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;
34
                    else state<=S0;
35
36
                  end
               S1: begin
37
                    if(data_in_A==data_in_B) state<=S2;
38
                    else state<=S0;
39
                   end
40
              S2: begin
41
                    if(data_in_A==data_in_B) state<=S3;
42
                    else state<=S0;
43
44
                   end
45
               S3: begin
                     if(data_in_A==data_in_B) state<=S4;
46
                    else state<=S0;
47
48
               S4: begin
49
                    if(data_in_A==data_in_B) state<=S4;
50
                    else state<=S0;
51
                  end
52
53
54
            endcase
         end
55
        end
```

```
60 S1: count_out=0;
61 S2: count_out=0;
62 S3: count_out=0;
63 S4: count_out=1;
64 endcase
65 end
66
67 endmodule
```

(Gray Code)

```
34
                     if(data_in_A==data_in_B) state<=S1;</pre>
                     else state<=S0;
35
36
                    end
                S1: begin
37
                     if(data_in_A==data_in_B) state<=S2;</pre>
38
                     else state<=S0;
39
                    end
40
               S2: begin
41
                     if(data_in_A==data_in_B) state<=S3;
42
                     else state<=S0;
43
44
                    end
               S3: begin
45
                     if(data_in_A==data_in_B) state<=S4;
46
                     else state<=S0;
47
                   end
48
                S4: begin
49
                     if(data_in_A==data_in_B) state<=S4;
50
                     else state<=S0;
51
                   end
52
53
             endcase
54
         end
55
       end
```

```
60 S1: count_out=0;
61 S2: count_out=0;
62 S3: count_out=0;
63 S4: count_out=1;
64 endcase
65 end
66
67 endmodule
```

(One Hot)

```
34
                      if(data_in_A==data_in_B) state<=S1;</pre>
                      else state<=S0;
35
36
                    end
                 S1: begin
37
                      if(data_in_A==data_in_B) state<=S2;</pre>
38
                      else state<=S0;
39
40
                S2: begin
41
                      if(data_in_A==data_in_B) state<=S3;
42
                      else state<=S0;
43
44
                     end
                 S3: begin
45
                      if(data_in_A==data_in_B) state<=S4;</pre>
46
                      else state<=S0;
47
48
                 S4: begin
49
                      if(data_in_A==data_in_B) state<=S4;
50
                      else state<=S0;
51
                    end
52
             endcase
53
          end
54
55
         end
```

```
60 S1: count_out=0;
61 S2: count_out=0;
62 S3: count_out=0;
63 S4: count_out=1;
64 endcase
65 end
66
67 endmodule
```

Mealy FSM (Two State Variables):

(Sequential)

```
output reg count_out
24
25
26
        parameter S0=3'b000, S1=3'b001, S2=3'b010, S3=3'b011, S4=3'b100;
        reg [2:0] present_state, next_state;
27
28
        always @(posedge clk or posedge reset) begin
29
          if(reset) present_state<=S0;
30
          else present_state<=next_state;
31
32
        end
```

```
40
                  enu
46
              S1: begin
47
                    if (data_in_A == data_in_B) begin
48
                       next_state<=S2;
                       count_out <= 0;
49
                    end
50
                    else begin
51
                      next_state<=S0;
52
                       count_out <= 0;
53
54
55
                  end
              S2: begin
56
                    if(data_in_A == data_in_B) begin
57
                       next_state<=S3;
58
                       count_out <= 0;
59
                    end
60
                    else begin
61
                      next_state<=S0;
62
                      count_out <= 0;
63
64
                 end
65
```

```
count_out -- o,
13
                    end
74
75
                  end
              S4: begin
76
                    if(data_in_A==data_in_B) begin
77
78
                       next_state<=S4;
                       count_out <= 1;
79
                    end
80
                    else begin
81
                       next_state<=S0;
82
                       count_out <= 0;
83
                    end
84
85
                  end
          endcase
86
         end
87
88
89 endmodule
```

```
(Gray Code) output reg count_out
25
         );
         parameter S0=3'b000, S1=3'b100, S2=3'b110, S3=3'b111, S4=3'b101;
26
27
         reg [2:0] present_state, next_state;
28
         always @(posedge clk or posedge reset) begin
29
          if(reset) present_state<=S0;
30
31
           else present_state<=next_state;
         end
32
```

