MARIO PALACIOS

LAB COURSE (CpE 64) SECTION #20

WENDESDAY

INSTRUCTOR: EMMANUEL DUPART

PART 1. BOOLEAN LOGIC STATE MACHINE

Description:

Use D Flip-Flops to create a state machine solution for the diagram below (Figure 1). Use K-maps to find equations for each of the inputs.

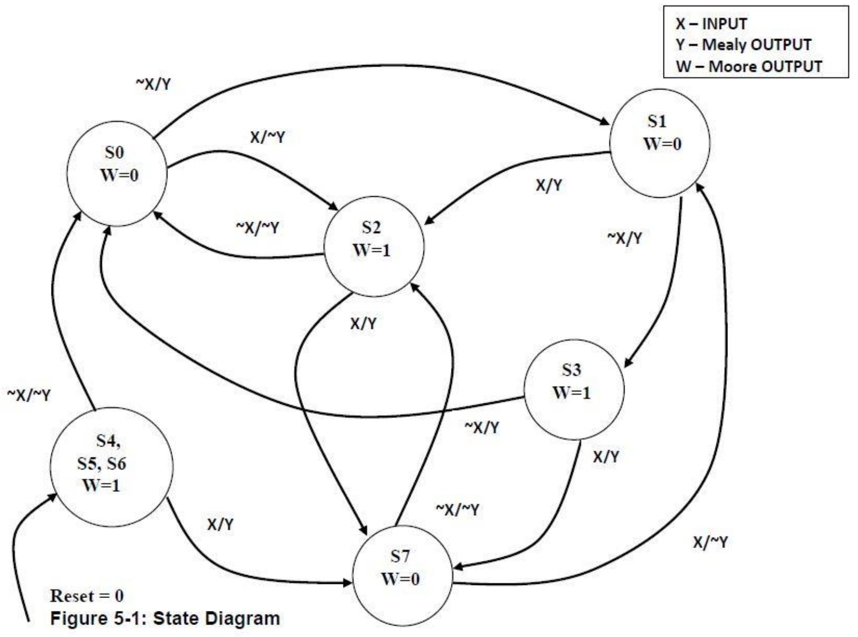
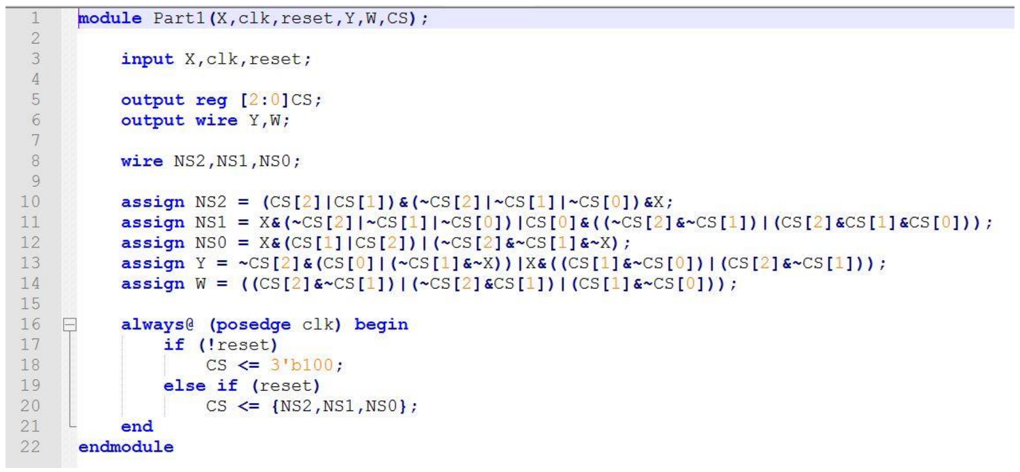


Figure 1. State Machine Diagram

Problem Description:

