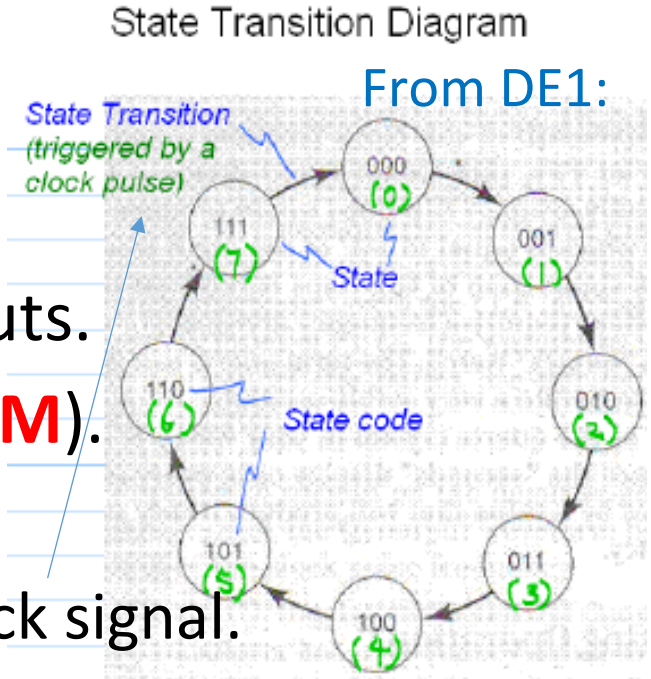
Chapter 3 - Synchronous Sequential Logic System



Sequential Logic Circuit:

- Outputs depend on current state & inputs.
- Also known as **Finite State Machine** (**FSM**).
- **Synchronous** sequential logic circuit:
  - Changes states **only at PGT/NGT** of clock signal.
  - Contains synchronised flip-flops for keeping track of its states.
  - Simple examples: synchronous counter, shift register etc.
  - Great majority of sequential logic circuits are synchronous.

(Driven by a common clock)

- **Asynchronous** sequential logic circuit: (Not covered in this module.)
  - No clock signal, output feedback to input.
  - A simple example: NAND gate latch.

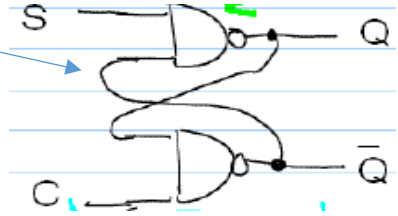
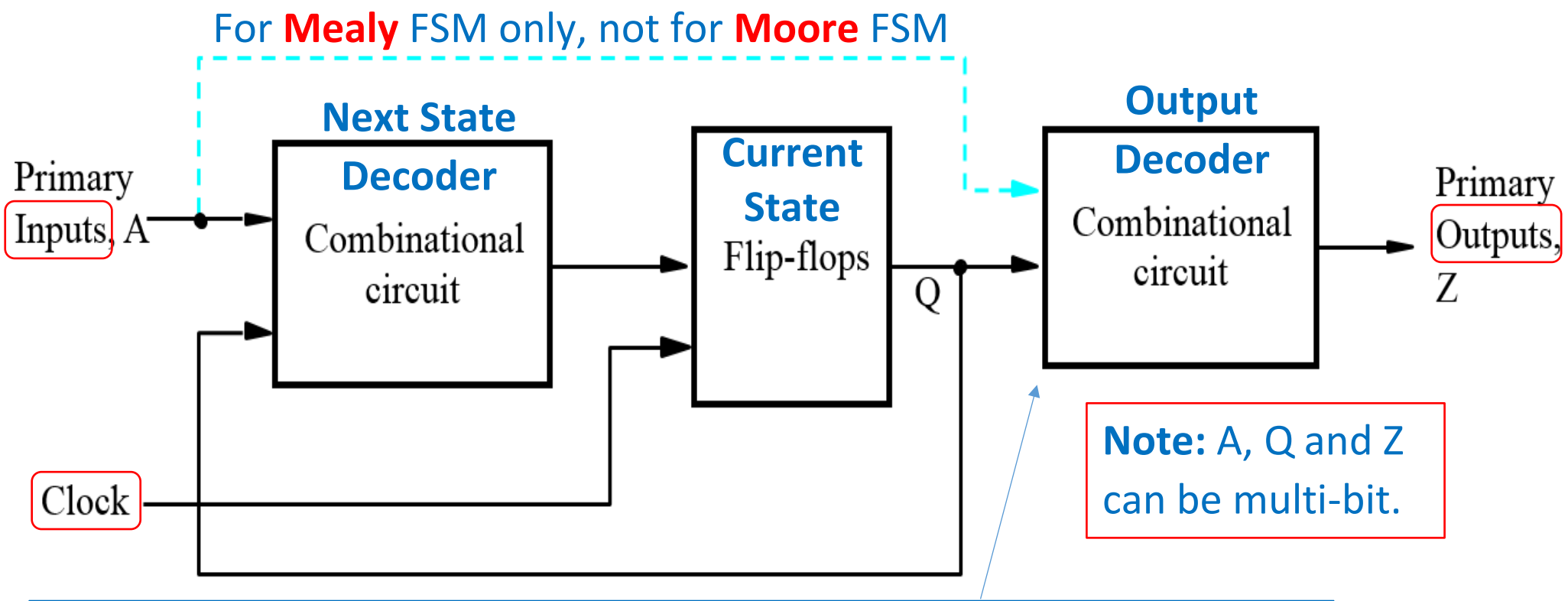


Figure 1 (Notes 3-1)

General block diagram of a synchronous sequential circuit



Moore type FSM – outputs depend on current state only.

