Lecture 8
Synchronous Sequential Circuits

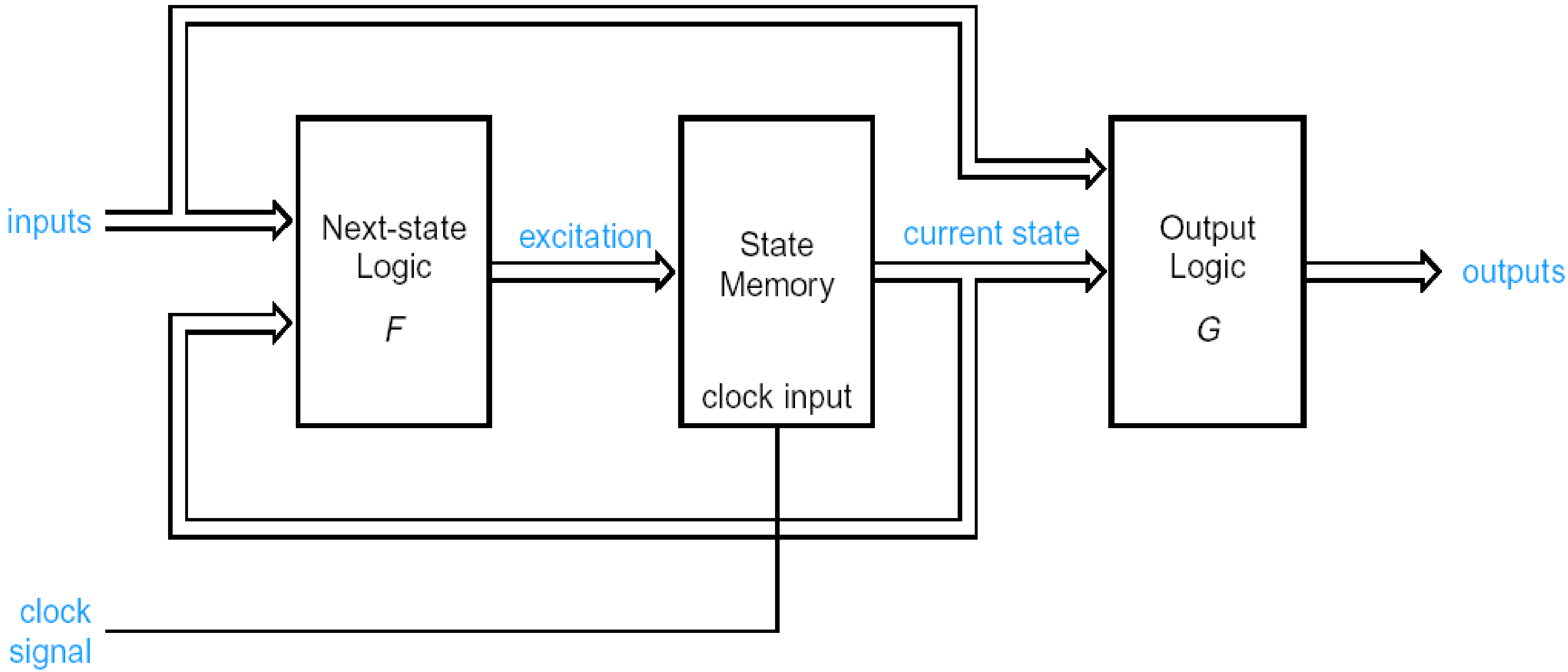
吳文中

Clocked Synchronous State-Machine Analysis

- "Clocked" refers to the fact that their storage elements (flip-flops) employ a clock input.
- "Synchronous" means that all of the flip-flops use the same clock signal.
- "State-machine" or "finite state-machine" is a generic name given to sequential circuits with flip-flops.
- Such a state machine changes state only when a triggering edge or "tick" occurs on the clock signal.

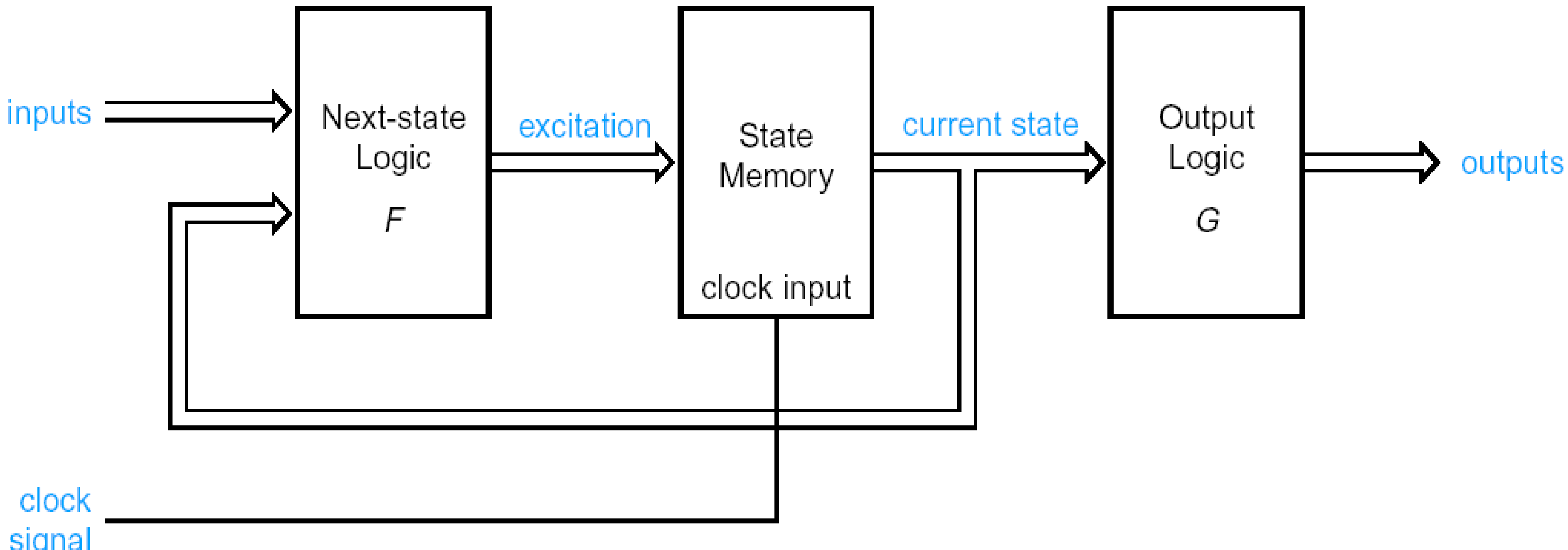
Mealy State-Machine

- The output depends on both state and input.
  - Next state = F (current state, input)
  - Output = G (current state, input)

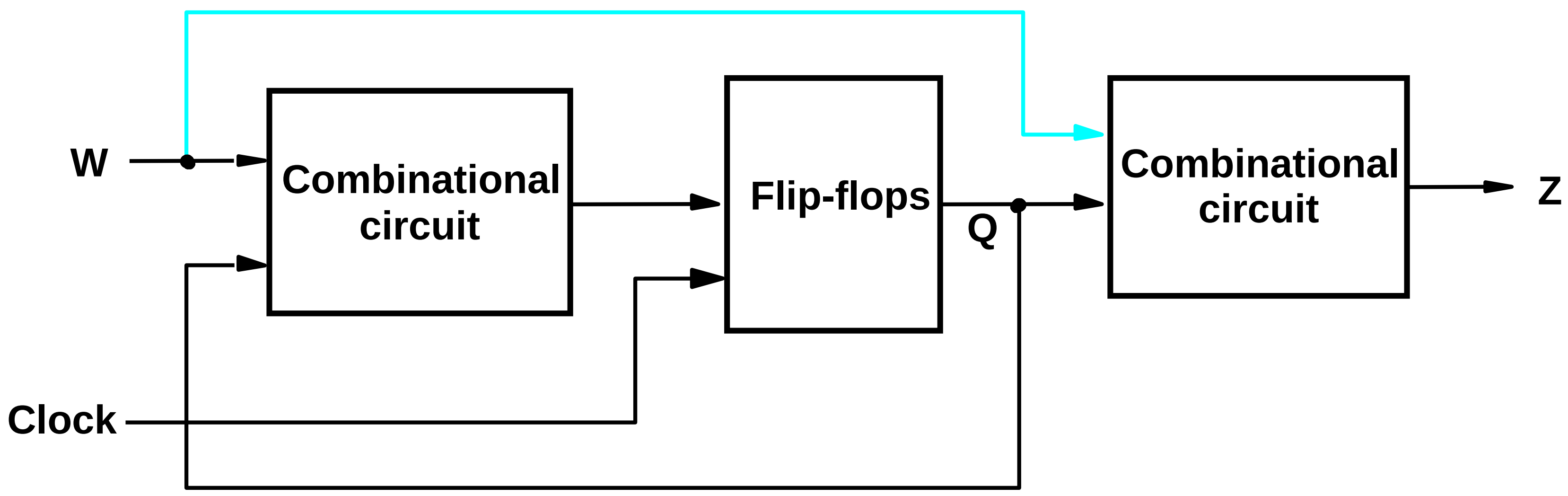


Moore State Machine

- Output only depends on the state alone.
  - Output = G (current state)
- Output-coded state machine: use state variables as output, no output logic G => no change during each



Finite State Machine

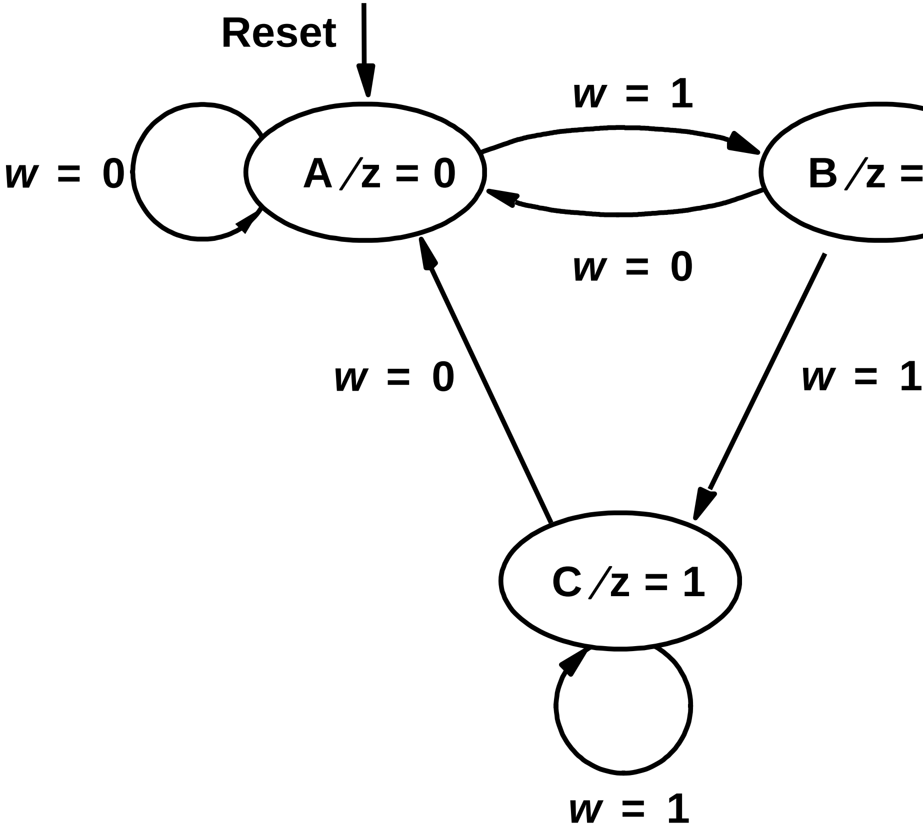


Basic Design Steps

- 1. The circuit has one input, w, and one output Z
- 2. All changes in the circuit occur on the positive edge of a clock signal
- 3. The output z is equal to 1 if during two immediately preceding clock cycles the input w was equal to 1. Otherwise the value of z is equal to 0.

State Diagram and State Table

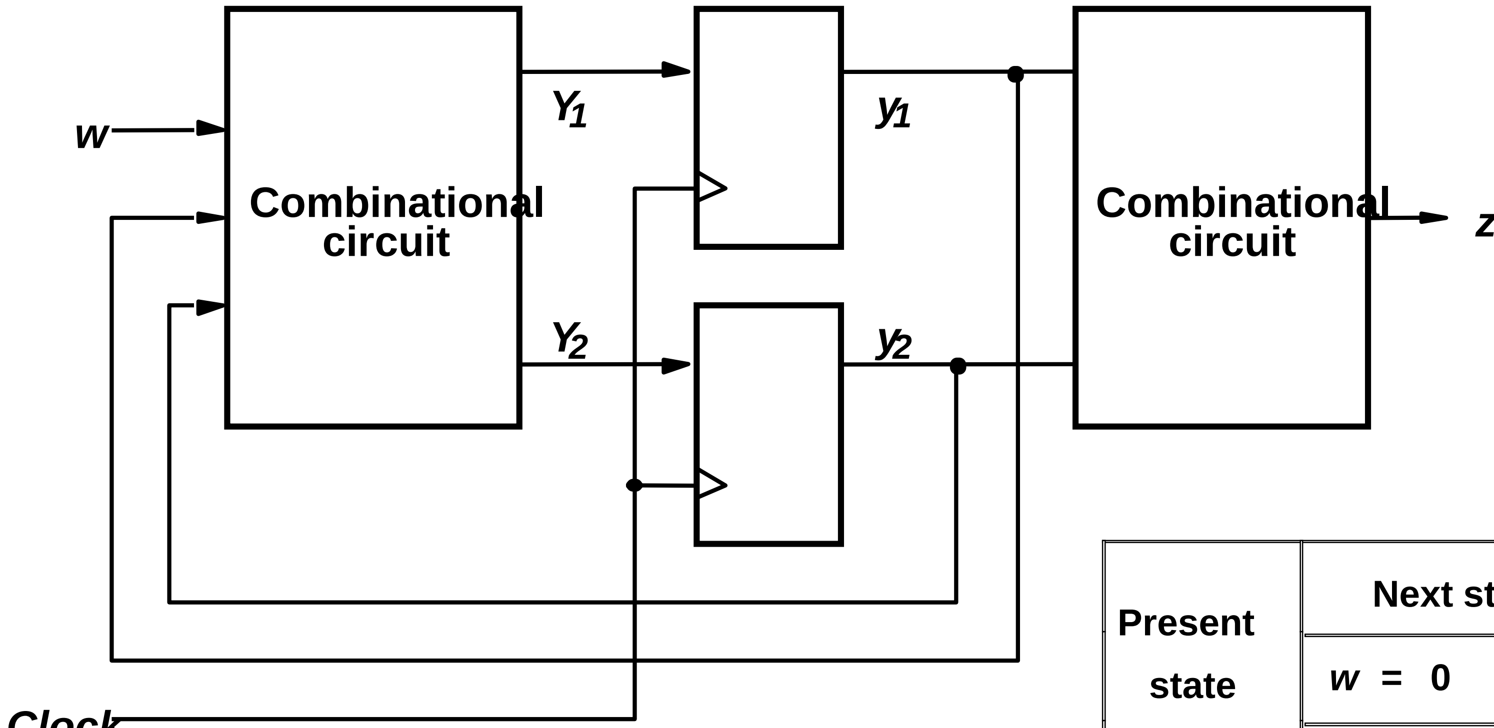
Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0



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

NTU ESOE

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

Choice of Flip-flips and Derivation of Next-state and Output Expressions

- Simplest choice: D flip-flops
- Derive logic eq. from state table

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

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

y_2 \ y_1	0	1
0	0	0
1	1	d

Ignoring don't cares

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

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

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

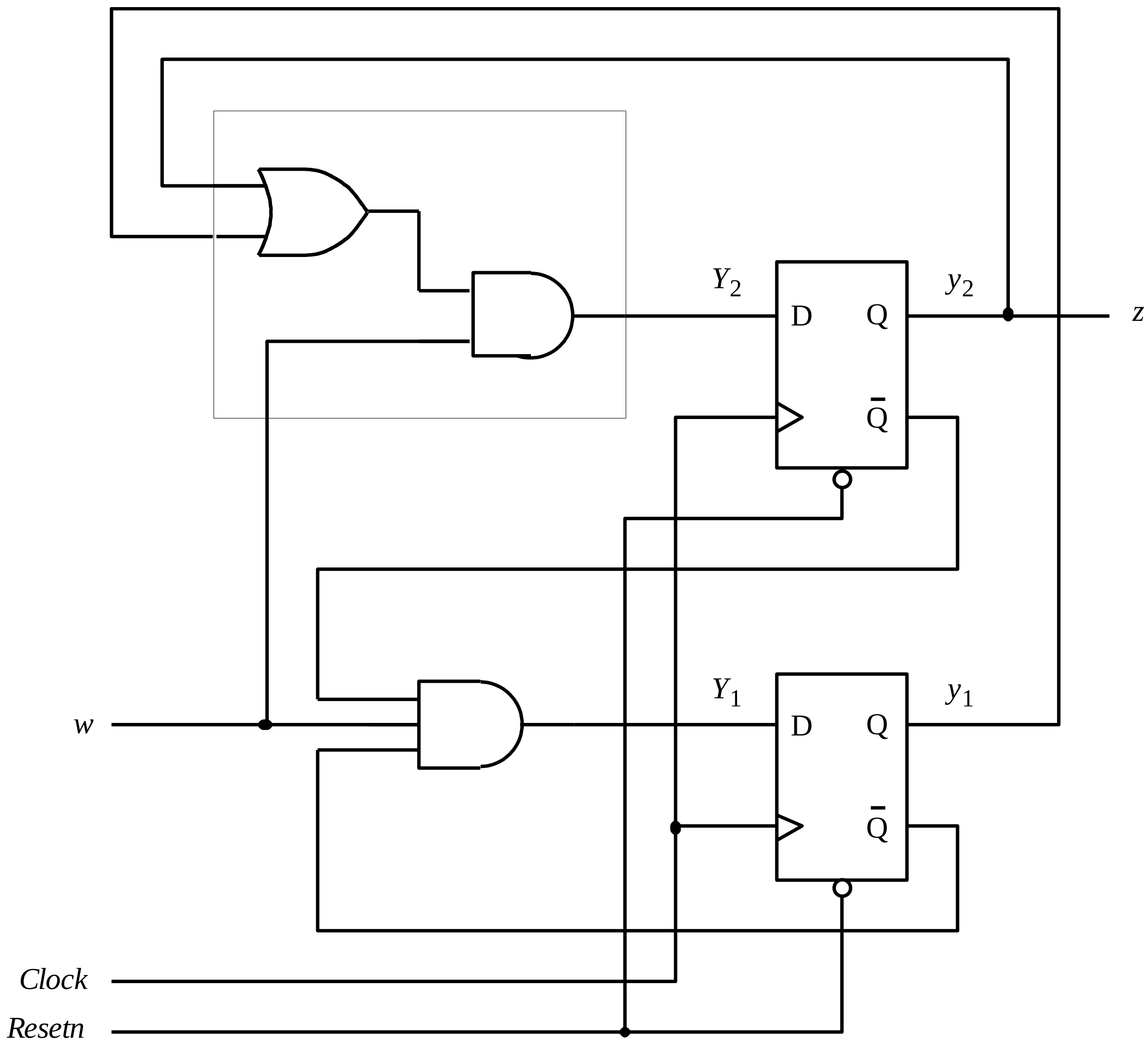
Using don't cares

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

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

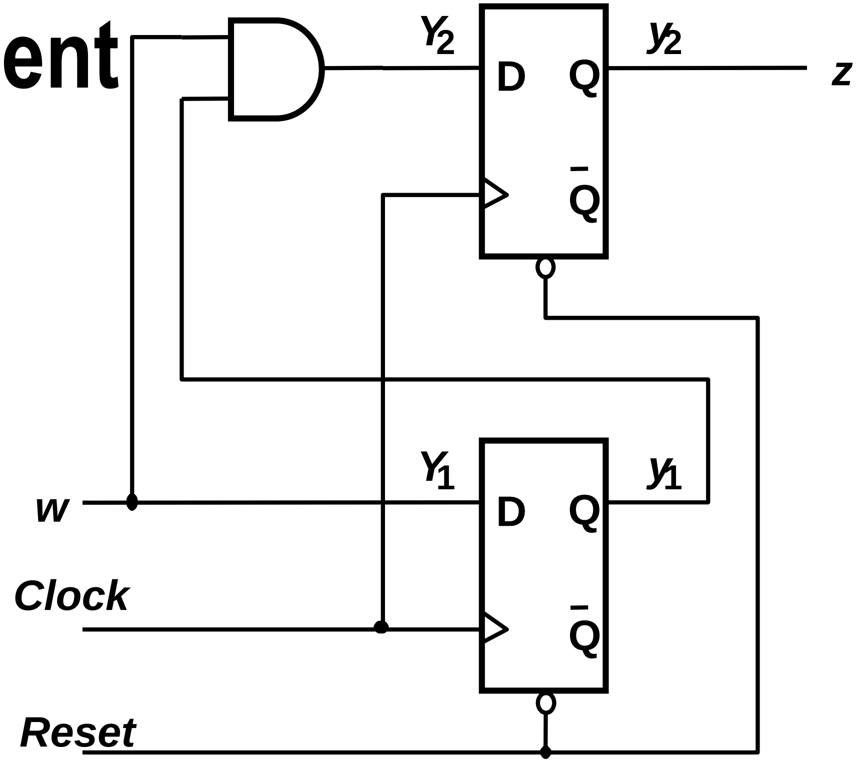
$$z = y_2$$

Final Implementation



Improved State Assignment

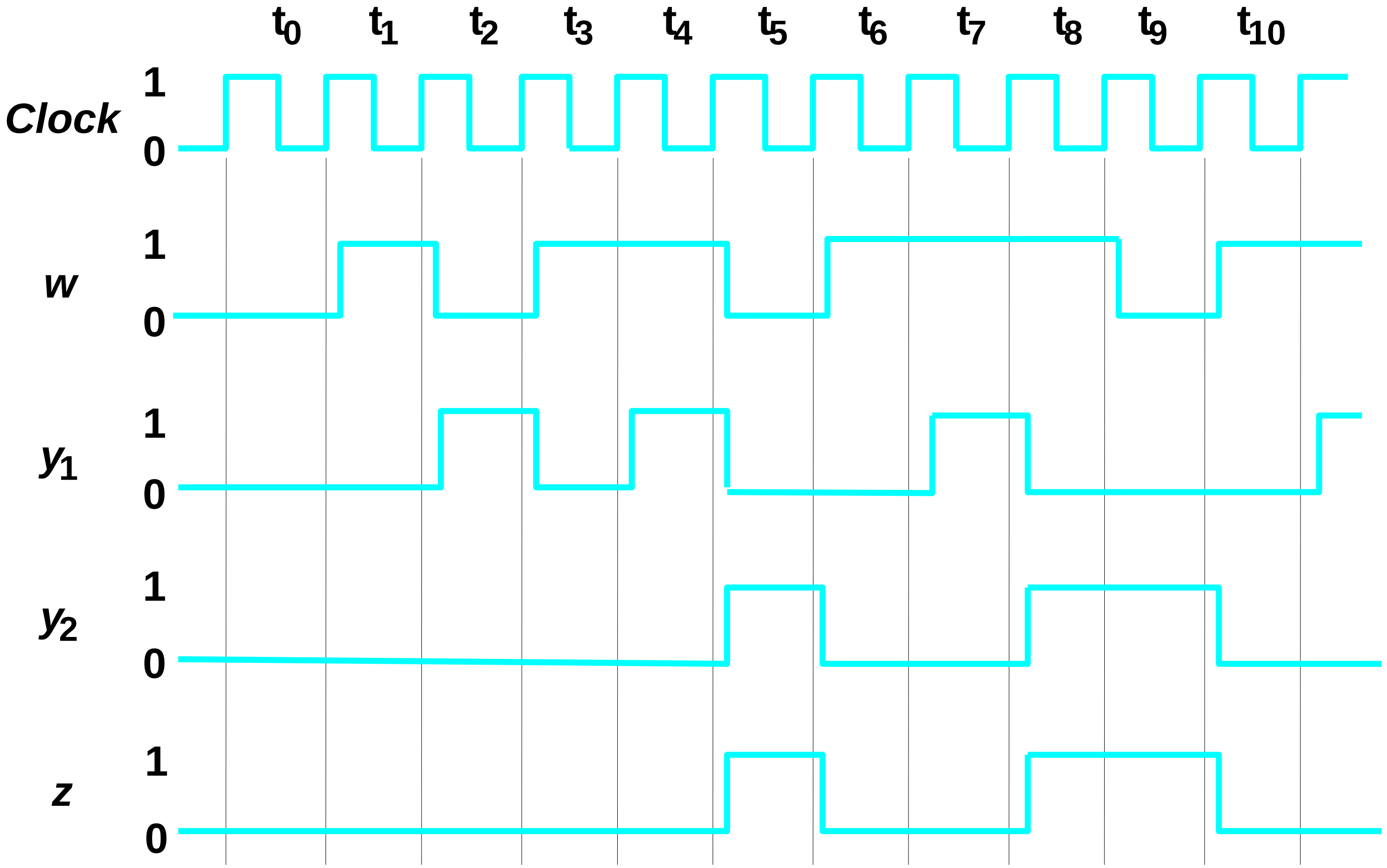
Present state	Next state $w = 0$	Next state $w = 1$	Output z
A	A	B	0
B	A	C	0
C	A	C	1



