Assignment 1

Submitted by: Adwaith K J (B190245EC), adwaith_b190245ec@nitc.ac.in

- Questions

1) Write a behavioral level Verilog code for a 4-bit mod-13 up-down counter and a testbench to verify.
2) Write a module in Verilog called bitreverse that has one input and one output signal. Output should be the bit reversed version of the input signal. Size of the signal is defined as a parameter called MAXBITS with a default value of 32. Implement the bit reversal as a
   a) Function
   b) Task
   Write different testbenches by instantiating the DUT with different parameter values 4 and 8. Simulate and display the results for atleast four different input signals.
3) Draw the state transition diagram of a sequence detector which detects 0011 using Moore FSM. Assume LSB enters the system first. Model this FSM in Verilog using separate always blocks for next state logic, sequential logic and output logic. Write a testbench to verify the design.
4) Write a parameterized N-bit ripple carry adder using generate statements. Write two different testbenches – one for verifying 8-bit adder and

Answers

1) 4 Bit mod 13 updown counter

Gate Level Circuit

Code

```
module updowncounter(clk,rst,s,count);


input clk,rst,s;// s =0 for count down and vice
versa


output reg [3:0] count;
```

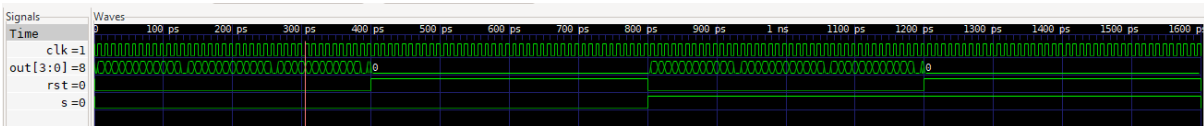
```
initial count=4'b0000;


always @(posedge (clk) or posedge(rst))
begin
if (rst==1'b1)
    count<=0;
else
    if (s==1)
        if (count==4'b1100)
            count=0;
        else
            count<=count+1;
    else
        if (count==0)
            count=4'b1100;
        else
            count<=count-1;

end


endmodule
```

Testbench

```
`timescale 1ps/1ps

module testbench;
reg clk,rst,s;
wire [3:0] out;
```

```
updowncounter DUT(clk,rst,s,out);


initial begin
    clk=1'b0;
    rst=1'b0;
    s=1'b0;
    #1600 $finish;
end

always #5 clk=~clk;
always #400 rst=~rst;
always #800 s=~s;


initial begin
    $dumpfile("1.vcd");
    $dumpvars;
            $monitor("%g   clk=%b   rst=%2b   s=%b,
count=%4b",$time,clk, rst,s, out);
end

endmodule
```

Waveform



2) Bit reversal

Truth Table

in=01000010010000000101000111001111
out=11110011100010100000001001000010

in=01010010110001100100000011010010
out=01001011000000100110001101001010

(Taking the reverse of the bitstream)

a) As function

Code:

```
module bitreverse(in,out);

    parameter MAXBITS=32;

    input [MAXBITS-1:0] in;
    output reg [MAXBITS-1:0] out;

    reg a[MAXBITS-1:0];

    always @(in)
    begin

        out=rev(in);

    end

    function [MAXBITS-1:0] rev(input [MAXBITS-1:0]
in);
```

```
            integer i;
            for (i=0; i<MAXBITS; i=i+1)
                begin
                    rev[i]=in[MAXBITS-1-i];
                end

    endfunction

endmodule
```

