Digital System Design

Lecture 7
Finite State Machines

Reading Assignment:

- Brown, “Fundamentals of Digital Logic with VHDL, pp. 485 - 519

Learning Objective:

- Modeling sequential logic using the behavioral modeling techniques.
- Design VHDL behavioral models for Moore and Mealy finite-state machines

Learning Examples:

- A Simple Sequence Recognizer (Moore)
- A Simple Sequence Recognizer (Mealy)
- A Gray-code counter
- An Arbiter
- A Serial Adder (Mealy)
- A Serial Adder (Moore)

Overview

- "State machine" (or "finite state machine") is a generic name given to sequential circuits
- "Clocked" indicates that the flip-flops employ a CLOCK (CLK) input
- "Synchronous" means that all the flip-flops in the state machine use the same CLOCK signal
- "Analysis" means to analyze the behavior of a given state machine
  - construct a PS-NS table
  - derive PS-NS equations
  - draw a state transition diagram
  - draw a timing chart

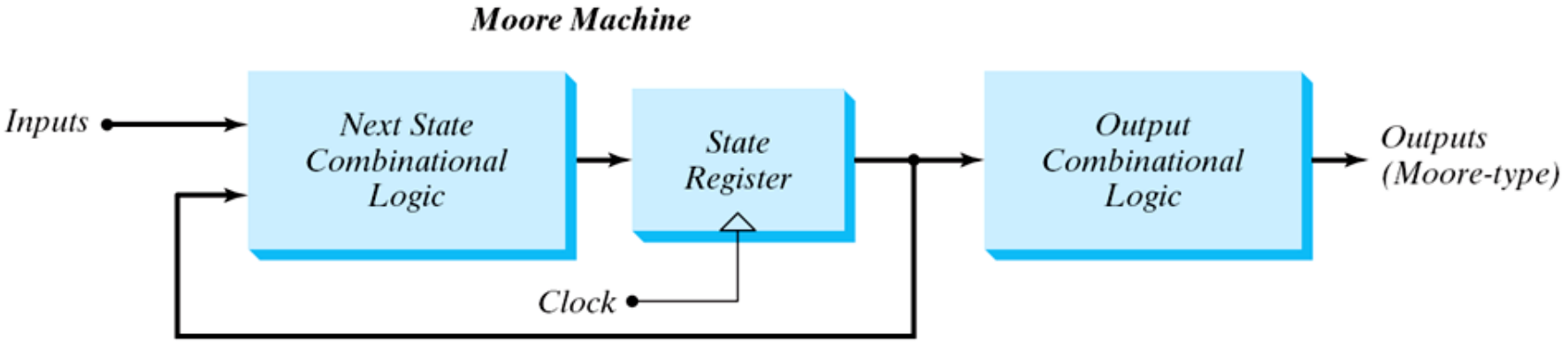
State Machine Structure

Clocked synchronous state machines consist of three basic blocks:

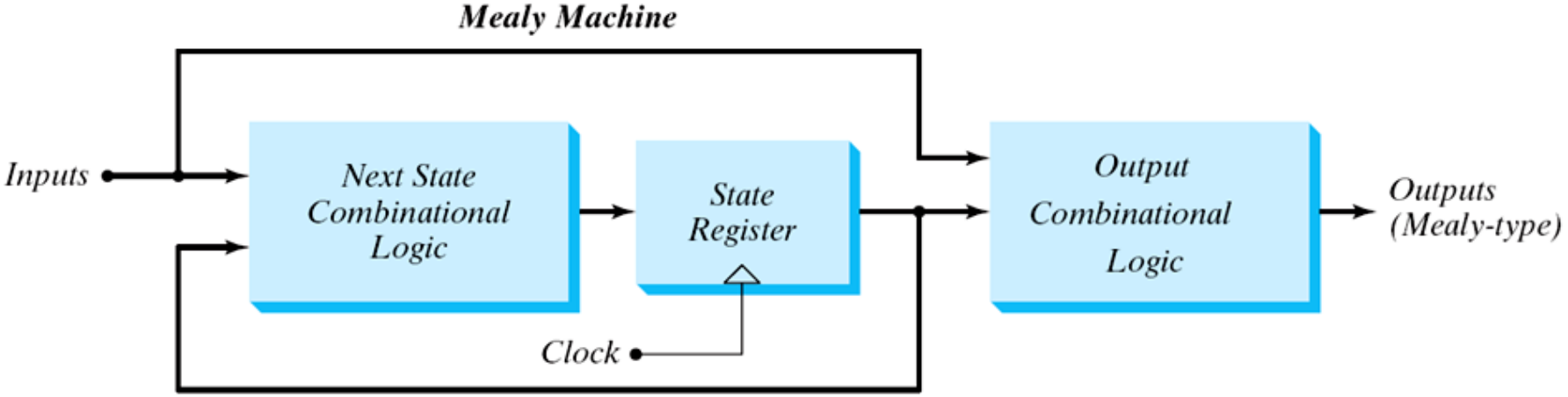
- next state logic – combinational circuitry that provides the "excitation" necessary to transition to the next state, based on the current state and the present inputs
- state memory (flip flops) – set of N flip-flops that store the current state of the machine (providing 2^N distinct states)
- output logic – combinational circuitry that uses the current state (and possibly current inputs) to determine the outputs generated

Machine Structures

In a Moore machine, the outputs are only a function of the current state

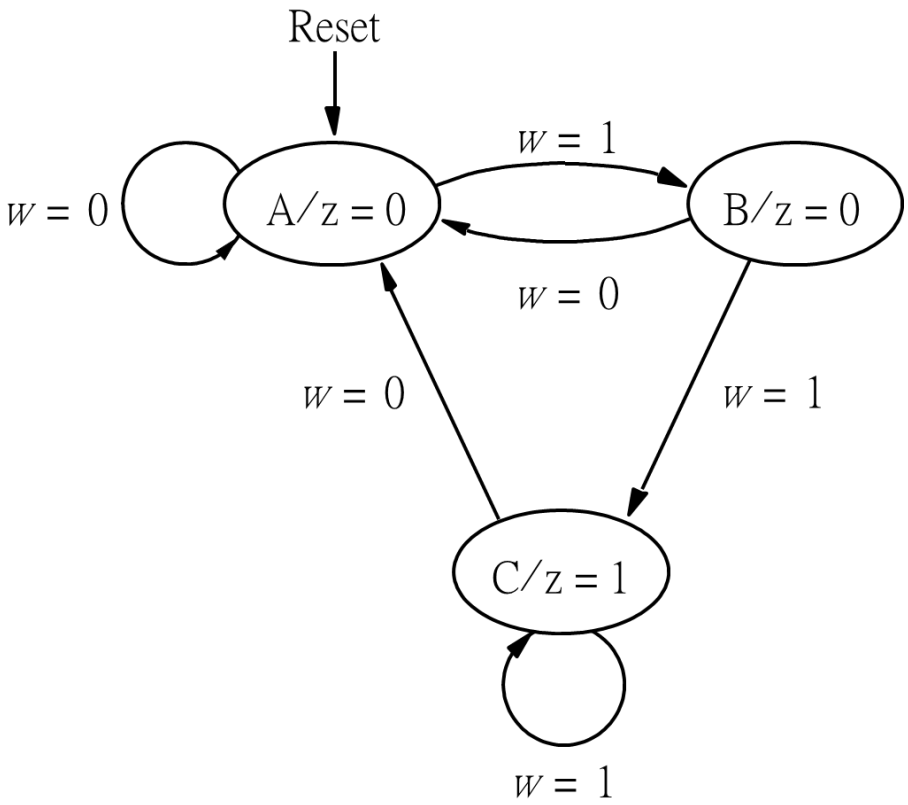


In a Mealy machine, the outputs are a function of the current state as well as the current inputs



Example – A Simple Sequence Recognizer (Moore)

- Circuit has one input, *w*, and one output, *z*.
- Changes occur on positive clock edge.
- *z* is 1 if *w* is 1 during last two clock cycles.

