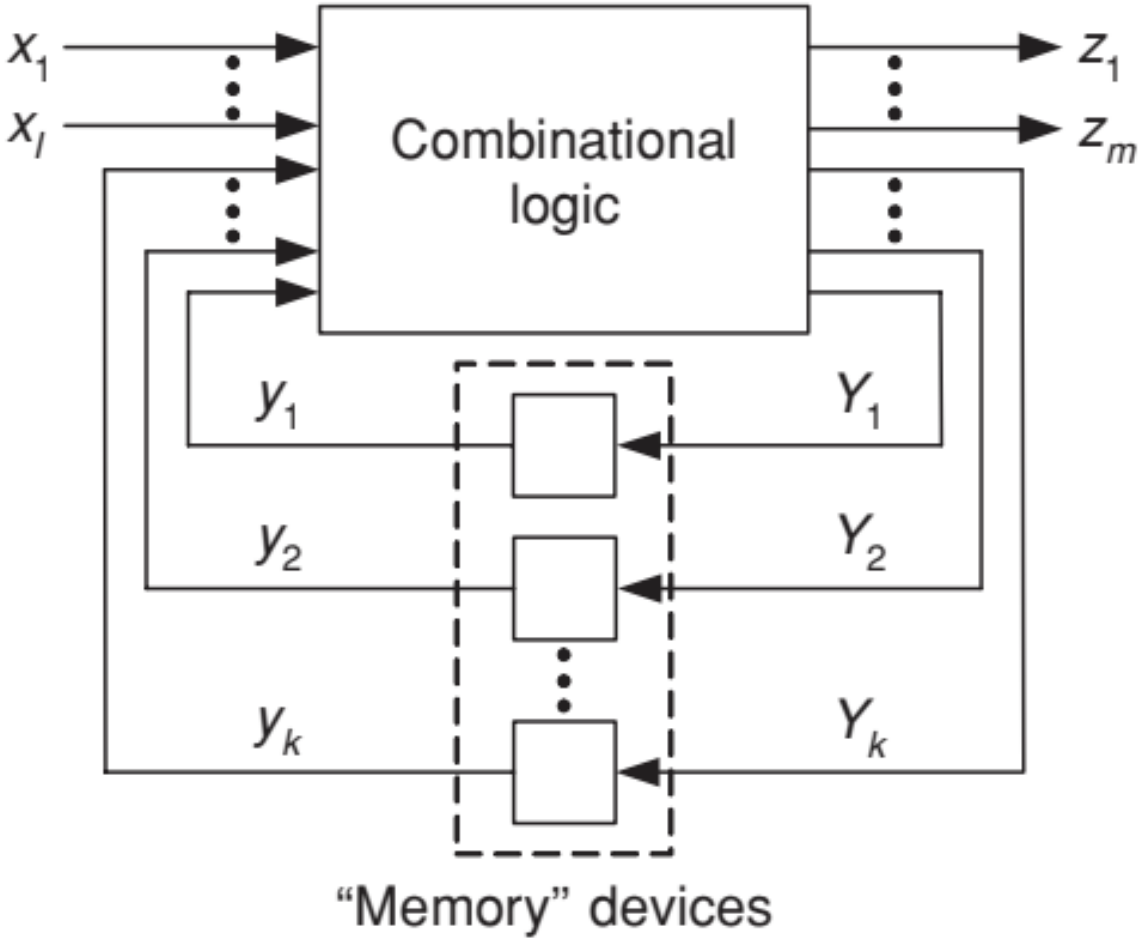
Verilog HDL:
Finite State Machines

Pravin Zode

Outline

- Mealy and Moore Machines
- Implementation Steps
- Exercises

Synchronous Sequential Machines

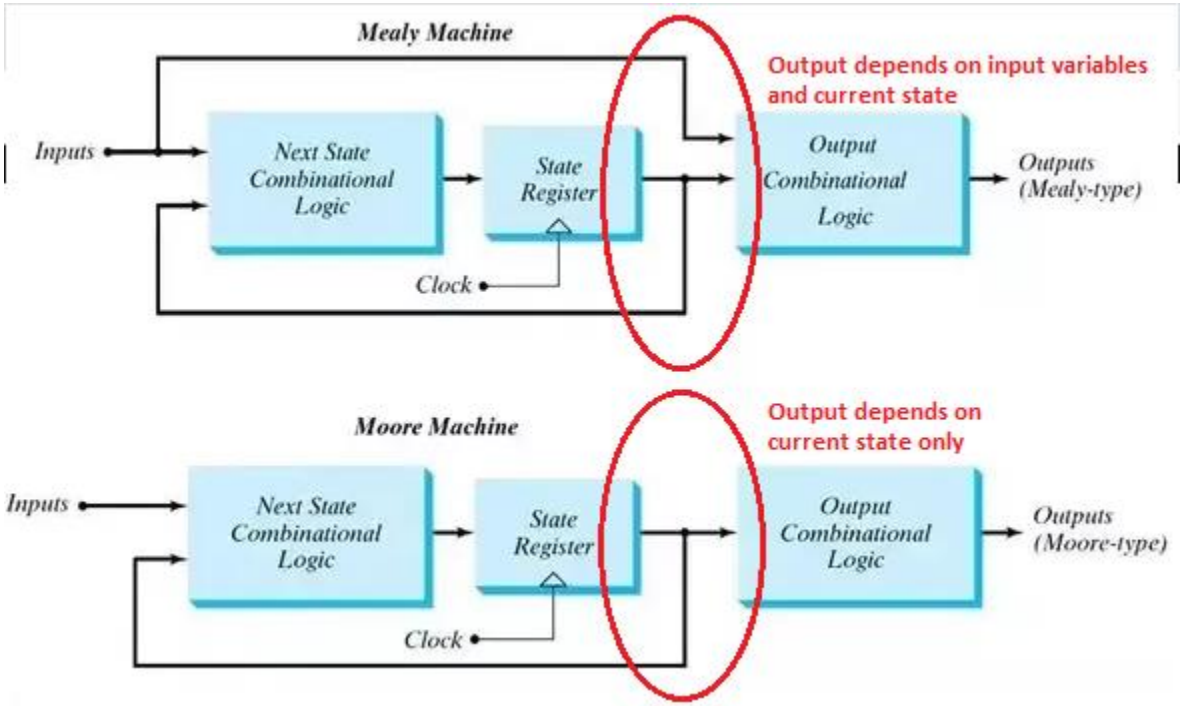


Introduction

- FSM is a sequential circuit that transitions through a finite number of states.
- Behavior is defined by current state and input.
- FSM = Combinational Logic + Memory (Flip-Flops)
- **Applications:** Protocol design, traffic lights, vending machines, control units

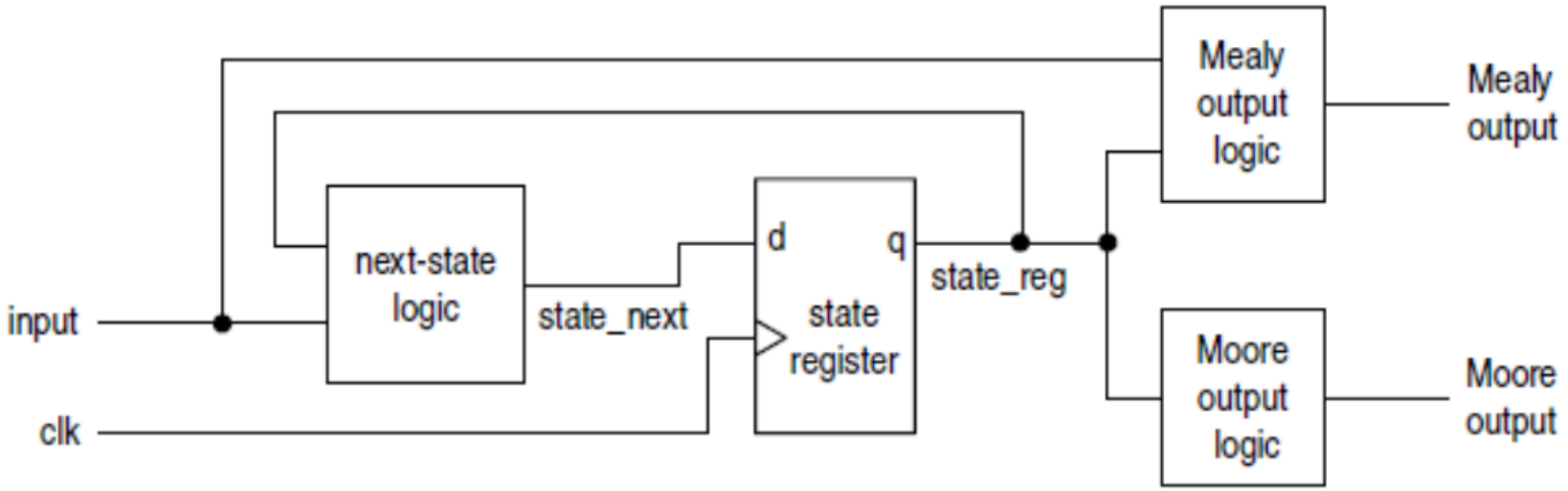
Types of FSM

- Moore Machine: Output depends only on the current state
- Mealy Machine: Output depends on current state and input



Output from State Machine

- Moore Machine: Output depends only on the current state
- Mealy Machine: Output depends on current state and input

