Department of Electrical Engineering and Technology

College of Engineering and Technology

Mindanao State University – Iligan Institute of Technology

.

Laboratory Report 2 - Vending Machine

In partial fulfillment for the course

ECE 134.1 —Introduction to Digital VLSI Design Laboratory

Submitted to

Prof. Kevin O. Maglinte

Faculty, DEET

COET, MSU – IIT

Submitted by

Bryan Jeff C. Saycon

2018-0807

BS ECE – IV

INTRODUCTION

Verilog is a Hardware Description Language (HDL). It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip−flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits. In this particular activity, we design a 3-peso cost item Vending Machine using Quartus Prime Lite Edition using the lessons discussed in the Laboratory.

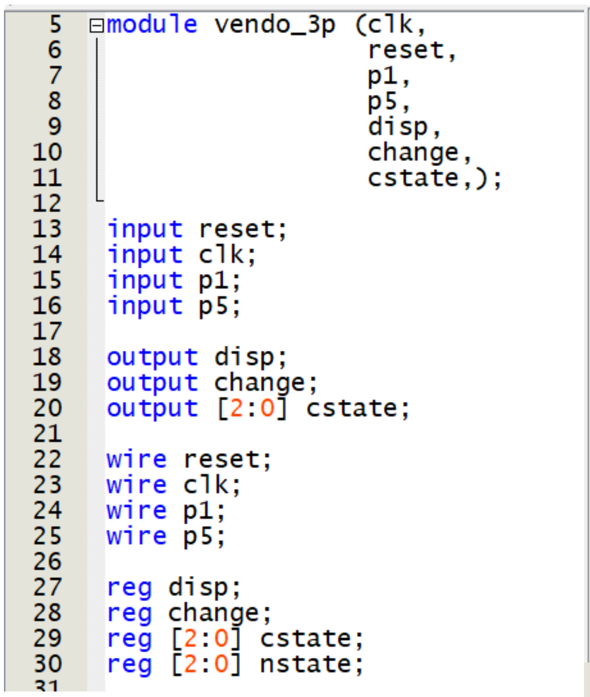
OBJECTIVES

Use Quartus Prime Lite Edition and ModelSim in coding (in Verilog HDL format), compiling, and simulation of a system.

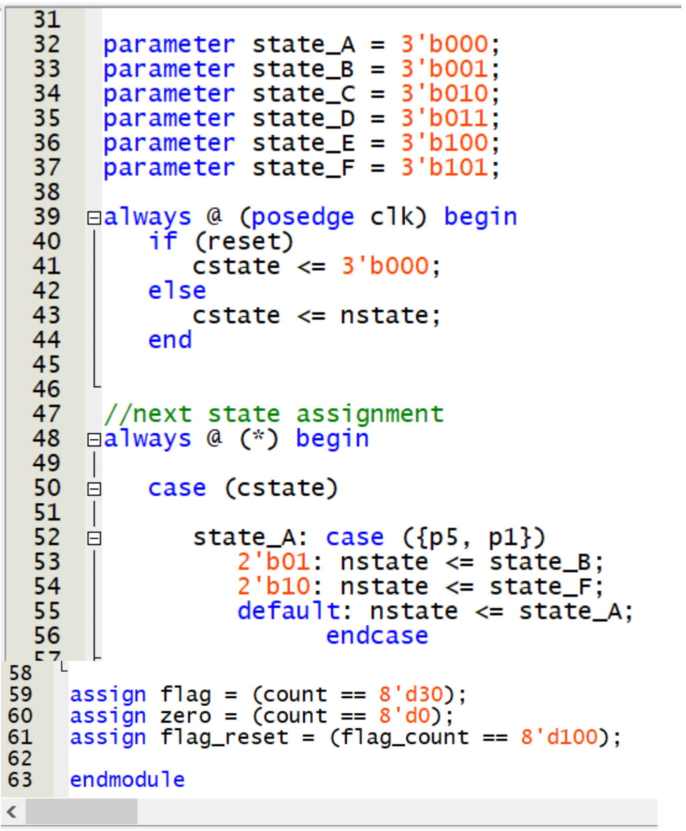
Properly design a 3-peso cost item Vending Machine with 1-Peso input and 5-peso input.

