# PING-PONG GAME IMPLEMENTATION ON BASYS3 BOARD USING VERILOG HDL

A project report submitted in partial fulfillment of the requirements for the

Degree of

# Bachelor of Technology

In

# Electronics and Communication Engineering By

P. Jaya Sankar Reddy(N181004) U. Manideep (N180605) B.Chakra Rao (N180276)

# Under the Guidance Of

Mr. SHYAM PERIKA



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES NUZVID CAMPUS**

**ANDHRA PRADESH-521 202**

**APRIL 2024**



***Dedication***

*I dedicate this thesis to my esteemed guide, Shyam Perika, whose expertise, guidance, and unwavering support have been instrumental in shaping this work. Your mentorship has not only enriched my academic journey but also inspired me to strive for excellence. Thank you for your patience, wisdom, and encouragement throughout this process. This thesis is a testament to your dedication and expertise, and I am deeply grateful for the opportunity to learn from you.*

1



# THESIS CERTIFICATE

This is to certify that the work entitled “**Ping pong game implementation using verilog on basys3 board using verilog Hdl**” is a bonafide record of authentic work carried out by N181004,N180605,N180276 under my supervision and guidance for the partial fulfillment of the requirement of the award of the degree of Bachelor of Technology in the department of Electronics and Communication Engineering at **RGUKT- NUZVID.**

#### The results embodied in this work have not been submitted to any other university or institute for the award of any degree or diploma. This thesis, in our opinion, is worthy of consideration for the award of the degree of Bachelor of Technology in accordance with the regulations of the institute.

**Project Guide Shyam Perika** Assistant Professor, Department of ECE, RGUKT – Nuzvid 521202

**Head of the Department** Department of ECE RGUKT Nuzvid campus Nuzvid - 521202

#### Date: - 06/04/2024 Place: - Nuzvid

#### Signature of External Examiner

2

**ACKNOWLEDGEMENT**

I would like to express my special thanks of gratitude to my project guide **Mr. Shyam Perika,** Asst.Prof, Dept. of ECE, who gave me the golden opportunity to do this wonderful project on the topic “**Ping pong game implementation using verilog on basys3 board using verilog Hdl**”, which also helped me in doing a lot of research and I came to know about so many new things I am really thankful to them.

Secondly, I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.



3

|  |  |
| --- | --- |
| **TABLE OF CONTENTS** | |
| **TITLE** | **1** |
| **CERTIFICATE** | **2** |
| **ACKNOWLEDGEMENT** | **3** |
| **ABSTRACT** | **6** |

1. **CHAPTER – 1: INTRODUCTION**

|  |  |  |
| --- | --- | --- |
| 1.1 | Introduction | 7 |
| 1.2 | Goals | 8 |
| 1.3 | Background | 8 |
| **2. CHAPTER – 2: FPGA BOARD** | | |
| 2.1 | Basys 3 board | 9 |
| 2.2 | Feature of basys3 board | 10 |
| 2.3 | FPGA baord with callouts | 11 |
| **3. CHAPTER – 3 VGA-MONITOR** | | |
| 3.1 | VGA-monitor | 13 |
| 3.2 | Flow chart | 14 |
| 3.2.1 Explanation | | 14-16 |
| **4. CHAPTER – 4: VERILOG MODULES** | | |
| 4.1 | Modules | 16 |
| 4.2 | Vga-controller module | 17 |

|  |  |  |  |
| --- | --- | --- | --- |
| 4.2.2.1 25 MHz clock | | | 18 |
| 4.2.2.2 H\_sysc & Vsync | | | 19 |
| 4.2.1 | | Block diagram | 20 |
| 4.2.2 | | Verilog code | 21-22 |
| 4.3 pixel generation module | | | 23 |
| 4.3.1 | | Block diagram | 24 |
| 4.3.2 | | Verilog code | 25-30 |
| 4.4 | Debounce module | | 30 |
| 4.4.1 | | Block diagram | 30 |
| 4.4.2 | | Verilog code | 30 |
| 4.5 | Ball\_rom | | 31 |
| 4.5.1 | | Block diagram | 31 |
| 4.5.2 | | Verilog code | 32 |
| 4.6 Top module | | | 33 |
| 4.6.1 | | Block diagram | 34 |
| 4.6.2 | | Verilog code | 35 |
| **5. CHAPTER – 5: INTERFACING WITH FPGA** | | | |
| 5.1 | FPGA implementation | | 36 |
| 5.2 | Result | | 36 |

**6. CHAPTER – 6**

|  |  |  |
| --- | --- | --- |
| 6.1 | Conclusion | 37 |
| 6.2 | References | 38 |

# ABSTRACT

In this project we are implementing Pong Game which is controlled by single player using knobs, and displays the game on a VGA monitor. It is a table-tennis inspired game featuring simple one, dimensional graphics. In it, the player controls the paddle by moving it vertically across the up or the downside of the screen. A sphere representing a ping pong ball travels across the screen in a linear trajectory. If the ball strikes the perimeter of the playing field, the obstacle in the middle of the room or one of the simulated paddles, the ball ricochets based on the angle of the impact . With every impact on the playing field or obstacle, the ball picks up speed. Speed is reduced upon hitting the paddle.The aim of this project is to create a Verilog - based application for Pong.

This project mainly focused on the design and implementation of an FPGA-based Ping Pong game, consisting of both the hardware and software design working coherently. For hardware-wise, the host computer was used to program and configure the design; FPGA was implemented for operating the game, and the VGA monitor for display. At the same time, the software part included the design of the overall system, input key module, VGA display module, as well as the game control module. The game design focused on "single Player Mode", which simulates the real-life PING-PONG Game. The movement of the paddles was controlled by pressing the push buttons of FPGA.

# CHAPTER-1

* 1. **INTRODUCTION**

In this project we are implementing Pong Game which is controlled by single player using knobs, and displays the game on a VGA monitor.It is a table-tennis inspired game featuring simple one, dimensional graphics.

In it, the player controls the paddle by moving it vertically across the up or the downside of the screen. A sphere representing a ping pong ball travels across the screen in a linear trajectory. If the ball strikes the perimeter of the playing field, the obstacle in the middle of the room or one of the simulated paddles, the ball ricochets based on the angle of the impact . With every impact on the playing field or obstacle, the ball picks up speed. Speed is reduced upon hitting the paddle.

The game consists of a display with a sliding paddle that can be controlled by the user, and a fixed wall that acts as the upper bound. When the user presses the start button, which is the up button, the ball will begin bounce up, and will bounce off the wall and attempt to hit the paddle. If the ball hits the paddle, it bounces up again and continues to bounce until it misses the paddle. The game is over when the ball fails to hit the paddle. The user will be able to use the left and right pushbuttons to determine the movement of the paddle. In order to reset the game, the user must push the center button. The time of the run will be recorded on a 7 segment display.

* 1. **GOALS:**

The goal of our project was to recreate the classic 1-player game, Pong, with added special features. The vision was to have the players hold and move physical paddles hit to a virtual ball being projected onto the screen. Image detection would be performed to detect the paddles, and calculations in hardware would detect collisions, as well as the velocities and angles of the paddles. The game would be played on a projected screen.

## Background:

Pong is one of the oldest and most popular arcade games. It is a digital version of tennis where two players have a racquet/paddle on either side of the screen and the object of the game is to hit the ball and score on the opponent. We wanted make this game more interactive such that the players have physical paddles in their hands and using image detection, we would detect and draw them on a screen. A ball will be drawn on the screen and the players have to move their paddles to interact with the ball. Contrary to the classic game where the paddles are constrained to move only up and down, our game would allow the players to move it in any direction. Calculations based on the angle and velocities would be done to correspondingly interact with the ball.

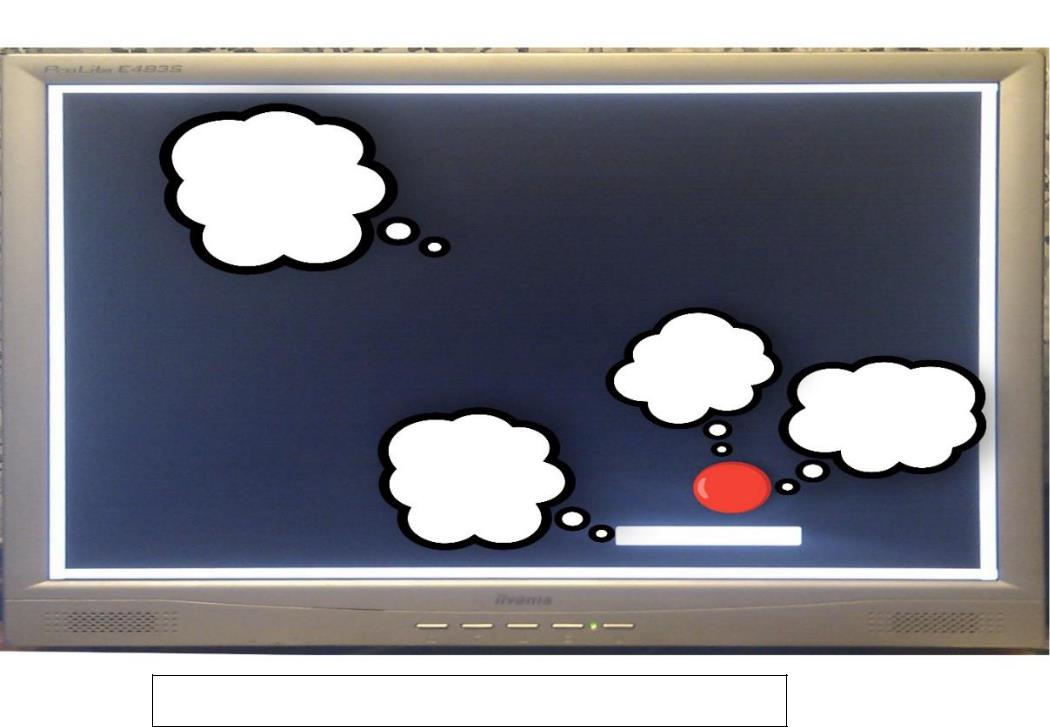


Fig1: Simple Archeitecture of the ping-pong game

# CHAPTER-2 FPGA BOARD

## Basys3 Board

The Basys3 board is a complete, ready-to-use digital circuit development platform based on the latest Artix®-7 Field Programmable Gate Array (FPGA) from Xilinx®. With its high-capacity FPGA (Xilinx part number XC7A35T1CPG236C), low overall cost, and collection of USB, VGA, and other ports, the Basys3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs, and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits. The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance, and more resources than earlier designs.

