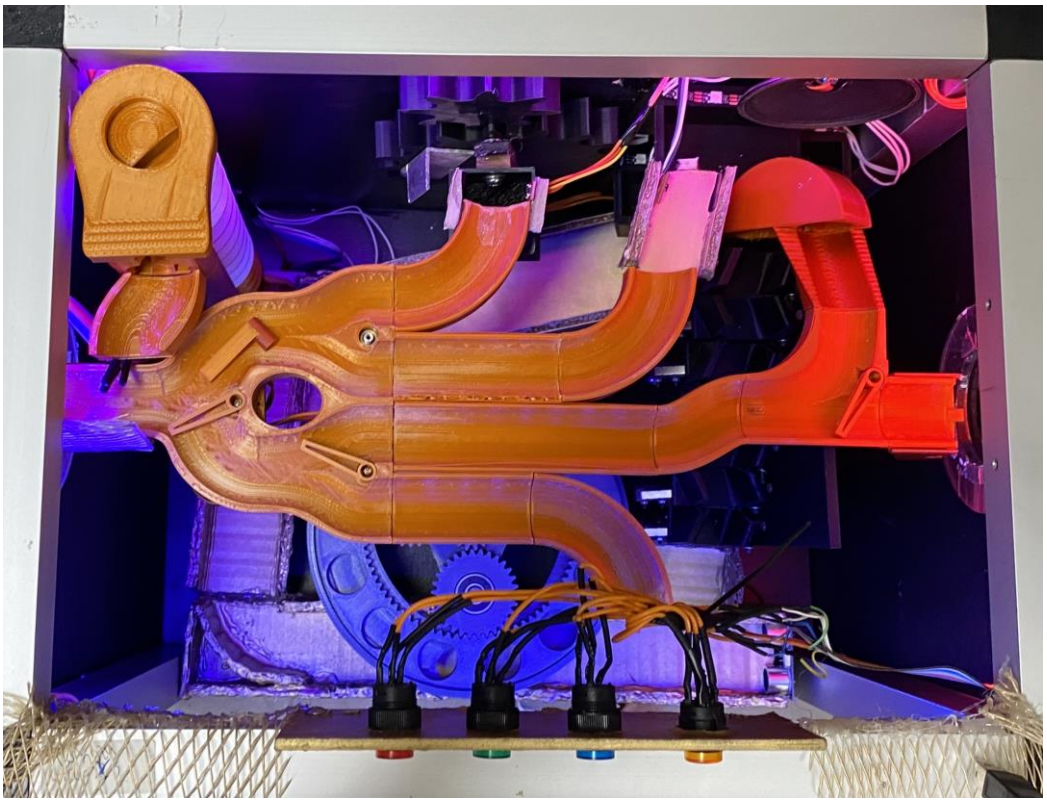
CMPE3006 Advanced Digital Design

Crazy Machine Project – Final Report



Group 2 - 2024 Semester 1

Bayezid Hossain 20189879
Khaled Ali Zulfikar 20166322
Ethan Rushack 20188041
Jonathon Vagg 20162401
Lea Bouska 20155414
Liam Ogg 20550967

Table of Contents

1. INTRODUCTION 1

2. CRAZY MACHINE PROJECT PLAN 3

2.1. TASK ALLOCATION 3
2.2. CRAZY MACHINE PROJECT GANTT CHART 4

3. CRAZY MACHINE PROJECT BLOCK DIAGRAMS 6

3.1. HARDWARE BLOCK DIAGRAM 6
3.2. SOFTWARE SEGMENT BLOCK DIAGRAMS 8

4. `SPLITTER AND SCREW LIFT 12

4.1. DESIGN 12
4.1.1. Design Plan 14
4.2. HARDWARE SPECIFICATIONS 15
4.2.1. Parts Required 15
4.2.2. Software Specifications 18
4.3. ARDUINO SOFTWARE SPECIFICATION AND FPGA DESIGN 32
4.3.1. Arduino Software 32
4.3.2. FPGA Design 34

5. PLANETARY REVOLUTION 49

5.1. ASSUMPTIONS 50
5.2. DECISIONS 51
5.3. DESIGN PROCESS 51
5.3.1. Design Plan 51
5.3.2. Initial design 52
5.3.3. Final design 53
5.4. IMPLEMENTATION 53
5.5. HARDWARE SPECIFICATION 56
5.6. VERIFICATION – PROPOSED TESTING METHODOLOGY & RESULTS 56

6. FERRIS WHEEL 57

6.1. ASSUMPTIONS 57
6.2. DECISIONS 58
6.3. DESIGN PROCESS 59
6.3.1. Design Plan 59
6.3.2. Initial Design 59
6.3.3. Final Design 60
6.3.4. Recycled Components 62
6.4. IMPLEMENTATION 63
6.4.1. Building 63
6.4.2. Hardware Specifications 66
6.4.3. Arduino Code 69
6.5. VERIFICATION 70
6.5.1. Testing of Individual Electronic Components 70
6.5.2. Testing of Electronic Components 72
6.5.3. Testing of Electronic & Mechanical Components 73

7. PIANO SLIDE 74

7.1. ASSUMPTIONS 74

7.2. Decisions 74
7.3. Design Process 75
7.3.1. Design Plan 75
7.3.2. Initial Design 75
7.3.3. Prototyping Design 77
7.3.4. Final Design 78
7.4. Implementation 78
7.4.1. Building 78
7.4.2. Hardware Specifications 78
7.4.3. Circuit Configuration 79
7.4.4. VHDL Design 80
7.5. Verification 82
7.5.1. Testing of Mechanical Components 82
7.5.2. Testing of Electronic Components 82

8. PINBALL LADDER 84

8.1. Assumptions 84
8.2. Decisions 85
8.3. Design Process 85
8.3.1. Initial Design 85
8.3.2. Step Final Design 87
8.3.3. LED Design 87
8.3.4. Hardware Specifications 89
8.3.5. Verification and implementation 89
8.3.6. FPGA Design 90

9. BALL RETURN AND DECORATIVE ELEMENTS 93

9.1. Assumptions 93
9.2. Decisions 93
9.3. Design Process 94
9.3.1. Initial Design 94
9.3.2. Final Design 94
9.4. Hardware Specifications 95
9.5. Circuit Diagram 95
9.6. Arduino Code 96
9.6.1. Ultrasonic Sensor Code 96
9.6.2. LED Brightness code 97
9.6.3. Testing and Debugging 97
9.7. LED decorations 98

10. INTEGRATION OVERVIEW 99

10.1. FPGA Design 99
10.1.1. Pin Assignments 101
10.2. Integration Challenges 104
10.2.1. Connecting Ball Splitter with the other sections 104
10.2.2. Ball Splitter Integrity 104
10.2.3. Ball Splitter Connection to Ferris Wheel 105
10.2.4. Ferris Wheel Ball Catcher 105
10.2.5. Ferris Wheel FSR Breakage 106
10.2.6. Ball Splitter connection to Planetary Revolution 106
10.2.7. Planetary revolution cable routing 106
10.2.8. Connectivity of Ball Splitter to Piano Slide 107

10.2.9. LED wiring Pinball Ladder 107
10.2.10. Connectivity of the Ball Splitter to Pinball Ladder 107

11. REFERENCES 108

1. Introduction

The Crazy Machine Project consists of designing, constructing and implementing a marble machine which runs for approximately 1 minute. The machine must fit the design requirements of using fixed entry and exit points, keeping the ball within the confines of the box and using a minimum number of different sensors and actuators in the design. Additionally, the machine should be reliable enough to perform a demonstration and should reset to its initial state following the completion of the machine. The machine should feature different combinations of sensors and actuators and operate using a mixture of Arduino and FPGA code with some form of communication between these devices.

The goal of our machine is to create a marble run where the different sections of the machine are able to be completed interchangeably. To achieve this, our machine is comprised of a central loop consisting of a marble splitter which determines the path of the ball, 4 different paths for the ball to take and a lift to return the ball to the starting position. As such the machine comprises of splitter, ball return, marble lift and 4 different sections that can be completed interchangeably. In order to allow the selection of the path of the ball, several buttons are used prior to entering the ball. These buttons are optional, with the machine falling back to a random order if user input is not provided. The expected process for the machine is:

1. The order of the sections is set, and the splitter is set to the selected path.
2. The ball is entered into the machine.
3. Sections 1-4 are completed in the specified order, with the ball returning to the starting position using a marble lift.
4. The splitter is set to the exit path and the ball exits the machine.

The theme of our machine is "Steampunk", allowing for mechanical motifs which are integrated into the different sections of the machine and the decoration. The decoration of the machine heavily features recycled components and is stylised towards a cobbled-together steampunk design. The different sections of our machine were designed with this in mind, featuring mechanical designs where “form follows function” rather than the other way around.

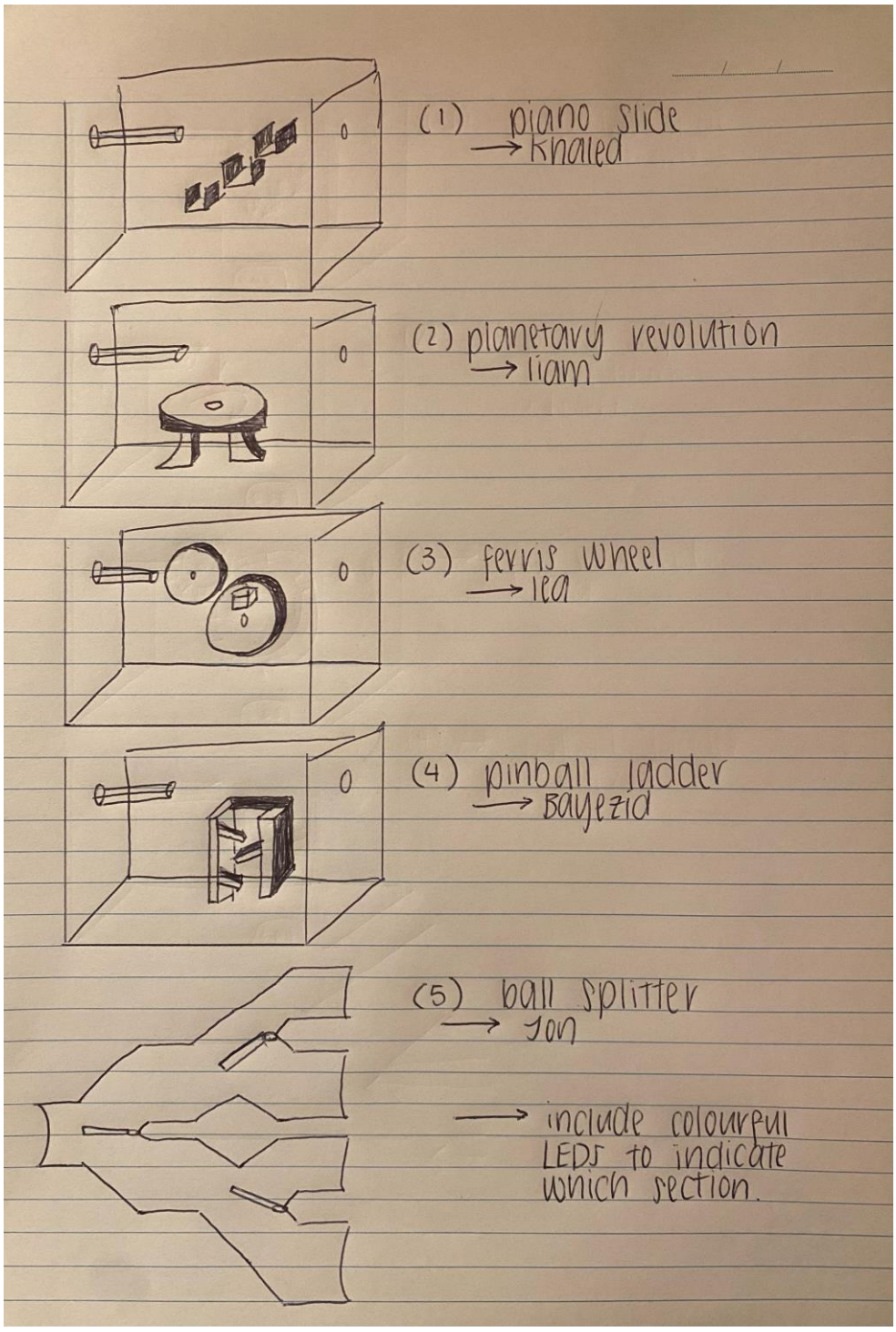


Figure 1 Initial Concept Drawing

Our initial sketches shown in **Error! Reference source not found.** outline our original design from the first lab session and highlights our core methodology of multiple sections interconnected to form the larger machine. These sketches show our initial concepts for the machine and form the basis of our designs moving forward.

2. Crazy Machine Project Plan

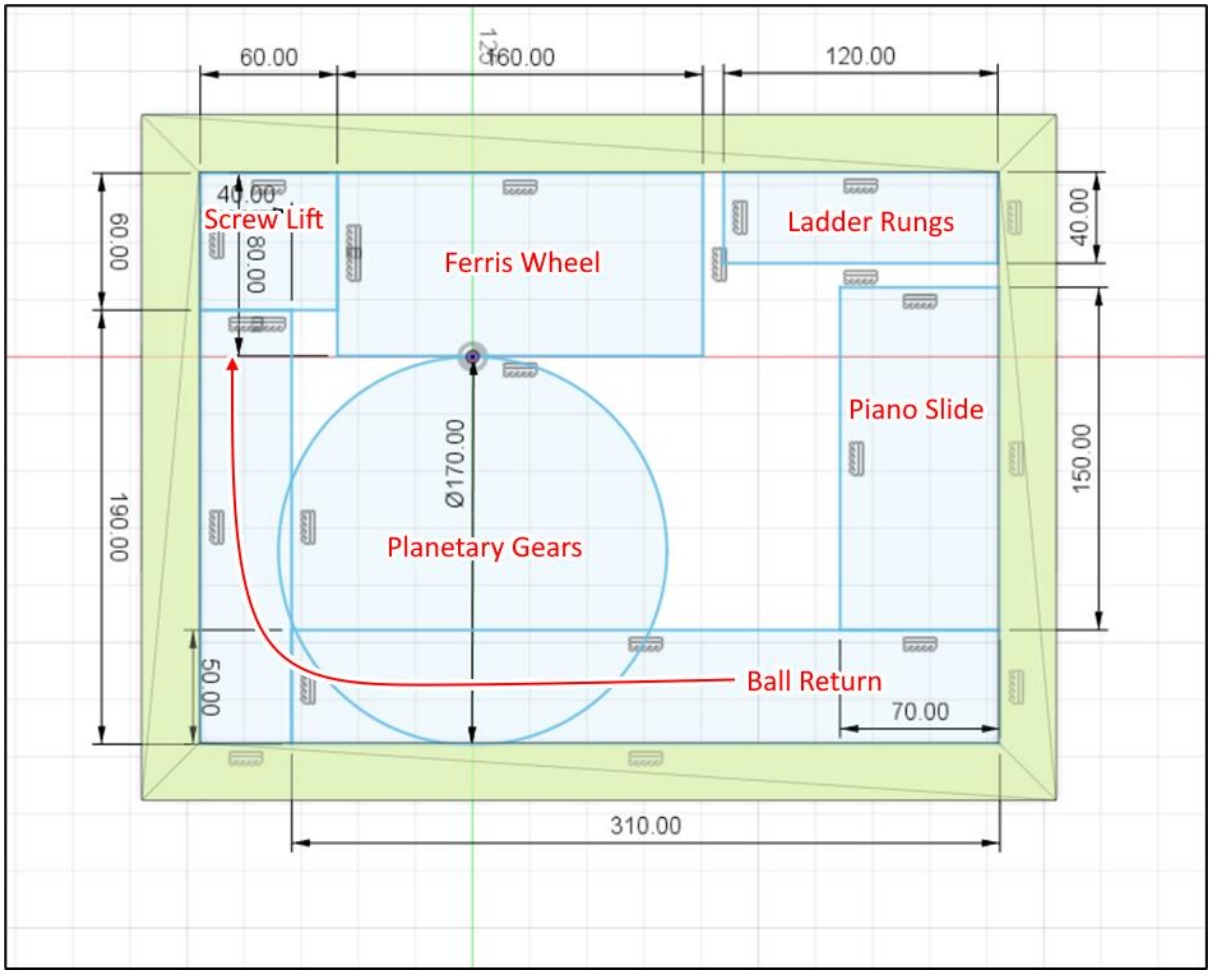


Figure 2: Top Level View - Section Space Allocation

2.1. Task Allocation

The timeline of the project has been split into 6 major stages, planning, design, constructing, coding, integration and testing & validation (as can be seen in the Gantt Chart). The planning stage consists of the entire group designing an initial sketch and deciding on which components to use.

In the design stage, each member is assigned to a specific design element. Most of the team members have previous experience in 3D computer aided design (CAD) software, such as Lea having experience with AutoCAD, Liam, Jon, and Ethan having experience with Fusion 360 and Khaled having experience with SolidWorks. With this in mind, the team members have opted to 3D print their designs. Collaboration must occur between the team members to ensure that the designs are seamless, and do not interfere with each other. To ensure that interference between sections does not occur, the team will maintain a collaborative 3D model to use as a reference and each member will be allocated space for their section. Importantly, the splitter will be allocated the space at the entry and exit point level which should not significantly impact the other sections.

In the construction stage each member is responsible for their section, with tests in the project box to ensure that sections are being constructed as to not interfere with one another. Following the completion of each section, group members will carry out testing to ensure that their section meets the design constraints and effectively interfaces with the other sections of the machine.

Simultaneously, the programming stage of the project takes place, with prototyping and testing of the code for the machine being performed for each of the sections. As a group we decided which sections of the code would be more applicable for FPGA and which sections are more suitable for use with the Arduino. Following the discussions Khaled was assigned the role of FPGA code manager with the goal of overseeing the integration of the FPGA code for different sections. Similarly, Jonathon was assigned the role of Arduino code manager and was responsible for ensuring the Arduino code is integrated correctly.

Throughout the integration, testing & validation stage, all the team members worked collaboratively to develop and integrate their sections with the rest of the machine. Due to the interconnectedness of the machine overall this required accurate and professional communication between the group members. As the validation stage approached this included integrating the different sections' code and components together and ensuring that the different components of the machine work together as intended. This stage includes final revisions to the machine and addition of decorative elements.

Crazy machine project

MACHINE SECTIONS	**TEAM MEMBER**
Planetary Revolution	Liam
Ferris Wheel	Lea
Piano Slide	Khaled
PINBALL LADDER	Bayezid
Splitter + Screw Lift	Jonathon
Ball Return	Ethan

Table 1: Task Allocation for Crazy Machine Project

2.1. Crazy Machine Project Gantt Chart

Figure 3 Steampunk Crazy Machine Gantt Chart

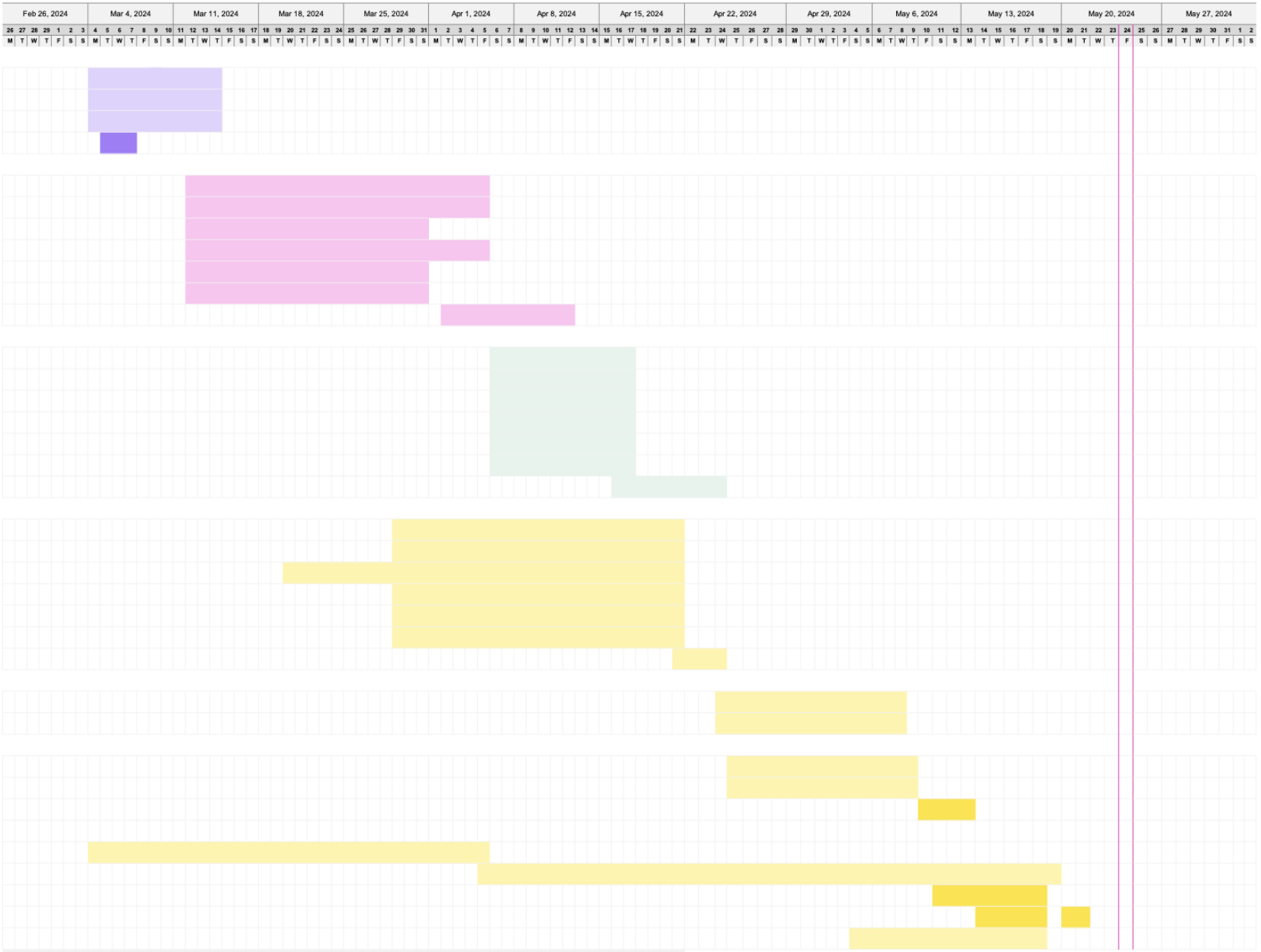
Crazy Machine Project

Bayezid Hossain, Jon Vagg, Khaled Ali, Lea Bouska, Liam Ogg, Ethan Rushack

Project start: **Mon, 3/4/2024**

Display week: 4th of June

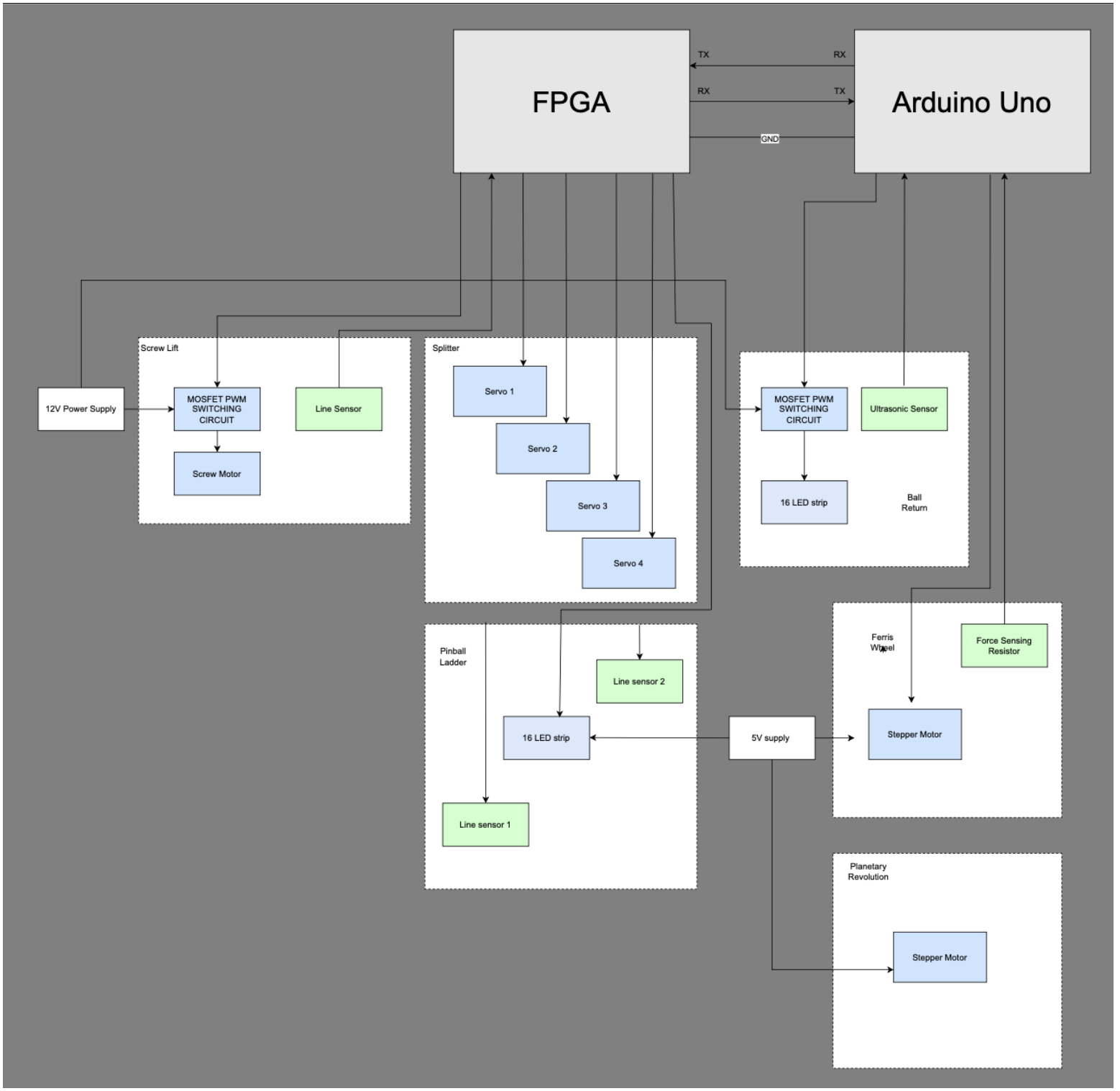
TASK	ASSIGNED TO	PROGRESS	START	END
Planning				
Area Division	All	100%	3/4/24	3/14/24
Component List	All	100%	3/4/24	3/14/24
Draft Sketch	All	100%	3/4/24	3/14/24
			3/5/24	3/7/24
Design				
Planetary Revolution	Liam Ogg	100%	3/12/24	4/5/24
Ferris Wheel	Lea Bouska	100%	3/12/24	4/5/24
Piano Slide	Khaled Ali	100%	3/12/24	3/31/24
Pinball Ladder	Bayezid Hossain	100%	3/12/24	4/5/24
Splitter + Ball Return	Jon Vagg	100%	3/12/24	3/31/24
LED and Ball interactions	Ethan Rushack	100%	3/12/24	3/31/24
Testing and Validation	All	100%	4/2/24	4/12/24
Constructing				
Planetary Revolution	Liam Ogg	100%	4/6/24	4/17/24
Ferris Wheel	Lea Bouska	100%	4/6/24	4/17/24
Piano Slide	Khaled Ali	100%	4/6/24	4/17/24
Pinball Ladder	Bayezid Hossain	100%	4/6/24	4/17/24
Splitter + Ball Return	Jon Vag	100%	4/6/24	4/17/24
LED and Ball interactions	Ethan Rushack	100%	4/6/24	4/17/24
Testing and Validation	All	100%	4/16/24	4/24/24
Coding				
Planetary Revolution	Liam Ogg	100%	3/29/24	4/21/24
Ferris Wheel	Lea Bouska	100%	3/29/24	4/21/24
Piano Slide	Khaled Ali	100%	3/20/24	4/21/24
Pinball Ladder	Bayezid Hossain	100%	3/29/24	4/21/24
Splitter + Ball Return	Jon Vag	100%	3/29/24	4/21/24
LED and Ball interactions	Ethan Rushack	100%	3/29/24	4/21/24
Testing and Validation	All	100%	4/21/24	4/24/24
Integration				
Integrate Code	All	100%	4/24/24	5/8/24
Integrate Components	All	100%	4/24/24	5/8/24
Testing and Validation				
Test Code	All	100%	4/25/24	5/9/24
Test Components	All	100%	4/25/24	5/9/24
Final Demo	All	0%	5/10/24	5/13/24
Deliverables				
Status report (20%)	All	100%	3/4/24	4/5/24
Final Report (30%)	All	100%	4/5/24	5/19/24
Crazy Machine Video (20%)	All	0%	5/11/24	5/18/24
Crazy Machine Demo (15%)	All	0%	5/14/24	5/21/24
Self Reflection (15%)	All	100%	5/4/24	5/18/24



3. Crazy Machine Project Block Diagrams

3.1. Hardware Block Diagram

The following image shows a Block Diagram of the Overall Hardware used in the Crazy Machine Project


FPGA
Arduino Uno
TX
RX
RX
TX
GND
Screw Lift
12V Power Supply
MOSFET PWM SWITCHING CIRCUIT
Line Sensor
Screw Motor
Splitter
Servo 1
Servo 2
Servo 3
Servo 4
MOSFET PWM SWITCHING CIRCUIT
Ultrasonic Sensor
16 LED strip
Ball Return
Pinball Ladder
Line sensor 2
16 LED strip
Line sensor 1
5V supply
Ferris Wheel
Force Sensing Resistor
Stepper Motor
Planetary Revolution
Stepper Motor

3.2. Software Segment Block Diagrams

The following segments depict the software block diagrams for the Steampunk Crazy Machine Project

