RV College of Engineering ®, Bengaluru-59

(Autonomous Institution Affiliated to VTU)

Department of Electronics and Communication Engineering

16EC7G4 VLSI TESTING FOR ICs

Assignment Report

Verilog code for SISR, MISR and Cellular Automaton

Name of the students	USN
Mahesh Bhat K	1RV17EC069
Nagesh B	1RV17EC084

Faculty Details	
Faculty Name	Namita Palecha
Faculty Signature with date	

QUESTION 1:

Design the five-stage MISR shown using $f(x) = 1 + x + x^3 + x^4 + x^5$. Compute the fault-free signature for fault free sequence M_0 = {011011}, M1= {101101}, M2 = {010101}, M3 = {011110} and M4={100001}. Then, compute the signature for the faulty sequences M0' = {011100}, M1' = {111101}, M2' = {010111}, M3' = {011111},M4'={100001}. Explain if the faulty sequences are detected or not detected.

Verilog code:

```
module MISR(input m0,m1,m2,m3,m4,clk,rst, output reg q0,q1,q2,q3,q4);
   always@(posedge clk)
   begin
      if(rst) {q0,q1,q2,q3,q4} = 5'b0;
      else
      begin
         q0 <= q4^m0;
         q1 <= q4^q0^m1;
         q2 <= q1^m2;
         q3 <= q4^q2^m3;
         q4 <= q4^q3^m4;
      end
   end
endmodule
```

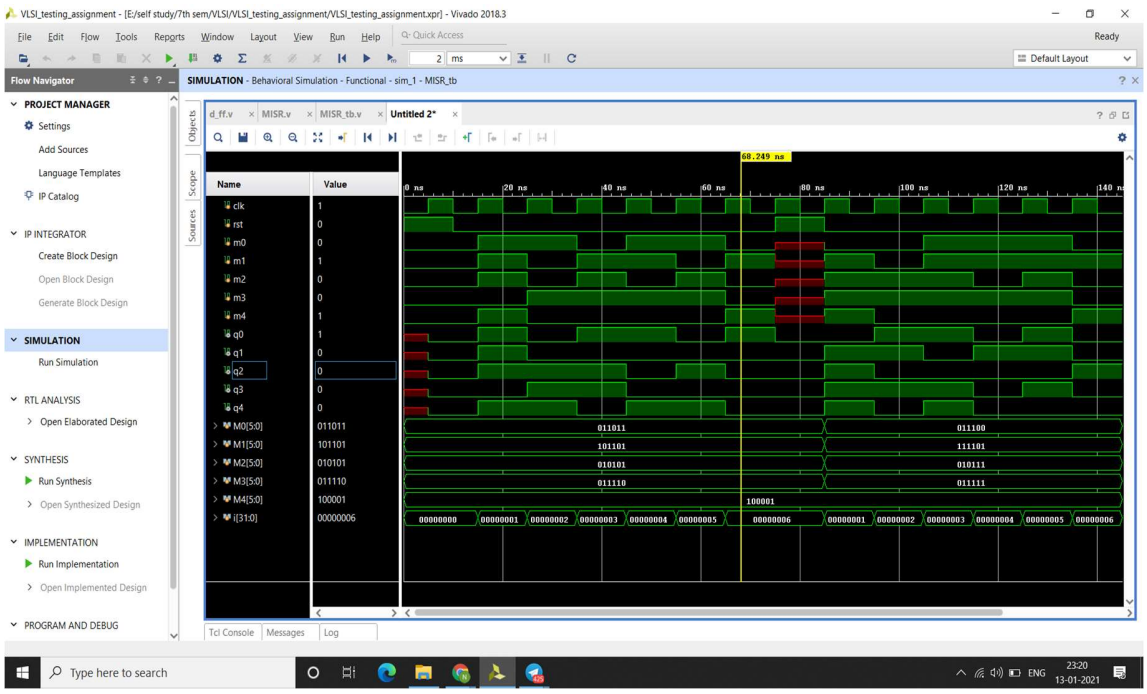
Test Bench:

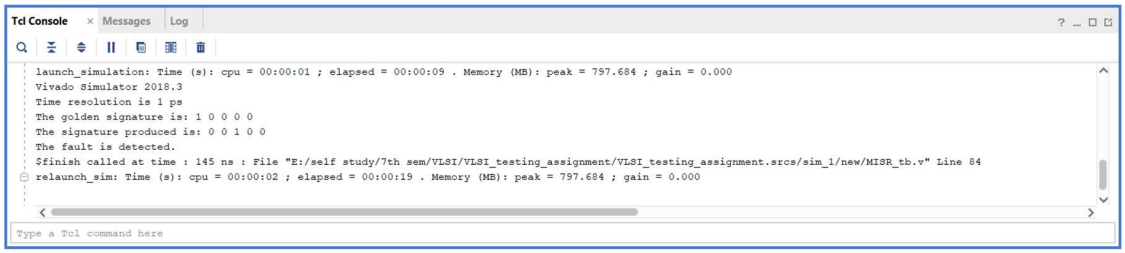
```
module MISR_tb();
reg clk,rst,m0,m1,m2,m3,m4;
wire q0,q1,q2,q3,q4;
reg [5:0]M0 = 6'b011011;
reg [5:0]M1 = 6'b101101;
reg [5:0]M2 = 6'b010101;
reg [5:0]M3 = 6'b011110;
reg [5:0]M4 = 6'b100001;
reg [4:0]gold_sig;
reg [4:0]sig;
integer i=0,j=0;

MISR dut(m0,m1,m2,m3,m4,clk,rst,q0,q1,q2,q3,q4);

initial
```

