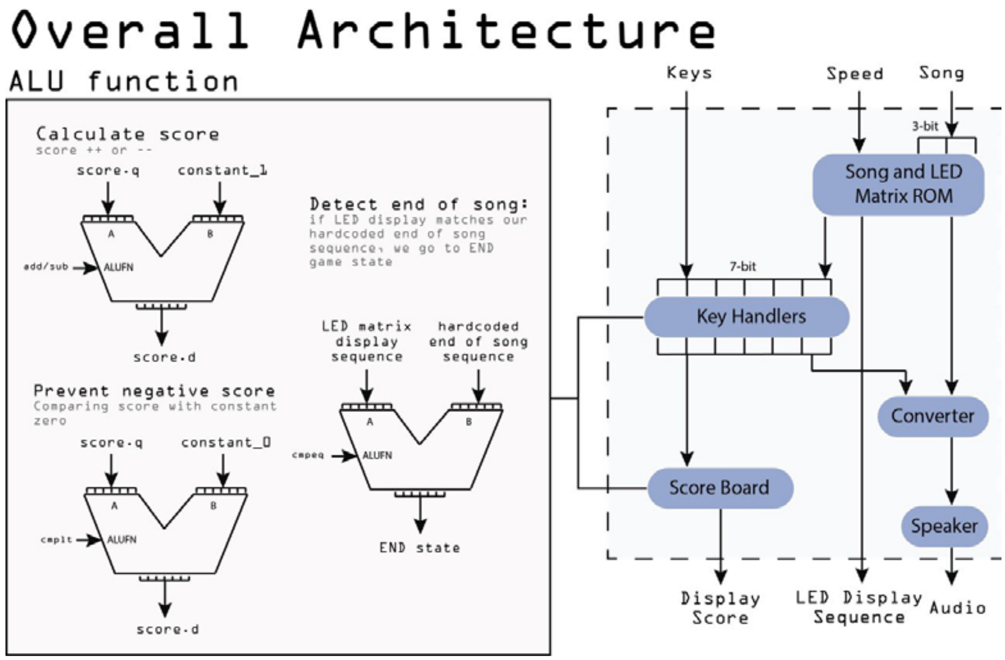


4. Building the Prototype

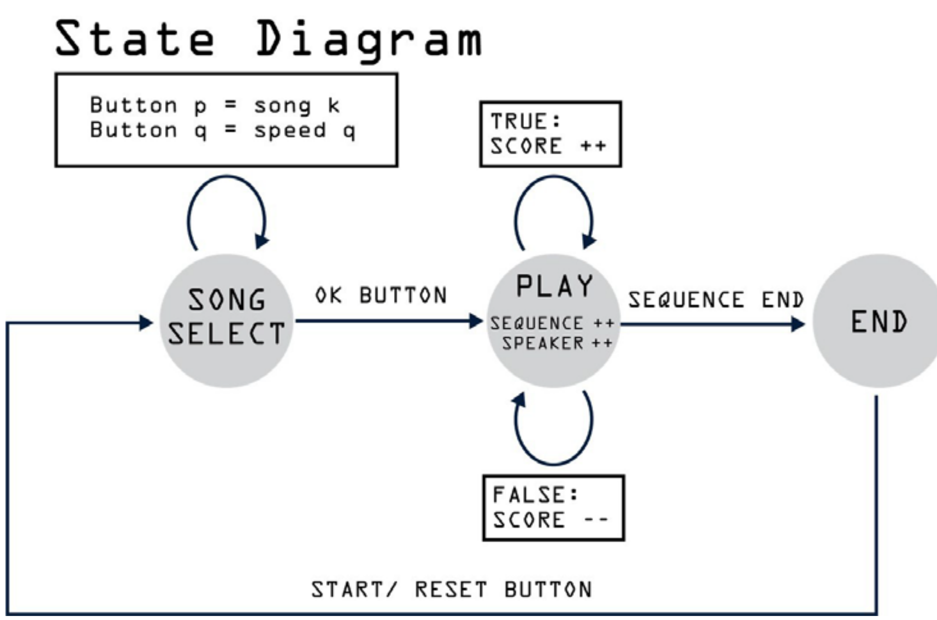
The prototype was built in the following stages:

