Digital System Design

Assignment 1

Submitted By:

16EC105 – Anirudh BH

16EC118 – Manan Sharma

Q1) Write a behavioural description in Verilog that swaps the values in two registers without using a temporary register. the new values should appear #2 after the positive edge.

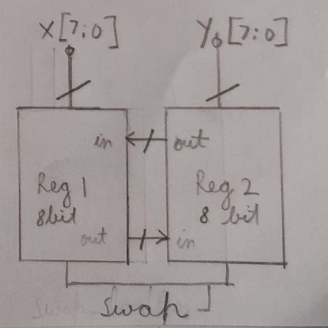


Figure 1: Block diagram for swapping registers

Question by both Manan and Anirudh.

Verilog Code:

`timescale 1ns / 1ps

// Q1 : Swap Registers

// Loading and Swapping registers using if - else and non-blocking statements

module swap(

output reg [7:0] A,B,

input [7:0] X,Y,

input clk, swap\_loadBar // whaen swap\_loadBar is 1 then it swaps else when 0 it l

);

always @(posedge clk) begin

if(!swap\_loadBar) begin

// if swap\_loadBar == 0 then load the registers

A <= X;

B <= Y;

end

else begin

// Swap the register values with each other

B <= #2 A;

A <= #2 B;

end

end

endmodule

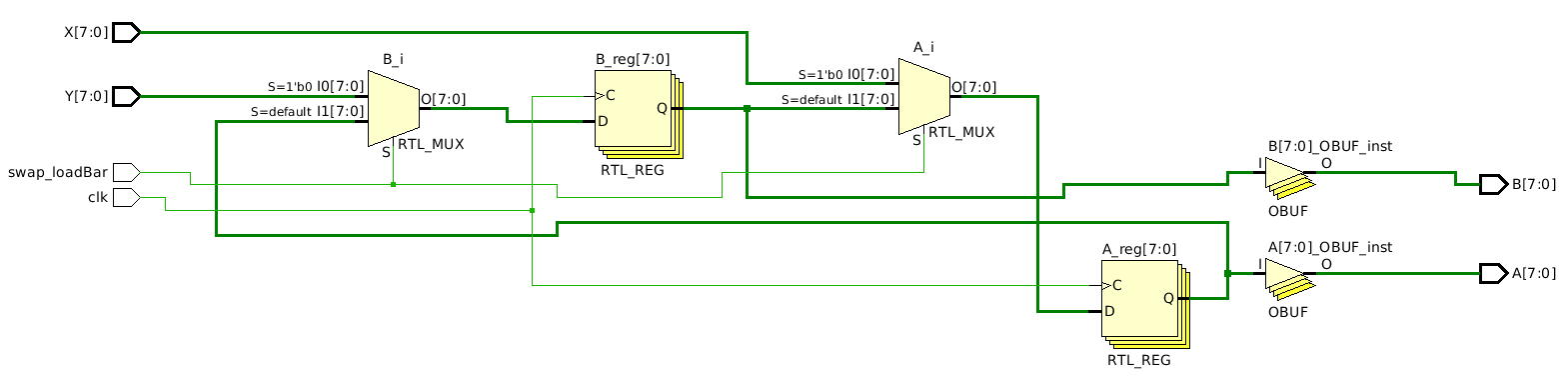


Figure 2: Elaborated RTL Schematic

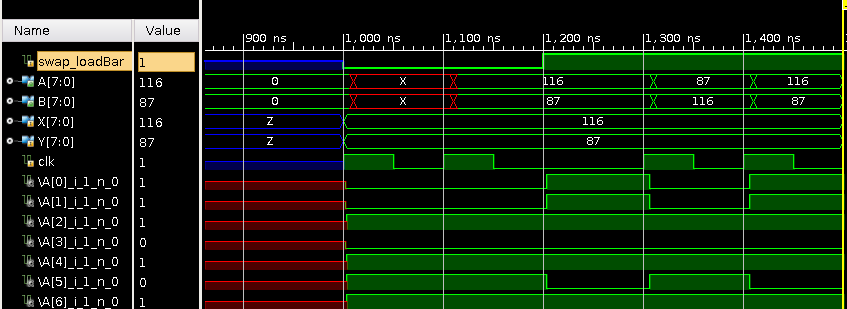


Figure 3: Post Implementation Timing Waveforms

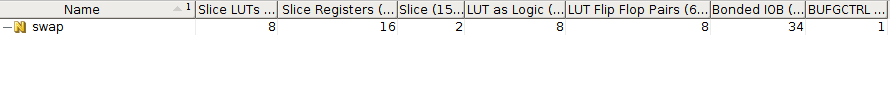


Figure 4: Resource Utilization

Q2) Design an ALU that performs eight functions according to the given table. The data inputs to the ALU are 8 bits. Use always blocks and case statements in your Verilog description.

