







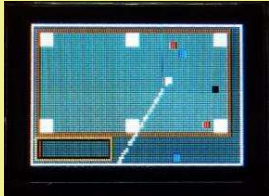


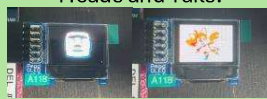
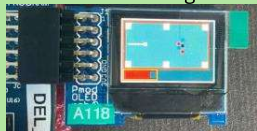


Personal and Team Improvements		
Student/ Improvement	Improvement Description	Images / Photos
Team S1_14 Pool	<p>A scaled-down version of the classic pool game, with 6 balls, including 1 white ball, 1 black ball, 2 stripes (red) balls and 2 solids (blue) balls. The game is played on the OLED module of 1 Basys board, and 2 mouses with 2 Basys boards, with the mouse signals (mouseX and Y coordinates) transmitted via UART. The mouse controlling the gameplay is controlled by sw[5]. Shooting and aiming the white ball is controlled by clicking the left click button of the mouse, and its positioning, respectively. Sw[0] can be turned on to simulate the flipping of a coin. When switching it on, it needs to be on for 5 seconds before the next flip can happen.</p> <p>Our implementation of the game is more simplified. The game logic will automatically determine the play available to the player, for example: If the white ball is potted, the opponent is then allowed to place the white ball anywhere on the table using the four directional buttons and center to confirm, and the opponent gains control. The current allowed move will be abbreviated and displayed on the 7-segment display. For example, if it is blue's turn to shoot the white ball, "bLuE" is displayed on the 7-segment display. If red potted the white ball and blue is allowed to place the white ball, "PLCb" is displayed. When a player wins, the letters of the display slowly cycle through. The game starts by displaying a pool player playing pool and will be unlocked after the 2 players engage in a coin flip to determine who starts the game. They can mutually assign heads and tails and then turn on sw[0] to know which player has won. This process is random. After that, sw[1] is turned on. The display then shows the empty pool table. We can set the balls to one of 4 random starting states by turning on and off sw[15]. A line is constantly drawn from the cursor to the ball to help with aiming. Do note that the ball will always travel in the opposite direction of the line drawn. The closer the cursor is to the ball, the greater the strength of the shot. A strength bar is also added for players to have a visualization of the strength of the shot. For example, if there is a line from the center of the ball to the left of the screen, the ball will travel to the right when mouse left is clicked. The game ends when a player wins and can be reset again by turning sw[15] on and off. The game can also end when the black ball is scored before any other ball, in which case nothing is displayed on the segments, and the game can be reset again by toggling sw[15].</p>	  
Student A: Tan Bingxue Gerard A0252610 U Game Logic, Ball control	<p>Implemented the score detection system, where a 6-bit register is populated from 000000 with 1s as the ball corresponding to the specific bit is identified as "scored" by the game. The ball is considered to be scored when the position of the centre of a ball overlaps with the hitbox of the scoring holes. The function also ensures that potted balls are teleported and displayed under the play area, both as a way to track the current progress of the game, as well as not to interfere with the movement of the other balls that are in play. The 6-bit register is used to determine the flow of the game, such as when the player is allowed to shoot, and which player is the one who is currently shooting, as well as to detect when the white ball has been potted, and the opponent player is allowed to place the white ball at any valid position.</p> <p>Designed the ball placer function, which makes use of the 5 directional buttons on the Basys 3 to intuitively manipulate the ball position and btnC to confirm it when the player is ready to return to the shooting mode. Created the skeleton of the ball shooting function, where the position of the mouse is read in and its relative position to the white ball is used to consider the resulting trajectory and speed of the ball, through calculations devised by Student B. Was also in charge of implementing all displays on the 7-segment with respect to the current game state.</p> <p>Helped Student B with the implementation of collisions, to ensure that as much as possible with the 3 x 3 pixel representation of our balls, by coming up with the 16 possible types of collisions that were possible and implementing them in the game in a way that would accurately reflect a similar collision between spherical balls in a perfectly elastic situation.</p>	<p>7-Segment during red's turn:</p>  <p>7-segment when ball is to be placed:</p> 
Student B: Tan Yi Xin A0252733 H Physics, Math and Logic	<p>Created the architecture for ball physics, which includes collisions between balls, friction, direction and speed. Came up with a way to roughly calculate a ball's speed and direction based on approximated values without using decimals and solely using bit manipulation when given an impulse. Also implemented a way for friction to be random to ensure no two games are the same through a desynced clock.</p> <p>Created a command system for the balls to be able to teleport them to outside the table when potted, change their speed and direction when colliding with walls or other balls, and reset them to their initial starting states when the reset switch is turned on. Of these starting states, they are pseudo randomized until the player turns the reset switch off. Collisions between balls were also handled in such a way that mimicked real life as much as possible, with conservation of energy and momentum being obeyed as much as possible. The physics engine was also implemented in a way to allow the white ball to be shot at almost any angle, as the direction and speed are based on the ratio of X and Y values. However, to optimise against heavy usage of mathematical operations, resultant collision directions were mostly approximated by hand.</p> <p>Helped Student A with the logic of determining the flow of the game, namely identifying who's turn is it (stripes/solids) based on the states of the balls and indicating the control should be changed when the current player misses, or when the game is won through the 7seg display. Helped with the logic of automating the state of placing the white ball when it detects that the white ball has been scored and no balls are moving.</p>	<p>Initial state of the board</p>  <p>Layout after "breaking" the balls and scoring a red ball.</p>  <p>Next shot after scoring the blue ball.</p> 

