# University of Miami

# EEN 414 Computer Organization and Design

# EXPERIMENT #5. Algorithmic State Machine

**Submitted by:** Brandon Rubio, Isabela Bandrich, Rainier Young, Nikeem Dunkelly-Allen

**On:** 10/28/22

**to :** Dr. Onur Tigli

## Objective

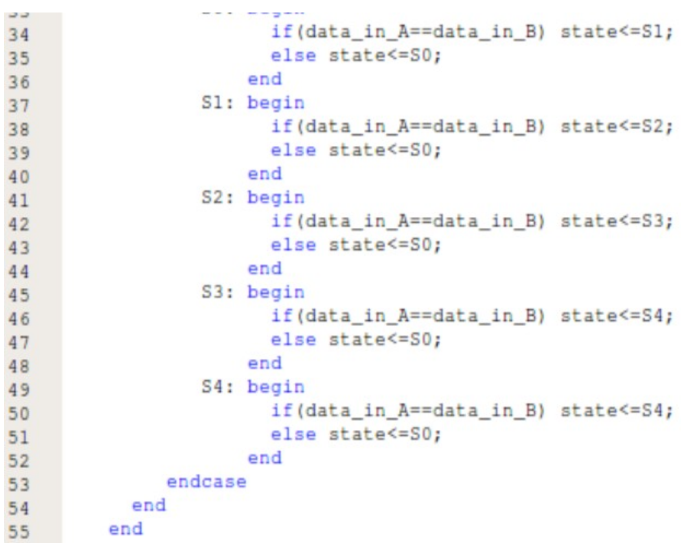
In this experiment, we are required to apply the design and optimization techniques of Algorithmic State Machines into Verilog HDL. This experiment will consist of the design and optimization of a simple finite state machine. We will be working with one and two-state variable machines with different coding styles. All coding, verification, simulations, synthesis, and timing analysis will be carried out using the Xilinx ISE tools. All verified designs will be implemented on the NEXYS-FPGA board to demonstrate the correct operations.

## Verilog HDL and Testbench Code

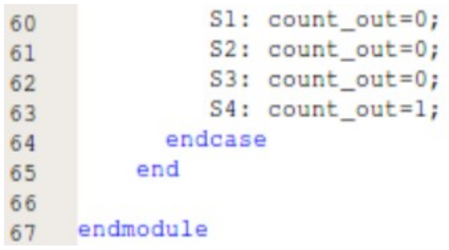
**MAIN CODE:**

**Moore FSM (One State Variable):**

**(Sequential)**

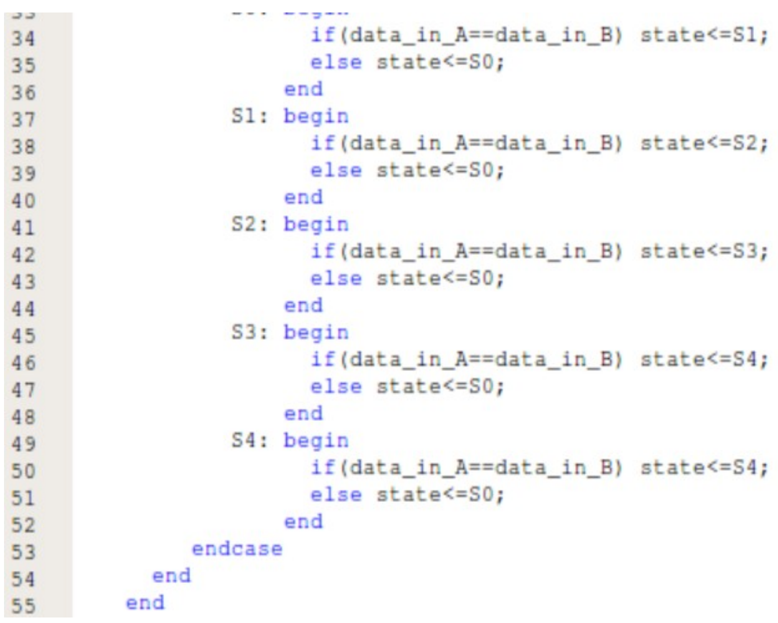


```
                 if(data_in_A==data_in_B) state<=S1;
                 else state<=S0;
               end
           S1: begin
                 if(data_in_A==data_in_B) state<=S2;
                 else state<=S0;
               end
           S2: begin
                 if(data_in_A==data_in_B) state<=S3;
                 else state<=S0;
               end
           S3: begin
                 if(data_in_A==data_in_B) state<=S4;
                 else state<=S0;
               end
           S4: begin
                 if(data_in_A==data_in_B) state<=S4;
                 else state<=S0;
               end
        endcase
     end
   end
```

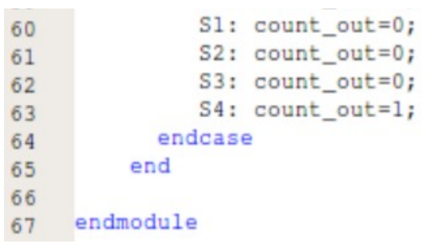


```
         S1: count_out=0;
         S2: count_out=0;
         S3: count_out=0;
         S4: count_out=1;
      endcase
    end

endmodule
```

**(Gray Code)**

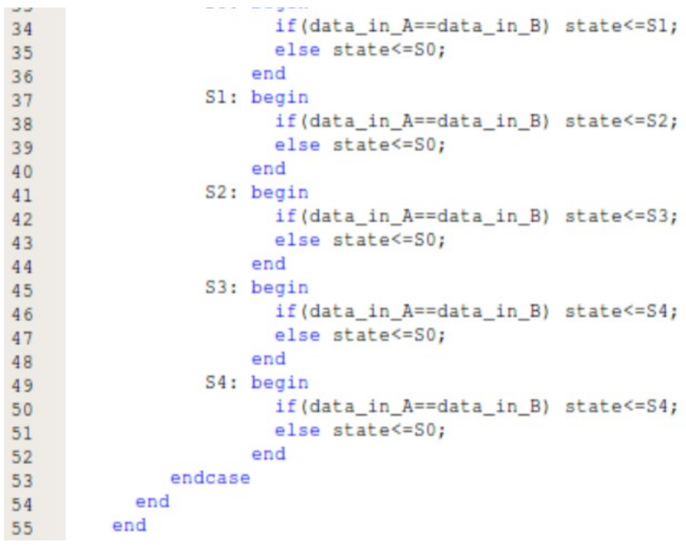


```
                    if(data_in_A==data_in_B) state<=S1;
                    else state<=S0;
                  end
              S1: begin
                    if(data_in_A==data_in_B) state<=S2;
                    else state<=S0;
                  end
              S2: begin
                    if(data_in_A==data_in_B) state<=S3;
                    else state<=S0;
                  end
              S3: begin
                    if(data_in_A==data_in_B) state<=S4;
                    else state<=S0;
                  end
              S4: begin
                    if(data_in_A==data_in_B) state<=S4;
                    else state<=S0;
                  end
           endcase
        end
      end
```

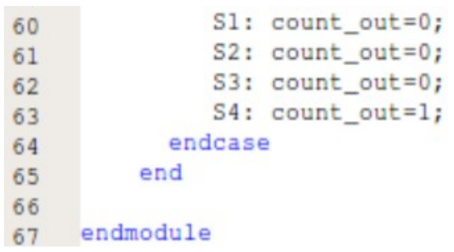


```
          S1: count_out=0;
          S2: count_out=0;
          S3: count_out=0;
          S4: count_out=1;
        endcase
      end

endmodule
```

**(One Hot)**



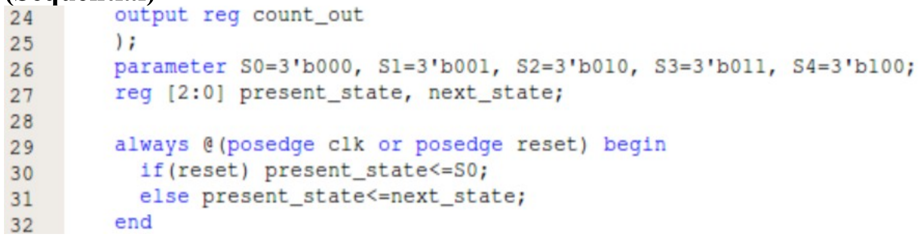
```
              if(data_in_A==data_in_B) state<=S1;
              else state<=S0;
            end
        S1: begin
              if(data_in_A==data_in_B) state<=S2;
              else state<=S0;
            end
        S2: begin
              if(data_in_A==data_in_B) state<=S3;
              else state<=S0;
            end
        S3: begin
              if(data_in_A==data_in_B) state<=S4;
              else state<=S0;
            end
        S4: begin
              if(data_in_A==data_in_B) state<=S4;
              else state<=S0;
            end
     endcase
  end
end
```



