Team Name: The Bomb Tamers

**B3S (Brain-Bending Bomb System)**

**Team Number**: 9

*\*Instructions*:

1. ***Directly*** *work on your report in this Google doc. Do not work elsewhere and paste it here.*
2. *Use the template formatting, if it is a heading (e.g., Heading 1, Heading 2 etc.), format it as such so the* ***Table of Contents*** *will reflect it.*
3. *The point of this Google doc is to be a* ***live*** *document that you continually update as you work on your 1D project. The teaching staff will be taking a look at these documents as you work on them to see how your team is progressing.*

*\*Note: This is an offline version of the original document in Google Docs (restricted access)*

**Group Members:**

Lim Jie Han (1006246)

Siddharth Ganesh (1006407)

Yash Yadav (1006199)

Esan Natraj (1006002)

Ryan Pey Jun Hao (1006048)

Sarang Nambiar (1006181)

# **Table of Contents**

[**Table of Contents**](#_Toc133671906)

[**Introduction**](#_Toc133671907)

[**Game Design**](#_Toc133671909)

[Description of the Game](#_Toc133671910)

[User Manual](#_Toc133671911)

[Design Inspirations](#_Toc133671912)

[Test Scenarios](#_Toc133671913)

[**Electronic Design**](#_Toc133671914)

[Input/Output](#_Toc133671915)

[Datapath](#_Toc133671916)

[FSM](#_Toc133671917)

[**Budget**](#_Toc133671918)

[**Summary**](#_Toc133671919)

[**References**](#_Toc133671920)

[**Appendix**](#_Toc133671921)

[1. ALU Design and Tests](#_Toc133671922)

[2. Prototype code + Repo link](#_Toc133671923)

[3. Project Management Log: Team Tasks](#_Toc133671924)

[4. Components’ Specifications](#_Toc133671925)

# 

# **Introduction**

<Describe your idea and prototype briefly, can include a photo of the final prototype here>

The initial idea was to create a game that was exciting, but also required players to react quickly and think strategically. To achieve this, we have come up with our “bomb defusal” game, where players must work against the clock in order to defuse our (obviously fake) bomb.

To defuse the bomb, the player must solve **TWO** mini games that test different skills: your problem solving and reflexes. The wire game forces players to make the right connections in the shortest amount of time possible while the reaction time game requires players to respond quickly to visual stimuli.

Solve both mini games to successfully defuse the bomb. Making a mistake in any of the mini games once will result in a set amount of time being deducted from the countdown. Once the countdown hits 0 before you can complete the mini-games, and the bomb explodes (RIP SUTD).

# 

# 

# 

**Wiring game Reaction Time game**

# **Game Design**

## Description of the Game

<Describe your game and its design>

As mentioned earlier, our game will consist of 2 separate mini-games within the physical prototype. Players will need to finish each of these 2 mini-games in order to “defuse the bomb”

**Wiring game**

For this, the player will be provided with a set of numbered wires (based on headphone wires) that they will need to plug into a series of holes. The value sequence in which the player needs to plug the wires will be predetermined. After the player plugs in all 4 wires, if the wires are in the incorrect order, the corresponding LEDs above the holes will light red, informing the player exactly which wire is in the incorrect position. While the wires are plugged in the incorrect order, the countdown will SPEED UP. If the player plugs the wires in the correct order, an LED in the bottom left corner will flash green to let them know the mini-game has been solved.

The ALU can be used to compare the order in which the wires are plugged with respect to the predetermined sequence. The ALU can be programmed to compare the input sequence with the correct sequence and send a signal to the FSM to change its state accordingly.

The FSM can be used to keep track of the state of the game and determine whether the player has plugged the wires in the correct order or not. Different states such as "idle," "waiting for input," "processing input," and "game over" can be defined. Each state can have specific actions and transitions to other states.

The predetermined sequence can be randomized at the start of each game using a pseudo-random number generator, using the pn\_gen module in Alchitry.

