ECEN302 Lab 4 - Finite State Machines

Joshua Benfell - 300433229

August 30, 2020

1 Objectives

The primary objectives of this lab was to implement and understand different types of finite state machines. The machines that are being implemented here are the Mealy FSM and the Moore FSM. From doing this we will gain an understanding of the pros and cons of each type and where to use them.

2 Methodology

2.1 Introduction

In this lab a mealy and moore state machine will be implemented. The mealy state machine will be implemented as a mod three counter. The moore state machine will be designed such a particular 2 bit input will result in a predefined output. This will be done using the VHDL language.

2.2 Mealy FSM

A mod three counter was implemented as a mealy state machine. How this works is by inputting individual bits one at a time, if the current number of 1s seen is a multiple of 3, it will output high. The bit value is passed into the state machine on the rising edge of the clock cycle.

This was very difficult to think about as a mealy state machine due to there being so few states. A mealy state machine defines it's output based on the current state and the current input, however, there weren't enough states to make this an easy mealy state machine to think about.

The diagram for this FSM can be seen in Figure 1. The way this functions is that any input on states 1 and 2 output a 0. On state 1, if the input is a 0, the output is zero and if the input is a 1 the output is 1. This has the bug in that, due to the input not being passed through until the clock is done, the initial change to 1 causes an output of 1.

To verify the functionality a test bench was created to toggle the reset and input bits. The results of this testbench can be seen in Figure 2. From this we can see that aside from the aforementioned bug, the FSM functions as intended. 0 has been considered not a multiple of 3 as the example testbench results provided had the issue of it not making reset output a 0. So because of the inaccuracies, I opted to keep the reset as a 0 output, and ignore no input as a multiple of 3. The other discrepancy in the lab script is that the count of 6, if the input is 0, the output is 0. This is despite a count of 6 being a multiple of 3.

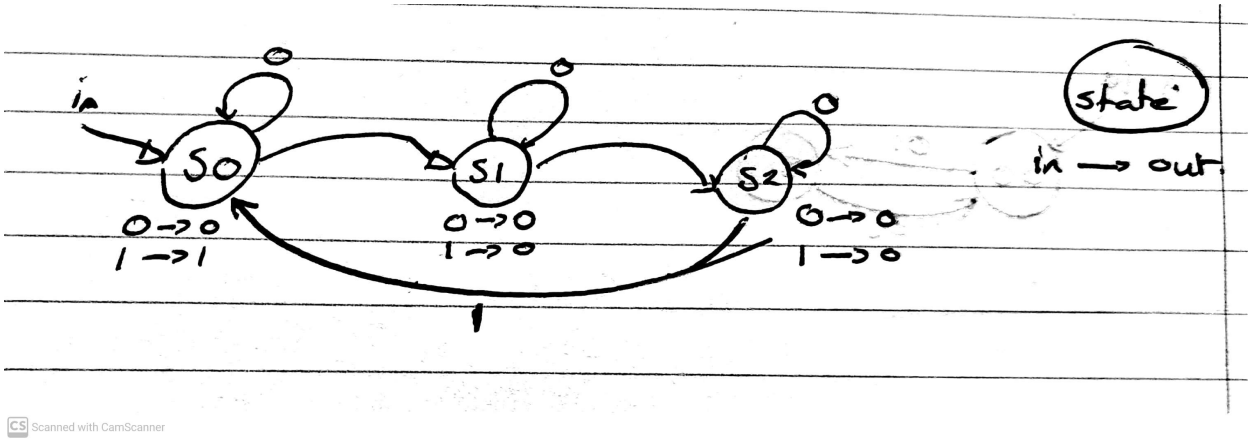

Figure 1: Mod 3 Mealy State Machine Diagram

So, this has been followed in this implementation, hence state 0 causing a 0 output on a 0 input.