4.1 SYSTEM DESIGN (SOFTWARE)

4.1.1 Overall Architecture



4.1.2 State Transition Diagram



There are 3 states: SONG SELECT, PLAY, END

1) SONG SELECTION:
   - Player selects song from piano key C and D which is based on the button press
   - Player then selects speed from slow or fast which is based on the button press
   - When player presses right most key button (B) , jump to PLAY state

2) PLAY:
   (Per note :)
   - If (key pressed == location of led)
     - ❖ Score ++
     - ❖ Sequence ++
     - ❖ Speaker ++ (speaker plays correct note)
   - Else (key not pressed correctly)
     - ❖ Score --
     - ❖ Sequence ++
     - ❖ Speaker ++ (speaker plays correct note)
3) END:
   - All LEDs will light up
   - Final score displayed in 7-seg display on top left corner
   - Jump to SONG SELECTION state

4.1.3 ALU function

The ALU function used for the project is as followed:

a. ADD to increase the score
b. SUB to decrease the score
c. CMPEQ to check if the song ended
d. CMPLT to ensure there will be no negative score

The ALU design and test can be seen in the Appendix below.

4.1.4 Software Implementation for LED Matrix

We use two 8*8 led matrices to display our notes. Each led matrix has 16 pins which is connected to mojo top. We hardcode the song notes into a file. In order to accomplish the goals of moving down the notes one by one, we take in the first 8 indexes of the hardcoded matrix at first and move it forward by one after each delay. The delay is accomplished by creating a global d-flip-flop.

```
if(gcounter.q ==0) {
  mycounter.d =mycounter.q +1;
  }
```

Figure 4.1.4.1 Code Snippet for counter

To display each frame, we create a function and it takes in a list of eight 8-bit binary number. And it outputs the row and column values of the led matrix. Each of the row and column output is an 8-bit binary number. The row output is initiated as b00000000, and the initial value of column output is b11111111. For each frame, we display the eight 8-bit numbers row by row. Since the interval is very small, we can see an integrated frame.

```
r =b00000000;
c =b11111111;

whichrow =counter_r.value;
r[whichrow] =1;
for(i =0; i <=7; i ++) {

  c[i] =~cr[i][7 -whichrow];

}
```

Figure 4.1.4.2 Code Snippet for frame

4.1.5 Software Implementation for Speaker

In the implementation of speaker system, speaker is connected to mojo through 3.5mm stereo headphone jack adapter, where mojo can control the tone of speaker by sending out square wave of certain frequency. For the convenience of tone management, we convert our acoustic frequency range to octaves and notes according to the **Note-Frequency Chart** shown below.

Note Frequency Chart

	Octave 0	Octave 1	Octave 2	Octave 3	Octave 4	Octave 5	Octave 6	Octave 7	Octave 8
C	16.35	32.70	65.41 1	130.81 13	261.63 25	523.25 37	1046.50 49	2093.00	4186.01
C#	17.32	34.65	69.30 2	138.59 14	277.18 26	554.37 38	1108.73 50	2217.46	4434.92
D	18.35	36.71	73.42 3	146.83 15	293.66 27	587.33 39	1174.66 51	2349.32	4698.64
D#	19.45	38.89	77.78 4	155.56 16	311.13 28	622.25 40	1244.51 52	2489.02	4978.03
E	20.60	41.20	82.41 5	164.81 17	329.63 29	659.26 41	1318.51 53	2637.02	5274.04
F	21.83	43.65	87.31 6	174.61 18	349.23 30	698.46 42	1396.91 54	2793.83	5587.65
F#	23.12	46.25	92.50 7	185.00 19	369.99 31	739.99 43	1479.98 55	2959.96	5919.91
G	24.50	49.00	98.00 8	196.00 20	392.00 32	783.99 44	1567.98 56	3135.96	6271.93
G#	25.96	51.91	103.83 9	207.65 21	415.30 33	830.61 45	1661.22 57	3322.44	6644.88
A	27.50	55.00	110.00 10	220.00 22	440.00 34	880.00 46	1760.00 58	3520.00	7040.00
A#	29.14	58.27	116.54 11	233.08 23	466.16 35	932.33 47	1864.66 59	3729.31	7458.62
B	30.87	61.74	123.47 12	246.94 24	493.88 36	987.77 48	1975.53 60	3951.07	7902.13