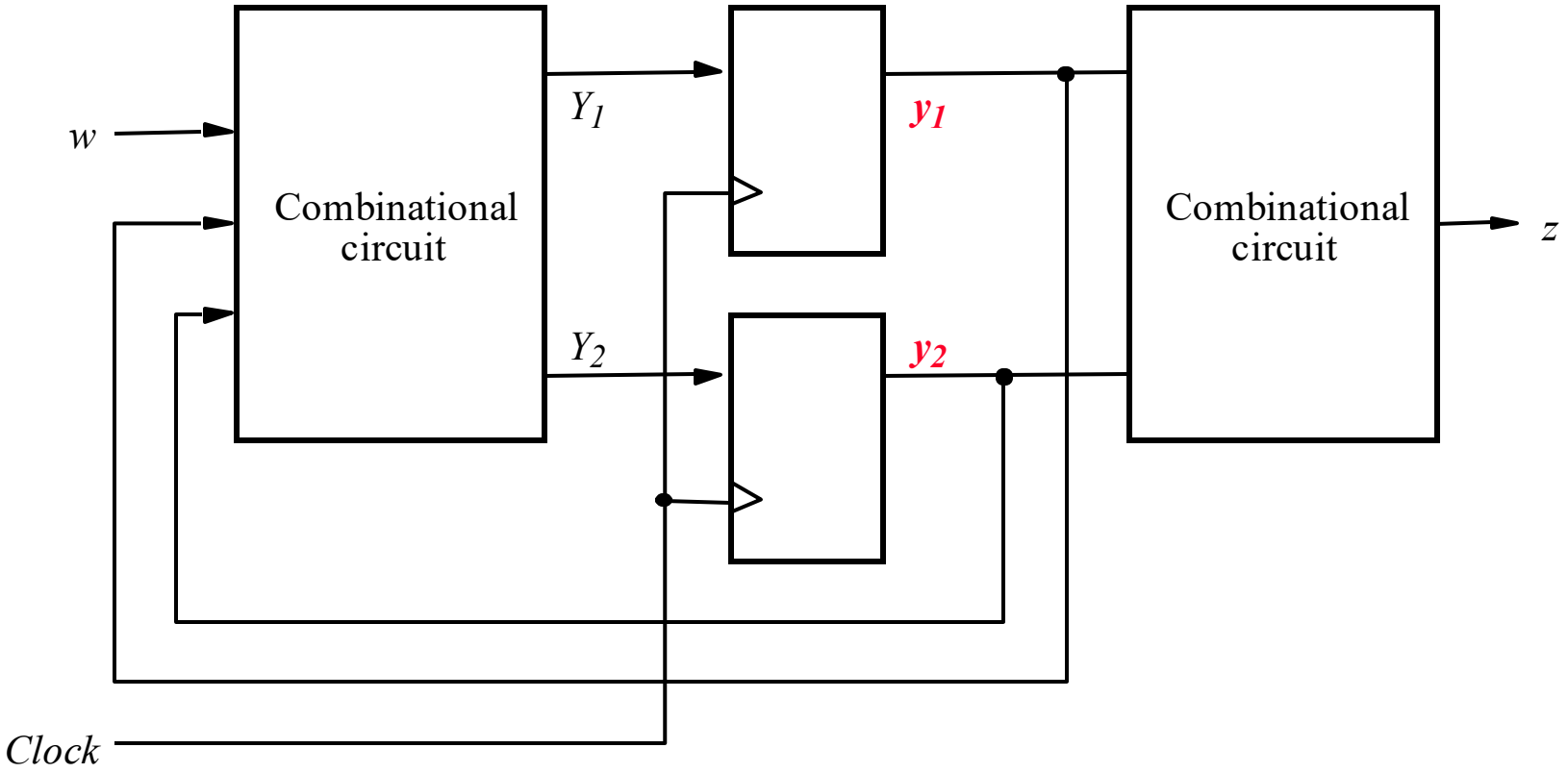


Example – A Simple Sequence Recognizer (Moore)

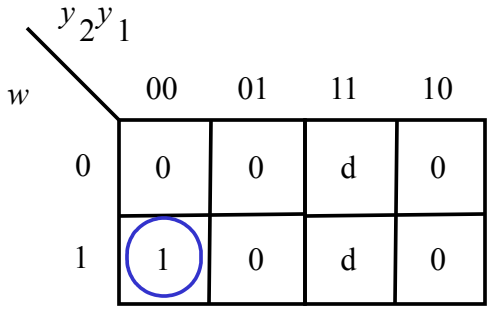
Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

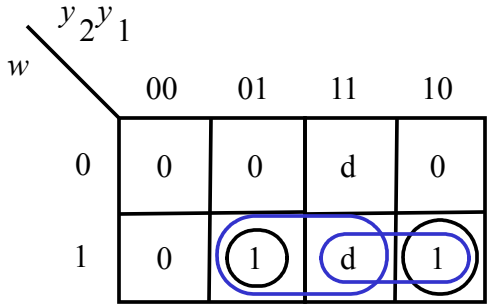
	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

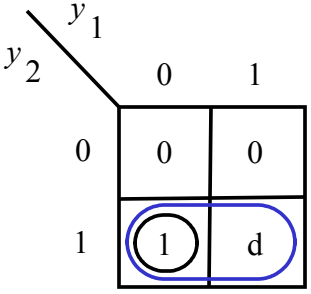
Example – A Simple Sequence Recognizer (Moore)



Example – A Simple Sequence Recognizer (Moore)