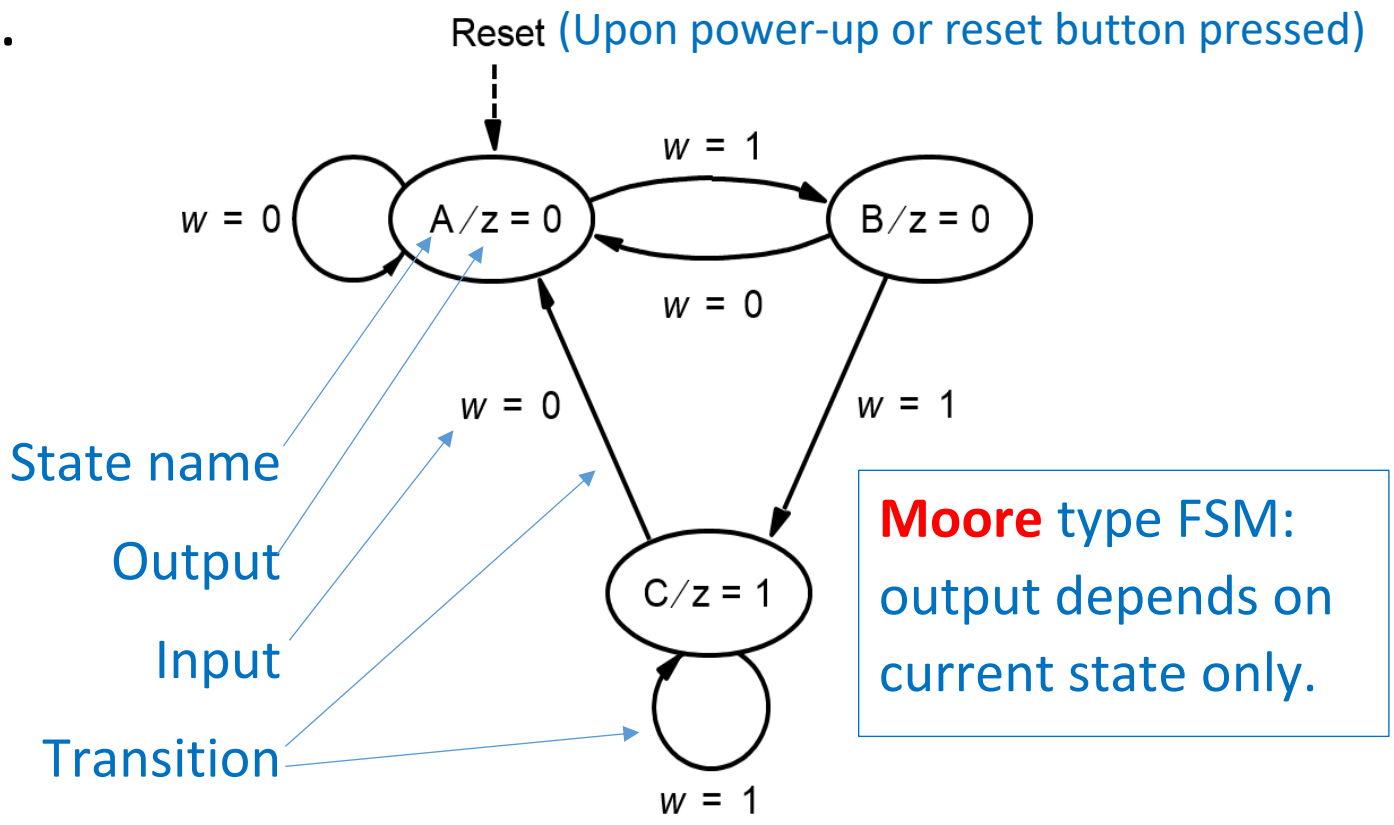
Mealy type FSM – outputs depend on current state as well as inputs.

Example (Notes 3-2)

Automatic vehicle speed controller:

- Input w=1: indicates over-speed.
- Output z=1: apply brake.
- Apply brake (z=1) if there are 2 consecutive measurements of over-speed (w=1).

2.1 - State Diagram:
(Notes 3-3)

