1D Project Group Report

Electronic Game Prototype

Team Number: 03-2

James Raphael Tiovalen, Ng Yu Yan, Sun Kairan, Cheow Wei Da Nicholas, Jodi Tan Kai Yu

---

Table of Contents

Introduction

“No, let’s play DigiSticks!”

DigiSticks is a fresh, digital variant of the old-school game we all know and love, Chopsticks. In this game, you can not only attack, you can not only split, but you can have all-new gameplay with our freshly introduced powerups! Choose from 9 Boolean Bitwise Operations to effectively beat your opponent in a few smart, swift moves!

The following report explains the process behind the design and building of our game: DigiSticks. Who knows, perhaps you can build your own DigiSticks!

Design

Description of the Game

Our fun game is a variant of the two-player Chopsticks hand game. Chopsticks is a hand game for two or more players, in which players extend a selected number of fingers from each hand and transfer those scores by taking turns to tap one hand against another. Chopsticks is an example of a combinatorial game and is solved in the sense that with perfect play, an optimal strategy from any point is known.

For our project, we digitized this hand game and initiated some reworking and remodeling of the whole design process. We utilize the variant with remainders and transfers, with an additional extension of including Boolean operations into the game (instead of just additions and modulos).

Points in our game represent fingers in the actual game. This is a base 5 game, therefore there is a roll-over amount of 5 for each hand (when the hand reaches 5, goes back to 0 points), this is represented by only 4 points/fingers in our game.

Design Inspirations

Since our game required buttons, our design ideas are mainly inspired by classic arcade games such as Street Fighter that have buttons as controls.

Figure 1: Classic Arcade Game Controls

Furthermore, since our inputs (buttons) and outputs (7-segments and LEDs) are simple, we opted for a simple layout that would display all the needed components clearly.

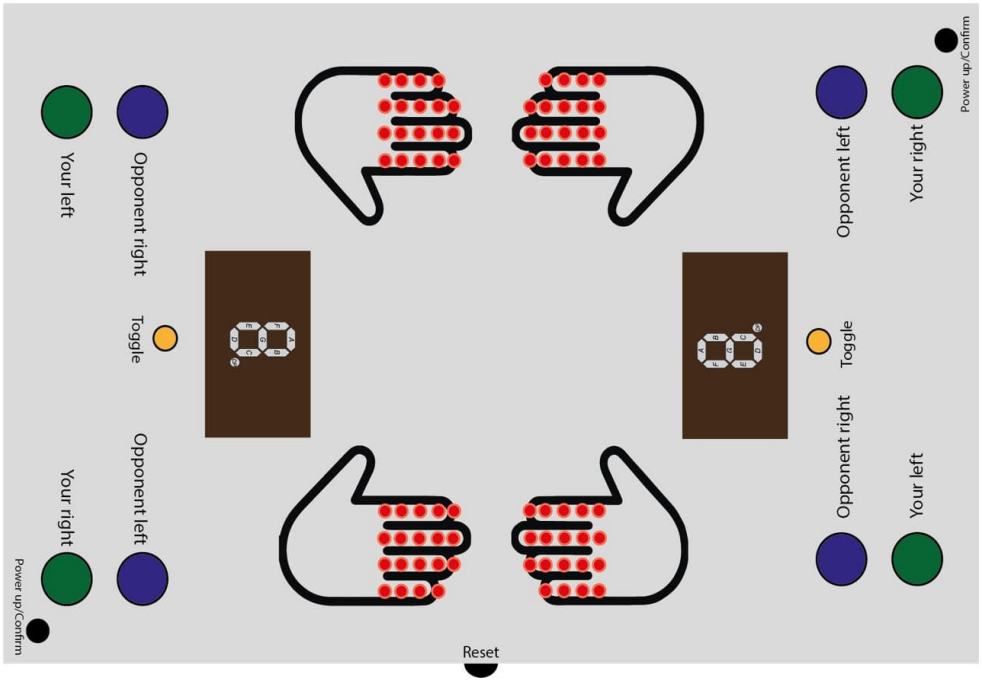

Figure 2: Initial Layout Design

Since our acrylic sheets were transparent (box) and translucent (top plates), we chose to print our custom designs and stick them onto the acrylic to beautify the game as well as cover up the circuitry.

