ECE 2214

Digital Logic Design

L-30 FSM design examples

Fall 2022

Outline

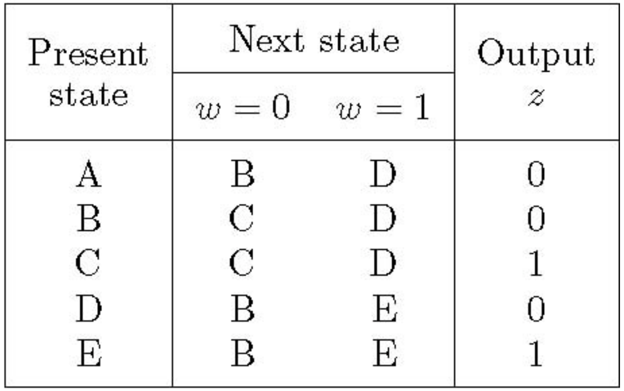
- Design of a sequence detector with a Moore-type FSM
  - Schematic and VHDL implementation
- Design of a sequence detector FSM combining smaller Moore-type FSMs
  - Schematic
- Design of a sequence detector with a Mealy-type FSM
  - Schematic and VHDL implementation

Summary of previous lecture

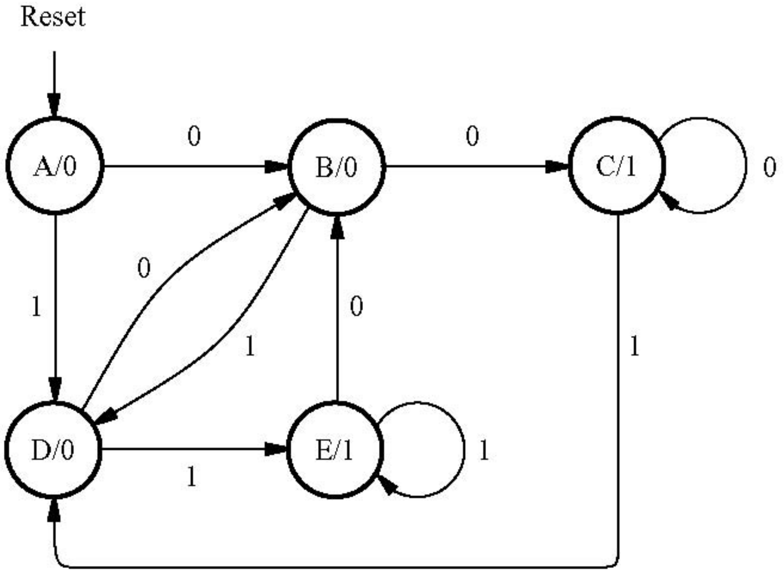
- FSM state minimization: Required when there are redundant states
  - Achieved with the ***partition minimization procedure***
- Hypothesis:
  - All the states in a given block are equivalent
  - States in different block are not equivalent
- Implementation
  - Start partitioning the blocks based on the k-successor method until no new group is generated
  - States in the same group are equivalent and can be minimized
  - Keep only one of them
- Analyzed an example
  - Method terminates when the next partition step do not reach any new group compared with the previous partition or…
  - Reach groups of only one variable

Example 1

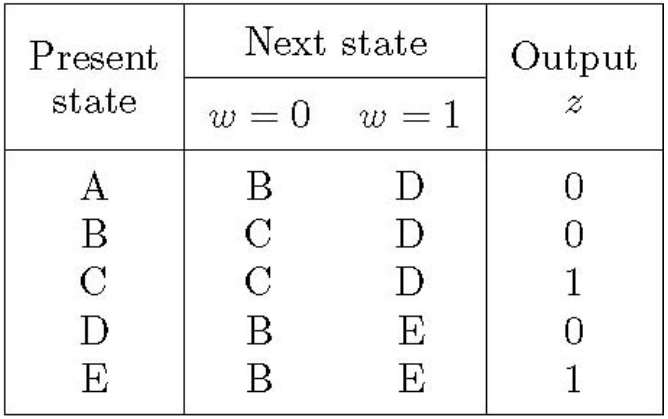
- Design a FSM that has an input ***w*** and an output z that…
  - Detects two consecutive previous values of the input in 00 or 11 → ***z*** = 1
  - Otherwise, ***z*** = 0.
- **Solution:**
  - A similar problem to the example analyzed in previous lecture that detect a sequence of 11
  - Will follow the same approach to solve this problem
  - Will use a Moore machine because w depend on two consecutive previous stages
- Let C be the state that will be active when the sequence 00 occurs
- Let E be the state that will be active when the sequence 11 occurs



