Competency 4: Uses logic gates to design basic digital circuits and devices.

4.1 Analyzes basic digital logic gates in term of their unique functionalities.

Time: 6 periods

Learning Outcomes

- Names basic logic gates and draws the appropriate symbols of them.
- Draws the truth tables for the basic logic gates
- Identify symbols that represent that negations of basic logic gates.
- Creates truth tables for given expressions
- Explains the need of universal gates.
- Explain the fabrication of any gate.

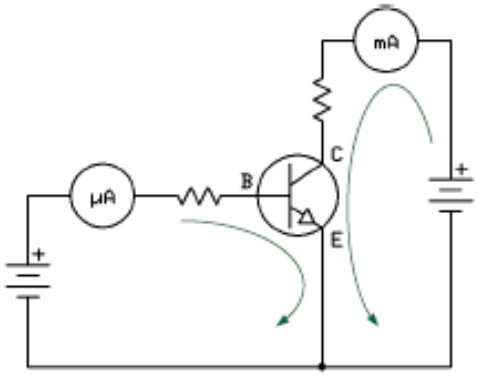
04.05.2022

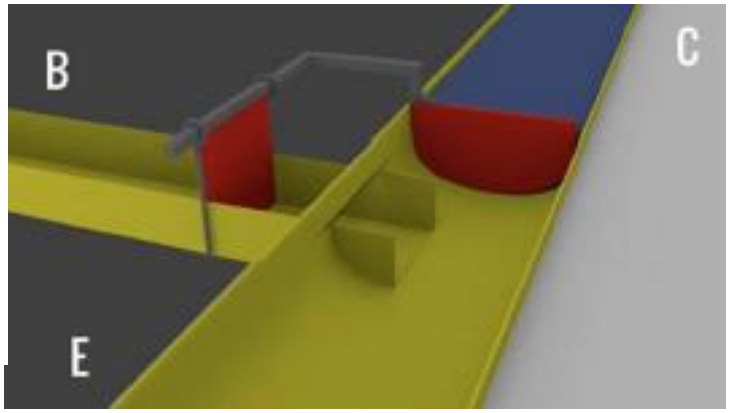
Computer is made up with several hardware levels

Hardware Level	Implemented with
Computer	Circuit boards
Circuit board	Many chips and other components
Chip	Many logic gates and other components
Gate	Many transistors and other components

Transistor

Transistor is a semiconductor device, used to switch electronic signals or simplify electrical power. When a small current flows in the Base-Emitter circuit, transistor would let to flow proportionately a large current in its Collector-Emitter circuit.





Gate

Gates are electronic components made up of with transistors. Gates are the fundamental building blocks of a digital circuit. Gates implements Boolean functions and Boolean functions used to make logical decisions.

How are digital circuits built? The answer lies in Boolean algebra. Programmers are familiar with the three basic Boolean functions:

AND (Conjunction) -True if, A and B are both true

OR (Disjunction) - True if, either A or B are true

NOT (negation) - True if input is false, false if input is true

Following figure lists the possible input values and the result of each function.

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

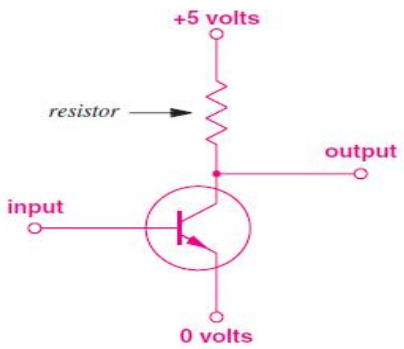
A	not A
0	1
1	0

Boolean functions and the result for each possible set of inputs. A logical value of zero represents *false*, and a logical value of one represents *true*.

Boolean functions are used in building digital hardware. More important, it is possible to use transistors to construct circuits that implement each of the Boolean functions. Thus, Boolean functions can be translated directly into hardware.

To understand the relationship between Boolean functions and hardware, consider the Boolean not.

Following figure illustrates a circuit that implements Boolean not.



A transistor and a resistor used to implement the Boolean function *not*. When the input is zero volts, the output is five volts, and vice versa.

To understand how the circuit operates, imagine the transistor to be a switch.

When it is turned on, the transistor connects the output to zero volts; when it is turned off, the transistor disconnects the output from zero volts, and the output registers five volts. An input of five volts causes the transistor to turn on, and an input of zero volts causes the transistor to turn off. Thus, the output is always the opposite of the input.

Boolean circuits are fundamental to digital systems, and are given the name **logic gates**. Engineers do not construct gates from individual transistors because manufacturers sell electronic parts (actually integrated circuits) that contain all the circuitry for a gate.

Basic characteristics of logic gates

- Each logic gate has a unique symbol and a unique logical function.
- A gate can have one input or multiple inputs and an output.
- An input can take 0 or 1 at a time.
- Input are processed according to the function of the particular logic gate and an output is provided.

What is a truth table?

- Truth table represents all the input combinations and corresponding output of a logic circuit.
- The following table shows all possible input combinations for two and three inputs only.

No. of Inputs	Value combinations	No. of combinations
2	0 0 0 1 1 0 1 1	$4 = 2^2$
3	0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	$8=2^3$

Therefore,

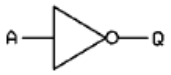
Number of input combination = $2^{\text{Number of inputs of logic gate/ circuit}}$

There are three basic logic gates.

1. NOT gate
2. AND gate
3. OR gate

NOT Gate

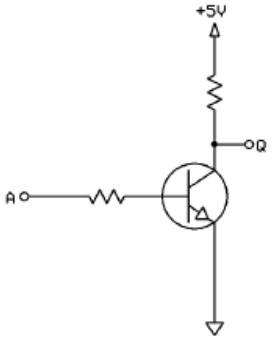
- Symbol, Truth table & Boolean expression



A	Q
0	1
1	0

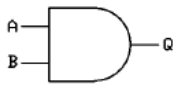
$Q = \overline{A}$

- Transistor implementation



AND Gate

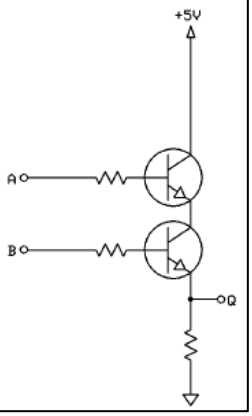
- Symbol, Truth table & Boolean expression

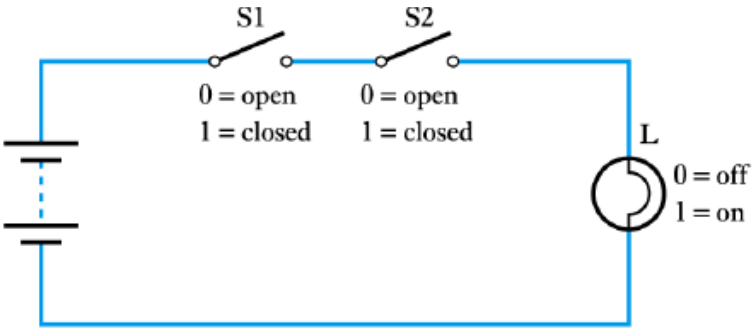


A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Q = A . B

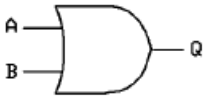
- Transistor implementation





OR Gate

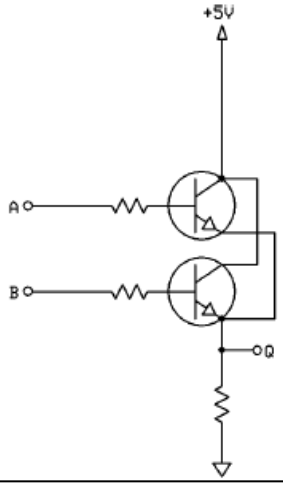
- Symbol, Truth table & Boolean expression