Ignoring don't cares

$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$$

$$z = \bar{y}_1y_2$$

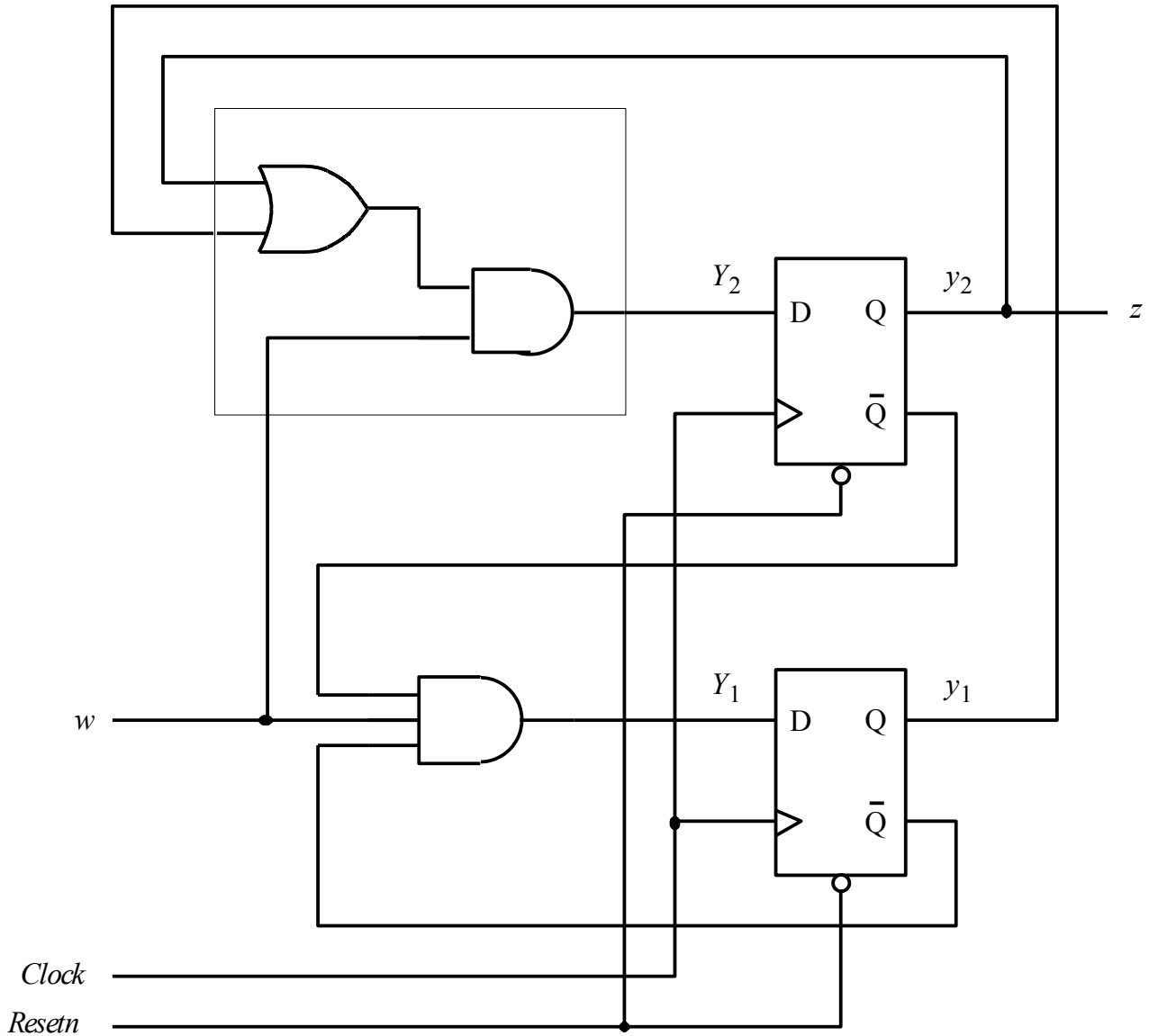
Using don't cares

$$Y_1 = w\bar{y}_1\bar{y}_2$$

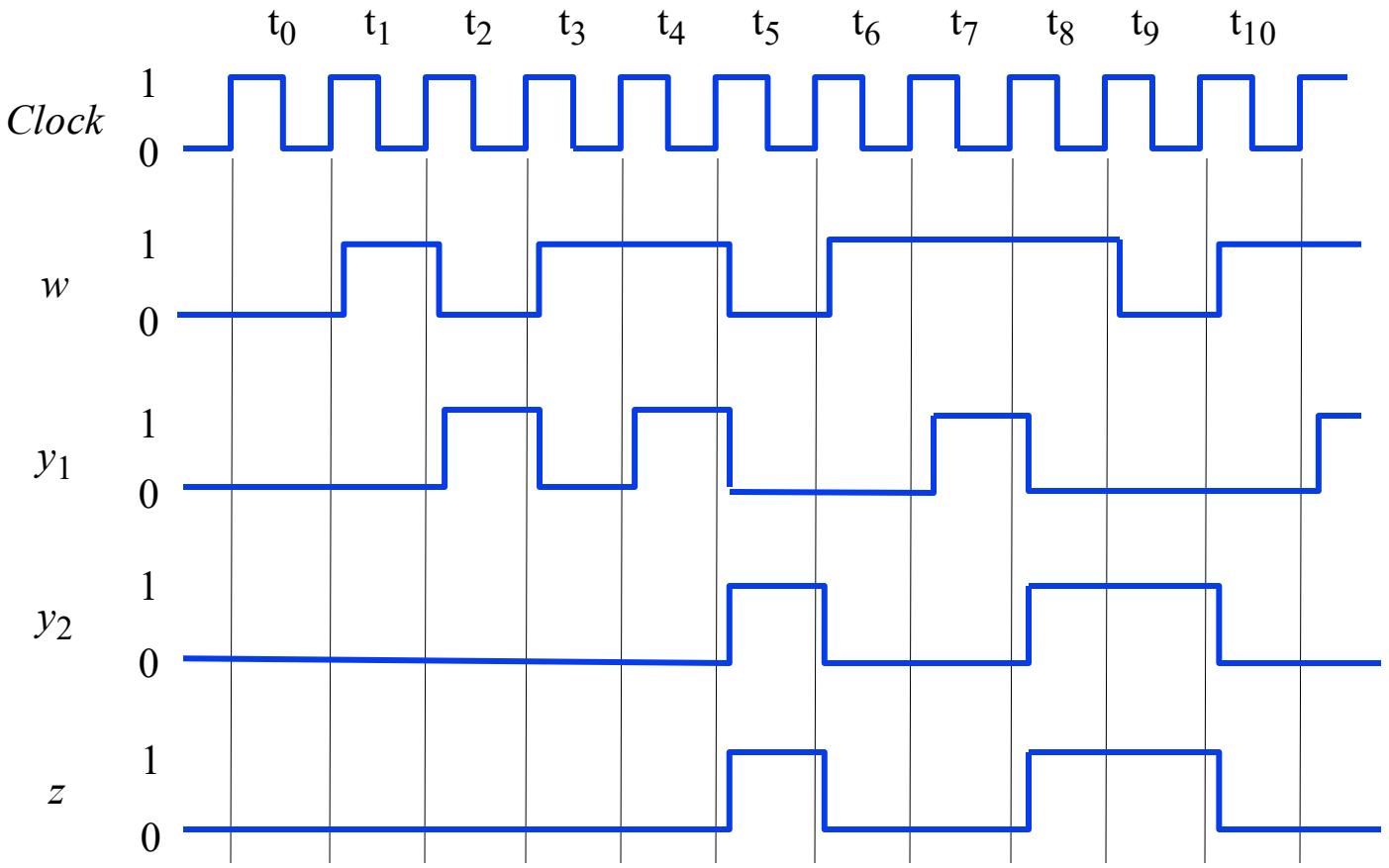
$$Y_2 = wy_1 + wy_2 = w(y_1 + y_2)$$

$$z = y_2$$

Example –A Simple Sequence Recognizer (Moore)