* 1. **FEATURES:**

1. 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)
2. 1,800 Kbits of fast block RAM
3. Five clock management tiles, each with a phase-locked loop (PLL)
4. 90 DSP slices x internal clock speeds exceeding 450MHz x On-chip analog-to-digital converter (XADC)

The Basys 3 also offers an improved collection of ports and peripherals, including:

1. 16 user switches x 16 user LEDs x 5 user pushbuttons
2. 4-digit 7-segment display
3. Three Pmod ports x Pmod for XADC signals
4. 12-bit VGA output
5. USB-UART Bridge
6. Serial Flash
7. Digilent USB-JTAG port for FPGA programming and communication
8. USB HID Host for mice, keyboards and memory sticks

The Basys 3 works with Xilinx's new high-performance Vivado™ Design Suite. Vivado includes many new tools and design flows that facilitate and enhance the latest design methods. It runs faster, allows better use of FPGA resources, and allows designers to focus their time evaluating design alternatives. The System Edition includes an on-chip logic analyzer, high-level synthesis tool, other cutting-edge tools, and the free WebPACK™ version allows Basys 3 designs to be created at no additional cost.

## FPGA board with callouts:

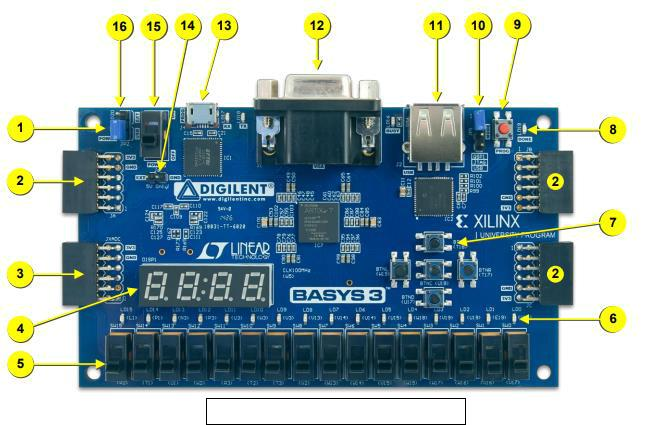
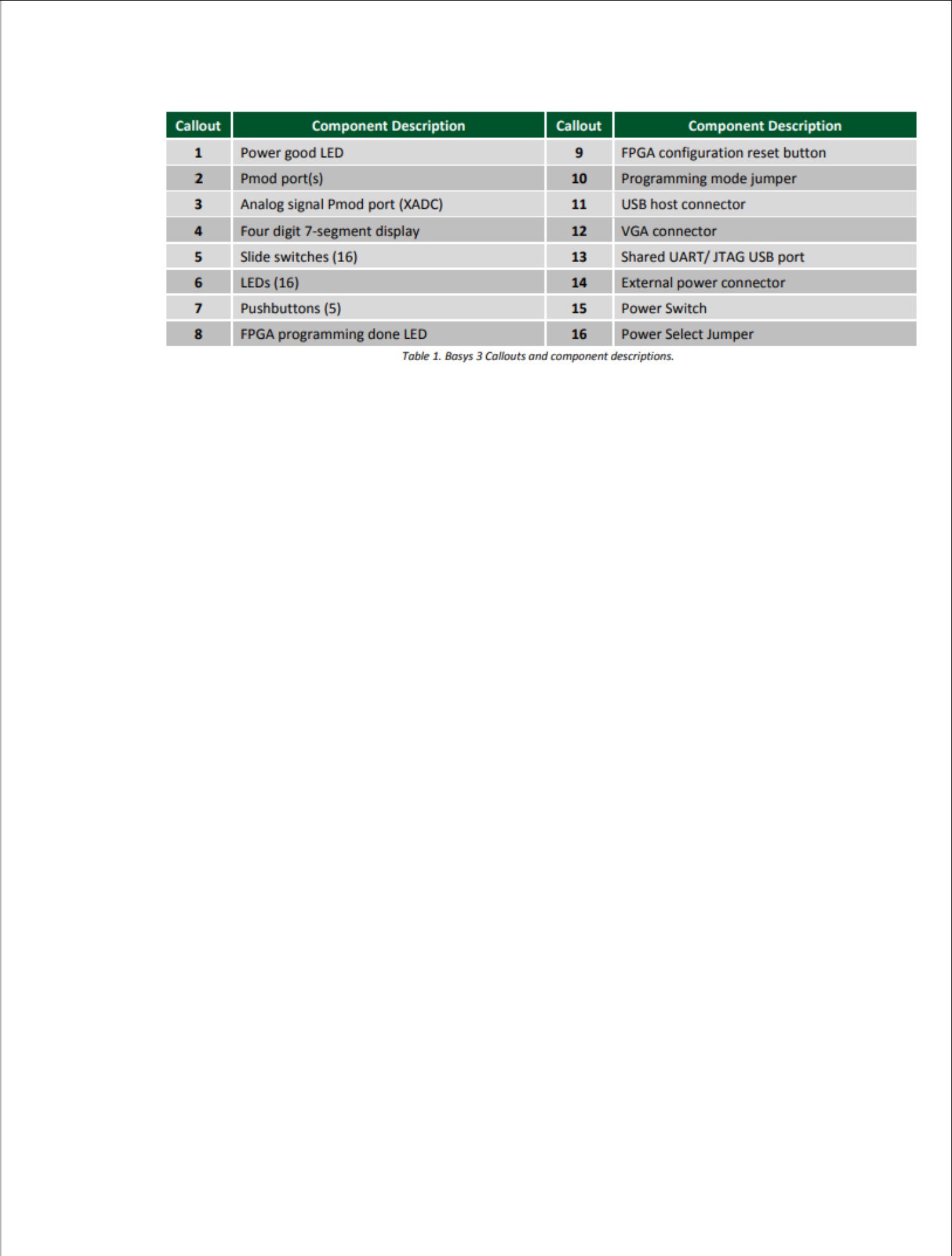


Fig2: Basys3 FPGA board with callouts

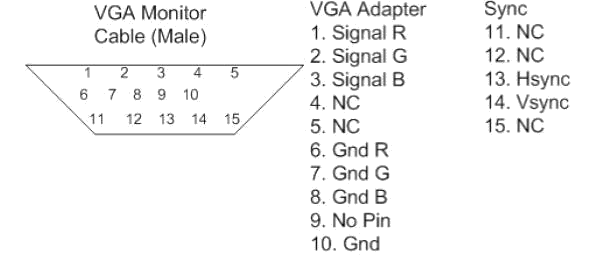


12

# CHAPTER-3 VGA MONITOR

* 1. **VGA-MONITOR:**

#### In this project, we employed a VGA monitor to display the game. A VGA monitor uses a 15 pin connection. Four of these pins are ground pins, one for master ground, and three for color signal ground. Three pins are the analog color pins, one for red, another green, and one for blue. In addition to these pins, two pins are used for the horizontal and vertical sync signals. The rest of the fifteen pins are unused. Figure 2 shows a diagram of the pins on a male connector, and their usage. For this project, the three color pins and the two sync pins were directly connected to the digital output from the FPGA.



A 640x480 pixel VGA display, used for this project, takes pixel data at a rate of 25.175 MHz. Using the Digital Clock Manager (DCM) built into the FPGA, and a 40 MHz clock signal, the 5/8 scaling produces a signal at 25 MHz. This is within the error permitted by a VGA monitor and so can be used to generate the signal. HSync and VSync are two signals which control the rate of the monitor, and inform it when to begin a new row of pixels and a new screen. These signals need to occur at the frequencies of 31.47 KHz for HSync and 59.94 Hz for VSync. HSync should have a pulse length of 64

25.175 MHz clock periods, or 3.813 us, and VSync should have a pulse length equal to the time required to draw two rows of pixels, or 63.555 us. Timing information is given in Table1.