©2011 ArtificialTunes.tumblr.com

Figure 4.1.5.1 Note Frequency Chart

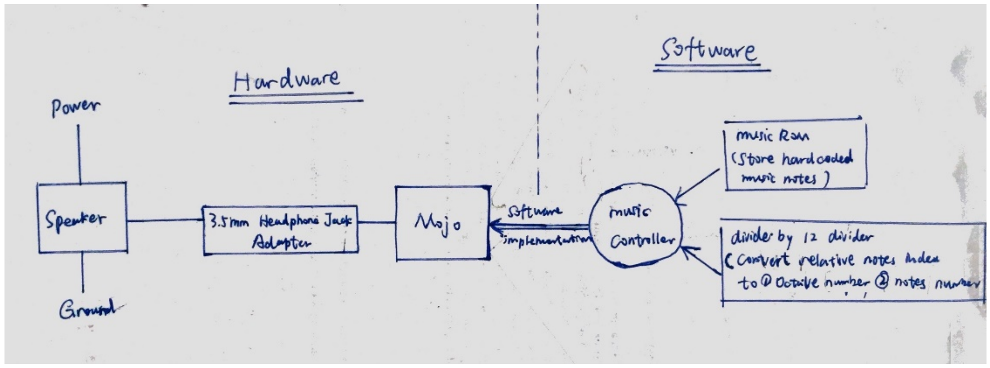


Figure 4.1.5.2 Illustration of the implementation

We know that each octave can be divided by twelve notes, and each next adjacent octave have doubled frequency of the current one. When composing our song in mojo FPGA, instead of hardcoding those arcane frequency directly, we encode a certain note by its **relative index**. Our music Read Only Memory (ROM) stores complete hardcode notes for our song so that our music logic can refer to anytime. For example, if we are going to play C note in Octave 4, we will store number 25 in our music ROM.

In our music control logic, we will take in the global clock given by our main program. After each clock tick, our music logic will fetch next note stored in music ROM, convert it back to frequency and send out the Pulse Width Modulation (PWM) signal to speaker.

4.2 SYSTEM DESIGN (HARDWARE)

4.2.1 Prototype

Sketches were made before the actual construction of the prototype for visualization purpose and can be view below at the appendix section of this report.

The actual prototype was built from cardboard and can be seen in figures below.

