Summary

Module II (10 Hours)

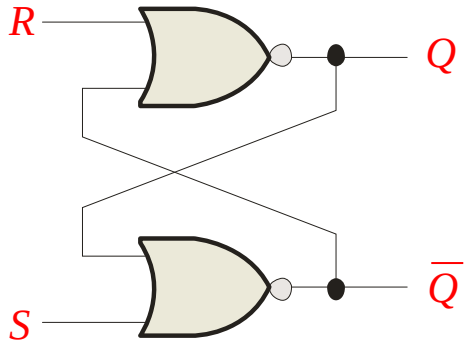
Sequential circuit - Clocking, Flip flops - SR – JK- D -T flip flops, Counters - Synchronous and asynchronous counters - UP/DOWN counters , Registers - Serial in serial out - Serial in parallel out - Parallel in serial out - Parallel in parallel out registers

Summary

Latches

A **latch** is a temporary storage device that has two stable states (bistable). It is a basic form of memory.

The S-R (Set-Reset) latch is the most basic type. It can be constructed from NOR gates or NAND gates. With NOR gates, the latch responds to active-HIGH inputs; with NAND gates, it responds to active-LOW inputs.



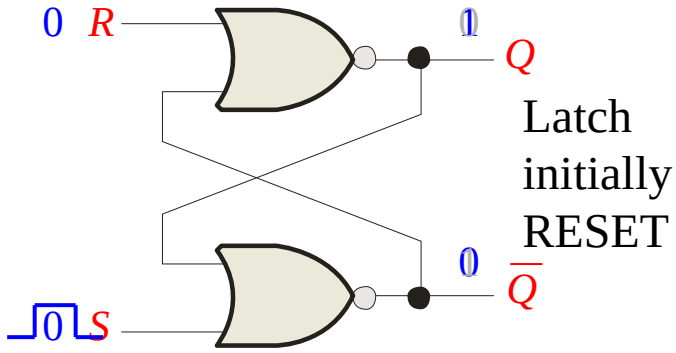
NOR Active-HIGH Latch

Summary

Latches

The active-HIGH *S*-*R* latch is in a stable (latched) condition when both inputs are LOW.

Assume the latch is initially RESET (Q = 0) and the inputs are at their inactive level (0). To SET the latch (Q = 1), a momentary HIGH signal is applied to the *S* input while the *R* remains LOW.



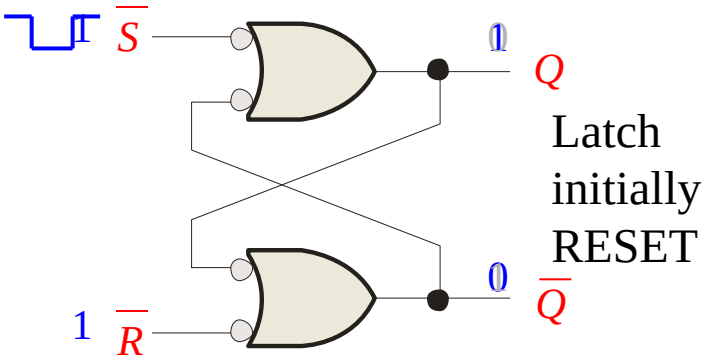
To RESET the latch (Q = 0), a momentary HIGH signal is applied to the *R* input while the *S* remains LOW.

Summary

Latches

The active-LOW $\overline{S}$-$\overline{R}$ latch is in a stable (latched) condition when both inputs are HIGH.

Assume the latch is initially RESET (Q = 0) and the inputs are at their inactive level (1). To SET the latch (Q = 1), a momentary LOW signal is applied to the $\overline{S}$ input while the $\overline{R}$ remains HIGH.



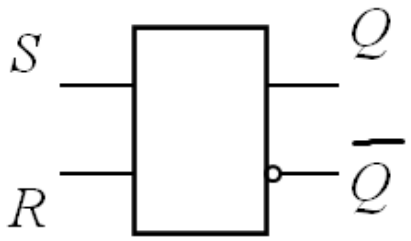
To RESET the latch a momentary LOW is applied to the $\overline{R}$ input while $\overline{S}$ is HIGH.

Never apply an active set and reset at the same time (invalid).

S-R Latch

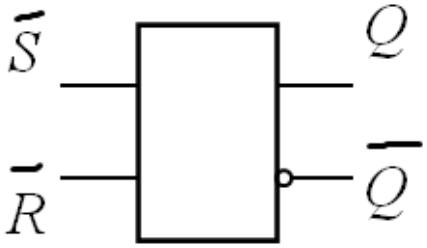
- Characteristics table for active-high input S-R latch:

S	*R*	*Q*	*Q'*	
0	0	NC	NC	No change. Latch remained in present state.
1	0	1	0	Latch SET.
0	1	0	1	Latch RESET.
1	1	0	0	Invalid condition.



- Characteristics table for active-low input S'-R' latch:

S'	*R'*	*Q*	*Q'*	
1	1	NC	NC	No change. Latch remained in present state.
0	1	1	0	Latch SET.
1	0	0	1	Latch RESET.
0	0	1	1	Invalid condition.

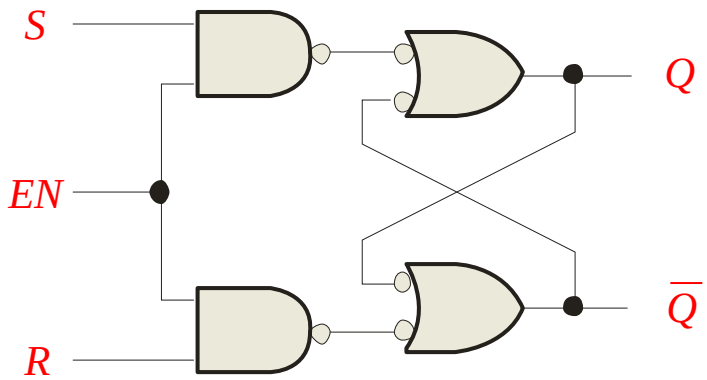


Summary

Latches

A gated latch is a variation on the basic latch.

The gated latch has an additional input, called enable (*EN*) that must be HIGH in order for the latch to respond to the *S* and *R* inputs.

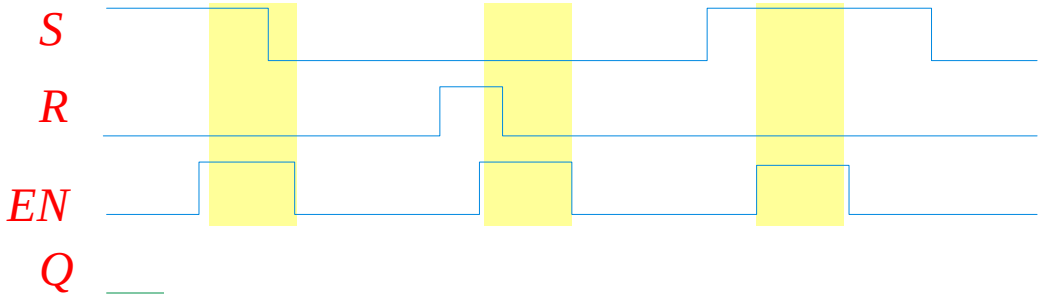


Example

Show the *Q* output with relation to the input signals. Assume *Q* starts LOW.

Solution

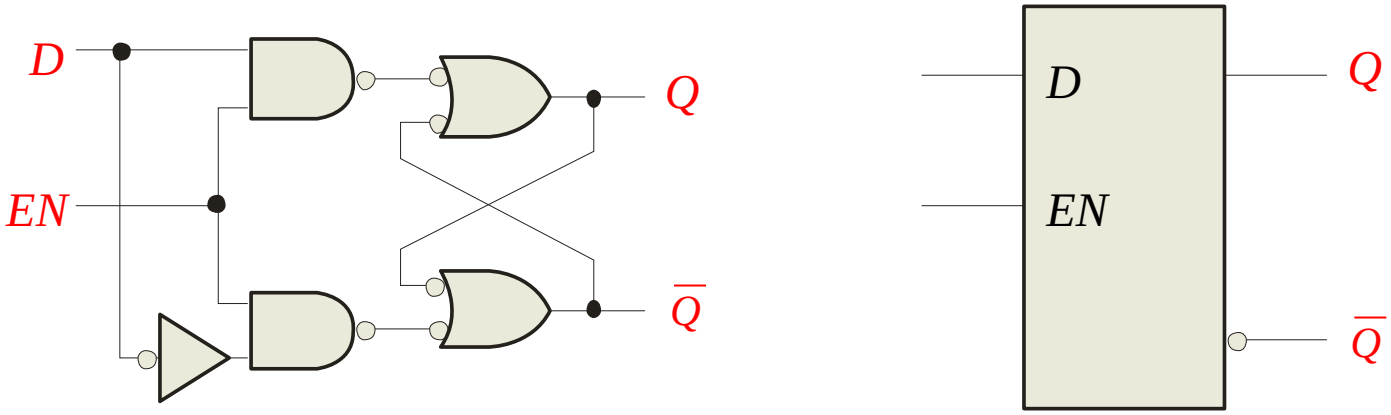
Keep in mind that *S* and *R* are only active when *EN* is HIGH.



Summary

Latches

The *D* latch is an variation of the *S*-*R* latch but combines the *S* and *R* inputs into a single *D* input as shown:



A simple rule for the *D* latch is:

Q follows *D* when the Enable is active.

Summary

Latches

The truth table for the *D* latch summarizes its operation. If *EN* is LOW, then there is no change in the output and it is latched.

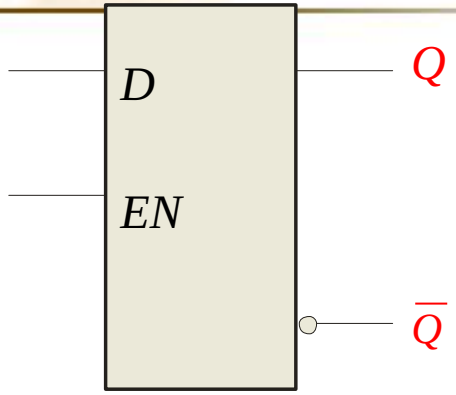
Inputs		Outputs		
D	*EN*	*Q*	$\bar{Q}$	Comments
0	1	0	1	RESET
1	1	1	0	SET
X	0	Q_0	$\bar{Q}_0$	No change

C	D	Q
0	x	No change
1	0	0
1	1	1

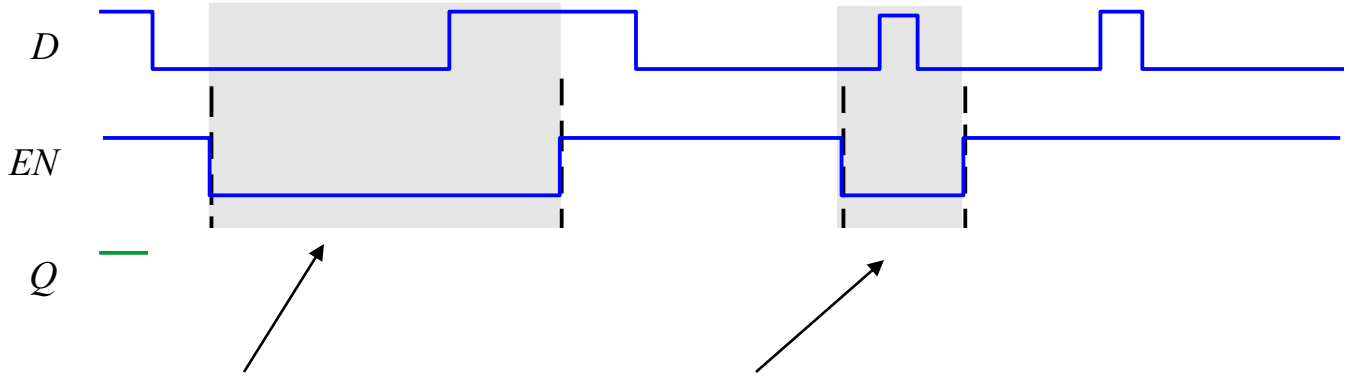
Summary

Latches

Example



Determine the *Q* output for the *D* latch, given the inputs shown.



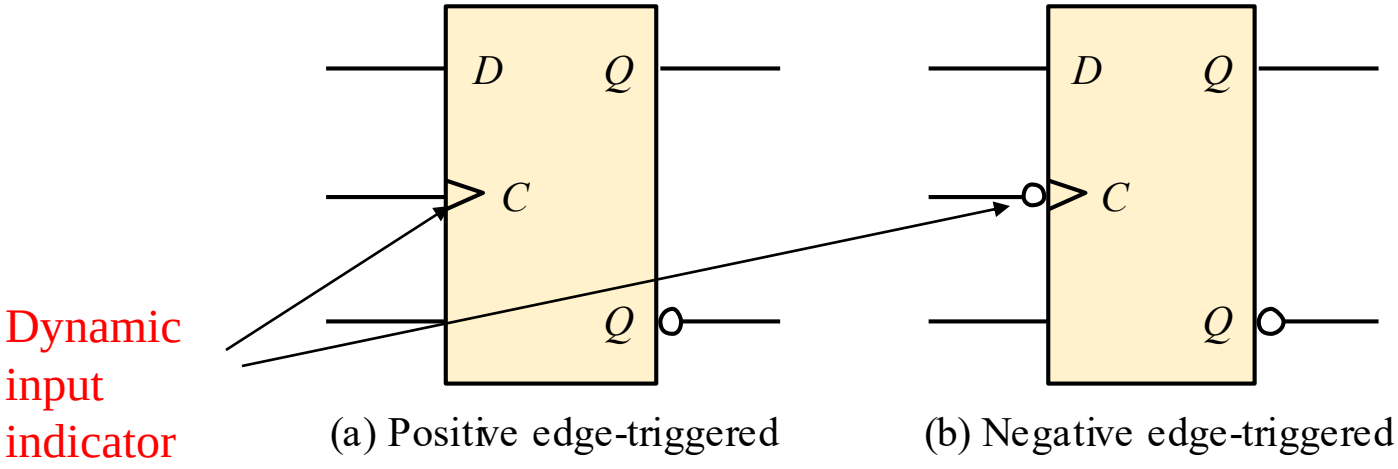
Notice that the Enable is not active during these times, so the output is latched.

Summary

Flip-flops

A flip-flop differs from a latch in the manner it changes states. A flip-flop is a clocked device, in which only the clock edge determines when a new bit is entered.

The active edge can be positive or negative.



FF vs. Latch

- Latches and flip-flops (FFs) are the basic building blocks of sequential circuits.
  - latch: bistable memory device with level sensitive triggering (no clock), watches all of its inputs continuously and changes its outputs, independent of a clocking signal.
  - flip-flop: bistable memory device with edge-triggering (with clock), samples its inputs, and changes its output only at times determined by a clocking signal.

Summary

Flip-flops

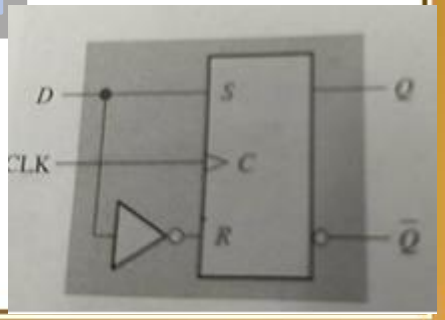
The truth table for a positive-edge triggered D flip-flop shows an up arrow to remind you that it is sensitive to its *D* input only on the rising edge of the clock; otherwise it is latched. The truth table for a negative-edge triggered D flip-flop is identical except for the direction of the arrow.

Inputs		Outputs		
D	CLK	*Q*	$\overline{Q}$	Comments
1	↑	1	0	SET
0	↑	0	1	RESET

(a) Positive-edge triggered

Inputs		Outputs		
D	CLK	*Q*	$\overline{Q}$	Comments
1	↓	1	0	SET
0	↓	0	1	RESET

(b) Negative-edge triggered



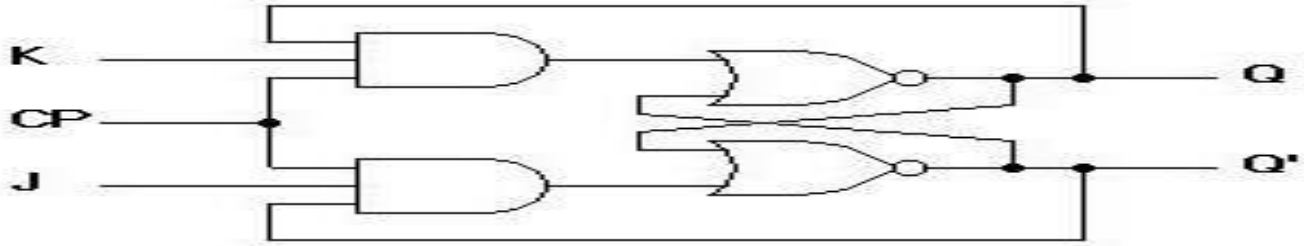
Summary

Flip-flops

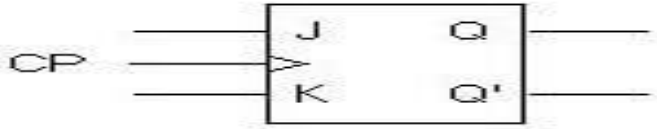
The J-K flip-flop is more versatile than the D flip flop. In addition to the clock input, it has two inputs, labeled *J* and *K*. When both *J* and *K* = 1, the output changes states (toggles) on the active clock edge (in this case, the rising edge).

