SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA

INDORE

Session: 2020-21

A Major Project Report on

"PONG Game on FPGA"

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

Guided By:

Mr. Dilip Mandloi

Submitted By:

Rutuja Gogulwar

17010BTEC01840

Department of Electrical & Electronics Engineering
Shri Vaishnav Institute of Technology & Science, Indore

SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA
INDORE

RECOMMENDATION

The project report entitled, **"PONG Game on FPGA"**, submitted to Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore, MP by **Rutuja Gogulwar** during the academic year 2020-21, as a partial fulfillment for the award of degree of the **Bachelor of Technology** in **Electronics & Communication,** is a record of their own work carried out by them under our direct supervision, in the **Department of Electrical & Electronics Engineering, SVITS, SVVV, Indore.** The work contained in the report is a satisfactory account of their project work and is recommended for the award of the degree.

Mr. Dilip Mandloi	**Dr. Namit Gupta**
Project Guide	**Head of Department**

SHRI VAISHNAV VIDYAPEETH VISHWAVIDYALAYA
INDORE

CERTIFICATE

This is to certify that the report entitled **"PONG Game on FPGA"** is a bonafide record of the minor project done by **Rutuja Gogulwar** Enrollment No: **17010BTEC01840** in partial fulfillment of the requirements for the award of degree of **Bachelor of Technology** in **Electronics and Communication Engineering** from Shri Vaishnav Institute of Technology and Science of Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore for the academic year 2020

_____ **_____**

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

We would like to take this opportunity to express our extreme gratitude towards our project guide **Mr. Dilip Mandloi**, Department of Electronics Engineering for his invaluable guidance, advice and support throughout the project. His motivation and help have been a source of great inspiration to us.

We are also grateful to our & Project Coordinator **Mr. Preet Jain,** Associate Professor in our department providing us adequate facilities because of which my project has been successful.

We are very indebted to **Dr. Namit Gupta, Director SVITS** who motivated us to improve the quality of our project and extended all supports required at institution level.

Last but not the least we are also thankful to all faculty & staff members of our department for corporation extended in completion of our project.

ABSTRACT

This project is the implementation of Pong Game on VGA monitor. *Pong* is a table tennis themed video game, featuring simple two-dimensional graphics. It was one of the earliest arcade video games. *Pong* is a two-dimensional game that simulates table tennis.

The game is implemented using FPGA development board. FPGA is a technology in which we can design any digital system by programming the CLBs in the FPGA. A FPGA kit act as any digital system based on our program. Hence, FPGAs can become video generators easily, so we are designing two player pong game using the FPGA kit. FPGA can be a great technology to use as we can design any digital circuit inside an FPGA, hence no external circuitry is required once we design circuits inside it.

The result is the exact recreation of the original pong game. Results show that, by programming a single chip according to our requirement, we can reduce the requirement of external circuitry. To implement the pong game, we only used FPGA development board, a keyboard and a VGA monitor.

The result suggest that the FPGA is a powerful technology. It can be reprogrammed until we achieve final bug free circuit. We can prototype the design on FPGA before actual manufacturing of IC. It can cut the cost as we can reprogram as many times as needed unlike other technologies.

CONTENTS

LIST OF ABBREVIATIONS

1. FPGA : Field Programmable Gate Array
2. PLD : Programmable Logic Devices
3. CPLD : Complex Programmable Logic Device
4. ASIC : Application Specific Integrated Circuit
5. CLB : Configuration Logic Block
6. HDL : Hardware Description Language
7. RTL : Register-Transfer Level
8. VGA : Video Graphics Array
9. UART : Universal Asynchronous Receiver Transmitter
10. ASCII : American Standard Code for Information Interchange
11. SSD : Seven Segment Display

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Introduction:

Love for table tennis comes only while playing it rather than watching it. In 1972, the first arcade video game, "*Pong*" was released. It was the first commercially successful video game. Even today, it is quite popular game. Pong is a two-dimensional table tennis themed game, in which two players compete against each other. Each player owns a paddle using which they can pass ball to the opponent. The aim of game is to score 5 points before the opponent. A player scores 1 point when its opponent fails to pass the ball i.e. opponent's paddle misses the ball.

This project is the recreation of the same game on VGA monitor, using a FPGA development kit. FPGA is a type of complex PLD, which can be programmed using HDLs. See Appendix: Programmable Logic Devices. Initially pong was built using TTL logic. Atari's Arcade Pong PCB contained 66 IC's. Gates and flip flops of every kind, a pair of 555 timer IC's and some transistors. It was simply glue logic TTL circuit and forgoes microprocessor and software-controlled video games. Later, ASICs were fabricated to design whole system on single chip. With the advancement in technology, FPGAs came into role. FPGAs became popular as they were fully programmable and circuit can be redesigned numerous times until desired output is achieved. We have designed Pong on FPGA, so no external circuitry was required.

A dedicated circuit for the gameplay is designed inside the FPGA. The system is divided in several modules. A bottom-up approach is used to design the system. We have used Verilog HDL to program our FPGA chip. We have connected output of the designed circuit to VGA pins, seven segment display available on the development board. Push buttons mounted on the development board are used as input to control the player 1 paddle. A UBS keyboard is connected to USB port of development board which would control the player 2 paddle. A LCD monitor with VGA input will be used to display the graphics of the game. The VGA driver works on 25MHz clock frequency with display resolution of 640x480 pixels.

1.2 Problem Definition:

In recent years, almost every electronic device we see comprises of digital circuits. Digital circuits are faster, more reliable than analog circuits. A discrete circuit is constructed of components which are manufactured separately. Later, these components are connected together by using conducted wires on a circuit board or a printed circuit board. Assembling and wiring of all individual discrete components take more time and occupies a larger space required. Replacement of a failed component is complicated in an existed circuit or system. All the elements are connected using soldering process so, that may have caused less reliability. To overcome these problems of reliability and space conservation, integrated circuits are developed. Now, digital systems can be fabricated on a single chip. The circuits designed inside a single chip to perform a specific application are called ASIC. Fabricating ASICs consume large amount of time, money and resources. So, to design bug free ASIC, a technology is needed where we can alter the design as many times as possible without wasting resources.

1.3 Outline:

The purpose of this project is to prototype design of Pong Game on a FPGA. Prototyping the design on FPGA will reduce consumption of designing time, and resources.

We referred to various papers and projects on Pong and developed our project using the knowledge gained form these references.

Through this project we accomplished designing circuit for Pong game on FPGA. The circuit is designed using Verilog. First, we designed submodules and finally created top entity by instantiating submodules.

The working of project was evaluated by verifying functional simulation on ModelSim. Verifying the design by simulation saved our time as by analyzing we can see if the functionality of design is achieved or not.

Finally, we arrived at the concluded the whole project and discussed future scope of work.

CHAPTER 2

LITERATURE OVERVIEW

Before working on the project we reviewed various paper, reports and journals related to this project. The creation of games was always in the mind of the engineers. From the start of engineering scientists always thought about how to create something to entertain themselves, not only the required research work. The beginning of the electronics gave big opportunities to the engineers to create games. The first games were also blinking lights and LEDs, but the real games could be made after the introduction of the displays. One of the first games created was the predecessor of the Pong game, which original name is Tennis For Two, developed on oscilloscope and created by William Higinbotham in 1958. On oscilloscope there were created two paddles and a ball, which could be hit by the paddles, if the ball was missed, the other player got a point.

Pong and other early games were implemented largely with discrete TTL chips, hence my choice of an FPGA. By contrast, most later games were processor-based and have been successfully emulated in software using an instruction-set simulator interacting with a high level ad hoc simulator for the video hardware. Atari's Arcade Pong PCB contained 66 IC's. Gates and flip flops of every kind, a pair of 555 timer IC's and some transistors. It was simply glue logic TTL circuit and forgoes microprocessor and software-controlled video games.

