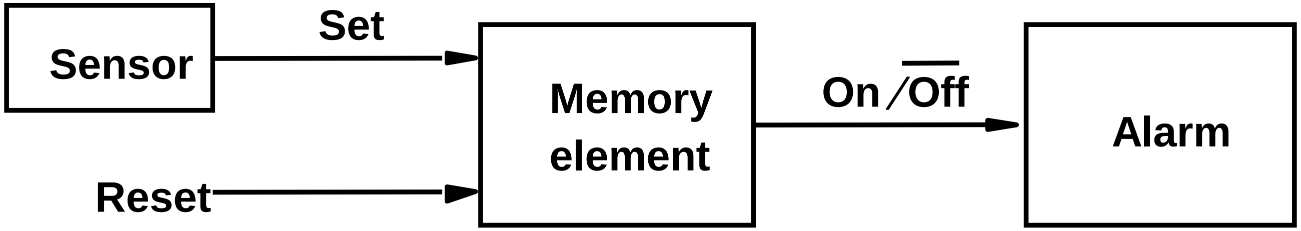
Lecture X
Final Review

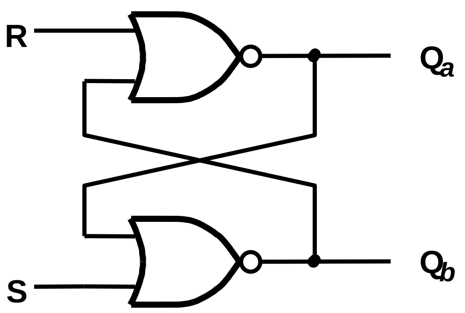
吳文中

“States” in a Circuit

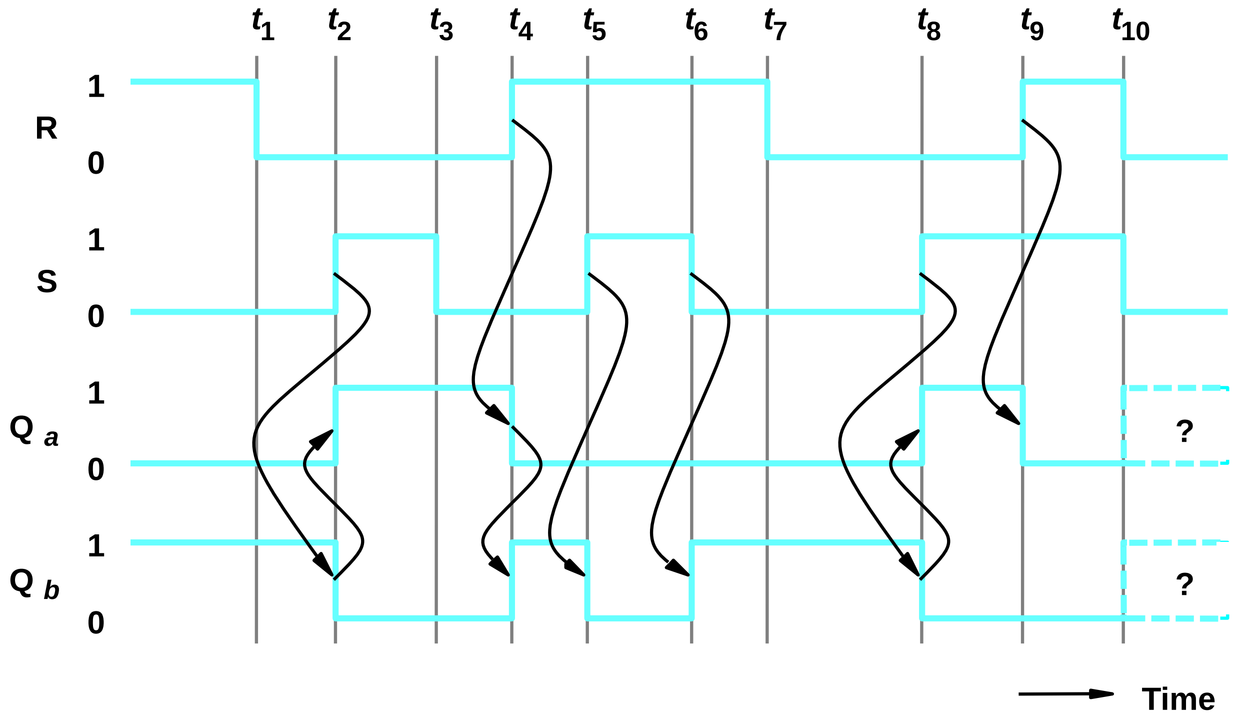


Control of an alarm system.

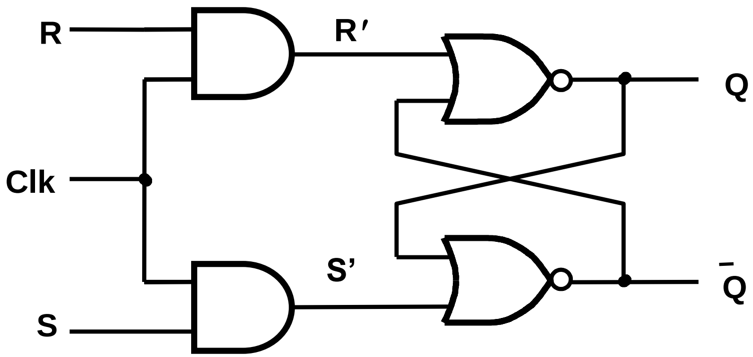
A Basic Latch Built with NOR Gates



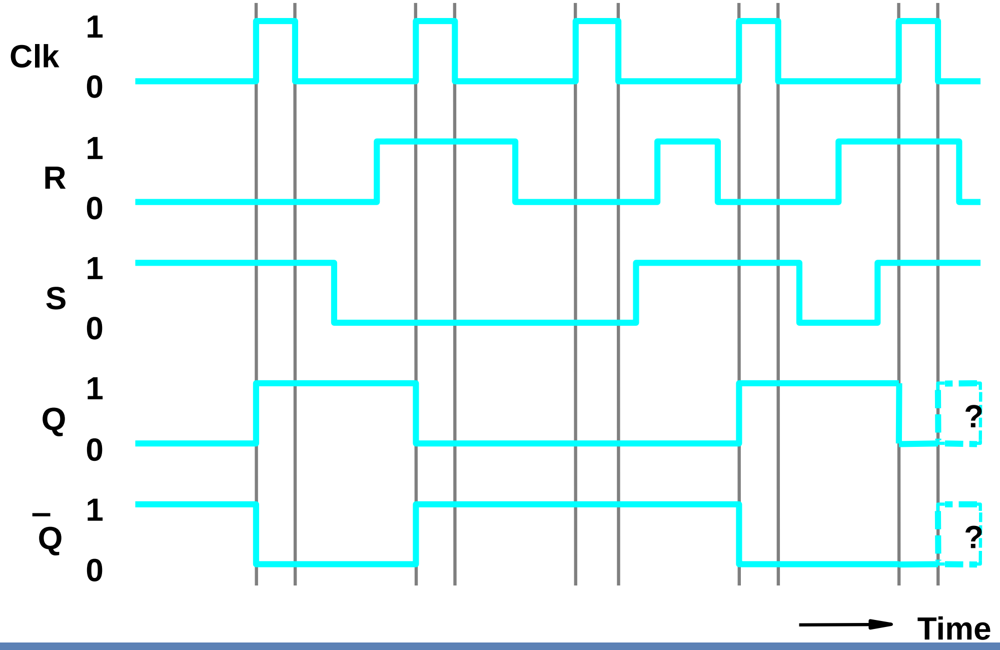
S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

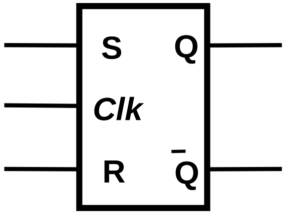


Gated SR Latch

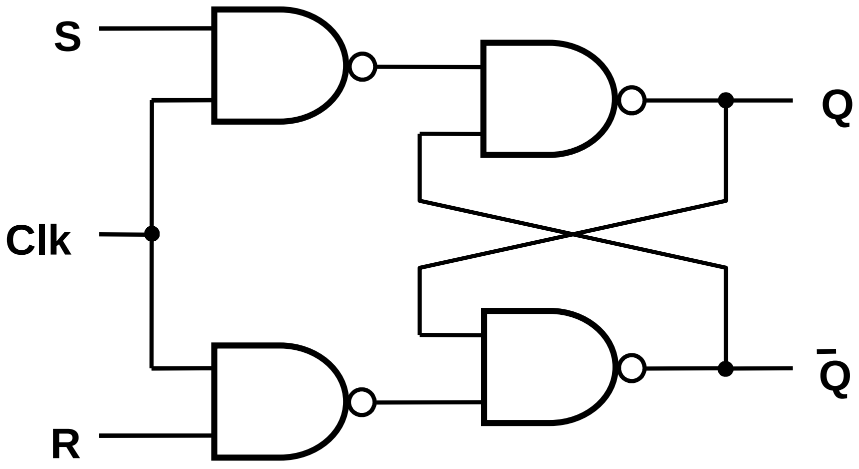


Clk	S	R	$Q(t+1)$
0	x	x	Q(*t*) (no change)
1	0	0	Q(*t*) (no change)
1	0	1	0
1	1	0	1
1	1	1	x



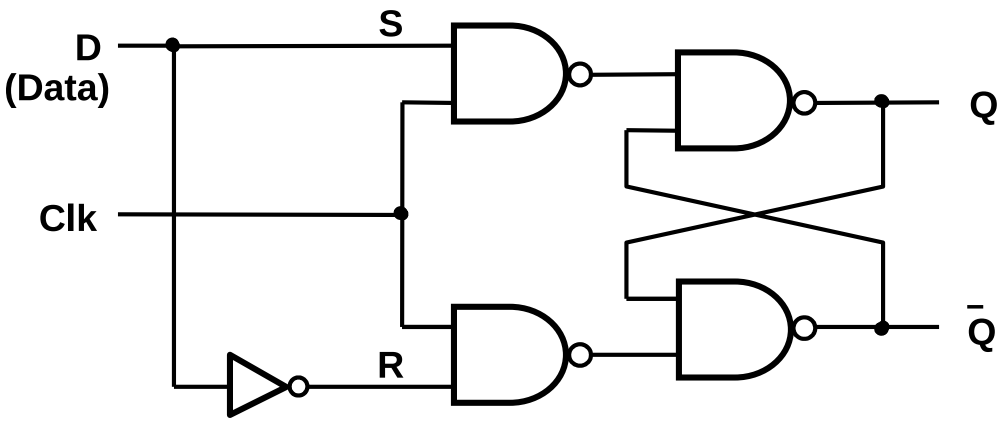


Gated SR Latch with NAND Gates

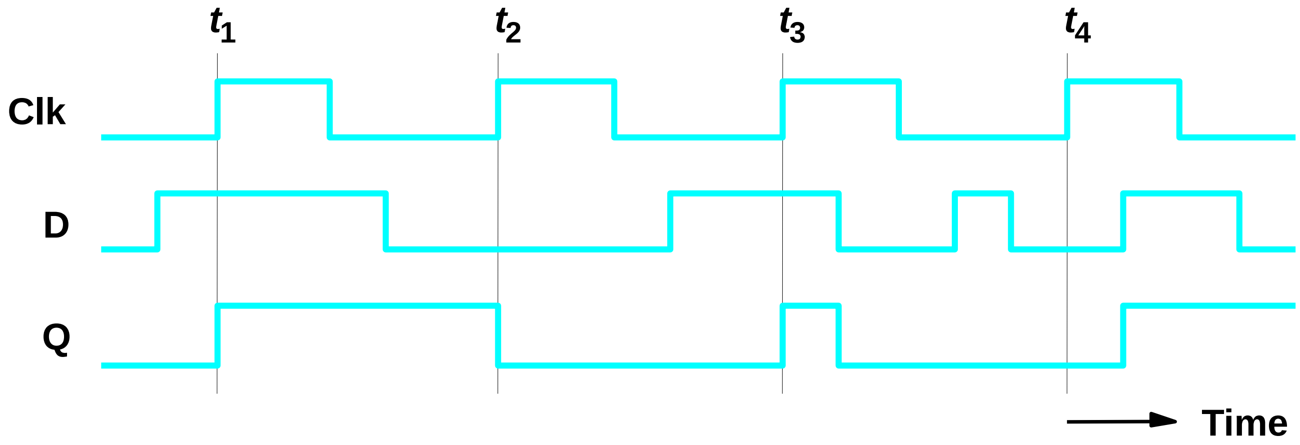


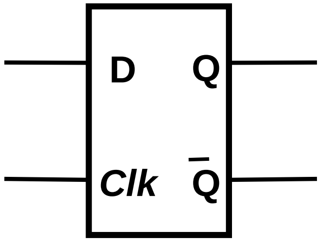
Clk	S	R	$Q(t+1)$
0	x	x	Q(*t*) (no change)
1	0	0	Q(*t*) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

Gated D Latch

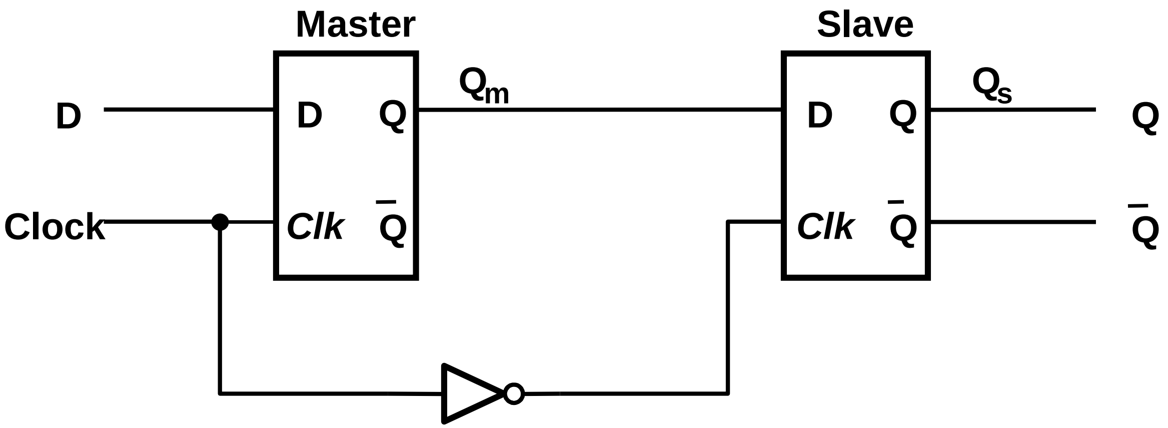


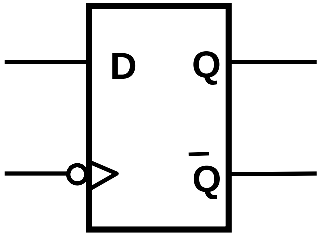
Clk	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1

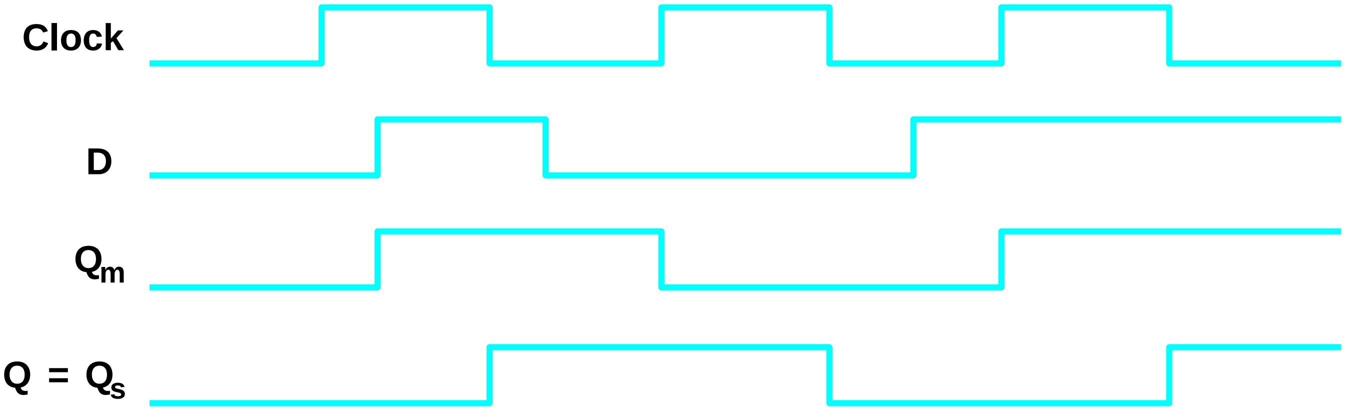




Master-Slave D Flip-Flop







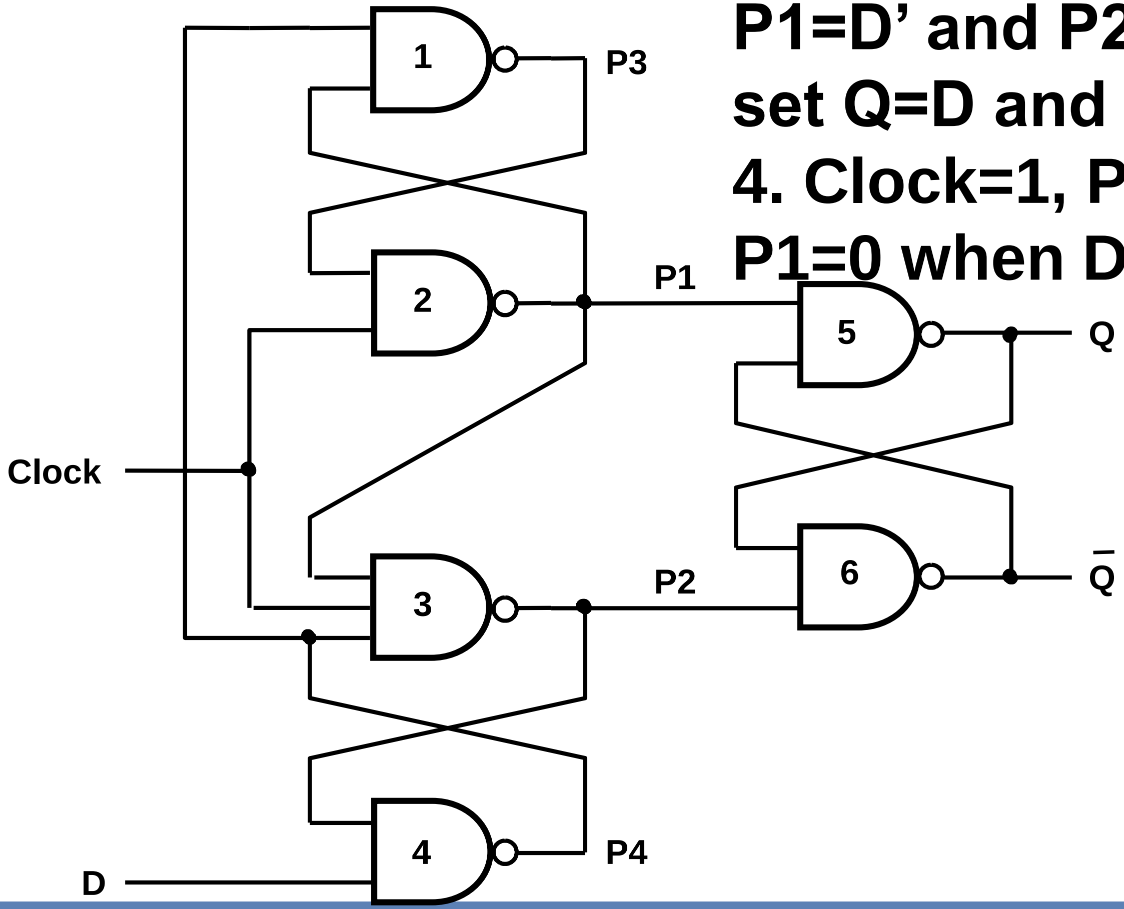
A Positive-edge-triggered D Flip-Flop

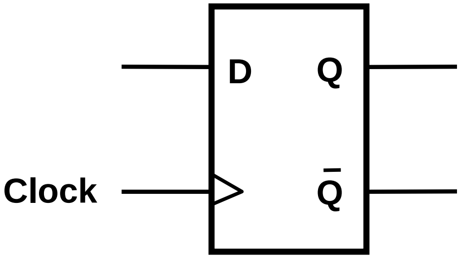
1.Clock=0, P1=P2=1; Latch Hold.

2. P4=D’ and P3 = D

3. Clock 0 to 1 (positive edge), P1=D’ and P2=D which set Q=D and Q’=D’

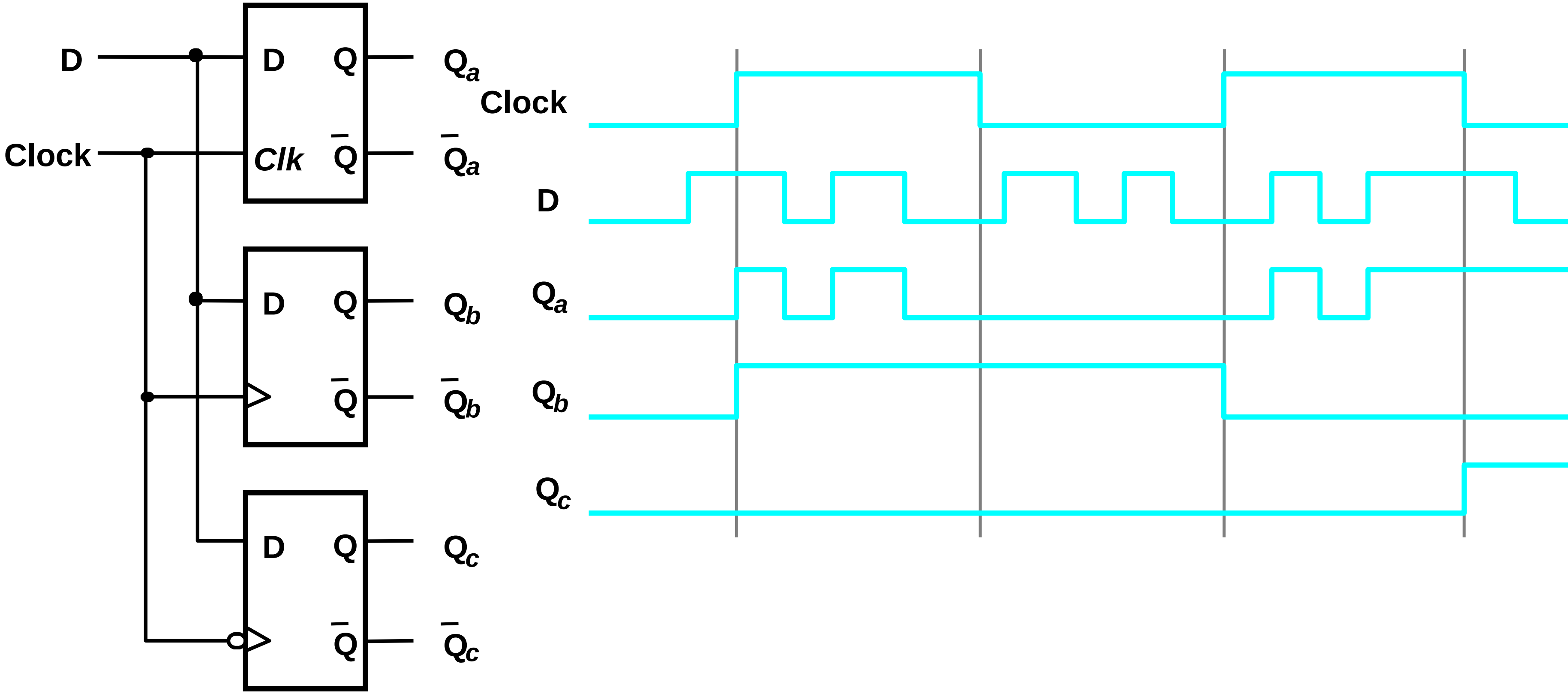
4. Clock=1, P2=0 when D=0; P1=0 when D=1



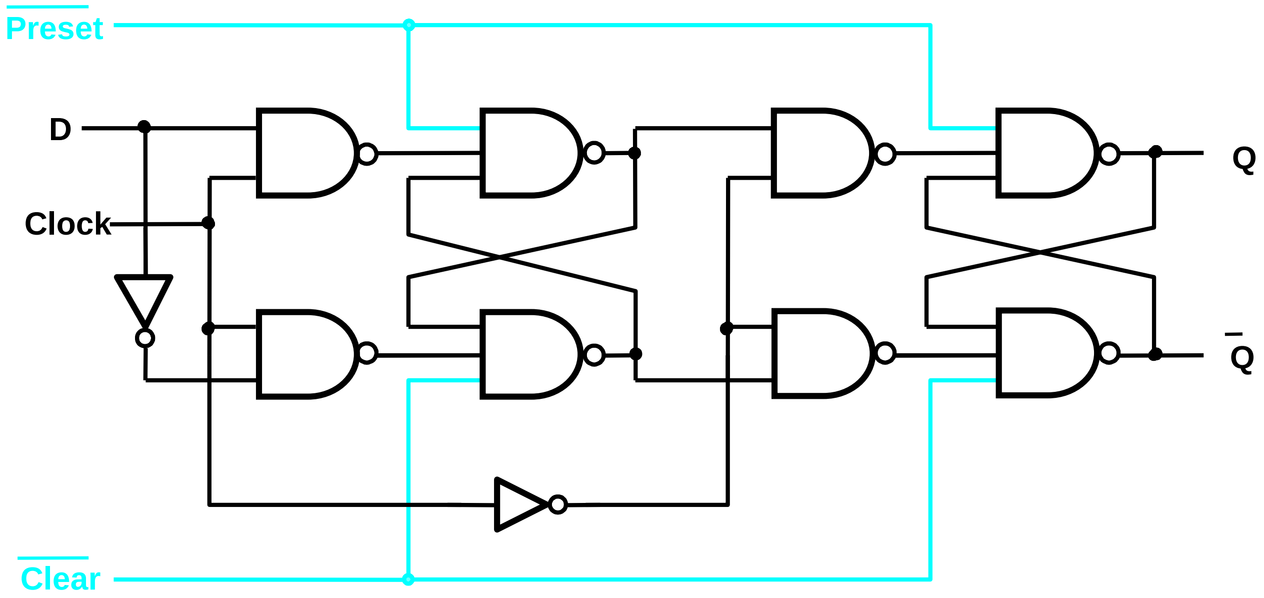


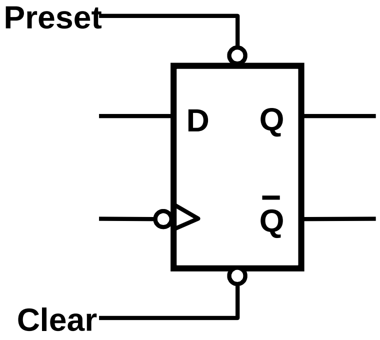
(b) Graphical symbol

Level-Sensitive versus Edge-Triggered

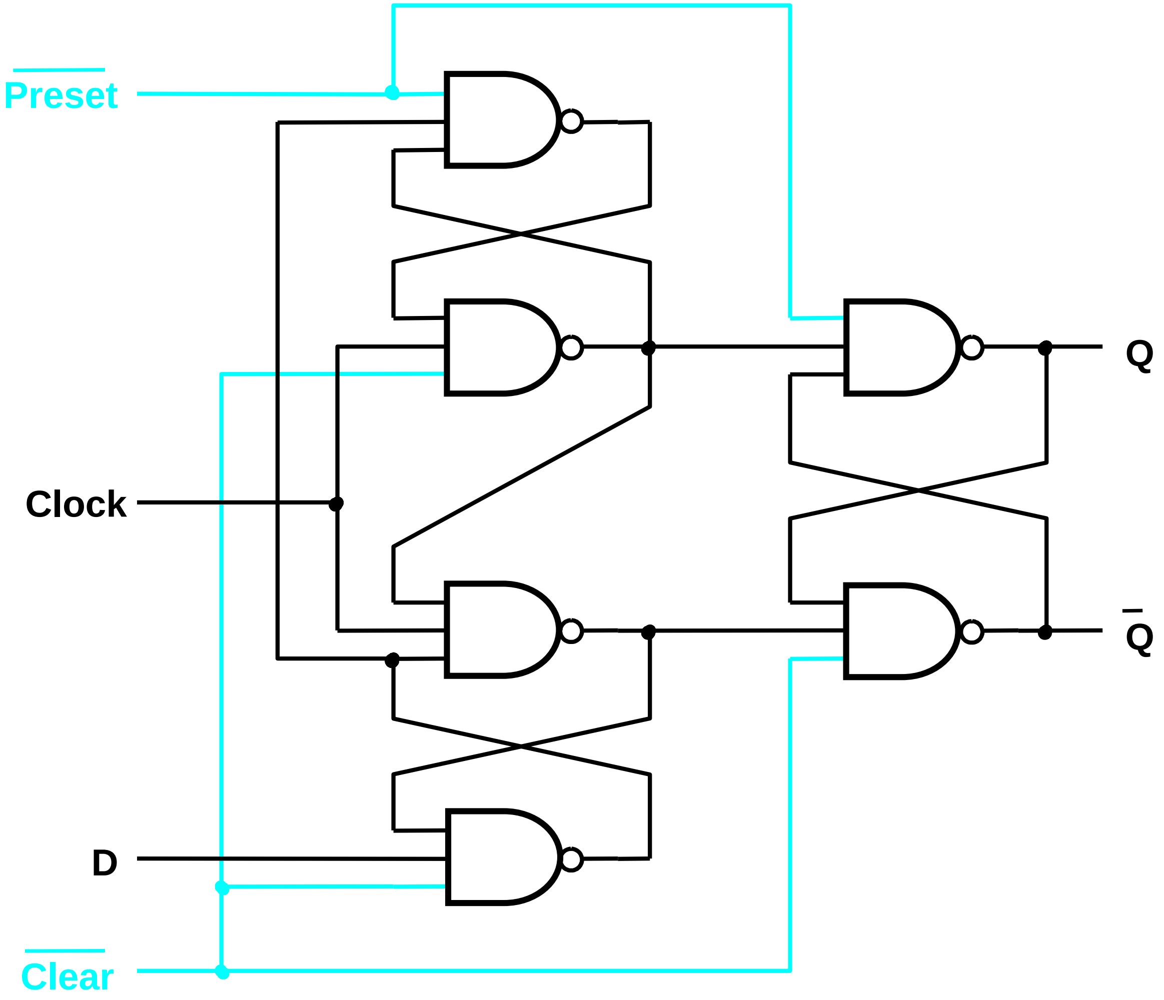


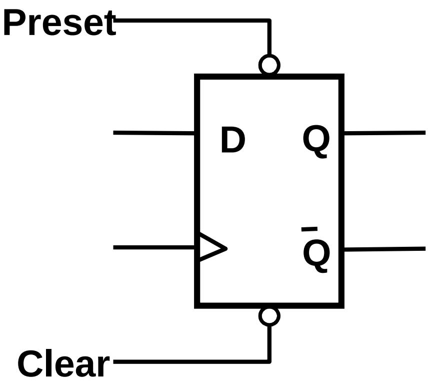
Master-Slave D Flip-flop with Clear and Preset