Inputs			Outputs		
J	*K*	CLK	*Q*	$\overline{Q}$	Comments
0	0	↑	Q_0	$\overline{Q}_0$	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	$\overline{Q}_0$	Q_0	Toggle

JK Flip-Flop



(a) Logic diagram



(b) Graphical symbol

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

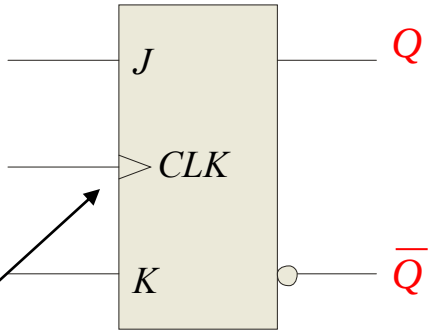
(c) Transition table

Clocked JK flip-flop

C	J	K	Q_{next}
0	x	x	No change
1	0	0	No change
1	0	1	0 (reset)
1	1	0	1 (set)
1	1	1	$Q'_{current}$

Summary

Flip-flops

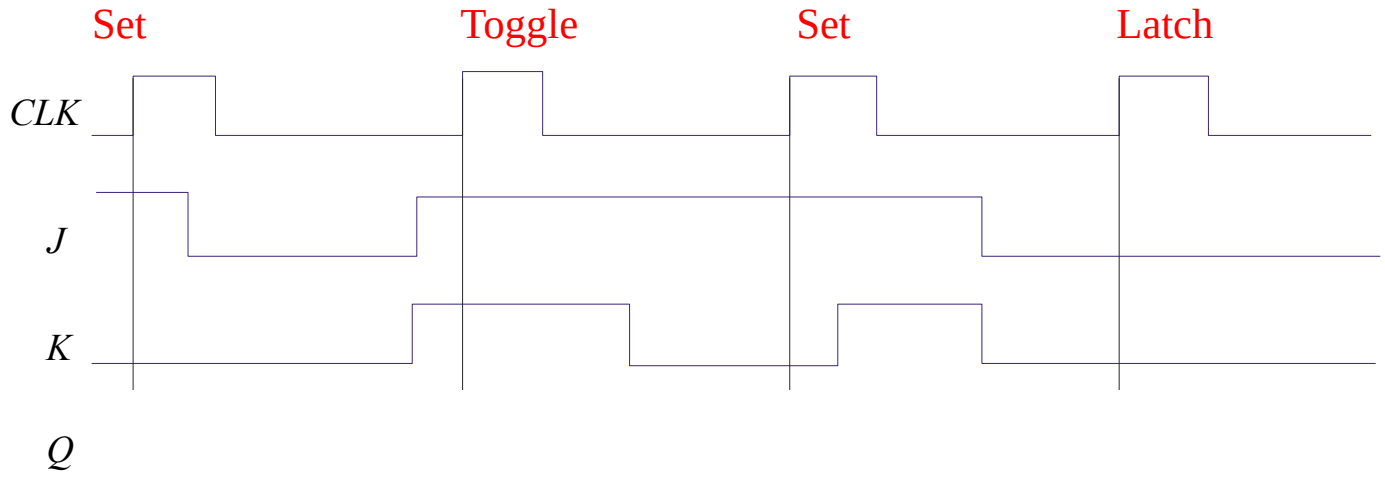


Example

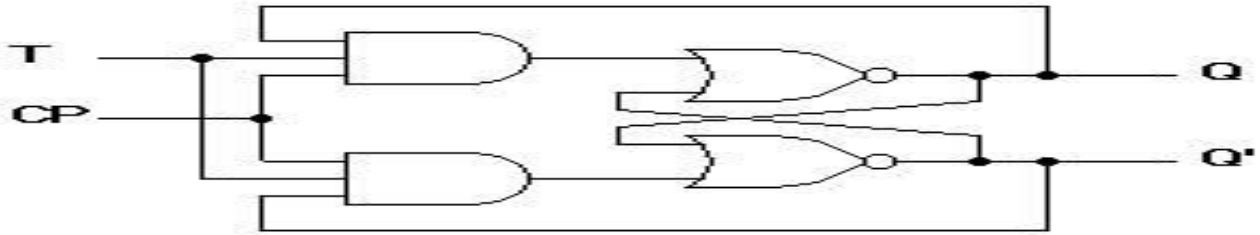
Determine the *Q* output for the *J-K* flip-flop, given the inputs shown.

Notice that the outputs change on the leading edge of the clock.

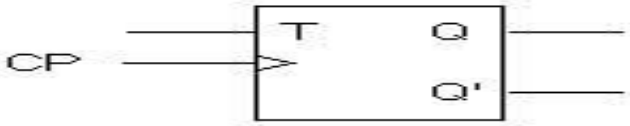
Solution



T Flip-Flop



(a) Logic diagram

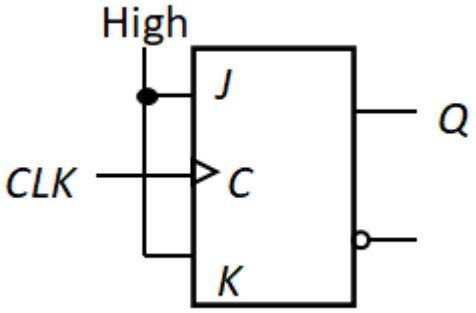


(b) Graphical symbol

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(c) Transition table

Clocked T flip-flop



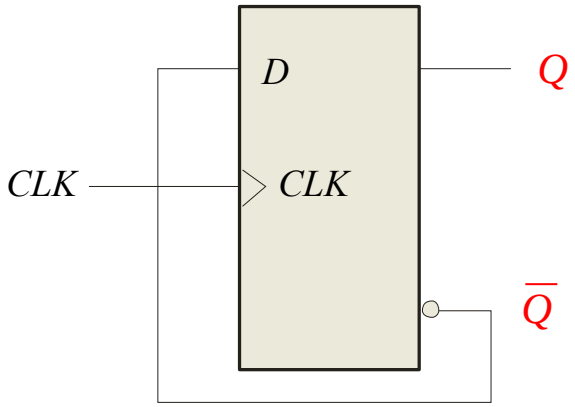
C	T	Q_{next}
0	x	No change
1	0	No change
1	1	$Q'_{current}$

Summary

Flip-flops

A D-flip-flop does not have a toggle mode like the J-K flip-flop, but you can hardwire a toggle mode by connecting $\overline{Q}$ back to *D* as shown. This is useful in some counters.

For example, if *Q* is LOW, $\overline{Q}$ is HIGH and the flip-flop will toggle on the next clock edge. Because the flip-flop only changes on the active edge, the output will only change once for each clock pulse.



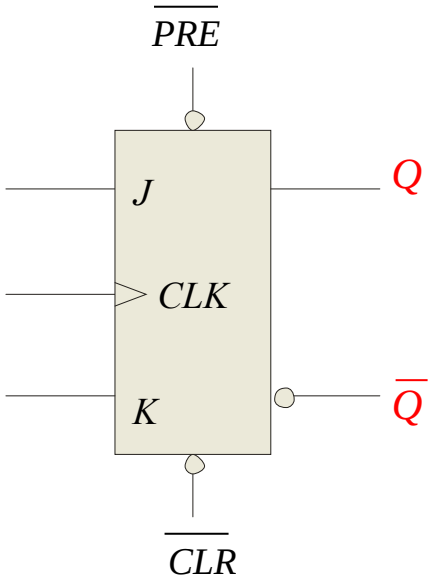
D flip-flop hardwired for a toggle mode

Summary

Flip-flops

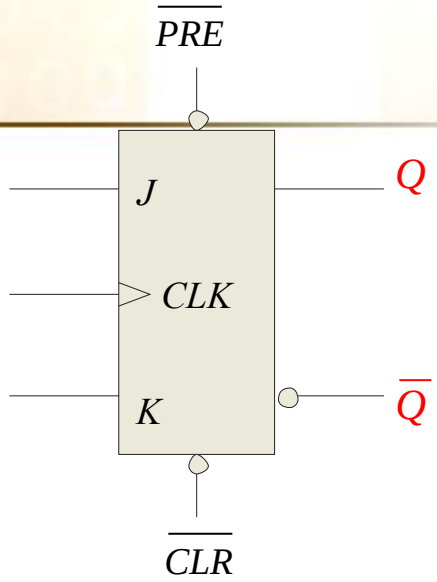
Synchronous inputs are transferred in the triggering edge of the clock (for example the *D* or *J-K* inputs). Most flip-flops have other inputs that are *asynchronous*, meaning they affect the output independent of the clock.

Two such inputs are normally labeled preset (*PRE*) and clear (*CLR*). These inputs are usually active LOW. A J-K flip flop with active LOW preset and CLR is shown.



Summary

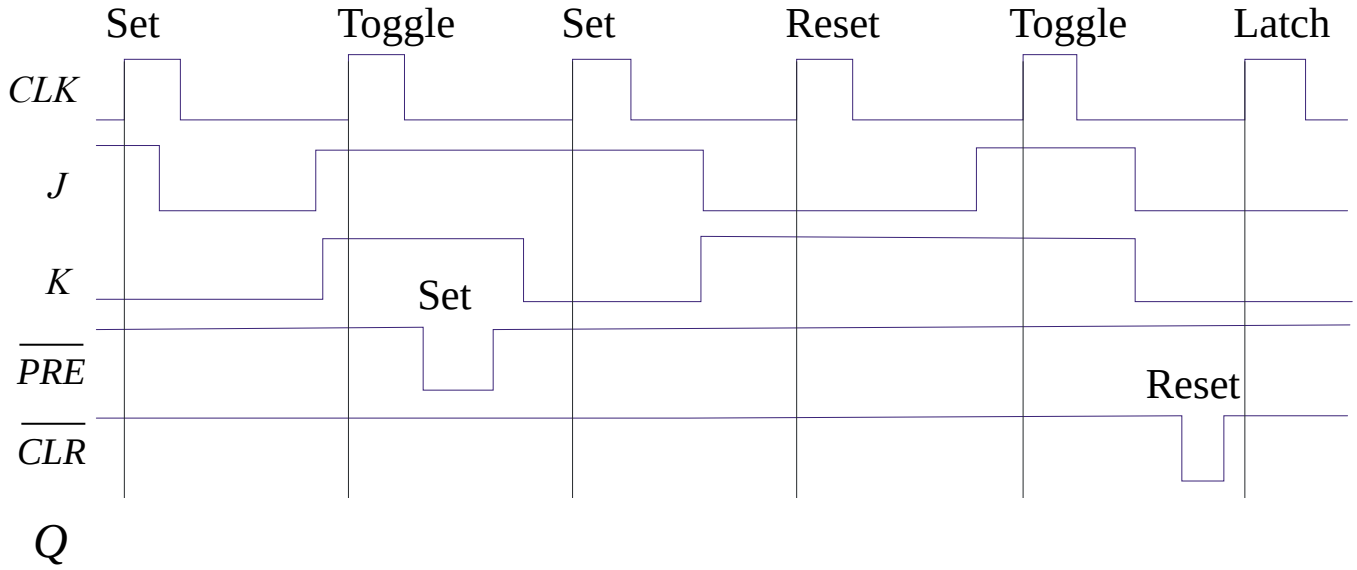
Flip-flops



Example

Determine the *Q* output for the *J-K* flip-flop, given the inputs shown.

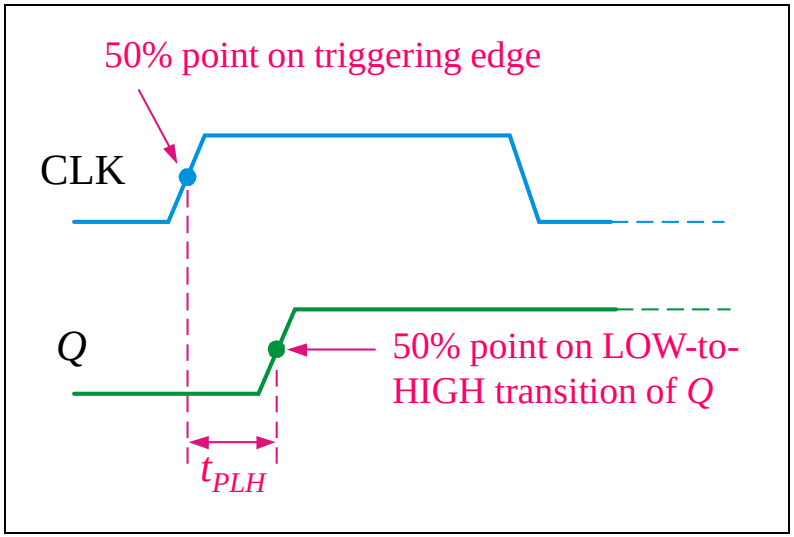
Solution



Summary

Flip-flop Characteristics

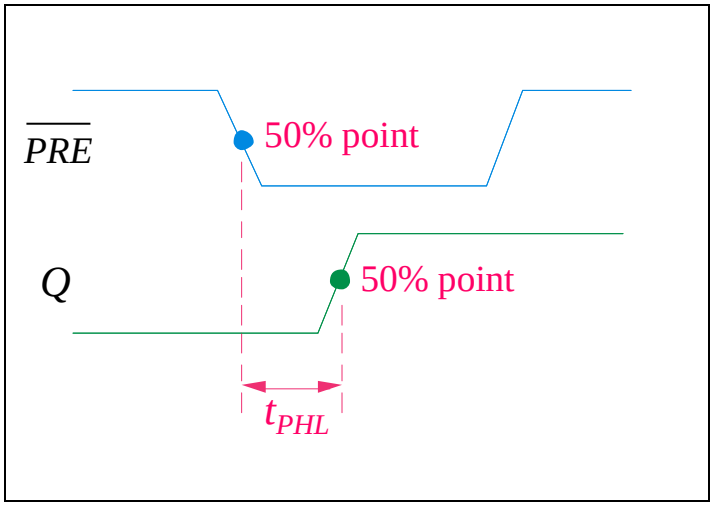
Propagation delay time is specified for the rising and falling outputs. It is measured between the 50% level of the clock to the 50% level of the output transition.



Summary

Flip-flop Characteristics

Another **propagation delay time** specification is the time required for an *asynchronous* input to cause a change in the output. Again it is measured from the 50% levels.

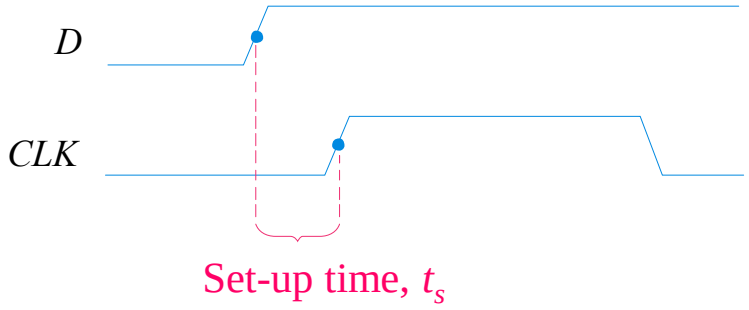


Summary

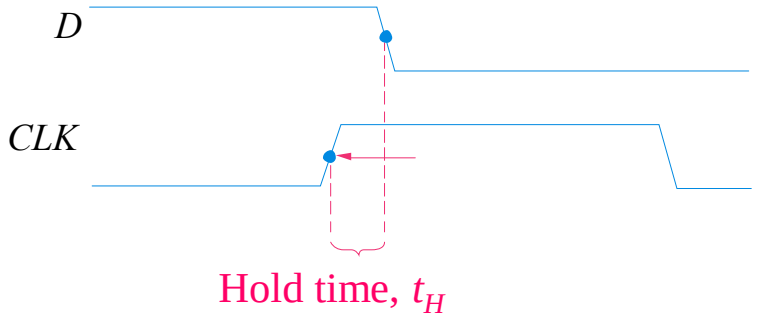
Flip-flop Characteristics

Set-up time and **hold time** are times required before and after the clock transition that data must be present to be reliably clocked into the flip-flop.

Setup time is the minimum time for the data to be present *before* the clock.



Hold time is the minimum time for the data to *remain* after the clock.



Summary

Flip-flop Characteristics

Other specifications include maximum clock frequency, minimum pulse widths for various inputs, and power dissipation. The power dissipation is the product of the supply voltage and the average current required.

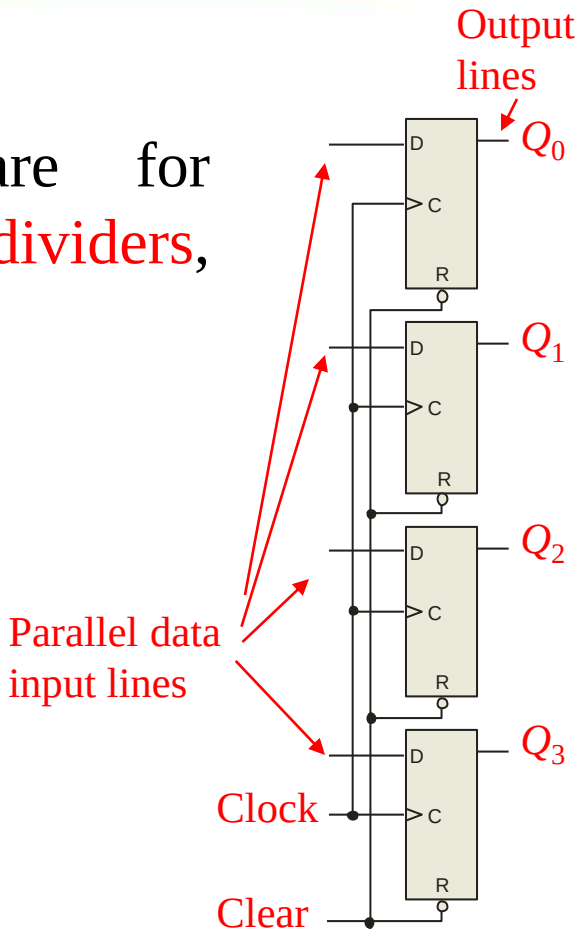
A useful comparison between logic families is the **speed-power product** which uses two of the specifications discussed: the average propagation delay and the average power dissipation. The unit is energy.

Summary

Flip-flop Applications

Principal flip-flop applications are for temporary data storage, as frequency dividers, and in counters

Typically, for **data storage** applications, a group of flip-flops are connected to parallel data lines and clocked together. Data is stored until the next clock pulse.

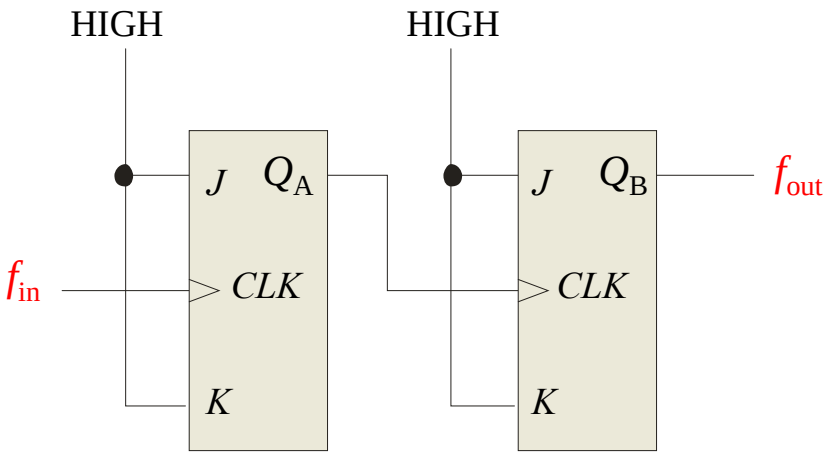


Summary

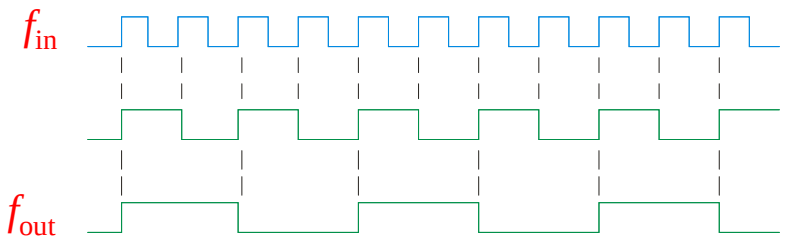
Flip-flop Applications

For **frequency division**, it is simple to use a flip-flop in the toggle mode or to chain a series of toggle flip flops to continue to divide by two.

One flip-flop will divide f_{in} by 2, two flip-flops will divide f_{in} by 4 (and so on).

