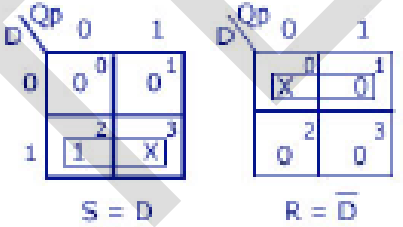
SR to D

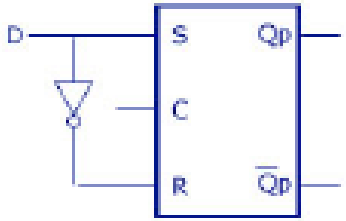
Conversion Table

D Input	Outputs		S-R Inputs	
	Qp	Qp+1	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

K-maps



Logic Diagram

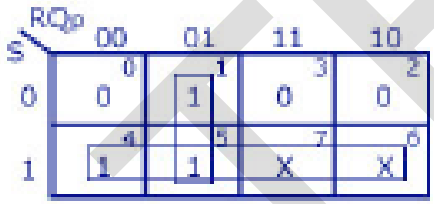


D to SR

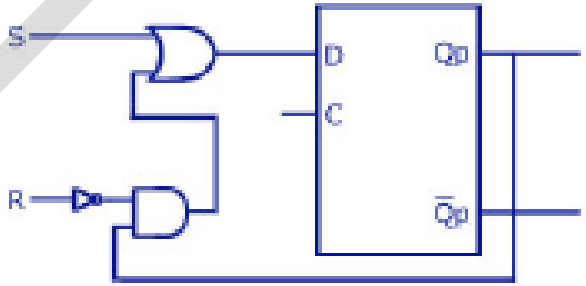
Conversion Table

S-R Inputs S	R	Outputs Qp	Qp+1	D Input
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	Invalid		Dont care
1	1	Invalid		Dont care

K-map



Logic Diagram

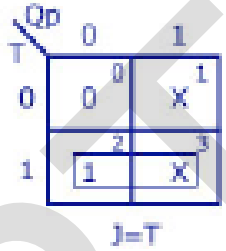


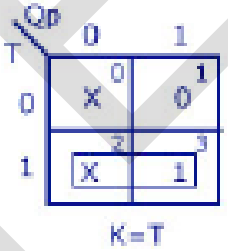
JK to T

Conversion Table

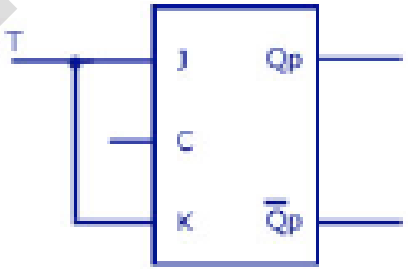
T Input	Outputs Qp	Outputs Qp+1	J-K Inputs J	J-K Inputs K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

K-maps

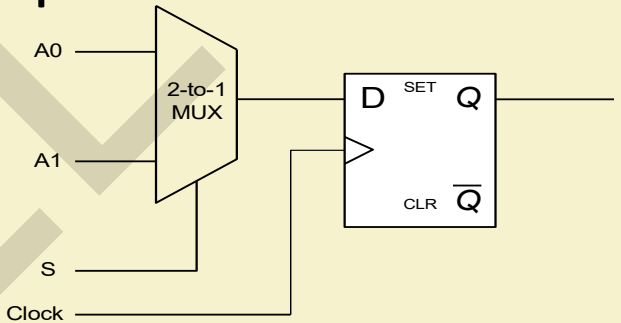


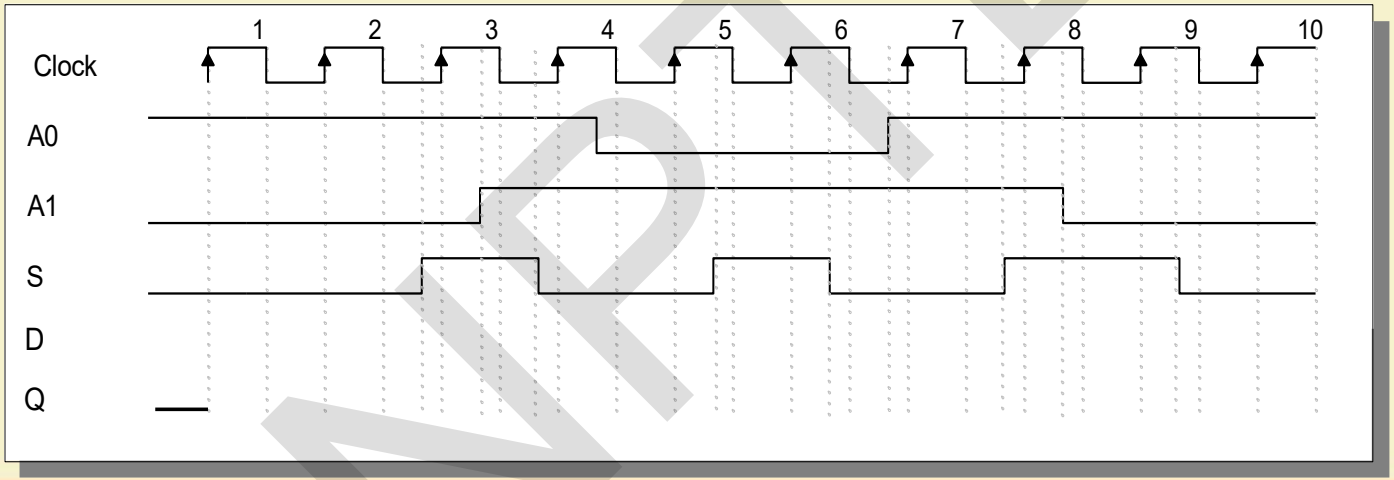


Logic Diagram

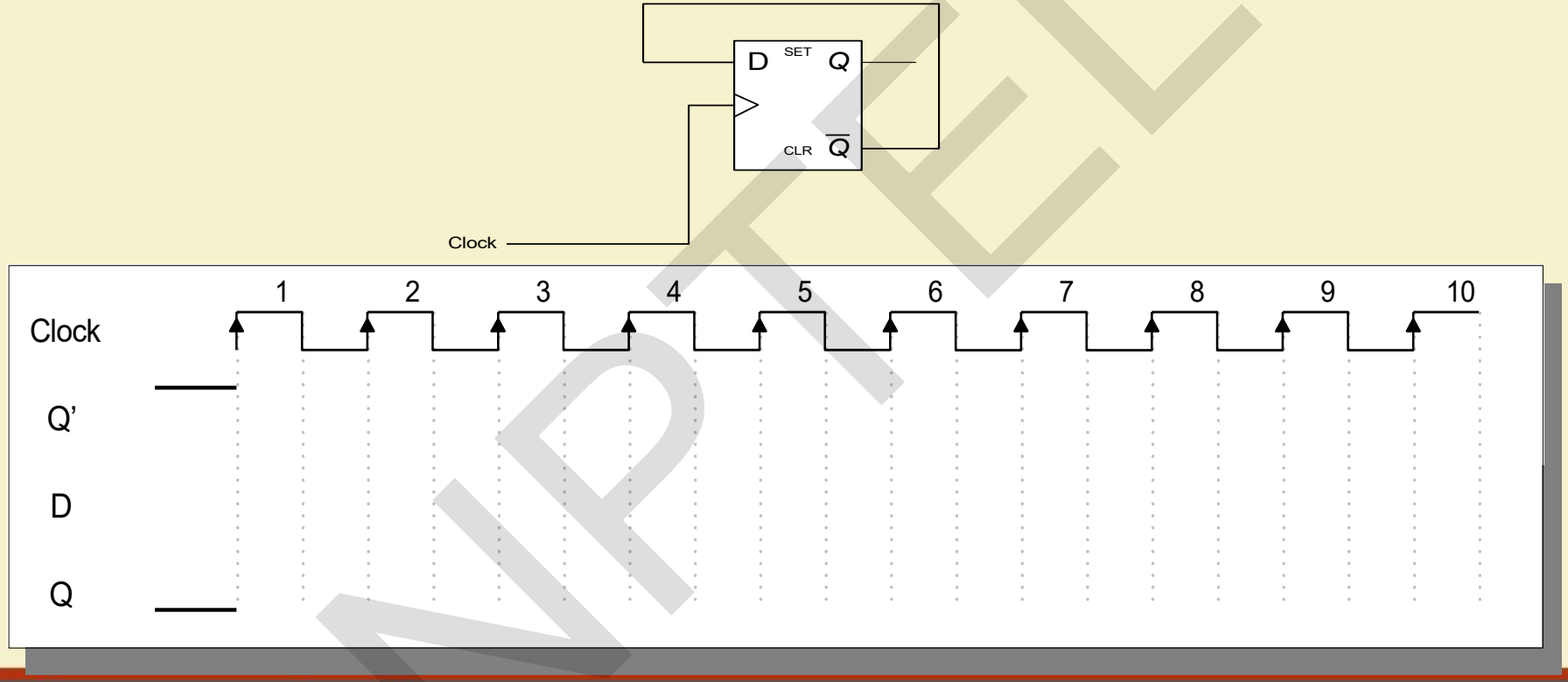


Sequential circuit example



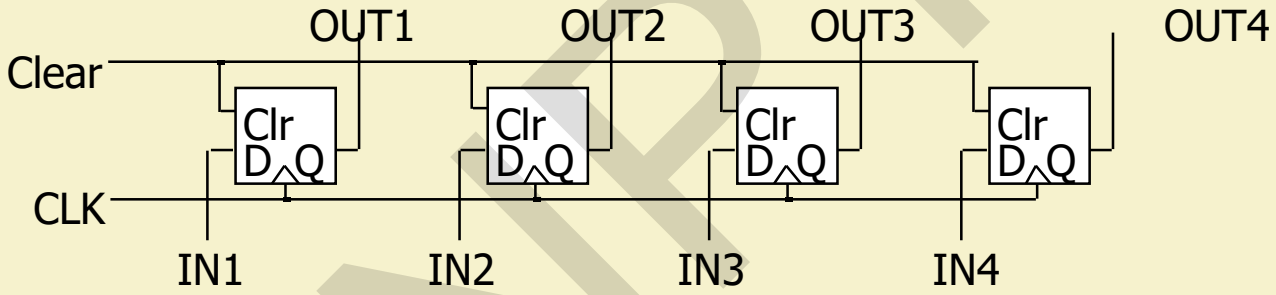


Sequential circuit example

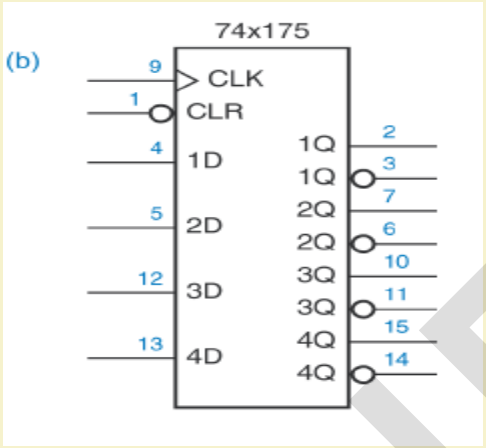


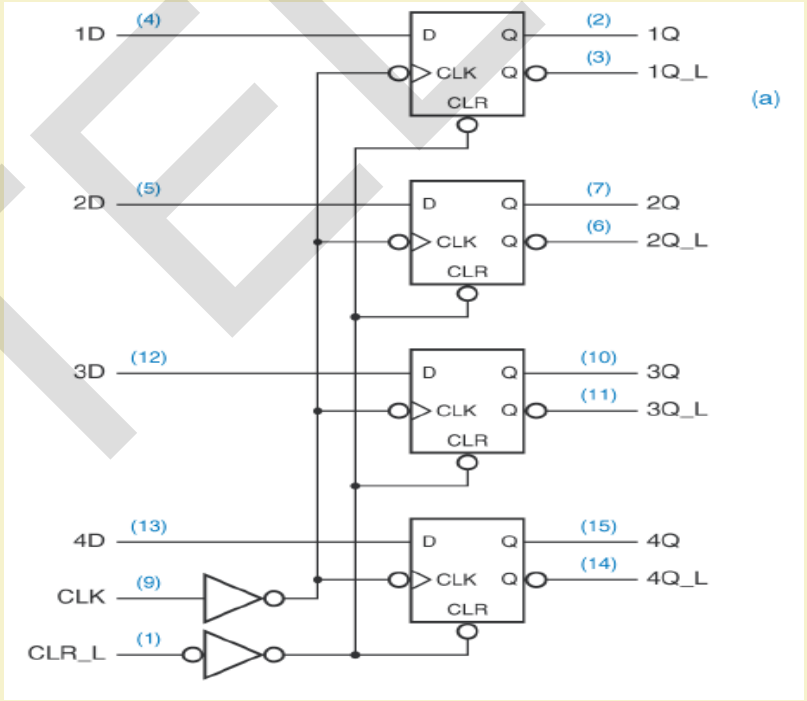
Registers

- A collection of 2 or more D flip flops with a common clock
- Registers are often used to store a collection of related bits (e.g. a byte of data in a computer)

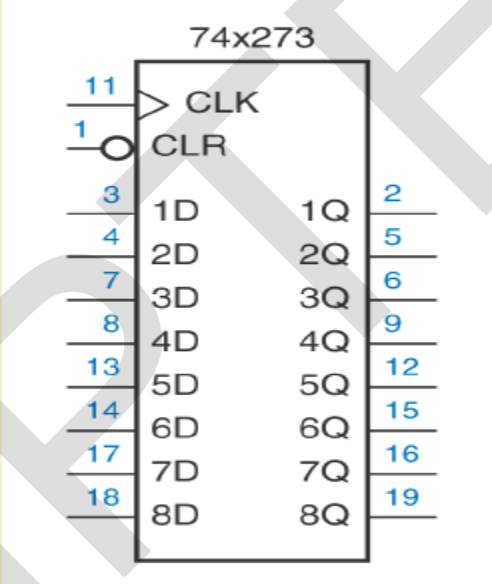


A “standard” 4-bit register IC

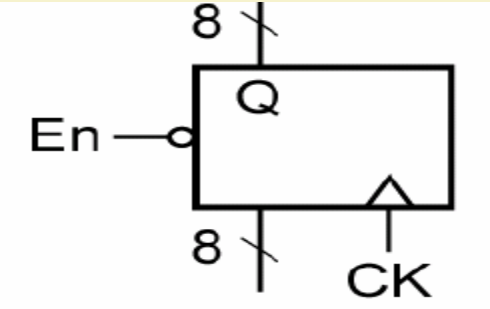




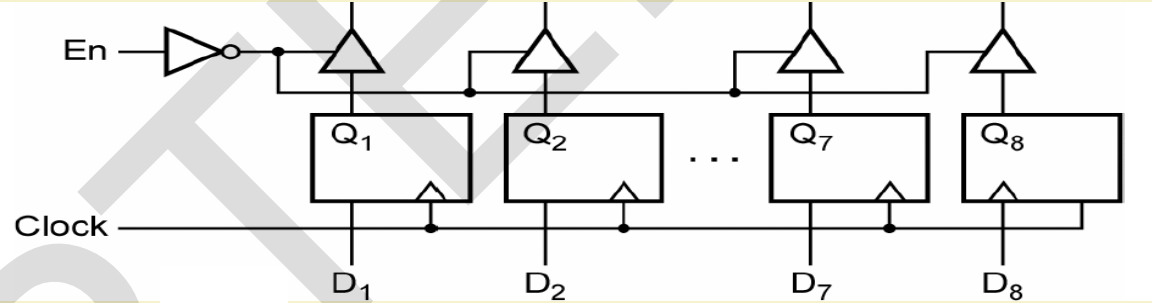
A “standard” 8-bit register IC



Registers with 3-state outputs



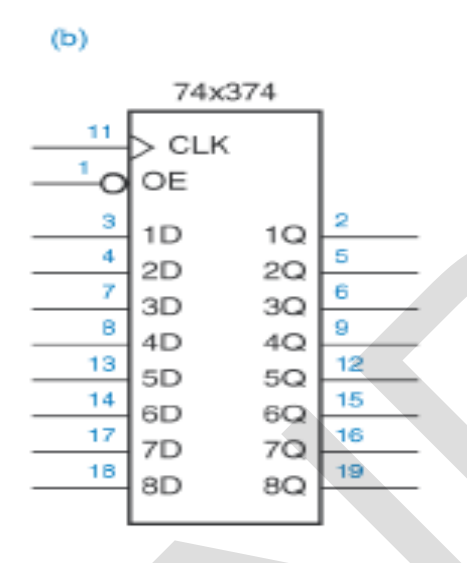
(a) Symbol

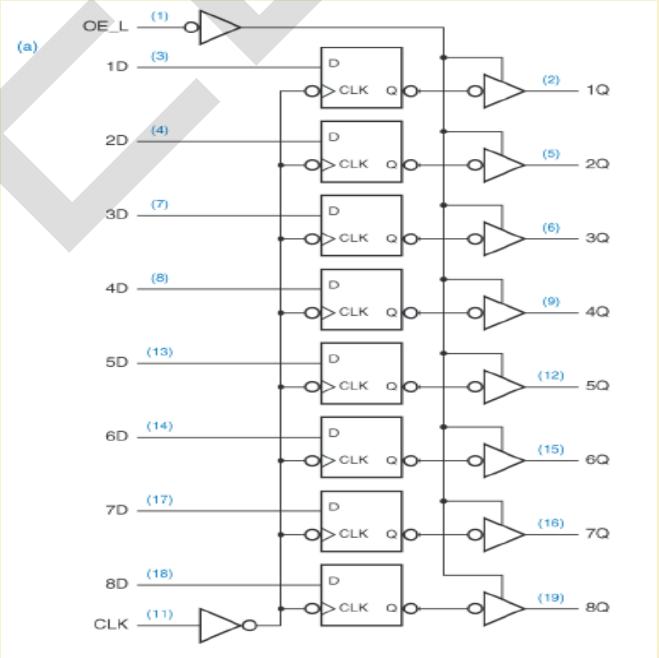


(b) Functional Diagram

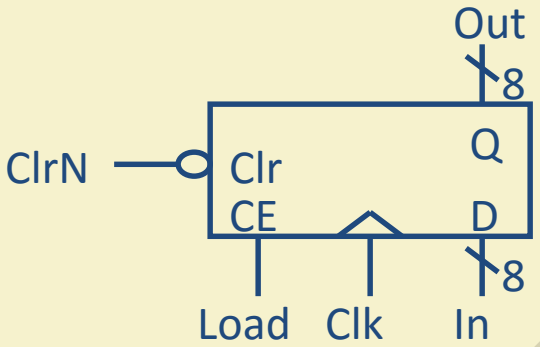
Logic Diagram for 8-Bit Register with Tri-State Output