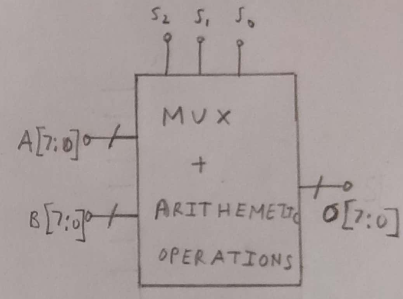


Figure 5: Block diagram for the ALU unit

Question done by Anirudh BH

`timescale 1ns / 1ps

// Q2 : Implementation of an ALU

// Implemnting an ALU using case satements

module ALU(

output reg [7:0] sum\_f1,

output carry\_f2,

input [7:0] A,B,

input [2:0] select

);

always @(select)begin

case(select)

3'b000: sum\_f1 = A + B ;

3'b001: sum\_f1 = A - B ;

3'b010: sum\_f1 = A + 2\*B ;

3'b011: sum\_f1 = A - 2\*B ;

3'b100: sum\_f1 = A[3:0] + B[3:0] ;

3'b101: sum\_f1 = ((A-B)>0) ? A : B ;

3'b110: sum\_f1 = A[7] ? ~A + 1 : A ;

3'b111: sum\_f1 = B ;

endcase

end

endmodule

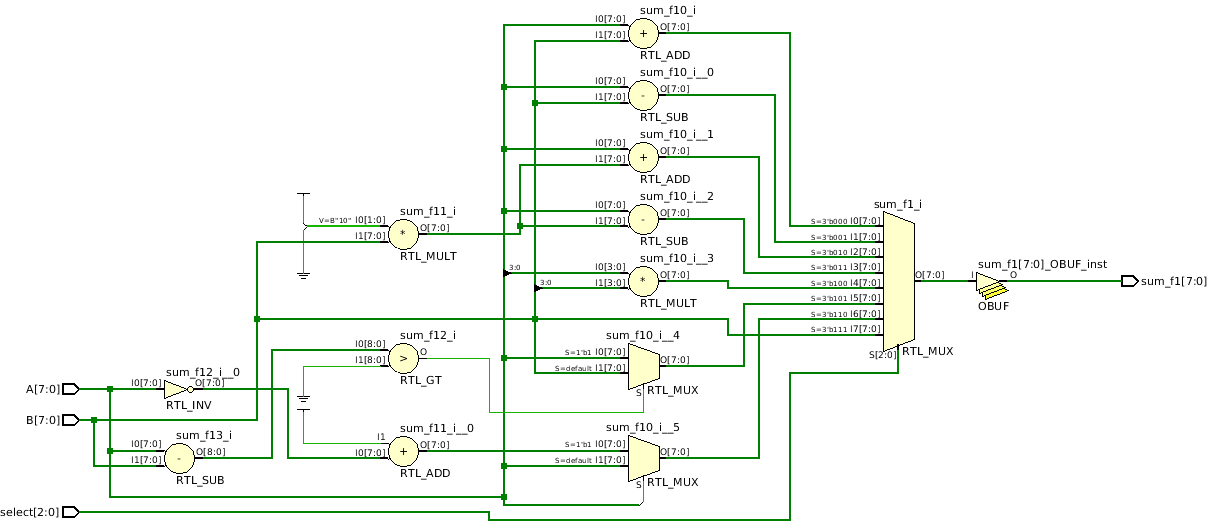


Figure 6: Elaborated RTL Schematic

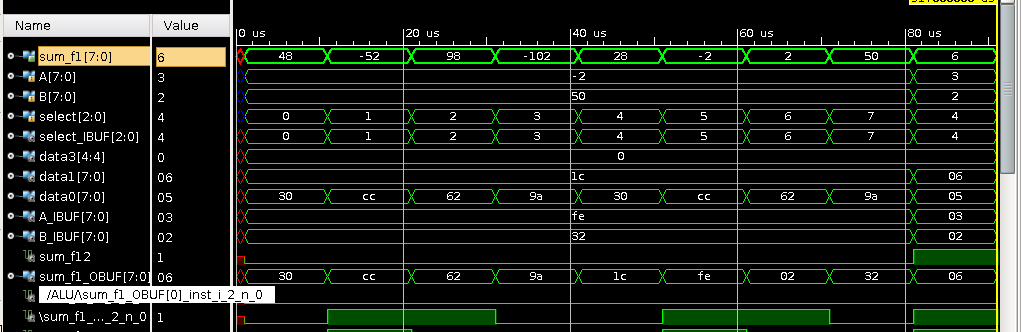


Figure 7: Post Implementation Timing Waveforms

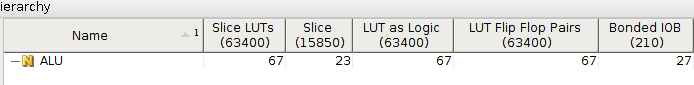


Figure 8: Resource Utilization

Q3) A barrel shifter is a combinational shifter that rotates an arbitrary number of bits. A barrel shifter is often used to shift and rotate n-bits in modern microprocessors, typically within a single clock cycle. Design a barrel shifter that takes in 8-bit data input and a 3-bit control signal, shift which specifies the amount to be rotated.

1. Write a verilog description using case statement

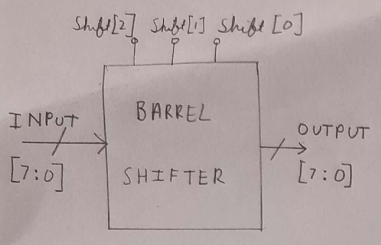


Figure 9: Block diagram for the ALU

Question Done by Manan Sharma.

Verilog Code:

`timescale 1ns / 1ps

//Q3a - Barrel Shifter using case statement.