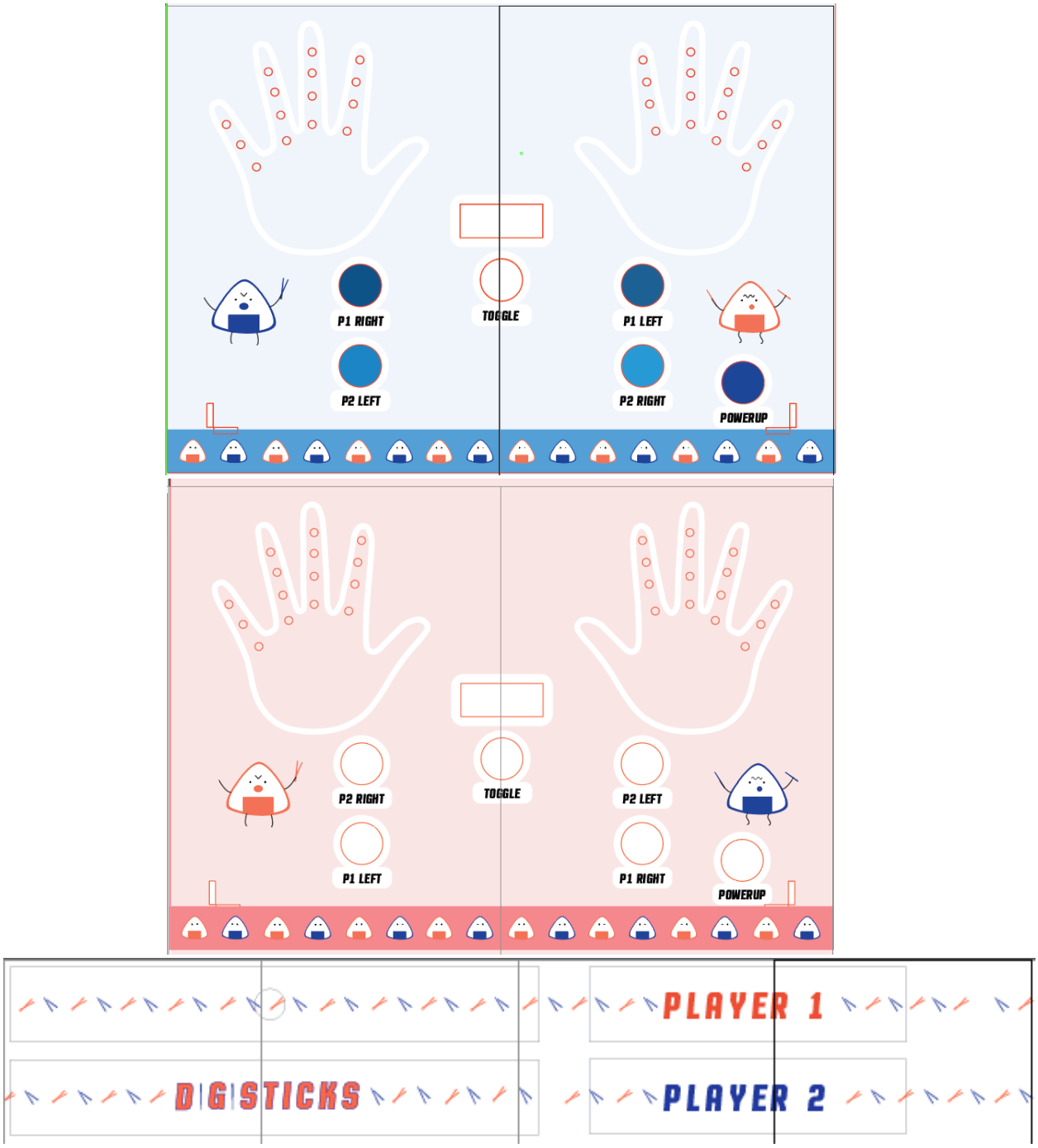

Figure 3: Final Sticker Designs for Top Plates and Sides of Box

Figure 4: Game with Side Stickers On

Test Scenarios

Scenario	Action	Result
P1 Left -> P1 Right Press Toggle to determine the number of points (1,2,3) to transfer	P1 Left Split (P1 L, P1 R)	(4,4) press 1 -> (3,0) (4,4) press 2 -> (2,1) (4,4) press 3 -> (1,2)
P2 Left -> P2 Right Press Toggle to determine the number of points (1,2,3) to transfer	P2 Left Split (P2 L, P2 R)	(4,3) press 1 -> (3,4) (4,3) press 2 -> (2,0) (4,3) press 3 -> (1,1) (4,2) press 1 -> (3,3)

		(4,2) press 2 -> (2,4) (4,2) press 3 -> (1,0) (4,1) press 1 -> (3,2) (4,1) press 2 -> (2,3) (4,1) press 3 -> (1,4) (4,0) press 1 -> (3,1) (4,0) press 2 -> (2,2) (4,0) press 3 -> (1,3) (3,4) press 1 -> (2,0) (3,4) press 2 -> (1,1) (3,4) press 3 -> (0,2) (3,3) press 1 -> (2,4) (3,3) press 2 -> (1,0) (3,3) press 3 -> (0,1) (3,2) press 1 -> (2,3) (3,2) press 2 -> (1,4) (3,2) press 3 -> (0,0) (3,1) press 1 -> (2,2) (3,1) press 2 -> (1,3) (3,1) press 3 -> (0,4) (3,0) press 1 -> (2,1) (3,0) press 2 -> (1,2) (3,0) press 3 -> (0,3) (2,4) press 1 -> (1,0) (2,4) press 2 -> (0,1) (2,4) CANNOT press 3 (2,3) press 1 -> (1,4) (2,3) press 2 -> (0,0) (2,3) CANNOT press 3 (2,2) press 1 -> (1,3) (2,2) press 2 -> (0,4) (2,2) CANNOT press 3

		(2,1) press 1 -> (1,2) (2,1) press 2 -> (0,3) (2,1) CANNOT press 3 (2,0) press 1 -> (1,1) (2,0) press 2 -> (0,2) (2,0) CANNOT press 3 (1,4) press 1 -> (0,0) (1,4) CANNOT press 2 (1,4) CANNOT press 3 (1,3) press 1 -> (0,4) (1,3) CANNOT press 2 (1,3) CANNOT press 3 (1,2) press 1 -> (0,3) (1,2) CANNOT press 2 (1,2) CANNOT press 3 (1,1) press 1 -> (0,2) (1,1) CANNOT press 2 (1,1) CANNOT press 3 (1,0) press 1 -> (0,1) (1,0) CANNOT press 2 (1,0) CANNOT press 3 (0,4), (0,3), (0,2), (0,1) CANNOT press anything
P1 Right -> P1 Left Press Toggle to determine the number of points (1,2,3) to transfer	P1 Right Split (P1 L, P1 R)	(4,4) press 1 -> (0,3) (4,4) press 2 -> (1,2) (4,4) press 3 -> (2,1)
P2 Right -> P2 Left Press Toggle to determine the number of points (1,2,3) to transfer	P2 Right Split (P2 L, P2 R)	(4,3) press 1 -> (0,2) (4,3) press 2 -> (1,1) (4,3) press 3 -> (2,0) (4,2) press 1 -> (0,1) (4,2) press 2 -> (1,0) (4,2) CANNOT press 3

		(4,1) press 1 -> (0,0) (4,1) CANNOT press 2 (4,1) CANNOT press 3 (3,4) press 1 -> (4,3) (3,4) press 2 -> (0,2) (3,4) press 3 -> (1,1) (3,3) press 1 -> (2,4) (3,3) press 2 -> (1,0) (3,3) press 3 -> (0,1) (3,2) press 1 -> (4,1) (3,2) press 2 -> (0,0) (3,2) CANNOT press 3 (3,1) press 1 -> (4,0) (3,1) CANNOT press 2 (3,1) CANNOT press 3 (2,4) press 1 -> (3,3) (2,4) press 2 -> (4,2) (2,4) press 3 -> (0,1) (2,3) press 1 -> (3,2) (2,3) press 2 -> (4,1) (2,3) press 3 -> (0,0) (2,2) press 1 -> (3,1) (2,2) press 2 -> (4,0) (2,2) CANNOT press 3 (2,1) press 1 -> (3,0) (2,1) CANNOT press 2 (2,1) CANNOT press 3 (1,4) press 1 -> (2,3) (1,4) press 2 -> (3,2) (1,4) press 3 -> (4,1) (1,3) press 1 -> (2,2) (1,3) press 2 -> (3,1) (1,3) press 3 -> (4,0)