```
S1: begin
46
47
                   if (data_in_A == data_in_B) begin
48
                     next_state<=S2;
                      count_out <= 0;
49
                   end
50
                   else begin
51
                     next_state<=S0;
52
                      count_out <= 0;
53
                   end
54
                 end
55
             S2: begin
56
                  if(data_in_A == data_in_B) begin
57
58
                     next_state<=S3;
59
                      count_out <= 0;
60
                  end
                   else begin
61
                     next_state<=S0;
62
                     count_out <= 0;
63
                   end
64
65
                 end
```

```
------,
                   end
74
75
                end
            S4: begin
76
                   if(data_in_A==data_in_B) begin
77
                     next_state<=S4;
78
                      count_out <= 1;
79
                   end
80
                   else begin
81
                     next_state<=S0;
82
                      count_out <= 0;
83
84
                 end
85
         endcase
86
        end
87
88
89 endmodule
```

(One Hot)

```
25
        parameter S0=5'b10000, S1=5'b01000, S2=5'b00100, S3=5'b00010, S4=5'b00001;
26
        reg [4:0] present_state, next_state;
27
28
        always @(posedge clk or posedge reset) begin
29
          if(reset) present_state<=S0;
30
          else present_state<=next_state;
31
32
44
45
                 end
             S1: begin
46
47
                   if (data_in_A == data_in_B) begin
48
                     next_state<=S2;
                      count_out <= 0;
49
50
                   end
                   else begin
51
                     next_state<=S0;
52
                      count_out <= 0;
53
                   end
54
                 end
55
             S2: begin
56
                   if(data_in_A == data_in_B) begin
57
                     next_state<=S3;
58
                      count_out <= 0;
59
                   end
60
                   else begin
61
62
                     next_state<=S0;
63
                     count_out <= 0;
64
                   end
                 end
65
```

```
74
                  end
75
                 end
            S4: begin
76
                  if(data_in_A==data_in_B) begin
77
                     next_state<=S4;
78
                     count_out <= 1;
79
                  end
80
                  else begin
81
                    next_state<=S0;
82
                     count_out <= 0;
83
84
                 end
85
          endcase
86
87
        end
88
89 endmodule
```

TESTBENCH:

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

Moore FSM (One State Variable):

```
(Sequential)
34 wire count_out;
35
       // Instantiate the Unit Under Test (UUT)
36
37
       Sequential_Moore uut (
          .data_in_A(data_in_A),
38
          .data_in_B(data_in_B),
39
          .clk(clk),
40
41
          .reset (reset),
          .count_out(count_out)
42
       );
43
       initial begin //clock
44
         clk = 1;
45
46
          forever #25 clk = ~clk;
47
       end
48
       initial begin //reset
49
         reset = 1;
50
          #10;
51
          reset = 0;
52
53
      end
```

```
64
          #50;
65
          data_in_A = 0; //4
          data_in_B = 0;
66
          #50;
67
          data_in_A = 1; //5
68
          data_in_B = 1;
69
          #50;
70
          data_in_A = 1; //6
71
          data_in_B = 0;
72
          #50;
73
          data_in_A = 1; //7
74
          data_in_B = 1;
75
          #50;
76
          data_in_A = 0; //8
77
          data_in_B = 0;
78
          #50;
79
          data_in_A = 0; //9
80
          data_in_B = 0;
81
          #50;
82
```

```
data_in_A = 1; //12
89
        data_in_B = 1;
90
         #50;
91
         data_in_A = 0; //13
92
         data_in_B = 1;
93
         #50;
94
      end
95
96
97 endmodule
```

49

50 51

52

53 54

```
(Gray Code)
35
       // Instantiate the Unit Under Test (UUT)
36
       Gray_Moore uut (
37
38
        .data_in_A(data_in_A),
         .data_in_B(data_in_B),
39
         .clk(clk),
40
         .reset (reset),
41
         .count_out(count_out)
42
43
44
45
      initial begin //clock
       clk = 1;
46
         forever #25 clk = ~clk;
47
      end
48
```

initial begin //reset

reset = 1;

#10; reset = 0;

end