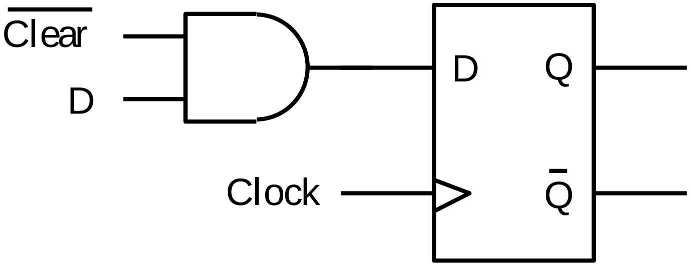




Positive-edge-trigger D Flip-flop with Clear and Preset

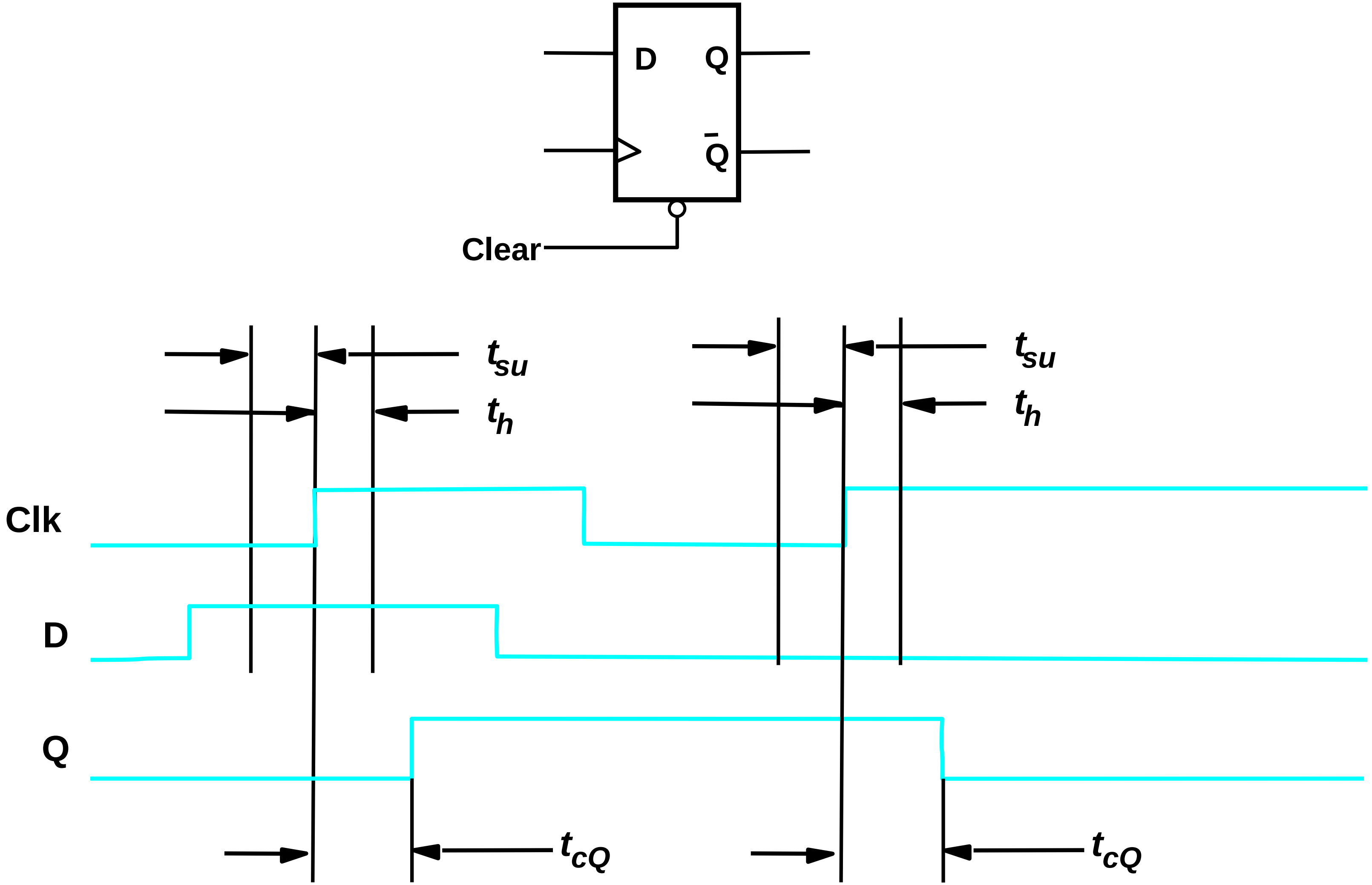




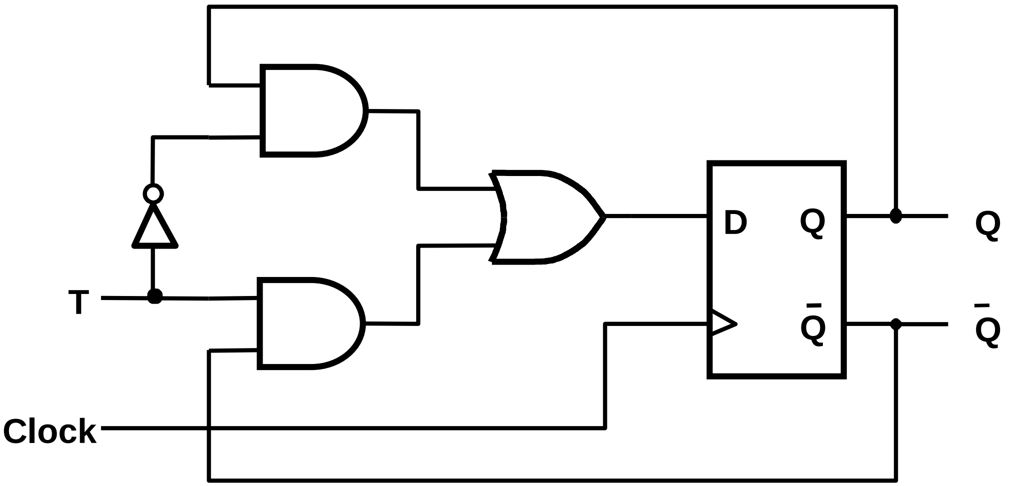


Adding a synchronous clear

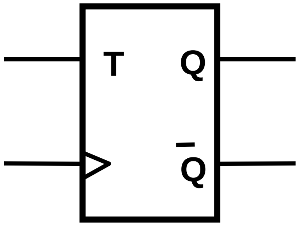
Flip-Flop Timing Parameter

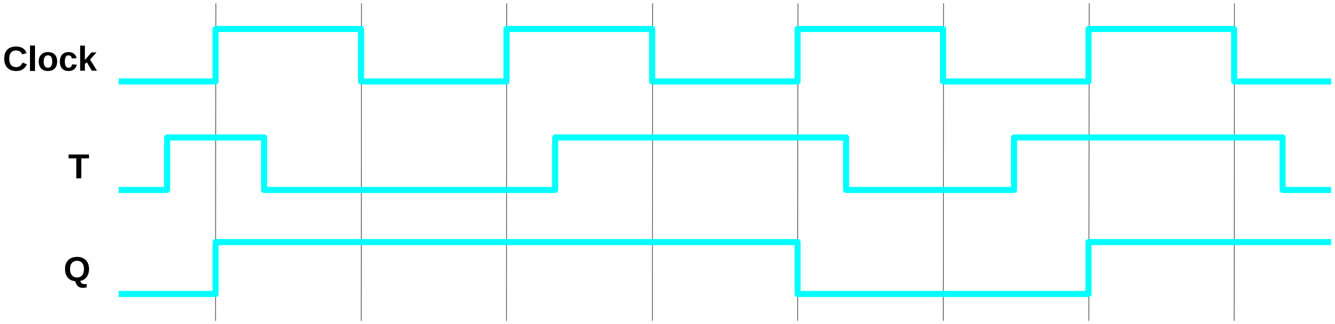


T Flip-Flop



T	$Q(t+1)$
0	$Q(t)$
1	$\overline{Q}(t)$

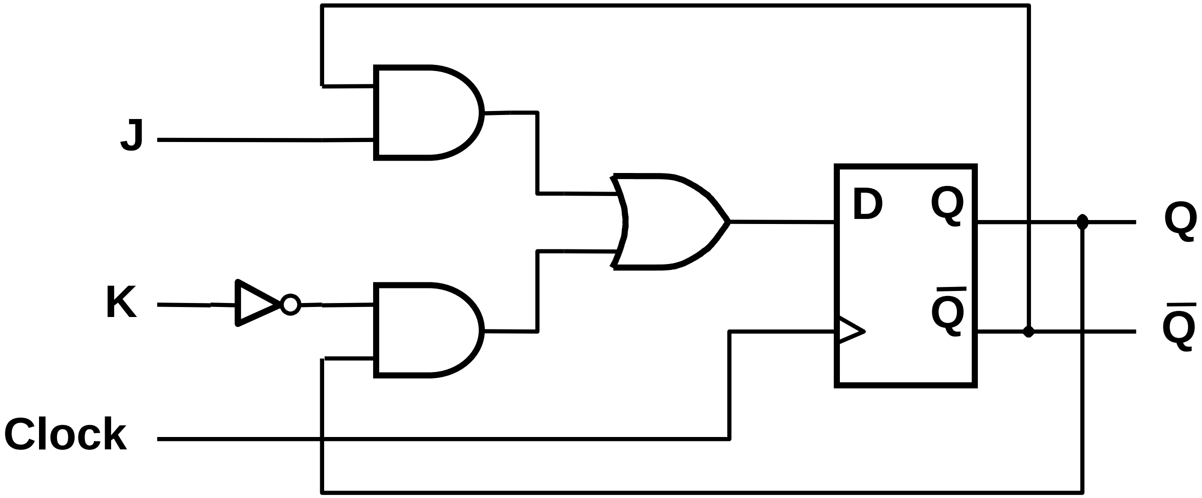




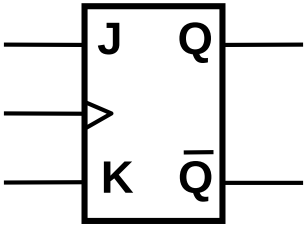
Configurable Flip-Flops

- In general purpose chips like PLDs, the flip-flops that are provided are sometimes *configurable*, which means that a flip-flop circuit can be configured to be either D, T, or some type.
- HOW?

JK Flip-Flop



J	K	Q (t+1)
0	0	Q (t)
0	1	0
1	0	1
1	1	$\overline{Q}$ (t)



Summary of Terminology

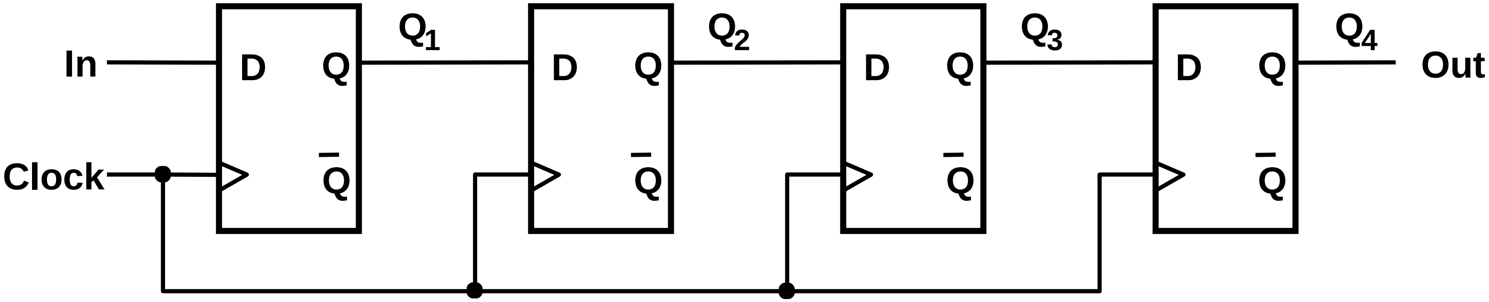
- **Basic latch** is a feedback connection of two NOR gates or two NAND gates, which can store one bit of information. It can be set to 1 using S or 0 using R.
- **Gated latch** is a basic latch that includes input gating and a control input signal.
  - **Gated SR latch** uses the S and R inputs to set the latch to 1 or reset to 0 (do have R=S=0 exception).
  - **Gated D latch** uses the D input to force the latch into a state that has the same lotic value as the D input

- A **flip-flop** is a storage element based on the bated latch principle, which can have its output sate changed only on the edge of the controlling clock signals.
  - **Edge-triggerd flip-flop** is affected only by the input values present when the active edge of the clock occurs.
  - **Master-slave flip-flop** is built with two gated latches. The master stage is active during half of the clock cycle, and the slave stage is active during the other half. The output value of the flip-flop changes on the edge of the clock that activates the transfer into the slave stage. It can be edge-triggered or level sensitive. (Gated D latch=>edge-triggered; Gates SR latch=> level-sensitive)

Registers

- A flip-flop stores one bit of information.
- When a **set** of *n* flip-flops is used to store *n* bits of information, such as an *n*-bit number, we refer to these flip-flops as a *register*.
- A common clock is used for each flip-flop in a register.

Shift Register

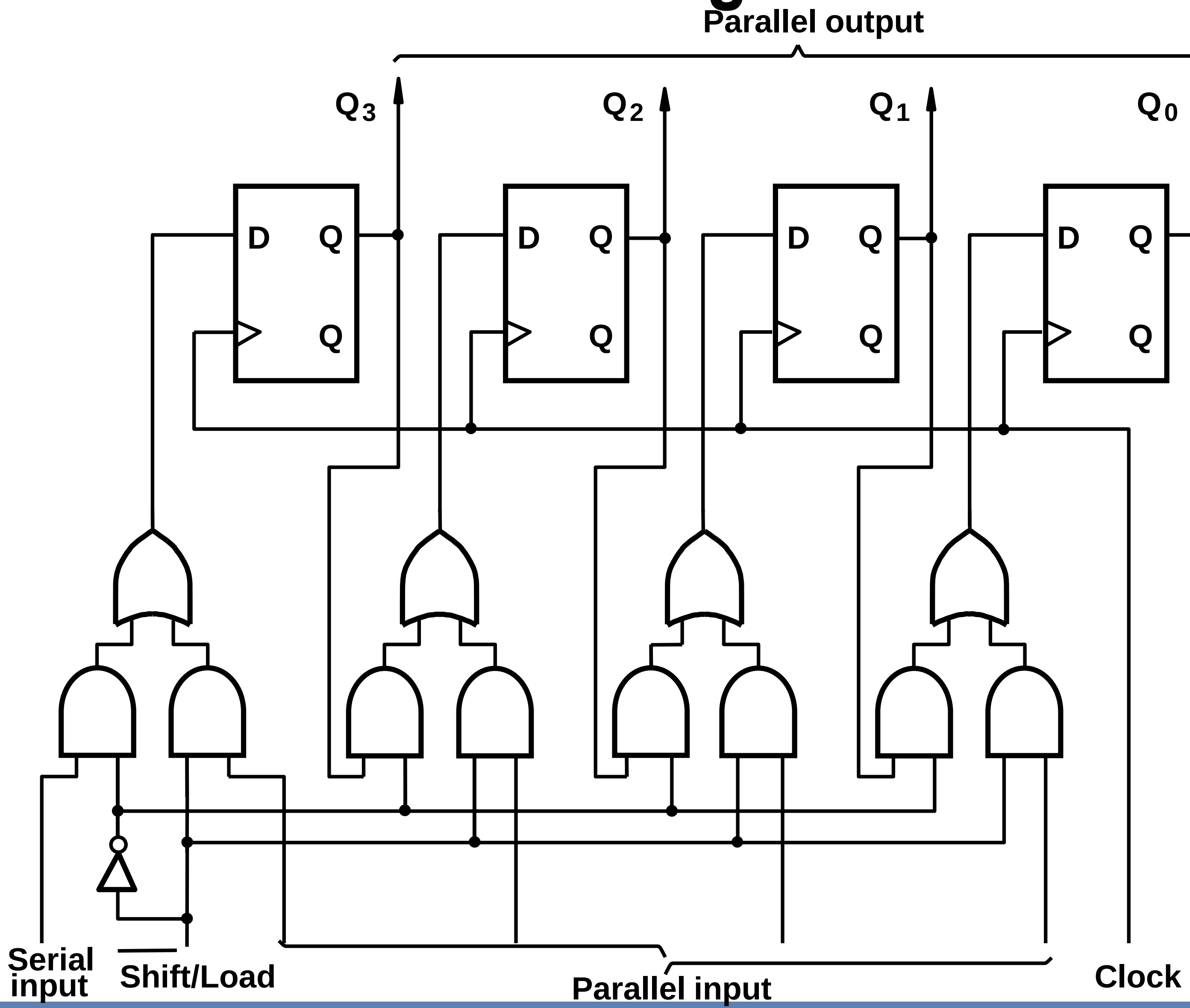


	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

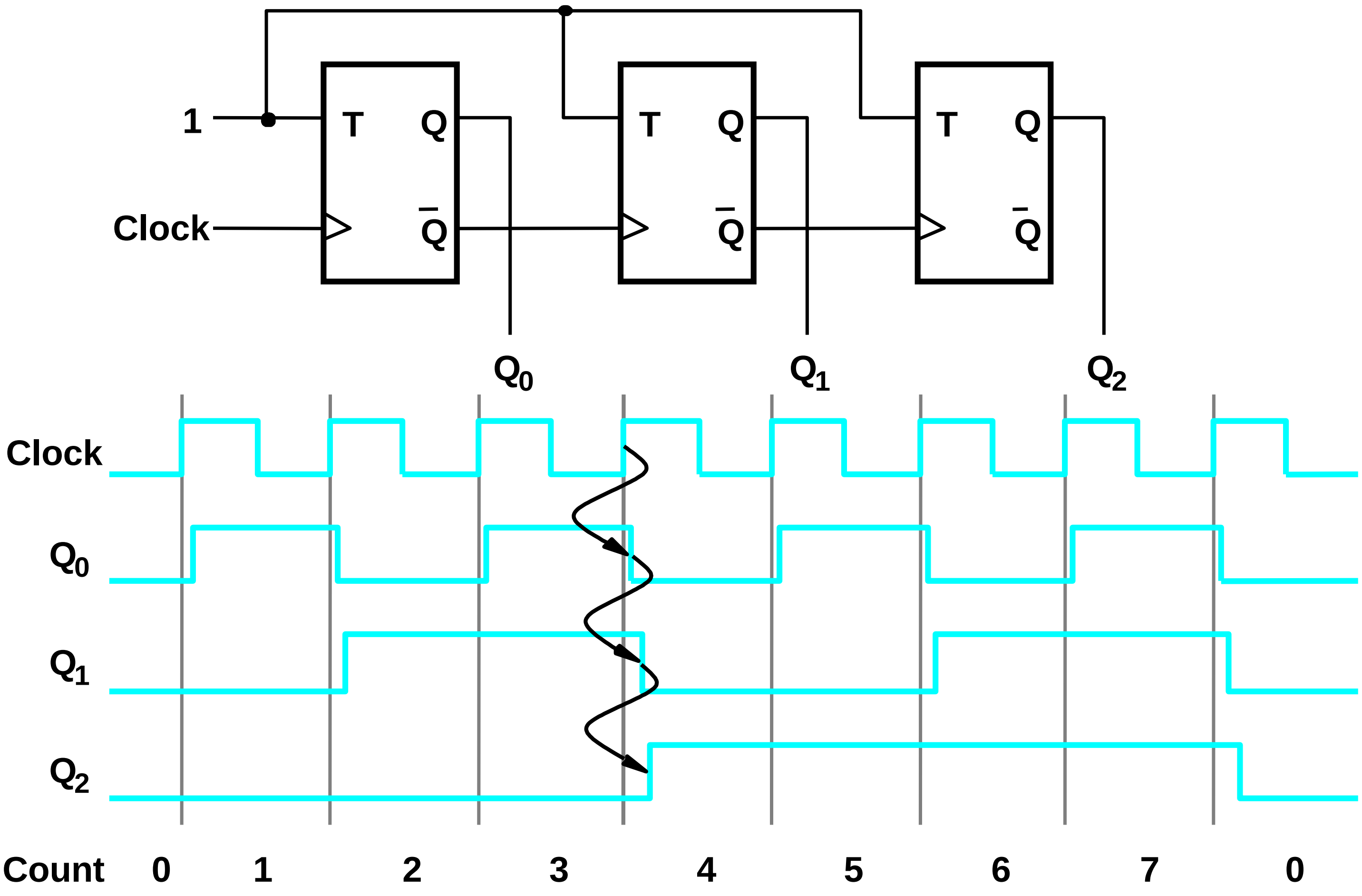
Parallel versus Serial

- Transmitting n bits at once using n separate wires. We say this scheme is **parallel transfer**.
- Transmitting n bits at once using a single wire, by performing the transfer one bit at a time, in n consecutive clock cycles. We say this scheme is **serial transfer**.

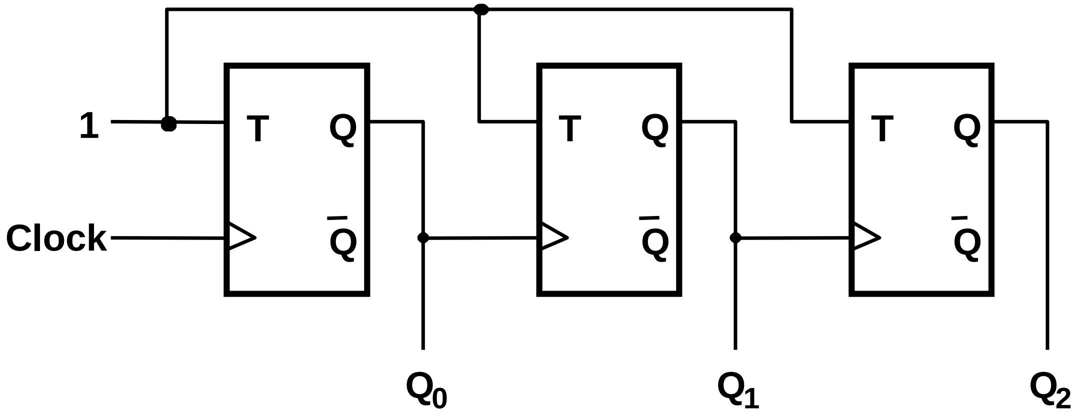
Parallel-Access Shift Register

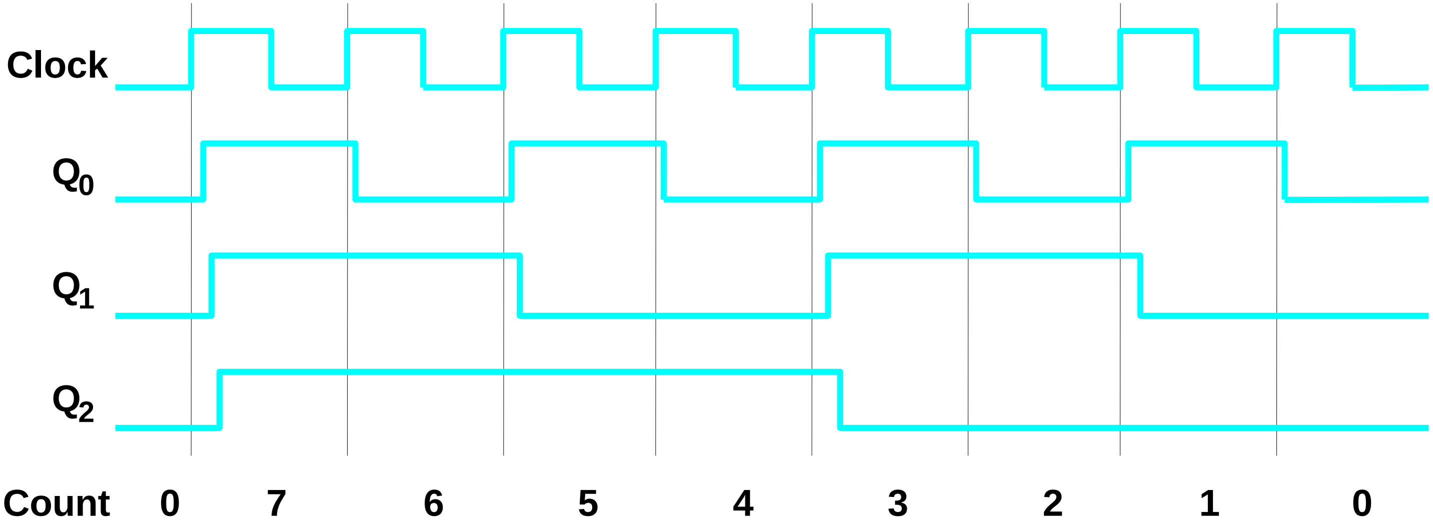


Asynchronous Up-Counter with T Flip-flops



Asynchronous Down-Counter with T Flip-flops

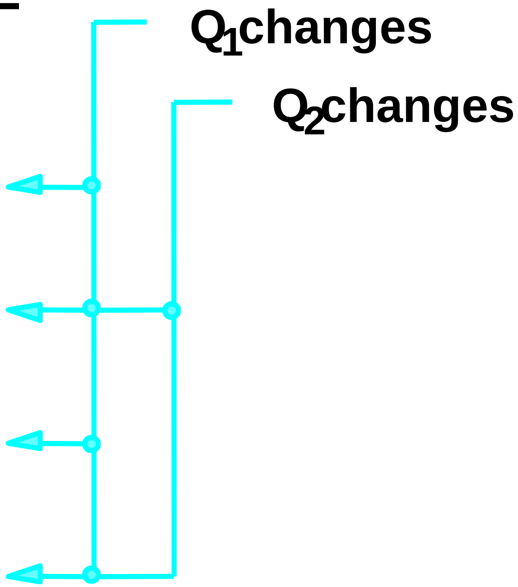




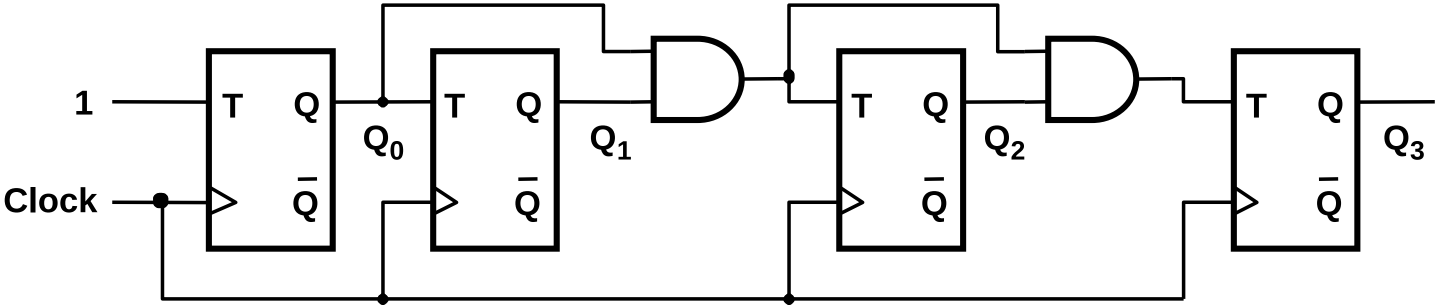
Synchronous Counter with T Flip-flops

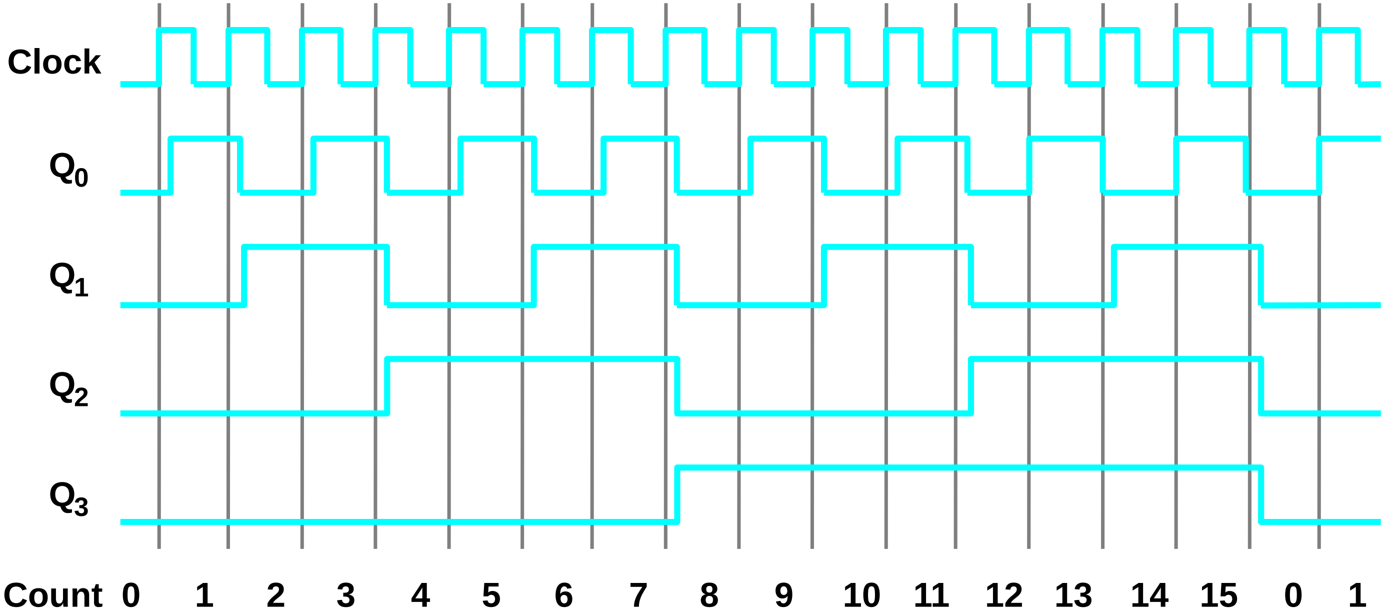
- $T_0=1$
- $T_1=Q_0$
- $T_2=Q_0Q_1$
- $T_3=Q_0Q_1Q_2$
- $T_n=Q_0Q_1\ldots.Q_{n-1}$