With advancement in technology, ASICs were designed to conserve space power and increase reliability of design. FPGA can be used to prototype designs before fabricating.

CHAPTER 3

PROPOSED METHODOLOGY

3.1 Project Overview

Pong Game circuit is designed by creating various submodules and then instantiating them in top level module. Whole design is divided in 10 modules namely: UART receiver, Debounce button, Sync pulse generator, Position counter, VGA driver, Paddle control, Ball control, Seven segment control, Display score, Game FSM. These 10 modules are instantiated in top level module, Pong Top.

UART receiver is used to establish communication between the FPGA and keyboard. Debounce button is used to nullify the effect of bouncing of mechanical switches. Sync pulse generator generates the horizontal synchronization (Hsync) and vertical synchronization (Vsync) pulses required to drive VGA monitor. Position counter module helps to keep track of where we are in the array of pixels. VGA driver provides output which is fed to the VGA monitor to display graphics. Paddle control module is responsible for the movement of paddle on screen according to user input. Ball control module controls the movement of ball in the game. Display score module encodes binary number to seven segment input. Game FSM module defines states of game and control the gameplay.

3.2 Hardware Details

3.2.1 Hardware Requirements

Sr. No.	Component	Specification
1.	FPGA Development Board	Altera DE0
2.	Monitor	LCD/CRT with VGA input
3.	Keyboard	USB port

Table 1

3.2.2 Component Description

1. *Altera DE0 development board:*

   The DE0 Development and Education board is designed in a compact size with all the essential tools for novice users to gain knowledge in areas of digital logic, computer organization and FPGAs.

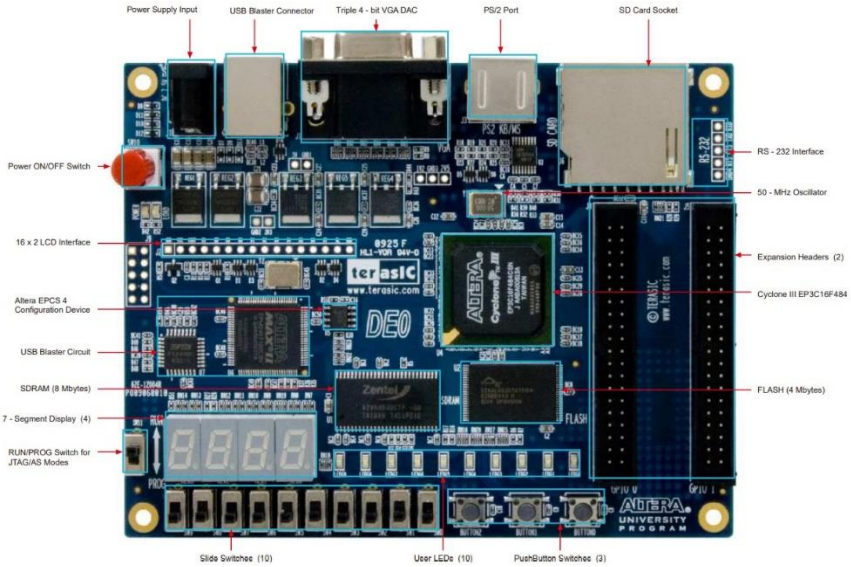


Figure 1

The following hardware is provided on the DE0 board:

FPGA device	Cyclone III EP3C16F484
Serial Configuration device	Altera EPCS4
USB Blaster	JTAG and Active Serial (AS) programming modes
SDRAM	8-Mbyte
Flash memory	4-Mbyte
SD Card socket	SPI and SD 1-bit mod SD Card access
Pushbutton switches (3)	Normally high
Slide switches (10)	UP/$\overline{\text{DOWN}}$

Green user LEDs (10)	Active high
Clock input	50-MHz oscillator
VGA output	4-bit resistor-network DAC
Serial ports	RS-232 port PS/2 port
Expansion headers (2)	40 pins

Table 2

FPGA device:

Cyclone III device family offers a unique combination of high functionality, low power and low cost. Based on Taiwan Semiconductor Manufacturing Company (TSMC) low-power (LP) process technology, silicon optimizations and software features to minimize power consumption, Cyclone III device family provides the ideal solution for your high-volume, low-power, and cost-sensitive applications.

Cyclone IIII 3C16 FPGA technical specifications:

Logic Elements	15,408
Number of M9K Blocks	56
Total RAM Bits	504K
18 x 18 Multipliers	56
PLLs	4
Global Clock Networks	20
Maximum User I/Os	346
Package	FineLine BGA 484-pin

Table 3

2. *LCD Monitor:*

A computer monitor is an electronic device that shows pictures for computers. Monitors often have higher display resolution than televisions. A high display resolution makes it easier to see smaller letters and fine graphics. A liquid-crystal display (LCD) is a flat-panel display or other electronically modulated optical device that uses the light-modulating properties of liquid crystals combined with polarizers. Liquid crystals do not emit light directly, instead using a backlight or reflector to produce images in color or monochrome. It works similar as CRT monitor. The LCD monitor, the most common kind of flat panel display. It is a newer technology than CRT. LCD monitors use much less desk space, are lightweight and use less electricity than CRT.

Figure 2

3. *USB Keyboard:*

A computer keyboard is a typewriter-style devicewhich uses an arrangement of buttons or keys to act as mechanical levers or electronic switches. Keyboard keys (buttons) typically have a set of characters engraved or printed on them, and each press of a key typically corresponds to a single written symbol. However, producing some symbols may require pressing and holding several keys simultaneously or in sequence. In normal usage, the keyboard is used as a text entry interface for typing text, numbers, and symbols into application software.

Figure 3

3.3 Software Details

3.3.1 Softwares Required

1. Quartus II : For programming FPGA.

3.3.2 Software Description

1. *Quartus II:*

Altera Quartus II is programmable logic device design software produced by Intel. The Altera Quartus II design software is a multiplatform design environment that easily adapts to your specific needs in all phases of FPGA and CPLD design. Quartus II software delivers the highest productivity and performance for Altera FPGAs, CPLDs, and HardCopy ASICs. Quartus II enables analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus Prime includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation.

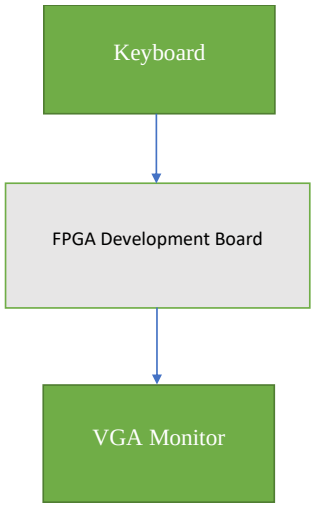
Figure 4

CHAPTER 4

BLOCK DIAGRAM

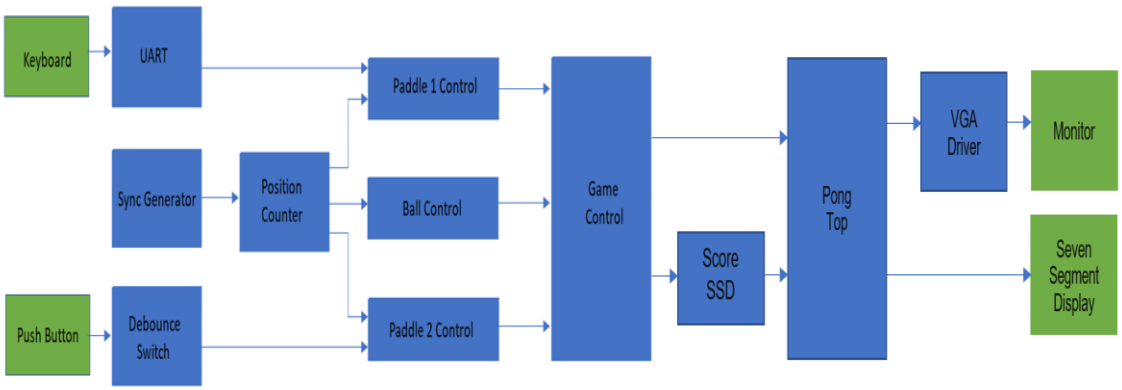
4.1 Block Diagram

Following is a basic block diagram of the project. A circiut is esigned inside the FPGA chip.



