Experiment 6: Exercise with Verilog Simulations – Counters

1. Synchronous 4-bit Gray-Code Counter:

A Gray-Code counter counts in such a manner that the difference between any two consecutive states only differs by 1 bit. It is used in a variety of scenarios especially where bit variations/state changes are prohibitive in terms of resource costs.

State Table:-
In the following table, I represent the gray code as a 4 bit sequence of Q1Q2Q3Q4 where Q1 is the MSB while Q4 is the LSB

Present State				Next State				S1 R1	S2 R2	S3 R3	S4 R4
Q1	Q2	Q3	Q4	Q1'	Q2'	Q3'	Q4'				
0	0	0	0	0	0	0	1	0 x	0 x	0 x	1 0
0	0	0	1	0	0	1	1	0 x	0 x	1 0	x 0
0	0	1	1	0	0	1	0	0 x	0 x	x 0	0 1
0	0	1	0	0	1	1	0	0 x	1 0	x 0	0 x
0	1	1	0	0	1	0	0	0 x	x 0	0 1	0 x
0	1	0	0	0	1	0	1	0 x	x 0	0 x	1 0
0	1	0	1	0	1	1	1	0 x	x 0	1 0	x 0
0	1	1	1	1	1	1	1	1 0	x 0	x 0	x 0
1	1	1	1	1	1	1	0	x 0	x 0	x 0	0 1
1	1	1	0	1	1	0	0	x 0	x 0	0 1	0 x
1	1	0	0	1	1	0	1	x 0	x 0	0 x	1 0
1	1	0	1	1	0	0	1	x 0	0 1	0 x	x 0
1	0	0	1	1	0	1	1	x 0	0 x	1 0	x 0
1	0	1	1	1	0	1	0	x 0	0 x	x 0	0 1
1	0	1	0	1	0	0	0	x 0	0 x	0 1	0 x
1	0	0	0	0	0	0	0	0 1	0 x	0 x	0 x

We see that we need 16 states and therefore, we will be requiring $\log_2 16$ = **4 SR-flip flops** since we are making a 4-bit gray-code counter. Following the table, we will now make K-maps to find a reduced expression for S_i and R_i for i={1,2,3,4}
In the following K-Maps, for simplicity,
A=Q1
B=Q2

C=Q3
D=Q4

S1: BCD

Map

	C'.D'	C'.D	C.D	C.D'
A'.B'	0	0	0	0
A'.B	0	0	1	0
A.B	x	x	x	X
A.B'	0	x	x	X

Groups

(7,15)	B.C.D

R1: B'C'D'

Map

	C'.D'	C'.D	C.D	C.D'
A'B'	x	x	x	X
A'B	x	x	0	X
A.B	0	0	0	0
A.B'	1	0	0	0

Groups

(0,8)	B'.C'.D'

S2: A'.C.D'

Map

	C'.D'	C'.D	C.D	C.D'

A'.B'	0	0	0	1
A'.B	x	x	x	X
A.B	x	x	x	X
A.B'	0	0	0	0

Groups

(2,6)	A'.C.D'

R2: A.C'.D

Map

	C'.D'	C'.D	C.D	C.D'
A'.B'	x	x	x	0
A'.B	0	0	0	0
A.B	0	1	0	0
A.B'	x	x	x	X

Groups

(9,13)	A.C'.D

S3: A'.D + B'.D

Map

	C'.D'	C'.D	C.D	C.D'
A'.B'	0	1	x	X
A'.B	0	1	x	0
A.B	0	0	x	0
A.B'	0	1	x	0

Groups

(1,3,5,7)	A'.D

(1,3,9,11)	B'.D

R3: B.D' + A.D'

Map

	C'.D'	C'.D	C.D	C.D'
A'.B'	x	0	0	0
A'.B	x	0	0	1
A.B	x	x	0	1
A.B'	x	0	0	1

Groups

(4,6,12,14)	B.D'
(8,10,12,14)	A.D'

S4: A'.C' + B.C'

Map

	C'.D'	C'.D	C.D	C.D'
A'.B'	1	x	0	0
A'.B	1	x	x	0
A.B	1	x	0	0
A.B'	0	x	0	0

Groups

(0,1,4,5)	A'.C'
(4,5,12,13)	B.C'

