ECE-811: Lab 4 (Consecutive 1s detector)

Group 4:
William Nitsch
Mohammad Raashid Ansari
Jonathan Frey

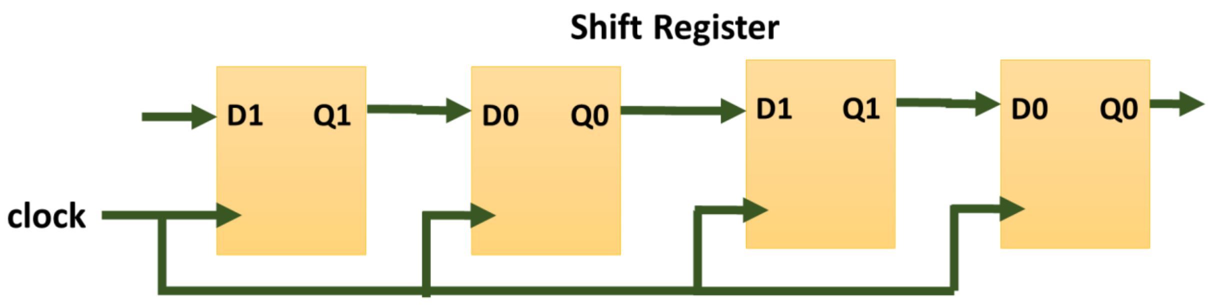
1. **Design details of the shifter and the 4 consecutive 1s detector with source codes (FIFO4.v (Shifter) and FIFO4_tapped.v (1s detector)):**

The shift registers at the input and output of the design are effectively the same exact design, using D-flip-flops from the component library to shift an input value sequentially through the chain of flip-flops with each clock cycle. The difference between the input and output shift registers is that the output shift register has an extra output that is used to indicate the detection of four sequential values of "1". This output is equivalent to the each of the shift register bits and-gated with one another.

a. FIFO4.v (Shifter):

```
`timescale 1ns/1ns
module FIFO4(In, Out, CLK);
    input In, CLK;
    output Out;

    wire In, CLK, D0, D1, D2, Out;

    DFQD1 FF1(.D(In), .CP(CLK), .Q(D0));
    DFQD1 FF2(.D(D0), .CP(CLK), .Q(D1));
    DFQD1 FF3(.D(D1), .CP(CLK), .Q(D2));
    DFQD1 FF4(.D(D2), .CP(CLK), .Q(Out));

Endmodule
```

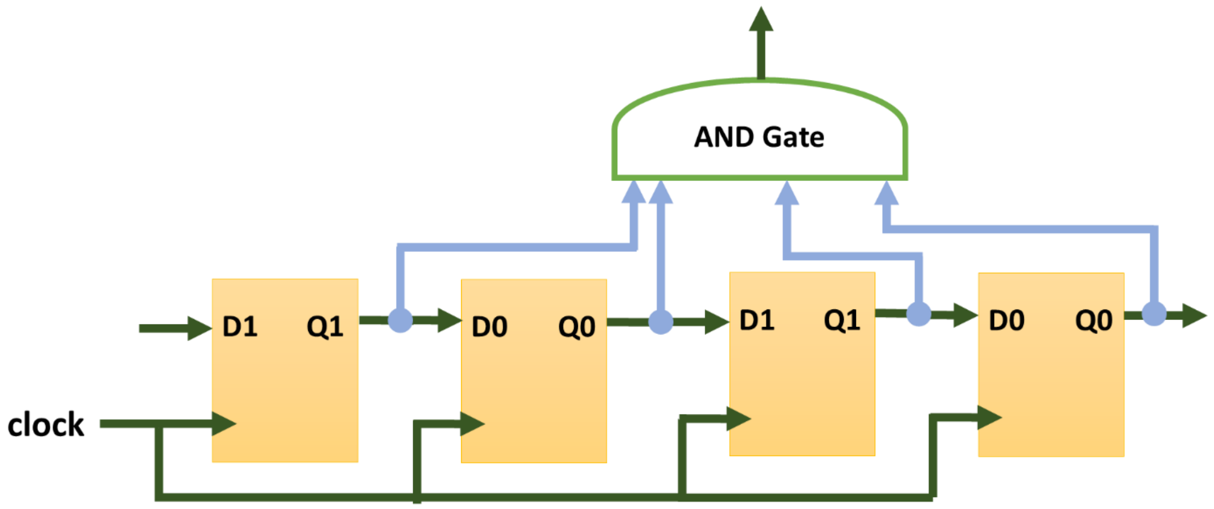
The figure below shows the schematic for the shift registers that were developed with the above code:



b. FIFO4_tapped.v (1s detector):

```
`timescale 1ns/1ns
module FIFO4_tapped(In, O0, O1, O2, O3, consec_out, CLK);
    input In, CLK;
    output O0, O1, O2, O3, consec_out;

    wire In, CLK, O0, O1, O2, O3;

    DFQD1 FF1(.D(In), .CP(CLK), .Q(O0));
    DFQD1 FF2(.D(O0), .CP(CLK), .Q(O1));
    DFQD1 FF3(.D(O1), .CP(CLK), .Q(O2));
    DFQD1 FF4(.D(O2), .CP(CLK), .Q(O3));

    assign consec_out = O0 && O1  && O2  && O3;

endmodule
```

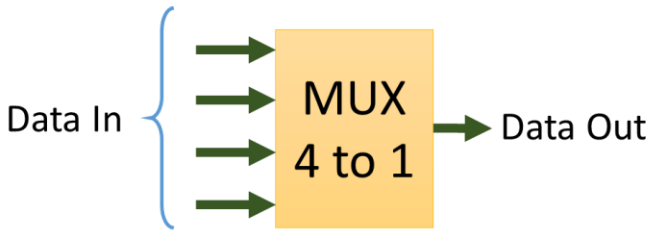
The figure below shows the representation of our 1's detector:



Consecutive Ones Detector

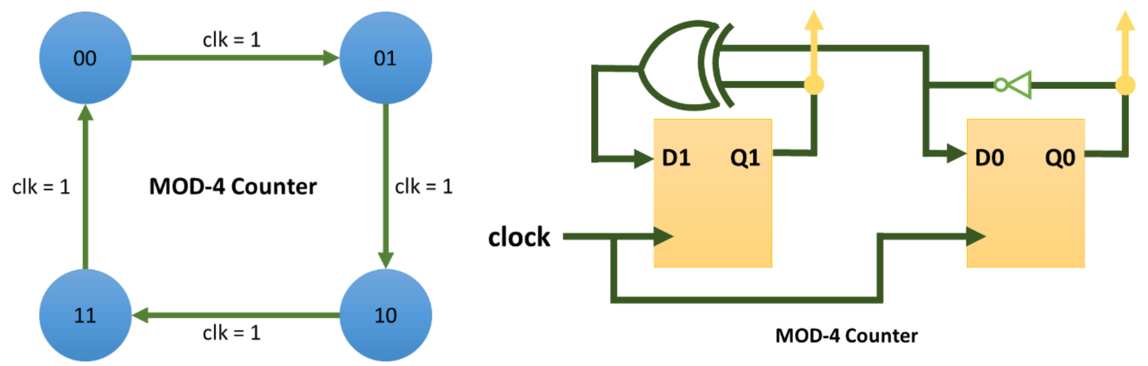
c. <u>4-to-1 Multiplexer</u> (mux4_1.v):

The 4 to 1 multiplexer was designed with 4-bit input (datain) and 1 bit output register (dataout). A 2 bit select signal (selectbit) was used to take inputs from the MOD-4 counter for selecting inputs of the MUX. Case statements were used in order to keep the code understandable and maintainable.



d. MOD-4 Counter (MOD4counter.v):

A MOD-4 counter was developed using Moore state machine model with an asynchronous reset. Two D Flip-Flops (D0 and D1) with outputs (Q0 and Q1) were given synchronous clock input. Output Q0 was inverted and fed back to D0. Outputs Q0 and Q1 were XORed and fed to D1. Outputs, Q0 and Q1 are tapped-out to be used as the select signal for the MUX.