Basic Block Diagram

The detailed block diagram of circuit designed inside FPGA chip is shown below. It contains the blocks for every submodule involved in the design.



Detailed Block Diagram

4.2 Block Diagram Description

- **UART:** UART is a serial communication protocol with help of which, two devices can communicate without clock signal. See Appendix-B: UART. UART reciever module recieves data from the keyboard, converts input 8-bit serial data into 8 byte data and stores it.

- **Debounce Switch:** When a mechanical switch is toggled they create physical contact to terminal. The mechanical switch tend to bounce. See Appendix-B: Bouncing of mechanical switches. To filter the glitchy button input debounce switch module is designed, which gives filtered output.

- **Sync Generator:** This module generates horizontal and vertical synchronization pulses required to drive VGA display.

- **Position Counter:** To keep the track of where are we present in horizontal and vertical pixels, this module is designed. It will be required ti determine whether the pixel value should on or off.

- **Paddle 1 control:** This module controls the position of paddle of player 1 based on the key pressed on the keyboard. It recieves input from UART receiver and decides the position of paddle.

- **Paddle 2 control:** This module works similar to paddle 1 control except of the input. It recieves input from the debounce switch and based on which switch is pressed, it will move paddle upwards or downwards.

- **Ball Control:** This module keeps track of the ball location in play area. The row/col that is being drawn on the VGA display is sent as an input to this module. If the current active pixel is the same pixel as where the ball is located, the output becomes a 1. Otherwise it's a 0.

- **Game Control:** This module is created by designing finite state machine which would control the gameplay. It will keep track of who scores the point and when the game will end.

- **Score SSD:** This module is responsile for displaying the scores of both the players on the seven segment display present on the FPGA board. It recieves score of player as input from the game control module and displays it on SSD.

- **VGA driver:** This module modifies the sync pulses to add in the front and back porches required to properly drive the VGA display. The output of this module should the actual VGA signals that go right to the VGA pins.

- **Pong Top:** All the submodules designed are instantiate in top entity to establish proper connection between them.

