EE/CS-172L/130L : Digital Logic Design

Final Project Report

Group Members

Syed Muhammad Ali Naqvi, sn07590
Syed Muhammad Ammar Ali Jaffri, sa07695
Syed Muslim Hussain, sa07730
Zaryan Ahmed Siddiqui, zs07191

Instructor

Dr. Farhan Khan

Research Assistants

Hira Mustafa

Wednesday, December 12 2022

Habib University, Fall 2022

Contents

1 Game Description

The project is an interactive two-player game based on FPGA programming, and the concepts learned in the course. The challenge of the game is to start on a grid of boxes and move on the grid to make blocks. The user which manages to make the most boxes in the given time wins. Our original idea was that each player while moving on the grid will leave a trail and the other player will lose if they collide with this trail. Furthermore, the player can also reclaim the area conquered by the other player by simply making the block in the opponent's area. But during development, we modified the game such that to make the region you have to visit the block and both players can occupy each other's region and at the end of some predefined time the player with the larger region wins. The player with the most score will win even if both players collide.

2 User Flow Diagram

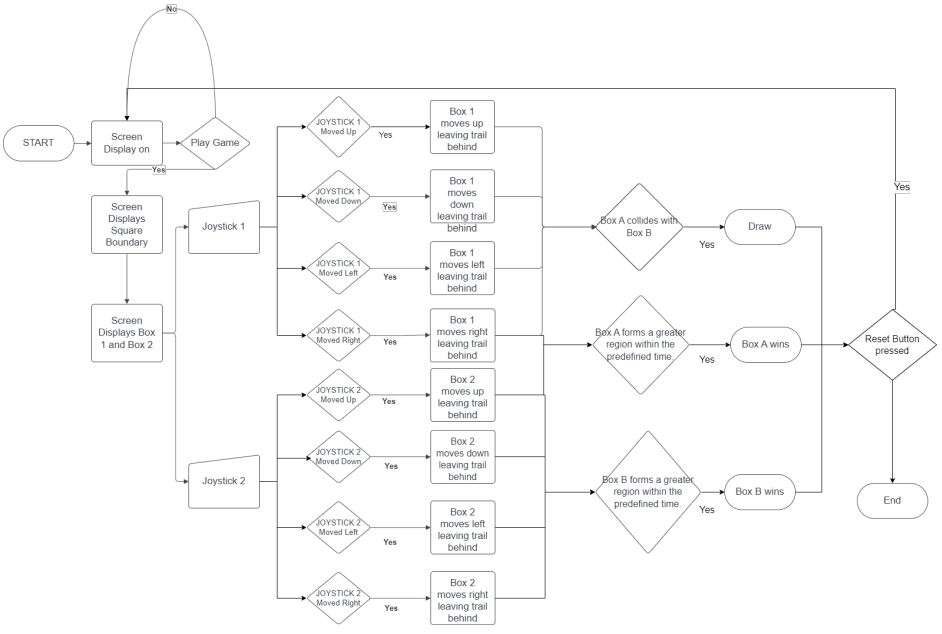

Figure 1: User Flow Diagram

3 Block Diagram for Game

The game is developed to the point that the screen and movement of the block are done through these respective modules, the block diagram for which is given below.

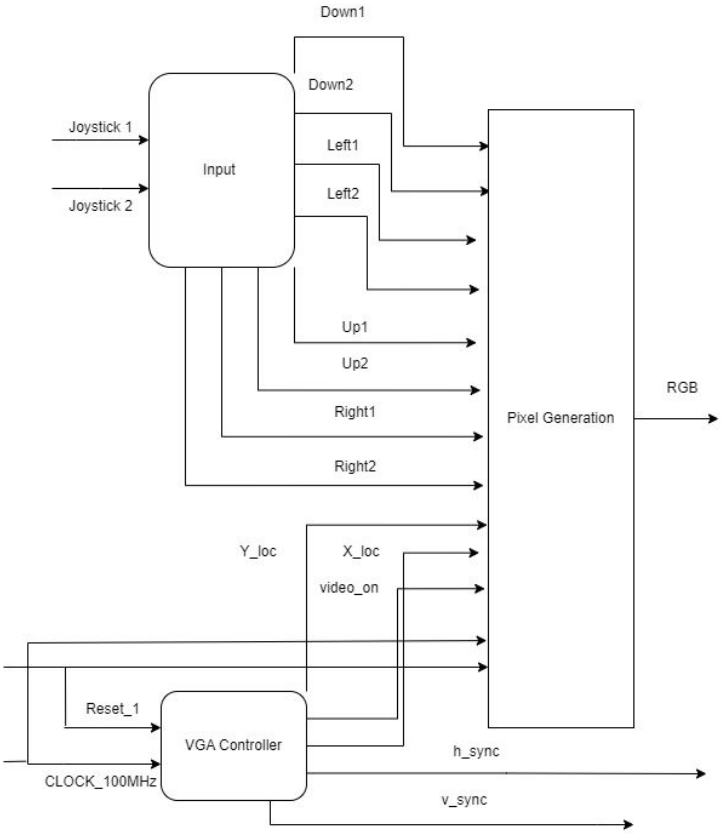

Figure 2: Block Diagram for modules used in Game

4 Input Block:

4.1 Joystick (Analog) Input

For the input, we have used two ordinary, non-pmod joysticks. A simple modular joystick contains two 10k Ohm potentiometers connected in parallel

each of which can be used to give two raw voltage signals respectively i.e one for each axis. They can take a maximum of 5V as the source voltage. For using it with the FPGA, we needed to convert the raw (analog) voltage signals to digital form. For this, we used the [1] built in the FPGA which can convert analog signal ranging from 0-1V to digital signal. We power the Joystick using the 3.3V source provided by the FPGA. Since the XADC can only take voltages ranging from 0-1V, we made a circuit to step down the voltage accordingly.