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



Example 1



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

- Partition minimization procedure

$$P_1 = (ABCDE)$$

$$P_2 = (ABD)(CE)$$

  - 0-sucessor of ABD →BCB
    ➔ B provides a 0-sucessor in a different group (C)
  - 1-sucessor of ABD →DDE
    ➔ D provides a 1-sucessor in a different group (E)
  - 0-sucessor of CE →CB ➔ Both 0-sucessors are in different groups
  - 1-sucessor of CE →DE ➔ Both 1-sucessors are in different groups

$$P_3 = (A)(BD)(C)(E)$$

$$P_4 = (A)(B)(C)(D)(E)$$

  - Since all variables appear in separate groups, the minimization procedure is stopped
  - The optimum number of states is 5 and do not contain any redundancy
- Flip-flop selection & number: Since there are 5 states: 3 FFs are needed
  - Will use D FF

Example 1

- State assignment
  - The simplest state assignment is to follow the binary sequence starting from 000 and ending in 100 (101, 110, and 111 are used as don't care).

	Present state $y_3y_2y_1$	Next state $w=0$ $Y_3Y_2Y_1$	Next state $w=1$ $Y_3Y_2Y_1$	Output z
A	000	001	011	0
B	001	010	011	0
C	010	010	011	1
D	011	001	100	0
E	100	001	100	1

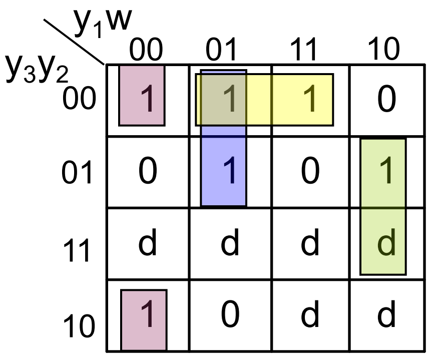
- Th :

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

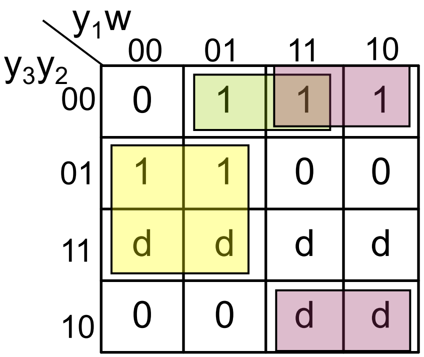
$\overline{y_1}\overline{y_2}$

- Th

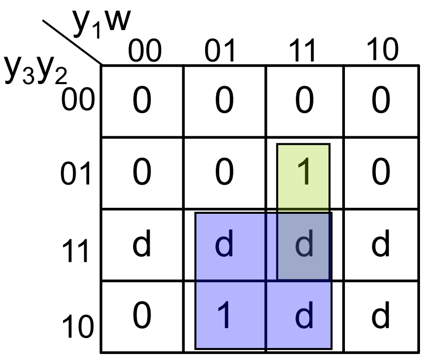
$$z = y_3 + \overline{y_1} y_2$$



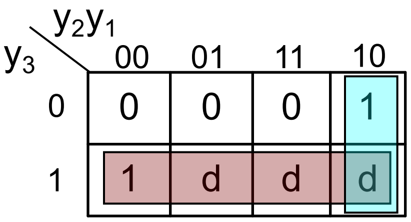
y_3y_2 \ y_1w	00	01	11	10
00	1	1	1	0
01	0	1	0	1
11	d	d	d	d
10	1	0	d	d



y_3y_2 \ y_1w	00	01	11	10
00	0	1	1	1
01	1	1	0	0
11	d	d	d	d
10	0	0	d	d



y_3y_2 \ y_1w	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	d	d	d	d
10	0	1	d	d



y_3 \ y_2y_1	00	01	11	10
0	0	0	0	1
1	1	d	d	d

Example 1

- Next state equations seem to be unnecessarily complex
- This denotes that there should be a better state assignment scheme
- Note that state A is only reached at the beginning when the machine is started or reset. Therefore, it is better to assign codes starting with 100 to 111 to states BCDE

	Present state	Next state		Output
		$w=0$	$w=1$	
	$y_3y_2y_1$	$Y_3Y_2Y_1$	$Y_3Y_2Y_1$	z
A	000	100	110	0
B	100	101	110	0
C	101	101	110	1
D	110	100	111	0
E	111	100	111	1

