03: Boolean Algebra and Circuit Design

IT1206 – Computer Systems

Level I - Semester 1

Boolean Algebra

Boolean Postulates

$0 . 0 = 0$

$1 + 1 = 1$

$0 + 0 = 0$

$1 . 1 = 1$

$1 . 0 = 0 . 1 = 0$

$1 + 0 = 0 + 1 = 1$

Basic Identities of Boolean Algebra

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0+x = x$
Null (or Dominance) Law	$0x = 0$	$1+x = 1$
Idempotent Law	$xx = x$	$x+x = x$
Inverse Law	$x\bar{x} = 0$	$x+\bar{x} = 1$
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$
Absorption Law	$x(x+y) = x$	$x+xy = x$
DeMorgan's Law	$(\overline{xy}) = \bar{x}+\bar{y}$	$(\overline{x+y}) = \bar{x}\bar{y}$
Double Complement Law	$\bar{\bar{x}} = x$	

Source - Linda Null and Julia Lobur, Essentials of Computer Organization and Architecture, 5th Edition

UCSC

BIT

Laws of Boolean Algebra

- $$A.B + A.\overline{B} = A$$
  $$(A + B)(A + \overline{B}) = A$$

- $$A + 0 = A$$
  $$A.0 = 0$$

Laws of Boolean Algebra

- $$A + \overline{A}.B = A + B$$

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

Boolean Expressions

- There can be multiple boolean expressions for a single operational behavior (single truth table).
- Thus, there can be multiple circuit structures with different logic gate arrangements for the same behavior.
- Simple and minimal circuit structure is always preferred.
  - Efficiency
  - Cost

Simple Expression

- Boolean algebra helps to simplify expressions.
  - $F1 = \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}$
  - $F2 = x\bar{y} + \bar{x}z$

- You will get two circuits if you draw circuits for F1 and F2.

x	y	z	$F2$	$F1$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Expression to Truth Table

- When a boolean expression is given, we can formulate the truth table for the expresssion.
- Identify the variables in the expression
  - If there are n variables, then there will be 2^n combinations of values
  - 2^n rows in the turth table
- Derive the resulting value for each value combination in the truth table rows.

Expression to Truth Table. (Cont.)

- Ex.

$$F(x,y,z) = x.y.\bar{z} + \bar{y}.z$$

  - Three variables in $F(x,y,z)$
  - There will be 2^3 combinations
  - There will be 2^3 rows in the truth table

Expression to Truth Table. (Cont.)

- Ex.

$$F(x,y,z) = x.y.\bar{z} + \bar{y}.z$$

x	*y*	*z*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Truth Table to Expression

- When a truth table is given, we can formulate the boolean algebric expression to represent it.
- There are two forms ($Canonical$ forms) in presenting the expression.
  - Sum of *Minterms* (Standard Products)
  - Product of *Maxterms* (Standard Sums)

Minterm

- In a three variable boolean expression, we can combine 3 variables with an AND operator.
- There are 8 possible such arrangements.
  - $\bar{x}.\bar{y}.\bar{z}$
  - $\bar{x}.\bar{y}.z$
  - ………..
  - $x.y.z$
- Each of these AND terms is called ***Minterm***

  (or Standard Product).
- $(x.y)$ is **not** a ***Minterm*** in this context.

Maxterm

- In a three variable boolean expression, we can combine 3 variables with an OR operator.
- There are 8 possible such arrangements.
  - $\bar{x} + \bar{y} + \bar{z}$
  - $\bar{x} + \bar{y} + z$
  - ……………
  - $x + y + z$
- Each of these OR terms is called ***Maxterm*** (or Standard Sum)
- $(\bar{y} + z)$ is not a ***Maxterm*** in this context.

[Sum – Product] in Mathematics

A **product** is the result of multiplying, or an expression that identifies factors to be multiplied.

A **sum** is the aggregate of two or more numbers or particulars as determined by or as if by the mathematical process of addition.

Sum of Minterms

- Disjunction of terms where each term is a conjunction of literals.
- The OR operations are performed on the terms that are made by AND operations.
- Ex.

$$F = \bar{x}.y.\bar{z} + x.\bar{y}.\bar{z} + x.\bar{y}.z + x.y.\bar{z} + x.y.z$$

- Each term in the expression is referred as ***Minterm*** or ***Standard Product***.

Product of Maxterms

- Conjunction of terms where each term is a disjunction of literals.
- The AND operations are performed on the terms that are made by OR operations.
- Ex.

$$F = (x + y + z).(x + y + \bar{z}).(x + \bar{y} + \bar{z})$$

- Each term in the expresssion is referred as ***Maxterm*** or ***Standard Sum***.

Standard Forms

- In ***Cannonical*** Form, each term must contain all the variables in the truth table.
- But ***Standard Form*** is an alternative way to express Boolean function.
- A term in an expression that is expressed in a standard form doesn't need to contain all the variables in the truth table.
- There are two standard forms.
  - Sum of Products
  - Product of Sums

Standard Form – Sum of Products

- OR operation is performed over several terms where each term is made up by performing AND operation over several literals.
- Ex.

$$F = \bar{x}.y + \bar{y}.\bar{z} + x.\bar{y}.z + y$$

✓

$$F = \bar{x}.y + \bar{y}.(x + \bar{z}) + x.\bar{y}.z + y$$

✗

Standard Form - Product of Sums

- AND operation is performed over several terms where each term is made up by performing OR operation over several literals.
- Ex.

$$F = y.(x + z).(\bar{x} + \bar{y} + \bar{z})$$

✓

$$F = y.(x + z.\bar{y}).(\bar{x} + \bar{y} + \bar{z})$$

✗

Logic Equations to Truth Tables

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

A	B	X
0	0	
0	1	
1	0	
1	1	

Sum of Minterms

- The **OR** operation performed on the products of the **AND** operation
- Fill the corresponding cells with **1** for each product, the other cells with **0**

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

$$A = 1, \overline{A} = 0$$

$$B = 1, \overline{B} = 0$$

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

Product of Maxterms

- The **AND** operation performed on the sums of the **OR** operation
- Fill the corresponding cells with **0** for each sum, the other cells with **1**

$$Y = (\overline{A} + \overline{B}).(\overline{A} + B)$$

$$A = 0, \overline{A} = 1$$

$$B = 0, \overline{B} = 1$$

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

Truth Tables to Logic Equations

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

- Sum of Minterms - consider 1s
  - Consider A=1,B=1

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

- Product of Maxterms – consider 0s
  - Consider A=0,B=0

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

Your Turn: Exercise1

- Convert the following equation which is in the form of Product-of-sums into the form of Sum-of-products

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

Answer: Exercise1

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

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

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

$$f(ABCD) = \overline{(\overline{A}.\overline{\overline{B}}.\overline{C})+(\overline{\overline{A}}.\overline{B}.\overline{\overline{C}}.\overline{\overline{D}})+(\overline{\overline{A}}.\overline{\overline{B}}.\overline{D})+(\overline{A}.\overline{\overline{C}})}$$

$$f(ABCD) = \overline{(\overline{A}.B.\overline{C})+(A.\overline{B}.C.D)+(A.B.\overline{D})+(\overline{A}.C)}$$

Implementation of Boolean Functions

- A Boolean function can be realised in either **SOP** or **POS** form
- At this point, it would seem that the choice would depend on whether the truth table contains more **1**s and **0**s for the output function
- The **SOP** has one term for each **1**, and the **POS** has one term for each **0**

Implementation of Boolean Functions

- However, there are other considerations:
  - It is generally possible to derive a simpler Boolean expression from truth table than either **SOP** or **POS**
  - It may be preferable to implement the function with a single gate type (NAND or NOR)

Implementation of Boolean Functions

- The significance of this is that, with a simpler Boolean expression, fewer gates will be needed to implement the function

- Methods that can be used to achieve simplification are:
  - Algebraic Simplification
  - Karnaugh Maps

Your Turn: Algebraic Simplification

- Simplify the following equation using Boolean algebra laws

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

Answer: Algebraic Simplification

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

$$f(ABC) = AA + A\overline{B}C + A\overline{B} + \overline{B}\,\overline{B}C + A\overline{C} + \overline{B}C\overline{C}$$

$$f(ABC) = A(1 + \overline{B}C + \overline{B} + \overline{C}) + \overline{B}C + \overline{B}C\overline{C}$$

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

Karnaugh Maps

- For purposes of simplification, the Karnaugh map is a convenient way of representing a Boolean function of a small number (up to 4 to 6) of variables

- The map is an array of 2^n squares, representing the possible combinations of values of **n** binary variables

Karnaugh Maps

- The map can be used to represent any Boolean function in the following way:
  - Each square corresponds to a unique product in the **sum-of-products** form.
  - With a **1** value corresponding to the variable and a **0** value corresponding to the NOT of that variable

Karnaugh Maps: 2 Values

B \ A	0	1
0	0	1
1	1	1

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

Karnaugh Maps: 2 Values

- The $A\bar{B}$ corresponds to the fourth square in the Figure
- For each such production in the function, 1 is placed in the corresponding square

$$\overbrace{\begin{array}{|c|c|c|c|} \multicolumn{1}{c}{00} & \multicolumn{1}{c}{01} & \multicolumn{1}{c}{11} & \multicolumn{1}{c}{10} \\ \hline & 1 & & 1 \\ \hline \end{array}}^{AB}$$

$$F = A\bar{B} + \bar{A}B$$

Karnaugh Maps: 3 Values

C \ AB	00	01	11	10
0	1	1	0	0
1	0	0	1	1

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

Karnaugh Maps: 4 Values

CD \ AB	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	0	1	0	1

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

Karnaugh Maps: Exercise 1

- Simplify the following Karnaugh Map using Boolean equations (Write your answers in both **SOP** and **POS**)

C \ AB	00	01	11	10
0	0	1	0	0
1	1	1	0	1

Karnaugh Maps: Answer

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

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

C \ AB	00	01	11	10
0	0	1	0	0
1	1	1	0	1

Karnaugh Maps: Exercise 2

- Simplify the following Karnaugh Map using Boolean equations (Write your answers in both **SOP** and **POS**)