		(1,2) press 1 -> (2,1) (1,2) press 2 -> (3,0) (1,2) CANNOT press 3 (1,1) press 1 -> (2,0) (1,1) CANNOT press 2 (1,1) CANNOT press 3 (1,0) press 1 -> (0,1) (1,0) CANNOT press 2 (1,0) CANNOT press 3 (4,0), (3,0), (2,0), (1,0) CANNOT press anything
P1 Left -> P2 Right	P1 Attack P2 (P1 L, P1 R)	E.g.: P1= (4,4) -> (3,4)
P1 Right -> P2 Left		E.g.: P1= (4,4) -> (4,3)
P2 Left -> P1 Right	P2 Attack P1 (P2 L, P2 R)	
P2 Right -> P1 Left		
P1 Left -> P2 Right Press Powerup and Toggle button to determine the function (AND, NAND, OR, NOR, XOR, XNOR, X, NOT ‘X’, NOT ‘Y’) to use	P1 “Powerup” Attack P2 (P1 L, P2 R)	E.g.: 1 AND 2 -> 1,0 2 NAND 4 -> 2,2 3 OR 2 -> 3,1 4 NOR 2 -> 4,1 3 XOR 4 -> 3,2 2 XNOR 3 -> 2,2 2 X 3 -> 2,2 2 NOT ’X’ 3 -> 2,1 3 NOT ‘Y’ 1 -> 3,1
P1 Right -> P2 Left Press Powerup and Toggle button to determine the function (AND, NAND, OR, NOR, XOR, XNOR, X, NOT ‘X’, NOT ‘Y’) to use	P1 “Powerup” Attack P2 (P1 R, P2 L)	
P2 Left -> P1 Right Press Powerup and Toggle button to determine the	P2 “Powerup” Attack P1 (P2 L, P1 R)	

function (AND, NAND, OR, NOR, XOR, XNOR, X, NOT ‘X’, NOT ‘Y’) to use		
P2 Right -> P1 Left Press Powerup and Toggle button to determine the function (AND, NAND, OR, NOR, XOR, XNOR, X, NOT ‘X’, NOT ‘Y’) to use	P2 “Powerup” Attack P1 (P2 R, P1 L)	
P1 Left = 0 & P1 Right = 0	P2 Wins (P2 L, P2 R)	(0,0), End game, Press restart button to restart game
P2 Left = 0 & P2 Right = 0	P1 Wins (P1 L, P1 R)	(0,0), End game, Press restart button to restart game

To ensure operability and functionality of the Alchitry Br’s pins, as well as continuity between buttons and LEDs to the FPGA after the whole prototype is assembled, we have crafted testing modules which are available on our GitHub repository under the `tests` folder.

User Manual

The rules of our DigiSticks game are as such:

1. Each player will start with two hands, each with one point, with all LEDs off.
2. The LEDs on each hand will indicate how many fingers are ‘out’. If the fingers are not ‘out’, their LEDs will be turned off. If the fingers are ‘out’, the corresponding number of LEDs will be turned on.
3. During each turn, each player can attack or split.
4. Hand that reaches 5 points is killed and goes back to 0 points. The player can split the remaining points on his other hand, reviving the killed hand on his turn thereafter.
5. **Attack**: Each hand can add to another hand (either one of the opponent's hand or either one of their own hand). Players can choose a powerup choice between 1 to 9, and implement it, replacing the addition and modulo operations for that turn. This can be done at most 3 times per player during the gameplay.
6. **Split:** Redistributes points between **the own player’s hand**.
7. The first player with both of their hands killed (zero value) loses, while the remaining player wins
8. To restart the game, press the reset button on the side of the hardware.

Below is a representation of what a player can do during his turn (either 1 or 2 or 3):

1. ATTACK, where an addition combined with modulo is applied. If the player wants to use his right hand to perform an attack on the opponent's right hand (an addition), he will press the “P1 Right” button followed by the “P2 Right” button.

2. “POWERUP” ATTACK, where a function among AND, NAND, OR, NOR, XOR, XNOR, X, NOT ‘X’, NOT ‘Y’ is chosen. If a player wants to use a powerup to attack from his right hand to his opponent’s right hand, he should press the “powerup” button at the start of his turn. He then chooses the powerup he wants using the “Toggle” button and confirms it using the “powerup” button. Afterwards, press on the “P1 Right” button followed by the “P2 Right” button.

3. SPLIT, where a number to transfer, between 1,2,3 is chosen. This number is checked to be lesser than the number of points the players hand (to transfer from), before the player can choose a number. If a player wants to split from his left hand to his right hand, he will press the “P1 Left” button followed by “P1 Right” button to transfer points from his right hand to his left hand. He then chooses the number of points to transfer using the “Toggle” button and confirms using the “Powerup” button.

Game Controls & Button Colour:

- (P1) Your right/left buttons: red/blue
- (P2) Opponent right/left buttons: blue/red
- Powerup button: green
- Toggle button: yellow
- Reset button: red (on the side of the hardware prototype)

Steps in Building the Prototype

Hardware

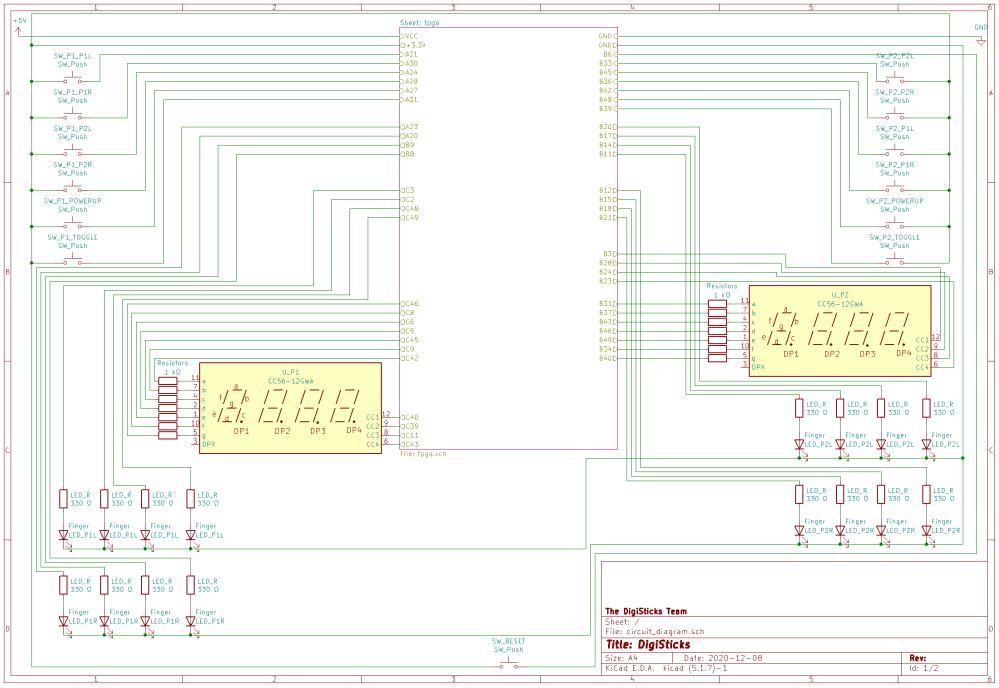

Figure 5: Circuit diagram schematics for DigiSticks.

