EDA322
Digital Design

Lecture 8:
Finite State Machines (FSMs)

Ioannis Sourdis

Outline of Lecture 8

- Finite State Machines
  - Moore
  - Mealy
- State assignment
- State minimization

General form of a sequential circuit

Realized using combinational logic and flip-flops

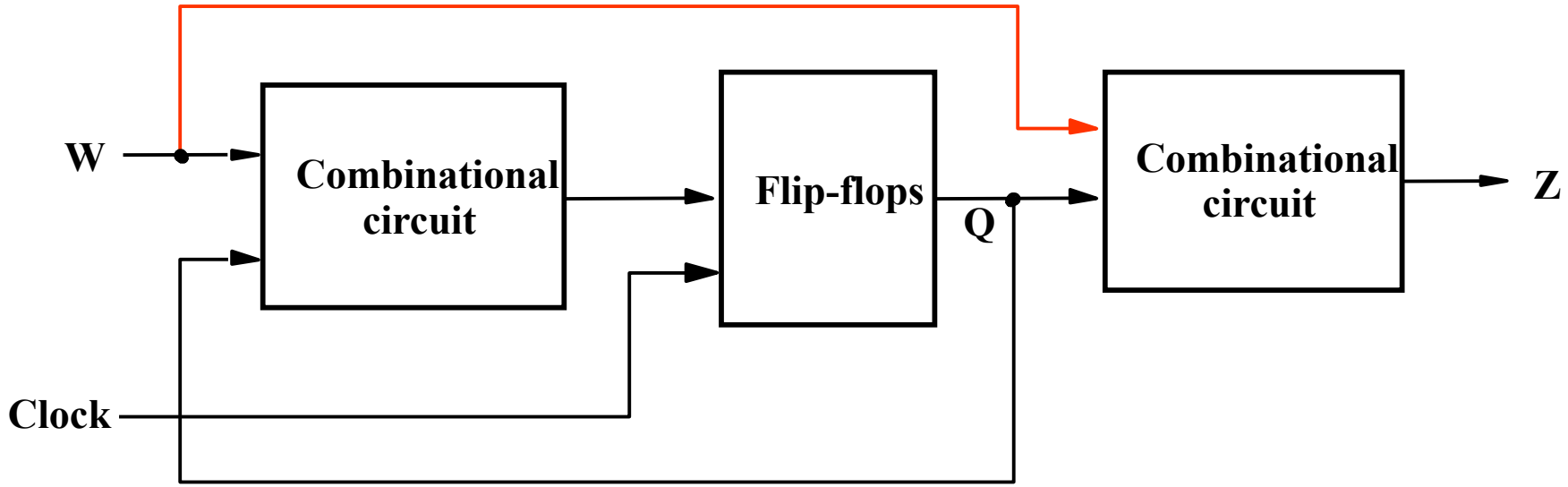
Primary inputs: w

Outputs: z

State: Q

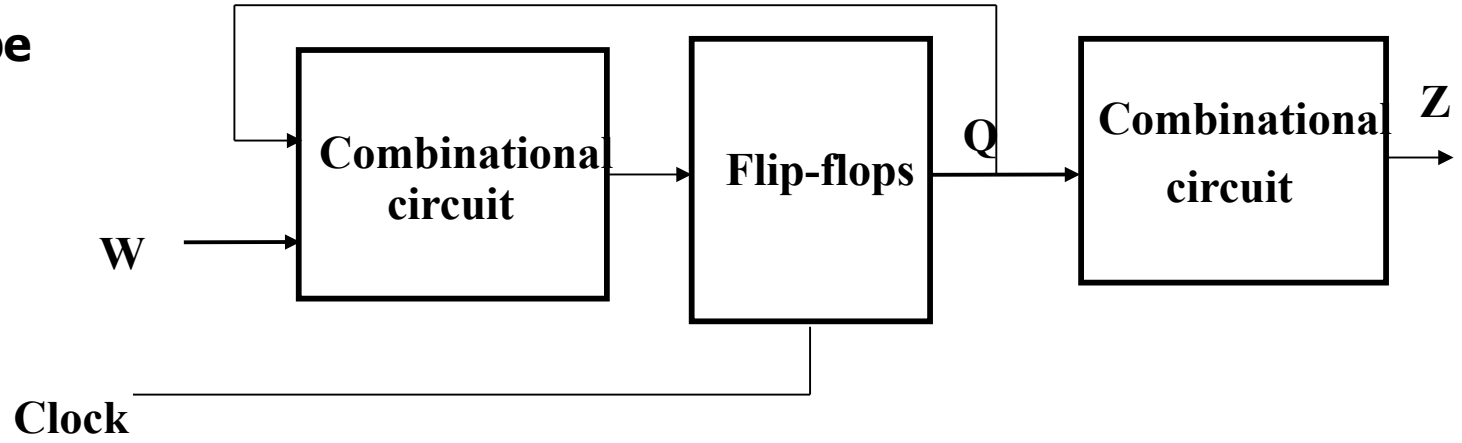
Moore FSMs: outputs depend only on the state

Mealy FSMs: outputs depend on both state and primary inputs

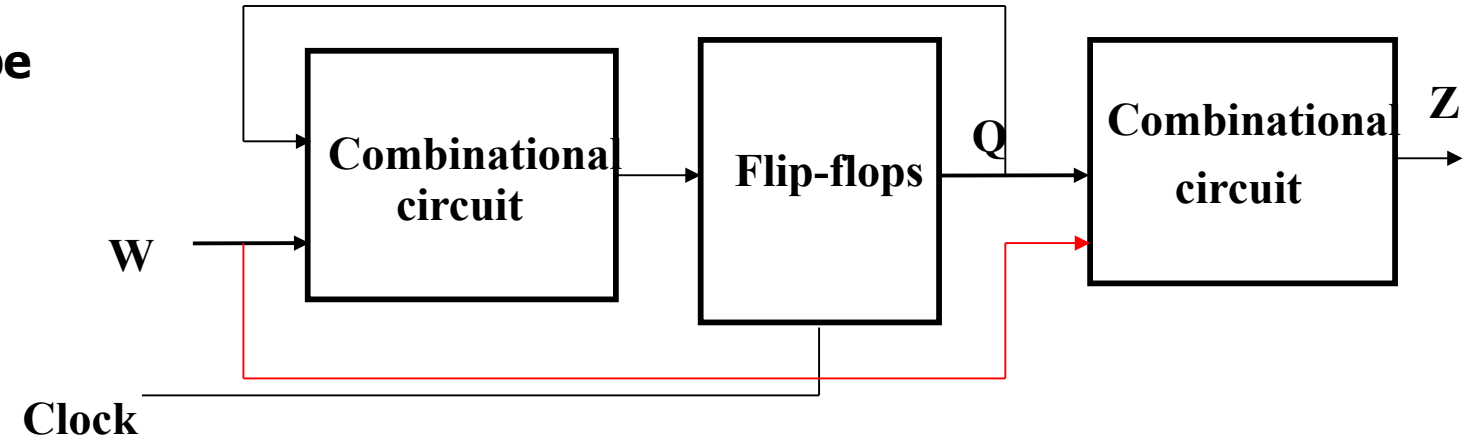


Two types of FSMs

Moore-type



Mealy-type



Example: 1-1 detector

1-1 detector: generate an output z=1 whenever a second w=1 is detected in consecutive clock cycles

Moore-type

The output z be equal to 1 in the clock Cycle that follows the detection of the second w=1

Mealy-type

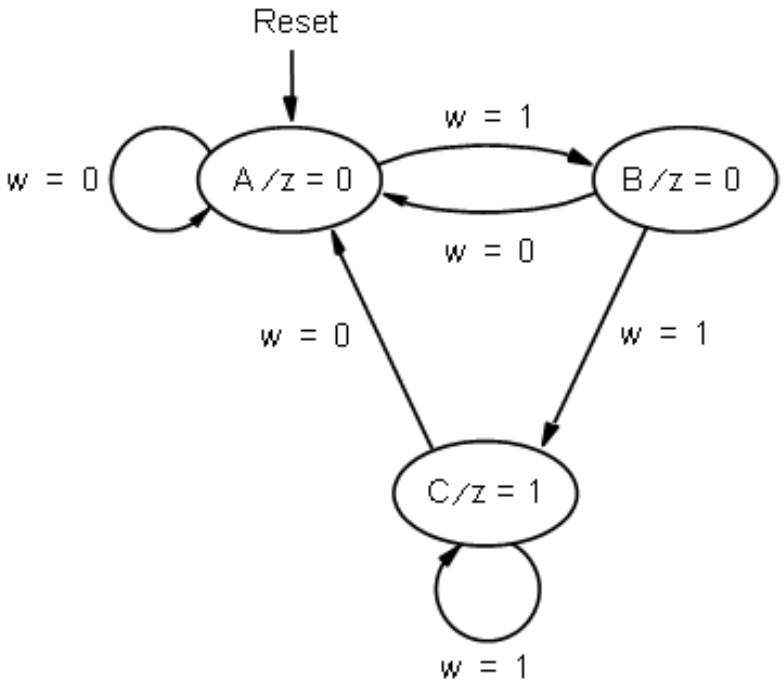
The output z be equal to 1 in the same clock cycle when the second w=1 is detected

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

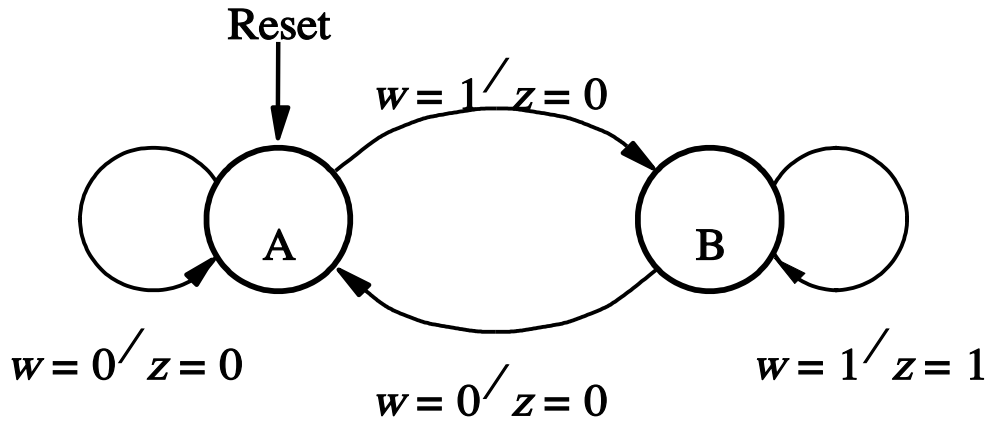
Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

Example: 1-1 detector: state diagram

Moore-type



Mealy-type



Fewer states

Example: 1-1 detector state table

Moore-type

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

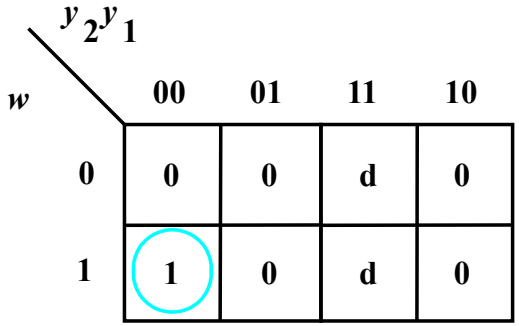
Mealy-type

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

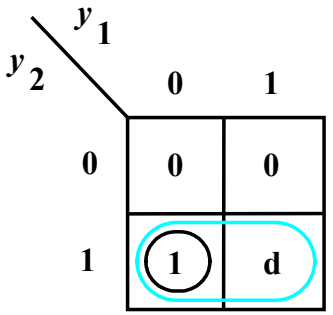
	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

Example: 1-1 detector
Boolean expressions derivation (Moore)

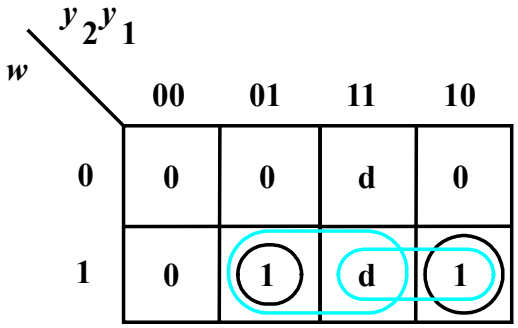
From state-assigned table: we have the following Karnaugh maps



