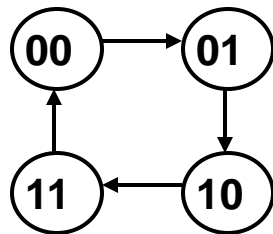


Synchronous Counter

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

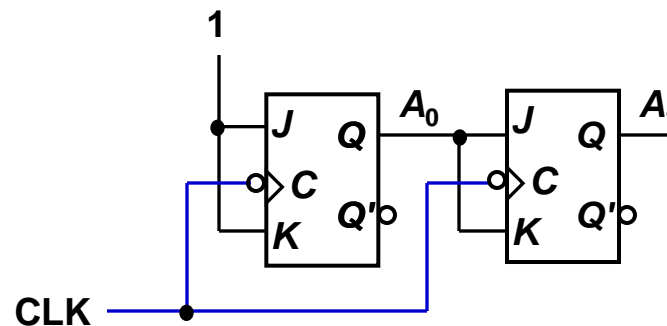
Synchronous (Parallel) Counters

- 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

$$TA_1 = A_0$$

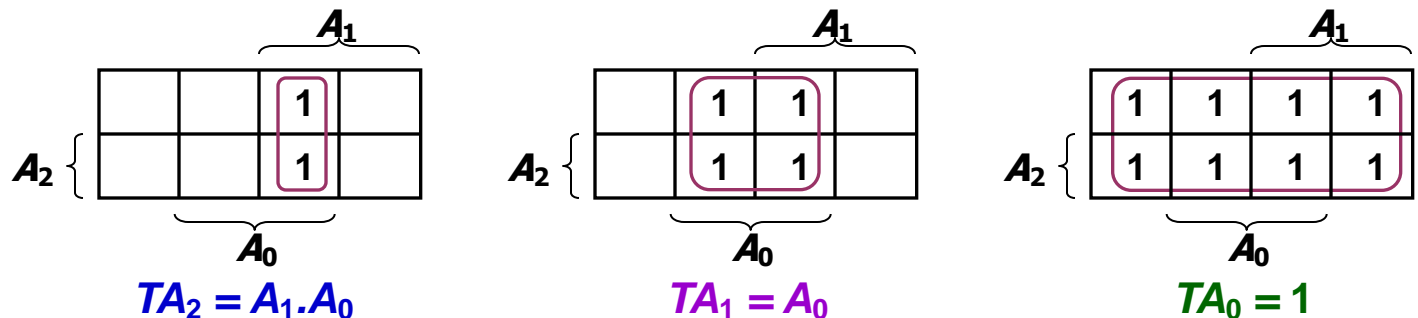
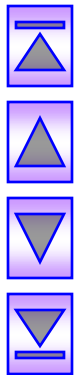
$$TA_0 = 1$$



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

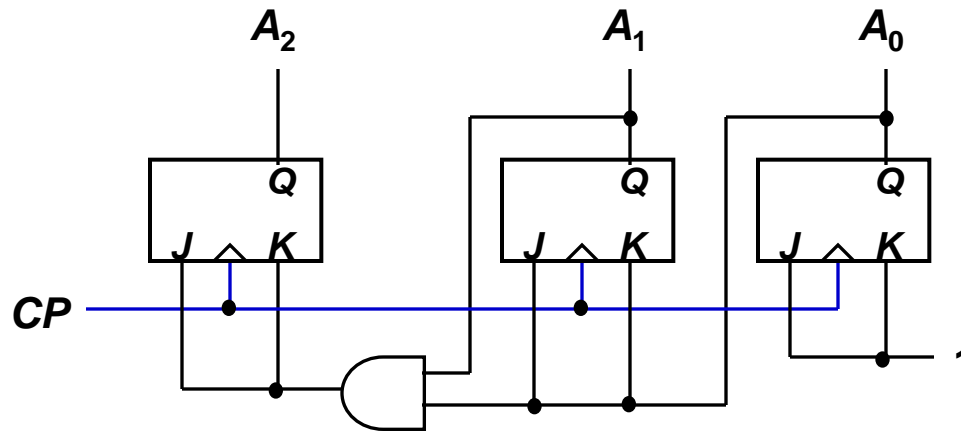


Synchronous (Parallel) Counters

- 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 n^{th} bit (shown underlined) is always complemented whenever

$$\underline{0}11\dots11 \rightarrow \underline{1}00\dots00$$

$$\text{or } \underline{1}11\dots11 \rightarrow \underline{0}00\dots00$$

- Hence, X_n is complemented whenever

$$X_{n-1}X_{n-2} \dots X_1X_0 = 11\dots11.$$

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

$$TX_n = X_{n-1} \cdot X_{n-2} \cdot \dots \cdot X_1 \cdot X_0$$

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

$$T_0 = 1$$

$$T_1 = Q_3' \cdot Q_0$$

$$T_2 = Q_1 \cdot Q_0$$

$$T_3 = Q_2 \cdot Q_1 \cdot Q_0 + Q_3 \cdot Q_0$$

Synchronous (Parallel) Counters

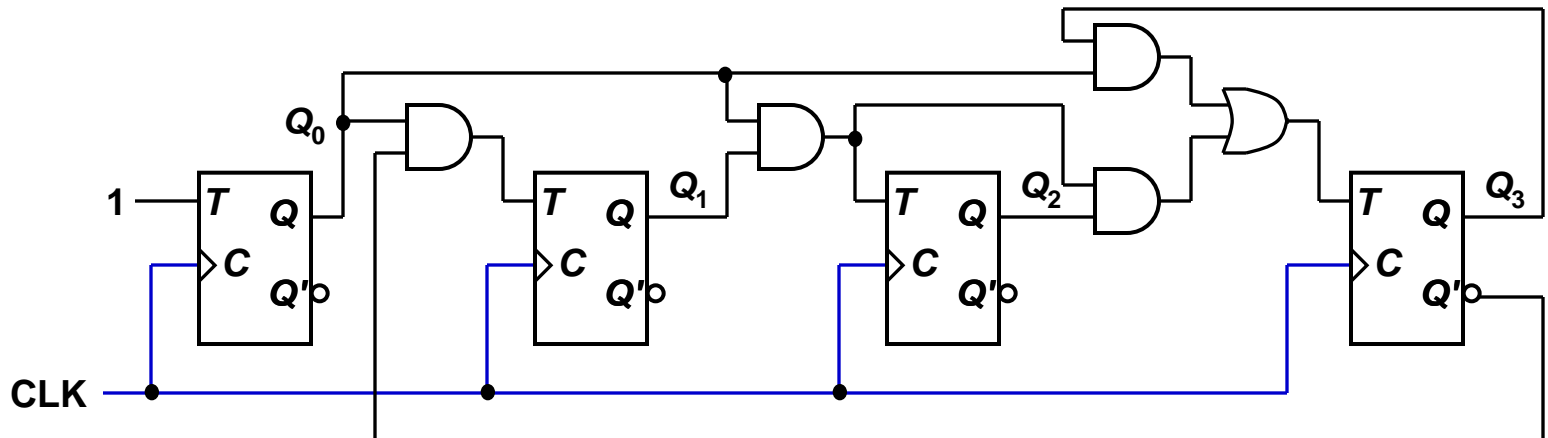
- Example: Synchronous decade/BCD counter (cont'd).

$$T_0 = 1$$

$$T_1 = Q_3' \cdot Q_0$$

$$T_2 = Q_1 \cdot Q_0$$

$$T_3 = Q_2 \cdot Q_1 \cdot Q_0 + Q_3 \cdot Q_0$$



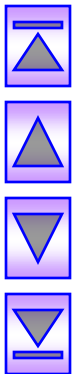
Up/Down Synchronous Counters

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

Up/Down Synchronous Counters

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

Clock pulse	<i>Up</i>	Q_2	Q_1	Q_0	<i>Down</i>
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	



$$TQ_0 = 1$$

$$TQ_1 = (Q_0 \cdot Up) + (Q_0' \cdot Up')$$

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

Up counter

$$TQ_0 = 1$$

$$TQ_1 = Q_0$$

$$TQ_2 = Q_0 \cdot Q_1$$

Down counter

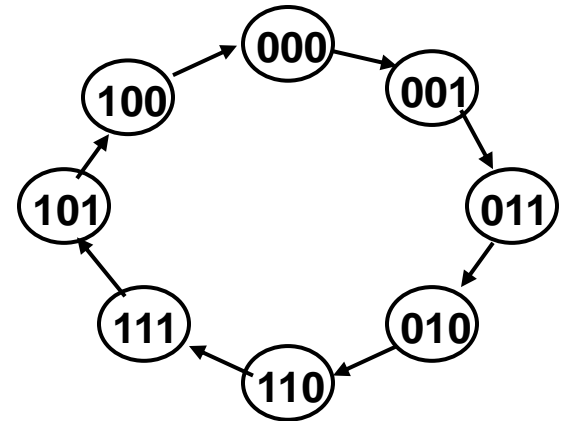
$$TQ_0 = 1$$

$$TQ_1 = Q_0'$$

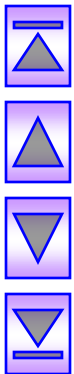
$$TQ_2 = Q_0' \cdot Q_1'$$

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.