Clock cycle	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

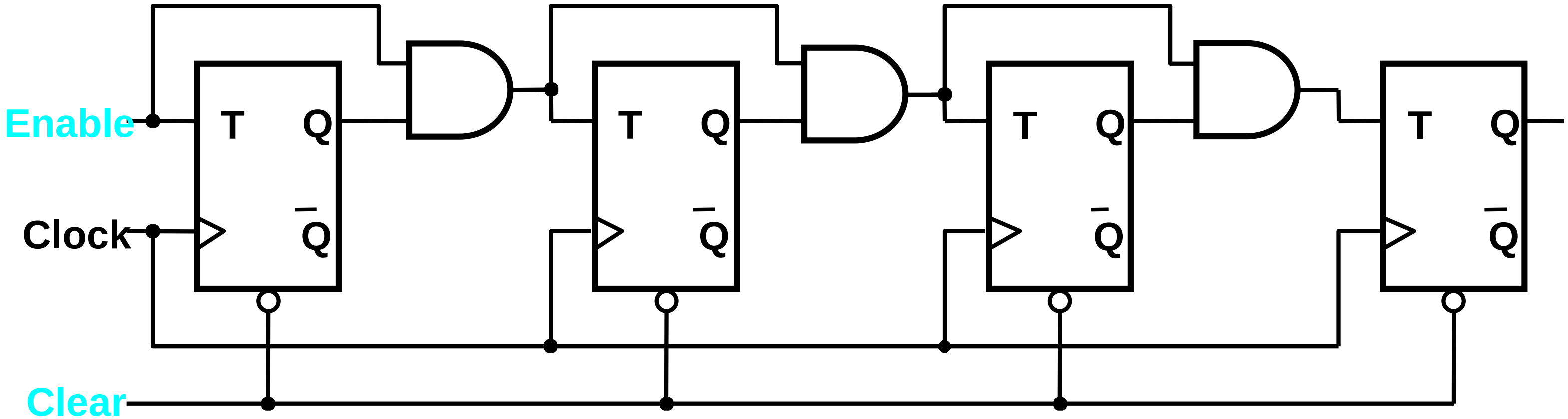


Synchronous Counter with T Flip-flops





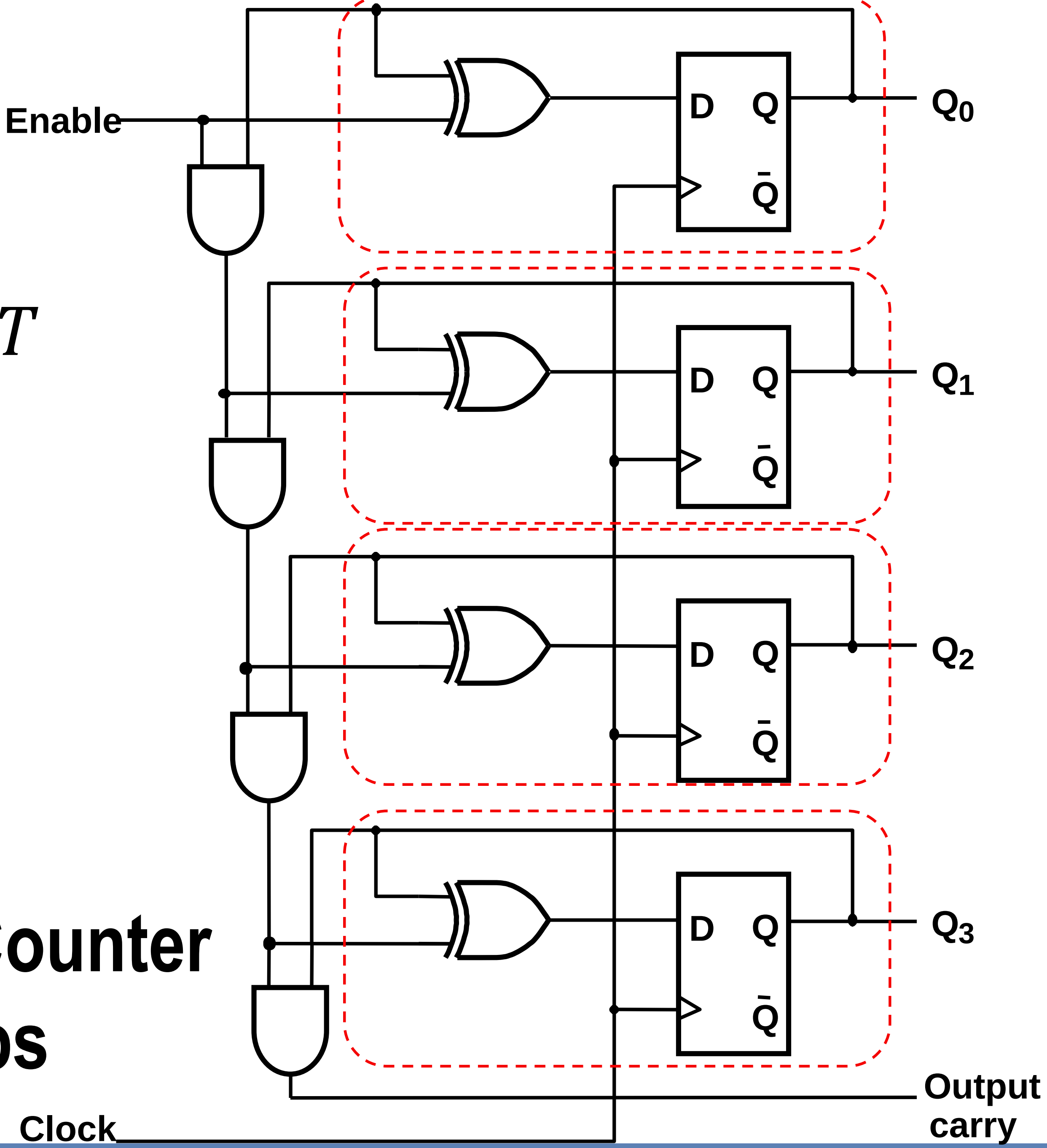
Enable and Clear



Synchronous Counter with D Flip-flops

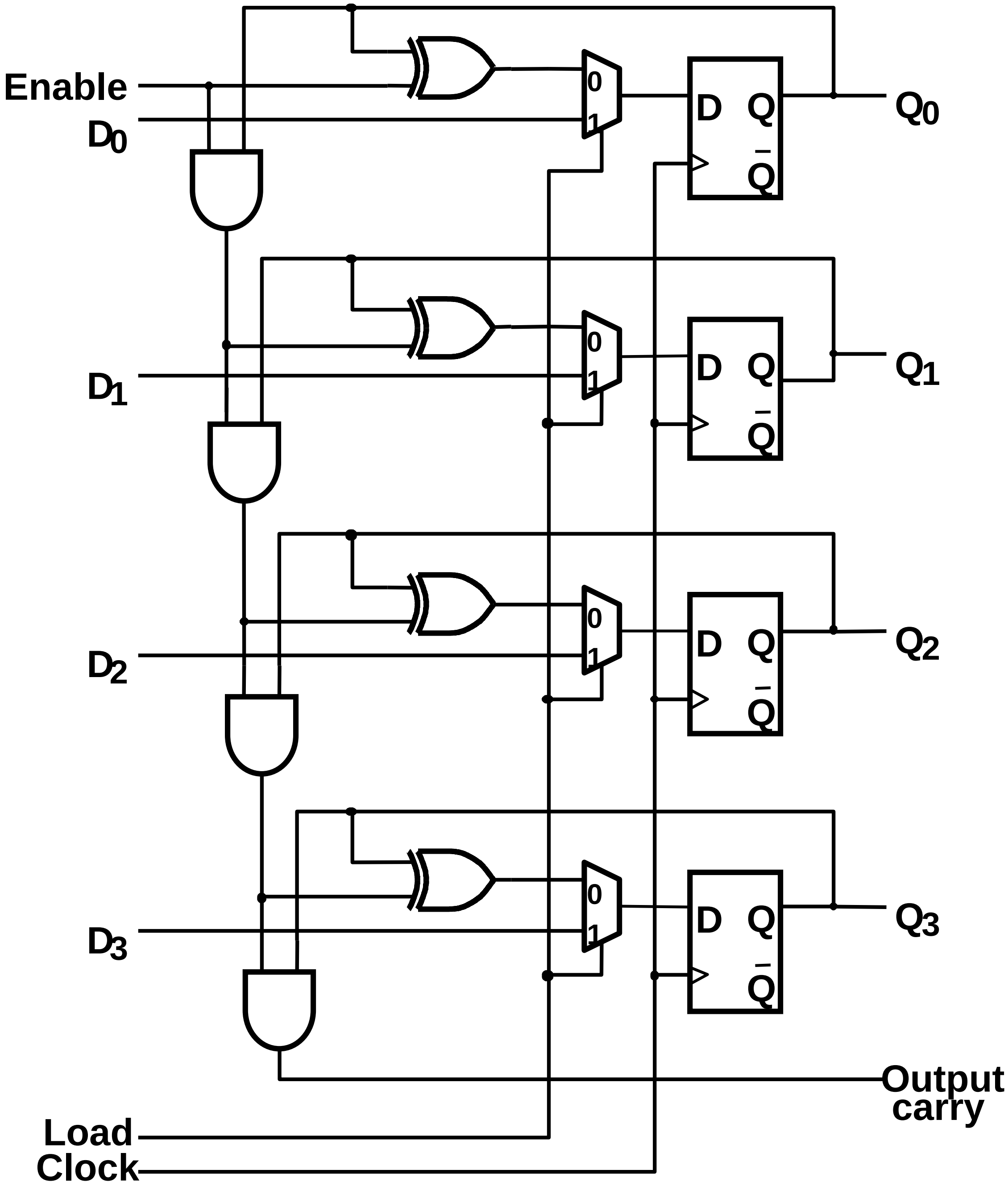
- $D_0 = \overline{Q_0} = 1 \oplus Q_0 (= Q_0 \oplus Enable)$
- $D_1 = Q_1 \oplus Q_0 (\cdot Enable)$
- $D_2 = Q_2 \oplus Q_1 Q_0 (\cdot Enable)$
- $D_3 = Q_3 \oplus Q_2 Q_1 Q_0 (\cdot Enable)$
- $D_i = Q_i \oplus Q_{i-1} Q_{i-2} \ldots . Q_1 Q_0 (\cdot Enable)$

- $D = Q\bar{T} + \bar{Q}T$
  $= Q \oplus T$



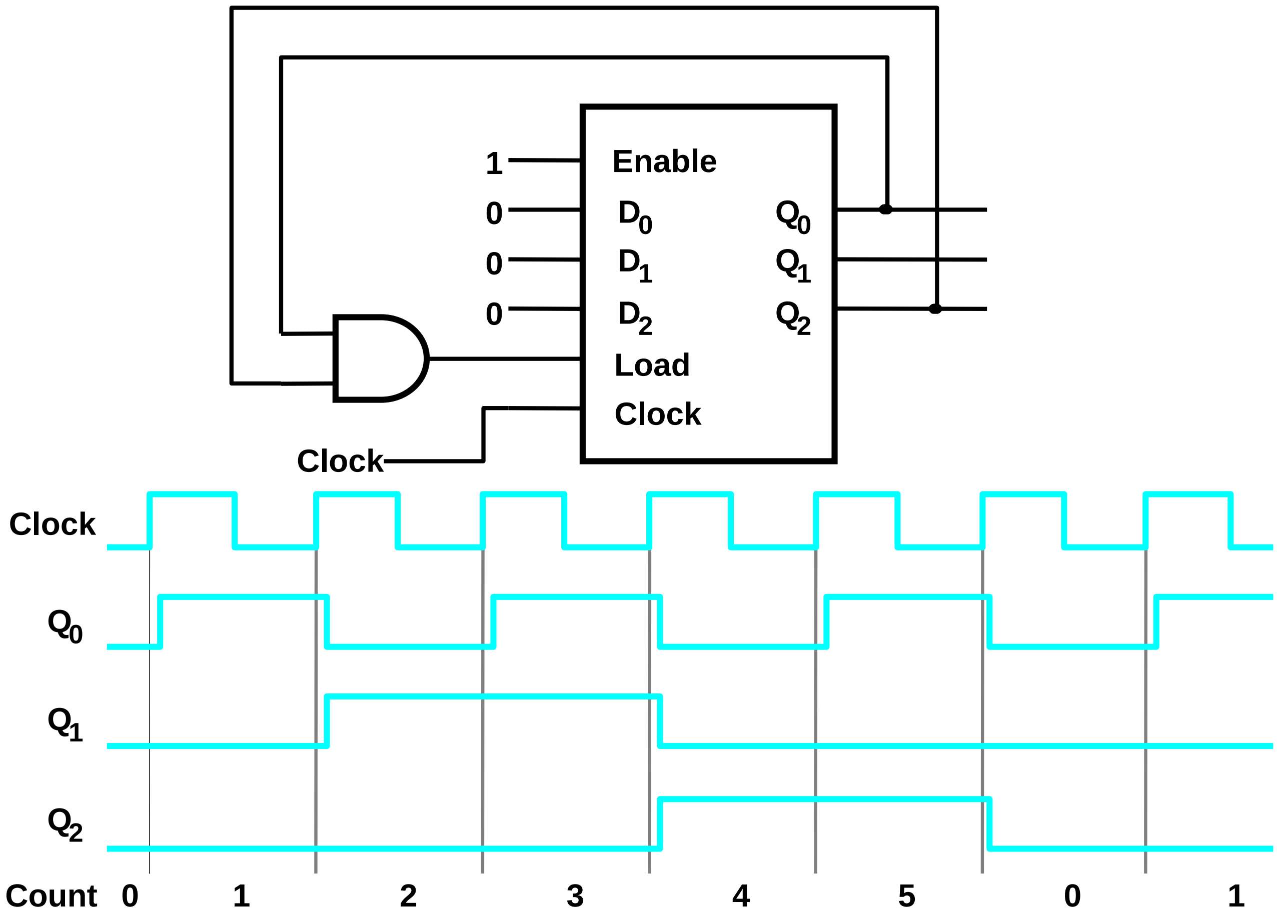
Synchronous Counter with D Flip-flops

Counter with Parallel-load



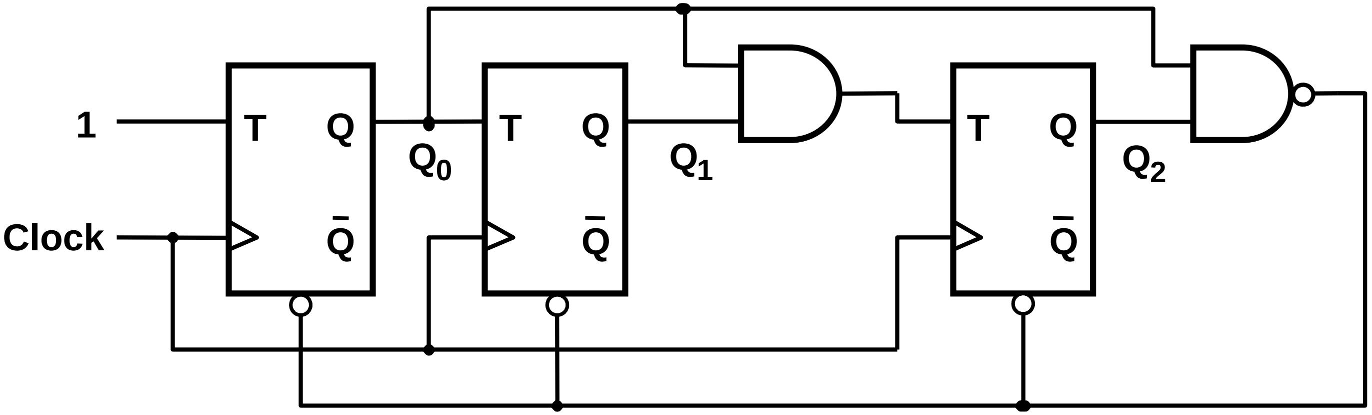
A Modulo-6 Counter with Synchronous Reset

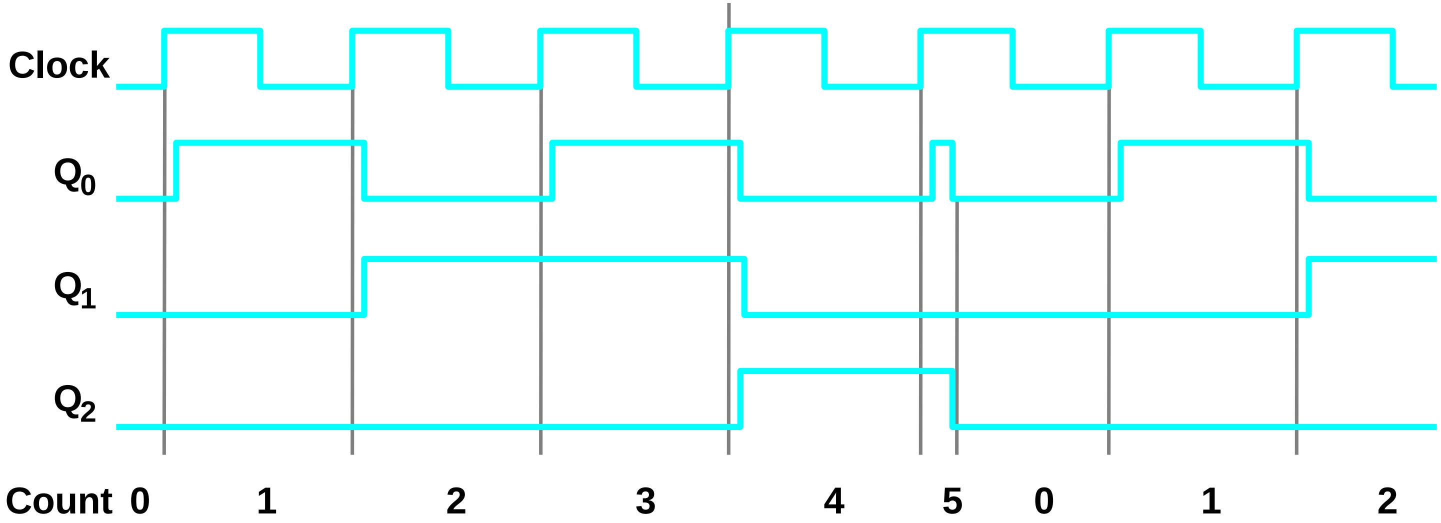
- Counting sequence: 0,1,2,3,4,5,0,1, and so on.



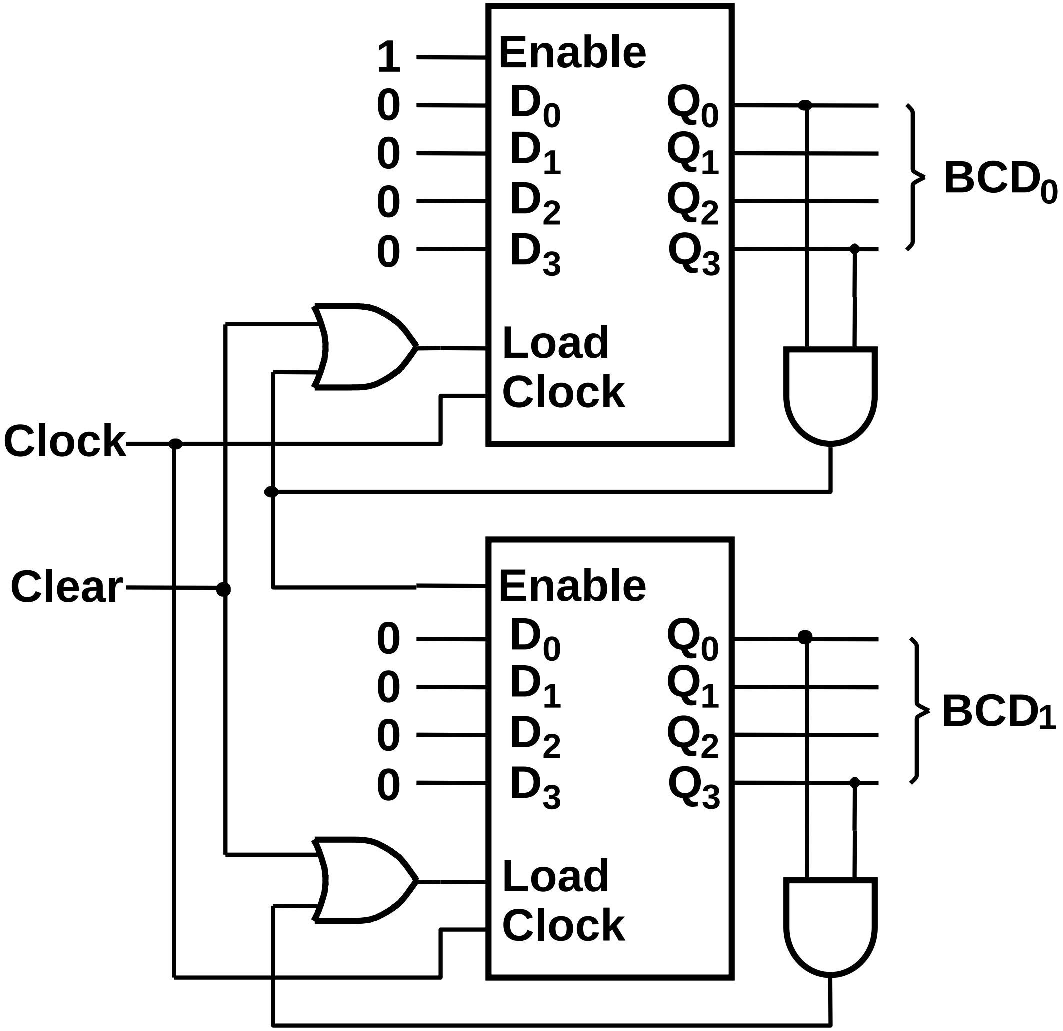
A Modulo-6 Counter with Asynchronous Reset

- Modulo 5 or 6?

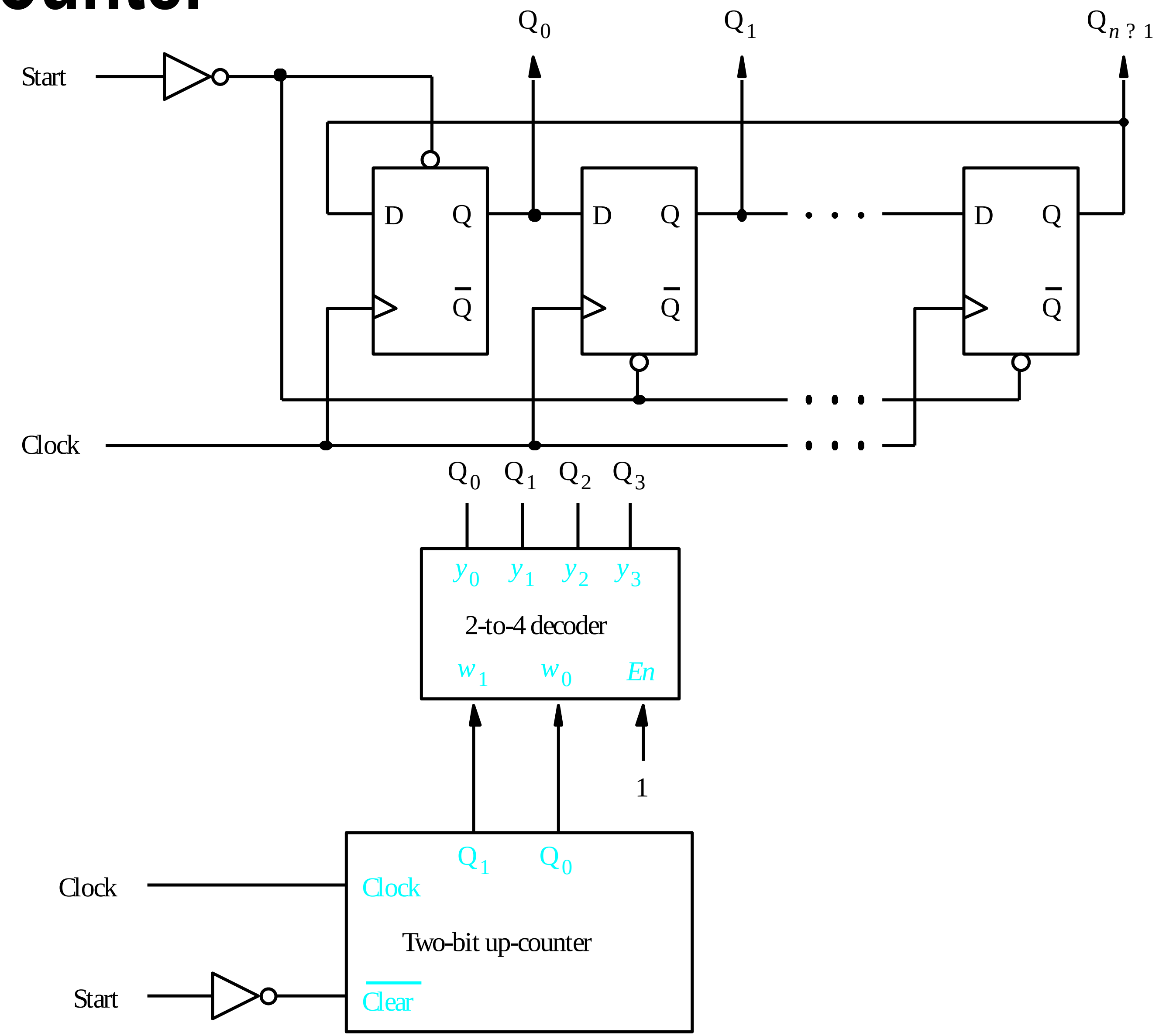




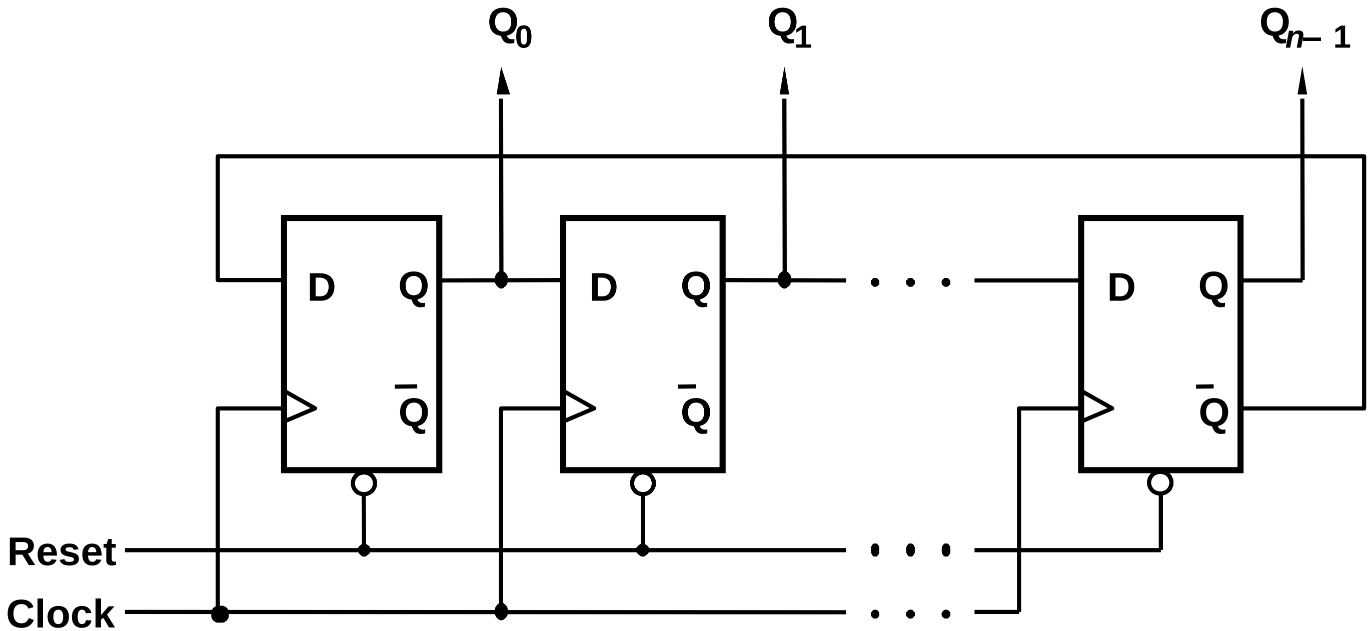
BCD Counter


1
0
0
0
0
Enable
D_0
D_1
D_2
D_3
Q_0
Q_1
Q_2
Q_3
BCD_0
Load
Clock
Clock
Clear
Enable
0
0
0
0
D_0
D_1
D_2
D_3
Q_0
Q_1
Q_2
Q_3
BCD_1
Load
Clock