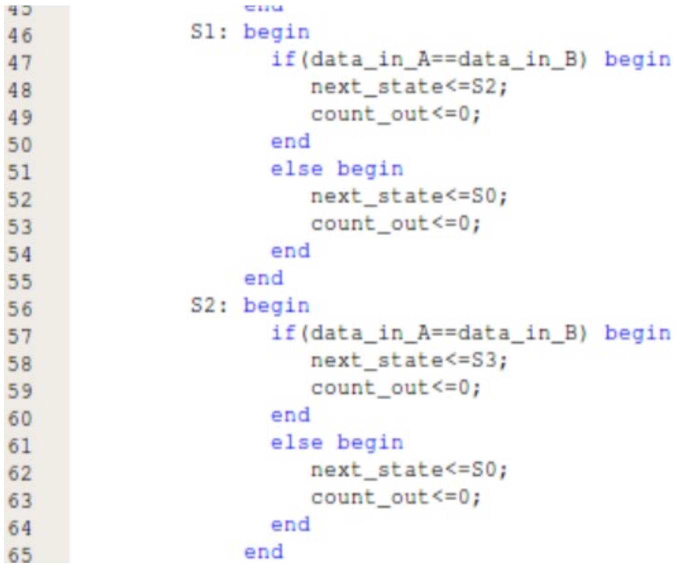
```
         S1: count_out=0;
         S2: count_out=0;
         S3: count_out=0;
         S4: count_out=1;
      endcase
    end

endmodule
```

**Mealy FSM (Two State Variables):**

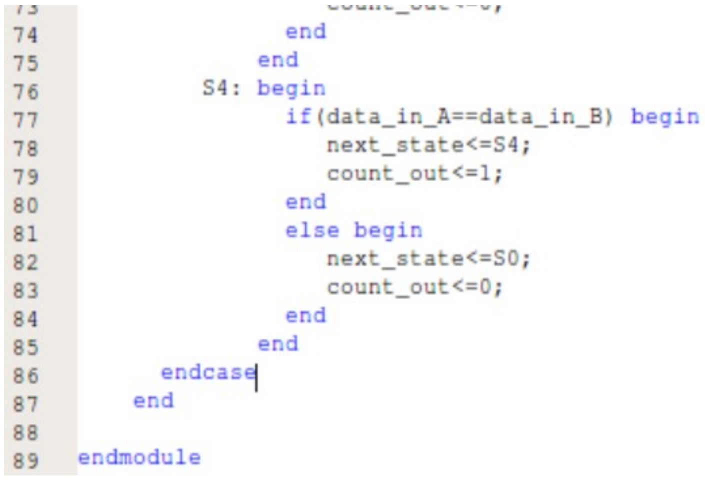
**(Sequential)**



```
output reg count_out
);
parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100;
reg [2:0] present_state, next_state;

always @(posedge clk or posedge reset) begin
  if(reset) present_state<=S0;
  else present_state<=next_state;
end
```

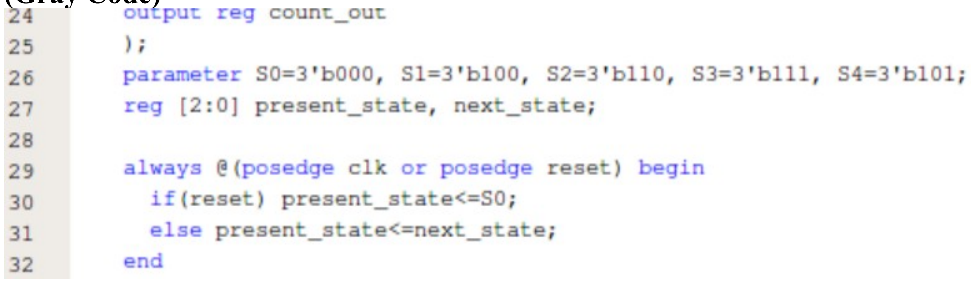


```
         S1: begin
               if(data_in_A==data_in_B) begin
                  next_state<=S2;
                  count_out<=0;
               end
               else begin
                  next_state<=S0;
                  count_out<=0;
               end
             end
         S2: begin
               if(data_in_A==data_in_B) begin
                  next_state<=S3;
                  count_out<=0;
               end
               else begin
                  next_state<=S0;
                  count_out<=0;
               end
             end
```

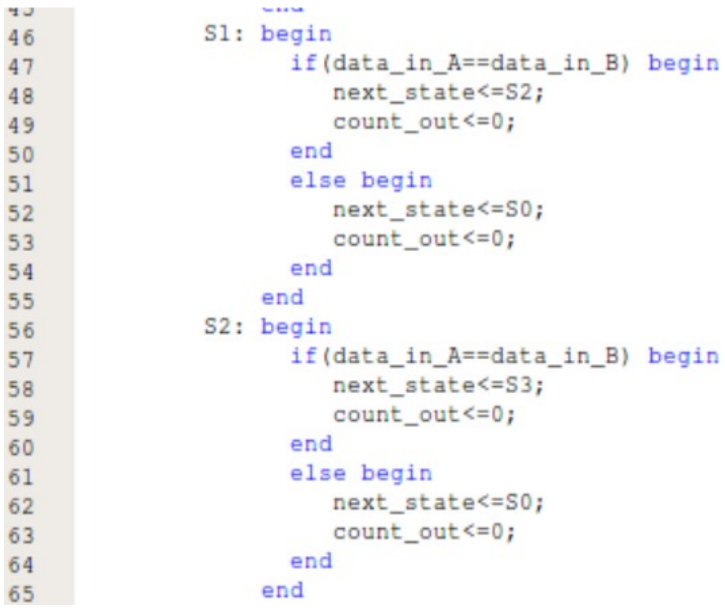


```
               end
             end
         S4: begin
               if(data_in_A==data_in_B) begin
                  next_state<=S4;
                  count_out<=1;
               end
               else begin
                  next_state<=S0;
                  count_out<=0;
               end
             end
      endcase
    end

endmodule
```

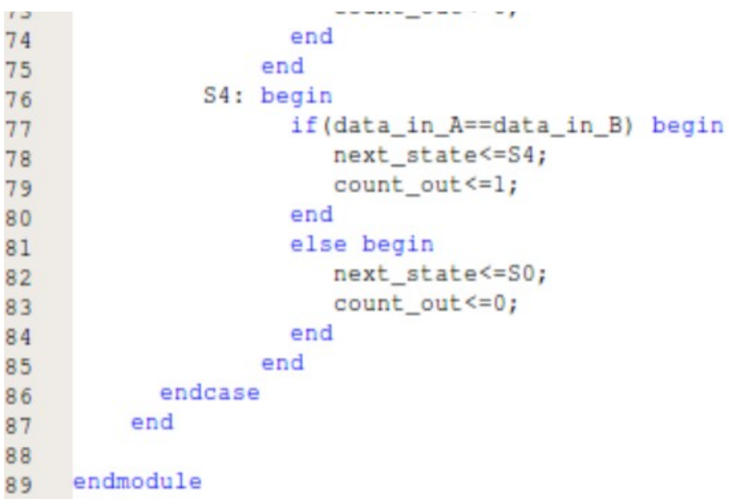
**(Gray Code)**



```
    output reg count_out
    );
    parameter S0=3'b000, S1=3'b100, S2=3'b110, S3=3'b111, S4=3'b101;
    reg [2:0] present_state, next_state;

    always @(posedge clk or posedge reset) begin
      if(reset) present_state<=S0;
      else present_state<=next_state;
    end
```

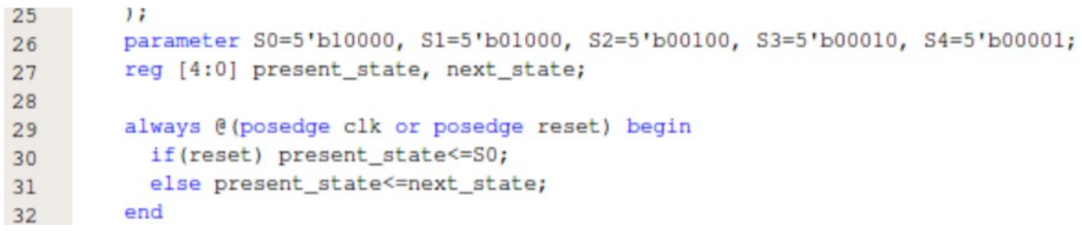


```
         S1: begin
               if(data_in_A==data_in_B) begin
                  next_state<=S2;
                  count_out<=0;
               end
               else begin
                  next_state<=S0;
                  count_out<=0;
               end
             end
         S2: begin
               if(data_in_A==data_in_B) begin
                  next_state<=S3;
                  count_out<=0;
               end
               else begin
                  next_state<=S0;
                  count_out<=0;
               end
             end
```

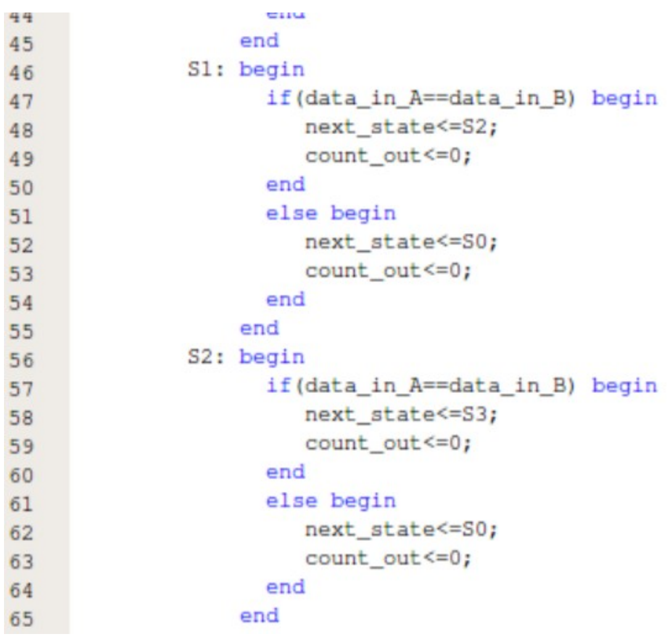


```
               end
             end
         S4: begin
               if(data_in_A==data_in_B) begin
                  next_state<=S4;
                  count_out<=1;
               end
               else begin
                  next_state<=S0;
                  count_out<=0;
               end
             end
      endcase
    end

endmodule
```

