
Chapter 2

Boolean Algebra and Logic Gates

Topics

- Boolean Algebra (Analysis Tools)
 - Boolean Expressions: functions
 - Truth Tables
 - Boolean Identities
 - Standard Forms: Sum of products (SOP) and Product of sums (POS)
- Logic Gates (Hardware)
 - Basic: AND, OR, NOT gates and Binary signals
 - Other gates: NOR, NAND, XOR, XNOR
 - Implementation of Boolean expressions
- Examples
 - Half-Adder, Full-Adder, Deriving SOP/POS from Truth Tables, Simplifying SOP/POS with Boolean Algebra

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 variable 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

- AND operator is a dot or by the absence of an operator

Example: $x \bullet y = z$ or $xy = z$

read: x AND y is equal to z

Interpretation: $z = 1$ if and only if $x = 1$ 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 xy (i.e. $x \bullet y$) for all possible values of x and y

Don't confuse this with binary multiplication operation

Basic Boolean Operations- *OR* operation

- *OR* Operator is 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 $x+y$ 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 x' 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: $x.y$, or xy

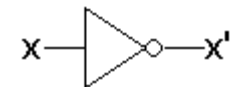
$x + y$

x'

Logic gate:

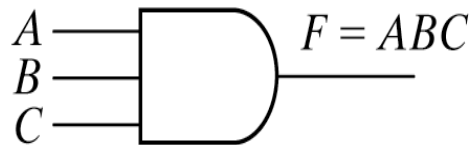


ITI1100

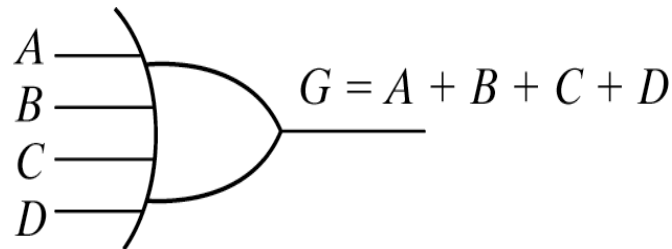


Gates with Multiple Inputs

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



(a) Three-input AND gate

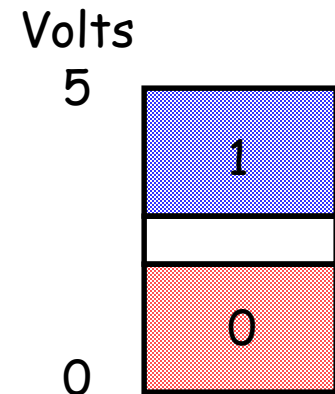


(b) Four-input OR gate

Binary Signals

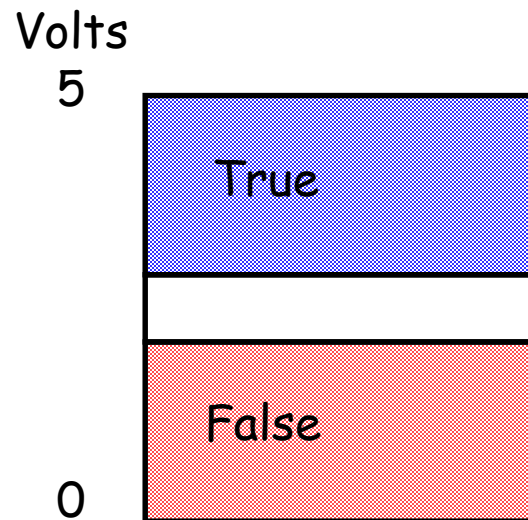
- Computers use voltages to represent information.
- Two voltage levels are used to represent a binary value
“1” and “0”
- Some digital systems for example may define that:
 - Binary “0” is equal to 0 Volt
 - Binary “1” is equal to 5 Volt

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

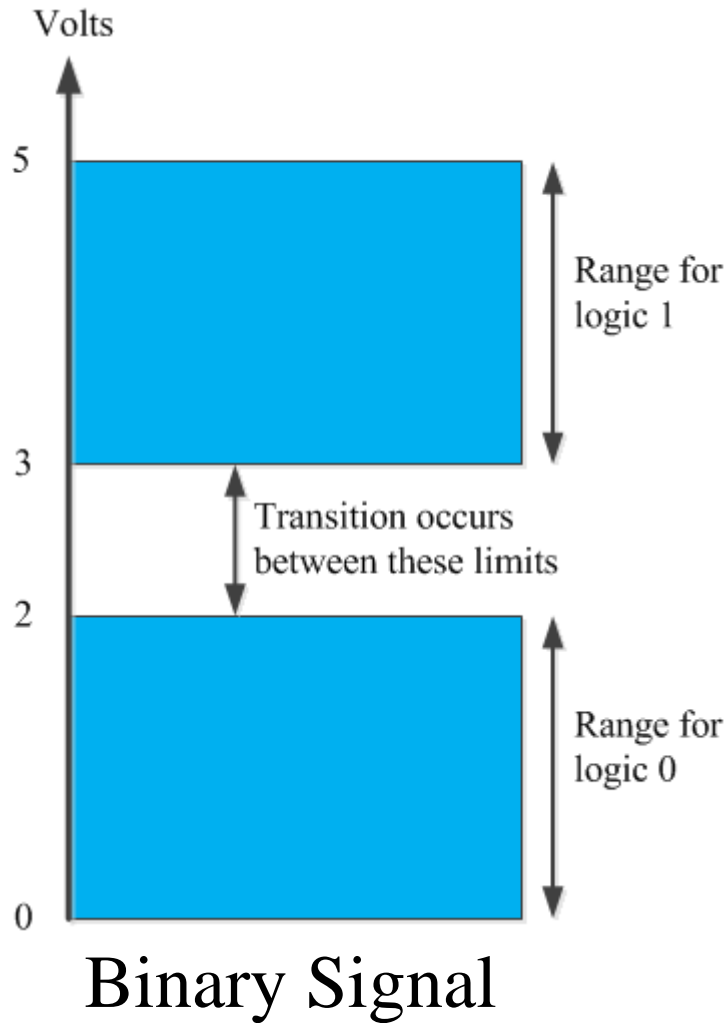


Binary Logic and Binary Signals

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

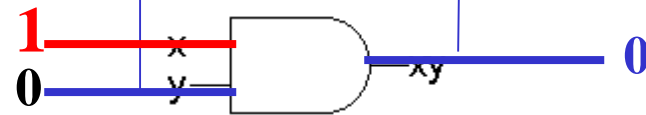


Logic Gates - Signals

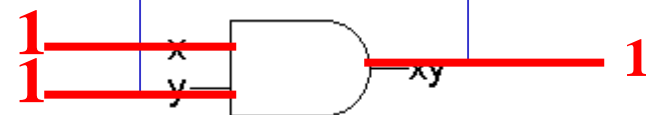


Example

two input signals *one output signal*



two input signals *one output signal*



Timing Diagram –Input and output signals

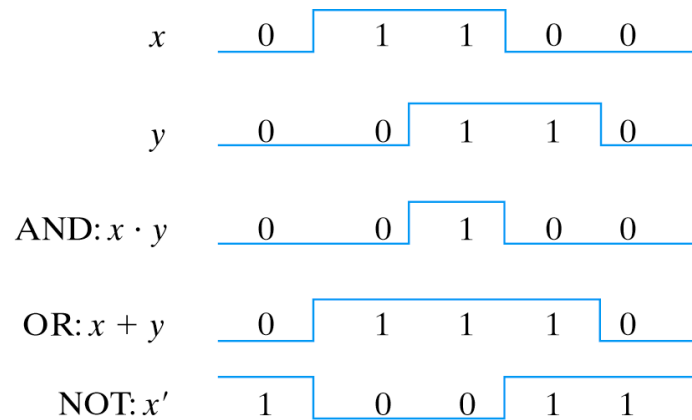
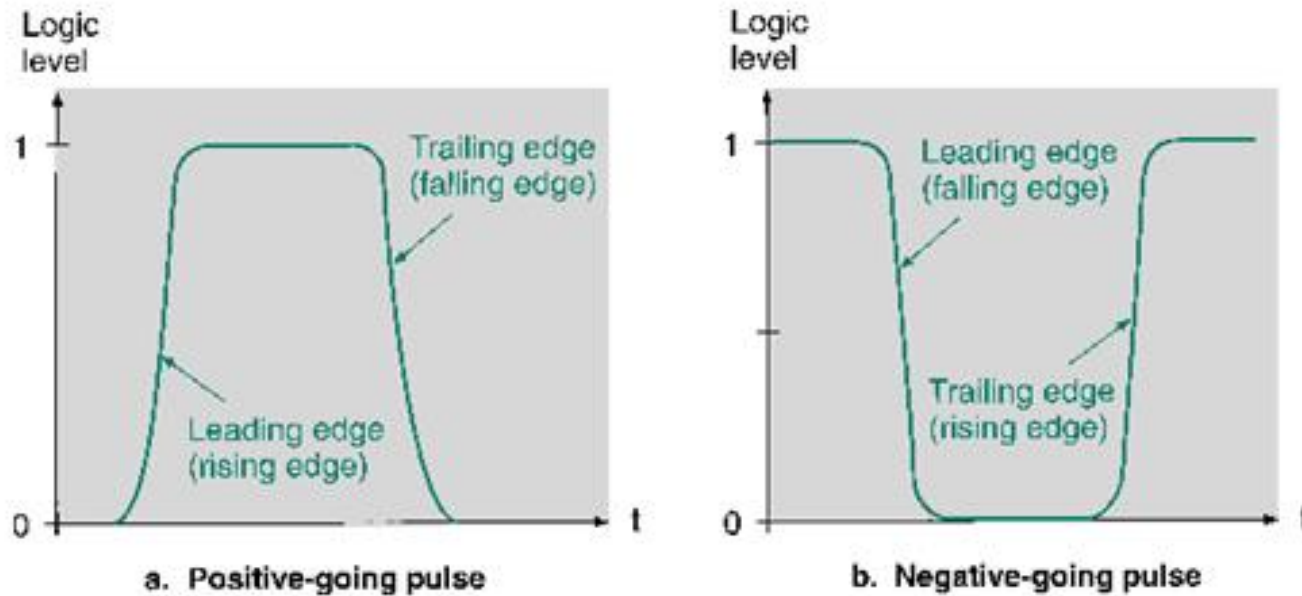


Fig. 1-5 Input-output signals for gates

Boolean Algebra

- 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 Algebra

- A *Boolean algebra* is defined as a **closed algebraic system** containing a set K or two or more elements and the two operators, \cdot and $+$, **provided 6 postulates are satisfied.**
- Useful for identifying and **minimizing circuit functionality**
- Identity elements
 - $a + 0 = a$
 - $a \cdot 1 = a$
- 0 is the identity element for the $+$ operation.
- 1 is the identity element for the \cdot operation.

