Digital Logic

Sanjeeb Prasad Panday (Ph.D.)

Assistant Professor

Department of Electronics and Computer Engineering , Pulchowk Campus

Tribhuvan University

Sequential Logic
Counters and Registers

Registers

- Introduction: Registers
  - Simple Registers
  - Registers with Parallel Load
- Using Registers to implement Sequential Circuits
- Shift Registers
  - Serial In/Serial Out Shift Registers
  - Serial In/Parallel Out Shift Registers
  - Parallel In/Serial Out Shift Registers
  - Parallel In/Parallel Out Shift Registers

Sequential Logic
Counters and Registers

- Bidirectional Shift Registers
- An Application – Serial Addition

Sequential Logic
Counters and Registers

Counters

- Introduction: Counters
- Asynchronous (Ripple) Counters
- Asynchronous Counters with MOD number < 2^n
- Asynchronous Down Counters
- Cascading Asynchronous Counters

Sequential Logic
Counters and Registers

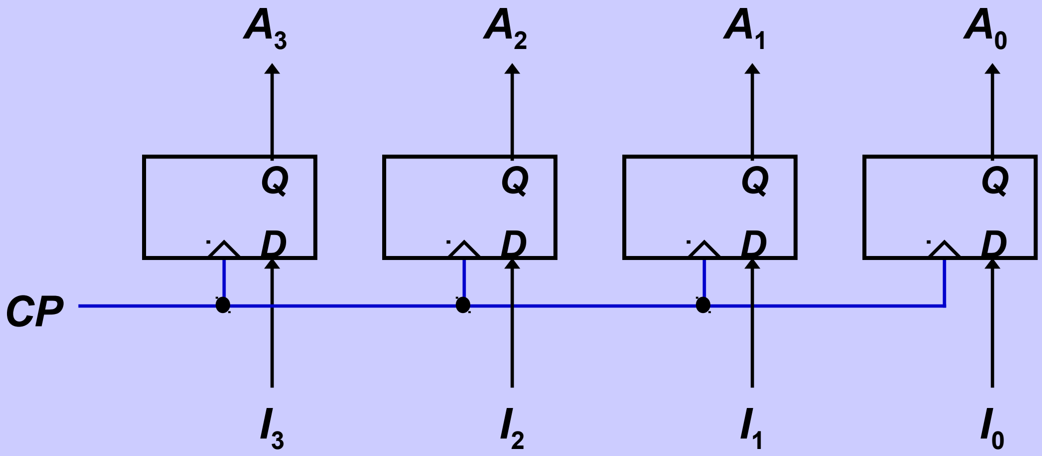
- Synchronous (Parallel) Counters
- Up/Down Synchronous Counters
- Designing Synchronous Counters
- Decoding A Counter
- Counters with Parallel Load
- Shift Register Counters
  - Ring Counters
  - Johnson Counters
- Random-Access Memory (RAM)

Introduction: Registers

- An *n*-bit register has a group of *n* flip-flops and some logic gates and is capable of storing *n* bits of information.
- The flip-flops store the information while the gates control when and how new information is transferred into the register.
- Some functions of register:
  - retrieve data from register
  - store/load new data into register (serial or parallel)
  - shift the data within register (left or right)

Simple Registers

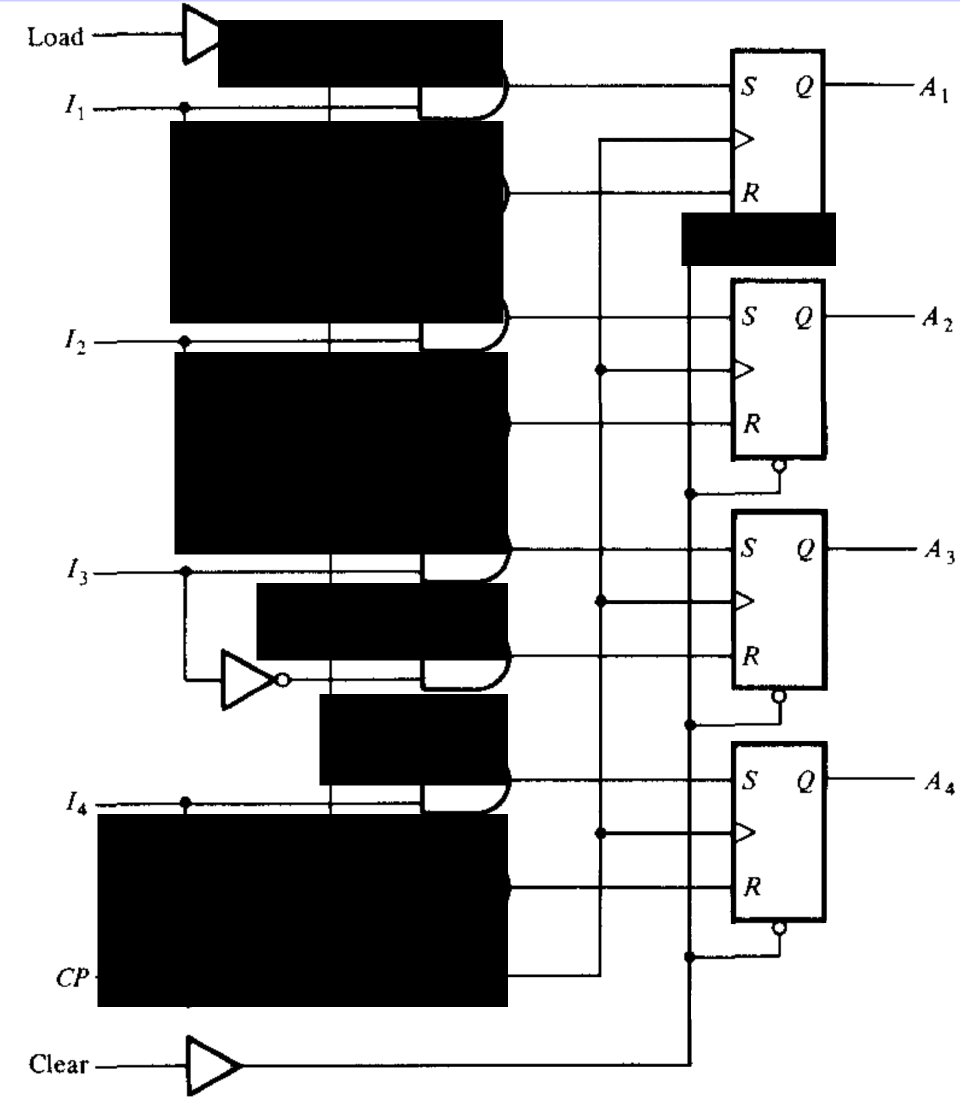
- No external gates.
- Example: A 4-bit register. A new 4-bit data is loaded every clock cycle.



Registers With Parallel Load

- Instead of loading the register at every clock pulse, we may want to control when to load.
- *Loading* a register: transfer new information into the register. Requires a *load* control input.
- *Parallel loading*: all bits are loaded simultaneously.

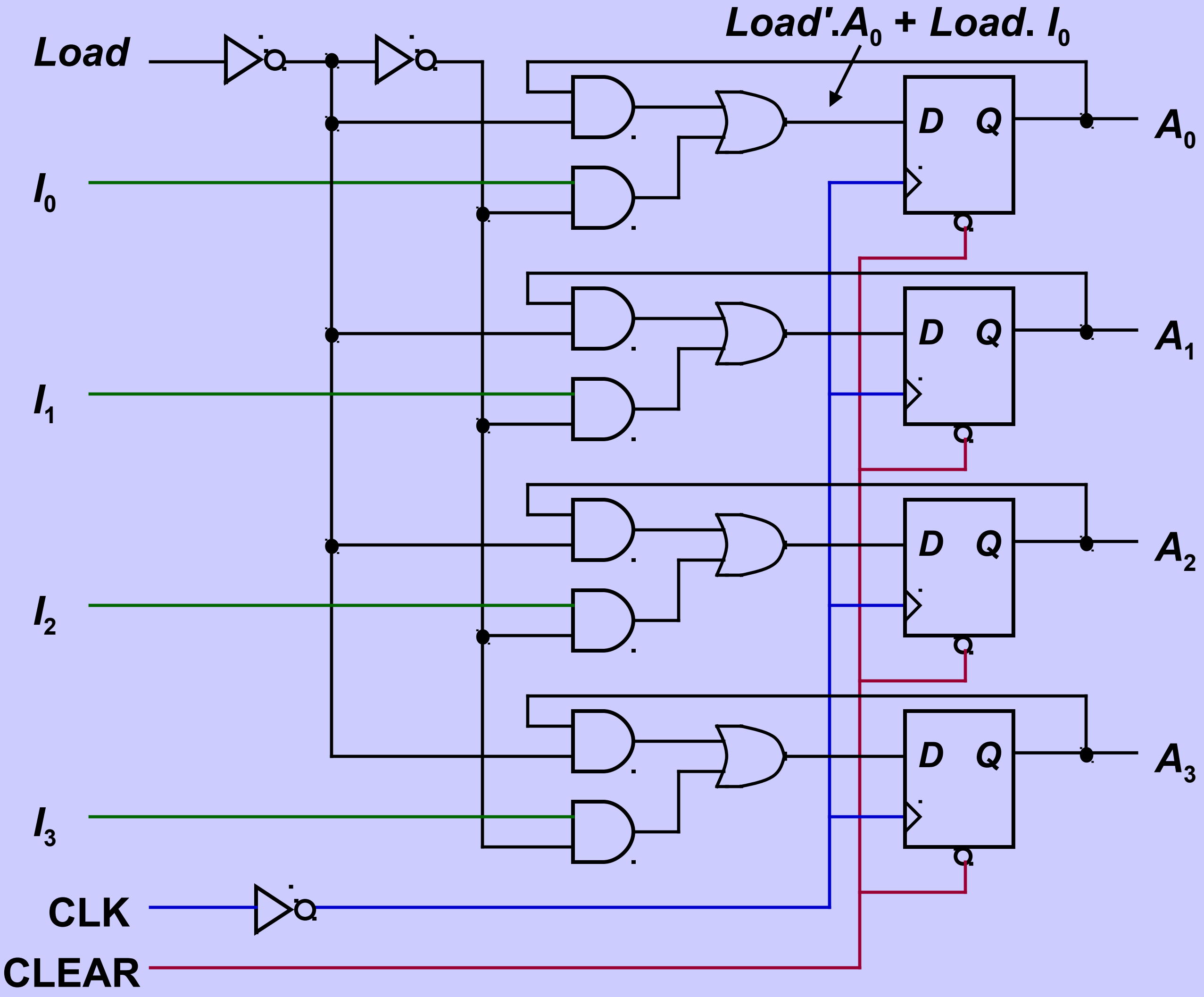
Registers With Parallel Load



[redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted]

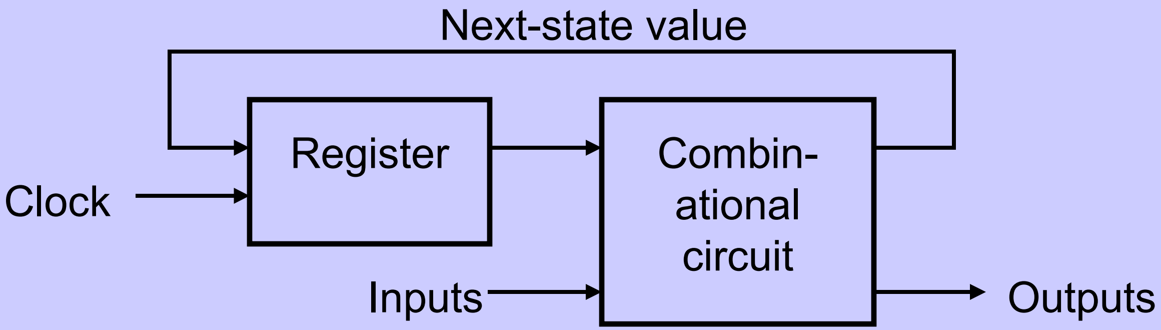
4-bit Register with parallel load

Registers With Parallel Load



Using Registers to implement Sequential Circuits

- A sequential circuit may consist of a *register* (memory) and a *combinational circuit*.



- The external inputs and present states of the register determine the next states of the register and the external outputs, through the combinational circuit.
- The combinational circuit may be implemented by any of the methods covered in *MSI components* and *Programmable Logic Devices*.

Using Registers to implement Sequential Circuits

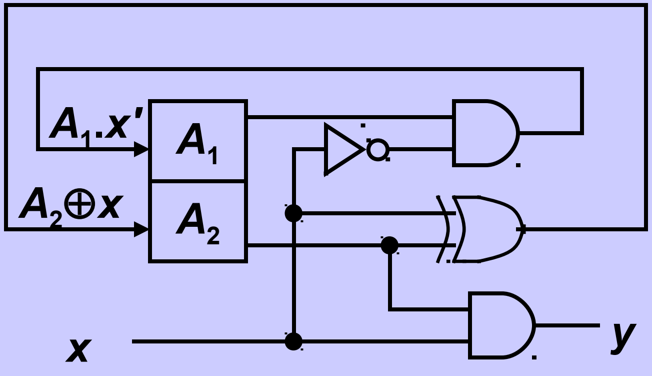
- Example 1:

$A_1^+ = \Sigma\, m(4,6) = A_1.x'$

$A_2^+ = \Sigma\, m(1,2,5,6) = A_2.x' + A_2'.x = A_2 \oplus x$

$y = \Sigma\, m(3,7) = A_2.x$

Present state		Input	Next State		Output
A_1	A_2	x	A_1^+	A_2^+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

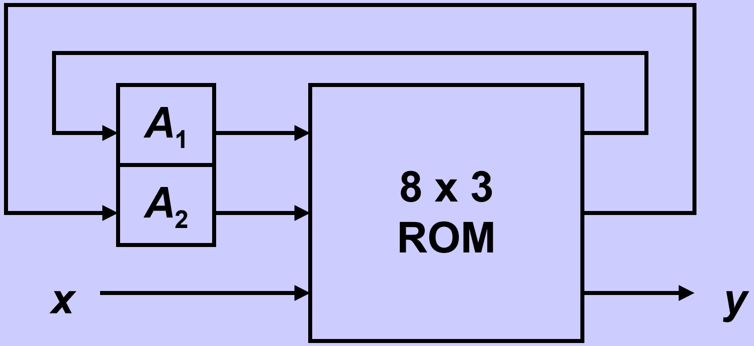


Using Registers to implement Sequential Circuits

- Example 2: Repeat example 1, but use a ROM.

Address			Outputs		
1	2	3	1	2	3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

ROM truth table

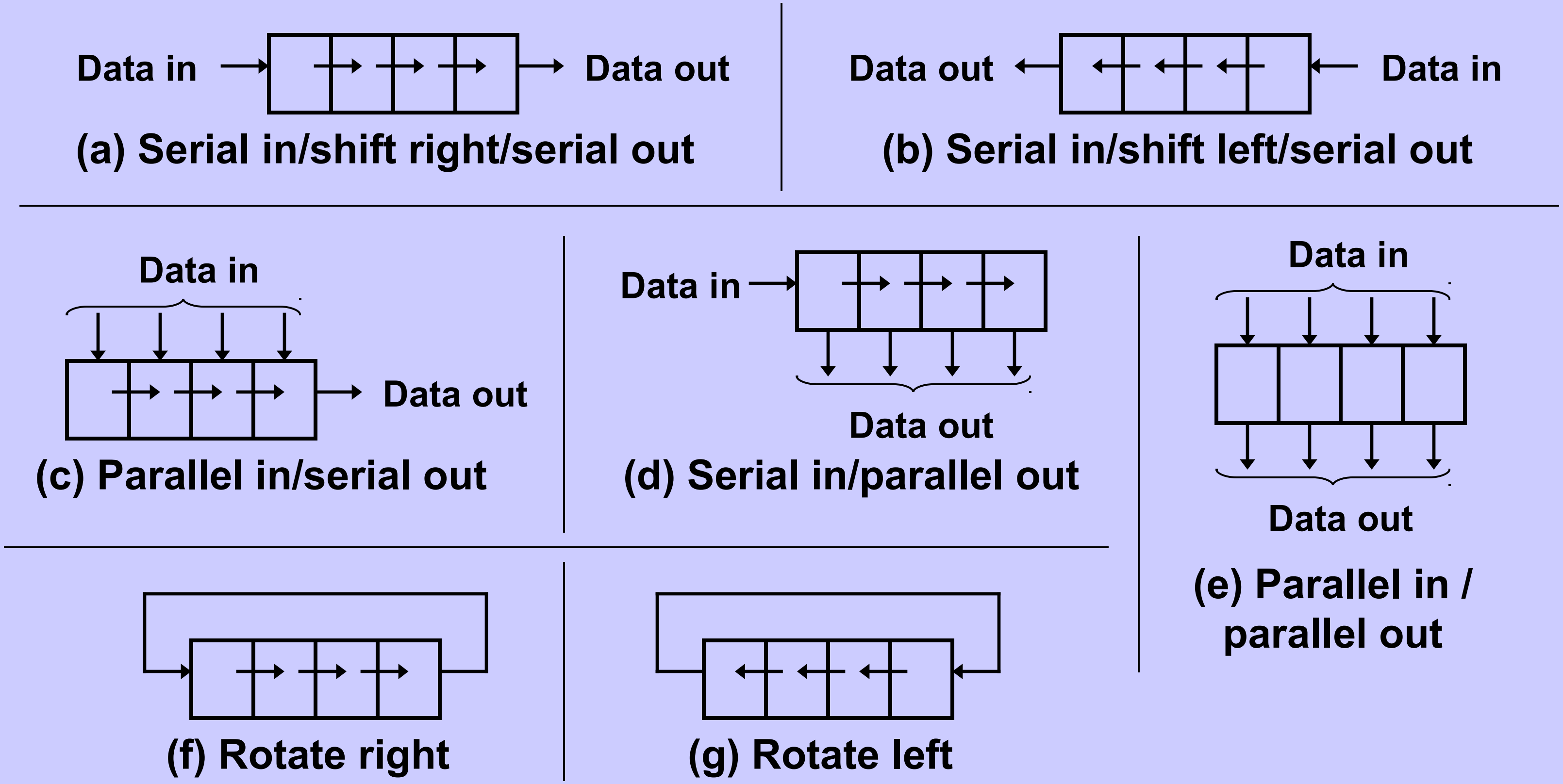


Shift Registers

- Another function of a register, besides storage, is to provide for *data movements*.
- Each *stage* (flip-flop) in a shift register represents one bit of storage, and the shifting capability of a register permits the movement of data from stage to stage within the register, or into or out of the register upon application of clock pulses.