CHAPTER 5

CIRCUIT DIAGRAM

5.1 Circuit Diagrams

1. UART Receiver:

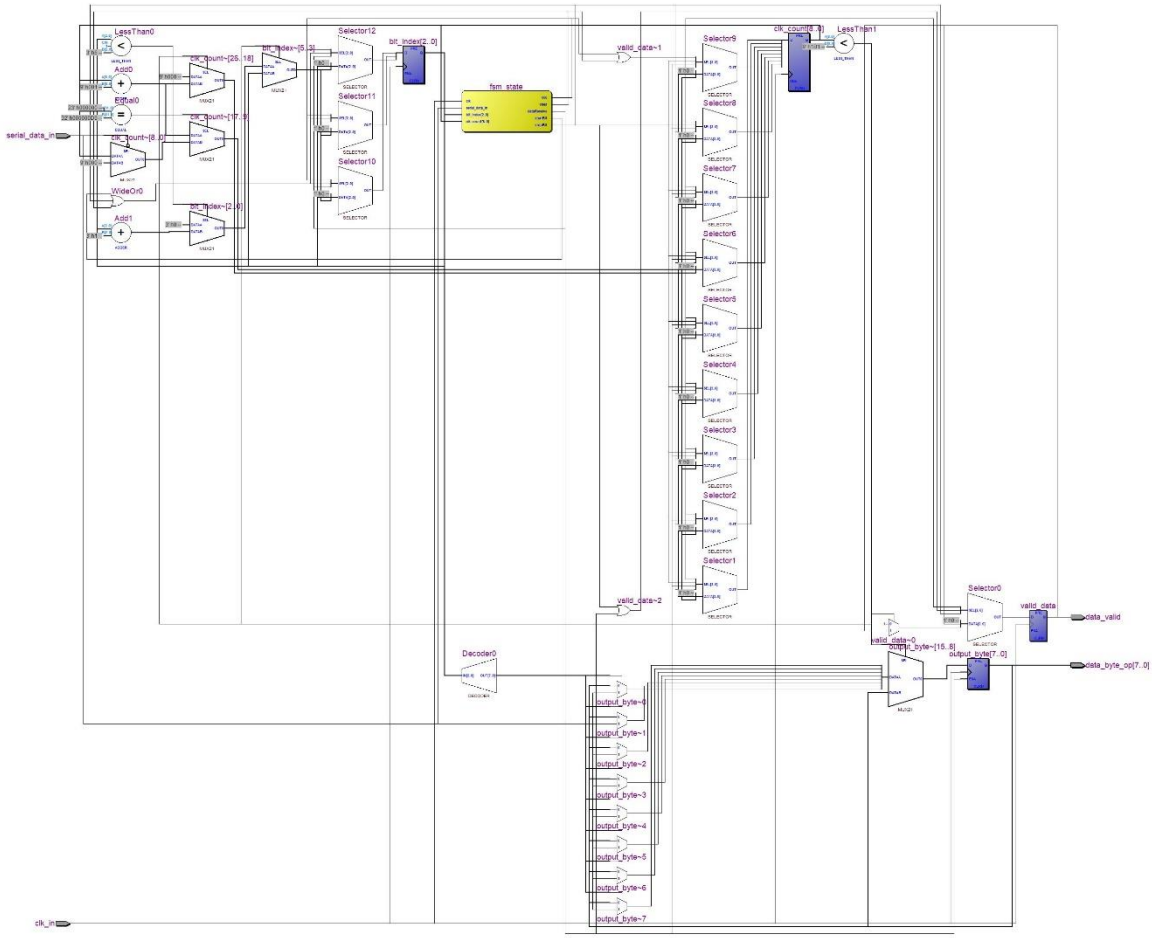


Figure 5

2. Debounce Switch:

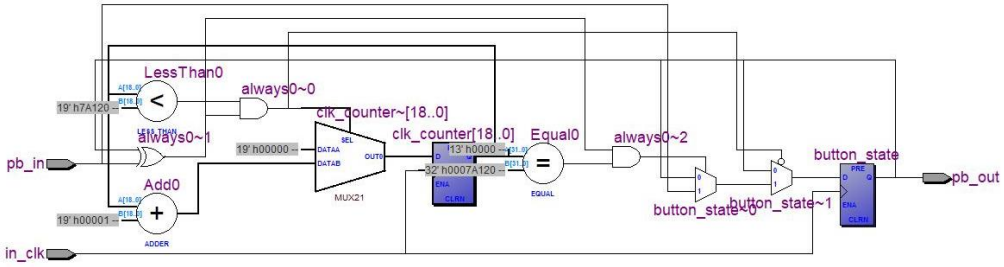


Figure 6

3. Paddle 1 Control:

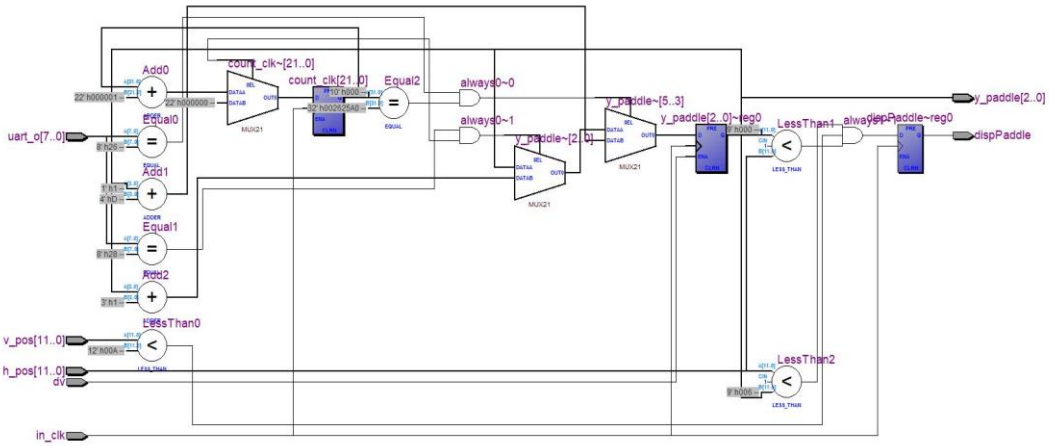


Figure 7

4. Paddle 2 Control:

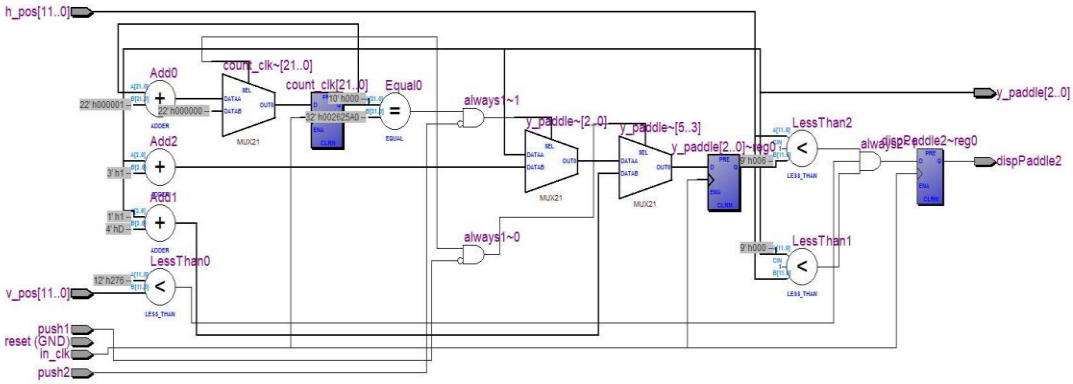


Figure 8

5. Ball Control:

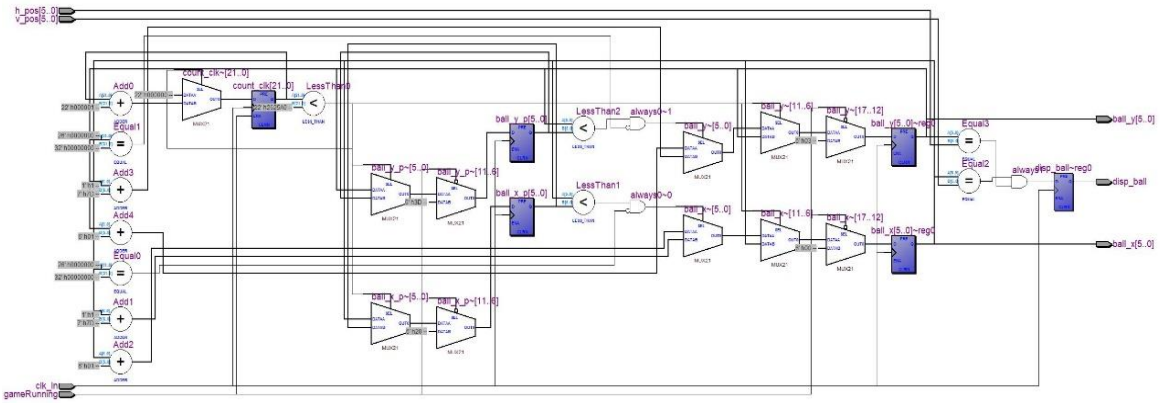


Figure 9

6. VGA Controller:

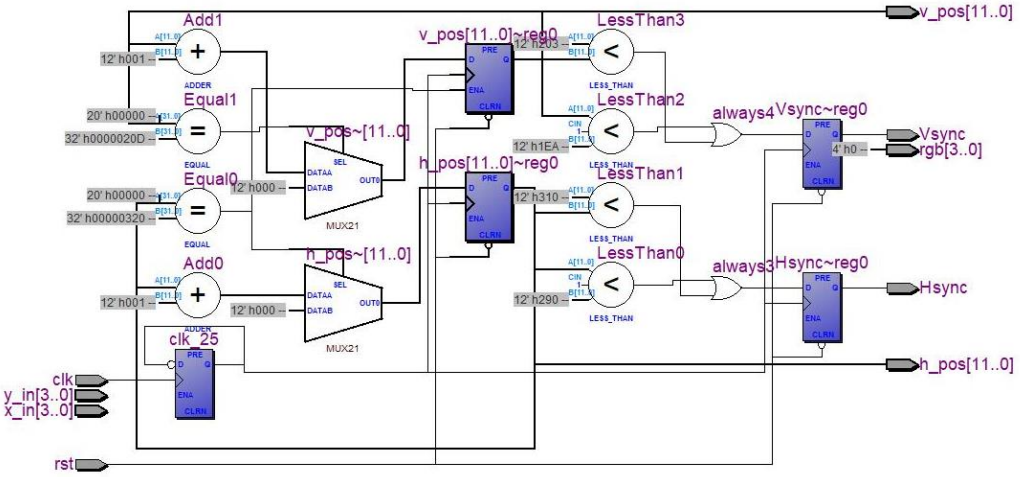


Figure 10

7. Game FSM:

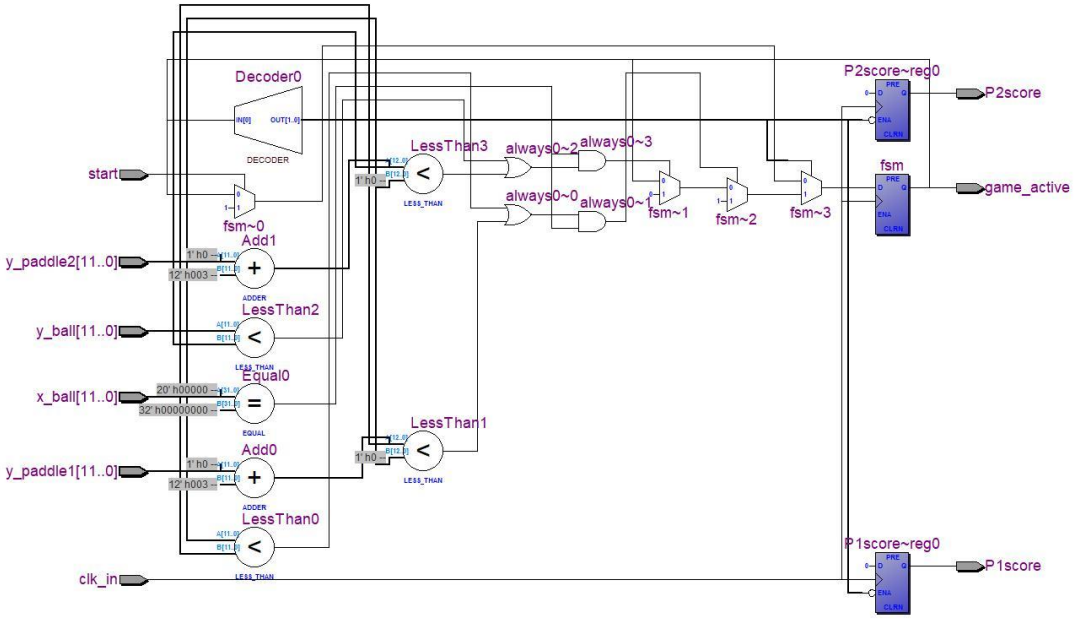


Figure 11

8. Display Score SSD:

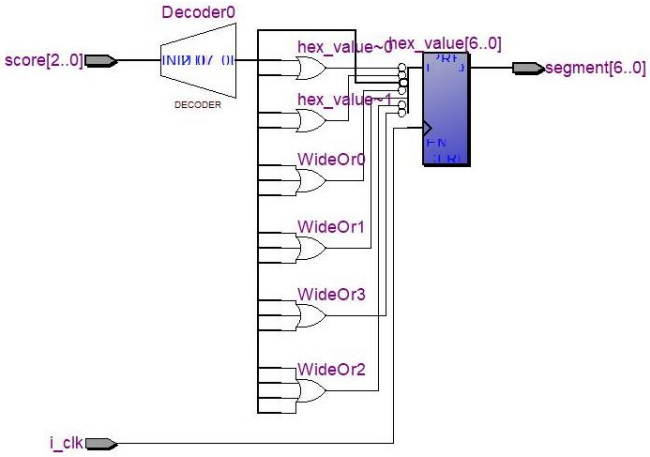


Figure 12

9. Pong TOP:

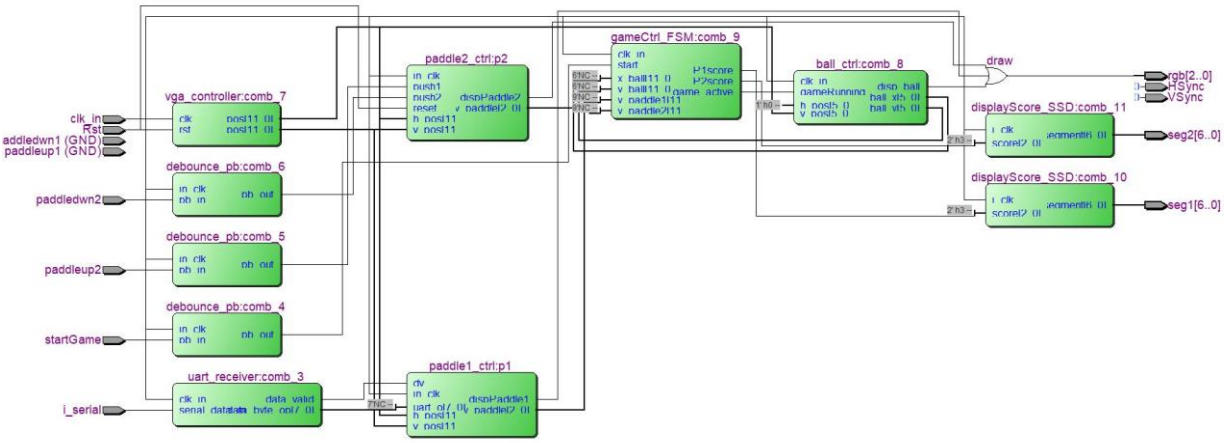


Figure 13

CHAPTER 6

RESULT AND DISCUSSION

6.1 Result

Our project resulted in a working implementation of the game Pong. The paddles can be seen on either side of the screen, with a square ball bouncing between them. Scores of each player are displayed on the seven segment displays present on the board. When either player scores 5 points before the opponent, that player will win the game.

6.2 Analysis

The above discussed circuit was implemented on the FPGA development board. During the process of designing circuit, many times the required output was not achieved in the circuit. So, we redesigned the circuit to achieve the required functionality. It is clear that, while working on FPGA, we can redesign circuits as many times as required.

We reviewed different methods to implement the game but in other methods, to design the game, large number of components, ICs are required which need to be connected together on PCB. Unlike other methods, we were able design game on single chip.

As discussed in above section, the required functionality is satisfied in the model built. Though the project works as expected, many modifications can be done to improve our project. Several functionalities can be added to make the game more interesting and interactive. We can easily add these functionalities in game as the FPGA is reprogrammable.

CHAPTER 7
CONCLUSION

7.1 Conclusion

As discussed in above section, we recreated the Pong game on FPGA. Our aim was to reconstruct one of the most popular games of computer game history. The Pong game was one of the first commercially successful electronic games which made the basis of computer games history. It was a good decision to design it on FPGA because we had a suitable development the board which has a VGA port for interfacing a computer monitor, seven segment displays to display scores and push buttons for controls, and USB blaster for USB communication. The advantage of designing this on FPGA was that the entire design was embedded inside the FPGA chip, because of which we din not have other dependencies, even OS dependency was ceased.

7.2 Future Scope of Work

- Score can be displayed on screen itself
- Speed ball can be changed dynamically with game running.
- Size of paddle can be set according to player's choice
- Colors of paddle and ball can be changed dynamically
- After end of game, interesting graphics can be displayed
- Various game modes can be added like single-player mode, two-player mode, three-player mode and four-player mode

REFERENCES

[1] S. A. Edwards, "Reconstructing Pong on an FPGA," Department of Computer Science, Columbia University, 2012.

[2] J. C. M. M. J. M. R.-A. C. P.-O. T.-A. M. A. A. F. d. Carlos Alberto Ramos-Arreguína, "FPGA Open Architecture Design for a VGA Driver," SciVerse science direct, 2012.

