CHW 261: Logic Design

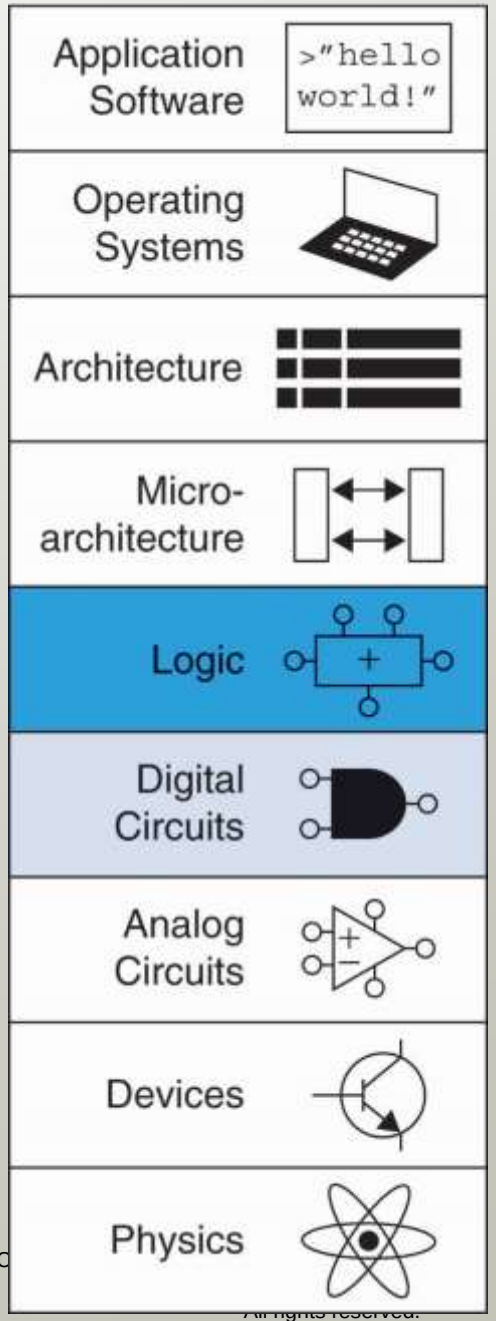
Instructors:
Prof. Hala Zayed http://www.bu.edu.eg/staff/halazayed14
Dr. Ahmed Shalaby http://bu.edu.eg/staff/ahmedshalaby14#

Digital Fundamentals

Digital Concepts

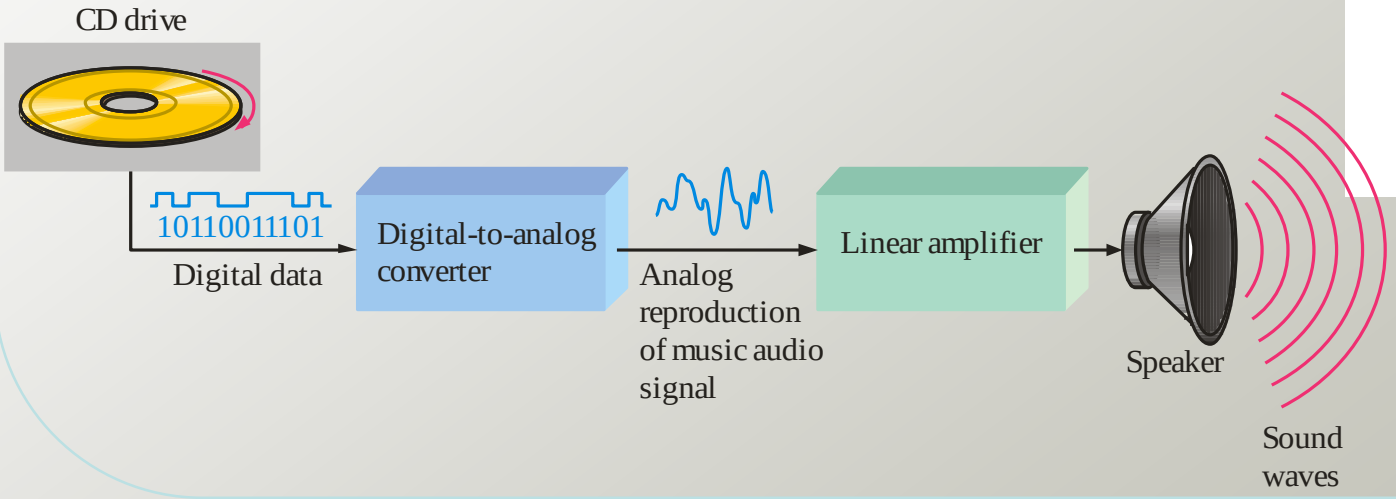
What ? Logic Design

- **Logic Design** defines the fundamentals of Digital systems, such as computers and cell phones.

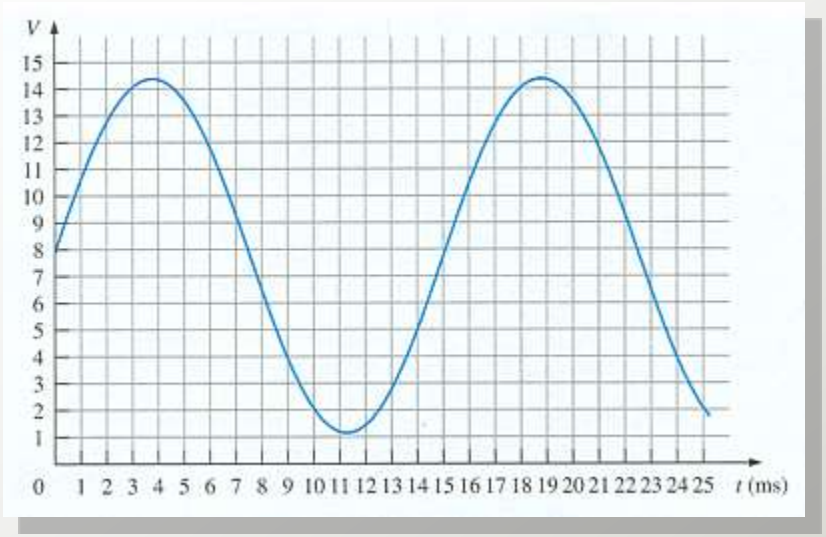


Digital System (Why)

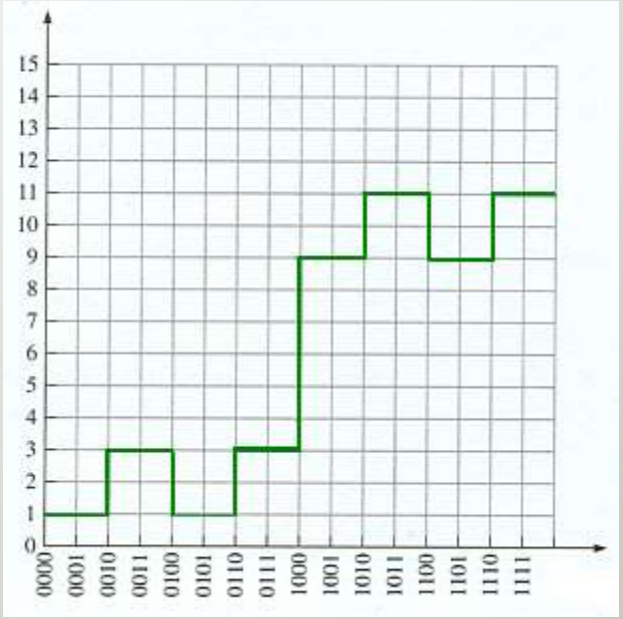
- Easier to design.
- Flexibility and functionality.
  easier to store, transmit and manipulate information.
- Cheaper device.



Digital and Analog Quantities



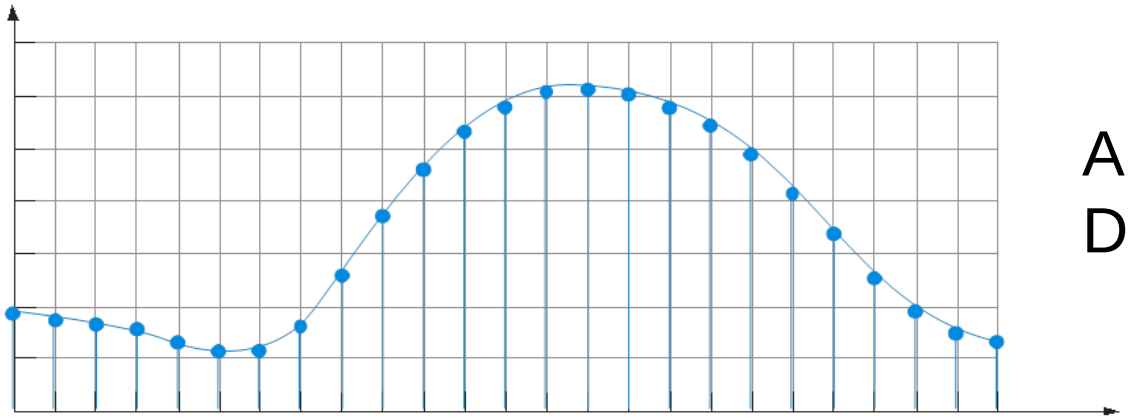
Analog quantities have continuous values



Digital quantities have discrete sets of values

Digital System (Why)
Analog vs. Digital

Most natural quantities (such as temperature, pressure, light intensity, …) are **analog** quantities that vary continuously.



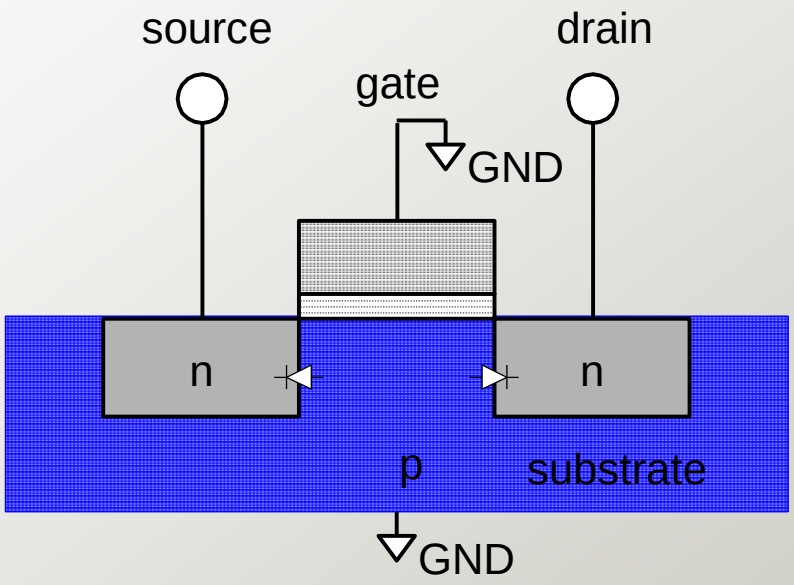
Analog = continuous
Digital = discrete

Digital systems can process, store, and transmit data more efficiently but can only assign discrete values to each point.

Transistors: nMOS

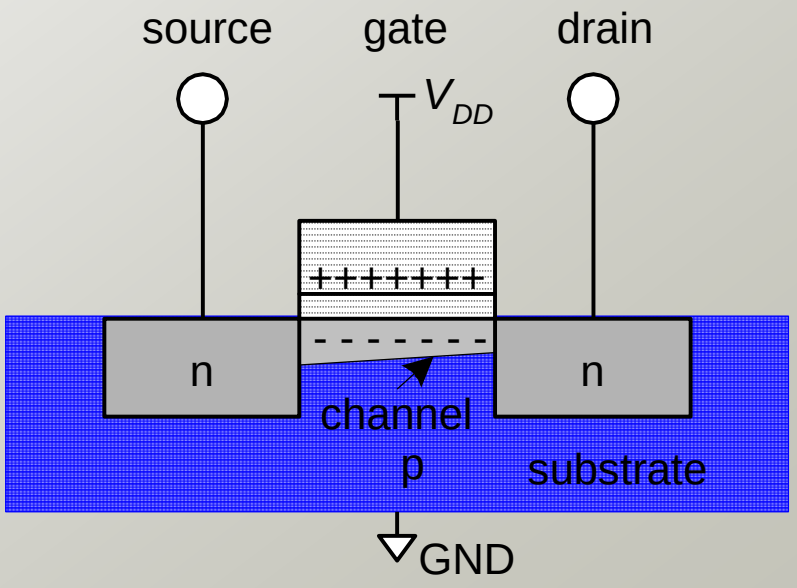
Gate = 0

OFF (no connection between source and drain)



Gate = 1

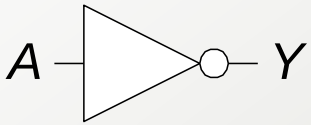
ON (channel between source and drain)



Transistor Function

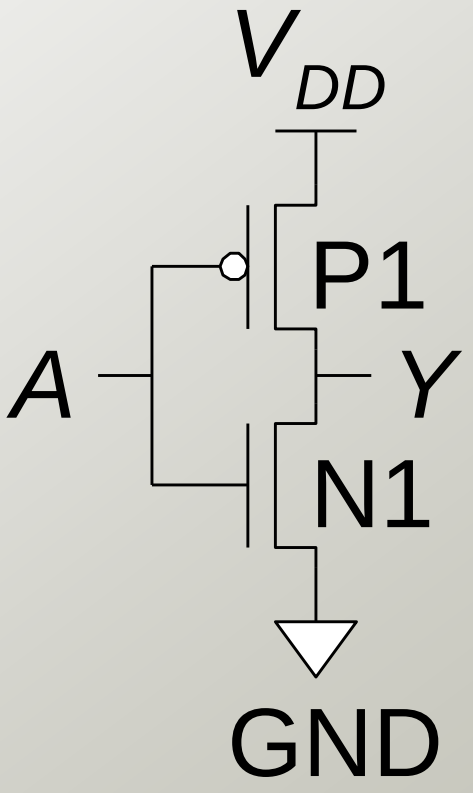
CMOS Gates: NOT Gate

NOT



$Y = \overline{A}$

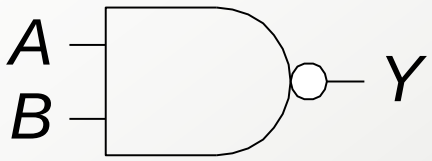
A	Y
0	1
1	0



A	P1	N1	*Y*
0	ON	OFF	1
1	OFF	ON	0

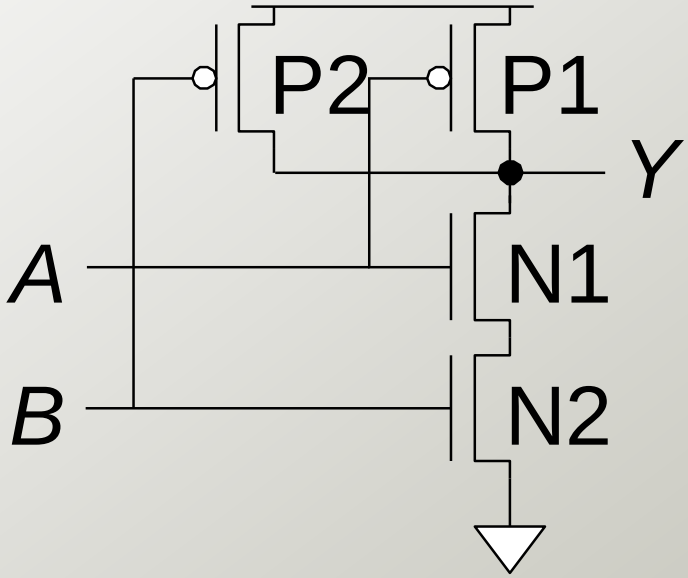
Logic Gates – NAND

NAND



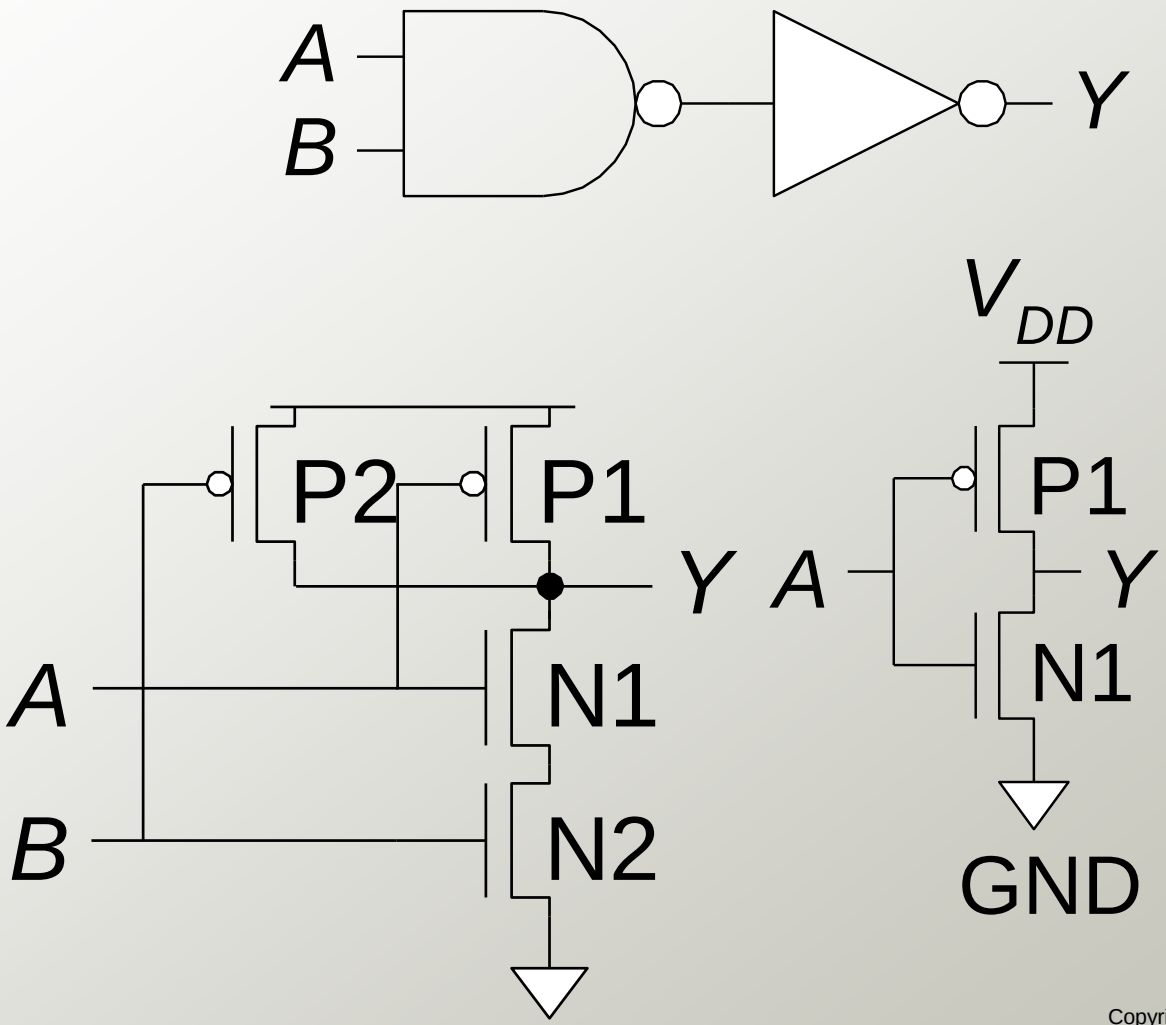
$$Y = \overline{AB}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



A	B	P1	P2	N1	N2	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

Logic Gates – AND



Digital Fundamentals

Number Systems, Operations, and Codes

Binary Numbers

For digital systems, the binary number system is used. Binary has a radix of two and uses the digits 0 and 1 to represent quantities.

The column weights of binary numbers are powers of two that increase from right to left beginning with $2^0 = 1$:

$$\ldots 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0.$$

For fractional binary numbers, the column weights are negative powers of two that decrease from left to right:

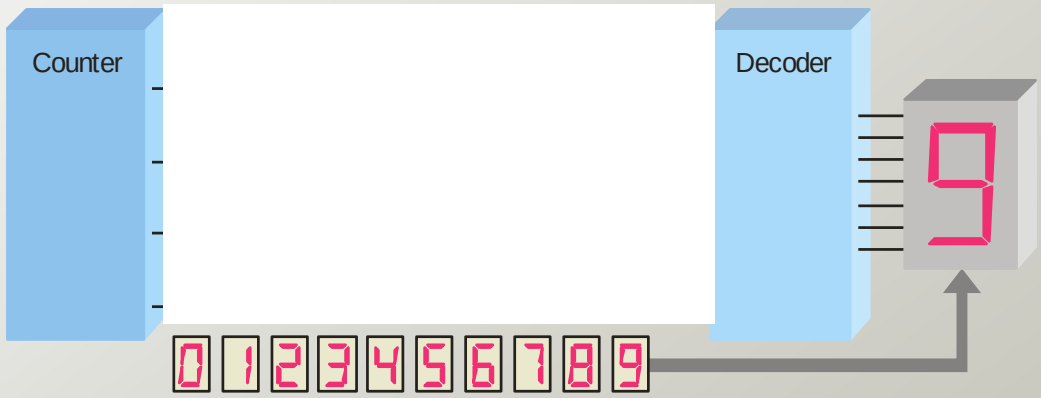
$$2^2\ 2^1\ 2^0.\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ \ldots$$

Decimal Vs. Binary

A binary counting sequence for numbers from zero to fifteen is shown.

Notice the pattern of zeros and ones in each column.

Digital counters frequently have this same pattern of digits:



Decimal Number	Binary Number
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

Binary Arithmetic

Binary Addition

The rules for binary addition are

0 + 0 = 0	Sum = 0, carry = 0
0 + 1 = 1	Sum = 1, carry = 0
1 + 0 = 1	Sum = 1, carry = 0
1 + 1 = 10	Sum = 0, carry = 1

```
  0 1 1 1
  00111      7
  10101     21
 -------   ----
  11100  =  28
```

Binary Arithmetic

Binary Subtraction

The rules for binary subtraction are

$$0 - 0 = 0$$
$$1 - 1 = 0$$
$$1 - 0 = 1$$
$$10 - 1 = 1 \text{ with a borrow of 1}$$

Example Subtract the binary number 00111 from 10101 and show the equivalent decimal subtraction.

Solution

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 10101 \\ 00111 \\ \hline 01110 \end{array} \quad \begin{array}{r} 21 \\ 7 \\ \hline = 14 \end{array}$$

Binary Arithmetic

Binary Multiplication

Partial products formed by multiplying a single digit of the multiplier with multiplicand.
Shifted partial products **summed** to form result.

Decimal		**Binary**
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
9660		0101
		+ 0000
	result	0100011

230 x 42 = 9660

5 x 7 = 35

Signed Numbers

Signed Binary Numbers

There are several ways to represent signed binary numbers. In all cases, the **MSB in a signed number is the sign bit**, that tells you if the number is positive or negative.

Computers use a modified 2's complement for signed numbers. Positive numbers are stored in *true* form (with a 0 for the sign bit) and negative numbers are stored in *complement* form (with a 1 for the sign bit).

For example, the positive number 58 is written using 8-bits as 00111010 (true form).

Sign bit

Magnitude bits

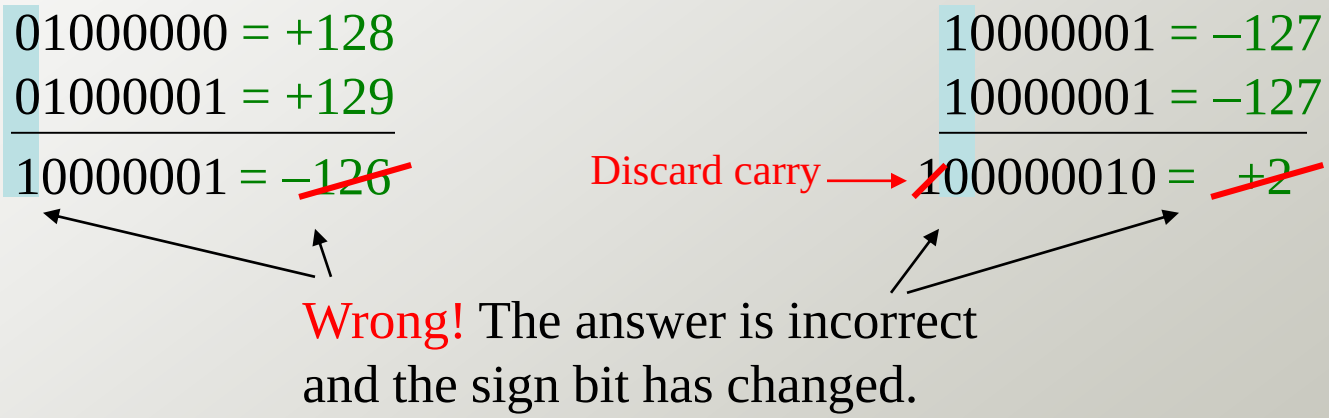
The sign bit is the left-most bit in a signed binary number

Arithmetic Operations with Signed Numbers - Overflow

Arithmetic Operations with Signed Numbers

Note that if the number of bits required for the answer is exceeded, overflow will occur. This occurs only if both numbers have the same sign. The overflow will be indicated by an incorrect sign bit.

