Finite State Machines

**Finite State Machines**

**Introduction**

Finite State Machines (FSM) are sequential circuit used in many digital systems to control the behavior of systems and dataflow paths. Examples of FSM include control units and sequencers. This lab introduces the concept of two types of FSMs, Mealy and Moore, and the modelling styles to develop such machines.

**Objectives**

After completing this lab, you will be able to:

* Model Mealy FSMs
* Model Moore FSMs

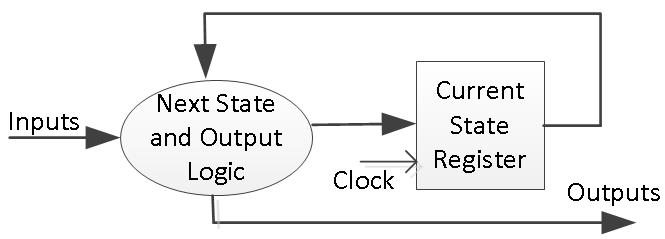
**Mealy FSM** **Part 1**

A finite-state machine (FSM) or simply a state machine is used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of user-defined states. The machine is in only one state at a time; the state it is in at any given time is called the *current state*. It can change from one state to another when initiated by a triggering event or condition; this is called a *transition*. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

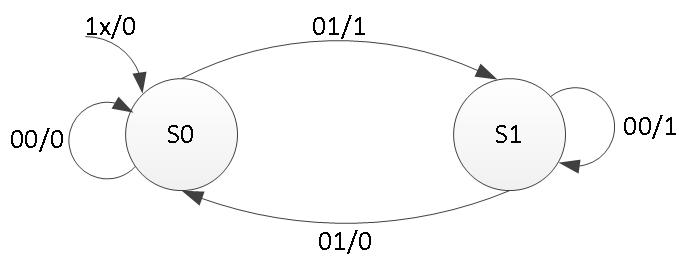
The behavior of state machines can be observed in many devices in modern society performing a predetermined sequence of actions depending on a sequence of events with which they are presented. Simple examples are vending machines which dispense products when the proper combination of coins are deposited, elevators which drop riders off at upper floors before going down, traffic lights which change sequence when cars are waiting, and combination locks which require the input of combination numbers in the proper order.

The state machines are modelled using two basic types of sequential networks- Mealy and Moore. In a Mealy machine, the output depends on both the present (current) state and the present (current) inputs. In Moore machine, the output depends only on the present state.

A general model of a Mealy sequential machine consists of a combinatorial network, which generates the outputs and the next state, and a state register which holds the present state as shown below. The state register is normally modelled as D flip-flops. The state register must be sensitive to a clock edge. The other block(s) can be modelled either using the always procedural block or a mixture of the always procedural block and dataflow modelling statements; the always procedural block will have to be sensitive to all inputs being read into the block and must have all output defined for every branch in order to model it as a combinatorial block. The two blocks Mealy machine can be viewed as



Here are the state diagram of a z checker Mealy machine and the associated model.



type state\_type is (S0, S1);

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, x) begin

z <= '0'; case (state) is

when S0 =>

if (x = '1') then z <= '1'; next\_state <= S1;

else

next\_state <= S0; end if;

when S1 =>

if (x = '1') then next\_state <= S0;

else

z <= '1'; next\_state <= S1;

end if; when others =>

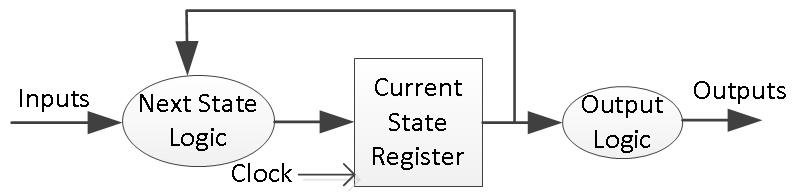
next\_state <= S0; end case;

end process;

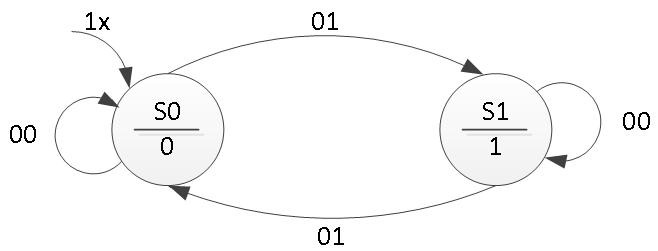
The state assignments can be of one-hot, binary, gray-code, and other types. Usually, the synthesis tool will determine the type of the state assignment, but user can also force a particular type by changing the synthesis property as shown below. The state assignment type will have an impact on the number of bits used in the state register; one-hot encoding using maximum number of bits but decodes very fast to compact (binary) encoding using smallest number of bits but taking longer to decode.

**Moore FSM** **Part 2**

A general model of a Moore sequential machine is shown below. Its output is generated from the state register block. The next state is determined using the present (current) input and the present (current) state. Here the state register is also modelled using D flip-flops. Normally Moore machines are described using three blocks, one of which must be a sequential and the other two can be modelled using always blocks or a combination of always and dataflow modelling constructs.



Here is the state graph of the same z checker to be modelled as a Moore machine. The associate model is shown below.



type state\_type is (S0, S1);

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;

OUTPUT\_DECODE : process (state) begin

case (state) is when S0 =>

z <= '0'; when S1 =>

z <= '1'; when others =>

z <= '0';

end case;

end process;

NEXT\_STATE\_DECODE : process (state, x) begin

next\_state <= S0; case (state) is

when S0 =>

if (x = '1') then next\_state <= S1;

end if; when S1 =>

if (x = '0') then next\_state <= S1;

end if; when others =>

next\_state <= S0;

end case;

end process;