Dylan Thornburg & Alan Rieger
5/30/2023

Lab 8: Vending Machine

Introduction:
In this lab we are simulating a vending machine through the use of Moore-style state machines. The two possible inputs are Dime (10 cents) and Nickel (5 cents). No FPGA, just simulation for this lab.

Procedures: After fixing some code and pre lab errors, we worked to get all eight perfect simulations. There were a lot of problems with module one but none with module two. Ultimately we got it to work after learning how to properly do the K-map thing.

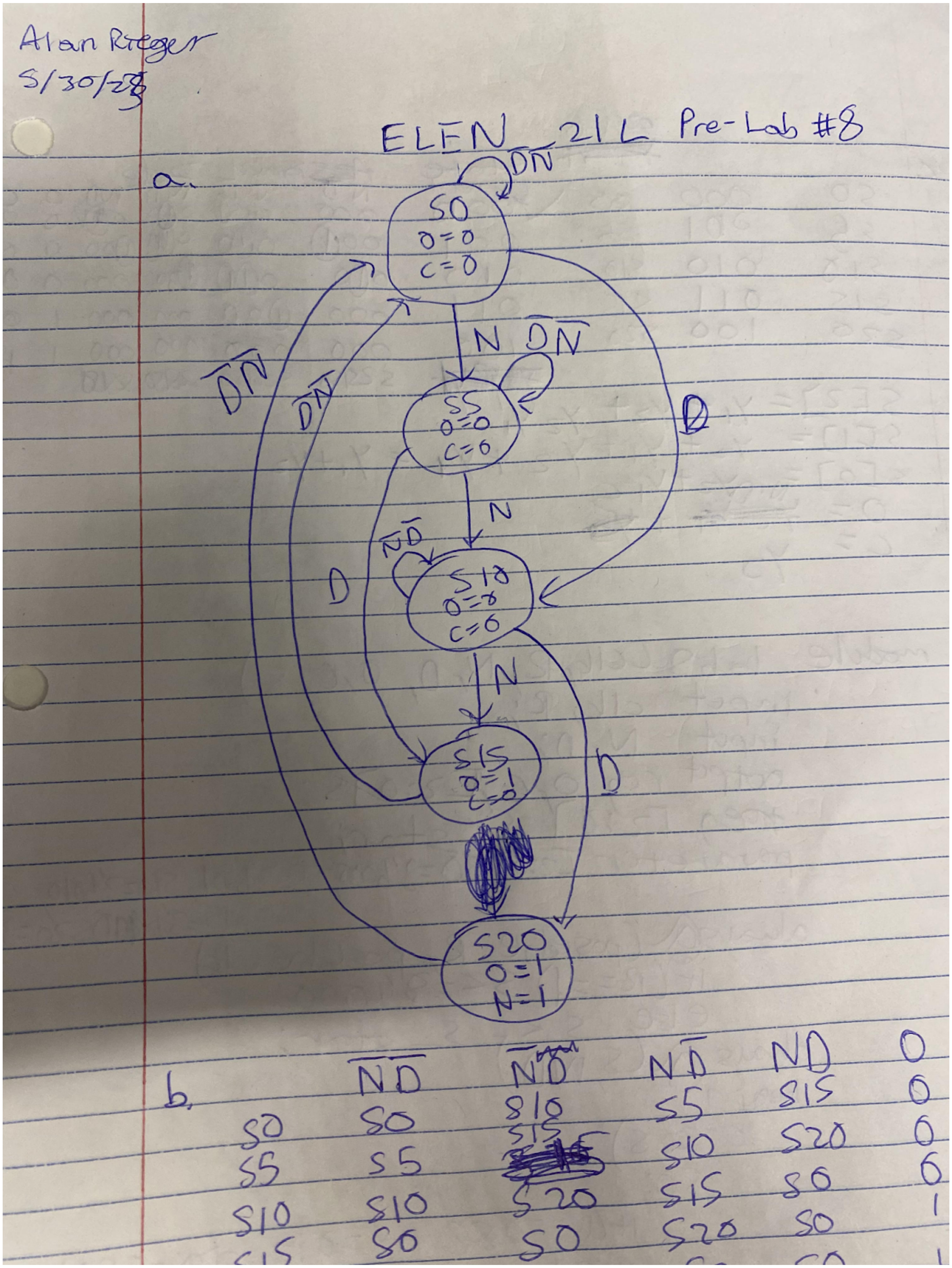
Conclusion: This lab was grueling and rough. Using module 2 was significantly easier and quicker than module 1. It just made more sense to solve for the next-states instead of solving for the next-state bits.

