Introduction

Finite state machines (FSM) are some of the many computational models that can be used to simulate sequential logic on an FPGA. FSM models can be classified into 2 categories, mealy FSM and moore FSM. Mealy FSM's outputs are determined by both the current state and current input. However, moore FSM's outputs are determined only by its current state. This lab aims to explore the implementation of a moore FSM on the NEXYS 4 DDR and its feasibility when compared to a test bench.

FSM logic

Moore FSM output is determined by its current state, as a result, it is important to consider every transition logic and the input required for these logic to be executed. A moore machine transition logic can be implemented on an FPGA with the statements below.

1. The input sequence ain[1:0] = 01, 00 causes the output to become 0.
2. The input sequence ain[1:0] = 11, 00 causes the output to become 1.
3. The input sequence ain[1:0] = 10, 00 causes the output to become toggle.

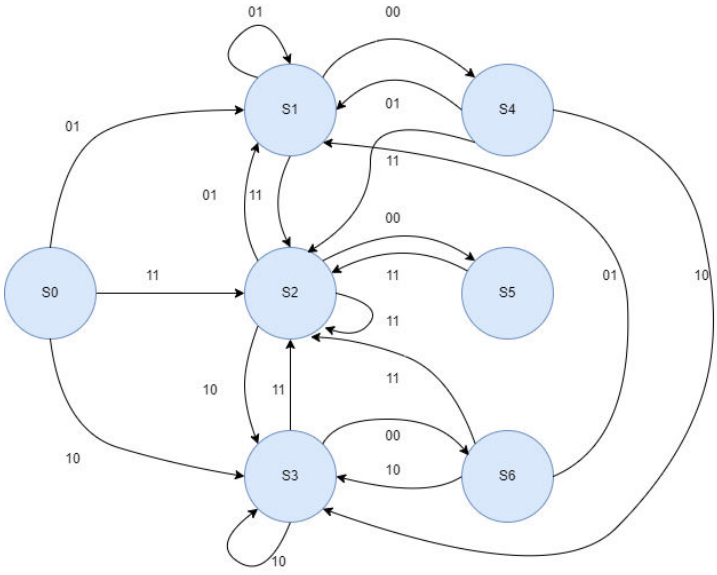

Figure 2.1: Moore state machine diagram for ain.

Figure 2.1 illustrates the required states for output. S4,S5 and S6 are the only states where the LED outputs may alter. In contrast, S1,S2 and S3 are buffer states. Additionally, S0 will be the initial state and output 0. Furthermore, these states and its corresponding output can be shown in Table 1. Previously mentioned states can be achieved with a provided input sequence "ain" such as 01, 11, 10. This design step is integral in developing an FSM as it simplifies the logic in great detail before implementation.

Current state	output
S0	0
S1	x
S2	x
S3	x
S4	0
S5	1
S6	Toggle

Table 1: Figure 2.1's states and corresponding outputs.

FSM implementation

Implementing the moore FSM requires 2 main processes, output decode logic, and transition logic. Additionally, states of these FSM will be declared as "type" to comply with the RTL architecture. Furthermore, "state" and "next_state" will be declared as signal, and "prev" signal was declared as a median to memorise the previous state output as shown in Figure 3.1

```
34  entity moore is
35      Port ( ain : in STD_LOGIC_VECTOR (1 downto 0);
36           clk : in STD_LOGIC; Reset : in STD_LOGIC;
37             m : inout STD_LOGIC);
38  end moore;
39
40  architecture Behavioral of moore is
41
42  type state_type is (S0, S1, S2, S3, S4, S5, S6);
43  signal state, next_state : state_type;
44  signal prev : STD_LOGIC := '0'; --previous state
```

Figure 3.1: Declaring moore states.

To transition to the 3 buffer states (S1, S2, S3), S0 must be the default FSM state. This will allow the S0 to transition to the buffer states based on the inputs provided in Table 1 to as shown in Figure 3.2. Consequently, for the buffer states to transition to the output states (S4, S5, S6) it will require a "00" input after reaching the buffer state; an example can be provided in Figure 3.3. This fits within the parameters of a moore machine as the output will only change based on the current state the machine is in e.g (S4 : 0, S5 : 1, S6 : toggle) as shown in Figure 3.4. However, the output state has the options to return back to buffer states or the default FSM as shown in Figure 3.5; following the logic in section 2.

```
when S0 =>
      if (ain = "01") then
            next_state <= S1;

      elsif (ain = "11") then
            next_state <= S2;

      elsif (ain = "10") then
            next_state <= S3;
      else
            next_state <= S0;
      end if;
```

Figure 3.2: Default state to buffer state transition logic.

```
when S3 =>
      if (ain = "01") then
            next_state <= S1;

      elsif (ain = "11") then
            next_state <= S2;

      elsif (ain = "10") then
            next_state <= S3;

      elsif (ain = "00") then
            next_state <= S6;
      else
            next_state <= S0;
      end if;
```