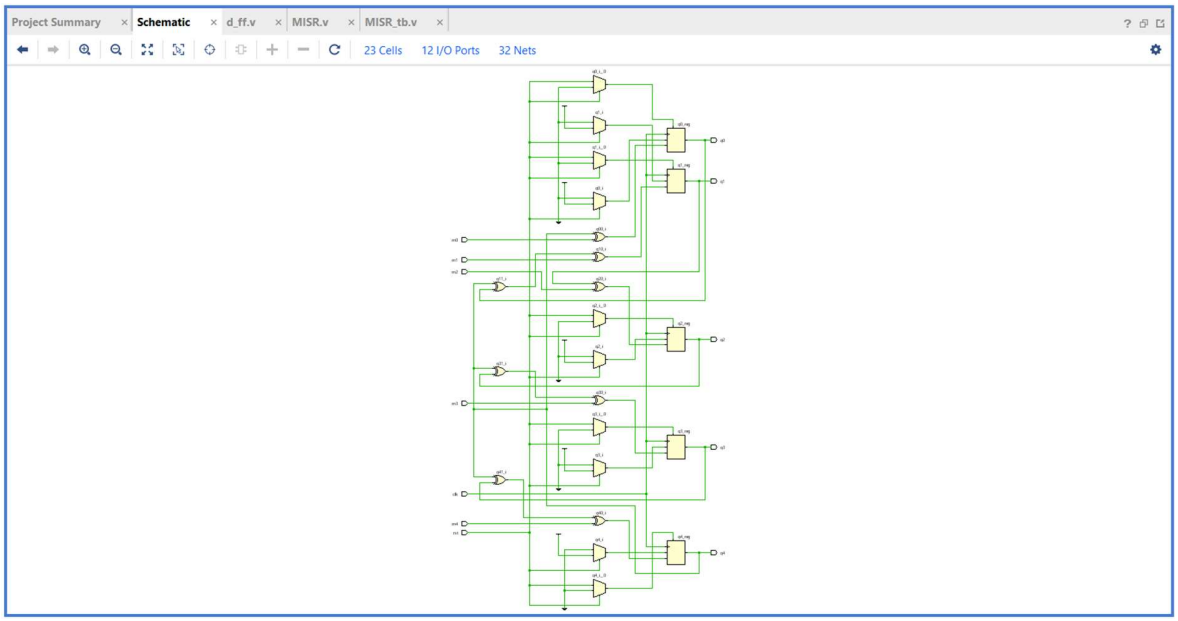
```
begin
   clk = 1'b0;
   rst = 1'b1;
   {m0,m1,m2,m3,m4} = 5'b0;
   #10 rst = 1'b0;
end

always
begin
   #5 clk = ~clk;
end

always@(posedge clk)
begin
   if(rst == 1'b0)
   begin
      if(j == 1)
      begin
         M0 = 6'b011100;
         M1 = 6'b111101;
         M2 = 6'b010111;
         M3 = 6'b011111;
         M4 = 6'b100001;
      end
      m0=M0[i];
      m1=M1[i];
      m2=M2[i];
      m3=M3[i];
      m4=M4[i];
      if(i==6 && j==0)
      begin
         gold_sig = {q0,q1,q2,q3,q4};
         $display("The golden signature is: %b %b %b %b %b",q0,q1,q2,q3,q4);
         j=1;
         rst=1;
         #10 rst=0;
         i=-1;
         //$finish;
      end
      if(j==1 && i==6)
      begin
```

```
        sig = {q0,q1,q2,q3,q4};
        $display("The signature produced is: %b %b %b %b %b",q0,q1,q2,q3,q4);
        if (gold_sig == sig) $display("The fault is undetected.");
        else $display("The fault is detected.");
        $finish;
      end
      i=i+1;
   end
end
endmodule
```

Simulation Results:







Inference:

- MISR is used as an Output Response Analyser.
- It uses basic LFSR and additional XOR gates to generate signatures.
- The Probability of aliasing is approximately $1/2^n$.

QUESTION 2:

Design a 4-bit maximal LFSR for primitive polynomial $1 + x + x^4$. Write the system of equation with the companion matrix for this LFSR. Compute the pattern generated by this LFSR if initialized as "0001", with the '1' in the least significant bit.