		$Q_1 Q_0$			
		00	01	11	10
Q_2	0				1
	1	X	X	X	X

$JQ_2 = Q_1 \cdot Q_0'$

		$Q_1 Q_0$			
		00	01	11	10
Q_2	0		1	X	X
	1			X	X

$JQ_1 = Q_2' \cdot Q_0$

		$Q_1 Q_0$			
		00	01	11	10
Q_2	0	1	X	X	
	1		X	X	1

$JQ_0 = Q_2 \cdot Q_1 + Q_2' \cdot Q_1'$
 $= (Q_2 \oplus Q_1)'$

		$Q_1 Q_0$			
		00	01	11	10
Q_2	0	X	X	X	X
	1	1			

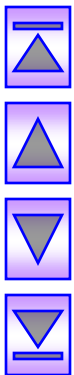
$KQ_2 = Q_1' \cdot Q_0'$

		$Q_1 Q_0$			
		00	01	11	10
Q_2	0	X	X		
	1	X	X	1	

$KQ_1 = Q_2 \cdot Q_0$

		$Q_1 Q_0$			
		00	01	11	10
Q_2	0	X		1	X
	1	X	1		X

$KQ_0 = Q_2 \cdot Q_1' + Q_2' \cdot Q_1$
 $= Q_2 \oplus Q_1$



Designing Synchronous Counters

- 3-bit Gray code counter: logic diagram.

$$JQ_2 = Q_1 \cdot Q_0'$$

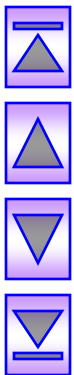
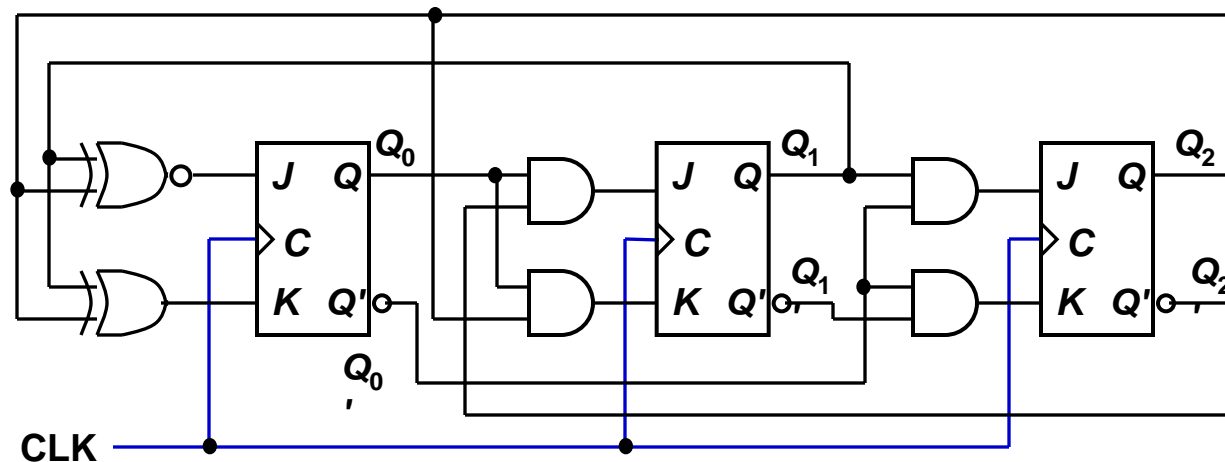
$$KQ_2 = Q_1' \cdot Q_0'$$

$$JQ_1 = Q_2' \cdot Q_0$$

$$KQ_1 = Q_2 \cdot Q_0$$

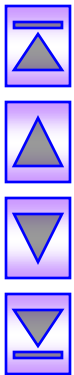
$$JQ_0 = (Q_2 \oplus Q_1)'$$

$$KQ_0 = Q_2 \oplus Q_1$$



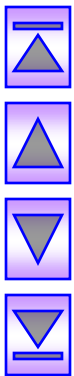
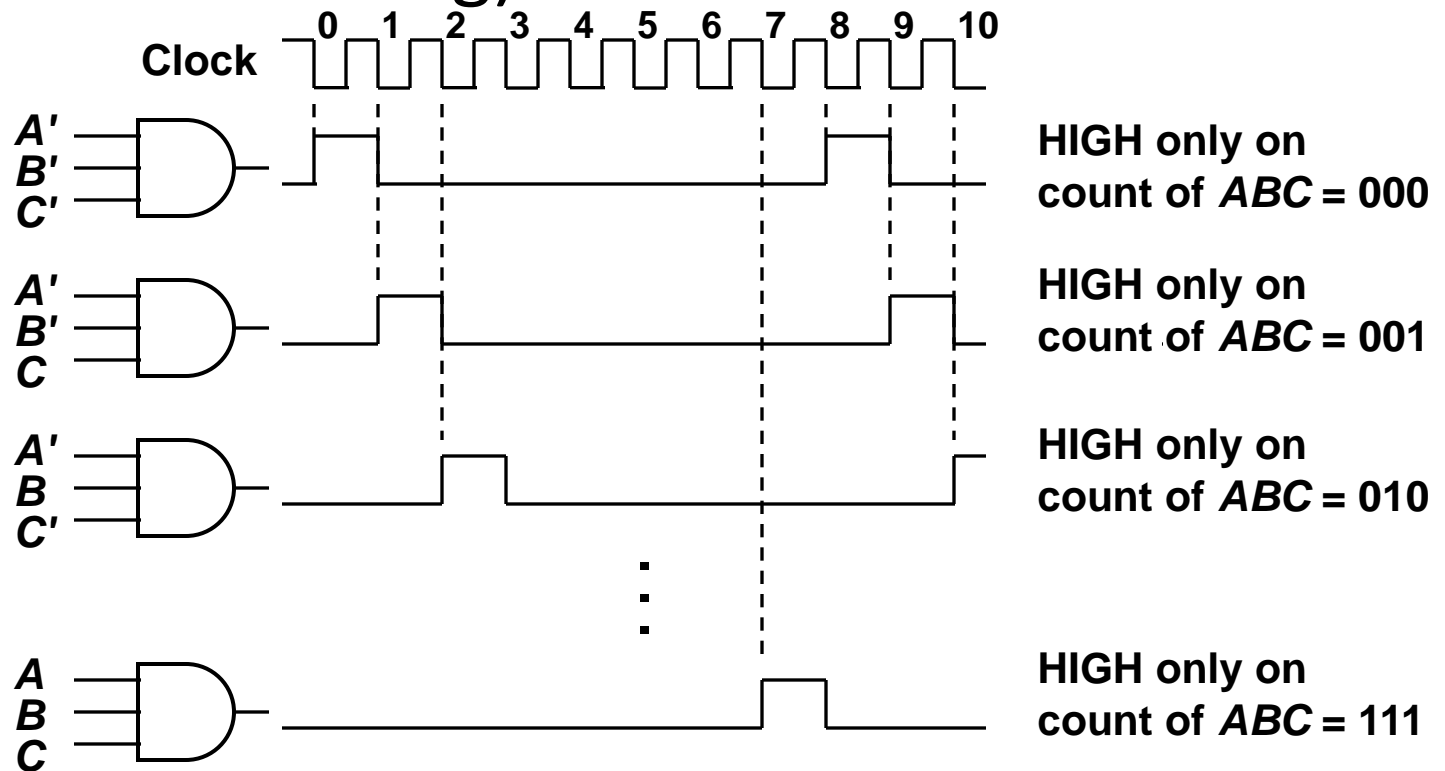
Decoding A Counter

- **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.



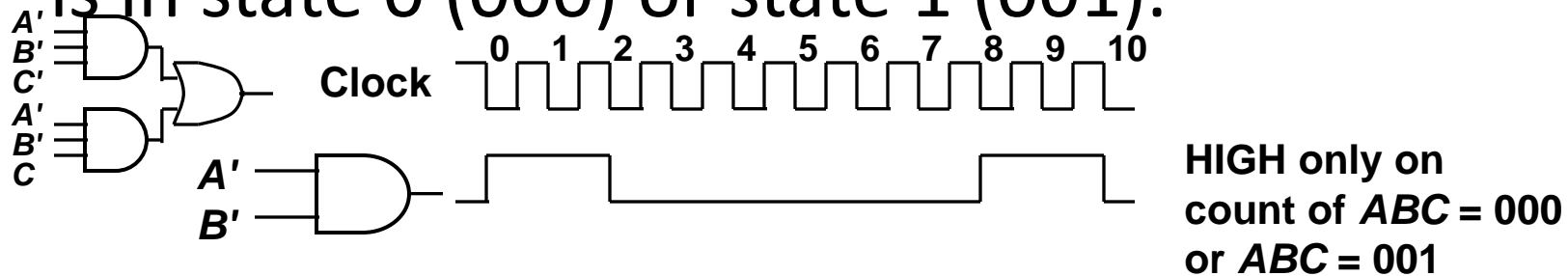
Decoding A Counter

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

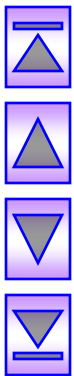
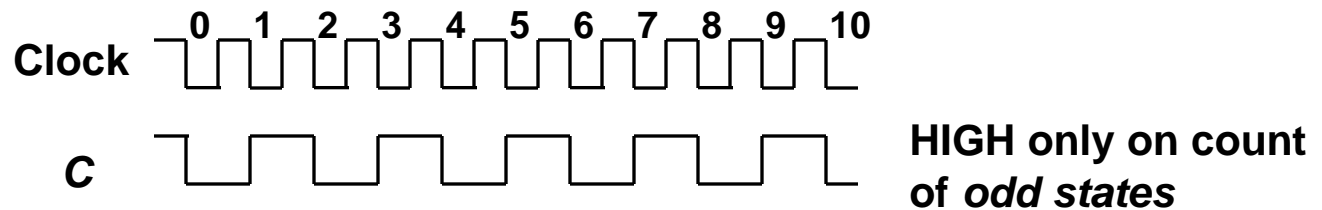


Decoding A Counter

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

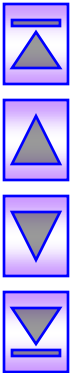


- Example: To detect that a MOD-8 counter is in the odd states (states 1, 3, 5 or 7), simply use C .