Table 2.1 Postulates and Theorems of Boolean Algebra

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Copyright ©2012 Pearson Education, publishing as Prentice Hall

- All can be proven **by truth tables.**
- Theorems can be proven **by postulates or by truth tables.**

Commutativity, Associativity and Distributivity of the Operators

- The Commutative Property: For every a and b in K ,
 - $a + b = b + a$
 - $a \cdot b = b \cdot a$
- The Associative Property: For every a , b , and c in K ,
 - $a + (b + c) = (a + b) + c$
 - $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
- The Distributive Property: For every a , b , and c in K ,
 - $a + (b \cdot c) = (a + b) \cdot (a + c)$
 - $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
- To simplify notation, the (\cdot) operator is frequently omitted.
 - $a + b \cdot c = (a + b) \cdot (a + c)$
 - $a + bc = (a + b)(a + c)$

Theorem 1

Theorem : $x+x = x$

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

$$x \cdot 1 = x$$

$$x+x'=1$$

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

$$x \cdot x' = 0$$

$$x+0=x$$

Theorem : $x x = x$

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

DeMorgan's Theorem

- A key theorem in simplifying Boolean algebra expression is DeMorgan's Theorem. It states:

$$(a + b)' = a'b'$$

$$(ab)' = a' + b'$$

- Proven by the truth table.

x	y	$x + y$	$(x + y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x'	y'	$x'y'$
1	1	1
1	0	0
0	1	0
0	0	0

- Complement the expression $a(b + z(x + a'))$ and simplify.

$$\begin{aligned}(a(b+z(x + a')))' &= a' + (b + z(x + a'))' \\ &= a' + b'(z(x + a'))' \\ &= a' + b'(z' + (x + a'))' \\ &= a' + b'(z' + x'a'') \\ &= a' + b'(z' + x'a)\end{aligned}$$

Absorption

- This theorem states:

$$a + ab = a$$

$$a(a+b) = a$$

- To prove the first half of this theorem:

$$\begin{aligned} a + ab &= a \cdot 1 + ab \\ &= a(1 + b) \\ &= a(b + 1) \\ &= a(1) \end{aligned}$$

$$a + ab = a$$

- To prove by the truth table:

x	y
0	0
0	1
1	0
1	1

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

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 (e.g. AND term, OR term): in this example f has two AND 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:

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

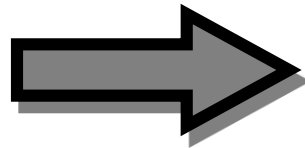
Truth Table

- 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.

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



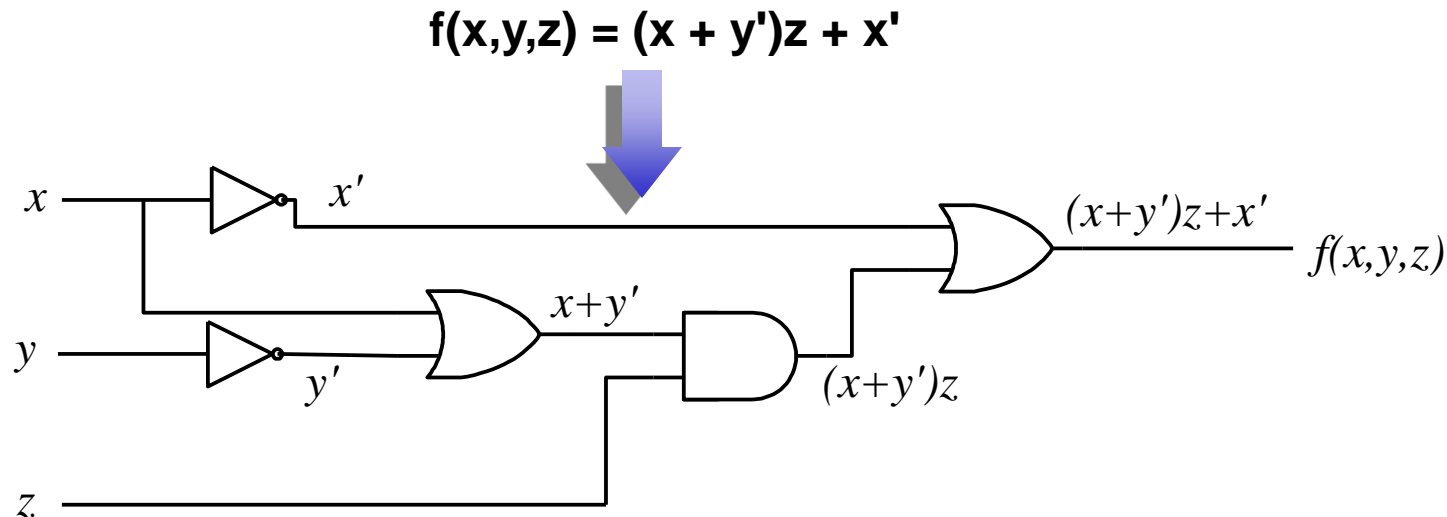
$$\begin{aligned}f(0,0,0) &= (0 + 1)0 + 1 = 1 \\f(0,0,1) &= (0 + 1)1 + 1 = 1 \\f(0,1,0) &= (0 + 0)0 + 1 = 1 \\f(0,1,1) &= (0 + 0)1 + 1 = 1 \\f(1,0,0) &= (1 + 1)0 + 0 = 0 \\f(1,0,1) &= (1 + 1)1 + 0 = 1 \\f(1,1,0) &= (1 + 0)0 + 0 = 0 \\f(1,1,1) &= (1 + 0)1 + 0 = 1\end{aligned}$$



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.



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 ₁	
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

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	

- Sometimes it is easier to work with the **terms that describe when the function is false.**
- For example, if a function has 3 variables, there are 8 possible states. If 6 out of 8 are true, then only 2 out of 8 are false. **Fewer terms makes it easier.**
- The Boolean expression can be obtained from the truth table using **false terms**
So we can also write the function **NOT F₁** as:

$$F_1'(A, B, C) = A'B'C' + A'B'C$$

Simplifying Boolean Expressions

→ Use the Rules of Boolean Algebra

We can simplify the function as:

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



Is it really the
simplest xpression?

Simplifying Boolean Expressions

Function with four variables

- Giving the following function:

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

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

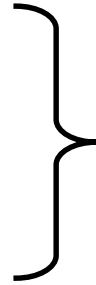
→ the two expressions are equivalent!

→ 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**

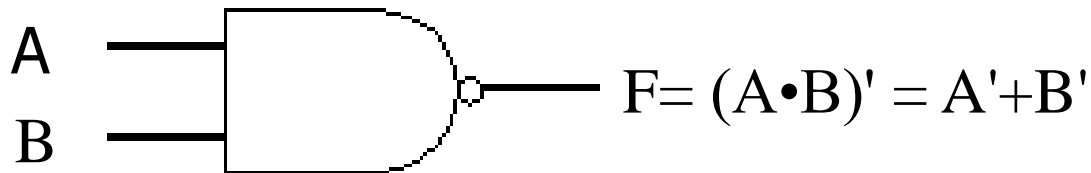
→ Both NAND and NOR gates are very valuable as any design can be realized using either one.

- 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

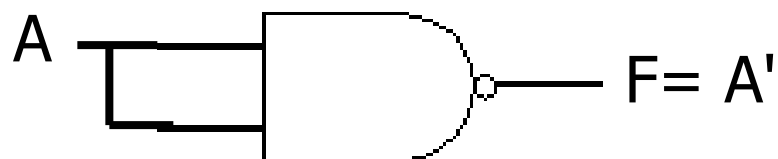
- The NAND gate is a combination of an AND gate followed by an inverter (NOT gate).
- We can use a NAND gate to implement all three of the *basic operations* (AND,OR,NOT).
- Such a gate is said to be **functionally complete**.