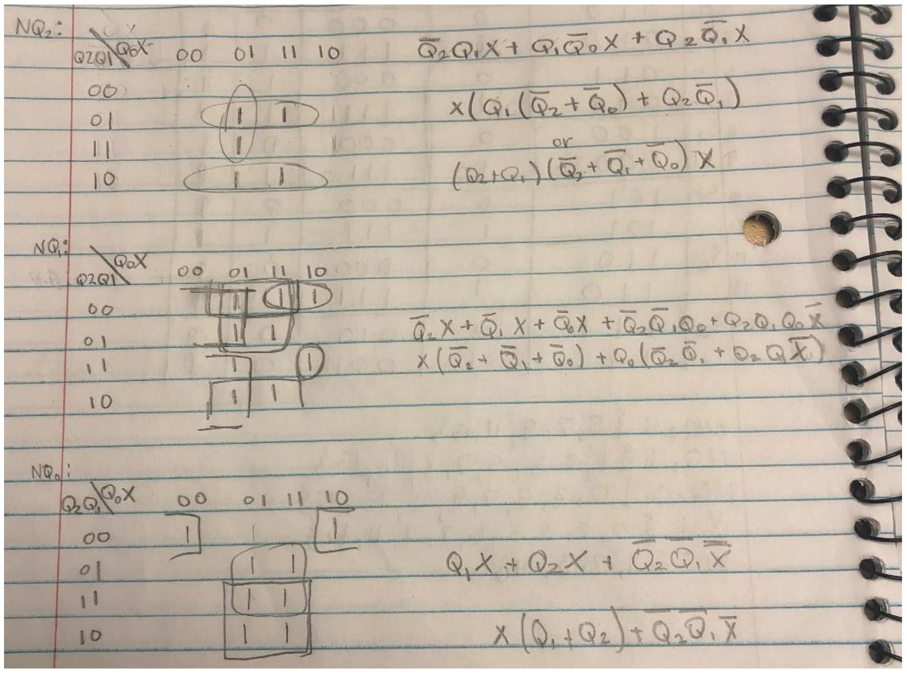
Implement the four equations from the K-Maps into Verilog code, following the Data Flow modeling. Since this type of state machine has an "implied" system clock, a clock must be provide for testing.

Engineering Data:



```
module Part1(X,clk,reset,Y,W,CS);

    input X,clk,reset;

    output reg [2:0]CS;
    output wire Y,W;

    wire NS2,NS1,NS0;

    assign NS2 = (CS[2]|CS[1])&(~CS[2]|~CS[1]|~CS[0])&X;
    assign NS1 = X&(~CS[2]|~CS[1]|~CS[0])|CS[0]&((~CS[2]&~CS[1])|(CS[2]&CS[1]&CS[0]));
    assign NS0 = X&(CS[1]|CS[2])|(~CS[2]&~CS[1]&~X);
    assign Y = ~CS[2]&(CS[0]|(~CS[1]&~X))|X&((CS[1]&~CS[0])|(CS[2]&~CS[1]));
    assign W = ((CS[2]&~CS[1])|(~CS[2]&CS[1])|(CS[1]&~CS[0]));

    always@ (posedge clk) begin
        if (!reset)
            CS <= 3'b100;
        else if (reset)
            CS <= {NS2,NS1,NS0};
    end
endmodule
```

Figure 2. Verilog Code of Equations in Data Flow



NQ_2:

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00				
01		1	1	
11		1		
10		1	1	

$\overline{Q_2}Q_1X + Q_1\overline{Q_0}X + Q_2\overline{Q_1}X$

$X(Q_1(\overline{Q_2}+\overline{Q_0}) + Q_2\overline{Q_1})$

or

$(Q_2+Q_1)(\overline{Q_2}+\overline{Q_1}+\overline{Q_0})X$

NQ_1:

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00		1	1	1
01		1	1	
11		1		1
10		1	1	

$\overline{Q_2}X + \overline{Q_1}X + \overline{Q_0}X + \overline{Q_2}\overline{Q_1}Q_0 + Q_2Q_1Q_0\overline{X}$

$X(\overline{Q_2}+\overline{Q_1}+\overline{Q_0}) + Q_0(\overline{Q_2}\overline{Q_1} + Q_2Q_1\overline{X})$

