Sequential Circuits

I. INTRODUCTION :

The output of a combinational circuit depends only on the inputs of circuits. This means combinational circuits do not have any memory elements. Sometimes a circuit's next value depends on it's past value. This means we need to store the previous value to figure out the next value. For example, a sequence detector for 100 needs to have previous information stored. In this section, Clocked Synchronous State Machine analysis will be done. There are two different FSMs(finite state machines):

- Moore Machine: If output is only function of a state
- Mealy Machine: If output is function of state and inputs.

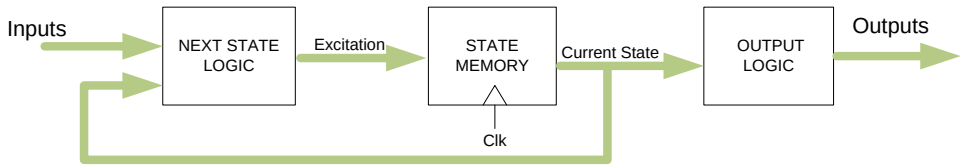


Figure 1 : Moore Machine

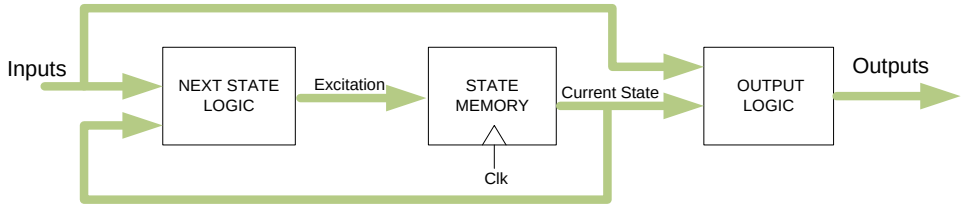


Figure 2 : Mealy Machine

II. BASIC ELEMENTS :

D-Flip flop

One of the main elements in FSM design is D-FF. State memories shown in Figures 1 and 2 can be designed using D-FF or other memory elements such as J-K Flip flops, T-Flip flops, and so on. In this section, the design would be done using D-FF but you may need to use other memory elements as well. D Flip Flops can be designed as follows:

- D-FF (Input D will be transferred to output Q when Clk is positive edge. At all other times the previous value will be held)

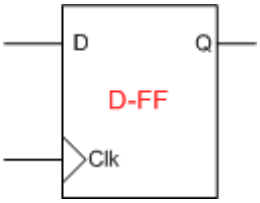


Figure 3.a : D-FF with Positive edge Clk

Clk	D	Q
↑	D	D
0	D	Q_0
1	D	Q_0

Figure 3.b : D-FF with Positive edge Clk Truth Table

VHDL Code and Simulation:

```
library ieee;
use ieee.std_logic_1164.all;
entity DFF is
        port(
        D : in std_logic;
        Clk : in std_logic;
        Q : out std_logic);
end DFF;

architecture behavior of DFF is
begin
  process (Clk)                          -- Change of Clk .
  begin
    if (Clk'event and Clk='1') then      -- Clk event and positive edge. (Change Clk='0' for negative edge)
    Q <= D;
  end if;
end process;
end behavior;
```

TestBench

```
library ieee;
use ieee.std_logic_1164.all;

entity DFF_Test is
end DFF_Test;

architecture behavior_test of DFF_Test is
constant T: time := 100 ns;            -- Constant for clock period
signal D_Test : std_logic;             -- Inputs and outputs are declared as signals
signal Clk : std_logic;
signal Q_Test : std_logic;

begin
  DUT: entity work.DFF                 -- Instantiation of Design Under Test
  port map (D => D_Test, Clk => Clk, Q => Q_Test);

  process                              -- Clock that runs T=100 ns
  begin
    Clk <= '0';
    wait for T/2;
    Clk <= '1';
    wait for T/2;
  end process;

  process                              -- Generation of input vectors.
    D_Test <= '0';                     -- D_Test value Low (0) for 200 ns
  wait for 2*T;
    D_Test <= '1';                     -- D_Test value High (1) for 300 ns
  wait for 3*T;
    D_Test <= '0';                     -- D_Test value Low (0) for rest of time
  end process;

end behavior_test;
```

