

# ITI1100/section A Winter 2013

---

## *DIGITAL SYSTEM I*

### *Lectures:*

Tuesday, 13:00 – 14:30 room: *STE-G0103*

Thursday, 11:30 – 13:00 room: *STE-G0103*

*Tutorial 1*-Thursday 17:30 - 19:00 DMS 1130

*Tutorial 2*- Wednesday 11:30 - 13:00 LEE A131

**LAB 1** Thursday 19:00 - 22:00 CBY B302

**LAB 2** Wednesday 14:30 - 17:30 CBY B302

**LAB 3** Friday 17:30 - 20:30 CBY B302

**Professor** : Dr. A. Karmouch, office **CBY A508**

**Mid-term exam**: Saturday March 2, 2013(10:00-11:30)

---

# Chapter 2

## Boolean Algebra and Logic Gates

# Binary Logic

---

- Binary logic deals with

- 1 - **Variables** that can take on **two** discrete **values**

→ *Values can be called **True, False, yes, no**, etc.*

- 2 - **Operations** that assume **LOGICAL** Meaning

→ *Binary logic is equivalent to **Boolean algebra***

# Boolean Algebra

---

- Basic mathematics required for the description of digital circuits
  - used to describe the different interconnections of digital circuits
  - the variables used in the Boolean algebra are **called Boolean variables**
- We will study **two-valued** Boolean algebra and functions with simplifications using basic Boolean Identities

# Two-valued Boolean Algebra

---

- It consists of

## 1- Boolean Variables

- Designated by letters of the alphabet such as A, B, C, x, y, z etc.
- Each variable **can have two and only two distinct values: 1 and 0 (True, False)**
- Can be a Function of some other Boolean variables  
( $F=ABC$ )

## 2- Boolean Operations

- There are three Basic logical operations:  
AND, OR, and NOT

# Basic Boolean Operations- AND operation

- Represented by a dot or by the absence of an operator

*Example:*  $x.y = or \quad xy=z$

*read:*  $x \text{ AND } y \text{ is equal to } z$

*Interpretation:*  $Z = 1 \text{ if and only if } x = 1 \text{ AND } y = 1$

*Otherwise  $z = 0$*

Truth table:

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

*Truth table gives the value of  $z$  for all possible values of  $x$  and  $y$*

*Don't confuse this with binary multiplication operation*

# Basic Boolean Operations- *OR* operation

---

- *Represented by a plus sign (+)*

*Example:*       $x + y = z$

*read:*       $x$  *OR*  $y$  *is equal to*  $z$

*Interpretation:*  $z = 1$  *if*  $x = 1$  *or if*  $y = 1$  *or if both*  $x = 1$  *and*  $y = 1$ .  $z = 0$  *if*  $x = 0$  *and*  $y = 0$

Truth table:

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

*Don't confuse this with binary addition operation*

*Truth table gives the value of  $z$  for all possible values for  $x$  and  $y$*

# Basic Boolean Operations- *NOT* operation

---

- *Represented by a prime or an overbar (also called complement)*

*Example:*  $x' = z$  (or  $\bar{x} = z$ )

*read:* **Not  $x$  is equal to  $z$**

*Interpretation:*  $z =$  “what  $x$  is not”

$x = 1$  then  $z = 0$ ;  $x = 0$  then  $z = 1$

Truth table:

$x$	$x'$
0	1
1	0

*Truth table gives the value of  $z$  for all possible values for  $x$*

# Binary Logic and Binary Signals

---

- For simplicity, we often still write digits instead:
  - 1 is true
  - 0 is false
- We will use this interpretation along with special operations to *design functions* and *logic circuits* for doing arbitrary computations.

# Logic Gates

---

- **Logic gates are electronic circuits that operate on one or more input signal to produce an output signal**
- Basic operations can be implemented in hardware using a Basic logic gate.
  - Symbols for each of the logic gates are shown below.
  - These gates output the **product**, **sum** or **complement** of their inputs

Logic Operation: **AND (product)**  
of two inputs

**OR (sum) of**  
two inputs

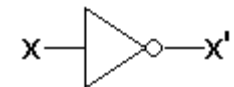
**NOT**  
(complement)  
With one input

Representation:  $x \cdot y$ , or  $xy$

$x + y$

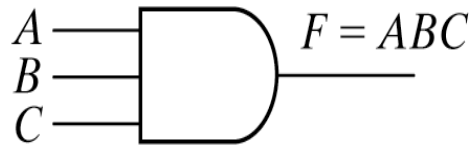
$x'$

Logic gate:

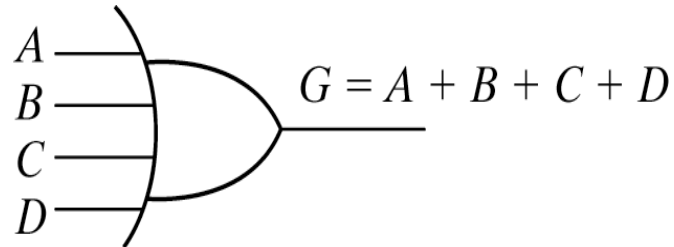


# Gates with Multiple Inputs

- AND and OR Gates may have more than 2 input signals



(a) Three-input AND gate



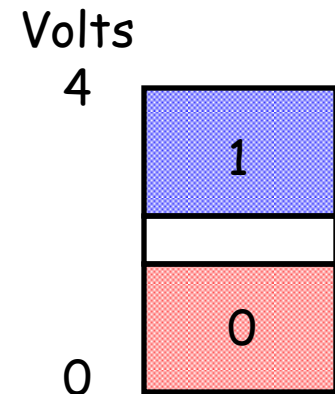
(b) Four-input OR gate

# Binary Signals

---

- Computers use voltages to represent information.
- Two voltage levels are used to represent a binary value  
“1” and “0”
- Some digital systems for example may define that:
  - Binary “0” is equal to 0 Volt
  - Binary “1” is equal to 4 Volt

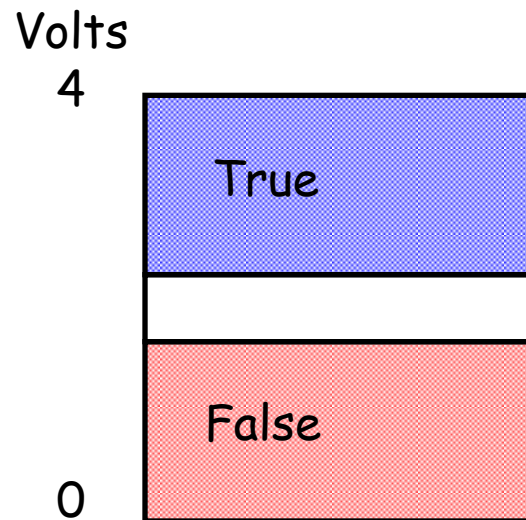
→ *It's convenient for us to translate these voltages into values 1 and 0.*



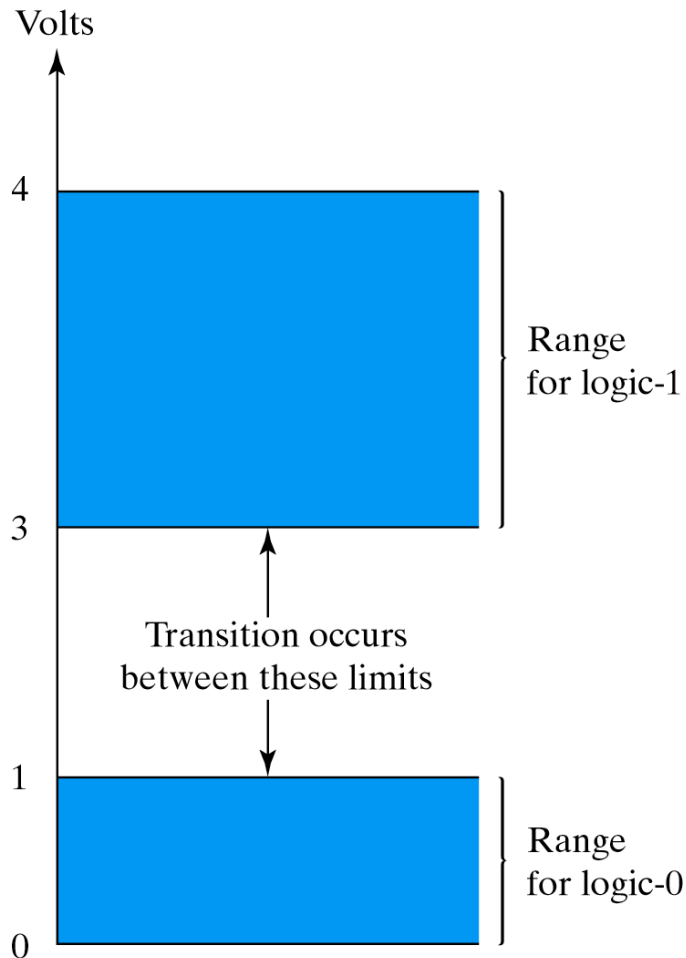
# Binary Logic and Binary Signals

---

- It's also possible to think of voltages as representing two *logical* values, *true* and *false*.
  - These logical values are called Boolean values

