EC204

Digital System Design Lab

Lab – 7

Utkarsh R Mahajan

201EC164

1] **Decade Counter** - Model a synchronous up/down decade counter with asynchronous reset. The counter is rising edge triggered.

If the **load** = 1, the data **data_in** is loaded into the counter.

If the **counter_on**=**counter_up**=1, the counter is incremented, the Terminal carry output TC=1 when the counter is in state 9.

If the **counter_on**=1 and **counter_up**=0, the counter is decremented, the Terminal carry output **TC**=1 when the counter is in state 0.

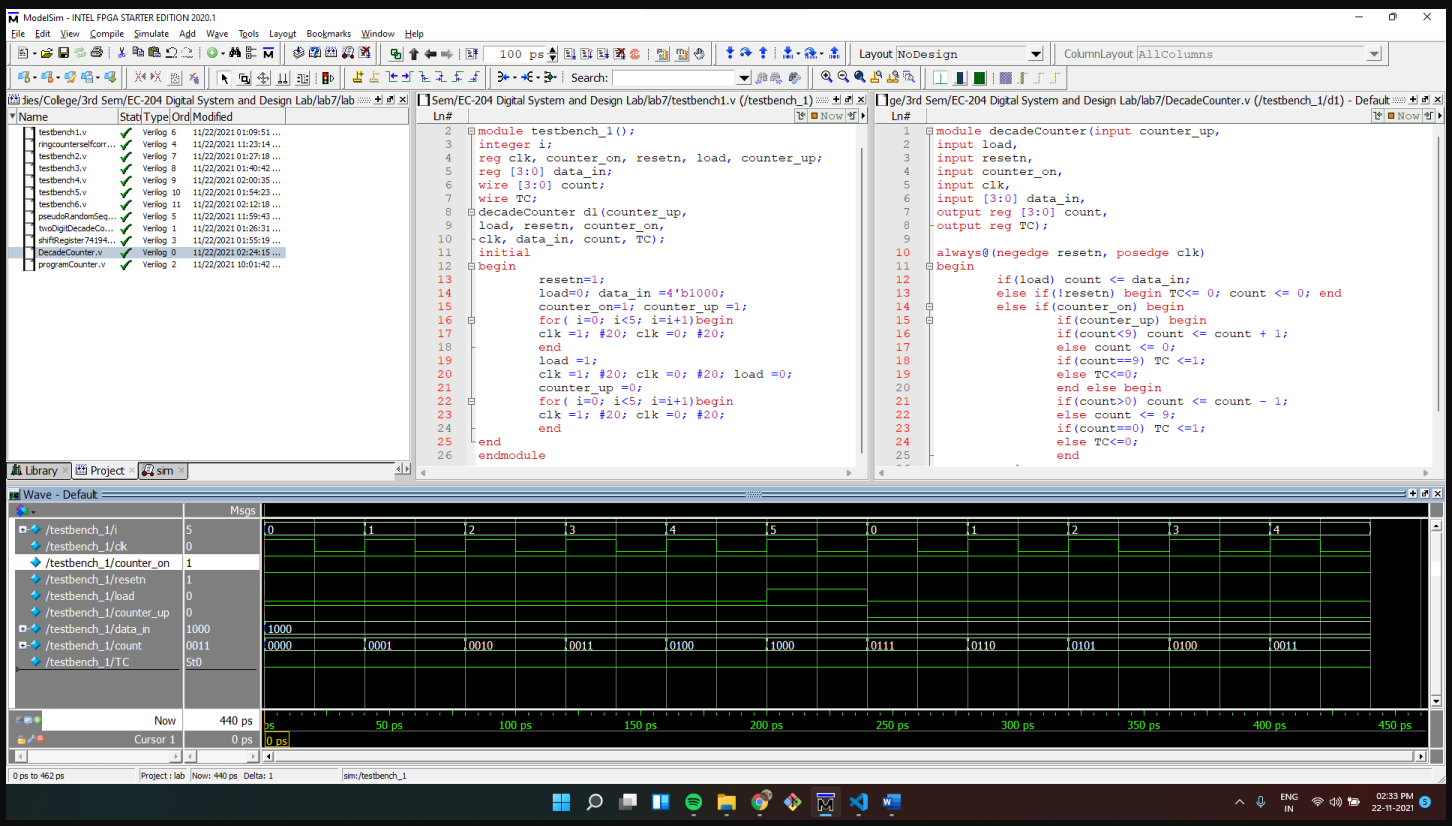
-> Verilog code for the counter:

```
module decadeCounter(input counter_up,
input load,
input resetn,
input counter_on,
input clk,
input [3:0] data_in,
output reg [3:0] count,
output reg TC);

always@(negedge resetn, posedge clk)
begin
    if(load) count <= data_in;
    else if(!resetn) begin TC<= 0; count <= 0; end
    else if(counter_on) begin
        if(counter_up) begin
        if(count<9) count <= count + 1;
        else count <= 0;
        if(count==9) TC <=1;
        else TC<=0;
        end else begin
        if(count>0) count <= count - 1;
        else count <= 9;
        if(count==0) TC <=1;
        else TC<=0;
        end
    end
end
endmodule
```

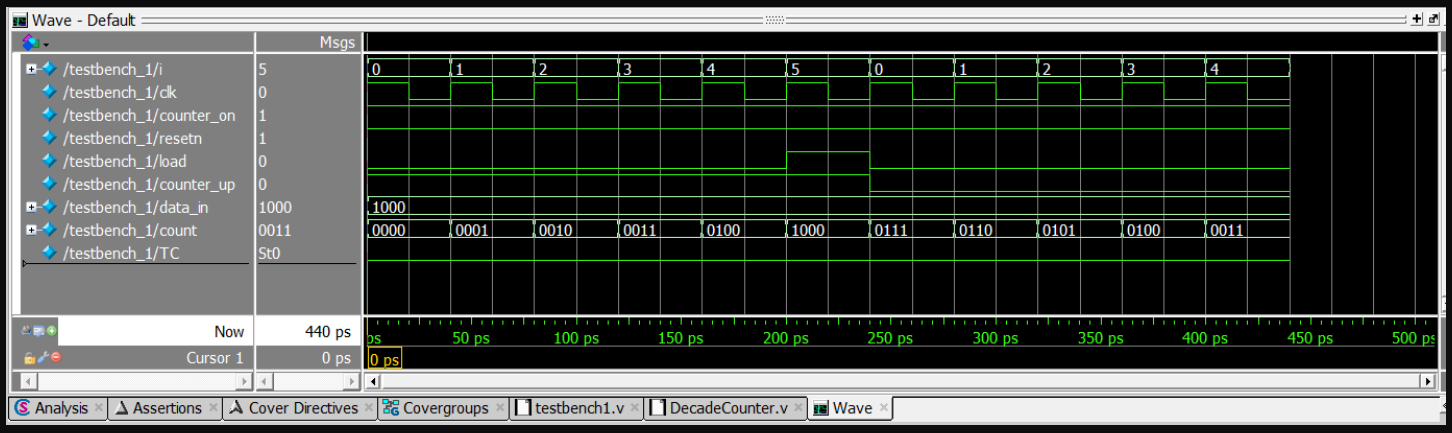
Verilog code for the testbench:

```
module testbench_1();
integer i;
reg clk, counter_on, resetn, load, counter_up;
reg [3:0] data_in;
wire [3:0] count;
wire TC;
decadeCounter d1(counter_up,
load, resetn, counter_on,
clk, data_in, count, TC);
initial
begin
    resetn=1;
    load=0; data_in =4'b1000;
    counter_on=1; counter_up =1;
    for( i=0; i<5; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
    load =1;
    clk =1; #20; clk =0; #20; load =0;
    counter_up =0;
    for( i=0; i<5; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
end
endmodule
```