	Present state 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*

- $Y_1 = D_1 = w$
- $Y_2 = D_2 = wy_1$
- $z = y_2$

Timing Diagram



NTU ESOE

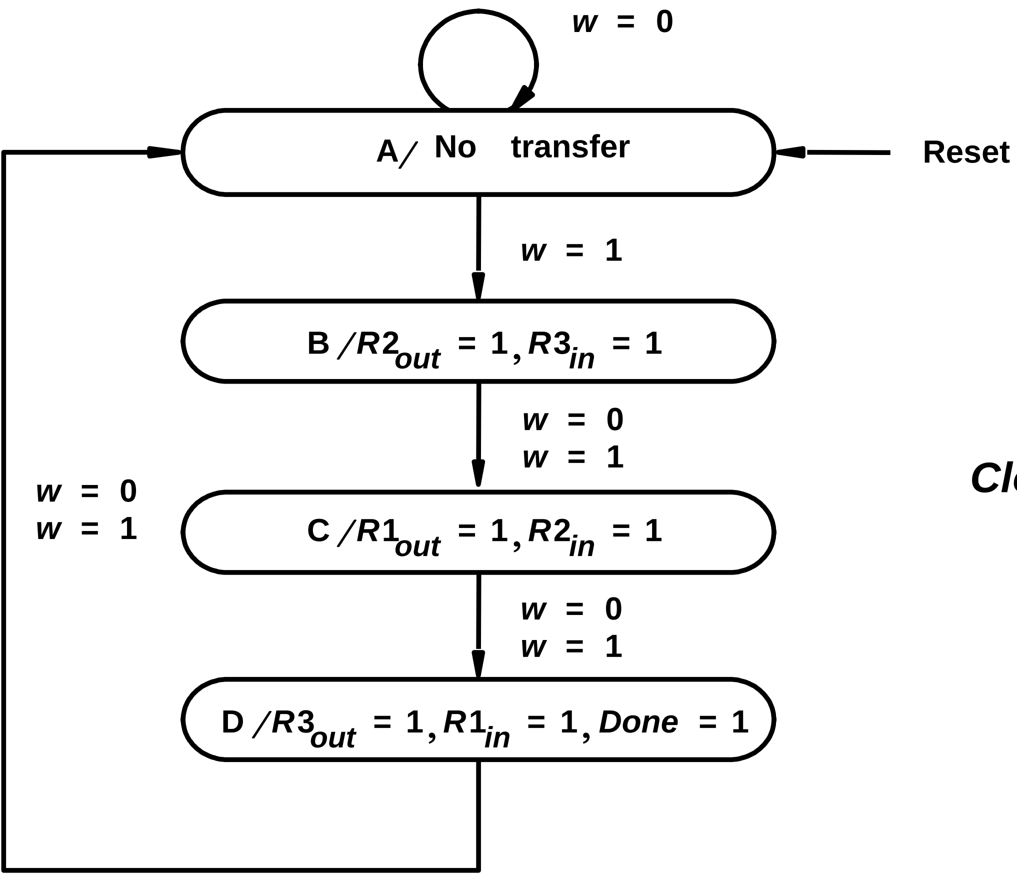
Summary of Design Steps

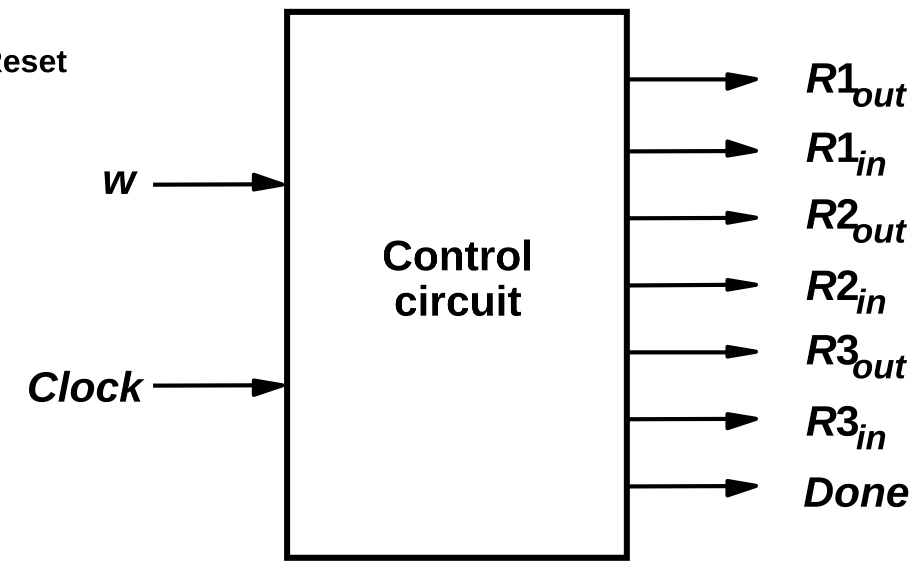
- 1. Obtain the specification of the desired circuit
- 2. Derive the states for the machine by first selecting a starting state. Then, given the specification of the circuit, create news states as needed for the machine to respond to all inputs. Create a state diagram accordingly (optional).
- 3. Create a state table.
- 4. State minimization
- 5. Decide the state variables needed and do state assignment

- 6. Choose the type of flip-flops to be used in the circuit. Derive the next-state logic expressions to control the inputs to all flip-flops and then derive logic expressions for the output of the circuit.
- 7. Implement the circuit as indicated by the logic expressions.

Ex 8.1 Bus control

- Swap R1 and R2 using R3 as a temporary storage locations



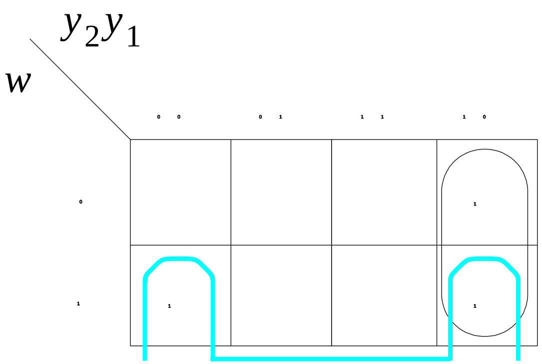


Ex 8.1 State table and state assignments

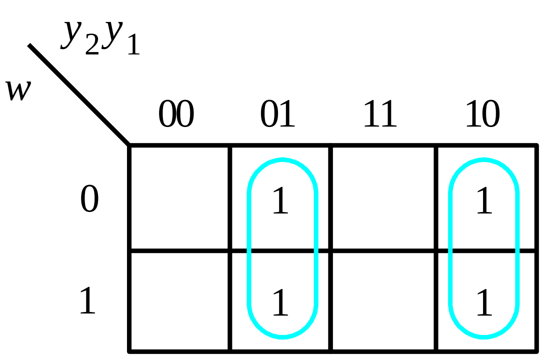
Present state	Next state		Outputs						
	$w = 0$	$w = 1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	A	B	0	0	0	0	0	0	0
B	C	C	0	0	1	0	0	1	0
C	D	D	1	0	0	1	0	0	0
D	A	A	0	1	0	0	1	0	1

	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

Ex 8.1 Logic expression of Next-state and output Logic



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

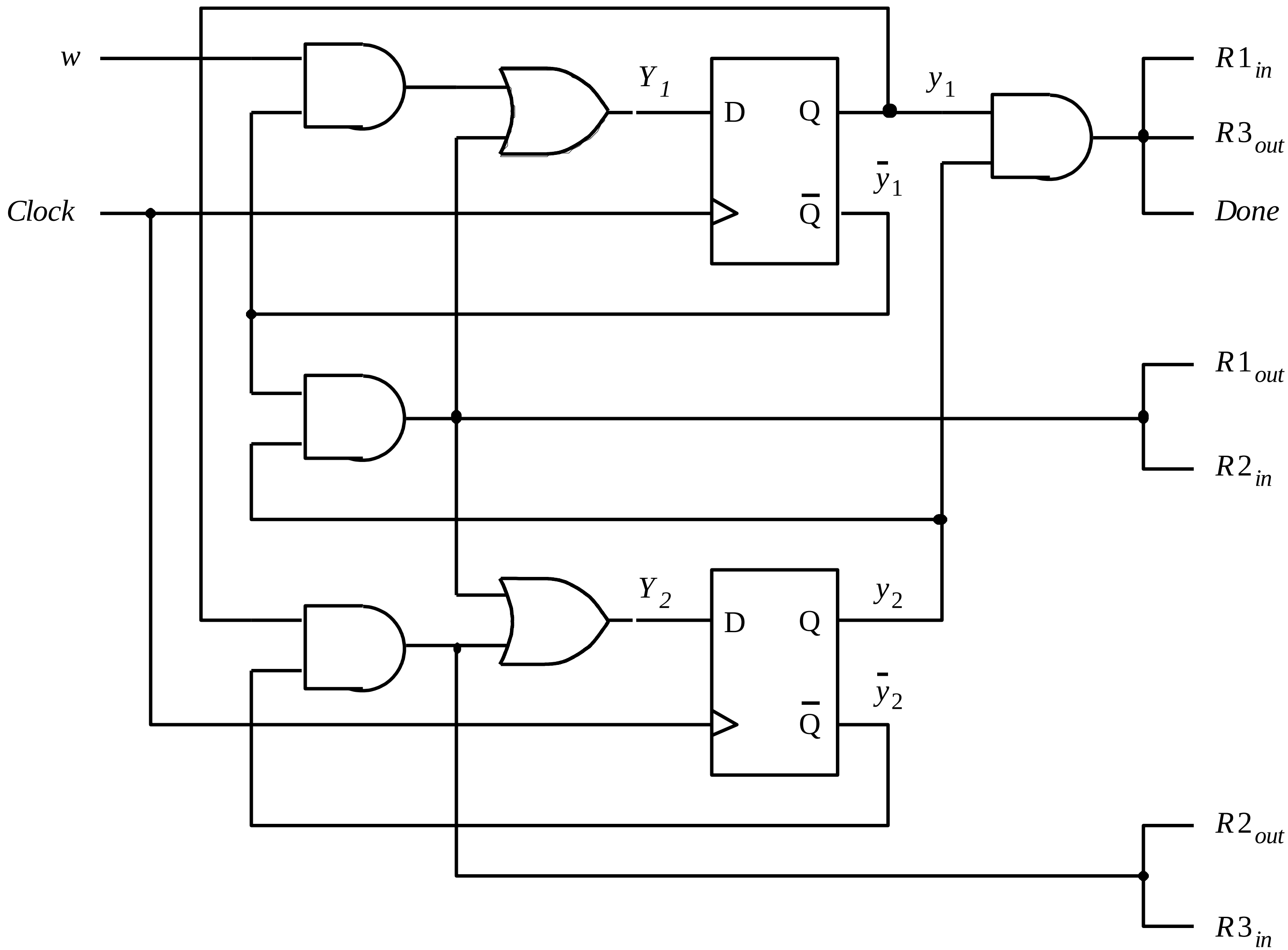


$Y_2 = y_1\bar{y}_2 + \bar{y}_1 y_2$

- Output control signals are
- $R1_{out} = R2_{in} = \overline{y_1}y_2$
- $R1_{in} = R3_{out} = Done = y_1y_2$
- $R2_{out} = R3_{in} = y_1\overline{y_2}$

NTU ESOE

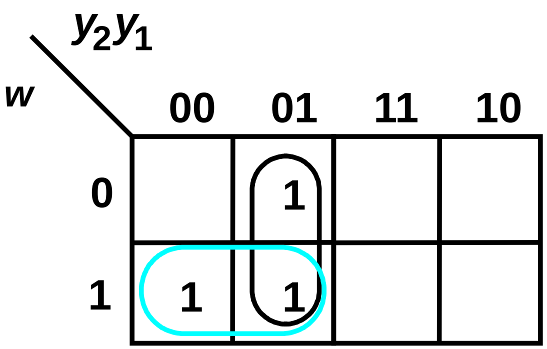
Ex 8.1 Final Implementation



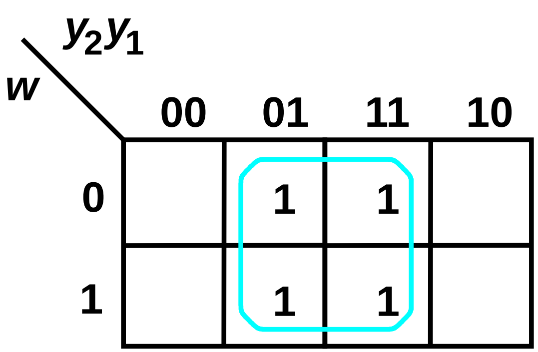
Improved State Assignment

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



$Y_1 = w\bar{y}_2 + y_1\bar{y}_2$



$Y_2 = y_1$

- $R1_{out} = R2_{in} = y_1 y_2$
- $R1_{in} = R3_{out} = Done = \overline{y_1} y_2$
- $R2_{out} = R3_{in} = y_1 \overline{y_2}$

One-Hot Encoding

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

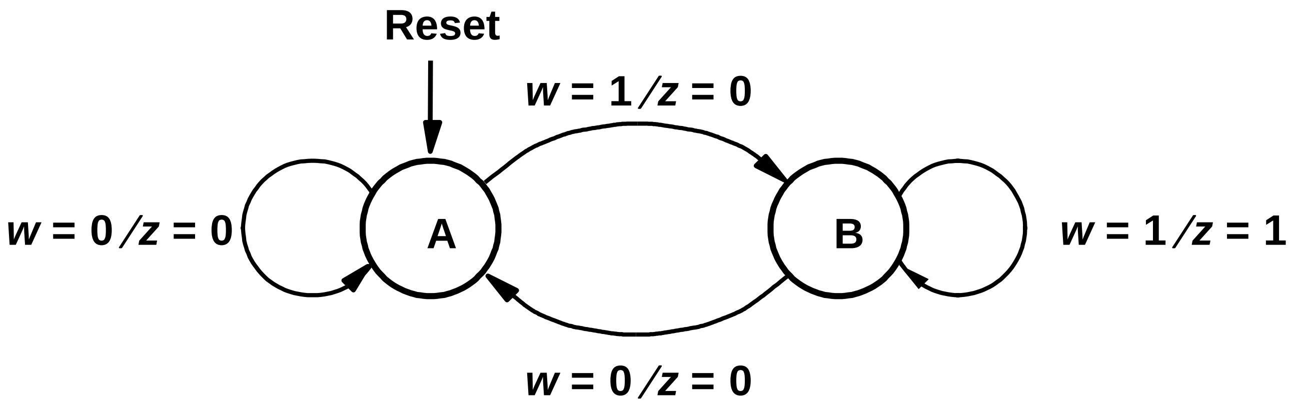
	Present state	Nextstate		Outputs						
		$w = 0$	$w = 1$							
	$y_4y_3y_2y_1$	$Y_4Y_3Y_2Y_1$	$Y_4Y_3Y_2Y_1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	0 001	0001	0010	0	0	0	0	0	0	0
B	0 010	0100	0100	0	0	1	0	0	1	0
C	0 100	1000	1000	1	0	0	1	0	0	0
D	1 000	0001	0001	0	1	0	0	1	0	1

- $Y_1 = \bar{w}y_1 + y_4$
- $Y_2 = wy_1$
- $Y_3 = y_2$
- $Y_4 = y_3$
- $R1_{out} = R2_{in} = y_{\mathbf{3}}$
- $R1_{in} = R3_{out} = Done = \overline{y_1}y_2$
- $R2_{out} = R3_{in} = y_1\overline{y_2}$

Mealy State Model

- Specification: z=1 for two '1's and switch in the same cycle.

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



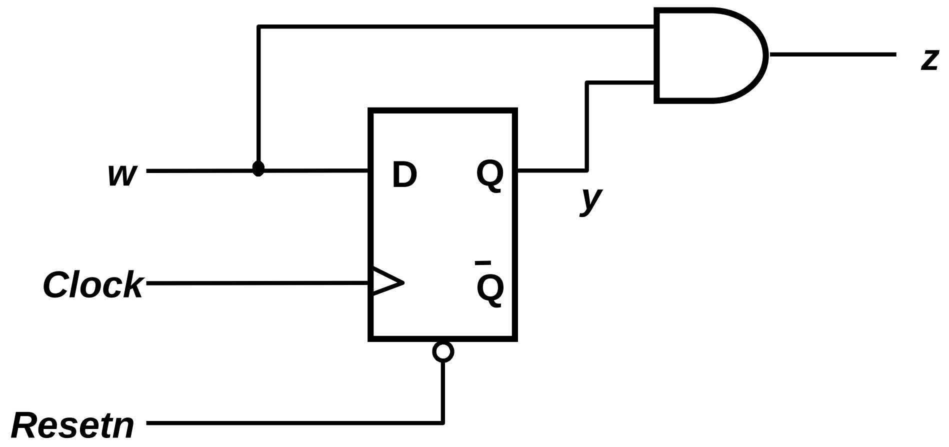
State Table and State Assignment

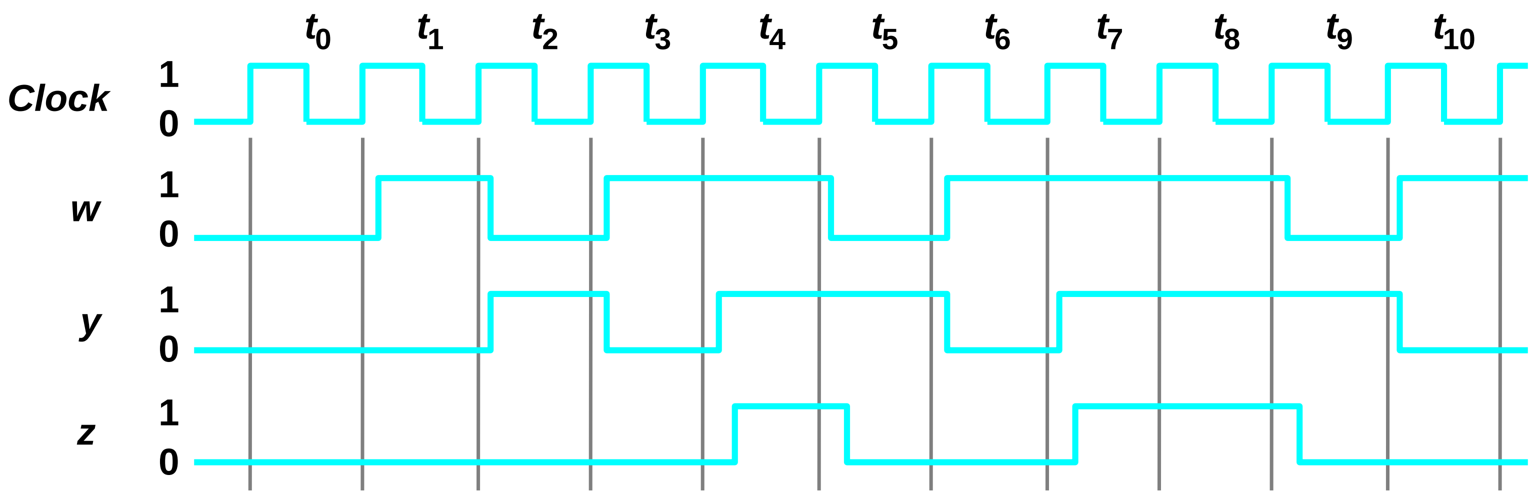
Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