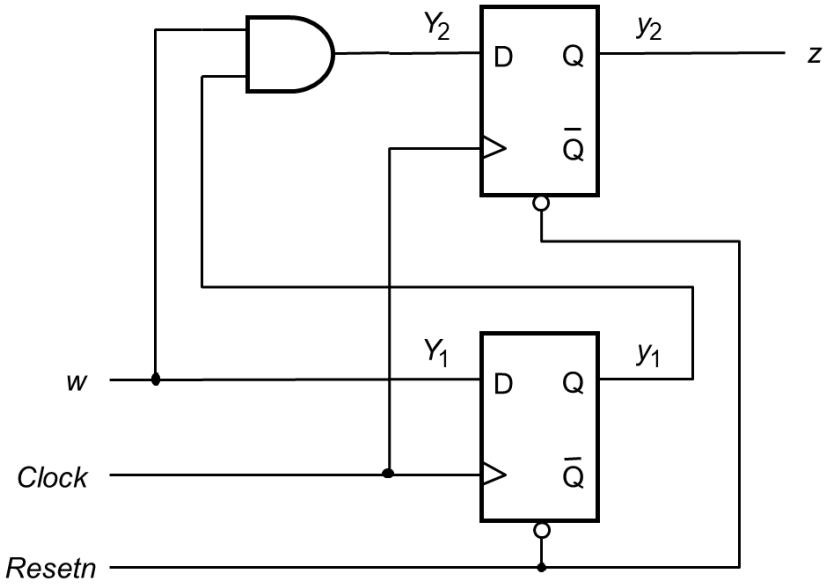
Example – A Simple Sequence Recognizer (Moore)



Example – A Simple Sequence Recognizer (Moore)

- Improved state assignment

	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	*dd*	*dd*	*d*



- One-hot state state assignment

	Present state $y_3y_2y_1$	Nextstate $w = 0$ $Y_3Y_2Y_1$	Nextstate $w = 1$ $Y_3Y_2Y_1$	Output z
A	001	001	010	0
B	010	001	100	0
C	100	001	100	1

VHDL code for a Simple Sequence Recognizer (Moore)

```
USE ieee.std_logic_1164.all ;

ENTITY simple IS
    PORT ( Clock, Resetn, w : IN  STD_LOGIC ;
           z                : OUT STD_LOGIC ) ;
END simple ;

ARCHITECTURE Behavior OF simple IS
    TYPE State_type IS (A, B, C) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            y <= A ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            CASE y IS
                WHEN A =>
                    IF w = '0' THEN
                        y <= A ;
                    ELSE
                        y <= B ;
                    END IF ;
                WHEN B =>
                    IF w = '0' THEN
                        y <= A ;
                    ELSE
                        y <= C ;
                    END IF ;
                WHEN C =>
                    IF w = '0' THEN
                        y <= A ;
                    ELSE
                        y <= C ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;
    z <= '1' WHEN y = C ELSE '0' ;
END Behavior ;
```

A Better VHDL code for a Simple Sequence Recognizer

```
(ENTITY declaration not shown)

ARCHITECTURE Behavior OF simple IS
        TYPE State_type IS (A, B, C) ;
        SIGNAL y_present, y_next : State_type ;
BEGIN
        PROCESS ( w, y_present )        -- next state logic
        BEGIN
                CASE y_present IS
                        WHEN A =>
                                IF w = '0' THEN
                                        y_next <= A ;
                                ELSE
                                        y_next <= B ;
                                END IF ;
                        WHEN B =>
                                IF w = '0' THEN
                                        y_next <= A ;
                                ELSE
                                        y_next <= C ;
                                END IF ;
```

A Better VHDL code for a Simple Sequence Recognizer

```
                WHEN C =>
                        IF w = '0' THEN
                                y_next <= A ;
                        ELSE
                                y_next <= C ;
                        END IF ;
        END CASE ;
END PROCESS ;

PROCESS (Clock, Resetn)          -- state memory
BEGIN
        IF Resetn = '0' THEN
                y_present <= A ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
                y_present <= y_next ;
        END IF ;
END PROCESS ;

z <= '1' WHEN y_present = C ELSE '0' ;      -- output logic
END Behavior ;
```

A user-defined attribute for manual state assignment

```
(ENTITY declaration not shown)

ARCHITECTURE Behavior OF simple IS
        TYPE State_TYPE IS (A, B, C) ;
        ATTRIBUTE ENUM_ENCODING                         : STRING ;
        ATTRIBUTE ENUM_ENCODING OF State_type     : TYPE IS "00 01 11" ;
        SIGNAL y_present, y_next : State_type ;
BEGIN

con't ...
```

Using constants for manual state assignment

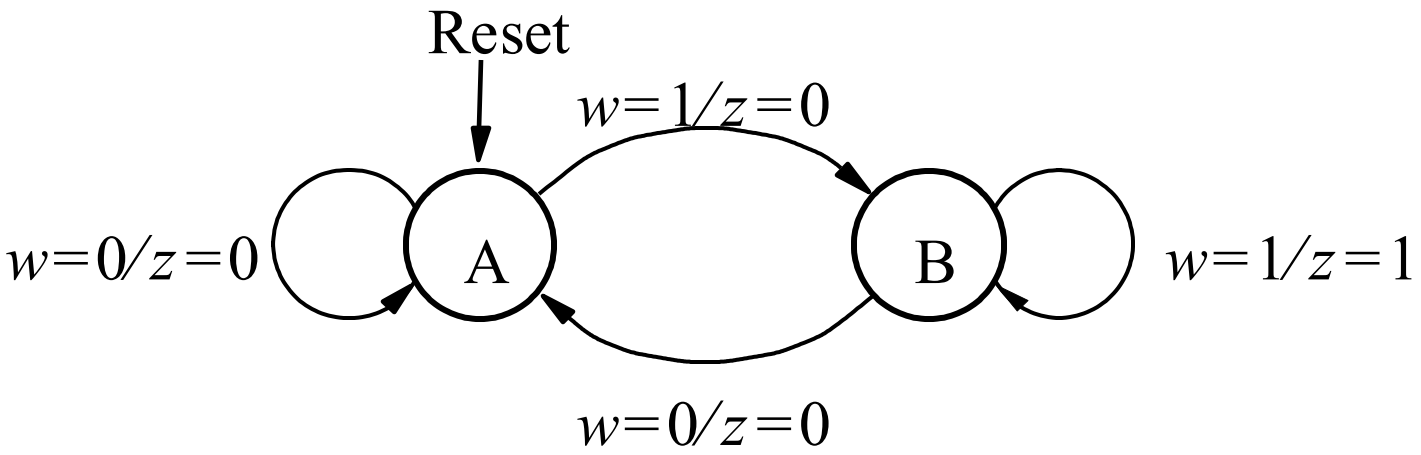
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY simple IS
        PORT (    Clock, Resetn, w        : IN  STD_LOGIC ;
                  z                       : OUT       STD_LOGIC ) ;
END simple ;

