**HW5**

**Daksh Thapar(2018137)**

**Tanmaya Gupta(2018200)**

Design a 4X4 multiplier Take the input from PMOD switches. Display the input on external SSD and display the resultant product on the on board seven segment display.

**Verilog Code-**

module top(

input clk,

inout [7:0] JA,

output [3:0] anode,

output [7:0] cathode,

output [6:0] cathode1,

output anode1

);

wire [3:0] ones,tens,hundreds,thousands;

wire clk\_1k;

// wire [1:0] second = 2'b10;

reg [7:0] result;

wire clk\_5M;

clk\_wiz\_0 c11(.clk\_in1(clk),.clk\_out1(clk\_5M));

clk\_div #(13) u1( .clk\_in(clk\_5M) , .clk\_out(clk\_1K));

clk\_div #(15) u2(.clk\_in(clk\_5M),.clk\_out(clk\_5ms));

wire [3:0] decode,decode1;

wire bouncer;

Decoder d0(.clk(clk),.Row(JA[7:4]),.Col(JA[3:0]),.DecodeOut(decode),.clking(bouncer));

wire bouncing;

// wire bool;

reg [3:0] first1,second1;

reg count;

wire temp\_bouncer;

debouncer d1(.clk\_in(clk\_5ms),.clr\_o(temp\_bouncer),.clr\_i(bouncer));

always @ (posedge temp\_bouncer)

begin

count <= count +1'd1;

if (count == 1'd1)

begin

first1 <= decode;

end

end

always @ (posedge temp\_bouncer)

begin

count <= count +1'd1;

if (count == 1'd0)

begin

second1 <= decode;

end

end

always@(\*)

begin

result = first1\*second1;

end

sevensegment s2(.clk(clk\_1K),.ones(second1),.tens(first1),.anode(anode1),.cathode(cathode1));

wire [7:0] result\_temp = result;

bin\_to\_bcd b1(.number(result\_temp),.ones(ones),.tens(tens),.hundreds(hundreds),.thousands(thousands));

sevenSeg s1(.clk(clk\_1K),.ones(ones),.tens(tens),.thousands(thousands),.hundreds(hundreds),.anode(anode),.cathode(cathode));

endmodule

module sevensegment(

input clk,

input [3:0] ones,

input [3:0] tens,

output [7:0] cathode,

output anode

);

reg [6:0] cathode\_temp = 7'b0000000;

reg anode\_temp = 1'b0;

reg count = 0;

always @(posedge clk)

begin

count = count + 1;

end

always @(posedge clk)

begin

case(count)

1'b0:

begin

case(ones)

4'd0 : cathode\_temp = 7'b1111110; //to display 0

4'd1 : cathode\_temp = 7'b0110000; //to display 1

4'd2 : cathode\_temp = 7'b1101101;

4'd3 : cathode\_temp = 7'b1111001;

4'd4 : cathode\_temp = 7'b0110011;

4'd5 : cathode\_temp = 7'b1011011;

4'd6 : cathode\_temp = 7'b1011111;

4'd7 : cathode\_temp = 7'b1110000;

4'd8 : cathode\_temp = 7'b1111111;

4'd9 : cathode\_temp = 7'b1111011;

4'd10 : cathode\_temp = 7'b1110111;

4'd11 : cathode\_temp = 7'b0011111;

4'd12 : cathode\_temp = 7'b1001110;

4'd13 : cathode\_temp = 7'b0111101;

4'd14 : cathode\_temp = 7'b1101111;

4'd15 : cathode\_temp = 7'b1000111;

default : cathode\_temp = 7'b0000000;

endcase

anode\_temp = 1'b0;

end

1'b1:

begin

case(tens)

4'd0 : cathode\_temp = 7'b1111110; //to display 0

4'd1 : cathode\_temp = 7'b0110000; //to display 1

4'd2 : cathode\_temp = 7'b1101101;

4'd3 : cathode\_temp = 7'b1111001;

4'd4 : cathode\_temp = 7'b0110011;

4'd5 : cathode\_temp = 7'b1011011;

4'd6 : cathode\_temp = 7'b1011111;

4'd7 : cathode\_temp = 7'b1110000;