13

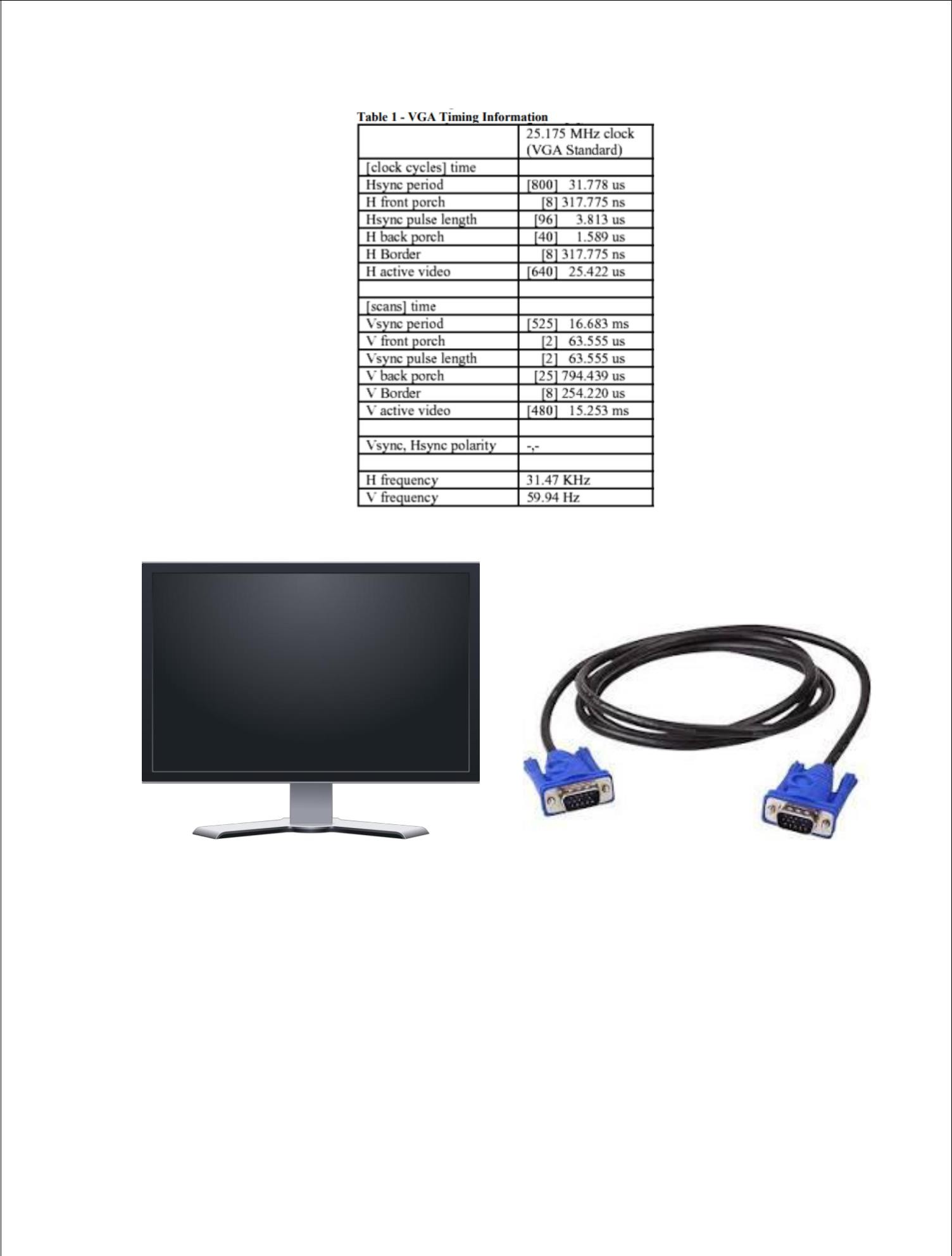
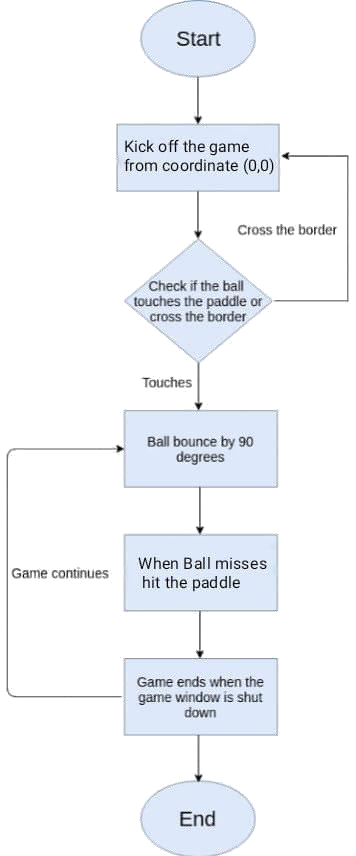


Fig3: VGA monitor Fig4: VGA connector

14

## Flow Chart:



* + 1. **Explanation:**

The flowchart you have represents the core logic behind a basic paddle-ball game, often likened to classic arcade games like Pong. Let's dissect this flowchart in more detail, exploring the intricacies of the game loop and potential enhancements.

### Initialization (Start):

The game begins with a setup phase. Here, essential elements like the game window, playing area boundaries, and objects (paddle and ball) are created and initialized with starting positions. The ball's initial position (0, 0) often refers to the center of the game window.

This phase might also involve setting up the ball's initial velocity vector, determining its direction and speed.

### Main Game Loop:

This is the heart of the game. It's a continuous cycle that keeps the game running until a specific condition (like the ball missing the paddle) is met.

**Within the loop:**

Input Detection: The game constantly checks for player input. This could involve detecting keyboard presses (up/down for paddle movement) or mouse movement (if the paddle is controlled by the mouse).

Paddle Movement: Based on the detected input, the game updates the position of the paddle on the screen. This ensures the player can control the paddle in real-time.

Ball Movement: The game updates the ball's position based on its current velocity. This creates the illusion of the ball moving continuously.

Collision Detection: The game checks for collisions between the ball and various objects:

Paddle Collision: If the ball collides with the paddle, the game logic calculates a new trajectory for the ball. This typically involves bouncing the ball back at an angle determined by the point of contact on the paddle.

Wall Collision: The game checks if the ball hits the top, bottom, or side walls of the playing area. Depending on the game design, the ball might simply bounce off these walls, or the game might end if the ball hits a specific wall (e.g., the bottom wall in Pong).

### Branching Paths - Ball Behavior:

The flowchart branches into two paths after collision detection:

Ball hits Paddle: If the ball collides with the paddle, the game updates the ball's velocity based on the collision point. This can introduce an element of skill, as players can strategically angle the paddle to control the ball's bounce direction.

Ball misses Paddle (Game Over): If the ball misses the paddle entirely (often by crossing the bottom boundary), the game ends. This signifies the player failing to keep the ball in play. Additional logic might display a "Game Over" message and offer options to restart or exit the game.

### Rendering and Refresh:

After updating the positions of the ball and paddle, the game redraws the entire scene onto the game window. This creates the illusion of smooth movement by constantly updating the visuals.

The game then refreshes the window, displaying the updated scene to the player.

## End of Game:

The game loop continues until a specific condition triggers the end of the game. As mentioned earlier, the ball missing the paddle is a common end condition. However, other possibilities might exist:



Completing levels or objectives.

* Reaching a score limit.
* Player intentionally exiting the game.

### Beyond the Basics - Enhancements:

The core logic outlined above represents a simple paddle-ball game. Developers can incorporate various enhancements to make the game more engaging:

Power-ups: Items that can appear on the screen, offering temporary benefits like increased paddle size, slower ball speed, or bonus points.

Multiple Levels: Increasing difficulty by introducing obstacles, changing ball speed, or modifying the playing area layout.

Scorekeeping: Tracking points earned by keeping the ball in play and completing objectives.

Visual and Audio Effects: Enhancing the game's atmosphere with sound effects for collisions and background music, as well as visual flourishes for power-ups and scoring events.

By understanding this basic flowchart and its potential for expansion, you gain a solid foundation for appreciating the development process behind even seemingly simple games. With creativity and additional logic, this core concept can be transformed into a rich and engaging gaming experience.

# CHAPTER-4 VERILOG MODULES



* 1. **MODULES:**

FPGA Verilog Hdl based Ping Pong Game project mainly consists of five modulues those are as

1. VGA controller module
2. Pixel- Generation module
3. Debounce module
4. Ball\_Rom module
5. Top module
   1. **VGA CONTROLLER MODULE:**

For this project we have chooen DELL monitor. The features available in dell monitor will be given in the datasheet there we can find the frequency of the monitor and the pixel rate as well..

In our FPGA laboratory we have DELL monitor which can take 25MHz frequency

* In horizontal there are 0 to 799 pixels.
* In vertical there are 0 to 524 pixels.

## 25MHz CLOCK:

Now we want 25MHz pixels, but monitor give~~s 100M~~Hz pixels. That’s why we can use frequency divider

100 ℎ

to convert 100MHz pixels to 25MHz pixels. ( 25 ℎ ) ∗ 200 It can give half clock period.

