EEL 4712C - Digital Design: Lab Report 3

Cole Rottenberg
11062528

March 4, 2024

Prelab Report

1. Part 1: One Process FSM
2. Part 2: Two Process FSM
3. Part 3: Demo
   (a) Test on hardware.
   (b) Assign inputs and outputs to the switches and LEDs.
   (c) Display to TA Fib(11) = 89.

Prelab Questions

Highlighting the key differences between a one process and two process FSM.
The major difference between a one process and two process FSM is the flow of states and the logic that controls when a state change occurs. In a one process FSM, our sensitivity list contains a **clk and rst** signal. This means that the state machine will only change states when the clock signal changes. In a two process FSM, our sensitivity list contains the **state and input(s)** signals. This means that the state machine will change states when the state or input signals change. However, the state change in a two process FSM is iterated by a clock signal. This means that the state machine will only change states when the clock signal changes. This adds a layer of abstraction to the state machine, allowing for more complex state machines to be designed.

Prelab Design and Implementation

The design process started with analyzing the given pseudo code and diagram of components of the Fibonacci Sequence from Lab 2. Then I was able to come up with a basic implementation of a state machine picture in figure 1. The state machine controls the enable of need register **i,x,y, and n** and the select lines of multiplexer for **i,x,y, and n**. The both state machines work by first checking for input signals such as **go and rst** before moving to an initial state. The initial loads the **n** register with the input value while other register loads base values for loops and arithmetic. The FSM then checks the **n** register for a zero value, if it is zero, the FSM moves to the final state and outputs a preset zero value. Otherwise, the FSM moves to the next state and begins the iterative process of calculating the Fibonacci Sequence. The a state checks if we have iterated up to the input value, if so we move to the done state. If we have not, we move to the compute stage which loads the **y** register into the **x** register. The FSM then moves to ADD state which loads the summation of **x** and **y** registers into the **y** register. We do so in two seperate states as the addition of the two registers requires a clock cycle. While moving the **y** register to the **x** register, we also iterate the **i** register as to count the number of iterations. We then loop back to the original CHECK state and continue the process until we reach the input value. As mentioned before, if we reach the input value, we move to the done state and output the **done** signal. Before moving from the done state, we must also check of the original **go** signal has reached zero. If it has, we move to the restart state which also outputs

the **done** signal. If the **go** signal is not zero, we stay in the done state until it is zero. From the restart state, we are able to move back to the initial state if we receive a **go** signal. This is the basic design of the state machine. The implementation of the state machine is done in VHDL and is shown in the appendix. The one process FSM code is shown in listing 1 and the two process FSM code is shown in listing 2.

Reflection

Prelab Homework

Postlab Report

Problem Statement

The goal of the lab was to implement a working FSM to control the datapath implemented in lab 2. The control focused around enabling certain registers and select lines. The inputs used by the FSM include: clk, rst, go, n_eq_0, and i_le_n. The outputs of the FSM include: done, n_en, result_en, result_sel, x_en, x_sel, y_en, y_sel, i_en, and i_sel. The function of the system is to control the datapath to calculate the Fibonacci Sequence. The FSM controls the enable and select lines of the registers and multiplexers in the datapath. The FSM also outputs the done signal when the computation is complete.

Design

Implementation

Testing

The testing of the design was done by first simulating the FSM in ModelSim. The simulation was done by creating a testbench that would simulate the FSM with a set of inputs. The inputs were chosen to test the FSM in a scenario entering a loop and then recieving the **i_le_n** signal. The simulation was successful and the FSM was able to iterate through the states as expected. After recieving the **i_le_n**, the FSM reached the done state and outputted the **done** signal. We also can observe the individual states of computation by analyzing the enable and select signals of the registers and multiplexers. The testbench code was used for both the one process and two process FSMs. The testbench code is shown in listing 3.