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	0	1	0
10	1	0	0	1

Karnaugh Maps: Answer

$$(\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}) + (A\overline{B}\,\overline{C}\,\overline{D}) + (\overline{A}B\overline{C}D) + (AB\overline{C}D) + (ABCD) + (\overline{A}\,\overline{B}C\overline{D}) + (A\overline{B}C\overline{D})$$

$$\Updownarrow$$

$$\overline{B}\,\overline{D} + B\overline{C}D + ABD$$

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	0	1	0
10	1	0	0	1

Karnaugh Maps: Answer

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

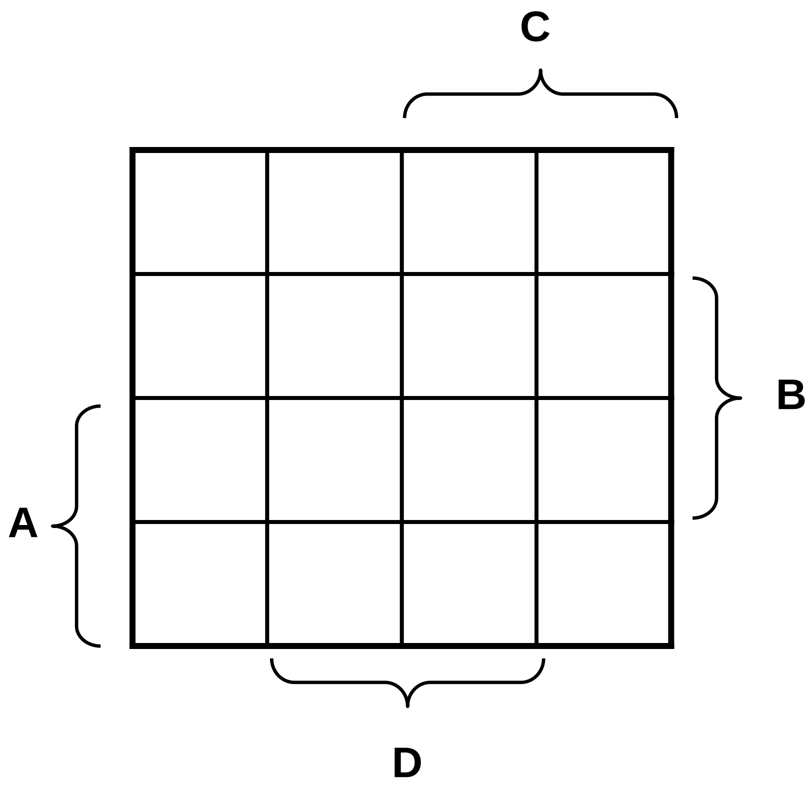
$$\Updownarrow$$

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

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	0	1	0
10	1	0	0	1

Simplified Labeling of Karnaugh Maps

- The labeling used in figure emphasizes the relationship between variables and the rows and columns of the map
  - The two rows embraced by the symbol **A** are those in which the variable **A** has the value **1**; the rows not embraced by the symbol **A** are those in which **A** is **0**



Simplified Labeling of Karnaugh Maps

- Once the map of a function is created, we can often write a simple algebraic expression for it by noting the arrangement of the **1**s on the map
- The principle is as follows:
  - Any two squares that are adjacent differ in only one of the variables
  - If two adjacent squares both have an entry of **1**, then the corresponding product terms differ in **only one variable**
  - In such a case, the two terms can be merged by **eliminating that variable**

UCSC

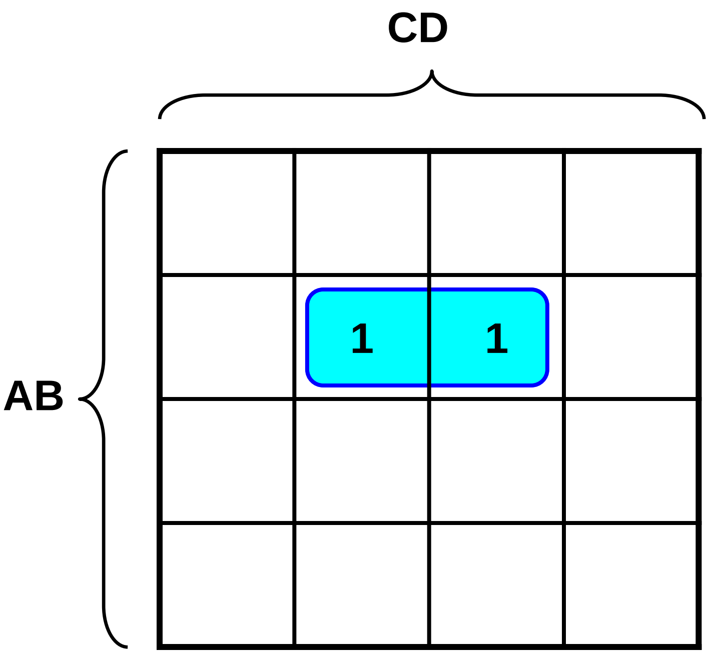
BIT

Simplified Labeling of Karnaugh Maps

- For example, in following FIGURE, the two adjacent squares correspond to the two terms $\bar{A}B\bar{C}D$ and $\bar{A}BCD$

- The function expressed is

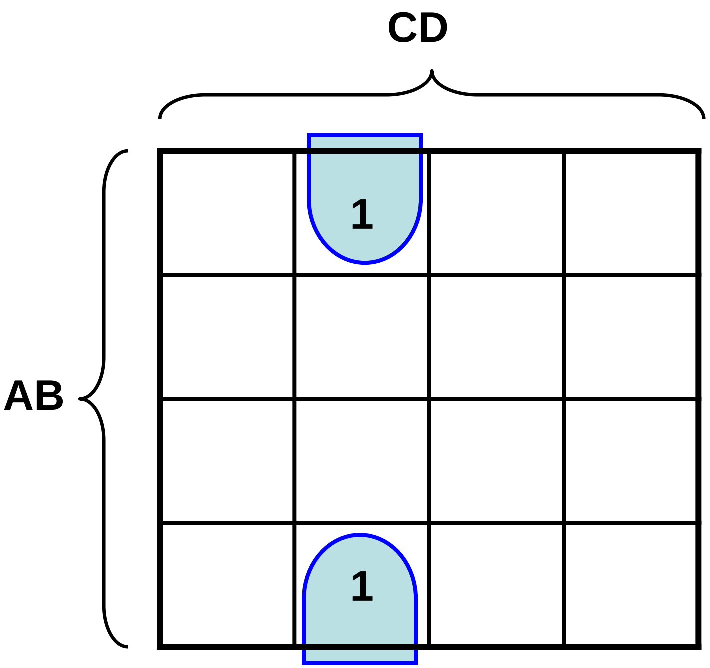
  $\bar{A}B\bar{C}D + \bar{A}BCD = \bar{A}BD$

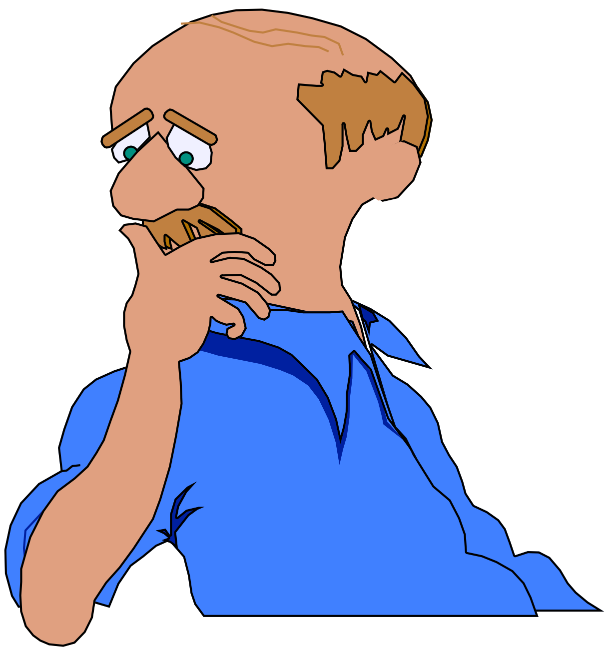


Simplified Labeling of Karnaugh Maps

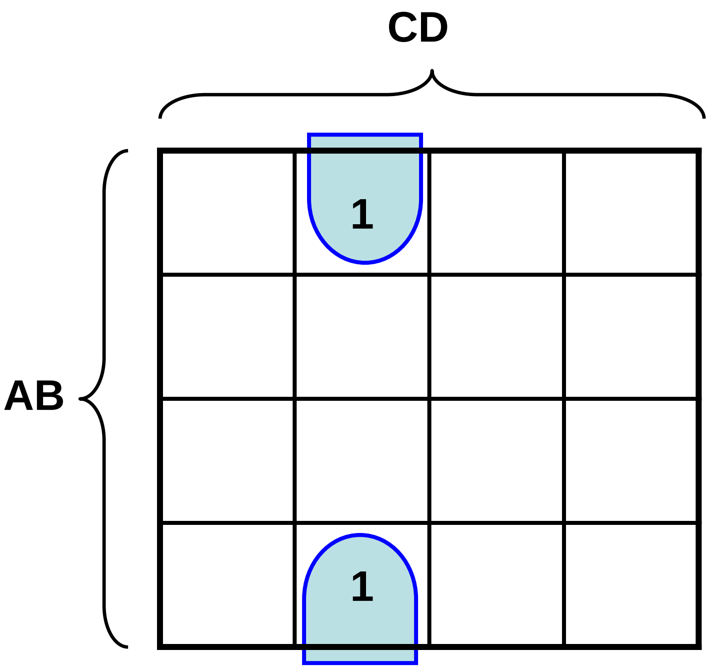
- This process can be extended in several ways:
  - First, the concept of adjacent can be extended to include wrapping around the edge of the map
  - Thus, the **top square of a column** is adjacent to the **bottom square**, and the **leftmost square of a row** is adjacent to the **rightmost square**
  - Second, we can group not just 2 squares but 2^n adjacent squares, that is, 4, 8, etc