//Barrel shifter is a combinational circuit that performs ROR operation.

module barrelshifter(

output reg [7:0] data\_out, //Data Output (8bits)

input [7:0] data\_in, //Data Input (8bits)

input [2:0] shift //Shift control

);

always @(\*) //Combinational Circuit

begin

case(shift)

/\*Using case statement to infer a MUX that selects the corresponding bits from input\*/

3'b000: data\_out = data\_in; //No rotate

3'b001: data\_out = {data\_in[0],data\_in[7:1]}; //Right Rotate by 1

3'b010: data\_out = {data\_in[1:0],data\_in[7:2]};//Right Rotate by 2

3'b011: data\_out = {data\_in[2:0],data\_in[7:3]};//Right Rotate by 3

3'b100: data\_out = {data\_in[3:0],data\_in[7:4]};//Right Rotate by 4

3'b101: data\_out = {data\_in[4:0],data\_in[7:5]};//Right Rotate by 5

3'b110: data\_out = {data\_in[5:0],data\_in[7:6]};//Right Rotate by 6

3'b111: data\_out = {data\_in[6:0],data\_in[7]}; //Right Rotate by 7

endcase

end

endmodule

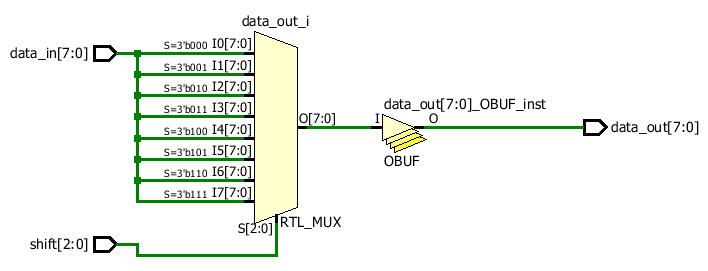


Figure 10: Elaborated RTL Schematic

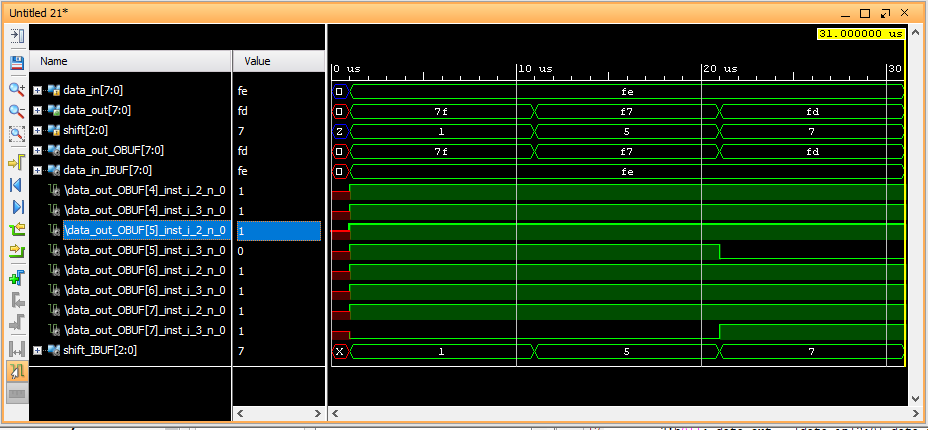


Figure 11:Post Implementation Timing Waveforms



Figure 12: Resource Utilization

1. Alternatively, you may construct a barrel shifter circuit by stages. In the n-th stage, the input signal is either passed directly to output or rotated by 2n positions. The n-th stage is controlled by the n-th bit of the shift signal. Write a verilog code using this logic.

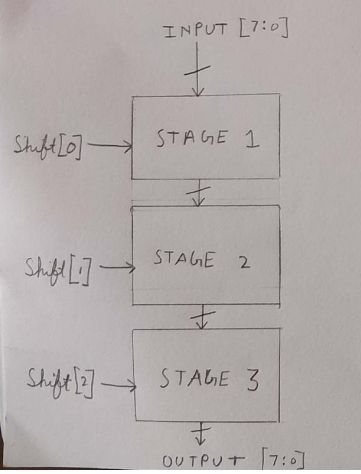


Figure 13: Block Diagram for Barrel Shifter using Stages

Question done by Manan Sharma.

Verilog Code:

`timescale 1ns / 1ps

//Q3b Barrel Shifter using Stage wise Rotation

//It rotates in stages, wherein Nth stage right rotates by 2^N.

module barrelshifter(

output reg [7:0] data\_out, //Data Output (8bits)

input [7:0] data\_in, //Data Input (8bits)

input [2:0] shift //Shift control

);

always @(\*)

begin

data\_out = shift[0] ? {data\_in[0],data\_in[7:1]} : data\_in;

//Loads a right rotated by 1 version of input if LSB of shift is 1, else loads in the input directly

data\_out = shift[1] ? {data\_out[1:0],data\_out[7:2]}: data\_out;

//Right rotates by 2 if middle bit of shift is 1, else retains the previous value

data\_out = shift[2] ? {data\_out[3:0],data\_out[7:4]} : data\_out;