- The equations of the next_state are:

$$Y_1 = wy_2 + \overline{w}y_3\overline{y_2}$$

$$Y_2 = w$$

$$Y_3 = 1$$

- The output expression is: $z = y_1$

y_3y_2 \ y_1w	00	01	11	10
00	0	0	d	d
01	d	d	d	d
11	0	1	1	0
10	1	0	0	1

y_3y_2 \ y_1w	00	01	11	10
00	0	1	d	d
01	d	d	d	d
11	0	1	1	0
10	0	1	1	0

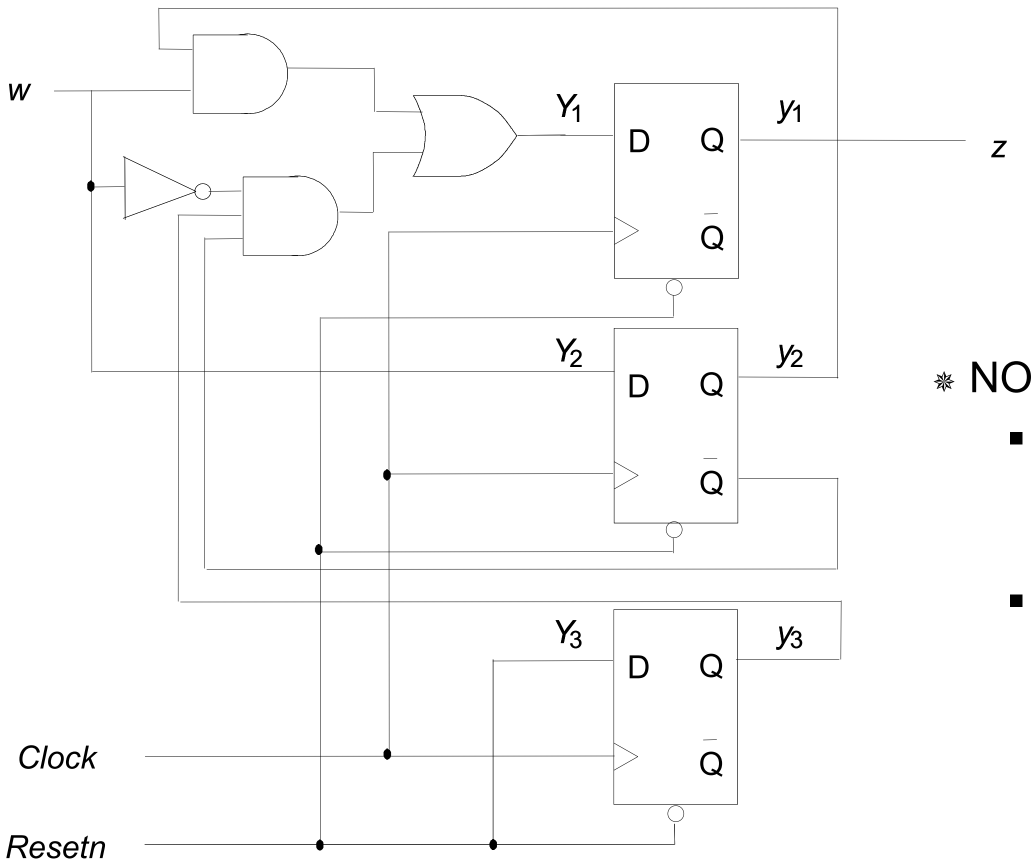
y_3y_2 \ y_1w	00	01	11	10
00	1	1	d	d
01	d	d	d	d
11	1	1	1	1
10	1	1	1	1