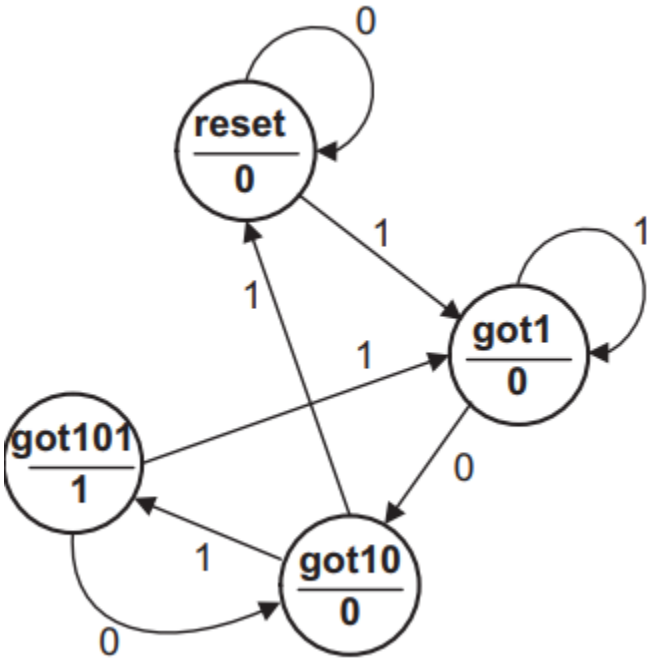


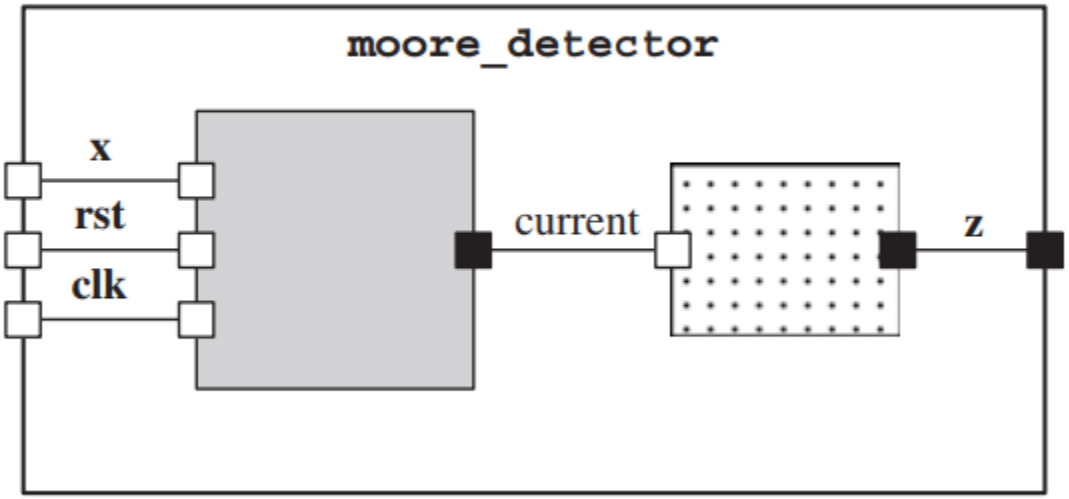
FSM Design Steps

- Define the states
- Draw the state transition diagram
- Create the state transition table
- Encode states (binary/gray/one-hot)
- Write Verilog code
  - State register
  - Next-state logic
  - Output logic
- Test using simulation.

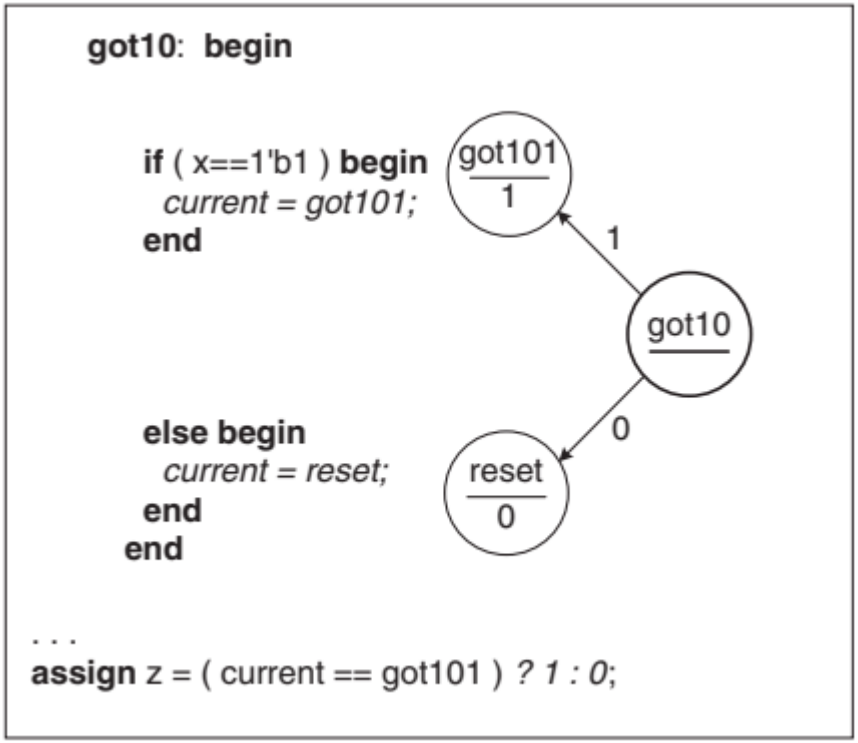
Moor Machine (101 Detector)