Your turn: Karnaugh Maps



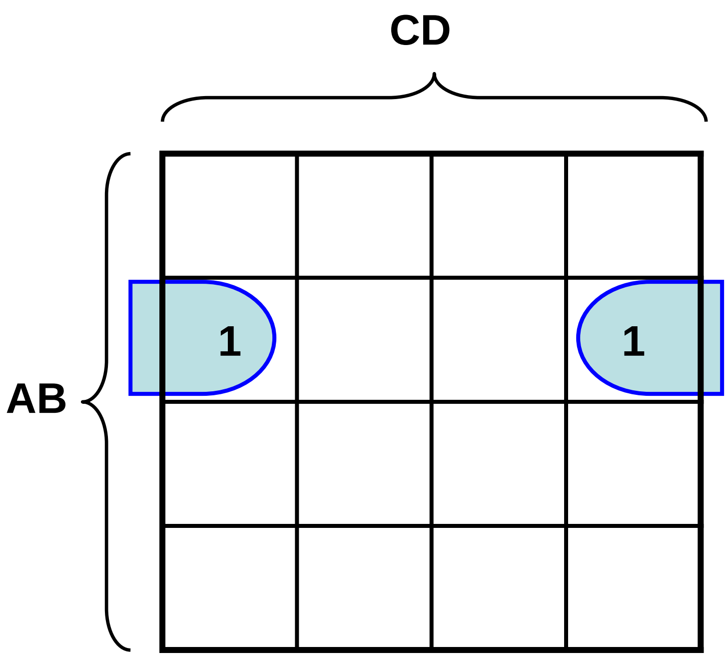


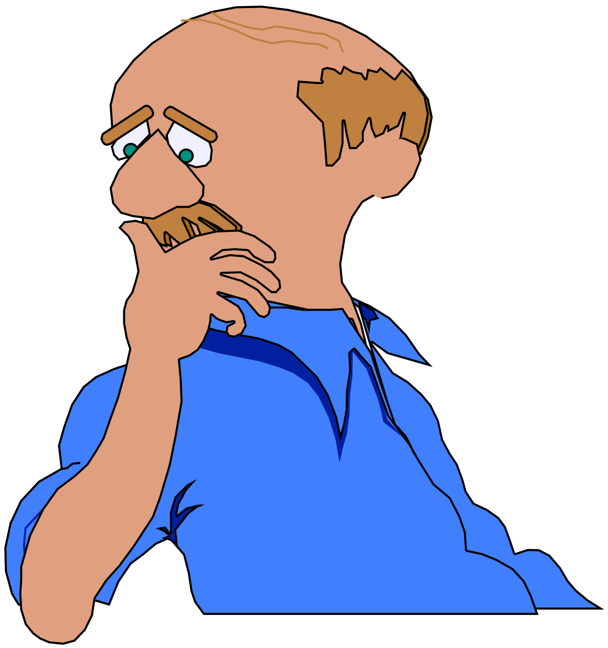
Answer: Karnaugh Maps



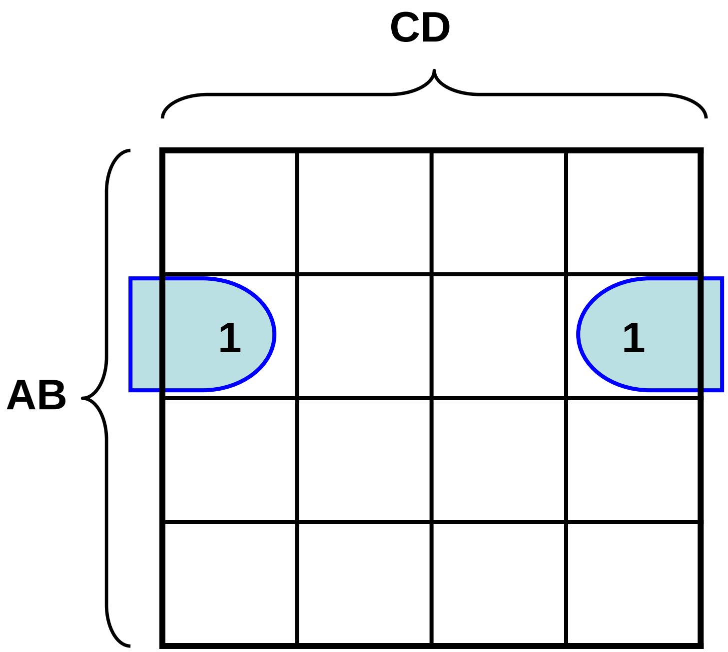
$\overline{B}\,\overline{C}D$

Your turn: Karnaugh Maps





Answer: Karnaugh Maps



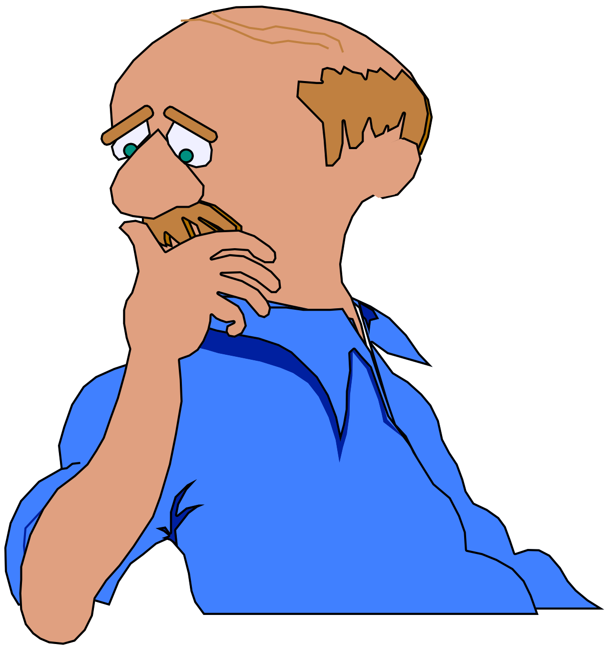
$\overline{A}B\overline{D}$

Your turn: Karnaugh Maps

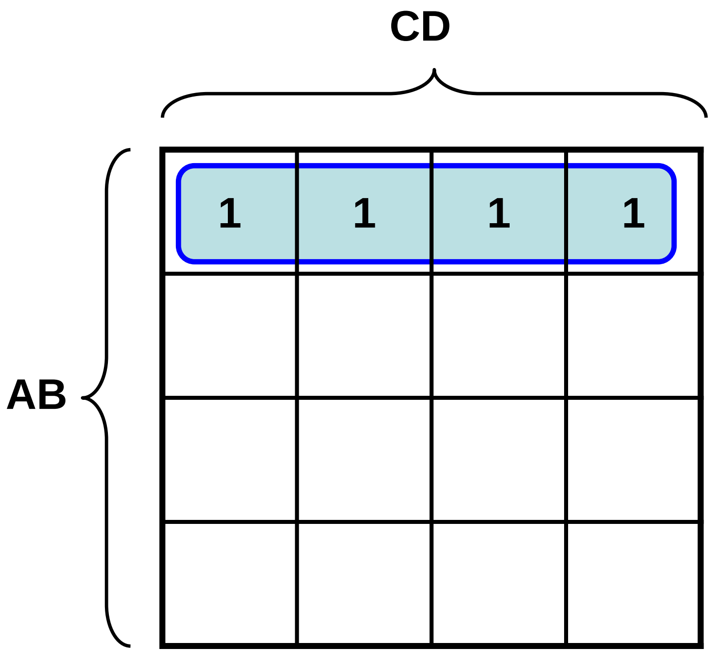
CD

AB

1	1	1	1

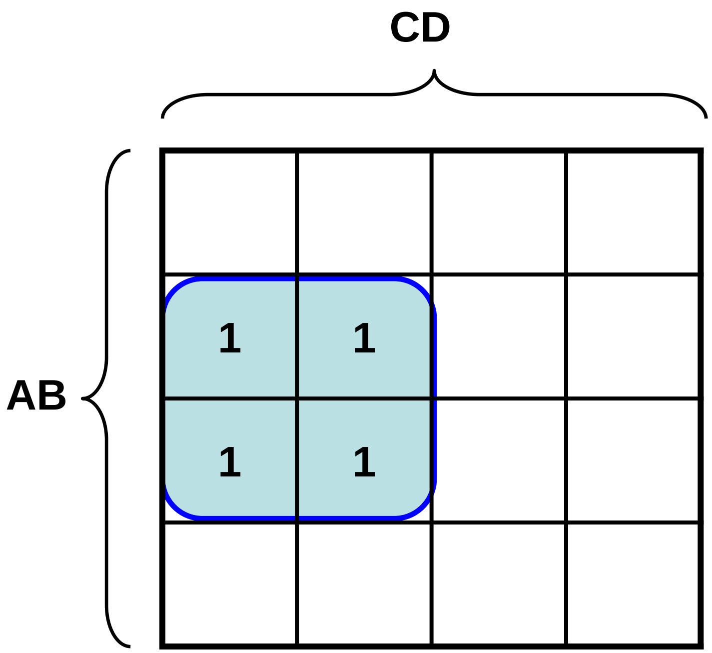


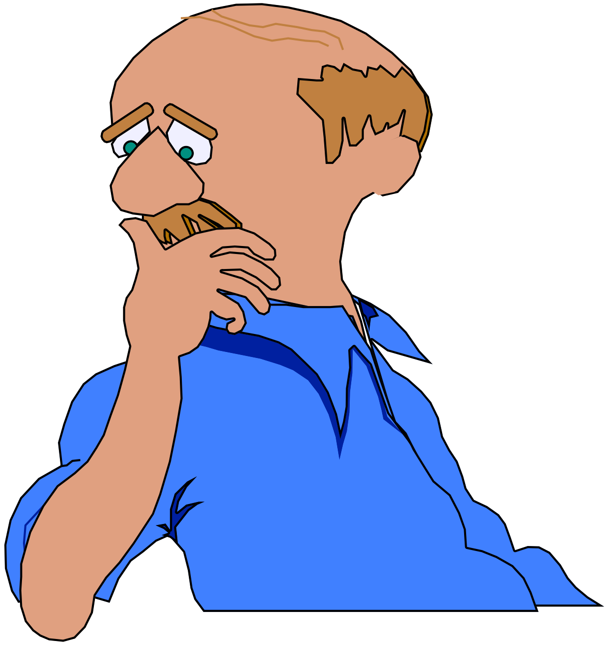
Answer: Karnaugh Maps



$$\overline{A}\,\overline{B}$$

Your turn: Karnaugh Maps





Answer: Karnaugh Maps

CD

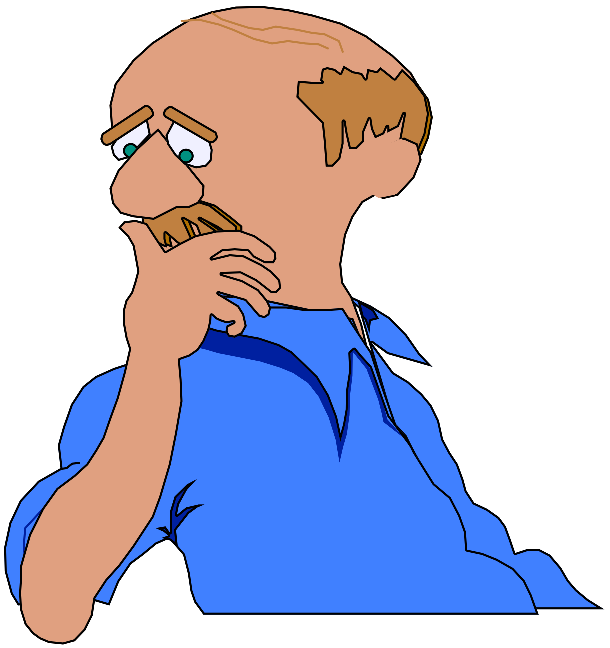
AB

1	1		
1	1		

$B\overline{C}$

Your turn: Karnaugh Maps

AB \ CD				
	1	1	1	1
	1	1	1	1



Answer: Karnaugh Maps

CD

AB

1	1	1	1
1	1	1	1

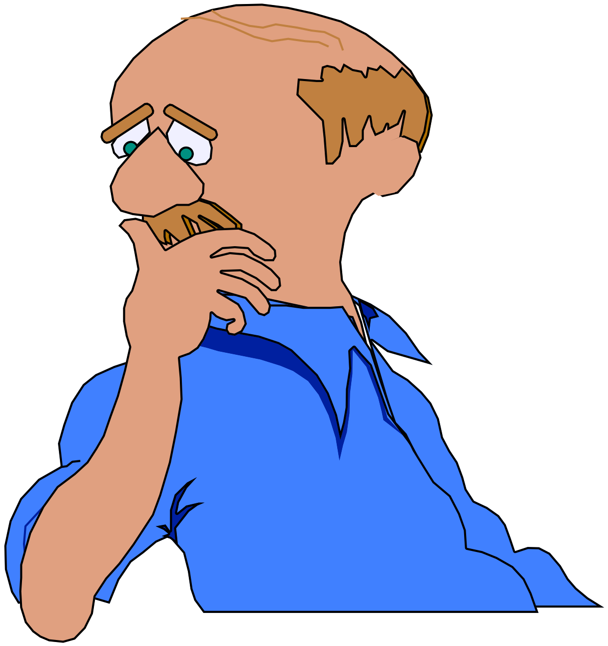
$\overline{A}$

Your turn: Karnaugh Maps

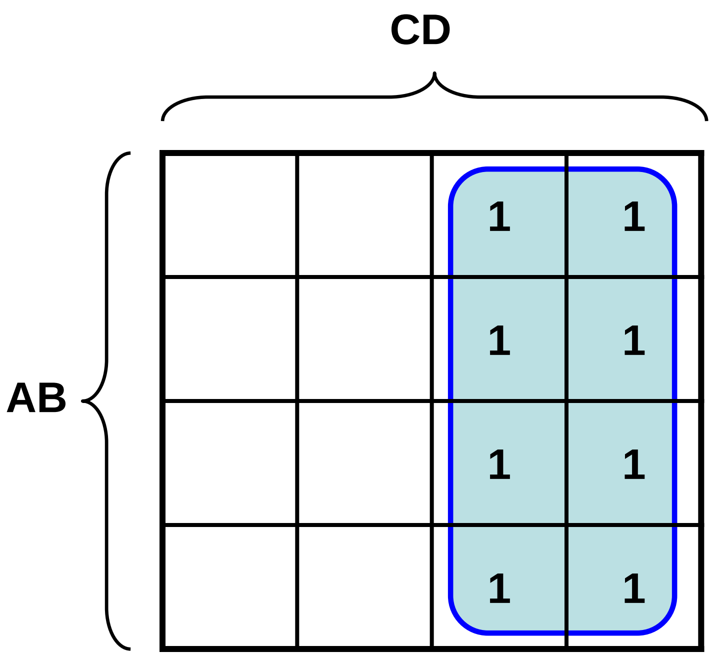
CD

AB

		1	1
		1	1
		1	1
		1	1



Answer: Karnaugh Maps