Add hardware to map the test-pattern "0011," which is not useful, into the pattern "0000," which detects several circuit faults.

Verilog code:

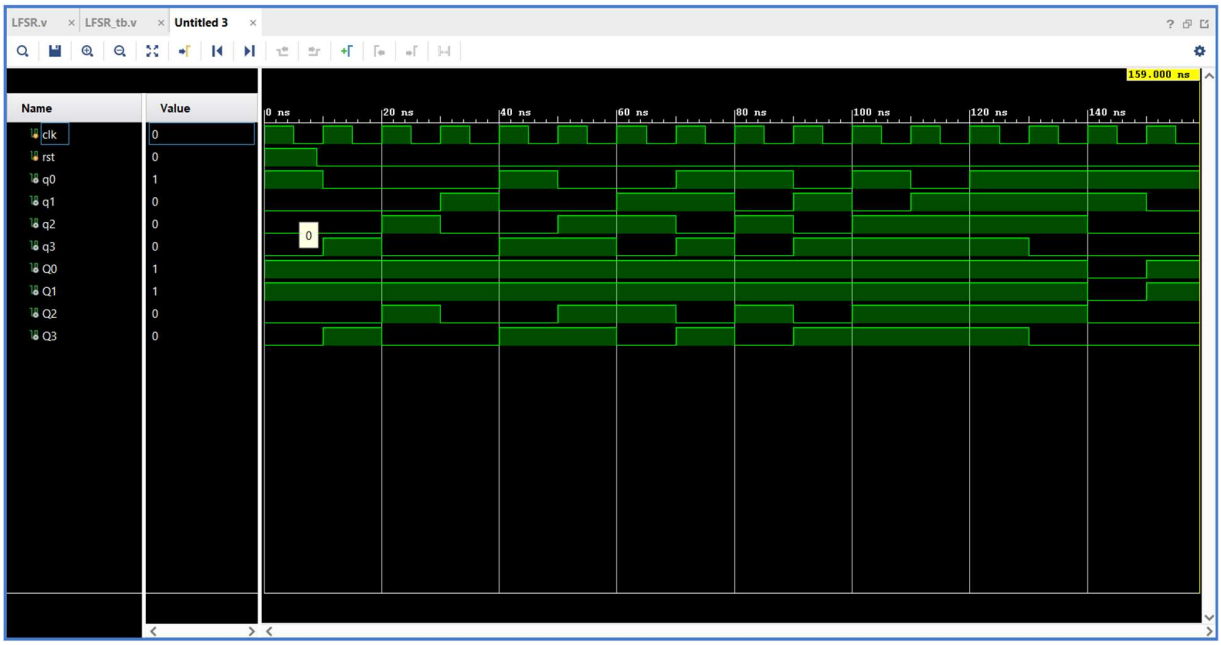
```
module LFSR(input rst,clk, output reg q0,q1,q2,q3, output Q0,Q1,Q2,Q3);
always@(posedge clk)
begin
   if(rst) {q0,q1,q2,q3} <= 4'b1000;
   else
   begin
      q0 <= q1;
      q1 <= q2;
```

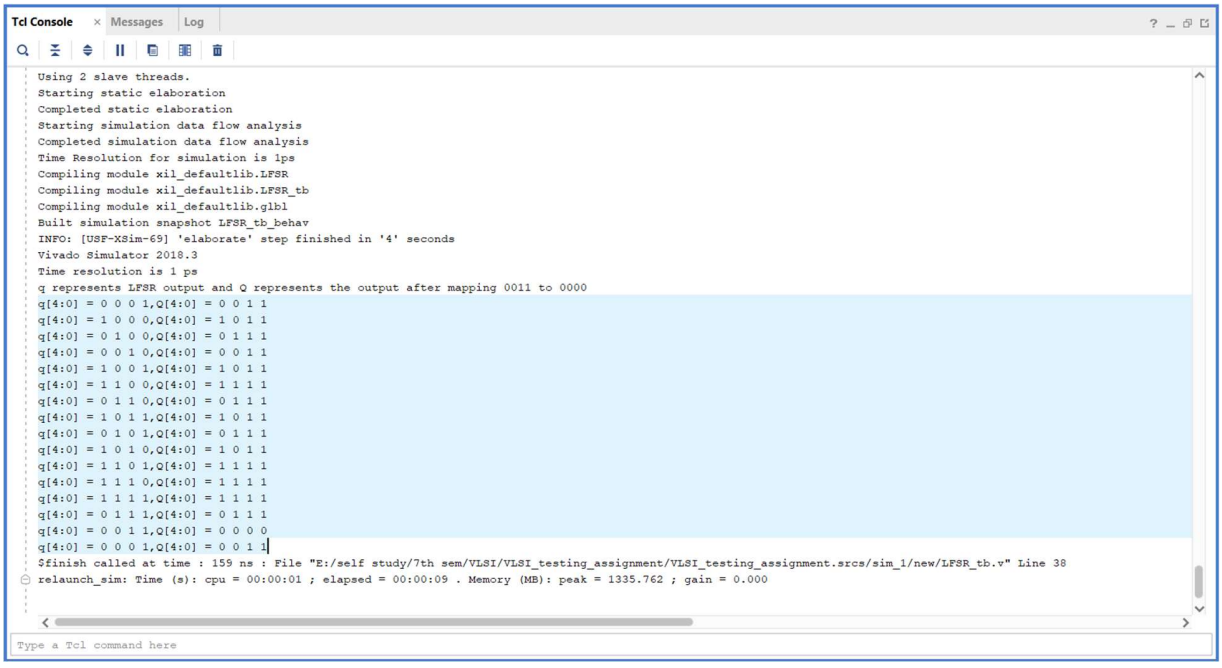
```
      q2 <= q3;
      q3 <= q0^q1;
   end
end
//external hardware to map 0011 to 0000
assign Q0 = ~(q0 && q1 && ~q2 && ~q3);
assign Q1 = ~(q0 && q1 && ~q2 && ~q3);
assign Q2 = q2;
assign Q3 = q3;
endmodule
```