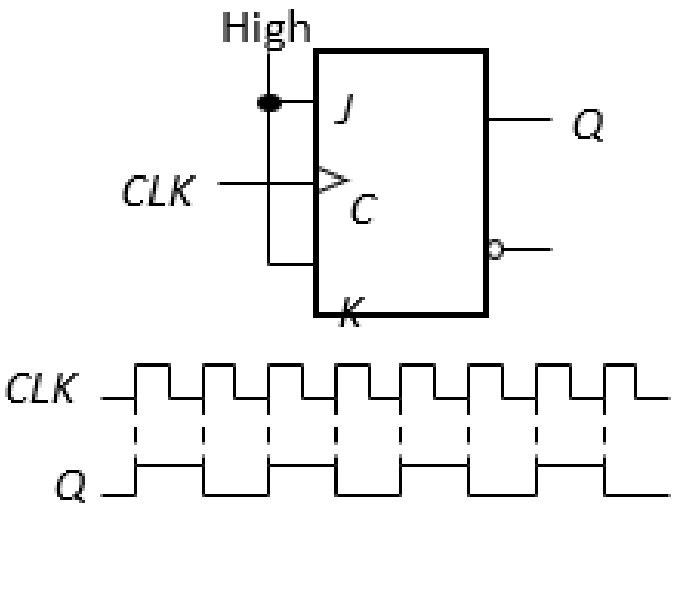


Waveforms:

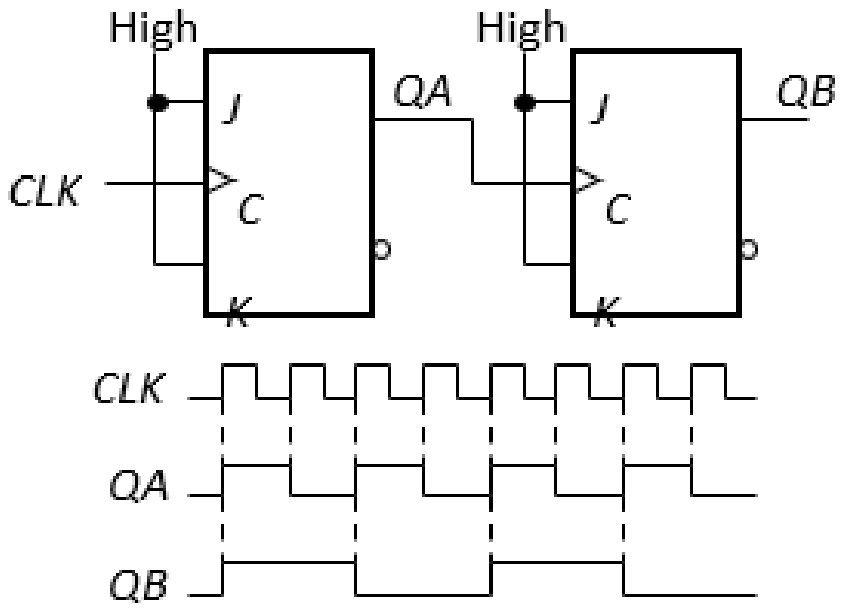


T Flip-flop

- Application: *Frequency division*.



Divide clock frequency by 2.



Divide clock frequency by 4.

- Application: *Counter*

Selected Key Terms

Latch	A bistable digital circuit used for storing a bit.
Bistable	Having two stable states. Latches and flip-flops are bistable multivibrators.
Clock	A triggering input of a flip-flop.
D flip-flop	A type of bistable multivibrator in which the output assumes the state of the *D* input on the triggering edge of a clock pulse.
J-K flip-flop	A type of flip-flop that can operate in the SET, RESET, no-change, and toggle modes.

Selected Key Terms

Propagation delay time	The interval of time required after an input signal has been applied for the resulting output signal to change.
Set-up time	The time interval required for the input levels to be on a digital circuit.
Hold time	The time interval required for the input levels to remain steady to a flip-flop after the triggering edge in order to reliably activate the device.

Summary

Module II (10 Hours)

Sequential circuit - Clocking, Flip flops - SR – JK- D -T flip flops, Counters - Synchronous and asynchronous counters - UP/DOWN counters , Registers - Serial in serial out - Serial in parallel out - Parallel in serial out - Parallel in parallel out registers

Counting in Binary

As you know, the binary count sequence follows a familiar pattern of 0's and 1's

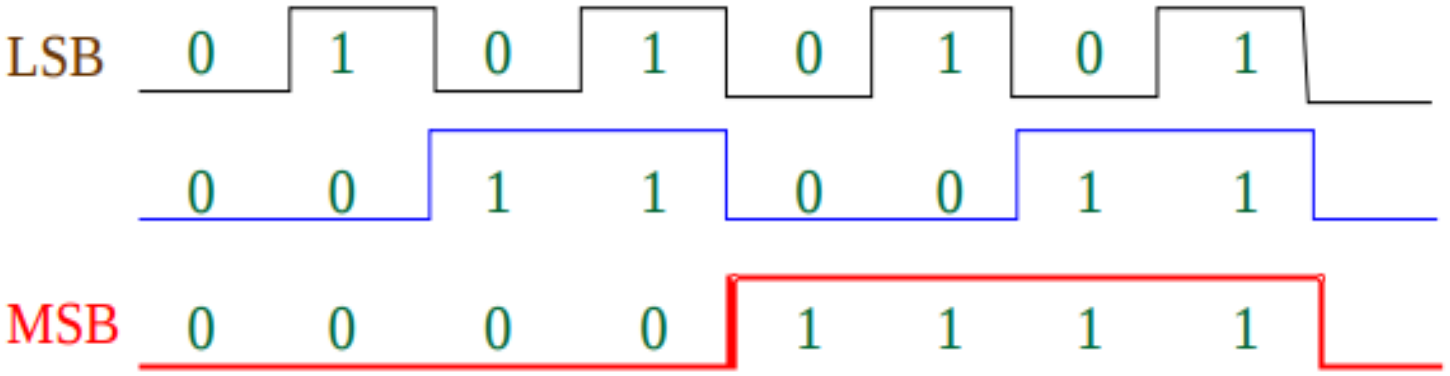
LSB changes on every number.

The next bit changes on every 2 ND number.

The next bit changes on every fourth number.

Counting in Binary

A counter can form the same pattern of 0's and 1's with logic levels. The first stage in the counter represents the least significant bit – notice that these waveforms follow the same pattern as counting in binary.



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.(do not have common clock pulse).
- Synchronous counters apply the same clock to all flip-flops.

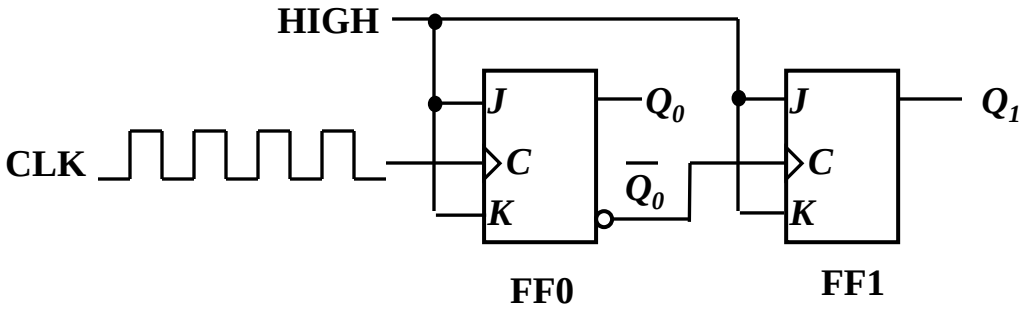
Asynchronous (Ripple) Counters

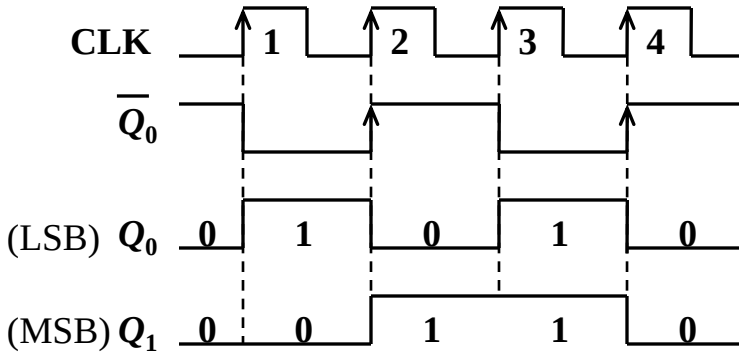
- Asynchronous counters: the flip-flops do not change states at exactly the same time as they do not have a common clock pulse.
- Also known as ripple counters, as the input clock pulse "ripples" through the counter – cumulative delay is a drawback.
- *n* flip-flops $\rightarrow$ a MOD (modulus) 2^n counter.

- 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(asynchronous) binary counter
- Clock is connected to the clock input of only the first flipflop.
- Output of one flip-flop is connected to the clock input of the next more-significant flip-flop.



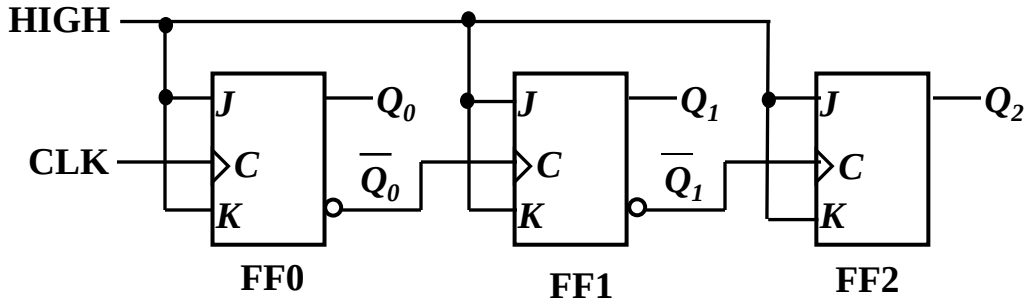


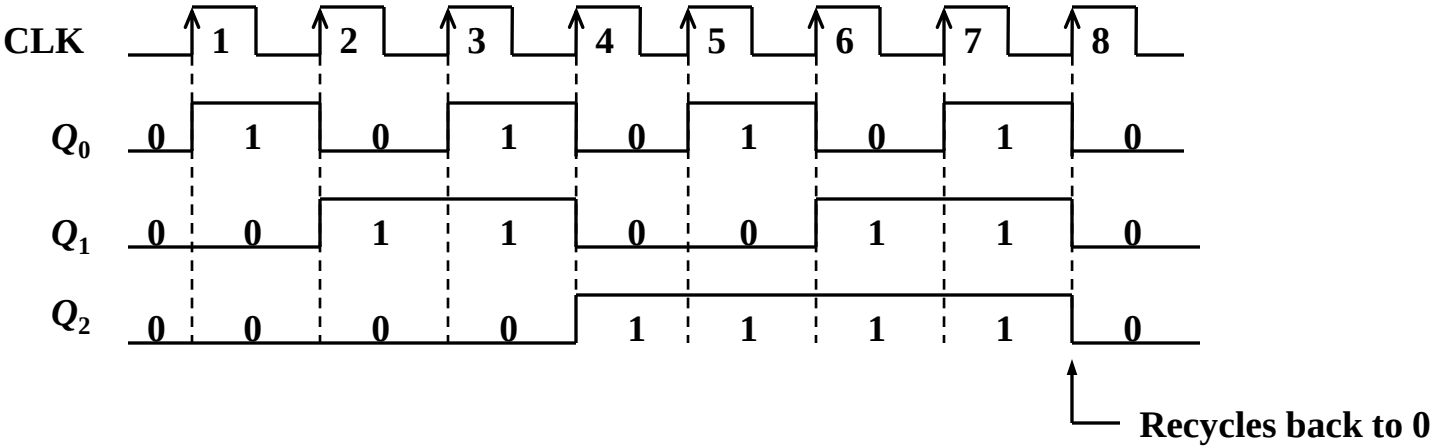
Timing diagram

$00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ ...

Asynchronous (Ripple) Counters

- Example: 3-bit ripple binary counter

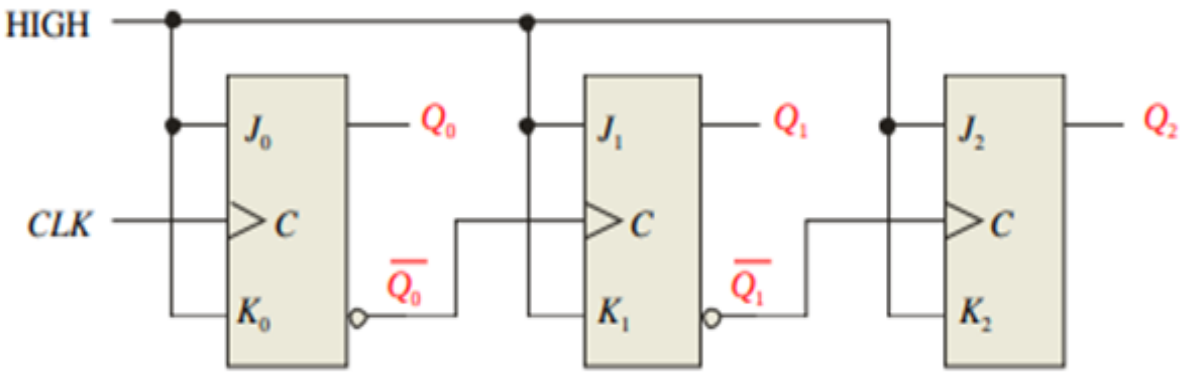




Three bit Asynchronous Counter

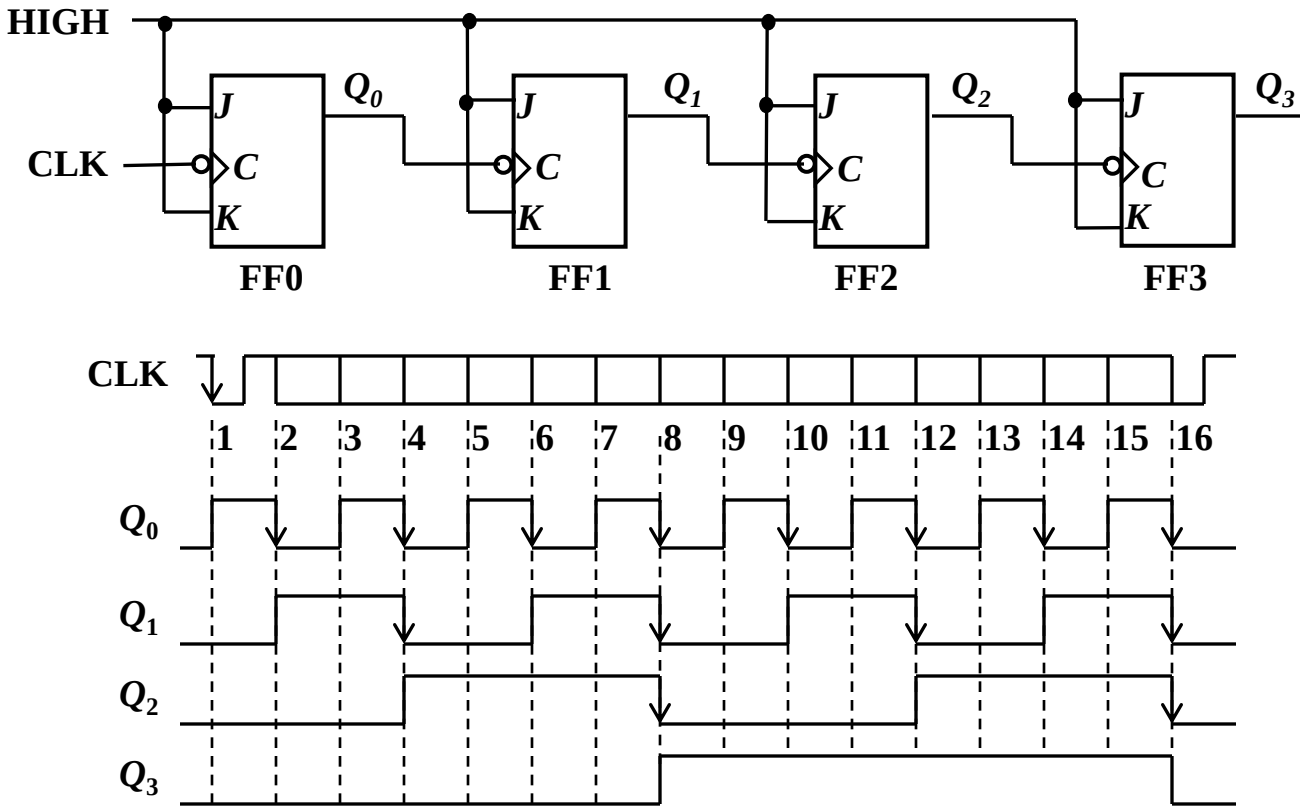
In an asynchronous counter, the clock is applied only to the first stage. Subsequent stages derive the clock from the previous stage.

The three-bit asynchronous counter shown is typical. It uses J-K flip-flops in the toggle mode.



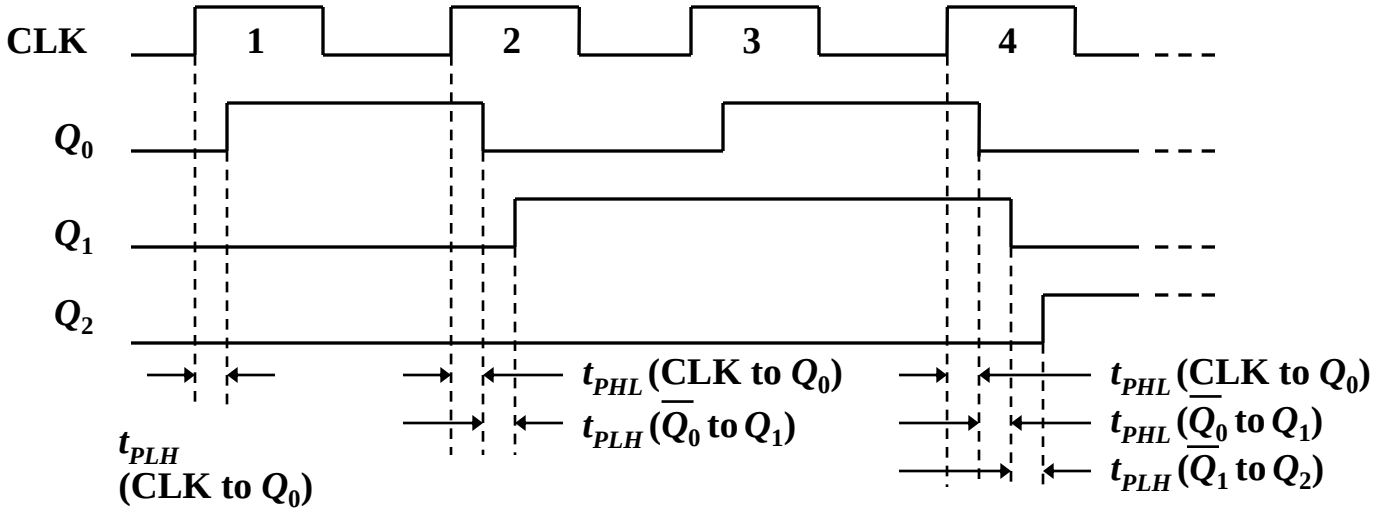
Asynchronous (Ripple) Counters

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



Asynchronous (Ripple) Counters

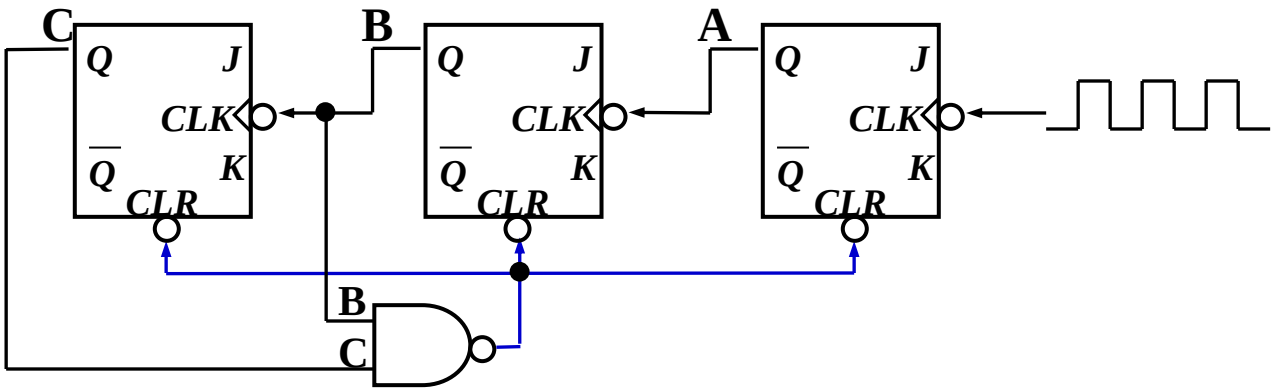
- Propagation delays in an asynchronous (ripple-clocked) binary counter.
- If the accumulated delay is greater than the clock pulse, some counter states may be misrepresented!