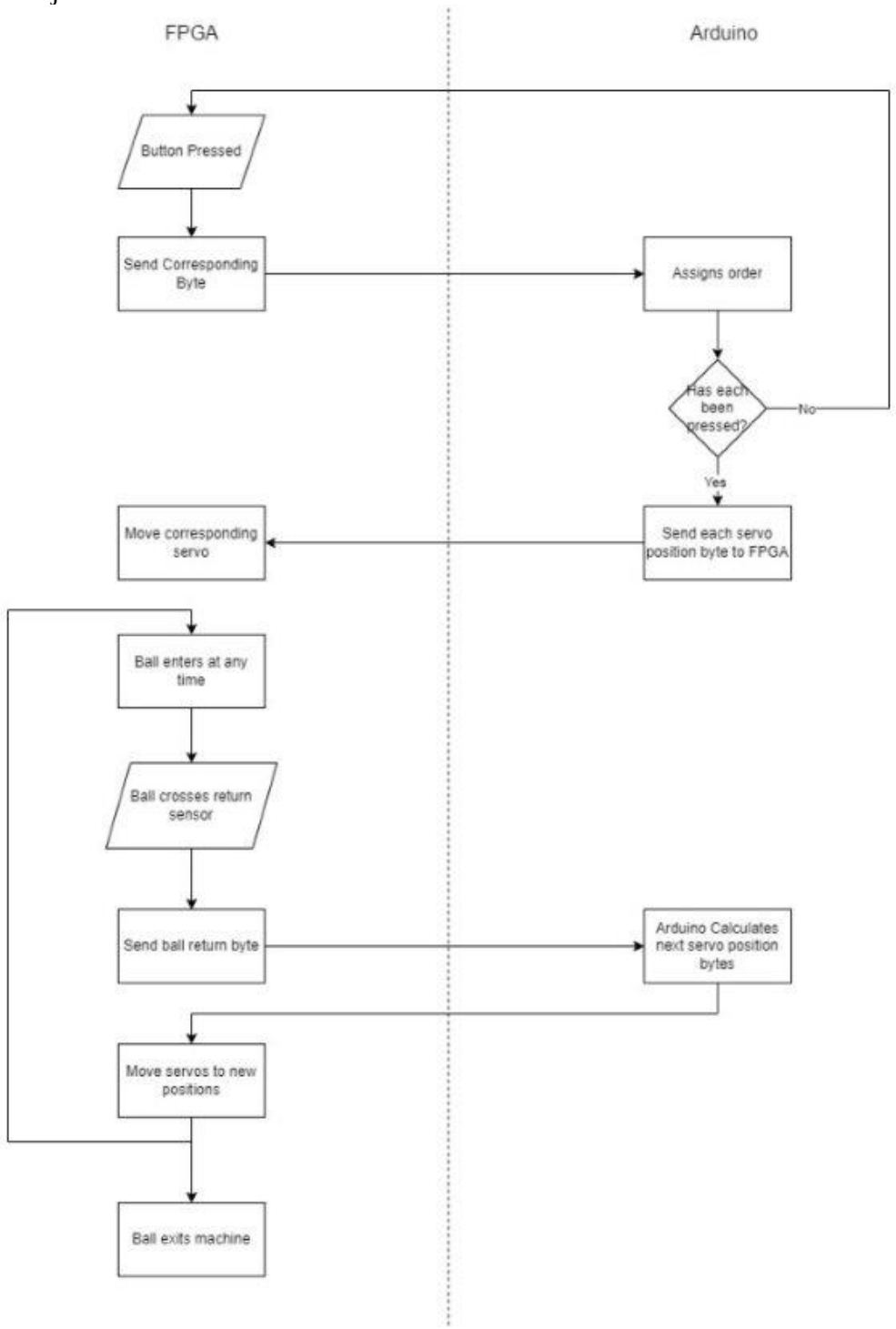


Figure 5 Ball Splitter & Screw Lift Block Diagram

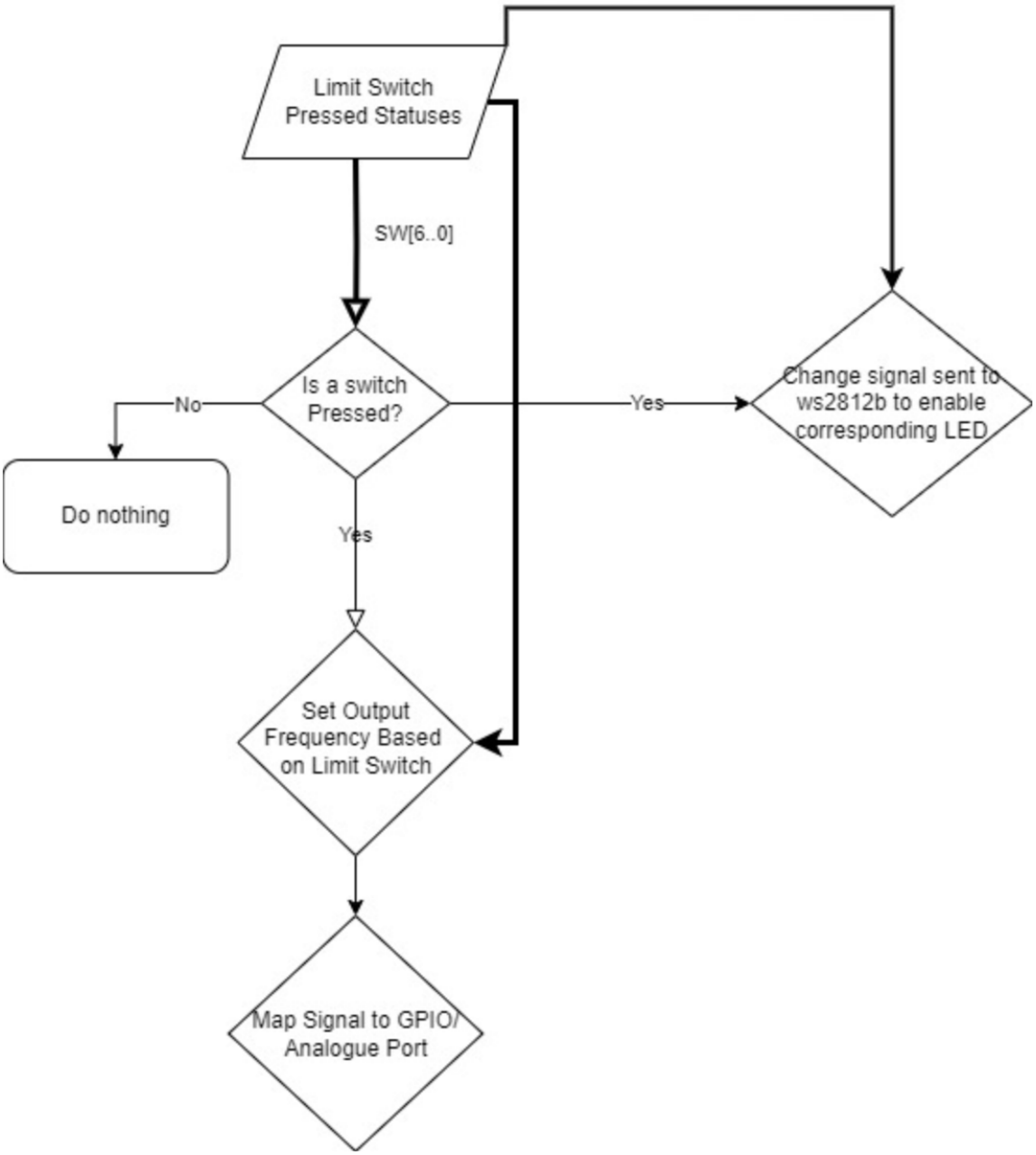


Figure 6 Piano Slide Block Diagram

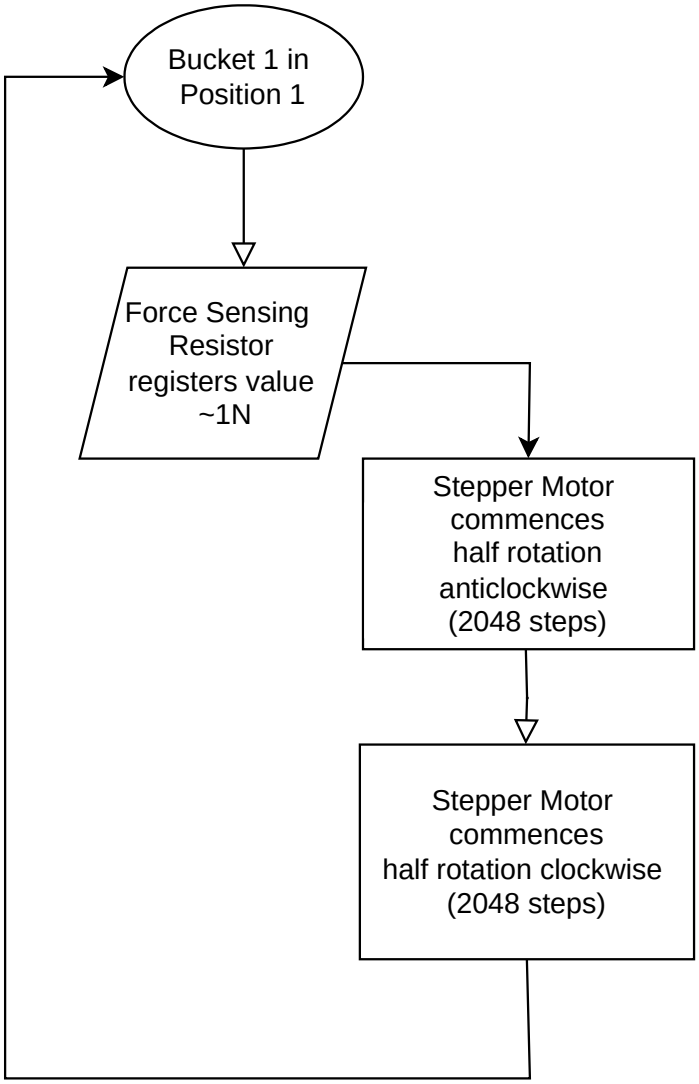

Bucket 1 in Position 1
Force Sensing Resistor registers value ~1N
Stepper Motor commences half rotation anticlockwise (2048 steps)
Stepper Motor commences half rotation clockwise (2048 steps)

Figure 7 Ferris Wheel Block Diagram

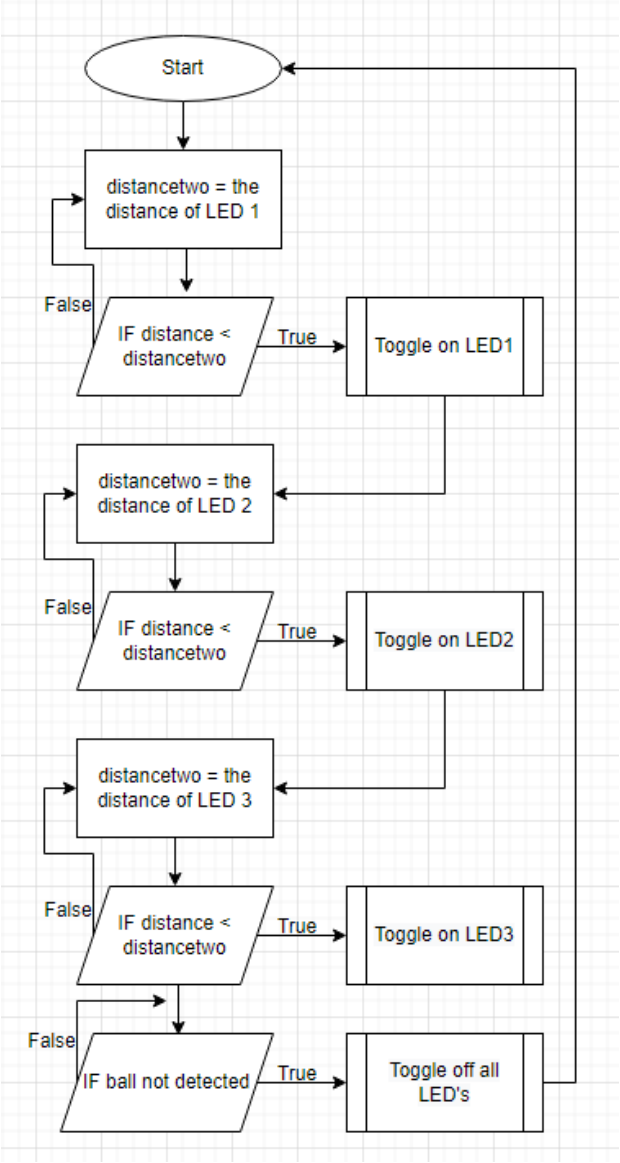


Figure 8 Decorative LED's Flowchart

Figure 9 Planetary Revolution Block Diagram

4. `Splitter and Screw Lift

The core element of our design is the marble interchange where the different sections of the machine are selected. This splitter connects the machine entrance, exit, ball return, and each of the stations together. The marble splitter comprises of a 1:5 splitter operated by 4 servos. The inputs to the splitter are the entrance and the output of the screw lift which are combined prior to entering the splitter. The outputs of the splitter are stations 1-4 and the exit. The screw lift is fed by the ball return and consists of a central helical screw which rotates and raises the ball back to the entry point of the splitter. The ball return consists of a collection tray at the front-bottom of the machine that guides the ball to the back left corner of the machine, where they will be fed into the screw lift.

To determine the order in which sections are completed, the machine will have 4 buttons mounted at the front. Prior to the ball being entered into the machine, the buttons may be pressed to determine the order of the different sections. The Arduino will parse this input to determine the order in which each section will be executed, e.g. 1-4-3-2. If these buttons are not pressed the machine will select a random order for the sections.

4.1. Design

The design for the screw lift and marble splitter consists of two main sections, the screw lift assembly and the marble splitter assembly. These sections are independently mounted in the box to increase modularity and overall, the design for these parts strives to maximise modularity. Modularity for this section is critically important as it allows the other group members to be more flexible with how their sections are constructed and allows for changes to be more easily made to the design to rectify unforeseen issues. The screw lift is mounted in the rear left corner of the machine and receive the ball at the base of the screw lift from the ball return section. The screw raises the ball to the top of the lift, where it releases the ball into the splitter in the same channel as the box entry point. In order to determine when the ball enters the lift, there is a line sensor at the base of the lift which detects the passing of the ball. This is important in determining when the ball has completed a section, so the machine knows when to change the servo positions. When this sensor is triggered, the FPGA sends the Arduino a request for the next position for the servos and based on the returned value, the servos controlling the path of the ball are changed to their appropriate position by the FPGA. Once all four sections are completed, the FPGA sets the servos to the exit pathway so that the ball may seamlessly exit the machine. The screw lift consists of a motor mounted internally inside the bottom screw section and is connected to a 3D printed screw approximately 270mm long. A set of guide rails raises the ball from the base of the screw lift to the start of the marble splitter. The motor operating the screw lift is configured to operate at an adjustable speed by using PWM modulation on the 12V input power. This allows for variable speed control to assist with getting a consistent 60 second timing for the machine. Upon reaching the top of the machine, the ball enters the return channel whereby it is returned to the entry point of the splitter.

The Splitter will consist of 4 splitters, arranged so that there are 5 possible pathways for the marble to take. The third channel accommodates the fourth splitter which splits the path near

the exit of the machine to allow the exit pathway to be selected. The initial design featured two servos and a solenoid to control the path of the ball but during testing it was determined that the final design must utilise four servos, one to split the original channel into two, and two more to split those channels into four. The decision to change the actuator arrangement will be explained in more detail in a later section of the report, but in summary, the change was made because using the solenoid in this way would require a significant amount of development time to develop a linkage mechanism which inherently would not offer significant improvement over simply using an additional servo.

Path selection is performed using four push buttons. When each button is pressed the FPGA detects the press and send a byte to the Arduino microcomputer for processing. After each button is pressed once, the Arduino will provide audio feedback and send a command to the FPGA to move the servos into the correct position. The station order is randomised on startup so that if the buttons are not pressed the machine falls back to a random order. The random order always consists of sections 1, 2, 3 and 4 being randomised with the exit always being the final path. The below diagram outlines the truth table required for different sections. It should be noted that during testing it was determined that the positions marked as 1 and 0 in the diagram below are reversed to their correct positions. The truth table below utilises the values from Figure 10 but the final code will have inverted values for the servo positions.

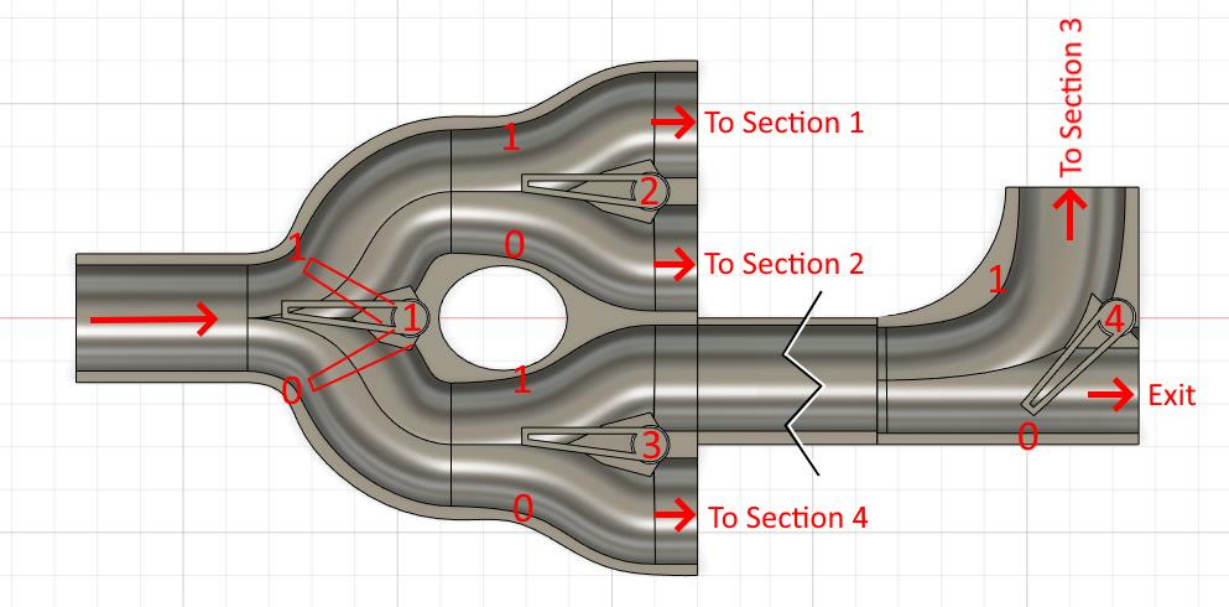


Figure 100: Splitter Design and Servo Position

Table 2 Truth Table for Servo Position Based on Pathway Selected

Servo 1	Servo 2	Servo 3	Servo 4	Pathway Selected
0	0	X	X	**1**
0	1	X	X	**2**
1	X	0	0	**3**
1	X	1	X	**4**
1	X	0	1	**Exit**

4.1.1. Design Plan

The design process for the Splitter and Screw Lift can be broken down into 5 sections. The planning stage was completed in the first week of semester and involved investigating different existing designs for the screw lift, drafting a sketch of the splitter design and determining the parts required for the design. Following this, the design stage heavily involved 3d modelling and validating the 3d modelled designs. During this stage several proof of concepts were created for each of the assemblies to ensure that the proposed designs effectively solved the problem and functioned as expected. This is to ensure that only valid designs are printed and hence reduce unusable prints and time. Simultaneously, the code and construction stages were performed. The code stage involved creating proof of concept code for each of the different functionalities of the system and then implementing that code in a way that the other group members could easily integrate their code with. This included using coding best practices and ample code documentation. Also performed simultaneously the construction stage prioritised validating the 3d printed designs were compatible with each other and fit in the box as desired. Finally, the integration stage comprised of combining code with the other group members and installing sections of the machine into the box and resolving any conflicts between sections that occurred as a result.

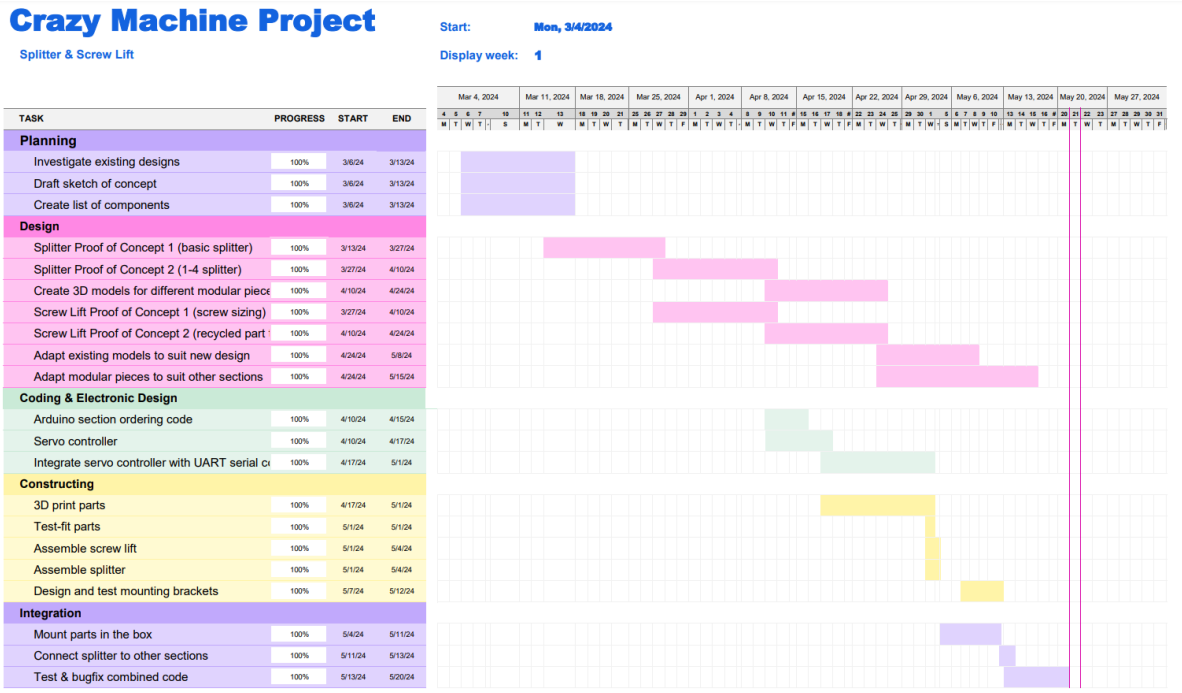


Figure 111 Splitter and Screw Lift Gantt Chart

4.2. Hardware Specifications

4.2.1. Parts Required

Table 3 Required Parts for Ball Splitter & Screw Lift

QTY.	Part
4	9g Servo
4	Push buttons
1	MOSFET Switching Board
1	DC Motor
1	Digital Line Sensor

9g Servo

These servos are used to set the path of the ball for the selection of different sections. The servos are mounted to the bottom of the splitter using the built-in mounting points and a custom 3d printed mount and operate splitter paddles to direct the ball to the different pathways. The servos are controlled using a PWM signal generator on the FPGA. As per the specifications of 9g servos, these signals are a PWM signal with a period of 20ms and an active time between 1ms and 2ms.

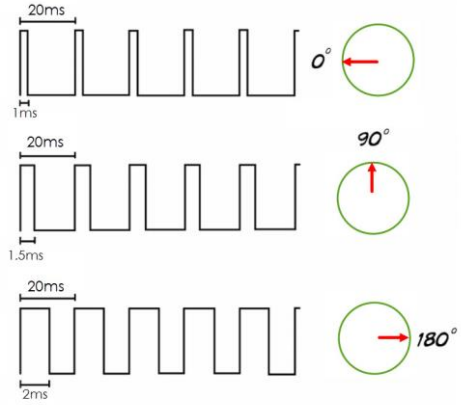


Figure 122 Servo PWM Chart [1]

Push Buttons

The push buttons are used to set the section order for the machine. These buttons do not affect the machine whilst operating and instead are usable prior to entering the ball into the machine to select the desired order for the sections. These buttons are integrated as digital sensors connected to a GPIO pin on the FPGA board and when triggered, the FPGA interprets them in order to send a serial signal to the Arduino.

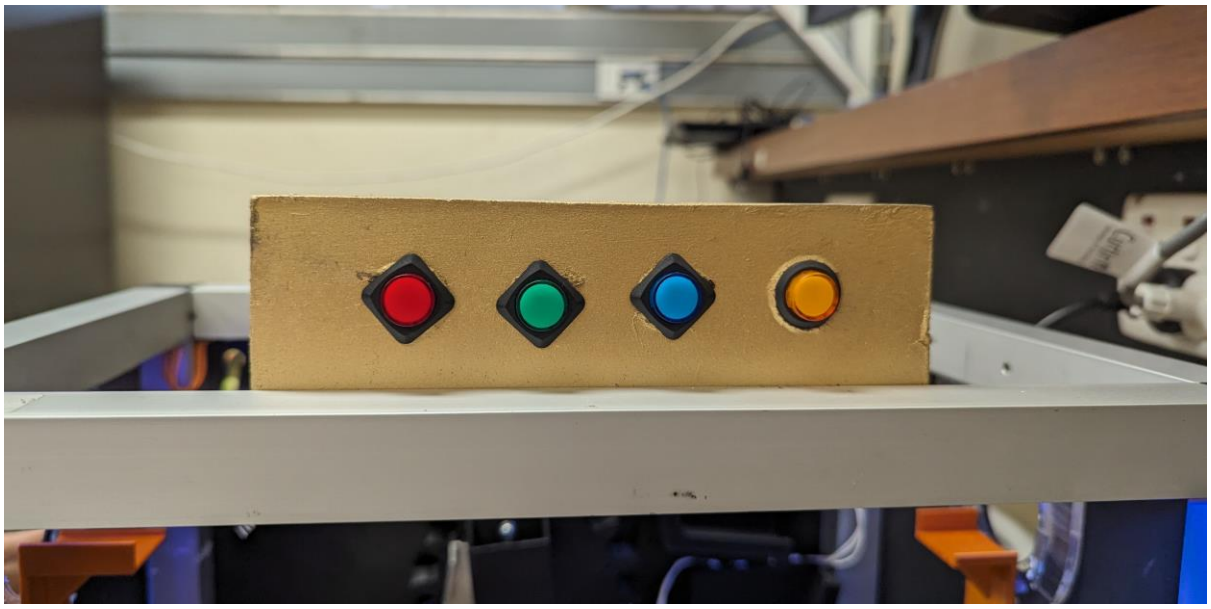


Figure 133 Button Assembly on Project Box

DC Motor

The DC motor used in the design is a small form factor motor which matches the one used in the JDO Solutions screw lift design. The motor is driven by 3-12V and features a maximum no-load RPM of 200 RPM at 12V. This motor was chosen because it matches the size profile of an existing 3d model which reduces the complexity of the design. Though there is a similar motor with a lower gear ratio offering higher torque with a maximum of 100 RPM, the higher RPM available to this motor is more desirable as it allows for the screw lift to be completed faster affording better control over the timing of the machine.

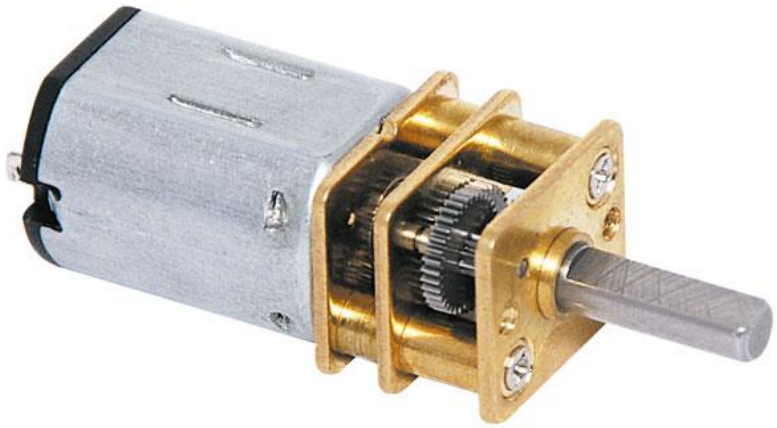


Figure 1414 Micro N20 Geared Motor 150:1 Gear Ratio [1]

MOSFET Switching Board

As the FPGA is only capable of delivering 5V and minimal current through the logic pin of the FPGA, this board is used to facilitate PWM control of the DC motor which requires 3V-12V input. This also allows the 12V power supply to be downscaled using PWM in order to vary the speed of the DC motor used for the screw lift. This can be used to fine tune timing of the machine. Figure 15 below indicates the pinout required to operate the MOSFET switching

board. The MOSFET switching board is capable of switching up to 36V supply voltage and 30A current. As the motor operates well below these conditions, the MOSFET switching board is suitable for the intended use case.

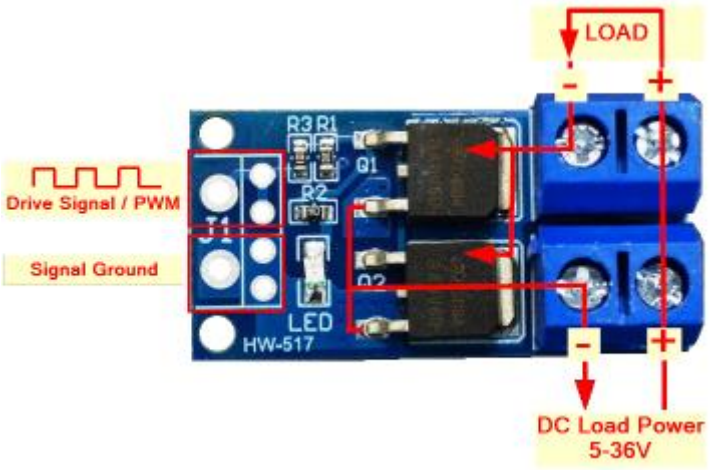


Figure 155: MOSFET Switching Board PWM Diagram [2]

Digital Line Sensor

The digital line sensor is mounted at the base of the screw lift and serves to track when the ball has completed a loop. This sensor is connected to the FPGA digital input and the trigger signal is sent by the FPGA to the Arduino in the form of a UART byte to act as a trigger for switching the servo position. This allows the Arduino to know when the ball has completed a circuit so that the next servo position byte can be sent.

3D Printed Parts

When constructing the parts for the ball splitter section of the machine a high degree of reliability is required. As such, it is advisable for the splitter to be made from 3D printed parts. These parts were designed modularly to allow for adjustments in the positions of the entry points for each section of the machine. In the section below, the designs for the 3D printed parts have been included, the models for the splitter were all created custom for this project and the screw lift models were adapted from the screw lift design by JDO Solutions [2]. While initial designs for these sections of the machine featured recycled parts, it was found during testing that using recycled materials for these sections was too unreliable and would inordinately increase testing time. As such all sections of the screw lift and splitter are primarily 3d printed, with some recycled materials used to strengthen or decorate the sections.

Table 4 Splitter and Screw Lift Required 3D Printed Parts

Section	Part	Quantity
Splitter	Proof of Concept 1	1

Screw Lift	Proof of Concept 2 (Main Splitter Body)	1
	Station 1/2 Turn	2
	Station 2 Straight	1
	Station 3 Straight	1
	Station 3 Adaptor	1
	Station 3/Exit Splitter	1
	Station 3 Adaptor 2	1
	Station 3 U-Turn	1
	Exit Straight	1
	Splitter Paddle	4
	Splitter Mounting Brackets	2
	Screw Lift Base	1
	Motor Adaptor	1
	Base Screw (motor adaptor compatible)	1
	Screw Segment	5
	Screw Top Segment	1
	Lift Rail	1
	Lift Exit	1
	Lift Top Cap	1
	Connector Pins	6
	Lift-Splitter Entry Channel	1

4.2.2. Software Specifications

This section of the machine will require communications between the Arduino and the FPGA board. The communication process will require the transfer of status codes to alert the microcontroller to the states of the FPGA and transfer instructions back to the FPGA. The required transmissions are:

Table 5 Serial UART Codes for the Marble Splitter

No.	Direction	Description	Byte value [Binary]
1	To Arduino	Button 1 pressed	65 [01000001]
2	To Arduino	Button 2 pressed	66 [01000010]
3	To Arduino	Button 3 pressed	67 [01000011]
4	To Arduino	Button 4 pressed	68 [01000100]
5	To Arduino	Ball return digital line sensor triggered	69 [01000101]
6	To FPGA	Move servos to connect section 1	70 [01000110]
7	To FPGA	Move servos to connect section 2	71 [01000111]
8	To FPGA	Move servos to connect section 3	72 [01001000]
9	To FPGA	Move servos to connect section 4	73 [01001001]
10	To FPGA	Move servos to connect exit path	74 [01001010]

The Arduino microcontroller will serve as the memory for the machine, tracking the order the buttons are pressed and storing that information, as well as directing the state of the crazy

machine when each section is completed. The FPGA will send a byte each time the line sensor is triggered and will react to the response containing next section information from the Arduino microcontroller. This process is outlined in the block diagram below.

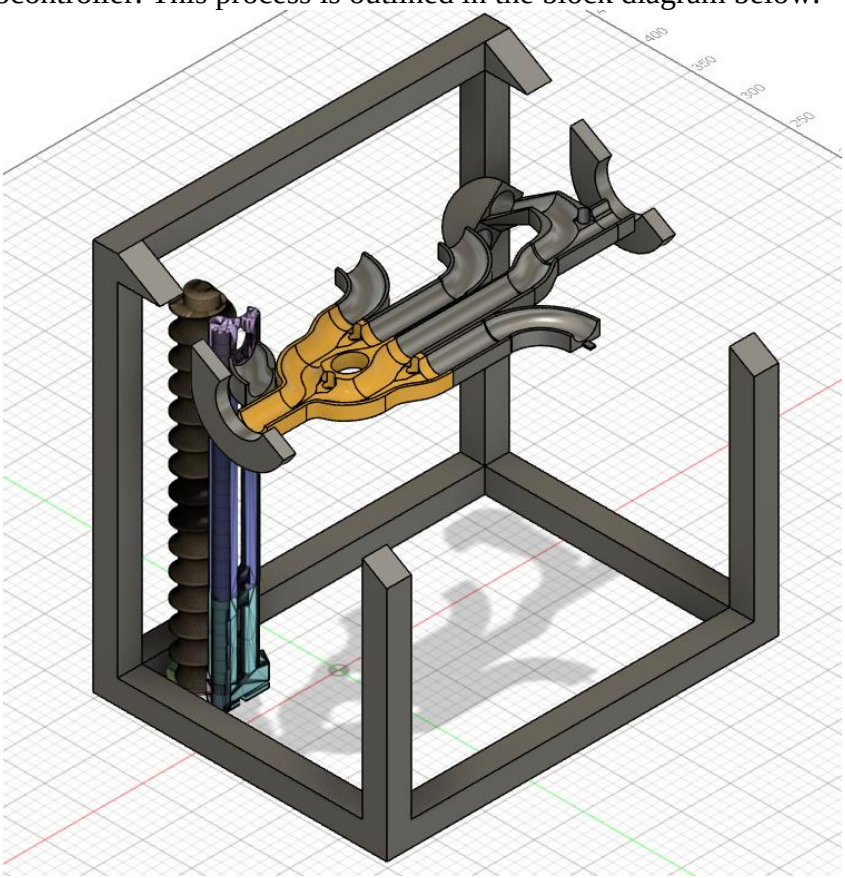


Figure 1616: 3d CAD Model of Splitter and Screw Lift

The first step of the design for the marble splitter was to determine the design requirements. As a group we determine that there would be 4 sections, adding in the exit results in a 1-5 splitter being required. Early on it was assumed that the box entry and the lift return point would combine prior to entering the splitter. This substantially decreases the complexity of the design as it is far easier to route 1-5 than 2-5. Additionally, in the early stages of the project it was determined that the splitter would effectively take up the entire width of the box at the ball entry and return level. By planning the other stations around the splitter space this greatly reduces the complexity of designing the splitter paths. The final design requirement was to maximise the modularity of the design. This would allow for changes throughout the design with minimal impact on the project timeline. Additionally, this would allow the other sections to be designed more effectively as they are not significantly limited by the entry point into that section.

Splitter Initial Design

The initial splitter design consisted of four identical splitter modules which could be connected at different points to allow for five different pathways. While this design, seen in

Figure 17 below, would allow for maximum modularity, it relied on a solenoid to drive both the second and third splitters simultaneously. Whilst this reduces the number of actuators required, it does increase the complexity of the machine.

