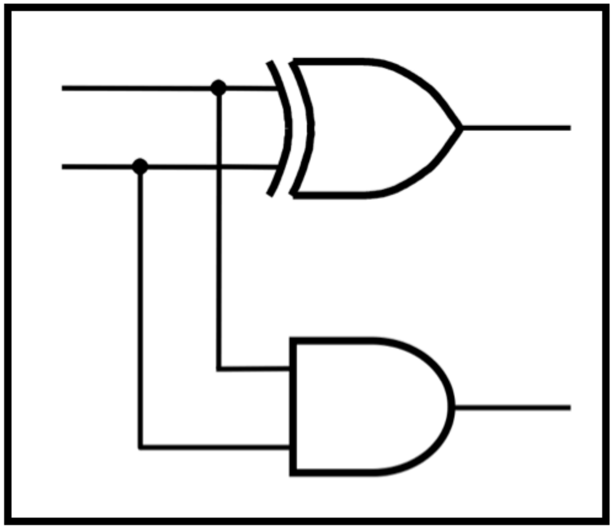


CprE 281: Digital Logic

Instructor: Alexander Stoytchev

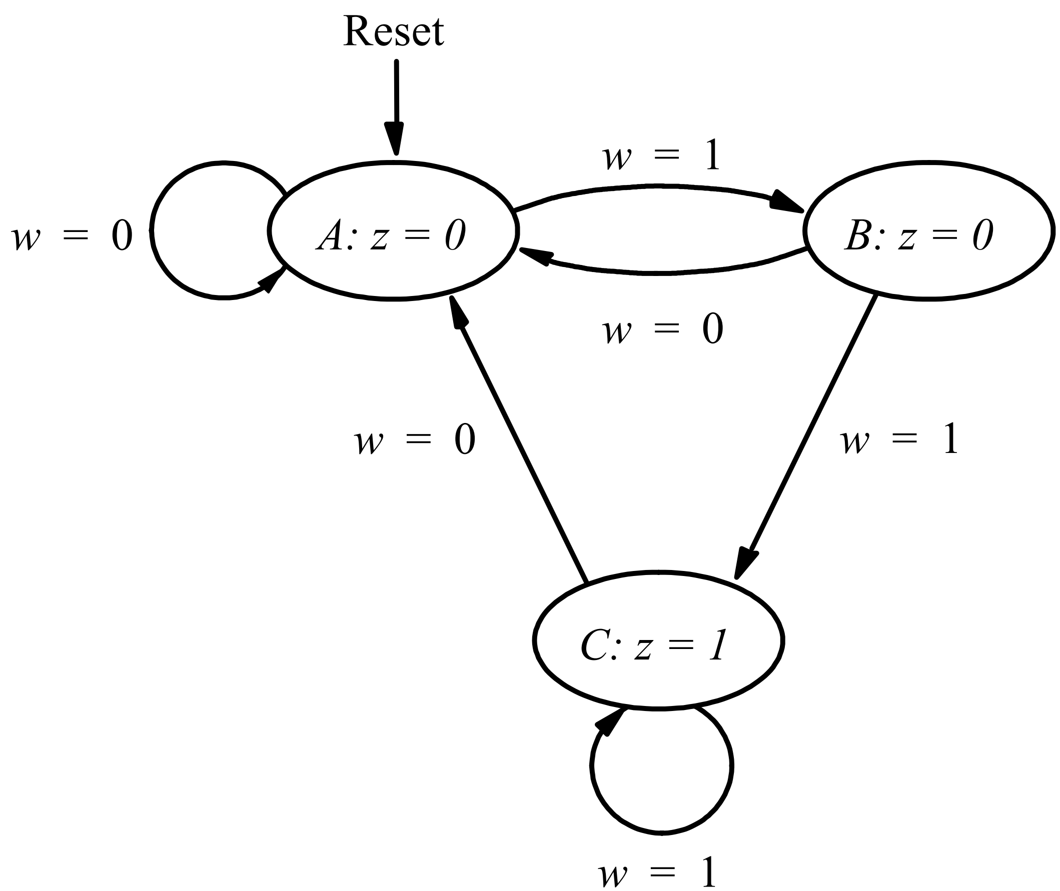
http://www.ece.iastate.edu/~alexs/classes/

Synchronous Sequential Circuits
Basic Design Steps

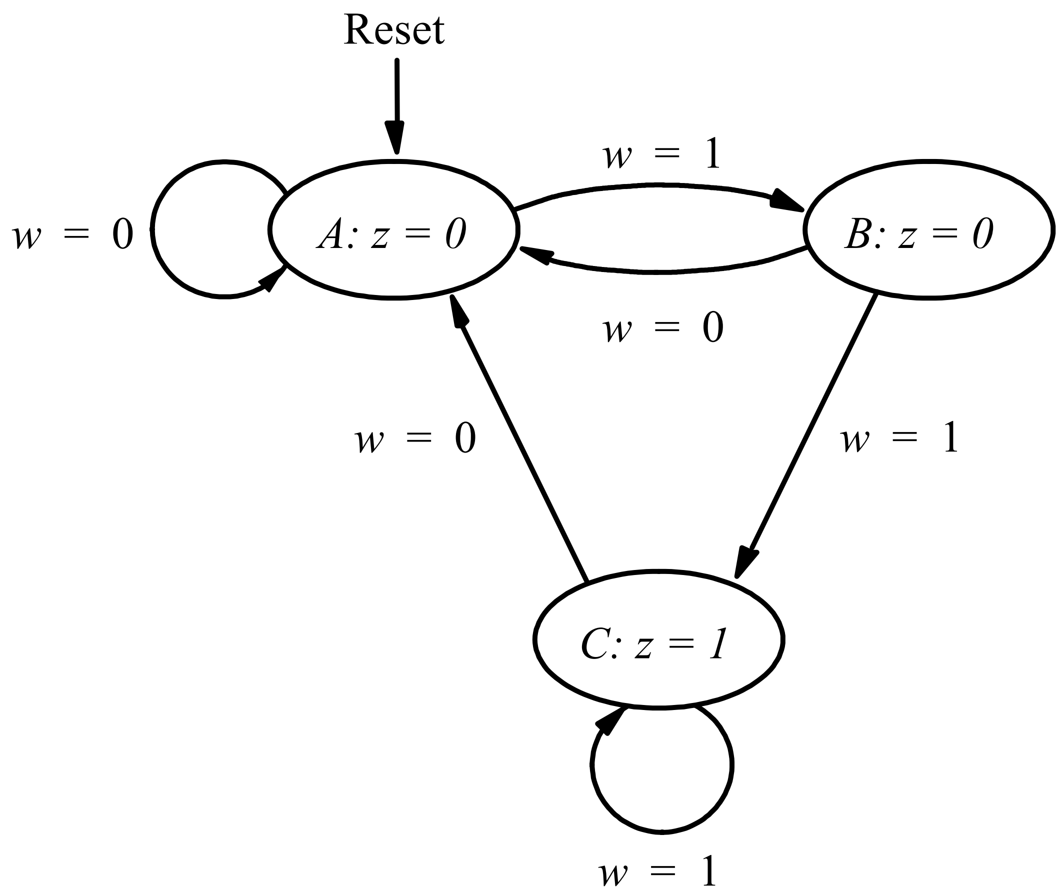
CprE 281: Digital Logic
Iowa State University, Ames, IA
Copyright © Alexander Stoytchev

First Design Pattern: Moore Machines

Moore Machine:
A Type of Finite State Machine (FSM)

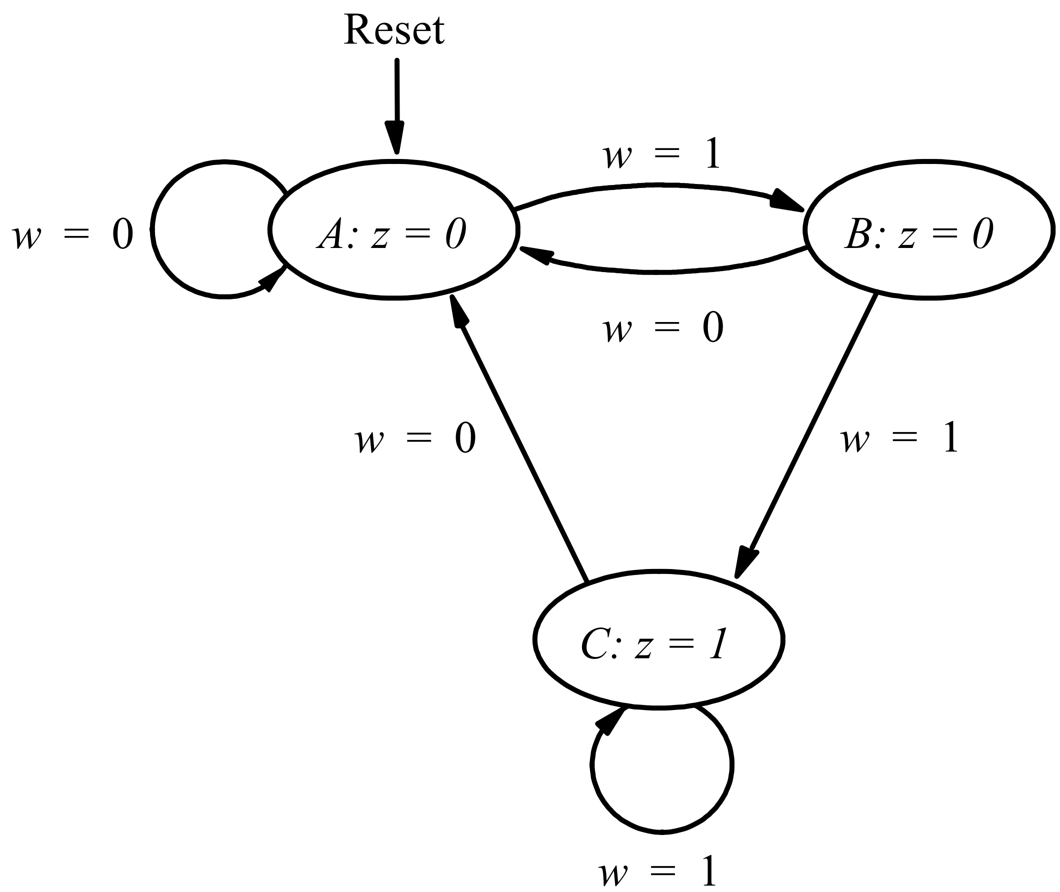


[Figure 6.3 from the textbook]



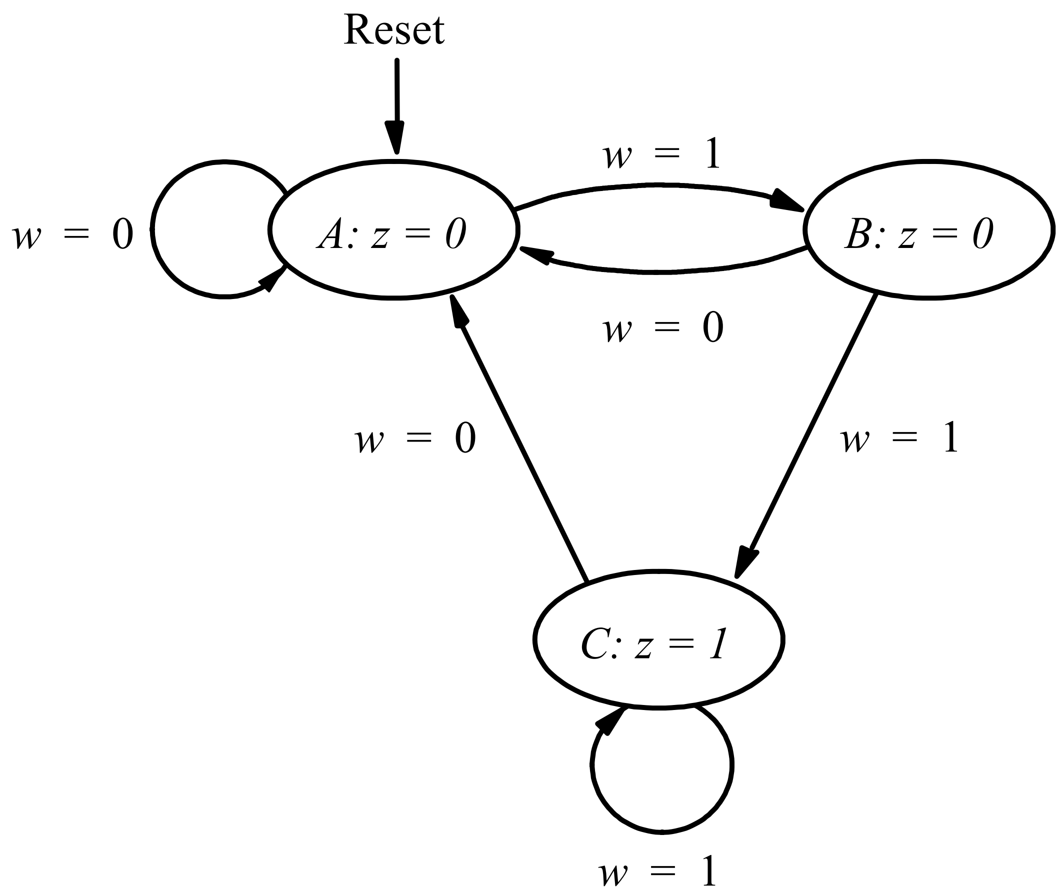
- Finite number of states (nodes).
- Discrete state transitions (edges).
- Only “in” one state at a time.
- One reset state
- Every state has an outgoing state transition for each possible input.

