

Seana Corners

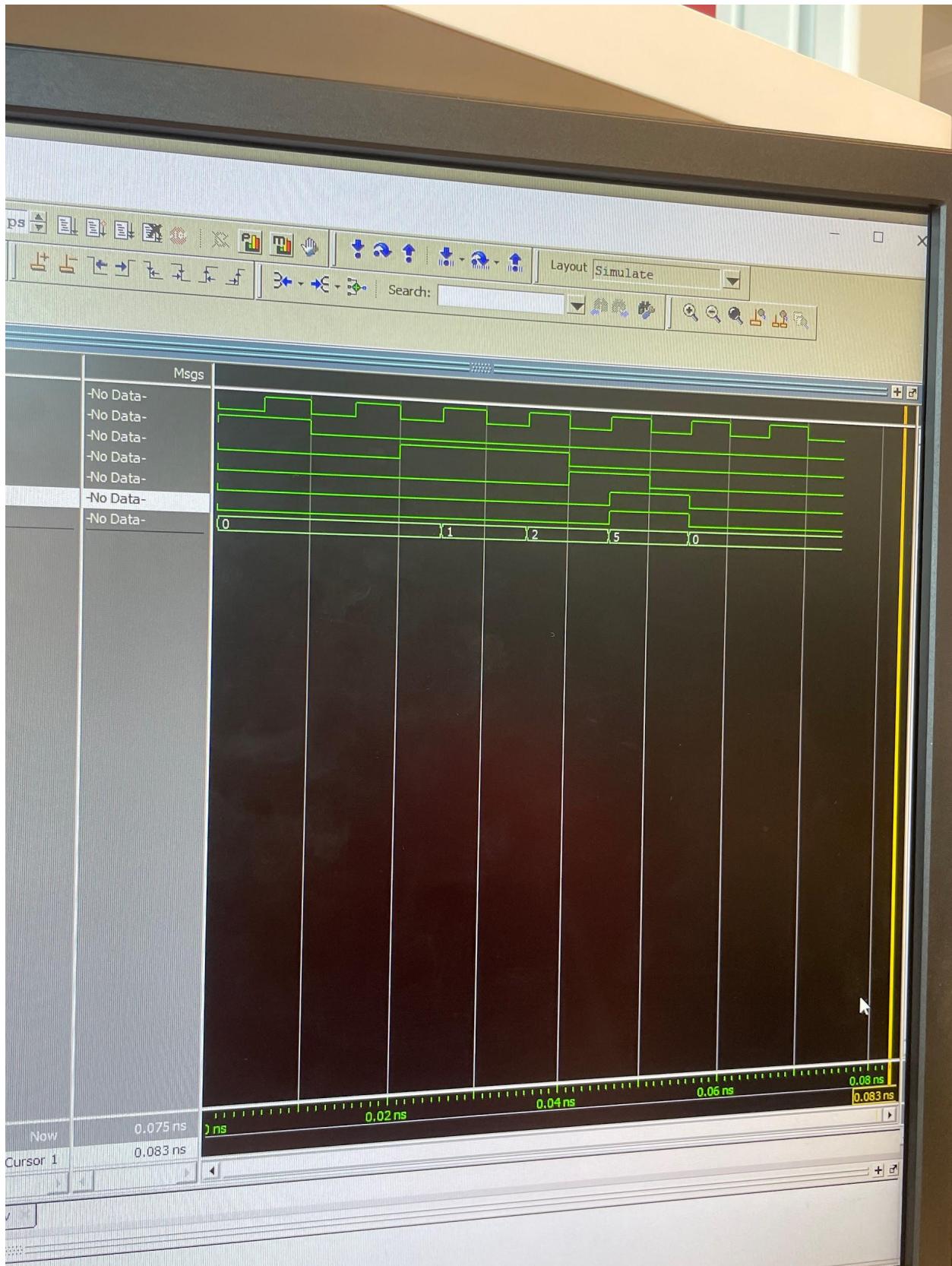
Ayden Dauenhauer

Lab 8 Post Lab Report

1. My state diagram and state tables were correct in my prelab.
2. I think that the longer Verilog module is more productive because it is more intuitive and easy to understand. Code that is easier to understand and makes more sense at first glance can lead to better understanding from other people. In a team setting, this is important so everyone can contribute and communicate effectively. Although the other version of the code is more concise, it is difficult for others to understand it in an efficient way.
3. Our design would change because we would add more inputs for more coins and bills. The state diagram for instance would completely change because it will have more combinations of possible changes to each state. The coins and bills would operate the same, just at different values. There is a greater chance of getting change (C) because the values of the inputs such as quarter (25) dollars (100, 500, 1000, etc) exceed the change limit of 15.

The screenshot shows a logic synthesis tool interface with a central code editor displaying Verilog code. The code defines a Moore machine with 16 states (S0 to S15) and two outputs (O and C). The logic is controlled by state assignments and a case statement.

```
// Output Logics
// Moore machine: output is only determined by state
always@(S)
begin
    case(S)
        S0:
            begin
                O = 0;
                C = 0;
