Digital Computing Fundamentals

Volume-1

Basics of Digital Logic Design & Analysis

Compiled by

Mr. Hussain Saleem

Assistant Professor (Sr.)

Department of Computer Science, UBIT,
University of Karachi, Pakistan.

15th April 2017

This material is extracted from

Digital Fundamentals

Tenth Edition

Thomas L. Floyd

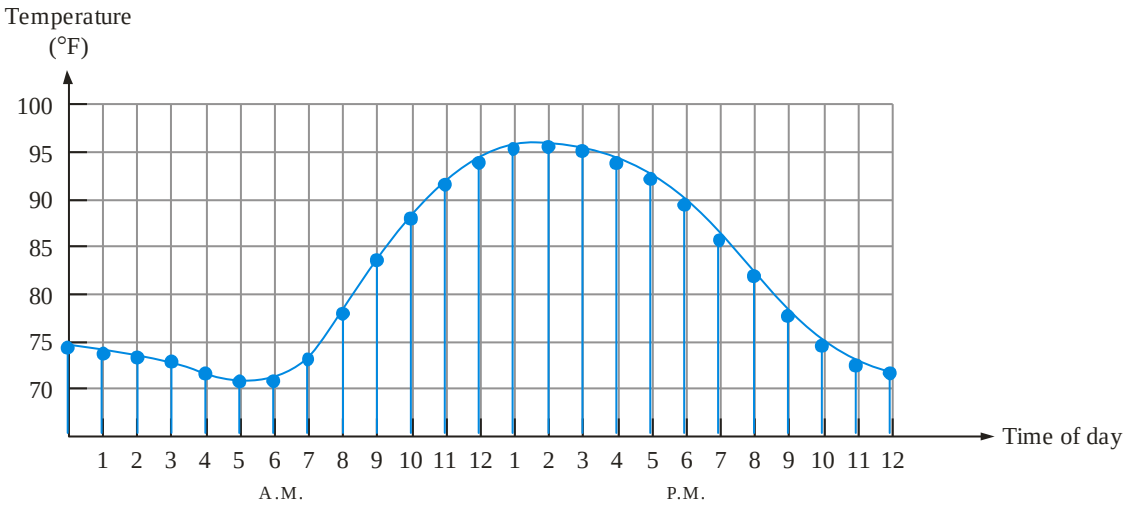
Courtesy of Pearson Education Inc.

Contents

Introduction to Digital Concepts

Analog Quantities

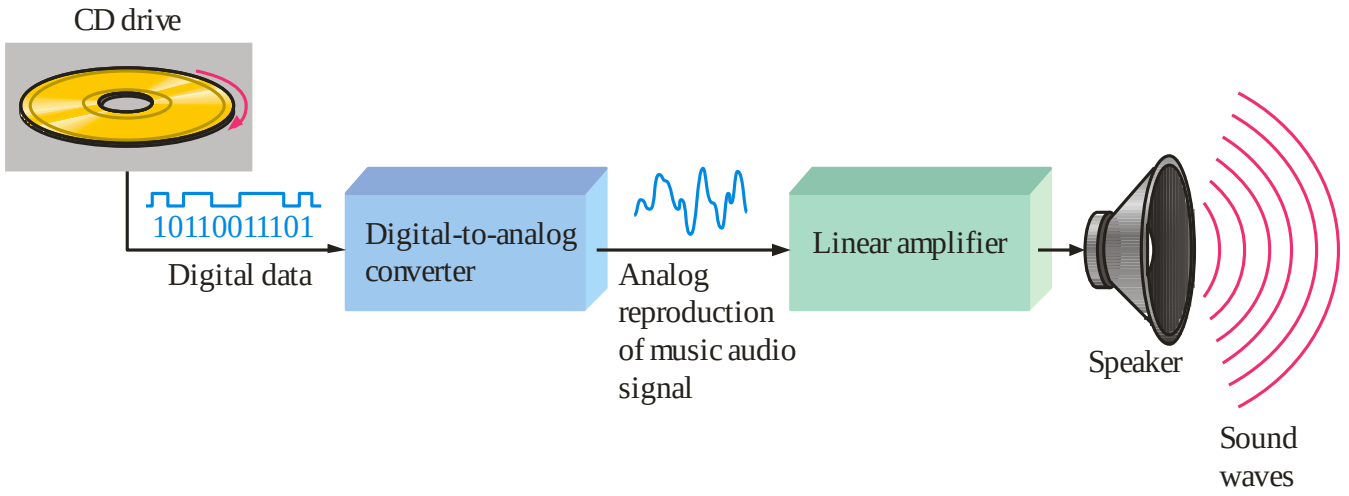
Most natural quantities that we see are **analog** and vary continuously. Analog systems can generally handle higher power than digital systems.



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

Analog and Digital Systems

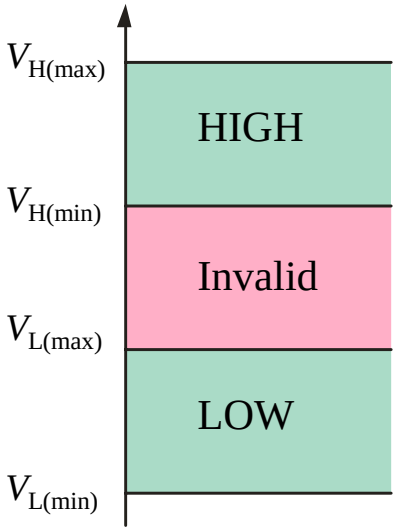
Many systems use a mix of analog and digital electronics to take advantage of each technology. A typical CD player accepts digital data from the CD drive and converts it to an analog signal for amplification.



Binary Digits and Logic Levels

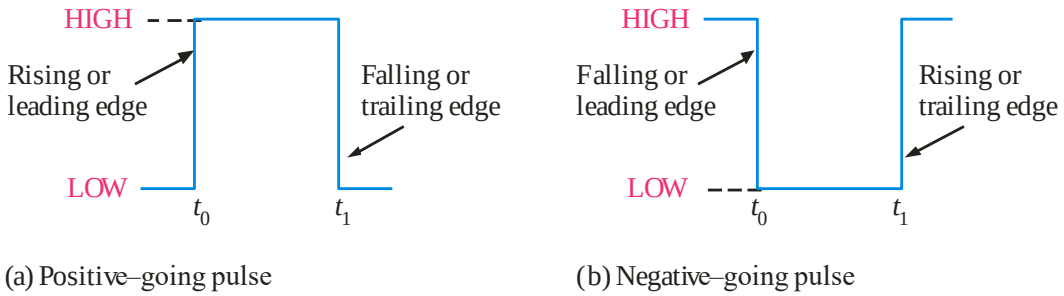
Digital electronics uses circuits that have two states, which are represented by two different voltage levels called HIGH and LOW. The voltages represent numbers in the binary system.

In binary, a single number is called a *bit* (for *b*inary dig*it*). A bit can have the value of either a 0 or a 1, depending on if the voltage is HIGH or LOW.



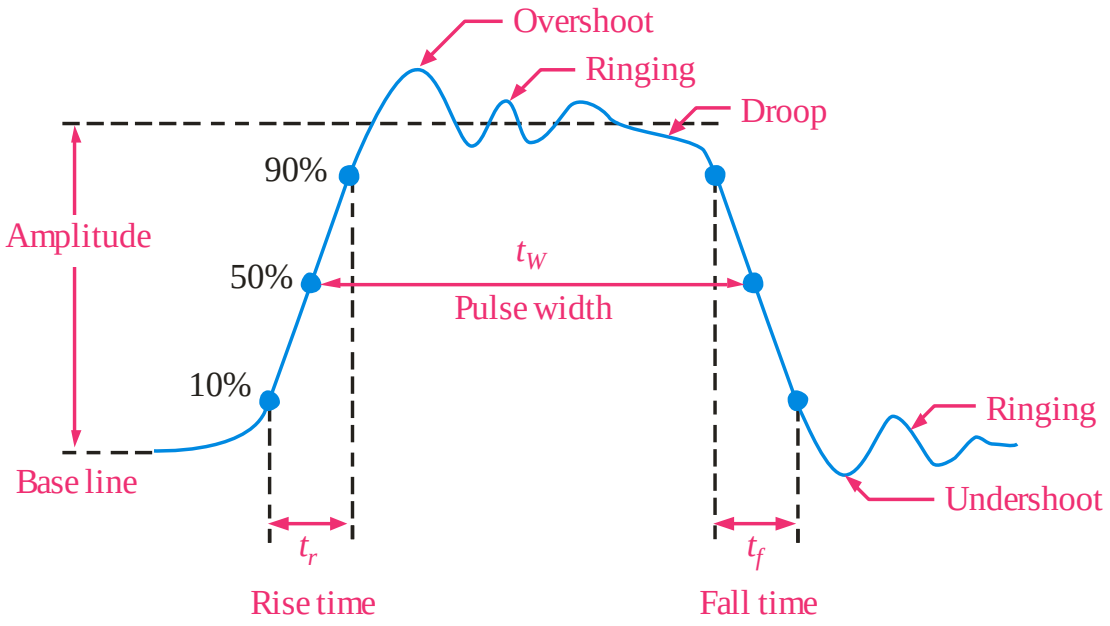
Digital Waveforms

Digital waveforms change between the LOW and HIGH levels. A positive going pulse is one that goes from a normally LOW logic level to a HIGH level and then back again. Digital waveforms are made up of a series of pulses.



Pulse Definitions

Actual pulses are not ideal but are described by the rise time, fall time, amplitude, and other characteristics.



Periodic Pulse Waveforms

Periodic pulse waveforms are composed of pulses that repeats in a fixed interval called the **period**. The **frequency** is the rate it repeats and is measured in hertz.

$$f = \frac{1}{T} \qquad T = \frac{1}{f}$$

The **clock** is a basic timing signal that is an example of a periodic wave.

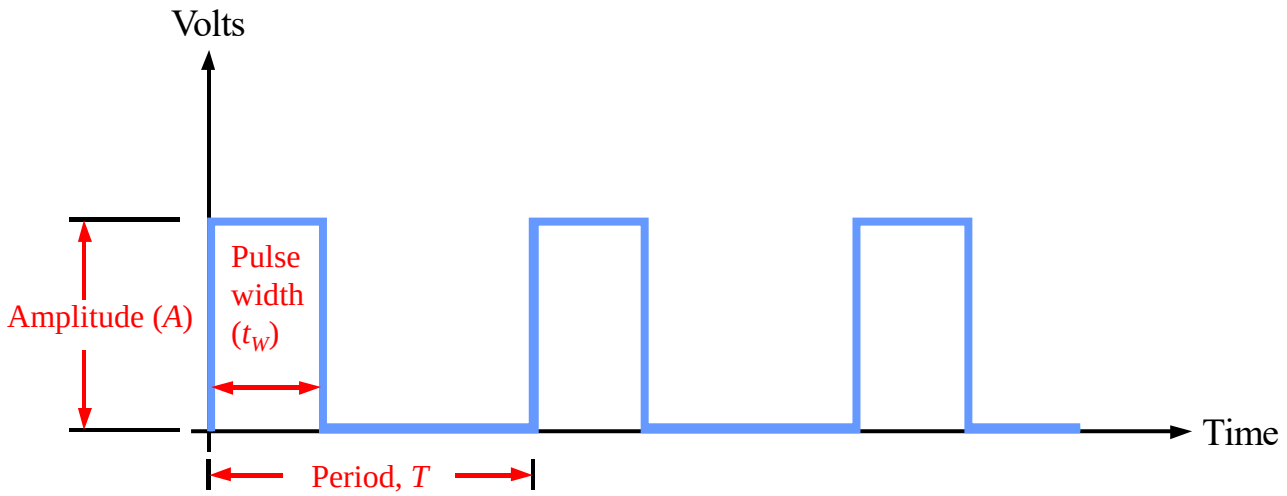
Example What is the period of a repetitive wave if $f = 3.2$ GHz?

Solution

$$T = \frac{1}{f} = \frac{1}{3.2\,\text{GHz}} = 313\ \text{ps}$$

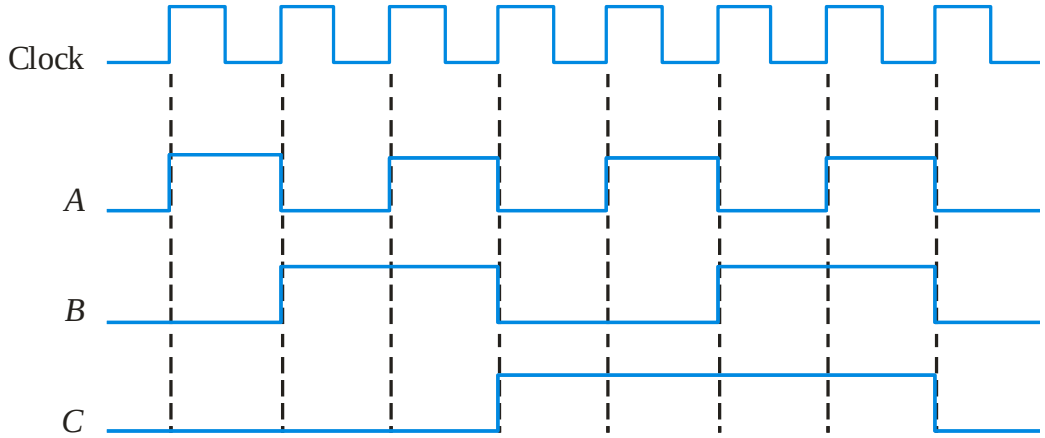
Pulse Definitions

In addition to frequency and period, repetitive pulse waveforms are described by the amplitude (A), pulse width (t_W) and duty cycle. Duty cycle is the ratio of t_W to T.



Timing Diagrams

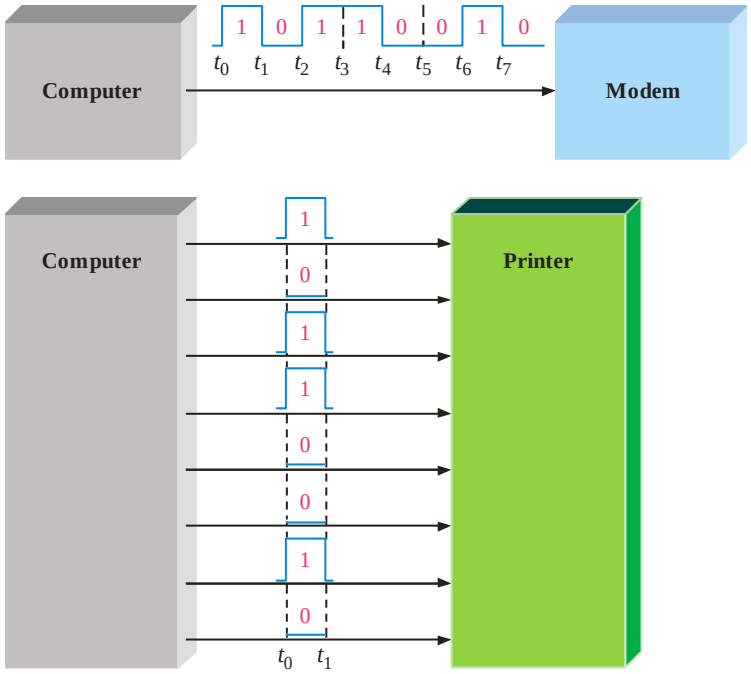
A timing diagram is used to show the relationship between two or more digital waveforms,



A diagram like this can be observed directly on a logic analyzer.

Serial and Parallel Data

Data can be transmitted by either serial transfer or parallel transfer.

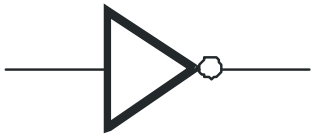


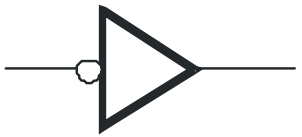
Basic Logic Functions

AND True only if ***all*** input conditions are true.

OR True only if ***one or more*** input conditions are true.

NOT Indicates the ***opposite*** condition.

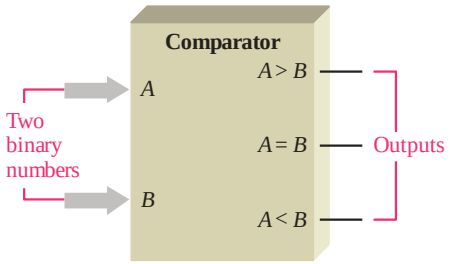




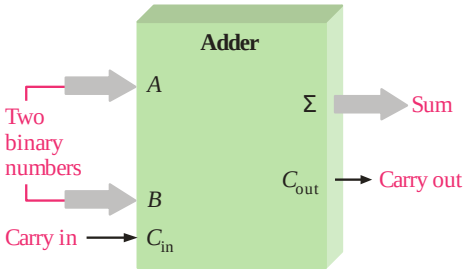
Basic System Functions

And, **or**, and **not** elements can be combined to form various logic functions. A few examples are:

The comparison function

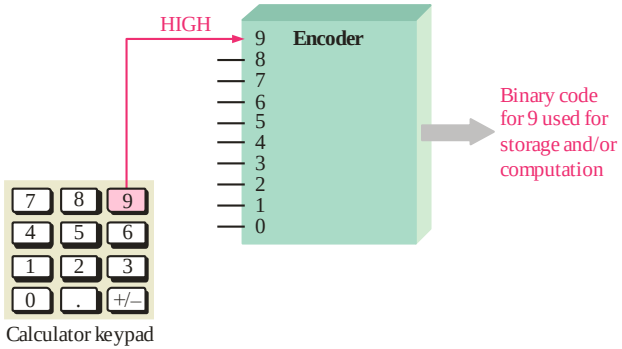


Basic arithmetic functions

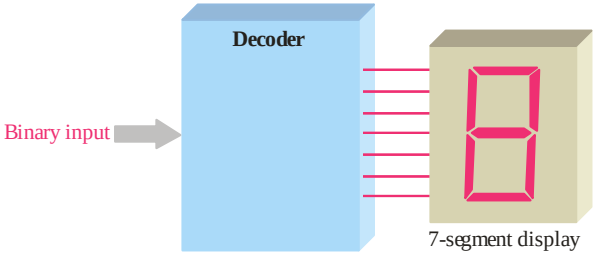


Basic System Functions

The encoding function

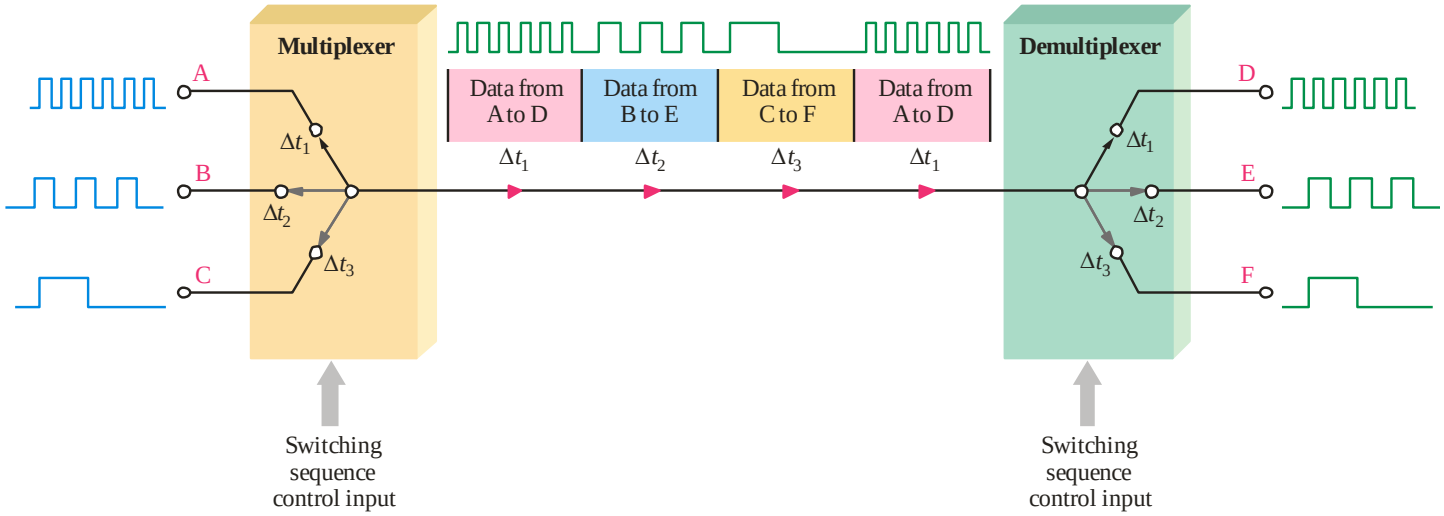


The decoding function



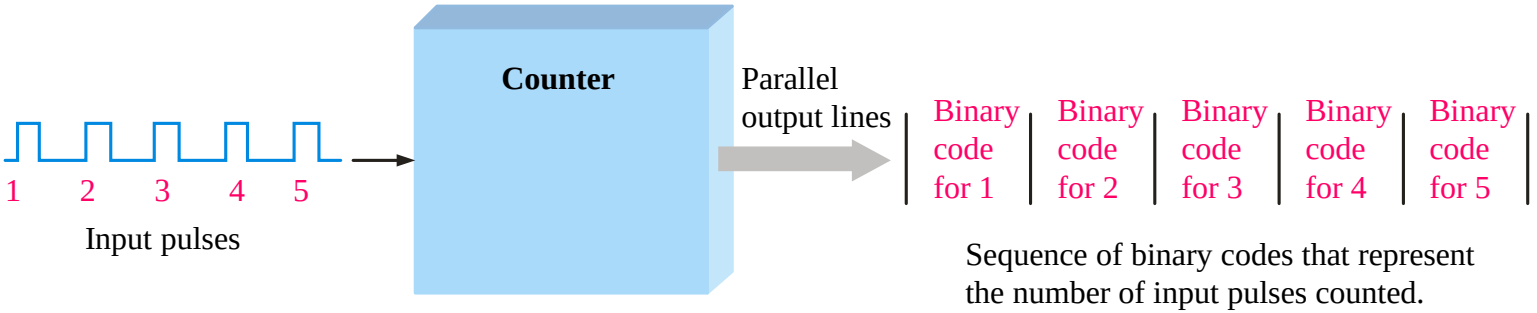
Basic System Functions

The data selection function



Basic System Functions

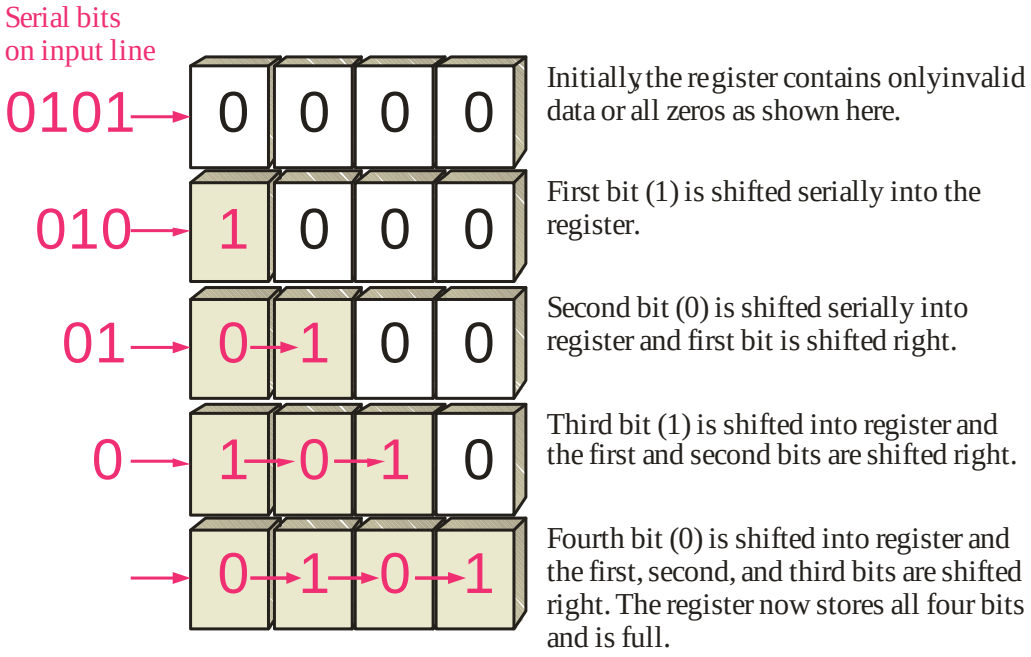
The counting function



…and other functions such as code conversion and storage.

