DIGITAL DESIGN PROJECT REPORT

Caleb Nelson

December 14, 2021

Walla Walla University
ENGR 433
Dr. Haun

Contents

Abstract 2
Project Description 2
Game Play and Project Design Criteria 2
Block Diagram 4
Main FSM State Diagram 5
Ball Position FSM State Diagram 6
Timing Diagram 7
Architecture Description 7
  Main FSM 8
  Ball Position FSM 8
  LED Matrix Display 9
  Successful Hit Logic and Counter 9
  Game Clock Manager 10
  Score Manager 10
  Score Display Manager 10
Problems Encountered 11
Resource Summary 11
Conclusion 11

Abstract

This project was the final project for ENGR 433 (Digital Design) at Walla Walla University. In summary, the goal of this project was to use a Xilinx FPGA, some input buttons, a 4 digit 7-segment display, and an 8-row by 16-column array of LED's to implement a simplified version of the classic arcade game Pong. In this simplified version of the game, the ball was to move left and right across the display and bounce off player's paddle if they were present or result in a point being scored if no paddle was present. The ball's motion was also to speed up after a number of successful hits. A more detailed project description can be found in the body of this paper. We ran into a few issues during development and had to slightly re-architect some of our state machines and logic. We especially had issues with the ball's speed logic as it needed to be updated both on every successful hit, and upon a new game. After many hours of labor however, we were able to get the entire project working as expected!

Project Description

In this project, we designed a simplified version of Pong, a classic arcade game where a ball is "batted" back and forth between two players. For this project, diagonal ball motion and vertical paddle motion was neglected; the ball simply moved in a straight line laterally. When the ball reached an edge, it would either bounce off a player's paddle if present or continue off the display and a point would be scored. In the latter case, the ball would then reappear on the other side and wait in place until the start button was pressed, at which point it would begin motion again. Additionally, the ball would speed up after an appropriate number of successful hits. The ball was to be displayed on the LED matrix display and the player's scores were to be tracked and shown on the 7-segment display. A player scores points every time their opponent misses the ball. The first player to reach 9 points wins the game, at which point the game can be reset and will initialize a new game. Greater detail regarding the exact game play mechanics, inputs, and outputs is provided in the next section.

Game Play and Project Design Criteria

<u>Buttons/inputs and outputs:</u>

- Switch 10 will be used for the left player's paddle (player1).
- Switch 12 will be used for the right player's paddle (player2).
- Switch 11 will be the start button.
- Switch 9 will be the reset button.

<u>Scoring, general rules, and gameplay:</u>

- The player on the left is considered player 1, the player on the right is player 2.
- When a new game begins, the ball will begin on the right by player 2.

- A player can score anytime their opponent misses a hit (it doesn't matter who's "serve" it was), every time a ball is missed, someone scores.
- Every time a player scores, the ball will begin on the victor's side.
- Scores will go up to 9, first one to 9 wins.
- Ball motion will begin again once the start button is pressed.
- For a hit to be considered successful, the player receiving the ball must have their paddle button active while the ball is in the column adjacent to/in front of their paddle.
- If we didn't want players to be able to always hold their paddle in position (and therefore never miss the ball) we could add an additional condition that the player's paddle must not be pressed when the ball is the third column from the paddle. However, we decided not to implement this as it's useful to be able to just hold the paddle buttons down for testing and verifying speed logic, scoring logic, winning logic, and restarting/resetting, plus it's fun.
- Once a player wins, the reset button will have to be pressed to reset the scores and proceed to the initialization state to prepare for a new game.
- The start button will have to be pressed to start the game from the initialization state.

<u>Display:</u>

- Paddles will occupy the inner 4 dots on the outer most columns.
- Paddles show whenever the respective paddle button is pressed.
- Ball will stay within the inner 14 columns when properly hit.
- Ball will reach the outer columns if not properly hit/returned.
- The ball will be displayed in row 4.
- Player 1 will have their score on the left most digit on the 7-segment display in base 10.
- Player 2 will have their score on the right most digit on the 7-segment display in base 10.