Full Screenshot:



Wave:



2] Use two decade counter modules to form a two-digit decimal up/down counter.

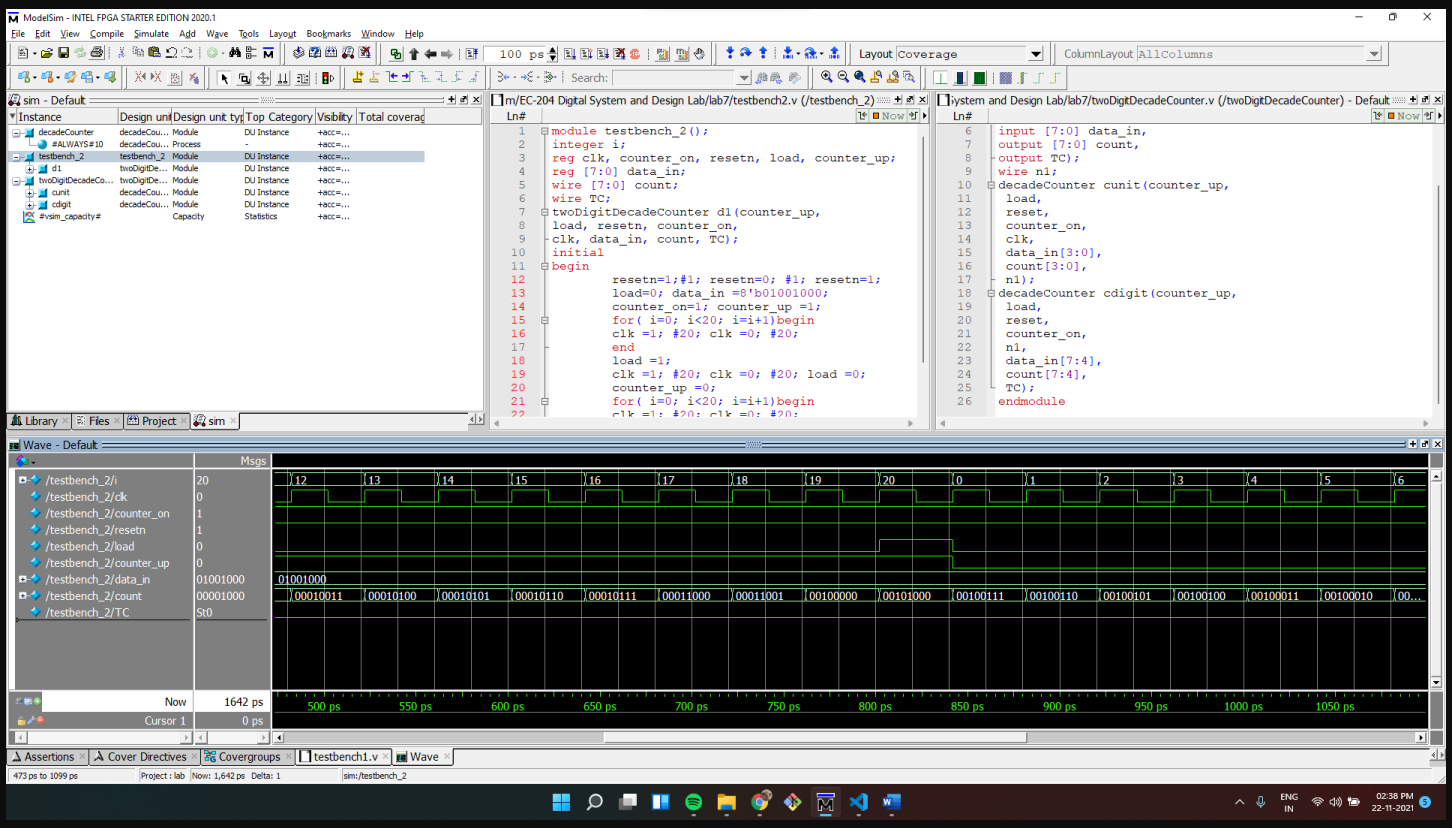
-> Verilog code for the counter:

```
module twoDigitDecadeCounter(input counter_up,
input load,
input reset,
input counter_on,
input clk,
input [7:0] data_in,
output [7:0] count,
output TC);
wire n1;
decadeCounter cunit(counter_up,
 load,
 reset,
 counter_on,
 clk,
 data_in[3:0],
 count[3:0],
 n1);
decadeCounter cdigit(counter_up,
 load,
 reset,
 counter_on,
 n1,
 data_in[7:4],
 count[7:4],
 TC);
endmodule
```