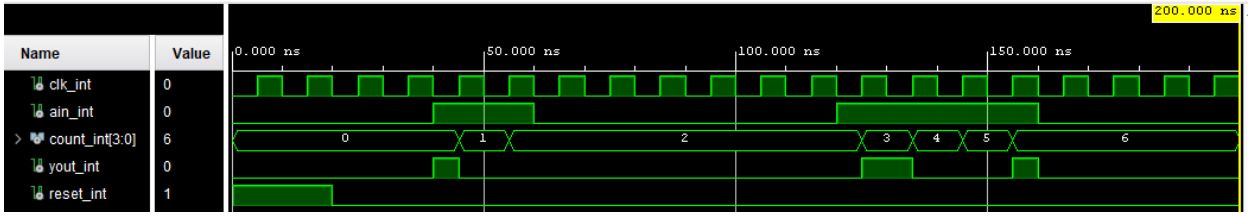

Figure 2: Mod 3 Mealy State Machine Testbench Output

2.3 Moore FSM

The next type of FSM implemented is the moore state machine. On the input of 01 then 00, the output will become 0. On the input of 11 then 00, the output will become 1. On the input of 10 then 00, the output will toggle from it's current state. This state machine was implemented with 9 states, 1 for each case including when it was coming from a specific output route. The states are articulated in Figure 3 and it is laid out as TargetState : Input. This state machine is better implemented as a mealy state machine with the output fed back in as an input as it has events like toggling, which caused the biggest issue when thinking about how to implement this. Also, due to the fact that the input was a sequence of a pair and then 00, it caused issues in that the sequence could change before a 00, which meant that each state needed to be able to go between itself and a different sequence, which became even more complicated due to the toggle state. So to combat this, there is a route for if the output is 0 and if the output is 1. This solved all of the issues, but is much more complicated than if this were a mealy state machine. It is definitely possible to simplify this system, I couldn't think of a simpler way to do it though.

A testbench was created that provided the same input sequence as that provided in the lab script. This resulted in a similar simulation output to tht provided in the lab script, shown in Figure 3. The major difference between this and the script image is the lack of a reset bit. This bit was not included because it wasn't really mentioned as an input into the system in the lab script as the primary input was the two bit sequence. So it was overlooked, and before I had noticed it was near the end of the session and I had already spent so long trying to get the state machines working in the first place.