## H\_sync & V\_sync:

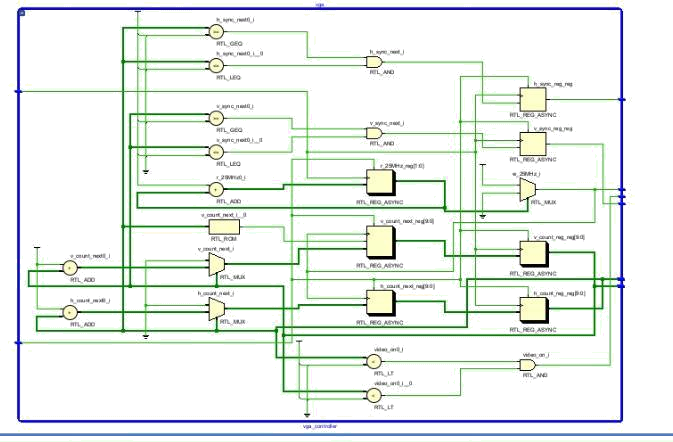
Hsync is "Horizontal Sync", it is a pulse that synchronizes the start of the horizontal picture scan line in the monitor with the picture source that created it.

Vsync is the equivalent vertical synchronization, it ensures the monitor scan starts at the top of the picture at the right time.

Pixels are not controlled by them at all. A Pixel is the smallest picture unit that can be displayed. Imagine the screen is a grid (like a spreadsheet) and each cell can contain one shade or color, the picture is composed of many of them, each small enough that the human eye cannot distinguish them individually so it sees them 'blend' into a picture. In the spreadsheet analogy, each cell is one pixel, Hsync would mean start at column zero and Vsync would mean start at row zero.

In an 800x600 pixel screen there are 480000 pixels (multiply them) so the time to scan the whole picture would be the period of one pixel (1/f) multiplied by the number of pixels. That only gives the time to wait on the visible part of the picture though, it doesn't include the time spent in sync pulses. The total is the sum of 800 horizontal sync pulse durations plus the time of all the pixels. Depending on where you start your timing, you might also have to add one vertical sync pulse duration. Without knowing how long those sync pulses last.

* + 1. **BLOCK DIAGRAM:**



* + 1. **VERILOG CODE:**

module vga\_controller(

input clk\_100MHz, // from Basys 3

input reset, // system reset

output video\_on, // ON while pixel counts for x and y and within display area output hsync, // horizontal sync

output vsync, // vertical sync

output p\_tick, // the 25MHz pixel/second rate signal, pixel tick output [9:0] x, // pixel count/position of pixel x, max 0-799 output [9:0] y // pixel count/position of pixel y, max 0-524

);

// Based on VGA standards found at vesa.org for 640x480 resolution

// Total horizontal width of screen = 800 pixels, partitioned into sections

parameter HD = 640; // horizontal display area width in pixels parameter HF = 48; // horizontal front porch width in pixels parameter HB = 16; // horizontal back porch width in pixels parameter HR = 96; // horizontal retrace width in pixels

parameter HMAX = HD+HF+HB+HR-1; // max value of horizontal counter = 799 // Total vertical length of screen = 525 pixels, partitioned into sections

|  |  |
| --- | --- |
| parameter VD = 480; | // vertical display area length in pixels |
| parameter VF = 10; | // vertical front porch length in pixels |
| parameter VB = 33; | // vertical back porch length in pixels |
| parameter VR = 2; | // vertical retrace length in pixels |

parameter VMAX = VD+VF+VB+VR-1; // max value of vertical counter = 524

// \*\*\* Generate 25MHz from 100MHz

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* reg [1:0] r\_25MHz;

wire w\_25MHz;

always @(posedge clk\_100MHz or posedge reset) if(reset)

r\_25MHz <= 0;

else

r\_25MHz <= r\_25MHz + 1;

assign w\_25MHz = (r\_25MHz == 0) ? 1 : 0; // assert tick 1/4 of the time

//

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*

// Counter Registers, two each for buffering to avoid glitches

reg [9:0] h\_count\_reg, h\_count\_next; reg [9:0] v\_count\_reg, v\_count\_next;

// Output Buffers

reg v\_sync\_reg, h\_sync\_reg; wire v\_sync\_next, h\_sync\_next;

// Register Control

always @(posedge clk\_100MHz or posedge reset) if(reset) begin

v\_count\_reg <= 0;

h\_count\_reg <= 0; v\_sync\_reg <= 1'b0; h\_sync\_reg <= 1'b0;

end

else begin

v\_count\_reg <= v\_count\_next; h\_count\_reg <= h\_count\_next; v\_sync\_reg <= v\_sync\_next; h\_sync\_reg <= h\_sync\_next;

end

//Logic for horizontal counter

always @(posedge w\_25MHz or posedge reset) // pixel tick

if(reset)

h\_count\_next = 0; else

if(h\_count\_reg == HMAX) h\_count\_next = 0;

else

h\_count\_next = h\_count\_reg + 1;

// end of horizontal scan

// Logic for vertical counter

always @(posedge w\_25MHz or posedge reset) if(reset)

v\_count\_next = 0; else

if(h\_count\_reg == HMAX) if((v\_count\_reg == VMAX))

v\_count\_next = 0; else

// end of horizontal scan

// end of vertical scan

v\_count\_next = v\_count\_reg + 1;

// h\_sync\_next asserted within the horizontal retrace area

assign h\_sync\_next = (h\_count\_reg >= (HD+HB) && h\_count\_reg <= (HD+HB+HR-1));

// v\_sync\_next asserted within the vertical retrace area

assign v\_sync\_next = (v\_count\_reg >= (VD+VB) && v\_count\_reg <= (VD+VB+VR-1));

// Video ON/OFF - only ON while pixel counts are within the display area

assign video\_on = (h\_count\_reg < HD) && (v\_count\_reg < VD); // 0-639 and 0-479 respectively

// Outputs

assign hsync = h\_sync\_reg; assign vsync = v\_sync\_reg; assign x = h\_count\_reg; assign y = v\_count\_reg; assign p\_tick = w\_25MHz;

endmodule

* 1. **PIXEL-GENERATION MODULE:**

It involves using the reconfigurable hardware of the FPGA to dynamically create and display visual content.

Here's a breakdown of the main approaches:

1. VGA Display:

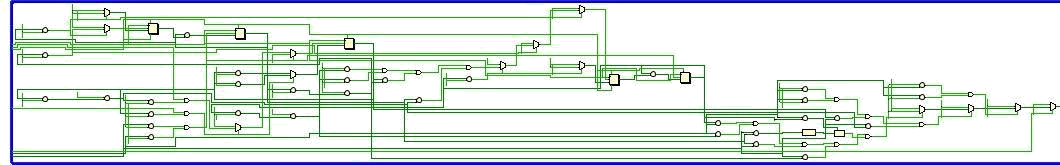
This is the classic method for generating basic graphics on an FPGA.The FPGA outputs digital signals for red, green, and blue (RGB) along with synchronization signals (horizontal and vertical sync) to drive a VGA monitor.

You can generate the pixel colors within the FPGA logic based on various criteria like pixel coordinates, pre-stored images, or real-time calculations. Requires understanding VGA timing and protocols, and involves writing Verilog.

1. Image Display from Memory:

Instead of generating colors on the fly, you can store image data in a dedicated memory block within the FPGA.The FPGA then reads the appropriate pixel values from memory based on the current position on the screen and outputs the RGB signals.This allows for displaying pre-defined images stored in various formats (BMP, JPEG, etc.), and enables faster rendering for complex graphics.Requires implementing memory access logic and managing image data formats within your FPGA design.

* + 1. **BLOCK DIAGRAM:**



* + 1. **VERILOG CODE:**

module pixel\_gen(

input clk, input reset, input up, input down,

input video\_on, input [9:0] x,

input [9:0] y,

output reg [11:0] rgb

);

// maximum x, y values in display area parameter X\_MAX = 639; parameter Y\_MAX = 479;

// create 60Hz refresh tick wire refresh\_tick;

assign refresh\_tick = ((y == 481) && (x == 0)) ? 1 : 0; // start of vsync(vertical retrace)

// WALL

// wall boundaries

parameter X\_WALL\_L = 32;

parameter X\_WALL\_R = 39; // 8 pixels wide

// PADDLE

// paddle horizontal boundaries parameter X\_PAD\_L = 600;

parameter X\_PAD\_R = 603; // 4 pixels wide

// paddle vertical boundary signals wire [9:0] y\_pad\_t, y\_pad\_b;

parameter PAD\_HEIGHT = 72; // 72 pixels high

// register to track top boundary and buffer reg [9:0] y\_pad\_reg, y\_pad\_next;

// paddle moving velocity when a button is pressed

parameter PAD\_VELOCITY = 3; // change to speed up or slow down paddle movement

// BALL

// square rom boundaries parameter BALL\_SIZE = 8;

// ball horizontal boundary signals wire [9:0] x\_ball\_l, x\_ball\_r;

// ball vertical boundary signals wire [9:0] y\_ball\_t, y\_ball\_b;

// register to track top left position reg [9:0] y\_ball\_reg, x\_ball\_reg;

// signals for register buffer

wire [9:0] y\_ball\_next, x\_ball\_next;

// registers to track ball speed and buffers reg [9:0] x\_delta\_reg, x\_delta\_next;

reg [9:0] y\_delta\_reg, y\_delta\_next;

// positive or negative ball velocity parameter BALL\_VELOCITY\_POS = 2; parameter BALL\_VELOCITY\_NEG = -2;