[Figure 6.3 from the textbook]



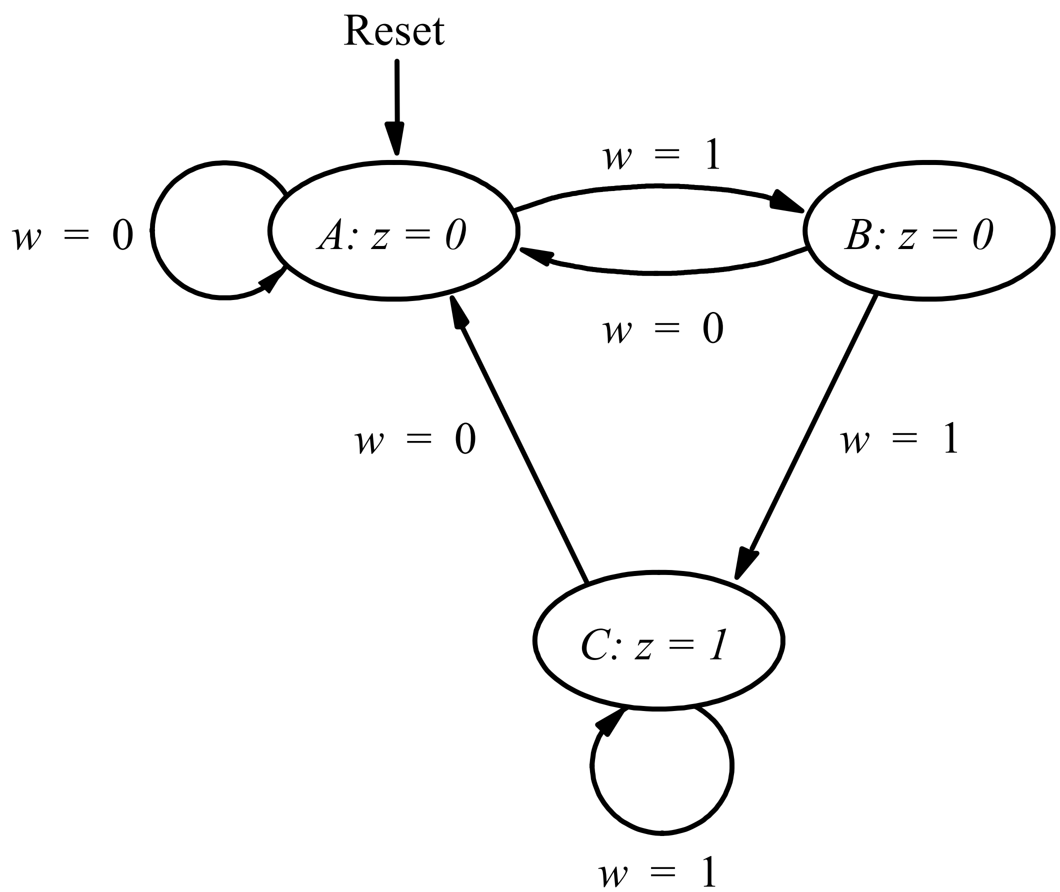
Key: The next state depends on both the current state and the current input.

[Figure 6.3 from the textbook]

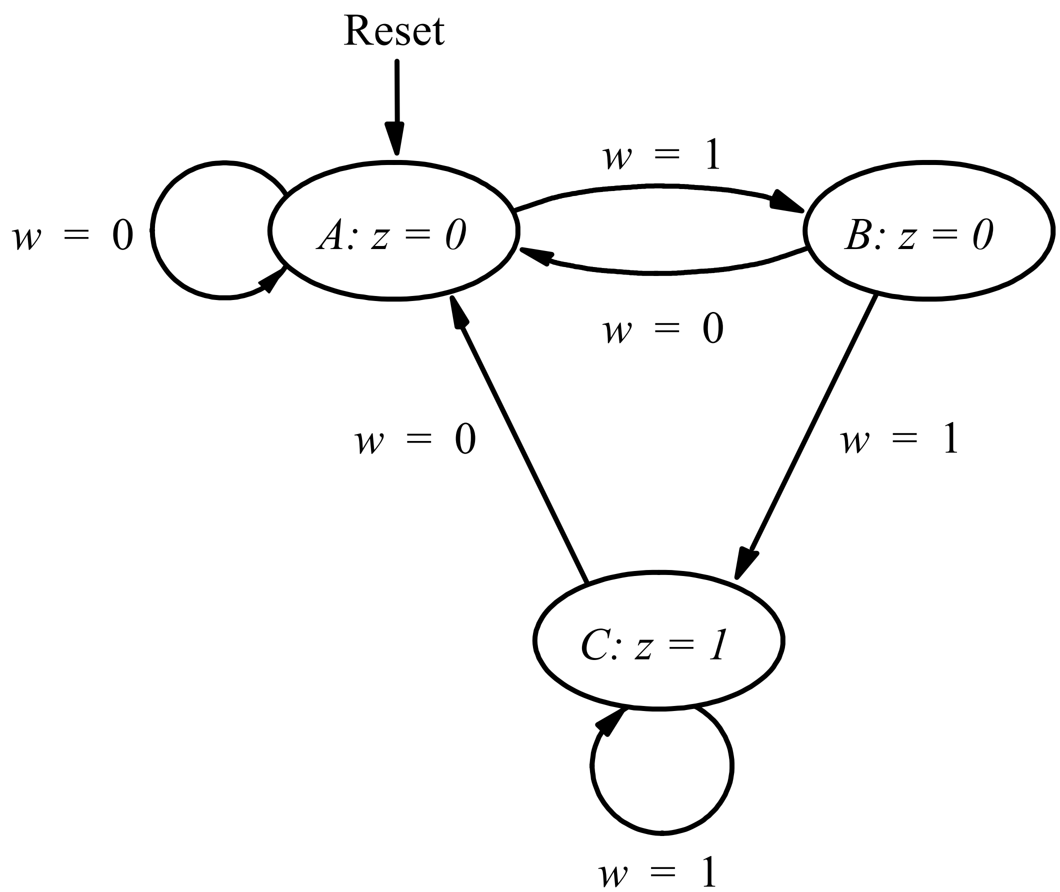


Key: The output depends only on the current state.

[Figure 6.3 from the textbook]

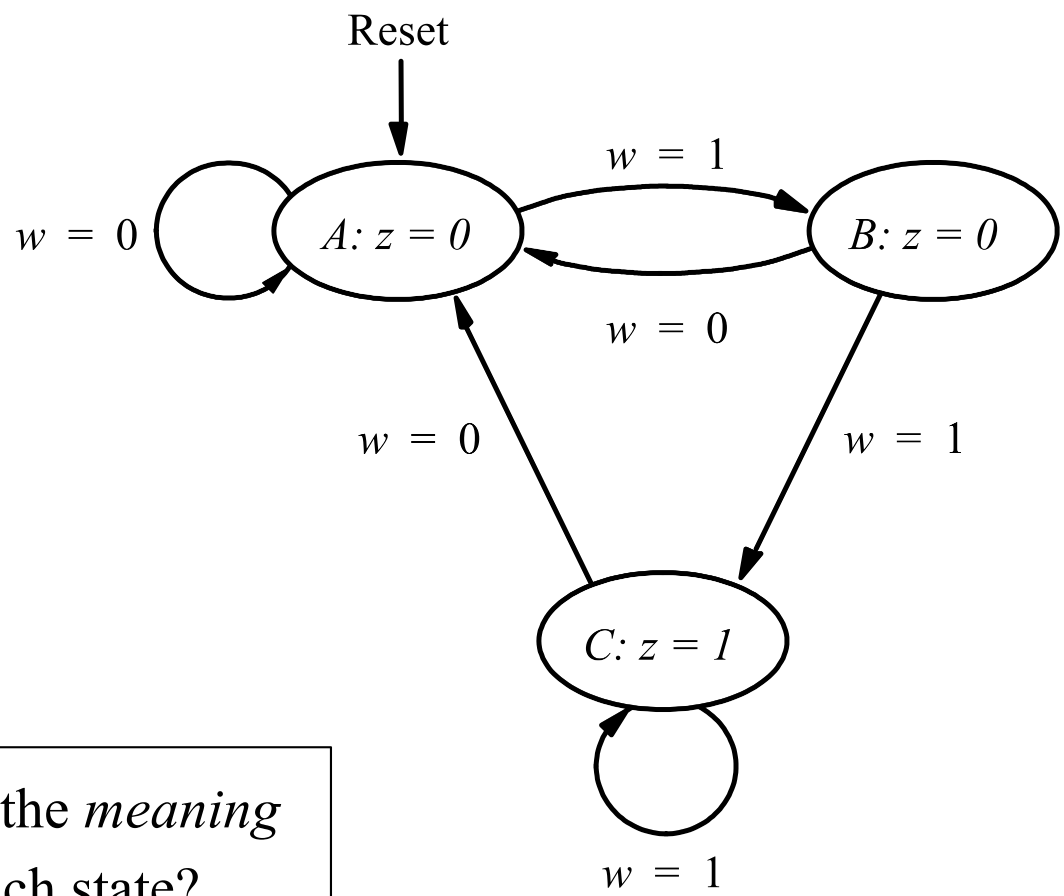


Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0



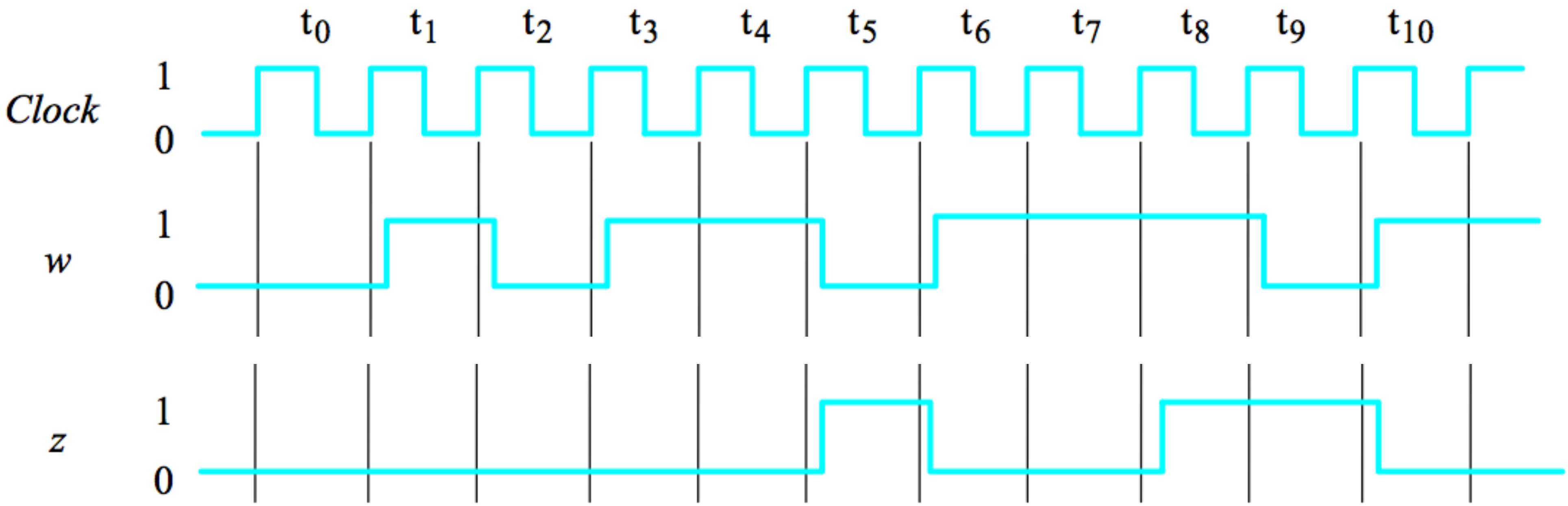
In general, we need to start tracing from the beginning to know which state the FSM is in. It may not be clear from a short sequence of outputs.

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0



What is the *meaning* of each state?

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0



Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

What is a State?

It is not really a memory of every past input
(We might run out of space to remember it all!)

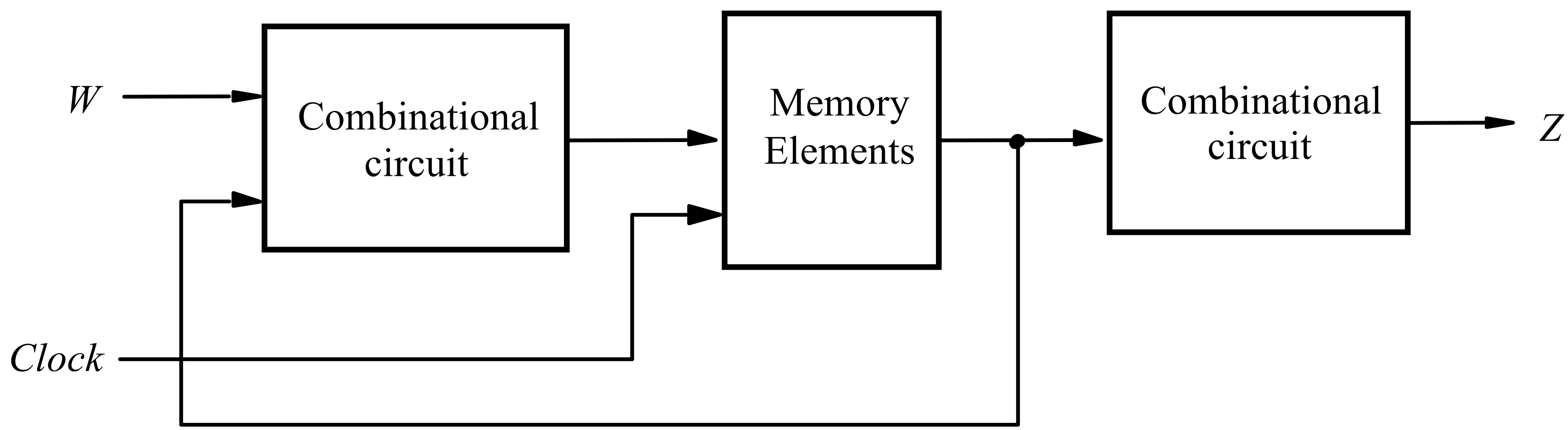
Rather, it is a characterization or snapshot of the pattern of inputs that have come before.