R4: B'C + AC

Map

	C'.D'	C'.D	C.D	C.D'
A'.B'	0	0	1	x
A'.B	0	0	0	x
A.B	0	0	1	x
A.B'	x	0	1	x

Groups

(2,3,10,11)	B'.C
(10,11,14,15)	A.C

Therefore, for our purpose:

S1 = q2.q3.q4

S2 = q1'.q3.q4'

S3 = (q1'.q4) + (q2'.q4)

S4 = q3'.(q1'+q2)

R1 = q2'.q3'.q4'

R2 = q1.q3'.q4

R3 = (q1.q4')+(q2.q4')

R4 = q3.(q1+q2')

Verilog Code:

Note: In my codes, MSB start from 1 i.e. for 4 bits Q, Q_1 is MSB and Q_4 is LSB

1. SR_new.v

```
module SR_new(S, R, clk, Q, rst);
    input S, R, clk, rst;
    output reg Q;

    always @(posedge clk) begin
        if (rst) begin
          Q <= 0;
        end
        else if(S == 1) begin
            Q <= 1;
        end
        else if(R == 1) begin
            Q <= 0;
        end
        else if(S == 0 & R == 0) begin
            Q <= Q;
        end
    end
endmodule
```

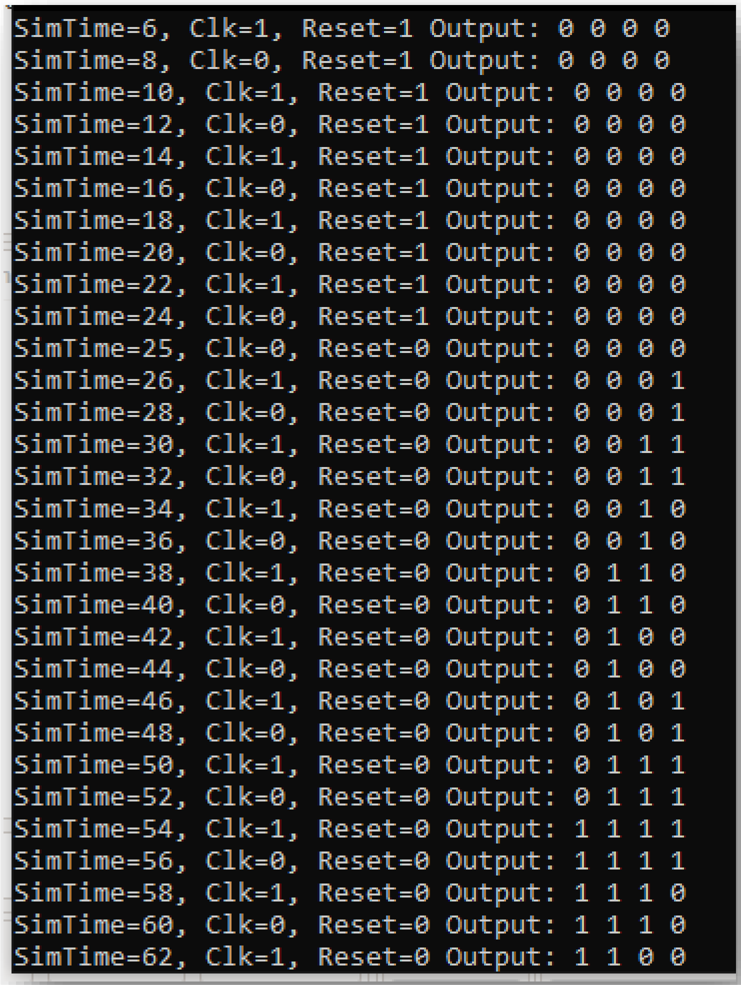
2. GCC.v

```
module GCC(clk, q1, q2, q3, q4, rst);
    input clk, rst;
    output q1, q2, q3, q4;
    wire s1,r1,s2,r2,s3,r3,s4,r4;

    assign s1 = q2 && q3 && q4;
    assign s2 = ~q1 && q3 && ~q4;
    assign s3 = (~q1 && q4) || (~q2 && q4);
    assign s4 = ~q3 && (~q1 || q2);

    assign r1 = ~q2 && ~q3 && ~q4;
    assign r2 = q1 && ~q3 && q4;
    assign r3 = (q1 && ~q4) || (q2 && ~q4);
    assign r4 = q3 && (q1 || ~q2);

    SR_new S1(s1, r1, clk, q1, rst);
    SR_new S2(s2, r2, clk, q2, rst);
    SR_new S3(s3, r3, clk, q3, rst);
    SR_new S4(s4, r4, clk, q4, rst);

endmodule
```

