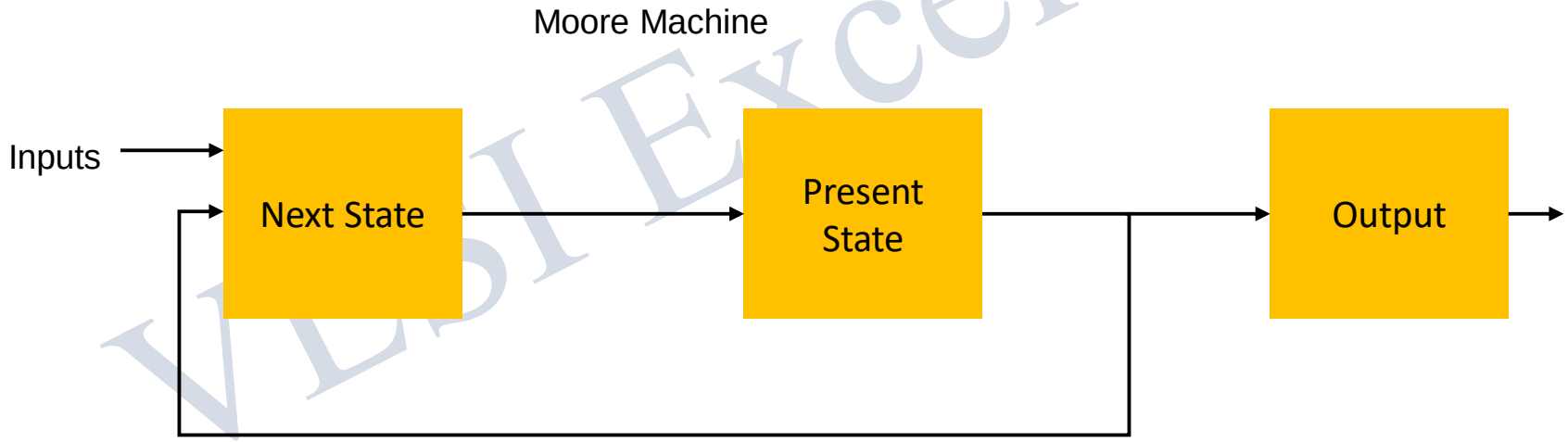
Module #13 : Verilog HDL 1001 Sequence Detector

13.1: Moore FSM:

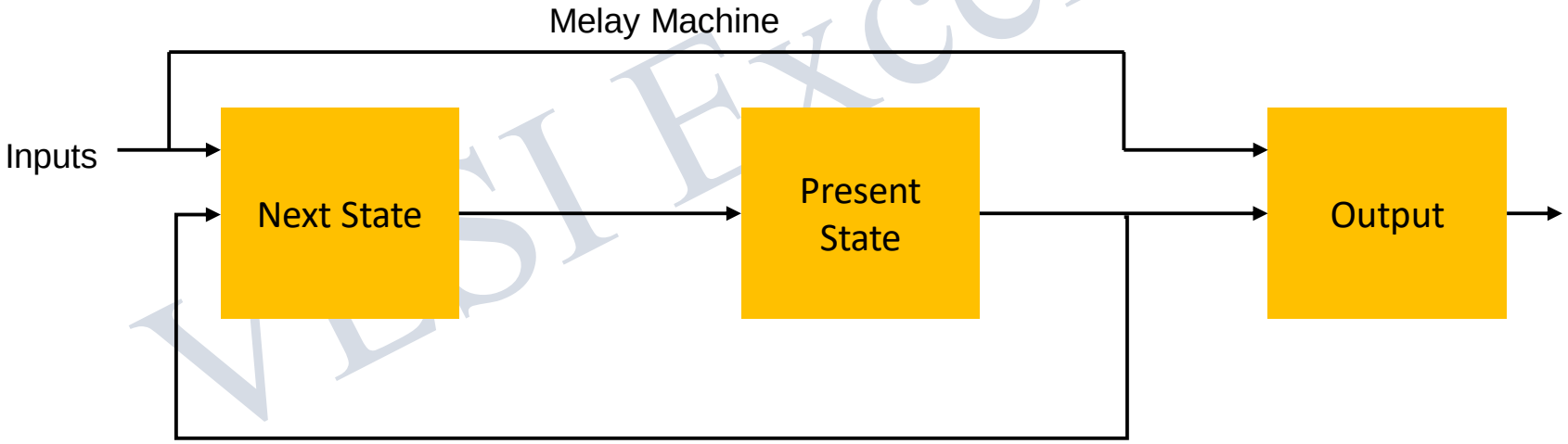
- In Moore FSM, output depends only on the present state
- Output is less prone to glitch because of registered present state
- More number of states are required
- They react slower to inputs (One clock cycle later)