A “standard” 8-bit register with 3-state outputs

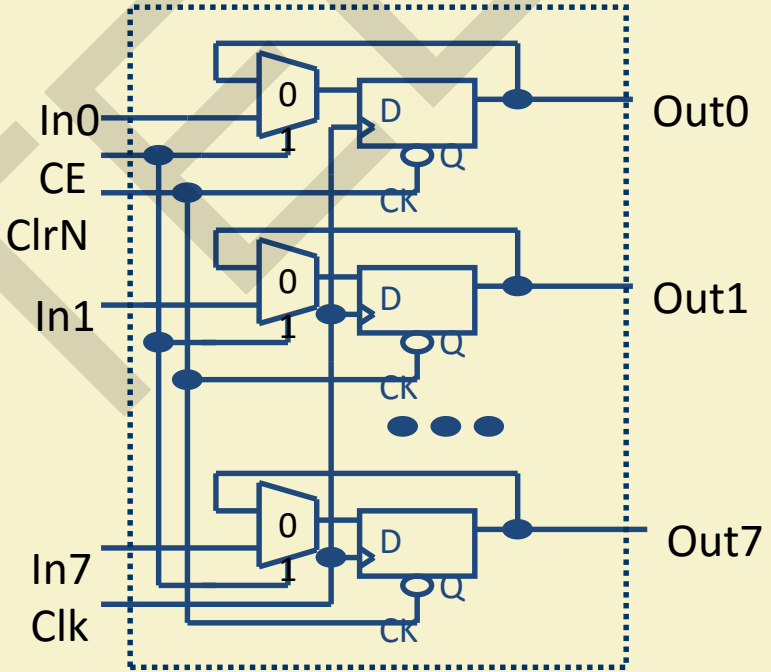




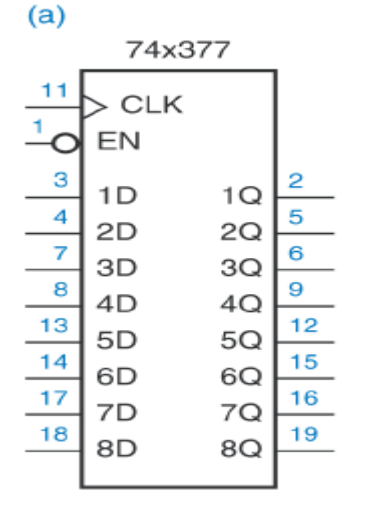
Registers with clock enable

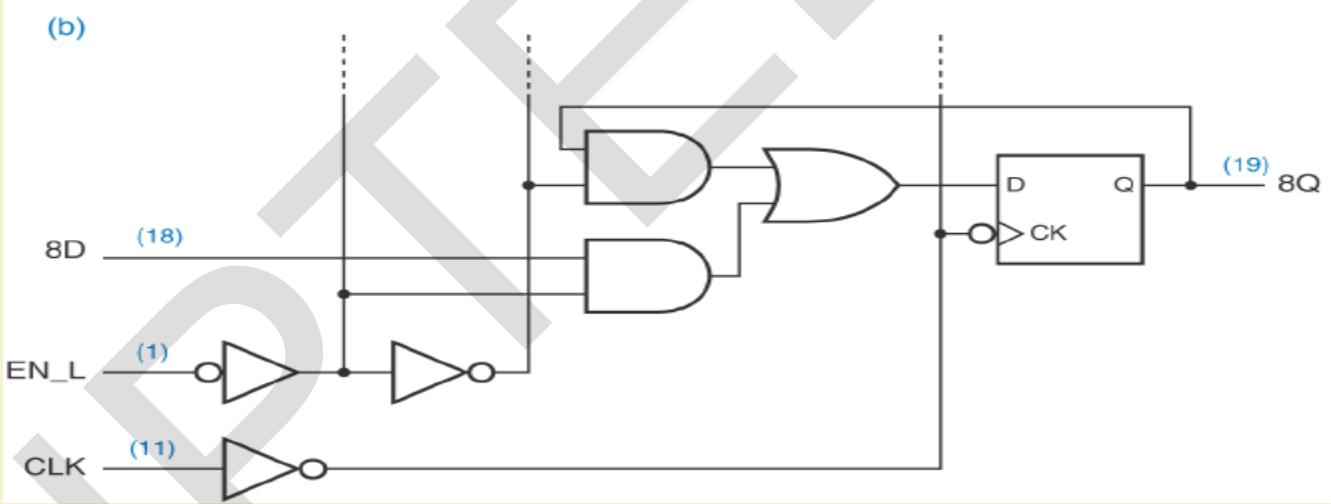


(a) Symbol

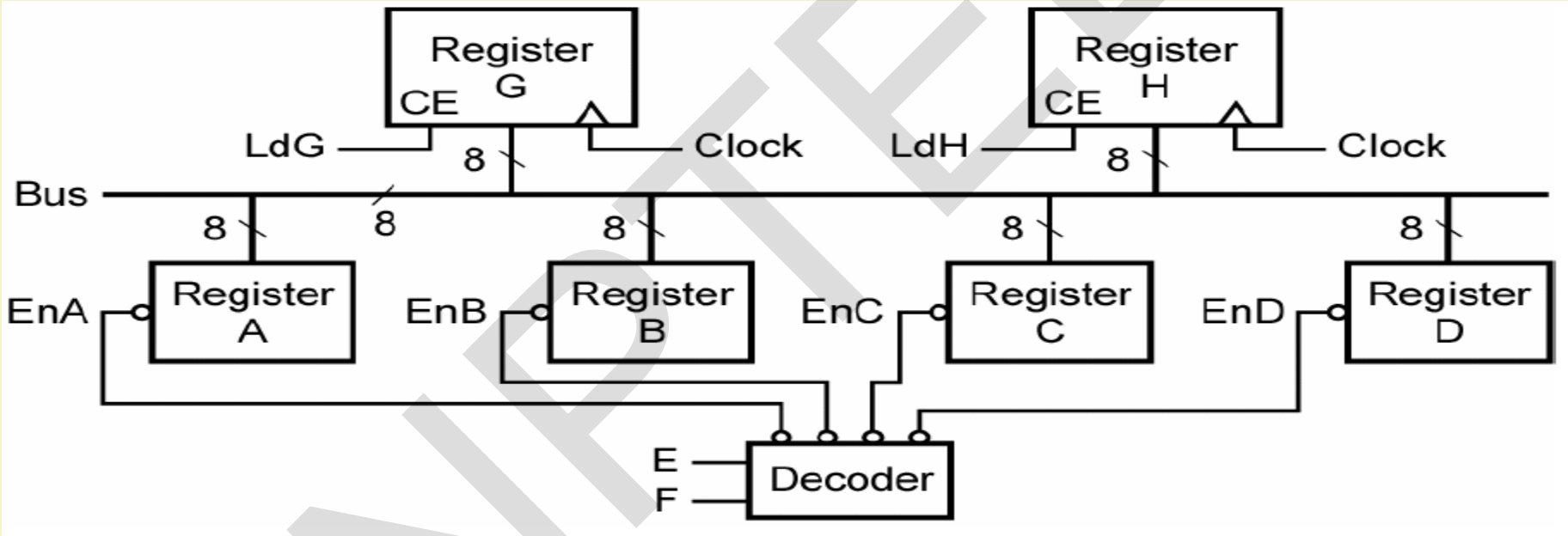


A standard 8 bit register with clock enable (= “gated” clock)



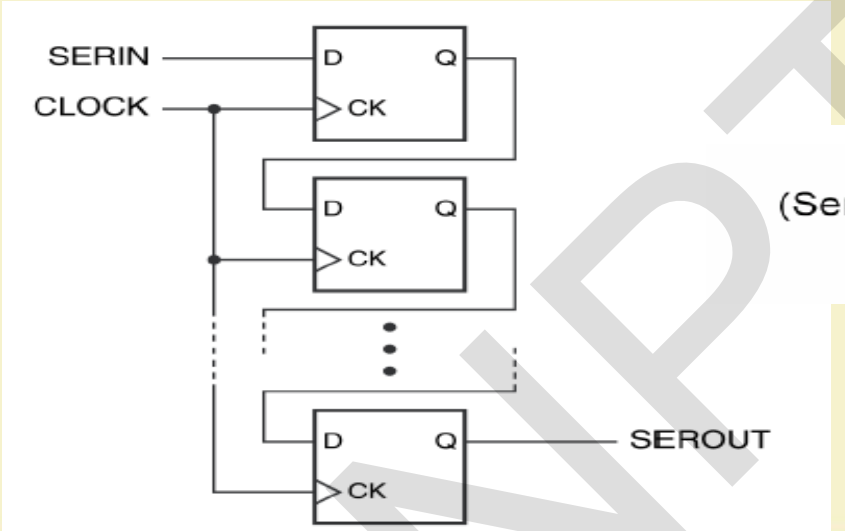


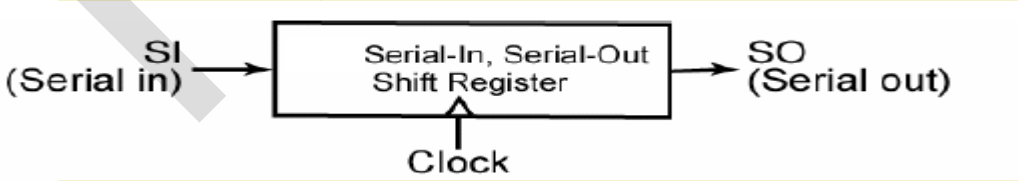
Registers application: Data Transfers



Shift Registers

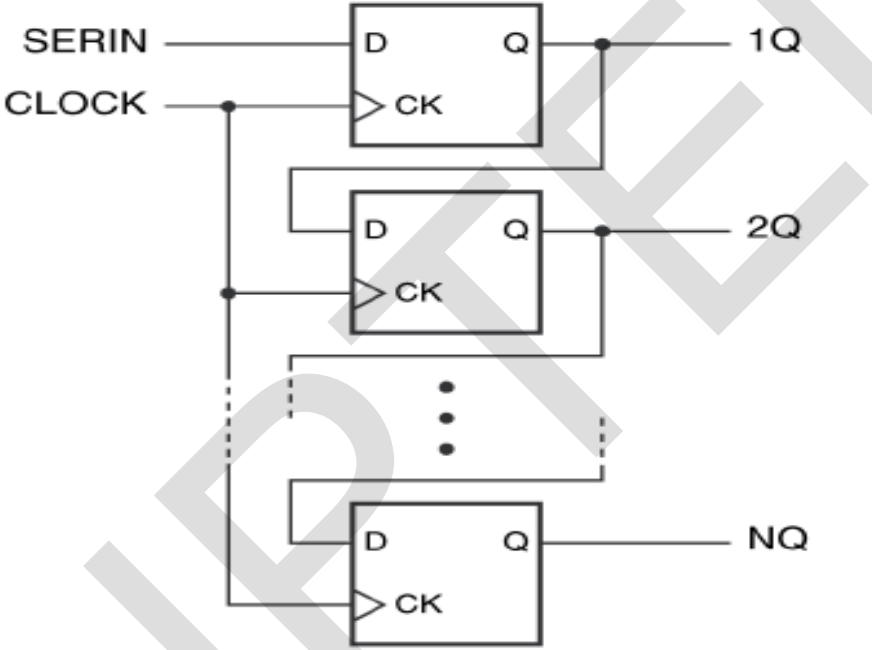
- It is a register that stores input values in sequence. At each clock tick the values stored are shifted from one flip flop to the adjacent





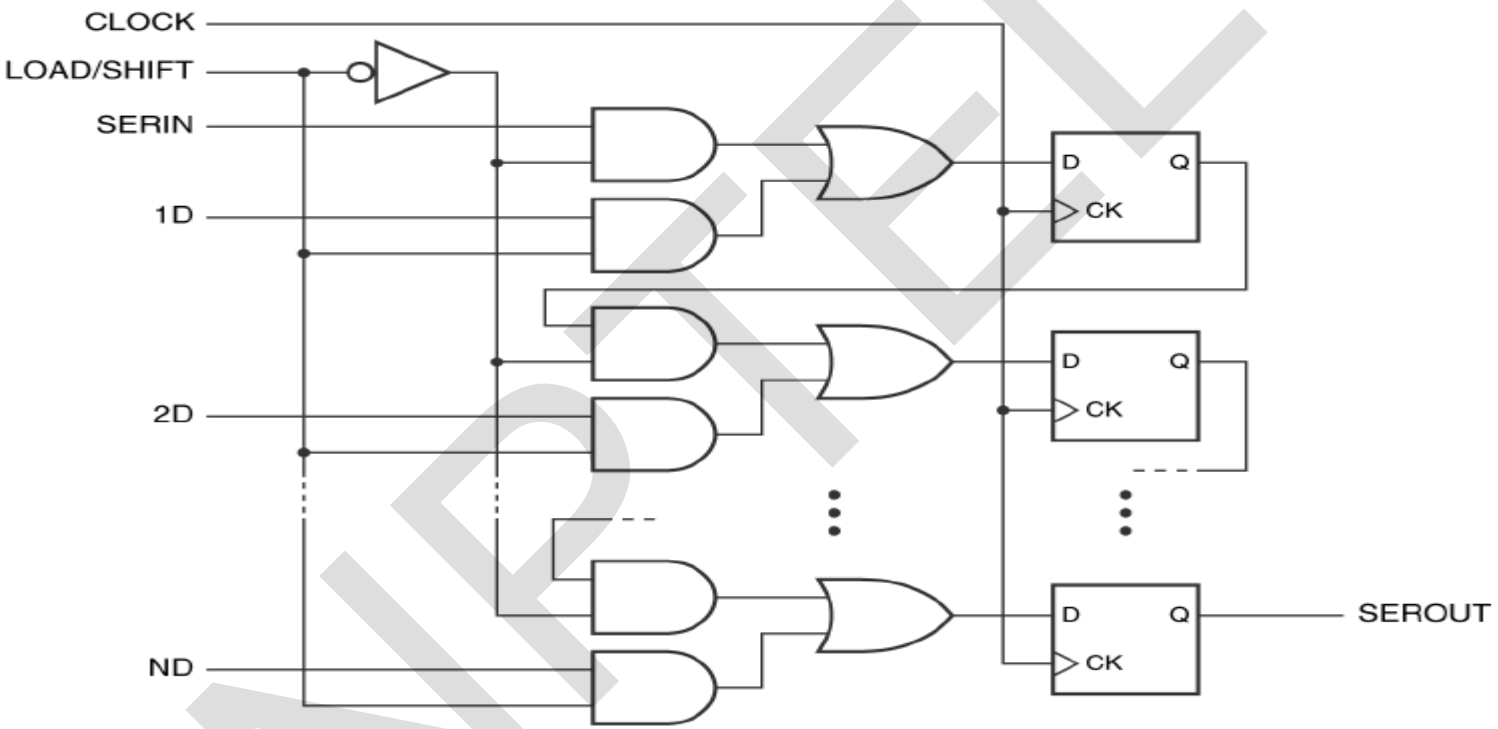
Serial-In, Serial-Out Shift Register

Shift Registers (cont'd)



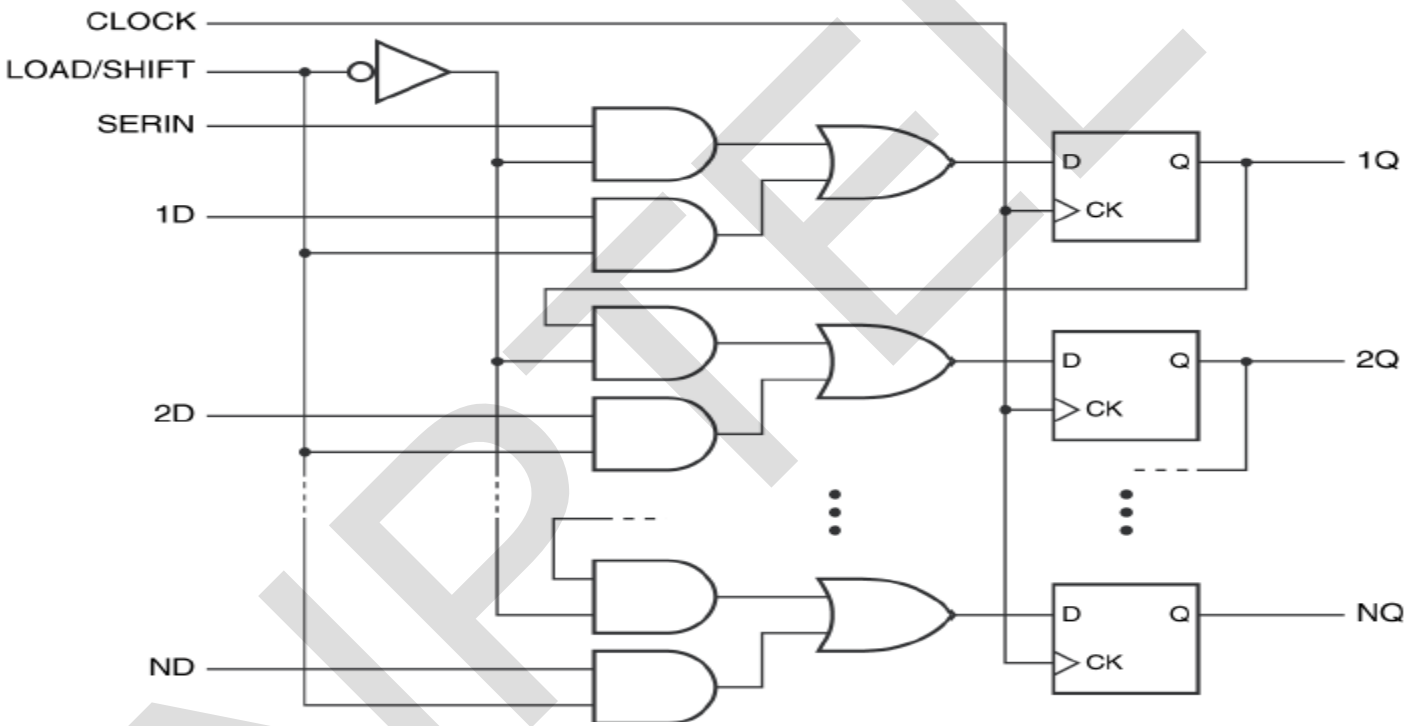
Structure of a serial-in, parallel-out shift register.

Shift Registers (cont'd)



Structure of a parallel-in, serial-out shift register.