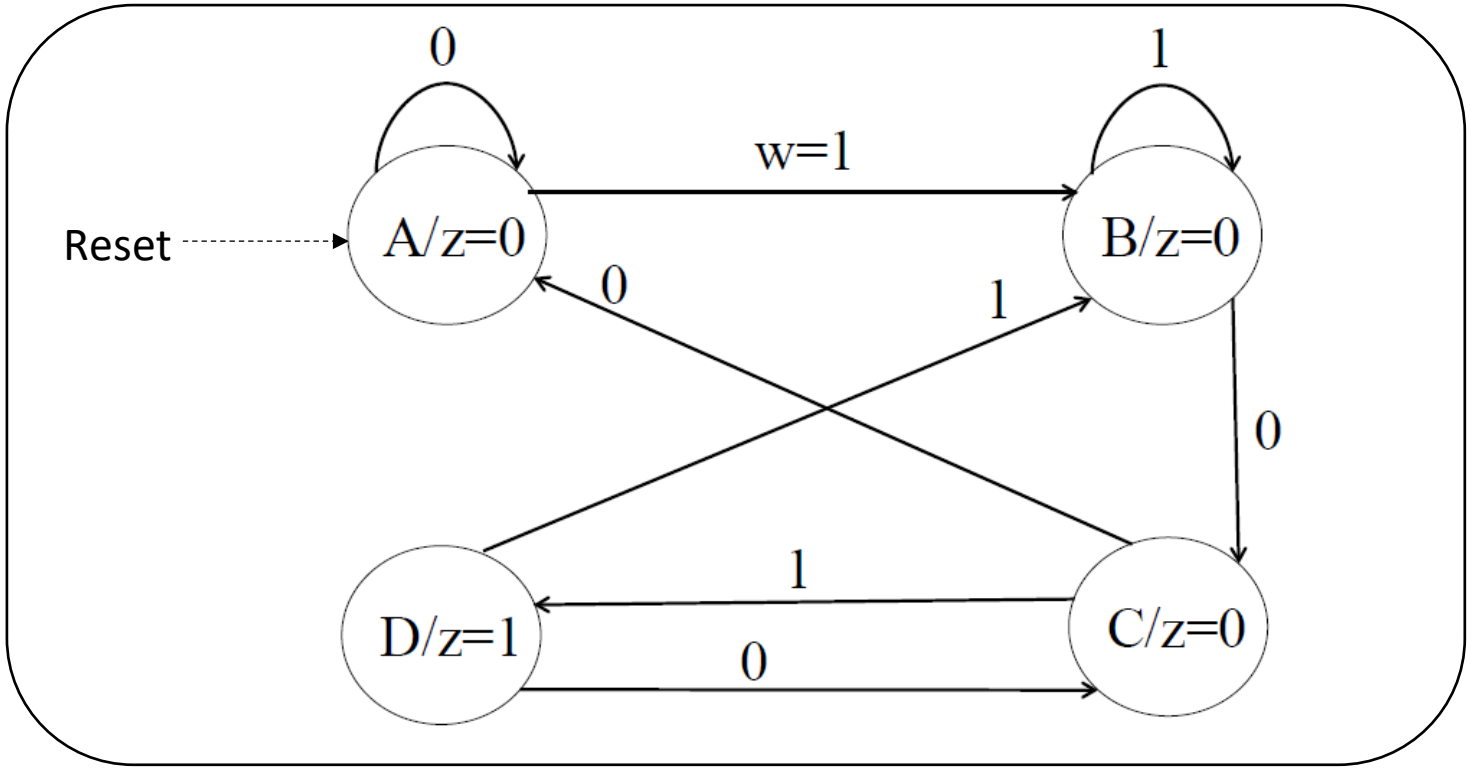


Example 1 (Notes 3-4)

Draw the state diagram for a FSM with the following specifications:

1. The circuit has one input, w, and one output, z.
2. All changes occur on positive edges of the clock signal.
3. Output z=1 if the input sequence for w is 101 for any three consecutive clock cycles. Otherwise, z=0.

Solution:



2.2 State Table (Notes 3-5)

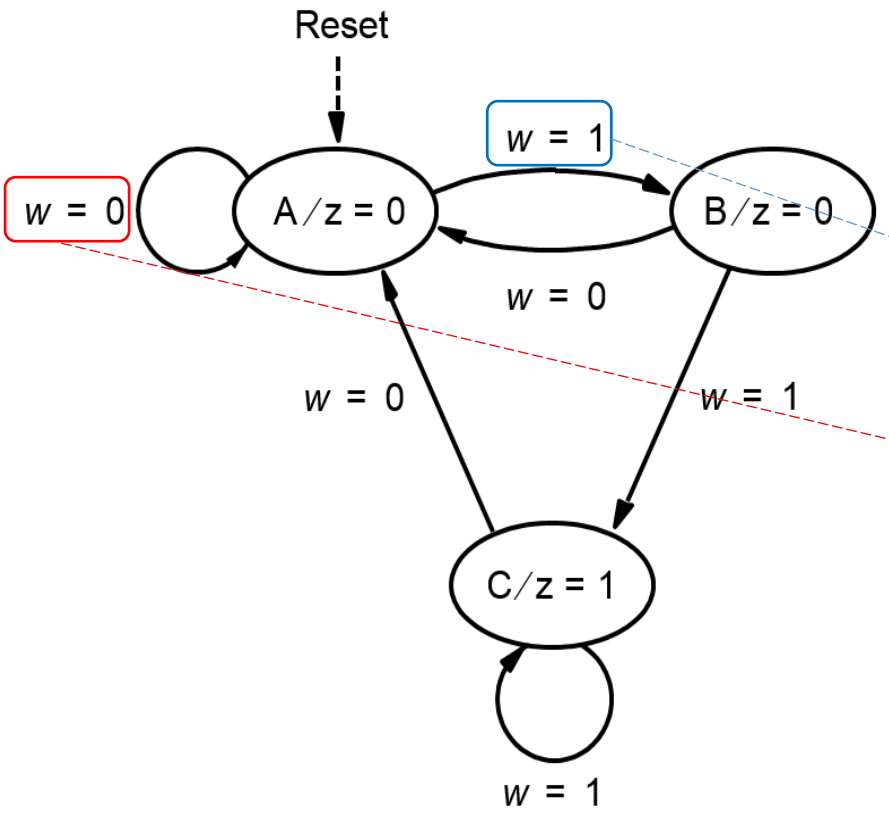
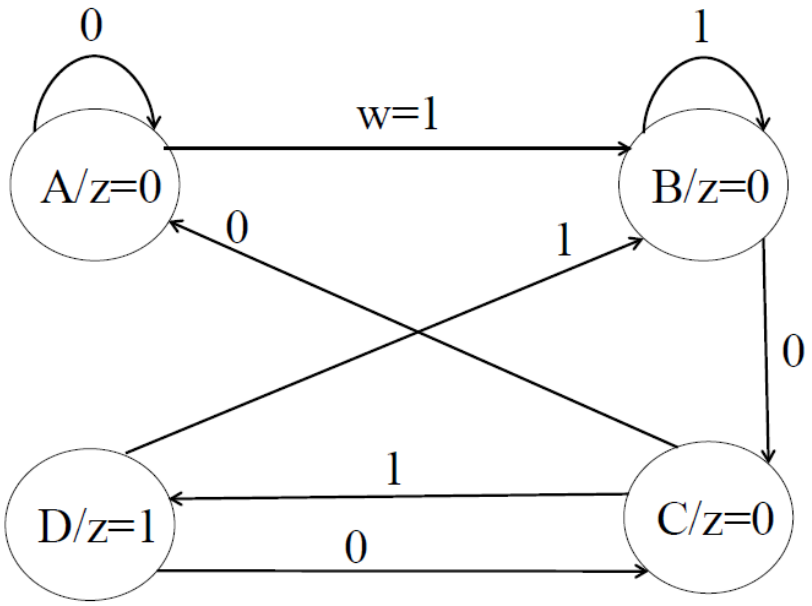


Fig 3.4 State table (for the State Diagram in Fig 3.3)

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

Example 2 (Notes 3-5)