//Right rotates by 4 if MSB of shift is 1, else retains previous output

end

endmodule

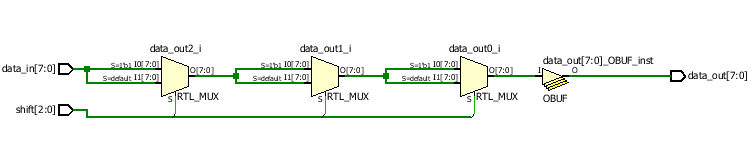


Figure 14: Elaborated RTL Schematic

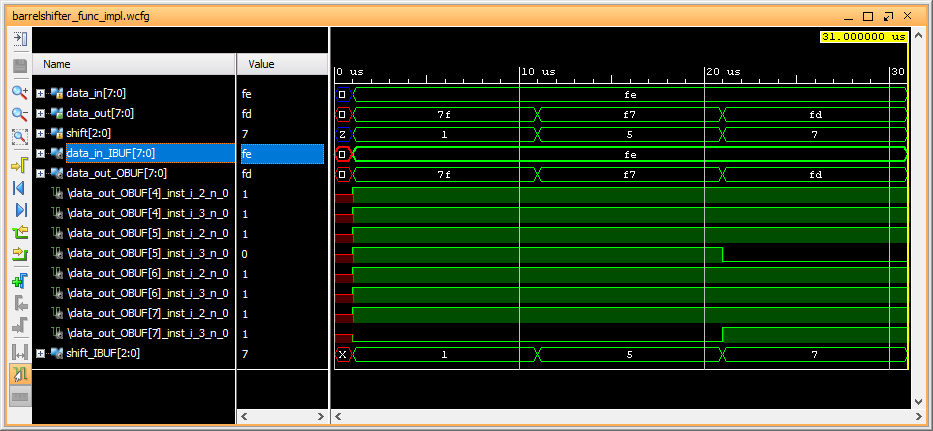


Figure 15: Post Implementation Timing Waveforms



Figure 16: Resource Utilization

1. List applications of barrel shifter

Barrel Shifter is used in:

1. Barrel shifter is used extensively in RISC architecture’s ALU datapath to shift numbers, and perform multiplication and division by 2 operations within 1 clock cycle.
2. It is used in floating point adders and subtractors to shift numbers.

Q4) A simple sequential multiplier is developed by adding A to itself B number of times. Both A and B inputs are 8-bit unsigned numbers. The circuit has a start input that becomes 1 for one clock when the inputs are valid. The inputs will not be valid when this signal is 0. After the start, the circuit sets its done output to 0 and starts the multiplication process. When done, it puts the result on R and sets done to 1. Write a verilog code for the sequential multiplier

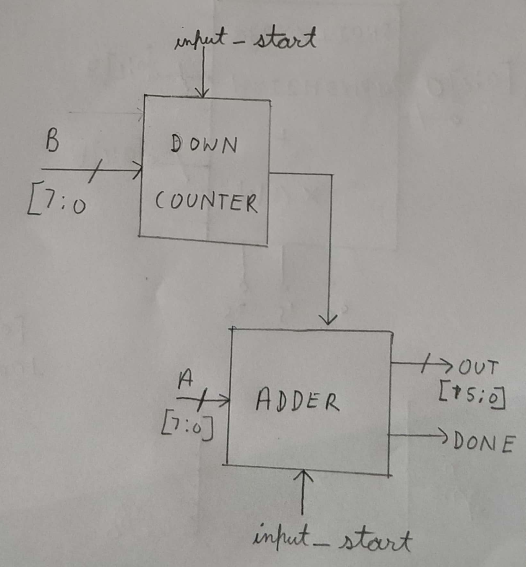


Figure 17: Block diagram of the multiplier

Question done by Anirudh BH

Verilog Code:

`timescale 1ns / 1ps

// Q4 : Sequential Multiplier

// Implementing a sequential multiplier using a temporary register and a buffer

module multi(

output reg [15:0] out,

output reg done,

input [7:0] A,B,

input clk, input\_start, clearBar

);

reg[15:0] temp;

reg [7:0] buff;

always @(posedge clk, negedge clearBar) begin

if(!clearBar) begin

// Clear all registers if clearBar == 0

out <= 0;

temp <= 0;

buff <= 0;

end

else if(input\_start==1)begin

// Signal to start the process. Load B into buffer for decrementing

done <= 0;

temp <= 0;

buff <= B;

end

else if(input\_start==0 && done==0) begin

temp = temp + A; // temp is used as a register to store repeated addtion of A

buff = buff - 1; // Decremtn buffer till buffer == 0

if(buff==0)

begin

out = temp; // When Buffer is 0 set done = 1 and load the value in temp to output

done = 1;

end

else done = 0;

end

else

out <= out; // If input and done are 0. Keep latching the value of output (value of previos answer)