Note: This circuit is already discussed in the [2] as shared earlier.

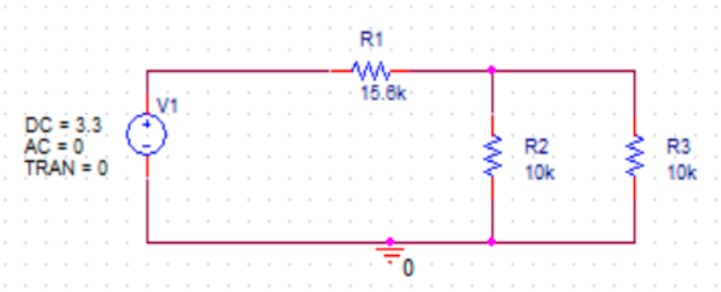

Figure 3: The Figure shows the circuit built to step down voltage from 3.3V to approximately 0.8 V, where R2 and R3 are two 10k ohm potentiometers inside the joystick.

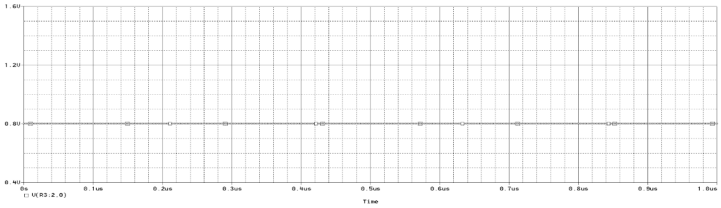

Figure 4: The Figures shows the voltage obtained,0.8V, either we move joystick up or right which results in maximum resistance of potentiometer of 10k ohm.

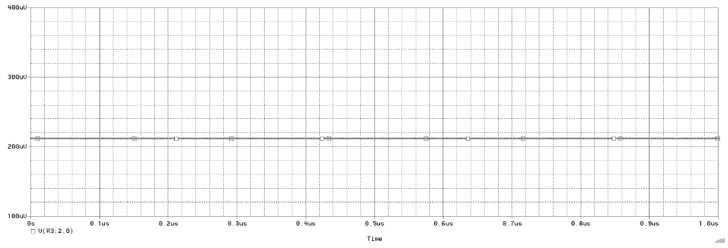

Figure 5: The Figures shows the voltage obtained,0.003V, either we move joystick down or left which results in minimum resistance of potentiometer of 1 ohm.

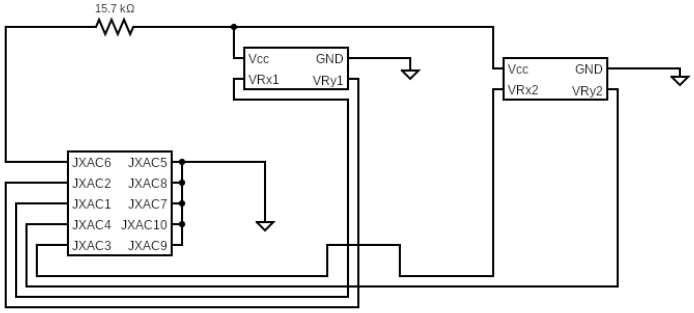

Figure 6: The figure shows the whole circuit. JXAC6 is the 3V voltage source from Basys3 and JXAC5 is the Basys ground

The XADC is capable of taking for 4 differential input voltage values. Each of the 4 input pairs is assigned a status register and its channel address as given [1]. The status registers are responsible for storing the ADC measurement from their corresponding input. The XADC converts the signal from each input pair into a digital signals and stores it in the corresponding status register. Since the XADC has only one data_out port available, the XADC can only cater to only one of the four inputs at one instance of time. i.e It can only read and direct output values from one status register to the data_out port at an instance of time. This means that we cannot have all of our four input signals available simultaneously and we have to read from the four status registers in a serial manner. The XADC takes an 'address

channel' as input which determines which of the 4 inputs' status registers will it be reading from at the output.