Two examples are:



Arithmetic Operations with Signed Numbers

Signs for Addition

- When both numbers are positive, the sum is positive.
- When both numbers are negative, the sum is negative (2's complement form). The carry bit is discarded.
- When the larger number is positive and the smaller is negative, the sum is positive. The carry is discarded.
- When the larger number is negative and the smaller is positive, the sum is negative (2's complement form).

Digital Numbers Representations

Hexadecimal Numbers

Hexadecimal is a weighted number system. The column weights are powers of 16, which increase from right to left.

Octal Numbers

Octal is also a weighted number system. The column weights are powers of 8, which increase from right to left.

BCD

Binary coded decimal (BCD) is a weighted code that is commonly used in digital systems when it is necessary to show decimal numbers such as in clock displays.

Digital Codes

Gray code

Gray code is an unweighted code that has a single bit change between one code word and the next in a sequence. Gray code is used to avoid problems in systems where an error can occur if more than one bit changes at a time.

ASCII

ASCII is a code for alphanumeric characters and control characters. In its original form, ASCII encoded 128 characters and symbols using 7-bits. The first 32 characters are control characters, that are based on obsolete teletype requirements, so these characters are generally assigned to other functions in modern usage.

Parity error codes

Parity Method

The parity method is a method of error detection for simple transmission errors involving one bit (or an odd number of bits). A parity bit is an "extra" bit attached to a group of bits to force the number of 1's to be either even (even parity) or odd (odd parity).

The parity can detect up to One bit error and can't correct the error.

EVEN PARITY		ODD PARITY	
P	BCD	*P*	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

Hamming error codes

- Hamming code can detect up to 2 bits and correct 1 bit error.
- Hex equivalent of the data bits

0000000	0
0000111	1
0011011	2
0011110	3
0101010	4
0101101	5
0110011	6
0110100	7
1001011	8
1001100	9
1010010	A
1010101	B
1100001	C
1100110	D
1111000	E
1111111	F

Signed Fixed-Point Numbers

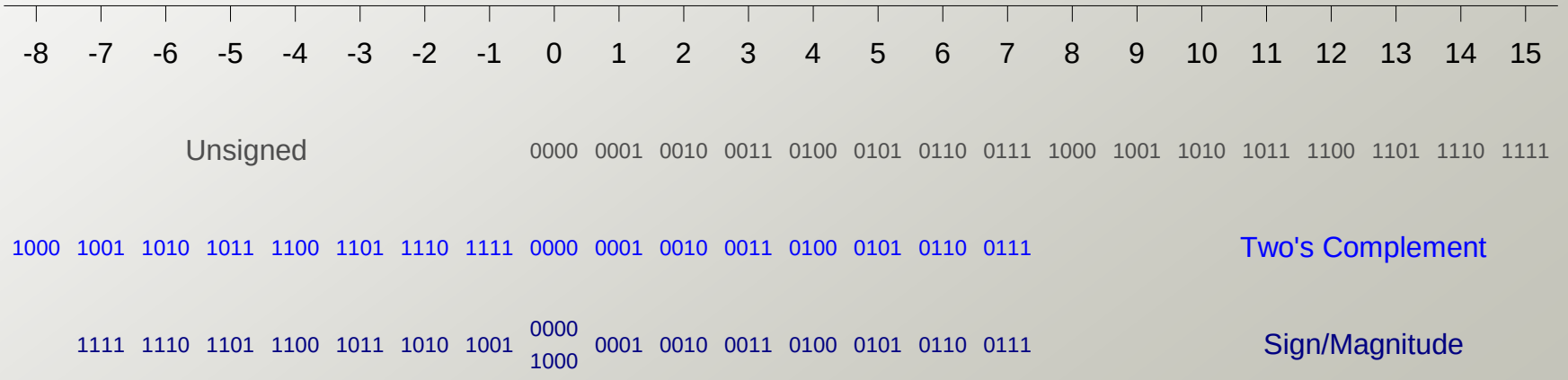
- **Representations:**
  - Sign/magnitude
  - Two's complement
- **Example:** Represent -7.5_{10} using 4 integer and 4 fraction bits
  - **Sign/magnitude:** 11111000
  - **1's complement:** 01111000
    10000111
  - **2's complement:**

1. +7.5:		01111000
2. Invert bits:		10000111
3. Add 1 to lsb:	+	1
		10001000

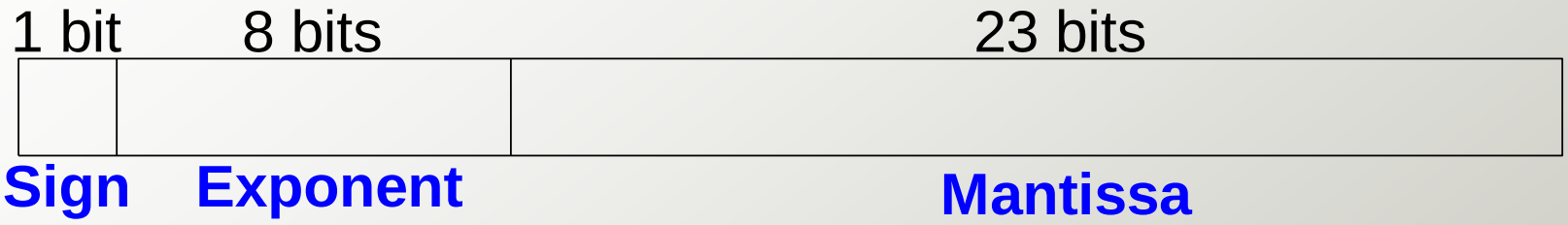
Signed Fixed-Point Numbers

Number System	Range
Unsigned	$[0, 2^N-1]$
Sign/Magnitude	$[-(2^{N-1}-1), 2^{N-1}-1]$
Two's Complement	$[-2^{N-1}, 2^{N-1}-1]$

For example, 4-bit representation:



Floating-Point Numbers



- In general, a number is written in scientific notation as:

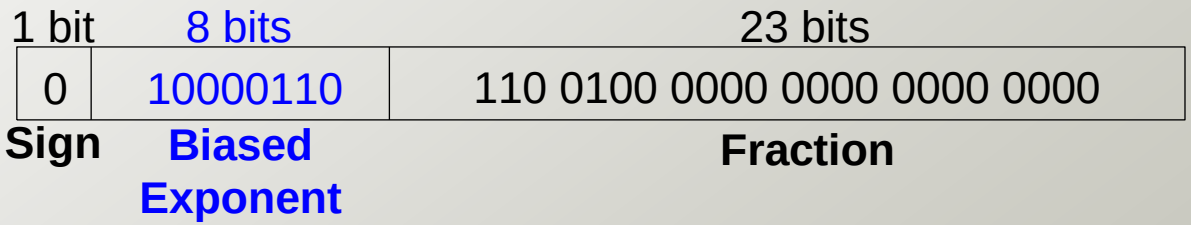
$$\pm M \times B^E$$

  - **M** = mantissa
  - **B** = base
  - **E** = exponent

IEEE 754 - floating-point standard

Floating-Point Precision

- **Single-Precision:**
  - 32-bit
  - 1 sign bit, 8 exponent bits, 23 fraction bits
  - bias = 127



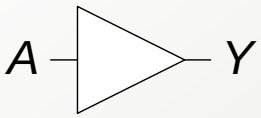
- **Double-Precision:**
  - 64-bit
  - 1 sign bit, 11 exponent bits, 52 fraction bits
  - bias = 1023

Digital Fundamentals

Logic Gates

Logic Gates

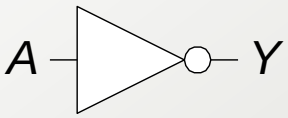
BUF



$Y = A$

A	Y
0	0
1	1

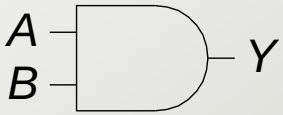
NOT



$Y = \overline{A}$

A	Y
0	1
1	0

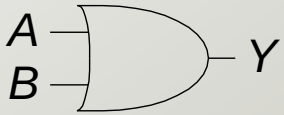
AND



$Y = AB$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

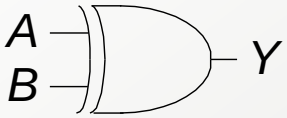


$Y = A + B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Logic Gates

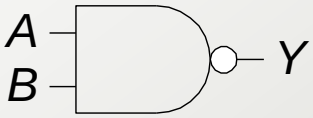
XOR



$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

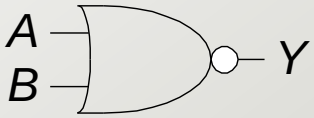
NAND



$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

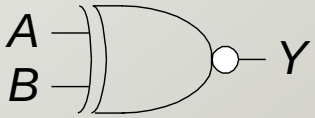
NOR



$Y = \overline{A + B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR



$Y = \overline{A \oplus B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Digital Fundamentals

Boolean Algebra and Logic Simplification

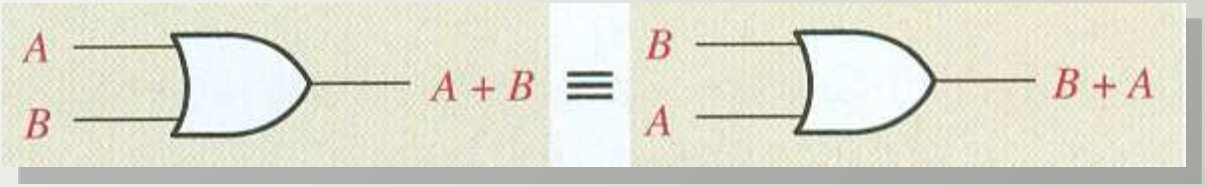
Laws Boolean Algebra

Commutative Laws

The **commutative laws** are applied to addition and multiplication.

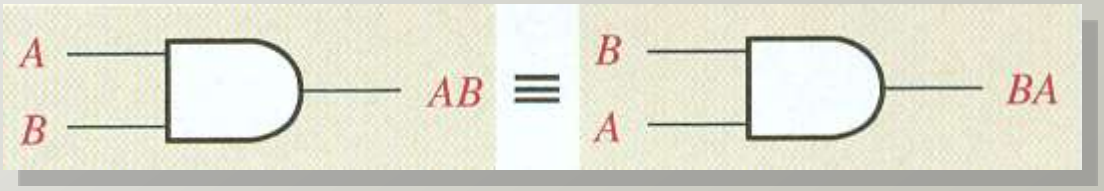
- **For addition, the commutative law states**
  In terms of the result, the order in which variables are ORed makes no difference.

$$A + B = B + A$$



- **For multiplication, the commutative law states**
  In terms of the result, the order in which variables are ANDed makes no difference.

$$AB = BA$$



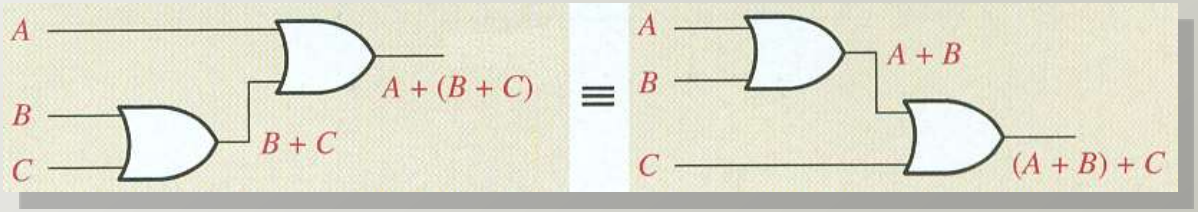
Laws Boolean Algebra

Associative Laws

The **associative laws** are also applied to addition and multiplication.

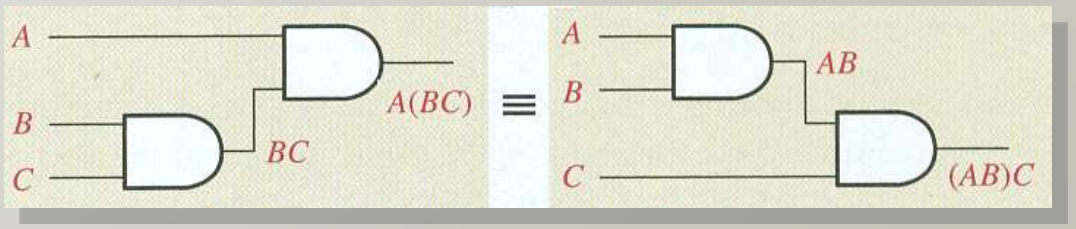
- **For addition, the associative law states**

$$A + (B + C) = (A + B) + C$$



- **For multiplication, the associative law states**

$$A(BC) = (AB)C$$

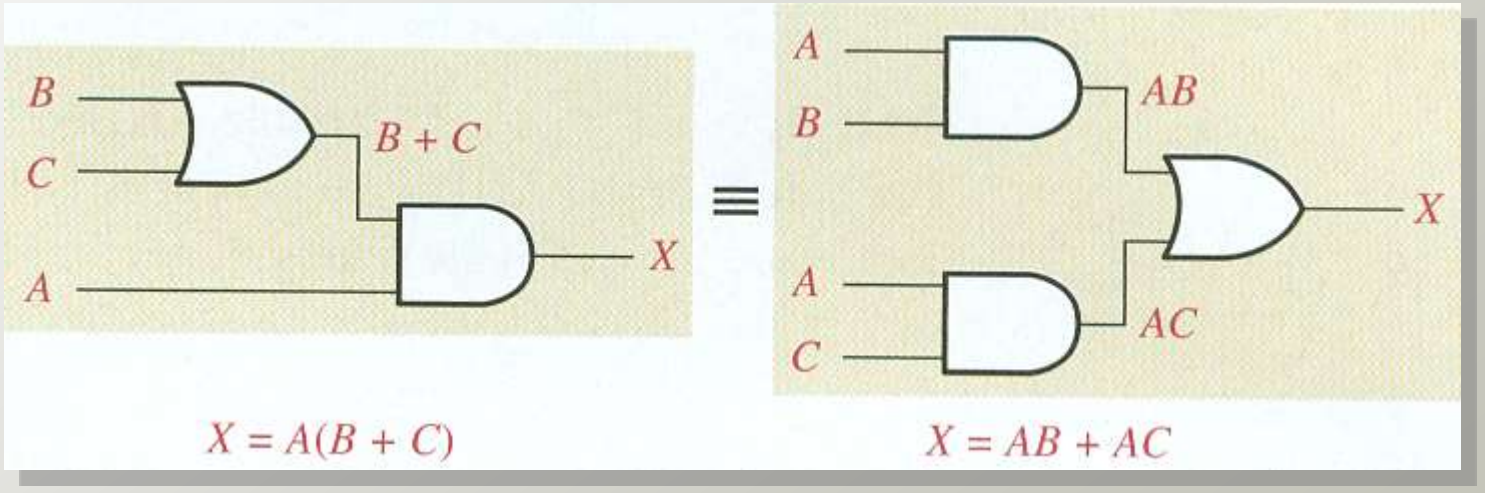


Laws Boolean Algebra

Distributive Law

The **distributive law** is the factoring law. A common variable can be factored from an expression just as in ordinary algebra. That is

$$A(B + C) = AB + AC$$



Rules of Boolean Algebra

1. $A + 0 = A$
2. $A + 1 = 1$
3. $A \cdot 0 = 0$
4. $A \cdot 1 = A$
5. $A + A = A$
6. $A + \overline{A} = 1$
7. $A \cdot A = A$
8. $A \cdot \overline{A} = 0$
9. $\overline{\overline{A}} = A$
10. $A + AB = A$
11. $A + \overline{A}B = A + B$
12. $(A + B)(A + C) = A + BC$

DeMorgan's Theorem

DeMorgan's Theorem

- Theorem 1 $\overline{XY} = \overline{X} + \overline{Y}$
- Theorem 2 $\overline{X + Y} = \overline{X}\,\overline{Y}$

Example

Apply DeMorgan's theorem to remove the overbar covering both terms from the expression $X = \overline{\overline{C} + D}$.

Solution

To apply DeMorgan's theorem to the expression, you can break the overbar covering both terms and change the sign between the terms. This results in $X = \overline{\overline{C}} \cdot \overline{D}$. Deleting the double bar gives $X = C \cdot \overline{D}$.

Digital Fundamentals

Combinational Logic Analysis

Boolean Analysis of Logic Circuits

SOP and POS forms

Boolean expressions can be written in the **sum-of-products** form (**SOP**) or in the **product-of-sums** form (**POS**). These forms can simplify the implementation of combinational logic, particularly with PLDs. In both forms, an overbar cannot extend over more than one variable.

An expression is in SOP form when two or more product terms are summed as in the following examples:

$\overline{A}\,\overline{B}\,\overline{C} + A\,B$ $\qquad$ $A\,B\,\overline{C} + \overline{C}\,\overline{D}$ $\qquad$ $C\,D + \overline{E}$

An expression is in POS form when two or more sum terms are multiplied as in the following examples:

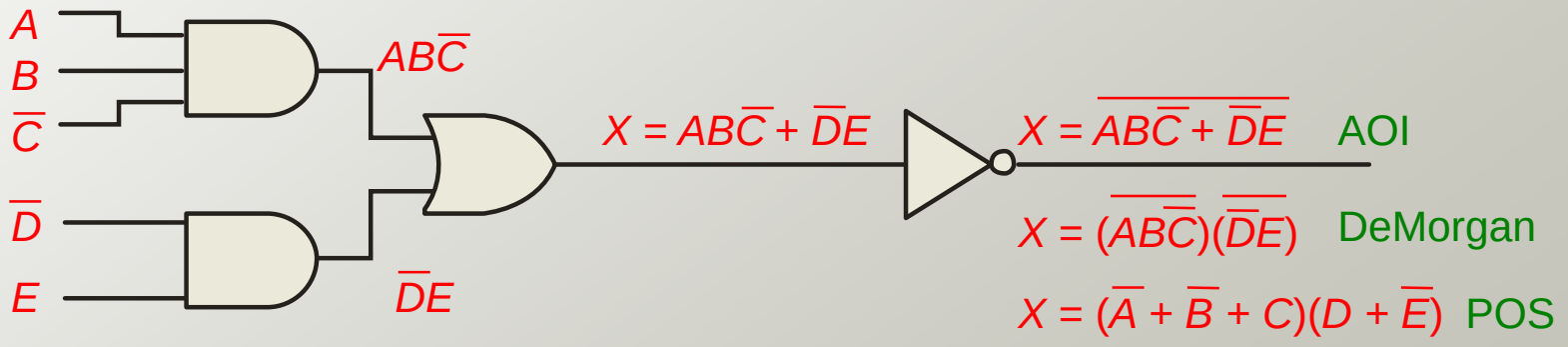
$(A + B)(\overline{A} + C)$ $\qquad$ $(A + B + \overline{C})(B + D)$ $\qquad$ $(\overline{A} + B)C$

Combinational Logic Analysis

Combinational Logic Circuits

When the output of a SOP form is inverted, the circuit is called an AND-OR-Invert (AOI) circuit. The AOI configuration lends itself to product-of-sums (POS) implementation.

An example of an AOI implementation is shown. The output expression can be changed to a POS expression by applying DeMorgan's theorem twice.



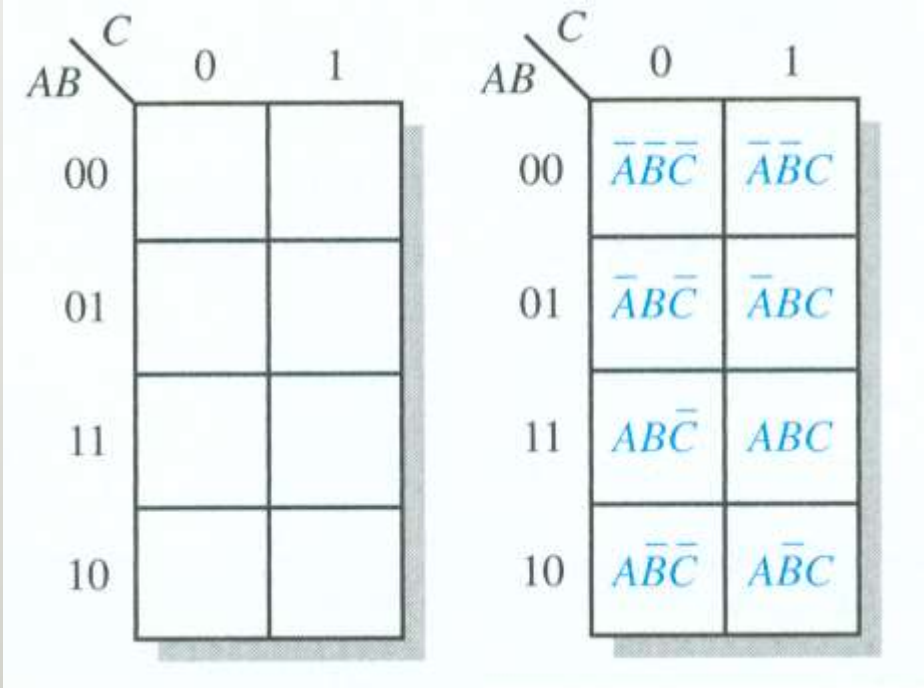
Boolean Analysis of Logic Circuits

Karnaugh maps

The Karnaugh map (K-map) is a tool for simplifying combinational logic with 3 or 4 variables. For 3 variables, 8 cells are required (2^3).

The map shown is for three variables labeled *A*, *B*, and *C*. Each cell represents one possible product term.

Each cell differs from an adjacent cell by only one variable.



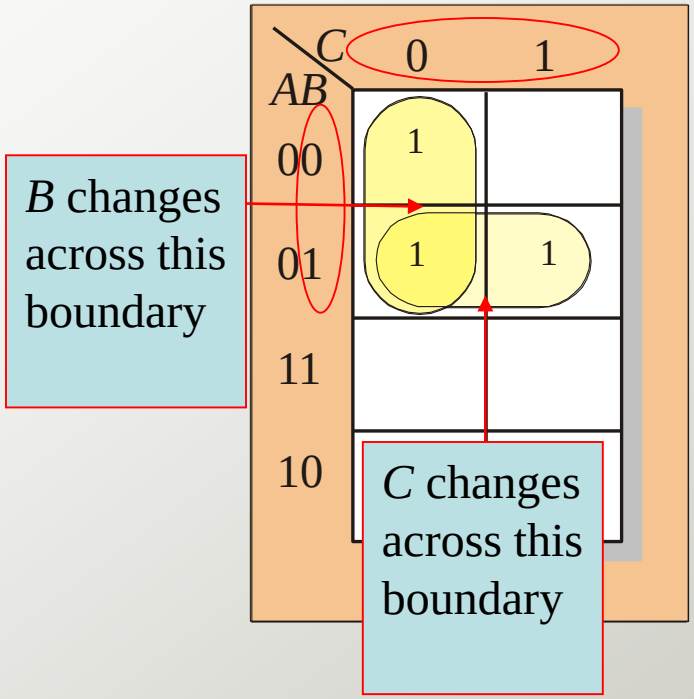
3-Variable Karnaugh Map

Boolean Analysis of Logic Circuits

Karnaugh maps

K-maps can simplify combinational logic by **grouping cells and eliminating variables that change**.

Example Group the 1's on the map and read the minimum logic.



Solution

1. Group the 1's into two overlapping groups as indicated.
2. Read each group by eliminating any variable that changes across a boundary.
3. The vertical group is read $\bar{A}\bar{C}$.
4. The horizontal group is read $\bar{A}B$.

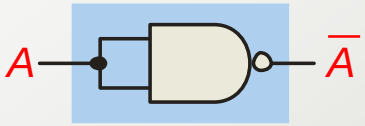
$$X = \bar{A}\bar{C} + \bar{A}B$$

Combinational Logic Analysis

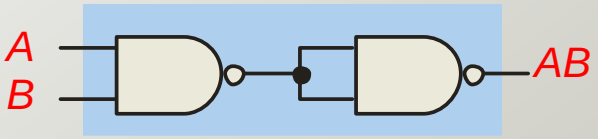
Universal Gates

NAND gates are sometimes called **universal** gates because they can be used to produce the other basic Boolean functions.