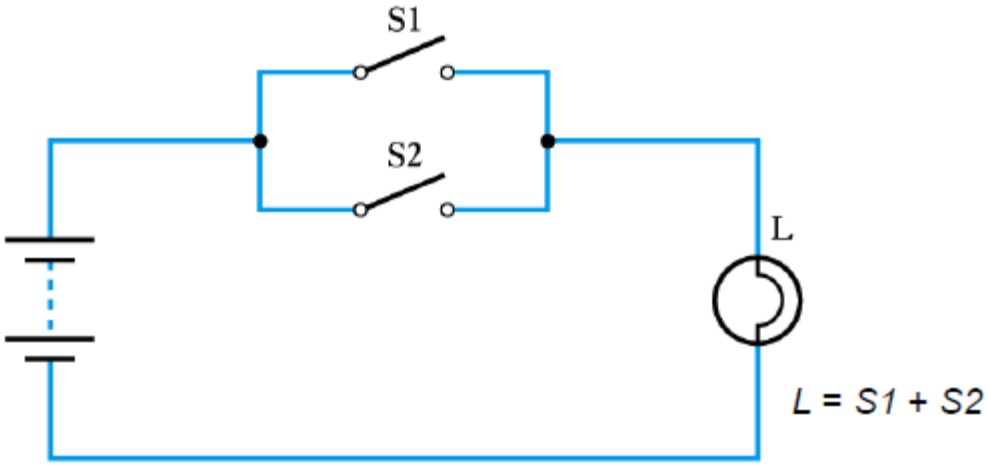


A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Q = A + B

- Transistor implementation

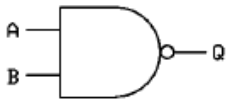




Other Gates

NAND Gate

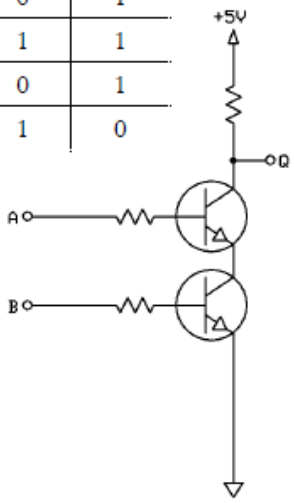
- Symbol, Truth table & Boolean expression



A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

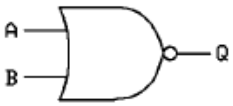
$$Q = \overline{A \cdot B}$$

- Transistor implementation



NOR Gate

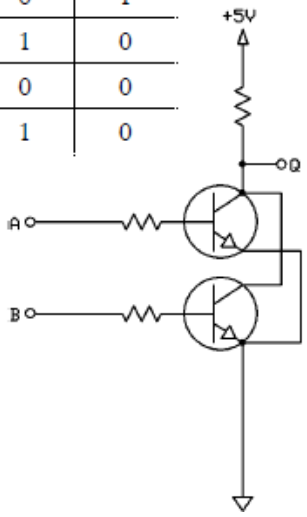
- Symbol, Truth table & Boolean expression



A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

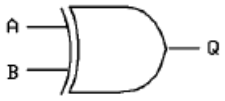
$$Q = \overline{A + B}$$

- Transistor implementation



XOR Gate

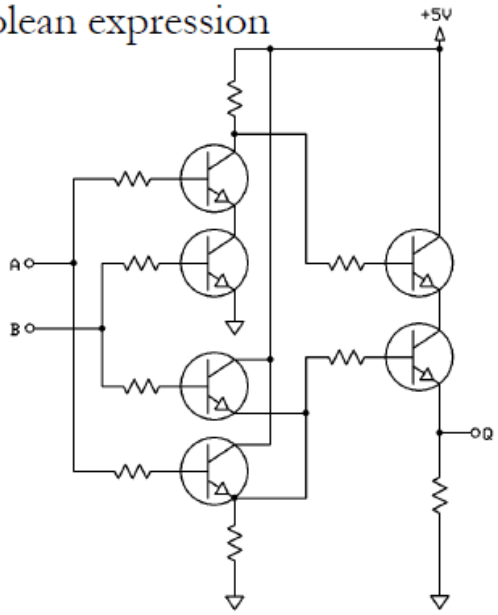
- Symbol, Truth table & Boolean expression



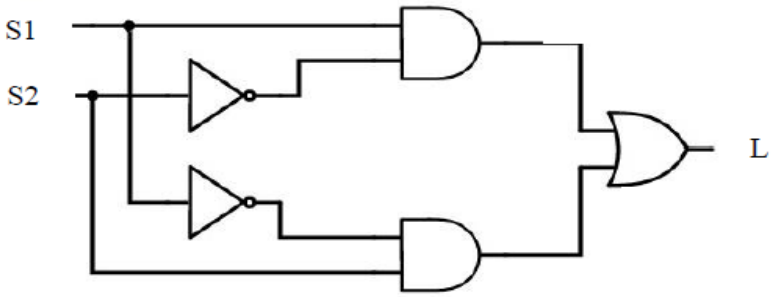
$Q = A \oplus B$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

- Transistor implementation



XOR gate using basic logic gates



S1	S2	$\overline{S1}$	$\overline{S2}$	$S1.\overline{S2}$	$\overline{S1}.S2$	$S1.\overline{S2}+\overline{S1}.S2$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

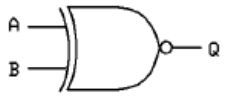
Truth table for three input XOR gate

INPUTS				Final Output
S1	S2	S3	S1 ⊕ S2	S1 ⊕ S2 ⊕ S3
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

- Compare S1 & S2 and if the inputs are same then output is zero (false). Otherwise output is one. Then compare the result of S1 ⊕ S2 with S3. if the values are same then final output is 0. else final output is 1.

XNOR Gate

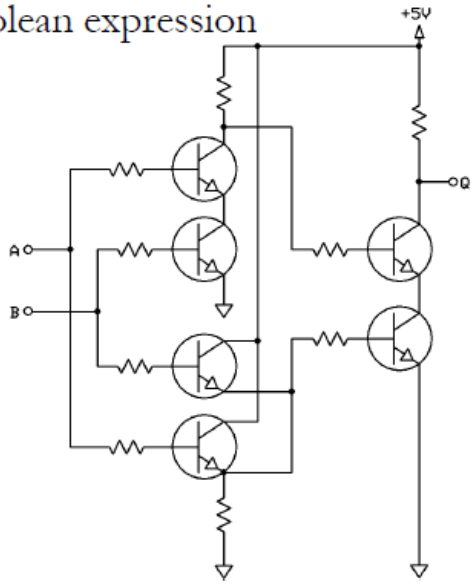
- Symbol, Truth table & Boolean expression