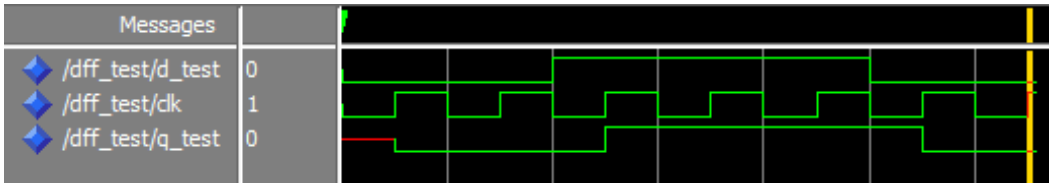


Figure 4 : Simulation Results

- D-FF with Asynchronous Reset (Input D will be transferred to output Q when Clk is positive edge. At all other times, the previous value will be held. Reset can reset the Q to zero at any time independently from Clk)

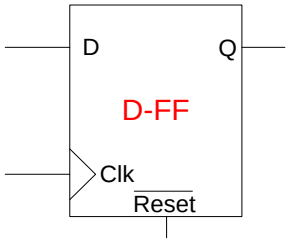


Figure 5.a : D-FF with Asynchronous Reset

Clk	D	Reset	Q
---	---	0	0
↑	D	1	D
0	D	1	Q_0
1	D	1	Q_0

Figure 5.b : Truth Table

VHDL Code and Simulation:

```
library ieee;
use ieee.std_logic_1164.all;
entity DFF_Reset is
        port(
                D : in std_logic;
                Clk : in std_logic;
   Reset : in std_logic;
   Q : out std_logic);
end DFF_Reset;

architecture behavior of DFF_Reset is
begin
  process (Clk,Reset)                    -- Change of Clk and Reset .
  begin
    if (Reset ='0') then
      Q <= '0';
    elsif (Clk'event and Clk='1') then   -- Clk event and positive edge. (Change Clk='0' for negative edge)
    Q <= D;
  end if;
end process;
end behavior;
```

TestBench

```
library ieee;
use ieee.std_logic_1164.all;

entity DFF_Reset_Test is
end DFF_Reset_Test;

architecture behavior_test of DFF_Reset_Test is
constant T: time := 100 ns;            -- Constant for clock period
signal D_Test : std_logic;             -- Inputs and outputs are declared as signals
signal Clk : std_logic;
signal Reset_Test : std_logic;
signal Q_Test : std_logic;
begin

  DUT: entity work.DFF_Reset           -- Instantiation of Design Under Test
  port map (D => D_Test, Clk => Clk, Reset => Reset_Test, Q => Q_Test);

  process
```

```
 begin                                  -- Clock that runs T=100 ns
   Clk <= '0';
   wait for T/2;
   Clk <= '1';
   wait for T/2;
 end process;

 process
 begin                                  -- Generation of input vectors.
   Reset_Test <= '0';
   D_Test <= '0';
 wait for 2*T;
   D_Test <= '1';
 wait for T;
   Reset_Test <= '1';
 wait for 2*T;
   D_Test <= '0';
 wait for 3*T;
   D_Test <= '1';
 end process;

end behavior_test;
```

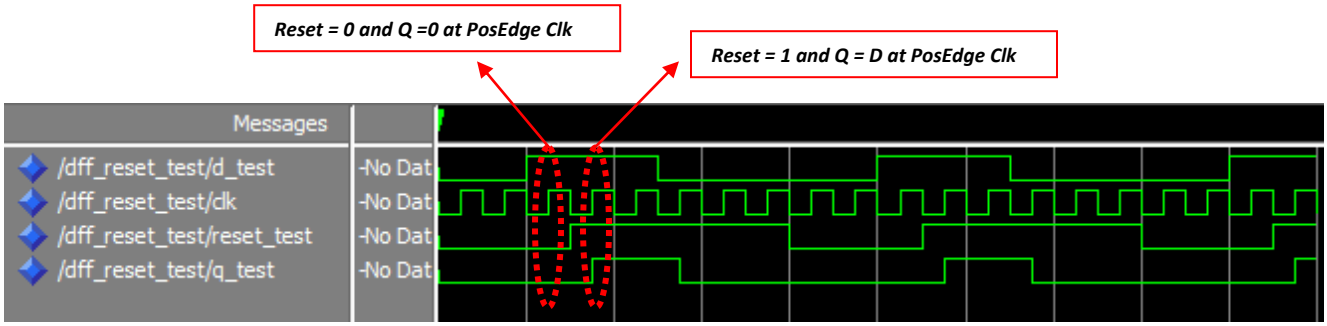
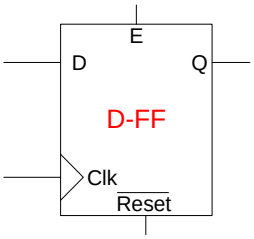


Figure 6 : Simulation Results

- D-FF with Asynchronous Reset and Synchronous Enable (Input D will be transferred to output Q when Clk is positive edge. At all other times, the previous value will be held. Reset can reset the Q to zero at any time independently from Clk and D-FF will be enabled when E=1 is at positive Clk edge).