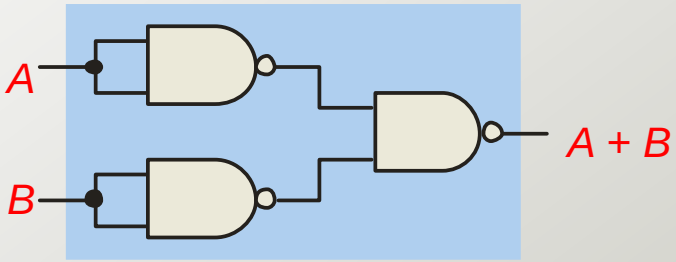
Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0



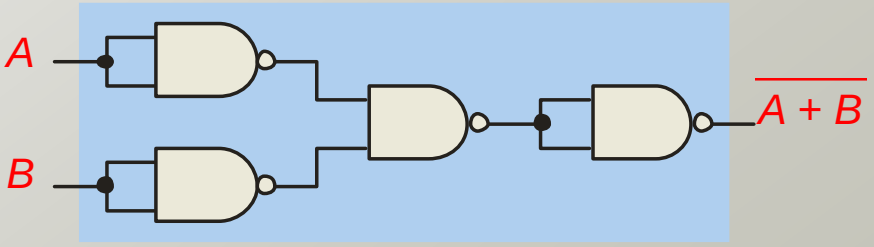
Inverter



AND gate



OR gate



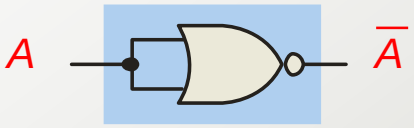
NOR gate

Combinational Logic Analysis

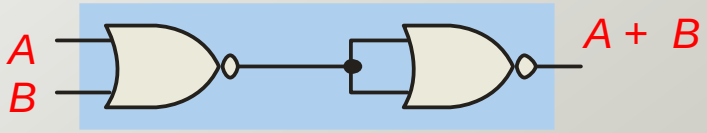
Universal Gates

NOR gates are also **universal** gates and can form all of the basic gates.

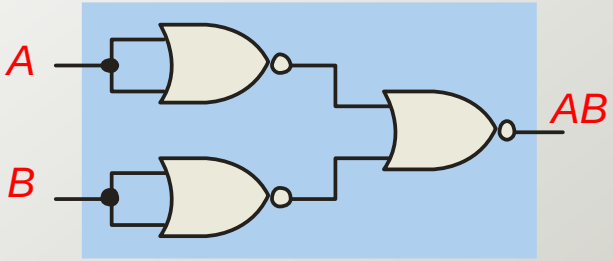
Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	0



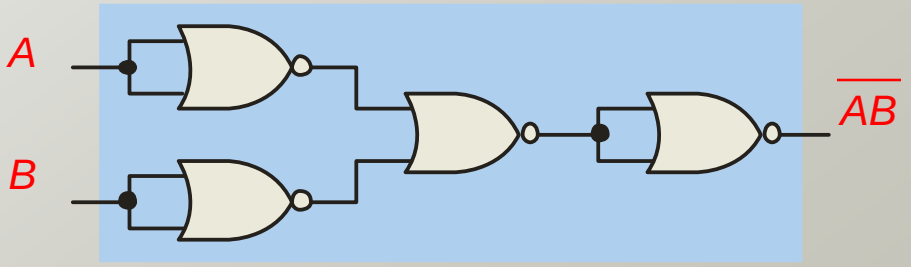
Inverter



OR gate



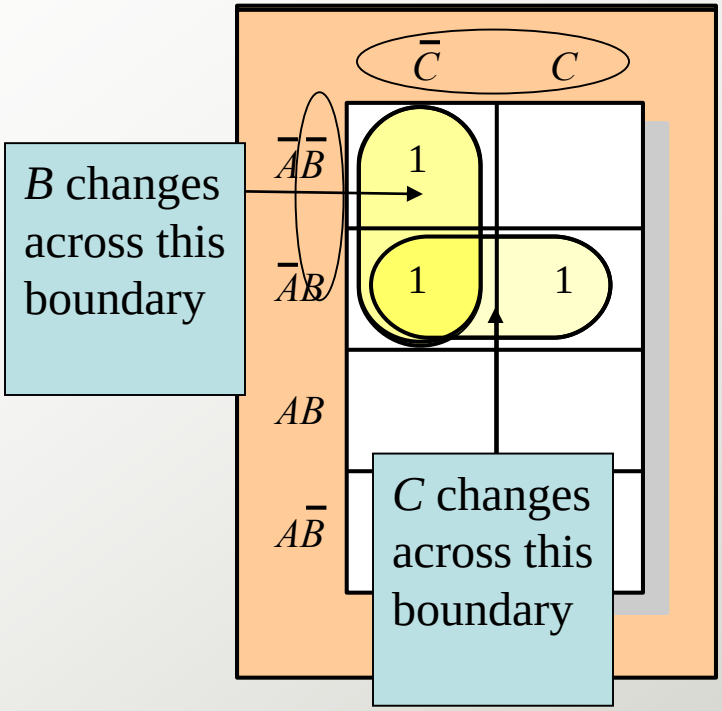
AND gate



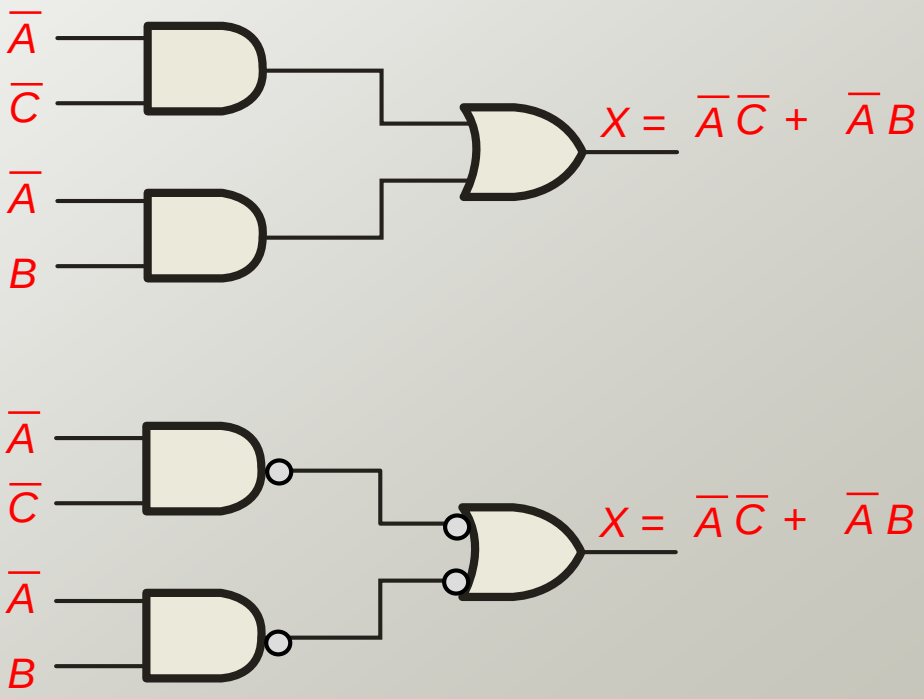
NAND gate

Combinational Logic Analysis

Solution



Circuit:



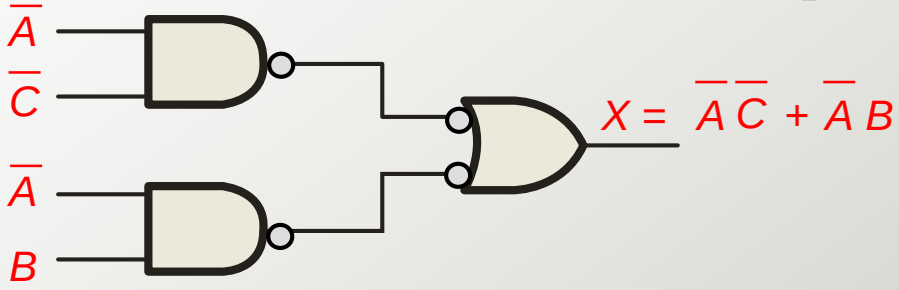
Recall from Boolean algebra that double inversion cancels. By adding inverting bubbles to above circuit, it is easily converted to NAND gates.

Combinational Logic Analysis

NAND Logic

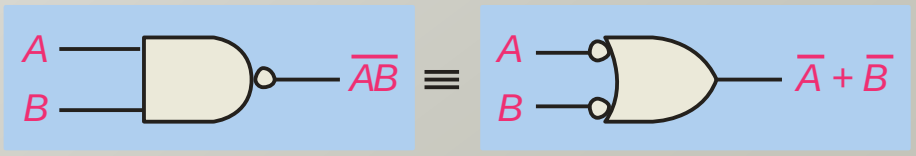
Recall from DeMorgan's theorem that $\overline{AB} = \overline{A} + \overline{B}$.

By using equivalent symbols, it is simpler to read the logic of SOP forms. The earlier example shows the idea:



Inputs		Output	
A	B	$\overline{AB}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

The logic is easy to read if you (mentally) cancel the two connected bubbles on a line.

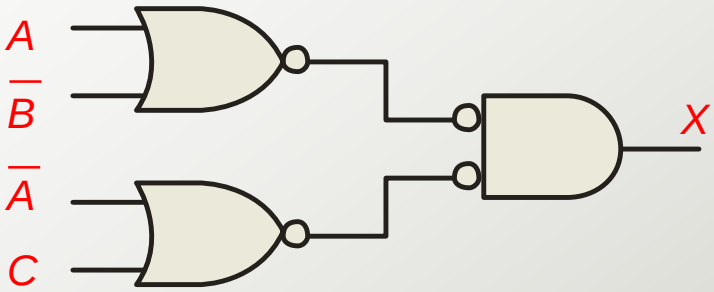


NAND Negative-OR

Combinational Logic Analysis

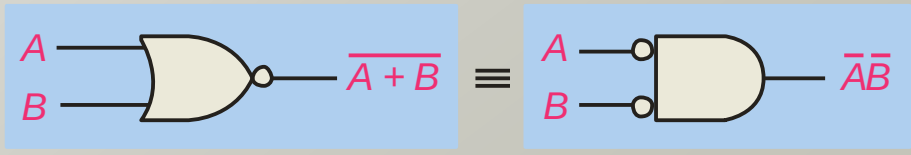
NOR Logic

Alternatively, DeMorgan's theorem can be written as $\overline{A + B} = \overline{A}\,\overline{B}$. By using equivalent symbols, it is simpler to read the logic of POS forms. For example,



Inputs		Output	
A	B	$\overline{A + B}$	$\overline{A}\,\overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Again, the logic is easy to read if you cancel the two connected bubbles on a line.



NOR Negative-AND

Combinational and Sequential

- A **combinational logic circuit** is a circuit whose output depends only on the circuit's current inputs.

  "Has no memory of the past."

  - Gates, Comparator, Decoders, Encoders, MUXs, DEMUXs, Adders
- A **sequential logic circuit** is a circuit whose output may depend on the circuit's previous states as well as its current inputs. "Has a memory."
  - Latches, FlipFlops, Counters, Shift registers, Memories.

Digital Fundamentals

Functions of Combinational Logic

Combinational Logic - Basic Adders

Half-Adder

Basic rules of binary addition are performed by a **half adder**, which has two binary inputs (*A* and *B*) and two binary outputs (Carry out and Sum).

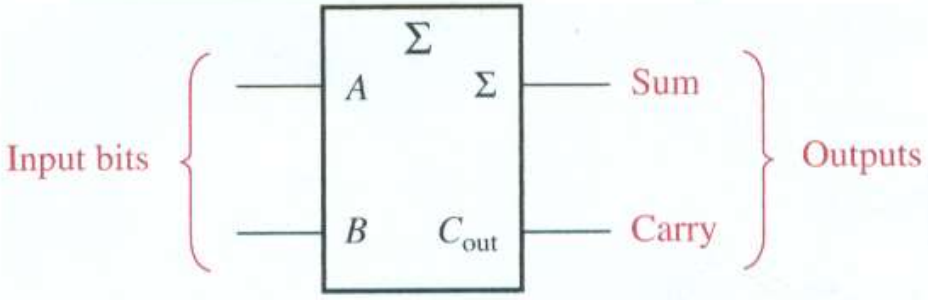
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

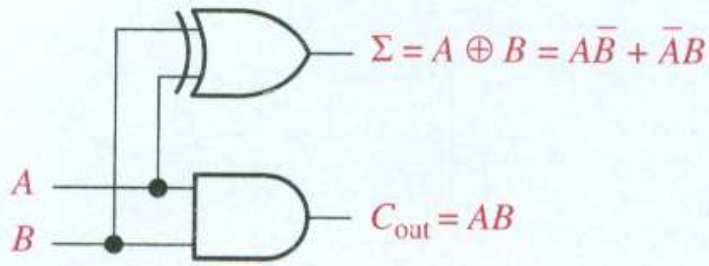
Σ = sum

C_{out} = output carry

A and *B* = input variables (operands)

Half-Adder Truth Table





Combinational Logic - Basic Adders

Full-Adder

By contrast, a **full adder** has three binary inputs (*A*, *B*, and Carry in) and two binary outputs (Carry out and Sum).

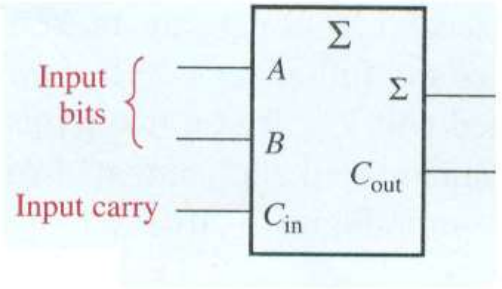
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C_{in} = input carry, sometimes designated as *CI*

C_{out} = output carry, sometimes designated as *CO*

Σ = sum

A and *B* = input variables (operands)



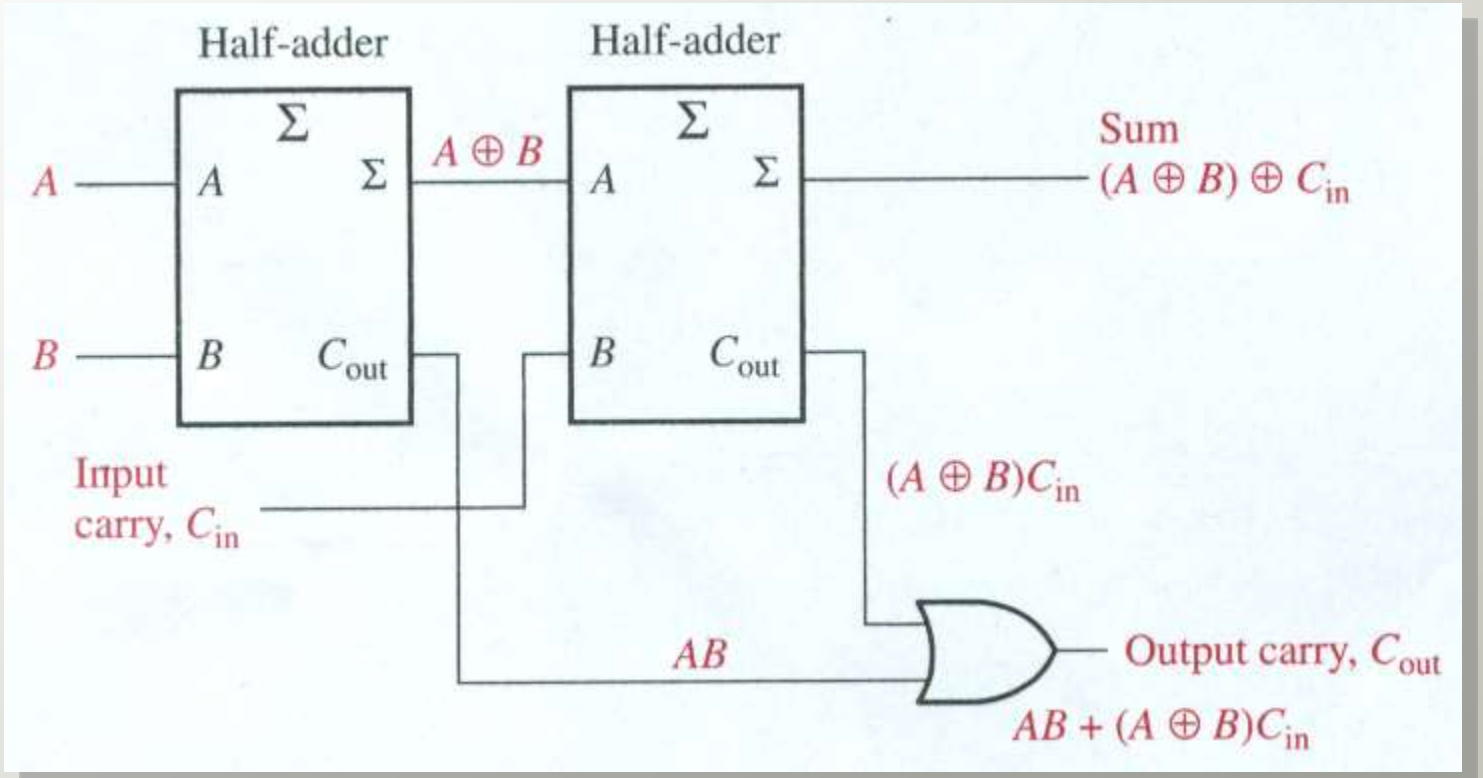
$$\Sigma = (A \oplus B) \oplus C_{in}$$

$$C_{out} = AB + (A \oplus B)C_{in}$$

Combinational Logic - Basic Adders

Full-Adder

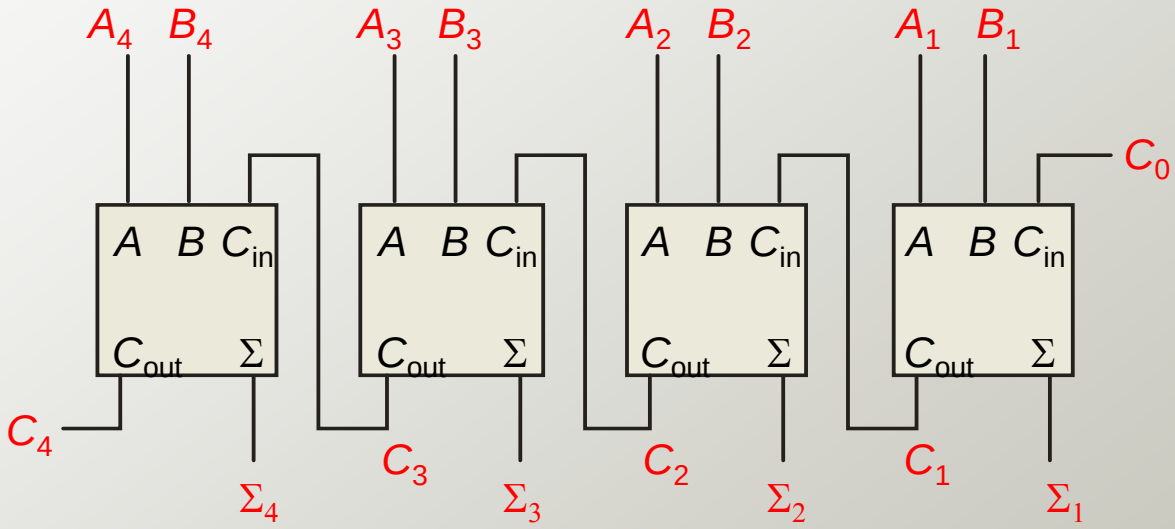
A full-adder can be constructed from two half adders as shown:



Combinational Logic - Parallel Binary Adders

Parallel Adders

Full adders are combined into parallel adders that can add binary numbers with multiple bits. A 4-bit adder is shown.



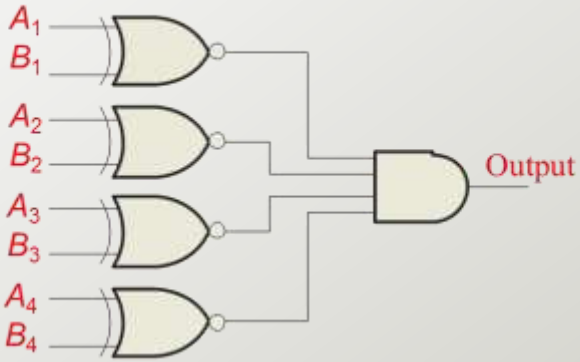
The output carry (C_4) is not ready until it propagates through all of the full adders. This is called *ripple carry*, delaying the addition process.

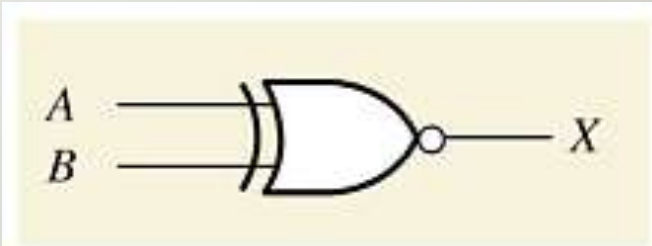
Combinational Logic - Comparators

Comparators

The function of a comparator is to compare the magnitudes of two binary numbers to determine the relationship between them. In the simplest form, a comparator can test for equality using XNOR gates.

AND the outputs of four XNOR gates.





A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

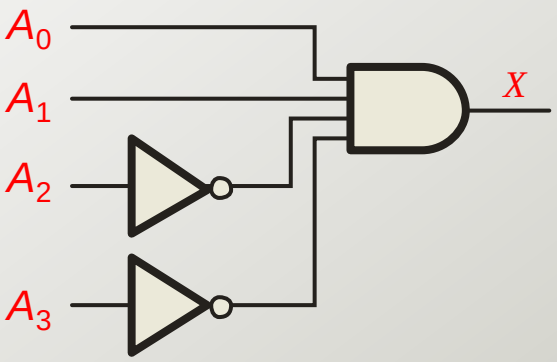
The output is 1 when the inputs are equal

Combinational Logic - Decoders

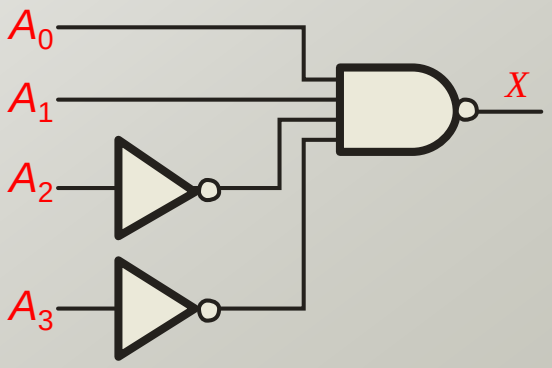
Decoders

A **decoder** is a logic circuit that detects the presence of a specific combination of bits at its input. Two simple decoders that detect the presence of the binary code 0011 are shown.

The first has an active HIGH output; the second has an active LOW output.



Active HIGH decoder for 0011



Active LOW decoder for 0011

Combinational Logic - Encoders

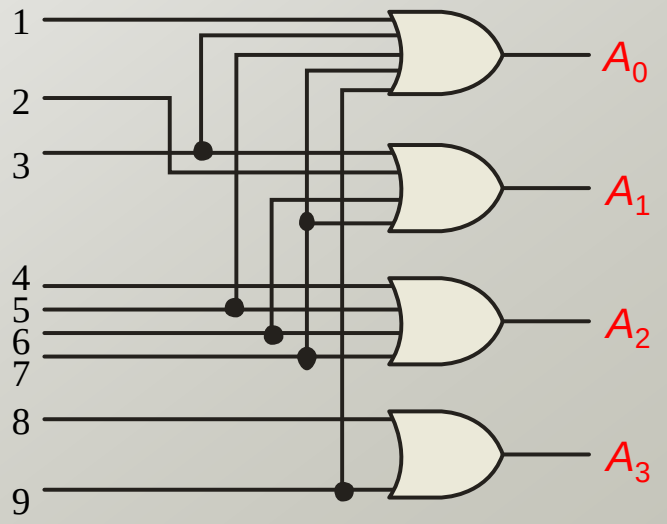
Encoders

An **encoder** accepts an active logic level on one of its inputs and converts it to a coded output, such as BCD or binary.

The decimal to BCD is an encoder with an input for each of the ten decimal digits and four outputs that represent the BCD code for the active digit.

The basic logic diagram is shown.

There is no zero input because the outputs are all LOW when the input is zero.



