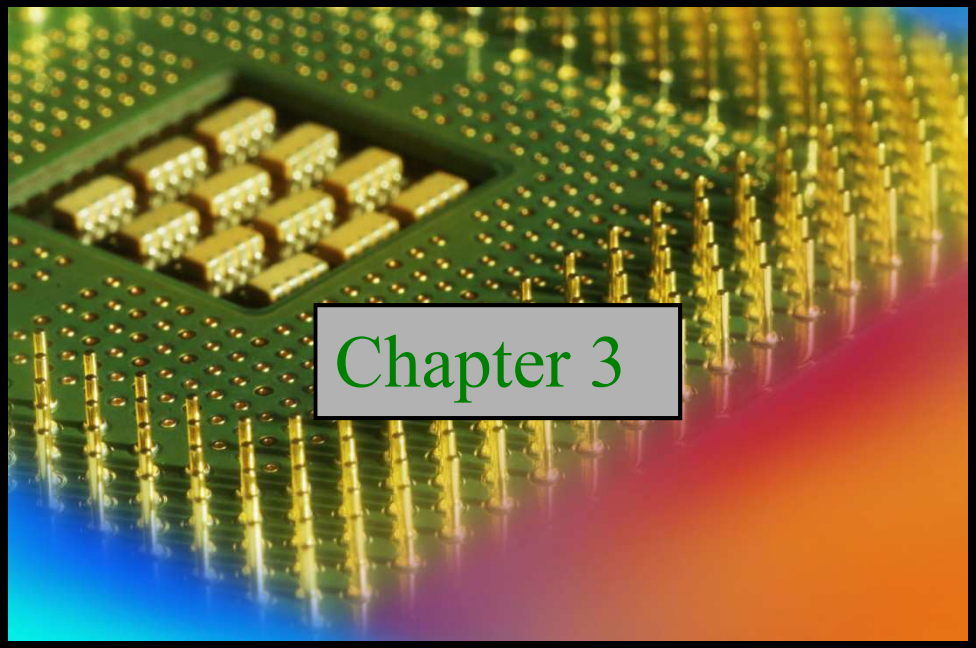
Digital Fundamentals

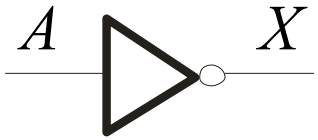
Tenth Edition

Floyd



Summary

The Inverter



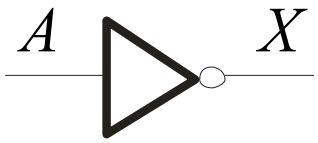
The inverter performs the Boolean **NOT** operation. When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW.

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

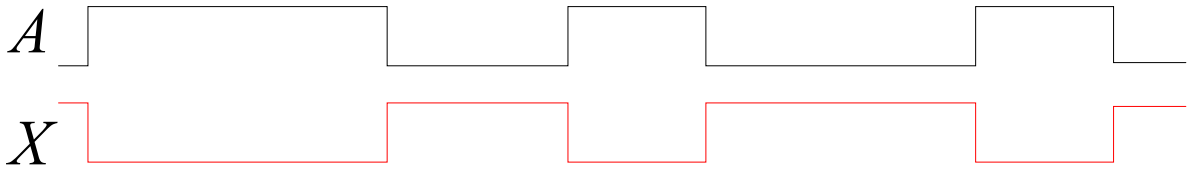
The **NOT** operation (complement) is shown with an overbar. Thus, the Boolean expression for an inverter is $X = \overline{A}$.

Summary

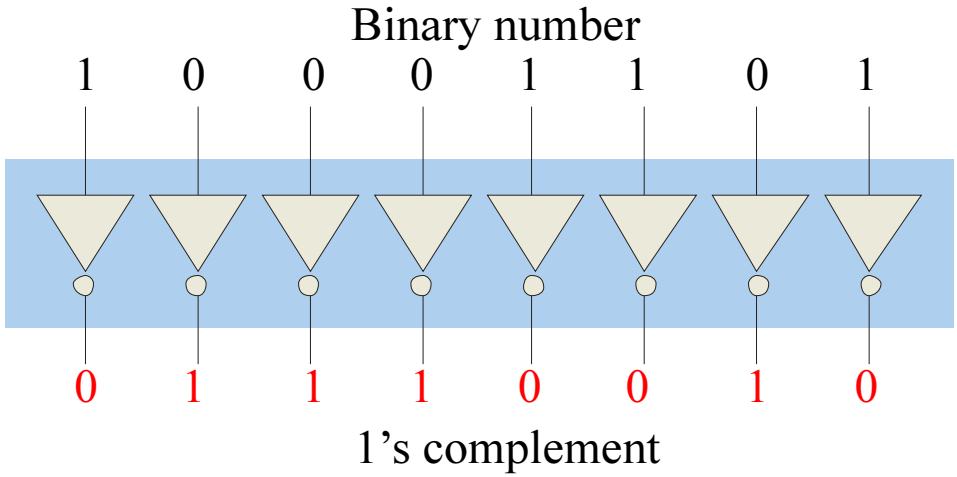
The Inverter



Example waveforms:

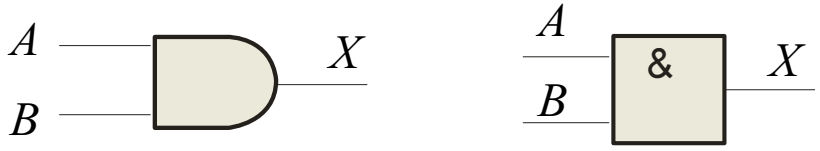


A group of inverters can be used to form the 1's complement of a binary number:



Summary

The AND Gate



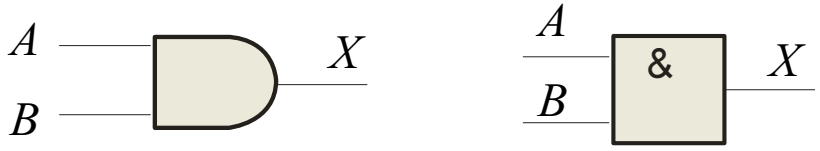
The **AND gate** produces a HIGH output when all inputs are HIGH; otherwise, the output is LOW. For a 2-input gate, the truth table is

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

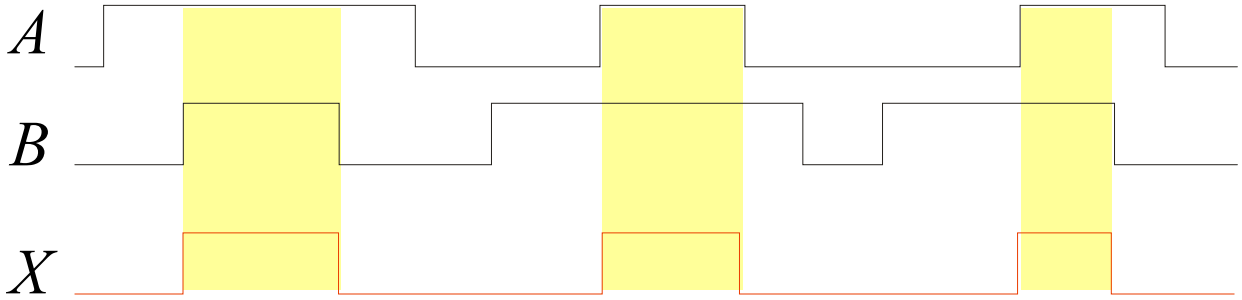
The **AND** operation is usually shown with a dot between the variables but it may be implied (no dot). Thus, the AND operation is written as $X = A \cdot B$ or $X = AB$.

Summary

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

Summary

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

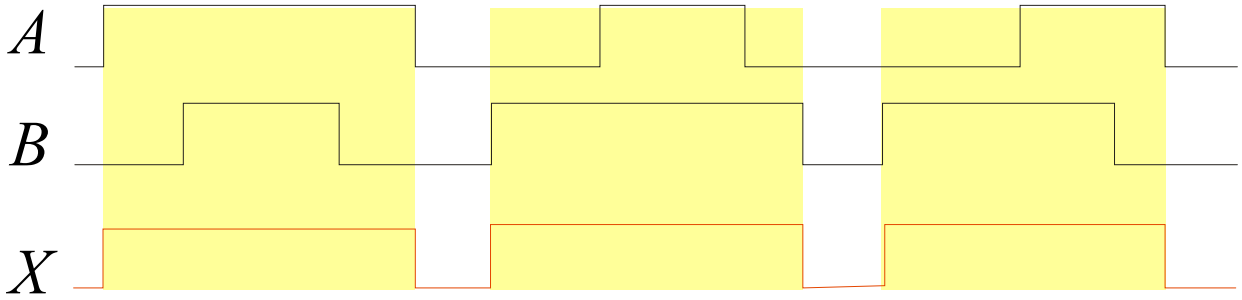
Inputs		Output
A	*B*	*X*
0	0	0
0	1	1
1	0	1
1	1	1

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

Summary

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.