Shift Registers

- Basic data movement in shift registers (four bits are used for illustration).



Data in
Data out

(a) Serial in/shift right/serial out

Data out
Data in

(b) Serial in/shift left/serial out

Data in
Data out

(c) Parallel in/serial out

Data in
Data out

(d) Serial in/parallel out

Data in
Data out

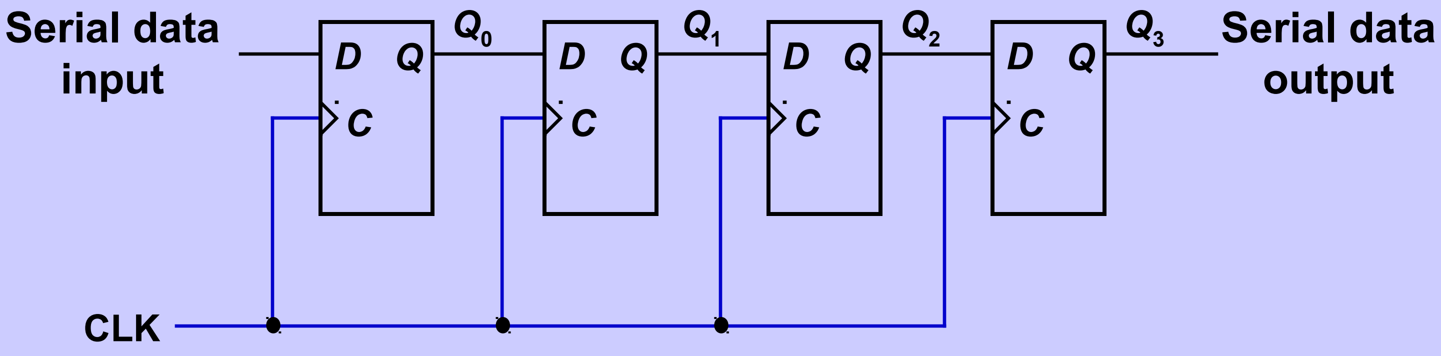
(e) Parallel in / parallel out

(f) Rotate right

(g) Rotate left

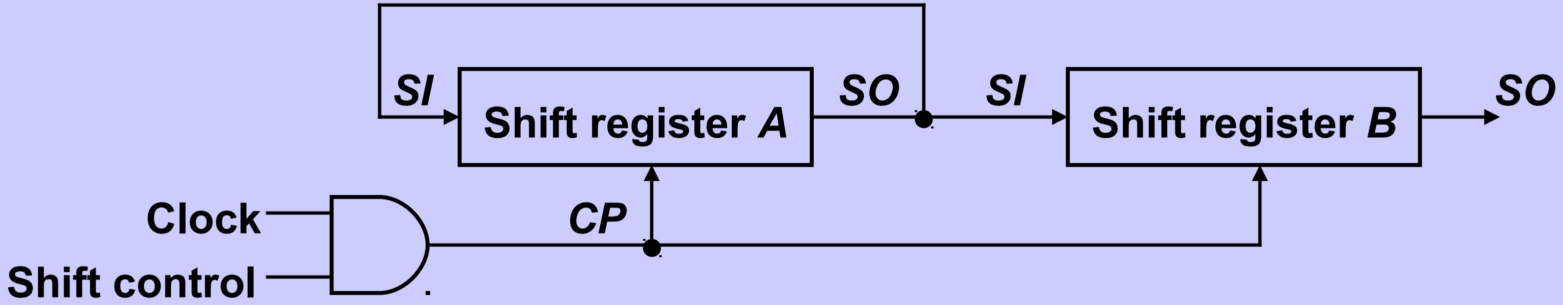
Serial In/Serial Out Shift Registers

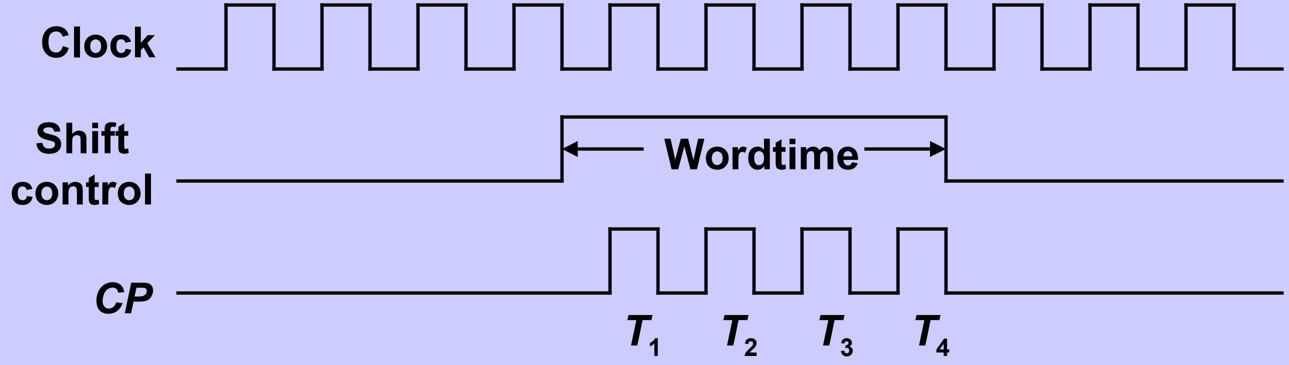
- Accepts data serially – one bit at a time – and also produces output serially.



Serial In/Serial Out Shift Registers

- Application: Serial transfer of data from one register to another.





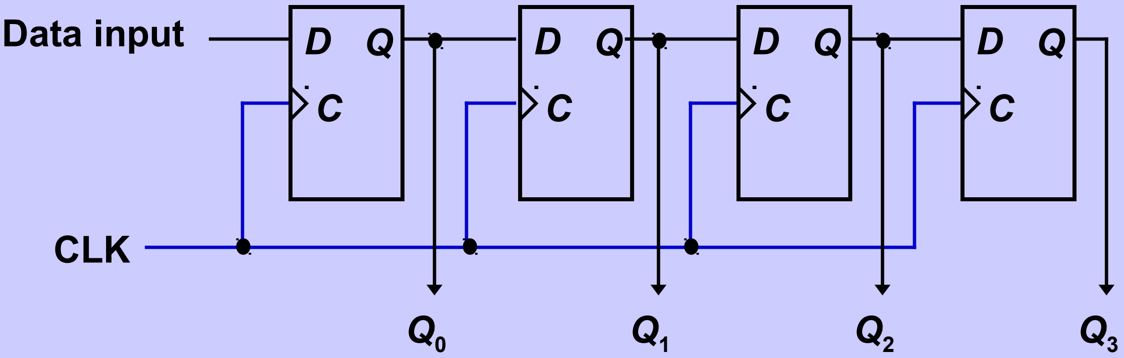
Serial In/Serial Out Shift Registers

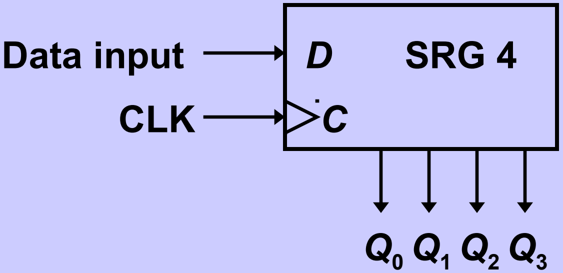
- Serial-transfer example.

Timing Pulse	Shift register *A*				Shift register *B*				Serial output of *B*
Initial value	1	0	1	1	0	0	1	0	0
After T_1	1	1	0	1	1	0	0	1	1
After T_2	1	1	1	0	1	1	0	0	0
After T_3	0	1	1	1	0	1	1	0	0
After T_4	1	0	1	1	1	0	1	1	1

Serial In/Parallel Out Shift Registers

- Accepts data serially.
- Outputs of all stages are available simultaneously.

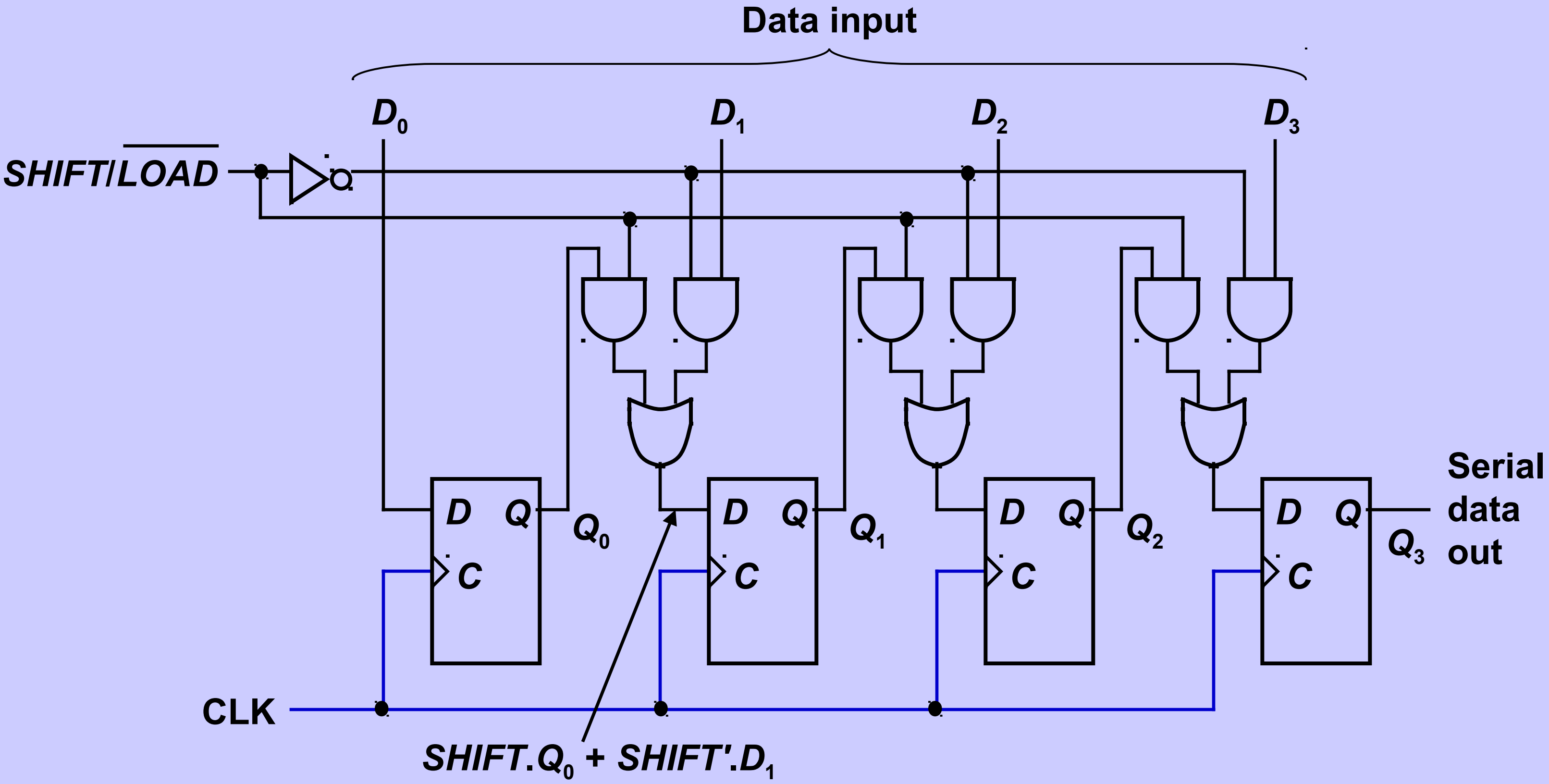




Logic symbol

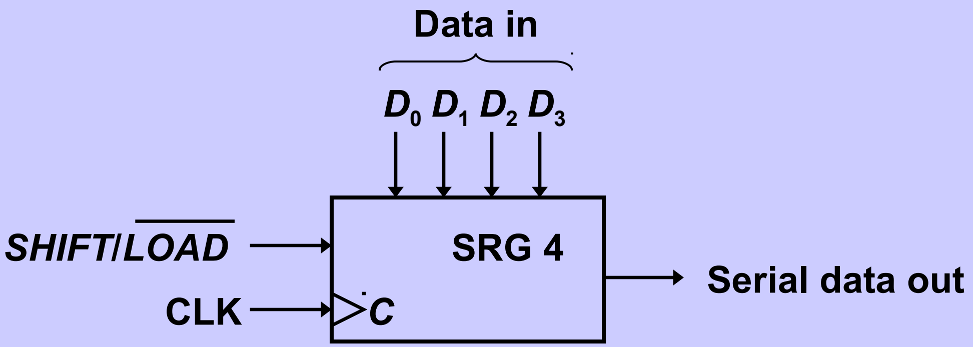
Parallel In/Serial Out Shift Registers

- Bits are entered simultaneously, but output is serial.



Parallel In/Serial Out Shift Registers

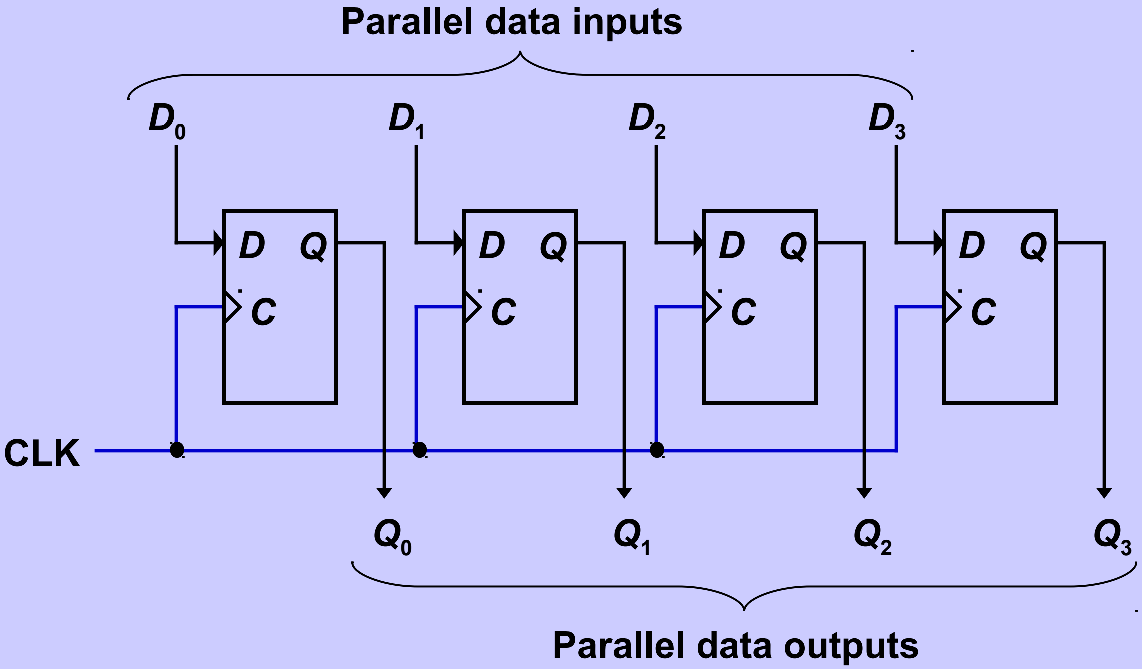
- Bits are entered simultaneously, but output is serial.



Logic symbol

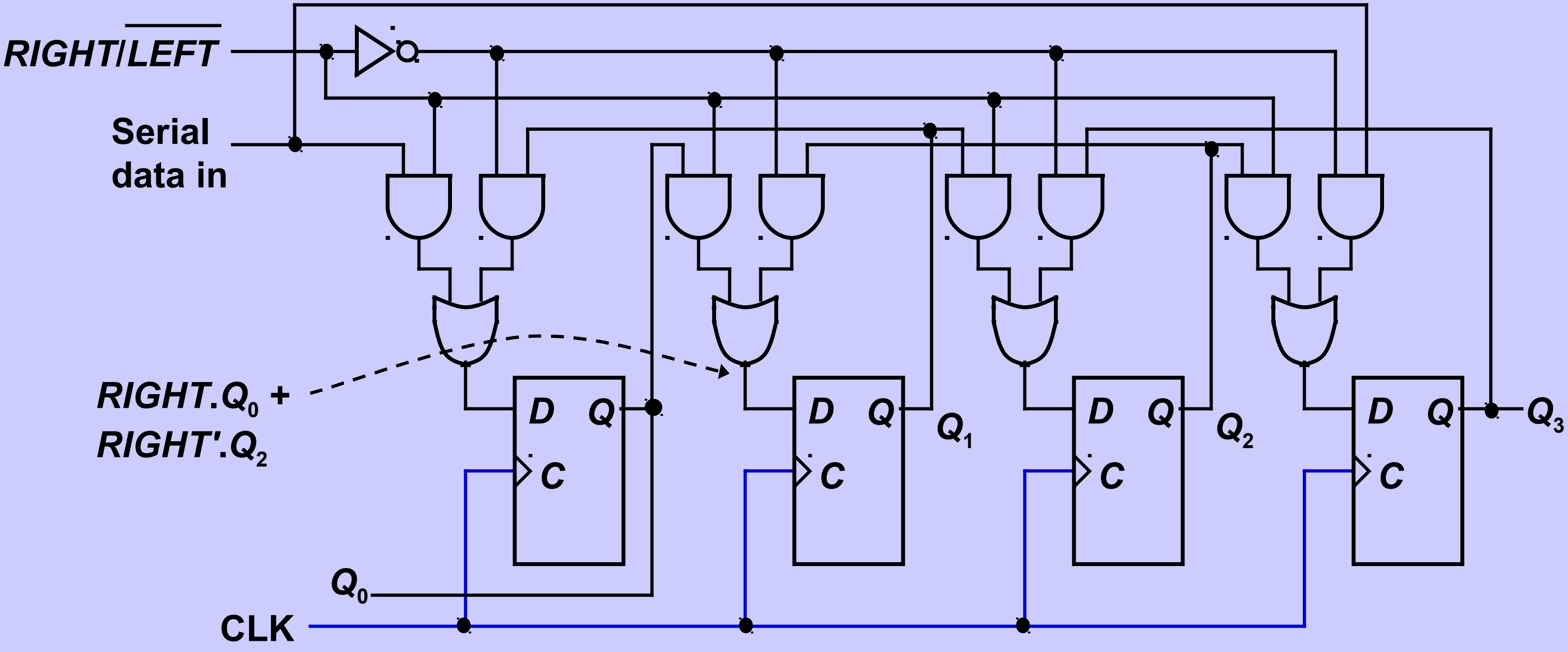
Parallel In/Parallel Out Shift Registers

- Simultaneous input and output of all data bits.