	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

- $Y=D=w$
- $z=wy$

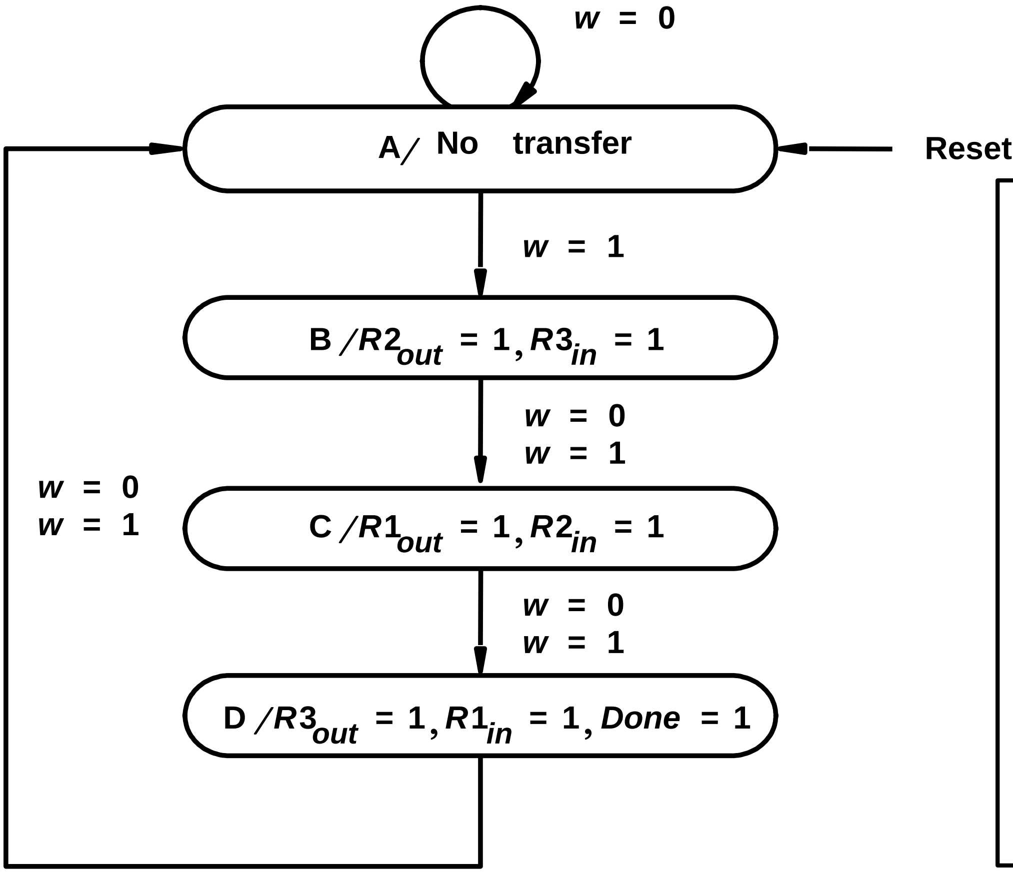
Circuit and Timing Diagram



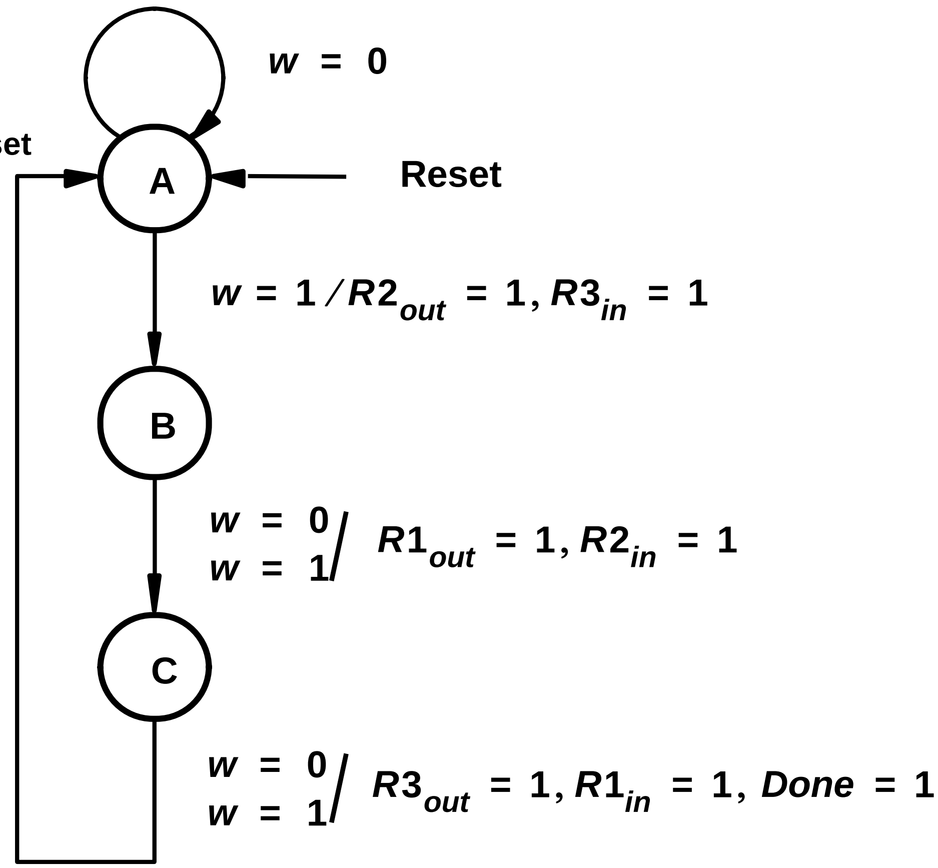


Ex. 8.4

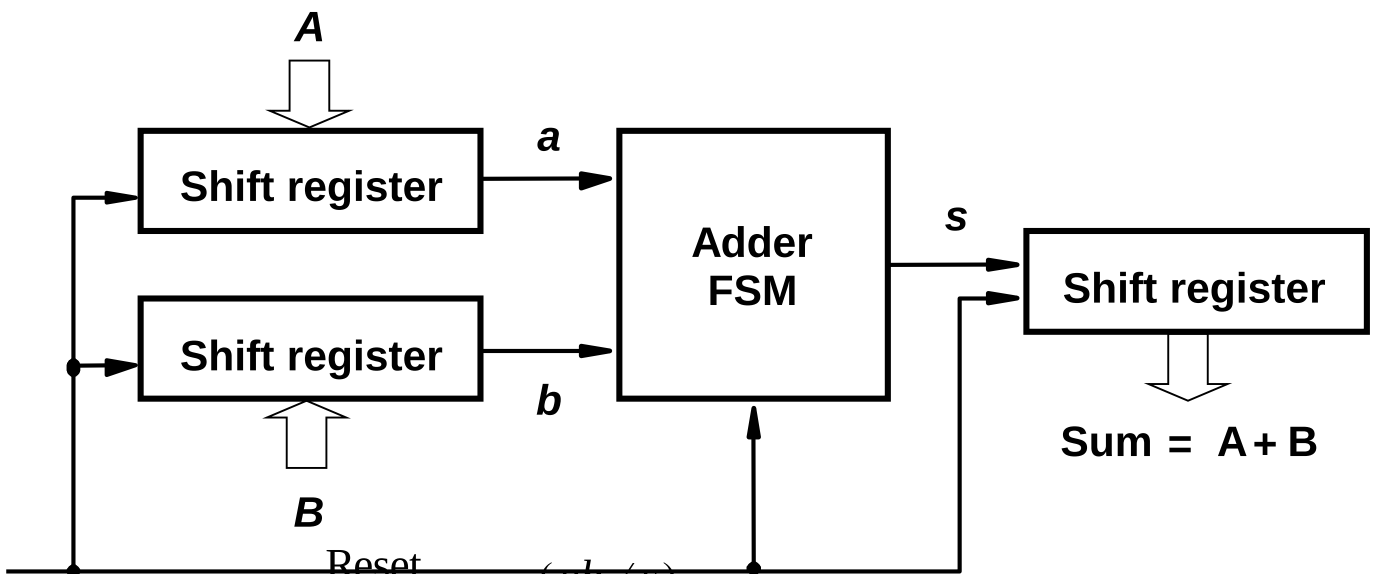
- Moore type

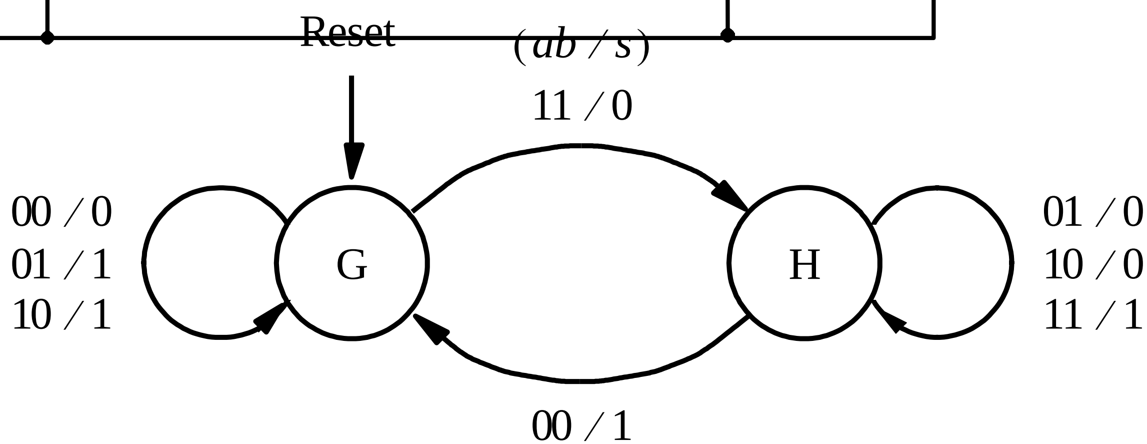


Mealy type



Mealy-Type FSM for Serial Adder





G: carry-in = 0

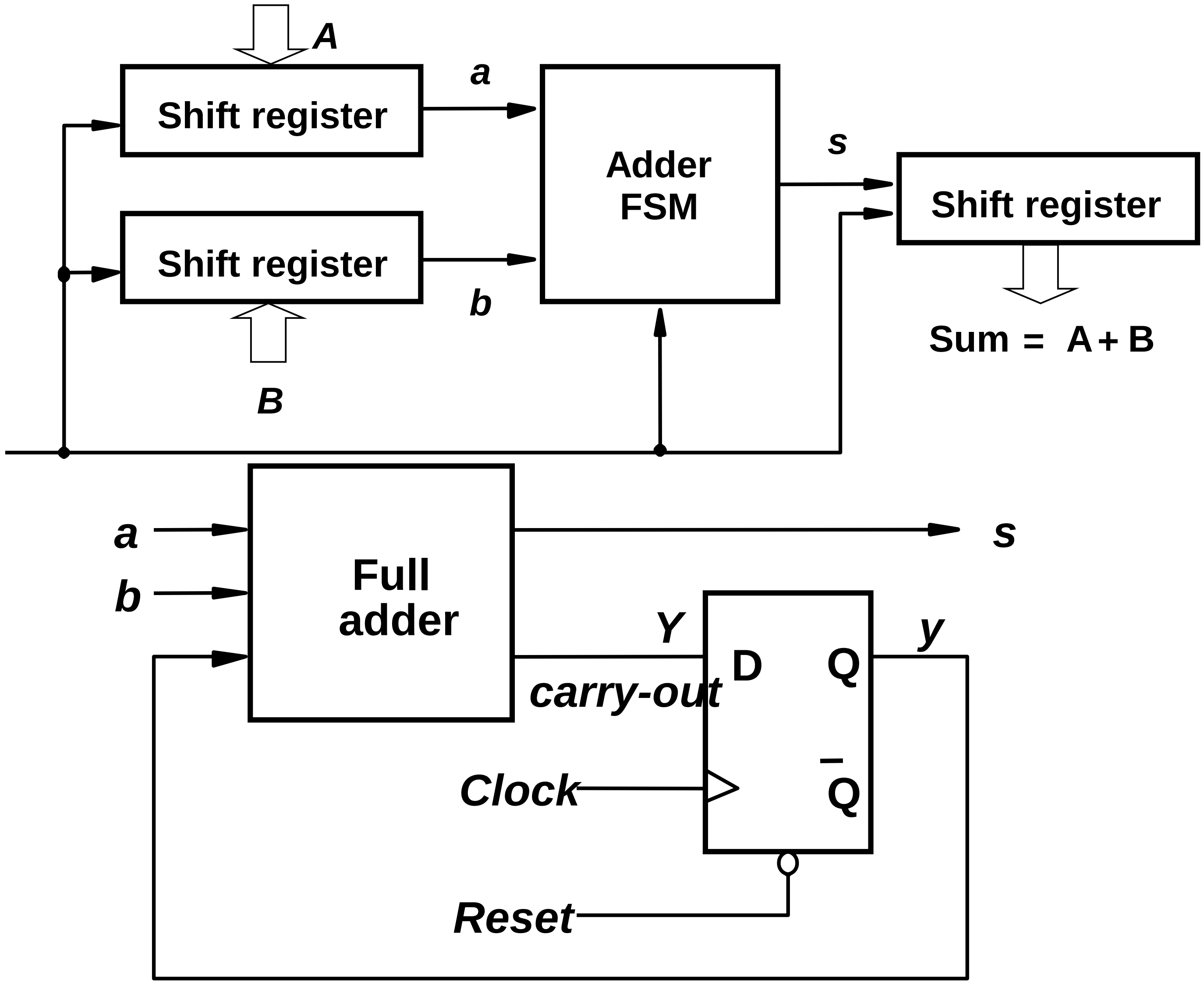
H: carry-in = 1

Present state	Next state				Outputs			
	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

Present state	Next state				Output			
	ab=00	01	10	11	00	01	10	11
y	*Y*				*s*			
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

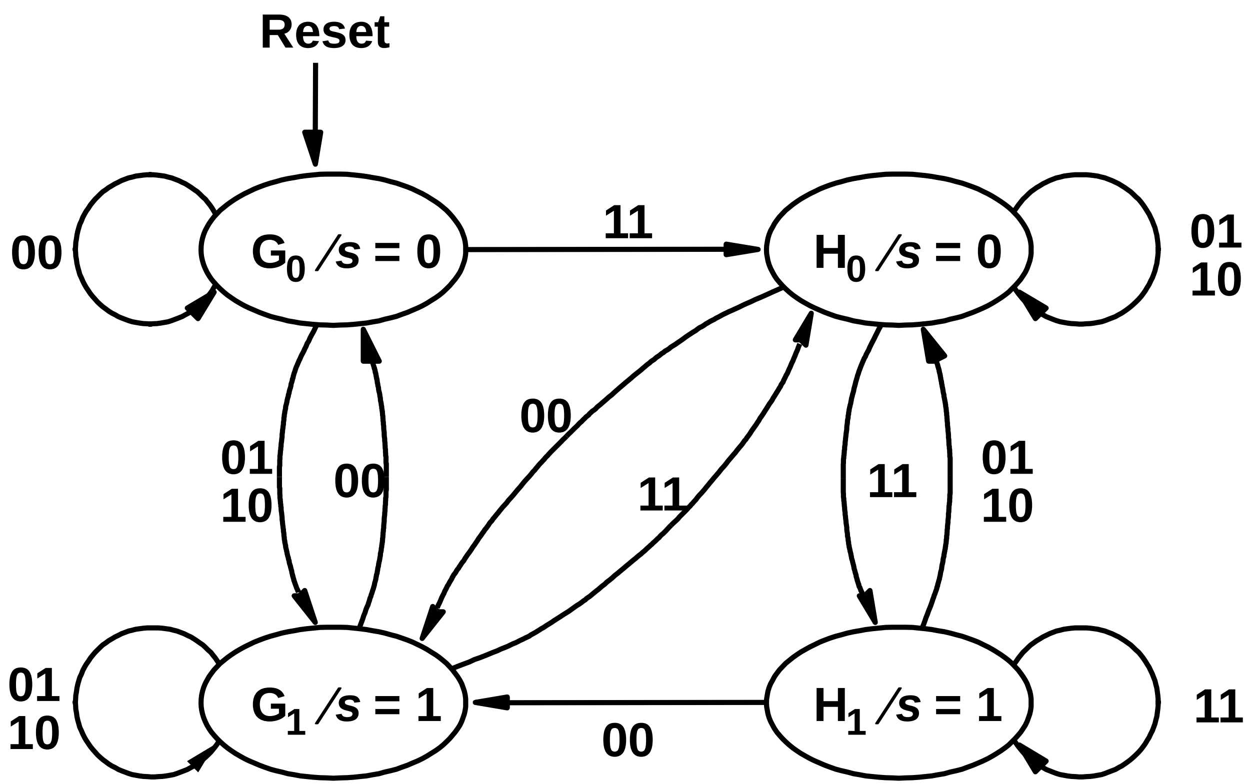
- $Y = ab + ay + by$
- $s = a \oplus b \oplus y$

Mealy-type Serial Adder Circuit



NTU ESOE

Moore-type FSM for Serial Adder

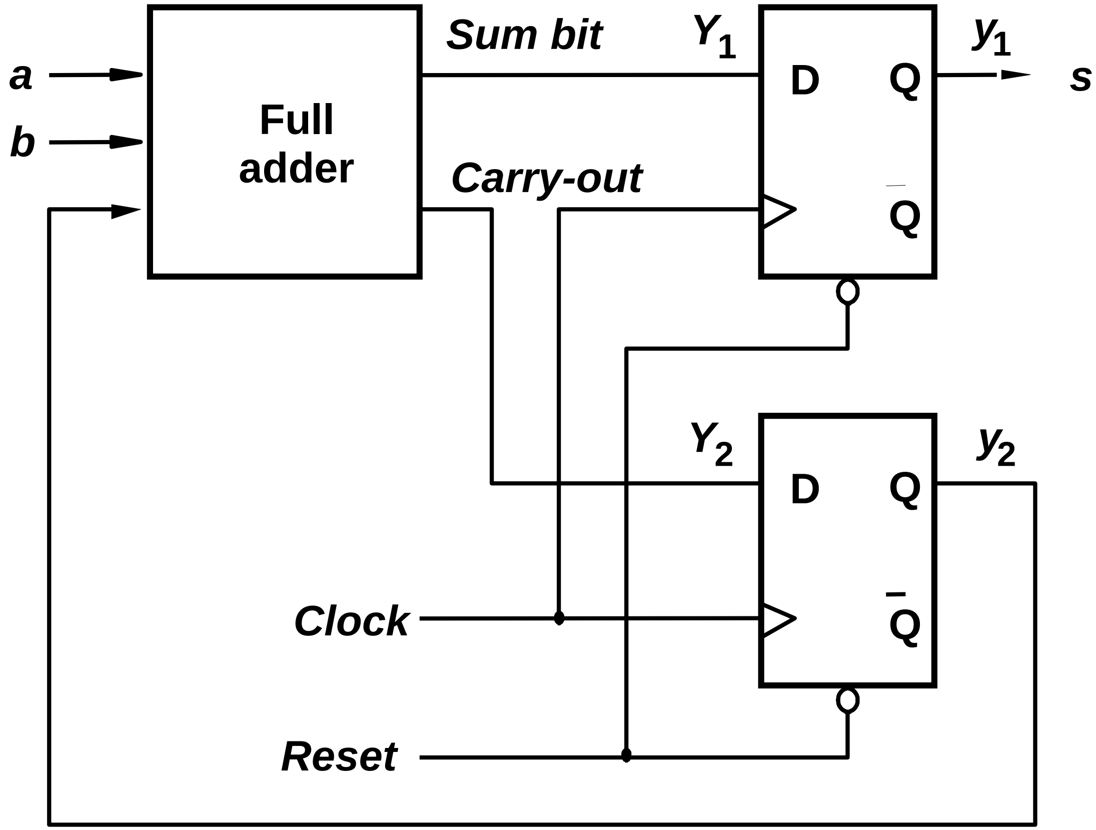


Present state	Nextstate				Output
	ab=00	01	10	11	*s*
G_0	G_0	G_1	G_1	H_0	0
G_1	G_0	G_1	G_1	H_0	1
H_0	G_1	H_0	H_0	H_1	0
H_1	G_1	H_0	H_0	H_1	1

Present state y_2y_1	Nextstate				Output *s*
	ab=00	01	10	11	
	Y_2Y_1				
00	0 0	01	0 1	10	0
01	0 0	01	0 1	10	1
10	0 1	10	1 0	11	0
11	0 1	10	1 0	11	1

- $Y_1 = a \oplus b \oplus y_2$
- $Y_2 = ab + ay_2 + bY_2$
- $s = y_1$

Moore-type Serial Adder Circuit



State Minimization

- Two state 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 weather S_i and S_j is the initial state
- If a input combination *k* is applied in the state *Si* and it causes the machine to move to state S_v, S_v is defined as a *k-successor* of S_i.
- A partition consist of one or more blocks, where each block comprises a subset of states that maybe equivalent

Ex 8.6

- Initial partition
  $P_1 = (ABCDEFG)$
- Next partition with diff. outputs
  P_2 = (ABD)(CEFG)
- Check 0-sucessors and 1-succesors
  P_3 = (ABD)(CEG)(F)
- Again
  P_4 = (AD)(B)(CEG)(F)
  P_5 = (AD)(B)(CEG)(F)