[3] A. G. ROLAND SZABÓ, "PONG GAME ON AN FPGA DEVELOPMENT BOARD USING A COMPUTER," *International Journal of Computer Science and Applications*, p. 11, 2015 .

[4] A. G. Roland Szabó, "Pong Game on FPGA with CRT or LCD display and Push Button Controls," in *Federated Conference on Computer Science and Information Systems*, 2014.

[5] Z. Z. X.-h. Y.-b. WANG Hong-bin, "VGA Display Driver design Based on FPGA," *IEEE Computer Society*, p. 6, 2019.

APPENDIX-A

SOURCE CODE

UART Receiver:

```
module uart_receiver(clk_in,serial_data_in,data_valid,data_byte_op);
input clk_in, serial_data_in;
output data_valid;
output [7:0]data_byte_op;

parameter baudRate          = 115200;
parameter clk_per_bit   = 434;                          //clk_in/baudRate=50,000,000/115200

parameter idle                      = 3'b 000;
parameter startBit          = 3'b 001;
parameter stopBit                   = 3'b 010;
parameter dataReceive       = 3'b 011;
parameter clear                     = 3'b 100;

reg [8:0]clk_count          = 0;
reg [2:0]bit_index          = 0;
reg [7:0]output_byte        = 0;
reg [2:0]fsm_state          = 0;
reg valid_data              = 0;

always @(posedge clk_in)
begin
        case(fsm_state)

                idle:
                begin
                        valid_data      <= 1'b 0;
                        clk_count       <= 0;
                        bit_index       <= 0;

                        if(serial_data_in == 0)
                                fsm_state <= startBit;
                        else
                                fsm_state <= idle;
                end

                startBit:
                begin
                        if(clk_count == (clk_per_bit/2))
                        begin
                                if(serial_data_in == 0)
                                begin
```

```
                clk_count <= 0;
                fsm_state <= dataReceive;
            end
            else
                fsm_state <= idle;
        end
        else
        begin
            clk_count <= clk_count + 1;
            fsm_state <= startBit;
        end
end

dataReceive:
begin
        if(clk_count < clk_per_bit-1)
        begin
            clk_count <= clk_count + 1;
            fsm_state <= dataReceive;
        end
        else
        begin
            clk_count                                   <= 0;
            output_byte[bit_index]          <= serial_data_in;

            if(bit_index <= 6)
            begin

                bit_index <= bit_index + 1;
                fsm_state <= dataReceive;
            end
            else
            begin
                bit_index <= 0;
                fsm_state <= stopBit;
            end
        end
end

stopBit:
begin
        if(clk_count < (clk_per_bit-1))
        begin
            clk_count <= clk_count + 1;
            fsm_state <= stopBit;
        end
        else
        begin
            valid_data <= 1'b 1;
            clk_count  <= 0;
```

```
                        fsm_state  <= clear;
                end
        end

        clear:
        begin
                clk_count       <= 0;
                valid_data      <= 1'b 0;
                fsm_state       <= idle;

        end

        default:
                fsm_state <= idle;

    endcase
end

assign data_valid               = valid_data;
assign data_byte_op   = output_byte;

endmodule
```

