CS 288 Project (Ping Pong)

Team Details: Group 10
Team Members:

- Shubham Mehta – 110050013
- Nilesh Satish Kulkarni – 110050007
- Shivam H Prasad – 110050041
- Prithviraj Billa – 110050065
- Nithin Kumar – 110050053

High Level Description:

We have implemented Ping-Pong game on Xilinx board in VHDL. Our design consists of following modules…

- **Paddle:** This is used to implement paddle movement in the game. Each paddle has dimensions (height=60px, width=12px). It takes two inputs say i1, i2 each using debounce.

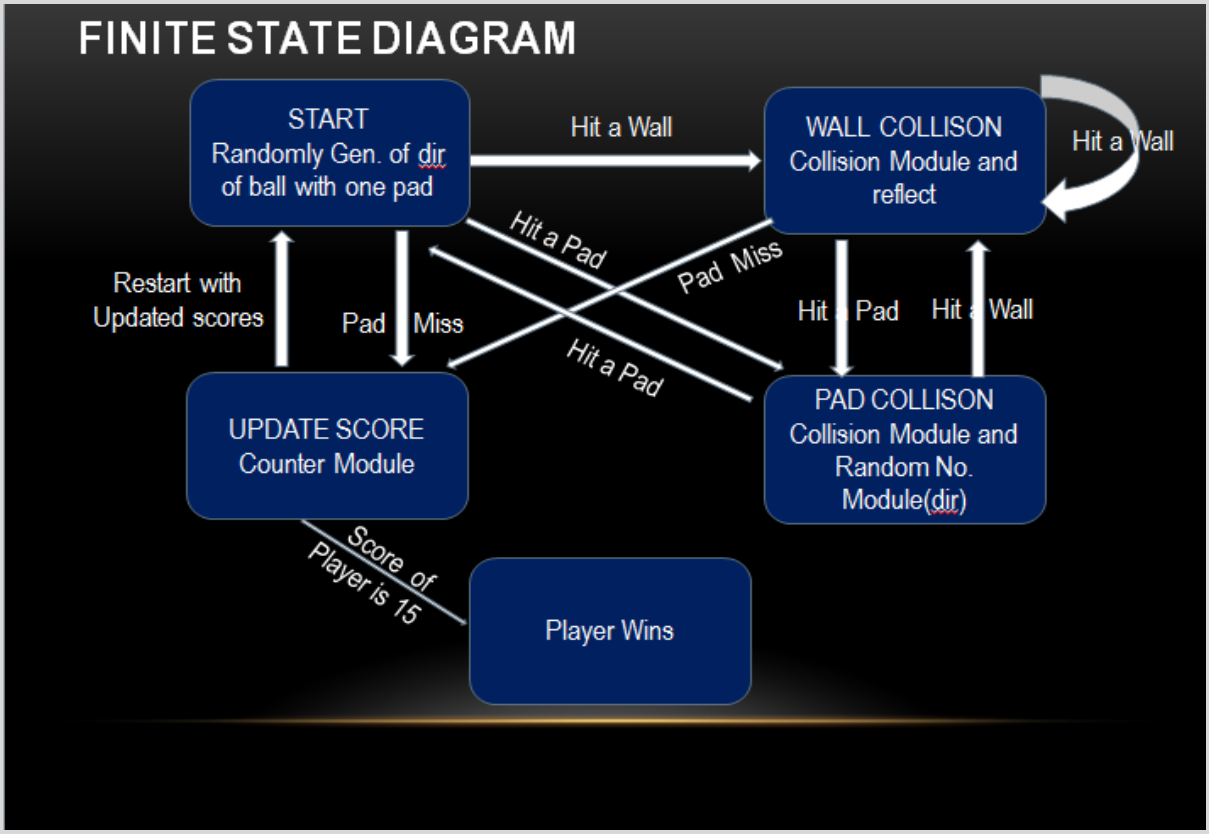
I1	I2	Result
'0'	'0'	No Movement
'0'	'1'	Moves UP
'1'	'0'	Moves Down
'1'	'1'	No Movement

- **RGB:** This module is used to get the colour info of any point on screen at any time. It takes co-ordinates (X,Y) as input and return R,G,B values of pixel corresponding to that co-ordinate as three outputs
- **Random Number Generator:** This module takes an integer input and generates a random binary number of input width.
- **Ball Module:** This module is heart and soul of the game. It takes y co-ordinates of both paddles, current x, y co-ordinates of ball as input. All this information is used to find the next updated position of ball at each rising edge of clock. It has velocity of ball as its internal signal. If the ball collides with the paddle then updated position and velocity of ball is the

position and velocity that it would have after collision. It also gives current scores as the output.

- **Pong:** This module consists of RGB, paddle, ball modules as its components. It takes X,Y co-ordinate of a point , its current RGB information, position of ball, and paddle movement related inputs (i.e. which paddle moved and in which direction) as inputs and gives the new RGB information of concerned point as output.
- **VTC_demo:** This module uses Pong module as its component and uses its RGB output to directly produce it on screen.

