جامعة
الملك سعود
King Saud University

CSC 220: Computer Organization

Unit 2
Digital Circuit Design

Prepared by:
Md Saiful Islam, PhD

Department of Computer Science
College of Computer and Information Sciences

Overview

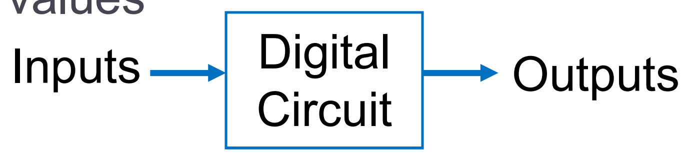
- Digital Circuit Design
  - Logic Gates
  - Logic Functions
  - Standard Forms (SOP/POS)
- Universal Gates (NAND/NOR)
- XOR and XNOR Gates
- Logical Equivalence
- Logic Chips

Chapter-2

M. Morris Mano, Charles R. Kime and Tom Martin, **Logic and Computer Design Fundamentals**, Global (5th) Edition, Pearson Education Limited, 2016. ISBN: 9781292096124

Digital Circuit

- Digital Circuit (hardware) manipulate binary information
  - **Input-output:** one or more binary values

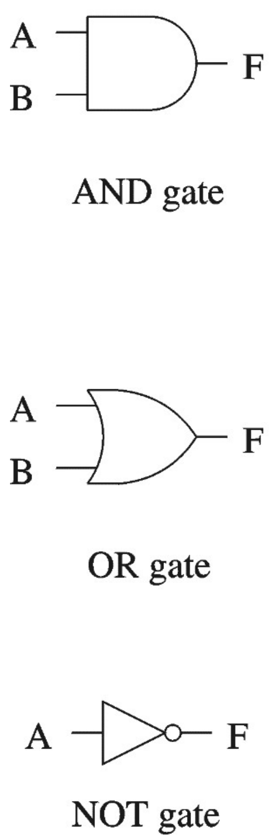


  - Hardware consists of a few simple building blocks called *logic gates*
  - **Logic gate:** a electronic device the operates on one or more input signals and produce an output.
    - **Basic Logic gates:** AND, OR, NOT, …
    - **Additional gates:** NAND, NOR, XOR, XNOR…
- Logic gates are built using transistors
    - NOT gate can be implemented by a single transistor
    - AND-OR gate requires 3 transistors
- Transistors are the fundamental devices
    - Pentium consists of 3 million transistors
    - Compaq Alpha consists of 9 million transistors
    - Now we can build chips with more than 100 million transistors

Logic Gates

- Basic gates
  - **AND**
  - **OR**
  - **NOT**
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination
- Precedence
  - NOT > AND > OR
  - $F = A\,\overline{B} + \overline{A}\,B$

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



Logic symbol

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

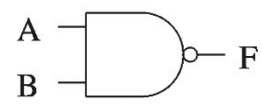
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

A	F
0	1
1	0

Truth table

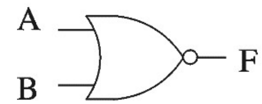
Logic Gates …

- Additional useful gates
  - NAND
  - NOR
  - XOR
  - XNOR
- NAND = AND + NOT
- NOR = OR + NOT
- NAND and NOR gates require only 2 transistors
  - AND and OR need 3 transistors!
- XOR implements exclusive-OR function
- XNOR is complement of XOR



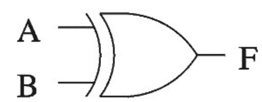
NAND gate

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



NOR gate

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



XOR gate

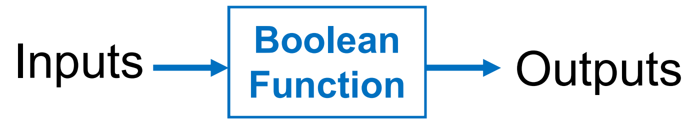
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Logic symbol

Truth table

Logic Functions

- A logic circuit implements a Boolean function.
- A Boolean function consists of
  - Binary variables
  - Constants 0, 1
  - Logic operators: AND (.), OR (+), NOT(-), …



- A function with N input variables
  - With *N* logical variables, we can define

    2^N combination of inputs
  - A **single-output function** relates the output (0/1) to inputs
  - Multiple-output Boolean function
    - More than one outputs
    - Each output (0/1) is related to same inputs

