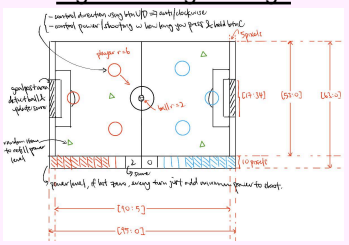
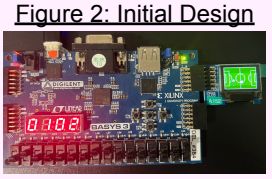
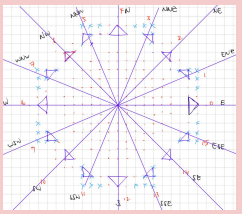
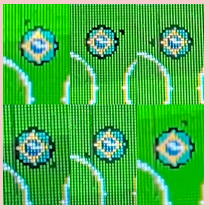
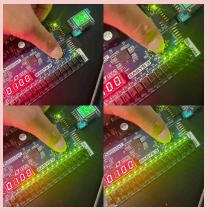
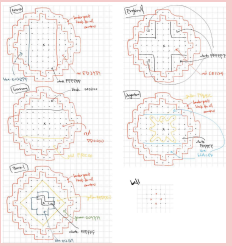
Personal and Team Improvements		
Student and Improvement Name	Improvement Description	Images / Photos
Team Name: Supa Strikerz	Welcome to Supa Strikerz, a 2 Player soccer game experience where **each player competes to score goals and claim victory.** Navigate through our menu page with **btnL, btnR and btnC**. When 2P is selected, this leads to the character selection screen. Choose your country flags, controlled by **sw[15:11] for Player 1 and sw[4:0] for Player 2**, then kick off to begin the match! The game follows a turn based system, allowing each player to take turns **aiming with btnU for anticlockwise rotation, and btnD for clockwise rotation**. **Press and hold btnC to shoot!** The first player to score 3 goals wins, with the **score displayed on the 7 seg display**. Each goal is celebrated with a “GOALLL SIUUU” on the 7 seg, and if the score reaches 2-2, “Golden Goal” will be displayed on the OLED and the next goal wins! Collision detection is one of the key parts of our game, where simple **elastic collision physics were implemented between players, ball and borders**. Resultant direction and resultant power is calculated based on the Point of Contact determined. To manage gameplay, **sw[8:7] is used to adjust the scores** of the players. **sw[9] resets the position of players and the ball**, which are used in scenarios where a goal is scored or if the ball is stuck in the corner. The respective winning screen will be displayed once 3 goals has been scored. Players can **press btnC to start a new game**. At any point in time, **pressing btnL, btnR, btnU, and btnD simultaneously will reset the game completely**.	 Figure 1: Logo Design  Figure 2: Initial Design  Figure 3: Overview of Game
Student A: Kin Player module	This main player module handles all aspects of a player’s actions, including **aiming, shooting, power, direction, and movement**. A turn selector manages when each player can aim and shoot. During a player’s turn, **btnU and btnD adjust the aiming arrow** across 16 directions, while **btnC is held to charge power**, capped at 15. LEDs display the current power level. When **btnC is released, the player shoots** in the chosen direction. **Friction simulated by power <= power * friction / 10**, creating a realistic deceleration. The module **dynamically applies either initial or resultant values** (from collision outcomes) based on factors like movement state, collision status, kickoff, turn, and borders. After extensive trial and refinement with Keane’s collision module, our logic accurately handles these scenarios. There are 3 main movement scenarios: **normal movement, collision with others, and collision with borders**. Normal movement is straightforward, and when a player's collision-detection flag is set, the **current direction and power are overridden by the resultant values** from Keane’s collision module. For border collisions, **elastic physics mirror the player’s direction**, creating a realistic bounce effect. In non-turn collisions, resultant values are also applied. To support real-time collision response, this module continuously outputs current x and y centers, direction, and power while taking in resultant values from Keane’s collision detection module. For visual feedback, the 16 OLED arrow positions, coded manually pixel by pixel, each correspond to a unique aiming direction. Each player’s OLED display also shows their selected country flag from the character selection module. The module is instantiated 3 times in the main module - for P1, P2, and the ball, with initial position and radius values (6 for players, 2 for the ball). **Inputs like sw, menu_count, and scored trigger a reset flag** when set to 1, resetting P1, P2, and the ball to their original positions.	 Figure 4: Final Design of 16 arrows direction  Figure 5: Some of the 16 directions  Figure 6: Charging power for shooting
Student A: Kin Country Flag Designs and Character Selection	In the character selection module, each player selects one of five country flags - France, Brazil, Germany, England, or Argentina, using designated **switches (sw[15:11] for Player 1 and sw[4:0] for Player 2)**. A white ring visually highlights the selected flag. The flags were designed pixel by pixel, selected for their distinct designs and colors, adding visual appeal and differentiation. Additionally, there’s a custom designed soccer ball graphic, as the ball itself is also instantiated from the main player module. “P1”/“P2” labels help guide player selection. **If and only if the two players have chosen different countries, pressing btnL and btnR together initiates the kickoff**, launching the game. Once kicked off, P1_selected and P2_selected will then be outputted to be used in the player module. Sean assisted with the initial layout of flag positions and selection rings. Throughout the game, a **layering trick is used where all sprite backgrounds are set to 16'h0001**. In the main module, only OLED data not equal to 16'h0001 is displayed, creating a “transparent” layer effect. This ensures clear, prioritized layering without color merging, keeping visuals organized and seamless.	 Figure 7: Character Selection  Figure 8: Country and Ball Designs