            end
        S5:
            begin
                O = 0;
                C = 0;
            end
        S10:
            begin
                O = 0;
                C = 0;
            end
        S15:
            begin
                O = 1;
                C = 0;
            end
        S20:
            begin
                O = 1;
                C = 1;
            end
        default:
            begin
                O = 0;
                C = 0;
            end
    endcase
end
endmodule
```



```
20      initial begin
21          Reset = 1;
22          N = 0; D = 0;
23          #10 Reset = 0;
24          #10 N = 1; D = 0;
25          #10 N = 1; D = 0;
26          #10 N = 1; D = 0;
27          #10 N = 0; D = 0;
28          #25 $finish;
29      end
30
31      initial begin
32          Reset = 1;
33          N = 0; D = 0;
34          #10 Reset = 0;
35          #10 N = 1; D = 0;
36          #10 N = 1; D = 0;
37          #10 N = 0; D = 1;
38          #10 N = 0; D = 0;
39          #25 $finish;
40
41          Reset = 1;
42          N = 0; D = 0;
43          #10 Reset = 0;
44          #10 N = 0; D = 1;
45          #10 N = 0; D = 1;
46          #10 N = 0; D = 0;
47          #10 N = 0; D = 0;
48          #25 $finish;
49
50          Reset = 1;
51          N = 0; D = 0;
52          #10 Reset = 0;
53          #10 N = 1; D = 0;
54          #10 N = 0; D = 1;
55          #10 N = 0; D = 0;
56          #10 N = 0; D = 0;
57          #25 $finish;
58      end
```