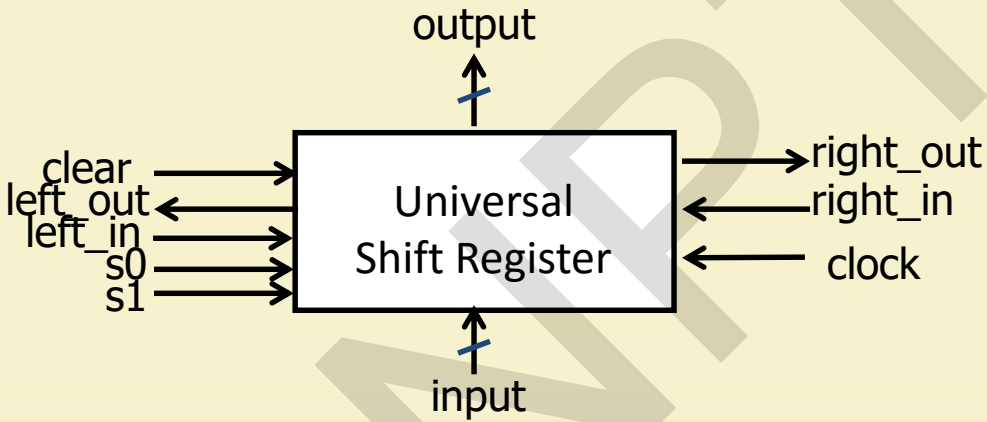
Shift registers (cont'd)



Structure of a parallel-in, parallel-out shift register.

Universal shift register

- serial or parallel inputs
- serial or parallel outputs
- permits shift left or right
- shift in new values from left or right



clear sets the register contents and output to 0

s1 and s0 determine the shift function

s0	s1	function
0	0	hold state
0	1	shift right
1	0	shift left
1	1	load new input

Shift Register Applications

Applications

Ring counters

Johnson counters

Pseudo-random binary sequences and encryption

Ready-made shift registers are available as integrated circuits, such as the ’165

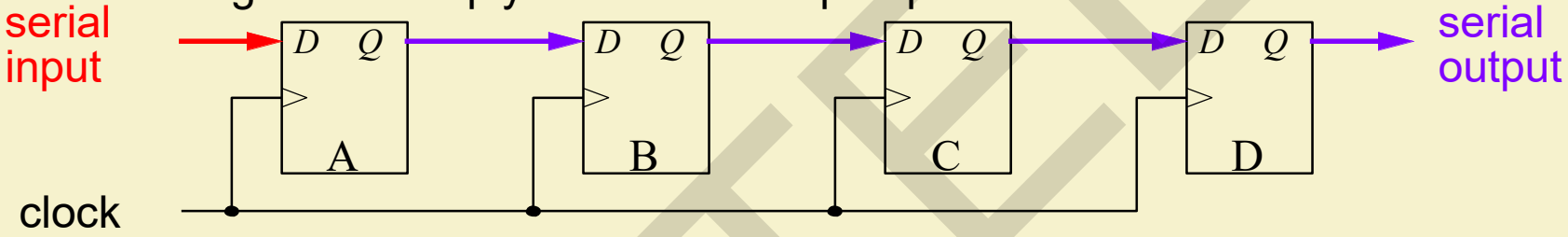
Conversion of data from serial to parallel and *vice versa*

Large-scale devices such as ‘universal asynchronous receiver transmitters’ (UARTs) are based on shift registers

Same functions available in microcontrollers (‘shift’ and ‘rotate’ instructions)

Basic shift register

A basic shift register is simply a chain of *D* flip-flops with a common clock.

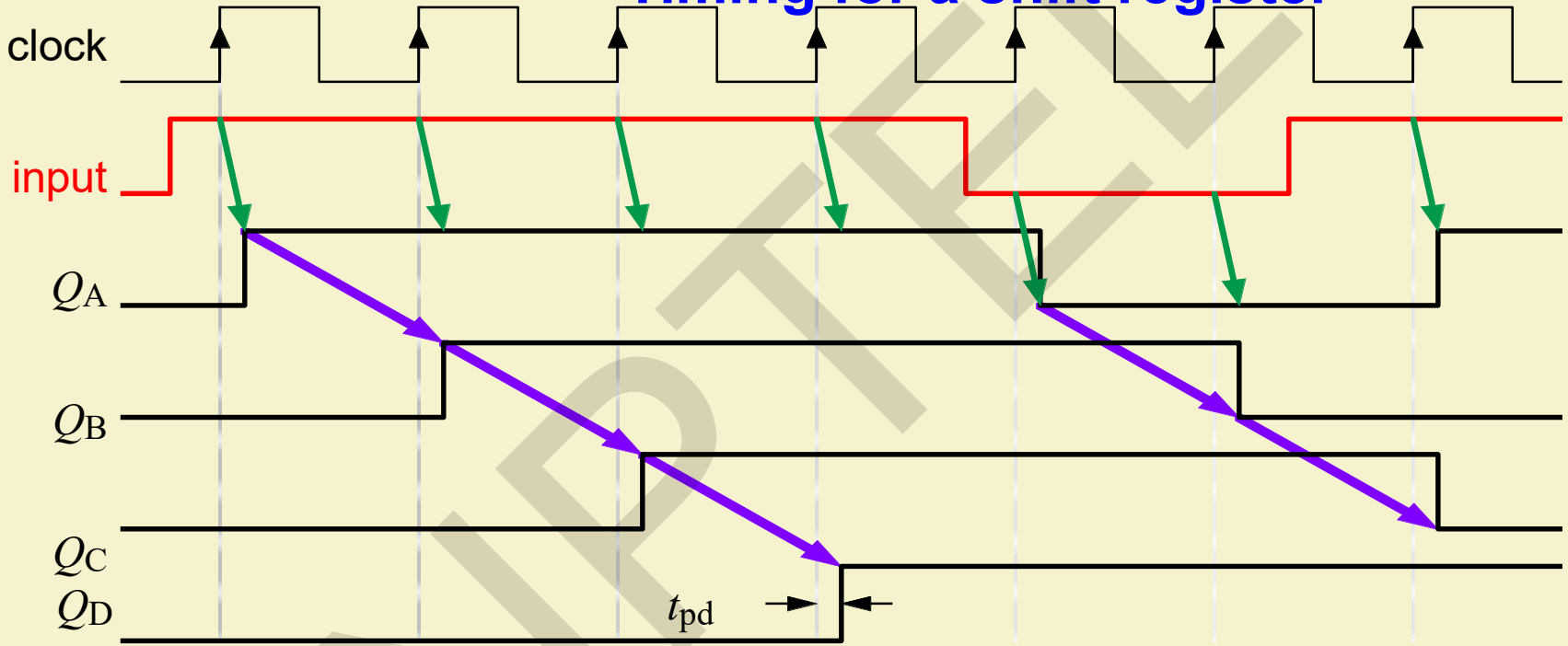


Each flip-flop transfers its *D* input to its *Q* output at a clock transition.

- The effect is to transfer data along the register, one flip-flop per clock cycle.

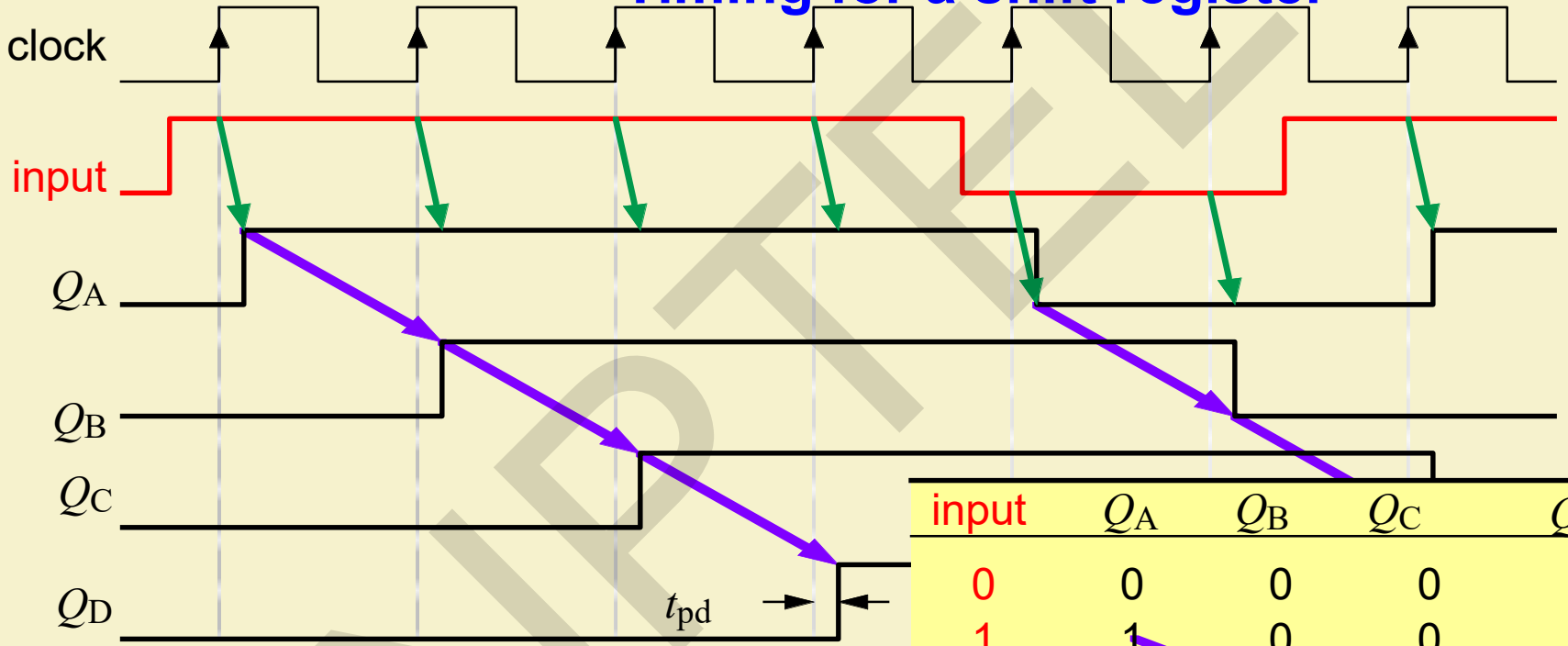
This type of register is called a serial input-serial output (SISO).

Timing for a shift register



Pattern in successive flip-flops moves to the right with each clock cycle to shift the pattern into and out of the register.

Timing for a shift register



input	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0
0	0	1	1	1

Applications of a basic shift register

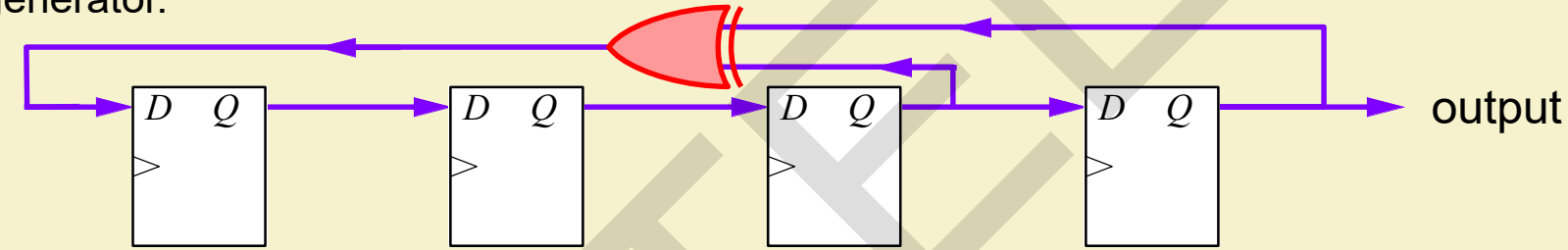
1. **Delay line** — *N* stages delay the signal by *N* clock cycles
2. **Multiplication and division by powers of 2**, because this just requires a shift of the binary number (like multiplication or division by 10 in decimal)

   Example: decimal 3 x 4 = 12 becomes 11 x 100 = 1100 in binary The arithmetic logic unit (ALU) of a computer processor uses a shift register for this purpose.

   Warning: the 'sense' of a shift — left or right — is usually based on its effect on binary numbers written in the usual way. For example,
   11 → 1100 is called a **left shift**. This is clearer if both numbers are written with 8-bits as 00000011 → 00001100. Similarly, dividing by 2 such as 00010110 → 00001011 is a **right shift**.

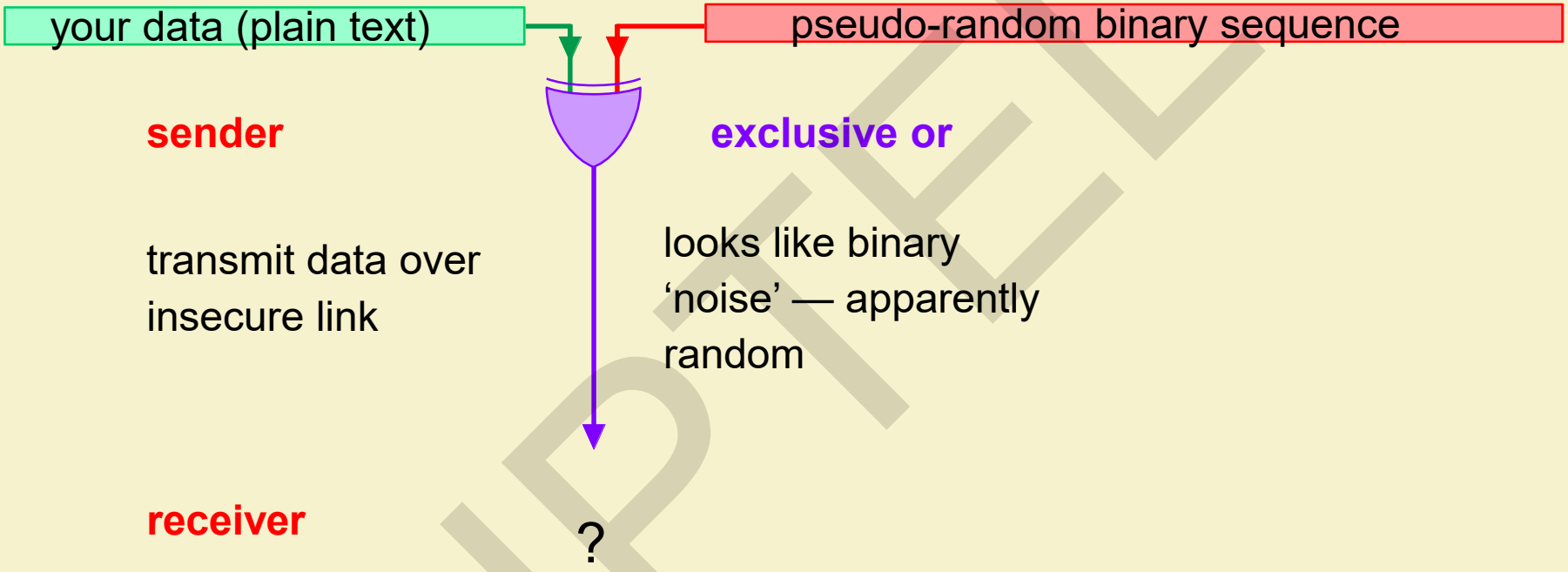
Pseudo-random number generator

A ring counter with feedback through an **exclusive-or gate** makes a simple pseudo-random number generator.

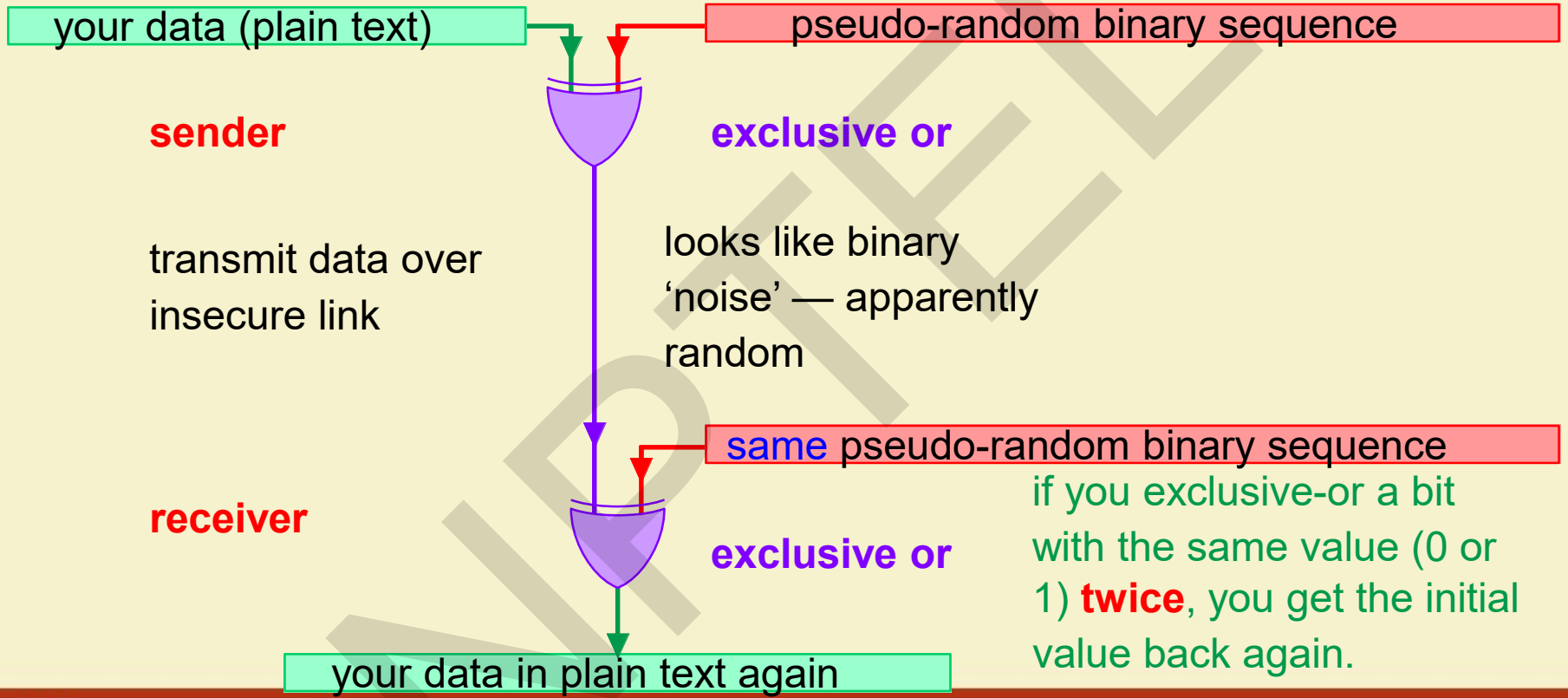


- Pseudo-random sequences of 1s and 0s have many applications, notably in encryption. They appear to be random over 'short' times but the sequence eventually repeats, hence the more accurate term 'pseudo-random'.
- Also, they can be reproduced perfectly if you know both:
  - **the method used to generate the sequence**
  - **the state in the sequence at which to start**
- This is an important feature! — see next sheet.
- The circuit above has a period of $2^4 - 1 = 15$ (the missing state is 0000 —why?).