Debounce Switch:

```
module debounce_pb(in_clk, pb_in, pb_out);
input in_clk;
input pb_in;
output pb_out;

reg button_state                = 1'b 1;                //not pressed
reg [18:0]clk_counter = 0;

parameter clks_ms = 500000;                     //clks cycles to elapse 10ms

always @(posedge in_clk)
begin
        if((pb_in !== button_state) && (clk_counter < clks_ms))
        begin
        clk_counter <= clk_counter + 1;
        end

        else if((pb_in !== button_state) && clk_counter == clks_ms)
        begin
        clk_counter  <= 0;
        button_state <= pb_in;
        end

        else
        begin
        clk_counter <= 0;
        end
end

assign pb_out = button_state;
endmodule
```

VGA Controller:

```
module vga_controller(clk,rst,x_in,y_in,Hsync,Vsync,rgb,h_pos,v_pos);
input clk,rst;
input [3:0]x_in,y_in;
output reg Hsync,Vsync;
output reg [3:0]rgb;
output reg [11:0]h_pos=0, v_pos=0;
integer active_display=0;
reg clk_25=0;

parameter h_display = 640;
parameter h_frontporch = 16;
parameter h_syncpulse = 96;
parameter h_backporch = 48;

parameter v_display = 480;
parameter v_frontporch = 10;
parameter v_syncpulse = 2;
parameter v_backporch = 33;


//clk divider
always @(posedge clk)
begin
	clk_25 <= ~(clk_25);
end

//horizontal position counter
always @(posedge clk_25, negedge rst)
begin
	if(!rst)
		h_pos <= 0;
	else if(h_pos == (h_display + h_frontporch + h_syncpulse + h_backporch))
		h_pos <= 0;
	else
		h_pos <= h_pos+1;
end

//verical position counter
always @(posedge clk_25, negedge rst)
begin
	if(!rst)
		v_pos <= 0;
	else if(h_pos == (h_display + h_frontporch + h_syncpulse + h_backporch))
	begin
		if(v_pos == (v_display + v_frontporch + v_syncpulse + v_backporch))
			v_pos <= 0;
		else
			v_pos <= v_pos+1;
```

```
        end
end

//Horizontal synchronization pulse generator
always @(posedge clk_25, negedge rst)
begin
        if(!rst)
                Hsync <= 0;
        else if((h_pos <= (h_display + h_frontporch)) || (h_pos > (h_display + h_syncpulse
+ h_backporch)))
                Hsync <= 1;
        else
                Hsync <= 0;
end

//Vertical synchronization pulse generator
always @(posedge clk_25, negedge rst)
begin
        if(!rst)
                Vsync <= 0;
        else if((v_pos <= (v_display + v_frontporch)) || (v_pos > (v_display + v_syncpulse
+ v_backporch)))
                Vsync <= 1;
        else
                Vsync <= 0;
end

//active display area
always @(posedge clk_25, negedge rst)
begin
        if(!rst)
                active_display <= 0;
        else if(h_pos <= h_display && v_pos <= v_display)
                active_display <= 1;
        else
                active_display <= 0;
end

endmodule
```

Player 1 paddle control:

```
// ASCII Codes:
// arrow up - (38)00100110
//arrow dwn - (40)00101000

module paddle1_ctrl(in_clk,y_paddle,uart_o,dv,dispPaddle1,h_pos,v_pos);
input in_clk,dv;
input [11:0]h_pos,v_pos;
input [7:0]uart_o;
output reg [2:0]y_paddle;
output reg dispPaddle1;

reg [21:0]count_clk = 0;

parameter paddleHeight = 48;
parameter paddleWidth = 10;

parameter up  = 8'b 00100110;
parameter dwn = 8'b 00101000;
parameter waitCycles = 2500000;

always @(posedge in_clk)
begin
        if(count_clk == waitCycles)
                count_clk <=0;
        else
                count_clk <= count_clk + 1;

        if(dv)
        begin
                if(uart_o == up && count_clk == waitCycles)
                        y_paddle <= y_paddle-1;
                else if(uart_o == dwn && count_clk == waitCycles)
                        y_paddle <= y_paddle+1;
                else
                        y_paddle <= y_paddle;
        end
        else
                y_paddle <= y_paddle;
end

always @(posedge in_clk)
begin
   if (v_pos < paddleWidth && h_pos >= y_paddle && h_pos <= y_paddle +
paddleHeight)
     dispPaddle1 <= 1'b1;
   else
     dispPaddle1 <= 1'b0;
end
endmodule
```

Player 2 paddle control:

```
module paddle2_ctrl(in_clk,reset,push1,push2,y_paddle,dispPaddle2,h_pos,v_pos);
input push1,push2,reset,in_clk;
input [11:0]h_pos,v_pos;
output reg [2:0]y_paddle;
output reg dispPaddle2;

reg [21:0]count_clk = 0;
parameter paddleHeight = 48;
parameter paddleWidth = 10;
parameter screenWidth =640;
parameter screenHeight = 480;
parameter waitCycles = 2500000;

always @(posedge in_clk)
begin
        if(count_clk == waitCycles)
                count_clk <=0;
        else
                count_clk <= count_clk + 1;
end

always @(posedge in_clk)
begin
        if(reset)
                y_paddle <= 480/2;
        if(push1 == 0 && count_clk == waitCycles)           // up
                y_paddle <= y_paddle-1;
        else if(push2 == 0 && count_clk == waitCycles)              // down
                y_paddle <= y_paddle+1;
        else
                y_paddle <= y_paddle;
end

always @(posedge in_clk)
begin
   if ((v_pos > screenWidth-paddleWidth) && (h_pos >= y_paddle) && h_pos <=
y_paddle + paddleHeight)
     dispPaddle2 <= 1'b1;
   else
     dispPaddle2 <= 1'b0;
end

endmodule
```

Ball Control:

```
module ball_ctrl(clk_in,gameRunning,h_pos,v_pos,disp_ball,ball_x,ball_y);
input clk_in,gameRunning;
input [5:0]h_pos;
input [5:0]v_pos;
output reg disp_ball;
output reg [5:0]ball_x = 0;
output reg [5:0]ball_y = 0;

parameter ball_width = 16;
parameter ball_height = 16;
parameter screenWidth = 640;
parameter screenHeight = 480;
parameter waitCycles = 2500000;

reg [5:0]  ball_x_p = 0;
reg [5:0]  ball_y_p = 0;
reg [21:0] count_clk = 0;

always @(posedge clk_in)
begin
	if (!gameRunning)
	begin
		ball_x <= screenWidth/2;
		ball_y <= screenHeight/2;
		ball_x_p <= screenWidth/2 + 1;
		ball_y_p <= screenHeight/2 - 1;
	end

	else
	begin
		if (count_clk < waitCycles)
			count_clk <= count_clk + 1;

		else
		begin
			count_clk <= 0;

			ball_x_p <= ball_x;
			ball_y_p <= ball_y;

			if ((ball_x_p < ball_x && ball_x == screenWidth-1) || (ball_x_p >
			ball_x && ball_x != 0))
				ball_x <= ball_x - 1;

			else
				ball_x <= ball_x + 1;
```

```
                        if ((ball_y_p < ball_y && ball_y == screenHeight-1) || (ball_y_p >
                        ball_y && ball_y != 0))
                                ball_y <= ball_y - 1;
                        else
                                ball_y <= ball_y + 1;
                end
        end
end


always @(posedge clk_in)
begin
        if (v_pos == ball_x && h_pos == ball_y)
                disp_ball <= 1'b1;
        else
                disp_ball <= 1'b0;
end
endmodule
```

