Lab 5: State Machine

CpE Section 1

Instructor: Telles, Eric

Friday: 4:30 pm – 6:50pm

Thao, Shammah

Part 1:

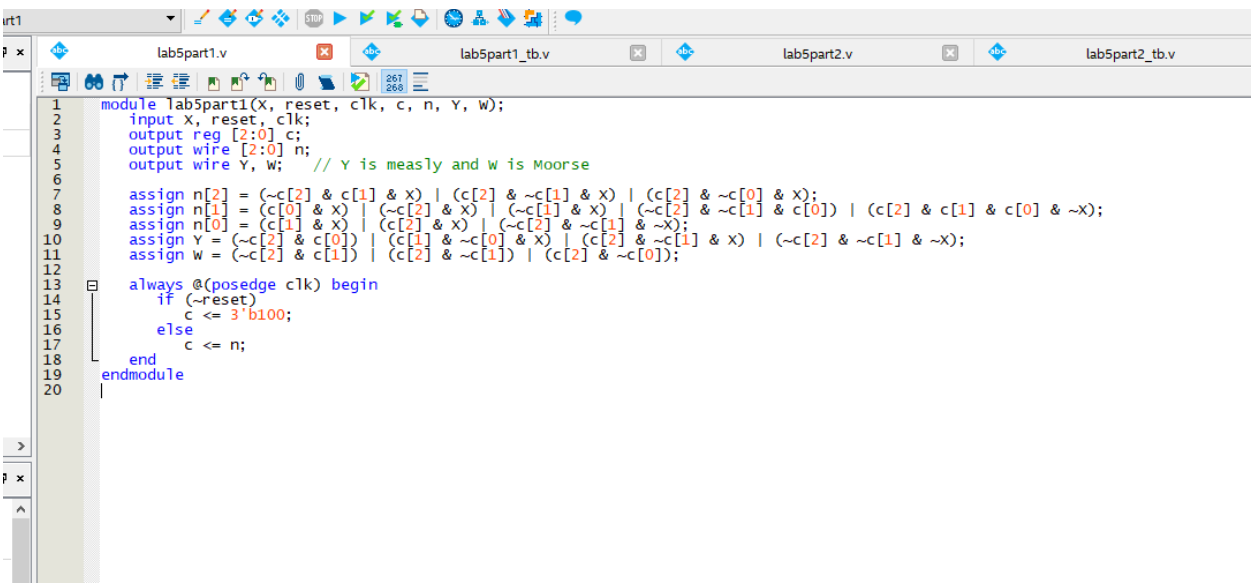
Description:

We are suppose to create a K-map using the truth table from a state machine diagraim that was given to use. We then are to make a Verilog coding that assign 4 different type of output.

Procedure:

Design a state machine solution for the state diagram in Figure 5-1. Use D flip-flops and only equations (assign statements) to implement this design. (No “if else” OR “case” statements can be used here, [just for RESET]). Use K-maps to find the equations for each of the inputs to the D flip-flops. Implement your equations from the K-maps into Verilog code, and bring your Verilog file to lab.

Engineering data:



```
module lab5part1(X, reset, clk, c, n, Y, W);
   input X, reset, clk;
   output reg [2:0] c;
   output wire [2:0] n;
   output wire Y, W;   // Y is measly and W is Moorse

   assign n[2] = (~c[2] & c[1] & X) | (c[2] & ~c[1] & X) | (c[2] & ~c[0] & X);
   assign n[1] = (c[0] & X) | (~c[2] & X) | (~c[1] & X) | (~c[2] & ~c[1] & c[0]) | (c[2] & c[1] & c[0] & ~X);
   assign n[0] = (c[1] & X) | (c[2] & X) | (~c[2] & ~c[1] & ~X);
   assign Y = (~c[2] & c[0]) | (c[1] & ~c[0] & X) | (c[2] & ~c[1] & X) | (~c[2] & ~c[1] & ~X);
   assign W = (~c[2] & c[1]) | (c[2] & ~c[1]) | (c[2] & ~c[0]);

   always @(posedge clk) begin
      if (~reset)
         c <= 3'b100;
      else
         c <= n;
   end
endmodule
```

Figure 1: Verilog

In figure 1 , it shows the implementing of the equation that we found from the k map.

```
module lab5part1_tb();
   reg X, reset, clk;
   wire [2:0] c, n;  //c : curret n: next state
   wire Y, W;

   lab5part1 uut(.X(X), .reset(reset), .clk(clk), .c(c), .n(n), .Y(Y), .W(W));

   initial begin
      clk = 1'b0;
      reset = 1'b0;
      X = 1'b0;

      #50;
      reset = 1'b1;
      X = 1'b0;

      #50;
      X = 1'b0;

      #50;
      X = 1'b1;

      #50;
      X = 1'b0;
      $stop;
   end

   always begin
      #25;
      clk = ~clk;
   end
endmodule
```

Figure 2: test bench

Now in figure 2, we did a truth table, in here where we set each one to 0 then increment it, as it get plug into the equation we get the wave form in figure 3.