Moore Machine Implementation

The state diagram is just an illustration to help us describe and reason about how the FSM will behave in each of its states.

So, how do we turn it into a circuit?

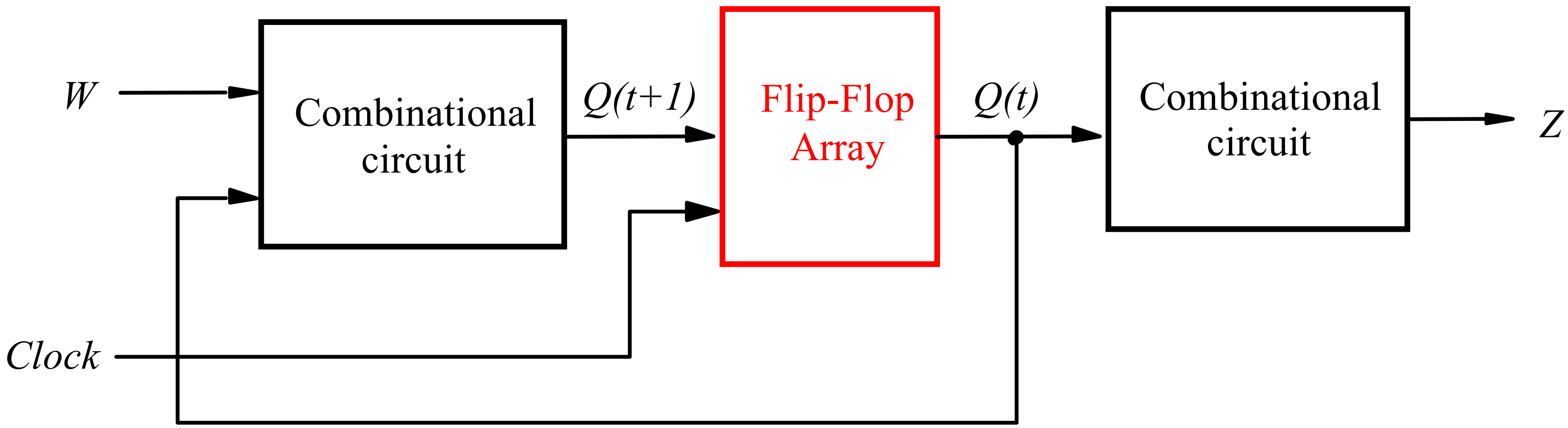
Moore Machine Implementation



Note: The *W* and *Z* lines need not be wires. They can be buses.

[Figure 6.1 from the textbook]

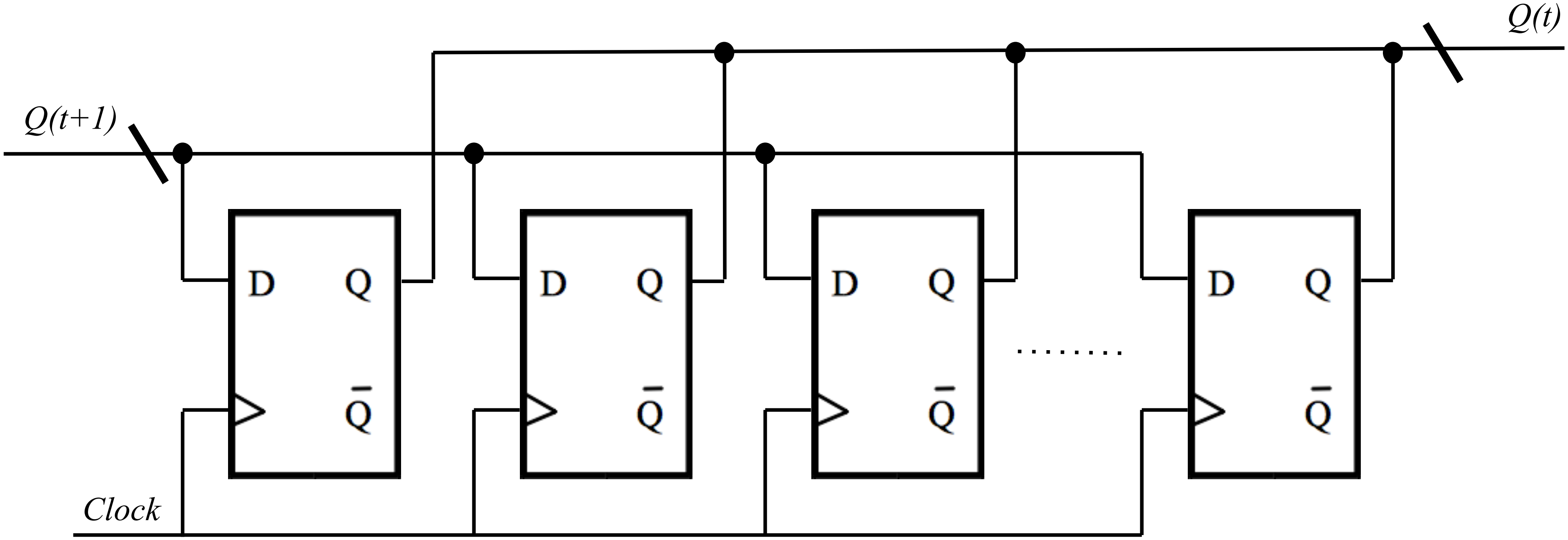
State Storage



Any usable "memory" of the preceding input sequence is encoded in the flip-flop array.

FSM States

The Flip-Flop array stores an encoding of the current state.



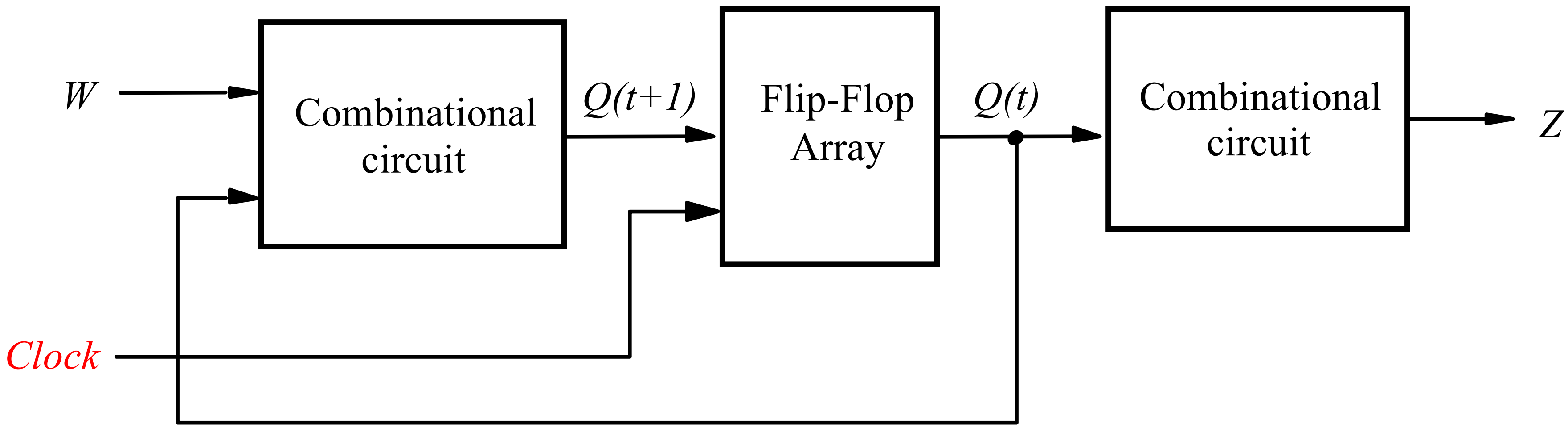
State Encoding

Each of the states in our design is identified by a distinct code.

If we use *3* flip-flops, then the FSM can have up to $2^3 = 8$ distinct states.

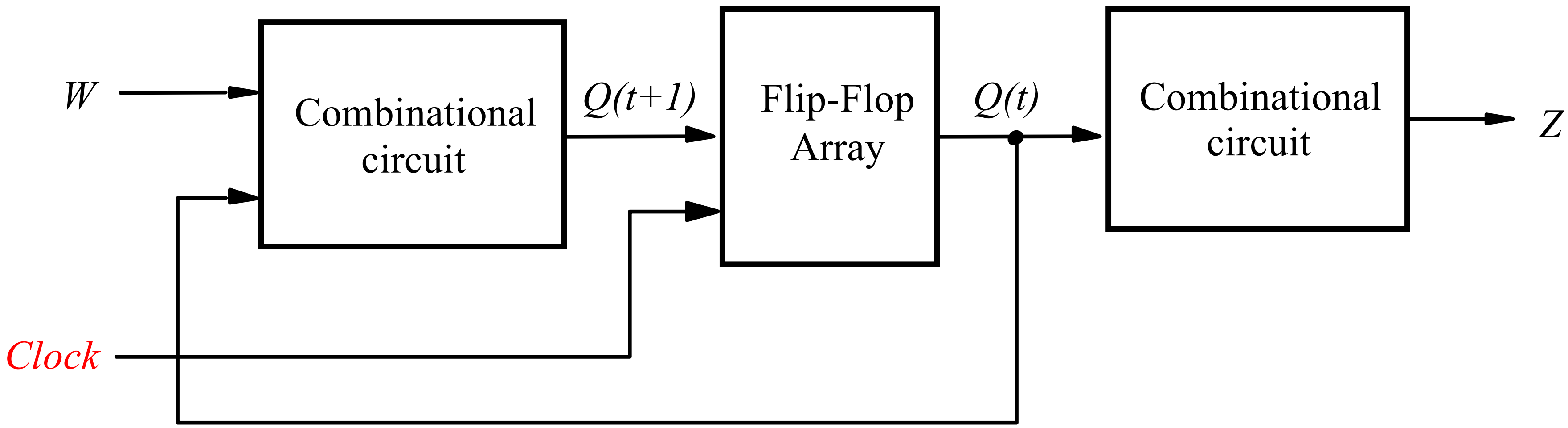
So, when the flip-flop array contains the code *011*, we say that the machine is in state *011*.

Synchronous Design



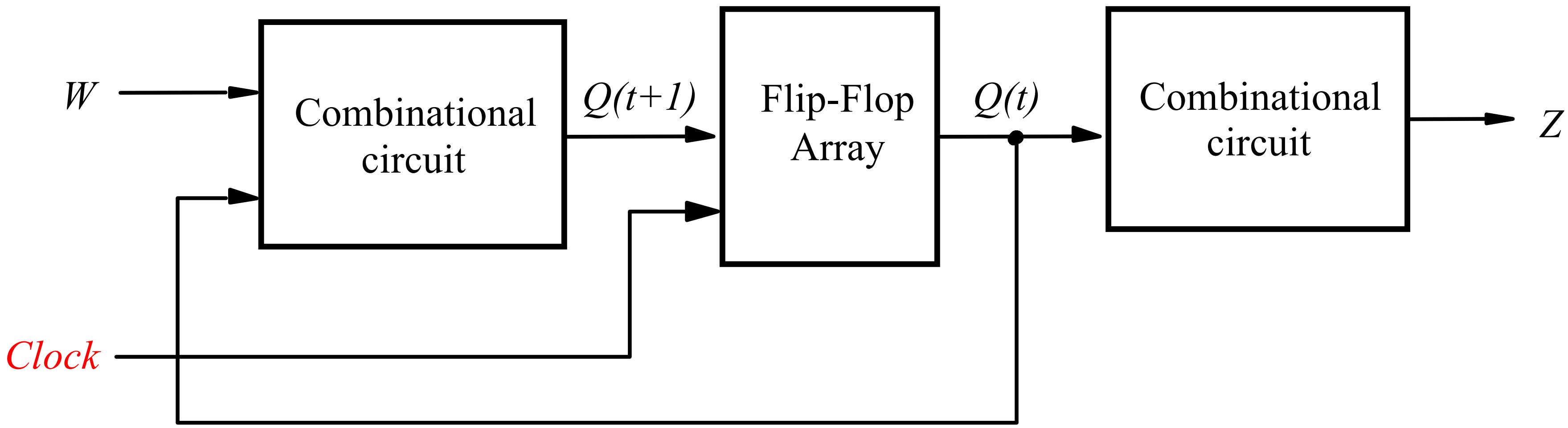
Every *active clock edge* causes a state transition.

Synchronous Design



We expect the input signals to be stable before the *active clock edge* occurs.