<u>Speed:</u>

- The initial speed of the ball will be 5 dots/second.
- Upon the first successful hit and then after every two successful hits thereafter, the speed will increase by 3 dots/second.
- The maximum speed is 50 dots/second – (We chose to have the speed setting represented by a 4-bit binary number giving 16 total options and 5 + 3*15 = 50).
- In order to increase game difficulty and keep games relatively short, the ball's speed is only reset upon a new game.

Block Diagram

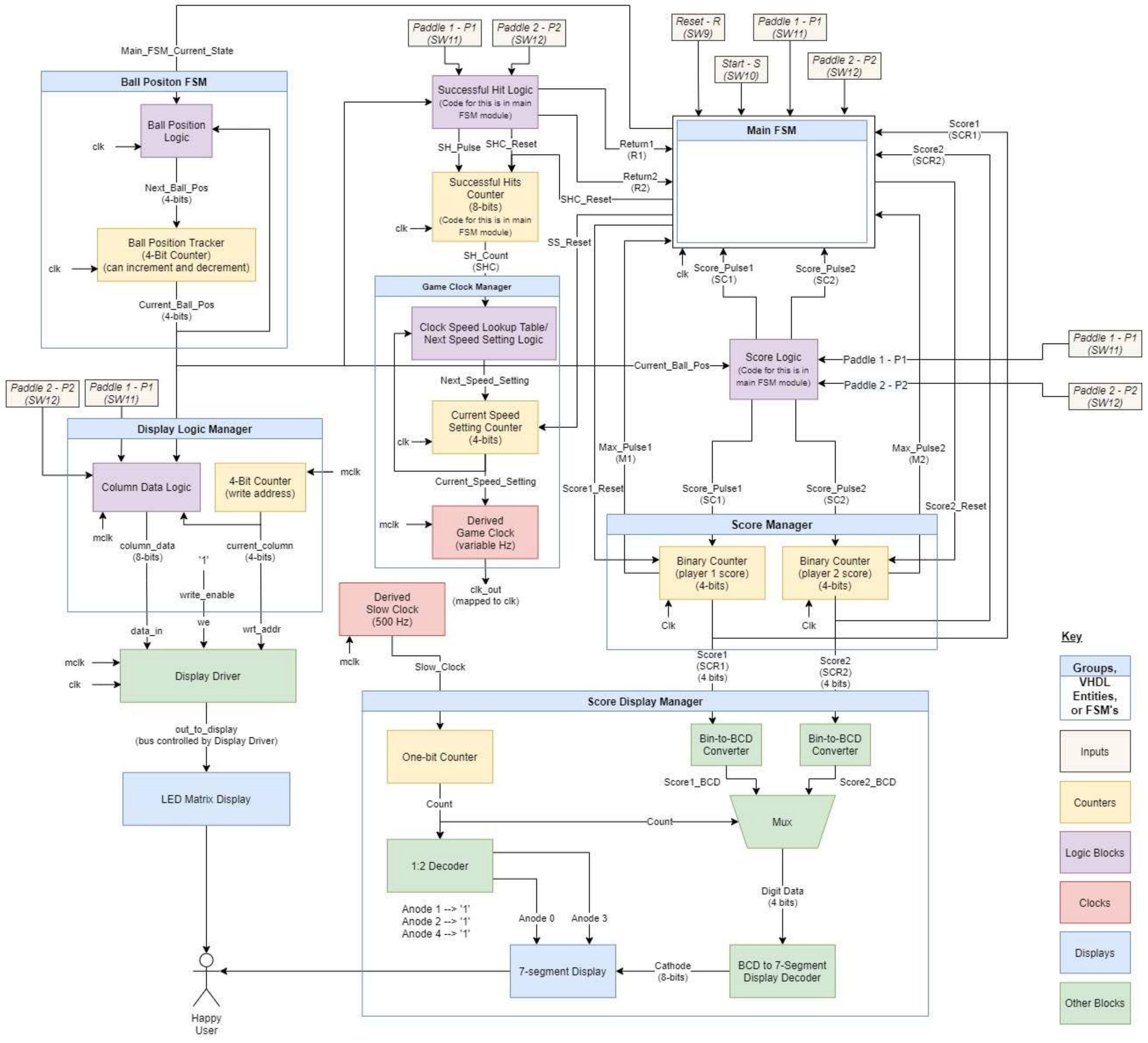
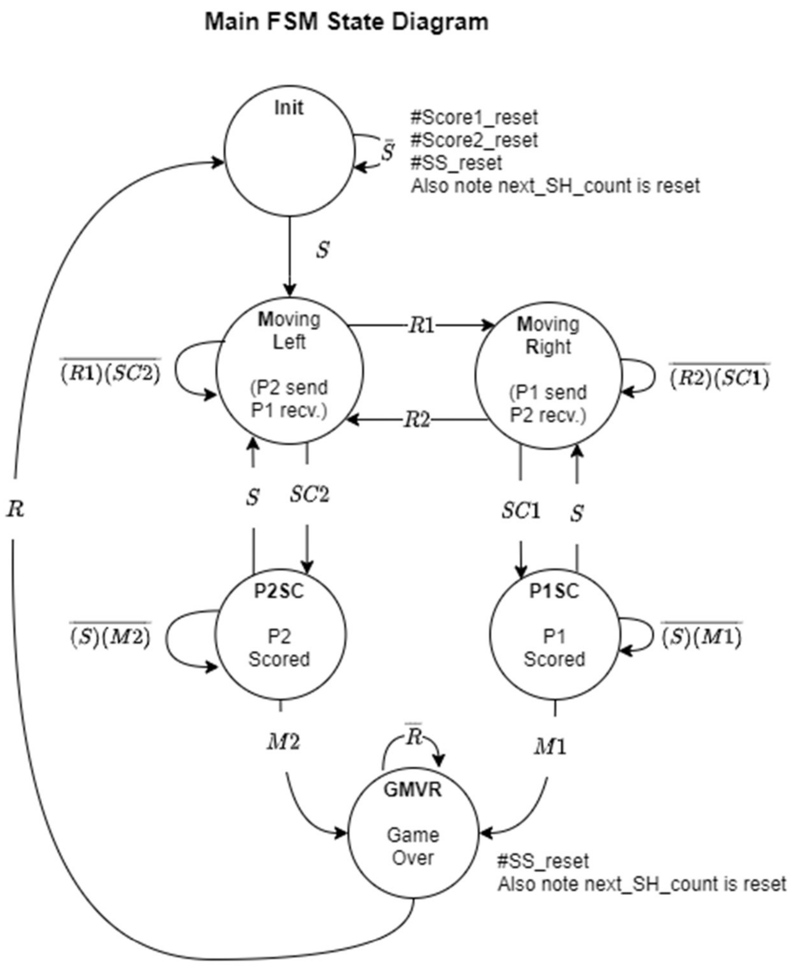