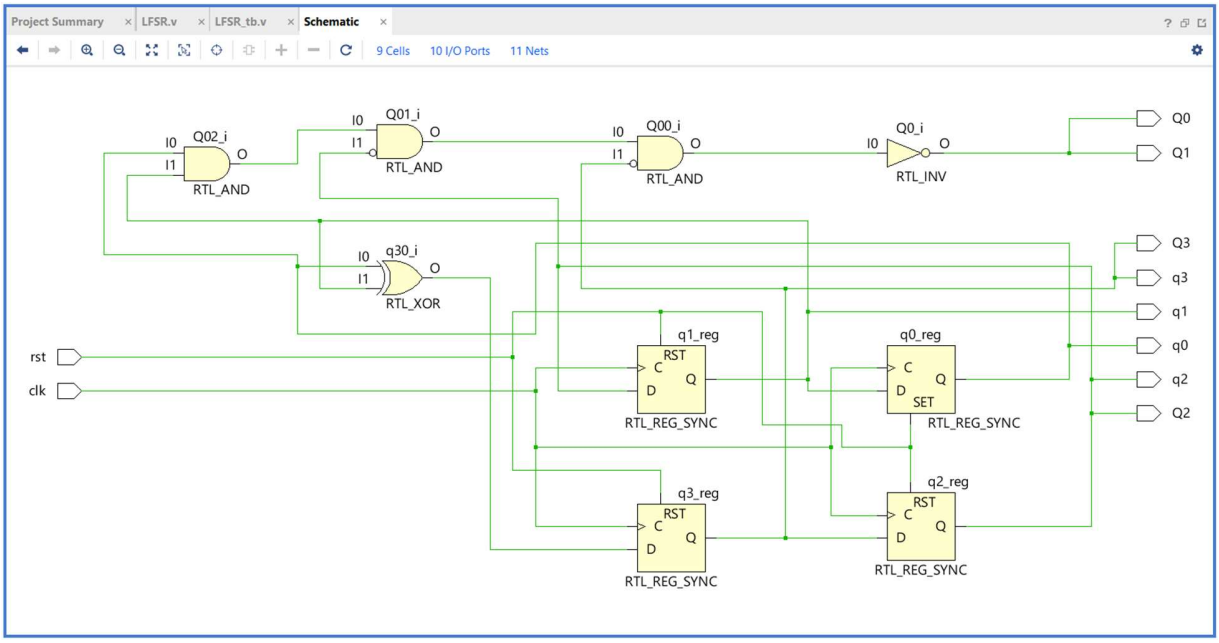
Test bench:

```
module LFSR_tb();
reg clk,rst;
wire q0,q1,q2,q3,Q0,Q1,Q2,Q3;

LFSR dut(rst,clk,q0,q1,q2,q3,Q0,Q1,Q2,Q3);

always
   #5 clk=~clk;
initial
begin
   clk = 1'b1;
   rst = 1'b1;
   #9 rst = 1'b0;
   $display("q represents LFSR output and Q represents the output after mapping 0011 to 0000");
   $monitor("q[4:0] = %b %b %b %b,Q[4:0] = %b %b %b %b",q3,q2,q1,q0,Q3,Q2,Q1,Q0);
   #150 $finish;
end
endmodule
```

Result:


LFSR.v
LFSR_tb.v
Untitled 3
Name
Value
clk
rst
q0
q1
q2
q3
Q0
Q1
Q2
Q3
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
159.000 ns