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

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

- Transistor implementation

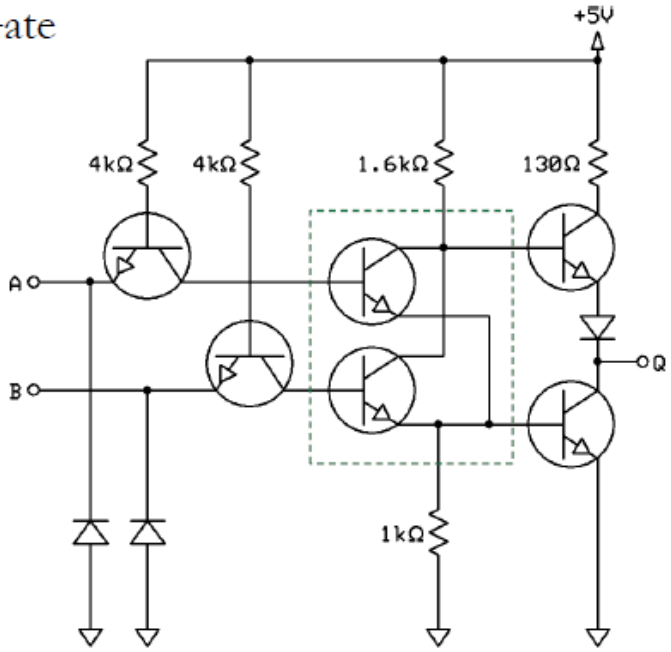


Truth table for three input XNOR gate

INPUTS					Final Output
S1	S2	S3	S1 ⊕ S2	S1 ⊕ S2 ⊕ S3	$\overline{S1 \oplus S2 \oplus S3}$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	0	0	1
1	1	1	0	1	0

Practical Implementation

- NOR Gate



05.05.2022

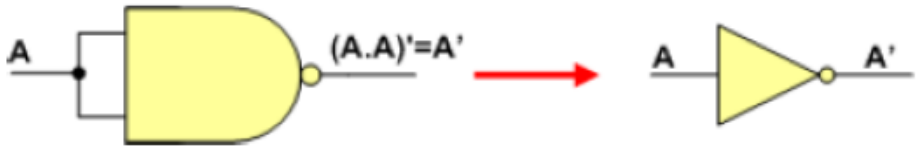
Universal Gates

- A universal logic gate is a logic gate that can be used to construct all other logic gates.
- The **NAND** gate and **NOR** gates can be considered as universal logic gates.

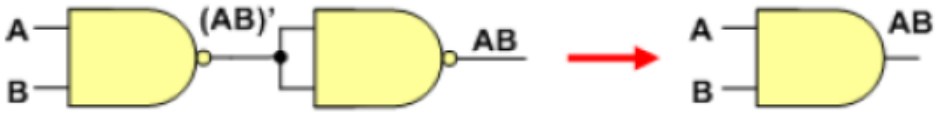
- **The advantage of universal gates**: NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

NAND Gate

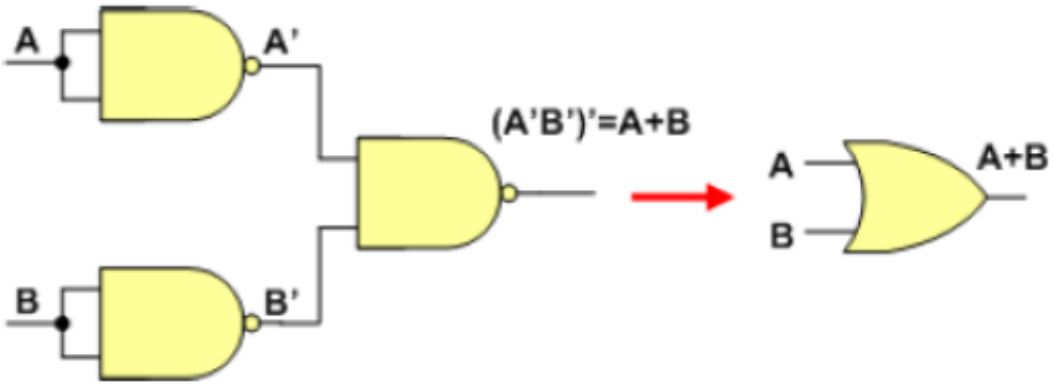
1. How to implement NOT gate using NAND gates?



2. How to implement AND gate using NAND gates?

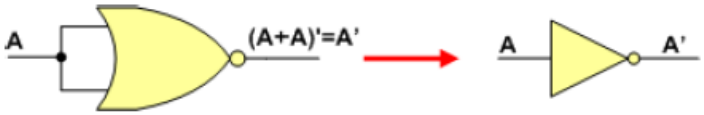


3. How to implement OR gate using NAND gates?



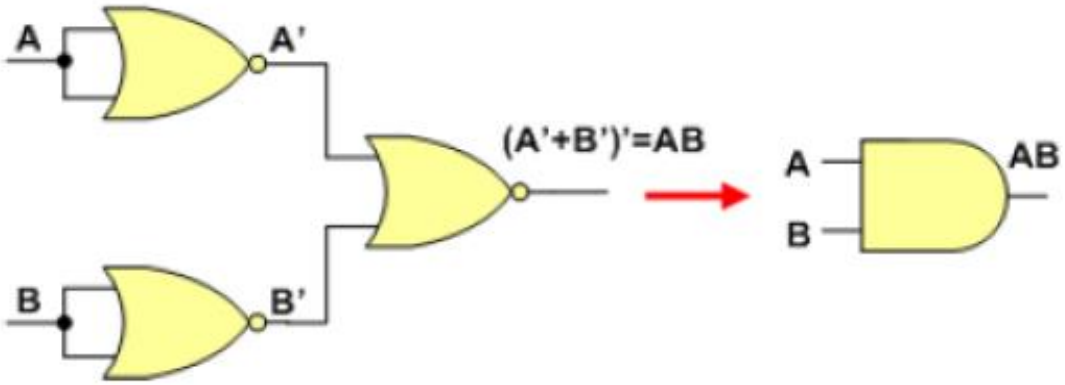
NOR Gate

1. How to implement NOT gate using NOR gates?



2. How to implement OR gate using NOR gates?

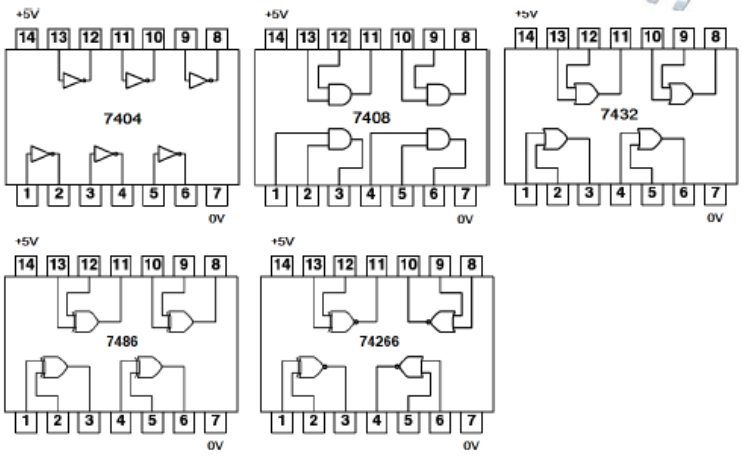
3. How to implement AND gate using NOR gates?



Packaging

- Gates are packaged into integrated circuits

- Examples (single/two input gates)
  - NOT
  - AND
  - OR
  - XOR
  - XNOR



- Examples (two input gates)
  - NAND
  - NOR
- When constructing logic circuits, why are NAND and NOR gates preferred over other gates?

06.05.2022

Simplifies logic expressions using law of Boolean algebra and Karnaugh map.

Boolean algebra

Boolean algebra (developed by George Bool -mathematician and logician) is an algebra for the manipulation of objects that can take on only two values, typically true and false, although it can be any pair of values.

Boolean Expressions

In addition to binary objects, Boolean algebra also has operations that can be performed on these objects, or variables. Combining the variables and operators yields Boolean expressions. A Boolean function typically has one or more input values and yields a result, based on these input values, in the range {0,1}.

In Boolean algebra, logical **negation (NOT operation), logical conjunction (AND operation) and logical disjunction (OR operation)** are performed using:

-(bar)

. (dot) and

+ (plus) operators respectively.