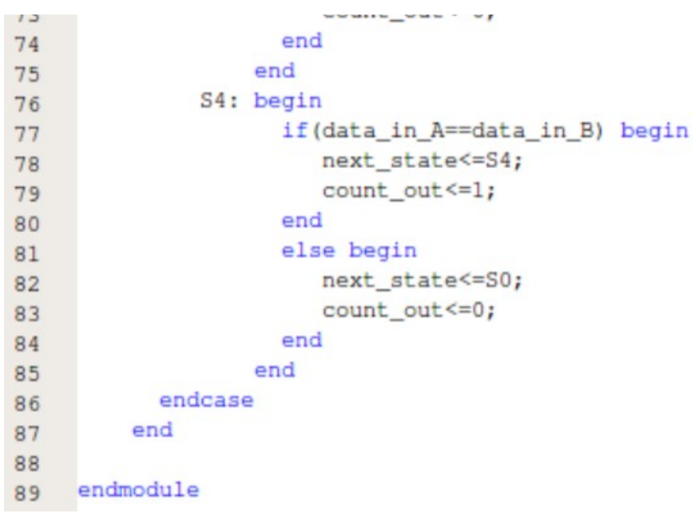
**(One Hot)**



```
    );
    parameter S0=5'b10000, S1=5'b01000, S2=5'b00100, S3=5'b00010, S4=5'b00001;
    reg [4:0] present_state, next_state;

    always @(posedge clk or posedge reset) begin
      if(reset) present_state<=S0;
      else present_state<=next_state;
    end
```



```
                end
            S1: begin
                  if(data_in_A==data_in_B) begin
                     next_state<=S2;
                     count_out<=0;
                  end
                  else begin
                     next_state<=S0;
                     count_out<=0;
                  end
                end
            S2: begin
                  if(data_in_A==data_in_B) begin
                     next_state<=S3;
                     count_out<=0;
                  end
                  else begin
                     next_state<=S0;
                     count_out<=0;
                  end
                end
```



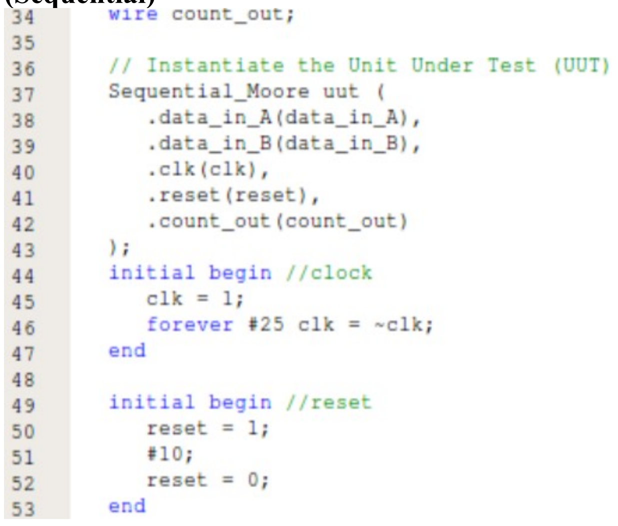
```
                  end
                end
             S4: begin
                  if(data_in_A==data_in_B) begin
                     next_state<=S4;
                     count_out<=1;
                  end
                  else begin
                     next_state<=S0;
                     count_out<=0;
                  end
                end
        endcase
      end

endmodule
```

**TESTBENCH:**

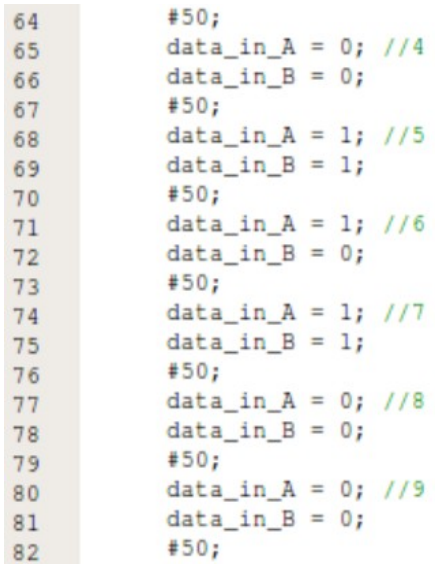
**(note: the testbenches for all codes are very similar except for module names)**

**Moore FSM (One State Variable):**

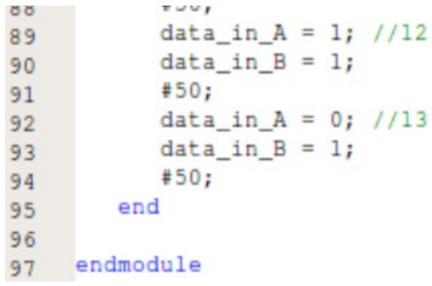
**(Sequential)**



```verilog
wire count_out;

// Instantiate the Unit Under Test (UUT)
Sequential_Moore uut (
   .data_in_A(data_in_A),
   .data_in_B(data_in_B),
   .clk(clk),
   .reset(reset),
   .count_out(count_out)
);
initial begin //clock
   clk = 1;
   forever #25 clk = ~clk;
end

initial begin //reset
   reset = 1;
   #10;
   reset = 0;
end
```

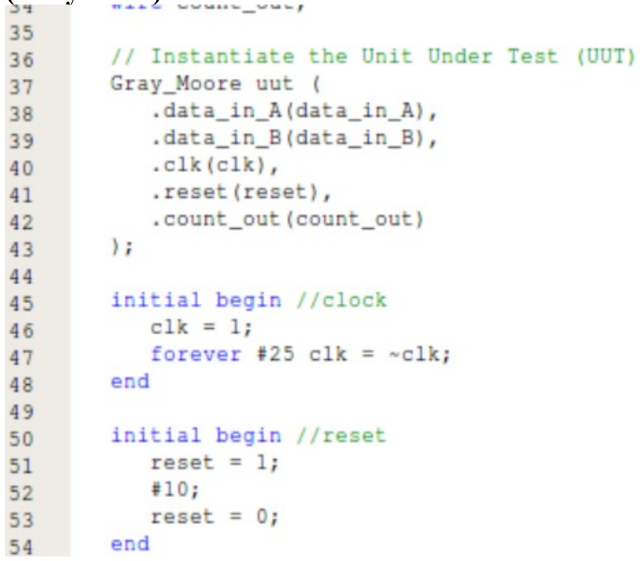


```verilog
      #50;
      data_in_A = 0; //4
      data_in_B = 0;
      #50;
      data_in_A = 1; //5
      data_in_B = 1;
      #50;
      data_in_A = 1; //6
      data_in_B = 0;
      #50;
      data_in_A = 1; //7
      data_in_B = 1;
      #50;
      data_in_A = 0; //8
      data_in_B = 0;
      #50;
      data_in_A = 0; //9
      data_in_B = 0;
      #50;
```

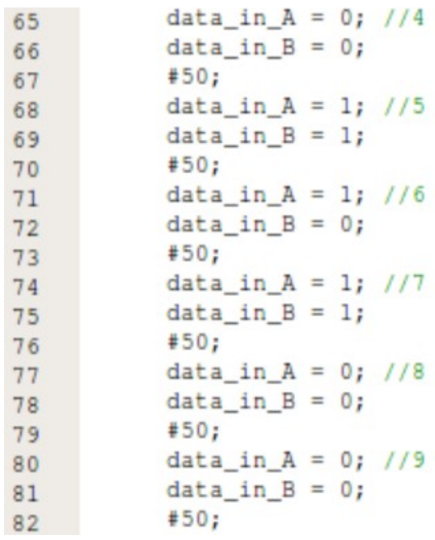


```
      data_in_A = 1; //12
      data_in_B = 1;
      #50;
      data_in_A = 0; //13
      data_in_B = 1;
      #50;
   end

endmodule
```

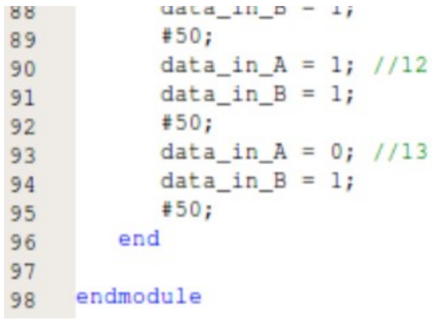
**(Gray Code)**



```

   // Instantiate the Unit Under Test (UUT)
   Gray_Moore uut (
      .data_in_A(data_in_A),
      .data_in_B(data_in_B),
      .clk(clk),
      .reset(reset),
      .count_out(count_out)
   );

   initial begin //clock
      clk = 1;
      forever #25 clk = ~clk;
   end

   initial begin //reset
      reset = 1;
      #10;
      reset = 0;
   end
```