**Reaction time game**

In this game, the player must respond as quickly as possible to visual stimuli, i.e an LED flash. As soon as the LED flashes, the player has to press the button attached within a specified amount of time. Successfully hitting the button in time will result in the one of the 5 green LEDs above the button to light up, signalling to the player they need to light all 5 to pass the minigame. Hitting the button after the time limit will result in a 4 seconds being DEDUCTED from the countdown. If the player manages to light up all 5 LEDs, an LED in the bottom left corner will flash green to let the player know the game has been solved.

The ALU can be used to measure the time elapsed between when the LED flashes and when the player presses the button. The ALU can be programmed to start measuring time when the LED flashes and stop measuring time when the button is pressed.

The FSM can be used to keep track of the state of the game and determine whether the player has pressed the button within the specified time or not. Different states such as "idle," "waiting for input," "processing input," and "game over." can be determined. Each state can have specific actions and transitions to other states.

**A player can only play one game at a time as the games are set to be played in a fixed order.**

## User Manual

<This should describe how one uses the prototype in the end after you’ve finished building your project>

User presses the Start button located on the right face of the box to begin the game. Timer will begin the countdown. User must solve the games IN ORDER, starting with the wire game. They can only play ONE mini-game at a time. Once they complete solving the 1st puzzle, they can move on to the next game (reaction time game). Completing all mini-games before the timer ends will lead to a successful bomb defusal and a win for the player.

**How to solve Wiring game:** Players will need to plug in the set of wires into the holes. Players will need some luck in getting the correct sequence, as they are randomised and will not be provided with it beforehand. If they successfully obtain the sequence after trial and error (or they get lucky and guess it early) they have solved the game.

**How to solve Reaction time game:** Quite simple. An LED will flash, signaling the player to hit the button attached. If they hit it within the specified timeframe, one of the 5 green LEDs above the button will light up. Light all 5 LEDs to solve this game successfully

**Successfully complete all mini-games in order to defuse the bomb!** (LEDs located at the bottom corners will let you know the status of each minigame.

Red: Mini game inactive / not played

Orange: Mini game in progress

Green: Mini game solved

## Design Inspirations

<Illustrate where you got your design ideas from, and how you have improved on it>

Insipired by the video game “Keep Talking and Nobody Explodes”. Our take on the game is Single-Player as opposed to 2-player. This way, anyone can play and enjoy the game at their convenience, as opposed to waiting for another person to join.

## Test Scenarios

<Expected input and output, even illegal inputs>

**Reaction Time Game**

Expected input: Pressing the button within the time limit.

Test Cases:

1. Pressing the button before and after the LED flashes → time deduction
2. Not pressing the button when the LED flashes → time deduction
3. Pressing the button when the LED is on → light up 1 more green LED

**Wiring game**

Expected input: Correct order of wires inserted.

Test cases:

1. Wrong wires inserted → timer speeds up, wire LEDs show incorrect wires
2. Correct wires inserted → game LED turns green, wire LEDs are off
3. Wires connected and disconnected rapidly (check integrity of FSM, ensures that states are not skipped due to invalid values)