Pseudo-random binary sequences and encryption



Pseudo-random binary sequences and encryption



if you exclusive-or a bit with the same value (0 or 1) **twice**, you get the initial value back again.

Transmission of data — serial format

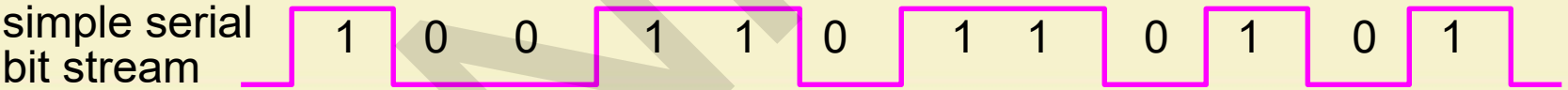
Data often has to be transmitted from one computer to another, or from a computer to peripheral equipment (printer, modem, …). This can be done in:

- **serial format**, one bit at a time
- **parallel format**, several bits at a time (e.g. byte at a time, 8 bits)

Serial format is most commonly used because it is simpler. Only a few wires are needed:

- traditional **serial 'COM' ports (RS-232)** need only 3 wires (transmitted data, received data and ground — but more may be used for control)
- **universal serial bus** (USB, common on modern computers) uses 4 wires (two for differential data plus power and ground)

Traditional serial transmission was slow but modern systems use much faster rates (USB version 1 up to 12 Mbits per second, FireWire 1 up to 400 Mbits per second), version 2 of both even faster.



Parallel data

Where higher speed is required, several bits (usually a small number of bytes, each of 8 bits) may be moved at once. More complicated connections are needed — more wires. Common applications include:

- **inside the processor itself**, e.g. our microcontroller handles bytes
- inside a computer system on the **bus** (e.g. PCI) and interfaces to **disk drives** (e.g. e.g. SCSI or IDE)— but these are now mainly serial

Interfaces have changed to serial because it is hard to ensure that all bits on a parallel bus arrive at the same time at the high speed of modern systems.

Parallel data

Where higher speed is required, several bits (usually a small number of bytes, each of 8 bits) may be moved at once. More complicated connections are needed — more wires. Common applications include:

- **inside the processor itself**, e.g. our microcontroller handles bytes
- inside a computer system on the **bus** (e.g. PCI) and interfaces to **disk drives** (e.g. e.g. SCSI or IDE)— but these are now mainly serial

Interfaces have changed to serial because it is hard to ensure that all bits on a parallel bus arrive at the same time at the high speed of modern systems.

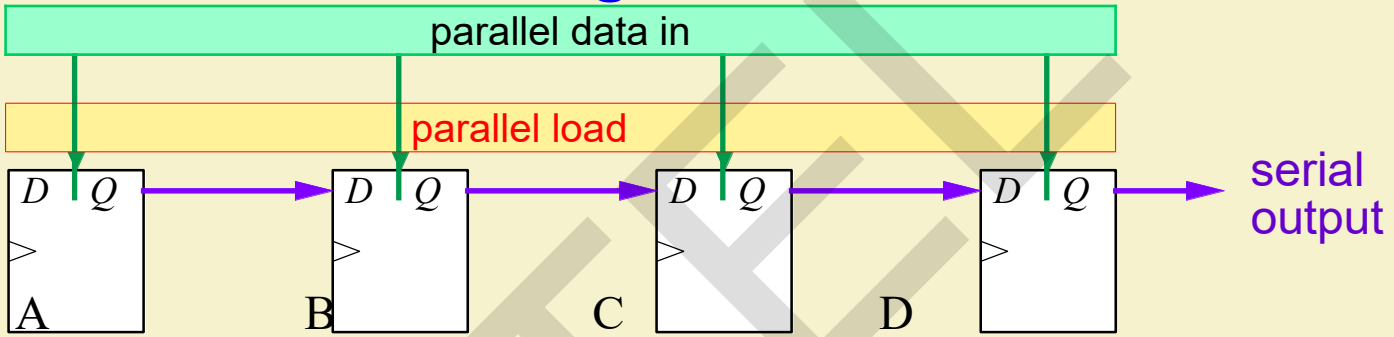
How do you interface a serial device to a computer?

How do we interface an external device that transmits serially with the bus of a computer that transfers one byte (8 bits) at a time?

- **Use a shift register.**

In practice this would almost certainly be buried inside a larger circuit called a **UART** (universal asynchronous receiver transmitter) or something similar.

Use of shift register to serialize data



Extra logic is added to the basic shift register so that all the flip-flops can be loaded in **parallel** (simultaneously), controlled by a shift/load input.

Once the data have been loaded, the clock is enabled and the values are shifted once per clock cycle. This causes the input data to be transferred to the output, one bit at a time — **serial output** (PISO).

The opposite process is used to read in serial data, fill up the shift register, and transfer it in parallel to a bus when the register is full (SIPO).

The register can also be parallel input – parallel output (PIPO).

Asynchronous and Synchronous Counters

Counters

* Counters are important digital electronic circuits.

* They are Sequential logic circuits because timing is obviously important and they need a memory characteristic.

* Digital counters have the following important characteristics,

1. Maximum number of count
2. Up-Down Count
3. Asynchronous or Synchronous Operation
4. Free-Running or Self-Stopping

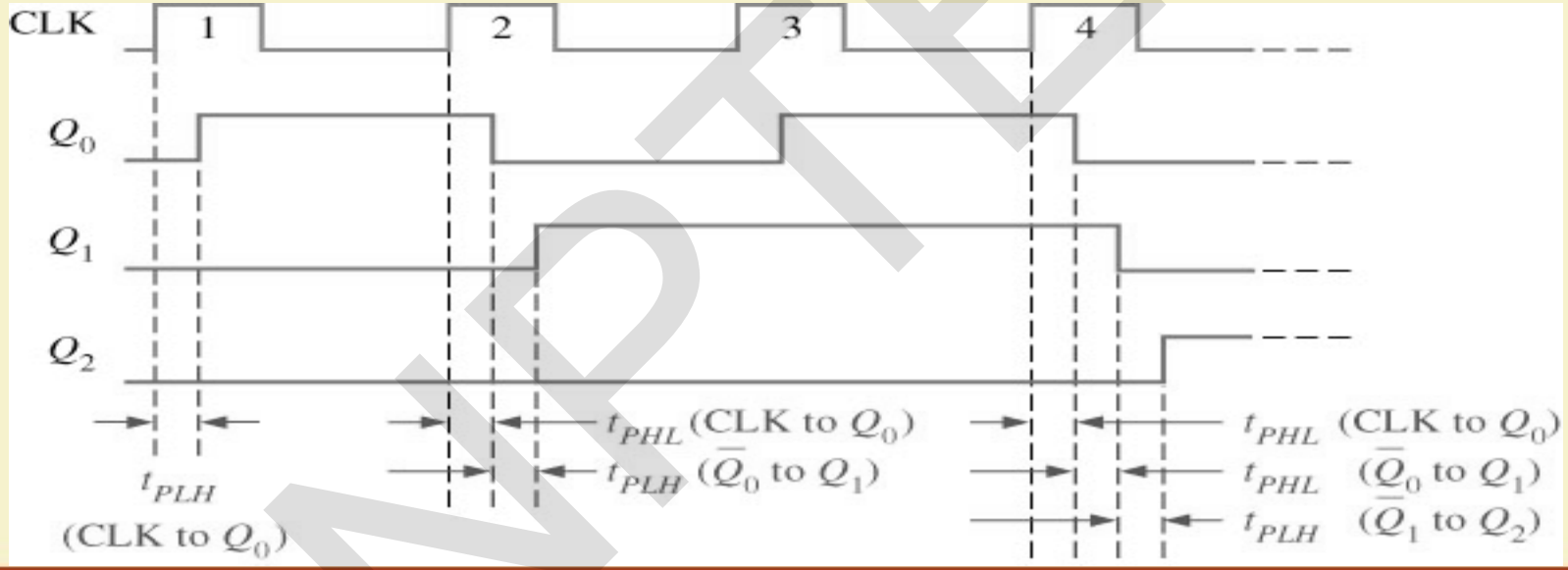
Asynchronous/Ripple Counter

- Asynchronous counters are commonly referred to as ripple counter because the effect of the input clock pulse is first “felt” by first flip-flop (FF0).
- Cannot get to the second flip-flop (FF1) immediately because of the propagation delay through FF0.
- So the effect of an input clock pulse “ripples” through the counter, taking some time, due to propagation delays, to reach the last flip-flop.

Only the first FF receive clock pulse from the source (clock genarator), others FFs receive clock pulse from either Q or Q’ of prior FF

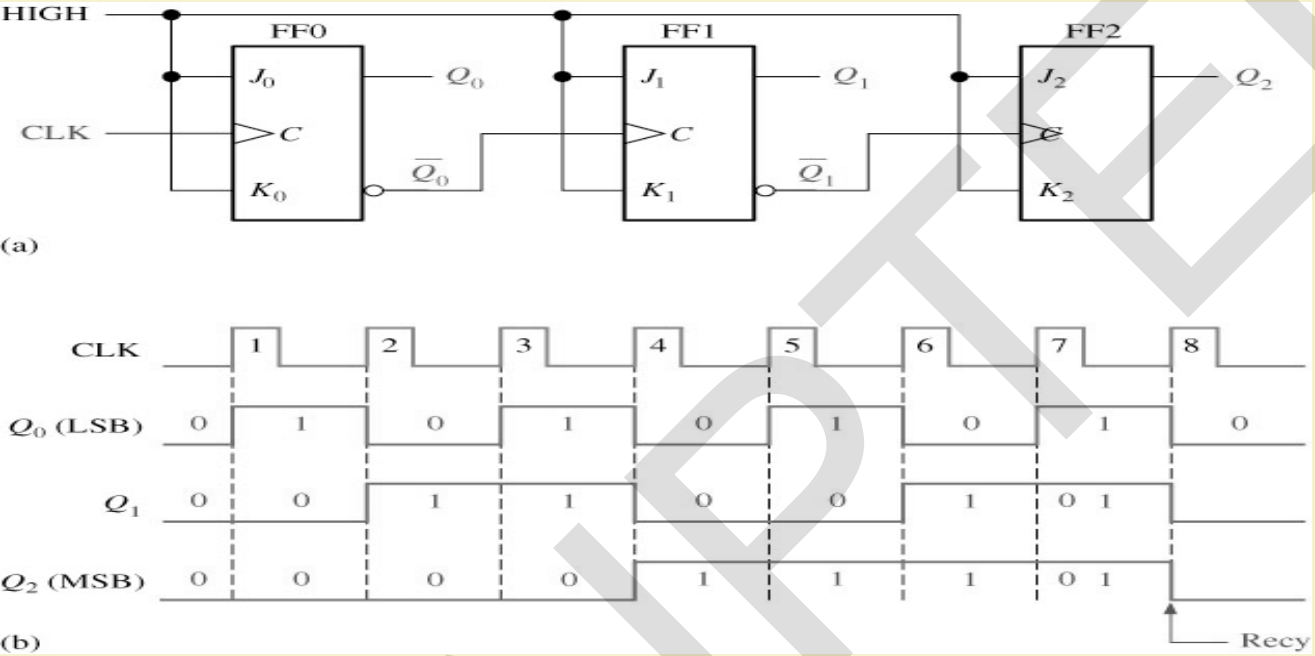
Asynchronous/Ripple Counter

Propagation delays in a 3-bit asynchronous (ripple-clocked) binary counter.



Asynchronous/Ripple Counter

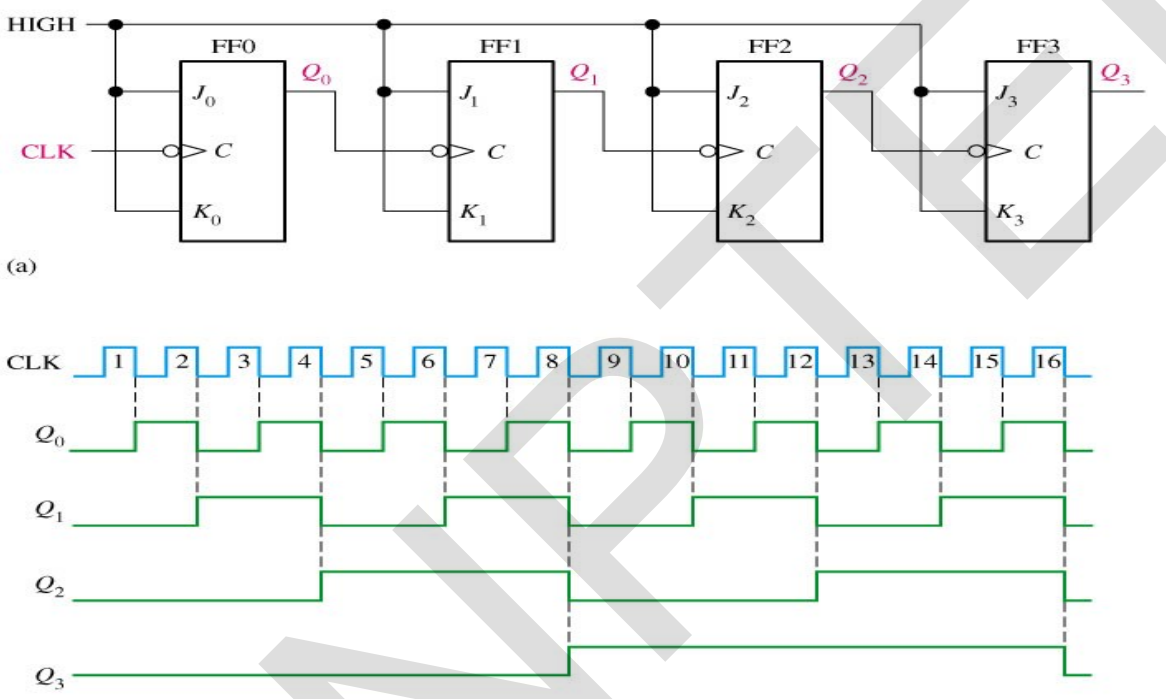
Three-bit asynchronous binary counter and its timing diagram for one cycle.



Clk pulse	Q2	Q1	Q0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (REPEAT)	0	0	0

Asynchronous/Ripple Counter

Four-bit asynchronous binary counter and its timing diagram.



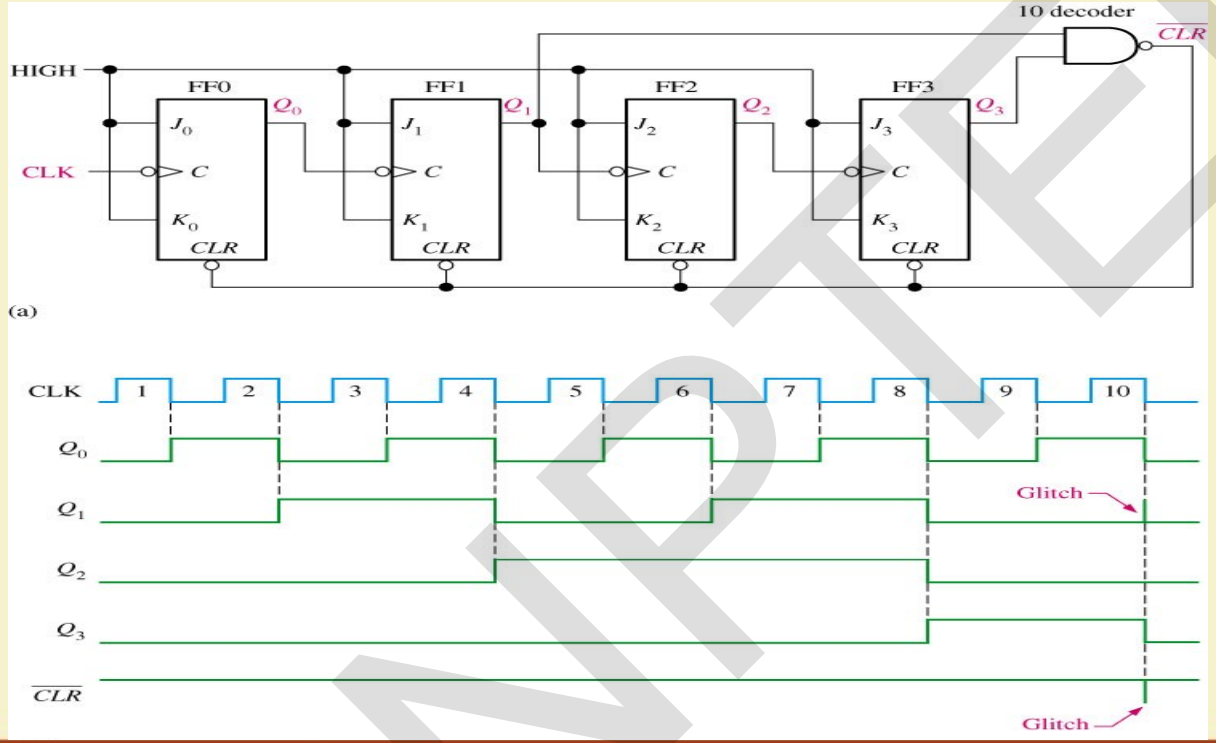
CLK PLUSE	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16 REPEAT	0	0	0	0

Asynchronous Decade Counter

- The Modulus of a counter is the number of unique states that the counter will sequence through.
- Counter can also be designed to have a number of states in their sequence that is less than the maximum of 2^n.
- Counters with the states in their sequence are called decade counters.
- To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states.
- One way to make the counter recycle after the count of nine (1001) is to decode count ten (1010) with a NAND gate and connect the output of the NAND gate to the clear (CLR) inputs of the flip-flops. The inputs the NAND gate are from the Q output from FF1 and FF3 (from 1010 -- FF3FF2FF1FF0)

Asynchronous Decade Counter

An asynchronously clocked decade counter with asynchronous recycling.