4'd8 : cathode\_temp = 7'b1111111;

4'd9 : cathode\_temp = 7'b1111011;

4'd10 : cathode\_temp = 7'b1110111;

4'd11 : cathode\_temp = 7'b0011111;

4'd12 : cathode\_temp = 7'b1001110;

4'd13 : cathode\_temp = 7'b0111101;

4'd14 : cathode\_temp = 7'b1101111;

4'd15 : cathode\_temp = 7'b1000111;

default : cathode\_temp = 7'b0000000;

endcase

anode\_temp = 1'b1;

end

endcase

end

assign anode = anode\_temp;

assign cathode = cathode\_temp;

endmodule

module sevenSeg(

input clk,

input [3:0] ones,

input [3:0] tens,

input [3:0] hundreds,

input [3:0] thousands,

output [3:0] anode,

output [7:0] cathode

);

reg [6:0] cathode\_temp = 7'b1111111;

reg [3:0] anode\_temp = 4'b1110;

reg [1:0] count = 0;

always @(posedge clk)

begin

count = count + 1;

end

always @ (posedge clk)

begin

case(count)

2'b00:

begin

case(ones)

4'd0 : cathode\_temp = 7'b0000001;

4'd1:cathode\_temp = 7'b1001111;

4'd2:cathode\_temp = 7'b0010010;

4'd3:cathode\_temp = 7'b0000110;

4'd4:cathode\_temp = 7'b1001100;

4'd5:cathode\_temp = 7'b0100100;

4'd6:cathode\_temp = 7'b0100000;

4'd7:cathode\_temp = 7'b0001111;

4'd8:cathode\_temp = 7'b0000000;

4'd9 :cathode\_temp = 7'b0000100;

default:cathode\_temp = 7'b1111111;

endcase

anode\_temp = 4'b1110;

end

2'b01:

begin

case(tens)

4'd0 : cathode\_temp = 7'b0000001;

4'd1:cathode\_temp = 7'b1001111;

4'd2:cathode\_temp = 7'b0010010;

4'd3:cathode\_temp = 7'b0000110;

4'd4:cathode\_temp = 7'b1001100;

4'd5:cathode\_temp = 7'b0100100;

4'd6:cathode\_temp = 7'b0100000;

4'd7:cathode\_temp = 7'b0001111;

4'd8:cathode\_temp = 7'b0000000;

4'd9 :cathode\_temp = 7'b0000100;

default:cathode\_temp = 7'b1111111;

endcase

anode\_temp = 4'b1101;

end

2'b10:

begin

case(hundreds)

4'd0 : cathode\_temp = 7'b0000001;

4'd1:cathode\_temp = 7'b1001111;

4'd2:cathode\_temp = 7'b0010010;

4'd3:cathode\_temp = 7'b0000110;

4'd4:cathode\_temp = 7'b1001100;

4'd5:cathode\_temp = 7'b0100100;

4'd6:cathode\_temp = 7'b0100000;

4'd7:cathode\_temp = 7'b0001111;

4'd8:cathode\_temp = 7'b0000000;

4'd9 :cathode\_temp = 7'b0000100;

default:cathode\_temp = 7'b1111111;

endcase

anode\_temp = 4'b1011;

end

2'b11:

begin

case(thousands)

4'd0 : cathode\_temp = 7'b0000001;

4'd1:cathode\_temp = 7'b1001111;

4'd2:cathode\_temp = 7'b0010010;

4'd3:cathode\_temp = 7'b0000110;

4'd4:cathode\_temp = 7'b1001100;

4'd5:cathode\_temp = 7'b0100100;

4'd6:cathode\_temp = 7'b0100000;

4'd7:cathode\_temp = 7'b0001111;

4'd8:cathode\_temp = 7'b0000000;