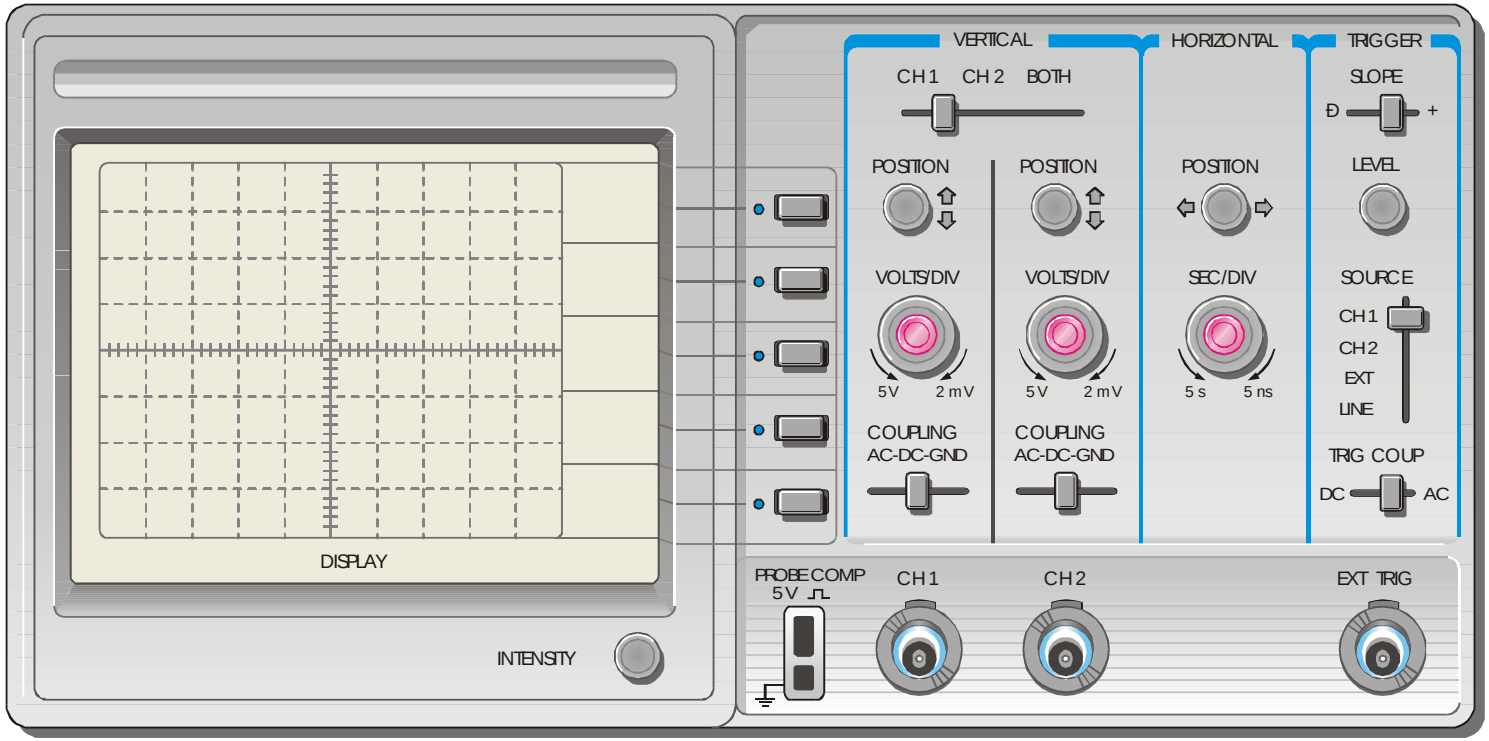
Basic System Functions

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



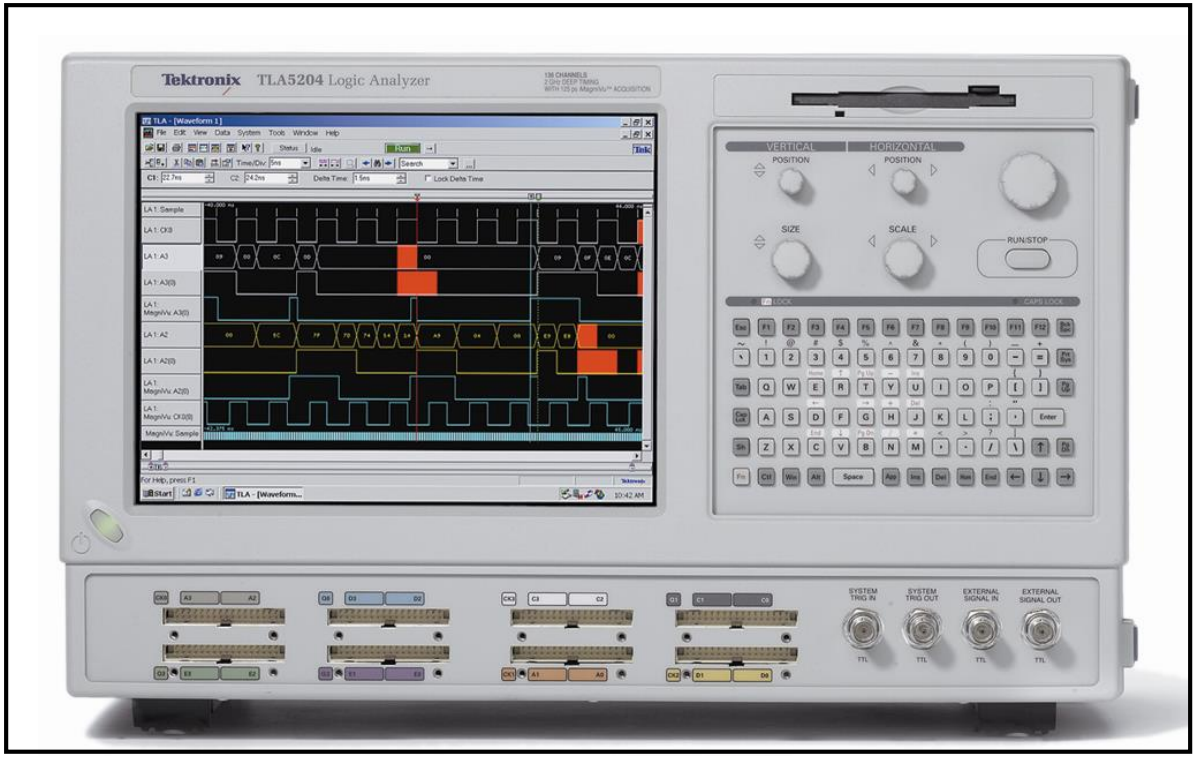
Test and Measurement Instruments

The front panel controls for a general-purpose oscilloscope can be divided into four major groups.



Test and Measurement Instruments

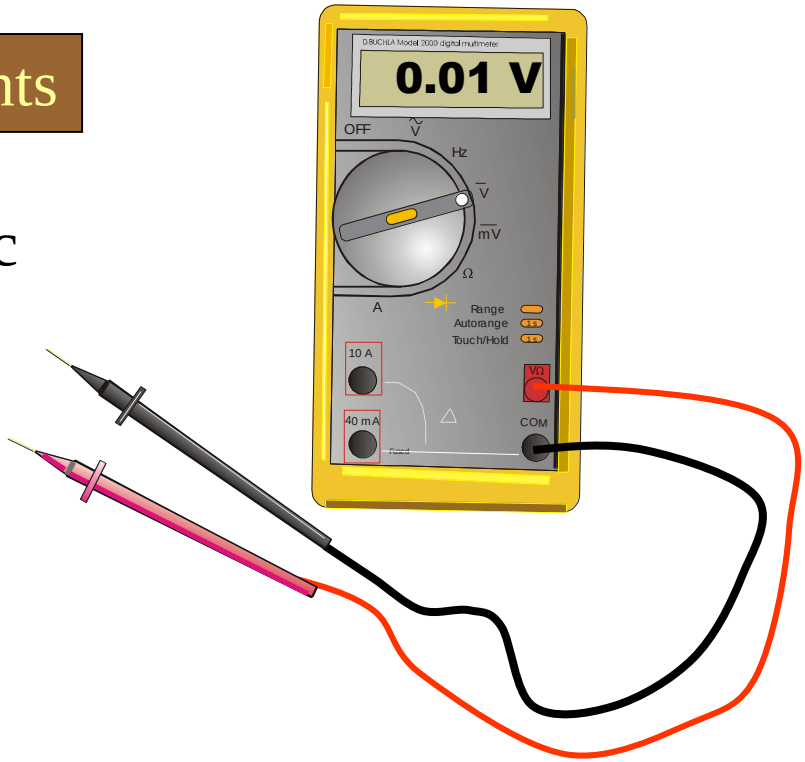
The logic analyzer can display multiple channels of digital information or show data in tabular form.



Test and Measurement Instruments

The DMM can make three basic electrical measurements.

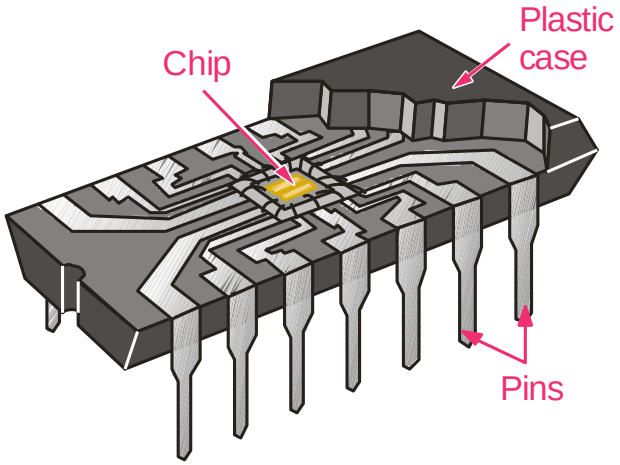
- Voltage
- Resistance
- Current



In digital work, DMMs are useful for checking power supply voltages, verifying resistors, testing continuity, and occasionally making other measurements.

Integrated Circuits

Cutaway view of DIP (Dual-In-line Pins) chip:

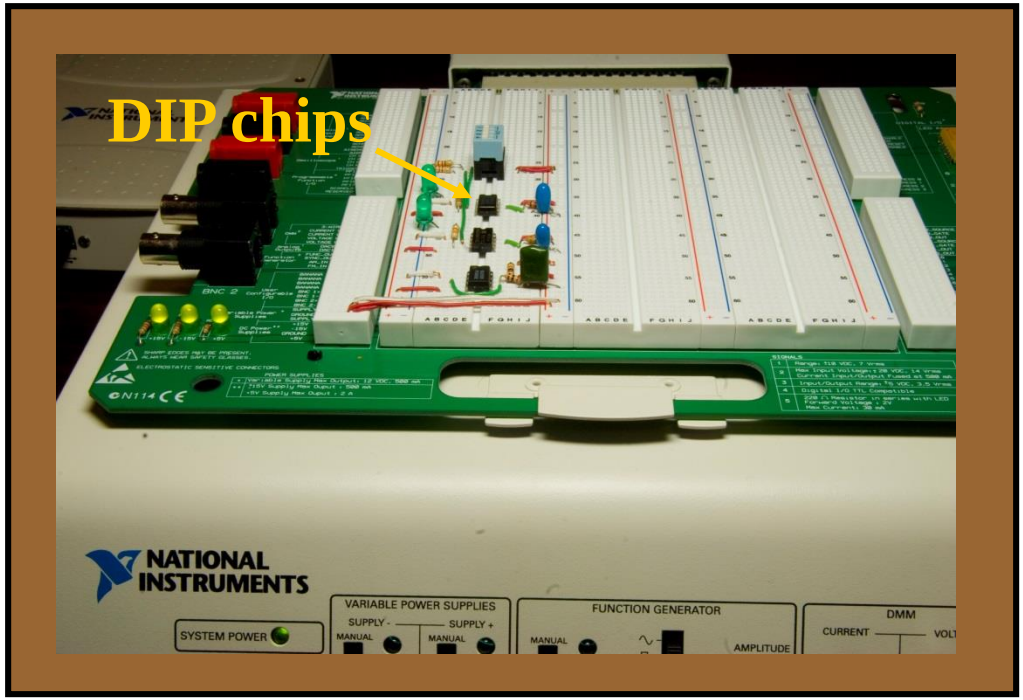


The TTL series, available as DIPs are popular for laboratory experiments with logic.

Integrated Circuits

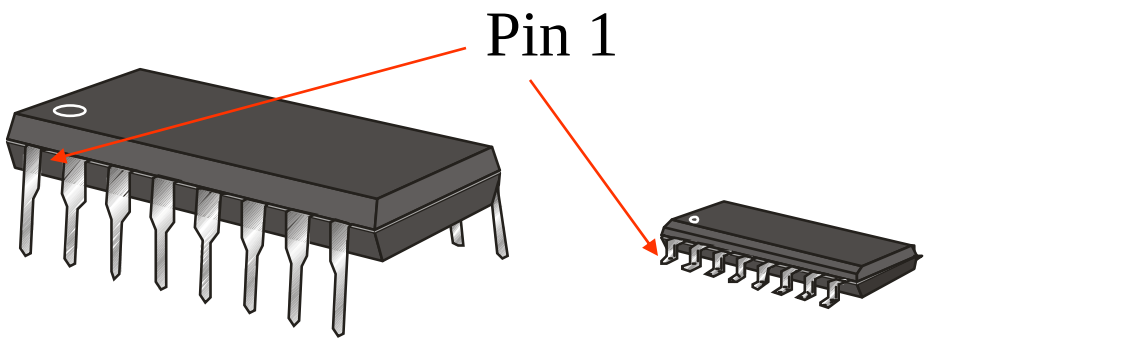
An example of laboratory prototyping is shown. The circuit is wired using DIP chips and tested.

In this case, testing can be done by a computer connected to the system.



Integrated Circuits

DIP chips and surface mount chips

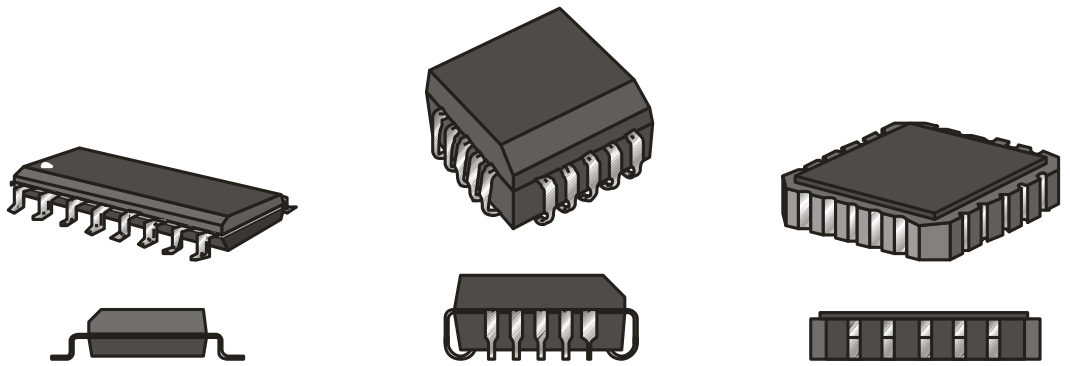


Dual in-line package

Small outline IC (SOIC)

Integrated Circuits

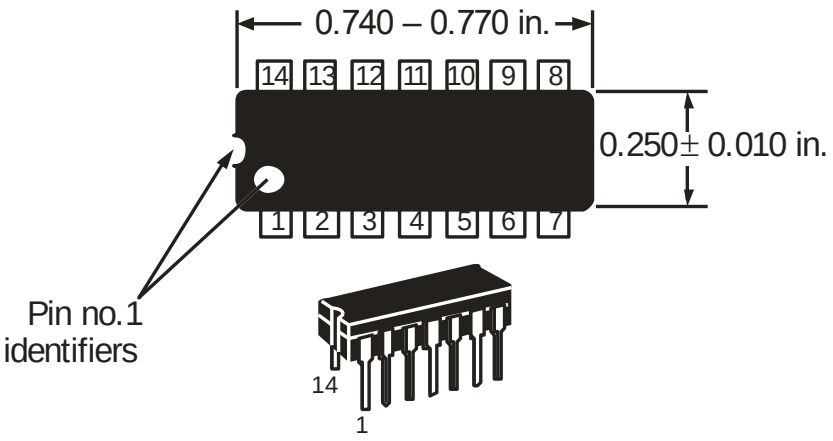
Other surface mount packages:



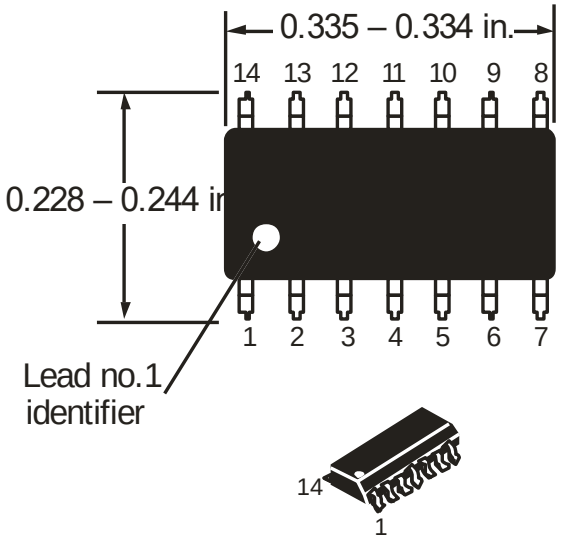
SOIC PLCC LCCC

Fixed Function Logic

Two major fixed function logic families are TTL and CMOS. A third technology is BiCMOS, which combines the first two. Packaging for fixed function logic is shown.



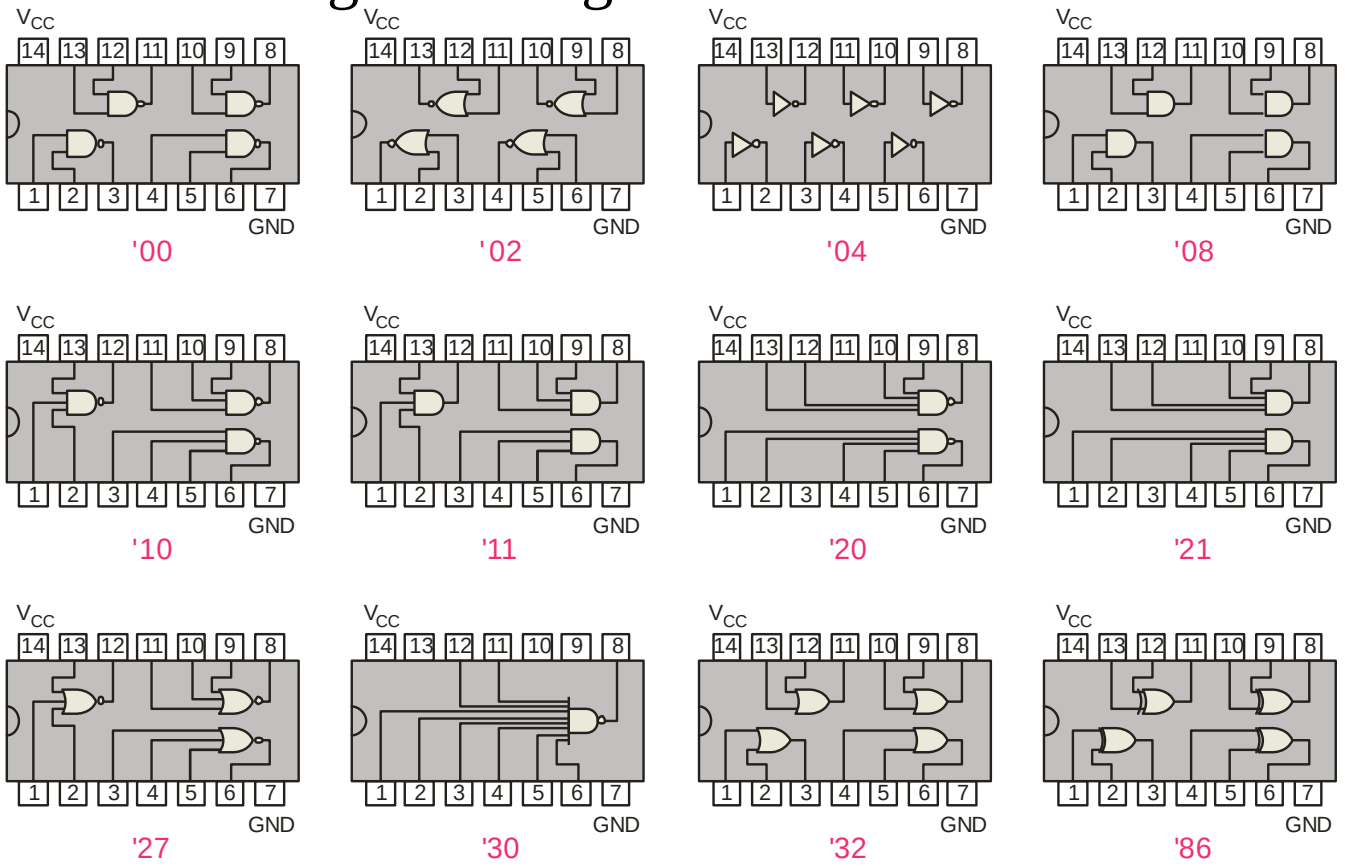
DIP package



SOIC package

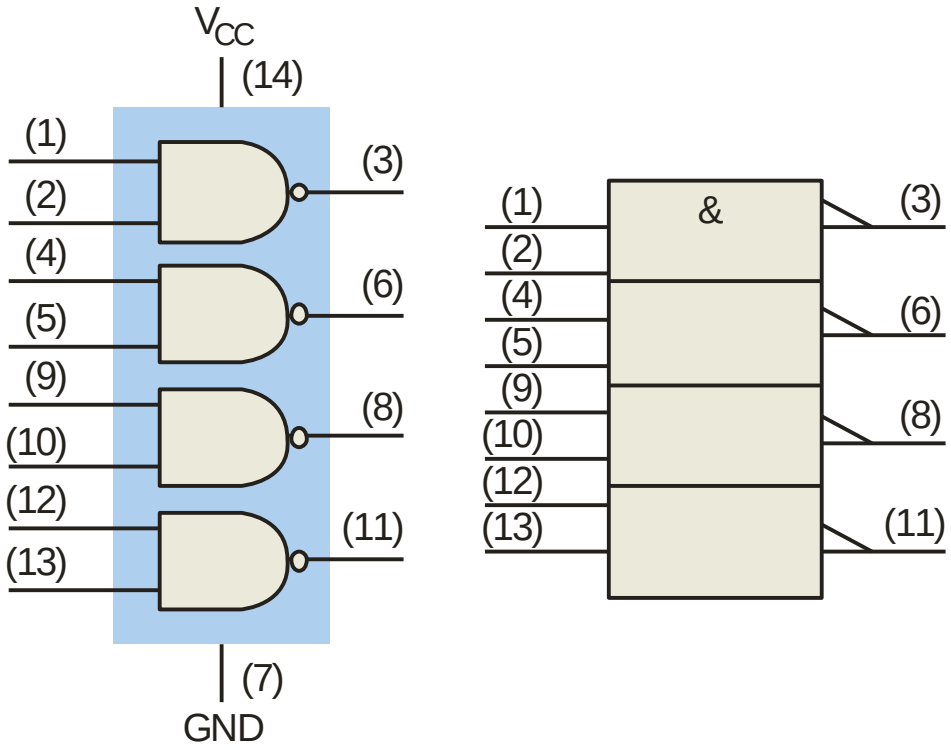
Fixed Function Logic

Some common gate configurations are shown.