// round ball from square image

wire [2:0] rom\_addr, rom\_col; // 3-bit rom address and rom column reg [7:0] rom\_data; // data at current rom address

wire rom\_bit; // signify when rom data is 1 or 0 for ball rgb control

// Register Control

always @(posedge clk or posedge reset) if(reset) begin

y\_pad\_reg <= 0;

x\_ball\_reg <= 0;

y\_ball\_reg <= 0; x\_delta\_reg <= 10'h002; y\_delta\_reg <= 10'h002;

end

else begin

y\_pad\_reg <= y\_pad\_next; x\_ball\_reg <= x\_ball\_next; y\_ball\_reg <= y\_ball\_next;

x\_delta\_reg <= x\_delta\_next; y\_delta\_reg <= y\_delta\_next;

end

// ball rom always @\*

case(rom\_addr)

3'b000 : rom\_data = 8'b00111100; // \*\*\*\*

3'b001 : rom\_data = 8'b01111110; // \*\*\*\*\*\*

3'b010 : rom\_data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'b011 : rom\_data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'b100 : rom\_data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'b101 : rom\_data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'b110 : rom\_data = 8'b01111110; // \*\*\*\*\*\*

3'b111 : rom\_data = 8'b00111100; // \*\*\*\* endcase

// OBJECT STATUS SIGNALS

wire wall\_on, pad\_on, sq\_ball\_on, ball\_on;

wire [11:0] wall\_rgb, pad\_rgb, ball\_rgb, bg\_rgb;

// pixel within wall boundaries

assign wall\_on = ((X\_WALL\_L <= x) && (x <= X\_WALL\_R)) ? 1 : 0;

// assign object colors assign wall\_rgb = 12'hAAA; assign pad\_rgb = 12'hAAA; assign ball\_rgb = 12'hFFF; assign bg\_rgb = 12'h111;

// gray wall

// gray paddle

// white ball

// close to black background

// paddle

assign y\_pad\_t = y\_pad\_reg; // paddle top position

assign y\_pad\_b = y\_pad\_t + PAD\_HEIGHT - 1; // paddle bottom position

assign pad\_on = (X\_PAD\_L <= x) && (x <= X\_PAD\_R) && // pixel within paddle boundaries (y\_pad\_t <= y) && (y <= y\_pad\_b);

// Paddle Control always @\* begin

y\_pad\_next = y\_pad\_reg; // no move

if(refresh\_tick)

if(up & (y\_pad\_t > PAD\_VELOCITY))

y\_pad\_next = y\_pad\_reg - PAD\_VELOCITY; // move up else if(down & (y\_pad\_b < (Y\_MAX - PAD\_VELOCITY)))

y\_pad\_next = y\_pad\_reg + PAD\_VELOCITY; // move down

end

// rom data square boundaries assign x\_ball\_l = x\_ball\_reg; assign y\_ball\_t = y\_ball\_reg;

assign x\_ball\_r = x\_ball\_l + BALL\_SIZE - 1; assign y\_ball\_b = y\_ball\_t + BALL\_SIZE - 1;

// pixel within rom square boundaries

assign sq\_ball\_on = (x\_ball\_l <= x) && (x <= x\_ball\_r) && (y\_ball\_t <= y) && (y <= y\_ball\_b);

// map current pixel location to rom addr/col

assign rom\_addr = y[2:0] - y\_ball\_t[2:0]; // 3-bit address assign rom\_col = x[2:0] - x\_ball\_l[2:0]; // 3-bit column index

assign rom\_bit = rom\_data[rom\_col]; // 1-bit signal rom data by column

// pixel within round ball

assign ball\_on = sq\_ball\_on & rom\_bit; // within square boundaries AND rom data bit == 1

// new ball position

assign x\_ball\_next = (refresh\_tick) ? x\_ball\_reg + x\_delta\_reg : x\_ball\_reg; assign y\_ball\_next = (refresh\_tick) ? y\_ball\_reg + y\_delta\_reg : y\_ball\_reg;

// change ball direction after collision always @\* begin

x\_delta\_next = x\_delta\_reg; y\_delta\_next = y\_delta\_reg;

if(y\_ball\_t < 1) // collide with top y\_delta\_next = BALL\_VELOCITY\_POS; // move down

else if(y\_ball\_b > Y\_MAX) // collide with bottom

y\_delta\_next = BALL\_VELOCITY\_NEG; // move up

else if(x\_ball\_l <= X\_WALL\_R) // collide with wall x\_delta\_next = BALL\_VELOCITY\_POS; // move right

else if((X\_PAD\_L <= x\_ball\_r) && (x\_ball\_r <= X\_PAD\_R) &&

(y\_pad\_t <= y\_ball\_b) && (y\_ball\_t <= y\_pad\_b)) // collide with paddle

x\_delta\_next = BALL\_VELOCITY\_NEG; // move left

end

// rgb multiplexing circuit always @\* if(~video\_on)

rgb = 12'h000; // no value, blank else

if(wall\_on)

rgb = wall\_rgb; // wall color else if(pad\_on)

rgb = pad\_rgb; // paddle color else if(ball\_on)

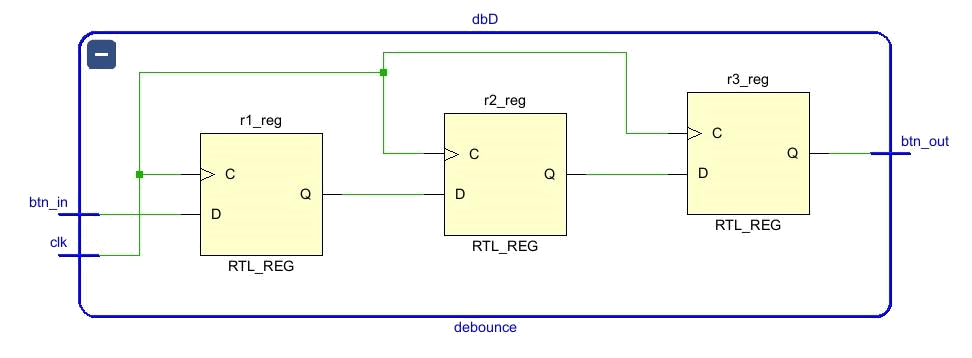
rgb = ball\_rgb; // ball color else

rgb = bg\_rgb; // background endmodule

* 1. **DEBOUNCE MODULE:**

To safely use buttons in our design, we need to debounce them. Debouncing ensures we get a single, clean transition from button presses.

* + 1. **BLOCK DIAGRAM:**



* + 1. **VERILOG CODE:**

module debounce(

input clk, // 100MHz input btn\_in,

output btn\_out

);

reg r1, r2, r3;

always @(posedge clk) begin

end

assign btn\_out = r3; endmodule /// debounce R

|  |  |
| --- | --- |
| r1 | <= btn\_in; |
| r2 | <= r1; |
| r3 | <= r2; |

* 1. **BALL\_ROM MODULE:**

Normally it is easy to print a box on a monitor as it has a specific vertex at each side so we can easily identify the range of horizontal as well as the vertical pixel and will give the RGB values through the VGA pin.

But here we need to print a ball which is a circle shape as we know that circle has no particular vertex as Like Square so that to print the circle on the monitor we implement a Rom (Read -Only- Memory) and stored the shape of the ball in binary format at each address of the Rom. By using the counters we will print the circle shape at our desired position in the monitor.

* + 1. **BLOCK DIAGRAM:**

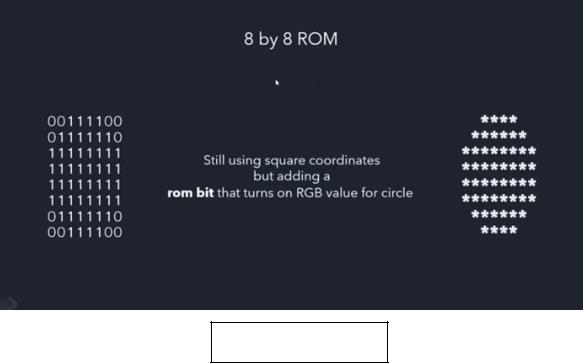


Fig: 5 Ball rom

* + 1. **VERILOG CODE:**

module ball\_rom(

input [2:0] addr, // 3-bit address

output reg [7:0] data // 8-bit data

);

always @\* case(addr)

3'b000 : data = 8'b00111100; // \*\*\*\*

3'b001 : data = 8'b01111110; // \*\*\*\*\*\*

3'b010 : data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'b011 : data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'b100 : data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'b101 : data = 8'b11111111; // \*\*\*\*\*\*\*\*

3'b110 : data = 8'b01111110; // \*\*\*\*\*\*

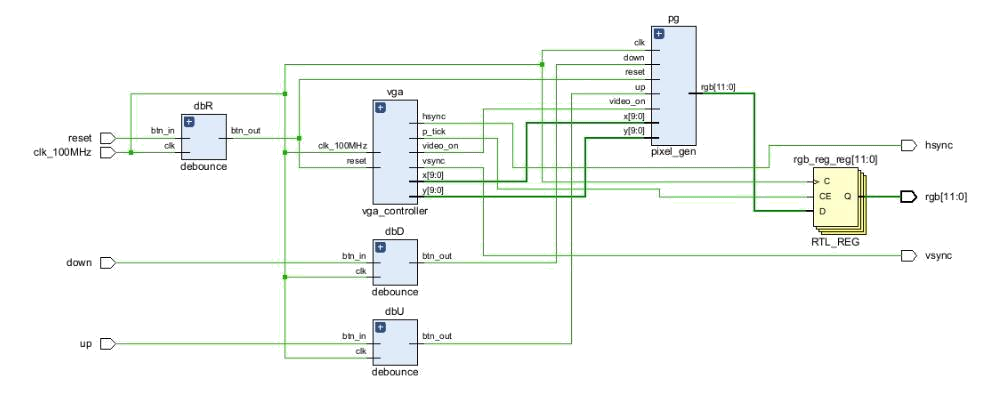
3'b111 : data = 8'b00111100; // \*\*\*\* endcase

endmodule

* 1. **TOP MODULE:**

We Instantiate all the above 3 modules that the Vga controller module, Debounce module and the pixel generation module in the top module to achieve the ping pong game.