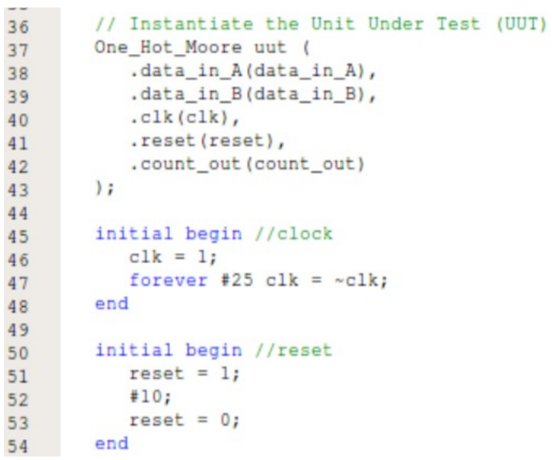


```
        data_in_A = 0; //4
        data_in_B = 0;
        #50;
        data_in_A = 1; //5
        data_in_B = 1;
        #50;
        data_in_A = 1; //6
        data_in_B = 0;
        #50;
        data_in_A = 1; //7
        data_in_B = 1;
        #50;
        data_in_A = 0; //8
        data_in_B = 0;
        #50;
        data_in_A = 0; //9
        data_in_B = 0;
        #50;
```

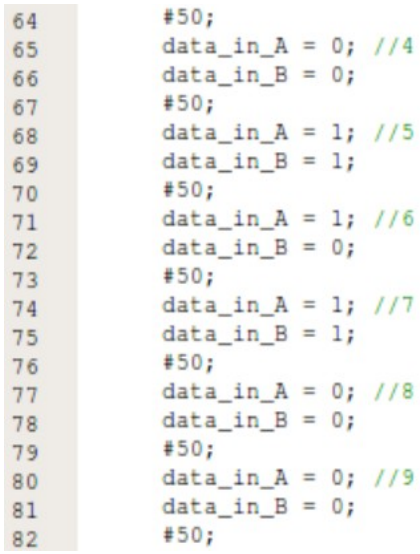


```
        data_in_B = 1;
        #50;
        data_in_A = 1; //12
        data_in_B = 1;
        #50;
        data_in_A = 0; //13
        data_in_B = 1;
        #50;
    end

endmodule
```

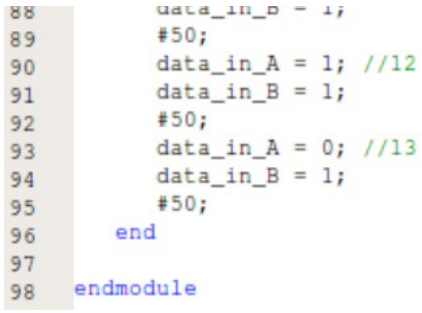
**(One Hot)**



```verilog
   // Instantiate the Unit Under Test (UUT)
   One_Hot_Moore uut (
      .data_in_A(data_in_A),
      .data_in_B(data_in_B),
      .clk(clk),
      .reset(reset),
      .count_out(count_out)
   );

   initial begin //clock
      clk = 1;
      forever #25 clk = ~clk;
   end

   initial begin //reset
      reset = 1;
      #10;
      reset = 0;
   end
```



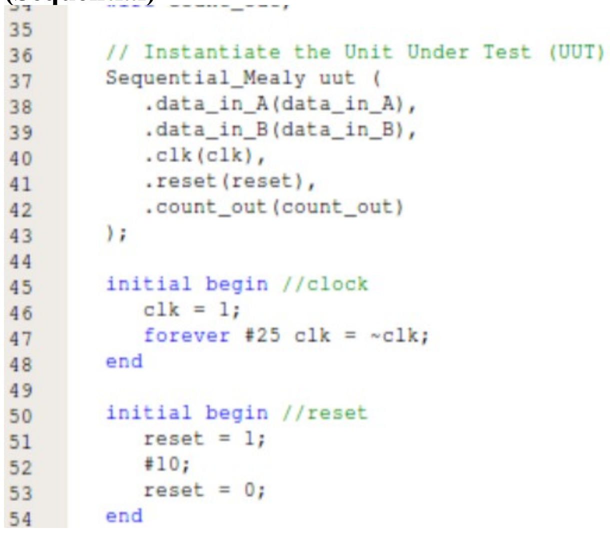
```verilog
        #50;
        data_in_A = 0; //4
        data_in_B = 0;
        #50;
        data_in_A = 1; //5
        data_in_B = 1;
        #50;
        data_in_A = 1; //6
        data_in_B = 0;
        #50;
        data_in_A = 1; //7
        data_in_B = 1;
        #50;
        data_in_A = 0; //8
        data_in_B = 0;
        #50;
        data_in_A = 0; //9
        data_in_B = 0;
        #50;
```



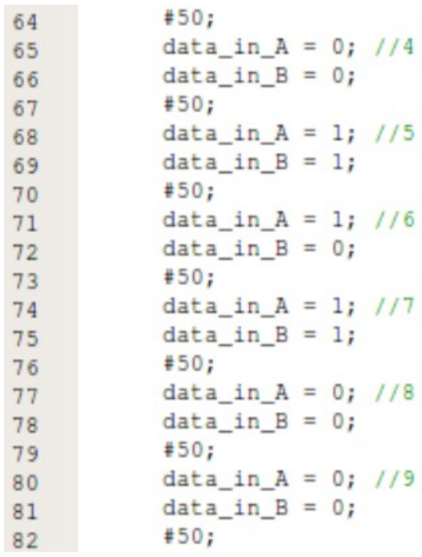
```verilog
        data_in_B = 1;
        #50;
        data_in_A = 1; //12
        data_in_B = 1;
        #50;
        data_in_A = 0; //13
        data_in_B = 1;
        #50;
   end

endmodule
```

## Mealy FSM (Two State Variables):

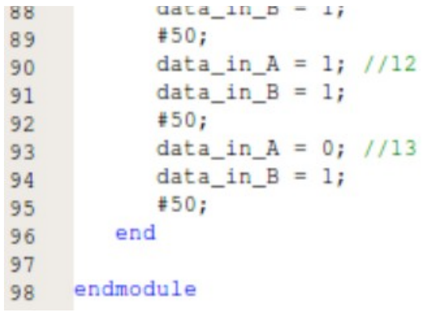
**(Sequential)**



```
   // Instantiate the Unit Under Test (UUT)
   Sequential_Mealy uut (
      .data_in_A(data_in_A),
      .data_in_B(data_in_B),
      .clk(clk),
      .reset(reset),
      .count_out(count_out)
   );

   initial begin //clock
      clk = 1;
      forever #25 clk = ~clk;
   end

   initial begin //reset
      reset = 1;
      #10;
      reset = 0;
   end
```

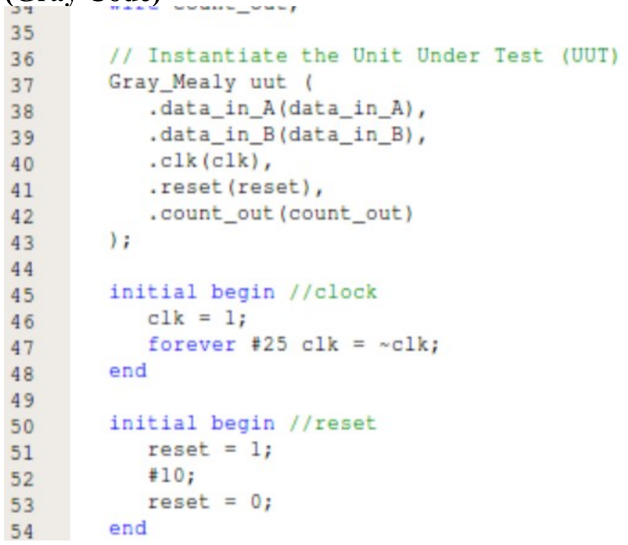


```
      #50;
      data_in_A = 0; //4
      data_in_B = 0;
      #50;
      data_in_A = 1; //5
      data_in_B = 1;
      #50;
      data_in_A = 1; //6
      data_in_B = 0;
      #50;
      data_in_A = 1; //7
      data_in_B = 1;
      #50;
      data_in_A = 0; //8
      data_in_B = 0;
      #50;
      data_in_A = 0; //9
      data_in_B = 0;
      #50;
```

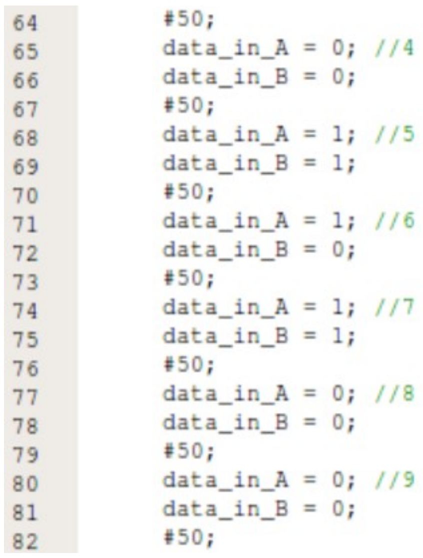


```
      data_in_B = 1;
      #50;
      data_in_A = 1; //12
      data_in_B = 1;
      #50;
      data_in_A = 0; //13
      data_in_B = 1;
      #50;
   end

endmodule
```

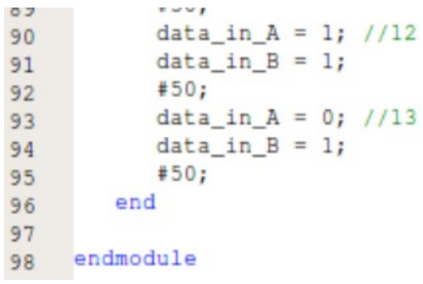
**(Gray Code)**



```verilog

   // Instantiate the Unit Under Test (UUT)
   Gray_Mealy uut (
      .data_in_A(data_in_A),
      .data_in_B(data_in_B),
      .clk(clk),
      .reset(reset),
      .count_out(count_out)
   );

   initial begin //clock
      clk = 1;
      forever #25 clk = ~clk;
   end

   initial begin //reset
      reset = 1;
      #10;
      reset = 0;
   end
```