Synchronous Design



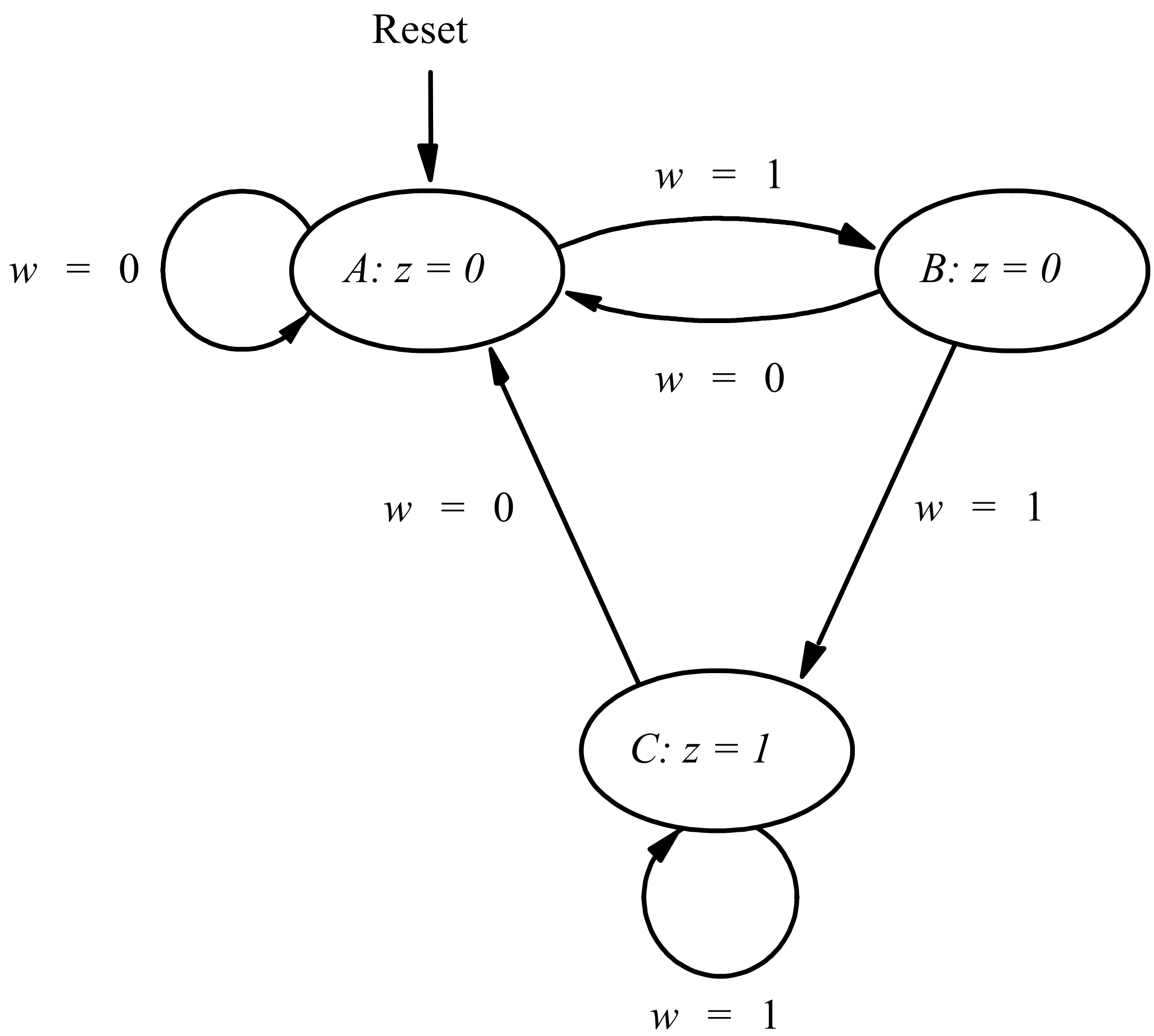
There is a whole other class of sequential circuits that are asynchronous, but we will not study them in this course.

Sequential Circuits: Key Ideas

The current output depends on something about the preceding sequence of inputs (and maybe the current output).

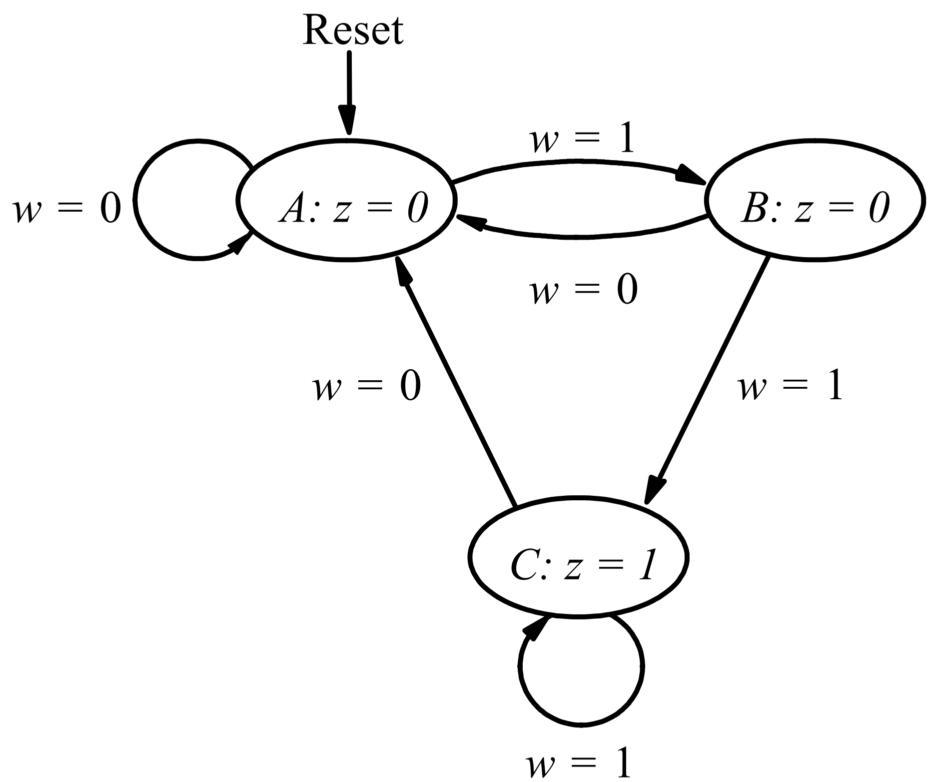
Using *memory elements* (i.e., flip-flops), we design the circuit to remember some *relevant* information about the prior inputs.

Example

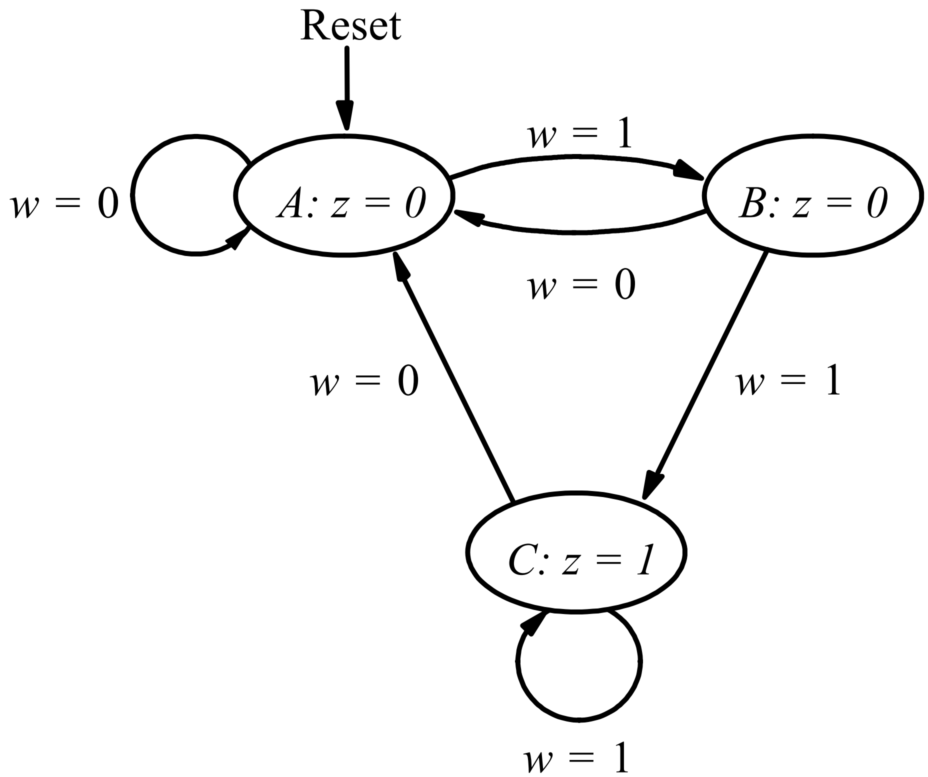


We need to find both the *next state logic* and the *output logic* implied by this machine.

[Figure 6.3 from the textbook]



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



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

[Figure 6.4 from the textbook]

How to represent the States?

One way is to encode each state with a 2-bit binary number

A ~ 00

B ~ 01

C ~ 10

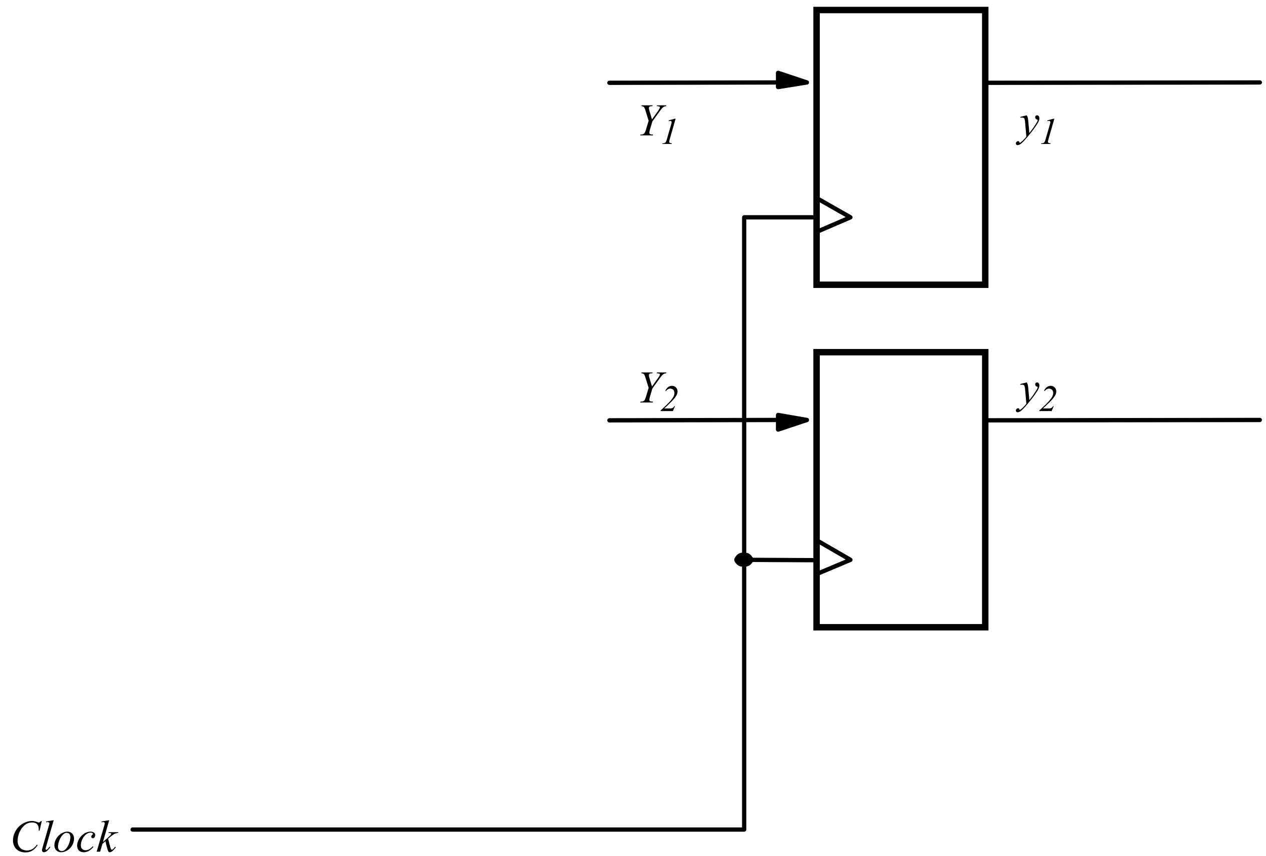
How to represent the states?

One way is to encode each state with a 2-bit binary number

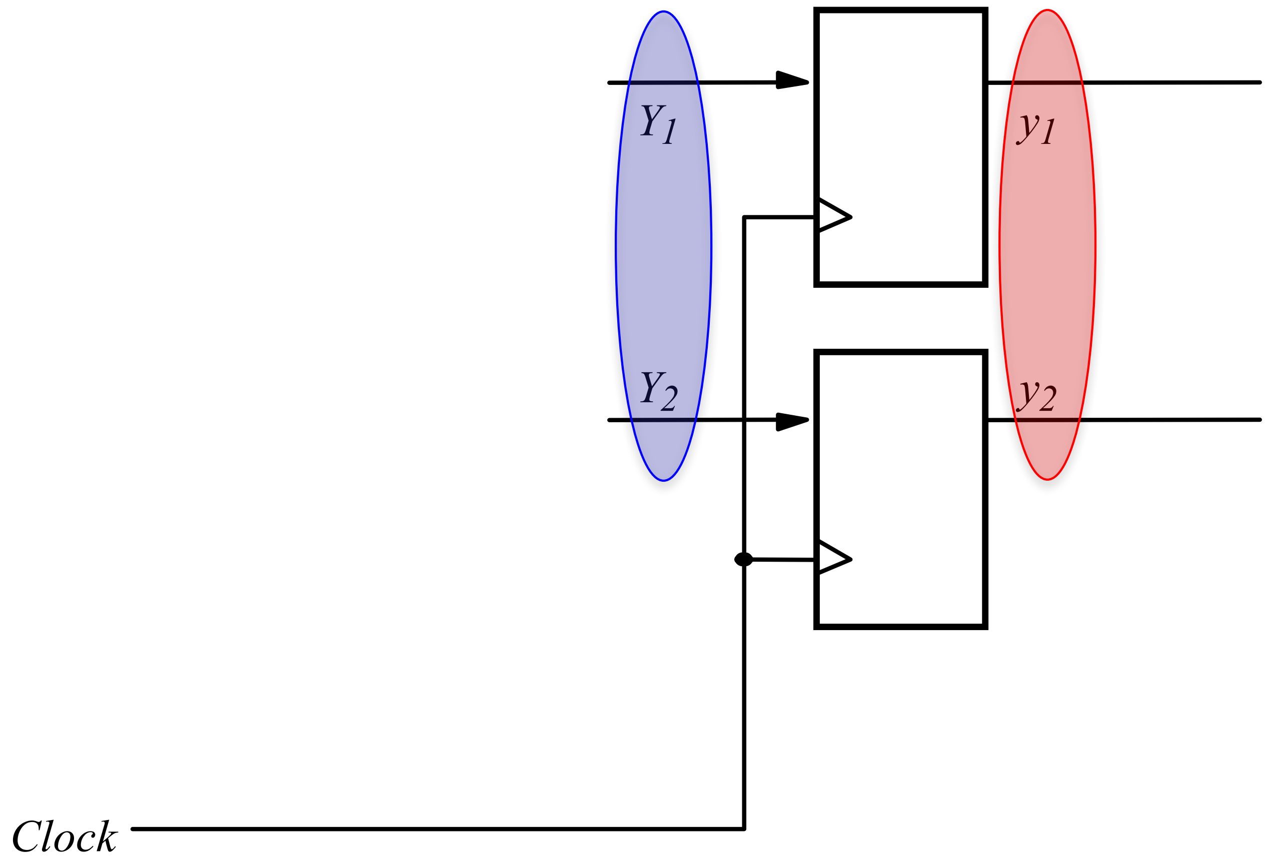
A ~ 00
B ~ 01
C ~ 10

How many flip-flops do we need?