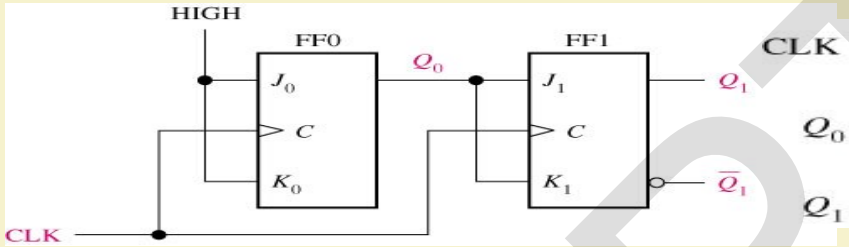


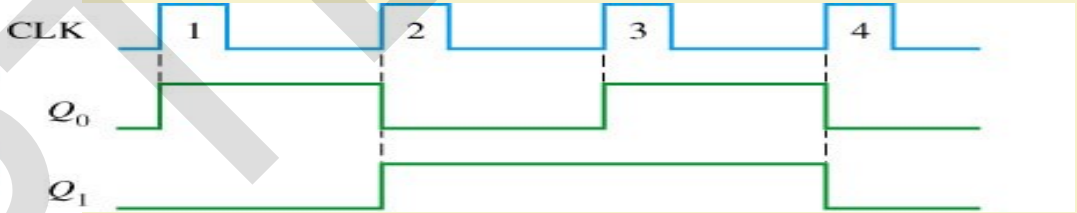
CLK PLUSE	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 GLITCH	0	0	0	0
11	0	0	0	1
12	0	0	1	0
13	0	0	1	1
14	0	1	0	0
15	0	1	0	1
16	0	1	1	0

Synchronous binary Counter

The term Synchronous refers to events that have a fixed time relationship with each other AND receive cllock pulse from a common source

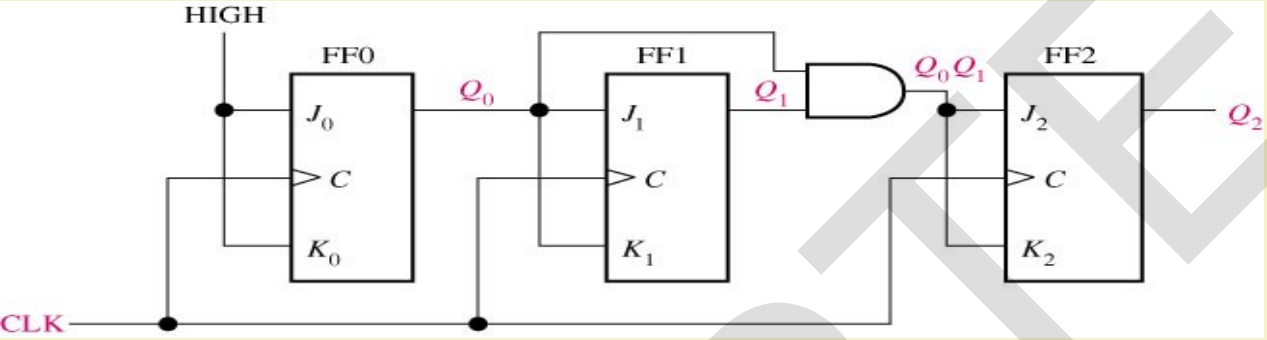
2-bit synchronous binary counter.

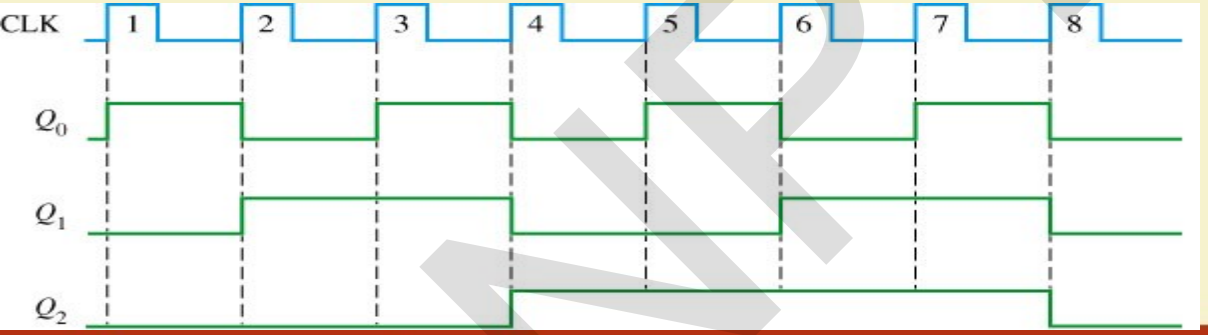




Synchronous binary Counter

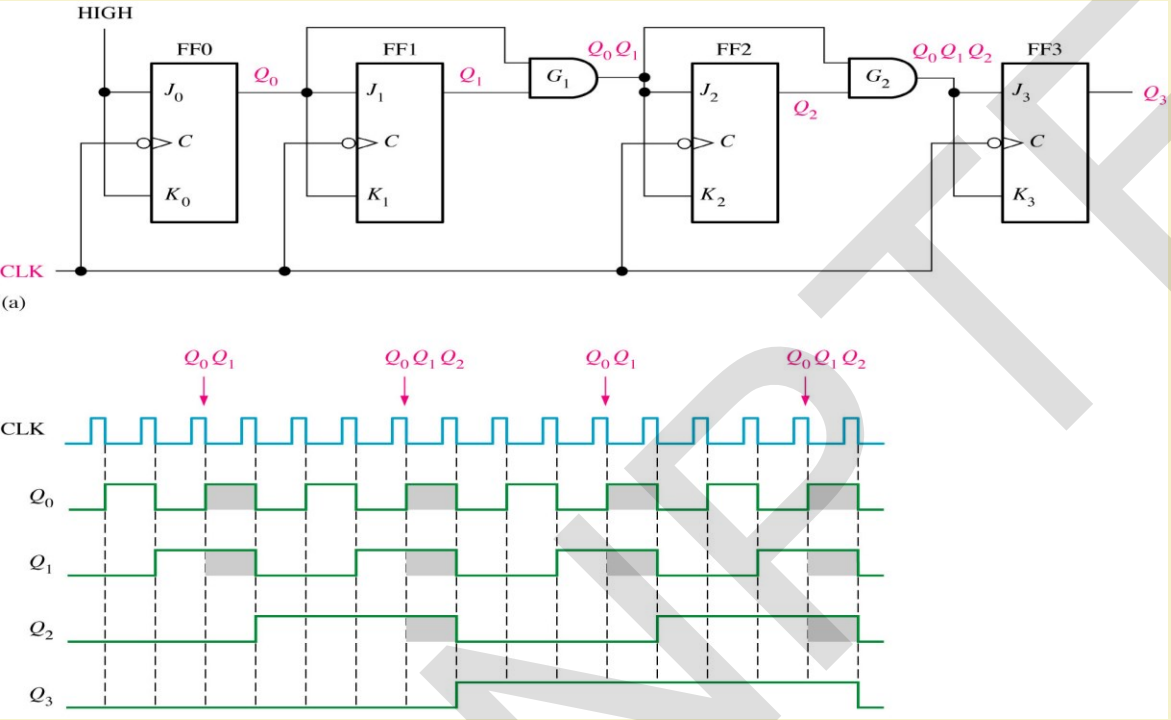
A 3-bit synchronous binary counter.





Clk pulse	Q2	Q1	Q0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (REPEAT)	0	0	0

Synchronous binary Counter



A 4-bit synchronous binary counter and timing diagram. Points where the AND gate outputs are HIGH are indicated by the shaded areas.

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 Counter Design

Several methods are available that follow arbitrary sequence.

Here we will learn one common method using JK flip-Flops.

In synchronous counters all the FF's are clocked at the same time.

J-K Excitation Table

Before begin the designing we must know the operation of the

J-K FF, let us analysis **Truth table for 74LS76 IC (JK flip-flop) and its** excitation table.

JK FF Excitation Table:

PRESENT	**NEXT**	**J**	**K**
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Synchronous Counter Design

J-K Excitation Table

TRANSITION AT OUTPUT	PRESENT STATE Q(N)	NEXT STATE Q(N+1)	J	K
0 → 0	0	0	0	X
0 → 1	0	1	1	X
1 → 0	1	0	X	1
1 → 1	1	1	X	0

0 to 0 TRANSITION; FF's Present status is 0 and it should remain in 0 when a clock pulse is applied. That can be either J=K=0 status or J=0,K=1.

That mean J=0 and K=0 or 1. That is, J=0 and K=X(don't care)

Synchronous Counter Design

0 → 1 TRANSITION: The present state is 0 and it has to change to 1. This can happen either J=1 and K=0 or J=K=1.

That mean always J=1 and K=0 or1

J=1 and K=X(don't care)

1 → 0 TRANSITION; The present state is 1 and it has to change to 0. This can happen either J=0 and K=1 or J=K=1.

That mean always K=1 and J=0 or1

K=1 and J=X(don't care)

1 → 1 TRANSITION; The present state is 1 and it has to change to 1. This can happen either J=K=0 or J=1 and K=0.

That mean always K=0 and J can be either level

K=0 and J=X(don't care)

Synchronous Counter Design

Design Procedure

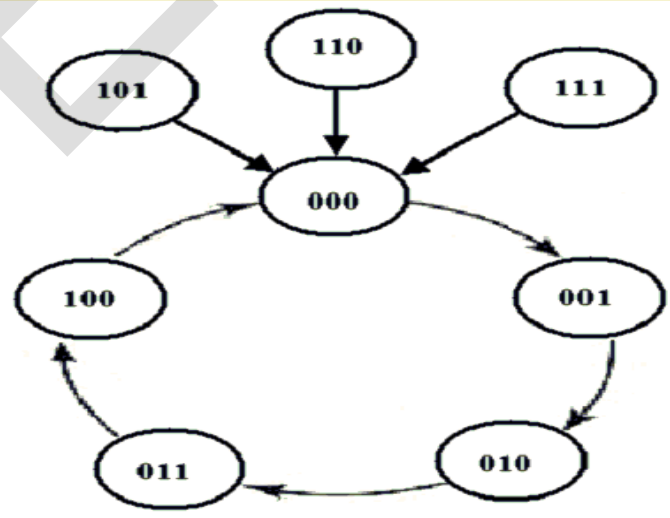
Given a Counter sequence,

C	*B*	*A*
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
0	0	0
0	0	1
etc.		

Synchronous Counter Design / Example (1)

STEP -1

Draw the state transition diagram showing all the possible states, including those that are not part of the desired counting sequence



Synchronous Counter Design / Example (1) ….cont.

STEP -2

Use the state transition diagram to set up a table that lists all **PRESENT** states and their **NEXT** states

	Present state			Next state		
	C	*B*	*A*	*C*	*B*	*A*
1	0	0	0	0	0	1
2	0	0	1	0	1	0
3	0	1	0	0	1	1
4	0	1	1	1	0	0
5	1	0	0	0	0	0
6	1	0	1	0	0	0
7	1	1	0	0	0	0
8	1	1	1	0	0	0

Synchronous Counter Design / Example (1) ….cont.

STEP -3

Add a column to this table for each J and K input. For each **PRESENT** state, indicate the level required at each J and K input in order to produce the transition to the **NEXT** state.

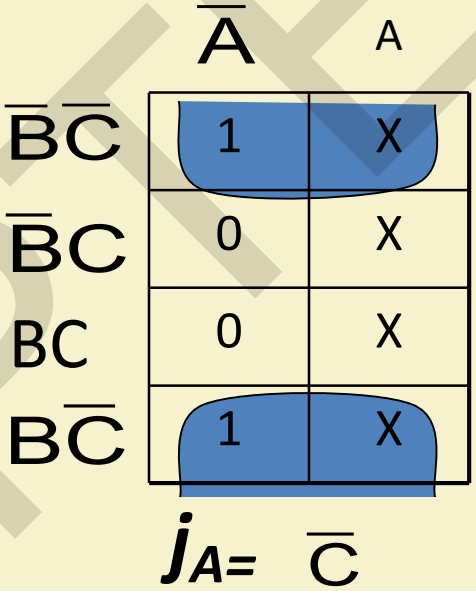
	Present state			Next state								
	C	B	A	C	B	A	j_C	k_C	j_B	k_B	j_A	k_A
1	0	0	0	0	0	1	0	X	0	X	1	X
2	0	0	1	0	1	0	0	X	1	X	X	1
3	0	1	0	0	1	1	0	X	X	0	1	X
4	0	1	1	1	0	0	1	X	X	1	X	1
5	1	0	0	0	0	0	X	1	0	X	0	X
6	1	0	1	0	0	0	X	1	0	X	X	1
7	1	1	0	0	0	0	X	1	X	1	0	X
8	1	1	1	0	0	0	X	1	X	1	X	1

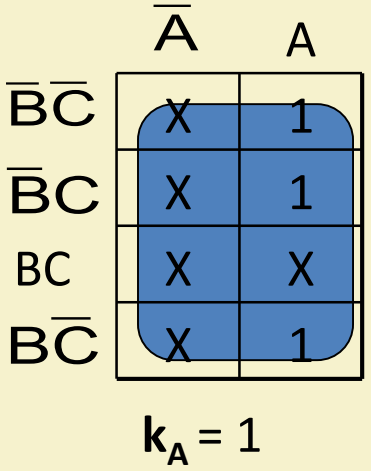
Synchronous Counter Design / Example (1) ….cont.

STEP- 4

Design the logic expression to generate the level required at each J and K, using K-maps.

Present state				
C	*B*	*A*	j_A	k_A
0	0	0	1	X
0	0	1	X	1
0	1	0	1	X
0	1	1	X	1
1	0	0	0	X
1	0	1	X	1
1	1	0	0	X
1	1	1	X	1

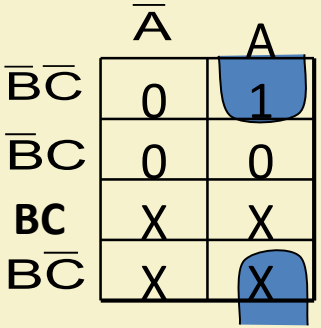




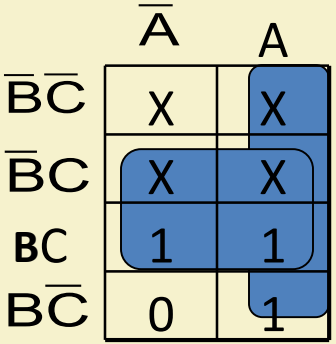
Synchronous Counter Design / Example (1) ….cont.

STEP- 4 …..cont.

Present state				
C	*B*	*A*	*jB*	*kB*
0	0	0	0	X
0	0	1	1	X
0	1	0	X	0
0	1	1	X	1
1	0	0	X	X
1	0	1	X	X
1	1	0	X	1
1	1	1	X	1



$j_B = A\,\bar{C}$



$k_B = A+C$

Synchronous Counter Design / Example (1) ….cont.

STEP- 4

…..cont.

Present state				
C	*B*	*A*	*jC*	*kC*
0	0	0	0	X
0	0	1	0	X
0	1	0	0	X
0	1	1	1	X
1	0	0	0	1
1	0	1	X	1
1	1	0	0	1
1	1	1	X	1

	$\bar{A}$	A
$\bar{B}\bar{C}$	0	0
$\bar{B}C$	X	X
BC	X	X
$B\bar{C}$	0	1

$j_C = AB$

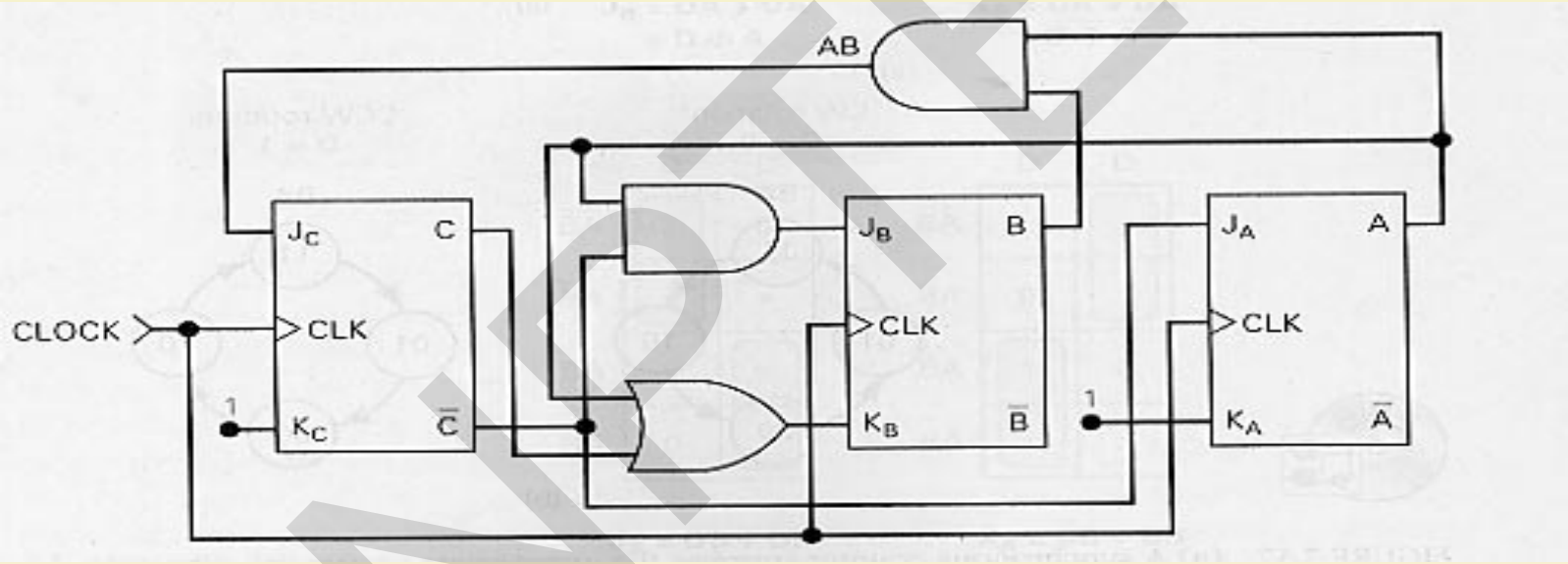
	$\bar{A}$	A
$\bar{B}\bar{C}$	X	X
$\bar{B}C$	1	1
BC	1	1
$B\bar{C}$	X	X

$k_C = 1$

Synchronous Counter Design / Example (1) ….cont.

SETP -5

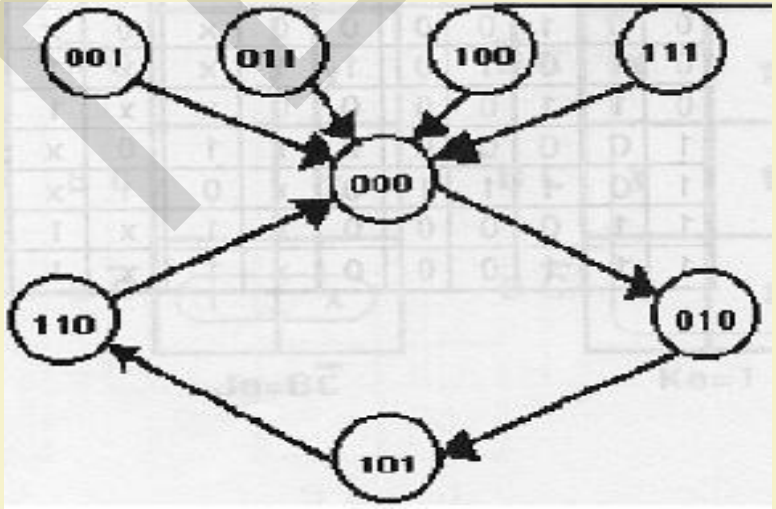
Finally to implement the final expressions.