State Diagram Briefing:

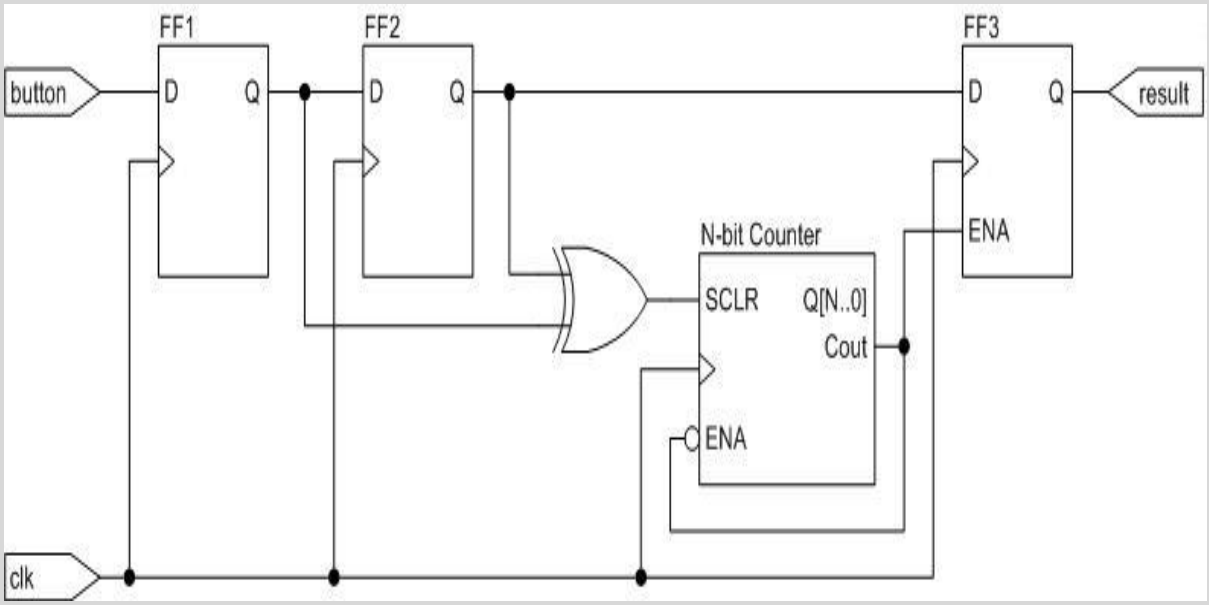


- The ball starts from a paddle
- Its speed and direction is decided by **randomNumber** module
- The collision detection is decided by the **BallMovement** module
- The speed is changed if collision occurs.
- The score counter is increased if some player misses a ball.

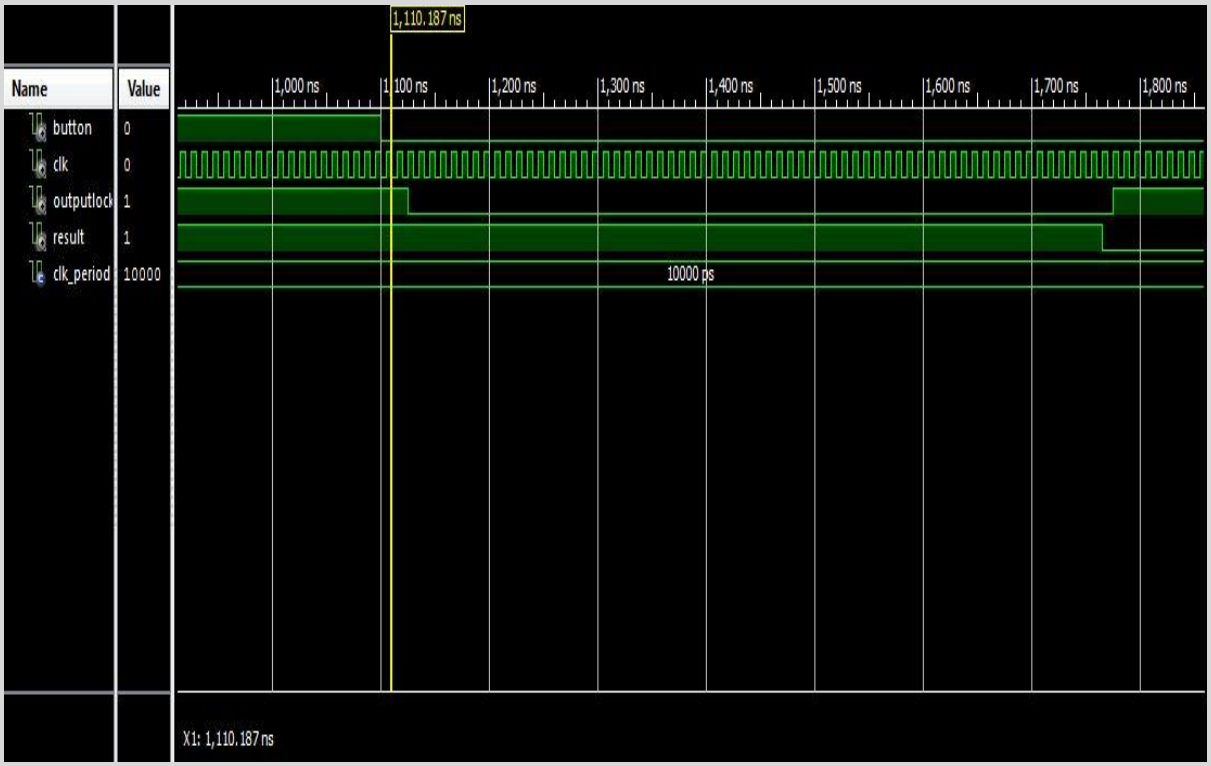
Detailed Description of Modules:

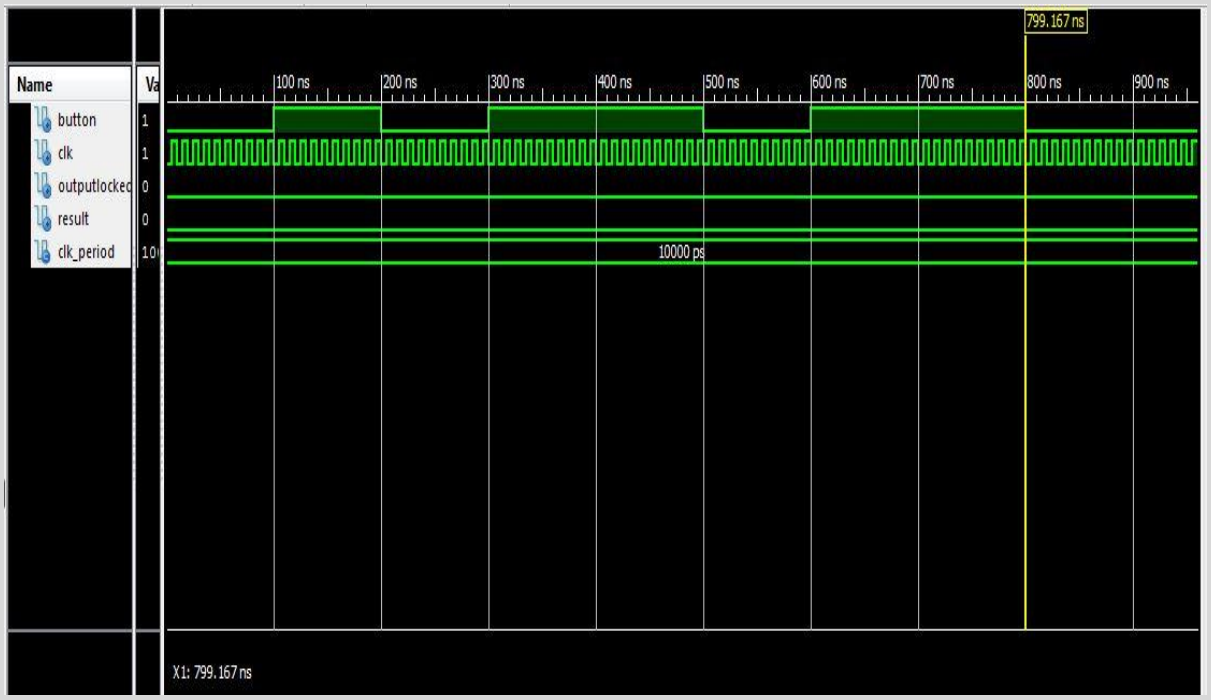
LOGIC BEHIND DEBOUNCE:

- **Bouncing** is a tendency to generate multiple signals when two metals come into contact.
- **De bouncing** is process which ensures that only one signal is sent when bouncing occurs.
- **De bouncing circuits:** The circuit consists of three D Flip flops, N bit counter and XOR gate.
- All the elements have same clk. N is dependent on the stability of button.
- The **D-Flip flops FF1 and FF2** are connected sequentially such that they store last two logic values of the button. Those two flip flops are connected to XOR gate which is in turn connected to N-bit counter.
- If the button levels changed, Values of FF1 and FF2 differ for a clock cycle, clearing n bit counter via XOR gate. If the levels are unchanged, then XOR gate releases counter synchronous clear and counter begins to count from N to 0, after counting it enables FF3 which gets values from FF2 and it is output.
- Size of the counter and the clk frequency determines the time period (P) that validates button stability.
- **P = (2^N + 2)/f**
- **Debounce Circuit Diagram**



- **Debounce implemented on ISE**



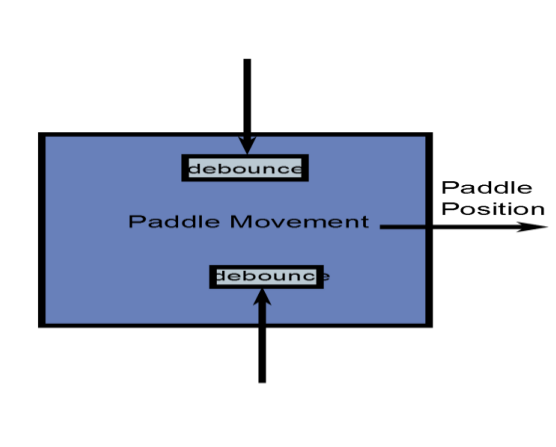


PADDLE MOVEMENT:

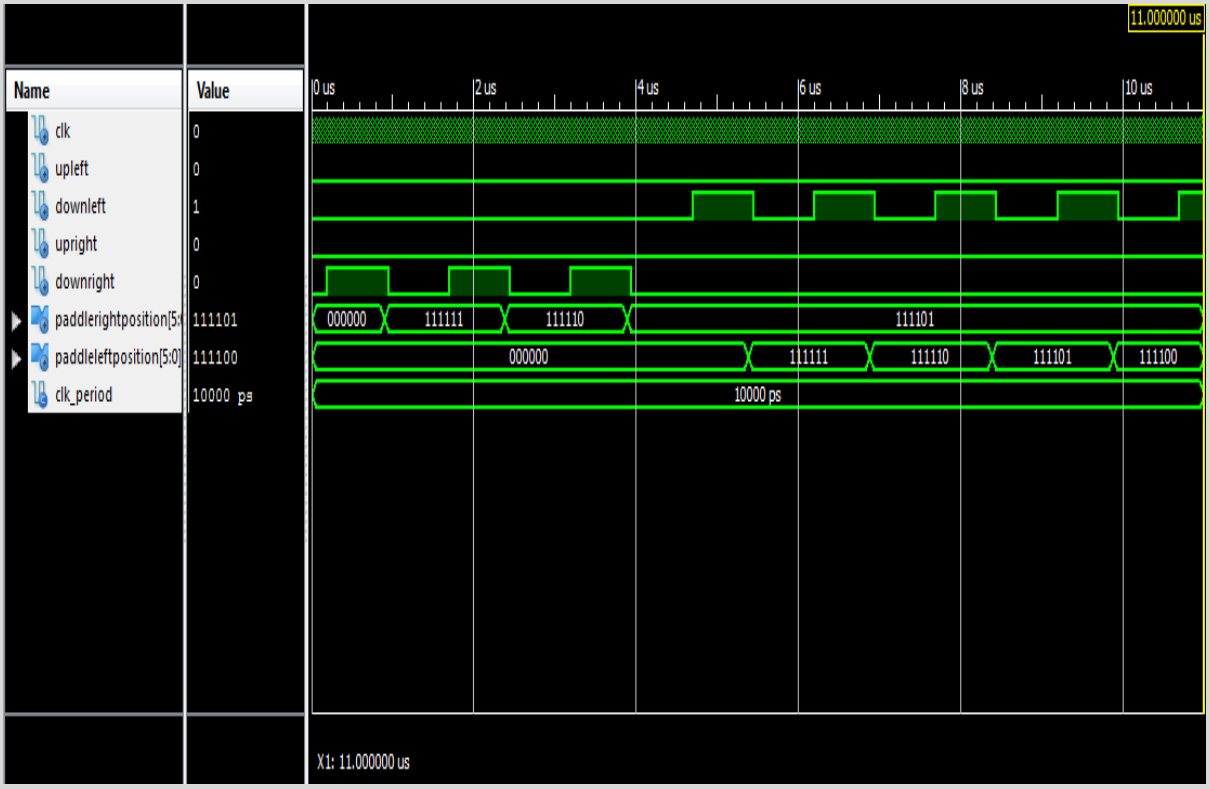
- **Paddle Movement** is controlled by two switches p1 and p2.