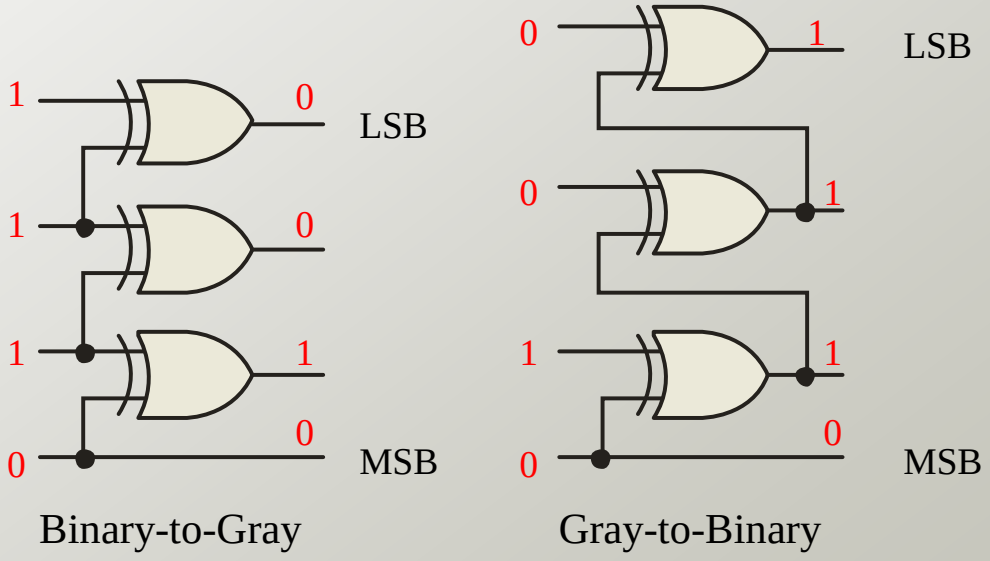
Combinational Logic – Code Converter

Code Converters

There are various code converters that change one code to another. Two examples are the four bit binary-to-Gray converter and the Gray-to-binary converter.

Example Show the conversion of binary 0111 to Gray and back.

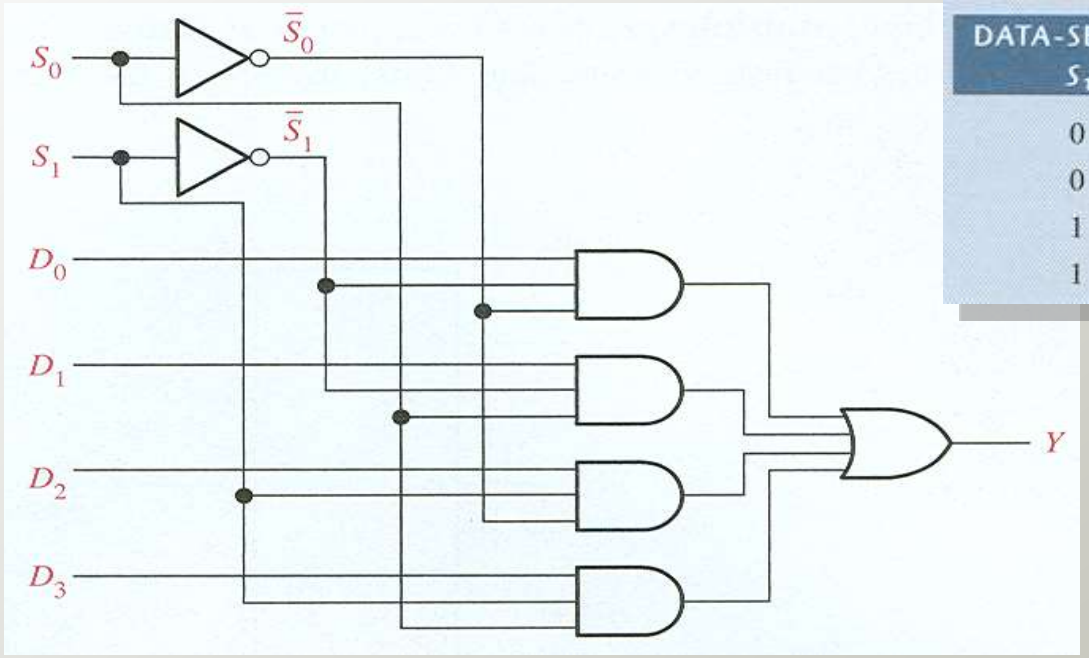
Solution



Combinational Logic – Multiplexer

Multiplexers

A multiplexer (MUX) selects one data line from two or more input lines and routes data from the selected line to the output. The particular data line that is selected is determined by the select inputs.



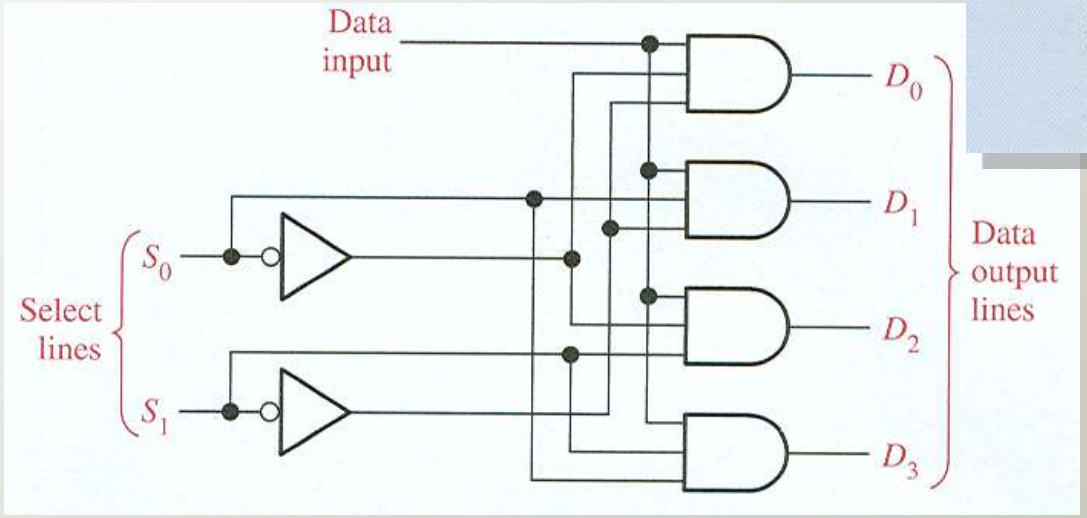
DATA-SELECT INPUTS S_1	S_0	INPUT SELECTED
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Combinational Logic – DeMultiplexer

Demultiplexers

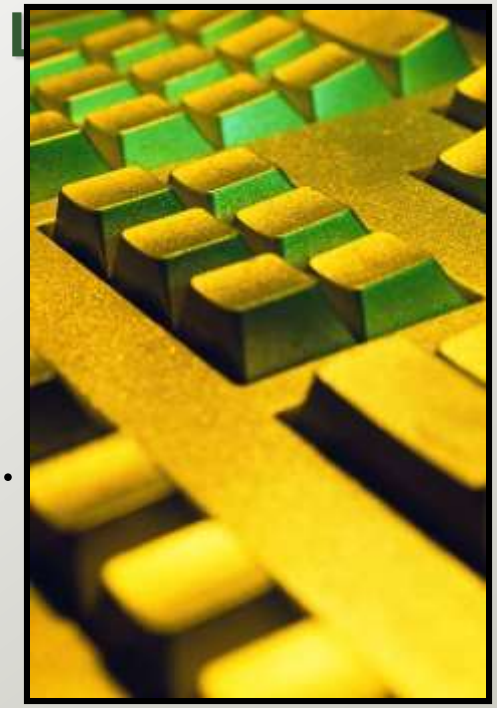
A demultiplexer (DEMUX) performs the opposite function from a MUX. It switches data from one input line to two or more data lines depending on the select inputs.

DATA-SELECT INPUTS S_1	S_0	OUTPUT SELECTED
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



Functions of Combinational L

Parity Generators/Checkers



Parity is an error detection method that uses an extra bit appended to a group of bits to force them to be either odd or even. In even parity, the total number of ones is even; in odd parity the total number of ones is odd.

Number of Inputs A–I That Are HIGH	Outputs Σ Even	Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

Digital Fundamentals

Latches, Flip-Flops and Timers

Latches

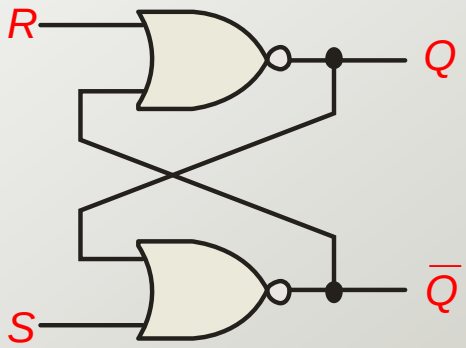
Latches

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

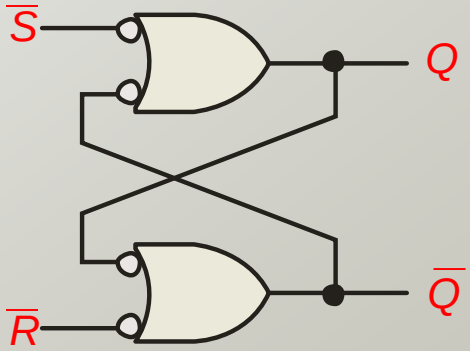
The S-R (Set-Reset) latch is the most basic type. It can be constructed from NOR gates or NAND gates.

With NOR gates, the latch responds to active-HIGH inputs.

With NAND gates, the latch responds to active-LOW inputs.



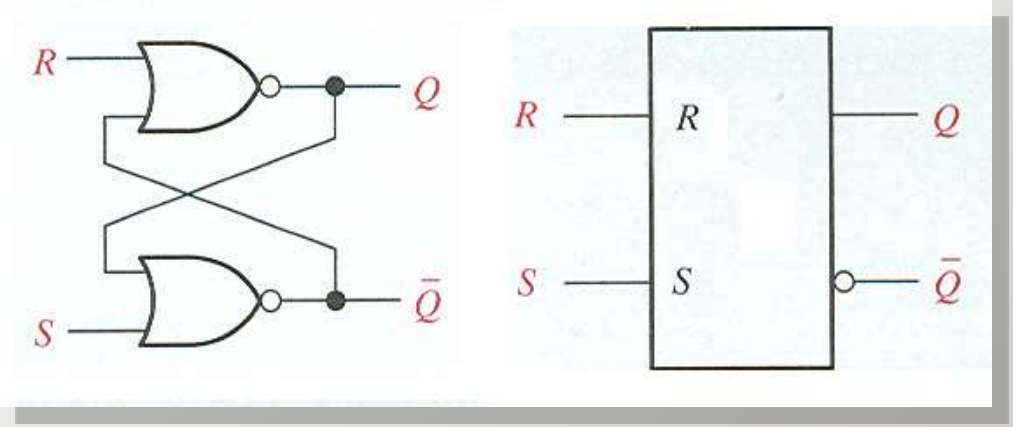
NOR Active-HIGH Latch



NAND Active-LOW Latch

Latches

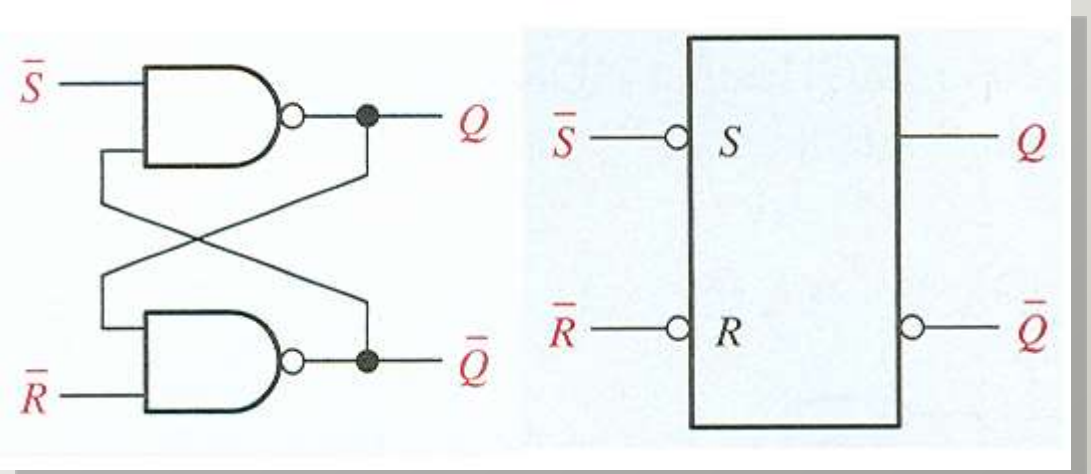
Active-HIGH S-R Latch



INPUTS S	INPUTS R	OUTPUTS Q	OUTPUTS $\bar{Q}$	COMMENTS
0	0	NC	NC	No change. Latch remains in present state.
0	1	0	1	Latch RESET.
1	0	1	0	Latch SET.
1	1	0	0	Invalid condition

Latches

Active-LOW S-R Latch



INPUTS		OUTPUTS		COMMENTS
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
1	1	NC	NC	No change. Latch remains in present state.
0	1	1	0	Latch SET.
1	0	0	1	Latch RESET.
0	0	1	1	Invalid condition

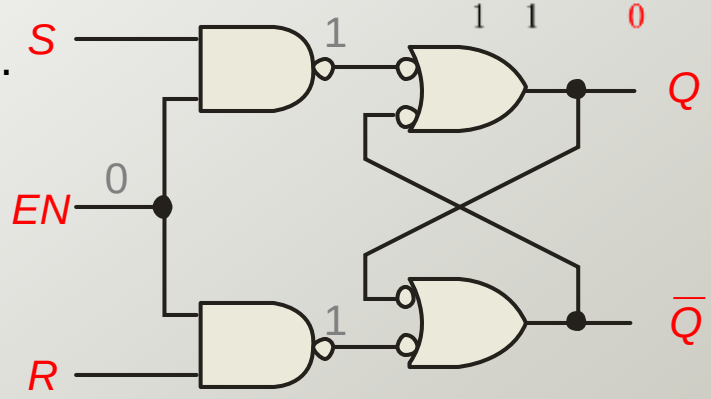
Latches

Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

Gated Latches

A gated latch is a variation on the basic latch.

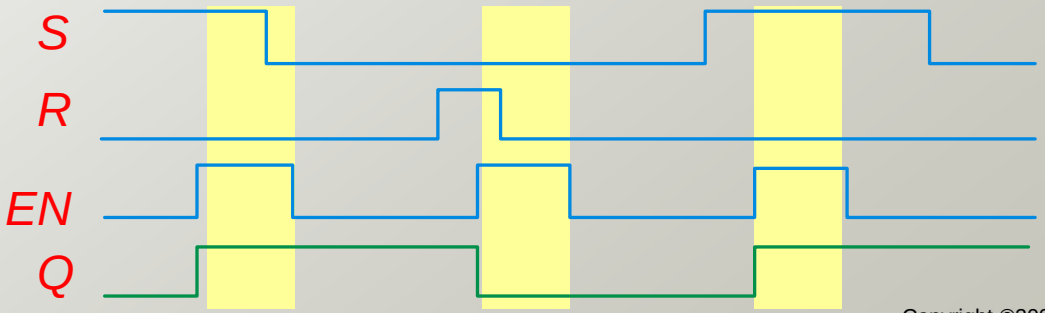
The gated latch has an additional input, called enable (*EN*) that must be HIGH in order for the latch to respond to the *S* and *R* inputs.



Example

Show the *Q* output with relation to the input signals.
Assume *Q* starts LOW.
Keep in mind that *S* and *R* are only active when *EN* is HIGH.

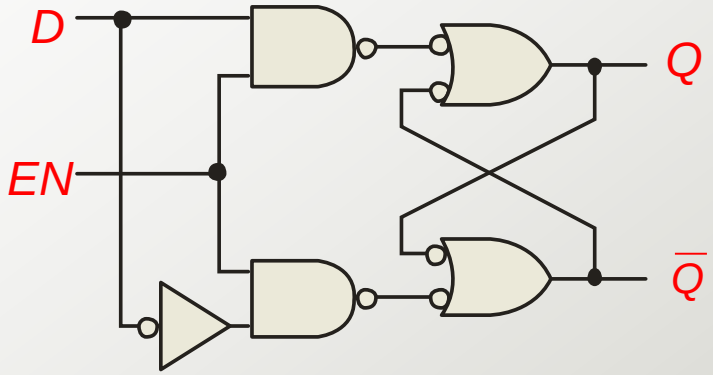
Solution

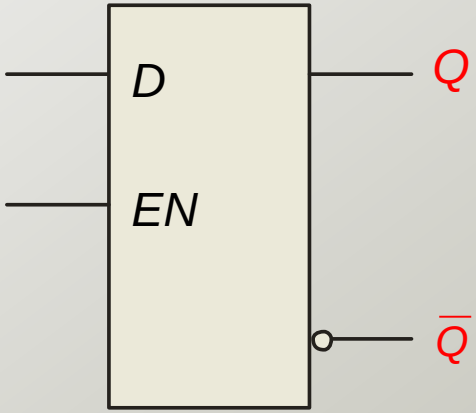


Latches

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.

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

Edge-Triggered Flip-Flops

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

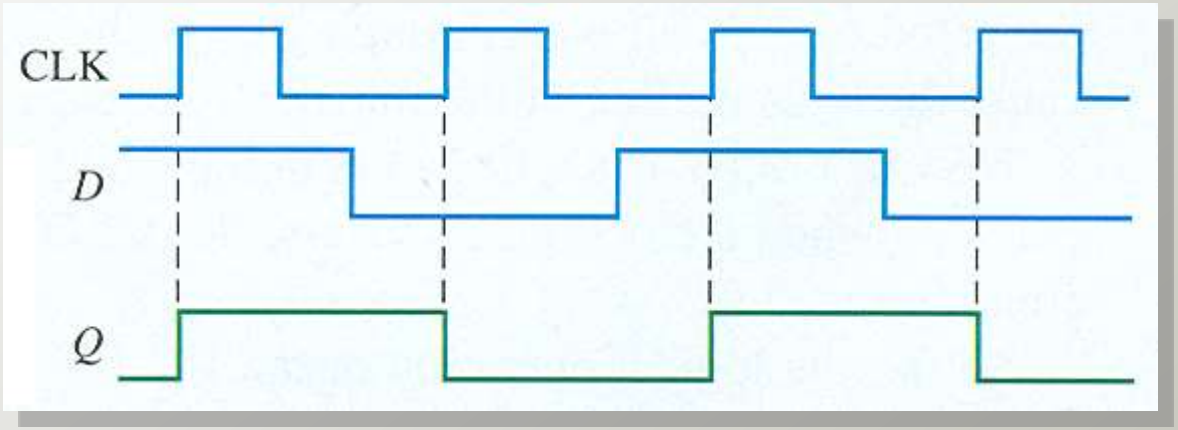
(b) Negative-edge triggered

Edge-Triggered Flip-Flops

Edge-triggered D flip-flop

INPUTS		OUTPUTS		COMMENTS
D	CLK	Q	$\overline{Q}$	
1	↑	1	0	SET (stores a 1)
0	↑	0	1	RESET (stores a 0)

↑ = clock transition LOW to HIGH



Edge-Triggered Flip-Flops

Flip-flops

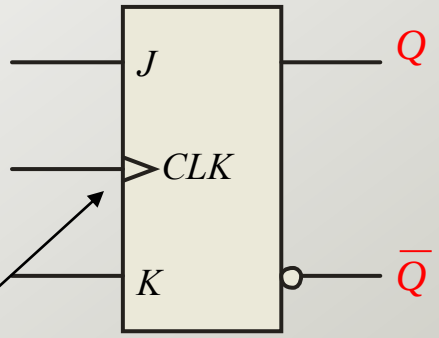
The J-K flip-flop is more versatile than the D flip flop.

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

Edge-Triggered Flip-Flops

Edge-triggered J-K flip-flop

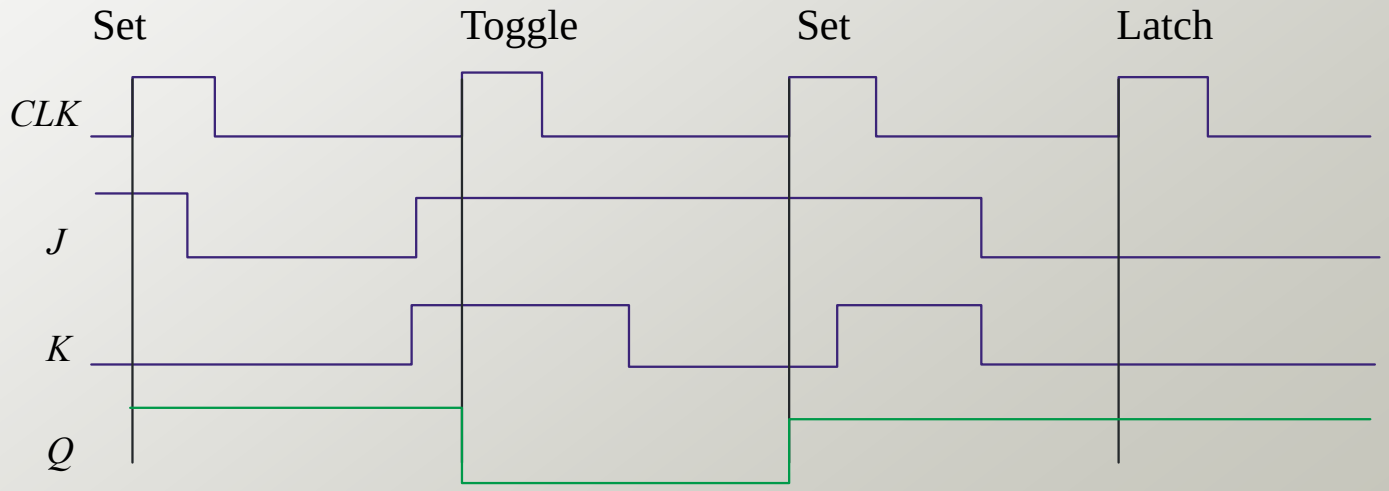


Example

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.

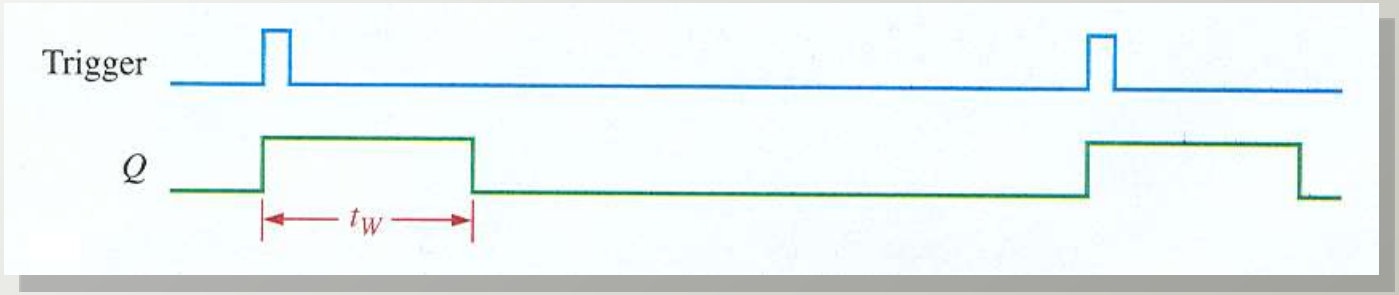
Solution



Multivibrator

Monostable

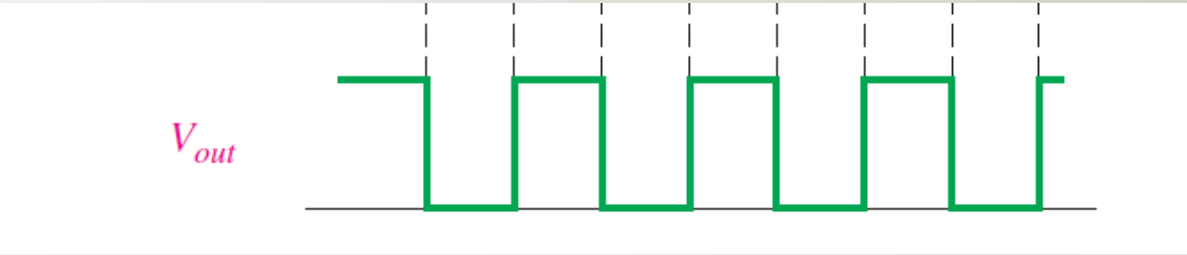
The **monostable** or **one-shot** multivibrator is a device with only one stable state. When triggered, it goes to its unstable state for a predetermined length of time, then returns to its stable state.