Module #13 : Verilog HDL 1001 Sequence Detector

13.2: Mealy FSM:

- In Mealy FSM, output depends on the present state as well as present input
- Output is more prone to glitch because of asynchronous dependency on present input
- Less number of states are required
- They react faster to inputs



Module #13 : Verilog HDL 1001 Sequence Detector

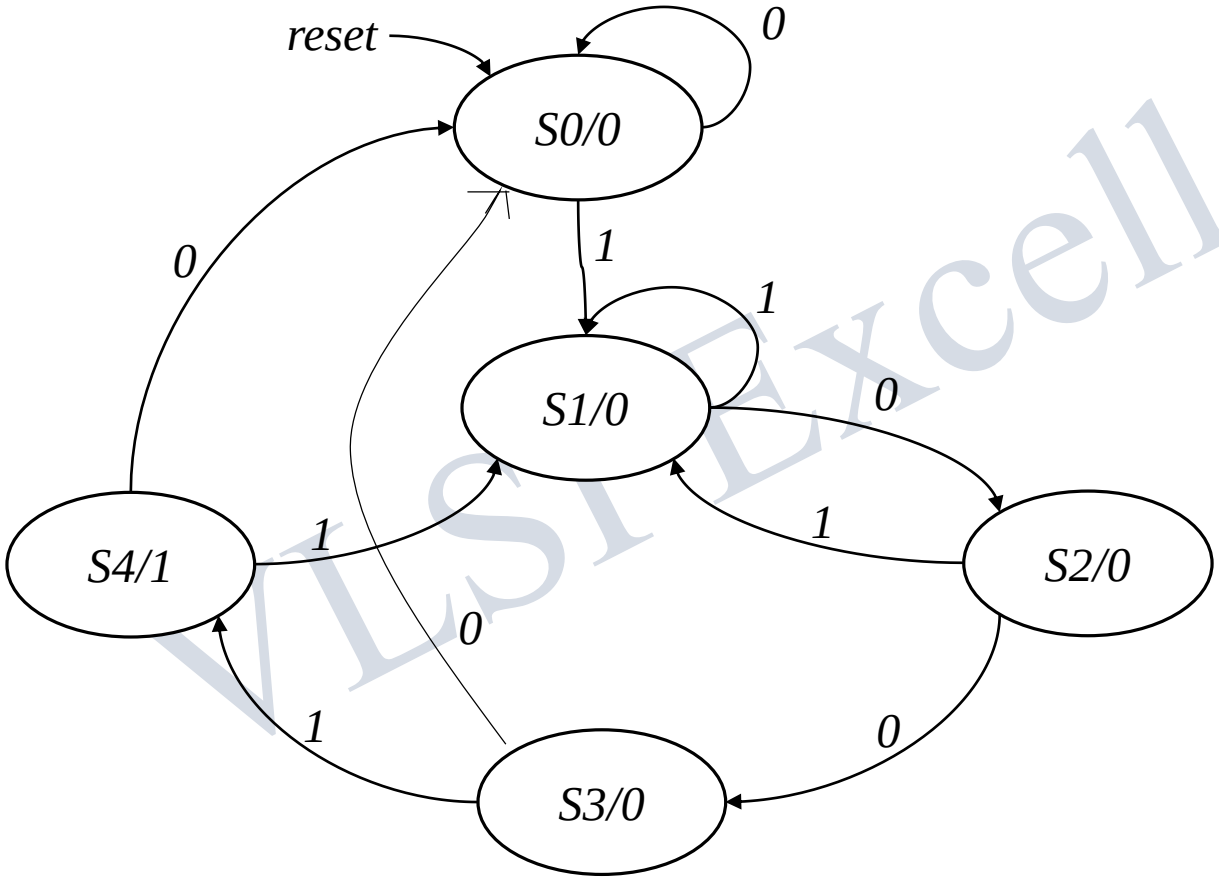
13.3: Verilog FSM Design Techniques :

1) Using a Single Process (Procedural Block) to Code Present State, Next State and Output Logic
2) Using Two Process, One to code Present State and Next State logic and another to code Output Logic,
3) Using Three Process each to code Present State, Next State and Output Logic

Note: When modeling finite state machines, it is recommended to separate the sequential current-state logic from the combinational next-state and output logic.

