



**Habib University**  
shaping futures

---

## **Ping Pong**

---

Submitted By

Muhammad Hasham Qazi<sup>1</sup> (05497)

Omer Rastgar<sup>1</sup> (05554)

Asad Tariq<sup>2</sup> (05439)

Fahad Shaikh<sup>2</sup> (05452)

EE-172/CS-130

Digital Logic and Design

Instructor

Junaid Ahmed Memon

RA

Hafsa Amanullah

From the departments of Electrical Engineering and Computer Science

Dhanani School of Science and Engineering

Habib University

9th December 2020

---

<sup>1</sup>Electrical Engineering 2023

<sup>2</sup>Computer Science 2023

## Game Features

The game presented is a single-player implementation of the classic sport Ping Pong. With respect to the salient features of the presented project, there is a implementation of a counter which serves as a score mechanism. The score is incremented with each successive hit off the player's paddle while the score is reset following the player's failure to make contact with the ball object. The main objective of the game is for the player to achieve a high of a score that is possible - the game ends when the player achieves a specific score.

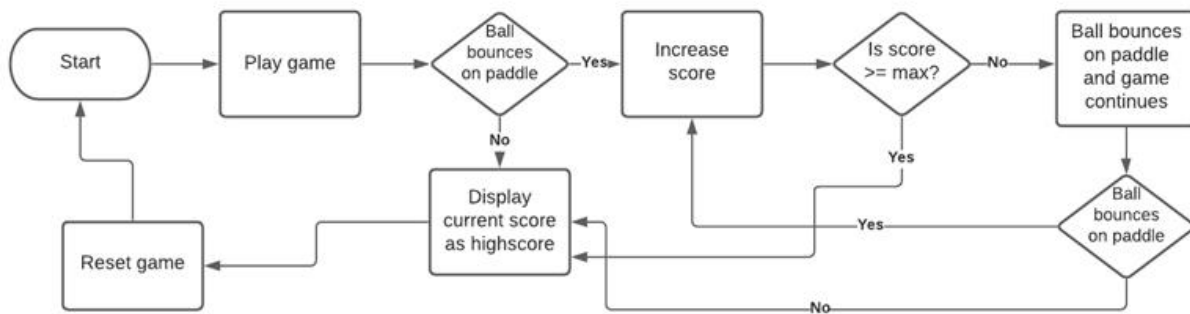


Figure 1: User Flow Diagram of the implemented game

The game will start with the ball in the middle of the screen and after pressing a defined key the ball will start moving. The aforementioned ball will collide with the borders of the screen causing it to bounce back and change direction. When the player makes contact with the ball via their paddle, the score will increase and the ball will bounce from the paddle and the game will continue. When the player fails to make contact with the ball and thus, allowing the ball to collide with the bottom border of the screen causing the game to **end**, **output** of the score on the screen and the game will **reset**. While the game is in play, if the score exceeds or equals to a defined '*winning score*' causing the game to **end**, **output** of the score on the screen and the game will **reset**. **Note**, when the game resets the score becomes 0.

## The Input Block

The UART (**Universal Asynchronous Receiver Transmitter**) is the simplest way of communicating with your FPGA board. We require the UART Receiver component to input serial data through our computer to the BASYS3 board. The data is sent one bit at a time and the receiver converts it to parallel data (i.e. to one full byte). The UART has several parameters which include: **Baud Rate**, **Number of data bits**, **Parity bit**, **Stop bit**. For the BASYS3 board we require a **Baud Rate** of **9600** bits per second. The **Number of data bits** corresponds to number of bits that constitute one whole byte - this is set usually to 8 bits (*some people keep this at 7 bits to accommodate the parity bit*). In this game implementation the **Parity bit** has been kept off. The **Stop bits** are used to determine the end of one complete byte transmission - it can either be set to **0**, **1** or **2**.

The **Micro USB connector J4** of the BASYS3 board allows us to use PC applications to communicate with the FPGA in question using standard Windows **COM** (Communication) ports. We will be using one of the two on-board status LEDs of the BASYS3 board that is, **LD 17** (*a.k.a* Receive LED) - which lights up when the BASYS3 board receives data through the UART.

The UART receiver module that we have written will consist of two inputs - the clock signal (**Clk**) and the serial data which the computer receives (**data**) - and one output - the output LEDs (**led**). This module activates at every negative edge of the clock signal. It inputs data into a register (**data\_curr** - which holds the current data) and it also sets two flags one for the **parity bit** and one for the **ending bit**. Then it prints the obtained data onto the LEDs at every positive edge of the flag and then in the end the data is sent to the **data\_pre** register.

## Code Listing for the Input Block

### Design Code

```
module keyboard (clk, data, d1, a1);
    input  clk; // Clock pin from keyboard
    input  data; // Data pin from keyboard
    output d1, a1;
    reg d; // checker1
    reg a; // checker2
    reg [7:0] data_curr;
    reg [7:0] data_pre;
    reg [3:0] b;
    reg flag;
    initial
        begin
```

```

d <= 0;
a<=0;
b<=4'h1;
flag <=1'b0;
data_curr <=8'hf0;
data_pre <=8'hf0;
end
always @(negedge clk) // Activating at negative edge of clock from keyboard
begin
    case(b)
        1:; // first bit
        2: data_curr[0] <= data;
        3: data_curr[1] <= data;
        4: data_curr[2] <= data;
        5: data_curr[3] <= data;
        6: data_curr[4] <= data;
        7: data_curr[5] <= data;
        8: data_curr[6] <= data;
        9: data_curr[7] <= data;
        10: flag <= 1'b1; // Parity bit
        11: flag <= 1'b0; // Ending bit
    endcase
    if (b <= 10)
        b <= b + 1;
    else if (b == 11)
        b <= 1;
    end
always @(posedge flag) // Printing data obtained to led
begin
    if (data_curr == 8'hf0)
    begin
        if (data_pre[0] == 0 && data_pre[1] == 0 && data_pre[2] == 1 &&
            data_pre[3] == 1 && data_pre[4] == 1 && data_pre[5] == 0 && data_pre[6]
            == 0 && data_pre[7] == 0 )

```

```

        d <= 1;
    else
        d <= 0;
        if ( data_pre[0] == 0 && data_pre[1] == 0 && data_pre[2] == 1 &&
            data_pre[3] == 1 && data_pre[4] == 1 && data_pre[5] == 0 && data_pre[6]
            == 0 && data_pre[7] == 0)
            a <= 1;
        else
            a <= 0;
        data_pre<=data_curr;
    end
else
    data_pre<=data_curr;
end
assign d1= d;
assign a1=a;
endmodule

```

## Testbench

```

module TB;
    reg clk , data;
    wire d,a;

    keyboard c1 (clk ,data ,d,a);

    initial
    begin
        clk = 1'b0;
        data = 1'b0;
    end

    always

    begin
        #5 clk =~clk;
    end

```

```

        data =~data;
    end

    initial
    begin
        $dumpfile ("dump.vcd ");
        $dumpvars(1,TB);
        $monitor(" clk=%1b, data=%1b, d=%1b,a=%1b \n", clk ,data ,d,a);
        #100 $finish;
    end
endmodule

```

## Timing Diagram

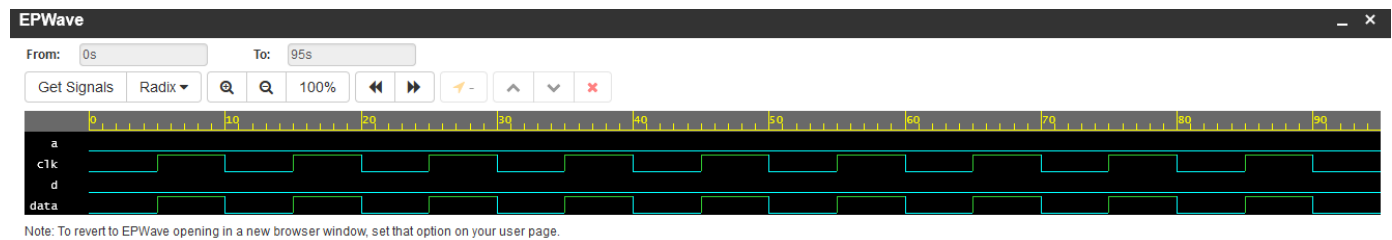


Figure 2: EP Wave for the Input Block.

## Log Window of Results

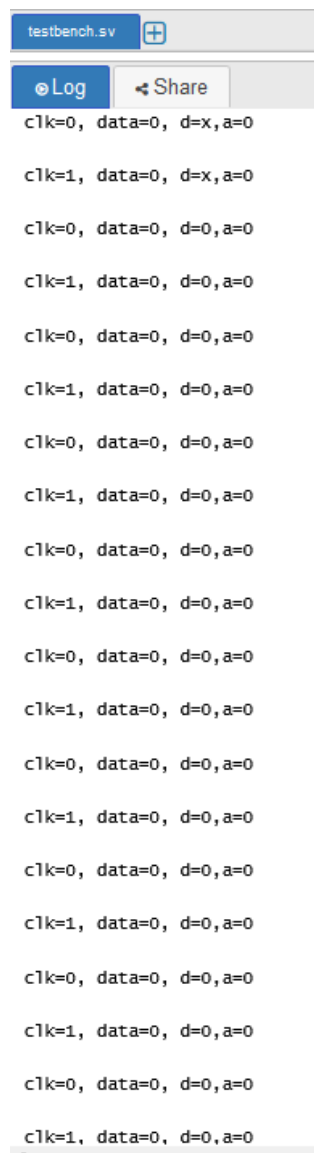
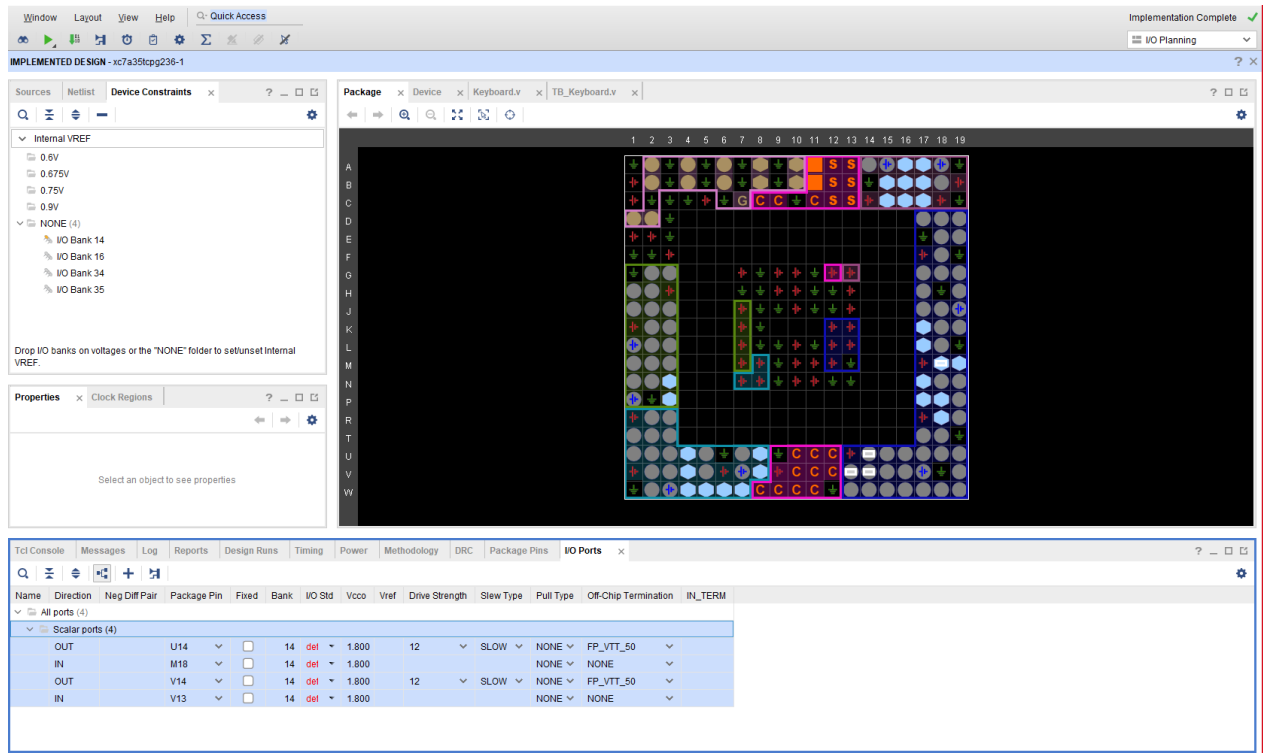