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

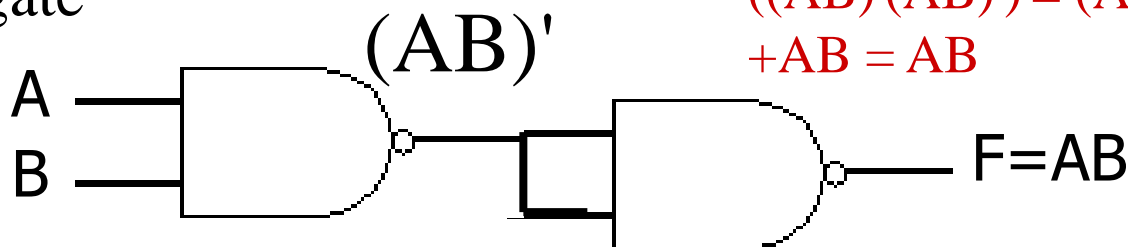
The NAND Gate

→ 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

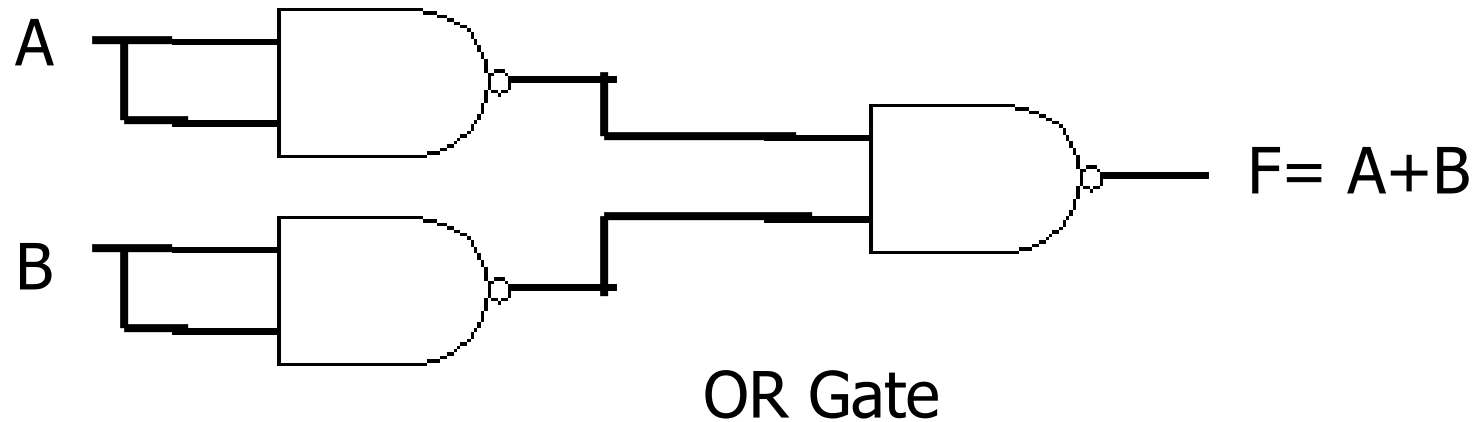


$$\begin{aligned} ((AB)'(AB)')' &= (AB)'' + (AB)'' = AB \\ + AB &= AB \end{aligned}$$

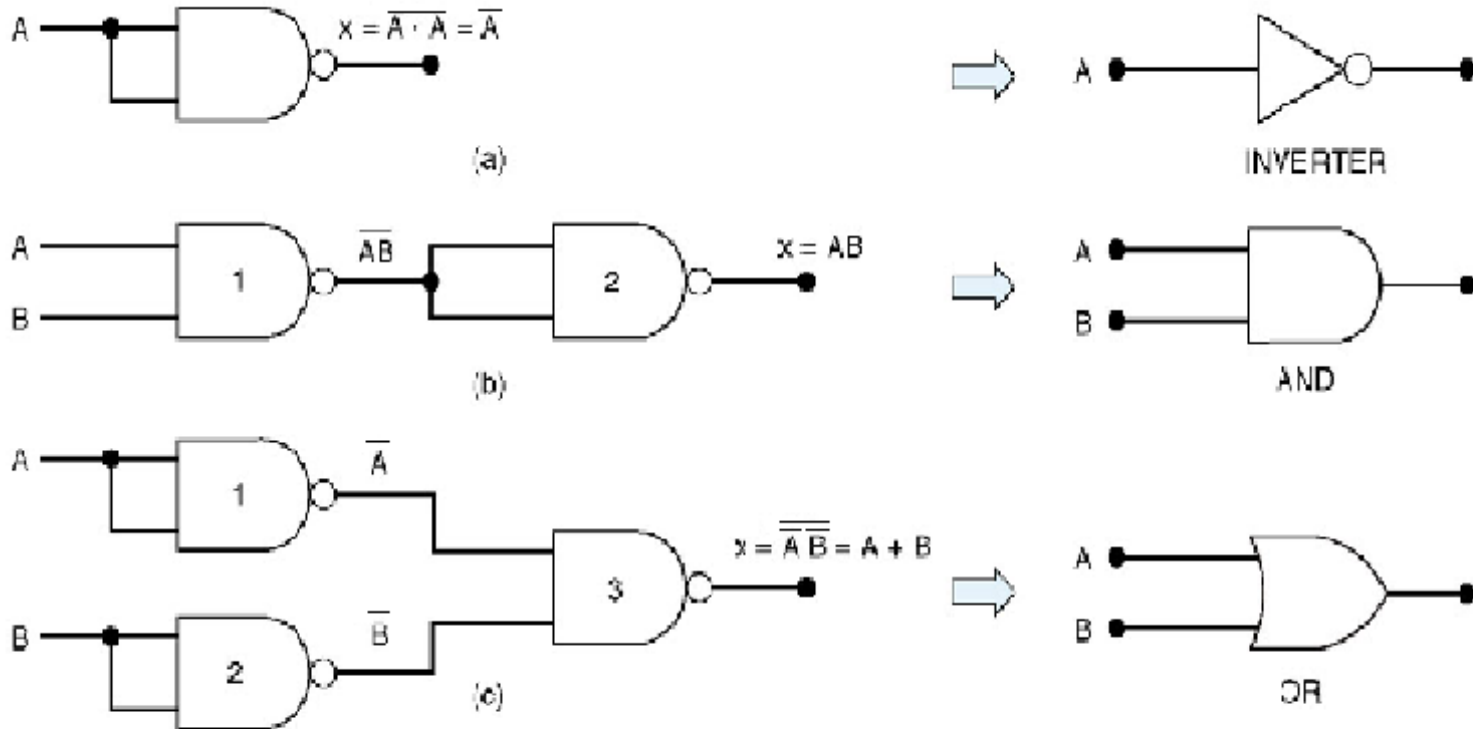
AND Gate

The NAND Gate

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

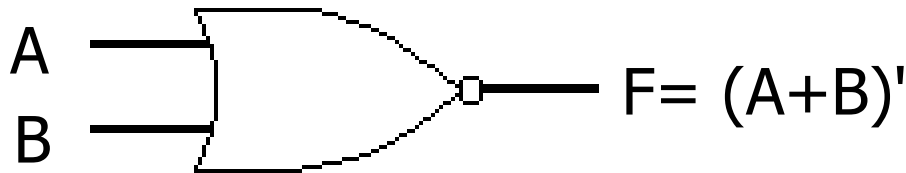


Universality of NAND



The NOR Gate

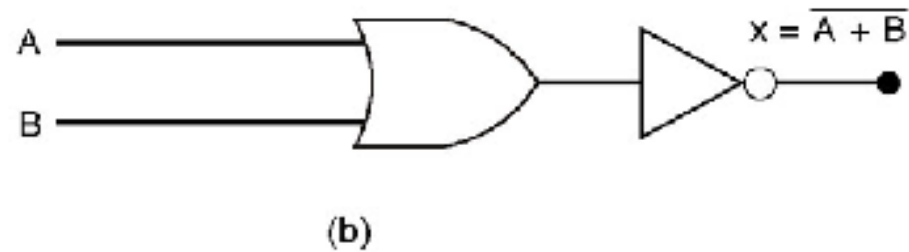
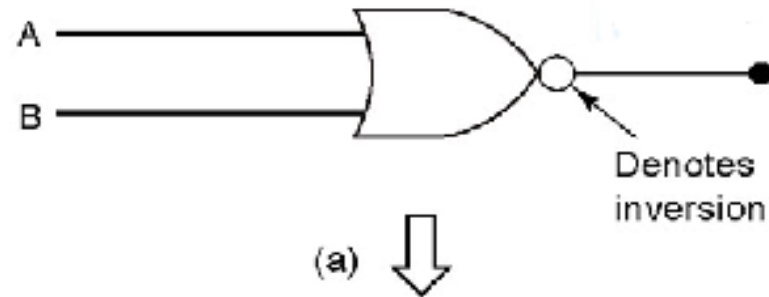
- This is a NOR gate. It is a combination of an OR gate followed by an inverter.
- like the NAND gate, the NOR gate is **functionally complete** → any logic function can be implemented using just NOR gates.



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

NOR Gate Equivalence

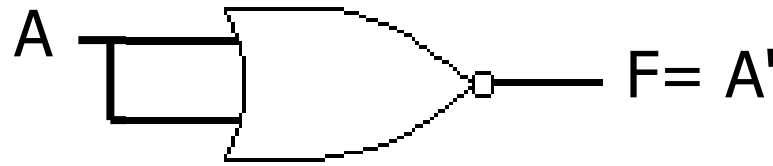
- NOR Symbol, Equivalent Circuit, Truth Table



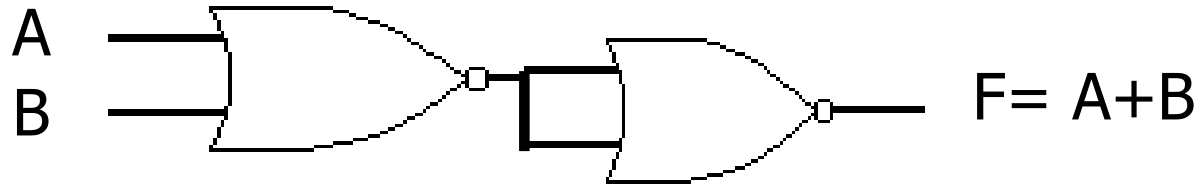
A	B	OR	NOR
		$A + B$	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

(c)