y_3 \ y_2y_1	00	01	11	10
0	0	d	d	d
1	0	1	1	0

- **This is a better solution!**

Example 1

- Implementation



$$Y_1 = wy_2 + \overline{w}y_3\overline{y_2}$$

$$Y_2 = w$$

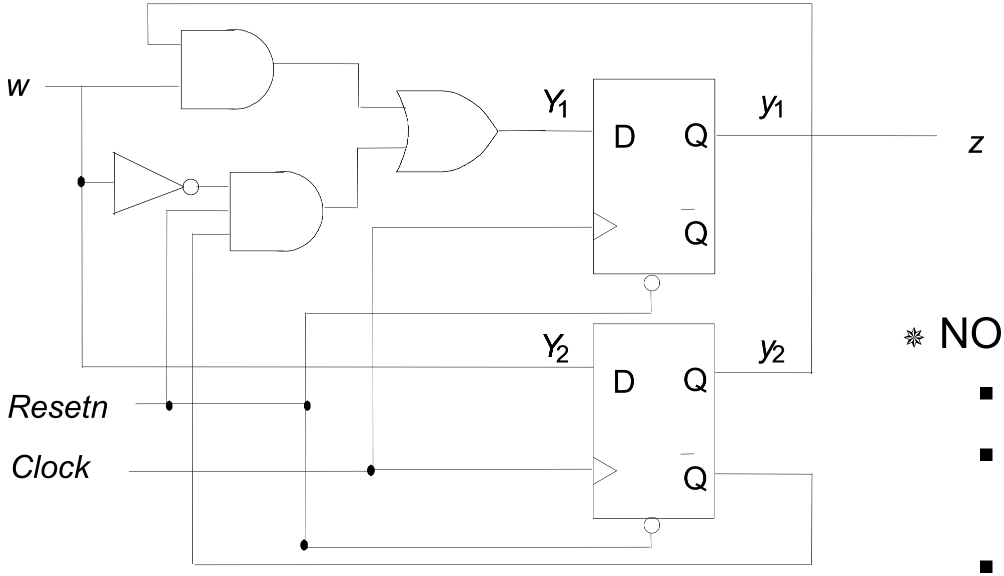
$$Y_3 = 1$$

$$z = y_1$$

- NOTE:
  - Y_3 is 1 all the time since D FF will output the input D at the positive edge of the clock
  - Can this help to further simplify provide?

Example 1

- Implementation



$$Y_1 = wy_2 + \overline{w}y_3\overline{y_2}$$

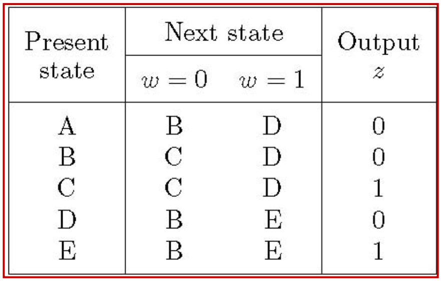
$$Y_2 = w$$

$$z = y_1$$

- NOTE:
  - Third FF can be eliminated
  - Reset input can be connected directly to the AND gate
  - This makes A a virtual state but … will make the machine to operate asynchronously on the first clock cycle after reset
  - This is more a practical solution since y_3 doesn't change after the reset

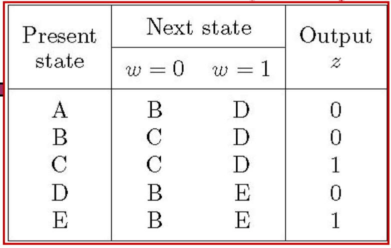
Example 1: VHDL implementation Automatic assignment

```
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
    TYPE State_type IS (A, B, C, D, E) ;
    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 <= B ;
                    ELSE
                        y <= D ;
                    END IF ;
                WHEN B =>
                    IF w = '0' THEN
                        y <= C ;
                    ELSE
                        y <= D ;
                    END IF ;
                WHEN C =>
                    IF w = '0' THEN
                        y <= C ;
                    ELSE
                        y <= D ;
                    END IF ;
                WHEN D =>
                    IF w = '0' THEN
                        y <= B ;
                    ELSE
                        y <= E ;
                    END IF ;
                WHEN E =>
                    IF w = '0' THEN
                        y <= B ;
                    ELSE
                        y <= E ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;
    z <= '1' WHEN y = C OR y = E ELSE '0' ;
END Behavior ;
```



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

Example 1: Alternative VHDL implementation Manual assignment



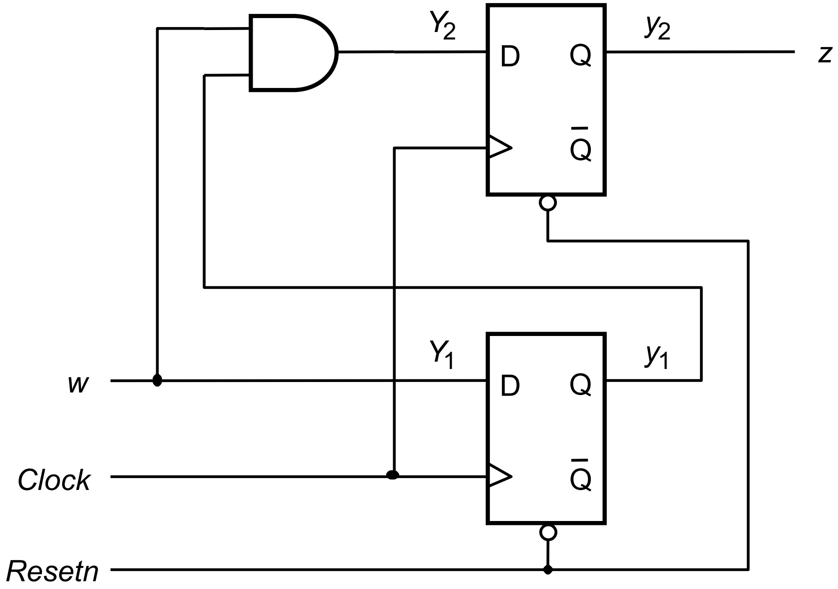
Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	B	D	0
B	C	D	0
C	C	D	1
D	B	E	0
E	B	E	1

```
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(2 DOWNTO 0);
    CONSTANT A :  STD_LOGIC_VECTOR(2 DOWNTO 0) := "000" ;
    CONSTANT B :  STD_LOGIC_VECTOR(2 DOWNTO 0) := "100" ;
    CONSTANT C :  STD_LOGIC_VECTOR(2 DOWNTO 0) := "101" ;
    CONSTANT D :  STD_LOGIC_VECTOR(2 DOWNTO 0) := "110" ;
    CONSTANT E :  STD_LOGIC_VECTOR(2 DOWNTO 0) := "111" ;
BEGIN
    PROCESS ( w, y_present )
    BEGIN
        CASE y_present IS
            WHEN A =>
                IF w = '0' THEN y_next <= B ;
                ELSE y_next <= D ;
                END IF ;
            WHEN B =>
                IF w = '0' THEN y_next <= C ;
                ELSE y_next <= D ;
                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 <= B ;
                ELSE y_next <= E ;
                END IF ;
            WHEN E =>
                IF w = '0' THEN y_next <= B ;
                ELSE y_next <= E ;
                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 <= '1' WHEN y_present = C OR y_present = E  ELSE '0' ;
END Behavior;
```