Let's look at the **Boolean operators AND, OR, and NOT** to see how each is represented, using both Boolean algebra and truth tables.

Inputs		Outputs
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

LE 3.1 **The Truth Table for AND**

Inputs		Outputs
x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 3.2 **The Truth Table for OR**

$\bar{x}$ and x'

Inputs	Outputs
x	$\bar{x}$
0	1
1	0

The Truth Table for NOT

Introduction to Boolean laws

What is the requirement of Boolean algebra simplification?

Frequently, a Boolean expression is not in its simplest form. Recall from algebra that an expression such as 2x + 6x is not in its simplest form; it can be reduced (represented by fewer or simpler terms) to 8x.

Boolean expressions can also be simplified, but we need new identities, or laws, that apply to Boolean algebra instead of regular algebra.

Simplification of Boolean expressions reduces the number of operations and number of logic gates for the implementation.

Most laws exist in two forms. One is multiplicative form and the other is additive form. At the multiplication, variables in the Boolean expression are multiplied by each other and at the addition, variables in the Boolean expression are added to each other.

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	$(\overline{xy}) = \bar{x}+\bar{y}$	$(\overline{x+y}) = \overline{xy}$
Double Complement Law	$\bar{\bar{x}} = x$	

1. Idempotent Law

Law and its representation from a truth table

A.A=A

A	A	A.A
0	0	0
1	1	1

A+A=A

A	A	A+A
0	0	0
1	1	1

$\overline{A}.\overline{A} = \overline{A}$

A	$\overline{A}$	$\overline{A}$	$\overline{A}.A$
0	1	1	1
1	0	0	0

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

A	$\overline{A}$	$\overline{A}$	$\overline{A}+\overline{A}$
0	1	1	1
1	0	0	0

Above highlighted columns are identical.

2. Identity Law

1.A=A

1	A	1.A
1	0	0
1	1	1

0+A=A

0	A	0+A
0	0	0
0	1	1

0.A = 0

0	A	0.A
0	0	0
0	1	0

1+A = 1

1	A	1+A
1	1	1
1	0	1

Above highlighted columns are identical.

3. Inverse/Complement Law

Multiplicative form

$A.\overline{A}=0$

A	$\overline{A}$	$A.\overline{A}$
0	1	0
1	0	0

Additive form

$A+\overline{A}=1$

A	$\overline{A}$	$A+\overline{A}$
0	1	1
1	0	1

4. De Morgan's Law

A mathematician named De Morgan developed a pair of important laws regarding group complementation in Boolean algebra.

Multiplicative form

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

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

Additive form

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

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

Above highlighted columns are identical.

5. Double Complement Law

$A = \overline{\overline{A}}$

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

6. Commutative Law

Multiplicative form

$AB = BA$

A	B	AB	BA
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Above highlighted columns are identical.

Additive form

$A + B = B + A$

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

7. Associative Law

Multiplicative form

A (BC) = (AB) C

A	B	C	BC	AB	A(BC)	(AB)C
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

Additive form.

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

A	B	C	B+C	A+B	A+(B+C)	(A+B)+C
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

8. Distributive Law

A (B+C) = AB+AC

A	B	C	B+C	AB	AC	A (B+C)	AB+AC
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1

9. Redundancy Law

Form 1

A + AB =A

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

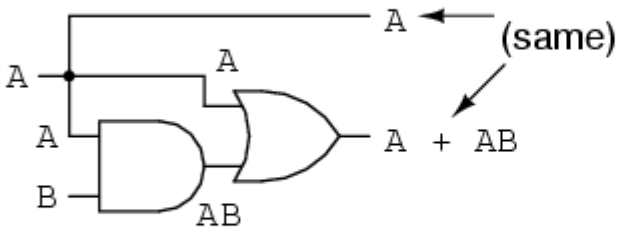
Form 2

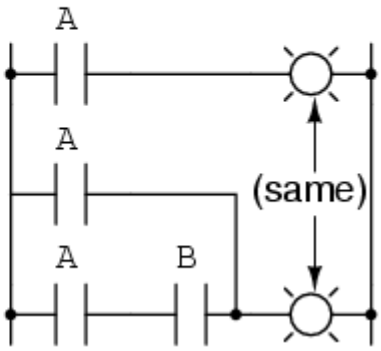
A + AB = A + B

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

Examples

A + AB = A





A + AB

↓ Factoring **A** out of both terms

A(1 + B)

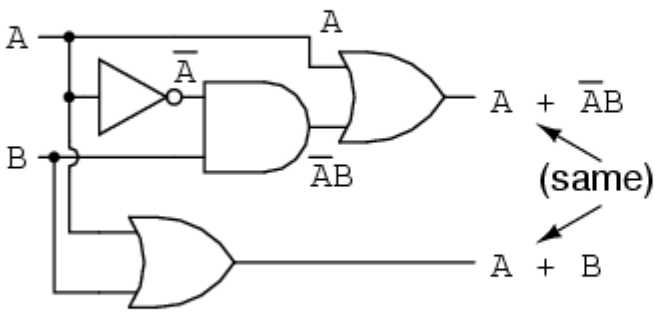
↓ Applying identity **A + 1 = 1**

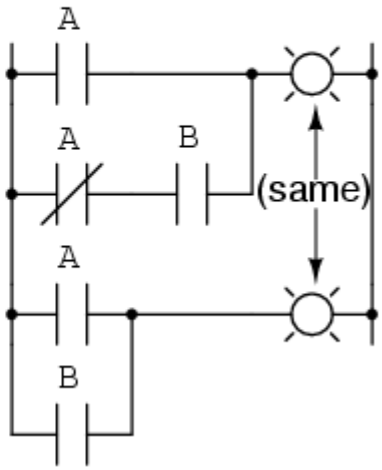
A(1)

↓ Applying identity **1A = A**

A

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





A + $\overline{A}$B

↓ Applying the previous rule to expand **A** term
A + AB = A

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

↓ Factoring **B** out of 2nd and 3rd terms

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

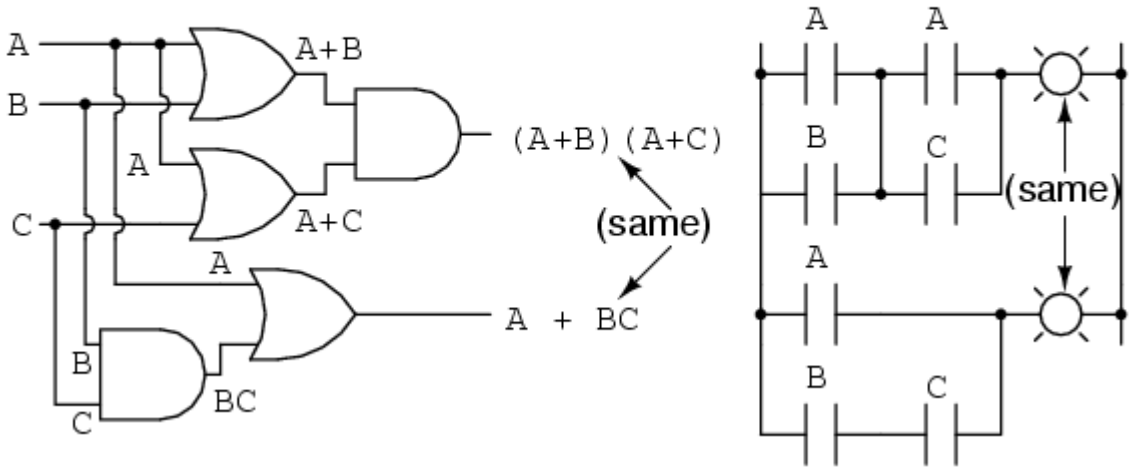
↓ Applying identity **A + $\overline{A}$ = 1**

A + B(1)

↓ Applying identity **1A = A**

A + B

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



(A + B)(A + C)

↓ Distributing terms

AA + AC + AB + BC

↓ Applying identity **AA = A**

A + AC + AB + BC

↓ Applying rule **A + AB = A** to the A + AC term

A + AB + BC

↓ Applying rule **A + AB = A** to the A + AB term

A + BC

10.05.2022

Standard forms in Boolean expressions

There exist two standard forms of Boolean expressions

-SOP (Sum of Products)

-POS (Product of Sum)

The Sum-of-Products (SOP) Form

- An SOP expression → when two or more product terms are summed by Boolean summation.
- Examples:

  A'B+A'BC'+AC

  ABC+CDE+B'CD'

  AB+ABC

- Also:

  A+A'B'C+BCD'

- In an SOP form, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar:

example:

$\overline{A}\,\overline{B}\,\overline{C}$ is correct.