Module #13 : Verilog HDL 1001 Sequence Detector

13.4: 1001 Sequence Detector *Moore (Non-Overlapping)* FSM Diagram :



S0 – On Reset
S1 – 1 is detected
S2 – 10 is detected
S3 – 100 is detected
S4 – 1001 is detected

Module #13 : Verilog HDL 1001 Sequence Detector

13.5: 1001 Sequence Detector Moore (Non Overlapping) Verilog Code :

Example 13.1:

```
module moore_1001_nonovr (clk, rst, data_i, data_o);
input clk, rst, data_i;
output  data_o; // data_o is declared reg so that it can
reg data_o;      // be assigned in an always block.
parameter S0=0, S1=1, S2=2, S3=3, S4 = 4;
reg [2:0] state, nxt_st;
always @ (state or data_i)
begin : next_state_logic //Name of always procedure.
case (state)
S0: begin
if (data_i) nxt_st = S1;
else nxt_st = S0;
end
S1: begin
if(data_i) nxt_st = S1;
else nxt_st = S2
end
S2: begin
if (data_i) nxt_st = S1;
else nxt_st = S3;
end
S3: begin
if (data_i) nxt_st = S3;
else nxt_st = S0;
end
S4: begin
if (data_i) nxt_st = S1;
else nxt_st = S0;
end
default: nxt_st = S0;
endcase
end
 // default is optional since
 //all 4 cases are covered
//specifically. Good practice
// says uses it
```

Module #13 : Verilog HDL Finite State Machines (FSMs)

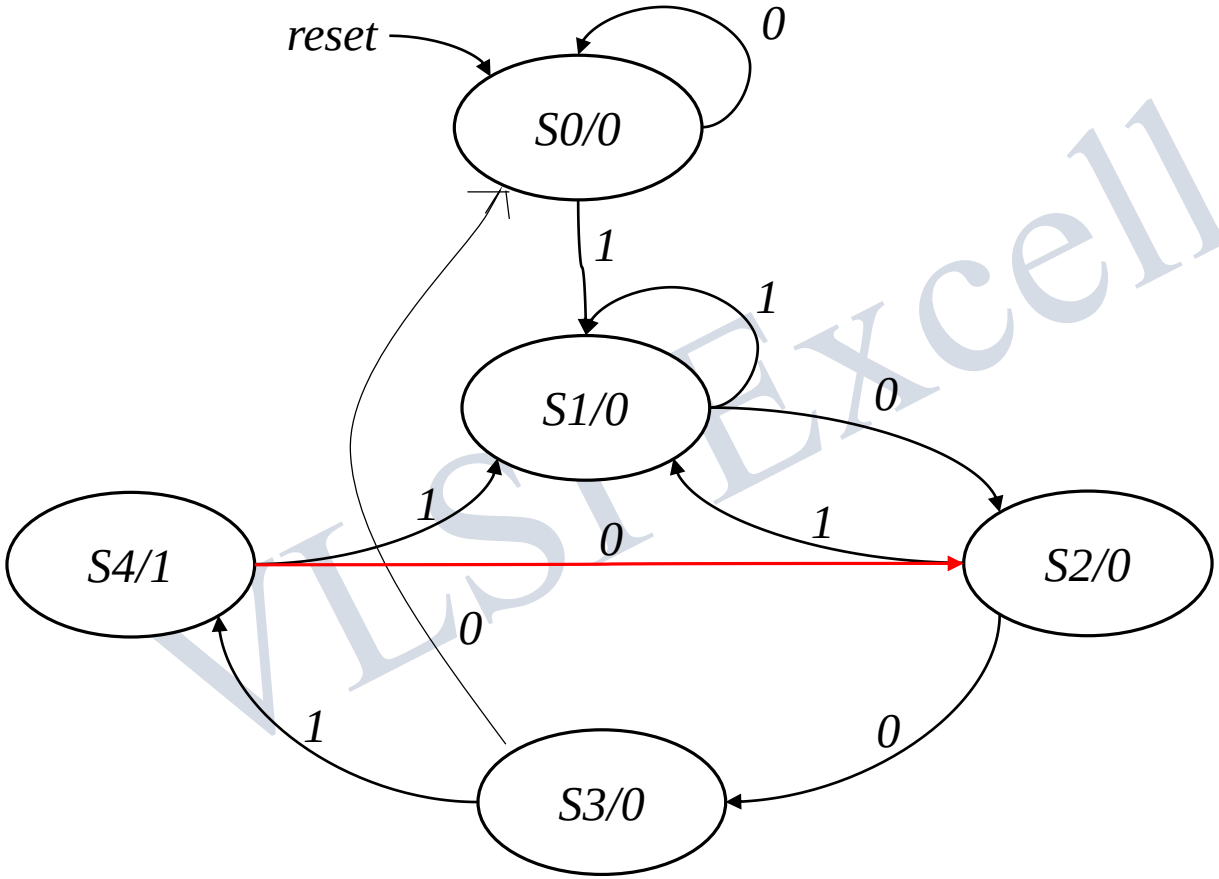
13.5: 1001 Sequence Detector Moore (Non Overlapping) Verilog Code :

```
always @(posedge clk or posedge rst)
begin : register_generation
if (rst) state = S0;
else state = nxt_st;
end
```

```
always @(state) begin : output_logic
case (state)
S0: data_o = 1'b0;
S1: data_o = 1'b0;
S2: data_o= 1'b0;
S3: data_o = 1'b0;
S4: data_o = 1'b1;
default: data_o = 1'b0;  // default avoids latches
endcase
end
endmodule
```