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

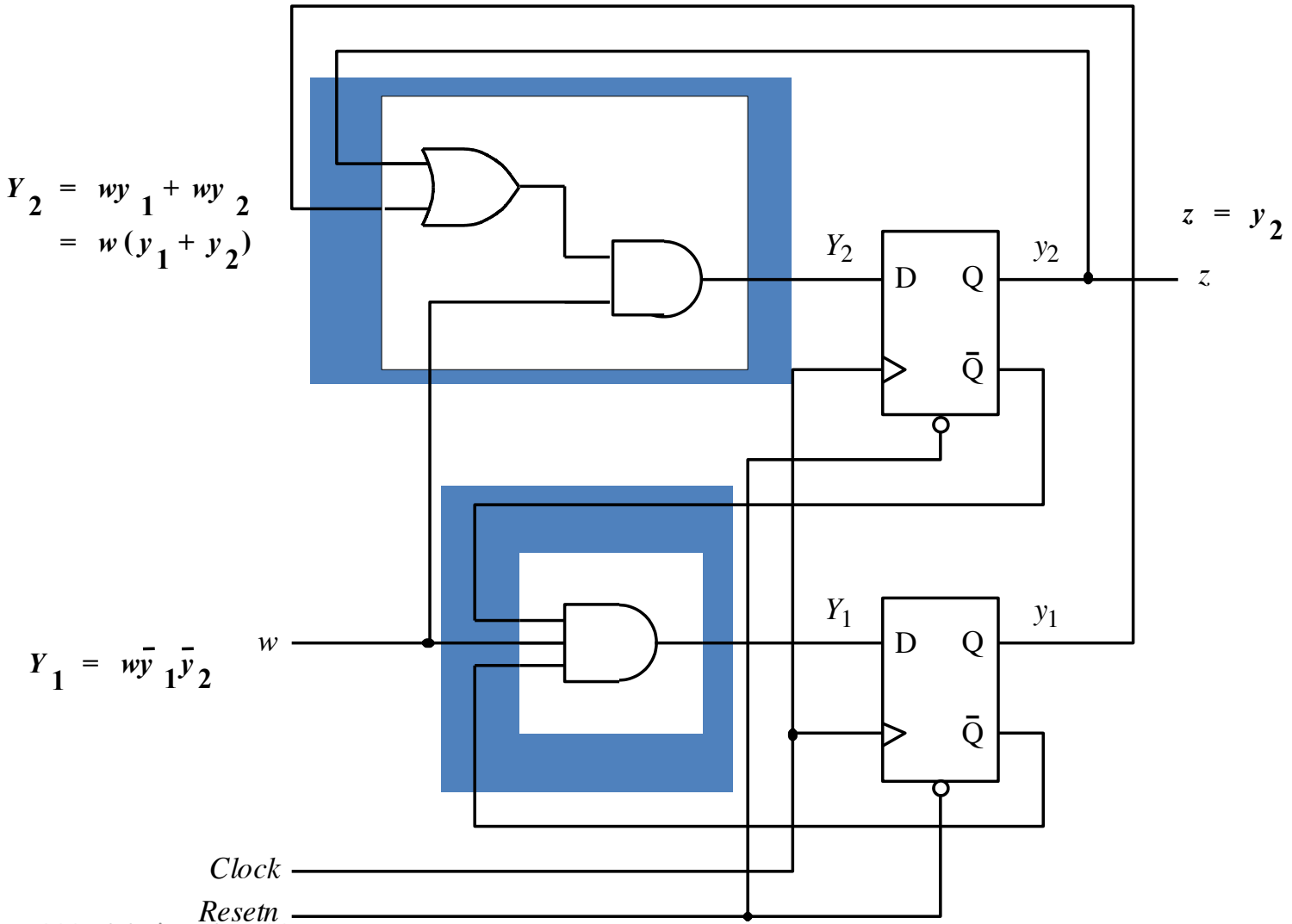


$$z = y_2$$



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

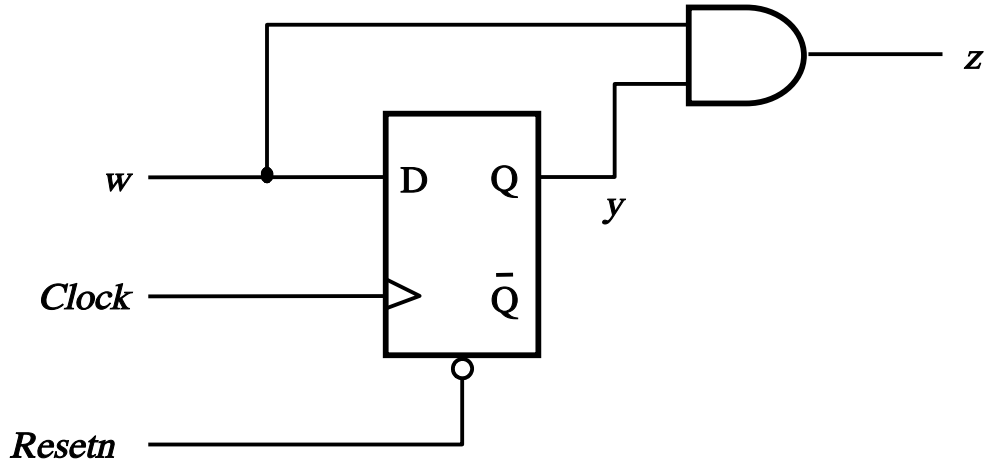
Example: 1-1 detector
Final implementation (Moore)



Example: 1-1 detector
Final implementation (Mealy)

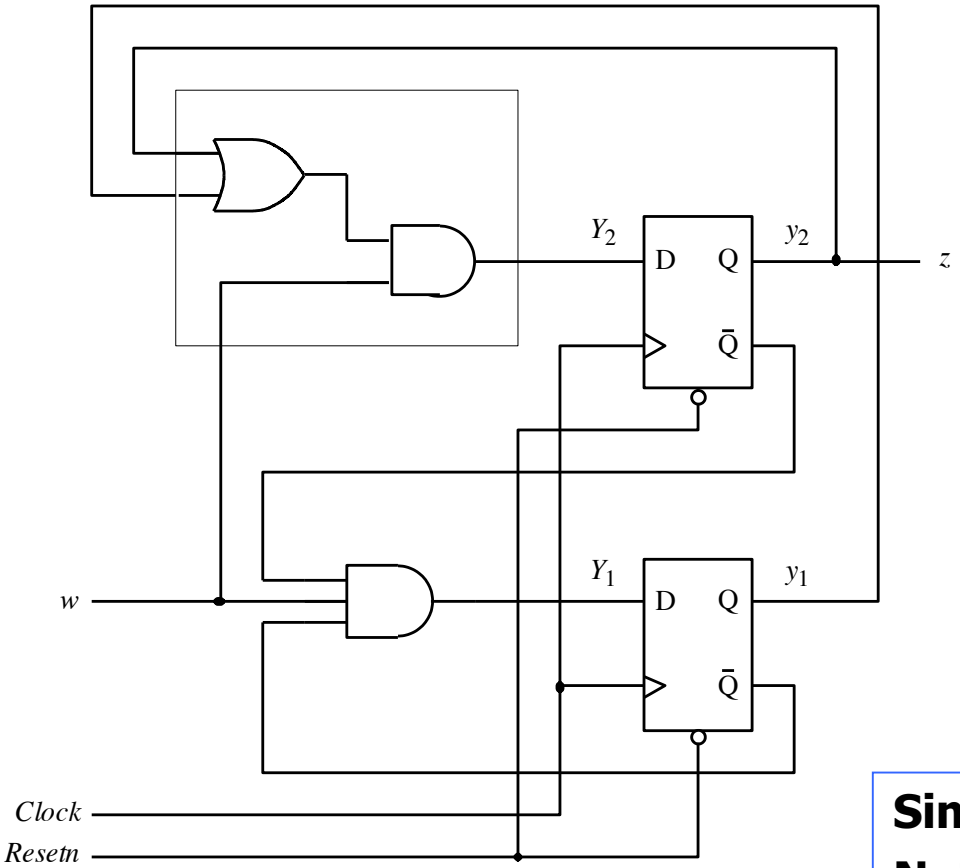
$$Y = w$$

$$z = wy$$

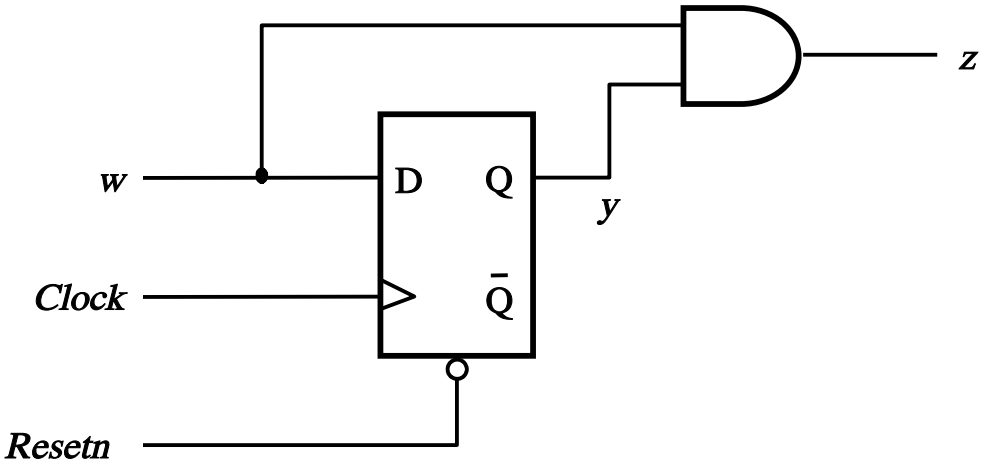


Example: 1-1 detector

Moore-type



Mealy-type



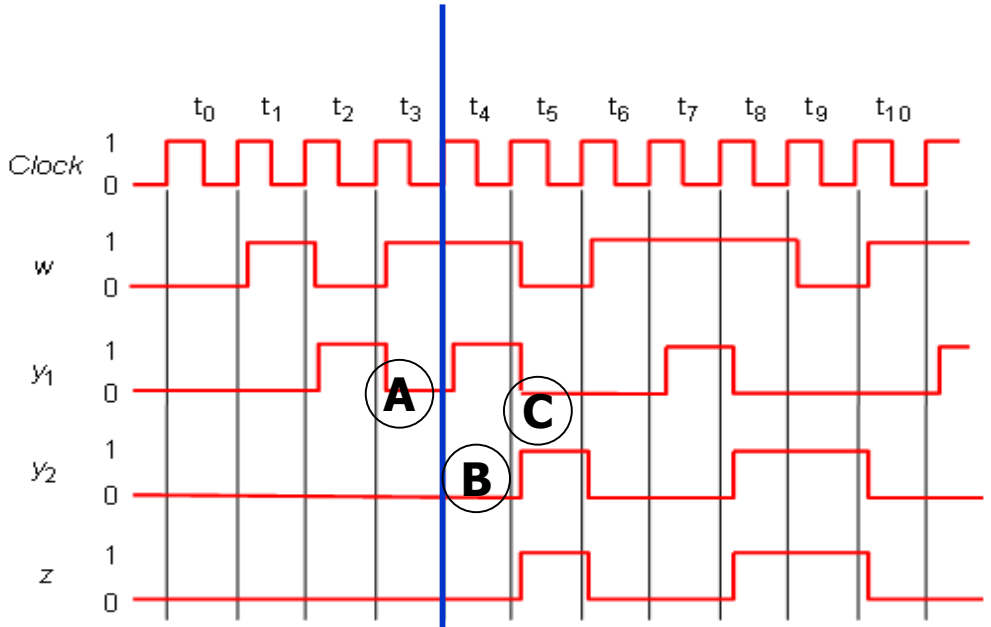
Simpler!

Note that the function is not exactly the same

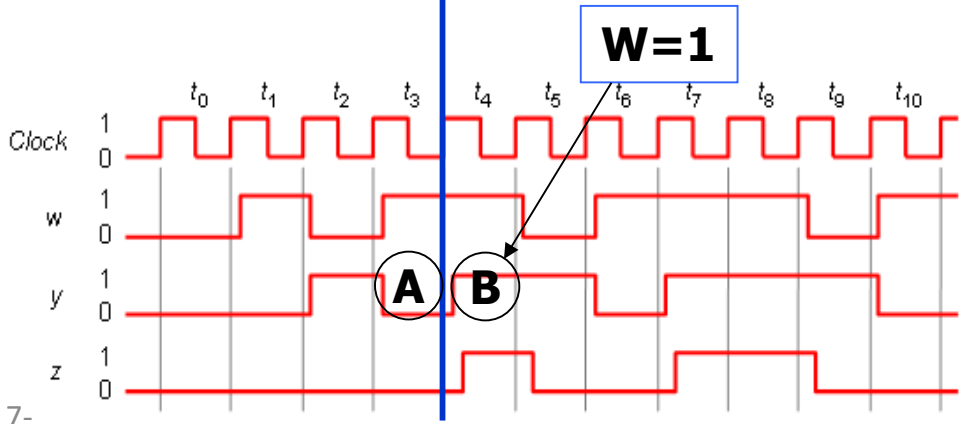
Example: 1-1 detector

Timing diagram

Moore-type

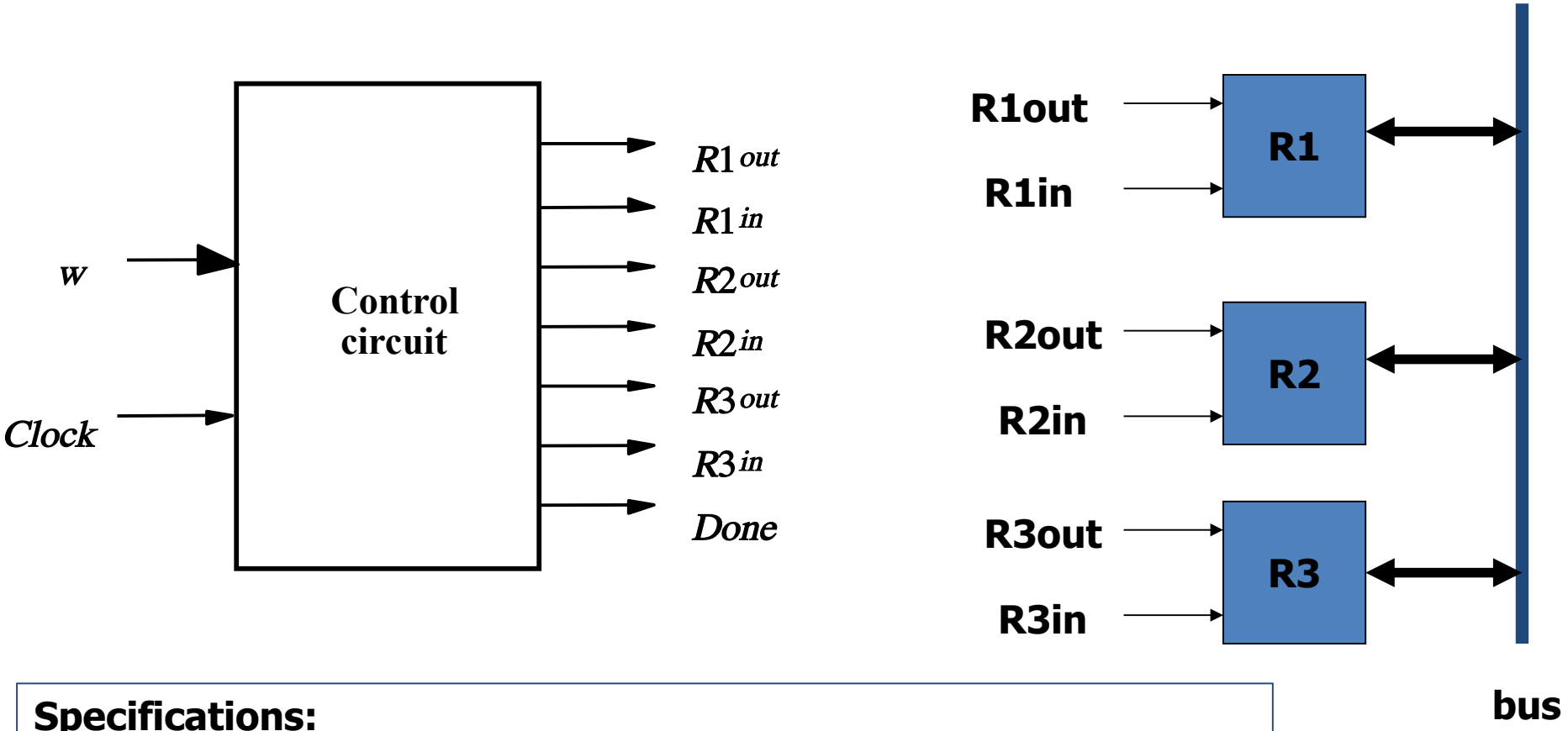


Mealy-type



Swapping Register contents

R2<->R1, using R3 as tmp



Specifications:

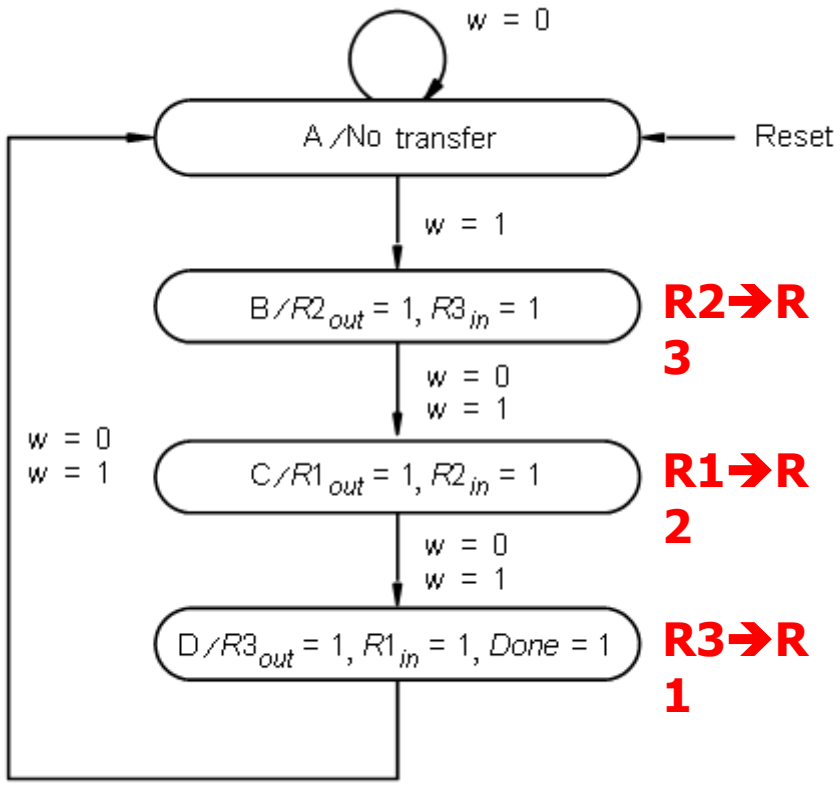
1) After w: 0 →1, R2→R3 (R2out=1, R3in=1).

2) then, R1→R2 (R1out=1, R2in=1), regardless of w

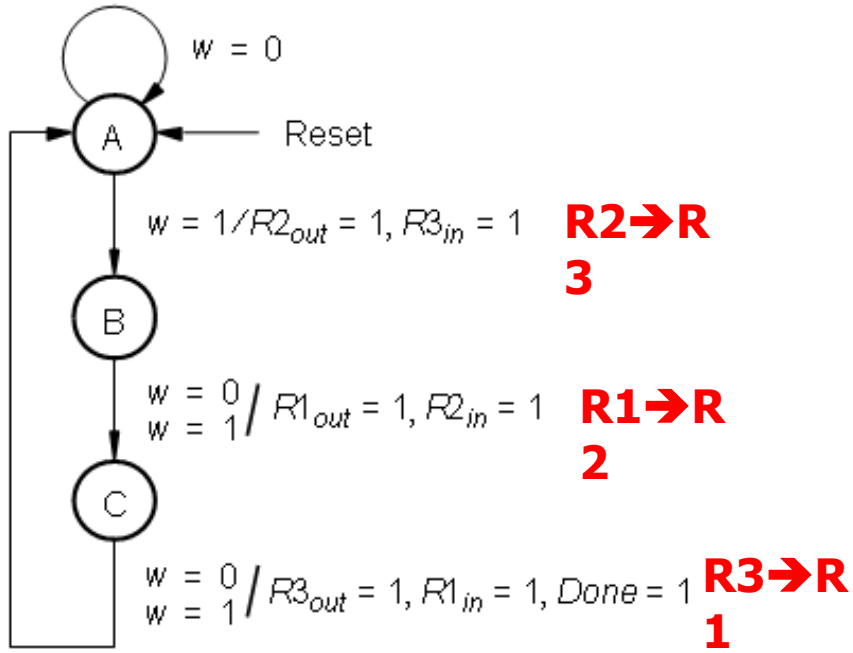
3) then, R3→R1 (R3out=1, R1in=1), regardless of w, done=1.

Example: control circuit for swapping registers

Moore-type



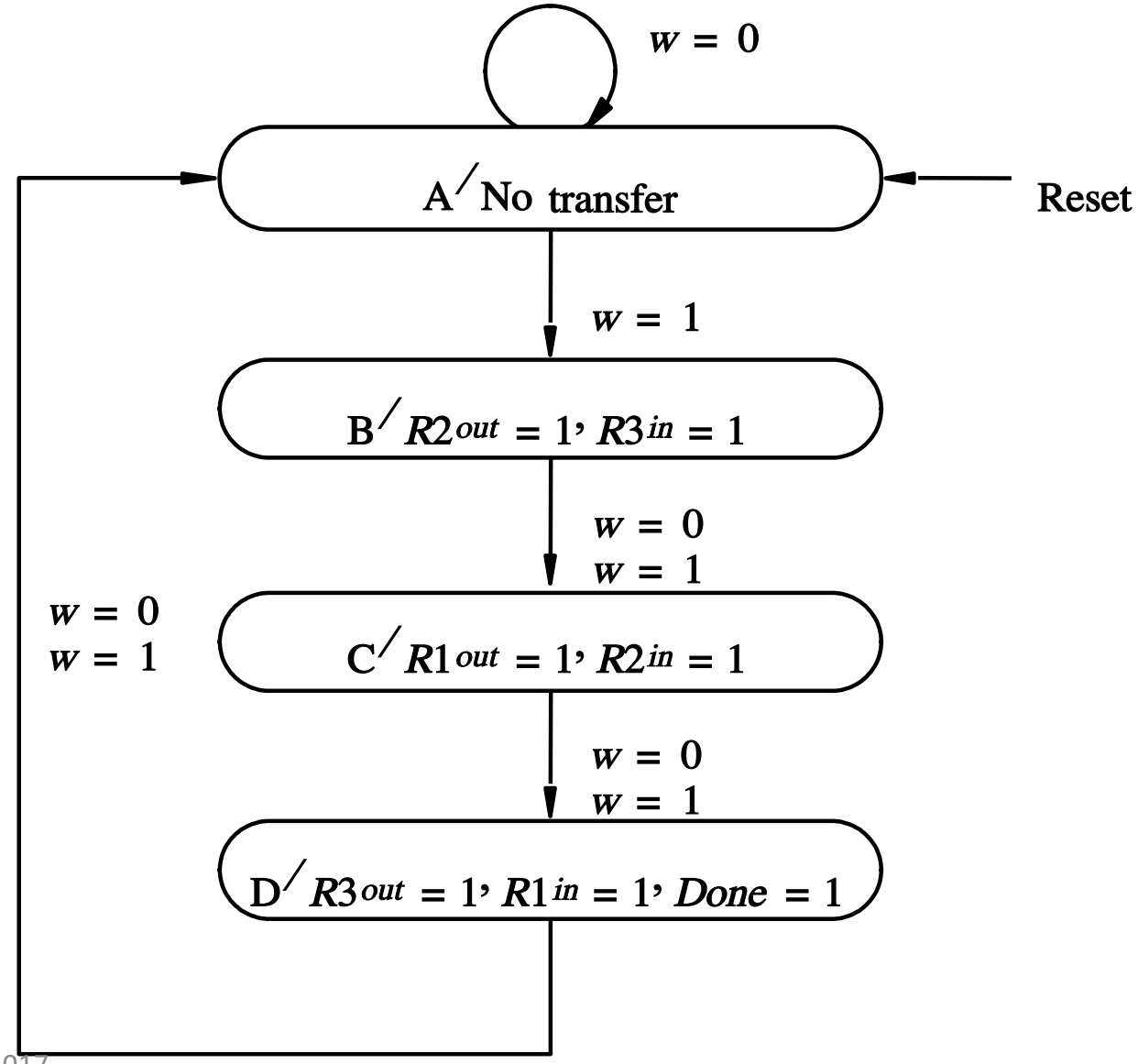
Mealy-type



4 states

3 states

Moore State diagram

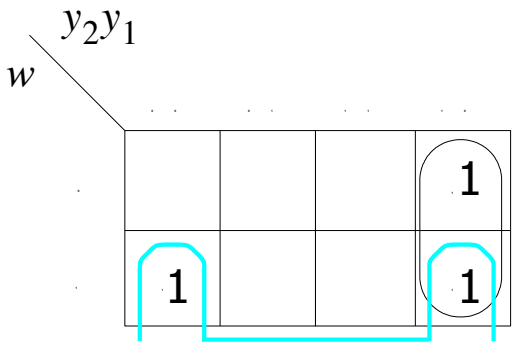


Moore State table

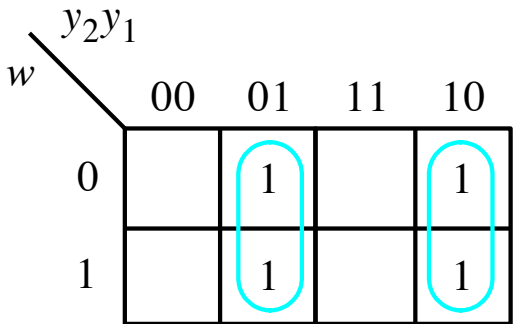
Present state	Next state		Outputs						
	$w=0$	$w=1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	A	B	0	0	0	0	0	0	0
B	C	C	0	0	1	0	0	1	0
C	D	D	1	0	0	1	0	0	0
D	A	A	0	1	0	0	1	0	1

	Present state	Nextstate		Outputs						
		$w=0$	$w=1$							
	y_2y_1	Y_2Y_1	Y_2Y_1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	0 1	0	0	0	0	0	0	0
B	01	10	1 0	0	0	1	0	0	1	0
C	10	11	1 1	1	0	0	1	0	0	0
D	11	00	0 0	0	1	0	0	1	0	1