* + 1. **BLOCK DIAGRAM:**



* + 1. **VERILOG CODE:**

//TOP MODULE

module top(

input clk\_100MHz, // from Basys 3 input reset, // btnR

input up, // btnU

input down, // btnD

output hsync, // to VGA port output vsync, // to VGA port

output [11:0] rgb // to DAC, to VGA port

);

wire w\_reset, w\_up, w\_down, w\_vid\_on, w\_p\_tick;

wire [9:0] w\_x, w\_y;

reg [11:0] rgb\_reg;

wire [11:0] rgb\_next;

vga\_controller vga(.clk\_100MHz(clk\_100MHz), .reset(w\_reset), .video\_on(w\_vid\_on),

.hsync(hsync), .vsync(vsync), .p\_tick(w\_p\_tick), .x(w\_x), .y(w\_y)); pixel\_gen pg(.clk(clk\_100MHz), .reset(w\_reset), .up(w\_up), .down(w\_down),

.video\_on(w\_vid\_on), .x(w\_x), .y(w\_y), .rgb(rgb\_next)); debounce dbR(.clk(clk\_100MHz), .btn\_in(reset), .btn\_out(w\_reset)); debounce dbU(.clk(clk\_100MHz), .btn\_in(up), .btn\_out(w\_up)); debounce dbD(.clk(clk\_100MHz), .btn\_in(down), .btn\_out(w\_down));

// rgb buffer

always @(posedge clk\_100MHz) if(w\_p\_tick)

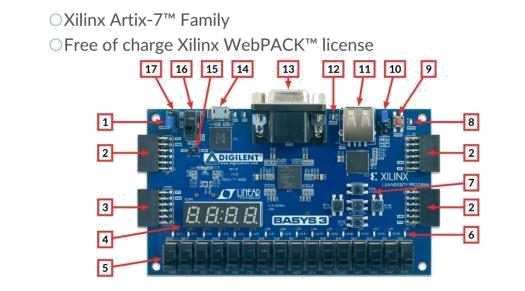
rgb\_reg <= rgb\_next; assign rgb = rgb\_reg;