Discuss the simulation process using screen record and voice-over.

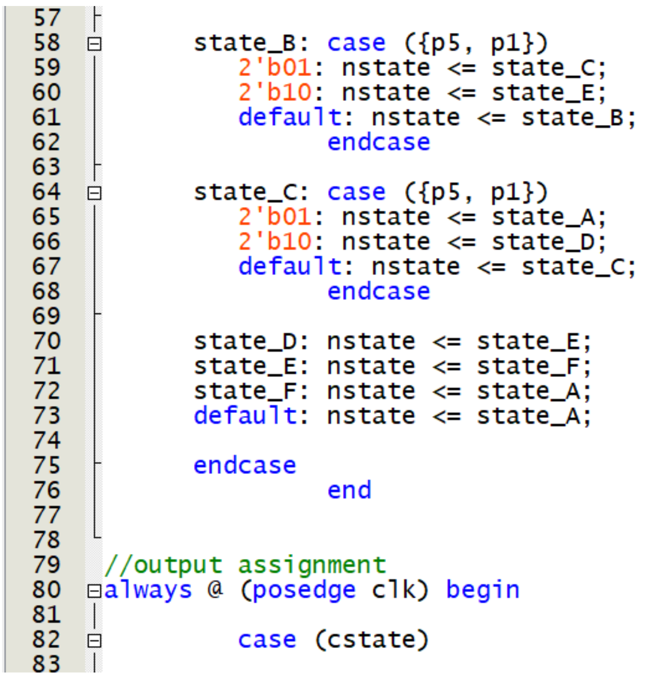
COUNTER VERILOG CODE



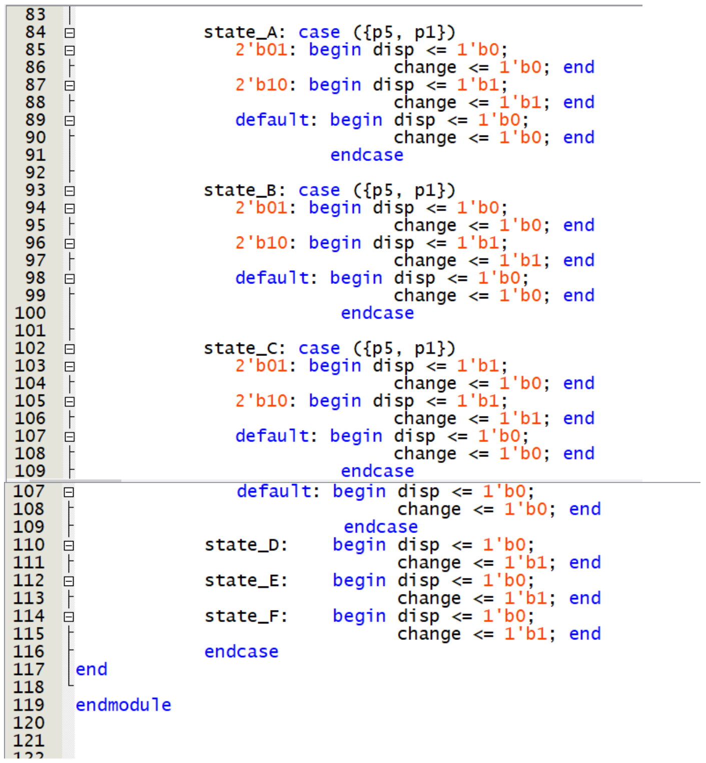
```
module vendo_3p (clk,
                 reset,
                 p1,
                 p5,
                 disp,
                 change,
                 cstate,);

input reset;
input clk;
input p1;
input p5;

output disp;
output change;
output [2:0] cstate;

wire reset;
wire clk;
wire p1;
wire p5;

reg disp;
reg change;
reg [2:0] cstate;
reg [2:0] nstate;
```



```

parameter state_A = 3'b000;
parameter state_B = 3'b001;
parameter state_C = 3'b010;
parameter state_D = 3'b011;
parameter state_E = 3'b100;
parameter state_F = 3'b101;

always @ (posedge clk) begin
    if (reset)
        cstate <= 3'b000;
    else
        cstate <= nstate;
    end


//next state assignment
always @ (*) begin

    case (cstate)

        state_A: case ({p5, p1})
            2'b01: nstate <= state_B;
            2'b10: nstate <= state_F;
            default: nstate <= state_A;
                endcase

assign flag = (count == 8'd30);
assign zero = (count == 8'd0);
assign flag_reset = (flag_count == 8'd100);

endmodule
```