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

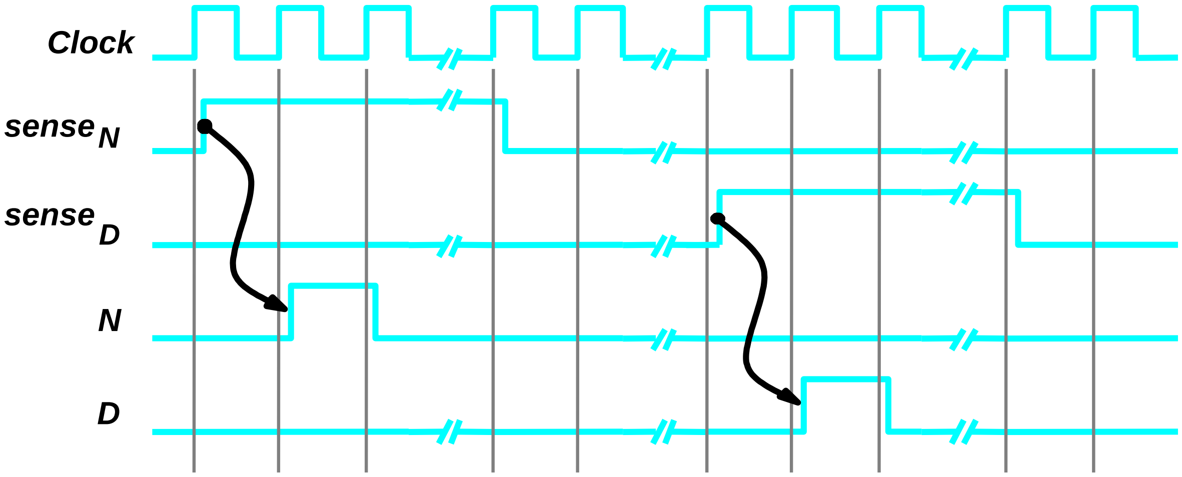
Present state	Nextstate w = 0	Nextstate w = 1	Output z
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

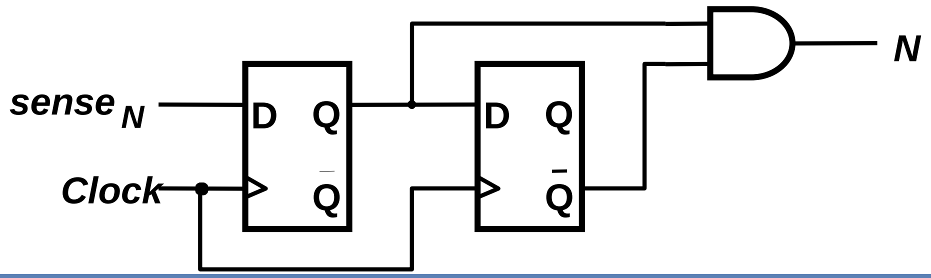
Ex8.7 Vending Machine

- The machine accepts nickels (5cent) and dimes (10cents)
- It takes 15 cents for a piece of candy to be releases.
- If 20 cents is deposited, the machine will not return the change, but it will credit the buyer with 5 cents and wait for the buyer to make a second purchase

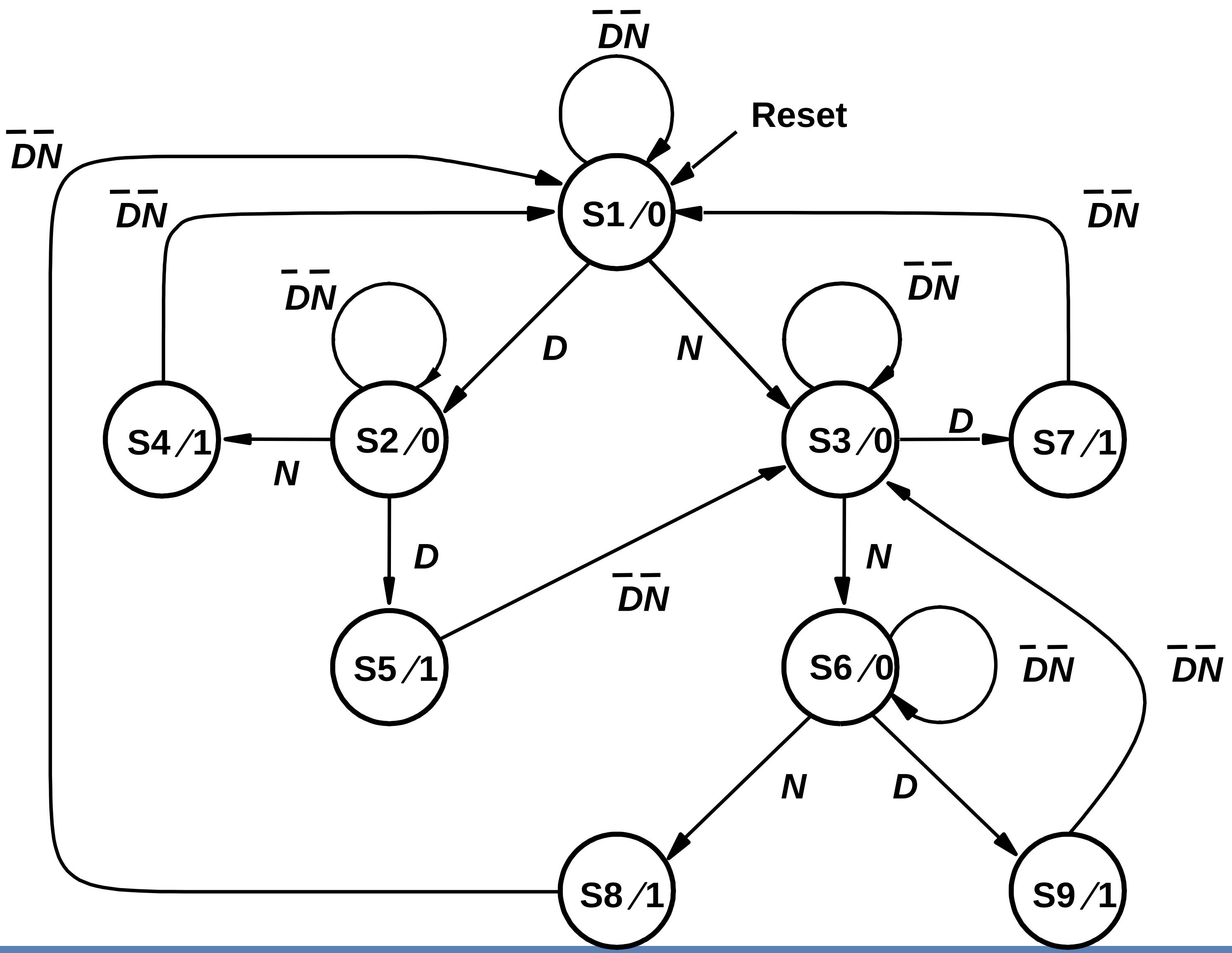
Denounced Circuit

- Mechanical sensor outputs $sense_N$ and $sense_D$ are slow and may stay on for several ticks.





State Diagram



State table and minimization

- P_1=(S1,S2,S3,S4,S5, S6,S7,S8,S9)
- P_2=(S1,S2,S3,S6)(S4 ,S5, S7,S8,S9)
- P_3=(S1)(S3)(S2,S6)(S4,S5, S7,S8,S9)
- P_4=(S1)(S3)(S2,S6)(S4, S7,S8)(S5,S9)
- P_5=(S1)(S3)(S2,S6)(S4, S7,S8)(S5,S9)

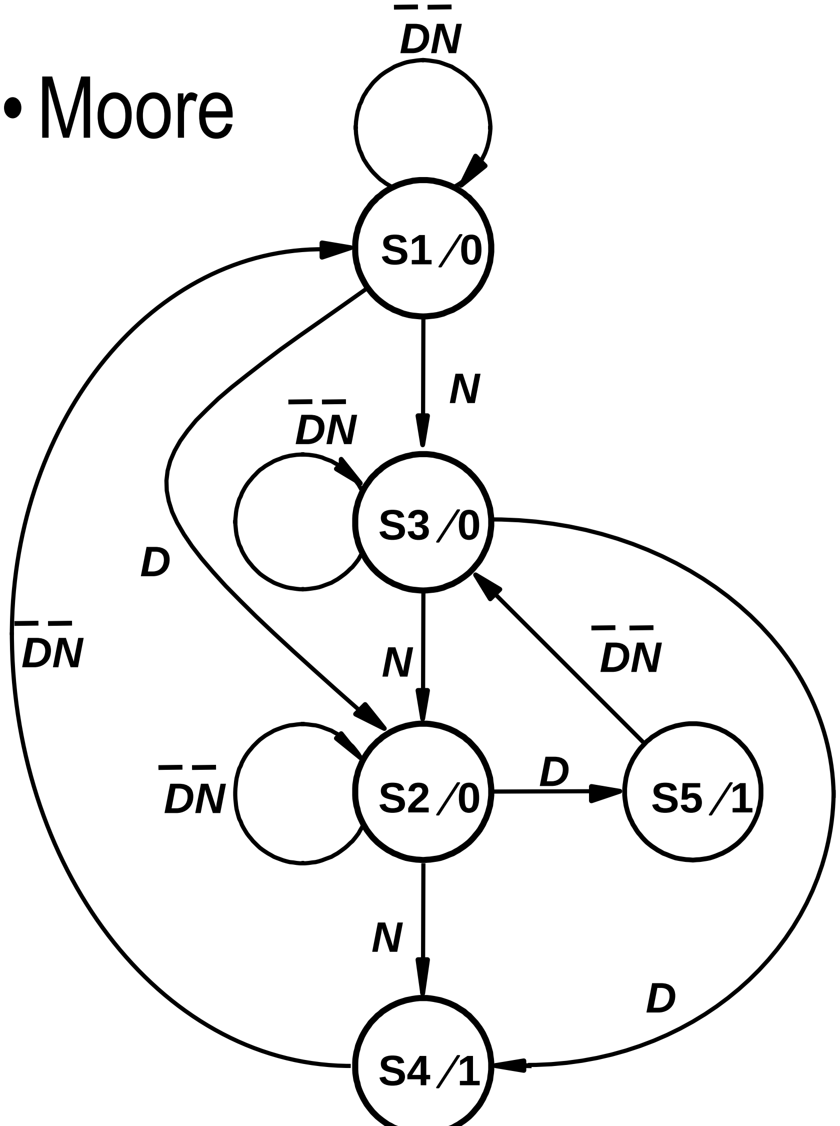
Present state	Next state				Output z
	DN =00	01	10	11	
S1	S1	S3	S2	–	0
S2	S2	S4	S5	–	0
S3	S3	S6	S7	–	0
S4	S1	–	–	–	1
S5	S3	–	–	–	1
S6	S6	S8	S9	–	0
S7	S1	–	–	–	1
S8	S1	–	–	–	1
S9	S3	–	–	–	1

Minimized State Table

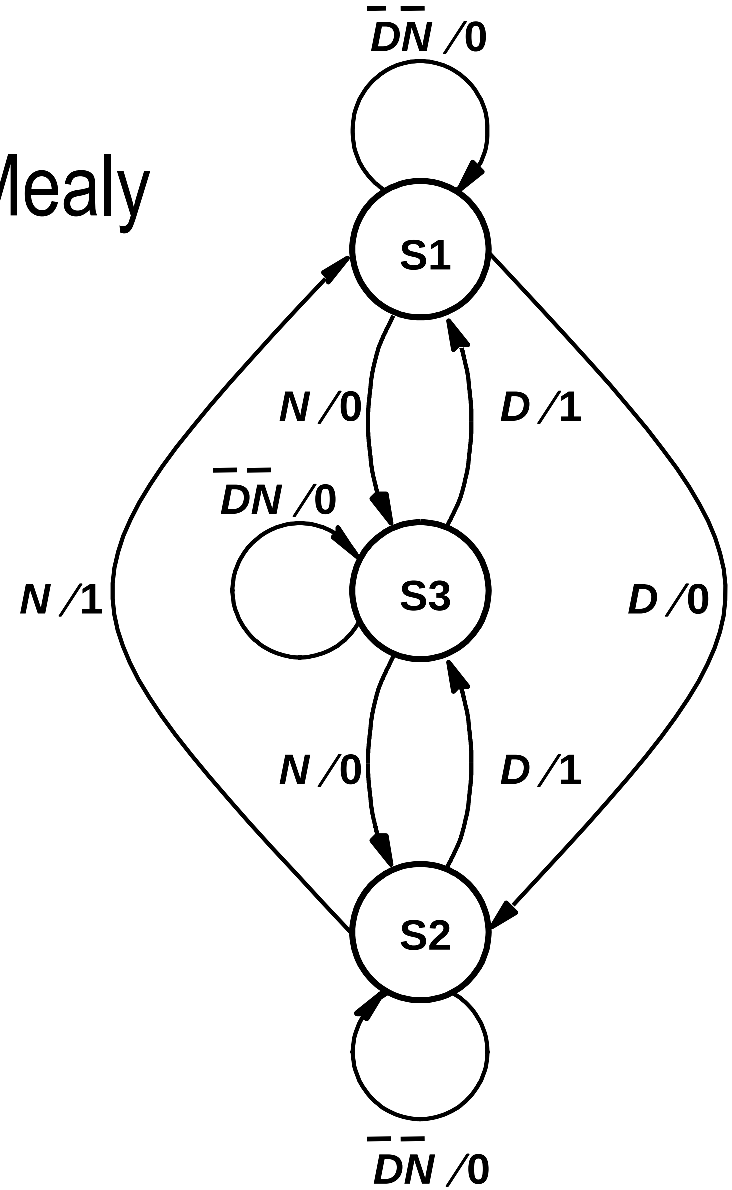
Present state	Next state				Output z
	DN =00	01	10	11	
S1	S1	S3	S2	–	0
S2	S2	S4	S5	–	0
S3	S3	S2	S4	–	0
S4	S1	–	–	–	1
S5	S3	–	–	–	1

State Diagram

- Moore



Mealy

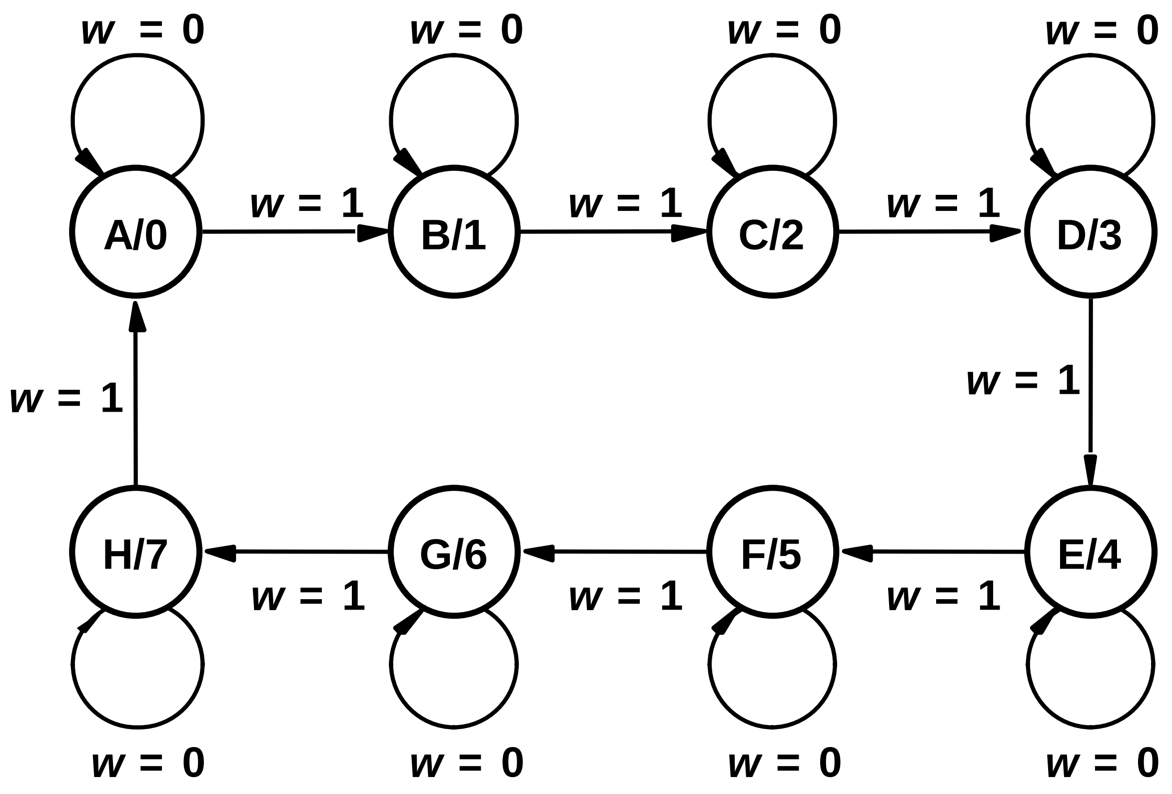


Ex. 8.8 Incompletely Specified State Table

- P_1=(ABCDEFG)
- Assume -=0
  P_2=(ABDG)(CEF)
- Check k-sucessors
  P_3=(AB)(D)(G)(CE)(F)
  P_4=(A)(B)(D)(G)(CE)(F)
  P_5=P_4
- Assume -=1
  P_2=(AD)(BCEFG)
  P_3=(AD)(B)(CEFG) P_4=(AD)(B)(CEG)(F) P_5=P_4i

Present state	Next state		Outputz	
	w = 0	w = 1	w = 0	w = 1
A	B	C	0	0
B	D	–	0	–
C	F	E	0	1
D	B	G	0	0
E	F	C	0	1
F	E	D	0	1
G	F	–	0	–

State Diagram for A Modulo-8 Counter



NTU ESOE

State Table for A Modulo-8 Counter

Present state	Next state $w = 0$	Next state $w = 1$	Output
A	A	B	0
B	B	C	1
C	C	D	2
D	D	E	3
E	E	F	4
F	F	G	5
G	G	H	6
H	H	A	7

	Present state $y_2y_1y_0$	Next state $w = 0$ $Y_2Y_1Y_0$	Next state $w = 1$ $Y_2Y_1Y_0$	Count $z_2z_1z_0$
A	000	000	001	000
B	001	001	010	001
C	010	010	011	010
D	011	011	100	011
E	100	100	101	100
F	101	101	110	101
G	110	110	111	110
H	111	111	000	111

Implementation Using D-type Flip-Flops

- $D_0 = Y_0$
  $= \bar{w}y_0 + w\bar{y}_0$
  $= w \oplus y_0$
- $D_1 = Y_1$
  $= \bar{w}y_1 + y_1\overline{y_0} + wy_0\bar{y}_1$
  $= wy_0 \oplus y_1$
- $D_2 = Y_2$
  $= \bar{w}y_2 + \bar{y}_0 y_2 + \overline{y_1} y_2$
  $+ wy_0y_1\bar{y}_2$
  $= wy_0y_1 \oplus y_2$

$wy_2 \backslash y_1y_0$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	1

$Y_0 = \bar{w}y_0 + w\bar{y}_0$

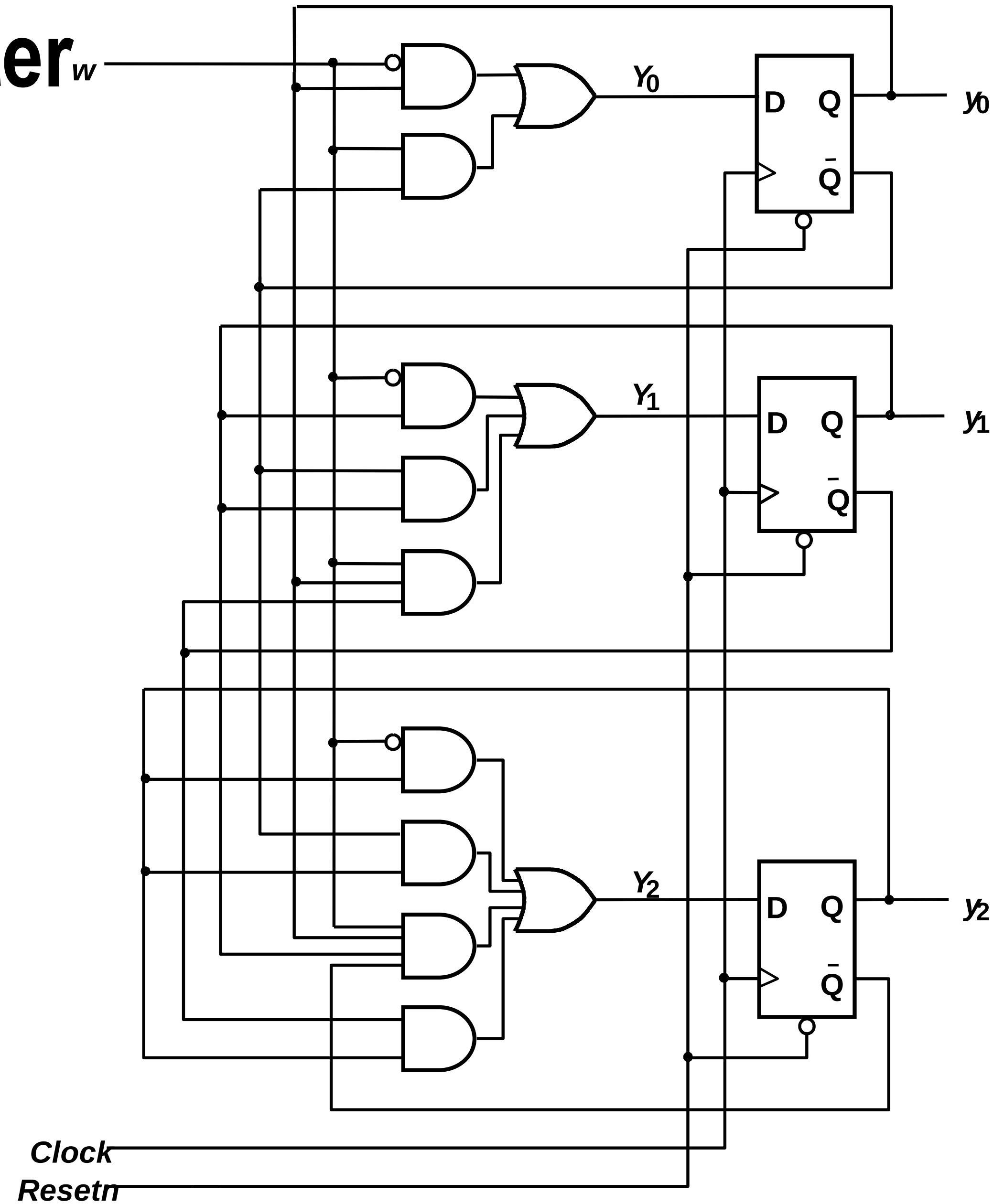
$wy_2 \backslash y_1y_0$	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	1	0	1
10	0	1	0	1

$Y_1 = \bar{w}y_1 + y_1\bar{y}_0 + wy_0\bar{y}_1$

$wy_2 \backslash y_1y_0$	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	0	1
10	0	0	1	0

$Y_2 = \bar{w}y_2 + \bar{y}_0y_2 + \bar{y}_1y_2 + wy_0y_1\bar{y}_2$

Circuit of the Counter



NTU ESOE

Implementation Using JK-type Flip-Flops

- If a flip-flop in state 0 is to remain in state 0, then J=0 and K=d. (where d means that K can be equal either 0 or 1)
- If a flip-flop in state 0 is to remain in state 1, then J=1 and K=d.
- If a flip-flop in state 1 is to remain in state 1, then J=d and K=0.
- If a flip-flop in state 1 is to remain in state 0, then J=d and K=1.