1. Producing the Top and Base Plates
    a. Prepare Illustrator files with accurate sizings for components (buttons, LEDs and 7-segments)
    b. Laser cut the acrylic, test to ensure correctly sized holes are cut into acrylic to fit components
    c. Bend the cut acrylic to make bottom box
2. Insert Components into Holes
    a. Insert buttons and secure them into plate with screw mechanism provided
    b. Insert LEDs and hot-glue them to ensure they stay in place. Ensure all positive legs point to same side for convenient soldering later on
    c. Insert 7-segments and hot-glue them as well
3. Preparing the Circuit Board
    a. Dedicate one side of the circuit board for components that need to connect to GND (LEDs), another side for components that need to connect to VDD (buttons) and a section in the centre for the 7-segments
    b. For the GND side, solder a bare wire across approximately ⅓ of the board and a jumper wire to plug into the FPGA's GND pin. Repeat for

the VDD side, leaving approximately ⅓ of the board empty for 7-segments circuitry

4. Connecting the LEDs
    a. Solder two wires (A&B) from each LED to circuit board
    b. An LED in the schematics is replaceable by a set of multiple LEDs in parallel to better represent a finger, if the reader so desires. Connect additional resistors as appropriate
    c. Since each finger should be controlled by one output from the FPGA (all LEDs turned on or off at the same time), solder all LEDs' (that constitute a finger) wire A to a single male jumper wire that will power the LEDs using the I/O pins on the FPGA
    d. Solder the LEDs' wire B to the GND wire that connects to the FPGA "G" pin to complete the circuit
5. Connecting the Buttons
    a. Solder two wires(A&B) to each button
    b. Solder wire A to the circuit board and connect it to a male jumper wire (soldered onto circuit board) to provide input to FPGA
    c. Solder wire B to circuit board and connect it to VDD that comes from FPGA "+3.3V" pins to complete the circuit
6. Connecting the 7-segments
    a. Connect female jumper wires to pins 12, 9, 8 and 6 and solder the other end of the wires to the circuit board. These are to control the digits, grounding them will activate their corresponding digit
    b. Connect female jumper wires to the rest of the pins on the 7-segments, except 3, which controls the dot, which we do not need. Solder the other end of the wires to the circuit board. Grounding the pins will deactivate their corresponding segments
    c. Solder all wires from the 7-segments on the circuit board to resistors, then to male jumper wires that will connect each wire to the FPGA. Each 7-segments pin should be connected to one FPGA I/O pin
7. Connecting to FPGA
    a. Insert all jumper wires from both sides to banks on the FPGA, taking note of which pins correspond to which components for use in the code later
    b. Close up the acrylic box

Software

The software can be divided into several sections: Arithmetic Logic Unit (ALU), Control Unit (CU), Register File (REGFILE), Input/Output (I/O), and testing.

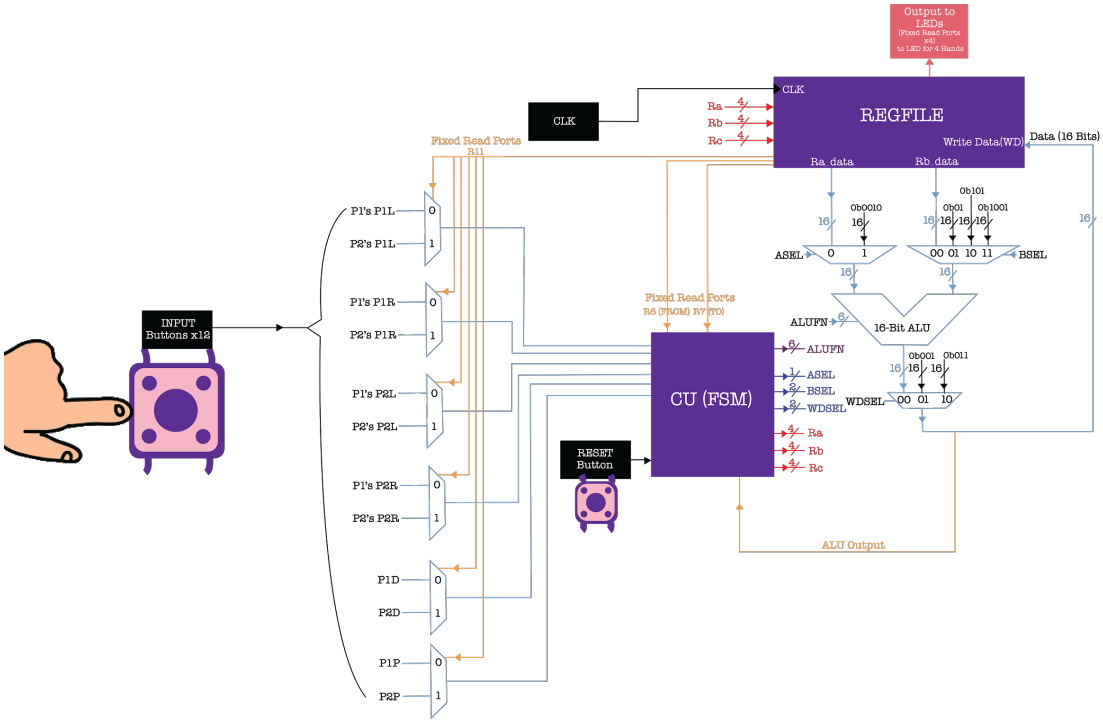

Figure 6: DigiSticks' overall datapath diagram.

1. ALU

We used the 16-bit ALU which was built for 1D part 2 checkoff 1. We implemented around 40 operations which are able to be selected by 6 bits ALUFN. In our game, we made use of ADD, SUB, MOD, CMPEQ, CMPLT, CMPLE, NOT 'Y', AND, NAND, OR, NOR, XOR, XNOR, 'X', NOT 'X', selecting by the assigned 6 bits ALUFN sent from the game control unit.