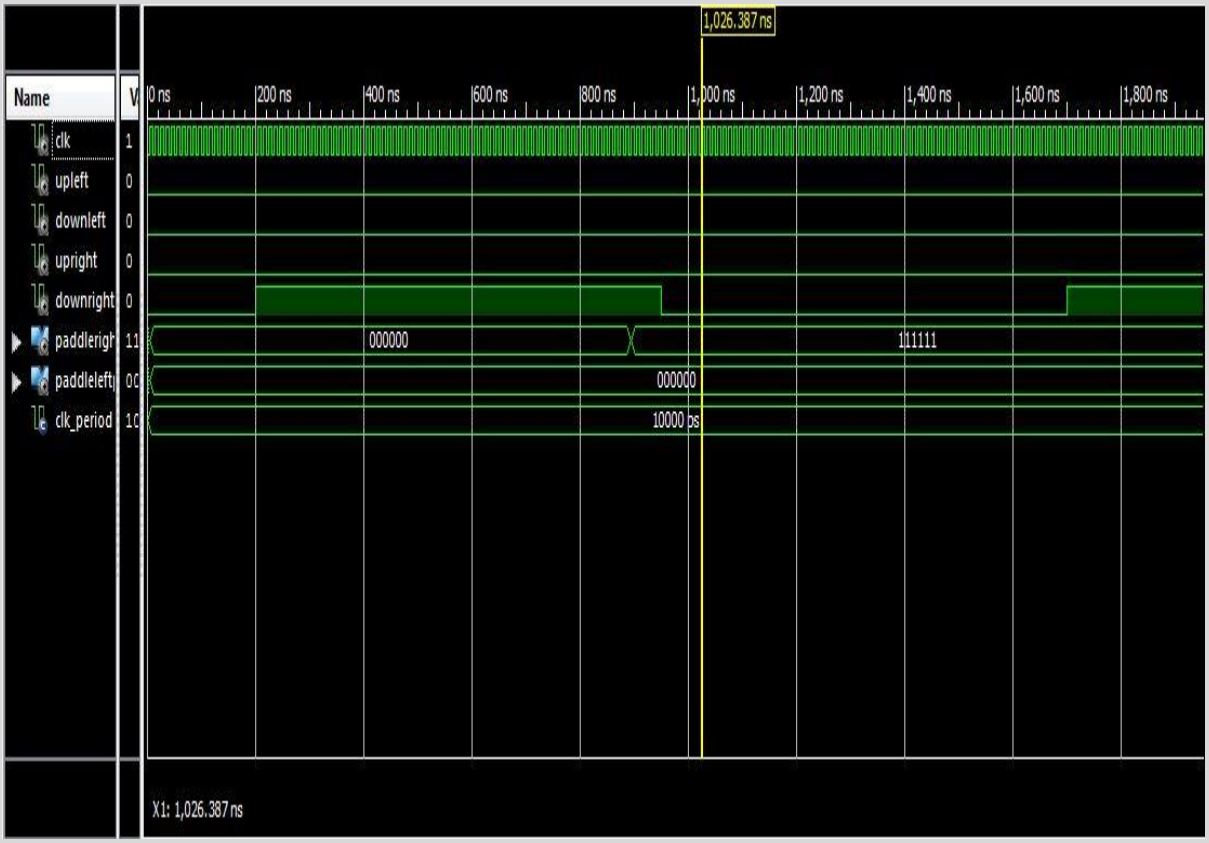
p1	p2	Result
'0'	'0'	No Movement
'0'	'1'	Moves UP
'1'	'0'	Moves Down
'1'	'1'	No Movement

- It outputs the paddle position.
- There are two modules which are associated with paddle movement.
- **upDebouce-debounce** which de bounces the up button and ensure the paddle movement in up direction.
- **downDebounce-debounce** which de bounces the down button and ensure the paddle movement in down direction.
- **Paddle Movement Diagram**



- **Paddle Module Simulated on ISE**



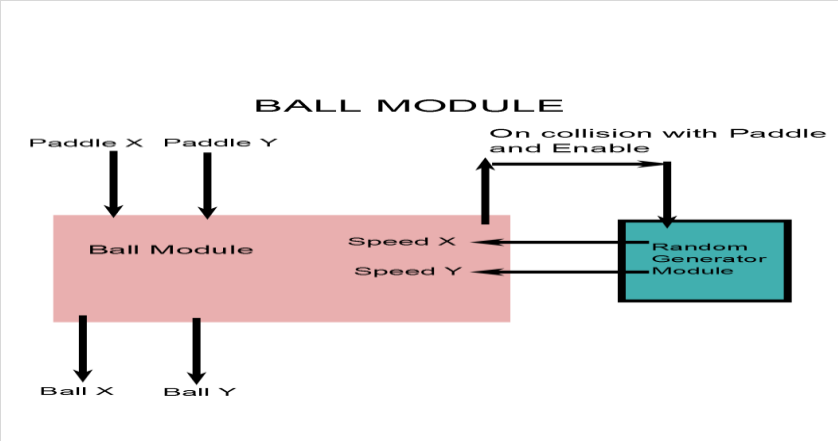


BALL MODULE:

- **Ball Module** checks if there is a collision and changes the speed of the ball accordingly.
- The position of paddles and clcok are given as inputs. The position of ball, the scores of the two players are outputs.
- Speed of the ball in internal variable and it is changed when collision occurs.
- There is a variable by name **checkCollision** which detects the collision with different surfaces.
- checkCollision states are

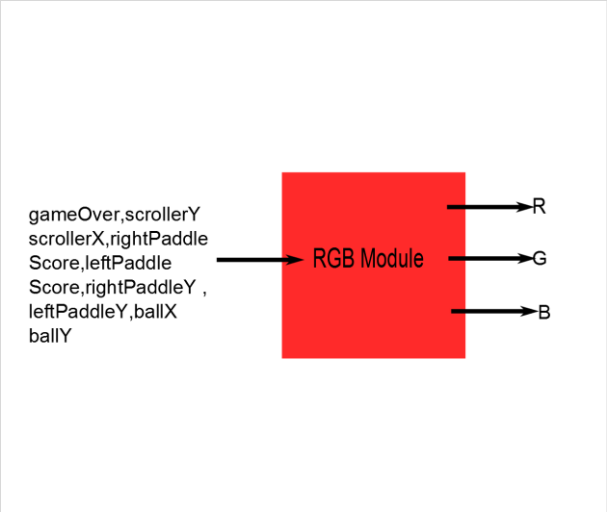
000	module detects no collision
001	module detects collision with left paddle
010	module detects collision with left wall
011	module detects collision with top wall
100	module detects collision with right paddle
101	module detects collision with right wall
110	module detects collision with bottom wall