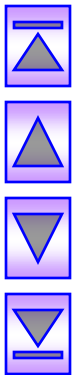
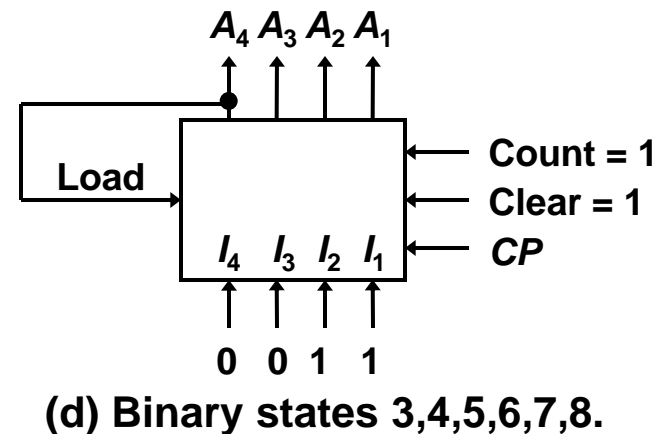
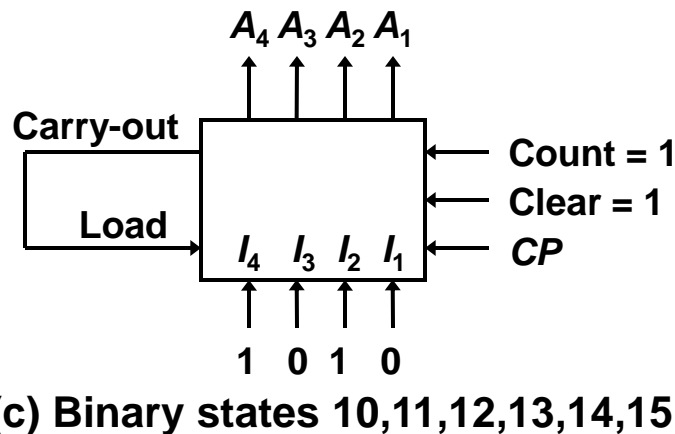
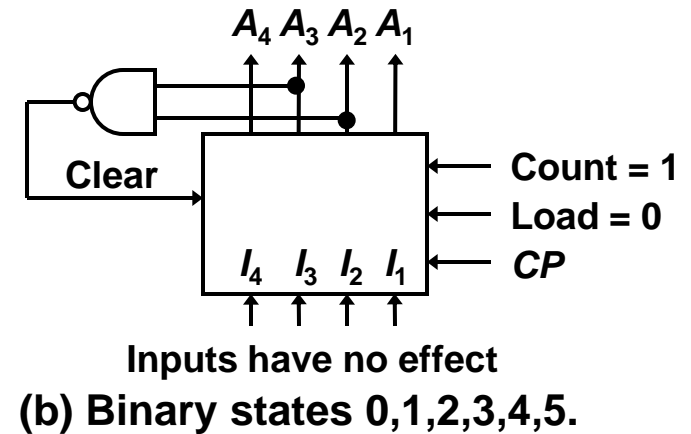
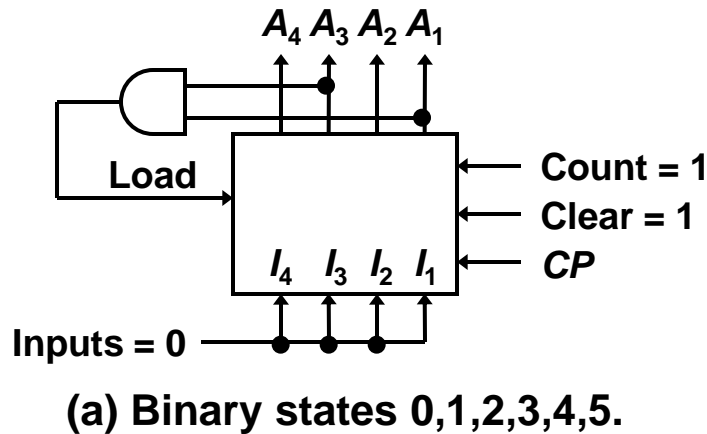
Counters with Parallel Load

- 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

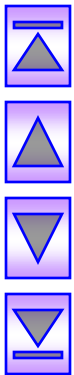
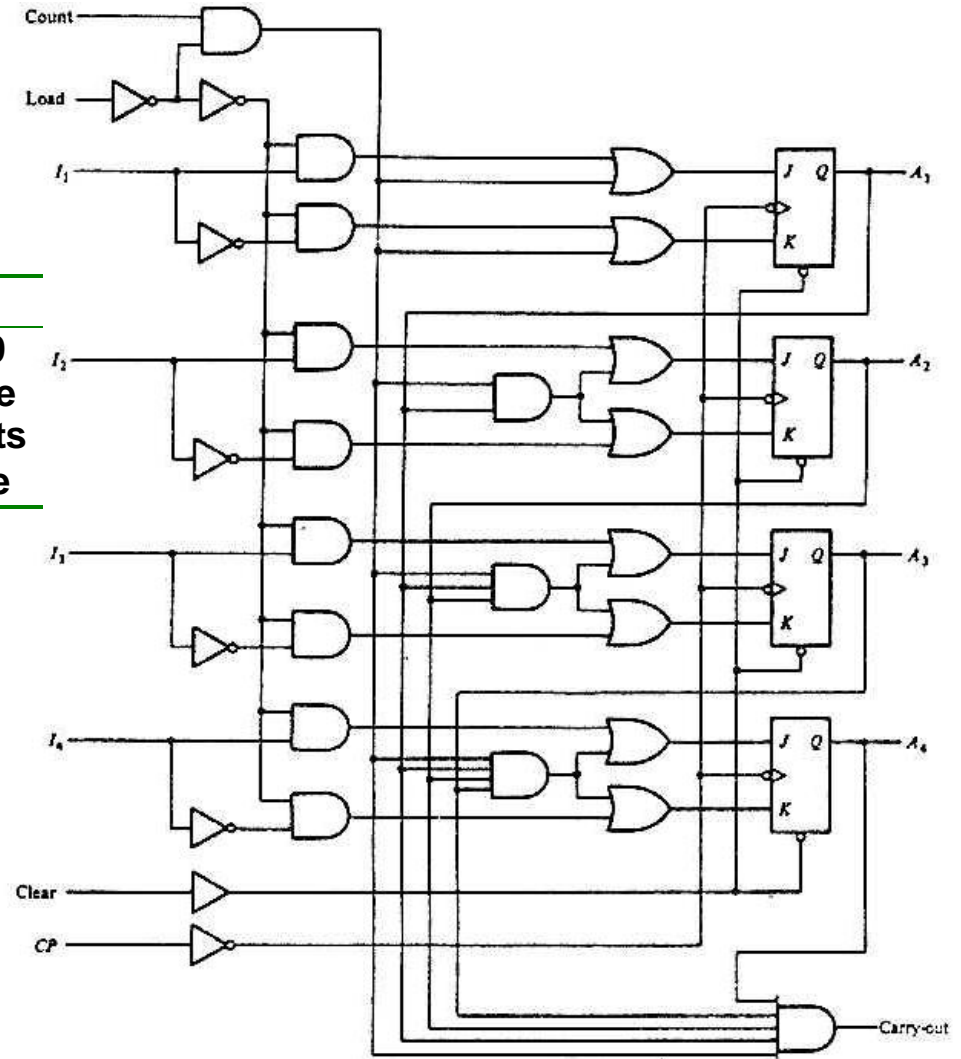
- Different ways of getting a MOD-6 counter:



Counters with Parallel Load

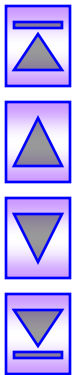
- 4-bit counter with parallel load

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



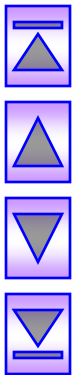
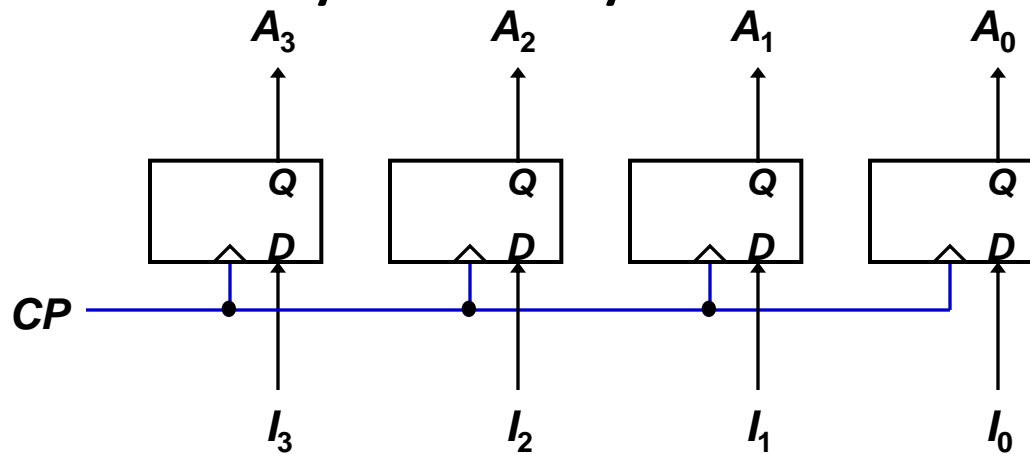
Introduction: Registers

- 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)



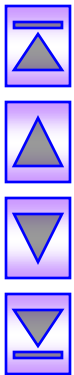
Simple Registers

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

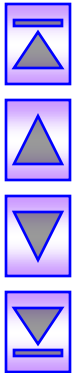
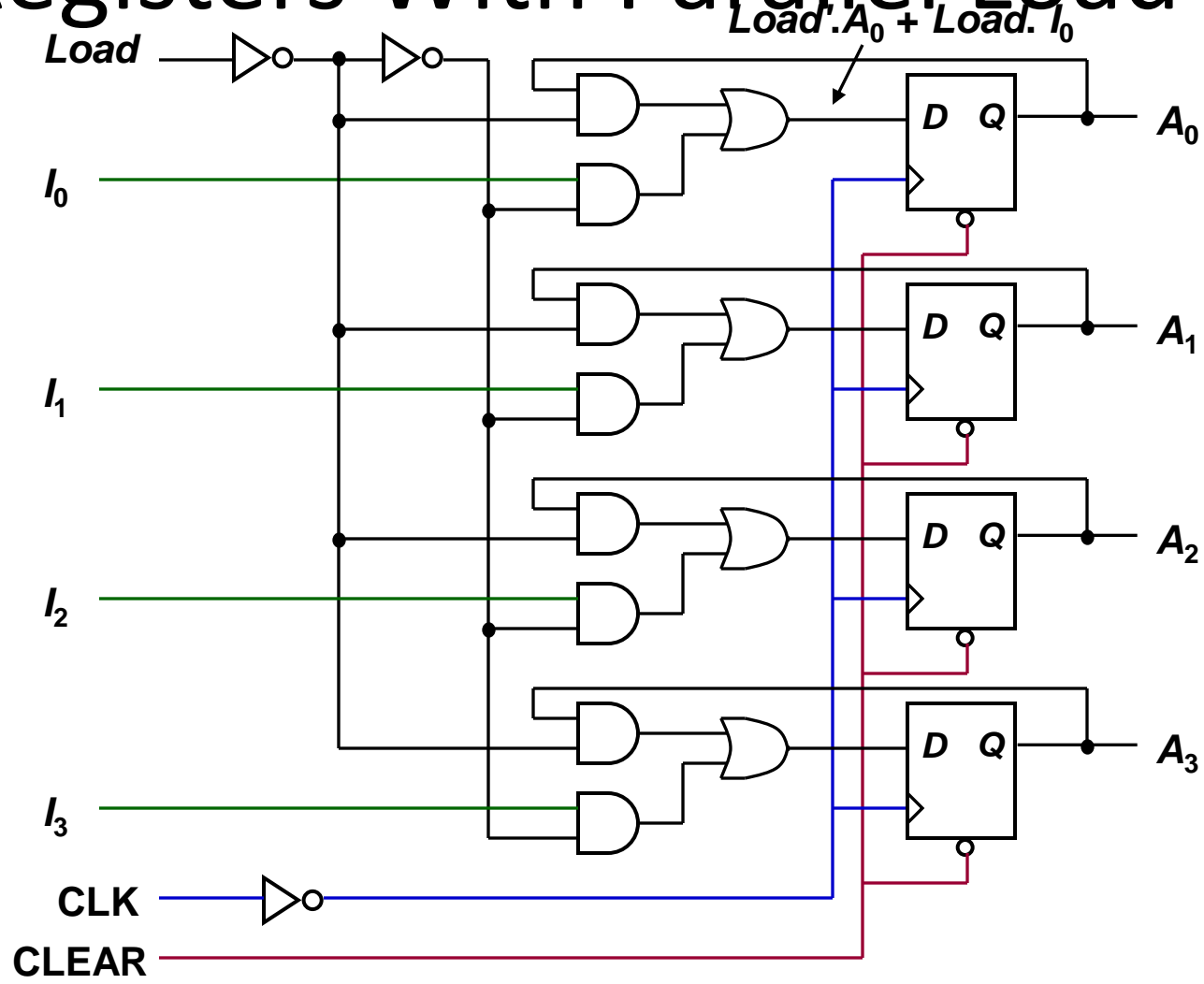


Registers With Parallel Load

- 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.