Example 2

- Design the previous FSM combining two smaller FSMs such that one detects the 11 sequence and the other the sequence of 00
- **Solution:** The state table and circuit for the FSM that detects 11 were introduced in previous lecture

	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	11	0
C	11	00	11	1
	10	*dd*	*dd*	*d*



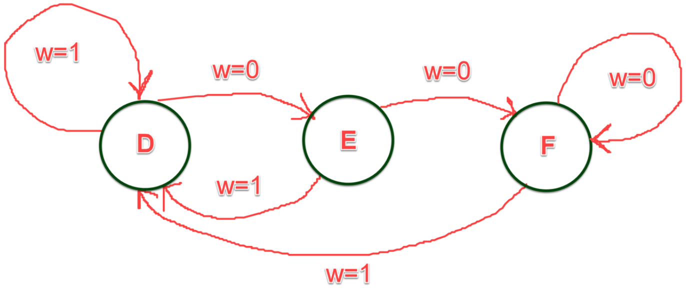
The expressions for this FSM are:

$$Y_1 = w \qquad z_{ONES} = y_2$$

$$Y_2 = wy_1$$

Example 2

- The state table for the FSM that detects 00 is:



Present state	Ne xt state		Output
	$w = 0$	$w = 1$	z_{zeros}
D	E	D	0
E	F	D	0
F	F	D	1

	Present state	Next state		Output
		$w = 0$	$w = 1$	z_{zeros}
	y_4y_3	Y_4Y_3	Y_4Y_3	
D	00	01	00	0
E	01	11	00	0
F	11	11	00	1
	10	dd	dd	d

The expressions for this FSM are:

$$Y_3 = \overline{w}$$

$$Y_4 = \overline{w}y_3$$

$$z_{ZEROS} = y_4$$

w \ y_4 y_3	00	01	11	10
0	1	1	1	d
1	0	0	0	d

w \ y_4y_3	00	01	11	10
0	0	1	1	d
1	0	0	0	d

y_4 \ y_3	0	1
0	0	0
1	d	1

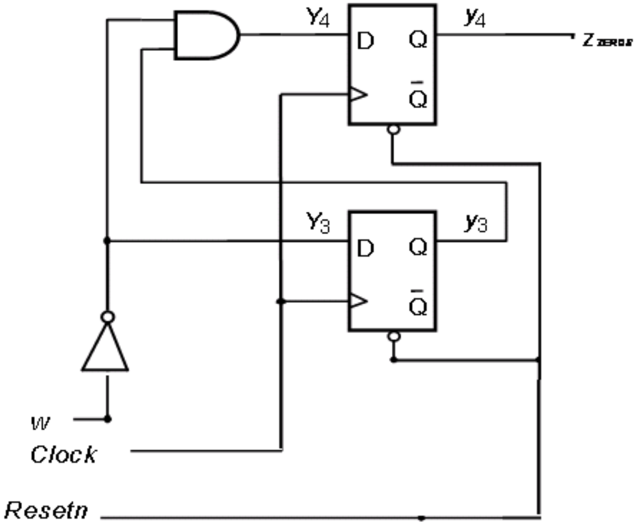
Example 2

- The circuit expression is: Z = $Z_{ONES} + Z_{ZEROS}$

$$Y_3 = \overline{w}$$

$$Y_4 = \overline{w} y_3$$

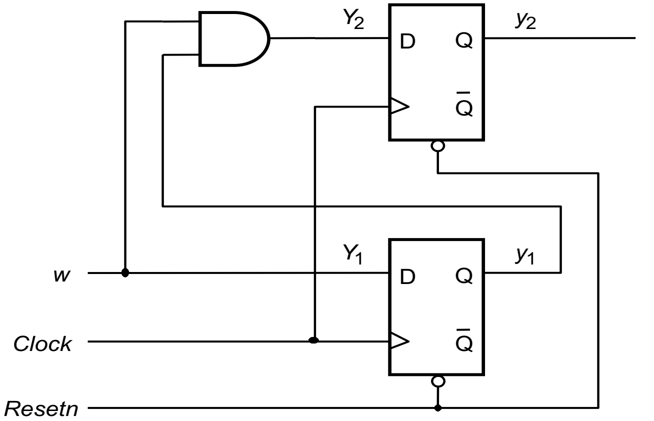
$$z_{ZEROS} = y_4$$

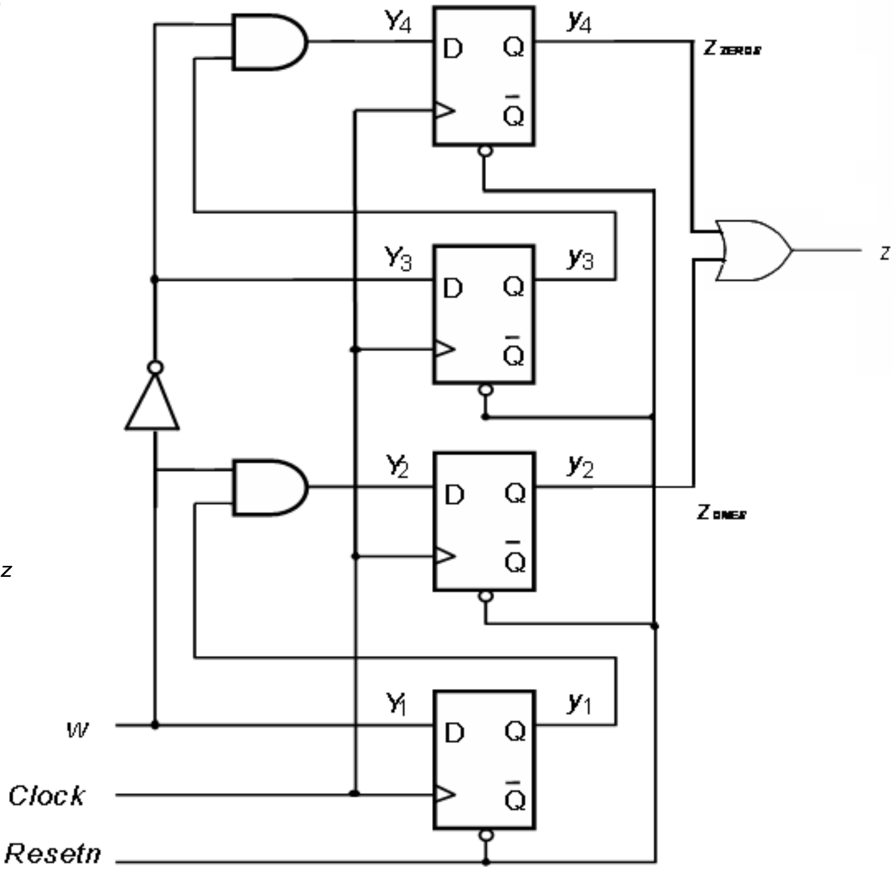


$$Y_1 = w$$

$$Y_2 = w y_1$$

$$z_{ONES} = y_2$$

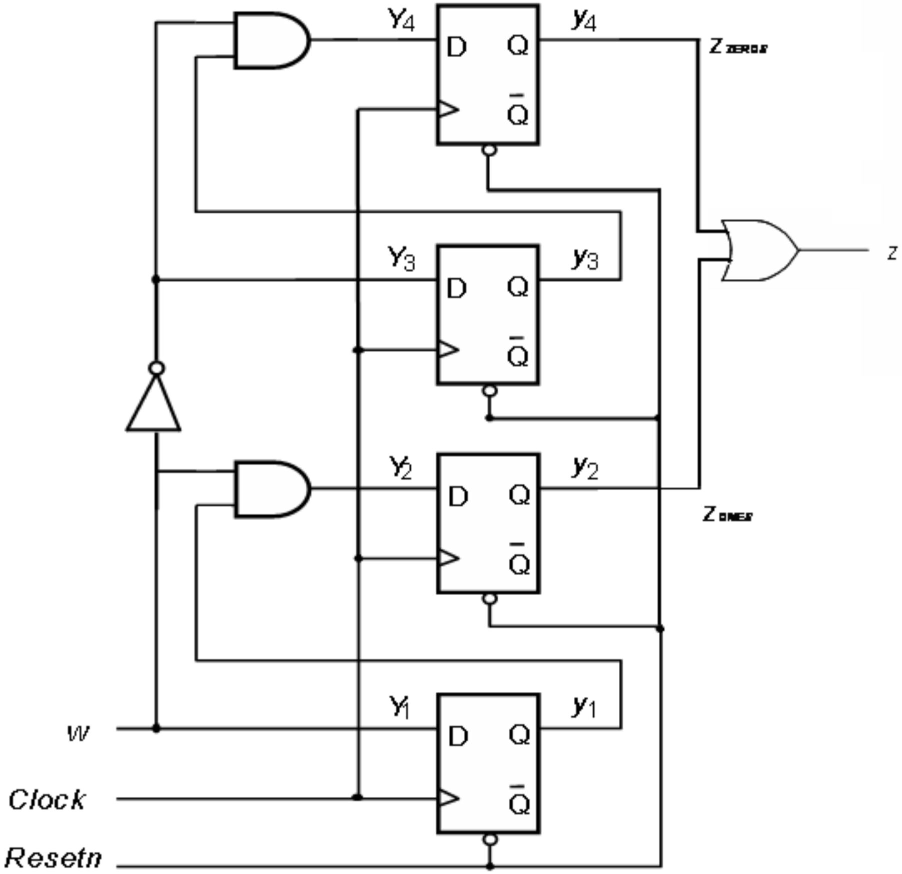




Example 2

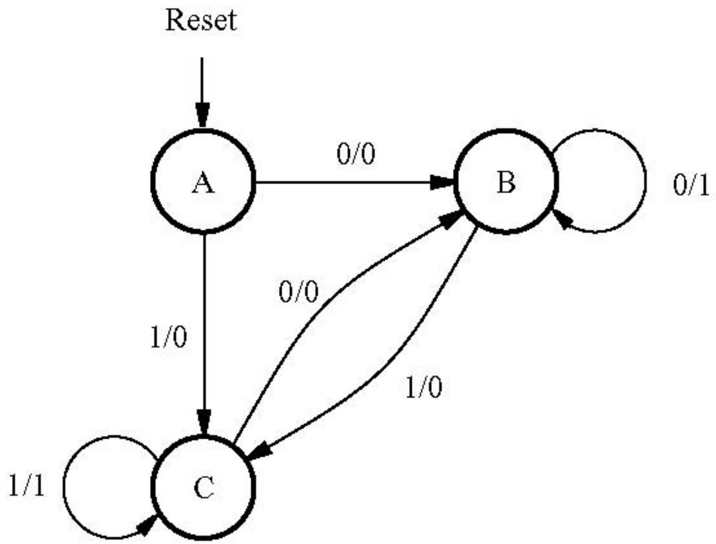
- The circuit expression is: $Z = Z_{ONES} + Z_{ZEROS}$

- Note that combination of two smaller and more specific FSMs can be performed to reach more complex FSM
- In this case, this approach leads to more expensive implementation than the whole sequence designed in only one FSM
- However, the design process is simpler
- Sharing circuits can lead to more cost-efficient implementations despite it might not be the most cost efficient implementation for a particular sequence



Example 3

- Design a Mealy-type FSM that detects the sequence 11 and 00 as in example 1 and write the VHDL code to implement the example in QUARTUS II
- **Solution:** The state diagram, state table and circuit for the Mealy FSM are:



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

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

Example 3

- This assignment leads to the next-state and output expressions:

$$Y_1 = 1$$

$$Y_2 = w$$

$$z = \overline{w} y_1 \overline{y_2} + w y_2$$

	Present state	Next state		Output	
		$w=0$	$w=1$	$w=0$	$w=1$
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	z	z
A	00	01	11	0	0
B	01	01	11	1	0
C	11	01	11	0	1