```
data_in_A = 0; //4
66
         data_in_B = 0;
         #50;
67
         data_in_A = 1; //5
68
         data_in_B = 1;
69
         #50;
70
71
         data_in_A = 1; //6
72
         data_in_B = 0;
         #50;
73
         data_in_A = 1; //7
74
         data_in_B = 1;
75
         #50;
76
         data_in_A = 0; //8
77
         data_in_B = 0;
78
         #50;
79
         data_in_A = 0; //9
80
         data_in_B = 0;
81
         #50;
82
```

```
data_III_D - 1;
88
         #50;
89
         data_in_A = 1; //12
90
         data_in_B = 1;
91
         #50;
92
         data_in_A = 0; //13
93
         data_in_B = 1;
94
         #50;
95
      end
96
97
98 endmodule
```

(One Hot)

```
// Instantiate the Unit Under Test (UUT)
37
      One_Hot_Moore uut (
       .data_in_A(data_in_A),
38
        .data_in_B(data_in_B),
39
40
        .clk(clk),
        .reset (reset),
41
        .count_out(count_out)
42
43
44
      initial begin //clock
45
       clk = 1;
46
        forever #25 clk = ~clk;
47
48
49
      initial begin //reset
50
       reset = 1;
51
52
        #10;
        reset = 0;
53
54
           #50;
64
           data_in_A = 0; //4
65
           data_in_B = 0;
66
           #50;
67
           data_in_A = 1; //5
68
          data_in_B = 1;
69
           #50;
70
          data_in_A = 1; //6
71
          data_in_B = 0;
72
           #50;
73
           data_in_A = 1; //7
74
           data_in_B = 1;
75
           #50;
76
           data_in_A = 0; //8
77
           data_in_B = 0;
78
           #50;
79
           data_in_A = 0; //9
80
           data_in_B = 0;
81
           #50;
82
          data_in_b = 1;
88
          #50;
89
          data_in_A = 1; //12
90
          data_in_B = 1;
91
          #50;
92
          data_in_A = 0; //13
93
94
          data_in_B = 1;
          #50;
95
```

Mealy FSM (Two State Variables):

end

98 endmodule

96 97

```
(Sequential)
35
       // Instantiate the Unit Under Test (UUT)
36
       Sequential_Mealy uut (
37
38
         .data_in_A(data_in_A),
         .data_in_B(data_in_B),
39
         .clk(clk),
40
         .reset (reset),
41
         .count_out(count_out)
42
43
44
      initial begin //clock
45
       clk = 1;
46
47
         forever #25 clk = ~clk;
      end
48
49
      initial begin //reset
50
        reset = 1;
51
         #10;
52
         reset = 0;
53
54
      end
```

```
#50;
64
          data_in_A = 0; //4
65
          data_in_B = 0;
66
          #50;
67
68
          data_in_A = 1; //5
          data_in_B = 1;
69
          #50;
70
          data_in_A = 1; //6
71
          data_in_B = 0;
72
          #50;
73
          data_in_A = 1; //7
74
          data_in_B = 1;
75
          #50;
76
          data_in_A = 0; //8
77
          data_in_B = 0;
78
79
          #50;
80
          data_in_A = 0; //9
          data_in_B = 0;
81
          #50;
82
```

```
data_in_b = 1;
88
         #50;
89
         data_in_A = 1; //12
90
         data_in_B = 1;
91
92
          #50;
         data_in_A = 0; //13
93
         data_in_B = 1;
94
95
          #50;
96
       end
97
98 endmodule
```

```
(Gray Code)
35
       // Instantiate the Unit Under Test (UUT)
36
       Gray_Mealy uut (
37
          .data_in_A(data_in_A),
38
          .data_in_B(data_in_B),
39
         .clk(clk),
40
         .reset (reset),
41
          .count_out(count_out)
42
      );
43
44
      initial begin //clock
45
       clk = 1;
46
         forever #25 clk = ~clk;
47
48
49
      initial begin //reset
50
        reset = 1;
51
          #10;
52
         reset = 0;
53
54
      end
```

```
#50;
64
65
          data_in_A = 0; //4
          data_in_B = 0;
66
          #50;
67
          data_in_A = 1; //5
68
          data_in_B = 1;
69
          #50;
70
          data_in_A = 1; //6
71
          data_in_B = 0;
72
          #50;
73
          data in A = 1; //7
74
75
          data_in_B = 1;
          #50;
76
          data_in_A = 0; //8
77
          data_in_B = 0;
78
          #50;
79
          data_in_A = 0; //9
80
          data_in_B = 0;
81
          #50;
82
```

