Basically a FSM consists of combinational, sequential and output logic. Combinational logic is used to decide the next state of the FSM, sequential logic is used to store the current state of the FSM. The output logic is a mixture of both combo and seq logic as shown in the figure below.

**Types of State Machines**

There are many ways to code these state machines, but before we get into the coding styles, let's first understand the basics a bit. There are two types of state machines:

* Mealy State Machine : Its output depends on current state and current inputs. In the above picture, the blue dotted line makes the circuit a mealy state machine.
* Moore State Machine : Its output depends on current state only. In the above picture, when blue dotted line is removed the circuit becomes a Moore state machine.

Combinational always blocks are always blocks that are used to code combinational logic functionality and are strictly coded using blocking assignments. A combinational always block has a combinational sensitivity list, a sensitivity list without "posedge" or "negedge" Verilog keywords.

Sequential always blocks are always blocks that are used to code clocked or sequential logic and are always coded using nonblocking assignments. A sequential always block has an edge-based sensitivy list.



Fig: Mealy & Moore FSMs

Encoding Style

Since we need to represent the state machine in a digital circuit, we need to represent each state in one of the following ways:

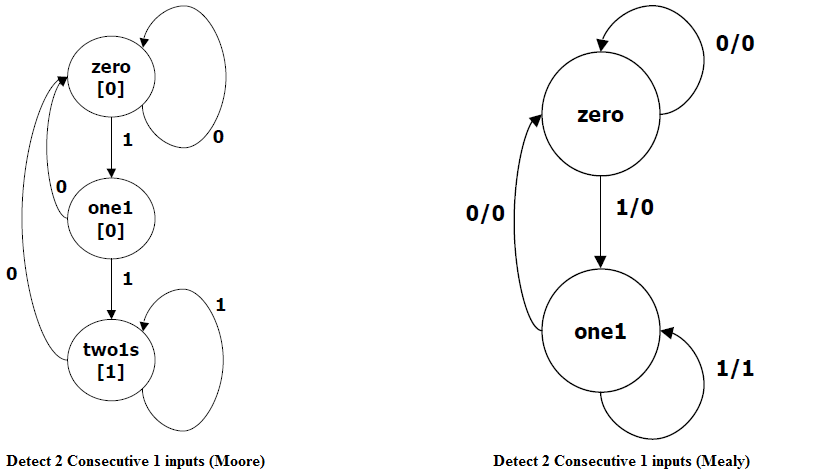
Binary encoding : each state is represented in binary code (i.e. 000, 001, 010....)

Gray encoding : each state is represented in gray code (i.e. 000, 001, 011,...)

One Hot : only one bit is high and the rest are low (i.e. 0001, 0010, 0100, 1000)

One Cold : only one bit is low, the rest are high (i.e. 1110,1101,1011,0111)

State diagram for Moore and Mealy machine



**General View of State Machine:**

module FSM (CLK, in, out);

input CLK;

input in;

output out;

reg out;

// state variable

reg [1:0] state;

// local variable

reg [1:0] next\_state;

always @(posedge CLK) // registers

state = next\_state;

always @(state or in)

// Compute next-state and output logic whenever state or inputs change.

// (i.e. put equations here for next\_state[1:0])

// Make sure every local variable has an assignment in this block!

endmodule

**Verilog coding for moore FSM**

`define zero 2'b00

`define one1 2'b01

`define two1s 2'b10

module moore\_fsm (CLK, reset, in, out);

input CLK, reset, in;

output out;

reg out;

reg [1:0] state; // state variables

reg [1:0] next\_state;

always @(posedge CLK)

if (reset) state = `zero;

else state = next\_state;

always @(in or state)

case (state)

`zero: // last input was a zero

begin

if (in) next\_state = `one1;

else next\_state = `zero;

end

`one1: // we've seen one 1

begin

if (in) next\_state = `two1s;

else next\_state = `zero;

end

`two1s: // we've seen at least 2 ones

begin

if (in) next\_state = `two1s;

else next\_state = `zero;

end

endcase

always @(state)

case (state)

`zero: out = 0;

`one1: out = 0;

`two1s: out = 1;

endcase

endmodule

**Verilog coding for Mealy FSM**

`define zero 2'b00

`define one1 2'b01

`define two1s 2'b10

module mealy\_fsm (clk, reset, in, out);

input clk, reset, in;

output out;

reg out;

reg state; // state variables

always @(posedge clk)

if (reset) state = `zero;

else

case (state)

`zero: // last input was a zero

begin

out = 0;

if (in) state = `one;

else state = `zero;

end

`one: // we've seen one 1

if (in) begin

state = `one; out = 1;

end else begin

state = `zero; out = 0;

end

endcase

endmodule

Verilog testbench program for Moore FSM

module moore\_fsm\_tb;

reg CLK, reset, in;

wire out;

moore\_fsm mooreFSM(.CLK(CLK), .reset(reset), .in(in), .out(out));

initial

begin

CLK = 0;

reset = 1;

in = 0;

#10 reset = 0;

#100 $finish;

end

always

#5 CLK = ! CLK;

always

#5 in = $random;

initial begin

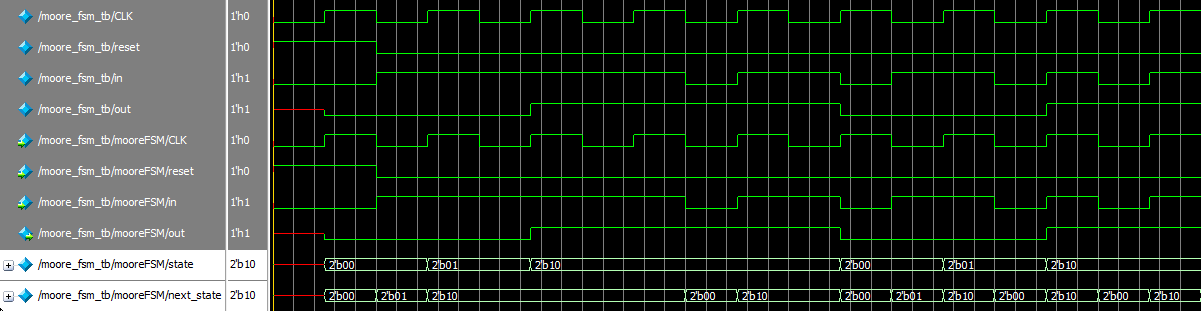
$display("\t\ttime,\tCLK,\treset,\tin,\tout");

$monitor("%d,\t%b,\t%b,\t%b,\t%b",$time, CLK,reset,in,out);

end

endmodule

**Expected waveform:**



**Verilog Testbench for Mealy Machine:**

module mealy\_fsm\_tb;

reg clk, reset, in;

wire out;

mealy\_fsm mooreFSM(.clk(clk), .reset(reset), .in(in), .out(out));

initial

begin

clk = 0;

reset = 1;

in = 0;

#10 reset = 0;

#100 $finish;

end

always

#5 clk = ! clk;

always

#5 in = $random;

initial begin

$display("\t\ttime,\tclk,\treset,\tin,\tout");

$monitor("%d,\t%b,\t%b,\t%b,\t%b",$time, clk,reset,in,out);

end

endmodule

**Expected waveform:**