e. Top Module (containing all other modules) (lab4_top.v)

This module condenses the other components into one module. Each of the input shift registers (s1, s2, s3 and s4) receive external inputs from the testbench (*inputvals[3:0]*), and with each clock cycle shift the inputs towards the outputs of each shift register. The output of each of these registers then goes to the input terminals of the multiplexer (s1 → input 1, s2 → input 2, s3 → input 3, s4 → input 4). The MOD-4 counter output (*SEL*) then determines the selected input of the multiplexer that is transferred to the output (*Muxout*). The output of the multiplexer is then directed to the input of the output shift register (s5). The output values within the shift register are all visible as outputs, as well as the flag set by 4 consecutive 1 values (*consec4*).

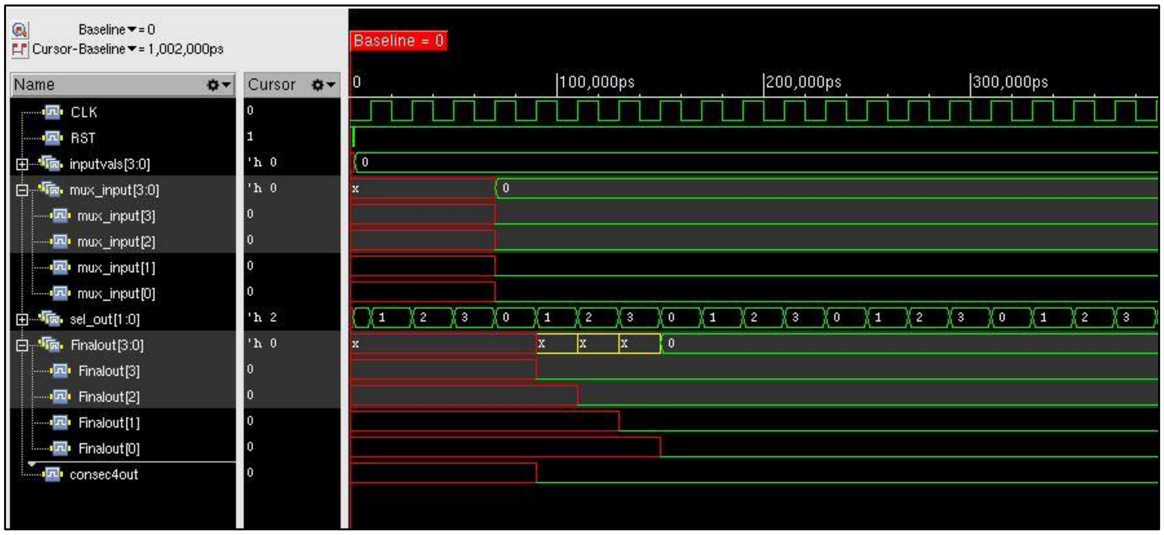
lab4_top.v:

```
'timescale 1ns/1ns
module lab4_top(inputvals, Muxin, final_sr_out, consec4, SEL, CLK, RST);
    input RST, CLK;
    input[3:0] inputvals;
    output[3:0] Muxin, final_sr_out;
    output[1:0] SEL;
    output consec4;

    wire Output0, Output1, Output2, Output3, Muxout, consec4;
    wire[1:0] SEL;
    wire[3:0] Muxin;

    FIFO4 s1(.In(inputvals[0]), .Out(Output0), .CLK(CLK));
    FIFO4 s2(.In(inputvals[1]), .Out(Output1), .CLK(CLK));
    FIFO4 s3(.In(inputvals[2]), .Out(Output2), .CLK(CLK));
    FIFO4 s4(.In(inputvals[3]), .Out(Output3), .CLK(CLK));
    FIFO4_tapped s5(.In(Muxout), .O0(final_sr_out[3]), .O1(final_sr_out[2]),
.O2(final_sr_out[1]), .O3(final_sr_out[0]), .consec_out(consec4),
.CLK(CLK));
    MOD4counter m1(.clk(CLK), .reset(RST), .q(SEL));
    mux4_1 m2(.datain(Muxin), .selectbit(SEL), .dataout(Muxout));

    assign Muxin = {Output3, Output2, Output1, Output0};
endmodule
```