Figure 3.3: Buffer state to output state logic example (ain = “00”).

```
61      OUTPUT_DECODE : process (state)
62      begin
63          m <= '0';
64          prev <= m;
65            case (state) is
66                  when S4 =>
67                        m <= '0';
68                        prev <= m;
69                  when S5 =>
70                        m <= '1';
71                        prev <= m;
72                  when S6 =>
73                        m <= not prev;
74                        prev <= m;
75                  when others =>
76                        prev <= m;
77                  end case;
78      end process;
```

Figure 3.4: Output decode state when the 3 output states are reached (S4 : 0, S5 : 1, S6 : T).

```
137
138   when S4 =>
139         if (ain = "01") then
140               next_state <= S1;
141
142         elsif (ain = "11") then
143               next_state <= S2;
144
145         elsif (ain = "10") then
146               next_state <= S3;
147
148         elsif (ain = "00") then
149               next_state <= S0;
150         else
151               next_state <= S4; -- else stays in this state
          end if;
```

Figure 3.5: Output state to buffer state/default state logic example (lines 138 to 148). .

FSM testbench

```
37
38      architecture Behavioral of moore_tb is
39      component moore is
40          Port ( ain : in STD_LOGIC_VECTOR (1 downto 0);
41               clk : in STD_LOGIC; Reset : in STD_LOGIC;
42                 m : inout STD_LOGIC);
43      end component;
44
45      signal clock : std_logic := '0';
46      signal r : std_logic := '0';
47      signal input : std_logic_vector(1 downto 0) := "00";
48      signal output : std_logic := '0';
49      constant clk_period : time := 10ns;
```

Figure 4.1: Moore machine testbench declaration.

As shown in Figure 4.1, the test bench for this FSM was created and the FSM must be declared as a component to allow for instanciatance. Consequently, signals must be declared to allow the testbench sequences to be inputted into the moore machine. Finally, the moore machine was instantiated and its inputs were routed to the previously declared signals as shown in figure 4.2.

```
53      moore_instance: moore PORT MAP (
54      clk => clock,
55      Reset => r,
56      ain => input,
57      m => output
58      );
```

Figure 4.2: Moore instance routed to testbench signals.

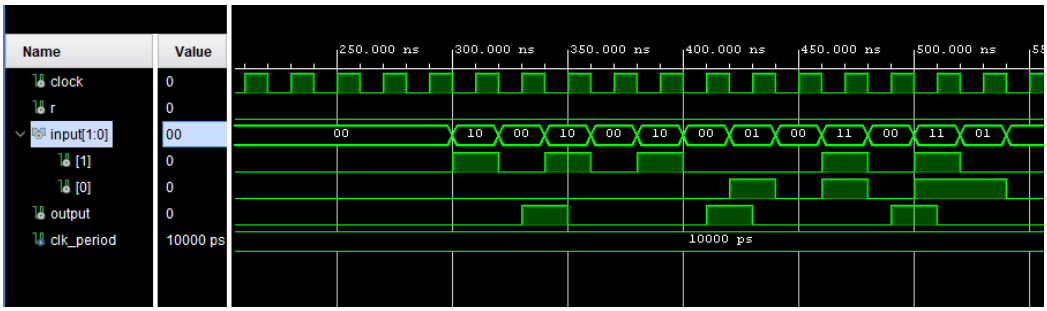

Figure 4.3: Testbench output of the moore machine.

As illustrated in Figure 4.3, the moore machine works as intended when given a set of input sequences (as shown in the appendix); its inputs are based on the logic stated in section 2. When ain = “11”, followed by ain = “00”, the LED outputs a 1. Additionally, the toggle state based on the previous output works accordingly (as shown between 300ns to 425ns in Figure 4.3).

Conclusion

In conclusion, the implementation of a moore FSM can be simplified with the use of diagrams and logic statements. Testbench is an important part of an FPGA circuit design as it allows developers to have an in-depth understanding of how the circuit design would react based on a set of automated inputs without the hassle of manual test cases. As section 4, the moore machine was quickly validated with the use of a testbench.