2. CU

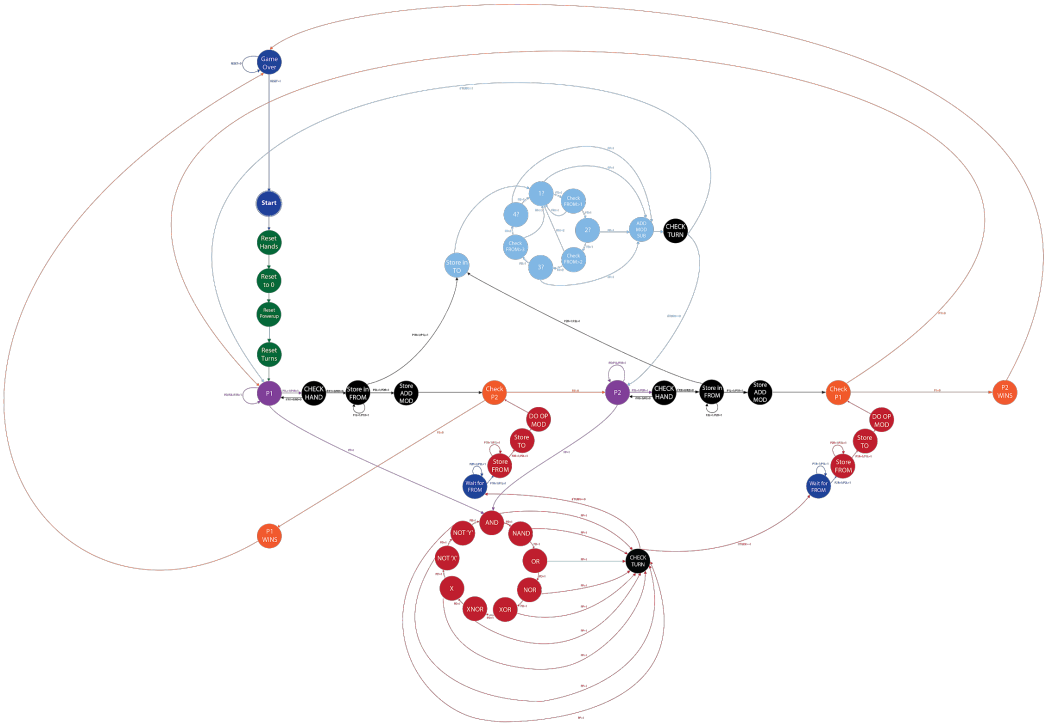

Figure 7: The FSM diagram representation of the state machine implemented by our game's Control Unit.

Our CU is made of a finite state machine (FSM) with 30 states to control the flow of our game. The CU will receive inputs from button press, fixed read ports of REGFILE and 16 bits ALU output. Most state transitions will depend on both current state and input (for example, when checking win conditions, we need to check if the two hands of the attacked player are zero. If it is not true, the game will update the turn and continue. Otherwise, the game will end.) While some state transitions only depend on current state, like doing a set of operations (for example: do ADD and MOD in order). Additionally, signals to turn on the 7-segments will be sent by CU as well.

3. REGFILE

The register file contains all the necessary data which needs to be remembered including player 1/player 2 hands, number for splitting, winner conditions etc. It consists of 16 16-bit registers (4-bit addressable) with 2 addressable combinational read ports, 9 fixed combinational read ports and 1 sequential write port. The table below shows usage of our registers in detail.

Register	Usage	Register	Usage
R0	P1 Right Hand	R8	LITERAL
R1	P1 Left Hand	R9	POWERUP
R2	P2 Right Hand	R10	WINNER
R3	P2 Left Hand	R11	TURN
R4	P1 Remaining Powerups	R12	TEMP
R5	P2 Remaining Powerups	R13	TEMP
R6	FROM	R14	TEMP
R7	TO	R15	0

Some remarks for the table:

a. P1: Player 1
b. P2: Player 2
c. LITERAL: number for splitting
d. FROM and TO: stores the addresses of the two registers (hands) to do operations on
e. WINNER: 0 (NONE), 1 (P1 WIN), 2 (P2 WIN)
f. TURN: 0 (P1), 1 (P2)
g. R15: Always ZERO
h. Nine fixed combinational read ports:
    - To external outputs (LEDs): R0, R1, R2, R3
    - To internal components: R6, R7, R8, R9, R11

4. I/O

All inputs of our game come from buttons. We have 13 buttons in total, one reset button and two sets of 6 buttons for two players. Button conditioner is used to detect the press for every button. Additionally, a 12-to-6 mux is used to cut down the number of buttons we need to take care since only 6 buttons will be used by one player each turn. The selector signal of the mux is obtained from one of the fixed read ports of REGFILE (R11, TURN).

Output contains 16 sets of LEDs (4 sets for one hand, four hands in total for two players) and 2 7-segments (one for each player). The output to LEDs is

from fixed read ports of REGFILE. A 4x4 matrix is used to analyze the data from the read ports and turn on certain sets of LEDs. The signal to turn on the 7-segments is sent by CU while the choice of what to display is from REGFILE.

5. Tester

Three tests are created to test our game.

a. BrPinTest: Test the operability of each SingleEndedIO pin of the custom Alchitry Br board. Simply set the output of all pins to 1 and check if they are able to make an LED light up one by one. A reset conditioner is used to synchronize the reset signal to the FPGA clock to ensure the entire FPGA comes out of reset at the same time.
b. ButtonTest: Test the functionality of a set of buttons and an LED (button is connected to VDD and B30, while an LED is connected to B21, resistor and GND). This test is to help us check how we can receive input from the button we bought and what is a proper way to code in Lucid. A reset conditioner is also used in the same way as in BrPinTest.
c. ButtonPressTest: Test the functionality of the complete circuitry to debug any potential/possible connection problems due to poor soldering/wiring. In this test, if the button of a hand is pressed, the set of LEDs representing the pressed hand will all light up. If the powerup button is pressed, two hands at the same side will light up. To test the 7-segments, we simply set numbers in Lucid which can be shown by the 7-segments when the assigned button is pressed. Similarly, the way we test the reset button is also through setting some obvious output which can be seen once the button is pressed. The output we predefined in this test is to turn on all LEDs and two 7-segments show 8888 when the reset button is pressed.

For future work, more tests could be implemented to further inspect and probe the correctness of the game's sequential logic.

Design Issues & Problems Solved

1. Acrylic Case
    a. Portability

        For portability, we first considered a foldable suitcase design as shown in Figure 8, but realised quickly that this would complicate our design more than necessary, thus we dismissed the idea of a foldable structure and stuck with a static acrylic case. Furthermore, the game would not be

of a large size, thus it would still be sufficiently portable (while still maintaining a significant amount of distance between the two players for the sake of *“Social Distancing”*).

Figure 8: Foldable Suitcase Design Inspiration

b. Maintenance

For easy maintenance, our top plates had to be removable so as to expose the FPGA and wires should any wires break or pins fall out of the FPGA. Thus, we thought of a hinge idea as shown in Figure 9. However, while designing the top plates, we realised we wanted a dual-colour top plate, which meant that the top plates would be two separate pieces. We also realised that it would be difficult to attach a hinge onto the acrylic pieces due to the thickness and strength of the acrylic and the methods available to do this were only by drilling or acrylic glue. Thus, we did not pursue the hinge method and simply made slots in our top plates and corresponding protrusions on our base plate (used to make the box) to fit into the slots, thus making our top plates removable but secure.

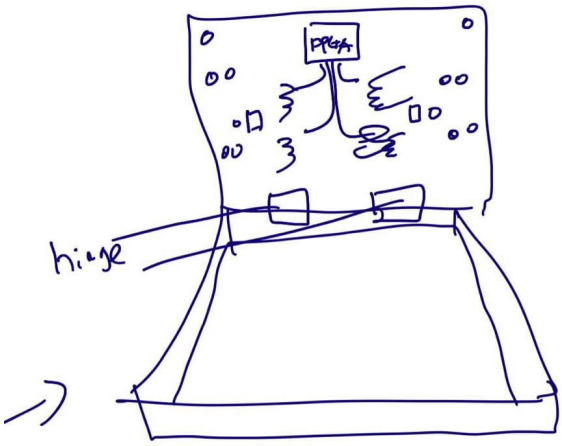

Figure 9: Hinge Design Idea

c. Box

When we first designed our box, we planned to attach the 5 pieces (4 side plates are shown on the right of Figure 11 and the base plate is on the left of Figure 11) together using, possibly, acrylic glue.