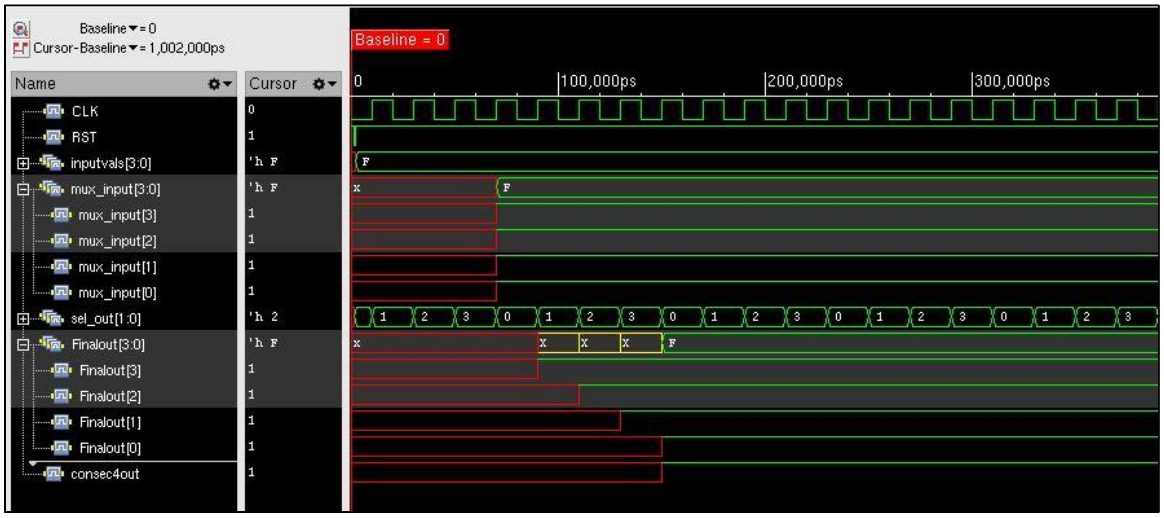
2. **Testbench Design:**

The test bench for this system tests three separate conditions. The first is a consistent input of "0" for all four input shift registers. The second condition is consistent input of "1" for all four input shift registers. The third condition is a random input for each of the four input shift registers. As expected, with all zero or one values as the input, the output of the circuit will be a consistent value of zero or one, respectively. Similarly, the value of the consecutive output flag is either always zero (for zeroes input) or always one (for ones input). This is of course after the appropriate number of clock cycles required to shift to the final output of the circuit. When the input is random, the circuit behaves as expected, shifting the proper values into the multiplexer and indicating when the shift register at the output is filled with ones.