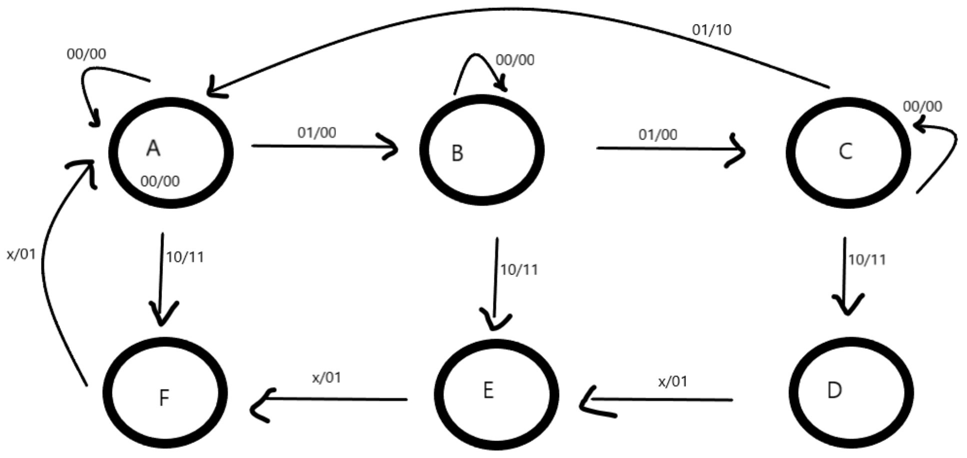
```
        state_B: case ({p5, p1})
            2'b01: nstate <= state_C;
            2'b10: nstate <= state_E;
            default: nstate <= state_B;
                    endcase

        state_C: case ({p5, p1})
            2'b01: nstate <= state_A;
            2'b10: nstate <= state_D;
            default: nstate <= state_C;
                    endcase

        state_D: nstate <= state_E;
        state_E: nstate <= state_F;
        state_F: nstate <= state_A;
        default: nstate <= state_A;

        endcase
                end

//output assignment
always @ (posedge clk) begin

            case (cstate)
```



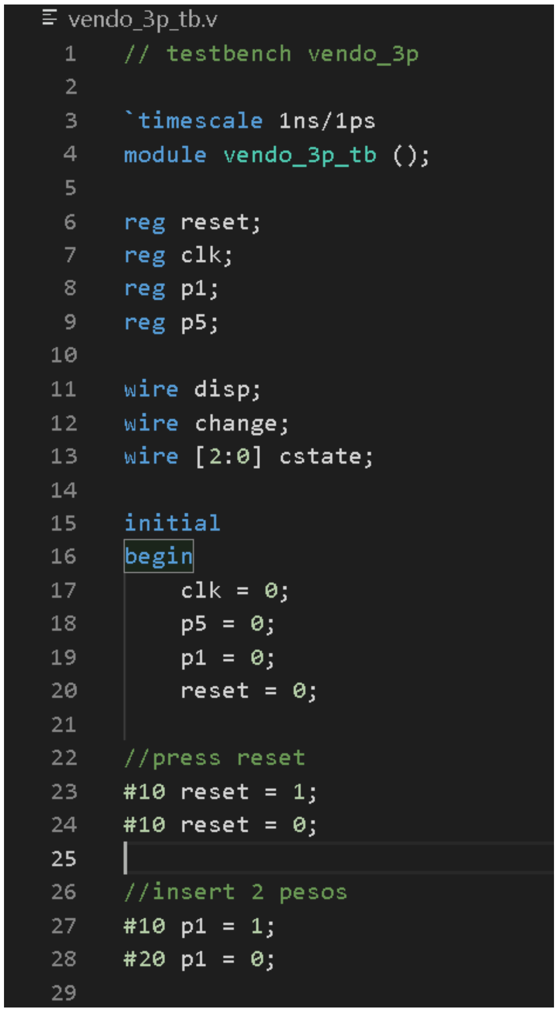
```

            state_A: case ({p5, p1})
               2'b01: begin disp <= 1'b0;
                            change <= 1'b0; end
               2'b10: begin disp <= 1'b1;
                            change <= 1'b1; end
               default: begin disp <= 1'b0;
                            change <= 1'b0; end
                        endcase

            state_B: case ({p5, p1})
               2'b01: begin disp <= 1'b0;
                            change <= 1'b0; end
               2'b10: begin disp <= 1'b1;
                            change <= 1'b1; end
               default: begin disp <= 1'b0;
                            change <= 1'b0; end
                         endcase

            state_C: case ({p5, p1})
               2'b01: begin disp <= 1'b1;
                            change <= 1'b0; end
               2'b10: begin disp <= 1'b1;
                            change <= 1'b1; end
               default: begin disp <= 1'b0;
                            change <= 1'b0; end
                         endcase
```

```
               default: begin disp <= 1'b0;
                             change <= 1'b0; end
                         endcase
            state_D:     begin disp <= 1'b0;
                             change <= 1'b1; end
            state_E:     begin disp <= 1'b0;
                             change <= 1'b1; end
            state_F:     begin disp <= 1'b0;
                             change <= 1'b1; end
            endcase
end

endmodule
```