Conclusions

Appendix

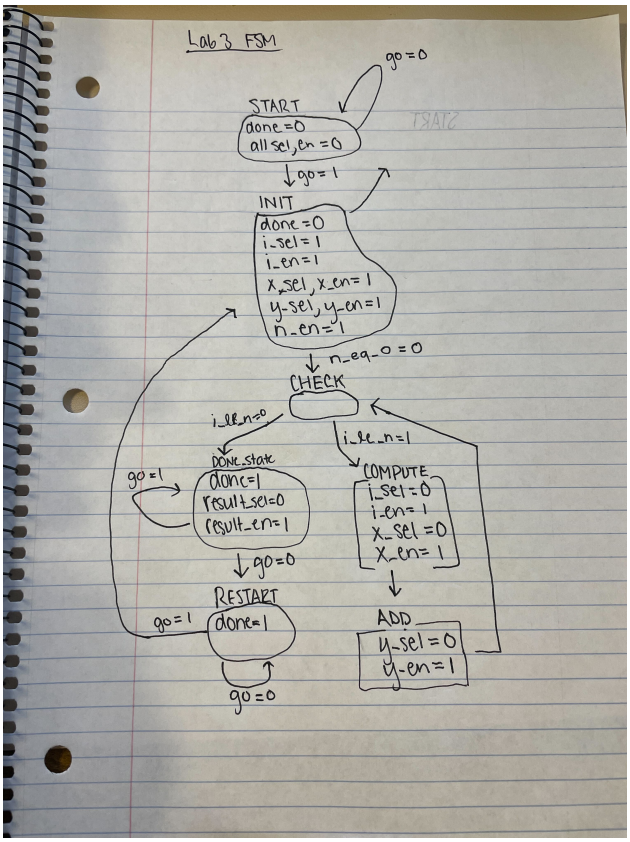

Figure 1: State Diagram

```
1   library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  -- FSM:
5  -- Building a FSM for the fibonacci datapath
6  entity fsm is
7      port(
8          -- Inputs
9          clk:          in std_logic;
10         rst:          in std_logic;
11         go:           in std_logic;
12         n_eq_0:       in std_logic;
13         i_le_n:       in std_logic;
14
15         -- Outputs
16         done:         out std_logic;
17
18         -- Control signals
19         n_en:         out std_logic;
```

```
20          result_en:  out std_logic;
21          result_sel: out std_logic;
22          x_en:       out std_logic;
23          x_sel:      out std_logic;
24          y_en:       out std_logic;
25          y_sel:      out std_logic;
26          i_en:       out std_logic;
27          i_sel:      out std_logic
28      );
29  end entity fsm;
30
31  architecture behavioral of fsm is
32      -- Define the states
33      type state_t is (START, INIT, COMPUTE, ADD, BUFF_LE, CHECK_LE,
            DONE_STATE, RESTART);
34      signal state, next_state:       state_t := START;
35
36  begin
37      -- No Clock Style
38      -- Logic for state transitions
39      process(clk, rst)
40      begin
41          if (rst = '1') then
42              done <= '0';
43              state <= START;
44          -- Default values
45          -- State transitions
46          elsif (rising_edge(clk)) then
47              done <= '0';
48              i_sel <= '0';
49              i_en <= '1';
50              x_sel <= '0';
51              x_en <= '1';
52              y_sel <= '0';
53              y_en <= '1';
54              n_en <= '1';
55              result_en <= '1';
56              result_sel <= '0';
57
58              case state is
59                  when START =>
60                      -- All to 0
61                      i_en <= '0';
62                      x_en <= '0';
63                      y_en <= '0';
64                      n_en <= '0';
65                      done <= '0';
66                      if go = '1' then
67                          state <= INIT;
68                      else
69                          state <= START;
70                      end if;
71                  -- Implement default defined values for every state
72
```

```
73                  when INIT =>
74                      -- Unable Done
75                      done <= '0';
76                      i_sel <= '1';
77                      i_en <= '1';
78                      x_sel <= '1';
79                      x_en <= '1';
80                      y_en <= '1';
81                      n_en <= '1';
82                      if (n_eq_0 = '0') then
83                          result_sel <= '0'; -- Select the result
84                          state <= CHECK_LE;
85                      else
86                          result_sel <= '1'; -- Select default 0 result if n = 0
87                          state <= DONE_STATE;
88                      end if;
89
90                  when CHECK_LE =>
91                      -- Check if i <= n
92                      if (i_le_n = '1') then
93                          state <= COMPUTE;
94                      else
95                          state <= DONE_STATE;
96                      end if;
97
98                  when COMPUTE =>
99                      i_sel <= '0';
100                     i_en <= '1'; -- Redundant
101                     x_sel <= '0';
102                     x_en <= '1'; -- Redundant
103
104                     state <= ADD;
105
106                 when ADD =>
107                     -- Allows Clock Cycle so the addition of x and y can be
                           done before y is loaded
108                     y_sel <= '0';
109                     y_en <= '1'; -- Redundant
110
111                     state <= BUFF_LE;
112
113                 when BUFF_LE =>
114                     state <= CHECK_LE;
115
116                 when DONE_STATE =>
117                     result_sel <= '0'; -- DBG
118                     result_en <= '1'; -- Only enable the result... the init
                           state handles which result to select
119                     -- Need to prevent bad state loops because of race
                           conditions
120                     done <= '1';
121                     if go = '1' then
122                         state <= DONE_STATE;
123                     else
```

```
124                         state <= RESTART;
125                     end if;
126                 when RESTART =>
127                     result_sel <= '0'; -- DBG
128                     result_en <= '1'; -- Only enable the result... the init
                            state handles which result to select
129                     done <= '1';
130                     -- Now we can restart the process if go is high
131                     if go = '1' then
132                         state <= INIT;
133                     else
134                         state <= RESTART;
135                     end if;
136                 when others => null;
137             end case;
138         end if;
139     end process;
140 end architecture behavioral;
```