Summary

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

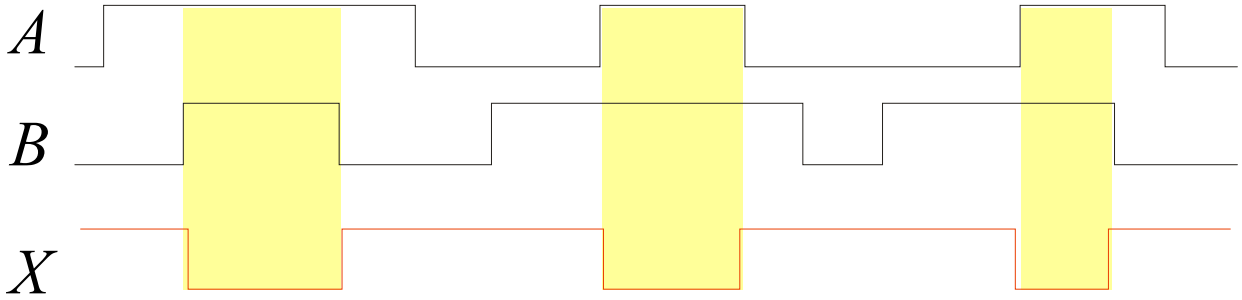
Inputs		Output
A	*B*	*X*
0	0	1
0	1	1
1	0	1
1	1	0

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, $X = \overline{AB}$.)

Summary

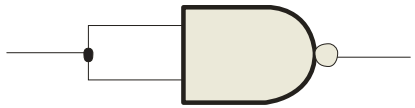
The NAND Gate

Example waveforms:



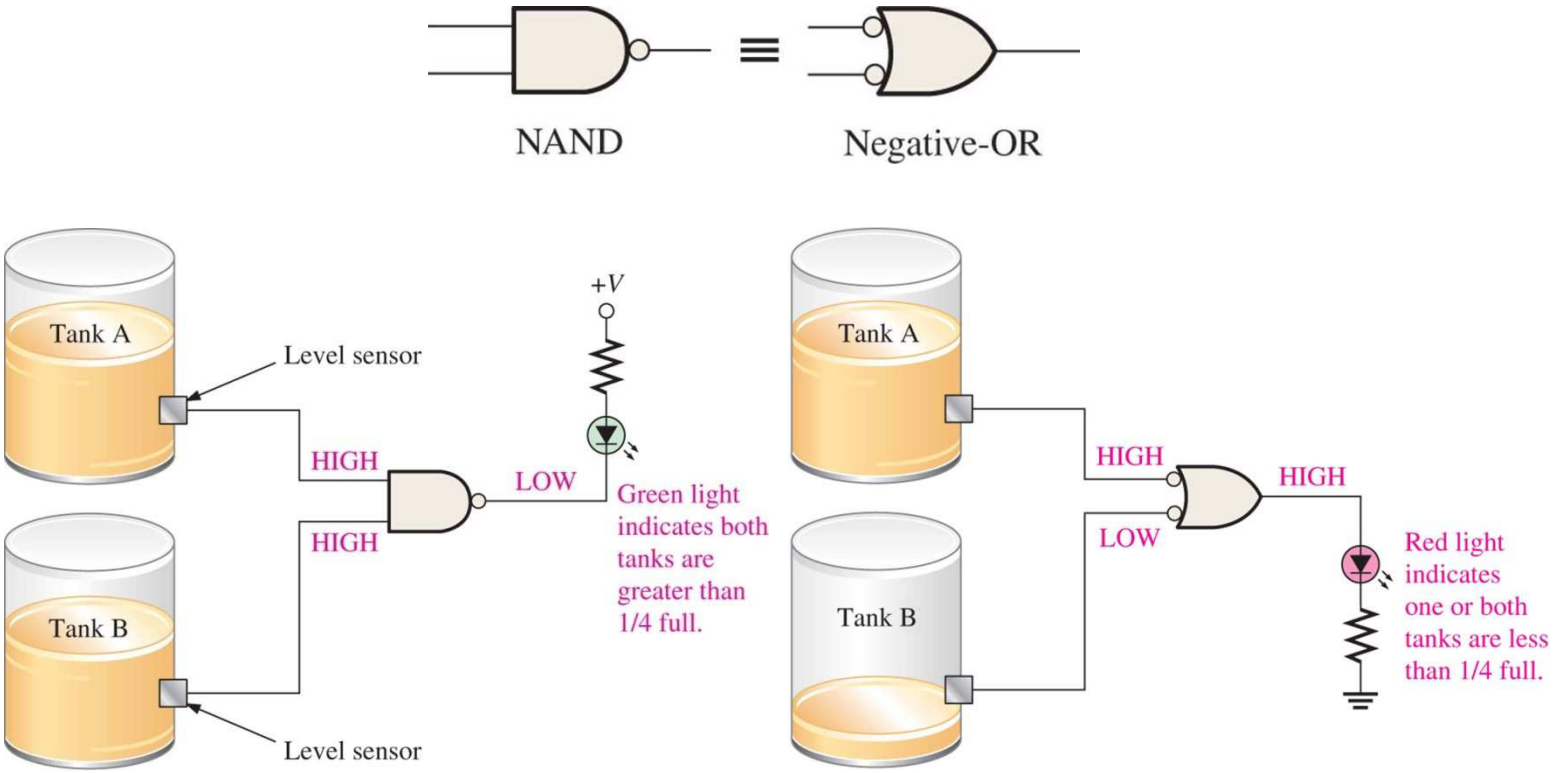
The NAND gate is particularly useful because it is a "universal" gate – all other basic gates can be constructed from NAND gates.

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



Summary

Negative-OR Equivalent



Summary

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

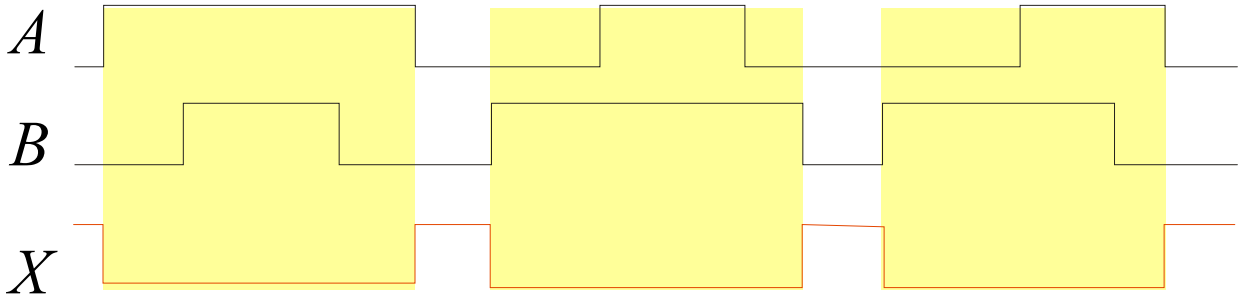
Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	0

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

Summary

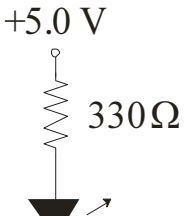
The NOR Gate

Example waveforms:

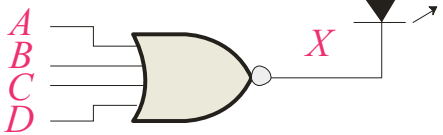


The NOR operation will produce a LOW if any input is HIGH.