Let's use two flip-flops to hold the machine's state



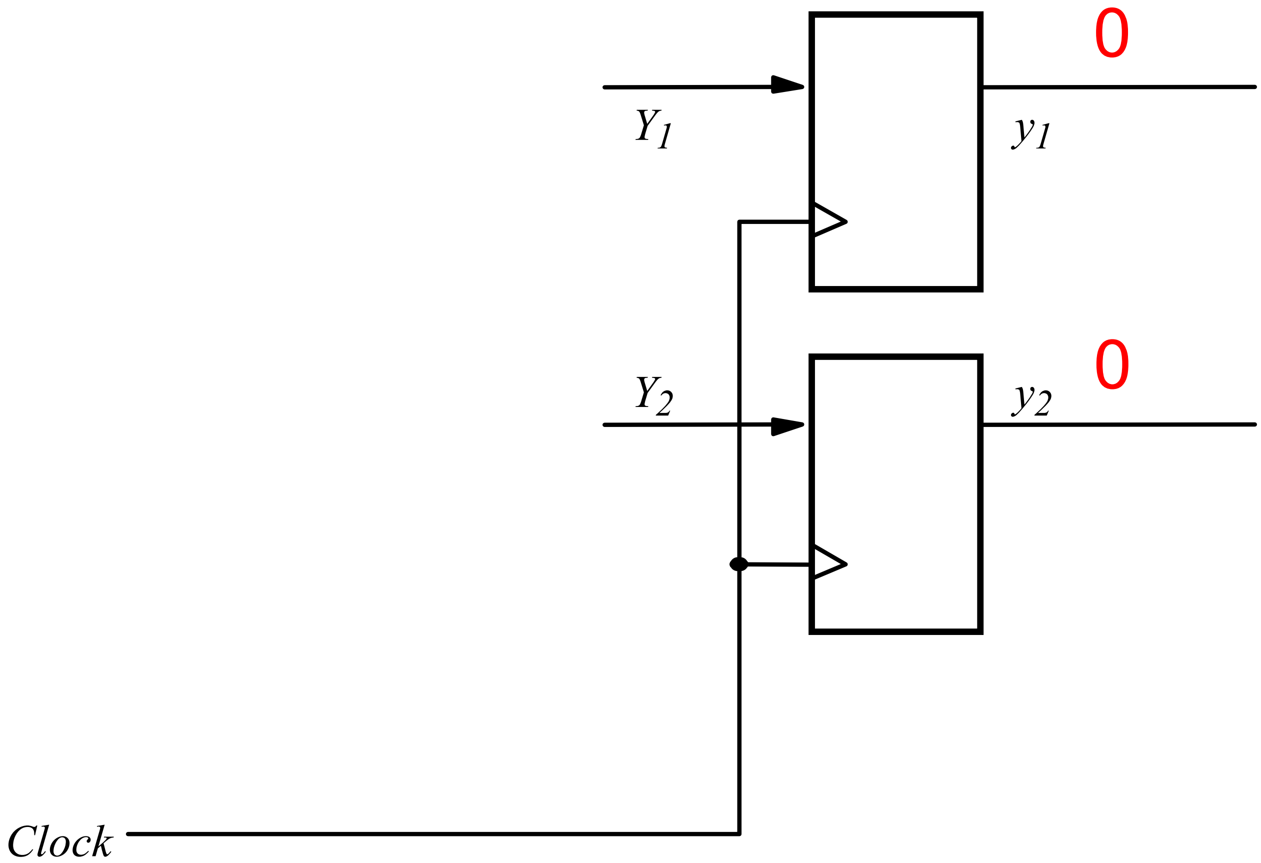
Y_1
y_1
Y_2
y_2
Clock



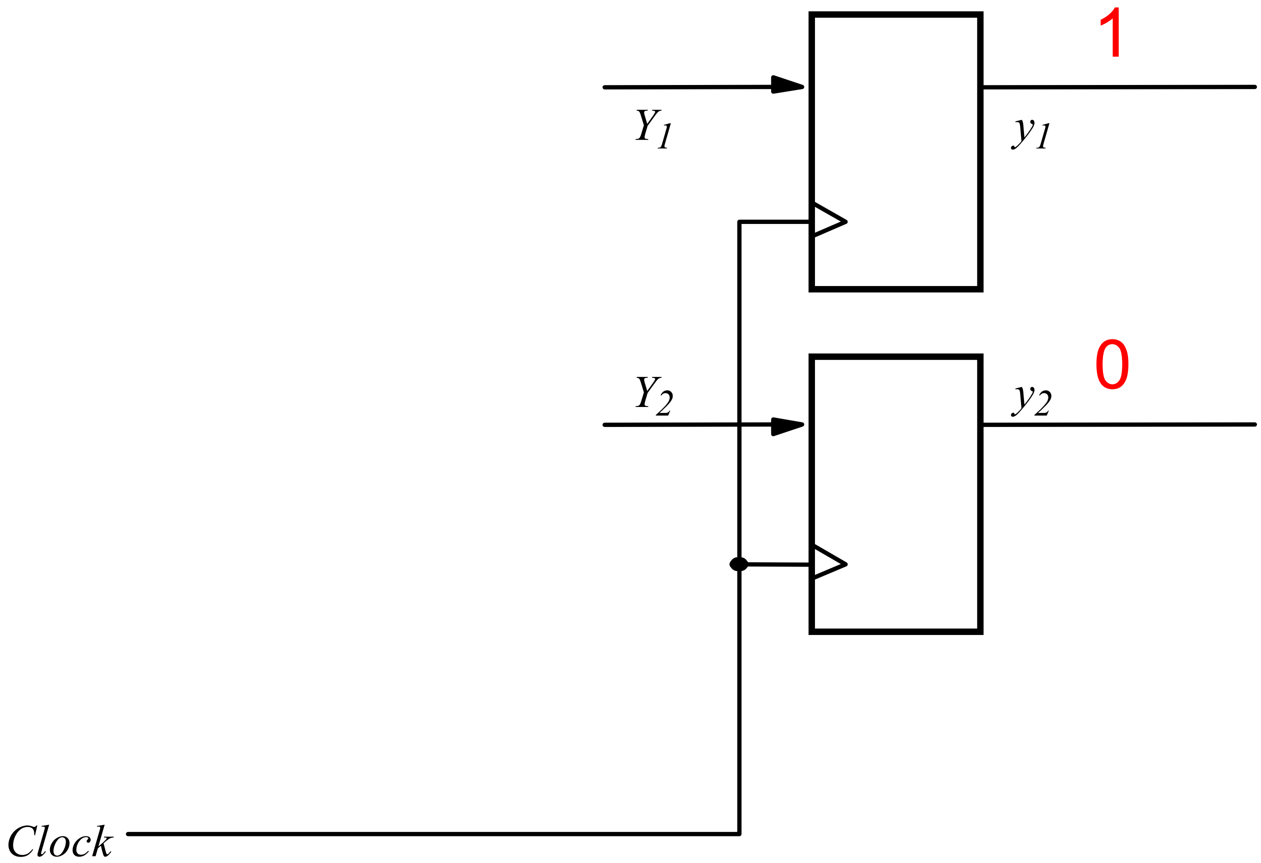
We will call y_1 and y_2 the *present state variables*.

We will call Y_1 and Y_2 the *next state variables*.

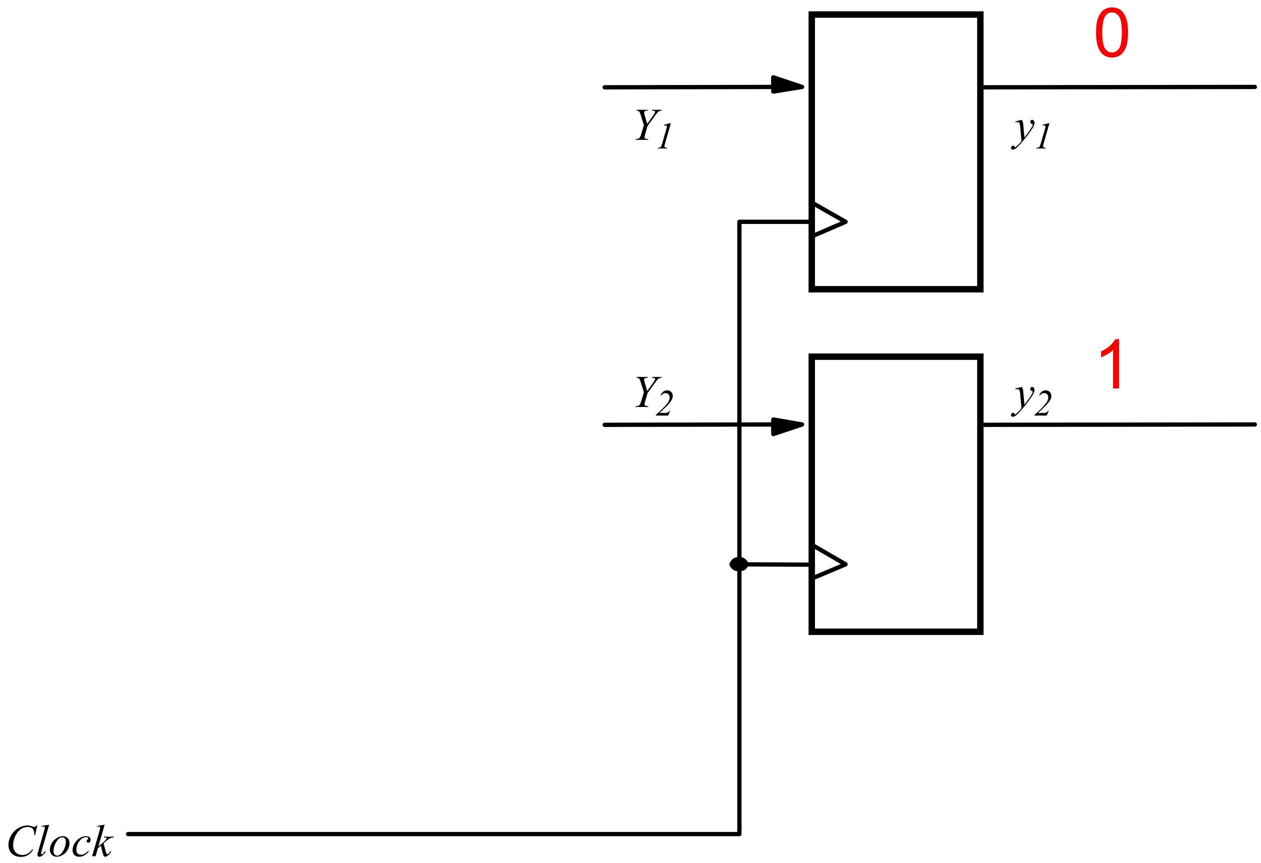
[Figure 6.5 from the textbook]



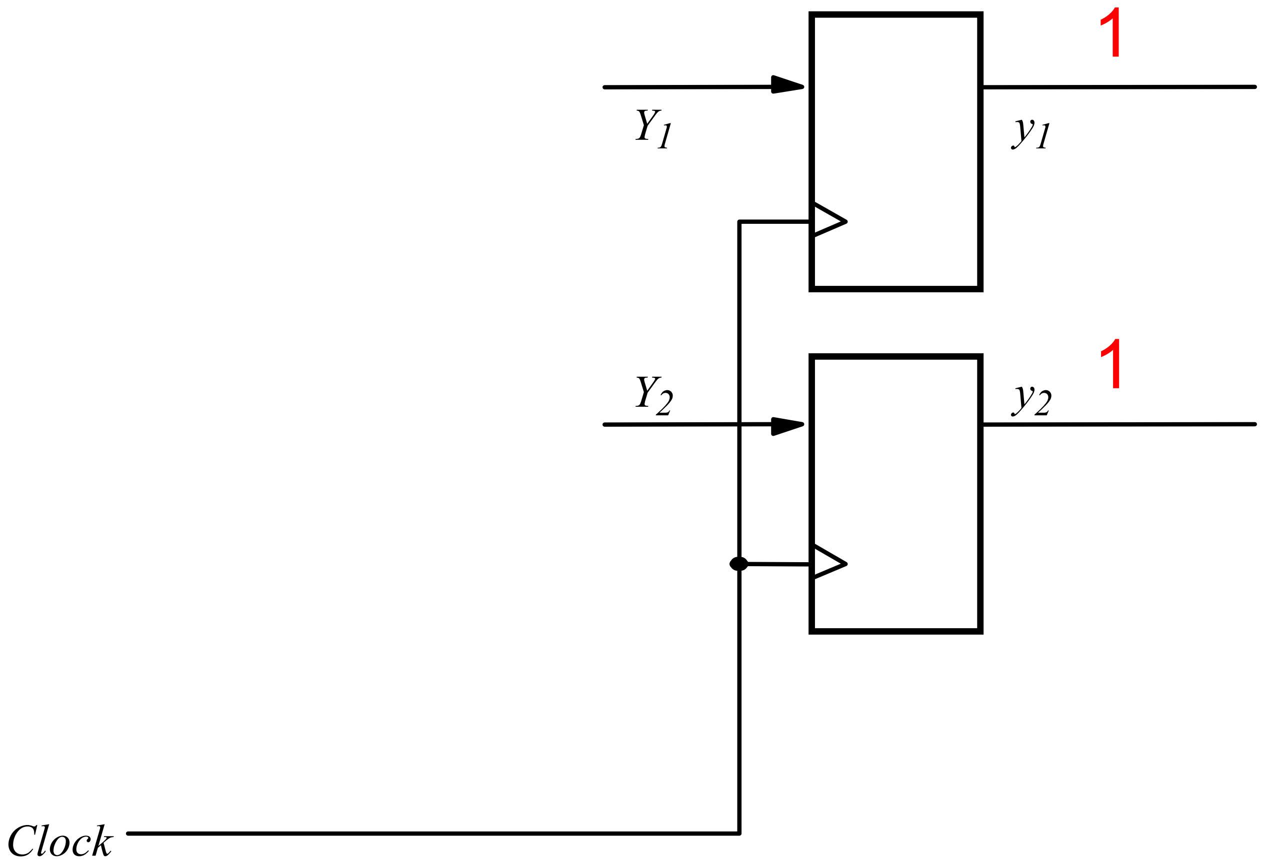
Two zeros on the output JOINTLY represent state A.