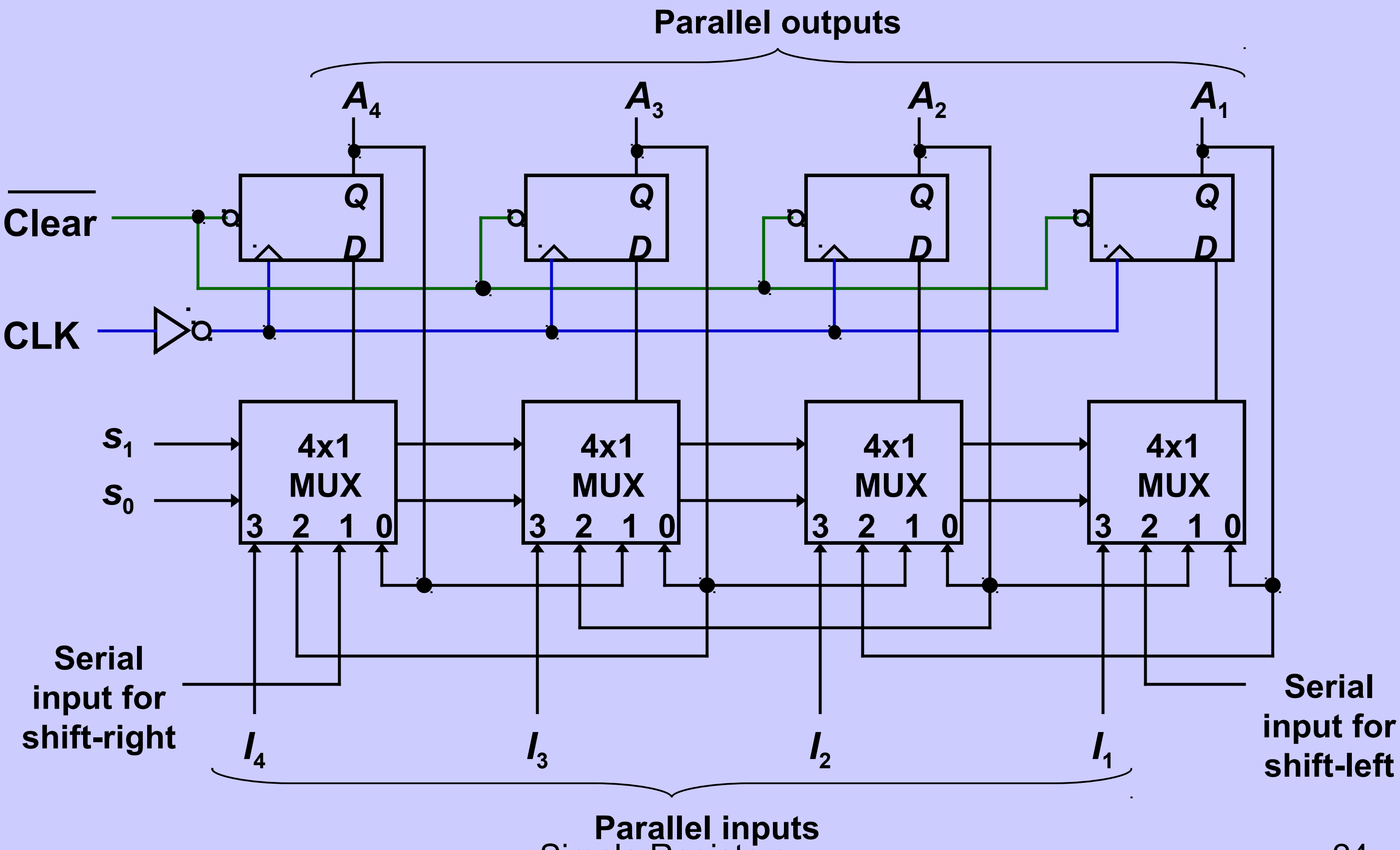
Bidirectional Shift Registers

- Data can be shifted either left or right, using a control line *RIGHT/$\overline{LEFT}$* (or simply *RIGHT*) to indicate the direction.



Bidirectional Shift Registers

- 4-bit bidirectional shift register with parallel load.



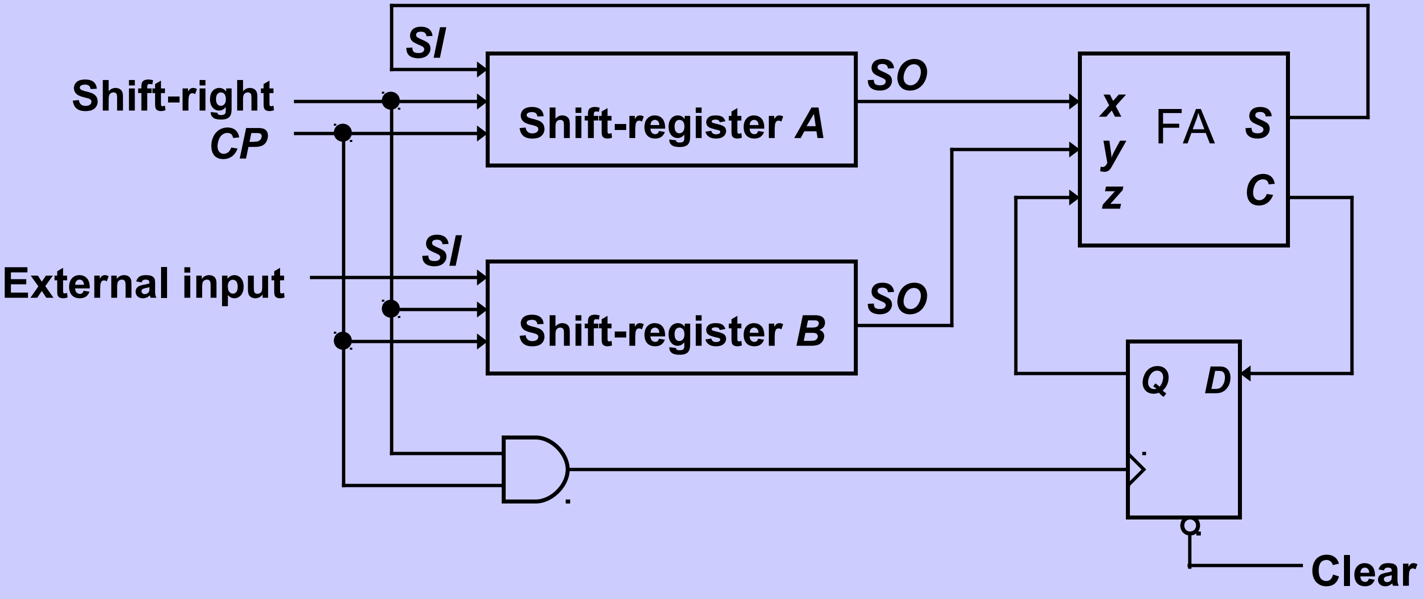
Bidirectional Shift Registers

- 4-bit bidirectional shift register with parallel load.

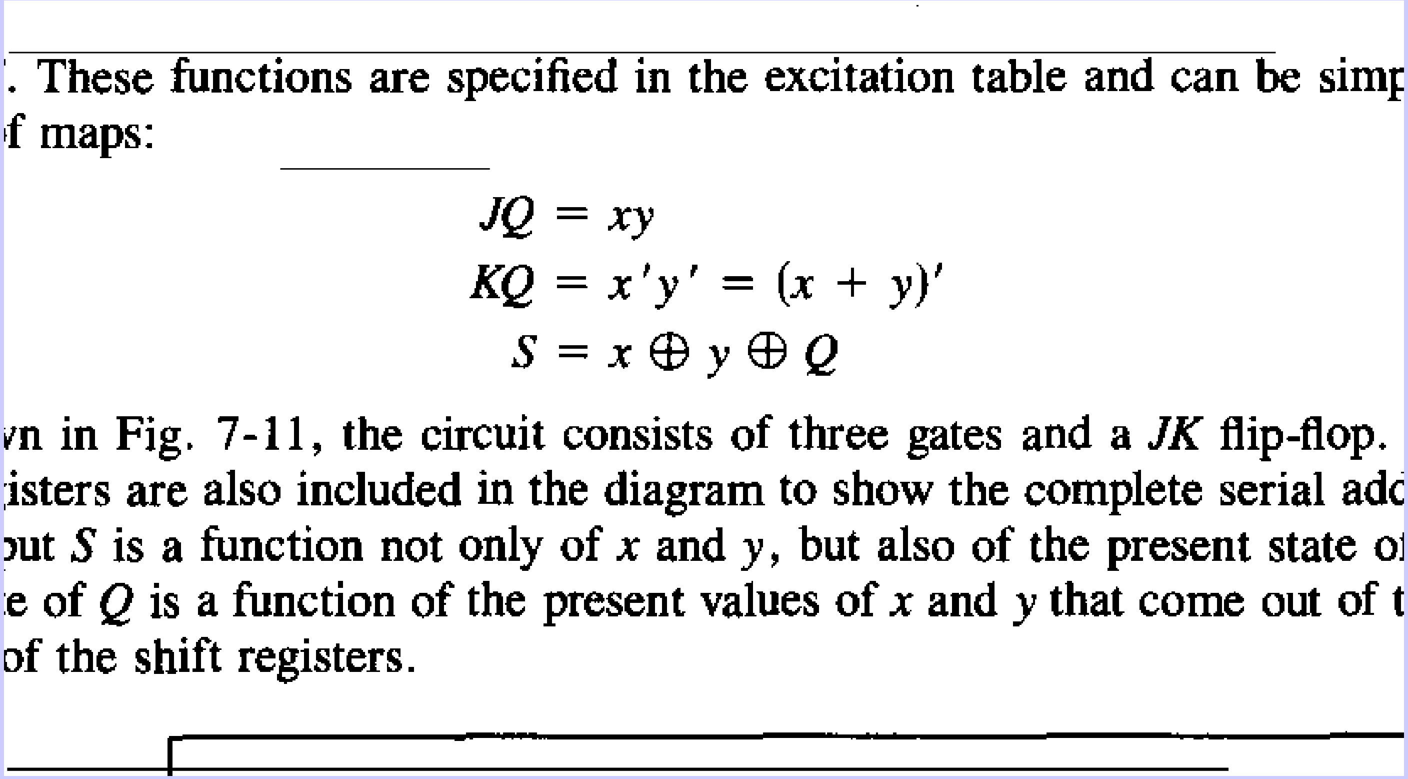
Mode Control		
s_1	s_0	*Register Operation*
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

An Application – Serial Addition

- Most operations in digital computers are done in parallel. Serial operations are slower but require less equipment.
- A serial adder is shown below. *A* ← *A* + *B*.



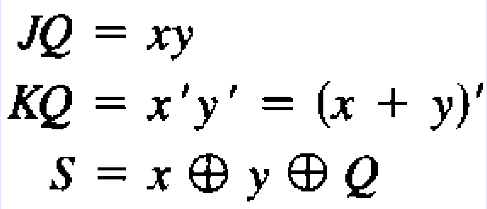
Serial Adder



. These functions are specified in the excitation table and can be simp f maps:

$$JQ = xy$$
$$KQ = x'y' = (x + y)'$$
$$S = x \oplus y \oplus Q$$

vn in Fig. 7-11, the circuit consists of three gates and a *JK* flip-flop. isters are also included in the diagram to show the complete serial add out *S* is a function not only of *x* and *y*, but also of the present state o e of *Q* is a function of the present values of *x* and *y* that come out of t of the shift registers.



$$JQ = xy$$
$$KQ = x'y' = (x + y)'$$
$$S = x \oplus y \oplus Q$$

Serial Adder

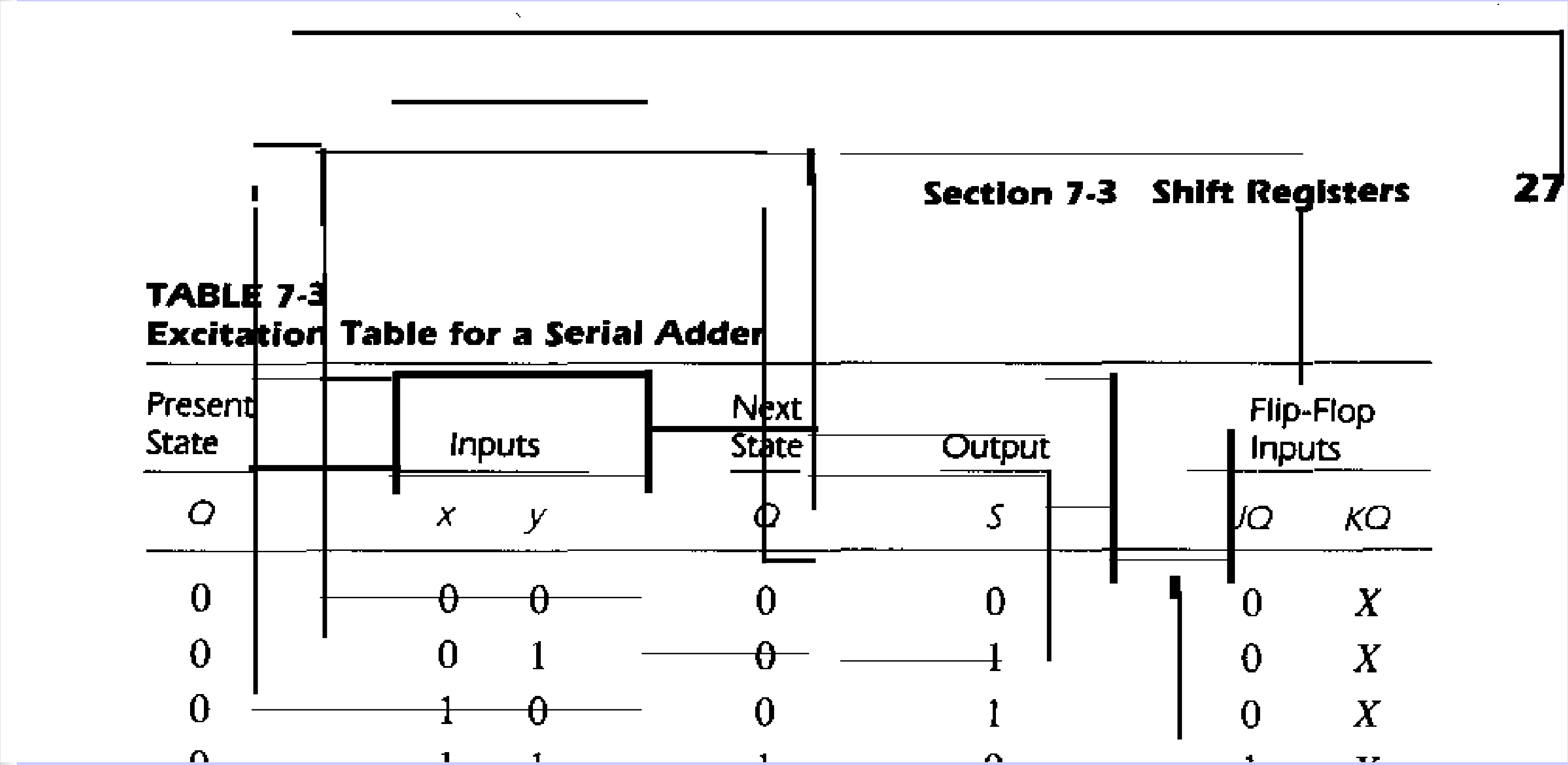


TABLE 7-3
Excitation Table for a Serial Adder

Present State	Inputs		Next State	Output	Flip-Flop Inputs	
Q	x	y	Q	S	JQ	KQ
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X

An Application – Serial Addition

- *A* = 0100; *B* = 0111. *A* + *B* = 1011 is stored in *A* after 4 clock pulses.

Initial:		*A*: 0 1 0 0 *B*: 0 1 1 1	Q: 0
Step 1:	**0 + 1 + 0** **S = 1, C = 0**	*A*: 1 0 1 0 *B*: x 0 1 1	Q: 0
Step 2:	**0 + 1 + 0** **S = 1, C = 0**	*A*: 1 1 0 1 *B*: x x 0 1	Q: 0
Step 3:	**1 + 1 + 0** **S = 0, C = 1**	*A*: 0 1 1 0 *B*: x x x 0	Q: 1
Step 4:	**0 + 0 + 1** **S = 1, C = 0**	*A*: 1 0 1 1 *B*: x x x x	Q: 0

Introduction: Counters

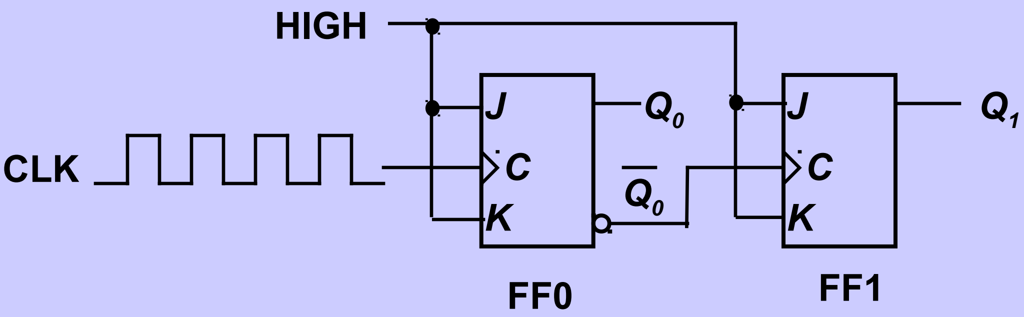
- Counters are circuits that cycle through a specified number of states.
- Two types of counters:
  - synchronous (parallel) counters
  - asynchronous (ripple) counters
- Ripple counters allow some flip-flop outputs to be used as a source of clock for other flip-flops.
- Synchronous counters apply the same clock to all flip-flops.