ARCHITECTURE Behavior OF simple IS
        SIGNAL y_present, y_next : STD_LOGIC_VECTOR(1 DOWNTO 0);
        CONSTANT A : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00" ;
        CONSTANT B : STD_LOGIC_VECTOR(1 DOWNTO 0) := "01" ;
        CONSTANT C : STD_LOGIC_VECTOR(1 DOWNTO 0) := "11" ;
BEGIN
        PROCESS ( w, y_present )       -- next state logic
        BEGIN
              CASE y_present IS
                    WHEN A =>
                          IF w = '0' THEN y_next <= A ;
                          ELSE y_next <= B ;
                          END IF ;
```

… con't

Using constants for manual state assignment

```
                WHEN B =>
                    IF w = '0' THEN y_next <= A ;
                    ELSE y_next <= C ;
                    END IF ;
                WHEN C =>
                    IF w = '0' THEN y_next <= A ;
                    ELSE y_next <= C ;
                    END IF ;
                WHEN OTHERS =>
                    y_next <= A ;
            END CASE ;
        END PROCESS ;

        PROCESS ( Clock, Resetn )       -- state memory
        BEGIN
            IF Resetn = '0' THEN
                y_present <= A ;
            ELSIF (Clock'EVENT AND Clock = '1') THEN
                y_present <= y_next ;
            END IF ;
        END PROCESS ;
        z <= '1' WHEN y_present = C ELSE '0' ;     -- output logic
END Behavior ;
```

Example – A Simple Sequence Recognizer (Mealy)



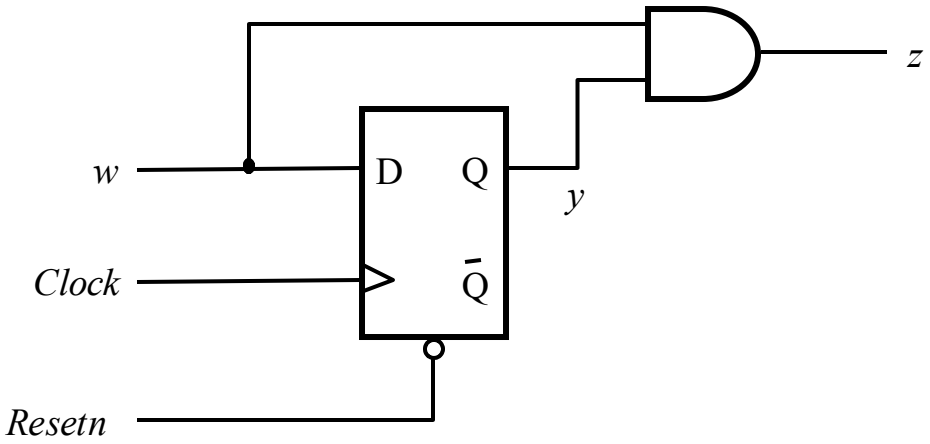
Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

Example – A Simple Sequence Recognizer (Mealy)

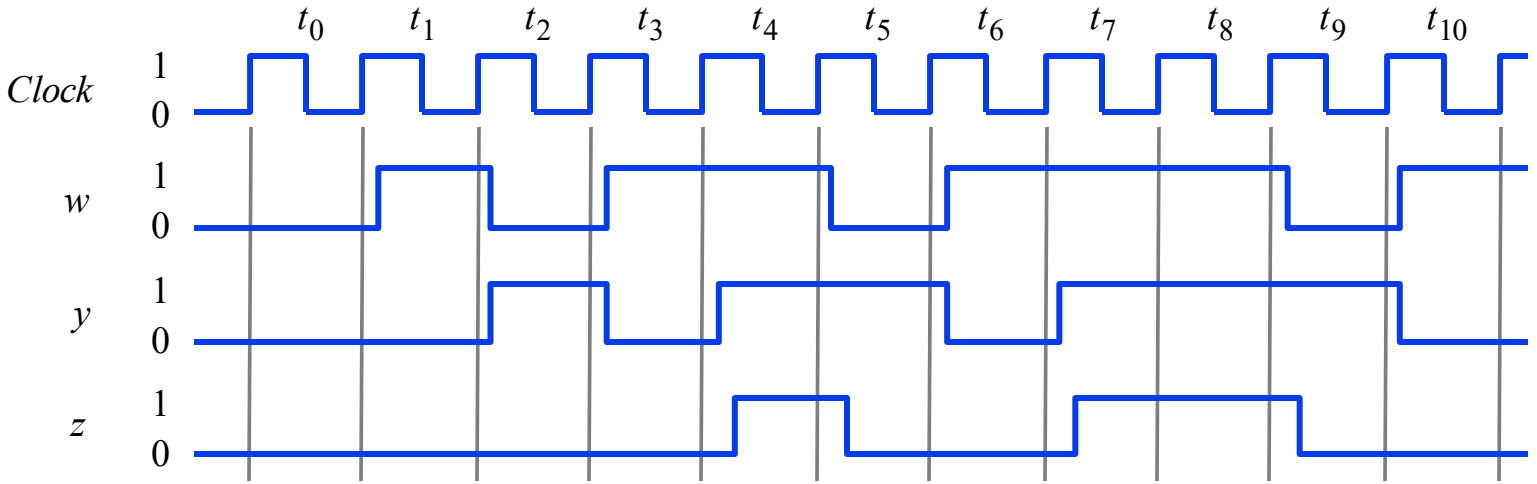
Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

Example – A Simple Sequence Recognizer (Mealy)



(a) Circuit



(b) Timing diagram

VHDL code for a Simple Sequence Recognizer (Mealy)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mealy IS
        PORT (    Clock, Resetn, w      : IN     STD_LOGIC ;
                  z                     : OUT    STD_LOGIC ) ;
END mealy ;

ARCHITECTURE Behavior OF mealy IS
        TYPE State_type IS (A, B) ;
        SIGNAL y_present, y_next : State_type ;
BEGIN
        PROCESS ( w, y_present )      -- next state logic
        BEGIN
              CASE y_present IS
                    WHEN A =>
                          IF w = '0' THEN  y_next <= A ;
                          ELSE  y_next <= B ;
                          END IF ;
                    WHEN B =>
                          IF w = '0' THEN  y_next <= A ;
                          ELSE  y_next <= B ;
                          END IF ;
                    END CASE ;
         END PROCESS ;
```

VHDL code for a Simple Sequence Recognizer (Mealy)

```
        PROCESS ( Clock, Resetn )      -- state memory
         BEGIN
              IF Resetn = '0' THEN
                   y_present <= A ;
              ELSIF (Clock'EVENT AND Clock = '1') THEN
                   y_present <= y_next ;
              END IF ;
         END PROCESS ;

        PROCESS (w, y_present )          -- output logic
         BEGIN
              CASE y_present IS
                   WHEN A =>
                        z <= '0' ;
                   WHEN B =>
                        z <= w ;
              END CASE ;
        END PROCESS ;
END Behavior ;
```

Example – Gray-code counter

Design an FSM that realizes a three-bit Gray-code counter, which counts in the sequence

000, 001, 011, 010, 110, 111, 101, 100, 000, ...