Synchronous Counter Design / Example (2)

Design a JK synchronous counter that has the following sequence:000,010,101,110 and repeat. The undesired states 001,011,100 and 111 must always go to 000 on the next clock pulse.

STEP -1 :State Transition Diagram



Synchronous Counter Design / Example (2) ….cont.

STEP- 2 : Table to list PRESENT and NEXT status

PRESENT State			NEXT State		
C	B	A	C	B	A
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	0	0	0

Synchronous Counter Design / Example (2) ….cont.

STEP- 3 : Table indicate the Level required at each J and K inputs in order to produce the transition to the NEXT

Present State			Next State								
C	B	A	C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	1	0	0	x	1	x	0	x
0	0	1	0	0	0	0	x	0	x	x	1
0	1	0	1	0	1	1	x	x	1	1	x
0	1	1	0	0	0	0	x	x	1	x	1
1	0	0	0	0	0	x	1	0	x	0	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	0	0	0	x	1	x	1	0	x
1	1	1	0	0	0	x	1	x	1	x	1

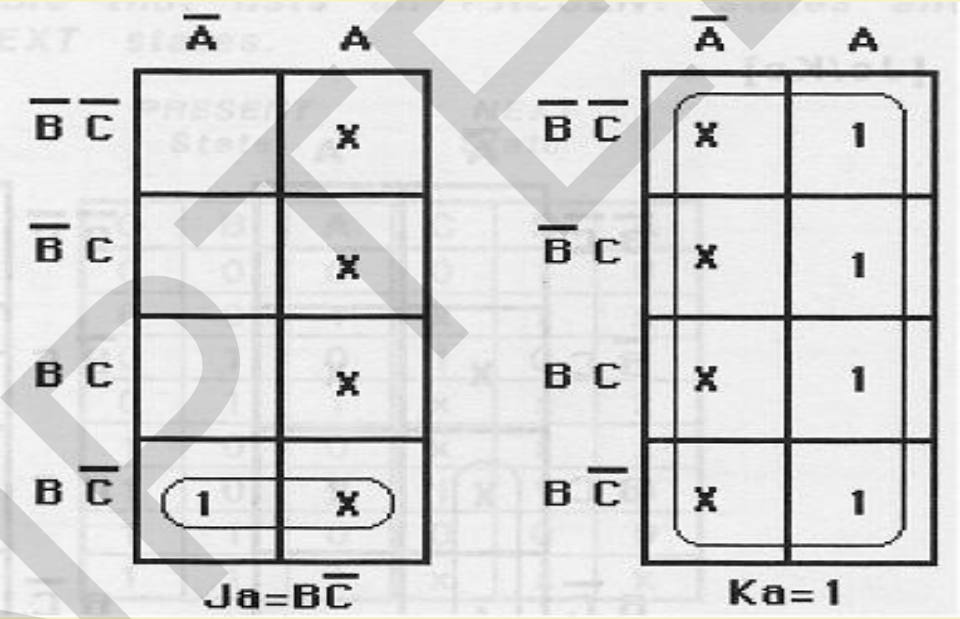
Synchronous Counter Design / Example (2) ….cont.

STEP- 4 :Design the logic circuits to generate the levels required at each J and K inputs

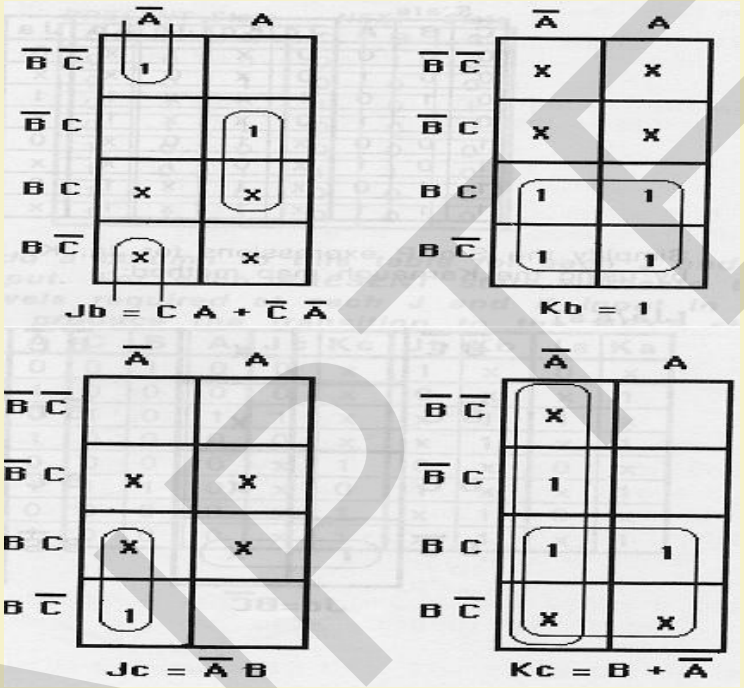
Present State								
C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	x	1	x	0	x
0	0	1	0	x	0	x	x	1
0	1	0	1	x	x	1	1	x
0	1	1	0	x	x	1	x	1
1	0	0	x	1	0	x	0	x
1	0	1	x	0	1	x	x	1
1	1	0	x	1	x	1	0	x
1	1	1	x	1	x	1	x	1

Synchronous Counter Design / Example (2) ….cont.

STEP- 5 :Simplify the SOP expression using K-maps



Synchronous Counter Design / Example (2) ….cont.

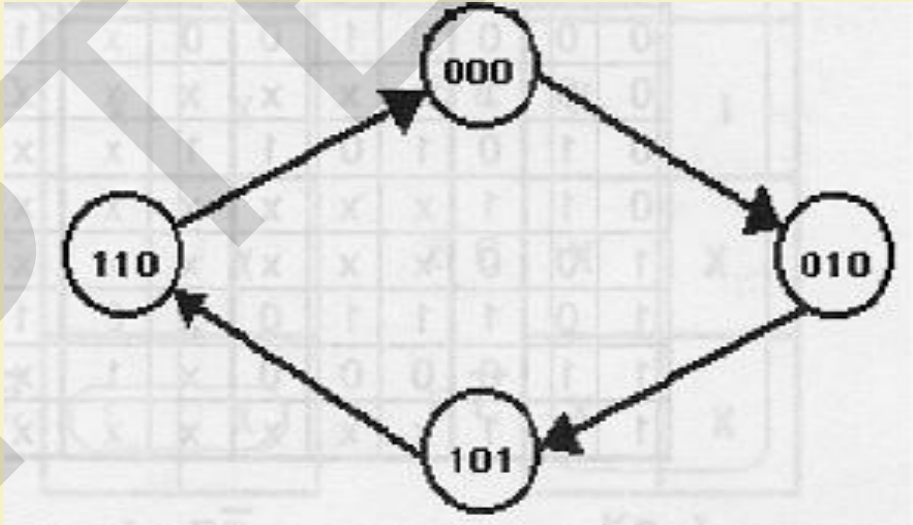


Synchronous Counter Design / Example (3)

?

Design a JK synchronous counter that has the following sequence:000,010,101,110 and repeat. For undesired states their NEXT states can be DON'T CARES.

STEP -1 :State Transition Diagram



Synchronous Counter Design / Example (3) ….cont.

STEP- 2 : Table to list PRESENT and NEXT status

PRESENT State			NEXT State		
C	B	A	C	B	A
0	0	0	0	1	0
0	0	1	x	x	x
0	1	0	1	0	1
0	1	1	x	x	x
1	0	0	x	x	x
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	x	x	x

Synchronous Counter Design / Example (3) ….cont.

STEP- 3 : Table indicate the Level required at each J and K inputs in order to produce the transition to the NEXT

Present State			Next State								
C	B	A	C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	1	0	0	x	1	x	0	x
0	0	1	x	x	x	x	x	x	x	x	x
0	1	0	1	0	1	1	x	x	1	1	x
0	1	1	x	x	x	x	x	x	x	x	x
1	0	0	x	x	x	x	x	x	x	x	x
1	0	1	1	1	0	x	0	1	x	x	1
1	1	0	0	0	0	x	1	x	1	0	x
1	1	1	x	x	x	x	x	x	x	x	x

Synchronous Counter Design / Example (3) ….cont.

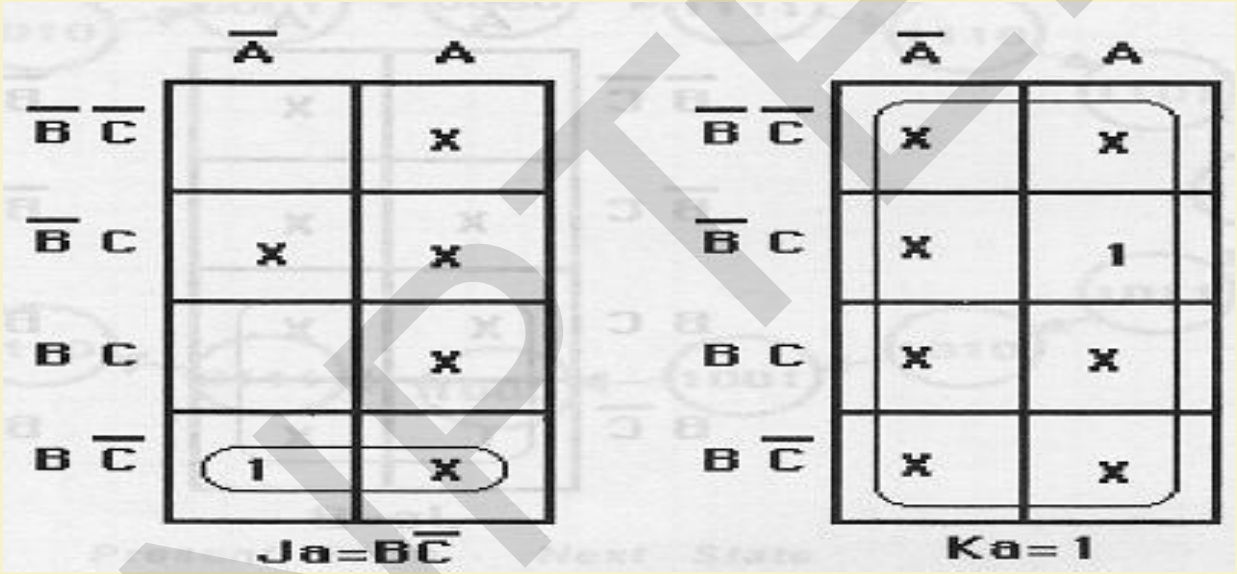
STEP- 4 :Design the logic circuits to generate the levels required at each J and K inputs

Present State

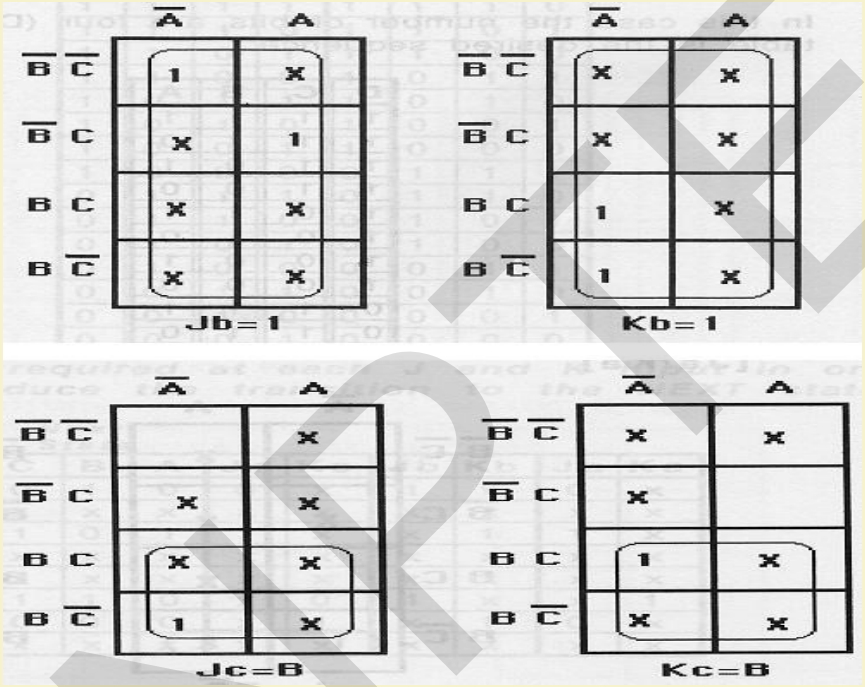
C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	x	1	x	0	x
0	0	1	x	x	x	x	x	x
0	1	0	1	x	x	1	1	x
0	1	1	x	x	x	x	x	x
1	0	0	x	x	x	x	x	x
1	0	1	x	0	1	x	x	1
1	1	0	x	1	x	1	0	x
1	1	1	x	x	x	x	x	x

Synchronous Counter Design / Example (3) ….cont.

STEP- 5 :Simplify the SOP expression using K-maps



Synchronous Counter Design / Example (3) ….cont.



Synchronous Counter Design / Example (4) ….cont.

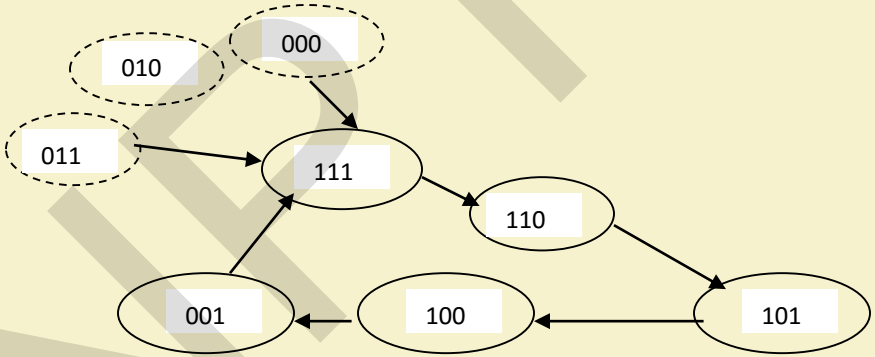
Objective:

To design a 3 bit counter (D FF) with the following count sequence 7,6,5,4,1. All unwanted stages go to 7.

Output sequence 7,6,5,4,1

In 3 bits format: 111,110, 101, 100, 001

State transition diagram:



Synchronous Counter Design / Example (4) ….cont.

D Flip Flop Excitation Table:

PRESENT	NEXT	D	
0	0	0	
0	1	1	
1	0	0	
1	1	1	

Synchronous Counter Design / Example (4) ….cont.

OUTPUT						INPUT		
PRESENT STATE			NEXT STATE			C	B	A
C	B	A	C	B	A	D_C	D_B	D_A
0	0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1
1	0	0	0	0	1	0	0	1
1	0	1	1	0	0	1	0	0
1	1	0	1	0	1	1	0	1
1	1	1	1	1	0	1	1	0

Synchronous Counter Design / Example (4) ….cont.

K- Map

	$\overline{A}$	A
$\overline{C}\overline{B}$	1	1
$\overline{C}B$	1	1
CB	1	1
$C\overline{B}$	0	1
	D_C =A+ C' + B	

	$\overline{A}$	A
$\overline{C}\overline{B}$	1	1
$\overline{C}B$	1	1
CB	0	1
$C\overline{B}$	0	0
	D_B = AB + C'	

	$\overline{A}$	A
$\overline{C}\overline{B}$	1	1
$\overline{C}B$	1	1
CB	1	0
$C\overline{B}$	1	0
	D_A = A' + C'	

Synchronous Counter Design (T Flip Flop based design / Example (5) ….cont.

T Flip Flop Excitation Table:

PRESENT	**NEXT**	**T**	
0	0	0	
0	1	1	
1	0	1	
1	1	0	

Synchronous Counter Design / Example (5) ….cont.

T Flip Flop Input Function Table

OUTPUT						INPUT		
PRESENT STATE			NEXT STATE			C	B	A
C	B	A	C	B	A	T_C	T_B	T_A
0	0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1	0
0	1	0	1	1	1	1	0	1
0	1	1	1	1	1	1	0	0
1	0	0	0	0	1	1	0	1
1	0	1	1	0	0	0	0	1
1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	0	0	1

Synchronous Counter Design / Example (5) ….cont.