- Define



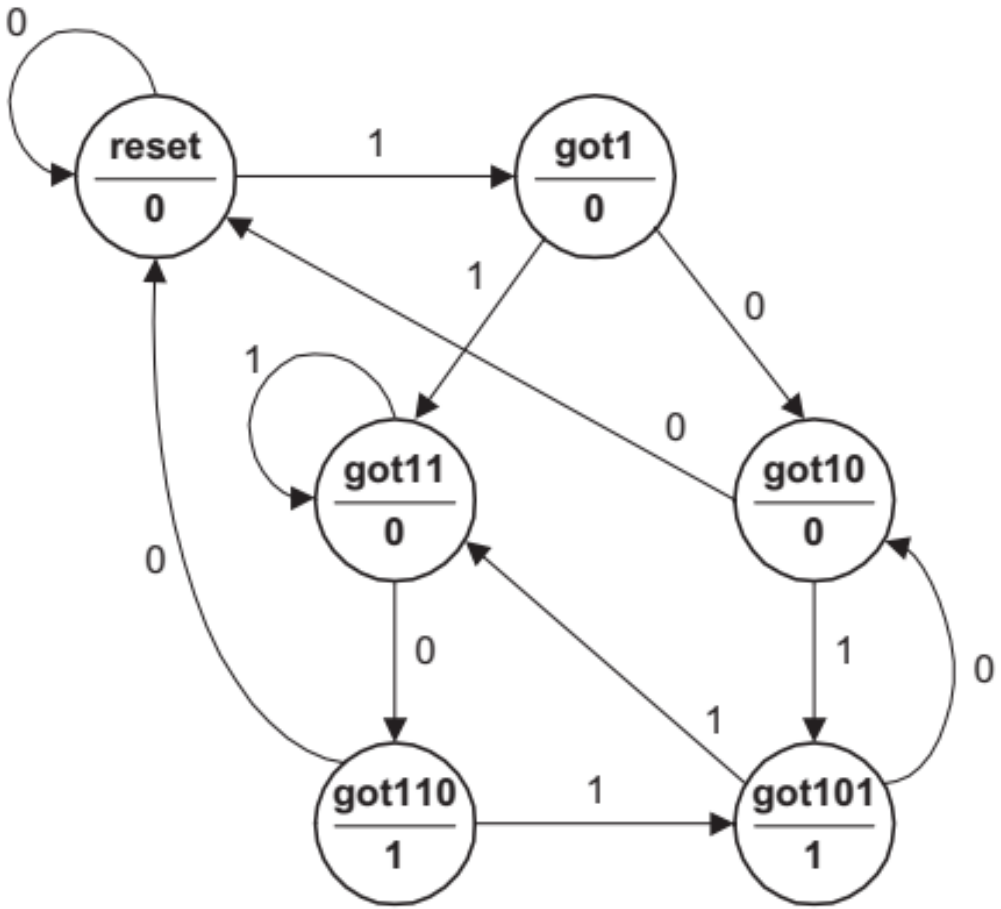


Mealy Machine (101 Detector)




```
`timescale 1ns/100ps

module moore_detector (input x, rst, clk, output z);
   localparam [1:0]
      reset=0, got1=1, got10=2, got101=3;
   reg [1:0] current;
   always @( posedge clk ) begin
      if( rst ) current <= reset;
      else case ( current )
         reset: begin
            if( x==1'b1 ) current <= got1;
            else current <= reset;
         end
         got1: begin
            if( x==1'b0 ) current <= got10;
            else current <= got1;
         end
         got10: begin
            if( x==1'b1 ) current <= got101;
            else current <= reset;
         end
         got101: begin
            if( x==1'b1 ) current <= got1;
            else current <= got10;
         end
         default: begin
            current <= reset;
         end
      endcase
         end

   assign z = (current==got101) ? 1 : 0;
endmodule
```

Moore Machine (101/110 Detector) Overlapping

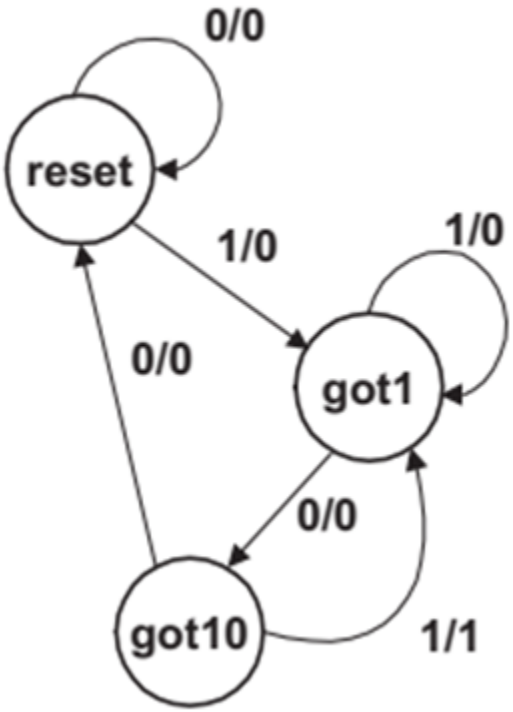


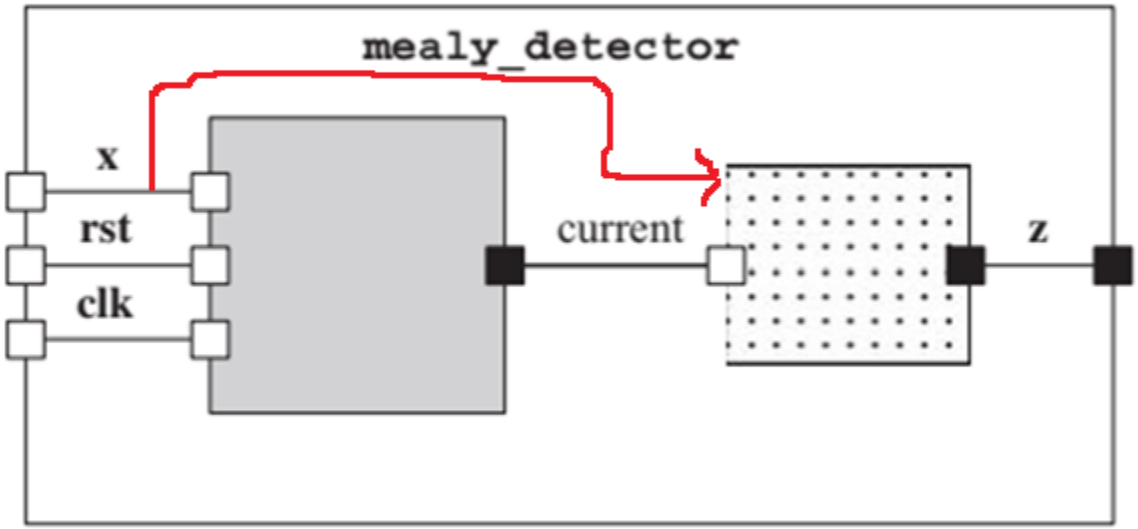
Moore Machine (101/110 Detector) Overlapping

```
`timescale 1ns/100ps

`define   reset    3'b000
`define   got1     3'b001
`define   got10    3'b010
`define   got11    3'b011
`define   got101   3'b100
`define   got110   3'b101
```