Multivibrator

Astable

An astable multivibrator is a device that has no stable states; it changes back and forth (oscillates) between two unstable states without any external triggering. The resulting output is typically **a square wave that is used as a clock signal** in many types of sequential logic circuits.



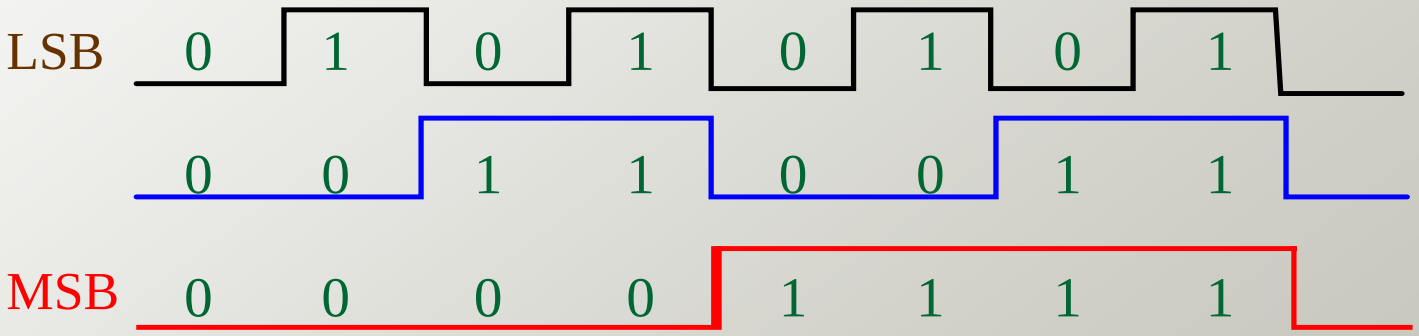
Digital Fundamentals

CHAPTER
Counters

Counters

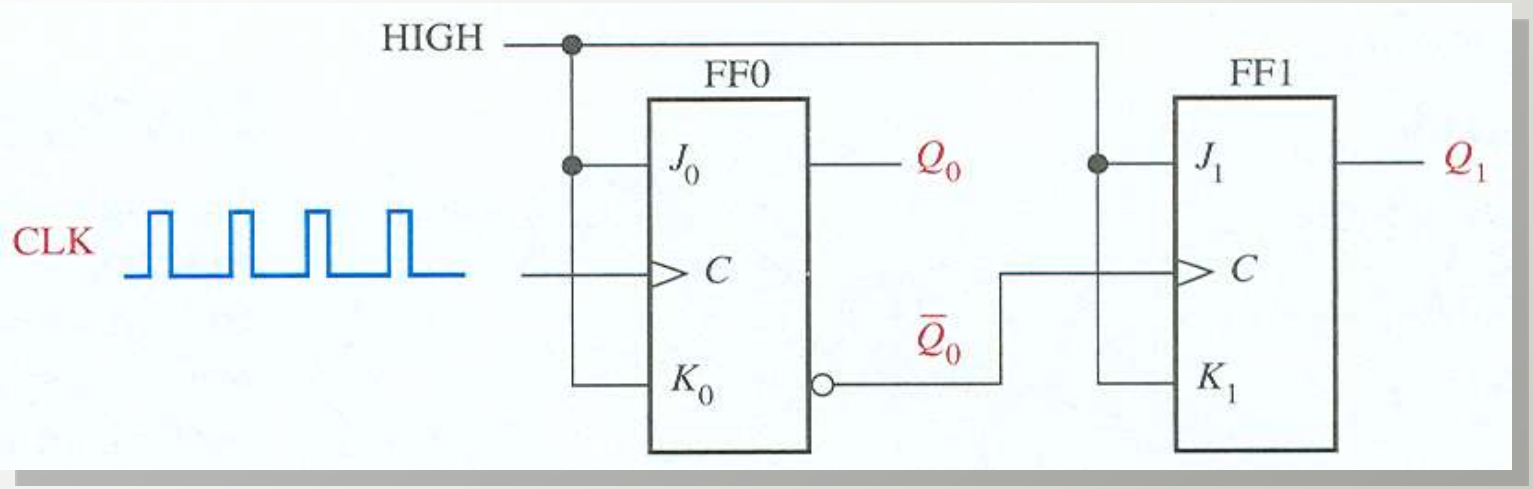
Counting in Binary

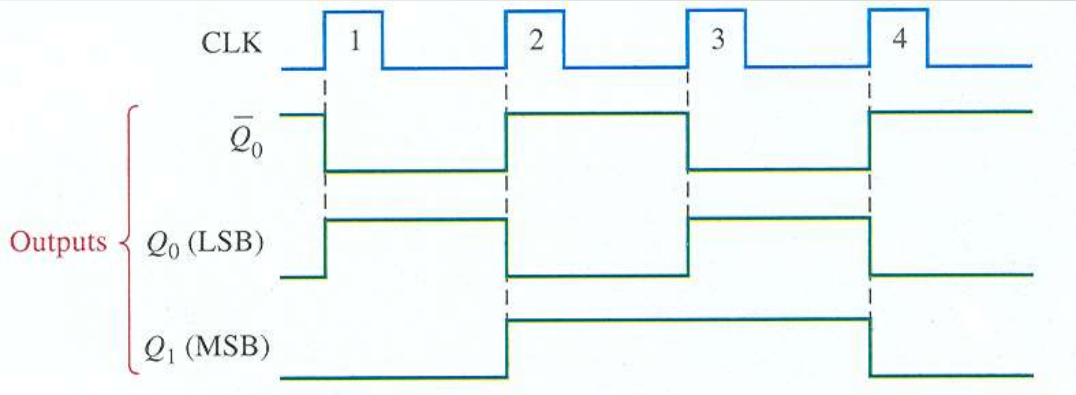
A counter can form the same pattern of 0's and 1's with logic levels. The first stage in the counter represents the least significant bit – notice that these waveforms follow the same pattern as counting in binary.



Asynchronous binary counter

2-bit Asynchronous binary counter

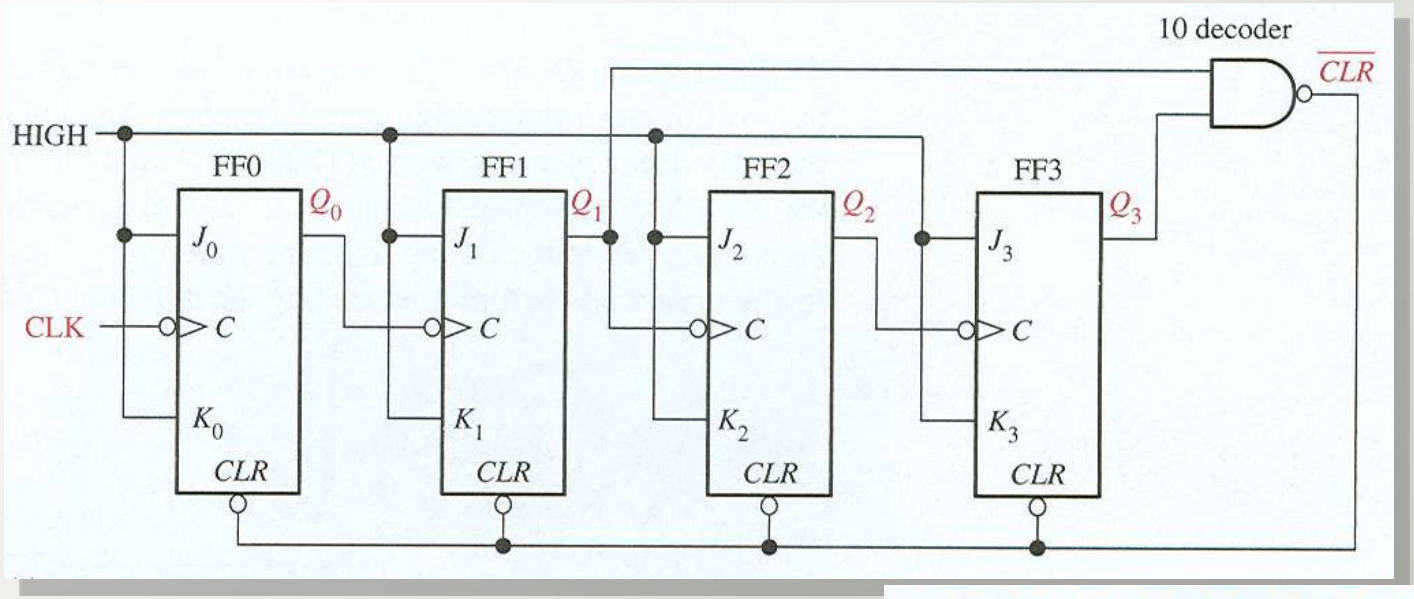


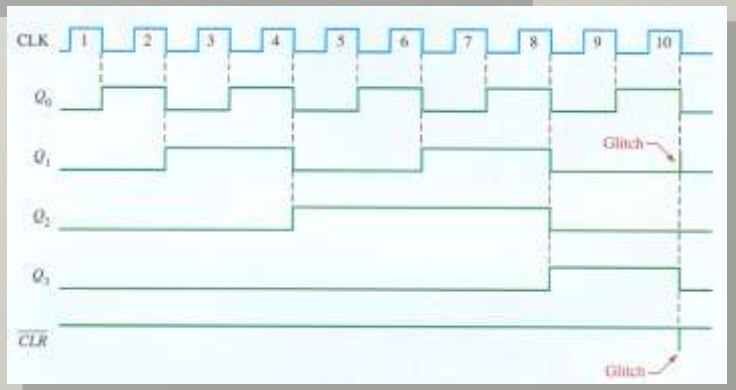


CLOCK PULSE	Q_1	Q_0
Initially	0	0
1	0	1
2	1	0
3	1	1
4 (recycles)	0	0

Asynchronous binary counter

Asynchronous decade counter



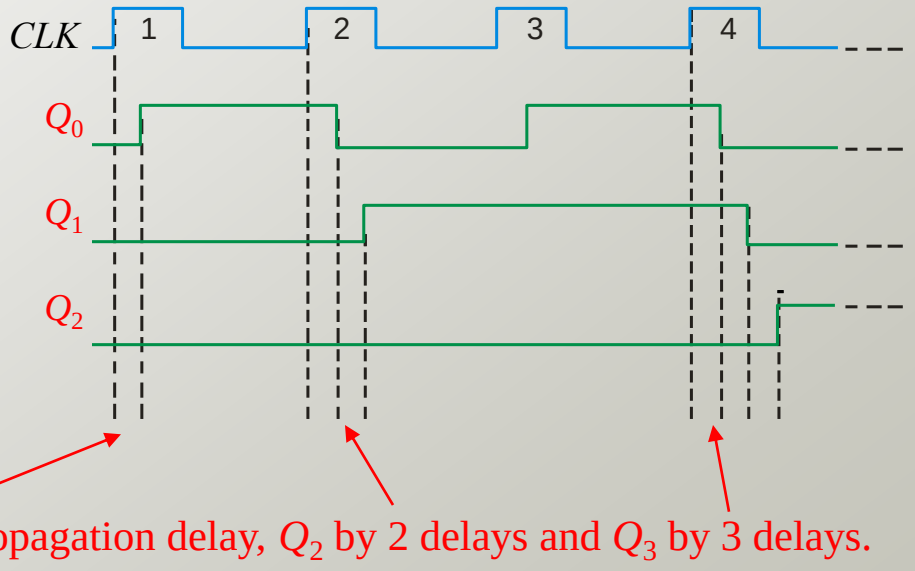


Asynchronous binary counter

Propagation Delay

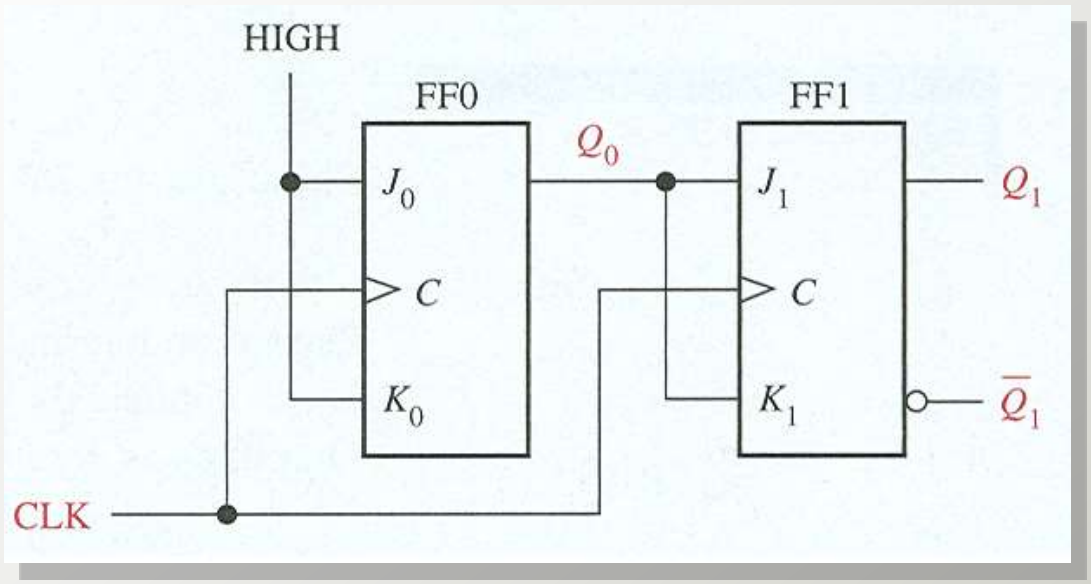
Asynchronous counters are sometimes called **ripple** counters, because the stages do not all change together. For certain applications requiring high clock rates, this is a major disadvantage.

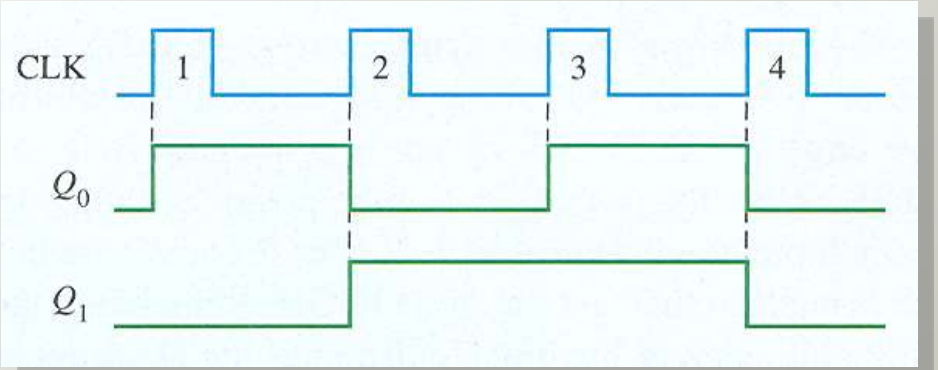
Notice how delays are cumulative as each stage in a counter is clocked later than the previous stage.



Synchronous binary counter

2-bit Synchronous binary counter

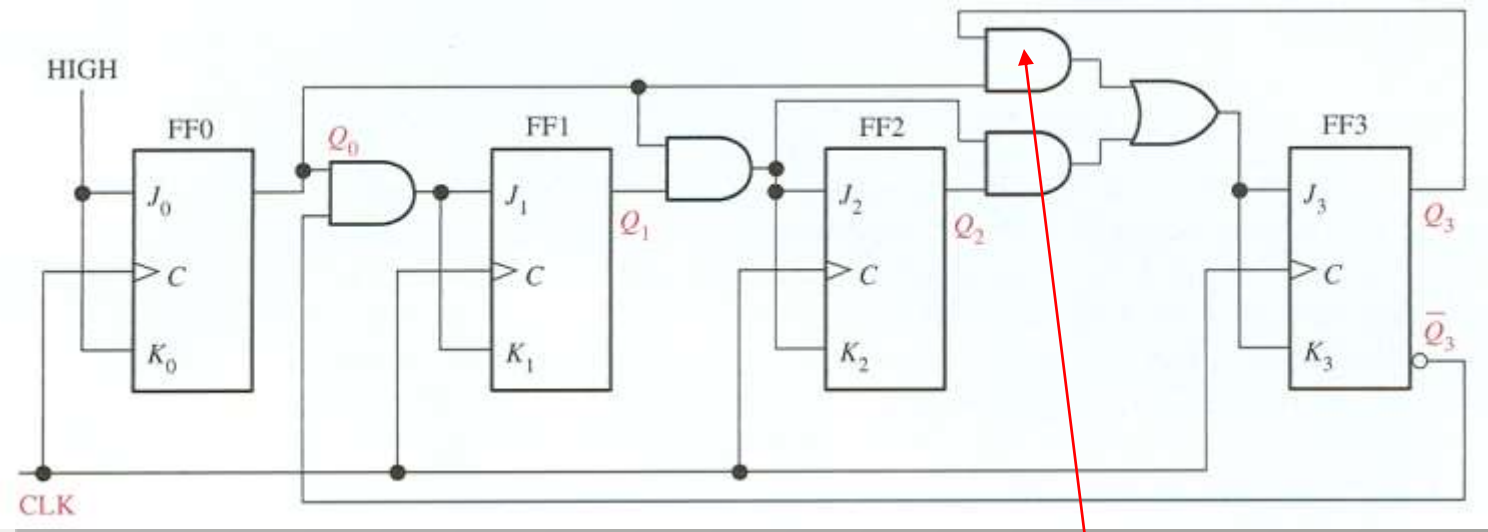


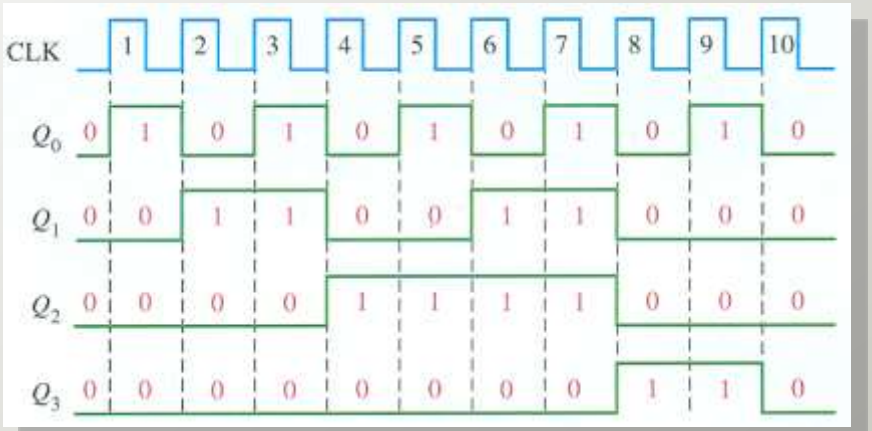


Inputs			Outputs		
J	*K*	CLK	*Q*	$\overline{Q}$	Comments
0	0	↑	Q_0	$\overline{Q}_0$	No change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	$\overline{Q}_0$	Q_0	Toggle

Synchronous binary counter

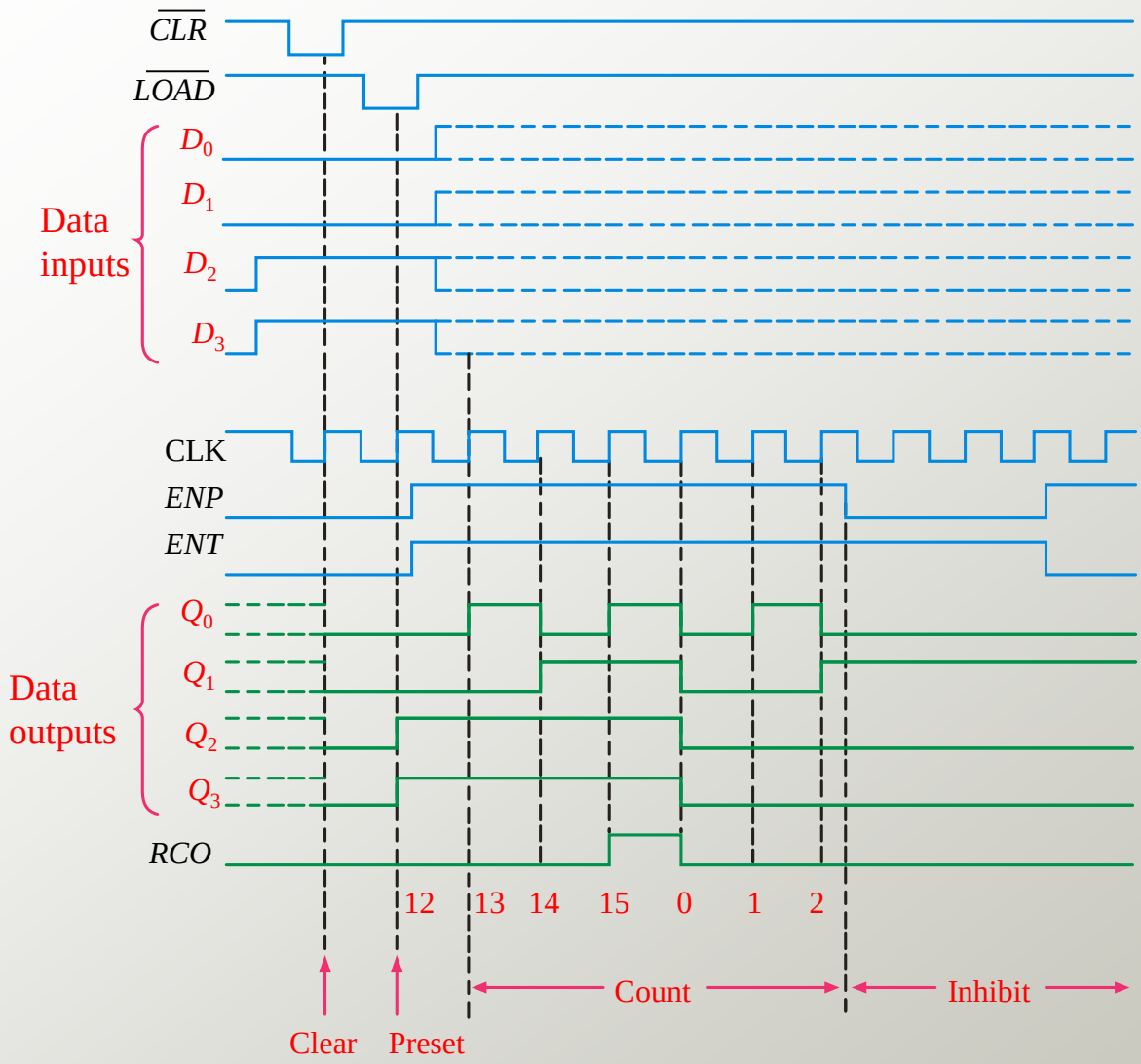
Synchronous decade counter

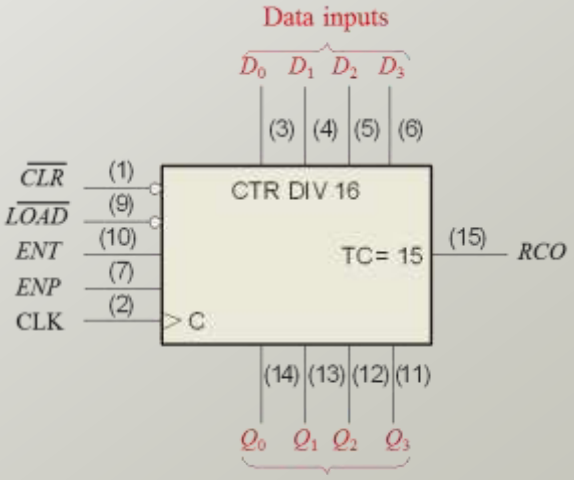




This gate detects 1001, and causes FF3 to toggle on the next clock pulse. FF0 toggles on every clock pulse. Thus, the count starts over at 0000.