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.)

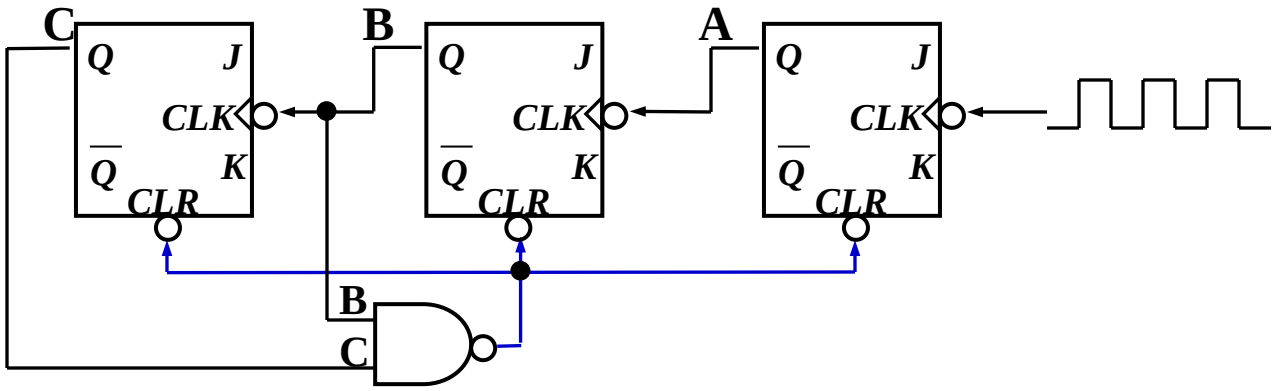
All *J*, *K* inputs are 1 (HIGH).

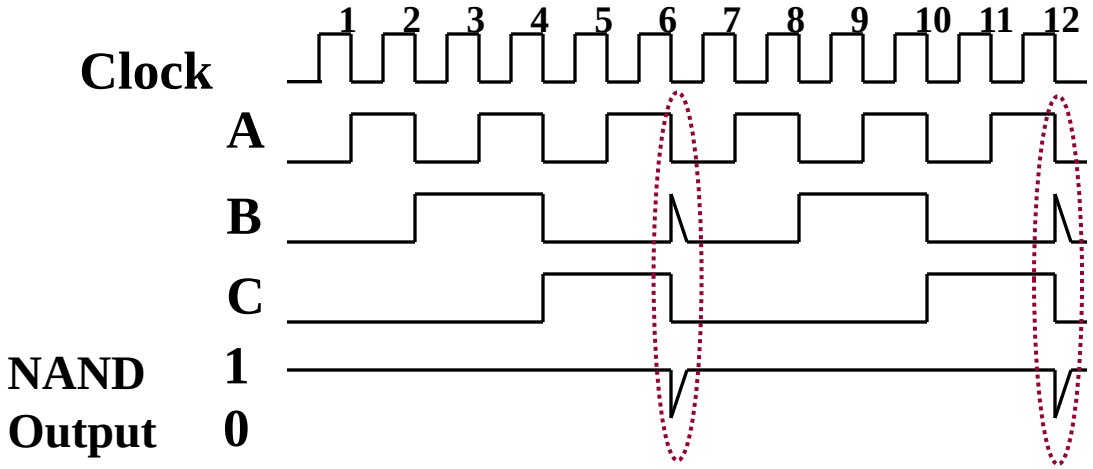


Asyn. Counters with MOD no. < 2^n

- Example (cont'd):

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

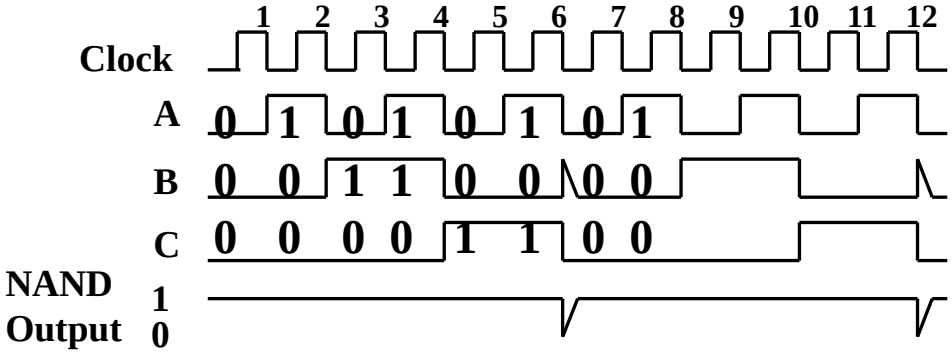


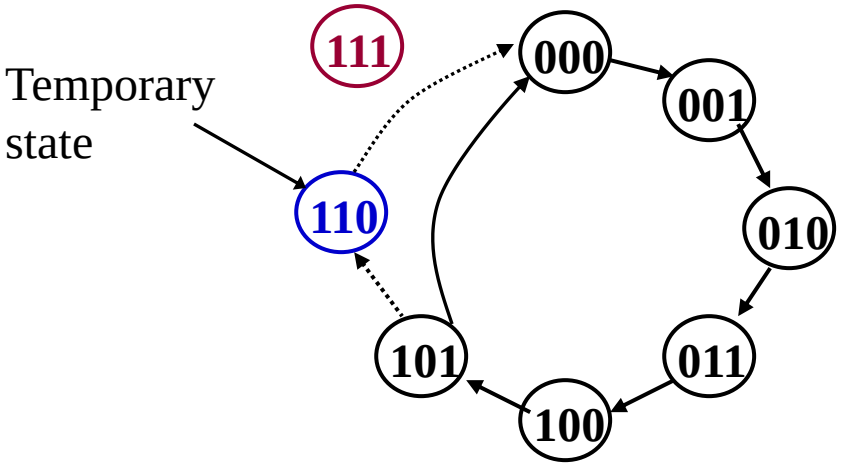


MOD-6 counter produced by clearing (a MOD-8 binary counter) when count of six (110) occurs.

Asyn. Counters with MOD no. < 2^n

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

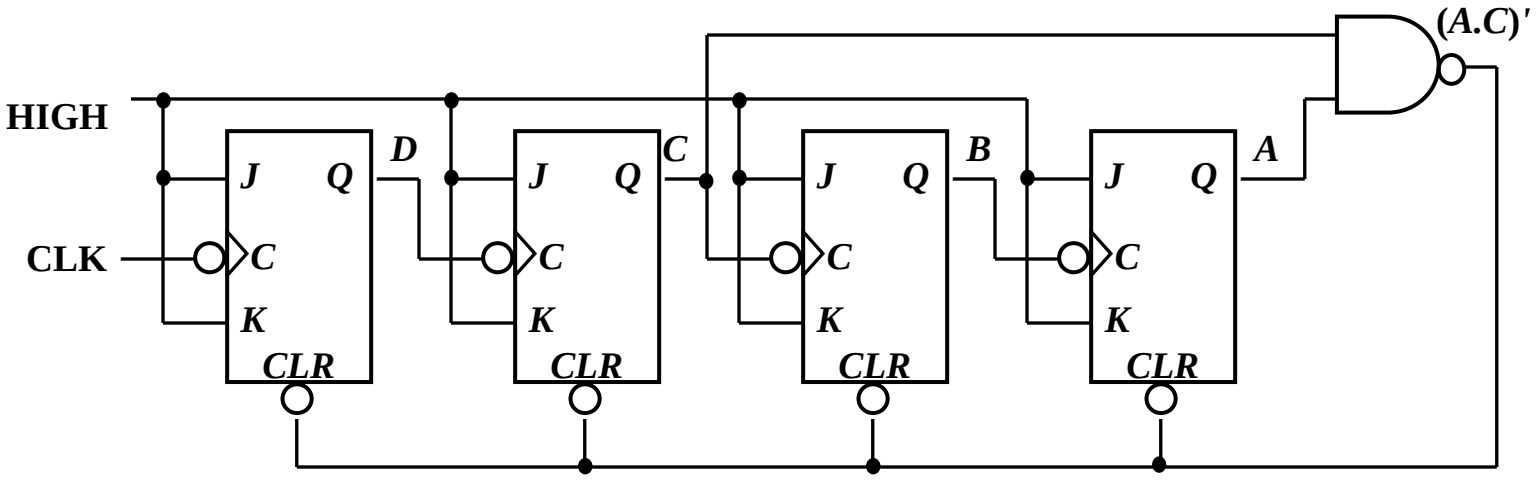




Counter is a MOD-6 counter.

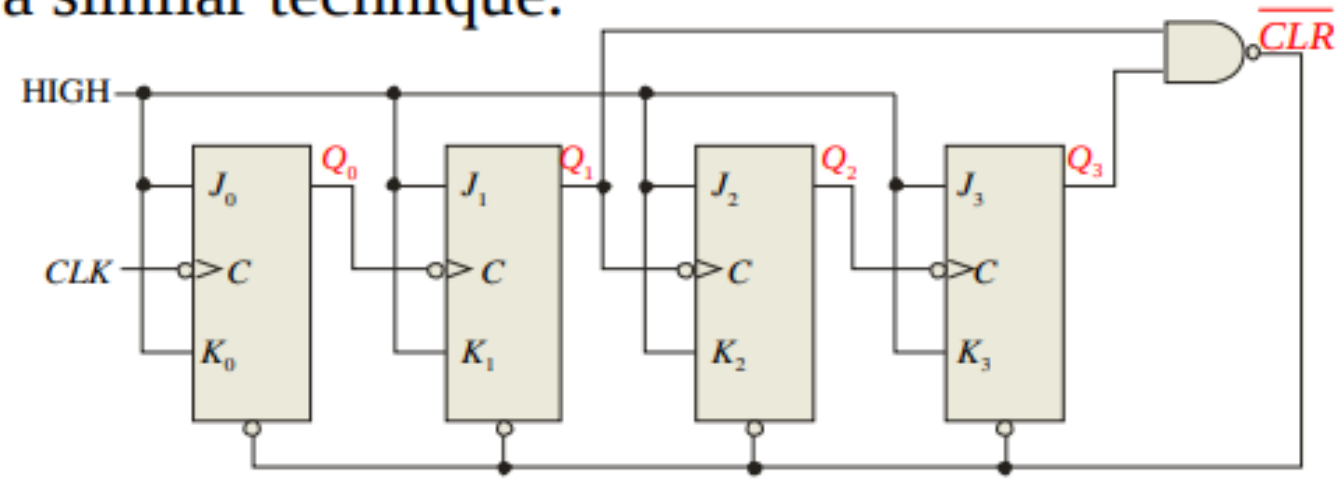
Asyn. Counters with MOD no. < 2^n

- 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.



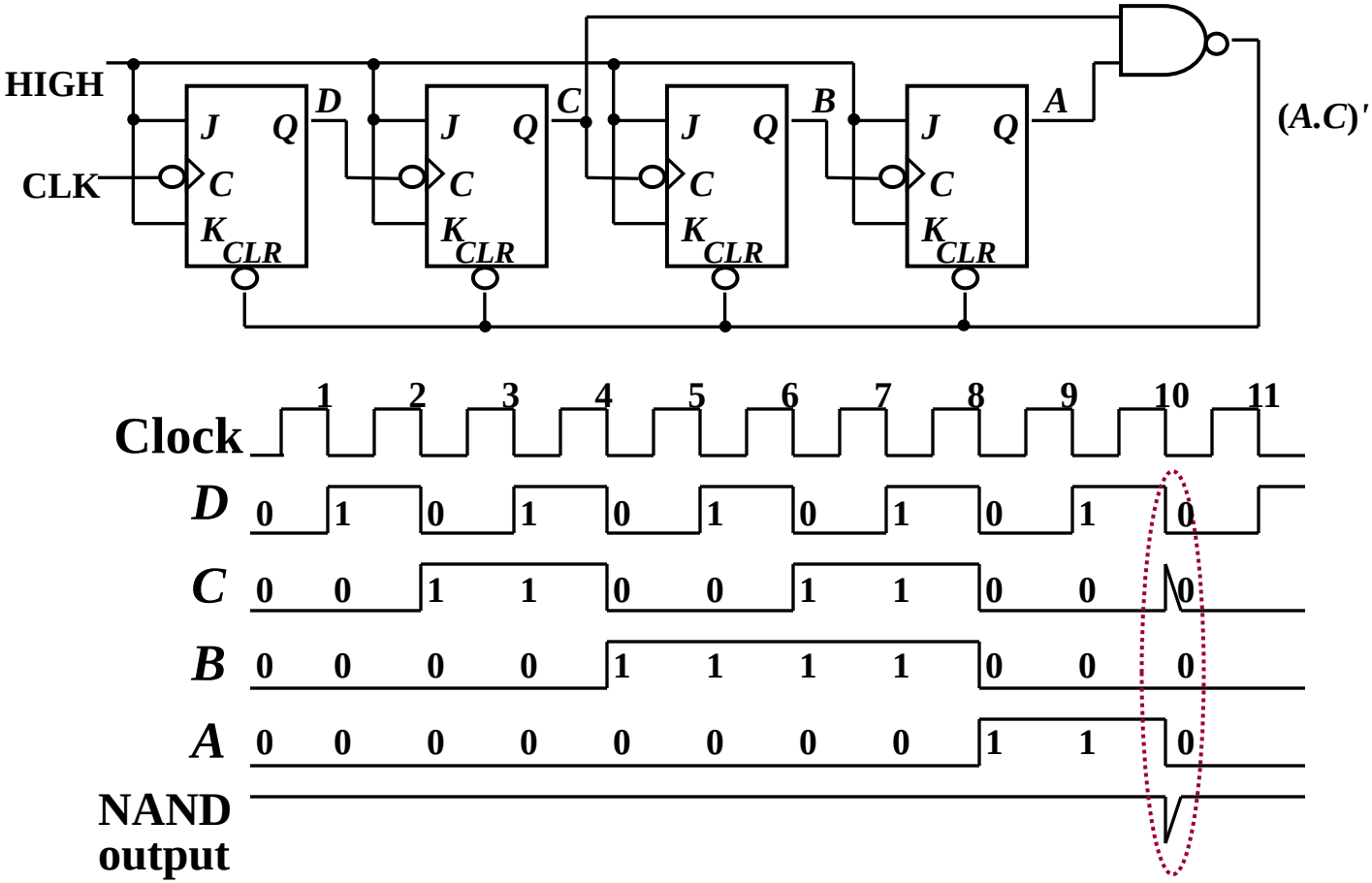
Asynchronous Decade Counter

This counter uses partial decoding to recycle the count sequence to zero after the 1001 state. The flip-flops are trailing-edge triggered, so clocks are derived from the *Q* outputs. Other truncated sequences can be obtained using a similar technique.



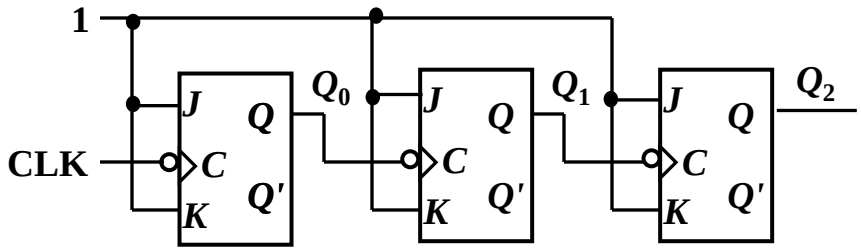
Asyn. Counters with MOD no. < 2^n

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

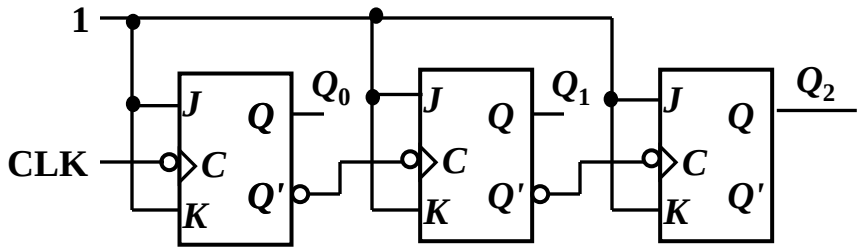


Asynchronous Down Counters

- 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.



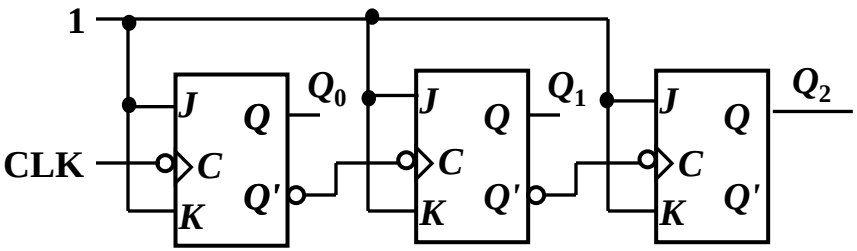
3-bit binary
up counter

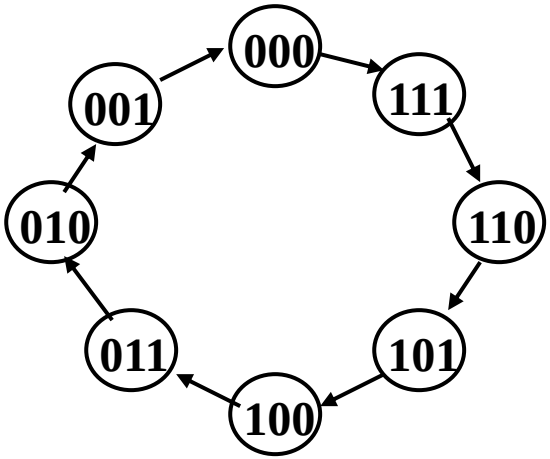


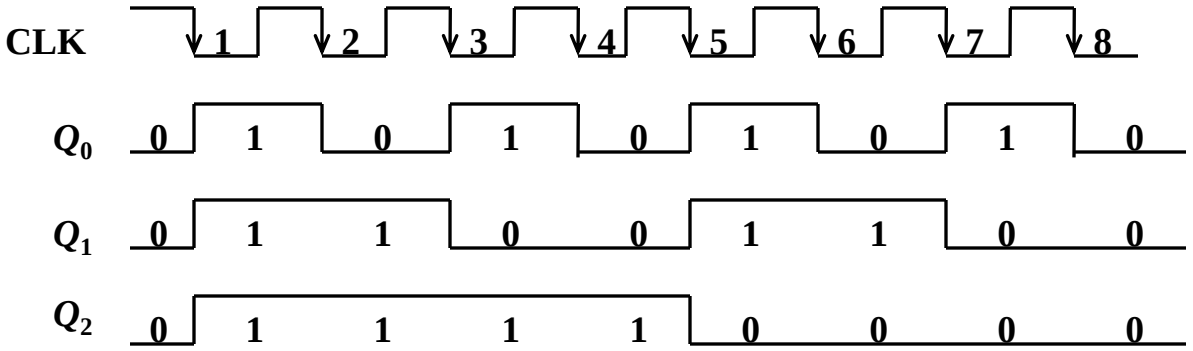
3-bit binary
down counter

Asynchronous Down Counters

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

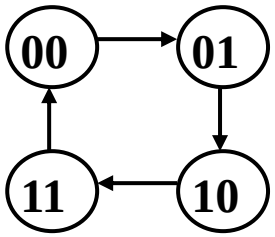






Synchronous (Parallel) Counters

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



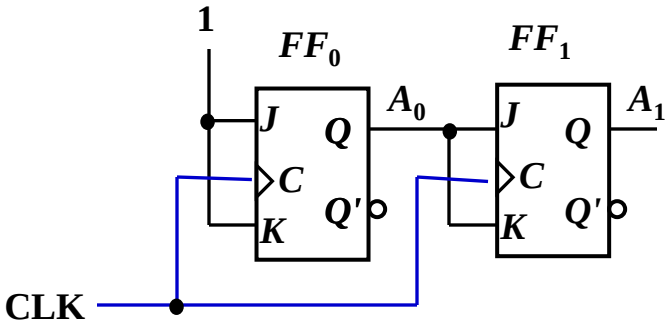
Present state		Next state		Flip-flop inputs	
A_1	A_0	A_1^+	A_0^+	TA_1	TA_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

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 Counters

In a **synchronous counter** all flip-flops are clocked together with a common clock pulse. Synchronous counters overcome the disadvantage of accumulated propagation delays, but generally they require more circuitry to control states changes.