Moore:
Expressions of next state and outputs



$$Y_1 = w\bar{y}_1 + \bar{y}_1 y_2$$



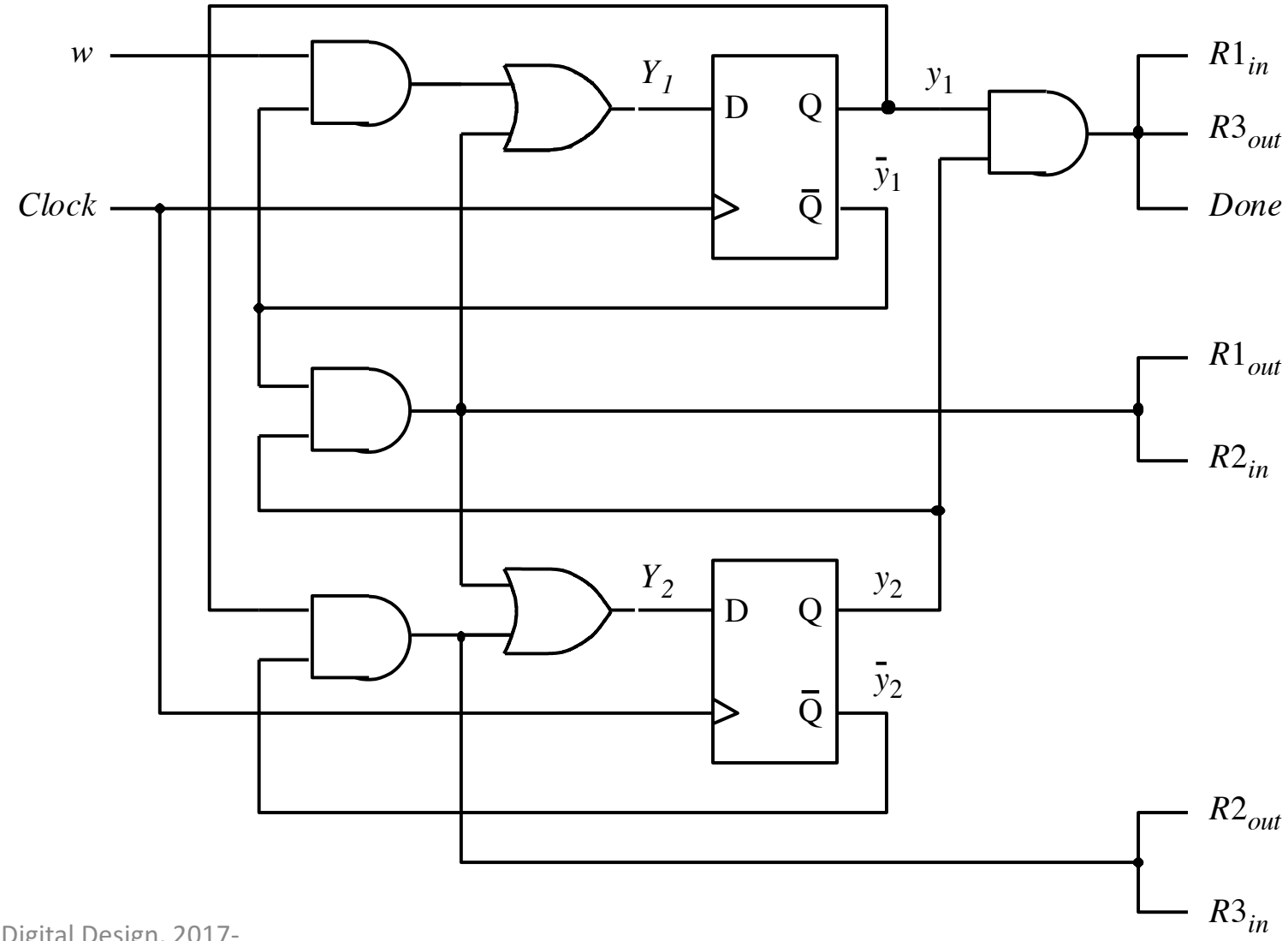
$$Y_2 = y_1\bar{y}_2 + \bar{y}_1 y_2$$

$$R1_{out} = R2_{in} = \bar{y}_1 y_2$$

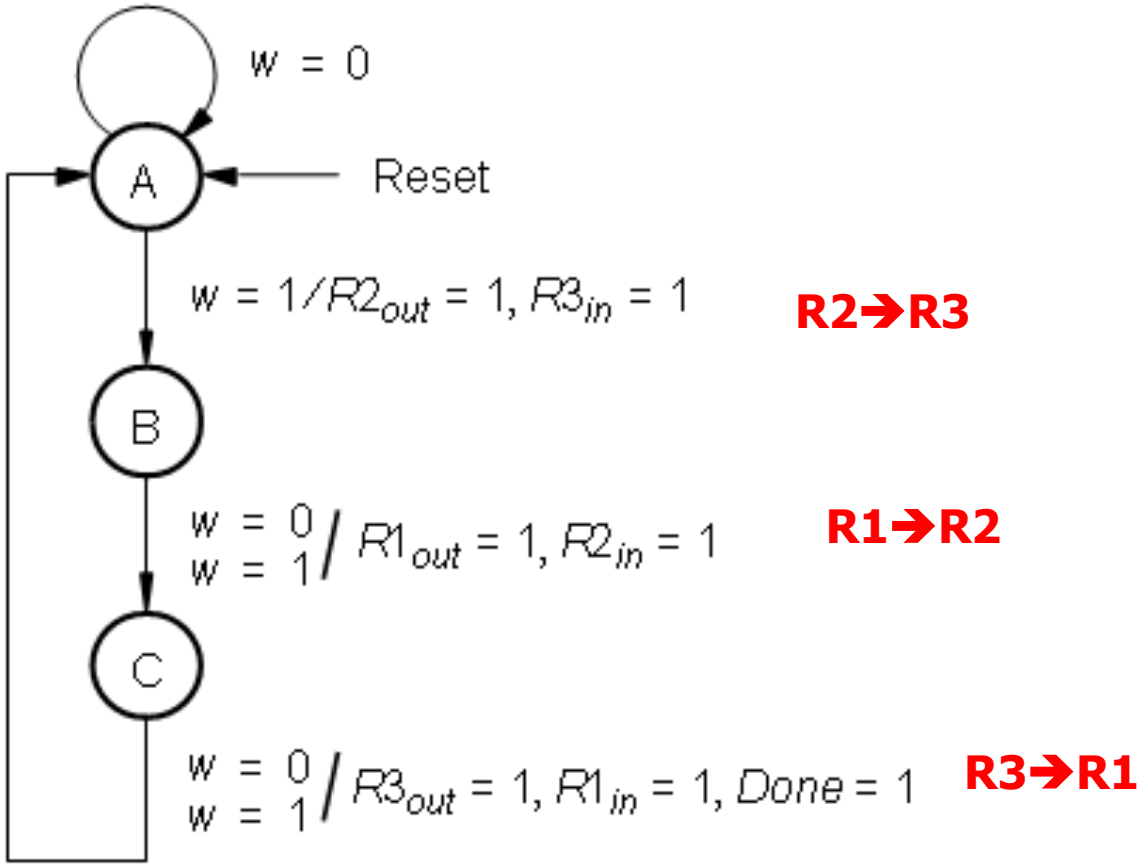
$$R1_{in} = R3_{out} = Done = y1y2$$

$$R2_{out} = R3_{in} = y_1\bar{y}_2$$

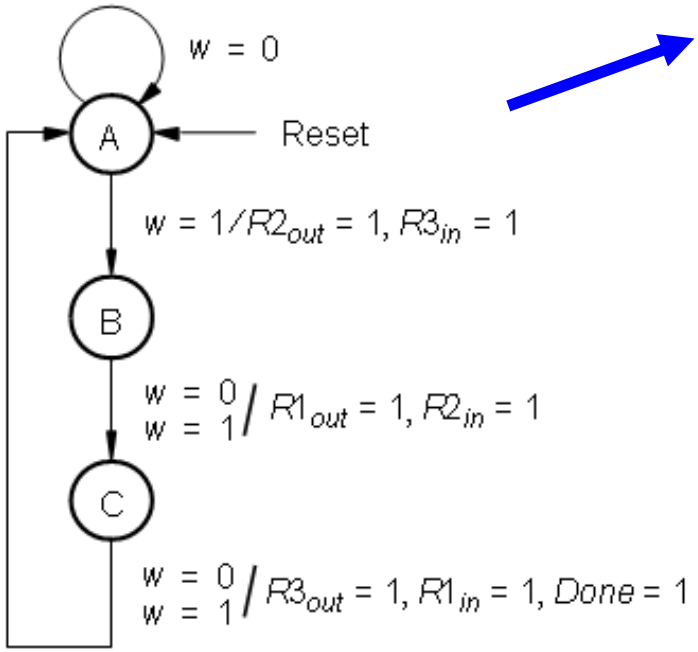
Final Moore implementation



Mealy state diagram



Mealy-type



3 states

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	R2out,R3in
B	C	C	R1out, R2in	
C	A	A	R3out, R1in,Done	

By inspection

y3y2y1

A:0 0 1
B:0 1 0
C:1 0 0

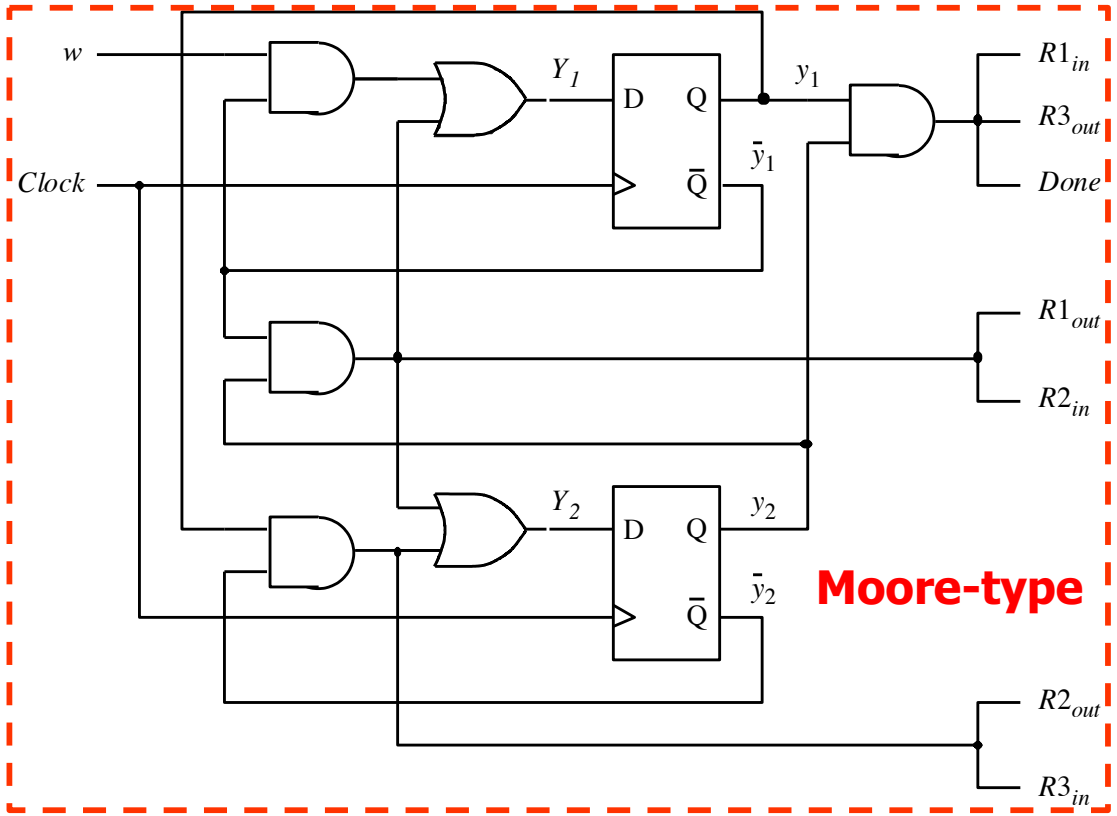
One-hot

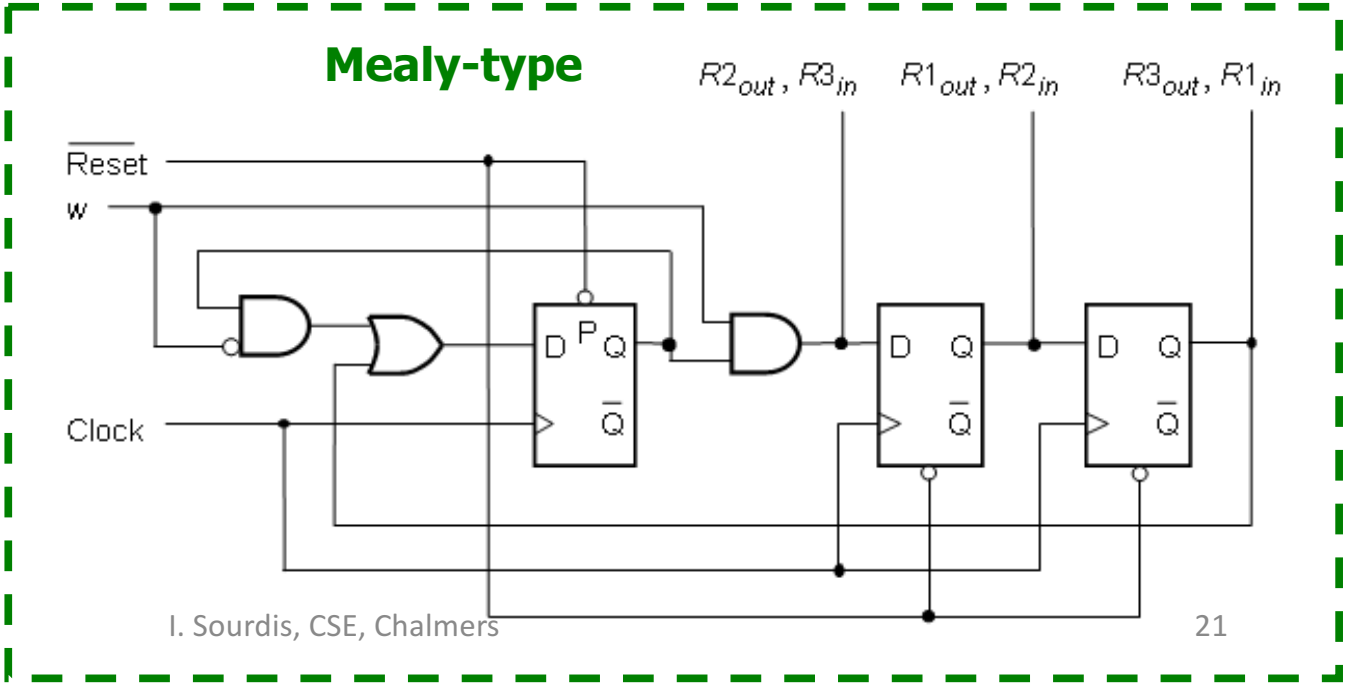
Y1=$\bar{w}$y1+y3 **R2out=R3in=wy1**

Y2=wy1 **R1out=R2in=y2**

Y3=y2 **R3out=R1in=done=y3**

Example: control circuit for swapping registers





Summary of FSMs

Most of tasks can be realized by Moore-type FSM, as well as by Mealy-type FSM, although these two type FSMs don't have the exact same logic function

Compared to Moore-type FSMs, Mealy-type FSMs have following features:

Less states;

Quicker response;

Usually, simpler circuit implementation;

But, sometimes longer critical path, all the way from input to output

(v.s. Moore: from state registers to output)

- one solution: registered outputs in Mealy

FSM design by CAD tools

- Different ways of FSM design using CAD tools:
  - Derive circuits from a state diagram manually → draw a schematic into CAD systems or HDL code → simulate → implement in a chip (e.g.PLD).
  - Enter state diagram into CAD systems using a graphical tool → automatically synthesize.
  - Write HDL code for state diagram → automatically synthesize. (✓)

Quiz 7-1

http://m.socrative.com/student/#joinRoom

room number: 713113

- Q1: A Mealy machine would change its output at the same clock cycle an input is modified.
  - True/false
- Q2: Each state of a Mealy machine has a single set of outputs
  - True/false
- Q3: The circuit of a Mealy machine is usually simpler
  - True/false

State Assignment Problem

State Assignment Problem

- Some state assignments are better than others.
- The state assignment influences the complexity of the state machine.
  - **The combinational logic required in the state machine design is dependent on the state assignment.**

- Types of state assignment
  - Binary encoding: 2^N states → N Flip-Flops
  - Gray-code encoding: 2^N states → N Flip-Flops
  - One-hot encoding: N states → N Flip-Flops

FSM: State Assignment

Example:

Design a FSM that detects a sequence of two or more consecutive ones on an input bit stream.

The FSM should output a 1 when the sequence is detected, and a 0 otherwise.

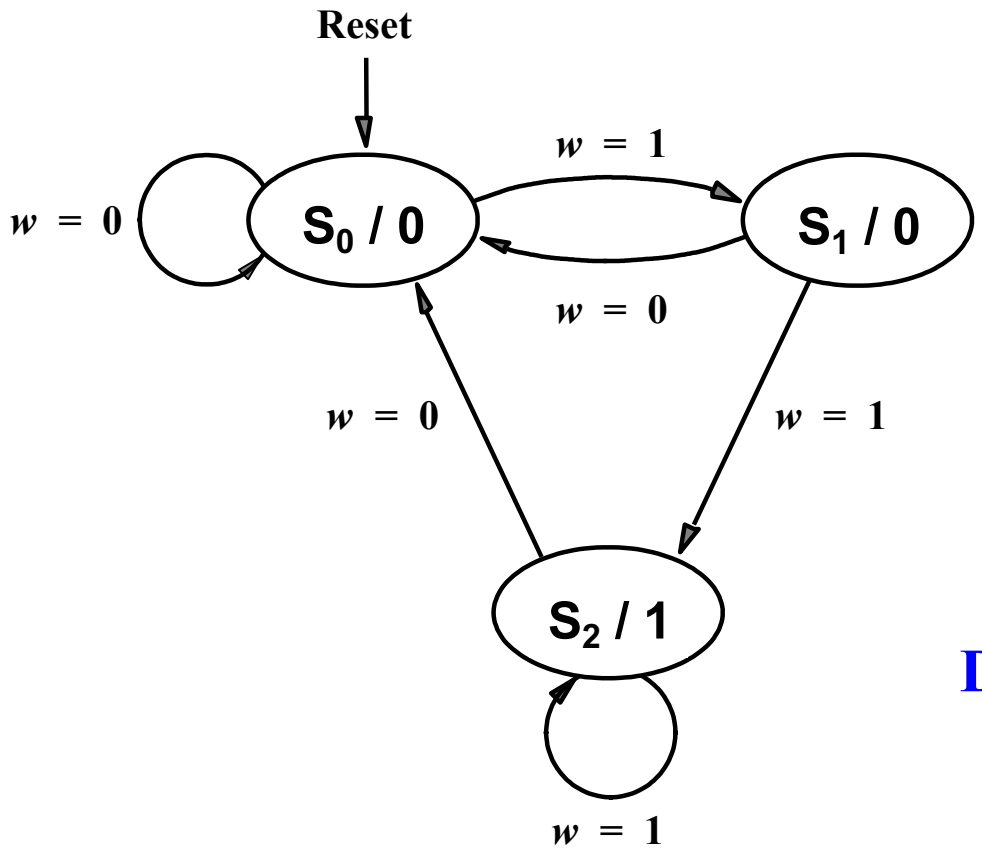
This is another example of a *sequence detector*.