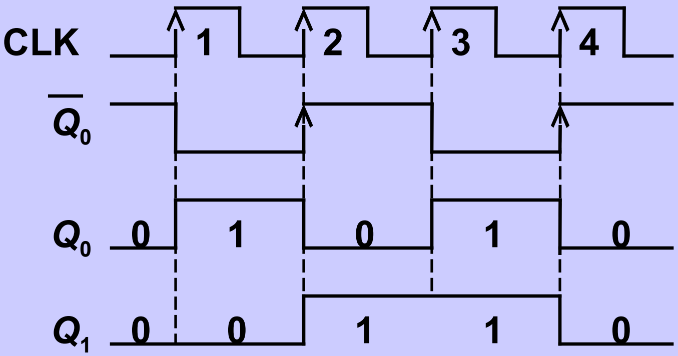
Asynchronous (Ripple) Counters

- Asynchronous counters: the flip-flops do not change states at exactly the same time as they do not have a common clock pulse.
- Also known as ripple counters, as the input clock pulse "ripples" through the counter – cumulative delay is a drawback.
- *n* flip-flops → a MOD (modulus) 2^n counter. (Note: A MOD-*x* counter cycles through *x* states.)
- Output of the last flip-flop (MSB) divides the input clock frequency by the MOD number of the counter, hence a counter is also a *frequency divider*.

Asynchronous (Ripple) Counters

- Example: 2-bit ripple binary counter.
- Output of one flip-flop is connected to the clock input of the next more-significant flip-flop.



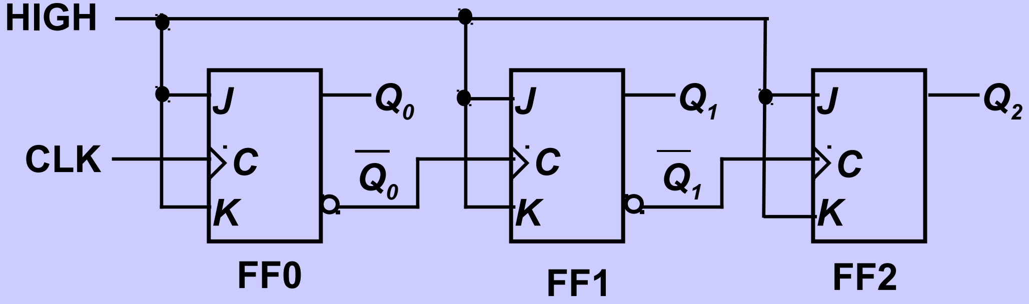


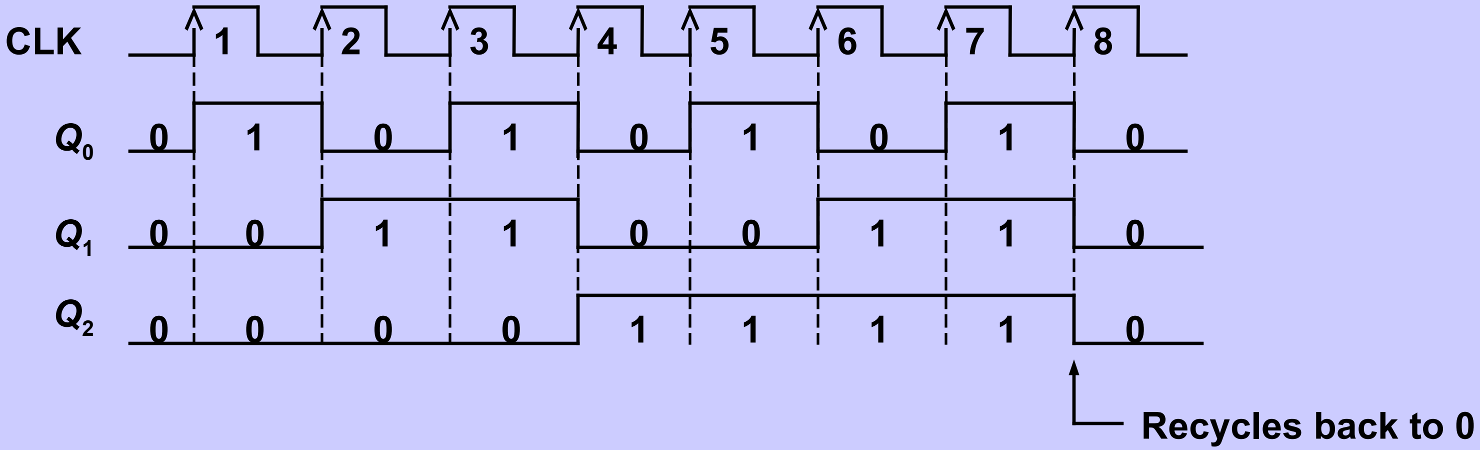
Timing diagram

00 → 01 → 10 → 11 → 00 ...

Asynchronous (Ripple) Counters

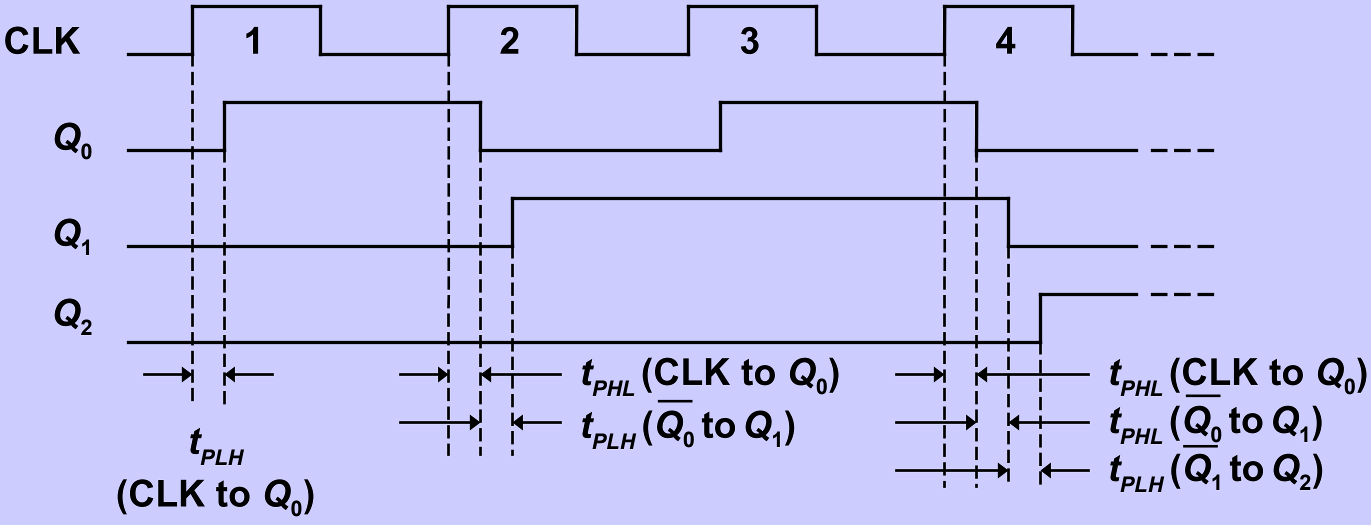
- Example: 3-bit ripple binary counter.





Asynchronous (Ripple) Counters

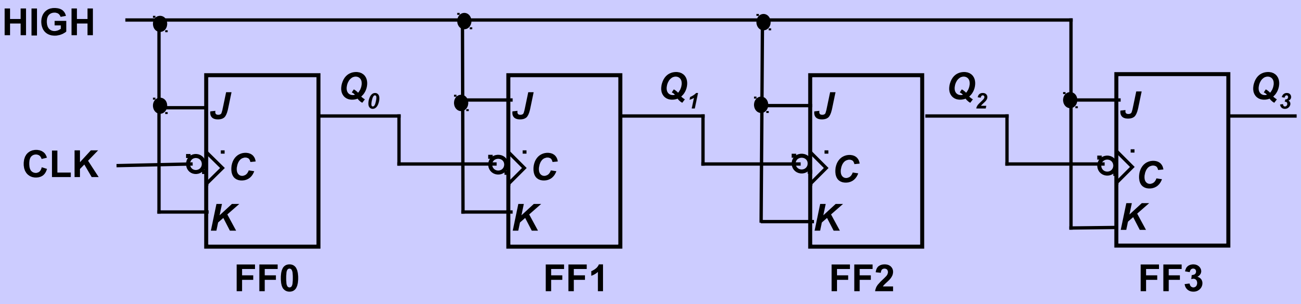
- Propagation delays in an asynchronous (ripple-clocked) binary counter.
- If the accumulated delay is greater than the clock pulse, some counter states may be misrepresented!

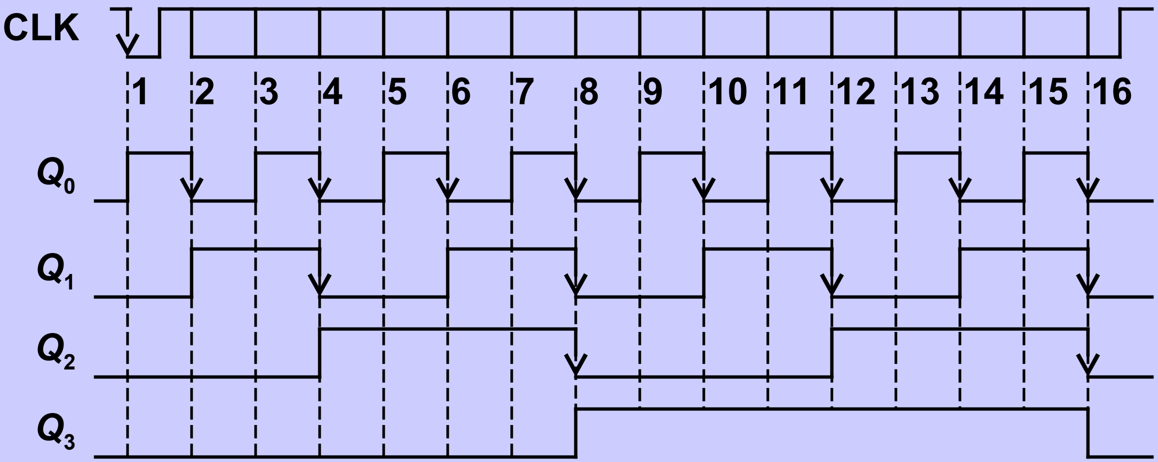


CLK
1
2
3
4
Q_0
Q_1
Q_2
t_{PLH} (CLK to Q_0)
t_{PHL} (CLK to Q_0)
t_{PLH} ($\overline{Q}_0$ to Q_1)
t_{PHL} (CLK to Q_0)
t_{PHL} ($\overline{Q}_0$ to Q_1)
t_{PLH} ($\overline{Q}_1$ to Q_2)

Asynchronous (Ripple) Counters

- Example: 4-bit ripple binary counter (negative-edge triggered).

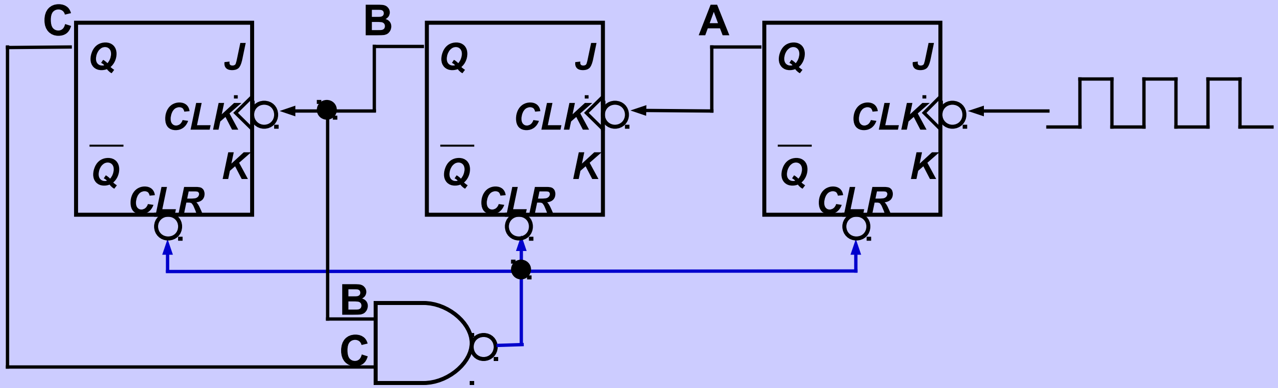




Asyn. Counters with MOD no. < 2^n

- States may be skipped resulting in a truncated sequence.
- Technique: force counter to *recycle before going through all of the states* in the binary sequence.
- Example: Given the following circuit, determine the counting sequence (and hence the modulus no.)

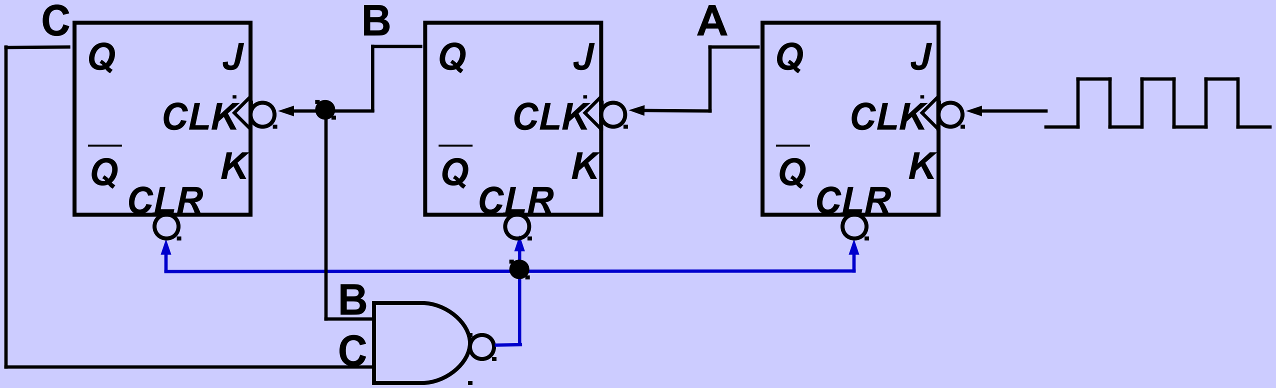
All *J*, *K* inputs are 1 (HIGH).

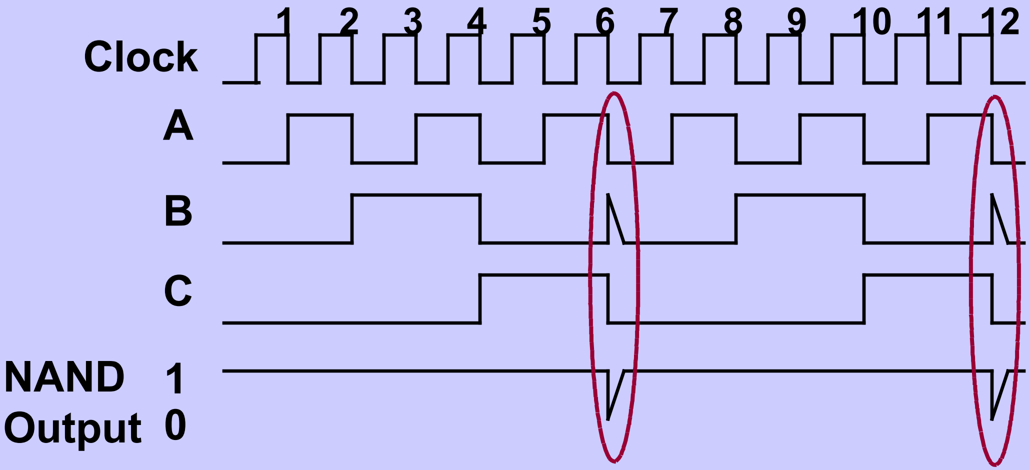


Asyn. Counters with MOD no. < 2^n

- Example (cont'd):

All *J*, *K* inputs are 1 (HIGH).

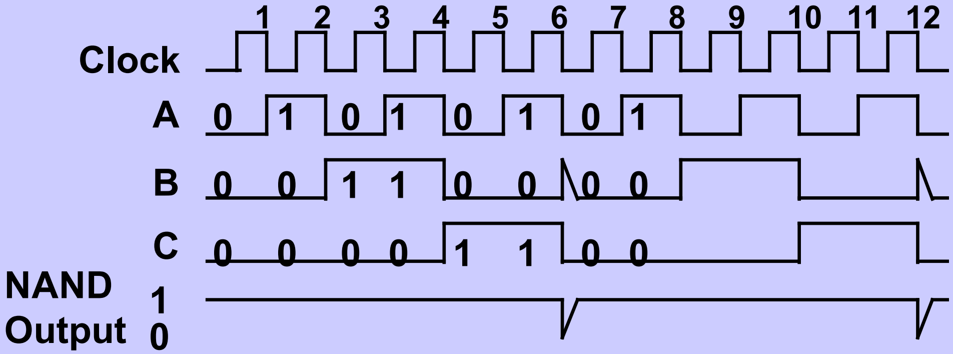


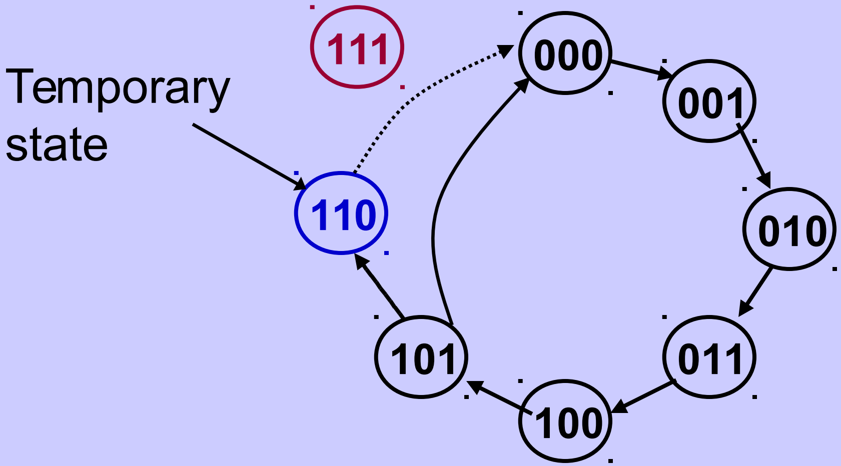


MOD-6 counter produced by clearing (a MOD-8 binary counter) when count of six (110) occurs.