FSM DIAGRAM {p1, p5 / dispense, change}

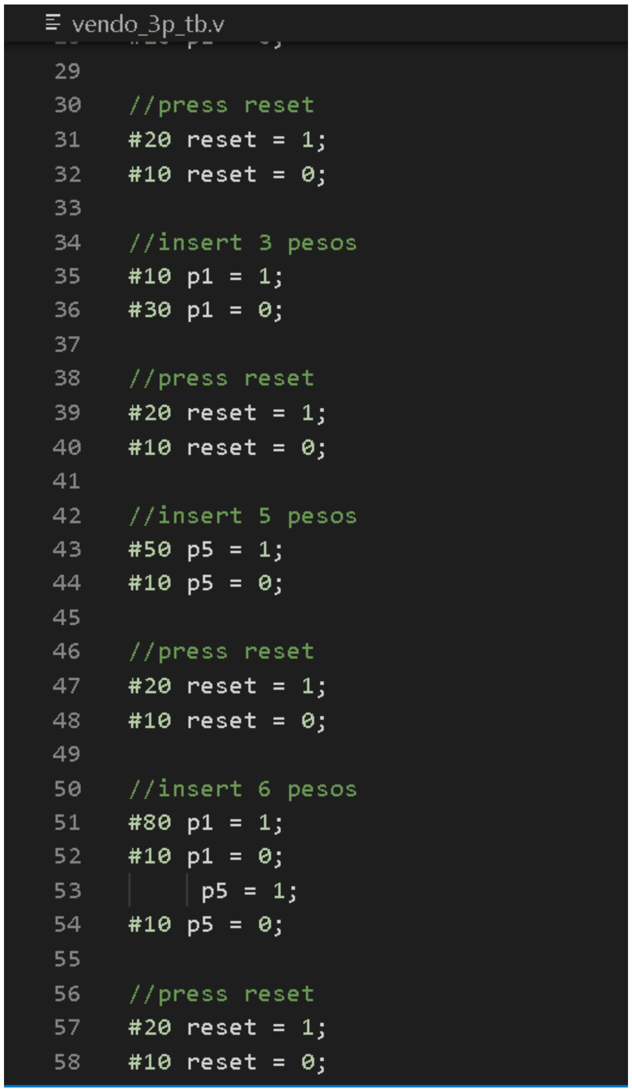


TEST BENCH CODE



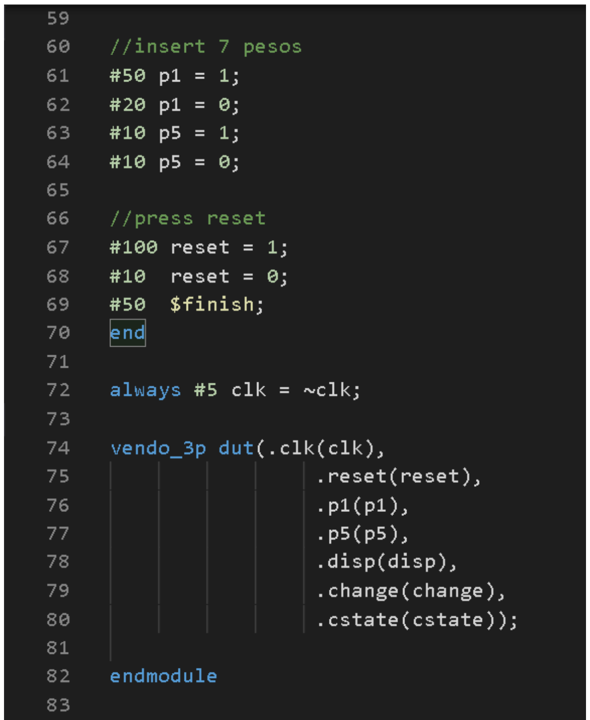
vendo_3p_tb.v

```verilog
// testbench vendo_3p

`timescale 1ns/1ps
module vendo_3p_tb ();

reg reset;
reg clk;
reg p1;
reg p5;

wire disp;
wire change;
wire [2:0] cstate;

initial
begin
    clk = 0;
    p5 = 0;
    p1 = 0;
    reset = 0;

//press reset
#10 reset = 1;
#10 reset = 0;

//insert 2 pesos