Student B: Keane

Elastic Collisions between players and ball

This module **continuously tracks the positions of the 3 objects on the field**, calculating proximity between them (e.g., proximity_p1p2). If any two objects are **detected to be in close proximity, the appropriate collision flag will be raised** (collision_detected_p1p2 etc). Multiple signed wires (e.g., dx, dy, d1x, d1y etc) are assigned the values of every single combination of differences between the 3 objects. Using some logic, they are then used to **determine the relative positions of the objects** (eg, if dx == 0 && if dy > 0 then player 2 must be at player 1's north etc). This method helps to **determine the estimated point of contact between the 2 objects colliding, enabling calculation of resultant directions.**

The module then **calculates the resultant power and direction of the 2 objects involved in the collision** and assigns the value to their respective OP and RD. After extensive research, we simplified it into 2 scenarios. **First being when one object is stationary and the other is moving,** the moving object's power transfers to the stationary one, and the resultant direction of the stationary object is determined by the Point of Contact mentioned above. The **2nd scenario being when both objects are moving**, they swap power and direction. The module outputs 3 sets of resultant directions (RD), output powers (OP), and collision flags, which are used in Kin's player module. In the top module, 3 collision_detected wires (collision_detected_p1/2/ball) are assigned conditional OR logic to indicate when each object is involved in a collision.

Initially, we explored a more complex elastic collision model which was extremely tricky without the use of vector calculations. For collision detection, we used individual flag points around each object's circumference, but this required excessive memory and slowed down performance as it uses pixel memory that constantly outputs the status of the entire board. To improve efficiency, we implemented larger hitboxes, reducing memory usage and calculation time. For resultant direction calculation, we considered scenarios from a different perspective from the ones mentioned above - whether the object is initiating or receiving collisions. The resultant direction of the object being hit would be 180° from the point of contact, while the object that initiates the collision would reflect its direction across the point of contact. It was challenging to reliably determine whether an object was initiating or receiving the collision, making the approach overly complex. Thus, we simplified the model to the 2 scenarios mentioned above, which eliminated the need to distinguish between initiating and receiving collisions, providing reliable results with lower complexity.