Inorder to interface two joysticks we used all four of the inputs (two for each joystick). To read from all four inputs, we used a 4x1 multiplexer which has address for each of the four XADC inputs. For selecting the address_in, we have used a counter from 0-3 which increments on the positive edge of the clock. In this way, at each positive edge of clock, we take input from each of the four potentiometers serially (one at a time), but the selection of address bus is so fast that it gives us the values for all four inputs almost immediately (without any noticeable delay). As discussed above, the output we get is a 16bit number. Since we do not require such precision, we only take the 4 MSB's from the output, thus giving us values from 0-15 in decimal. When the voltage is around 0.76 (maximum) i.e when an axis of the joystick is moved to its maximum, we get value around 15 and when it is 0V (minimum) i.e when an axis of joystick is moved to its minimum, we get 0. And in this way, we get values between 0-15 linearly when the joystick is moved because to the linear behaviour of resistor.

Also, we have an output signal drdy_out which tells us when to read values from the output of the xadc.

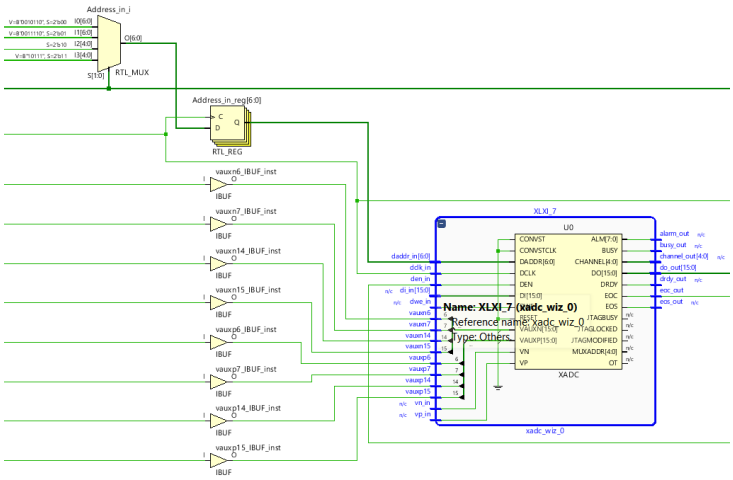

Figure 7: RTL schematic for joystick input

4.2 Reset Button

The reset button is an on FPGA push button which gives a high signal when pressed. At any instance of time, when the reset button is pressed, the game resets. Its exact function is discussed in the game control.

5 Game Control

5.1 Using the Joystick

For our game, we need four movements for each of the two players. We define 4 registers, two for each of the joystick inputs. i.e VrX1, VrX2, VrY1, VrY2. Also, note that in addition to switching addresses, we are at the same time directing the output from XADC to four different wires. So for instance at some time t_0, when the data is being read from the potentiometer (say) Y_1, then the value of VrY1 is updated while the other registers maintain their previous read values. Accordingly, the values of up1 and down1 are also updated at this instance of time while the others maintain their state. This is discussed further below.

We have 8 binary signals, i.e up1, down1, left1, right1, up2, down2, left2, right2. We know that when the joystick is in idle state, the voltages for both x and y lie somewhere between (0.76-0)/2 = 0.3V. This also means that at idle state we would get value around (15-0)/2 = 6 to 8 (decimal). So we define our conditions such that when the input for (say) VrY1 is less than 6, the game module should receive a signal for moving down i.e down1=1 , up1=0. And when it is greater than 8 then the game should receive a signal to move up i.e up1=1, down1=0. And when VRY1 is between 6 and 8 then both up1 and down1 are given 0. Similarly we set conditions for right1, down1 and for the other joystick too.

5.2 Reset Button

When the game module receives a high signal for reset, it resets the game. This means that the game goes back to its initial state. The reset button can be used to restart the game at any time be it in the middle of the game or when the game ends.

6 Output Block:

The output block takes input from the input block, processes it and gives 3 outputs:

- 12-bit RGB value
- h_sync
- v_sync

h_sync and v_sync are the values for the x and y value of the current pixel and the RGB value determines the color of the current pixel.

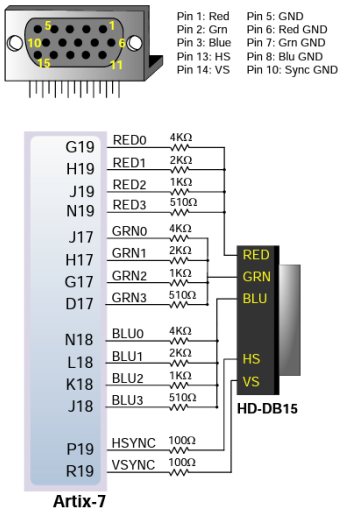

Figure 8: Pin configurations of VGA cable

Out of the 12 RGB bits ,the first 4 bits are for the color Red, the middle 4 bits are for the color Green and the last 4 bits are for the color Blue. These 12 bits are going into an Digital to Analog Converter (DAC), as shown in Figure 8, and that analog signal is given to the VGA cable.

