KIIT, Deemed to be University
School of Electronics Engineering
Digital System Design Laboratory [EC 29005]

EXPERIMENT - 8

Aim:

Design and simulation of a finite state machine in Verilog to detect a given sequence of bits.

Component/Software Used:

Component/Software	**Specification**
ICs	-
Bread Board, Power supply, LEDs, Resistors,Switches, Connecting wires	-
Software(s) Used	Vivado 2016.1

Theory:

A finite state machine is a sequential logic circuit that has a finite number of defined states that can be represented. A finite state machine requires the use of memory to store the state of the machine. Combinational logic is used to combine the values of the present state along with inputs to the system to determine the next state of the system.

An example of a simple state machine could be a counter that counts from from 0 to 1 to 2 to 3 and back to 0. In this case, the state machine does not have any input at all, it uses the past state and increments the value every clock cycle. An example of a complex state machine would be a computer. In this case, the computer can have many different inputs and has many different states. Input data can come from the keyboard, network, mouse, memory, etc., while the state would normally be associated with the address in memory of the program being run. In this text, the state machines will be like the counter just described and certainly nothing as complex as a computer. State machines are used in more than just computers. Any process that can be defined with a given predictable algorithm can often be represented by an electronic state machine.

There are two methods to design finite state machines, first is **Mealy** and second is **Moore**. In the Mealy model, the output is a function of **both the present state and the input**.

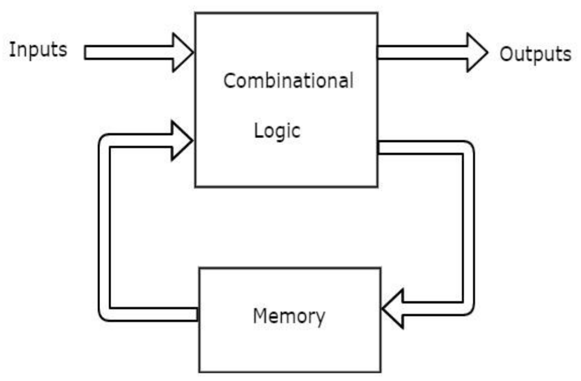

Figure 8.1: Block diagrams of Mealy state machine

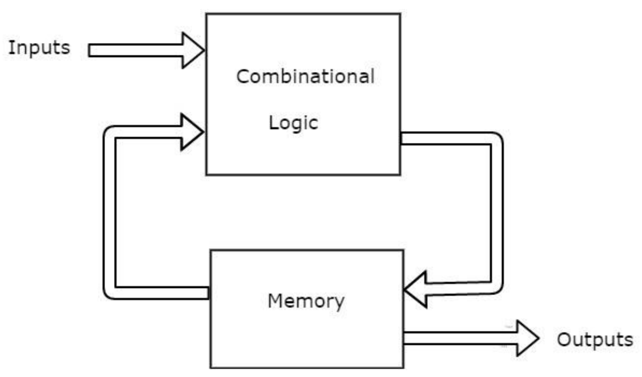

Figure 8.1: Block diagrams of Moore state machine

In a Moore model, the outputs of the sequential circuit are synchronized with the clock, because they depend only on flip-flop outputs that are synchronized with the clock. In a Mealy model, the outputs may change if the inputs change during the clock cycle. Moreover, the outputs may have momentary false values because of the delay encountered from the time that the inputs change and the time that the flip-flop outputs change. In order to synchronize a Mealy-type circuit, the inputs of the sequential circuit must be synchronized with the clock and the outputs must be sampled immediately before the clock edge. The inputs are changed at the inactive edge of the clock to ensure that the inputs to the flip-flops stabilize before the active edge of the clock occurs.

In order to design a state machine one would need to recognize the inputs of the system, the states, and how it transitions from one state to the next. This is graphically represented with a state transition diagram. Then, the transition diagram should be used to create a truth table that has the inputs to the system and current state values as inputs in that table. The output of the truth table is the next state of the system. Combinational logic is used to implement the functions required to obtain the next state values for the state machine. The Boolean logic minimization techniques are used.

Sequence detector is sequential machine that generates an output **1** every time the desired sequence is detected and output **0** at rest all other times. In this experiment a sequence detector for bit sequence ‘1011’ is designed. The input is represented with D_{in} and output is represented with D_{out}. The design of the sequence detector is illustrated below using both the methods.

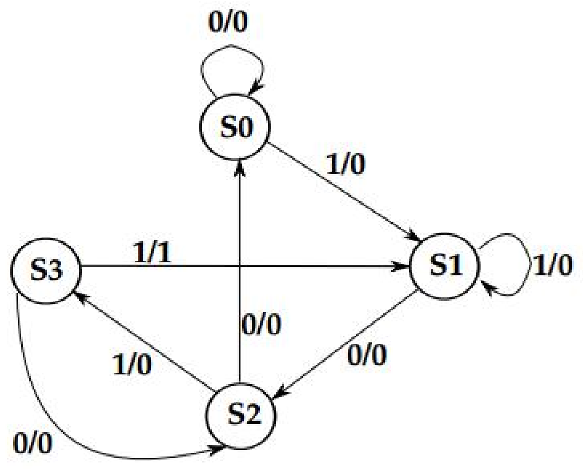

Figure 8.1: **Mealy State Machine for Detecting a Sequence of '1011'**

- When in initial state (S0) the machine gets the input of '1' it jumps to the next state with the output equal to '0'. If the input is '0' it stays in the same state.
- When in 2nd state (S1) the machine gets an input of '0' it jumps to the 3rd state with the output equal to '0'. If it gets an input of '1' it stays in the same state.
- When in the 3rd state (S2) the machine gets an input of '1' it jumps to the 4th state with the output equal to '0'. If the input received is '0' it goes back to the initial state.
- When in the 4th state (S3) the machine gets an input of '1' it jumps back to the 2nd state, with the output equal to '1'. If the input received is '0' it goes back to the 3rd state.