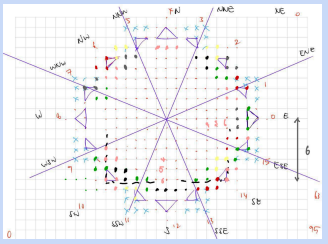


Figure 9: Initial pixel_memory method of detecting collision, the coloured dots around the circle represents the flags designed to detect if collision had occurred

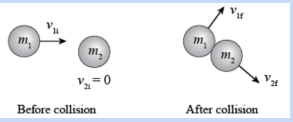


Figure 10: Example of an Elastic Collision (Used to model the collision_detection module)
4

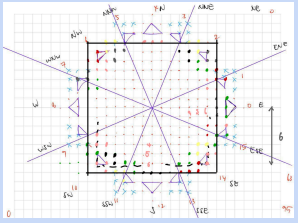


Figure 11: Final method of detecting collision, turned the individual detection flags into a hitbox of sorts

Student C: Sean

Menu Page Design and Flow

Menu Page Flow

The menu page flow is as shown in the flowchart. There are a few sections to our menu page. a) **Main menu**: counts 0 and 1. b) **Control explanation**: counts 2, 11 and 12. c) **Player Mode Selection**: counts 3, 4 and 5. d) **Character Selection**: counts 6 and 7. e) **Game Play**: count 8. f) **Win Screen**: counts 9 and 10. g) **Tie Breaker**: count 13.

Menu Page Controls

1. **btnL and btnR** are used to move **left and right**, toggling between pages.
2. **btnC** is used to **confirm** your choice.
3. At count 6, **pressing both btnL and btnR simultaneously** will kick off.

Game Flow

1. In P2 mode, if P1 wins, P1 win screen will appear. Vice versa for P2.
2. If there is a **tie**, "Golden Goal" will briefly flash and the next goal wins

Reset: Press **btnU, btnD, btnL and btnR at the same time** at any page to reset back to count 0.

Menu Page Design

The "SUPA STRIKERZ" logo will be displayed throughout the choices apart from the control explanations. The **control explanations will show the user which buttons to use to play the game.** To show the user their choice, a white border will be displayed. If that is not the user's choice, it will be a black border. To make the choice even clearer, the **white border will blink at 1hz**.

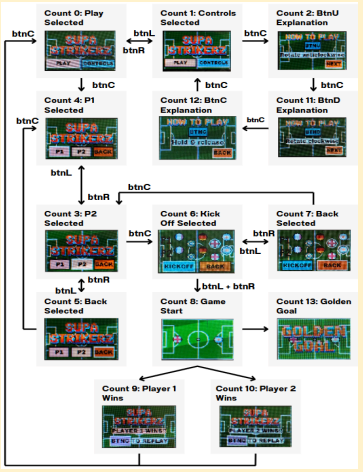


Figure 12: Menu Page Flowchart

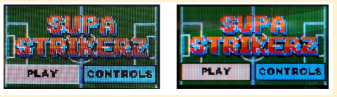


Figure 13: Blinking of Borders

Student D: Darren

Goal detection, Score counter display, and Soccer field background

Goal Detection Mod (Game Logic)

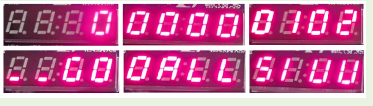
The goal detection module checks if a game is active using menu_count before displaying anything. If a game is ongoing, when it detects goals, it calculates total goals for each player, and sends two output signals to the score display module: a scored signal (1 or 0) and a 4-bit score tally signal, where the 2 MSBs represent player 1's score and the 2 LSBs represent player 2's score (e.g., 4'b0000 for 0-0 and 4'b1111 for the maximum score). This module is crucial for managing gameplay flow and game logic.

Score Display Mod (Game Flow)

At the start of the game, this module receives inputs from the goal detection module and displays an initial score of 0-0. It constantly monitors for goals, displaying "Goallll Siuuu" when a goal is detected, then updates to show the new score tally. This module outputs three signals to the main modules: 7-segment display signals, anode control, and a game-set signal, which help other modules track the game state. It manages score display, win-lose conditions, and game flow, allowing players to see current scores and track goals needed to win.

Soccer field background (Game Flow)

This soccer field would be displayed throughout in the background, as it helps to give the players a seamless and immersive gaming experience. Its aim is to draw the players into the soccer stadium feeling and will engage the players better.