Asyn. Counters with MOD no. < 2^n

- Example (cont'd): Counting sequence of circuit (in CBA order).

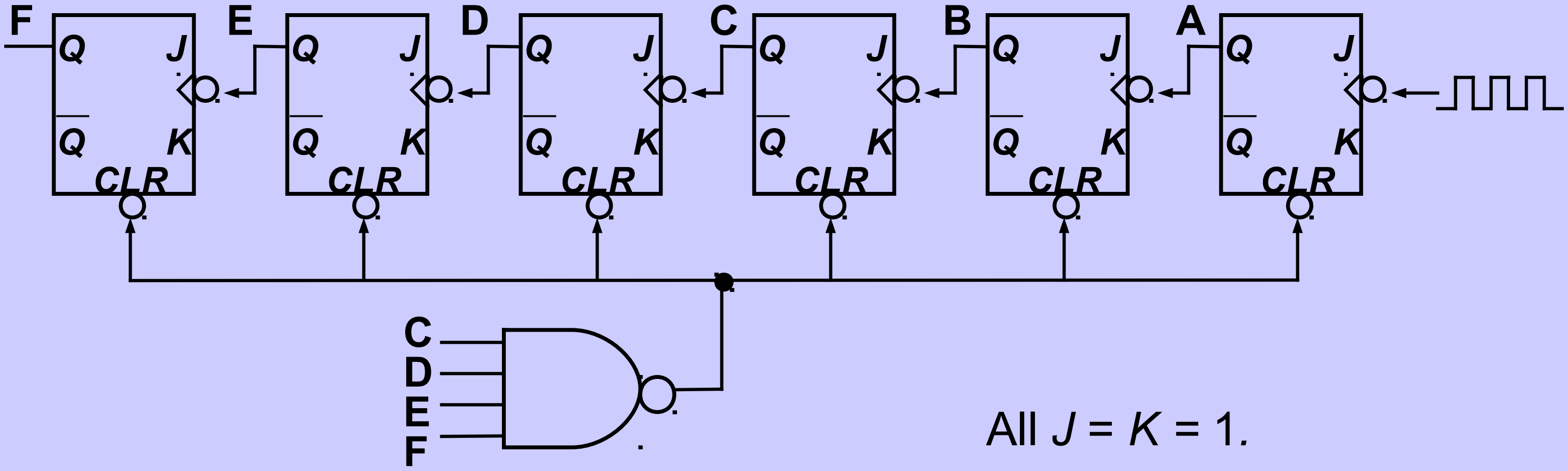




Counter is a MOD-6 counter.

Asyn. Counters with MOD no. < 2^n

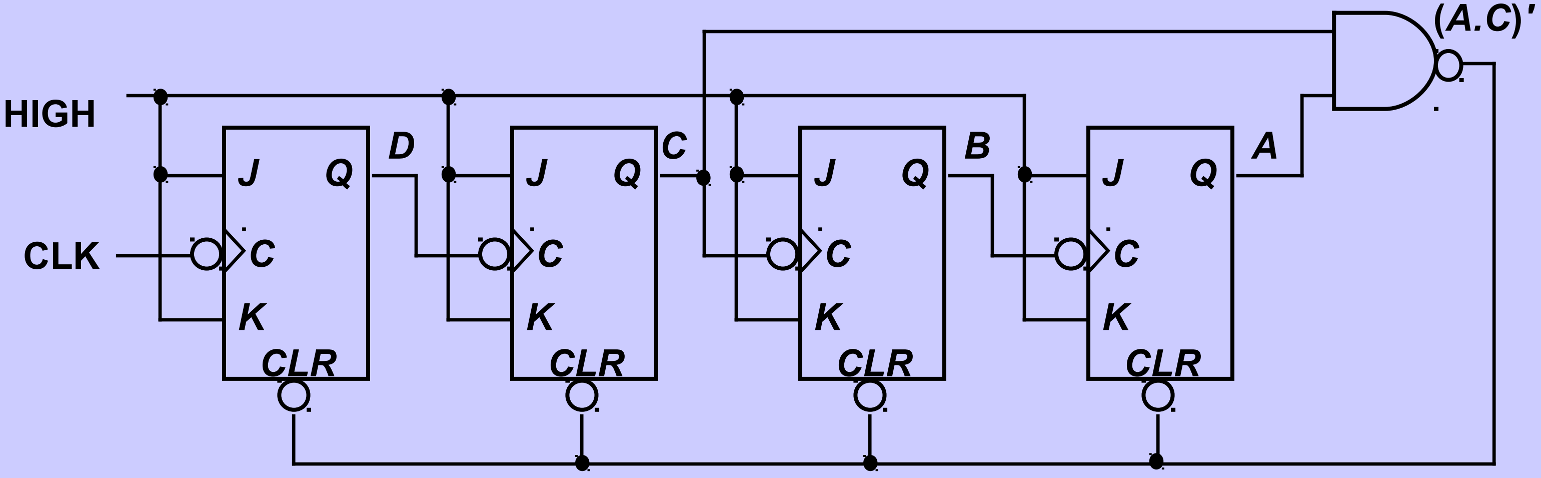
- *Exercise:* How to construct an asynchronous MOD-5 counter? MOD-7 counter? MOD-12 counter?
- *Question:* The following is a MOD-? counter?



All $J = K = 1$.

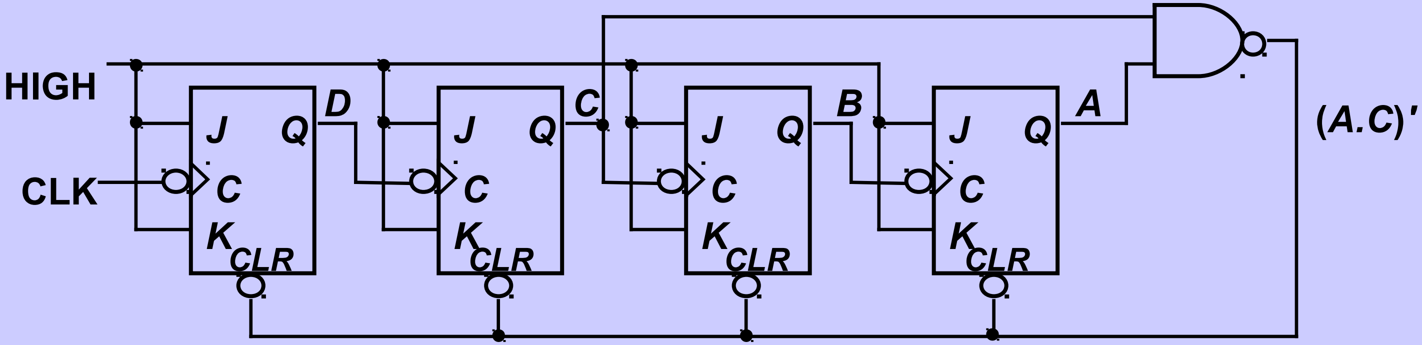
Asyn. Counters with MOD no. < 2^n

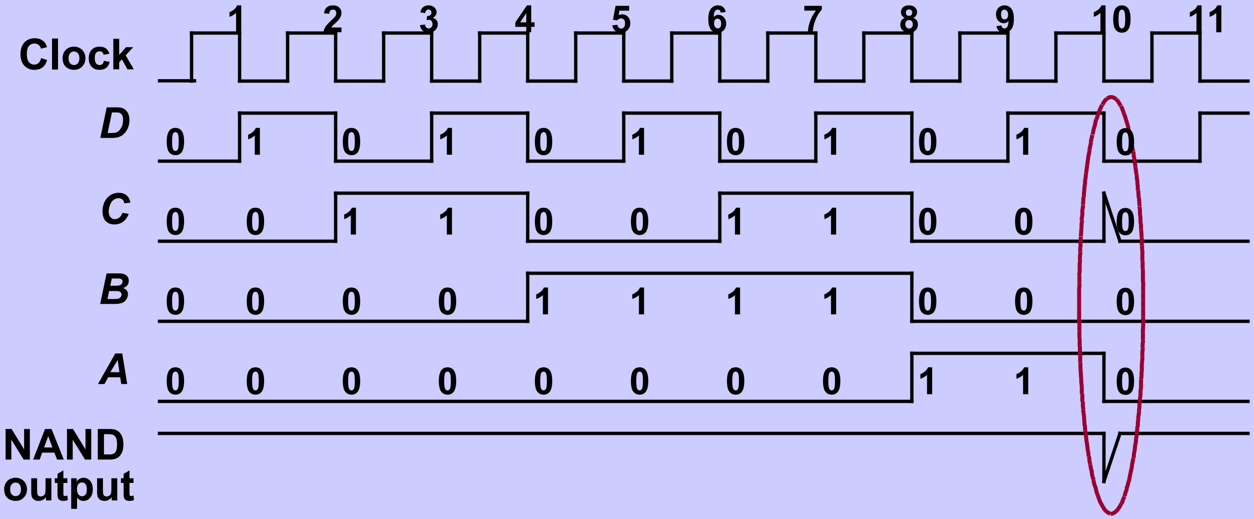
- Decade counters (or BCD counters) are counters with 10 states (modulus-10) in their sequence. They are commonly used in daily life (e.g.: utility meters, odometers, etc.).
- Design an asynchronous decade counter.



Asyn. Counters with MOD no. < 2^n

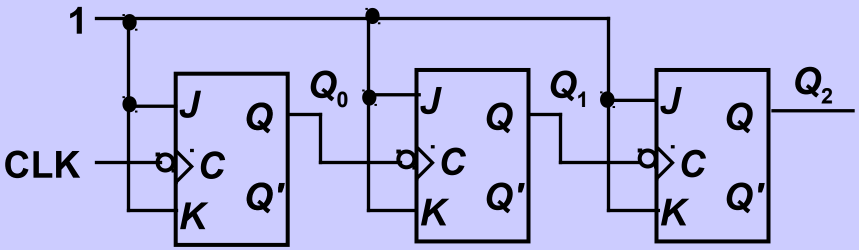
- Asynchronous decade/BCD counter (cont'd).



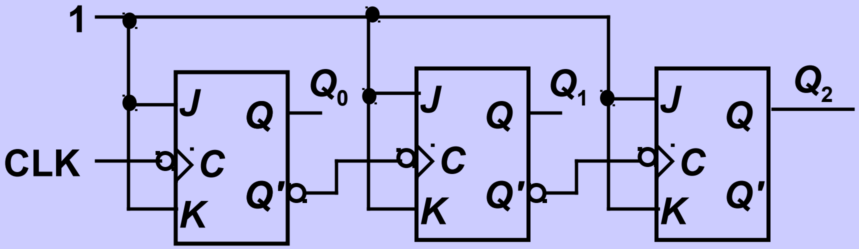


Asynchronous Down Counters

- So far we are dealing with *up counters*. *Down counters*, on the other hand, count downward from a maximum value to zero, and repeat.
- Example: A 3-bit binary (MOD-2^3) down counter.



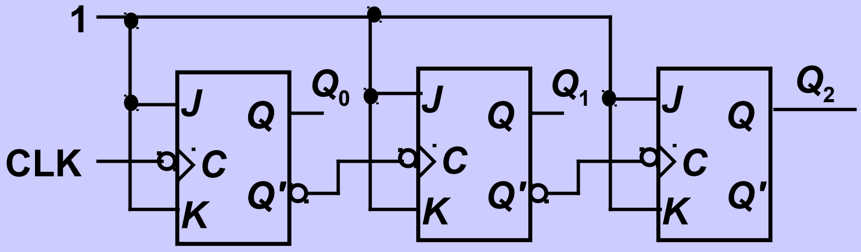
3-bit binary
up counter

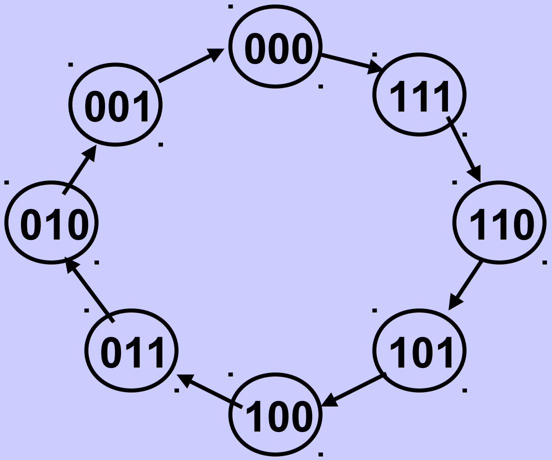


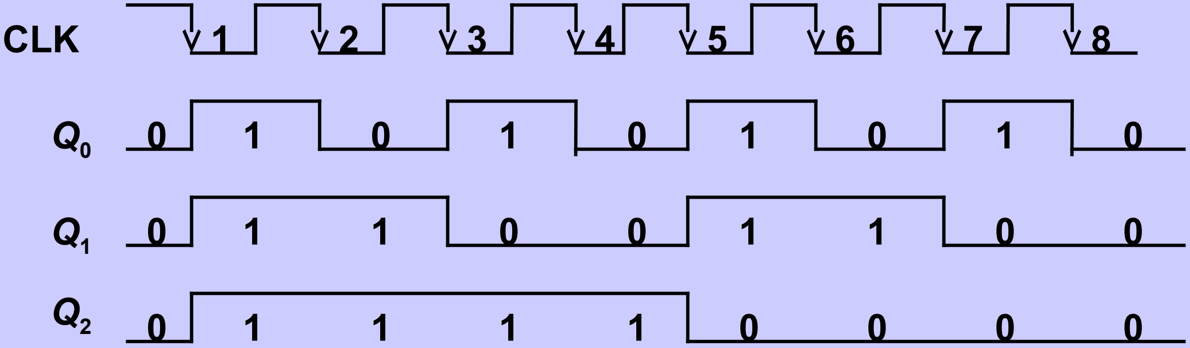
3-bit binary
down counter

Asynchronous Down Counters

- Example: A 3-bit binary (MOD-8) down counter.

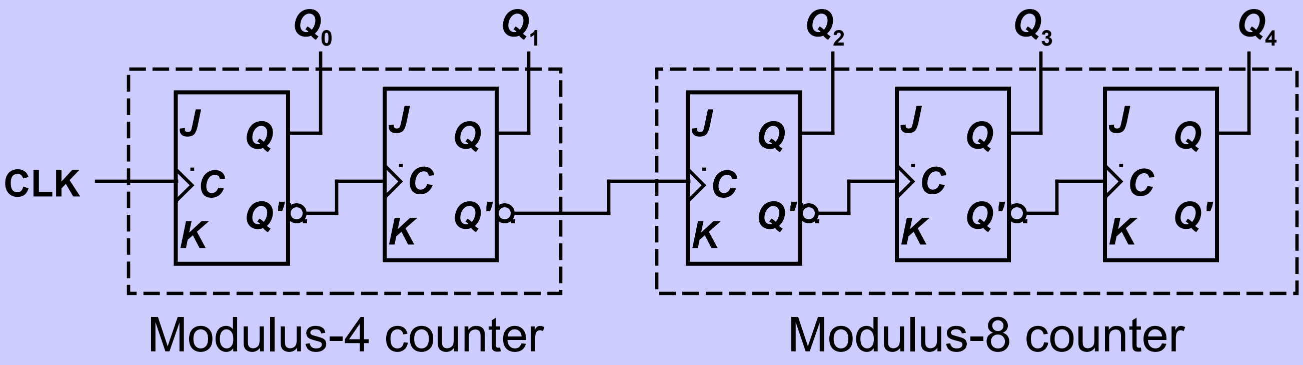






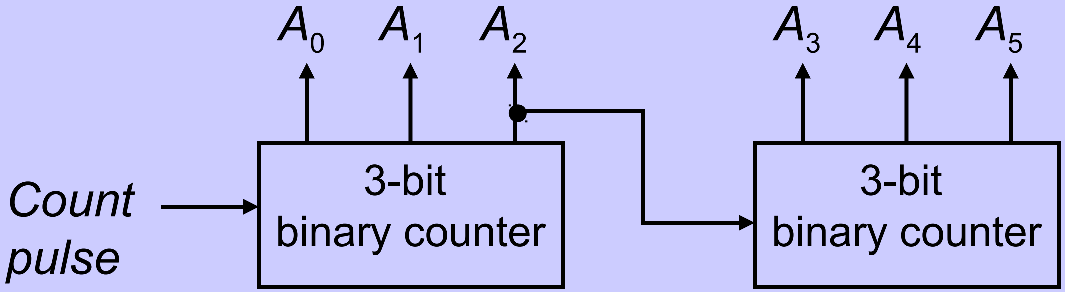
Cascading Asynchronous Counters

- Larger asynchronous (ripple) counter can be constructed by cascading smaller ripple counters.
- Connect last-stage output of one counter to the clock input of next counter so as to achieve higher-modulus operation.
- Example: A modulus-32 ripple counter constructed from a modulus-4 counter and a modulus-8 counter.



Cascading Asynchronous Counters

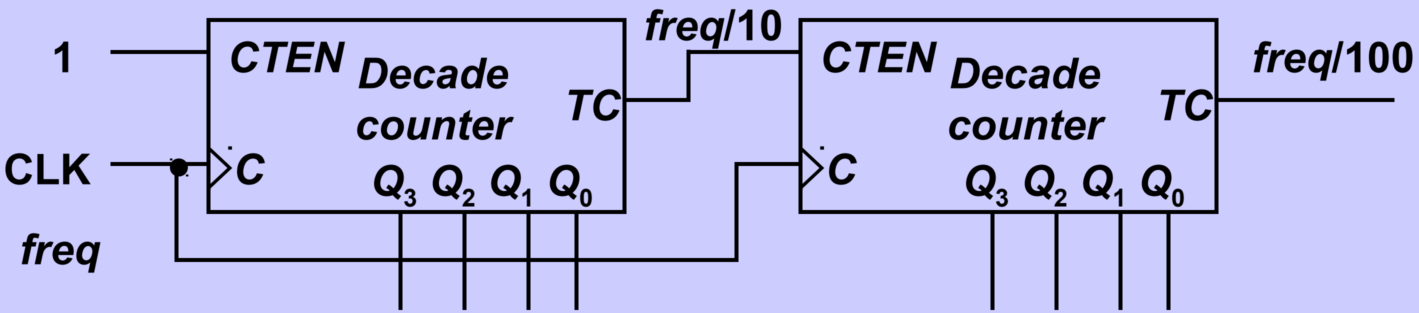
- Example: A 6-bit binary counter (counts from 0 to 63) constructed from two 3-bit counters.