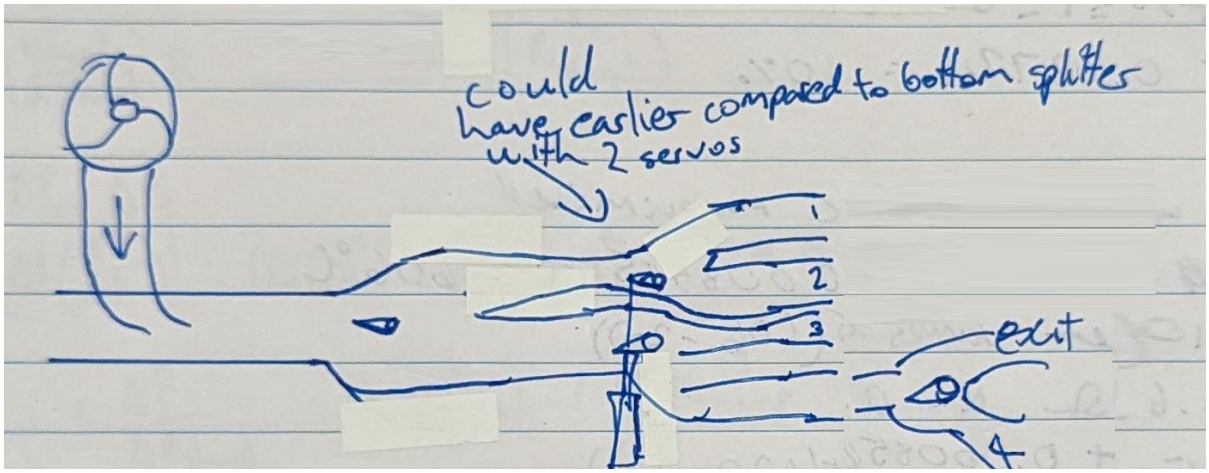


Figure 177 Initial Splitter Design

After investigating the datasheet and testing the solenoid provided by Curtin, it was determined that utilising a pull-type solenoid with an effective throw of less than 4.5mm [4] was impractical as it would require a complex linkage to operate both splitter paddles simultaneously. Instead, a design featuring four servos, with one operating each splitter was chosen.

Splitter Proof of Concept 1

After the requirements of the splitter were determined, a 3d model was created and printed to serve as a proof of concept for the track and splitter design. This proof of concept, in Figure 18 below, was formed by generating a 30mm diameter pipe and applying the union-cut modifier with the general shape of the splitter to form a channel for the marble to follow. The objectives of this proof of concept were to test whether the pipe-cut method was effective at creating a channel for the ball and whether the splitter was sufficient to change the path of the ball. To determine this, the ball was to be run through the design and its path noted. A successful test would require the ball to remain in the central channel without significant lateral movement and successfully change path when the paddle was placed at a 15° angle from the centre line.

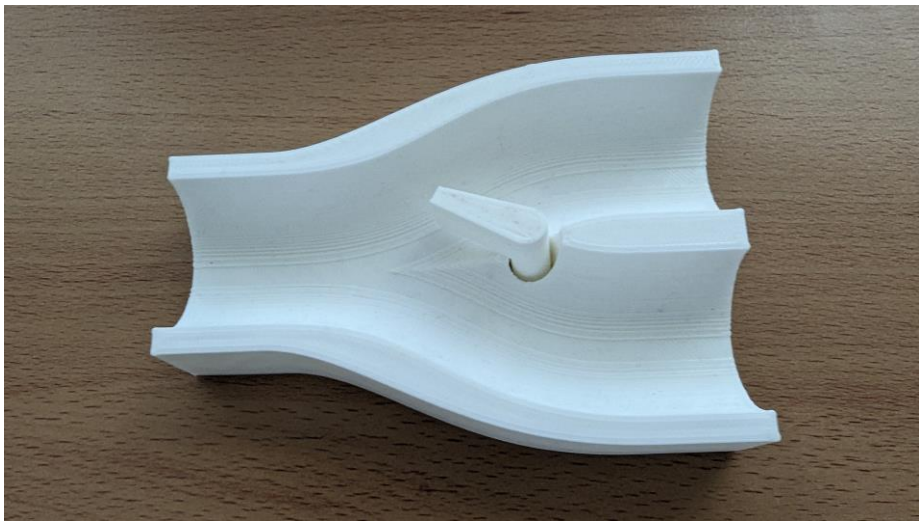


Figure 188 First Splitter Proof of Concept

The proof of concept was a resounding success. The ball effectively followed the path and the paddle design worked as expected. However, the ball experienced a significant amount of lateral movement when some lateral force was implied on the ball on release. In the final machine this would be caused by the screw lift entry point as well as possibly by the other splitters. This lateral freedom will be resolved in future designs by reducing the diameter of the pipe used to cut the ball channel from 30mm to 25mm which should restrict the ball to a straighter path. An additional characteristic of the proof of concept is the ridges formed by the step limitations while 3d printing. While initially seen as a potential hazard, these ridges served to impede the lateral movement of the ball on turns by increasing the difficulty to move up the walls of the channel. This is useful in maintaining predictability of the ball and hence will be kept in future versions of the design.
In summary, the results of the proof of concept are:

- A channel diameter of 25mm offers more control over the ball.
- The splitter and paddle design is effective at controlling the path of the ball.
- The ridges formed by layer height restrictions while 3d printing assist in control of the ball when changing direction.

Redesign of the Splitter

With this information, the new design of the marble tracks was determined. This version features the same design method, however, has a 25mm marble track rather than 30mm. As a result, the wall height is slightly reduced as to remain below the 50% depth of the track cutout. This allows for the marble to be inserted and removed from any section of the track which is required for the screw lift entry. While this in theory allows for easier escape of the ball, in the first proof of concept which featured proportionally similar wall heights it was found that it is difficult for the ball to derail with its own momentum alone.

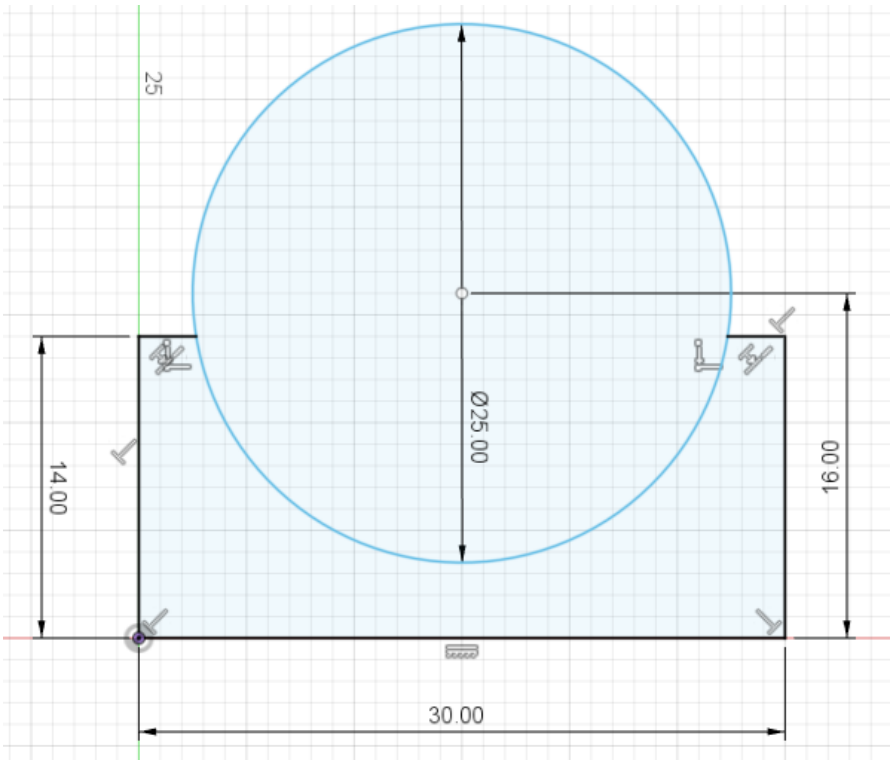


Figure 199 Dimensions of the Marble Channel

Additionally, during this redesign, a system of tabs and cutouts was created with the aim of reducing the difficulty of assembly by allowing for additional surface area while glueing and providing a keyed connection for easier alignment. As shown in Figure , the dimensions of the key were chosen by the depth and thickness limitations of the track. Importantly the key side was made smaller than the socket to allow for some adjustment during assembly. To maximise modularity a consistent key system is used, so that sections of the splitter can be rearranged if required. This ended up being useful during assembly as an unneeded section could be reused instead of reprinted.

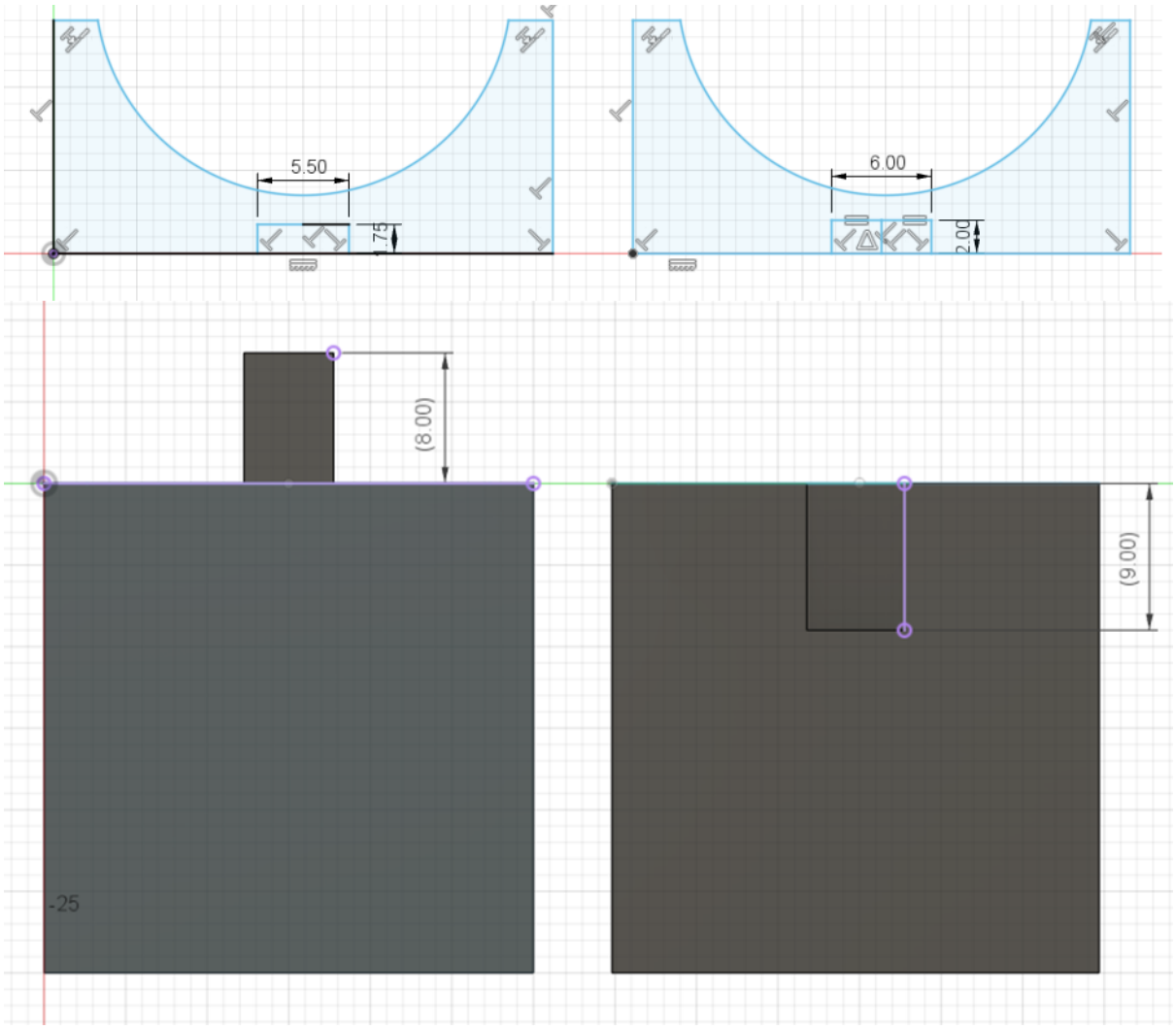


Figure 20 Dimensions of the Key and Socket System

Splitter Proof of Concept 2

Using these new dimensions a new version of the splitter was created. Serving as proof of concept two this design featured the narrower channels, keyed system, and comprised of splitters 1, 2 and 3 combined into a single piece. During the design stage of this section, it was found that to accommodate the position of the Ferris Wheel section, the distance between splitters 1 and 2 was negligible. To maximise the structural integrity of this sections, it was thus decided to combine splitters 1, 2 and 3 into a single piece. As seen in Figure 2120 below, this results in a large assembly that can serve as the main body of the splitter section. This is important as it will form a base for the structure of the splitter, with the other modular segments connecting to this section.

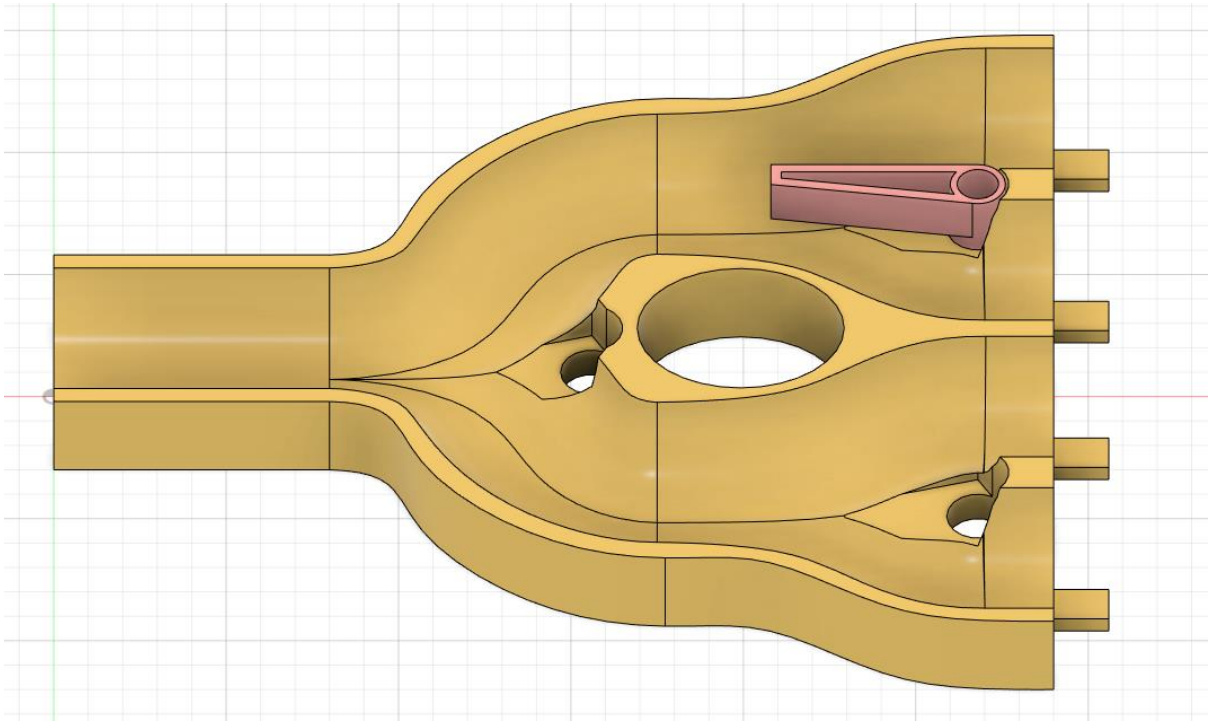


Figure 2120 3D Model of Splitter Proof of Concept 2

Proof of Concept 2, shown in Figure 2120 and Figure 21, is intended to test the capabilities of the splitter design and to assess the validity of the dimensions derived from proof of concept 1. As such the objectives of this test are to validate the 25mm channel diameter, test the new design is functional and reliable, and to test the splitter and paddle design is effective. This proof of concept will also be used to test the servo VHDL code that will be incorporated into the final design. To validate the proof of concept, the splitter will be positioned with an angle of depression of 5°, the ball will be run through the splitter similarly to proof of concept 1, with several different velocities and varying degrees of lateral force. The test will be considered successful if the ball can travel down the appropriate path in the majority of tests, without leaving the channel.

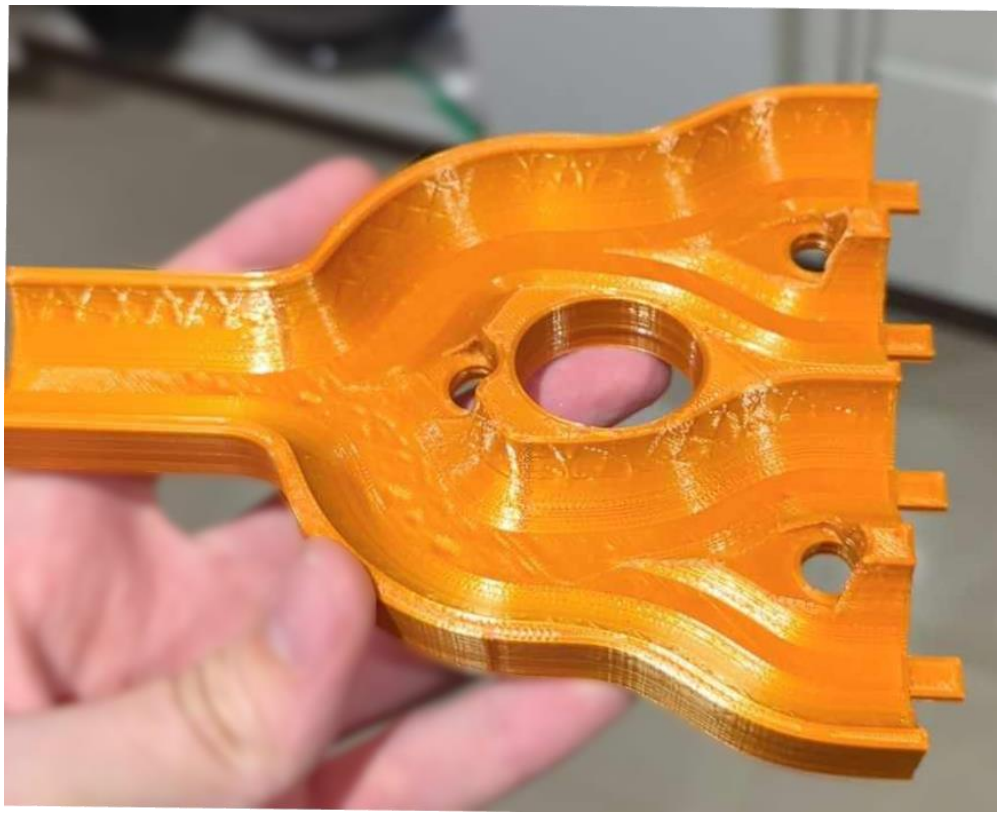


Figure 212 Proof of Concept 2

The results of the testing in proof of concept 2 were that the ball was successful in all trials at passing through the splitter, importantly during this trial the ball withstood relatively high lateral movement provided the ball remained in the channel initially. This highlights the need for some form of guard or barrier to prevent the ball from immediately leaving the splitter as the ball enters from the screw lift. This was deemed sufficient for the applicable use case as a barricade to prevent the ball leaving the track was already expected to be required at several different points in the machine. Additionally, it was found that due to an oversight in the design, the paddles used for directing the ball collided with the track and obstructed the movement of the ball for splitter 1 only, this is due to the relatively sharp split angle and did not affect the other two splitters. To resolve this issue a cut was made to the splitter 1 paddle in order to allow for full clearance without requiring reprinting of the splitter paddles.

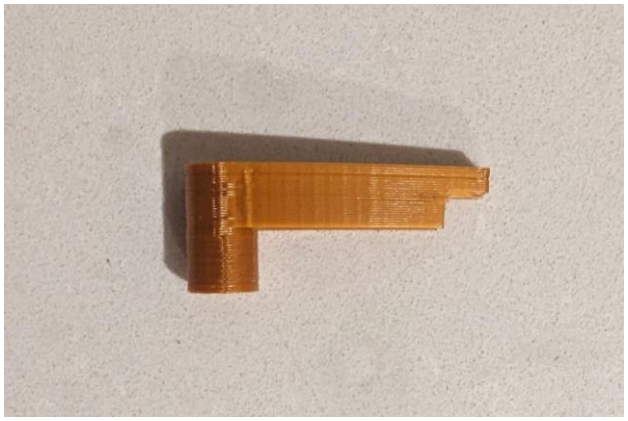


Figure 223 Cut made to Splitter Paddle in Proof of Concept 2

Overall, the proof of concept 2 was a resounding success and as such it is a suitable design for the main section of the splitter. Additionally, this test demonstrated that the proposed channel dimensions with a diameter of 25mm are appropriate for the paths for the rest of the design.

Splitter Assembly

Following proof of concept 2, the designs for the rest of the parts were created and each part validated in the 3d model to ensure the overall design was sized consistently. As part of this design, shown in Figure 1616, a number of 3d printed parts were required, a full list of the parts for the splitter and screw lift is included in Table 4. Serving as a final test of the design, the requirements for this fit-up are that the parts join together, providing a contiguous path for the ball, and that the assembled splitter fits in the project box without significant adjustment. The parts were printed and then assembled according to the design. During assembly it was found that the straight station pieces did not have sufficient clearance to be mounted effectively. To resolve this issue, a small file was used to remove material from the keys as well as the socket of the station 2 straight. Following this adjustment the pieces fit sufficiently to assemble the design. Additionally, when assembled there was a small lip between the main splitter and the station 2 and 3 straights. This interface also required sanding to smoothen the transition between these pieces. Additionally, during transportation of the splitter pieces, it was found that the glued faces were not sufficient structurally to transport the splitter assembly. To strengthen the assembly, pieces of clear plastic were glued to strengthen the joints of the splitter assembly. This was successful and significantly strengthened the parts during transportation and test-fitting in the project box. During the box assembly check, it was found that the station 3 U-turn had some interference with the station 2 Ladder Rungs. This was due to unforeseen space limitations and to resolve this issue, the bottom corner of the station 3 U-turn was sanded to remove excess material. Finally, there was an interference between the exit splitter servo and the station 3 Piano Keys, this was unforeseen and required removing excess material from the Piano Keys mounting plate. Following these changes, the splitter was deemed structurally sound and fit in the box in its intended position.

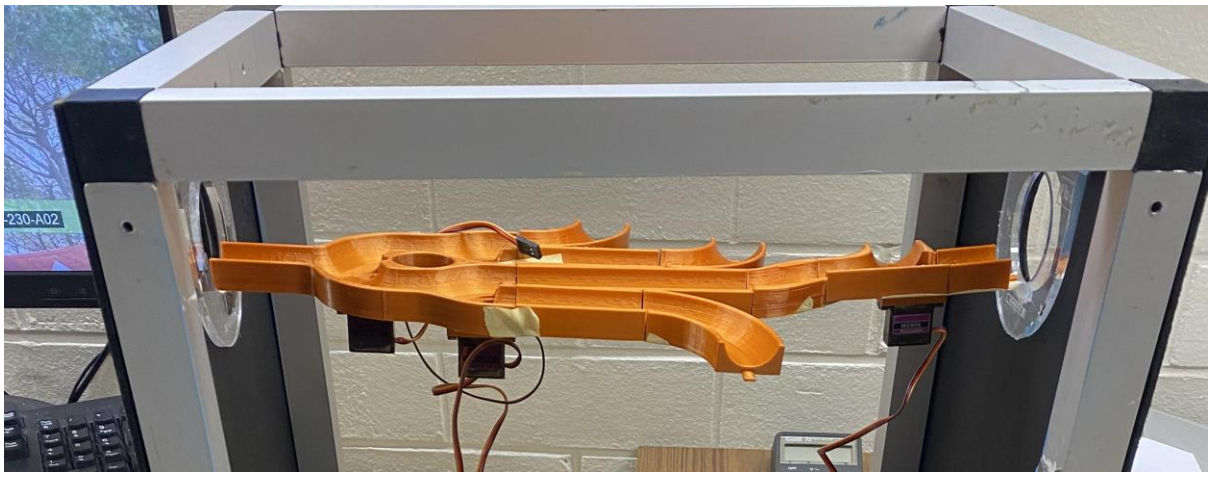


Figure 234 Splitter Assembly in Box Prior to Glueing

Splitter Mounting Brackets

The splitter is intended to have a slight angle of declination across the length of the box, this is to ensure the ball always has sufficient momentum to travel from one end of the box to the other. Accurately positioning the splitter is difficult as small variances in the box and construction can lead to issues. In order to combat this the decision was made to design adjustable mounting brackets for the splitter in order to allow for adjustment of the position after assembly. This is to be achieved using slots instead of boltholes for mounting to the box. This design is relatively common and allows for a high degree of vertical movement, ideal for getting the correct angle of declination. Additionally due to incorporating some additional space into the screw slots, it is also possible to achieve some degree of horizontal movement. This allows for an adjustable system where the splitter can be adjusted to vary the entry and exit heights.

The design progress for the mounting brackets began with a proof of concept consisting of a simple panel with slots and a horizontal rest for the splitter. The goals for this mounting bracket test were to find a design which allows the splitter to slot into the top rest, and prove the effectiveness of the slot mounting point design. After the proof of concept was printed, the slots were tested to ensure that the bolts would fit and that the bolts would slide effectively and lock into place when tightened. This proved successful proving the slots were appropriate for adjustability and the splitter effectively fit into the design without much lateral play. When the brackets were transported to Curtin for testing, the brackets broke due to the lack of support between the 90° sections. This was accommodated in the second design which featured a rib between the panels in order to strengthen the joint. This proved successful as the second iteration did not break during transport and provided better support. When testing the brackets inside the box, it was found that the design neglects to accommodate the thickness of the acrylic entry point and hence could not mount properly to the box. This was incorporated into the final design which features an elongated bracket with an extra spacer to account for the thickness of the acrylic. Additionally, this revision of the mounting bracket features an extra high right wall on the mounting bracket, this allows for additional area for glueing to make attaching the screw lift barricade easier. When tested in

the box, this version of the bracket had sufficient clearance to fit flush to the wall and provide a stable and adjustable mounting point for the splitter to rest on.

Screw Lift

The first step of the design process for the screw lift was looking at similar designs online. Of these designs the most applicable to our project was the one created by JDO Solutions and licenced under the Creative Commons Attribution-NonCommercial Licence [3]. This marble lift featured a simple 3d printed design and was designed for a motor available at Altronics. Initially the plan was to scale the design to suit 20mm balls and 3d print just the screw. This was important as the pitch of the screw is very important to the efficacy of the screw. The support frame and exits in the initial design were to be made from recycled materials.

Screw Lift Proof of Concept 1

With this in mind, the first proof of concept consists of a 3d printed screw section. This section will be used to test whether the scaling of the JDO solutions design is a valid approach for the screw dimensions. Additionally, by holding the ball in place the screw action will be tested. A successful test will be categorised by the ball effectively fitting in the screw track and the 3d printed screw successfully executing the screw action when tested. This test was performed using both one and two screw sections joined together and while there were some burrs caused by insufficient support while printing, both configurations satisfied both criteria of testing.

Figure 245 Screw Lift Proof of Concept 1

Screw Lift Proof of Concept 2

Based on the success of these tests, a follow-up test was performed using a cut tube can as the lift rails to hold the ball in place as the screw turns. This test is intended to assess the

complexity of using recycled materials for this portion of the design. The results of this test will be based on the effectiveness of using a cut tube for this portion of the test and will be assessed based on the ease of alignment and whether the tube will have sufficient space to house the turning mechanism.
This test was performed using two screw segments and highlighted the drawbacks of using recycled materials for such a precise mechanism. During the test it was observed that it was very difficult to correctly line up the screw with correctly to allow the mechanism to work. If this version of the design were chosen it would likely require many prototypes to correctly position the screw. Due to the size of the motor mount required and the number of prototypes it was determined that any material savings due to recycled materials would be dwarfed by the additional material used in prototyping. Additionally, as the JDO Solutions design is relatively material efficient, using 3d printed rails was determined to be the more efficient solution. As such, moving forward the screw lift was to be made entirely from 3d printed sections and modifications made where required to adapt the JDO Solutions design to the design requirements of this section.

Adjusted Screw Lift Design

Following the results of Proof of Concept 2 the design will be adapted to utilise 3d printed parts to fulfil the role of the rails of the screw lift. Additionally due to the JDO Solutions being scaled for 0.5” balls compared to the 20mm balls of this project some changes will need to be made to better suit this use case. Firstly, the JDO solution features three sets of rails to allow for three channels for the ball to ascend the screw lift. However, this results in the design being approximately 110mm in width and diameter once scaled to the 20mm ball size. This is too big for the accommodations made for the screw lift in our design. As such, the lift will only feature one set of rails and trim the model as much as possible to reduce the size. Whilst this reduces the stability of the design, by mounting the lift at the top and bottom of the machine this should minimise the negative effect on stability caused by reducing the number of rails. This results in a final size of approximately 75mm for length and width which is far closer to the initial project estimate for the lift.
As the motor size has not changed in the design but the screw has scaled up, accommodations must be made in the design to interface the increased screw diameter with the motor mount in the design. In order to achieve this the existing design for the base was split into two sections, the external rail interface, and the internal motor interface. Then, in order to ensure compatibility, the external rail interface was scaled up to 20mm dimensions whilst the internal motor interface was left as is. These two sections were then rejoined in order to maintain the structure of the design. This process was then repeated for the bottom screw section which has a triangular connection point for the motor.

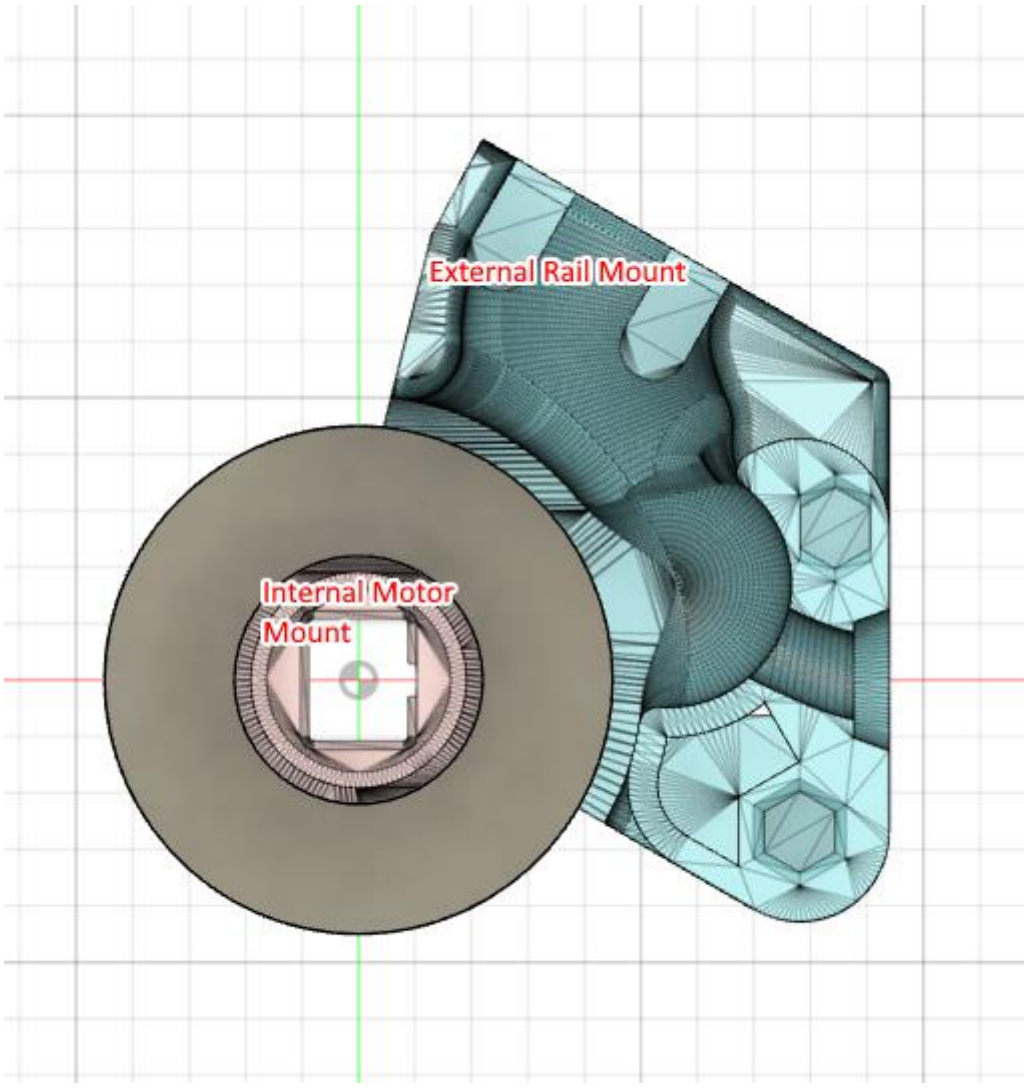


Figure 256 Modifications Made to the Screw Lift Base

This design was then tested with the motor to ensure that the screw effectively carried the ball and that the parts did not interfere with one another. This test was a success, with the ball successfully carried to the top of the screw. With this test demonstrating the success of these parts, the decision was made to continue with the 3d printed approach. Following this the remaining parts outlined in Table 4.

Screw Lift Final Assembly

With the parts printed, the screw lift is to be assembled as per the design. Once assembled the screw lift will be tested at 5V and 12V with the goal of verifying effective and consistent motion of the screw. During this test the screw will carry the ball from the entrance point to the exit which will be at the height used in the final machine. The time required for the ball to reach the top of the lift will be recorded for motor voltages of 5V, 8.5V and 12V to ensure the lift will meet the time requirements of the machine. A successful test will consist of the ball

consistently reaching the top of the machine in a consistent time, with the minimum time of approximately 5 seconds. The results of that experiment are presented in Table 6 below.