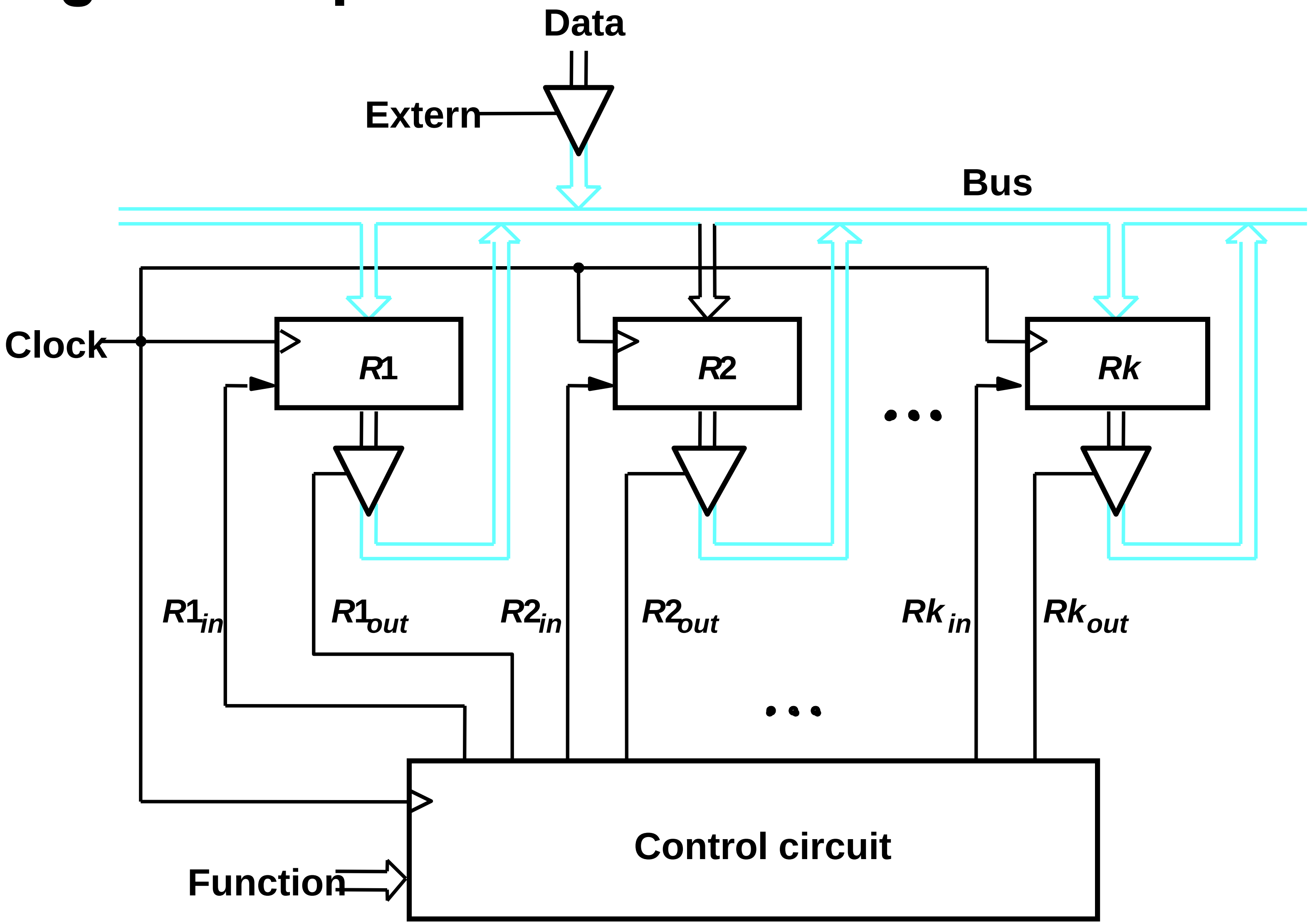
Ring Counter



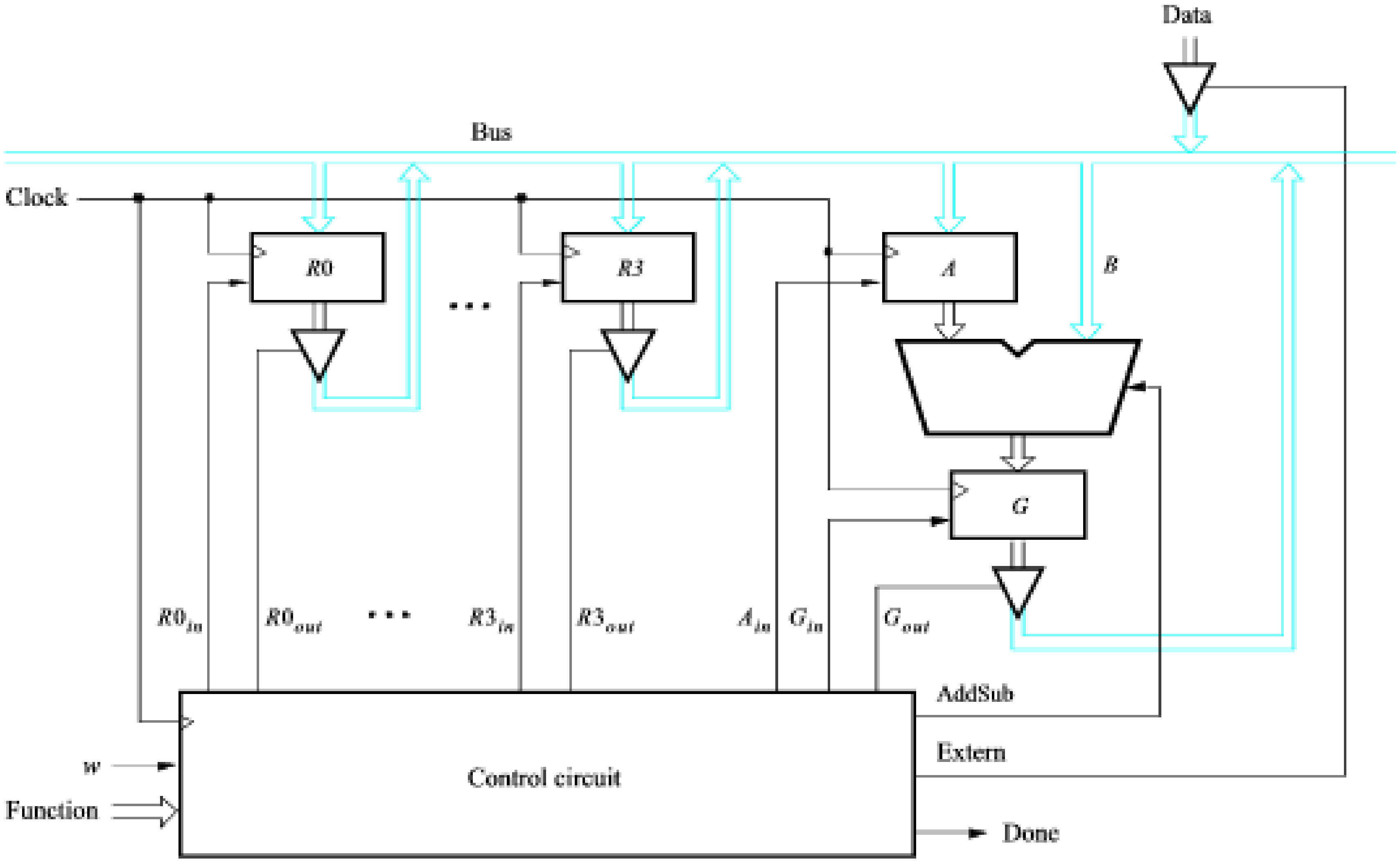
Johnson Counter



Design Example: Bus Structure



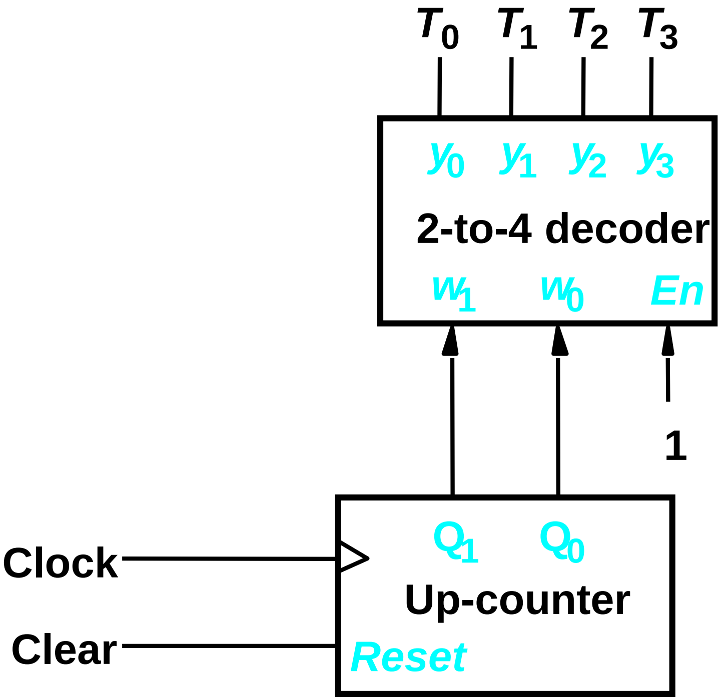
Simple Processor



Operation Performed in the Processor

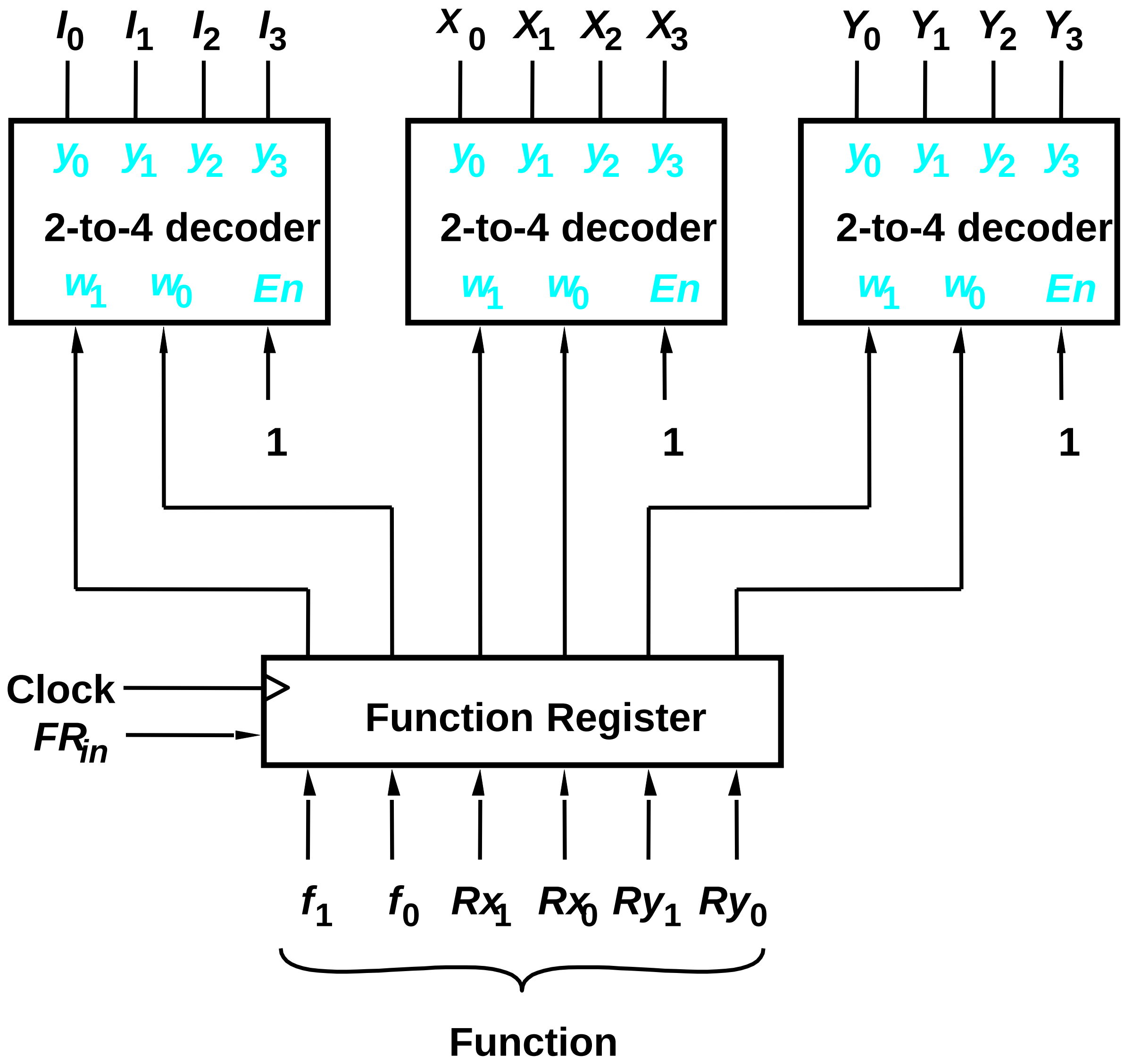
Operation	Function performed
Load Rx,$Data$	$Rx \leftarrow Data$
Move Rx, Ry	$Rx \leftarrow [Ry]$
Add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
Sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Control Circuit

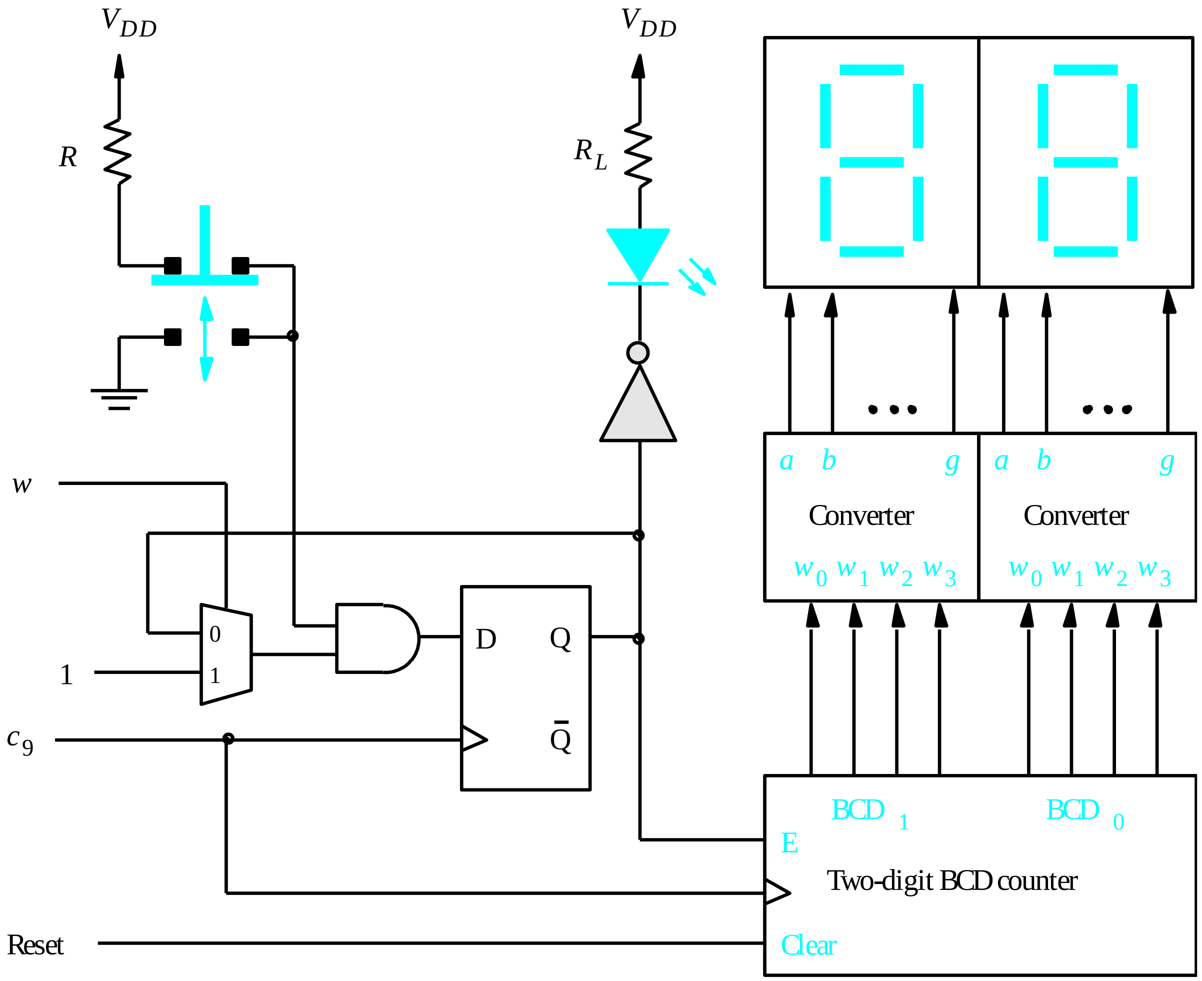


	T_1	T_2	T_3
(Load): I_0	$Extern$, $R_{in} = X$, $Done$		
(Move): I_1	$R_{in} = X$, $R_{out} = Y$, $Done$		
(Add): I_2	$R_{out} = X$, A_{in}	$R_{out} = Y$, G_{in}, $AddSub = 0$	G_{out}, $R_{in} = X$, $Done$
(Sub): I_3	$R_{out} = X$, A_{in}	$R_{out} = Y$, G_{in}, $AddSub = 1$	G_{out}, $R_{in} = X$, $Done$

Control Circuit

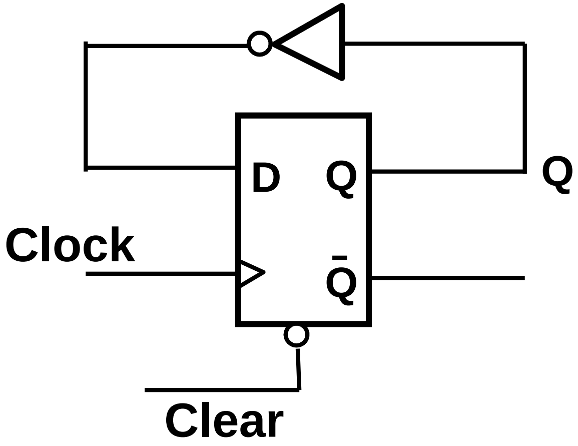


Reaction Timer



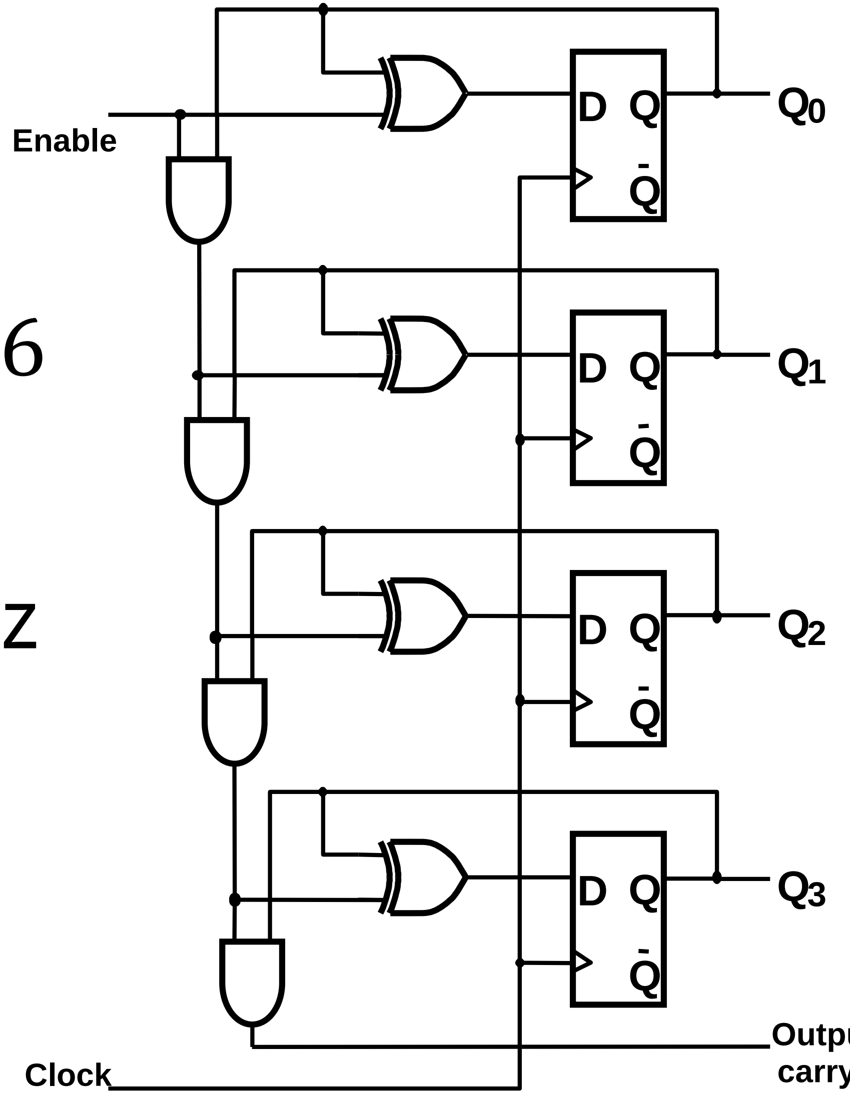
Timing Analysis

- $T_{min} = t_{cQ} + t_{NOT} + t_{su}$
- $F_{max} = \frac{1}{T_{min}}$
- A simple flip-flop circuit
  - $T_{min} = 1.0 + 1.1 + 0.6 = 2.7ns$
  - $F_{max} = \frac{1}{2.7ns} = 370.37\text{MHz}$



Timing Analysis for a 4-bit Counter

- $T_{min} = t_{cQ} + 3(t_{AND})$
  $+ \mathrm{t_{XOR}} + t_{su}$
  $= 1.0 + 3(1.2) + 1.2 + 0.6$
  $= 6.4ns$
- $F_{max} = \dfrac{1}{6.4ns} = 156.25\ \mathrm{MHz}$

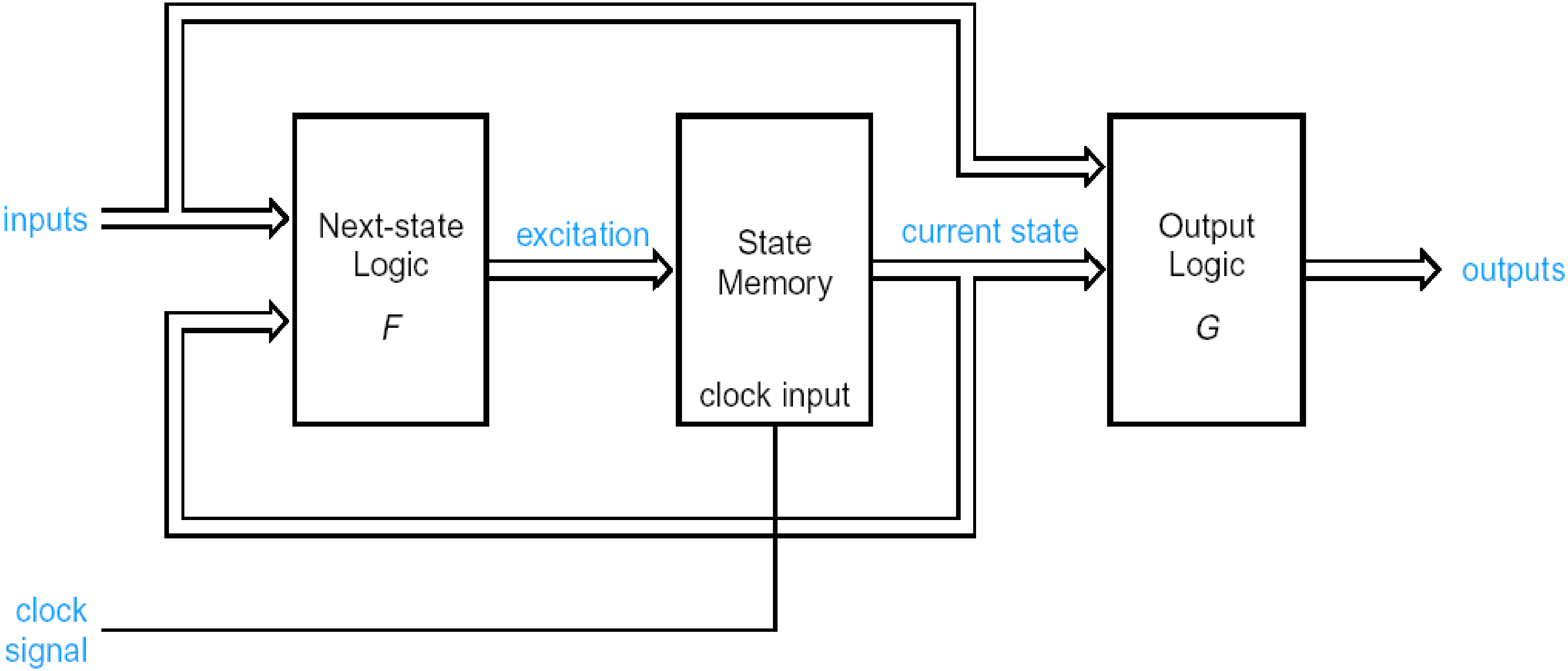


Clocked Synchronous State-Machine Analysis

- “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.

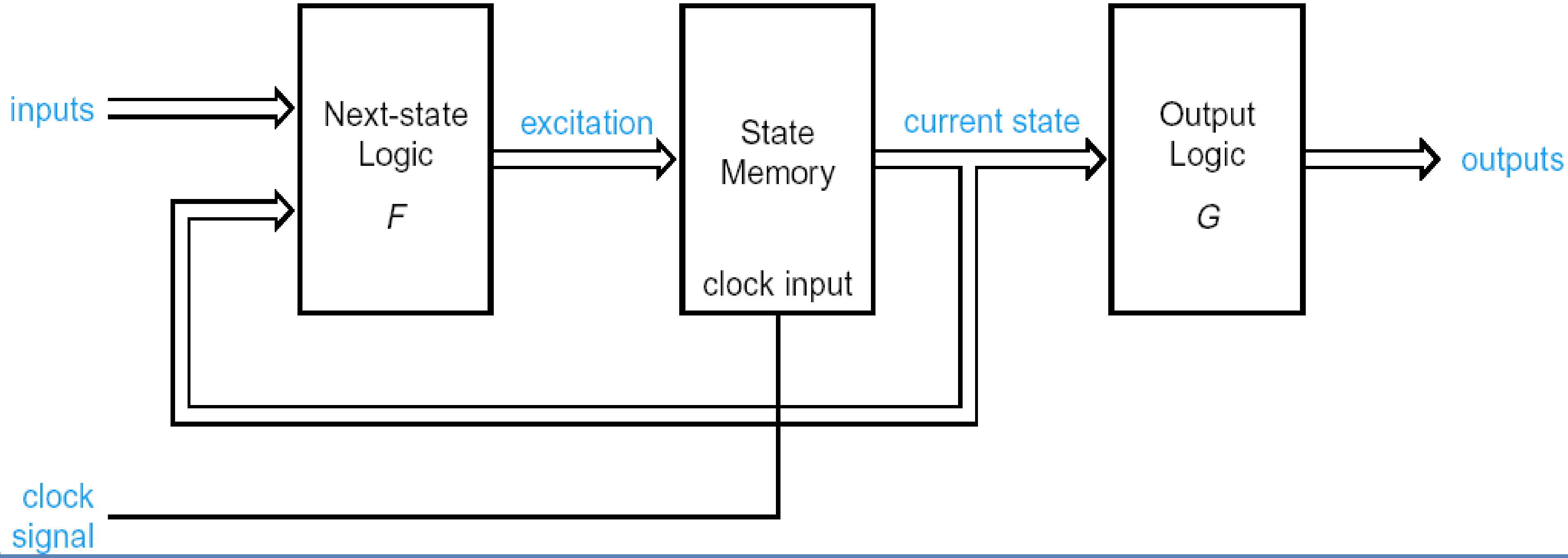
Mealy State-Machine

- 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

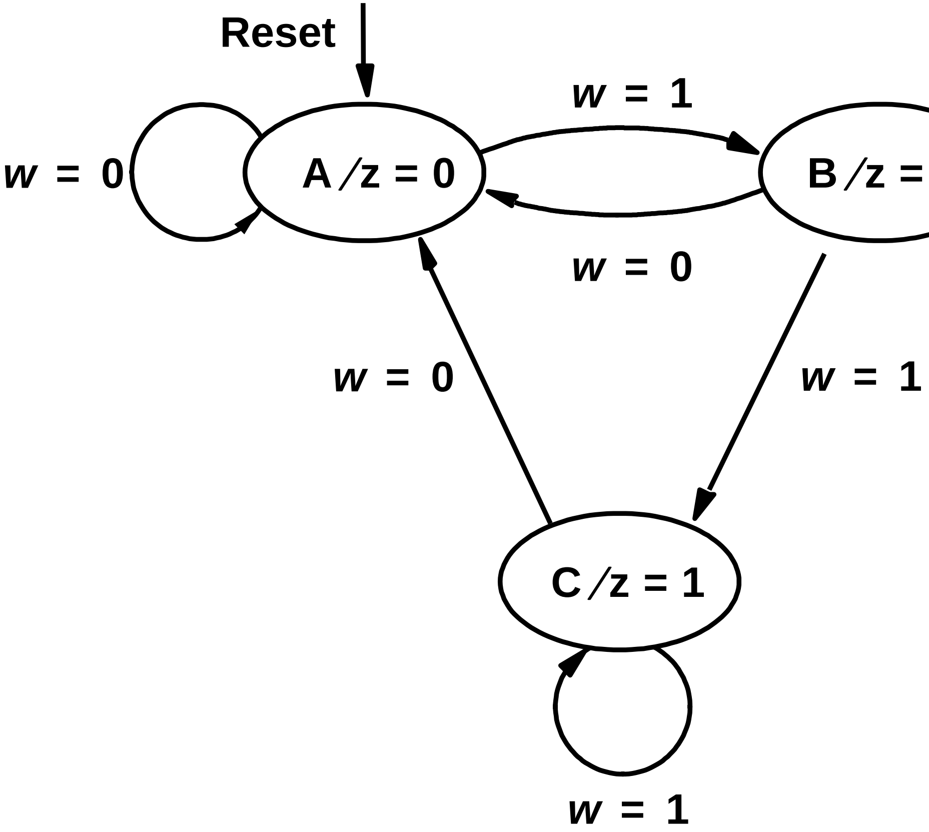


Basic Design Steps

- 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.