Clk	D	E	Reset	Q
---	---	---	0	0
↑	D	1	1	D
↑	D	0	1	Q_0
0	D	---	1	Q_0
1	D	---	1	Q_0

Figure 7.a : D-FF with Asynchronous Reset

Figure 7.b : Truth Table

VHDL Code and Simulation:

```
library ieee;
use ieee.std_logic_1164.all;
entity DFF_R_E is
port(
 D : in std_logic;
 Clk : in std_logic;
```

```
 Reset : in std_logic;
 Enable: in std_logic;
 Q : out std_logic);
end DFF_R_E;

architecture behavior of DFF_R_E is
begin
  process (Clk,Reset)                          -- Change of Clk and Reset  (Enable is not is process ).
  begin
    if (Reset ='0') then
      Q <= '0';
    elsif (Clk'event and Clk='1') then         -- Clk event and positive edge. (Change Clk=’0’ for negative edge)
      if (Enable ='1') then                    -- During posedge clock whenever Enable = 1  , Q = D
      Q <= D;
  end if;                                      -- During posedge clock whenever Enable =0  , Q = Q0
end if;
end process;
end behavior;
```

TestBench

```
library ieee;
use ieee.std_logic_1164.all;

entity DFF_R_E_Test is
end DFF_R_E_Test;

architecture behavior_test of DFF_R_E_Test is
constant T: time := 100 ns;                    -- Constant for clock period
signal D_Test : std_logic;                     -- Inputs and outputs are declared as signals
signal Clk : std_logic;
signal Reset_Test : std_logic;
signal Enable_Test : std_logic;
signal Q_Test : std_logic;
begin

  DUT: entity work.DFF_R_E
   port map (D => D_Test, Clk => Clk, Reset => Reset_Test, Enable => Enable_Test, Q => Q_Test);

  process
  begin                                        -- Constant for clock period
    Clk <= '0';
    wait for T/2;
    Clk <= '1';
    wait for T/2;
  end process;

  process
  begin                                        -- Clock that runs T=100 ns
    Reset_Test <= '0';                         -- Generation of input vectors.
    D_Test <= '0';
    Enable_Test <= '0';
  wait for 2*T;
    D_Test <= '1';
  wait for T;
    Reset_Test <= '1';
  wait for 2*T;
    D_Test <= '0';
  wait for 3*T;
    D_Test <= '1';
    Enable_Test <= '1';
  wait for 2*T;
    D_Test <= '0';
  wait for 2*T;
    D_Test <= '1';
  wait for 6*T;
```

end process;

end behavior_test;

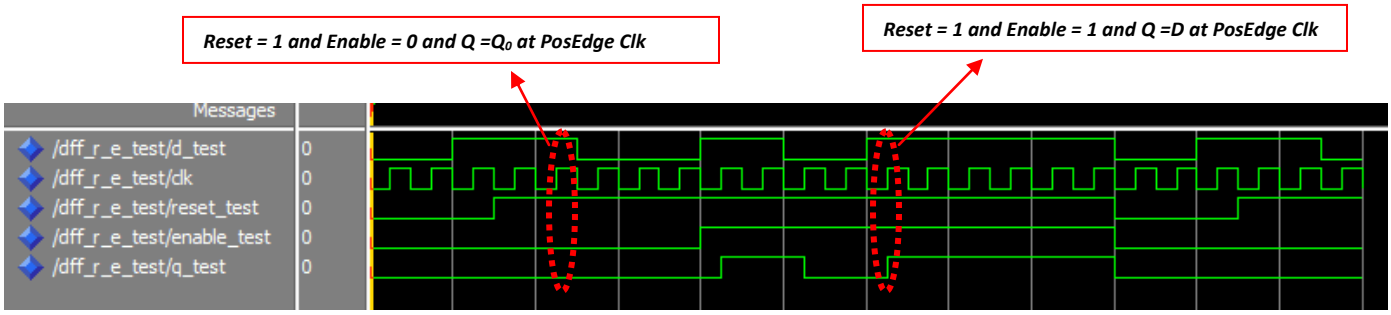


Figure 8 : Simulation Results

Finite State Machine (FSM)