- If reset is true, ball is returned to its initial coordinates, ball speed is returned to initial value. At every instant, the ball's position coordinates are calculated for next instant.
- If the next position coordinates are less than left Paddle coordinates, collision is detected on Left Paddle and if less than right Paddle coordinates, collision is detected on right Paddle.
- If the next Position coordinates exceeds the height or less than certain height, collision with the surface is detected. The speed of the ball of is changed depending upon the **random number generation module**.
- **Ball Module Diagram**



RGB MODULE:

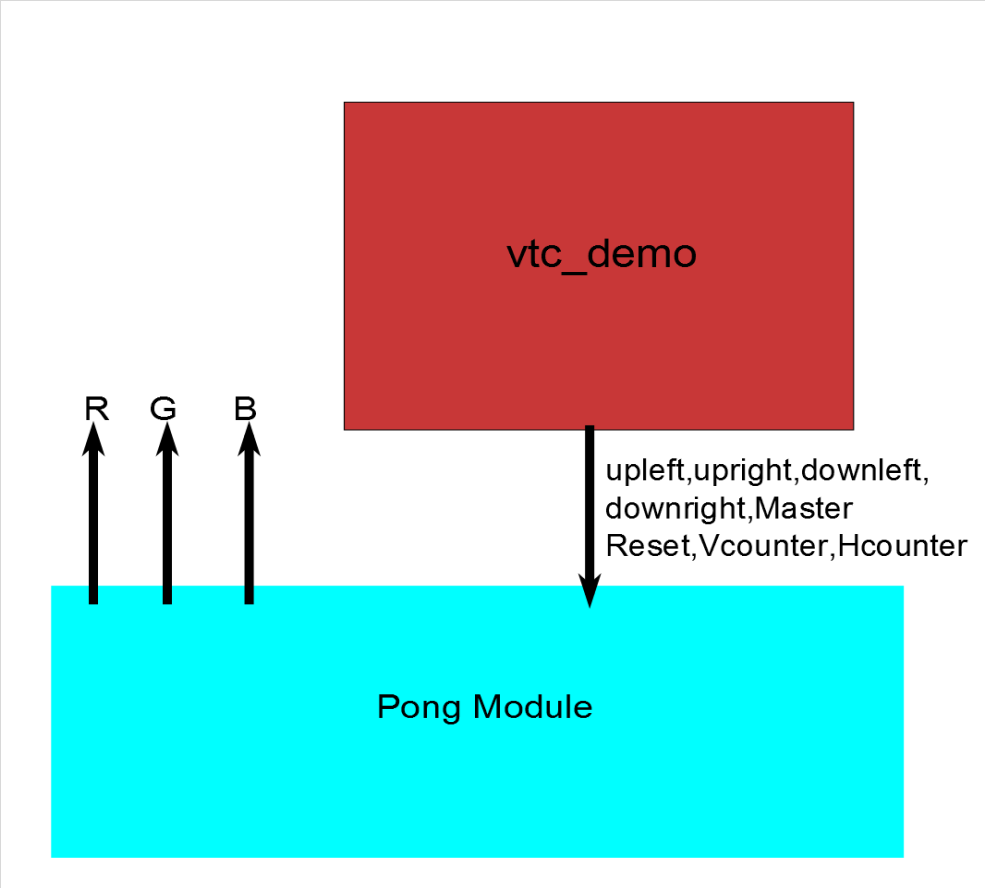
- The input for RGB module are leftPaddleY, rightPaddleY, ballX, ballY, leftPaddleScore, rightPaddleScore, scrollerX, scrollerY, gameOver
- **ballx, bally** are the positions of ball
- **leftPaddleY** is the position of left paddle
- **rightPaddleY** is the position of right paddle
- **scrollerX, scrollerY** are the current pixel values
- **RGB module** checks if the position of the scrollerX and scrollerY is on the ball or scoreboard or white line in between or the paddle and displays colour likewise
- **RGB Module Diagram**



PONG MODULE:

- The input for Pong Module are upLeft, downLeft, upRight, downRight, MasterReset, Start, Hcounter, Vcounter
- **upLeft, downLeft , upRight, downRight** are for giving inputs
- **MasterReset and Start** are for reset and start
- RGB Module in Pong Module takes **Hcounter, Vcounter** and gives values of red, green, blue as output