$\overline{ABC}$ Is not correct.

- A standard SOP expression is one in which all the variables in the domain appear in each product term in the expression.

  Example:

  AB'CD+ A'B'CD' +ABC'D'

Convert the following General Boolean expression into standard SOP form

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

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

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

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

$$A\overline{B}C + \overline{A}\,\overline{B} + AB\overline{C}D = A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}D$$

- Standard SOP expressions are important in:
  - Constructing truth tables
  - The Karnaugh map simplification method

The Product-of-Sum (POS) Form

- When two or more sum terms are multiplied, the result expression is a product-of-sums (POS):

Examples:

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

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

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

Also: → $\overline{A}(\overline{A} + \overline{B} + C)(B + C + \overline{D})$

- In a POS form, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar:

example: $\overline{A}+\overline{B}+\overline{C}$ is correct.

However, $\overline{A+B+C}$ is not correct

The Standard POS Form

A standard POS expression is one in which all the variables in the domain appear **in each sum term** in the expression.

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

Standard POS expressions are important in:

- Constructing truth tables
- The Karnaugh map simplification method

Convert the following Boolean expression into standard POS form:

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

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

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

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

Boolean expressions and truth tables

- All standard Boolean expression can be easily converted into truth table format using binary values for each term in the expression.

- Also, standard SOP or POS expression can be determined from the truth table.

Develop a truth table for the standard SOP expression

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

Inputs			Output	Product Term
A	B	C	X	
0	0	0	0	
0	0	1	1	$\overline{A}\,\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

- **Develop a truth table for the standard SOP expression**

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

Inputs			Output	Product Term
A	B	C	X	
0	0	0	0	$(A+B+C)$
0	0	1	1	
0	1	0	0	$(A+\overline{B}+C)$
0	1	1	0	$(A+\overline{B}+\overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A}+B+\overline{C})$
1	1	0	0	$(\overline{A}+\overline{B}+C)$
1	1	1	1	

Determining Standard expression from a truth table

I/P			O/P
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- There are four 1s in the output and the corresponding binary value are 011, 100, 110, and 111.
  - $011 \rightarrow \overline{A}BC$
  - $100 \rightarrow A\overline{B}\overline{C}$
  - $110 \rightarrow AB\overline{C}$
  - $111 \rightarrow ABC$

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

- There are four 0s in the output and the corresponding binary value are 000, 001, 010, and 101.
  - $000 \rightarrow A+B+C$
  - $001 \rightarrow A+B+\overline{C}$
  - $010 \rightarrow A+\overline{B}+C$
  - $101 \rightarrow \overline{A}+B+\overline{C}$

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

Minterms and Maxterms

Minterm

- Each row of a truth table can be associated with a minterm, which is a product (AND) of all variables in the function, in direct or complemented form. A minterm has the property that it is equal to 1 on exactly one row of the truth table.

Maxterm

- Each row of a truth table is also associated with a maxterm, which is a sum (OR) of all the variables in the function, in direct or complemented form. A maxterm has the property that it is equal to 0 on exactly one row of the truth table.

A	B	C	Z		
0	0	0	0	$A+B+C$	
0	0	1	1	$\bar{A}\bar{B}C$	
0	1	0	0	$A+\bar{B}+C$	Maxterm
0	1	1	0	$A+\bar{B}+\bar{C}$	
1	0	0	0	$\bar{A}+B+C$	
1	0	1	1	$A\bar{B}C$	Minterm
1	1	0	1	$AB\bar{C}$	
1	1	1	1	ABC	

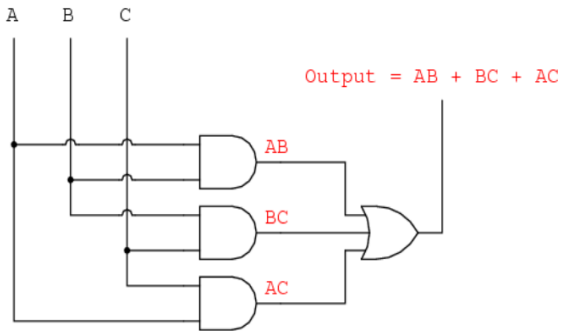
Boolean equation based on SOP method

$$Z = \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C} + ABC$$

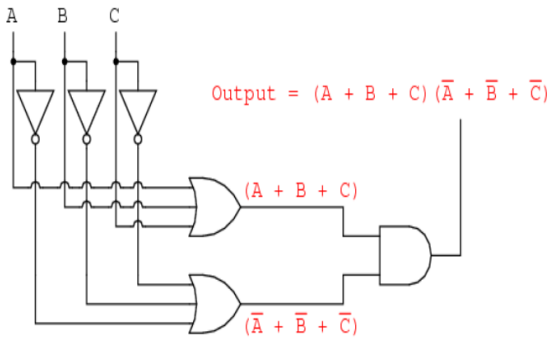
Boolean equation based on POS method

$$Z = (A+B+C)(A+\bar{B}+C)$$
$$(A+\bar{B}+\bar{C})(\bar{A}+B+C)$$

Logic circuit based on SOP



Logic circuit based on POS



Transform SOP into POS and vice versa

SOP TO POS	
$F = A.\overline{B} + B.\overline{C} + \overline{A}.C$	*Obtain the compliment of whole equation*
$= \overline{A.\overline{B} + B.\overline{C} + \overline{A}C}$	*by adding an over bar to the whole*
$= \overline{A.\overline{B}} + \overline{B.\overline{C}} + \overline{\overline{A}C}$	De Morgan's Law
$= (\overline{A} + \overline{\overline{B}}).(\overline{B} + \overline{\overline{C}}) + (\overline{\overline{A}} + \overline{C})$	De Morgan's Law
$= \underline{(\overline{A} + B).(\overline{B} + C).(A + \overline{C})}$	Double Inverse

POS TO SOP	
$F = (\overline{A} + B).(\overline{B} + C).(A + \overline{C})$	*Obtain the compliment of whole equation*
$= \overline{(\overline{A} + B).(\overline{B} + C).(A + \overline{C})}$	*by adding an over bar to the whole*
$= (\overline{\overline{A} + B}) + (\overline{\overline{B} + C}) + (\overline{A + \overline{C}})$	De Morgan's Law
$= (\overline{\overline{A}.\overline{B}}) + (\overline{\overline{B}.\overline{C}}) + (\overline{A}.\overline{\overline{C}})$	De Morgan's Law
$= \underline{A.\overline{B} + B.\overline{C} + \overline{A}.C}$	Double Inverse

07.06.2022

Simplifies logic expressions using Karnaugh map

Karnaugh maps, or Kmaps, are a graphical way to represent Boolean functions. Karnaugh maps provide an alternative way of simplifying logic circuits.