```
data_in_A = 1; //12
90
        data_in_B = 1;
91
        #50;
92
        data_in_A = 0; //13
93
        data_in_B = 1;
94
        #50;
95
      end
96
97
98 endmodule
```

```
(One Hot)
```

```
35
       // Instantiate the Unit Under Test (UUT)
36
       One_Hot_Mealy uut (
37
       .data_in_A(data_in_A),
38
        .data_in_B(data_in_B),
39
        .clk(clk),
40
        .reset (reset),
41
         .count_out(count_out)
42
43
44
      initial begin //clock
45
        clk = 1;
46
         forever #25 clk = ~clk;
47
      end
48
49
      initial begin //reset
50
        reset = 1;
51
         #10;
52
        reset = 0;
53
     end
54
```

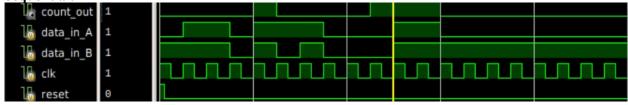
```
data_in_A = 0; //4
65
66
           data_in_B = 0;
           #50;
67
68
           data_in_A = 1; //5
69
           data_in_B = 1;
           #50;
70
           data_in_A = 1; //6
71
           data_in_B = 0;
72
           #50;
73
           data_in_A = 1; //7
74
           data_in_B = 1;
75
           #50;
76
77
           data_in_A = 0; //8
           data_in_B = 0;
78
           #50;
79
           data_in_A = 0; //9
80
           data_in_B = 0;
81
           #50;
82
          data_in_b = 1;
88
          #50;
          data_in_A = 1; //12
          data_in_B = 1;
          #50;
```