A state machine is specified by State Diagrams. As described in the beginning, FSM can be a Moore or Mealy machine. A state diagram is made of nodes with a transition arrow between the nodes. Nodes represent states and transition arrows represent logic expressions. The arrow direction represents the transaction from current state to the next state and this will happen when the transaction arrow logic expression is true. Figures 9.1 and 9.2 show node and transaction arrows of Mealy and Moore machines, respectively.

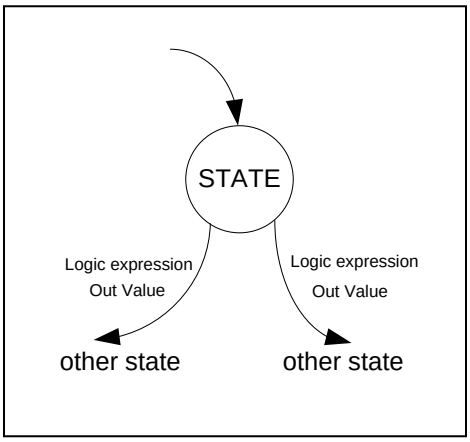


Figure 9.1 : Mealy Machine

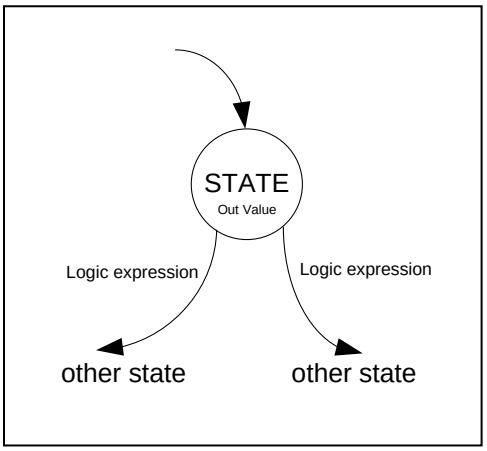


Figure 9.2 : Moore Machine

Device Type	Characteristic Equation
S-R Latch	Q* = S + R'.Q
D Latch	Q* = D
Edge-triggered D flip-flop	Q* = D
D flip-flop with Enable	Q* = EN.D + EN'.Q
Master/slave S-R flip-flop	Q* = S + R'.Q
Master/slave J-K flip-flop	Q* = J.Q' + K'.Q
Edge Triggered J-K flip-flop	Q* = J.Q' + K'.Q
T flip-flop	Q*= Q'
T flip-flop with enable	Q* = EN.Q' + EN'.Q

Figure 10. Latch and flip-flop characteristic equations

State Machine VHDL Code:

Mealy Machine: We mostly use 2 processes

- Modeling the state registers and decide the next state
- Updating the output and next state

Moore Machine: We mostly use 2-3 processes

- Modeling the state registers and decide the next state
- Updating the next state
- Output logic

VHDL CODE EXAMPLE

In this section, we will examine a simple state diagram and its VHDL code. Please check the comments carefully.

a. Moore Machine:

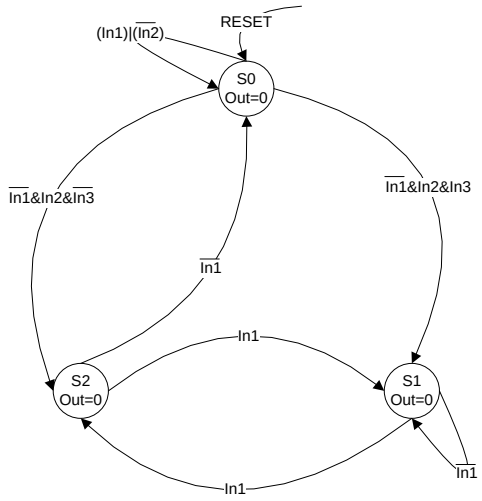


Figure 10.1 : State Diagram

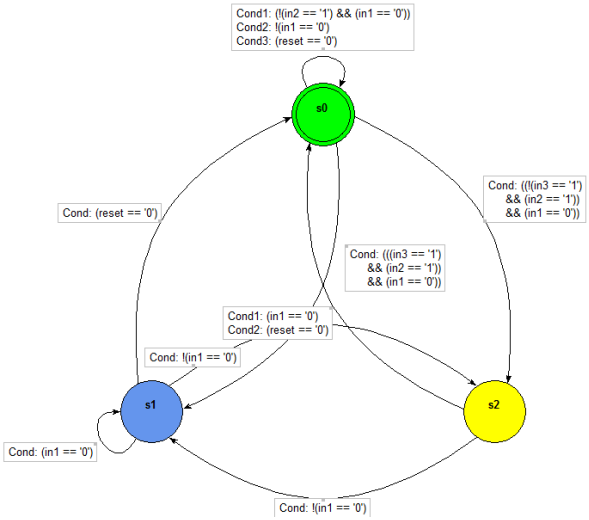


Figure 10.2 : State Diagram (Generated via Modelsim)

```
library ieee;
use ieee.std_logic_1164.all;
entity FSM_MOORE is
        port(
        Reset : in std_logic;
        In1, In2, In3 : in std_logic;
        Clk : in std_logic;
        Out1 : out std_logic);
end FSM_MOORE;

architecture behavior of FSM_MOORE is
type states is (S0, S1, S2);                          -- User defined enumerator "states"
signal present_state, next_state : states;            -- Using enumerator "states" as signal
begin
 process (Clk, Reset)                                 -- Clk and Reset in process ( Asynchronous Reset).
 begin
  if (Reset = '0') then                               -- When Reset = 0 State = S0
  present_state <= S0;
 elsif (Clk'event and Clk='1') then                   -- When Reset = 1 and posedge Clk state assignment
  present_state <= next_state;
 end if;
end process;

process (present_state,In1, In2, In3)                 -- present_state, and inputs are in process
begin
 case present_state is
  when S0 =>
   if In1 = '0' then                                  -- Transition to S1 when (~In1 &In2&In3)
    if In2='1' then
     if In3='1' then
      next_state <= S1;
     else
     next_state <= S2;                                -- Transition to S2 when (~In1 &In2&~In3)
     end if;
    else
     next_state <= S0;                                -- Transition to S0 all others
    end if;
   else
     next_state <= S0;
   end if;
  when S1 =>
   if In1 = '0' then                                  -- When State is S1 stay at S1 when (In1)
     next_state <= S1;
     else
     next_state <= S2;                                -- When State is S1 transition to S2 when (~In1)
```

```
    end if;

    when S2 =>
    if In1 = '0' then
       next_state <= S0;                    -- When State is S2 transition to S0 when (~In1)
       else
       next_state <= S1;                    -- When State is S2 transition to S1 when (In1)
    end if;
  end case;
  end process;

    process (present_state)                 -- Moore machine process for output. Only present_state
      begin                                 -- This process could have been integrated into previous
      case present_state is                 -- process. This could have latching problem.ISE  XST(Xilinx
        when S0 | S1 =>                     -- Synthesis Tool) will give both same result.
          Out1 <= '0';                      -- Out1 is 0  (zero) when states are S0 and S1
        when S2 =>
          Out1 <= '1';                      -- Out1 is 1  (one) when state  S2
      end case;
    end process;
    end behavior;
```

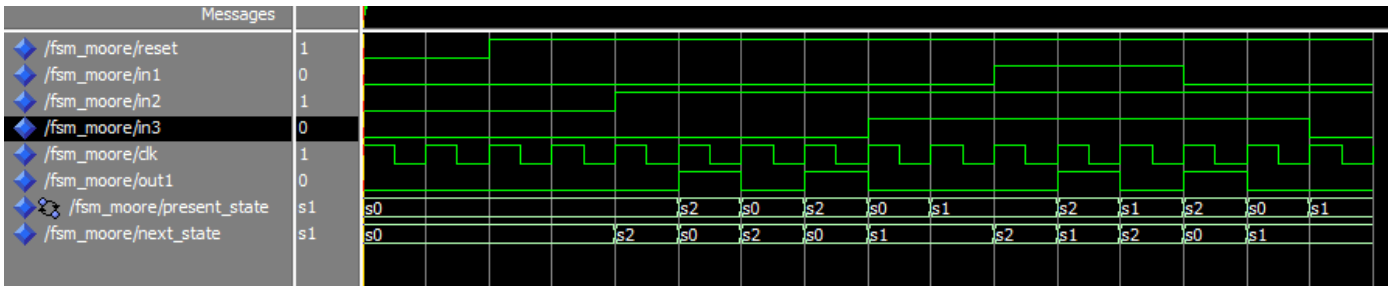


Figure 11 : Simulation Results

b. Mealy Machine:

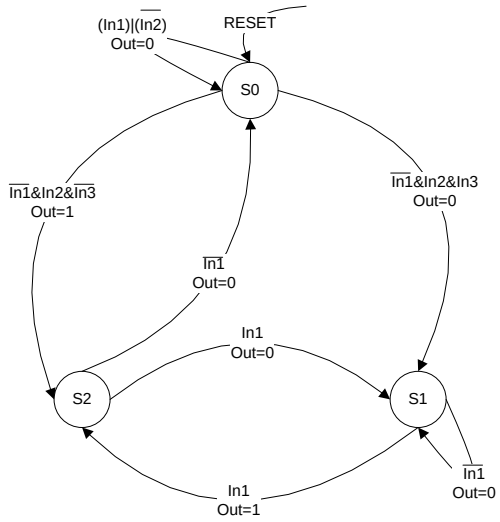


Figure 12.1 : State Diagram

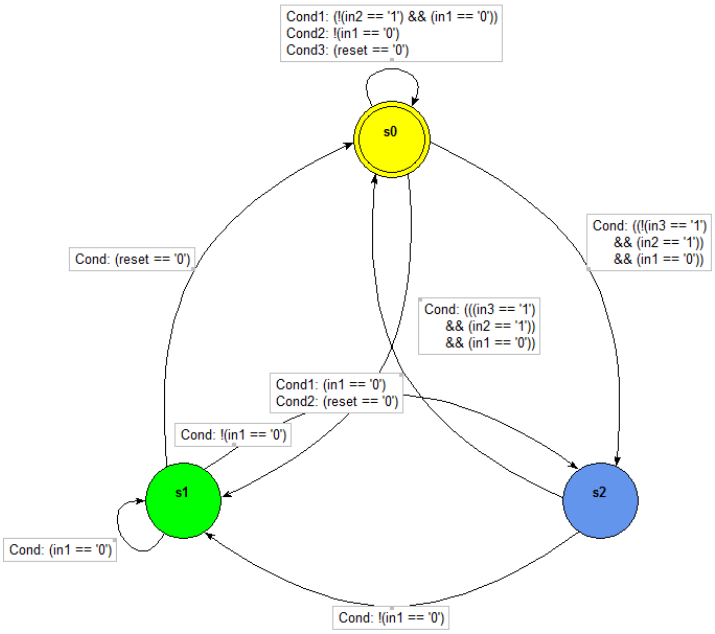


Figure 12.2 : State Diagram (Generated via Modelsim)

```
library ieee;
use ieee.std_logic_1164.all;
entity FSM_MEALY is
        port(
        Reset : in std_logic;
        In1, In2, In3 : in std_logic;
        Clk : in std_logic;
        Out1 : out std_logic);
end FSM_MEALY;

architecture behavior of FSM_MEALY is
type states is (S0, S1, S2);                          -- User defined enumerator "states"
signal present_state, next_state : states;            -- Using enumerator "states" as signal
begin
 process (Clk, Reset)                                 -- Clk and Reset in process ( Asynchronous Reset).
 begin
  if (Reset = '0') then                               -- When Reset = 0 State = S0
  present_state <= S0;
 elsif (Clk'event and Clk='1') then                   -- When Reset = 1 and posedge Clk state assignment
  present_state <= next_state;
 end if;
end process;

process (present_state,In1, In2, In3)                 -- present_state, and inputs are in process
begin
 case present_state is
  when S0 =>
   if In1 = '0' then
    if In2='1' then
     if In3='1' then
      next_state <= S1;                               -- Transition to S1 when (~In1 &In2&In3)
     else
     next_state <= S2;                                -- Transition to S2 when (~In1 &In2&~In3)
     end if;
    else
     next_state <= S0;                                -- Stay at S0 all others
    end if;
    else
     next_state <= S0;
   end if;
```

```
   when S1 =>
    if In1 = '0' then
       next_state <= S1;                    -- When State is S1 stay at S1 when (In1)
       else
       next_state <= S2;                    -- When State is S1 transition to S2 when (~In1)
    end if;

    when S2 =>
    if In1 = '0' then
       next_state <= S0;                    -- When State is S2 transition to S0 when (~In1)
       else
       next_state <= S1;                    -- When State is S2 transition to S1 when (In1)
    end if;
   end case;
   end process;

    process (present_state, In1, In2, In3)  -- Mealy machine process for output.( present_state and inputs are in process)
     begin
     case present_state is
       when S0 =>
        if (In1 ='0' and In2='1' and In3 = '1') then  -- Output depends on state values and inputs. This could have integrated
        Out1 <= '0';                                  -- into previous process.
       else
        Out1 <= '1';
       end if;
       when S1 =>
        if In1 = '0' then
        Out1 <= '0';
       else
        Out1 <= '1';
       end if;
       when S2 =>
       Out1 <= '0';
      end case;
    end process;

end behavior;
```