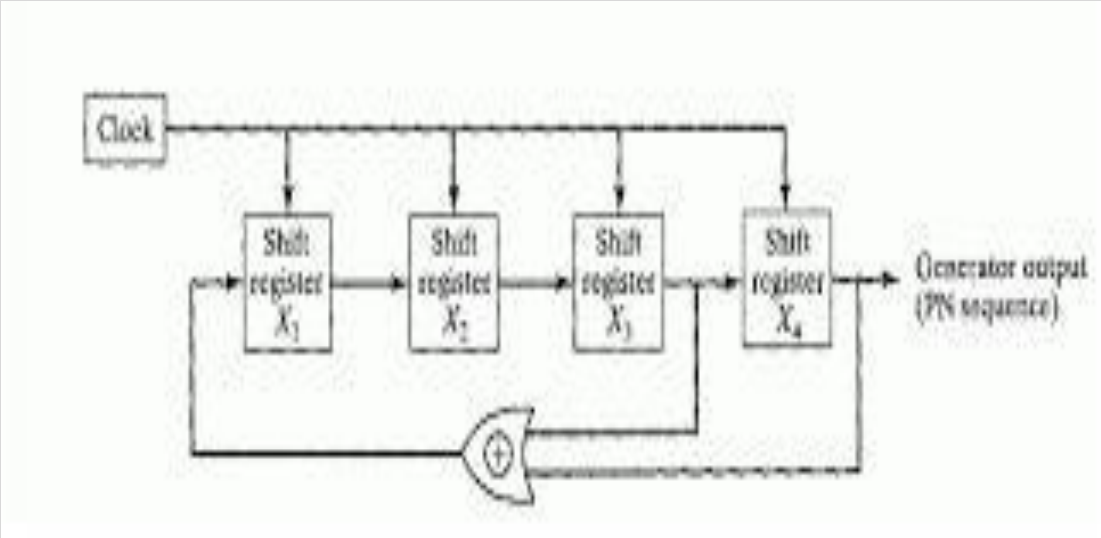
- This process takes care of scores, some user interrupts like the start button
- Reset Button, it also ensures whether the game is over or not by checking the Paddle scores whether they are 15 or not. If any of them are 15 it generates a **Game Over signal**.
- **Pong Module Diagram**



RANDOM NUMBER GENERATOR MODULE:

- This module takes **width (32bit integer)** as input. This defines the size of random logic vector that is generated at each rising clock edge
- If the value of width is n then random_temp is a vector which is defined as “100…” with n zeros

- At each rising edge next random number random_num is calculated as **temp=rand_temp(width-1) XOR random_temp(width-2)**, **rand_temp(-1 to 1) is assigned rand_temp(-2 to 0)** and rand_temp(0)=0
- **Random_num** is now made equal to this new rand_temp
- **Random Number Generator Diagram**

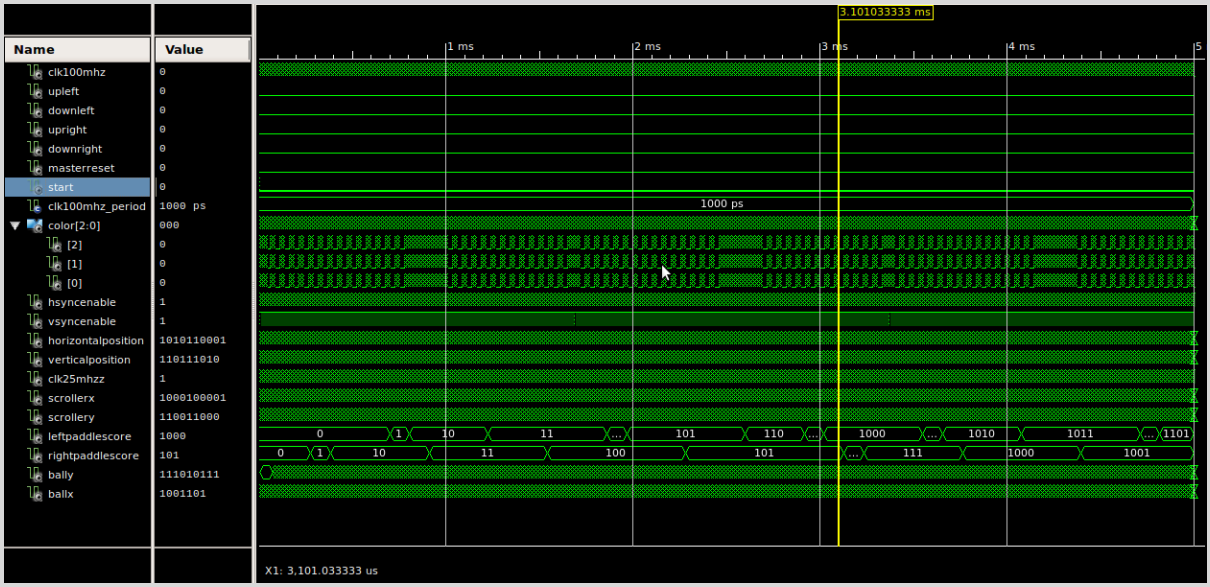


Conclusion:

We have successfully implemented ping pong game on FPGA board. Our all the modules are functioning properly.
We have taken inputs on FPGA boards from following pins…