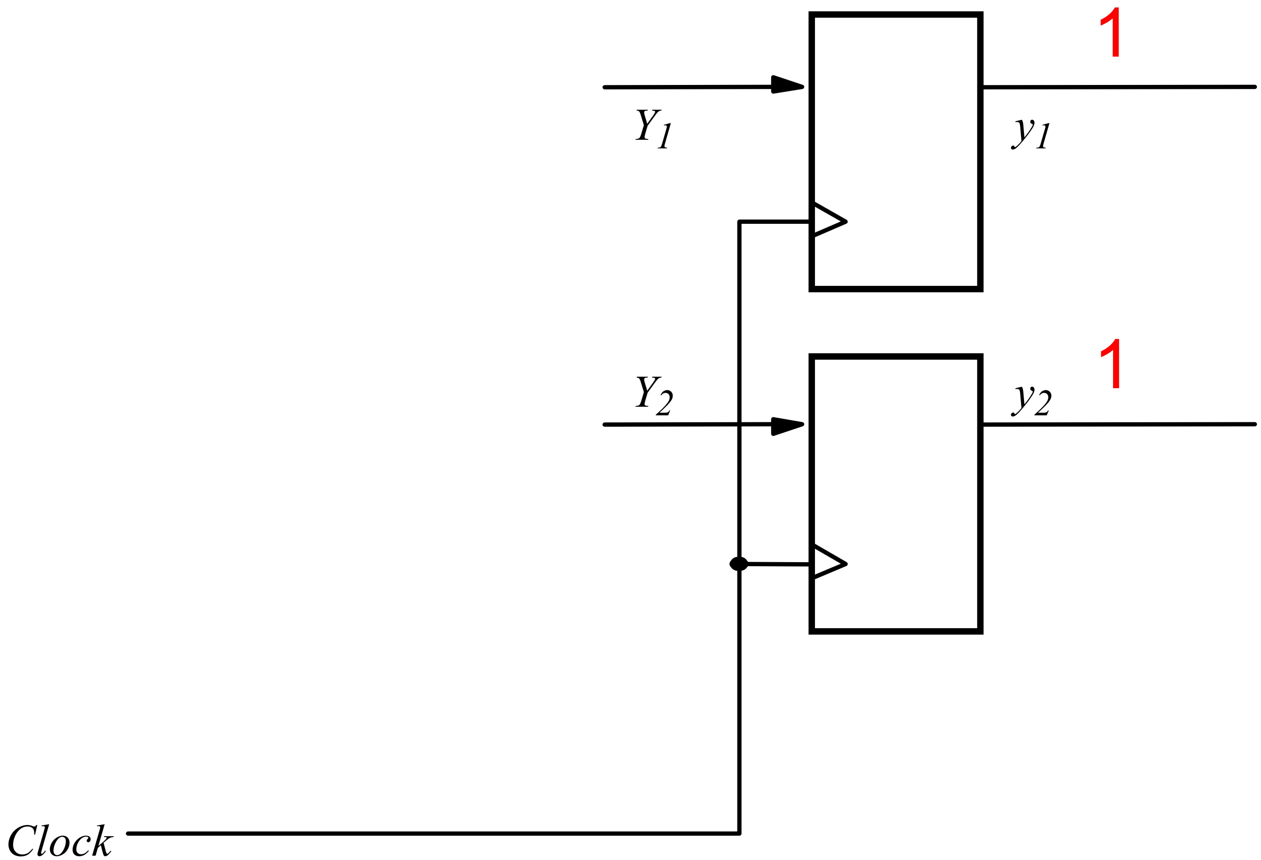
This flip-flop output pattern represents state B.



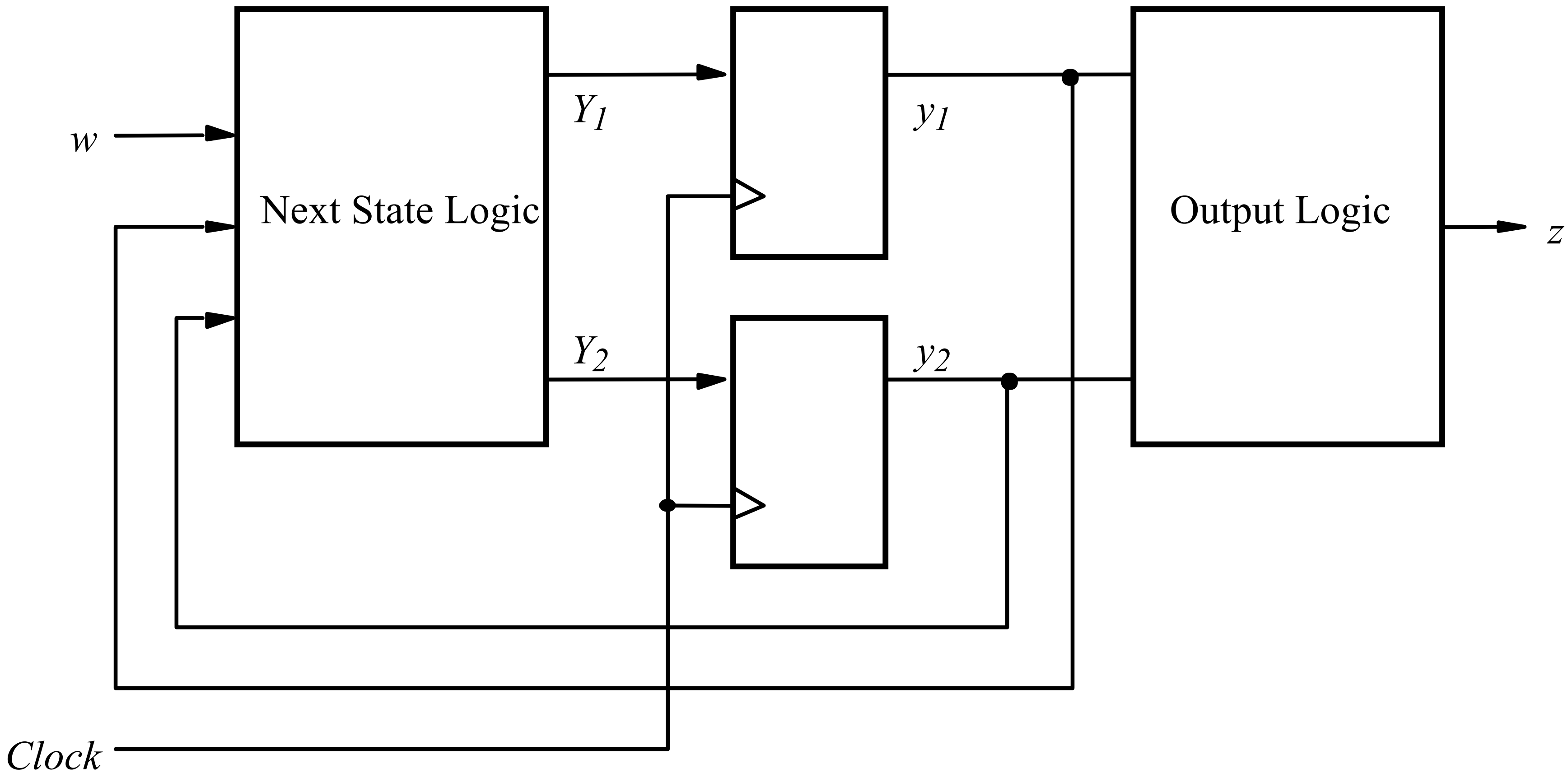
This flip-flop output pattern represents state C.



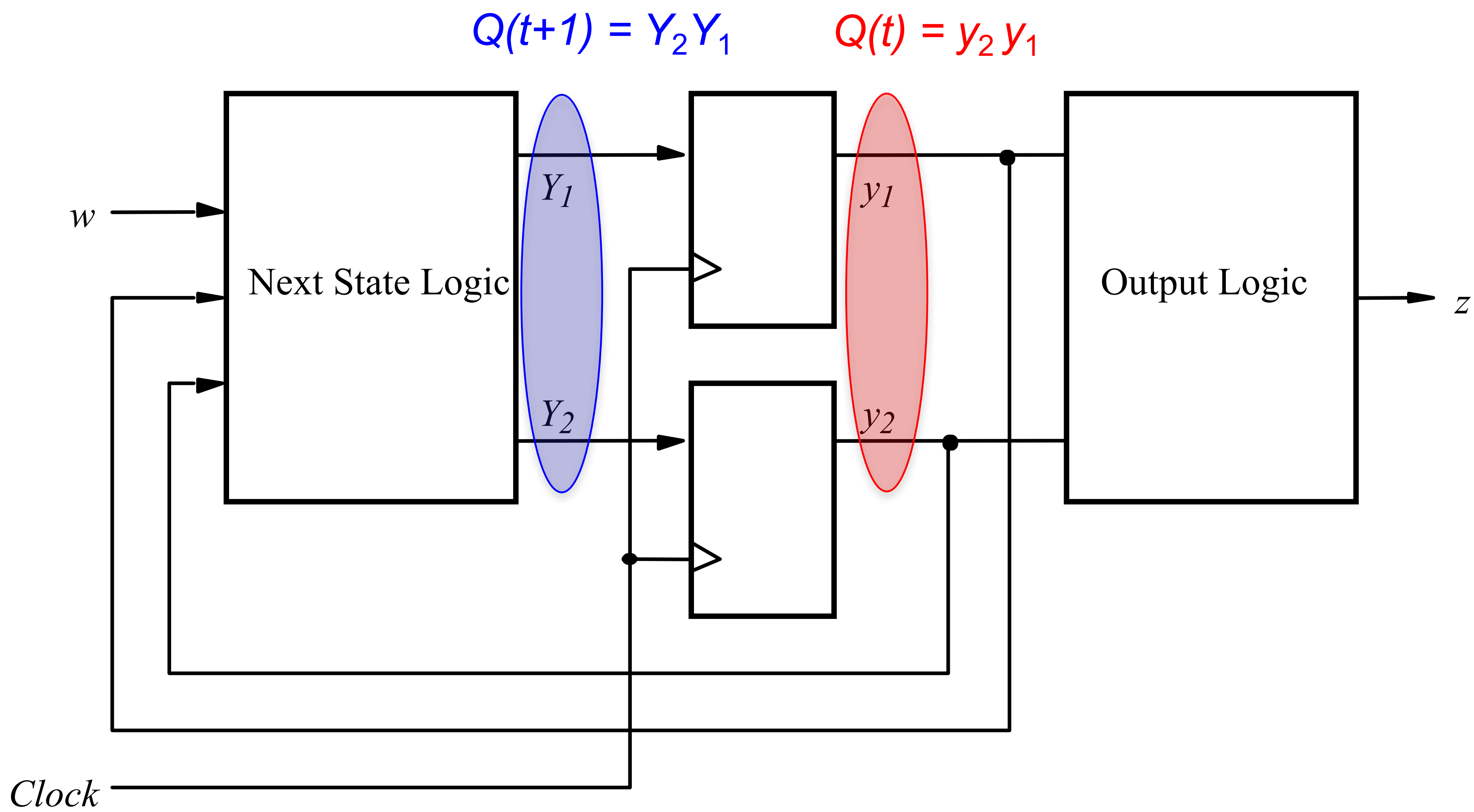
What does this flip-flop output pattern represent?



This would be state D, but we don't have one in this example. So this is an impossible state.