The output block consists of the following modules:

- VGA Controller

- Pixel Generation

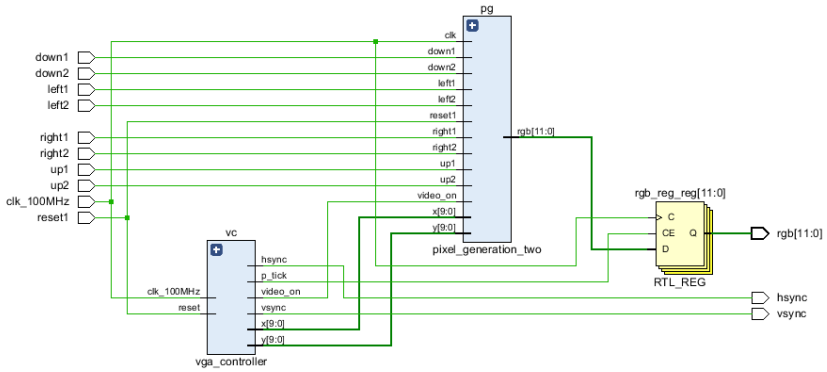

Figure 9: The block diagram of the Output Block implemented as of yet for the generation of VGA screen for the static and dynamic frame.

6.1 VGA Controller

- Hcounter
- Vcounter
- Clock Divider
- VGA Sync

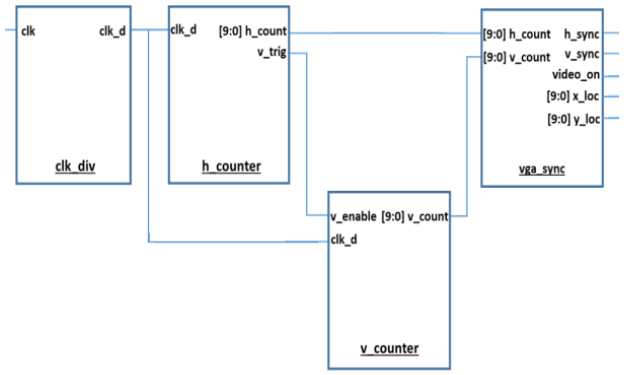

Figure 10: The block diagram of the VGA Controller as implemented in the lab.

H count and v count are the counters for the pixels for horizontal and vertical pixels, respectively. hcount goes from 0 to 799 which is 800 pixels and vcount goes from 0 to 524 which is 525 pixels. The visible region is only 640x480 pixels, the other values are for horizontal and vertical back and front porches and retrace area for syncing. As there are 800 pixels on the horizontal axis and 525 pixels on the vertical axis we want to update each frame 60 times a second so:

$$800_{\text{hcount}} * 525_{\text{vcount}} * 60_{\text{fps}} = 25 \text{ MHz Clock}$$

Hence, we are using the clock divider to convert the 100MHz default clock (W5) in BASYS-3 into a 25MHz clock.

6.2 Pixel Generation

The Pixel Generation block consists of movement and VGA display block.

6.2.1 VGA Display Block

As the colors of the background, borders and the blocks is constant so, in the Pixel Generation module we have defined some parameters for the colors of the background, borders and the blocks (i.e. the players). Moreover the

length of the square block is constant so we also created a parameter of the square size. Then some 10 bit registers are defined to store the values of the x and y location of each player (the position of the top left corner of the player).

By adding the square size in the x and y coordinates of the player we can find the whole area that the player is occupying and color that area according to the color of the player. For coloring we defined some 1 bit wires for instance, the wire for player A will set to 1 of the current pixel is is in the area that the block of player A is occupying and then at the end of the code there is a conditional block that checks which wire is ON for the current pixel and outputs the corresponding RGB value which is predefined in the form of parameters.

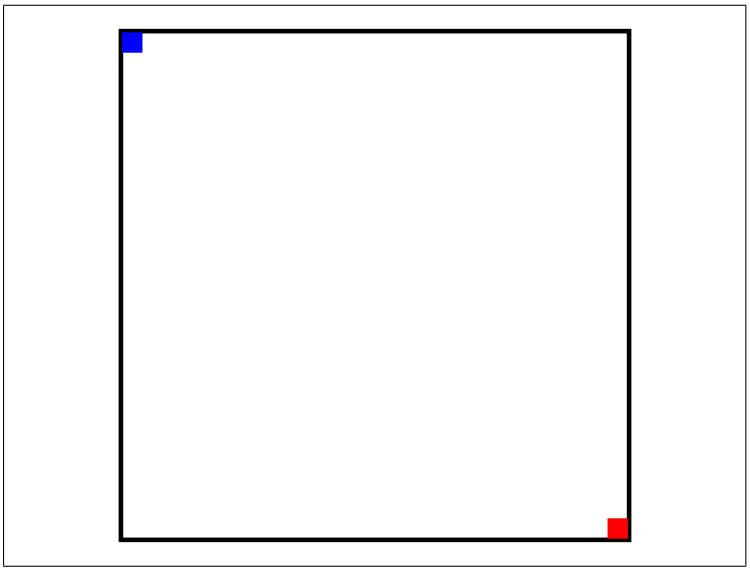

Figure 11: Static frame of the game

6.2.2 Movement Block

The pixel generation takes eight inputs for the movement of the two players. The name of inputs for player 1 are

- Up1
- Down1
- Right1
- Left1

For player 2, the name of inputs are

- Up2
- Down2
- Right2
- Left2

*Note that the inputs for the pixel generation are coming from the input block that takes the input using the joystick.