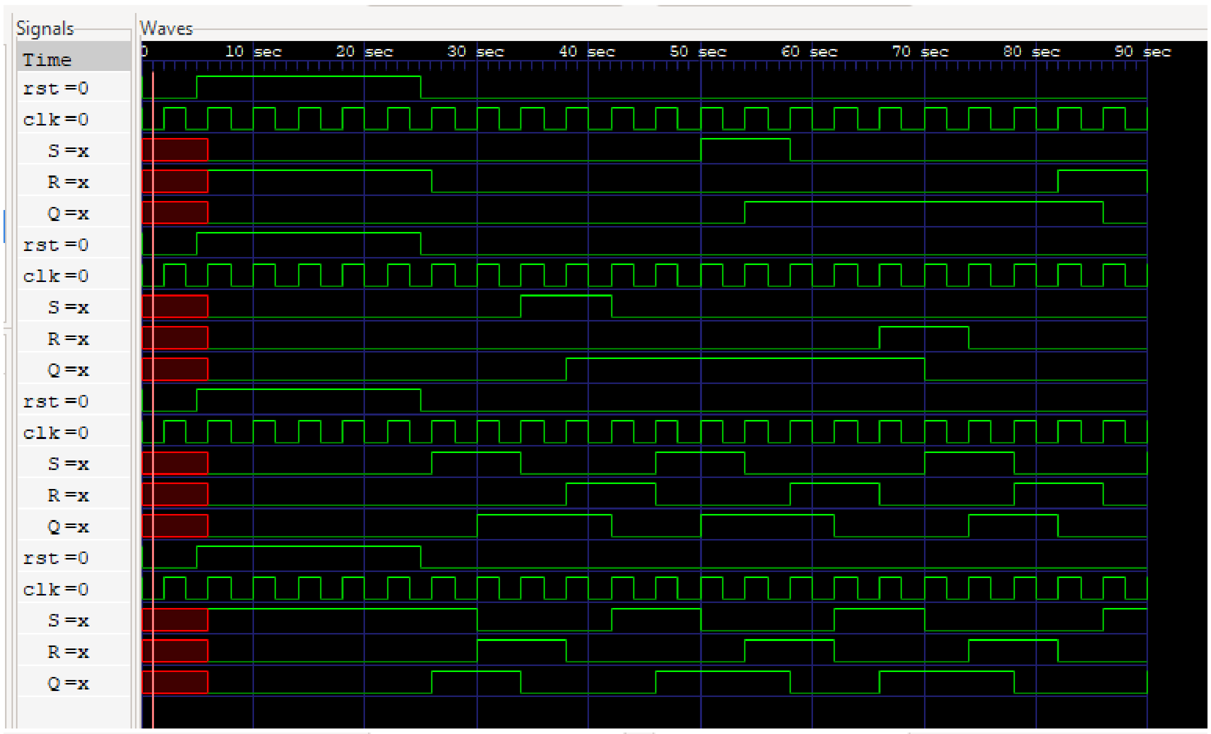
3. tb_GCC.v

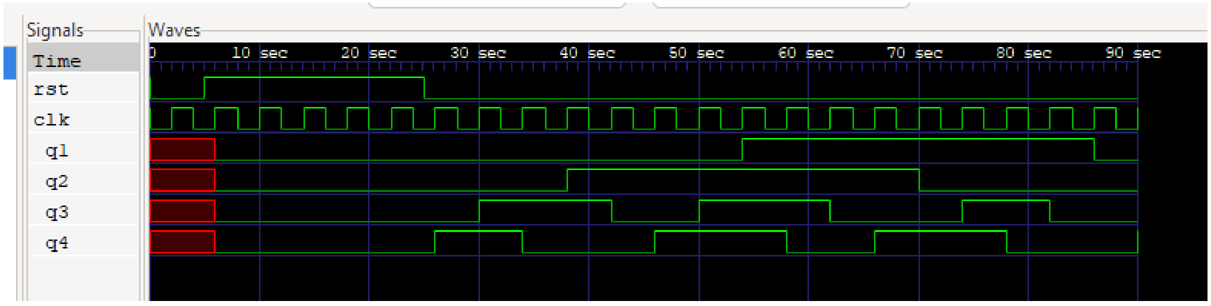
```
module tb_GCC;
    wire q1, q2, q3, q4;
    reg clk, rst;

    GCC DUT(.clk(clk), .q1(q1), .q2(q2), .q3(q3), .q4(q4), .rst(rst));

    always #2 clk = ~clk;
    initial begin
        clk = 1'b0;
        rst = 0;
        #5 rst = 1'b1;
        #20 rst = 1'b0;
    end

    initial begin
        $dumpfile("GCC.vcd");
        $dumpvars(0, tb_GCC);
        $monitor("SimTime=%g, Clk=%b, Reset=%b Output: %b %b %b %b", $time,
 clk, rst, q1, q2, q3, q4);
        #90 $finish;
    end
endmodule
```