Output:

b) Task

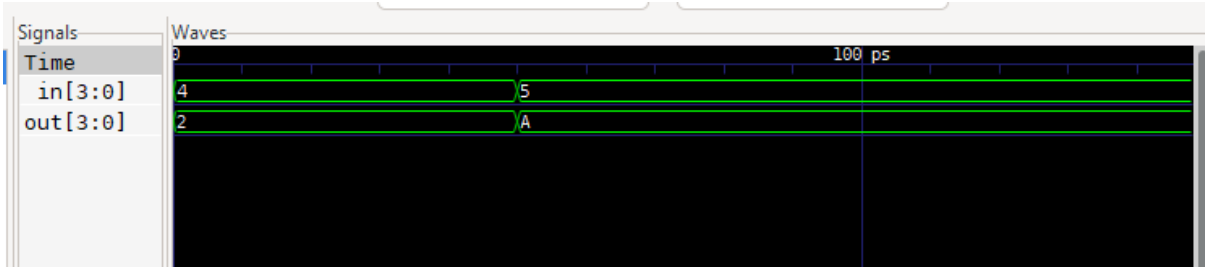
```
module bitreverse(in,out);

    parameter MAXBITS=32;

    input [MAXBITS-1:0] in;
    output reg [MAXBITS-1:0] out;


    reg a[MAXBITS-1:0];

    always @(in)
    begin
        rev(in,out);
    end

    task rev(input [MAXBITS:0] in,output [MAXBITS:0]
out);
```

```
        integer i;
        for (i=0; i<MAXBITS; i=i+1)
            begin
                out[i]=in[MAXBITS-1-i];
            end

    endtask

endmodule
```

Testbench for N=4:

```
`timescale 1ps/1ps

module testbench;
reg [3:0]in;
wire [3:0] out;
defparam DUT.MAXBITS=4;
bitreverse DUT(in,out);



initial begin
    in=4'b0100;
#50 in=4'b0101;
    #100 $finish;
end




initial begin
    $dumpfile("2.vcd");
```

```
    $dumpvars;
    $monitor("%g in=%32b out=%32b",$time,in,out);
end

endmodule
```

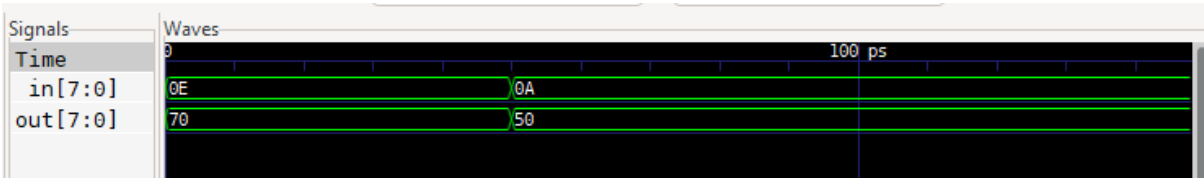


Testbench for N=8:

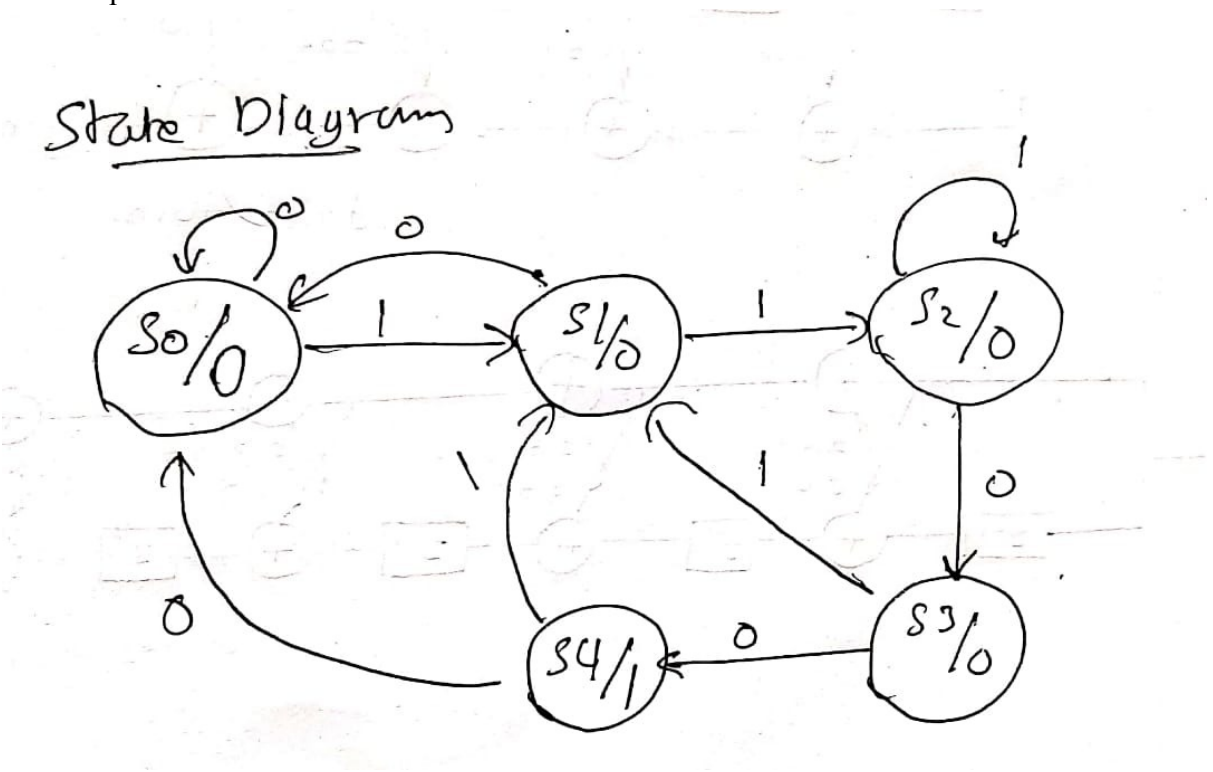
```
`timescale 1ps/1ps

module testbench;
reg [7:0]in;
wire [7:0] out;
defparam DUT.MAXBITS=8;
bitreverse DUT(in,out);


initial begin
    in=4'b0100_1110;
#50 in=4'b0101_1010;
    #100 $finish;
end




initial begin
```

```
    $dumpfile("2.vcd");
    $dumpvars;
    $monitor("%g in=%32b out=%32b",$time,in,out);
end

endmodule
```

Output:



3) 0011 Sequence Detector



State Diagram

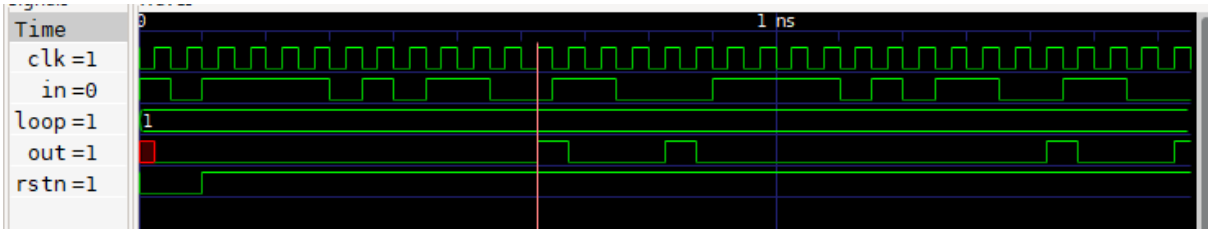
Code:

```
module fsm( input clk,
            input rstn,
            input in,
            output reg out);

    parameter   IDLE    =0,
                S1      =1,
                S11     =2,
                S110    =3,
                S1100   =4;

    reg[2:0] cur_state, next_state;
```

```
always@(cur_state)begin
    if (cur_state==S1100)
    out=1;
    else
    out=0;
end

always @(posedge clk) begin
    if (!rstn)
        cur_state=IDLE;
    else
        cur_state<=next_state;
end

always@ (cur_state or in ) begin
    case(cur_state)
            IDLE: begin
                if (in) next_state=S1;
                else next_state=IDLE;
            end
            S1: begin
                if (in) next_state=S11;
                else next_state=IDLE;
            end
            S11: begin
                if (in) next_state=S11;
                else next_state=S110;
            end
            S110: begin
                if (in) next_state=S1;
```