Listing 1: One Process FSM VHDL Code

```
 1   library ieee;
 2 use ieee.std_logic_1164.all;
 3 use ieee.numeric_std.all;
 4
 5 -- FSM:
 6 -- Building a FSM for the fibonacci datapath
 7
 8 entity fsm is
 9     port(
10         -- Inputs
11         clk:        in std_logic;
12         rst:        in std_logic;
13         go:         in std_logic;
14         n_eq_0:     in std_logic;
15         i_le_n:     in std_logic;
16
17         -- Outputs
18         done:       out std_logic;
19
20         -- Control signals
21         n_en:       out std_logic;
22         result_en:  out std_logic;
23         result_sel: out std_logic;
24         x_en:       out std_logic;
25         x_sel:      out std_logic;
26         y_en:       out std_logic;
27         y_sel:      out std_logic;
28         i_en:       out std_logic;
29         i_sel:      out std_logic
30     );
31
32 end entity fsm;
33
34 architecture behavioral of fsm is
```

```
35      -- Define the states
36      type state_type is (START, INIT, COMPUTE, ADD, CHECK_LE, DONE_STATE,
            RESTART);
37      signal state, next_state:       state_type;
38
39  begin
40      -- Logic for clock and reset
41      process(clk, rst)
42      begin
43          if rst = '1' then
44              state <= START; -- Reset state
45          elsif rising_edge(clk) then
46              state <= next_state; -- Update state
47          end if;
48
49      end process;
50
51      -- Logic for state transitions
52      process(state)
53      begin
54          -- Default values to prevent latches
55          -- State transitions
56          next_state <= state;
57          case state is
58
59              when START =>
60                  -- All to 0
61                  i_en <= '0';
62                  x_en <= '0';
63                  y_en <= '0';
64                  n_en <= '0';
65                  done <= '0';
66
67                  if go = '1' then
68                      next_state <= INIT;
69                  else
70                      next_state <= START;
71                  end if;
72                 -- Implement default defined values for every state
73
74              when INIT =>
75                  -- Unable Done
76                  done <= '0';
77                  i_sel <= '1';
78                  i_en <= '1';
79                  x_sel <= '1';
80                  x_en <= '1';
81                  y_sel <= '1';
82                  y_en <= '1';
83                  n_en <= '1';
84                  next_state <= CHECK_LE;
85
86              when CHECK_LE =>
87                  -- Check if i <= n
```

```
88                  if (i_le_n = '1') then
89                      next_state <= COMPUTE;
90                  else
91                      next_state <= DONE_STATE;
92                  end if;
93
94              when COMPUTE =>
95                  i_sel <= '0';
96                  i_en <= '1'; -- Redundant
97                  x_sel <= '0';
98                  x_en <= '1'; -- Redundant
99
100                 next_state <= ADD;
101
102             when ADD =>
103                 -- Allows Clock Cycle so the addition of x and y can be done
                        before y is loaded
104                 y_sel <= '0';
105                 y_en <= '1'; -- Redundant
106
107                 next_state <= CHECK_LE;
108             when DONE_STATE =>
109                 result_sel <= '0'; -- DBG
110                 result_en <= '1'; -- Only enable the result... the init state
                        handles which result to select
111                 -- Need to prevent bad state loops because of race conditions
112                 done <= '1';
113                 if go = '1' then
114                     next_state <= DONE_STATE;
115                 else
116                     next_state <= RESTART;
117                 end if;
118             when RESTART =>
119                 result_sel <= '0'; -- DBG
120                 result_en <= '1'; -- Only enable the result... the init state
                        handles which result to select
121                 done <= '1';
122                 -- Now we can restart the process if go is high
123                 if go = '1' then
124                     next_state <= INIT;
125                 else
126                     next_state <= RESTART;
127                 end if;
128             when others => null;
129         end case;
130     end process;
131 end architecture behavioral;
```