Figure 1 - Block Diagram

Main FSM State Diagram



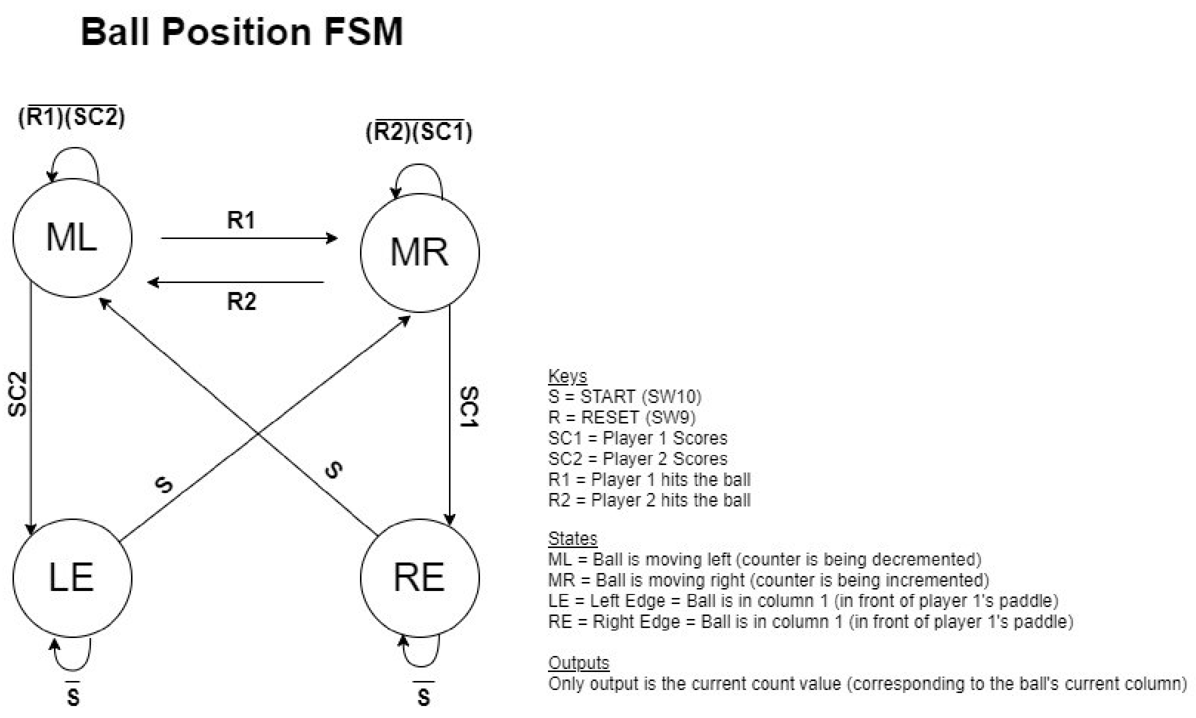
Key
R = reset = sw9
S = start = sw11
SC1 = score pulse 1
SC2 = score pulse 2
R1 = return 1 = played 1 returned the ball
R2 = return 2 = player 2 returned the ball
M1 = max pulse 1 = player 1 has reached max score and won
M2 = max pulse 2 = player 2 has reached max score and won

Outputs
Score1_reset = score 1 reset
Score2_reset = score 2 reset
SS_reset = speed setting reset
next_SH_count = next count for the successful hits counter

States
ML = Ball Moving Left State
MR = Moving Right State
P1SC = P1 Scored State
P2SC = P2 Scored State
INIT = Initialization State
GMVR = Game Over State