4'd9 :cathode\_temp = 7'b0000100;

default:cathode\_temp = 7'b1111111;

endcase

anode\_temp = 4'b0111;

end

endcase

end

assign anode = anode\_temp;

assign cathode = {cathode\_temp,1'b1};

endmodule

module debouncer(

input clk\_in,

input clr\_i,

output clr\_o

);

reg d1, d2, d3;

always @(posedge clk\_in)

begin

d1 <= clr\_i;

d2 <= d1;

d3 <= d2;

end

assign clr\_o = d1 && d2 && d3;

endmodule

module Decoder(

input clk,

input [3:0] Row,

output reg [3:0] Col,

output clking,

output reg [3:0] DecodeOut

);

reg [3:0] temp;

reg [19:0] sclk = 0 ;

always @ (posedge clk)

begin

if (sclk == 20'b00011000011010100000)

begin

Col <= 4'b0111;

sclk <= sclk + 1'b1;

temp[0] <= 1'b1;

end

else if(sclk ==20'b00011000011010101000)

begin

if (Row==4'b0111) begin

DecodeOut <= 4'b0001;

end

else if (Row == 4'b1011) begin

DecodeOut <= 4'b0100;

end

else if (Row == 4'b1101) begin

DecodeOut <= 4'b0111;

end

else if (Row == 4'b1110) begin

DecodeOut <= 4'b0000;

end

else begin

temp[0] = 1'b0;

end

sclk <= sclk + 1'b1;

end

else if (sclk == 20'b00110000110101000000) begin

Col <= 4'b1011;

sclk <= sclk +1'b1;

temp[1]<= 1'b1;

end

else if (sclk == 20'b00110000110101001000) begin

if (Row==4'b0111) begin

DecodeOut <= 4'b0010;

end

else if (Row == 4'b1011) begin

DecodeOut <= 4'b0101;

end

else if (Row == 4'b1101) begin

DecodeOut <= 4'b1000;

end

else if (Row == 4'b1110) begin

DecodeOut <= 4'b1111;

end

else

begin

temp[1] <= 1'b0;

end

sclk <= sclk + 1'b1;

end

else if (sclk == 20'b01001001001111100000) begin

Col <= 4'b1101;

sclk <= sclk +1'b1;

temp[2]<= 1'b1;

end

else if (sclk == 20'b01001001001111101000) begin

if (Row==4'b0111) begin

DecodeOut <= 4'b0011;

end

else if (Row == 4'b1011) begin

DecodeOut <= 4'b0110;

end

else if (Row == 4'b1101) begin

DecodeOut <= 4'b1001;

end

else if (Row == 4'b1110) begin

DecodeOut <= 4'b1110;

end

else

begin

temp[2]<=1'b0;

end

sclk <= sclk + 1'b1;

end

else if (sclk == 20'b01100001101010000000) begin

Col <= 4'b1110;

sclk <= sclk +1'b1;

temp[3]<= 1'b1;

end

else if (sclk == 20'b01100001101010001000) begin

if (Row==4'b0111) begin

DecodeOut <= 4'b1010;

end

else if (Row == 4'b1011) begin

DecodeOut <= 4'b1011;

end

else if (Row == 4'b1101) begin

DecodeOut <= 4'b1100;

end

else if (Row == 4'b1110) begin

DecodeOut <= 4'b1101;

end

else

begin

temp[3] <= 1'b0;

end

sclk <= 20'b00000000000000000000;

end

else begin

sclk <= sclk + 1'b1;

end

end

assign clking = temp[0]|temp[1]|temp[2]|temp[3];

endmodule

module clk\_div

#(

parameter divide\_by = 19

)

( input clk\_in,

output clk\_out );

reg [divide\_by-1:0] count = 0 ;

always @ (posedge clk\_in)

begin

count = count+1 ;

end

assign clk\_out = count[divide\_by-1] ;

endmodule

module bin\_to\_bcd(

input [7:0] number ,

output reg [3:0] thousands,

output reg [3:0] hundreds,

output reg [3:0] tens,

output reg [3:0] ones

);

reg [19:0] shift = 0;

integer i;

always @(number)

begin

shift[19:8] = 0;

shift[7:0] = number;

for (i=0; i<8; i=i+1)

begin

if (shift[11:8] >= 5)

shift[11:8] = shift[11:8] + 3;

if (shift[15:12] >= 5)

shift[15:12] = shift[15:12] + 3;

if (shift[19:16] >= 5)

shift[19:16] = shift[19:16] + 3;

shift = shift << 1;

end

thousands <= 4'b0000; // as an 8-bit number can be stored in 3 digit BCD code.