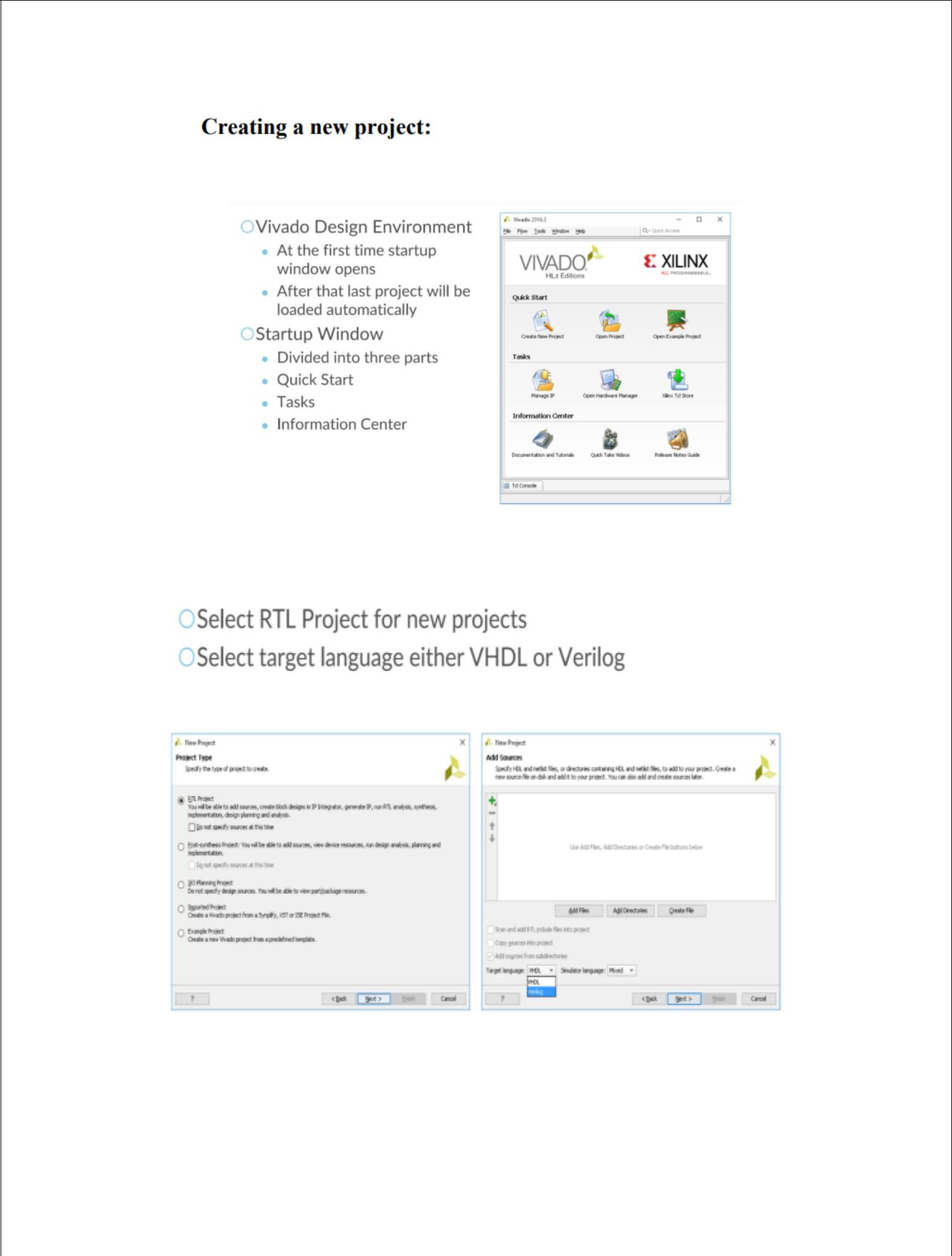
endmodule

# CHAPTER: 5

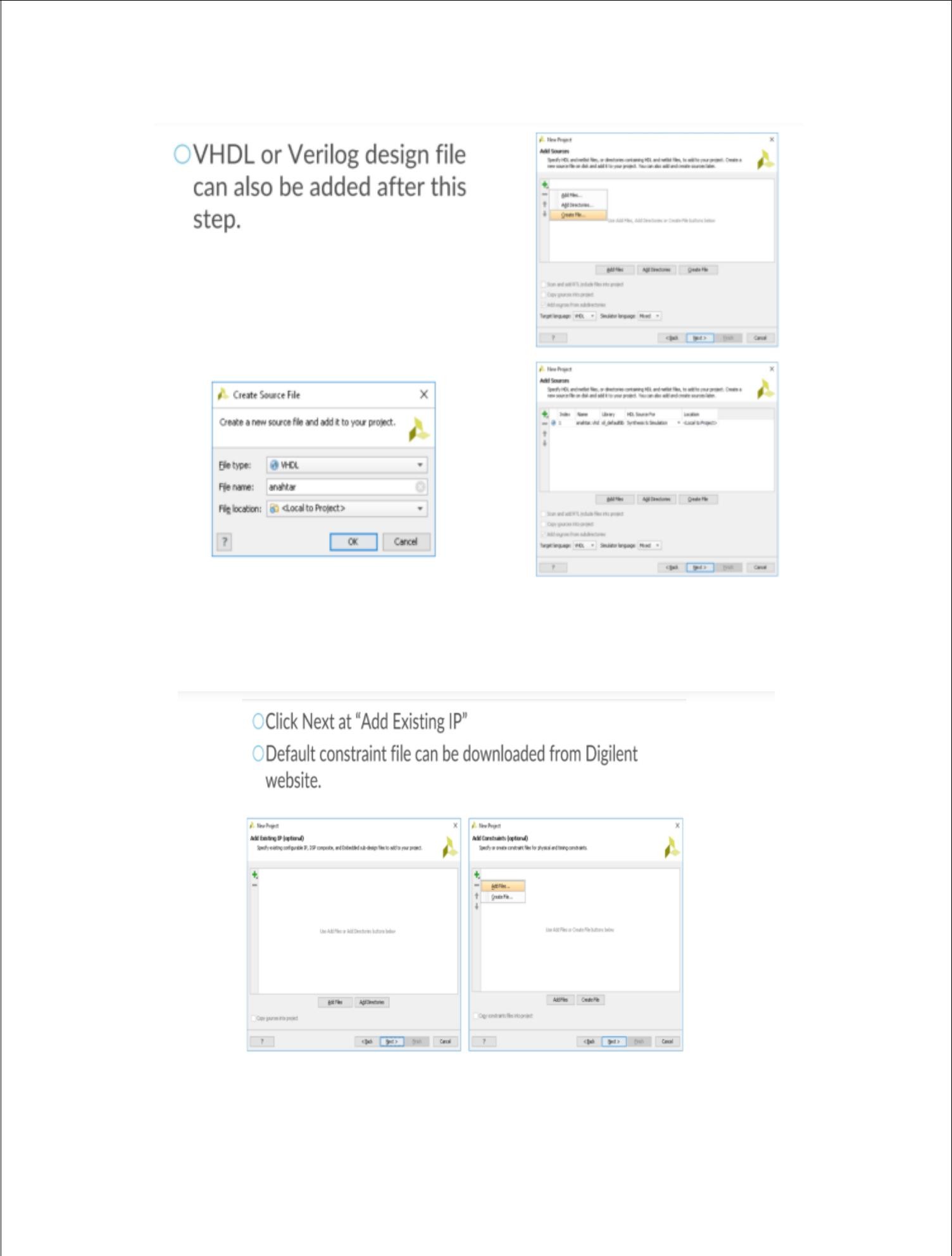
**INTERFACING WITH FPGA**

* 1. **FPGA IMPLEMENTATION:**

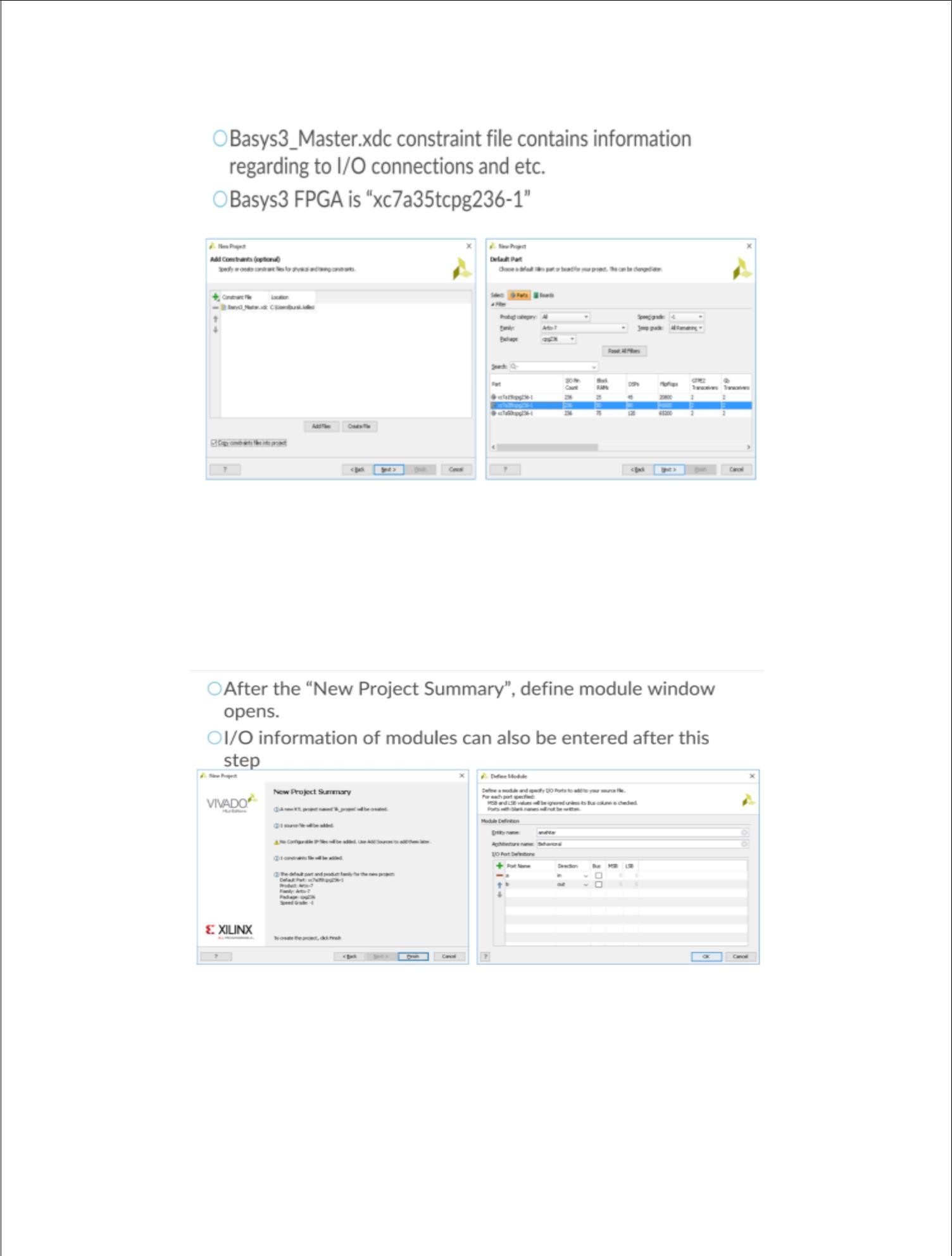




33



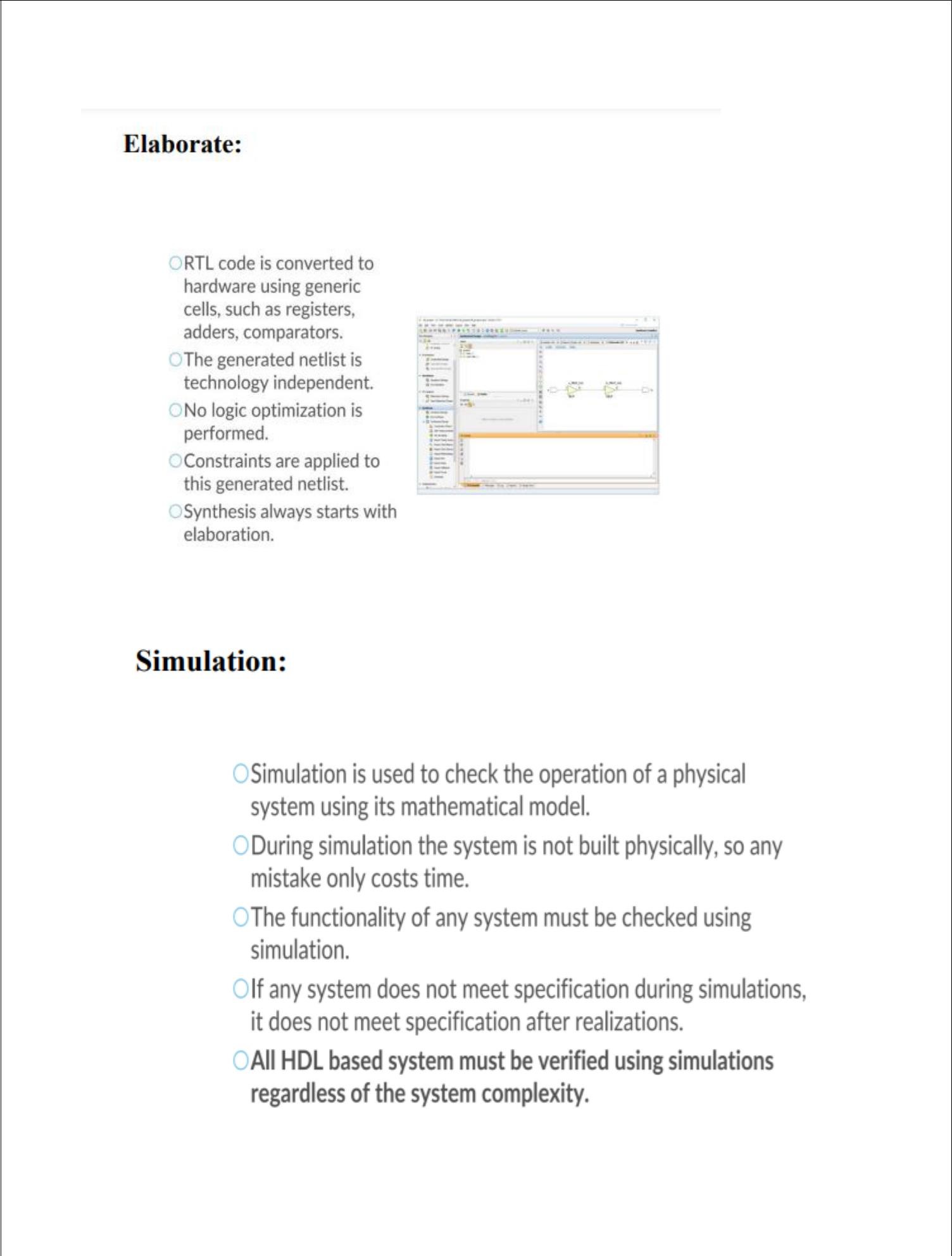
34



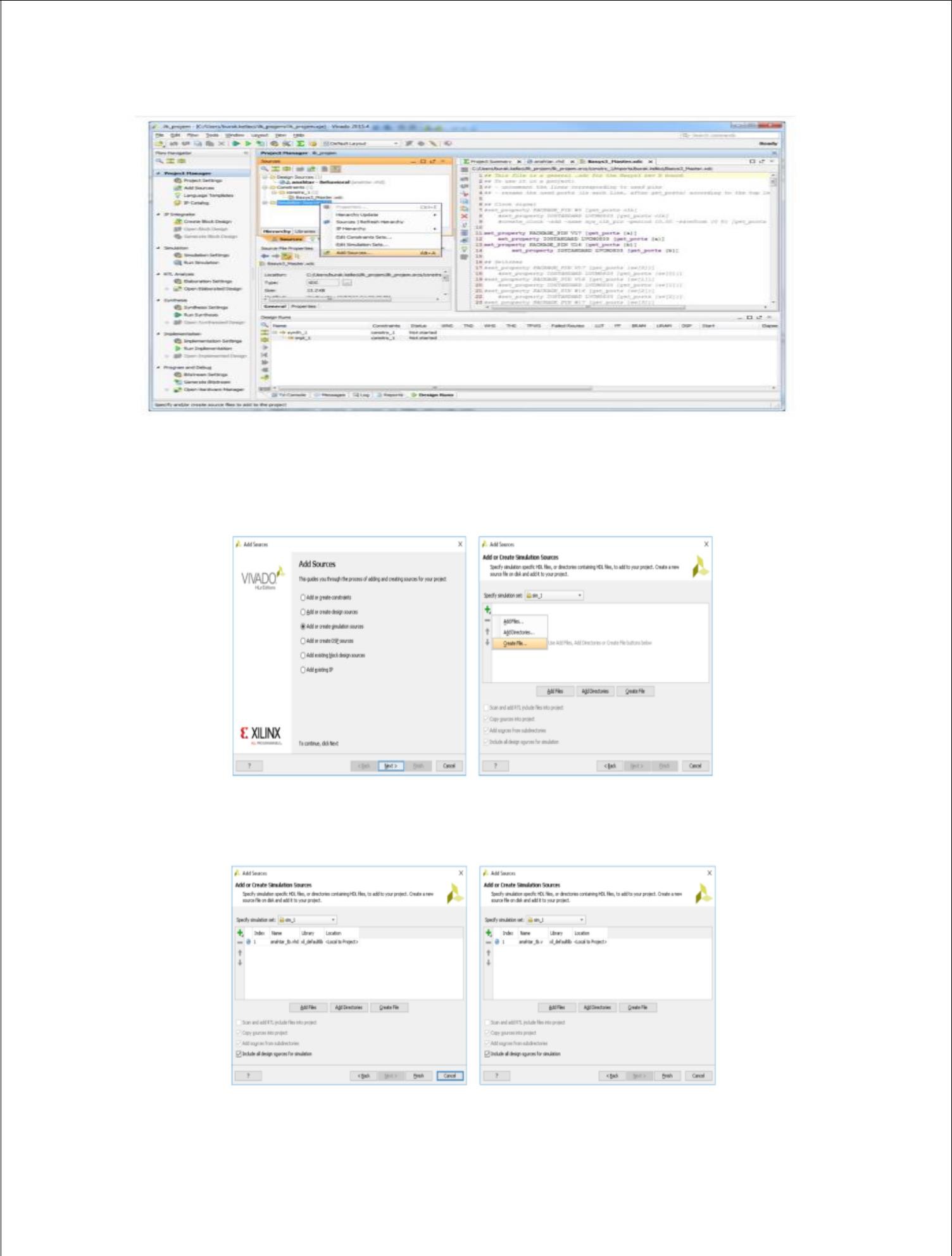
35



36

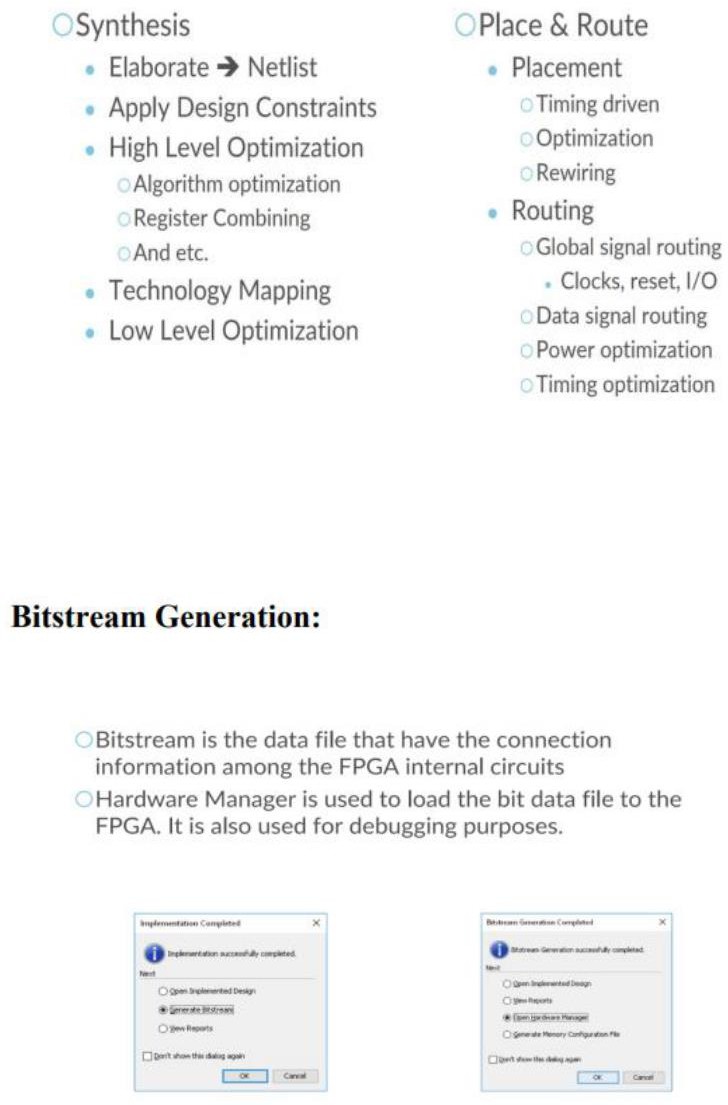


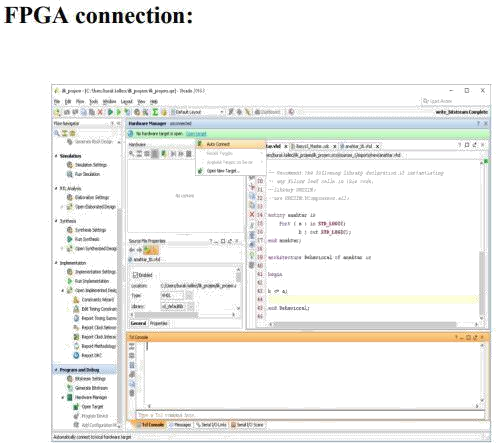
37



38







Below are the simple steps we follow while interfacing with FPGA board:

▶

Open the project that contains the pong game code.

▶

Connect the FPGA board to the PC USB cable.

▶

Click on synthesis.

▶

Click on implementation.

▶

Click on Bit stream generation.

Open the target file.

Connect the VGA controller to the monitor and FPGA board.

Dump the code in FPGA.

* 1. **RESULT:**



Fig: 8

Here we attached the set of pictures of the resultant game in monitor.

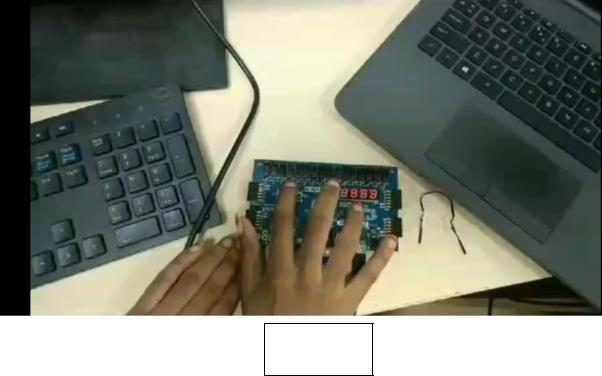


Fig: 6



Fig: 7

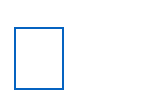
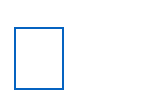
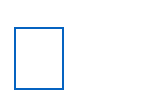
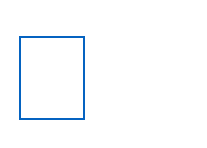
Fig:9

# CHAPTER: 6

* 1. **CONCLUSION:**

In conclusion, the Ping Pong game developed on the FPGA Basys 3 board encapsulates the convergence of hardware and software ingenuity, presenting an immersive gaming experience. Harnessing the FPGA's real-time processing capabilities, the game delivers seamless interaction and fluid gameplay. Its adaptability allows for customizable features, tailoring the experience to diverse player preferences and skill levels. Additionally, the project serves as an educational platform, offering insights into digital design principles and FPGA programming techniques. Through hands-on exploration of hardware/software co-design, students and enthusiasts gain valuable knowledge applicable to various domains, including embedded systems and digital signal processing. Moreover, the scalability and efficiency of the Basys 3 board showcase the versatility of FPGA-based systems in crafting engaging gaming experiences while pushing the boundaries of digital innovation. As FPGA technology continues to evolve, projects like the Ping Pong game exemplify the transformative potential of FPGA-based solutions in shaping the future of interactive entertainment and real-time applications.

* 1. **REFERENCES:**



<https://www.edaboard.com/threads/>

<https://pubmed.ncbi.nlm.nih.gov/23845963/> <https://opencores.org/projects/vga_lcd> <https://github.com/FedorChervyakov/de10lite-hello-vga>.