Figure 2 - Main FSM State Diagram

Ball Position FSM State Diagram



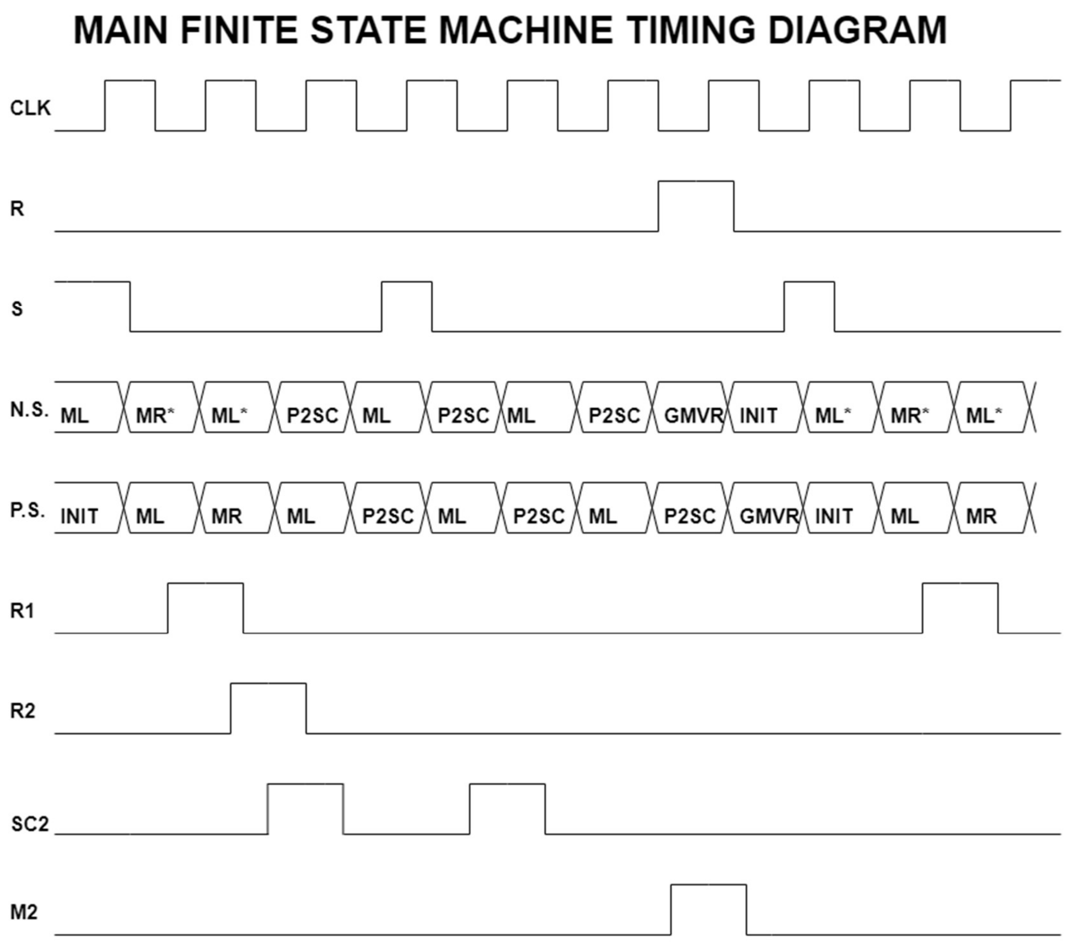
Keys
S = START (SW10)
R = RESET (SW9)
SC1 = Player 1 Scores
SC2 = Player 2 Scores
R1 = Player 1 hits the ball
R2 = Player 2 hits the ball

States
ML = Ball is moving left (counter is being decremented)
MR = Ball is moving right (counter is being incremented)
LE = Left Edge = Ball is in column 1 (in front of player 1's paddle)
RE = Right Edge = Ball is in column 1 (in front of player 1's paddle)

Outputs
Only output is the current count value (corresponding to the ball's current column)

Figure 3 - Ball Position FSM State Diagram

Timing Diagram


MAIN FINITE STATE MACHINE TIMING DIAGRAM
CLK
R
S
N.S.
ML MR* ML* P2SC ML P2SC ML P2SC GMVR INIT ML* MR* ML*
P.S.
INIT ML MR ML P2SC ML P2SC ML P2SC GMVR INIT ML MR
R1
R2
SC2
M2

This game is a best out of two game for limitation of time and document restrictions. The start (*) means that the ball moved from one side to the board to the other and did it in the significant amount of clock cycles. In the diagram, it only uses one cycle due to the limitation of sizing for the diagram . When SC2 is asserted it means that the player one paddle(SW11) is not asserted which shows PSC2 as the projected next state. This can go either way for player one. When the paddle button is pressed, the projected next state will be MR, the state where the ball moves from left to right. There is more to the timing diagram but this is a typical case scenario game.

Figure 4 - Timing Diagram

Architecture Description

Figure 1 outlines all the primary components in our design as well as the signal names used for transferring information between the various components. Each of the main blocks within this diagram are broken down and described in detail in the subsections below. Additionally, state diagrams for our two primary finite state machines are provided in figures 2 and 3. The exact operation and function of these state machines can also be found in the following sections.

Main FSM

The Main Finite State Machine (FSM) is responsible for the overall game state and progression. Figure 2 provides the state diagram for this state machine. This FSM has 6 states including: initialization (INIT), moving left (ML), moving right (MR), player 2 scored (P2SC), player 1 scored (P1SC), and game over (GMVR).

The game will start in the initialization state and remain there until the start button is pressed. It will also return to this state once the reset button is pressed after a game is won. While in this state, the FSM will reset/initialize the players scores, the successful hits counter, and the current speed setting.