Module #13 : Verilog HDL 1001 Sequence Detector

13.4: 1001 Sequence Detector *Moore (Overlapping)* FSM Diagram :



S0 – On Reset
S1 – 1 is detected
S2 – 10 is detected
S3 – 100 is detected
S4 – 1001 is detected

Module #13 : Verilog HDL 1001 Sequence Detector

13.5: 1001 Sequence Detector Moore (Overlapping) Verilog Code :

Example 13.2:

```
module moore_1001_ovr (clk, rst, data_i, data_o);
input clk, rst, data_i;
output  data_o; // data_o is declared reg so that it can
reg data_o;      // be assigned in an always block.
parameter S0=0, S1=1, S2=2, S3=3, S4 = 4;
reg [2:0] state, nxt_st;
always @ (state or data_i)
begin : next_state_logic //Name of always procedure.
case (state)
S0: begin
if (data_i) nxt_st = S1;
else nxt_st = S0;
end
S1: begin
if(data_i) nxt_st = S1;
else nxt_st = S2
end
S2: begin
if (data_i) nxt_st = S1;
else nxt_st = S3;
end
S3: begin
if (data_i) nxt_st = S3;
else nxt_st = S0;
end
S4: begin
if (data_i) nxt_st = S1;
else nxt_st = S2;
end
default: nxt_st = S0;
endcase
end
 // default is optional since
 //all 4 cases are covered
//specifically. Good practice
// says uses it
```

Module #13 : Verilog HDL Finite State Machines (FSMs)

