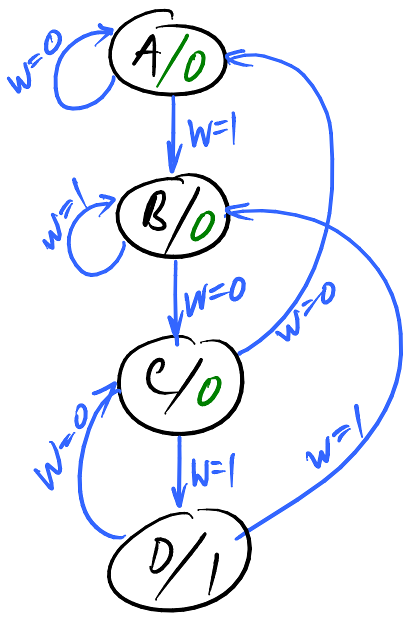
6.2, 6.8 FSM (cont.)

Let's work on the same "101" pattern detection problem from last lecture. (2nd version)

1. State Diagram

→ This design allows overlapping



State

A – Waiting for "1"

B – seeing first "1" in the pattern

C – seeing "1" followed by "0"

D – seeing "101" pattern

2. State Table

P.S	N.S $w=0$	N.S $w=1$	Output z
A	A	B	0
B	C	B	0
C	A	D	0
D	C	B	1

3. 2 FF's $y_2y_1=00$ (A), $y_2y_1=01$ (B), $y_2y_1=10$ (C), $y_2y_1=11$ (D)

State-assigned Table

	P.S. y_2y_1	N.S $w=0$ Y_2Y_1	N.S $w=1$ Y_2Y_1	Output z
(A)	0 0	00	01	0
(B)	0 1	10	01	0
(C)	1 0	00	11	0
(D)	1 1	10	01	1

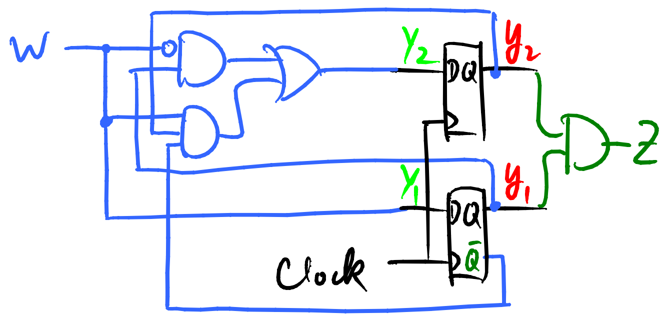
4. Next-state and output expressions

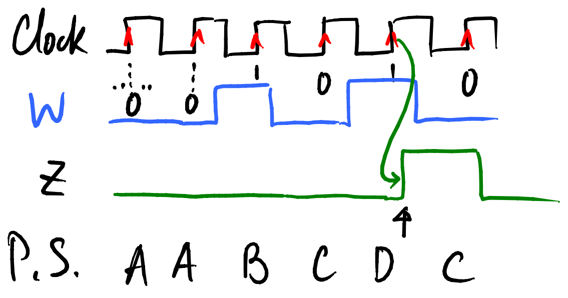
$w \backslash y_2y_1$	00	01	11	10
0		1	1	
1				1

$Y_2 = y_1\bar{w} + y_2\bar{y}_1w$

$Y_1 = w$ (by inspection)

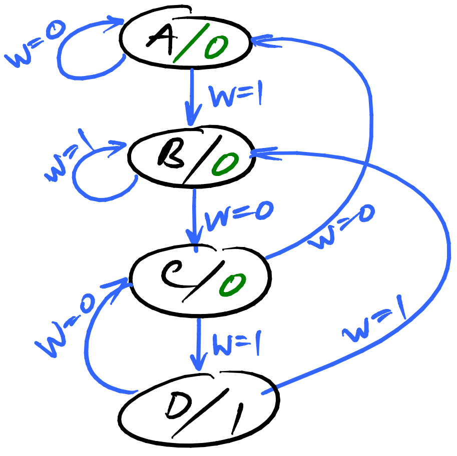
$z = y_2y_1$

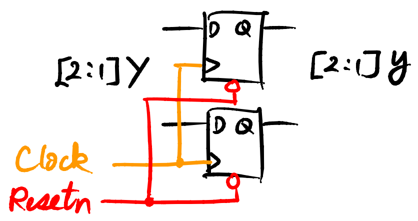




Verilog code for a FSM can be written by using 3 blocks:
combinational circuit A, FFs, combinational circuit B