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

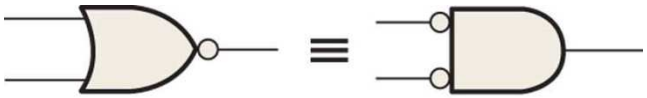


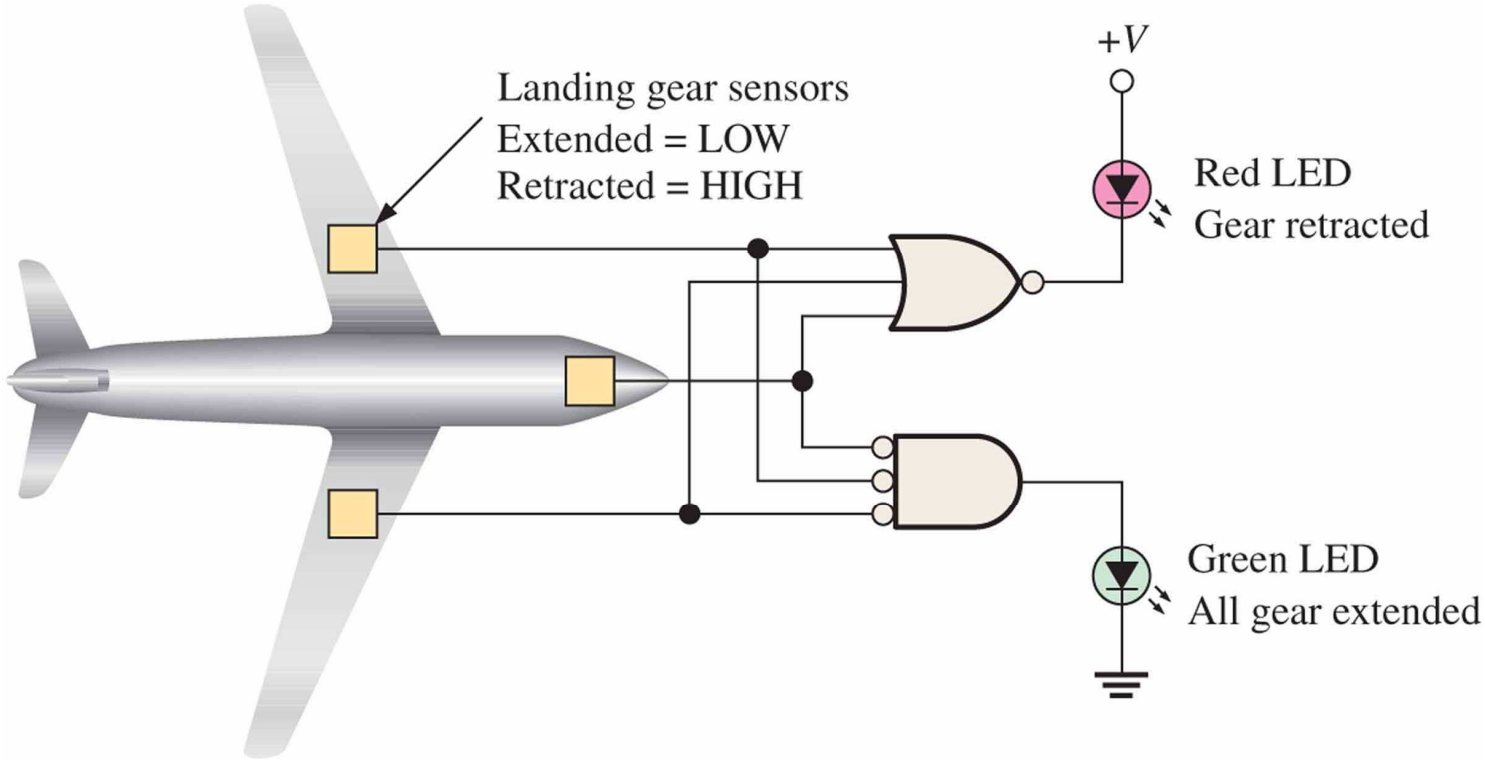
Solution The LED will be on when any of the four inputs are HIGH.



Summary

Negative-AND Equivalent





Summary

The XOR Gate

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

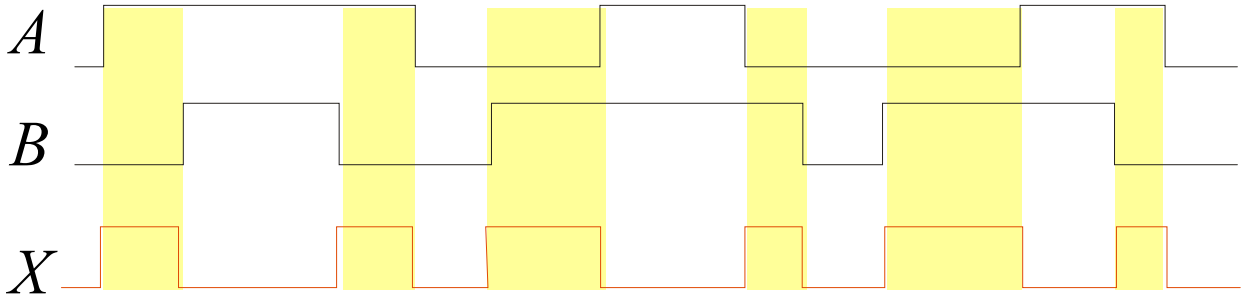
Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

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 $X = A \oplus B$.

Summary

The XOR Gate

Example waveforms:



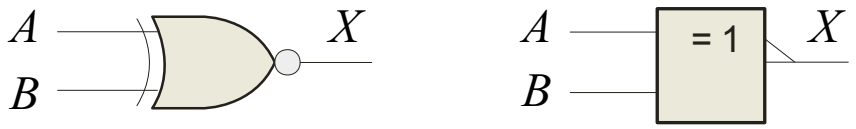
Notice that the XOR gate will produce a HIGH only when exactly one input is HIGH.

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.

Summary

The XNOR Gate



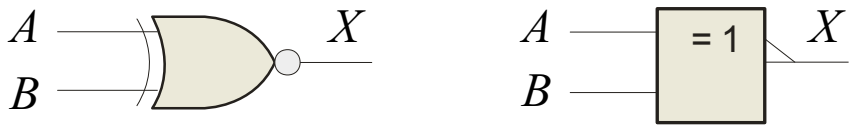
The **XNOR gate** produces a HIGH output only when both inputs are at the same logic level. The truth table is

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

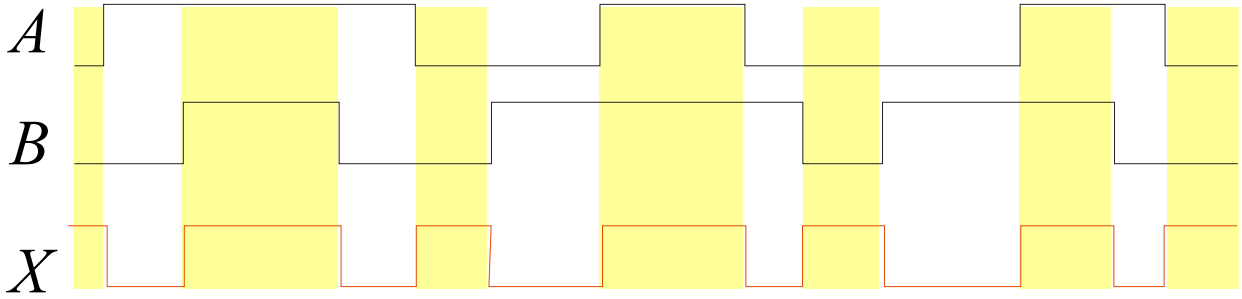
The **XNOR** operation shown as $X = \overline{A}\,\overline{B} + AB$. Alternatively, the XNOR operation can be shown with a circled dot between the variables. Thus, it can be shown as $X = A \odot B$.

Summary

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.

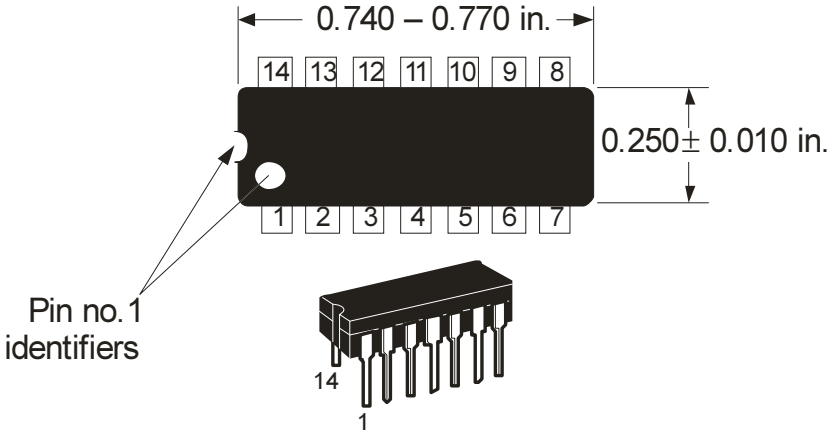
Question If the *A* waveform is inverted but *B* remains the same, how is the output affected?