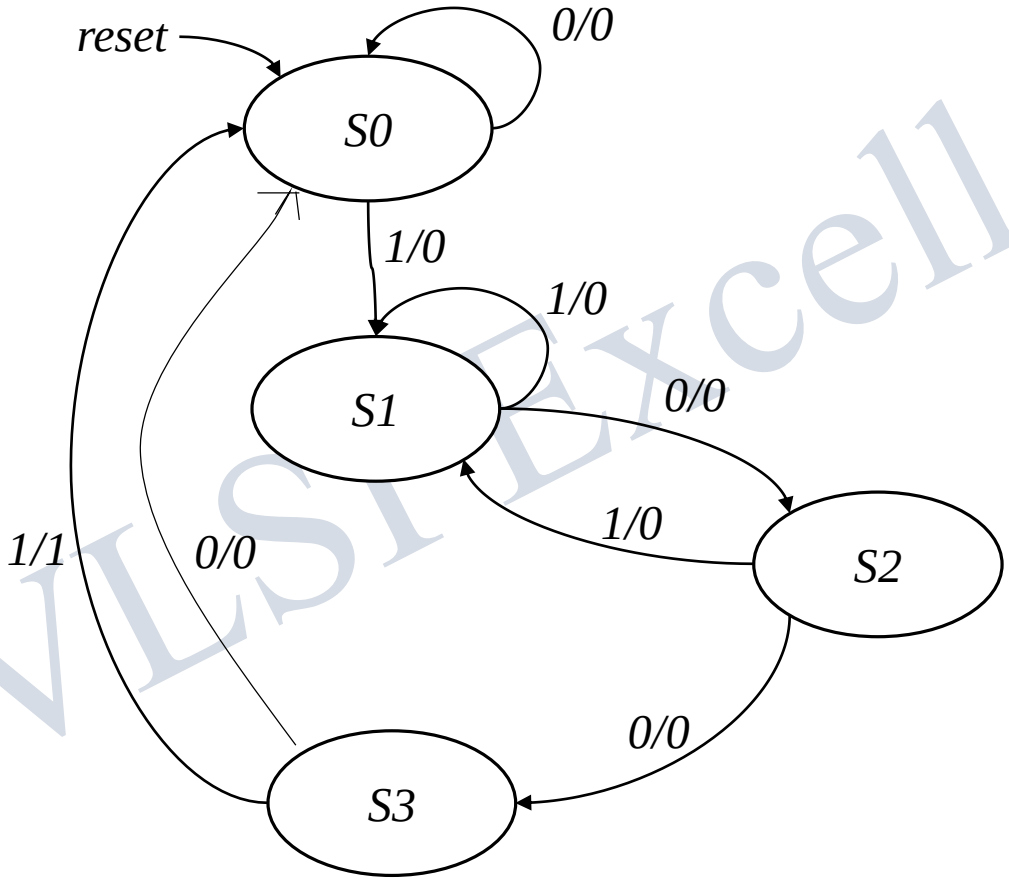
13.5: 1001 Sequence Detector Moore (Overlapping) Verilog Code :

```
always @(posedge clk or posedge rst)
begin : register_generation
if (rst) state = S0;
else state = nxt_st;
end
```

```
always @(state) begin : output_logic
case (state)
S0: data_o = 1'b0;
S1: data_o = 1'b0;
S2: data_o= 1'b0;
S3: data_o = 1'b0;
S4: data_o = 1'b1;
default: data_o = 1'b0;  // default avoids latches
endcase
end
endmodule
```

Module #13 : Verilog HDL 1001 Sequence Detector

13.4: 1001 Sequence Detector *Mealy (Non-Overlapping)* FSM Diagram :



S0 – On Reset
S1 – 1 is detected
S2 – 10 is detected
S3 – 100 is detected

Module #13 : Verilog HDL 1001 Sequence Detector

13.5: 1001 Sequence Detector Mealy (Non Overlapping) Verilog Code :

Example 13.3:

```
module mealy_1001_nonovr (clk, rst, data_i, data_o);
input clk, rst, data_i;
output  data_o; // data_o is declared reg so that it can
reg data_o;      // be assigned in an always block.
parameter S0=0, S1=1, S2=2, S3=3, S4 = 4;
reg [2:0] state, nxt_st;
always @ (state or data_i)
begin : next_state_logic //Name of always procedure.
case (state)
S0: begin
if (data_i) nxt_st = S1;
else nxt_st = S0;
end
S1: begin
if(data_i) nxt_st = S1;
else nxt_st = S2
end
S2: begin
if (data_i) nxt_st = S1;
else nxt_st = S3;
end
S3: begin
if (data_i) nxt_st = S0;
else nxt_st = S0;
end
default: nxt_st = S0;
endcase
end
 // default is optional since
 //all 4 cases are covered
//specifically. Good practice
// says uses it
```

Module #13 : Verilog HDL Finite State Machines (FSMs)