Top Row: Initialization; Game Start; eg. Score 1-2
Bottom Row: Goal Detected "GoalllSiuuu"

Figure 14: Score & Goal Display

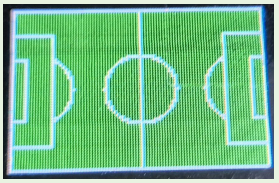


Figure 15: Soccer Field Background

References

Game inspiration: https://platoapp.com/en/games/table-soccer

Our Project's github page: https://github.com/keanneeee/EE2026.git

Inspiration for collision detection and collision handling: https://github.com/bogini/Pong

Basic Physics for elastic Collisions: https://texasgateway.org/resource/83-elastic-and-inelastic-collisions

This video describes code logic for Elastic Collisions: https://www.youtube.com/watch?v=dJNFPv9Mj-Y&t=1065s&ab_channel=TheCodingTrain

The logo font that SUPA STRIKERZ logo is based of: https://www.textstudio.com/logo/831/pixel

7-segment multiple display logic (eg. display "1234"):
https://forum.arduino.cc/t/displaying-multiple-digits-on-7-segment-display-simultaneously/642679/2

7-segment multiple display case logic:
https://adaptivesupport.amd.com/s/question/0D52E00006hpL5gSAE/different-numbers-on-seven-segment-display-basys-3-vivado-vhdl?language=en_US

7-segment multiple display case logic:
https://stackoverflow.com/questions/45064003/displaying-different-numbers-on-2-seven-segment-displays-on-vhdl-spartan-3

Inspiration for the menu page buttons:

AI Declaration

We did not use any code generators for the coding in our project, only ChatGpt to refine our ideas on Elastic Collision between objects.
We also used an image to pixel converter we got from a friend of ours, unfortunately, we only have the file itself but no links for it.

Feedback

Kin: This has genuinely been one of the most enjoyable modules I've taken, largely because we can visually track our progress, something different compared to other computing modules. However, while I understand the constraints on time, I feel that the period allocated for the final project isn't sufficient to achieve the best results. During the last two months of the semester, many assignments and exams from other modules compete for our time, limiting how much we can dedicate to the project. I believe we need to have a balance of everything and in this context, time and project quality. I suggest allowing us to start the project earlier, perhaps before recess week. Introducing the foundational basics earlier would give us more time to build a strong foundation and plan our approach, resulting in better overall outcomes for our final projects.

Keane: This module has been one of the most demanding modules I've taken throughout my time in NUS. However I've learnt so much from this module and despite it being very tiring, I've grown to enjoy the process. The project was actually a fun addition to the module, allowing us to apply the skills we learnt in the labs directly to a project where we are free to use our creativity to design anything we want. This really solidified my learning and increased my interest in FPGAs in general. However the time frame is really too short and throughout the time where we were supposed to be working on our project, some of us were busy with exams and projects from other modules, hence I feel that extending it by a week or two would be a better timeframe for us. Furthermore, a review session or a few consultation sessions with the professor or even some TAs would be nice as most of us are beginners in this language and we are still learning, it would have been great to have had some guidance.

Sean: I feel the project can have a mock trial whereby the TAs or Professors can give feedback before the final run. This is so that we know what we can further improve on before the final run. I also feel that the verilog evaluation can be slightly longer because I feel that the duration given is not enough. Moreover, if it is possible, can the lectures and tutorials start on time and end on time. I appreciate that the lectures give us students time to grab a bite before coming to lectures just in case we have any lessons before, but as someone who has lessons after the lectures, I would appreciate it if the lectures could end at least 15 minutes earlier so that I can catch the bus / walk to my next lesson. Overall the course has been fun and eye-opening, as something like game design is something i enjoy, and programming the different assignments while tedious, it gives me a sense of accomplishment after I completed and can play around with the final result.

Darren: I feel that the overall learning experience was quite rushed during the labs assignments as we are suppose to build the lab 1,2, and 3 in a week, in conjunction with our other modules assignments and i just felt that i could have a bit more time to get used to Vivado, and i think maybe why it was time consuming for me was when I am doing the testings for my code, teach time I generate bitstream it just takes up 50% of my coding process and it was just 2-3 mins of me sitting there looking at my code and reading it to double confirm just for the code to load up and not work as intended and I make a small change in 10 seconds just to repeat the whole waiting process.