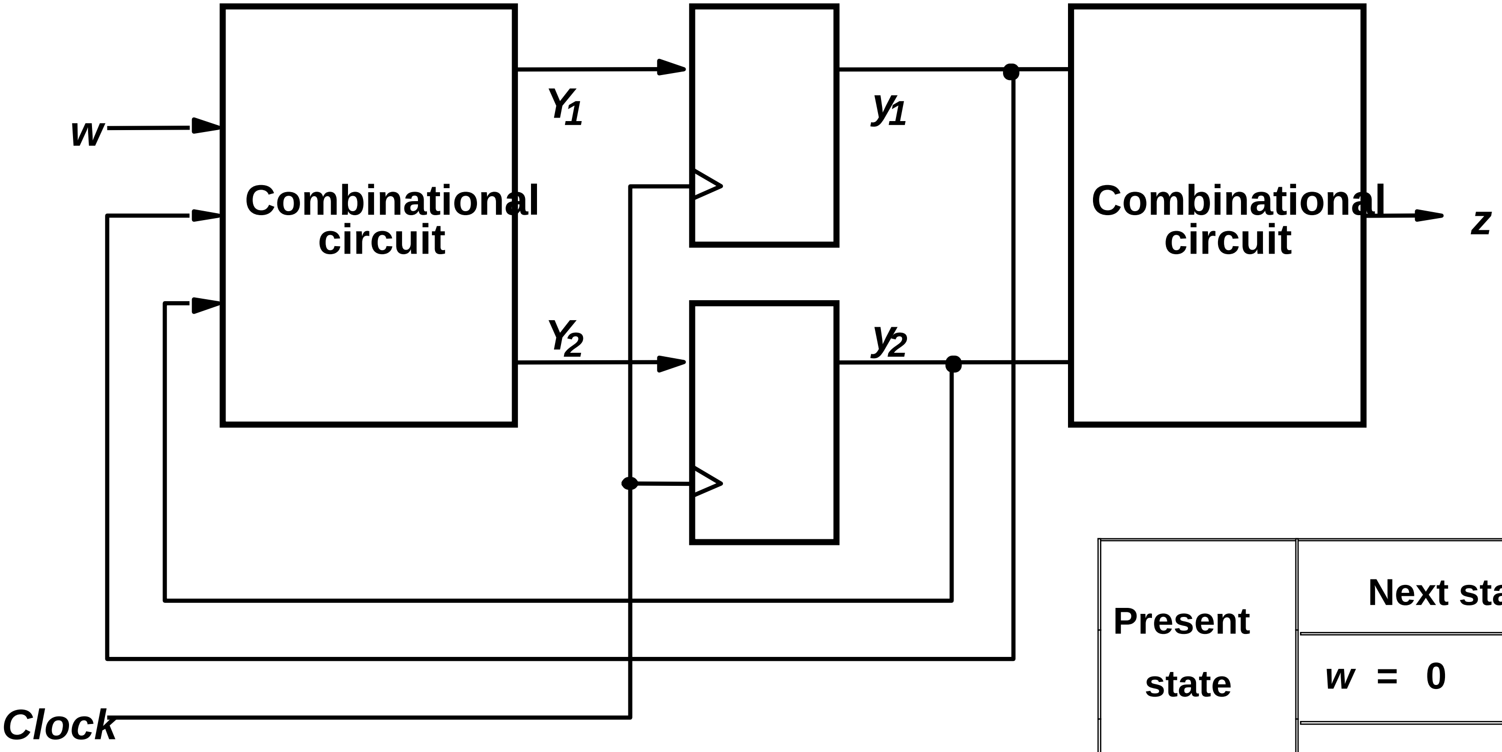
State Diagram and State Table

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

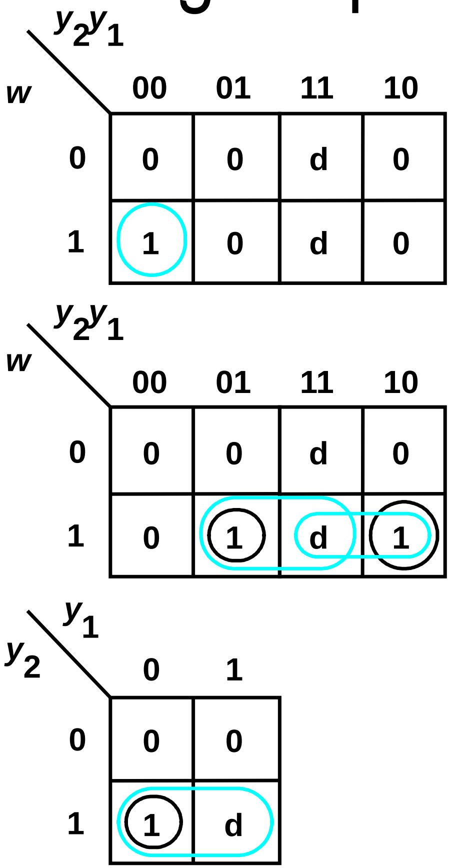
State Assignment



	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*

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

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



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

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

$y_2 \backslash y_1$	0	1
0	0	0
1	1	d

Ignoring don't cares

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

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

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

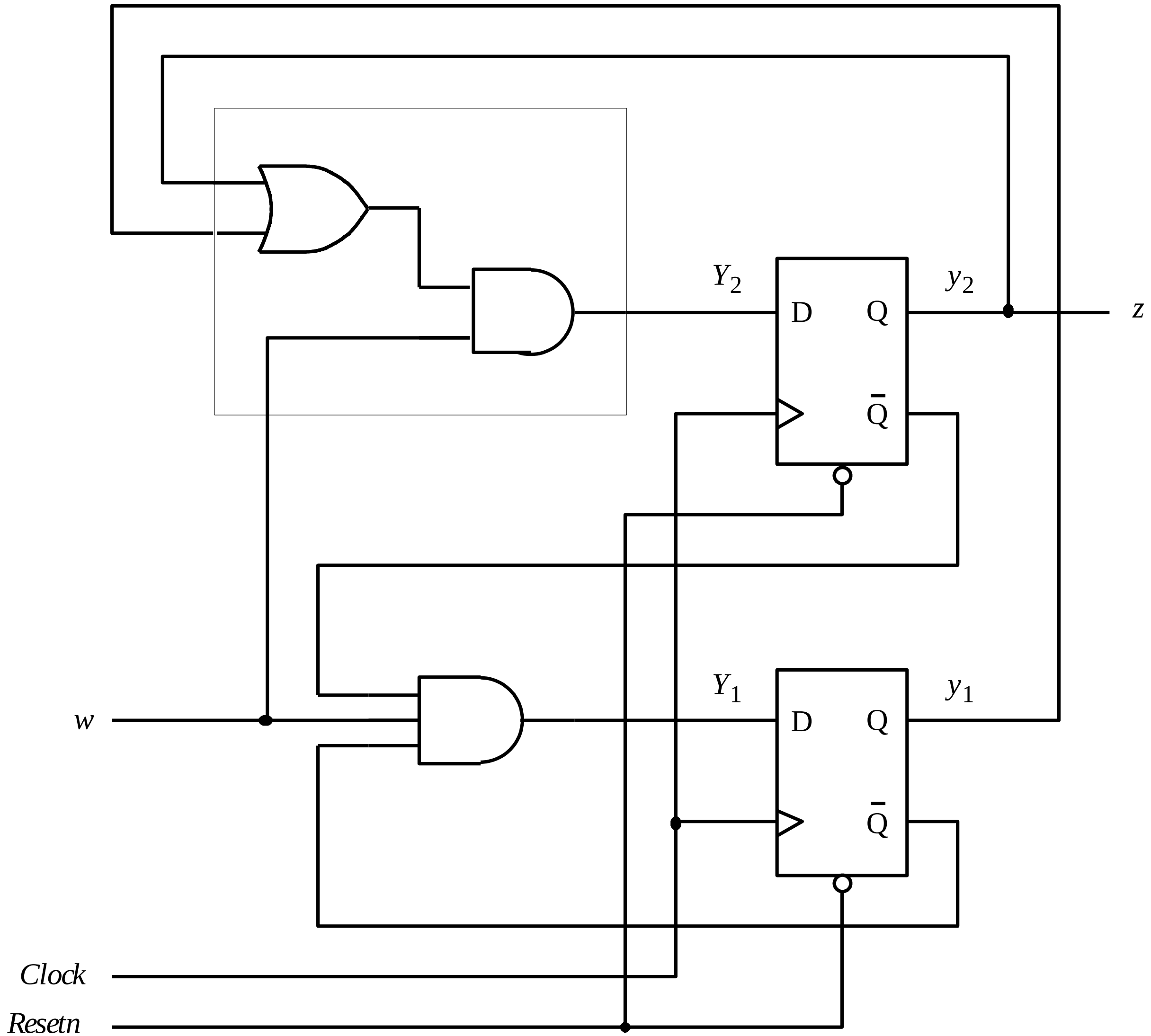
Using don't cares

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

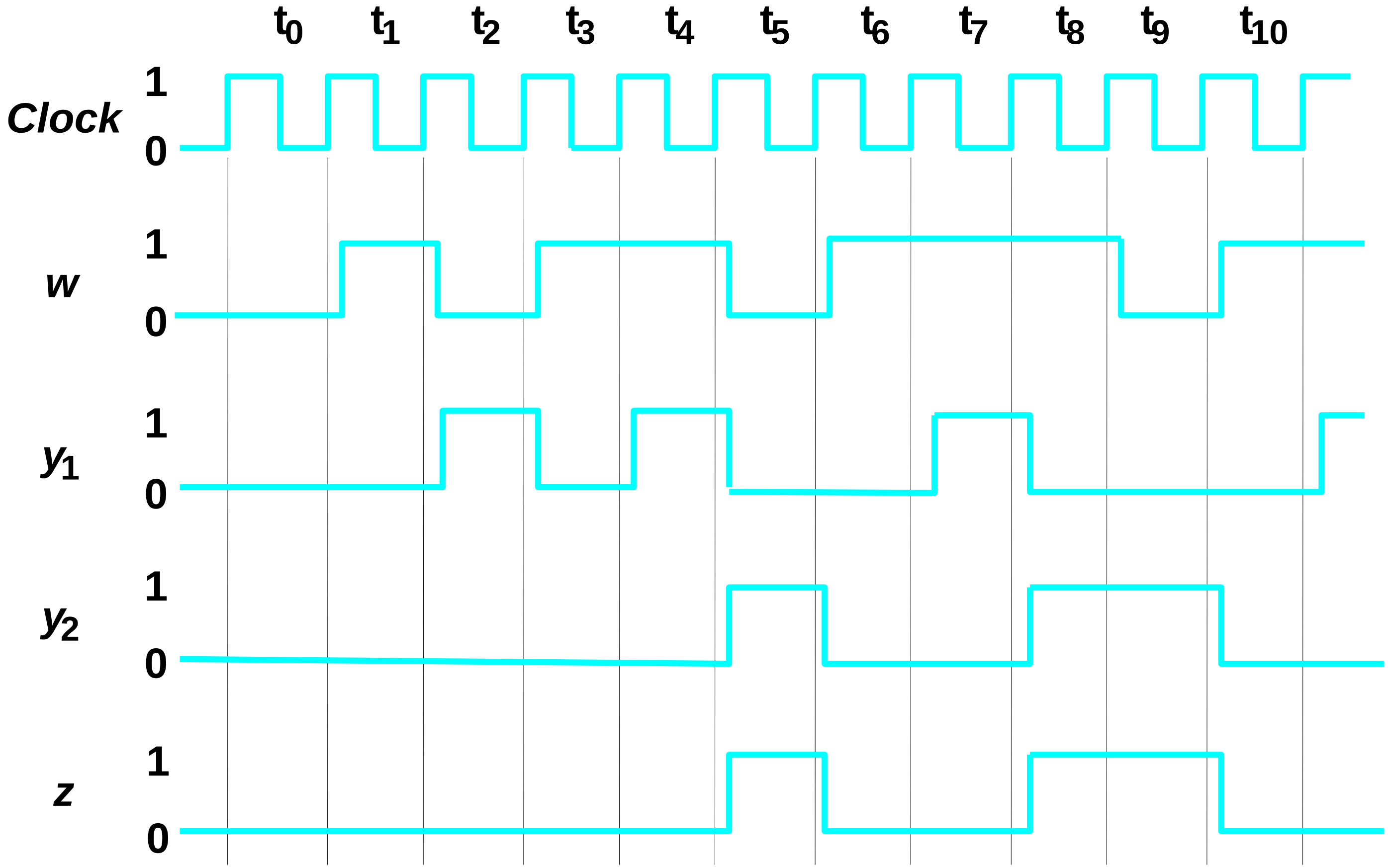
$$\begin{aligned} Y_2 &= wy_1 + wy_2 \\ &= w(y_1 + y_2) \end{aligned}$$

$$z = y_2$$

Final Implementation



Timing Diagram



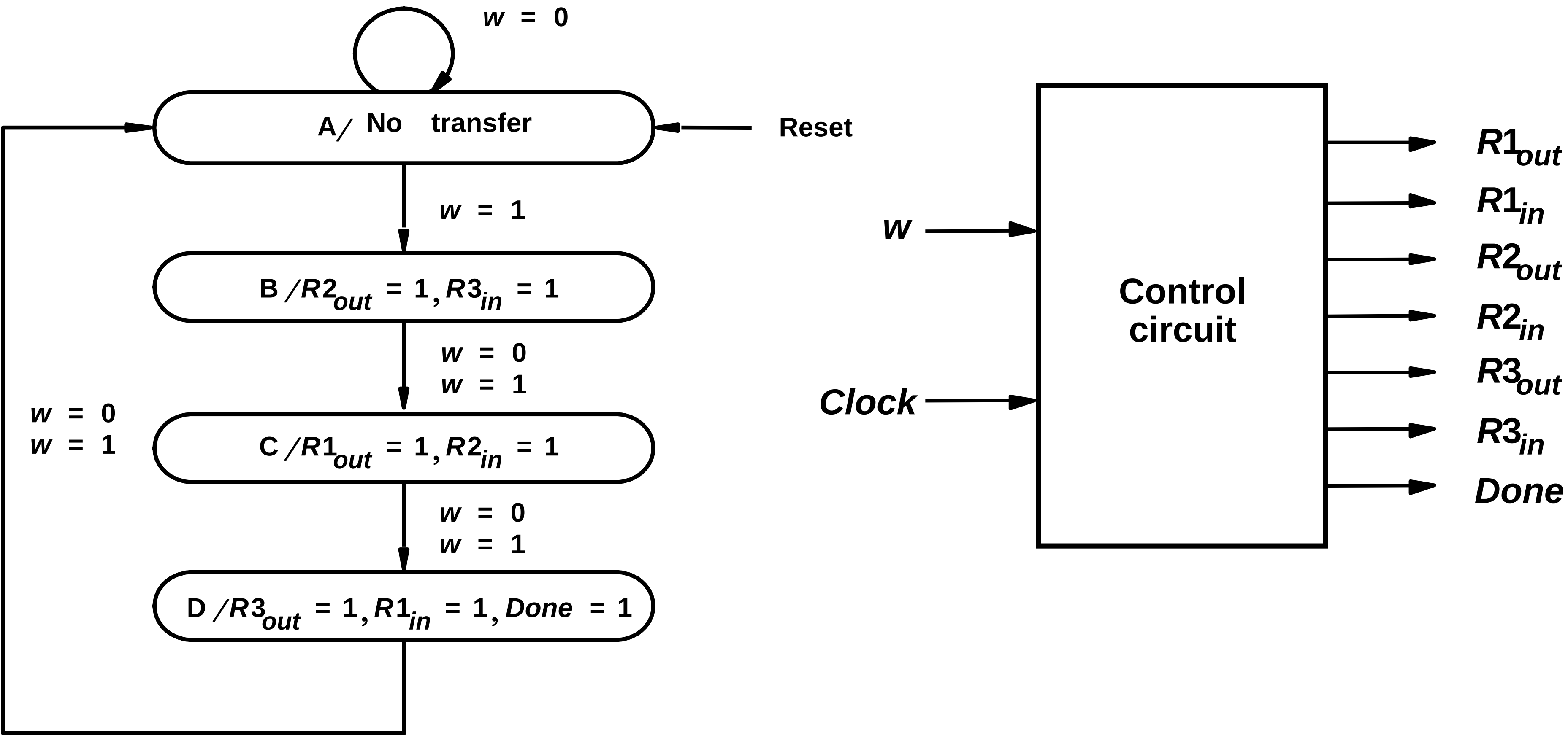
Summary of Design Steps

- 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

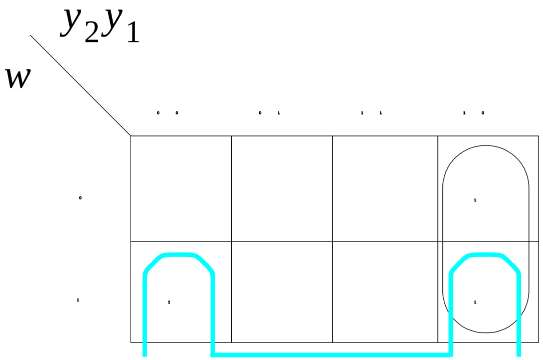


Ex 8.1 State table and state assignments

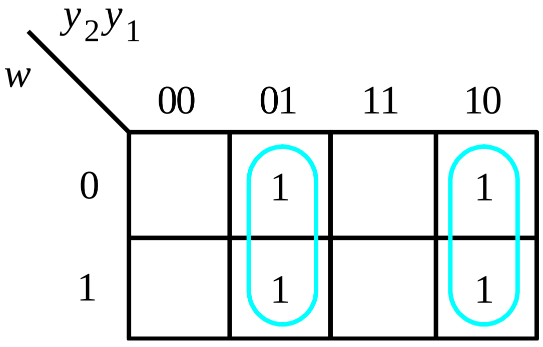
Present state	Next state		Outputs						
	w = 0	*w* = 1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	A	B	0	0	0	0	0	0	0
B	C	C	0	0	1	0	0	1	0
C	D	D	1	0	0	1	0	0	0
D	A	A	0	1	0	0	1	0	1

	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



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

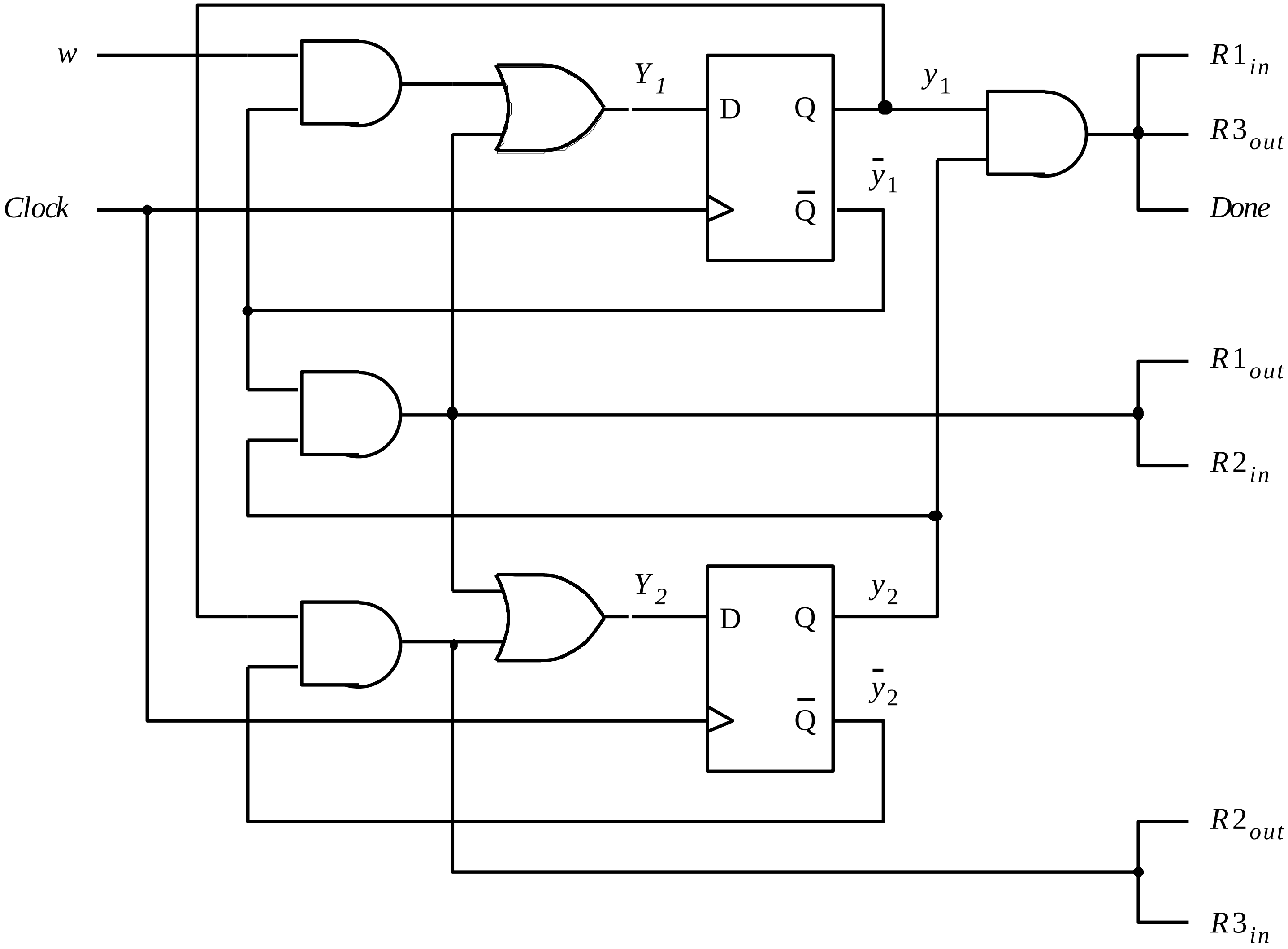


w \ y_2y_1	00	01	11	10
0		1		1
1		1		1

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

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

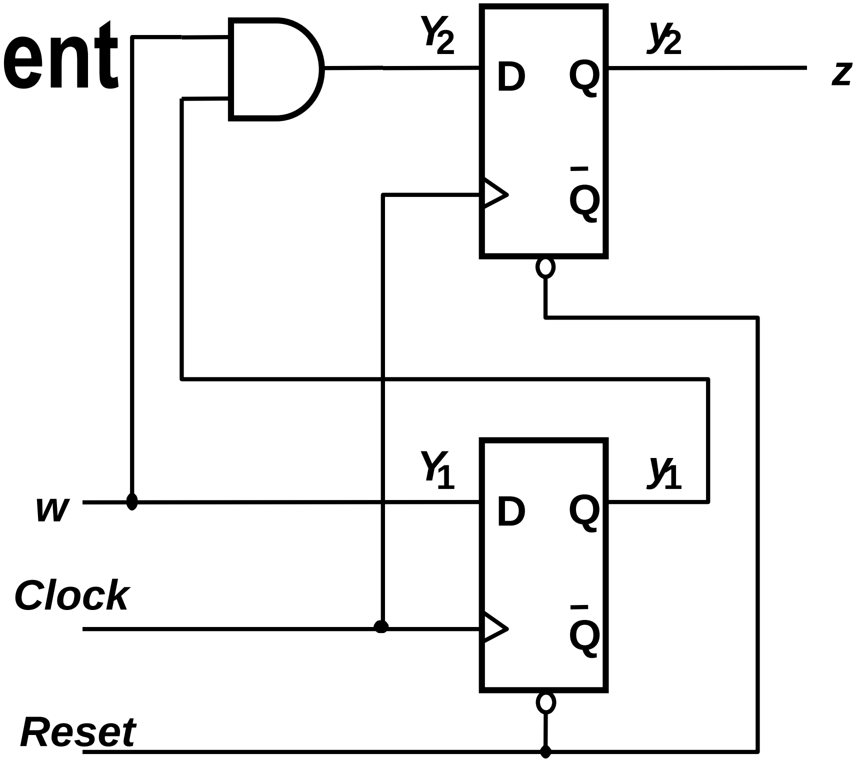
Ex 8.1 Final Implementation



Improved State Assignment

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
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	*dd*	*dd*	*d*

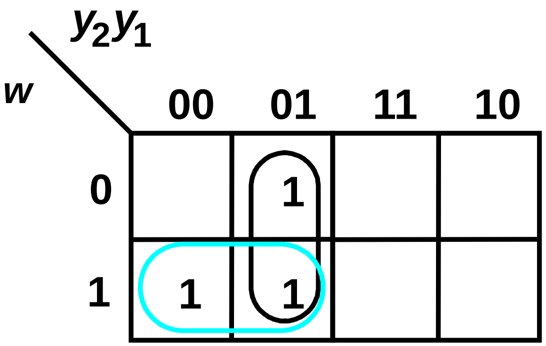


- $Y_1 = D_1 = w$
- $Y_2 = D_2 = wy_1$
- $z = y_2$

Improved State Assignment

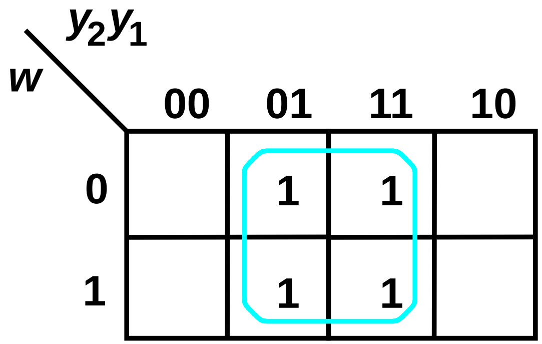
Present state	Next state		Outputs						
	w = 0	*w* = 1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	A	B	0	0	0	0	0	0	0
B	C	C	0	0	1	0	0	1	0
C	D	D	1	0	0	1	0	0	0
D	A	A	0	1	0	0	1	0	1

	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	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



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

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



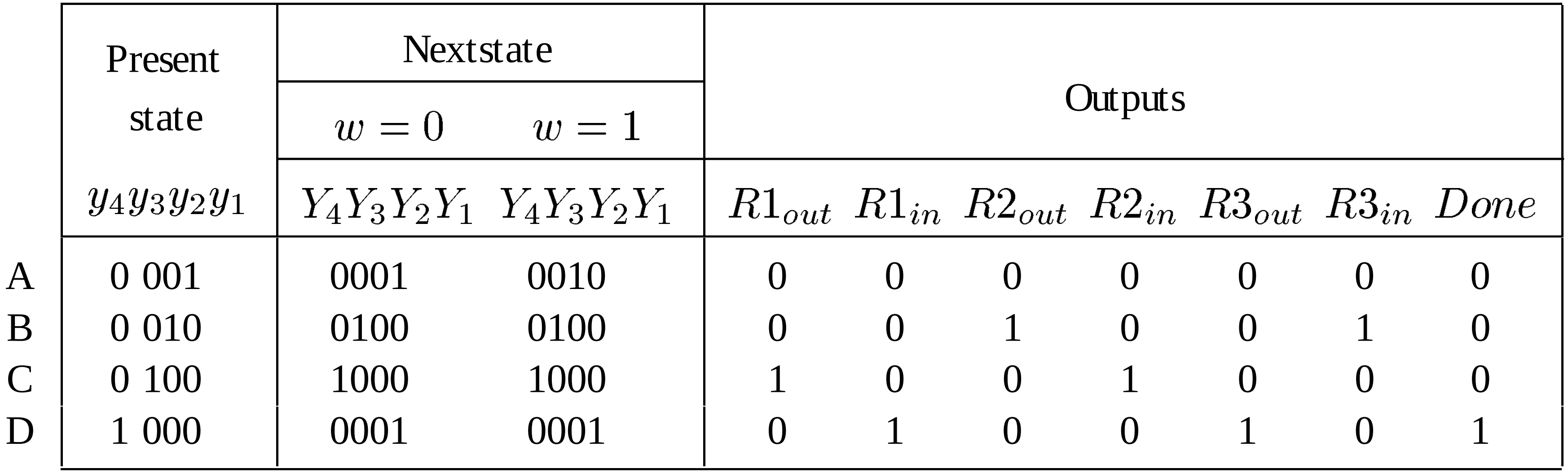
w \ y_2y_1	00	01	11	10
0		1	1	
1		1	1	

$Y_2 = y_1$

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

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



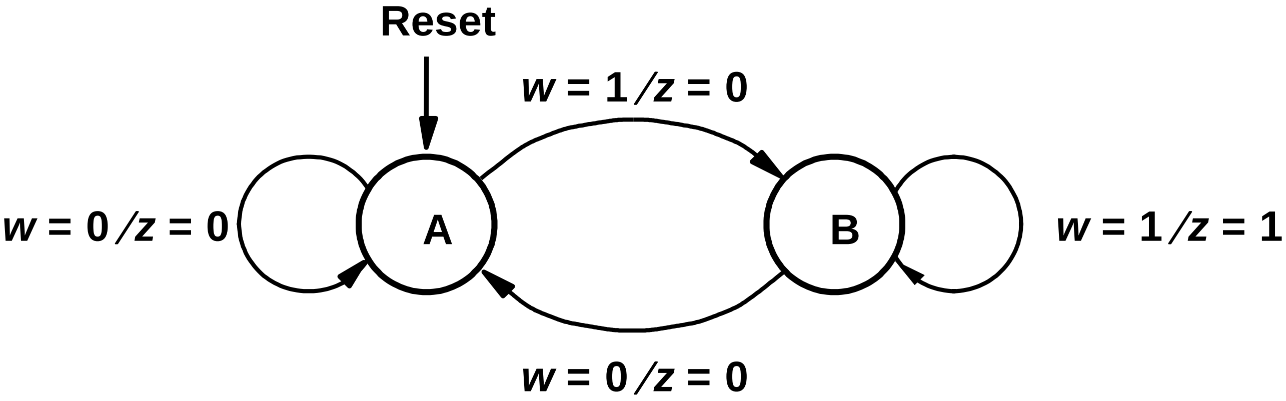
	Present state	Nextstate		Outputs						
		$w = 0$	$w = 1$							
	$y_4y_3y_2y_1$	$Y_4Y_3Y_2Y_1$	$Y_4Y_3Y_2Y_1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	0 001	0001	0010	0	0	0	0	0	0	0
B	0 010	0100	0100	0	0	1	0	0	1	0
C	0 100	1000	1000	1	0	0	1	0	0	0
D	1 000	0001	0001	0	1	0	0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

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

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



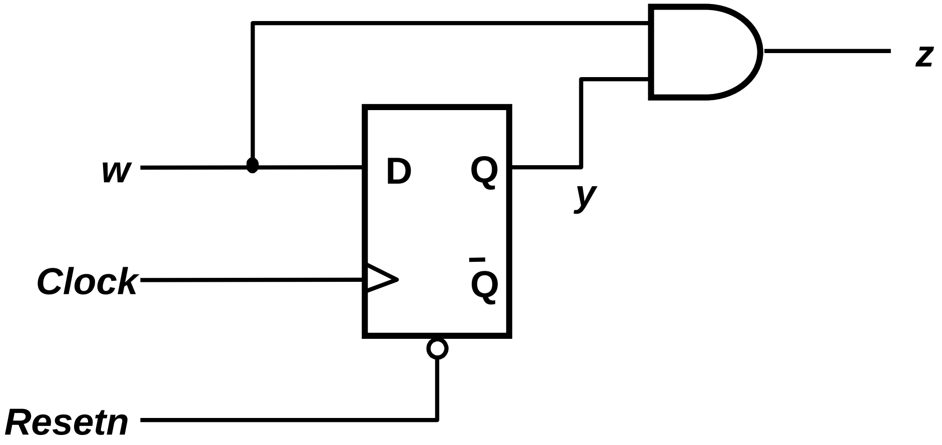
State Table and State Assignment

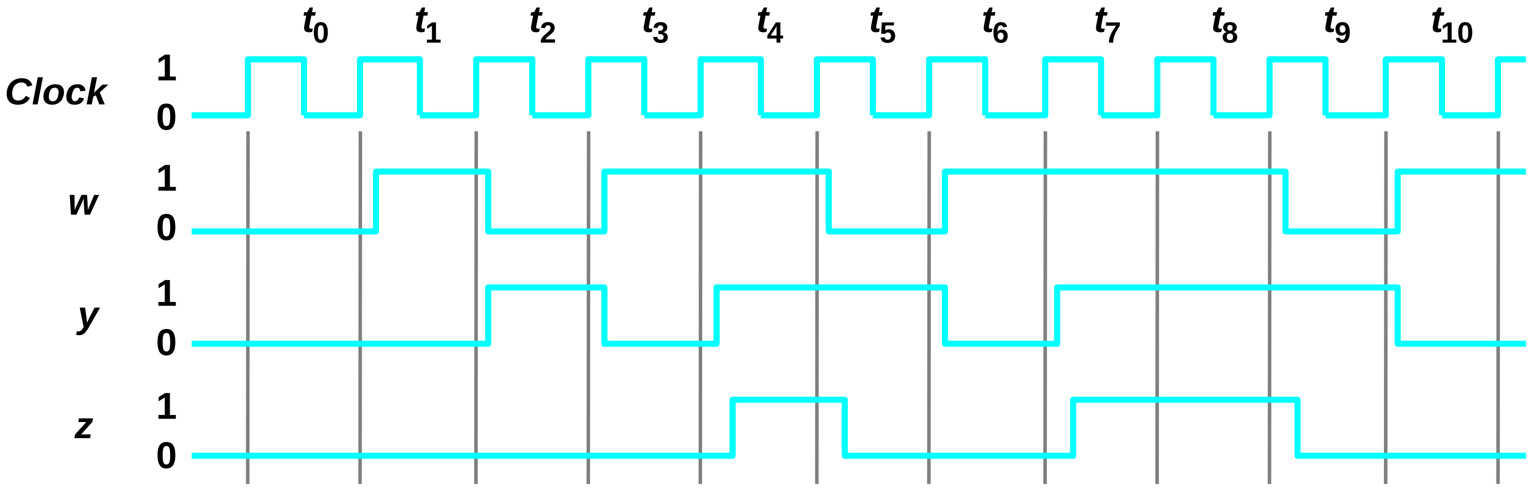
Present state	Next state		Output *z*	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