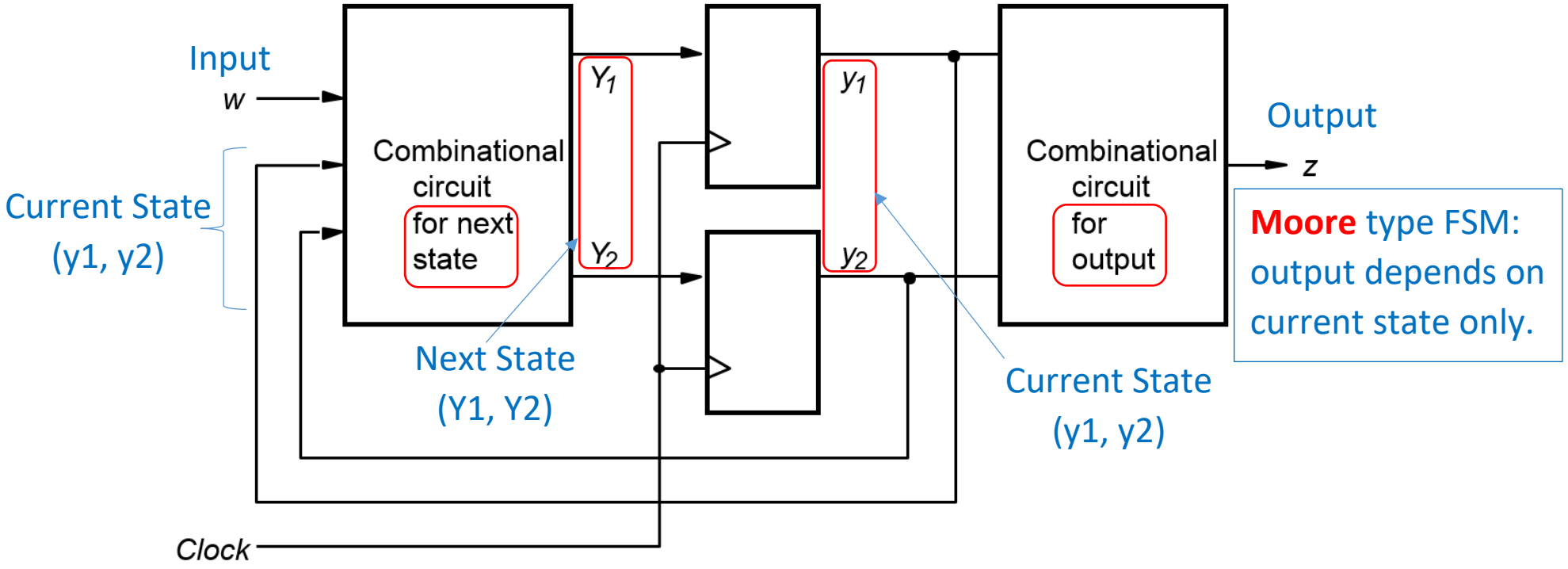


State table for the State Diagram in Ex 1:

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

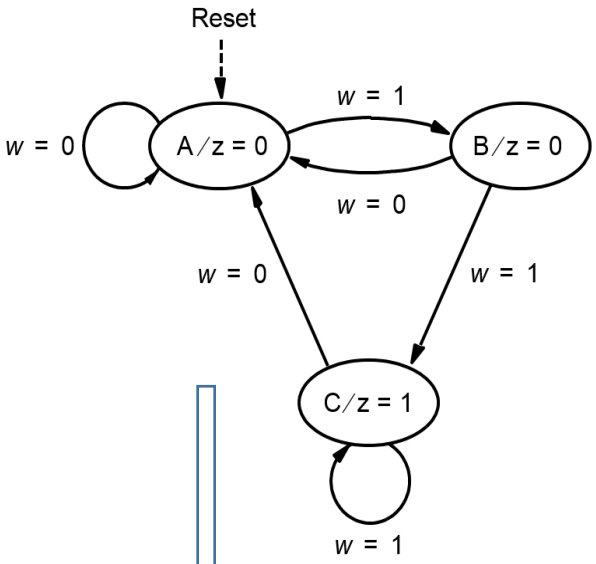
2.3 State Assignment (Notes 3-6)

- Assign a unique N-bit binary number for each state.
- Each bit (state variable) is stored at the Q output of a flip-flop.
- **Binary state assignment** is simple but not most efficient.
- Proper state assignment leading to simpler circuit will be covered in Section 7 (Notes 3-24).



Example:

Fig 3.3 State Diagram



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

Fig 3.4 State table

Fig 3.6 State-assigned table

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

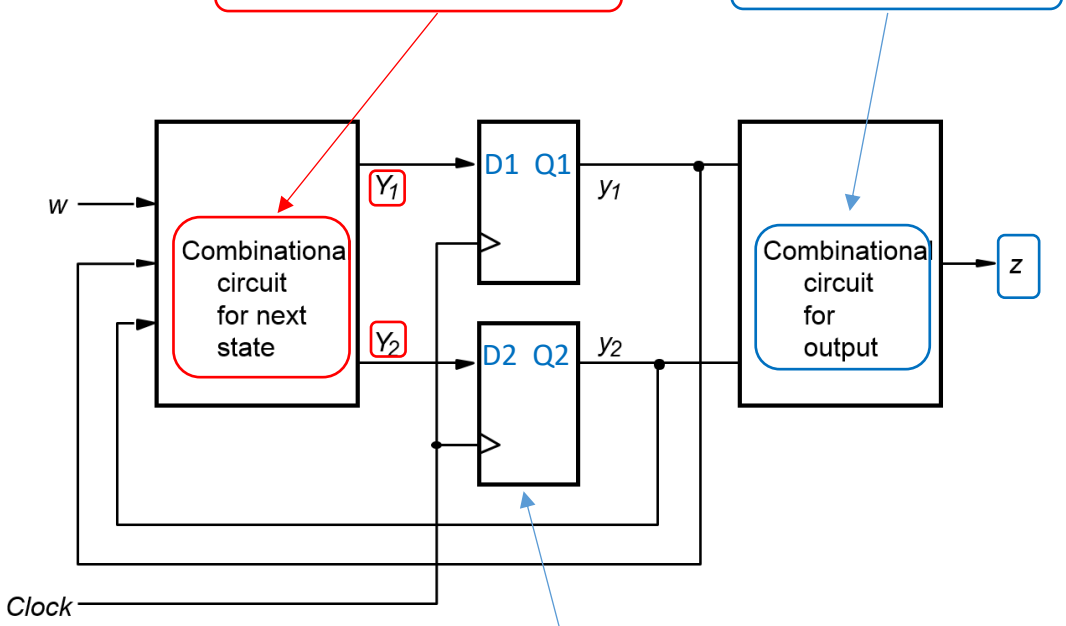
Assign any 3 binary numbers to the 3 states (A, B, C).