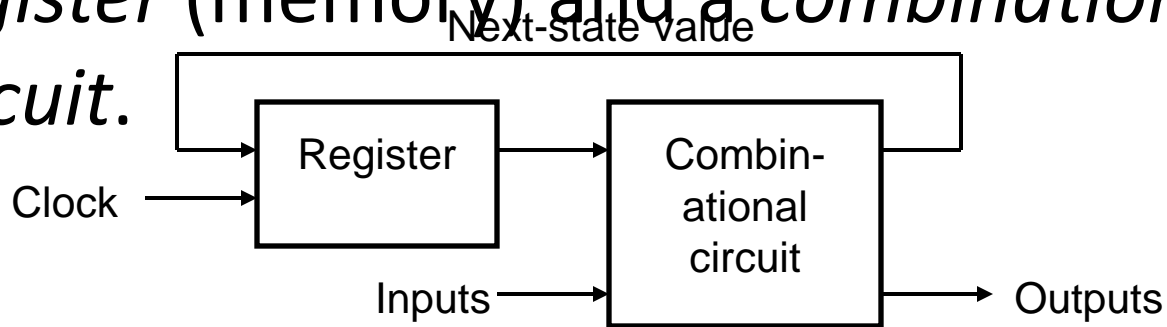


Registers With Parallel Load

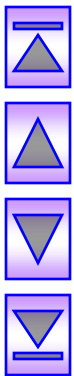


Using Registers to implement Sequential Circuits

- 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*.

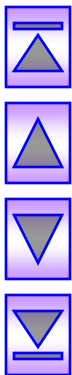
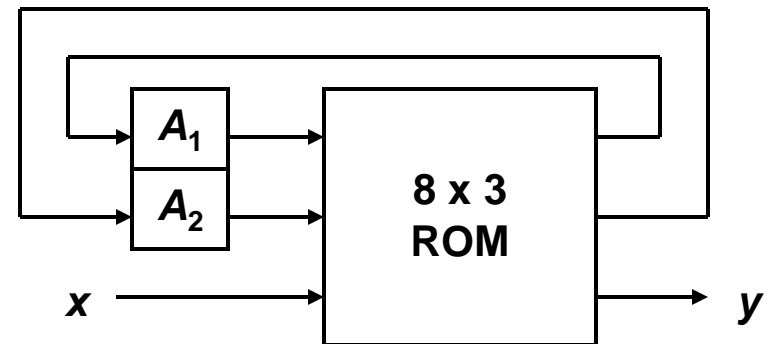


Using Registers to implement Sequential Circuits

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

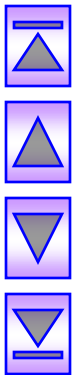
Address			Outputs		
1	2	3	1	2	3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

ROM truth table



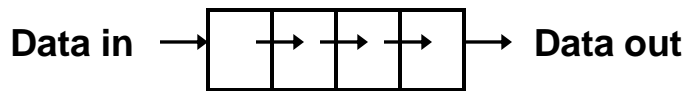
Shift Registers

- 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.

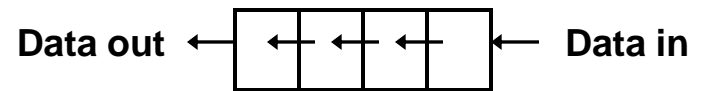


Shift Registers

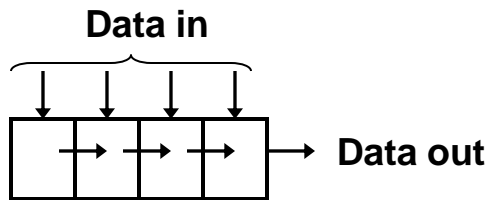
- Basic data movement in shift registers (four bits are used for illustration).



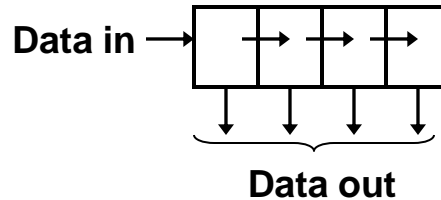
(a) Serial in/shift right/serial out



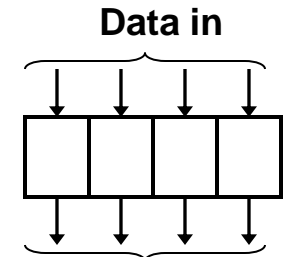
(b) Serial in/shift left/serial out



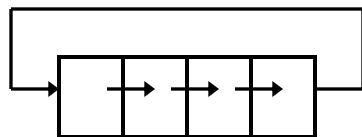
(c) Parallel in/serial out



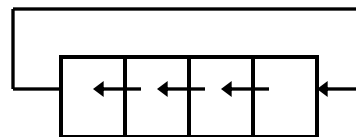
(d) Serial in/parallel out



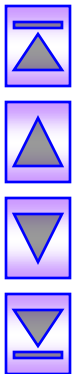
(e) Parallel in / parallel out



(f) Rotate right

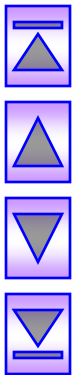
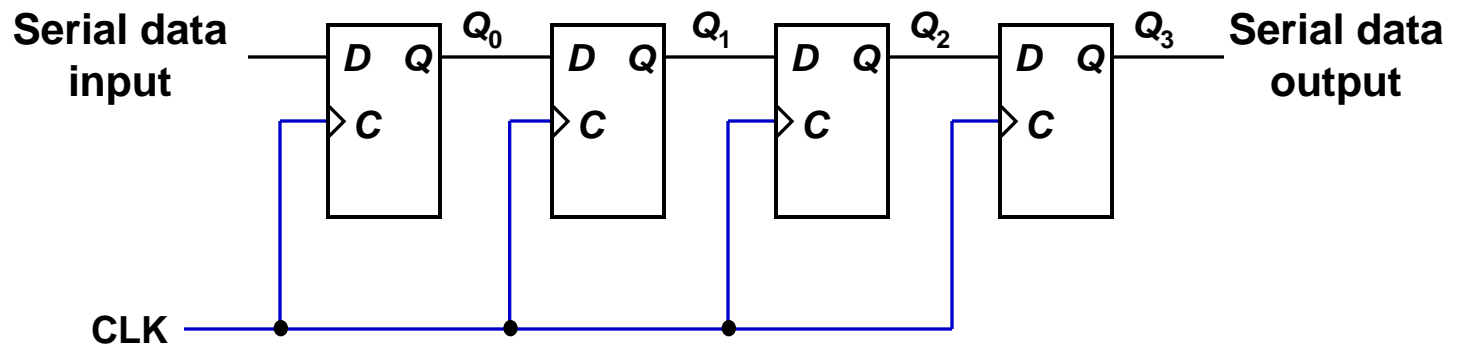


(g) Rotate left



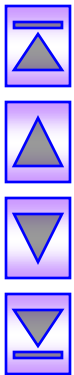
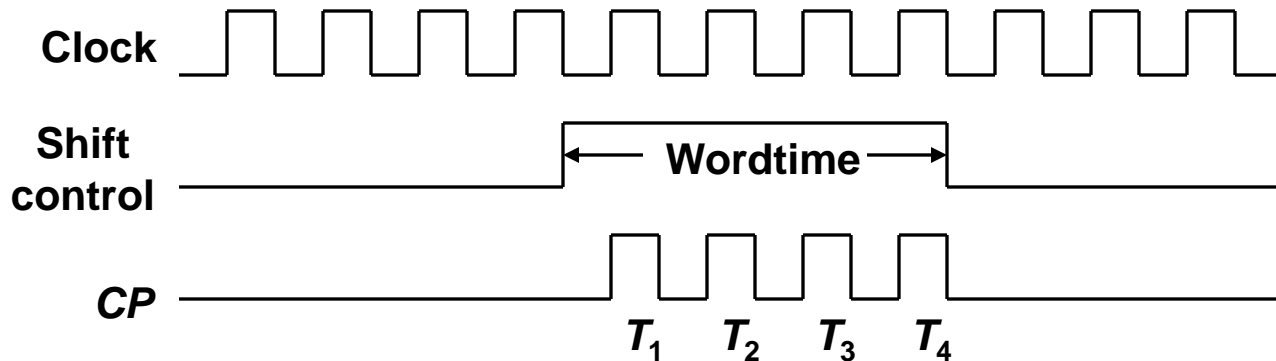
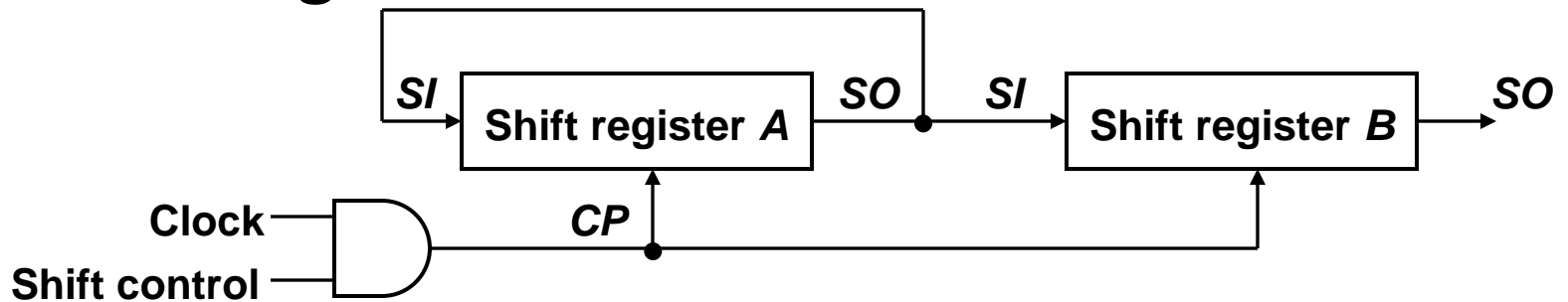
Serial In/Serial Out Shift Registers

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



Serial In/Serial Out Shift Registers

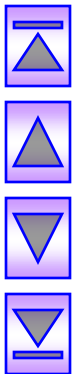
- Application: Serial transfer of data from one register to another.