# 

# **Electronic Design**

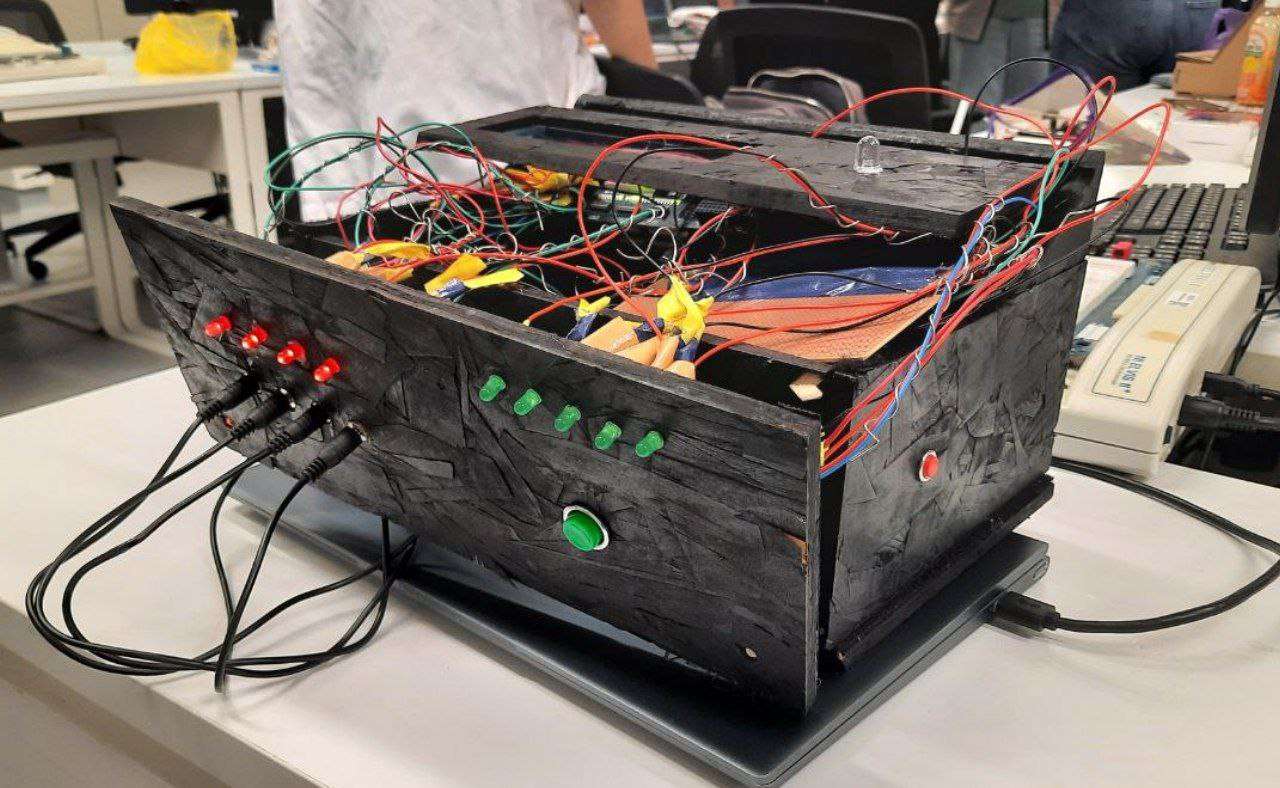
*Lots of diagrams are expected here.*

## Input/Output

<Input and output devices for your final prototype, **all** expected values of output and input>