```
89
90
91
92
93
          data_in_A = 0; //13
94
          data_in_B = 1;
          #50;
95
96
       end
97
98 endmodule
```

Simulation Results

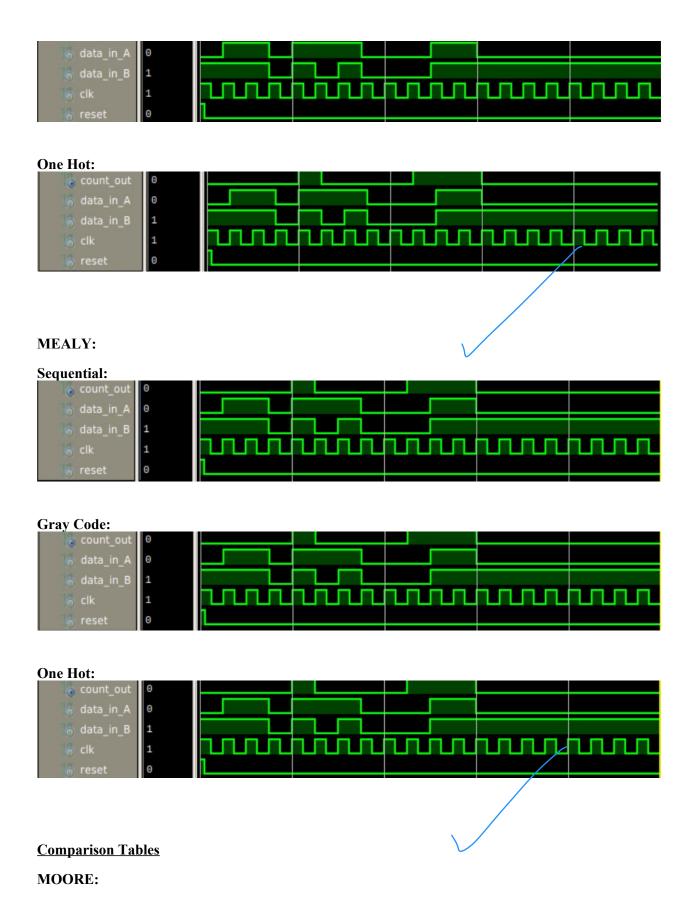
MOORE:

Sequential:



Gray Code:





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

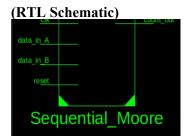
One Hot:

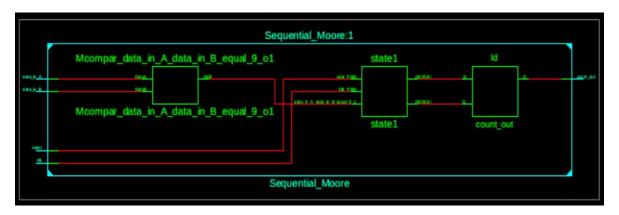
Best Path Delay	Worst Path Delay	Area	Power
2.892 ns	2.892 ns	82	0.082 W

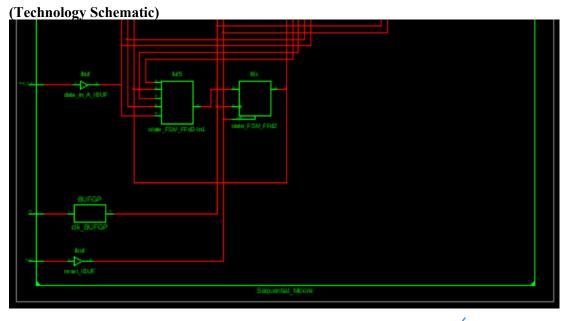
RTL Schematic and Technology Schematic

MOORE:

Sequential:

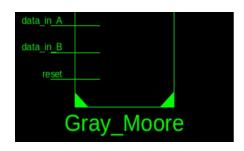


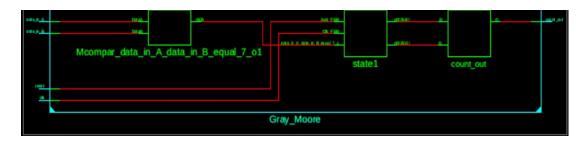


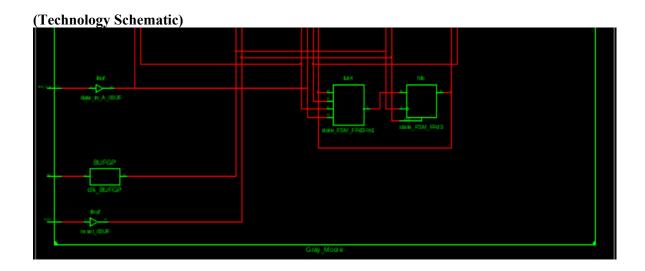


Gray Code:

(RTL Schematic)

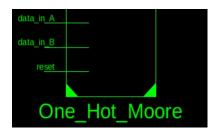


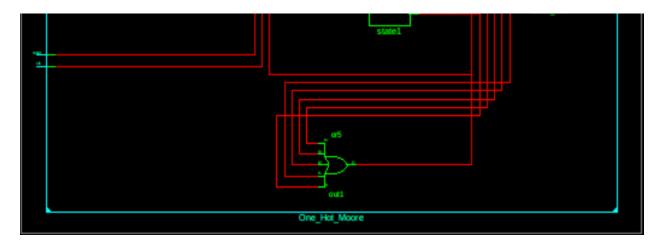




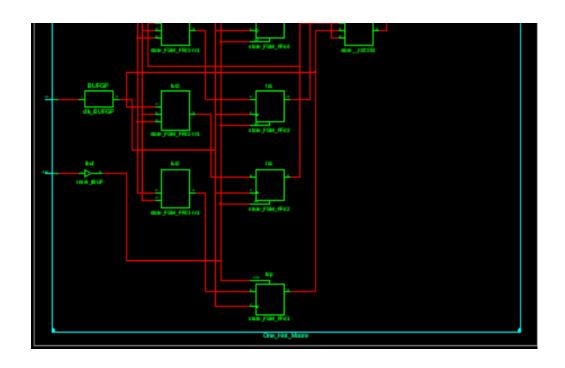
One Hot:

(RTL Schematic)



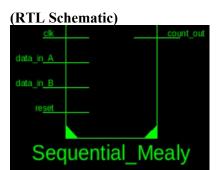


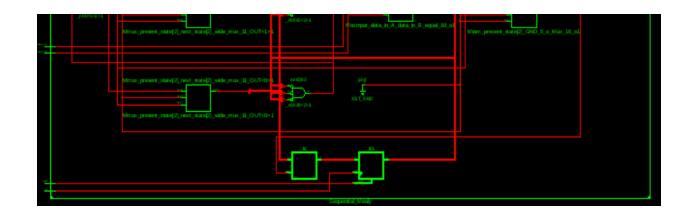
