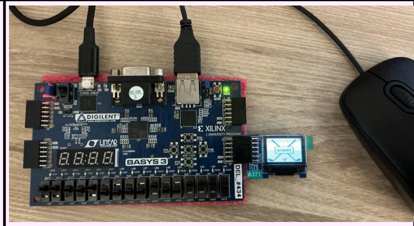
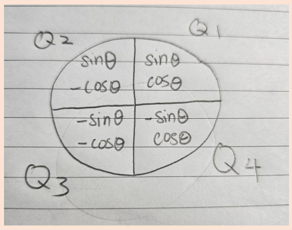
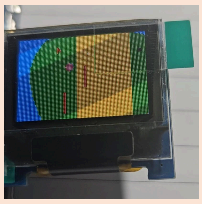
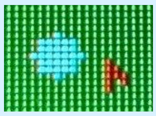
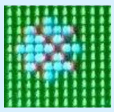
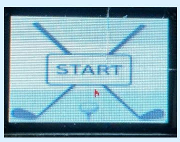
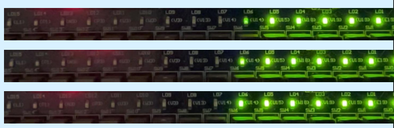
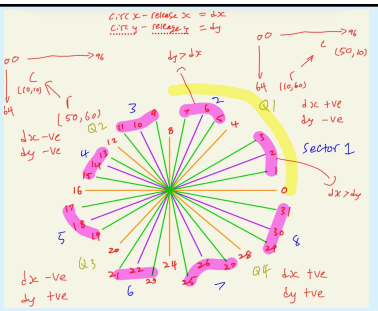
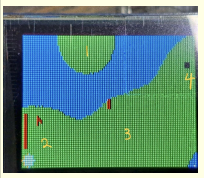
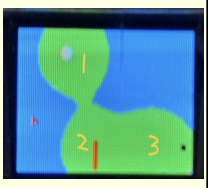
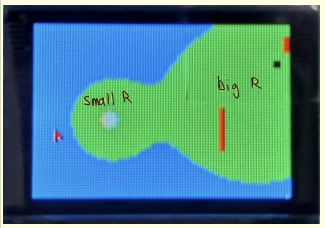
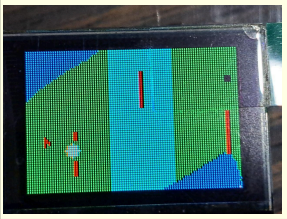
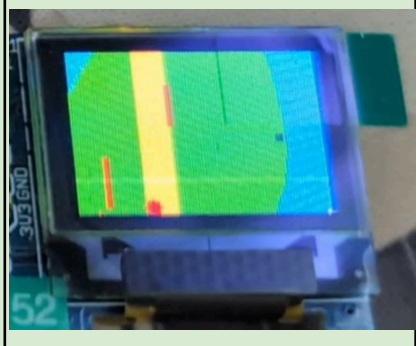
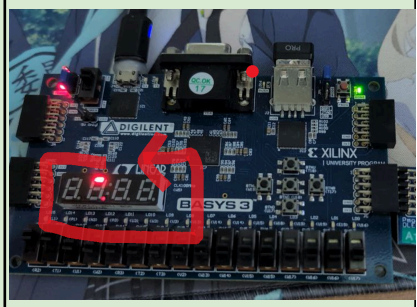
Group ID: S3_07

Member A: Wen Jun Yu (A0309098U)
Member B: Yeoh Soo Leong (A0243539Y)
Member C: Wesley Low (A0307650J)
Member D: Dylan Lim (A0307381H)

