ECEN302 : Integrated Digital Electronics
Lab 4 Submission

Daniel Eisen : 300447549

September 11, 2020

1 Objectives

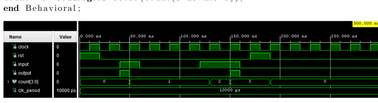
The purpose of these lab exercises was to explore and utilise Finite State Machine (FSM) modelling in implementing a solution in HDL. Specifically it aims to solve to distinct problems with each a Mealy and a Moore FSM, exploring the differences in design, implementation and function/behaviours; both advantages and downsides.

2 Methodology

2.1 Mealy

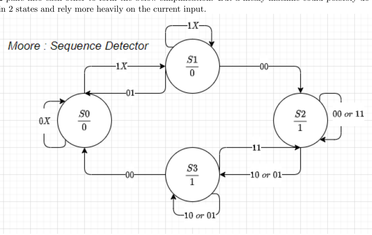
Mealy State Machines change their output based on their current input and present state, rather than just the present state. This means that the output can be asynchronous to the clock as the input can be external and independent from the state change clock. Due the lack of absolute reliance of state-based output they can generally have less states. However, less states doesn't always mean simpler to implement.
This was shown with these exercises as each tasks was better suited to the other kind of FSM.

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

entity FSM_Mealy1 is
    Port (  ain : in STD_LOGIC;
            clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            yout : out STD_LOGIC;
            countv: out std_logic_vector(3 downto 0));
end FSM_Mealy1;

architecture Behavioral of FSM_Mealy1 is

type state_type is (S0, S1, S2);
signal state, next_state : state_type;
signal count : unsigned(3 downto 0) := (others => '0');

begin

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

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

NEXT_STATE_DECODE : process (state, ain)
begin
    next_state <= state;
    case (state) is
        when S0 =>
            if (ain = '1') then
                next_state <= S1;
            end if;
        when S1 =>
            if (ain = '1') then
                next_state <= S2;
            end if;
        when S2 =>
            if (ain = '1') then
                next_state <= S0;
            end if;
    end case;
end process;
countv <= std_logic_vector(count(3 downto 0));
end Behavioral;
```

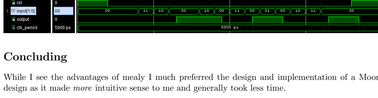


2.2 Moore

Moore machines only change states on the clock edge, and because of this predictability they can be safer to use, simply to design but can generally lead to more states and indeed real world circuitry to implementing. Due to this it is possible for a Mealy machine to operate faster, but you loose the synchronicity.
In this case I initially design the diagram with 6 states but after spending some time could collapse 2 pairs into each other to form the below simplification. But a mealy machine could possibly do it in 2 states and rely more heavily on the current input.




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

entity FSM_Moore is
    Port (
        clk : in std_logic;
        reset : in std_logic;
        ain : in std_logic_vector (1 downto 0);
        yout : out std_logic);
end FSM_Moore;

architecture Behavioral of FSM_Moore is

type state_type is (S0, S1, S2, S3);
signal state, next_state : state_type;

begin


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

NEXT_STATE_DECODE : process (state, ain)
begin
    next_state <= S0;
    case (state) is
        when S0 =>
            if (ain = "10" or ain = "11") then
                next_state <= S1;
            else
                next_state <= state;
            end if;
        when S1 =>
            if (ain = "00") then
                next_state <= S2;
            elsif (ain = "01") then
                next_state <= S0;
            else
                next_state <= state;
            end if;
        when S2 =>
            if (ain = "10" or ain = "01") then
                next_state <= S3;
            else
                next_state <= state;
            end if;
        when S3 =>
            if (ain = "11") then
                next_state <= S2;
            elsif (ain = "00") then
                next_state <= S0;
            else
                next_state <= state;
            end if;
    end case;
end process;

OUTPUT_DECODE : process (state)
begin
    case (state) is
        when S0 =>
            yout <= '0';
        when S1 =>
            yout <= '0';
        when S2 =>
            yout <= '1';
        when S3 =>
            yout <= '1';
        end case;
end process;

end Behavioral;
```



Concluding

While I see the advantages of mealy I much preferred the design and implementation of a Moore design as it made *more* intuitive sense to me and generally took less time.