Synchronous (Parallel) Counters

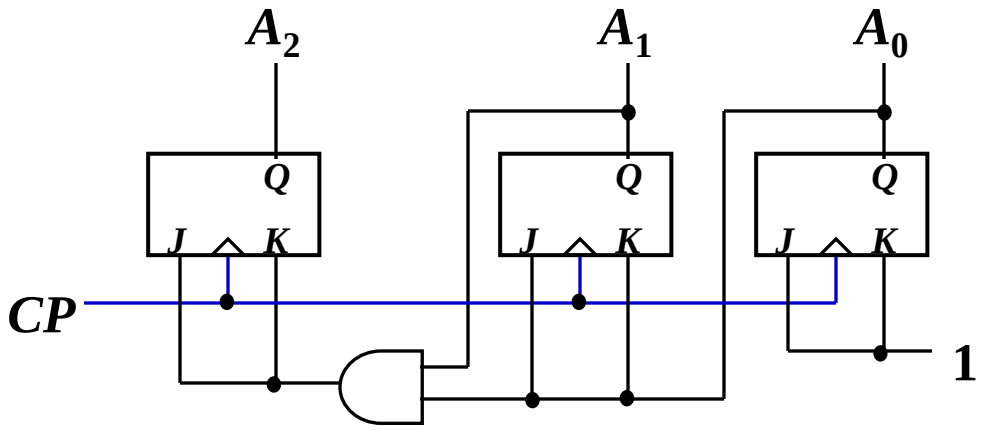
- Example: 3-bit synchronous binary counter (using T flip-flops, or JK flip-flops with identical J, K inputs).

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).

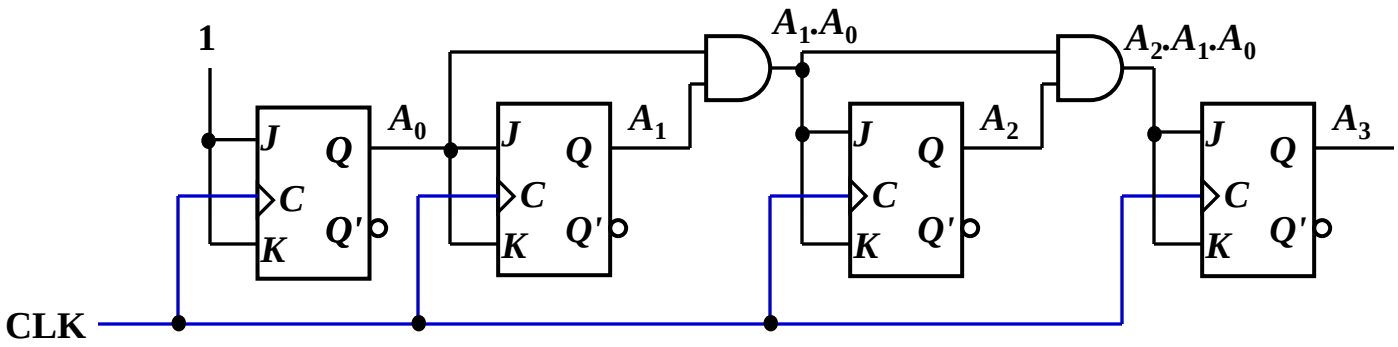
$TA_2 = A_1.A_0$ $TA_1 = A_0$ $TA_0 = 1$



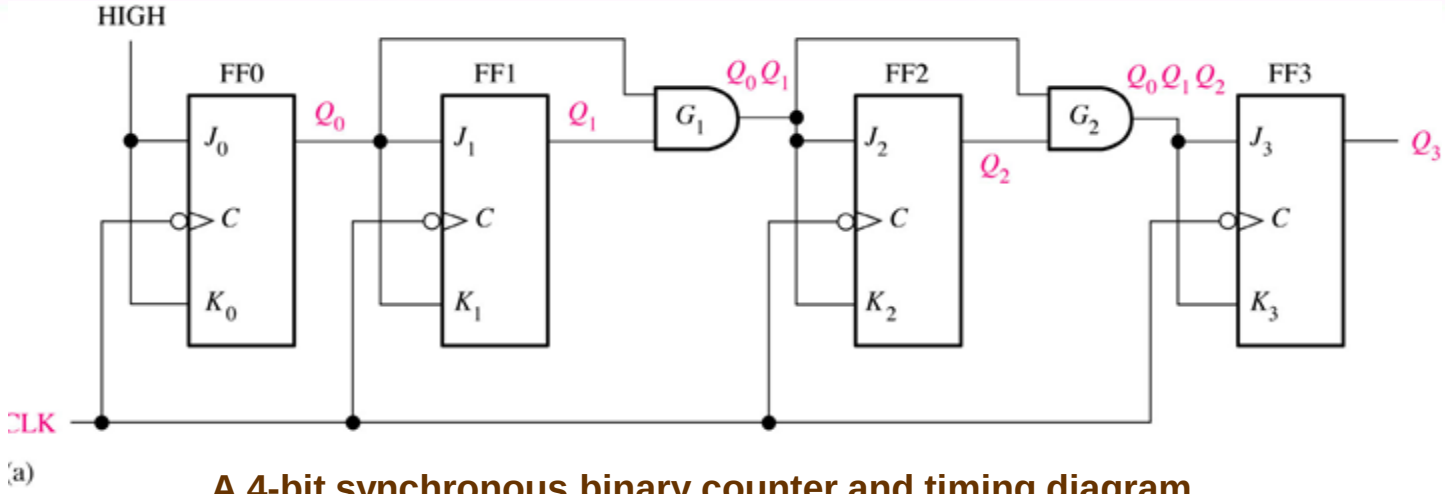
Synchronous (Parallel) Counters

- Example: 4-bit synchronous binary counter

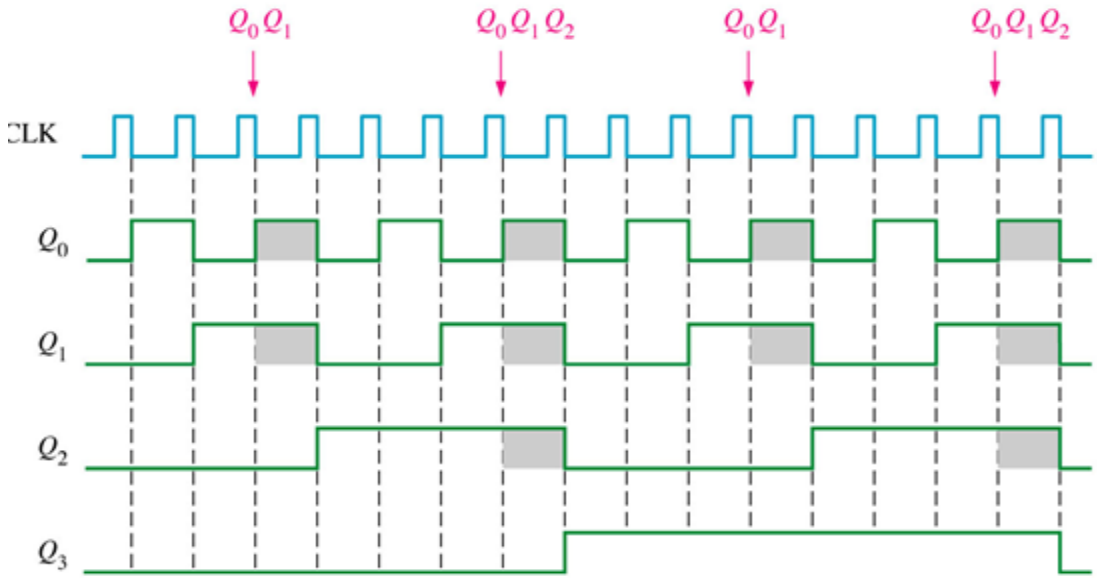
$$TA_3 = A_2 \cdot A_1 \cdot A_0$$
$$TA_2 = A_1 \cdot A_0$$
$$TA_1 = A_0$$
$$TA_0 = 1$$



Synchronous binary Counter



A 4-bit synchronous binary counter and timing diagram.



CLK PLUSE	Q3	Q2	Q1	Q0
0	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	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16 REPEAT	0	0	0	0

Synchronous (Parallel) Counters

- Example: Synchronous decade/BCD counter

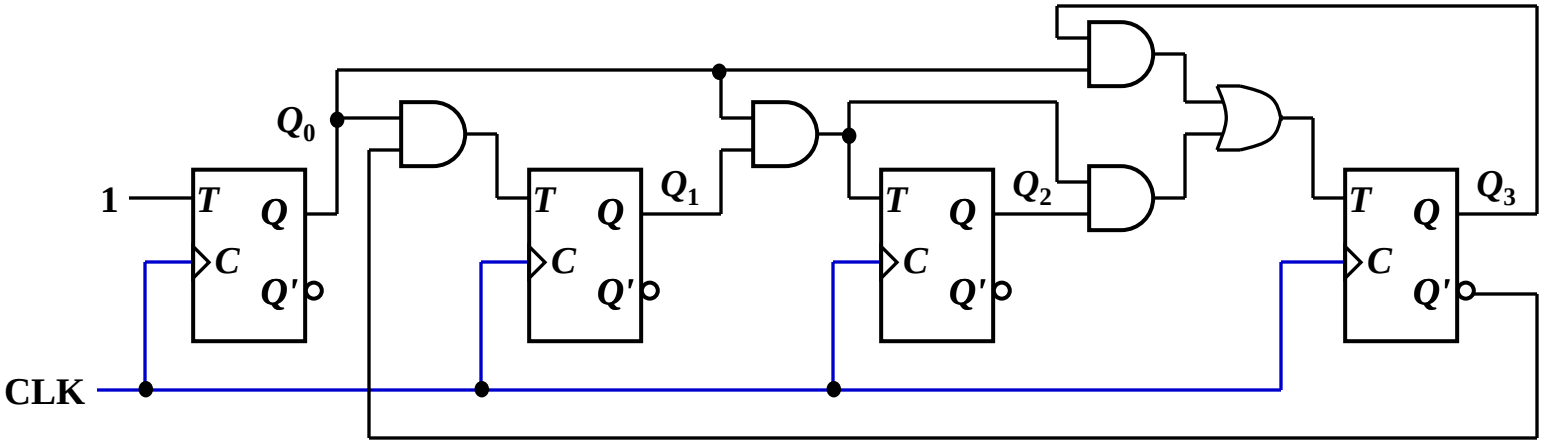
Clock pulse	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 (recycle)	0	0	0	0

$$T_0 = 1$$
$$T_1 = Q_3'.Q_0$$
$$T_2 = Q_1.Q_0$$
$$T_3 = Q_2.Q_1.Q_0 + Q_3.Q_0$$

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 *Up*/$\overline{Down}$ (or simply *Up*) specifies the direction of counting.
  - *Up*/$\overline{Down}$ = 1 $\rightarrow$ Count upward
  - *Up*/$\overline{Down}$ = 0 $\rightarrow$ 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	

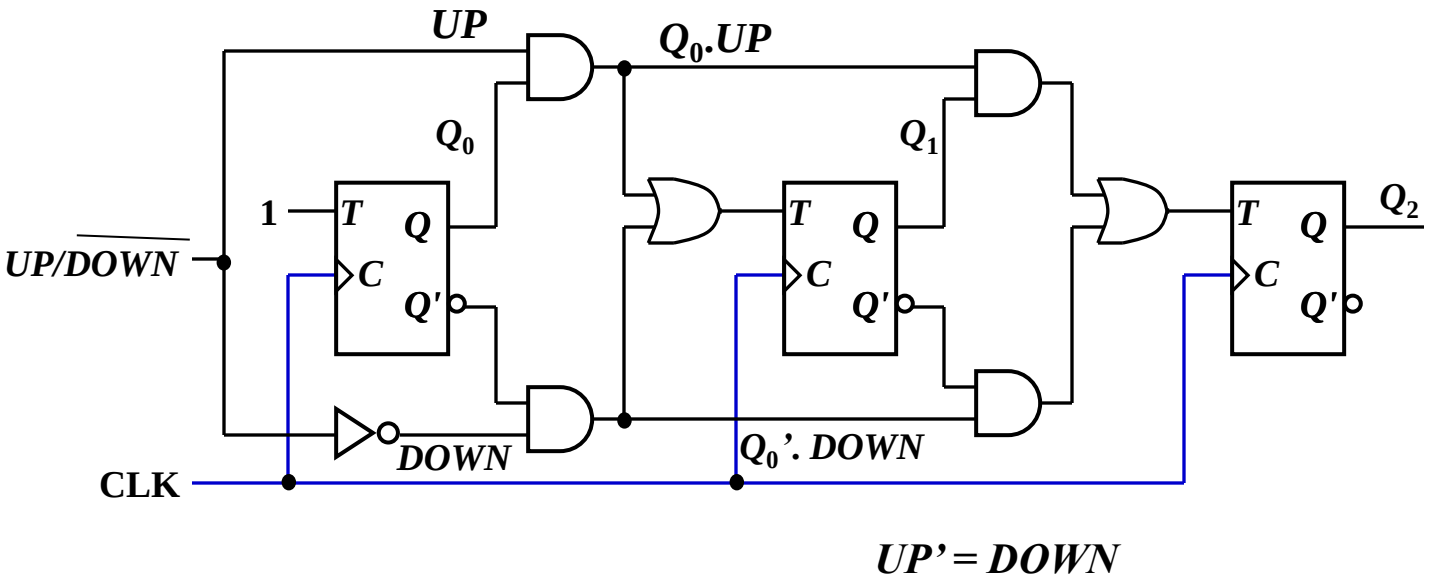
$$TQ_0 = 1$$
$$TQ_1 = (Q_0.Up) + (Q_0'.Up')$$
$$TQ_2 = (Q_0.Q_1.Up) + (Q_0'.Q_1'.Up')$$

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)

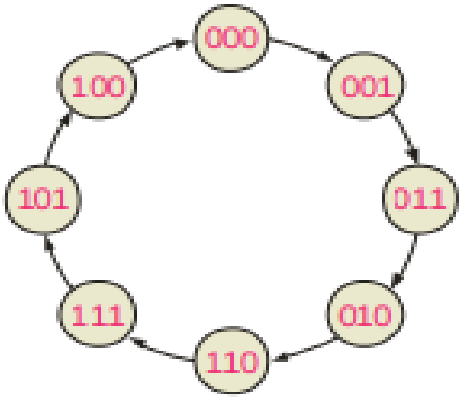
$$TQ_0 = 1$$
$$TQ_1 = (Q_0.Up) + (Q_0'.Up')$$
$$TQ_2 = (Q_0.Q_1.Up) + (Q_0'.Q_1'.Up')$$



Designing Synchronous Counters

- Example: A 3-bit Gray code counter (using JK flip-flops)

1. State diagram:



2. Next state table:

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

Designing Synchronous Counters

- Example: A 3-bit Gray code counter (using JK flip-flops)

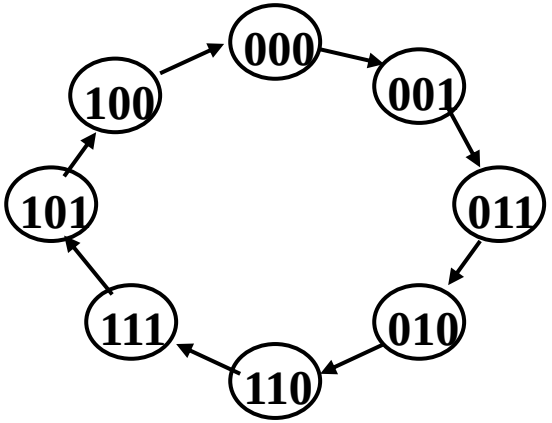
3. FLIP-FLOP TRANSITION TABLE

Output Transitions Q_N	Q_{N+1}	Flip-Flop Inputs J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Designing Synchronous Counters

- Example: A 3-bit Gray code counter (using JK flip-flops)

Output Transitions Q_N	Q_{N+1}	Flip-Flop Inputs J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0



	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	0	1	0	X	0	X	1	X
1	0	0	1	0	1	1	0	X	1	X	X	0
3	0	1	1	0	1	0	0	X	X	0	X	1
2	0	1	0	1	1	0	1	X	X	0	0	X
6	1	1	0	1	1	1	X	0	X	0	1	X
7	1	1	1	1	0	1	X	0	X	1	X	0
5	1	0	1	1	0	0	X	0	0	X	X	1
4	1	0	0	0	0	0	X	1	0	X	0	X

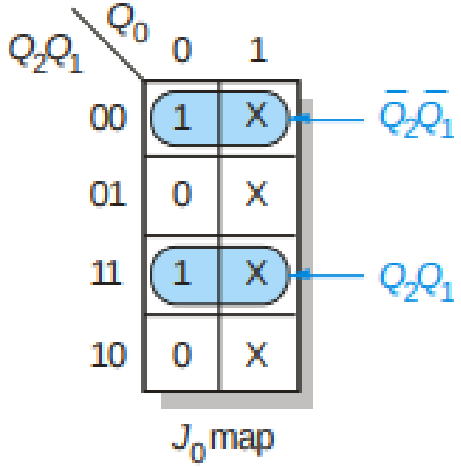
Designing Synchronous Counters

- Example: A 3-bit Gray code counter (using JK flip-flops)

4.KARNAUGH MAPS

Each time a flip-flop is clocked, the J and K inputs required for that transition are mapped onto a K-map.

An example of the J_0 map is:

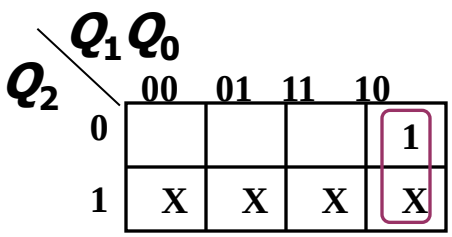


Designing Synchronous Counters

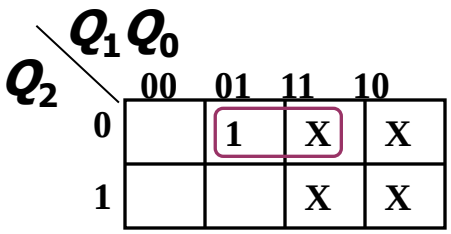
- 3-bit Gray code counter: flip-flop inputs.