- If w = 0, the present count remains the same; if w = 1, the count is incremented

State table (Moore Machine)

Present state	Next state		Output
	$w = 0$	$w = 1$	$z_2 z_1 z_0$
A	A	B	0 0 0
B	B	C	0 0 1
C	C	D	0 1 1
D	D	E	0 1 0
E	E	F	1 1 0
F	F	G	1 1 1
G	G	H	1 0 1
H	H	A	1 0 0

VHDL for Gray-code counter

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob8_45 IS
      PORT ( Clock, Resetn, w : IN STD_LOGIC ;
                  z : OUT STD_LOGIC_VECTOR(2 DOWNTO 0) ) ;
END prob8_45 ;

ARCHITECTURE Behavior OF prob8_45 IS
      SIGNAL y_present, y_next : STD_LOGIC_VECTOR(2 DOWNTO 0);
      CONSTANT A : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000" ;
      CONSTANT B : STD_LOGIC_VECTOR(2 DOWNTO 0) := "001" ;
      CONSTANT C : STD_LOGIC_VECTOR(2 DOWNTO 0) := "011" ;
      CONSTANT D : STD_LOGIC_VECTOR(2 DOWNTO 0) := "010" ;
      CONSTANT E : STD_LOGIC_VECTOR(2 DOWNTO 0) := "110" ;
      CONSTANT F : STD_LOGIC_VECTOR(2 DOWNTO 0) := "111" ;
      CONSTANT G : STD_LOGIC_VECTOR(2 DOWNTO 0) := "101" ;
      CONSTANT H : STD_LOGIC_VECTOR(2 DOWNTO 0) := "100" ;
```

. . . con't

VHDL for Gray-code counter

```
BEGIN
    PROCESS ( w, y_present )
    BEGIN
        CASE y_present IS
            WHEN A =>
                IF w = '0' THEN y_next <= A ;
                ELSE y_next <= B ;
                END IF ;
            WHEN B =>
                IF w = '0' THEN y_next <= B ;
                ELSE y_next <= C ;
                END IF ;
            WHEN C =>
                IF w = '0' THEN y_next <= C ;
                ELSE y_next <= D ;
                END IF ;
            WHEN D =>
                IF w = '0' THEN y_next <= D ;
                ELSE y_next <= E ;
                END IF ;
            WHEN E =>
                IF w = '0' THEN y_next <= E ;
                ELSE y_next <= F ;
                END IF ;
            WHEN F =>
                IF w = '0' THEN y_next <= F ;
                ELSE y_next <= G ;
                END IF ;
            WHEN G =>
                IF w = '0' THEN y_next <= G ;
                ELSE y_next <= H ;
                END IF ;
            WHEN H =>
                IF w = '0' THEN y_next <= H ;
                ELSE y_next <= A ;
                END IF ;
        END CASE ;
    END PROCESS ;

    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            y_present <= A ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            y_present <= y_next ;
        END IF ;
    END PROCESS ;