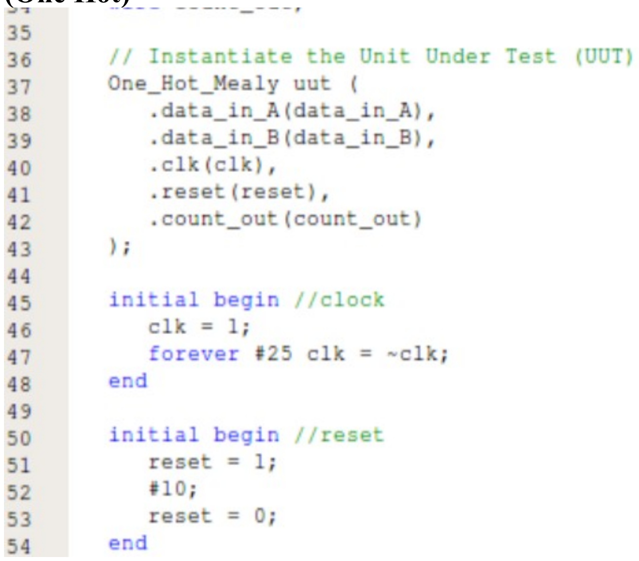


```verilog
      #50;
      data_in_A = 0; //4
      data_in_B = 0;
      #50;
      data_in_A = 1; //5
      data_in_B = 1;
      #50;
      data_in_A = 1; //6
      data_in_B = 0;
      #50;
      data_in_A = 1; //7
      data_in_B = 1;
      #50;
      data_in_A = 0; //8
      data_in_B = 0;
      #50;
      data_in_A = 0; //9
      data_in_B = 0;
      #50;
```

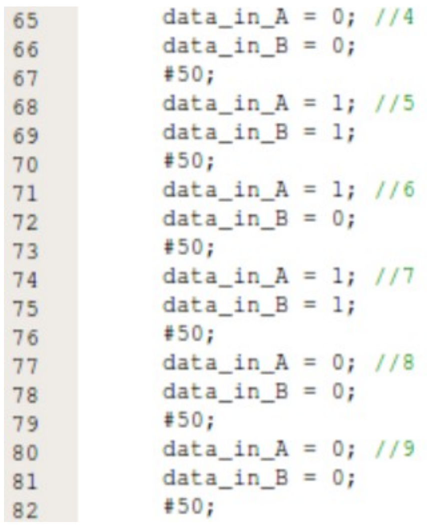


```
      data_in_A = 1; //12
      data_in_B = 1;
      #50;
      data_in_A = 0; //13
      data_in_B = 1;
      #50;
   end

endmodule
```

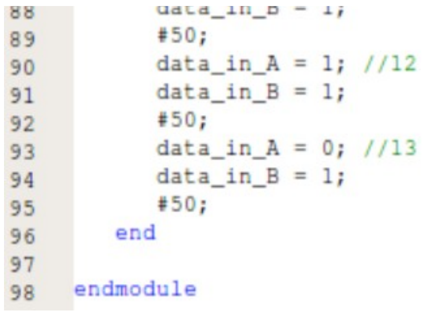
**(One Hot)**



```

   // Instantiate the Unit Under Test (UUT)
   One_Hot_Mealy uut (
      .data_in_A(data_in_A),
      .data_in_B(data_in_B),
      .clk(clk),
      .reset(reset),
      .count_out(count_out)
   );

   initial begin //clock
      clk = 1;
      forever #25 clk = ~clk;
   end

   initial begin //reset
      reset = 1;
      #10;
      reset = 0;
   end
```



```
        data_in_A = 0; //4
        data_in_B = 0;
        #50;
        data_in_A = 1; //5
        data_in_B = 1;
        #50;
        data_in_A = 1; //6
        data_in_B = 0;
        #50;
        data_in_A = 1; //7
        data_in_B = 1;
        #50;
        data_in_A = 0; //8
        data_in_B = 0;
        #50;
        data_in_A = 0; //9
        data_in_B = 0;
        #50;
```

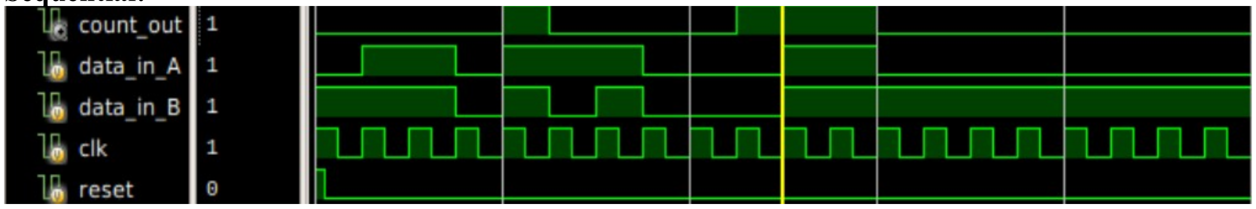


```
        data_in_B = 1;
        #50;
        data_in_A = 1; //12
        data_in_B = 1;
        #50;
        data_in_A = 0; //13
        data_in_B = 1;
        #50;
    end

endmodule
```

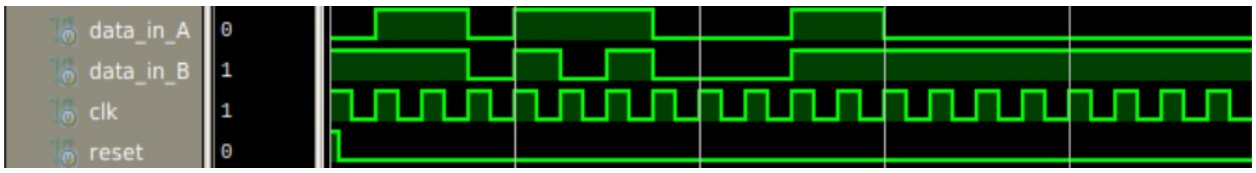
## Simulation Results

### MOORE:

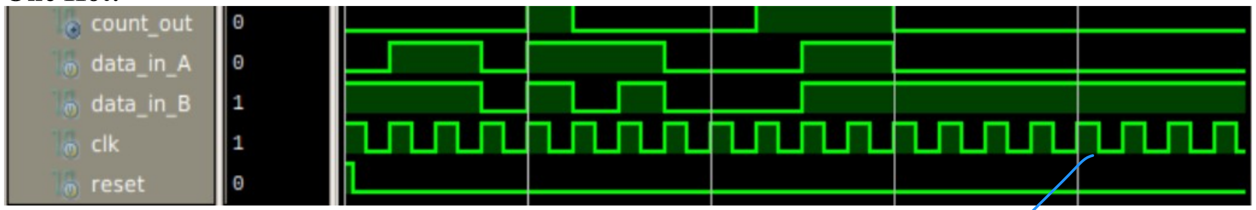
### Sequential:



### Gray Code:



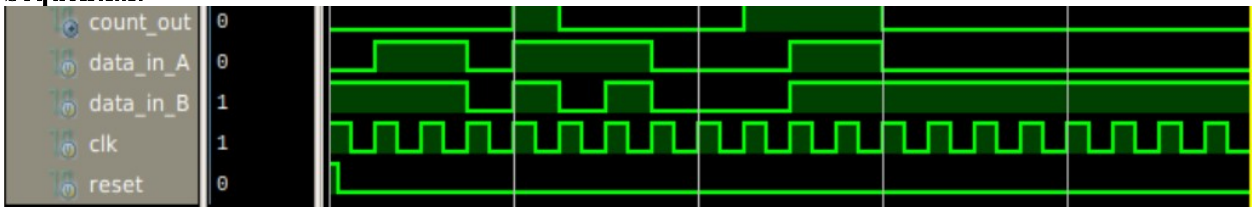
**One Hot:**



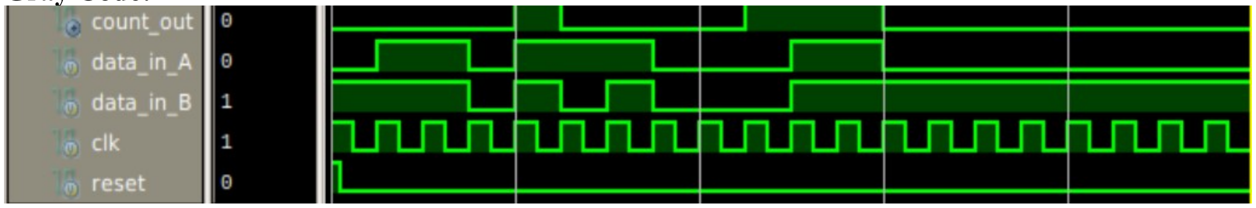
✓

**MEALY:**

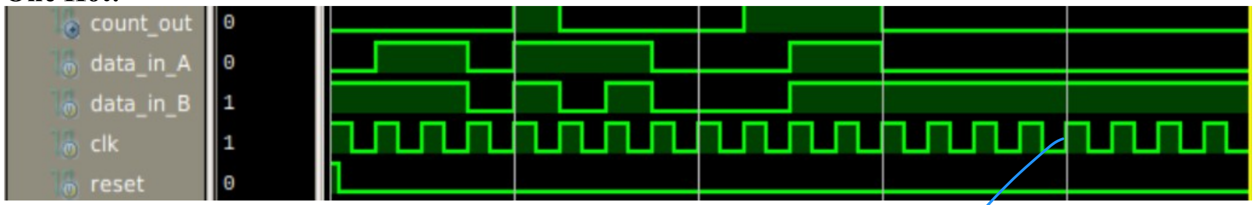
**Sequential:**



**Gray Code:**



**One Hot:**



✓

## Comparison Tables

**MOORE:**

**Sequential:**

| Best Path Delay | Worst Path Delay | Area | Power |
|---|---|---|---|
| 1.119 ns | 1.119 ns | 35 | 0.082 W |