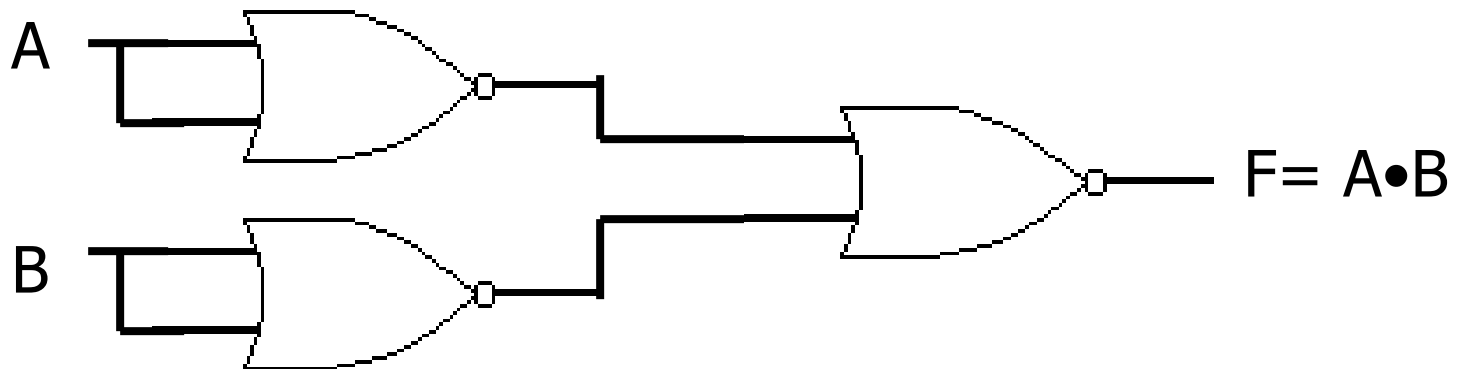
NOR Gates-functionally complete



NOT Gate

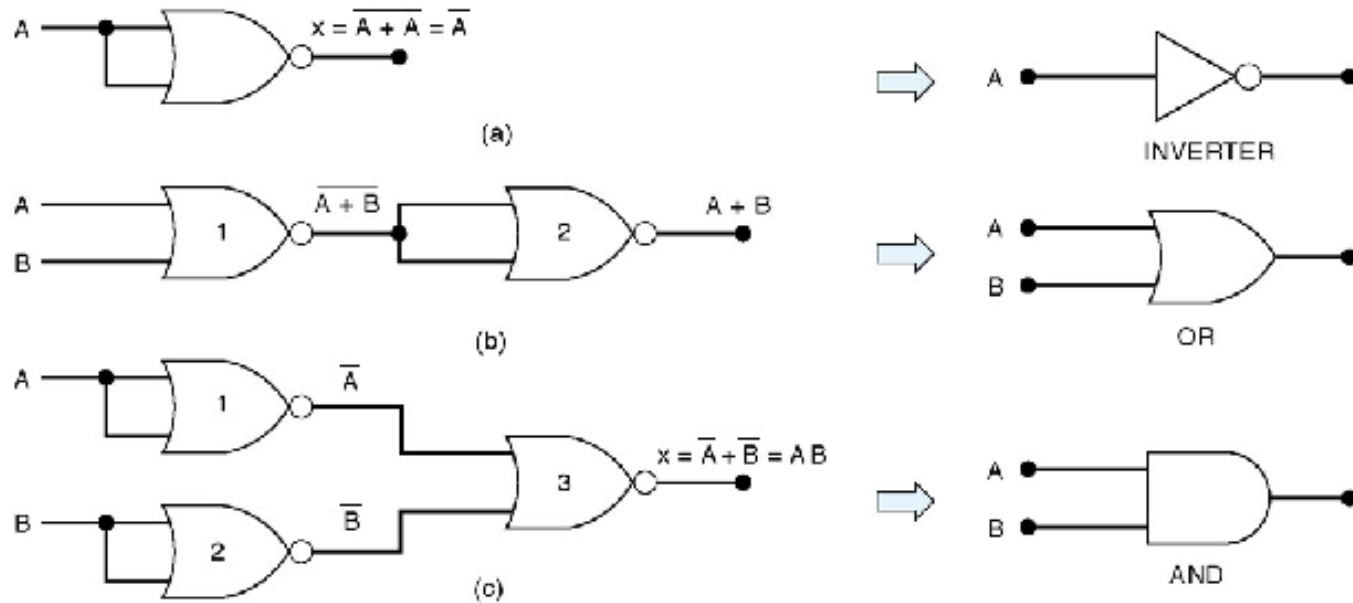


OR Gate



AND Gate

Universality of NOR gate



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

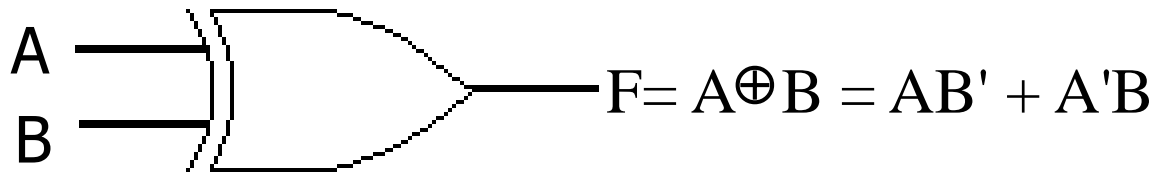
$$(A+A)' = A'A' = A'$$

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

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

The XOR Gate (Exclusive-OR)

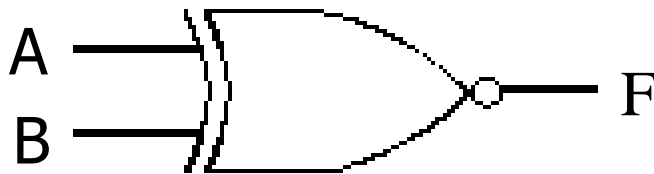
- This is a XOR gate.
- XOR gates assert their output when exactly one of the inputs is asserted, hence the name.
- The operator symbol for this operation is \oplus
 $1 \oplus 1 = 0$ and $1 \oplus 0 = 1$.



A	B	F
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 $1 \odot 0 = 0$.



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

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

Decimal value	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:

$$\mathbf{F(A,B,C) = A'B'C' + A'BC + C'A'B + C'AB' + BAC + BAC}$$

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} :

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

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

Truth Table

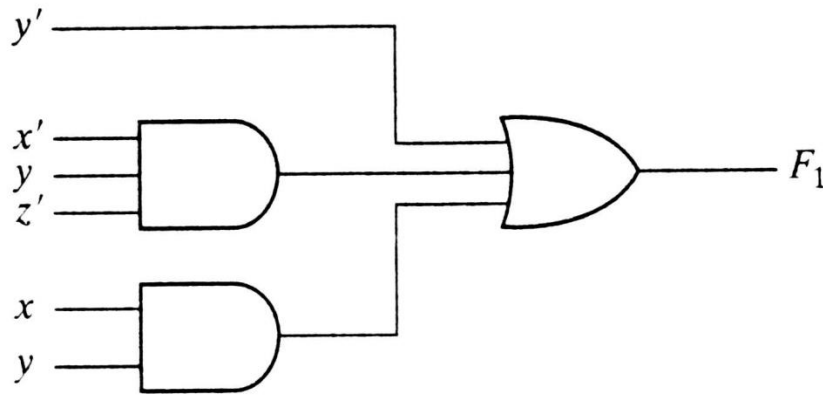
Decimal value	A	B	C	F	Minterm	Maxterm	$M_i = \bar{m}_i$
0	0	0	0	0	m_0	M_0	$A+B+C$
1	0	0	1	0	m_1	M_1	$A+B+C'$
2	0	1	0	1	m_2	M_2	$A+B'+C$
3	0	1	1	1	m_3	M_3	$A+B'+C'$
4	1	0	0	1	m_4	M_4	$A'+B+C$
5	1	0	1	1	m_5	M_5	$A'+B+C'$
6	1	1	0	1	m_6	M_6	$A'+B'+C$
7	1	1	1	1	m_7	M_7	$A'+B'+C'$

Obtain SOP and POS from a given expression

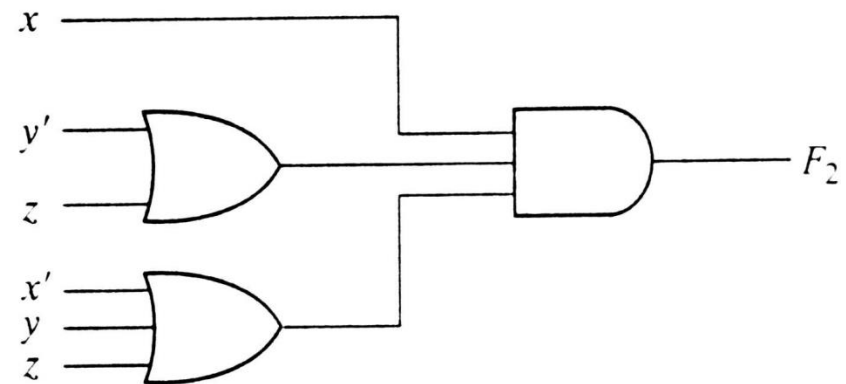
- Given an arbitrary Boolean expression
- Work out number of terms (2^n) for n inputs.
- Generate truth table and identify terms for which the function is true - the Minterms.
- Write function as:
$$F = \sum_{i=0}^{n-1} m_i \quad \text{where } m_i \text{ is } 1$$
- Alternatively, identify terms for which the function is false and use a Maxterm description.
- Write function :
$$F = \prod_{j=0}^{n-1} M_j \quad j \neq i \text{ (i.e. } M_j = 0)$$