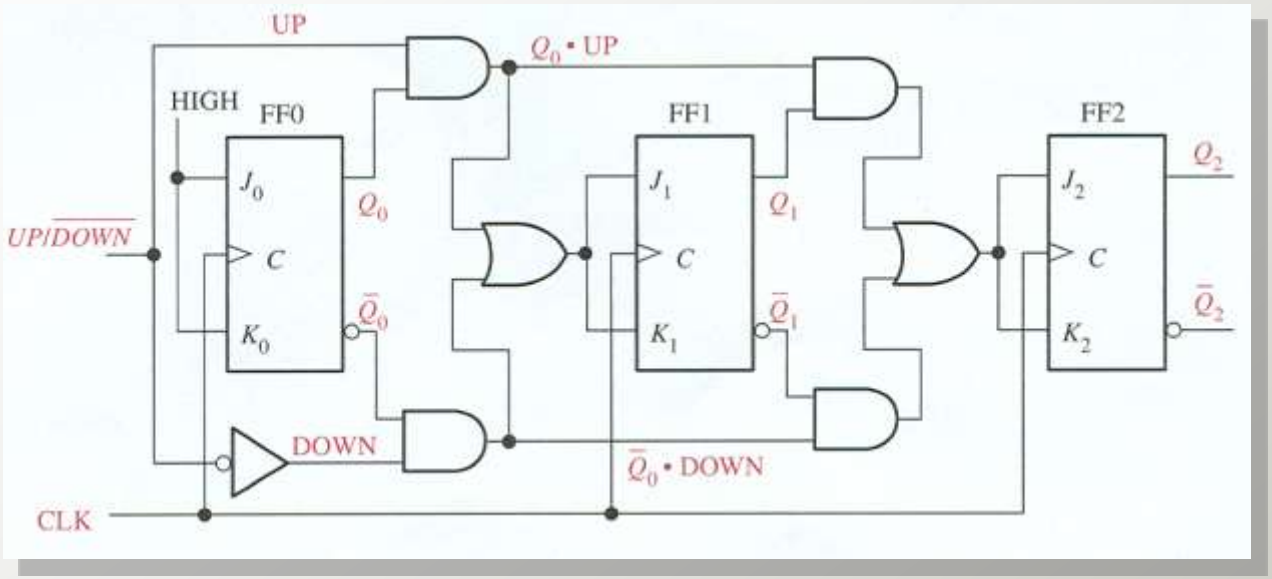
A 4-bit Synchronous Binary Counter





Synchronous binary counter

Up/Down Synchronous Counters



Up counter

Q1	Q0
0	0
0	1
1	0
1	1

Down counter

Q1	Q0	$\overline{Q0}$
0	0	1
1	1	0
1	0	1
0	1	0

An up/down counter is capable of progressing in either direction depending on a control input.

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	

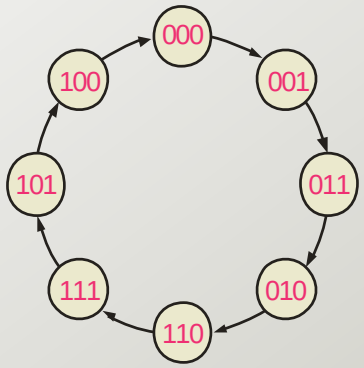
Synchronous binary counter

Synchronous Counter Design

Most requirements for synchronous counters can be met with available ICs. In cases where a special sequence is needed, you can apply a step-by-step design process.

Start with the desired sequence and draw a state diagram and next-state table. The gray code sequence from the text is illustrated:

State diagram:



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

Synchronous binary counter

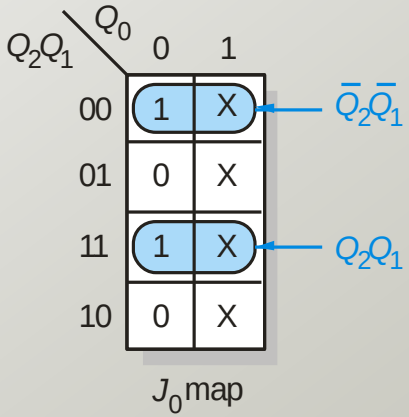
Synchronous Counter Design

The J-K transition table lists all combinations of present output (Q_N) and next output (Q_{N+1}) on the left. The inputs that produce that transition are listed on the right.

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

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

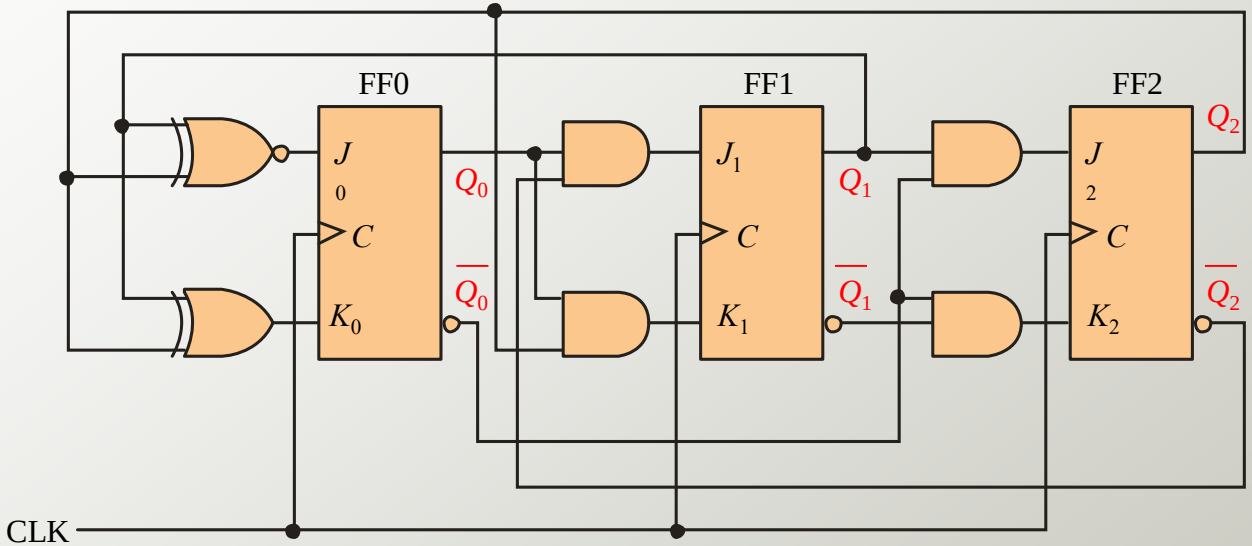
An example of the J_0 map is:



Present State Q_2	Q_1	Q_0	Next State 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

Synchronous binary counter

Synchronous Counter Design



The logic for each input is read and the circuit is constructed. The slide shows the circuit for the gray code counter…

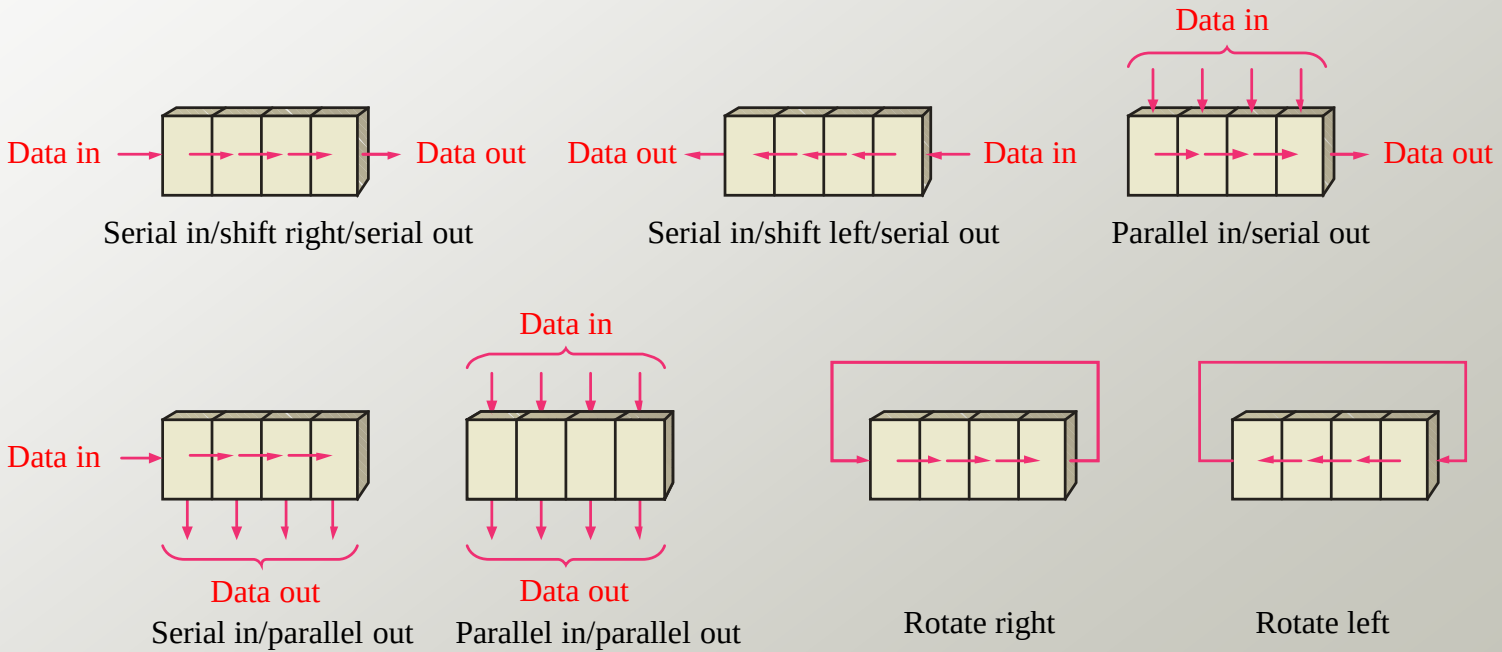
Digital Fundamentals

Shift Registers

Shift Registers

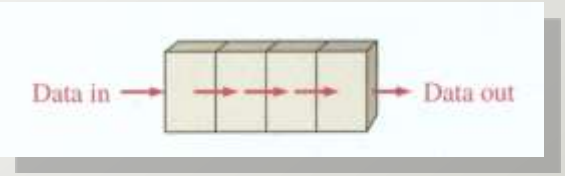
Basic Shift Register Operations

A shift register is an arrangement of flip-flops with important applications in storage and movement of data. Some basic data movements are illustrated here.



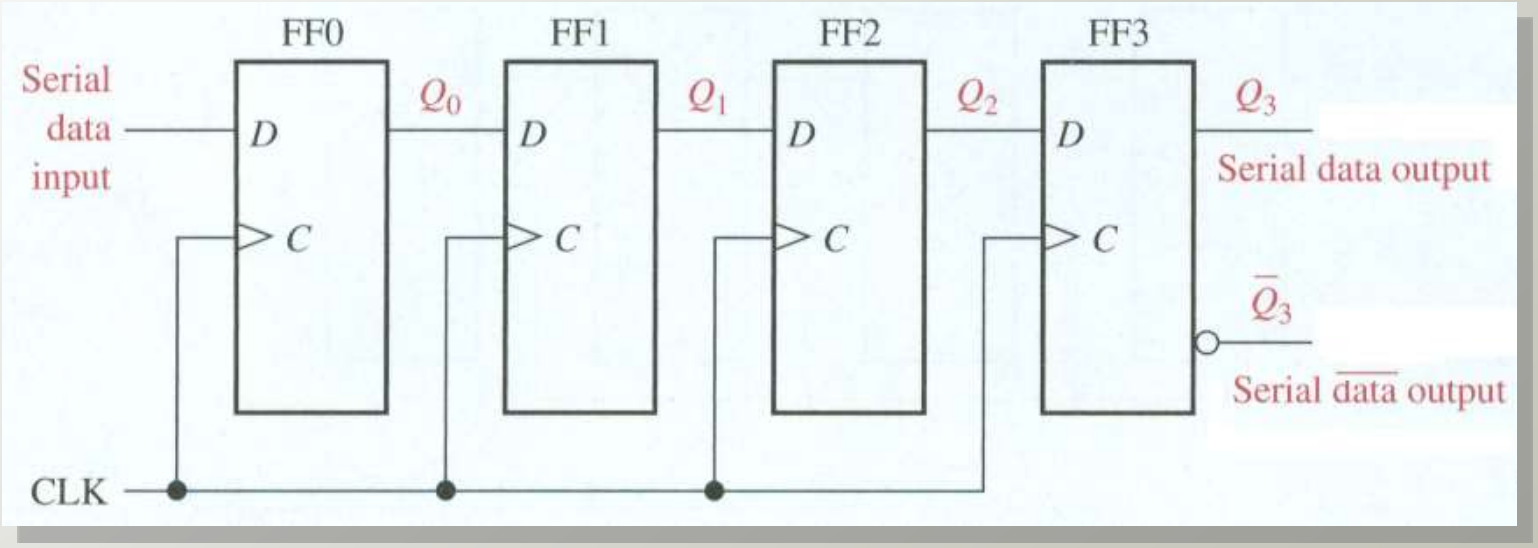
Shift Registers

Serial-in/Serial out Shift Register



Shift registers are available in IC form or can be constructed from discrete flip-flops.

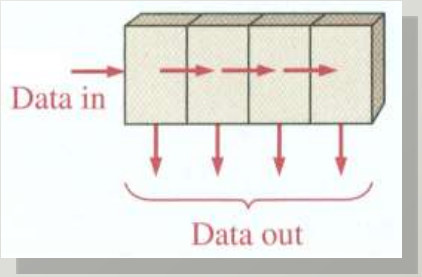
Each clock pulse will move an input bit to the next flip-flop. For example, a 1 is shown as it moves across.

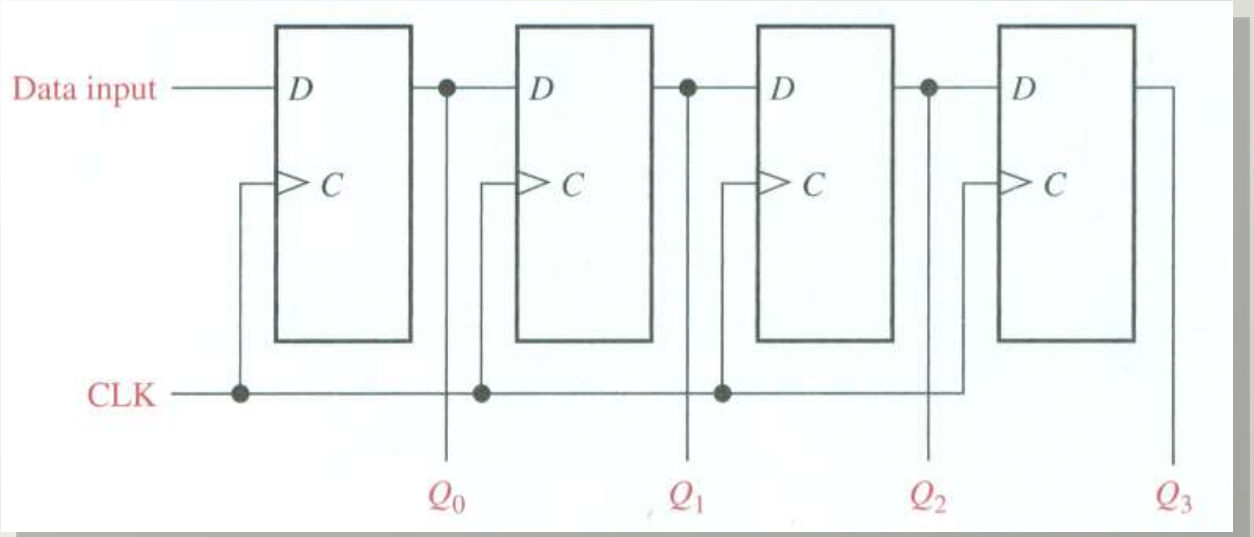


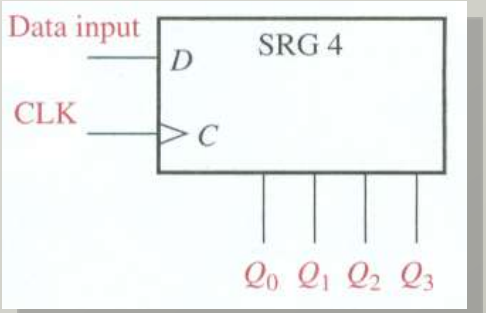
Shift Registers

Serial in/Parallel out Shift Register

An application of shift registers is conversion of serial data to parallel form.

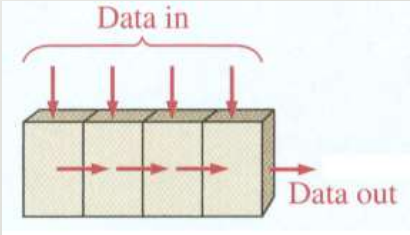




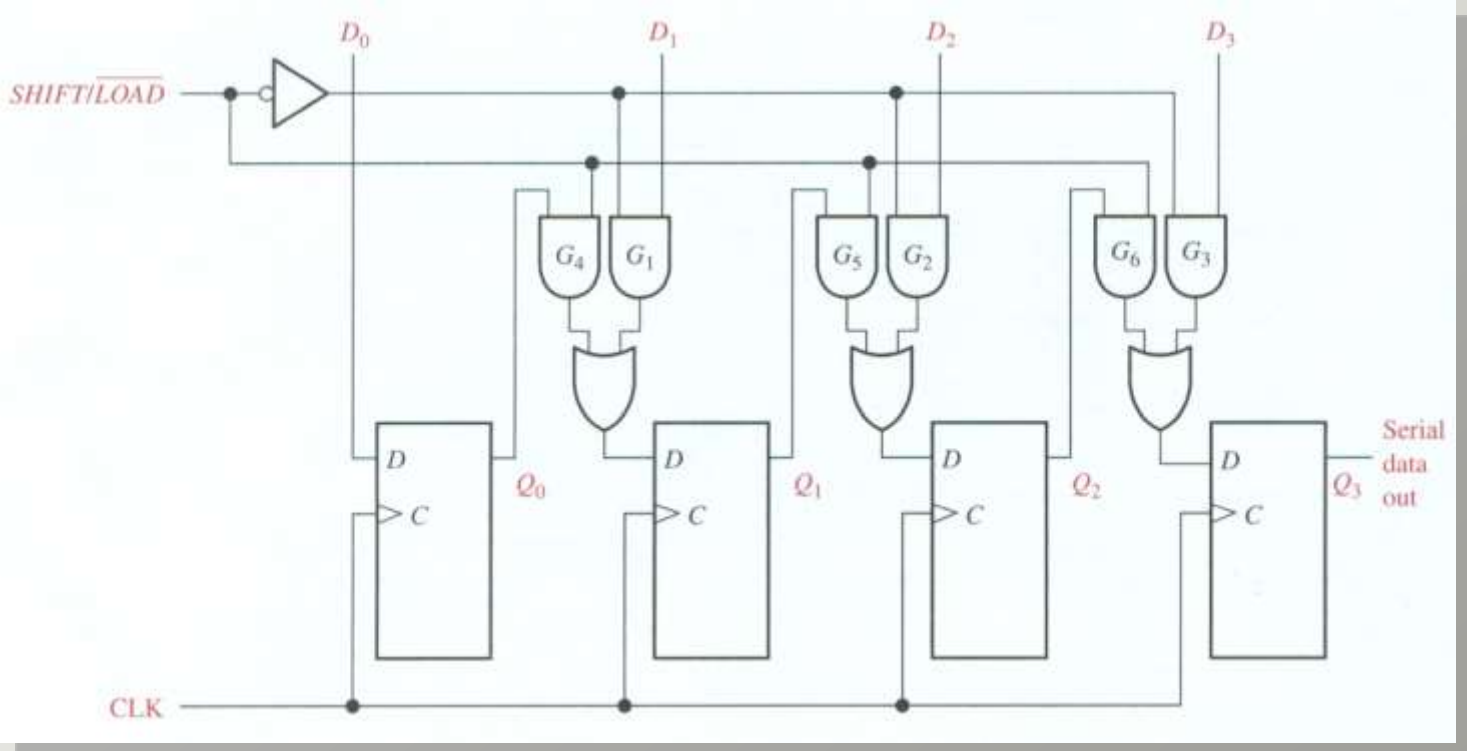


Shift Registers

Parallel in/Serial out Shift Register

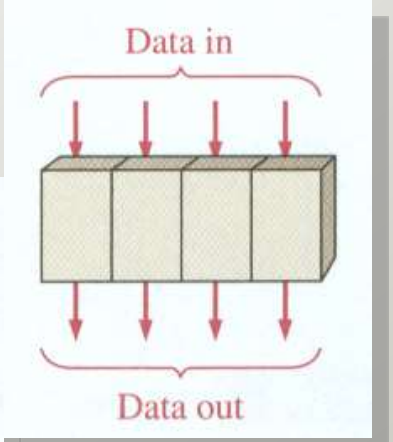


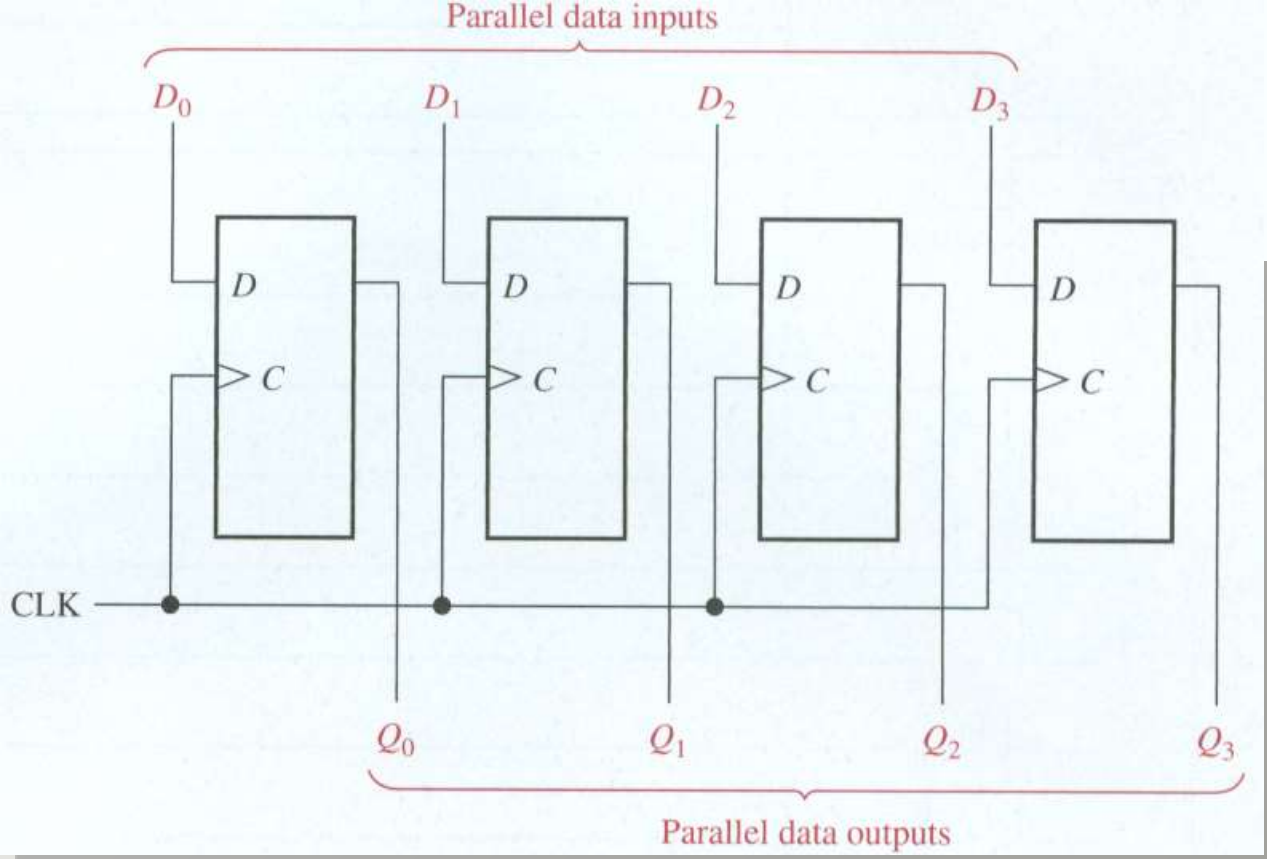
An application of shift registers is conversion of parallel data to serial form.