Figure 3: Log window of results for the Input Block.

## Vivado I/O Window



The screenshot shows the Vivado I/O Window with the IO Planning tab selected. The top panel displays the IO Pin Map, a grid of pins with various colored arrows indicating signal direction and strength. The bottom panel shows the IO Ports table, which lists the pins and their configurations.

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	IO Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (4)													
Scalar ports (4)													
OUT			U14		14	del	1.800	12	SLOW	NONE	FP_VTT_50		
IN			M18		14	del	1.800				NONE		
OUT			V14		14	del	1.800	12	SLOW	NONE	FP_VTT_50		
IN			V13		14	del	1.800				NONE		

Figure 4: This figure shows that the module synthesized in the Vivado software without any errors.



## Vivado Simulation Window

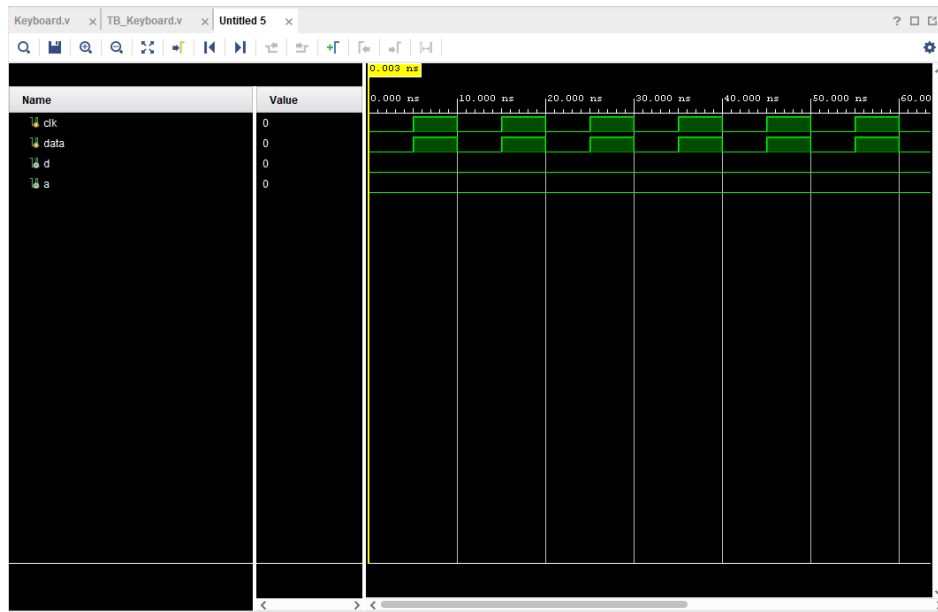


Figure 5: Ports for the constraint file for the Input Block.

## Vivado I/O Ports

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (4)													
Scalar ports (4)													
OUT			U14	<input type="checkbox"/>	14	del	1.800	12		SLOW	NONE	FP_VTT_50	
IN			M18	<input type="checkbox"/>	14	del	1.800				NONE	NONE	
OUT			V14	<input type="checkbox"/>	14	del	1.800	12		SLOW	NONE	FP_VTT_50	
IN			V13	<input type="checkbox"/>	14	del	1.800				NONE	NONE	

Figure 6: This figure shows that the module simulated in the Vivado software without any errors.

## The Control Block

### Major FSM

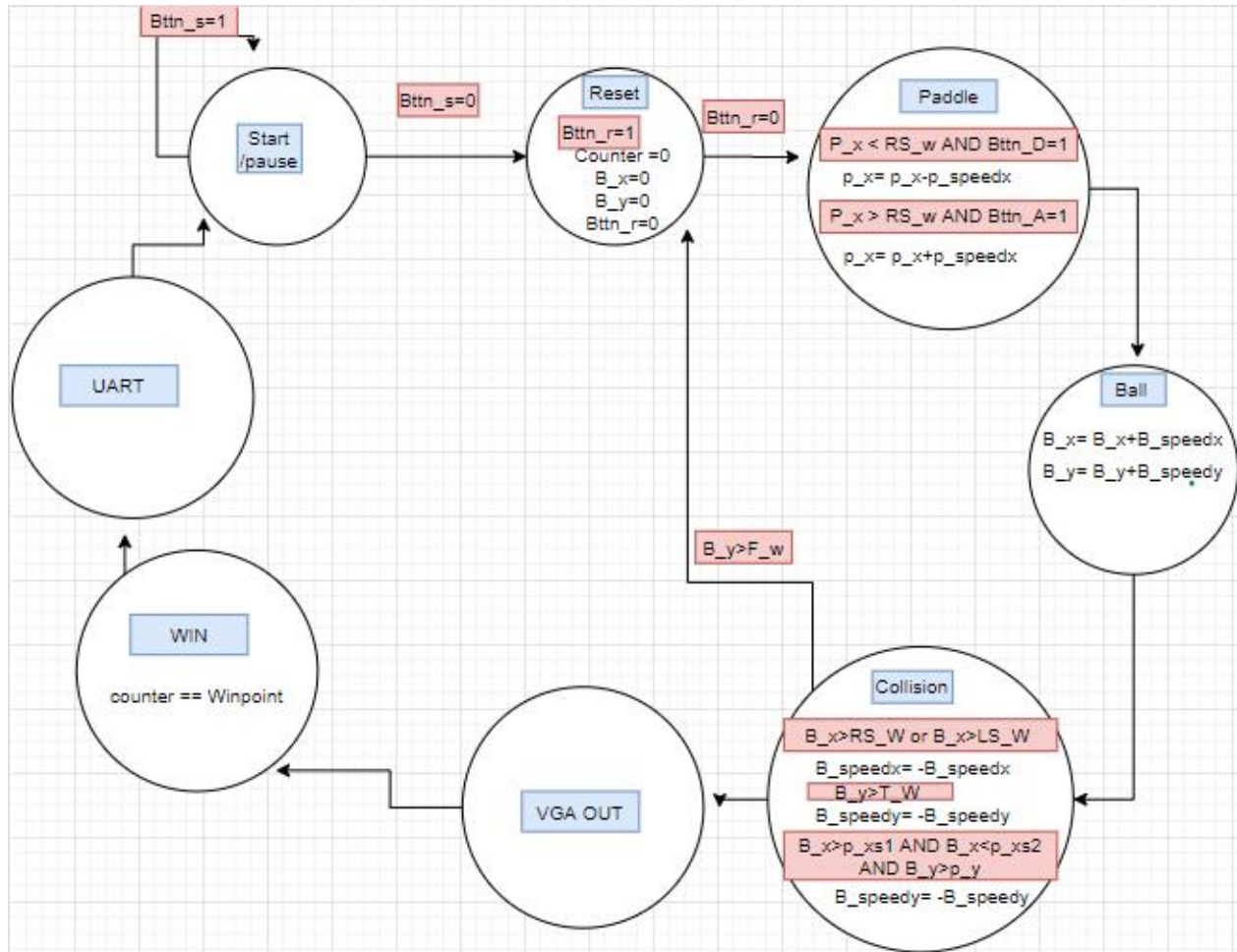


Figure 7: State transition diagram of the major FSM

The FSM implemented is a **Mealy Machine** as will be apparent by the descriptions given below.

**For the Top Level module code listing refer to page 38.**

As seen in Figure 7, each state represents a module that is design to do a specific task.

#### Start/Pause:

This takes input **btn\_s**. At the start this variable will be one therefore the state would not change until someone presses a key on the key board that is mapped to this specific variable whereby changing the variable to zero. At

this point the game would go to the next state. This is also used as a pause button since if pressed during game the variable will again be one and it would loop in this state.

#### Reset:

This takes input `btn_r`. If this is high at any point in the game the game would reset, meaning the counter value and the positions of the ball would reset. This also changes its own variable to zero hence going in to the next state. In the collision module if the ball touches the floor, `btn_r = 1`. Which means that in the next turn when it reaches the reset state, the game would reset.

#### Ball:

In this state we update the location of the ball in variables. `B_x` and `B_y` are the x and y coordinates respectively. We are adding `B_speedx` and `B_speedy` which are the distances that the ball travels in one cycle. These variables change in polarity when a collision occurs. This is why as if the collision occurs on the top wall the y coordinates start to decrease again as the ball has changed direction, thus the speed would be negative. Figure 2 shows this principle.