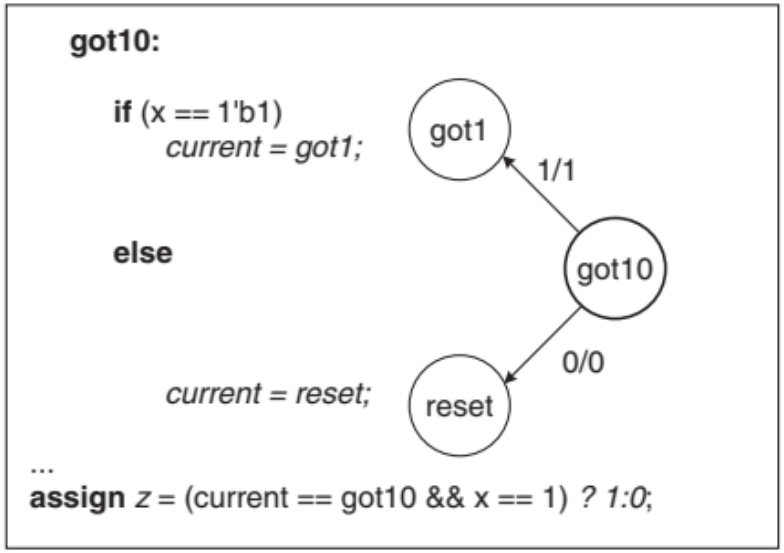
```
module moore_detector3 (input x, rst, clk, output z);
   reg [2:0] current;
   always @( posedge clk or posedge rst ) begin
      if( rst ) current = `reset;
      else
         case ( current )
            `reset:
               if( x==1'b1 ) current <= `got1;
               else current <= `reset;
            `got1:
               if( x==1'b0 ) current <= `got10;
               else current <= `got11;
            `got10:
               if( x==1'b1 ) current <= `got101;
               else current <= `reset;
            `got11:
               if( x==1'b1 ) current <= `got11;
               else current <= `got110;
            `got101:
               if( x==1'b1 ) current <= `got11;
               else current <= `got10;
            `got110:
               if( x==1'b1 ) current <= `got101;
               else current <= `reset;
            default:
               current <= `got101;
         endcase
   end

   assign z = (current == `got101 || current == `got110);
endmodule
```