	Present state $y_2y_1y_0$	Flip-flop inputs								Count $z_2z_1z_0$
		$w = 0$				$w = 1$				
		$Y_2Y_1Y_0$	J_2K_2	J_1K_1	J_0K_0	$Y_2Y_1Y_0$	J_2K_2	J_1K_1	J_0K_0	
A	000	000	0d	0d	0d	001	0d	0d	1d	000
B	001	001	0d	0d	d0	010	0d	1d	d1	001
C	010	010	0d	d0	0d	011	0d	d0	1d	010
D	011	011	0d	d0	d0	100	1d	d1	d1	011
E	100	100	d0	0d	0d	101	d0	0d	1d	100
F	101	101	d0	0d	d0	110	d0	1d	d1	101
G	110	110	d0	d0	0d	111	d0	d0	1d	110
H	111	111	d0	d0	d0	000	d1	d1	d1	111

- $J_0 = K_0 = w$
- $J_1 = K_1 = wy_0$
- $J_2 = K_2 = wy_0y_1$

wy_2 \ y_1y_0	00	01	11	10
00	0	d	d	0
01	0	d	d	0
11	1	d	d	1
10	1	d	d	1

$J_0 = w$

wy_2 \ y_1y_0	00	01	11	10
00	d	0	0	d
01	d	0	0	d
11	d	1	1	d
10	d	1	1	d

$K_0 = w$

wy_2 \ y_1y_0	00	01	11	10
00	0	0	d	d
01	0	0	d	d
11	0	1	d	d
10	0	1	d	d

$J_1 = wy_0$

- $J_2 = K_2 = (wy_0)y_1 = J_1Y_1$
- $J_n = K_n = (wy_0 \dots y_{n-2})y_{n-1} = J_{n-1}Y_{n-1}$

wy_2 \ y_1y_0	00	01	11	10
00	d	d	0	0
01	d	d	0	0
11	d	d	1	0
10	d	d	1	0

$K_1 = wy_0$

wy_2 \ y_1y_0	00	01	11	10
00	0	0	0	0
01	d	d	d	d
11	d	d	d	d
10	0	0	1	0

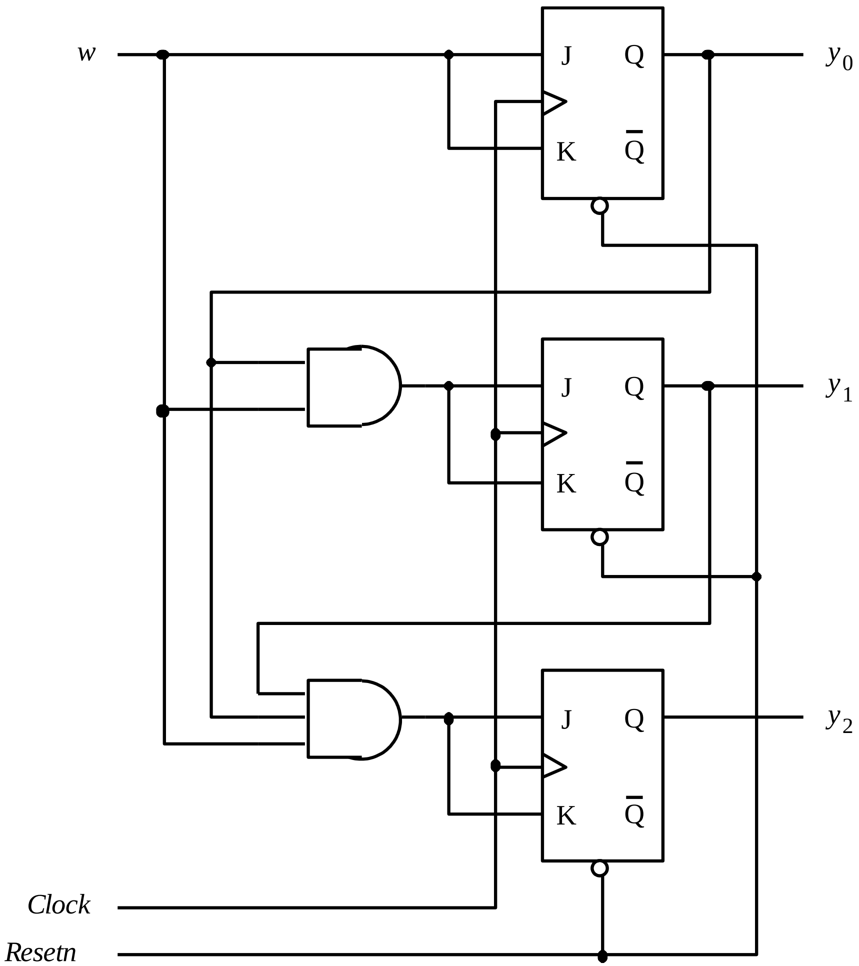
$J_2 = wy_0y_1$

wy_2 \ y_1y_0	00	01	11	10
00	d	d	d	d
01	0	0	0	0
11	0	0	1	0
10	d	d	d	d

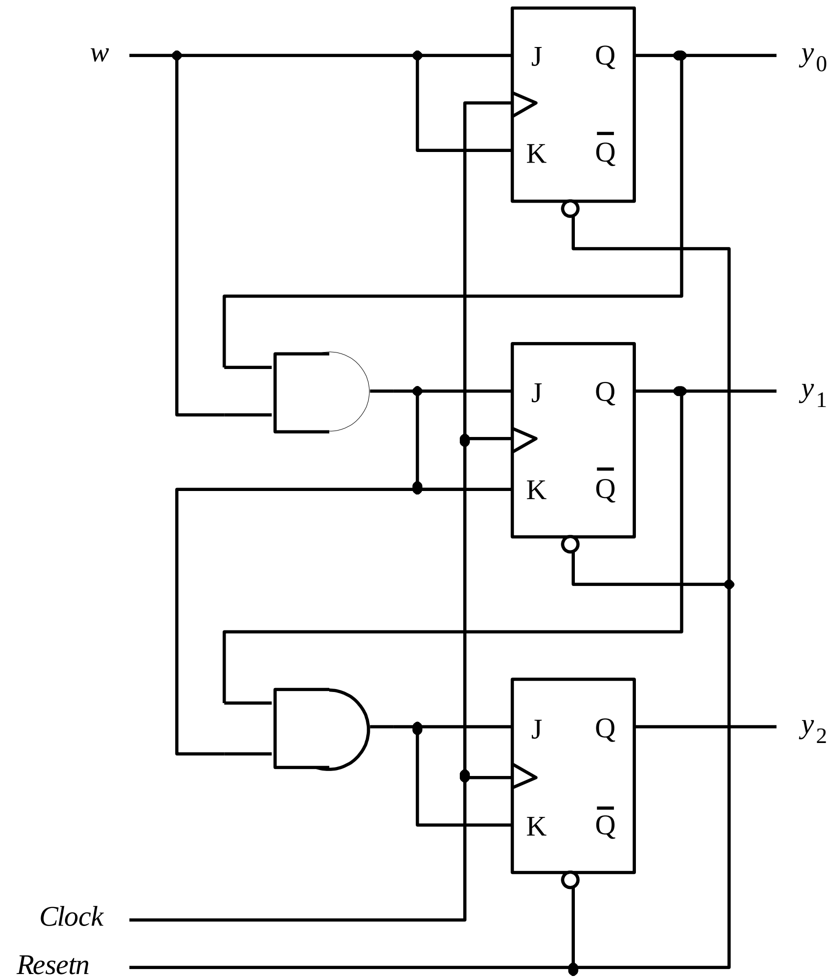
$K_2 = wy_0y_1$

Circuit Diagram

- Ordinary



Factored form



Counting 0,4,2,6,1,5,3,7,0,4 …

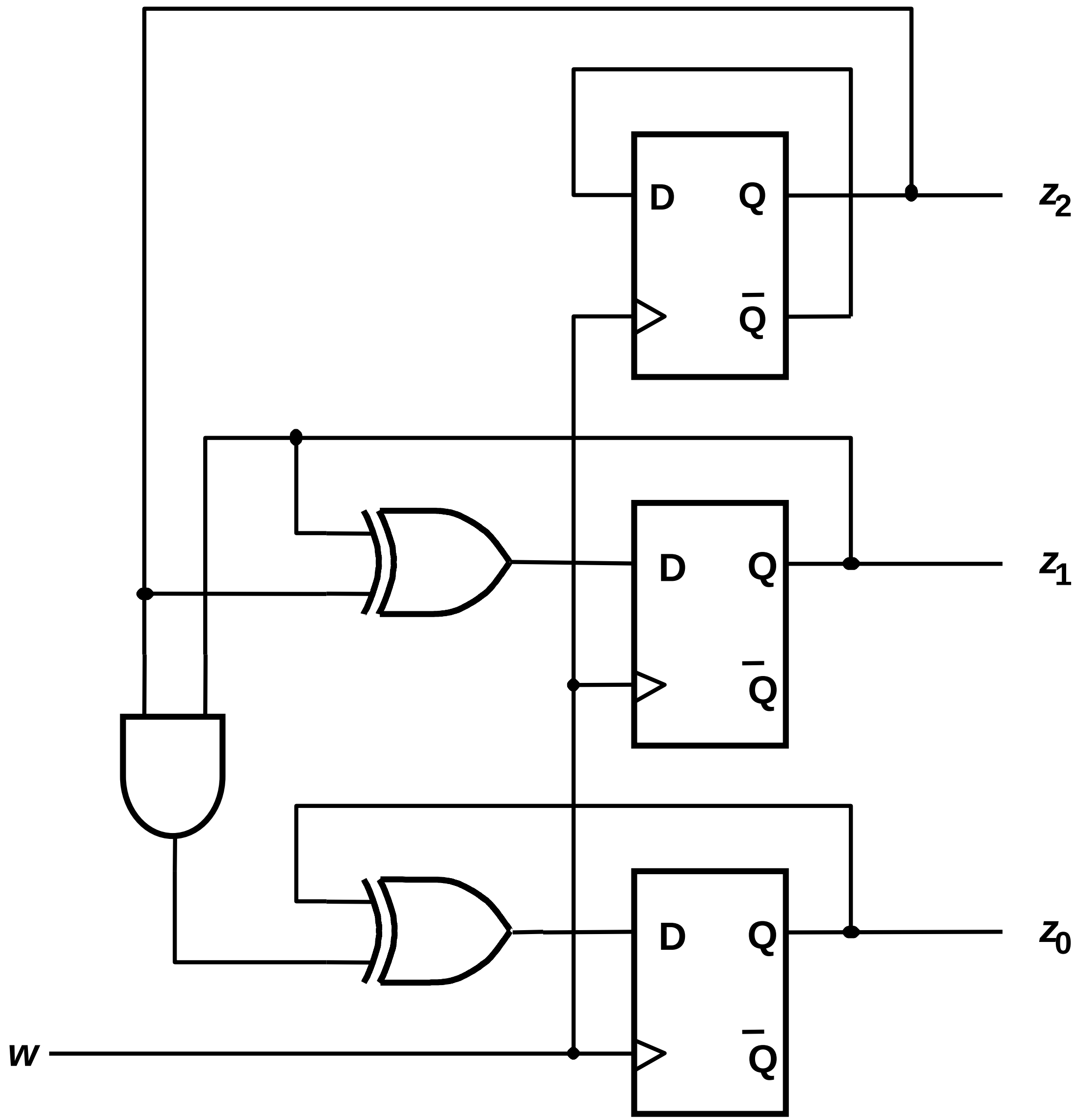
$$D_2 = Y_2 = \overline{y_2}$$

$$D_1 = Y_1 = y_1 \oplus y_2$$

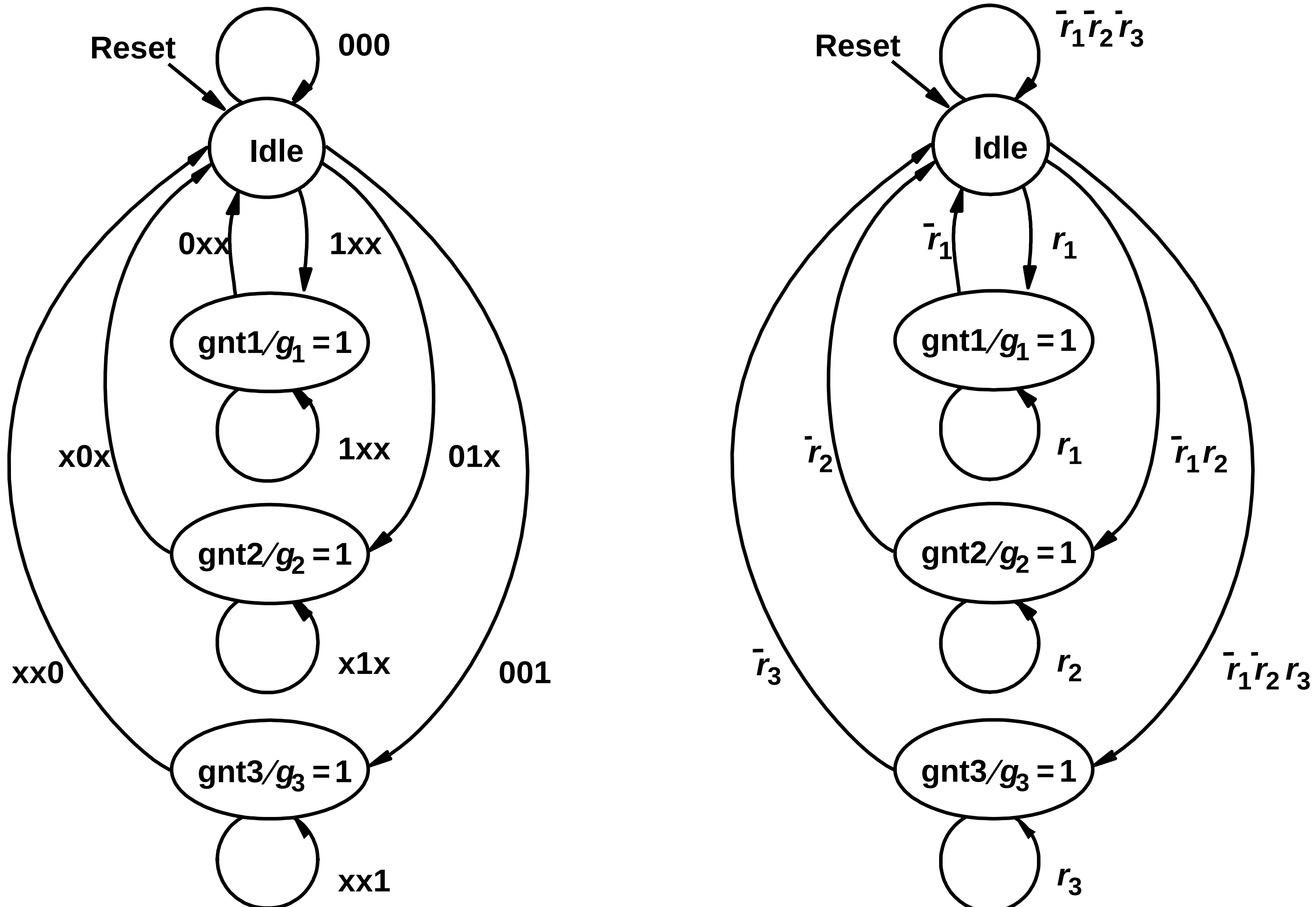
Present state	Next state	Output $z_2z_1z_0$
A	B	000
B	C	100
C	D	010
D	E	110
E	F	001
F	G	101
G	H	011
H	A	111

Present state $y_2y_1y_0$	Next state $Y_2Y_1Y_0$	Output $z_2z_1z_0$
000	100	000
100	010	100
010	110	010
110	001	110
001	101	001
101	011	101
011	111	011
111	000	111