The output will be inverted.

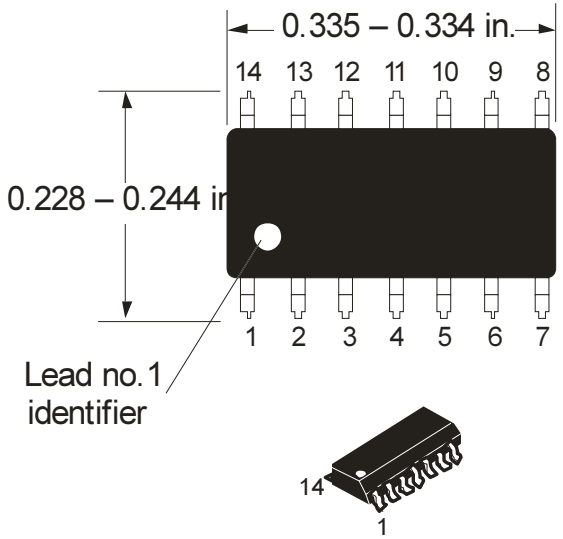
Summary

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

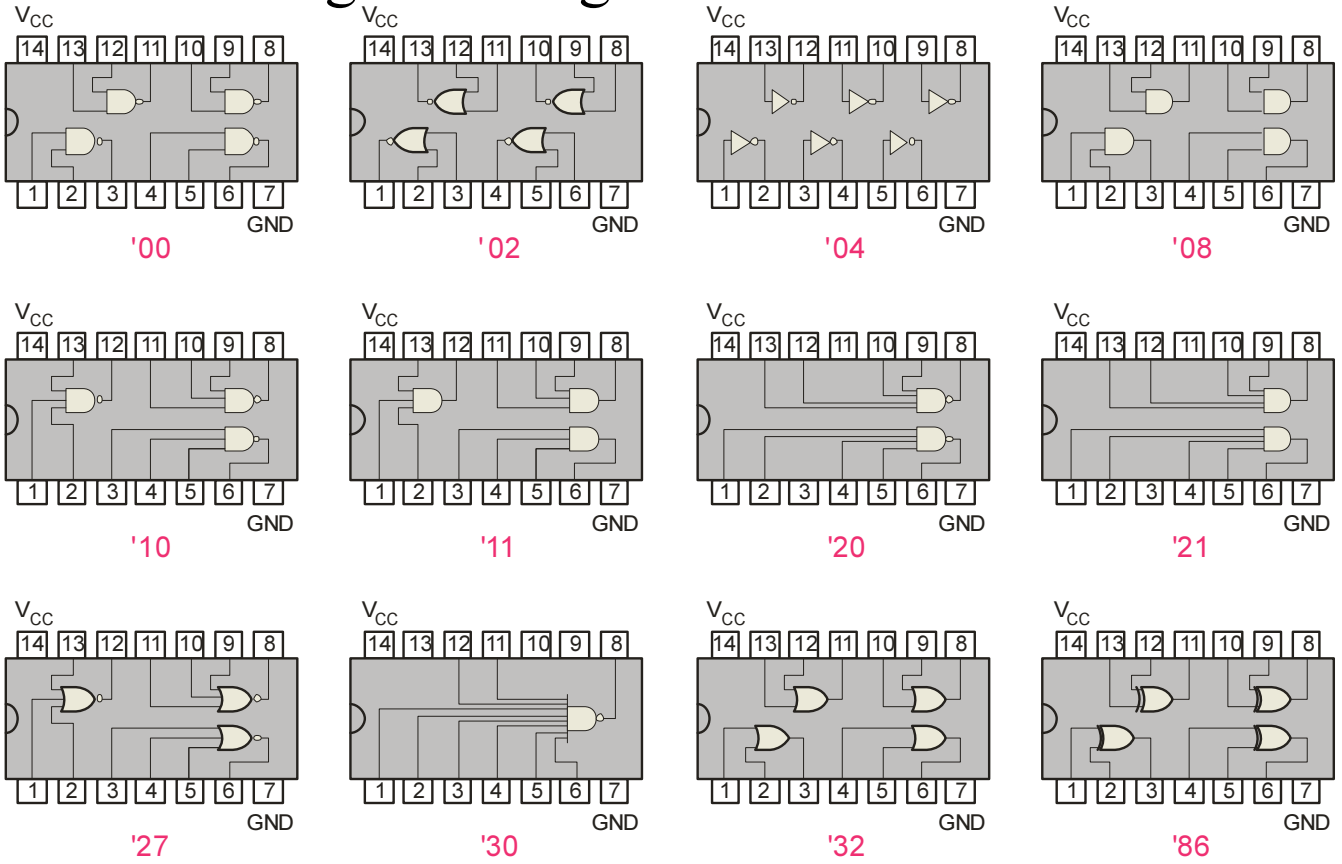


SOIC package

Summary

Fixed Function Logic

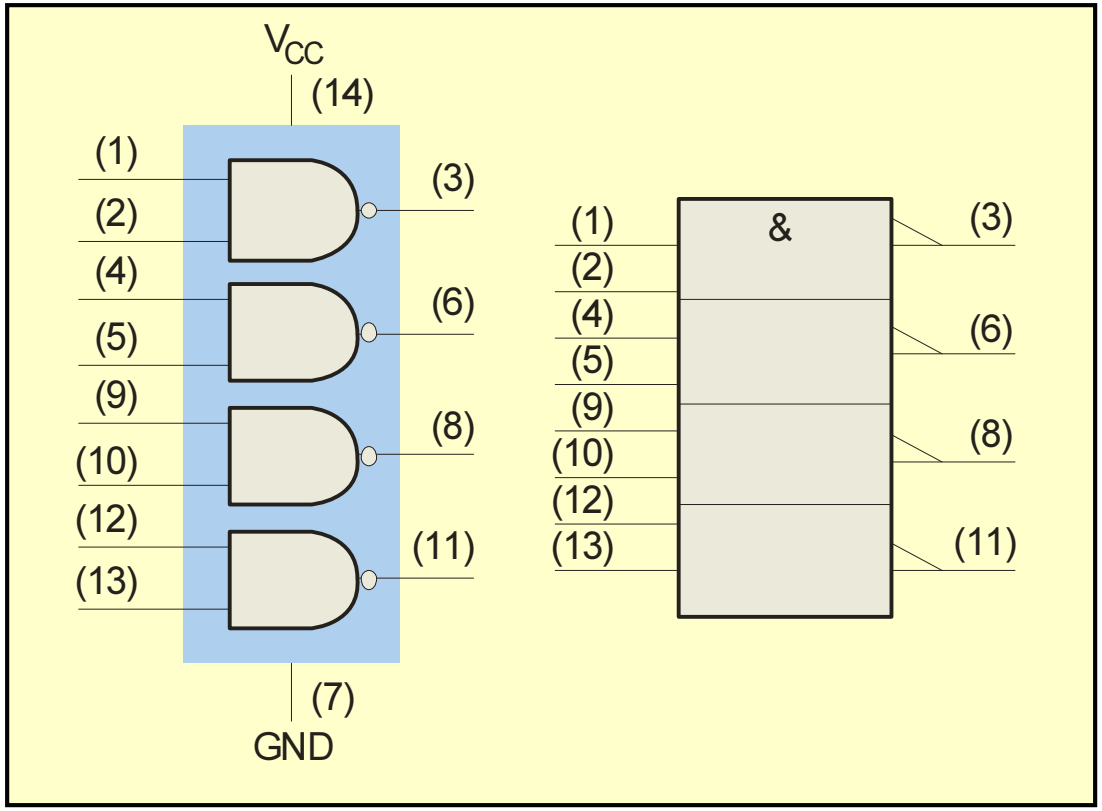
Some common gate configurations are shown.