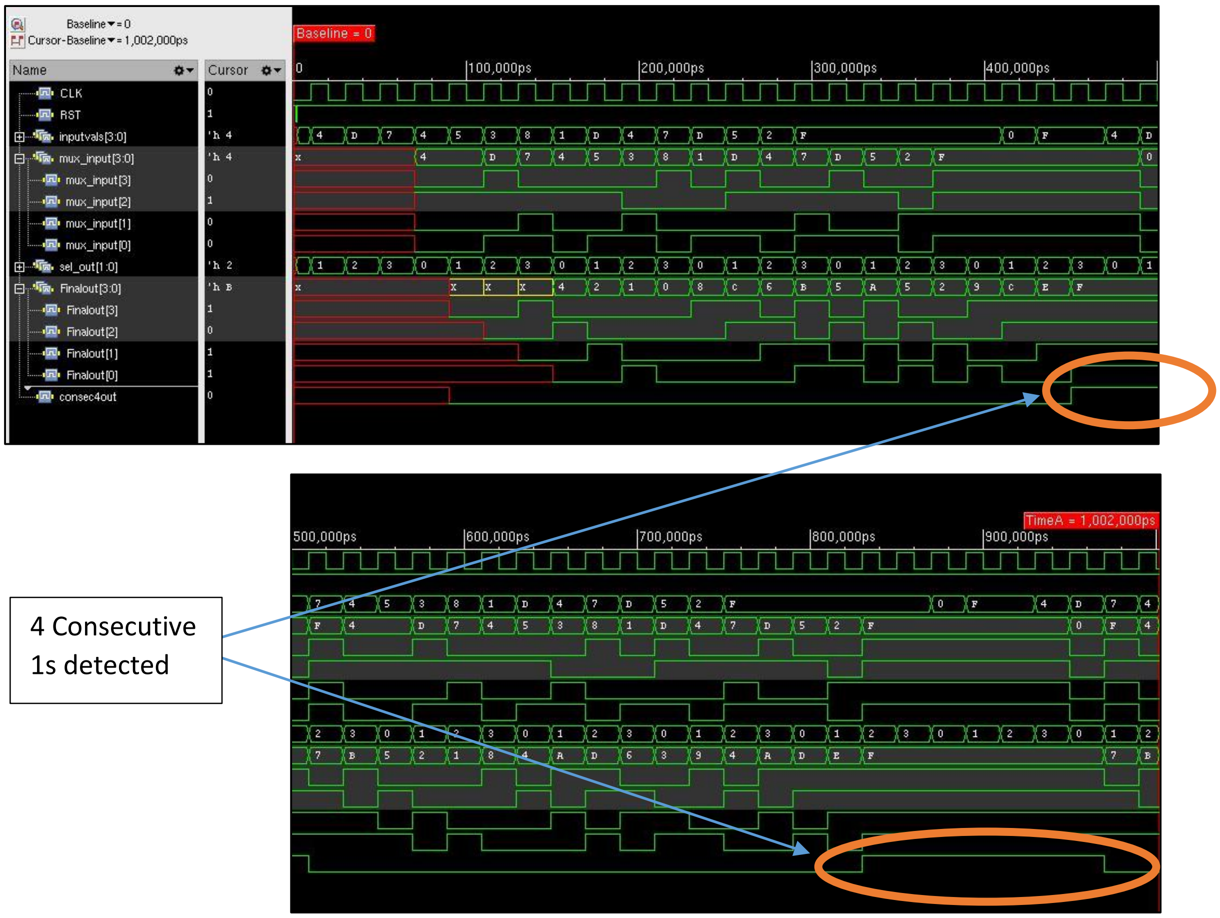
a. <u>All Zeroes:</u>



b. <u>All Ones:</u>



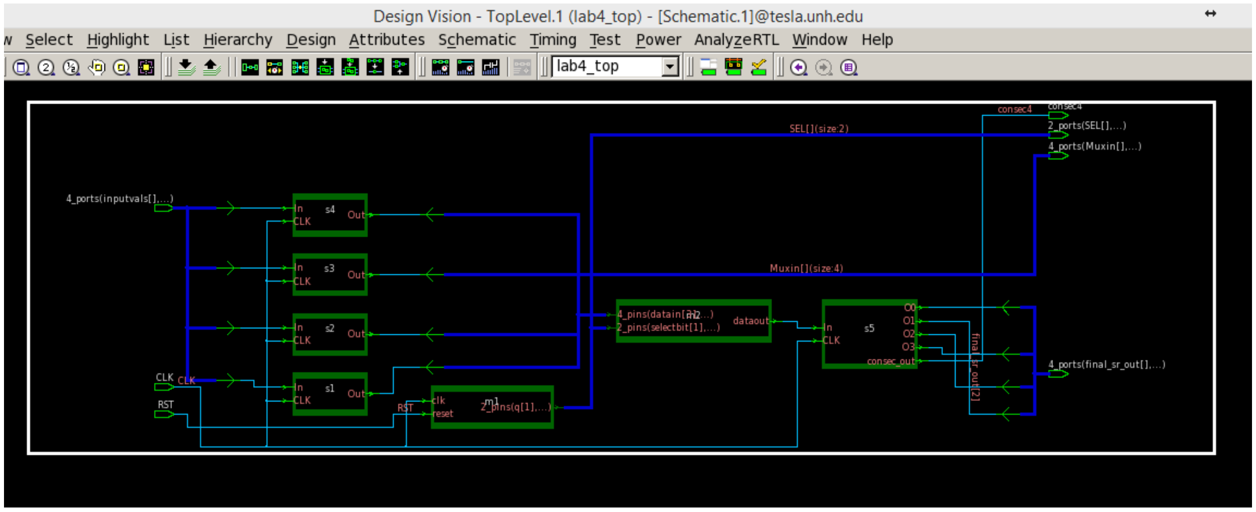
c. Random Input:



lab4_tb.v (Source code for the testbench):

```
`timescale 1ns/1ns
module lab4_tb;
    reg CLK, RST;
    wire consec4out;
    reg[3:0] inputvals;
    wire[3:0] mux_input, Finalout;
    wire[1:0] sel_out;

    lab4_top u1(.inputvals(inputvals), .Muxin(mux_input),
.final_sr_out(Finalout), .consec4(consec4out), .SEL(sel_out), .CLK(CLK),
.RST(RST));

    initial
    begin
      CLK = 1'b0;
      RST = 1'b1;
      #1 RST = 1'b0;
      #1 RST = 1'b1;
      //enable below for 1's input
      inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};
      //enable below for 0's input
      //inputvals[3:0] = {1'b0, 1'b0, 1'b0, 1'b0};
      #1000 $finish;
    end

    always
    begin
      #10 CLK = ~CLK;
    end

    always@(posedge CLK)
    begin
    //enable below for 'random' value input
    /*inputvals[3:0] = {1'b0, 1'b1, 1'b0, 1'b0};
    #20
    inputvals[3:0] = {1'b1, 1'b1, 1'b0, 1'b1};
    #20
    inputvals[3:0] = {1'b0, 1'b1, 1'b1, 1'b1};
    #20
    inputvals[3:0] = {1'b0, 1'b1, 1'b0, 1'b0};
    #20
    inputvals[3:0] = {1'b0, 1'b1, 1'b0, 1'b1};
    #20
    inputvals[3:0] = {1'b0, 1'b0, 1'b1, 1'b1};
    #20
    inputvals[3:0] = {1'b1, 1'b0, 1'b0, 1'b0};
    #20
    inputvals[3:0] = {1'b0, 1'b0, 1'b0, 1'b1};
    #20
    inputvals[3:0] = {1'b1, 1'b1, 1'b0, 1'b1};
    #20
    inputvals[3:0] = {1'b0, 1'b1, 1'b0, 1'b0};
    #20
    inputvals[3:0] = {1'b0, 1'b1, 1'b1, 1'b1};
    #20
```

```
     inputvals[3:0] = {1'b1, 1'b1, 1'b0, 1'b1};
     #20
     inputvals[3:0] = {1'b0, 1'b1, 1'b0, 1'b1};
     #20
     inputvals[3:0] = {1'b0, 1'b0, 1'b1, 1'b0};
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};  //begin 1's sequence to test 
output flag
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};
     #20
     inputvals[3:0] = {1'b0, 1'b0, 1'b0, 1'b0};  //0's to end test flag
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};
     #20
     inputvals[3:0] = {1'b1, 1'b1, 1'b1, 1'b1};*/
     end
endmodule
```