C

Simplified Labeling of Karnaugh Maps

- In attempting to simplify, first look for the largest grouping possible:
  - When you are circling groups, you are allowed to use the same **1** more than **once**
  - If any isolated **1**s remain after the groupings, then each of these is circled as a group of **1**s
  - Finally, before going from the map to a simplified Boolean expression, any group of **1**s that is **completely overlapped by other groups can be eliminated**

Karnaugh Maps: Overlapping Groups

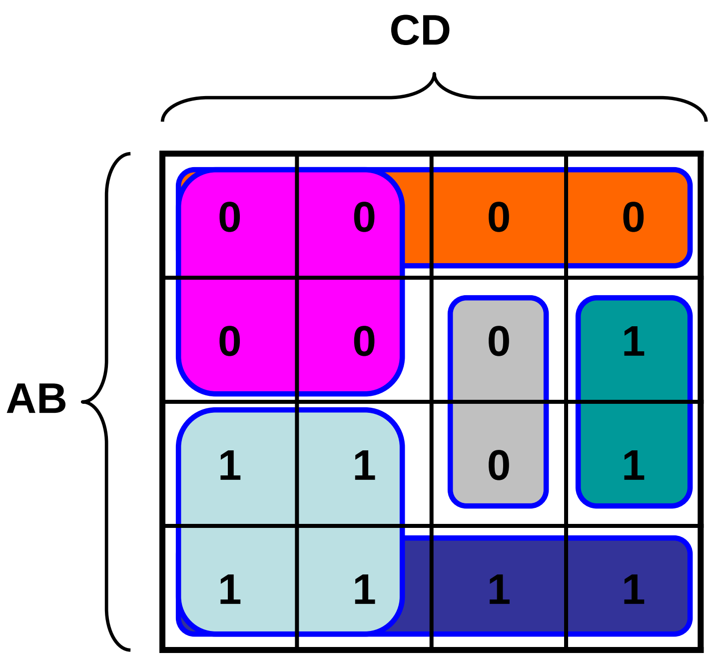
A \ BC	00	01	11	10
0			1	1
1				1

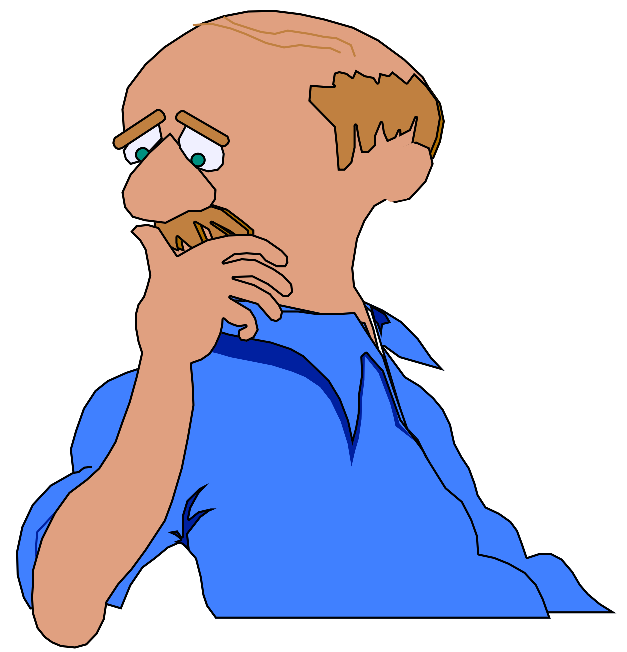
$$F = \overline{A}B + B\overline{C}$$

AB \ CD	00	01	11	10
00				
01		1		
11		1	1	
10			1	

$$F = B\overline{C}D + ACD$$

Your turn: Karnaugh Maps





Answer: Karnaugh Maps

AB \ CD				
	0	0	0	0
	0	0	0	1
	1	1	0	1
	1	1	1	1

$$F = A\bar{C} + A\bar{B} + BC\bar{D}$$

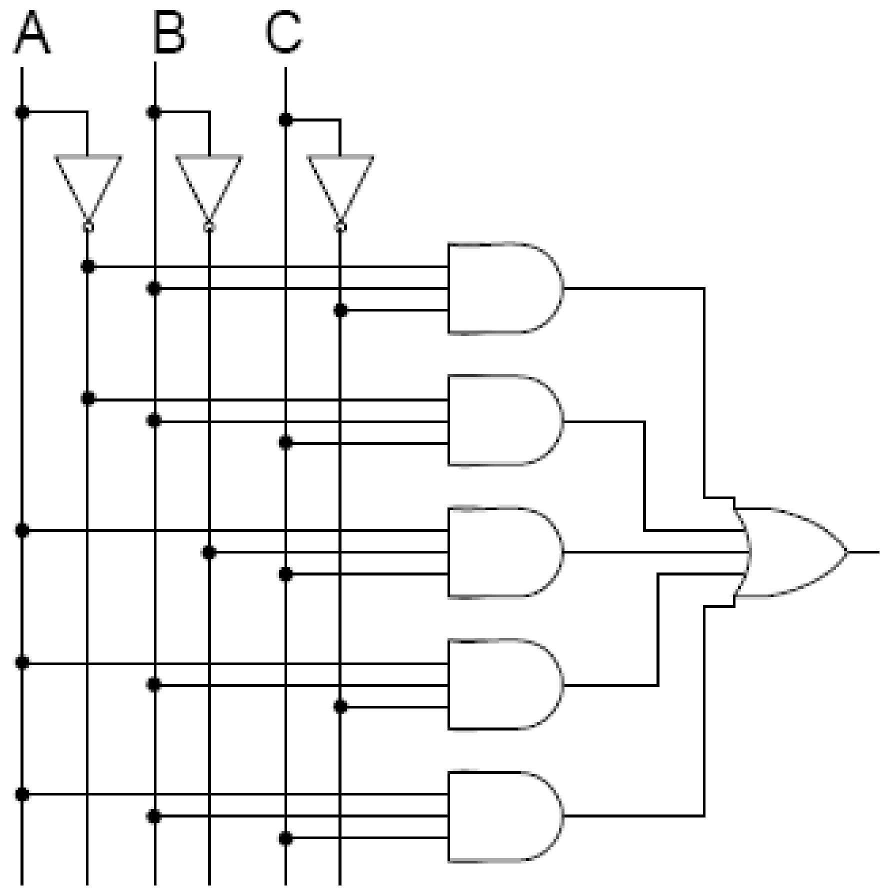
$$F = (A+C).(A+B).(\bar{B}+\bar{C}+\bar{D})$$