Mealy Machine (101) Overlapping





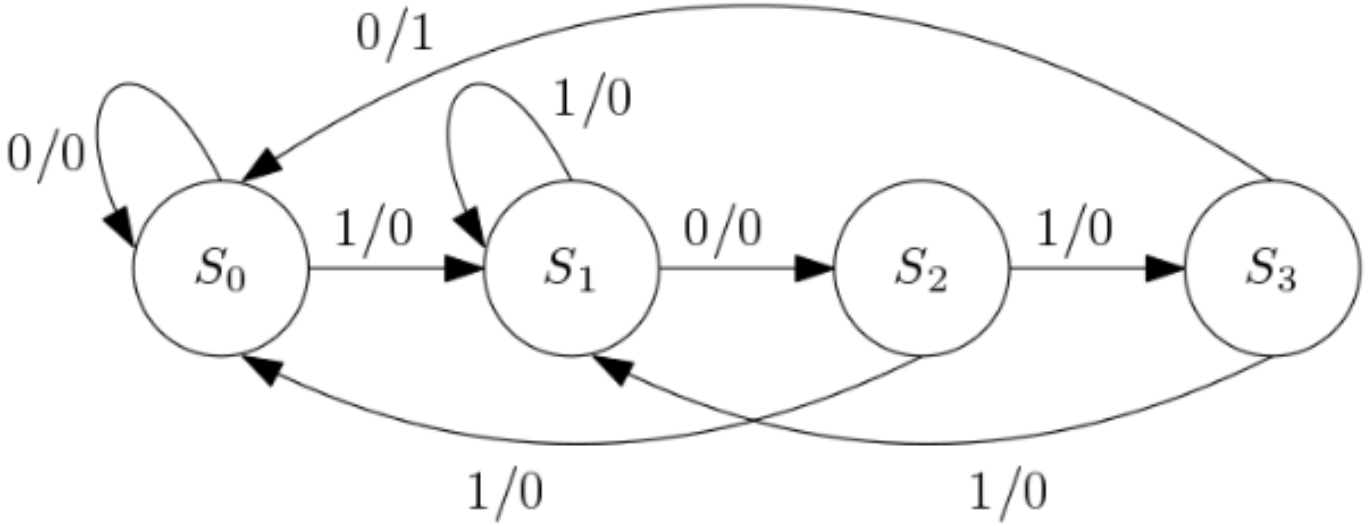
Mealy Machine (101) Overlapping



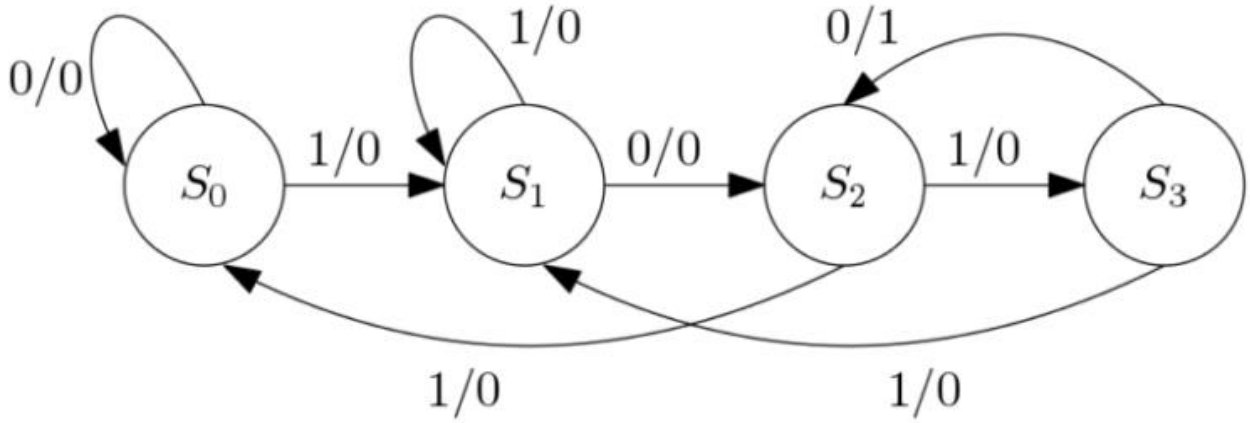

```
`timescale 1ns/100ps

module mealy_detector2 (input x, rst, clk, output z);

   localparam [1:0]
      reset = 0, // 0 = 0 0
      got1 = 1, // 1 = 0 1
      got10 = 2; // 2 = 1 0

   reg [1:0] current;

   always @( posedge clk ) begin
      if (rst) current <= reset;
      else case ( current )
         reset:
            if( x==1'b1 ) current <= got1;
            else current <= reset;
         got1:
            if( x==1'b0 ) current <= got10;
            else current <= got1;
         got10:
            if( x==1'b1 ) current <= got1;
            else current <= reset;
         default:
            current <= reset;
      endcase
   end

   assign z = ( current==got10 && x==1'b1 ) ? 1'b1 : 1'b0;