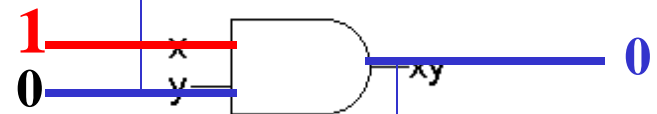


# Logic Gates - Signals



*Example*

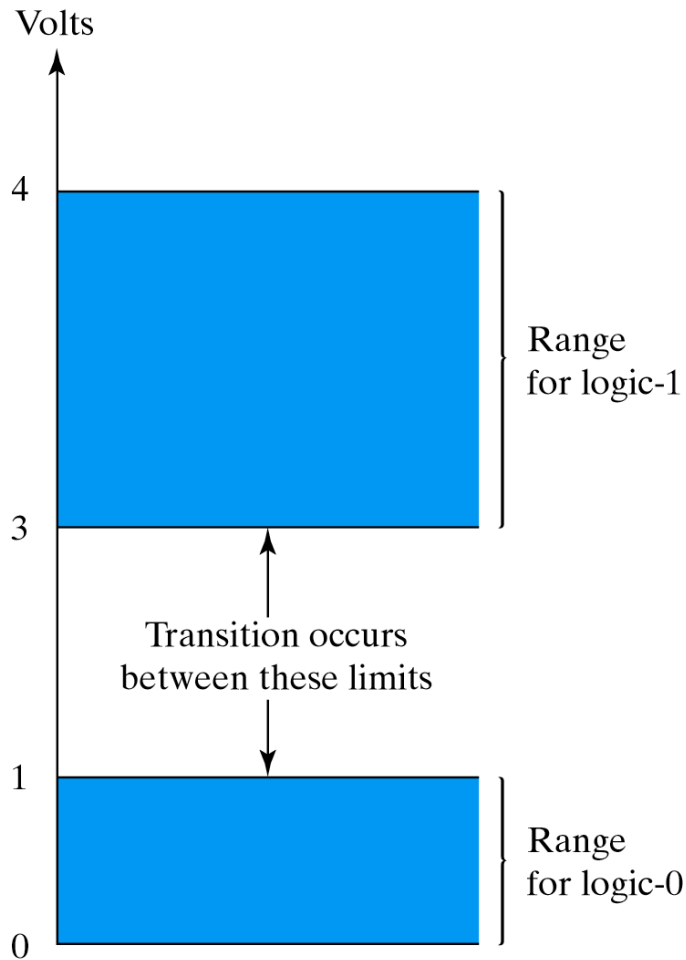
*two input signals*



*one output signal*

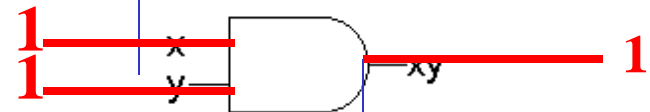
Fig. 1-3 Example of binary signals

# Logic Gates - Signals



*Example*

*2 input signals*



*1 output signal*

Fig. 1-3 Example of binary signals

# Timing Diagram –Input and output signals

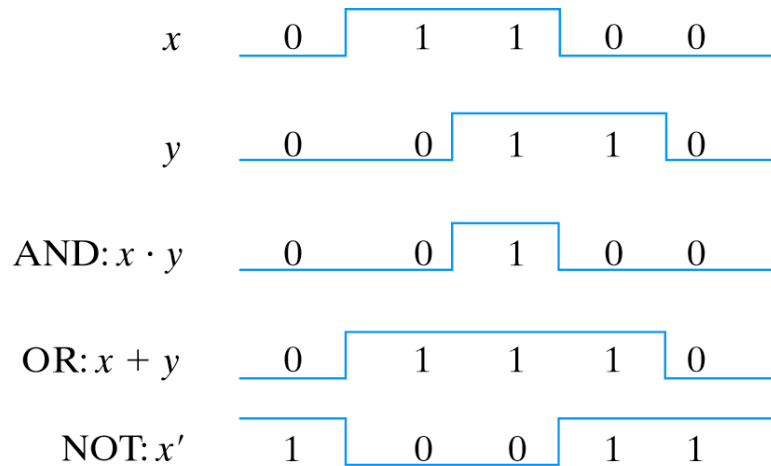
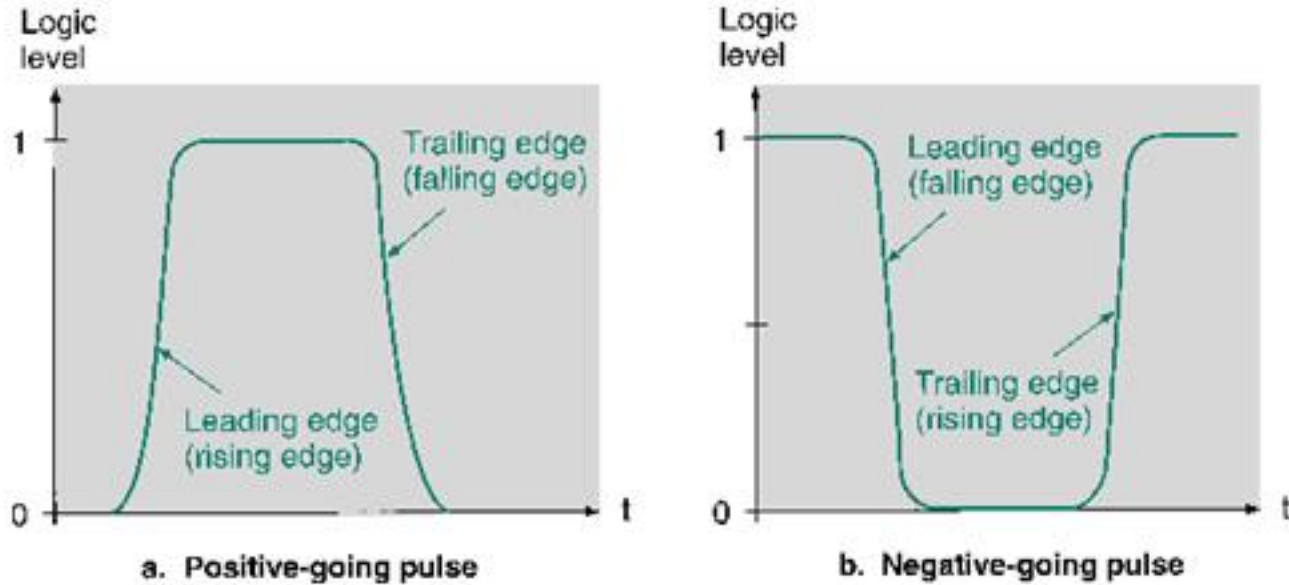


Fig. 1-5 Input-output signals for gates

# Boolean expressions (functions)

---

- **We can use the basic operations to form more complex expressions:**

$$f(x,y,z) = x y' + z x'$$

- **Some terminology and notation:**

- **f** is the name of the function.

- **Term** is an implementation with a gate: in this example f has two terms  $x y'$  and  $z x$

- $(x,y,z)$  are the **input variables**, each representing 1 or 0.

- A **literal** is any occurrence of an input variable or its complement. The function above has four literals:  $x$ ,  $y'$ ,  $z$ , and  $x'$ .

# Precedence for Evaluation of Boolean Expression

---

- Precedence are important.

- Parentheses first (if any) then

- NOT has the highest precedence, followed by AND, and then OR.

$$\rightarrow f(x,y,z) = (x + y')z + x'$$

- Fully parenthesized, the function above would be kind of messy:

$$f(x,y,z) = (((x + (y'))z) + x')$$

# Truth Table

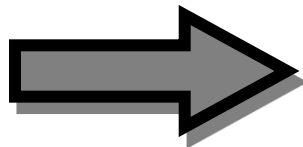
---

- A truth table shows all possible inputs and outputs of a function. Each input variable represents either 1 or 0.
  - A function with  $n$  variables has  $2^n$  possible combinations of inputs.
- Inputs are listed in binary order-example, from 000 to 111.

$$f(x,y,z) = (x + y')z + x'$$



$$\begin{aligned}f(0,0,0) &= (0 + 1)0 + 1 = 1 \\f(0,0,1) &= (0 + 1)1 + 1 = 1 \\f(0,1,0) &= (0 + 0)0 + 1 = 1 \\f(0,1,1) &= (0 + 0)1 + 1 = 1 \\f(1,0,0) &= (1 + 1)0 + 0 = 0 \\f(1,0,1) &= (1 + 1)1 + 0 = 1 \\f(1,1,0) &= (1 + 0)0 + 0 = 0 \\f(1,1,1) &= (1 + 0)1 + 0 = 1\end{aligned}$$

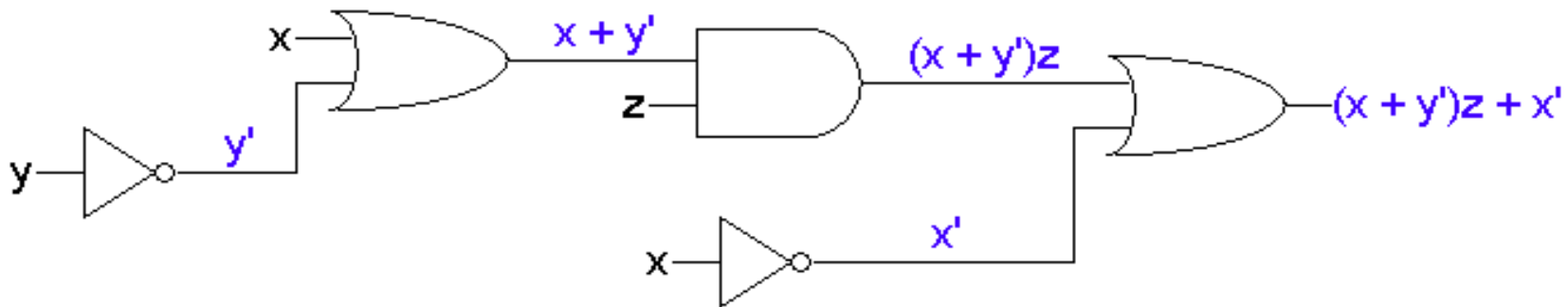


x	y	z	f(x,y,z)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

# Boolean Expression and Logic Circuits

- A Boolean expression (function) can be converted into a circuit by *combining* basic gates.
- Example:
  - The diagram below shows the inputs and outputs of each gate.
  - The precedences are explicit in a circuit.

$$f(x,y,z) = (x + y')z + x'$$



# Obtaining Boolean Expressions

---

- **Expressions may be obtained from:**
  - English language description
  - Truth table;
  - Logic circuit.

# Obtaining Boolean Expressions

- The Boolean expression (un-simplified) can be obtained from the truth table: Consider the following arbitrary Truth Table

A	B	C	F <sub>1</sub>	
0	0	0	0	
0	0	1	0	
0	1	0	1	$A'BC'$
0	1	1	1	$A'BC$
1	0	0	1	$AB'C'$
1	0	1	1	$AB'C$
1	1	0	1	$ABC'$
1	1	1	1	$ABC$

We can also write the function as:

$$F_1(A,B,C) = A'BC' + A'BC + AB'C' + AB'C + ABC' + ABC$$

# Obtaining Boolean Expressions

---

## *Using the false terms in the truth table*

- Sometimes it is easier to work with the terms that describe when the function is false.
- For example, if a function has four variables then there are sixteen possible states.
  - If for instance thirteen out of sixteen were true, then only three out of sixteen are false. **Fewer terms makes it easier**
- In our example, two out of eight are false.