A map is simply a table used to enumerate the values of a given Boolean expression for different input values. The rows and columns correspond to the possible values of the function's inputs. Each cell represents the outputs of the function for those possible inputs.

The arrangement of 0's and 1's within the map helps you to visualize the logic relationships between the variables and leads directly to a simplified Boolean statement.

Cell = 2^n where n is a number of variables

Two input K-Map
Cell = 2^2 =4

A / B	0	1
0		
1		

Three input K-Map
Cell = 2^3 =8

AB / C	00	01	11	10
0				
1				

Four input K-Map
Cell = 2^4 =16

AB / CD	00	01	11	10
00				
11				
11				
10				

If a product term includes all of the variables exactly once, either complemented or not complemented, this product term is called a **minterm.**

As an example, consider the following Boolean function

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

Minterm	x	y
$\bar{X}\bar{Y}$	0	0
$\bar{X}Y$	0	1
$X\bar{Y}$	1	0
XY	1	1

Minterms for Two Variables

The minterms for three variables, along with the input values they represent, are shown in following table.

Minterm	x	y	z
$\bar{X}\bar{Y}\bar{Z}$	0	0	0
$\bar{X}\bar{Y}Z$	0	0	1
$\bar{X}Y\bar{Z}$	0	1	0
$\bar{X}YZ$	0	1	1
$X\bar{Y}\bar{Z}$	1	0	0
$X\bar{Y}Z$	1	0	1
$XY\bar{Z}$	1	1	0
XYZ	1	1	1

Minterms for Three Variables

K- map with 2 variables

Consider the function F(x,y) = xy and its truth table

$$F(x,y) = xy$$

x	y	x y
0	0	0
0	1	0
1	0	0
1	1	1

The corresponding Kmap is:

x \ y	0	1
0	0	0
1	0	1

$$F(x,y) = x + y$$

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

x \ y	0	1
0	0	1
1	1	1

$$
\begin{aligned}
F(x,y) &= \bar{x}y + x\bar{y} + xy \\
&= \bar{x}y + xy + x\bar{y} + xy \qquad \text{(remember, } xy + xy = xy\text{)} \\
&= y(\bar{x} + x) + x(\bar{y} + y) \\
&= y + x \\
&= x + y
\end{aligned}
$$

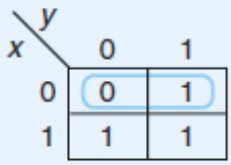
How did we know to add in an extra xy term? Algebraic simplification using Boolean identities can be very tricky. This is where K-maps can help.

x \ y	0	1
0	0	1
1	1	1

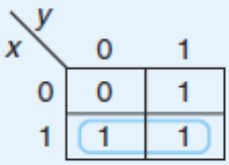
Kmap for *F(x,y)* = *x* + *y*

Rules of K- map simplification

1. No zeros allowed. 2. No diagonals. 3. Only power of 2 number of cells in each group. 4. Groups should be as large as possible.	5. Every "one" must be in at least one group. 6. Overlapping allowed. 7. Wrap around allowed. 8. Fewest number of groups possible.

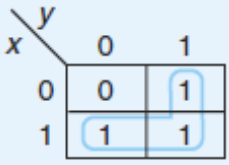


a) Incorrect

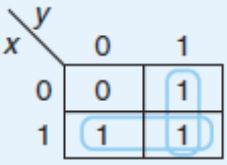


b) Correct

FIGURE 3A.4 **Groups Contain Only 1s**

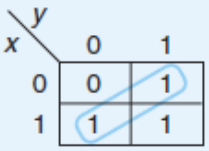


a) Incorrect

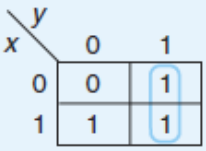


b) Correct

FIGURE 3A.6 **Groups Must Be Powers of 2**

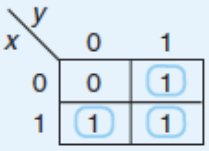


a) Incorrect

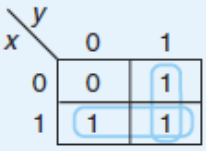


b) Correct

FIGURE 3A.5 **Groups Cannot Be Diagonal**

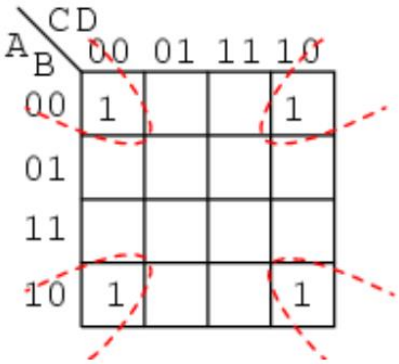


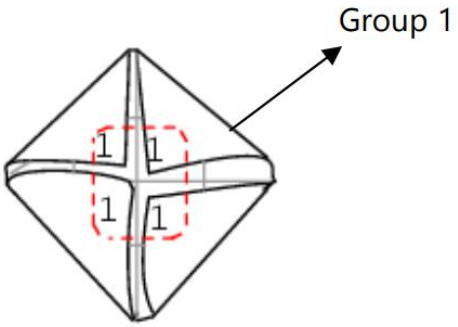
a) Incorrect



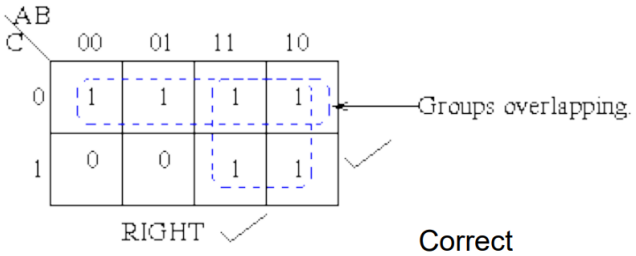
b) Correct

FIGURE 3A.7 **Groups Must Be as Large as Possible**

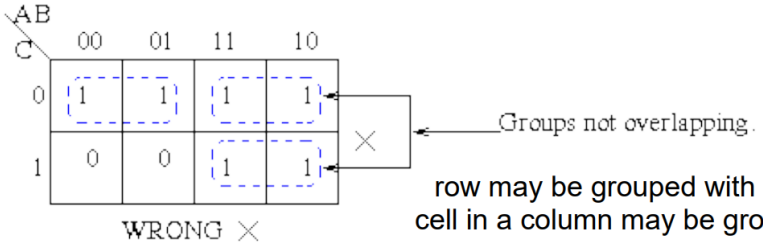




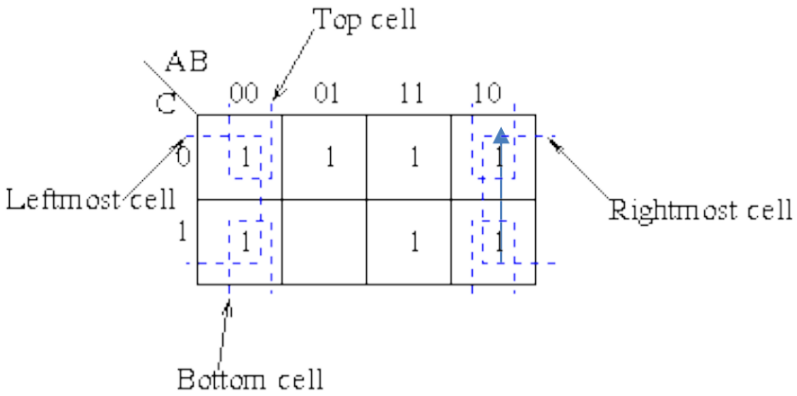
Groups may overlap.