endmodule
```

Sequence Detector (1010)

Overlapping Not Allowed



Sequence Detector (1010) Overlapping Allowed



FSM Coding Style

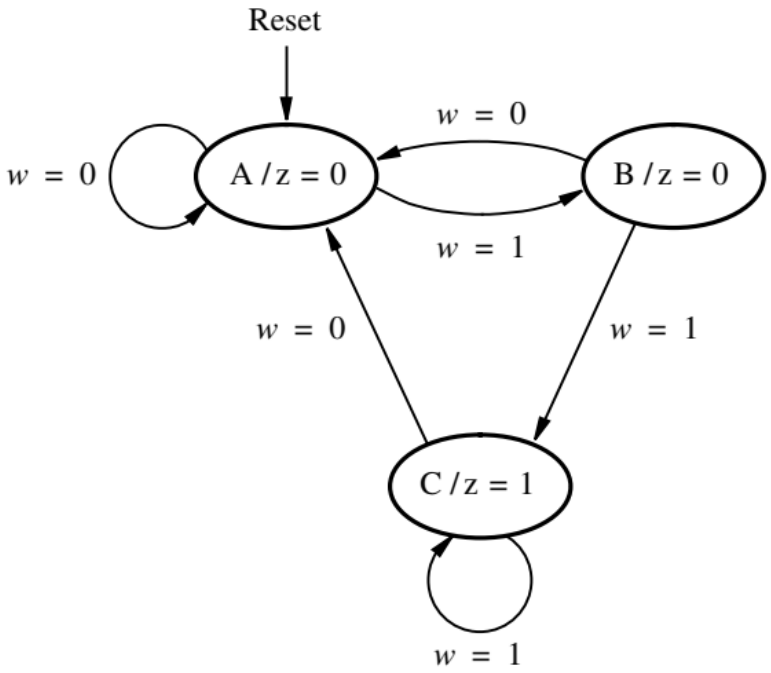
State Register (with synchronous reset):

```
always @(posedge clk) begin
  if (reset)
    state <= IDLE;
  else
    state <= next_state;
end
```

Next-State Logic

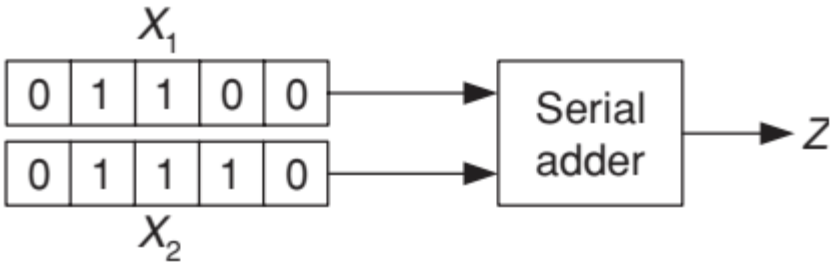
```
always @(*) begin
  case(state)
    IDLE: next_state = (in) ? S1 : IDLE;
    S1:   next_state = (in) ? S2 : IDLE;
    ...
  endcase
end
```

Moore Machine

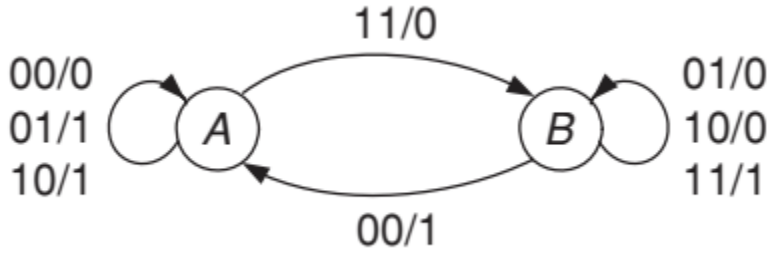


Next-State Logic

Sequential Adder

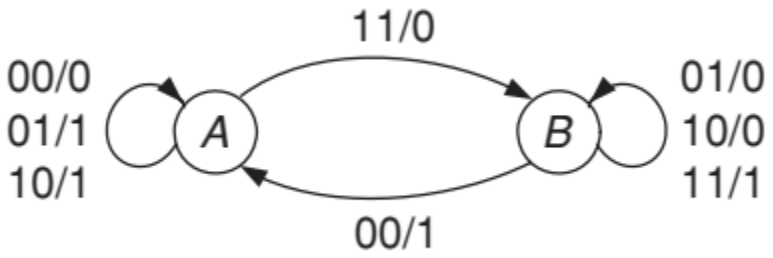


$$
\begin{array}{rccccc}
 & t_5 & t_4 & t_3 & t_2 & t_1 \\
 & 0 & 1 & 1 & 0 & 0 = X_1 \\
+ & 0 & 1 & 1 & 1 & 0 = X_2 \\
\hline
 & 1 & 1 & 0 & 1 & 0 = Z
\end{array}
$$