Different Counter Circuit Diagram



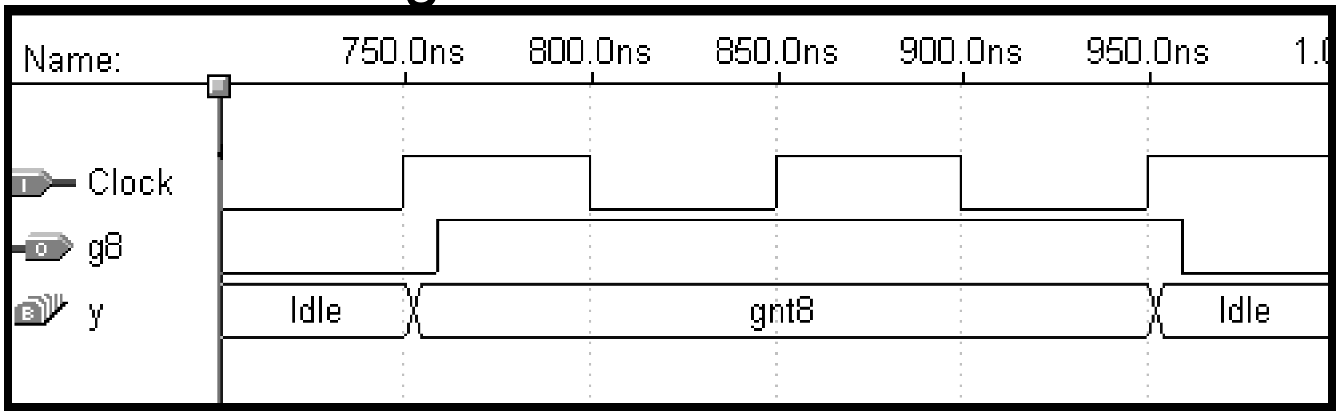
Arbiter for a 3 Device System



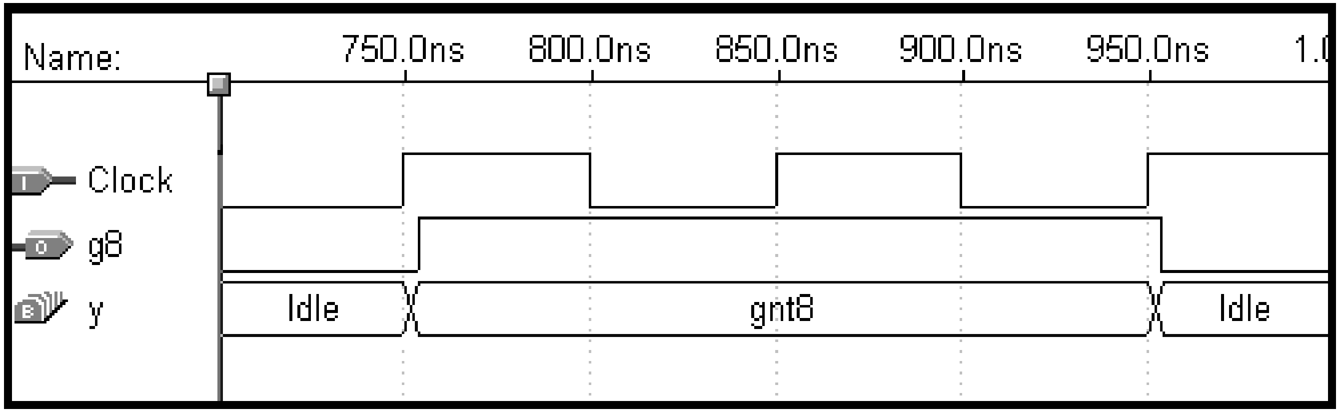
NTU ESOE

Minimized Delay

- 7ns in initial design



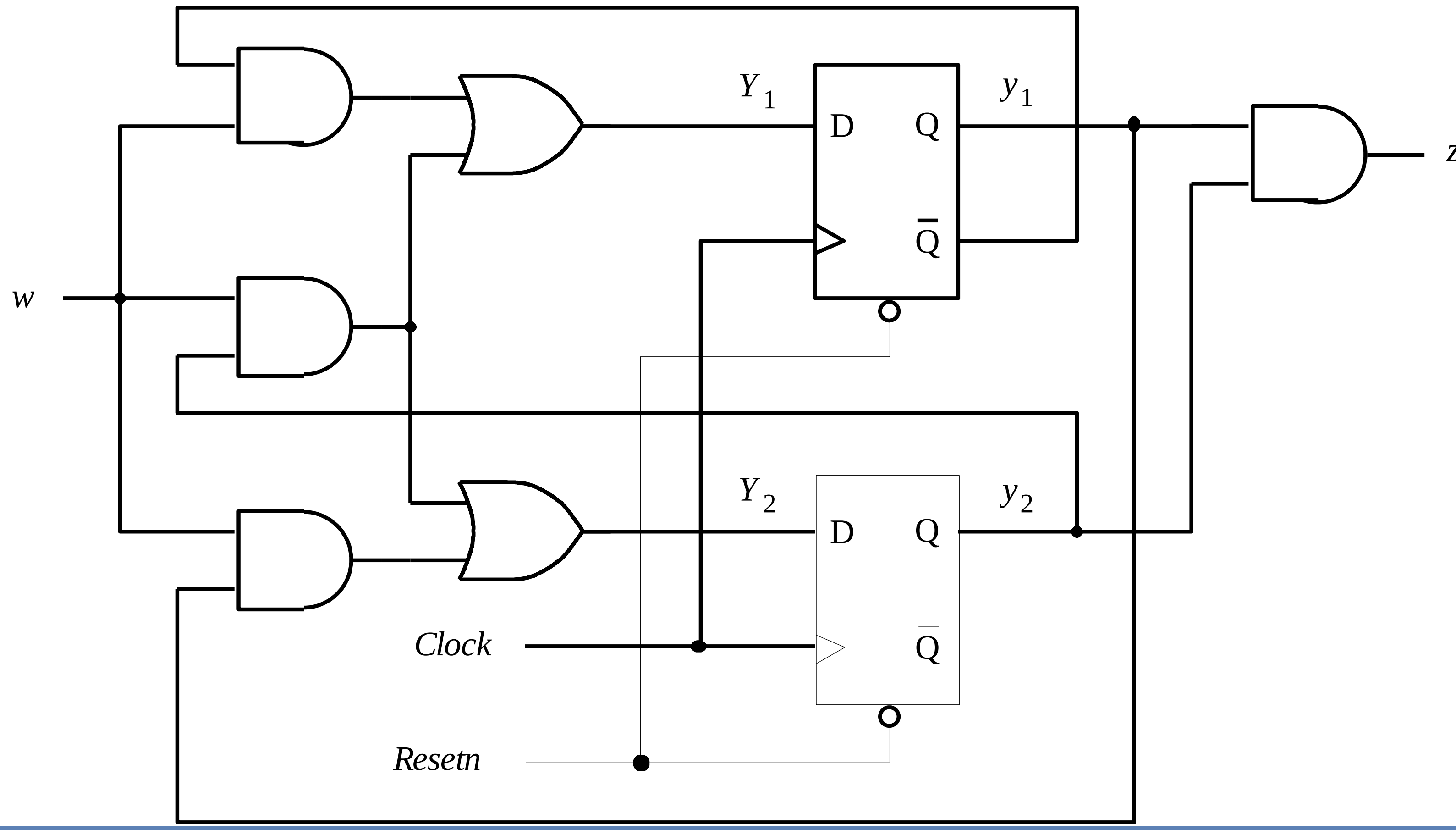
- 2ns with one-hot encoding



NTU ESOE

Ex. 8.9 Analyze the FSM

- $Y_1 = w\overline{y_1} + wy_2 \quad Y_2 = wy_1 + wy_2 \quad z = y_1 y_2$



NTU ESOE

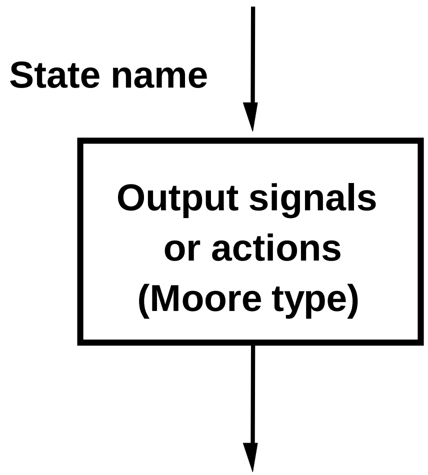
State Table

- $Y_1 = w\overline{y_1} + wy_2 \quad Y_2 = wy_1 + wy_2 \quad z = y_1y_2$

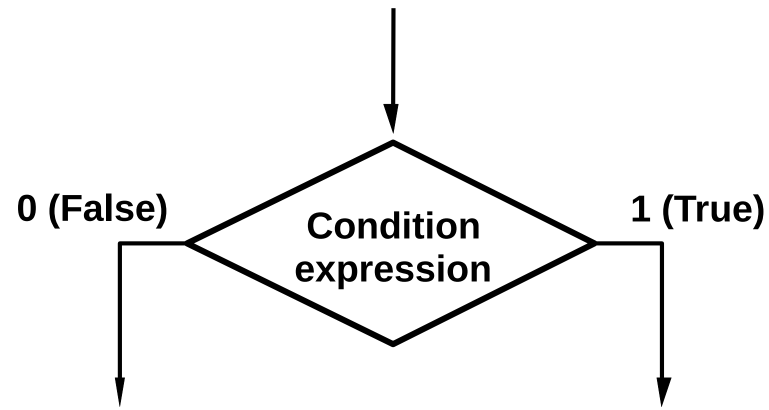
Present state y_2y_1	Next State w = 0 Y_2Y_1	Next State w = 1 Y_2Y_1	Output z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	0 0	11	0
1 1	0 0	11	1

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

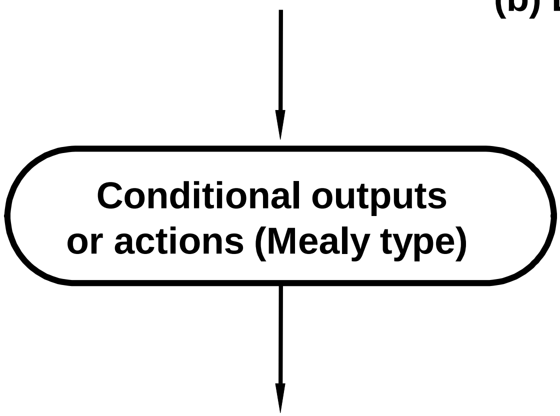
Algorithmic State Machine (ASM) Charts



(a) State box

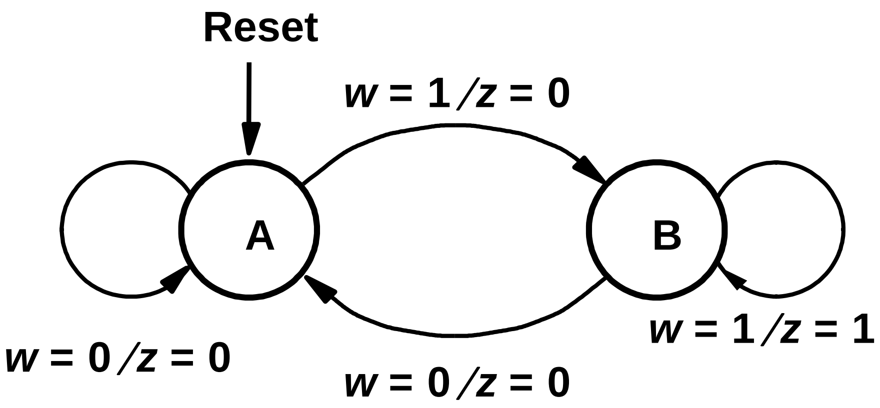


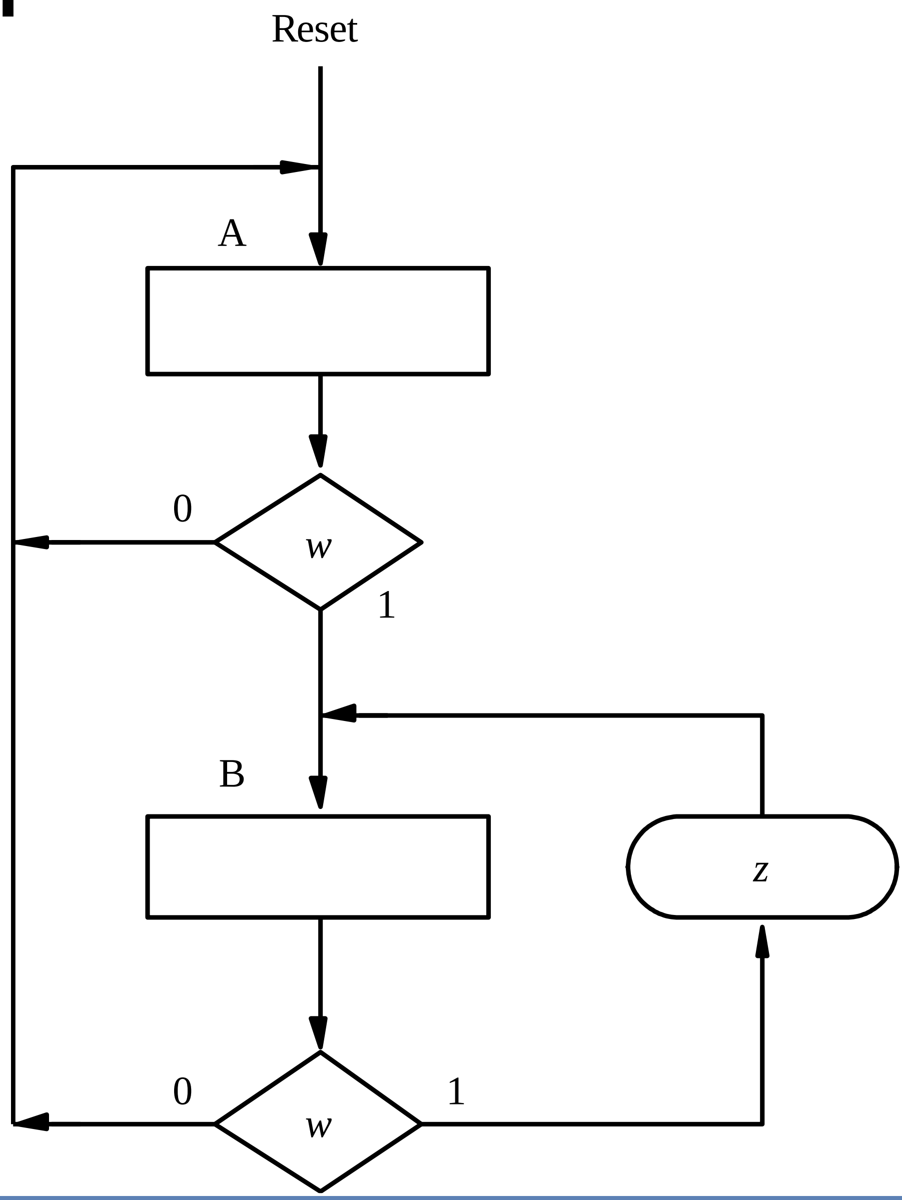
(b) Decision box



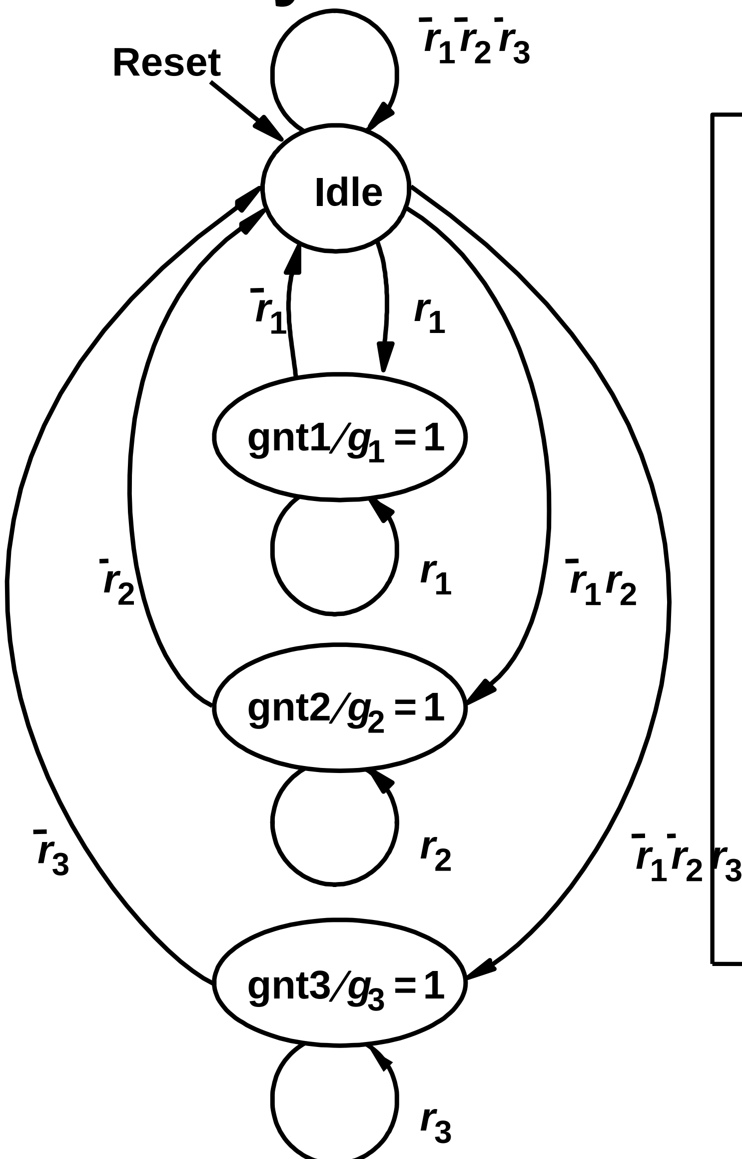
(c) Conditional output box

ASM Style State Diagram

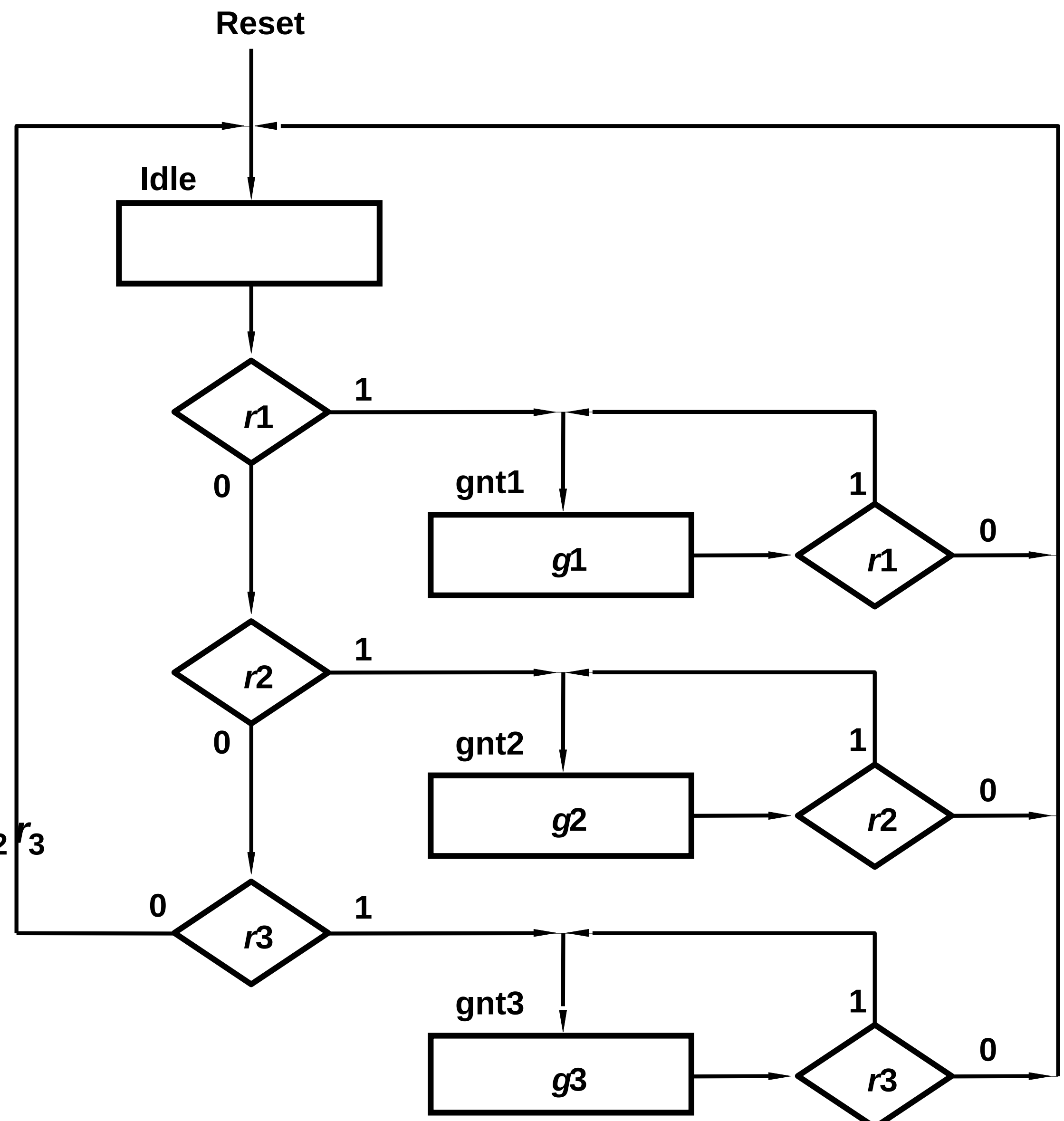




ASM Style



Reset
$\bar{r}_1\bar{r}_2\bar{r}_3$
Idle
$\bar{r}_1$
r_1
gnt1/$g_1 = 1$
r_1
$\bar{r}_2$
$\bar{r}_1 r_2$
gnt2/$g_2 = 1$
r_2
$\bar{r}_3$
$\bar{r}_1\bar{r}_2 r_3$
gnt3/$g_3 = 1$
r_3



Reset
Idle
$r1$
1
0
gnt1
$g1$
1
$r1$
0
$r2$
1
0
gnt2
$g2$
1
$r2$
0
0
$r3$
1
gnt3
$g3$
1
$r3$
0

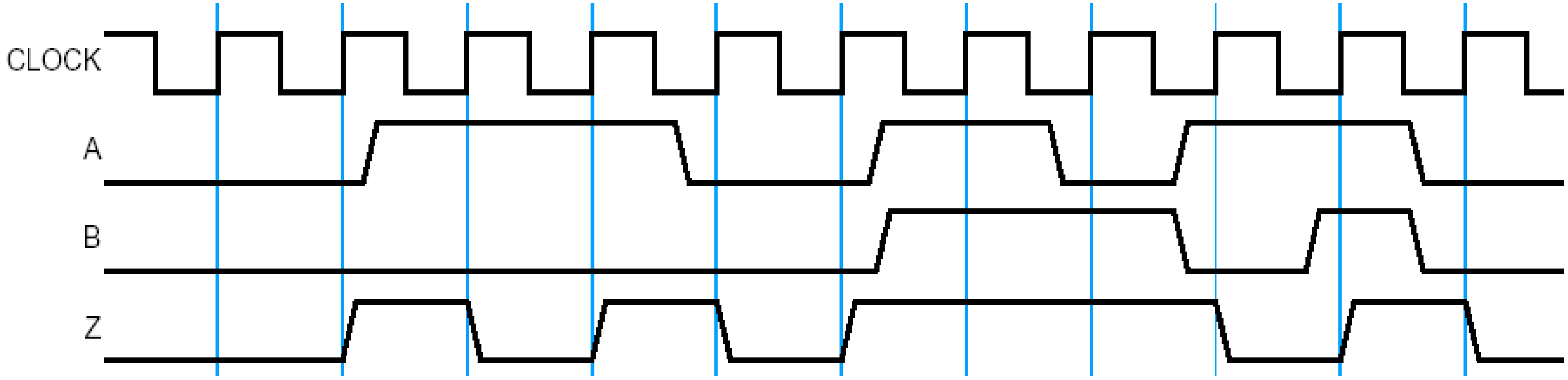
More Examples (not on our textbook)

Mutual Exclusion and All Inclusion

- The transition expressions on arcs leaving a particular state must be mutually exclusive and all inclusive:
- Mutually exclusive: No two transition expressions can equa1 for the same input combination, since a machine can’t have two next states for one input combination.
- All inclusive: For every possible input combination, some transition expression must equal 1, so that all next states are defined.

A State-Table Design Example

- Design a clocked synchronous state machine with two inputs, A and B, and a ingle output Z that is 1 if:
  - A had the same value at each of the two previous clock ticks, *or*
  - B has been 1 since the last time that the first condition was true



Evolution of State Table

(a)

Meaning	S	00	01	11	10	Z
		AB				
Initial state	INIT					0
	. . .					
	. . .					
	. . .					
		S*				

(b)

Meaning	S	00	01	11	10	Z
		AB				
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0					0
Got a 1 on A	A1					0
		S*				

(c)

Meaning	S	00	01	11	10	Z
		AB				
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1					0
Got two equal A inputs	OK					1
		S*				

(d)

Meaning	S	00	01	11	10	Z
		AB				
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK					1
		S*				

Evolution of State Table (Cont')

(a)

Meaning	S	AB 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1
		S*				

(b)

Meaning	S	AB 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1
		S*				

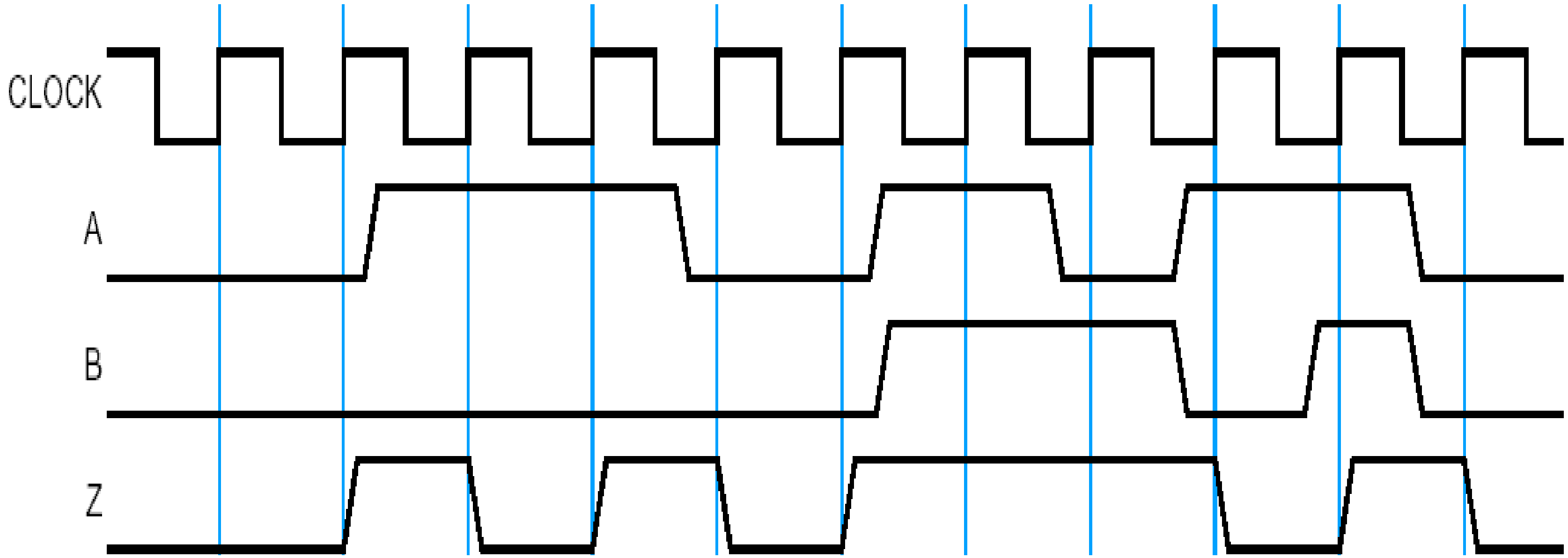
(c)

Meaning	S	AB 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1
		S*				

(d)

Meaning	S	AB 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1
		S*				

Timing Diagram with States



(a)

Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK00	OK00	A1	A1	0
Got a 1 on A	A1	A0	A0	OK11	OK11	0
Got 00 on A	OK00	OK00	OK00	OKA1	A1	1
Got 11 on A	OK11	A0	OKA0	OK11	OK11	1
OK, got a 0 on A	OKA0	OK00	OK00	OKA1	A1	1
OK, got a 1 on A	OKA1	A0	OKA0	OK11	OK11	1
		S*				

(b)

Meaning	S	AB 00	AB 01	AB 11	AB 10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK00	OK00	A1	A1	0
Got a 1 on A	A1	A0	A0	OK11	OK11	0
Got 00 on A	OK00	OK00	OK00	A001	A1	1
Got 11 on A	OK11	A0	A110	OK11	OK11	1
Got 001 on A, B=1	A001	A0	AE10	OK11	OK11	1
Got 110 on A, B=1	A110	OK00	OK00	AE01	A1	1
Got bb...10 on A, B=1	AE10	OK00	OK00	AE01	A1	1
Got bb...01 on A, B=1	AE01	A0	AE10	OK11	OK11	1
		S*				

State Assignment Example

	A B				
S	00	01	11	10	Z
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
	S*				

	Assignment			
State name	Simplest Q1–Q3	Decomposed Q1–Q3	One-hot Q1–Q5	Almost one-hot Q1–Q4
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

Unused States

- *Minimal risk* : Assumes that it is possible for the state machine somehow to get into one of the unused (or “illegal”) states, perhaps because of a hardware failure, an unexpected input, or a design error. Therefore, all of the unused state-variable combinations are identified and explicit next-state entries are made so that, for any input combination, the unused states go to the “initial” state, the “idle” state, or some other “safe” state.
- *Minimal cost* : Assumes that the machine will never enter an unused state. Therefore, in the transition and excitation tables, the next-state entries of the unused states can be marked as “don’t-cares.” In most cases this simplifies the excitation logic. However, the machine’s behavior my be weird if it ever does enter an unused state.