During gameplay, the FSM will transition between 4 main states, a state where the ball is moving to the left, a state where the ball is moving to the right, a state for when player 1 scores, and a state for when player 2 scores. When the ball is in motion, it will remain in motion (state ML or MR) until the receiving player returns the ball or misses the ball. If the receiving player misses the ball, the opponent scores a point, and the ball appears on their side and waits to be begin motion again until the start button is pressed. If the receiving player returns the ball, the ball will "bounce off" their paddle and begin traveling in the opposite direction. This gameplay will continue until the max score is reached.

Once one of the players reaches the max score, the FSM will transition to the game over (GMVR) state and wait there until the reset button is pressed. While in this state, the FSM will reset the ball's speed setting and reset the successful hits counter. Once the reset button is pressed, the FSM will progress to the initialization state to prepare for a new game.

Ball Position FSM

The Ball Position Finite State Machine is responsible for the ball's motion. It is basically a glorified 4-bit counter which has 16 states. The state of this state machine, equivalent to its count, is its only output. This output corresponds to the current column the ball is displayed in on the 16-column display. The state diagram for this FSM (Figure 3) contains only 4 states since the primary operation can easily be simplified to 4 states. The ball is either moving right (MR) (counter is being incremented), moving left (ML) (counter is being decremented), or is staying fixed in position on the right-side edge (RE), or the left-side edge (LE). Special wrap around logic for the moving states is unneeded because one of the two conditions for transitioning out of the current moving state will always be met once the ball reaches an edge. Either the ball will bounce off a paddle (return signal) and transition to the state where it travels in the opposite direction, or a point will be scored, and the ball will be fixed in a specified location. Whenever a point is scored, the ball is fixed in position in front of the victor's paddle. It will then transition to the proper moving state once the start button is pressed.

LED Matrix Display

There are two primary components used to drive the LED Matrix Display, the Display Logic Manager, and the Display Driver. The Display Driver module was written by Dr. Aamodt, and we simply made use of it. It took three primary inputs (data_in, we, and wrt_addr) and interfaced between our code and the display itself. We provided the Display Driver with the column we wanted to write to (wrt_addr), the rows to light in that column (data_in), and a write enable signal to indicate whether a write was to take place (we). While the Display Driver does have internal ram and performs its own continual raster scan of the display, we chose to perform a continual raster scan of the display in our code as well. The Display Driver takes two clocks, the master clock (mclk) and a slower glock (the game clock). This is so that the display's LED's can be continually scanned at the master clock rate, but the actual display can be updated at the game clock rate and updates only have to be written to the driver when an LED's value changes. Instead of only writing updates to the Display Driver at the game's clock rate, we decided to provide it the master clock on both clock inputs and continually update it. We choose this for a few reasons. One of the biggest reasons for this is it decouples the display updates from the main FSM state progression and game clock. This allowed the paddles to be updated independently from the ball's motion. For example, with our setup, the paddles can, in theory, appear and disappear multiple times while the ball remains in one position. This is because the ball's position is updated with the game clock, while the display is updated with the master clock. Another reason we choose to do this is because it seemed more logical to us and easier to implement as we didn't have to deal with doing two separate writes every time we wanted to update the ball position (one to clear its old position and one to place it in the new position).

The Display Logic Manager is made up of three parts, a continually incrementing 4-bit counter which is used for the current column to write to, a write enable signal which is always high since we are always writing, and a column data logic block. The column data logic is fairly simple combinational logic which takes the current column, current ball position, and two paddles as inputs. If a paddle is active and the current column is that paddles column, it will light the inner 4 rows in that column. If the current column is the column the ball should be in, it will light the 4th row in that column to represent the ball. If neither of those conditions are met, none of the rows in the current column will be lit.

Successful Hit Logic and Counter

The successful hit logic and counter block is responsible for determining if a player successfully hit and returned the ball, and if so, which player it was. This block will assert signals return 1 (R1) and return 2 (R2) to indicate a successful return of the ball by either player 1 or player 2 respectively. A player successfully returns the ball if their paddle is active when the ball is in the column directly in front of the paddle. This block also generates a successful hit pulse (when either player successfully returns the ball) which increments a counter that tracks the total number of successful hits in the game. This counter can be cleared/reset by the main FSM when the game is reset, or a new game is initialized. The output of this counter goes to the next important entity, the Game Clock Manager.