Tcl Console
Messages
Log
Using 2 slave threads.
Starting static elaboration
Completed static elaboration
Starting simulation data flow analysis
Completed simulation data flow analysis
Time Resolution for simulation is 1ps
Compiling module xil_defaultlib.LFSR
Compiling module xil_defaultlib.LFSR_tb
Compiling module xil_defaultlib.glbl
Built simulation snapshot LFSR_tb_behav
INFO: [USF-XSim-69] 'elaborate' step finished in '4' seconds
Vivado Simulator 2018.3
Time resolution is 1 ps
q represents LFSR output and Q represents the output after mapping 0011 to 0000
q[4:0] = 0 0 0 1,Q[4:0] = 0 0 1 1
q[4:0] = 1 0 0 0,Q[4:0] = 1 0 1 1
q[4:0] = 0 1 0 0,Q[4:0] = 0 1 1 1
q[4:0] = 0 0 1 0,Q[4:0] = 0 0 1 1
q[4:0] = 1 0 0 1,Q[4:0] = 1 0 1 1
q[4:0] = 1 1 0 0,Q[4:0] = 1 1 1 1
q[4:0] = 0 1 1 0,Q[4:0] = 0 1 1 1
q[4:0] = 1 0 1 1,Q[4:0] = 1 0 1 1
q[4:0] = 0 1 0 1,Q[4:0] = 0 1 1 1
q[4:0] = 1 0 1 0,Q[4:0] = 1 0 1 1
q[4:0] = 1 1 0 1,Q[4:0] = 1 1 1 1
q[4:0] = 1 1 1 0,Q[4:0] = 1 1 1 1
q[4:0] = 1 1 1 1,Q[4:0] = 1 1 1 1
q[4:0] = 0 1 1 1,Q[4:0] = 0 1 1 1
q[4:0] = 0 0 1 1,Q[4:0] = 0 0 0 0
q[4:0] = 0 0 0 1,Q[4:0] = 0 0 1 1
$finish called at time : 159 ns : File "E:/self study/7th sem/VLSI/VLSI_testing_assignment/VLSI_testing_assignment.srcs/sim_1/new/LFSR_tb.v" Line 38
relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:09 . Memory (MB): peak = 1335.762 ; gain = 0.000
Type a Tcl command here



Inference:

- LFSR generates deterministic patterns. Hence the fault coverage may be low.
- LFSR designed with primitive polynomial generate 2^n-1 patterns.
- For non-primitive polynomials, the number of patterns generated is less than 2^n-1 patterns.

Seed value also plays an important role in the number of patterns generated.

- LFSR is simple and easy to design.

QUESTION 3:

Build a four flip-flop rule *cellular automaton* (CA),using rule 90 and compute its pattern sequence. Seed the CA pattern generator with "0001."What is the period of the cellular automaton?

Verilog Code:

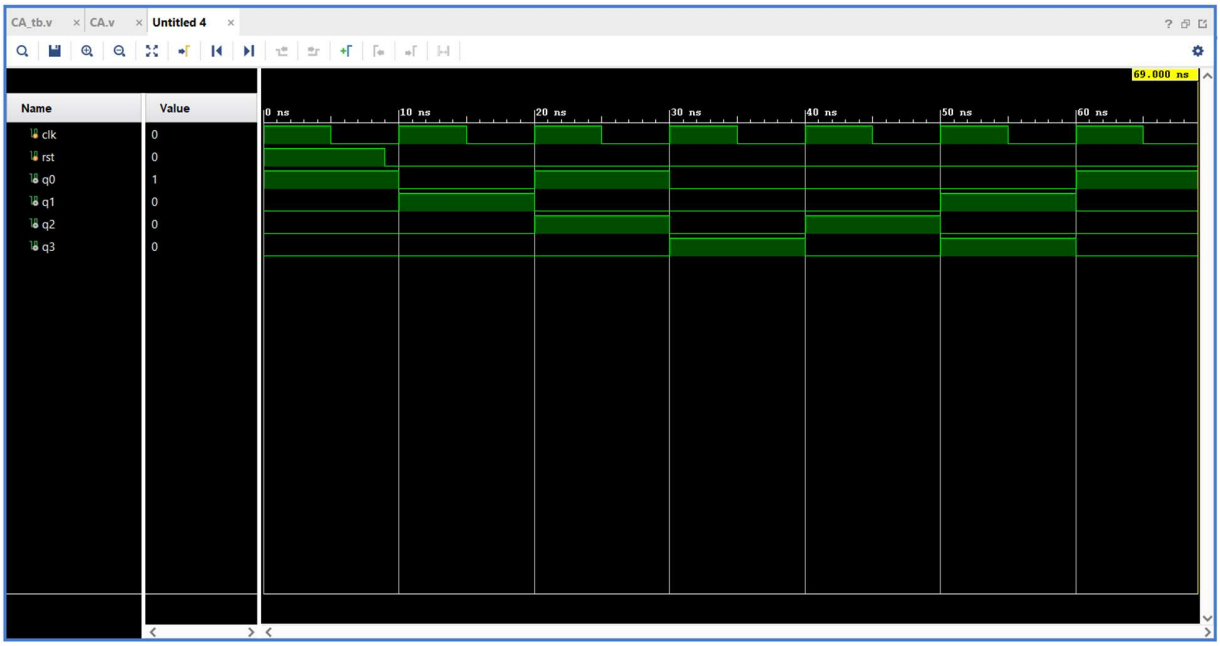
```
module CA(input rst,clk, output reg q0,q1,q2,q3);
always@(posedge clk)
begin
   if(rst) {q0,q1,q2,q3} <= 4'b1000;
   else
   begin
      q0 <= q1;
      q1 <= q0^q2;
      q2 <= q1^q3;
      q3 <= q2;
```