Table 6 Screw Lift Times at Specified Motor Voltages

Test No.	5V	8.5V	12V
1	20.4	11.4	8.3
2	20.6	11.5	8.4
3	20.2	11.7	8.2

During the test, the ball completed the screw lift in all attempts, with the times for each voltage being relatively consistent with a standard deviation of 0.16, 0.12 and 0.08 respectively. This indicates that the screw lift is both reliable and consistent and hence minimal modification is required for the screw lift to perform appropriately. During this test it was found that there was some motion in the screw lift due to some play in the parts. As some sections of the screw were not glued together during the test in order to allow the lift to be changed if needed, this play can likely be resolved by glueing the remaining sections.

Figure 27 Screw Lift Final Assembly

Screw Lift Final Assembly

Before glueing the remaining sections of the screw lift, a final test was conducted to ensure the screw lift fit in the project box effectively, received balls from the ball return and properly fed into the ball splitter. This test was conducted in the project box and the criteria for this test was that balls reliably enter and exit the lift without getting stuck. The expected feedback from this test is to determine how large the barricade on the marble splitter needs to be and

what modifications need to be made to the ball return to effectively allow the return of the ball. This test showed initial success with the screw lift fitting in the box as intended and positioned perfectly to feed into the splitter without requiring modification. When performing the marble test, it was found that only a relatively small barricade was required to keep the ball feeding into the splitter. However, as expected, there was some modifications required for the ball return section as feeding into the screw lift was somewhat unreliable. Following adjustments to the ball return, the return effectively fed into the screw lift which reliably delivered the ball to the splitter.

4.3. Arduino Software Specification and FPGA Design

4.3.1. Arduino Software

4.3.1.1. Arduino Proof of Concept 1

The Arduino control code serves to store the order of stations to operate, manage the feedback to the FPGA when it detects section changes and forms a backbone for the sensors and actuators for other sections that run from the Arduino. The following descriptions will refer only to the portions of the Arduino controller code that are related to the screw lift and marble splitter.

The initial proof of concept of the Arduino control code was developed early into the development of the machine and aims to detail the process by which the station order is stored, accessed and changed by the machine. The initial proof of concept intended to:

- Create the data structure to store the station order.
- Randomise the order on startup.
- Set the order based on different received serial byte.
- Step through the order when a specific serial byte is received.

The proof of concept will be validated on its ability to provide this functionality and failure to fulfill these functionality requirements will result in continual redevelopment until such requirements are fulfilled.
The initial code created for the proof of concept consists of a 5-element array which stores the desired order for stations.

```
1. int stations[5] = {1, 2, 3, 4, 5}; // exit is station 5
```

The first four elements of the array are then randomised on startup always leaving the exit as the final station. These stations are then printed to the serial monitor for debugging purposes.

```
1. for (int i = 0; i < 4; i++) {
2.     int index = random(0, 4);
3.     int temp = stations[i];
4.     stations[i] = stations[index];
```

```
5.      stations[index] = temp;
6.    }
7.    print_array(stations);
```

The program then initialises the following state variables as integers. The station_count variable is used to track what position in the order the selection process is up to. The set variables store whether the corresponding station has already been set in the order. And the stations_completed variable stores how far the machine is into running the sequence of stations.

```
1. station_count = 0;
2. one_set = 0;
3. two_set = 0;
4. three_set = 0;
5. four_set = 0;
6. stations_completed = 0;
7. station_sound_play = 0;
```

The program then listens to the serial connection for a byte corresponding with the different machine sections. If the byte is correct, the program sets the correct element of the array to the corresponding station and marks the station as set. Once all stations are set, the station_count is equal to 4, the code prints the now ordered set of stations and triggers the code to send the first station as it would in the final version. It is at this point that it also resets the variables to the machine is ready to start once again.

```
 1. void loop() {
 2.   int signal = Serial.read();
 3.   if (signal == 65) {
 4.     // Station 1
 5.     replace(1, station_count);
 6.     station_count++;
 7.     one_set = 1;
 8.     delay(100);
 9.   }
10.   ...
11.   if (station_count == 4) {
12.     print_array(stations);
13.     //send byte for first station
14.     send_station(stations);
15.     station_count = 0;
16.     one_set = 0;
17.     two_set = 0;
18.     three_set = 0;
19.     four_set = 0;
20.     stations_completed = 0;
21.   }
22.   if (signal == 69) {
23.     // line sensor trigger
24.     stations_completed++;
25.     Serial.println("Station completed sending next");
26.     send_station(stations);
27.     delay(1000);
28.   }
29. }
```

Finally, the code also listens for a specific byte corresponding to the line sensor in the final implementation. When this byte is received, the program will send the byte corresponding to the next station in the order over serial.

For testing during the proof of concept, the serial monitor is used instead of the FPGA. This allows for greater control over the information sent and received and mitigates any potential issues stemming from implementation errors on the FPGA. This test was uploaded to the Arduino microcontroller and thoroughly tested. The proof-of-concept code was found to

fulfill all of the requirements for this section of the code and proved to reliably be able to order the stations. Due to the relatively simple data structure and program logic, the implementation was straight forward and did not have any significant issues. As such this test was a success and this code can be used in the future Arduino controller code.

4.3.2. FPGA Design

4.3.2.1. UART Transmission

The UART transmission code is split into 2 programs, UART_RX and UART_TX. RX is the receiving code where the FPGA reads incoming bits from the Arduino, and TX is the transmission code where the FPGA sends outgoing bits to the Arduino. The FPGA and Arduino are connected via 2 cables, each boards RX pin connected to the opposite boards TX pin and a common ground connection between them.

UART is an asynchronous transmission protocol, which means that there is no requirement for the two boards to be synchronised and instead works by initially idling in a ‘high state’ while waiting for a falling edge, which indicates the start bit. This is then followed by 8 bits sent one by one which the other device receives, and then finally a stop bit is sent to notify that the transmission has finished. In order to ensure these bits are set for the correct duration of time, a consistent Baud rate (the rate information is transferred at) is set between both devices and the correct frequency the FPGA is running at is set in the code. On the FPGA side this frequency is divided by the baud rate, which returns the amount of clock cycles required per bit.

4.3.2.1.1. UART TX

Code Description

The transmission side works by using a state machine, with 4 states. As the Arduino has no parity bit, this was not needed. Baud rate was chosen at 9600. These are:

1. Idle - where the TX line is driven high to indicate idle.
2. Start bit - where the TX line is driven low to indicate the start bit.
3. Data bits - where the bits are sent one by one and stop bit; where the TX line is driven high to indicate the start bit.
4. Stop bit.

In the idle state, TX line is set high and is constantly being checked to see if there is a request to start the transmission. Once this request is received the TX line data to be sent is saved to the variable ‘data’ and the state is changed to the next stage. If this request is not met, then the state is kept in idle.

In the start bit stage, the TX line is set low to send a start bit. This low is set for 5208 clock cycles, which is calculated by dividing the frequency (50Mhz) by the baud rate (9600). Once the TX line has been kept low for the correct duration the next state is set.

In the data bits stage, the first bit is sent out to the TX line from the data stored in the array. This data is set for 5208 clock cycles, where the data sent out on the TX line is changed to the next bit in the data array. This is done until the final bit in the data array is sent, where the state is then changed to the next state.

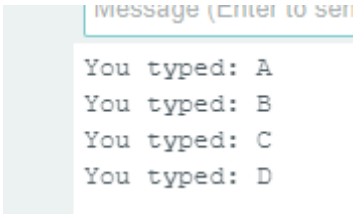
The stop bit stage works the exact same way as the start bit stage, with the only difference been that the TX line is set to inactive, and the transmission is set as done. Once completed the state is changed back to the original idle state.

Testing and Debugging

This module was tested using the serial monitor of Arduino IDE by connecting an Arduino's RX pin to the FPGA via GPIO and using the debounced push button signal that was connected to the GPIO pin of the FPGA.

```
 1. void setup() {
 2.   Serial.begin(9600);
 3. }
 4.
 5. void loop() {
 6.   while (Serial.available() > 0) {
 7.     char incomingByte = Serial.read();
 8.
 9.     incomingMessage += incomingByte;
10.
11.     if (incomingByte == '\n') {
12.
13.       Serial.print("You typed: ");
14.       Serial.print(incomingMessage);
15.       incomingMessage = "";
16.     }
17.   }
18. }
19.
```

```
ascii_table(0) <= "01000001"; --A
ascii_table(1) <= "01000010"; --B
ascii_table(2) <= "01000011"; --C
ascii_table(3) <= "01000100"; --D
```



As the TX out signal was a buffer of its input, the buttons were programmed to only write to the TX module data input once, upon a rising edge, preventing multiple transmissions of the button signal when the button was pressed.

```
 1. FOR i IN 0 TO 3 loop
 2.                 btn_sync(i,0) <= BTN_O(i);
 3.                 btn_sync(i,1) <= btn_sync(i,0);
 4.                 btn_pulse(i)   <= not btn_sync(i,1) and btn_sync(i,0);     --is 1 when rising
edge of a button

 5.
 6.       if btn_pulse(i) = '1' and active_TX = '0' then
 7.                 dataValid_TX <= '1';
 8.                 data_TX <= ascii_table(i);
 9.       end if;
10. end loop;
11.
```

4.3.2.1.2. UART RX

Code Description

The receiver side works similarly, by using a state machine with four states. The baud rate is set to 9600, and the clock frequency is 50MHz.

In the idle state, the receiver waits for the start bit (a low signal) on the input serial data line. If the start bit is detected, it transitions to the start bit state.

In the start bit state, the receiver checks if the start bit is still low at the middle of the bit period. If the start bit is still low, it transitions to the data bits state. If the start bit is not low (a false read), it goes back to the idle state.

In the data bits state, the receiver reads each bit of the data frame by sampling the input serial data line at the middle of each bit period. It stores the received bits in a temporary storage and keeps track of the bit index. After reading all 8 data bits, it transitions to the stop bit state.

In the stop bit state, the receiver waits for the stop bit (a high signal) at the middle of the bit period. If the stop bit is detected, it sets a flag indicating that the data frame has been received successfully. It then transitions back to the idle state.

Testing and Debugging

The code was tested by mapping the pinouts of the onboard LEDs of the FPGA to the buffer output of the RX code. An Arduino software was then created to "loopback" any signal that was typed into the serial monitor of the Arduino IDE, data can be transmitted to the FPGA.

```
#define BAUD_RATE 9600

void setup() {
  Serial.begin(BAUD_RATE);
}

void loop() {
  if (Serial.available() > 0) {
    char incomingByte = Serial.read();
    Serial.print(incomingByte);
  }
}

```

Upon testing, it was decided to implement a register that will set the RX data values from the buffer to the output signal if the signal matches one of the signals from a predefined table of the table of values that we are "listening" for. This increased the reliability of the RX connection and meant that a TX signal from the Arduino only needed to be sent once, and the TX component will "save" this value in the register.

```
	ascii_table(0) <= "00000001"; --PATH1
	ascii_table(1) <= "00000010"; --PATH2
	ascii_table(2) <= "00000100"; --PATH3
	ascii_table(3) <= "00001000"; --PATH4
	ascii_table(4) <= "00010000"; --PATH5

	output_register : process (CLK)
	begin
		for i in 0 to 4 loop
			if RX_DATA_buffer = ascii_table(i) then
				RX_DATA <= ascii_table(i);
			end if;
		end loop;
	end process;

```

Integrating UART Serial to the Servo Controller

To integrate the servo controller with the UART module, the servo controller needed to be changed in several ways. The first change was to allow control of 4 servos simultaneously instead of just one. This can be achieved by splitting the servo_pos variable and servo output to vectors. Additionally, the output setting code must also be converted to a for loop to enable checking of all four servos easily.

```
variable servo_pos : position_servo;
servo     : out std_logic_vector(3 downto 0)
servo_pos(0) := 1;
servo_pos(1) := 1;
servo_pos(2) := 0;
servo_pos(3) := 0;
for i in 0 to 3 loop
```

```
 8.       if servo_pos(i) = 0 then
 9.                 if cnt >= duty_count_0 then
10.                           servo(i) <= '0';
11.                 else
12.                           servo(i) <= '1';
13.                 end if;
14.       elsif servo_pos(i) = 1 then
15.                 if cnt >= duty_count_1 then
16.                           servo(i) <= '0';
17.                 else
18.                           servo(i) <= '1';
19.                 end if;
20.       end if;
21. end loop;
```

The second change required is to convert the toggle switch trigger to a multi-selection control structure. This change allows the code to read any byte-like data and compare that to a known value. This can be used to program the multiple servos to change to certain positions when a byte-like variable is set to a specific value.

```
1. if logic_byte = "00000001" then
2.        -- Path 1
3.        servo_pos(0) := 1; --servo 0
4.        servo_pos(1) := 1;
5.        servo_pos(2) := 0;
6.        servo_pos(3) := 0; --servo 3
7. elsif logic_byte = "00000010" then
8. ...
```

The third change is to integrate the servo controller as an RTL instance to the top-level entity, this allows the servo code to be integrated with the UART RTL instance whereby the aforementioned logic_byte can be connected to the UART RX line. This allows for the servo controller RTL instance to listen to the incoming RX bytes and upon receiving the relevant byte will change the servos to move to the new position. This code was compiled to the FPGA and formed the basis of the servo UART integration test. The intended outcome of this test was to validate reliable UART connectivity and servo control by listening for servo commands from the Arduino. This test will be validated on the basis of the reliability of servo movement and consistency of transmission. The test was performed using the Arduino echoing serial commands from the serial monitor and sending them over a software serial pin to the FPGA. At the same time, the TX from the FPGA was tested for the line sensor code and was found to be reliable. However, during testing the RX code was shown to be unreliable. This issue is elaborated in the RX section, but the result was requiring a different RX system. Following this change the test was performed again under the same conditions and this time it was shown that the FPGA readily accepted the commands of the Arduino. Thus, the second iteration of this test was deemed a success. It was noted during this test that the FPGA took some time to establish the RX connection, while unexpected, this suggests that to maximise reliability it is ideal to wait a short time between powering on the FPGA and running the machine.

4.3.2.2. Button Debounce and Toggle

Code Description

This VHDL code implements a button debounce for an external momentary push button, and also includes a toggle functionality generated when it recognises a rising edge of the push button. Essentially, button denouncers aim to transform signals that look like this:

```
1. “00000001010101010011100111111111111101010100111111111111000000000”
```

To debounced like this:

```
1. “00000000000000000000000000011111110000000000000000000000000000000”
```

And finally, toggle when it senses a rising edge like this:

```
1. “00000000000000000000000000011111111111111111111111111111111111111...”
```

And upon another button press:

```
1. “...11111111111111111100000000000000000000000000000000000000000000000”
```

The code ensures that any button signal change is considered valid only if it remains stable for a specific duration. This duration can be adjusted in the code.

The functionality above is distributed across three processes. The first process toggles the button state register if the counter reaches its maximum value. The second process manages the counter, resetting it if the button state matches or is stable, and incrementing it if there is a potential state change. The third process synchronizes the button state register to generate a debounced pulse and manage the toggle output based on this pulse and the reset signal. The debounced button state and toggle state are then assigned to the output ports.

This code ensures that the button signal is processed to eliminate noise or spurious changes, providing a reliable debounced signal and an additional toggle output that flips its state on each valid button press.

Testing and Debugging

Upon testing the code with the push buttons, it was found that X"FFFFF" or 2^{20} clock cycles was the most accurate at debouncing the buttons. For reference this is around 1/50 of a second at 50MHz clock cycle of the FPGA. Hence, the button state will only be registered if the button reads ‘0’ (modified the code for normally open) for this duration.

It was also very difficult to test the button, as there was interference when wiggling the wires. Upon further research, it was found that a pull up resistor (20k ohms) was needed to prevent the button from reading a false low (in this case the button was connected between GPIO pin and ground and normally open).

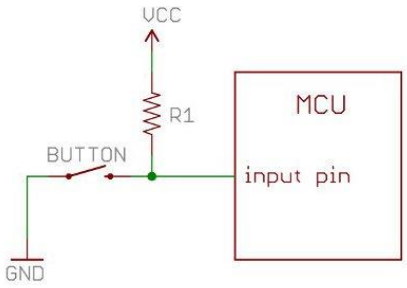


Figure 28 Button Schematic

For the application of the push buttons, a reset feature was also added to the code so that the toggle output was reset to zero on command.

```
1.       btn_toggle_process : process(CLK)
2.       begin
3.                 if (rising_edge(CLK)) then
4.                           btn_sync(0) <= btn_reg;
5.                           btn_sync(1) <= btn_sync(0);
6.                           btn_pulse   <= not btn_sync(1) and btn_sync(0);
7.                           if RES = '1' THEN
8.                                     btn_toggle <= '0';
9.
10.                          else if   btn_pulse = '1' and RES = '0' then
11.                                              btn_toggle <= not btn_toggle;
12.
13.                          end if;
14.                          end if;
15.                end if;
16.      end process;
17.
```

4.3.2.3. WS2812B Controller

Each WS2812 controller will strip the first 24 bits from a received signal and forward the remaining stripped signal onto the next controller. After each received signal, a reset code must be sent before the next signal can be sent. This is shown in the figure below:

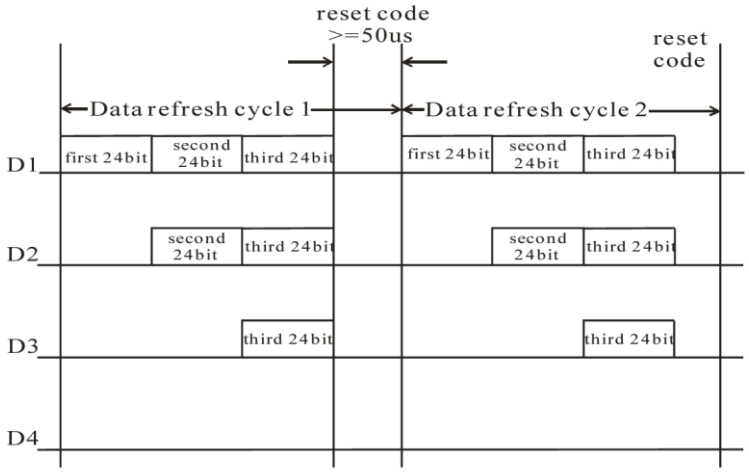


Figure 29: Data Transmission example

The 24 bits that were stripped determine the brightness of each of the three onboard LEDs; green, red, and blue where 8 bits are allocated to each. Each 8 bits represents binary for the intensity of the LED from 0 to 255. The controllers can receive 3 kinds of information: a 0 code, a 1 code and a reset code, and their corresponding signals are based on the following sequence chart:

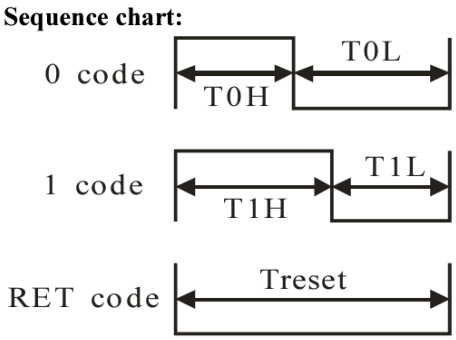


Figure 29: Data Sequence Chart

Time Interval Type	Time	Allowed Error
T0H	0.35μS	±150ns
T1H	0.7μS	±150ns
T0L	0.8μS	±150ns
T1L	0.6μS	±150ns
RES	Over 50μS	

4.3.2.4. Line Sensor

The line sensor that was used was a digital version of the SparkFun Line Sensor Breakout - QRE1113. The breakout board contains power (V_{CC}), ground and an "inout" pin. The device is different from all other sensors used in the project, contains a pin that acts as both an input and output device, so it is triggered and then the device is listened to.

The circuit contains a resistor that limits the current for the diode so that it continues to put out IR light. It also contains a capacitor that will stay charged until it is pulled high.

When the line sensor is charges through the in out pin, it takes both the plates to the same voltage which discharges the capacitor. Whatever light the sensor is seeing will allow charge to flow to the capacitor at a turn on rate. However long is takes to charge is how much light is on the transistor. This is shown in the following waveform:

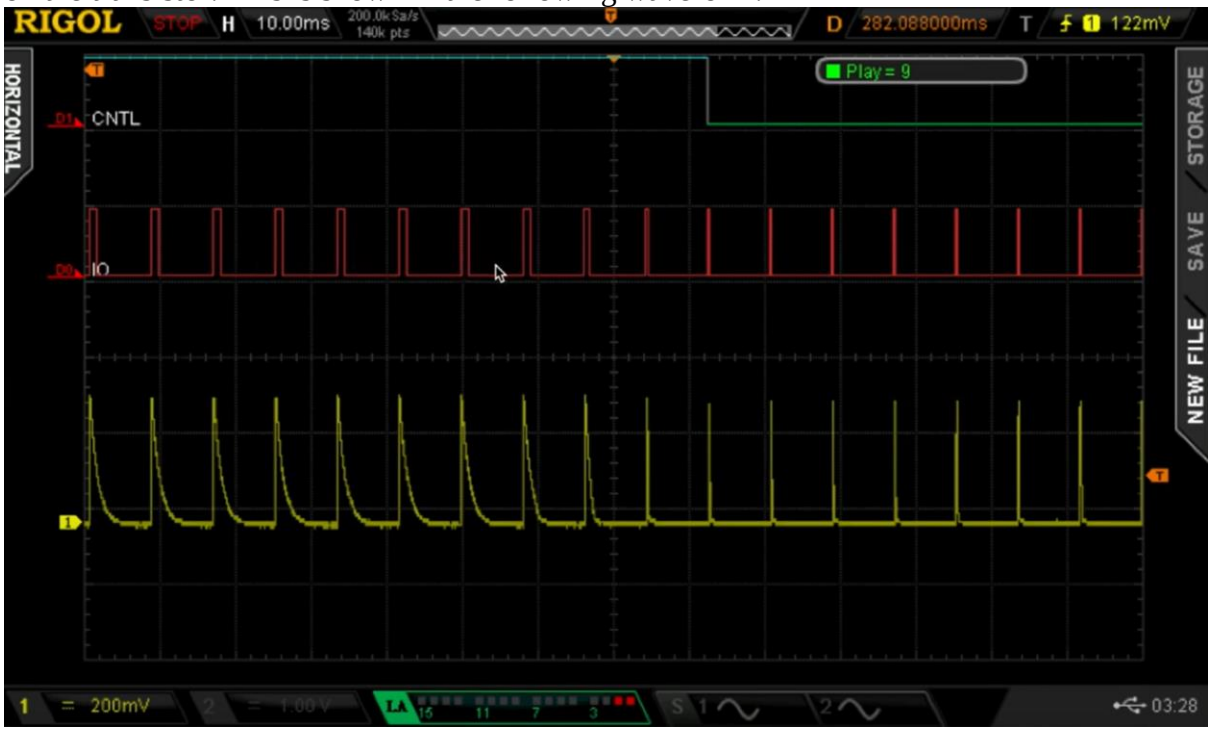


Figure 30: Capacitor Waveform

The FPGA code is designed to charge the sensor through the inout pin, and the discharge time indicates how much light is detected. A reference discharge time is used, and the module returns a high value when the discharge time is less than the reference discharge, indicating an object is detected as there will be more reflected light bouncing back from the object onto the sensor.

Testing and Debugging

The code for the line sensor was inspired by a Verilog code, available at [5], which is a YouTube video that thoroughly describes interfacing with this sensor in HDL by using a tristate buffer, and an inout pin.

Several modifications were made to the code, including writing the code completely in VHDL. A modification was made to changing the 'calibrate button' feature, to setting a range of values that will trip the IR sensor to adjust its sensitivity. This range of values was as high as possible to increase the sensitivity.

A pulse signal was also created to send a pulse output of adjustable time to decrease the number of rising edges within a certain period for debouncing purposes. This was used in conjunction with the TX component to avoid multiple messages being sent to the Arduino when triggered by the moving ball.

4.3.2.5. Servo Controller

The servo controller code on the FPGA underwent several revisions and tests during development. The first test for the servo controller was to generate the PWM waveform required to drive the servos. The waveform to drive the servos needs to be approximately 5V peak, with a period of 20ms, with a duty cycle of between 1ms and 2ms (5-10%). The validation requirements for this proof of concept are to generate a signal with the appropriate PWM characteristics and provide a method of changing the effective PWM duty cycle algorithmically. The PWM signal was measured using an oscilloscope and pre-scaler and duty cycle values were changed in order to generate a measurably correct PWM signal.
The code developed for this test utilises a pre-scaler which acts as a secondary counter to convert the 50MHz clock to a 12.75KHz clock. This is useful as the full clockspeed range is not required for a 20ms signal and 12.75KHz clock speed is evenly divisible by 255 which is my chosen duty cycle granularity. Following the pre-scaling, the cnt variable is used as the secondary clock counter, as the count reaches the duty cycle toggle points for position 0 (0°) and 1 (approximately 20°) the PWM signal will be toggled on and otherwise it will be turned off. The reason two different duty cycles are required is that the servo is stable when receiving no PWM signal and hence requires a signal with duty cycle of 5% in order to return to the 0° position.

```
period := CLK_FREQ / TARGET_FREQ;
duty_count_0 := DUTY_CYCLE_0 * MAX_COUNT / 255;
duty_count_1 := DUTY_CYCLE_1 * MAX_COUNT / 255;

-- Each clk cycle
if (rising_edge(clk)) then
                -- Use a switch to change the state from 0 to 1 and vice versa.
                servo_pos := toggle;

                -- Prescale the 50MHz clock to 12.75kHz
                if freq_div_count <= period then
                        freq_div_count := freq_div_count + 1;
                else
                        cnt := cnt + 1;
                                freq_div_count := 0;
```

```
16.                         end if;
17.
18. -- Depending on the requested position of the servo use a different duty cycle count
19.                 if servo_pos = 0 then
20.                         if cnt >= duty_count_0 then
21.                                 servo <= '0';
22.                         else
23.                                 servo <= '1';
24.                           end if;
25.                 elsif servo_pos = 1 then
26.                         if cnt >= duty_count_1 then
27.                                 servo <= '0';
28.                         else
29.                                 servo <= '1';
30.                         end if;
31.                 end if;
32.
33.                 -- Reset the counter if it reaches the maximum count
34.                 if cnt >= MAX_COUNT then
35.                         cnt := 0;
36.                 end if;
37. end if;
```

The proof of concept proved effective at generating a PWM signal. While the initial pre-scaler values generated a signal, initially the signal did not match the required waveform characteristics as it had a PWM frequency of 1ms instead of 20ms. Through the use of the oscilloscope, and recalculating the pre-scaler values, at the conclusion of the test, the FPGA could successfully control a 9g servo using a PWM signal with 20ms frequency and 1-2ms duty cycle with a toggle switch to toggle between the two servo states.

4.3.2.6. Integration of VHDL Components

Code Functional Description

The code ‘Button_RTL.vhd’ integrates all the components except for an RX module and the servo module and consist of the following components:

- 4 instances of the button debounce code, one for each button.
- A UART TX module to transmit a corresponding signal upon a rising edge of the button toggle as well as upon rising edge of the IR sensor detection (sends signal once per button press).
- A line sensor interface component to register when the ball is detected at the bottom of the screw lift.
- A WS2812B controller to control the longer LED strip.
- A WS2812B module that signals the LEDs to have a rainbow pattern.
- The module also has an RX input so it can detect the state of the RX register (sets the colour of the long LED).

Each colour LED (Red, blue, yellow and green) correspond to a different path of the crazy machine (Ferris wheel, piano slide, planetary revolution and ladder). When the ball is going through a section, the long LED strip colour should match the colour assigned to the section.

The function of the code is when no buttons are pressed, the long-led strip will play the rainbow pattern and all the button LEDs are off. Once a button is pressed, then the long led will have a circular pattern of the colour pressed (only one led is on at a time and turns off after a short delay time, and then the led next to it lights up), as well as the button LED to latch until all buttons are pressed. This continues until the last button is pressed. After the last button is pressed, the button LEDs will flash on/off at 0.5s intervals for 60s, ie. the time limit of the Crazy Machine. During this time, the long LED will also have a circular pattern, however, its colour will be dependent on the value of the RX register. The RX register will contain the signal sent by the Arduino to control the servo positions. This is also the current section that the ball will be passing through. Hence, the colour of the LED will match the colour of the section the ball is passing through.

Code Description

The code consists of many processes which describe how the system operates. The details of this system described in Butoon_RTL will be outlined in this section.

This state describes the state of the button LEDs depending on their pressed states. The buttons that were purchased for this project were momentary, hence the necessary debouncing and toggling was done in the button and debouncing section in 4.3.2.2. However, as core electronics decided to not ship our yellow momentary push button, it was found that the only option was to purchase a non-momentary (latching) push button from Altronics, as this was all that was in stock locally. Hence, this code adjusts for this latching push button (button(3)), which toggles upon a rinsing edge, as well as a falling edge to compensate. The rest of the buttons only toggle upon rising edge only.

1) LED Strip Processes (two processes)

The LED strip is operated through two processes. The first defines the LED state (on/off) when called to behave in a circular pattern, as well as the colour. The second implements the first process and renders the data to the LED strip.

- LED Pressed Process

The colour of the LEDs are based on a register, btn_pulse(i) which stores the value of the button which most recently had a rising edge. The state of the LEDs are stored in the vector, LED_PRESSED_STATE. The code initialises the value of this vector to be zeros initially. The following code increments this value by one. A timer is used to set the delay between when an LED is on, and when the next LED should be turned on and it should be turned off. This is shown in the for loop in the following code:

```
 1.                          if rising_edge( CLK ) then
 2.                                   LED_pressed_timer <= LED_pressed_Timer + 1;
 3. --detect a rising edge of TOGGLE_O(i). If a rising edge occurs then then set the colour
register. btn_pulse(i) is 1
```

```
 4. --upon rising edge
 5.
 6. --led state. All leds are off except for one. the on led address increments by one for a
delay led_pressed_delay.
 7.                                      if LED_pressed_Timer > LED_pressed_delay- 1 then

 8.                                              LED_pressed_Timer <= 0;
 9. -- shift LED address
10.                                              if (LED_PRESSED_STATE =
(LED_PRESSED_STATE'range => '0')) then
11.                                                                      LED_PRESSED_STATE(0) <=
'1';
12.                                              else
13.                                                       for i in 0 to LENGTH_LONG_LED_STRIP-1
loop -- code to activate leds based on LED_timer and LED_lagtime
14.                                                                      if LED_PRESSED_STATE(i) =
'1' then
15.
          LED_PRESSED_STATE(i) <= '0';
16.                                                                              if i =
LENGTH_LONG_LED_STRIP-1 then
17.
LED_PRESSED_STATE(0) <= '1'; -- Set LED_PRESSED_STATE(0) to '1' when i reaches
LENGTH_LONG_LED_STRIP-1
18.                                                                              else
19.
LED_PRESSED_STATE(i+1) <= '1';
20.                                                                              end if;
21.                                                                      end if;
22.                                                       end loop;
23.                                              end if;
24.                                      end if;
25.
26.                             end if;
27.
```

- LED Render Process

The following code describes a for loop, which loops across every LED address. The code then assigns the LED state and colour described in the LED pressed process to each corresponding LED, and generates the entire colour information for each LED and saves it in the WS2812B controller memory, and then renders the data when all the LEDs have been written to.

```
 1.                                      for i in 0 to LENGTH_LONG_LED_STRIP-1 loop
 2.                                              if unsigned(colIdx) = to_unsigned(i-
1,integer(ceil(log2(real(LENGTH_LONG_LED_STRIP - 1))))) then
 3.                                                                      if LED_PRESSED_STATE(i) =
'1' then
 4.                                                                                              data_red
<= std_logic_vector(to_unsigned(colour(1),8));
 5.
          data_green <= std_logic_vector(to_unsigned(colour(0),8));
 6.
          data_blue  <= std_logic_vector(to_unsigned(colour(2),8));
 7.                                                                      else
 8.                                                                                              data_red
<= (others => '0');
 9.
          data_green <= (others => '0');
```

```
10.
          data_blue  <= (others => '0');

11.                                                                    end if;
12.                                                 end if;
13.                                       end loop;
14.
```

2) Button LED State Process

The button LED State Process describes a state machine with two states, one for when the buttons are being pressed and the ball has not entered the machine, and one for when all the buttons have been pressed, and messages have been sent to the Arduino to update the state of the servos.