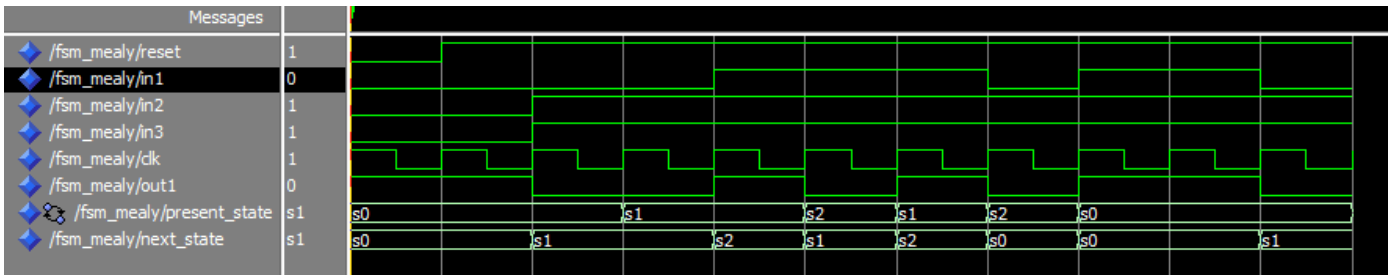


Figure 13 : Simulation Results

DESIGN EXAMPLE (SEQUENCE RECOGNIZER for "101"):