A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	:	:	:
0	0	0	1	1	1
0	0	**1**	0	0	0
0	0	1	0	0	1
:	:	:	:	:	:

Cascading Asynchronous Counters

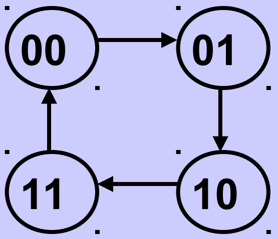
- If counter is a not a binary counter, requires additional output.
- Example: A modulus-100 counter using two decade counters.



TC = 1 when counter recycles to 0000

Synchronous (Parallel) Counters

- Synchronous (parallel) counters: the flip-flops are clocked at the same time by a common clock pulse.
- We can design these counters using the sequential logic design process (covered in Lecture #12).
- Example: 2-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J,K inputs).



Present state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

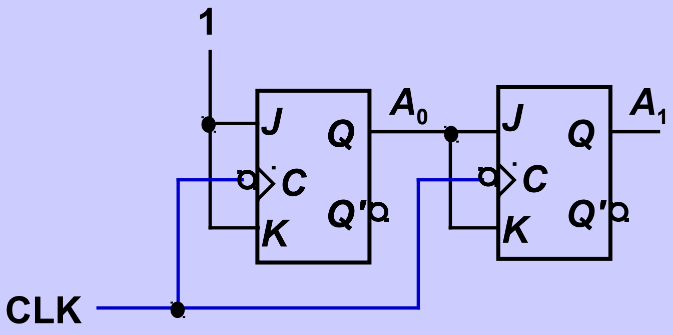
Synchronous (Parallel) Counters

- Example: 2-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J,K inputs).

Present state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

$TA_1 = A_0$

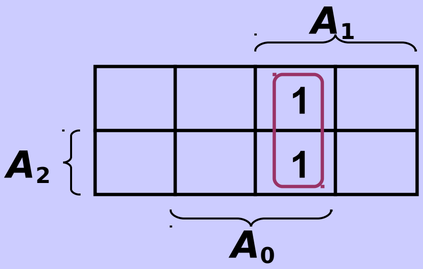
$TA_0 = 1$



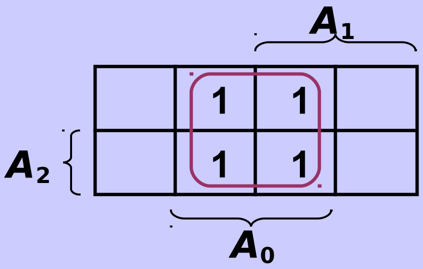
Synchronous (Parallel) Counters

- Example: 3-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J, K inputs).

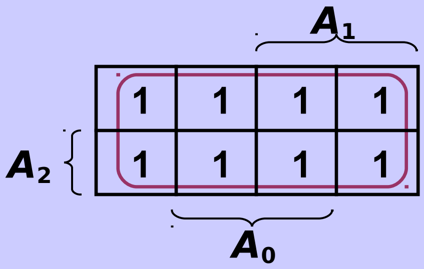
Present state			Next state			Flip-flop inputs		
A_2	A_1	A_0	A_2^+	A_1^+	A_0^+	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



$TA_2 = A_1.A_0$



$TA_1 = A_0$

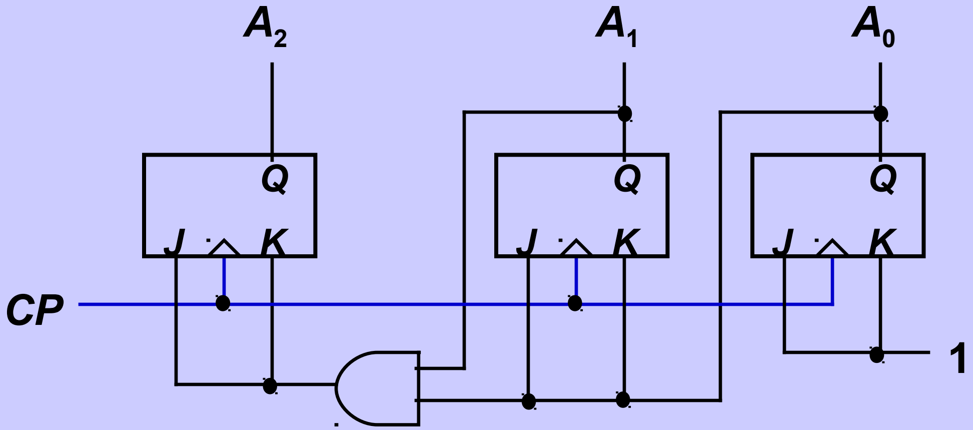


$TA_0 = 1$

Synchronous (Parallel) Counters

- Example: 3-bit synchronous binary counter (cont'd).

$TA_2 = A_1.A_0$ $TA_1 = A_0$ $TA_0 = 1$



Synchronous (Parallel) Counters

- Note that in a binary counter, the nth bit (shown underlined) is always complemented whenever

$$\underline{0}11\ldots11 \to \underline{1}00\ldots00$$

or $$\underline{1}11\ldots11 \to \underline{0}00\ldots00$$

- Hence, X_n is complemented whenever

  $X_{n-1}X_{n-2} \ldots X_1X_0 = 11\ldots11.$

- As a result, if T flip-flops are used, then

  $TX_n = X_{n-1} \,.\, X_{n-2} \,.\, \ldots \,.\, X_1 \,.\, X_0$

Synchronous (Parallel) Counters

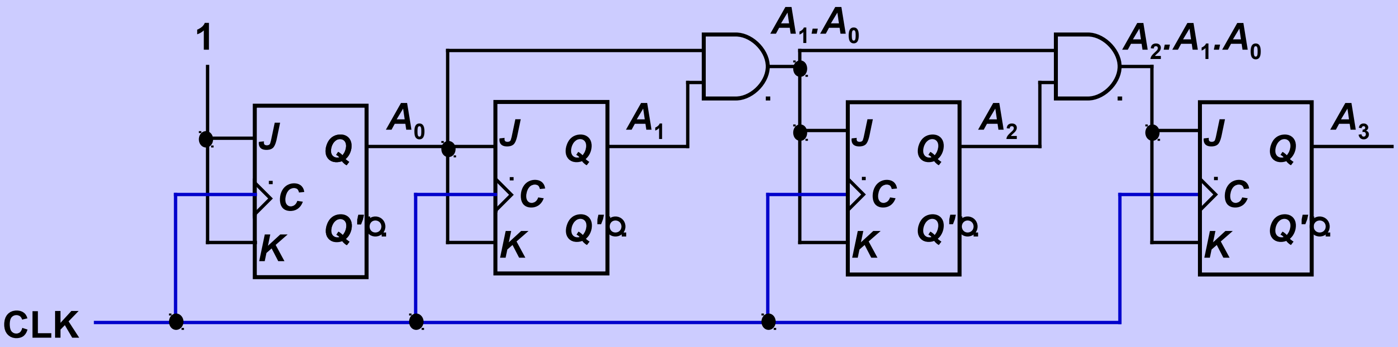
- Example: 4-bit synchronous binary counter.

$TA_3 = A_2 \,.\, A_1 \,.\, A_0$

$TA_2 = A_1 \,.\, A_0$

$TA_1 = A_0$

$TA_0 = 1$



Synchronous (Parallel) Counters

- Example: Synchronous decade/BCD counter.

Clock pulse	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (recycle)	0	0	0	0

$T_0 = 1$

$T_1 = Q_3'.Q_0$

$T_2 = Q_1.Q_0$

$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$

Synchronous (Parallel) Counters

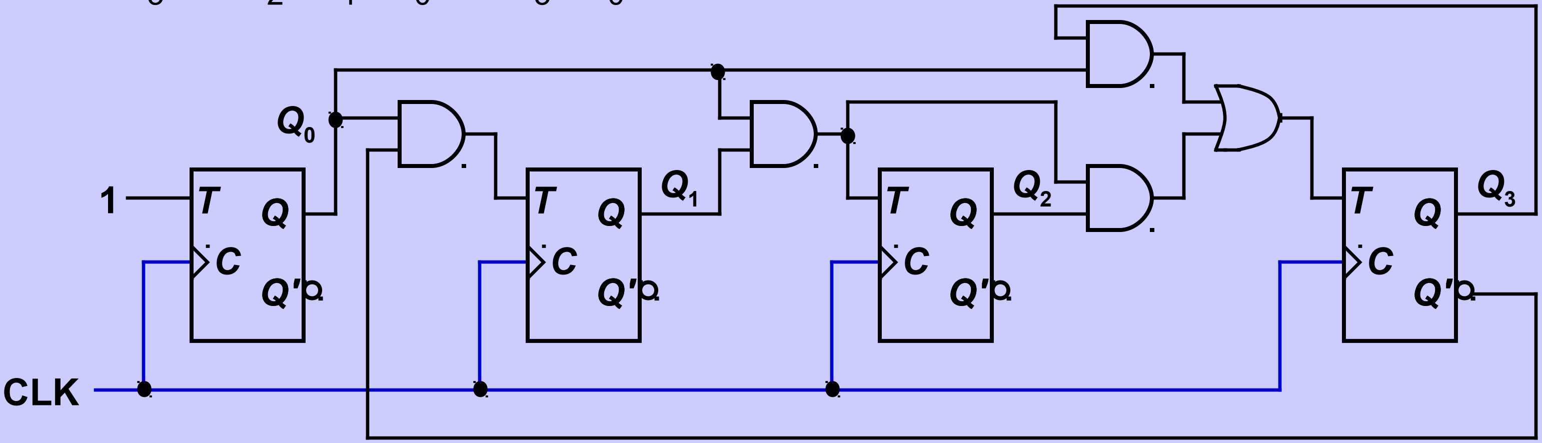
- Example: Synchronous decade/BCD counter (cont'd).

$T_0 = 1$

$T_1 = Q_3'.Q_0$

$T_2 = Q_1.Q_0$

$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$



Up/Down Synchronous Counters

- Up/down synchronous counter: a *bidirectional* counter that is capable of counting either up or down.
- An input (control) line $\mathit{Up}/\overline{\mathit{Down}}$ (or simply *Up*) specifies the direction of counting.
  - ❖ $\mathit{Up}/\overline{\mathit{Down}}$ = 1 → Count upward
  - ❖ $\mathit{Up}/\overline{\mathit{Down}}$ = 0 → Count downward

Up/Down Synchronous Counters

- Example: A 3-bit up/down synchronous binary counter.

Clock pulse	Up	Q_2	Q_1	Q_0	Down
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	

$TQ_0 = 1$

$TQ_1 = (Q_0.Up) + (Q_0'.Up')$

$TQ_2 = (\ Q_0.Q_1.Up\ ) + (Q_0'.\ Q_1'.\ Up'\ )$

Up counter	Down counter
$TQ_0 = 1$	$TQ_0 = 1$
$TQ_1 = Q_0$	$TQ_1 = Q_0'$
$TQ_2 = Q_0.Q_1$	$TQ_2 = Q_0'.Q_1'$

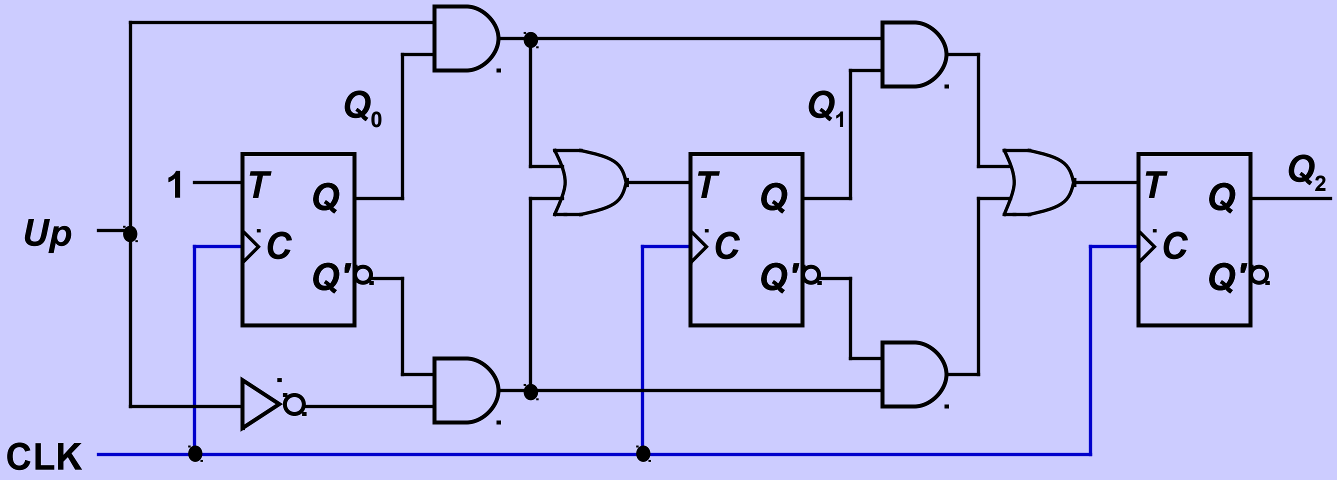
Up/Down Synchronous Counters

- Example: A 3-bit up/down synchronous binary counter (cont'd).

$TQ_0 = 1$

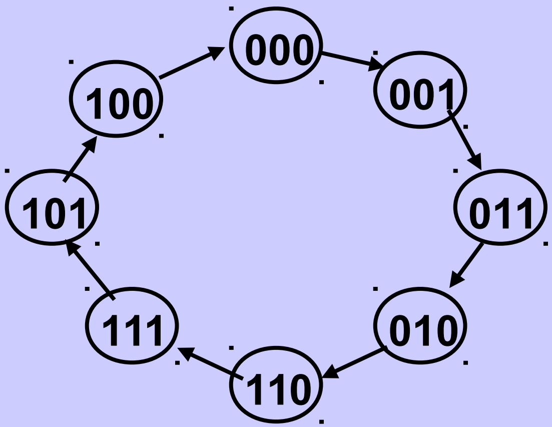
$TQ_1 = (Q_0.Up) + (Q_0'.Up')$

$TQ_2 = (\ Q_0.Q_1.Up\ ) + (Q_0'.\ Q_1'.\ Up')$



Designing Synchronous Counters

- Covered in Lecture #12.
- Example: A 3-bit Gray code counter (using JK flip-flops).



Present state			Next state			Flip-flop inputs					
Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	JQ_2	KQ_2	JQ_1	KQ_1	JQ_0	KQ_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	1	1	0	1	X	X	0	0	X
0	1	1	0	1	0	0	X	X	0	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	1	0	0	X	0	0	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	1	X	0	X	1	X	0

Designing Synchronous Counters

- 3-bit Gray code counter: flip-flop inputs.

Q_1Q_0

Q_2	00	01	11	10
0				1
1	X	X	X	X

$JQ_2 = Q_1.Q_0'$

Q_1Q_0

Q_2	00	01	11	10
0	X	X	X	X
1	1			

$KQ_2 = Q_1'.Q_0'$

Q_1Q_0

Q_2	00	01	11	10
0		1	X	X
1			X	X

$JQ_1 = Q_2'.Q_0$

Q_1Q_0

Q_2	00	01	11	10
0	X	X		
1	X	X	1	

$KQ_1 = Q_2.Q_0$

Q_1Q_0

Q_2	00	01	11	10
0	1	X	X	
1		X	X	1

$JQ_0 = Q_2.Q_1 + Q_2'.Q_1'$

$= (Q_2 \oplus Q_1)'$

Q_1Q_0

Q_2	00	01	11	10
0	X		1	X
1	X	1		X

$KQ_0 = Q_2.Q_1' + Q_2'.Q_1$

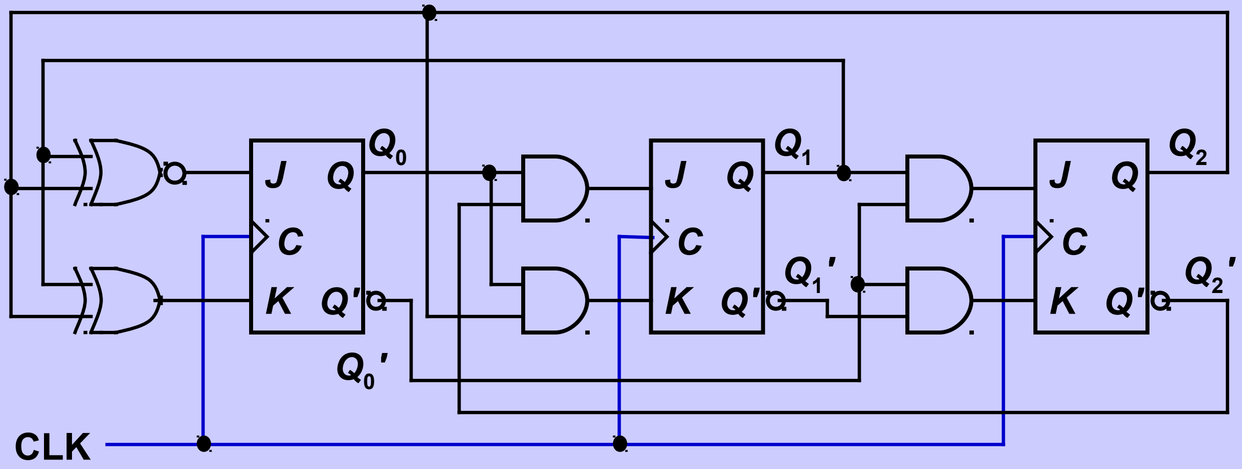
$= Q_2 \oplus Q_1$

Designing Synchronous Counters

- 3-bit Gray code counter: logic diagram.