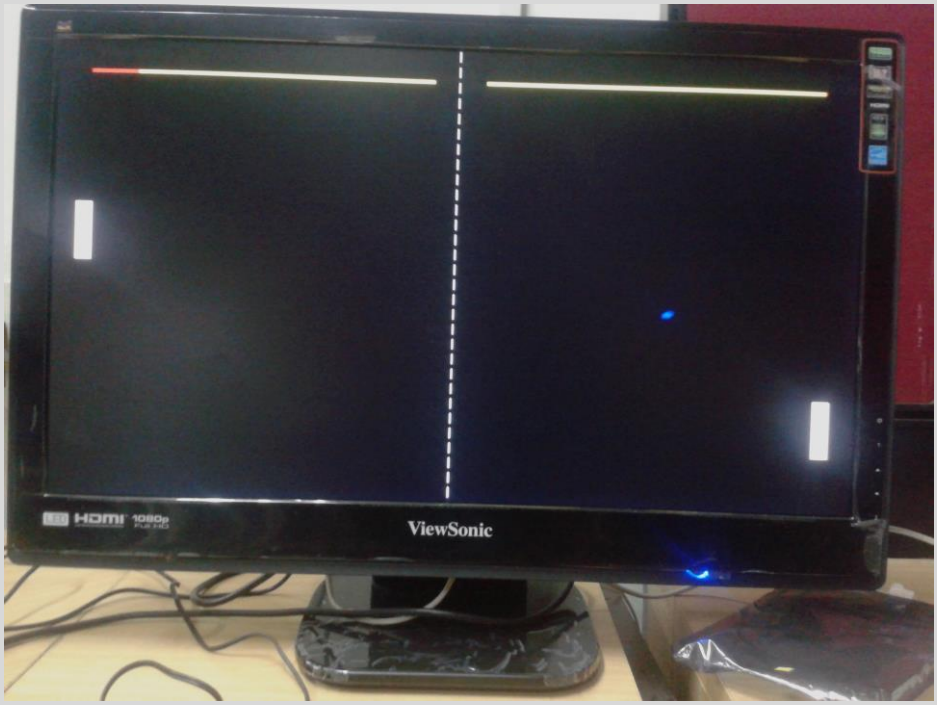
- N4 – up movement of left paddle (upLeft)
- P4 – down movement of left paddle (dowLeft)
- F6 – up movement of right paddle (upRight)
- P3 – down movement of right paddle (downRight)
- F5 – start button (Start)
- L15 – system clock (sys_clk)
- T15 – reset button (rstbtn)

Final Simulation of game on ISE

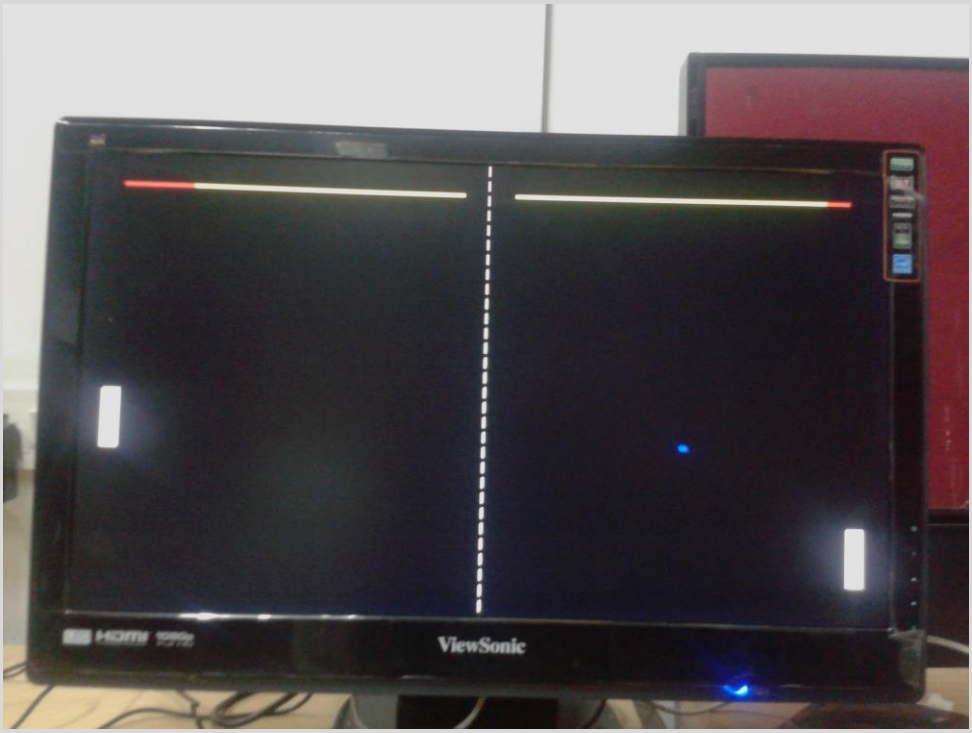


Here are some pictures of our game in practical working…

PIC1



PIC2



PIC3