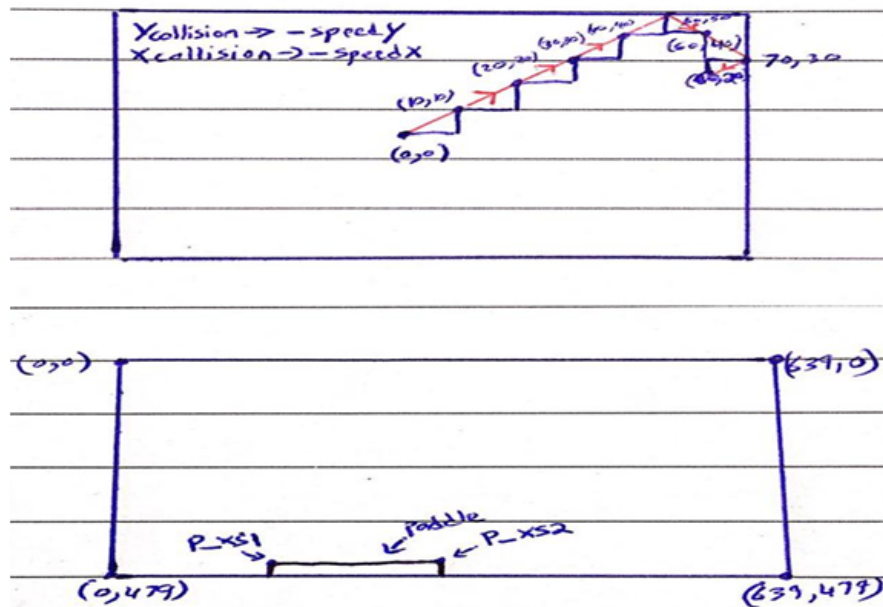


Figure 8: Ball movement and paddle variable

#### Collision:

In this state we detect if the ball has collided with any walls or the paddle. We can find that out by comparing the x and y values of the ball with the x or y values of the wall. As we can see in figure one the `RS_w` (right side wall)

and LS\_w (left side wall) are variable containing the x coordinates of the wall. These are compared with the x coordinates of the ball. If they exceed this, we will consider it as a collision and change the speed by taking twos complement of the speed variable (keeping in mind that we are considering it in base two.) thus when we add the speed now the speed would decrease the previous values hence the ball would move away. Same principle is applied to the T\_w (Top wall). For the F\_w (floor wall) we change the variable Btt\_n\_r =1 hence the ball has touch the floor and the game is over. For the paddle collision we see both the x values and the y values of the paddle if they coincide then we change the y speed.

#### Paddle:

This takes in two inputs Btt\_n\_A and Btt\_n\_D. these both are mapped on the keyboard, hence when pressed the variable would be one. For the paddle we have to see if the P\_xs1 (paddle\_x1) is greater than the left side wall x coordinate and P\_sx2 is less than the X coordinate of the right side wall. Thus the paddle would be contained within the screen.

#### VGA out:

The input for this module are the location of the paddle and the ball. Since we will have the specific pixel where these two are. The VGA driver will be able to mark these places at every cycle thus appearing as if the ball and the paddle are moving.

#### Win:

In this state the counter values is compared if it reaches a specific value the game would display a message that shows that you have won and then it will reset.

#### UART Input:

The keyboard is interfaced by using UART protocol. The keys are mapped with the variable Btt\_n\_s, Btt\_n\_A, Btt\_n\_D, Btt\_n\_r. when the buttons are pressed these variable change to one or zero.

#### Variables used:

##### 1. Button variables:

- (a) Btt\_n\_A= maps to 'a'
- (b) Btt\_n\_D= maps to 'd'
- (c) Btt\_n\_s= maps to 's'
- (d) Btt\_n\_r= maps to 'r'

##### 2. Var:

- (a) B\_x= ball center x location
- (b) B\_y= ball center y location
- (c) B\_speedx= speed x axis

- (d) B\_speedy= speed y axis
- (e) P\_xs1= border pixel of the paddle x location (see figure 2 for details)
- (f) P\_xs2= border pixel of the paddle x location (see figure 2 for details)
- (g) P\_x= paddle x coordinate from center.

3. Fixed variable:

- (a) RS\_W= right side wall x coordinate
- (b) LS\_W= left side wall x coordinate
- (c) T\_W= top wall y coordinate
- (d) F\_W= floor wall y coordinate
- (e) P\_Y= paddle y coordinate from center
- (f) P\_length= paddle length

## Minor FSM

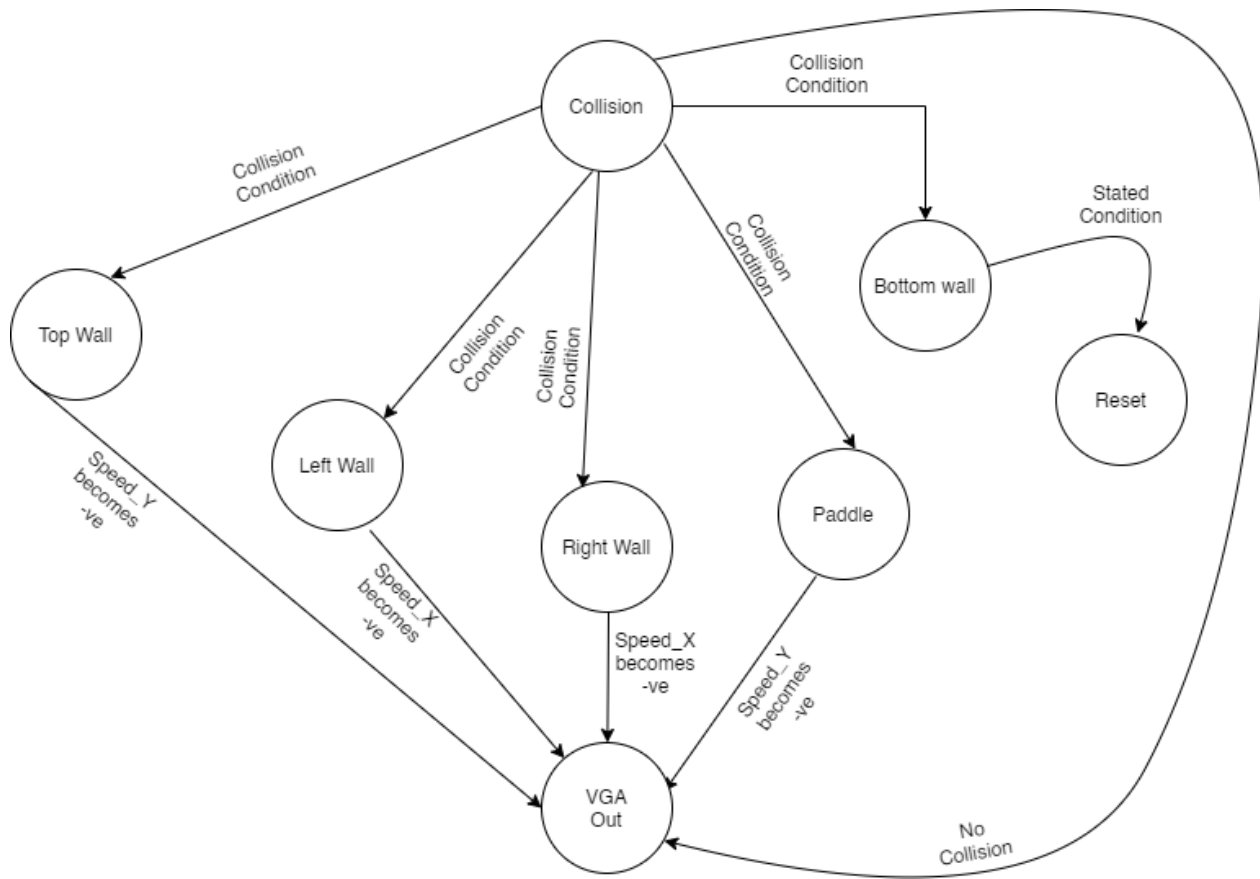


Figure 9: State transition diagram of the minor FSM

This Minor FSM, as shown in Figure 9, pertains to the **Collision** state of the Major FSM. Depending upon the collision surface - either the **paddle** or the **walls** - the velocity of the **ball** will change accordingly.

## Code Listing for the Collision Module of the Control Block

### Design Code

```
module collision (B_x,B_y,B_speedx,B_speedy,P_x,speedx,speedy,r,score);
    input [9:0] B_x, B_y, P_x;
    input [4:0] B_speedx,B_speedy;
    output [4:0] speedx,speedy;
```

```

output r,score;

parameter
paddle_sizex= 10'd5,
paddle_sizey=10'd1,
P_y = 10'd453;

assign speedx = (B_x < 10'd10) ? -B_speedx : (B_x > 633) ? -B_speedx:B_speedx;
assign speedy = (B_y < 10'b1010) ? -B_speedx: B_speedy;
assign r = (B_y > 10'b111001111) ? 1:0;
assign score = (B_x< (P_x+paddle_sizex)) ? (B_x> (P_x-paddle_sizex)) ?(B_y< (P_y+paddle_
sizey)) ? (B_y> (P_y-paddle_sizey)) ? 1:0:0 :0 :0 ;
endmodule

```

## Testbench

```

module TB;
reg [9:0] B_x,B_y,P_x;
reg [4:0]B_speedx,B_speedy;
wire [4:0] speedx,speedy;
wire r, score;

initial
begin
B_x = -10'd10;
B_y= 10'd10;
P_x = 10'd100;
B_speedx= 4'd10;
B_speedy= 4'd10;
#10
B_x = 10'd10;
B_y= 10'd9;
P_x = 10'd100;
B_speedx= 4'd10;

```

```

        B_speedy= 4'd10;
        #10
        B_x = 10'd100;
        B_y= 10'd453;
        P_x = 10'd100;
        B_speedx= 4'd10;
        B_speedy= 4'd10;
        #10
        B_x = 10'd100;
        B_y= 10'd479;
        P_x = 10'd100;
        B_speedx= 4'd10;
        B_speedy= 4'd10;
    end

collision_l1(B_x,B_y,B_speedx,B_speedy,P_x,speedx,speedy,r,score);

// Response
initial
begin
    $dumpfile("testResults.vcd");
    $dumpvars(1,TB);
    $monitor("B_x=%9b, B_y=%9d, B_speedx=%1d,B_speedy=%1d
    ,P_x=%9d,speedx=%1d,speedy=%1d,r=%1d,score=%1d \n",
    B_x,B_y,B_speedx,B_speedy,P_x,speedx,speedy,r,score);
end
endmodule