b8

Tools Window Help

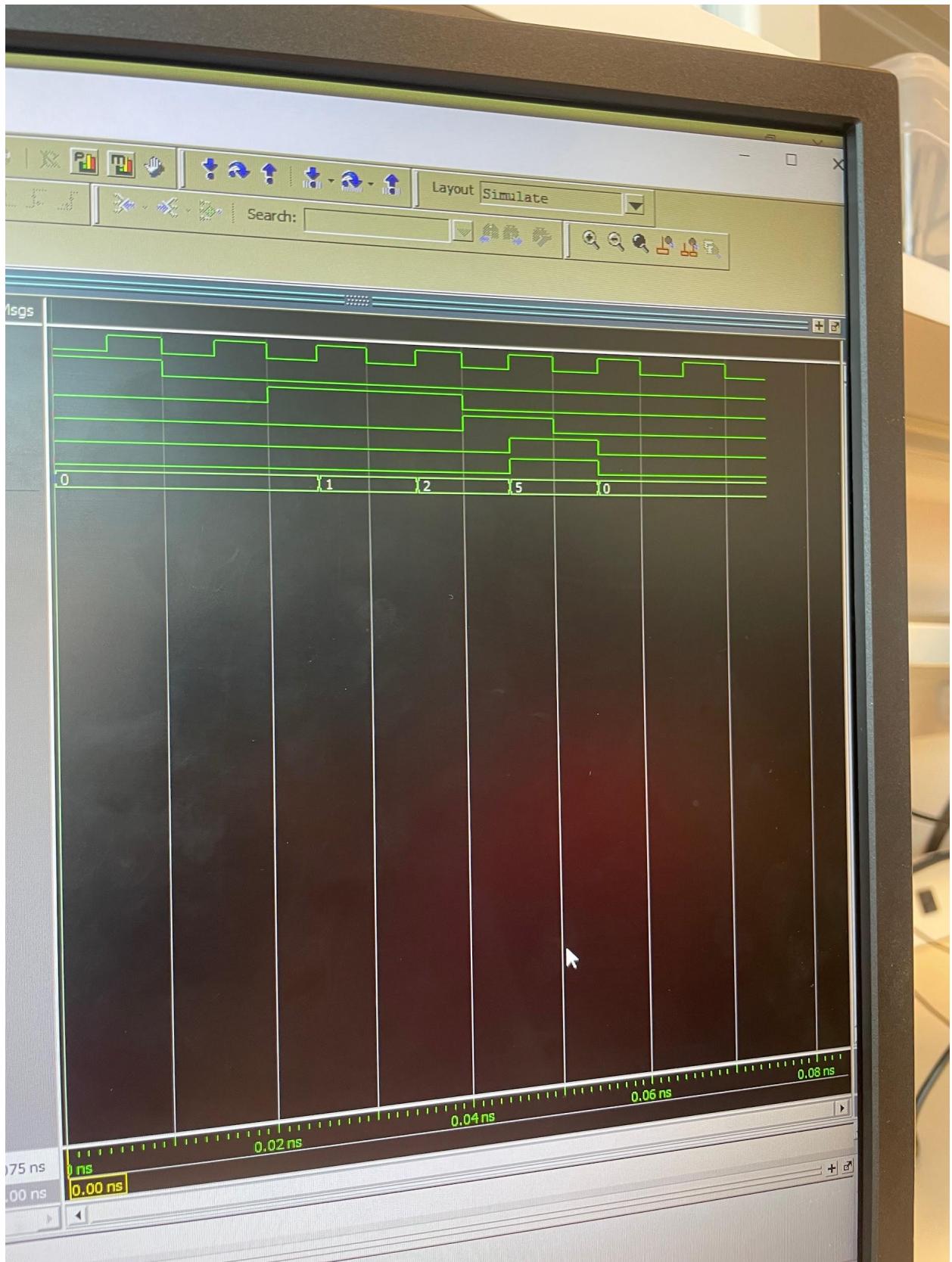
abc lab8.v X abc tb.v X abc lab8_part1.v (Read-Only) X abc lab8_part2.v (Read-Only) X Compilation Report - lab8 X

File Edit View Insert Cell Properties Window Help

1 module Lab8 (Clock, Reset, N, D, O, C, S);
2 input Clock, Reset; // Clock and Reset
3 input N, D; // Nickel and Dime sensors (active-high)
4 output O, C; // Open and Change (active-high)
5 output reg [2:0] S; // current state
6 wire [2:0] S_star; // next state
7
8 // Flip-flops for state (state memory)
9 always @(posedge Reset, posedge Clock)
10 if (Reset == 1) S <= 3'b000; // initialization
11 else S <= S_star;
12
13 // Next state logics
14 // input: S, N, D
15 // output: S_star
16 assign S_star[2] = ~S[2] & (S[0] & D | S[1] & D | S[1] & N);
17 assign S_star[1] = ~S[2] & (~S[1] & ~S[0] & D | S[0] & N | S[1] & ~N & ~D);
18 assign S_star[0] = ~S[2] & (~S[1] & ~S[0] & N | S[1] & D | S[0] & ~N & ~D);
19
20 // Output logics
21 // Moore machine: output is only determined by the current state (S), that is, input is S
22 assign O = S[2];
23 assign C = S[2] & S[0];
24
25 endmodule
26
27
28