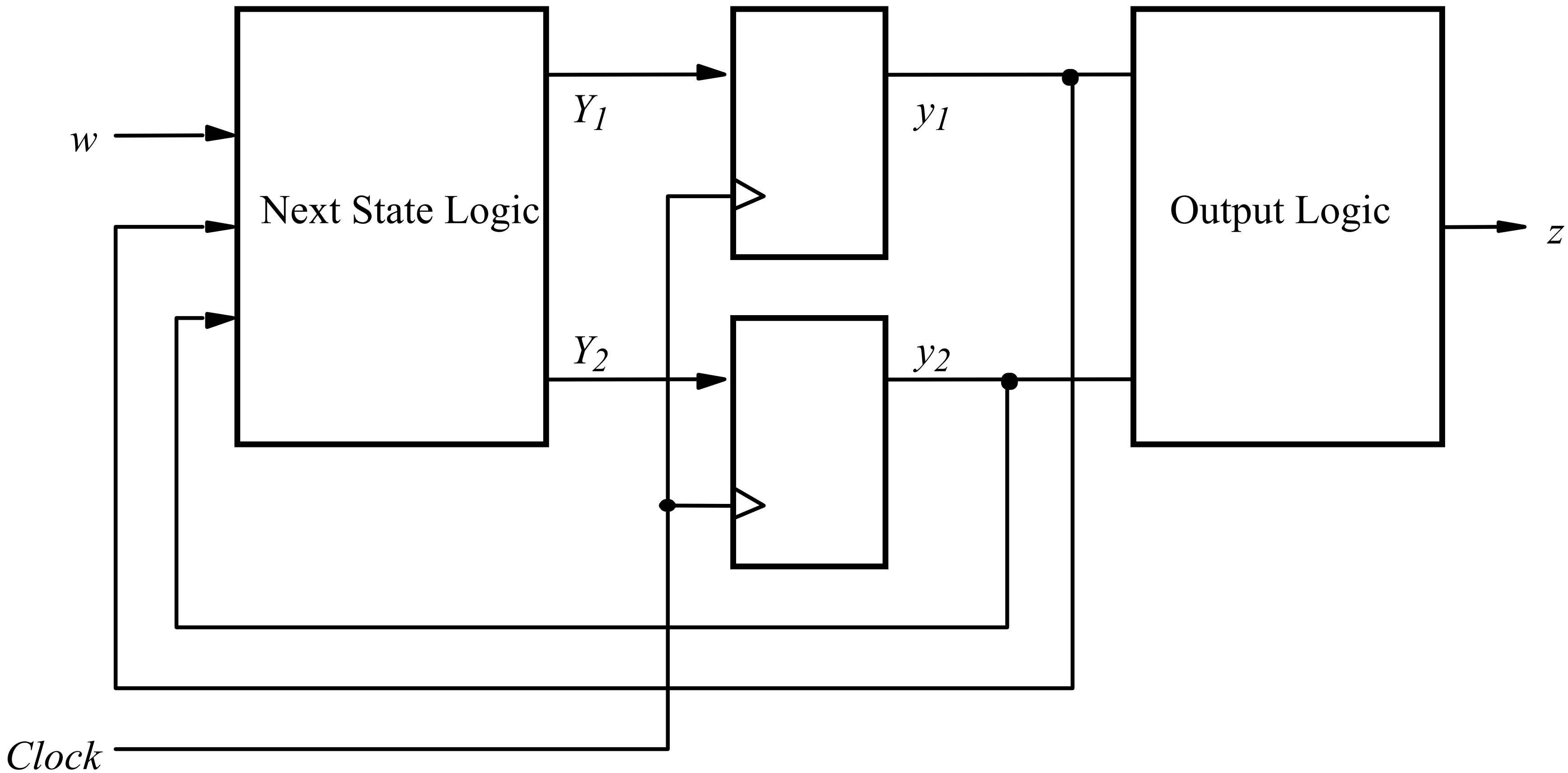
[Figure 6.5 from the textbook]



We will call y_1 and y_2 the *present state variables*.

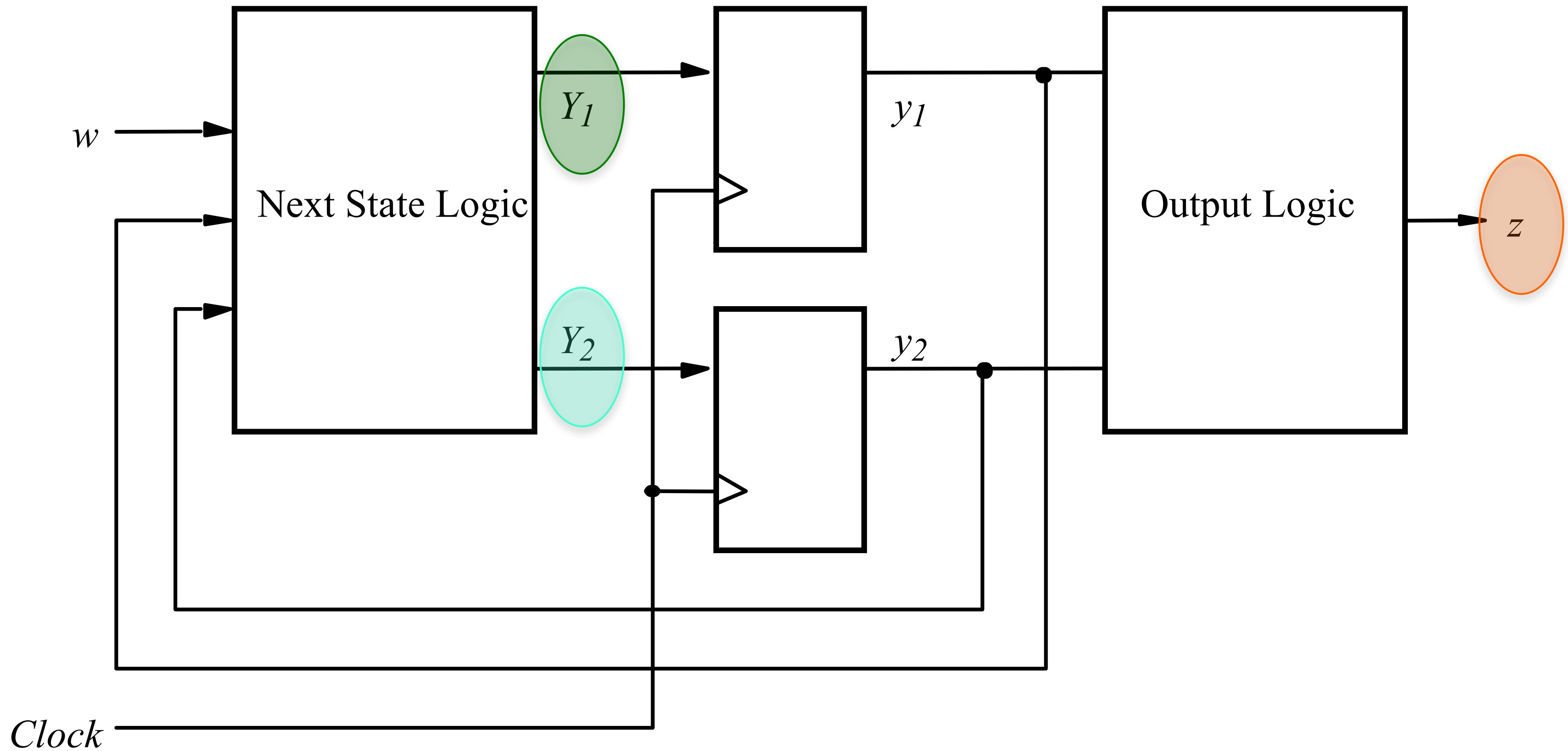
We will call Y_1 and Y_2 the *next state variables*.

[Figure 6.5 from the textbook]



We need to find logic expressions for
$Y_1(w, y_1, y_2)$, $Y_2(w, y_1, y_2)$, and $z(y_1, y_2)$.

[Figure 6.5 from the textbook]



We need to find logic expressions for $Y_1(w, y_1, y_2)$, $Y_2(w, y_1, y_2)$, and $z(y_1, y_2)$.

[Figure 6.5 from the textbook]

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

Suppose we encoded our states in the same order in which they were labeled:

A ~ 00
B ~ 01
C ~ 10

[Figure 6.4 from the textbook]

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

	Present state	Next state $w = 0$	Next state $w = 1$	Output z
A	00			
B	01			
C	10			
	11			

The finite state machine will never reach a state encoded as 11.

[Figure 6.6 from the textbook]

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

We arbitrarily chose these as our state encodings. We could have used others.

	Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

[Figure 6.6 from the textbook]

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

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

y_2	y_1	z
0	0	
0	1	
1	0	
1	1	

[Figure 6.6 from the textbook]

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

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

[Figure 6.6 from the textbook]

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

[Figure 6.6 from the textbook]

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

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

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	d	
1	0	0		
1	0	1		
1	1	0		
1	1	1		

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

[Figure 6.6 from the textbook]

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

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

w	y_2	y_1	Y_2	Y_1
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	d	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	d	

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

[Figure 6.6 from the textbook]

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

[Figure 6.6 from the textbook]

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	d	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	d	

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	d	

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

[Figure 6.6 from the textbook]

Present state $y_2 y_1$	Next state $w = 0$ $Y_2 Y_1$	Next state $w = 1$ $Y_2 Y_1$	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

$Q(t) = y_2 y_1$ and $Q(t+1) = Y_2 Y_1$

w	y_2	y_1	Y_2	Y_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	d	d

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

[Figure 6.6 from the textbook]

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

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

w	y_2	y_1	Y_2	Y_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	d	d

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

[Figure 6.6 from the textbook]

Note that the textbook draws these K-Maps differently from all previous K-maps
(the least significant bits index the columns, instead of the most significant bits).

$Q(t) = y_2y_1$ and $Q(t+1) = Y_2Y_1$

Y_1

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

w	y_2	y_1	Y_2	Y_1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	d	d

Y_2

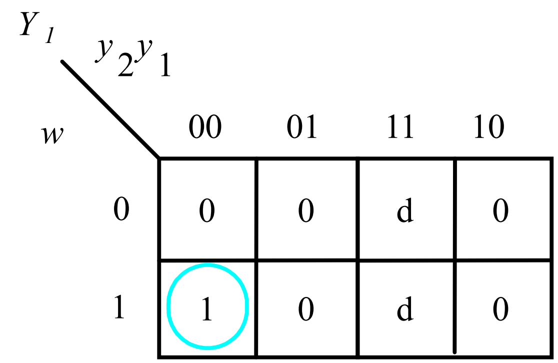
w \ y_2y_1	00	01	11	10
0	0	0	d	0
1	0	1	d	1