    z <= y_present ;
END Behavior ;
```

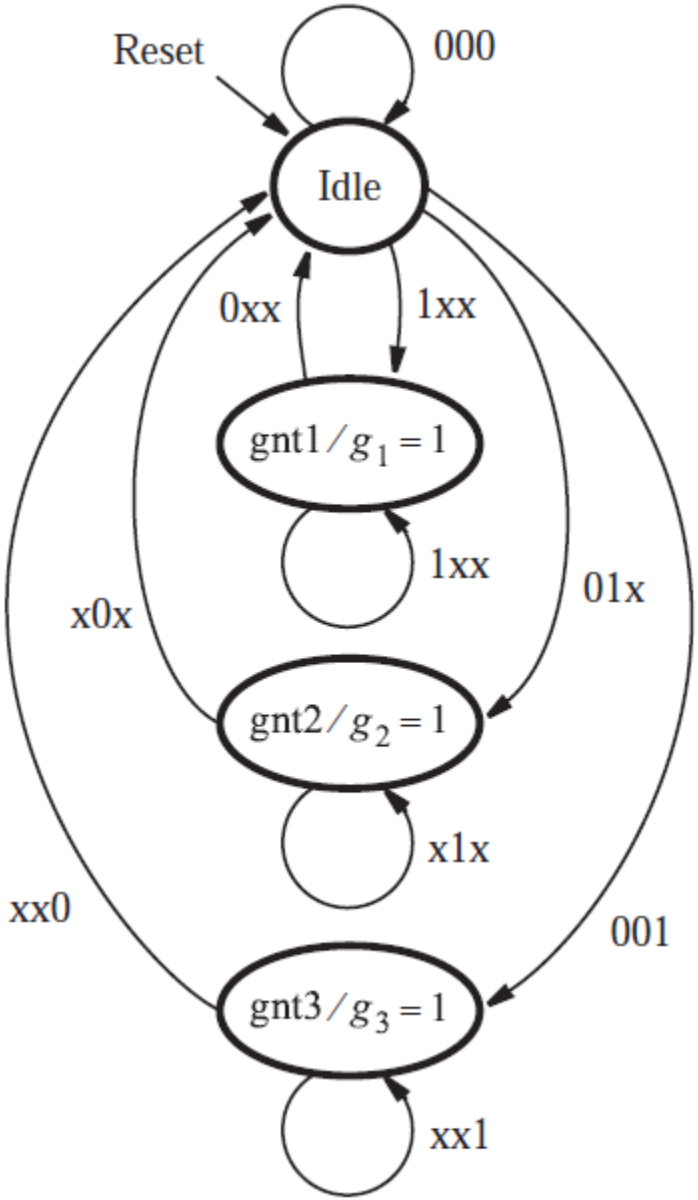
Example – An Arbiter Circuit

The purpose of the machine is to control access by various devices to a shared resource in a given system.

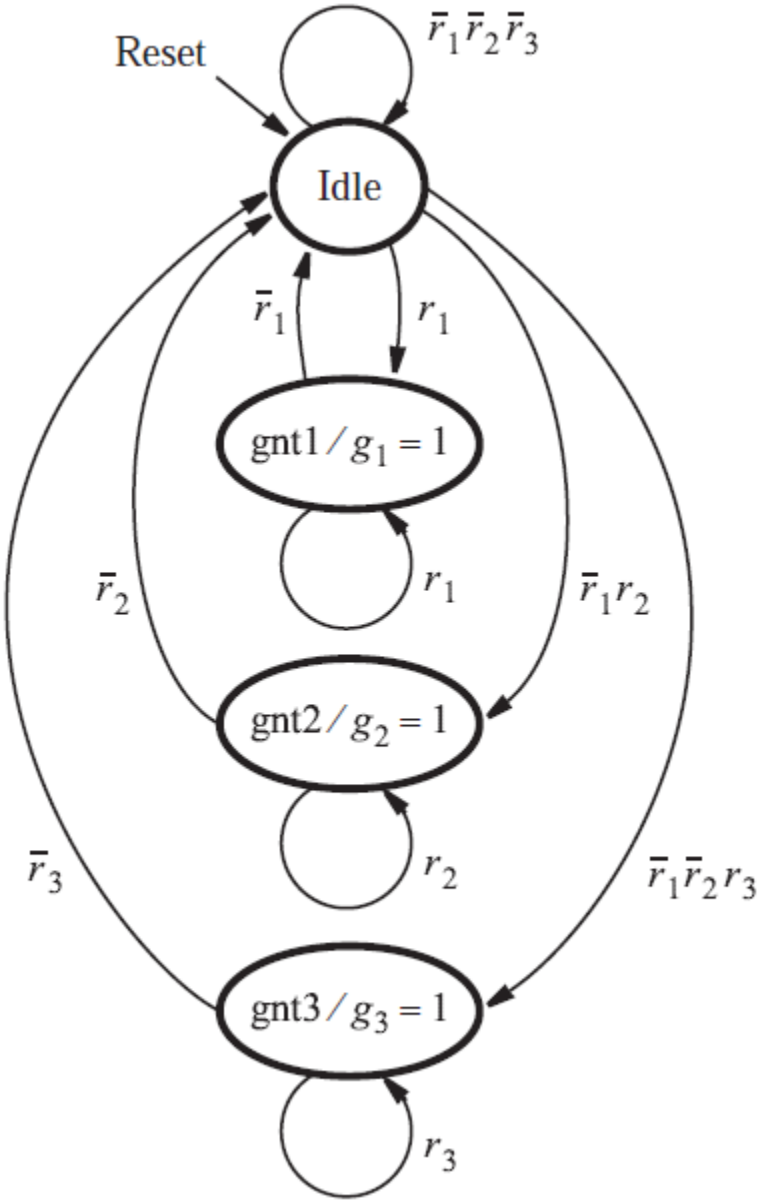
- Only one device can use the resource at a time.
- Each device provides one input to the FSM, called a *request*. A device indicates its need to use the resource by asserting its request signal.
- FSM produces a separate output for each device, called a *grant*.
- Based on a priority scheme, it selects one of the requesting devices and asserts its grant signal. When the device is finished using the resource, it de-asserts its request signal.

- We will assume that there are three devices in the system, called device 1, device 2, and device 3.
- The request signals are named r_1, r_2, and r_3, and the grant signals are called g_1, g_2, and g_3.
- Priority: device 1 > device 2 > device 3

State Diagram for the Arbiter



Alternative style of State Diagram for the Arbiter



VHDL for the Arbiter

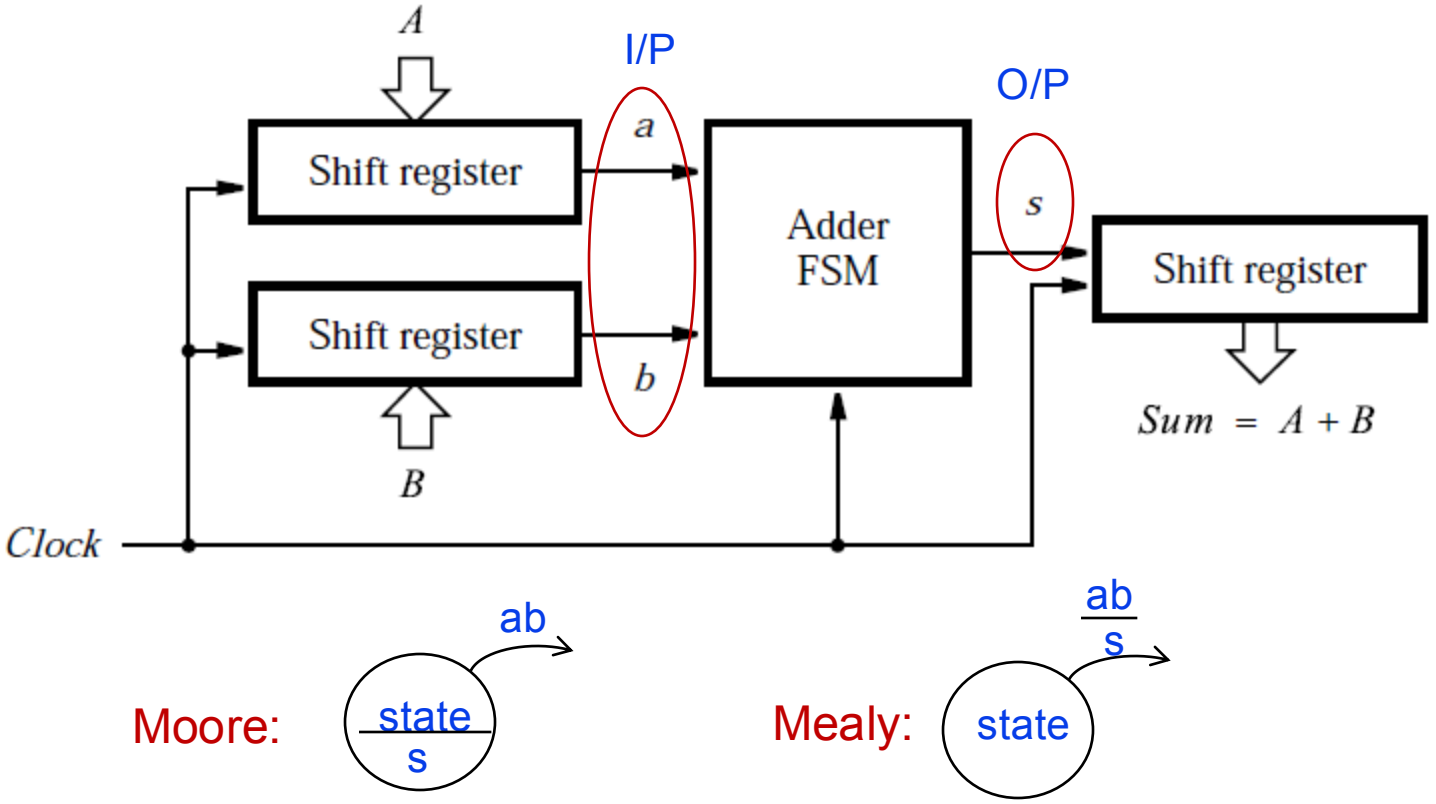
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY arbiter IS
        PORT (    Clock, Resetn    : IN     STD_LOGIC ;
                  r                : IN     STD LOGIC VECTOR(1 TO 3) ;
                  g                : OUT    STD LOGIC VECTOR(1 TO 3) ) ;
END arbiter ;

ARCHITECTURE Behavior OF arbiter IS
        TYPE State_type IS (Idle, gnt1, gnt2, gnt3) ;
        SIGNAL y_present, y_next : State_type ;
BEGIN
        PROCESS ( r, y_present )        -- next state logic
        BEGIN
             CASE y_present  IS
                  WHEN Idle =>
                       IF  r(1) = '1'          THEN        y_next <= gnt1 ;
                       ELSIF r(2) = '1'   THEN        y_next <= gnt2 ;
                       ELSIF r(3) = '1'   THEN        y_next <= gnt3 ;
                       ELSE                           y_next <= Idle ;
                       END IF ;
                  WHEN gnt1 =>
                       IF  r(1) = '1'          THEN        y_next <= gnt1 ;
                       ELSE                           y_next <= Idle ;
                       END IF ;
```