end

endmodule

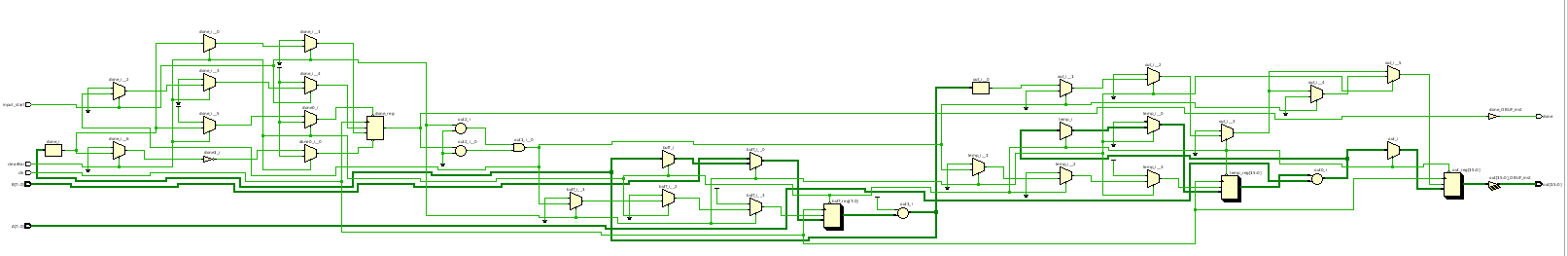


Figure 18: Elaborated RTL Schematic

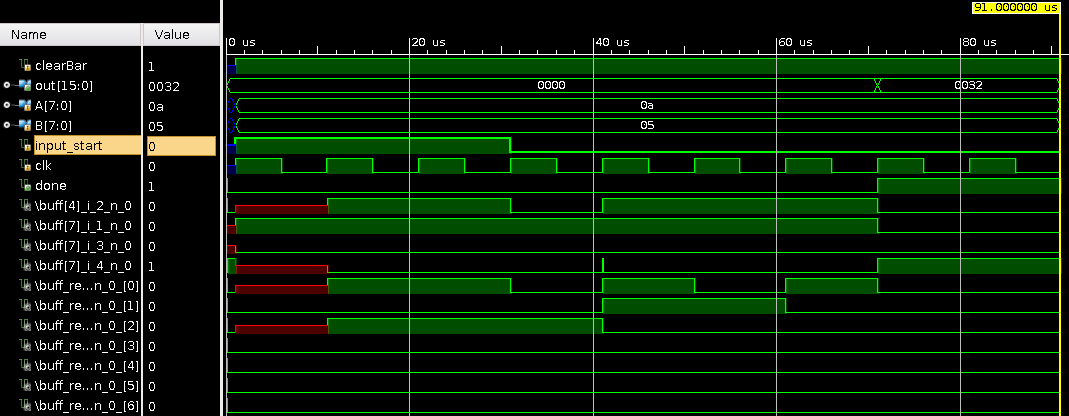


Figure 19: Post Implementation Timing Waveforms

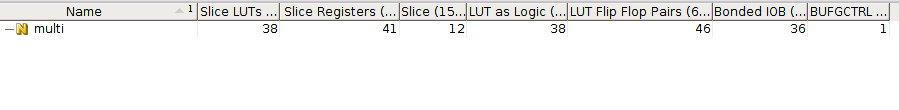


Figure 20: Resource Utilization

Q5) Design a 4 bit counter that has a select line that allows you to count up or down. When the counter counts up, it should increment by 3. When the counter counts down, it should decrement by 5. In addition, the counter has an active high enable. When the enable is high, the counter should hold its current count. When the counter is low, it should count in the manner described previously. The counter also has an active low reset. When the reset is low, the output of the counter should reset to a binary zero.

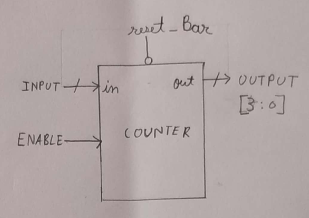


Figure 21: Block Diagram of Counter

Question done by Manan Sharma.

Verilog Code:

`timescale 1ns / 1ps

// Q5 Up Down Counter

// Counts up by 3 when enable=0 and up\_downBar=1

// Counts down by 5 when enable=0 and up\_downBar=0

// Count value is retained when enable=1

// clearbar is fed as asynchronous clear

module counter\_up\_down(

output reg [3:0] out, //Output of the counter

input enable, clearBar, up\_downBar, clk

);

always @(posedge clk, negedge clearBar)

begin

if(clearBar == 0) out<= 4'b0000; //To clear the output

else begin

case({enable,up\_downBar})