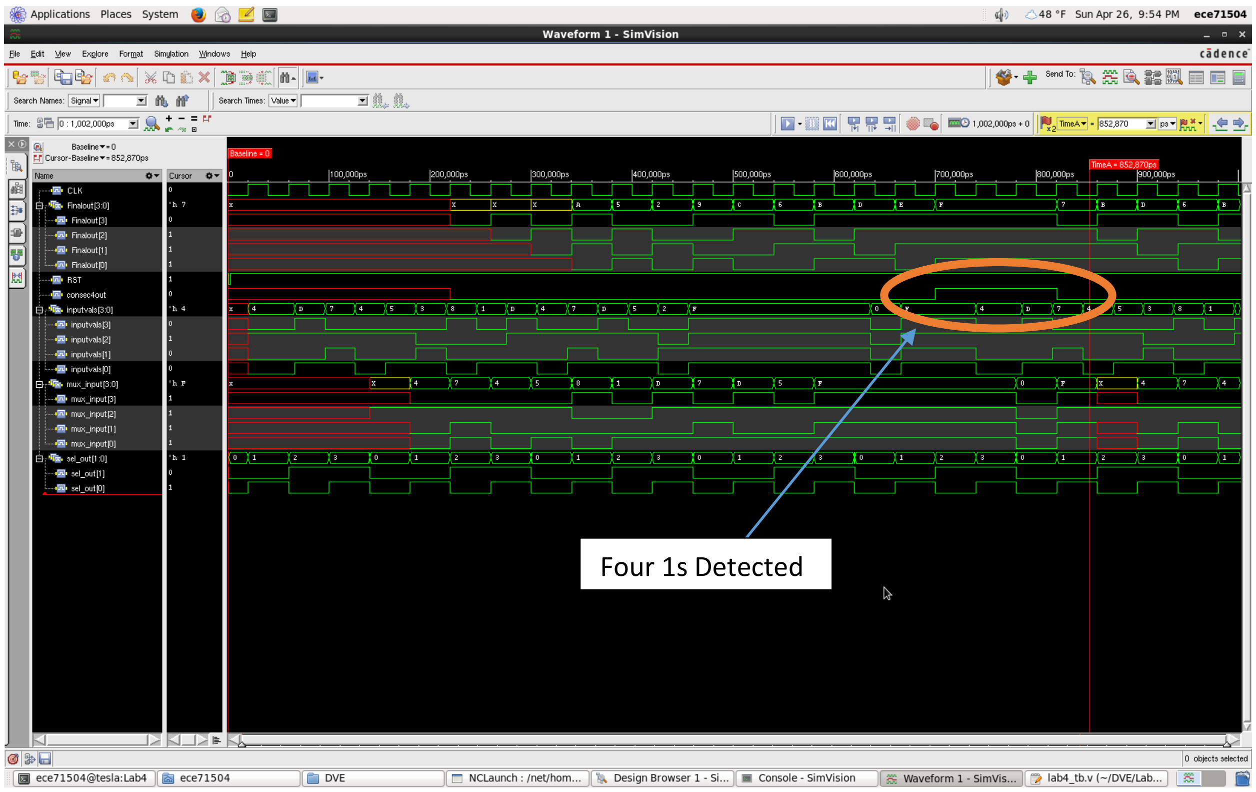
Synthesized Design

3. Sythesized netlist:

```
`timescale 1ns/1ns
module FIFO4_0 ( In, Out, CLK );
  input In, CLK;
  output Out;
  wire   D0, D1, D2;
  DFQD1 FF1 ( .D(In), .CP(CLK), .Q(D0) );
  DFQD1 FF2 ( .D(D0), .CP(CLK), .Q(D1) );
  DFQD1 FF3 ( .D(D1), .CP(CLK), .Q(D2) );
  DFQD1 FF4 ( .D(D2), .CP(CLK), .Q(Out) );
endmodule


module FIFO4_tapped ( In, O0, O1, O2, O3, consec_out, CLK );
  input In, CLK;
  output O0, O1, O2, O3, consec_out;


  DFQD1 FF1 ( .D(In), .CP(CLK), .Q(O0) );
  DFQD1 FF2 ( .D(O0), .CP(CLK), .Q(O1) );
  DFQD1 FF3 ( .D(O1), .CP(CLK), .Q(O2) );
  DFQD1 FF4 ( .D(O2), .CP(CLK), .Q(O3) );
  AN4D0 U1 ( .A1(O3), .A2(O2), .A3(O1), .A4(O0), .Z(consec_out) );
endmodule


module MOD4counter ( clk, reset, q );
  output [1:0] q;
  input clk, reset;
  wire   d1, n1;

  DFCNQD1 \q_reg[0]  ( .D(n1), .CP(clk), .CDN(reset), .Q(q[0]) );
  DFCNQD1 \q_reg[1]  ( .D(d1), .CP(clk), .CDN(reset), .Q(q[1]) );
  CKND0 U3 ( .I(q[0]), .ZN(n1) );
  CKXOR2D0 U4 ( .A1(q[1]), .A2(q[0]), .Z(d1) );
endmodule


module mux4_1 ( datain, selectbit, dataout );
  input [3:0] datain;
  input [1:0] selectbit;
  output dataout;


  MUX4D0 U1 ( .I0(datain[0]), .I1(datain[2]), .I2(datain[1]), .I3(datain[3]),
        .S0(selectbit[1]), .S1(selectbit[0]), .Z(dataout) );
endmodule


module FIFO4_1 ( In, Out, CLK );
  input In, CLK;
  output Out;
  wire   D0, D1, D2;

  DFQD1 FF1 ( .D(In), .CP(CLK), .Q(D0) );
  DFQD1 FF2 ( .D(D0), .CP(CLK), .Q(D1) );
  DFQD1 FF3 ( .D(D1), .CP(CLK), .Q(D2) );
  DFQD1 FF4 ( .D(D2), .CP(CLK), .Q(Out) );
endmodule


module FIFO4_2 ( In, Out, CLK );
```

```
  input In, CLK;