**Gray Code:**

| Best Path Delay | Worst Path Delay | Area | Power |
|---|---|---|---|
| 1.086 ns | 1.086 ns | 35 | 0.082 W |

**One Hot:**

| Best Path Delay | Worst Path Delay | Area | Power |
|---|---|---|---|
| 1.247 ns | 1.247 ns | 66 | 0.082 W |

✓

**MEALY:**

**Sequential:**

| Best Path Delay | Worst Path Delay | Area | Power |
|---|---|---|---|
| 2.896 ns | 2.896 ns | 53 | 0.082 W |

**Gray Code:**

| Best Path Delay | Worst Path Delay | Area | Power |
|---|---|---|---|
| 2.896 ns | 2.896 ns | 57 | 0.082 W |

**One Hot:**

| Best Path Delay | Worst Path Delay | Area | Power |
|---|---|---|---|
| 2.892 ns | 2.892 ns | 82 | 0.082 W |

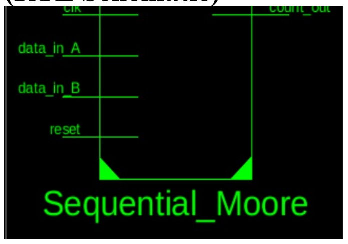
✓

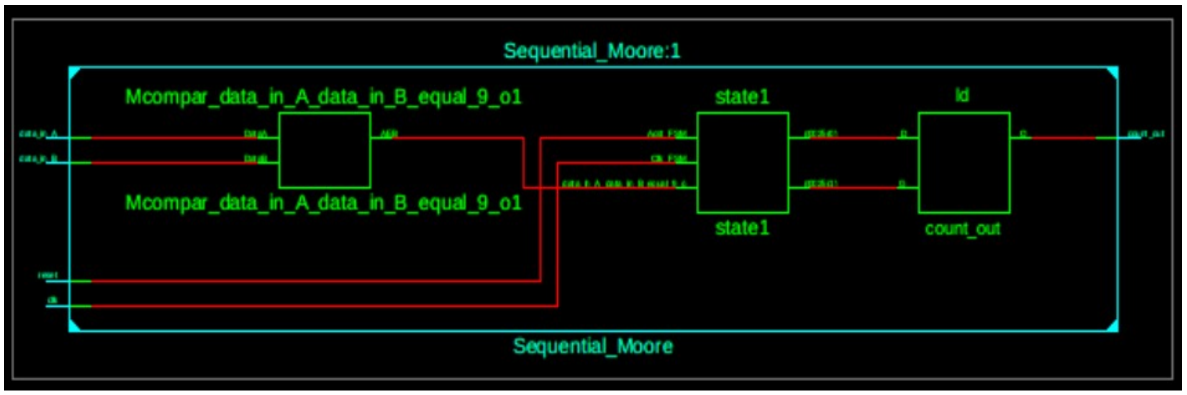
**<u>RTL Schematic and Technology Schematic</u>**

**MOORE:**

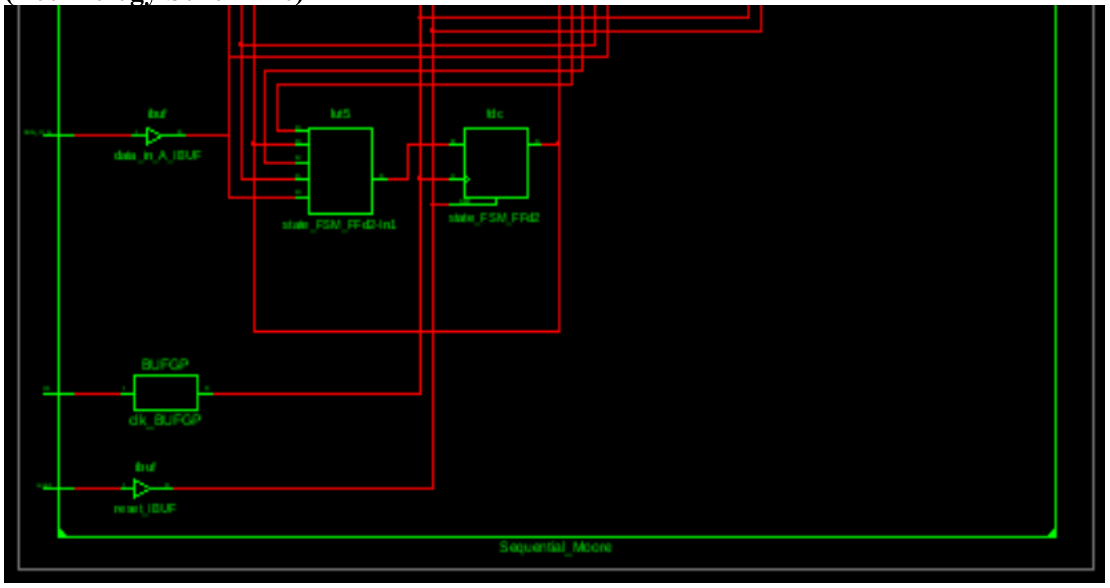
**Sequential:**

(RTL Schematic)



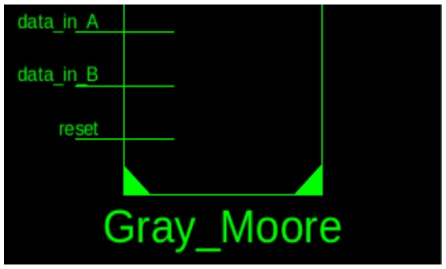

Sequential_Moore:1
Mcompar_data_in_A_data_in_B_equal_9_o1
state1
ld
Mcompar_data_in_A_data_in_B_equal_9_o1
state1
count_out
Sequential_Moore

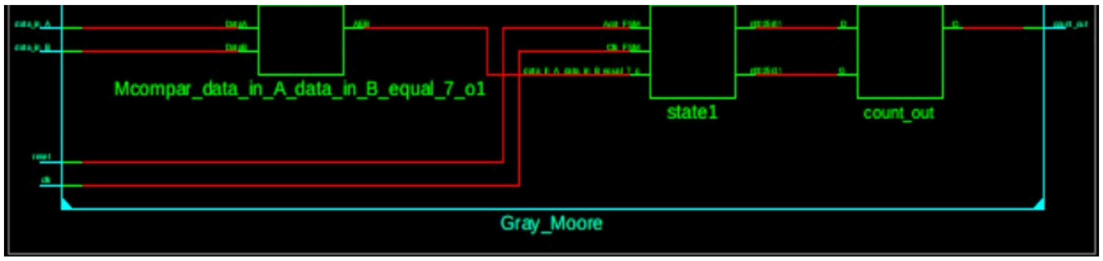
(Technology Schematic)