2'b00: out <= out-5; //Down Count

2'b01: out <= out+3; //Up Count

default: out <= out; //Retain value

endcase

end

end

endmodule

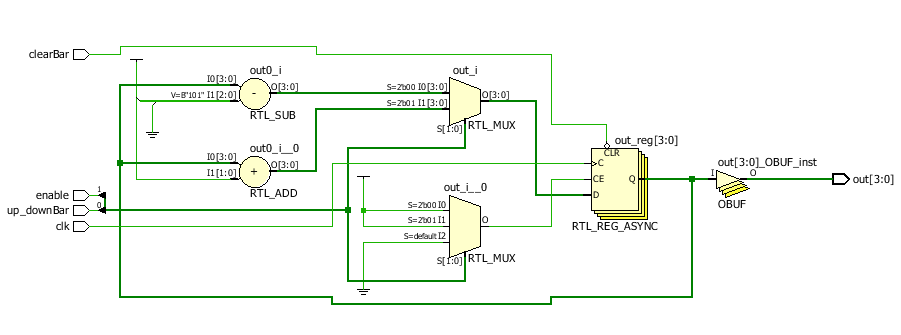


Figure 22: Elaborated RTL Schematic

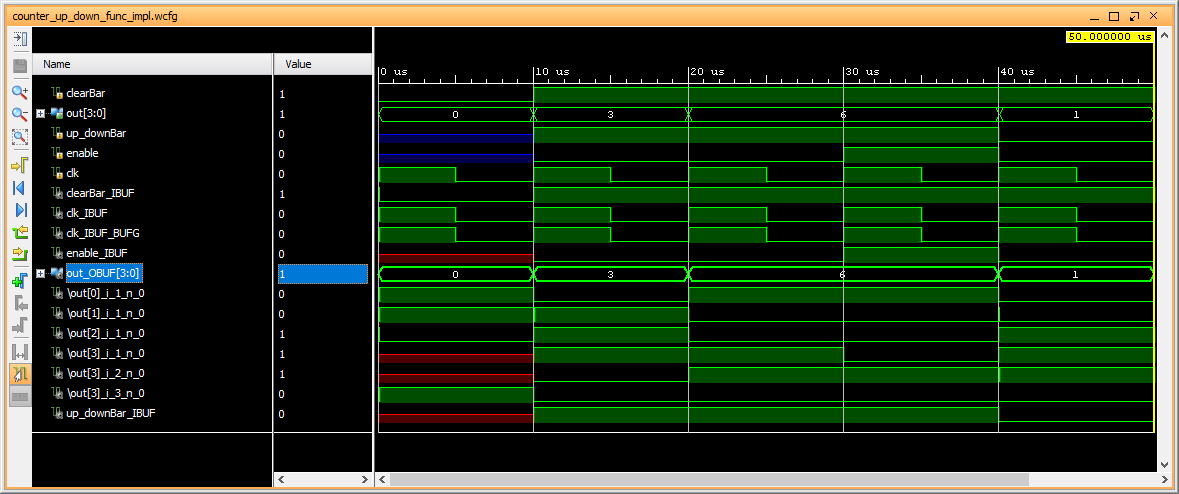


Figure 23: Post Implementation Timing Waveforms



Figure 24: Resource Utilization

Q6) Let the board position in an X & 0 (tic-tac-toe) game be denoted by (v11, v12, v13, v21, v22, v23,  
v31, v32, v33) where vij’s are 2-bit values. Let vij=11 (00) denote presence of X (0) and any other  
value denote an empty block. Design a circuit to output whether the player whose turn it is can win  
the game in the current move.

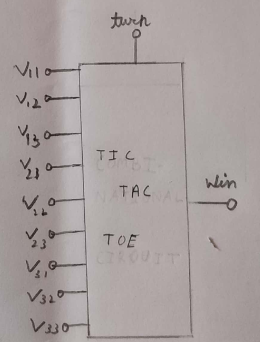


Figure 25: Block Diagram of Tic Tac Toe Game

Question done by Anirudh BH

Verilog Code:

`timescale 1ns / 1ps

// Q6: Tic Tac Toe Game

// Code to see if the given player can win in the current turn, given the board value

module tic\_tac\_toe(

output out,

input turn, // turn = 0 for O turn = 1 for X

input[1:0] v11, v12, v13, v21, v22, v23, v31, v32, v33

);

wire r1,r2,r3,c1,c2,c3,d1,d2;

// Check of rows

check3 cr1(v11,v12,v13,turn,r1);

check3 cr2(v21,v22,v23,turn,r2);

check3 cr3(v31,v32,v33,turn,r3);

//Check for columns

check3 cc1(v11,v21,v31,turn,c1);

check3 cc2(v12,v22,v32,turn,c2);

check3 cc3(v13,v23,v33,turn,c3);

// Check for diagonal 3's

check3 cd1(v11,v22,v33,turn,d1);

check3 cd2(v13,v22,v31,turn,d2);

// Output is 1 if any of the previous 3's are true

assign out = r1 | r2 | r3 | c1 | c2 | c3 | d1 | d2;

endmodule

`timescale 1ns / 1ps

// Auxilary Module for Q6

module check3(

input [1:0] v1, v2, v3,

input turn,

output out

);

wire isP1, isP2, isP3;

wire isB1, isB2, isB3;

boxValue bv1(v1,turn, isP1, isB1);

boxValue bv2(v2,turn, isP2, isB2);

boxValue bv3(v3,turn, isP3, isB3);

// win is possible if any 2 are correctly marked and the other is blank

assign out = (isB1 & isP2 & isP3) | (isP1 & isB2 & isP3) | (isP1 & isP2 & isB3);

endmodule

`timescale 1ns / 1ps

// Auxilary Function for Q6

module boxValue(

input[1:0] value,

input turn,

output isPresent, isBlank

);

wire isX,isZero;

assign isX = (value[1] & value[0]); // Box is marked X if both bits are 1

assign isZero = ~(value[1] | value[0]); // Box is marked O if both bits are 0

assign isPresent = (turn & isX) | (~turn & isZero); // Box is marked if its the given player's turn and if thier mark is present