Questions:
Were the state diagrams/state tables you created in your pre-lab correct or not? If it was incorrect, in what way was it incorrect? What do you think led you to your incorrect diagram/table?

They were almost fully correct. Only S20 was wrong. The idea of not having a change output definitely led to this error as the example we did in class didn't output change so we got the two confused.

If your pre-lab was incorrect, provide a corrected answer.

Corrected state diagram (any incorrect code submitted will be fixed in the coding pictures below):



Alan Rieger
5/30/23

ELEN 21L Pre-Lab #8

a.

$\overline{D}\overline{N}$
S0
O=0
C=0
N
$\overline{D}\overline{N}$
D
S5
O=0
C=0
$\overline{D}\overline{N}$
$\overline{D}\overline{N}$
N
$\overline{N}\overline{D}$
D
S10
O=0
C=0
N
D
S15
O=1
C=0
S20
O=1
N=1

b.

	$\overline{ND}$	$\overline{N}D$	$N\overline{D}$	ND	0
S0	S0	S10	S5	S15	0
S5	S5	S15 ~~S15~~	S10	S20	0
S10	S10	S20	S15	S0	6
S15	S0	S0	S20	S0	1
					1

Comparing the two different implementation methods, which one do you think is more productive (from the perspective of specification)? Why do you think so?

Module 2's method is more productive. We understood it better and were able to make it the solution quicker and thus more productive.

How will your design change if you have to accept all types of coins (nickels, dimes, pennies and quarters). What about if you accept dollar bills?

If we had to accept all coins and potentially dollars, we would have to increase the amount of states and code for those states respectively.

Pictures:
Figure 1: Verilog Code Module 1:

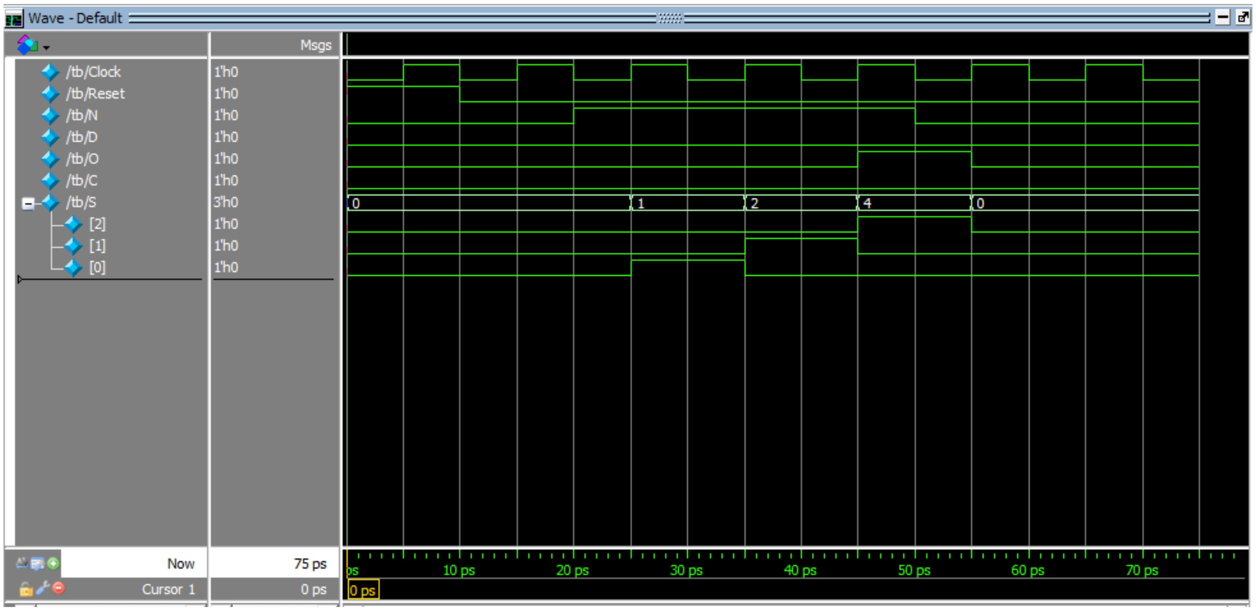
```
module Lab8 (Clock, Reset, N, D, O, C, S);

  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] & ((~N & ~D & S[1]) | (S[0] & N) | (~S[1] & ~ S[0] & D));
  assign S_star[0] = ~S[2] & ((S[1] & D) | (S[0] & ~N & ~D) | (~S[1] & ~S[0] & N));

  // 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
```