Fixed Function Logic

Logic symbols show the gates and associated pin numbers.

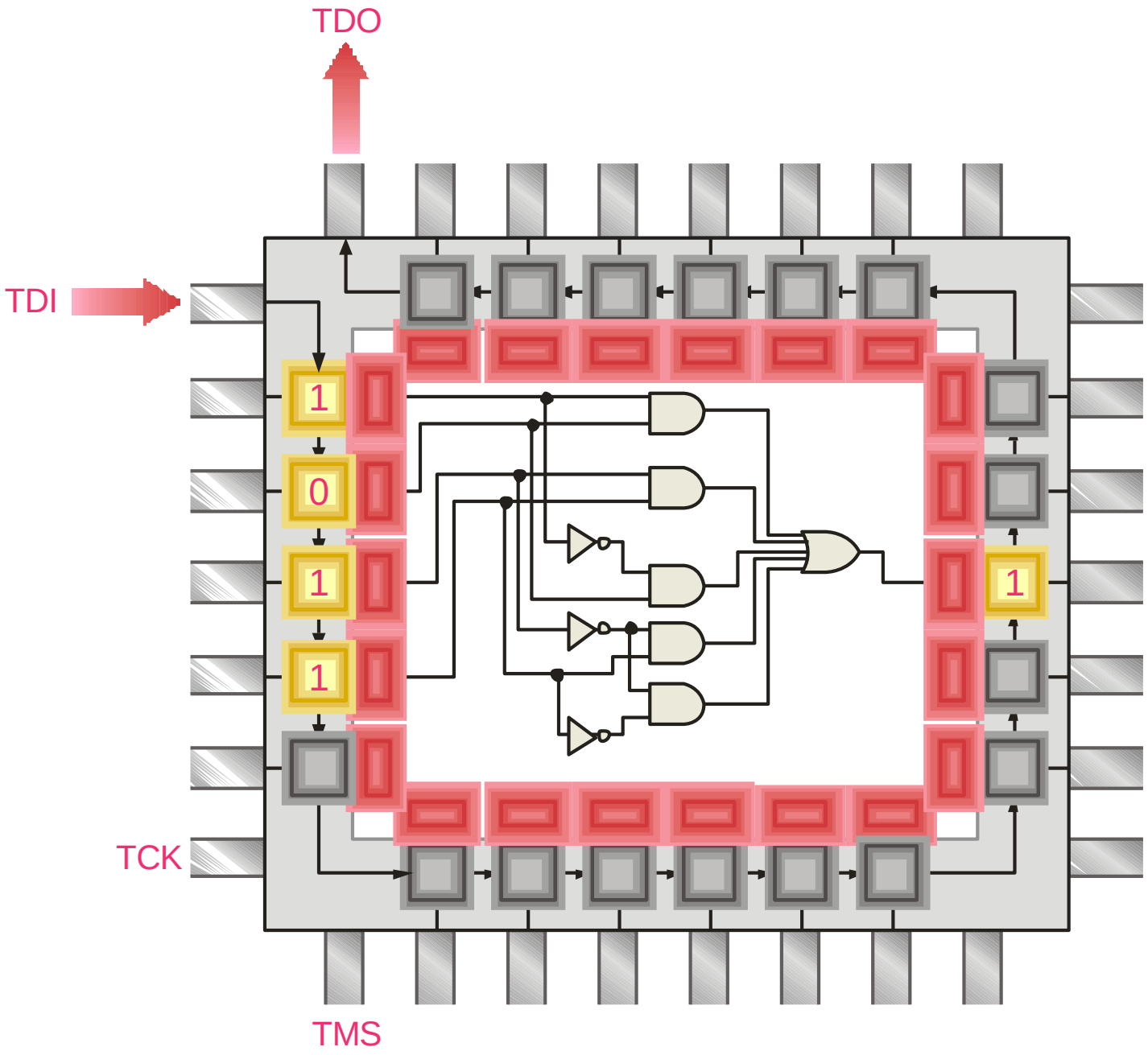


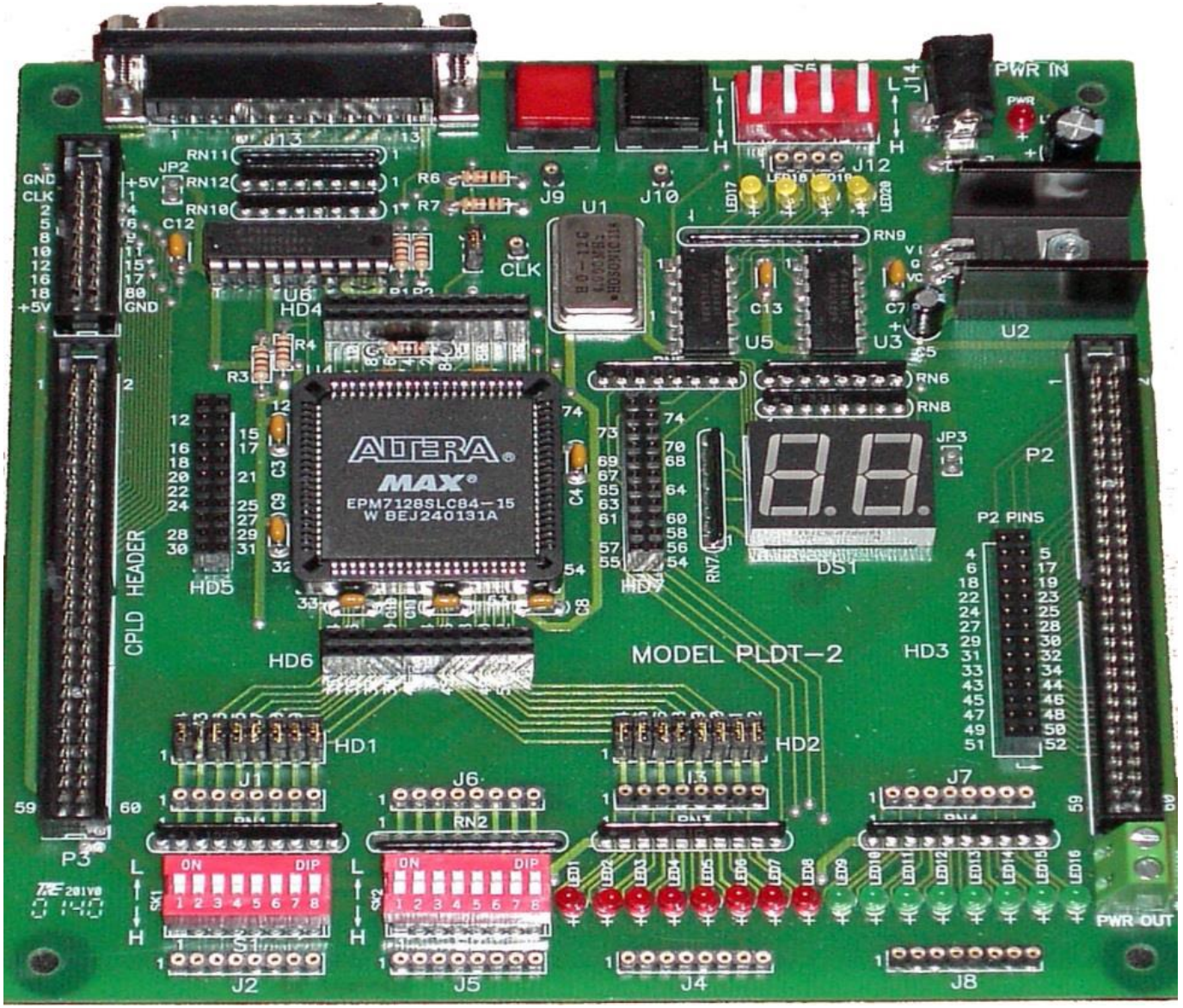
Fixed Function Logic

Data sheets include limits and conditions set by the manufacturer as well as DC and AC characteristics. For example, some maximum ratings for a 74HC00A are:

MAXIMUM RATINGS

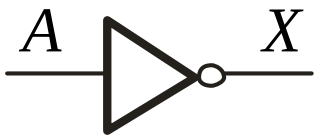
Symbol	Parameter	Value	Unit
V_{CC}	DC Supply Voltage (Referenced to GND)	–0.5 to + 7.0 V	V
V_{in}	DC InputVoltage (Referenced to GND)	–0.5 to V_{CC} +0.5 V	V
V_{out}	DC Output Voltage (Referenced to GND)	– 0.5 to V_{CC} +0.5 V	V
I_{in}	DC Input Current, per pin	±20	mA
I_{out}	DC Output Current, per pin	±25	mA
I_{CC}	DC Supply Current, V_{CC} and GND pins	±50	mA
P_D	Power Dissipation in Still Air, Plastic or Ceramic DIP † SOIC Package † TSSOP Package †	750 500 450	mW
T_{stg}	Storage Temperature	–65 to + 150	°C
T_L	Lead Temperature, 1 mm from Case for 10 Seconds Plastic DIP, SOIC, or TSSOP Package Ceramic DIP	 260 300	°C


TDO
TDI
TCK
TMS
1
0
1
1
1


PWR IN
PWR
J13
J14
J12
RN11
RN12
RN10
JP2
+5V
GND
CLK
R6
R7
J9
J10
U1
C12
CLK
RN9
C13
C7
U5
U3
U2
U6
HD4
R4
R3
RN6
RN8
JP3
P2
ALTERA
MAX
EPM7128SLC84-15
W BEJ240131A
DS1
P2 PINS
HD5
CPLD HEADER
HD7
RN7
HD6
MODEL PLDT-2
HD3
HD1
HD2
J1
J6
RN2
J7
P3
ON
DIP
S1
S2
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
LED9
LED10
LED11
LED12
LED13
LED14
LED15
LED16
PWR OUT
J2
J5
J4
J8

Logic Functions and Operations

The Inverter

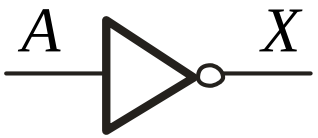


The inverter performs the Boolean **NOT** operation. When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW.

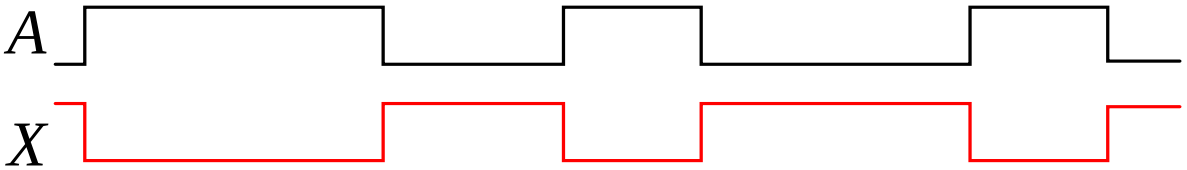
Input	Output
A	*X*
LOW (0)	HIGH (1)
HIGH (1)	LOW(0)

The **NOT** operation (complement) is shown with an overbar. Thus, the Boolean expression for an inverter is $X = \overline{A}$.

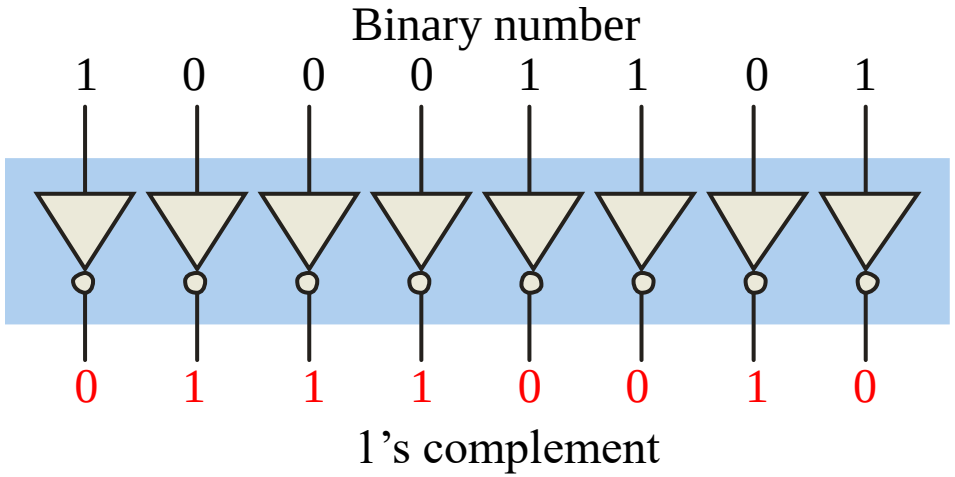
The Inverter



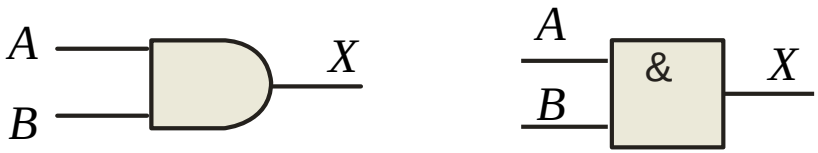
Example waveforms:



A group of inverters can be used to form the 1's complement of a binary number:



The AND Gate

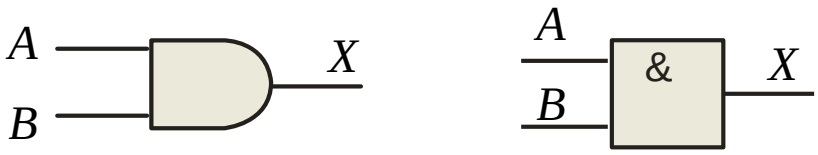


The **AND gate** produces a HIGH output when all inputs are HIGH; otherwise, the output is LOW. For a 2-input gate, the truth table is

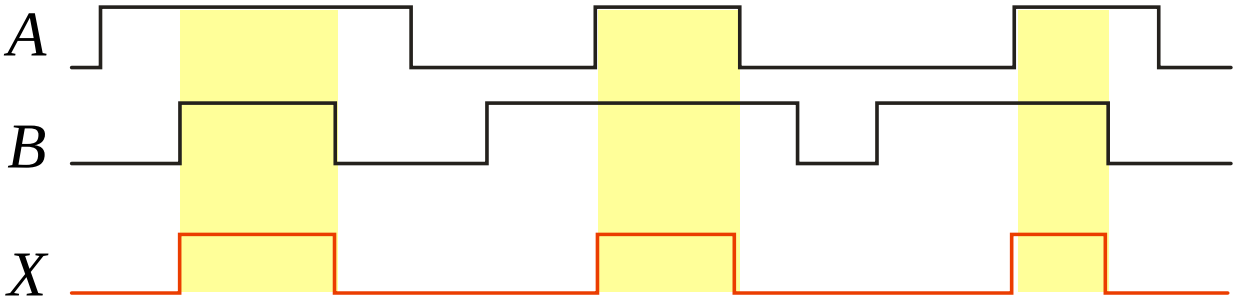
Inputs		Output
A	*B*	*X*
0	0	0
0	1	0
1	0	0
1	1	1

The **AND** operation is usually shown with a dot between the variables but it may be implied (no dot). Thus, the AND operation is written as $X = A \cdot B$ or $X = AB$.

The AND Gate



Example waveforms:



The AND operation is used in computer programming as a selective mask. If you want to retain certain bits of a binary number but reset the other bits to 0, you could set a mask with 1's in the position of the retained bits.

Example If the binary number 10100011 is ANDed with the mask 00001111, what is the result? 00000011

The OR Gate

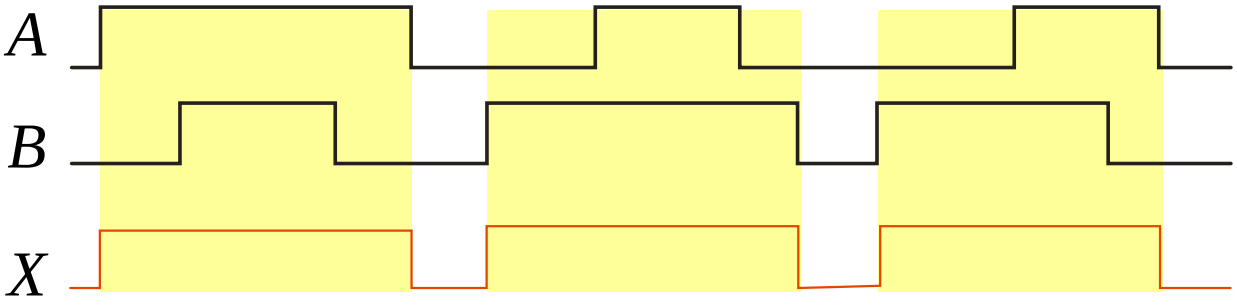
The **OR gate** produces a HIGH output if any input is HIGH; if all inputs are LOW, the output is LOW. For a 2-input gate, the truth table is

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

The **OR** operation is shown with a plus sign (+) between the variables. Thus, the OR operation is written as $X = A + B$.

The OR Gate

Example waveforms:



The OR operation can be used in computer programming to set certain bits of a binary number to 1.

Example ASCII letters have a 1 in the bit 5 position for lower case letters and a 0 in this position for capitals. (Bit positions are numbered from right to left starting with 0.) What will be the result if you OR an ASCII letter with the 8-bit mask 00100000?

Solution The resulting letter will be lower case.

The NAND Gate

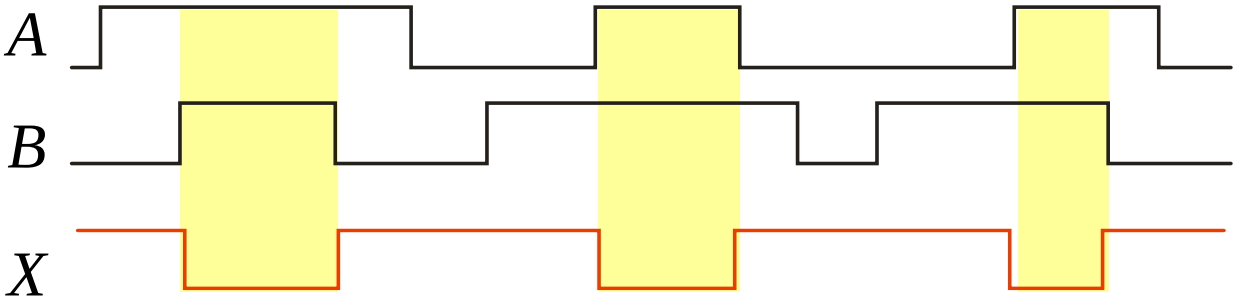
The **NAND gate** produces a LOW output when all inputs are HIGH; otherwise, the output is HIGH. For a 2-input gate, the truth table is

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

The **NAND** operation is shown with a dot between the variables and an overbar covering them. Thus, the NAND operation is written as $X = \overline{A \cdot B}$ (Alternatively, $X = \overline{AB}$.)

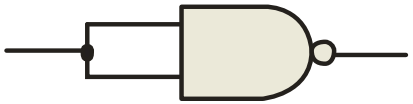
The NAND Gate

Example waveforms:



The NAND gate is particularly useful because it is a "universal" gate – all other basic gates can be constructed from NAND gates.

Question How would you connect a 2-input NAND gate to form a basic inverter?



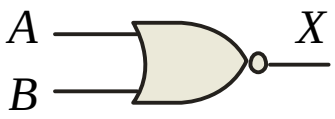
The NOR Gate

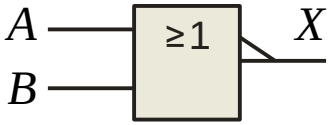
The **NOR gate** produces a LOW output if any input is HIGH; if all inputs are HIGH, the output is LOW. For a 2-input gate, the truth table is

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

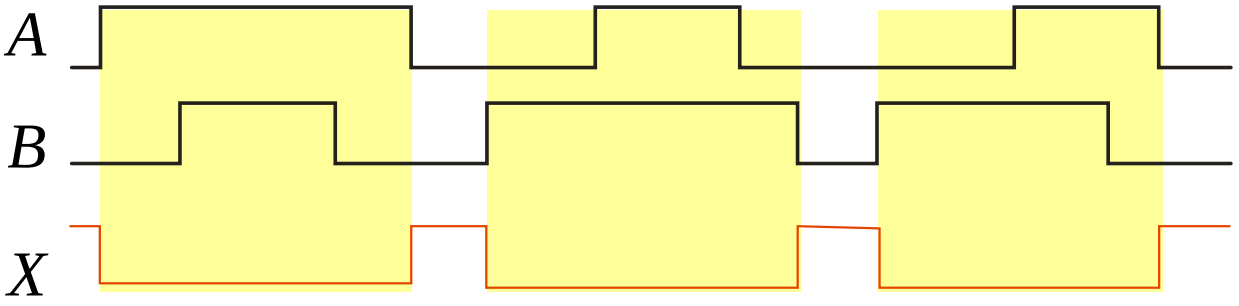
The **NOR** operation is shown with a plus sign (+) between the variables and an overbar covering them. Thus, the NOR operation is written as $X = \overline{A + B}$.

The NOR Gate



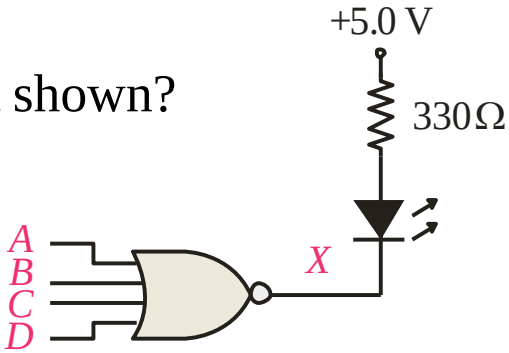


Example waveforms:



The NOR operation will produce a LOW if any input is HIGH.

Example When is the LED is ON for the circuit shown?



Solution The LED will be on when any of the four inputs are HIGH.

The XOR Gate

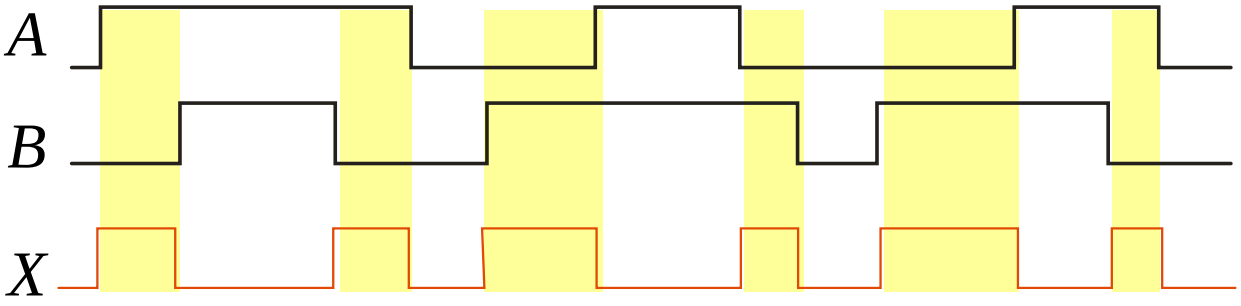
The **XOR gate** produces a HIGH output only when both inputs are at opposite logic levels. The truth table is

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

The **XOR** operation is written as $X = \overline{A}B + A\overline{B}$. Alternatively, it can be written with a circled plus sign between the variables as $X = A \oplus B$.

The XOR Gate

Example waveforms:

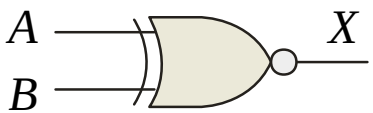


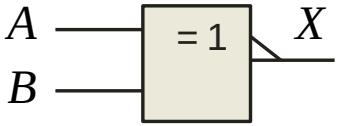
Notice that the XOR gate will produce a HIGH only when exactly one input is HIGH.

Question If the *A* and *B* waveforms are both inverted for the above waveforms, how is the output affected?

There is no change in the output.