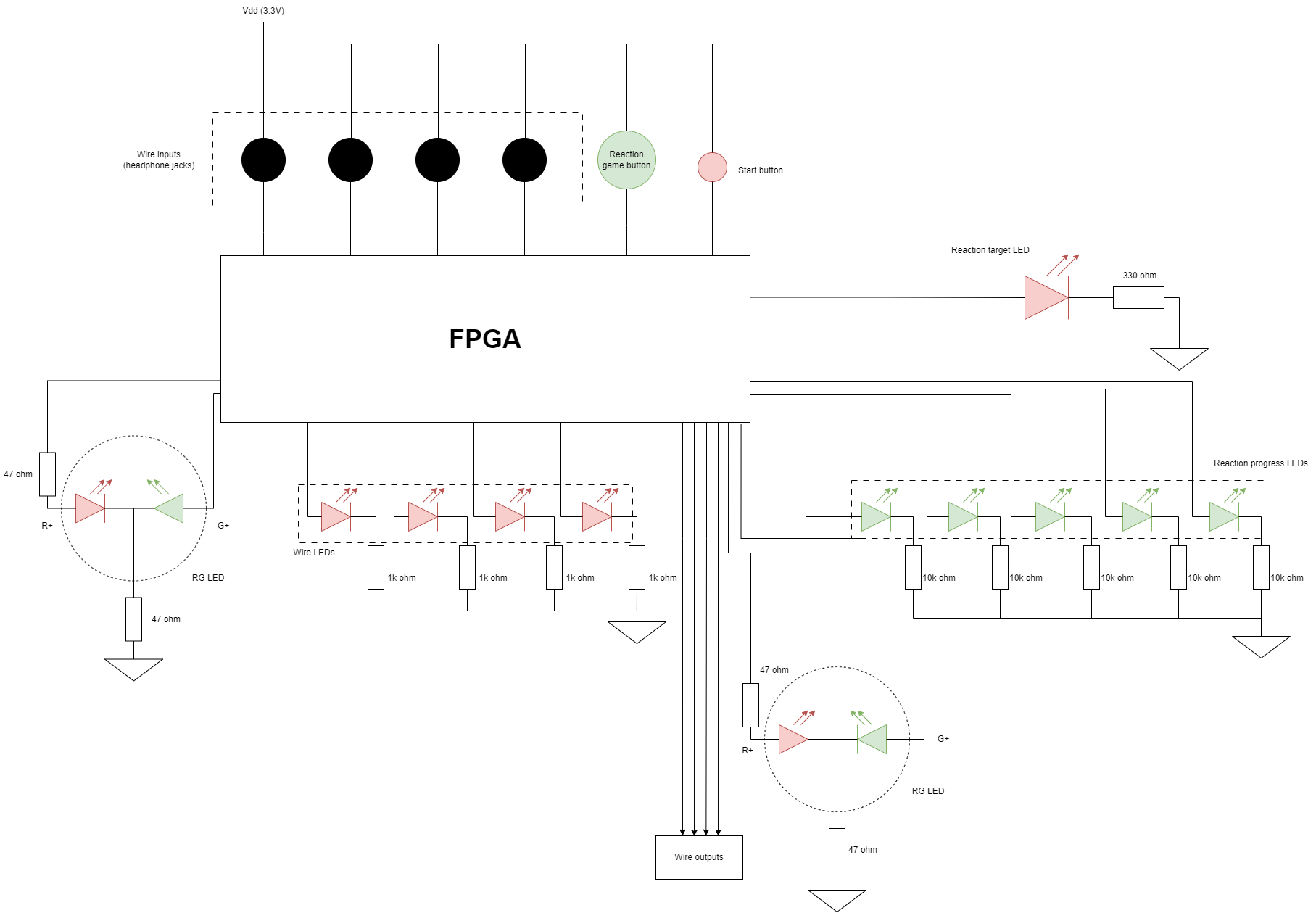
Reaction: 1 big red LED, 1 big button, 5 small green LEDs

Wire: 4 3.5mm jack wires & receptacles, 4 small red LEDs

Main: Start button on right-hand face. 2 RG LEDs, 7-segment display on Io shield, 