Summary

Fixed Function Logic

Logic symbols show the gates and associated pin numbers.



Summary

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:

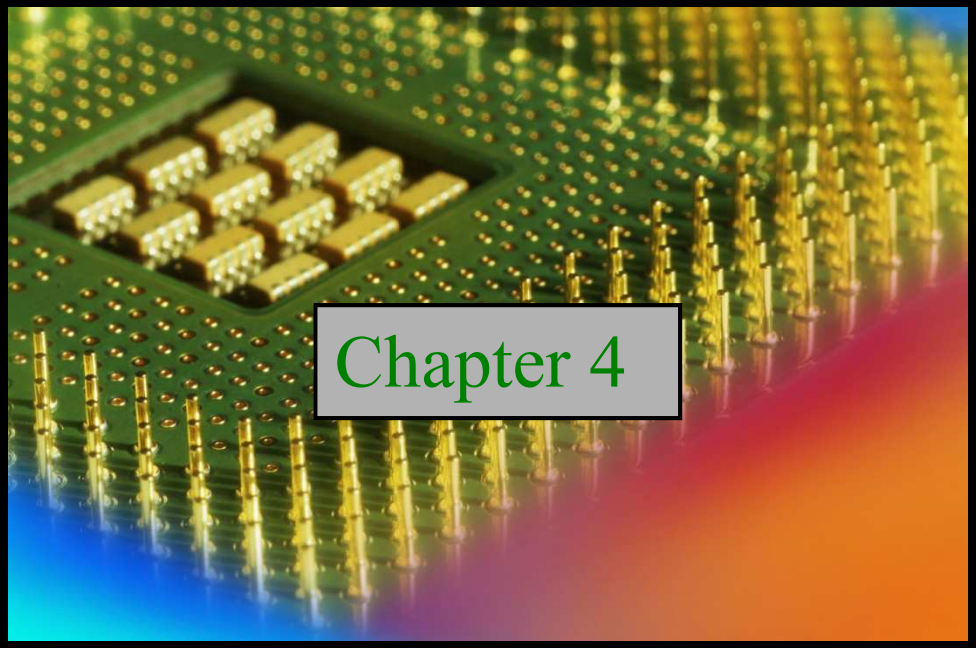
MAXIMUM RATINGS

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 StillAir, 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

Digital Fundamentals

Tenth Edition

Floyd



Summary

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 $C = 1$.

Summary

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 $A \cdot \overline{B} \cdot \overline{C} = 1$?

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

Summary

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.

$$AB = BA$$

Summary

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.

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

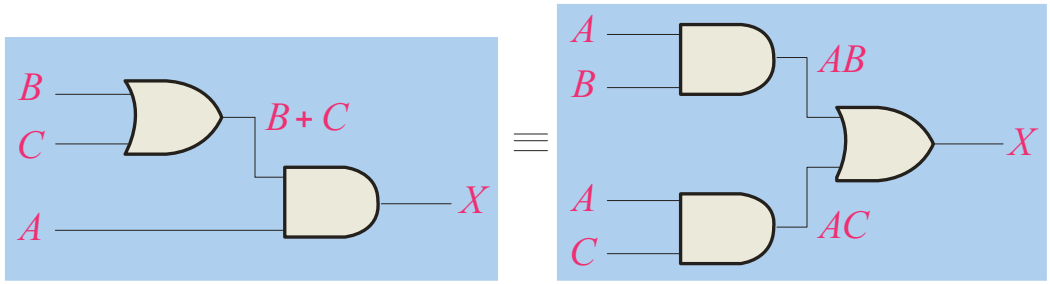
Summary

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:



$A(B + C)$ $\qquad$ $AB + AC$

Summary

Rules of Boolean Algebra

1. $A + 0 = A$
2. $A + 1 = 1$
3. $A \cdot 0 = 0$
4. $A \cdot 1 = 1$
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. $(A + B)(A + C) = A + BC$

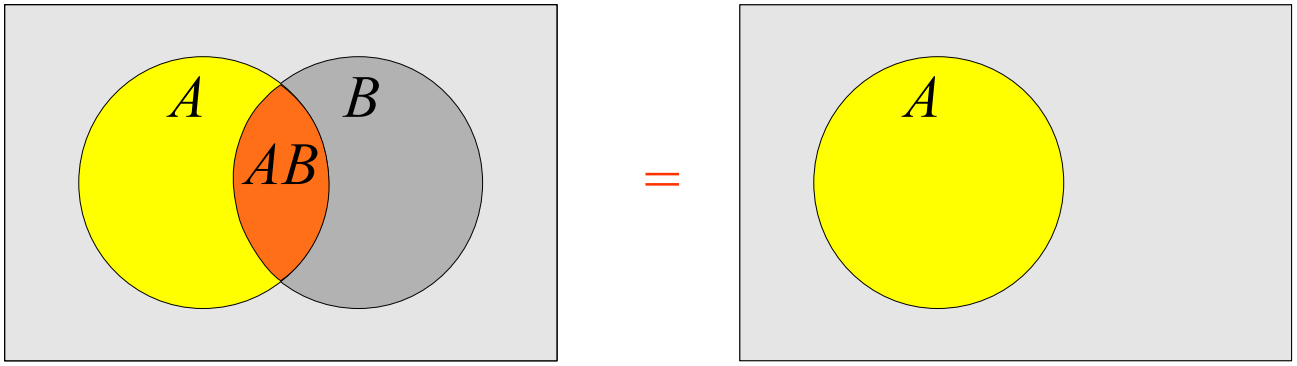
Summary

Rules of Boolean Algebra

Rules of Boolean algebra can be illustrated with *Venn* diagrams. The variable *A* is shown as an area.

The rule $A + AB = A$ can be illustrated easily with a diagram. Add an overlapping area to represent the variable *B*.

The overlap region between A and B represents *AB*.



The diagram visually shows that $A + AB = A$. Other rules can be illustrated with the diagrams as well.

Summary

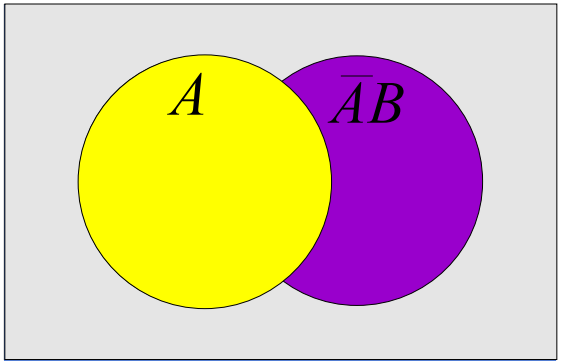
Rules of Boolean Algebra

Example

Illustrate the rule $A + \overline{A}B = A + B$ with a Venn diagram.

Solution

This time, $\overline{A}$ is represented by the blue area and B again by the red circle. The intersection represents $\overline{A}B$. Notice that $A + \overline{A}B = A + B$



Summary

Rules of Boolean Algebra

Rule 12, which states that $(A + B)(A + C) = A + BC$, can be proven by applying earlier rules as follows:

$$\begin{aligned}(A + B)(A + C) &= AA + AC + AB + BC \\ &= A + AC + AB + BC \\ &= A(1 + C + B) + BC \\ &= A \cdot 1 + BC \\ &= A + BC\end{aligned}$$

This rule is a little more complicated, but it can also be shown with a Venn diagram, as given on the following slide…

Summary

Three areas represent the variables *A*, *B*, and *C*.