Transition/ Excitation Table

- Transition/Output table.

	AB				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	Q1* Q2* Q3*				

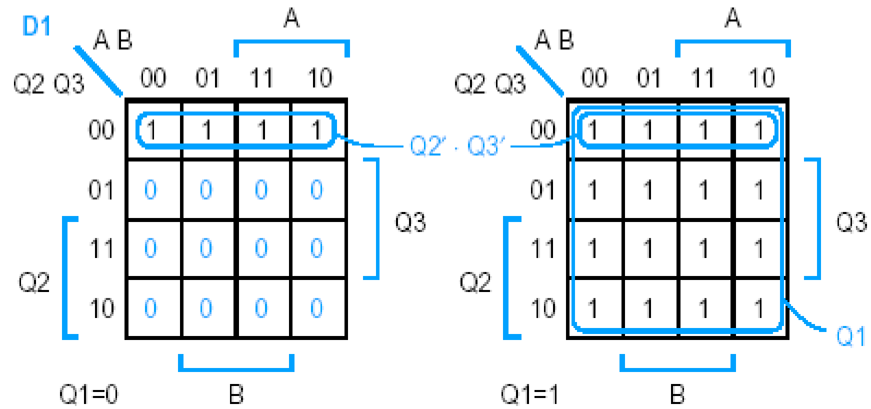
Excitation/Output table.

	AB				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1 D2 D3				

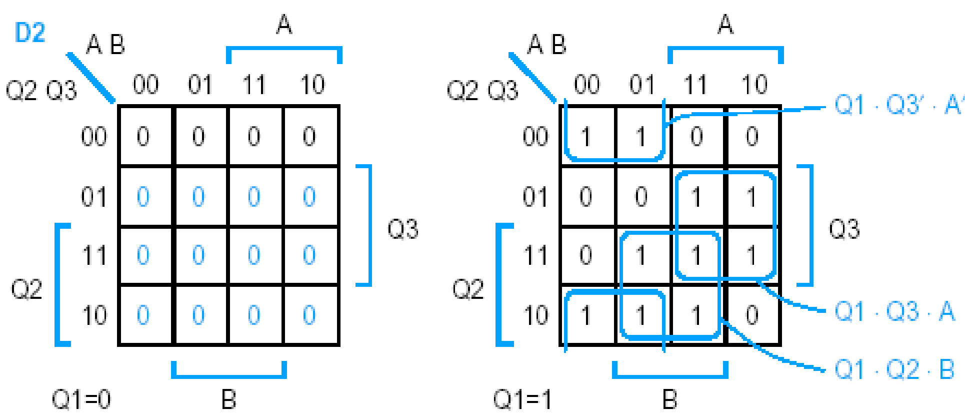
Minimal Risk Excitation Maps

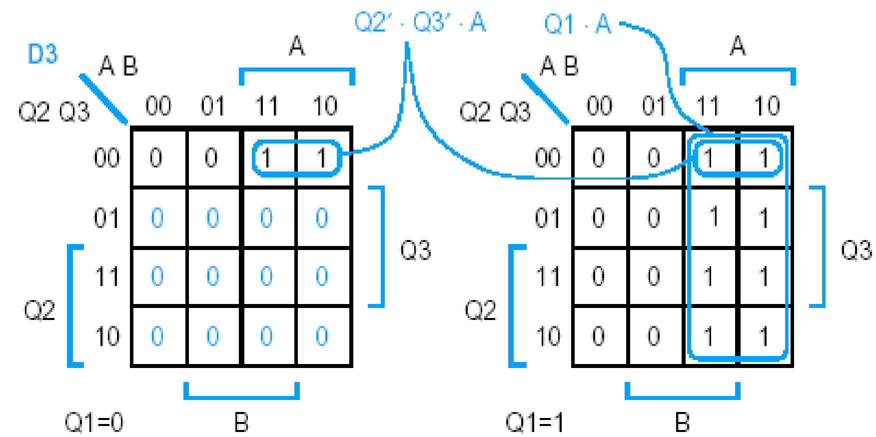
Minimal risk: Assume unused states go to state 000

	A B				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1 D2 D3				



D1= Q1 + Q2’·Q3’





D2= Q1·Q3’·A’+ Q1·Q3·A+Q1·Q2·B D3= Q1·A+ Q2·’Q3’·A

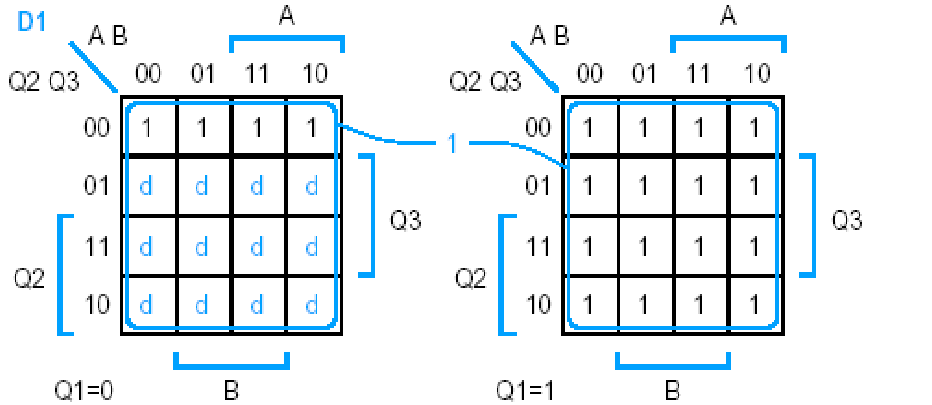
Minimal Risk Logic Equation

- Excitation equations
  - $D1 = Q1 + Q2' \cdot Q3'$
  - $D2 = Q1 \cdot Q3' \cdot A' + Q1 \cdot Q3 \cdot A + Q1 \cdot Q2 \cdot B$
  - $D3 = Q1 \cdot A + Q2 \cdot ' Q3' \cdot A$
- Output Equations (Z =1 for 110 and 111)
  - $Z = Q1 \cdot Q2 \cdot Q3' + Q1 \cdot Q2 \cdot Q3$
    $= Q1 \cdot Q2$

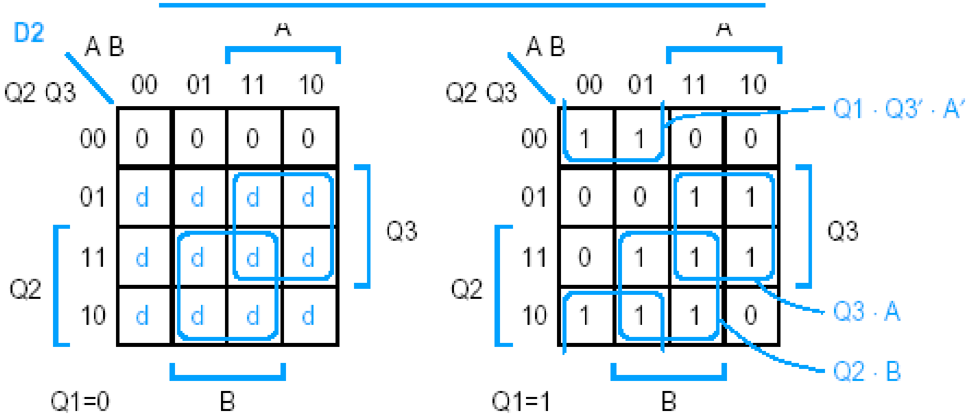
Minimal Cost Excitation Map

- Next-state of unused states are "don't care".
- Output equation Z = Q2.

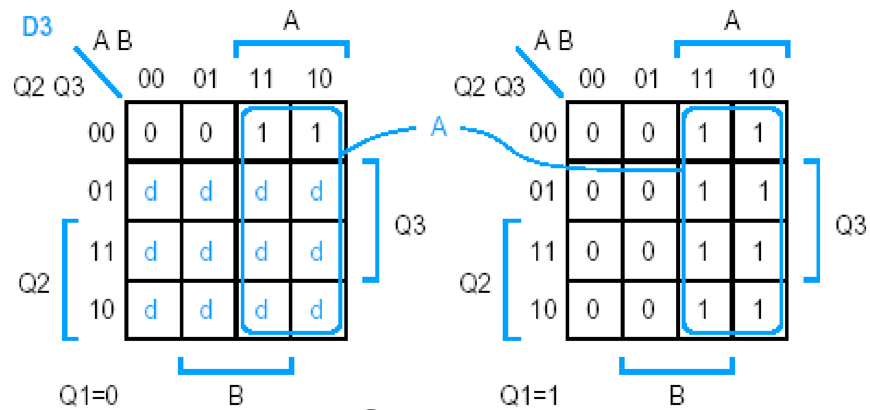
Q1 Q2 Q3	AB = 00	AB = 01	AB = 11	AB = 10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1 D2 D3				



D1 = 1

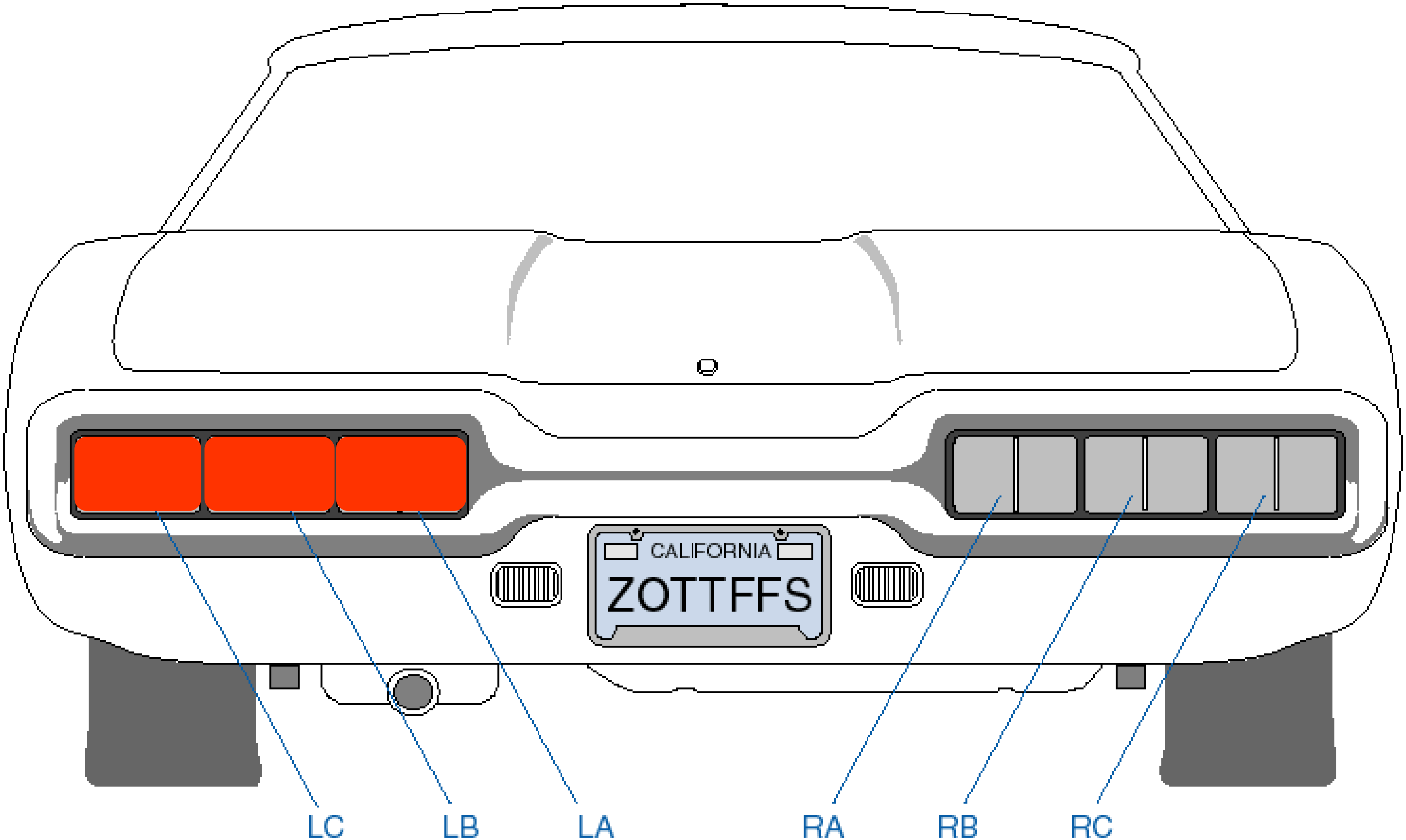


D2= Q1·Q3' ·A'+Q3·A+Q2·B

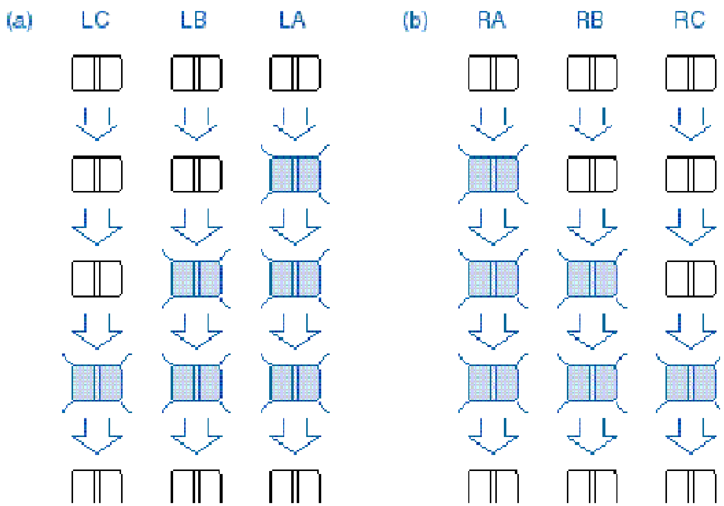


D3= A

T-bird tail-lights example

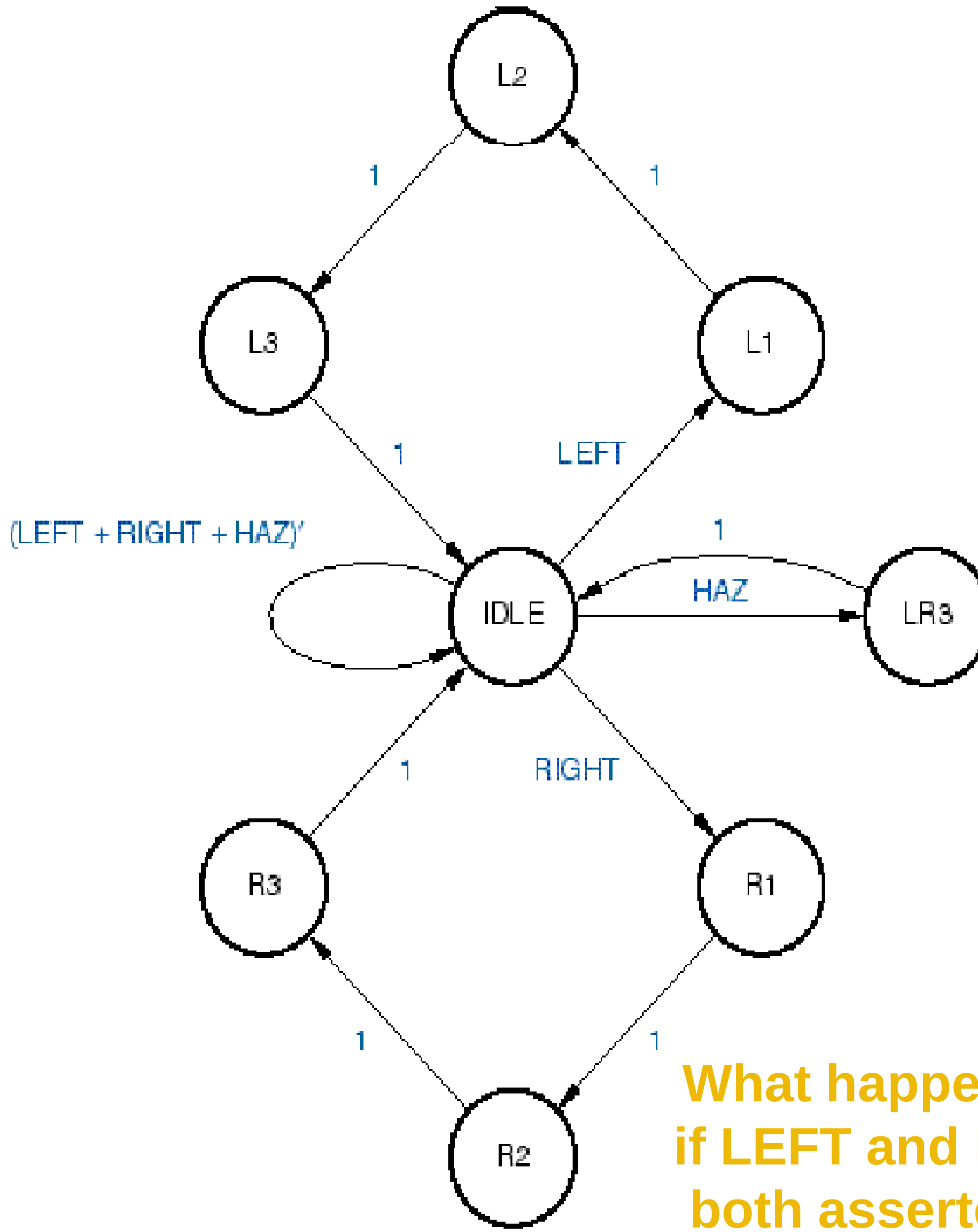


First Trial State Diagram



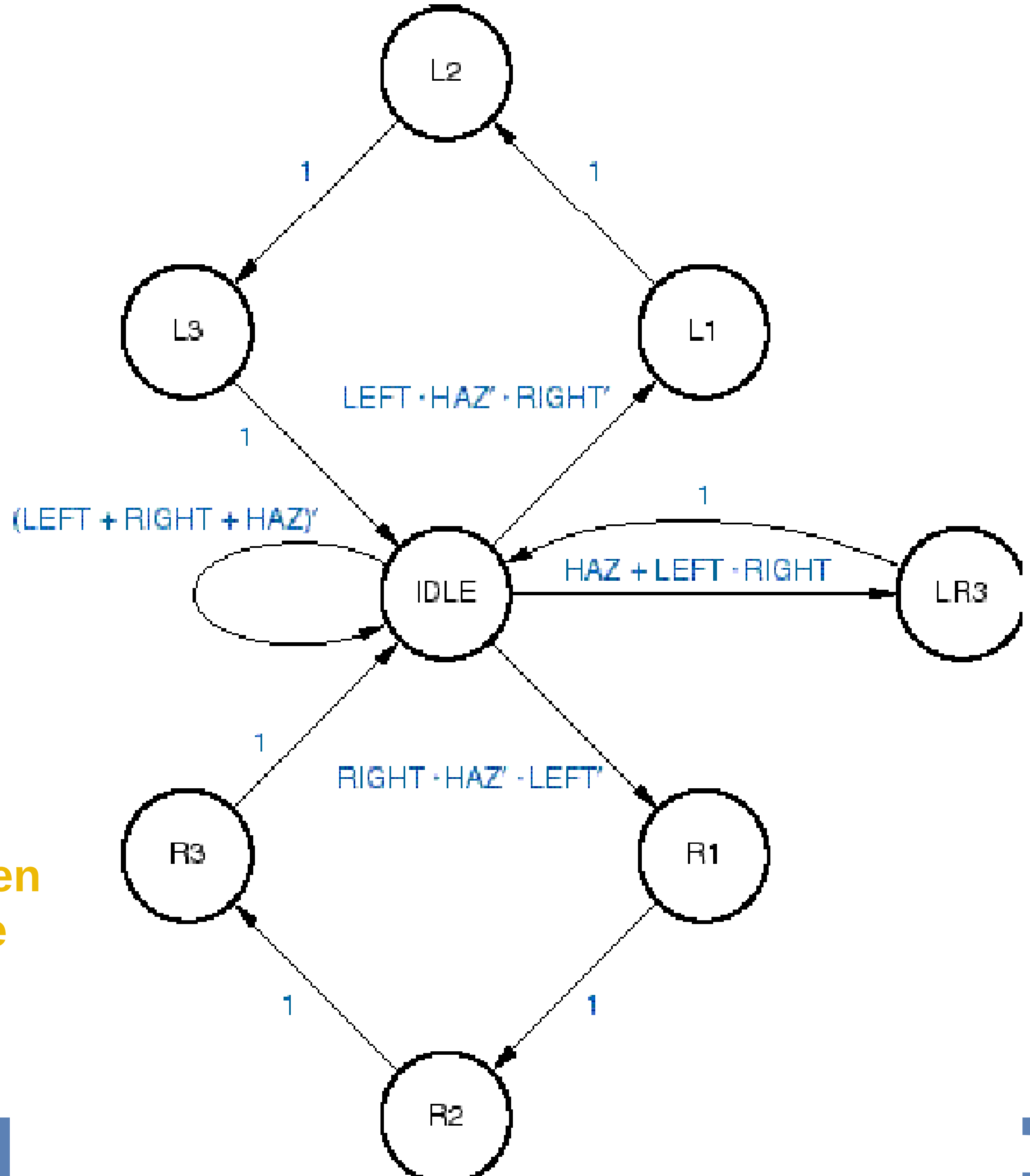
Output Table

State	LC	LB	LA	RA	RB	RC
IDLE	0	0	0	0	0	0
L1	0	0	1	0	0	0
L2	0	1	1	0	0	0
L3	1	1	1	0	0	0
R1	0	0	0	1	0	0
R2	0	0	0	1	1	0
R3	0	0	0	1	1	1
LR3	1	1	1	1	1	1