Logic Functions …

- Designing a Logic Circuit (4 steps)
  - Step 1: Represent a logic function using a truth table
  - Step 2: Obtain a logical expression from truth table
  - Step 3: Simplify the function
  - Step 4: Transfer the function to logic diagram of the circuit
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function

Logic Functions …

- **Truth Table:**

3-input majority function

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

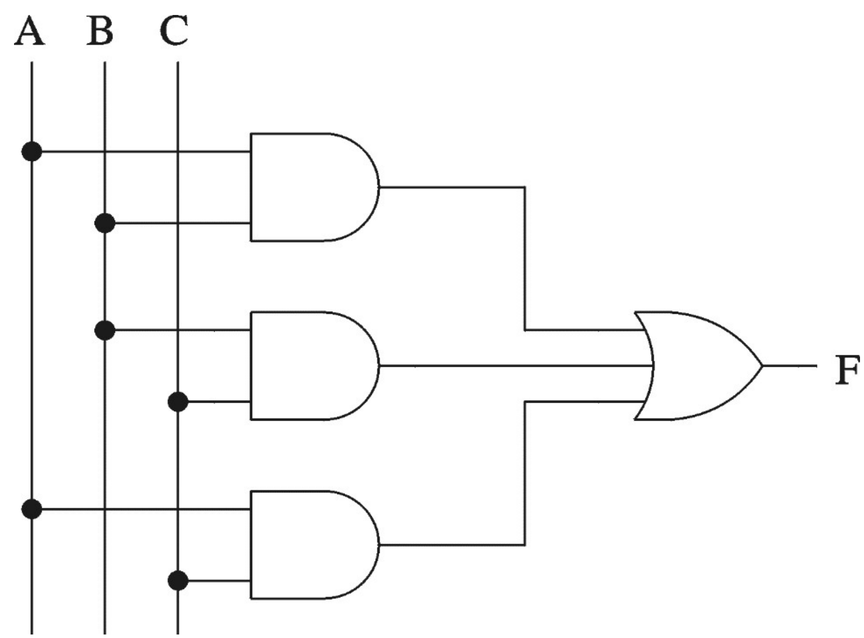
- Logical expression form

F = A'BC + AB'C + ABC' + ABC

- Simplification

F = AB + BC + AC

- Logic Diagram



Standard Forms

Standard Forms Boolean Expressions

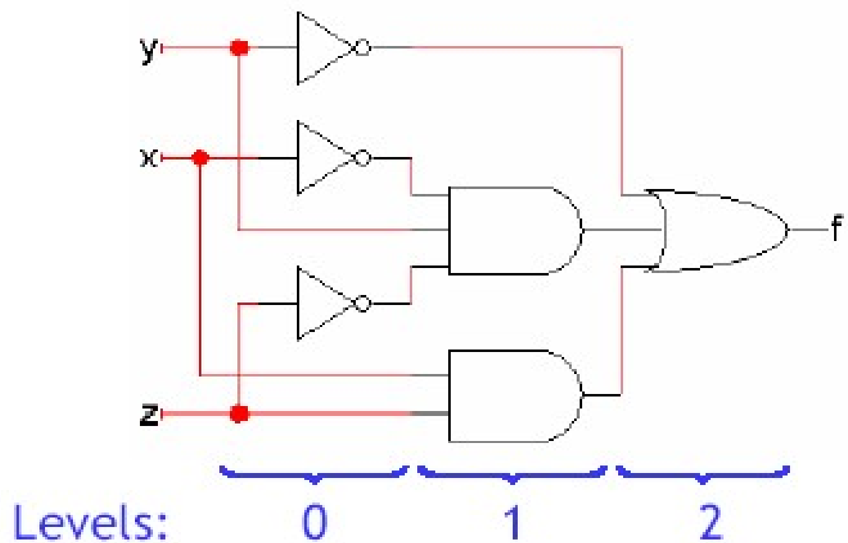
- Sum-of-Products (SOP)
  - Derived from the Truth table for a function by considering those rows for which F = 1.
  - The logical sum (OR) of product (AND) terms.
  - Realized using an AND-OR circuit.
- Product-of-Sums (POS)
  - Derived from the Truth table for a function by considering those rows for which F = 0.
  - The logical product (AND) of sum (OR) terms.
  - Realized using an OR-AND circuit.

Sum of products expressions

- There are many equivalent ways to write a function, but some forms turn out to be more useful than others.
- A sum of products or SOP expression consists of:
  - One or more terms *summed* (OR'ed) together.
  - Each of those terms is a *product of literals*.