Drawing a Circuit

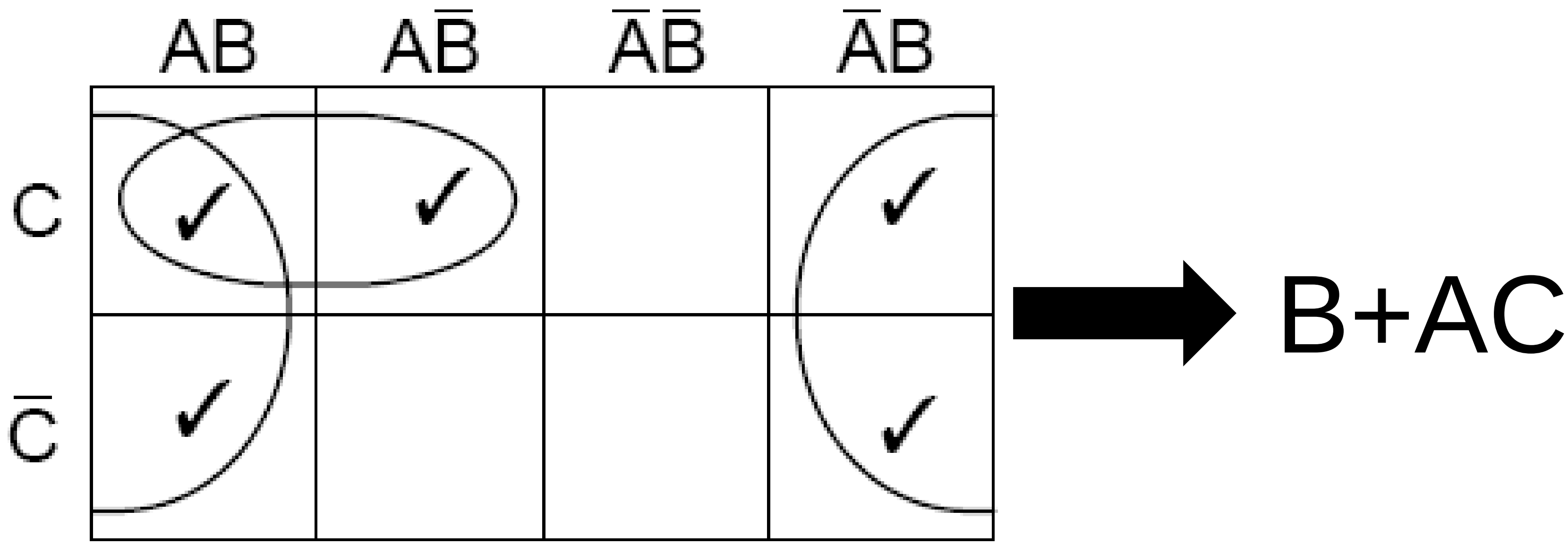
Sum-of-Products Expression

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

Digital Logic Circuit

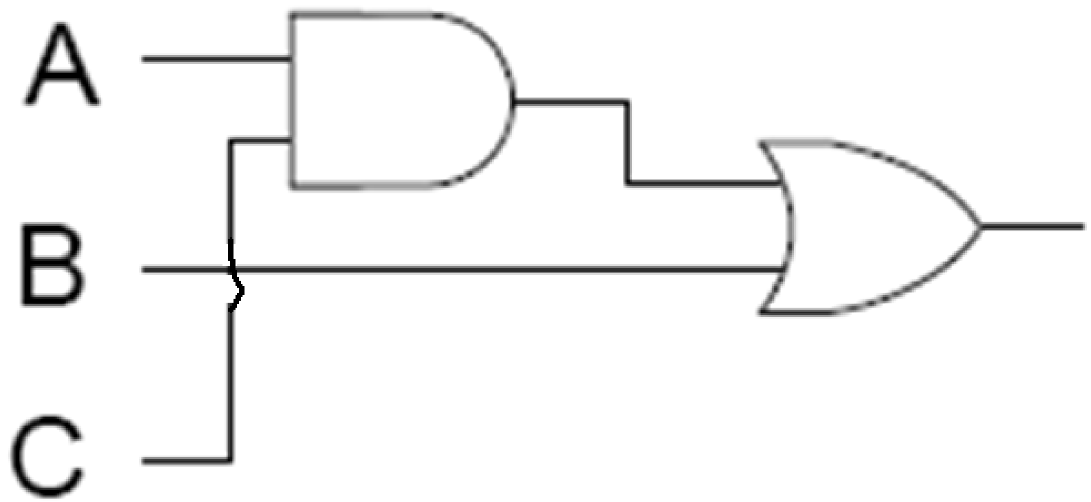


Drawing a Circuit



Drawing a Circuit

B+AC



Logic Operators

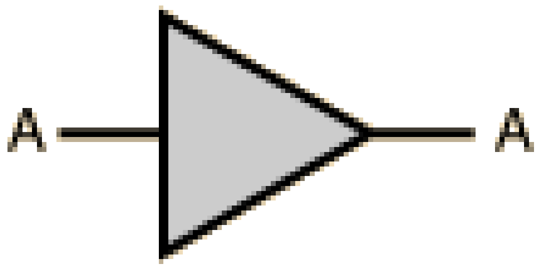
Logic Operations

- Basic logic operators and logic gates
- Boolean algebra
- Combinational Circuits
- Basic circuit design

Basic Logic Operators and Logic Gates

- AND
- OR
- NOT
- XOR (Exclusive OR)
- NOR
- NAND
- XNOR

Buffer



A	B
0	0
1	1

AND Operation

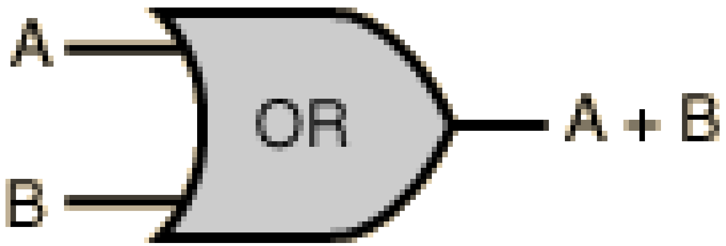
- . Operator
- ^ Operator
- **A . B = A ^ B**

A	B	A . B
0	0	0
0	1	0
1	0	0
1	1	1

OR Operation

- **+** Operator
- **v** Operator
- **A + B = A v B**

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

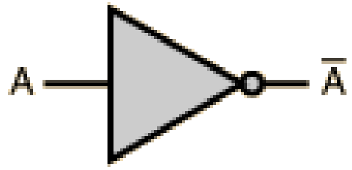


NOT Operation

- ~ Operator
- ¬ Operator

$$\overline{A} = \neg A = \sim A = A'$$

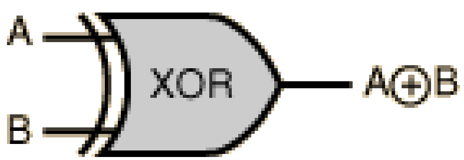
A	A’
0	1
1	0



XOR Operation

- $\oplus$ Operator

$$A \oplus B$$



A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

NAND Operation

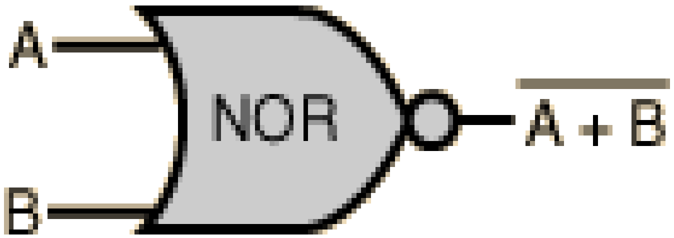
$$(\overline{A.B}) = (A.B)'$$

A	B	$(\overline{A.B})$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Operation

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

A	B	$(\overline{A+B})$
0	0	1
0	1	0
1	0	0
1	1	0



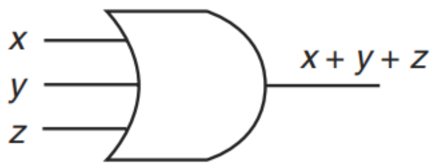
XNOR Operation

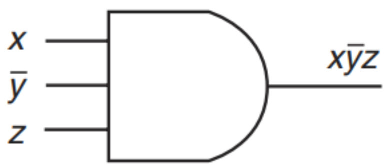
$$(\overline{A \oplus B})$$

A	B	$(\overline{A \oplus B})$
0	0	1
0	1	0
1	0	0
1	1	1

Multiple Input Gates

- Gates are not limited to two inputs.
- There can be logic gates with more than two inputs.