	JQ_2	KQ_2	JQ_1	KQ_1	JQ_0	KQ_0
0	0	X	0	X	1	X
1	0	X	1	X	X	0
3	0	X	X	0	X	1
2	1	X	X	0	0	X
6	X	0	X	0	1	X
7	X	0	X	1	X	0
5	X	0	0	X	X	1
4	X	1	0	X	0	X

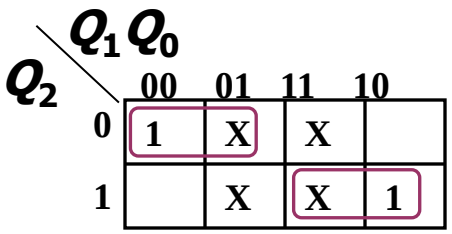
4. KARNAUGH MAPS



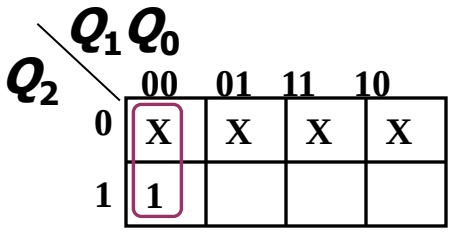
$JQ_2 = Q_1.Q_0'$



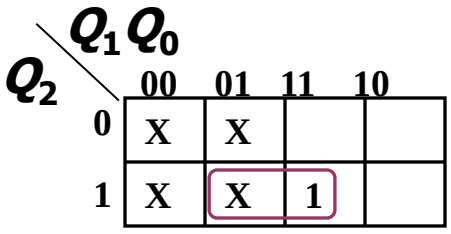
$JQ_1 = Q_2'.Q_0$



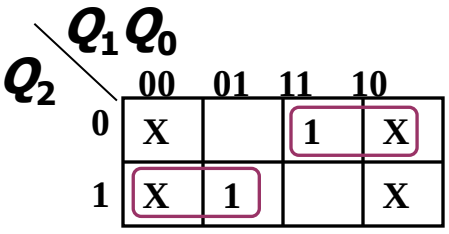
$JQ_0 = Q_2.Q_1 + Q_2'.Q_1'$
$= (Q_2 \oplus Q_1)'$



$KQ_2 = Q_1'.Q_0'$



$KQ_1 = Q_2.Q_0$



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

Designing Synchronous Counters

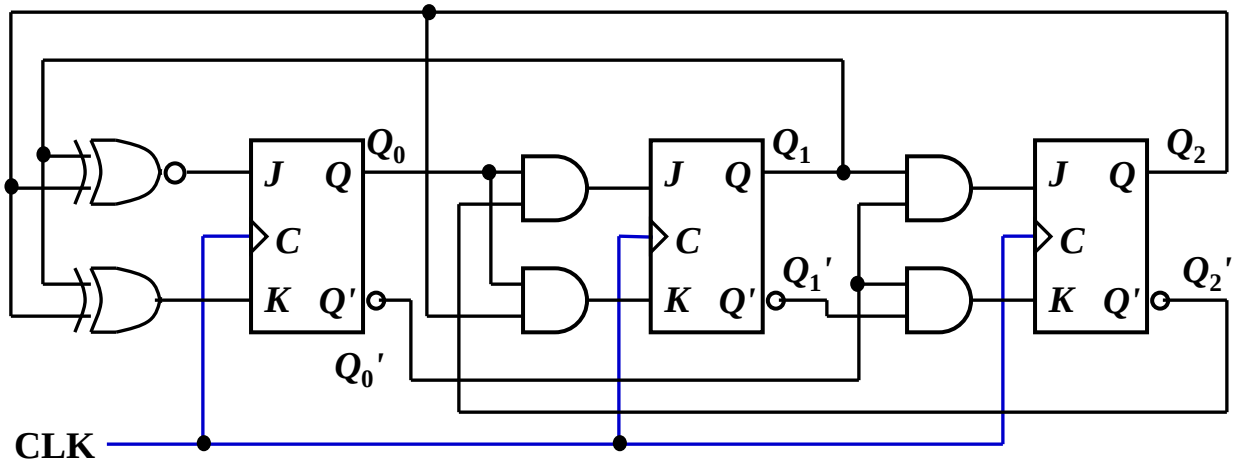
- 3-bit Gray code counter: logic diagram.

5. Logic Expressions for Flip-Flop Inputs

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

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

6. Counter Implementation



Synchronous Binary Counters

J-K Flip Flop Design of a 4-bit Binary Up Counter

TABLE 4-10
Flip-Flop Excitation Tables

(a) *JK* Flip-Flop

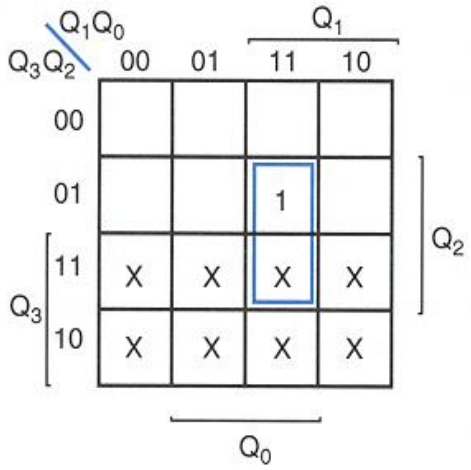
$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Present state				Next state				Flip-flop inputs							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q3}	K_{Q3}	J_{Q2}	K_{Q2}	J_{Q1}	K_{Q1}	J_{Q0}	K_{Q0}
0	0	0	0	0	0	0	1	0	×	0	×	0	×	1	×
0	0	0	1	0	0	1	0	0	×	0	×	1	×	×	1
0	0	1	0	0	0	1	1	0	×	0	×	×	0	1	×
0	0	1	1	0	1	0	0	0	×	1	×	×	1	×	1
0	1	0	0	0	1	0	1	0	×	×	0	0	×	1	×
0	1	0	1	0	1	1	0	0	×	×	0	1	×	×	1
0	1	1	0	0	1	1	1	0	×	×	0	×	0	1	×
0	1	1	1	1	0	0	0	1	×	×	1	×	1	×	1
1	0	0	0	1	0	0	1	×	0	0	×	0	×	1	×
1	0	0	1	1	0	1	0	×	0	0	×	1	×	×	1
1	0	1	0	1	0	1	1	×	0	0	×	×	0	1	×
1	0	1	1	1	1	0	0	×	0	1	×	×	1	×	1
1	1	0	0	1	1	0	1	×	0	×	0	0	×	1	×
1	1	0	1	1	1	1	0	×	0	×	0	1	×	×	1
1	1	1	0	1	1	1	1	×	0	×	0	×	0	1	×
1	1	1	1	0	0	0	0	×	1	×	1	×	1	×	1

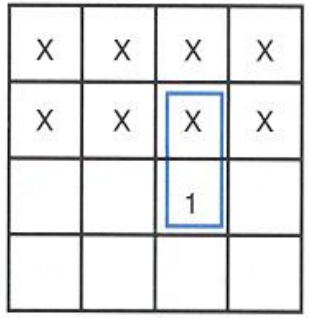
Synchronous Binary Counters

J-K Flip Flop Design of a Binary Up Counter (cont.)

Present state				Next state					
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q3}	K_{Q3}
0	0	0	0	0	0	0	1	0	X
0	0	0	1	0	0	1	0	0	X
0	0	1	0	0	0	1	1	0	X
0	0	1	1	0	1	0	0	0	X
0	1	0	0	0	1	0	1	0	X
0	1	0	1	0	1	1	0	0	X
0	1	1	0	0	1	1	1	0	X
0	1	1	1	1	0	0	0	1	X
1	0	0	0	1	0	0	1	X	0
1	0	0	1	1	0	1	0	X	0
1	0	1	0	1	0	1	1	X	0
1	0	1	1	1	1	0	0	X	0
1	1	0	0	1	1	0	1	X	0
1	1	0	1	1	1	1	0	X	0
1	1	1	0	1	1	1	1	X	0
1	1	1	1	0	0	0	0	X	1



$J_{Q3} = Q_0 Q_1 Q_2$

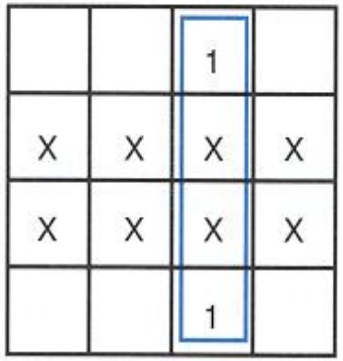


$K_{Q3} = Q_0 Q_1 Q_2$

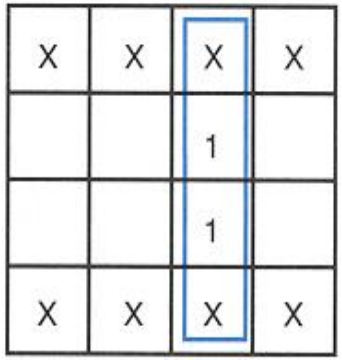
Synchronous Binary Counters

J-K Flip Flop Design of a Binary Up Counter (cont.)

Present state				Next state				Flip-flo	
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q2}	K_{Q2}
0	0	0	0	0	0	0	1	0	×
0	0	0	1	0	0	1	0	0	×
0	0	1	0	0	0	1	1	0	×
0	0	1	1	0	1	0	0	1	×
0	1	0	0	0	1	0	1	×	0
0	1	0	1	0	1	1	0	×	0
0	1	1	0	0	1	1	1	×	0
0	1	1	1	1	0	0	0	×	1
1	0	0	0	1	0	0	1	0	×
1	0	0	1	1	0	1	0	0	×
1	0	1	0	1	0	1	1	0	×
1	0	1	1	1	1	0	0	1	×
1	1	0	0	1	1	0	1	×	0
1	1	0	1	1	1	1	0	×	0
1	1	1	0	1	1	1	1	×	0
1	1	1	1	0	0	0	0	×	1



$J_{Q2} = Q_0Q_1$

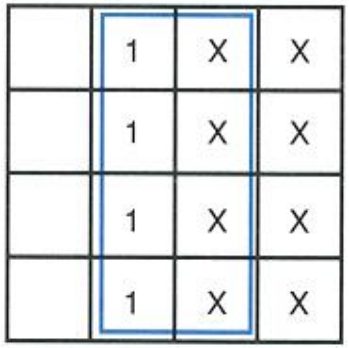


$K_{Q2} = Q_0Q_1$

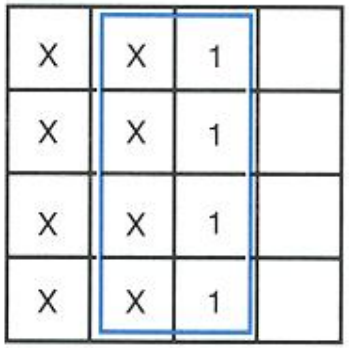
Synchronous Binary Counters

J-K Flip Flop Design of a Binary Up Counter (cont.)

Present state				Next state				p inputs	
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q1}	K_{Q1}
0	0	0	0	0	0	0	1	0	X
0	0	0	1	0	0	1	0	1	X
0	0	1	0	0	0	1	1	X	0
0	0	1	1	0	1	0	0	X	1
0	1	0	0	0	1	0	1	0	X
0	1	0	1	0	1	1	0	1	X
0	1	1	0	0	1	1	1	X	0
0	1	1	1	1	0	0	0	X	1
1	0	0	0	1	0	0	1	0	X
1	0	0	1	1	0	1	0	1	X
1	0	1	0	1	0	1	1	X	0
1	0	1	1	1	1	0	0	X	1
1	1	0	0	1	1	0	1	0	X
1	1	0	1	1	1	1	0	1	X
1	1	1	0	1	1	1	1	X	0
1	1	1	1	0	0	0	0	X	1



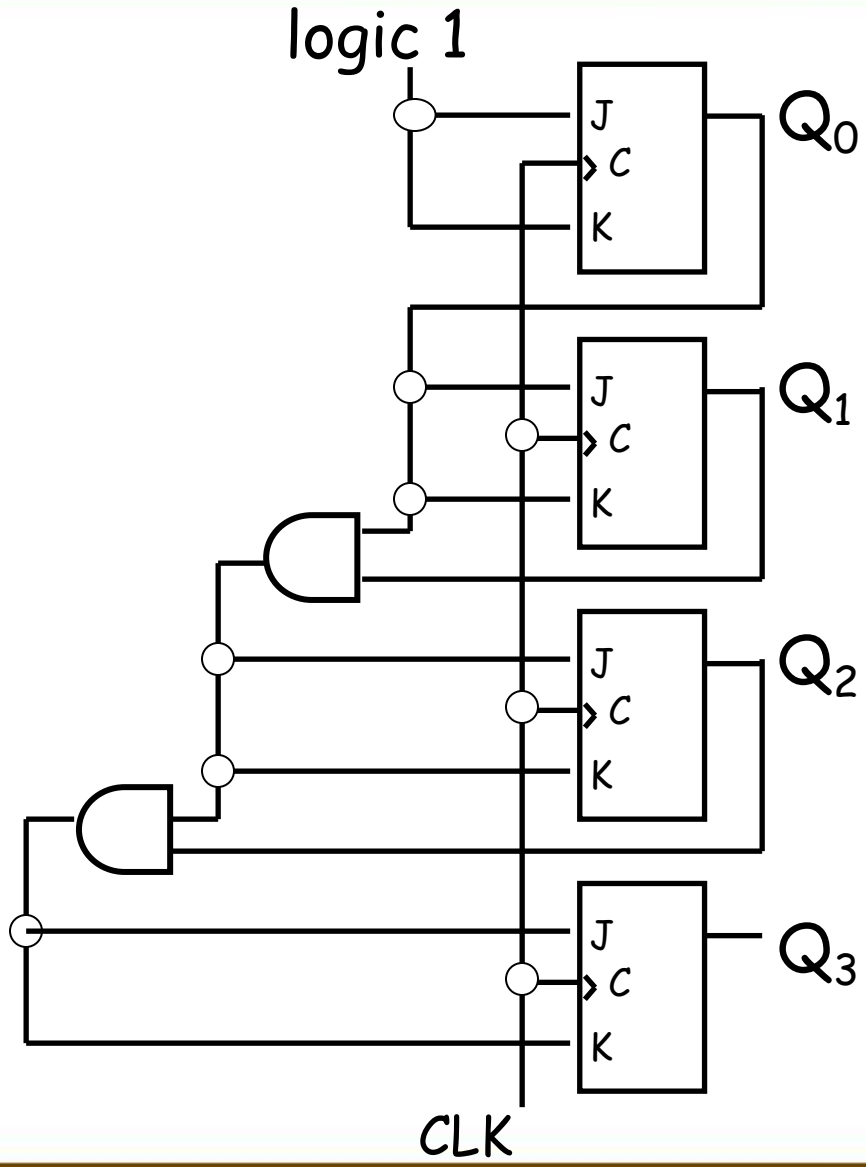
$J_{Q1} = Q_0$



$K_{Q1} = Q_0$

Synchronous Binary Counters

J-K Flip Flop Design of a Binary Up Counter (cont.)



$J_{Q0} = 1$
$K_{Q0} = 1$

$J_{Q1} = Q_0$
$K_{Q1} = Q_0$

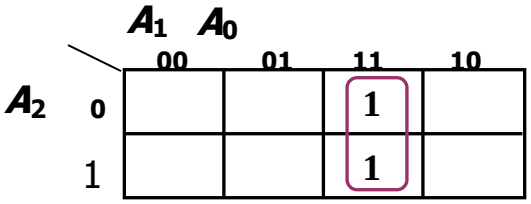
$J_{Q2} = Q_0 Q_1$
$K_{Q2} = Q_0 Q_1$

$J_{Q3} = Q_0 Q_1 Q_2$
$K_{Q3} = Q_0 Q_1 Q_2$

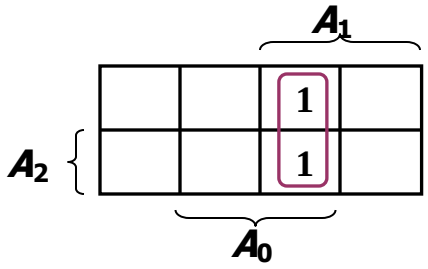
Synchronous (Parallel) Counters

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

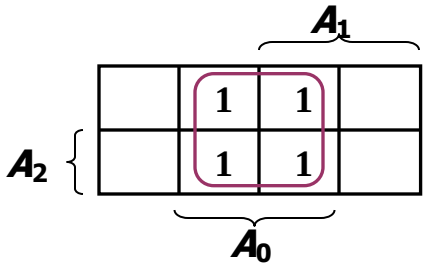
Present state			Next state			Flip-flop inputs		
A_2	A_1	A_0	A_2^+	A_1^+	A_0^+	TA_2	TA_1	TA_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