# Obtaining Boolean Expressions

- The Boolean expression (un-simplified) can be obtained from the truth table using **false terms**

So we can also write the function **NOT**  $F_1$  as:

$$\overline{F_1}(A,B,C) = A'B'C' + A'B'C$$

A	B	C	$F_1$	
0	0	0	0	$A'B'C'$
0	0	1	0	$A'B'C$
0	1	0	1	
0	1	1	1	
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	1	

# Obtaining The Truth Table - Example

---

Design a digital circuit that will be used to control an Alarm bell. This Alarm bell is to be installed in a room to protect it from unauthorized entry.

Sensor devices provide the following logic signals

**C = 1** The control system is active

**D = 1** The room door is closed

**M = 1** There is a motion in the room

**Q = 1** The room is open to the public

i) Obtain the truth table

ii) derive the Boolean expression using **true** terms

# Truth table

---

C	D	M	Q	Alarm
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	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

$$\text{Alarm} = \text{CD}'\text{M}'\text{Q}' + \text{CD}'\text{M}\text{Q}' + \text{CD}\text{M}\text{Q}'$$

Door should not be open

room is closed to the public → door open + motion

room is closed to the public + door closed → motion

# Boolean Identities

---

- Boolean algebra is used in digital design to **reduce any logical function (expression) to its simplest form**
  - the minimization of the number of literals and the number of terms
  - a circuit with less equipment
- **It is a hard problem (no specific rules to follow)**

# Boolean Identities

---

1.  $x + 0 = x$
2.  $x \cdot 1 = x$
3.  $x + x = 1$
4.  $x \cdot x = 0$
5.  $x + x = x$
6.  $x \cdot x = x$
7.  $x + 1 = 1$
8.  $x \cdot 0 = 0$
9.  $(x) = x$

Basic to Boolean algebra

- 
- |                                 |              |
|---------------------------------|--------------|
| 10. $x + y = y + x$             | Commutative  |
| 11. $xy = yx$                   | Commutative  |
| 12. $x + (y + z) = (x + y) + z$ | Associative  |
| 13. $x(yz) = (xy)z$             | Associative  |
| 14. $x(y + z) = xy + xz$        | Distributive |
| 15. $x + yz = (x+y)(x+z)$       | Associative  |
- 
16.  $(x + y)' = x'y'$  DeMorgan
  17.  $(xy)' = x' + y'$  DeMorgan
  18.  $x + xy = x$  Absorption
  19.  $x(x + y) = x$  Absorption

# Verifying Boolean Identities-Examples

**Theorem :  $x+x = x$**

$$\begin{aligned}x+x &= (x+x) 1 \\ &= (x+x) (x+x') \\ &= x+xx' \\ &= x+0 \\ &= x\end{aligned}$$

$$x.1=x$$

$$x+x'=1$$

$$x+yz = (x+y)(x+z)$$

$$x.x'=0$$

$$x+0=x$$

**Theorem :  $x x = x$**

$$\begin{aligned}xx &= x x + 0 \\ &= xx + xx' \\ &= x (x + x') \\ &= x 1 \\ &= x\end{aligned}$$

# Verifying Boolean Identities-Examples

- DeMorgan's Theorems

$$\rightarrow (x+y)' = x' y'$$

$$\rightarrow (x y)' = x' + y'$$

- By means of truth table

x	y	x+y	$(x+y)'$	$x'$	$y'$	$x' y'$
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

# Verifying Boolean Identities-Examples

---

- **Theorem**  $x + xy = x$

By means of truth table

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

# Simplifying Boolean Expressions

---

→ Use the Rules of Boolean Algebra

We can simplify the function as:

$$\begin{aligned}F_1(A,B,C) &= A'BC' + A'BC + AB'C' + AB'C + ABC' + ABC \\&= A'B(C'+C) + AB'(C+C') + AB(C'+C) \\&= A'B + AB' + AB \\&= A'B + A(B'+B) \\&= A + A'B\end{aligned}$$

# Simplifying Boolean Expressions

---

## Function with four variables

- Giving the following function:

$$\begin{aligned}F_{2a}(A,B,C,D) &= (AB'(C + BD) + A'B')C \\&= (AB'C + \mathbf{AB'BD} + A'B')C \\&= (AB'C + \mathbf{A0D} + A'B')C \\&= (AB'C + \mathbf{0} + A'B')C \\&= (AB'C + A'B')C \\&= AB'\mathbf{CC} + A'B'C \\&= AB'\mathbf{C} + A'B'C \\&= (\mathbf{A+A'})B'C \\&= B'C\end{aligned}$$

$$F_{2b}(A,B,C,D) = B'C$$

→ the two expressions are equivalent!

→  $F_{2a}$  requires more logic gates than  $F_{2b}$

# Basic and Other Logic gates

---

## • Basic Logic gate

- AND
- OR
- NOT



These are called “fundamental logic gates” as all other gates and digital Circuits can be created from these gates.

## • Other Logic gates

- NAND
- NOR



These are called “Universal logic gates” as any digital circuit can be designed by just using these gates

- XOR
- XNOR

# The NAND & NOR Gates

---

- We can use a NAND and NOR gates to implement all three of the *basic operations* (AND,OR,NOT).

→ They are said to be **functionally complete**

→ Both NAND and NOR gates are very valuable as any design can be realized using either one.

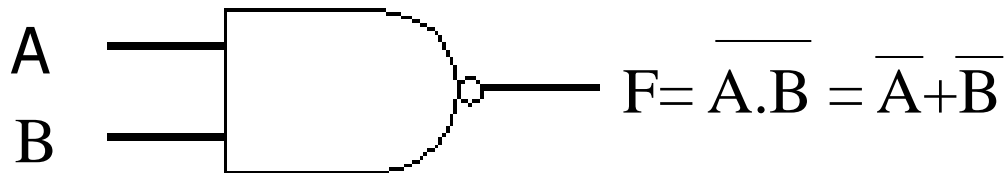
- It is easier to build digital circuits using all NAND or NOR gates than to combine AND,OR, and NOT gates.

- NAND/NOR gates are typically faster and cheaper to produce.

# The NAND Gate

---

- The NAND gate is a combination of an AND gate followed by an inverter (NOT gate).
- We can use a NAND gate to implement all three of the *basic operations* (AND,OR,NOT).
- Such a gate is said to be **functionally complete**.

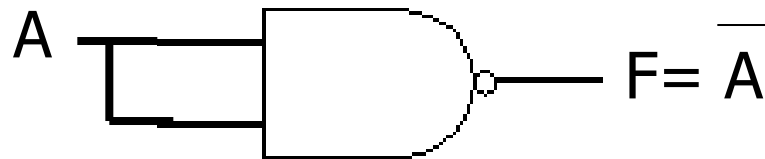


A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

# The NAND Gate

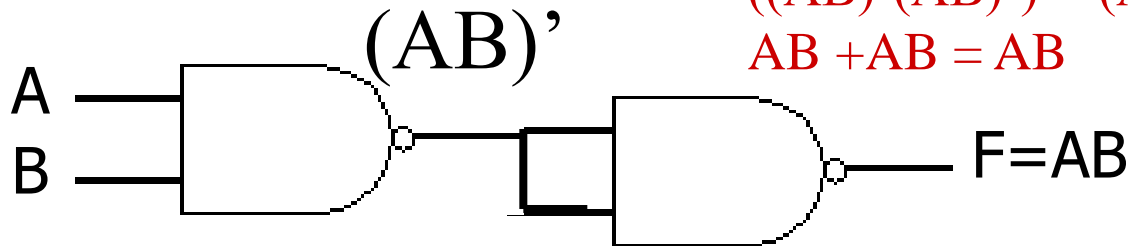
---

→ a NAND gate with both of its inputs driven by the same signal is equivalent to a NOT gate



NOT Gate

→ a NAND gate whose output is complemented is equivalent to an AND gate



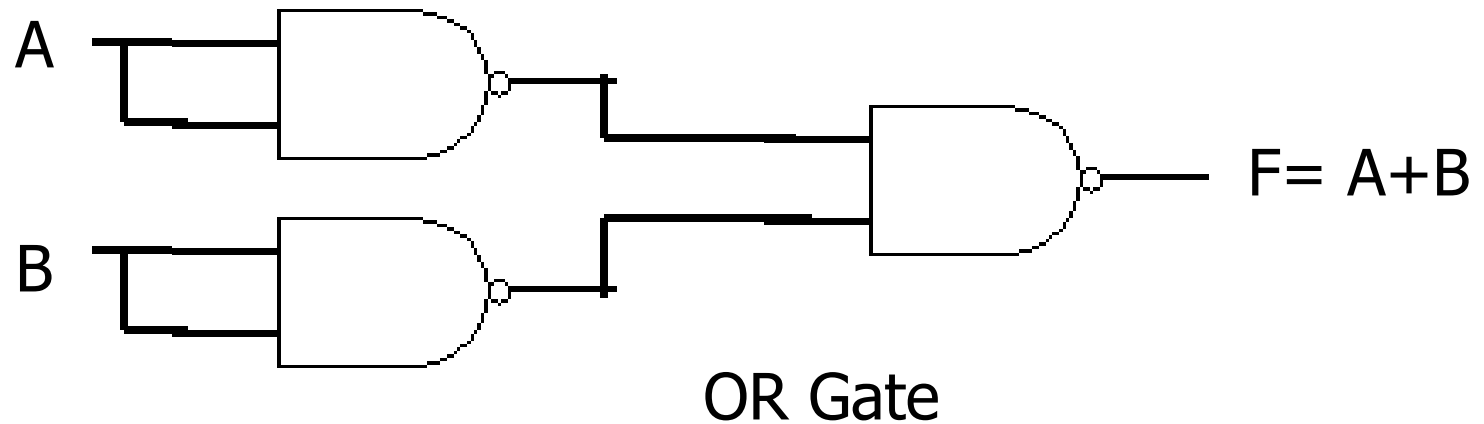
$$\begin{aligned} ((AB)'(AB)')' &= (AB)'' + (AB)'' = \\ &= AB + AB = AB \end{aligned}$$