	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

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

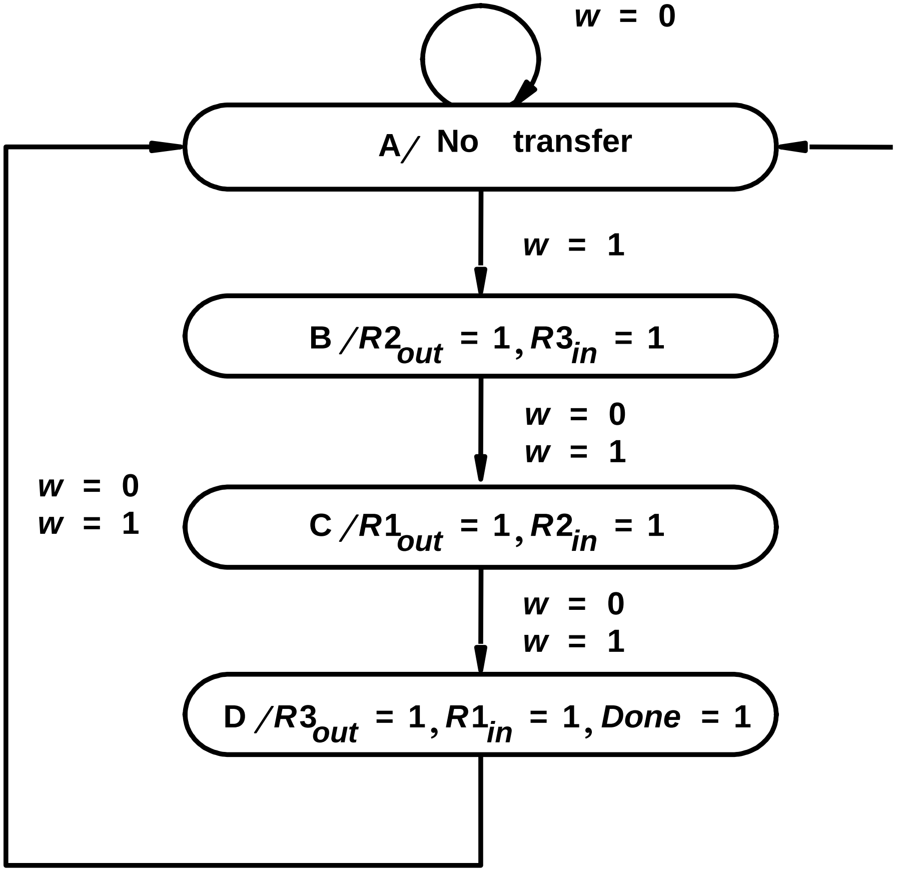
Circuit and Timing Diagram





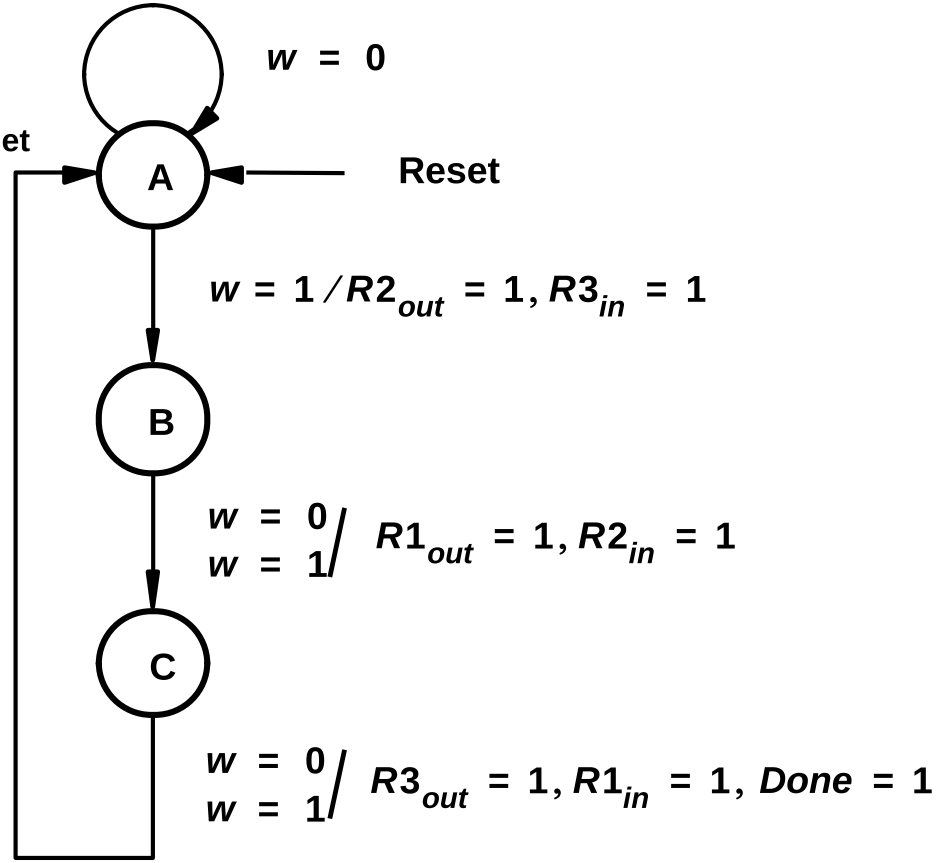
Ex. 8.4

- Moore type



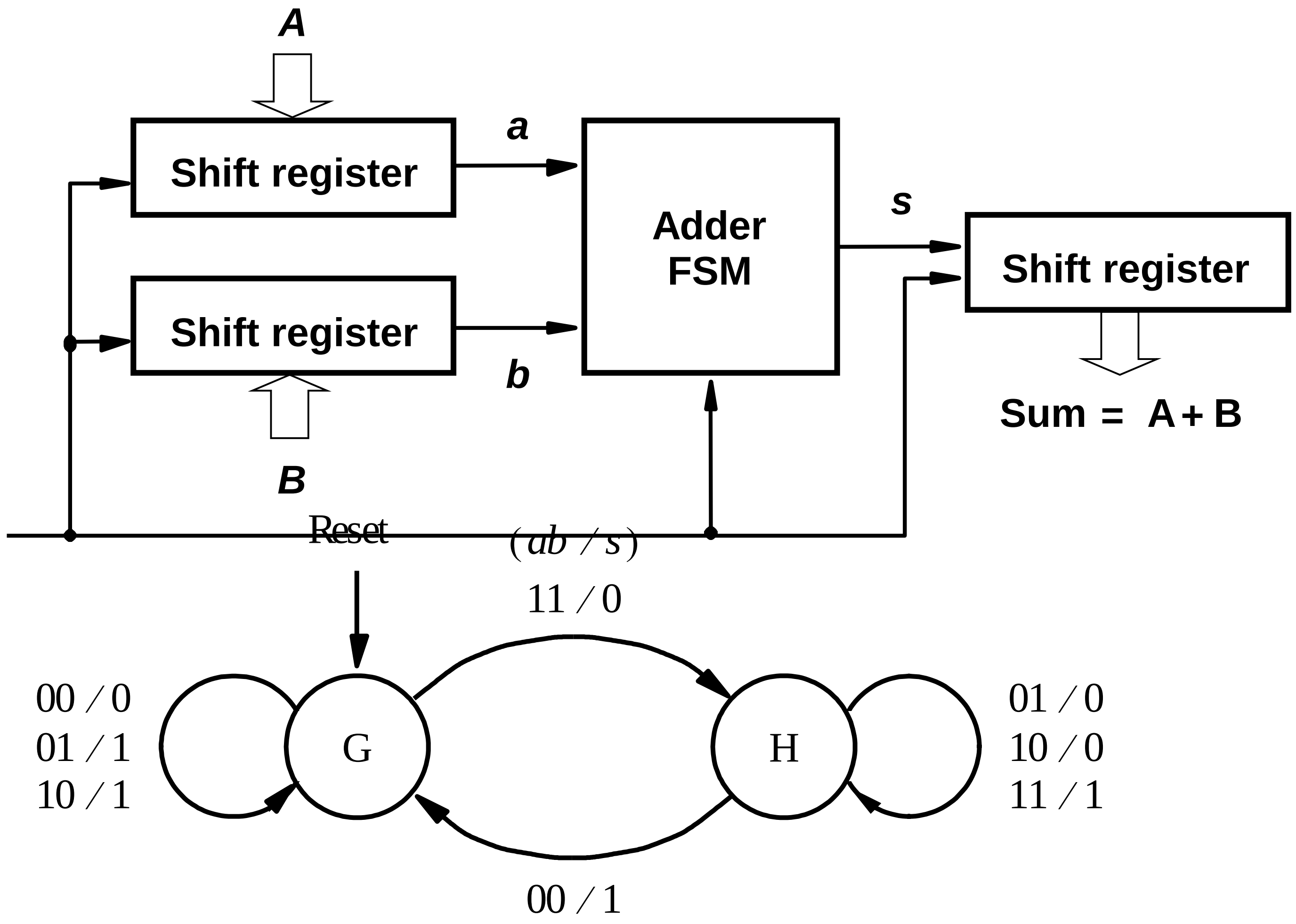
$w = 0$
A/ No transfer
Reset
$w = 1$
B $/R2_{out} = 1, R3_{in} = 1$
$w = 0$
$w = 1$
C $/R1_{out} = 1, R2_{in} = 1$
$w = 0$
$w = 1$
D $/R3_{out} = 1, R1_{in} = 1, Done = 1$
$w = 0$
$w = 1$

Mealy type



$w = 0$
A
Reset
$w = 1 / R2_{out} = 1, R3_{in} = 1$
B
$w = 0$
$w = 1$
$/ R1_{out} = 1, R2_{in} = 1$
C
$w = 0$
$w = 1$
$/ R3_{out} = 1, R1_{in} = 1, Done = 1$

Mealy-Type FSM for Serial Adder



G: carry-in = 0

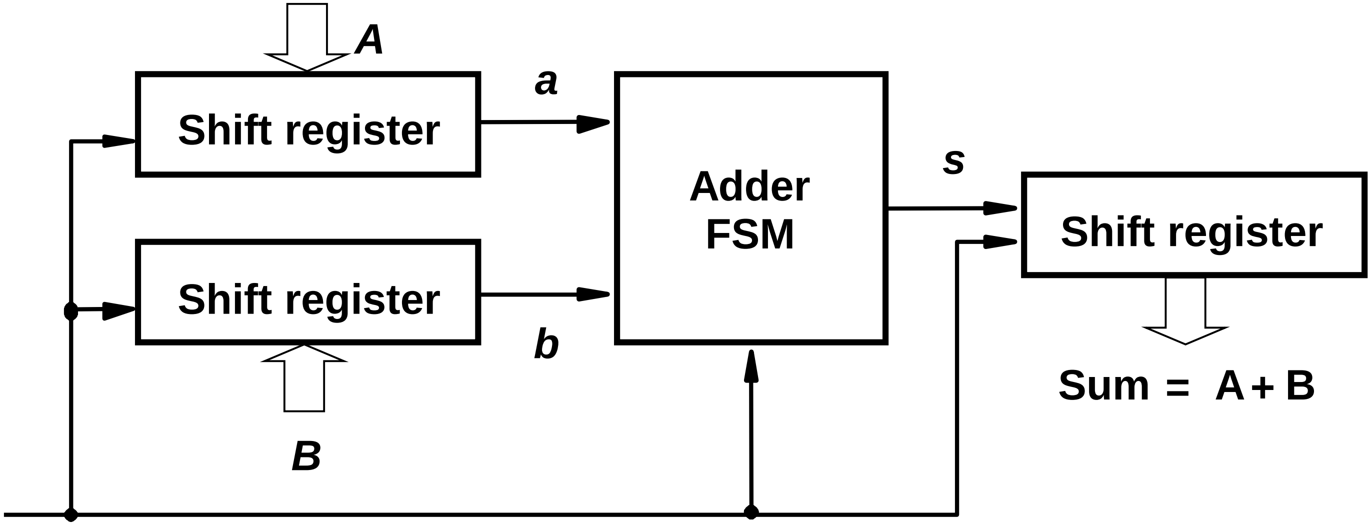
H: carry-in = 1

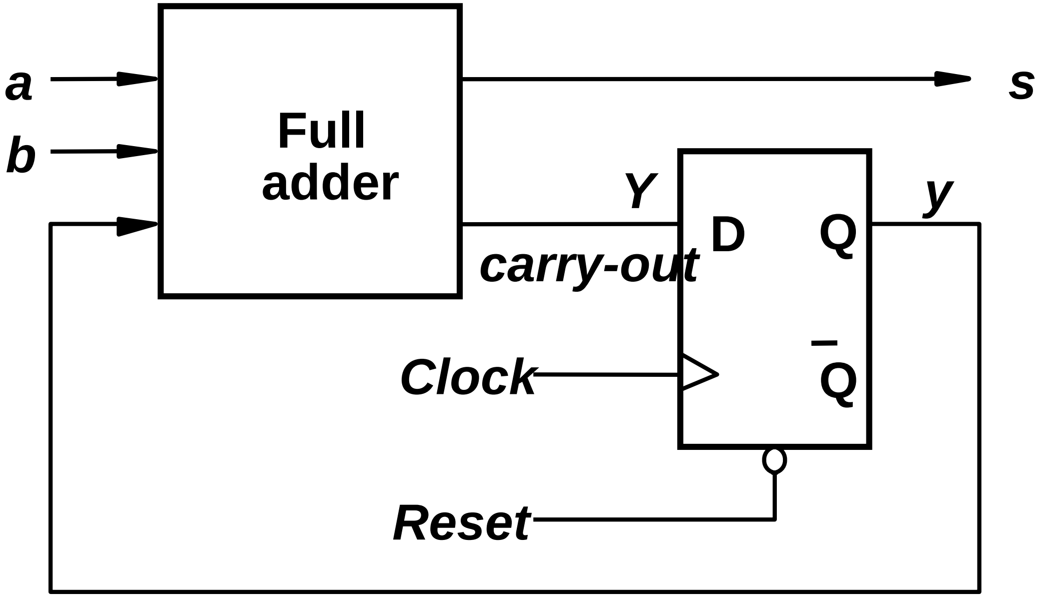
Present state	Next state				Outputs			
	ab=00	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

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

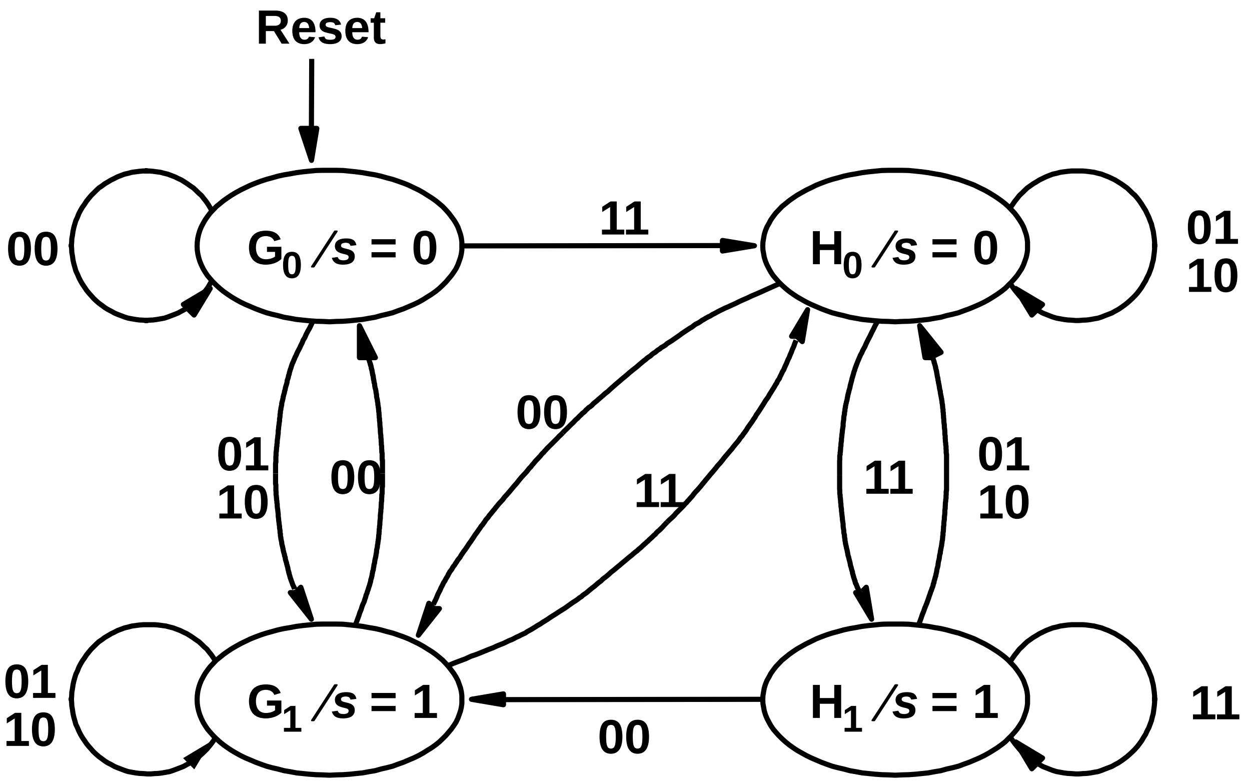
- $Y = ab + ay + by$
- $s = a \oplus b \oplus$ y

Mealy-type Serial Adder Circuit





Moore-type FSM for Serial Adder

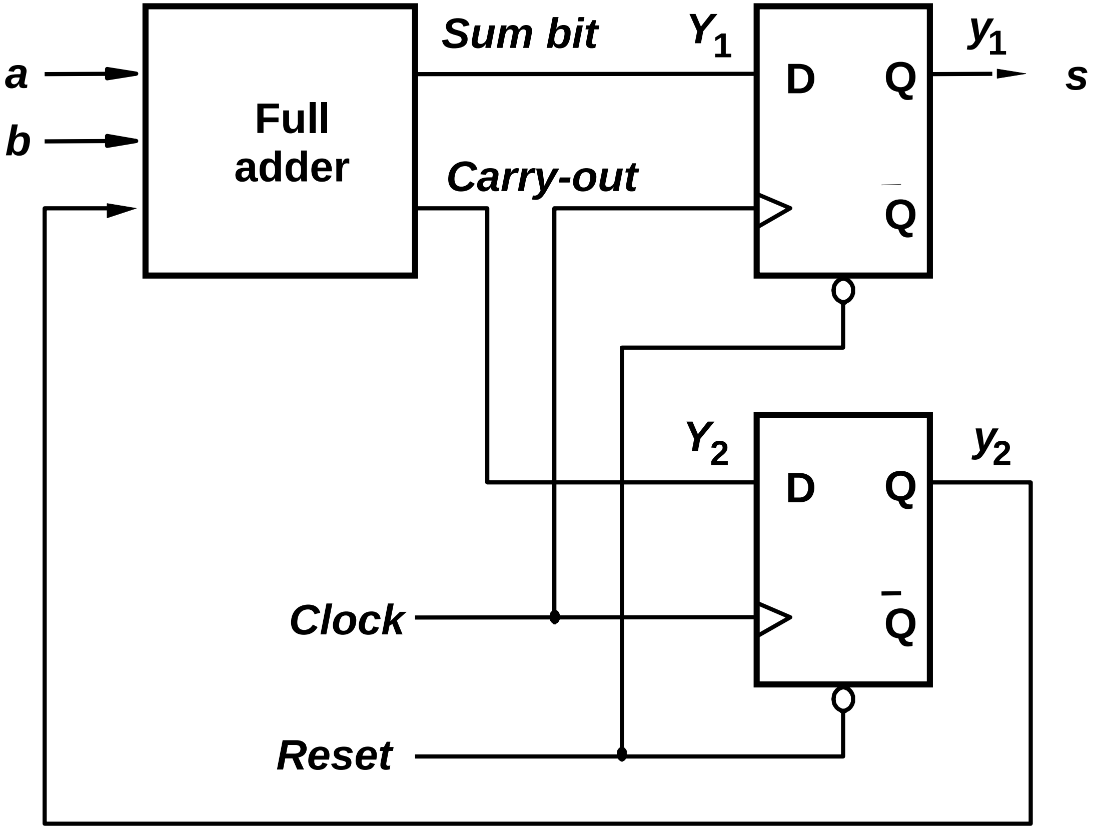


Present state	Nextstate				Output
	ab=00	01	10	11	*s*
G_0	G_0	G_1	G_1	H_0	0
G_1	G_0	G_1	G_1	H_0	1
H_0	G_1	H_0	H_0	H_1	0
H_1	G_1	H_0	H_0	H_1	1

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

  $P_2 = (\mathrm{ABD})(\mathrm{CEFG})$

- Check 0-sucessors and 1-succesors

  $P_3 = (\mathrm{ABD})(\mathrm{CEG})(\mathrm{F})$

- Again

  $P_4 = (\mathrm{AD})(\mathrm{B})(\mathrm{CEG})(\mathrm{F})$

  $P_5 = (\mathrm{AD})(\mathrm{B})(\mathrm{CEG})(\mathrm{F})$

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

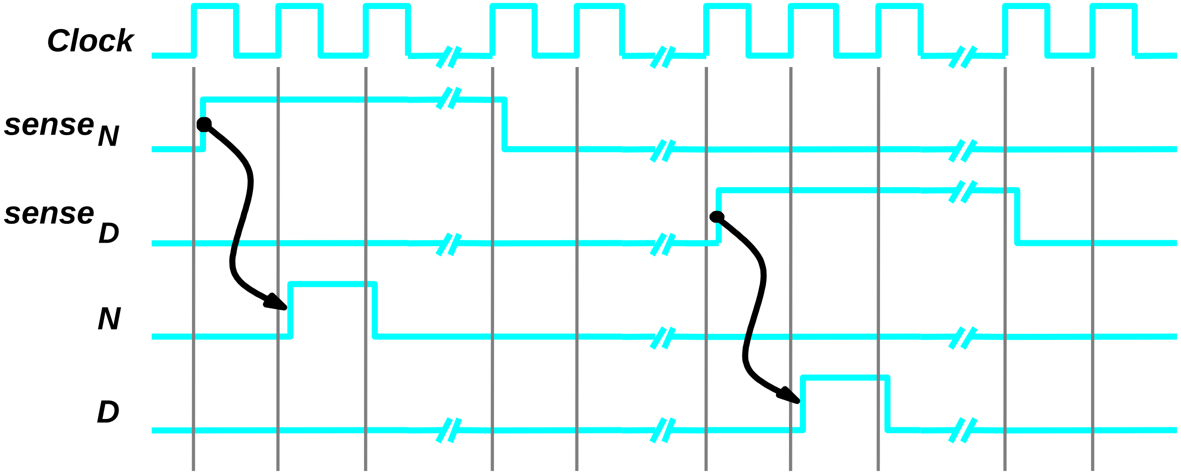
Present state	Nextstate		Output z
	w = 0	w = 1	
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

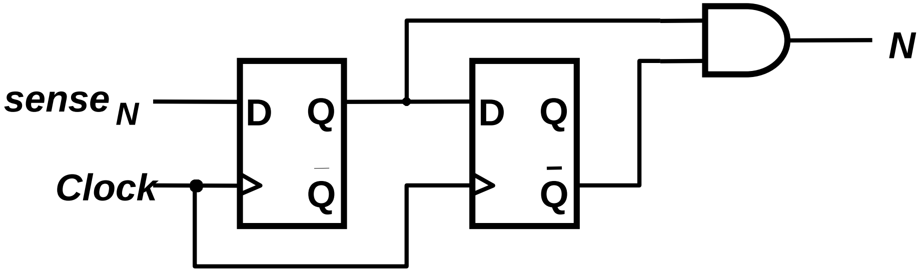
Ex8.7 Vending Machine

- 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

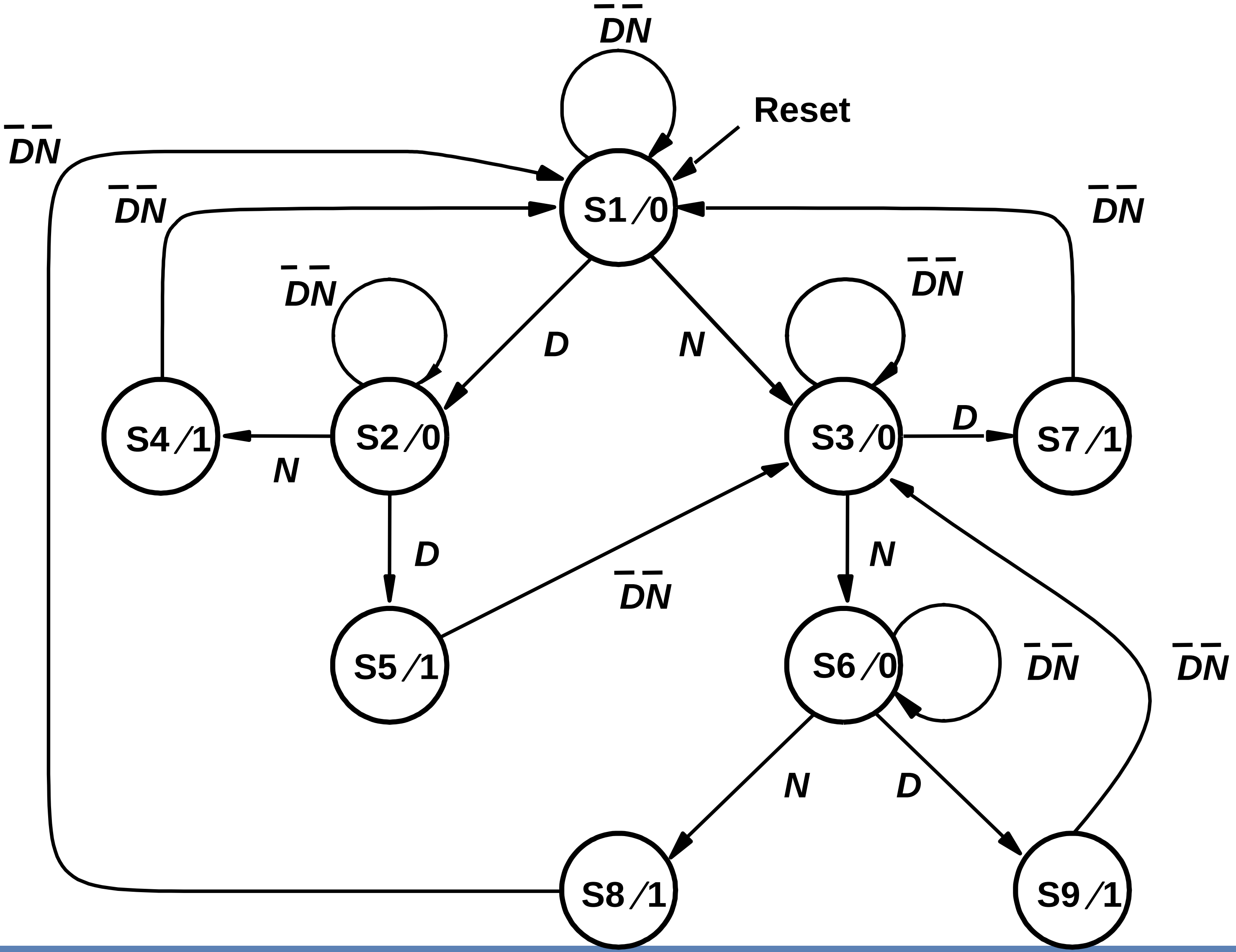
Denounced Circuit

- 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)

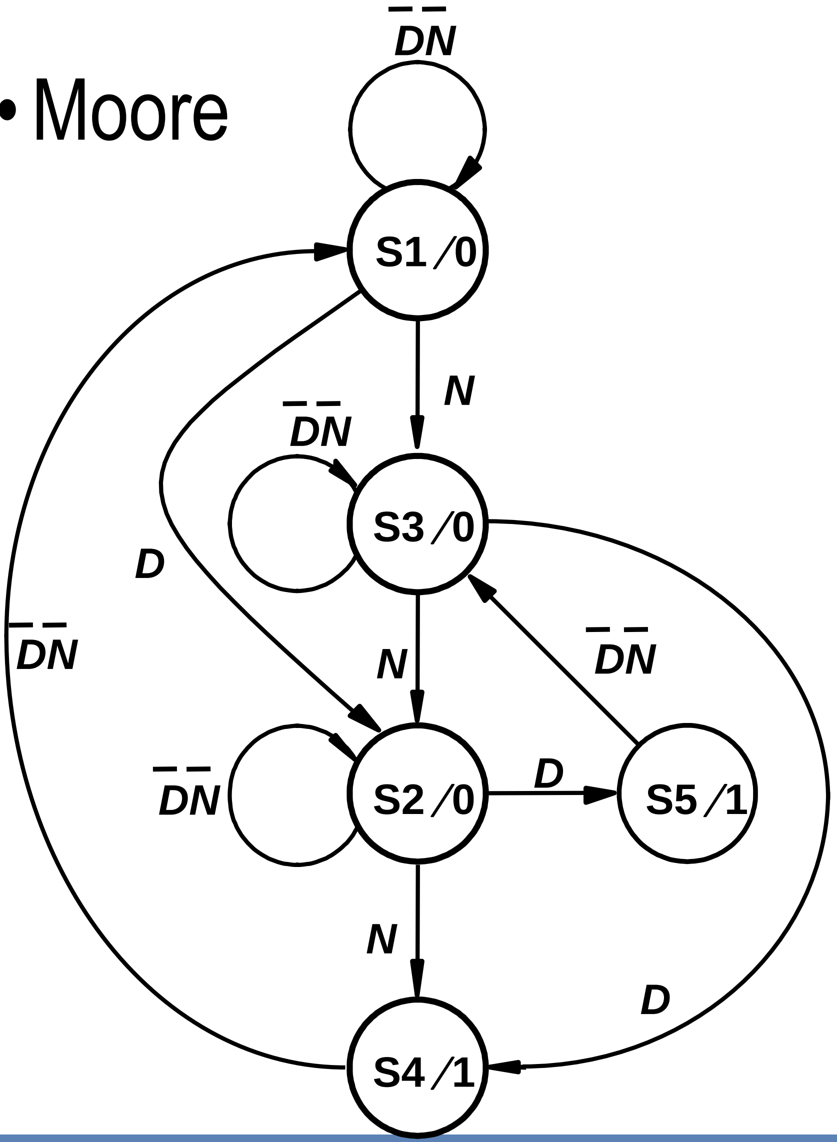
Present state	Next state				Output *z*
	DN =00	01	10	11	
S1	S1	S3	S2	–	0
S2	S2	S4	S5	–	0
S3	S3	S6	S7	–	0
S4	S1	–	–	–	1
S5	S3	–	–	–	1
S6	S6	S8	S9	–	0
S7	S1	–	–	–	1
S8	S1	–	–	–	1
S9	S3	–	–	–	1

Minimized State Table

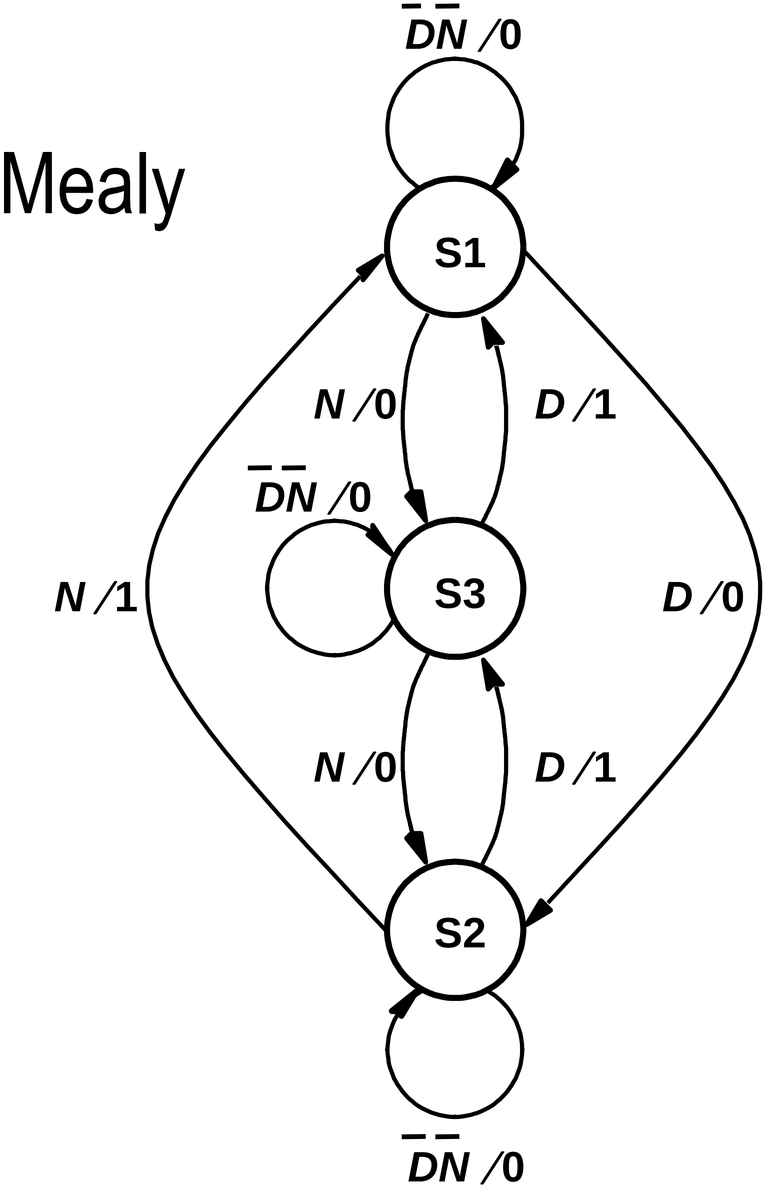
Present state	Next state				Output z
	DN =00	01	10	11	
S1	S1	S3	S2	–	0
S2	S2	S4	S5	–	0
S3	S3	S2	S4	–	0
S4	S1	–	–	–	1
S5	S3	–	–	–	1

State Diagram

- Moore


$\overline{D}\overline{N}$
S1/0
N
$\overline{D}\overline{N}$
S3/0
D
$\overline{D}\overline{N}$
N
$\overline{D}\overline{N}$
$\overline{D}\overline{N}$
S2/0
D
S5/1
N
D
S4/1

Mealy


$\overline{D}\overline{N}$/0
S1
N/0
D/1
$\overline{D}\overline{N}$/0
N/1
S3
D/0
N/0
D/1
S2
$\overline{D}\overline{N}$/0

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)