$JQ_2 = Q_1.Q_0'$ $JQ_1 = Q_2'.Q_0$ $JQ_0 = (Q_2 \oplus Q_1)'$

$KQ_2 = Q_1'.Q_0'$ $KQ_1 = Q_2.Q_0$ $KQ_0 = Q_2 \oplus Q_1$

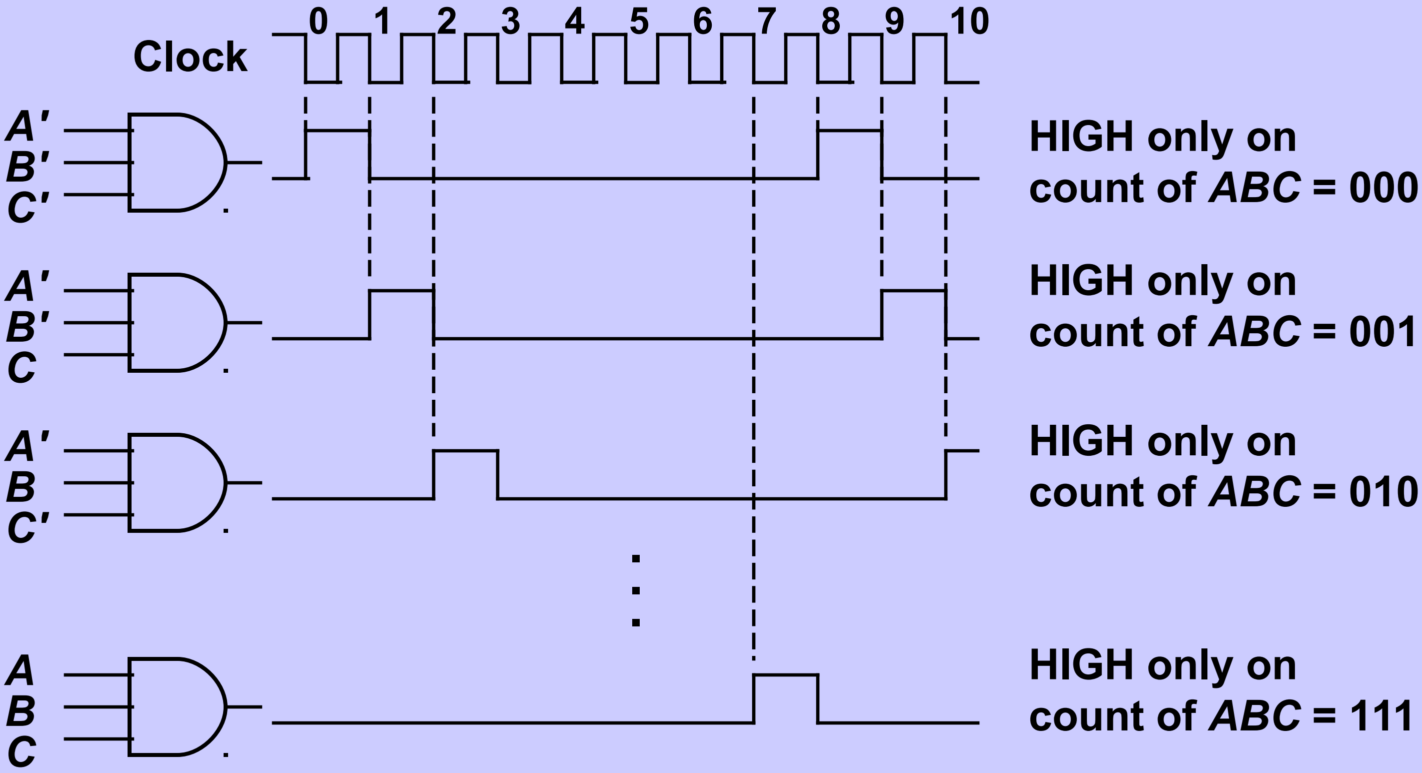


Decoding A Counter

- Decoding a counter involves determining which state in the sequence the counter is in.
- Differentiate between *active-HIGH* and *active-LOW* decoding.
- Active-HIGH decoding: output HIGH if the counter is in the state concerned.
- Active-LOW decoding: output LOW if the counter is in the state concerned.

Decoding A Counter

- Example: MOD-8 ripple counter (active-HIGH decoding).



HIGH only on count of *ABC* = 000

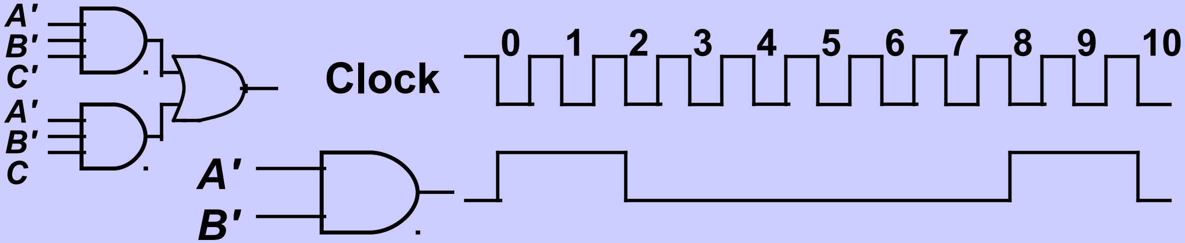
HIGH only on count of *ABC* = 001

HIGH only on count of *ABC* = 010

HIGH only on count of *ABC* = 111

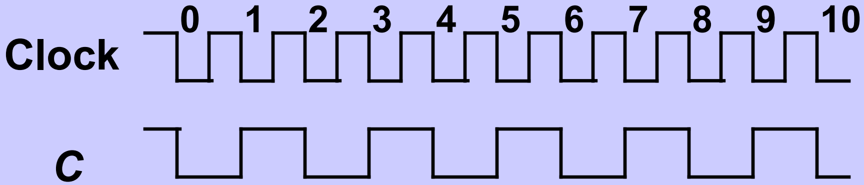
Decoding A Counter

- Example: To detect that a MOD-8 counter is in state 0 (000) or state 1 (001).



HIGH only on count of *ABC* = 000 or *ABC* = 001

- Example: To detect that a MOD-8 counter is in the odd states (states 1, 3, 5 or 7), simply use *C*.



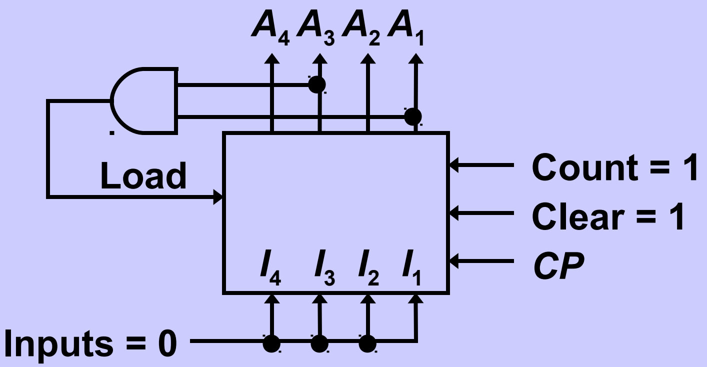
HIGH only on count of *odd states*

Counters with Parallel Load

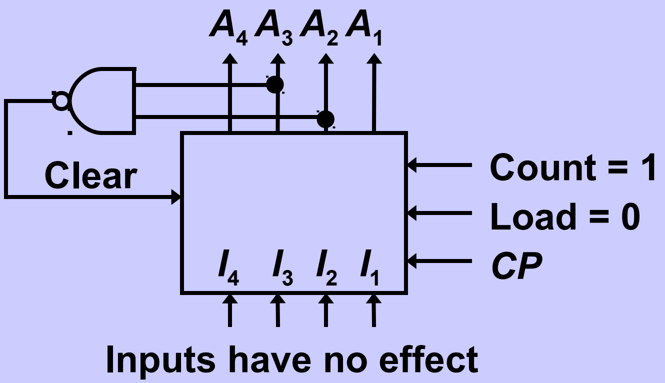
- Counters could be augmented with parallel load capability for the following purposes:
  - To start at a different state
  - To count a different sequence
  - As more sophisticated register with increment/decrement functionality.

Counters with Parallel Load

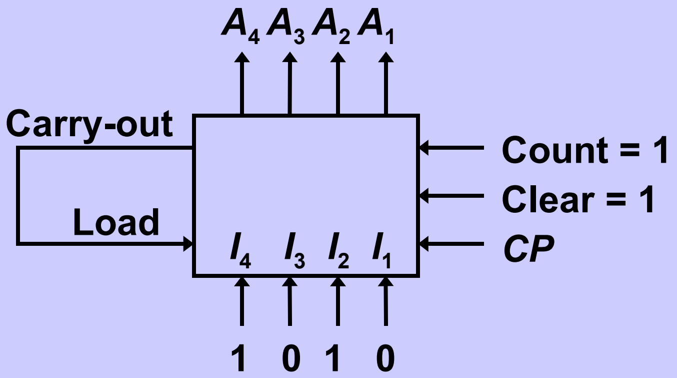
- Different ways of getting a MOD-6 counter:



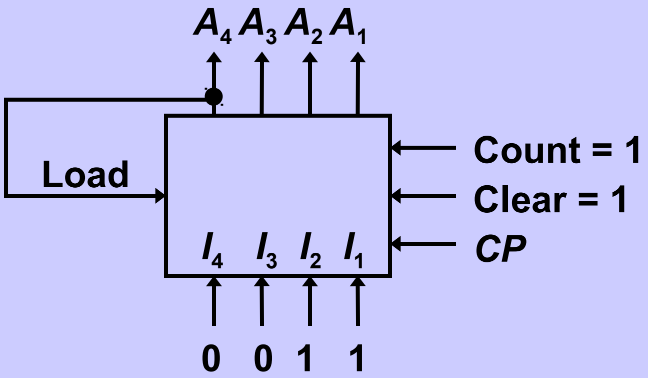
(a) Binary states 0,1,2,3,4,5.



(b) Binary states 0,1,2,3,4,5.



(c) Binary states 10,11,12,13,14,15.

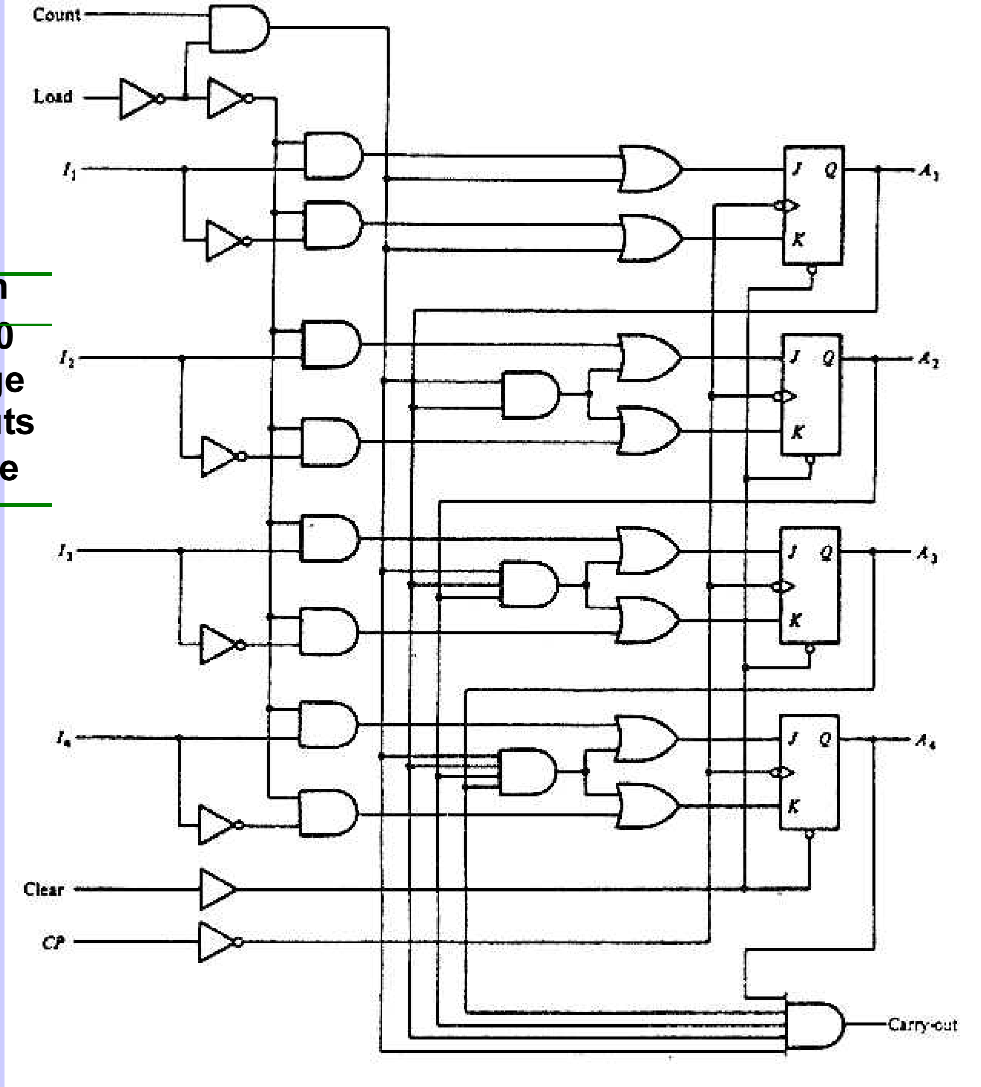


(d) Binary states 3,4,5,6,7,8.

Counters with Parallel Load

- 4-bit counter with parallel load.

Clear	CP	Load	Count	Function
0	X	X	X	Clear to 0
1	X	0	0	No change
1	↑	1	X	Load inputs
1	↑	0	1	Next state



Shift Register Counters

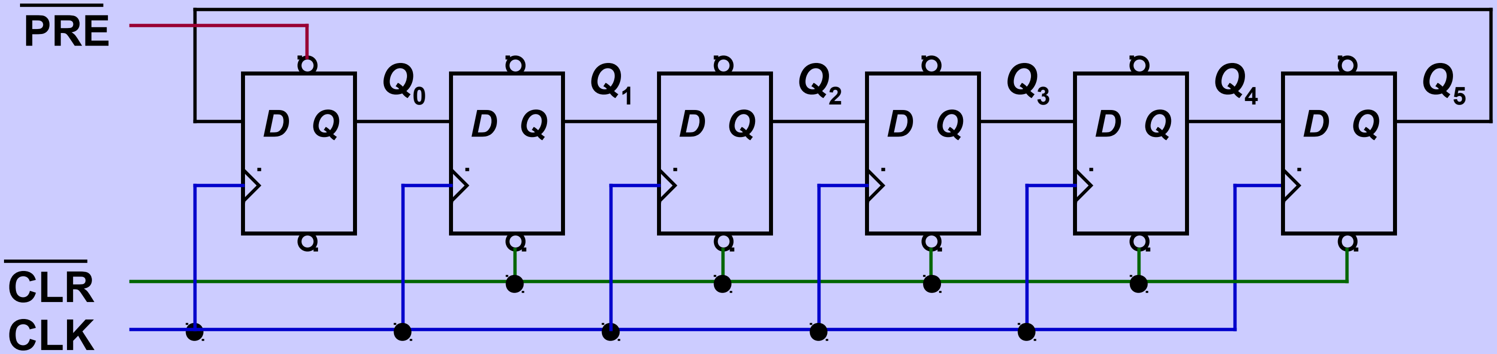
- Shift register counter: a shift register with the serial output connected back to the serial input.
- They are classified as counters because they give a specified sequence of states.
- Two common types: the *Johnson counter* and the *Ring counter*.

Ring Counters

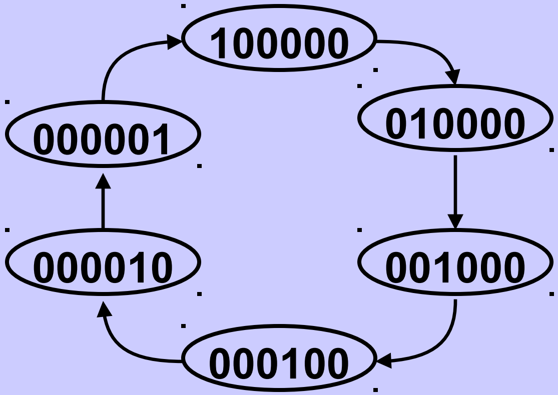
- One flip-flop (stage) for each state in the sequence.
- The output of the last stage is connected to the D input of the first stage.
- An *n*-bit ring counter cycles through *n* states.
- No decoding gates are required, as there is an output that corresponds to every state the counter is in.

Ring Counters

- Example: A 6-bit (MOD-6) ring counter.



Clock	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1

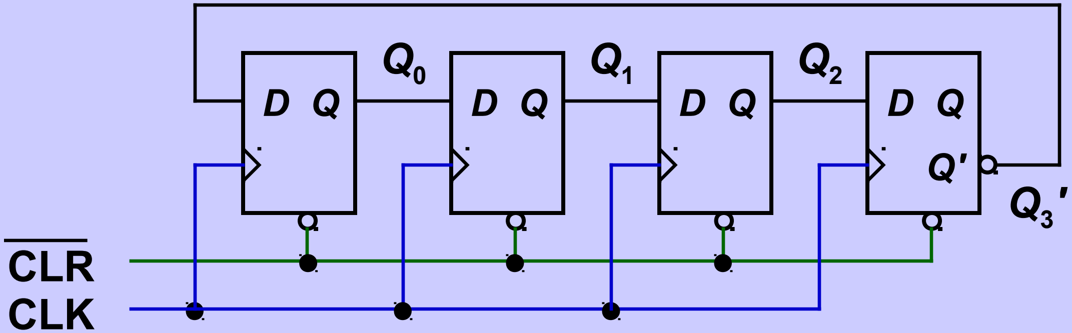


Johnson Counters

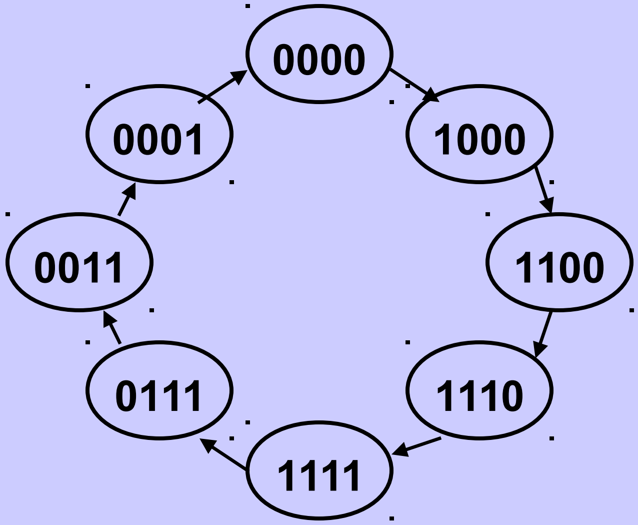
- The complement of the output of the last stage is connected back to the D input of the first stage.
- Also called the *twisted-ring counter*.
- Require fewer flip-flops than ring counters but more flip-flops than binary counters.
- An *n*-bit Johnson counter cycles through 2*n* states.
- Require more decoding circuitry than ring counter but less than binary counters.