Outputs and Waveforms:



```
SimTime=6, Clk=1, Reset=1 Output: 0 0 0 0
SimTime=8, Clk=0, Reset=1 Output: 0 0 0 0
SimTime=10, Clk=1, Reset=1 Output: 0 0 0 0
SimTime=12, Clk=0, Reset=1 Output: 0 0 0 0
SimTime=14, Clk=1, Reset=1 Output: 0 0 0 0
SimTime=16, Clk=0, Reset=1 Output: 0 0 0 0
SimTime=18, Clk=1, Reset=1 Output: 0 0 0 0
SimTime=20, Clk=0, Reset=1 Output: 0 0 0 0
SimTime=22, Clk=1, Reset=1 Output: 0 0 0 0
SimTime=24, Clk=0, Reset=1 Output: 0 0 0 0
SimTime=25, Clk=0, Reset=0 Output: 0 0 0 0
SimTime=26, Clk=1, Reset=0 Output: 0 0 0 1
SimTime=28, Clk=0, Reset=0 Output: 0 0 0 1
SimTime=30, Clk=1, Reset=0 Output: 0 0 1 1
SimTime=32, Clk=0, Reset=0 Output: 0 0 1 1
SimTime=34, Clk=1, Reset=0 Output: 0 0 1 0
SimTime=36, Clk=0, Reset=0 Output: 0 0 1 0
SimTime=38, Clk=1, Reset=0 Output: 0 1 1 0
SimTime=40, Clk=0, Reset=0 Output: 0 1 1 0
SimTime=42, Clk=1, Reset=0 Output: 0 1 0 0
SimTime=44, Clk=0, Reset=0 Output: 0 1 0 0
SimTime=46, Clk=1, Reset=0 Output: 0 1 0 1
SimTime=48, Clk=0, Reset=0 Output: 0 1 0 1
SimTime=50, Clk=1, Reset=0 Output: 0 1 1 1
SimTime=52, Clk=0, Reset=0 Output: 0 1 1 1
SimTime=54, Clk=1, Reset=0 Output: 1 1 1 1
SimTime=56, Clk=0, Reset=0 Output: 1 1 1 1
SimTime=58, Clk=1, Reset=0 Output: 1 1 1 0
SimTime=60, Clk=0, Reset=0 Output: 1 1 1 0
SimTime=62, Clk=1, Reset=0 Output: 1 1 0 0
```


Signals
Waves
Time
rst =0
clk =0
S =x
R =x
Q =x
rst =0
clk =0
S =x
R =x
Q =x
rst =0
clk =0
S =x
R =x
Q =x
rst =0
clk =0
S =x
R =x
Q =x
0
10 sec
20 sec
30 sec
40 sec
50 sec
60 sec
70 sec
80 sec
90 sec


Signals
Waves
Time
rst
clk
q1
q2
q3
q4
0
10 sec
20 sec
30 sec
40 sec
50 sec
60 sec
70 sec
80 sec
90 sec