```



## Timing Diagram

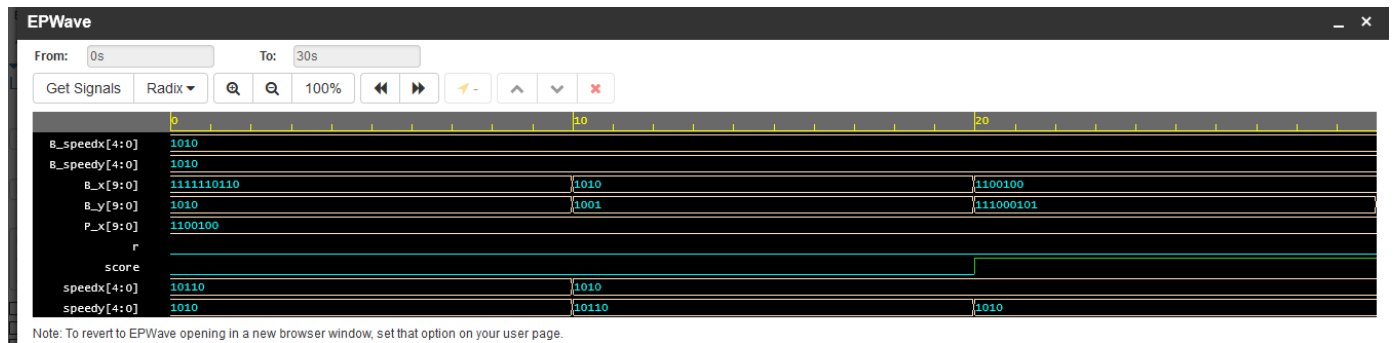
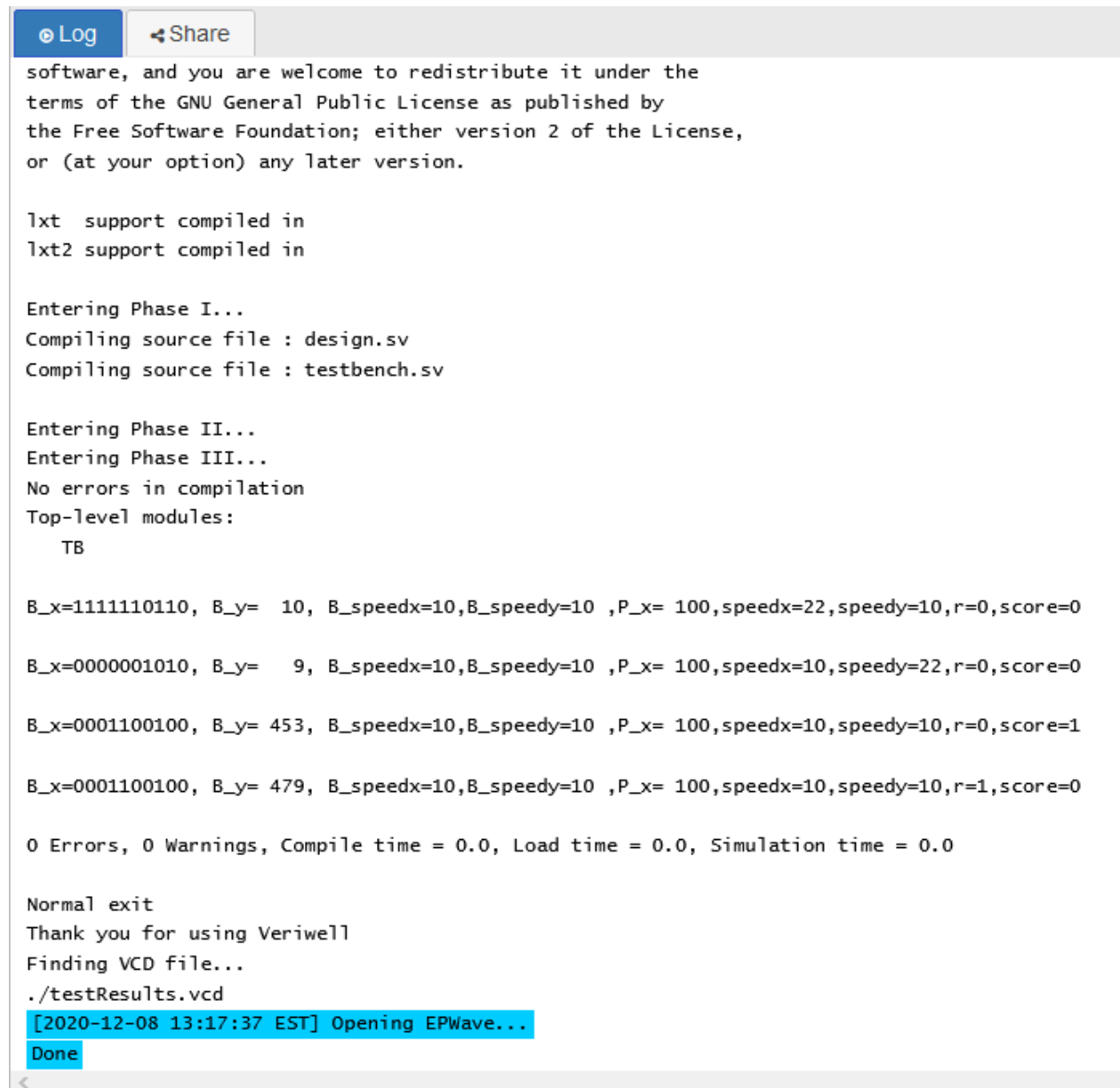


Figure 10: EP Wave for the Collision Detection Module of the Control Block.

## Log Window of Results



```
software, and you are welcome to redistribute it under the
terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License,
or (at your option) any later version.

lxt support compiled in
lxt2 support compiled in

Entering Phase I...
Compiling source file : design.sv
Compiling source file : testbench.sv

Entering Phase II...
Entering Phase III...
No errors in compilation
Top-level modules:
    TB

B_x=1111110110, B_y= 10, B_speedx=10,B_speedy=10 ,P_x= 100,speedx=22,speedy=10,r=0,score=0

B_x=0000001010, B_y=  9, B_speedx=10,B_speedy=10 ,P_x= 100,speedx=10,speedy=22,r=0,score=0

B_x=0001100100, B_y= 453, B_speedx=10,B_speedy=10 ,P_x= 100,speedx=10,speedy=10,r=0,score=1

B_x=0001100100, B_y= 479, B_speedx=10,B_speedy=10 ,P_x= 100,speedx=10,speedy=10,r=1,score=0

0 Errors, 0 Warnings, Compile time = 0.0, Load time = 0.0, Simulation time = 0.0

Normal exit
Thank you for using Veriwell
Finding VCD file...
./testResults.vcd
[2020-12-08 13:17:37 EST] Opening EPWave...
Done
```

Figure 11: Log window of results for the Collision Detection Module of the Control Block.

## Vivado I/O Window

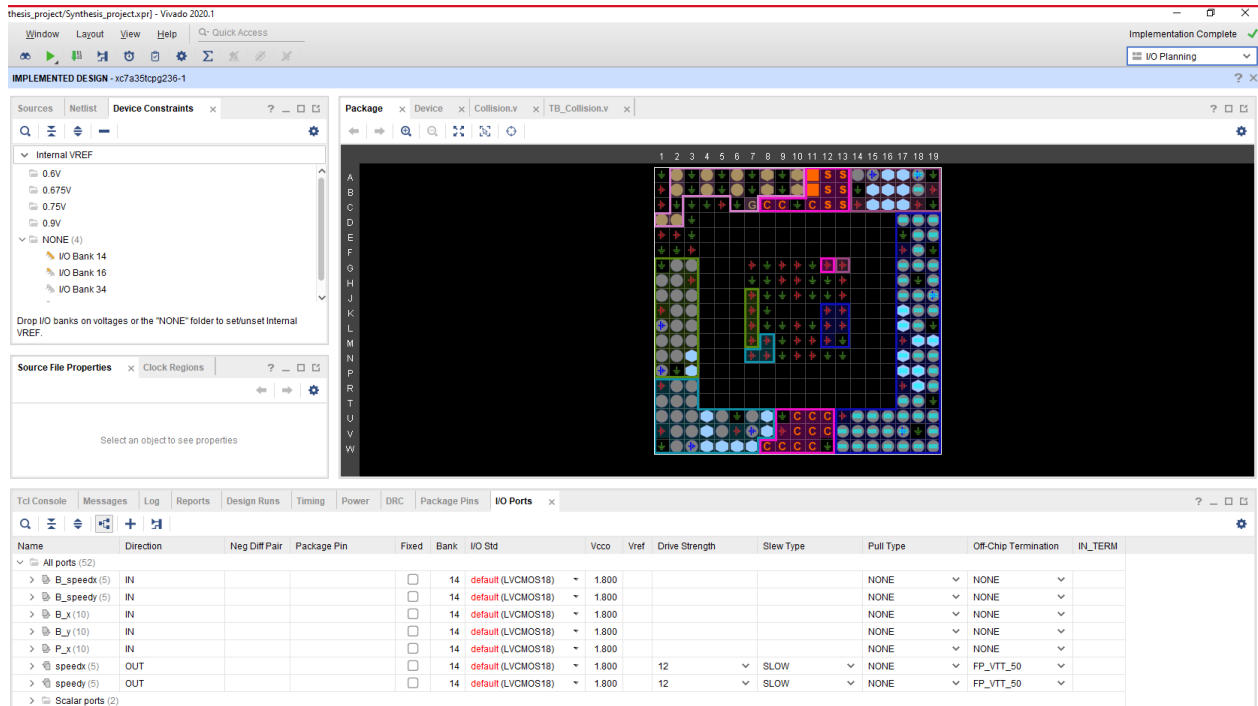


Figure 12: This shows that the module synthesized within the Vivado software without any errors.

## Vivado Simulation Window

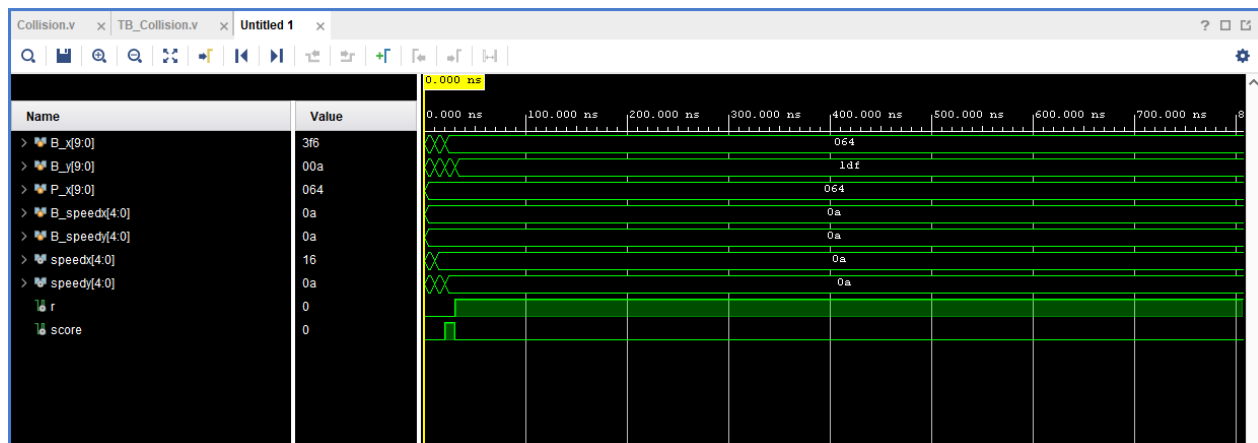


Figure 13: This shows that the module simulated within the Vivado software without any errors.