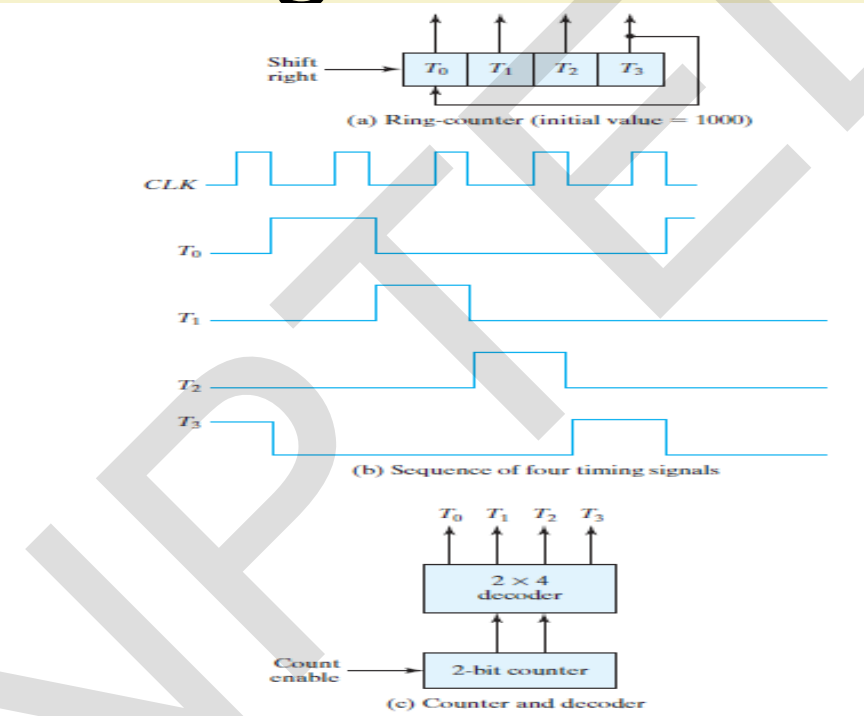
K- Map

	$\overline{A}$	A
$\overline{C}\overline{B}$	1	1
$\overline{C}B$	1	1
CB	0	0
$C\overline{B}$	1	0
	T_C =A'B' + C'	

	$\overline{A}$	A
$\overline{C}\overline{B}$	1	1
$\overline{C}B$	0	0
CB	1	0
$C\overline{B}$	0	0
	T_B = B'C' + A'BC	

	$\overline{A}$	A
$\overline{C}\overline{B}$	1	0
$\overline{C}B$	1	0
CB	1	1
$C\overline{B}$	1	1
	T_A = A' + C	

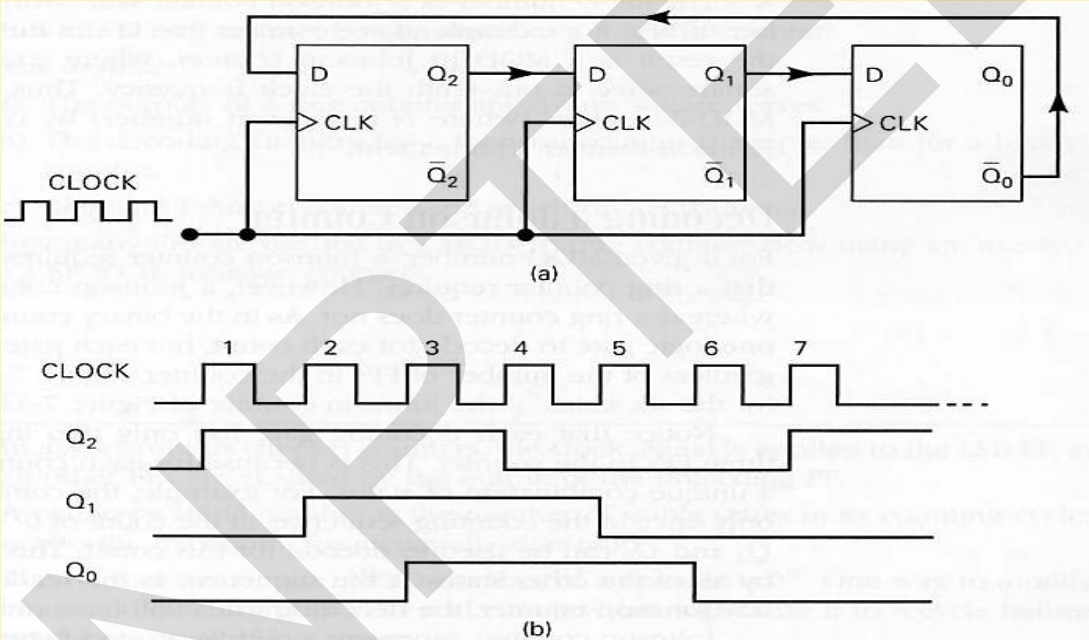
Ring Counter



Johnson Counter

- Also known as the twisted-ring counter.
- Same as the ring counter except that the inverted output of the last FF is connected to the input of the first FF.
- Counting sequence:
  000→100→110→111→011→001→000
- A MOD-6 counter (twice the number of FFs)

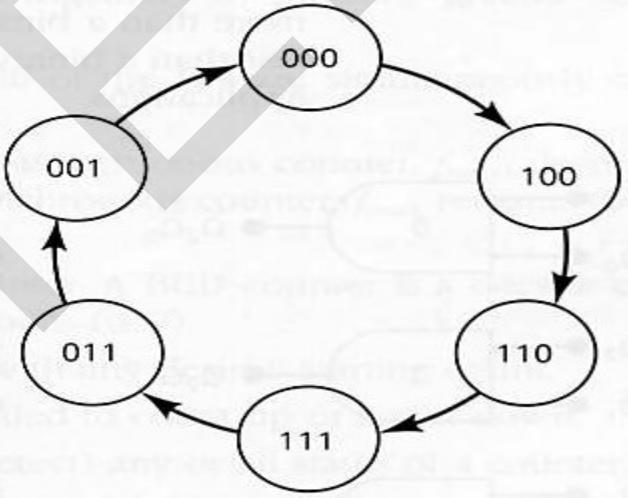
MOD-6 Johnson Counter



State Transition Diagram

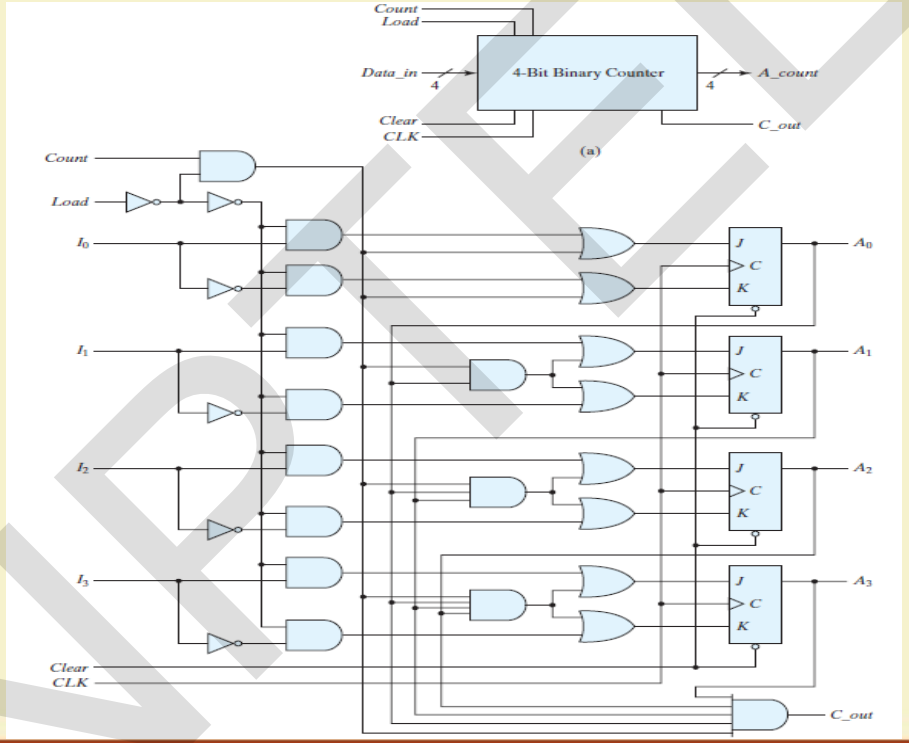
Q_2	Q_1	Q_0	CLOCK pulse
0	0	0	0
1	0	0	1
1	1	0	2
1	1	1	3
0	1	1	4
0	0	1	5
0	0	0	6
1	0	0	7
1	1	0	8
.	.	.	.
.	.	.	.
.	.	.	.

(c)



(d)

Binary Counter with Parallel Load



Finite State Machine

Santanu Chattopadhyay
Electronics and Electrical Communication Engineering

Finite State Machine

- An electronic machine which has
  - external inputs
  - externally visible outputs
  - internal state
- Output and next state depend on
  - inputs
  - current state

Abstract Model of FSM

Machine is $M = (S, I, O, \delta)$

S: Finite set of states

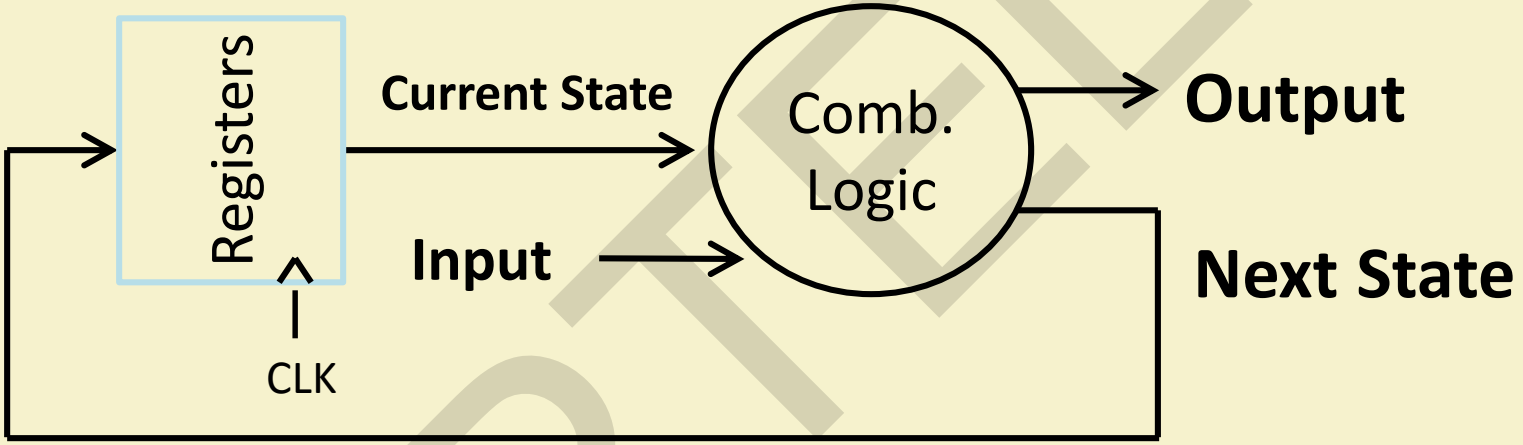
I: Finite set of inputs

O: Finite set of outputs

δ: State transition function

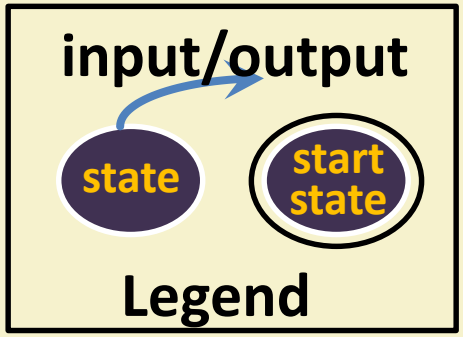
- Next state depends on present input *and* present state

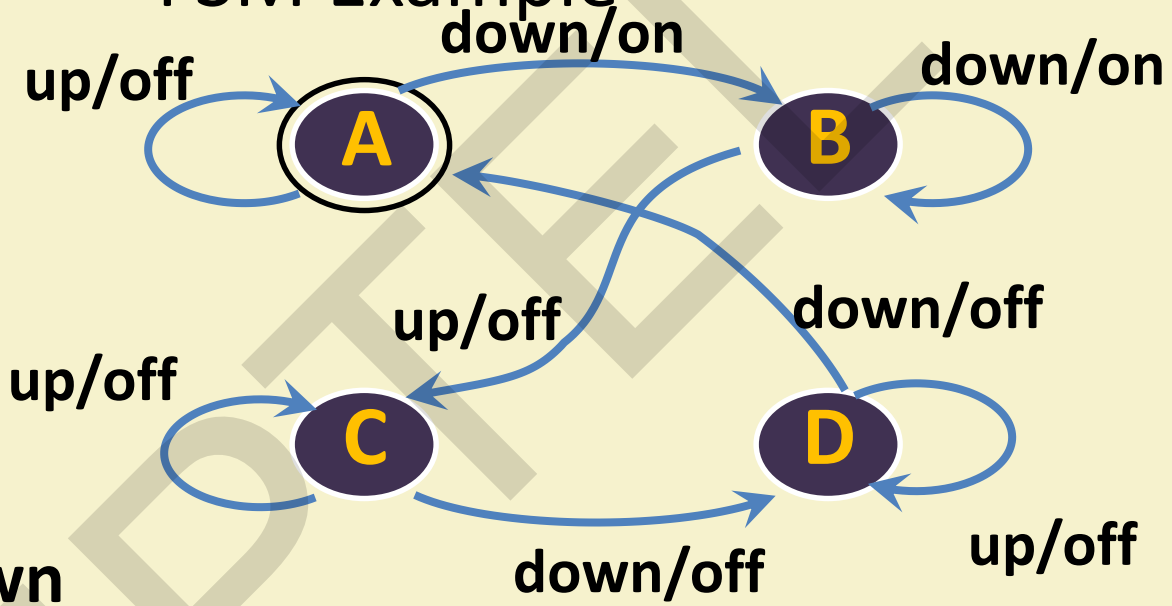
Automata Model



- inputs from external world
- outputs to external world
- internal state
- combinational logic

FSM Example



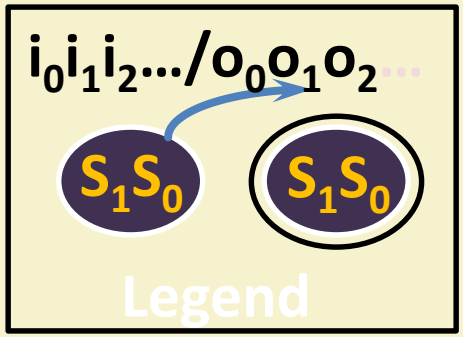


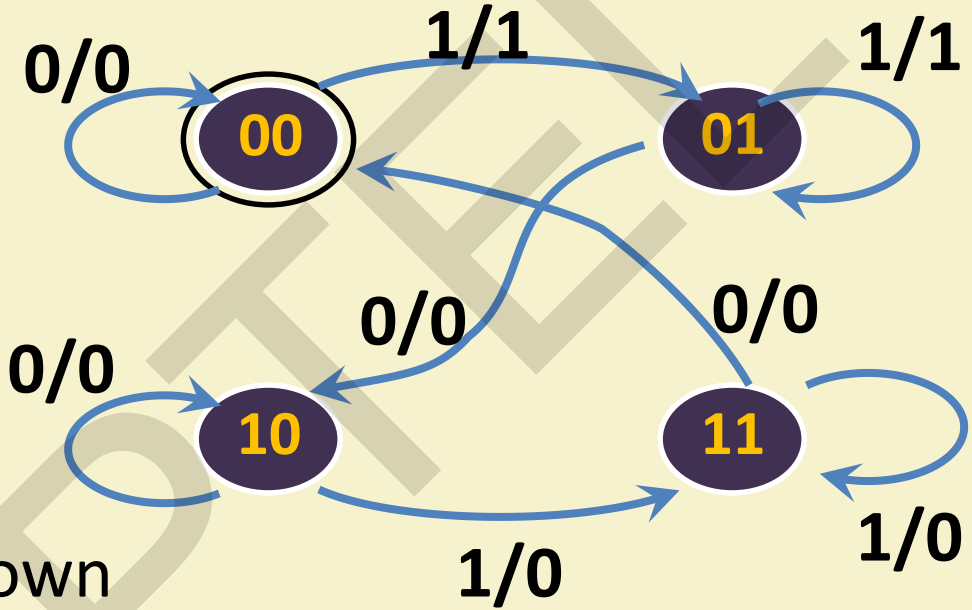
Input: **up** or **down**

Output: **on** or **off**

States: A, B, C, or D

FSM Example



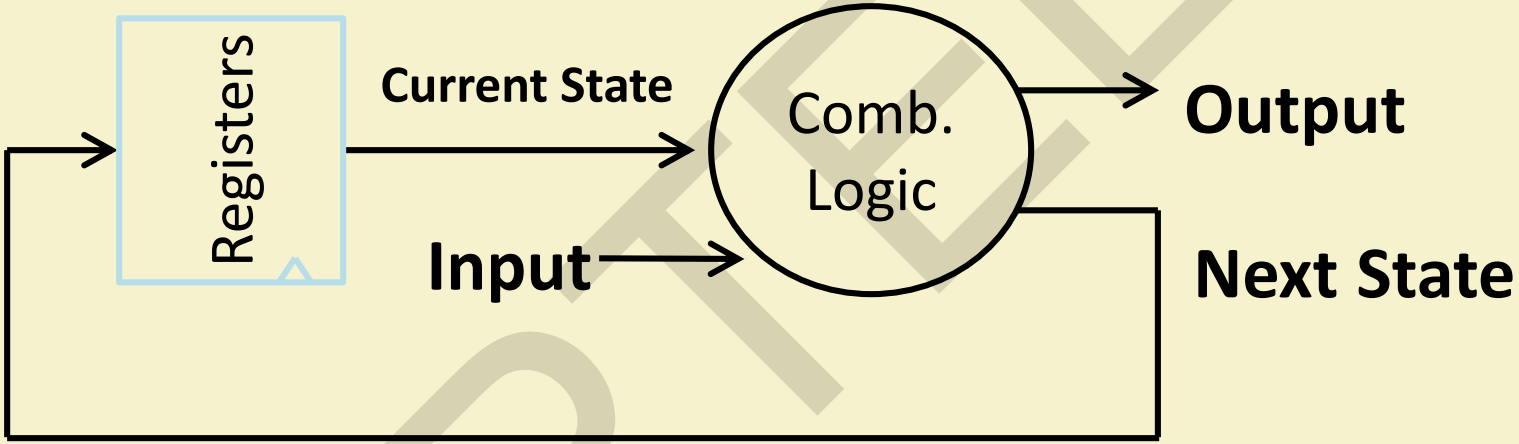


Input: **0**=up or **1**=down

Output: **1**=on or **0**=off

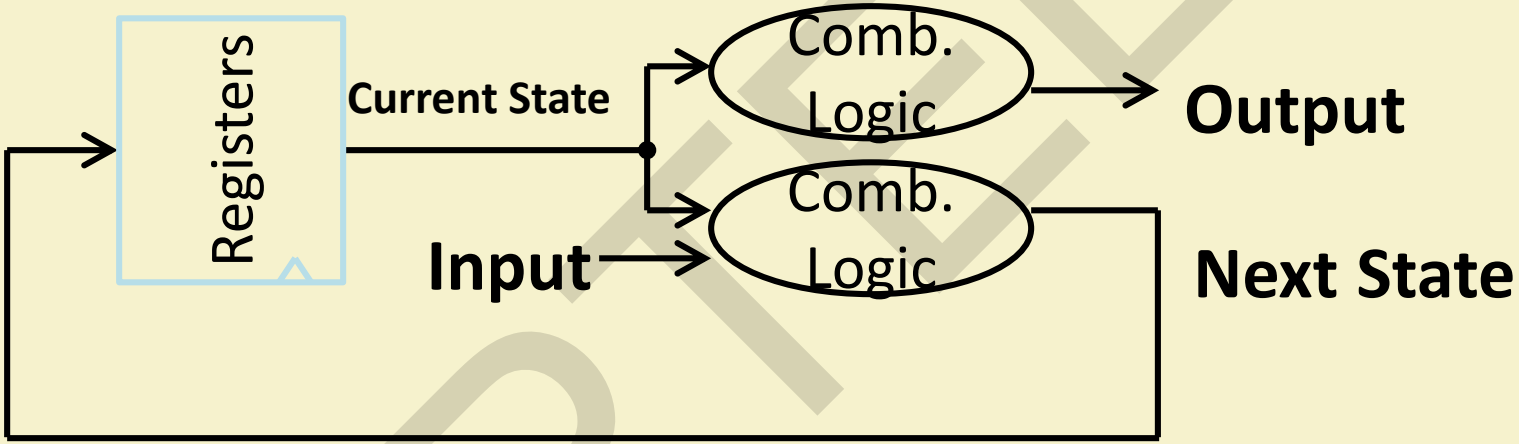
States: 00=A, 01=B, 10=C, or 11=D

Mealy Machine



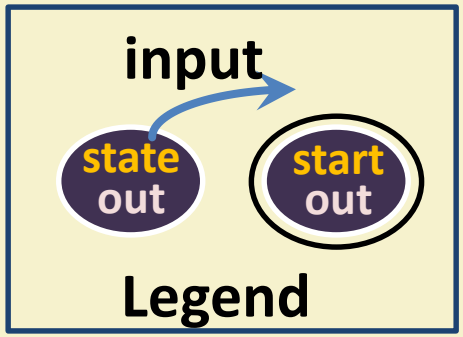
- Outputs and next state depend on both current state and input