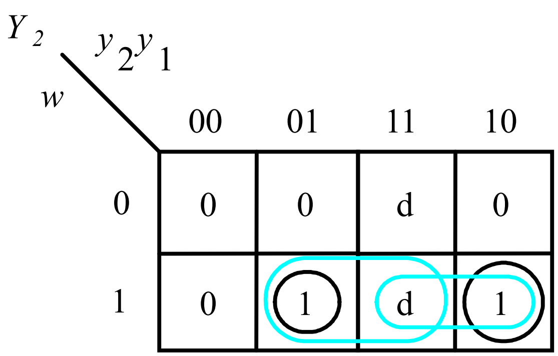
z

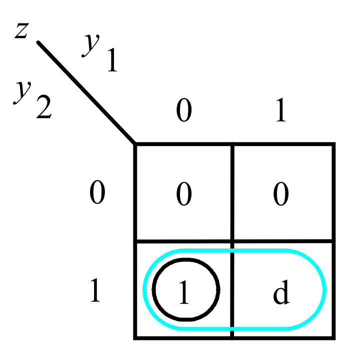
y_2 \ y_1	0	1
0	0	0
1	1	d

y_2	y_1	z
0	0	0
0	1	0
1	0	1
1	1	d

Don’t care conditions simplify the combinatorial logic







Ignoring don't cares

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

$Y_2 = wy_1\bar{y}_2 + \bar{w}y_1y_2$

$z = \bar{y}_1y_2$

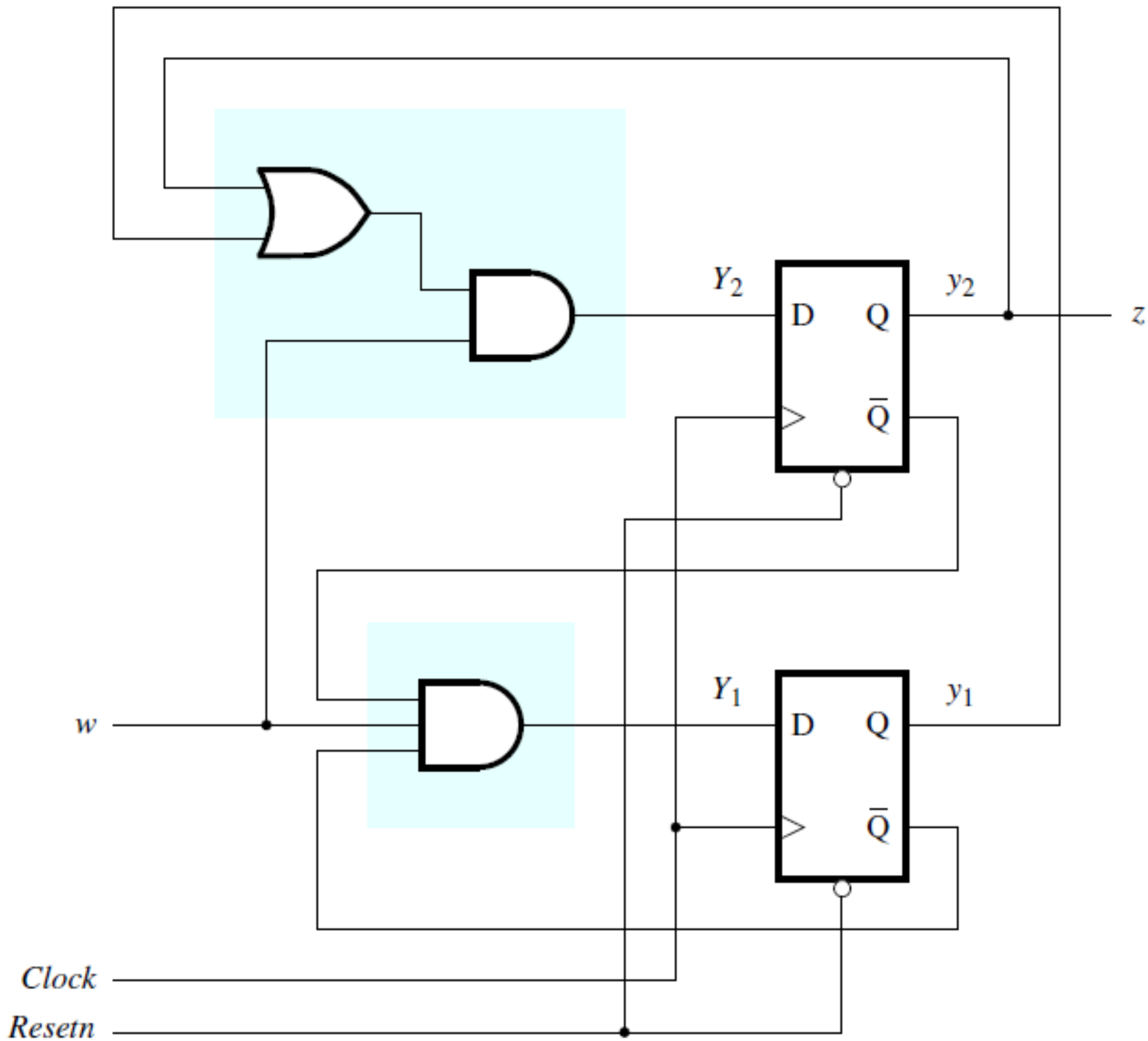
Using don't cares

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

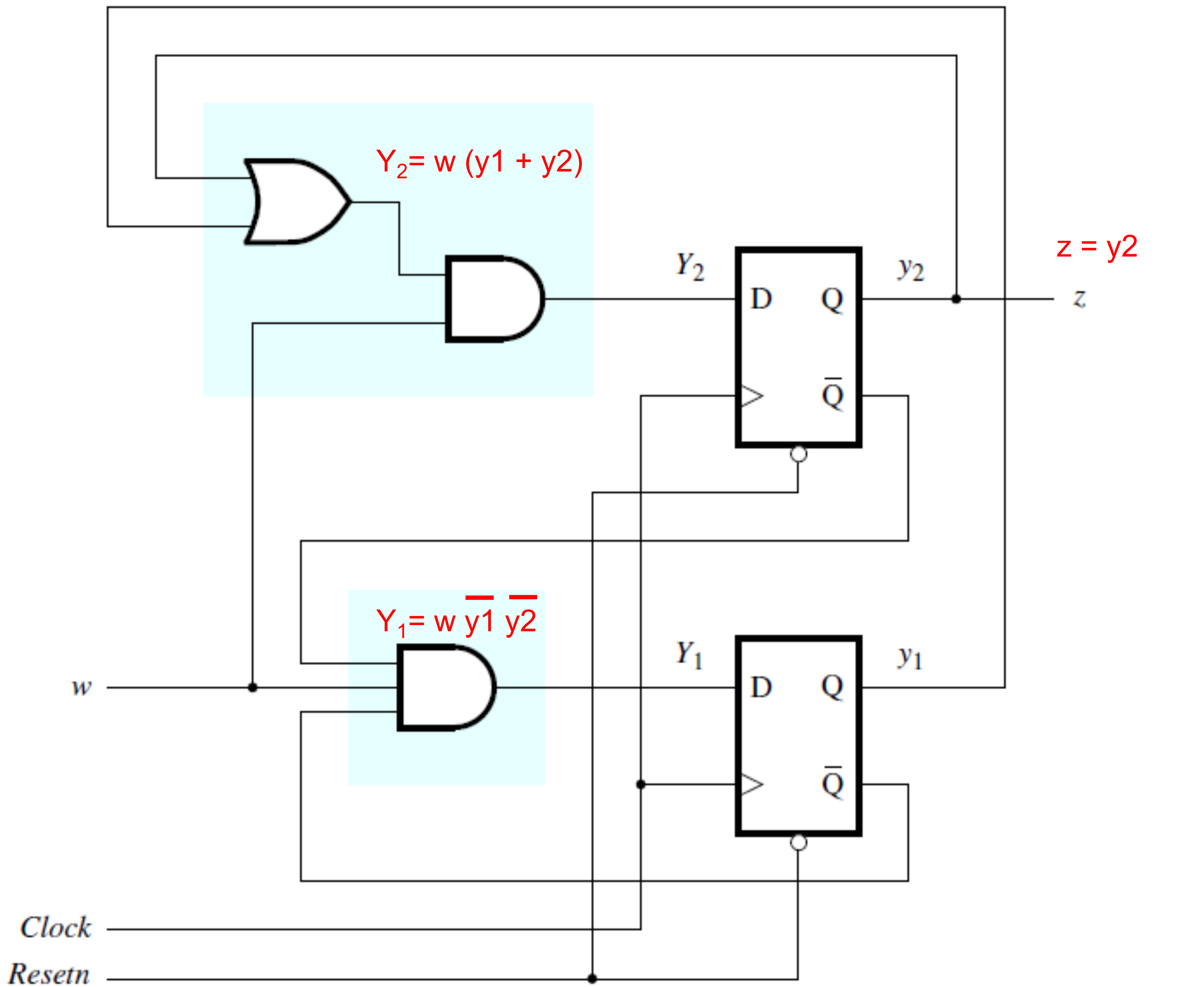
$Y_2 = wy_1 + wy_2$
$= w(y_1 + y_2)$

$z = y_2$

[Figure 6.7 from the textbook]

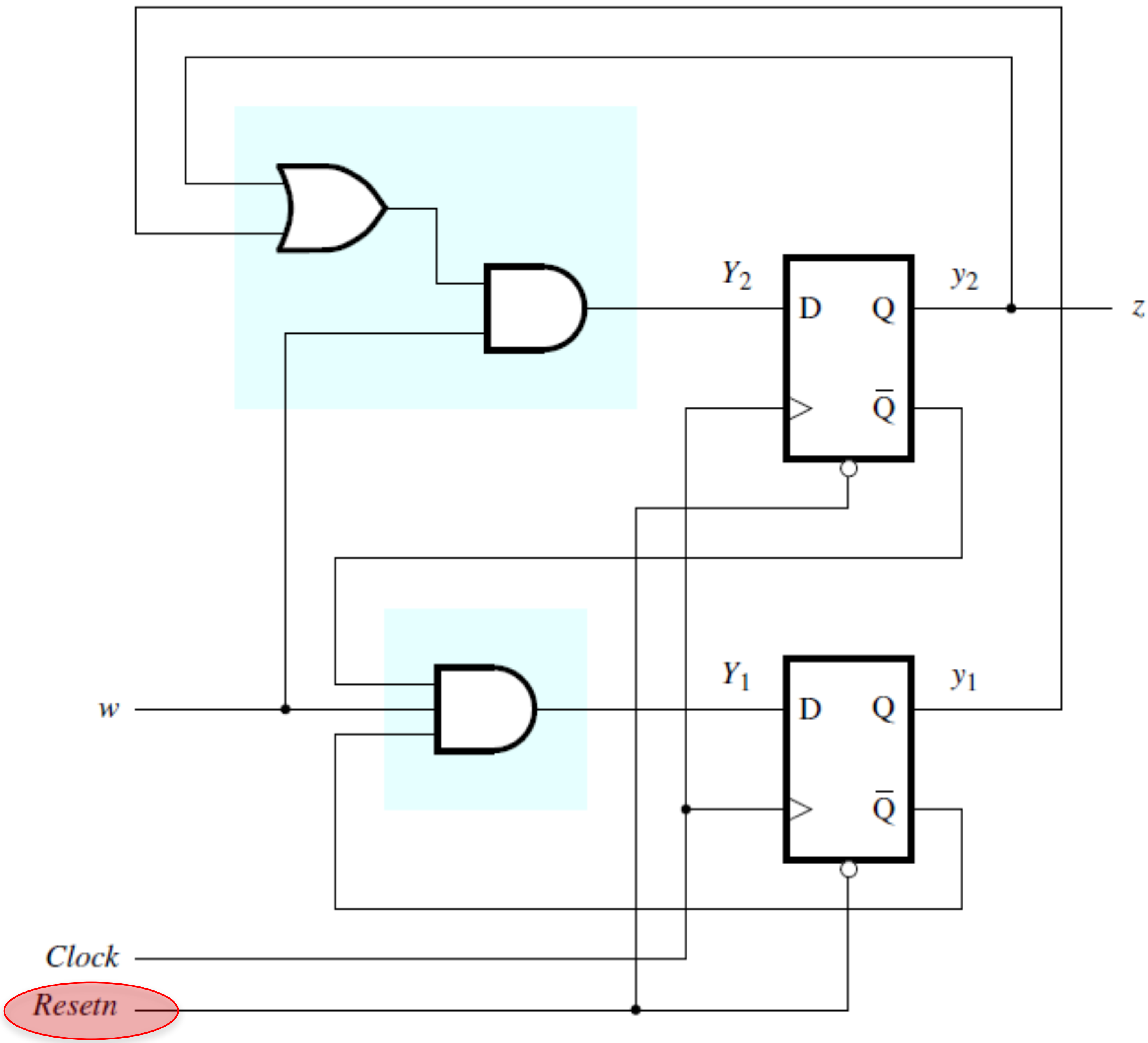


[Figure 6.8 from the textbook]



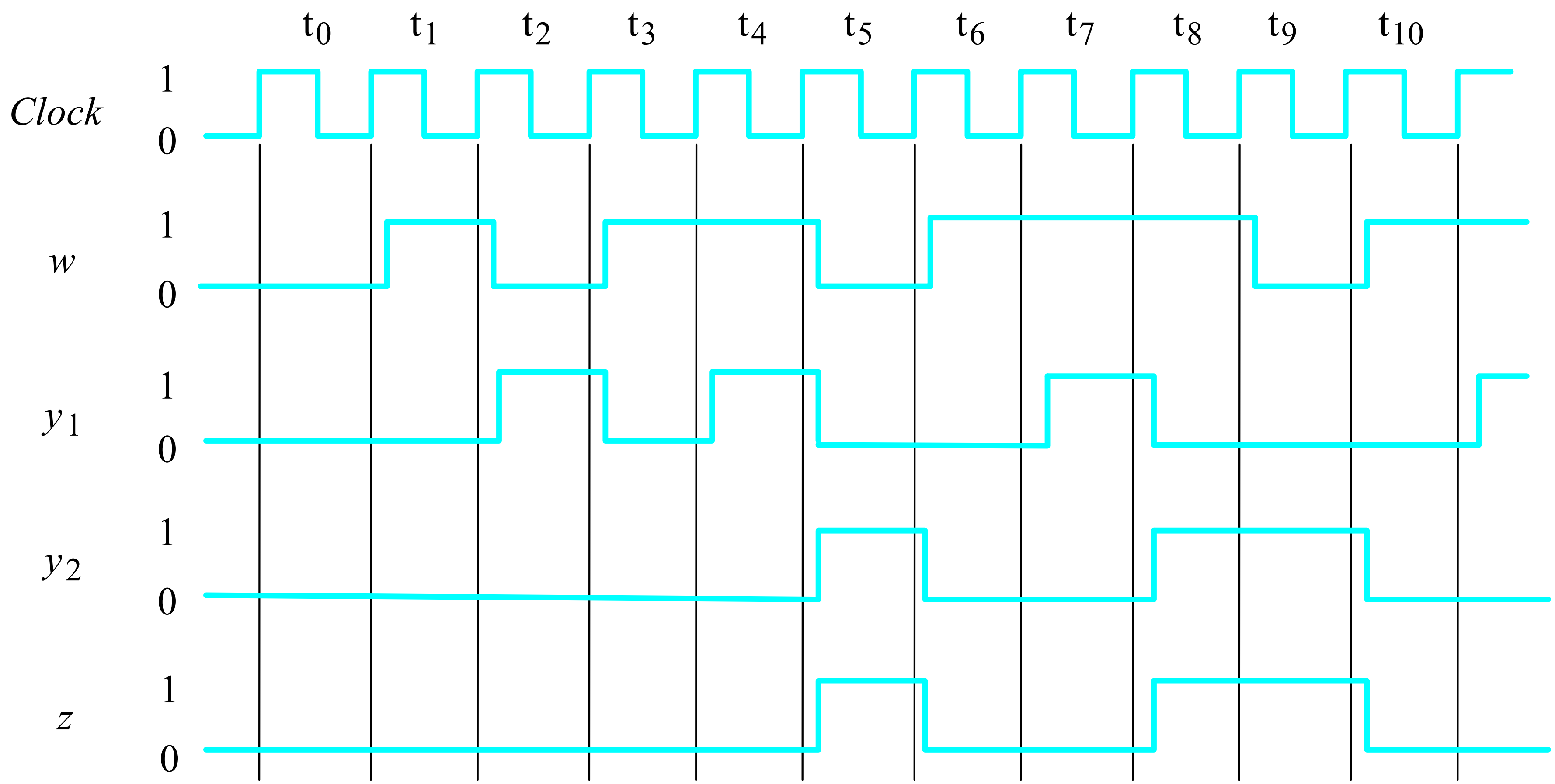
[Figure 6.8 from the textbook]

Lastly, we add a reset signal, which forces the machine back to its start state, which is state *00* in this case.



[Figure 6.8 from the textbook]

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0



[Figure 6.9 from the textbook]

Summary: Designing a Moore Machine

- Obtain the circuit specification.
- Derive a state diagram.
- Derive the state table.
- Decide on a state encoding.
- Encode the state table.
- Derive the output logic and next-state logic.
- Add a reset signal.

Questions?

THE END