- Buttons being pressed state (s_notflashing)

Progression to next state and Button LED Behaviour
This code describes how the system will behave before all the buttons are pressed. Hence the state will progress to the next state when all of the button LEDs have been latched. Behaviour of the push button LEDs are defined in section 4.3.2.2.

```
1.                                                  if TOGGLE_O(2 downto 0) = "111" and
SW_TOGGLE_O(3) = '1' then
2.                                                            state <= s_flashing;
3.
4.                                                  else
5.                                                            state <= s_notflashing;
6.                                                  end if;
7.
```

LED Strip Behaviour
It was chosen that when zero buttons have been pressed, the crazy machine will enter a ‘show mode’ where a signal will be sent to the LEDs to make them enter a cool rainbow pattern. However, it was also decided that when a button has been pressed, the long LED strip should enter a cool circular pattern, which is defined in 1), button LED process, which is ‘so’. This was done in the following code, note that SW_TOGGLE_O detects rising edges and falling edges for the non-momentary push button:

```
1.                                                  if TOGGLE_O(2 downto 0) = "000" and
SW_TOGGLE_O(3) = '0' then
2.                                                  LONG_LED_STRIP <= DEMO_LED;
3.                                                  else
4.
5.                                                  LONG_LED_STRIP <= so;-- strip signal upon
button is pressed eg. when toggle is 0001-1110
6.                                                  end if;
7.
```

TX communication
The code also describes that upon a button being pressed, a signal should be sent to the Arduino via the TX port to tell it to latch the order of the crazy machine sections. This is implemented by detecting a rising edge from all the push buttons when they have been pressed. This is done in the following lines of code, note that ascii_table is an array of all the pre-defined messages that have functionality in the Arduino code:

```
                                                    1. btn_sync(3,0) <= SW_TOGGLE_O(3); -- BAD
BUTTON*********************
 2.                                                 btn_sync(3,1) <= btn_sync(3,0);
 3.                                                 btn_pulse_falling <=  btn_sync(3,1) and not
btn_sync(3,0);
 4.                                                 btn_pulse_rising   <= not btn_sync(3,1) and
btn_sync(3,0);      --is 1 when rising edge of a button
 5.                                                 btn_pulse(3) <= btn_pulse_falling or
btn_pulse_rising;
 6.
 7.                                                 if btn_pulse(3) = '1' and active_TX = '0' then
 8.                                                           dataValid_TX <= '1';
 9.                                                           data_TX <= ascii_table(3);
10.                                                 end if;
11.
```

- All LEDs are pressed state (s_flashing)

Progression to Initial State
This state ends once a 60s-time limit is reached, which is the time limit set in the rubric for the crazy machine. Once this time limit is reached, the button states are reset so they can be pressed again if a re-run is required.

Button LED Behaviour
It was decided that all the push buttons should flash at an interval of 0.5s when the ball is going through the crazy machine. Hence, the following code generates a variable frequency of 0.5Hz, and the code following shows the section of code in the state machine that maps this generated signal to the button LEDs when they are in this state. Please note the constant r_CNT_1HZ was varied in the final code to be 0.5Hz.

```
  1. p_1_HZ : process (CLK) is --1HZ COUNTER (for flashing LEDs)
 2.   begin
 3.     if rising_edge(CLK) then
 4.       if r_CNT_1HZ = c_CNT_1HZ-1 then  -- -1, since counter starts at 0
 5.         flash_led <= not flash_led;
 6.         r_CNT_1HZ    <= 0;
 7.       else
 8.         r_CNT_1HZ <= r_CNT_1HZ + 1;
 9.       end if;
10.     end if;
11.   end process p_1_HZ;
12.
```

```
1.                                                  LED_O(0) <= flash_led;
2.                                                  LED_O(1) <= flash_led;
3.                                                  LED_O(2) <= flash_led;
4.                                                  LED_O(3) <= flash_led;
5.
```

LED Strip Behaviour
In this state, the LED strip behaves similar to the previous state, however, it was chosen that instead of the colour being the state of the push buttons, the colour should match the current section that the ball is passing. This is done by mapping the incoming RX signal (which

consists of 5 possible signals that control the servo positions (path 1-4 and exit). This was done in the following code:

```
 1.                                                  if RX_DATA = ascii_tableRX(4) then
 2.                                                          LONG_LED_STRIP <= so;--
strip signal upon button is pressed eg. when toggle is 0001-1110
 3.                                                  else
 4.                                                          for i in 0 to 3 loop
 5.                                                                  if RX_DATA =
ascii_tableRX(i) then
 6.
          LED_pressed_colour <= std_logic_vector(to_unsigned(i,2));
 7.                                                                  end if;
 8.                                                          end loop;
 9.                                                  end if;
10.
```

4.3.2.6.1. Top Level for Splitter and Screw Lift

All components related to the splitter and screw lift were connected into one file, Servo_RTL, and is called upon later in the final top-level entity in the integration section 10.1.

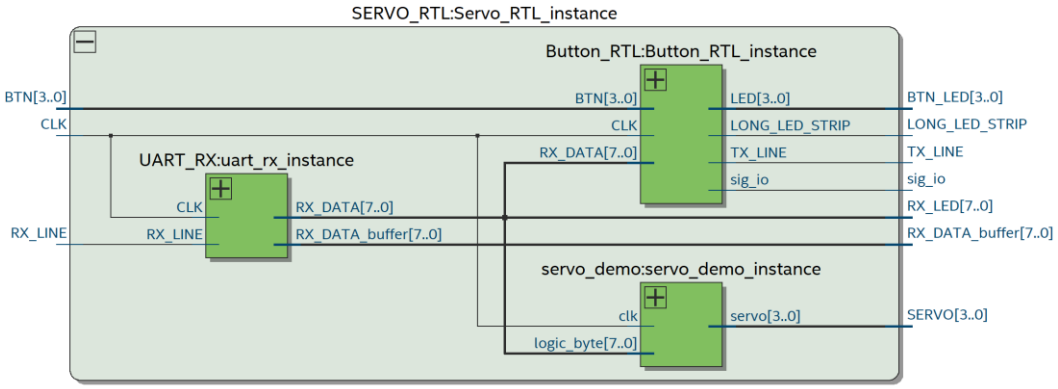


Figure 31 Top Level Entity

5. Planetary Revolution

The planetary revolution mechanism consists of interconnected gears that work together to orbit a marble 360 degrees around a central point. This rotation is driven by a single stepper motor attached to a central gear.

```
1. #include <Stepper.h>
```

```
#include <string.h>
#include <SoftwareSerial.h>

Stepper liamStepper(stepsPerRev, 4, 6, 5, 7);
int current_count = 0;
int target_position = 0;
int liam_count = 0;
int liam_target = 0;

void send_station(int arr[]) {
  if (stations_completed <= 4) {
    SoftSerial.write(stationMap[arr[stations_completed] - 1]);
   Serial.print(arr[stations_completed] - 1);
    Serial.print(" ");
    Serial.println(stationMap[arr[stations_completed] - 1]);
    // tmrpcm.play(soundMap[arr[stations_completed] - 1]);
    Serial.print("Soundmap: ");
    Serial.println(soundMap[arr[stations_completed] - 1]);
    if (arr[stations_completed] == 4) {
      liam_target = 60 * 512;
      liam_count = 0;
    }
    stations_completed++;
  } else {
    Serial.println("Error: stations out of range");
  }
}
}

void liam_spin() {
  if (liam_count < liam_target) {
    Serial.println(liam_count);
    liamStepper.step(-512);
    liam_count += 512;
  }

  liam_spin();
}
```

5.1. Assumptions

1. **Integration with Ball splitter:**
   The main method of integration of the planetary revolution to the rest of the crazy machine project is through the Ball splitter segment. The dimensions of the ball splitter are consistent with drawings and designs.

2. **Aesthetic of planetary revolution:**
   The visual and functional design of the planetary revolution should integrate and match the overall theme of the marble crazy machine.

3. **Material Durability:**
   The 3D printed gears will require minimal maintenance and will be durable enough to function effectively over the duration of the project.

4. **Software Compatibility:**
   The 3D modelling software is compatible with the 3D printing systems file format.

5.2. Decisions

1. **Material Selection:**
   PLA was chosen for the 3D printing filament due to it's ease of use, strength and affordability.

2. **Motor Specification:**
   The motor was chosen with the torque and speed characteristics which were capable of moving the planetary revolution gear system.

3. **Modularity:**
   The design was made modular to allow for easy replacement of individual parts in the case that they fail or need adjustment.

4. **Aesthetic design:**
   The design of the planetary revolution is fits in with the design theme of steampunk.

5.3. Design Process

5.3.1. Design Plan

Crazy Machine Project

Planetary Revolution Section

Start: Mon, 3/4/2024

Display w 1

TASK	PROGRESS	START	END
Planning			
Select section of Crazy Machine	100%	3/6/24	3/15/24
Draft sketch of concept	100%	3/6/24	3/15/24
Create component list	100%	3/6/24	3/15/24
Verify plan with team members	100%	3/6/24	3/15/24
Design			
Design individual components on Fusion360	100%	3/15/24	3/29/24
Create 3D model of segment	100%	3/15/24	3/29/24
Design Ball Return	100%	3/15/24	4/12/24
Coding & Electronic Design			
Test sample code and circuit of component	100%	3/15/24	3/20/24
Build overall circuit	100%	3/21/24	4/16/24
Integrate electronics with mechanical components	100%	4/17/24	4/22/24
Make adjustments to code	100%	4/22/24	4/24/24
Test final code and circuit	100%	4/24/24	4/26/24
Constructing			
Print 3D parts	100%	4/12/24	4/22/24
Test sizing of 3D Parts	100%	4/22/24	4/29/24
Acquire recycled parts	100%	4/30/24	5/3/24
Construct segment	100%	4/17/24	4/20/24
Test mechanics of system	100%	5/6/24	5/11/24
Integration			
Test electronic circuit with mechanical components	50%	5/12/24	5/15/24
Combine section into box with other sections	60%	5/12/24	5/14/24
Integrate circuit with others	50%	5/14/24	5/23/24
Verify with team members	25%	5/23/24	5/26/24

Figure 262: Planetary Revolution Gantt Chart

5.3.2. Initial design

The initial design shown in figure 7 can be broken down into four components. The first component is the outside ring, which will carry the ball around the loop. The second component is the base, which will secure the gears and the outside ring and has an oval-shaped hole for the marble to roll out of. The third component consists of the three matching gears used to distribute the power and spin the outside ring. Finally, the last component is the central gear, which has the appropriate slot to mount the stepper motor and is used to move the three identical gears.

In the initial design the main concern was about friction from the gears sitting directly on plastic. If the stepper motor struggles to move the planetary revolution, bearings will be used in the centre of the gears to reduce friction.

Figure 273: Planetary Revolution

Figure 284: Planetary Revolution Component 1

Figure 295: Planetary Revolution Component 2

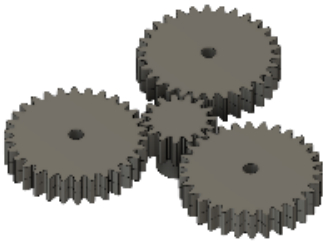


Figure 36: Planetary Revolution Component 3

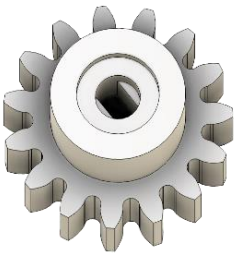


Figure 307: Planetary Revolution Component 4

5.3.3. Final design

Before printing the planetary revolution, I was analysing the mechanics of the planetary revolution system. I felt the planetary revolutions dimensions were large and was going to require a lot of torque from the motor. Unsure of the capacity of the motor I made the decision to integrate bearings into the design. These bearings would mitigate friction and alleviate the workload on the stepper motor.

Figure 3831: Planetary Revolution Final Diagram

5.4. Implementation

The next phase of the process of creating the model was going to printing the planetary revolution. Ideally, I only wanted to 3d print the project one time so I made sure to that the design would work and not need altering. I wanted to do this to reduce the amount of wasted print filament as possible. This is also the justification as to why the project was printed with lots of infill to mitigate the risk of it breaking and the need to reprint. As I have limited 3D printing experience, I was concerned about how much tolerance to leave in my design to allow for movement of the planetary revolution. This was largely going to depend on the accuracy and speed of the printer. In preparation to print the parts I made sure to configure the 3D printer to the following settings to allow for maximum accuracy. Firstly, I set the printers parameters to work well with my filament type. I was using a very standard PLA. I deliberately made my nozzle to 0.2mm which is a small nozzle size and would take longer to print however, it would make the project more precise. I set the printer to work at a slow

speed; this increases accuracy by decreasing vibrations caused by the constant movement of the motor nozzle.

The planetary printed after 6 hours using two separate 3D printers. I was very relieved to see that all the pieces printed to a high standard. I needed to get the exact bearings which I had used in my 3d model which were sized on bearing I found online at bunnings, without these bearings the planetary revolution would not spin properly. Here I made a logistical fault in which I assumed bunnings would keep stock at all time. However, this item was an online only item which had to be ordered and delayed my ability to test pushing me off schedule while I waited for parts.

Once the bearings arrived, I was very happy to see the planetary revolution print was a great success. It was able to move with very little force and the tolerances were all appropriate. Upon attempting to install the motor into the specially designed insert, it became apparent that the motor was too large to fit properly. I had originally thought I'd allowed for enough tolerance however this the to be remade to accommodate the motor's dimensions with some more tolerance.

Figure 329 Final Design of Planetary Revolution

The next issue I had to solve was how to mount the planetary revolution to the box. It has to sit roughly 15cm off of the box floor to accommodate the ball return and the motor. My original plan was to design and 3D print something to hold this, however I believed that I could be going down a rabbit hole of 'over-engineering'. I found some components used in a previous crazy machine project which I thought I could recycle and use to raise the planetary revolution instead.

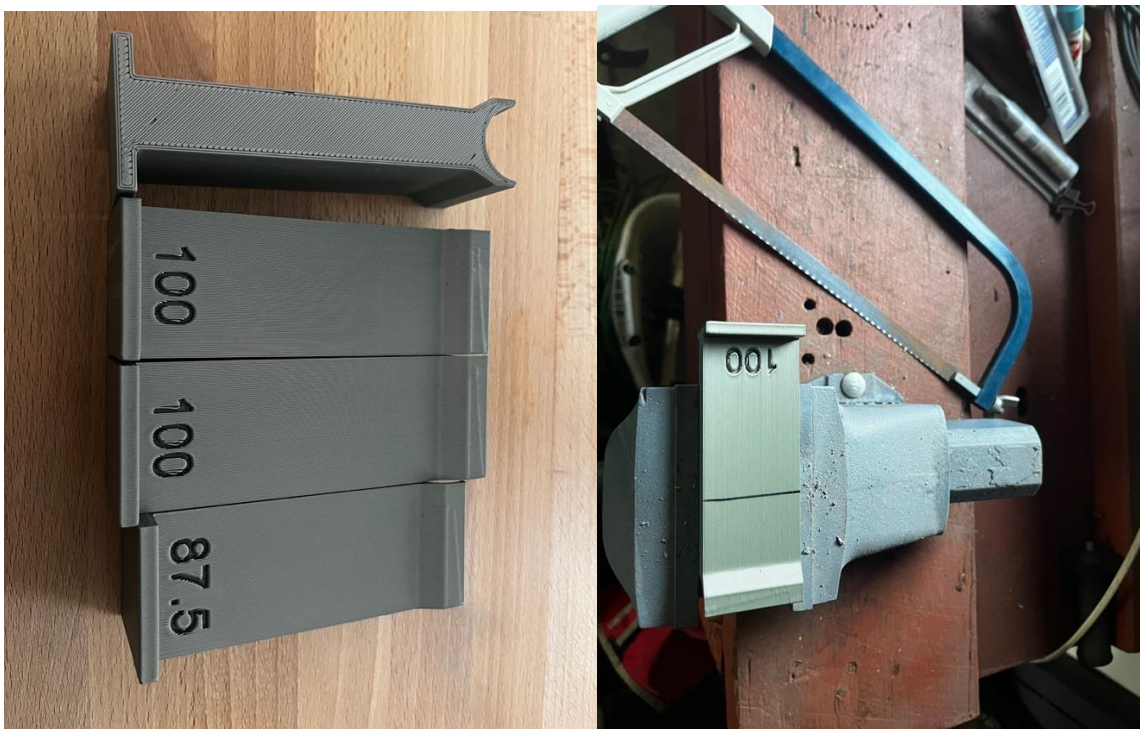


Figure 40 Modifications Made to Planetary Revolution Stand

I measured the recycled parts and using a handsaw I cut the parts to length. Once all the parts had been cut I used some Velcro strips to secure them to the base of the revolution. I made one of the legs shorter to sit under the motor and prevent the motor from moving.

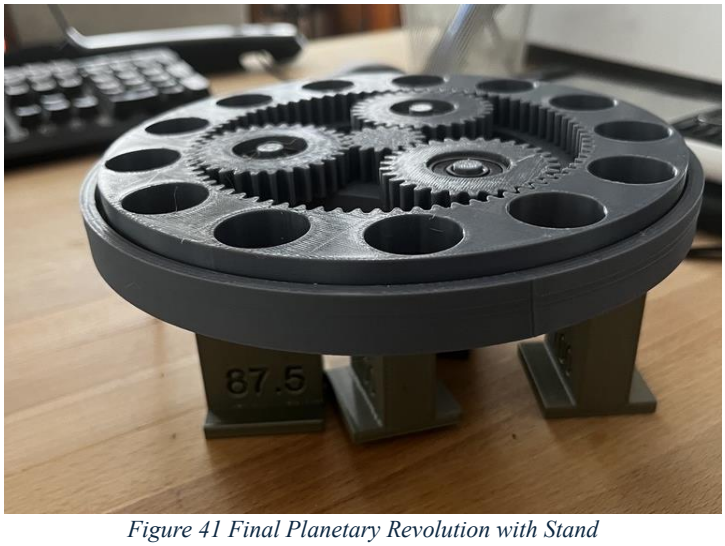


Figure 41 Final Planetary Revolution with Stand

5.5. Hardware specification

There are several advantages to using a stepper motor instead of a DC motor. Firstly, a stepper motor allows for precise control over rotation. Additionally, stepper motors provide higher torque at slow speeds, which is ideal for this application due to the high frictional resistance. The stepper motor also features an easily integrable rotating arm, as shown in figure 12. This arm allows direct connection between the motor and the gears. Moreover, the stepper motor is highly repeatable, ensuring it can reliably return to the same position each time.

Table 10 5V DC Stepper Motor Specifications

5V DC Stepper motor	
PART NUMBER	XC4458
Description	Duinotech Arduino Compatible 5V Stepper Motor with Controller
Stepper Motor Size	28mm diameter
Nominal Torque	3Ncm
Step Angle	5.625 x 1/64
Operating Voltage	5VDC
Dimensions	35(L) x 32(W) x 10(H)

5.6. Verification – Proposed Testing Methodology & Results

The following code is the code used for controlling the stepper motor. The code works by using the stepper and SoftwareSerial libraries. It initializes a stepper motor with specific pins and sets up variables to track the motor's position and target steps. The send_station() function sends data corresponding to stations via serial communication and updates target steps if a specific station is reached, printing relevant information to the serial monitor. The liam_spin() function is designed to incrementally move the stepper motor towards the target position; then, it uses recursion to continue spinning.

```
1. #include <Stepper.h>
 2. #include <string.h>
 3. #include <SoftwareSerial.h>
 4.
 5. Stepper liamStepper(stepsPerRev, 4, 6, 5, 7);
 6. int current_count = 0;
 7. int target_position = 0;
 8. int liam_count = 0;
 9. int liam_target = 0;
10.
11. void send_station(int arr[]) {
12.   if (stations_completed <= 4) {
13.     SoftSerial.write(stationMap[arr[stations_completed] - 1]);
```

```
14.     Serial.print(arr[stations_completed] - 1);
15.      Serial.print(" ");
16.      Serial.println(stationMap[arr[stations_completed] - 1]);
17.      // tmrpcm.play(soundMap[arr[stations_completed] - 1]);
18.      Serial.print("Soundmap: ");
19.      Serial.println(soundMap[arr[stations_completed] - 1]);
20.      if (arr[stations_completed] == 4) {
21.        liam_target = 60 * 512;
22.        liam_count = 0;
23.      }
24.      stations_completed++;
25.    } else {
26.      Serial.println("Error: stations out of range");
27.    }
28. }
29. }
30.
31. void liam_spin() {
32.   if (liam_count < liam_target) {
33.     Serial.println(liam_count);
34.     liamStepper.step(-512);
35.     liam_count += 512;
36.   }
37.
38.   liam_spin();
39. }
```

Once this code was completed the planetary revolution could be tested. The test shown in figure 38 was the primary test done to test the whole system together. The Audino sent communication and power to the driver. This driver is connected to the stepper motor. After correcting some small errors in the code the system worked well and the planetery revolution spun.

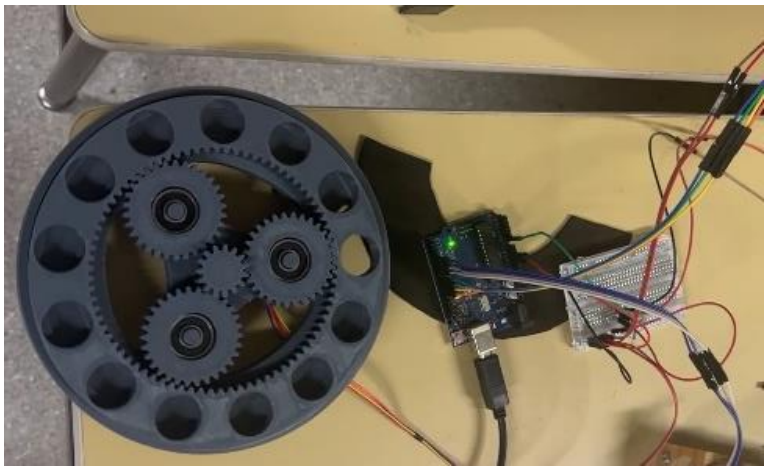


Figure 42 Final Planetary Revolution Verification

6. Ferris Wheel

6.1. Assumptions

The following assumptions formed the foundation of designing the Ferris Wheel segment in the Crazy Machine Project:

1. Ball Splitter segment will allow for ball to be delivered successfully into bucket –

The primary assumption is that the Ball splitter segment of the Crazy Machine Project will effectively create a divider that will allow for the ball to smoothly be delivered into the buck without any obstruction.

2. Force sensing resistor can be mounted in ball splitter segment

The assumption that the Force Sensing Resistor (FSR) can be accurately mounted within the ball splitter segment, enables accurate detection of forces exerted by the ball as it travels into the Ferris wheel segment.

3. Construction of Ferris Wheel components will be required to be completed first to allow for testing with other sections

It was anticipated that the full assembly of the Ferris wheel section would occur prior to the integration of other sections to ensure all sections could coexist in the Crazy Machine Project without impacting others.

4. All sections will stay within their initial allocation space –

It was assumed that each segment of the Crazy Machine Project would remain within its initially allocated space throughout the entire process of construction and testing, to ensure that there is no interference between segments.

6.2. Decisions

After detailed planning and verification of the conceptual design, the following decisions were made in regard to the Ferris Wheel segment in the Crazy Machine Project:

1. Remove pull solenoid and digital line sensor –

In the process of refining the circuitry of the Ferris wheel section, it was decided that the pull solenoid and digital line sensor components would be removed. This decision was made due to the initial assumption that the solenoid was a push solenoid being false. To streamline the design and remove unnecessary complexity, the digital line sensor was removed as its planned purpose was redundant.

2. Use Arduino for components rather than FPGA–

After consideration between using a combination of FPGA and Arduino for controlling components, it was determined that using Arduino would be more advantageous. This choice priorities simplicity and ease of integration with the wider group.

6.3. Design Process

6.3.1. Design Plan

The Ferris Wheel section was designed following the initial plan of using gears in a 2:1 ratio, however the initial idea of use of 3 gears was reduced to the use of 2 gears. The large gear has a single bucket that will catch the ball and transport it in a full rotation. The smaller gear drives the rotation of the large gear with use of a stepper motor.

The model is composed of a single bucket, metal arm, screws and two gears. All components aside from the gears and bucket are recycled materials, that were gathered from parts around the home. The 3D Printed components were designed through use of AutoCAD.

The following Gantt Chart represents a detailed overview of the plan for development of the Ferris Wheel segment of the Crazy Machine Project.

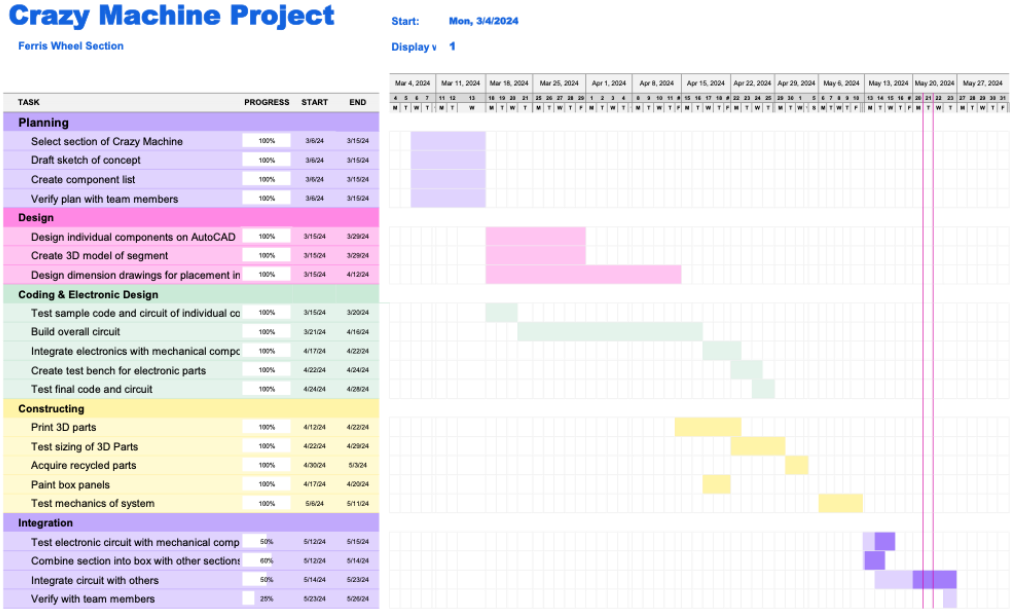


Figure 43 Ferris Wheel Gantt Chart

6.3.2. Initial Design

The original design is displayed below in figure __. while the concept was still maintained, several features were altered for practical assembly within the Crazy Machine Box.

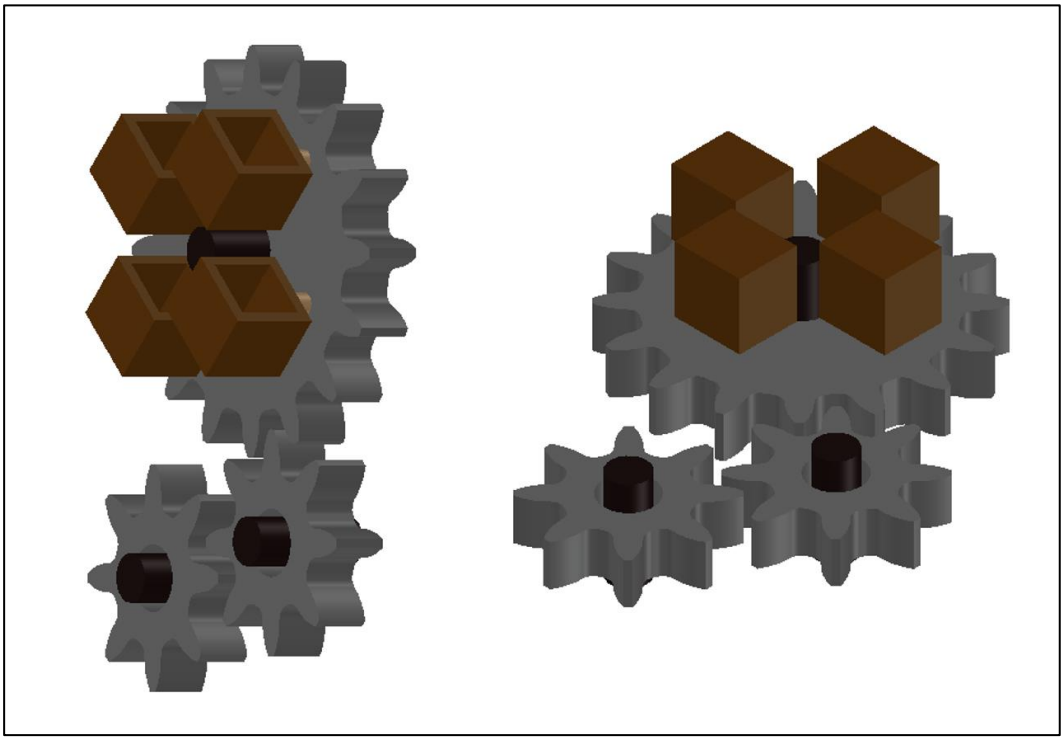


Figure 44 – Ferris Wheel Original Design

Various design challenges were faced in printing 3D parts due to minor oversights in technical proficiencies. The following challenges were faced:

1. Incorrect initial measurements
2. Lack of planning for how to fit stepper motor onto Small Gear
3. Incorrect hole sizes for screws
4. Unnecessarily large wall dimensions for bucket
5. Minimal access to 3D printers

6.3.3. Final Design

To address the aforementioned adversities, Curtin University 3D Printing license was acquired to make use of the Makerspace facilities in Curtin University where multiple prototype models were developed to measure and test sizing of components. To address incorrect initial dimensions of various components, gears were resized and tested before re-printing them and testing them with different hardware readily available at home.

Proof of Concept

As shown below, prototypes for stepper motor shaft attachment sizing are shown below, that was used for testing the fit of the shaft into the centre of the small gear. The different sizing of for stepper motor shaft attachments is displayed below, printed inserts were required differing in 0.5mm for both length and width in order to find an attachment size that worked effectively. The incorrect initial sizing for gear arrangement strategies is also displayed.

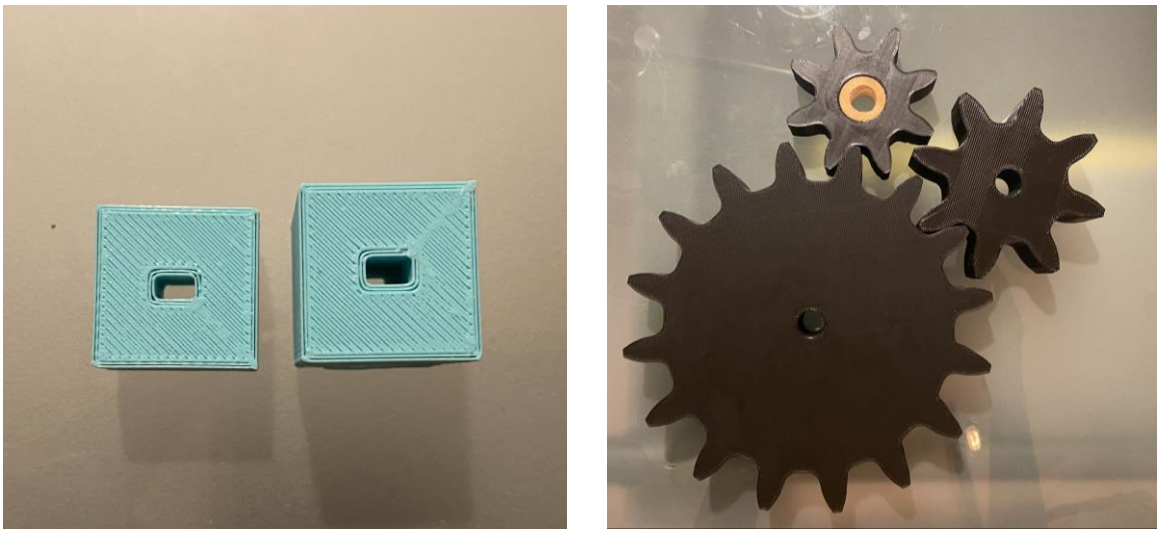


Figure 45 – Ferris Wheel Prototype Designs

Components to Print

As shown below, the followings components were printed:

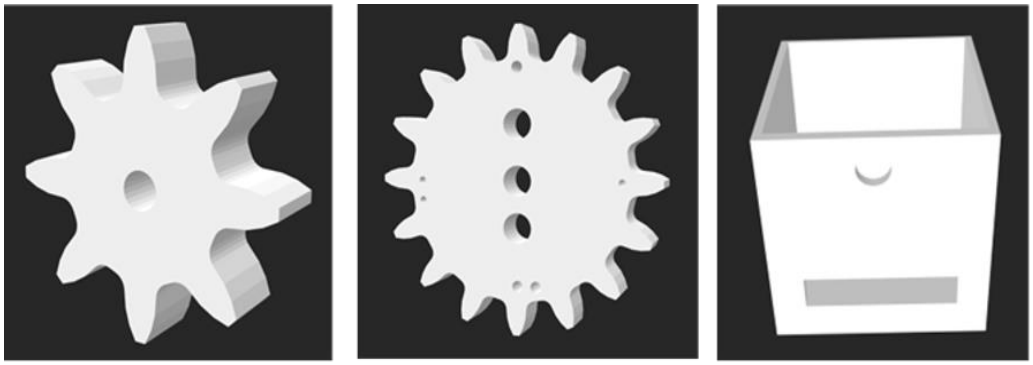


Figure 46– Ferris Wheel Final Component Designs

Final Model

A final model design is displayed below:

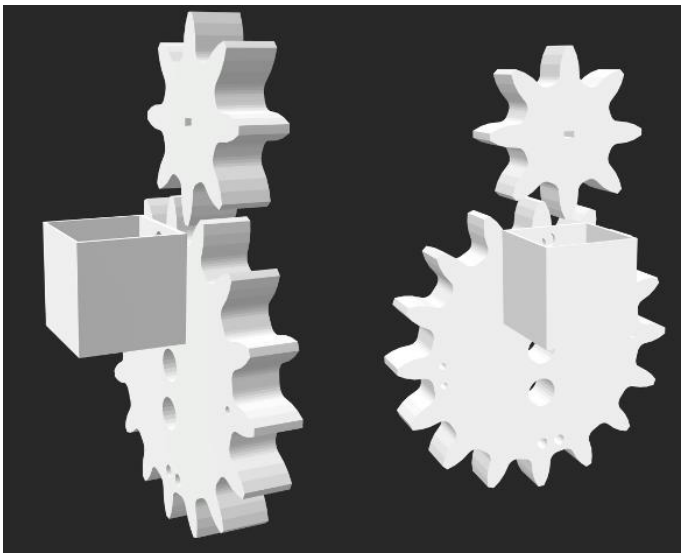


Figure 47 – Ferris Wheel Final Model Design

6.3.4. Recycled Components

Table 11 Recycled Component List

Number of Parts	Part Description
2	Wood blocks as backing for large gear and bucket
2	13mm O-rings
5	Assorted screws
1	Scrap metal as tipping piece
1	Metal pipe as dowel

6.4. Implementation

6.4.1. Building

To construct my part of the project, a larger role was taken in terms of spray painting, measuring, and constructing the backing panels for the crazy machine project. The reasoning behind this was because my components required direct attachments to the panels to successfully move. Initially matte black spray paint was acquired to spray the backing panels for the box, aiming to enhance the overall project aesthetics and have a hand in the creative aspect of the project.