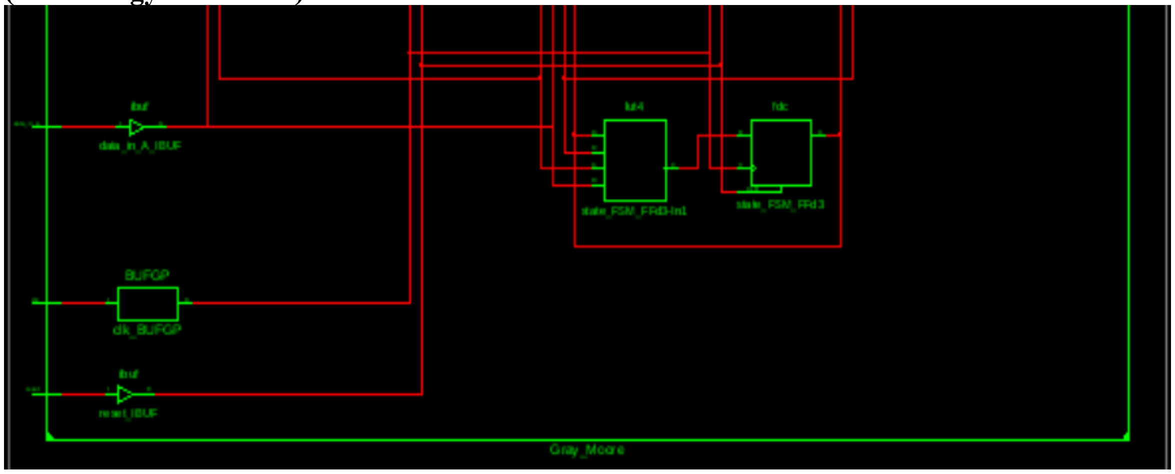
**Gray Code:**

**(RTL Schematic)**



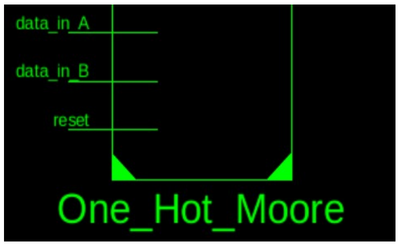


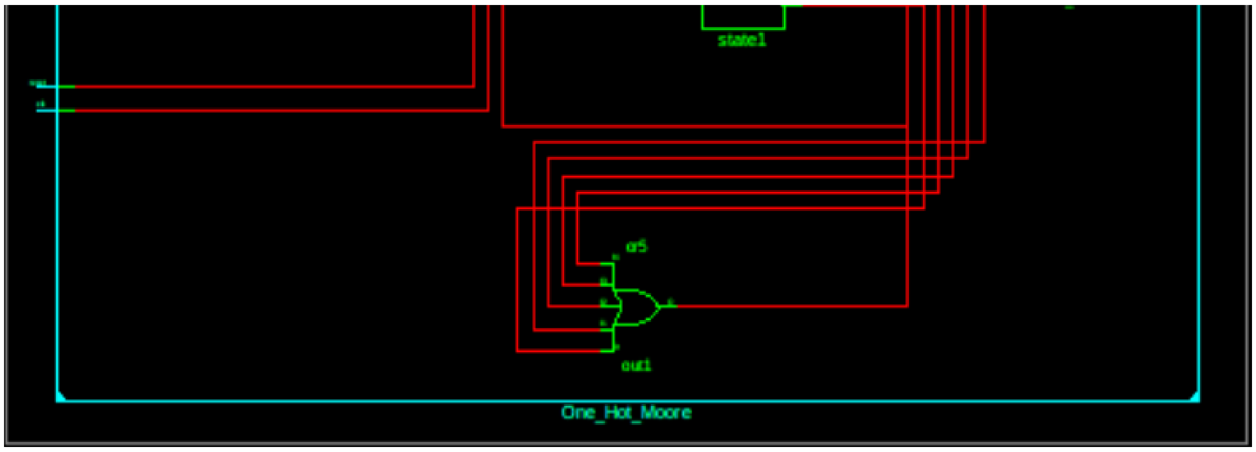
**(Technology Schematic)**



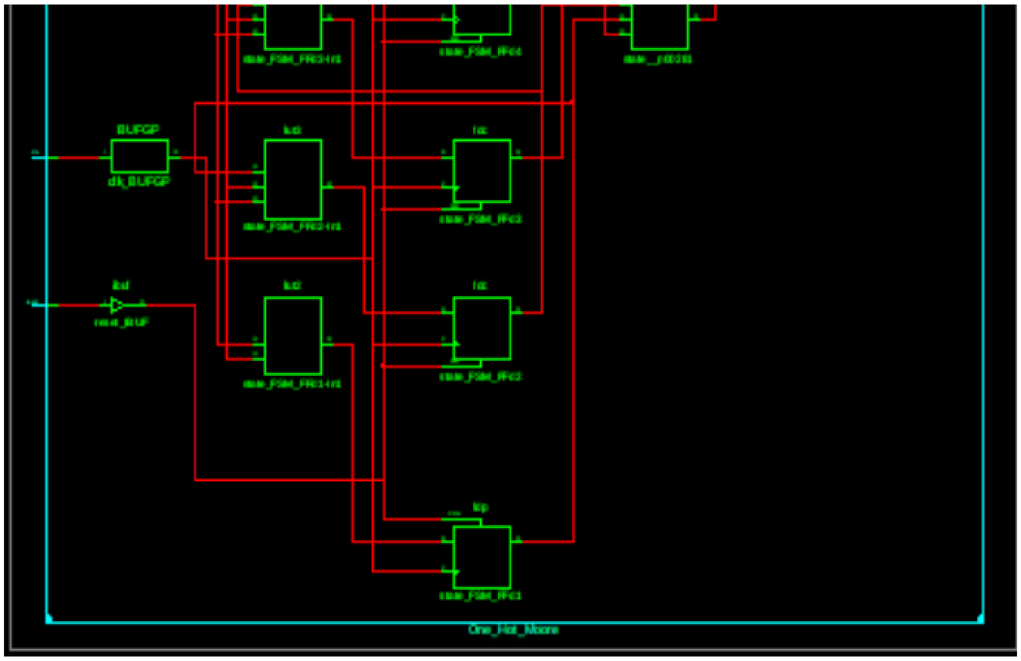
**One Hot:**

**(RTL Schematic)**





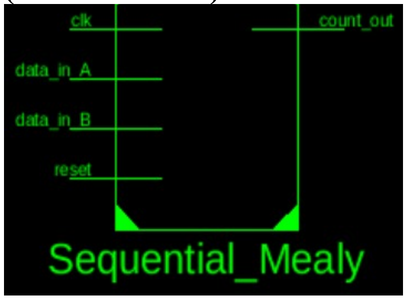
**(Technology Schematic)**

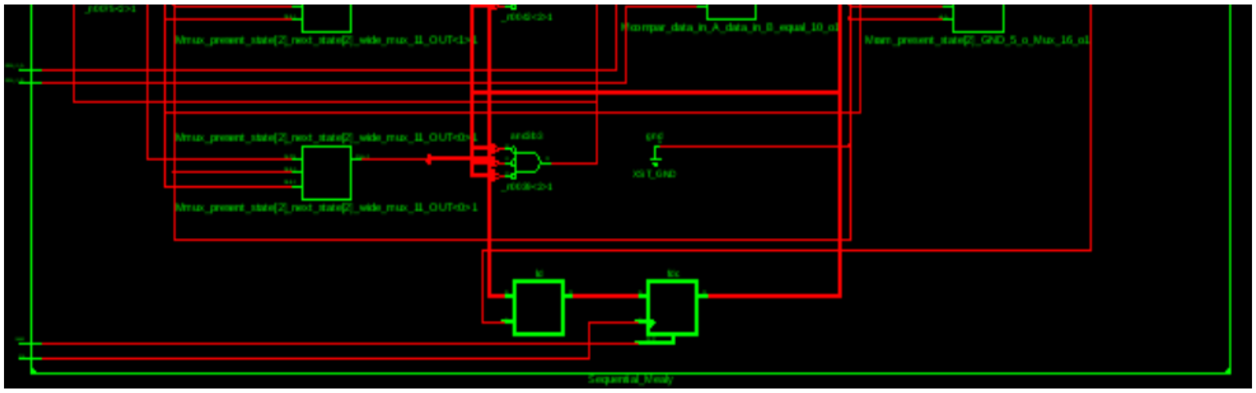


**MEALY:**

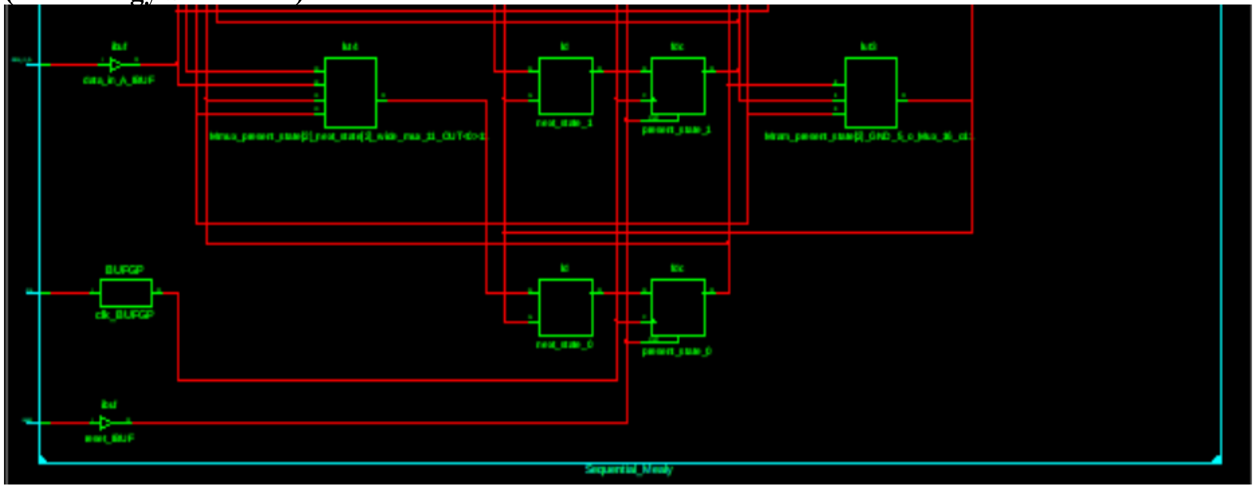
**Sequential:**

**(RTL Schematic)**



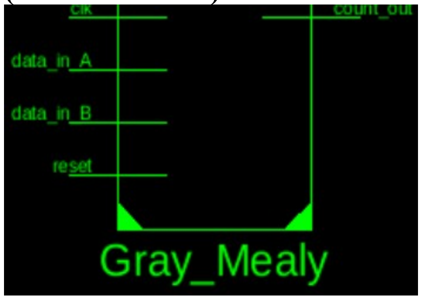


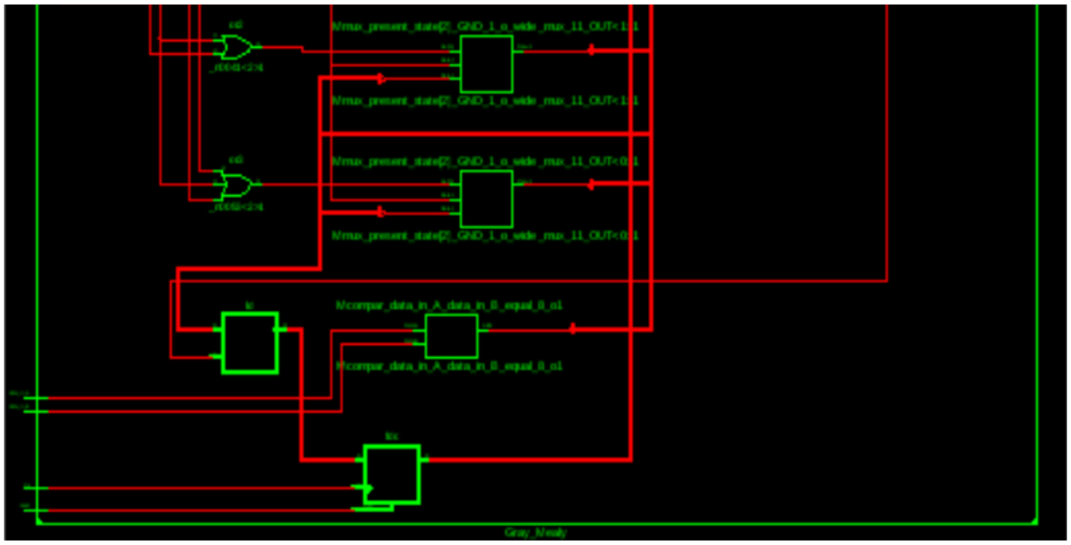
(Technology Schematic)