The XNOR Gate



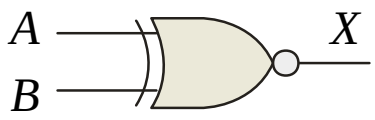


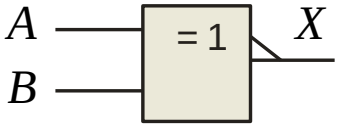
The **XNOR gate** produces a HIGH output only when both inputs are at the same logic level. The truth table is

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

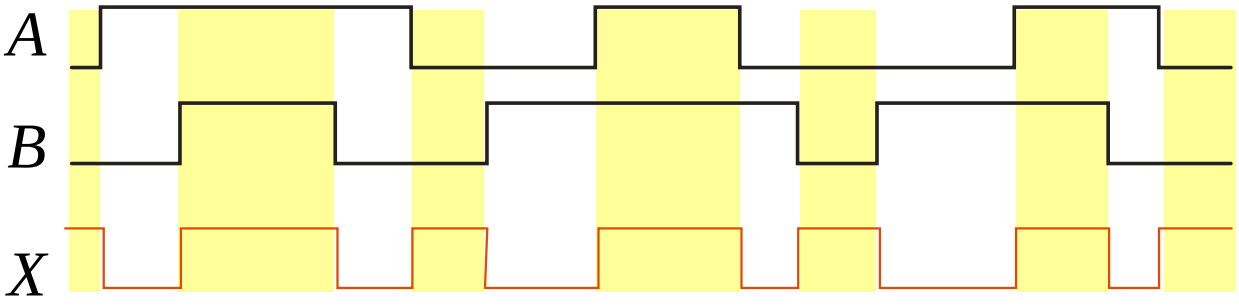
The **XNOR** operation shown as $X = \overline{A}\,\overline{B} + AB$. Alternatively, the XNOR operation can be shown with a circled dot between the variables. Thus, it can be shown as $X = A \odot B$.

The XNOR Gate





Example waveforms:



Notice that the XNOR gate will produce a HIGH when both inputs are the same. This makes it useful for comparison functions.

Question If the *A* waveform is inverted but *B* remains the same, how is the output affected?

The output will be inverted.

Inverter	A logic circuit that inverts or complements its inputs.
Truth table	A table showing the inputs and corresponding output(s) of a logic circuit.
Timing diagram	A diagram of waveforms showing the proper time relationship of all of the waveforms.
Boolean algebra	The mathematics of logic circuits.
AND gate	A logic gate that produces a HIGH output only when all of its inputs are HIGH.

OR gate	A logic gate that produces a HIGH output when one or more inputs are HIGH.
NAND gate	A logic gate that produces a LOW output only when all of its inputs are HIGH.
NOR gate	A logic gate that produces a LOW output when one or more inputs are HIGH.
Exclusive-OR gate	A logic gate that produces a HIGH output only when its two inputs are at opposite levels.
Exclusive-NOR gate	A logic gate that produces a LOW output only when its two inputs are at opposite levels.

Quiz

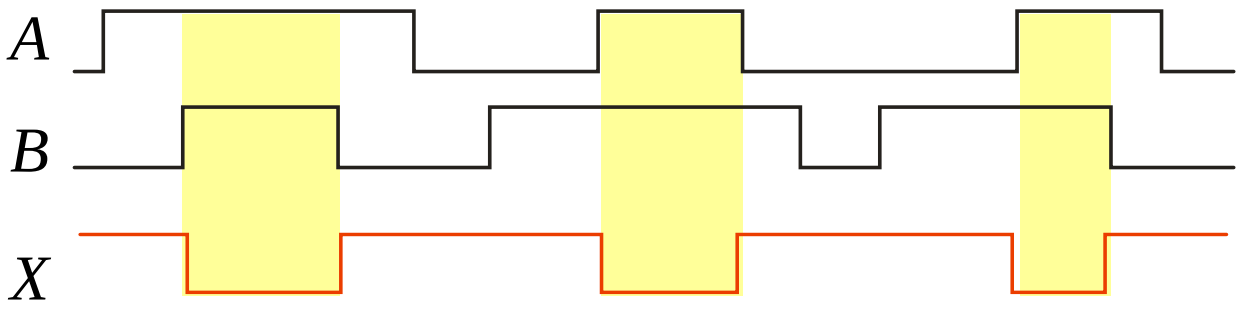
A 2-input gate produces the output shown. (*X* represents the output.) This is a(n)

a. OR gate

b. AND gate

c. NOR gate

d. NAND gate



Logic Algebra & Circuit Minimization

Boolean Addition

In Boolean algebra, a **variable** is a symbol used to represent an action, a condition, or data. A single variable can only have a value of 1 or 0.

The **complement** represents the inverse of a variable and is indicated with an overbar. Thus, the complement of A is $\overline{A}$.

A **literal** is a variable or its complement.

Addition is equivalent to the OR operation. The sum term is 1 if one or more if the literals are 1. The sum term is zero only if each literal is 0.

Example Determine the values of A, B, and C that make the sum term of the expression $\overline{A} + B + \overline{C} = 0$?

Solution Each literal must = 0; therefore $A = 1$, $B = 0$ and $C = 1$.

Boolean Multiplication

In Boolean algebra, multiplication is equivalent to the AND operation. The product of literals forms a product term. The product term will be 1 only if all of the literals are 1.

Example What are the values of the A, B and C if the product term of $A \cdot B \cdot C = 1$?

Solution Each literal must = 1; therefore $A = 1$, $B = 0$ and $C = 0$.

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.

$$AB = BA$$

Associative Laws

The **associative laws** are also applied to addition and multiplication. For addition, the associative law states

When ORing more than two variables, the result is the same regardless of the grouping of the variables.

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

For multiplication, the associative law states

When ANDing more than two variables, the result is the same regardless of the grouping of the variables.

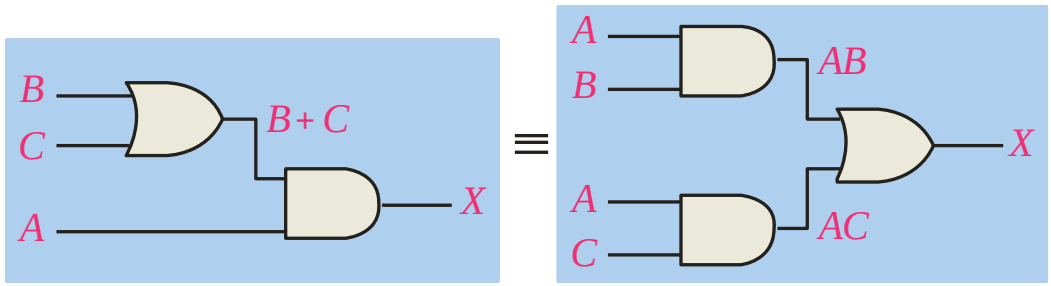
$$A(BC) = (AB)C$$

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

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

The distributive law can be illustrated with equivalent circuits:



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

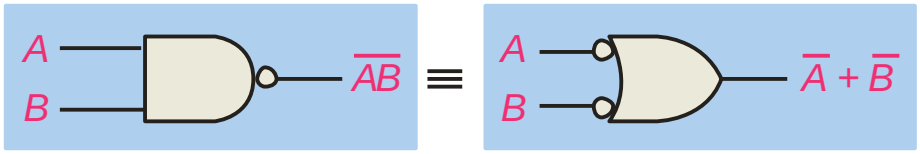
DeMorgan's Theorem

DeMorgan's 1st Theorem

The complement of a product of variables is equal to the sum of the complemented variables.

$$\overline{AB} = \overline{A} + \overline{B}$$

Applying DeMorgan's first theorem to gates:



NAND Negative-OR

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

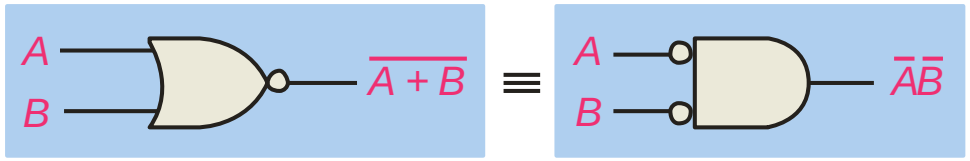
DeMorgan's Theorem

DeMorgan's 2nd Theorem

The complement of a sum of variables is equal to the product of the complemented variables.

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

Applying DeMorgan's second theorem to gates:



NOR Negative-AND

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

DeMorgan's Theorem

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 $X = C \cdot \overline{D}$.

Boolean Analysis of Logic Circuits

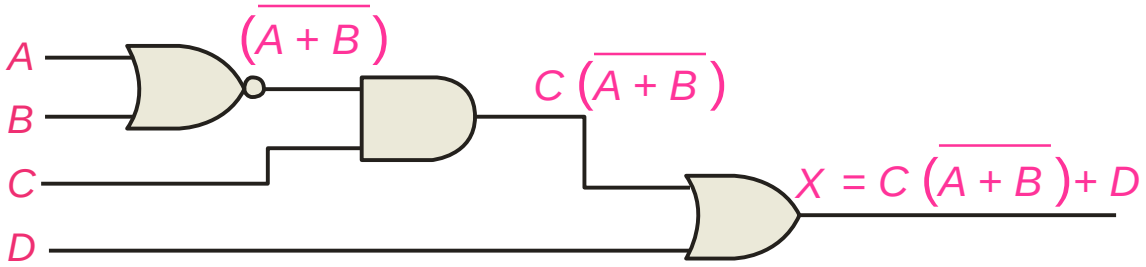
Combinational logic circuits can be analyzed by writing the expression for each gate and combining the expressions according to the rules for Boolean algebra.

Example

Apply Boolean algebra to derive the expression for *X*.

Solution

Write the expression for each gate:



Applying DeMorgan's theorem and the distribution law:

$$X = C(\overline{A}\ \overline{B}) + D = \overline{A}\,\overline{B}\,C + D$$

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:

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

SOP Standard form

In **SOP standard form**, every variable in the domain must appear in each term. This form is useful for constructing truth tables or for implementing logic in PLDs.

You can expand a nonstandard term to standard form by multiplying the term by a term consisting of the sum of the missing variable and its complement.

Example

Convert $X = \overline{A}\,\overline{B} + A\,B\,C$ to standard form.

Solution

The first term does not include the variable C. Therefore, multiply it by the $(C + \overline{C})$, which = 1:

$$X = \overline{A}\,\overline{B}\,(C + \overline{C}) + A\,B\,C$$

$$= \overline{A}\,\overline{B}\,C + \overline{A}\,\overline{B}\,\overline{C} + A\,B\,C$$

POS Standard form

In **POS standard form**, every variable in the domain must appear in each sum term of the expression.

You can expand a nonstandard POS expression to standard form by adding the product of the missing variable and its complement and applying rule 12, which states that $(A + B)(A + C) = A + BC$.

Example Convert $X = (\overline{A} + \overline{B})(A + B + C)$ to standard form.

Solution The first sum term does not include the variable C. Therefore, add $C\,\overline{C}$ and expand the result by rule 12.

$$X = (\overline{A} + \overline{B} + C\,\overline{C})(A + B + C)$$

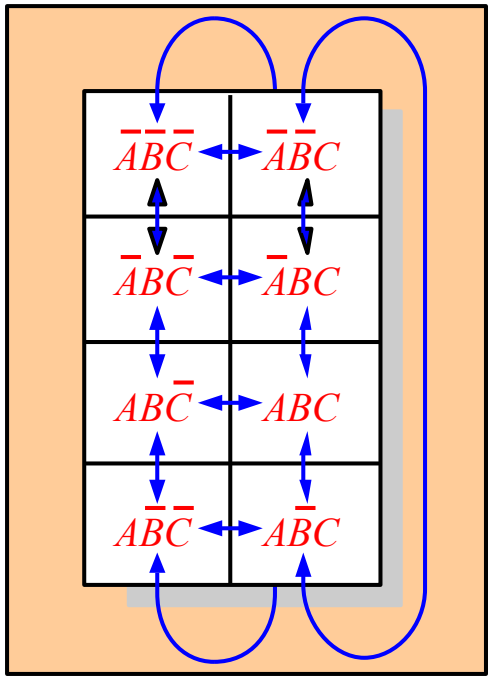
$$= (\overline{A} + \overline{B} + C)(\overline{A} + \overline{B} + \overline{C})(A + B + C)$$

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).

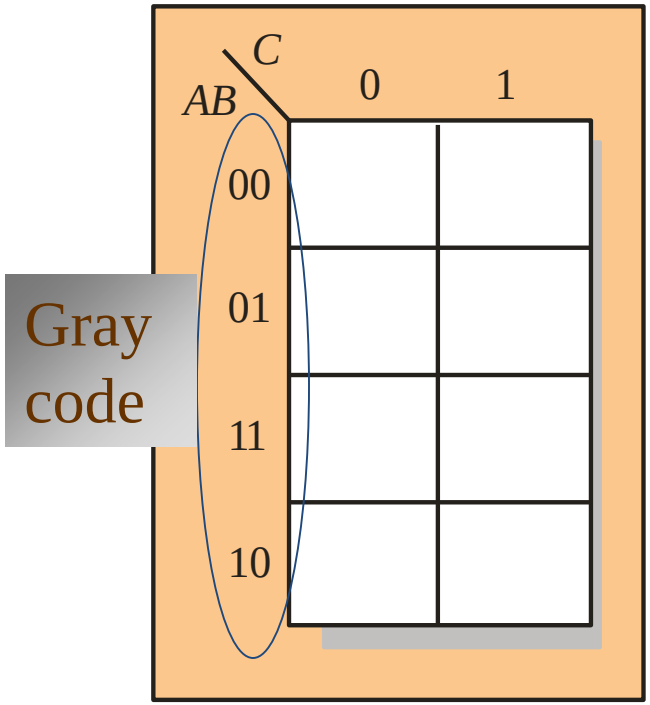
The map shown is for three variables labeled *A*, *B*, and *C*. Each cell represents one possible product term.

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



Karnaugh maps

Cells are usually labeled using 0's and 1's to represent the variable and its complement.



The numbers are entered in gray code, to force adjacent cells to be different by only one variable.

Ones are read as the true variable and zeros are read as the complemented variable.

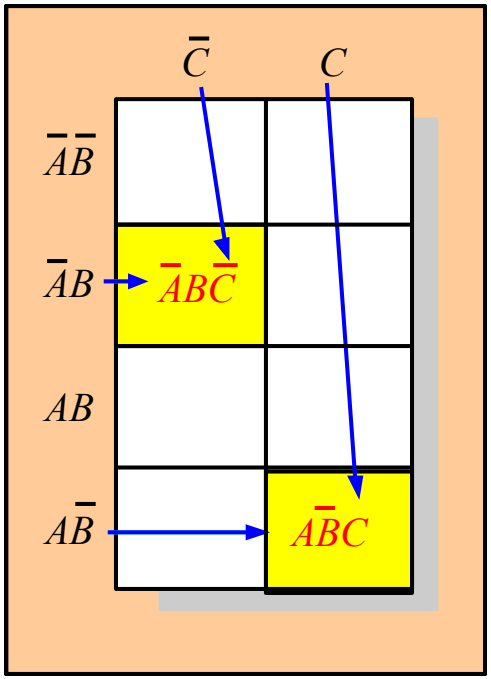
Karnaugh maps

Alternatively, cells can be labeled with the variable letters. This makes it simple to read, but it takes more time preparing the map.

Example Read the terms for the yellow cells.

Solution

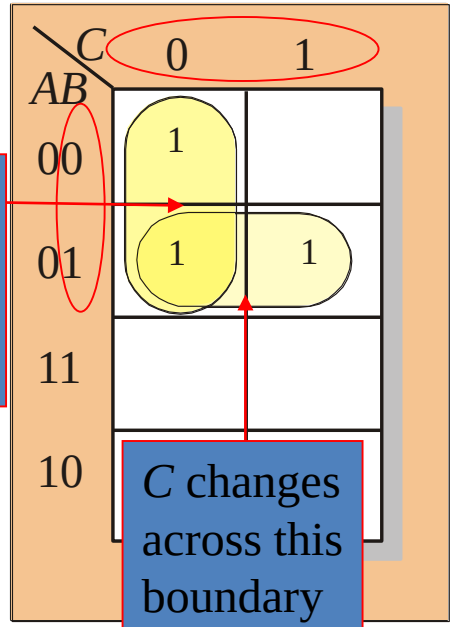
The cells are $\bar{A}B\bar{C}$ and $A\bar{B}C$.



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.



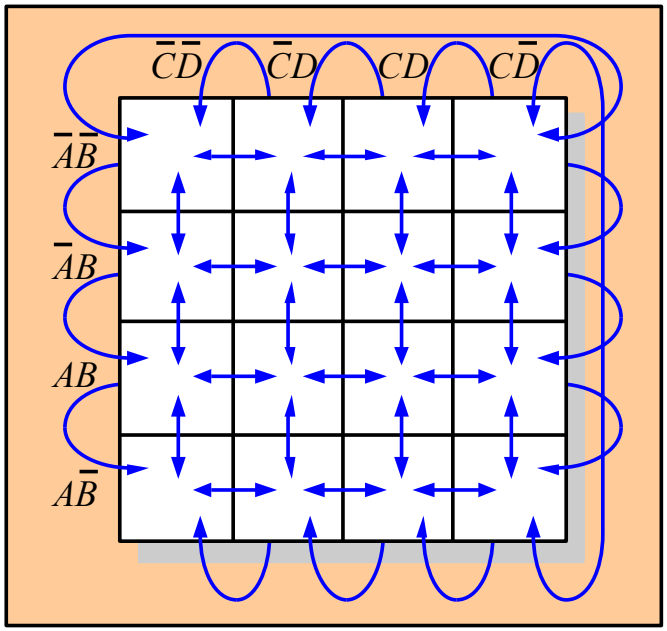
Solution

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 $\overline{A}\,\overline{C}$.
4. The horizontal group is read $\overline{A}B$.

$$X = \overline{A}\,\overline{C} + \overline{A}B$$

Karnaugh maps

A 4-variable map has an adjacent cell on each of its four boundaries as shown.



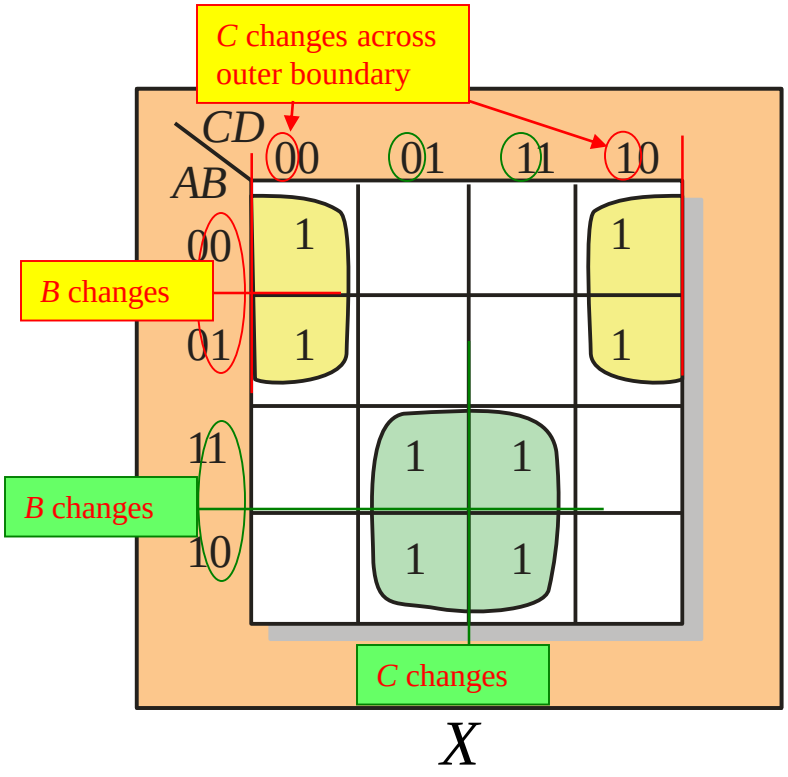
Each cell is different only by one variable from an adjacent cell.

Grouping follows the rules given in the text.

The following slide shows an example of reading a four variable map using binary numbers for the variables…

Karnaugh maps

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



Solution

1. Group the 1's into two separate groups as indicated.
2. Read each group by eliminating any variable that changes across a boundary.
3. The upper (yellow) group is read as $\bar{A}\bar{D}$.
4. The lower (green) group is read as AD.

$$X = \bar{A}\bar{D} + AD$$

Variable	A symbol used to represent a logical quantity that can have a value of 1 or 0, usually designated by an italic letter.
Complement	The inverse or opposite of a number. In Boolean algebra, the inverse function, expressed with a bar over the variable.
Sum term	The Boolean sum of two or more literals equivalent to an OR operation.
Product term	The Boolean product of two or more literals equivalent to an AND operation.

Sum-of-products (SOP)	A form of Boolean expression that is basically the ORing of ANDed terms.
Product of sums (POS)	A form of Boolean expression that is basically the ANDing of ORed terms.
Karnaugh map	An arrangement of cells representing combinations of literals in a Boolean expression and used for systematic simplification of the expression.
VHDL	A standard hardware description language. IEEE Std. 1076-1993.

Quiz

The Boolean expression $A \cdot 1$ is equal to

a. A

b. B

c. 0

d. 1

Quiz

The Boolean expression $A + 1$ is equal to

a. A

b. B

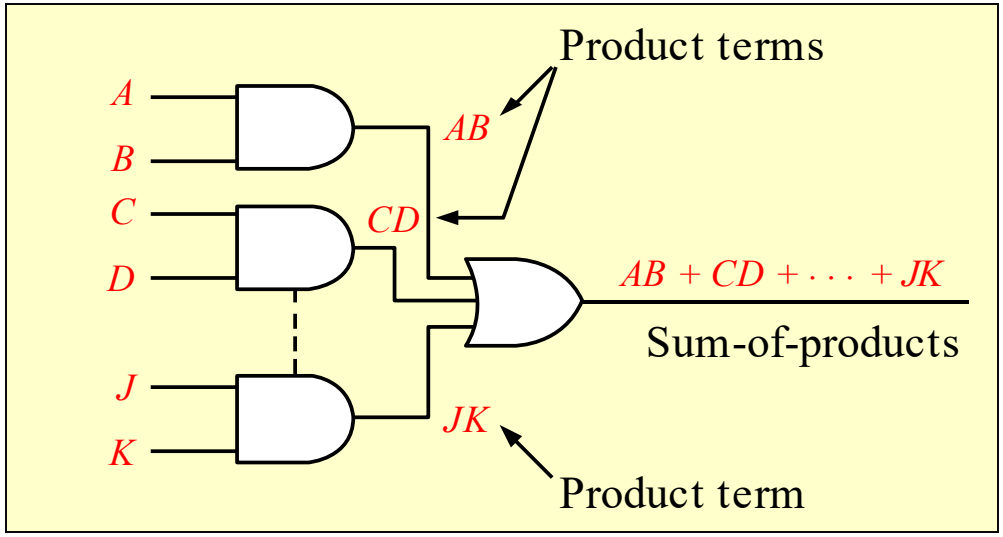
c. 0

d. 1

Combinational Logic Analysis

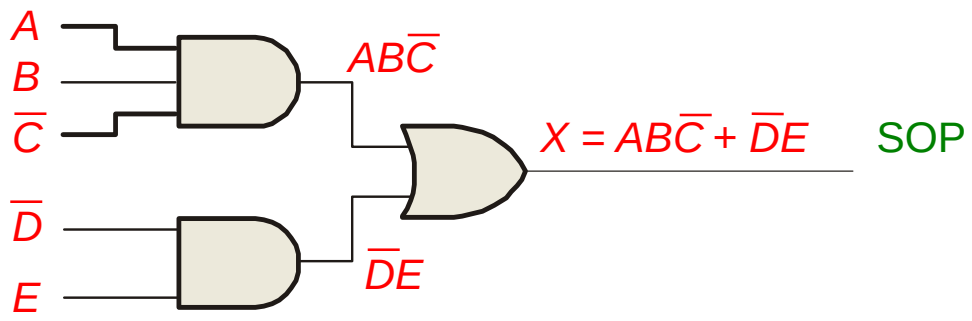
Combinational Logic Circuits

In Sum-of-Products (SOP) form, basic combinational circuits can be directly implemented with AND-OR combinations if the necessary complement terms are available.