Before constructing took place, it was necessary to draw designs with measurements for components placements and dimensions of components, these are portrayed in the following figures. A side view of the Ferris wheel model shows a side view containing dimensions for the bucket, large gear, small gear, stepper motor and weighting for the back of the large gear. The dimensions ensure that the drop off point from the ball splitter segment are of an appropriate height to drop into the bucket.

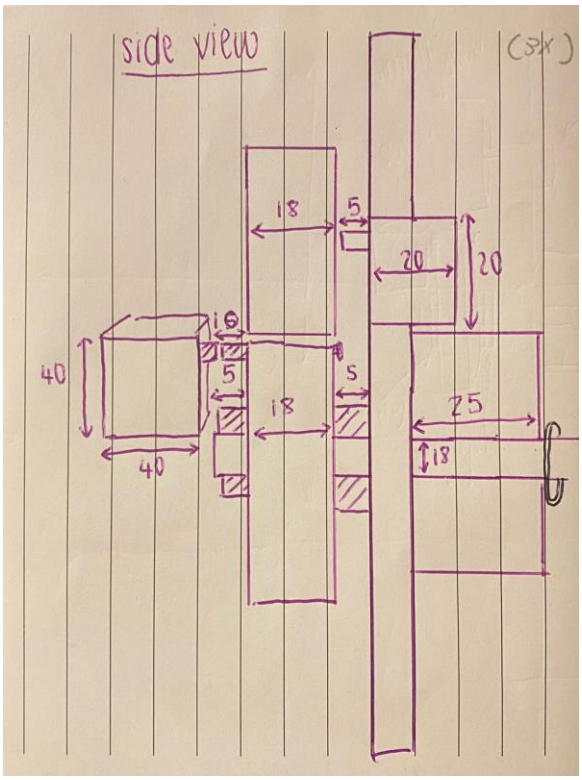


Figure 48 – Ferris Wheel Side View Dimensioned Drawing

The following image portrays a front view of the small and large gear, and their measurements from the side, base and top of the backing panel. Considering the section allocations for both the pinball ladder segment and the ball lift segment.

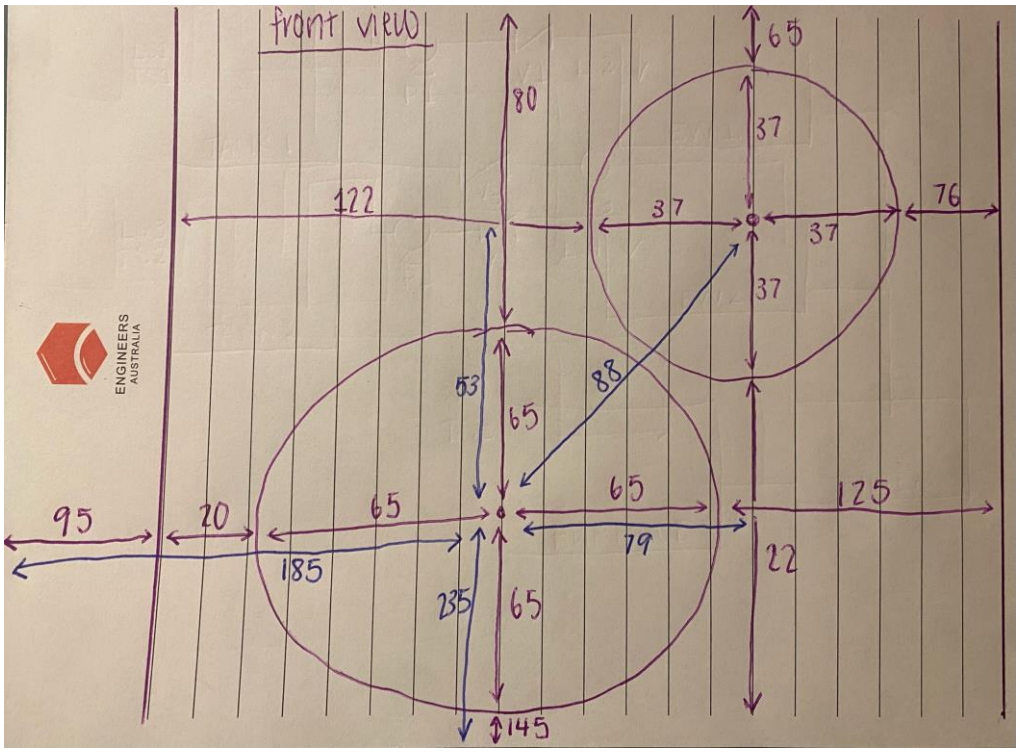


Figure 49 – Ferris Wheel Front View Dimensioned Drawing

Following the technical draft drawings with accurate dimensions, the construction of the Ferris wheel segment was commenced.

Figure 50 – Ferris Wheel Building Process

Using a circular saw and pre-planned dimensions for the box, backing panels were cut out, ensuring they were correctly sized to seamlessly fit the metal structure. Designing and measuring the dimensions for my components within the box, allowed for drill holes to successfully hold all components. Using a drill and regular drill bits, a hole was created for the metal dowel that holds the large gear. Using a drill and a coring drill bit, a structure was created to mount the stepper motor in the box. To further refine the fit of the stepper motor in the backing panel a small hand was used to saw to adjust the dimensions of the hole. Use of a metal saw allowed for a metal pole to be custom sliced and repurposed as the main metal dowel to hold the large gear. This choice was predominantly aesthetic but also benefitted the structure by providing a smooth surface for the gear to rotate upon.

For the assembly of my part of the project, specifically sized screws and washers were sourced from parts that were readily available at home to affix all components securely to the box. This accurate sizing allowed for structural stability and future repetition of motion. The integration of accurately sized rubber O-rings ensured that the components were secure.

Overall, the construction of the Ferris Wheel section of the project took a great deal of time to ensure that all dimensions were correct as the process was highly planned and designed.

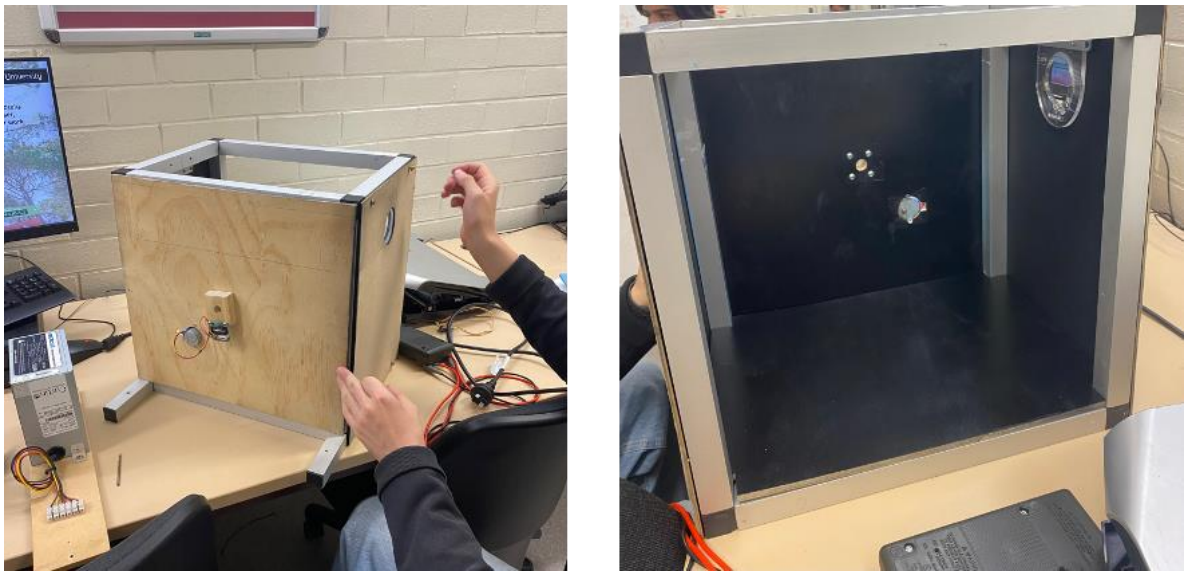


Figure 51 – Ferris Wheel Building Process 2

In class the following week, the panels were drilled into the box and successfully fit the box as shown in Figure 4.

6.4.2. Hardware Specifications

The required hardware for the Ferris Wheel component of the Crazy Machine Project include:

a. a force sensing element to register when the ball has entered the primary bucket.
b. a motor to spin the smaller gear in order to cause the large gear to make a full rotation.

Selected Hardware Elements

Table 12 Hardware Component List

FERRIS WHEEL

NUMBER OF COMPONENTS	COMPONENT TYPE
1	Duinotech Arduino Compatible 5V Stepper Motor
1	Force-Sensing Resistor: 0.6"-Diameter Circle

Force Sensing Resistor

A force sensing resistor (FSR) was used to indicate the presence of the ball before it is dropped from the ball splitter section into the Ferris wheel section. With a relevant delay, the FSR triggers the motion of the stepper motor to control the movement of the large gear.

Justification

A force sensing resistor, namely the FSR402, was the most suitable choice for a sensor in the Ferris wheel segment. The primary justification for using a FSR is the size allowed it to be placed conveniently at the drop off point of the ball splitter section without impacting the motion of the ball. This type of sensor is lightweight and compact which allowed it to be easily inserted. While it is not a necessity for the sensor to be highly sensitive, the high sensitivity of the FSR402 contributed to its selection as it was able to reliably detect the presence of the ball without delay. Additionally, the sensor also offered a fast response time which allowed the presence of the ball within the bucket to be detected almost instantly, the sensor registered the force exerted and triggered the next mechanism rapidly.

Specification

Table 13 Force Sensing Resistor Hardware Specifications

FSR402	
SKU	POLOLU-1696
DESCRIPTION	Force Sensing Resistor
ACTUATION FORCE	~0.2N min
FORCE RESOLUTION	Continuous (analog)
FORCE SENSITIVITY RANGE	~0.2N – 20N
DEVICE RISE TIME	< 3 Microseconds

Stepper Motor

A stepper motor is necessary to turn the smaller gear and by extension trigger the movement of the larger gear.

Justification

A stepper motor was the best choice for this mechanism as it can move in distinct steps and high torque. The chosen stepper motor makes 64 steps per revolution allowing for precise steps with rotation angles of 5.625° [2]. These precise steps allowed for the gears to be precisely moved in a full rotation. This full rotation allowed for the mechanism to reset itself simply after completing an entire rotation. The Stepper motor also exhibited a high nominal torque value of 3Ncm, this higher torque value made this motor a suitable choice for driving a gear that is needed to turn a larger gear.

Specification

Table 14 5V Stepper Motor Hardware Specifications

5V STEPPER MOTOR	
PART NUMBER	XC4458
DESCRIPTION	Duinotech Arduino Compatible 5V Stepper Motor with Controller
STEPPER MOTOR SIZE	28mm diameter
NOMINAL TORQUE	3Ncm
STEP ANGLE	5.625 x 1/64
OPERATING VOLTAGE	5VDC
DIMENSIONS	35(L) x 32(W) x 10(H)

6.4.3. Arduino Code

This initial code was used for Arduino control of the stepper motor and FSR for the Ferris wheel segment of the Crazy Machine Project

```
#include <Stepper.h>

const int sensorPin = A0;    // Input pin for the FSR
const int stepsPerRev = 64;  // Number of steps per revolution for the stepper motor

Stepper myStepper(stepsPerRev, 8, 10, 9, 11); // Initialise the stepper motor
int sensorValue = 0;                          // Variable for sensor value

void setup() {
 myStepper.setSpeed(350);  // Set the speed of the stepper motor
 Serial.begin(9600);       // Initialise serial communication
}

void loop() {

 sensorValue = analogRead(sensorPin); // Read the value from the fsr


 int force = map(sensorValue, 0, 1023, 0, 100); // Map the sensor value to a % force value:

 Serial.print("The force is: ");
 Serial.print(force);
 Serial.println("%");

 // Check if the force exceeds a certain threshold (manually toggle later)
 if (force > 60) {
   Serial.println("Applying force. Moving motor...");
   myStepper.step(-8000); // Move the motor by 8000 steps
   delay(10);          // Stability delay
   myStepper.step(8000); // Move the motor backwards by 8000 steps
   delay(10);          // Stability delay
 } else {
   Serial.println("Force removed. Stopping motor...");
   myStepper.step(0); // Stop the motor
   delay (10);
 }
}

```

The code includes the Stepper.h library for control of the stepper motor. The constants are defined as the Analog input pin for the FSR and the number of steps for the specific type of stepper motor, which is 64. The stepper motor is initialised with these parameters and a variable to store the sensor value readings.

In the setup function, the stepper motor speed is set to 350rpm, and serial communication is initialised at a baud rate of 9600.

In the loop function, readings from the FSR are taking using the analogRead function. This sensor value is then mapped to a percentage force value which ranges from 0 to 100, using the map function. The force value is continuously printed to the serial monitor for general debugging and testing purposes.

The code checks whether the force exceeds a threshold of 60, if it does then a message is printed and then stepper motor is commanded to move 8000 steps forwards and then 8000 steps backward with a short delay for stability purposes. If the force is below the threshold, a message indicating the force has been removed is printed and the motor is stopped by setting the final step count to 0.

This code worked to successfully control the stepper motor in response to the corresponding force applied to the FSR as it accurately reads the analog input from the FSR and converts this input into a force value using the mapping function.

6.5. Verification

6.5.1. Testing of Individual Electronic Components

Initially it was planned to use TinkerCad to test out circuits for individual components. However, limitations were encountered as TinkerCad does not support stepper motor and driver components. Despite this, TinkerCad was able to be used to test the circuit for the force sensing resistor (FSR). Since the Arduino cannot directly measure changes in resistance, a voltage divider circuit was created. This involved connecting a fixed resistor to the 5V pin of the Arduino and one of the analog input pins, with the FSR connected between the analog input pins and ground. By using a 10k ohm resistor, a full range of voltage readings from 0-5V was achieved, due to the acceptable voltage drop across the resistor. This configuration allows the Arduino to accurately measure the variation in voltage caused by changes in resistance of the FSR. Using readily available Arduino sample code, testing of the FSR and confirm the circuit worked as expected.

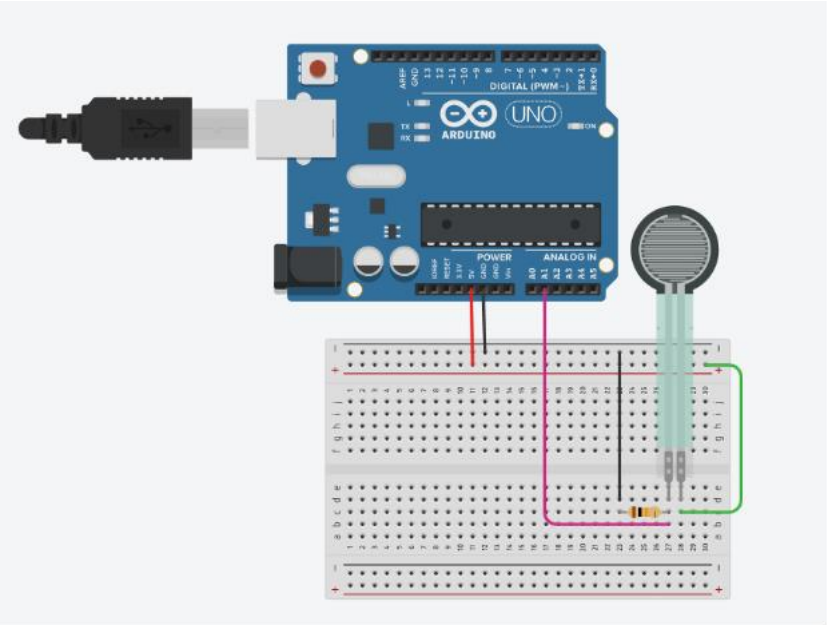


Figure 52 – Force Sensing Resistor Circuit Design

Proceeding this, testing the movement of the stepper motor when connected to the small gear to confirm it worked as expected. The 5V stepper motor is a 5-wire unipolar design. It came with a ULN2003 driver that is able to be used with an Arduino. Both the ground pin on the driver and the 5V pin on the driver to the ground pin on the Arduino and the 5V pin on the

Arduino respectively, were connected. The driver features four input pins that were connected to the Arduino's digital I/O pins with the following pin assignments:
IN1 - Pin 8
IN2 - Pin 9
IN3 - Pin 10
IN4 - Pin 11

Using available Arduino sample code, adjustments were made to the following line of code from:

Stepper stepper(STEPS, 8, 9, 10, 11);

To the following:

Stepper stepper(STEPS, 8, 10, 9, 11);

A stepper motor was correctly configured to rotate both forwards and backwards. With the small gear attached affirmation of the correct operation proceeded.

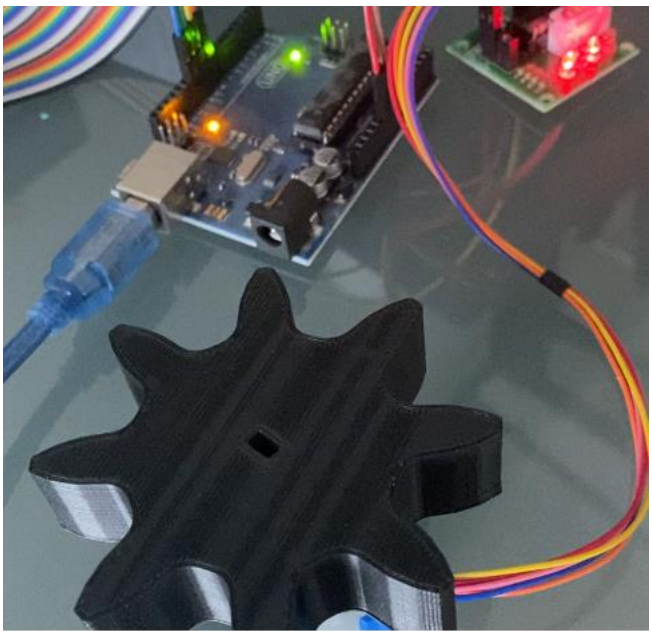


Figure 53 – Small Gear and Stepper Motor Testing

6.5.2. Testing of Electronic Components

After affirming that both components work individually, both parts were combined into a larger circuit where it was tested and confirmed that all components work together in a circuit.

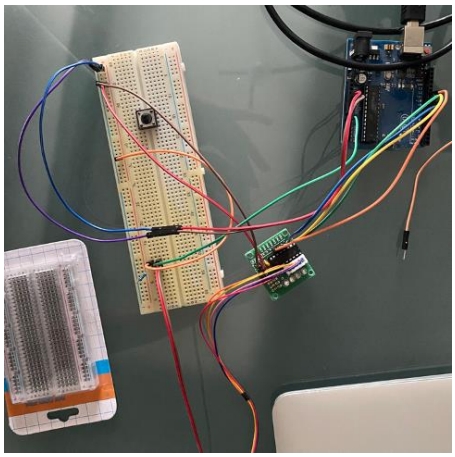


Figure 54 – Ferris Wheel Circuit

6.5.3. Testing of Electronic & Mechanical Components

All the 3D printed components were integrated to create a functional test benchtop. This setup allowed trialling the components running together, powered by the electronic components in the circuit. By doing so, it was possible to simulate the mechanisms in the Crazy Machine before the actual assembly procedure. This testing environment was important to identify any potential issues or faults as necessary adjustments to change the functionality of the system could occur.

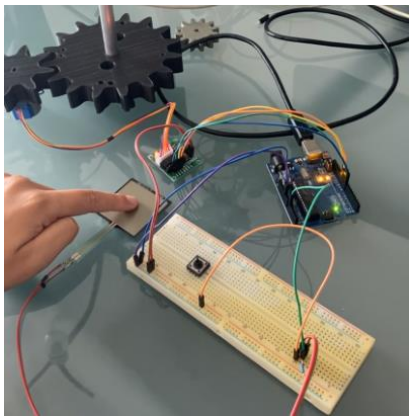


Figure 55 – Ferris Wheel Circuit Testing

When testing the Ferris Wheel segment in the Crazy Machine Project it was found that the overheating of the stepper motor was caused when it was run for excessively long periods of time, which occurred once during testing. To overcome this challenge, back up small gears

were printed in case this occurred again. This was not deemed a major challenge as the stepper motor would never be running for periods more than 1 minute at a time.

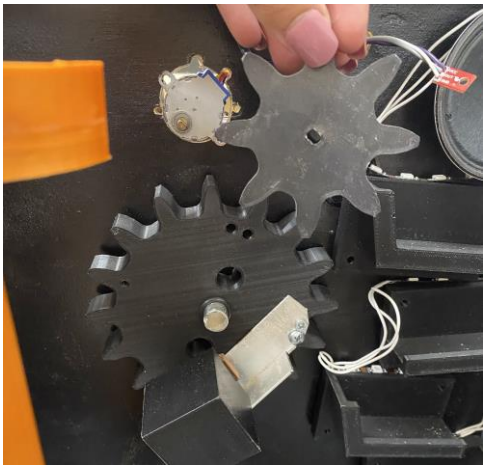


Figure 56 – Small Gear Stepper Motor Connection

7. Piano Slide

7.1. Assumptions

The following assumptions formed the foundation of designing the Ferris Wheel segment in the Crazy Machine Project:

Ball Splitter segment will deliver ball into the first Piano Key step

The most important assumption is that the ball splitter segment will efficiently deliver the ball to the first step of the piano key steps.

Other sections will stay in their allocated area–

The second assumption is that all other team members will cohere to their space allocations and not interfere with the motion of the ball through the Piano slide.

7.2. Decisions

After detailed design and testing of the conceptual design, the following decisions were made in regard to the Piano slide segment in the Crazy Machine Project:

7.3. Design Process

7.3.1. Design Plan

The Piano Slide segment of the Crazy Machine Project consists of seven individual keys. The ball falls into the first key, after it is navigated via the Ball Splitter segment, the ball rolls over each of the keys and a sensing element is activated. This activation will trigger a sound frequency for each key, which is played through a speaker. Simultaneously, corresponding LED lights provide a visual element to match the auditory element. The model comprises several 3D printed keys attached to the Piano Slide Stand.

The following Gantt Chart represents a detailed overview of the plan for development of the Piano Slide segment of the Crazy Machine Project.

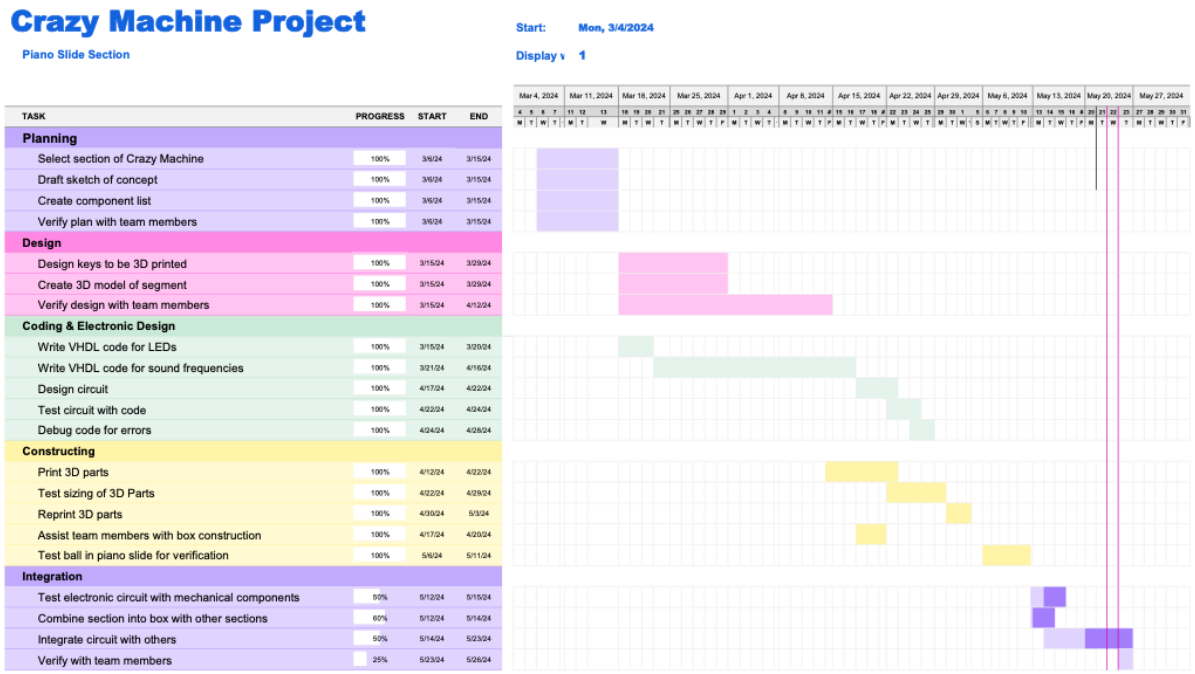


Figure 57 – Piano Slide Gantt Chart

7.3.2. Initial Design

The initial design of the Piano Slide segment for the Crazy Machine Project was designed using AutoCAD and 3D printed in ABS plastic material. The design was predicted to be confined within the space allocations; limited but the other team members sections. The Piano Slide was roughly designed

to fit within a top-view space of 150x170mm. The Initial design is displayed below in the following two figures.

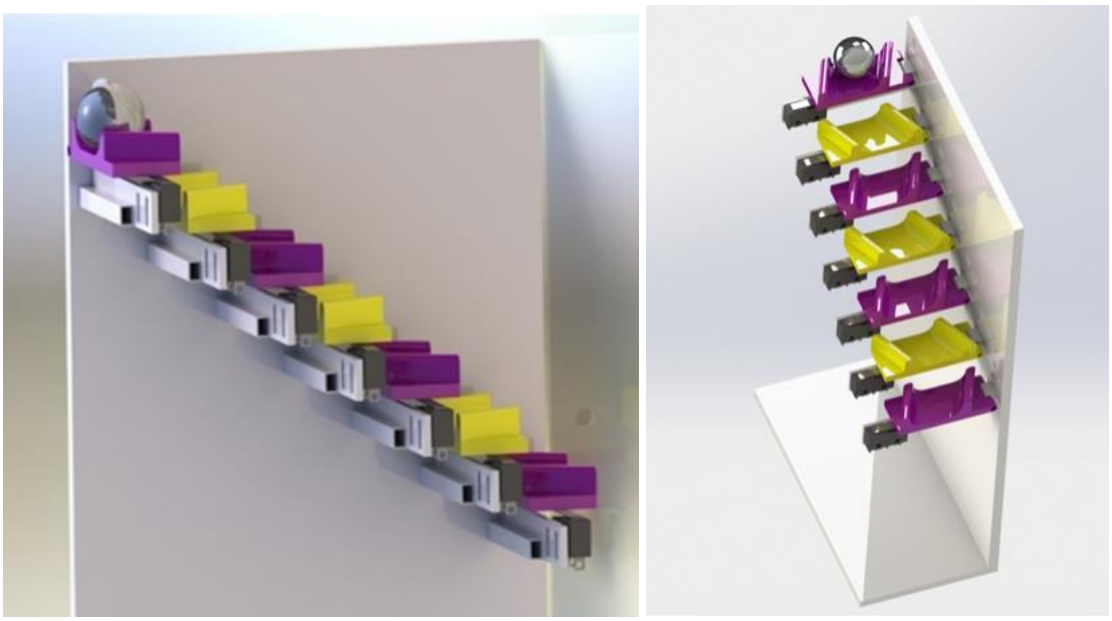


Figure 58 – Piano Slide Original Design

Within the initial design, the following design specifications were subdivided for the keys and for the stand. The seven keys were designed for the ball to roll through the grooves and drop onto the next key which allows for the reliable motion of the ball. The keys are slanted, on a 65-degree angle which is intended for aesthetic visual appeal and for system efficiency. An initial design of a 3mm diameter hole at the stand side which supports keys rotating around an axis.

The starting key specifications are shown below:

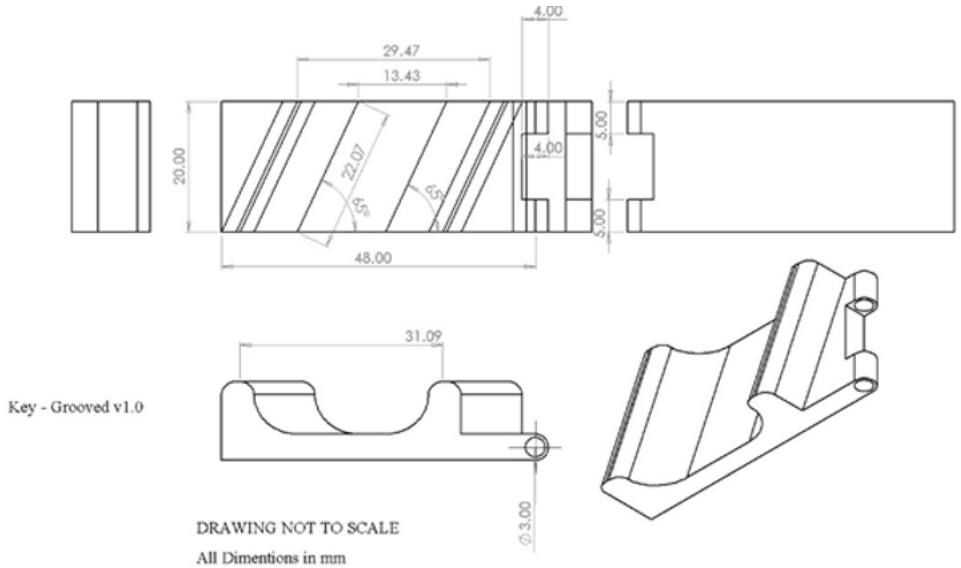


Figure 59 – Piano Slide Original Design 2

The second element in the design is a stand which supports both the piano keys and the limit switches. It contains allocated pathways for neatly managing wires that support the limit switches. A small 5.4mm hole is aligned above each piano key so that the LEDs can sit flush above each key when they are present.

The initial stand specifications are displayed below:

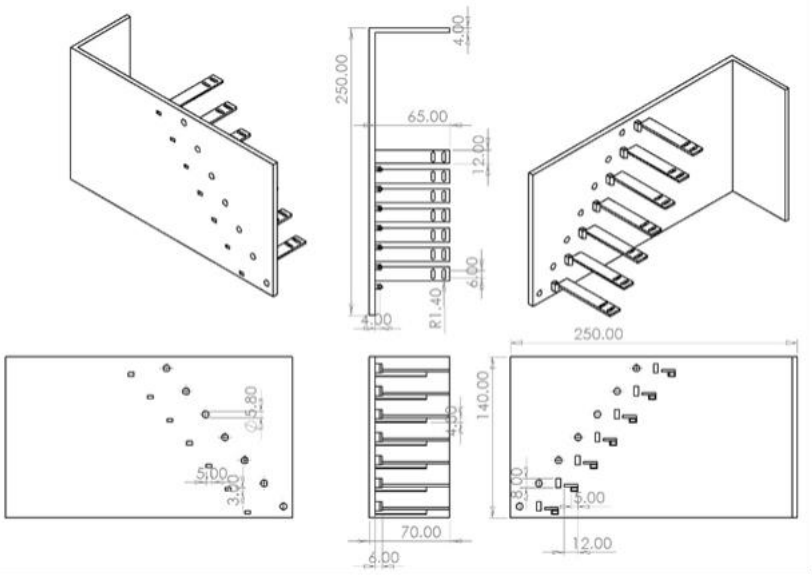


Figure 60– Piano Slide Original Design 3

7.3.3. Prototyping Design

During the design phase of the Crazy Machine Project for the Piano Slide segment, three major revisions were incurred due to various oversights in design.

In Revision One, the internal distance of the keys was too large, causing the ball to fall out rather than slide smoothly down the keys. To address this adversity, the design of the keys undertook a second revision which saw the variable size being increased to prevent the ball from falling through the keys. However, this revision created the further challenge whereby the internal diameter of the keys was far too small, hindering the passage of the ball.

In the final revision, a more balanced approach was found by increasing the height of the keys on the stand and reducing the variable size fillet. This revision ensured that the ball could slide smoothly through the keys without falling out and getting stuck, as initially planned. A final adjustment involved sanding down the keys to achieve the optimal fit for the ball

Through each stage of the building process, various tests were conducted to validate the design structure and ensure the ball was able to move through the keys as proposed.