#10 p1 = 1;
#20 p1 = 0;

```



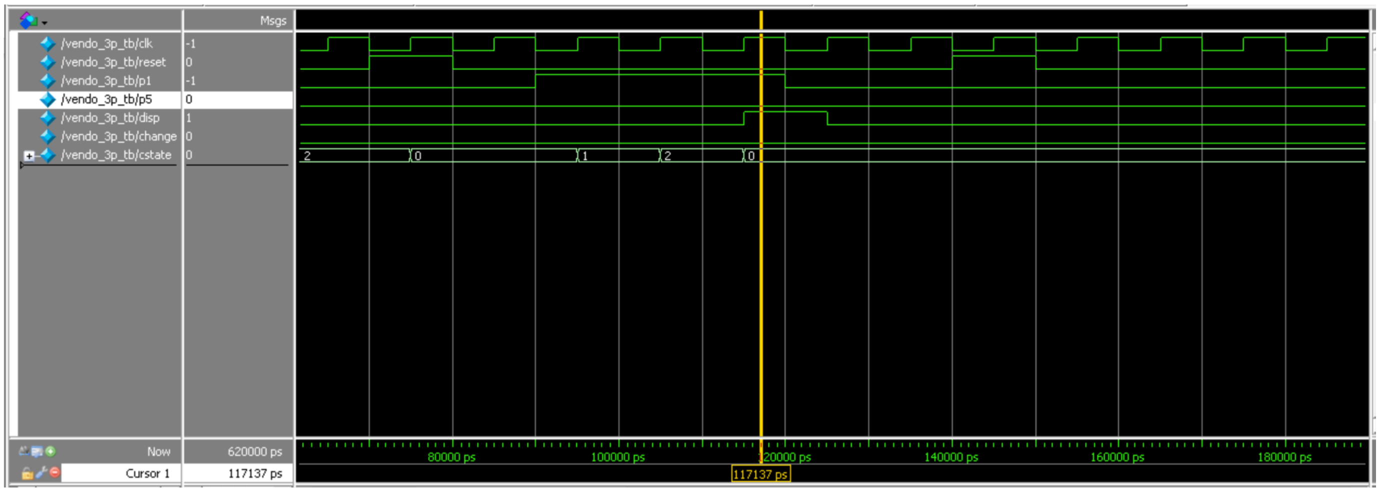
vendo_3p_tb.v

```verilog

//press reset
#20 reset = 1;
#10 reset = 0;

//insert 3 pesos
#10 p1 = 1;
#30 p1 = 0;

//press reset
#20 reset = 1;
#10 reset = 0;

//insert 5 pesos
#50 p5 = 1;
#10 p5 = 0;

//press reset
#20 reset = 1;
#10 reset = 0;

//insert 6 pesos
#80 p1 = 1;
#10 p1 = 0;
        p5 = 1;
#10 p5 = 0;

//press reset
#20 reset = 1;
#10 reset = 0;
```