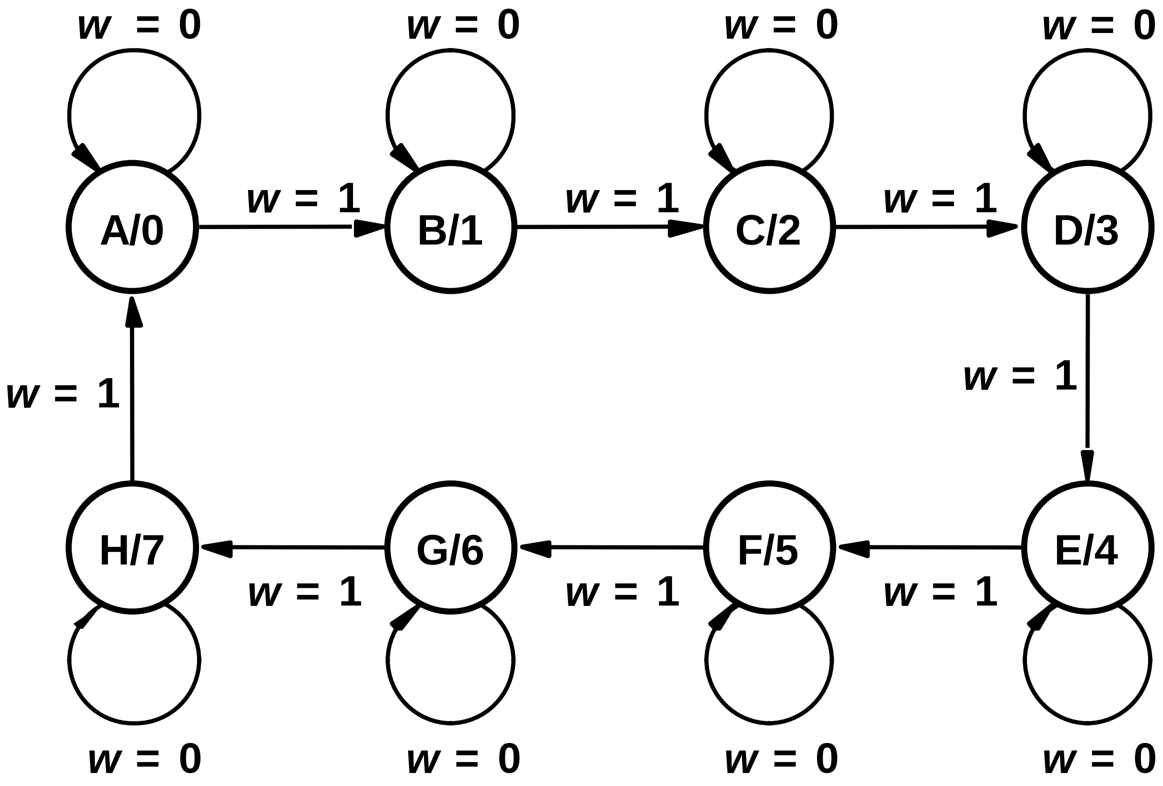
  P_5=P_4
- Assume -=1

  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



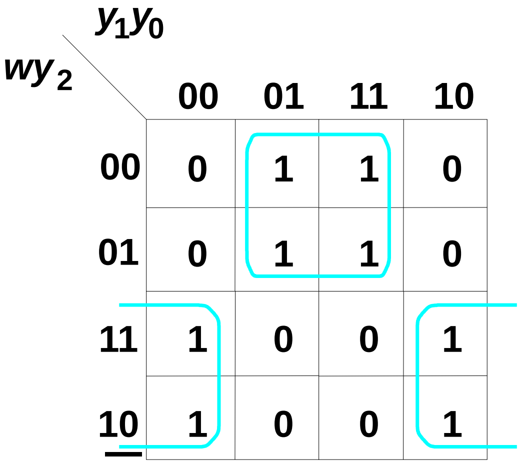
State Table for A Modulo-8 Counter

Present state	Next state		Output
	$w = 0$	$w = 1$	
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	Next state		Count
		$w = 0$	$w = 1$	
	$y_2y_1y_0$	$Y_2Y_1Y_0$	$Y_2Y_1Y_0$	$z_2z_1z_0$
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

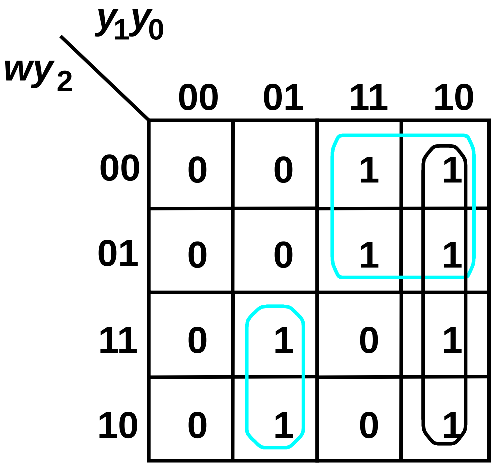
Implementation Using D-type Flip-Flops

- $D_0 = Y_0$
  $= \bar{w}y_0 + w\bar{y}_0$
  $= w \oplus y_0$
- $D_1 = Y_1$
  $= \bar{w}y_1 + y_1\overline{y_0} + wy_0\bar{y}_1$
  $= wy_0 \oplus y_1$
- $D_2 = Y_2$
  $= \bar{w}y_2 + \bar{y}_0y_2 + \overline{y_1}y_2$
  $+ wy_0y_1\bar{y}_2$
  $= wy_0y_1 \oplus y_2$



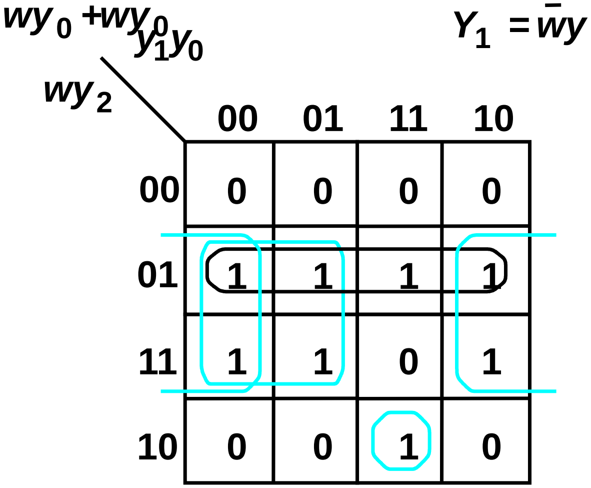
wy_2 \ y_1y_0	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	1

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



wy_2 \ 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

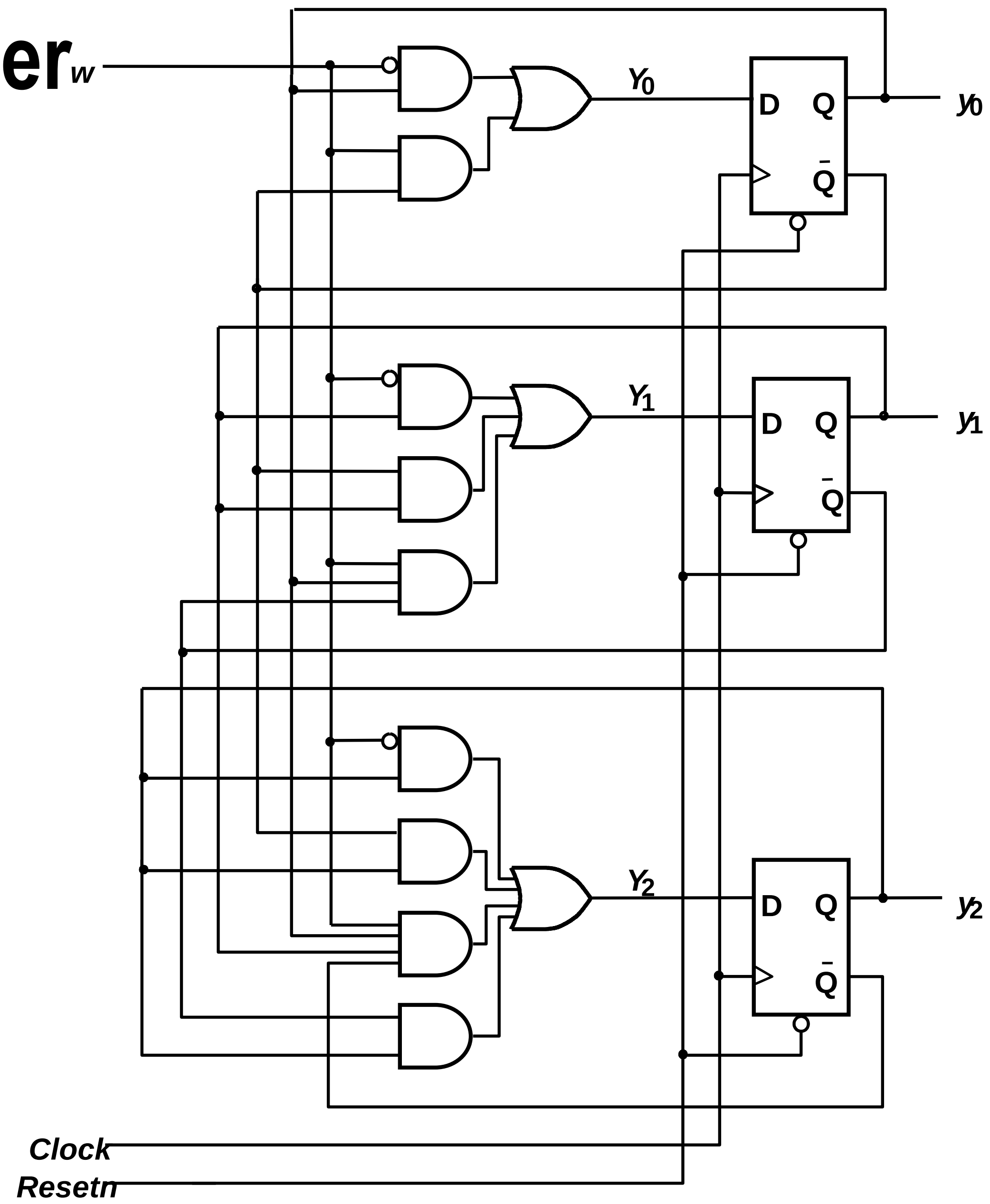
$Y_1 = \bar{w}y_1 + y_1\bar{y}_0 + wy_0\bar{y}_1$



wy_2 \ 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



Counting 0,4,2,6,1,5,3,7,0,4 …

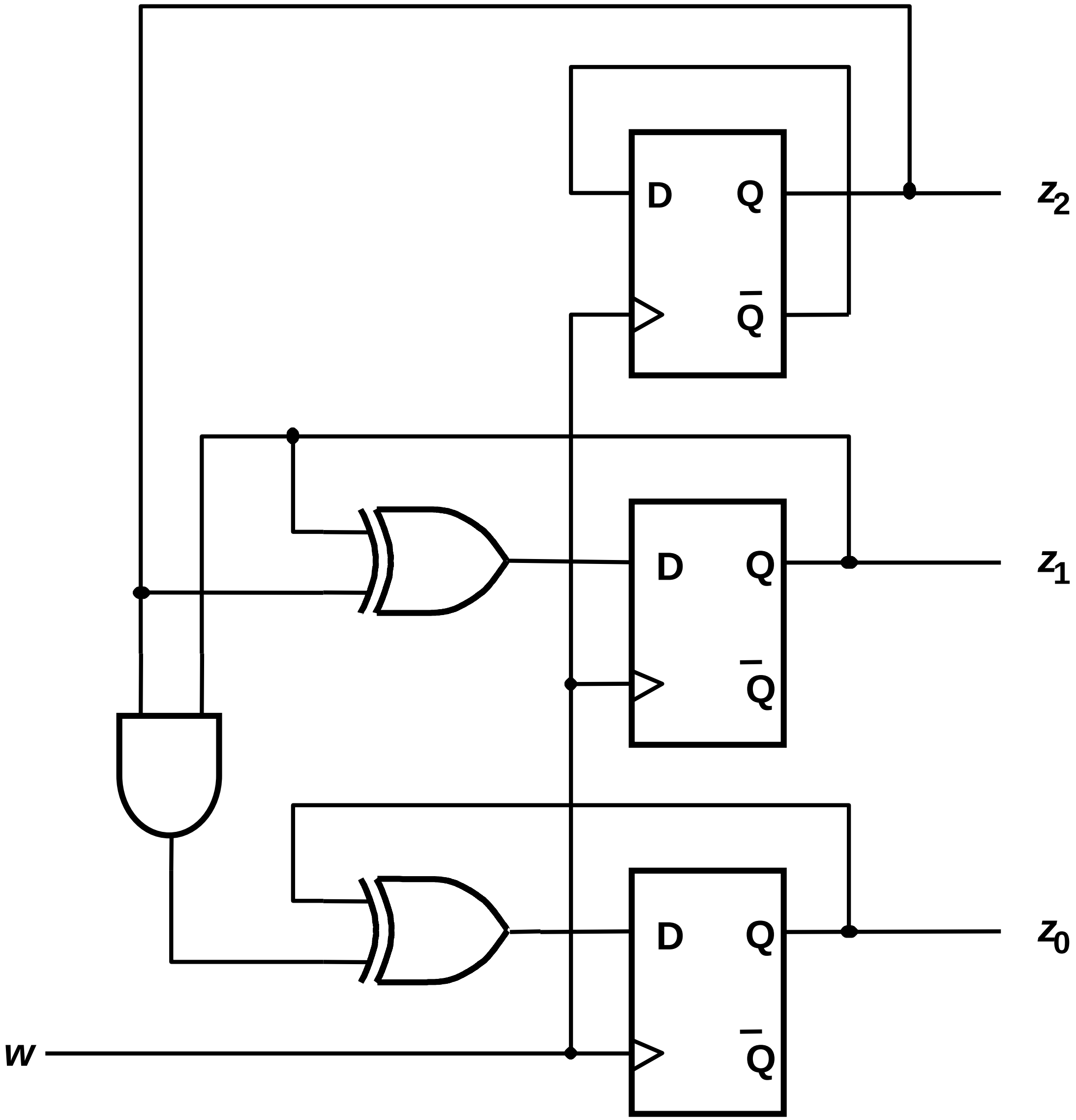
$$D_2 = Y_2 = \overline{y_2}$$

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

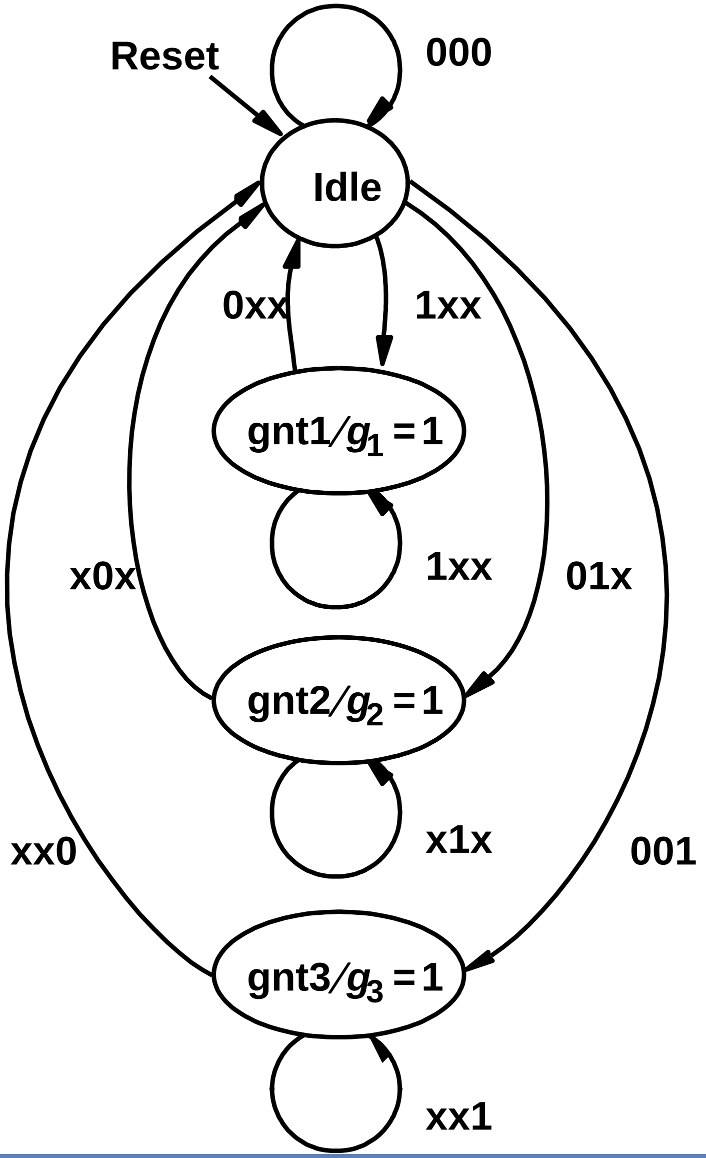
Present state	Next state	Output $z_2z_1z_0$
A	B	000
B	C	100
C	D	010
D	E	110
E	F	001
F	G	101
G	H	011
H	A	111

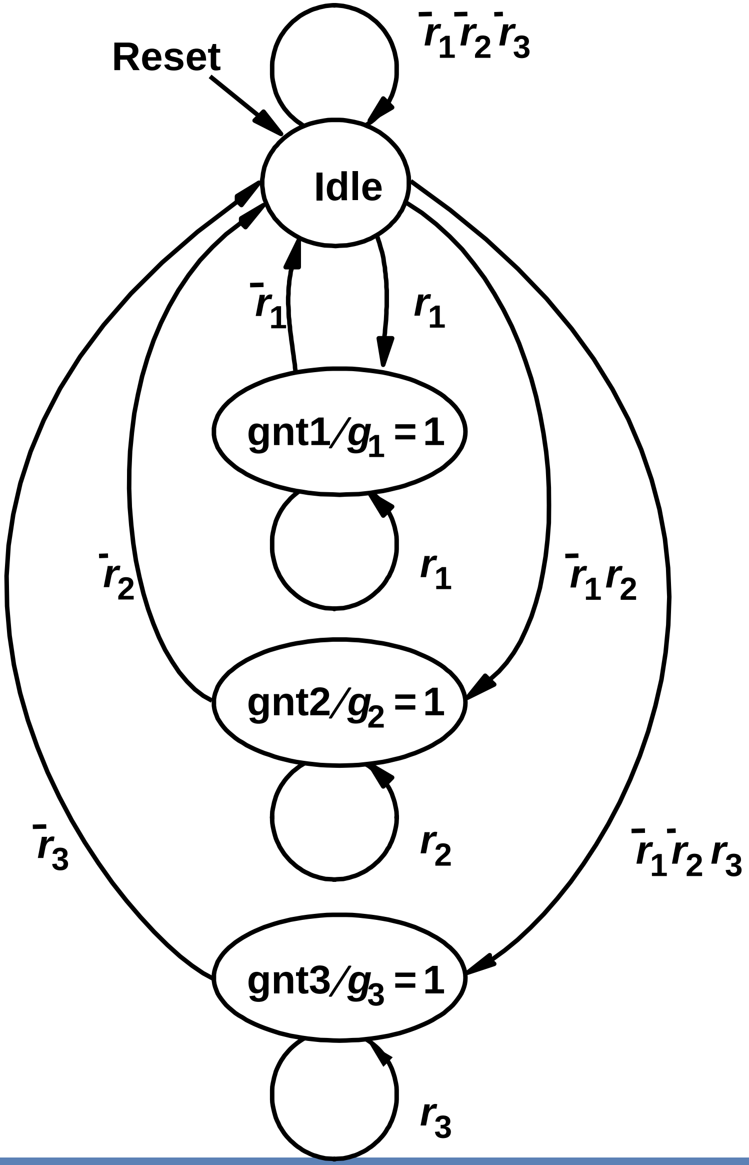
Present state $y_2y_1y_0$	Next state $Y_2Y_1Y_0$	Output $z_2z_1z_0$
000	100	000
100	010	100
010	110	010
110	001	110
001	101	001
101	011	101
011	111	011
111	000	111

Different Counter Circuit Diagram



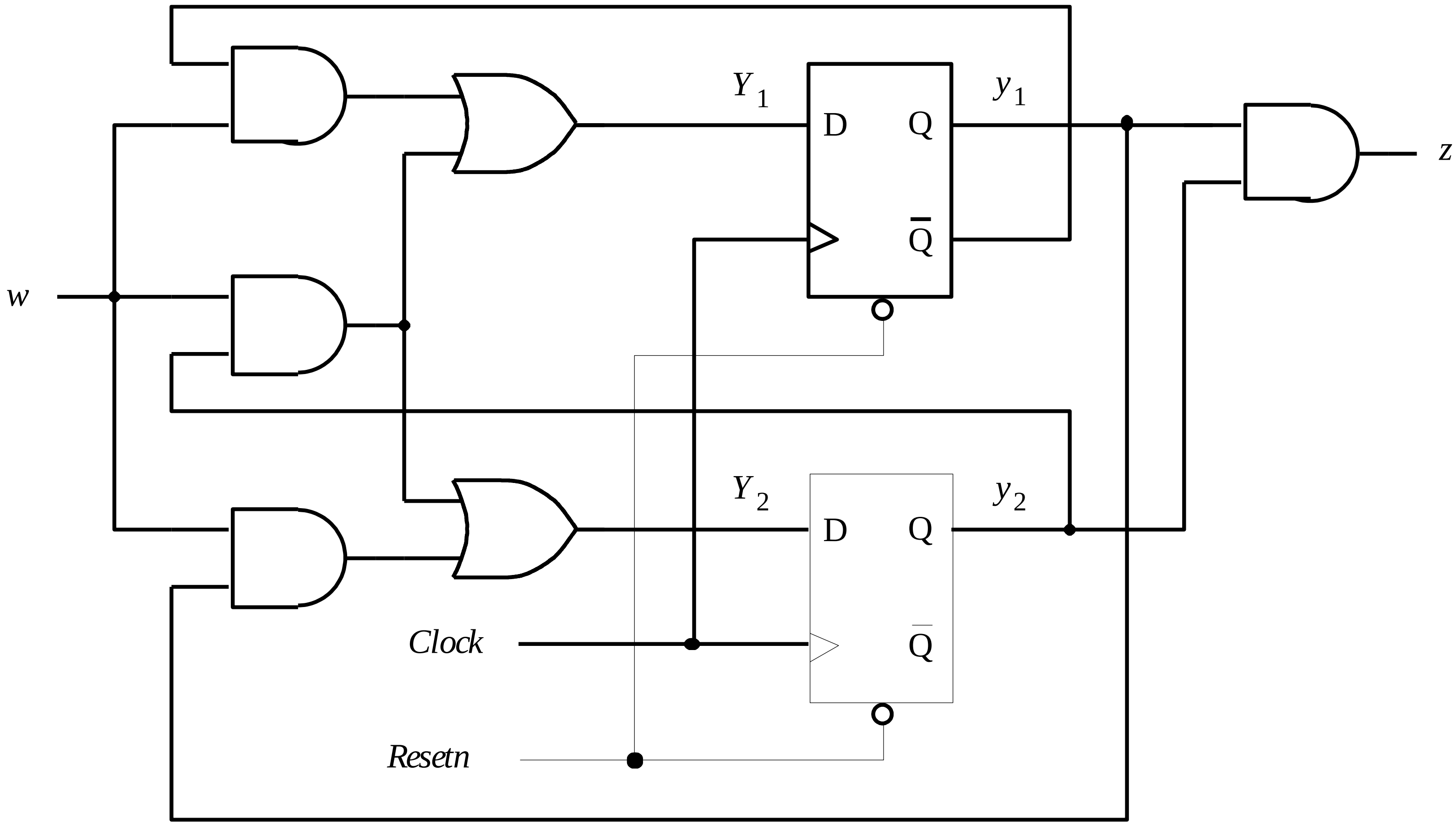
Arbiter for a 3 Device System





Ex. 8.9 Analyze the FSM

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



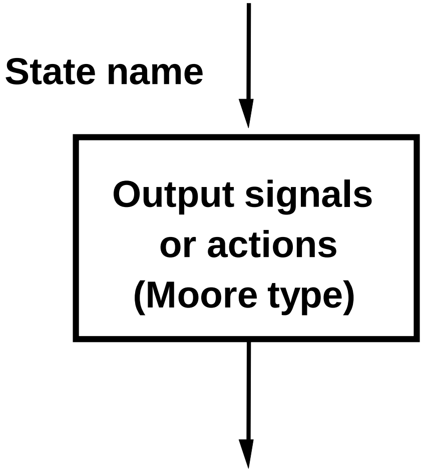
State Table

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

Present state	Next State		Output
	w = 0	w = 1	z
y_2y_1	Y_2Y_1	Y_2Y_1	
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		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

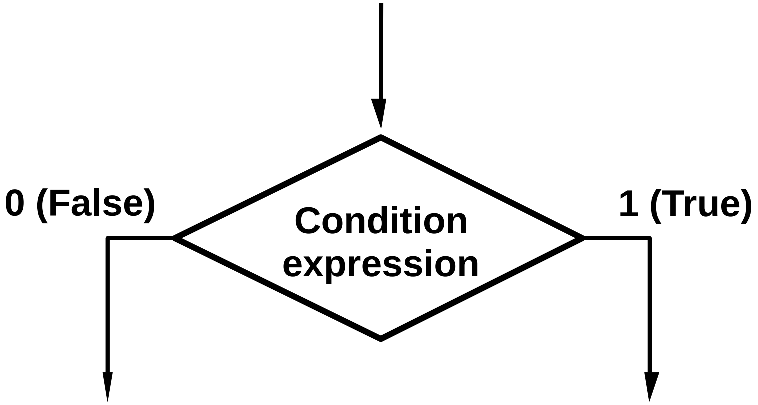
Algorithmic State Machine (ASM) Charts



State name

Output signals
or actions
(Moore type)

(a) State box

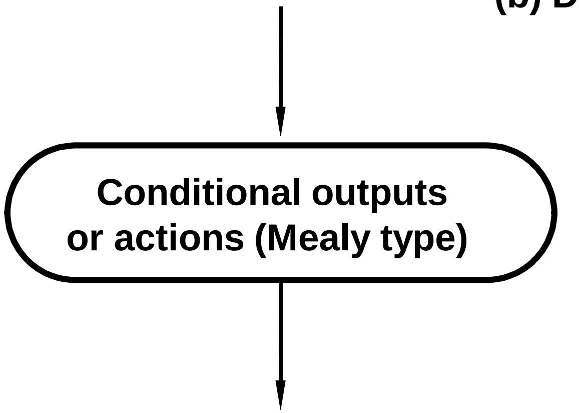


0 (False)

Condition
expression

1 (True)

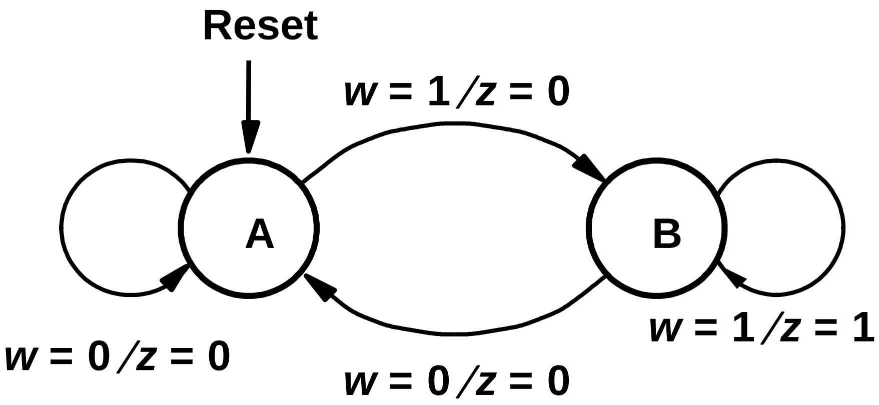
(b) Decision box

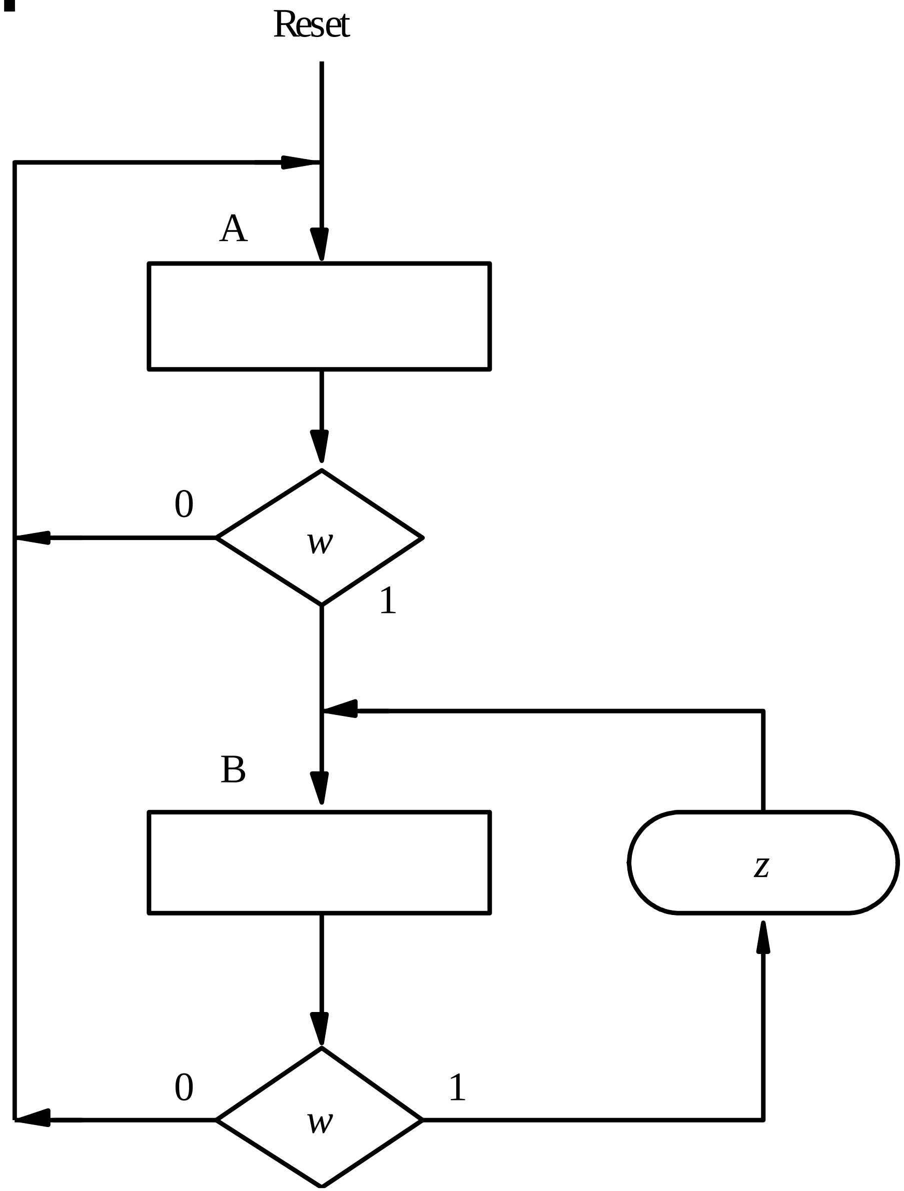


Conditional outputs
or actions (Mealy type)

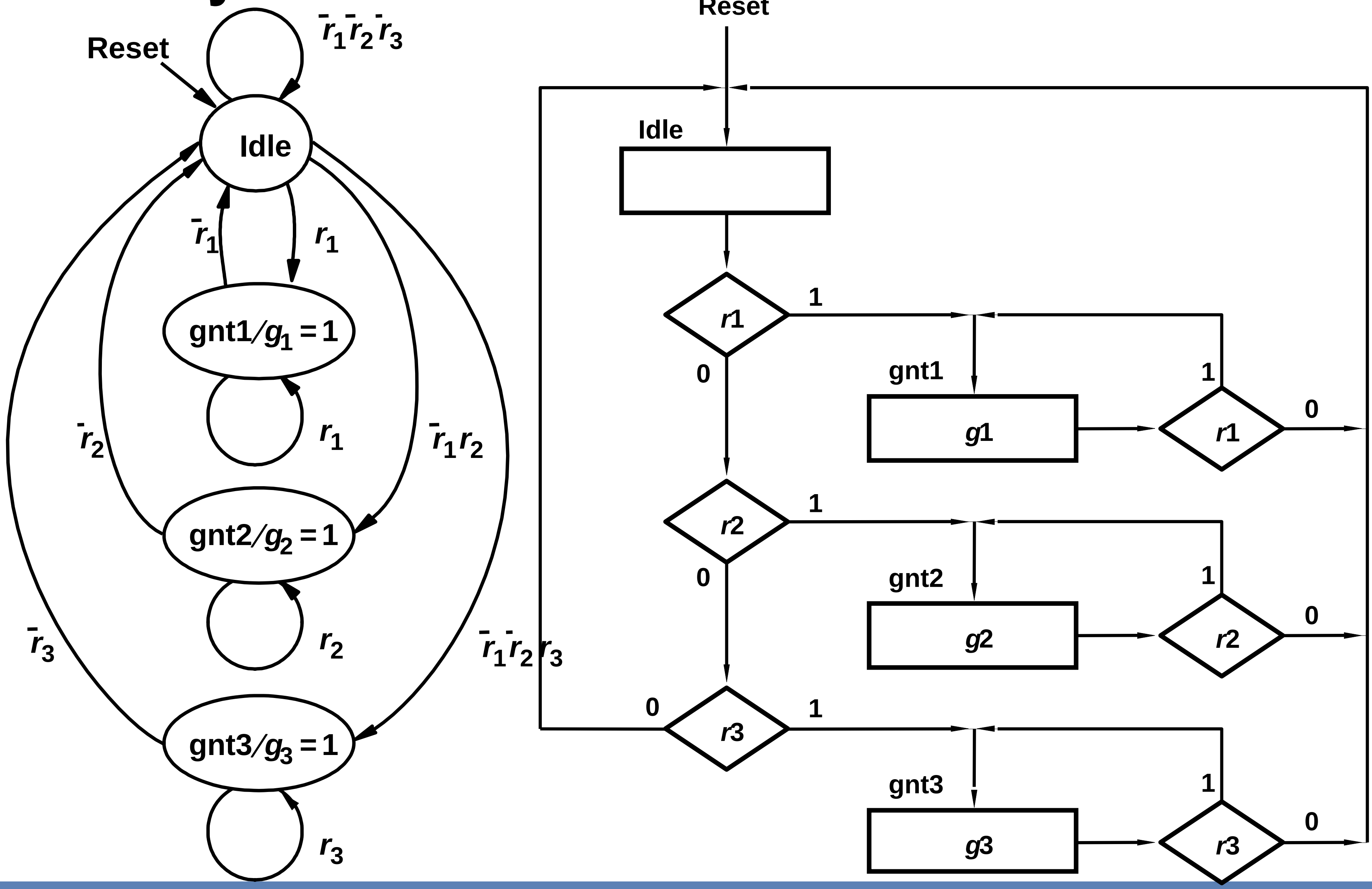
(c) Conditional output box

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

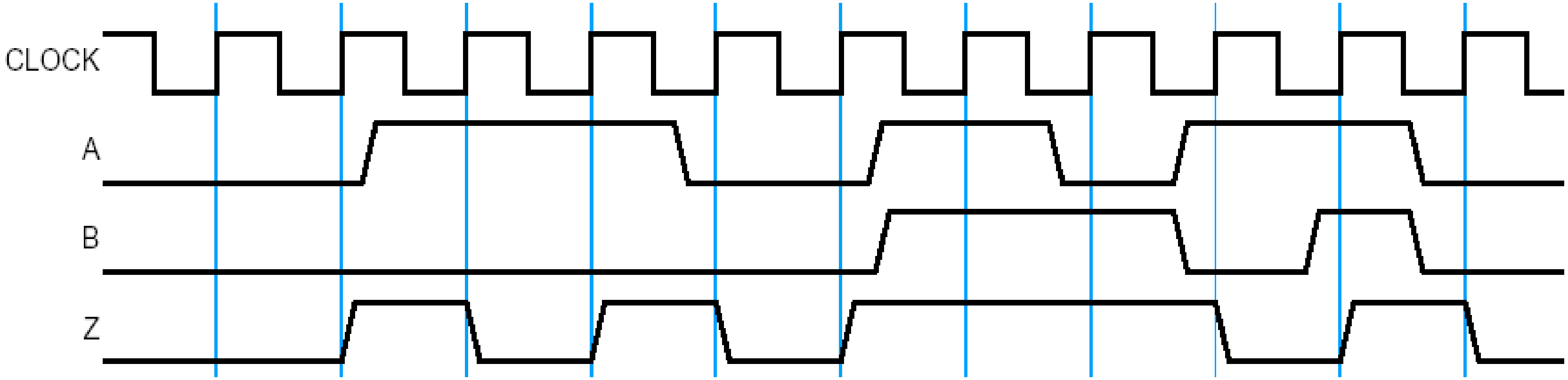
Mutual Exclusion and All Inclusion

- 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.