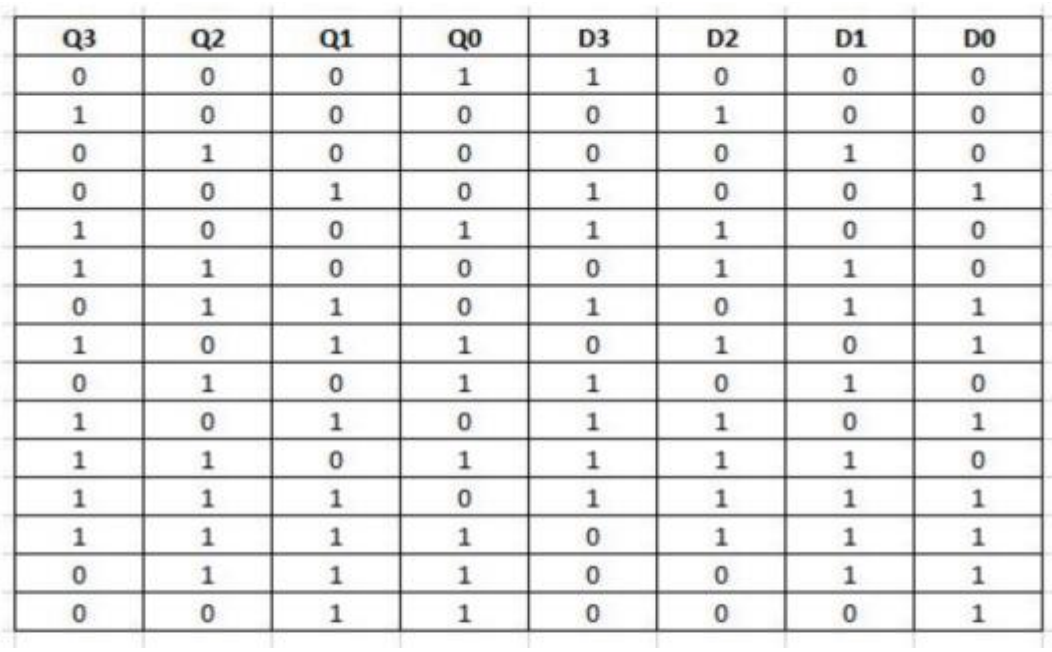
2. Synchronous Ring Counter

A Ring counter behaves like a Shift-Register, except that the first Flip-Flop gets an input which can be a combination of any of the current states. Johnson Counter that is covered in class, is an example of a Ring Counter.

Since we are covering 4-bit counter, we will be requiring a total of **4 D-flip flops** to implement the circuit**.** Following the general ring counter, and the disclaimer in the question, we only check the input for our first D flip flop. In a ring counter, it is just the output of the last flip flop, in Johnson counter, it is complement of that output whereas, by the table given, we deduce that it is q_2 xor q_3 (where the first flip flop and output, i.e. the MSB flip flop and output start from 0)

My Entry Number: 2019EE10143 corresponds to the starting point 0011

State table (given):



Q3	Q2	Q1	Q0	D3	D2	D1	D0
0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0
0	0	1	0	1	0	0	1
1	0	0	1	1	1	0	0
1	1	0	0	0	1	1	0
0	1	1	0	1	0	1	1
1	0	1	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	1	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1
0	1	1	1	0	0	1	1
0	0	1	1	0	0	0	1

D3: C'D + CD'

Map

	C'.D'	C'.D	C.D	C.D'
A'.B'	X	1	0	1
A'.B	0	1	0	1
A.B	0	1	0	1

A.B’	0	🟧 1	0	🟦 1

Groups

(1,5,9,13)	C’.D
(2,6,10,14)	C.D’

Therefore, for notations as given in the table,
D3 = Q1’.Q0 + Q0’.Q1
D2 = Q3
D1 = Q2
D0 = Q1

Verilog Code:

Note: Once again, I would like the grader to note that instead of starting from D3 to D0, I have named my flip flops from D0 to D3. Similarly, inputs go from Q0 to Q3 instead of Q3 to Q0

In the D-Flip Flop, I have used asynchronous reset and set to feed in the initial values to the variables Q1->Q4 (0011). All the flip flops which receive a high SET signal get an initial value of 1

1. D_ff.v

```verilog
module D_ff(d, clk, q, rst, set);
    input d, clk, rst, set;
    output q;
    reg q;

    always @(posedge clk) begin
        if (rst) begin
            if (set) begin
                q <= 1;
            end
            else begin
                q <=0;
            end
        end else begin
        q <= d;
        end
    end
endmodule
```