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

- Sum of products expressions can be implemented with two-level circuits.



Minterms

- A minterm is a special product of literals, in which each input variable appears exactly once.
- A function with n input variables has 2^n possible minterms.
- For instance, a three-variable function f(x,y,z) has 8 possible minterms:

x'y'z'	x'y'z	x'y z'	x'y z
x y'z'	x y'z	x y z'	x y z

- Each minterm is true for exactly one combination of inputs.

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2 x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1 x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1 x_2 x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2 x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1 x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1 x_2 x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

Sum of minterms expressions

- A sum of minterms is a special kind of sum of products.
- Every function can be written as a *unique* sum of minterms expression.
- A truth table for a function can be rewritten as a sum of minterms just by finding the table rows where the function output is 1.

x	y	z	C(x,y,z)	C'(x,y,z)
0	0	0	0	1
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	0

$$C = x'yz + xy'z + xyz' + xyz$$
$$= m_3 + m_5 + m_6 + m_7$$
$$= \Sigma m(3,5,6,7)$$

$$C' = x'y'z' + x'y'z + x'yz' + xy'z'$$
$$= m_0 + m_1 + m_2 + m_4$$
$$= \Sigma m(0,1,2,4)$$

C' contains all the minterms *not* in C, and vice versa.

Sum-of-Products

- Any function F can be represented by a sum of minterms, where each minterm is ANDed with the corresponding value of the output for F.
  - $F = \Sigma (m_i \cdot f_i)$
    - where m_i is a minterm
    - and f_i is the corresponding functional output

Denotes the logical sum operation

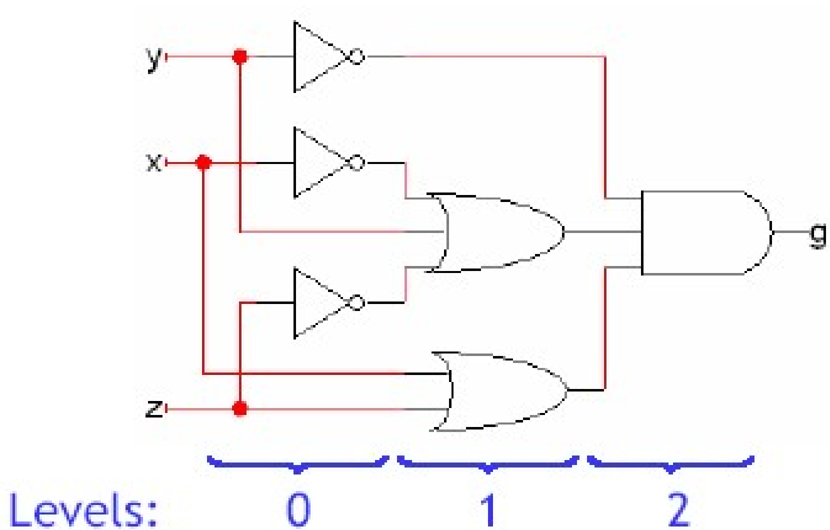
  - Only the minterms for which $f_i = 1$ appear in the expression for function F.
  - $F = \Sigma (m_i) = \Sigma m(i)$ ← shorthand notation
- Sum of minterms are a.k.a. Canonical Sum-of-Products
- Synthesis process
  - Determine the Canonical Sum-of-Products
  - Use Boolean Algebra (and K-maps) to find an optimal, functionally equivalent, expression.

Product of sums expressions

- As you might expect, we can work with the duals of these ideas too.
- A product of sums or POS consists of:
  - One or more terms *multiplied* (AND'ed) together.
  - Each of those terms is a *sum of literals*.

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

- Products of sums can also be implemented with two-level circuits.



Maxterms

- A maxterm is a *sum* of literals where each input variable appears once.
- A function with *n* input variables has 2^n possible maxterms.
- For instance, a function with three variables x, y and z has 8 possible maxterms:

x + y + z	x + y + z’	x + y’+ z	x + y’+ z’
x’+ y + z	x’+ y + z’	x’+ y’+ z	x’+ y’+ z’

- Each maxterm is *false* for exactly one combination of inputs.