Appendix

```
34      entity moore is
35          Port ( ain : in STD_LOGIC_VECTOR (1 downto 0);
36               clk : in STD_LOGIC; Reset : in STD_LOGIC;
37                 m : inout STD_LOGIC);
38      end moore;
39
40      architecture Behavioral of moore is
41
42      type state_type is (S0, S1, S2, S3, S4, S5, S6);
43      signal state, next_state : state_type;
44      signal prev : STD_LOGIC := '0'; --previous state
45
46
47      begin
48
49
50      SYNC_PROC : process (clk)
51      begin
52          if rising_edge(clk) then
53                  if (reset = '1') then
54                        state <= S0;
55                  else
56                        state <= next_state;
57                  end if;
58           end if;
59      end process;
60
61      OUTPUT_DECODE : process (state)
62      begin
63          m <= '0';
64          prev <= m;
65            case (state) is
66                  when S4 =>
67                        m <= '0';
68                        prev <= m;
69                  when S5 =>
70                        m <= '1';
71                        prev <= m;
72                  when S6 =>
73                        m <= not prev;
74                        prev <= m;
75                  when others =>
76                        prev <= m;
77                  end case;
78      end process;
```

```
80      NEXT_STATE_DECODE : process (state, ain)
81      begin
82            next_state <= S0;
83            case (state) is
84                  when S0 =>
85                        if (ain = "01") then
86                              next_state <= S1;
87
88                        elsif (ain = "11") then
89                              next_state <= S2;
90
91                        elsif (ain = "10") then
92                              next_state <= S3;
93                        else
94                              next_state <= S0;
95                        end if;
96                  when S1 =>
97                        if (ain = "01") then
98                              next_state <= S1;
99
100                       elsif (ain = "11") then
101                             next_state <= S2;
102
103                       elsif (ain = "10") then
104                             next_state <= S3;
105
106                       elsif (ain = "00") then
107                             next_state <= S4;
108                       end if;
109                 when S2 =>
110                       if (ain = "01") then
111                             next_state <= S1;
```

```
112                                     _
113                     elsif (ain = "11") then
114                           next_state <= S2;
115
116                     elsif (ain = "10") then
117                           next_state <= S3;
118
119                     elsif (ain = "00") then
120                           next_state <= S5;
121               end if;
122         when S3 =>
123               if (ain = "01") then
124                     next_state <= S1;
125
126               elsif (ain = "11") then
127                     next_state <= S2;
128
129               elsif (ain = "10") then
130                     next_state <= S3;
131
132               elsif (ain = "00") then
133                     next_state <= S6;
134               else
135                     next_state <= S0;
136               end if;
137         when S4 =>
138               if (ain = "01") then
139                     next_state <= S1;
140
```

```
140
141                     elsif (ain = "11") then
142                         next_state <= S2;
143
144                     elsif (ain = "10") then
145                         next_state <= S3;
146
147                     elsif (ain = "00") then
148                         next_state <= S0;
149                     else
150                         next_state <= S4; -- else stays in this state
151                     end if;
152                 when S5 =>
153                     if (ain = "01") then
154                         next_state <= S1;
155
156                     elsif (ain = "11") then
157                         next_state <= S2;
158
159                     elsif (ain = "10") then
160                         next_state <= S3;
161
162                     elsif (ain = "00") then
163                         next_state <= S0;
164                     else
165                         next_state <= S5; -- else stays in this state
166                     end if;
167                 when S6 =>
168                     if (ain = "01") then
169                         next_state <= S1;
170                     elsif (ain = "11") then
171                         next_state <= S2;
172
173                     elsif (ain = "10") then
174                         next_state <= S3;
175                     elsif (ain = "00") then
176                         next_state <= S0;
177                     else
178                         next_state <= S6; -- else stays in this state
179                     end if;
180                 when others =>
181                     next_state <= S0;
182             end case;
183     end process;
184     end Behavioral;
185
```

Testbench

```
34      entity moore_tb is
35      --  Port ( );
36      end moore_tb;
37
38      architecture Behavioral of moore_tb is
39      component moore is
40          Port ( ain : in STD_LOGIC_VECTOR (1 downto 0);
41               clk : in STD_LOGIC; Reset : in STD_LOGIC;
42                 m : inout STD_LOGIC);
43      end component;
44
45      signal clock : std_logic := '0';
46      signal r : std_logic := '0';
47      signal input : std_logic_vector(1 downto 0) := "00";
48      signal output : std_logic := '0';
49      constant clk_period : time := 10ns;
50
51      begin
52
53      moore_instance: moore PORT MAP (
54      clk => clock,
55      Reset => r,
56      ain => input,
57      m => output
58      );
59
```

```
61      process
62      begin
63          r <= '1';
64          wait for 100ns;
65          r <= '0';
66          wait for 200ns;
67          input <= "10";
68          wait for 20ns;
69          input <= "00"; -- test the 3rd condition (S3, S6)
70          wait for 20ns;
71          input <= "10";
72          wait for 20ns;
73          input <= "00"; -- test the 3rd condition (S3, S6)
74          wait for 20ns;
75          input <= "10";
76          wait for 20ns;
77          input <= "00"; -- test the 3rd condition (S3, S6)
78          wait for 20ns;
79          input <= "01";
80          wait for 20ns;
81          input <= "00"; -- test the 1st condition (S1, S4)
82          wait for 20ns;
83          input <= "11";
84          wait for 20ns;
85          input <= "00"; -- test the 2nd condition (S2, S5)
86          wait for 20ns;
87          input <= "11"; -- test jump conditions (S2 to S1)
88          wait for 20ns;
89          input <= "01";
90          wait for 20ns;
91          input <= "00"; -- test the 1st condition (S1, S4)
92          wait for 20ns;
93          wait;
94      end process;
95
96      clock <= not clock after clk_period;
97
98
99      end Behavioral;
```