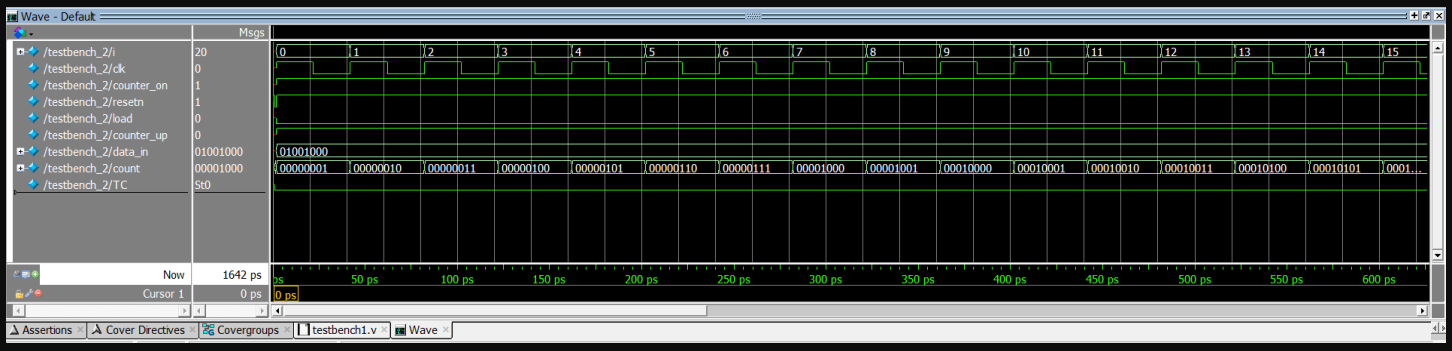
Verilog code for the testbench:

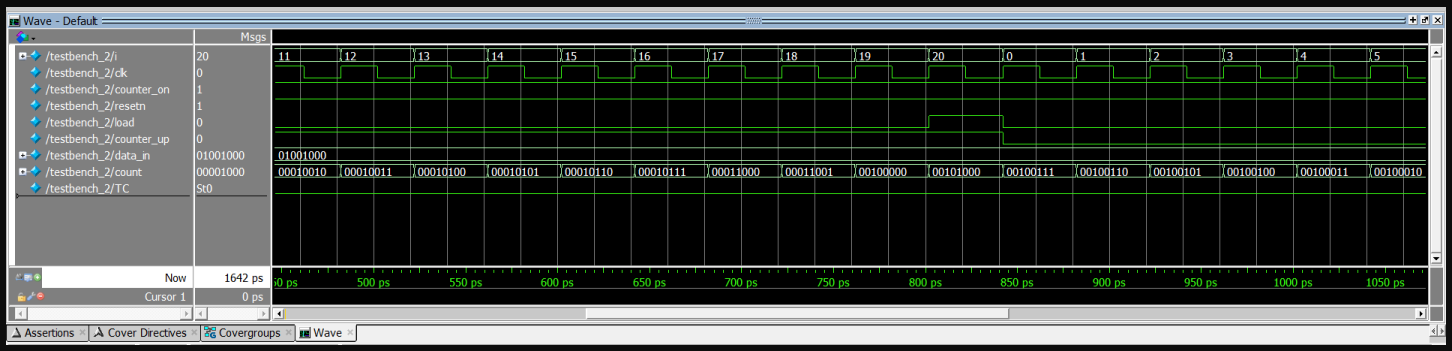
```
module testbench_2();
integer i;
reg clk, counter_on, resetn, load, counter_up;
reg [7:0] data_in;
wire [7:0] count;
wire TC;
twoDigitDecadeCounter d1(counter_up,
load, resetn, counter_on,
clk, data_in, count, TC);
initial
begin
    resetn=1;#1; resetn=0; #1; resetn=1;
    load=0; data_in =8'b01001000;
    counter_on=1; counter_up =1;
    for( i=0; i<20; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
    load =1;
    clk =1; #20; clk =0; #20; load =0;
    counter_up =0;
    for( i=0; i<20; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
end
endmodule
```

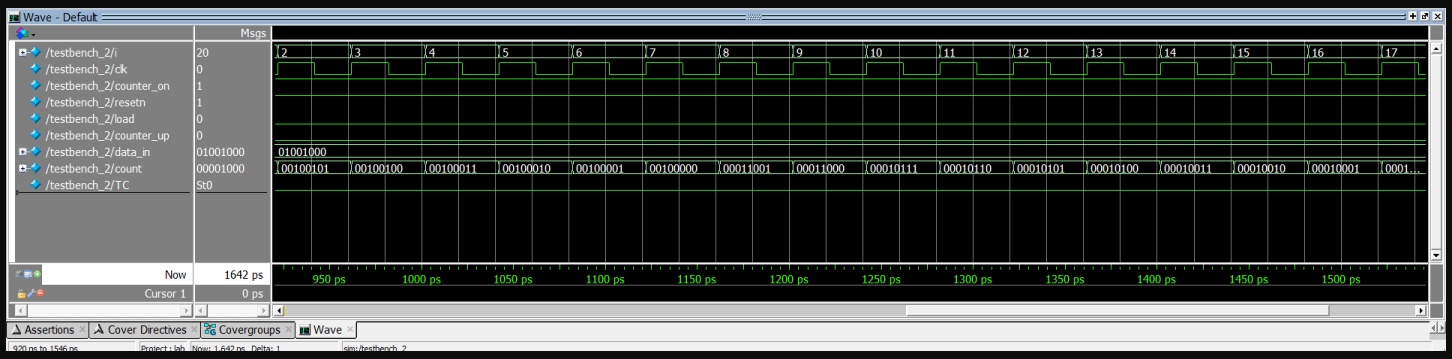
Full Screenshot:



Wave:







3] Design an n-bit program counter (PC) that has the following functions:

a. It is cleared to 0 when the asynchronous reset is asserted
b. It is loaded a new value in parallel when the PCload is asserted
c. It is incremented by 1 when the PCinc is asserted

We will consider n as 4 for our working example.

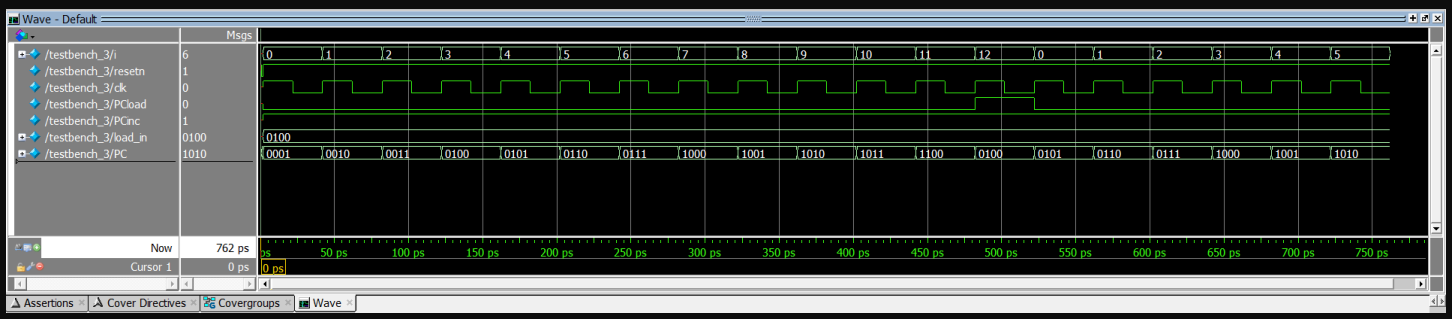
-> Verilog code for the Program counter:

```
module programCounter(reset, clk, PCload, PCinc, load_in, PC);
parameter n=4;
input reset, clk, PCload, PCinc;
input [n-1:0] load_in;
output reg [n-1:0] PC;

always @(negedge reset, posedge clk)
begin
    if(!reset) PC <= 0;
    else if (PCload) PC <= load_in;
    else if(PCinc) PC <= PC+1;
end
endmodule
```