Serial In/Serial Out Shift Registers

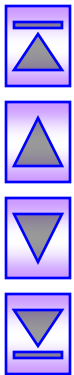
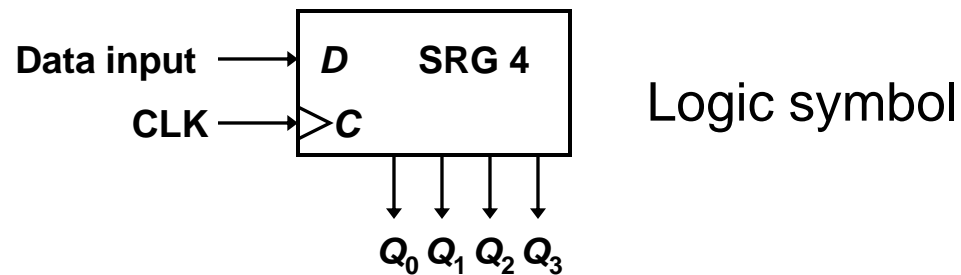
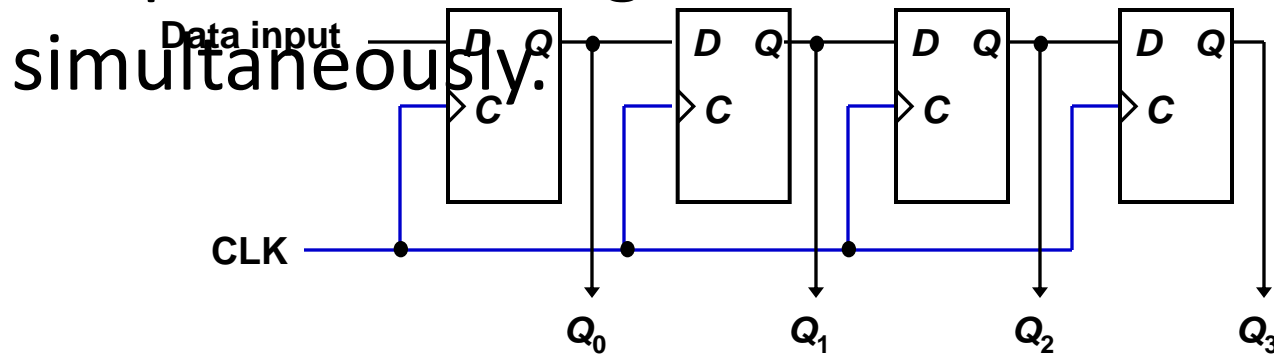
- Serial-transfer example.

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



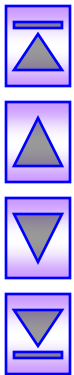
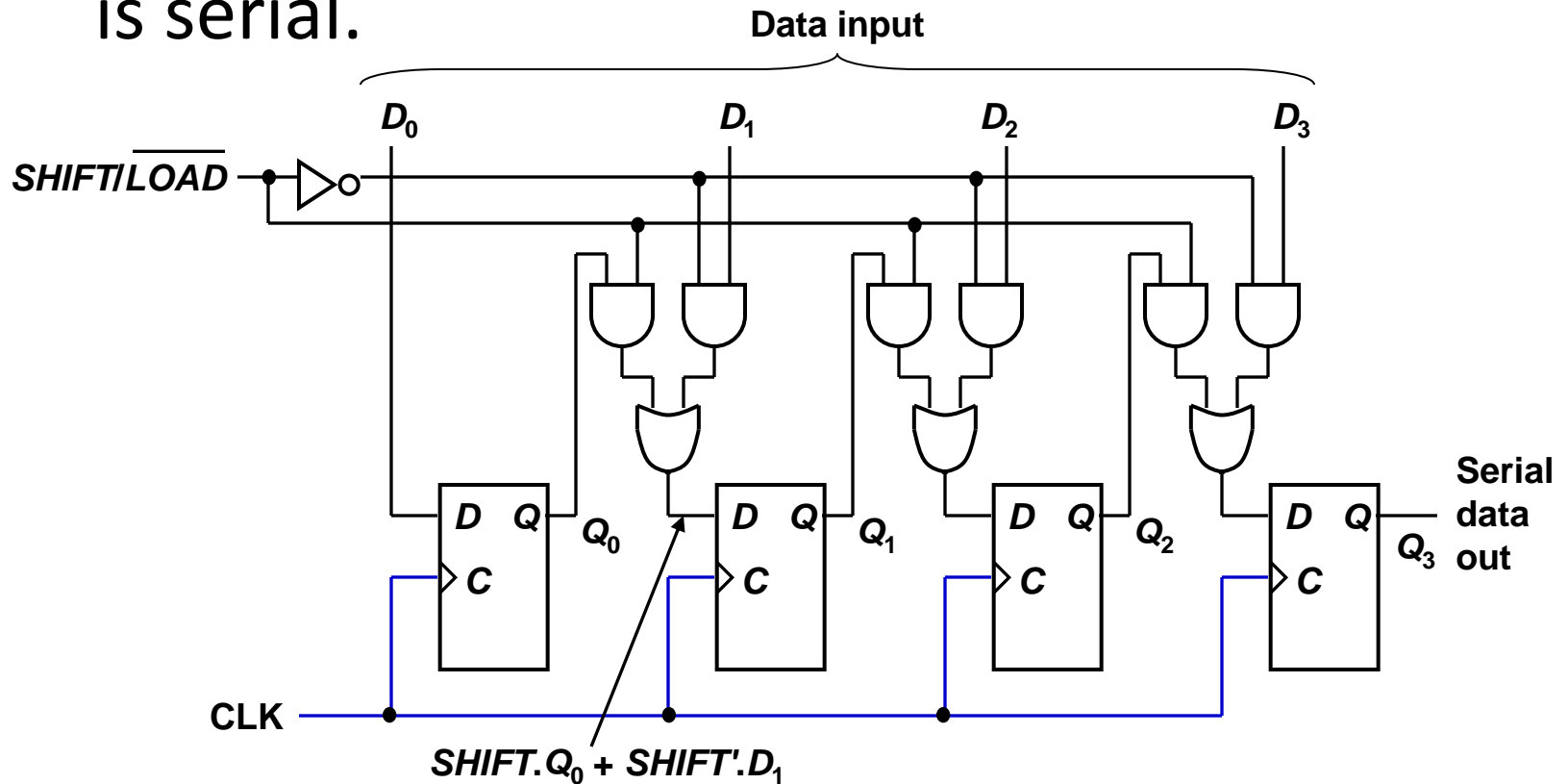
Serial In/Parallel Out Shift Registers

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



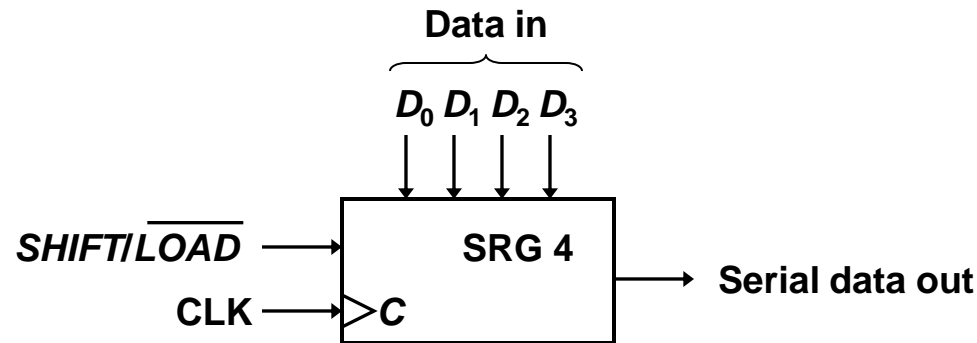
Parallel In/Serial Out Shift Registers

- Bits are entered simultaneously, but output is serial.

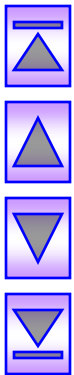


Parallel In/Serial Out Shift Registers

- Bits are entered simultaneously, but output is serial.

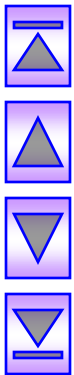
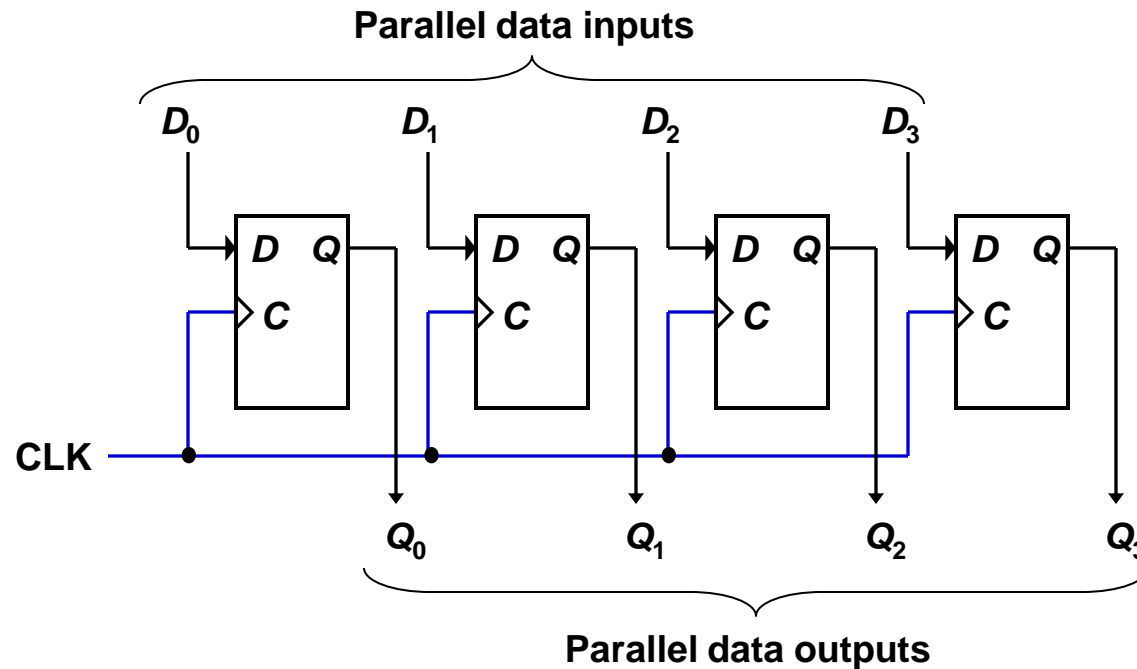