Listing 2: Two Process FSM VHDL Code

```
1   library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use ieee.numeric_std.all;
5
```

```
6  -- Testbench for the our top_level Fibonacci state machine
7
8  entity fsm_tb is
9  end fsm_tb;
10
11 architecture tb of fsm_tb is
12   -- Inputs
13   signal clk :          std_logic := '0';
14   signal rst :          std_logic := '0';
15   signal go :           std_logic := '0';
16   signal n_eq_0 :       std_logic := '0';
17   signal i_le_n :       std_logic := '0';
18
19   -- Outputs
20   signal done:          std_logic := '0';
21
22   -- Signals for controlling Datapath
23   signal n_en:          std_logic := '0';
24   signal result_en:     std_logic := '0';
25   signal result_sel:    std_logic := '0';
26   signal x_en:          std_logic := '0';
27   signal x_sel:         std_logic := '0';
28   signal y_en:          std_logic := '0';
29   signal y_sel:         std_logic := '0';
30   signal i_en:          std_logic := '0';
31   signal i_sel:         std_logic := '0';
32
33   -- Datapath inputs
34
35   -- Clock
36   constant clk_period : time := 5 ns;
37
38 begin
39   -- Instantiate the FSM
40   UUT : entity work.fsm(behavioral)
41     port map (
42       clk => clk,
43       rst => rst,
44       go => go,
45       n_eq_0 => n_eq_0,
46       i_le_n => i_le_n,
47       done => done,
48
49       n_en => n_en,
50       result_en => result_en,
51       result_sel => result_sel,
52       x_en => x_en,
53       x_sel => x_sel,
54       y_en => y_en,
55       y_sel => y_sel,
56       i_en => i_en,
57       i_sel => i_sel
58     );
59
```

```
60    -- Clock process definitions
61    clk <= not clk after clk_period;
62
63    -- Stimulus process
64
65    process
66    begin
67      -- Reset
68      rst <= '1';
69      wait until rising_edge(clk);
70      rst <= '0';
71      -- Start the FSM
72      wait for 10 ns;
73      go <= '1';
74      i_le_n <= '1';
75      wait for 10 ns;
76
77      wait for 100 ns;
78      i_le_n <= '0';
79
80      wait for 20 ns;
81      go <= '0';
82      -- Wait for the FSM to finish
83      wait until done = '1';
84
85      -- End the simulation
86      wait;
87    end process;
88  end tb;library ieee;
89  use ieee.std_logic_1164.all;
90  use ieee.std_logic_arith.all;
91  use ieee.numeric_std.all;
92
93  -- Testbench for the our top_level Fibonacci state machine
94
95  entity fsm_tb is
96  end fsm_tb;
97
98  architecture tb of fsm_tb is
99    -- Inputs
100   signal clk :          std_logic := '0';
101   signal rst :          std_logic := '0';
102   signal go :           std_logic := '0';
103   signal n_eq_0 :       std_logic := '0';
104   signal i_le_n :       std_logic := '0';
105
106   -- Outputs
107   signal done:          std_logic := '0';
108
109   -- Signals for controlling Datapath
110   signal n_en:          std_logic := '0';
111   signal result_en:     std_logic := '0';
112   signal result_sel:    std_logic := '0';
113   signal x_en:          std_logic := '0';
```

```
114   signal x_sel:          std_logic := '0';
115   signal y_en:           std_logic := '0';
116   signal y_sel:          std_logic := '0';
117   signal i_en:           std_logic := '0';
118   signal i_sel:          std_logic := '0';
119
120   -- Datapath inputs
121
122   -- Clock
123   constant clk_period : time := 5 ns;
124
125 begin
126   -- Instantiate the FSM
127   UUT : entity work.fsm(behavioral)
128     port map (
129       clk => clk,
130       rst => rst,
131       go => go,
132       n_eq_0 => n_eq_0,
133       i_le_n => i_le_n,
134       done => done,
135
136       n_en => n_en,
137       result_en => result_en,
138       result_sel => result_sel,
139       x_en => x_en,
140       x_sel => x_sel,
141       y_en => y_en,
142       y_sel => y_sel,
143       i_en => i_en,
144       i_sel => i_sel
145     );
146
147   -- Clock process definitions
148   clk <= not clk after clk_period;
149
150   -- Stimulus process
151
152   process
153   begin
154     -- Reset
155     rst <= '1';
156     wait until rising_edge(clk);
157     rst <= '0';
158     -- Start the FSM
159     wait for 10 ns;
160     go <= '1';
161     i_le_n <= '1';
162     wait for 10 ns;
163
164     wait for 100 ns;
165     i_le_n <= '0';
166
167     wait for 20 ns;
```

```
168       go <= '0';
169       -- Wait for the FSM to finish
170       wait until done = '1';
171
172       -- End the simulation
173       wait;
174     end process;
175 end tb;
```

Listing 3: Testbench VHDL Code