## Vivado I/O Ports

Tcl Console Messages Log Reports Design Runs Timing Power DRC Package Pins <b>IO Ports</b> x													
Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	IO Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
▼ All ports (52)													
> B_speedx (5)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	▼ NONE	▼
> B_speedy (5)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	▼ NONE	▼
> B_x (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	▼ NONE	▼
> B_y (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	▼ NONE	▼
> B_x (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	▼ NONE	▼
> speedx (5)	OUT			<input type="checkbox"/>	14	default (LVCMOS18)	1.800	12	▼ SLOW	▼	NONE	▼ FP_VTT_50	▼
> speedy (5)	OUT			<input type="checkbox"/>	14	default (LVCMOS18)	1.800	12	▼ SLOW	▼	NONE	▼ FP_VTT_50	▼
▼ Scalar ports (2)													

Figure 14: Ports for the constraint file for the Collision Detection Module.

## Code Listing for the Scoring Module of the Control Block

### Design Code

```
// Code your design here
`timescale 1ns / 1ps

// Code your design here
module clk_div (clk,clk_d);
    parameter div_value = 1;
    input clk;
    output clk_d;
    reg clk_d;
    reg count;
    initial
    begin
        clk_d = 0;
        count = 0;
    end
    always @(posedge clk)
    begin
        if (count == div_value)
            count <=0;
        else
```

```

        count <= count + 1;
    end

    always @(posedge clk)
    begin
        if (count==div_value)
            clk_d <= ~ clk_d;
        end
    endmodule

```

```

module scoring(score,prev_score,score_out,clk);
    input clk;
    input score;
    input [9:0] prev_score;
    output [9:0] score_out;

    assign score_out = prev_score + score;

endmodule

```

## Testbench

```

// Code your testbench here
// or browse Examples
`timescale 1ns/1ps
module testbench();
    reg s;
    reg [9:0] ps;
    wire [9:0] so;
    reg clk;

    //initial begin
    //end

```

```

initial
begin
    clk = 1'b0;
    s=1; ps=5;
end
scoring test1t1 (.score(s),.prev_score(ps),.score_out(so),.clk(clk));
always
    #5 clk =~clk;
initial begin
    $dumpfile("dump.vcd");
    $dumpvars(1,testbench);
    #10
    $display ("Score=%d, Previous_score=%d, Score_out=%d",s,ps,so);
end
endmodule

```

## Timing Diagram

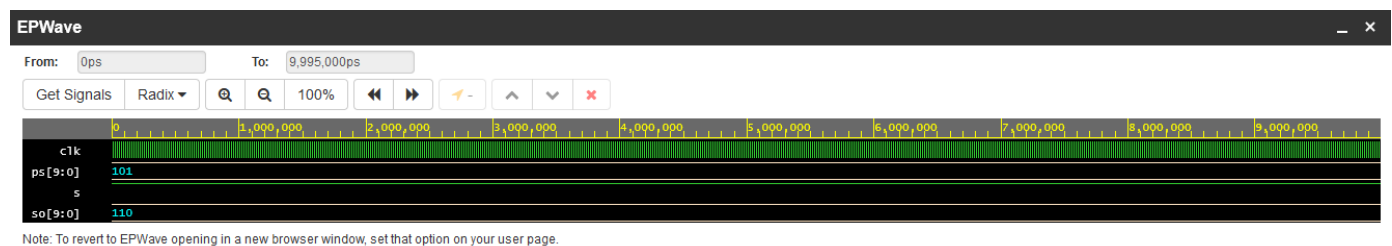
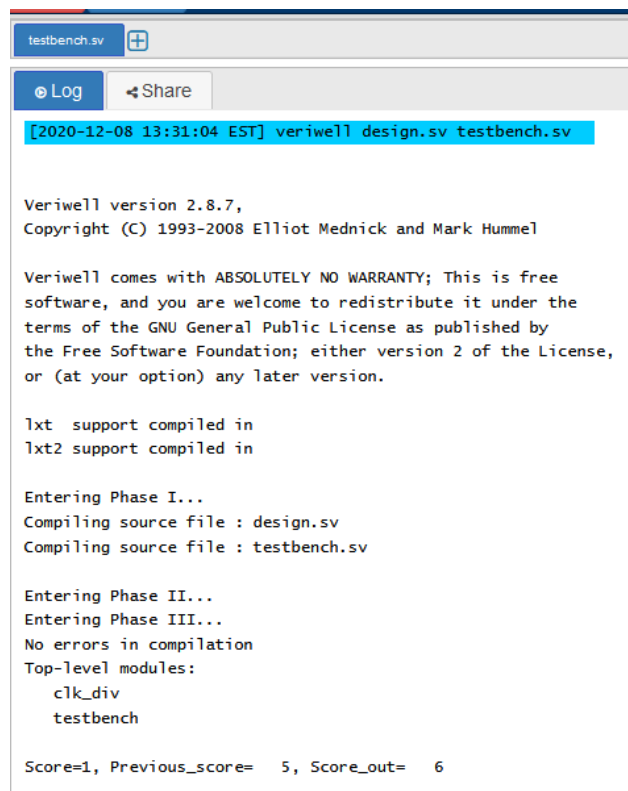


Figure 15: EP Wave for the Scoring Module of the Control Block.

## Log Window of Results



The screenshot shows a web-based interface for the Veriwell software. At the top, there is a header bar with a tab labeled 'testbench.sv' and a plus icon. Below the header, there are two buttons: 'Log' and 'Share'. The main content area displays a log window with the following text:

```
[2020-12-08 13:31:04 EST] veriwell design.sv testbench.sv

Veriwell version 2.8.7,
Copyright (C) 1993-2008 Elliot Mednick and Mark Hummel

Veriwell comes with ABSOLUTELY NO WARRANTY; This is free
software, and you are welcome to redistribute it under the
terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License,
or (at your option) any later version.

lxt support compiled in
lxt2 support compiled in

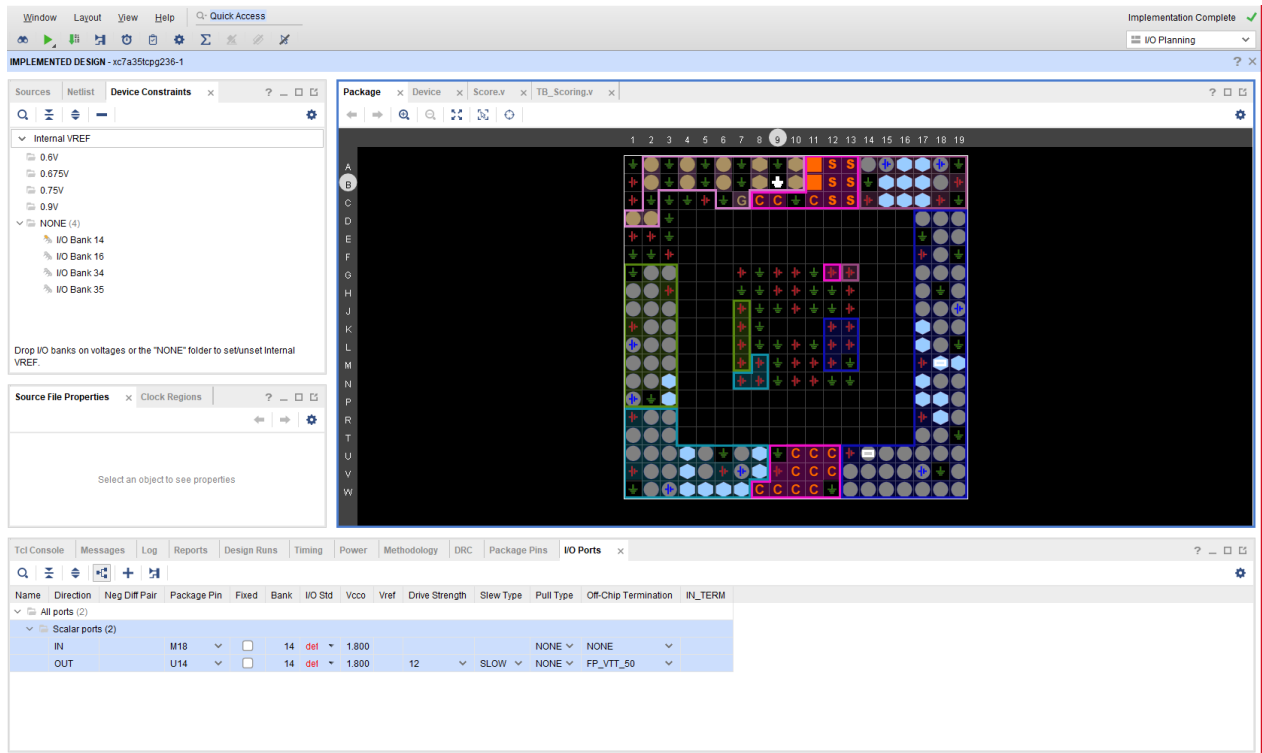
Entering Phase I...
Compiling source file : design.sv
Compiling source file : testbench.sv

Entering Phase II...
Entering Phase III...
No errors in compilation
Top-level modules:
  clk_div
  testbench

Score=1, Previous_score= 5, Score_out= 6
```

Figure 16: Log window of results for the Scoring Module of the Control Block.

## Vivado I/O Window



The screenshot shows the Vivado I/O Window with the 'IO Planning' tab selected. The window displays a grid of package pins with various I/O standards and signals assigned. The 'IO Ports' tab is active, showing a table of port configurations.

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	IO Std	Vcco	Vref	Drive Strength	Stew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (2)													
Scalar ports (2)													
IN			M18		14	del	1.800			NONE	NONE		
OUT			U14		14	del	1.800		12	SLOW	NONE	FP_VTT_50	

Figure 17: This shows that the module synthesized within the Vivado software without any errors.