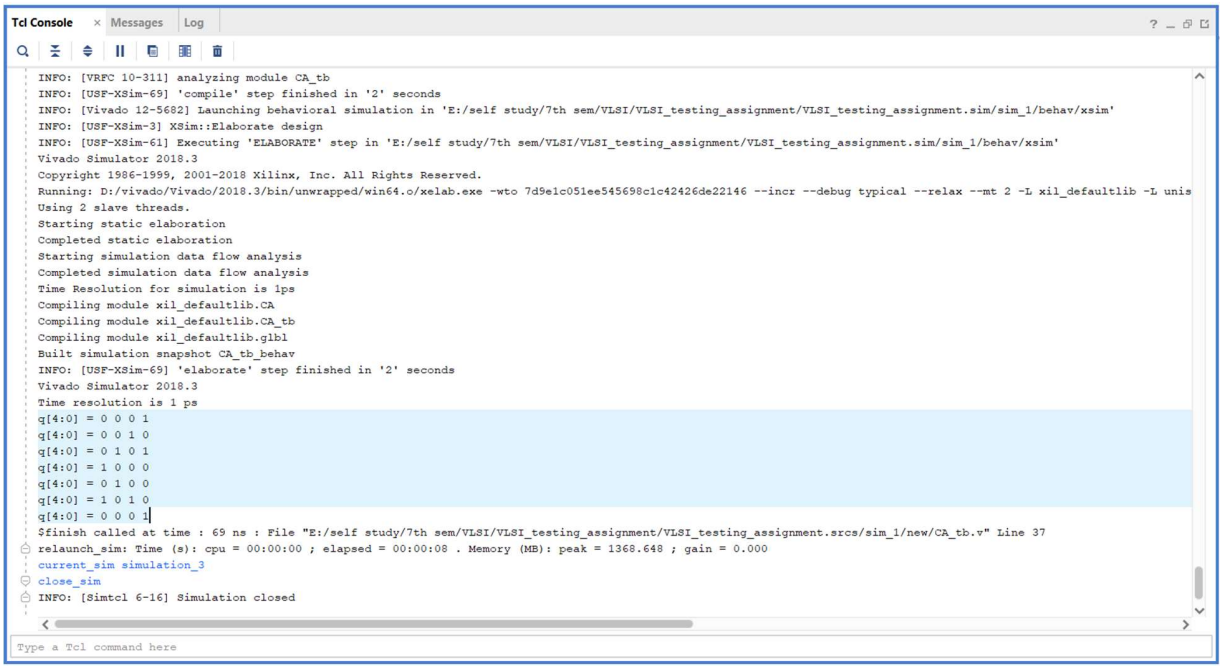
```
    end
end
endmodule
```

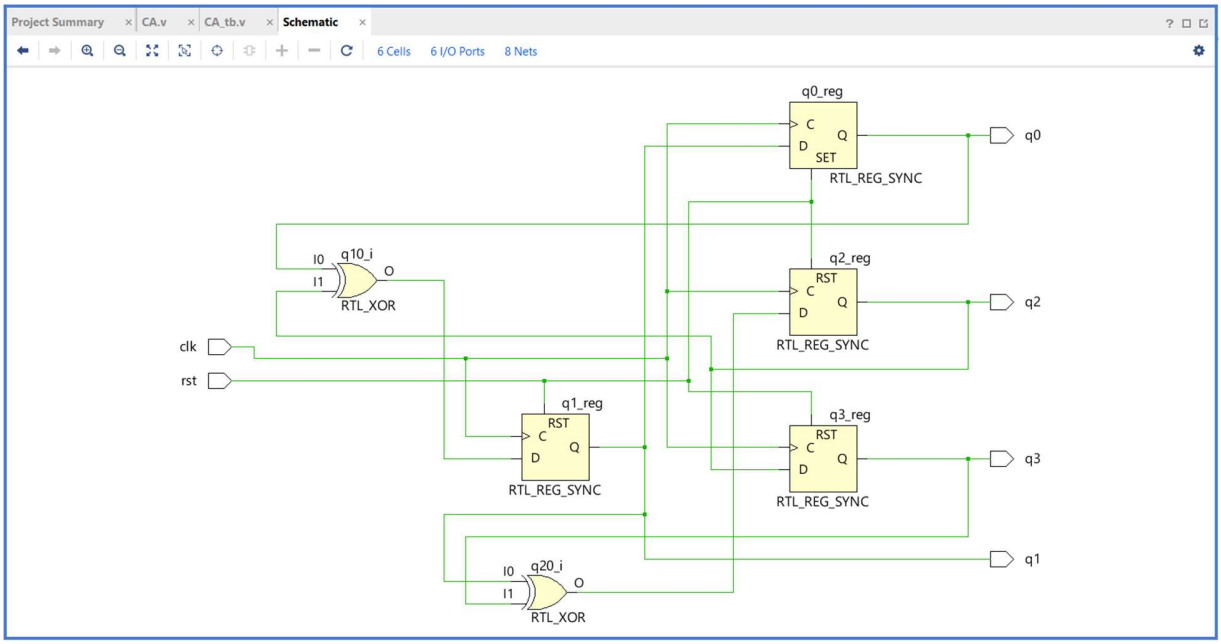
Testbench:

```
module CA_tb();
reg clk,rst;
wire q0,q1,q2,q3;

CA dut(rst,clk,q0,q1,q2,q3);

always
    #5 clk=~clk;
initial
begin
    clk = 1'b1;
    rst = 1'b1;
    #9 rst = 1'b0;
    $monitor("q[4:0] = %b %b %b %b",q3,q2,q1,q0);
    #60 $finish;
end
endmodule
```

Results:







Inference:

- Using Cellular Automaton, more patterns which are random in nature can be generated. Hence, high fault coverage is obtained.
- The period of the above design is 6.
- By choosing an appropriate seed value and rule used in design, the period can be increased.

QUESTION 4:

Consider the circuit implementing the function f=a+a'c' and an input sequence consisting of all eight input patterns generated by 3-bit counter that starts in the state (a,b,c)=(0,0,0) and produces the sequence (0,0,0),(0,0,1),(0,1,0)...(1,1,1). Assume the response is compressed using a modular based signature analyser that has characteristics polynomial $x^5 + x^4 + x^2 + 1$ and whose initial state is all 0's.For the given specification:

Design SISR output response analyser.

Verilog Code:

```
module sisr(input clk,rst,start_counter, output reg q0,q1,q2,q3,q4,end_signal);
reg [3:0]counter;
wire a,b,c;
wire f;
reg [7:0]y;
reg [3:0]i;
reg sisr;
always@(posedge clk)
begin
if(rst==0)
   begin y<=8'b0;
        q0<=0;q1<=0;q2<=0;q3<=0;q4<=0;
        counter<=3'b0;end_signal=0;
        sisr=0;
   end
if(start_counter==1&& (~sisr))
   begin   counter<=counter+1;
          if(counter==3'b111)
          begin
             counter<=3'b0;
```

```
            sisr<=1;
        end
    end

end
assign a=counter[2];
assign b=counter[1];
assign c=counter[0];
assign f= (a ||(~a&&~c));

always@(posedge clk)
case(counter)
    0:y[0]<=f;
    1:y[1]<=f;
    2:y[2]<=f;
    3:y[3]<=f;
    4:y[4]<=f;
    5:y[5]<=f;
    6:y[6]<=f;
    7:y[7]<=f;
    default: y<=7'b0;
endcase

always@(posedge clk)
begin
if(rst==0) begin
    q0<=0;q1<=0;q2<=0;q3<=0;q4<=0;i<=0;
```