Assign “don’t care” (*x* or *d*) to the unused state for “**next state**” and “**output**” in order to simplify their circuits.

2.4 Excitation Table (Notes 3-7)

Excitation table is basically the truth table of the next state circuit and output circuit:

	Present state y_2y_1 (Q2 Q1)	Next state $w=0$: Y_2Y_1 (D2 D1)	Next state $w=1$: Y_2Y_1 (D2 D1)	Output z
A	00	A 00	B 01	0
B	01	A 00	C 10	0
C	10	A 00	C 10	1
	11	dd	dd	d



Implementing with D flip-flops.

y_2 \ y_1	0	1
0	0	0
1	1	d

$z = y_2$

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

$Y_1 = D_1 = w\bar{y}_1\bar{y}_2$

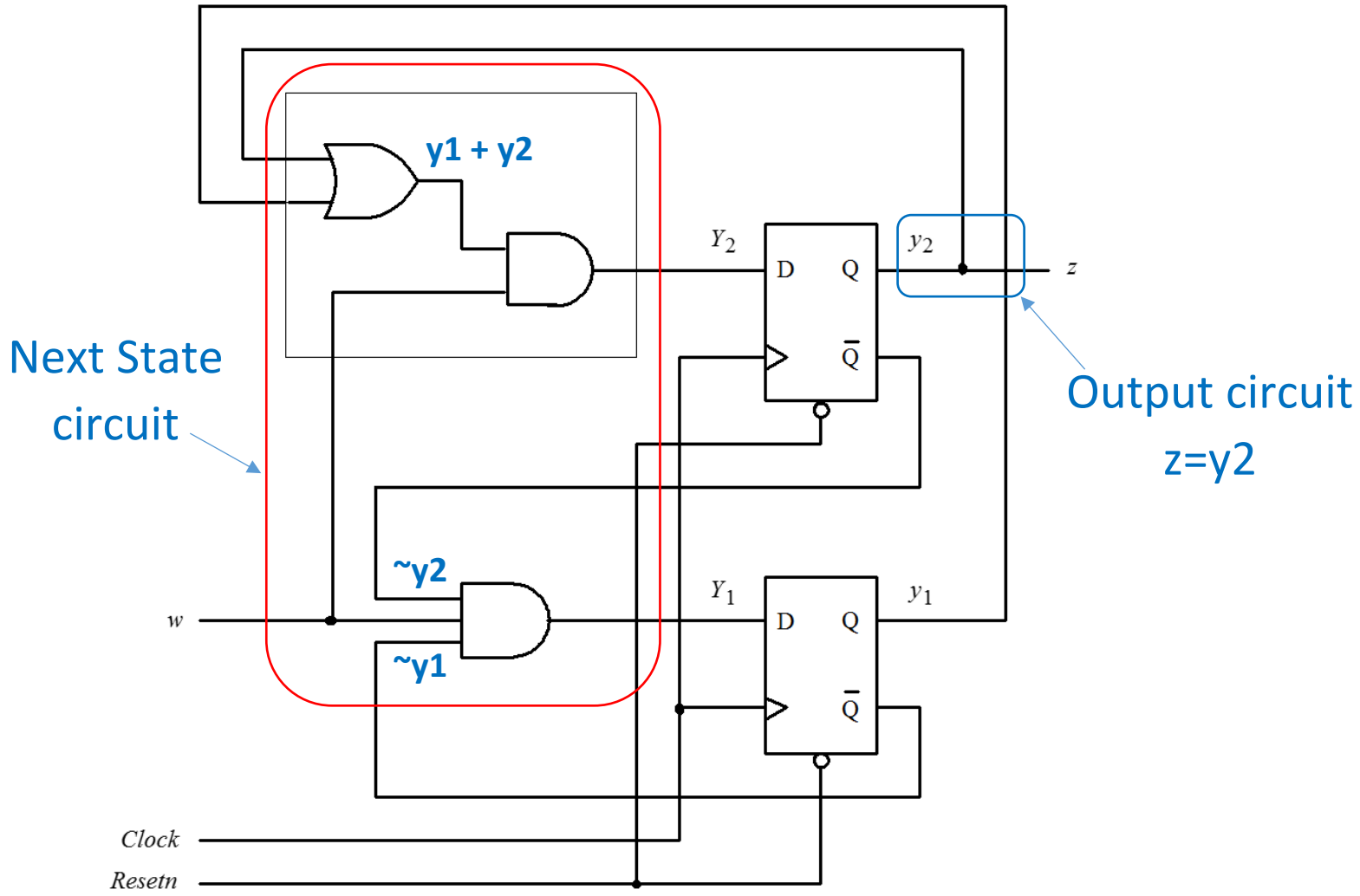
w \ y_2y_1	00	01	11	10
0	0	0	d	0
1	0	1	d	1

$Y_2 = D_2 = wy_1 + wy_2$ — 3 gates

$= w(y_1 + y_2)$ — 2 gates

Figure 9 (Notes 3-8)

"11" sequence detector implemented using D-type flip-flops:



While it is easier to design with D flip-flops, using J-K flip-flops often results in simpler logic for the “next state” decoder.