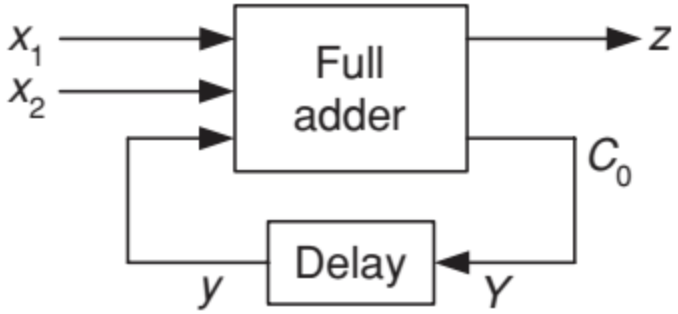


	NS, z			
PS	$x_1x_2 = 00$	01	11	10
A	$A, 0$	$A, 1$	$B, 0$	$A, 1$
B	$A, 1$	$B, 0$	$B, 1$	$B, 0$

Sequential Adder



	NS, z			
PS	$x_1x_2 = 00$	01	11	10
A	$A, 0$	$A, 1$	$B, 0$	$A, 1$
B	$A, 1$	$B, 0$	$B, 1$	$B, 0$

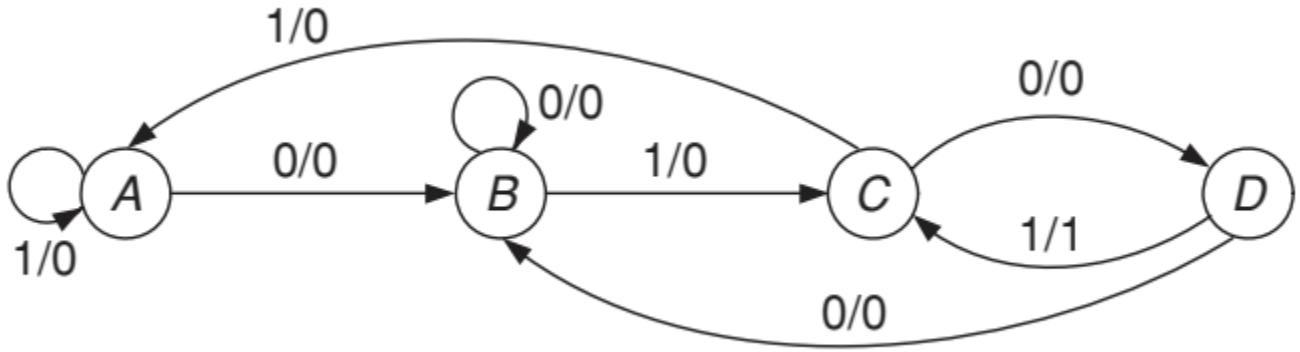


	Next state Y				Output z			
	x_1x_2				x_1x_2			
y	00	01	11	10	00	01	11	10
0	0	0	1	0	0	1	0	1
1	0	1	1	1	1	0	1	0

Sequence Detector (0101)

Input 010101

Output 000101



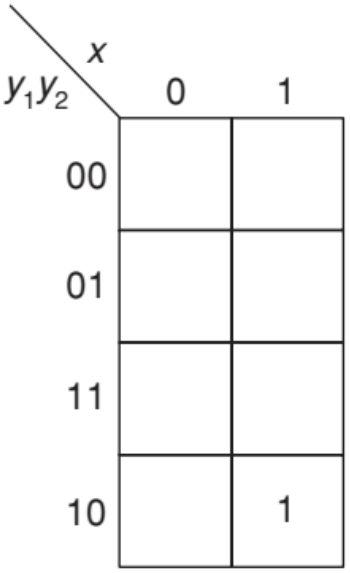
	NS, z	
PS	$x=0$	$x=1$
A	$B, 0$	$A, 0$
B	$B, 0$	$C, 0$
C	$D, 0$	$A, 0$
D	$B, 0$	$C, 1$

		Y_1Y_2		z	
	y_1y_2	$x=0$	$x=1$	$x=0$	$x=1$
A	00	01	00	0	0
B	01	01	11	0	0
C	11	10	00	0	0
D	10	01	11	0	1

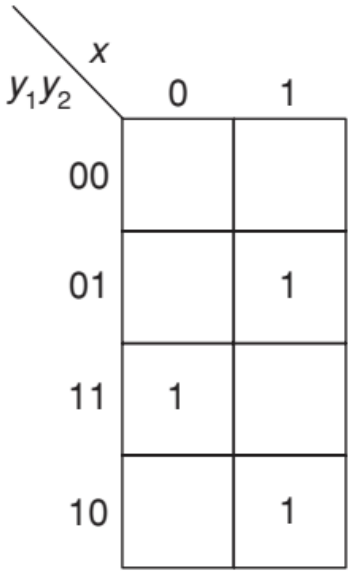
Sequence Detector (0101)

		Y_1Y_2		z	
	y_1y_2	$x=0$	$x=1$	$x=0$	$x=1$
A	00	01	00	0	0
B	01	01	11	0	0
C	11	10	00	0	0
D	10	01	11	0	1