7.3.4. Final Design

The final design of the piano keys is depicted below:

Figure 61 – Piano Slide Final Design

7.4. Implementation

7.4.1. Building

All major components, such as the keys and the stand, for the Piano Slide segment of the Crazy Machine Project were 3D printed. However, M2 sized bolts were acquired from United Fasteners to be the correct sizing for the limit switches. After all parts were purchased and printed, components were drilled into the stand and all electronic components were soldered to provide a structurally sound and durable design.

7.4.2. Hardware Specifications

For this section of the system, the requirement of the hardware is:

a. A sensing element to register a key being pressed by the ball.
b. A single speaker for audio feedback for each key.
c. LEDs for visual feedback for each key

The micro limit switch has a high sensitivity to mechanical motion. Furthermore, limit switches are renowned for their simplicity, with easy integration and requiring only one GPIO port on the DE-10 LITE FPGA and a ground connection. Each key is predicted to rest upon a single micro limit switch that is activated when a physical contact is applied. Micro limit switches are not impacted by external factors like infrared radiation and are able to withstand repeated use over time

The micro limit switch has the following characteristics:

Table 15 Limit Switch Hardware Specifications

Part Number	SM1038
Description	Miniature Micro Limit Switch with Lever
Operating Force	10 ± 4gf
Fixed Point (FP)	18.2 ± 1.5mm
Operating Point (OP)	11.6 ± 2.0mm
Rated Current	5A
Rated Voltage	125/250VAC

For auditory feedback, the chosen element was a speaker, which was be able to play different frequencies depending on the key pressed. The speaker specifications are displayed below:

Table 16 LED Hardware Specifications

Part Number	F5-C
Description	5mm integrate RGB LED lamp with automatic color fading

Table 17 LED Hardware Specifications 2

Emitting Colour	Wavelength (nm)	Luminous Intensity (mcd)	Current (mA)	Voltage (V)
Red	620-630	550-700	20	1.8-2.2
Green	515-530	1100-1400	20	3.0-3.3
Blue	465-475	200-400	20	3.2-3.4

7.4.3. Circuit Configuration

A 20k ohm pull-up resistor was used for the limit switches due to issues with a reliable connection. The LEDS were soldered in a daisy chain form and placed in their respective locations.

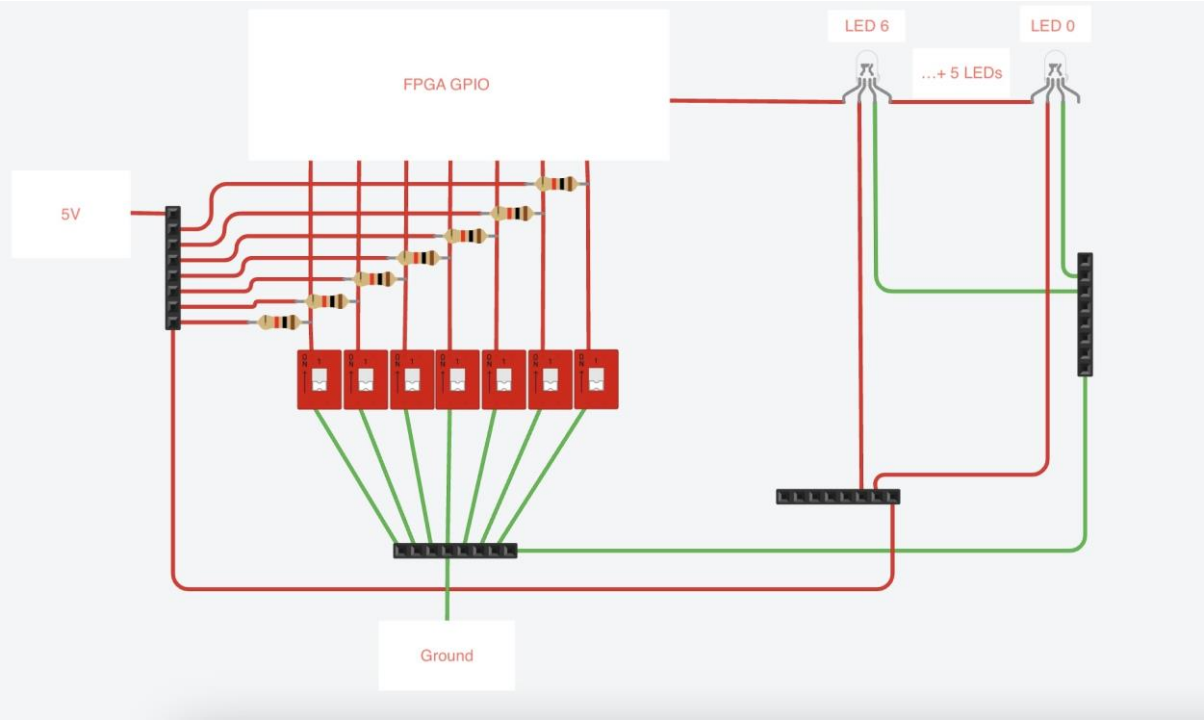


Figure 62 – Piano Slide Original Design 3

7.4.4. VHDL Design

The code for the piano slide consists of two main processes, the first process takes an input of seven keys and depending on the state of each key, determines if a given frequency for each key will be played.

Piano Audio Process

The piano audio process is simple in implementation, where a divided clock signal was used to assign the frequency for each key. The division constant was implemented through a counter where upon the counter reaching certain value, (the value is half the desired clock frequency), the signal will change. Each key was given a frequency of increasing frequency so the keys sound like ‘do, re mi, fa, so, la, ti’. This was done with trial and error with changing frequencies and listening to the sound produced.

```
1. Piano_audio : process(clk, note)
2.       begin
3.       if(rising_edge(clk)) then
4.       case note is
5.                                    when "000" =>
6.                                    if (counter >= 18180  ) then
7.                                                counter <= 0;
8.                                                temp_out <= not temp_out;
9.                                    else
10.                                               counter <= counter+ 1;
11.                                   end if;
12.                                   when "001" =>
13.                                   if (counter >= 16192) then   -- 440Hz do
14.                                               counter <= 0;
```

```
15.                                                 temp_out <= not temp_out;
16.                                 else
17.                                                 counter <= counter+ 1;
18.                                 end if;
19.                                 when "010" =>
20.                                 if (counter >= 15290) then    --494hz re
21.                                                 counter <= 0;
22.                                                 temp_out <= not temp_out;
23.                                 else
24.                                                 counter <= counter+ 1;
25.                                 end if;
26.                                 when "011" =>
27.                                 if (counter >= 13628) then   -- 593hz
28.                                                 counter <= 0;
29.                                                 temp_out <= not temp_out;
30.                                 else
31.                                                 counter <= counter+ 1;
32.                                 end if;
33.                                 when "100" =>
34.                                 if (counter >= 12140) then
35.                                                 counter <= 0;
36.                                                 temp_out <= not temp_out;
37.                                 else
38.                                                 counter <= counter+ 1;
39.                                 end if;
40.                                 when "101" =>
41.                                 if (counter >= 11461) then
42.                                                 counter <= 0;
43.                                                 temp_out <= not temp_out;
44.                                 else
45.                                                 counter <= counter+ 1;
46.                                 end if;
47.                                 when "110" =>
48.                                 if (counter >= 10203) then
49.                                                 counter <= 0;
50.                                                 temp_out <= not temp_out;
51.                                 else
52.                                                 counter <= counter+ 1;
53.                                 end if;
54.                                 when others =>
55.                                           temp_out <= '0';
56.                                           counter <= 0;
57.                 end case;
58.       end if;
59.
```

LED Colour and State Process

The LED Colour process is similar to the processes mentioned in Ladder_RTL for the Ladder LEDs and Button_RTL for the longer LED strip, with the only difference being that the LEDs were assigned to 7 colours of the rainbow in order, and their state was dependant on if their switch was turned on or not. The colour array was defined in the start of the architecture sections.

The for loop in code section below loops through all the LED addresses, and assigns a colour (red, green, blue) to the address in the WS2812B controller memory. Once all the addresses have been given a colour, a signal is given to the controller to render the data and wait a certain period before sending the next signal. This period was given 10,000 clock cycles at 50MHz which was larger than the ‘reset duration’ mentioned in the datasheet.

```
 1.                                  for i in 0 to 6 loop
 2.                                          if unsigned(colIdx) = to_unsigned(i-1,4) then
 3.                                                  if SW(i) = '0' then
 4.                                                                  data_red   <=
std_logic_vector(to_unsigned(colour(i,0), 8));
 5.                                                                  data_green <=
std_logic_vector(to_unsigned(colour(i,1), 8));
 6.                                                                  data_blue  <=
std_logic_vector(to_unsigned(colour(i,2), 8));
 7.                                                          else
 8.                                                                  data_red   <=
(others => '0');
 9.                                                                  data_green <=
(others => '0');
10.                                                                  data_blue  <=
(others => '0');
11.                                                          end if;
12.                                          end if;
13.                                 end loop;
14.
```

Testing and Debugging

The code was verified by testing if the LEDs were outputting the correct colour and state to the right LEDs. Initially, as each component was tested individually, the LEDs were tested by using the switches from the FPGA. The limit switches were also tested individually by mapping the GPIO port from the switch signal to the on board LEDs of the FPGA.

7.5. Verification

7.5.1. Testing of Mechanical Components

The mechanics of the Piano Slide segment were relatively unassuming and required minimal testing to ensure they worked as proposed in the initial design. In order to test the mechanical components of the piano slide, the design featured adjustable keys attached to the stand with hinges. The movement of the keys was tested by passing through them to confirm the correct range of motion. However, the initial hinges were found to be weak and frequently broke. To address this issue, the hinges were welded, providing the necessary strength and durability to prevent further breakages.

7.5.2. Testing of Electronic Components

To test the electronic components in the Piano Slide segment, certain factors were considered. While only digital limit switches were used, there were seven in the initial proposed design. The end of the limit switch levers were rated for 12g which led to the challenge of positioning them correctly so that the spring effect would be able to work

effectively. Through rigorous testing, the limit switches needed to be placed in precise positions to avoid being too close and inhibiting springing or being placed too far and being overly sensitive to the weight of the ball.

In the design for the Piano Slide segment, the limit switches a large amount of electrical interference which impact their reliability. To address this challenge, a pull-up resistor was connected to the FPGA side of the circuit to keep the signal at a high voltage level when the switch is deactivated. A sizeable resistor was further incorporated so that when the limit switch is activated, it created a direct pathway to ground, which ensured the signal is pulled to a low voltage level. This configuration mitigates the electrical interference and makes the limit switches reliable by evidently distinguishing between the inactive and activate states.

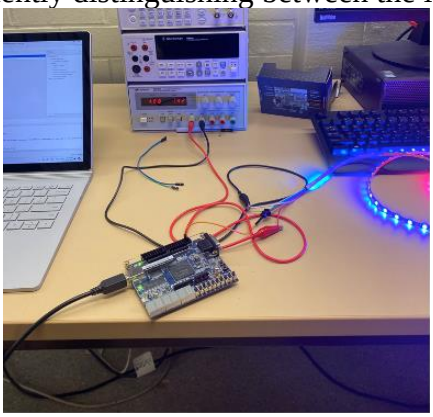


Figure 63 – Piano Slide LED Testing

Regarding the connection between the speaker and the FPGA, several adjustments were made to ensure a clear sound that is not subject to distortion. The speaker was tediously adjusted to avoid sound distortion through fine tuning the output signal from the FPGA. Furthermore, resistors were added in series with the signal to prevent the sound distortion. The resistors act as a form of signal condition and prevent unintentional noise. Finally, to achieve a high sound quality, the frequencies generated by the FPGA required testing and adjustments. By an experimental process, the ideal frequency range was selected.

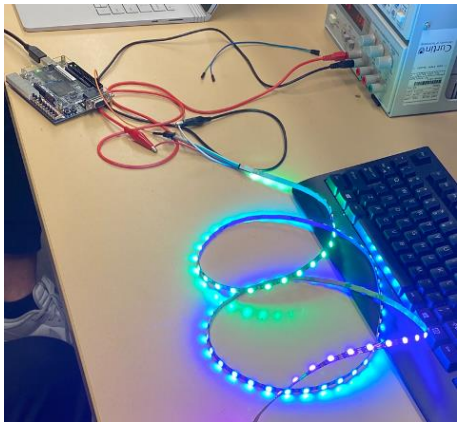


Figure 64 – Piano Slide LED Testing 2

For the setup of the second speaker, a schematic from the LM386 audio amplifier, a low-voltage power amplifier, datasheet was used as a reference to design the operation. A gain

potentiometer allowed for the adjustment of the amplifiers gain, to control the amplification levels of the audio signal and a line level potentiometer was used to adjust the input level signal to ensure the audio input is at an appropriate level prior to amplification. A high capacitance capacitor was added in series with the speaker to help filter out noise from the audio signal which resulted in a clean sound output. This configuration ensured that the sound quality was high, and the performance was replicable and reliable.

8. Pinball Ladder

The design process and competition of the Pinball Ladder section is covered within this section. For the Pinball Ladder section, the following Gantt Chat depicts the schedule for the design, implementation and coding timeline.

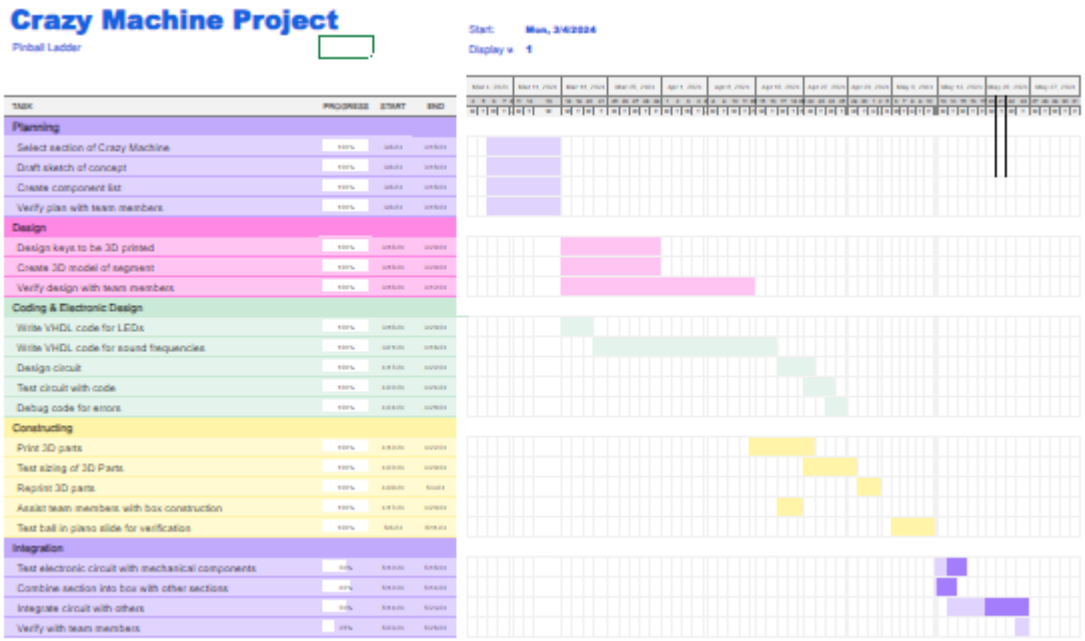


Figure 65 – Pinball Ladder Gantt Chart

8.1. Assumptions

The primary assumptions for the Pinball Ladder section are as follows:

Ball will drop evenly with equal time delay from each ladder:
It is critical for the creation of the code that the ball spends equal or approximate time in each ladder step. Since the main function of the LEDs will be based on the time delay rather than any specific input sensors that this assumption gives a foundation for the code to be written.

Space allocation:
The design of the Ladder Steps was specifically made based on the allocated area. Therefor any discrepancies between the allocation will result in the Ladder Steps not properly fitting into the section. However, for the Ladder Step design not all allocated space was used for considering any issues that may occur with the area allocation.

Quality building material component:
One of the early decisions made into the construction of the Ladder Steps as outlined in Status Report is 3D printing would be used. One concern regarding 3D printing is that quality of the material may lack. An assumption is made such that the quality of Ladder Steps will be excellent, and no wear and tear damage will occur over the required operation time.

8.2. Decisions

Addressable LEDs rather than single LEDs
Once major downside to using single LEDs is that it will require one input pin per LED. Therefor using a large quantity of LEDs is not suitable. Therefor the decision is made such that using an addressable LED would provide as a better alternative.

Using Addressable LED strips rather than single LEDs
As per the previous decision, single addressable LEDs were purchased. However, due the large number of wires involved the decision was made to use LED strips instead. This would reduce the number of wires and overall provide a cleaner outlook to the design.

FPGA over Arduino
The original concept was to originally use the Arduino to programme the LED strip. However, due to the incentive of the rubric the FPGA was used for the LED control. The Arduino would serve as a backup if the FPGA did not accomplish the proposed design.

8.3. Design Process

8.3.1. Initial Design

The initial design of the Pinball Ladder concept can be seen in Figure 33, which was inspired by the classic 2D ‘Pinabll’ game. There are 5 Ladder Steps with each Ladder having its own single LED. When the ball enters the module Sensor 1 is detected. Upon detection the LED will turn on, one by one in sequence. The delay of the turn on will be based on how fast the ball moves through the module.

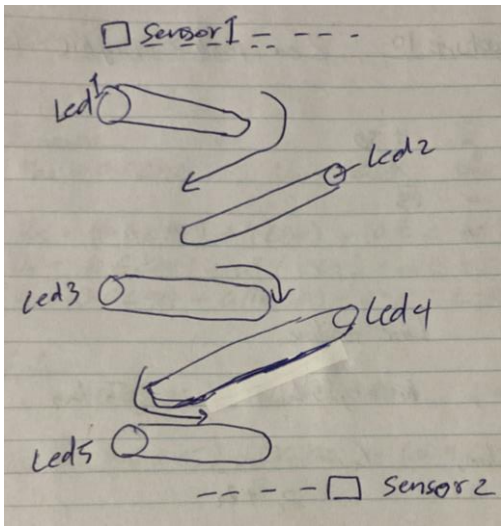


Figure 66 Pinball Ladder Original design

As stated in decision sections using non-addressable LED was replaced by using single addressable LEDs. Which would ultimately remove the need to need to using multiple input pins.

8.3.1.1. Ladder Step Design

One major issue with the original design concept is that the Ladder Steps is that the ball needs to be contained within the module and not fall out. With this intention in mind the 3D design concept was made as seen in Figure 34.

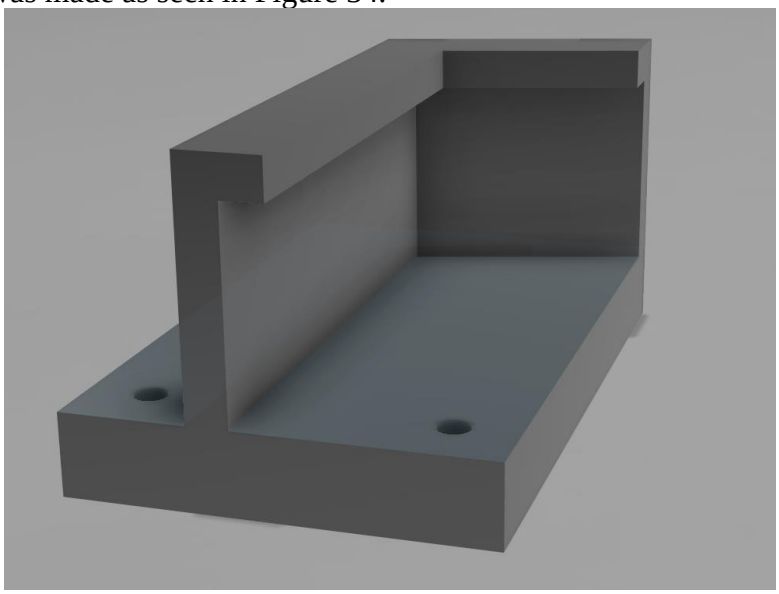


Figure 67 3D printed Ladder step

Figure 35 depicts one of 5 Ladder steps that were 3D printed. The Ladder steps has a slight original angle which will help the ball roll downwards. There is also a small wall on the side

facing outwards which prevents the ball from falling forwards. The Ladder steps are intentionally printed separately rather than a single complete module to help for any discrepancies such as area allocation. If the speed of the ball needs to be increased the Ladder Steps can be slightly angled downwards and vice versa, the Ladder Steps can be angled upwards to decrease the speed of the ball. The design considers the overall need for the group and any expected precautionary issues, such as area allocation and speeding up ball movement or slowing down.

8.3.2. Step Final Design

The Pinball Ladder initially featured a design with 5 ladder steps, each accompanied by an LED. However, several changes were made throughout the design and implementation process to improve functionality and address practical limitations.

Initially, 5 ladder steps were 3D printed according to the specifications shown in Figure 36. Due to vertical space limitations, only 4 of the 5 steps were placed in the module area, as seen in Figure 46. A significant modification involved angling the ladders slightly to ensure the ball movement smoothly down the steps. This adjustment was necessary because the initial design did not provide enough downward force for the ball to descend properly as not enough speed was available when placed perfectly straight.

Figure 68 Mounted Ladder steps

8.3.3. LED Design

As mentioned in the decisions the LEDs went through multiple iteration of changed. The final two iterations were the use of single addressable LEDs to addressable LED stripes. Originally, the LEDs were intended to be placed on the side of each ladder step, with 12 F5-C LEDs, two per step. However, the design was modified to place the LEDs on top of each Ladder step, this decision was made due to the limited space available on the side of the

module. This design unlimitedly proved more effective as it was an improvement to visual aesthetics. The LED type was also changed to ALITOVE WS2812B LED Strip, which reduced the wiring complexity. Each ladder step now has 4 LEDs, totalling 16 LEDs.

Using an LED strip simplified the wiring, as the 3 nodes (5V, G, Di) could be connected to each end. The challenge was cutting the strip into 4 pieces, each containing 4 LEDs, and soldering them back together. This intricate soldering required precision to avoid damaging the LEDs or causing shorts between nodes. Figure 37 shows the LED strip prior to being mounted on the ladder steps. The final configuration, where each LED segment is connected is shown in Figure 38, prior to being hidden behind tape for demonstration. Due to the reduction from 5 ladders steps to 4 the LED strip needed to be readjusted. Initially the full strip consisted of 20 LEDs however 4 LEDs needed to be removed to make an adjustment to meet the 4 Ladder steps.

Figure 69 Pinball Ladder LED strip

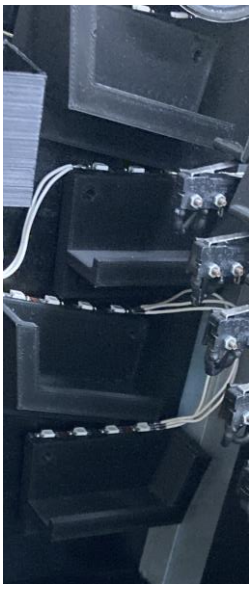


Figure 70 Pinball Ladder LED strip

These modifications improved the overall design, enhancing both the mechanical and electrical performance of the Pinball Ladder as well as overall improvement to visual aesthetics.

8.3.4. Hardware Specifications

As stated in the previous section the final iteration of LEDs used are WS2812B LED Strip, 5mm Width Super Narrow RGB Addressable LED Light Strip. The two Digital Line sensors have not changed since the original design concept.

Table 18 Digital Line Sensor Hardware Specifications

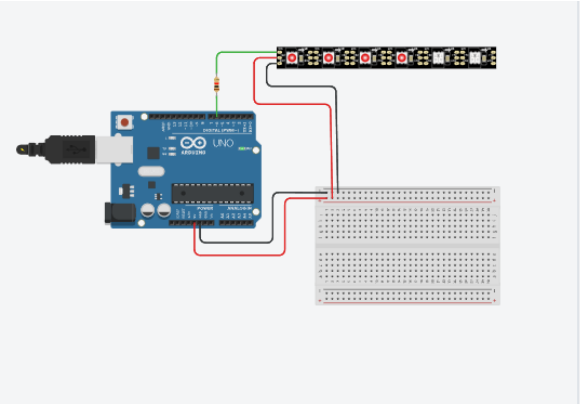
2 X Digital line sensor	
PART NUMBER	QRE1113
Description	Line Sensor Breakout - (Digital)
Operating voltage	5VDC
Supply current	25mA
Optimal Sensing Distance	3mm
Dimensions	0.76cm x 1.4cm

Table 19 LED Strip Hardware Specifications

WS2812B LED Strip	
PART NUMBER	Purchased from Amazon (PN NA)
Description	WS2812B LED Strip, 5mm Width Super Narrow RGB Addressable LED Light Strip
Operating voltage	5VDC
NUm	Multi colour RGB
Size	120 LED PK

8.3.5. Verification and implementation

As stated in the Decision section the LED were originally intended to be coded for the Arduino. However, the decision was made to move to the FPGA for more utilization of the FPGA over the Arduino. Figure 39 depicts a Tinkercad simulation of an LED strip connected to an Arduino, with a small snippet of the code section. The following Arduino code served as a backup in case the LEDs were not able to be utilized by the FPGA. However, the FPGA was utilized and the Arduino code was not used.



```
// Pin definitions
#define LED_PIN     6
#define BUTTON_PIN  2

// Constants
#define NUM_LEDS    6  // Number of LEDs

Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LEDS, LED_PIN,

bool lastButtonState = HIGH;  // Assume button is not pressed i
bool ledsOn = false;          // Track whether LEDs are current

void setup() {
  // Initialize the LED strip
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'

  // Configure the button pin as an input with pull-up resistor
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  bool currentButtonState = digitalRead(BUTTON_PIN);
```

Figure 40 Unused code

Figure 41 depicts the LED strip testing directly on the FPGA, the code running is complimentary to section 6.2.2.7, Pinball Ladder code. Originally the code was tested using the internal single addressable LEDs and the sensors were replaced with the switches, due to delivery delays in the components. Second iteration of testing was done with the addressable LED strips which would ultimately confirm the ability of the WS2812B controller code in section 6.2.2.6. The Pinball Ladder section FPGA code has been documented in detail in the in the section below 6.2.2.7.

Figure 72 FPGA coded LED strip.

8.3.6. FPGA Design

The ladder code is split into two processes sections, one of them controls the state of the LEDs when the delay is larger than a certain constant, and the other controls LED colour processing.

LED State Process

The process is a state machine with two states, idle and ball detected. Initially the state is idle/standby which has all the LEDs off. When the first line sensor is triggered, it transitions to the ball detected state. The ball detected state turns the LEDs 0 to 15 on one by one with an

LED delay timer, this delay is based on how long the LED will take to clear the section and will be determined in the testing and debugging section. The delay operates on a clock signal which is connected to 50Hz. In this mode the LEDs are turned on in sequence as the LED timer increments through each LED, as shown in the for loop below.

If the second switch is triggered or if the timer exceeds the designated maximum operation time, then the state returns to original idle or standby mode. The maximum operation time feature was added for a failsafe if the second IR sensor did not detect the ball.

```
 1. led_state_process : process(clk, ir_signal)
 2.       begin
 3.                          if rising_edge( CLK ) then
 4.                                   case state is
 5.                                            when s_idle =>
 6.
 7.                                                     led_state <= (others => '0');
 8.
 9.                                                     if ir_signal(0) = '1' then
10.                                                              state <= s_balldetected;

11.                                                     else
12.                                                              state <= s_idle;
13.                                                     end if;

14.
15.                                            when s_balldetected =>

16.                                                     if ir_signal(1) = '1' or LED_Timer >
LED_maxtime- 1 then
17.                                                              state <= s_idle;
18.                                                              LED_Timer <= 0;

19.                                                     else
20.                                                              LED_Timer <= LED_Timer + 1;
21.                                                              state <= s_balldetected;
22.                                                              for i in 0 to 15 loop
23.                                                                       if LED_lagtime*i <
LED_Timer then -- if led_lagtime(i)
24.
          led_state(i) <= '1';
25.                                                                                 else
26.
          led_state(i) <= '0';
27.                                                                                 end if;

28.                                                              end loop;

29.                                                     end if;

30.                                            when others =>
31.                                                     state <= s_idle;
32.                                   end case;
33.                          end if;
34.       end process led_state_process;
35.
```

LED Colour Process

The colour of each LEDs are determined in the following for loop. The WS2812B code is structured such that it cycles through each LED address and then a signal is given to write a colour to the LEDs address, and it is stored in the memory of the WS2812B component, and once all LEDs have been given their colours, a signal is given to render the data, and output the LED signal to the LED strip. The number of bits for the address are given in this function, and for 16 LEDs it returns 4 bits.

```
1.        signal addr          : std_logic_vector(integer(ceil(log2(real(length - 1)))) downto
0) := (others => '1');
2.
```

To turn off an LED its values for red, green and blue are set to 0,0,0. With each iteration of the for loop, it checks if the current index of the LED and writes a corresponding colour, which is given by the colour array mentioned in the start of the architecture section, and the state are mentioned in the LED state process. This process repeats until all 16 LEDs are written to, ensuring only the LED matching the specific index is updated.

```
 1. for i in 0 to 15 loop
 2.       if unsigned(colIdx) = to_unsigned(i-1,5) then
 3.                if led_state(i) = '1' then
 4.                                   data_red   <= std_logic_vector(to_unsigned(colour(i,1),
8));
 5.                                   data_green <= std_logic_vector(to_unsigned(colour(i,0),
8));
 6.                                   data_blue  <= std_logic_vector(to_unsigned(colour(i,2),
8));
 7.                else
 8.                                   data_red   <= (others => '0');
 9.                                   data_green <= (others => '0');
10.                                   data_blue  <= (others => '0');

11.               end if;
12.      end if;
13. end loop;
14.
```

8.3.6.1.1. Testing and Debugging

The code written was tested and debugged using a strip of 5 LEDs in length. Initially, the LED strip was tested with the input being the FPGA onboard switches instead of the IR sensor, as it was convenient and will provide the rising edge signal that is also produced by the IR sensor VHDL component.

For the testing and debugging of this section came with ensuring that the colours observed in the colour array that was defined in the code matched the colours on the LED strip, and their address. Also, the state of the LEDs was tested rigorously until the correct output had been observed. Most of the testing for this section is based on the led delay. This exact timer is determined after testing the ball drop and measuring the time in the module. This can only be performed when the section was complete, and was one of the final tasks.

The final value of the LED delay constant was 480000 clocks, which corresponds to 0.096 seconds. This means that the total time taken for the ball to reach the entire path was 1.536 seconds.

```
1.        constant LED_lagtime    : natural := 4800000; -- 25000000 is 0.5s, 2500000 is 0.05s
```

The failsafe, was given a value of 15 seconds.

```
1.        constant LED_maxtime    : natural := 25000000*30; -- 15s
```

9. Ball Return and Decorative Elements

9.1. Assumptions

The primary assumptions for the Pinball Ladder section are as follows:

The pinball ladder will drop the ball into the ball return
There will be a gap between the pinball ladder and the ball return and therefore a ramp may need to be created to guide the ball to the return. The structure will also need to be strong enough to support the ball being dropped from a height onto the return ramp.

Other sections will stay in their allocated area
Since other members have set allocated space the ball return ramp will be shaped in order to not interfere with the other sections.

The spiral lift entrance will be facing 90° from the ball return
A suitable guider will need to be designed in order to smoothly navigate the ball from the ball return into the lift.

9.2. Decisions

The ball return will be constructed from cardboard
To easily construct and alter the design, the return ramp is being constructed from cardboard. This also has the benefit of being readily available and extremely cost effective.

The ball return will be spray painted gold
In order to suitably ensure the ball return matches the theme of the machine, the ball return will be spray painted gold which gives it a vibrant and appealing look.

A 12v LED strip will be placed on the underside of the ball return
Due to the black floor of the machine, having the LED on the underside of the ball return will ensure the lights are visible and easily seen.

9.3. Design Process

9.3.1. Initial Design

The original design for the ball return was for it to be 3D printed, however this plan has been altered and instead will be constructed from cardboard. This has been done due to the design being very simple, therefore very easy to construct from cardboard, being a much more environmentally sustainable material then 3D printer filament and the ability to easy adjust the cardboard design. The cardboard has been sourced from boxes that were destined for the landfall, compared to the alternative of 3D printer filament which is plastic and much harder to recycle. Throughout the design process, having a easily altered ramp has been useful as it allows for quick changes to be made without having to reprint the entire design. Changes such as adjusting the tilt of the ramp, making the length shorter, adjusting the corner turns and fine tuning the guide into the lift entrance.