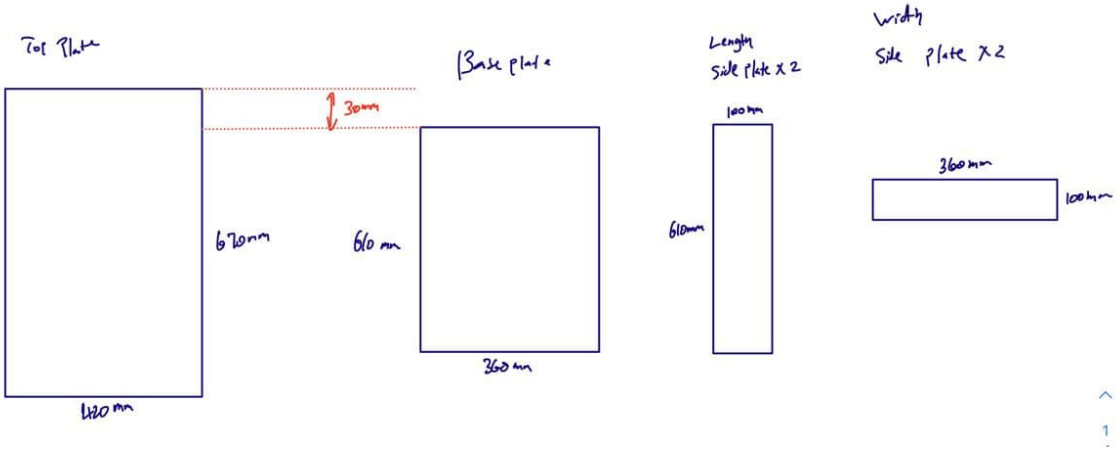

Figure 10: Rough Initial Plan of Top and Base Plates to Cut

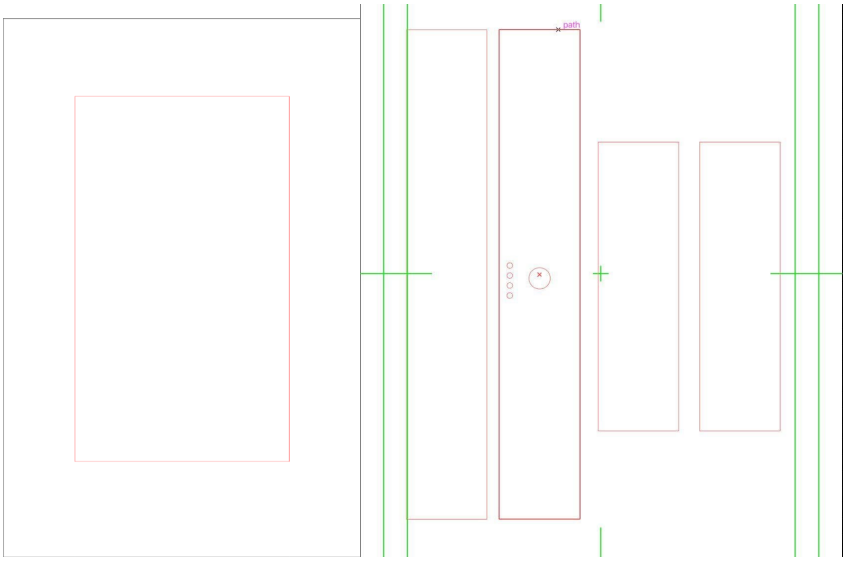

Figure 11: Base and Side Plates Initial Design

However, we realised that it would take a long time to obtain and dry and thus, with the help of the FabLab staff, we redesigned our box such that we would have one big piece and bend the four sides to obtain the box, as shown below.

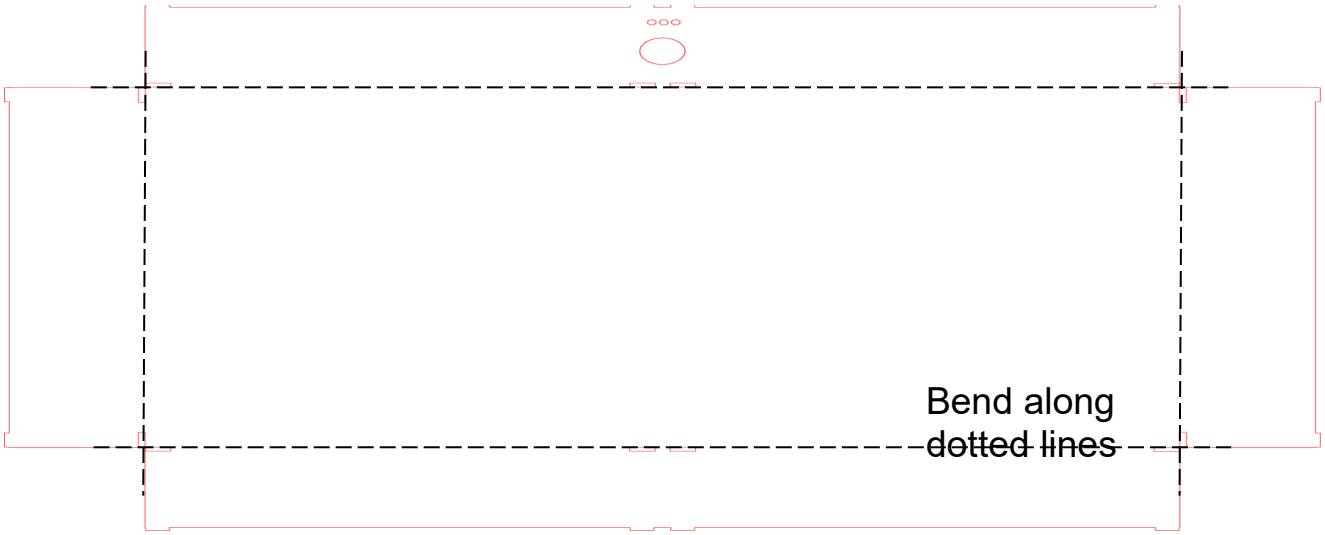

Figure 12: Final Base Plate Design

d. Top Plate Locking Mechanism

After we came up with the initial design with all the holes for the buttons, LEDs and 7-segments, we realised that we wanted each half to be of different colours, one red and another blue. Thus, we had to come up with a way to attach or secure the two halves together as well as to the box. Hence, we decided to cut additional rectangular slots along the border of the top plate so that we can insert protrusions on the base plate into those slots, this would fix the top plates to the base plate. To ensure the top plates did not move apart from each other, we decided to use an L-locking mechanism, shown in the L-shape we used to secure the two top plates together.