AND Gate

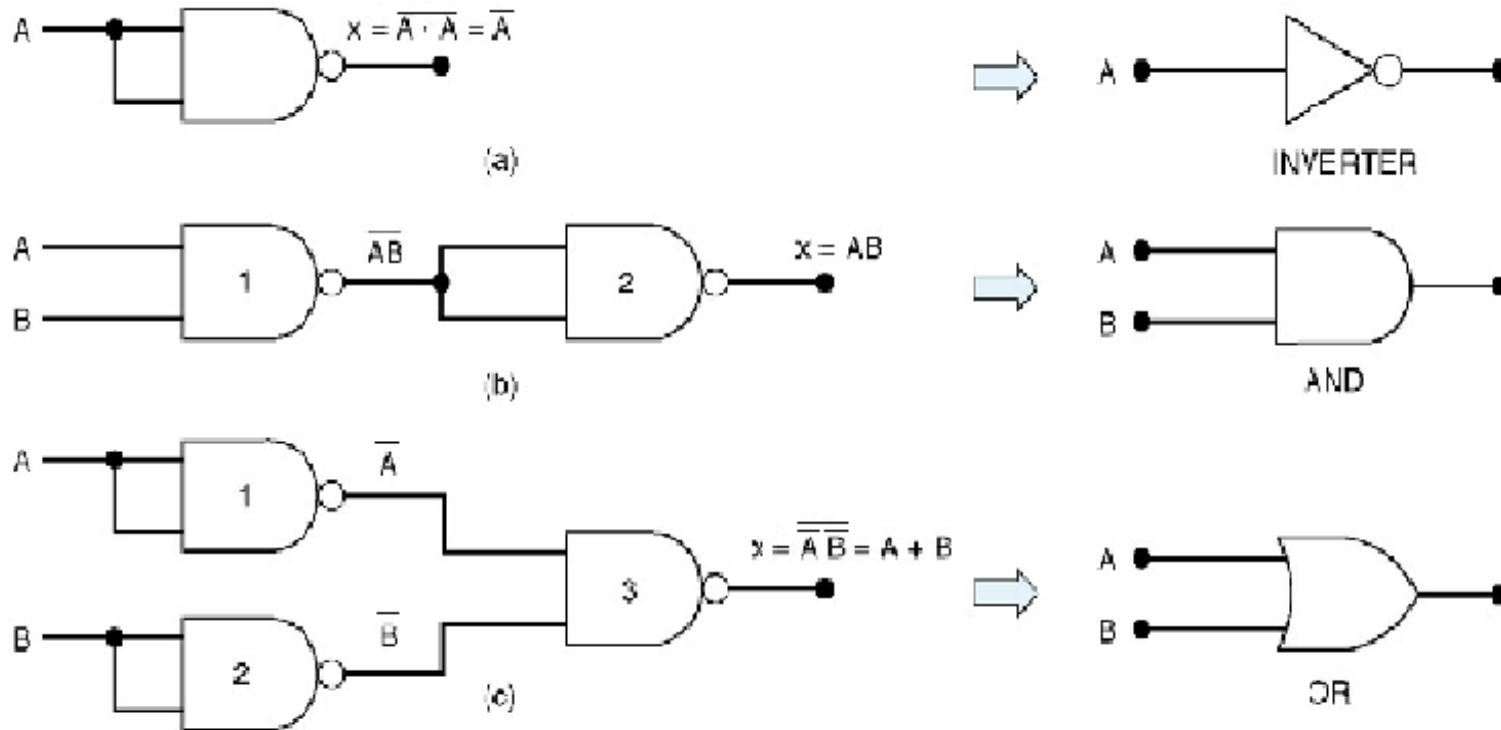
# The NAND Gate

---

- a NAND gate with complemented inputs acts as an OR gate.



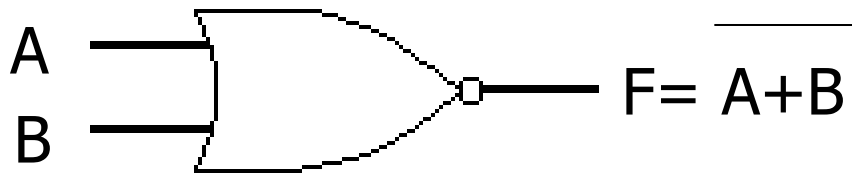
# Universality of NAND



# The NOR Gate

---

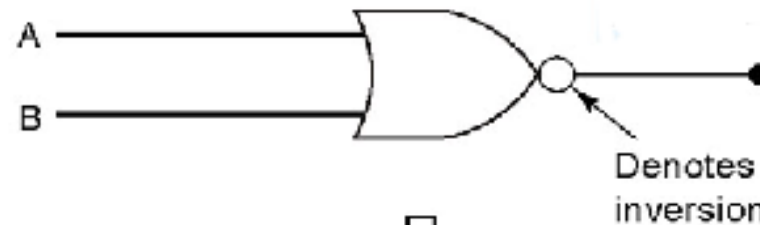
- This is a NOR gate. It is a combination of an OR gate followed by an inverter.
- like the NAND gate, the NOR gate is **functionally complete** → any logic function can be implemented using just NOR gates.



A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

# NOR Gate Equivalence

- NOR Symbol, Equivalent Circuit, Truth Table



(a) ↓



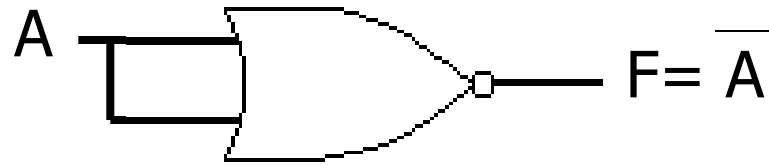
(b)

A	B	OR		NOR	
		A + B	$\overline{A + B}$		
0	0	0	1		
0	1	1	0		
1	0	1	0		
1	1	1	0		

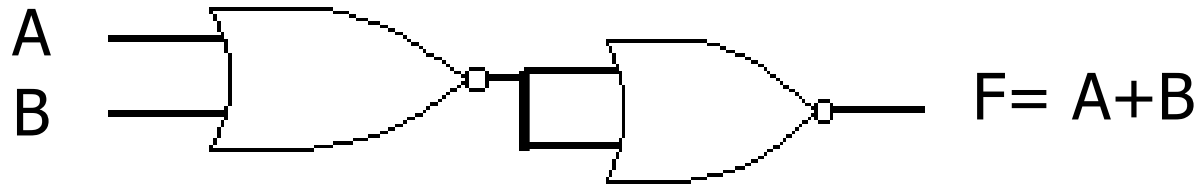
(c)

# NOR Gates-functionally complete

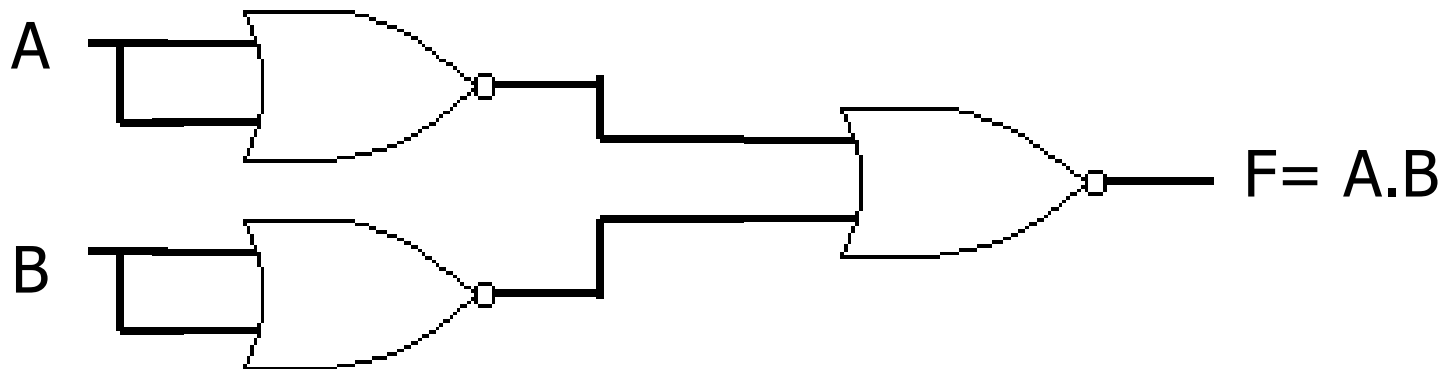
---



NOT Gate



OR Gate

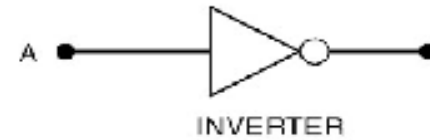


AND Gate

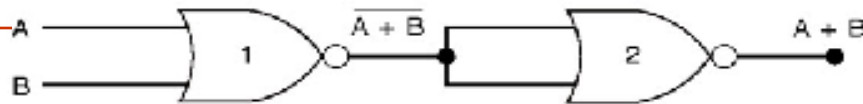
# Universality of NOR gate



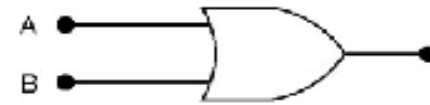
(a)



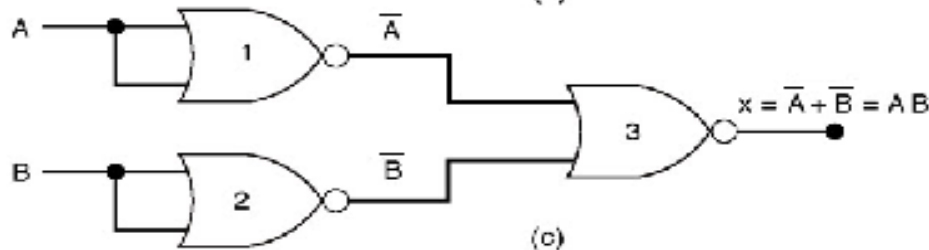
INVERTER



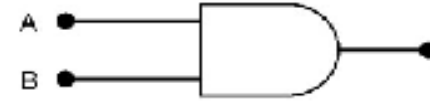
(b)



OR



(c)



AND

- Equivalent representations of the AND, OR, and NOT gates

$$(A+A)' = A'A' = A'$$

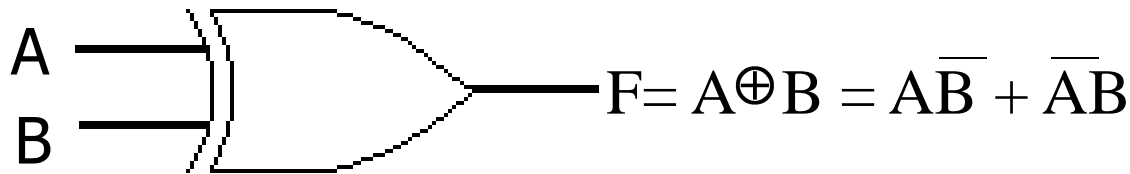
$$((A+B)' + (A+B)')' = (A+B)''(A+B)'' = (A'B')'(A'B')'$$

$$= (A''+B'')(A''+B'') = (A+B)(A+B) = (A+B)$$

# The XOR Gate (Exclusive-OR)

---

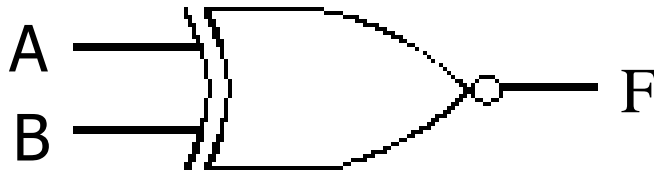
- This is a XOR gate.
- XOR gates assert their output when exactly one of the inputs is asserted, hence the name.
- The operator symbol for this operation is  $\oplus$   
 $1 \oplus 1 = 0$  and  $1 \oplus 0 = 1$ .