4.2.2 Assembly of System

1. The housing is built from cardboard and was glued together.

Figure 4.2.2.1 Top View of housing

Figure 4.2.2.2 Side View of housing

2. The circuitry for the keyboard was soldered and the cardboard layer was placed on top of it.

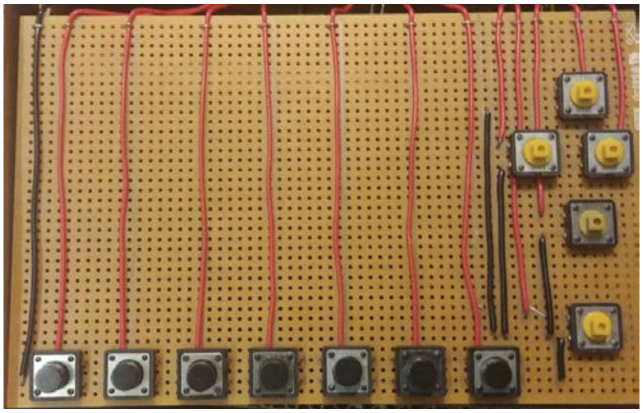


Figure 4.2.2.3 Circuitry of Keyboard

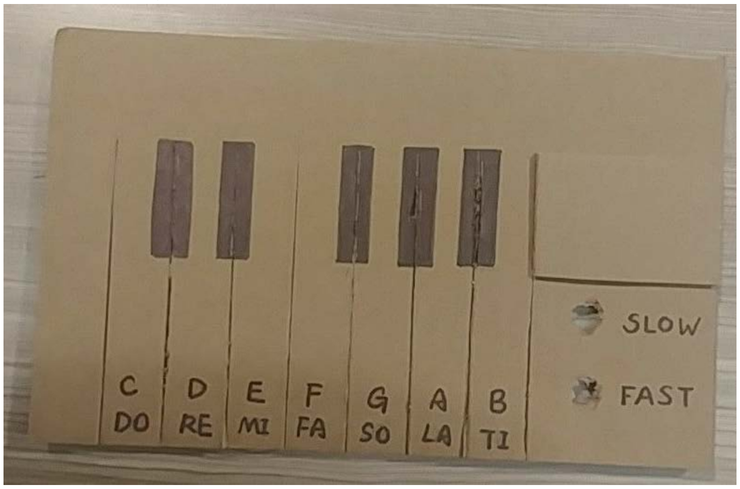


Figure 4.2.2.4 Cardboard Layer of Keyboard

3. The circuitry for the screen was soldered

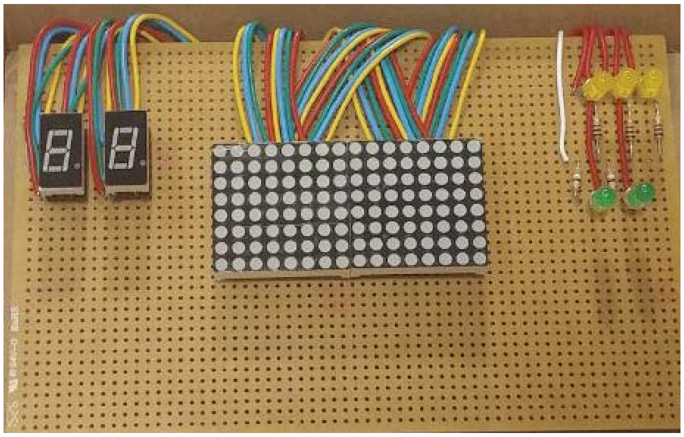


Figure 4.2.2.5 Circuitry of the Interface

4. Parts from 1-3 were placed together to form the system.

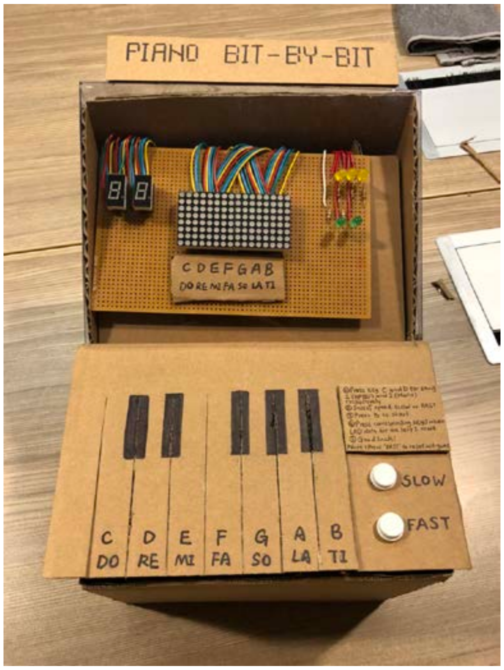


Figure 4.2.2.6 End result

5. Improvements were then decided and made. (The type of improvements made will be mentioned in the later section of this report.)

4.3 DESIGN ISSUES

The design issues faced during the project timeframe are as followed:

1. The interior of the housing requires a storage shelves to allow placement of the Mojo and speakers.
2. The keyboard interface also requires a backing below so that the push buttons can be pressed.
3. The speaker is too loud at the current location and hard to place in the housing.
4. The LED matrix was intended to be designed with serial peripheral interface (SPI) provided on the board given. However, we realize that the board was meant for Arduino and could not interface with MOJO.