The Movement block is modelled as a Finite State Machine, precisely Moore Machine FSM, where there are two states

- Increment
- Decrement

At any one time, the input is taken through a MUX which switches between the inputs coming from the joystick in order such that at one time the combination of input received is

(R1, L1), (U1, D1), (R2, L2), (U2, D2).

Therefore there are two inputs to cater to at every positive edge of the clock. Hence the FSM was designed keeping this in mind. The inputs to the FSM are either of the above combinations of inputs. Assigning the variables A and B to the inputs. Either A can be 1 or B can be 1. Since the input is taken from the joystick, therefore A and B cannot be 1 at one time. Hence the combination A and B equal to 1 is taken as a don't care condition.

The desired output from the FSM is the signal which tells whether the block is moving up or down, left or right, and therefore is essentially a signal telling whether the location of the block has to be decremented or incremented.

The defined states and respective output are

State	Output
Idle State	No Movement
Increment	Increase the x/y location
Decrement	Decrease the x/y location

Table 1: Defined State Diagrams and Outputs

The variable assignment for outputs is as follows

Output	Variable Assignment
Increase the x/y location	O_1
Decrease the x/y location	O_0

Table 2: Variable Assignment for Output

The State assignment for the inputs and states is as follows, and is done in binary numbers

Input	Movement Type
A	Down/Right
B	Up/Left

Table 3: State Assignment for Input

A is 1 for down/right and 0 otherwise. Likewise, B is 1 for Up/Left and 0 otherwise.

Since there are three states, after the binary assignment of the states we get two state variables which we have named, Q_1 and Q_0

The binary assignment for the states in terms of binary values of state variables is

State	Binary State Assignment	State Outputs
	Q_1Q_0	O_1 O_0
Idle State	00	0 0
Increment	01	1 0
Decrement	10	0 1

Table 4: State Assignment for States and Outputs

The resulting encoded state transition table is as follows

PresentState		*Input*		*NextState*		*Output*	
Q_1	Q_0	A	B	Q_1	Q_0	O_1	O_0
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1
0	0	1	0	0	1	1	0
0	0	1	1	X	X	X	X
0	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	0
0	1	1	1	X	X	X	X
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
1	0	1	0	0	1	0	1
1	0	1	1	X	X	X	X

Table 5: State Table for Movement of Block FSM

The resulting state transition diagram is as follows

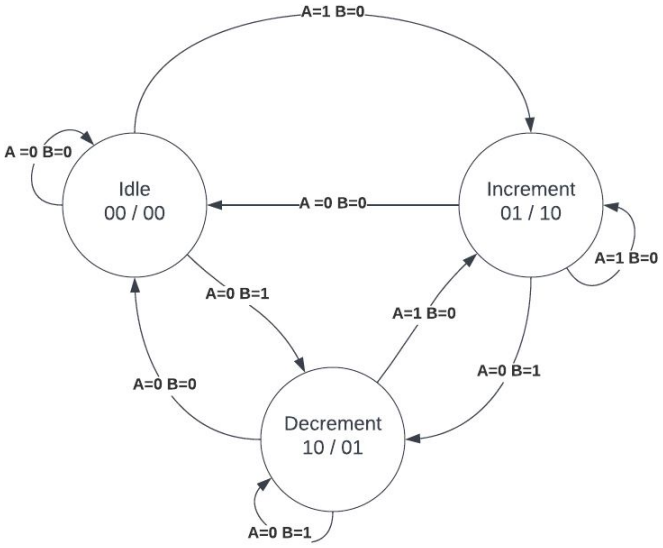

Figure 12: Movement of Block FSM

7 Game Logic Moore FSM

The grid of the game was divided into 625 blocks of dimension 18X18 pixels. For each block, 5 states are defined namely

- Idle State
- Player A
- Player B
- Region A
- Region B

Furthermore there are two inputs to the FSM. The output of the FSM is the states themselves, the value of which would be stored in the registers.

The two inputs are A and B. A corresponds to the input when the block is visited by A, that is when the coordinates of the block on the grid match the coordinates of player A. B corresponds to the input when the block is visited by B, that is when the coordinates of the block on the grid match the coordinates of player B.

7.1 States Description

Idle state is when the block is occupied by neither player and the block is empty that is it does not contain the region for either player A or player B.

Player A state is when the block is occupied by Player A, that is the current location of player A is in the block.

Player B state is when the block is occupied by Player B, that is the current location of player B is in the block.

Region A is the state when the block is in the region formed by A while moving.

Region B is the state when the block is in the region formed by B while moving.

Another sixth state was defined which was the collision state, which is when the two players are colliding with each other.

7.2 Input and State Assignment

The inputs A and B are assigned binary values.

A is 1 when the coordinates of the block match the location of player A, that is A has visited the block, and 0 otherwise.

B is 1 when the coordinates of the block match the location of player B, that is B has visited the block, and 0 otherwise.

Furthermore as for states, since there are six states, after the binary assignment of the states we get three state variables which we have named as , Q_2 and Q_1 and Q_0.