Table 1 - JK flip-flop truth table

J	K	$Q_{(t+1)}$
0	0	$Q_{(t)}$
0	1	0
1	0	1
1	1	$Q_{(t)}'$

Implies

Table 2 – Required JK values for different transitions in Q

$Q_{(t)}$	$Q_{(t+1)}$	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Next Q

Present Q

No change

Reset

Set

Toggle

Don’t care

For ref. only

Recall from DE1:

Notes 3-7

Present state y2y1	w=0 Next state Y_2Y_1	w=0 flip-flop input For Y2 J_2K_2	w=0 flip-flop input For Y1 J_1K_1	w=1 Next state Y_2Y_1	w=1 flip-flop input J_2K_2	w=1 flip-flop input J_1K_1	Output z
A 00	00 A	0d	0d	01 B	0d	1d	0
B 01	00 A	0d	d1	10 C	1d	d1	0
C 10	00 A	d1	0d	10 C	d0	0d	1
11	dd	dd	dd	dd	dd	dd	d

Figure 7. Excitation table for JK flip-flops. (Notes 3-7)

Some may prefer to present the above excitation table in the traditional truth table form:

For ref. only

y2 → Y2 in red

y1 → Y1 in blue

Present State		Primary Input	Next State		For y2→Y2		For y1→Y1		Primary Output
	y2	y1	w	Y2	Y1	J2	K2	J1	K1	Z
A	0	0	0	0	0	0	X	0	X	0
A	0	0	1	0	1	0	X	1	X	0
B	0	1	0	0	0	0	X	X	1	0
B	0	1	1	1	0	1	X	X	1	0
C	1	0	0	0	0	X	1	0	X	1
C	1	0	1	1	0	X	0	0	X	1
	1	1	0	X	X	X	X	X	X	X
	1	1	1	X	X	X	X	X	X	X

Inputs to the **next state** circuit.

The required **next state**.