hundreds <= shift[19:16];

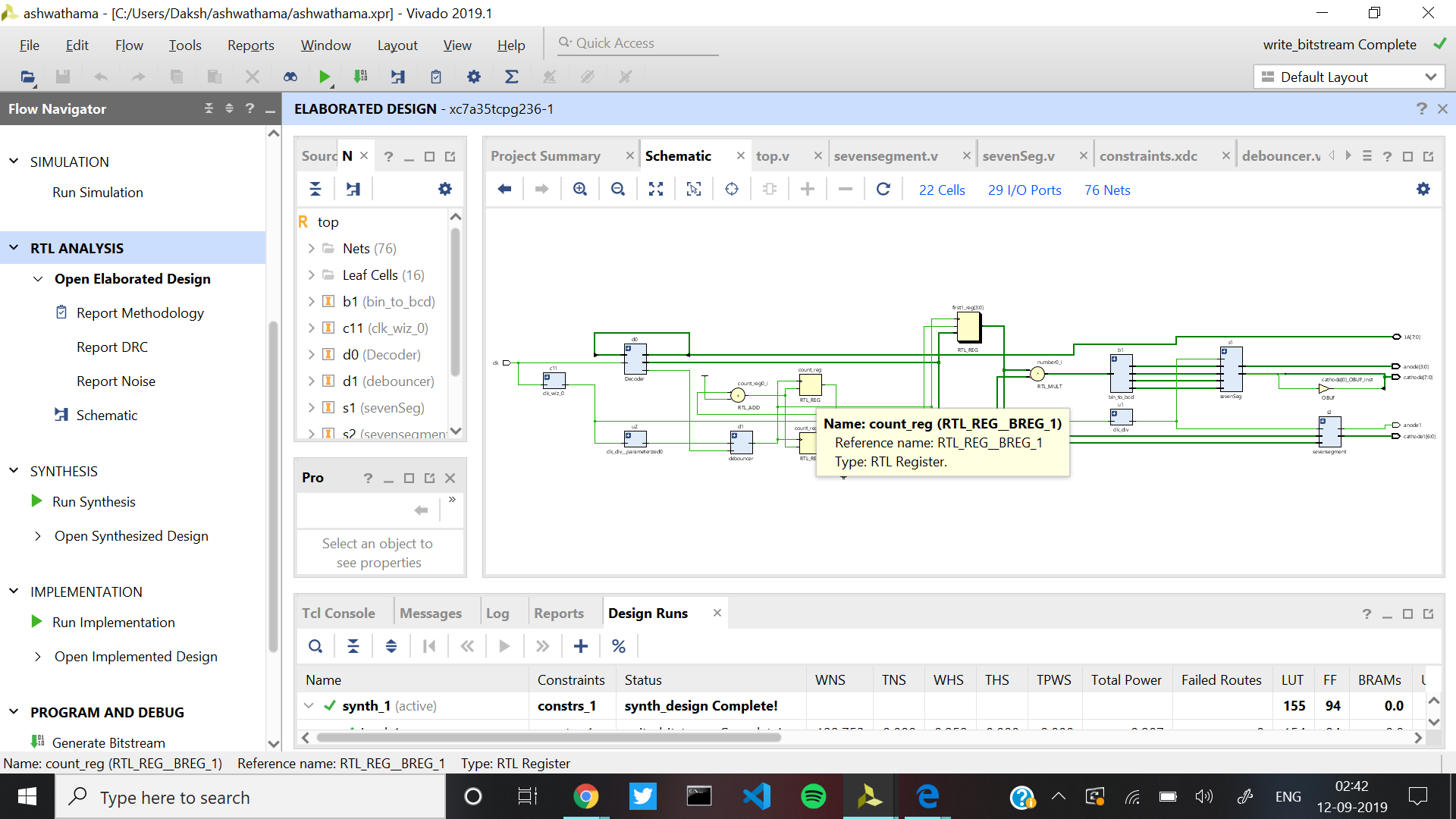
tens <= shift[15:12];

ones <= shift[11:8];

end

endmodule

**RTL Design-**



**Post Synthesis-**