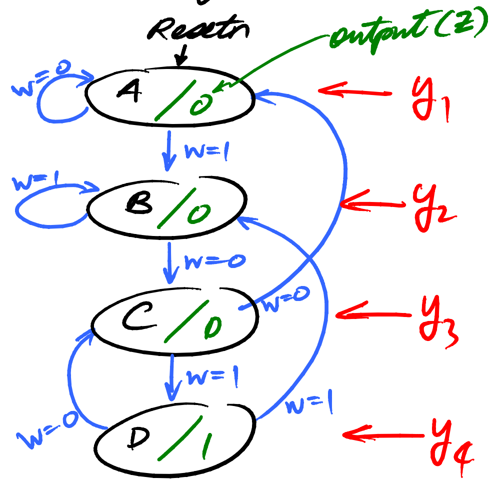
```
module seq101(input clock, W, Resetn, output Z);
  reg [2:1] y, Y;   // y2, y1 are the present states, Y2, Y1 are the next states
  parameter A=2'b00, B=2'b01, C=2'b10, D=2'b11; // state assignment
  // combinational circuit A, describing state transitions
  always@(W, y)
    case (y)
      A: if (!W) Y=A;
         else  Y=B;
      B: if (!W) Y=C;
         else  Y=B;
      C: if (!W) Y=A;
         else  Y=D;
      D: if (!W) Y=C;
         else  Y=B;
    endcase
  // FFs describing D-ffs with synchronous reset
  always@(posedge clock)
    if (Resetn==0)
      y<=A;
    else
      y<=Y;
  // combinational circuit B, describing output logics
   assign Z=(y==D);   // Z=1 if p.s is D
endmodule
```





⟹ Another style of state encoding → One-hot encoding
let's work on the same "101" pattern detection example

State diagram



Use 4 FFs to represent this design

$y_4 y_3 y_2 y_1 =$	0001	(A)
	0010	(B)
	0100	(C)
	1000	(D)

Describe next-state logics.

$$Y_1 = y_1\bar{w} + y_3\bar{w} = (y_1+y_3)\bar{w}$$

gives you input value

gives you current state

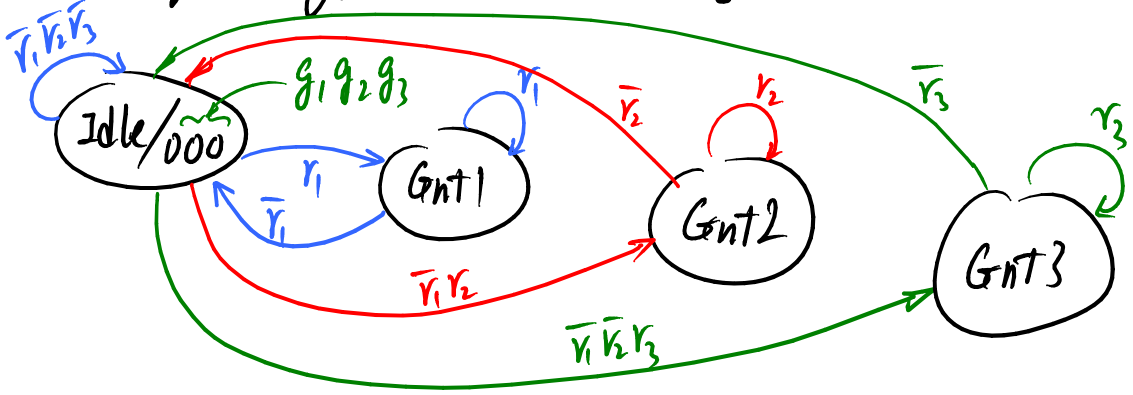
$$Y_2 = y_1 w + y_2 w + y_4 w = (y_1+y_2+y_4)w$$

$$Y_3 = y_2\bar{w} + y_4\bar{w} = (y_2+y_4)\bar{w}$$

$$Y_4 = y_3 w$$

Output logic $z = y_4$

Design example (Arbiter) – design a FSM that controls access to a shared resource by three devices. Each device requests use of the resource by asserting r_1, r_2 or r_3. The Arbiter decides which device "gets" the resource, and sets its grant signal g_1, g_2 or g_3. There is a priority scheme: $r_1 > r_2 > r_3$.



$\bar{r}_1\bar{r}_2\bar{r}_3$
Idle / 000
$g_1 g_2 g_3$
r_1
$\bar{r}_1$
r_1
Gnt1
$\bar{r}_2$
r_2
Gnt2
$\bar{r}_1 r_2$
$\bar{r}_3$
r_3
Gnt3
$\bar{r}_1\bar{r}_2 r_3$