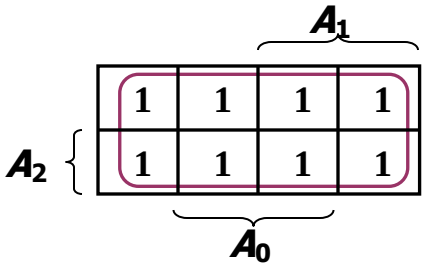
$TA_2 = A_1.A_0$



$TA_2 = A_1.A_0$



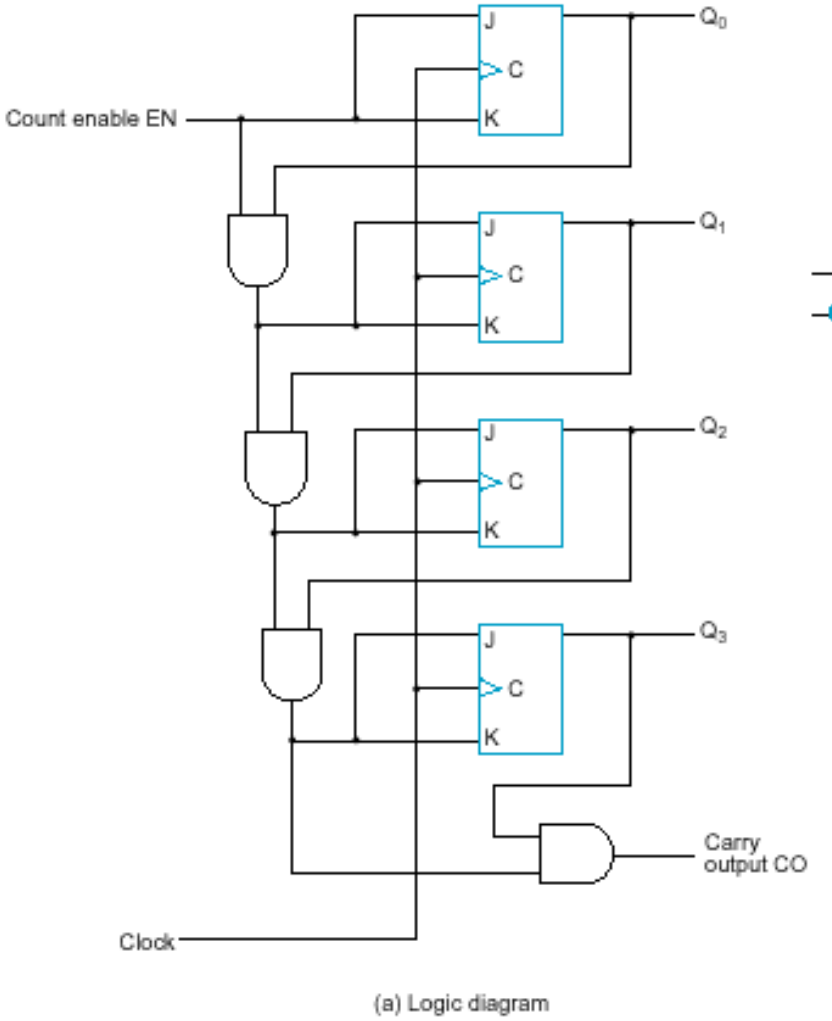
$TA_1 = A_0$



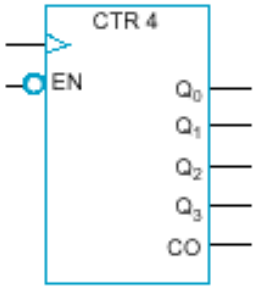
$TA_0 = 1$

Synchronous Binary Counters

J-K Flip Flop Design of a Binary Up Counter with EN and CO



(a) Logic diagram



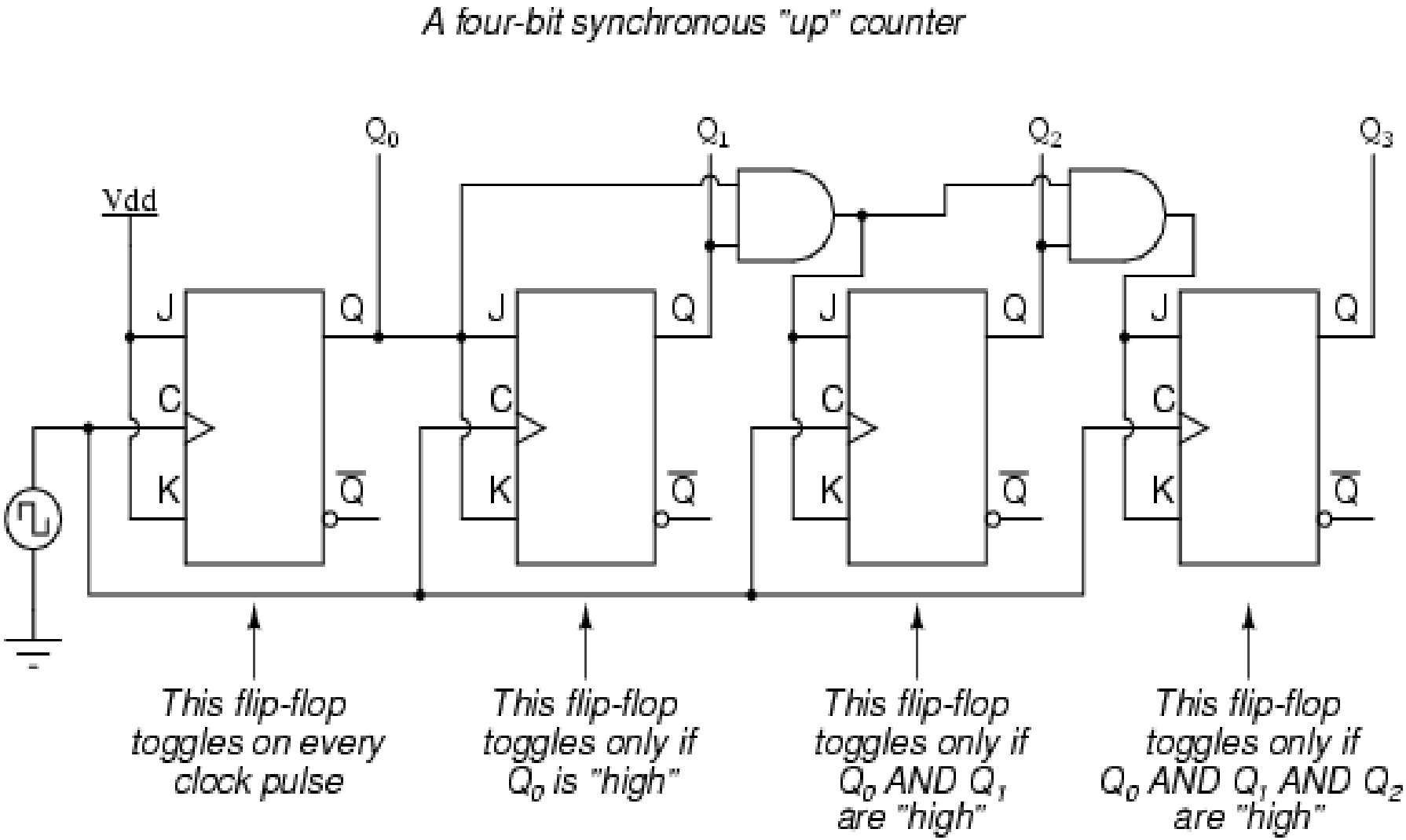
(b) Symbol

EN = enable control signal, when 0 counter remains in the same state, when 1 it counts

CO = carry output signal, used to extend the counter to more stages

$J_{Q0} = 1 \cdot EN$
$K_{Q0} = 1 \cdot EN$
$J_{Q1} = Q_0 \cdot EN$
$K_{Q1} = Q_0 \cdot EN$
$J_{Q2} = Q_0 Q_1 \cdot EN$
$K_{Q2} = Q_0 Q_1 \cdot EN$
$J_{Q3} = Q_0 Q_1 Q_2 \cdot EN$
$K_{Q3} = Q_0 Q_1 Q_2 \cdot EN$
$CO = Q_0 Q_1 Q_2 Q_3 \cdot EN$

Counter


A four-bit synchronous "up" counter
Vdd
Q0
Q1
Q2
Q3
J
C
K
Q
This flip-flop toggles on every clock pulse
This flip-flop toggles only if Q0 is "high"
This flip-flop toggles only if Q0 AND Q1 are "high"
This flip-flop toggles only if Q0 AND Q1 AND Q2 are "high"

Summary

Module II (10 Hours)

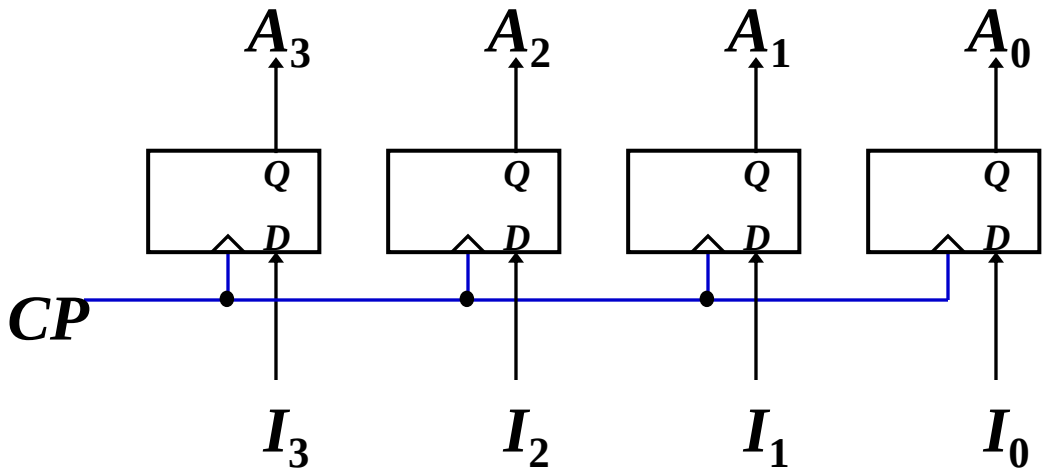
Sequential circuit - Clocking, Flip flops - SR – JK- D -T flip flops, Counters - Synchronous and asynchronous counters - UP/DOWN counters, Registers - Serial in serial out - Serial in parallel out - Parallel in serial out - Parallel in parallel out registers

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.

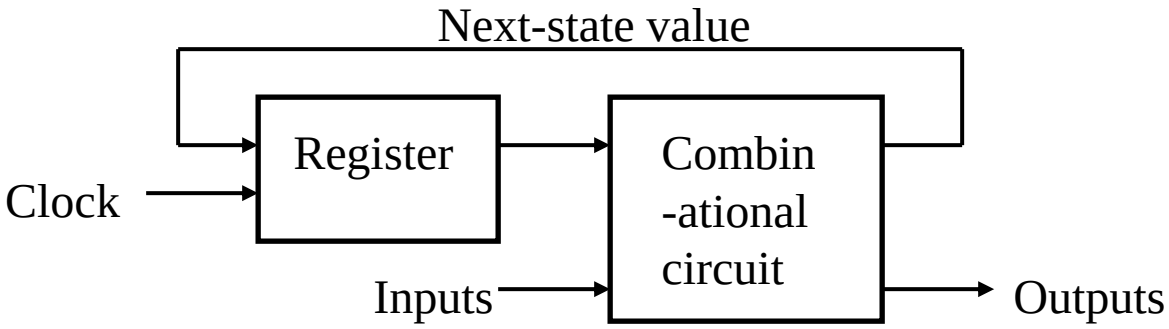


Registers With Parallel Load

- 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.

Using Registers to implement Sequential Circuits

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



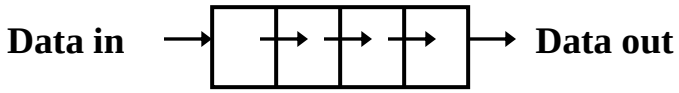
- The external inputs and present states of the register determine the next states of the register and the external outputs, through the combinational circuit.

Shift Registers

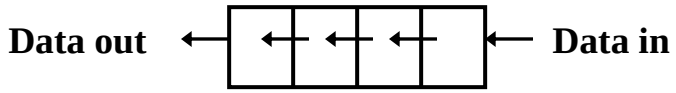
- Another function of a register, besides storage, is to provide for *data movements*.
- Each *stage* (flip-flop) in a shift register represents one bit of storage, and the shifting capability of a register permits the movement of data from stage to stage within the register, or into or out of the register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

Shift Registers

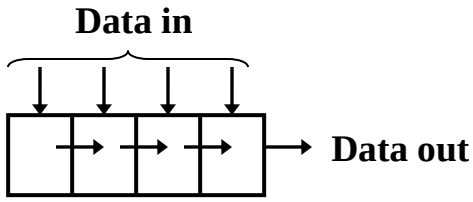
- Basic data movement in shift registers (four bits are used for illustration).



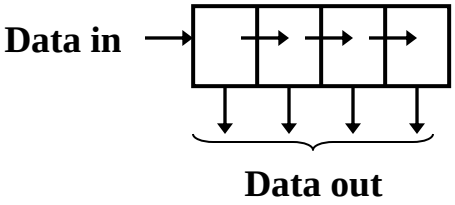
(a) Serial in/shift right/serial out



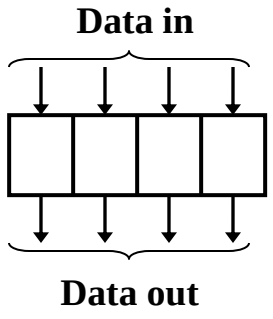
(b) Serial in/shift left/serial out



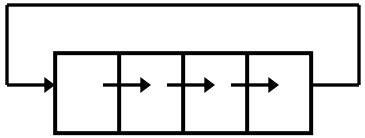
(c) Parallel in/serial out



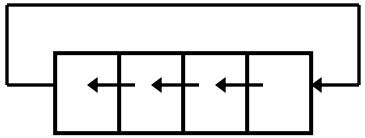
(d) Serial in/parallel out



(e) Parallel in / parallel out



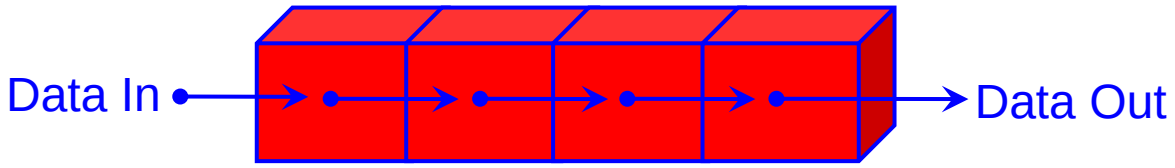
(f) Rotate right



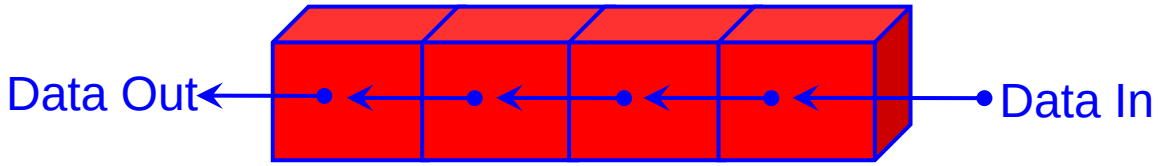
(g) Rotate left

Shift Register

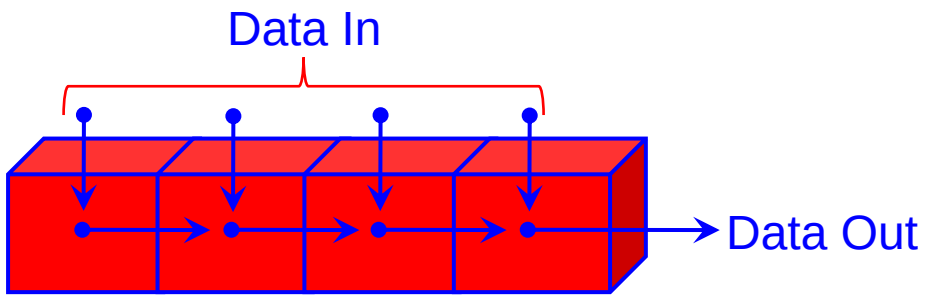
Serial In / Serial Out
Left-to-Right



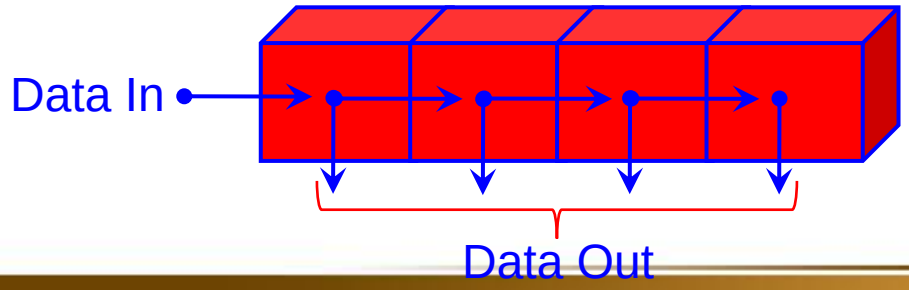
Serial In / Serial Out
Right-to-Left



Parallel In / Serial Out

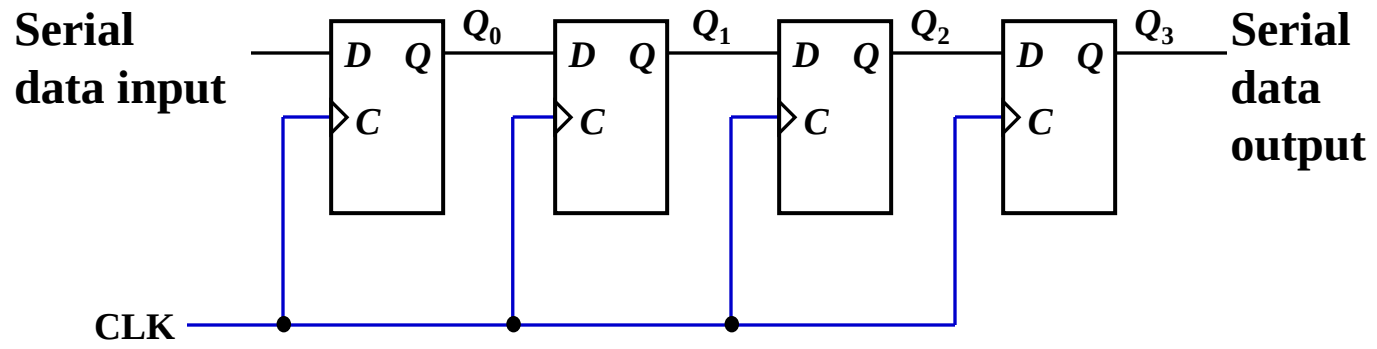


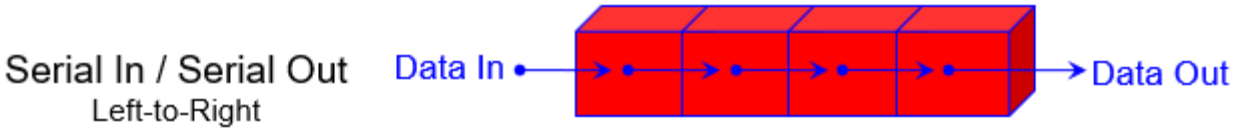
Serial In / Parallel Out



Serial In/Serial Out Shift Registers

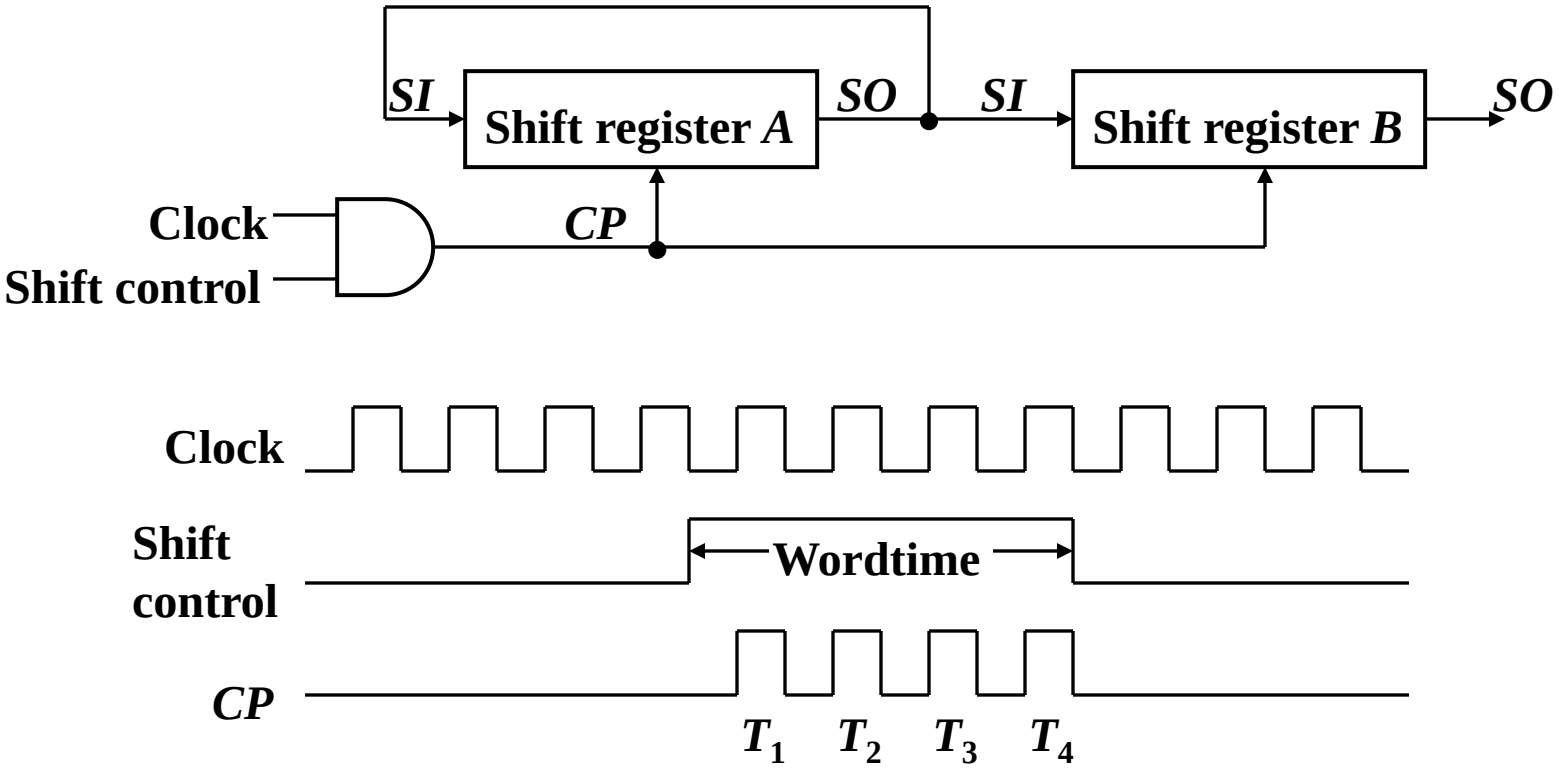
- Accepts data serially – one bit at a time – and also produces output serially.