Logic symbol



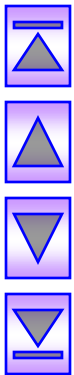
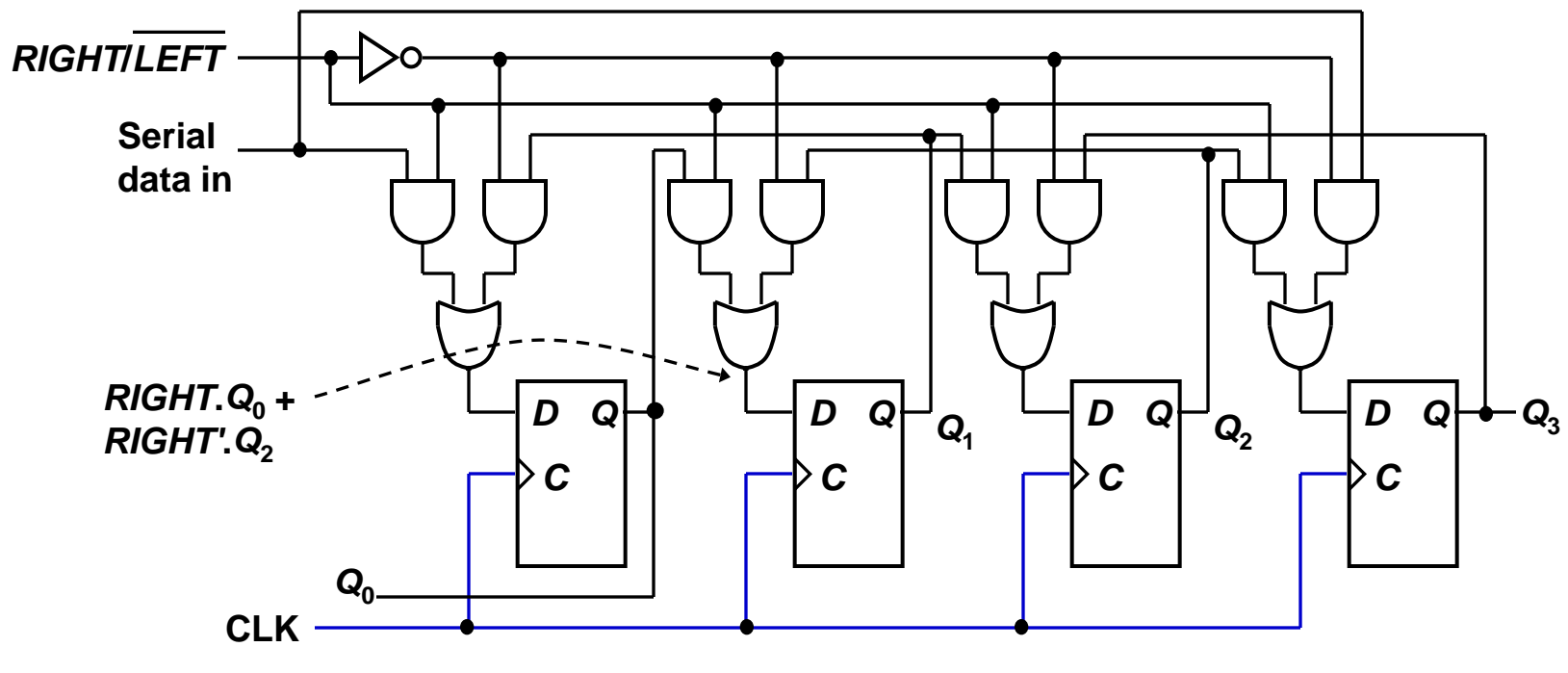
Parallel In/Parallel Out Shift Registers

- Simultaneous input and output of all data bits.



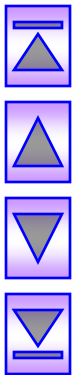
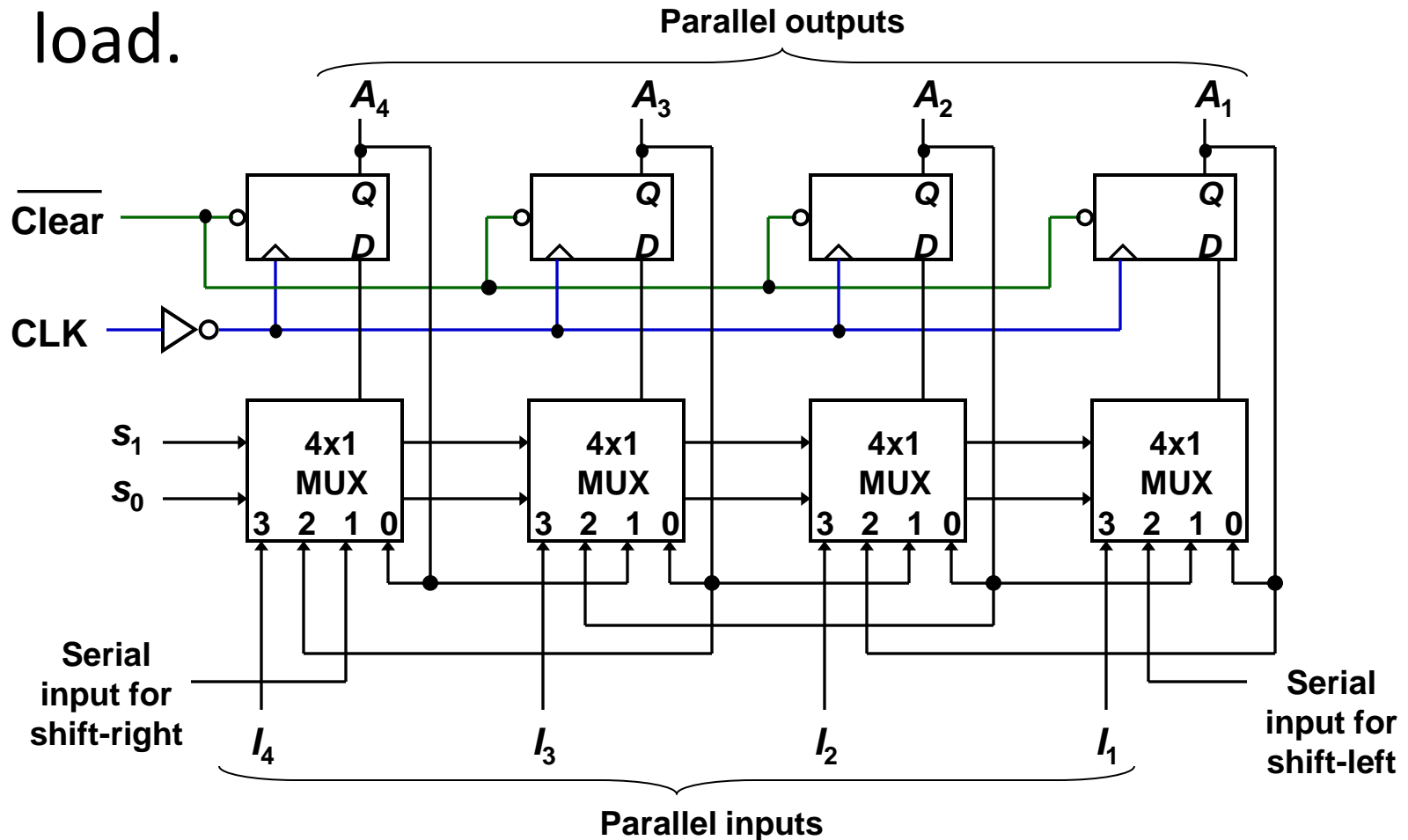
Bidirectional Shift Registers

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



Bidirectional Shift Registers

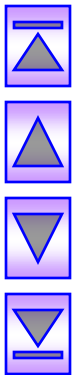
- 4-bit bidirectional shift register with parallel load.



Bidirectional Shift Registers

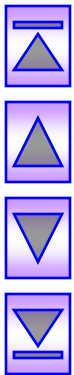
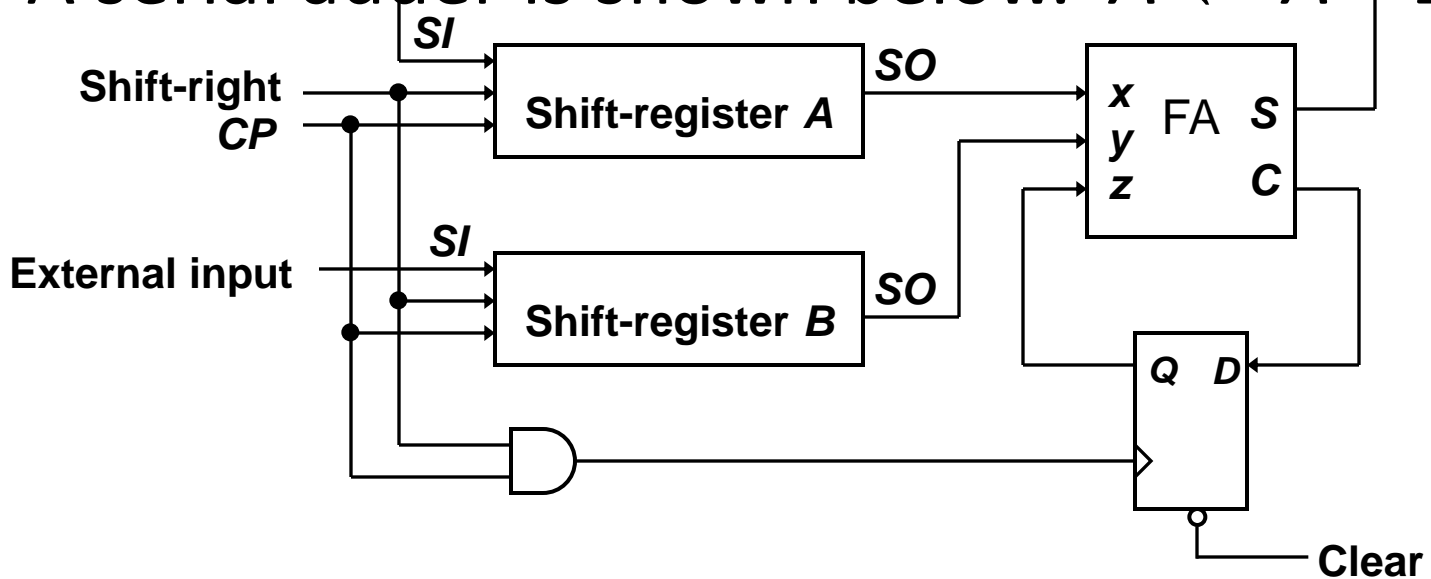
- 4-bit bidirectional shift register with parallel load.