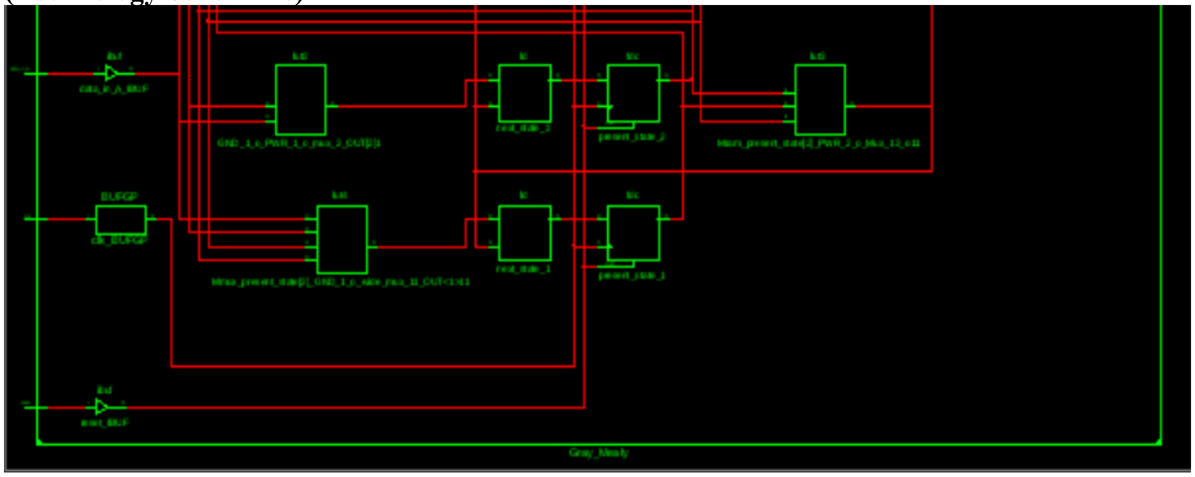
**Gray Code:**

(RTL Schematic)



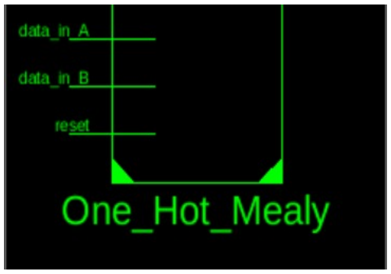


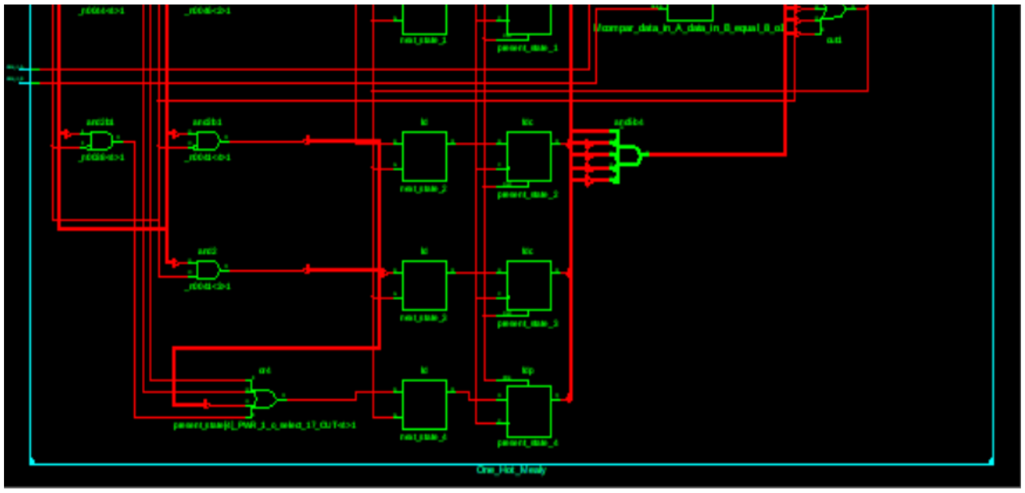
**(Technology Schematic)**



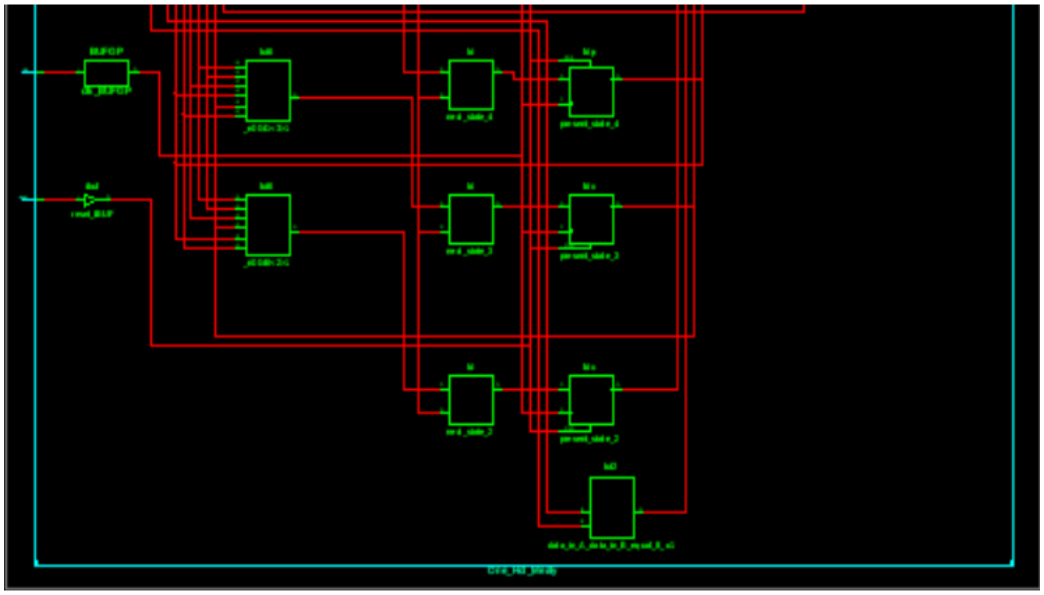
**One Hot:**

**(RTL Schematic)**





**(Technology Schematic)**



**Worst and Best Approach**

When looking at both RTL and technology schematics for the state machines, The coding styles for Moore (one state variable) seem to be simpler than those used for Mealy (two state variables). As for the coding styles themselves, the one-hot coding style seems to have the most complex design. Based on our findings, the best approach for algorithmic state machines is the Moore state machine in the gray code style. This is because it has the best path delay of all other coding styles. The worst approach is the Mealy state machine in the gray code style. It is tied with the Mealy sequential style in best path delay but it takes up more area. The Mealy hot-one style is the fastest of all the Mealy styles, but it uses the most area out of all the state machines.

**Conclusions**

In conclusion, this lab allowed us to better understand the design patterns and applications of Mealy and Moore algorithmic state machines. Using Xilinx ISE tools on the NEXYS-4 FPGA board we were able to implement three different state coding styles for each ASM design. The styles consisted of the sequential, gray code, and one-hot approaches. Through our testing we were able to find that gray code resulted in the fastest Moore ASM and one-hot resulted in the fastest Mealy ASM.

**References**

[1]: Nexys4™ FPGA Board Reference Manual, Nexys-4 rev. B; Revised November 19, 2013 by Diligent Beyond Theory

[2]: Introduction to FPGA ECE414 - Fall 2020 pdf by Randil Gajasinghe and Prof. Onur Tigli

[3]: ECE 414 Computer Organization and Design - Experiment 2. Adder/Subtractor with 7-Segment Display

[4]: Brown, S., Vranesic, Z., “Fundamentals of Digital Logic with VHDL Design,” McGraw Hill, 2000.

# ECE414 Computer Organization and Design Lab
# Responsibility and Demonstration Sheet

| | |
|---|---|
| Lab # | 5 |
| Lab Description | Algorithmic State Machine |

Lab Responsibility Assignment

| Student Name | Responsibility | Signature |
|---|---|---|
| Nikeem Dunkelly-Allen | One-hot mealy testbench | [illegible] |
| Brandon Rubio | Gray-moore testbench | [illegible] |
| Rainier Young | synthesis Pin-Mapping | [illegible] |
| Isabela Bandrich | State Machine Code Report | [illegible] |
| | | |

Lab Demonstration

| Teaching Assistant Signature | Date |
|---|---|
| [illegible] | 7-11-22 |

Comments:
Demonstration shown on time.
ASM working.