SOP & POS Implementation using AND and OR

- Two-level implementation

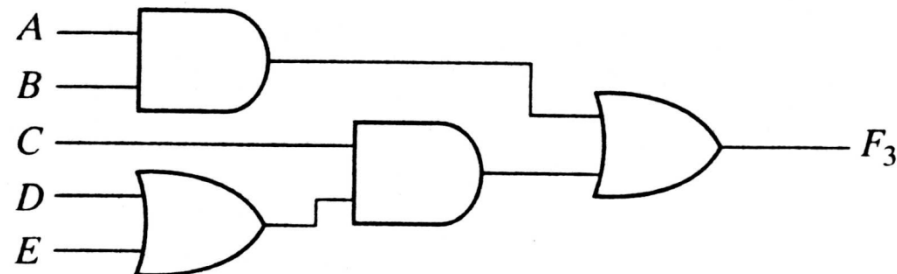


(a) Sum of Products



(b) Product of Sums

- Multi-level implementation



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

Truth Table

A	B	C	C_o	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

$$\begin{aligned}C_0 &= m_3 + m_5 + m_6 + m_7 \\&= A'BC + AB'C + ABC' + ABC \\&= M_0M_1M_2M_4 \\&= (A + B + C)(A + B + C')(A + B' + C)(A' + B + C)\end{aligned}$$

$$\begin{aligned}S &= m_1 + m_2 + m_4 + m_7 \\&= (A'B'C) + (A'BC') + (AB'C') + (ABC) \\&= M_0M_3M_5M_6 \\&= (A + B + C)(A + B' + C')(A' + B + C')(A' + B' + C)\end{aligned}$$

Examples

1- Half Adder

2- Full Adder

3- Deriving SOP and POS from a truth table

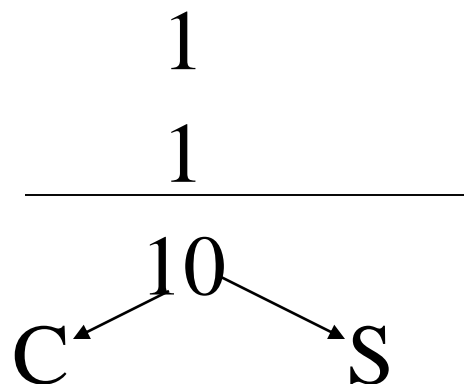
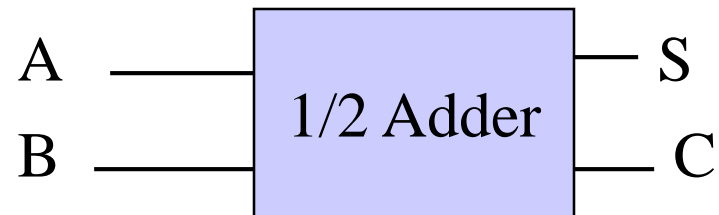
4- Simplifying SOP & POS using Boolean identities

Half Adder

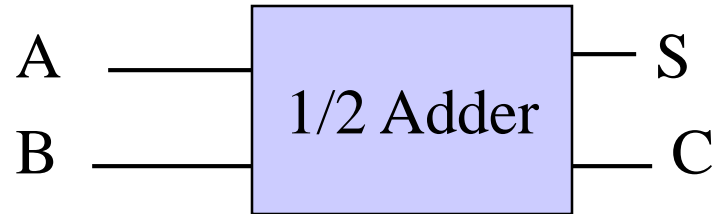
→ The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs: a sum bit and a carry bit.

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

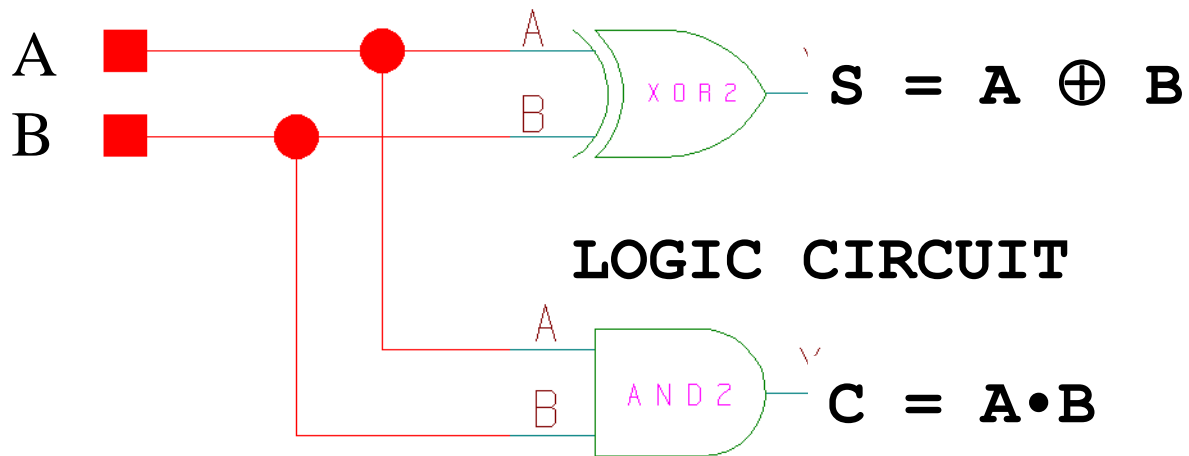
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Logic Function

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

$$S = A \oplus B$$

$$C = A \cdot B$$



Full Adder

Truth Table

A	B	C	C_o	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

→ 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

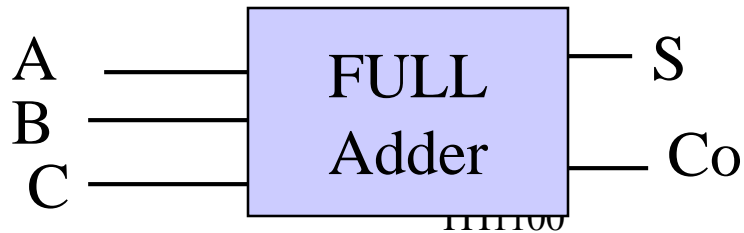
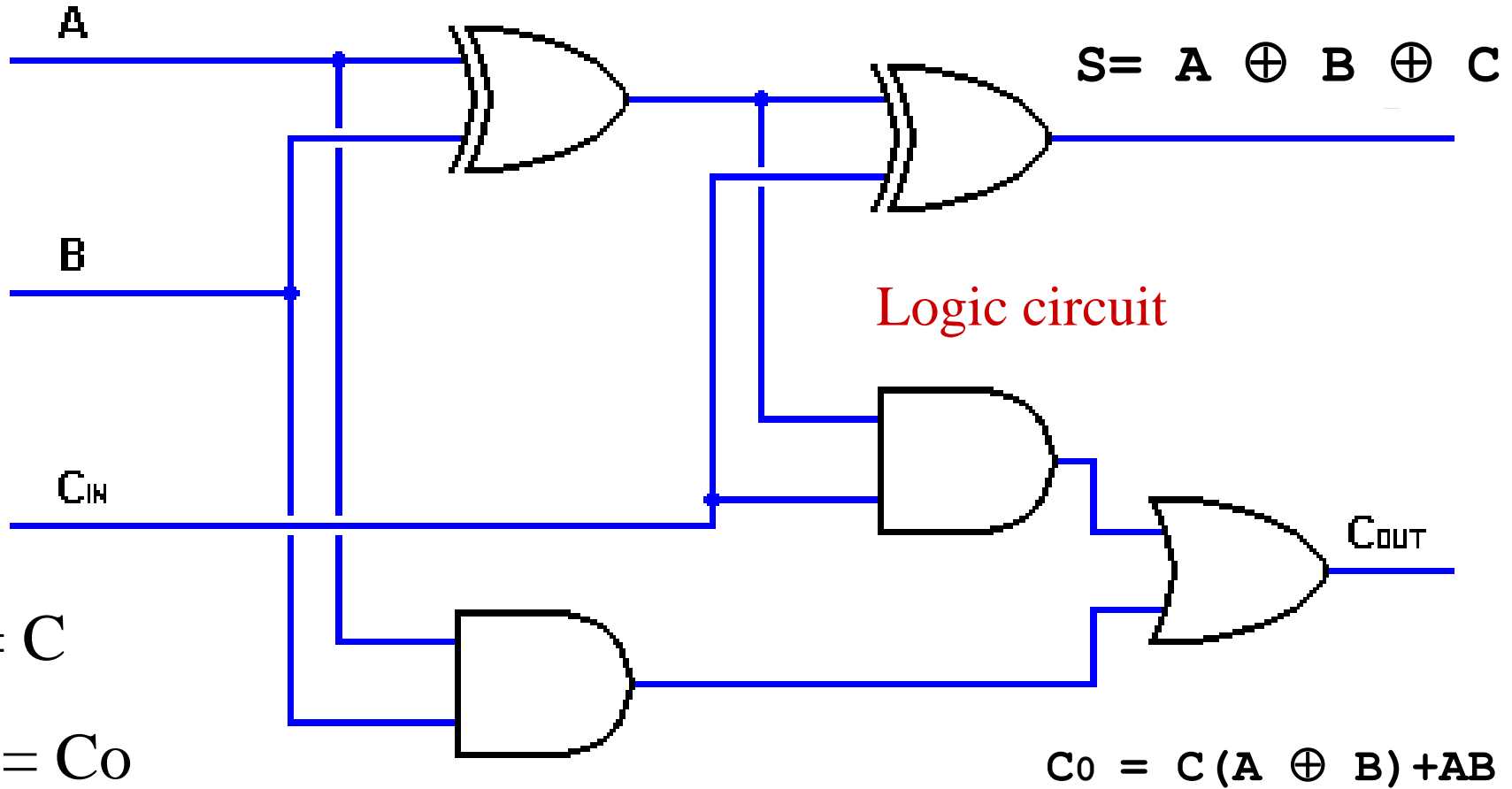
Truth Table