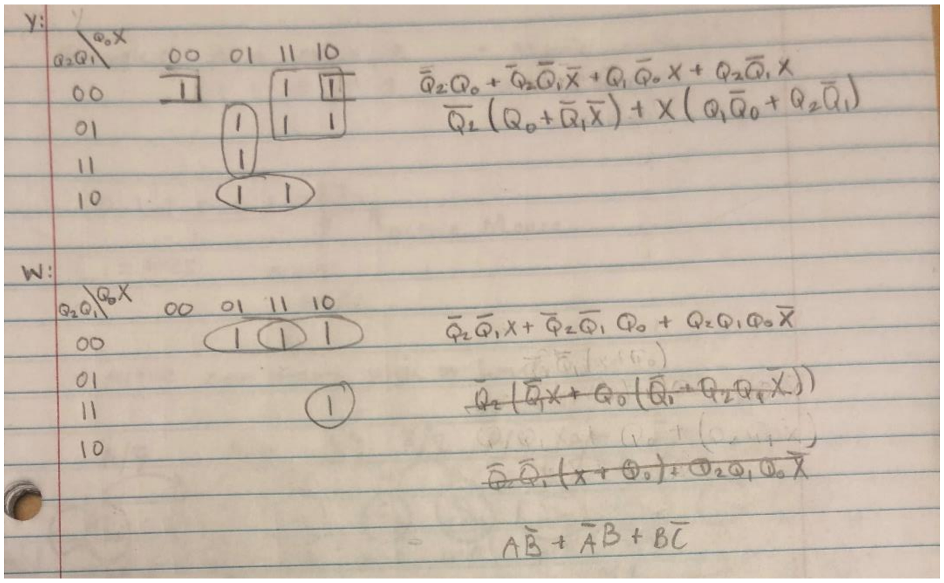
NQ_0:

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00	1			1
01		1	1	
11		1	1	
10		1	1	

$Q_1X + Q_2X + \overline{Q_2}\overline{Q_1}\overline{X}$

$X(Q_1+Q_2) + \overline{Q_2}\overline{Q_1}\overline{X}$

Figure 3. K-Map Displaying How Equations Were Derived



Y:

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00	1		1	1
01		1	1	1
11		1		
10		1	1	

$\overline{Q_2}Q_0 + \overline{Q_2}\overline{Q_1}\overline{X} + Q_1\overline{Q_0}X + Q_2\overline{Q_1}X$

$\overline{Q_2}(Q_0+\overline{Q_1}\overline{X}) + X(Q_1\overline{Q_0}+Q_2\overline{Q_1})$

W:

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00		1	1	1
01				
11				1
10				

$\overline{Q_2}\overline{Q_1}X + \overline{Q_2}\overline{Q_1}Q_0 + Q_2Q_1Q_0\overline{X}$

~~$\overline{Q_2}(\overline{Q_1}X + Q_0(\overline{Q_1}+Q_2Q_1\overline{X}))$~~

~~$\overline{Q_2}\overline{Q_1}(X+Q_0)+Q_2Q_1Q_0\overline{X}$~~

$A\overline{B}+\overline{A}B+B\overline{C}$

Figure 4. K-Map Displaying How Equations Were Derived

When deriving the equations it was important to keep track of the current state and how the input changes the state. There was a total of five equations to implement.

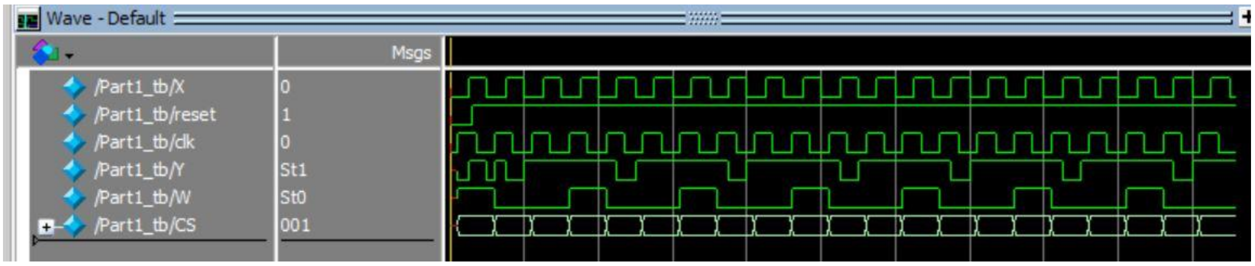


Figure 5. Waveform for State Machine Created Through Equations

Conclusion:

In conclusion to determine how many D Flip-Flops will be needed, depends on the amount of states there are present. In this case there were six states present and two to the power of three is seven; meaning there must be at least three D Flip-Flops in order for this state machine to operate properly. This part of the lab took me about an hour to complete because I knew nothing about State Machines. However at the end I knew a good start is to create a truth table and from there get equations through K-Maps.

PART 2. STATE MACHINE USING BEHAVRIOAL MODELING

Description:

Use Verilog Behavioral Modeling to implement the State Machine in figure 1.

Problem Description:

Behavioral Modeling means the use of "if else" or "case" statements. Afterwards determine the advantage of writing the description using this method over the other.