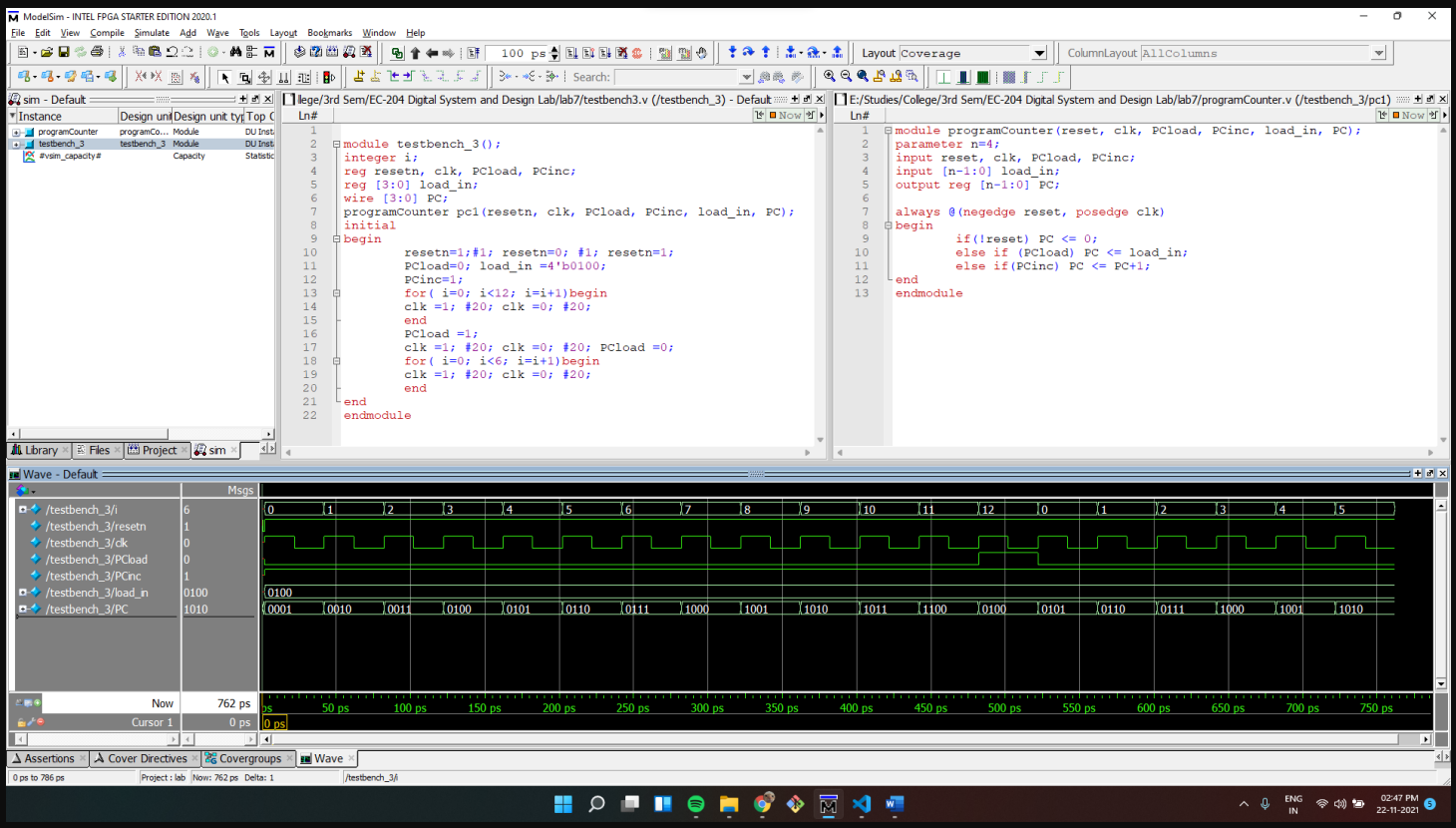
Verilog code for the testbench:

```
module testbench_3();
integer i;
reg resetn, clk, PCload, PCinc;
reg [3:0] load_in;
wire [3:0] PC;
programCounter pc1(resetn, clk, PCload, PCinc, load_in, PC);
initial
begin
    resetn=1;#1; resetn=0; #1; resetn=1;
    PCload=0; load_in =4'b0100;
    PCinc=1;
    for( i=0; i<12; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
    PCload =1;
    clk =1; #20; clk =0; #20; PCload =0;
    for( i=0; i<6; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
end
endmodule
```

Wave:



Full Screenshot:



4] Model the functionality of Shift register 74194.

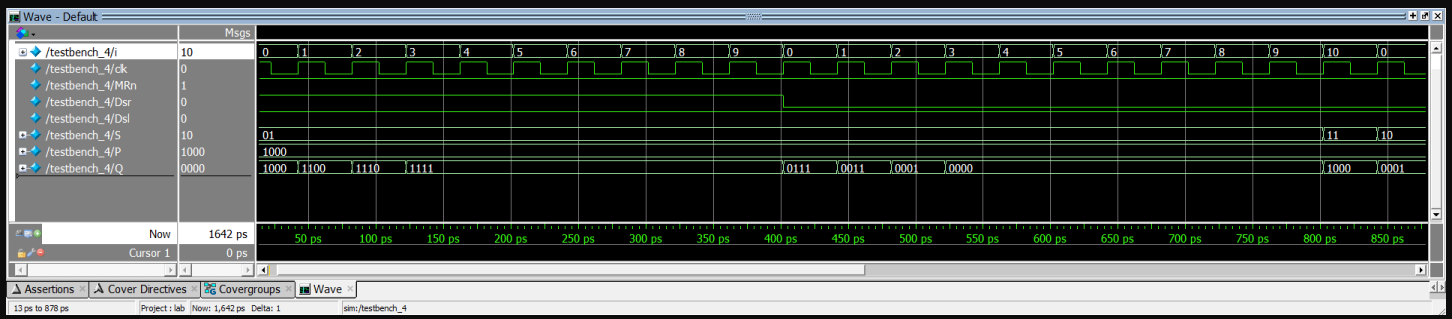
-> Verilog code for the Shift Register:

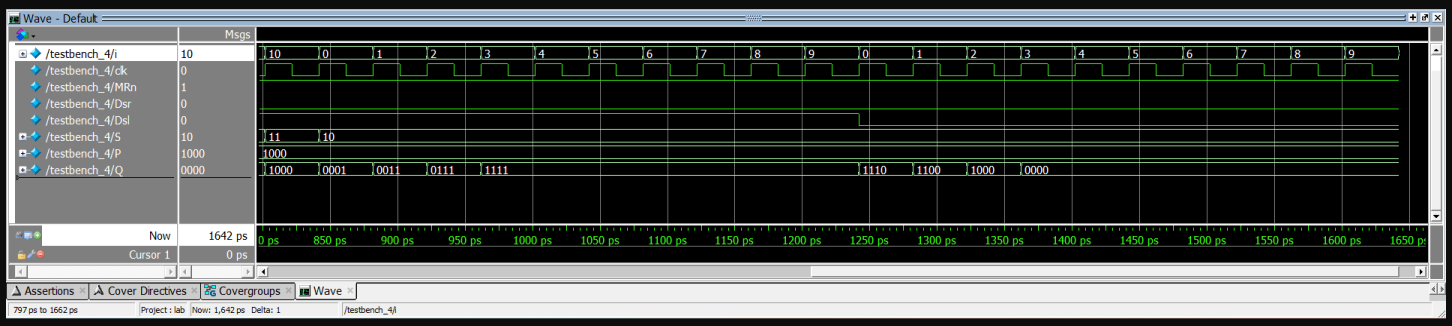
```
module shiftRegister74194(input MRn, CP, Dsr, Dsl,
  input [1:0]S, input [3:0] P, output reg [3:0] Q);

always@(negedge MRn, posedge CP)
begin
    if(!MRn) Q <= 0;
    else if(S==2'b01) begin
        if(Dsr) Q<= {1'b1, Q[3:1]};
        else  Q<= {1'b0, Q[3:1]};
    end else if(S==2'b10) begin
        if(Dsl) Q<= {Q[2:0], 1'b1};
        else  Q<= {Q[2:0], 1'b0};
    end else if(S==2'b11) Q <= P;
end
endmodule
```

Verilog code for the testbench:

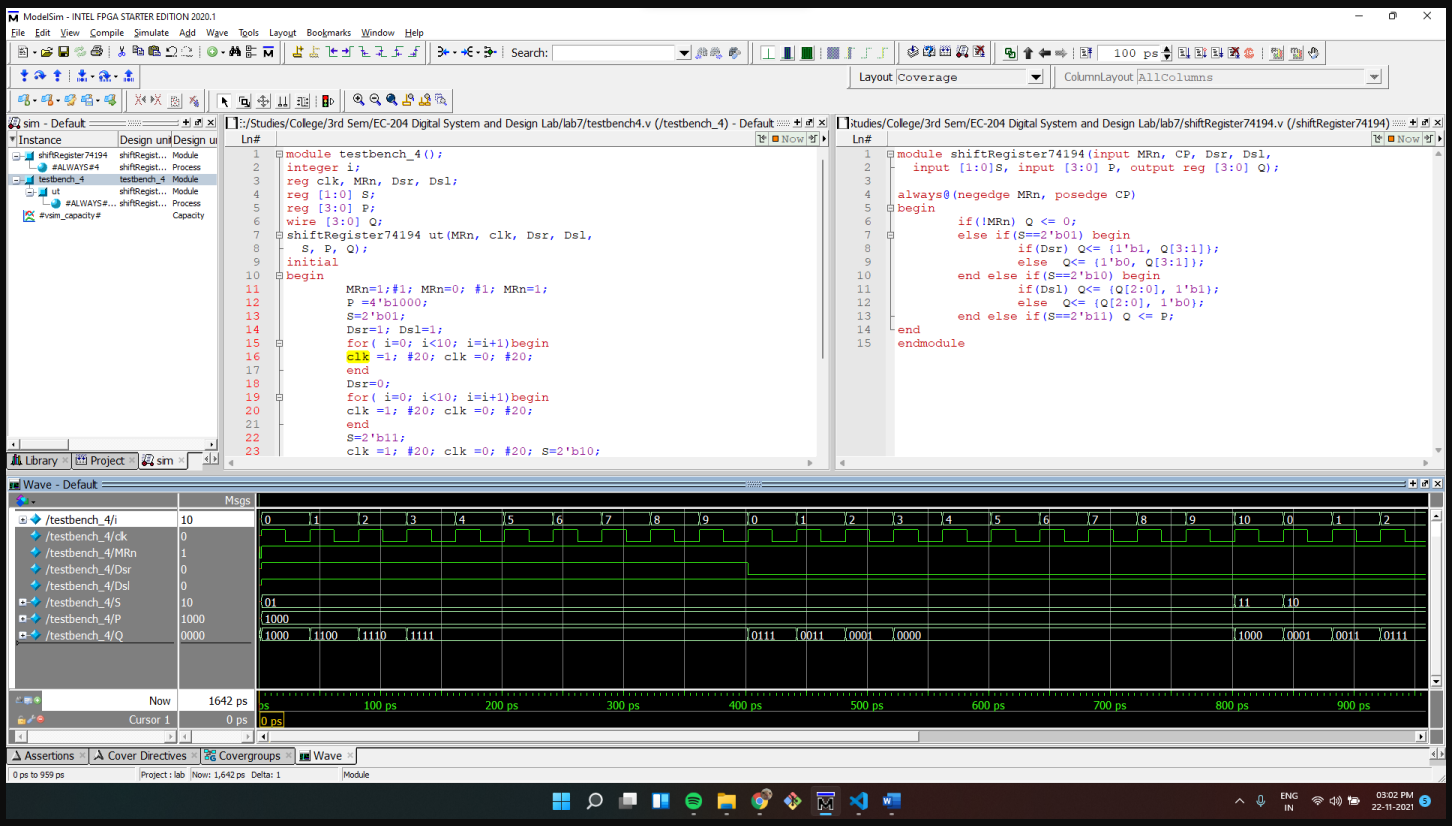
```
module testbench_4();
integer i;
reg clk, MRn, Dsr, Dsl;
reg [1:0] S;
reg [3:0] P;
wire [3:0] Q;
shiftRegister74194 ut(MRn, clk, Dsr, Dsl,
  S, P, Q);
initial
begin
    MRn=1;#1; MRn=0; #1; MRn=1;
    P =4'b1000;
    S=2'b01;
    Dsr=1; Dsl=1;
    for( i=0; i<10; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
    Dsr=0;
    for( i=0; i<10; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
    S=2'b11;
    clk =1; #20; clk =0; #20; S=2'b10;
    for( i=0; i<10; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
    Dsl=0;
    for( i=0; i<10; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
end
endmodule
```

Wave:





Full Screenshot:



5] Write a Verilog module for a 4 bit self-correcting ring counter.