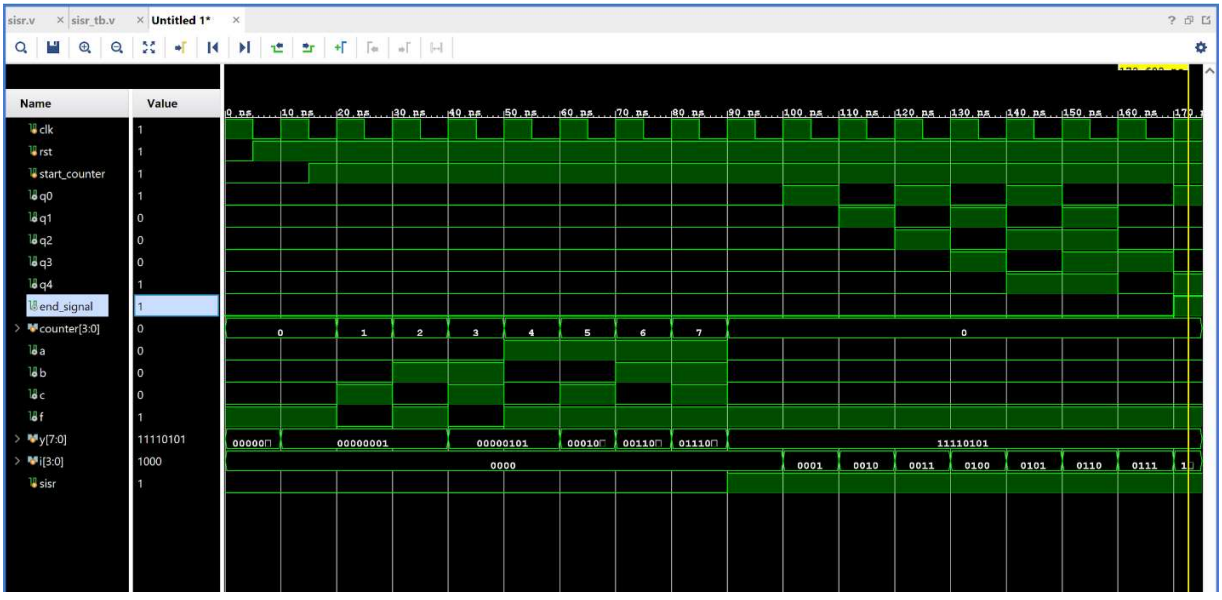
```
        end
if(sisr==1 )
begin
   q0<=q4^y[i];
   q1<=q0;
   q2<=q4^q1;
   q3<=q2;
   q4<=q4^q3;
   i<=i+1;
   if(i==3'b111)
   begin
      end_signal<=1'b1;
   end
end
end
endmodule
```

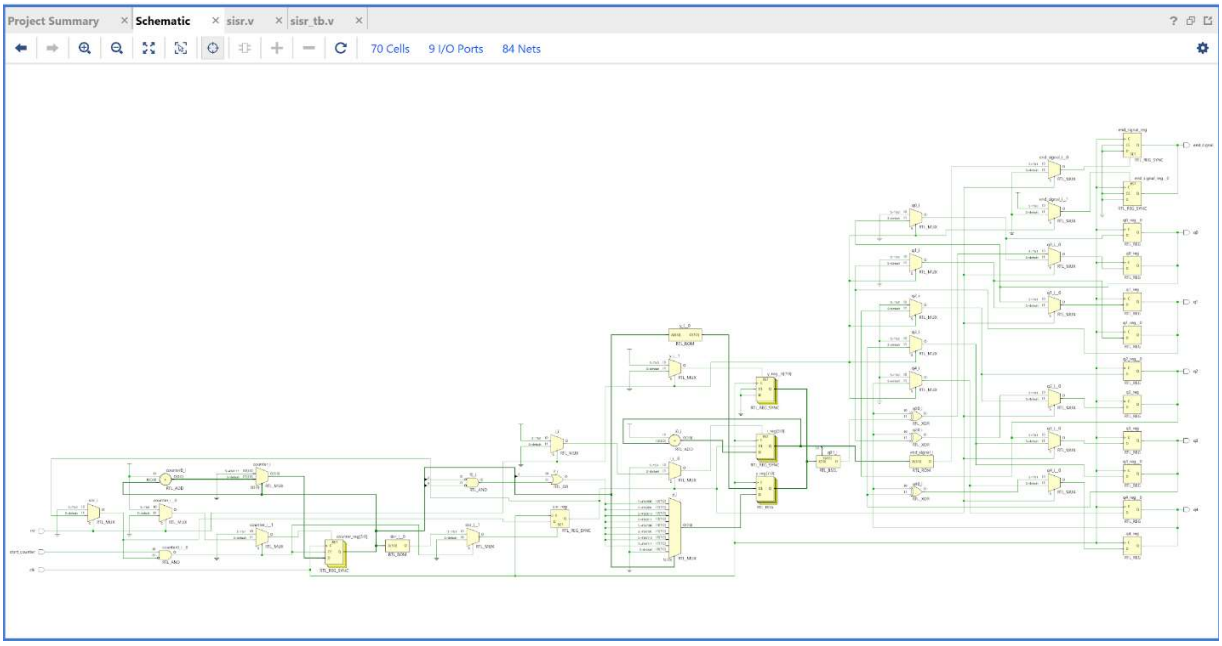
Test Bench:

```
module sisr_tb(   );
reg clk,rst,start_counter;
wire q0,q1,q2,q3,q4,end_signal;

sisr dut(clk,rst,start_counter,q0,q1,q2,q3,q4,end_signal);
initial
begin
clk=1;
rst=0;
```

```
start_counter=0;
#5 rst=1;
#10 start_counter=1;
end
always
#5 clk=~clk;
always@(end_signal)
if(end_signal)
begin
   $display("q0,q1,q2,q3,q4=%b %b %b %b %b ",q0,q1,q2,q3,q4);
   #5 $finish;
end
endmodule
```

Results:





Inferences:

- SISR is used as an Output Response Analyser.
- It uses basic LFSR and one extra XOR gate to generate signatures.
- The Probability of aliasing is approximately $1/2^n$.