  output Out;
  wire   D0, D1, D2;

  DFQD1 FF1 ( .D(In), .CP(CLK), .Q(D0) );
  DFQD1 FF2 ( .D(D0), .CP(CLK), .Q(D1) );
  DFQD1 FF3 ( .D(D1), .CP(CLK), .Q(D2) );
  DFQD1 FF4 ( .D(D2), .CP(CLK), .Q(Out) );
endmodule


module FIFO4_3 ( In, Out, CLK );
  input In, CLK;
  output Out;
  wire   D0, D1, D2;

  DFQD1 FF1 ( .D(In), .CP(CLK), .Q(D0) );
  DFQD1 FF2 ( .D(D0), .CP(CLK), .Q(D1) );
  DFQD1 FF3 ( .D(D1), .CP(CLK), .Q(D2) );
  DFQD1 FF4 ( .D(D2), .CP(CLK), .Q(Out) );
endmodule


module lab4_netlist ( inputvals, Muxin, final_sr_out, consec4, SEL, CLK, RST
);
  input [3:0] inputvals;
  output [3:0] Muxin;
  output [3:0] final_sr_out;
  output [1:0] SEL;
  input CLK, RST;
  output consec4;
  wire   Muxout;

  FIFO4_0 s1 ( .In(inputvals[0]), .Out(Muxin[0]), .CLK(CLK) );
  FIFO4_3 s2 ( .In(inputvals[1]), .Out(Muxin[1]), .CLK(CLK) );
  FIFO4_2 s3 ( .In(inputvals[2]), .Out(Muxin[2]), .CLK(CLK) );
  FIFO4_1 s4 ( .In(inputvals[3]), .Out(Muxin[3]), .CLK(CLK) );
  FIFO4_tapped s5 ( .In(Muxout), .O0(final_sr_out[3]), .O1(final_sr_out[2]),
        .O2(final_sr_out[1]), .O3(final_sr_out[0]), .consec_out(consec4),
        .CLK(CLK) );
  MOD4counter m1 ( .clk(CLK), .reset(RST), .q(SEL) );
  mux4_1 m2 ( .datain(Muxin), .selectbit(SEL), .dataout(Muxout) );
endmodule
```

Post-synthesize simulation waveform that shows your design successfully detects four consecutive 1s:



NOTE: To obtain proper output after post-synthesis, we had to increase the period of clock since we were facing issues with setup and hold times for the MUX. The clock period was increased to 20ns from 10ns. Also, the inputs were changed every 30ns rather than 20ns.