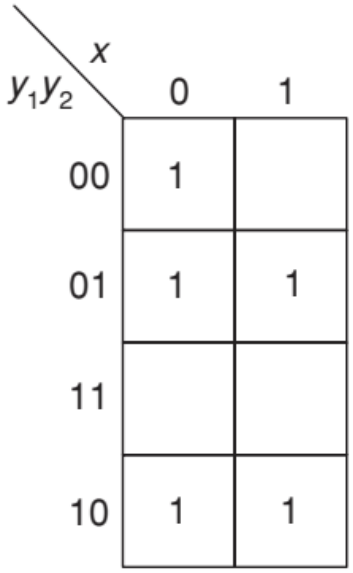
$$z = xy_1y_2',$$
$$Y_1 = x'y_1y_2 + xy_1'y_2 + xy_1y_2',$$
$$Y_2 = y_1y_2' + x'y_1' + y_1'y_2.$$



(a) z map.

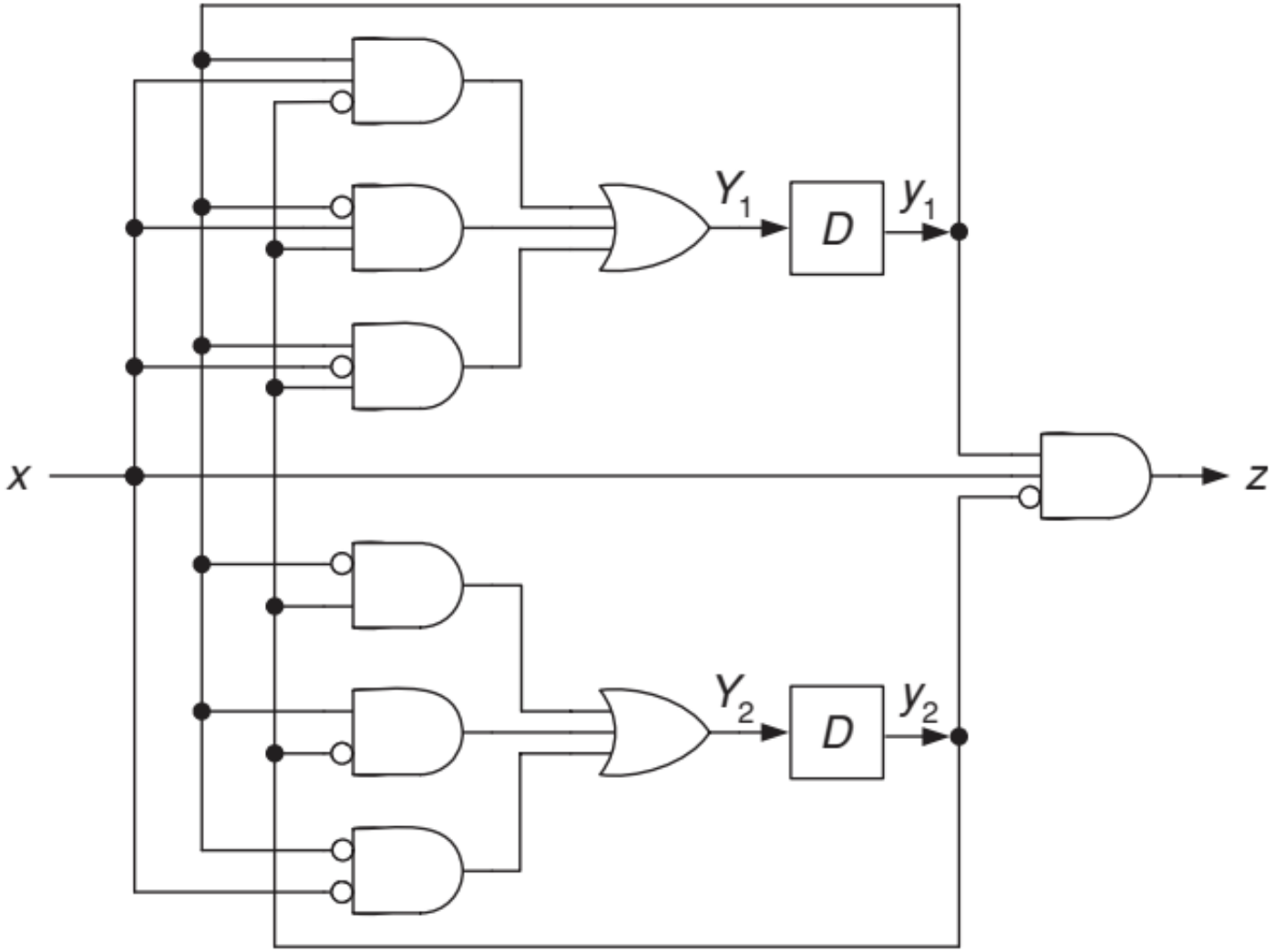


(b) Y_1 map.



(c) Y_2 map.

Sequence Detector (0101)



Applications of FSM

- **CPU / Processor Control Unit** – Sequence instructions and generate control signals
- **Sequence Detectors** – Detect specific patterns in serial data (e.g., 101)Communication Protocols – UART, SPI, I2C, Ethernet controllers
- **Counters and Timers** – Up/Down counters, decade counters, programmable timers
- **Traffic Light & Elevator Controllers** – Manage states based on timers or inputs
- **Memory & Storage Controllers** – Manage read/write operations in RAM/FIFO/Cache
- **Vending Machines / Payment Systems** – Track coins, dispense products/change
- **Motor Controllers** – Stepper and servo motor sequencing Test & Diagnostic Systems – Built-in self-test (BIST) for ICs

Thank you !

Happy Learning