Game control FSM:

```
module
gameCtrl_FSM(clk_in,start,x_ball,y_ball,y_paddle1,y_paddle2,game_active,P1score,P2sc
ore);
input clk_in,start;
input [11:0]x_ball,y_ball,y_paddle1,y_paddle2;
output game_active;
output reg P1score, P2score;


parameter screenWidth  = 640;
parameter screenHeight = 480;
parameter max_score = 5;
parameter paddleHeight = 6;

parameter idle    = 3'b000;
parameter gameRunning = 3'b001;
parameter scoreP1 = 3'b010;
parameter scoreP2 = 3'b011;
parameter clear = 3'b100;

reg fsm = idle;


always @(posedge clk_in)
  begin
    case (fsm)
    idle :
    begin
                    P1score =0;
                    P2score =0;
      if (start == 1'b1)
        fsm <= gameRunning;
    end


    gameRunning :
    begin
      if (x_ball == 0 && (y_ball < y_paddle1 ||  y_ball > y_paddle1 + paddleHeight))
        fsm <= scoreP2;

      else if (x_ball == 0 && (y_ball < y_paddle2 ||  y_ball > y_paddle2 + paddleHeight))
        fsm <= scoreP1;
    end

    scoreP1 :
    begin
      if (P1score < max_score)
        P1score <= P1score + 1;
```

```
      else
      begin
        P1score <= 0;
        fsm <= clear;
      end
    end

    scoreP2 :
    begin
      if (P2score < max_score)
        P2score <= P2score + 1;
      else
      begin
        P2score <= 0;
        fsm <= clear;
      end
    end

    clear :
      fsm <= idle;
    endcase
  end

assign game_active = (fsm == gameRunning) ? 1'b1 : 1'b0;
endmodule
```

Display score SSD:

```
module displayScore_SSD(i_clk,score,segment);
input i_clk;
input [2:0]score;
output[6:0]segment;

reg [6:0] hex_value = 7'h 00;

always @(posedge i_clk)
begin
	case(score)
		0 : hex_value <= 7'h 7E;
		1 : hex_value <= 7'h 30;
		2 : hex_value <= 7'h 6D;
		3 : hex_value <= 7'h 79;
		4 : hex_value <= 7'h 33;
		5 : hex_value <= 7'h 5B;
		6 : hex_value <= 7'h 5F;
		7 : hex_value <= 7'h 70;
		8 : hex_value <= 7'h 7F;
		9 : hex_value <= 7'h 7B;
	endcase
end

assign segment[6] = hex_value[6];
assign segment[5] = hex_value[5];
assign segment[4] = hex_value[4];
assign segment[3] = hex_value[3];
assign segment[2] = hex_value[2];
assign segment[1] = hex_value[1];
assign segment[0] = hex_value[0];

endmodule
```

Pong Top Module:

```
module
pongTop(clk_in,Rst,startGame,i_serial,paddleup1,paddledwn1,paddleup2,paddledwn2,rgb,
HSync,VSync,seg1,seg2);
input clk_in,Rst,startGame,i_serial,paddleup2,paddledwn2,paddleup1,paddledwn1;
output HSync,VSync;
output [6:0]seg1,seg2;
output [2:0]rgb;

wire displayBall,displayPaddle1,displayPaddle2;
wire dataValid, dataop;
wire pu2,pd2;
wire GameStart;
wire [11:0]H_pos,V_pos;
wire [2:0]paddleY1, paddleY2;
wire GAMErunning;
wire [5:0]ballX,ballY;
wire [2:0]score_p1,score_p2;
wire [6:0]seg;

uart_receiver
(.clk_in(clk_in),.serial_data_in(i_serial),.data_valid(dataValid),.data_byte_op(dataop));
debounce_pb (.in_clk(clk_in), .pb_in(startGame), .pb_out(GameStart));
debounce_pb (.in_clk(clk_in), .pb_in(paddleup2), .pb_out(pu2));
debounce_pb (.in_clk(clk_in), .pb_in(paddledwn2), .pb_out(pd2));
vga_controller
(.clk(clk_in),.rst(Rst),.Hsync(Hsync),.Vsync(Hsync),.h_pos(H_pos),.v_pos(V_pos));
paddle1_ctrl
p1(.in_clk(clk_in),.y_paddle(paddleY1),.uart_o(dataop),.dv(dataValid),.dispPaddle1(displa
yPaddle1),.h_pos(H_pos),.v_pos(V_pos));
paddle2_ctrl
p2(.in_clk(clk_in),.reset(Rst),.push1(pu2),.push2(pd2),.y_paddle(paddleY2),.dispPaddle2(
displayPaddle2),.h_pos(H_pos),.v_pos(V_pos));
ball_ctrl
(.clk_in(clk_in),.gameRunning(GAMErunning),.h_pos(H_Pos),.v_pos(H_pos),.disp_ball(d
isplayBall),.ball_x(ballX),.ball_y(ballY));
gameCtrl_FSM
(.clk_in(clk_in),.start(GameStart),.x_ball(ballX),.y_ball(ballY),.y_paddle1(paddleY1),.y_p
addle2(paddleY2),.game_active(GAMErunning),.P1score(score_p1),.P2score(score_p2));
displayScore_SSD (.i_clk(clk_in),.score(score_p1),.segment(seg1));
displayScore_SSD (.i_clk(clk_in),.score(score_p2),.segment(seg2));

assign draw = displayBall | displayPaddle1 | displayPaddle2;
assign rgb = draw ? 3'b111 : 3'b000;

endmodule
```

APPENDIX-B

Programmable Logic Devices

An IC that contains large numbers of gates, flip-flops, etc. that can be configured by the user to perform different functions is called a Programmable Logic Device (PLD). The internal logic gates and/or connections of PLDs can be changed/configured by a programming process. One of the simplest programming technologies is to use fuses. In the original state of the device, all the fuses are intact. Programming the device involves blowing those fuses along the paths that must be removed in order to obtain the particular configuration of the desired logic function. PLDs are typically built with an array of AND gates (AND-array) and an array of OR gates (OR-array).

There are three fundamental types of standard PLDs: PROM, PAL, and PLA. A fourth type of PLD, is the Complex Programmable Logic Device (CPLD) e.g. Field Programmable Gate Array (FPGA).

The FPGA consists of 3 main structures:

1. Programmable logic structure

   The programmable logic structure FPGA consists of a 2-dimensional array of configurable logic blocks (CLBs). Each CLB can be configured (programmed) to implement any Boolean function of its input variables. Typically, CLBs have between 4-6 input variables. Functions of larger number of variables are implemented using more than one CLB. In addition, each CLB typically contains 1 or 2 FFs to allow implementation of sequential logic. Large designs are partitioned and mapped to a number of CLBs with each CLB configured (programmed) to perform a particular function. These CLBs are then connected together to fully implement the target design. Connecting the CLBs is done using the FPGA programmable routing structure.
2. Programmable routing structure

   To allow for flexible interconnection of CLBs, FPGAs have 3 programmable routing resources:

   1. Vertical and horizontal routing channels which consist of different length wires that can be connected together if needed. These channels run vertically and horizontally between columns and rows of CLBs as shown in the Figure.

2. Connection boxes, which are a set of programmable links that can connect input and output pins of the CLBs to wires of the vertical or the horizontal routing channels.

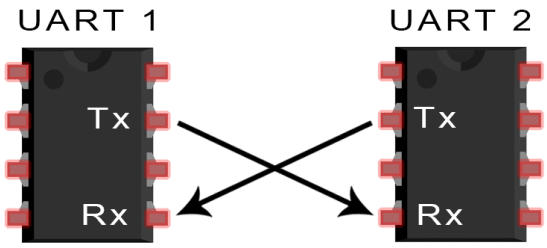
3. Switch boxes, located at the intersection of the vertical and horizontal channels. These are a set of programmable links that can connect wire segments in the horizontal and vertical channels.

3. Programmable Input/Output (I/O)

These are mainly buffers that can be configured either as input buffers, output buffers or input/output buffers. They allow the pins of the FPGA chip to function either as input pins, output pins or input/output pins.