Combinational Logic Circuits

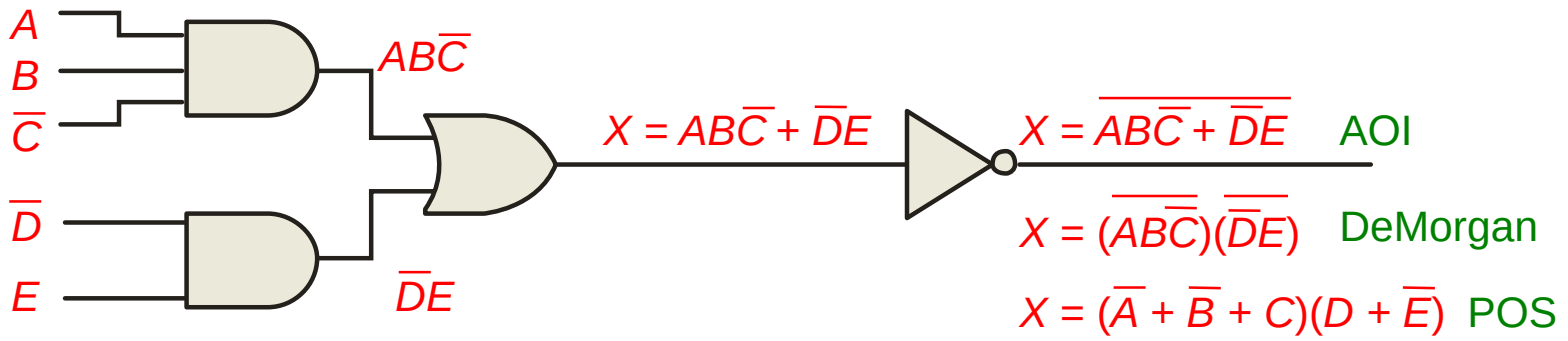
An example of an SOP implementation is shown. The SOP expression is an AND-OR combination of the input variables and the appropriate complements.



Combinational Logic Circuits

When the output of a SOP form is inverted, the circuit is called an AND-OR-Invert 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.



Exclusive-OR Logic

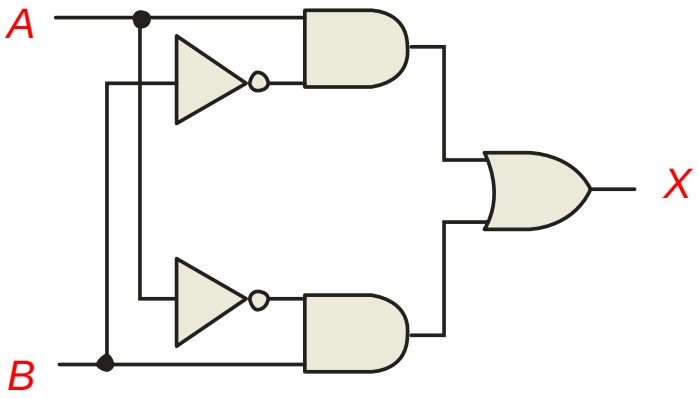
The truth table for an exclusive-OR gate is

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

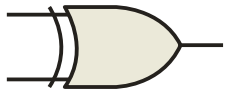
Notice that the output is HIGH whenever *A* and *B* disagree.

The Boolean expression is $X = \bar{A}B + A\bar{B}$

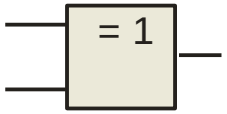
The circuit can be drawn as



Symbols:



Distinctive shape



Rectangular outline

Exclusive-NOR Logic

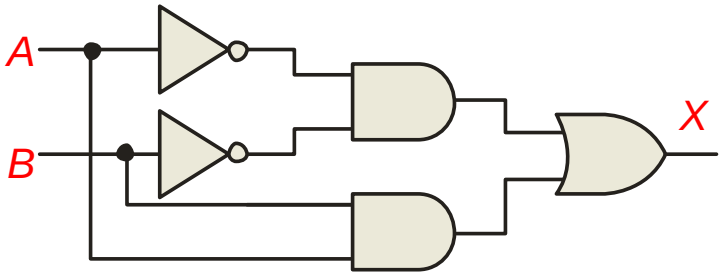
The truth table for an exclusive-NOR gate is

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

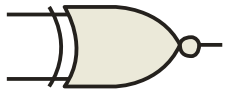
Notice that the output is HIGH whenever *A* and *B* agree.

The Boolean expression is $X = \overline{A}\,\overline{B} + AB$

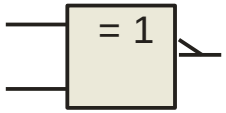
The circuit can be drawn as



Symbols:



Distinctive shape



Rectangular outline

Implementing Combinational Logic

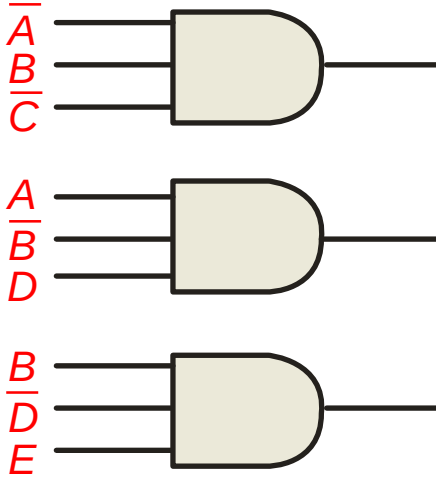
Implementing a SOP expression is done by first forming the AND terms; then the terms are ORed together.

Example

Show the circuit that will implement the Boolean expression $X = \overline{A}B\overline{C} + A\overline{B}D + B\overline{D}E$. (Assume that the variables and their complements are available.)

Solution

Start by forming the terms using three 3-input AND gates. Then combine the three terms using a 3-input OR gate.

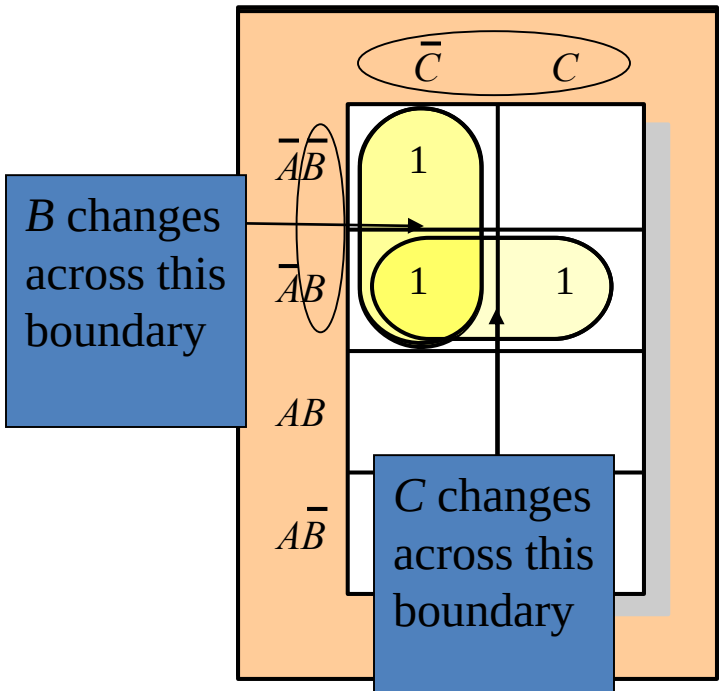


$$X = \overline{A}B\overline{C} + A\overline{B}D + B\overline{D}E$$

Karnaugh Map Implementation

For basic combinational logic circuits, the Karnaugh map can be read and the circuit drawn as a minimum SOP.

Example A Karnaugh map is drawn from a truth table. Read the minimum SOP expression and draw the circuit.



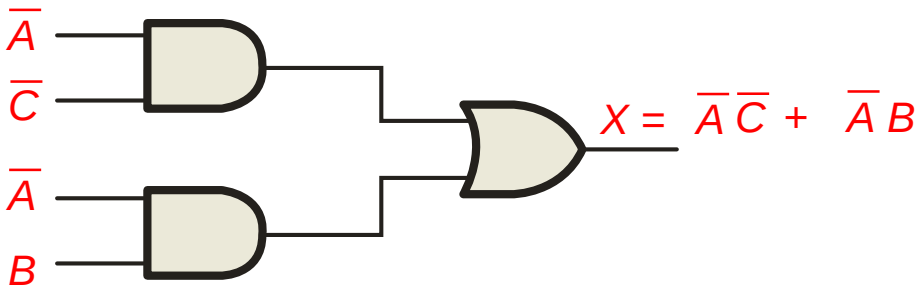
Solution

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 $\bar{A}B$.

The circuit is on the next slide:

Solution *continued…*

Circuit:



The result is shown as a sum of products.

It is a simple matter to implement this form using only NAND gates as shown in the text and following example.

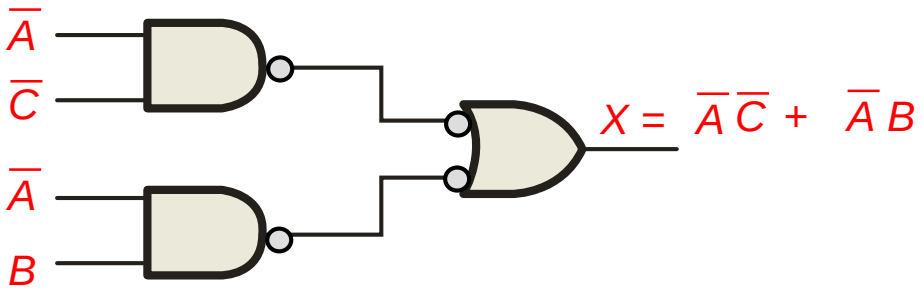
NAND Logic

Example

Convert the circuit in the previous example to one that uses only NAND gates.

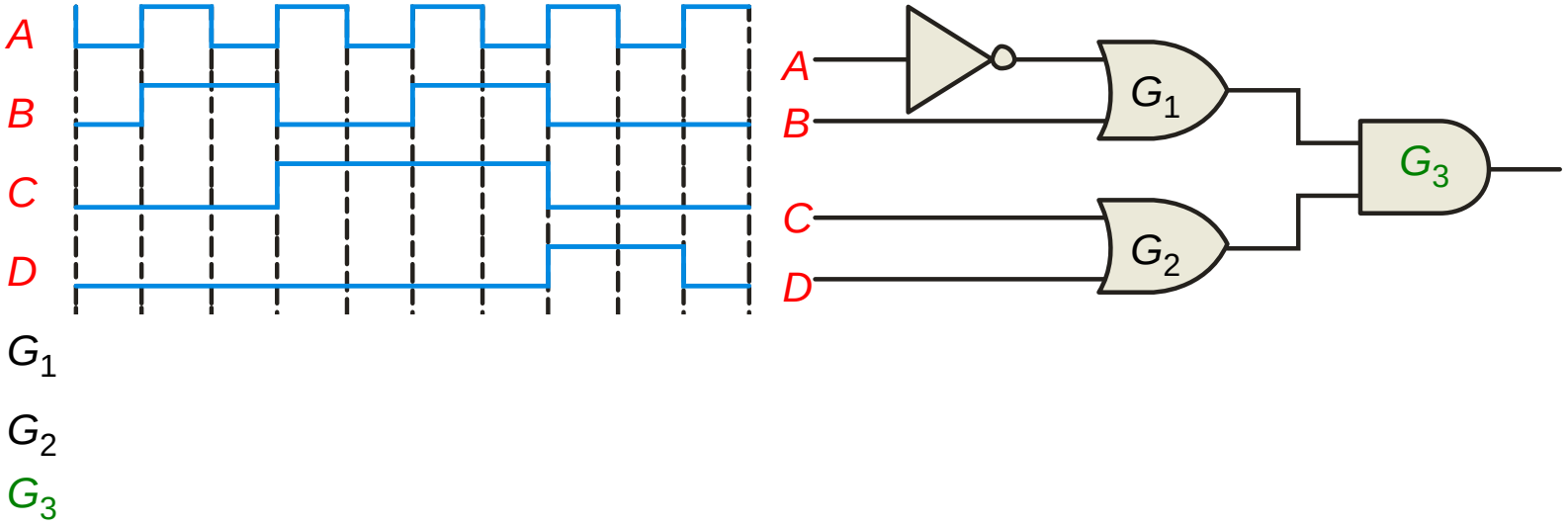
Solution

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



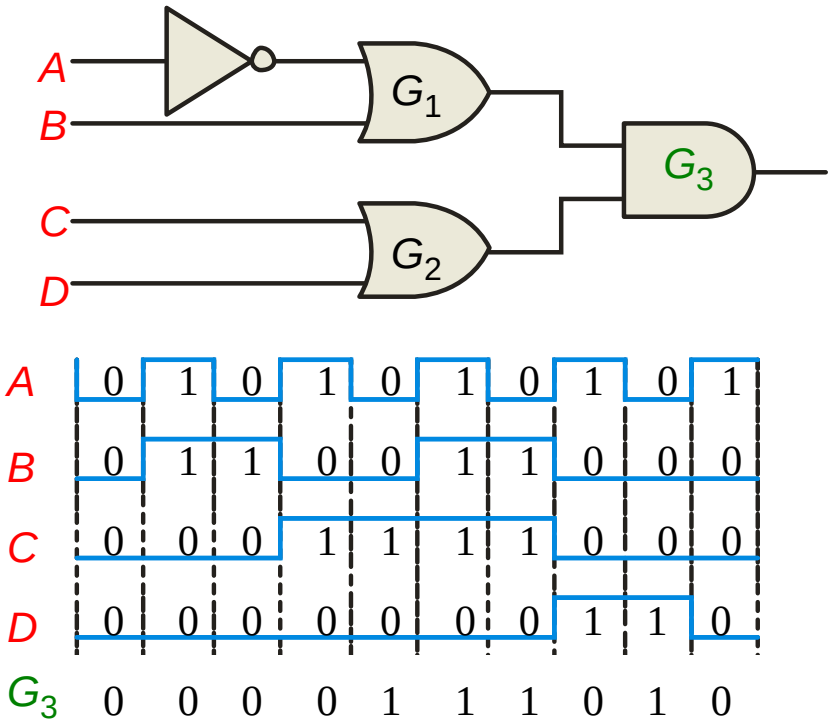
Pulsed Waveforms

For combinational circuits with pulsed inputs, the output can be predicted by developing intermediate outputs and combining the result. For example, the circuit shown can be analyzed at the outputs of the OR gates:



Pulsed Waveforms

Alternatively, you can develop the truth table for the circuit and enter 0's and 1's on the waveforms. Then read the output from the table.

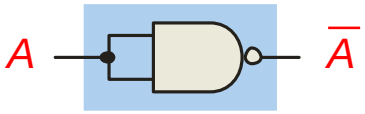


Inputs				Output
A	*B*	*C*	*D*	*X*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

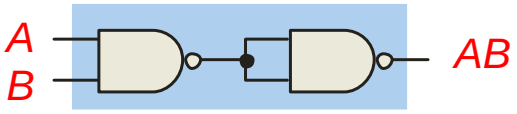
Logic Function Transformation

Universal Gates

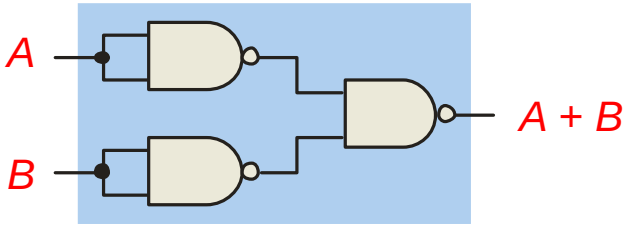
NAND gates are sometimes called **universal** gates because they can be used to produce the other basic Boolean functions.



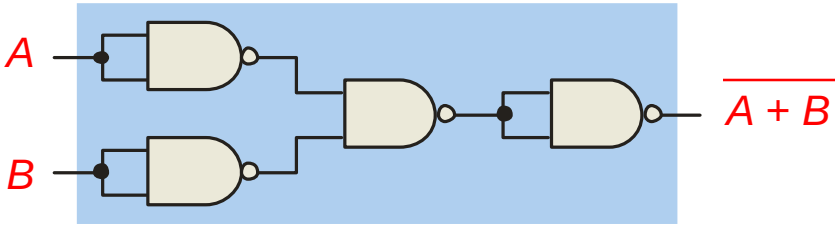
Inverter



AND gate



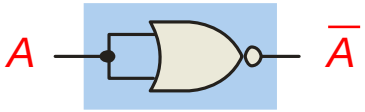
OR gate



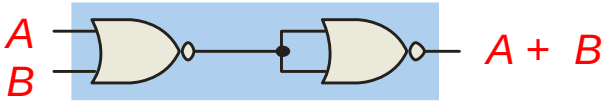
NOR gate

Universal Gates

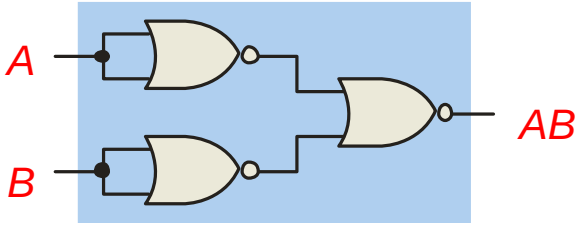
NOR gates are also **universal** gates and can form all of the basic gates.



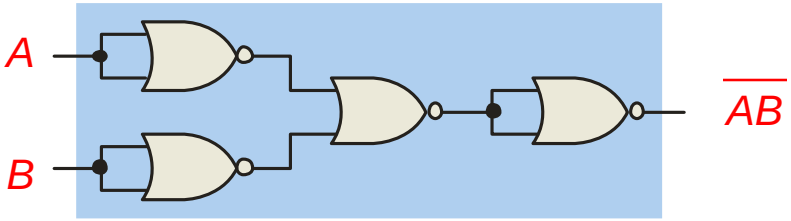
Inverter



OR gate



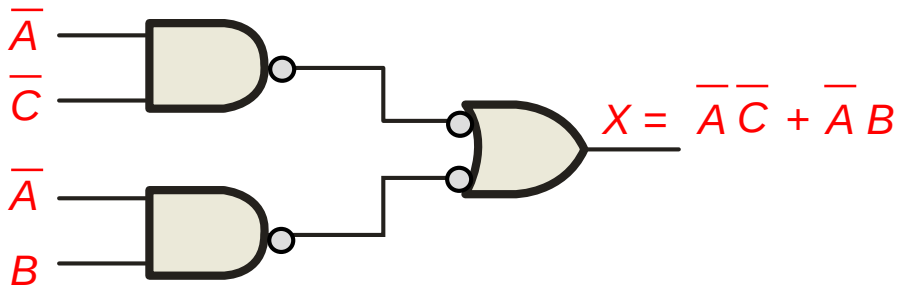
AND gate



NAND gate

NAND Logic

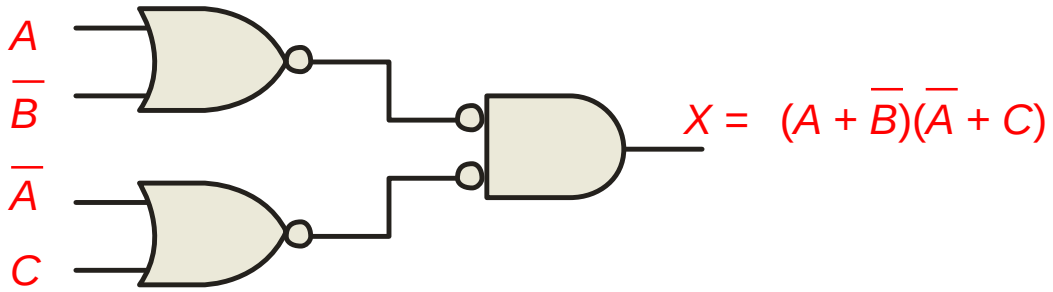
Recall from DeMorgan's theorem that $\overline{AB} = \overline{A} + \overline{B}$. By using equivalent symbols, it is simpler to read the logic of SOP forms. The earlier example shows the idea:



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

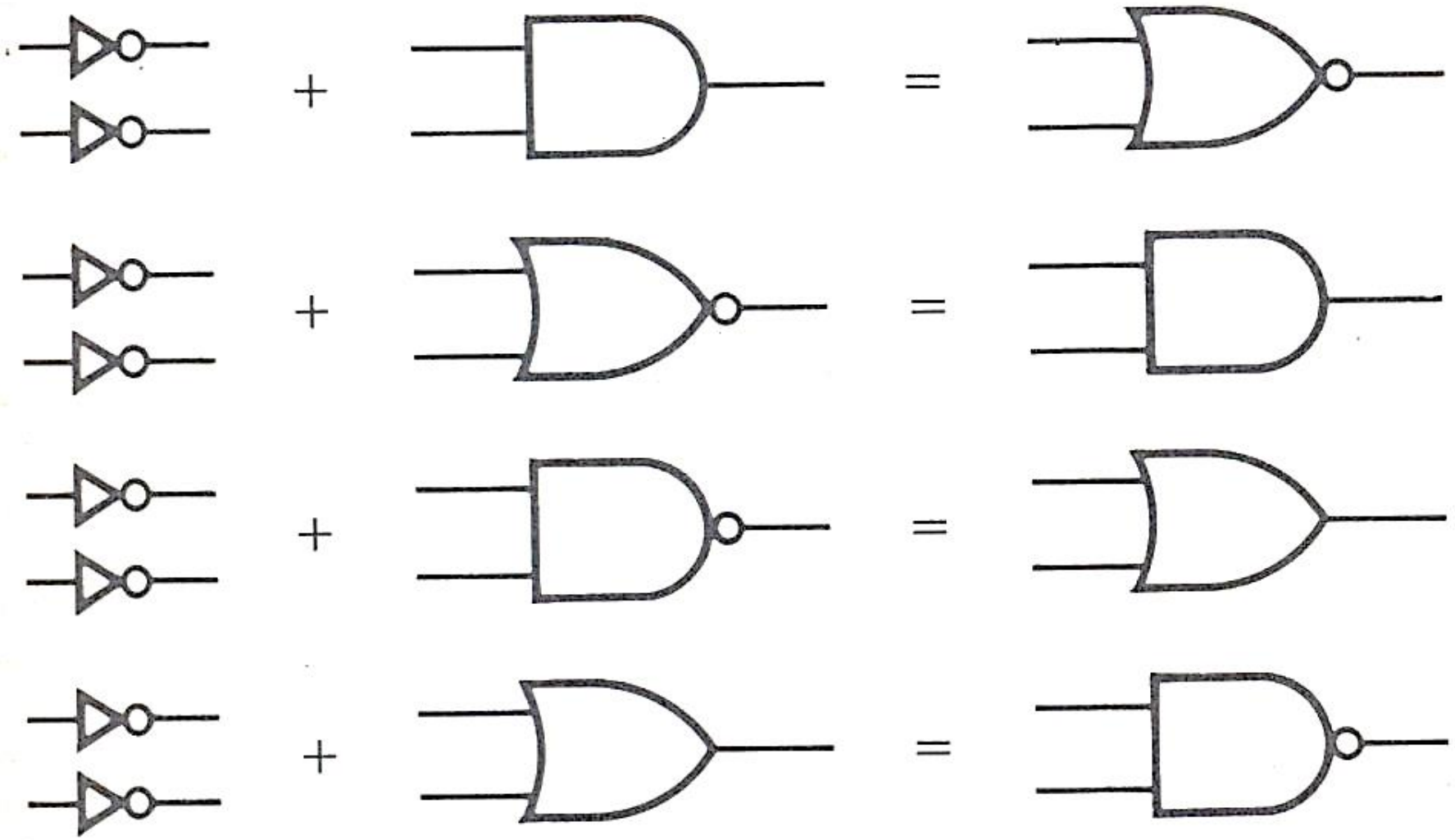
NOR Logic

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,

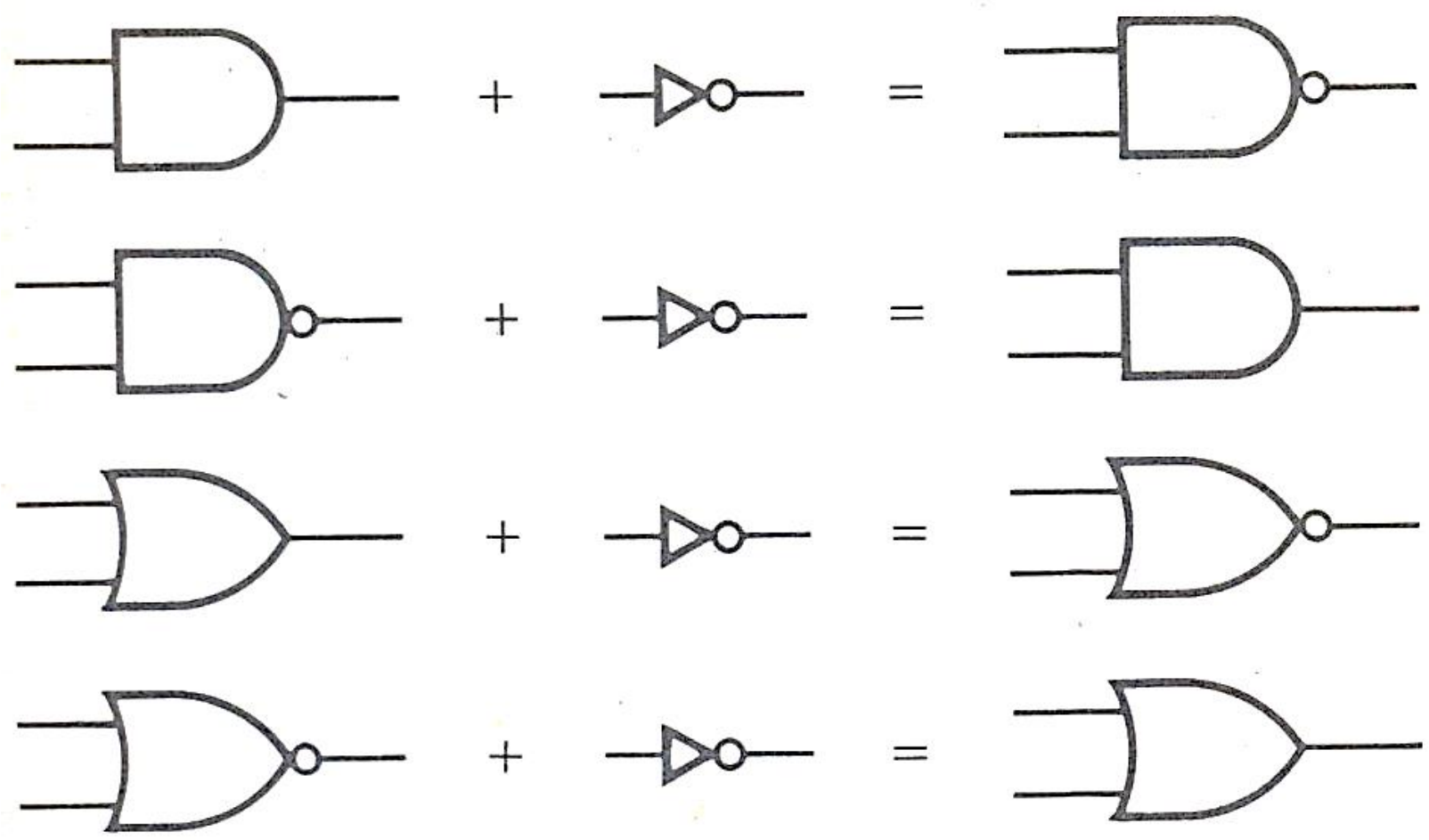


Again, the logic is easy to read if you cancel the two connected bubbles on a line.

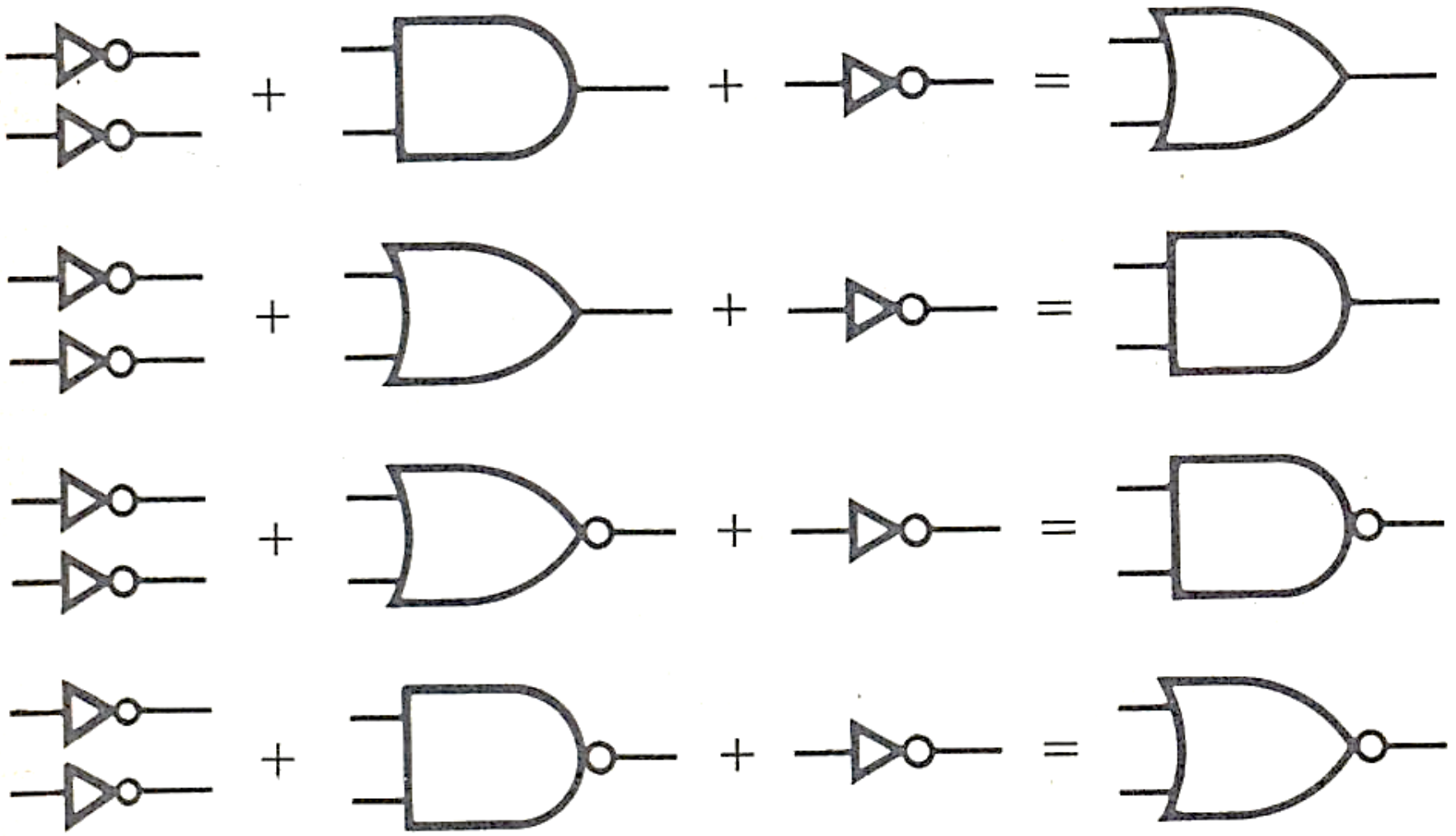
Logic Replica – Inverting Inputs



Logic Replica – Inverting Output



Logic Replica – Inverting Input(s) & Output



Universal gate	Either a NAND or a NOR gate. The term universal refers to a property of a gate that permits any logic function to be implemented by that gate or by a combination of gates of that kind.
Negative-OR	The dual operation of a NAND gate when the inputs are active-LOW.
Negative-AND	The dual operation of a NOR gate when the inputs are active-LOW.

Quiz

Assume an AOI expression is $\overline{AB + CD}$. The equivalent POS expression is

a. $(A + B)(C + D)$

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

c. $\overline{(A + B)}\,\overline{(C + D)}$

d. none of the above

Quiz

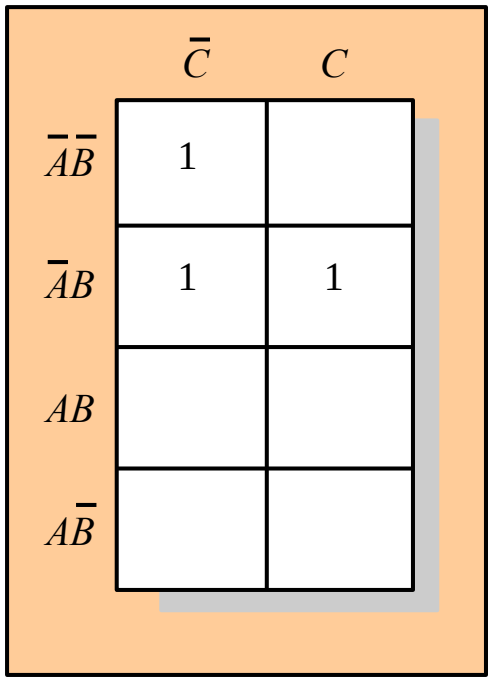
Reading the Karnaugh map, the logic expression is

a. $\overline{A}\,\overline{C} + \overline{A}B$

b. $\overline{A}B + A\overline{C}$

c. $A\overline{B} + B\overline{C}$

d. $\overline{A}B + \overline{A}\,\overline{C}$



Quiz

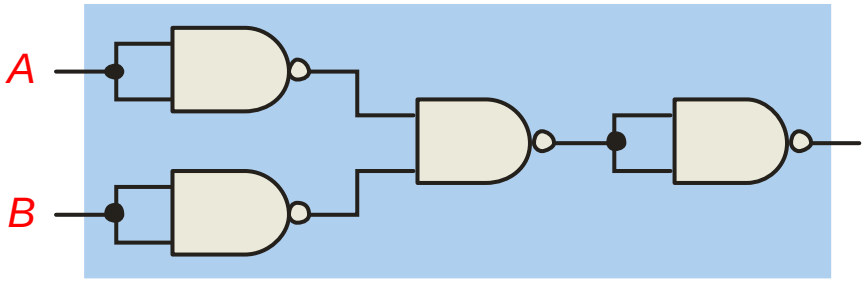
The circuit shown is equivalent to an

a. AND gate

b. XOR gate

c. OR gate

d. none of the above



Quiz

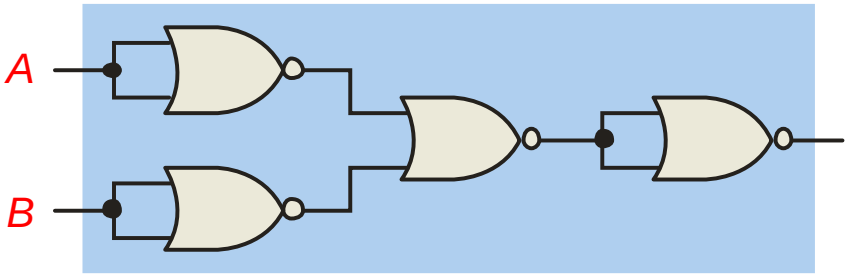
The circuit shown is equivalent to

a. an AND gate

b. an XOR gate

c. an OR gate

d. none of the above



Quiz

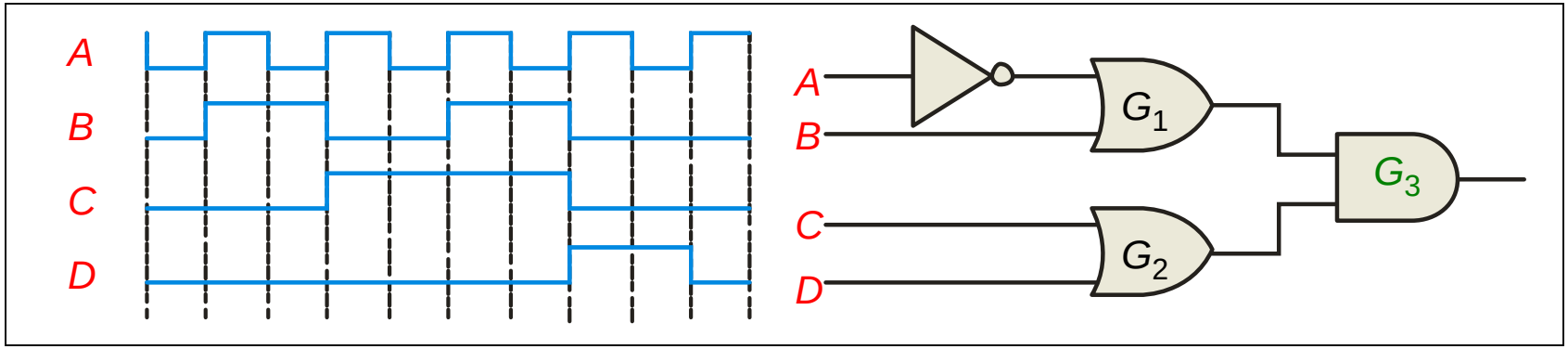
During the first *three* intervals for the pulsed circuit shown, the output of

a. G_1 is LOW and G_2 is LOW

b. G_1 is LOW and G_2 is HIGH

c. G_1 is HIGH and G_2 is LOW

d. G_1 is HIGH and G_2 is HIGH



Number System and Digital Codes

Decimal Numbers

The position of each digit in a weighted number system is assigned a weight based on the **base** or **radix** of the system. The radix of decimal numbers is ten, because only ten symbols (0 through 9) are used to represent any number.

The column weights of decimal numbers are powers of ten that increase from right to left beginning with $10^0 = 1$:

$\ldots 10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0$.

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

$10^2\ 10^1\ 10^0.\ 10^{-1}\ 10^{-2}\ 10^{-3}\ 10^{-4}\ \ldots$

Decimal Numbers

Decimal numbers can be expressed as the sum of the products of each digit times the column value for that digit. Thus, the number 9240 can be expressed as

$$(9 \times 10^3) + (2 \times 10^2) + (4 \times 10^1) + (0 \times 10^0)$$

or

$$9 \times 1{,}000 + 2 \times 100 + 4 \times 10 + 0 \times 1$$

Example

Express the number 480.52 as the sum of values of each digit.

Solution

$$480.52 = (4 \times 10^2) + (8 \times 10^1) + (0 \times 10^0) + (5 \times 10^{-1}) + (2 \times 10^{-2})$$

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:

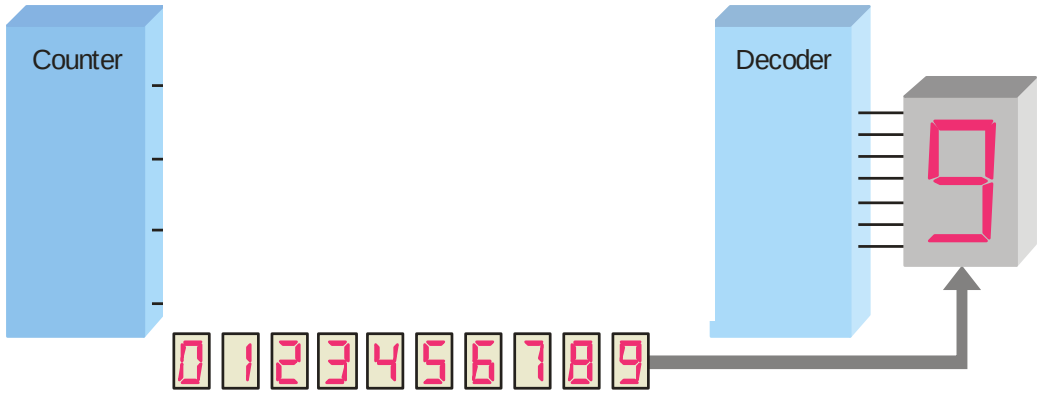
$$2^2\ 2^1\ 2^0.\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ \ldots$$

Binary Numbers

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 Conversions

The decimal equivalent of a binary number can be determined by adding the column values of all of the bits that are 1 and discarding all of the bits that are 0.

Example Convert the binary number 100101.01 to decimal.

Solution Start by writing the column weights; then add the weights that correspond to each 1 in the number.

2^5	2^4	2^3	2^2	2^1	2^0 .	2^{-1}	2^{-2}
32	16	8	4	2	1 .	½	¼
1	0	0	1	0	1.	0	1

32 +4 +1 +¼ = 37¼

Binary Conversions

You can convert a decimal whole number to binary by reversing the procedure. Write the decimal weight of each column and place 1's in the columns that sum to the decimal number.

Example

Convert the decimal number 49 to binary.

Solution

The column weights double in each position to the right. Write down column weights until the last number is larger than the one you want to convert.

2^6	2^5	2^4	2^3	2^2	2^1	2^0.
64	32	16	8	4	2	1.
0	1	1	0	0	0	1.

Binary Conversions

You can convert a decimal fraction to binary by repeatedly multiplying the fractional results of successive multiplications by 2. The carries form the binary number.

Example

Convert the decimal fraction 0.188 to binary by repeatedly multiplying the fractional results by 2.

Solution

$0.188 \times 2 = 0.376$ carry = 0
$0.376 \times 2 = 0.752$ carry = 0
$0.752 \times 2 = 1.504$ carry = 1
$0.504 \times 2 = 1.008$ carry = 1
$0.008 \times 2 = 0.016$ carry = 0

MSB ↓

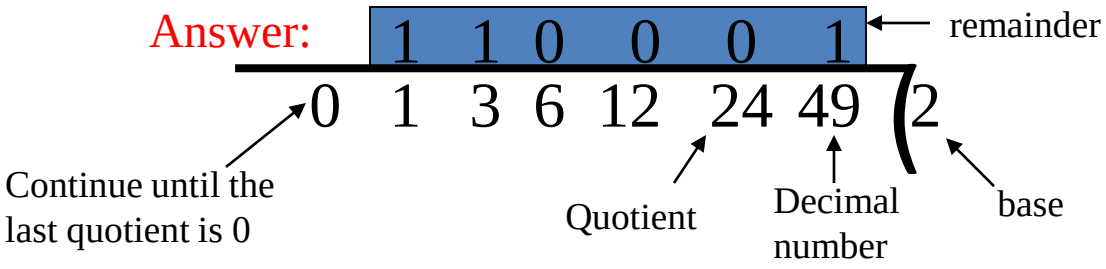
Answer = .00110 (for five significant digits)

Binary Conversions

You can convert decimal to any other base by repeatedly dividing by the base. For binary, repeatedly divide by 2:

Example Convert the decimal number 49 to binary by repeatedly dividing by 2.

Solution You can do this by "reverse division" and the answer will read from left to right. Put quotients to the left and remainders on top.



Binary Addition

The rules for binary addition are

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

When an input carry = 1 due to a previous result, the rules are

1 + 0 + 0 = 01	Sum = 1, carry = 0
1 + 0 + 1 = 10	Sum = 0, carry = 1
1 + 1 + 0 = 10	Sum = 0, carry = 1
1 + 1 + 1 = 10	Sum = 1, carry = 1

Binary Addition

Example Add the binary numbers 00111 and 10101 and show the equivalent decimal addition.

Solution

$$\begin{array}{r} {\scriptstyle 0\ 1\ 1\ 1} \\ 00111 \\ +\ 10101 \\ \hline 11100 \end{array} \quad \begin{array}{r} 7 \\ +\ 21 \\ \hline = 28 \end{array}$$

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

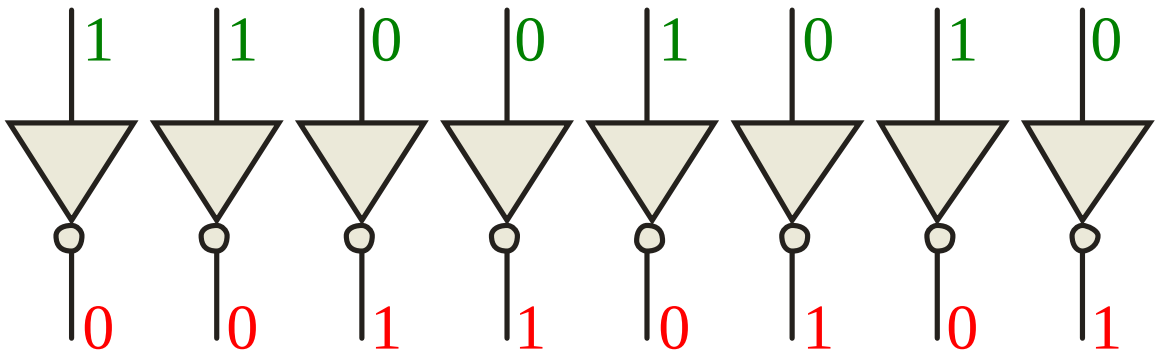
$$\begin{array}{rr} {\scriptstyle 1\ 1\ 1} & \\ 10101 & 21 \\ 00111 & 7 \\ \hline 01110 & = 14 \end{array}$$

1's Complement

The 1's complement of a binary number is just the inverse of the digits. To form the 1's complement, change all 0's to 1's and all 1's to 0's.

For example, the 1's complement of 11001010 is 00110101

In digital circuits, the 1's complement is formed by using inverters:



2's Complement

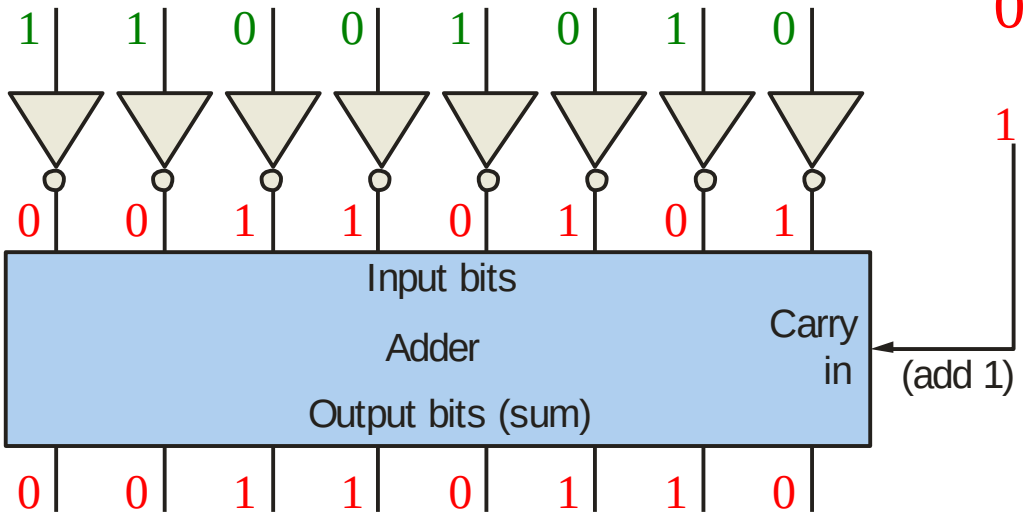
The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

Recall that the 1's complement of 11001010 is 00110101 (1's complement)

To form the 2's complement, add 1:

$$\begin{array}{r} 00110101 \\ +\quad\;\; 1 \\ \hline 00110110 \end{array}$$

(2's complement)

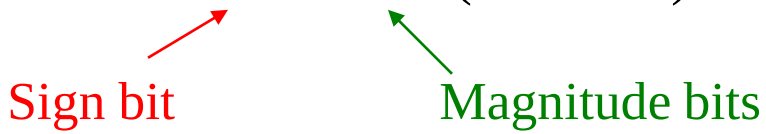


Signed Binary Numbers

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).