Student and Component Name	Component Description	Images / Photos
Team “Micro Golf”	Welcome to Micro Golf, an interactive golf game implemented on the FPGA. This is a real-time system which mimics real world physics of a ball. The user can flip the **switches** to set the **colour** of the ball. The game is played using a **mouse** which interacts with an OLED screen. The **direction & power** (reflected on the LEDs) of the ball is determined using the mouse release point and duration the mouse button is held. A **random map** with obstacles will be generated as soon as the game starts. During motion, the ball experiences **friction** and **bounces.** The **score** of the user is tracked on the seven segment display. Once the user manages to score, they can replay on a new map or go back to the starting page. The user can enter the **cheat code** (LRLRLLRR), bringing them straight to the end screen and light chase around the perimeter of the seven segments. Button btnC: Reset to start position Button btnU: Generates a new map Button btnD: Reset to start menu Button btnL/btnR: Cheatcode	
Student A: Wen Jun Yu Directional Movement of ball Tracking of ball location Scoring Mechanism	**<u>Direction</u>** **General**: The movement of the ball is determined by its **centre** coordinates. After each update cycle, the coordinates will be updated with new coordinates which are dependent on the angle the ball is travelling in. **Angles**: The angles are based on **8 sample angles** in the first quadrant which are then used to calculate the other 3 quadrants, where each angle has its respective sine and cosine values, representing the **vertical** and **horizontal** movement. To determine the sine and cosine values for the remaining angles, the values of sine and cosine are swapped in the 2nd and 4th quadrants using **complementary angles** and sign flipping is used in all 3 quadrants where appropriate. **Bounce**: Upon collision with a wall, the ball will move based on the new **calculated angle** depending on the type of collision. For a **vertical** wall collision, the velocity in the **x-direction flips**. Similarly, during a **horizontal** wall collision, the velocity in the **y-direction flips**. **Stability**: The friction module used to determine the new velocity of the ball at various times uses the ball’s previous cycle velocity to calculate. This affects the bounce direction if the calculated velocity is passed straight to the centre coordinates of the ball. To ensure that the ball moves in the **correct** direction, the output velocities are converted to speed (**magnitude**) before applying to the current velocity of the ball. **<u>Tracking/Movement</u>** Ball **position** is constantly updated using the coordinates of the **centre of the ball**. The **last** known **stationary** position of the ball is also tracked in the event the ball goes out of bounds (enters the blue areas). **<u>Score</u>** The number of shots taken (mouse clicks during game) will be **updated** in real-time using the seven segment display. The score will be live as soon as the map is loaded up. The score will **reset** whenever map switches (btnU), the reset button (btnC) is pressed or the game is replayed(replay button).	    
Student B: Yeoh Soo Leong Mouse integration (click) Cursor & hover effects Click and hold Charging mechanism with LED PWM	**<u>Mouse Integration</u>** Integrated the use of mouse in the gameplay. Supports **clicking** on screen buttons to go through the various menus (start,colour,game,end). Ensures cursor stays within bounds of the OLED display. **<u>Cursor</u>** Included a 5x5 **cursor that changes** when **hovering** over the ball in game. The cursor point changes depending on the cursor (ie tip of the inverted v cursor, and center of X cursor) **<u>Hover</u>** Added **highlighting** effect for all menu buttons when cursor hovers over. **<u>Click and drag + LED PWM</u>** Added feature to **click and drag** to ‘charge up’ the ball to determine direction and power to launch the ball. Includes debouncing of the mouse button such that it does not release prematurely. The duration the mouse button is held for determines the power to launch the ball, and is reflected on the **LEDs** using **PWM** was used to create a **smooth ‘charging up’ effect.**	    

Translating angle and power to ball movement Ball colour Game menu FSM	**Translating direction and power to ball movement module** When the user clicks, drags and releases the ball: The coordinates of the center of the ball and release point of the mouse are used to **determine the direction** of the ball across 32 discrete angles, by normalising to the first quadrant, then calculating the ratio of dy to dx. The duration that the ball is held **determines the initial velocity** the ball should be launched at, and is quantised into 16 discrete levels. **Ball Colour** Allow users to change the colour of the ball using the **switches.** **Game Menu** Implemented FSM to change between different game menus.	
Student C: Wesley Low Map Outline Generation Map Feature Generation Map Checking	**Map Generation** Whenever the “Game Mode” state is entered or when **BtnU** is pressed, a new map would be generated. The shape of the map is generated using the **metaball** equation shown on the right. If F exceeds a threshold then that pixel is considered as part of the map play area. The x,y coordinates as well as the R values are randomly generated using a **Linear Feedback Shift Register (LFSR)** for 4 metaballs. The number of visible metaballs present in the screen will also be randomised using this LFSR value. **Map Features** After the area is generated, the **start point** is determined by the center of the left most metaball within the screen. A search algorithm places the **end goal** by searching from the right until it finds a point within the map play area. The **obstacles** are also sampled from the LFSR with a maximum of 4 obstacles that cannot be generated near the start or end goal (bottom picture). There’s a 50% chance that either **ice** or **sand** will be generated which is relative to the start position so that it is guaranteed to be between the start position and end goal, position and size is from the LFSR. **Map Checking** After the map and features are generated. A modified **Breadth-first search** is implemented to check if the start point is connected to the end goal. Another check is done to see if the start point is far enough from the end goal and if either of these conditions fail, a signal will be sent to generate a new map with a different map seed.	$$F(x,y) = \sum_{i=1}^{N} \frac{R_i^2}{(x-x_i)^2 + (y-y_i)^2}$$    
Student D: Dylan Lim Friction Model Friction Implementation for Different Terrain Cheat Code Horizontal/Vertical Collision Detection End Point (Hole) Detection	**Friction Model** The main formula that the physics model used was the **coulomb friction law**, to **determine the current speed** that the ball should be moving at, snapping to **zero within a dead-zone**. The module will process the formula of **vnext=vcurrent−FA ·sgn(vcurrent)**; if vnext<Dead-zone threshold, the ball will stop. **Friction Implementation for Different Terrain** Taking in the different coordinates for the different terrains (demarked by the different colors in the map), the module will **slow down the ball proportional to the type of terrain detected** (Sand, higher friction (yellow), Grass, normal friction (green) or Ice, no friction (blue)) **Cheat Code** Implemented a **cheat code module**, inspired from the Konami Code. If the user inputs the button sequence of btn(LRLRLLRR), it will **immediately bring to the win screen**, with the **7 segment displays** displaying a chase animation that will **move around the perimeter**. Once the correct sequence is inputted, it will produce a signal signalling that the cheat code has been activated. **Each button press is its own state**, which will bring it to the last state where the **output signal for unlocked will be outputted**. **Horizontal/Vertical Collision** determines if the collision is **horizontal, vertical, or a corner hit** (both). The module will **indicate the type of contact with the respective walls**. When a collision occurs, these signals are updated and sampled on a single-cycle valid strobe to prevent multiple detections within one frame. For **Corner handling** (both horizontal and vertical at the same time), the module has **a flag** that can be changed, whether for it to be **detected as a vertical or horizontal hit.** The module **uses States** to determine whether it is a horizontal, vertical or no hit. **End Point (Hole) Detection** The module will **output a signal** once the **position of the ball** is detected to be **at the coordinates where the end hole** is.	  