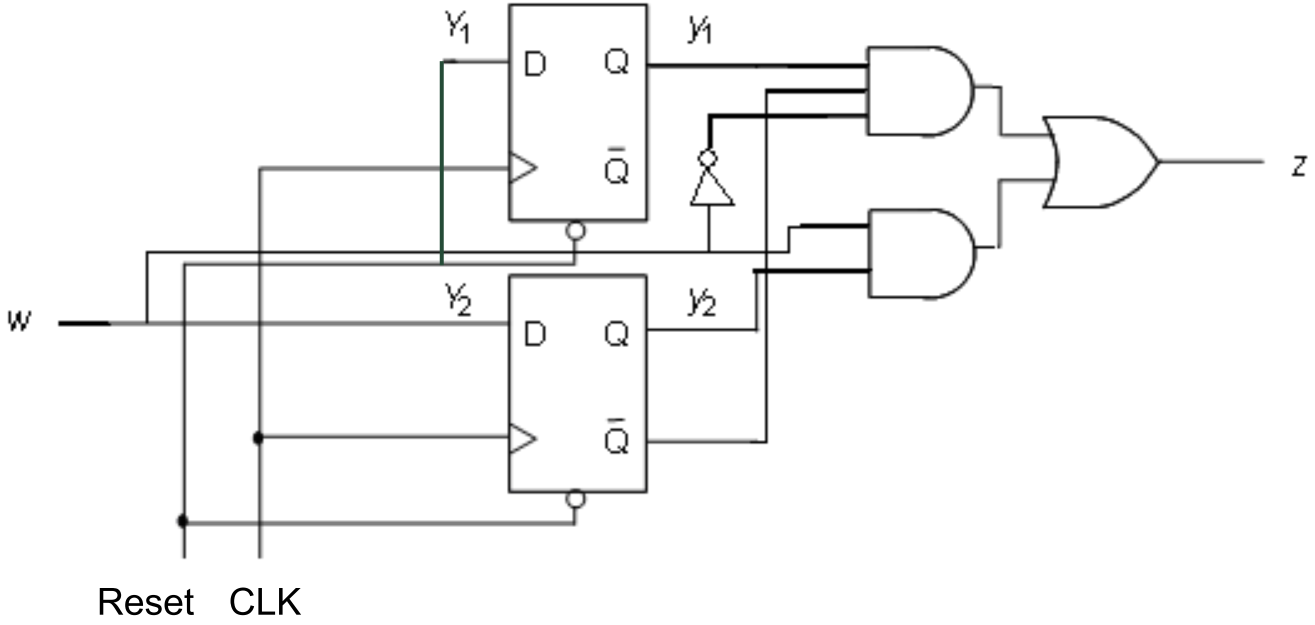
- Note that there are great savings using the input to determine the output
- Implementation changes from 3 FF in the Moore machine to 2 FF in the Mealy machine
- Logic cost is the same

w \ y_2y_1	00	01	11	10
0	1	1	1	d
1	1	1	1	d

w \ y_2y_1	00	01	11	10
0	0	0	0	d
1	1	1	1	d

w \ y_2y_1	00	01	11	10
0	0	1	0	d
1	0	0	1	d

Example 3 implementation



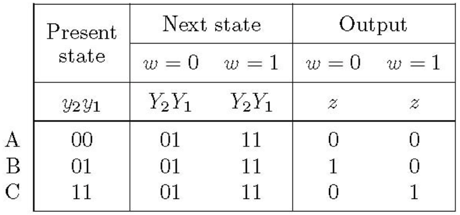
$$Y_1 = 1$$

$$Y_2 = w$$

$$z = \overline{w} y_1 \overline{y_2} + w y_2$$

Example 3: VHDL implementation Automatic assignment

```
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, 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 <= B ;
                    ELSE y <= C ;
                    END IF ;
                WHEN B =>
                    IF w = '0' THEN y <= B ;
                    ELSE y <= C ;
                    END IF ;
                WHEN C =>
                    IF w = '0' THEN y <= B ;
                    ELSE  y <= C;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;

    PROCESS ( y, w )
    BEGIN
        CASE y IS
            WHEN A =>
                z <= '0' ;
            WHEN B =>
                z <= NOT w ;
            WHEN C =>
                z <= w ;
        END CASE ;
    END PROCESS ;
END Behavior ;
```



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

Summary

- Example 1
  - Design of a sequence detector with a Moore-type FSM
  - Schematic and VHDL implementations
- Example 2
  - Design of a sequence detector FSM combining smaller Moore-type FSMs
- Example 3
  - Design of a sequence detector with a Mealy-type FSM
  - Schematic and VHDL implementations
- Recommended reading
  - Fundamentals of Digital Logic with VHDL Design 2nd Edition Stephen Brown, Zvonko Vranesic; McGraw-Hill, 2005. Chapter 8, pp. 496 – 535
- Next lecture
  - Design of counter using sequential circuit approach
    - Fundamentals of Digital Logic with VHDL Design 3/e, Stephen Brown, Zvonko Vranesic; McGraw-Hill, 2009. Chapter 8, pp. 535 - 553