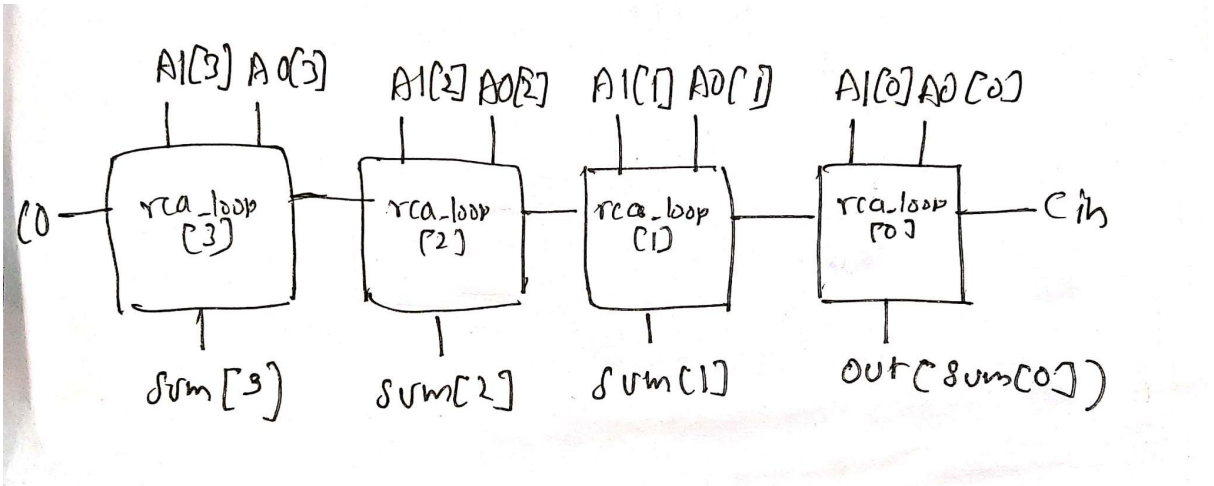
```
                    else next_state=S1100;
                end
                S1100: begin
                    if (in) next_state=S1;
                    else next_state=IDLE;
                end

            endcase

        end

    endmodule
```

Output:



4) N- bit Ripple carry adder

Circuit for N=4



Code:

```
module rca (output co, output [BITS-1:0] sum, input
[BITS-1:0] a0,a1,input ci);


parameter BITS=8;
wire [BITS:0] carry;
assign carry[0]=ci;



genvar i;
generate for (i=0 ; i<BITS; i=i+1) begin: rca_loop

    wire w1,w2,w3;
    xor x1( w1,a0[i],a1[i]);
    xor x2( sum[i], w1,carry[i]);
    and a1( w2,a0[i],a1[i]);
    and a2( w3,w1,carry[i]);
    or o1( carry[i+1],w2,w3);
end

endgenerate// end of the generate block

assign co=carry[BITS];

endmodule
```

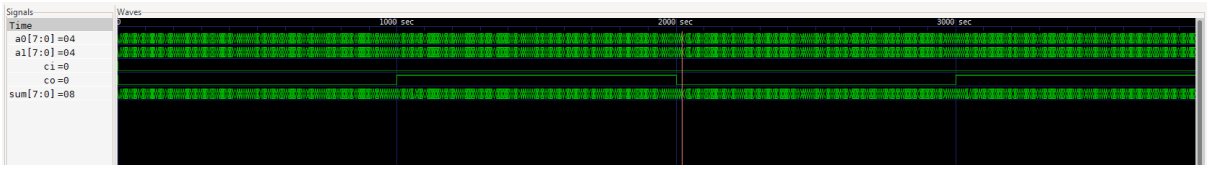
Testbench(N=8)

```
module testbench;

wire co;
wire [7:0] sum;
reg [7:0] a0,a1;
```

```
reg ci;

defparam DUT.BITS=8;

rca DUT(co, sum, a0, a1,ci);


initial
begin
    ci=0;
    a0=8'b0;
    a1=8'b0;

end

always #1000 a0[7]=~a0[7];
always #500 a0[6]=~a0[6];
always #250 a0[5]=~a0[5];
always #125 a0[4]=~a0[4];
always #62.5 a0[3]=~a0[3];
always #31.25 a0[2]=~a0[2];
always #15.625 a0[1]=~a0[1];
always #7.8125 a0[0]=~a0[0];

always #1000 a1[7]=~a1[7];
always #500 a1[6]=~a1[6];
always #250 a1[5]=~a1[5];
always #125 a1[4]=~a1[4];
always #62.5 a1[3]=~a1[3];
always #31.25 a1[2]=~a1[2];
always #15.625 a1[1]=~a1[1];
```

```
always #7.8125 a1[0]=~a1[0];


initial #4000 $finish;


initial begin
    $dumpfile("4.vcd");
    $dumpvars;
    $monitor("%g sum=%b co=%b a0=%8b
a1=%8b",$time,sum,co,a0,a1);
end



endmodule
```