Engineering Data:

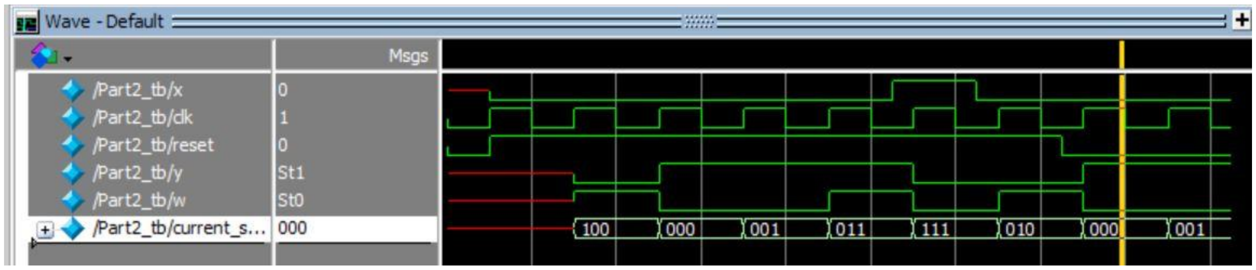


Figure 6. Waveform of State Machine Going Through All Possible States

When writing the Verilog Code States four, five and six all are in the same state, so I decided to remove it from the code itself.

```
module Part2(clk,reset,x,y,w,current_state);

input wire x,reset,clk;
output reg [1:0] y;
output wire w;

output reg [2:0] current_state;
reg [2:0] next_state;

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@(posedge clk) begin
current_state <= next_state;
end

assign w = (current_state == s2) | (current_state == s3)
| (current_state == s4) | (current_state == s5) |
(current_state == s6);

/*Default is state 4 b/c reset = 0 and there is
 *multiple ones so we skipping state 5 & 6  */
        always@(current_state or x) begin
                case(current_state)
                        s0: if(x) begin
                                        next_state <= s2;
                                        y = 0;
                                end
                                else if (!x) begin
                                        next_state <= s1;
                                        y = 1;
                                end
```

Figure 7. Verilog in Behavioral Modeling on Both Text Boxes (Left and Right)

```
s1: if(x) begin
        next_state <= s2;
        y = 1;
        end
        else if (!x) begin
        next_state <= s3;
        y = 1;
        end
s2: if(x) begin
        next_state <= s7;
        y = 1;
        end
        else if (!x) begin
        next_state <= s0;
        y = 0;
        end
s3: if(x) begin
        next_state <= s7;
        y = 1;
        end
        else if (!x) begin
        next_state <= s0;
        y = 1;
        end
s4: if (x) begin
        next_state <= s7;
        y = 1;
        end
        else if (!x) begin
        next_state <= s0;
        y = 0;
        end
s7: if(x) begin
        next_state <= s1;
        y = 0;
        end
        else if (!x) begin
        next_state <= s2;
        y = 0;
        end
default:
        next_state <= s4;
endcase
end
endmodule
```

Conclusion:

In conclusion writing Verilog in Behavioral Modeling is a lot faster because, the amount of time spent on the creating the truth table and deriving the equations using K-maps took longer than using "if else" and case statements. This part of the lab took me about 45 minutes because of getting the inputs to display each state was the most time consuming.

PART 3. SPECIAL STATE MACHINE DESIGN

Description:

Design a solution for a sequence detector. Create a state diagram before starting to write Verilog code.

Problem Description:

The sequence that needs to be replicated is as follows: 0 1 1 0 1 1 0 1 0. The input is called A, where the Moore output is active on the last "0" of the sequence. The Mealy output goes active when the sate machine is in the state before the Moore output goes active.

Engineering Data:

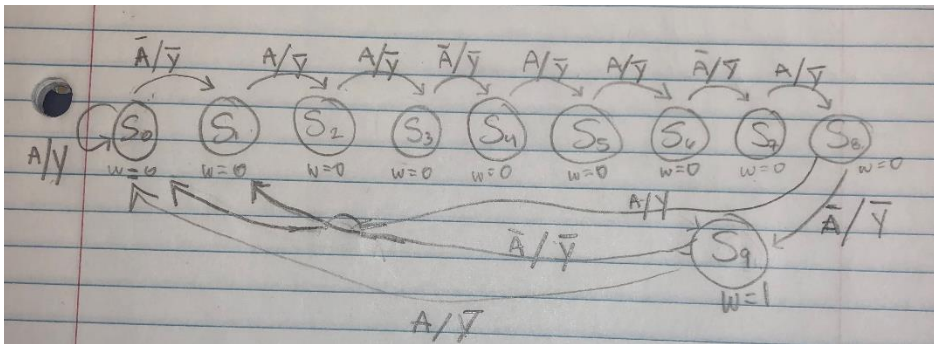


Figure 8. State Machine Diagram for the Sequence

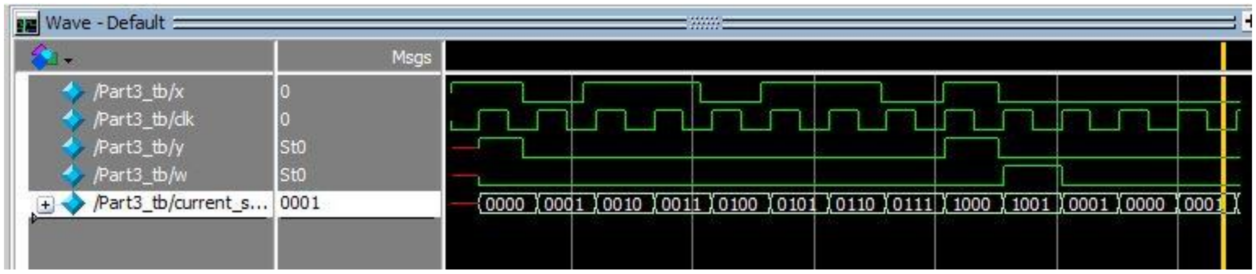


Figure 9. Waveform Diagram of State Machine

```
module Part3(clk,x,y,w,current_state);

 input wire x,clk;
 output reg [1:0] y;
 output wire w;

 output reg [3:0] current_state;
 reg [3:0] next_state;

 parameter s0 = 4'b0000,
                        s1 = 4'b0001,
                        s2 = 4'b0010,
                        s3 = 4'b0011,
                        s4 = 4'b0100,
                        s5 = 4'b0101,
                        s6 = 4'b0110,
                        s7 = 4'b0111,
                        s8 = 4'b1000,
                        s9 = 4'b1001,
                        s10 = 4'b1010,
                        s11 = 4'b1011,
                        s12 = 4'b1100,
                        s13 = 4'b1101,
                        s14 = 4'b1110,
                        s15 = 4'b1111;
 always@(posedge clk) begin
        current_state <= next_state;
 end

 assign w = (current_state == s9)
  always@(*) begin
        case(current_state)
                s0: if(x) begin
                                next_state <= s0;
                                y = 1;
                        end
                        else if (!x) begin
                                next_state <= s1;
                                y = 0;
                        end
```

```
s1: if(x) begin
        next_state <= s2;
        y = 0;
        end
        else if (!x) begin
        next_state <= s0;
        y = 0;
        end
s2: if(x) begin
        next_state <= s3;
        y = 0;
        end
        else if (!x) begin
        next_state <= s0;
        y = 0;
        end
s3: if(x) begin
        next_state <= s0;
        y = 0;
        end
        else if (!x) begin
        next_state <= s4;
        y = 0;
        end
s4: if(x) begin
        next_state <= s5;
        y = 0;
        end
        else if (!x) begin
        next_state <= s0;
        y = 0;
        end
s5: if(x) begin
        next_state <= s6;
        y = 0;
        end
        else if (!x) begin
        next_state <= s0;
        y = 0;
        end
s6: if(x) begin
        next_state <= s0;
        y = 0;
        end
        else if (!x) begin
        next_state <= s7;
        y = 0;
        end
```

```
s7: if(x) begin
        next_state <= s8;
        y = 0;
        end
        else if (!x) begin
        next_state <= s0;
        y = 0;
        end
s8: if(x) begin
        next_state <= s0;
        y = 1;
        end
        else if (!x) begin
        next_state <= s9;
        y = 0;
        end
s9: if(x) begin
        next_state <= s0;
        y = 0;
        end
        else if (!x) begin
        next_state <= s1;
        y = 0;
        end
default:
        next_state <= s0;
endcase
end
endmodule
```

Figure 10. Verilog Code of Sequential Detector in Behavioral Modeling

Conclusion:

In conclusion when creating this I drew inspiration from the previous section because the testbench for it was similar to this part. In the end there were only nine states however there must 15 states in the parameter because we need to use four D Flip-Flops. This part took me about 45 minutes to complete. However at the end I came to understand the meaning of Mealy and Moore outputs and how they play a role in State Machines.