Find... Find Next

NativeLink simulation (quartus_sh -t "c:/apps/intelfpga_lite/21.1/quartus/common/tcl/internal/nativeLink_simulation.rpt")
NativeLink execution see the NativeLink log file z:/ELEN_21L/lab8/lab8_nativeLink_simulation.rpt
NativeLink simulation (quartus_sh -t "c:/apps/intelfpga_lite/21.1/quartus/common/tcl/internal/nativeLink_simulation.rpt")
NativeLink execution see the NativeLink log file z:/ELEN_21L/lab8/lab8_nativeLink_simulation.rpt
NativeLink simulation (quartus_sh -t "c:/apps/intelfpga_lite/21.1/quartus/common/tcl/internal/nativeLink_simulation.rpt")
NativeLink execution see the NativeLink log file z:/ELEN_21L/lab8/lab8_nativeLink_simulation.rpt
NativeLink simulation (quartus_sh -t "c:/apps/intelfpga_lite/21.1/quartus/common/tcl/internal/nativeLink_simulation.rpt")
NativeLink execution see the NativeLink log file z:/ELEN_21L/lab8/lab8_nativeLink_simulation.rpt



ion - Z:/ELEN_21L/lab8/lab8 - lab8

File Assignments Processing Tools Window Help

lab8

lab8.v tb.v Compilation Report - lab8

Instance

I5F29C7

Task

Design

Design & Synthesis

Place & Route

Compiler (Generate programm

Analysis

Testlist Writer

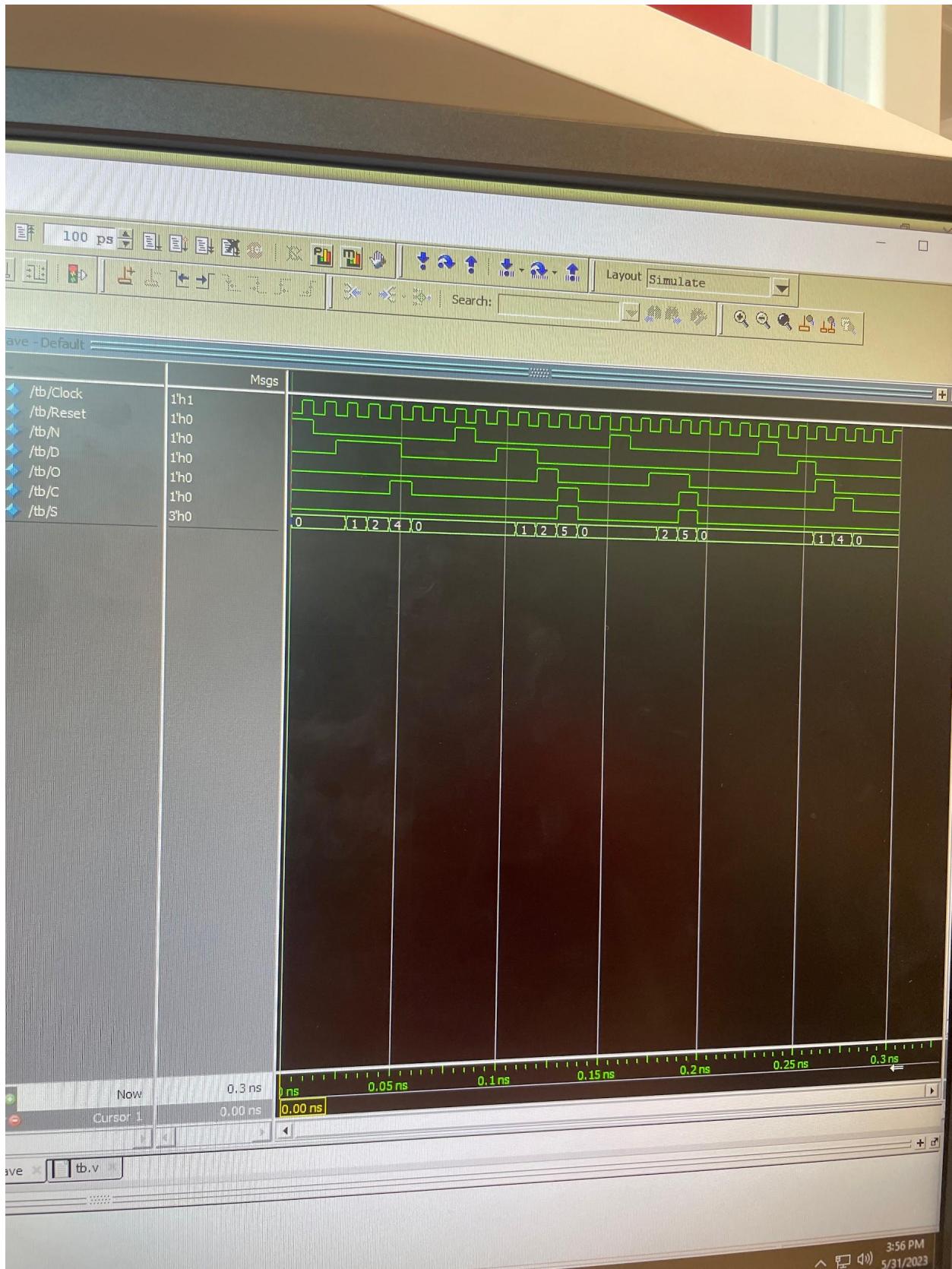
Messages

<<Filter>>

Find... Find Next

```
10 parameter [2:0] S0=3'b000, S5=3'b001, S10=3'b010, S15=3'b100, S20=3'b101;
11 // Flip-flops for state (state memory)
12 always @(posedge Reset, posedge Clock)
13     if (Reset == 1) S <= 3'b000; // initialization
14     else S <= S_star;
15 // Next state logics
16 // input: S, N, D
17 // output: S_star
18 always@(S,N,D)
19 begin
20     case(S)
21         S0:
22             if (N==0 && D==0) S_star = S0;
23             else if (N==0 && D==1) S_star = S10;
24             else if (N==1 && D==0) S_star = S5;
25             else S_star = S0;
26         S5:
27             if (N==0 && D==0) S_star = S5;
28             else if (N==0 && D==1) S_star = S15;
29             else if (N==1 && D==0) S_star = S10;
30             else S_star = S5;
31         S10:
32             if (N==0 && D==0) S_star = S10;