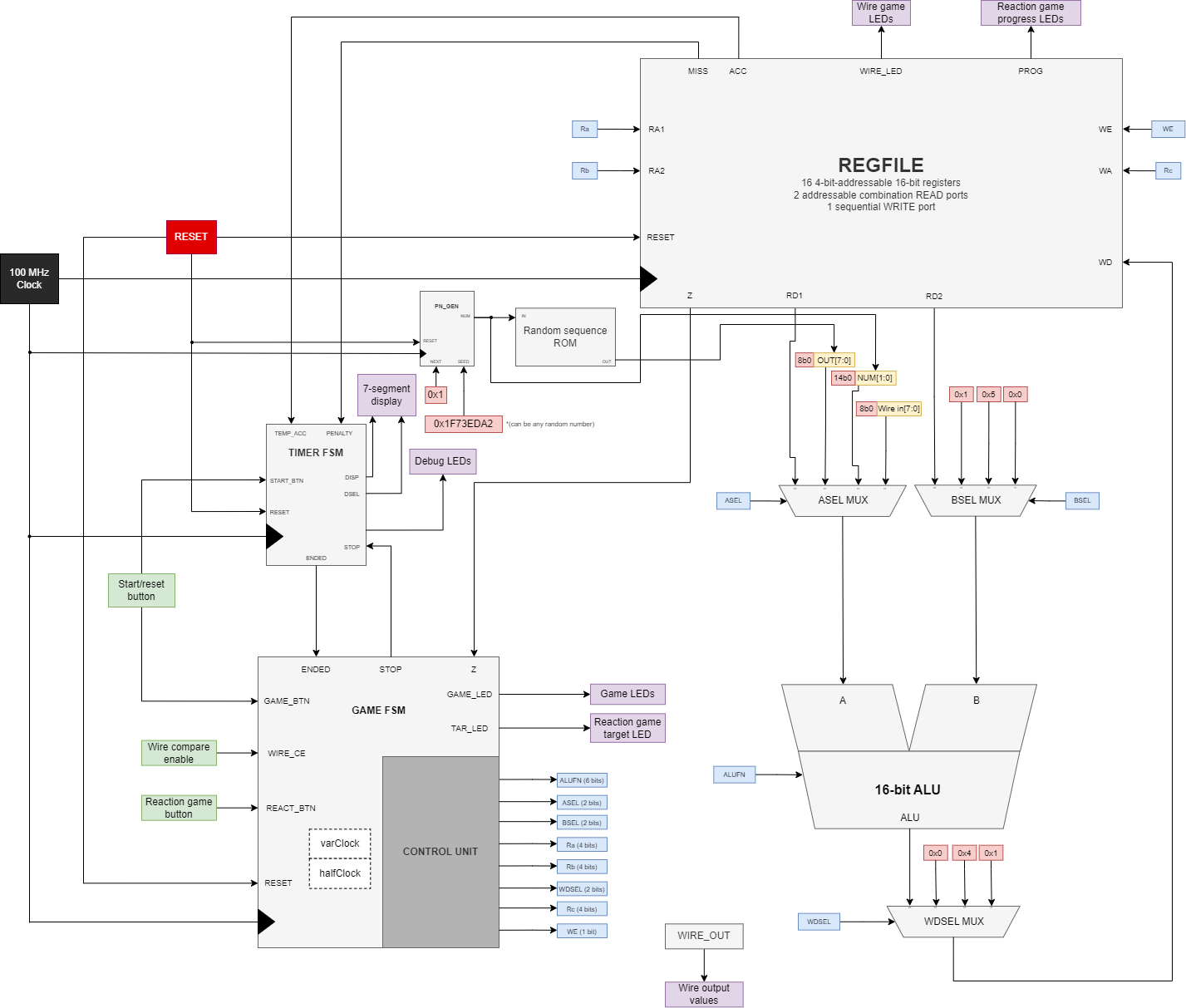
## 

## 



## Datapath

<Describe the datapath of your game, how many regs used, addressing, control signals, etc>