Figure 2:Verilog Code Module 2:

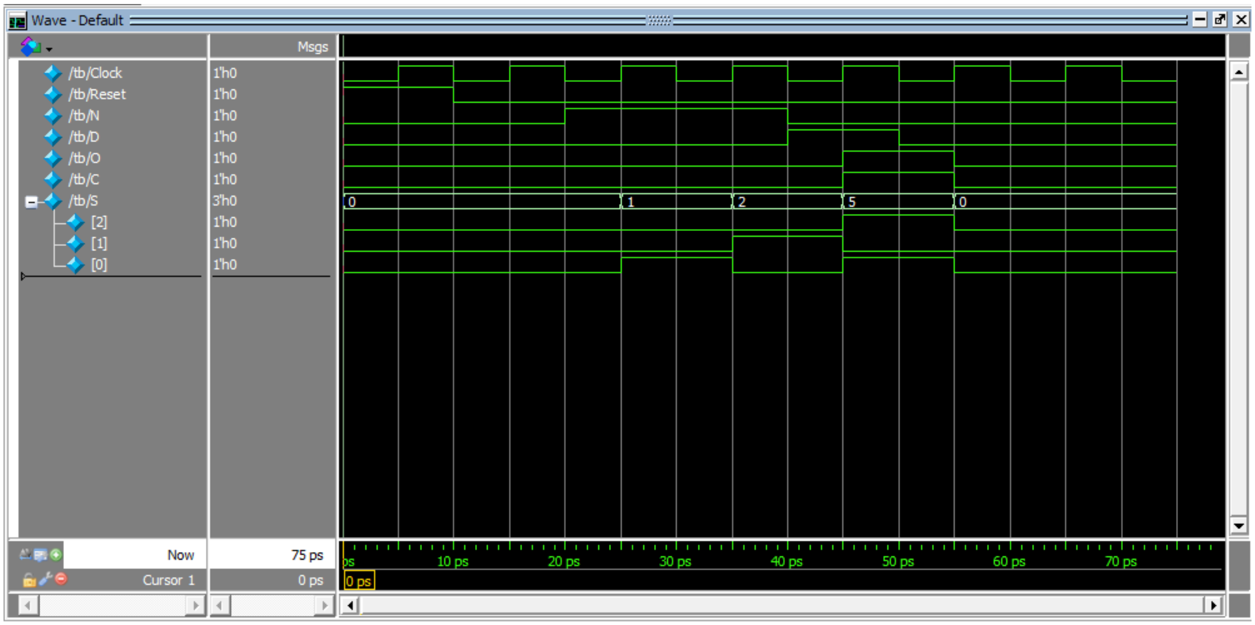
```
module Lab8 (Clock, Reset, N, D, O, C, S);
  input Clock, Reset;    // Clock and Reset
  input N, D;            // Nickel and Dime sensors (active-high)
  output reg O, C;       // Open and Change (active-high)
  output reg [2:0] S;    // current state
  reg [2:0] S_star;      // next state
  parameter [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;
          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
```