Present State	Next State		Output (D_{out})	
	D_{in} = 0	D_{in} = 1	D_{in} = 0	D_{in} = 1
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S0	S3	0	0
S3	S2	S1	0	1

Table 8.1: **Mealy State table for Detecting a Sequence of '1011'**

As there are no redundant states thus the final states are S0, S1, S2 and S3. There four states therefore two state variables are required. Binary values are assigned to the states arbitrarily.

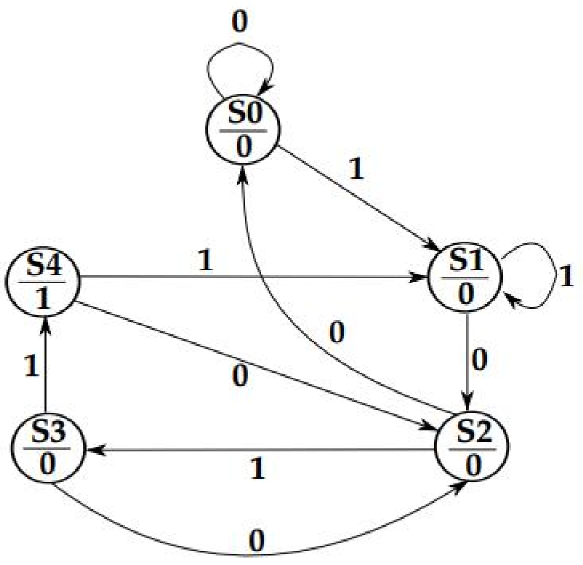

Figure 8.2 : Moore State Machine for Detecting a Sequence of '1011'

- In initial state (S0) the output of the detector is '0'. When machine gets the input of '1' it jumps to the next state. If the input is '0' it stays in the same state.
- In 2nd state (S1) the output of the detector is '0'. When machine gets an input of '0' it jumps to the 3rd state. If it gets an input of '1' it stays in the same state.
- In the 3rd state (S2) the output of the detector is '0'. When machine gets an input of '1' it jumps to the 4th state. If the input received is '0' it goes back to the initial state.
- In the 4th state (S3) the output of the detector is '0'. When machine gets an input of '1' it jumps to the 5th state. If the input received is '0' it goes back to the 3rd state.
- In the 5th state the output of the detector is '1'. When machine gets an input of '0'it jumps to the 3rd state, otherwise it jumps to the 2nd state.

Present State	Next State		Output (D_{out})
	$D_{in} = 0$	$D_{in} = 1$	
S0	S0	S1	0
S1	S2	S1	0
S2	S0	S3	0
S3	S2	S4	0
S4	S2	S1	1

Table 8.2: **Moore State table for Detecting a Sequence of '1011'**

As there are no redundant states thus the final states are S0, S1, S2, S3 and S4. There five states therefore three state variables are required. Binary values are assigned to the states arbitrarily.

After designing the state machines the models have to be transformed into Verilog code describing it.

```
// Design source of Sequence detector of sequence “1011”
module Seq_Det (input clk, input rst, input Din, output Dout);
reg  [1:0] state;
reg  Dout;
initial
begin
state <= 2'b00;
end
always @ ( posedge clk, rst )
begin
if ( rst )
state <= 2'b00;
else
begin
case( {state,Din} )
3'b000: begin
state <= 2'b00;
end
3'b001: begin
state <= 2'b01;
end
3'b010: begin
state <= 2'b10;
end
3'b011: begin
state <= 2'b01;
end
3'b100: begin
state <= 2'b10;
end
3'b101: begin
state <= 2'b11;
end
3'b110: begin
state <= 2'b10;
end
3'b111: begin
state <= 2'b01;
end
endcase
end
assign Dout = (({state,Din}) == 3'b111) ? 1'b1 : 1'b0 ;
end
endmodule
```

Procedure

a) Create a module with required number of variables and mention it's input / output.

b) Write the behabioural description of sequence detector circuit.

c) Synthesize to create RTL Schematic.

d) Create another module referred as test bench to verify the functionality and to obtain the waveform of input and output.

e) Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table.

f) Take the screenshots of the RTL schematic and simulated waveforms.

Observation:

To be written by students

Conclusion:

To be written by students.

Sample viva-voice questions

1. Design a FSM to detect the sequence '1010'.
2. Design a state flow diagram for the sequence detector FSM '10010'.
3. Design the state table for the sequence detector FSM '10010'.
4. Design a FSM to detect the sequence '1011'.