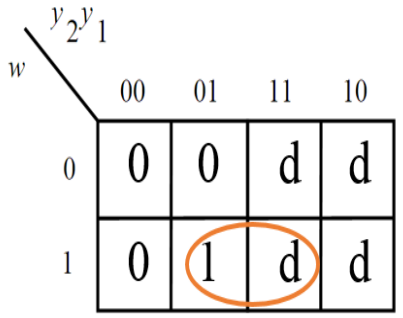
Transition

$Q_{(t)}$	$Q_{(t+1)}$	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

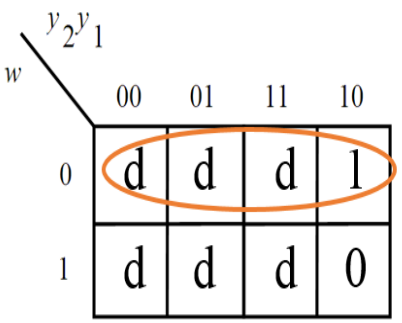
Notes 3-9

For ref. only

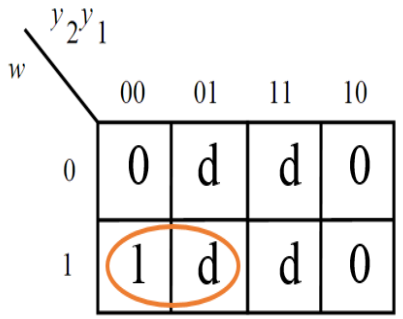
Figure 10 - K-map for J2, K2, J1 and K1



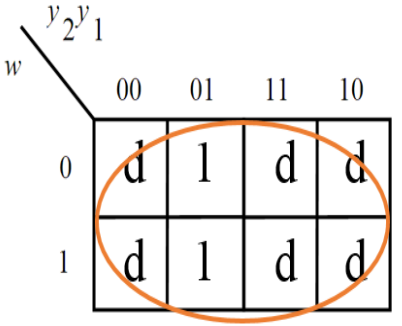
$J_2 = wy_1$



$K_2 = \overline{w}$



$J_1 = w\overline{y_2}$



$K_1 = 1$

Figure 11 - "11" sequence detector implemented using JK flip-flops

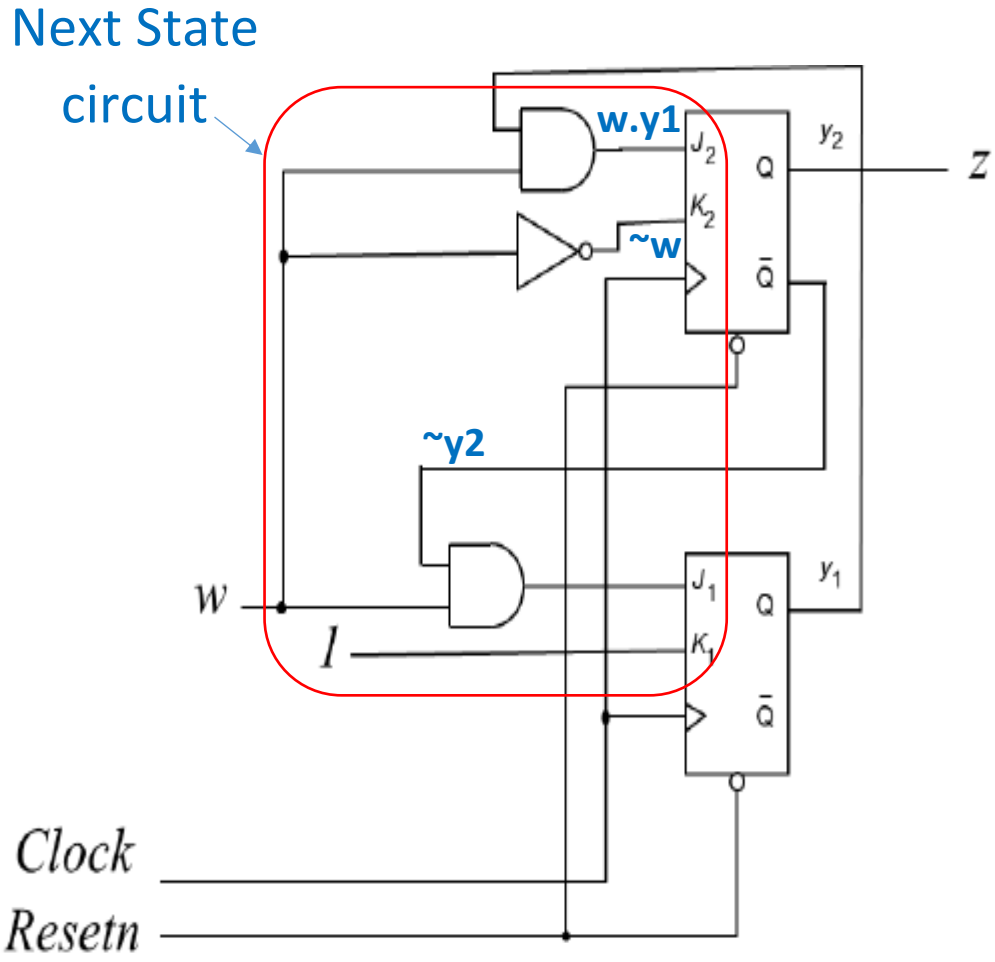
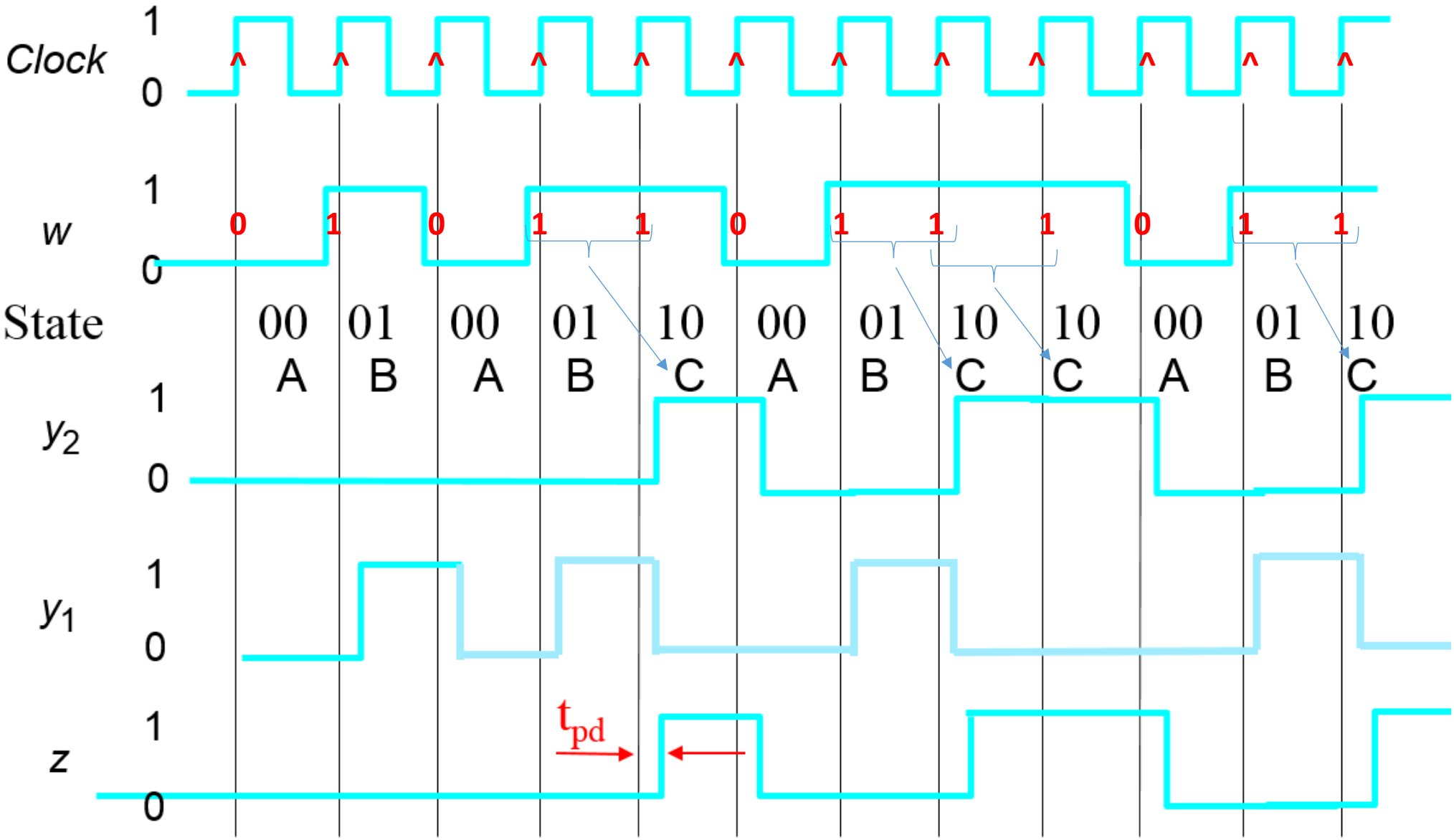
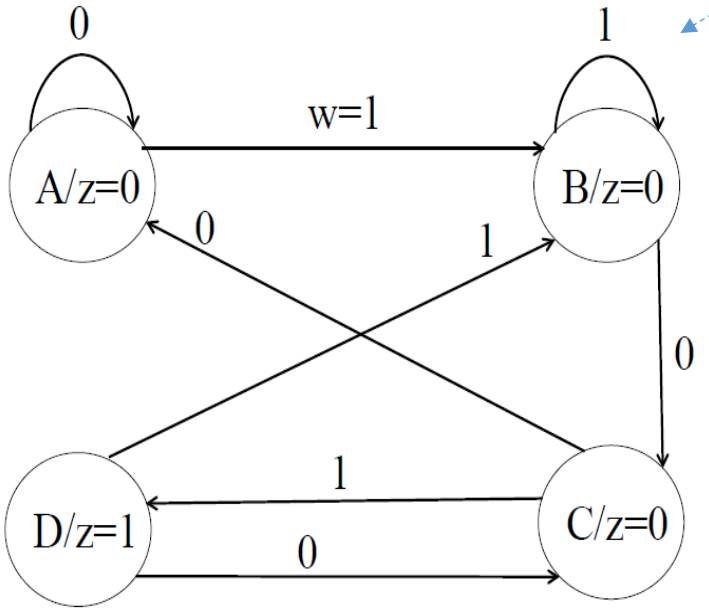


Figure 12 - Timing diagram of “11” sequence detector (Notes 3-10)



Example 3 (Notes 3-10)

Derive the state-assigned table for the FSM in Example 1 and hence draw the schematic diagram of the FSM using D-type flip-flops.