	Did the mathematics and bit manipulation behind applying an impulse to the white/cue ball when a shot by the player is detected, while also preventing the player from shooting while any ball was moving. This resulted in near perfect angles when shooting.	
Student C: Ian Freda Hariyanto A0258293 U Design and Integration of Line Generation Algorithm, Power Bar, Mouse Integration.	<p><u>Design of Line Generation Algorithm</u> Implemented a line generation algorithm in Verilog, inspired by Bresenham’s Line Algorithm. The algorithm generates a straight line between any two points on the OLED. This is done using a mathematical decision parameter that is incrementally generated pixel by pixel. This modified algorithm successfully considers all the possible inputs, namely 8 total cases consisting of 4 possible directions, each having a gradient of magnitude <1 or >1.</p> <p><u>Integration of Line Generation Algorithm</u> While the algorithm helps determine which pixels to fill in, four 2D arrays are used to store calculated pixels values to be displayed by OLED. These arrays, in addition to using different states, solve the issue of incremental generation of line, making displaying of a solid line possible. The size of 2D arrays has been optimized to only store and check the necessary pixels to form the line.</p> <p><u>Design and Implementation of Power Bar</u> Created a smoothly animated power bar that displays the strength of player’s shot based on the coordinates of white ball and cursor. Values calculated by Student’s B algorithm are processed through bit manipulation so that minimum and maximum power achievable by players are properly scaled to use the entire length of power bar.</p> <p><u>Mouse Integration</u> Integrated mouse together with the line generation algorithm and the power bar to continuously display a solid line between the cursor and the white ball together with its corresponding power.</p>	<p>Low Power Shot:</p>  <p>High Power Shot:</p> 
Student D: Gandhi Parth Sanjay A0252403 U Parallel Inter-board UART, Randomizer module, Game States, Pixel Data Generation	<p><u>Inter Board UART</u> Implemented parallel Inter-FPGA Universal Asynchronous Receive and Transmit (UART) for X and Y coordinates of mouse to enable 2 players to play the game. Used a transmit module running at a frequency of 1000Hz, which took in data supplied by the mouse module and parsed it into a 16-bit data packet for transmission. The data packet consisted of 1 start bit at logic LOW followed by 12-bit of coordinate data at various logic states. The rest of the bits in the data packet were mainly fillers for the 16-bit data packet and were set to logic HIGH. Used a receive module running at a frequency of 1000Hz, which polled for the starting bit in LOW state from the transmit module. Once the start bit was received, bit concatenation was used to update the received mouse coordinate data. Also implemented a slower 30Hz clock to update the mouse data at an acceptable timeframe to prevent hyper-updating the mouse coordinates. There were 2 UART modules running concurrently for the X-coordinates and y-coordinates respectively to ensure that the data transfer was more efficient. <u>Led 12-15 light up for the receiving board (Master board) and led 13,15 light up for transmitting board (2nd board) when UART is running as intended.</u></p> <p><u>Randomizer Module</u> Coded out a randomizer module which uses an 8-bit counter running at 100Mhz. We then XOR all the individual bit data to generate a random high or low state. At the start of the pool game, the 2 players can choose between heads and tails to determine who will start the pool. The high state corresponds to the person who has chosen heads, and the low state corresponds to the person who has chosen tails.</p> <p><u>Different Game States</u> Coded out the different game states and transfer of mouse control in this game. Used various switches (sw[0] to sw[2]) and added 5-second delays counters in all the states to enable smooth gameplay. Used these delays when switch 0, 1, or 2 are turned on. Integrated code to handle the transfer of the utilizing the mouse coordinate and mouse-click data from master board or the 2nd player board using sw[5].</p> <p><u>OLED Data Generation</u> Used a combination of MATLAB Codes and Python Code to generate bit values for the 6144 pixels on the OLED Display for 4 images. Hard-coded values for the pool table which is displayed while playing the game.</p>	<p>Master Board:</p>  <p>2nd Player’s Board:</p>  <p>Main Starting Screen:</p>  <p>Heads and Tails:</p>  <p>Main Pool Page:</p>  <p>End/Trophy Page:</p> 