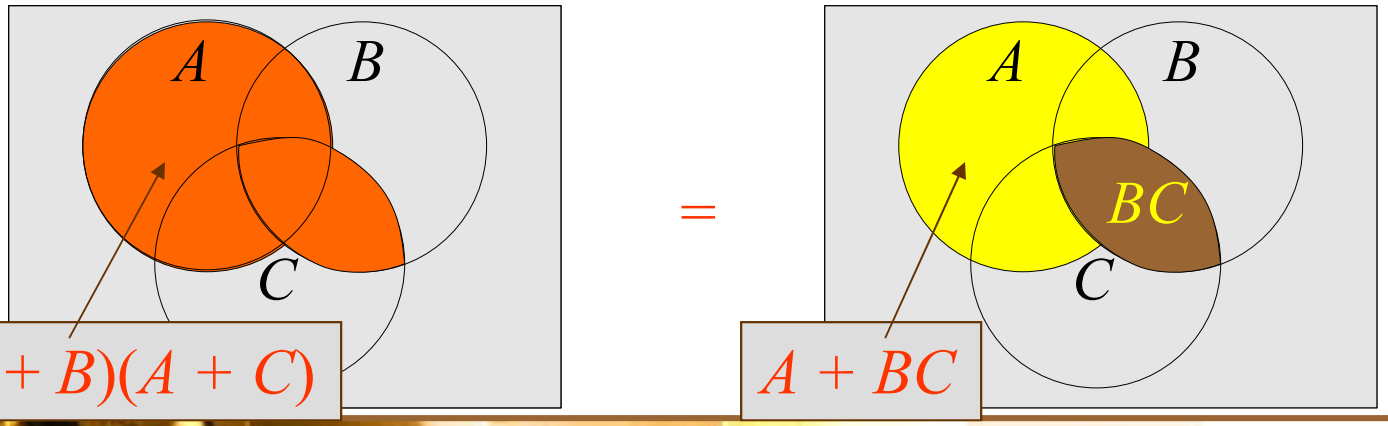
The area representing $A + B$ is shown in yellow.

The area representing $A + C$ is shown in red.

The overlap of red and yellow is shown in orange.

The overlapping area between *B* and C represents *BC*.

ORing with *A* gives the same area as before.



Summary

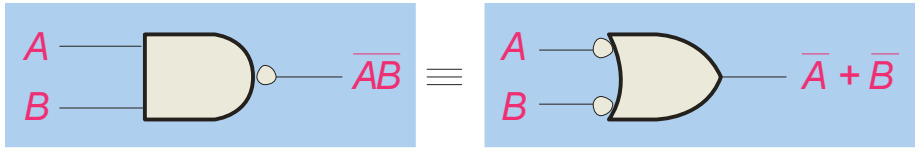
DeMorgan's Theorem

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

Summary

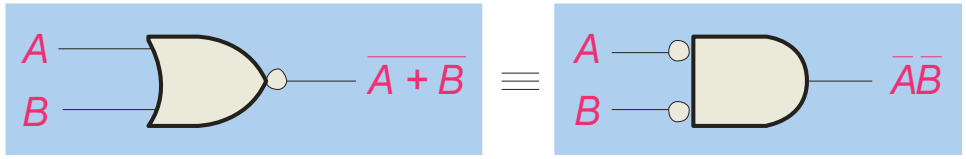
DeMorgan's Theorem

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

Summary

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

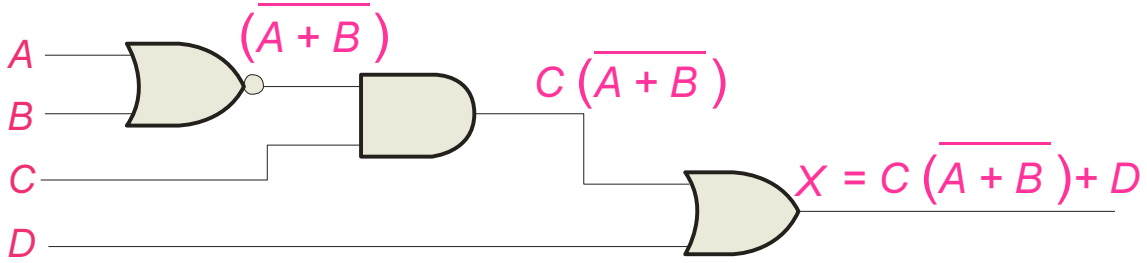
Summary

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:

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

Summary

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:

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

Summary

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:

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

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

Summary

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 (rule 8: $A \cdot \overline{A} = 0$) 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.

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

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

Summary

Converting Standard SOP to Standard POS

$X = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + AB\overline{C} + ABC$

INPUTS A	B	C	OUTPUT X	
0	0	0	1	$\overline{A}\overline{B}\overline{C}$
0	0	1	0	$\overline{A}\,\overline{B}\,C = \overline{A+B+\overline{C}}$
0	1	0	0	$\overline{A}\,B\,\overline{C} = \overline{A+\overline{B}+C}$
0	1	1	0	$\overline{A}\,B\,C = \overline{A+\overline{B}+\overline{C}}$
1	0	0	1	$A\overline{B}\overline{C}$
1	0	1	0	$A\,\overline{B}\,C = \overline{\overline{A}+B+\overline{C}}$
1	1	0	1	$AB\overline{C}$
1	1	1	1	ABC

SOP ≡ active-high logic

POS ≡ active-low logic

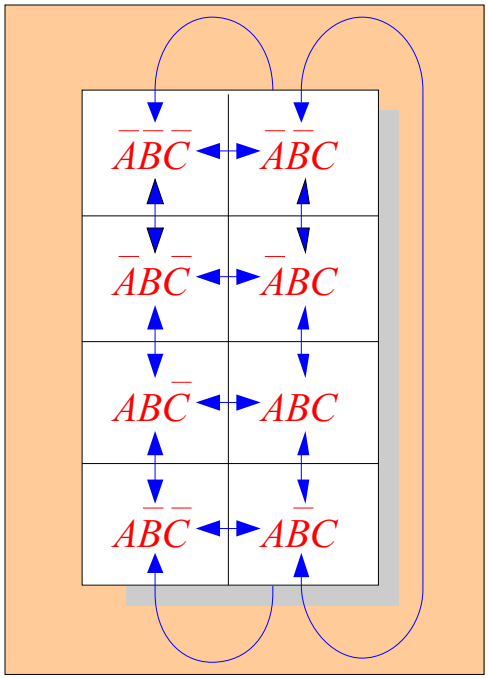
Summary

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.

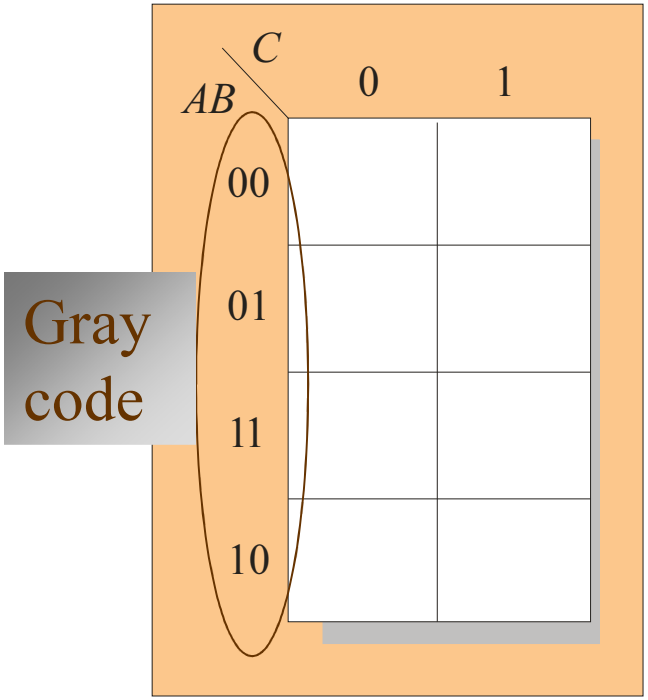
Each cell differs from an adjacent cell by only one variable.



Summary

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.

Summary

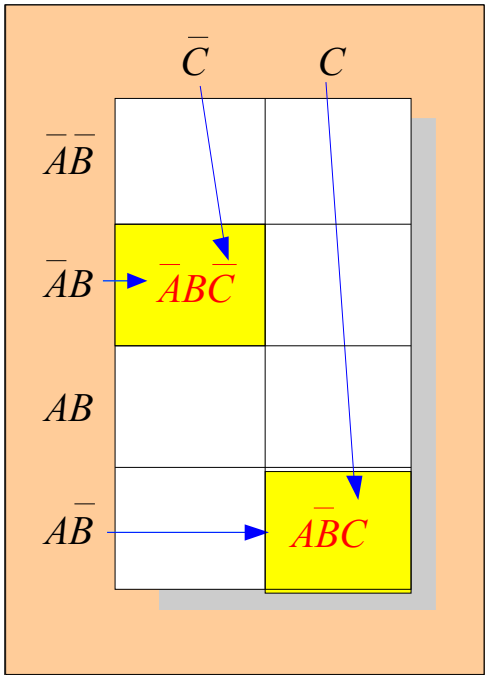
Karnaugh maps

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.