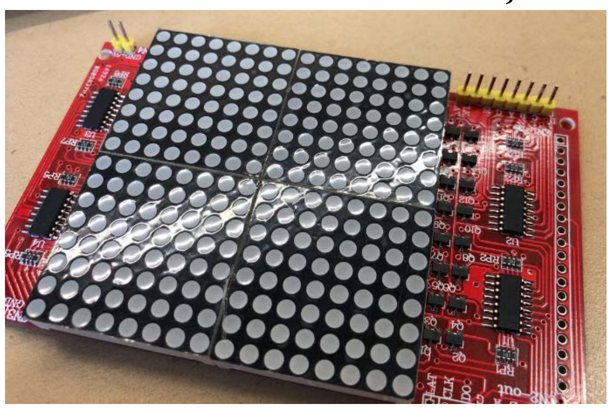


Figure 4.3.1 LED Matrix Board

4.4 PROBLEM SOLVED

The team managed to solve the design issues that are mentioned in the section above by implementing the following:

1. The hardware team built a storage shelves as shown below as an improvement to the prototype to allow the storage of the Mojo and speaker.

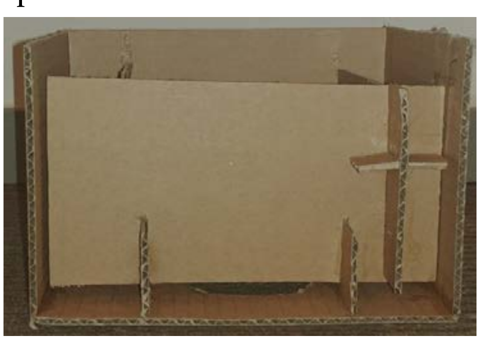


Figure 4.4.1 Top View of Shelves

Figure 4.4.2 Side View of Shelves

2. A backing was place for the keyboard interface.

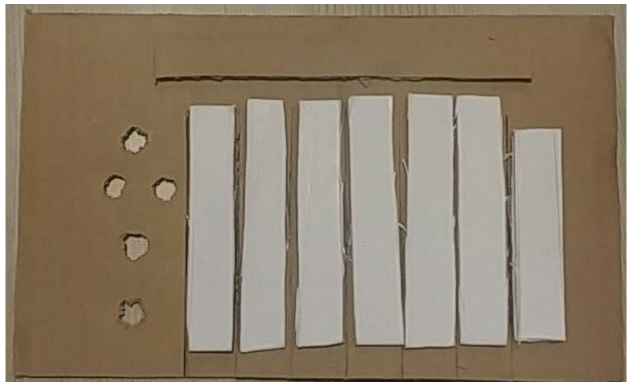


Figure 4.4.3 Backing

3. The speaker was placed inside the housing without any opening, surrounded with cloth to muffle out the loudness.

Figure 4.4.4 New Speaker Location

Figure 4.4.5 Cloth muffler

4. The LED matrix was de-soldered from the board and soldered on the strip board instead. Pin headers were placed for easy replacement of the LED matrix and connectors were used so as to secure the wiring to the MOJO

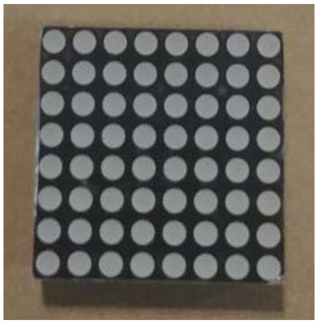


Figure 4.4.6 Front of LED matrix (de-soldered)

Figure 4.4.7 Back of LED matrix (de-soldered)

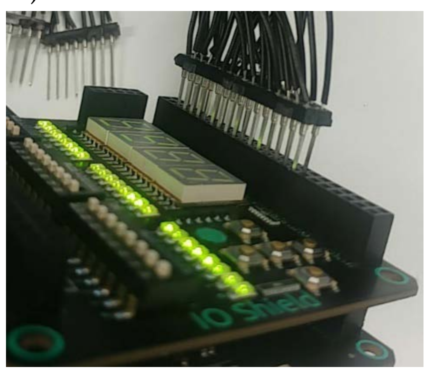


Figure 4.4.8 Connector pins

5. Components Budget

Hardware	Category	Amount	Cost (SGD/each)	Total (SGD)
Push Buttons	Housing [Reset Button]	1	$0 Taken from FabLab/DSLab	$0*13
	Keyboard Interface [Arrows, keyboard and OK button]	12		
7-Segment Display	Screen Interface [Score]	5	$0 Taken from FabLab/DSLab	$0*5
Cardboard	Housing and Piano Keys	3	$0 Recycled from Angelia stockpile	$0*3
LEDs	Screen Interface [Speed and Song Selection]	8	$0 Taken from FabLab/DSLab	$0*8
Circuitry Components such as resistors, soldering iron etc.	Circuitry	∞ depend on circuit	$0 Taken from FabLab/DSLab	$0
LED Matrix	Screen Interface	2	$0 Taken from DSLab	$0*2
Speaker	Housing	1	$8.50	$8.50
Stereo Headphone Jack		1	$1.50	$1.50
Mojo	Mojo	1	$0	$0
			Total Cost (SGD)	$10