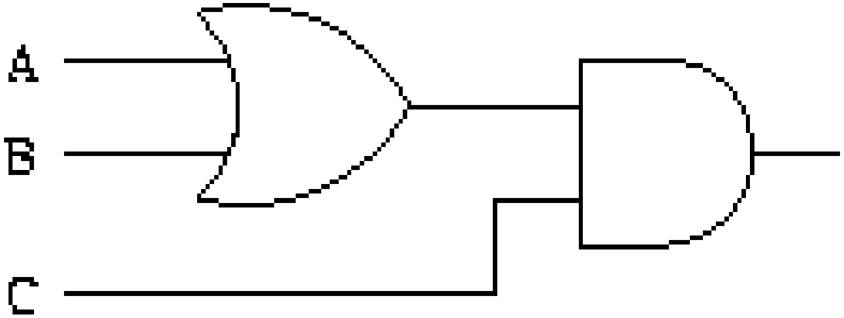
Drawing Logic Gates

- In addition to the basic gates, gates with 3,4, or more inputs can be used

    E.g. **x + y + z** can be implemented with a single **OR** gate with 3 inputs

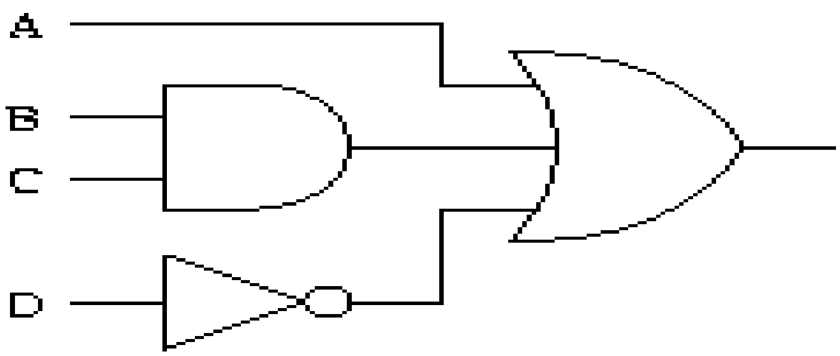
Drawing Logic Gates

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



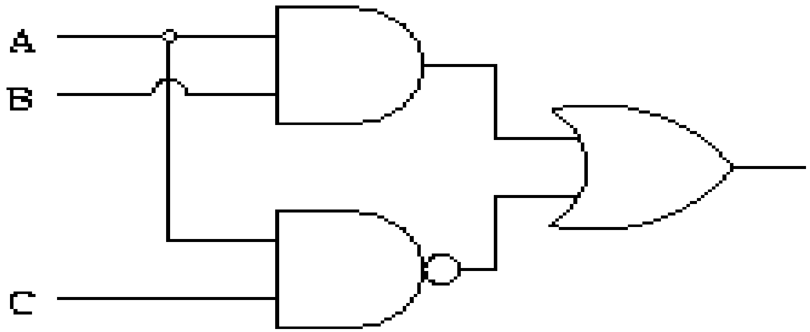
Drawing Logic Gates

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



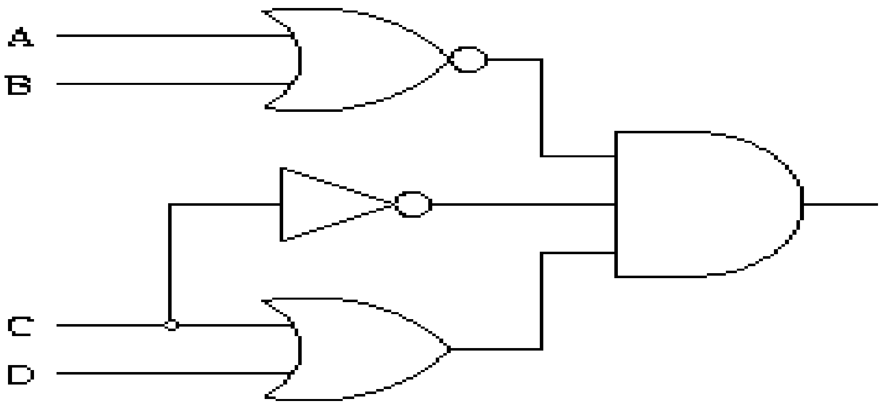
Drawing Logic Gates

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



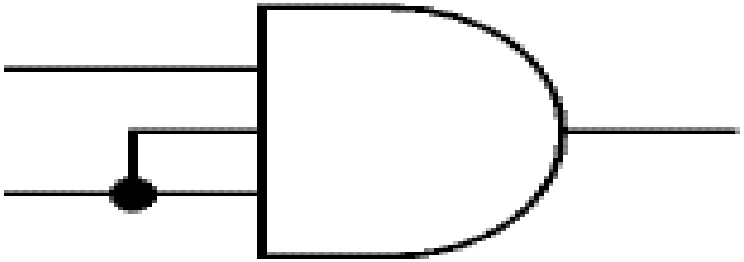
Drawing Logic Gates

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



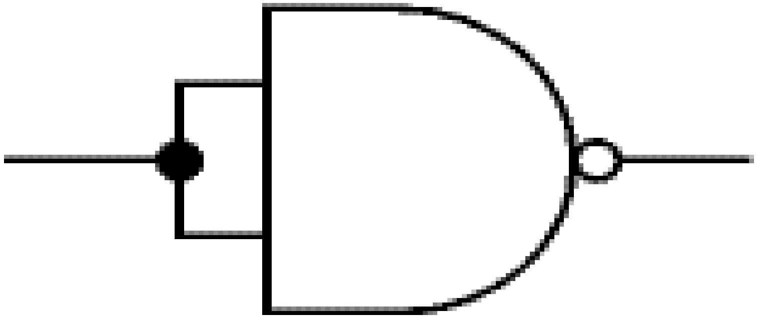
Reducing Logic Gates

- Reducing the number of inputs
  - The number of inputs to a gate can be reduced by connecting two (or more) inputs together
  - The diagram shows a 3-input **AND** gate operating as a 2-input **AND** gate



Reducing Logic Gates

- Reducing the number of inputs
  - Reducing a NOT gate from a NAND or NOR gate
  - The diagram shows this for a 2-input **NAND** gate



Logic Gates

- Typically, not all gate types are used in implementation
  - Design and fabrication are simpler if only one or two types of gates are used
  - Therefore, it is important to identify ***functionally complete*** sets of gates
  - This means that any **Boolean function** can be implemented using only the gates in the set

Logic Gates

- The following are functionally complete sets:
  - **AND, OR, NOT**
  - **AND, NOT**
  - **OR, NOT**
  - **NAND**
  - **NOR**

Logic Gates

- **AND**, **OR**, and **NOT** gates constitute a functionally complete set, since they represent the 3 operations of ***Boolean algebra***
- For the **AND** and **NOT** gates to form a functionally complete set, there must be a way to synthesize the **OR** operation from the **AND** and **NOT** operations

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

A OR B = NOT((NOT A) AND (NOT B))

Logic Gates

- Similarly, the **OR** and **NOT** operations are functionally complete because they can be synthesize the **AND** operation

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

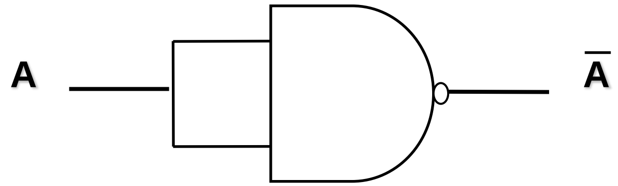
A AND B = **NOT**((**NOT A**) **OR** (**NOT B**))

Universal Gates

- The **AND, OR** and **NOT** functions can be implemented solely with **NAND** gates, and the same thing for **NOR** gates.
- For this reason, digital circuits can be, and frequently are, implemented solely with **NAND** gates or solely with **NOR** gates
- Therefore, **NAND** and **NOR** gates are referred to as a **Universal Gate**

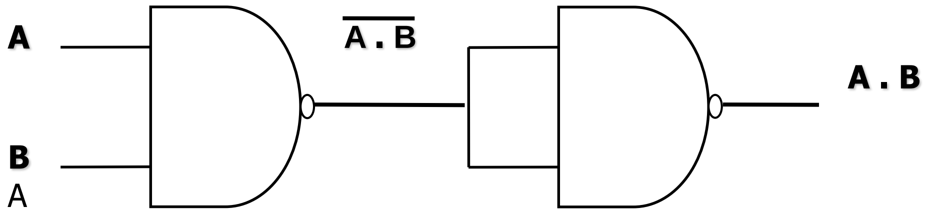
Logic Gates

- The diagram shows how the **NOT** function can be implemented solely with **NAND** gate



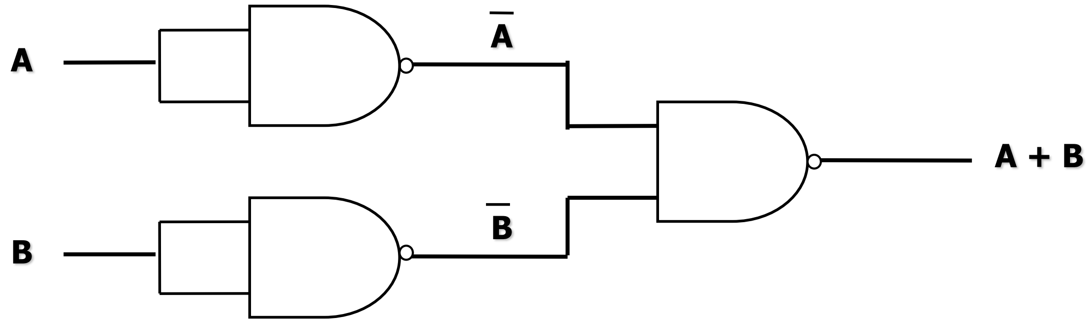
Logic Gates

- The diagram shows how the **AND** function can be implemented solely with **NAND** gate