<b>A</b>	<b>B</b>	<b>F</b>
0	0	0
0	1	1
1	0	1
1	1	0

# The XNOR Gate

- This functions as an exclusive-NOR gate, or simply the complement of the XOR gate.
- The symbol for this operation is  $\odot$   
 $1 \odot 1 = 1$  and  $1 \odot 0 = 0$ .



$$F = \overline{A \oplus B} = \overline{AB} + \overline{A \cdot B} = AB + A'B'$$

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

# Standard Forms

---

- We have seen how to interpret truth tables, obtain Boolean expressions (functions) then build logic circuits.
  - We have simplified Boolean expressions using Boolean algebra.
- There is a “standard” way of writing Boolean expressions (Functions):**
- **The standard Sum of Products (SOP)**
  - **The standard Product of Sums (POS)**

# The standard Sum of Product-Minterms

- A Minterm is one in which all variables appear (only) once.
- Each Minterm represents exactly one combination (row) in truth table.
- $n$  variables give  $2^n$  Minterms.

Truth Table

Decimalvalue	A	B	C	F	Minterm
0	0	0	0	0	$m_0$
1	0	0	1	0	$m_1$
2	0	1	0	1	$m_2$
3	0	1	1	1	$m_3$
4	1	0	0	1	$m_4$
5	1	0	1	1	$m_5$
6	1	1	0	1	$m_6$
7	1	1	1	1	$m_7$

# The standard Sum of Products-Function

---

- SOP are expressions of the form:

$$F(A,B,C, \dots) = (\dots) + (\dots) + (\dots) + \dots$$

- Brackets can contain single or multiple variables
- Such expressions can be implemented using:

$$F(A,B,C, \dots) = (\text{AND's}) \text{ OR } (\text{AND's}) \text{ OR } (\text{AND's}) \text{ OR } \dots$$

# The standard Sum of Product-Function

- SOP form not unique, and doesn't necessarily contain all variables, for example:

$$F(A,B,C) = A'B'C' + A'BC + C'A'B + C'AB' + BAC + BAC$$

and  $F(A,B,C) = B + B'C'$

**are both valid SOP expressions.**

# The standard Sum of Product-Function

- We Can obtain SOP from truth table (below)

$$F(A,B,C) = A'BC' + A'BC + AB'C' + AB'C + ABC' + ABC$$

A simpler notation is to write as

$$F(A,B,C) = m_2 + m_3 + m_4 + m_5 + m_6 + m_7$$

$$= \sum m_i (2, 3, 4, 5, 6, 7)$$

Decimal value	A	B	C	F	Minterm	
0	0	0	0	0	$m_0$	$A'B'C'$
1	0	0	1	0	$m_1$	$A'B'C$
2	0	1	0	1	$m_2$	$A'BC'$
3	0	1	1	1	$m_3$	$A'BC$
4	1	0	0	1	$m_4$	$AB'C'$
5	1	0	1	1	$m_5$	$AB'C$
6	1	1	0	1	$m_6$	$ABC'$
7	1	1	1	1	$m_7$	$ABC$

# Product of Sums: Function

• From truth table we have

$$F(A,B,C) = (A'B'C' + A'B'C)$$

• Therefore we obtain  $F$  from  $\bar{F}$ :

$$\begin{aligned} F(A,B,C) &= \overline{\overline{F(A,B,C)}} = \overline{(A'B'C' + A'B'C)} \\ &= (A' + B'' + C'') \cdot (A'' + B'' + C') = (A + B + C) \cdot (A + B + C') \end{aligned}$$

**Form Compact**  $F = M_0 \cdot M_1 = \prod M_i (0, 1)$

Truth Table

Decimal value	A	B	C	F	Minterm	Maxterm	$M_i = \bar{m}_i$
0	0	0	0	0	$m_0$	$M_0$	$A+B+C$
1	0	0	1	0	$m_1$	$M_1$	$A+B+C'$
2	0	1	0	1	$m_2$	$M_2$	$A+B'+C$
3	0	1	1	1	$m_3$	$M_3$	$A+B'+C'$
4	1	0	0	1	$m_4$	$M_4$	$A'+B+C$
5	1	0	1	1	$m_5$	$M_5$	$A'+B+C'$
6	1	1	0	1	$m_6$	$M_6$	$A'+B'+C$
7	1	1	1	1	$m_7$	$M_7$	$A'+B'+C'$

# Obtain SOP and POS from a given expression

---

- Given an arbitrary Boolean expression
- Work out number of terms ( $2^n$ ) for n inputs.
- Generate truth table and identify terms for which the function is true - the Minterms.

- Write function as: 
$$F = \sum_{i=0}^{n-1} m_i$$

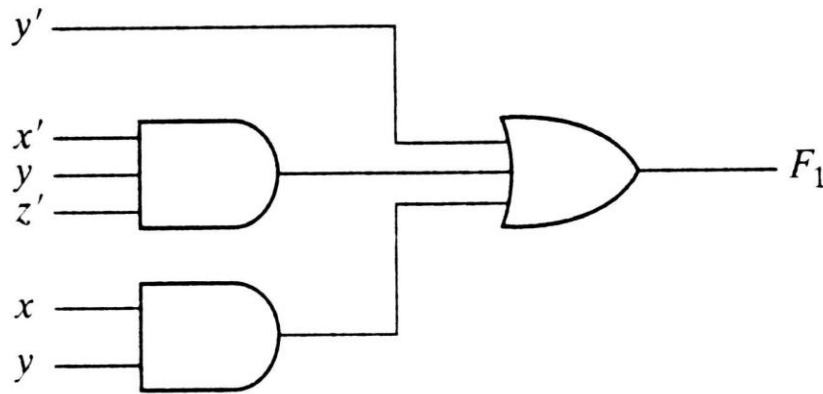
- Alternatively, identify terms for which the function is false and use a Maxterm description.
- Write function :

$$F = \prod_{j=0}^{n-1} M_j \quad j \neq i$$

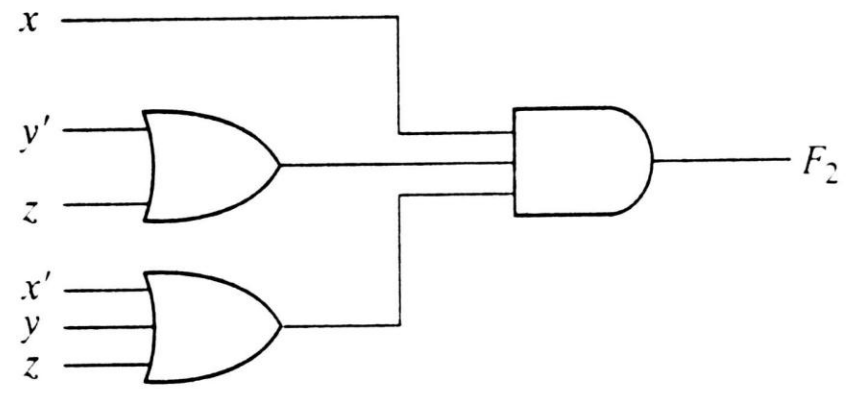
# SOP & POS Implementation using AND and OR

---

- Two-level implementation

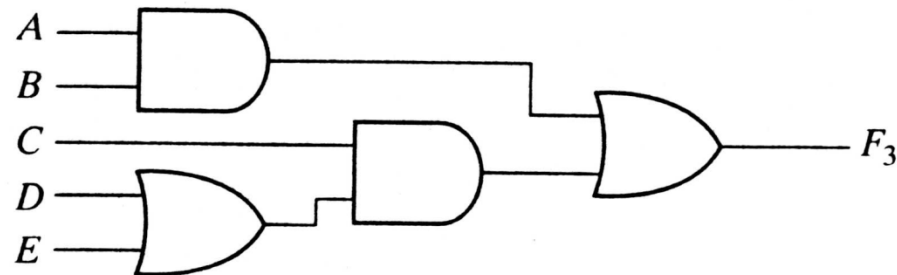


(a) Sum of Products



(b) Product of Sums

- Multi-level implementation



(a)  $AB + C(D + E)$

---

## Truth Table

<b>A</b>	<b>B</b>	<b>C</b>	<b>C<sub>o</sub></b>	<b>S</b>
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

# Examples

---

1- Half Adder

2- Full Adder

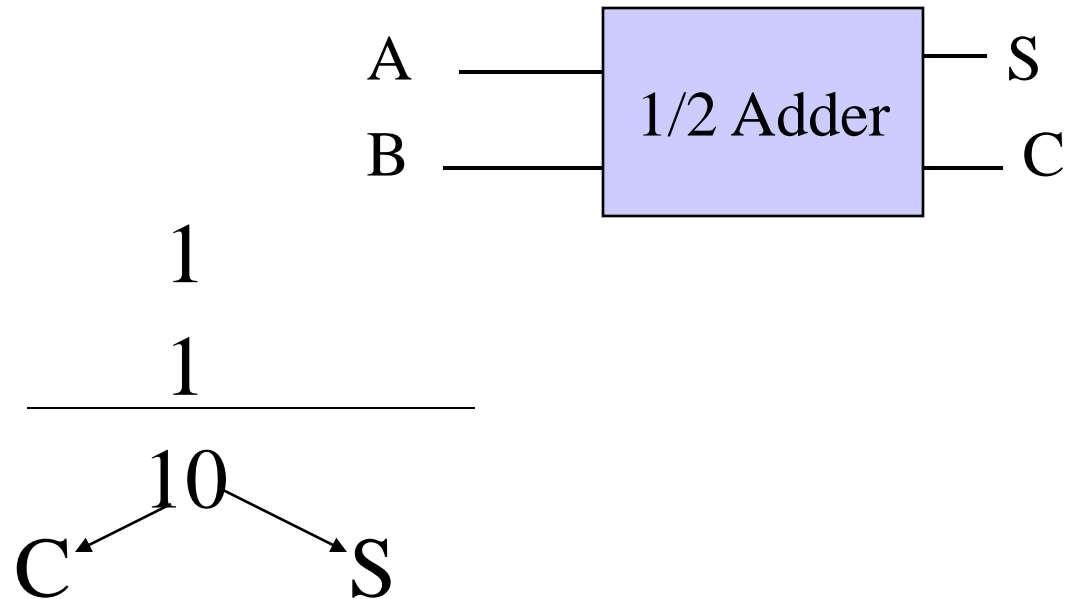
3- Deriving SOP and POS from a truth table

4- Simplifying SOP & POS using Boolean identities

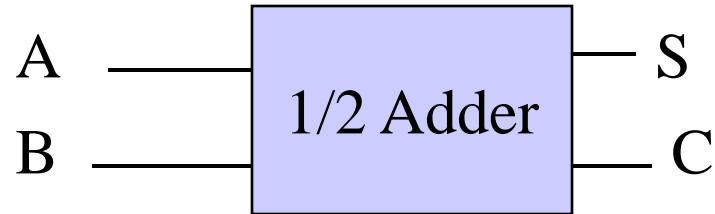
# Half Adder

→ The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs: a sum bit and a carry bit.