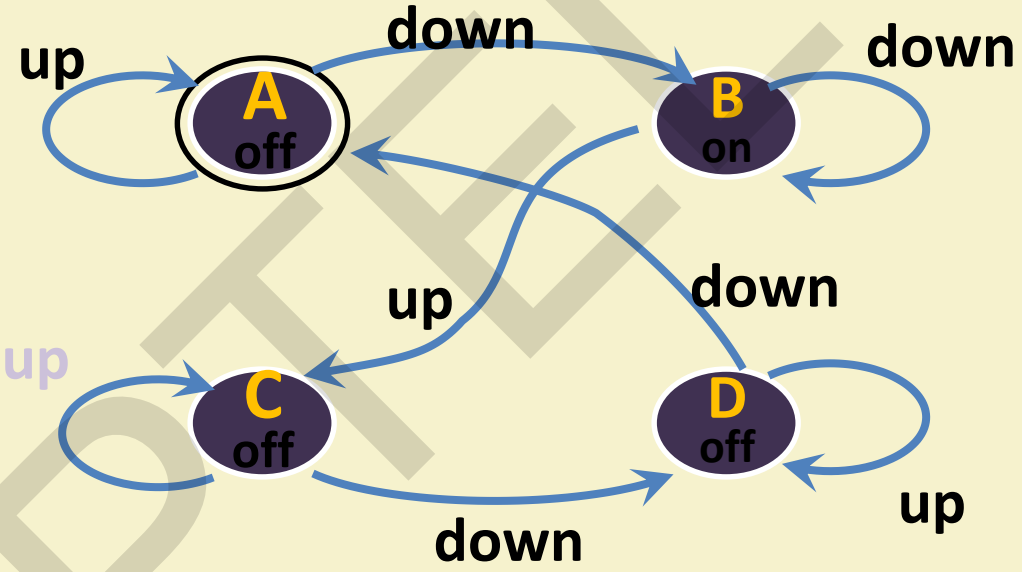
Moore Machine



- Outputs depend only on current state

Moore Machine FSM Example



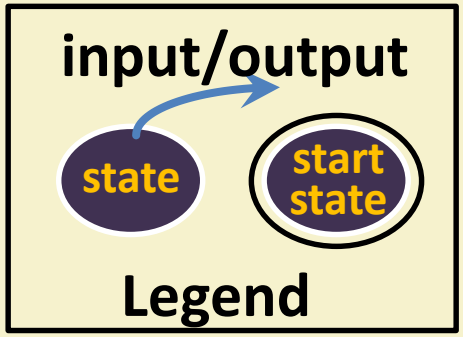


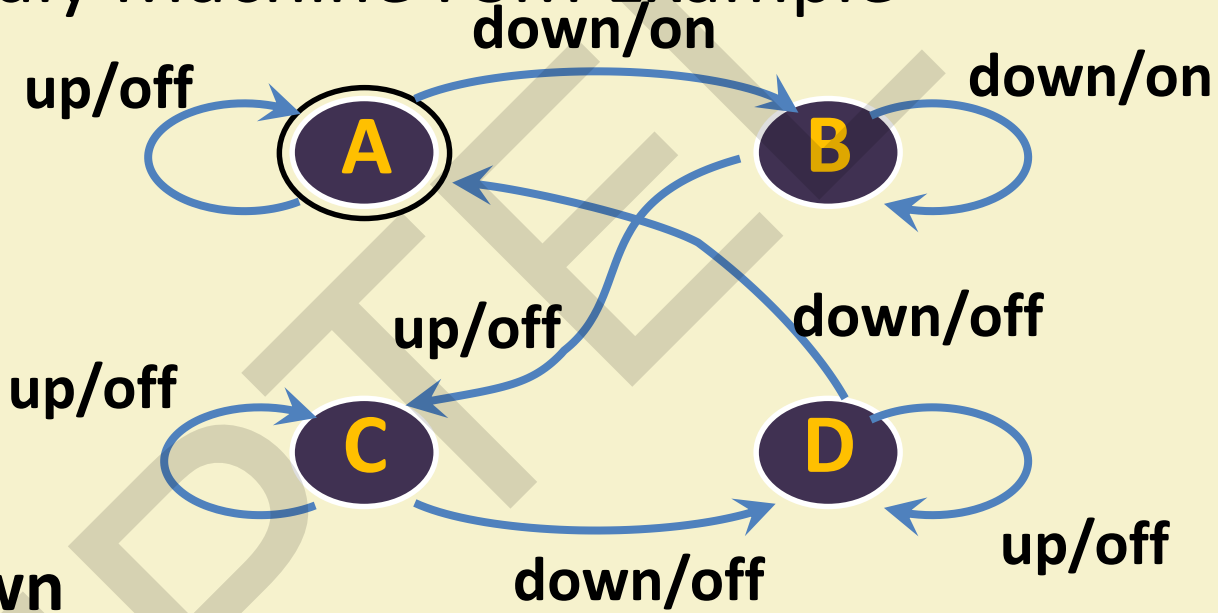
Input: **up** or **down**

Output: **on** or **off**

States: A, B, C, or D

Mealy Machine FSM Example





Input: **up** or **down**

Output: **on** or **off**

States: A, B, C, or D

Create a Logic Circuit for a Serial Adder

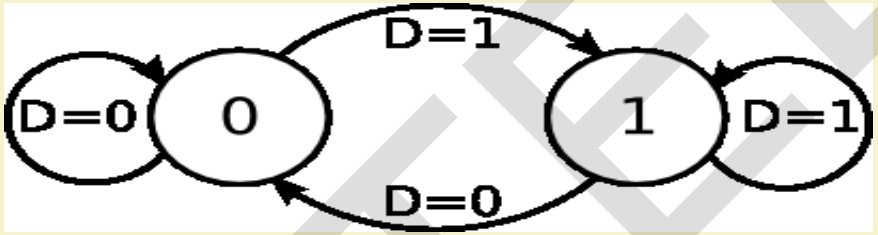
- Add two infinite input bit streams
  - streams are sent with least-significant-bit (lsb) first

  - How many states are needed to represent FSM?
  - Draw and Fill in FSM diagram

Strategy:

(1) Draw a state diagram (e.g. Mealy Machine)
(2) Write output and next-state tables
(3) Encode states, inputs, and outputs as bits
(4) Determine logic equations for next state and outputs

Another look at D latch/flip-flop

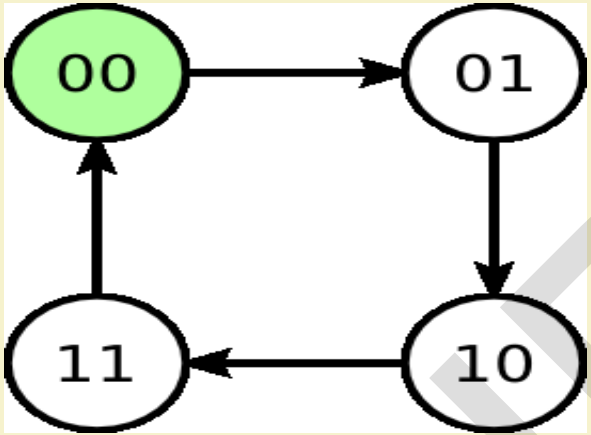


q_{old}	D	q_{new}
0	0	0
0	1	1
1	0	0
1	1	1

This is an example of a **state diagram**
more specifically a **Moore** machine

$q_{new} = D$

Another example - 2-bit counter



Counter starts at 0 (green) and increments each time the clock cycles, until it gets to 3 and then overflows back to 0.

Only input is the clock, we don't show that.

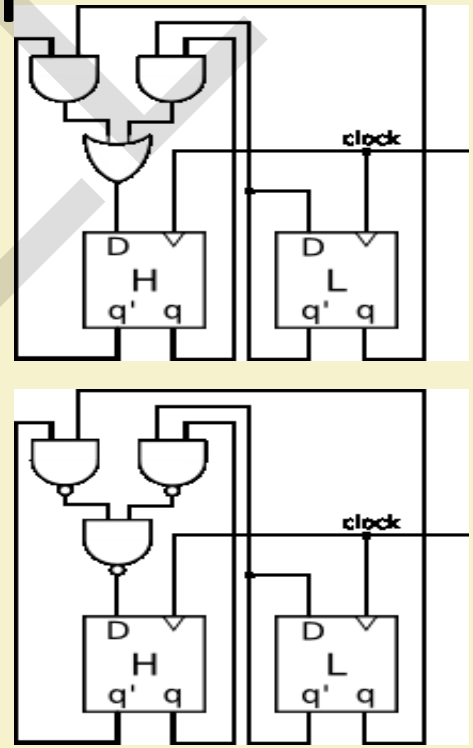
H_{old}	L_{old}	H_{new}	L_{new}
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

2-bit counter

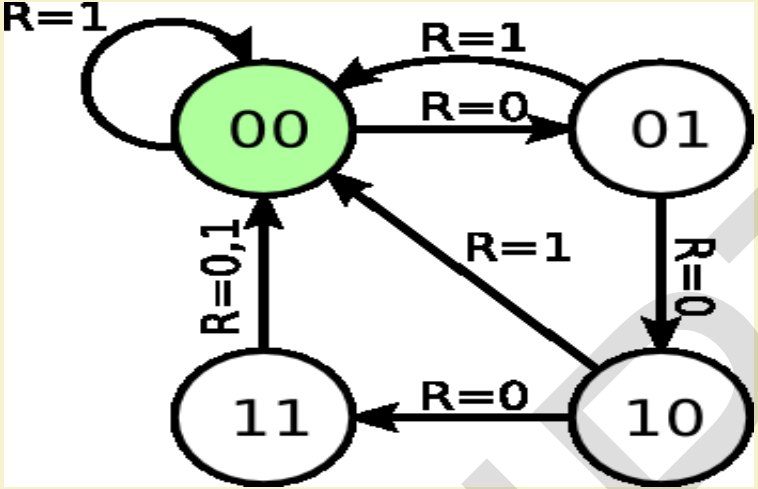
H_{old}	L_{old}	H_{new}	L_{new}
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

$L_{new} = H_{old}'L_{old}' + H_{old}L_{old}' = L_{old}'$

$H_{new} = H_{old}'L_{old} + H_{old}L_{old}'$



2-bit counter with reset

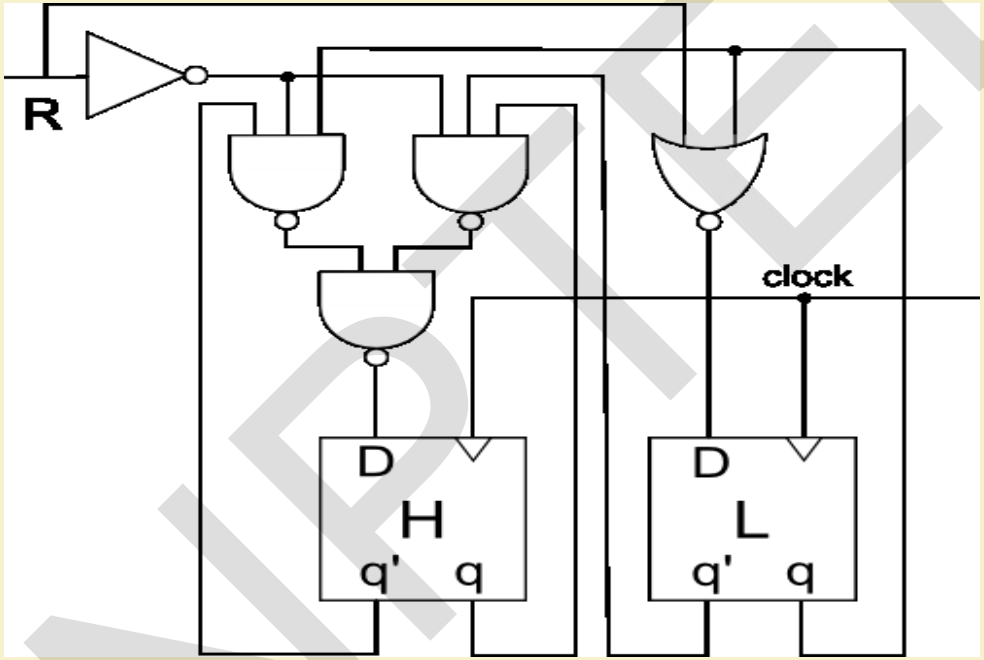


R	H_{old}	L_{old}	H_{new}	L_{new}
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	x	x	0	0

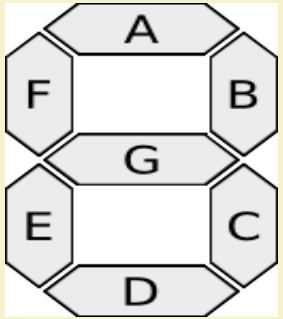
$L_{new} = R'H_{old}'L_{old}' + R'H_{old}L_{old}'$
$= R'L_{old}' = (R + L_{old})'$

$H_{new} = R'H_{old}'L_{old} + R'H_{old}L_{old}'$
$= R'(H_{old}'L_{old} + H_{old}L_{old}')$

2-bit counter with reset

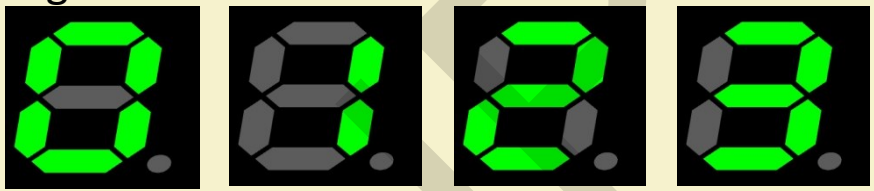


Counter with 7-segment display



Each segment in the display can be lit independently to allow all 10 decimal digits to be displayed (also hex)

2-bit counter will need to display digits 0-3, so will output a 1 for each segment to be lit for a given state



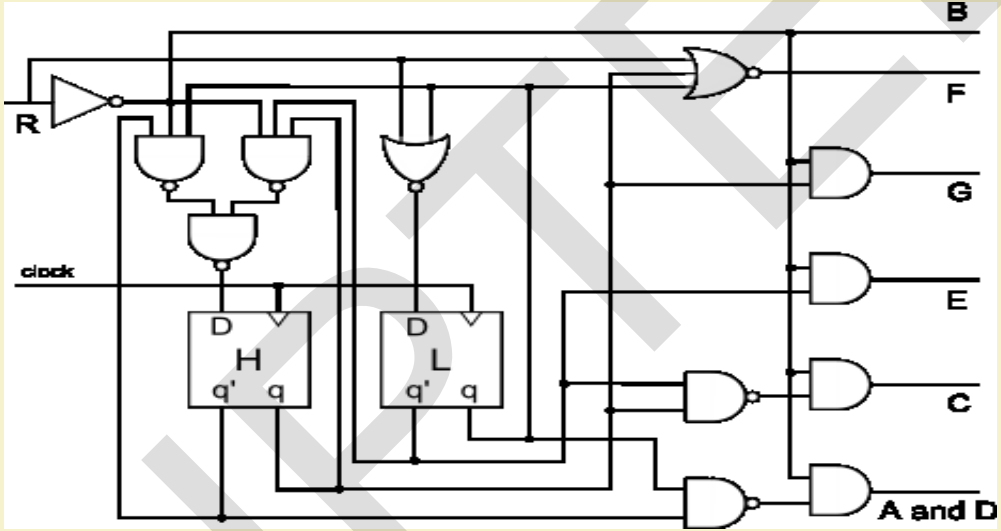
Counter with output functions

R	H_o	L_o	H_n	L_n	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	0	0	1	1	0	0	0	0
0	1	0	1	1	1	1	0	1	1	0	1
0	1	1	0	0	1	1	1	1	0	0	1
1	x	x	0	0	0	0	0	0	0	0	0

$A = D = R'H_o'L_o' + R'H_oL_o' + R'H_oL_o = R'(H_o'L_o)'$ $B = R'$ $C = R'(H_oL_o')'$

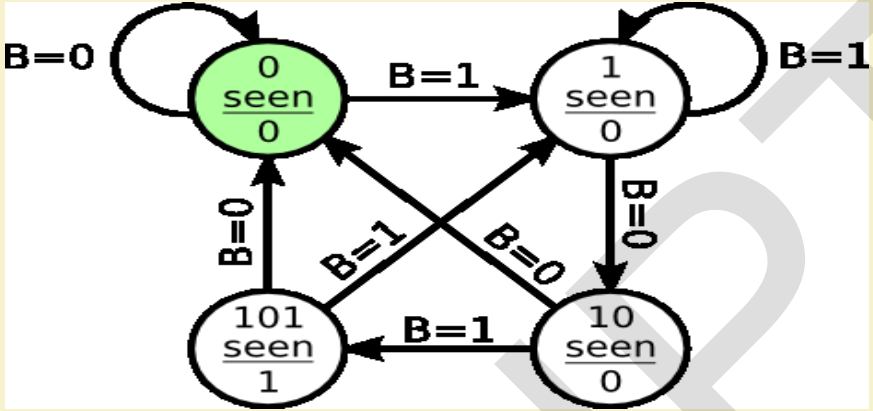
$E = R'L_o'$ $F = R'H_o'L_o' = (R+H_o+L_o)'$ $G = R'H_o$

7-segment output logic



Example - 101 lock

Combination lock with 101 being the combination



B is input signal to the lock,
X is output signal to unlock

B	H_o	L_o	H_n	L_n	X
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	0	1	0

101 combination lock

$X = H_oL_o$

$H_n = B'H_o'L_o + BH_oL_o'$

$L_n = BH_o'L_o + BH_oL_o' + BH_oL_o$
$= BH_o'L_o + BH_oL_o + BH_oL_o' + BH_oL_o$
$= BL_o + BH_o$