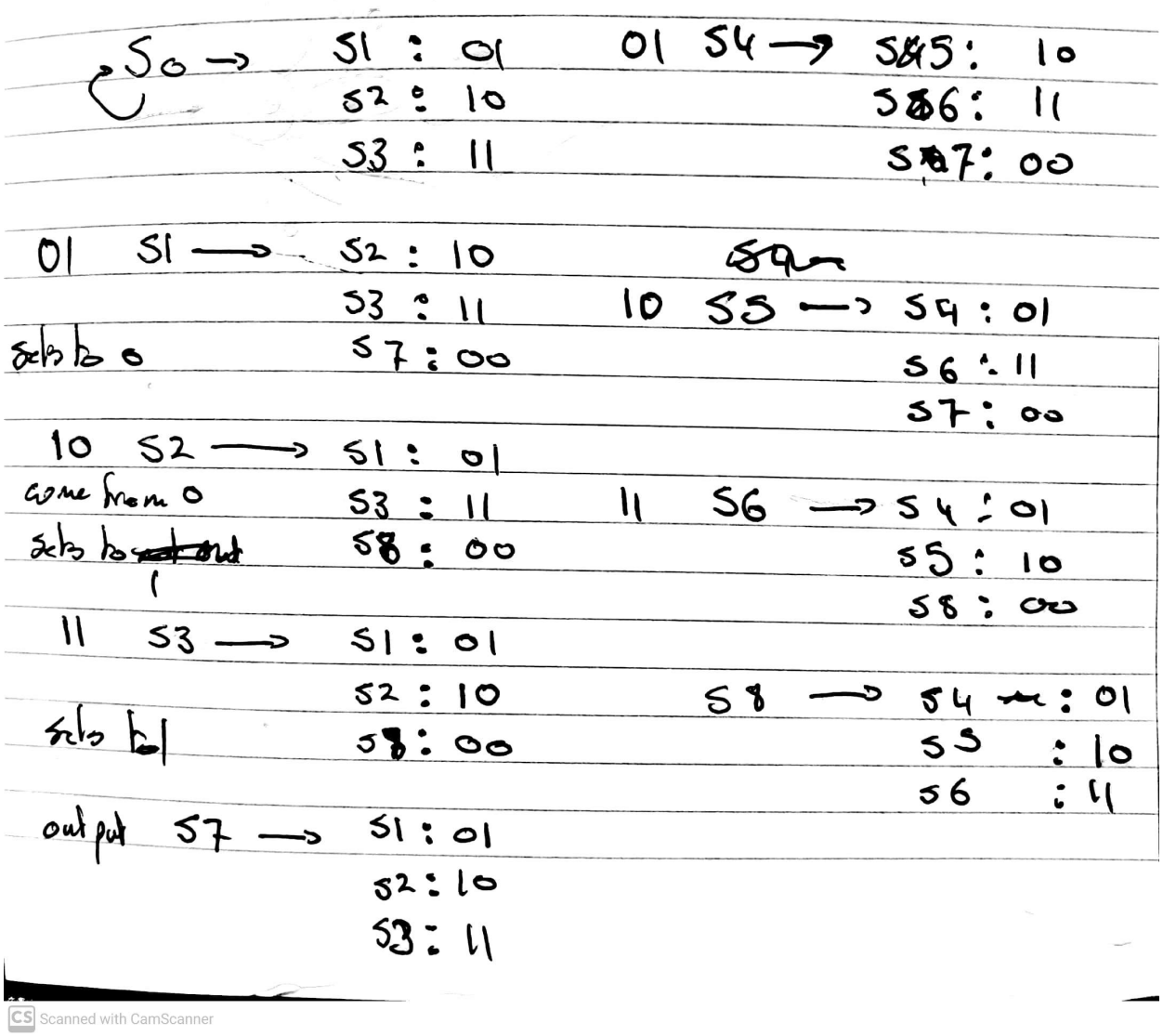

Figure 3: Moore State Machine Diagram

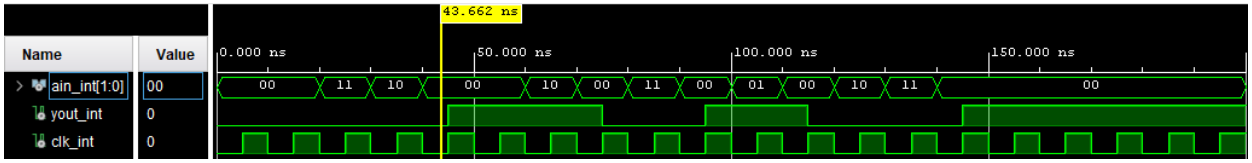

Figure 4: Moore State Machine Testbench Output

2.4 Code

2.4.1 Mealy State Machine

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity Mealy_FSM_Mod_Three is
    Port ( ain : in STD_LOGIC;
           yout : out STD_LOGIC;
           clk : in STD_LOGIC;
           reset : in STD_LOGIC;
```

```
            count : out STD_Logic_vector (3 downto 0) := "0000");
end Mealy_FSM_Mod_Three;

architecture Behavioral of Mealy_FSM_Mod_Three is

type state_type is (S0, S1, S2);
signal state, next_state : state_type; -- Define signals for the states
signal counter : unsigned (3 downto 0) := "0000";

begin

SYNC_PROC : process (clk)
begin
    if rising_edge(clk) then
        if (reset = '1') then
            state <= S0;
            counter <= "0000";
        else
            state <= next_state;
        end if;

        if (ain = '1') then
            counter <= counter + 1;
        end if;
    end if;
end process;

OUTPUT_DECODE : process(state, ain)
begin
    case (state) is
        when S0 =>
            if (ain = '1') then
                yout <= '1';
            else
                yout <= '0';
            end if;
        when others =>
            yout <= '0';
    end case;
end process;

NEXT_STATE_DECODE : process (state, ain)
begin
    next_state <= S0;
    case (state) is
        when S0 =>
            if (ain = '1') then
                next_state <= S1;
            else
                next_state <= S0;
            end if;
        when S1 =>
            if (ain = '1') then
                next_state <= S2;
            else
                next_state <= S1;
            end if;
        when S2 =>
            if (ain = '1') then
```

```
                next_state <= S0;
            else
                next_state <= S2;
            end if;
        end case;
end process;

count <= std_logic_vector (counter);

end Behavioral;
```

code/Mealy_FSM_Mod_Three.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mealy_tb is
--  Port ( );
end Mealy_tb;

architecture Behavioral of Mealy_tb is
    component Mealy_FSM_Mod_Three
    Port ( ain : in STD_LOGIC;
           yout : out STD_LOGIC;
           clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           count : out std_logic_vector (3 downto 0)
         );
    end Component;

    signal clk_int      : std_logic := '0';
    Signal ain_int      : STD_LOGIC := '0';
    signal count_int    : std_logic_vector (3 downto 0) := "0000";
    signal yout_int     : std_logic := '0';
    signal reset_int    : std_logic := '0';
begin
    uut: Mealy_FSM_Mod_Three port map (
        ain => ain_int,
        yout => yout_int,
        clk => clk_int,
        reset => reset_int,
        count => count_int
    );




    process
    begin
        reset_int <= '1';
        wait for 20ns;
        reset_int <= '0';
        wait for 20ns;
        ain_int <= '1';
        wait for 20ns;
        ain_int <= '0';
        wait for 60ns;
        ain_int <= '1';
        wait for 40ns;
        ain_int <= '0';
```

```
48          wait for 40ns;
49      end process;
50
51      clk_int <= not clk_int after 5ns;
52  end Behavioral;
```

code/Mealy_tb.vhd

2.4.2 Moore State Machine

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity MOORE_FSM is
5      Port ( ain : in STD_LOGIC_VECTOR (1 downto 0);
6             yout : out STD_LOGIC;
7             clk : in STD_LOGIC);
8  end MOORE_FSM;
9
10 architecture Behavioral of MOORE_FSM is
11
12 type state_type is (S0, S1, S2, S3, S4, S5, S6, S7, S8);
13 signal state, next_state : state_type;
14 signal output : std_logic := '0';
15
16 begin
17
18     SYNC_PROC : process (clk)
19     begin
20         if rising_edge(clk) then
21             state <= next_state;
22         end if;
23     end process;
24
25     OUTPUT_DECODE : process (state)
26     begin
27         case (state) is
28             when S0 =>
29                 output <= '0';
30             when S7 =>
31                 output <= '0';
32             when S8 =>
33                 output <= '1';
34             when others =>
35                 output <= output;
36         end case;
37      end process;
38
39      yout <= output;
40
41     NEXT_STATE_DECODE : process (state, ain)
42     begin
43         next_state <= S0;
44         case (state) is
45             -- State 0
46             when S0 =>
47                 if (ain = "01") then
48                     next_state <= S1;
49                 elsif (ain = "10") then
50                     next_state <= S2;
```