Figure 3: Wave Diagrams for Module 1:

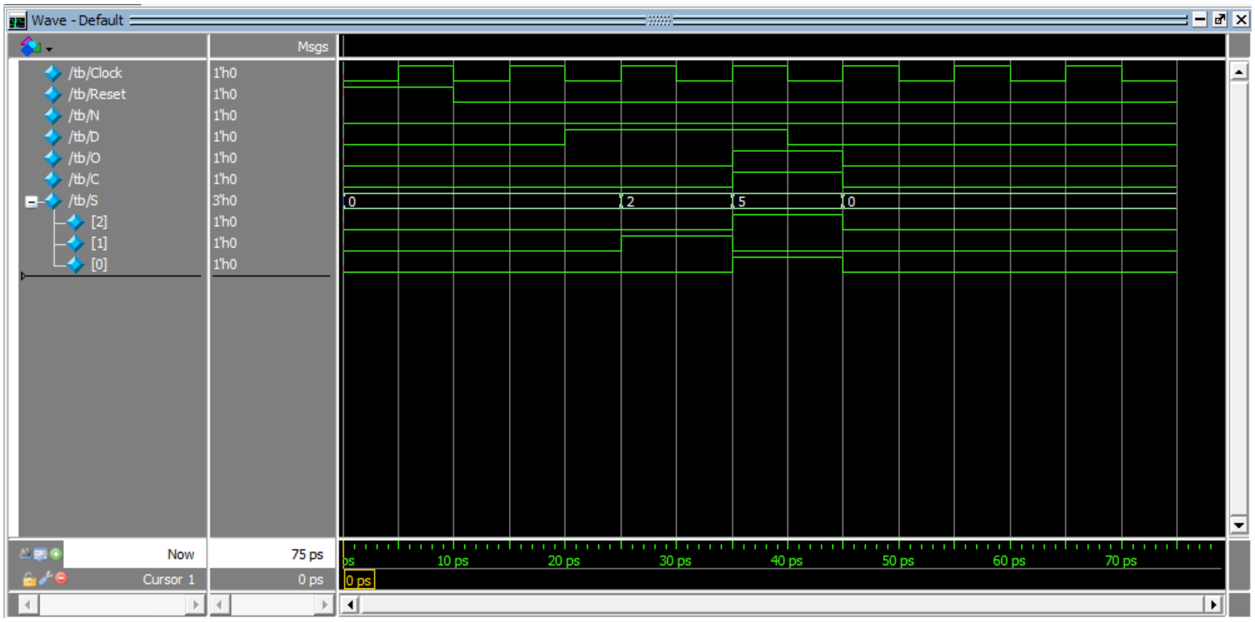
Nickel-Nickel-Nickel



Nickel-Nickel-Dime



Dime-Dime