FSM: State Assignment

Input: 0 1 1 1 0 1 0 1 1 0 1 1 1 0 1 ...

Output: 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 ...

FSM: State Assignment



State Diagram

FSM: State Assignment

Present State	Next State		Output
	w = 0	w = 1	
S_0	S_0	S_1	0
S_1	S_0	S_2	0
S_2	S_0	S_2	1

State Table

FSM: State Assignment #1

State Assigned Table

Present State			Next State						Output
			w = 0			w = 1			
	Q_A	Q_B		Q_A^+	Q_B^+		Q_A^+	Q_B^+	z
S_0	0	0	S_0	0	0	S_1	0	1	0
S_1	0	1	S_0	0	0	S_2	1	0	0
S_2	1	0	S_0	0	0	S_2	1	0	1
	1	1		d	d		d	d	d

Using Binary Encoding for the State Assignment

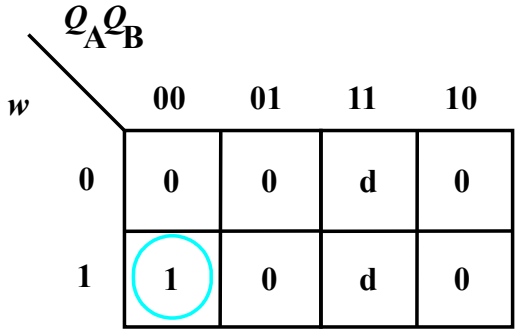
FSM: State Assignment #1

State Assigned Table

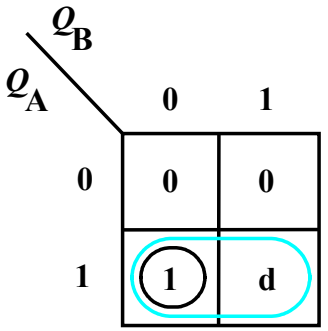
Present State			Next State				FF Inputs			
			w = 0		w = 1		w = 0		w = 1	
	Q_A	Q_B	Q_A^+	Q_B^+	Q_A^+	Q_B^+	D_A	D_B	D_A	D_B
S_0	0	0	0	0	0	1	0	0	0	1
S_1	0	1	0	0	1	0	0	0	1	0
S_2	1	0	0	0	1	0	0	0	1	0
	1	1	d	d	d	d	d	d	d	d

Characteristic Equation: $D = Q^+$

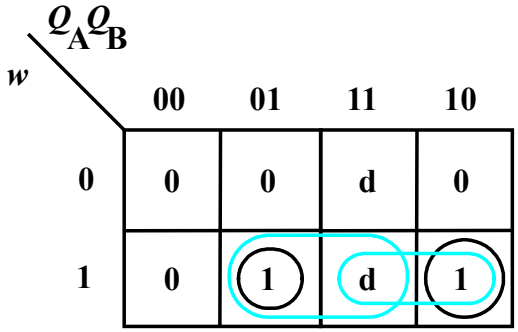
FSM: State Assignment #1



$$D_B = w\bar{Q}_A\,\bar{Q}_B$$

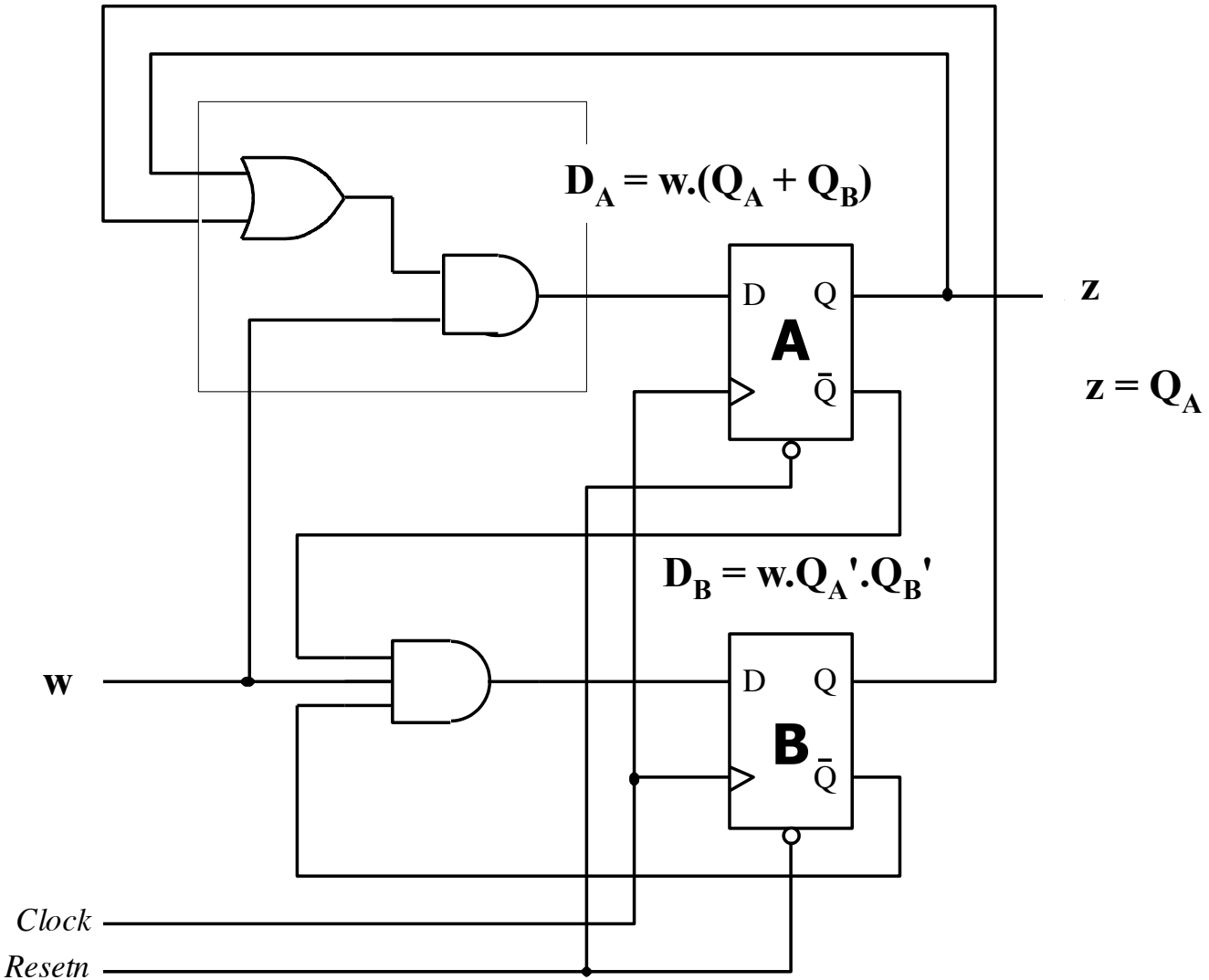


$$z = Q_A$$



$$D_A = wQ_A + wQ_B$$
$$= w(Q_A + Q_B)$$

FSM: State Assignment #1



FSM: State Assignment #2

State Assigned Table

Present State			Next State						Output
			w = 0			w = 1			
	Q_A	Q_B		Q_A^+	Q_B^+		Q_A^+	Q_B^+	z
S_0	0	0	S_0	0	0	S_1	0	1	0
S_1	0	1	S_0	0	0	S_2	1	1	0
S_2	1	1	S_0	0	0	S_2	1	1	1
	1	0		d	d		d	d	d

Using Gray-code Encoding for the State Assignment

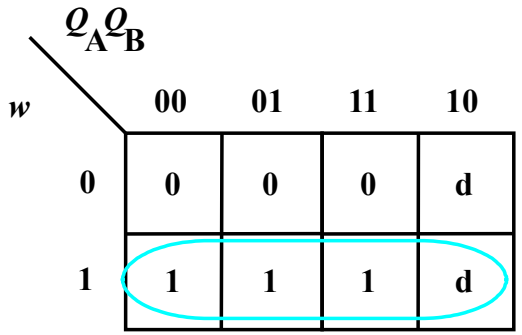
FSM: State Assignment #2

State Assigned Table

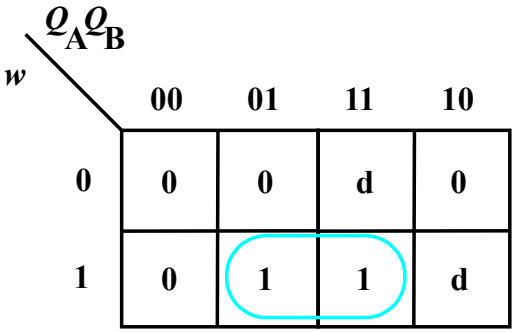
Present State			Next State				FF Inputs			
			w = 0		w = 1		w = 0		w = 1	
	Q_A	Q_B	Q_A^+	Q_B^+	Q_A^+	Q_B^+	D_A	D_B	D_A	D_B
S_0	0	0	0	0	0	1	0	0	0	1
S_1	0	1	0	0	1	1	0	0	1	1
S_2	1	1	0	0	1	1	0	0	1	1
	1	0	d	d	d	d	d	d	d	d

Characteristic Equation: $D = Q^+$

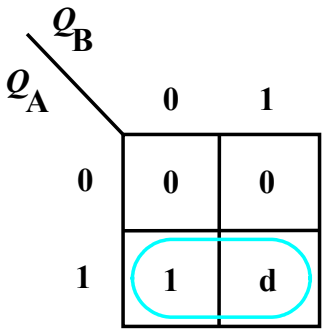
FSM: State Assignment #2



$D_B = w$

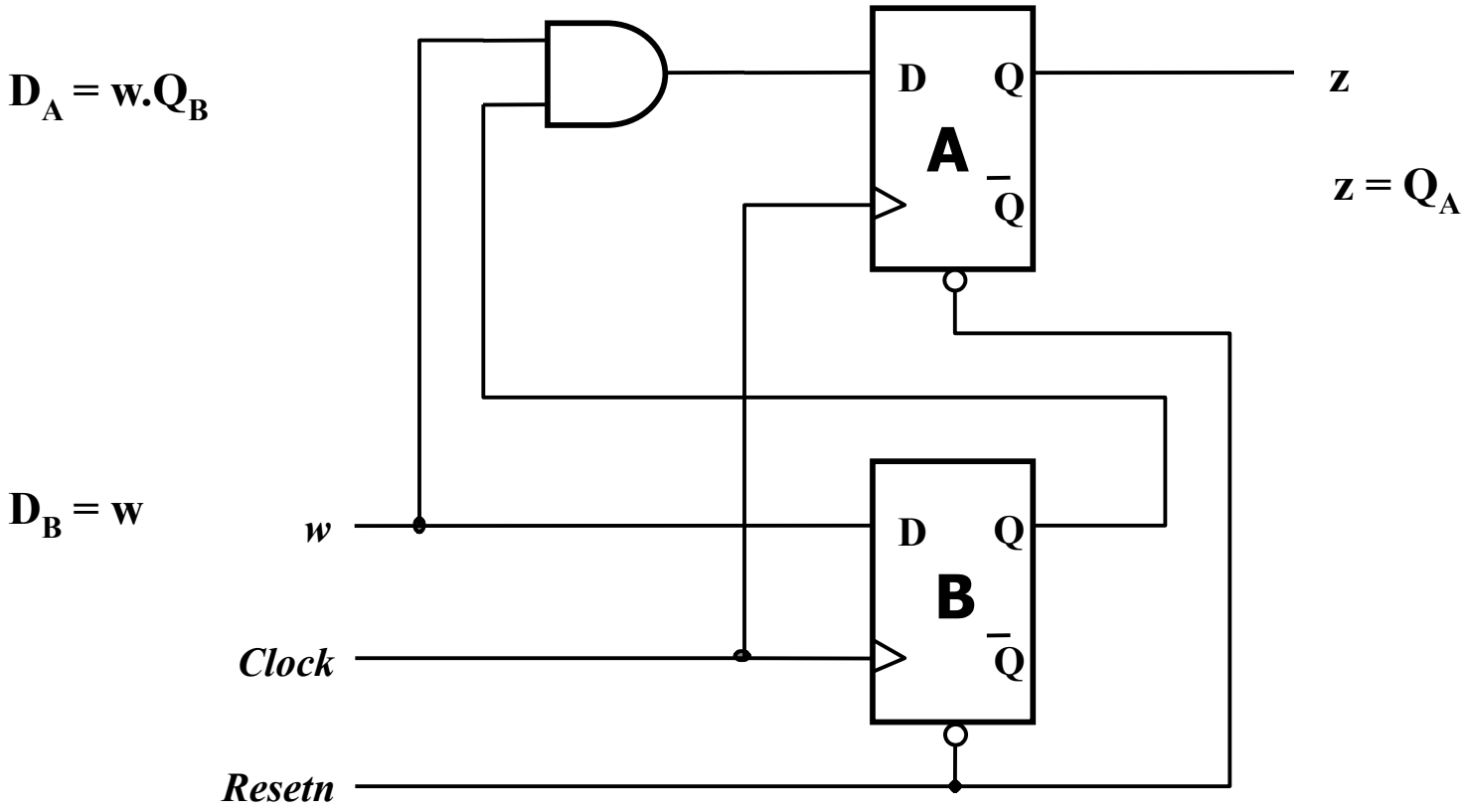


$D_A = wQ_B$



$z = Q_A$

FSM: State Assignment #2



FSM: State Assignment #3

State Assigned Table

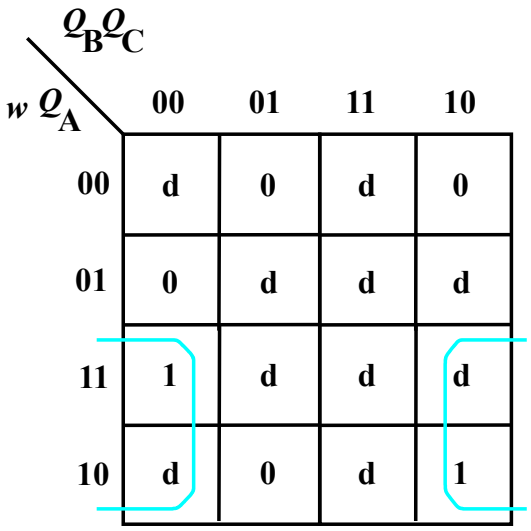
Present State				Next State							
				w = 0				w = 1			
	Q_A	Q_B	Q_C		Q_A^+	Q_B^+	Q_C^+		Q_A^+	Q_B^+	Q_C^+
S_0	0	0	1	S_0	0	0	1	S_1	0	1	0
S_1	0	1	0	S_0	0	0	1	S_2	1	0	0
S_2	1	0	0	S_0	0	0	1	S_2	1	0	0