Present state $Q_2\,Q_1$ y_2y_1	Next state, Y_2Y_1 w=0 D2 D1	Next state, Y_2Y_1 w=1 D2 D1	Output z
A 00	A 00	B 01	0
B 01	C 10	B 01	0
C 10	A 00	D 11	0
D 11	C 10	B 01	1

Solution:

$$z = y_2y_1$$

$$D_2 = Y_2 = w'y_1 + wy_2y_1$$

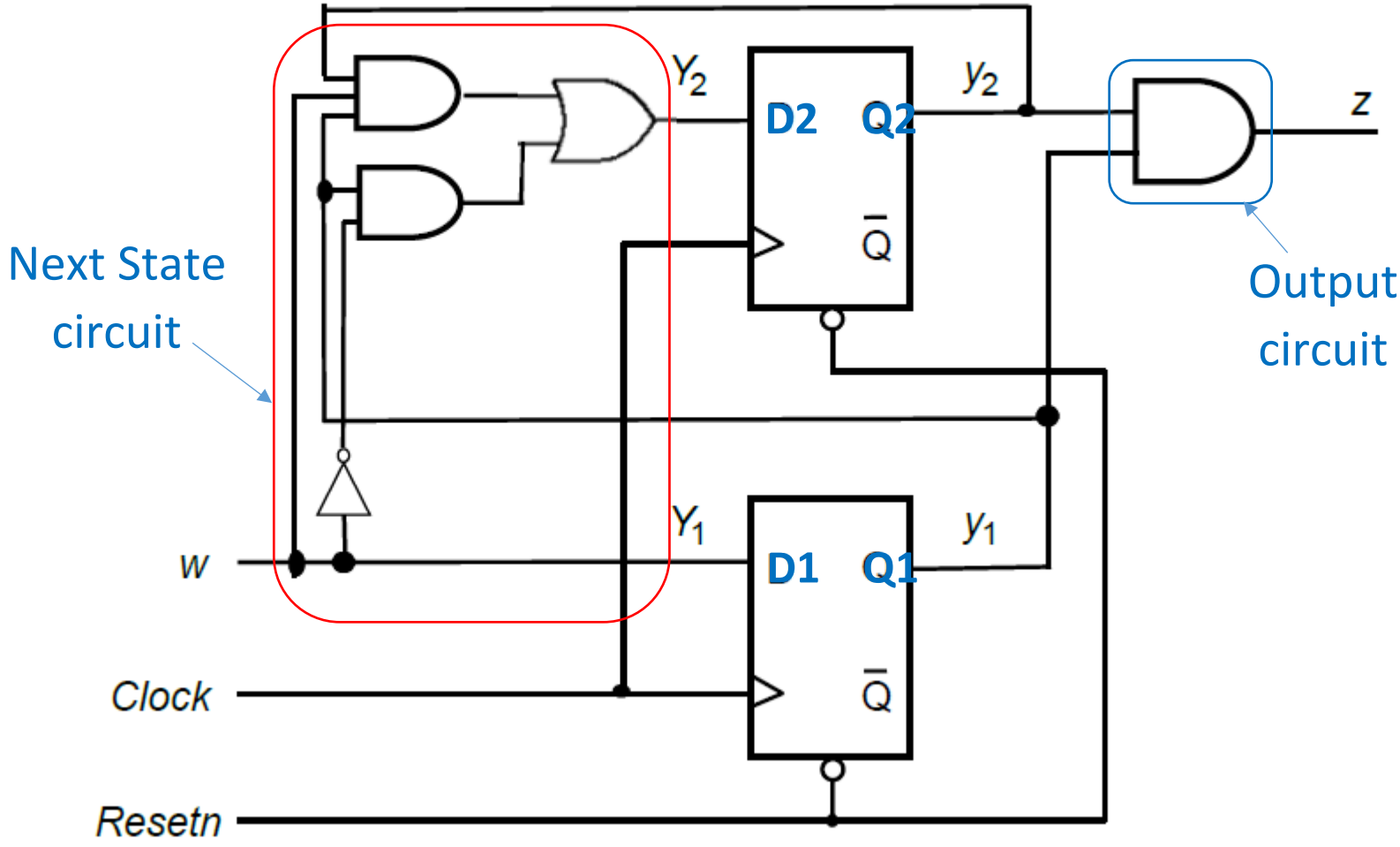
$$D_1 = Y_1 = w$$

D_2		w=0	w=1
y_2y_1	00	0	0
	01	1	0
	11	1	0
	10	0	1

D_1		w=0	w=1
y_2y_1	00	0	1
	01	0	1
	11	0	1
	10	0	1

Schematic diagram of the FSM using D-type flip-flops:

$z = y_2 y_1$ $\quad D_2 = Y_2 = w'y_1 + wy_2y_1$ $\quad D_1 = Y_1 = w$



2.6 Summary of FSM design procedure (Notes 3-11)

1. Obtain the specification of the desired FSM.
2. Draw the state diagram.
3. Derive the state table.
4. Assign state code.
5. Derive the state-assigned table and/or excitation table.
6. Derive the output & flip-flop input equations.
7. Implement the circuit from the equations derived.