Shift Registers

Parallel in/Parallel out Shift Register

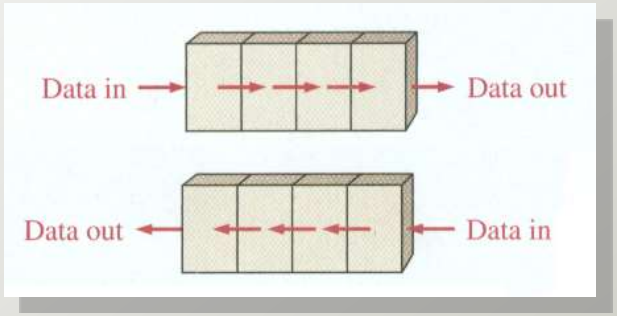


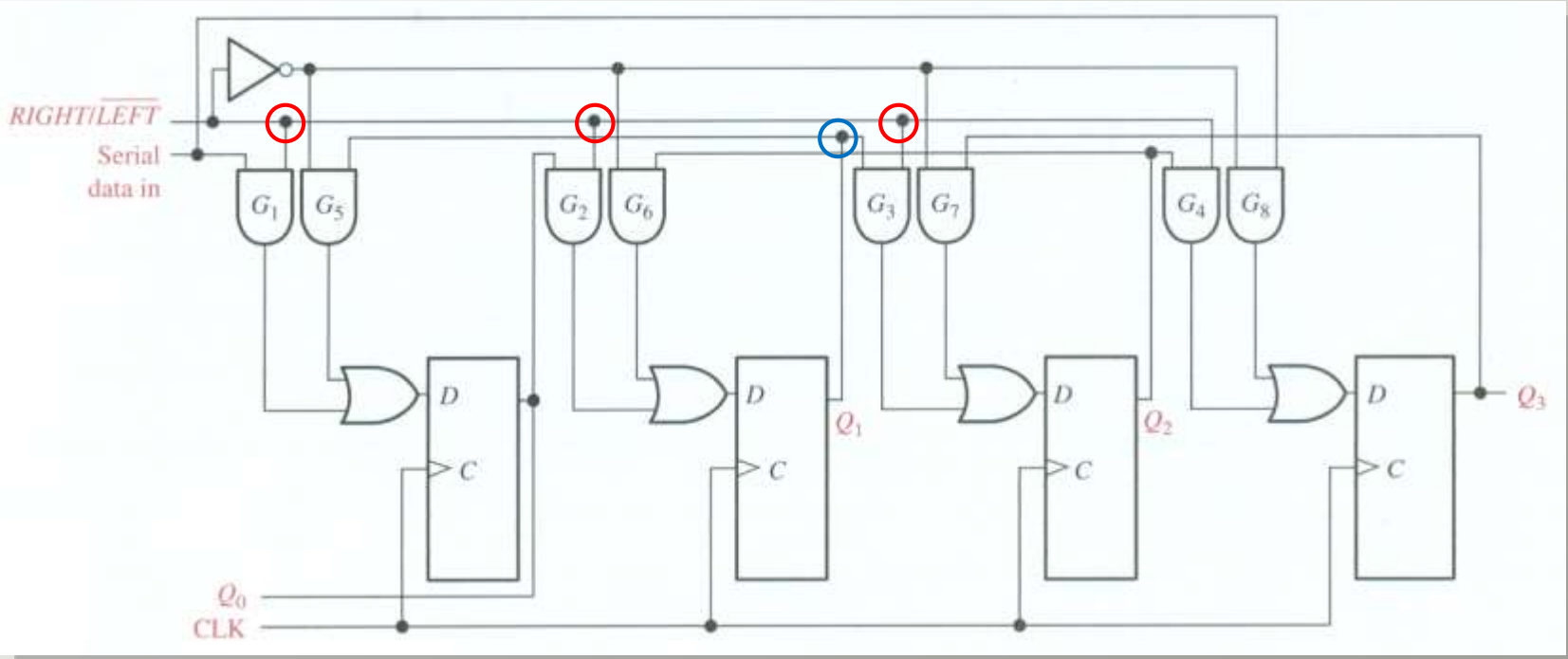


Shift Registers

Bidirectional Shift Register

Bidirectional shift registers can shift the data in either direction using a *RIGHT/$\overline{LEFT}$* input.





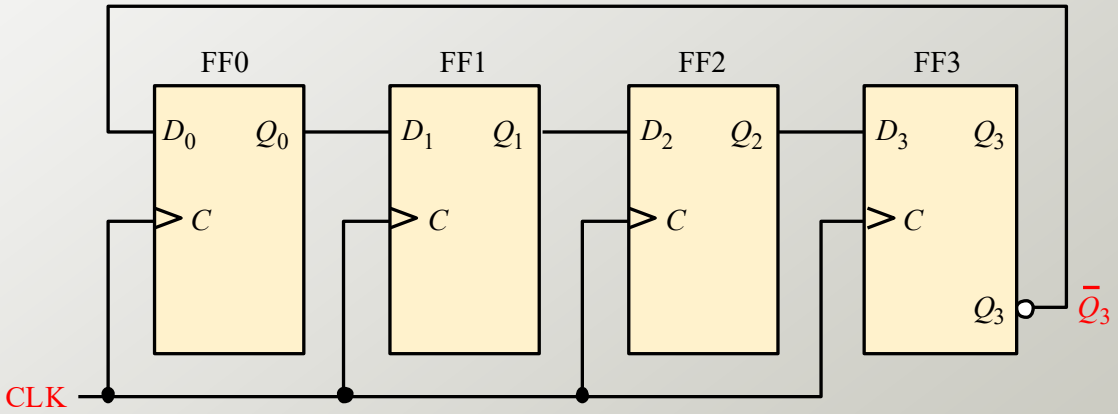
Shift Registers

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

Shift Register Counters

Shift registers can form useful counters by recirculating a pattern of 0's and 1's. Two important shift register counters are the *Johnson counter* and the *ring counter*.

The Johnson counter is useful when you need a sequence that changes by only one bit at a time but it has a limited number of states ($2n$, where n = number of stages).



CLK	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

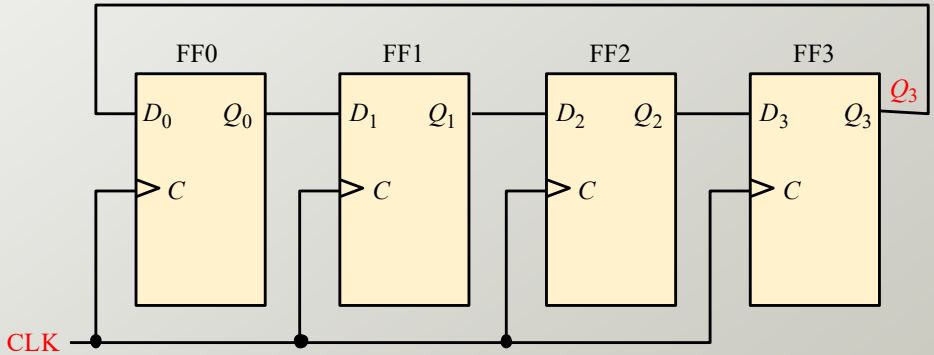
The Johnson counter can be made with a series of D flip-flops

Shift Registers

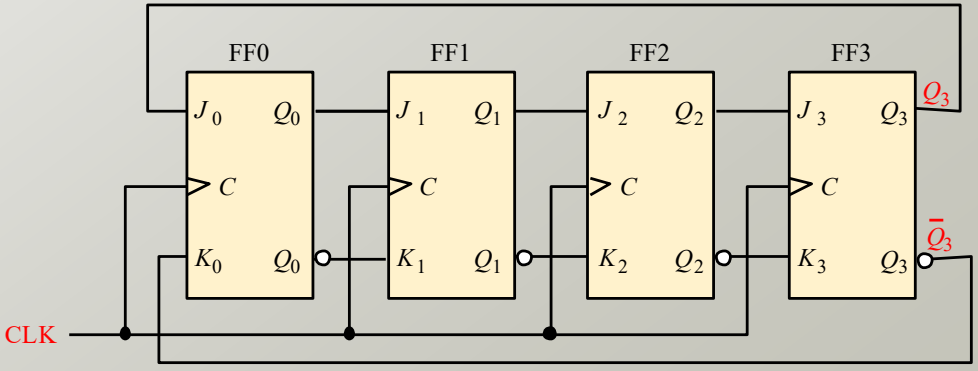
Ring Counter

The ring counter can also be implemented with either D flip-flops or J-K flip-flops.

Here is a 4-bit ring counter constructed from a series of D flip-flops.
Notice the feedback.



Like the Johnson counter, it can also be implemented with J-K flip flops.



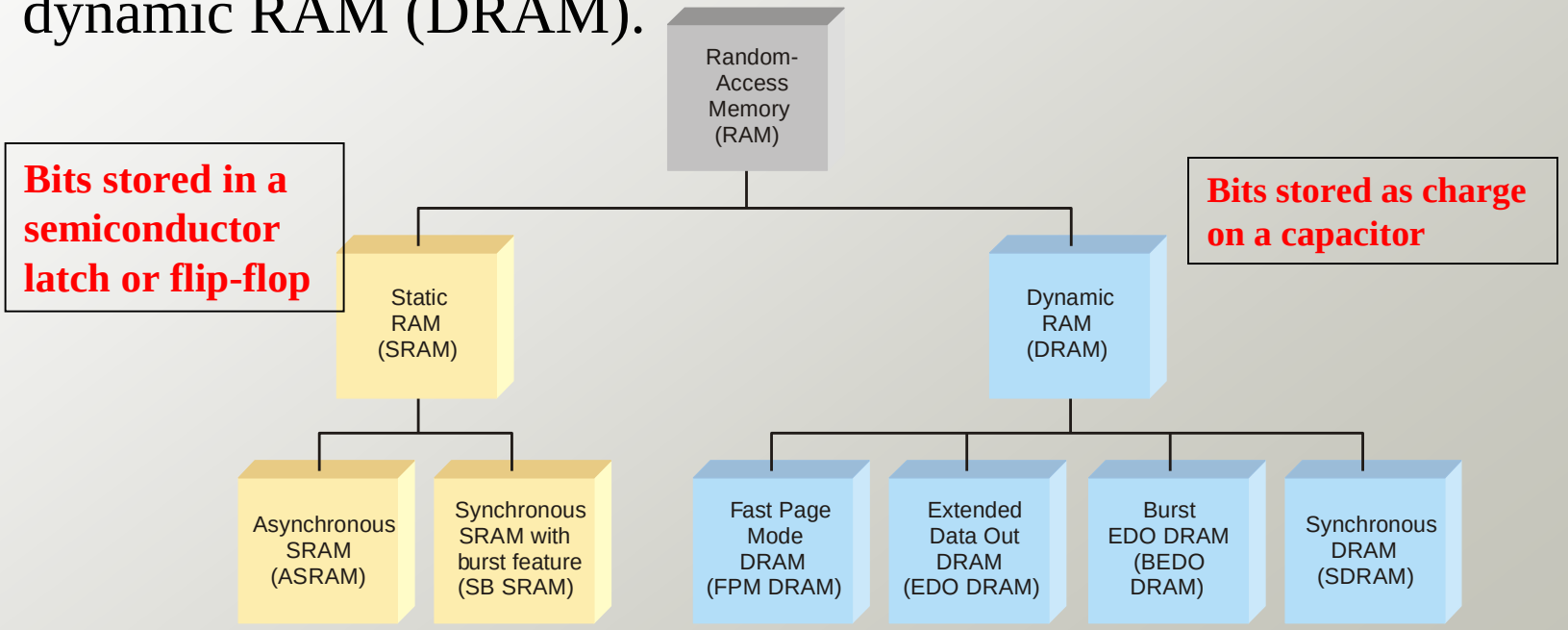
Digital Fundamentals

Applications

Applications

Random Access Memory

RAM is for temporary data storage. It is read/write memory and can store data only when power is applied, hence it is *volatile*. Two categories are static RAM (SRAM) and dynamic RAM (DRAM).

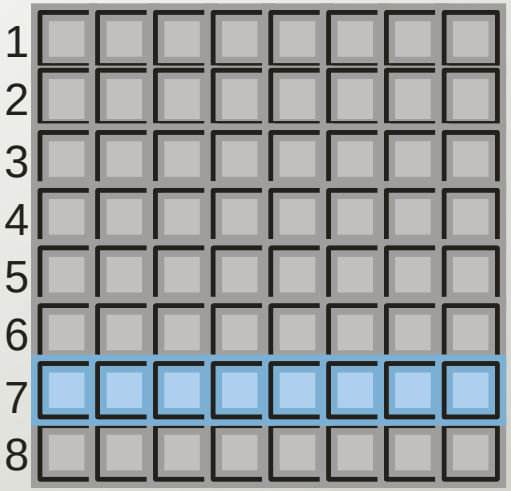


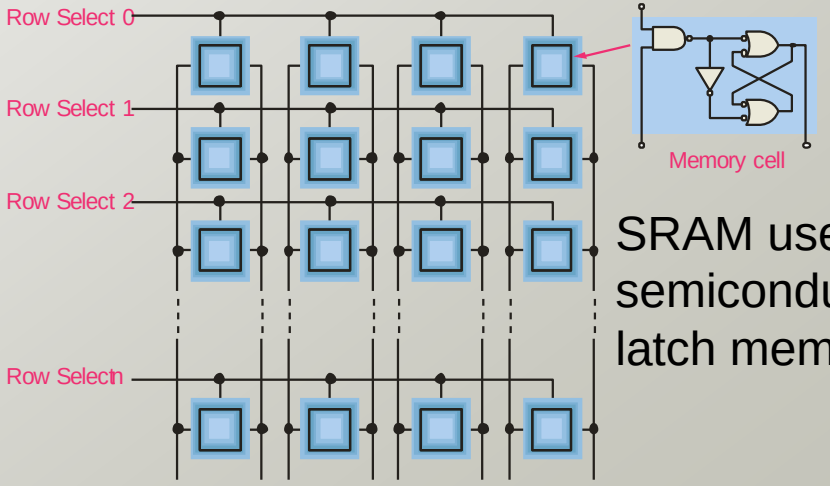
Applications

Memory Units

The location of a unit of data in a memory is called the **address**. In PCs, a byte is the smallest unit of data that can be accessed.

In a 2-dimensional array, a byte is accessed by supplying a row number. For example the blue byte is located in row 7.





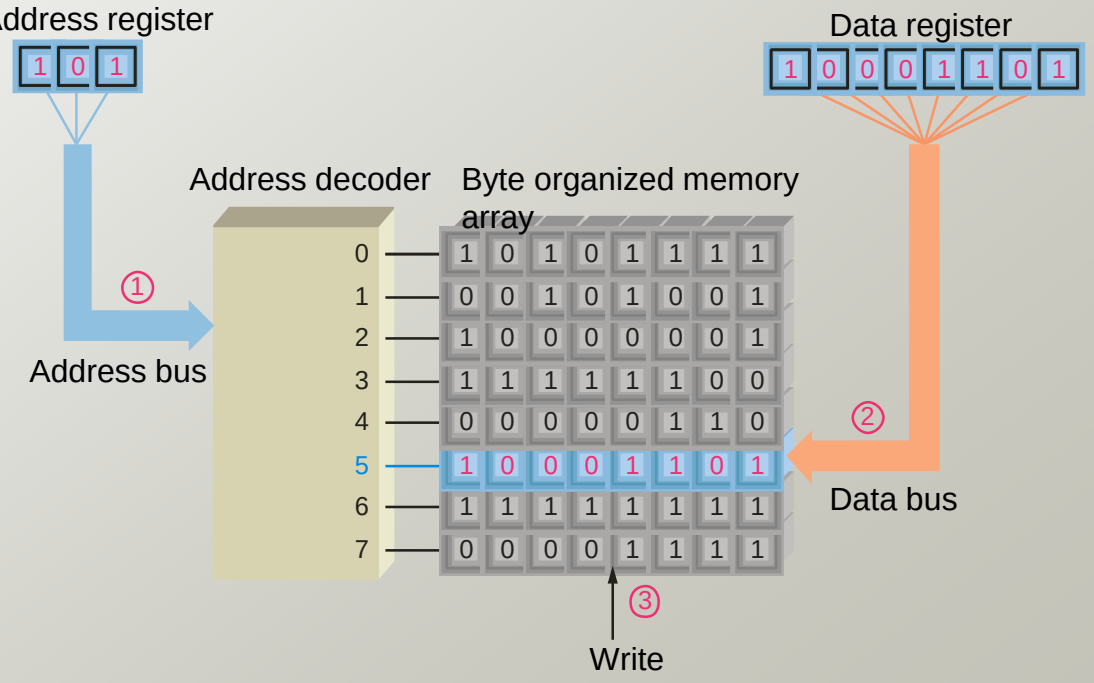
SRAM uses semiconductor latch memory cells

Applications

Write Operation

The two main memory operations are called **read** and **write**. A simplified write operation is shown in which new data overwrites the original data. Data moves *to* the memory.

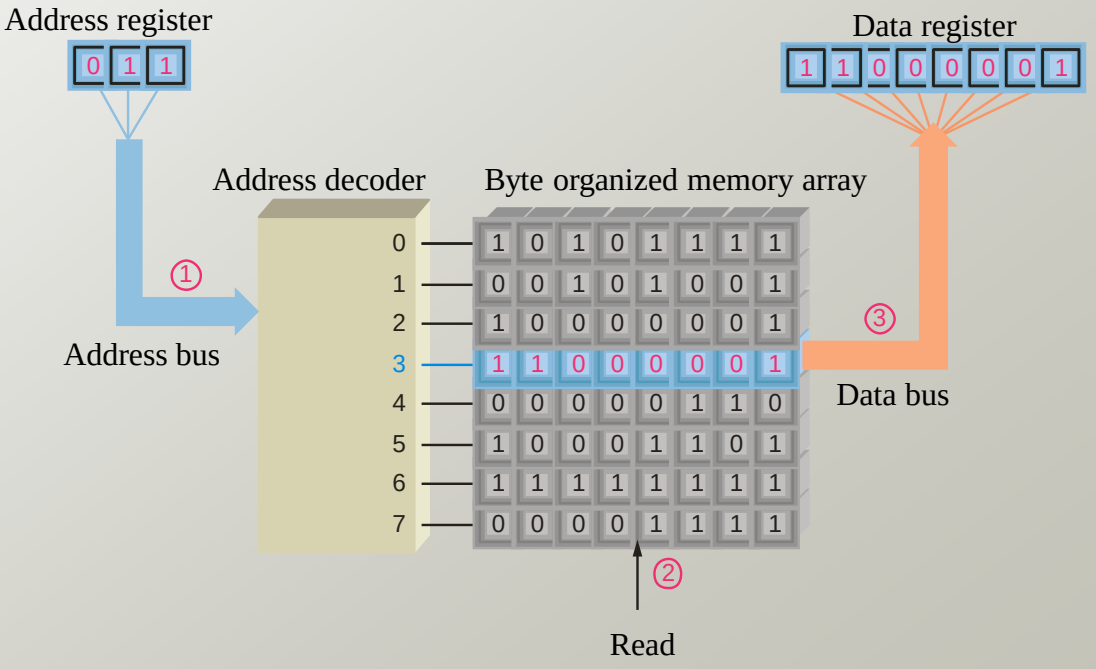
1. The address is placed on the address bus.
2. Data is placed on the data bus.
3. A write command is issued.



Applications

Read Operation

The read operation is actually a "copy" operation, as the original data is not changed. The data bus is a "two-way" path; data moves *from* the memory during a read operation.

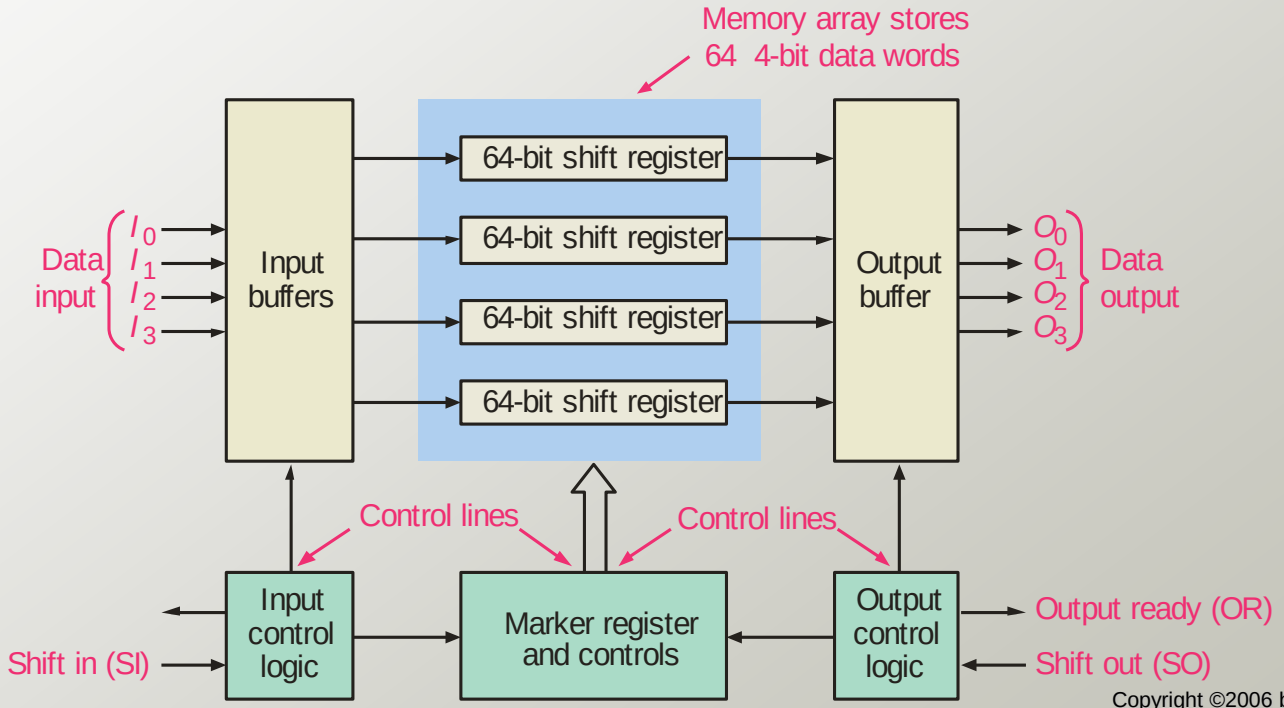


1. The address is placed on the address bus.
2. A read command is issued.
3. A copy of the data is placed in the data bus and shifted into the data register.

Applications

FIFO Memory

FIFO means first in-first out. This type of memory is basically an arrangement of shift registers. It is used in applications where two systems communicate at different rates.



Applications

Programmable Logic

Programmable Logic Devices (PLDs) are ICs with a large number of gates and flip flops that can be configured with basic software to perform a specific logic function or perform the logic for a complex circuit. Major types of PLDs are:

SPLD: (Simple PLDs) are the earliest type of array logic used for fixed functions and smaller circuits with a limited number of gates. (**The PAL and GAL are both SPLDs**).

CPLD: (Complex PLDs) are multiple SPLDs arrays and inter-connection arrays on a single chip.

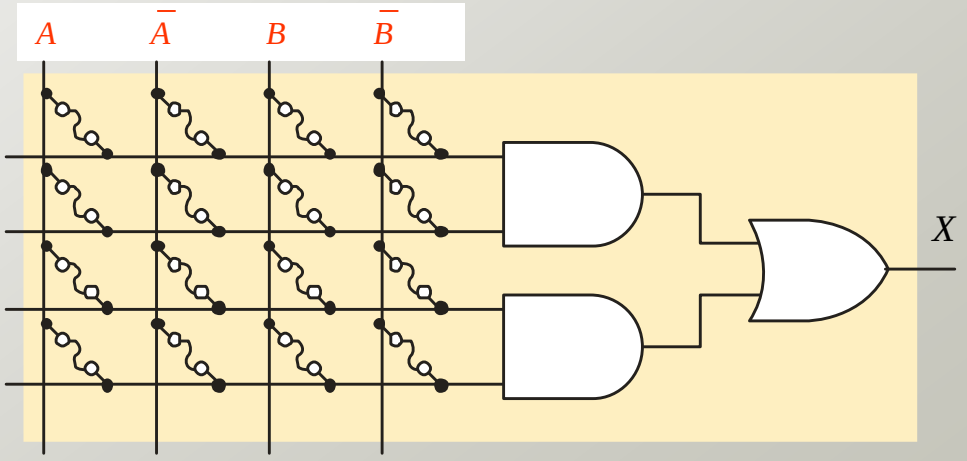
FPGA: (Field Programmable Gate Array) are a more flexible arrangement than CPLDs, with much larger capacity.

Applications

PALs and GALs

All PLDs contain arrays. Two important SPLDs are **PALs** (Programmable Array Logic) and **GALs** (Generic Array Logic). A typical array consists of a matrix of conductors connected in rows and columns to AND gates.

PALs have a one time programmable (OTP) array, in which fuses are permanently blown, creating the product terms in an AND array.