6. Summary

In the end, what makes our game stand out is our persistence and the lack of desire to conform. While other groups were laser cutting materials for large, fancy housings, we made ours out of full cardboard such that it is as efficient and robust as can be. Late nights were well spent on our heads-turning Mario theme song and "sounds like birds chirping" (quote Prof Yuen Chau) Happy Birthday tune. Thanks go out to DS-lab, E-lab and other teams who graced us with their electrical components and advice on Mojo.

We have enjoyed making our game by slowly building up on what we have learnt from both theory and practical lessons over the weeks. Constant revision, testing, rushing for check-offs and reflections on log book have allowed us and many others to play piano bit-by-bit.

References

Software implementation of the speaker understanding:

- http://www.fpga4fun.com/MusicBox3.html

Software implementation of the matrix understanding:

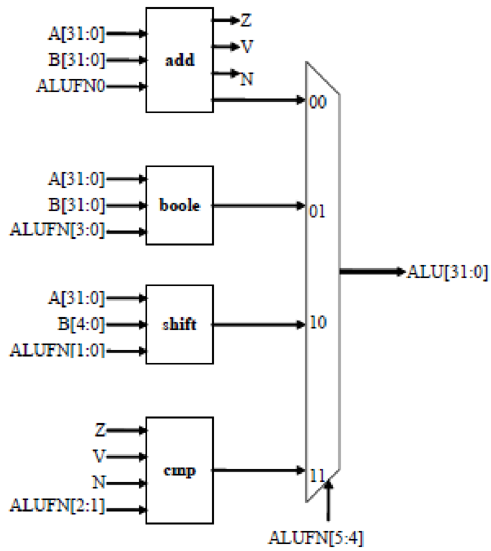
- https://embeddedmicro.com/tutorials/lucid/io-shield

Mojo Tutorials:

- https://embeddedmicro.com/tutorials/mojo

Appendix

ALU DESIGN AND TEST



The design for the 32-bit ALU is as shown above, thus, the design for an 8-bit ALU is the same, with the only difference being an 8-bit output. The corresponding ALUFN[5:0] will correspond to the following operations as shown in the table below.

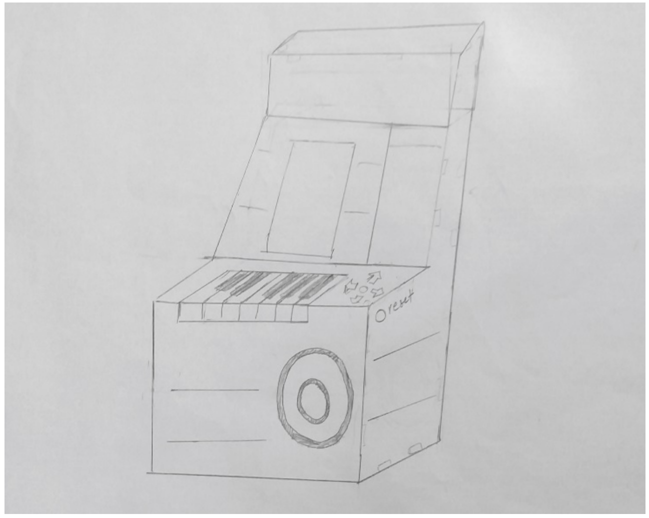
ALUFN[3:0]	Operation	Code	Hex
0000	-	-	4h0
0001	NOR	~(A|B)	4h1
0010	-	-	4h2
0011	-	-	4h3
0100	-	-	4h4
0101	-	-	4h5
0110	XOR	A^B	4h6
0111	NAND	~(A&B)	4h7
1000	AND	A&B	4h8
1001	XNOR	~(A^B)	4h9
1010	A	A	4hA
1011	-	-	4hB
1100	B	B	4hC
1101	-	-	4hD
1110	OR	A|B	4hE
1111	-	-	4hF