```
51                  elsif (ain = "11") then
52                      next_state <= S3;
53                  else
54                      next_state <= S0;
55                  end if;
56              -- State 1
57              when S1 =>
58                  if (ain = "01") then
59                      next_state <= S1;
60                  elsif (ain = "10") then
61                      next_state <= S2;
62                  elsif (ain = "11") then
63                      next_state <= S3;
64                  else
65                      next_state <= S7;
66                  end if;
67              -- State 2
68              when S2 =>
69                  if (ain = "01") then
70                      next_state <= S1;
71                  elsif (ain = "10") then
72                      next_state <= S2;
73                  elsif (ain = "11") then
74                      next_state <= S3;
75                  else
76                      next_state <= S8;
77                  end if;
78              -- State 3
79              when S3 =>
80                  if (ain = "01") then
81                      next_state <= S1;
82                  elsif (ain = "10") then
83                      next_state <= S2;
84                  elsif (ain = "11") then
85                      next_state <= S3;
86                  else
87                      next_state <= S8;
88                  end if;
89              -- State 4
90              when S4 =>
91                  if (ain = "01") then
92                      next_state <= S4;
93                  elsif (ain = "10") then
94                      next_state <= S5;
95                  elsif (ain = "11") then
96                      next_state <= S6;
97                  else
98                      next_state <= S7;
99                  end if;
100             -- State 5
101             when S5 =>
102                 if (ain = "01") then
103                     next_state <= S4;
104                 elsif (ain = "10") then
105                     next_state <= S5;
106                 elsif (ain = "11") then
107                     next_state <= S6;
108                 else
109                     next_state <= S7;
```

```
                end if;
            -- State 6
            when S6 =>
                if (ain = "01") then
                    next_state <= S4;
                elsif (ain = "10") then
                    next_state <= S5;
                elsif (ain = "11") then
                    next_state <= S6;
                else
                    next_state <= S8;
                end if;
            -- State 7
            when S7 =>
                if (ain = "01") then
                    next_state <= S1;
                elsif (ain = "10") then
                    next_state <= S2;
                elsif (ain = "11") then
                    next_state <= S3;
                else
                    next_state <= S7;
                end if;
            -- State 8
            when S8 =>
                if (ain = "01") then
                    next_state <= S4;
                elsif (ain = "10") then
                    next_state <= S5;
                elsif (ain = "11") then
                    next_state <= S6;
                else
                    next_state <= S8;
                end if;
             when others =>
                next_state <= S0;
        end case;
    end process;

end Behavioral;
```

code/MOORE_FSM.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MOORE_TB is
--  Port ( );
end MOORE_TB;

architecture Behavioral of MOORE_TB is

    component MOORE_FSM
    Port ( ain : in STD_LOGIC_VECTOR (1 downto 0);
           yout : out STD_LOGIC;
           clk : in STD_LOGIC);
    end component;

    signal ain_int  : std_logic_vector (1 downto 0) := "00";
    signal yout_int : std_logic := '0';
```

```
18      signal clk_int  : std_logic := '0';
19
20  begin
21      uut: MOORE_FSM PORT MAP (
22          ain => ain_int,
23          yout => yout_int,
24          clk => clk_int
25      );
26
27      clk_int <= not clk_int after 5ns;
28
29      process
30      begin
31          wait for 20ns;
32          ain_int <= "11";
33          wait for 10ns;
34          ain_int <="10";
35          wait for 10ns;
36          ain_int <= "00";
37          wait for 20ns;
38          ain_int <= "10";
39          wait for 10ns;
40          ain_int <= "00";
41          wait for 10ns;
42          ain_int <= "11";
43          wait for 10ns;
44          ain_int <= "00";
45          wait for 10ns;
46          ain_int <= "01";
47          wait for 10ns;
48          ain_int <= "00";
49          wait for 10ns;
50          ain_int <= "10";
51          wait for 10ns;
52          ain_int <= "11";
53          wait for 10ns;
54          ain_int <= "00";
55          wait for 60ns;
56
57      end process;
58
59  end Behavioral;
```

code/MOORE_TB.vhd