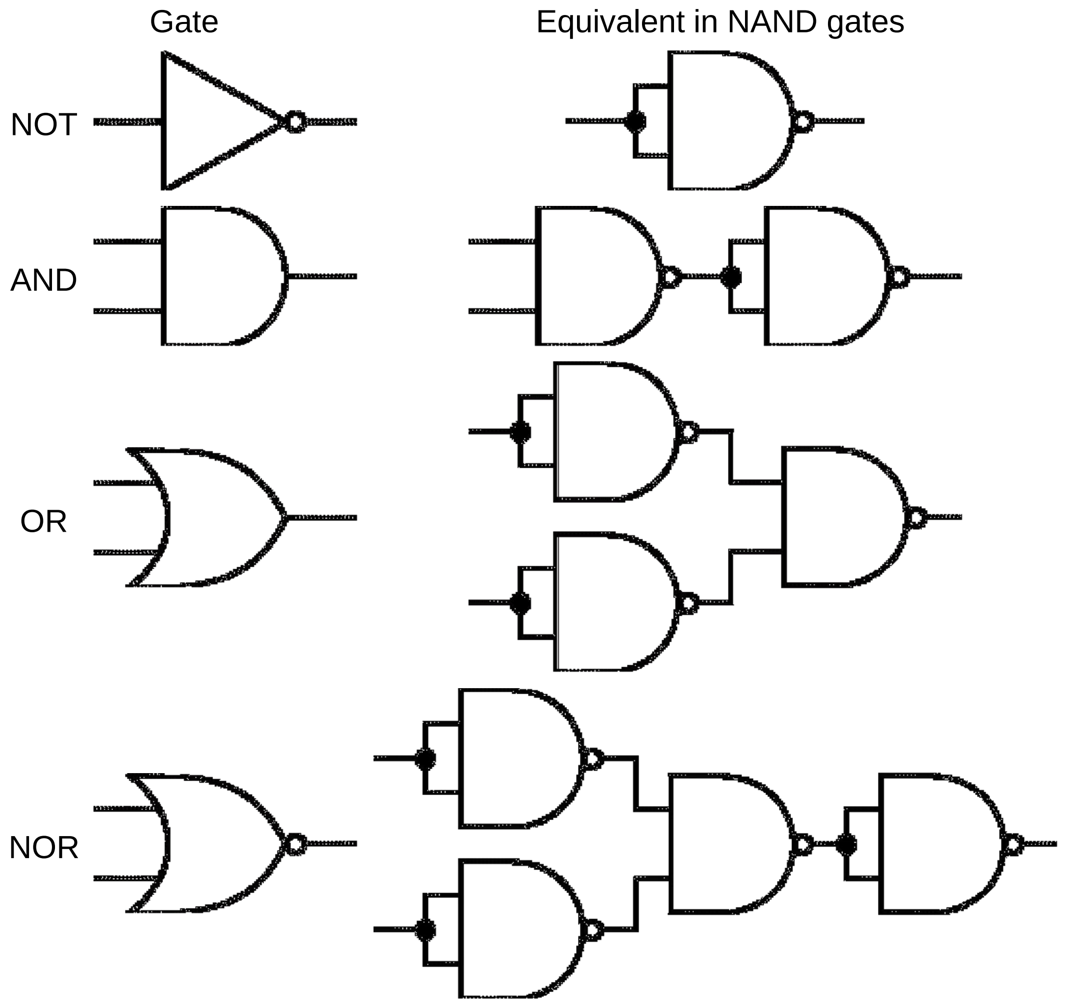


Logic Gates

- The diagram shows how the **OR** function can be implemented solely with **NAND** gate

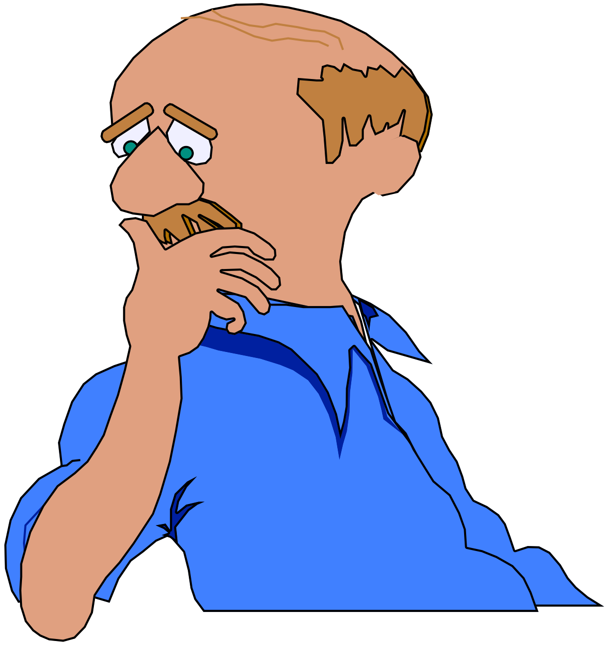


Logic Gates



Your turn

- Draw a diagram that shows how the **NOT** function can be implemented solely with **NOR** gate

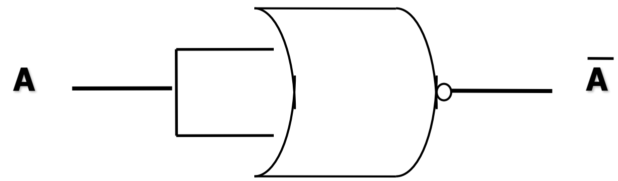


UCSC

BIT

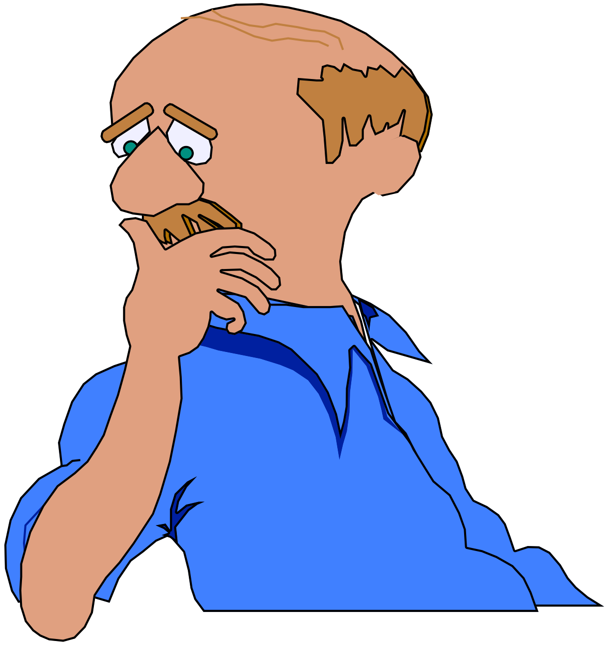
Logic Gates

- The diagram shows how the **NOT** function can be implemented solely with **NOR** gate



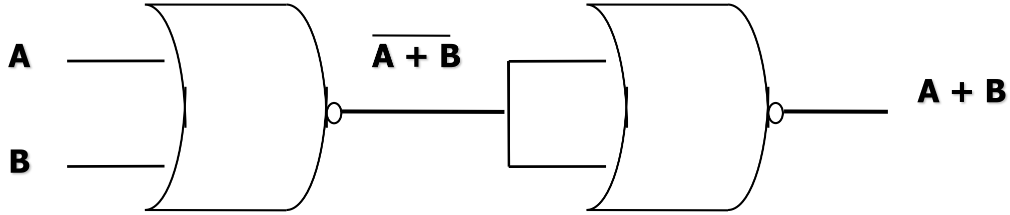
Your turn

- Draw a diagram that shows how the **OR** function can be implemented solely with **NOR** gate



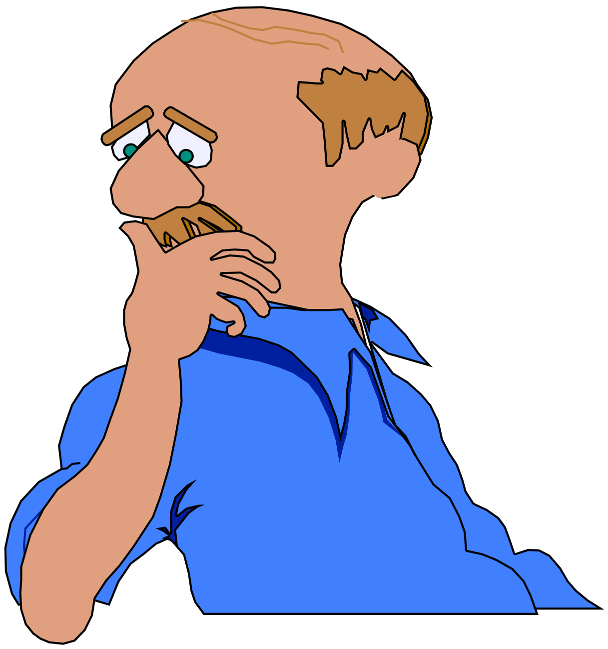
Logic Gates

- The diagram shows how the **OR** function can be implemented solely with **NOR** gate



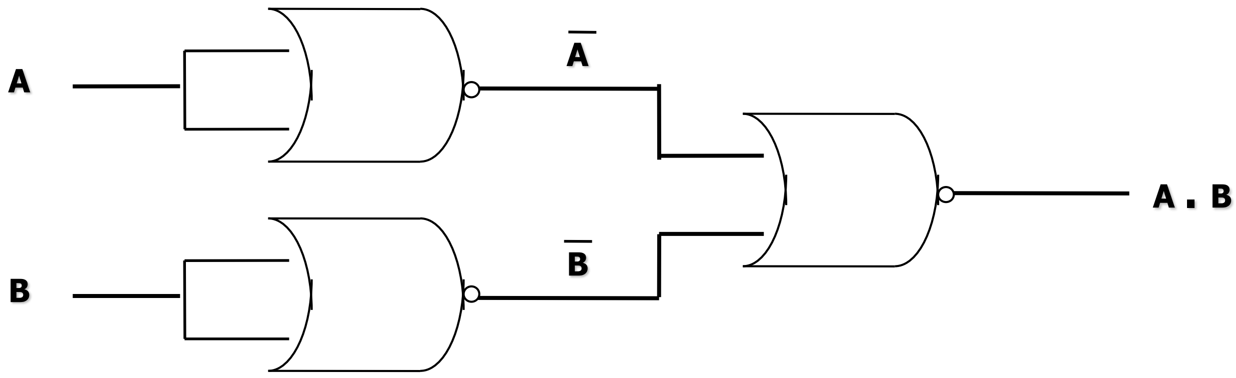
Your turn

- Draw a diagram that shows how the **AND** function can be implemented solely with **NOR** gate