## Vivado Simulation Window

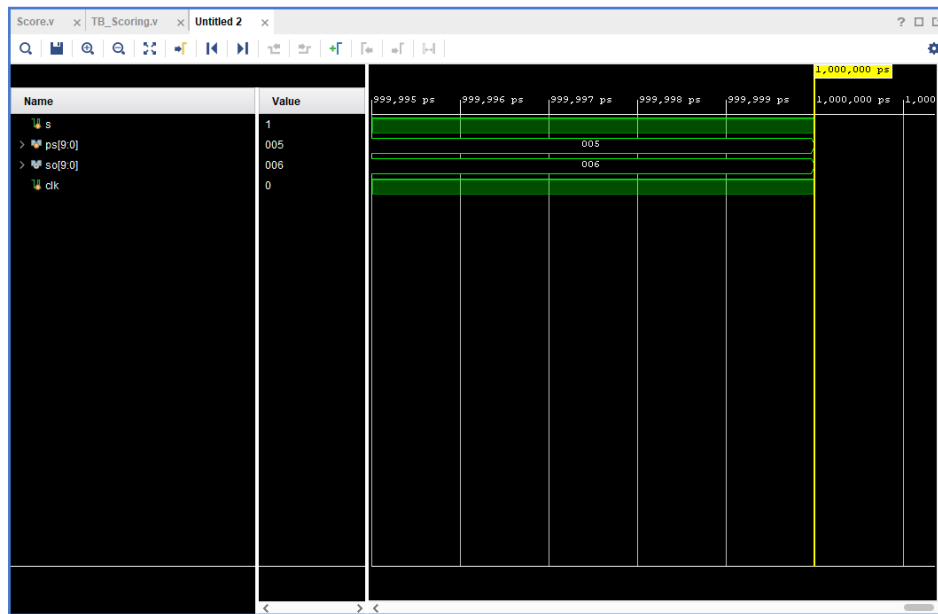


Figure 18: This shows that the module simulated within the Vivado software without any errors.

## Vivado I/O Ports

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (2)													
Scalar ports (2)													
IN			M18	<input type="checkbox"/>	14	del	1.800			NONE	NONE		
OUT			U14	<input type="checkbox"/>	14	del	1.800	12	SLOW	NONE	FP_VTT_50		

Figure 19: Ports for the constraint file for the Scoring Module.

## The Output Block

For this we made a total of six modules, these are the following:

1. `h_counter`
2. `v_counter`
3. `clk_div`
4. `vga_sync`
5. `pixel_gen`
6. `toplevel`

The electron beam which is used to drive the output to the VGA monitor moves using two synchronization signals called **h\_sync** and **v\_sync**. These signals are included in the **vga\_sync** module. The **h\_sync** signal is obtained by the **h\_counter** module and the **v\_sync** signal is obtained from the **v\_counter** module. The **vga\_sync** module also takes another parameter called **video\_on** which indicates whether the current targeted pixel is in the displayable region or not. The **clk\_div** module is used to obtain a clock signal of a certain frequency as the frequency to drive the VGA monitor is different to that of a BASYS3 board. The **pixel\_gen** module generates three four-bit colour signals which are known as the **RGB signal**. Furthermore, this module outputs a colour value according to the current co-ordinates of the pixel (**x\_loc** and **y\_loc**). The **topLevel** module combines all of these modules to give a final output. In the **pixel\_gen** module we also input the current co-ordinates of the ball and paddle and after setting the required pixel size for both the ball and paddle we can display them on the screen according to their location. This can be seen in the frames shown below:

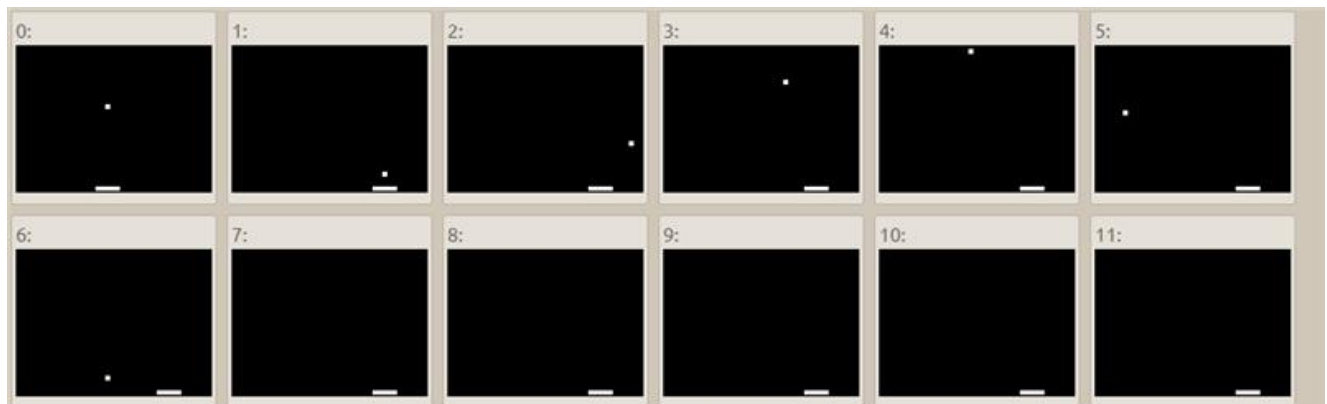


Figure 20: 12 frames of the game where different co-ordinates were input for both the ball and the paddles. It also depicts collision as well.

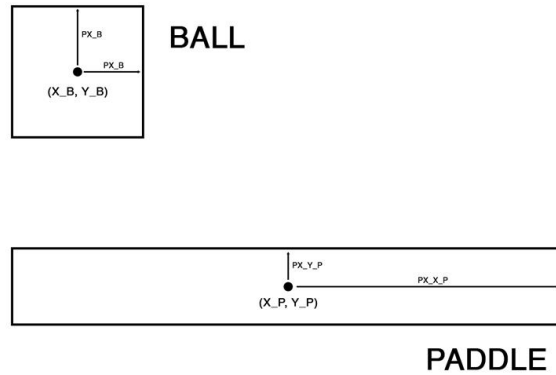


Figure 21: Implementation technique for pixel generation.

The implementation as shown in Figure 21, was done along the following line: The center co-ordinate is chosen which will move across the screen using the **Movement** module and the **paddle** input module. Afterwards, using certain pre-defined pixel measurements as shown in Figure 21, the pixels for the ball and paddle will be generated.

The co-ordinates for the ball will be received through the ball movement module/implementation and the coordinates for the paddle will be received from the input module which will change according to the User's input.

### Time taken to generate one frame

Clock Frequency = 25 MHz

Time period = 40 ns

time taken by program= 300000000 ns

Cycles =  $\frac{300000000}{40} = 7500000$  cycles

Total frames= 16

clk delay= 5 ns

Time takes with clk delay =  $(300000000 - (7500000 * 5)) = 262500000$  ns

Time for one frame =  $\frac{262500000}{16} = 16406250$  ns

## Code Listing for the Output Block

**Note:** The following code includes both the VGA and the movement functionality - said functionality has been integrated successfully.

### Design Code

```
`timescale 1ns / 1ps

module h_counter( clk , h_count , trig_v );
    input  clk;
    output trig_v;
    reg trig_v;
    output [9:0] h_count;
    reg [9:0] h_count;
    initial
        begin
            h_count <= 0;
            trig_v <= 0;
        end
    always @(posedge clk)
        begin
            if (h_count < 799)
                begin
                    h_count <= h_count + 1;
                    trig_v <= 0;
                end
            else
                begin
                    h_count <= 0;
                    trig_v <= 1;
                end
        end
endmodule
```

//-----

```

// Code your design here
module v_counter(clk ,v_count ,enable_v );
    input clk;

    input enable_v;
    //wire enable_v;

    output [9:0] v_count;
    reg [9:0] v_count;

    initial
        begin
            v_count <= 0;
        end

    always @ (posedge clk)
        begin
            if (enable_v == 1)
                begin
                    if (v_count < 524)
                        begin
                            v_count <= v_count + 1;
                        end
                    else
                        begin
                            v_count <= 0;
                        end
                end
            else
                begin
                    v_count <= v_count+0;
                end
        end
end

```

```
endmodule
```

```
//-----
```

```
// Code your design here
```

```
module clk_div (clk,clk_d);
```

```
    parameter div_value = 1;
```

```
    input clk;
```

```
    output clk_d;
```

```
    reg clk_d;
```

```
    reg count;
```

```
    initial
```

```
        begin
```

```
            clk_d = 0;
```

```
            count = 0;
```

```
        end
```

```
    always @(posedge clk)
```

```
        begin
```

```
            if (count == div_value)
```

```
                count <=0;
```

```
            else
```

```
                count <=count + 1;
```

```
        end
```

```
    always @(posedge clk)
```

```
        begin
```

```
            if (count==div_value)
```

```
                clk_d <= ~ clk_d;
```

```
        end
```

```
endmodule
```

```
//-----
```

```

`timescale 1ns / 1ps
module vga_sync (
    input [9:0] h_count,
    input [9:0] v_count,
    output h_sync,
    output v_sync,
    output video_on,
    output [9:0] x_loc,
    output [9:0] y_loc);

    localparam HD = 640;
    localparam HF = 16;
    localparam HB = 48;
    localparam HR = 96;

    localparam VD = 480;
    localparam VF = 10;
    localparam VB = 33;
    localparam VR = 2;

    assign x_loc = h_count;
    assign y_loc = v_count;

    assign h_sync = (h_count<(656)) || (h_count>752);

    assign v_sync = (v_count<490) || (v_count>492);

    assign video_on = (h_count<640) || (v_count<480);

endmodule

//-----

```

```

module pixel_gen(
    input clk_d ,
    input [9:0] pixel_x ,
    input [9:0] pixel_y ,
    input video_on ,
    output reg [3:0] red=0,
    output reg [3:0] green=0,
    output reg [3:0] blue=0,
    input [9:0] xcoordball ,
    input [9:0] ycoordball ,
    input [9:0] xcoordpad
);
    reg ball ;
    reg paddle1 ;

    always @(posedge clk_d) begin
        if (pixel_x>640) begin
            red <= 4'hF;
            green <= 4'hF;
            blue <= 4'hF;
        end
        else
            begin
                assign ball = ((pixel_x >=(xcoordball-8) && pixel_x <=(xcoordball+8))&&(pixel_y >=
                    (ycoordball-8) && pixel_y <=(ycoordball+8))); // coordinates and +- is pixel size

                assign paddle1 = ((pixel_x >=(xcoordpad-40)&&pixel_x <=(xcoordpad+40))&&(pixel_y >=462
                    && pixel_y <=474));

                red <= video_on ? (((ball || paddle1) ? 4'hF : 4'h0)) : (4'h0);
                green <= video_on ? (((ball || paddle1) ? 4'hF : 4'h0)) : (4'h0);
                blue <= video_on ? (((ball || paddle1) ? 4'hF : 4'h0)) : (4'h0);
            end
        end
    end