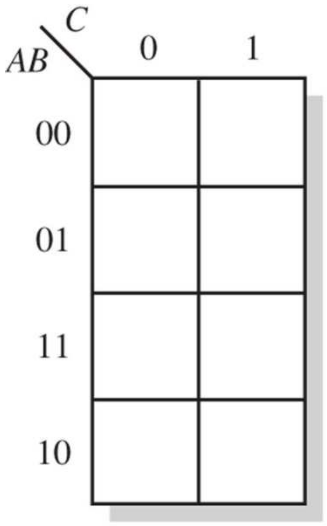
Solution

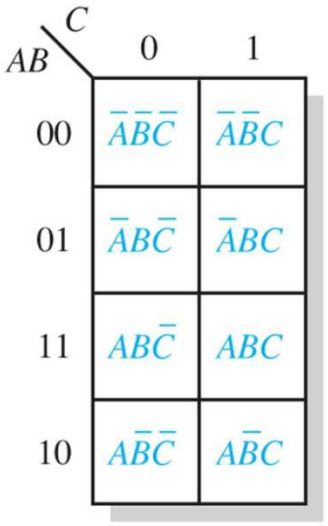
The cells are $\bar{A}B\bar{C}$ and $A\bar{B}C$.

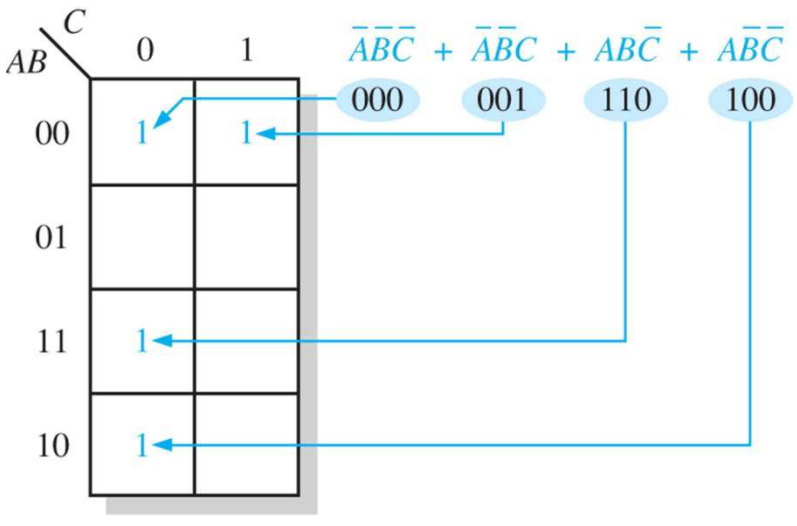


Summary

SOP Standard form in Karnaugh Map





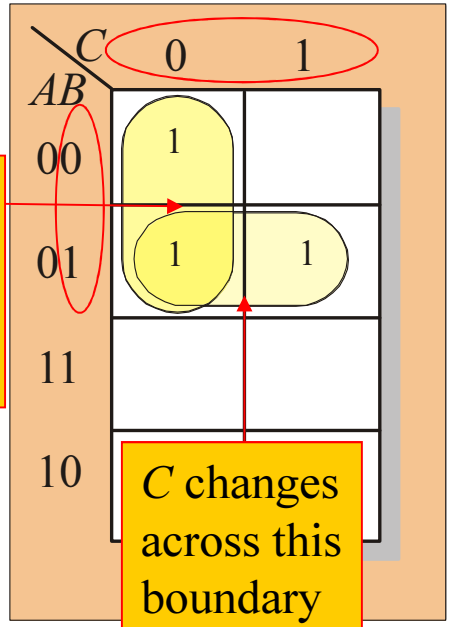


Summary

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

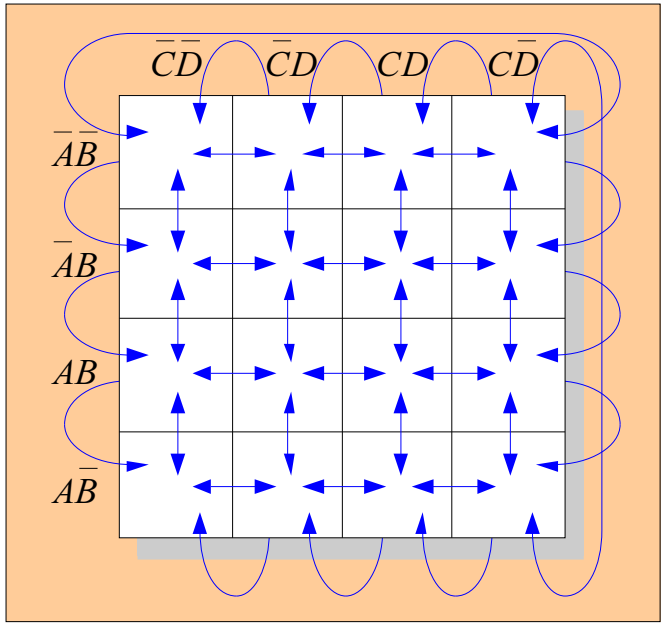
1. Group the 1's into two overlapping groups as indicated.
2. Read each group by eliminating any variable that changes across a boundary.
3. The vertical group is read $\overline{A}\,\overline{C}$.
4. The horizontal group is read $\overline{A}B$.

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

Summary

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.

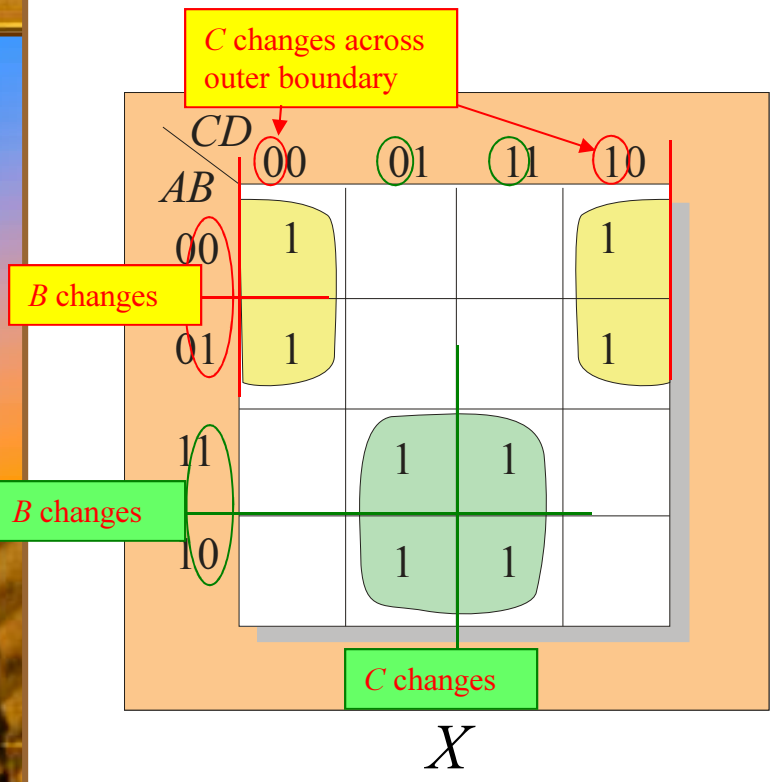
Grouping follows the rules given in the text.

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

Summary

Karnaugh maps

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



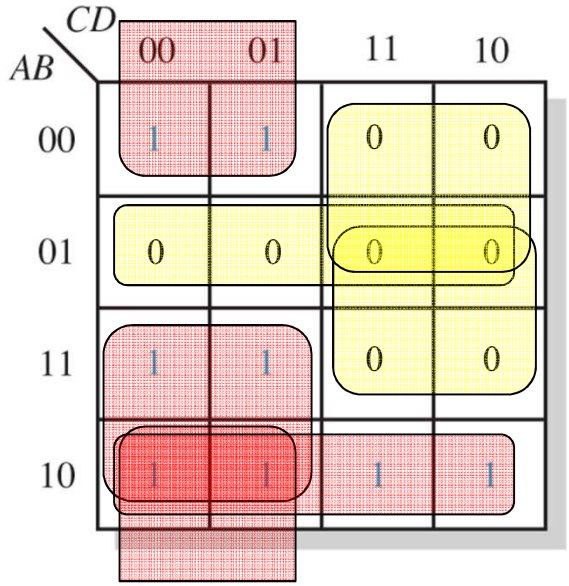
Solution

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 $\overline{A}\overline{D}$.
4. The lower (green) group is read as AD.