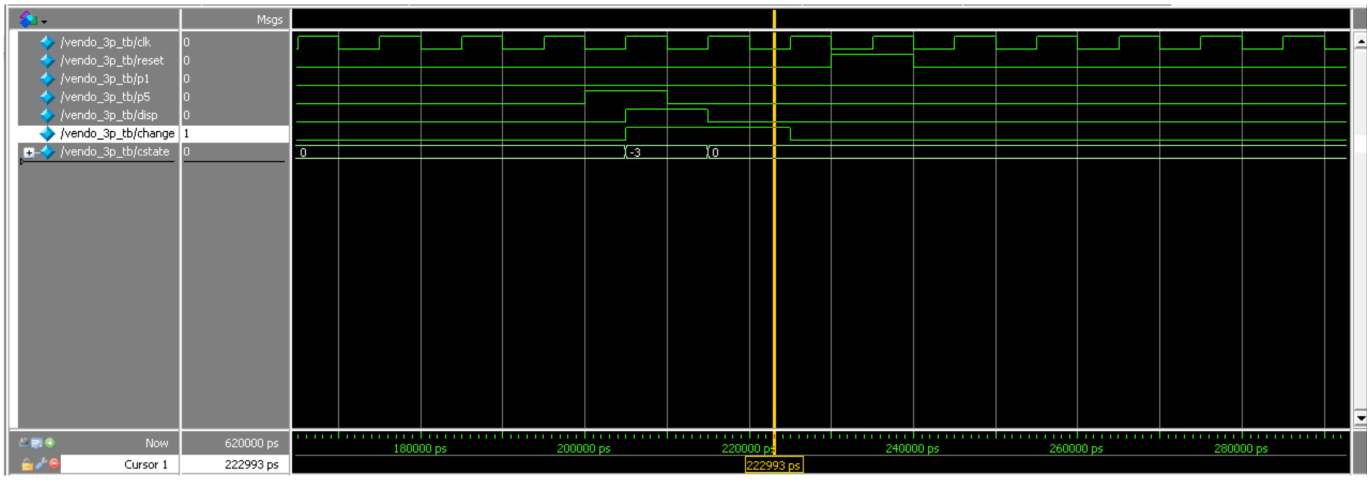


```verilog

//insert 7 pesos
#50 p1 = 1;
#20 p1 = 0;
#10 p5 = 1;
#10 p5 = 0;

//press reset
#100 reset = 1;
#10  reset = 0;
#50  $finish;
end

always #5 clk = ~clk;

vendo_3p dut(.clk(clk),
                 .reset(reset),
                 .p1(p1),
                 .p5(p5),
                 .disp(disp),
                 .change(change),
                 .cstate(cstate));

endmodule