Game Clock Manager

The Game Clock Manager is responsible for generating a derived clock signal which is used to control the rate of the game and ultimately the speed of the ball's motion. The output of the successful hits counter is used to determine the current speed setting, which determines the speed of the game clock (which in turn controls the speed of the ball's motion). The "Clock speed lookup table/next speed setting logic" block is responsible for increasing the speed every 2 successful hits until the predetermined max speed is reached. The speed setting is represented by a 4-bit binary number, and therefore there are 16 total speeds. Each successive speed is 3 dots/second faster than the previous. This block uses a select statement in VHDL and selects a "count" value based on the current speed setting. To create the derived clock, a counter continually counts up to this "count" value and once this count value is reached, the output is inverted. This generates a square wave with frequency $\frac{\text{mclk}}{2(\text{count}+1)}$. This signal is then put on a clock line and routed to all the other components which need it.

Score Manager

The Score Manager block contains two counters, one for each player's score. Its operation is simple, a player's score is incremented every time a score pulse is received. The score pulse comes from the score logic block (the code for which is within the main FSM module). The logic here is simple, the receiving player's paddle is not active when the ball is in the column directly in front of their paddle, it is considered a miss and the opposing player scores a point. Once the max score of 9 has been reached, an output signal is asserted to indicate a player has won. There are two possible output signals, max pulse 1, and max pulse 2, these indicate which player has won with the numbers directly corresponding to the player that has won.

Score Display Manager

The Score Display Manager takes the two players' scores as inputs and displays player 1's score in decimal format on the left most digit on the 7-segment display, and player 2's score on the right most digit. This is accomplished by the use of a 2 binary to binary-coded-decimal (BCD) converters, a multiplexer, a counter, and a BCD to 7-segment display decoder. First the binary scores are converted to BCD's using the converters. The counter is only a one-bit counter as we only have two digits to multiplex between. The counter is used to determine which BCD gets passed on to the BCD to 7-segment display decoder and to determine which anode is powered. If anode 0 is powered, the leftmost digit will be lit, if anode 3 is powered, the rightmost digit will be lit. The BCD to 7-segment display decoder is used to determine which cathodes (and hence which display segments) are lit. This whole block is run off another derived clock which provides a 500Hz frequency. This is because there is no need to update the display any faster than this and the use of a slower clock can save power.

Problems Encountered

There were a number of problems encountered during the development process. For example, we had some trouble trying to interface with the LED matrix display and getting the ball to display and move properly. After experimenting for a bit, we learned we had some of the signals inverted. We also had issues with our original architecture which separated out all the state machines and logic blocks and placed them separate files/modules. While they can be thought of as logically separate, a lot of them depend on the current state of the Main FSM, and we didn't want to have to create ports in many separate entities to pass the Main FSM's current state around, so we ended up including multiple state machines (and some of the other logic blocks) inside the single VDHL file/module for the Main FSM. We also had some issues with trying to use combinational logic for some of the more advanced outputs that ended up being depending on both an input (i.e. paddle button) and the Main FSM's state.

We also had an issue with the ball's position upon power. We wanted the ball to appear in column 14 for a new game, but upon powerup it defaulted to column 0. We did some research online and were able to fix this by specifying an initialization value in the VHDL code.

We also encountered and number of issues with getting the ball to speed up properly. We needed the ball to speed up every two successful hits, but we also needed it to update independent of hit pulses when there was a new game and it needed to be reset. This took awhile to figure out how to accomplish, but in the end the solution was fairly simple. We simply created a process and included both a successful hit pulse and a reset signal in the sensitivity list.

Resource Summary

A resource use summary generated by Xilinx ISE can be found in the attached documents.

Conclusion

Overall, this project was very time consuming, but it was a great learning process and a fun project to work on. We encountered a number of challenges and had to use some creative thinking and problem-solving skills to overcome them and workout a solution, but it was highly rewarding to see the circuit functioning exactly as desired in the end!