Johnson Counters

- Example: A 4-bit (MOD-8) Johnson counter.



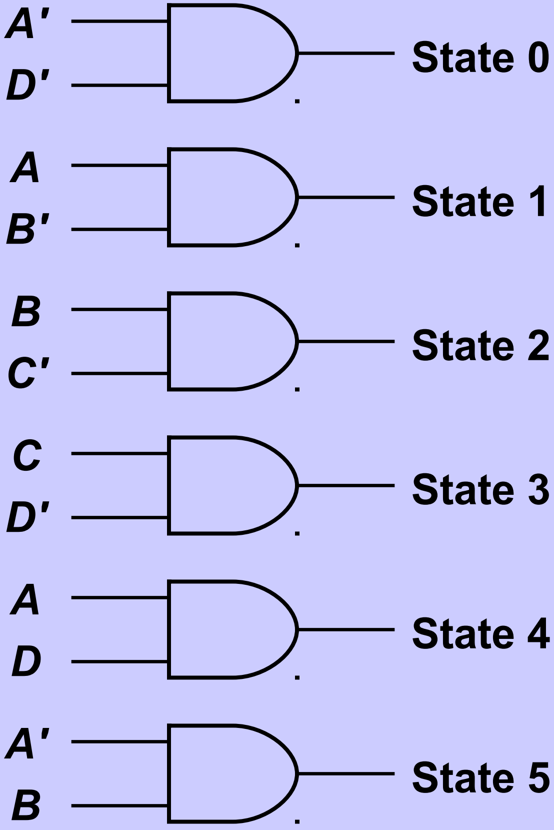
Clock	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

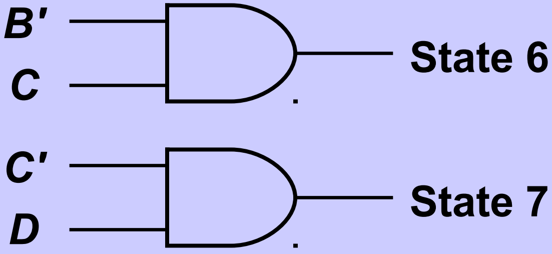


Johnson Counters

- Decoding logic for a 4-bit Johnson counter.

Clock	*A*	*B*	*C*	*D*	Decoding
→0	0	0	0	0	*A'.D'*
1	1	0	0	0	*A.B'*
2	1	1	0	0	*B.C'*
3	1	1	1	0	*C.D'*
4	1	1	1	1	*A.D*
5	0	1	1	1	*A'.B*
6	0	0	1	1	*B'.C*
7	0	0	0	1	*C'.D*



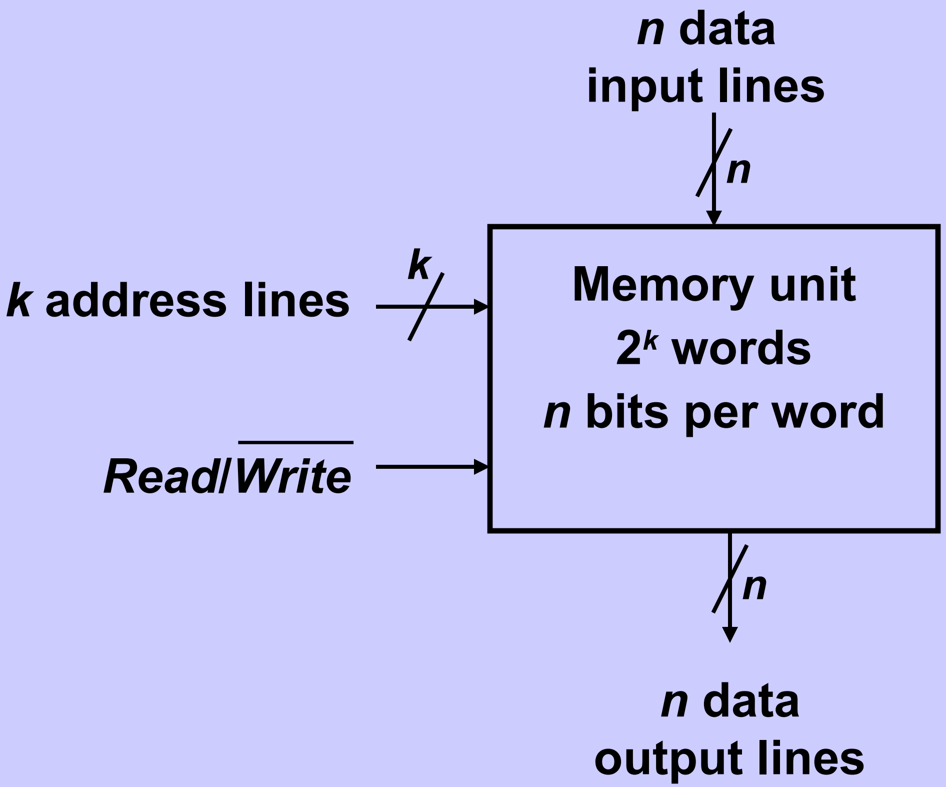


Random Access Memory (RAM)

- A memory unit stores binary information in groups of bits called *words*.
- The data consists of *n* lines (for *n*-bit words). Data input lines provide the information to be stored (*written*) into the memory, while data output lines carry the information out (*read*) from the memory.
- The address consists of *k* lines which specify which word (among the 2^k words available) to be selected for reading or writing.
- The control lines *Read* and *Write* (usually combined into a single control line *Read/Write*) specifies the direction of transfer of the data.

Random Access Memory (RAM)

- Block diagram of a memory unit:



Random Access Memory (RAM)

- Content of a 1024 x 16-bit memory:

Memory address		Memory content
binary	decimal	
0000000000	0	1011010111011101
0000000001	1	1010000110000110
0000000010	2	0010011101110001
:	:	:
:	:	:
1111111101	1021	1110010101010010
1111111110	1022	0011111010101110
1111111111	1023	1011000110010101

Random Access Memory (RAM)

- The Write operation:
  - Transfers the address of the desired word to the address lines
  - Transfers the data bits (the word) to be stored in memory to the data input lines
  - Activates the *Write* control line (set $\mathit{Read}/\overline{\mathit{Write}}$ to 0)
- The Read operation:
  - Transfers the address of the desired word to the address lines
  - Activates the *Read* control line (set $\mathit{Read}/\overline{\mathit{Write}}$ to 1)

Random Access Memory (RAM)

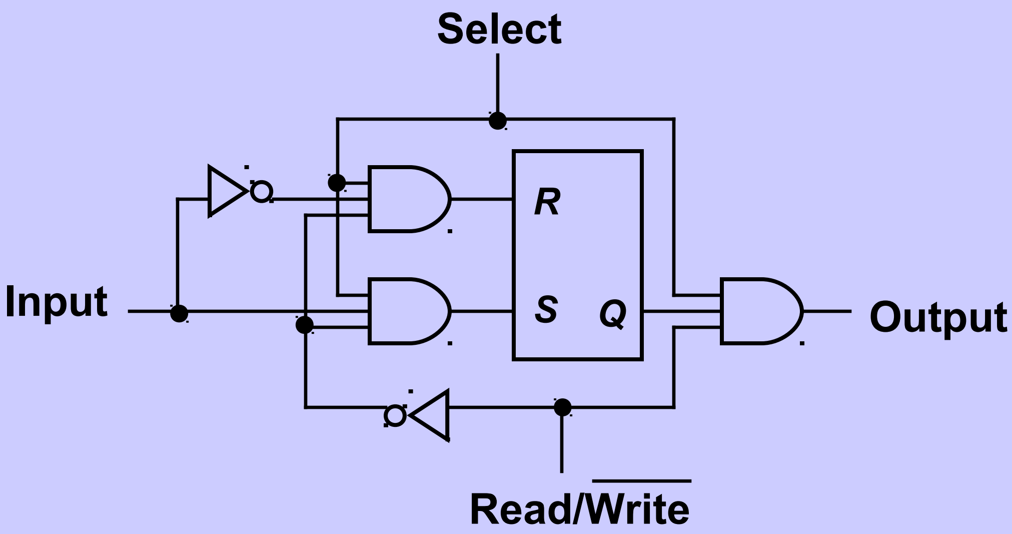
- The Read/Write operation:

Memory Enable	*Read/Write*	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

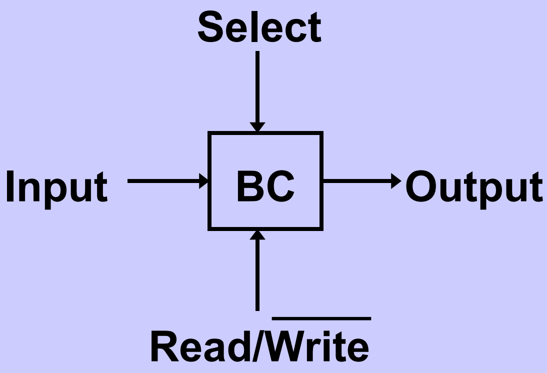
- Two types of RAM: Static and dynamic.
  - Static RAMs use flip-flops as the memory cells.
  - Dynamic RAMs use capacitor charges to represent data. Though simpler in circuitry, they have to be constantly refreshed.

Random Access Memory (RAM)

- A single memory cell of the static RAM has the following logic and block diagrams.



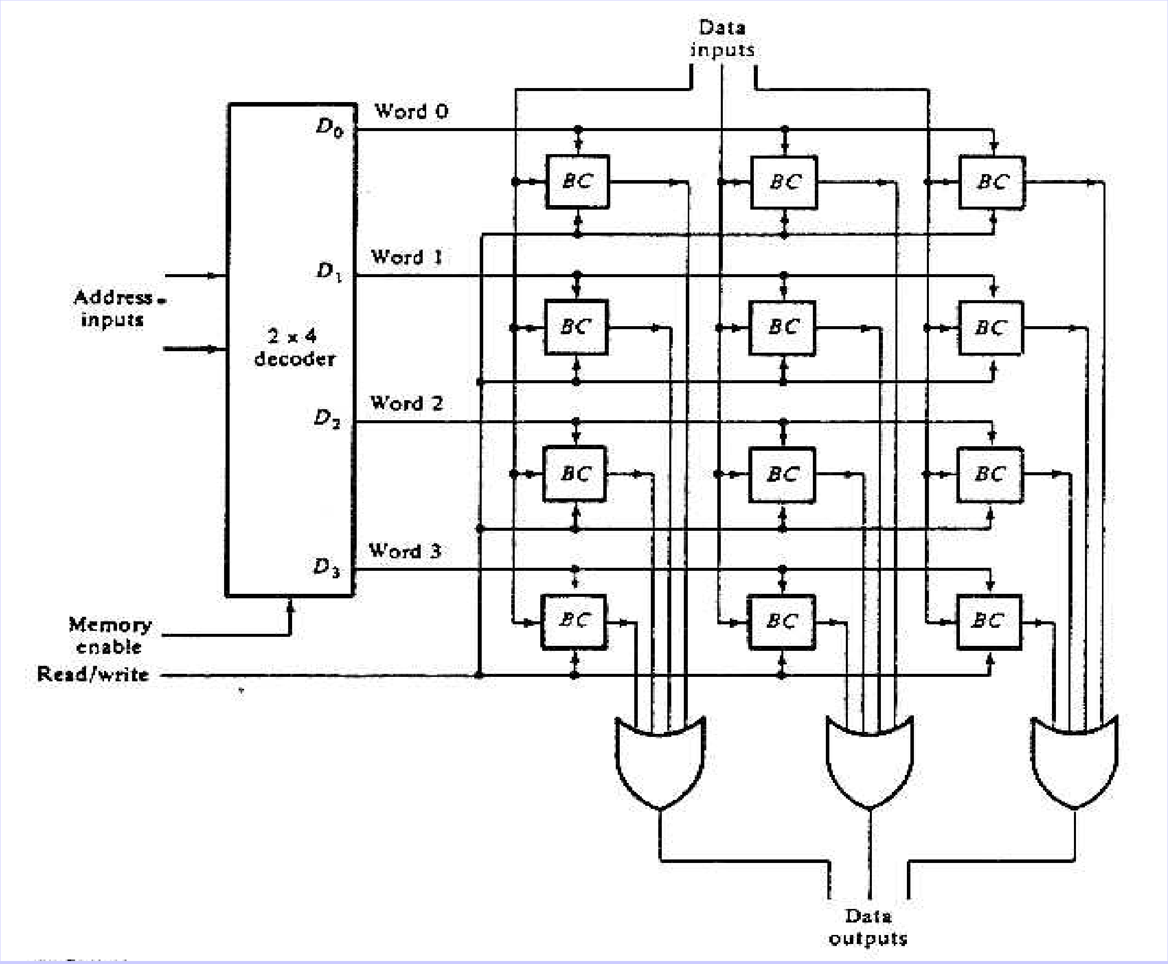
Logic diagram



Block diagram

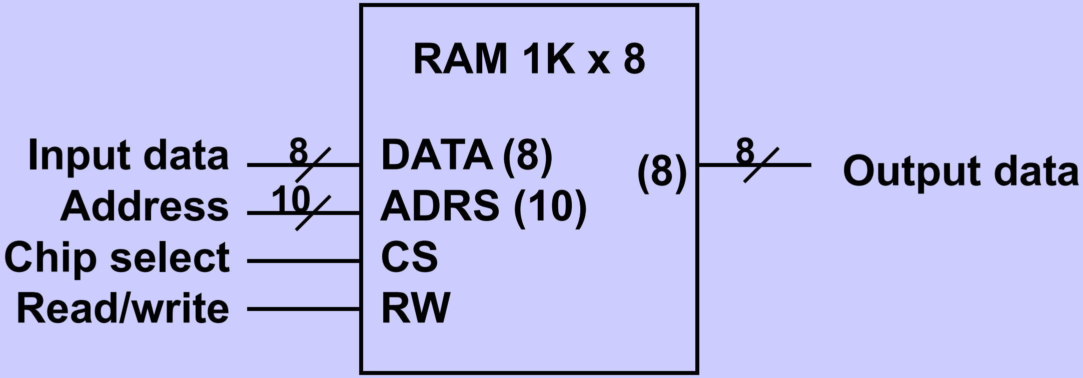
Random Access Memory (RAM)

- Logic construction of a 4 x 3 RAM (with decoder and OR gates):



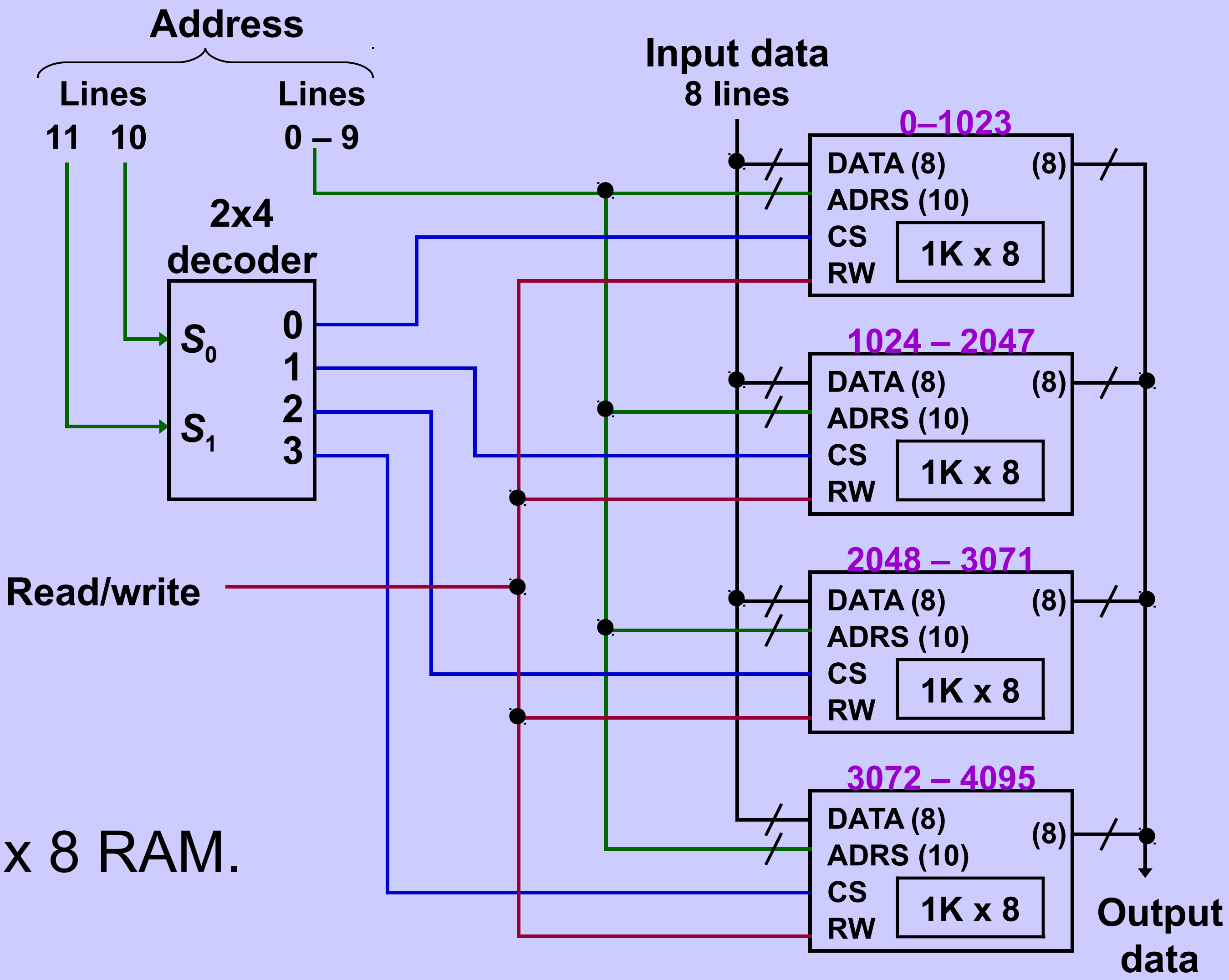
Random Access Memory (RAM)

- An array of RAM chips: memory chips are combined to form larger memory.
- A 1K x 8-bit RAM chip:



Block diagram of a 1K x 8 RAM chip

Random Access Memory (RAM)



- 4K x 8 RAM.

End of segment