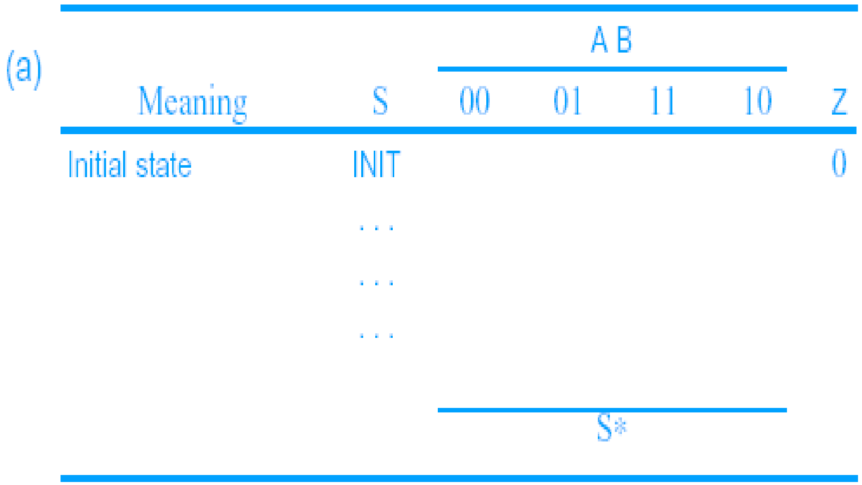
More Examples (not on our textbook)

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

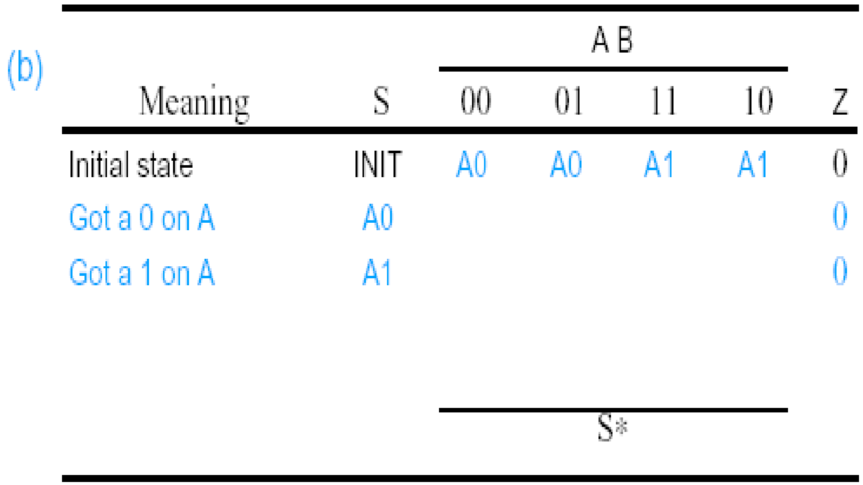


Evolution of State Table



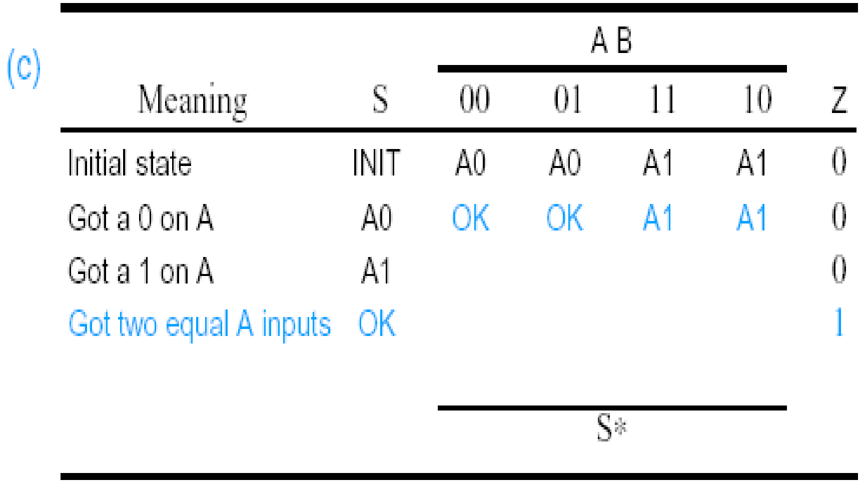
(a)

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



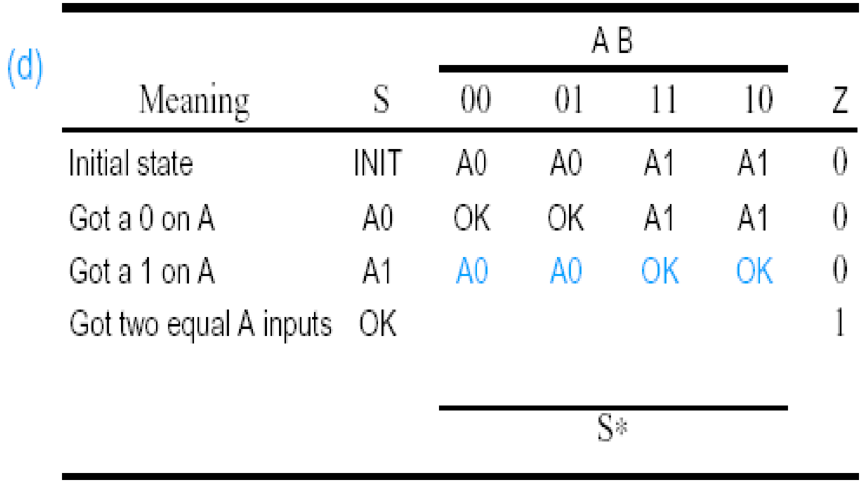
(b)

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



(c)

		A B				
Meaning	S	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					0
Got two equal A inputs	OK					1
		S*				



(d)

		A B				
Meaning	S	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					1
		S*				

Evolution of State Table (Cont')

(a)

		A B				
Meaning	S	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)

		A B				
Meaning	S	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*				

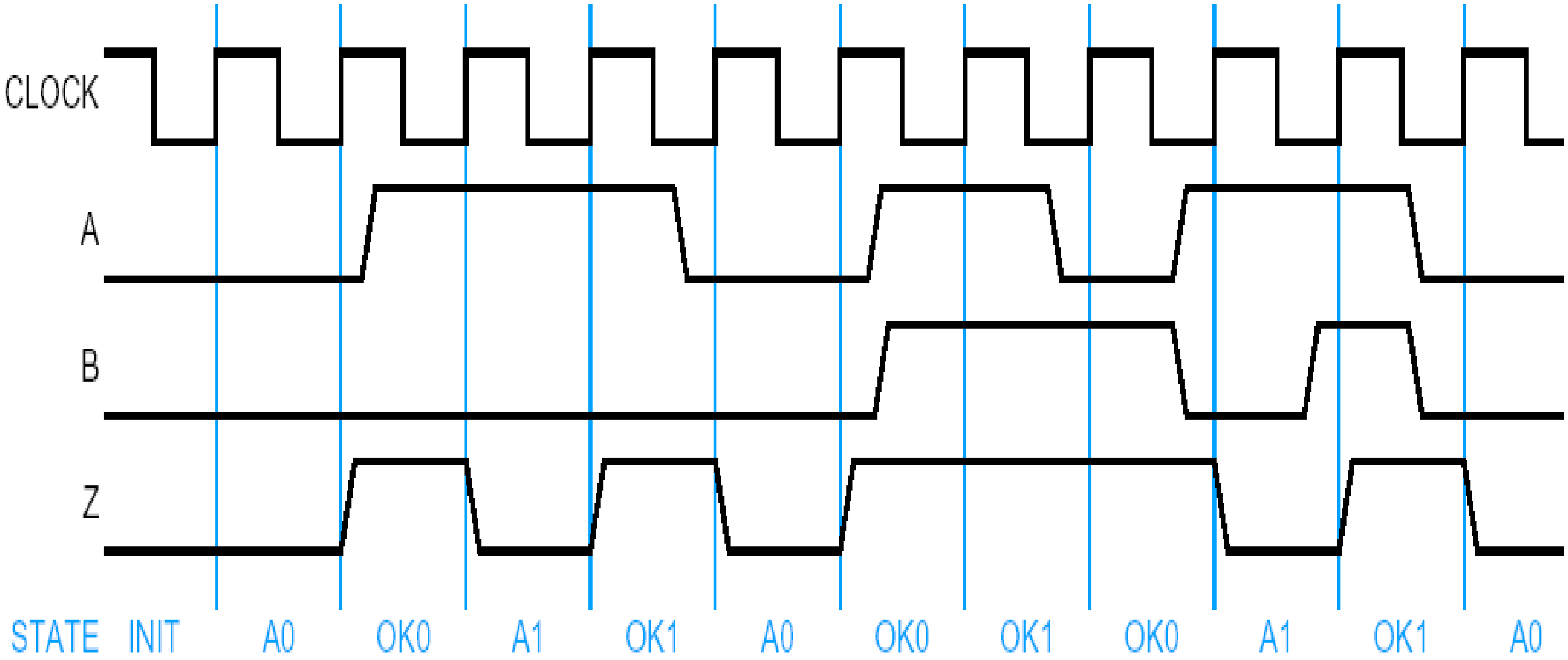
(c)

		A B				
Meaning	S	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)

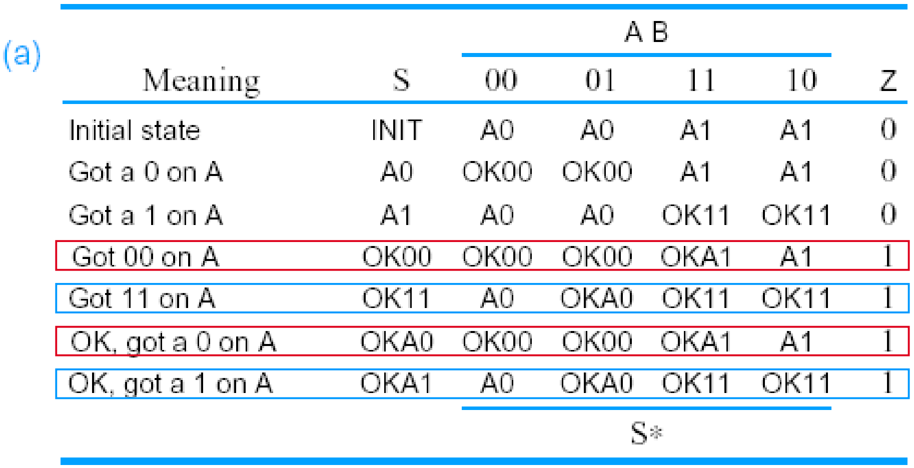
		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1
		S*				

Timing Diagram with States



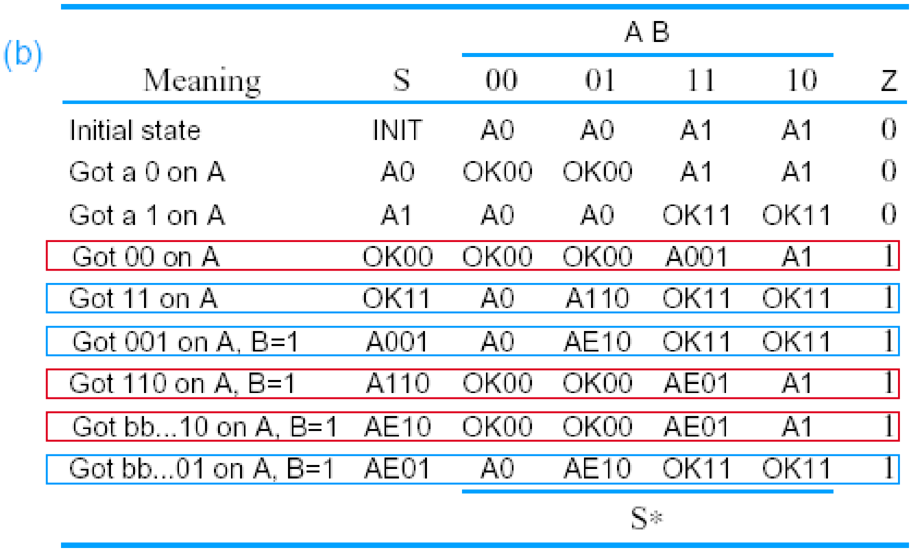
State Minimization

- Two states S1 and S2 are equivalent if two conditions are true.
  - First, S1 and S2 must produce the same values at the state-machine output(s); in the Mealy machine, this must be true for all input combinations.
  - Second, for each input combination, S1 and S2 must have tither the same next state or equivalent next states.



(a)

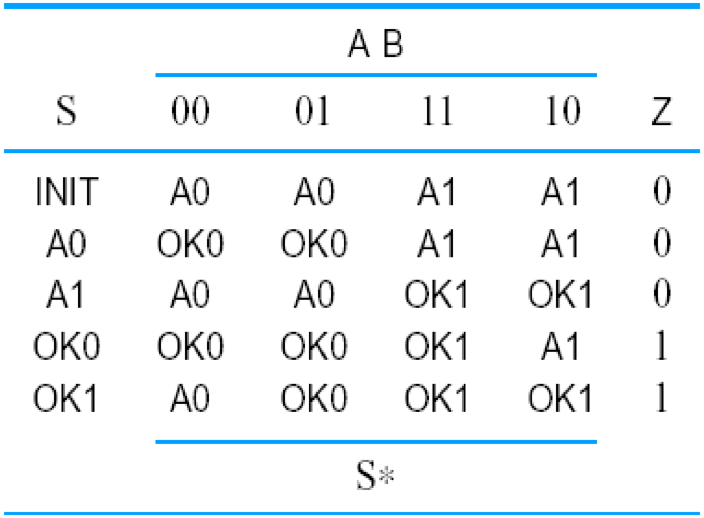
		A B				
Meaning	S	00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK00	OK00	A1	A1	0
Got a 1 on A	A1	A0	A0	OK11	OK11	0
Got 00 on A	OK00	OK00	OK00	OKA1	A1	1
Got 11 on A	OK11	A0	OKA0	OK11	OK11	1
OK, got a 0 on A	OKA0	OK00	OK00	OKA1	A1	1
OK, got a 1 on A	OKA1	A0	OKA0	OK11	OK11	1
		S*				



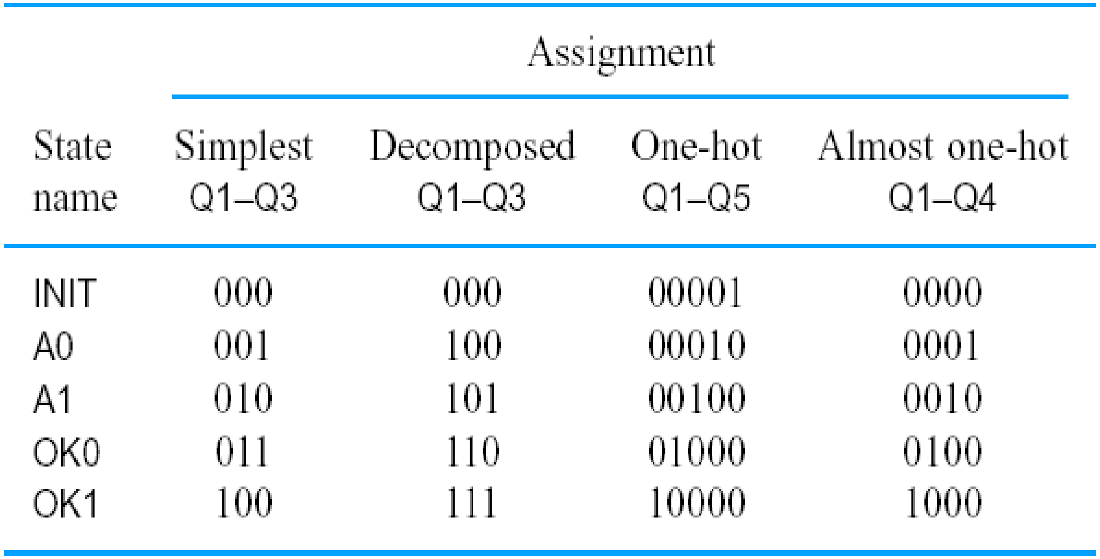
(b)

		A B				
Meaning	S	00	01	11	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*				



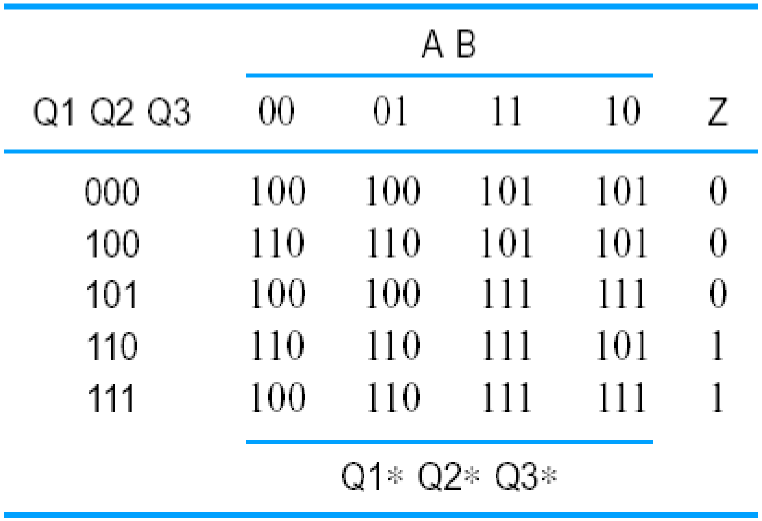
	Assignment			
State name	Simplest Q1–Q3	Decomposed Q1–Q3	One-hot Q1–Q5	Almost one-hot Q1–Q4
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

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.

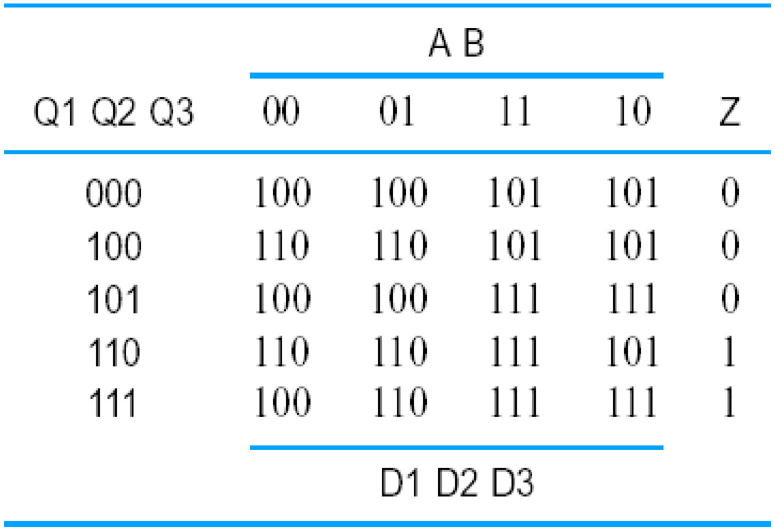
Transition/ Excitation Table

- Transition/Output table.



	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
	Q1* Q2* Q3*				

Excitation/Output table.

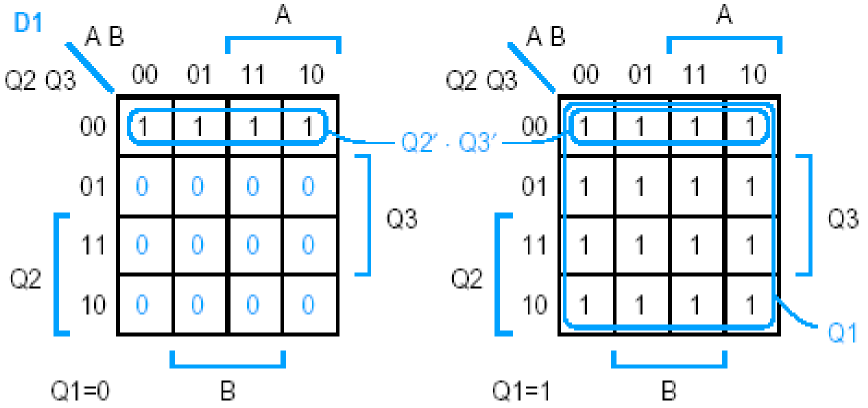


	A B				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1 D2 D3				

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				



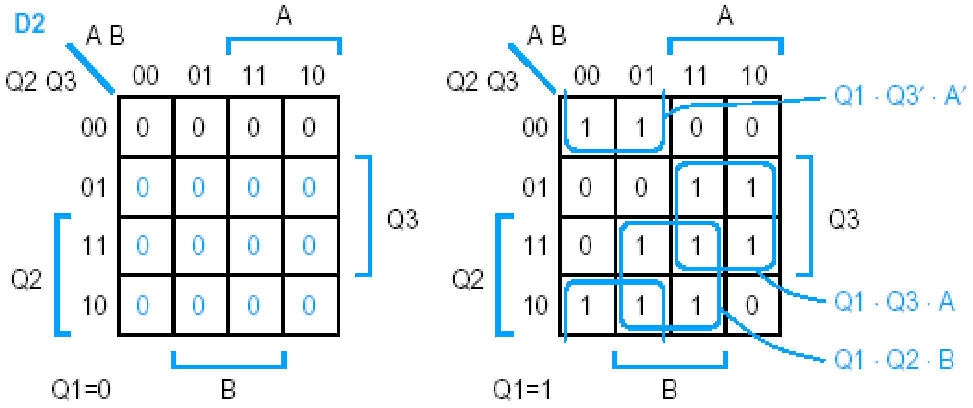
Q1=0

Q2 Q3 \ A B	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

Q1=1

Q2 Q3 \ A B	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

D1= Q1 + Q2’·Q3’

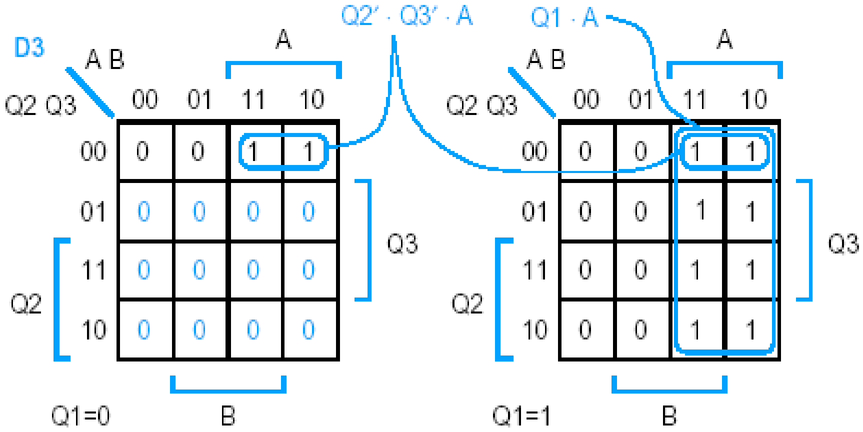


Q1=0

Q2 Q3 \ A B	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

Q1=1

Q2 Q3 \ A B	00	01	11	10
00	1	1	0	0
01	0	0	1	1
11	0	1	1	1
10	1	1	1	0



Q1=0

Q2 Q3 \ A B	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

Q1=1

Q2 Q3 \ A B	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

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

Minimal Risk Logic Equation

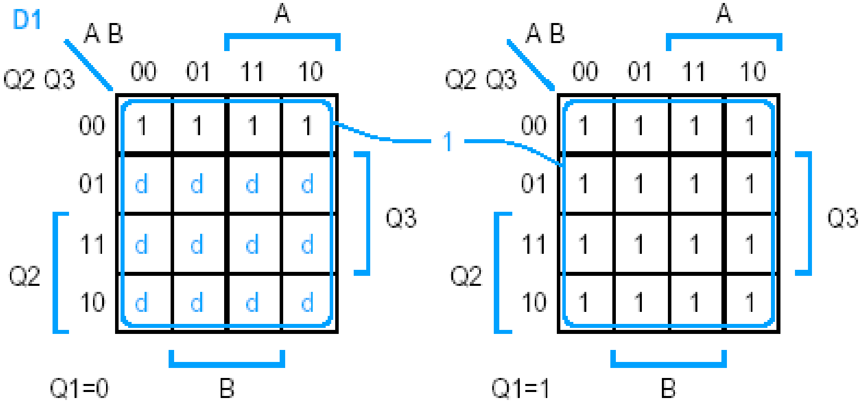
- Excitation equations
  - $D1 = Q1 + Q2' \cdot Q3'$
  - $D2 = Q1 \cdot Q3' \cdot A' + Q1 \cdot Q3 \cdot A + Q1 \cdot Q2 \cdot B$
  - $D3 = Q1 \cdot A + Q2 \cdot {}'Q3' \cdot A$
- Output Equations (Z =1 for 110 and 111)
  - $Z = Q1 \cdot Q2 \cdot Q3' + Q1 \cdot Q2 \cdot Q3$

    $= Q1 \cdot Q2$

Minimal Cost Excitation Map

- Next-state of unused states are “don’t care”.
- Output equation Z = Q2.

	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				



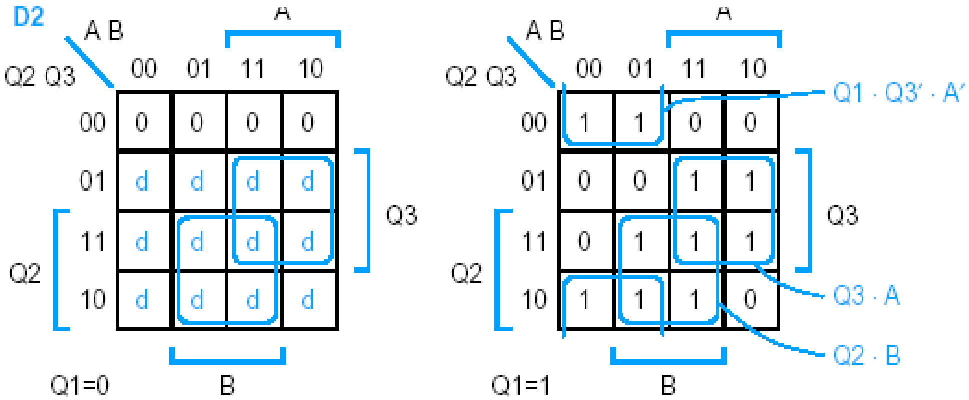
Q1=0

Q2 Q3 \ A B	00	01	11	10
00	1	1	1	1
01	d	d	d	d
11	d	d	d	d
10	d	d	d	d

Q1=1

Q2 Q3 \ A B	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

D1 = 1



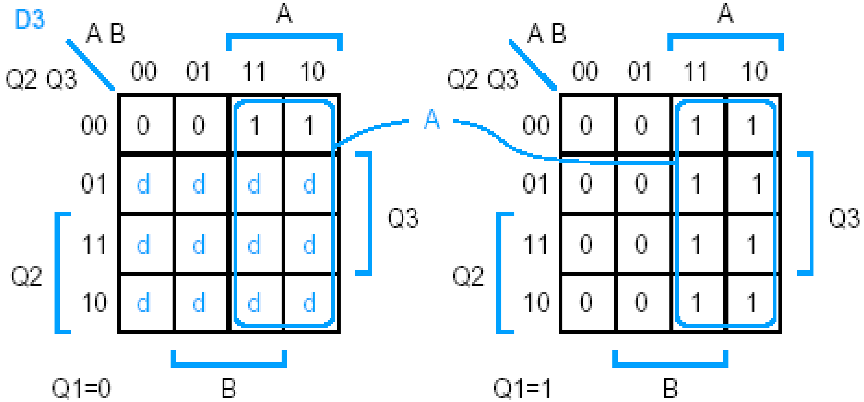
Q1=0

Q2 Q3 \ A B	00	01	11	10
00	0	0	0	0
01	d	d	d	d
11	d	d	d	d
10	d	d	d	d

Q1=1

Q2 Q3 \ A B	00	01	11	10
00	1	1	0	0
01	0	0	1	1
11	0	1	1	1
10	1	1	1	0

D2= Q1·Q3’ ·A’+Q3·A+Q2·B



Q1=0

Q2 Q3 \ A B	00	01	11	10
00	0	0	1	1
01	d	d	d	d
11	d	d	d	d
10	d	d	d	d

Q1=1

Q2 Q3 \ A B	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

D3= A