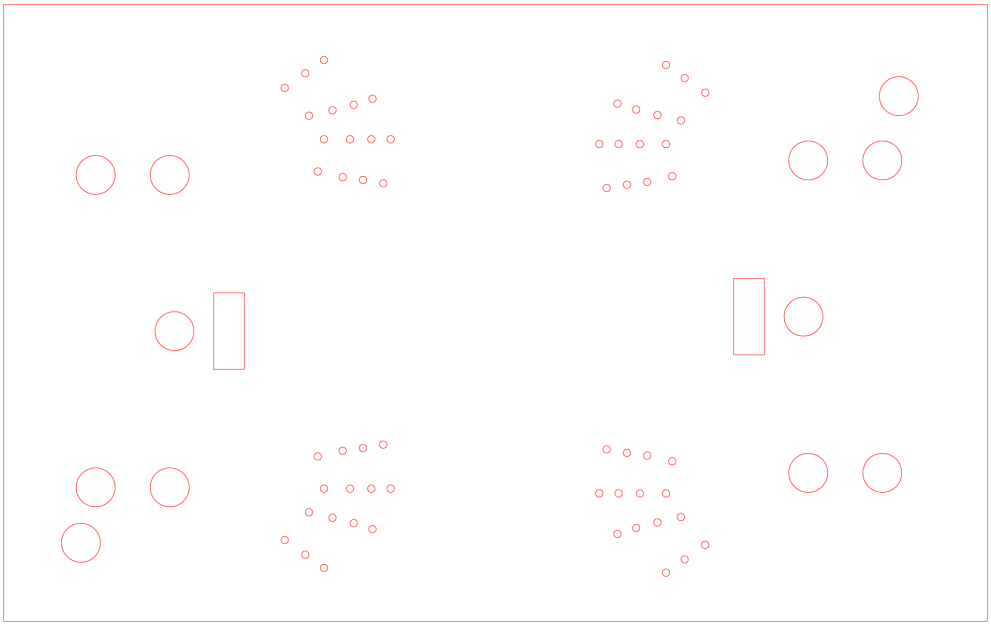

Figure 13: Initial Top Plate Design

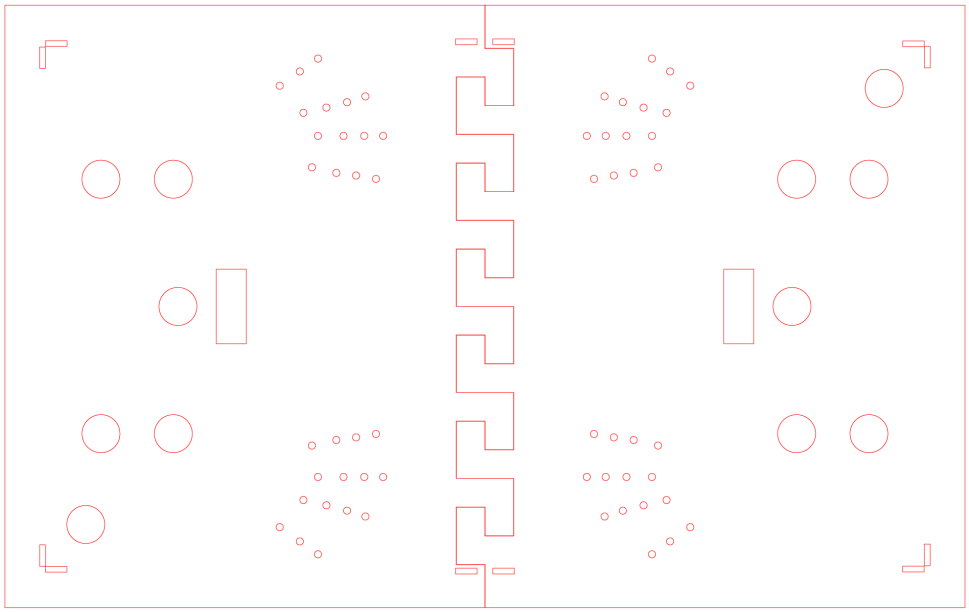

Figure 14: Final Top Plates Design

2. LED Circuit Design
    a. Initial Design

        Initially, the plan was to connect the 3 or 4 LEDs in a row that represented a finger in series and have them connected to a single I/O pin on the FPGA. However, we soon realised that the 3.3V that a single pin provides was only enough to power 1 LED (since 1 LED requires about 2V), thus we thought of using the +5V pin on the FPGA. Since 5V could power 2 LEDs in series, we decided to connect every 2 LEDs in series instead.

    b. Power and Pin Constraints

        After soldering the circuit for 2 LEDs in series and each set of 2 LEDs that constitute a finger in parallel, we realised that the input to each finger circuit would only be 3.3V. Thus, we decided to power only alternate LEDs for each finger. Ideally, we could have put every LED in a finger in parallel with an input to the circuit from the FPGA to light up all the LEDs if we had more time. We also had different LEDs having different brightness as we used LEDs from both the FabLab, DSL and the initial set provided at the start of the project. As we also experienced some

LEDs failing over time, we had no choice but to utilize LEDs of different brightness in our final machine.

3. Cable management and PCB placement

   As we have a total of 11 buttons, and 60 LEDs and two 7-segments displays, we had an extremely significant number of wires that needed to connect our components to the donut board first before connecting to the FPGA. As we were running our LEDs in parallel, we could have cut down on the number of wires needed to connect the donut board by soldering wires directly across all LEDs on the same finger to implement the parallel circuit instead of handling this on the donut board. This would allow our PCB to be smaller and have less wires running up to the case. As we had a very large number of wires, cable management was extremely hard, and we were not able to use any cable sleeves that we bought and there were frequent failures where wires would break or solder joints break.

4. FSM Design

   a. Transition between states

      Even though we spent a lot of time designing and optimizing our finite state transition diagram, when transferring it into Lucid we found more problems regarding the transitions between states. We realized we defined several continuous states which depend on the same external input. This will cause the state either gets stuck there or continues with wrong inputs since the several states need several clock cycles to complete while we only get the correct input in the first clock cycle of these states. To solve this problem, we combined some states together into one state or saved the input in REGFILE if there are quite a few states depending on it. The other problem is to clearly define what is the condition to do state transitions. The transition should either depend on only current state or current state together with input. Last but not least, make sure what is synchronous with the clock cycle and what is asynchronous. We can get output from ALU in the same clock cycle but to get the update data from the fixed read ports of REGFILE, we need to wait till the next clock cycle.

   b. Connection with external inputs and outputs

      We need to make sure how our hardware components work before coding in Lucid, especially for buttons and 7-segments. Besides, we also met problems about how to make our 7-segments turn on only when there is a need to. In order to do this, we added one more variable in our control unit which will be set on 1 only at the states that 7-segments need

to be turned on. They will get the signal from the control unit but the content that they are supposed to display will be sent by REGFILE.