Correct



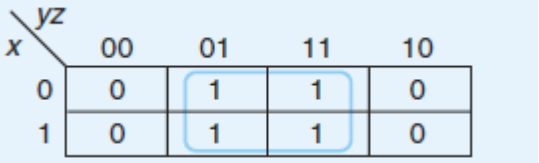
Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.



K-map with 3 variables

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

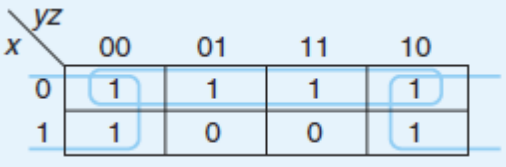
x \ yz	00	01	11	10
0	0	1	1	0
1	0	1	1	0



Answer is - F(x,y,z) = z

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

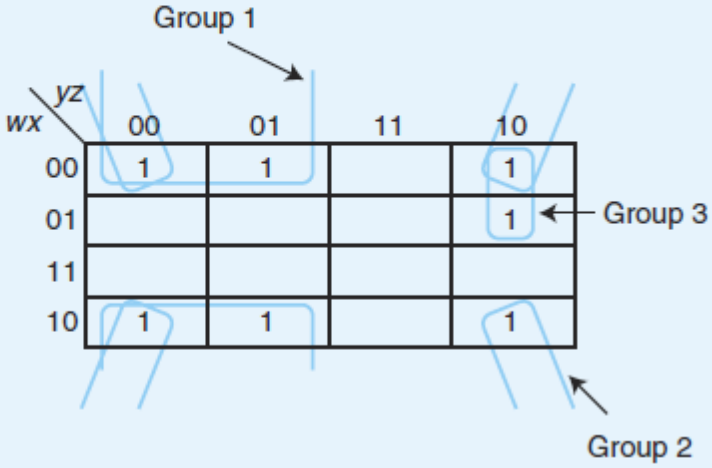
x \ yz	00	01	11	10
0	1	1	1	1
1	1	0	0	1



the final minimized function is $F(x,y,z) = \bar{x} + \bar{z}$.

K-map with 4 variables

$$F(w, x, y, z) = \overline{w}\,\overline{x}\,\overline{y}\,\overline{z} + \overline{w}\,\overline{x}\,\overline{y}z + \overline{w}\,\overline{x}y\overline{z} + \overline{w}xy\overline{z} + w\overline{x}\,\overline{y}\,\overline{z} + w\overline{x}\,\overline{y}z + w\overline{x}y\overline{z}$$



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

10.06.2022

How combinational logic circuits are used in CPU and sequential circuits in physical memory

Major building blocks of CPU

- Half Adder
- Full Adder

An adder is a digital circuit that performs addition of numbers.

Half Adder

The half adder adds two binary digits.

The addition of single bits.

- 0+0 = 0
- 0+1 = 1
- 1+0 = 1
- 1+1 = 10

These are the least possible single-bit combinations. However, the result for 1+1 is 10. Therefore, the sum result must be re-written as a 2-bit output. Thus, the above equations can be written as

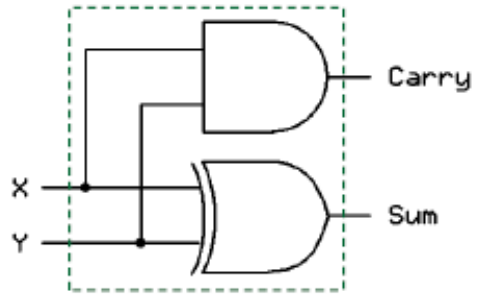
0+0 = 00

0+1 = 01

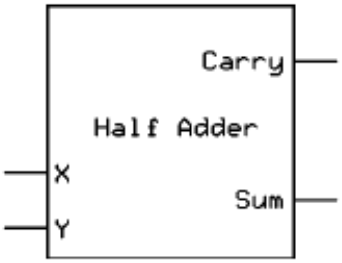
1+0 = 01

1+1 = 10

Here the output '1'of '10' becomes the carry-out. The result is shown in a truth-table below. 'SUM' is the normal output and 'CARRY' is the carry-out.



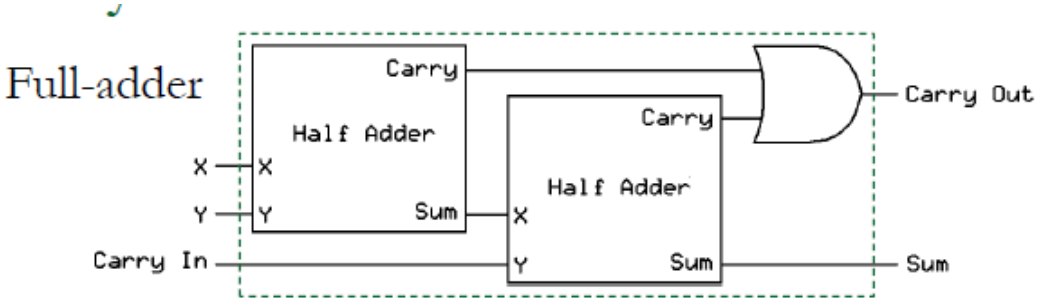
X	Y	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



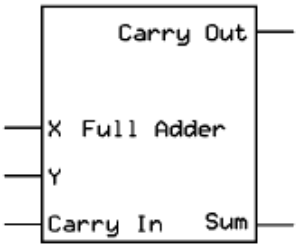
Full Adder

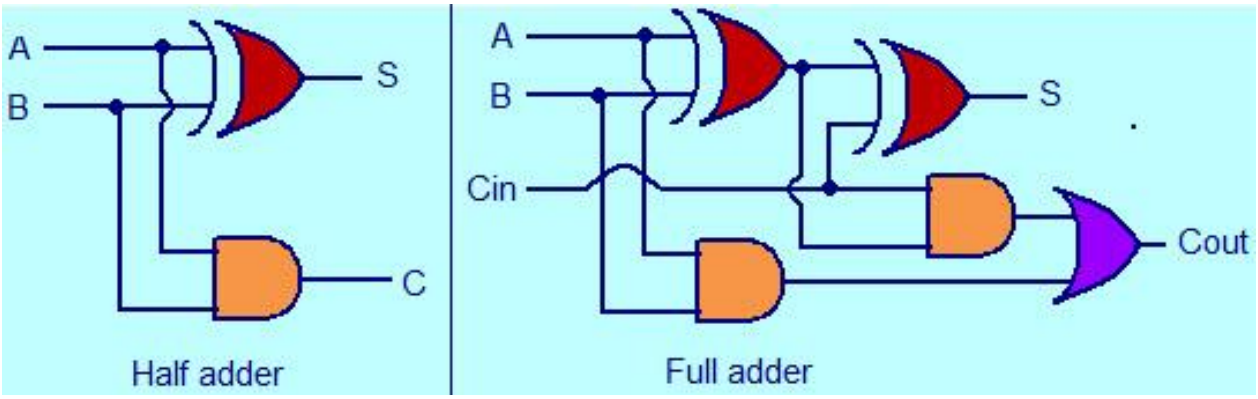
This adder is difficult to implement than a half-adder.

The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas half adder has only two inputs and two outputs.