Nickel-Dime

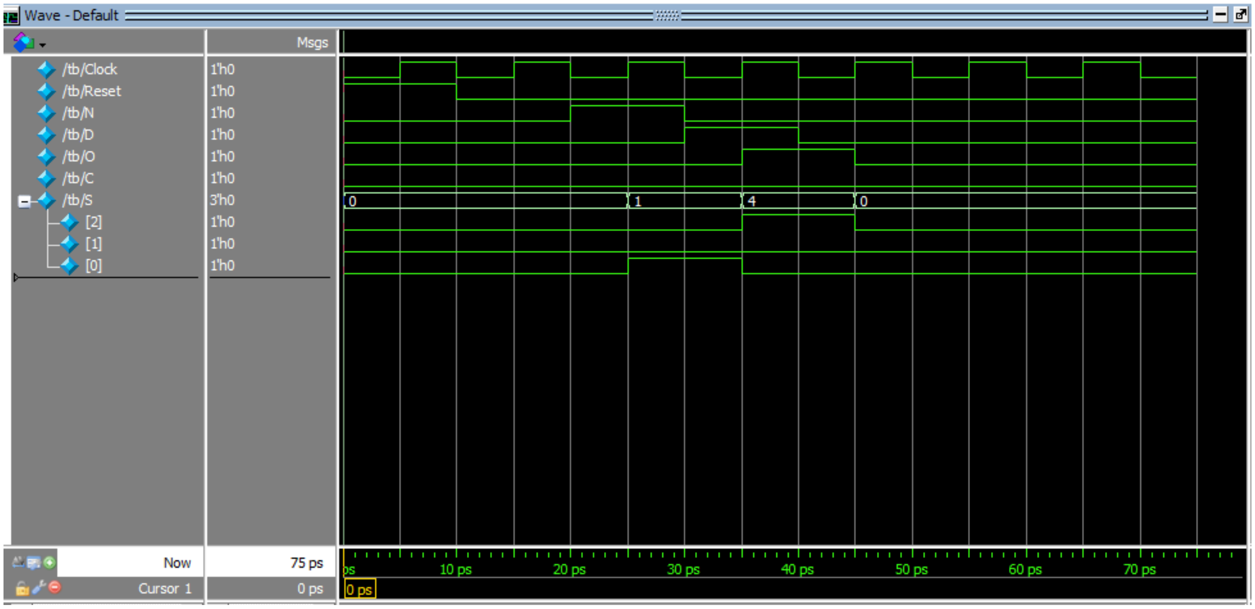
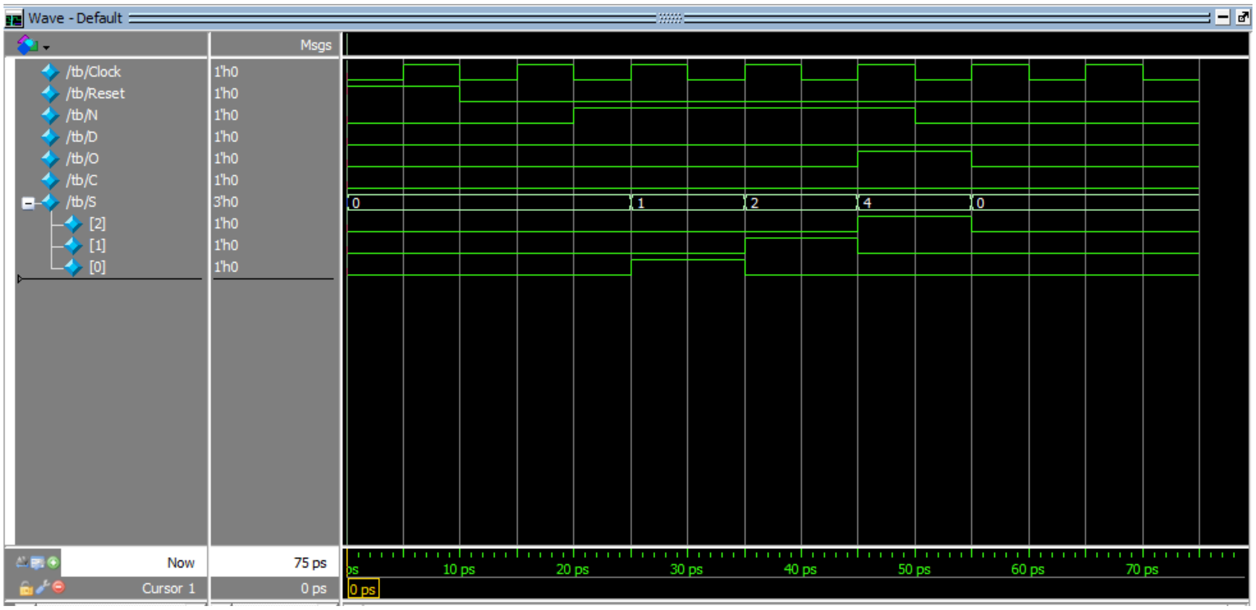
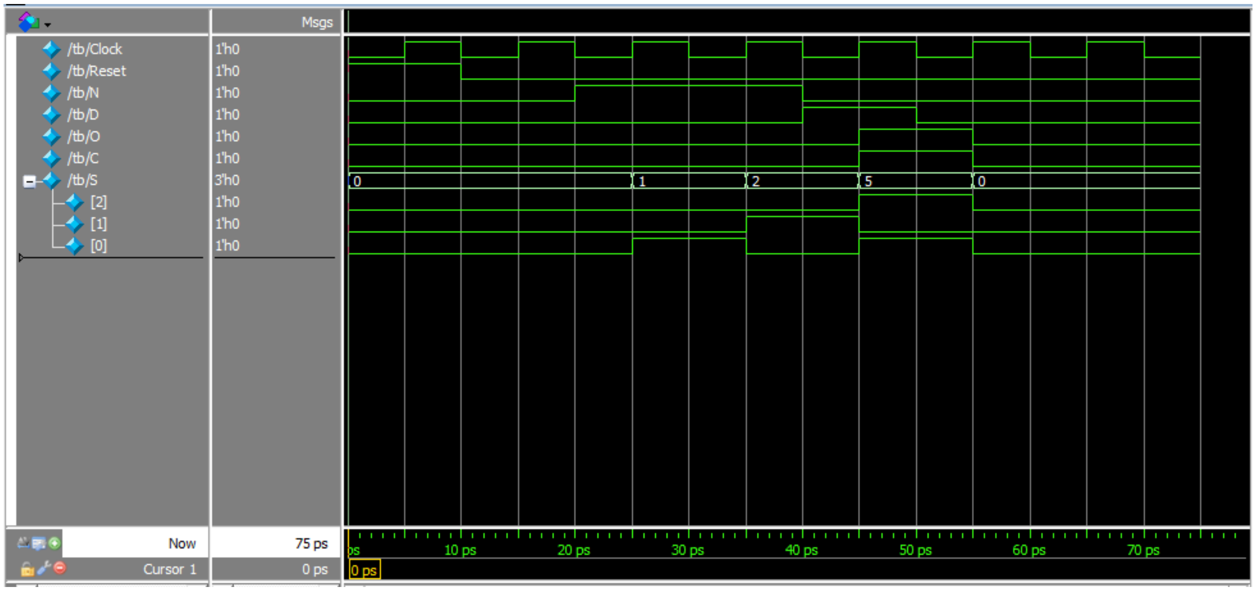