Output:



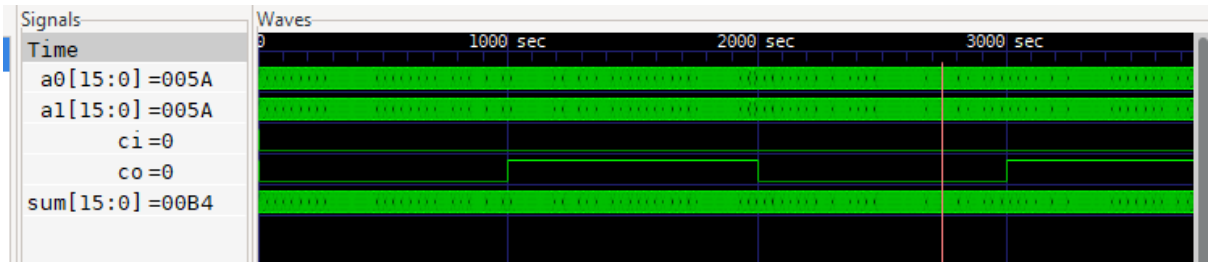
Testbench for N=16

```
module testbench;


wire co;
wire [15:0] sum;
reg [15:0] a0,a1;
reg ci;


defparam DUT.BITS=16;


rca DUT(co, sum, a0, a1,ci);
```

```
initial
begin
    ci=0;
    a0=16'b0;
    a1=16'b0;

end

always #1000 a0[15]=~a0[15];
always #1000 a0[14]=~a0[14];
always #1000 a0[13]=~a0[13];
always #1000 a0[12]=~a0[12];
always #1000 a0[11]=~a0[11];
always #1000 a0[10]=~a0[10];
always #1000 a0[9]=~a0[9];
always #1000 a0[8]=~a0[8];
always #1000 a0[7]=~a0[7];
always #500 a0[6]=~a0[6];
always #250 a0[5]=~a0[5];
always #125 a0[4]=~a0[4];
always #62.5 a0[3]=~a0[3];
always #31.25 a0[2]=~a0[2];
always #15.625 a0[1]=~a0[1];
always #7.8125 a0[0]=~a0[0];

always #1000 a1[15]=~a1[15];
always #1000 a1[14]=~a1[14];
always #1000 a1[13]=~a1[13];
always #1000 a1[12]=~a1[12];
always #1000 a1[11]=~a1[11];
always #1000 a1[10]=~a1[10];
```

```
always #1000 a1[9]=~a1[9];

always #1000 a1[8]=~a1[8];

always #1000 a1[7]=~a1[7];

always #500 a1[6]=~a1[6];

always #250 a1[5]=~a1[5];

always #125 a1[4]=~a1[4];

always #62.5 a1[3]=~a1[3];

always #31.25 a1[2]=~a1[2];

always #15.625 a1[1]=~a1[1];

always #7.8125 a1[0]=~a1[0];


initial #4000 $finish;


initial begin

    $dumpfile("4.vcd");

    $dumpvars;

    $monitor("%g sum=%b co=%b a0=%8b
a1=%8b",$time,sum,co,a0,a1);

end



endmodule
```

Output:



The End