Using One-hot Encoding for the State Assignment

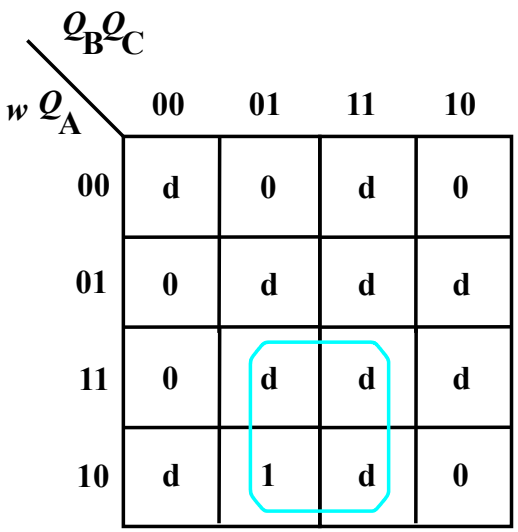
For each state only one flip-flop is set to 1.
The remaining combination of state variables are not used.

Characteristic Equation: $D = Q^+$

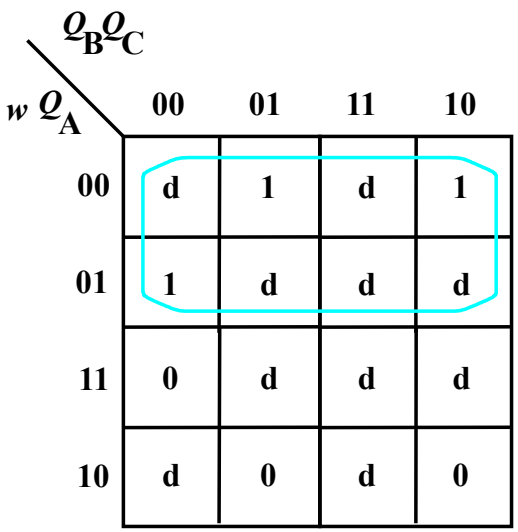
FSM: State Assignment #3



$$D_A = w\bar{Q}_C$$

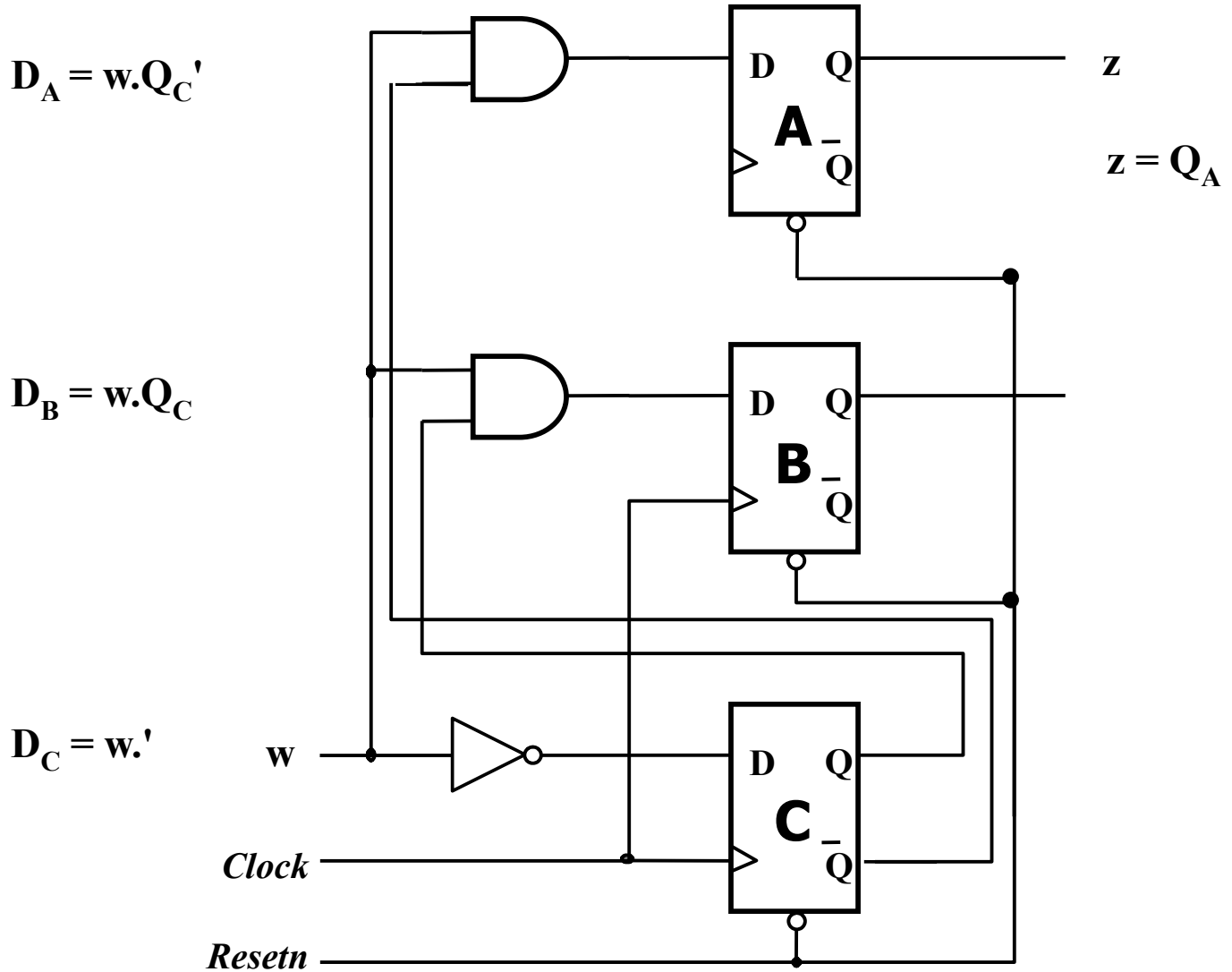


$$D_B = wQ_C$$

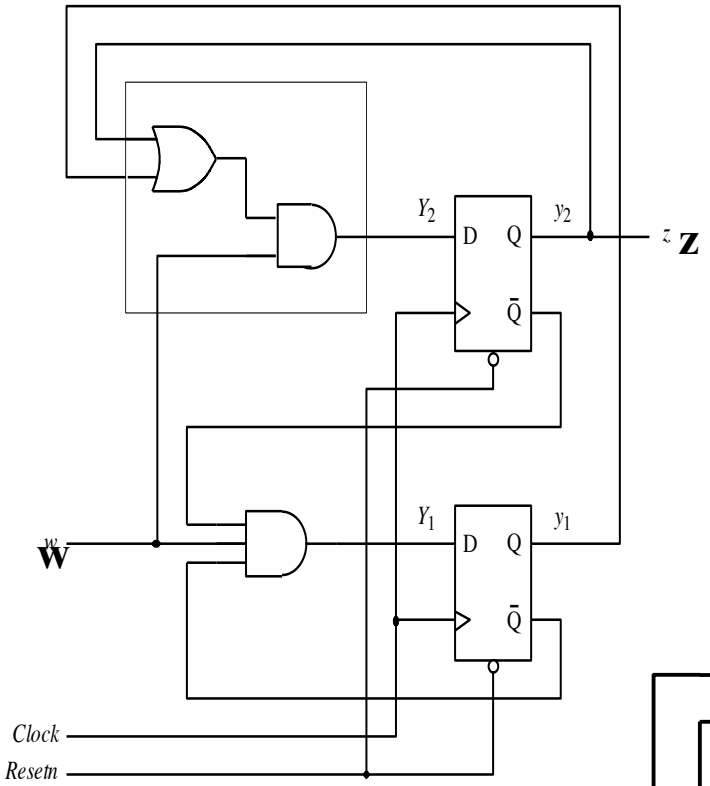


$$D_C = \bar{w}$$

FSM: State Assignment #3

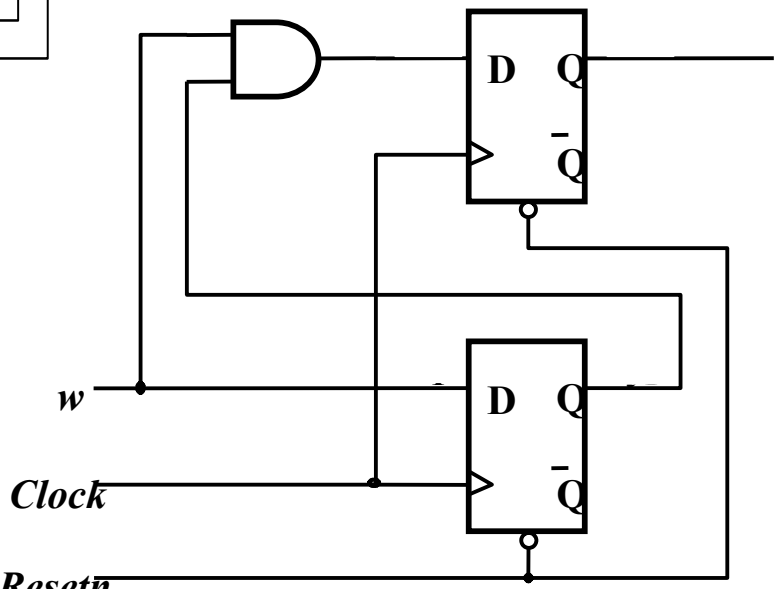


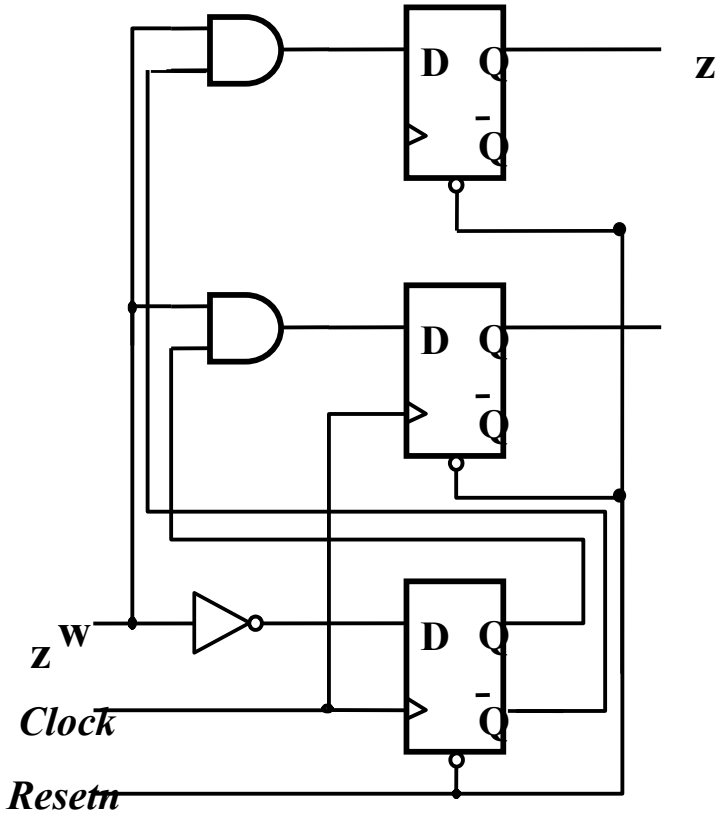
State Assignments: comparison



Binary

Gray





One-hot

State Minimization

FSM: State Minimization

Definition:

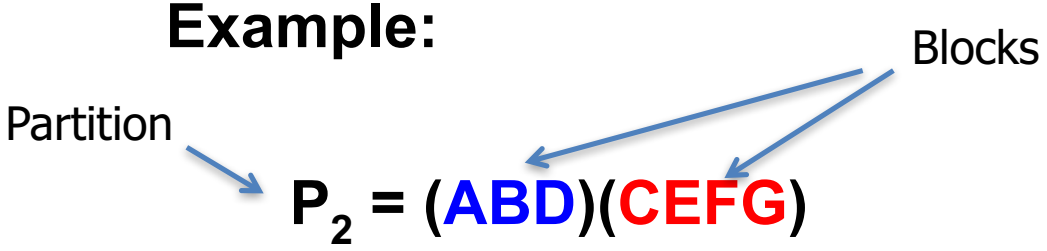
Two states S_i and S_j are said to be equivalent if and only if for every possible input sequence, the same output sequence will be produced regardless of whether S_i or S_j is the initial state.

State Minimization: Partitioning

Definition:

A partition consists of one or more blocks, where each block comprises a subset of states that *may* be equivalent, but the states in a given block are *definitely not* equivalent to the states in other blocks.

Example:



State Minimization: Partitioning

- State Minimization through Partitioning:
  - Form an initial partition (P_1) that includes **all** states.
  - Form a second partition (P_2) by separating the states into blocks based upon their output values.
  - Form a third partition (P_3) by separating the states into blocks corresponding to the next state values.
  - Continue partitioning until two successive partitions are the same (i.e. $P_{N-1} = P_N$).
  - All states in any one block are equivalent.
    - Equivalent states can be combined into a single state.