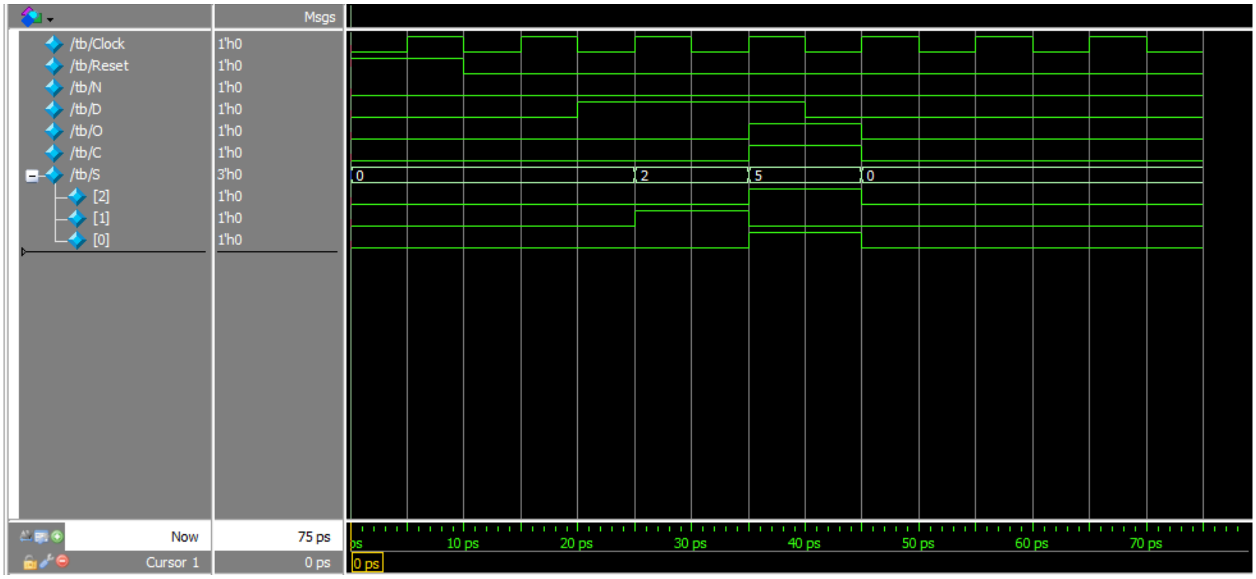
Figure 4: Wave Diagrams for Module 2:
Nickel-Nickel-Nickel