```

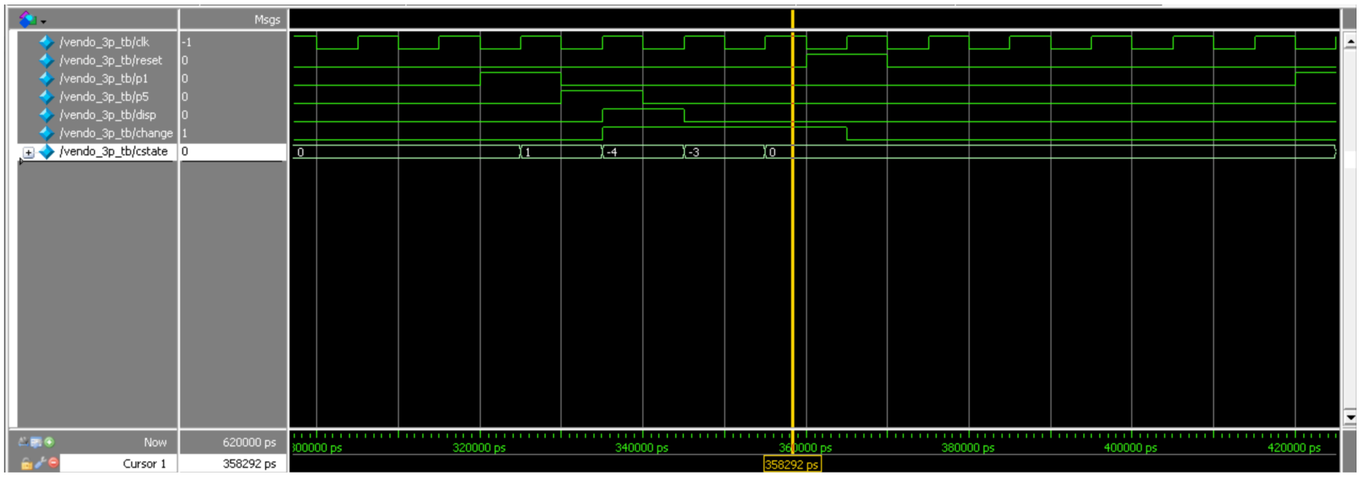
RESULTS



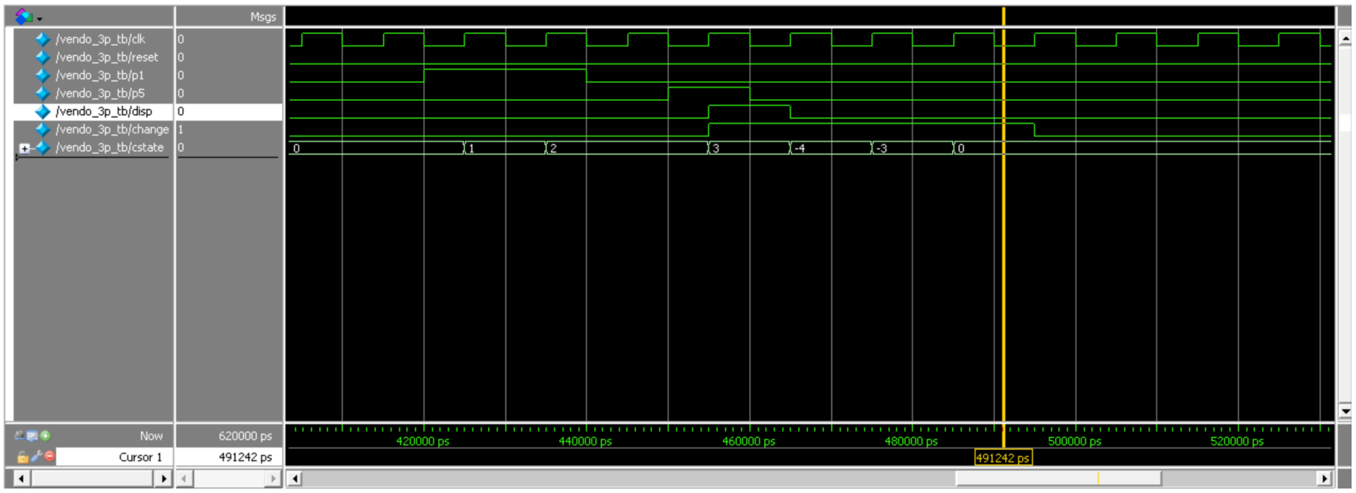
The screenshot above shows the dispense of a 3-peso cost item with three 1-peso coins inserted in the vending machine.



The screenshot above shows the dispense of the item with a 5-peso coin inputted in the machine, the result shows the machine drops 2 instances of change in response to 5 peso input for a 3-peso cost item.



The screenshot above shows the result of a 6-peso total input, first the user puts 1-peso coin then a 5-peso coin dispensing the item and 3 instances of change.



The screenshot above shows the result of a 7-peso total input, where the user puts two 1-peso coins then a 5-peso coin resulting in a dispensation of the 3-peso cost item and 4 instances of change.

CONCLUSION

The simulation of the program follows exactly what is formulated in the FSM Diagram. Everytime the input total reaches 3 it immediately dispenses the item and drops the change if it is 5, 6 or 7 in total which is already calculated in the FSM diagram. The program works fine for every instance as long as the counter doesn't reset interruptly to the input, then the program works just fine.

LABORATORY QUESTION

Why can't combinational circuits be modeled by Finite State Machines?

Combinational circuits can't be modeled by finite state machines because those types don't use memory. The logic behind finite state machines is that it takes several changes every clock cycle where the previous state of the input affects the output. And which is the reason it is not efficient on combinational circuits because of the absence of memory.