Budget

All prices are shown in SGD.

Components	Purpose	Quantity	Cost of 1 unit
Circular Buttons	Input for hands and reset button.	13	$1.50
Wires	Connect all the components together to the FPGA.	6 bundles (40 pax per bundle)	$5.00 / bundle
Acrylic	To make the base and overall structure of the prototype.	3 (red, blue and translucent)	Free (self-sourced)
7-segments	Display the powerup options and number of fingers for split functionality.	2	$4.00
Hot Glue	Adhesives like hot glue were used to put together unruly pieces to develop our final product.	2 sticks	$0.50
Donut Board	Donut board was used to implement complex circuitry such as resistors, wires, and other components with a liberal and generous amount of soldering.	2	$4.00
LED	Implement user interface of fingers by illuminating fingertips.	64	Free (Fablab)

Resistors	Prevent LEDs and 7-segments from blowing up due to too much power by regulating current and voltage.	16 + 64 = 80	Free (DSL & Fablab)
FPGA	To provide the central main logic for the entire game (receive and send electrical signals).	1	Free (sourced from DSL)
			Total Cost: $66.5

Summary

Chopsticks is a fairly simple game with little rules. However, there is much logic involved, involving many different states. Furthermore, for the purpose of this digital game, we implemented additional mechanisms such as POWERUP and SPLIT so as to present a different take on the traditional Chopsticks. This complicates the logical process as the normal “always-winning” moves do not apply any more. As such, our game is still complex enough in terms of scale and yet is simple enough with rules that are easy to understand, even with the additional twists. We strongly believe that our game has been effective at applying concepts taught during the Computation Structures course as it allowed us to design a computer architecture to support the functionalities of our game. The different BOOLEAN operations available during a POWERUP are also as relevant to this course, as they allow players to exercise their logical and mathematical thinking skills before executing their moves. We hope that this game has inspired the reader to explore more about the field of computer architectures, build their own hardware electronics game or maybe even their own DigiSticks and improve on our design.

Whenever you want to play Chopsticks, always remember to consider playing DigiSticks!

References

- 7-segment schematics referenced from: https://raspi.tv/wp-content/uploads/2015/11/7seg-pinout-annotated_1500.jpg

- Datapath inspired and modified from the official Beta datapath: https://www.dropbox.com/s/7vn4p9ucsydqu9e/beta.png

- Custom Alchitry Br schematics referenced from the provided documentation by SUTD ISTD

- Rules and variants of Chopsticks: https://en.wikipedia.org/wiki/Chopsticks_(hand_game)

Appendix

1. ALU Design and Tests

The ALU was initially designed for our earlier checkoff deadline, and it was further modified with additional control signals (ASEL, BSEL and WDSEL) for the purpose of the game.

Besides our newly created Control Unit and Register File, together with the Arithmetic Logic Unit, we also created a new module called `button_muxes` which serves as the large multiplexer to select corresponding input button signals accordingly depending on the current state. This allows a single ALU to perform multiple operations according to the defined datapath. All of the different components are connected together in one unified datapath prescribed by the hardware description in `au_top.luc`.

For testing, due to the high clock rate of the Alchitry Au, it was feasible to execute a brute force method of testing a multitude of different inputs. Any errors encountered by the Alchitry Au would halt the Alchitry from any further processing and an error message would be displayed.

Since the ALU had passed our initial logic tests for Checkoff 2, it was thus deemed fit for usage with our game.

2. Prototype Schematics

Mechanical case and electronics wiring schematics have been attached to their respective sections of this report. The datapath and FSM for DigiSticks are also available under the Software section of this document.

3. Project Management Log: Team Tasks

Task	Date Completed	Members Involved
Finalise Case Design	17/11/2020	James, Yu Yan & Nicholas
Finalise Components Required	18/11/2020	ALL
Components Procurement at Sim Lim Tower	19/11/2020	ALL
Laser Cut	25/11/2020	ALL
1st Major Session for Component Assembly, Soldering and Programming at FabLab	27/11/2020	ALL
2nd Major Session for Component Assembly, Soldering and Programming in front of LT1	29/11/2020	ALL
3rd Major Session for Component Assembly, Soldering and Programming at DSL	30/11/2020	ALL
4th Major Session for Component Assembly, Soldering and Programming at DSL and in front of LT1	1/12/2020	ALL
Final Hardware Testing	1/12/2020	Yu Yan, Nicholas & Jodi
Final Software Testing	1/12/2020	James & Kairan
Poster Design	1/12/2020	Jodi & Yu Yan
Video Production	1/12/2020	ALL

4. Components' Specifications

Circular Buttons	2.65 cm
Wires	1.5mm single core
Acrylic	3mm thick 92x60cm
4-Digits 7-Segments	5x2cm
LEDs	5mm
Donut Board	6.5x14.5cm
Jumper Wires	28 AWG

5. Ideation

These are some of the older brainstormed game ideas that we came up with during our ideation process.

Idea	About/Goal	Component Needed	Comments
Button Smashing/Bishi-Bashi	Played between 2 students, the one that presses faster wins Row of LEDs imitates a success bar	1 row of LEDs 2 buttons	Too easy in terms of scale. Add additional features: player is required to do something while smashing the button (answer Boolean answers to questions given).
Digital Chopsticks	Variation 1: choose to add or subtract numbers then modulo 5	Input: Buttons 7-segments Output:	Easy to implement, can change some rules to add complexity to the game.

	If implement multiplication or power then modulo, the number will be too big, player cannot win the game If implement shift, for example: shift left then modulo (2 shift left 2 bits become 8, then mod 5 become 3) Suggestion to make game easier, use unsigned numbers (no negative), then mod Variation 2: Randomly choose from pool of instructions Implement by changing rules of game every 15 sec Variation 3: Special skill	LEDs (represent each finger) Other: Multiplexer (to toggle, 4 output mux) 3 pin transistors	Connection of battery, to transistor, to 7-segments and to ground). Registers involved: - Timer register - Score register - Temporary registers to check logic in next cycle States: Do, Store, Check, Update, Idle Basic rules we came up with: - This is a hand game with two players. - Each player begins with one finger raised on each hand which will be represented using LEDs. - On a player's turn, they must choose either attack or split.

Morra (say a number, compare total number of fingers raised = number)	Using comparison functions <=, =, >= Implement game using a scoring system, if player score less than said number must put back hand, else, player can keep hand and play the next round		Too simple.

Factors we considered in choosing the game idea:

- What makes the game original/different from existing games?
- Complexity of the game