-> In a self-correcting ring counter, the current state bits resets if they enter a invalid state.

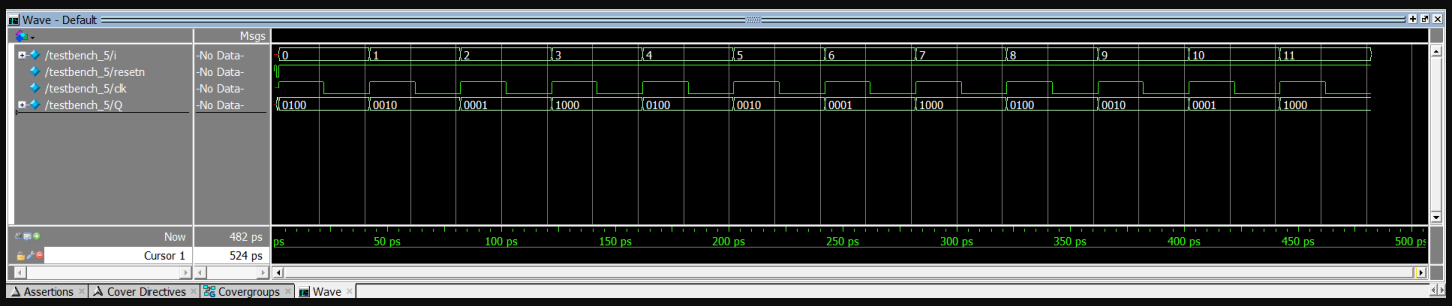
Verilog code for the ring counter:

```
module rignCounterSelfCorrecting(input clk,
 input resetn,
 output reg [3:0] Q);

always @(negedge resetn, posedge clk)
begin
    if(!resetn) Q <= 4'b1000;
    else if(Q==4'b1000 || Q==4'b0100 || Q==4'b0010) Q <= Q >> 1;
    else Q<=4'b1000;
end
endmodule
```

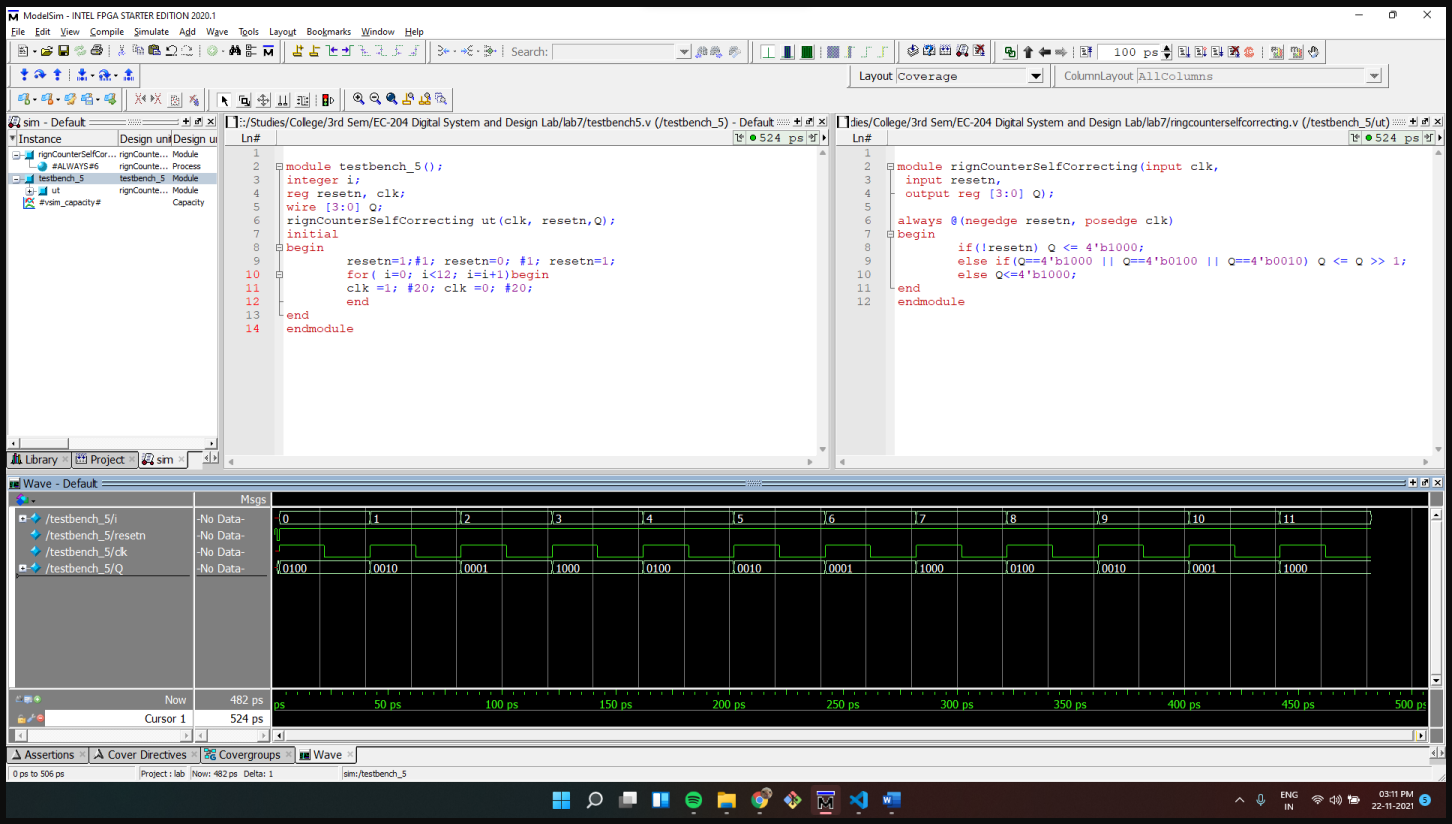
Verilog code for the testbench:

```
module testbench_5();
integer i;
reg resetn, clk;
reg [3:0] load_in;
wire [3:0] Q;
rignCounterSelfCorrecting ut(clk, resetn,Q);
initial
begin
    resetn=1;#1; resetn=0; #1; resetn=1;
    for( i=0; i<12; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
end
endmodule
```

Wave:



Full Screenshot:



6] Write a Verilog module for a Pseudo random sequence generator for n=8. Primitive polynomial is $1 + x^2 + x^3 + x^4 + x^8$.