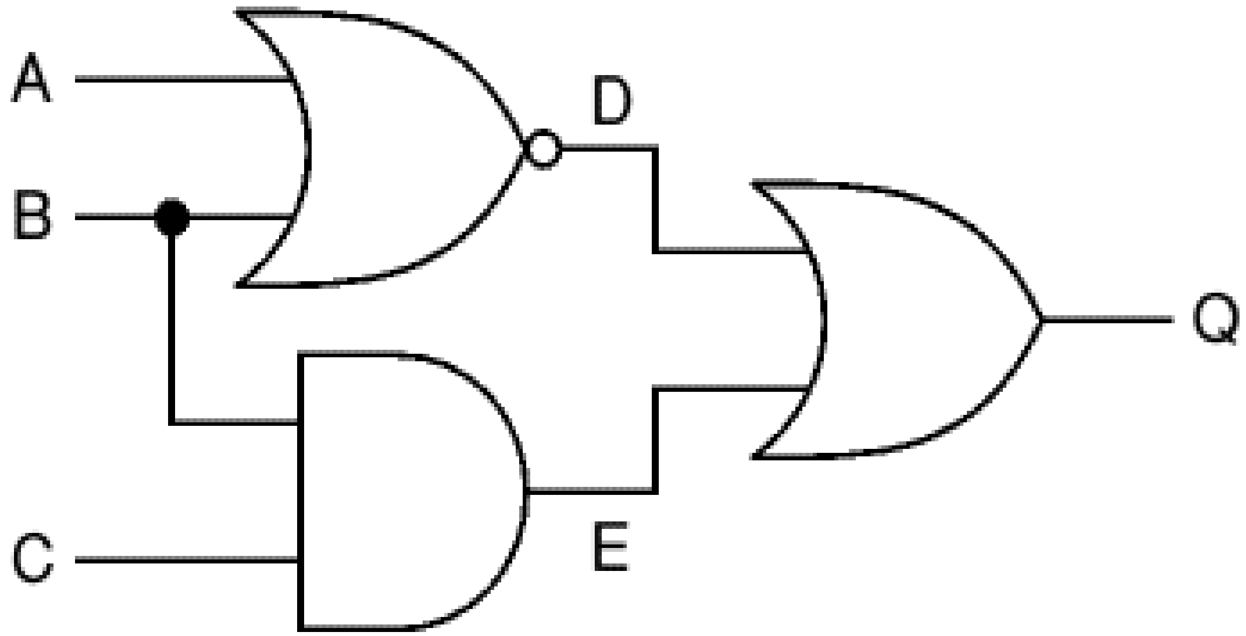


Logic Gates

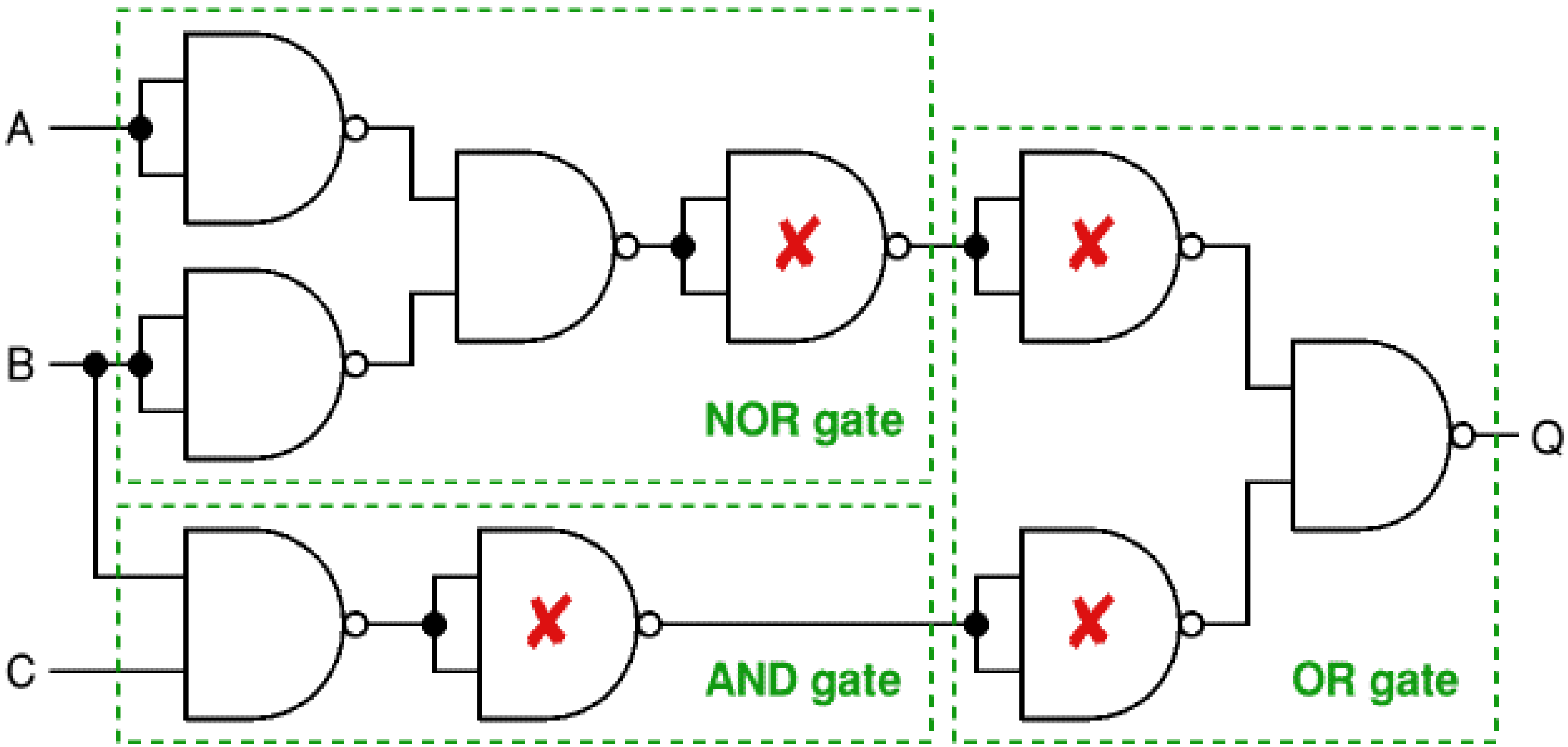
- The diagram shows how the **AND** function can be implemented solely with **NOR** gate



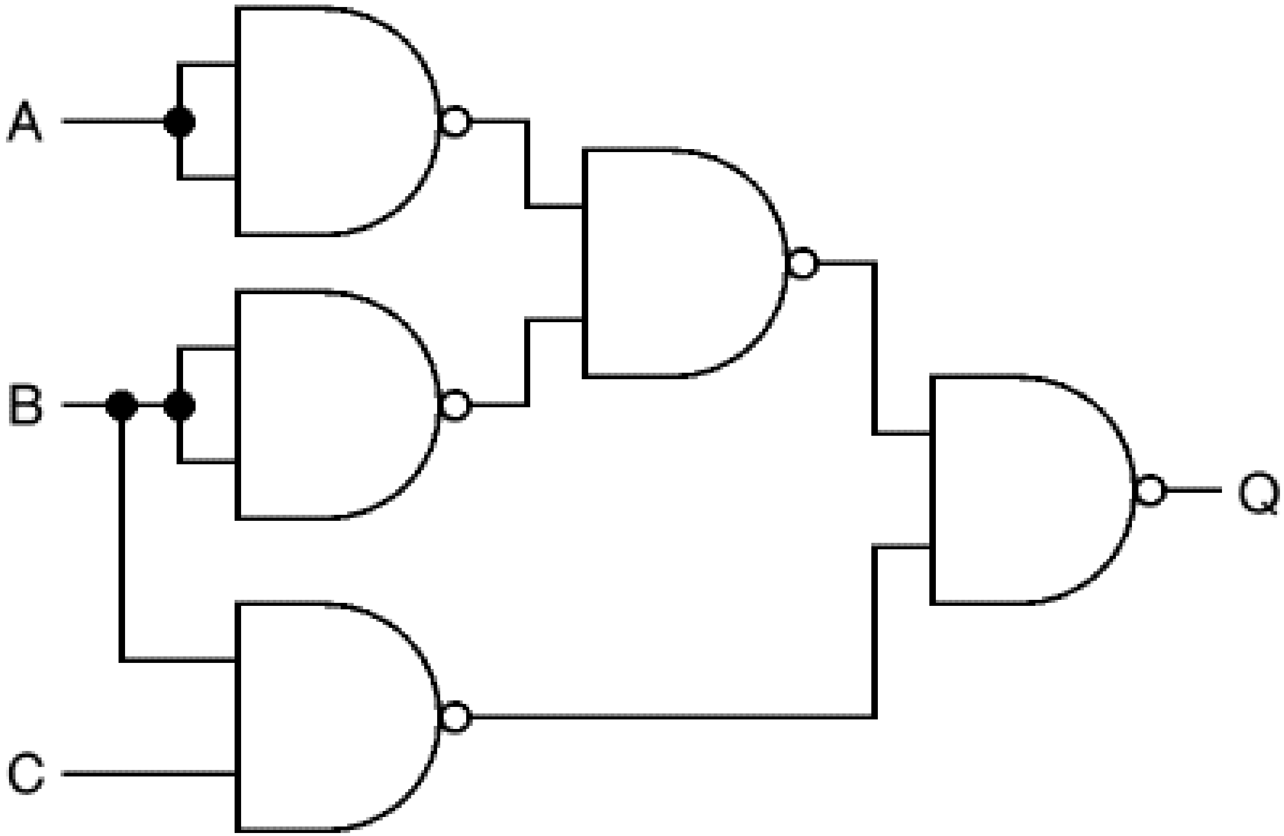
Substituting Gates in a Logic System



Substituting Gates in a Logic System



Substituting Gates in a Logic System



Digital Circuits and Boolean Algebra

- There are digital circuit elements to represent primary boolean algbeabric operations.
  - Boolean OR operation = OR Gate
  - Boolean AND operation = AND Gate
  - Boolean NOT operation = NOT Gate
- Complex boolean functions can be represented as combinations of logic gates.

Integrated Circuits

- Typically gates are not available as individual circuit units.
- Instead, multiple gates which are built into chips are available.
- A chip is typically mounted in a ceramic or plastic container with enternal pins to make connections.
- Not only these chips, but also complex circuits built into such a chip is generally regarded as an Integrated Circuit (IC).

Thank You