Truth Table			
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



# Half-Adder



## Truth Table

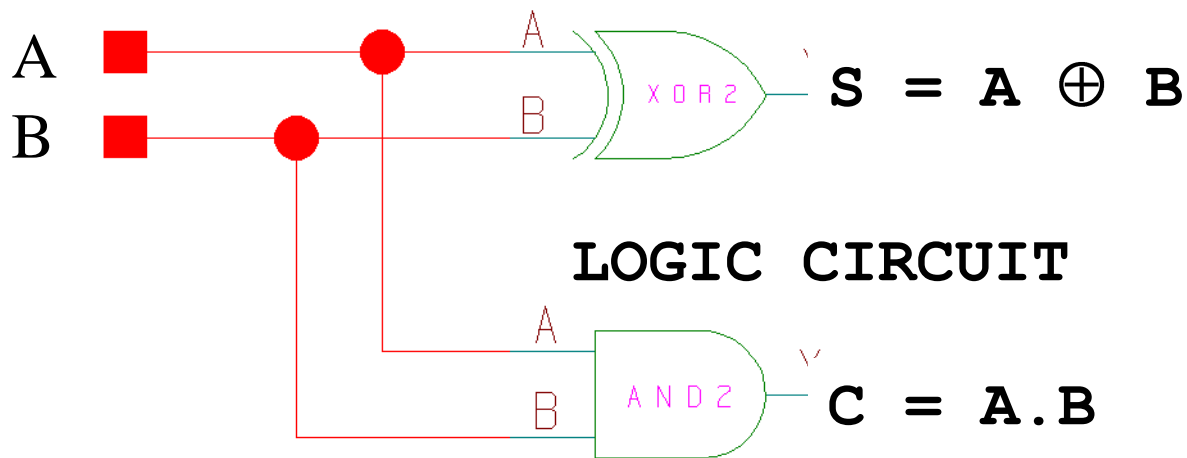
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

## Logic Function

$$S = A'B + AB'$$

$$S = A \oplus B$$

$$C = A.B$$



# Full Adder

**Truth Table**

<b>A</b>	<b>B</b>	<b>C</b>	<b>C<sub>o</sub></b>	<b>S</b>
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

→ The Full-adder accepts two input bits and an input carry and generates a sum output and an output carry

→ Basic difference between a full and a half adder is that the full adder **accepts an input carry**



# Full-Adder

## Logic FUNCTION

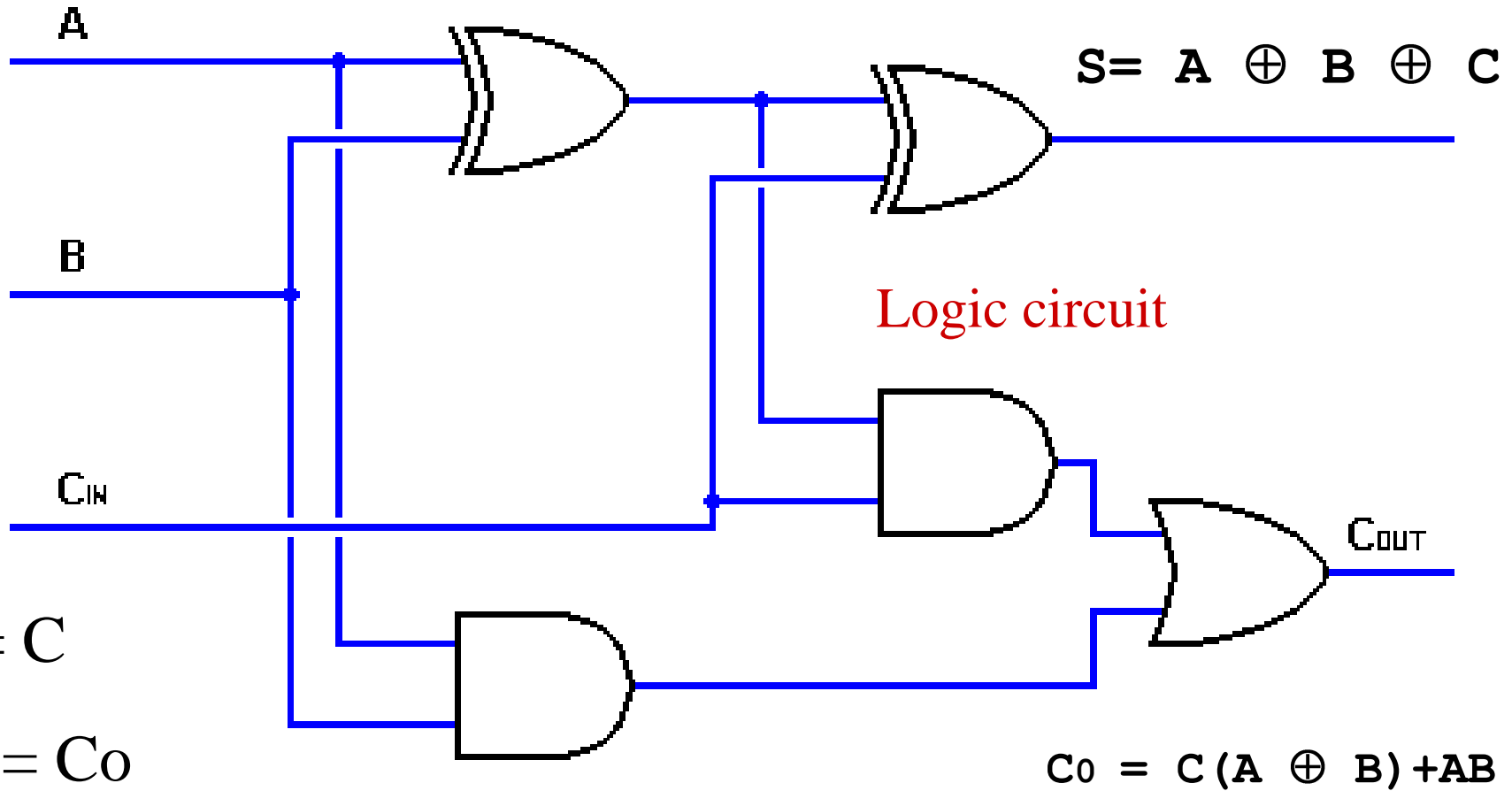
Truth Table

A	B	C	C <sub>o</sub>	S
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

$$\begin{aligned}C_o &= A'BC + AB'C + ABC' + ABC \\&= C[A'B + AB'] + AB[C' + C] \\&= C[A \oplus B] + AB.1 \\&= C(A \oplus B) + AB\end{aligned}$$

$$\begin{aligned}S &= A'B'C + A'BC' + AB'C' + ABC \\&= A'[B'C + BC'] + A[B'C' + BC] \\&= A'[B \oplus C] + A[B \oplus C]' \\&= A' \overset{[X]}{X} + A \overset{[X]'}{x'} \\&= A \oplus X \\&= A \oplus B \oplus C\end{aligned}$$

# Full Adder



# Two bit parallel adder

---

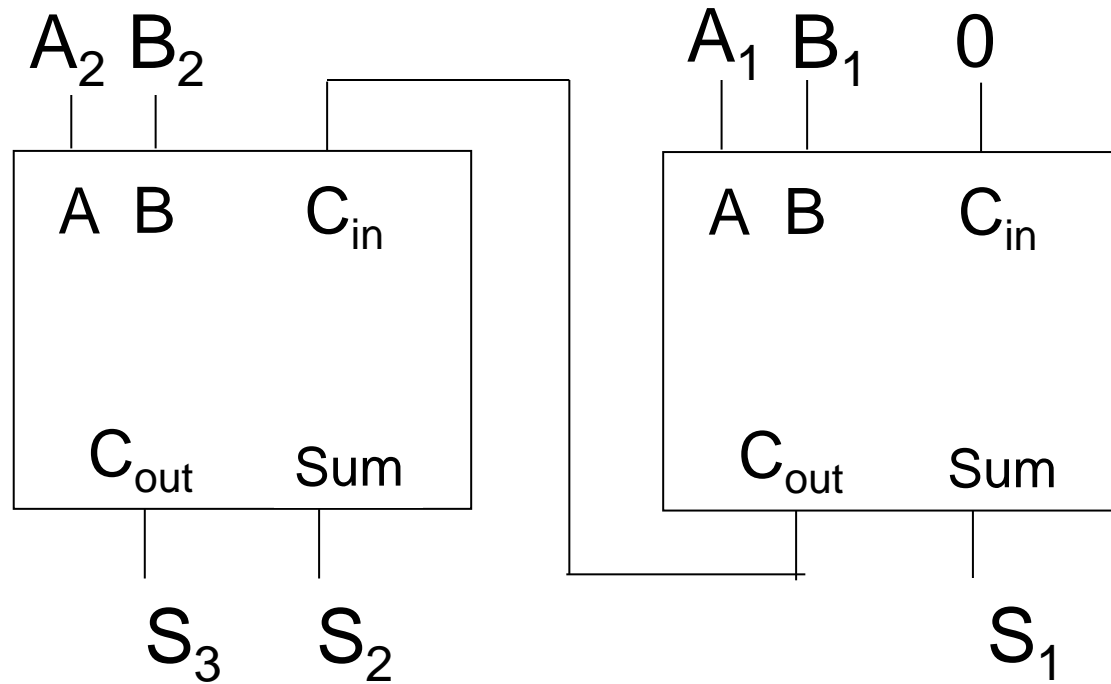
$$\begin{array}{r} \mathbf{1} \leftarrow \text{Carry bit from right column} \\ 1\ 1 \\ +\ 0\ 1 \\ \hline \end{array}$$