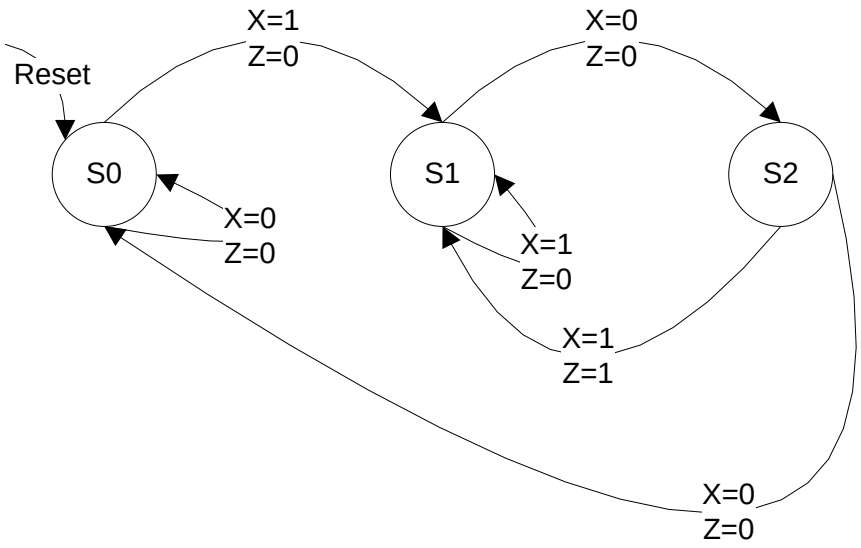


Figure 14 :Mealy Machine State diagram

```
library ieee;
use ieee.std_logic_1164.all;
entity SEQ_MEALY is
        port(
        Reset : in std_logic;
        X : in std_logic;
        Clk : in std_logic;
        Z : out std_logic);
end SEQ_MEALY;

architecture behavior of SEQ_MEALY is
type states is (S0, S1, S2);                    -- User defined enumerator "states"
signal present_state, next_state : states;      --  Using enumerator "states" as signal
begin
  process (Clk, Reset)                          --  Clk and Reset in process ( Asynchronous Reset).
  begin
    if (Reset = '0') then                       --  When Reset = 0 State = S0
    present_state <= S0;
  elsif (Clk'event and Clk='1') then            --  When Reset = 1 and posedge Clk state assignment
    present_state <= next_state;
  end if;
end process;

process (present_state,X)                       --  present_state, and inputs are in process
begin
  case present_state is
    when S0 =>
      if X = '1' then
          next_state <= S1;                     --  Transition to S1 when (~In1 &In2&In3)
        else
        next_state <= S0;                       --  Transition to S2 when (~In1 &In2&~In3)
        end if;
    when S1 =>
      if X = '0' then
```

```
        next_state <= S2;                    -- When State is S1 stay at S1 when (In1)
        else
        next_state <= S1;                    -- When State is S1 transition to S2 when (~In1)
      end if;
        when S2 =>
      if X = '1' then
        next_state <= S1;                    -- When State is S2 transition to S0 when (~In1)
        else
        next_state <= S0;                    -- When State is S2 transition to S1 when (In1)
      end if;
    end case;
    end process;

     process (present_state,X)               -- Mealy machine process for output.( present_state and inputs are in process)
       begin
       case present_state is
         when S0 =>
              Z<='0';
         when S1 =>
              Z<='0';
         when S2 =>
if X = '1' then
Z <= '1';
Else
Z <= '0';
End if;
Out1 <= '0';
       end case;
      end process;
      end behavior;
```

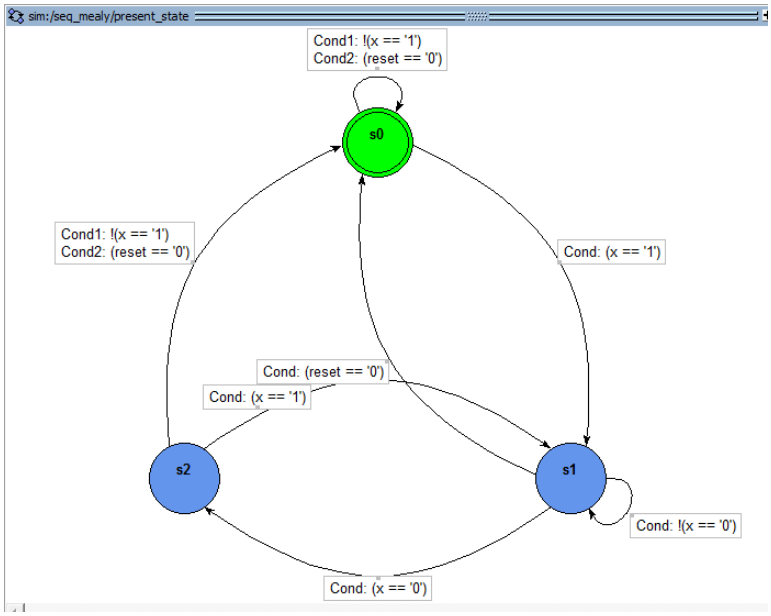


Figure 15 : Modelsim State diagram view

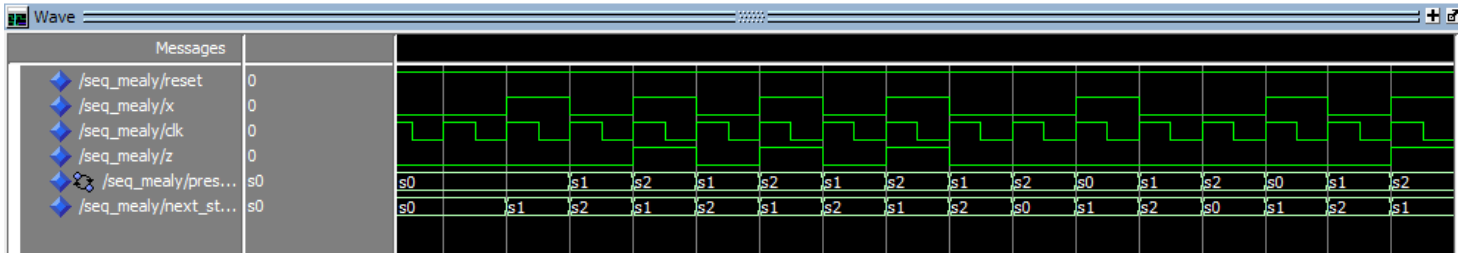


Figure 16 : Simulation Results