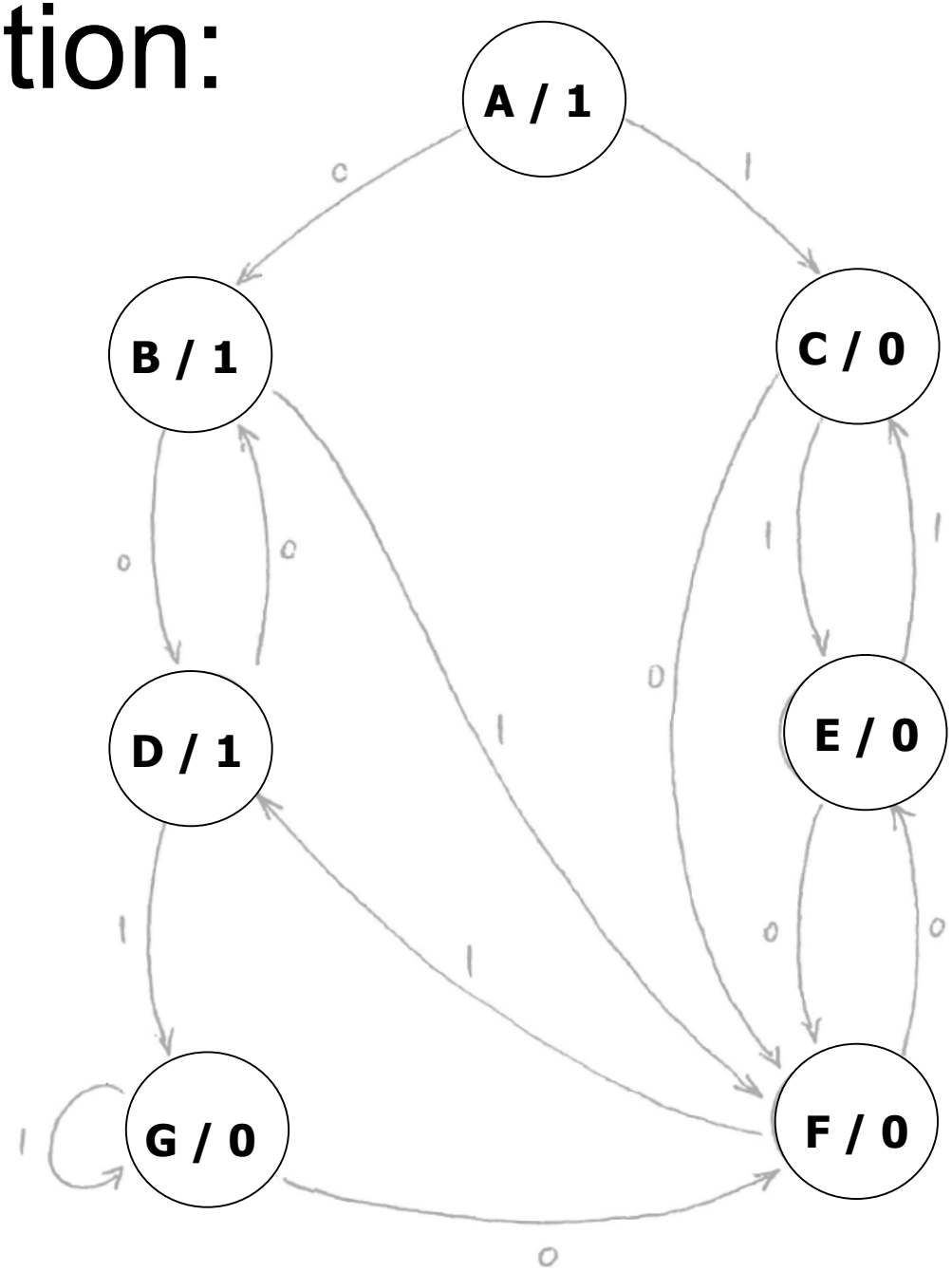
State Minimization: Partitioning

Example:

Use partitioning to minimize the number of states in the following Finite State Machine (FSM).

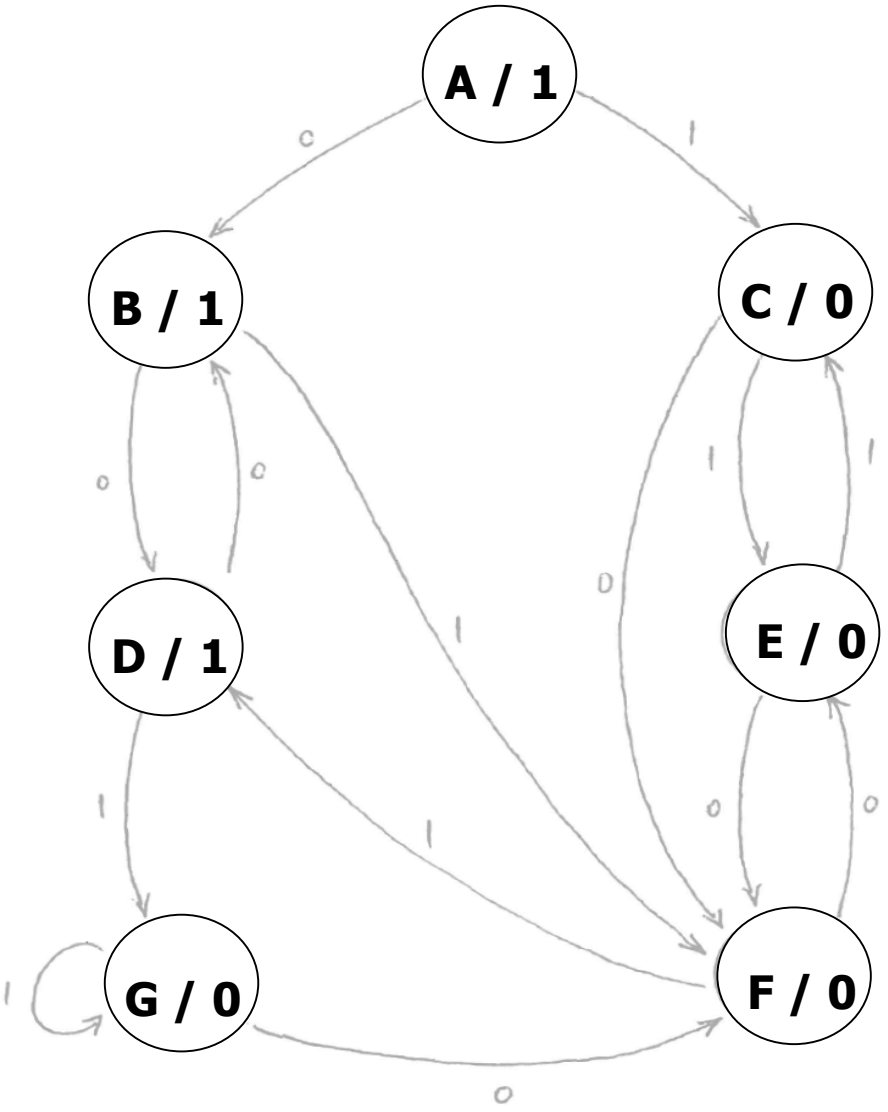
State Minimization: Partitioning

State Diagram



State Minimization: Partitioning

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



State Minimization: Partitioning

Initial Partition:

P_1 = (ABCDEFG)

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

The initial partition contains all states in the state diagram / table.

State Minimization: Partitioning

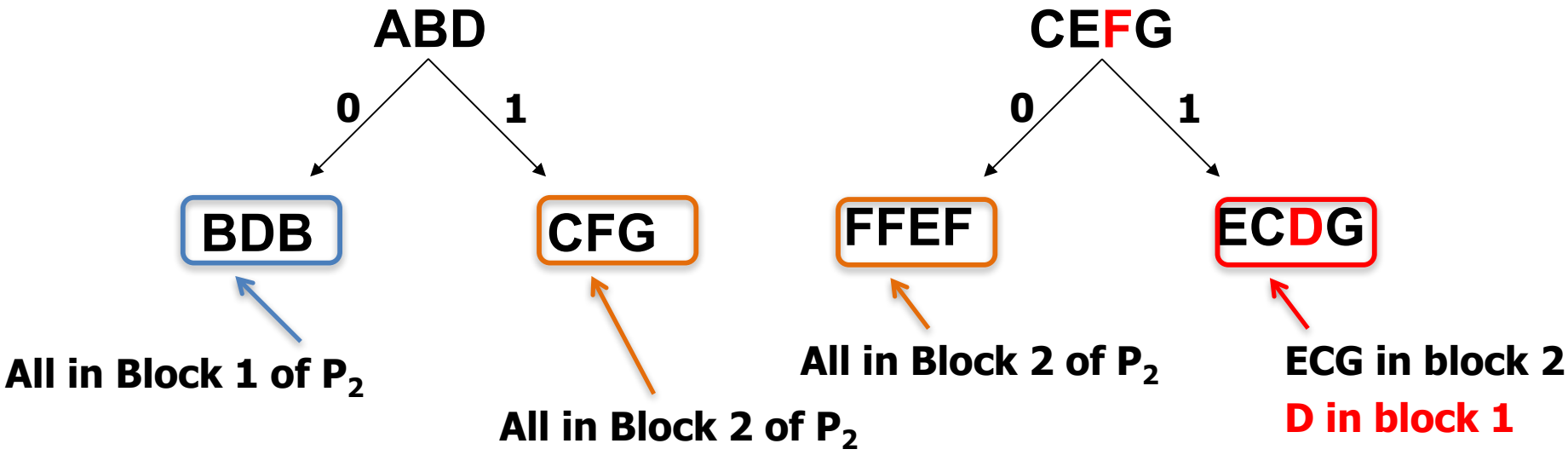
- Separate states based on output value.
  - P_2 = (ABD)(CEFG)

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

State Minimization: Partitioning

- Separate states based on next state values.
- P_2 = (ABD)(CEFG)

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

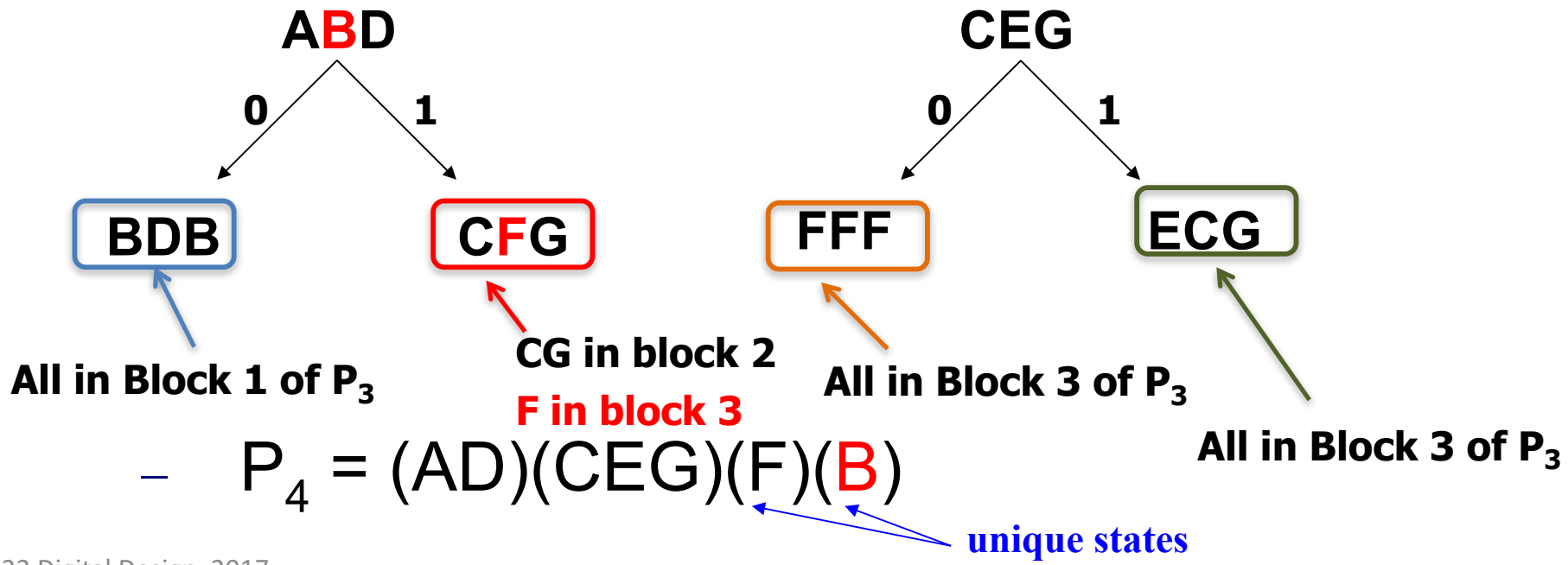


- P_3 = (ABD)(CEG)(F) unique state

State Minimization: Partitioning

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

- Separate states based on next state values.
- $P_3 = (ABD)(CEG)(F)$



- $P_4 = (AD)(CEG)(F)(B)$

State Minimization: Partitioning

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

- Separate states based on next state values.