**Registers (16-bit) used**

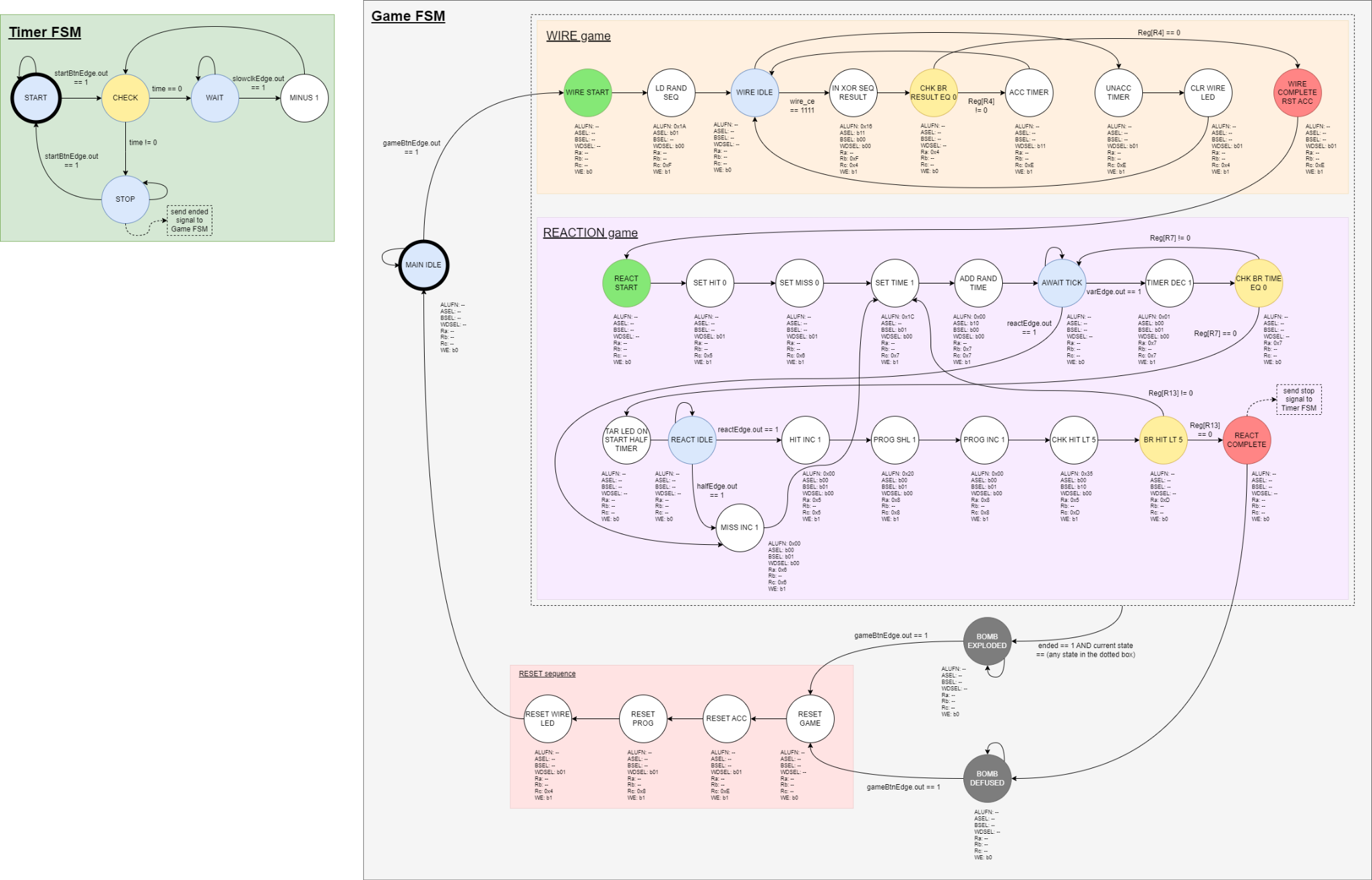
* 0x4: Wire LED output register
* 0x5: Hit counter
* 0x6: Miss counter
* 0x7: Random timer register
* 0x8: Reaction progress LED output register
* 0xC to 0xD: Temporary registers
* 0xE: Temporary timer acceleration register
* 0xF: Wire sequence register
* Unused: 0x0 to 0x3, 0x9 to 0xB

**Control Signals used:**

* ASEL(2 bits)
* BSEL(2 bits)
* WE(1 bit)
* WDSEL(2 bits)
* Ra(4 bits)
* Rb(4 bits)
* Rc(4 bits)
* ALUFN(6 bits)
* RESET(1 bit)

## FSM

<Describe your game state machine and all control signal values + output at each state>



*\*For full resolution diagrams:* [*https://drive.google.com/drive/folders/1AzVUuHfrcTRT30M7o3WsQar\_jser\_3g\_?usp=sharing*](https://drive.google.com/drive/folders/1AzVUuHfrcTRT30M7o3WsQar_jser_3g_?usp=sharing)

# **Budget**

<Components and their cost>

|  |  |
| --- | --- |
| **Component** | **Cost (S$)** |
| 10mm red LED x1 | 2.00 |
| 47 Ohm resistor x6 | 1.00 |
| 10mm momentary button x1 | 1.50 |
| 15mm momentary button x1 | 2.00 |
| 5mm RG LED x2 | 4.00 |
| 3.5mm headphone jack x4 | 6.00 |
| 3.5mm male-to-male cables x2 | 12.00 |
| 5mm heat-shrink tubing (0.5m long) | 1.50 |
| Black spray paint | 8.00 |
| **TOTAL** | **38.00** |

# 

# **Summary**

B3S is a bomb defusal game that tests the players’ reaction speed and problem solving skills Its logic is constructed using FSM and components of a basic CPU such as ALU and regfile, while the physical assembly consists of the FPGA, LEDs and buttons, as well as 3.5mm headphone wires and jacks. Players are required to solve all minigames before the timer ends to “defuse the bomb”, or risk “blowing up the campus”. Building this game was a fulfilling experience as it allowed us to apply the concepts and knowledge learnt throughout the 50.002 course.