Signed Binary Numbers

Negative numbers are written as the 2's complement of the corresponding positive number.

The negative number –58 is written as:

–58 = 11000110 (complement form)

Sign bit — Magnitude bits

An easy way to read a signed number that uses this notation is to assign the sign bit a column weight of –128 (for an 8-bit number). Then add the column weights for the 1's.

Example

Assuming that the sign bit = –128, show that 11000110 = –58 as a 2's complement signed number:

Solution

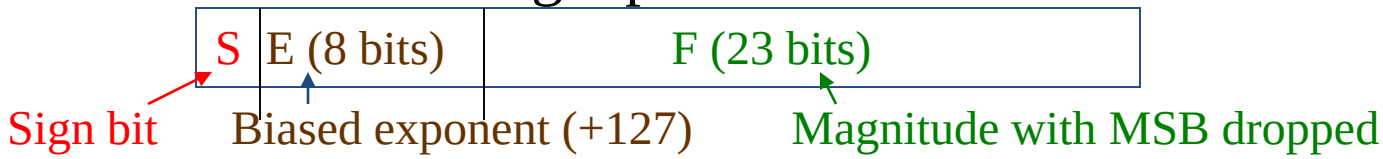
Column weights: –128 64 32 16 8 4 2 1.

1 1 0 0 0 1 1 0

–128 +64 +4 +2 = –58

Floating Point Numbers

Floating point notation is capable of representing very large or small numbers by using a form of scientific notation. A 32-bit single precision number is illustrated.



Example

Express the speed of light, c, in single precision floating point notation. ($c = 0.2998 \times 10^9$)

Solution

In binary, $c = 0001\ 0001\ 1101\ 1110\ 1001\ 0101\ 1100\ 0000_2$.

In scientific notation, $c = 1.001\ 1101\ 1110\ 1001\ 0101\ 1100\ 0000 \times 2^{28}$.

S = 0 because the number is positive. E = $28 + 127 = 155_{10} = 1001\ 1011_2$.

F is the next 23 bits after the first 1 is dropped.

In floating point notation, $c =$

0	10011011	001 1101 1110 1001 0101 1100

Arithmetic Operations with Signed Numbers

Using the signed number notation with negative numbers in 2's complement form simplifies addition and subtraction of signed numbers.

Rules for **addition**: Add the two signed numbers. Discard any final carries. The result is in signed form.

Examples:

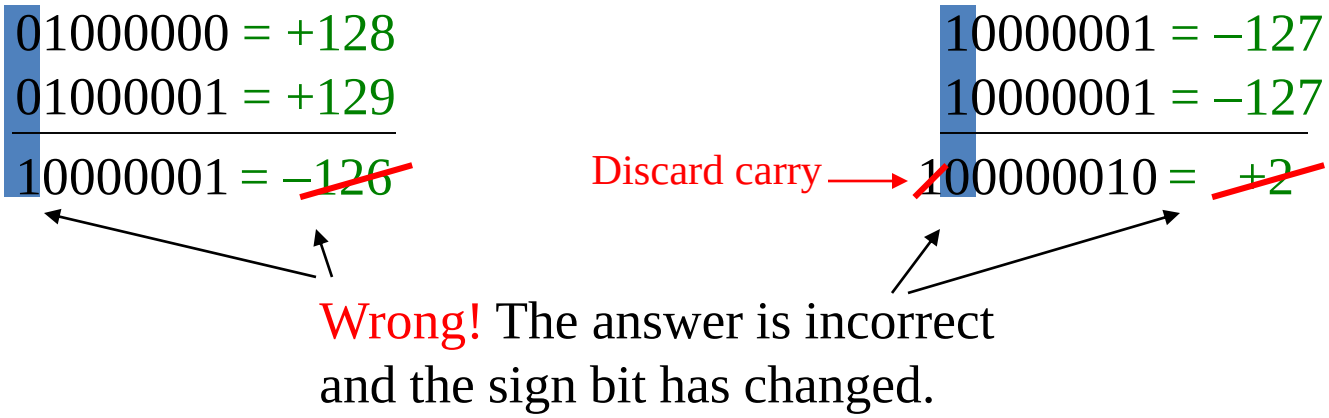
00011110 = +30	00001110 = +14	11111111 = –1
00001111 = +15	11101111 = –17	11111000 = –8
00101101 = +45	11111101 = –3	1 11110111 = –9

Discard carry

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

Rules for **subtraction**: 2's complement the subtrahend and add the numbers. Discard any final carries. The result is in signed form.

Repeat the examples done previously, but subtract:

	00011110	(+30)		00001110	(+14)		11111111	(–1)
–	00001111	–(+15)	–	11101111	–(–17)	–	11111000	–(–8)

2's complement subtrahend and add:

	00011110	= +30		00001110	= +14		11111111	= –1
	11110001	= –15		00010001	= +17		00001000	= +8
	100001111	= +15		00011111	= +31		100000111	= +7

Discard carry

Discard carry

Hexadecimal Numbers

Hexadecimal uses sixteen characters to represent numbers: the numbers 0 through 9 and the alphabetic characters A through F.

Large binary number can easily be converted to hexadecimal by grouping bits 4 at a time and writing the equivalent hexadecimal character.

Example

Express $1001\ 0110\ 0000\ 1110_2$ in hexadecimal:

Solution

Group the binary number by 4-bits starting from the right. Thus, 960E

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Hexadecimal Numbers

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

Column weights { 16^3 16^2 16^1 16^0.
 4096 256 16 1 .

Example Express $1A2F_{16}$ in decimal.

Solution Start by writing the column weights:

4096 256 16 1

1 A 2 F_{16}

$1(4096) + 10(256) + 2(16) + 15(1) = 6703_{10}$

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Octal Numbers

Octal uses eight characters the numbers 0 through 7 to represent numbers. There is no 8 or 9 character in octal.

Binary number can easily be converted to octal by grouping bits 3 at a time and writing the equivalent octal character for each group.

Example

Express $1\ 001\ 011\ 000\ 001\ 110_2$ in octal:

Solution

Group the binary number by 3-bits starting from the right. Thus, 113016_8

Decimal	Octal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

Octal Numbers

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

Column weights	8^3	8^2	8^1	8^0	.
	512	64	8	1	.

Example Express 3702_8 in decimal.

Solution Start by writing the column weights:

512	64	8	1
3	7	0	2_8

$3(512) + 7(64) + 0(8) + 2(1) = 1986_{10}$

Decimal	Octal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111

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.

The table illustrates the difference between straight binary and BCD. BCD represents each decimal digit with a 4-bit code. Notice that the codes 1010 through 1111 are not used in BCD.

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	00010000
11	1011	00010001
12	1100	00010010
13	1101	00010011
14	1110	00010100
15	1111	00010101

BCD

You can think of BCD in terms of column weights in groups of four bits. For an 8-bit BCD number, the column weights are: 80 40 20 10 8 4 2 1.

Question: What are the column weights for the BCD number 1000 0011 0101 1001?

Answer:

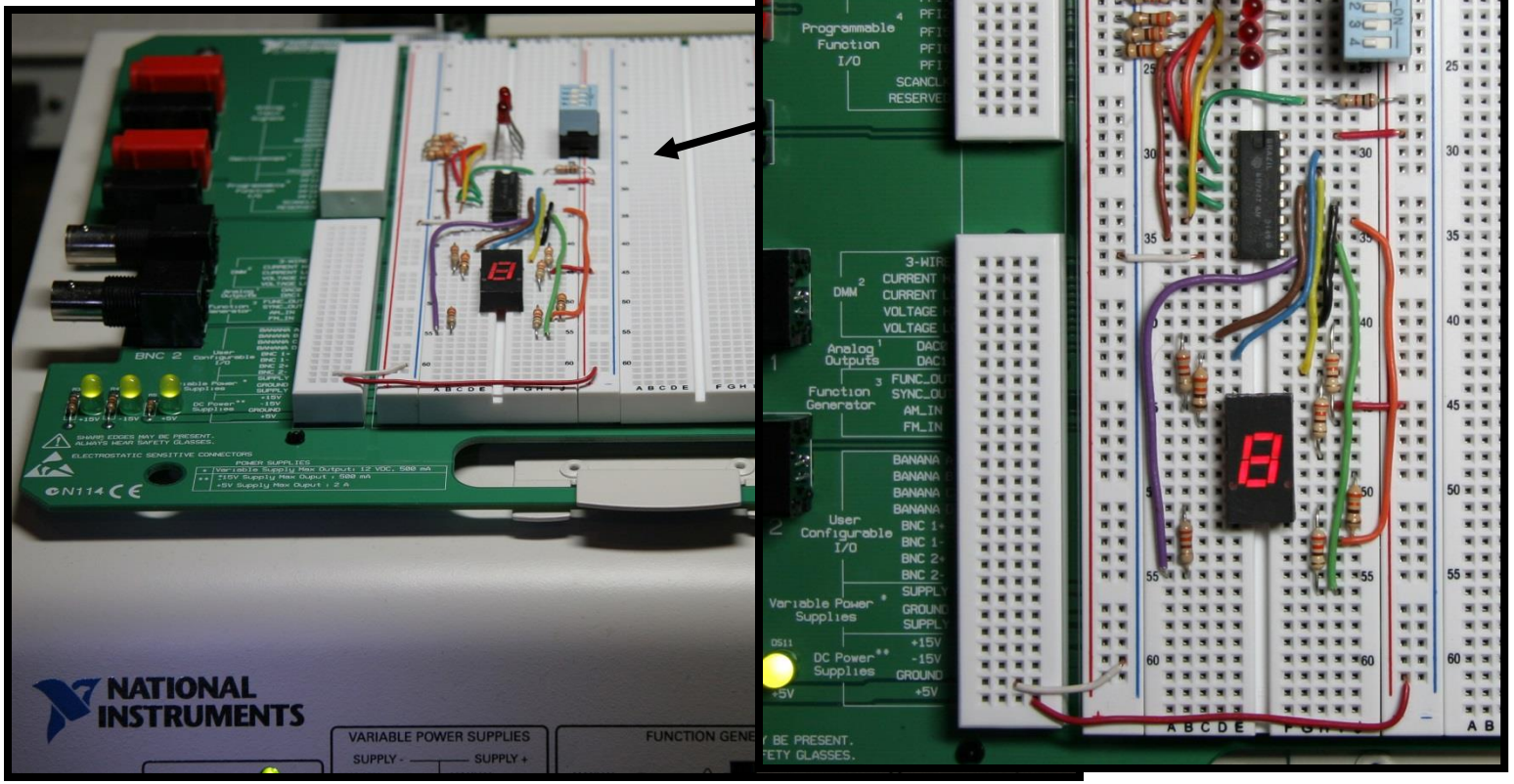
8000 4000 2000 1000 800 400 200 100 80 40 20 10 8 4 2 1

Note that you could add the column weights where there is a 1 to obtain the decimal number. For this case:

$$8000 + 200 + 100 + 40 + 10 + 8 + 1 = 8359_{10}$$

BCD

A lab experiment in which BCD is converted to decimal is shown.



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.

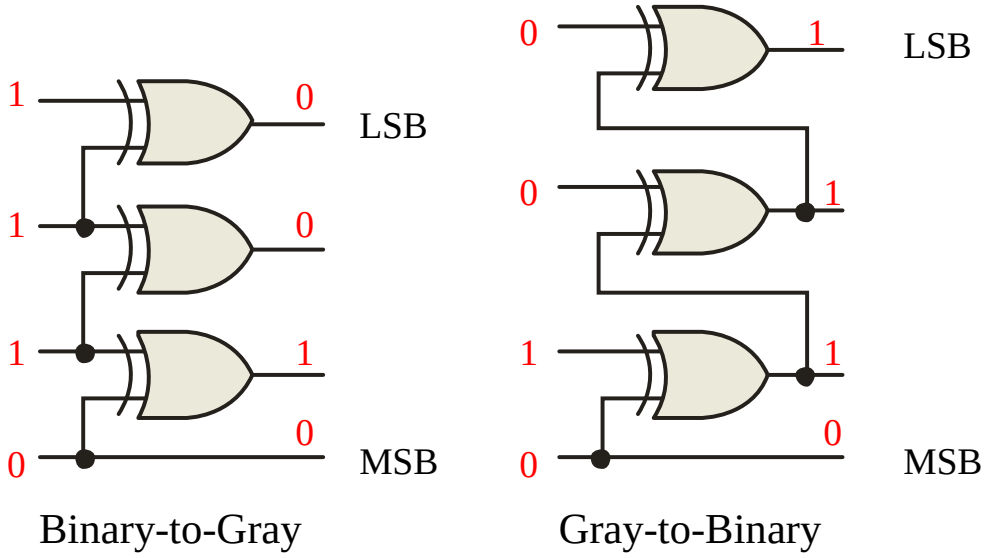
Decimal	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

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.

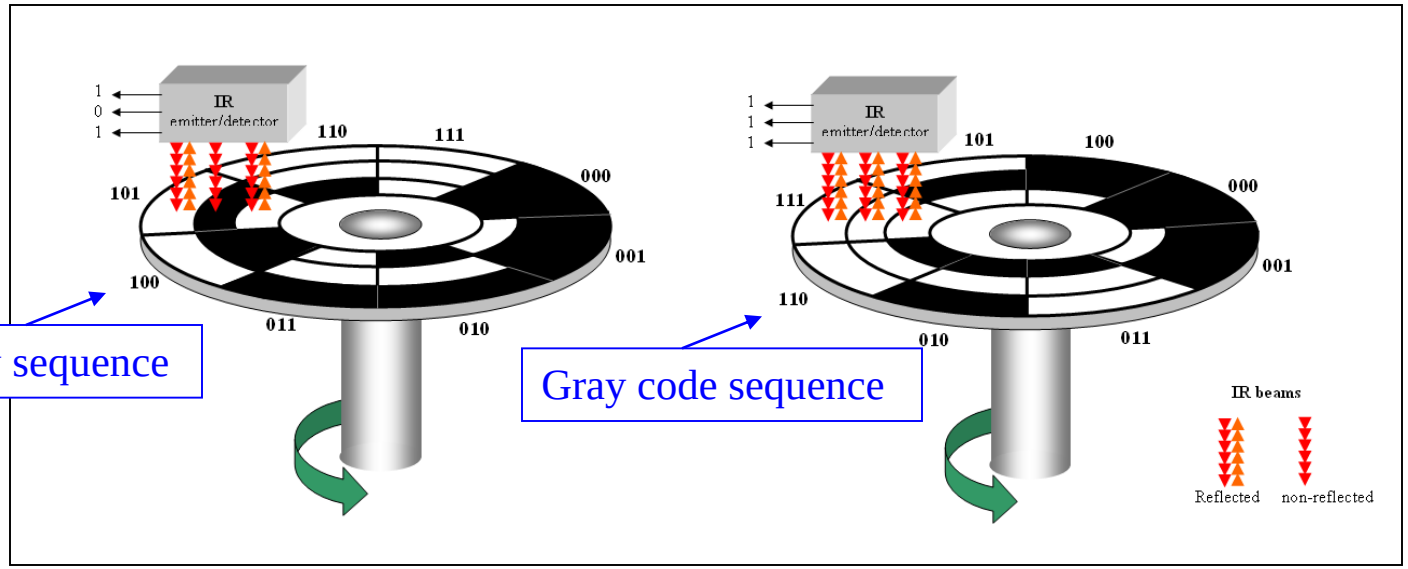
Example Show the conversion of binary 0111 to Gray and back.

Solution



Gray code

A shaft encoder is a typical application. Three IR emitter/detectors are used to encode the position of the shaft. The encoder on the left uses binary and can have three bits change together, creating a potential error. The encoder on the right uses gray code and only 1-bit changes, eliminating potential errors.



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.

In 1981, IBM introduced extended ASCII, which is an 8-bit code and increased the character set to 256. Other extended sets (such as Unicode) have been introduced to handle characters in languages other than English.

Parity Method

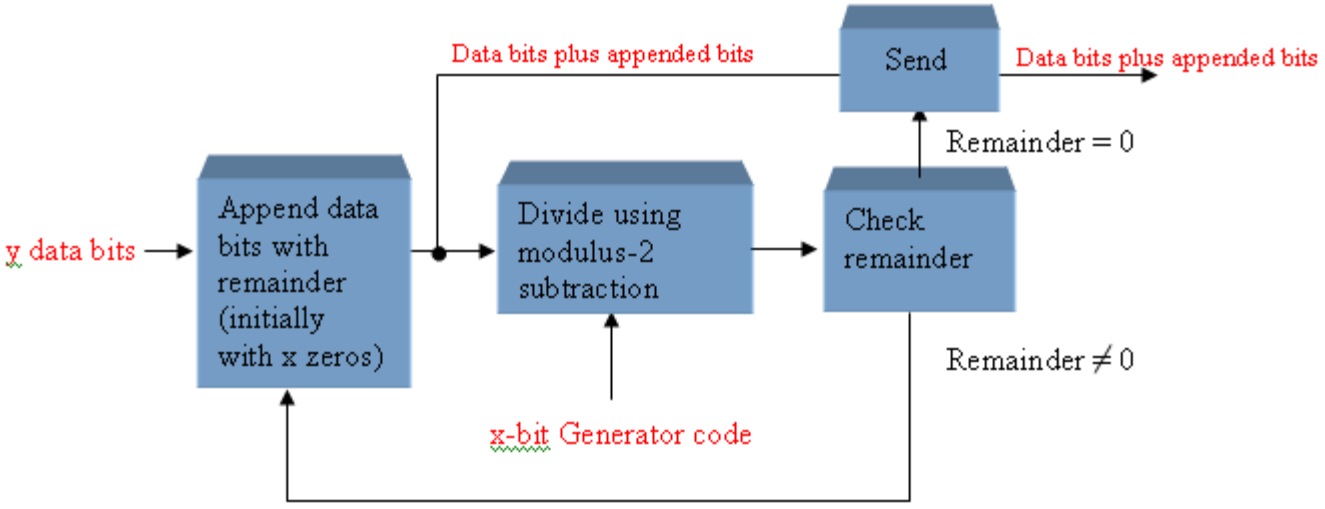
The parity method is a method of error detection for simple transmission errors involving one bit (or an odd number of bits). A parity bit is an "extra" bit attached to a group of bits to force the number of 1's to be either even (even parity) or odd (odd parity).

Example The ASCII character for "a" is 1100001 and for "A" is 1000001. What is the correct bit to append to make both of these have odd parity?

Solution The ASCII "a" has an odd number of bits that are equal to 1; therefore the parity bit is 0. The ASCII "A" has an even number of bits that are equal to 1; therefore the parity bit is 1.

Cyclic Redundancy Check

The cyclic redundancy check (CRC) is an error detection method that can detect multiple errors in larger blocks of data. At the sending end, a checksum is appended to a block of data. At the receiving end, the check sum is generated and compared to the sent checksum. If the check sums are the same, no error is detected.



Byte	A group of eight bits
Floating-point number	A number representation based on scientific notation in which the number consists of an exponent and a mantissa.
Hexadecimal	A number system with a base of 16.
Octal	A number system with a base of 8.
BCD	Binary coded decimal; a digital code in which each of the decimal digits, 0 through 9, is represented by a group of four bits.

Alphanumeric	Consisting of numerals, letters, and other characters
ASCII	American Standard Code for Information Interchange; the most widely used alphanumeric code.
Parity	In relation to binary codes, the condition of evenness or oddness in the number of 1s in a code group.
Cyclic redundancy check (CRC)	A type of error detection code.

Solid-State Circuits

Inverter

V_{IN}	V_{OUT}
Low	High
High	Low

V_{IN}	V_{OUT}
0	1
1	0

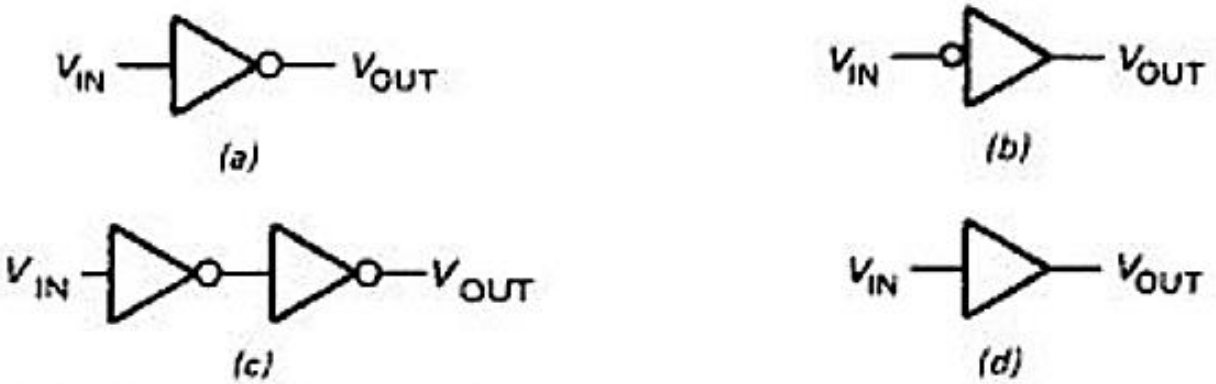
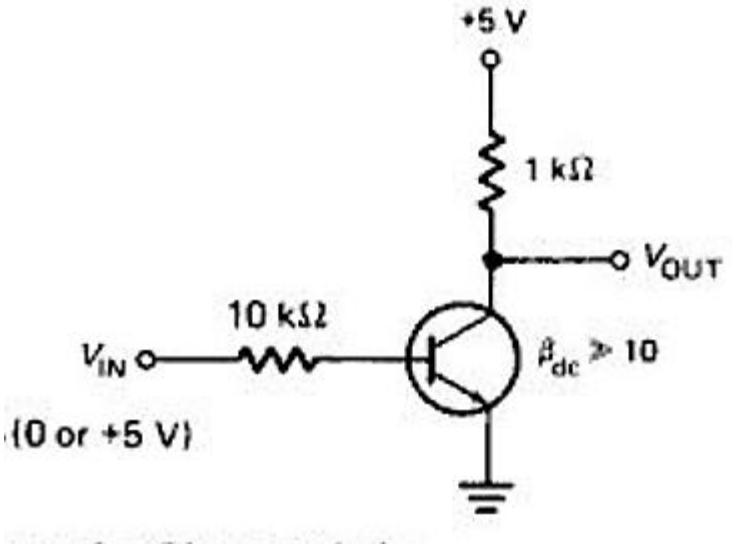


Fig. 2-2 Logic symbols: (*a*) inverter; (*b*) another inverter symbol; (*c*) double inverter; (*d*) buffer.

Simple Solid-State Circuits

Transistor Inverter



V_{IN}	V_{OUT}
Low	High
High	Low

V_{IN}	V_{OUT}
0	1
1	0

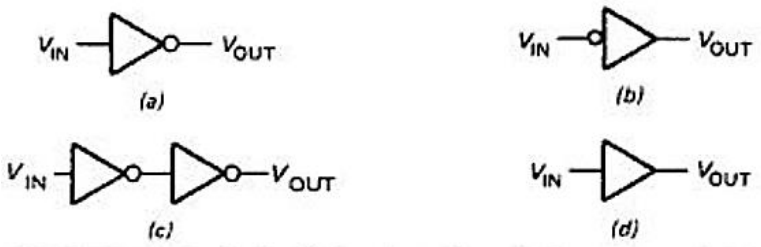
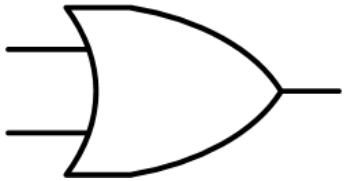


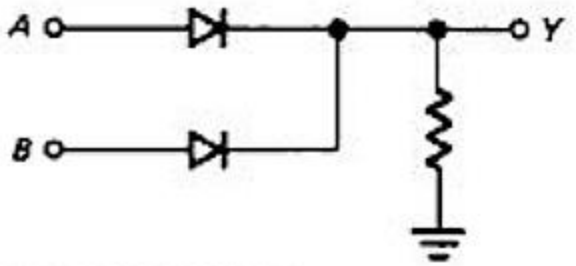
Fig. 2-2 Logic symbols: (*a*) inverter; (*b*) another inverter symbol; (*c*) double inverter; (*d*) buffer.