-> This can be done using xor gates.

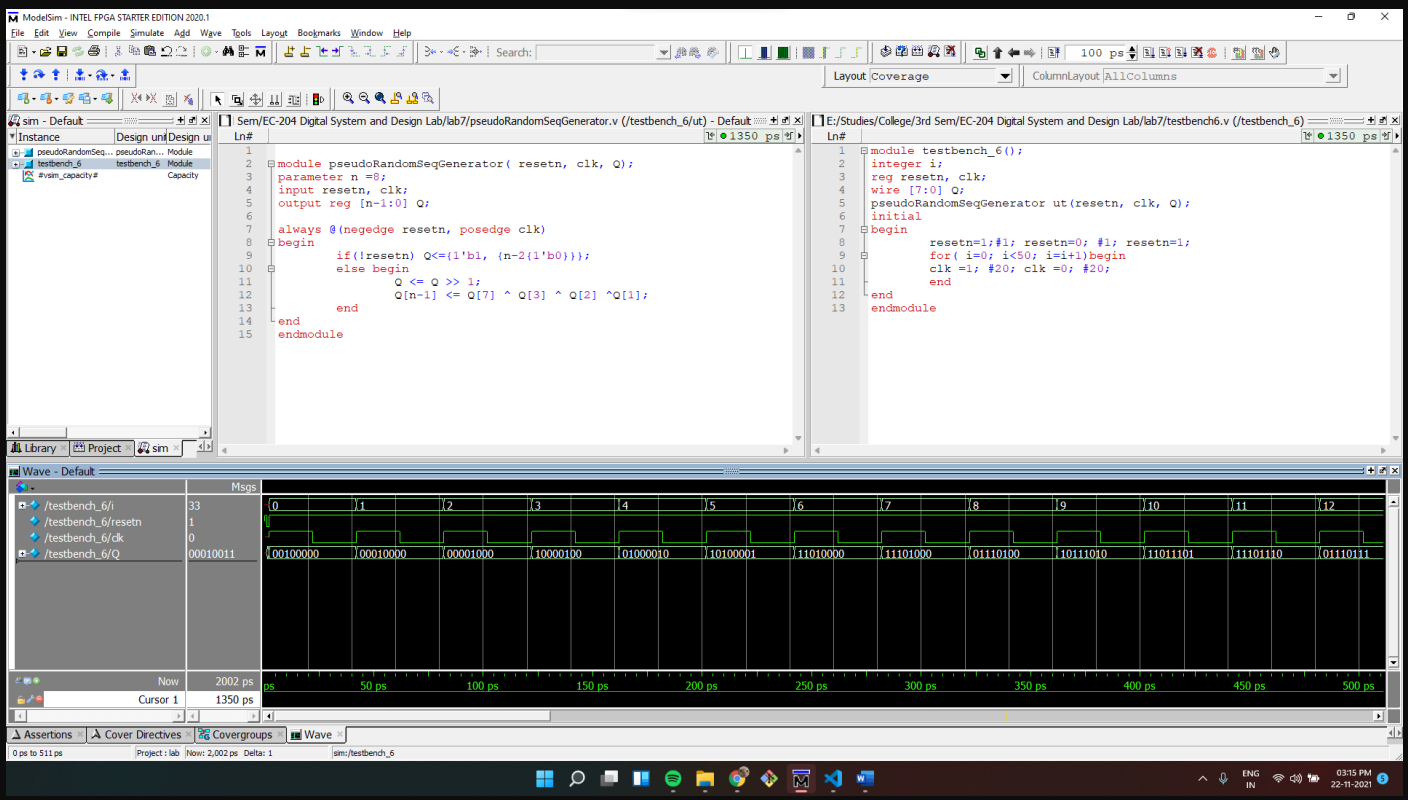
Verilog code for the sequence generator:

```
module pseudoRandomSeqGenerator( resetn, clk, Q);
parameter n =8;
input resetn, clk;
output reg [n-1:0] Q;

always @(negedge resetn, posedge clk)
begin
    if(!resetn) Q<={1'b1, {n-2{1'b0}}};
    else begin
        Q <= Q >> 1;
        Q[n-1] <= Q[7] ^ Q[3] ^ Q[2] ^Q[1];
    end
end
endmodule
```

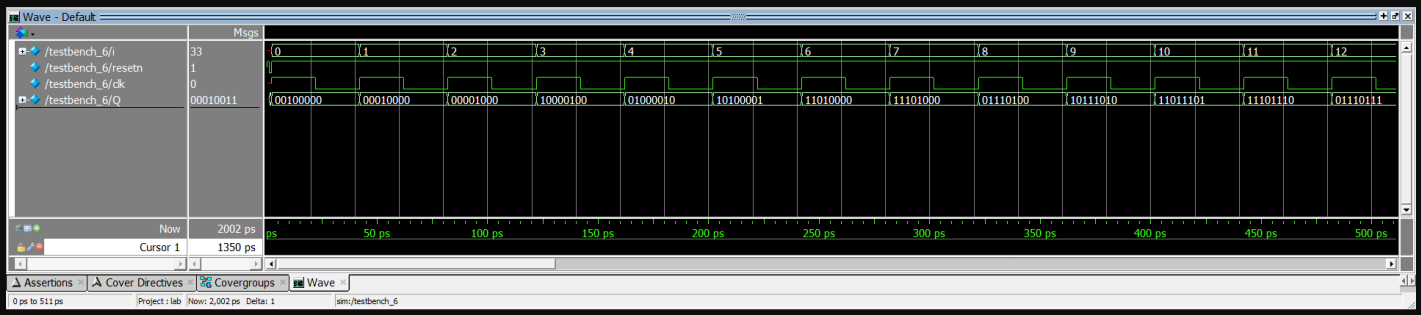
Verilog code for the testbench:

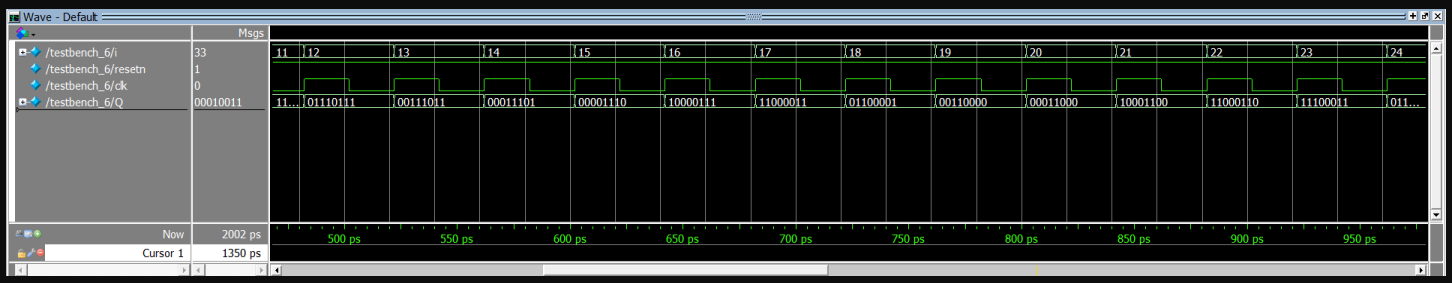
```
module testbench_6();
integer i;
reg resetn, clk;
wire [7:0] Q;
pseudoRandomSeqGenerator ut(resetn, clk, Q);
initial
begin
    resetn=1;#1; resetn=0; #1; resetn=1;
    for( i=0; i<50; i=i+1)begin
    clk =1; #20; clk =0; #20;
    end
end
endmodule
```

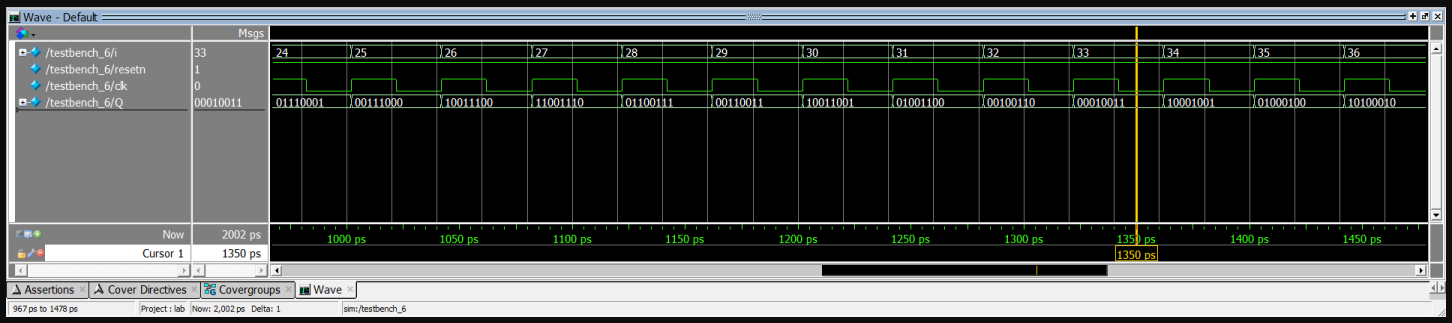
Full Screenshot:

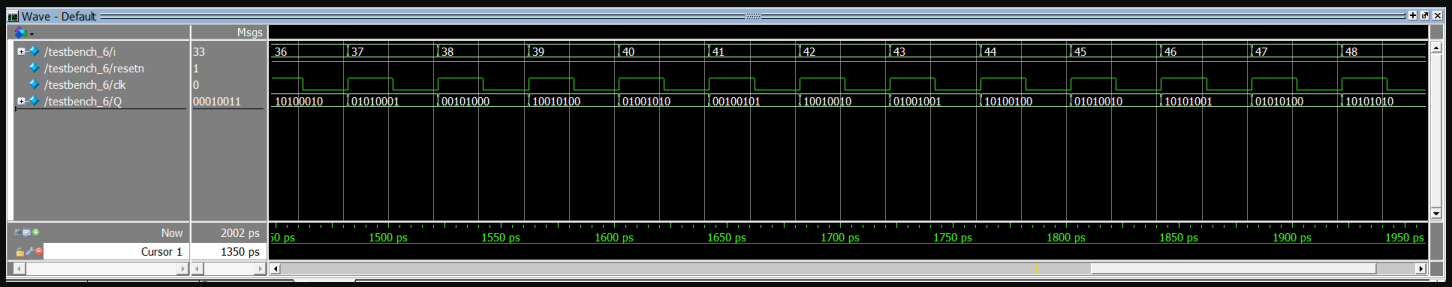


Wave:




Wave - Default
Msgs
/testbench_6/i
/testbench_6/resetn
/testbench_6/ck
/testbench_6/Q
33
1
0
00010011
11 12 13 14 15 16 17 18 19 20 21 22 23 24
11... 01110111 00111011 00011101 00001110 10000111 11000011 01100001 00110000 00011000 10001100 11000110 11100011 011...
Now
2002 ps
Cursor 1
1350 ps
500 ps 550 ps 600 ps 650 ps 700 ps 750 ps 800 ps 850 ps 900 ps 950 ps


Wave - Default
Msgs
/testbench_6/i
/testbench_6/resetn
/testbench_6/ck
/testbench_6/Q
33
1
0
00010011
24 25 26 27 28 29 30 31 32 33 34 35 36
01110001 00111000 10011100 11001110 01100111 00110011 10011001 01001100 00100110 00010011 10001001 01000100 10100010
Now
2002 ps
Cursor 1
1350 ps
1000 ps 1050 ps 1100 ps 1150 ps 1200 ps 1250 ps 1300 ps 1350 ps 1400 ps 1450 ps
1350 ps
Assertions
Cover Directives
Covergroups
Wave
967 ps to 1478 ps
Project : lab
Now: 2,002 ps
Delta: 1
sim:/testbench_6


Wave - Default
Msgs
/testbench_6/i
/testbench_6/resetn
/testbench_6/ck
/testbench_6/Q
33
1
0
00010011
36 37 38 39 40 41 42 43 44 45 46 47 48
10100010 01010001 00101000 10010100 01001010 00100101 10010010 01001001 10100100 01010010 10101001 01010100 10101010
Now
2002 ps
Cursor 1
1350 ps
0 ps 1500 ps 1550 ps 1600 ps 1650 ps 1700 ps 1750 ps 1800 ps 1850 ps 1900 ps 1950 ps