33             else if (N==0 && D==1) S_star = S20;
34             else if (N==1 && D==0) S_star = S15;
35             else S_star = S10;
36         S15:
37             if (N==0 && D==0) S_star = S0;
38             else S_star = S15;
39         S20:
40             if (N==0 && D==0) S_star = S0;
41             else S_star = S20;
42         default:
43             S_star = S0;
44     endcase
45 end
46 // output logics
```

Message
Successfully launched NativeLink simulation (quartus_sh -t "c:/apps/intelfpga_lite/21.1/quartus/common/tcl/internal/rpt" messages from NativeLink execution see the NativeLink log file Z:/ELEN_21L/lab8/lab8_nativelink_simulation.rpt)



The screenshot shows the Quartus II software interface with three tabs open: `lab8.v`, `tb.v`, and `Compilation Report - lab8`. The `lab8.v` tab is active, displaying Verilog code. The code consists of four initial blocks, each with a different sequence of assignments for variables `Reset`, `N`, and `D`, followed by a `#25` delay. The final block ends with a `$finish;` statement. The code is color-coded: `initial`, `begin`, `end`, and `#` are in blue; variable names (`Reset`, `N`, `D`) are in red; and assignment operators (`=`) are in green. The Quartus II toolbar and status bar are visible at the top and bottom of the window.

```
19 initial begin
20     Reset = 1;
21     N = 0; D = 0;
22     #10 Reset = 0;
23     #10 N = 1; D = 0;
24     #10 N = 1; D = 0;
25     #10 N = 1; D = 0;
26     #10 N = 0; D = 0;
27     #25;
28
29     Reset = 1;
30     N = 0; D = 0;
31     #10 Reset = 0;
32     #10 N = 1; D = 0;
33     #10 N = 1; D = 0;
34     #10 N = 0; D = 1;
35     #10 N = 0; D = 0;
36     #25;
37
38     Reset = 1;
39     N = 0; D = 0;
40     #10 Reset = 0;
41     #10 N = 0; D = 1;
42     #10 N = 0; D = 1;
43     #10 N = 0; D = 0;
44     #10 N = 0; D = 0;
45     #25;
46
47     Reset = 1;
48     N = 0; D = 0;
49     #10 Reset = 0;
50     #10 N = 1; D = 0;
51     #10 N = 0; D = 1;
52     #10 N = 0; D = 0;
53     #10 N = 0; D = 0;
54     #25 $finish;
55
56 end
57
```