# 

# **References**

**“**“Keep Talking and Nobody Explodes” <https://www.youtube.com/watch?v=axEAce1fN_I>

# 

# **Appendix**

## ALU Design and Tests

Design and Functionality

Our ALU is based on the Beta CPU ALU design, and is capable of performing the following operations:

|  |  |  |
| --- | --- | --- |
| Operation | ALUFN[5:0] in binary | ALUFN in hex |
| ADD | 000000 | 0x00 |
| SUB | 000001 | 0x01 |
| MULT | 000010 | 0x02 |
| AND | 011000 | 0x18 |
| OR | 011110 | 0x1E |
| XOR | 010110 | 0x16 |
| LD ‘A’ | 011010 | 0x1A |
| NAND | 010111 | 0x17 |
| NOR | 010001 | 0x11 |
| SHL | 100000 | 0x20 |
| SHR | 100001 | 0x21 |
| SRA | 100011 | 0x23 |
| CMPEQ | 110011 | 0x33 |
| CMPLT | 110101 | 0x35 |
| CMPLE | 110111 | 0x37 |

\* Additional functionalities

On top of these functions, the Boolean unit in our ALU can perform any 1- or 2-input combinational logic function, since the truth table is encoded within the ALUFN code itself.

Test Cases

To make sure that our ALU works as intended, we tested our ALU against some test cases that are known to be correct. We created 8 types of test cases, with 8 test cases for each type, hence a total of 64 test cases are stored in a ROM module and accessed by feeding in the 6-bit test case address using a DFF. The automatic tester then compares the expected output with the actual ALU output from the given inputs, and shows an error when a mismatch is encountered.

We have included the following test cases in our automatic tester, in ascending order of test case address:

* ADD test cases
  + 0xC400 + 0xF880 = 0xBC80, ZVN = 001
  + 0x1A80 + 0x70D4 = 0x8B54, ZVN = 011 (positive overflow)
  + 0x8293 + 0xAC14 = 0x2EA7, ZVN = 010 (negative overflow)
  + 0x7C09 + 0x83F7 = 0x0000, ZVN = 100
  + 0x89AE + 0x7F53 = 0x0901, ZVN = 000
  + 0x4586 + 0x265C = 0x6BE2, ZVN = 000
  + 0x5DBC + 0x7955 = 0xD711, ZVN = 011
  + 0x9917 + 0xA3D1 = 0x3CE8, ZVN = 010
* SUB test cases
  + 0xFFFF - 0xFFFF = 0x0000, ZVN = 100
  + 0xA5DC - 0x766F = 0X2F6D, ZVN = 010
  + 0xA745 - 0xFFFF = 0xA746, ZVN = 001
  + 0x46FF - 0xFFF1 = 0x470E, ZVN = 000
  + 0x8C0F - 0x71FA = 0x1A15, ZVN = 010
  + 0x5465 - 0xB6A9 = 0x9DBC, ZVN = 011
  + 0x3354 - 0xA45E = 0x8EFH, ZVN = 011
  + 0x3C61 - 0xF080 = 0x4BE1, ZVN = 000
* MULT test cases
  + 0x01C8 × 0xFD39 = 0x0D88
  + 0xC459 × 0xED38 = 0x5878
  + 0x4385 × 0x1117 = 0xE5F3
  + 0x0555 × 0x00ED = 0xEFB1
  + 0x04BE × 0x000F = 0x4722
  + 0x1579 × 0x2D0A = 0x1BBA
  + 0x00FA × 0x00CE = 0xC92C
  + 0x77C8 × 0xF391 = 0xB048
* BOOL test cases
  + 0x0483 OR 0xD554 = 0xD5D7
  + 0xC053 XOR 0x250E = 0xE55D
  + 0x92D3 LD ‘A’ 0x8205 = 0x92D3
  + 0x52A7 AND 0x9C88 = 0x1080
  + 0x0542 LD ‘A’ 0xFFFF = 0x0542
  + 0xFFEE XOR 0x0111 = 0xFEFF
  + 0x00D1 NAND 0xFFAF = 0xFF7E
  + 0xFA34 NOR 0x0001 = 0x05CA
* SHL test cases
  + 0x8641 << 0xB = 0x0800
  + 0x3FFC << 0xE = 0x0000
  + 0x7DE5 << 0x7 = 0xF280
  + 0x5942 << 0x9 = 0x8400
  + 0x22C5 << 0xA = 0x1400
  + 0x0FF6 << 0x2 = 0x3FD8
  + 0x5521 << 0x1 = 0xAA42
  + 0xFFFF << 0x5 = 0xFFE0
* SHR test cases
  + 0x54BF >> 0x2 = 0x152F
  + 0xD1B5 >> 0xD = 0x0006
  + 0x3184 >> 0x6 = 0x00C6
  + 0x7652 >> 0x9 = 0x003B
  + 0xFFFF >> 0x1 = 0x7FFF
  + 0xFFEE >> 0xF = 0x0001
  + 0x0001 >> 0x3 = 0x0000
  + 0xFA34 >> 0x4 = 0x0FA3
* SRA test cases
  + 0XF542 >>> 0xD = 0xFFFF
  + 0x6A20 >>> 0x7 = 0x00D4
  + 0x9DB2 >>> 0xB = 0xFFF3
  + 0x3541 >>> 0x2 = 0x0D50
  + 0x7FDF >>> 0x8 = 0x007F
  + 0x8293 >>> 0x5 = 0xFC14
  + 0x1A80 >>> 0x6 = 0x006A
  + 0xC400 >>> 0x3 = 0xF880
* CMP test cases
  + 0xA88B <= 0x7633 = 0x1, ZVN = 010
  + 0x5745 < 0x7242 = 0x1, ZVN = 001
  + 0x9566 == 0x7453 = 0x0, ZVN = 010
  + 0x68A5 <= 0xA425 = 0x0, ZVN = 011
  + 0x8C45 == 0x8C45 = 0x1, ZVN = 100
  + 0x4432 < 0x8D3C = 0x0, ZVN = 011
  + 0xDD33 <= 0xDD33 = 0x1, ZVN = 100
  + 0x263A < 0xFF73 = 0x0, ZVN = 000

Note: Manual input is also possible to test specific inputs and outputs

## Prototype code + Repo link

GitHub repository link (restricted access): <https://github.com/50002-computation-structures/1d-project-group09>

## Project Management Log: Team Tasks

|  |  |
| --- | --- |
| Team member | Task |
| Lim Jie Han | Wiring game (code + FSM + Datapath) |
| Yash Yadav | Wiring game (code + FSM + Datapath) |
| Sarang Nambiar | Reaction time game (code + FSM + Datapath) |
| Ryan Pey Jun Hao | Reaction time game (code + FSM + Datapath) |
| Siddharth Ganesh | Hardware; Assembly + Soldering |
| Esan Natraj | Hardware; Assembly + Soldering |

## Components’ Specifications

Hardware:

* 5mm LEDs x11 (2 RG, 5 green, 4 red; for reaction time, wiring game & game progress)
* Large LED x1 (reaction time)
* Momentary buttons x2 (10mm x1, 15mm x1; for reaction time, start game)
* 3.5mm male-to-male cable x2 (wiring game)
* 3.5mm jack x4 (wiring game)
* Alchitry Au FPGA board with Xilinx Artix 7 x1 (controls logic of entire game)
* Alchitry Br breakout board x1 (connects FPGA to external I/O)
* Alchitry Io shield x1 (timer display + LEDs)
* Wood, acrylic (housing)

Software (Lucid modules):

* pn\_gen (random number generator)
* button\_conditioner (button & wire input debouncer)
* edge\_detector (detect rising and/or falling edge for external input)
* counter (slower clock for timers)
* decoder (for 7-segment timer display)
* bin\_to\_dec (convert binary/hex values to decimal for timer display)