```



```

        end
    end
endmodule

```

```

//-----

```

```

module toplevel(clk,clk_D, h_sync, v_sync, R, G, B, xcoordball, ycoordball, xcoordpad);
    input clk;
    output h_sync;
    output v_sync;
    wire [9:0] HC;
    wire [9:0] VC;
    output [3:0] R;
    output [3:0] G;
    output [3:0] B;
    output clk_D;
    wire tgv;
    wire v_on;
    wire [9:0] xloc;
    wire [9:0] yloc;
    input [9:0] xcoordball;
    input [9:0] ycoordball;
    input [9:0] xcoordpad;

    clk_div g0(.clk(clk), .clk_d(clk_D));

    h_counter a1(.clk(clk_D), .h_count(HC), .trig_v(tgv));
    v_counter a2(.enable_v(tgv), .clk(clk_D), .v_count(VC));

```

```

vga_sync a3(.h_count(HC), .v_count(VC), .h_sync(h_sync), .v_sync(v_sync), .video_on(v_on),
.x_loc(xloc), .y_loc(yloc));

pixel_gen a4(.clk_d(clk_D), .pixel_x(xloc), .pixel_y(yloc), .video_on(v_on), .red(R),
.green(G), .blue(B),.xcoordball(xcoordball),.ycoordball(ycoordball),.xcoordpad(xcoordpad));

endmodule

```

## Testbench

```

`timescale 1ns/1ps
module testbench_vga();

    reg clk =0;
    reg [15:0] simtime = 16'd5;

    wire h_sync;
    wire v_sync;

    wire clk_D;

    wire [3:0] R;
    wire [3:0] G;
    wire [3:0] B;

    reg [9:0] xcoordball;
    reg [9:0] ycoordball;
    reg [9:0] xcoordpad;

    //coordinates are changed with input
    initial begin
        #16410000
        xcoordball = 300;
        ycoordball = 200;
    end
endmodule

```

```

xcoordpad = 300;
#16410000
xcoordball = xcoordball+200;
ycoordball = ycoordball+220;
xcoordpad = xcoordpad +200;
    #16410000
xcoordball = xcoordball+100;
ycoordball = ycoordball-100;
#16410000
xcoordball = xcoordball-200;
ycoordball = ycoordball-200;
#16410000
xcoordball = xcoordball-100;
ycoordball = ycoordball-100;
#16410000
xcoordball = xcoordball-200;
ycoordball = ycoordball+200;
#16410000
xcoordball = xcoordball+200;
ycoordball = ycoordball+200;
    #16410000
xcoordball = xcoordball+100;
ycoordball = ycoordball+100;
end
integer file_ID;
toplevel test1t1 (clk,clk_D, h_sync, v_sync, R, G, B, xcoordball, ycoordball, xcoordpad);

initial
    #10 file_ID = $fopen("output_log_file.txt", "w");
always
    #5 clk =~clk;
initial begin
    // $dumpfile("dump.vcd");
    // $dumpvars(1, testbench_bga);

```

```

#3000000000 $fclose(file_ID);
#10 $finish;
end

always @(posedge clk_D)
    $fwrite(file_ID , "%05d ns: %b %b %b %b %b\n", $realtime , h_sync , v_sync , R, G, B);
endmodule

```

## Vivado I/O Window

The screenshot shows the Vivado I/O Window with the implementation complete status. The pin diagram displays the physical layout of the I/O pins on the device. The table below lists the I/O ports and their properties.

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
> B (4)	OUT			<input type="checkbox"/>	14	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	
> G (4)	OUT			<input type="checkbox"/>	14	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	
> R (4)	OUT			<input type="checkbox"/>	14	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	
> xcoordball (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	NONE	
> xcoordpad (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	NONE	
> ycoordball (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	NONE	
> ycoordpad (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	NONE	
> Scalar ports (4)													

Figure 22: This shows that the module synthesized within the Vivado software without any errors.

## Vivado Simulation Window

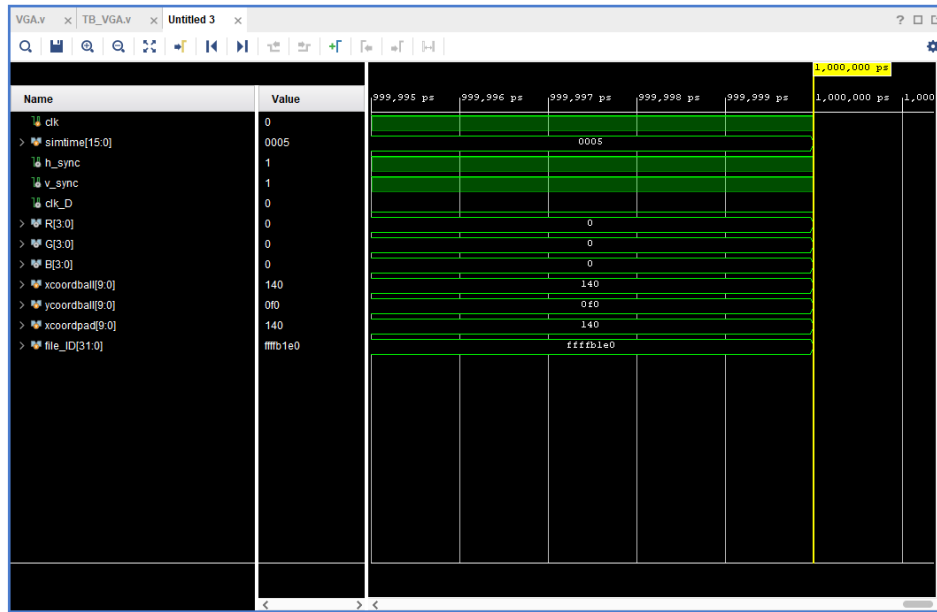


Figure 23: This shows that the module simulated within the Vivado software without any errors.

## Vivado I/O Ports

Tcl Console Messages Log Reports Design Runs Timing Power Methodology DRC Package Pins I/O Ports x													
Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (45)													
> B (4)	OUT			<input type="checkbox"/>	14	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	
> G (4)	OUT			<input type="checkbox"/>	14	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	
> R (4)	OUT			<input type="checkbox"/>	14	default (LVCMOS18)	1.800		12	SLOW	NONE	FP_VTT_50	
> xcoordball (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	NONE	
> xcoordpad (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	NONE	
> ycoordball (10)	IN			<input type="checkbox"/>	14	default (LVCMOS18)	1.800				NONE	NONE	
> Scalar ports (4)													

Figure 24: Ports for the constraint file for the Scoring Module.

# Conclusion and Results

## Milestone 1

With respect to the **Input Block**, we were able to verify that the input from the keyboard was indeed transmitted and received by the **BASYS3** board. This verification was done by a visual inspection of the LEDs on the **BASYS3** board. As far as further work is concerned with respect to the aforementioned block, we are yet to successfully integrate with the **toplevel** module (*yet to be made*) of the **Control Block**.

With regards to the **Control Block**, we successfully designed the **user-flow** and how it will integrate with the game logic. With regards to the Finite State Machine (**FSM**), we have decided to use the **Mealy Machine** as the FSM requires both the current input and the current state to transition to the next-state.

**It is important to note:** During the project demo, we were told to further clarify and simplify the proposed states for the FSM, on which significant progress has already been made.

With regards to the **Output Block**, we were able to display the static screen for our game on the VGA monitor thus, proving its working condition. However, the colours were slightly faded - we will continue to determine the origin of the problem, whether it is a fault with the block itself or with the VGA monitor.

Far from finished, greater focus has now been attained following the demo and we strive to achieve a timely and efficient and working conclusion to this enterprise.

## Milestone 2

In our **VGA** module we have successfully shown that the **ball** changes both its speed and direction when it collides with either the walls or the paddle. This can also be seen in Figure 20. Accordingly, if the ball successfully bounces off the paddle, the current score of the Player increments by an seed of 1 (*as shown in our **Scoring module***). However, if the Player is unsuccessful in making contact, the score is **reset** back to 0 and thus, **loses** the game. We have implemented our major state transitions using the **Major** FSM while the **Collision** state contains **Minor** FSM in order to simplify the functionality of the state transition system. In this way, we have used the **Major-Minor** FSM system.

The FSM is modeled as a **Circular FSM** - at every positive edge of clock cycle there will be a state transition. These transitions would occur rapidly thus, keeping the illusion of a simultaneous execution of sorts.

## Appendix

### Code Listing for Control Block Top Level Module

#### Design Code

```
module scoring(score,prev_score,score_out);
```

```
    input score;
```

```
    input [9:0] prev_score;
```

```
    output [9:0] score_out;
```

```
    assign score_out = prev_score + score;
```

```
endmodule
```

```
//-----
```

```
module keyboard(clk,data,d1,a1);
```

```
    input  clk;// Clock pin form keyboard
```

```
    input  data;//Data pin form keyboard
```

```
    output d1,a1;
```

```
    reg d; //checker1
```

```
    reg a; //checker2
```

```
    reg [7:0] data_curr;
```

```
    reg [7:0] data_pre;
```

```
    reg [3:0] b;
```

```
    reg flag;
```

```
    initial
```

```
        begin
```

```
            d <= 0;
```

```
            a<=0;
```

```

    b<=4'h1;
    flag <=1'b0;
    data_curr <=8'hf0;
    data_pre <=8'hf0;
end
always @(negedge clk)// Activating at negative edge of clock from keyboard
begin
    case(b)
        1:; // first bit
        2:data_curr[0]<=data;
        3:data_curr[1]<=data;
        4:data_curr[2]<=data;
        5:data_curr[3]<=data;
        6:data_curr[4]<=data;
        7:data_curr[5]<=data;
        8:data_curr[6]<=data;
        9:data_curr[7]<=data;
        10:flag <=1'b1; // Parity bit
        11:flag <=1'b0; // Ending bit
    endcase
    if (b<=10)
        b<=b+1;
    else if (b==11)
        b<=1;
    end
always@(posedge flag)// Printing data obtained to led
begin
    if (data_curr==8'hf0)
        begin
            if (data_pre[0] == 0 && data_pre[1] == 0 && data_pre[2] == 1 && data_pre[3] == 1
                && data_pre[4] == 1&& data_pre[5] == 0&& data_pre[6] == 0 && data_pre[7] == 0 )
                d <= 1;
            else
                d <= 0;
        end
    end

```



```

        if( data_pre[0] == 0 && data_pre[1] == 0 && data_pre[2] == 1 && data_pre[3] == 1
        && data_pre[4] == 1&& data_pre[5] == 0&& data_pre[6] == 0 && data_pre[7] == 0)
            a <= 1;
        else
            a <= 0;
        data_pre<=data_curr;
        end
    else
        data_pre<=data_curr;
    end
    assign d1= d;
    assign a1=a;

endmodule