Fully Launched NativeLink simulation (quartus_sh -t "c:/apps/intelfpga_lite/2
messages from NativeLink execution see the NativeLink log file Z:/ELEN_21L/lab8/

Verilog:

```

module lab8 (Clock, Reset, N, D, O, C, S);

input Clock, Reset; // Clock and Reset
input N, D; // Nickel and Dime sensors (active-high)
output O, C; // Open and Change (active-high)

output reg [2:0] S; // current state
wire [2:0] S_star; // next state

// Flip-flops for state (state memory)
always @(posedge Reset, posedge Clock)
    if (Reset == 1) S <= 3'b000; // initialization
    else S <= S_star;

// Next state logics
// input: S, N, D
// output: S_star
assign S_star[2] = ~S[2] & (S[0] & D | S[1] & D | S[1] & N);
assign S_star[1] = ~S[2] & (~S[1] & ~S[0] & D | S[0] & N | S[1] & ~N & ~D);
assign S_star[0] = ~S[2] & (~S[1] & ~S[0] & N | S[1] & D | S[0] & ~N & ~D);

// Output logics
// Moore machine: output is only determined by the current state (S), that is, input is S
assign O = S[2];
assign C = S[2] & S[0];

endmodule

```

```

module lab8 (Clock, Reset, N, D, O, C, S);

input Clock, Reset; // Clock and Reset
input N, D; // Nickel and Dime sensors (active-high)
output reg O, C; // Open and Change (active-high)

output reg [2:0] S; // current state
reg [2:0] S_star; // next state

parameter [2:0] S0=3'b000, S5=3'b001, S10=3'b010, S15=3'b100, S20=3'b101;

```

```

// Flip-flops for state (state memory)
always @(posedge Reset, posedge Clock)
    if (Reset == 1) S <= 3'b000; // initialization
    else S <= S_star;

// Next state logics
// input: S, N, D
// output: S_star
always@(S,N,D)
begin
    case(S)
        S0:
            if (N==0 && D==0) S_star = S0;
            else if (N==0 && D==1) S_star = S10;
            else if (N==1 && D==0) S_star = S5;
            else S_star = S0;
        S5:
            if (N==0 && D==0) S_star = S5;
            else if (N==0 && D==1) S_star = S15;
            else if (N==1 && D==0) S_star = S10;
            else S_star = S5;
        S10:
            if (N==0 && D==0) S_star = S10;
            else if (N==0 && D==1) S_star = S20;
            else if (N==1 && D==0) S_star = S15;
            else S_star = S10;
        S15:
            if (N==0 && D==0) S_star = S0;
            else S_star = S15;
        S20:
            if (N==0 && D==0) S_star = S0;
            else S_star = S20;
        default:
            S_star = S0;
    endcase
end

// Output logics
// Moore machine: output is only determined by the current state (S), that is, input is S
always@(S)
begin
    case(S)
        S0:
            begin
                O = 0;
            end

```

```
C = 0;  
end  
S5:  
begin  
    O = 0;  
    C = 0;  
end  
S10:  
begin  
    O = 0;  
    C = 0;  
end  
S15:  
begin  
    O = 1;  
    C = 0;  
end  
S20:  
begin  
    O = 1;  
    C = 1;  
end  
default:  
begin  
    O = 0;  
    C = 0;  
end  
endcase  
end  
endmodule
```