Diode OR Gate

TABLE 2-3. TWO INPUT OR GATE

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

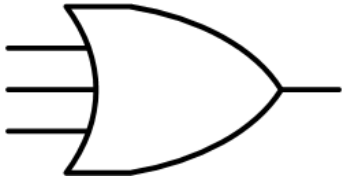


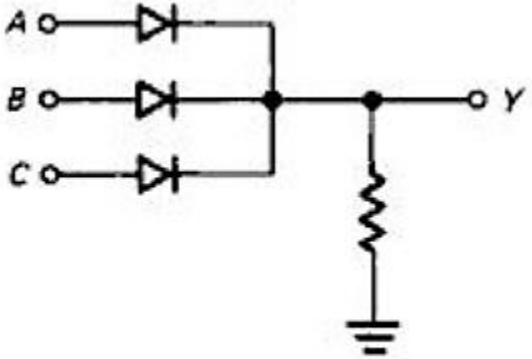


2-input diode OR gate.

TABLE 2-4. THREE-INPUT OR GATE

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1





3-input diode OR gate.

Diode AND Gate

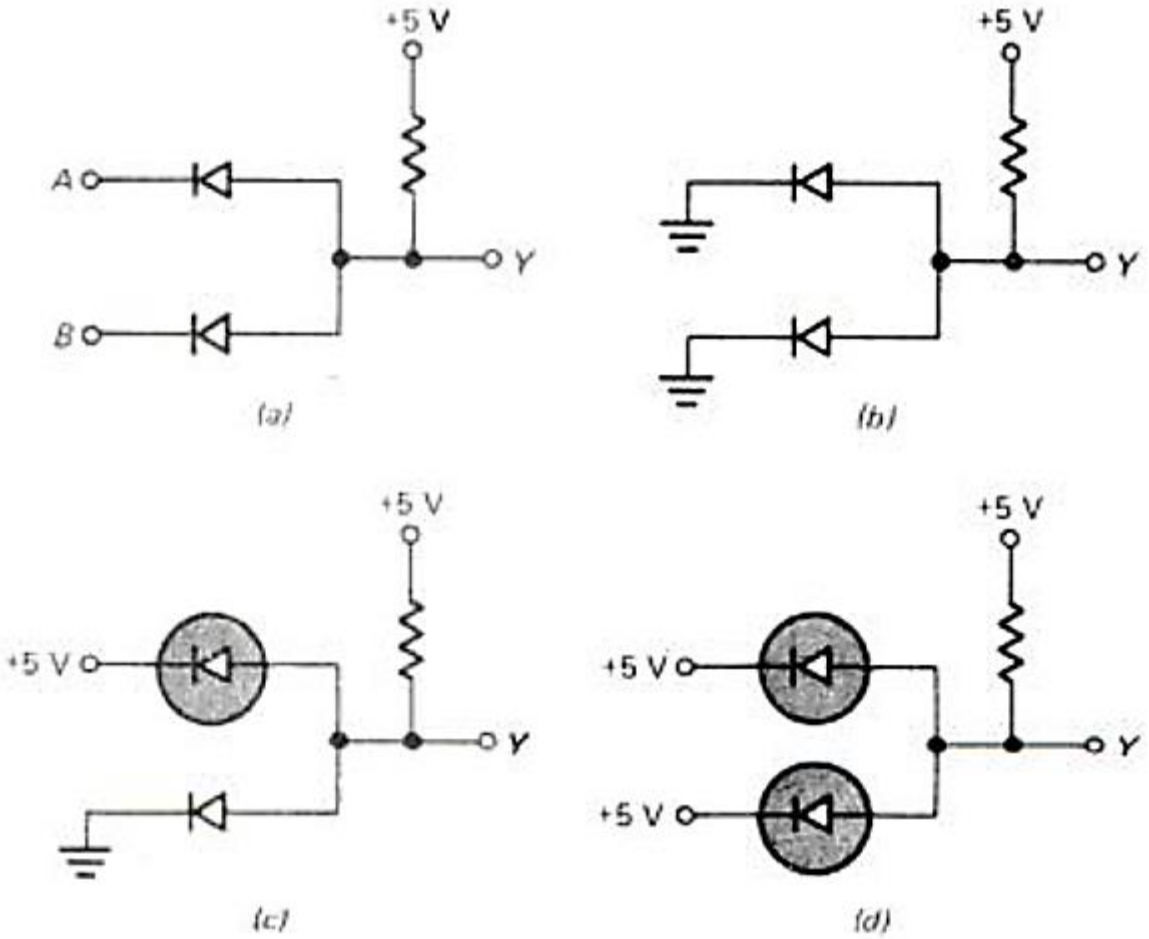


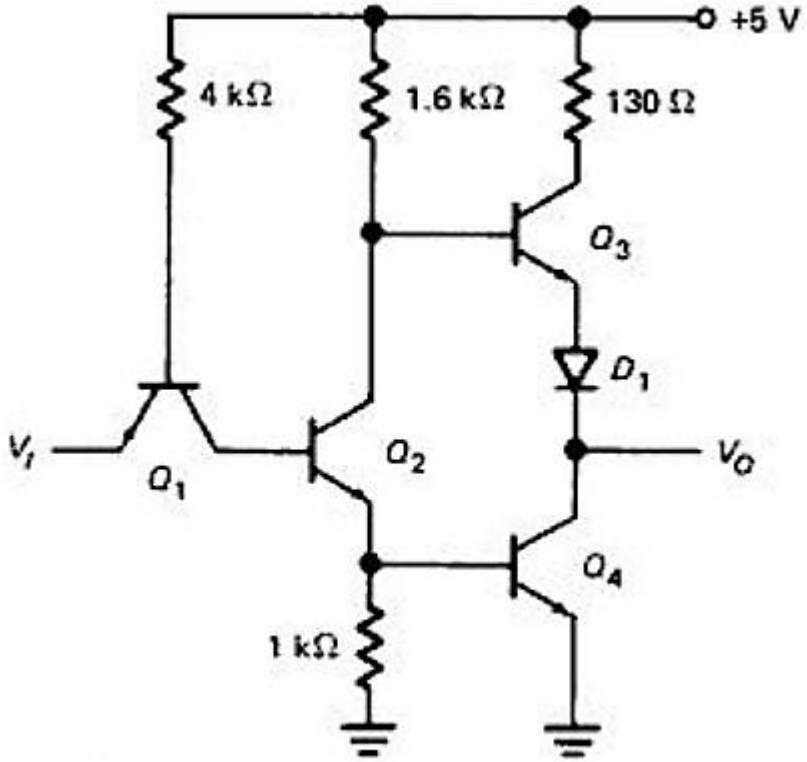
Fig. 2-8 A 2-input AND gate. (*a*) circuit; (*b*) both inputs low; (*c*) 1 low input, 1 high; (*d*) both inputs high.

Standard Solid-State Circuits

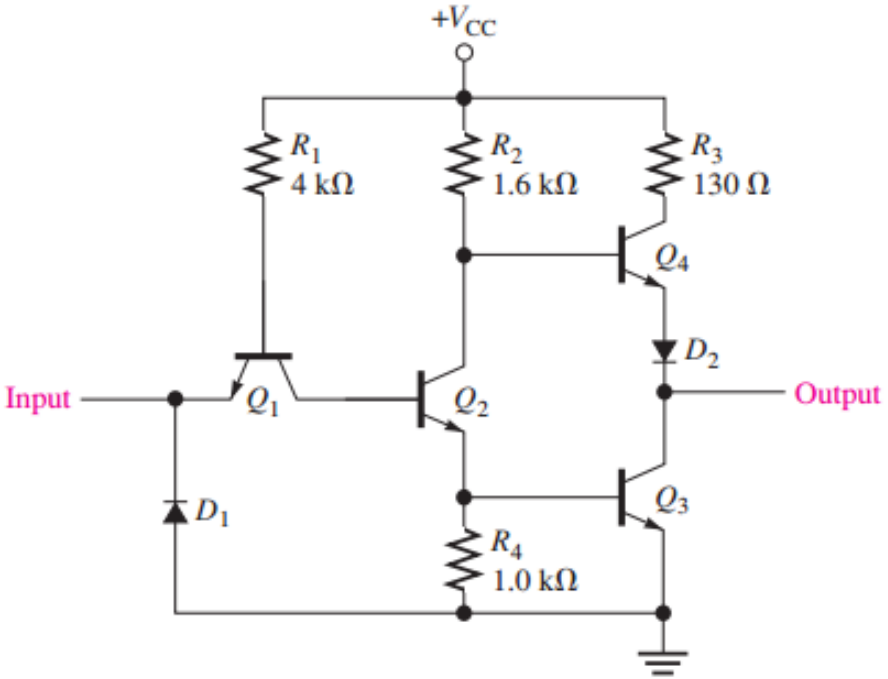
Standard TTL Inverter

V_{IN}	V_{OUT}
Low	High
High	Low

V_{IN}	V_{OUT}
0	1
1	0



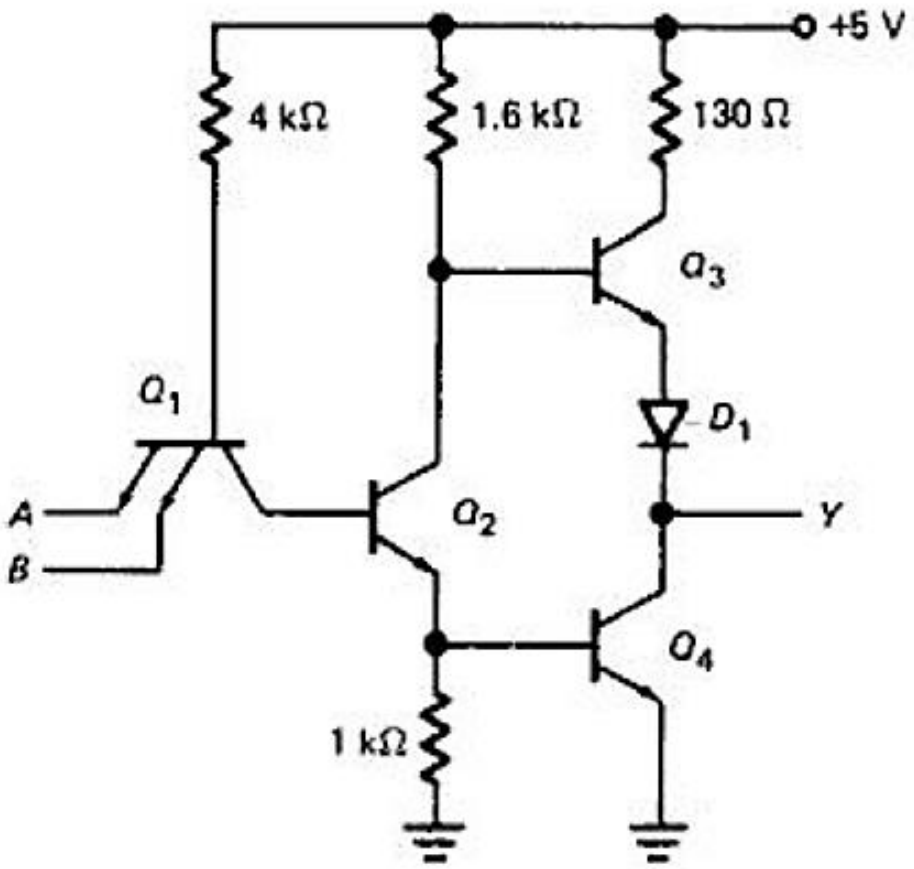
TTL inverter.



Standard TTL NAND Gate

TWO-INPUT NAND GATE

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



Standard TTL NAND gate.

Standard TTL NOR Gate

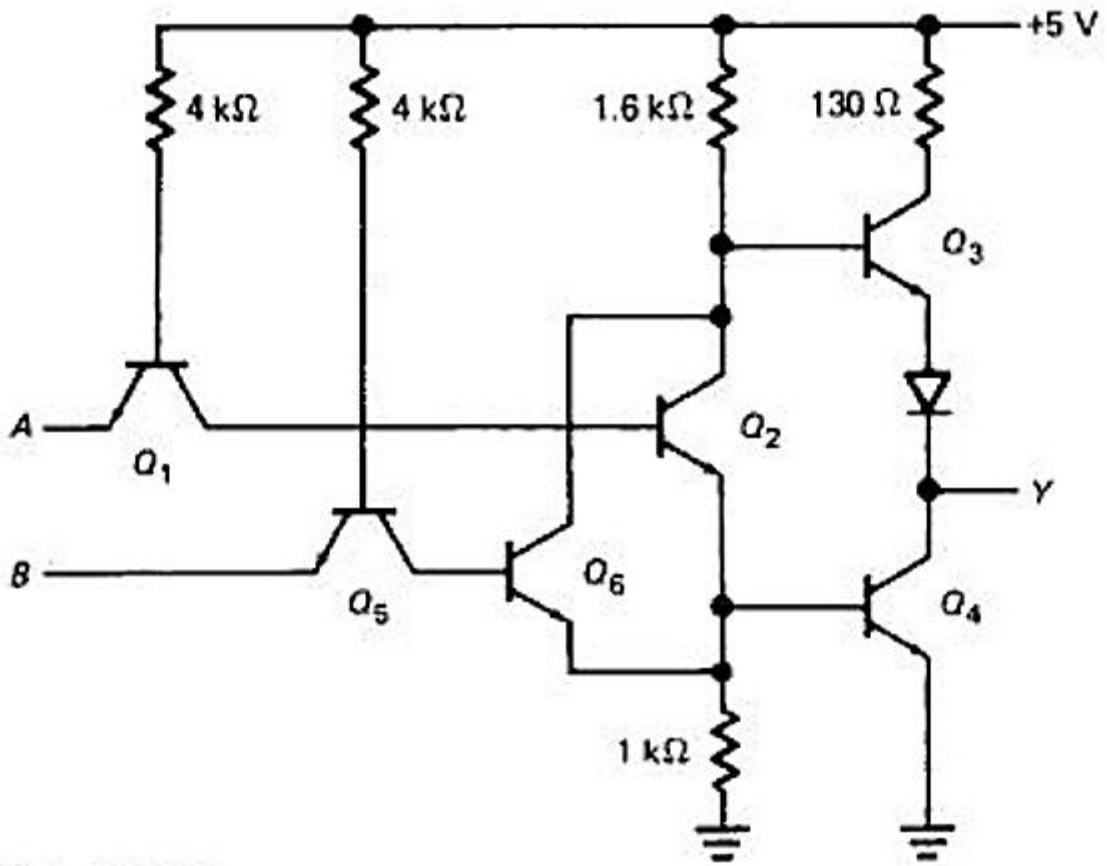
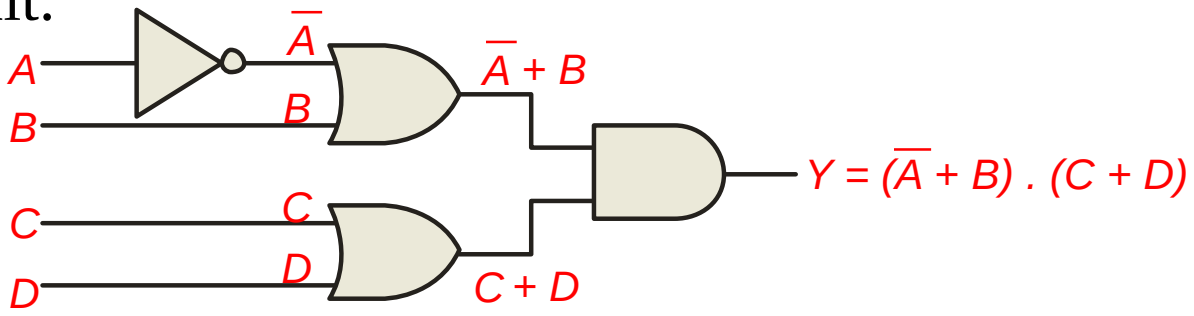


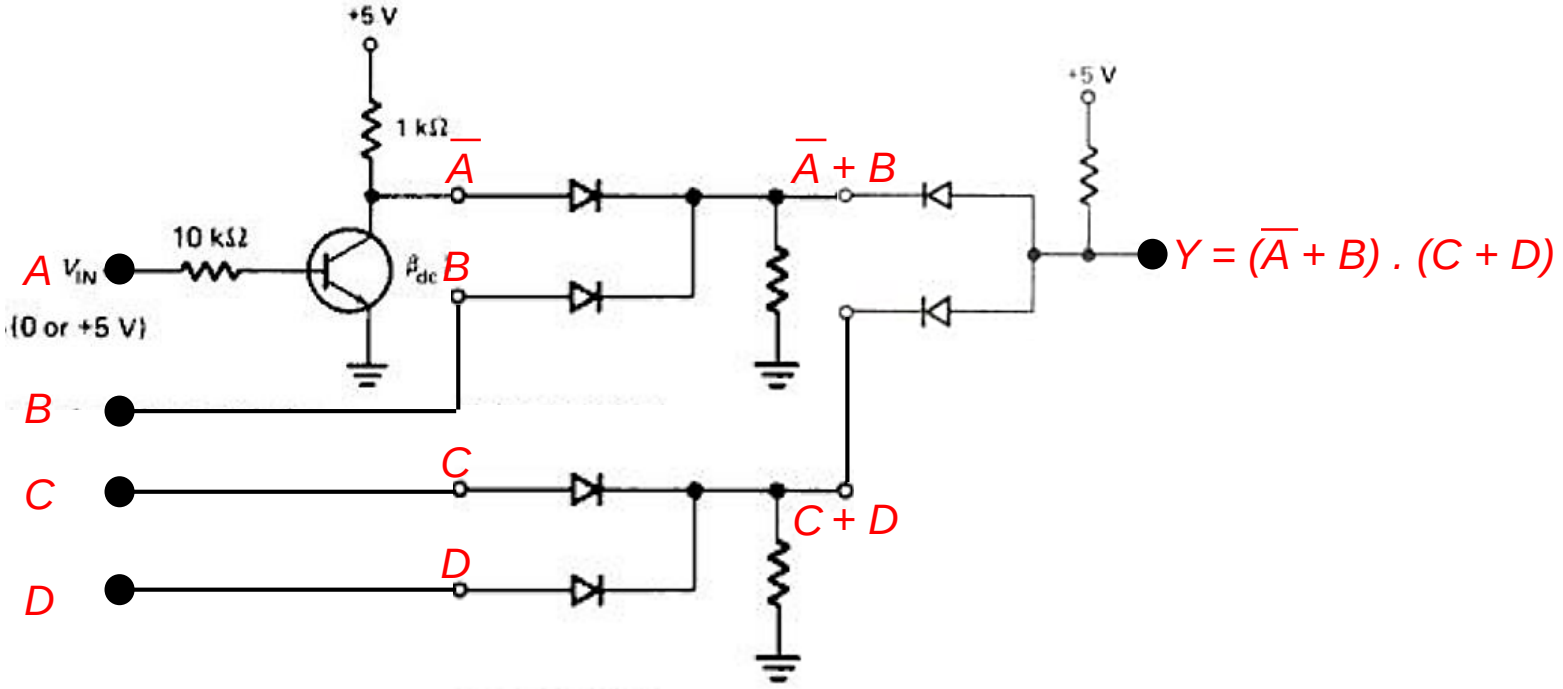
Fig. 4-8 TTL NOR gate.

Example

Transform the given Logic Map to Simple TTL/DDL Logic based Solid-State Circuit.

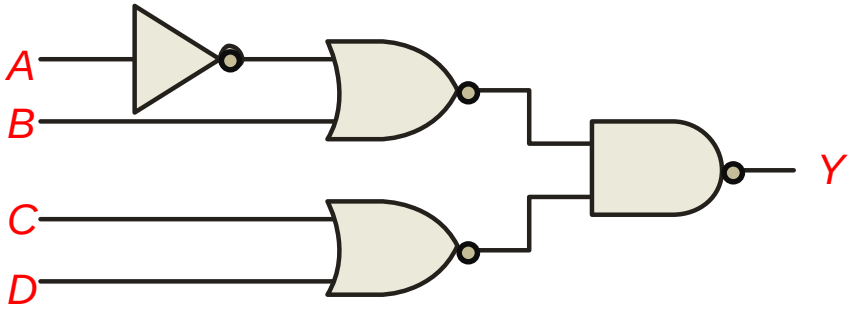
Solution



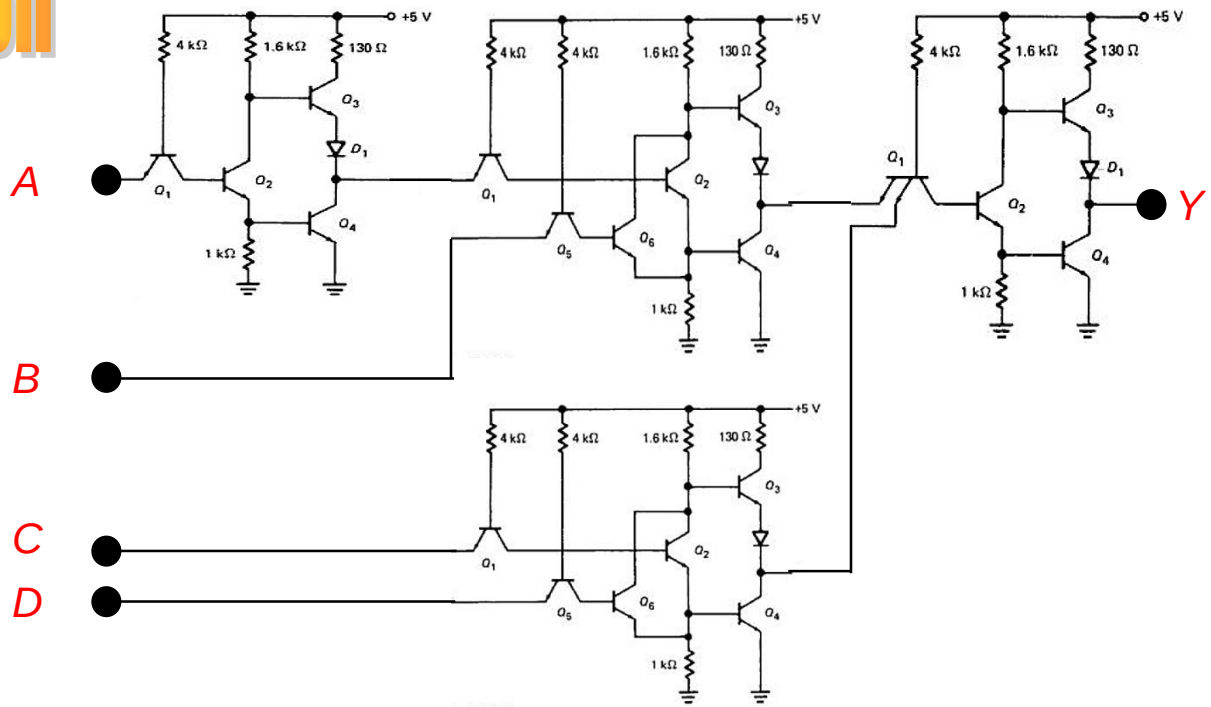


Example

Transform the given Logic Map to Standard & Detailed TTL Logic based Solid-State Circuit.



Solution



Continued to…

Digital Computing Fundamentals

Volume-2

Digital Logic & Applied Circuits Design & Analysis

Compiled by

Mr. Hussain Saleem

Assistant Professor (Sr.)

Department of Computer Science, UBIT,
University of Karachi, Pakistan.

15th April 2017

This material is extracted from

Digital Fundamentals

Tenth Edition

Thomas L. Floyd

Courtesy of Pearson Education Inc.