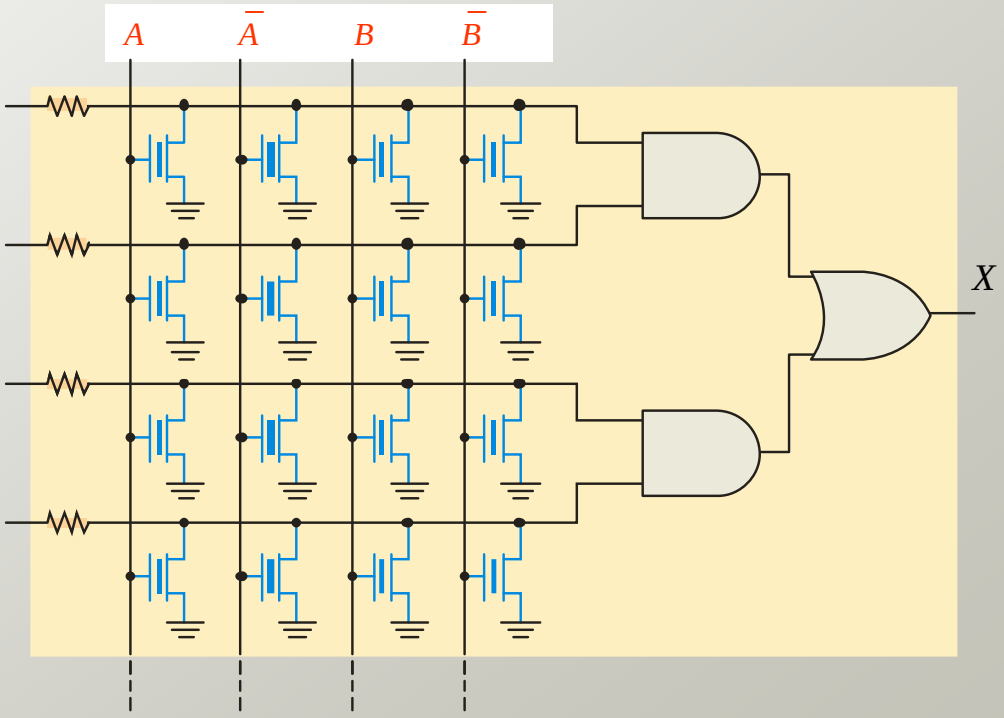
Simplified AND-OR array

Applications

PALs and GALs

The GAL (Generic Array Logic) is similar to a PAL but can be reprogrammed. For this reason, they are useful for new product development (prototyping) and for training purposes.

GALs were developed by Lattice Semiconductor. They are high speed, extremely fast devices and can interface with both 3.3 V or 5 V logic signals.

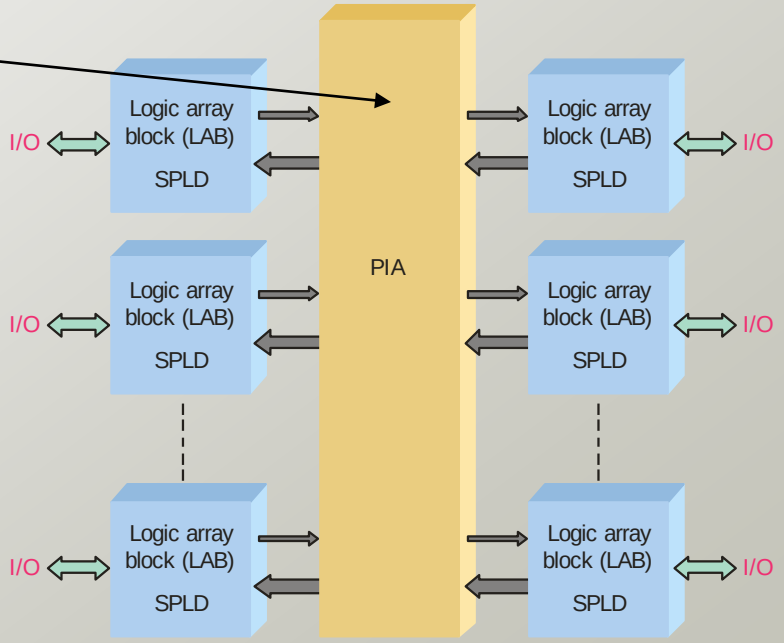


Applications

CPLDs

A complex programmable logic device (**CPLD**) has multiple logic array blocks (**LAB**s) that are actually SPLDs on a single IC. LABs are connected via a programmable interconnect array (**PIA**). Various CPLDs have different structures for these elements.

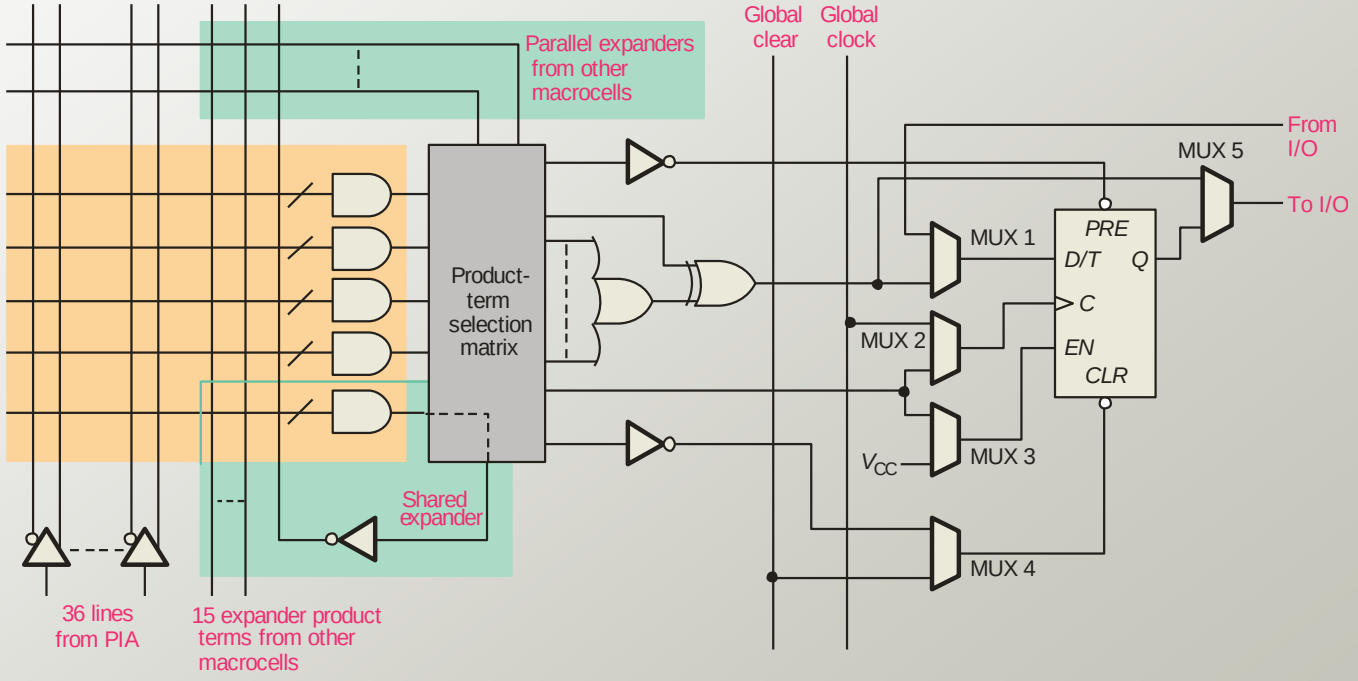
The PIA is the interconnection between the LABs. Logic is fitted to the CPLD and routing is determined by a high-level programming language called a hardware description language (HDL).



Applications

Macrocells

In addition to combination logic, some **macrocells** have registered outputs available (using programmable flip-flops). This allows the CPLD to perform sequential logic.

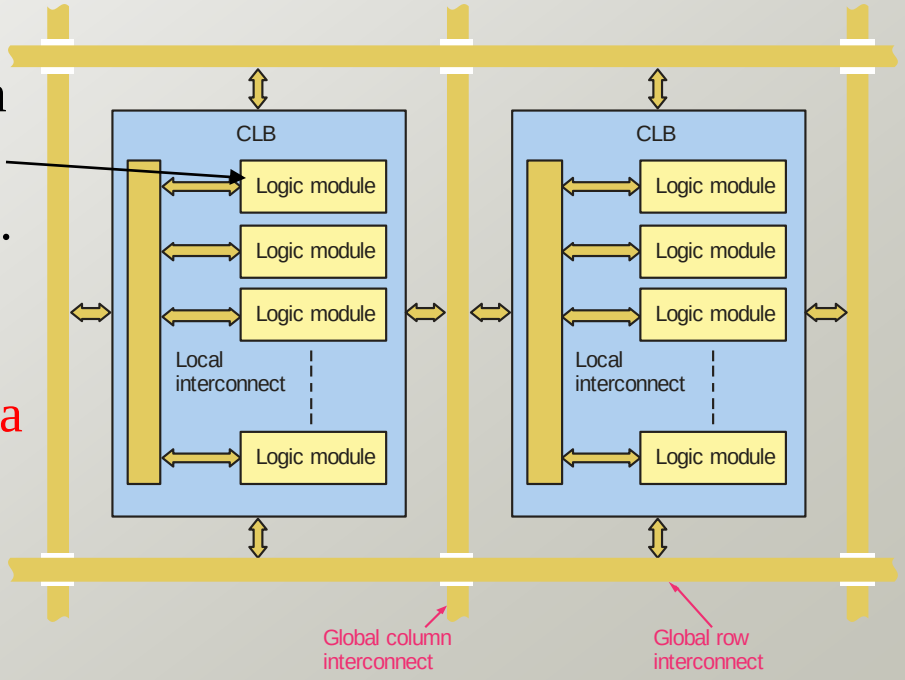


Applications

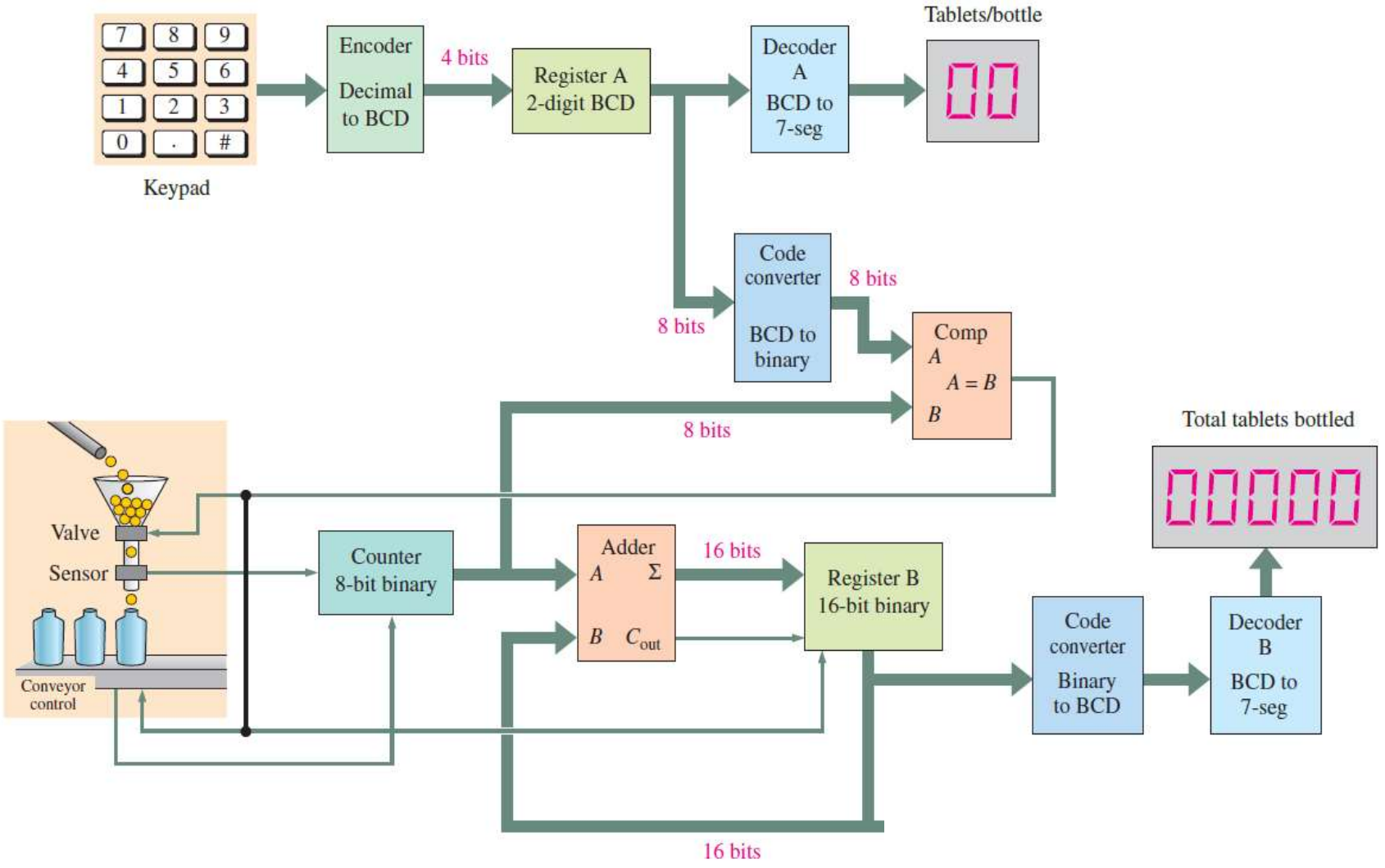
FPGAs

A field programmable gate array (**FPGA**) uses a different architecture than a CPLD. The configurable logic block (**CLB**) is the basic element which is replicated many times.

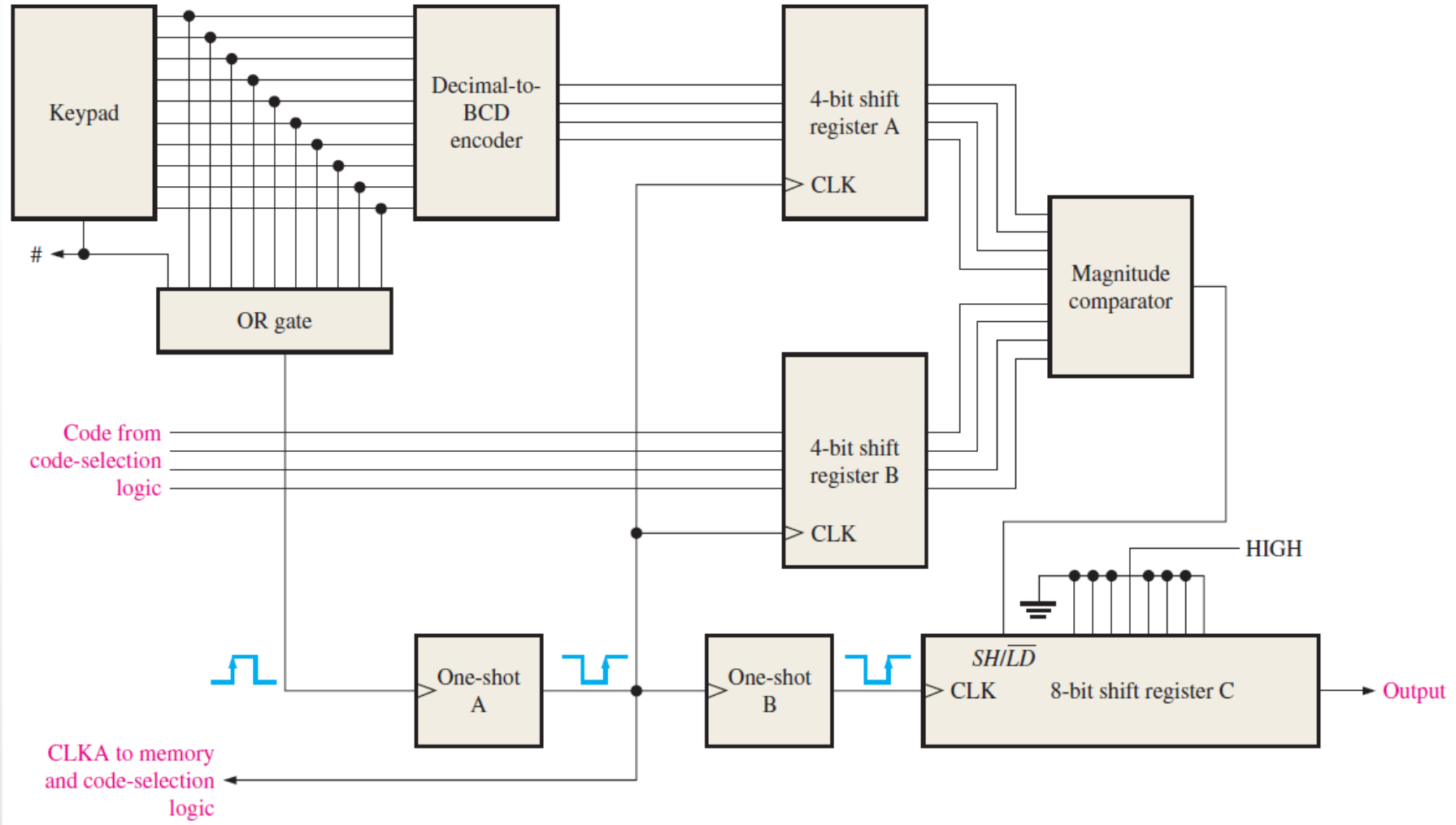
CLBs are arranged in a row and column structure. Within the CLBs are **logic modules** joined by local interconnects. Generally, the logic modules are composed of a look-up table (LUT), a flip-flop, and a MUX that can be used to bypass the flip-flop for strictly combinational logic.



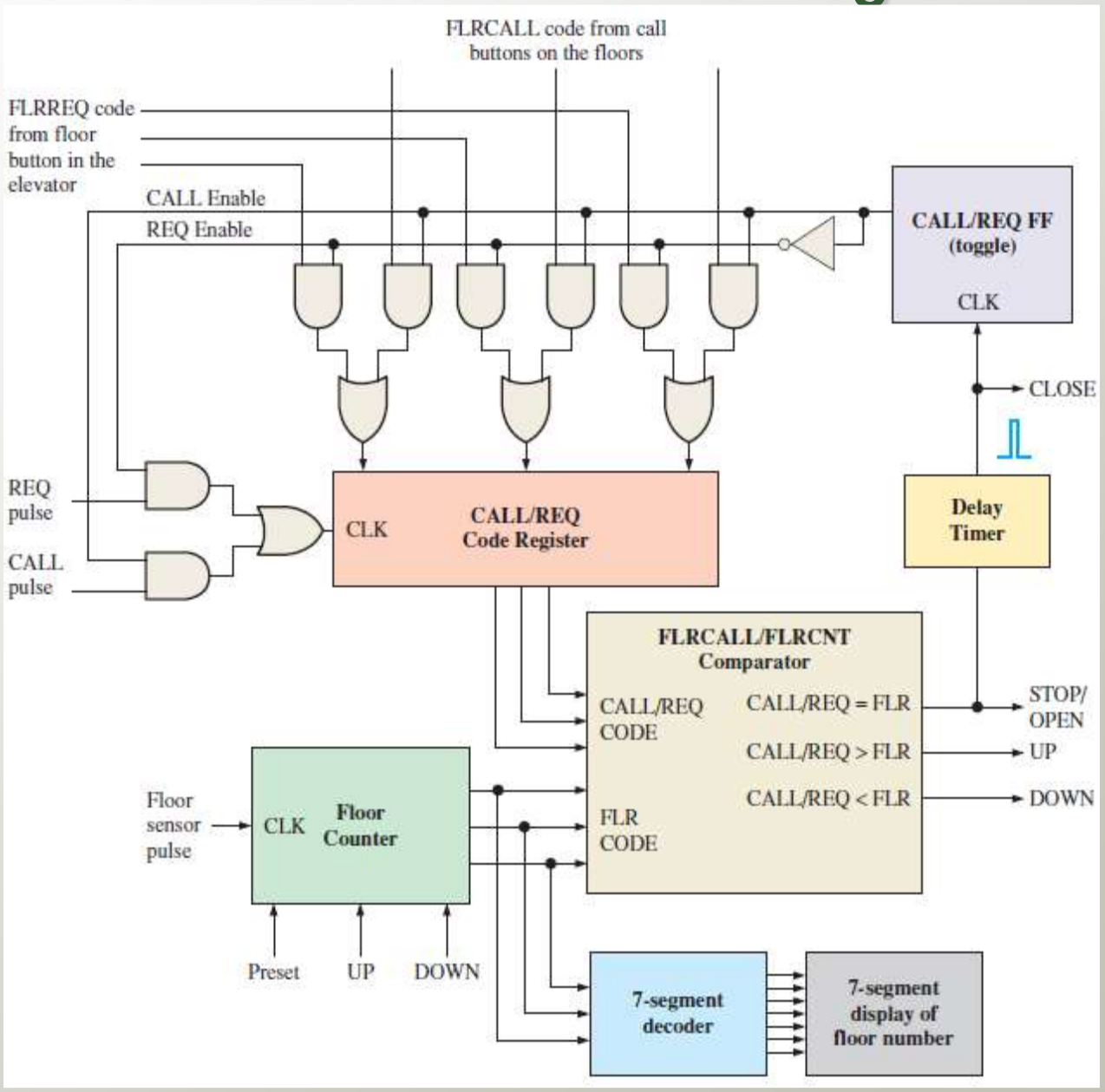
Applications : Tablet-bottling System



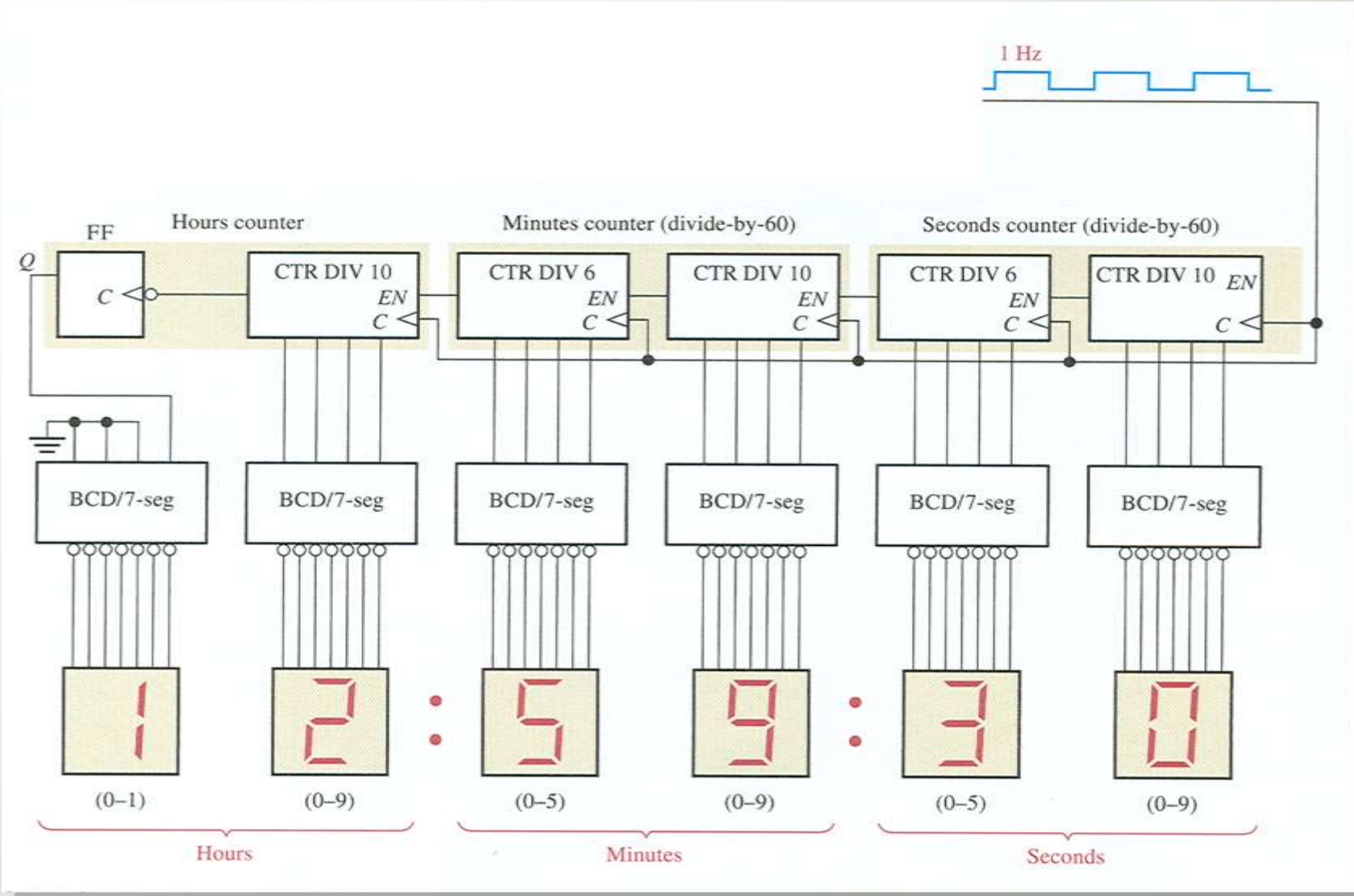
Applications: Security Code Logic with Keypad



Applications : Elevator controller logic



Applications : Digital Clock



Applications : Traffic Light Controller