(Technology Schematic)

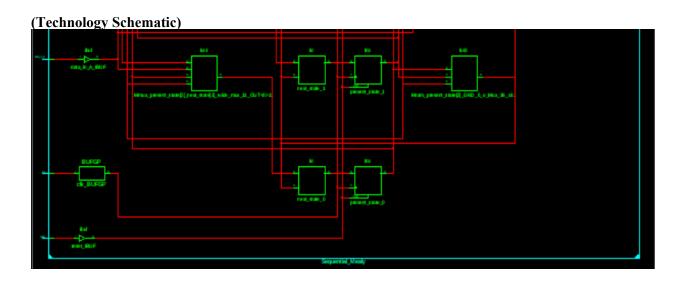


MEALY:

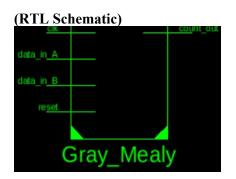
Sequential:

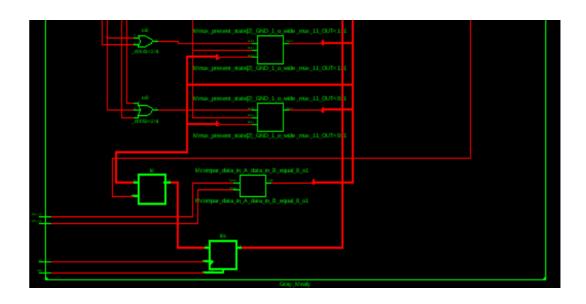


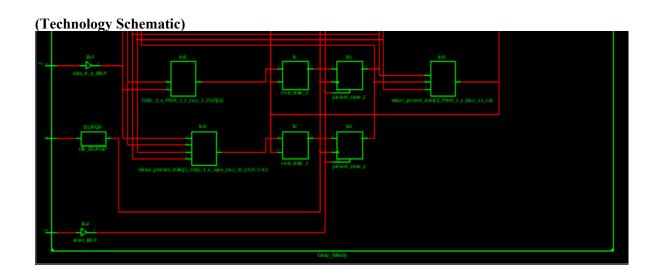




Gray Code:

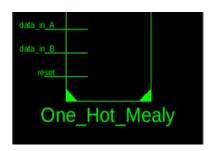


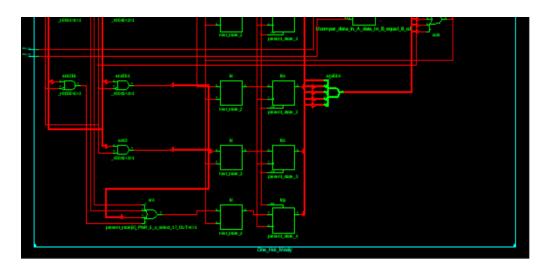




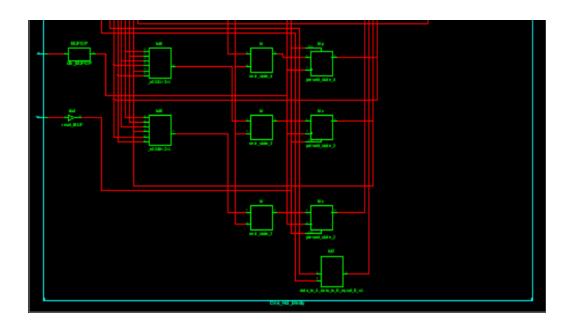
One Hot:

(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]: Nexys4TM 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 #	S	
Lab Description	Algorithm	ic State Machine

Lab Responsibility Assignment

Student Name	Responsibility	Signature
Nikeem Dunkelly-Allen	One - hur menty + extbonin	Walun
Brandon Rubio	Glay - moore withenin	Roads Rulis
Rainier Young	PIN-Mapping	gure
Isabela Bandrich	state Machine Coda Report	Estable Band: M

Lab Demonstration

Teaching Assistant Signature	Date
thurana-	7-11-22
Comments:	
Demonstration sho	oun on time.
ASM WOLKINg.	