$$\mathbf{1}\ 0\ 0$$

Carry bit from second column  
becomes a sum bit

# Two bit parallel adder

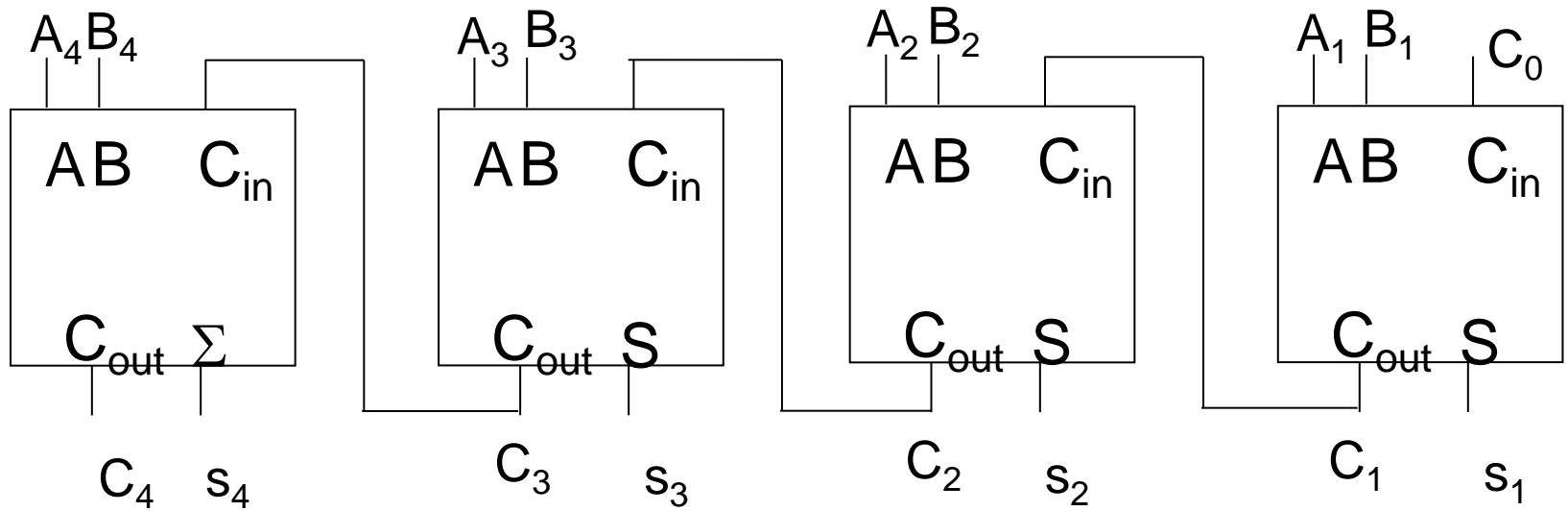
---



$$\begin{array}{r} A_2 A_1 \\ + B_2 B_1 \\ \hline S_3 S_2 S_1 \end{array}$$

# Four bit parallel adder

---



# Overflow Examples (review from chapter 1)

---

- In a 6-bit register

$$+ 17 = \quad 010001$$

$$+ 16 = \quad +\underline{010000}$$

$$= 100001 \quad \rightarrow \text{Overflow}$$

- **100001 = 2's : - (11111) = -(31)<sub>10</sub> instead of + (33)<sub>10</sub>**
- Same with a 7-bit register

$$+ 17 = \quad 0\ 010001$$

$$+ 16 = \quad +\underline{0\ 010000}$$

$$= 0\ 100001$$

$$\mathbf{0100001 = + 33 \text{ No Overflow}}$$

# Four bit parallel adder

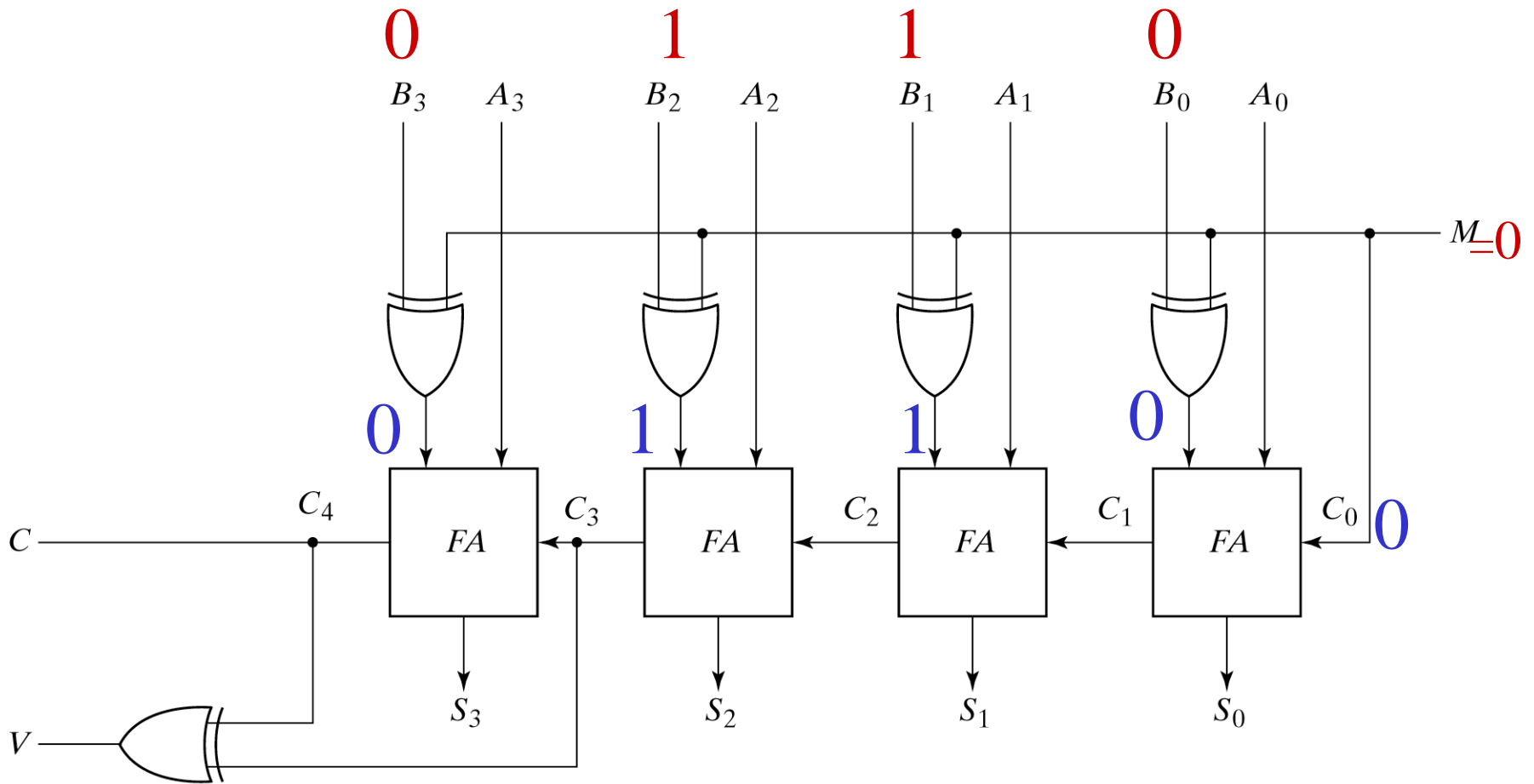
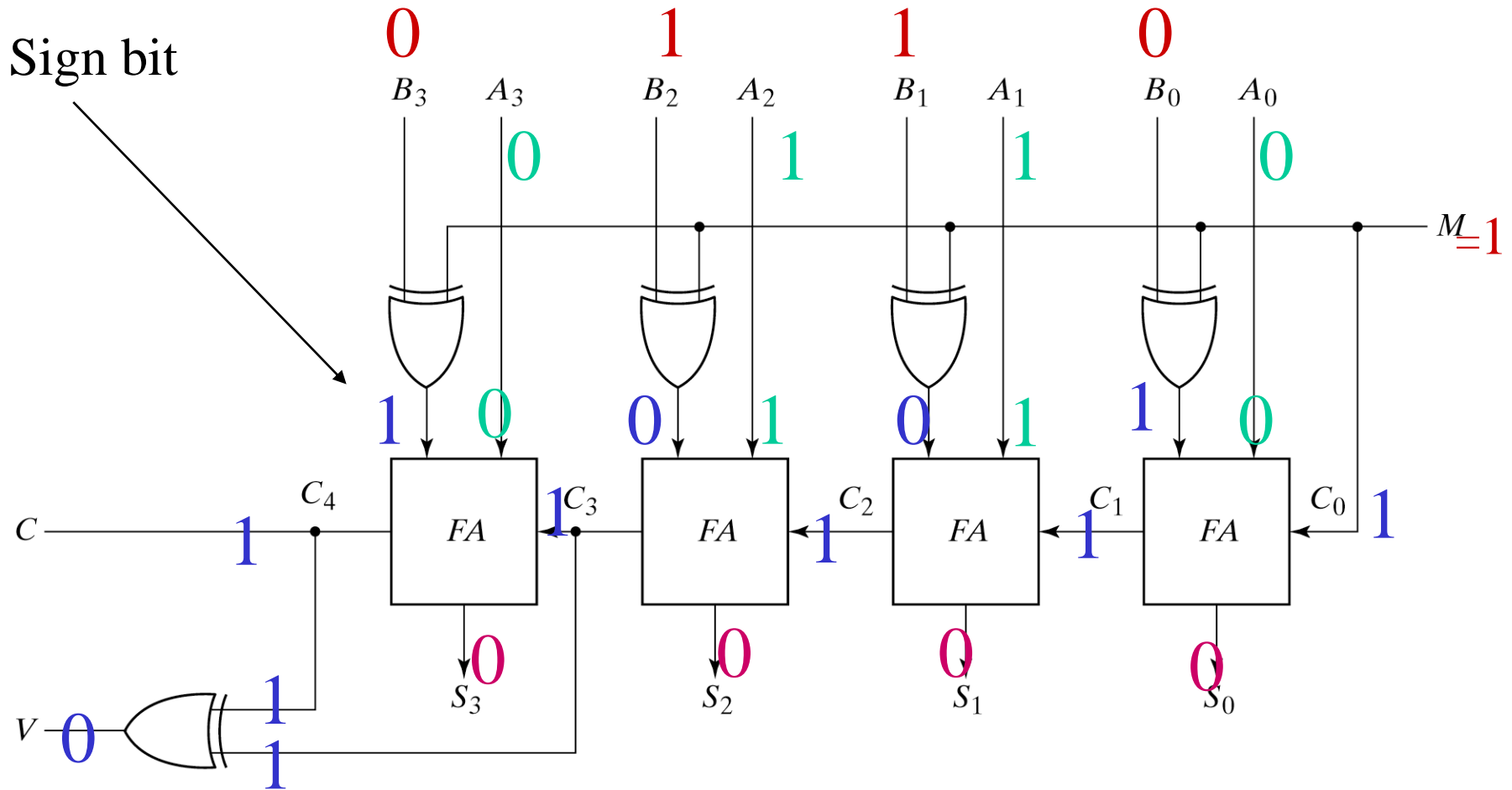