9.3.2. Final Design

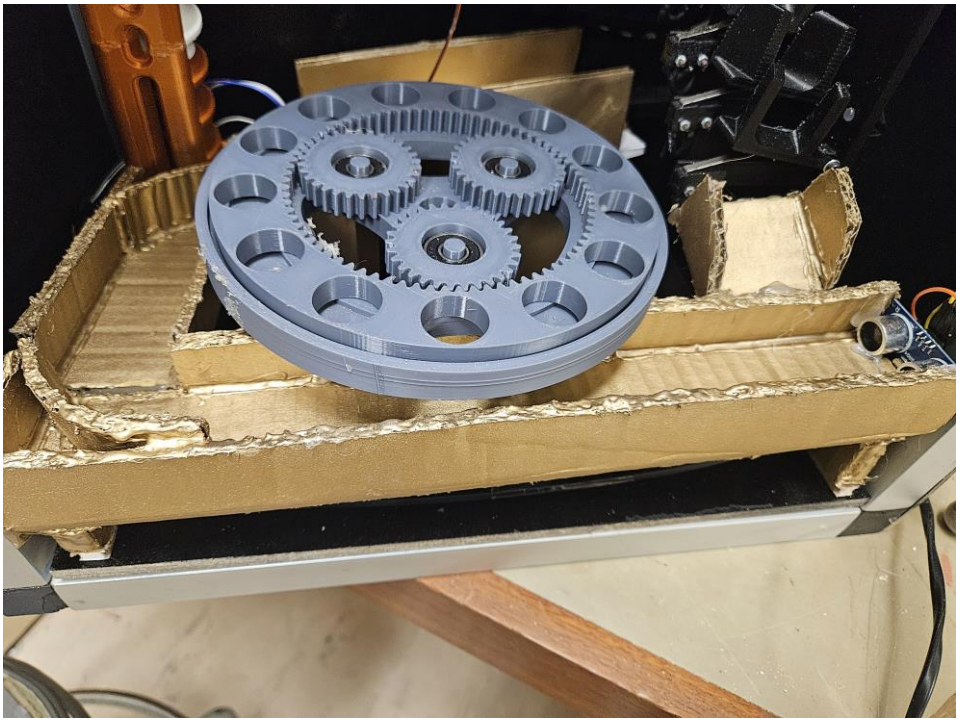


Figure 73 Ball Return Final Design

For the decoration component, a 12V Analog led strip has been used on the underside of the ball return which will start turned off and the brightness will be increased as the ball moves down the ramp. To detect this a ultrasonic sensor has been utilized which has been placed on the end the ball enters at. The original plan was to have multiple LED's which individually lit up as the ball moved down, however there are currently not enough available pins on the

Arduino for the amount of LEDs required and the only addressable LED strip we have is currently in use for another section

Figure 69 Final Ball Return Design

9.4. Hardware Specifications

Table 20 Ultrasonic Sensor Hardware Specifications

1 X Ultrasonic sensor	
PART NUMBER	HC-SR04
Description	Ultrasonic Distance Sensor
Operating voltage	5VDC
Angle of detection	15°
Optimal Sensing Distance	2cm – 450cm
Dimensions	45mm x 20mm x 13mm

Table 21 Analog LED Strip Hardware Specifications

1 X led strip	
PART NUMBER	N/A
Description	Analog RGB LED strip
Operating voltage	12V
Pin connections	Vcc, Red, Blue, Green
Dimensions	10mm x 900mm

Table 22 MOSFET Hardware Specifications

3 X n-channel mosfet	
PART NUMBER	MTP3055VL
Description	N Channel MOSFET Package:TO-220
voltage Rating	60V
current rating	12A
Dimensions	10mm x 4.5mm x 15mm

9.5. Circuit Diagram

The ultrasonic sensor has 4 pins, Vcc, Ground, Trig and Echo which are all connected to the arduino. Vcc provides 5v to the sensor to power it, while trigger sends out an ultrasonic wave and echo receives the ultrasonic waves once it’s reflected back to the sensor.

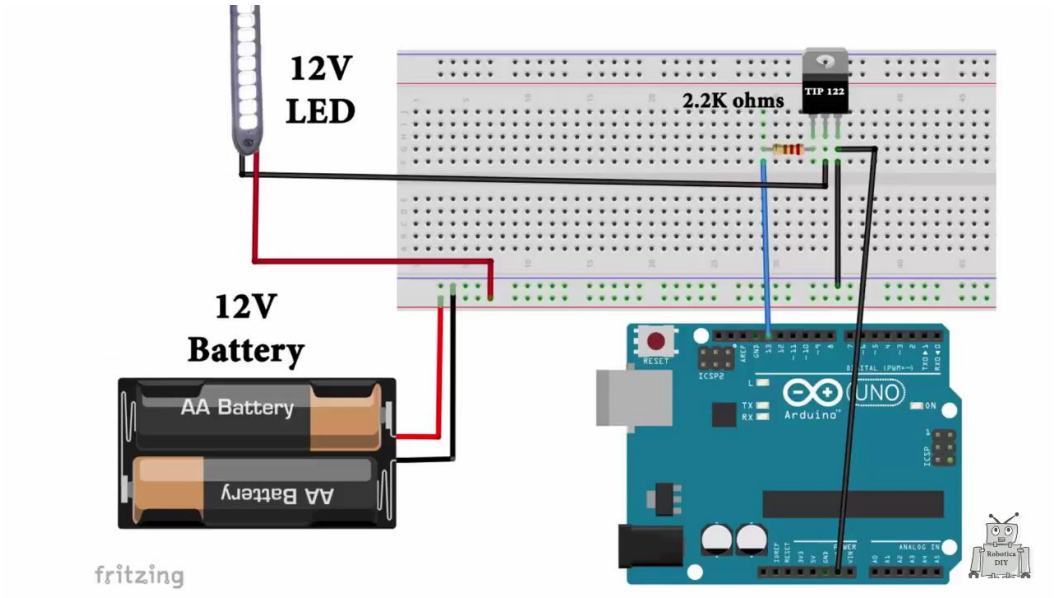


Figure 74 LED Circuit Diagram

The 12V analog LED strip has 4 pins, Vcc, Red, Green and Blue. Vcc provides 12v of power to light up the strip of LEDs and the other pins are used to control the control the individual LED colours. The RGB pins are connected to the Arduino output PWM pins, in order for PWM to be utilized which is how the brightness is controlled. However, they cannot be connected directly to the Arduino since the pins cannot output enough voltage in order to control the LEDs and instead each pin uses a MOSFET, acting as a switch which allows the Arduino to control the 12v to the LED strip. A N-type MOSFET has been since this will be switched on when the gate to source voltage is positive.
The above figure is a clear view of how the circuit is connected however instead of a mono LED, a RGB LED is used meaning there will be an additional 2 pins coming out of the LED which will require an additional 2 MOSFETS. The resistor will also not be required as the LED strip that is being used already has resistors embedded within it. The 12v source will not be a battery and instead will be supplied from the power supply.

9.6. Arduino Code

9.6.1. Ultrasonic Sensor Code

```
1.   digitalWrite(trigPin, LOW);
2.   delayMicroseconds(2);
3.   digitalWrite(trigPin, HIGH);
4.   delayMicroseconds(10);
5.   digitalWrite(trigPin, LOW);
6.   duration = pulseIn(echoPin, HIGH);
7.   distance = duration * 0.0343 / 2;
```

The above code snippet is how the ultrasonic sensor works. The trigger pin is held high for 10 microseconds, which sends out a ultrasonic wave burst. The pulseIn function is used to find

the duration the echo pin is high for, which is how long it takes for the ultrasonic wave to be reflected back to the sensor. The equation duration*0.0343/2 is then used to return the distance from the ultrasonic sensor.

9.6.2. LED Brightness code

```
converted = map(distance,0,30,0,255);
if (converted > 255) {
  converted = 255;
}
analogWrite(REDPIN, converted);
if (converted > previous) {
  for (g = previous; g < converted; g = g+1) {
    analogWrite(REDPIN, g);
    analogWrite(GREENPIN, g);
    delay(FADESPEED);
    g = g+1;
    previous = g;
    }
}
 else if (converted == previous) {
    delay(1);
    }
 else {
    for (g = previous; g > converted; g = g-1) {
      analogWrite(REDPIN, g);
      analogWrite(GREENPIN, g);
      delay(FADESPEED);
      previous = g;
     }
    }
```

The above code snippet shows how the brightness of the LED is changed. The brightness works by using PWM, where a signal is sent out at a specific duty cycle (0 to 255). The distance from the ultrasonic is first mapped from the distance of the ramp to the duty cycle range (giving a equivalent range from the distance into the duty cycle). A if statement checks to see whether this value is greater then or less then the previous seen value from the ultrasonic sensor, if its greater the brightness is increased and if its less then the brightness is decreased. Both these paths work the same way, with the difference being if the brightness is being increased the duty cycle is increased and if the decreased if the brightness is being decreased. To ensure a smooth fade transition is achieved, a for loop is used to change the duty value by 1 increment with a delay until the desired value is reached. Using the analogWrite() function the brightness can be changed by setting the duty cycle and different colour combinations can be achieved by mixing the strength of red green and blue.

9.6.3. Testing and Debugging

During the testing phase of the section, various parts of the code were tested in depth to ensure the code runs consistently and with no errors. These include testing the length read from the ultrasonic sensor in the ball return, finding the best placement of the ultrasonic sensor, testing different delay times for the fade to find the smoothest looking that didn't have a to large delay while the ball is moving.

The circuit was first constructed and tested on a separate bread board and Arduino from the main ones on the machine. This made it easy to debug and issues that came up and ensured that the circuit ran as expected. Once this was confirmed the components were installed into the ball return and some fine tuning was done, such as adjusting the LED fade amount and the distance values for when the LED toggles on.

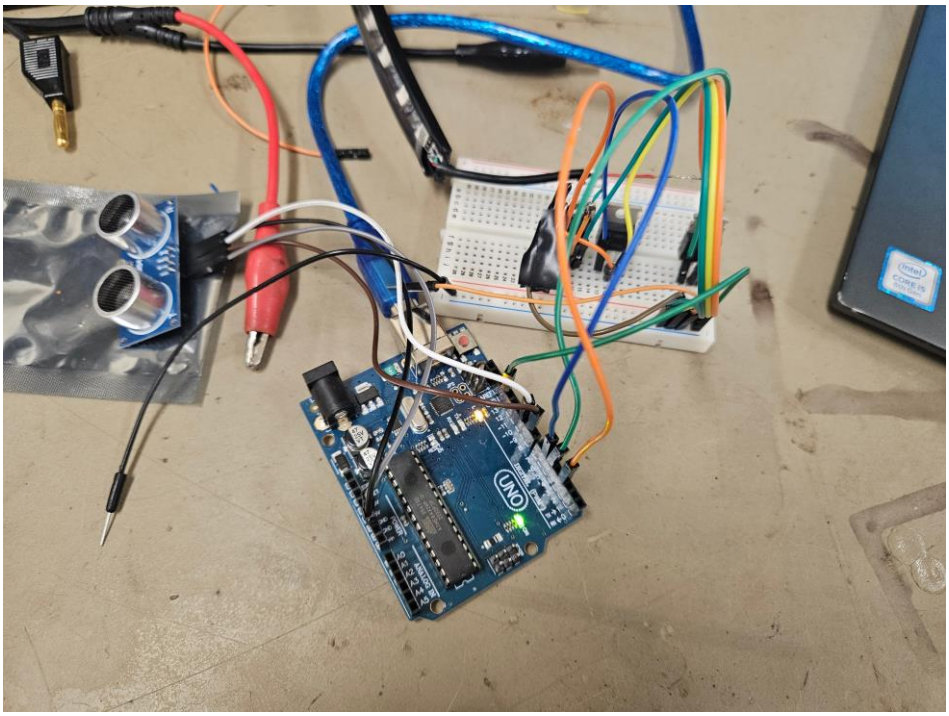


Figure 75 Circuit Configuration

9.7. LED decorations

There is no initial design for this concept. Instead, it was an enhancement to an already existing idea to decorate the interior of the box with LEDs.

For the LED decorations, ALITOVE WS2812B LED Strip, 5mm Width Super Narrow RGB Addressable LED Light Strip were used. The placement of the LEDs can be seen in figure 46. The LEDs are placed beneath the metal railing of the box. The LED strip cover the entire railing coming together in 360 degrees. Due to the rectangular shape of the box the LEDs strip needed to be cut into 4 different pieces. The longer side of the rectangular railing has 20 LEDs, and the shorter sides have 14. A total of 68 LEDs were used. Difficulties relating to the soldering were mentioned in the previous Pinball Ladder section.

Figure 76 Decoration LED strip mounting

10. Integration Overview

To ensure that the Crazy Machine project progressed smoothly, segment divisions were planned thoroughly to prevent obstruction and overlapping. Each team member was assigned set segments based on earlier area allocations and weekly meetings were held to ensure synchronisation of design. Following plans designated in both the overall Gantt Chart and individual Gantt Charts, our approach was structured and incurred minimal challenges.

FPGA and Arduino were used to create a cohesive project whereby integration between the two was established.

For our integration strategy comprised of combining FPGA and Arduino to create a cohesive project. To integrate these components, we established a clear communication protocol between the FPGA and Arduino.

10.1. FPGA Design

The individually tested VHDL defined components were combined to make an overall top-level entity, TOP.vhd. In the Crazy Machine design, VHDL was used to construct several components that interface with various sensors and actuators as well as FPGA-Arduino communication. This section describes the whole system, and how they are connected. The following components were used in our VHDL design and have been detailed in various sections of this report. This section of the report will focus on their final integration in the top-level entity used for the crazy machine.

- UART transmitter (4.3.2.1.1).
- UART receiver (4.3.2.1.1).
- WS2812B controller code (4.3.2.3).
- WS2812B rainbow sample code.
- Button debouncing and toggling (4.3.2.2).
- Digital line sensor code (4.3.2.4).
- Button code to control button LEDs using Button debouncing and toggling as well as the large addressable LED strip and send a byte to the Arduino on rising edge of the button toggle (4.3.2.6).
- Piano slide code which used WS2812b to control LED colours and status as well as small speaker frequency (7.4.4).
- Ladder code which turns on each led in the ladder run based on a time delay after a line sensor is triggered (8.3.6).
- Servo PWM control code (4.3.2.5).

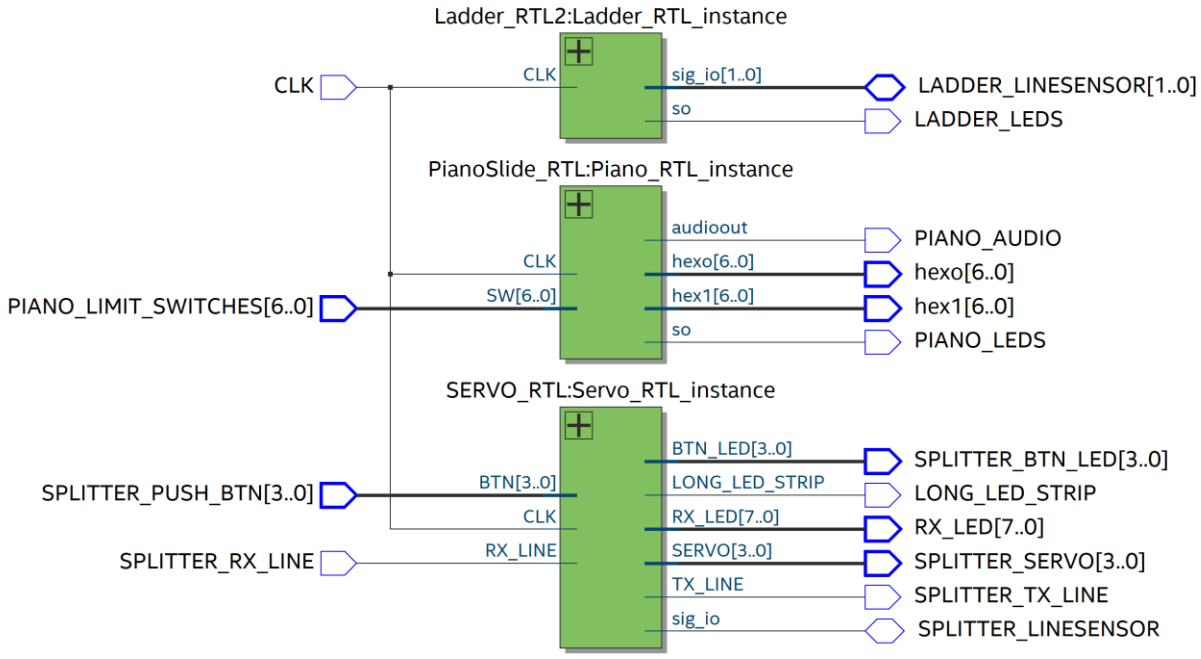


Figure 77 FPGA Top Level Entity Block Diagram

10.1.1. Pin Assignments

After all the components had been connected in a final top-level entity, pins were assigned according to the following table:

GPIO 1:	ODDS ROW 1						
Signal	Name	FPGA	No.	Description	I/O		
GPIO_1	PIN_W7	GPIO	[1]	3.3-V	LVTTL	SPLITTER_BTN_LED[0]	
GPIO_3	PIN_W8	GPIO	[3]	3.3-V	LVTTL	SPLITTER_BTN_LED[1]	
GPIO_5	PIN_W9	GPIO	[5]	3.3-V	LVTTL	SPLITTER_BTN_LED[2]	
GPIO_7	PIN_W10	GPIO	[7]	3.3-V	LVTTL	SPLITTER_BTN_LED[3]	
GPIO_9	PIN_AB13	GPIO	[9]	3.3-V	LVTTL	SPLITTER_PUSH_BTN[0]	
GPIO_11	PIN_W13	GPIO	[11]	3.3-V	LVTTL	SPLITTER_PUSH_BTN[1]	
GPIO_13	PIN_AA15	GPIO	[13]	3.3-V	LVTTL	SPLITTER_PUSH_BTN[2]	
GPIO_15	PIN_V5	GPIO	[15]	3.3-V	LVTTL	SPLITTER_PUSH_BTN[3]	
GPIO_17	PIN_Y11	GPIO	[17]	3.3-V	LVTTL	SPLITTER_SERVO[3]	
GPIO_19	PIN_W11	GPIO	[19]	3.3-V	LVTTL	SPLITTER_SERVO[2]	
GPIO_21	PIN_AA10	GPIO	[21]	3.3-V	LVTTL	SPLITTER_SERVO[1]	
GPIO_23	PIN_Y8	GPIO	[23]	3.3-V	LVTTL	SPLITTER_SERVO[0]	
GPIO_25	PIN_Y7	GPIO	[25]	3.3-V	LVTTL	PIANO_AUDIO	
GPIO_27	PIN_Y6	GPIO	[27]	3.3-V	LVTTL	LADDER_LINESENSOR[1]	
GPIO_29	PIN_Y5	GPIO	[29]	3.3-V	LVTTL	LADDER_LEDS	
GPIO_31	PIN_Y4	GPIO	[31]	3.3-V	LVTTL	LADDER_LINESENSOR[0]	
GPIO_33	PIN_Y3	GPIO	[33]	3.3-V	LVTTL	LONG_LED_STRIP	
GPIO_35	PIN_AA2	GPIO	[35]	3.3-V	LVTTL	SPLITTER_RX_LINE	

GPIO 1:	ODDS ROW 1						
Signal	Name	FPGA	No.	Description	I/O		
GPIO_1	PIN_W7	GPIO	[1]	3.3-V	LVTTL	SPLITTER_BTN_LED[0]	
GPIO_3	PIN_W8	GPIO	[3]	3.3-V	LVTTL	SPLITTER_BTN_LED[1]	
GPIO_5	PIN_W9	GPIO	[5]	3.3-V	LVTTL	SPLITTER_BTN_LED[2]	
GPIO_7	PIN_W10	GPIO	[7]	3.3-V	LVTTL	SPLITTER_BTN_LED[3]	
GPIO_9	PIN_AB13	GPIO	[9]	3.3-V	LVTTL	SPLITTER_PUSH_BTN[0]	
GPIO_11	PIN_W13	GPIO	[11]	3.3-V	LVTTL	SPLITTER_PUSH_BTN[1]	
GPIO_13	PIN_AA15	GPIO	[13]	3.3-V	LVTTL	SPLITTER_PUSH_BTN[2]	
GPIO_15	PIN_V5	GPIO	[15]	3.3-V	LVTTL	SPLITTER_PUSH_BTN[3]	
GPIO_17	PIN_Y11	GPIO	[17]	3.3-V	LVTTL	SPLITTER_SERVO[3]	

GPIO_19	PIN_W11	GPIO	[19]	3.3-V	LVTTL	SPLITTER_SERVO[2]
GPIO_21	PIN_AA10	GPIO	[21]	3.3-V	LVTTL	SPLITTER_SERVO[1]
GPIO_23	PIN_Y8	GPIO	[23]	3.3-V	LVTTL	SPLITTER_SERVO[0]
GPIO_25	PIN_Y7	GPIO	[25]	3.3-V	LVTTL	PIANO_AUDIO
GPIO_27	PIN_Y6	GPIO	[27]	3.3-V	LVTTL	LADDER_LINESENSOR[1]
GPIO_29	PIN_Y5	GPIO	[29]	3.3-V	LVTTL	LADDER_LEDS
GPIO_31	PIN_Y4	GPIO	[31]	3.3-V	LVTTL	LADDER_LINESENSOR[0]
GPIO_33	PIN_Y3	GPIO	[33]	3.3-V	LVTTL	LONG_LED_STRIP
GPIO_35	PIN_AA2	GPIO	[35]	3.3-V	LVTTL	SPLITTER_RX_LINE

GPIO 2:	EVENS ROW 2					
GPIO_0	PIN_V10	GPIO	[0]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[6]
GPIO_2	PIN_V9	GPIO	[2]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[5]
GPIO_4	PIN_V8	GPIO	[4]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[4]
GPIO_6	PIN_V7	GPIO	[6]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[3]
GPIO_8	PIN_W6	GPIO	[8]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[2]
GPIO_10	PIN_W5	GPIO	[10]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[1]
GPIO_12	PIN_AA14	GPIO	[12]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[0]
GPIO_14	PIN_W12	GPIO	[14]	3.3-V	LVTTL	PIANO_LEDS
GPIO_16	PIN_AB12	GPIO	[16]	3.3-V	LVTTL	SPLITTER_LINESENSOR
GPIO_18	PIN_AB11	GPIO	[18]	3.3-V	LVTTL	
GPIO_20	PIN_AB10	GPIO	[20]	3.3-V	LVTTL	
GPIO_22	PIN_AA9	GPIO	[22]	3.3-V	LVTTL	
GPIO_24	PIN_AA8	GPIO	[24]	3.3-V	LVTTL	
GPIO_26	PIN_AA7	GPIO	[26]	3.3-V	LVTTL	
GPIO_28	PIN_AA6	GPIO	[28]	3.3-V	LVTTL	
GPIO_30	PIN_AA5	GPIO	[30]	3.3-V	LVTTL	
GPIO_32	PIN_AB3	GPIO	[32]	3.3-V	LVTTL	
GPIO_34	PIN_AB2	GPIO	[34]	3.3-V	LVTTL	SPLITTER_TX_LINE

GPIO 2:	EVENS ROW 2					
GPIO_0	PIN_V10	GPIO	[0]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[6]
GPIO_2	PIN_V9	GPIO	[2]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[5]

GPIO_4	PIN_V8	GPIO	[4]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[4]
GPIO_6	PIN_V7	GPIO	[6]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[3]
GPIO_8	PIN_W6	GPIO	[8]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[2]
GPIO_10	PIN_W5	GPIO	[10]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[1]
GPIO_12	PIN_AA14	GPIO	[12]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[0]
GPIO_14	PIN_W12	GPIO	[14]	3.3-V	LVTTL	PIANO_LEDS
GPIO_16	PIN_AB12	GPIO	[16]	3.3-V	LVTTL	SPLITTER_LINESENSOR
GPIO_18	PIN_AB11	GPIO	[18]	3.3-V	LVTTL	
GPIO_20	PIN_AB10	GPIO	[20]	3.3-V	LVTTL	
GPIO_22	PIN_AA9	GPIO	[22]	3.3-V	LVTTL	
GPIO_24	PIN_AA8	GPIO	[24]	3.3-V	LVTTL	
GPIO_26	PIN_AA7	GPIO	[26]	3.3-V	LVTTL	
GPIO_28	PIN_AA6	GPIO	[28]	3.3-V	LVTTL	
GPIO_30	PIN_AA5	GPIO	[30]	3.3-V	LVTTL	
GPIO_32	PIN_AB3	GPIO	[32]	3.3-V	LVTTL	
GPIO_34	PIN_AB2	GPIO	[34]	3.3-V	LVTTL	SPLITTER_TX_LINE

GPIO 2:	EVENS ROW 2						
GPIO_0	PIN_V10	GPIO	[0]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[6]	
GPIO_2	PIN_V9	GPIO	[2]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[5]	
GPIO_4	PIN_V8	GPIO	[4]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[4]	
GPIO_6	PIN_V7	GPIO	[6]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[3]	
GPIO_8	PIN_W6	GPIO	[8]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[2]	
GPIO_10	PIN_W5	GPIO	[10]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[1]	
GPIO_12	PIN_AA14	GPIO	[12]	3.3-V	LVTTL	PIANO_LIMIT_SWITCHES[0]	
GPIO_14	PIN_W12	GPIO	[14]	3.3-V	LVTTL	PIANO_LEDS	
GPIO_16	PIN_AB12	GPIO	[16]	3.3-V	LVTTL	SPLITTER_LINESENSOR	
GPIO_18	PIN_AB11	GPIO	[18]	3.3-V	LVTTL		
GPIO_20	PIN_AB10	GPIO	[20]	3.3-V	LVTTL		
GPIO_22	PIN_AA9	GPIO	[22]	3.3-V	LVTTL		
GPIO_24	PIN_AA8	GPIO	[24]	3.3-V	LVTTL		
GPIO_26	PIN_AA7	GPIO	[26]	3.3-V	LVTTL		
GPIO_28	PIN_AA6	GPIO	[28]	3.3-V	LVTTL		
GPIO_30	PIN_AA5	GPIO	[30]	3.3-V	LVTTL		
GPIO_32	PIN_AB3	GPIO	[32]	3.3-V	LVTTL		
GPIO_34	PIN_AB2	GPIO	[34]	3.3-V	LVTTL	SPLITTER_TX_LINE	

The following LEDs were used for debugging purposes when interfacing with the Arduino:

FPGA LEDS		DEBUGGING				
LEDR0	PIN_A8	LED	[0]	3.3-V	LVTTL	RX_LED[0]
LEDR1	PIN_A9	LED	[1]	3.3-V	LVTTL	RX_LED[1]
LEDR2	PIN_A10	LED	[2]	3.3-V	LVTTL	RX_LED[2]

LEDR3	PIN_B10	LED	[3]	3.3-V	LVTTL	RX_LED[3]
LEDR4	PIN_D13	LED	[4]	3.3-V	LVTTL	RX_LED[4]
LEDR5	PIN_C13	LED	[5]	3.3-V	LVTTL	RX_LED[5]
LEDR6	PIN_E14	LED	[6]	3.3-V	LVTTL	RX_LED[6]
LEDR7	PIN_D14	LED	[7]	3.3-V	LVTTL	RX_LED[7]
LEDR8	PIN_A11	LED	[8]	3.3-V	LVTTL	
LEDR9	PIN_B11	LED	[9]	3.3-V	LVTTL	

10.2. Integration Challenges

10.2.1. Connecting Ball Splitter with the other sections

An integration issue encountered while assembling the machine was that while the splitter had clearance for most sections of the machine, there were some unexpected interferences caused by some sections of the splitter. The first was the 180° turn for the piano slide section, while this turn was in the design for some time, the position of the turn was changed close to the end of the project, as such, the turn was now mounted at a 45° angle and the previous allowance for space between it and the ladder rungs was no longer sufficient. This caused the splitter to not sit correctly in the mounting brackets resulting in the splitter not being angled correctly and the ball not rolling correctly. This was rectified by filing away sections of the turn until the section no longer clashed with the ladder rungs. Following this change there was another collision where the vertical height of the servo mounted below the splitter was not accounted for in the clearance between the splitter and the top of the mounting plate of the piano slide. In this case the splitter could not be modified to resolve this issue so the top portion of the piano slide mounting plate was removed to accommodate the additional height of the servo.

10.2.2. Ball Splitter Integrity

Another integration issue was the integrity of the marble splitter. While it was assumed that the superglue combined with the key system for connecting pieces would be sufficient to keep the pieces together, it was found that while moving the pieces in and out of the box as others were changing and testing integration, the marble splitter connection for one of the joints broke. The weak point was determined to be the amount of surface area available to glue and the resulting weakness to vertical torque. In order to fix this issue, clear plastic pieces were super glued to the bottom of the splitter to strengthen the existing joints. This proved to be very effective, and the splitter was far more durable for the integration into the machine.

10.2.3. Ball Splitter Connection to Ferris Wheel

A significant integration challenge for the Ferris Wheel Segment was ensuring the successful transport of the ball from the Ball Splitter Segment into the bucket of the Ferris Wheel. This challenge encompassed several factors, including the ball's speed as it travels into the bucket, the placement of the bucket on the large gear and the stability of the bucket itself.

To address these challenges, precise measurements were taken for the ball splitter entrance to the Ferris wheel segment, to ensure the drop-off point was directly aligned with the large gear and bucket. The speed of the ball posed potential issues relating to the force exerted on the bucket causing it to tip upon ball entry. To mitigate these issues an additional weight was added to the back of the large gear using a wood block and a wooden support was installed on the back of the bucket.

Figure 78 Sanding Down Ball Splitter Drop Off to Ferris Wheel

Measurements were also taken at the ball splitter drop-off point to ensure the alignment of the large gear was accurate. The placement of the bucket eventuated to not be a challenge as the rotation of the large gear allowed the bucket to be initially positioned in various locations, ensuring proper alignment.

10.2.4. Ferris Wheel Ball Catcher

Another challenge faced by the Ferris Wheel segment resulted from the ball's speed as it fell from the bucket incurring an unpredictable falling path and he need to create a ball return pathway that did not interfere with other team members sections. Various ideas were

suggested to mitigate these issues such as allowing the ball to fall into the Planetary Revolution section and creating an individual ball return. However, both these solutions have their own drawbacks.

The drawback of the planetary revolution ball idea wasthat it was unpredictable due to the constant motion of the segment. The drawbacks of the individual ball return were that it consumed excessive space and interfered with other team members sections.

The final solution was to design a ball return that collected the ball from both the pinball ladder and the Ferris wheel through a single pathway and directed the ball to the base of the screw lift. This solution was the neatest, simplest and most visually appealing while not impeding on other sections.

10.2.5. Ferris Wheel FSR Breakage

A major challenge that occurred, was when attempting to super glue the FSR to the base of the Ball Splitter channel it broke. This challenge meant that a quick solution needed to occur, as there were no spare FSR's available. The solution to this challenge was to use a spare analog line sensor in place of the FSR. By mounting the new sensor to the outer wall of the Ball Splitter channel, the ball was able to be accurately passed into the bucket.

10.2.6. Ball Splitter connection to Planetary Revolution

Integrating a integrating a ball connection between the ball splutter at the top of the box and the planetary revolution mechanism at the bottom presents a unique engineering challenge. The goal is to establish a seamless connection that allows for smooth movement and transfer of energy between these two components, while also ensuring structural integrity and durability. Because this occurs at the centre of the box there is not access to any supports so the solution must be lightweight and be able to attach to the ball splitter.

To address this challenge an innovative solution has been created. Using oversized clear pipe cut into small sections and inserted into a cylindrical mesh I was able to create a flexible and lightweight tunnel. By cutting the pipe into sections the bending radius of the tunnel increases allowing it to bend and adapt to fit the ball splitter and planetary revolution mechanism.

10.2.7. Planetary revolution cable routing.

Managing cables effectively in the context of the planetary revolution mechanism, situated centrally within the box, is crucial for ensuring neat organisation to help for troubleshooting. One challenge arose with the connection cable between the stepper motor and the driver,

which was initially fixed at 12cm. This posed an issue as all cables needed approximately 30cm of length for proper routing to the back of the box.

To address this challenge and maintain a tidy cable layout a strategic solution was implemented. Initially, the cable from the stepper motor to the driver was routed through the base of the box, allowing for more flexibility and freedom in cable management. From there, the cable was neatly guided to the driver unit situated under the box. Next the signal cables and power cables were carefully routed from the driver, which was now positioned conveniently under the box, to the breadboard located at the back of the box. This method minimised clutter and increased the overall organisation of the system.

10.2.8. Connectivity of Ball Splitter to Piano Slide

Similar to other segments, the connection point from the ball splitter into the entrance point of the piano slide was a challenge that was incurred due to initial design oversights. When components were fitted into the Crazy Machine Box, the Ball Spitter segment was not at an optimal elevation and obstructed the Piano Slide Stand. To overcome this challenge, a corner of the stand was cut off and sanded down to ensure the Ball Splitter could sit comfortably over the top of the Piano Slide stand.

10.2.9. LED wiring Pinball Ladder

As mentioned previously the LED strip wiring have nodes with small distances therefor when soldering the pieces together it was quite difficult as not have any of the nodes touching. Which would cause problems such as shorts. As mentioned previously the 4 Ladder steps were used instead 5. As a result, one of the LED rungs needed to be removed. Which required moving the soldered piece and connecting re-connecting the wires.

10.2.10. Connectivity of the Ball Splitter to Pinball Ladder

Due to discrepancies from the modules mounting the Pinball Ladder section was moved slightly to the right from the initial design, approximately 4 cm. As a result, there was a mismatch between the connection of the Ball Splitter to the Pinball Ladder. As a result, a small additional section was needed to be added to all the ball to move smoothly from between the sections.

11. References

[1] “Micro N20 Geared Motor 150:1 50-200RPM,” Altronics. https://www.altronics.com.au/p/j0054-micro-m20-geared-motor-150:1-50-200rpm/ (accessed May 24, 2024).

[2] “Printables,” www.printables.com. https://www.printables.com/en/model/213682-marble-track-motorized-lift-2021-version-v4-stand- (accessed May 24, 2024).

[3] “Printables,” www.printables.com. https://www.printables.com/en/model/213682-marble-track-motorized-lift-2021-version-v4-stand- (accessed May 24, 2024).

[4] “Solenoid - 5V (Small) - ROB-11015 - SparkFun Electronics,” www.sparkfun.com. https://www.sparkfun.com/products/11015

[5] https://www.youtube.com/watch?v=V0UeqR4-NRE&ab_channel=GEEK