References and Feedback:

Member A: Wen Jun Yu (A0309098U)
Member B: Yeoh Soo Leong (A0243539Y)
Member C: Wesley Low (A0307650J)
Member D: Dylan Lim (A0307381H)

References:

We used Claude.ai to clarify on how to use decimal numbers during calculations.
Prompt:
1. How to create cosine and sine values in the first quadrant
2. How do I get the magnitude of two velocities
How the outputs are used: (fixed point Q8.8/Q8.16, MSB 8 bits integers and LSB 8/16 bits fractional)

```
case(angle_index)
    3'd0: begin  // 0°
        sin_val = 16'sd0;      cos_val = 16'sd256;
    end
    3'd1: begin  // 11.25°
        sin_val = 16'sd50;     cos_val = 16'sd251;
    end
    3'd2: begin  // 22.5°
        sin_val = 16'sd98;     cos_val = 16'sd236;
    end
    3'd3: begin  // 33.75°
        sin_val = 16'sd142;    cos_val = 16'sd213;
    end
    3'd4: begin  // 45°
        sin_val = 16'sd181;    cos_val = 16'sd181;
    end
    3'd5: begin  // 56.25°
        sin_val = 16'sd213;    cos_val = 16'sd142;
    end
    3'd6: begin  // 67.5°
        sin_val = 16'sd236;    cos_val = 16'sd98;
    end
    3'd7: begin  // 78.75°
        sin_val = 16'sd251;    cos_val = 16'sd50;
    end
endcase
```

```
function automatic [15:0] velocity_magnitude;
    input signed [15:0] vx;
    input signed [15:0] vy;
    reg [15:0] abs_vx, abs_vy;
    reg [15:0] max_val, min_val;
    begin
        abs_vx = vx[15] ? (~vx + 16'd1) : vx;
        abs_vy = vy[15] ? (~vy + 16'd1) : vy;
        if (abs_vx > abs_vy) begin
            max_val = abs_vx;
            min_val = abs_vy;
        end else begin
            max_val = abs_vy;
            min_val = abs_vx;
        end
        velocity_magnitude = max_val + (min_val >> 1);
    end
endfunction
```

For generation of pictures, I used the picture2pixel python library
URL:https://www.comp.nus.edu.sg/~guoyi/project/picture2pixel/

To ensure readability of the code and easier implementation of modules by my group mates, I used ChatGPT with the following general prompts:
"<Insert module .v file here> With reference to the file attached, I want you to generate comments and labelling to allow for readability. At the top of the code where possible, comment the Input, Outputs and what the module does in general. Make sure that each chunk of code is labelled and categorised well. DO NOT change any of the original code.

Feedback

Name	Feedback
Student A: Wen Jun Yu	I think this module has been a fun and enriching experience, with exposure to new languages (verilog). The assignments were hard but fulfilling when accomplished. However, I think there can be consultation sessions closer to the submission deadline to clarify our problems and doubts with the instructors.
Student B: Yeoh Soo Leong	The mod was enjoyable. I enjoyed the project and appreciate the extension of the project deadline. The lectures are fun and engaging, and the prof and tutors are passionate about the subject.
Student C: Wesley Low	This module was very interesting and enjoyable. I enjoyed learning a different method of approaching problems. Thinking about code in both a hardware and software context adds layers to issues in a way that I have never considered before.
Student D: Dylan Lim	This module is the most fun module I am taking. Having a keen interest in computers, this module really allowed me to appreciate what exactly happens in a computer. The profs and TAs are all super passionate and willing to help where possible.