VHDL for the Arbiter

```
                WHEN gnt2 =>
                    IF  r(2) = '1'      THEN      y_next <= gnt2 ;
                    ELSE                          y_next <= Idle ;
                    END IF ;
                WHEN gnt3 =>
                    IF  r(3) = '1'      THEN      y_next <= gnt3 ;
                    ELSE                          y_next <= Idle ;
                    END IF ;
        END CASE ;
    END PROCESS ;

    PROCESS (Clock, Resetn)          -- state memory
    BEGIN
        IF Resetn = '0' THEN
            y_present <= Idle ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            y_present <= y_next ;
        END IF ;
    END PROCESS ;

    g(1) <= '1' WHEN y_present = gnt1 ELSE '0' ;     -- output logic
    g(2) <= '1' WHEN y_present = gnt2 ELSE '0' ;
    g(3) <= '1' WHEN y_present = gnt3 ELSE '0' ;

END Behavior ;
```

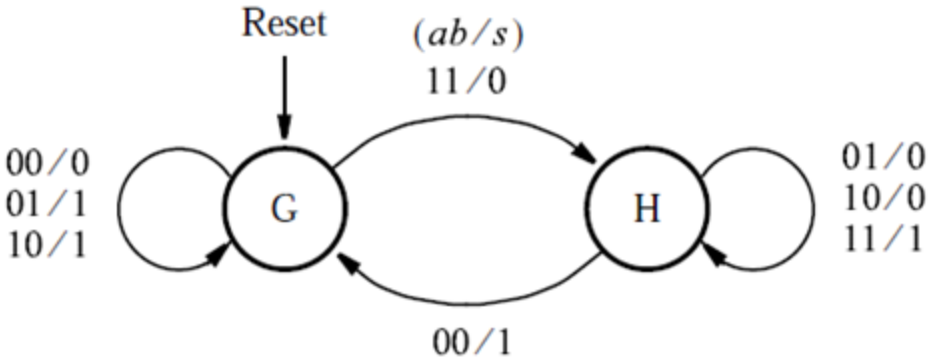
Example – Serial Adder

- A = $a_{n-1}a_{n-2} \cdots a_0$ and B = $b_{n-1}b_{n-2} \cdots b_0$ be two unsigned numbers that have to be added to produce Sum = $s_{n-1}s_{n-2} \cdots s_0$
- The addition task begins by loading the values of A and B into shift registers. Then in each clock cycle, a pair of bits is added by the adder FSM, and at the end of the cycle the resulting sum bit is shifted into the Sum register.



Serial Adder (Mealy FSM)

- Two states are needed: let G and H denote the states where the carry-in values are 0 and 1, respectively
- Each transition is labeled using the notation ab/s, which indicates the value of s for a given valuation ab



G: carry-in = 0
H: carry-in = 1

Present state	Next state				Output s			
	$ab = 00$	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

VHDL for Serial Adder (Mealy FSM)

Code for a left-to-right shift register with an enable input

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

- - left-to-right shift register with parallel load and enable
ENTITY shiftrne IS
      GENERIC ( N : INTEGER := 4 ) ;
      PORT ( R       : IN     STD_LOGIC_VECTOR(N–1 DOWNTO 0) ;
             L, E, w : IN     STD_LOGIC ;
             Clock   : IN     STD_LOGIC ;
             Q       : BUFFER STD_LOGIC_VECTOR(N–1 DOWNTO 0) ) ;
END shiftrne ;

ARCHITECTURE Behavior OF shiftrne IS
BEGIN
      PROCESS
      BEGIN
            WAIT UNTIL Clock'EVENT AND Clock = '1' ;
            IF E = '1' THEN
                  IF L = '1' THEN
                        Q <= R ;
                  ELSE
                        Genbits: FOR i IN 0 TO N–2 LOOP
                              Q(i) <= Q(i+1);
                        END LOOP ;
                        Q(N–1) <= w ;
                  END IF ;
            END IF ;
      END PROCESS ;
END Behavior ;
```

VHDL for Serial Adder (Mealy FSM)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY serial IS
    GENERIC ( length : INTEGER := 8 ) ;
    PORT ( Clock  : IN      STD_LOGIC ;
           Reset  : IN      STD_LOGIC ;
           A, B   : IN      STD_LOGIC_VECTOR(length-1 DOWNTO 0) ;
           Sum    : BUFFER  STD_LOGIC_VECTOR(length-1 DOWNTO 0) );
END serial ;

ARCHITECTURE Behavior OF serial IS
    COMPONENT shiftrne
        GENERIC ( N : INTEGER := 4 ) ;
        PORT( R       : IN      STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
              L, E, w : IN      STD_LOGIC ;
              Clock   : IN      STD_LOGIC ;
              Q       : BUFFER  STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
    END COMPONENT ;

    SIGNAL QA, QB, Null_in : STD_LOGIC_VECTOR(length-1 DOWNTO 0) ;
    SIGNAL s, Low, High, Run : STD_LOGIC ;
    SIGNAL Count : INTEGER RANGE 0 TO length ;
    TYPE State_type IS (G, H) ;
    SIGNAL y : State_type ;
```

VHDL for Serial Adder (Mealy FSM)

```
BEGIN
    Low <= '0' ; High <= '1' ;
    ShiftA: shiftrne GENERIC MAP (N => length)
        PORT MAP ( A, Reset, High, Low, Clock, QA ) ;
    ShiftB: shiftrne GENERIC MAP (N => length)
        PORT MAP ( B, Reset, High, Low, Clock, QB ) ;

    AdderFSM: PROCESS ( Reset, Clock )
    BEGIN
        IF Reset = '1' THEN
            y <= G ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            CASE y IS
                WHEN G =>
                    IF QA(0) = '1' AND QB(0) = '1' THEN y <= H ;
                    ELSE y <= G ;
                    END IF ;
                WHEN H =>
                    IF QA(0) = '0' AND QB(0) = '0' THEN y <= G ;
                    ELSE y <= H ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS AdderFSM ;
```

VHDL for Serial Adder (Mealy FSM)

```
    WITH y SELECT
        s <=  QA(0) XOR QB(0)          WHEN G,
              NOT ( QA(0) XOR QB(0) ) WHEN H ;

    Null_in <= (OTHERS => '0') ;
    ShiftSum: shiftrne GENERIC MAP ( N => length )
                       PORT MAP ( Null_in, Reset, Run, s, Clock, Sum ) ;

    Stop: PROCESS
    BEGIN
        WAIT UNTIL (Clock'EVENT AND Clock = '1') ;
        IF Reset = '1' THEN
            Count <= length ;
        ELSIF Run = '1' THEN
            Count <= Count -1 ;
        END IF ;
    END PROCESS ;

    Run <= '0' WHEN Count = 0 ELSE '1' ;      -- stops counter and ShiftSum
END Behavior ;
```

Serial Adder (Moore FSM)

- We will use G_0 and G_1 to denote the fact that the carry is 0 and that the sum is either 0 or 1, respectively.
- Similarly, we will use H_0 and H_1 to denote the fact that the carry is 1 and that the sum is either 0 or 1, respectively.