The binary assignment for the states in terms of binary values of state variables is

State	Binary State Assignment $Q_2Q_1Q_0$
Idle	000
Player A	001
Player B	010
Region A	011
Player B	100
Collision	101

Table 6: State Assignment for States

7.3 State Table

The state table for the FSM is as follows

$PresentState$ Q_2	Q_1	Q_0	$Input$ A	B	$NextState$ Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	0	0	1	0	0	0	1
0	0	0	1	1	1	0	1
0	0	1	0	0	0	1	1
0	0	1	0	1	0	1	0
0	0	1	1	0	0	0	1
0	0	1	1	1	1	0	1
0	1	0	0	0	1	0	0
0	1	0	0	1	0	1	0
0	1	0	1	0	1	0	1
0	1	0	1	1	1	0	1
0	1	1	0	0	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	0	0	1	1
0	1	1	1	1	1	0	1
1	0	0	0	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	0	0	0	1
1	0	0	1	1	1	0	1
1	0	1	0	0	X	X	X
1	0	1	0	1	X	X	X
1	0	1	1	0	X	X	X
1	0	1	1	1	X	X	X

Table 7: State Table for Movement FSM

7.4 State Diagram

The resulting state diagram from the state table is as follows

Figure 13: State Diagram for the FSM

8 Play Screen switching FSM

The screen begins from the Welcome screen, where the user is given the option to choose between the two modes of the game, that is the Easy and Torture Mode, as shown below.

Figure 14: Welcome screen of the Game

From here, the user moves onto either of the two modes, by choosing from the button on the FPGA.

The FSM implemented for the screen switching is gate-level implementation.

From the description of our requirement, we need three states, that is the idle state, which will be the state which shows the welcome screen. Then we have two input buttons, when button C is selected, the Easy mode welcome screen is shown along with the rules for the game.

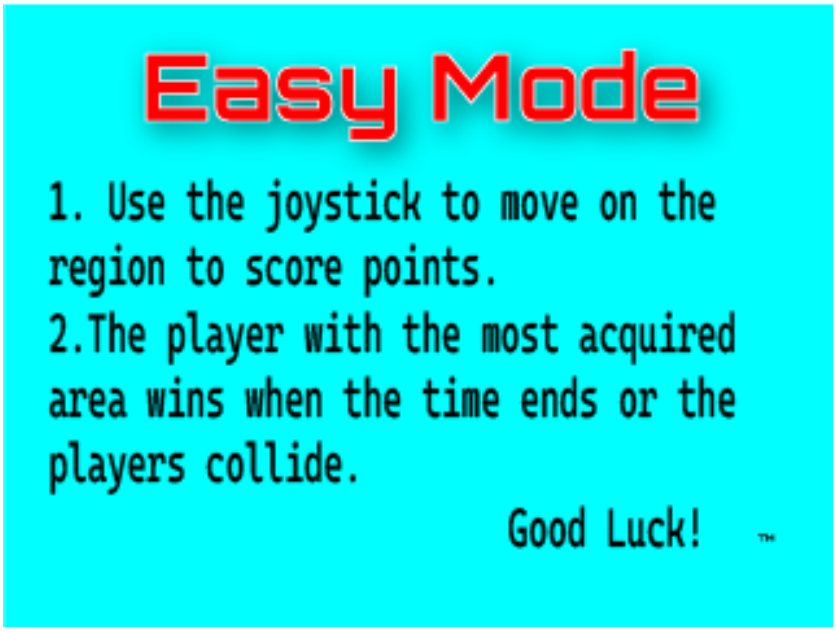

Figure 15: Easy Mode Game screen

When button D is pressed, then the game enters the state, where the Torture mode screen is shown with its rules, as shown below.

Figure 16: Easy Mode Game screen

8.1 State Assignment and Variables

There are three states, and hence there are two state variables A and B. The following table shows the state assignment and state type.

State	State Assignment (A B)
Idle	00
Easy	01
Torture	10

Table 8: State Assignment and Variables

8.2 State Table Diagram

Present State		Input		Next State		D Flip Flop Inputs	
A	B	C	D	A	B	D_A	D_B
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	1
0	0	1	0	1	0	1	0
0	0	1	1	1	0	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	0
1	0	1	1	1	0	1	0
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Table 9: Screen Selection (Moore) FSM