```

//-----

```

module ballmove(B_x,B_y,b1_x,b1_y,speedx,speedy);
    input [9:0] B_x,B_y;
    input speedx,speedy;
    output [9:0] b1_x,b1_y;

    assign b1_x = B_x + speedx;
    assign b1_y = B_y + speedy;

endmodule

```

endmodule

//-----

```
module scoring(score,prev_score,score_out);
```

```
    input score;
```

```
    input [9:0] prev_score;
```

```
    output [9:0] score_out;
```

```
    assign score_out = prev_score + score;
```

```
endmodule
```

```
//-----
```

```
module collision(B_x,B_y,B_speedx,B_speedy,P_x,speedx,speedy,r,score);
```

```
    input [9:0] B_x, B_y, P_x;
```

```
    input [4:0] B_speedx,B_speedy;
```

```
    output [4:0] speedx,speedy;
```

```
    output r,score;
```

```
    parameter
```

```
    paddle_sizex= 10'd5,
```

```
    paddle_sizey=10'd1,
```

```
    P_y = 10'd453;
```

```
    assign speedx = (B_x < 10'd10) ? -B_speedx : (B_x > 633) ? -B_speedx:B_speedx;
```

```
    assign speedy = (B_y < 10'b1010) ? -B_speedx: B_speedy;
```

```
    assign r = (B_y > 10'b111001111) ? 1:0;
```

```
    assign score = (B_x< (P_x+paddle_sizex)) ? (B_x> (P_x-paddle_sizex)) ?(B_y<
```

```
(P_y+paddle_sizey)) ? (B_y> (P_y-paddle_sizey)) ? 1:0:0 :0 :0 ;
```

```
endmodule
```

```
//-----
```

```
`timescale 1ns / 1ps
```

```
module h_counter(clk,h_count,trig_v);
```

```
input clk;
```

```
output trig_v;
```

```
reg trig_v;
```

```
output [9:0] h_count;
```

```
reg [9:0] h_count;
```

```
initial
```

```
begin
```

```
h_count <= 0;
```

```
trig_v <= 0;
```

```
end
```

```
always @(posedge clk)
```

```
begin
```

```
if (h_count < 799)
```

```
begin
```

```
h_count <= h_count + 1;
```

```
trig_v <= 0;
```

```
end
```

```
else
```

```
begin
```

```
h_count <= 0;
```

```
trig_v <= 1;
```

```
end
```

```

        end
    endmodule

//-----

module v_counter( clk , v_count , enable_v );
    input  clk;

    input  enable_v;
    //wire enable_v;

    output [9:0] v_count;
    reg [9:0] v_count;

    initial
    begin
        v_count <= 0;
    end

    always @ (posedge clk)
    begin
        if (enable_v == 1)
        begin
            if (v_count < 524)
            begin
                v_count <= v_count + 1;
            end
            else
            begin
                v_count <= 0;
            end
        end
    else
        begin

```

```

        v_count <= v_count+0;
    end
end

```

```

endmodule

```

```

//-----

```

```

module clk_div (clk,clk_d);
    parameter div_value = 1;
    input clk;
    output clk_d;
    reg clk_d;
    reg count;
    initial
        begin
            clk_d = 0;
            count = 0;
        end
    always @(posedge clk)
        begin
            if (count == div_value)
                count <=0;
            else
                count <=count + 1;
            end

        always @(posedge clk)
            begin
                if (count==div_value)
                    clk_d <= ~ clk_d;
                end
    endmodule

```

```
//-----
```

```
'timescale 1ns / 1ps
module vga_sync (h_count,v_count,h_sync,v_sync,video_on, x_loc,y_loc);
    input [9:0] h_count;
    input [9:0] v_count;
    output h_sync;
    output v_sync;
    output video_on;
    output [9:0] x_loc;
    output [9:0] y_loc;
    initial
        begin
            localparam HD = 640;
            localparam HF = 16;
            localparam HB = 48;
            localparam HR = 96;

            localparam VD = 480;
            localparam VF = 10;
            localparam VB = 33;
            localparam VR = 2;
            end
        assign x_loc = h_count;
        assign y_loc = v_count;

        assign h_sync = (h_count<(656)) || (h_count>752);

        assign v_sync = (v_count<490) || (v_count>492);

        assign video_on = (h_count<640) || (v_count<480);
```

```
endmodule
```

```
//-----
```

```
module pixel_gen(  
    input clk_d ,  
    input [9:0] pixel_x ,  
    input [9:0] pixel_y ,  
    input video_on ,  
    output reg [3:0] red=0 ,  
    output reg [3:0] green=0 ,  
    output reg [3:0] blue=0 ,  
    input [9:0] xcoordball ,  
    input [9:0] ycoordball ,  
    input [9:0] xcoordpad  
);  
    reg ball ;  
    reg paddle1 ;  
  
    always @(posedge clk_d) begin  
        if (pixel_x>640) begin  
            red <= 4'hF ;  
            green <= 4'hF ;  
            blue <= 4'hF ;  
        end  
        else  
            begin  
                assign ball = ((pixel_x >=(xcoordball-8) && pixel_x <=(xcoordball+8))&&(pixel_y >=  
                    (ycoordball-8) && pixel_y <=(ycoordball+8))); // coordinates and +- is pixel size  
  
                assign paddle1 = ((pixel_x >=(xcoordpad-40)&&pixel_x <=(xcoordpad+40))&&(pixel_y >=462  
                    && pixel_y <=474));
```

```

        red <= video_on ? (((ball||paddle1) ? 4'hF : 4'h0)) : (4'h0);
        green <= video_on ? (((ball||paddle1) ? 4'hF : 4'h0)) : (4'h0);
        blue <= video_on ? (((ball||paddle1) ? 4'hF : 4'h0)) : (4'h0);

    end
end
endmodule

```

//-----

```

module toplevel(clk, h_sync, v_sync, R, G, B, xcoordball, ycoordball, xcoordpad);
    input clk;
    output h_sync;
    output v_sync;
    wire [9:0] HC;
    wire [9:0] VC;
    output [3:0] R;
    output [3:0] G;
    output [3:0] B;
    reg clk_D;
    wire tgv;
    wire v_on;
    wire [9:0] xloc;
    wire [9:0] yloc;
    input [9:0] xcoordball;
    input [9:0] ycoordball;
    input [9:0] xcoordpad;

    clk_div g0(.clk(clk), .clk_d(clk_D));

```



```

h_counter a1(.clk(clk_D), .h_count(HC), .trig_v(tgv));
v_counter a2(.enable_v(tgv), .clk(clk_D), .v_count(VC));

vga_sync a3(.h_count(HC), .v_count(VC), .h_sync(h_sync), .v_sync(v_sync), .video_on(v_on),
.x_loc(xloc), .y_loc(yloc));

pixel_gen a4(.clk_d(clk_D), .pixel_x(xloc), .pixel_y(yloc), .video_on(v_on), .red(R),
.green(G), .blue(B), .xcoordball(xcoordball), .ycoordball(ycoordball), .xcoordpad(xcoordpad));

endmodule

//-----

module counter(x,q);
    input [3:0] x;
    output [3:0] q;
    assign q= x+1;
endmodule

//-----

//module send(x,q);
//    input x ;
//    output q;
//    assign q = ~x;
//    endmodule

//-----

module sque_detec (clk ,rst ,sque_in ,out ,check ,start ,h_sync , v_sync , R, G, B);
    input sque_in ,rst ,clk ,start ;

```

```

output out;
output [3:0] check ;
output h_sync;
output v_sync;
output [3:0] R;
output [3:0] G;
output [3:0] B;
reg Dh_sync;
reg Dv_sync;
reg [3:0] DR;
reg [3:0] DG;
reg [3:0] DB;
reg [3:0] x,p;
reg [9:0] xcoordball;
reg [9:0] ycoordball;
reg [9:0] xcoordpad;
reg [3:0] DDR;
reg [3:0] DDG;
reg [3:0] DDB;
reg DDh_sync;
reg DDv_sync;
reg [3:0] B_speedx;
reg [3:0] B_speedy;
reg [3:0] speedx,speedy,score;
reg s,r;

wire [3:0] q;
initial
begin
x = 4'b0000;
p=0;
xcoordball = 300;
ycoordball = 200;
xcoordpad = 300;

```

```

DR = 0;
DG = 0;
DB = 0;
Dv_sync=0;
Dh_sync=0;

end

counter c2 (.x(x),.q(q));
toplevel c1(clk , DDh_sync, DDv_sync, DDR, DDG, DDB, xcoordball , ycoordball , xcoordpad);
collision c3(xcoordball ,ycoordball ,B_speedx,B_speedy,xcoordpad ,speedx ,speedy ,rst ,s);
parameter uart= 3'b001,
          ball= 3'b010,
          collision= 3'b011,
          paddle= 3'b100,
          win= 3'b101,
          vga= 3'b110,
          S6= 3'b111;

reg [2:0] present,next;
always@(posedge clk)
begin
    if (rst)
        present <= uart;
    else
        present <= next;
    if (start)
        present <= uart;
    else
        present <= next;
end
always@(present or sque_in)
begin

```

```

    next = uart;
case( present )
    start: begin if(sque_in) next = uart; end
    uart: begin if(sque_in) next = ball;x=q; end
    ball: begin if(sque_in) next = collision;
            else next= ball;end
    collision: begin if(sque_in) next = paddle; end
    paddle: begin if(sque_in) next = win; end
    win: begin if(sque_in) next = vga;if (score== 15) r=1; end
    vga: begin if(sque_in) next = S6;
    Dh_sync= DDh_sync;Dv_sync=DDv_sync;DR=DDR;DG=DDG;DB=DDB; end
    S6: begin if(sque_in) next = uart; end
endcase
end
assign h_sync = Dh_sync ;
assign v_sync = Dv_sync;
assign R = DR;
assign G = DG;
assign B = DB;
assign out = (present == ball) ? 1:(present == paddle) ? 1:0;
assign check = (present == ball) ? q:p;
assign rst = 1;

endmodule

```

## Testbench

```

module tb();
    reg clk ,rst ,sque_in ,start ;
    wire out;
    wire [3:0]check;
    wire h;
    wire v;

```

```

wire [3:0] R;
wire [3:0] G;
wire [3:0] B;

sque_detec c1(.clk(clk),.rst(rst),.sque_in(sque_in),.out(out),.check(check),.start(start),
.h_sync(h),.v_sync(v),.R(R),.G(G),.B(B));

initial
begin
clk = 1'b0;
rst=1'b0;
sque_in= 1'b1;
start=1'b0;

end
always
begin
#5 clk = ~clk;
end
initial
begin
$dumpfile("dump.vcd");
$dumppvars(1,tb);
$monitor("Time =", $time, " clk = %d", clk, "out = %d",out,"sque_in = %d",sque_in);
#6000 $finish;
end
endmodule

```