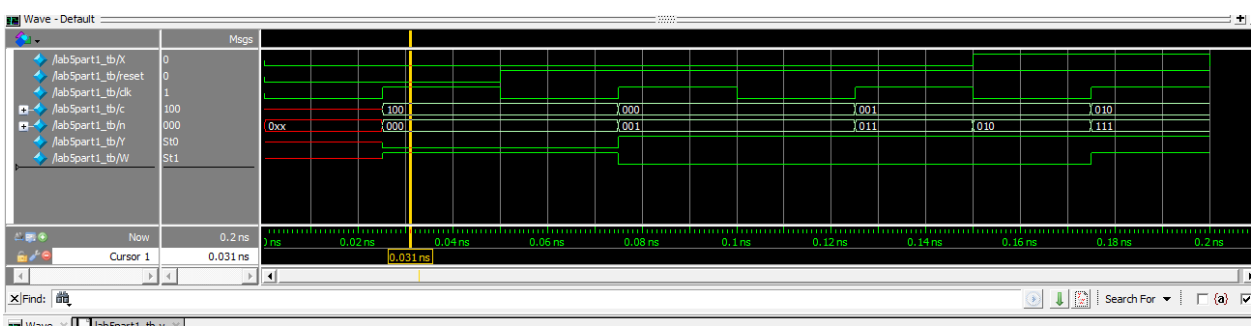


Figure 3: waveform

Part 2:

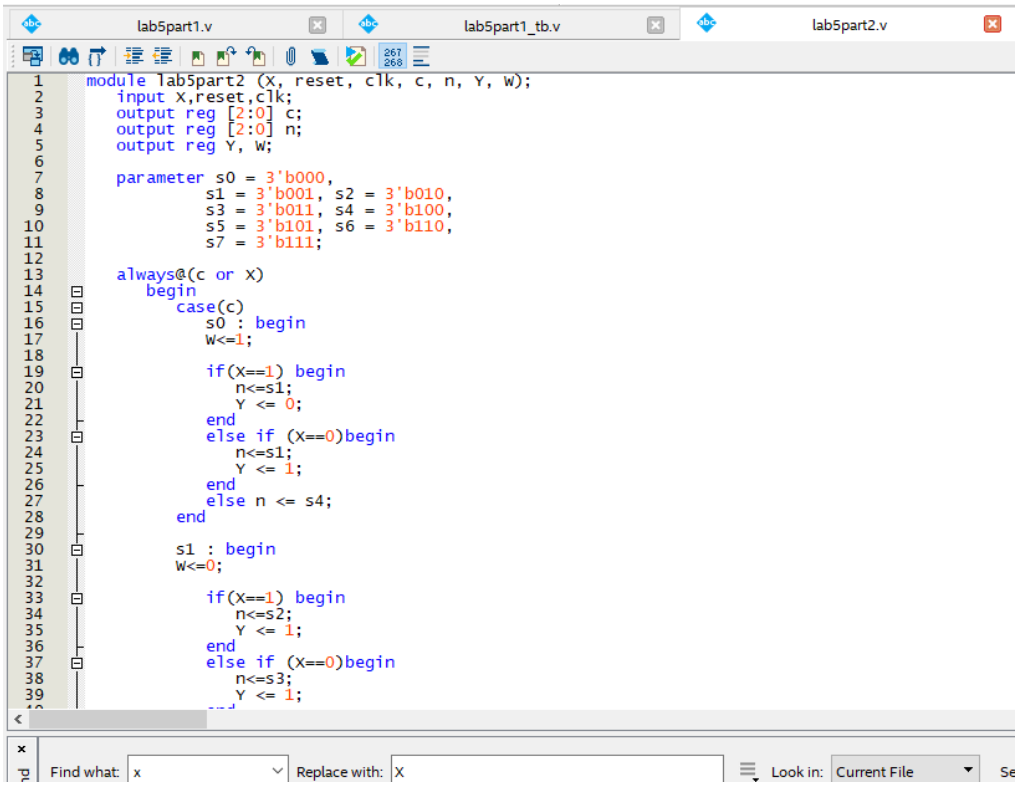
Description:

We are to design the same thing from the first part, but this time implement if else or case statement.