<i>Mode Control</i>		<i>Register Operation</i>
<i>s₁</i>	<i>s₀</i>	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load



An Application – Serial Addition

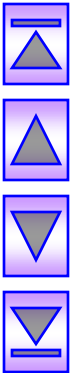
- Most operations in digital computers are done in parallel. Serial operations are slower but require less equipment.
- A serial adder is shown below. $A \leftarrow A + B$.



An Application – Serial Addition

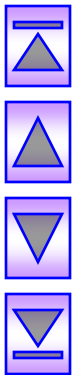
- $A = 0100$; $B = 0111$. $A + B = 1011$ is stored in A after 4 clock pulses.

Initial:	A: 0 1 0 0	Q: 0
	B: 0 1 1 1	
<hr/>		
Step 1: 0 + 1 + 0	A: 1 0 1 0	Q: 0
S = 1, C = 0	B: x 0 1 1	
<hr/>		
Step 2: 0 + 1 + 0	A: 1 1 0 1	Q: 0
S = 1, C = 0	B: x x 0 1	
<hr/>		
Step 3: 1 + 1 + 0	A: 0 1 1 0	Q: 1
S = 0, C = 1	B: x x x 0	
<hr/>		
Step 4: 0 + 0 + 1	A: 1 0 1 1	Q: 0
S = 1, C = 0	B: x x x x	



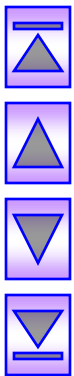
Shift Register Counters

- **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*.



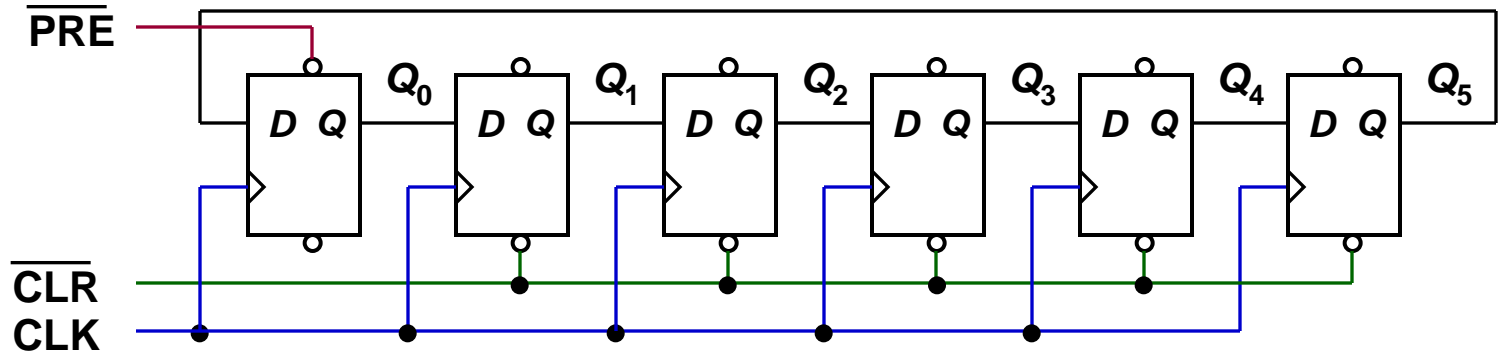
Ring Counters

- 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.

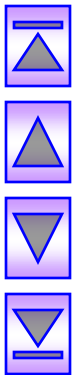
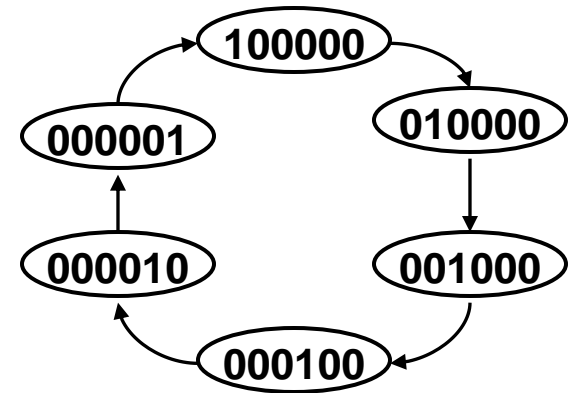


Ring Counters

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

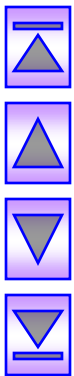


Clock	Q ₀	Q ₁	Q ₂	Q ₃	Q ₄	Q ₅
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1



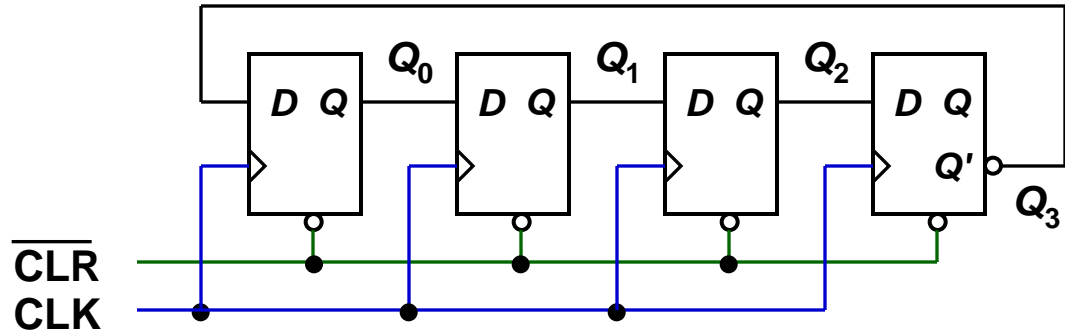
Johnson Counters

- 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 $2n$ states.
- Require more decoding circuitry than ring counter but less than binary counters.

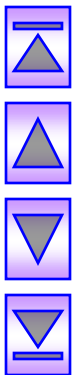
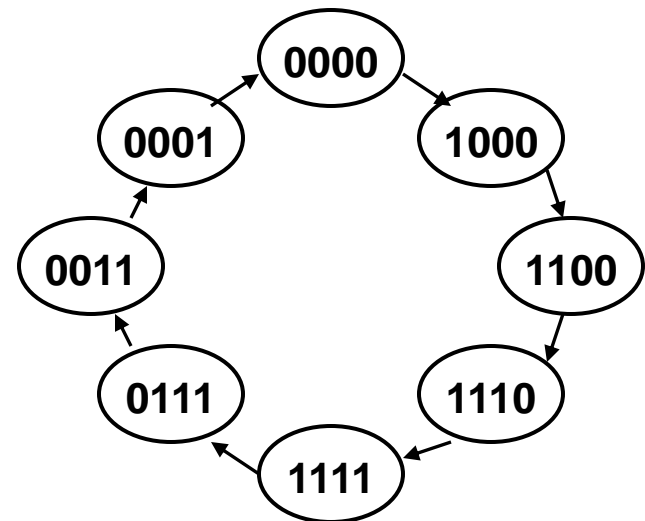


Johnson Counters

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



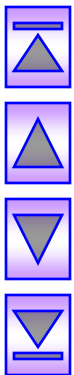
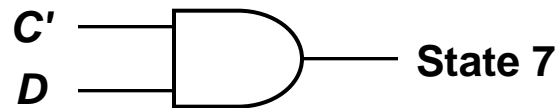
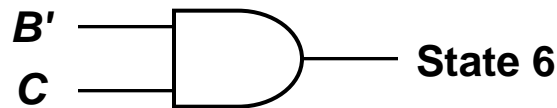
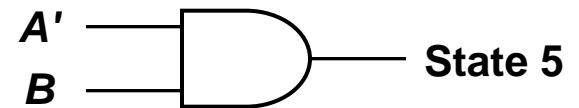
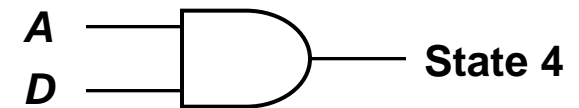
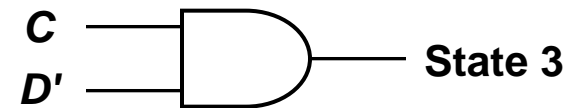
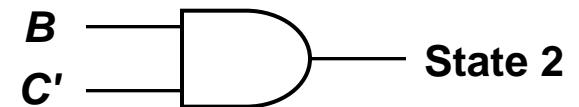
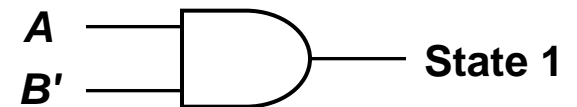
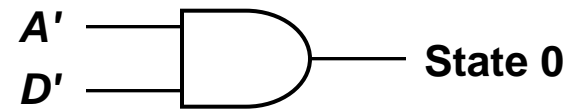
Clock	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1



Johnson Counters

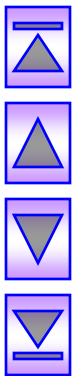
- Decoding logic for a 4-bit Johnson counter.