assign isBlank = ~(isX | isZero); // Blank if no mark present

endmodule

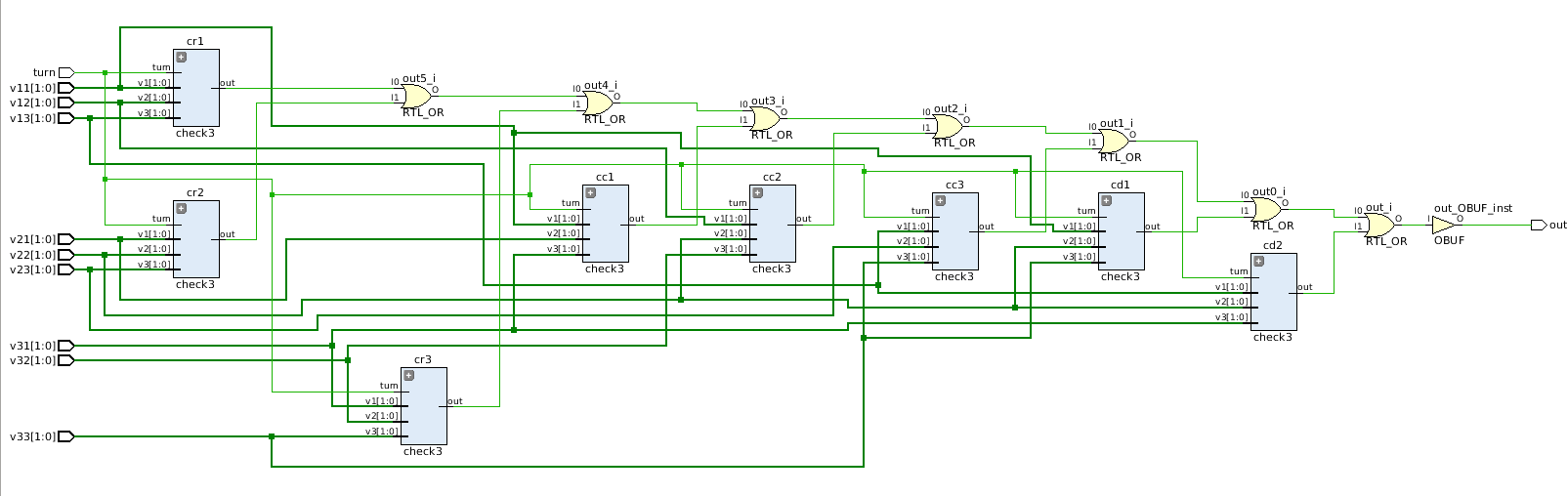


Figure 26: Elaborated Schematic

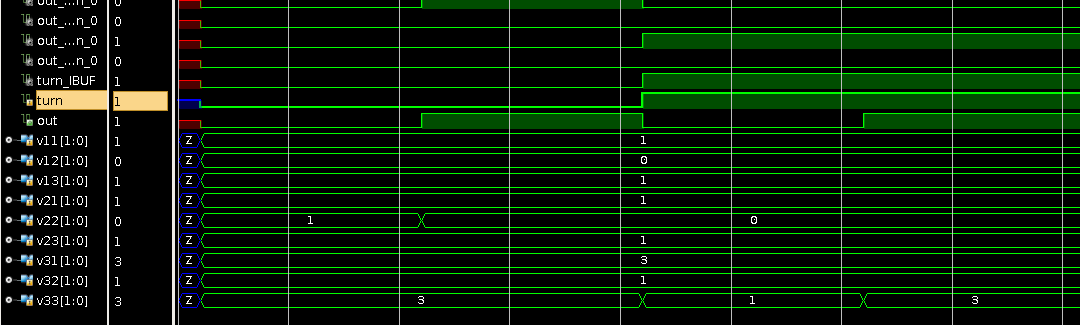


Figure 27: Post Implementation Timing Waveforms

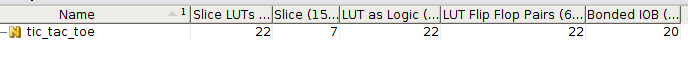


Figure 28: Resource Utilization

Q7) With the help of verilog code, design a register unit that performs operations shown below. The unit has a 3-bit mode (md) input, an asynchronous reset (rs) input, a 1-bit output control (oc) input, and an 8-bit bidirectional io bus. The internal register drives the io bus when oc is 1 and md is not 111.  
md=000: does nothing  
md=001: right shift the register  
md=010: left shift the register  
md=011: up count, Gray (000, 001, 011, 010, 110, 111, 101, 100)  
md=100: down count, Gray (opposite of the above)  
md=101: complement register contents  
md=110: swap right and left 4 bits  
md=111: parallel load

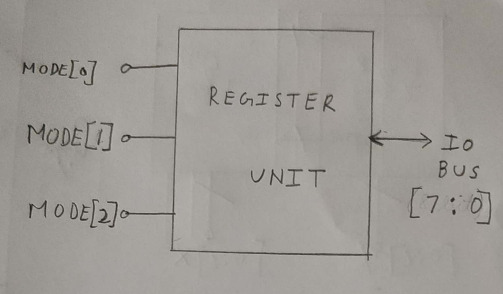


Figure 29: Block Diagram of Register Unit

Question done by Manan Sharma.

Verilog Code:

`timescale 1ns / 1ps

//Q7 Register Unit. Performs operations on bidirectional 8-bit bus.

module registerunit(

inout [7:0] io\_bus, //Bidirectional bus

input [2:0] mode\_input, //Mode Input

input clk,

input reset, //Active low asynchronus reset

input output\_control //1 bit output control

);

reg[7:0] a,b; //Internal registers used for

assign io\_bus = output\_control?b:8'bZZZZZZZZ; //assigns the output depending on output\_control value

always @(posedge clk, negedge reset)

begin

if(reset==0) //used to reset a and b

begin

a <= 8'b00000000;

b <= 8'b00000000;

end

else

case(mode\_input)

3'b000: b = a; //Do nothing. Holding the previous value

3'b001: begin //Right shift by 1 value

a = {a[0],a[7:1]};

b = a;

end

3'b010: begin //Left shift by 1 value

a = {a[6:0],a[7]};

b = a;

end

3'b011: //8 bit Gray code upcount by 1

begin

a = a+1;

b= {a[7],a[7]^a[6],a[6]^a[5],a[5]^a[4],a[4]^a[3],a[3]^a[2],a[2]^a[1],a[1]^a[0]};

end

3'b100: //8 bit Gray code downcount by 1

begin

a = a-1;

b= {a[7],a[7]^a[6],a[6]^a[5],a[5]^a[4],a[4]^a[3],a[3]^a[2],a[2]^a[1],a[1]^a[0]};

end

3'b101: begin //Bitwise NOT the register contents

a = ~a;

b = a;

end

3'b110: begin // Swapping right and left 4 bits

a = {a[3:0],a[7:4]};

b = a;

end

3'b111: begin //parallel load to register

a = io\_bus;

b = a;