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.

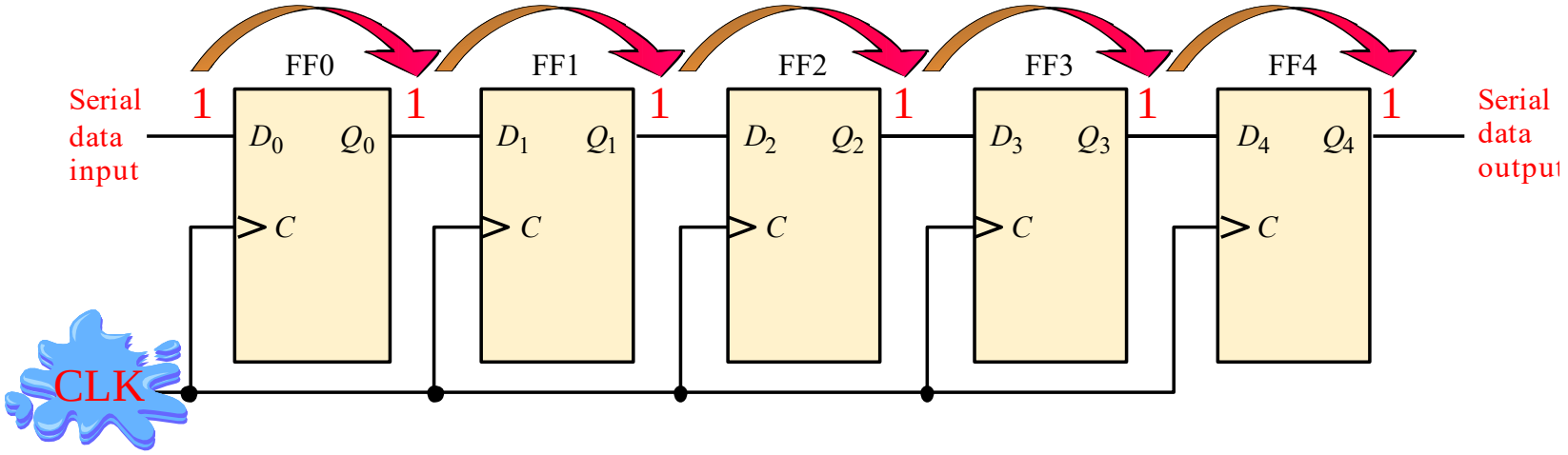
Timing Pulse	Shift register *A*				Shift register *B*				Serial output of *B*
Initial value	1	0	1	1	0	0	1	0	0
After T_1	1	1	0	1	1	0	0	1	1
After T_2	1	1	1	0	1	1	0	0	0
After T_3	0	1	1	1	0	1	1	0	0
After T_4	1	0	1	1	1	0	1	1	1

Summary

Serial-in/Serial out Shift Register

Shift registers are available in IC form or can be constructed from discrete flip-flops as is shown here with a five-bit serial-in serial-out register.

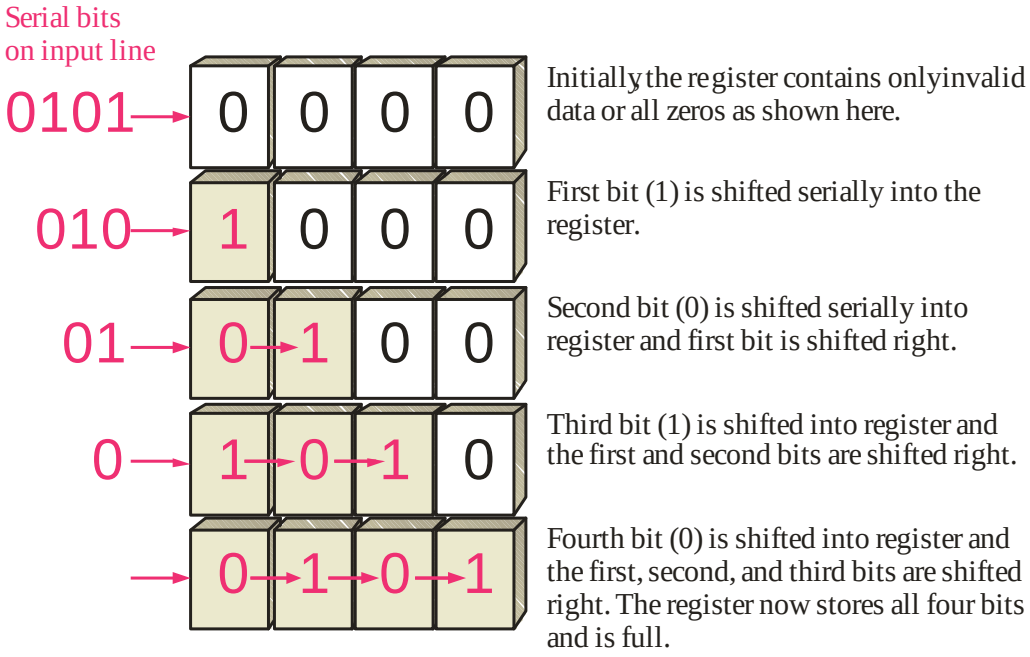
Each clock pulse will move an input bit to the next flip-flop. For example, a 1 is shown as it moves across.



Summary

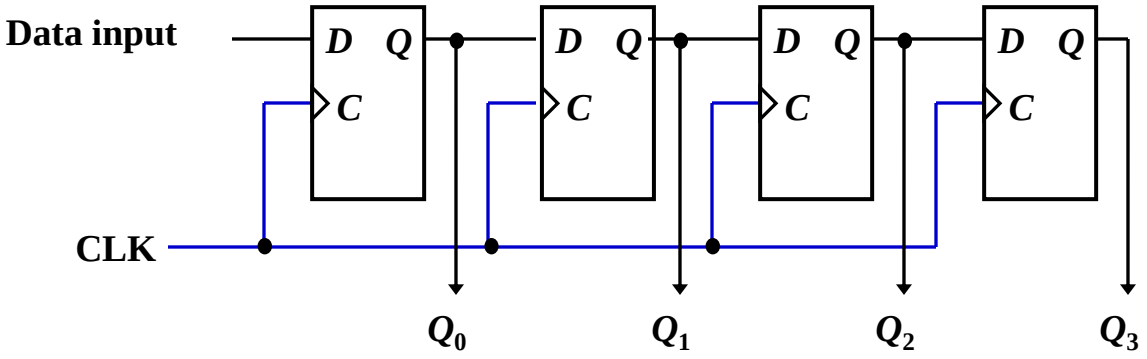
Basic System Functions

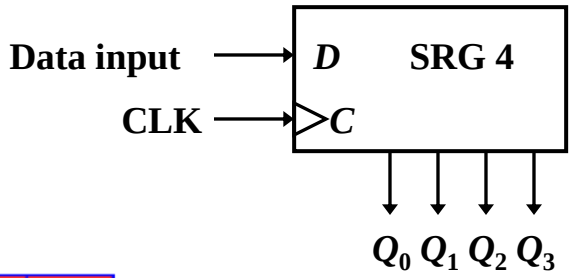
One type of storage function is the shift register, that moves and stores data each time it is clocked.



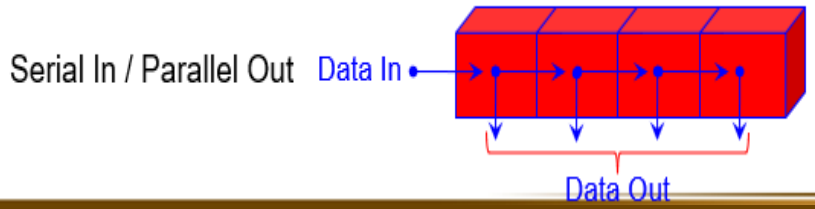
Serial In/Parallel Out Shift Registers

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



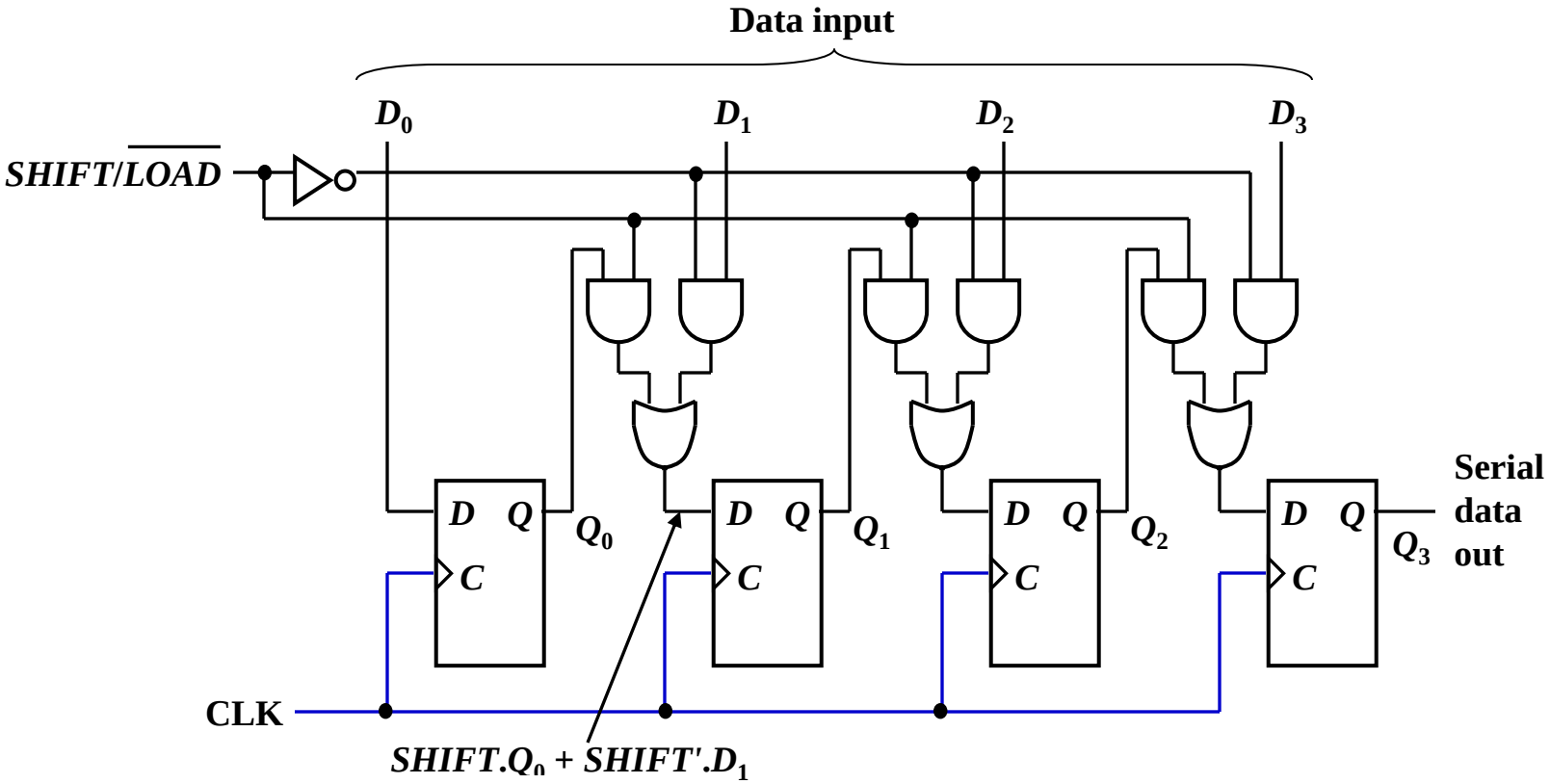


Logic symbol



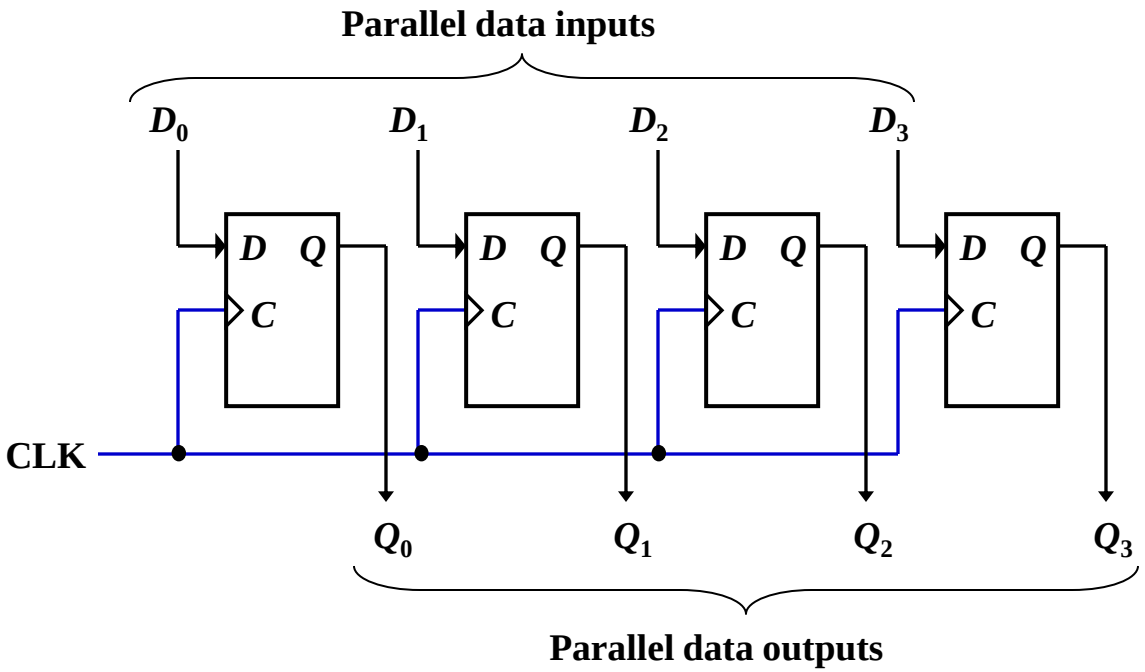
Parallel In/Serial Out Shift Registers

- Bits are entered simultaneously, but output is serial.



Parallel In/Parallel Out Shift Registers

- Simultaneous input and output of all data bits.



Shift Register Counters

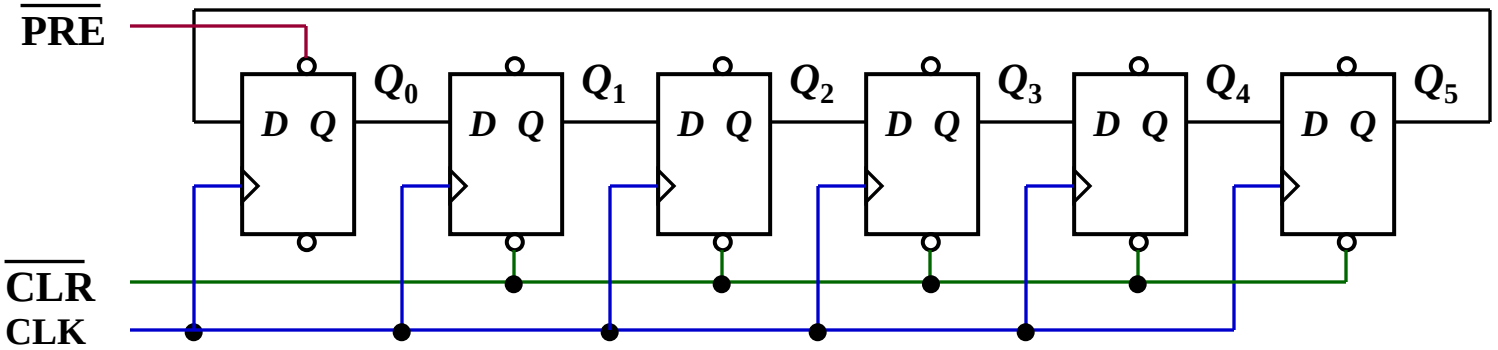
- Shift register counter: a shift register with the serial output connected back to the serial input.
- They are classified as counters because they give a specified sequence of states.
- Two common types: the *Johnson counter* and the *Ring counter*.

Ring Counters

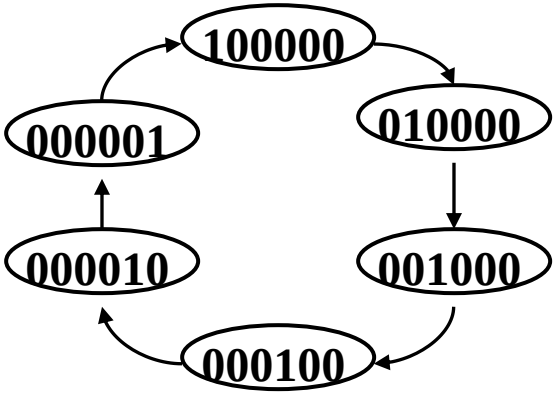
- One flip-flop (stage) for each state in the sequence.
- The output of the last stage is connected to the D input of the first stage.
- An *n*-bit ring counter cycles through *n* states.
- No decoding gates are required, as there is an output that corresponds to every state the counter is in.

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
6	1	0	0	0	0	0
7	0	1	0	0	0	0

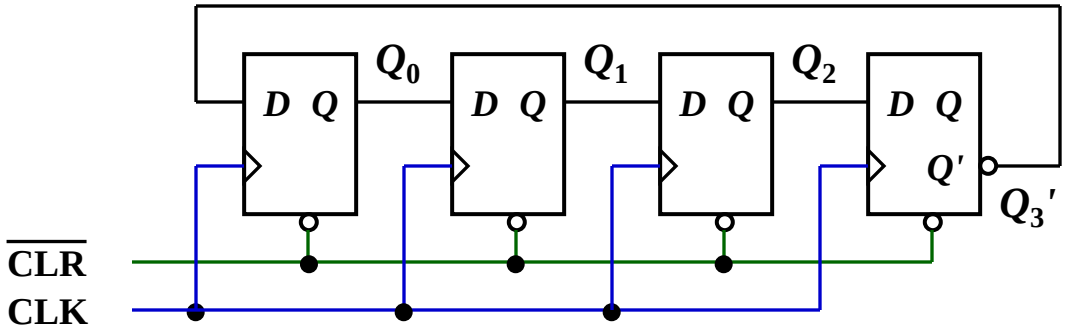


Johnson Counters

- 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