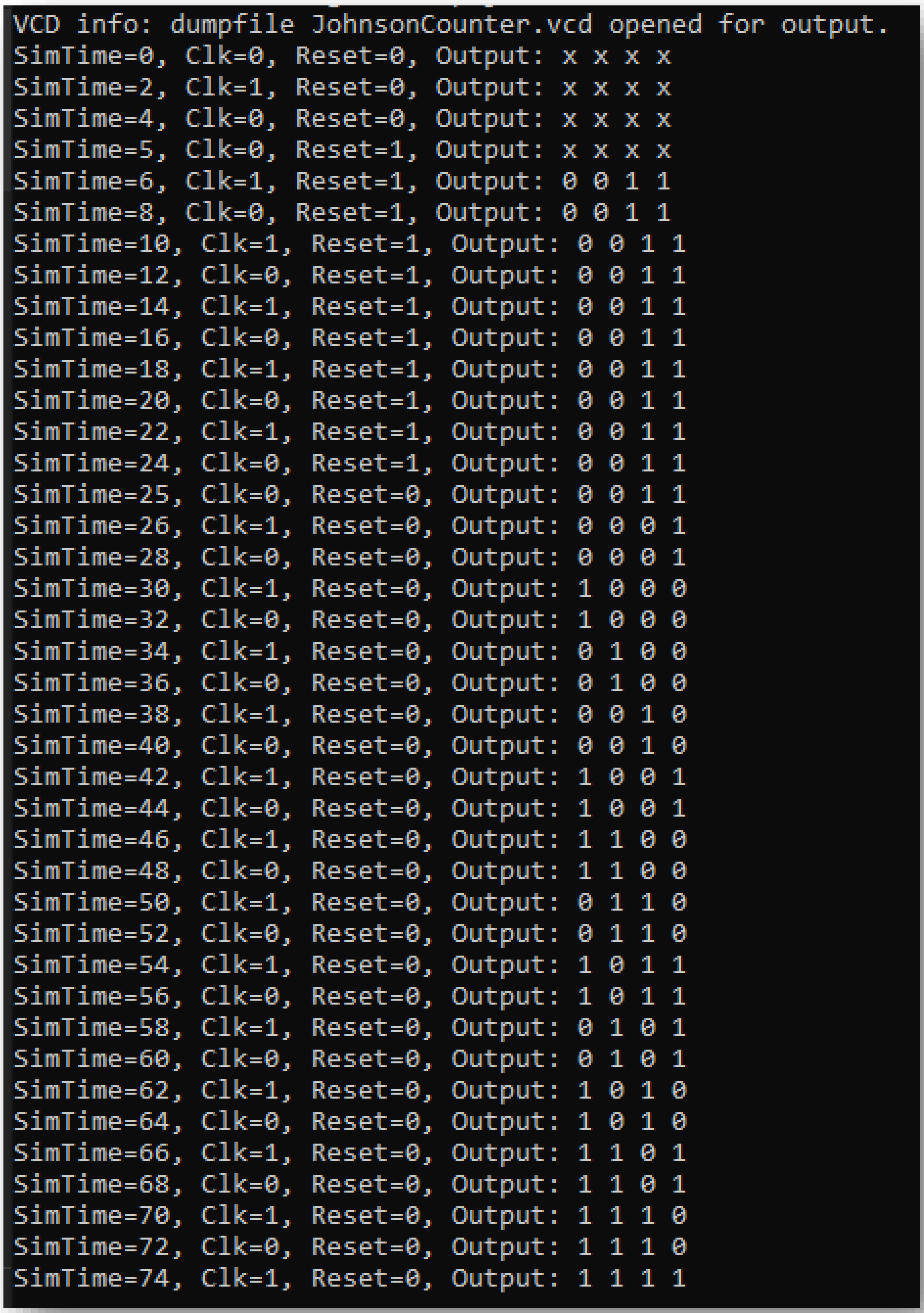
2. JohnsonCounter.v

```verilog
module JohnsonCounter(clk, q0, q1, q2, q3, rst);
    input clk, rst;
    output q0, q1, q2, q3;
    //0011
    D_ff d0((~q3&&q2)||(~q2&&q3),clk,q0,rst,1'b0);
    D_ff d1(q0,clk,q1,rst,1'b0);
    D_ff d2(q1,clk,q2,rst,1'b1);
    D_ff d3(q2,clk,q3,rst,1'b1);
endmodule
```