Procedure:

Now you can use the Verilog's "if else" or "case" statement to implement the state machine in Part 1.

Engineering data:




```
module lab5part2 (X, reset, clk, c, n, Y, W);
   input X,reset,clk;
   output reg [2:0] c;
   output reg [2:0] n;
   output reg Y, W;

   parameter s0 = 3'b000,
             s1 = 3'b001, s2 = 3'b010,
             s3 = 3'b011, s4 = 3'b100,
             s5 = 3'b101, s6 = 3'b110,
             s7 = 3'b111;

   always@(c or X)
      begin
         case(c)
            s0 : begin
            W<=1;

            if(X==1) begin
               n<=s1;
               Y <= 0;
            end
            else if (X==0)begin
               n<=s1;
               Y <= 1;
            end
            else n <= s4;
         end

         s1 : begin
         W<=0;

            if(X==1) begin
               n<=s2;
               Y <= 1;
            end
            else if (X==0)begin
               n<=s3;
               Y <= 1;
```

Figure 4: Verilog

In figure 4, it shows the Verilog coding of the state machine in if else statement, which also include cases. We used the same test bench from part 1 which is in figure 5.

```
module lab5part2_tb();
   reg X, reset, clk;
   wire [2:0] c, n;
   wire Y, W;  //Y is mealy    W is moorse

   lab5part2 uut(.X(X), .reset(reset), .clk(clk), .c(c), .n(n), .Y(Y), .W(W));

   initial begin
      clk = 1'b0;
      reset = 1'b0;
      X = 1'b0;

      #50;
      reset = 1'b1;
      X = 1'b0;

      #50;
      X = 1'b0;

      #50;
      X = 1'b1;

      #50;
      X = 1'b0;
      $stop;
   end

   always begin
      #25;
      clk = ~clk;
   end
endmodule
```

Figure 5: testbench

Using figure 5, we acquire the waveform figure 6.

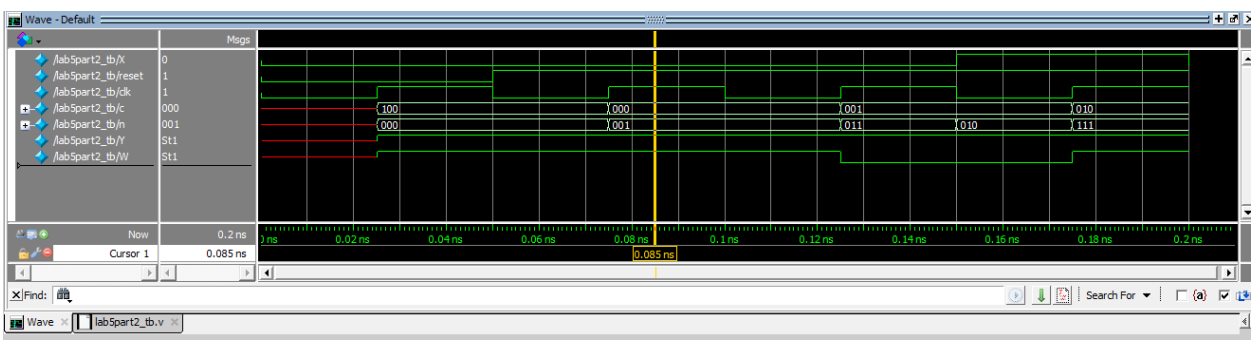


Figure 6:waveform

Part 3:

Description:

We are to implement a Verilog coding that searches for the sequence 011011010, when it is correct then it could exit.

Procedure:

This is a useful state machine design with Mealy and Moore outputs. Design a solution to the following sequence detector. There is one input called A. For example, the sequence A, A, /A, A, /A, would be represented by 1 1 0 1 0. Find this sequence 0 1 1 0 1 1 0 1 0

Engineering data:

NA

Conclusion:

In conclusion this lab wasn't so bad, there was only 3 part, procasination kill us when it was time to do it and turn it in. The last part, which is part 3, was more complicated then expected as we couldn't figure out the test bench that would be good to test the Verilog coding on.