8.3 State Transition Diagram

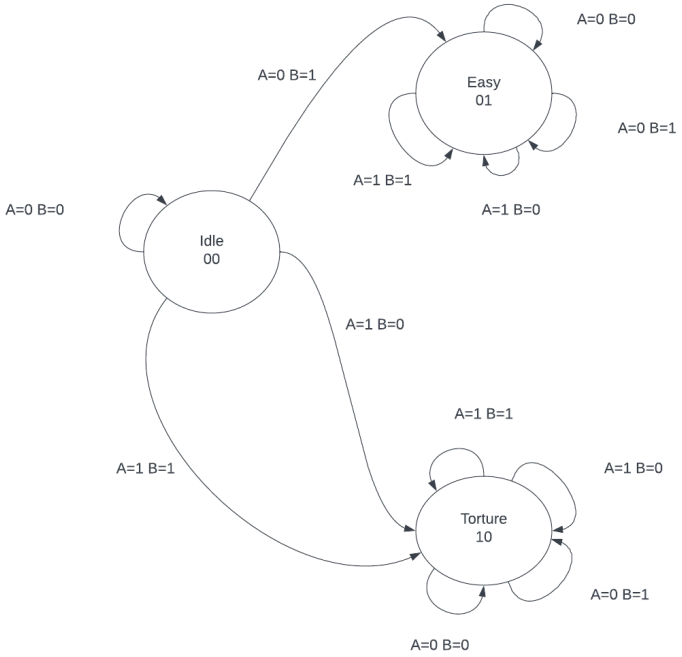

Figure 17: State Transition Diagram for Screen Selection FSM

8.4 Input Equations

$$D_A = A+B'C \tag{1}$$

$$D_B = B+A'C'D \tag{2}$$

8.5 Gate Level Implementation

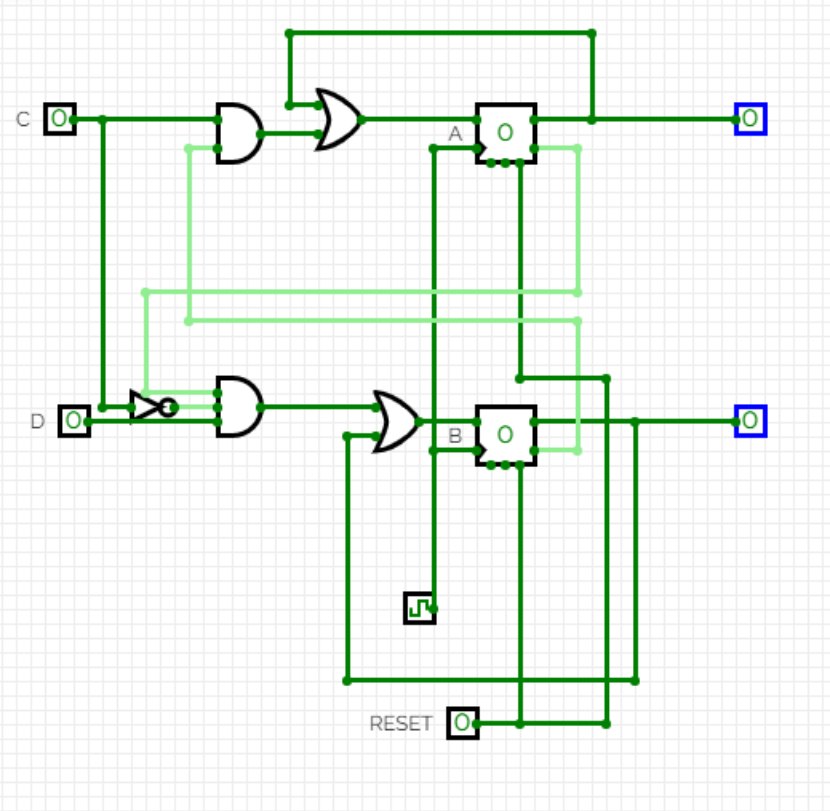

Figure 18: Gate Level Implementation of Selection FSM [3]

9 Arithmetic Logic Unit

The Arithmetic Logic Unit unit is used to perform the following tasks;

1. Scoring Mechanism
2. Comparator

9.1 Scoring Mechanism

The scoring mechanism is implemented on each of the blocks of the grid and it works almost on the same principle as the main FSM. There are 2 registers of 10-bit each to store the score of the players, both are being updated at each positive cycle of the clock. If the grid block is in the IDLE state then 0 is added to the total of red and 0 is added to the score of blue. If blue comes on the empty block, then 1 is added to the score of blue; if red comes on the empty block, then 1 is added to the score of red. If blue comes on the region of red then 1 is subtracted from the score of red and 1 is added to the score of blue and vice versa. Moreover, a bonus block appears after 20 seconds are passed into the game. if a player acquires the bonus block then 10 is added to the score of the player.

9.1.1 State Assignment and Variables

There are four states, and hence there are two state variables A and B. The following table shows the state assignment and state type.

State	State Assignment (A B)
Idle	00
Player A	01
Player B	10
Collision	11

Table 10: State Assignment and Variables

9.1.2 State Table Diagram

Q_t		Inputs		$Q(t+1)$		Outputs					
Q_1	Q_2	A	B	Q_1	Q_2	O_6	O_5	O_4	O_3	O_2	O_1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	0	0	0	0	0
0	0	1	0	1	0	0	0	1	0	0	0
0	0	1	1	X	X	X	X	X	X	X	X
0	1	0	0	0	1	0	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0	0	0
0	1	1	0	1	0	0	1	1	0	0	0
0	1	1	1	X	X	X	X	X	X	X	X
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	1	0	0	1	0	0
1	0	1	0	1	0	0	0	0	0	0	0
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	1	0
1	1	1	0	1	0	0	0	0	0	0	1
1	1	1	1	1	1	0	0	0	0	0	0