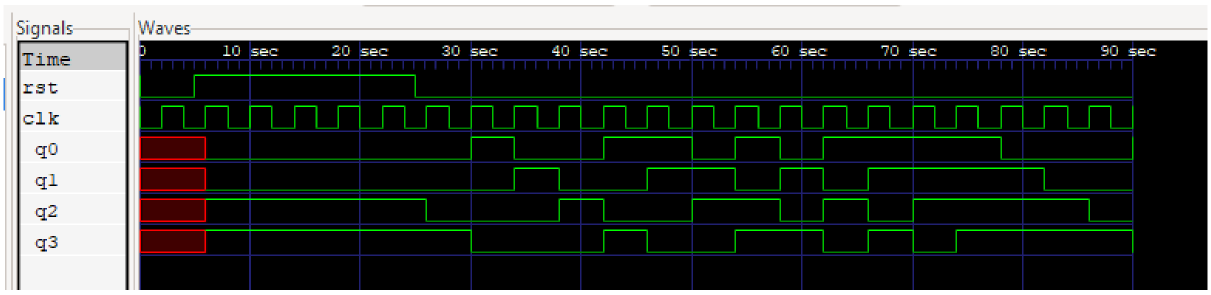
3. tb_ JohnsonCounter.v

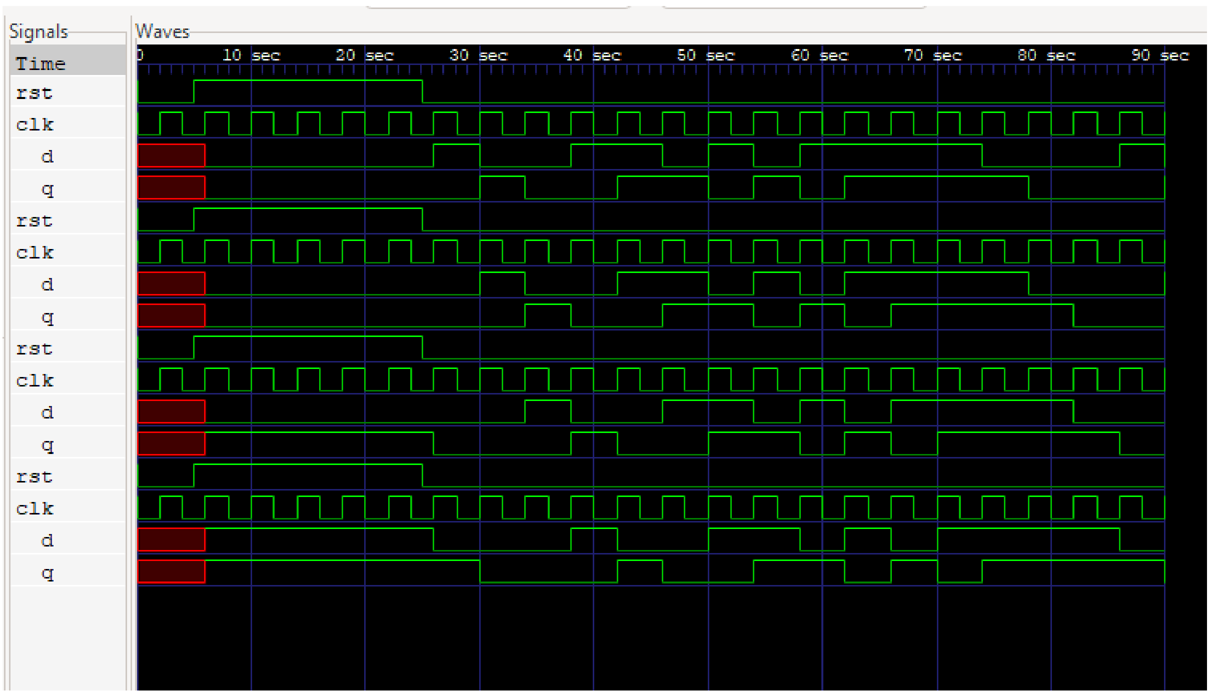
```verilog
module tb_JohnsonCounter;
    wire q3, q2, q1, q0;
    reg clk, rst;
    JohnsonCounter DUT(.clk(clk), .q0(q0), .q1(q1), .q2(q2), .q3(q3), .
rst(rst));
    always #2 clk = ~clk;
    initial begin
        clk = 1'b0;
        rst = 0;
        #5 rst = 1'b1;
        #20 rst = 1'b0;
    end
    initial begin
        $dumpfile("JohnsonCounter.vcd");
        $dumpvars(0, tb_JohnsonCounter);
        $monitor("SimTime=%g, Clk=%b, Reset=%b, Output: %b %b %b %b", $
time, clk, rst, q0, q1, q2, q3);
        #90 $finish;
    end
endmodule
```