Truth Table		B	A	Output
ALUFN[0]	NOR	0	0	1
ALUFN[1]		0	1	0
ALUFN[2]		1	0	0
ALUFN[3]		1	1	0
ALUFN[0]	OR	0	0	0
ALUFN[1]		0	1	1
ALUFN[2]		1	0	1
ALUFN[3]		1	1	1
ALUFN[0]	B	0	0	0
ALUFN[1]		0	1	0
ALUFN[2]		1	0	1
ALUFN[3]		1	1	1
ALUFN[0]	NAND	0	0	1
ALUFN[1]		0	1	1
ALUFN[2]		1	0	1
ALUFN[3]		1	1	0
ALUFN[0]	AND	0	0	0
ALUFN[1]		0	1	0
ALUFN[2]		1	0	0
ALUFN[3]		1	1	1
ALUFN[0]	A	0	0	0
ALUFN[1]		0	1	1
ALUFN[2]		1	0	0
ALUFN[3]		1	1	1
ALUFN[0]	XNOR	0	0	1
ALUFN[1]		0	1	0
ALUFN[2]		1	0	0
ALUFN[3]		1	1	1
ALUFN[0]	XOR	0	0	0
ALUFN[1]		0	1	1
ALUFN[2]		1	0	1
ALUFN[3]		1	1	0

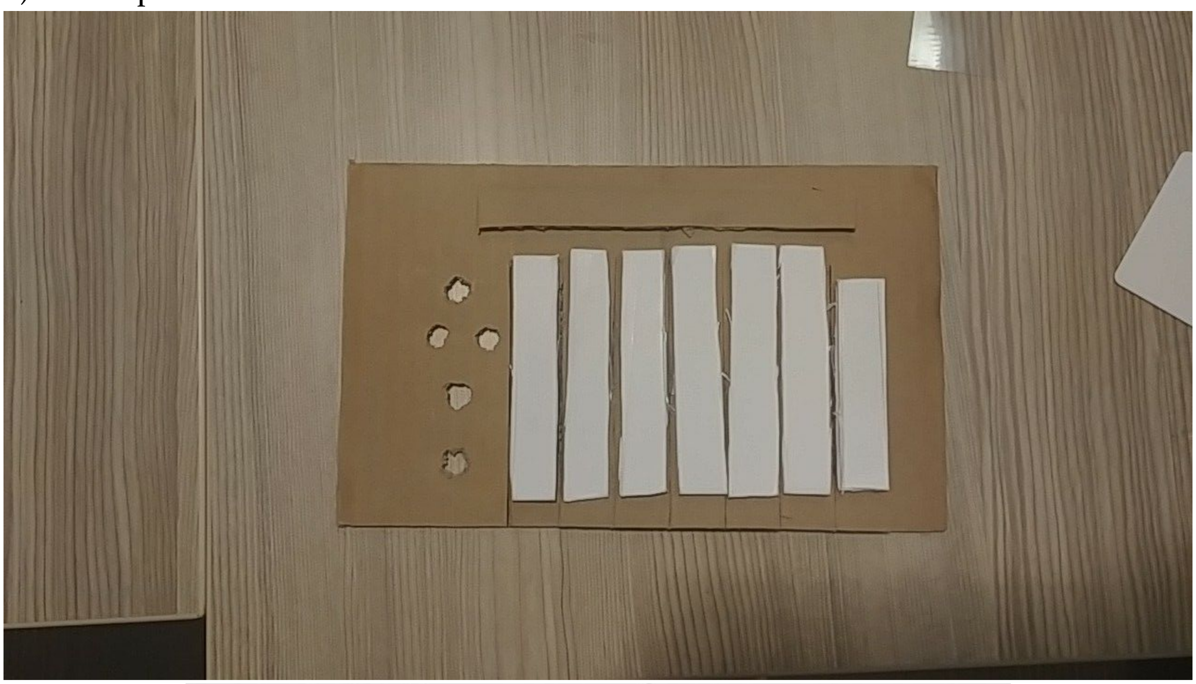
The ALU is tested by building a testing circuit on MOJO (checkoff 1) to feed it edge cases and some general cases. The output is the compared with the expected output and any mismatch will be indicative of a faulty ALU. However, the ALU works perfectly with all the test cases for the operations mentioned, thus deeming its reliability.