Fig. 4-13 4-Bit Adder Subtractor





# Four bit parallel adder: more examples



**NO** Overflow

Fig. 4-13 4-Bit Adder Subtractor

# SOP & POS Standard Forms- Example

---

From an arbitrary Truth table (next slide)

- **Part one**

- 1- obtain SOP representation for F

- 2- obtain the two level implementation for F without simplification

- 3- simplify F using Boolean identities

- 4- obtain two level-implementation for F

- 5- compare the design obtained in question 4 with the one of question 2

- **Part two**

Repeat part one using POS

# Deriving SOP and POS from a truth table

---

Consider the following arbitrary Truth Table

i) SOP

ii) POS

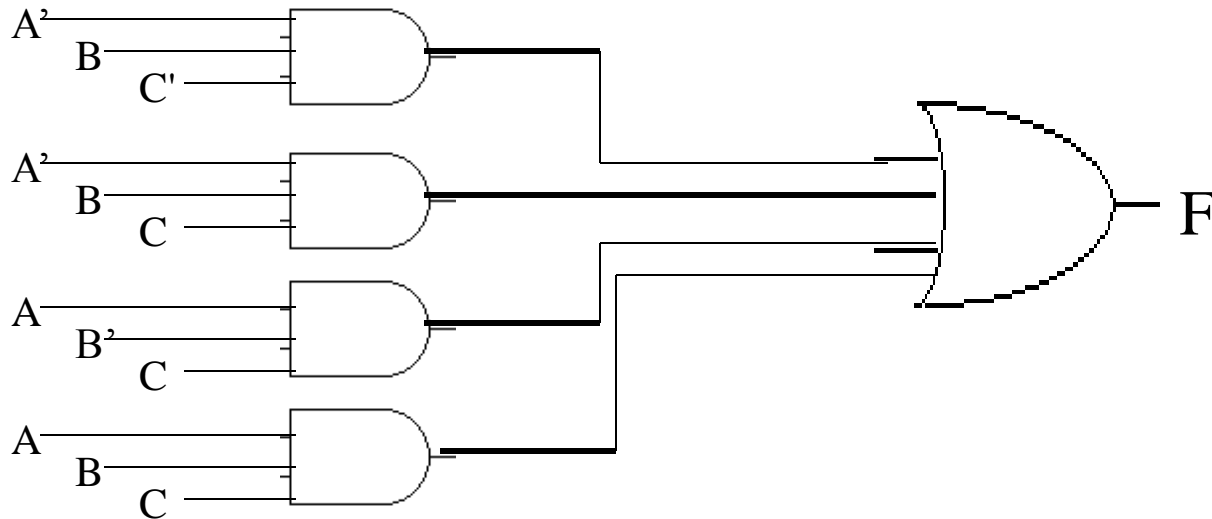
i	A	B	C	F	Minterms
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	1	$\rightarrow m_2 = A'BC'$
3	0	1	1	1	$\rightarrow m_3 = A'BC$
4	1	0	0	0	
5	1	0	1	1	$\rightarrow m_5 = AB'C$
6	1	1	0	0	
7	1	1	1	1	$\rightarrow m_7 = ABC$

# Deriving SOP from a truth table

## 1- Expression sum of products

$$\begin{aligned} \text{a) } F &= m_2 + m_3 + m_5 + m_7 \\ &= A'BC' + A'BC + AB'C + ABC \end{aligned}$$

b) Implementation with logic gates (unsimplified )



Two level-Implementation

# Simplifying SOP using Boolean identities

## c) Simplification

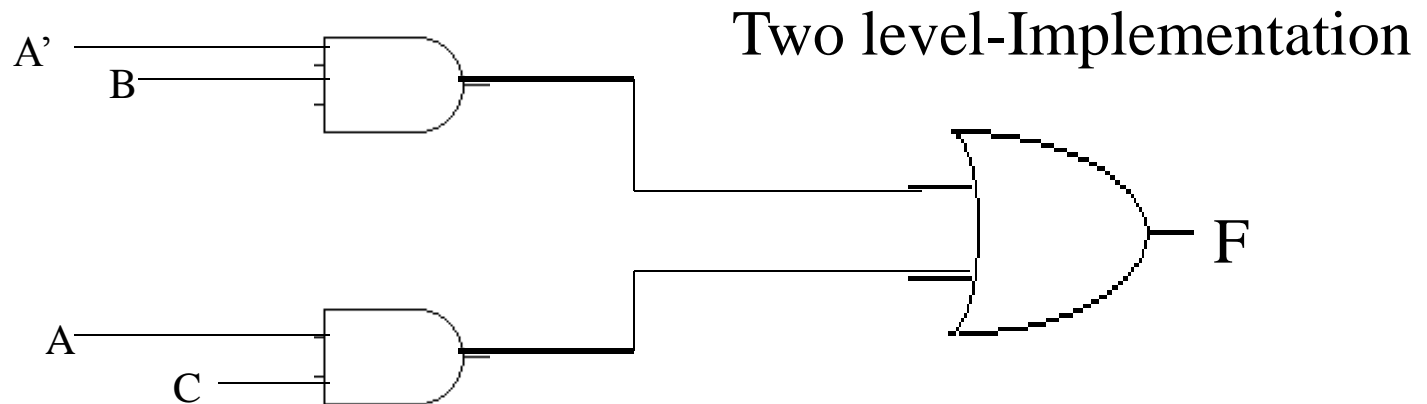
Using the Boolean identity absorption  $xy + xy' = x$

We can simplify  $m_2$  with  $m_3$  and  $m_5$  with  $m_7$

$$m_2 + m_3 = (A'B)C' + (A'B)C = A'B$$

$$m_5 + m_7 = (AC)B' + (AC)B = AC$$

**Therefore  $F = A'B + AC$**



Circuit with 3 gates instead of 5

# Deriving POS from a truth table

## 2- Expression Product of sums

i	A	B	C	F	Maxterms
0	0	0	0	0	$\rightarrow M_0 = A+B+C$
1	0	0	1	0	$\rightarrow M_1 = A+B+C'$
2	0	1	0	1	
3	0	1	1	1	
4	1	0	0	0	$\rightarrow M_4 = A'+B+C$
5	1	0	1	1	
6	1	1	0	0	$\rightarrow M_6 = A'+B'+C$
7	1	1	1	1	

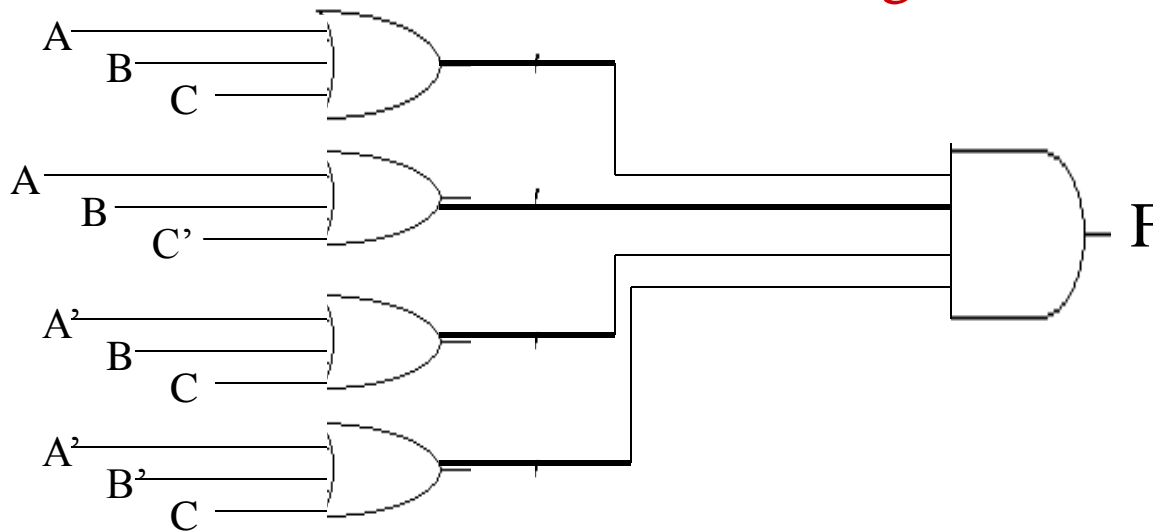
# Deriving SOP from a truth table

## a) Function

$$F = M_0 \cdot M_1 \cdot M_4 \cdot M_6$$

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

Logic circuit



Two level-Implementation

# Simplifying POS using Boolean identities

## b) Simplification

Using  $(X + Y) (X+Y') = X$

Verification

X	Y	Y'	X+ Y	X + Y'	(X + Y) (X+Y')
0	0	1	0	1	0
0	1	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1

$$M0.M1 = [ (A+B) + C] [ (A+B) + C'] = \mathbf{A+B}$$

$$M4.M6 = [ A' + C) + B] [(A'+C) + B'] = \mathbf{A'+C}$$

$$\mathbf{F = (A+B)(A'+C)}$$

# Simplifying POS using Boolean identities

## C) Two level Implementation