end

endcase

end

endmodule

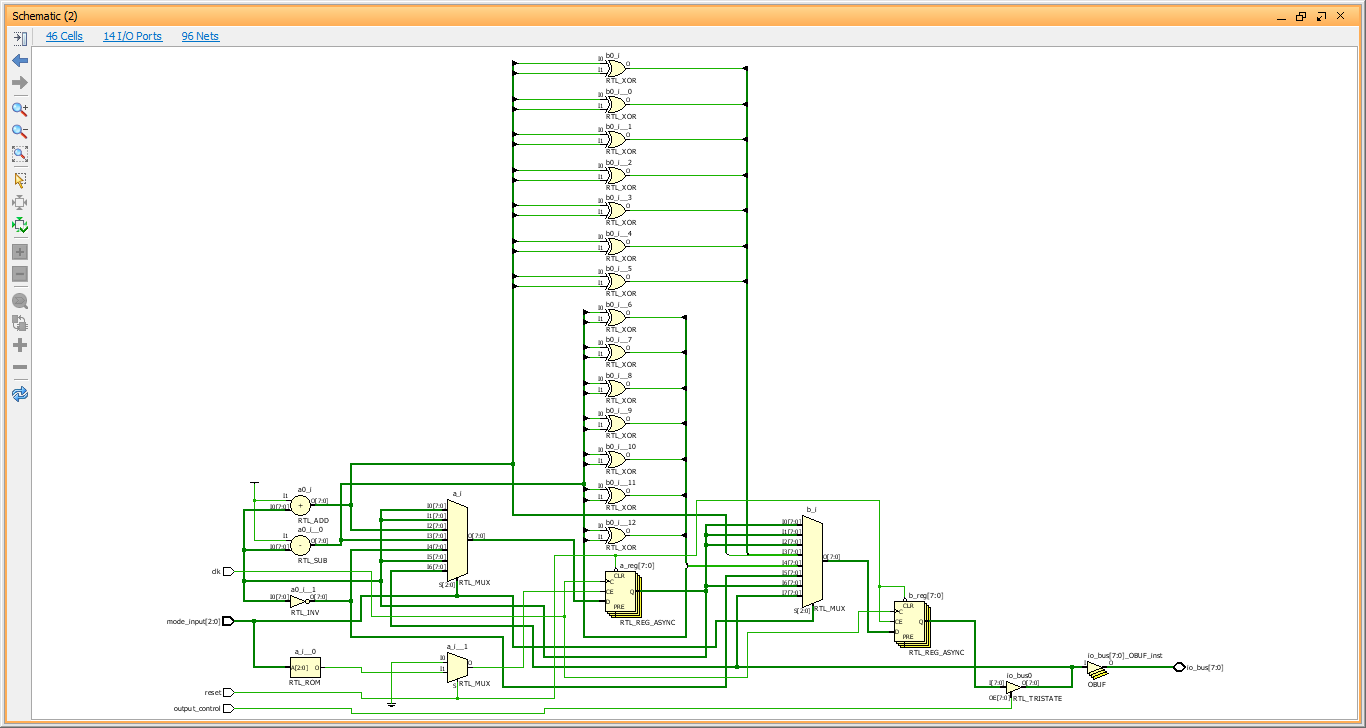


Figure 30: Elaborated RTL Schematic

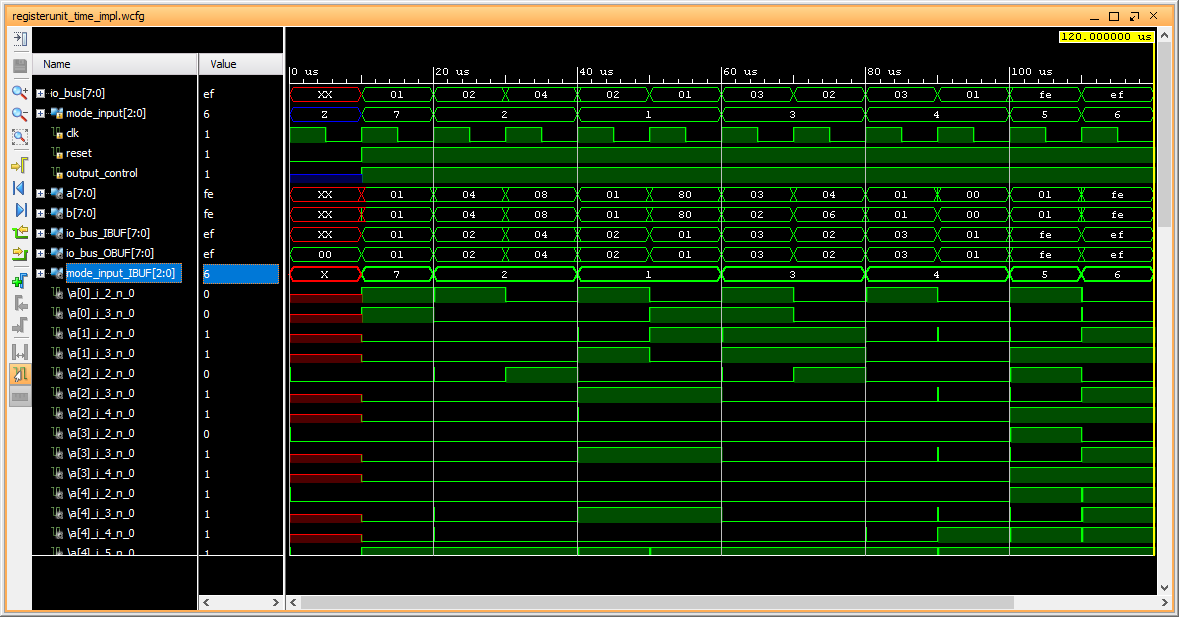


Figure 31: Post Implementation Timing Waveforms



Figure 32: Resource Utilization