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

Summary

Converting Standard SOP to Standard POS with Karnaugh



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

$\overline{SOP} = \overline{A}C\cdot + \overline{A}B + BC$

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

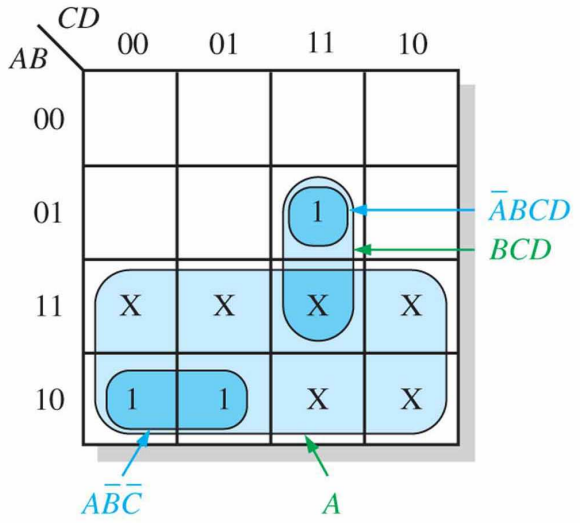
Summary

"Don't Care" Conditions

INPUTS				OUTPUT
A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

(a) Truth table

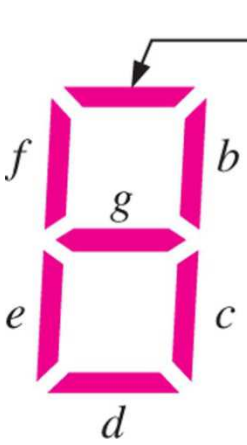
Don't cares



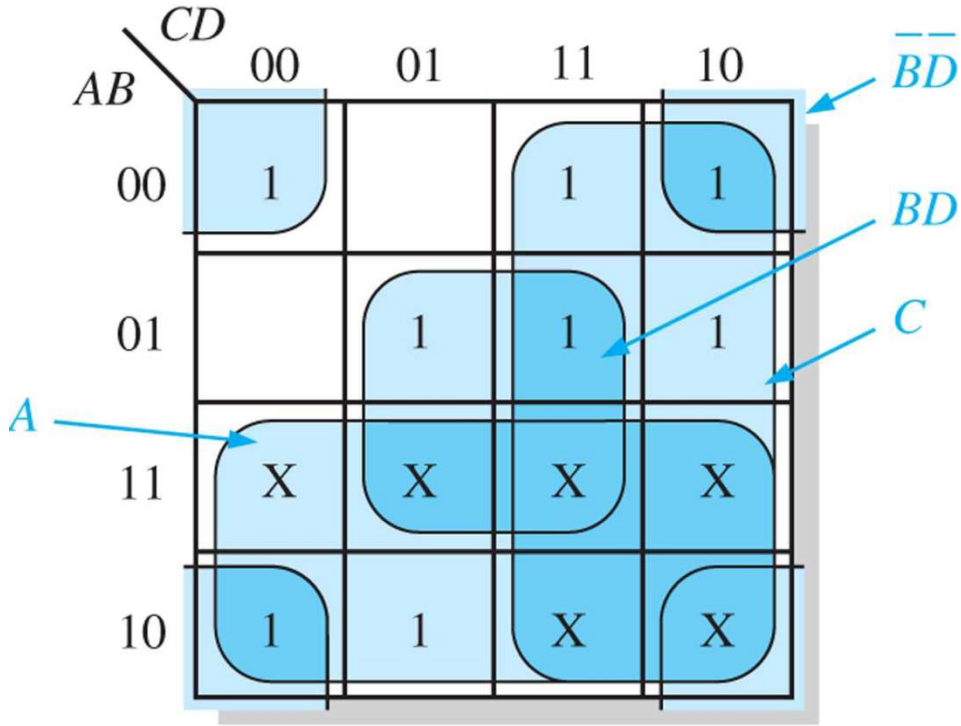
(b) Without "don't cares" $Y = A\bar{B}\bar{C} + \bar{A}BCD$
With "don't cares" $Y = A + BCD$

Summary

"Don't Care" Conditions

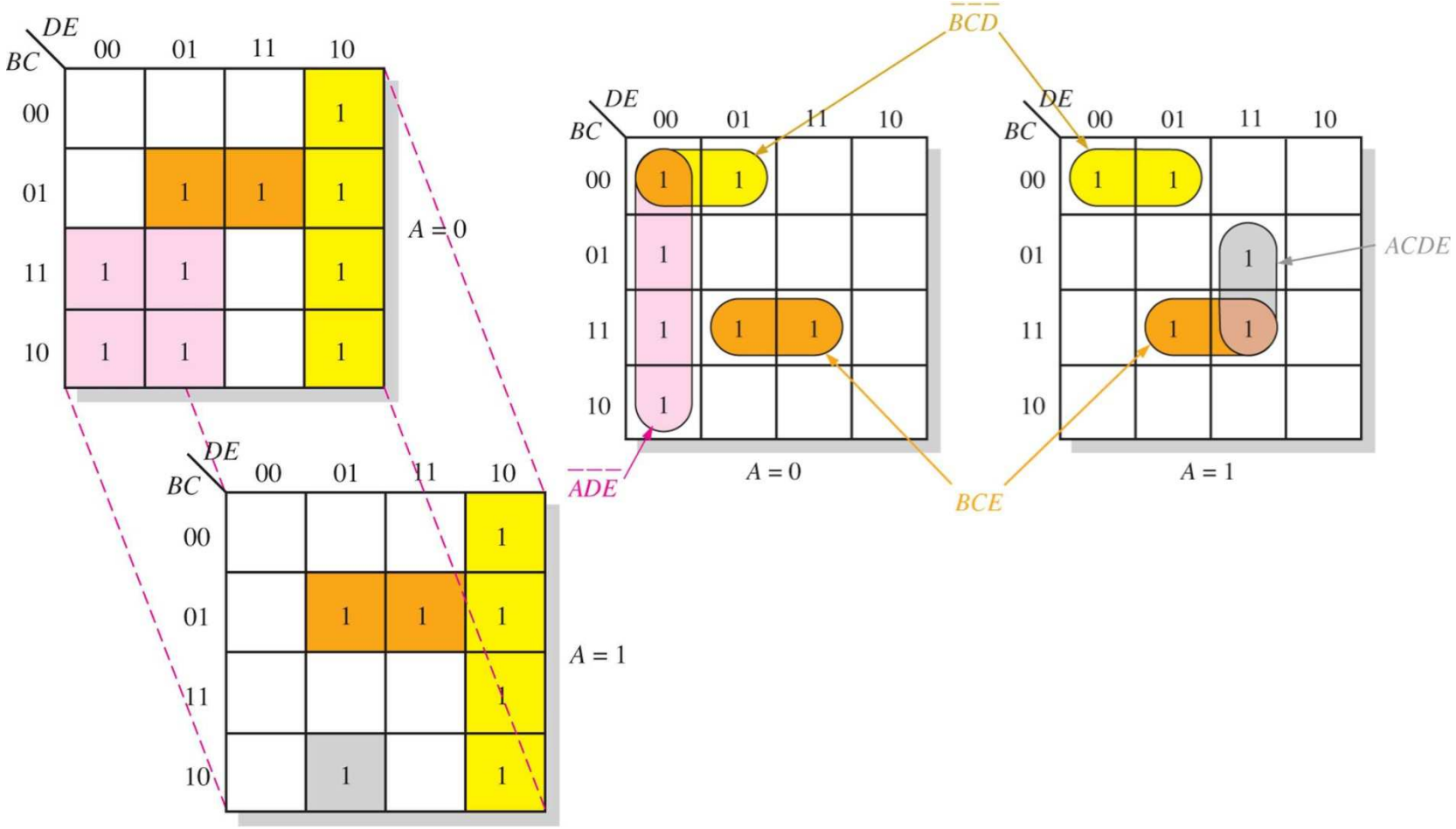


0 2 3 5 6 7 8 9



Summary

Five-Variable Karnaugh Maps



Homework 7

- Chapter 3 (2, 12, 18, 45)
- Chapter 4 (11, 13, 21, 36 a, 44)