UART

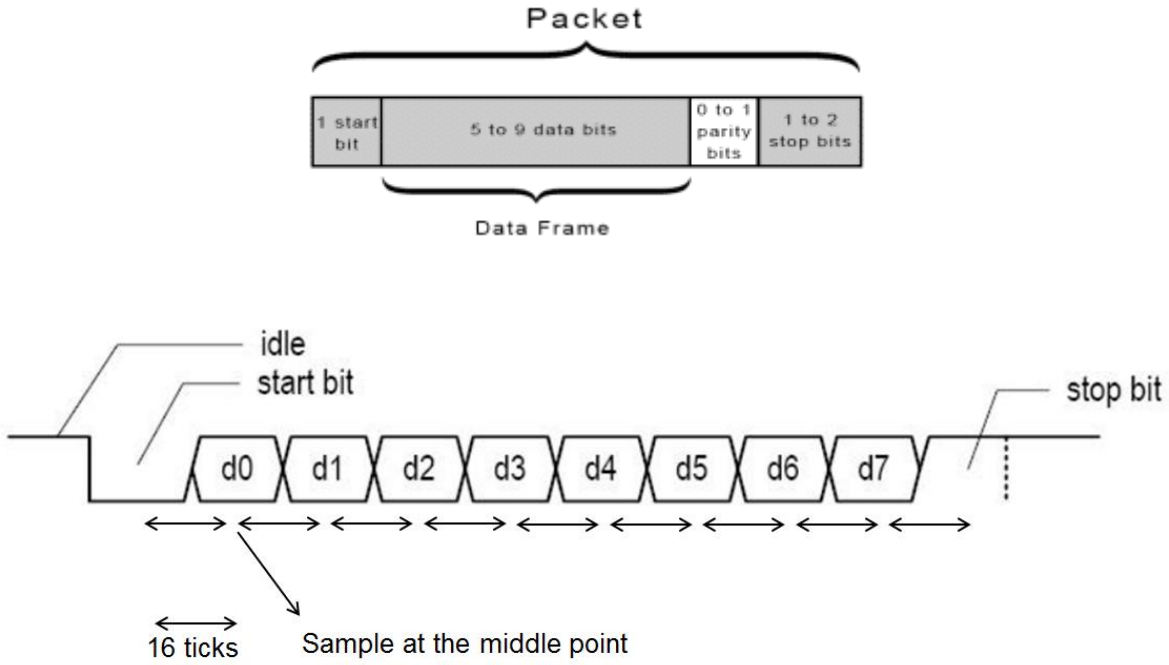
In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:



UARTs transmit data *asynchronously*, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

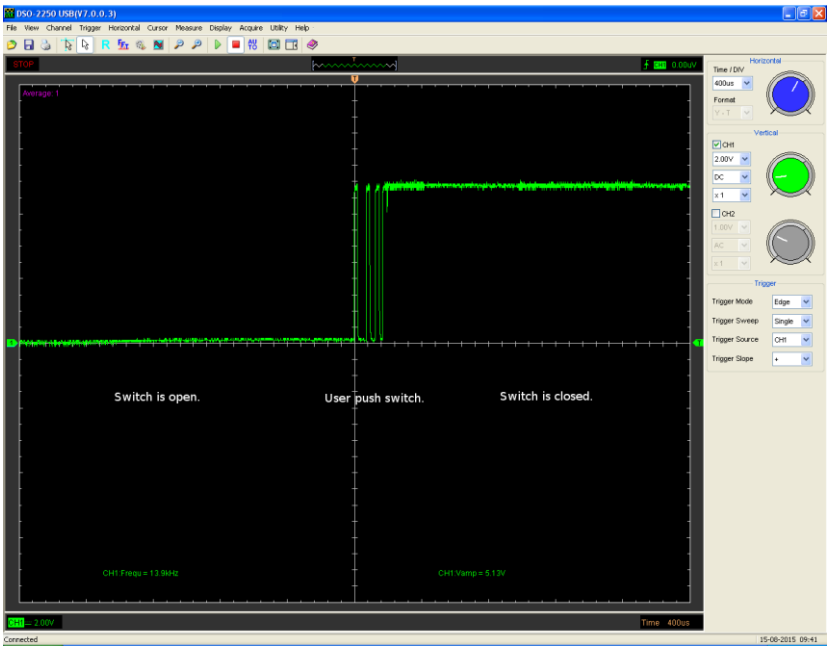
When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the *baud rate*. Baud rate is a measure of the speed of data

transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off. UART transmitted data is organized into *packets*. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional *parity* bit, and 1 or 2 stop bits:



Bouncing of mechanical switches

When you push a button, press a micro switch or flip a toggle switch, two metal parts come together. For the user, it might seem that the contact is made instantly. That is not quite correct. Inside the switch there are moving parts. When you push the switch, it initially makes contact with the other metal part, but just in a brief split of a microsecond. Then it makes contact a little longer, and then again, a little longer. In the end the switch is fully closed. The switch is bouncing between in-contact, and not in-contact. "When the switch is closed, the two contacts actually separate and reconnect, typically 10 to 100 times over a period of about 1ms." ("The Art of electronics", Horowitz & Hill, Second edition, pg 506.) Usually, the hardware works faster than the bouncing, which results in that the hardware thinks you are pressing the switch several times. The hardware is often an integrated circuit. The following screenshots illustrates a typical switch bounce, without any sort of bounce control:



VGA

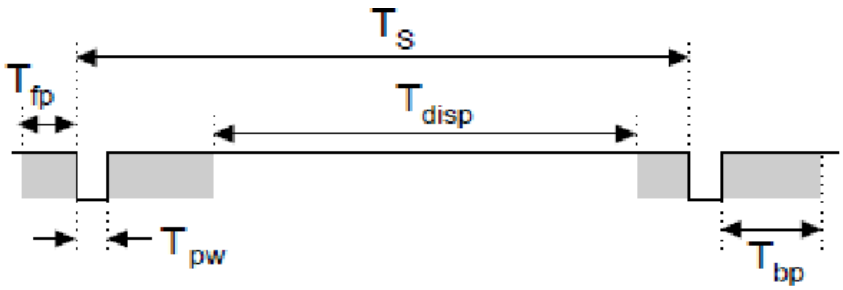
Video Graphics Array (VGA) is a video display controller and accompanying de facto graphics standard, first introduced with the IBM PS/2 line of computers in 1987, which became ubiquitous in the PC industry within three years. The term can now refer either to the computer display standard, the 15-pin D-subminiature VGA connector, or the 640×480 resolution characteristic of the VGA hardware.

The VGA supports all graphics modes supported by the MDA, CGA and EGA cards, as well as multiple new modes.

Standard graphics modes

- 640×480 in 16 colors or monochrome
- 640×350 or 640×200 in 16 colors or monochrome (EGA compatibility)
- 320×200 in 256 colors (Mode 13h)
- 320×200 in 4 or 16 colors (CGA compatibility)

Unlike the cards that preceded it, which used binary TTL signals to interface with a monitor (or composite, in the case of the CGA,) the VGA introduced a video interface using pure analog RGB signals, 0.7 volts peak-to-peak max. In conjunction with an 18-bit RAMDAC this permitted a colour gamut of 262,144 colours.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

The timing for a resolution of 640x480.

The original VGA specifications are as follows:

- Selectable 25.175 MHz or 28.322 MHz master pixel clock
- Maximum of 640 horizontal pixels
- Maximum of 480 lines
- Refresh rates at up to 70 Hz
- Vertical blank interrupt (Not all clone cards support this.)
- Planar mode: up to 16 colors (4 bit planes)
- Packed-pixel mode: 256 colors (Mode 13h)
- Hardware smooth scrolling support
- No Blitter
- Supports fast data transfers via "VGA latch" registers
- Barrel shifter
- Split screen support