A	B	C	C _o	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

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

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

Full Adder



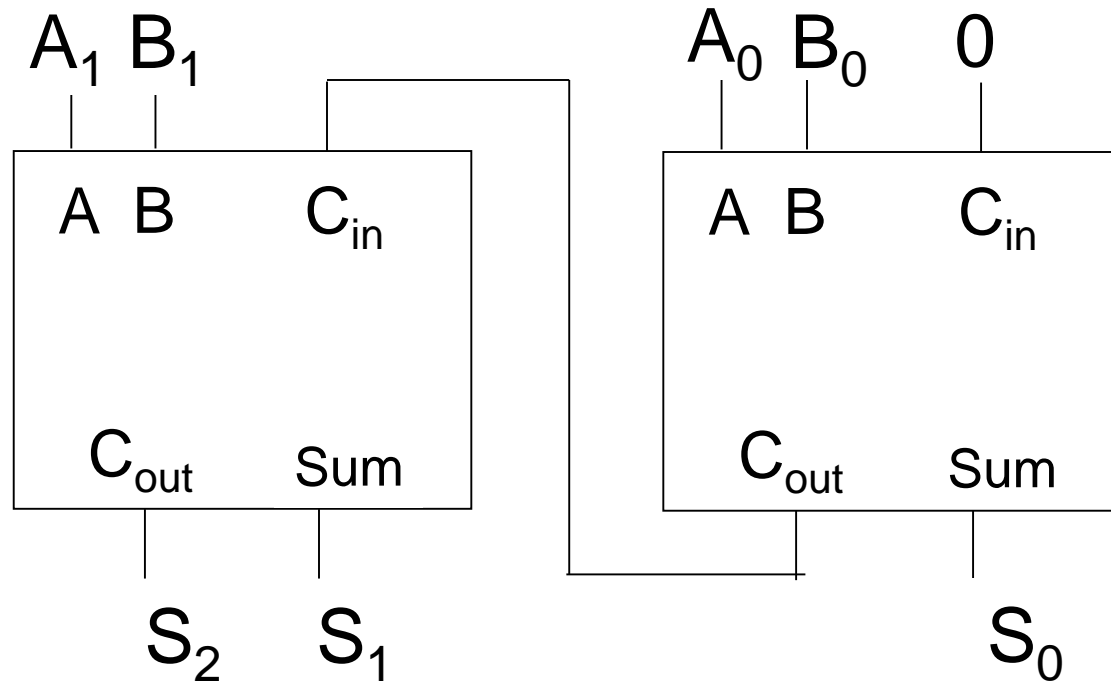
Two bit parallel adder

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

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

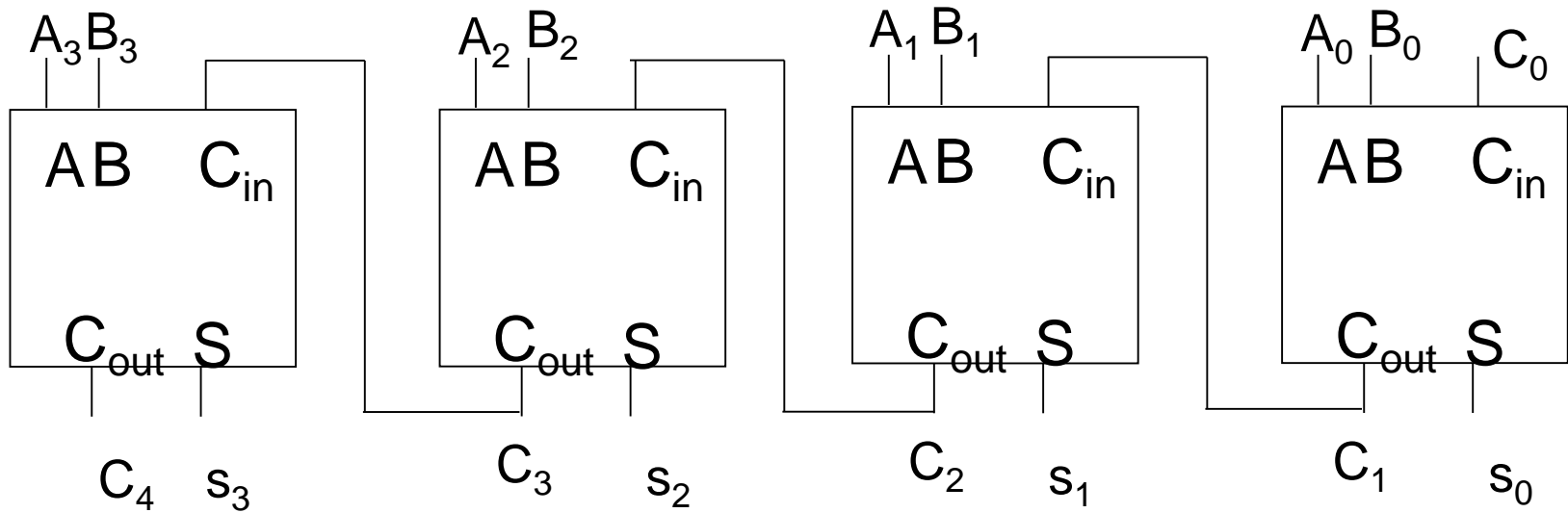
Carry bit from second column
becomes a sum bit

Two bit parallel adder



$$\begin{array}{r} A_1 A_0 \\ + B_1 B_0 \\ \hline S_2 S_1 S_0 \end{array}$$

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)₁₀ instead of + (33)₁₀**
- 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