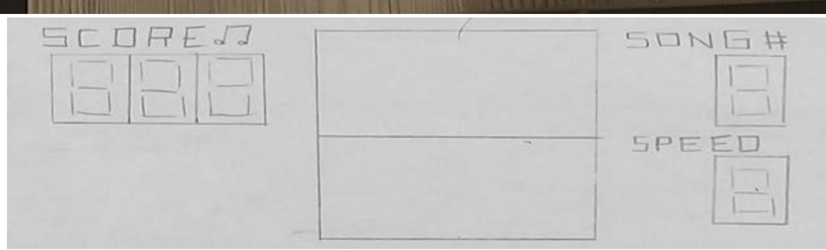
PROTOTYPE SCHEMATICS

1) Console Interface

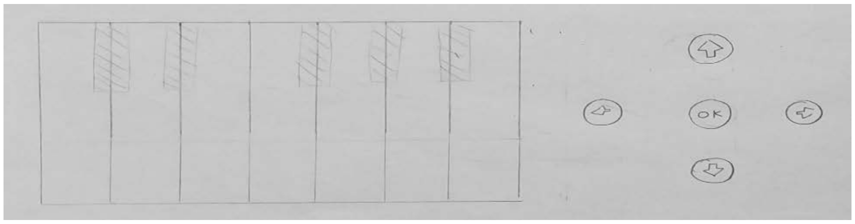


2) Top view of Screen Interface

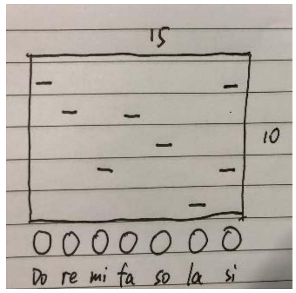




3) Top view of Keyboard Interface



4) LED Screen Interface



- Dash denotes a falling music "note"
- When it reaches the last row, like for the note la, player needs to press the corresponding key

PROJECT MANAGEMENT LOG: TEAM TASKS

Task	Person IC	Due Date	Status
Design & Implementation of ALU	All	20th October 2017 (Checkoff 1)	Completed
Game Ideation & Draft Design	All	10th November 2017 (Checkoff 2)	Completed
Game Prototype: Software Implementation	Guan Lun & Xingxuan	8th December 2017 (Checkoff 3)	Completed
Game Prototype: Hardware Implementation	Angelia & Kim	8th December 2017 (Checkoff 3)	Completed
Project Poster	All	4th December 2017	Completed
Final Debugging & Troubleshooting	All	12th December 2017	Completed
Project Exhibition & Report	All	13th December 2017 (Checkoff 4,5)	Completed

COMPONENT SPECIFICATIONS

- Speaker Specifications

Speaker	3W(1KHz,THD10%), 36mm, 4 Ω
Frequency Response	20Hz～20KHz
S/N ratio	≥85dBA
Battery	Built-in 180mA li-on battery , DC3.7V
Power Supply	DC 5v
Playing Time	About 1~2 hours
Product Size	D52 x H38mm
Material	ABS material, with rubber oil plated

(Taken from: https://cdn.sparkfun.com/datasheets/Components/General/hamburgerspeaker.pdf)

- Stereo Headphone Adapter Specifications
  - Stereo jack
  - Central ground pin and disconnecting switches at the left and right channels
    (2.54 mm spacing between pins)
  - 3.5mm stereo cable

  (Taken from: https://www.sgbotic.com/index.php?dispatch=products.view&product_id=2432)

- LED Matrix Specifications
  - 16x16 dot red LED display
  - Column control 74HC595
  - Line control 74HC138
  - Cascadable

  (Taken from: https://www.elabpeers.com/led-matrix-display.html)