Table 11: Scoring Mechanism (Mealy) FSM

9.1.3 State Transition Diagram

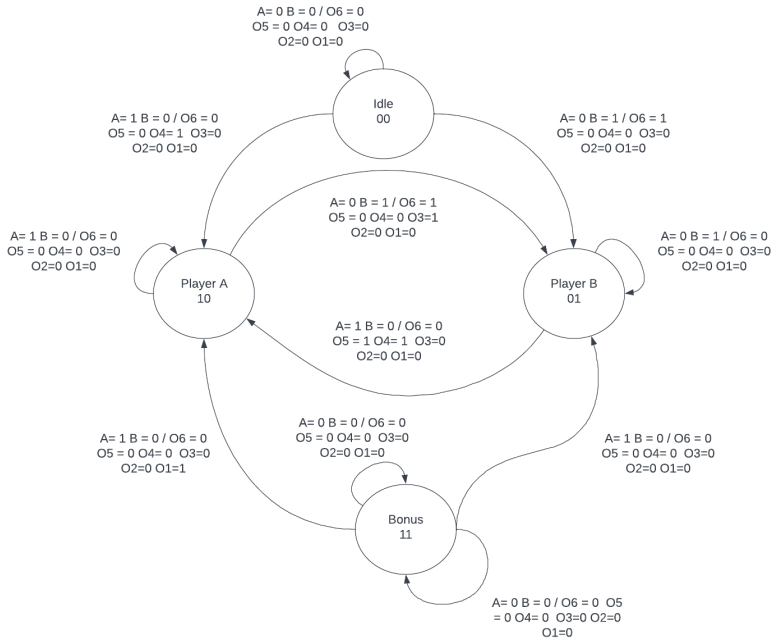

Figure 19: State Transition Diagram for Scoring FSM

9.2 Comparator

When the timer ends or the players collide, the game then goes to a comparator unit which compares the scores of both the players and outputs the 2-bit number accordingly.

Outputs (win_0win_1)	Actions
00	Draw
01	Player 1 Wins
10	Player 2 Wins
11	Don't care

10 Timer Implementation

For the movement FSM we are using a 10Hz clock which we made using a clock divider module that divides the 100MHz clock to 10Hz, we have used the same clock to implement the timer. We created a register to store the current value of the timer. as we wanted our timer to change every second and we had a 10Hz clock so we decremented the value of the timer at every 10^{th} cycle of the 10Hz clock. Which allowed the timer to go from 60 to 00 in exactly 1 minute. And when the timer hits 00, the game stops and goes to the comparator block. When we reset the game, the timer is again reset to the value of 60.

11 Score Display

We have displayed the score on the screen for which we used 2 main modules. In one of the modules, we defined the ASCII characters and stored them using a bit map. This module takes the rom-address as the input and gives the 8-bit output, which is the bitmap output corresponding to one y value(vertical count).

11.1 Binary to BCD

This module takes a 10-bit binary number as input. The largest value that a 10-bit binary number can show is 1023. So, the output of this module is 4 separate BCD numbers, each representing a single digit of the decimal equivalent number. The RTL schematic is as follows

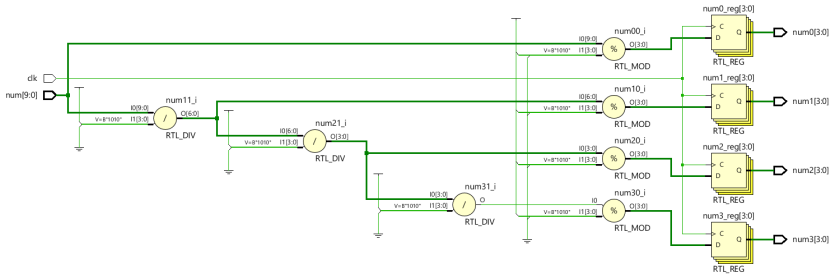

Figure 20: RTL Schematic for Binary to BCD

11.2 BCD to HEX

Next module is BCD to HEX which takes a BCD number as input and outputs its corresponding HEX value. The table below shows the BCD and HEX values of digits.

Digit	BCD	HEX
0	0000	30
1	0001	31
2	0010	32
3	0011	33
4	0100	34
5	0101	35
6	0110	36
7	0111	37
8	1000	38
9	1001	39

Table 12: BCD to HEX value Table

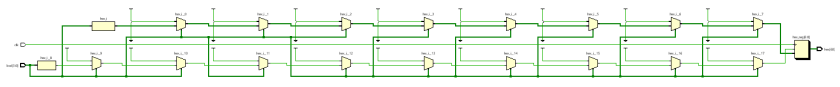

Figure 21: State Transition Diagram for Scoring FSM

12 Conclusion

In conclusion, the game is implemented by integrating the various blocks such as ALU, Memory, and Control Block.

13 Code Github Live Link

- Game-Project (Whole Repository) [4]

References

[1] "7 series fpgas and zynq-7000 soc xadc dual 12-bit 1 msps analog-to-digital converter user guide (ug480)." "Accessed: 12-12-2022". (2022).

[2] S. M. A. Naqvi, "Interfacing the ordinary joystick module using basys3 xadc," 2022, "Accessed: 04-11-2022".

[3] Z. A. Siddiqui. "Game fsm." "Accessed: 14-12-2022". (2022).

[4] "Territorialwarsbasys3." "Accessed: 12-12-2022". (2022).