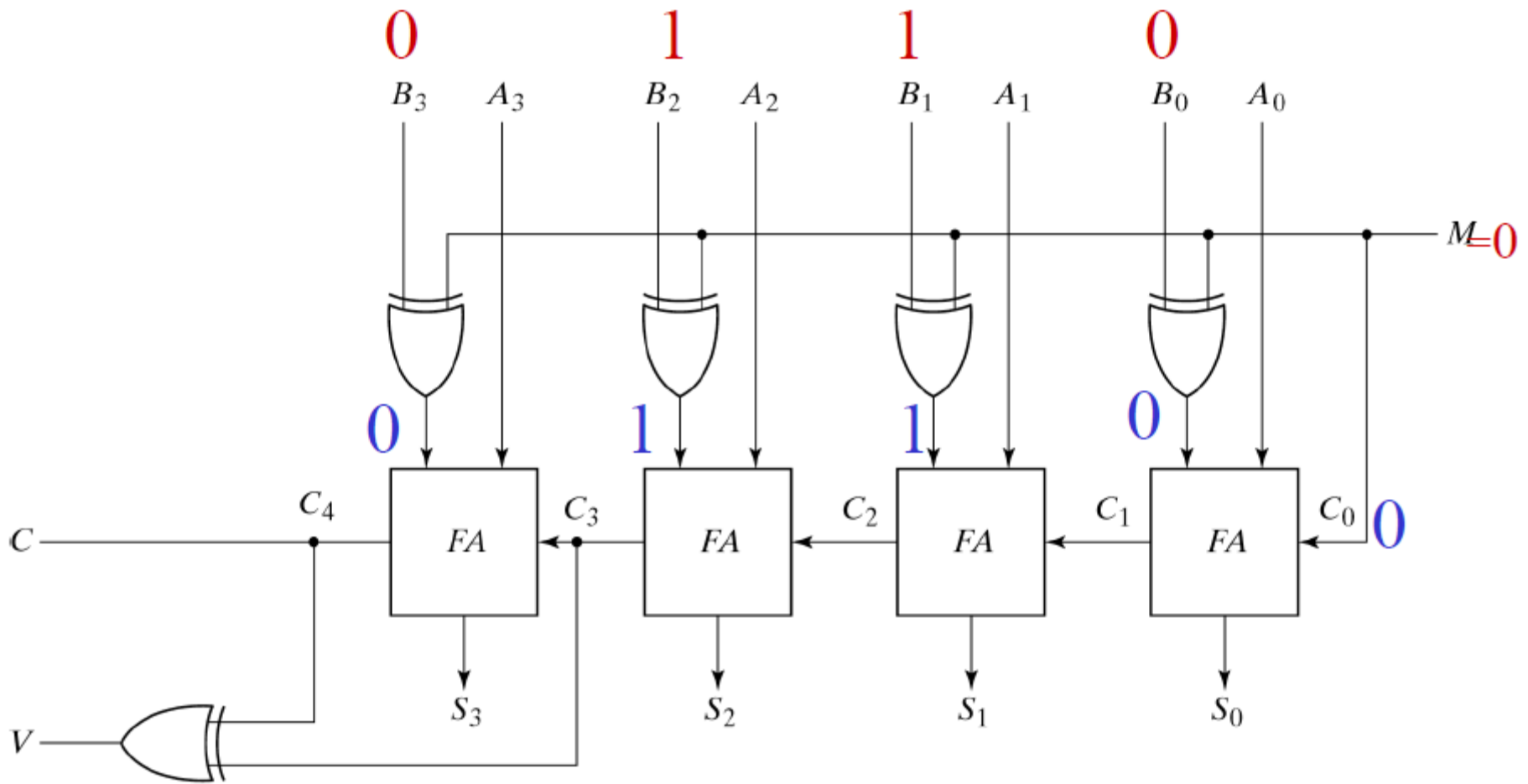


Fig. 4-13 4-Bit Adder Subtractor

Four bit parallel adder

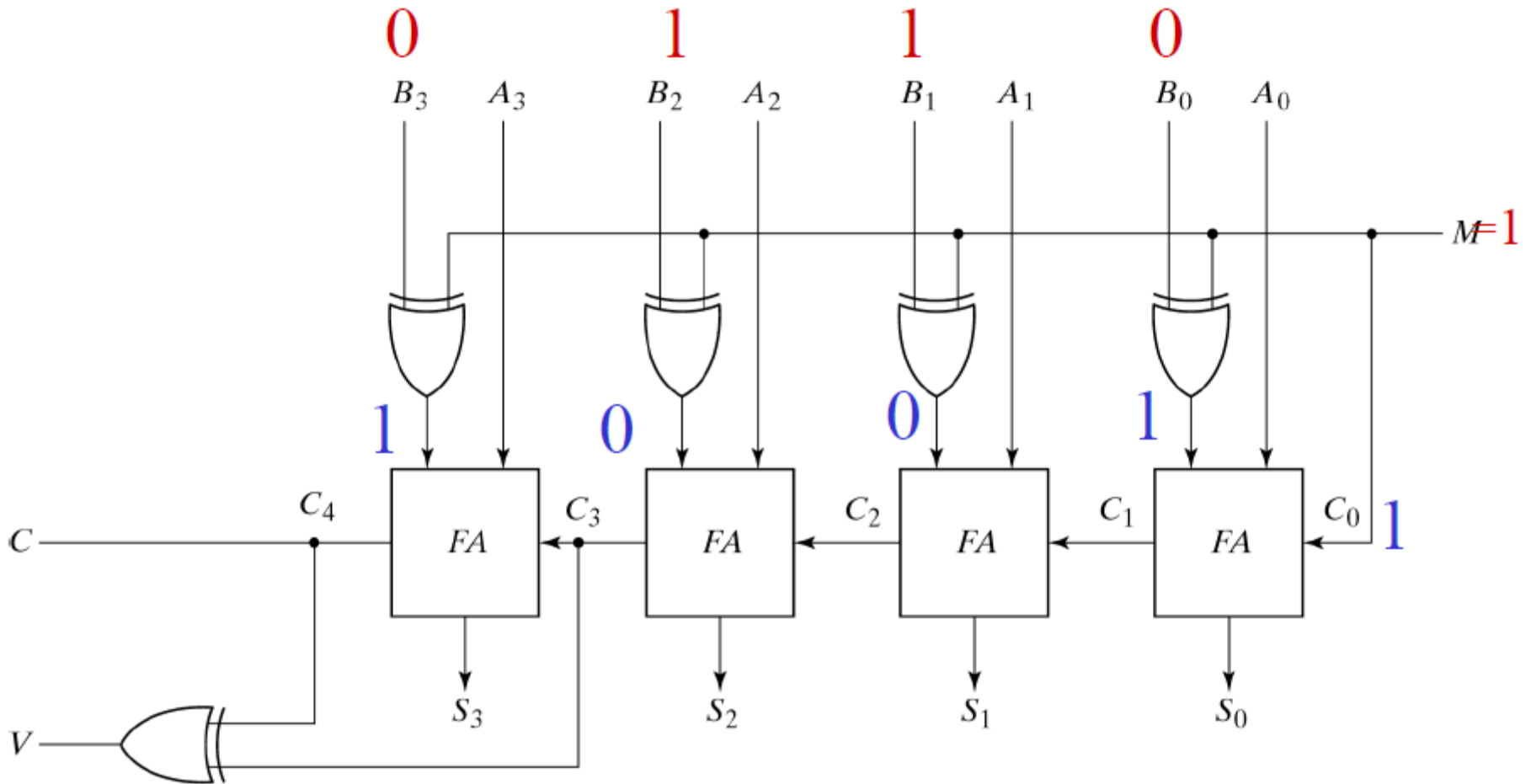
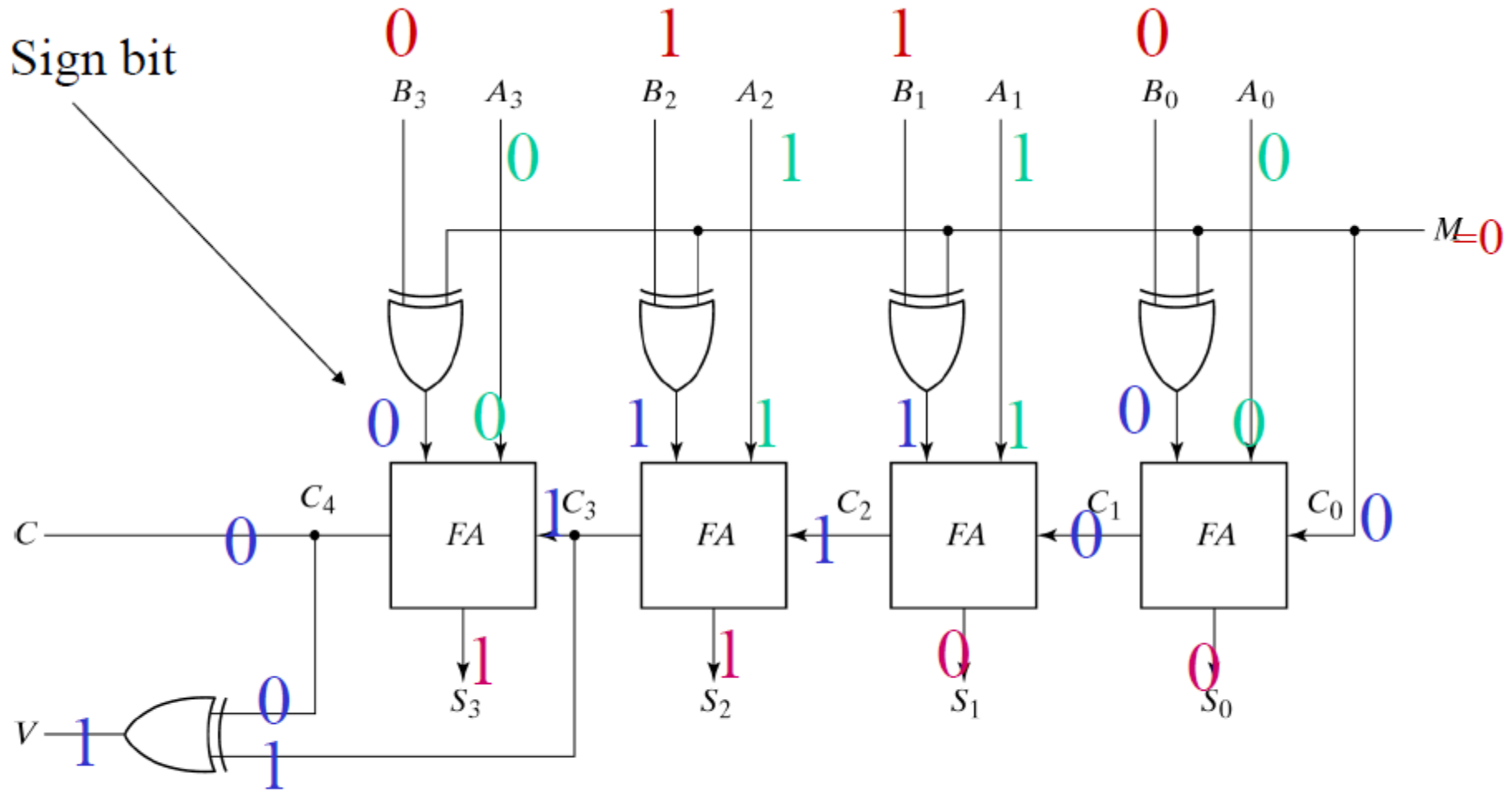