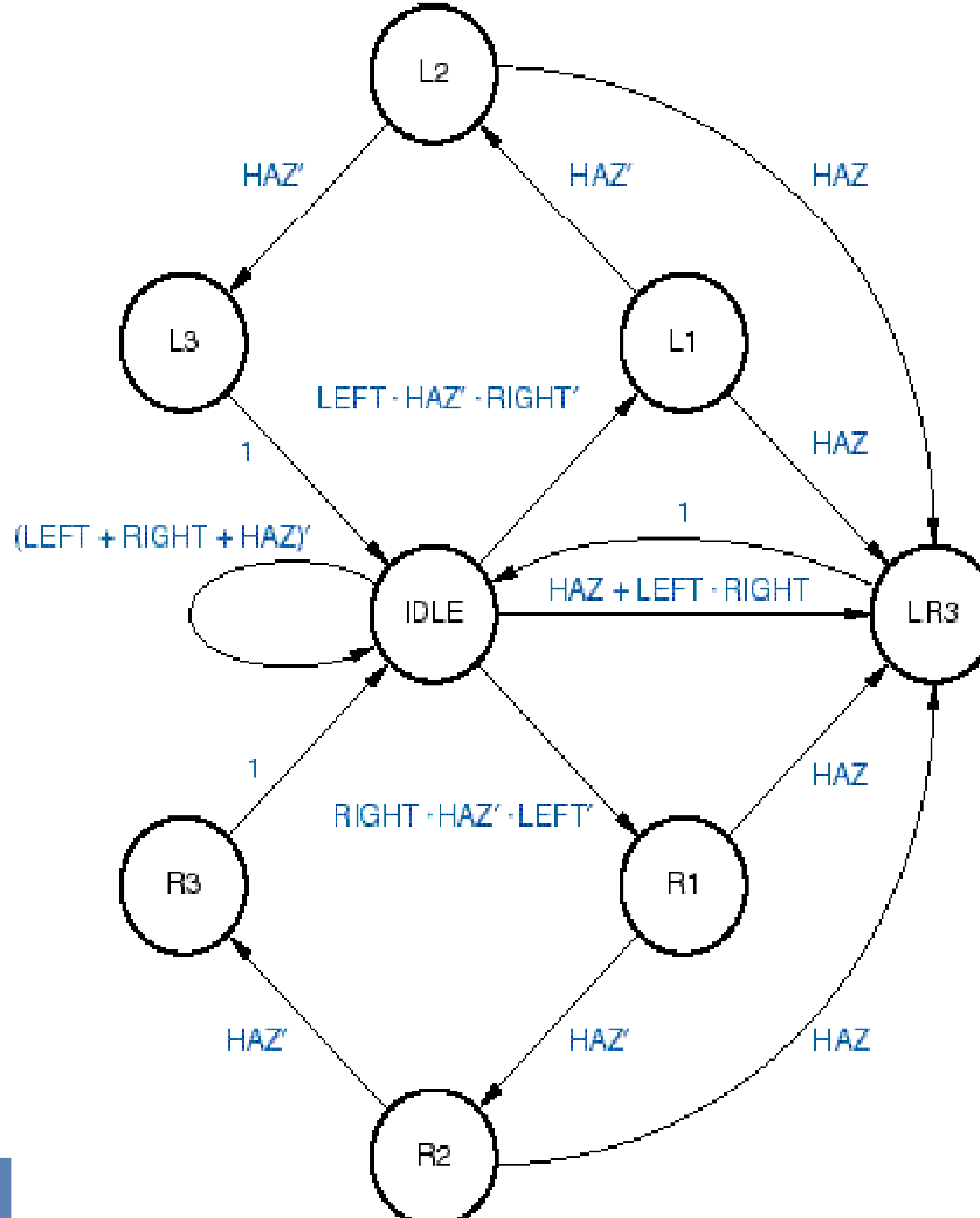
What happened if LEFT and HAZ both asserted?

Second Trial State Diagram



What happened if HAZ asserted when LEFT or RIGHT are already asserted?

Final State Diagram



State Assignment and Output Logic

State	*Q2*	*Q1*	*Q0*
IDLE	0	0	0
L1	0	0	1
L2	0	1	1
L3	0	1	0
R1	1	0	1
R2	1	1	1
R3	1	1	0
LR3	1	0	0

- LA = L1+L2+L3+LR3
- LB = L2+L3+LR3
- LC = L3+LR3

- RA = R1+R2+R3+LR3
- RB = R2+R3+LR3
- RC = R3+LR3

Transition List

S	*Q2*	*Q1*	*Q0*	*Transition Expression*	*S**	*Q2**	*Q1**	*Q0**
IDLE	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1
IDLE	0	0	0	HAZ + LEFT · RIGHT	LR3	1	0	0
IDLE	0	0	0	RIGHT · HAZ′ · LEFT′	R1	1	0	1
L1	0	0	1	HAZ′	L2	0	1	1
L1	0	0	1	HAZ	LR3	1	0	0
L2	0	1	1	HAZ′	L3	0	1	0
L2	0	1	1	HAZ	LR3	1	0	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	HAZ′	R2	1	1	1
R1	1	0	1	HAZ	LR3	1	0	0
R2	1	1	1	HAZ′	R3	1	1	0
R2	1	1	1	HAZ	LR3	1	0	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

Excitation Equation from Transition List

- $Q2^* = Q2' \cdot Q1' \cdot Q0' \cdot (HAZ + LEFT \cdot RIGHT)$
  $+ Q2' \cdot Q1' \cdot Q0 \cdot (RIGHT \cdot HAZ' + LEFT')$
  $+ Q2' \cdot Q1' \cdot Q0 \cdot (HAZ)$
  $+ Q2' \cdot Q1 \cdot Q0 \cdot (HAZ)$
  $+ Q2 \cdot Q1' \cdot Q0 \cdot (HAZ')$
  $+ Q2 \cdot Q1' \cdot Q0 \cdot (HAZ)$
  $+ Q2 \cdot Q1 \cdot Q0 \cdot (HAZ')$
  $+ Q2 \cdot Q1 \cdot Q0 \cdot (HAZ)$

  $= Q2' \cdot Q1' \cdot Q0' \cdot (HAZ + RIGHT)$
  $+ Q2' \cdot Q0 \cdot (HAZ)$
  $+ Q2 \cdot Q0$

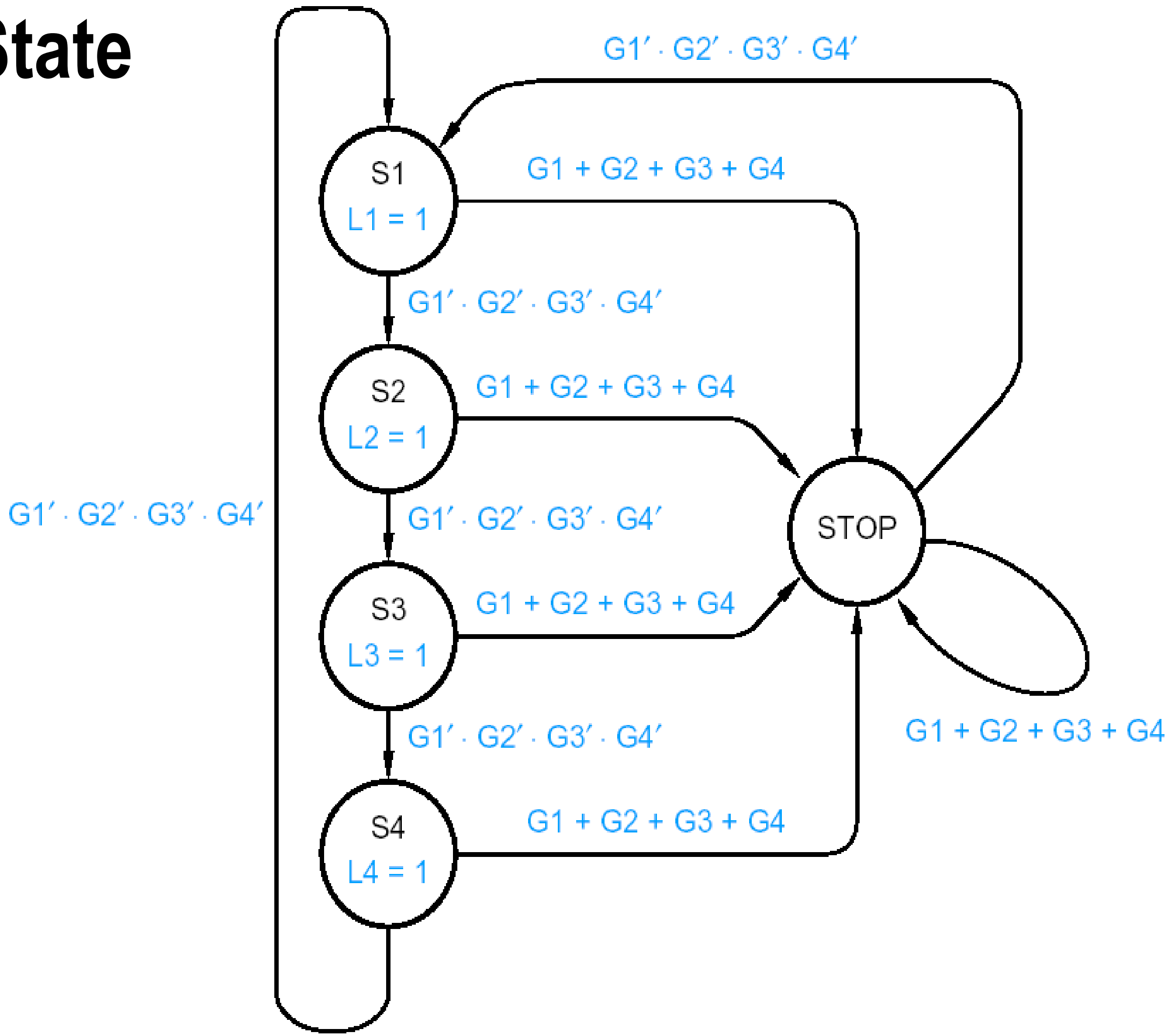
Excitation Equation from Transition List (Cont')

- $Q1^* = Q2' \cdot Q1' \cdot Q0 \cdot (HAZ')$
  $+ Q2' \cdot Q1 \cdot Q0 \cdot (HAZ')$
  $+ Q2 \cdot Q1' \cdot Q0 \cdot (HAZ')$
  $+ Q2 \cdot Q1 \cdot Q0 \cdot (HAZ')$
  $= Q0 \cdot HAZ'$
- $Q0^* = Q2' \cdot Q1' \cdot Q0' \cdot (LEFT \cdot HAZ' \cdot RIGHT')$
  $+ Q2' \cdot Q1' \cdot Q0' \cdot (RIGHT \cdot HAZ' + LEFT')$
  $+ Q2' \cdot Q1' \cdot Q0 \cdot (HAZ')$
  $+ Q2 \cdot Q1' \cdot Q0 \cdot (HAZ')$
  $= Q2' \cdot Q1' \cdot Q0' \cdot HAZ' + Q1' \cdot Q0 \cdot HAZ'$

The Guessing game Example

- Design a clocked synchronous state machine with four inputs, G1-G4, that are connected to pushbuttons.
- The Machine has four outputs, L1-L4, connected to lamps or LEDs located near the like-numbered pushbuttons.
- There is also an ERR output connected to a red lamp. In normal operation the l1-L4 outputs display a 1out-of4 pattern. At each clock tick, the pattern is rotated by one position; the clock frequency is about 4 Hz.
- Guesses are made by pressing a pushbutton, which asserts and input Gi When any Gi input is asserted, the ERR output is asserted if the "wrong" pushbutton was pressed, that is, if the Gi input detected at the clock tick does not have same number as the lamp output that was asserted before the clock tick.
- Once a guess has been made, play stops and the ERR output maintains the same value for one or more clock ticks until the Gi input is negated, then play resumes.

First Try State Diagram



Correct State Diagram

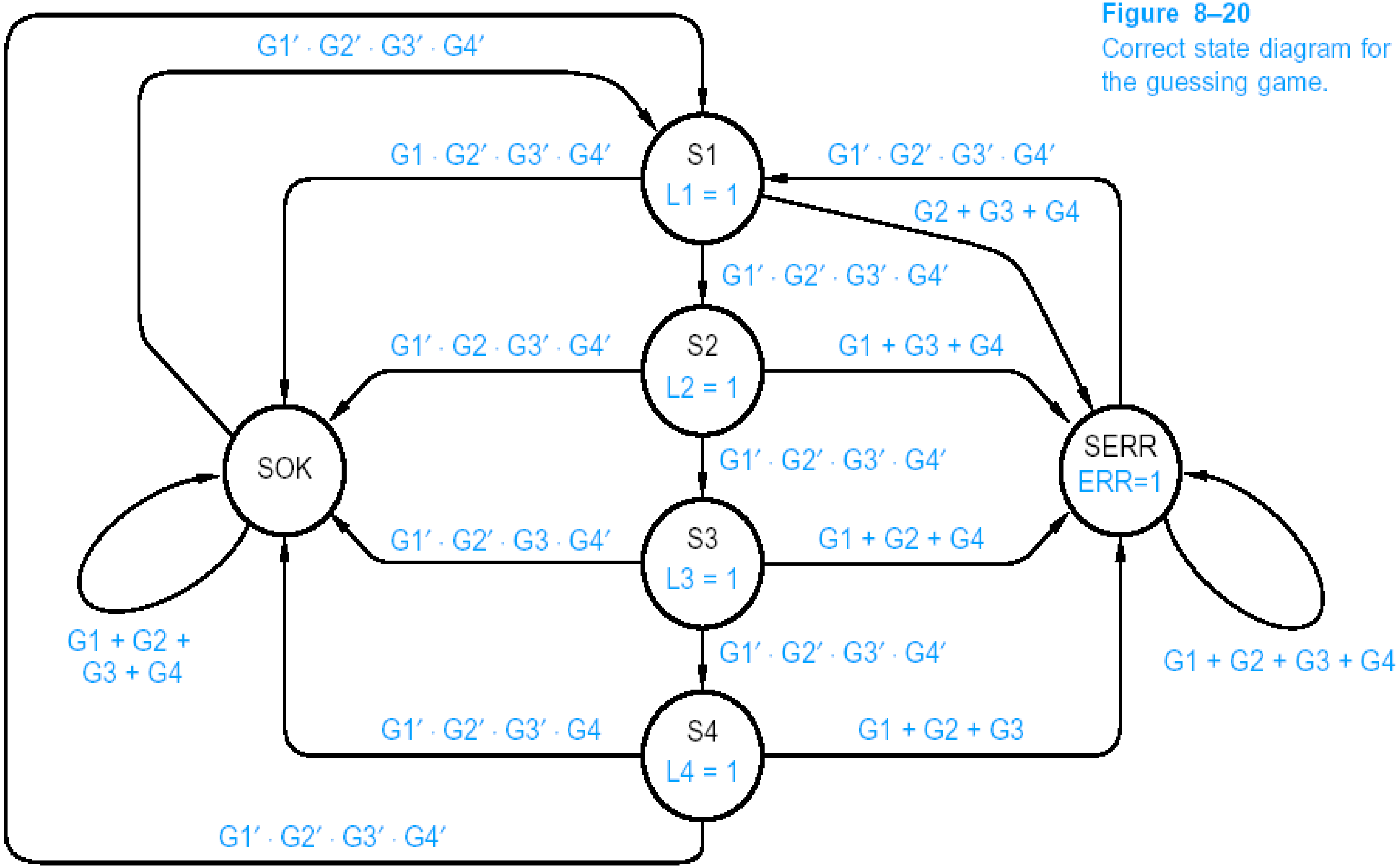


Figure 8–20
Correct state diagram for the guessing game.

Transition List

Current State					Next State				Output				
S	*Q2*	*Q1*	*Q0*	*Transition Expression*	*S**	*Q2**	*Q1**	*Q0**	*L1*	*L2*	*L3*	*L4*	*ERR*
S1	0	0	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S2	0	0	1	1	0	0	0	0
S1	0	0	0	$G1 \cdot G2' \cdot G3' \cdot G4'$	SOK	1	0	0	1	0	0	0	0
S1	0	0	0	$G2 + G3 + G4$	SERR	1	0	1	1	0	0	0	0
S2	0	0	1	$G1' \cdot G2' \cdot G3' \cdot G4'$	S3	0	1	1	0	1	0	0	0
S2	0	0	1	$G1' \cdot G2 \cdot G3' \cdot G4'$	SOK	1	0	0	0	1	0	0	0
S2	0	0	1	$G1 + G3 + G4$	SERR	1	0	1	0	1	0	0	0
S3	0	1	1	$G1' \cdot G2' \cdot G3' \cdot G4'$	S4	0	1	0	0	0	1	0	0
S3	0	1	1	$G1' \cdot G2' \cdot G3 \cdot G4'$	SOK	1	0	0	0	0	1	0	0
S3	0	1	1	$G1 + G2 + G4$	SERR	1	0	1	0	0	1	0	0
S4	0	1	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	0	0	0	0	0	0	1	0
S4	0	1	0	$G1' \cdot G2' \cdot G3' \cdot G4$	SOK	1	0	0	0	0	0	1	0
S4	0	1	0	$G1 + G2 + G3$	SERR	1	0	1	0	0	0	1	0
SOK	1	0	0	$G1 + G2 + G3 + G4$	SOK	1	0	0	0	0	0	0	0
SOK	1	0	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	0	0	0	0	0	0	0	0
SERR	1	0	1	$G1 + G2 + G3 + G4$	SERR	1	0	1	0	0	0	0	1
SERR	1	0	1	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	0	0	0	0	0	0	0	1

Transition list using outputs as state variables

Current State							Next State					
S	*L1*	*L2*	*L3*	*L4*	*ERR*	*Transition Expression*	*S**	*L1**	*L2**	*L3**	*L4**	*ERR**
S1	1	0	0	0	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S2	0	1	0	0	0
S1	1	0	0	0	0	$G1 \cdot G2' \cdot G3' \cdot G4'$	SOK	0	0	0	0	0
S1	1	0	0	0	0	$G2 + G3 + G4$	SERR	0	0	0	0	1
S2	0	1	0	0	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S3	0	0	1	0	0
S2	0	1	0	0	0	$G1' \cdot G2 \cdot G3' \cdot G4'$	SOK	0	0	0	0	0
S2	0	1	0	0	0	$G1 + G3 + G4$	SERR	0	0	0	0	1
S3	0	0	1	0	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S4	0	0	0	1	0
S3	0	0	1	0	0	$G1' \cdot G2' \cdot G3 \cdot G4'$	SOK	0	0	0	0	0
S3	0	0	1	0	0	$G1 + G2 + G4$	SERR	0	0	0	0	1
S4	0	0	0	1	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	1	0	0	0	0
S4	0	0	0	1	0	$G1' \cdot G2' \cdot G3' \cdot G4$	SOK	0	0	0	0	0
S4	0	0	0	1	0	$G1 + G2 + G3$	SERR	0	0	0	0	1
SOK	0	0	0	0	0	$G1 + G2 + G3 + G4$	SOK	0	0	0	0	0
SOK	0	0	0	0	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	1	0	0	0	0
SERR	0	0	0	0	1	$G1 + G2 + G3 + G4$	SERR	0	0	0	0	1
SERR	0	0	0	0	1	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	1	0	0	0	0

Transition list using don't-care state codings

Current State							Next State					
S	*L1*	*L2*	*L3*	*L4*	*ERR*	*Transition Expression*	*S**	*L1**	*L2**	*L3**	*L4**	*ERR**
S	1	x	x	x	x	$G1' \cdot G2' \cdot G3' \cdot G4'$	S2	0	1	0	0	0
S	1	x	x	x	x	$G1 \cdot G2' \cdot G3' \cdot G4'$	SOK	0	0	0	0	0
S	1	x	x	x	x	$G2 + G3 + G4$	SERR	0	0	0	0	1
S2	0	1	x	x	x	$G1' \cdot G2' \cdot G3' \cdot G4'$	S3	0	0	1	0	0
S2	0	1	x	x	x	$G1' \cdot G2 \cdot G3' \cdot G4'$	SOK	0	0	0	0	0
S2	0	1	x	x	x	$G1 + G3 + G4$	SERR	0	0	0	0	1
S3	0	0	1	x	x	$G1' \cdot G2' \cdot G3' \cdot G4'$	S4	0	0	0	1	0
S3	0	0	1	x	x	$G1' \cdot G2' \cdot G3 \cdot G4'$	SOK	0	0	0	0	0
S3	0	0	1	x	x	$G1 + G2 + G4$	SERR	0	0	0	0	1
S4	0	0	0	1	x	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	1	0	0	0	0
S4	0	0	0	1	x	$G1' \cdot G2' \cdot G3' \cdot G4$	SOK	0	0	0	0	0
S4	0	0	0	1	x	$G1 + G2 + G3$	SERR	0	0	0	0	1
SOK	0	0	0	0	0	$G1 + G2 + G3 + G4$	SOK	0	0	0	0	0
SOK	0	0	0	0	0	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	1	0	0	0	0
SERR	0	0	0	0	1	$G1 + G2 + G3 + G4$	SERR	0	0	0	0	1
SERR	0	0	0	0	1	$G1' \cdot G2' \cdot G3' \cdot G4'$	S1	1	0	0	0	0