Outputs and Waveforms:



```
VCD info: dumpfile JohnsonCounter.vcd opened for output.
SimTime=0, Clk=0, Reset=0, Output: x x x x
SimTime=2, Clk=1, Reset=0, Output: x x x x
SimTime=4, Clk=0, Reset=0, Output: x x x x
SimTime=5, Clk=0, Reset=1, Output: x x x x
SimTime=6, Clk=1, Reset=1, Output: 0 0 1 1
SimTime=8, Clk=0, Reset=1, Output: 0 0 1 1
SimTime=10, Clk=1, Reset=1, Output: 0 0 1 1
SimTime=12, Clk=0, Reset=1, Output: 0 0 1 1
SimTime=14, Clk=1, Reset=1, Output: 0 0 1 1
SimTime=16, Clk=0, Reset=1, Output: 0 0 1 1
SimTime=18, Clk=1, Reset=1, Output: 0 0 1 1
SimTime=20, Clk=0, Reset=1, Output: 0 0 1 1
SimTime=22, Clk=1, Reset=1, Output: 0 0 1 1
SimTime=24, Clk=0, Reset=1, Output: 0 0 1 1
SimTime=25, Clk=0, Reset=0, Output: 0 0 1 1
SimTime=26, Clk=1, Reset=0, Output: 0 0 0 1
SimTime=28, Clk=0, Reset=0, Output: 0 0 0 1
SimTime=30, Clk=1, Reset=0, Output: 1 0 0 0
SimTime=32, Clk=0, Reset=0, Output: 1 0 0 0
SimTime=34, Clk=1, Reset=0, Output: 0 1 0 0
SimTime=36, Clk=0, Reset=0, Output: 0 1 0 0
SimTime=38, Clk=1, Reset=0, Output: 0 0 1 0
SimTime=40, Clk=0, Reset=0, Output: 0 0 1 0
SimTime=42, Clk=1, Reset=0, Output: 1 0 0 1
SimTime=44, Clk=0, Reset=0, Output: 1 0 0 1
SimTime=46, Clk=1, Reset=0, Output: 1 1 0 0
SimTime=48, Clk=0, Reset=0, Output: 1 1 0 0
SimTime=50, Clk=1, Reset=0, Output: 0 1 1 0
SimTime=52, Clk=0, Reset=0, Output: 0 1 1 0
SimTime=54, Clk=1, Reset=0, Output: 1 0 1 1
SimTime=56, Clk=0, Reset=0, Output: 1 0 1 1
SimTime=58, Clk=1, Reset=0, Output: 0 1 0 1
SimTime=60, Clk=0, Reset=0, Output: 0 1 0 1
SimTime=62, Clk=1, Reset=0, Output: 1 0 1 0
SimTime=64, Clk=0, Reset=0, Output: 1 0 1 0
SimTime=66, Clk=1, Reset=0, Output: 1 1 0 1
SimTime=68, Clk=0, Reset=0, Output: 1 1 0 1
SimTime=70, Clk=1, Reset=0, Output: 1 1 1 0
SimTime=72, Clk=0, Reset=0, Output: 1 1 1 0
SimTime=74, Clk=1, Reset=0, Output: 1 1 1 1
```





We can see that the transitions follow the order as shown in the state table drawn earlier. It starts from 0011 and after a series of 15 transitions, gets back to this position. The reset signal is only present for a short while so that the values are not set on every transition

- It is to be noted that the counter developed, does not go through all 16 states. This is because if it were to ever reach the state 0000, it would get stuck there till the reset value is 1 again. We reach this stagnant stage as the value 0 keeps getting fed into the flip flops. Even 0 xor 0 is 0.
- It is also worth noting that the starting point does not affect the sequence at all, we still reach the same 15 states in due time.