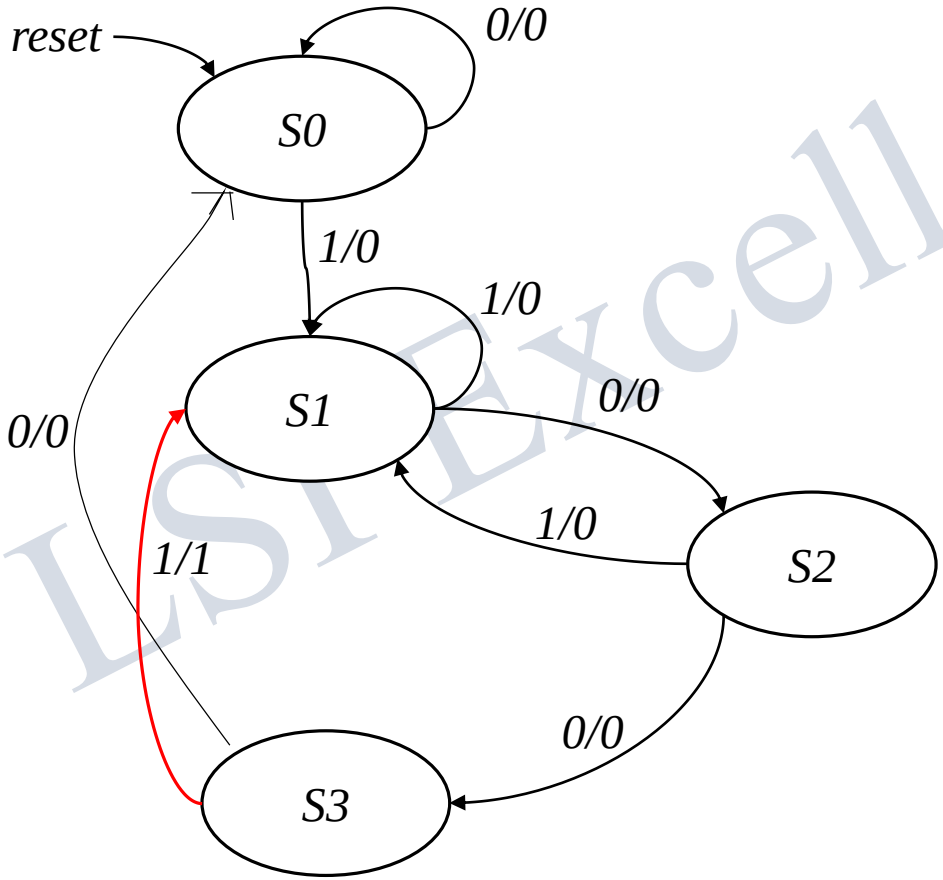
13.5: 1001 Sequence Detector Mealy (Non Overlapping) Verilog Code :

```
always @(posedge clk or posedge rst)
begin : register_generation
if (rst) state = S0;
else state = nxt_st;
end

always @(state or data_i) begin : output_logic
case (state)
S0: data_o = 1'b0;
S1: data_o = 1'b0;
S2: data_o= 1'b0;
S3: begin
        if(data_i) data_o = 1'b1;
        else data_o = 1'b0;
    end
default: data_o = 1'b0;  // default avoids latches
endcase
end
endmodule
```

Module #13 : Verilog HDL 1001 Sequence Detector

13.4: 1001 Sequence Detector *Mealy (Overlapping)* FSM Diagram :



S0 – On Reset
S1 – 1 is detected
S2 – 10 is detected
S3 – 100 is detected

Module #13 : Verilog HDL 1001 Sequence Detector

13.5: 1001 Sequence Detector Mealy (Overlapping) Verilog Code :

Example 13.4:

```
module mealy_1001_ovr (clk, rst, data_i, data_o);
input clk, rst, data_i;
output  data_o; // data_o is declared reg so that it can
reg data_o;      // be assigned in an always block.
parameter S0=0, S1=1, S2=2, S3=3, S4 = 4;
reg [2:0] state, nxt_st;
always @ (state or data_i)
begin : next_state_logic //Name of always procedure.
case (state)
S0: begin
if (data_i) nxt_st = S1;
else nxt_st = S0;
end
S1: begin
if(data_i) nxt_st = S1;
else nxt_st = S2
end
S2: begin
if (data_i) nxt_st = S1;
else nxt_st = S3;
end
S3: begin
if (data_i) nxt_st = S1;
else nxt_st = S0;
end
default: nxt_st = S0;
endcase
end
 // default is optional since
 //all 4 cases are covered
//specifically. Good practice
// says uses it
```

Module #13 : Verilog HDL Finite State Machines (FSMs)

13.5: 1001 Sequence Detector Mealy (Non Overlapping) Verilog Code :

```
always @(posedge clk or posedge rst)
begin : register_generation
if (rst) state = S0;
else state = nxt_st;
end
```

```
always @(state or data_i) begin : output_logic
case (state)
S0: data_o = 1'b0;
S1: data_o = 1'b0;
S2: data_o= 1'b0;
S3: begin
        if(data_i) data_o = 1'b1;
        else data_o = 1'b0;
    end
default: data_o = 1'b0;  // default avoids latches
endcase
end
endmodule
```