Fig. 4-13 4-Bit Adder Subtractor

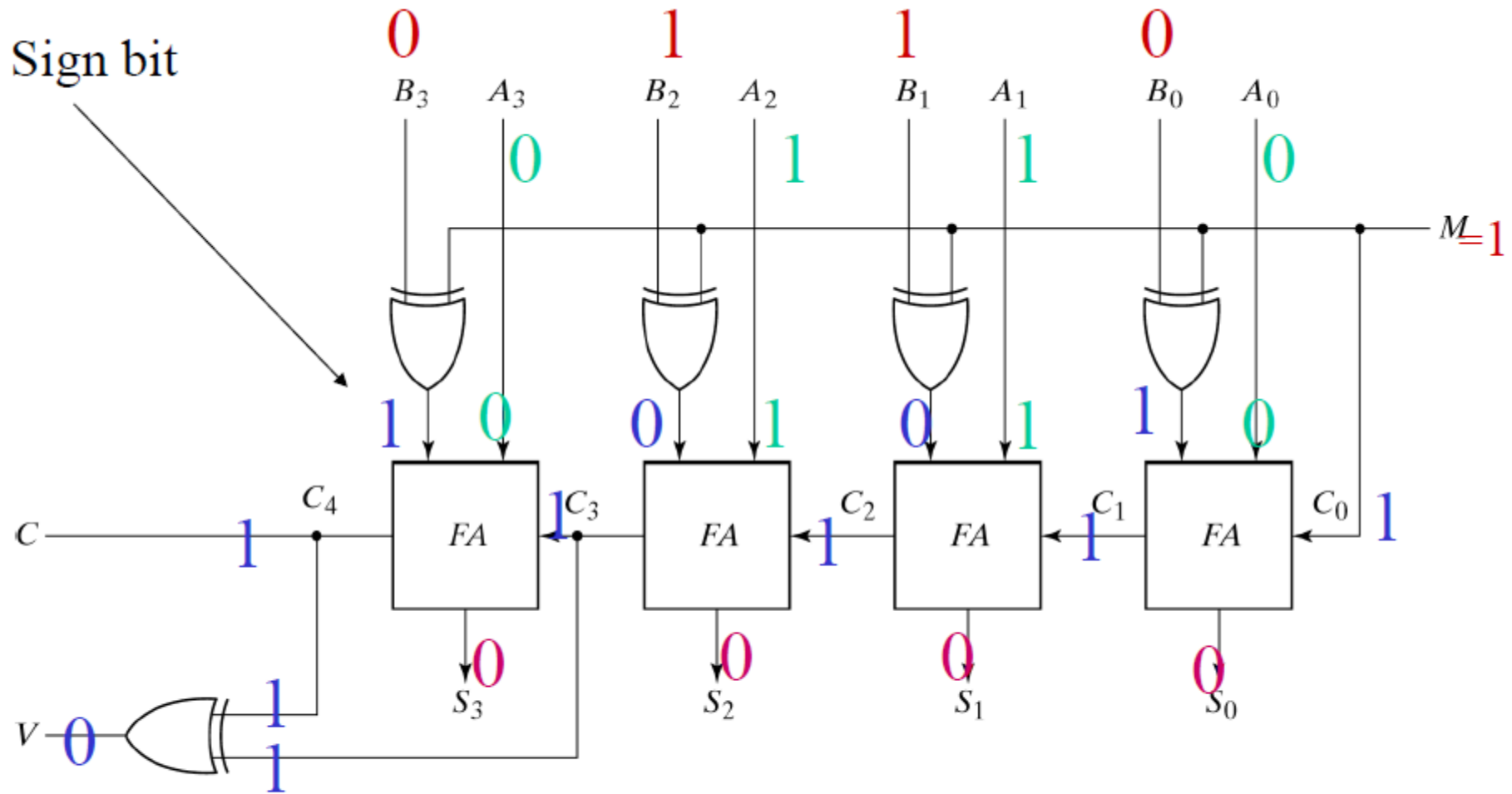
Four bit parallel adder: more examples



Overflow

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

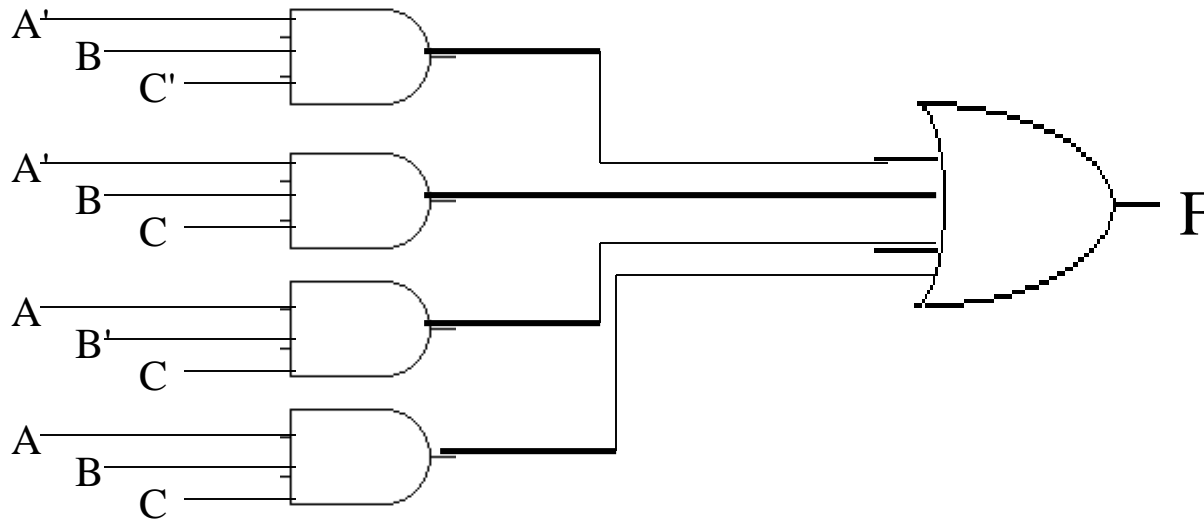
i	A	B	C	F	Minterms
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	1	$\rightarrow m_2 = A'BC'$
3	0	1	1	1	$\rightarrow m_3 = A'BC$
4	1	0	0	0	
5	1	0	1	1	$\rightarrow m_5 = AB'C$
6	1	1	0	0	
7	1	1	1	1	$\rightarrow m_7 = ABC$

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

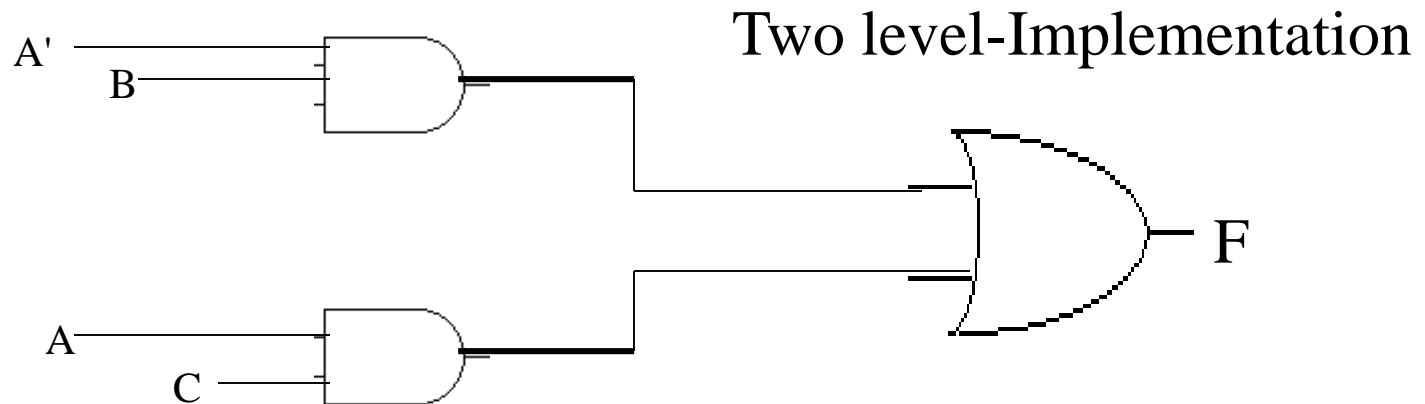
Using the Boolean identity absorption $xy + xy' = x$

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	

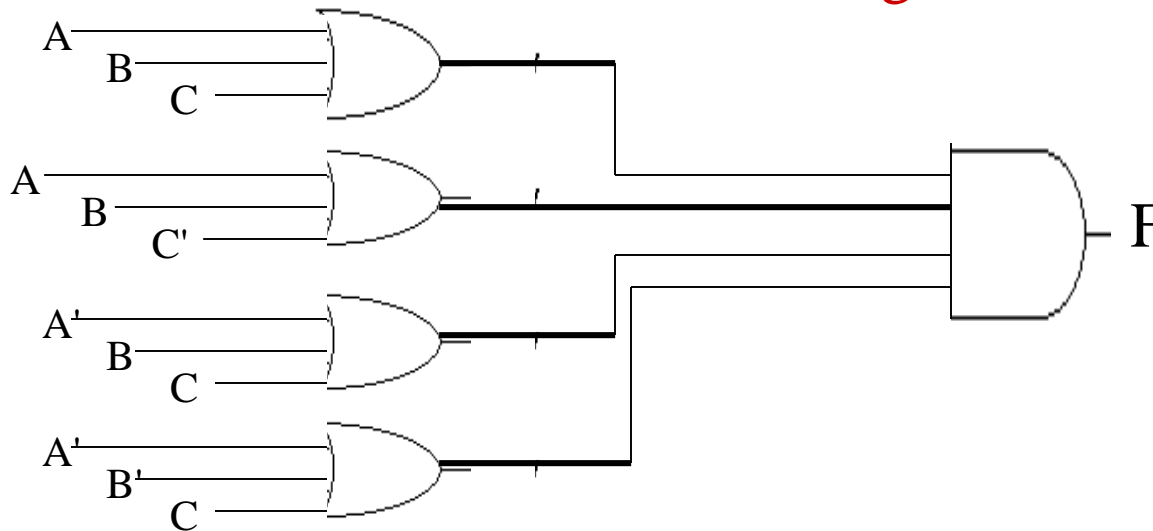
Deriving SOP from a truth table

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

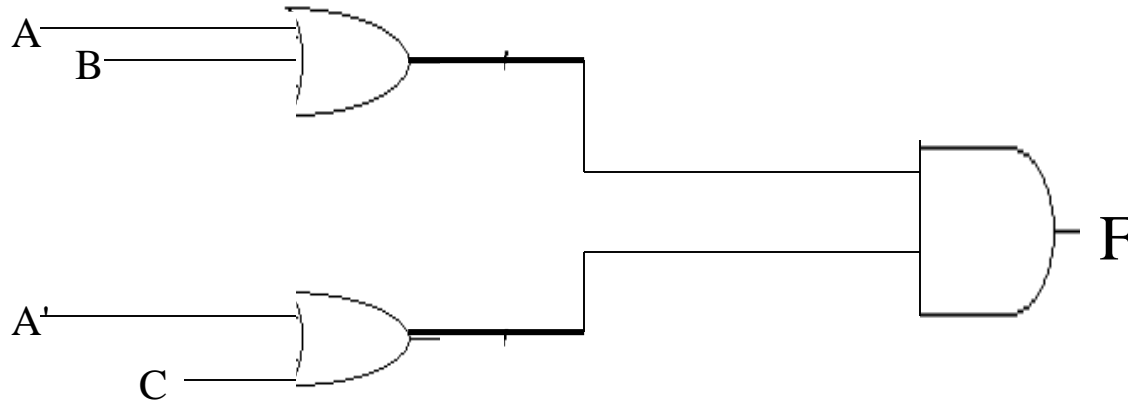
$$M0 \cdot M1 = [(A+B) + C] [(A+B) + C'] = \mathbf{A+B}$$

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

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

Simplifying POS using Boolean identities

c) Two level Implementation



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

$$\text{Alarm} = \overline{C}\overline{D}\overline{M}\overline{Q} + \overline{C}\overline{D}M\overline{Q} + C\overline{D}M\overline{Q}$$

Door should not be open

room is closed to the public → door open + motion

room is closed to the public + door closed → motion