Nickel-Nickel-Dime

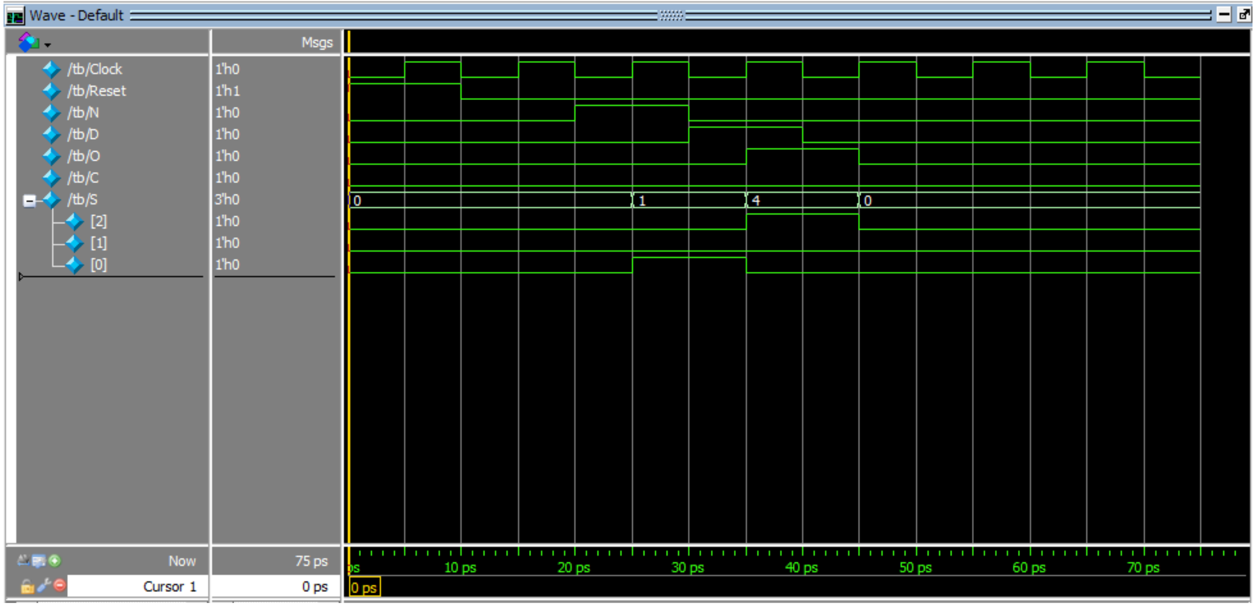


Dime-Dime



	Msgs
/tb/Clock	1'h0
/tb/Reset	1'h0
/tb/N	1'h0
/tb/D	1'h0
/tb/O	1'h0
/tb/C	1'h0
/tb/S	3'h0
[2]	1'h0
[1]	1'h0
[0]	1'h0
Now	75 ps
Cursor 1	0 ps

Nickel-Dime



Wave - Default

	Msgs
/tb/Clock	1'h0
/tb/Reset	1'h1
/tb/N	1'h0
/tb/D	1'h0
/tb/O	1'h0
/tb/C	1'h0
/tb/S	3'h0
[2]	1'h0
[1]	1'h0
[0]	1'h0
Now	75 ps
Cursor 1	0 ps