- $P_5 = (AD)(CEG)(F)(B)$

P_5 same as previous partition (P_4) => state minimization is completed

State Minimization: Partitioning

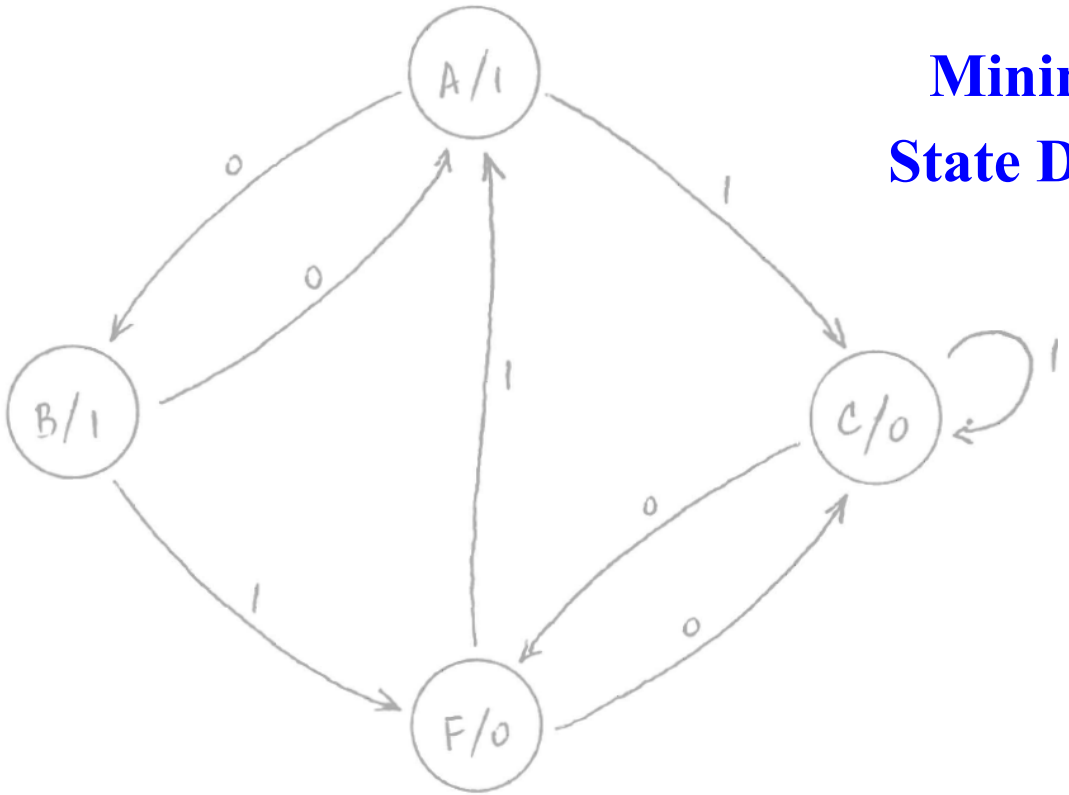
- Since $P_4 = P_5$, state minimization is complete.
- The equivalent states are:
  - A = D
  - C = E = G
  - B
  - F
- Thus, the FSM can be realized with just 4 states.

FSM: State Minimization

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

Minimized State Table

FSM: State Minimization

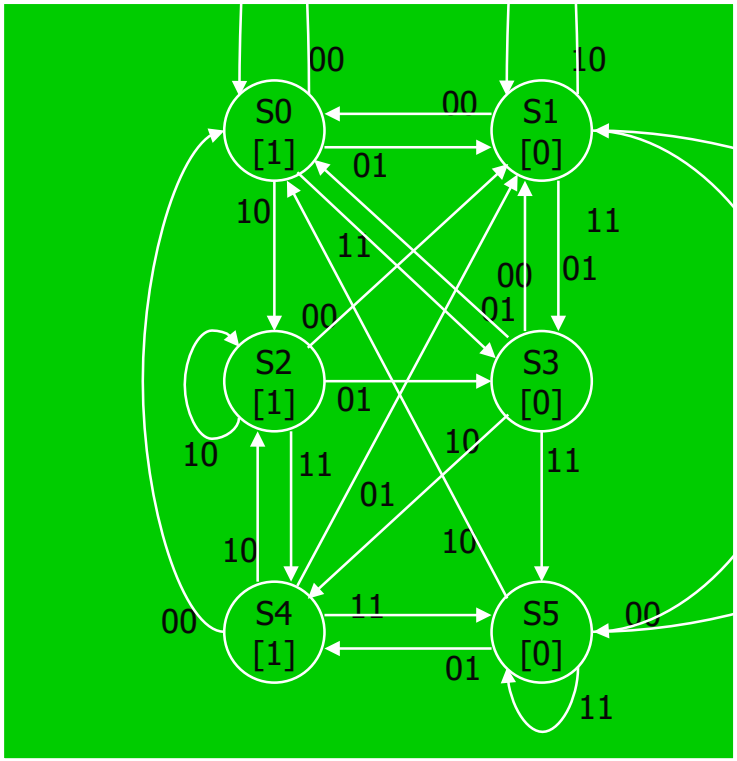


Minimized State Diagram

State minimization: Implication Table

- Multiple input example

inputs here

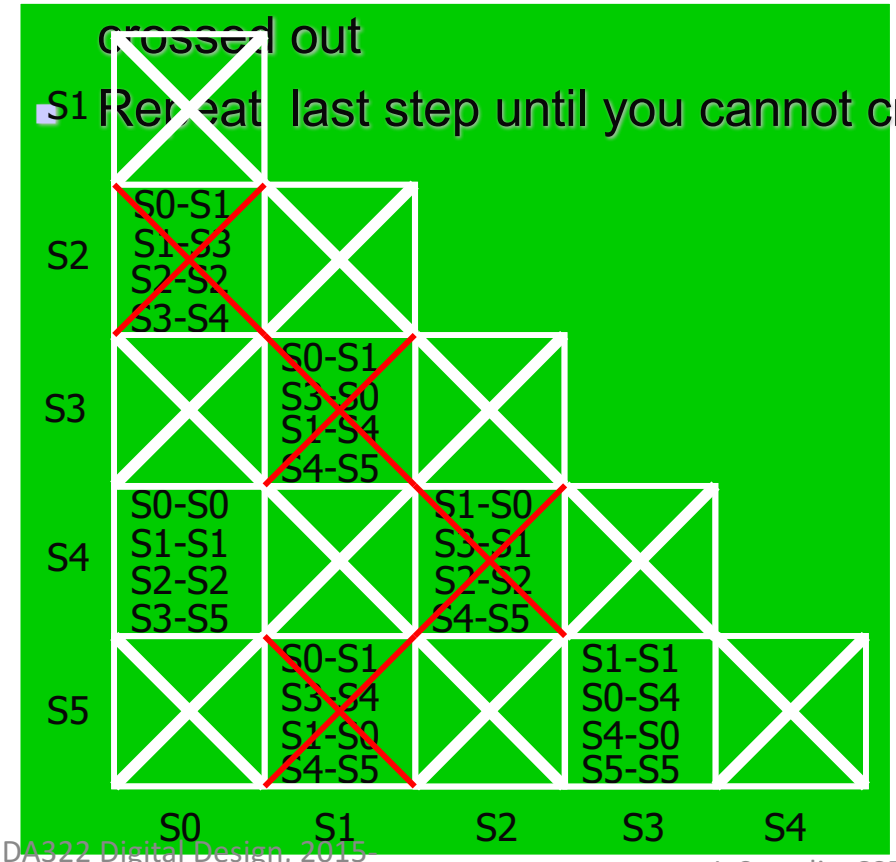


present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

symbolic state transition table

Implication Table Method

- Build a table and label vertical axis: S_1, S_2,.. S_N and horizontal S_0, S_1,.. S_{N-1}
  - **Each cell is the crossection of a pair of states S_i-S_j**
- Cross out pairs of states with different outputs
- In the rest of the cells put the pairs of **next**-states that need to be the same for S_i-S_j to be the same
- Then cross-out more cells if at least one next-state pair has been already crossed out
- Repeat last step until you cannot cross-out any more cells

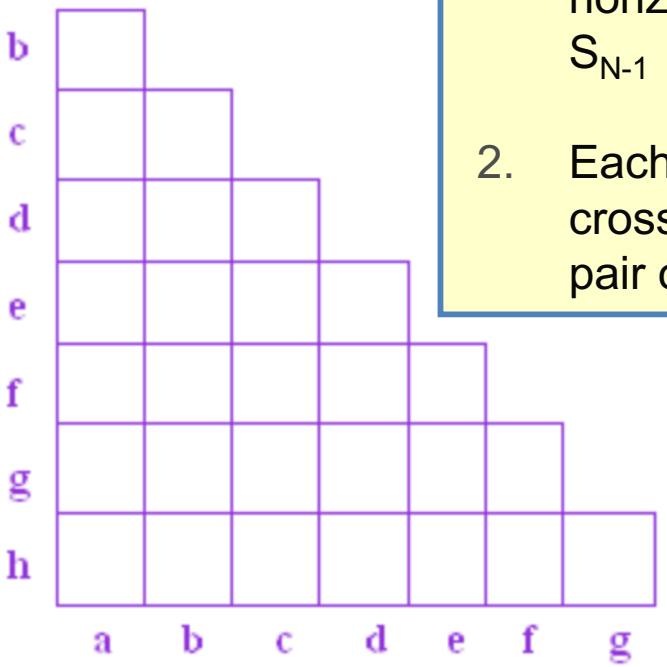


present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S3	S2	S5	1
S5	S1	S4	S0	S5	0

minimized state table
(S0==S4) (S3==S5)

Implication Table Method example 2

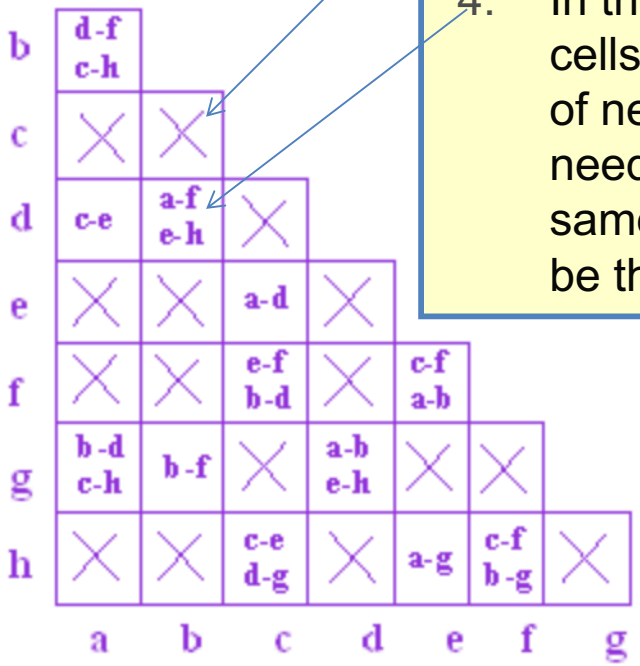
Current state	NxtState IN: 0	1	OUT
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



1. Build a table and label vertical axis: S_1, S_2,.. S_N and horizontal S_0, S_1,.. S_{N-1}
2. Each cell is the crossection of a pair of states S_i-S_j

Implication Table Method example 2

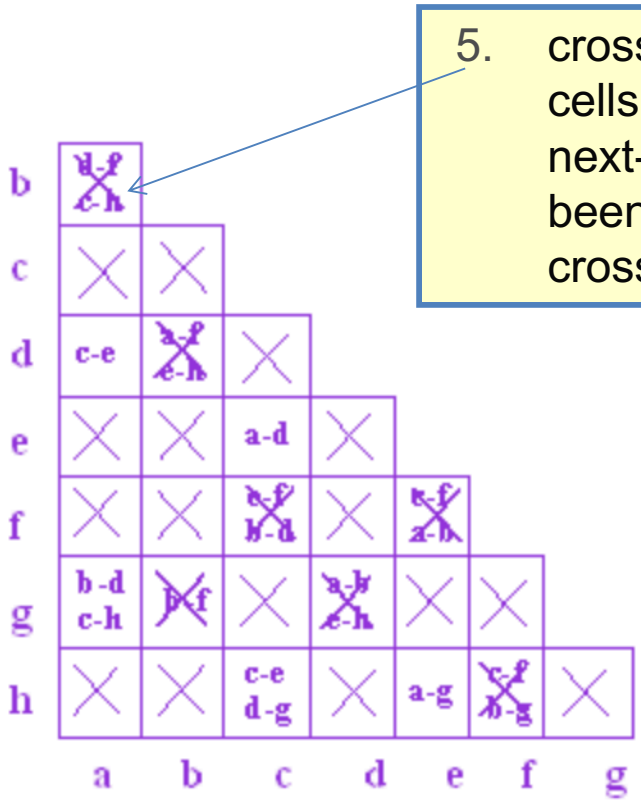
Current state	NxtState IN: 0	1	OUT
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



3. Cross out pairs of states with different outputs
4. In the rest of the cells put the pairs of next-states that need to be the same for S_i-S_j to be the same

Implication Table Method example 2

Current state	NxtState IN: 0	1	OUT
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

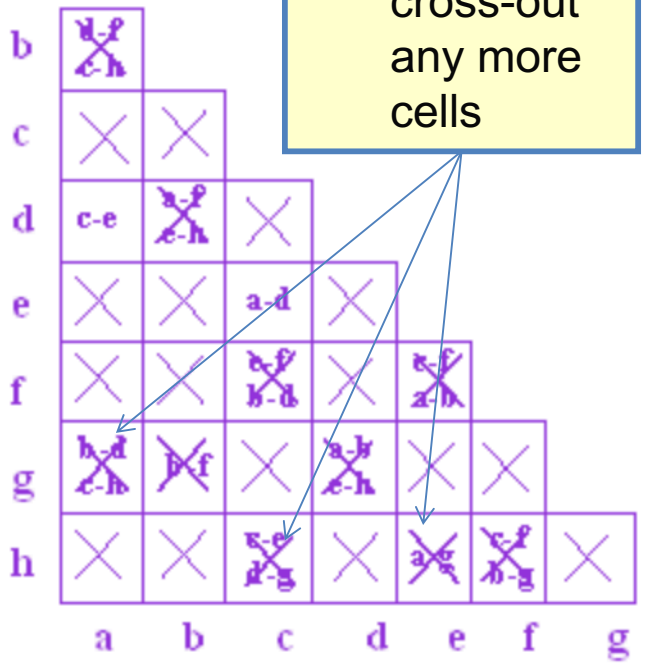


Implication Table Method example 2

Current state	NxtState IN: 0	1	OUT
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

7. The cells that are not crossed out indicate states that can be merged

6. Repeat last step until you cannot cross-out any more cells



Cu-state	Nstate IN: 0	1	OUT
a	a	c	0
b	f	h	0
c	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

a=d
C=e

Quiz 7-2

http://m.socrative.com/student/#joinRoom

room number: 713113

- Q1: How many states can be represented by N bits in:
  a. Binary encoding
  b. Gray-code encoding
  c. One-hot encoding
- Q2: One hot encoding requires fewer flipflops than binary and gray-code
  - True/false
- Q3: One hot encoding has usually simpler output logic than the binary and gray-code
  - True/false
- Q4: In the partitioning state minimization algorithm: A state needs to be removed from a block if:
  - Its next state (for a particular input) is different than the next states of the rest
  - Its next state (for a particular input) does not belong to the same group as the other state
  - Its next state (for a particular output) belongs to the same group as the other state

Summary of Lecture 8

- Finite State Machines
  - Moore
  - Mealy
- State assignment
- State minimization

- Book (complimentary to the slides):
  - Sections 14, 19.1 - 19.3

- Next Lecture 9:
  - VHDL for Finite State Machines

FSMs intro:

https://www.youtube.com/watch?v=vhiiia1_hC4