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)

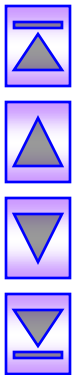
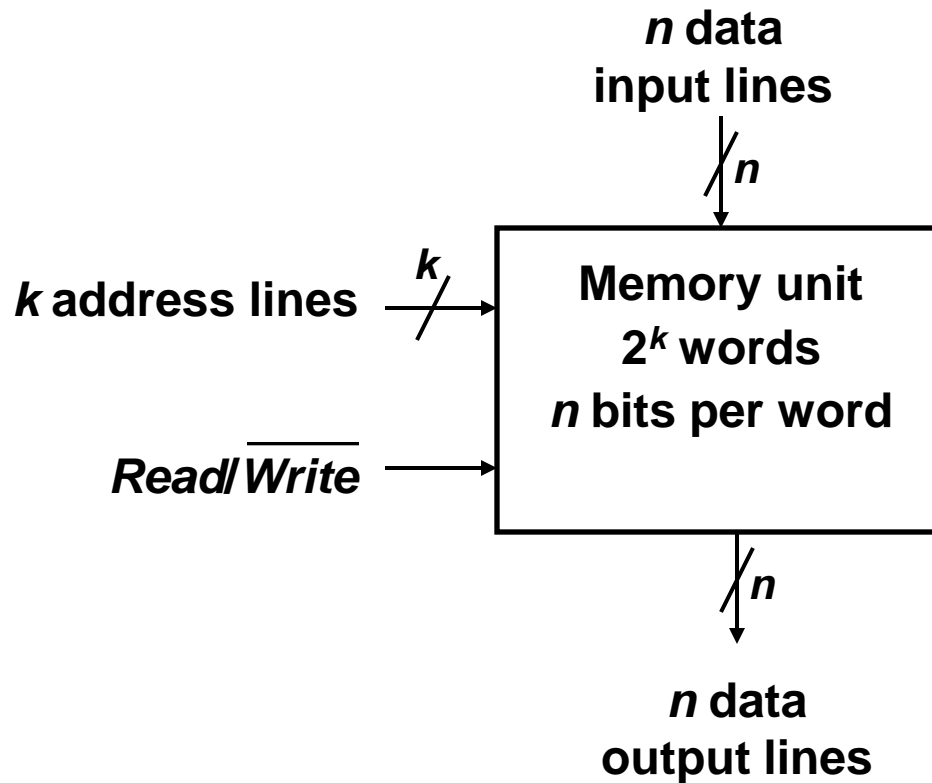
- A memory unit stores binary information in groups of bits called *words*.
- The data consists of n lines (for n -bit words). **Data input lines** provide the information to be stored (*written*) into the memory, while **data output lines** carry the information out (*read*) from the memory.
- The **address** 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

Random Access Memory (RAM)

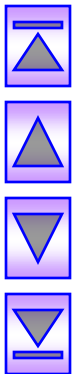
- Block diagram of a memory unit:



Random Access Memory (RAM)

- Content of a 1024 x 16-bit memory:

Memory address		Memory content
binary	decimal	
000000000	0	1011010111011101
000000001	1	1010000110000110
000000010	2	0010011101110001
:	:	:
:	:	:
111111101	1021	1110010101010010
111111110	1022	0011111010101110
111111111	1023	1011000110010101



Random Access Memory (RAM)

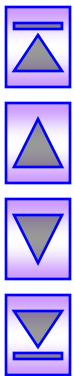
- The **Write** operation:

- ❖ 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 *Read/Write* to 0)

- The **Read** operation:

- ❖ Transfers the address of the desired word to the address lines

- ❖ Activates the *Read* control line (set *Read/Write*

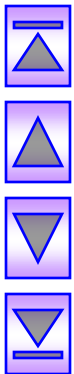


Random Access Memory (RAM)

- The Read/Write operation:

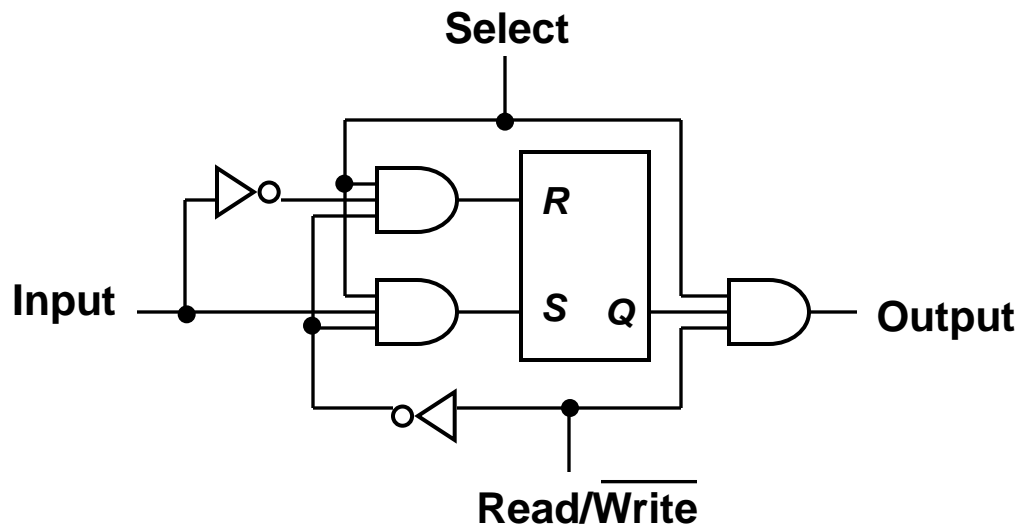
Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

- 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.

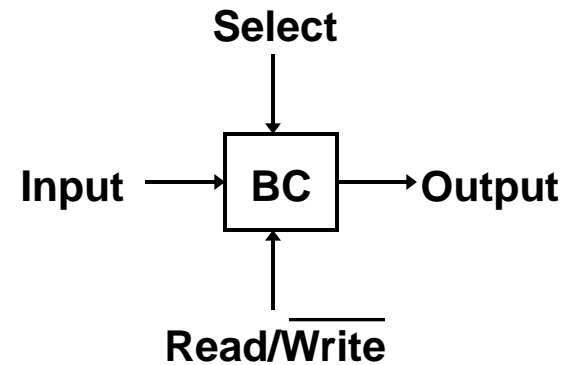


Random Access Memory (RAM)

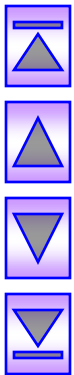
- A single memory cell of the static RAM has the following logic and block diagrams.



Logic diagram

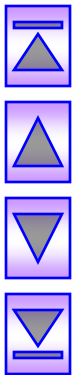
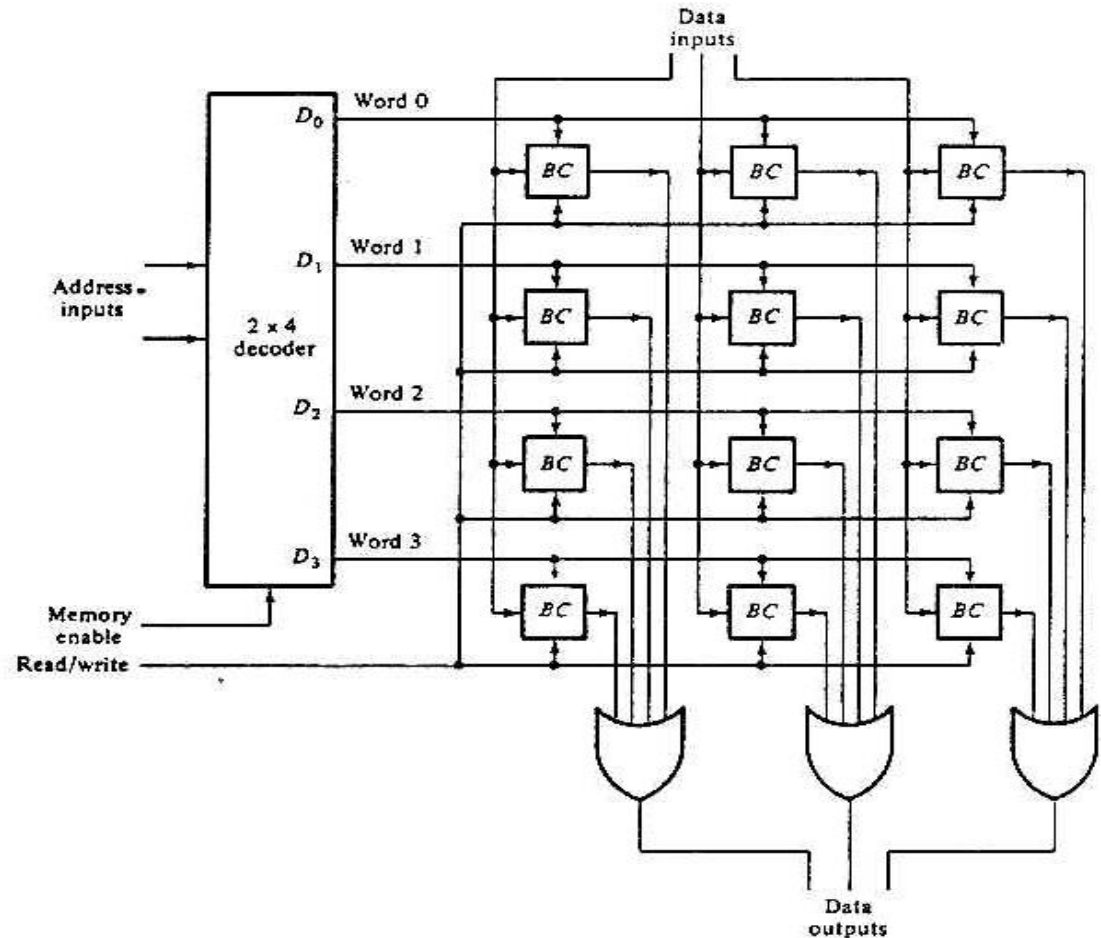


Block diagram



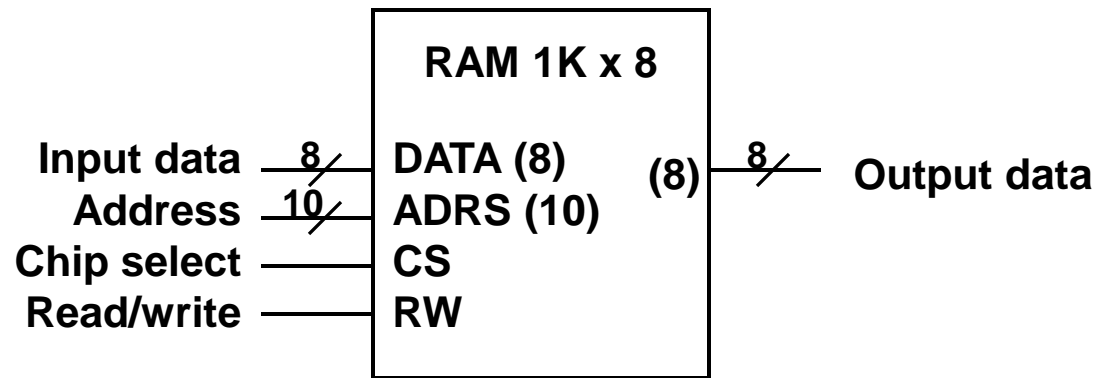
Random Access Memory (RAM)

- Logic construction of a 4 x 3 RAM (with decoder and BC)

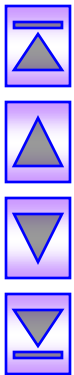


Random Access Memory (RAM)

- 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)