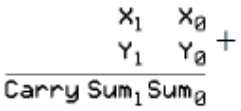


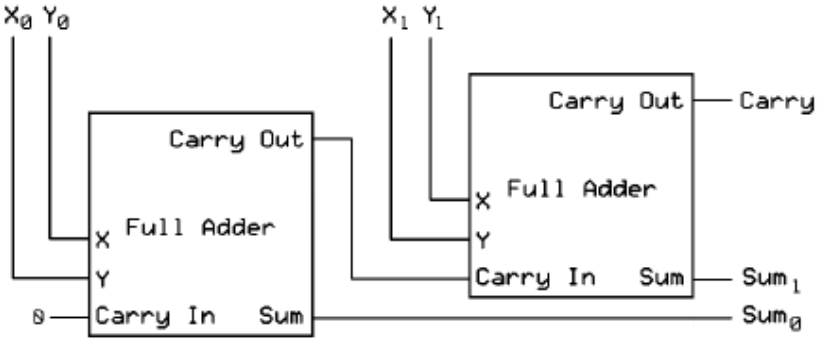
X	Y	Carry In	Carry Out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1





2-bit word adder





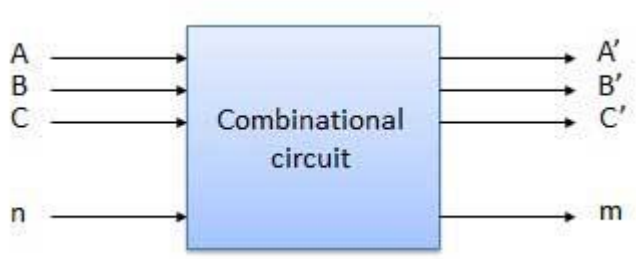
There are 2 types of digital circuits

1. Combinational circuits
2. Sequential circuits

Combinational circuit

Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are following –

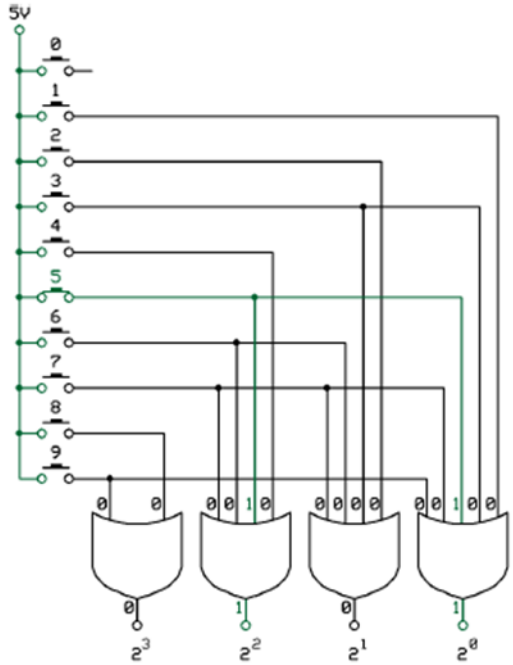
- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory.
- The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an n number of inputs and m number of outputs.



Half adder is a combinational logic circuit with two inputs and two outputs.

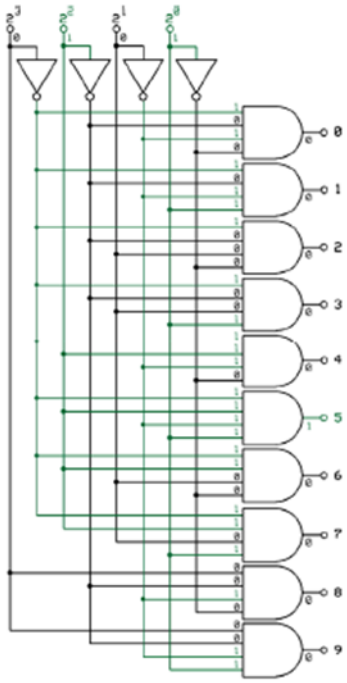
The full adder is a three input and two output combinational circuit.

D2B Encoder



Decimal Input	Binary Output			
	2^3	2^2	2^1	2^0
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

B2D Decoder



Binary Input				Decimal Output									
2^3	2^2	2^1	2^0	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0

Problem of Combinational circuits –

When user presses power button, computer requires to perform a sequence of actions such as test battery, test memory, start disk drive, power up monitor, load OS and start processor without getting any further inputs.

Solution

Sequential circuit

Sequential circuit

A sequential circuit is a logical circuit, where the output depends on the present value of the input signal as well as the sequence of past inputs.

Sequential circuits are used for storage (SRAM) and timing. Output depend s not only on the current inputs but also on the previous inputs.

Sequential circuits are based on a concept called feedback (i.e output is fed back to input)

Sequential circuit is a generalization of Flip-flops.

Flip Flops

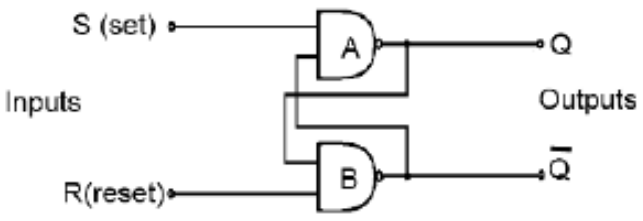
Flip flops are actually an application of logic gates. With the help of Boolean logic, you can create memory with them. Flip flops can also be considered as the most basic idea of a Random-Access Memory. When a certain input value is given to them, they will be remembered.

One form of flip-flop acts exactly like the power switch on a computer: the first time its input becomes 1, the flip flop turns on the output, and the second time its input becomes 1, the flip-flop turns off the output. That is, receiving an input of 1 causes the flip-flop to change the output from the current state to the opposite. Like a push-button switch used to control power, a flip-flop does not respond to a continuous input — the input must return to 0 before a value of 1 will cause the flip-flop to change state. Below figure shows a sequence of inputs and the resulting output.

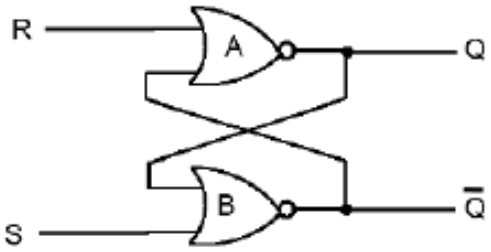
Latch Flip-flop/ R-S (Reset Set) flip flop

R-S (Reset Set) flip flop is the simplest flip flop of all and easiest to understand. It is basically a device which has two outputs one output being the inverse or complement of the other, and two inputs. A pulse on one of the inputs to take on a particular logical state. The outputs will then remain in this state until a similar pulse is applied to the other input. The two inputs are called the Set and Reset input (sometimes called the preset and clear inputs)

Such flip flop can be made simply by cross coupling two inverting gates either NAND or NOR gate could be used Figure 1(a) shows on RS flip flop using NAND gate and Figure (b)

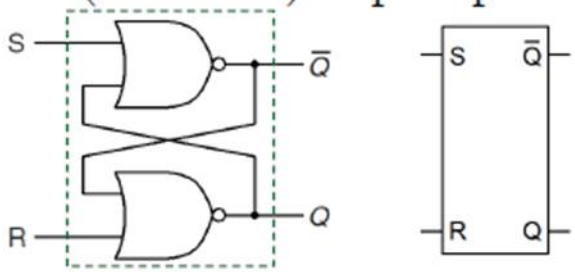


(a) Latch Flip Flop NAND Gate



(b) RS Latch Flip Flop NOR Gate

SR (Set-Reset) Flip-flop



S	R	Q_t	Q_{t+1}	State
0	0	0	0	Unchanged
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	-	Unstable
1	1	1	-	

- If (S, R) = (0, 0), output would remain unchanged
- If (S, R) = (0, 1), no matter the current output, next output would be 0
- If (S, R) = (1, 0), no matter the current output, next output would be 1

S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	-

References

Teachers' Guide 2017

https://www.technobyte.org/sequential-combinational-logic-circuits-types/

https://www.tutorialspoint.com/computer_logical_organization/sequential_circuits.htm