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2 x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1 x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1 x_2 x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2 x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1 x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1 x_2 x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

Product of maxterms expressions

- Every function can also be written as a unique product of maxterms.
- A truth table for a function can be rewritten as a product of maxterms just by finding the table rows where the function output is 0.

x	y	z	C(x,y,z)	C'(x,y,z)
0	0	0	0	1
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	0

$$C = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)$$
$$= M_0 M_1 M_2 M_4$$
$$= \prod M(0.1.2.4)$$ **When the o/p is Zero**
$$= \Sigma m(3,5,6,7)$$ **When the o/p is 1**

$$C' = (x + y' + z')(x' + y + z')(x' + y' + z)(x' + y' + z')$$
$$= M_3 M_5 M_6 M_7$$
$$= \prod M(3,5,6,7)$$

C' contains all the maxterms *not* in C, and vice versa.

Product-of-Sums

- Any function F can be represented by a product of Maxterms, where each Maxterm is ANDed with the *complement* of the corresponding value of the output for F.
  - $F = \prod (M_i \cdot f'_i)$
    - where M_i is a Maxterm
    - and f'_i is the complement of the corresponding functional output

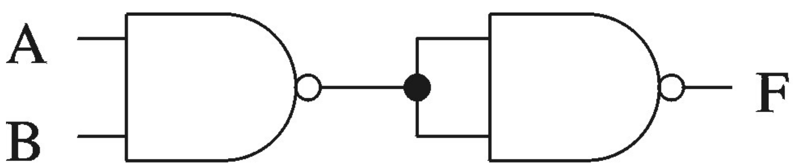
Denotes the logical product operation

  - Only the Maxterms for which $f_i = 0$ appear in the expression for function F.
  - $F = \prod (M_i) = \prod M(i)$ ← shorthand notation
- The Canonical Product-of-Sums for function F is the Product-of-Sums expression in which each sum term is a Maxterm.
- Synthesis process
  - Determine the Canonical Product-of-Sums
  - Use Boolean Algebra (and K-maps) to find an optimal, functionally equivalent, expression.

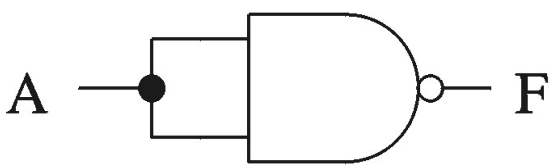
Universal Gates

- NAND and NOR gates are called universal gets

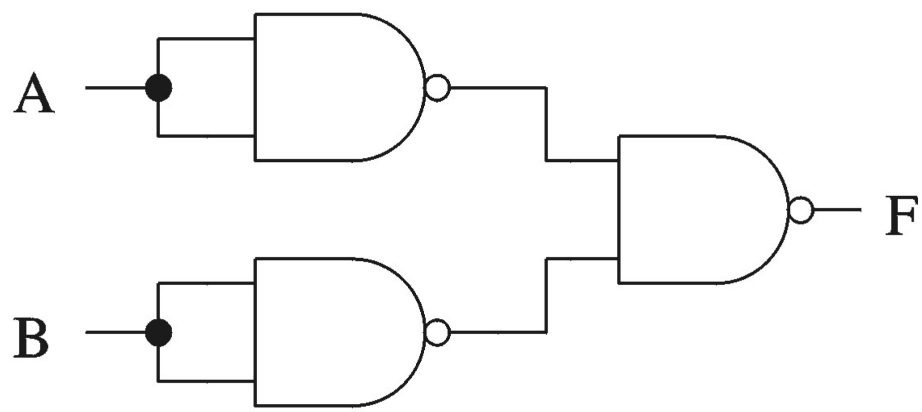
- Proving NAND gate is universal



AND gate



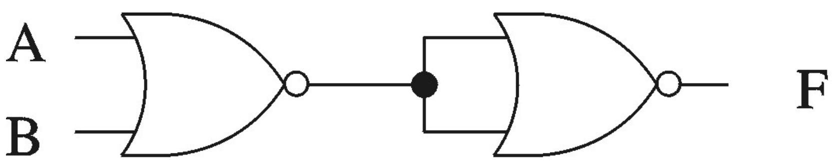
NOT gate



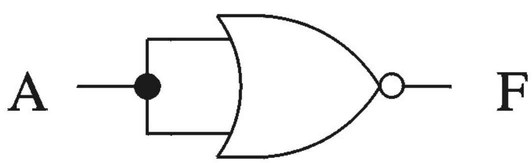
OR gate

Universal Gates…

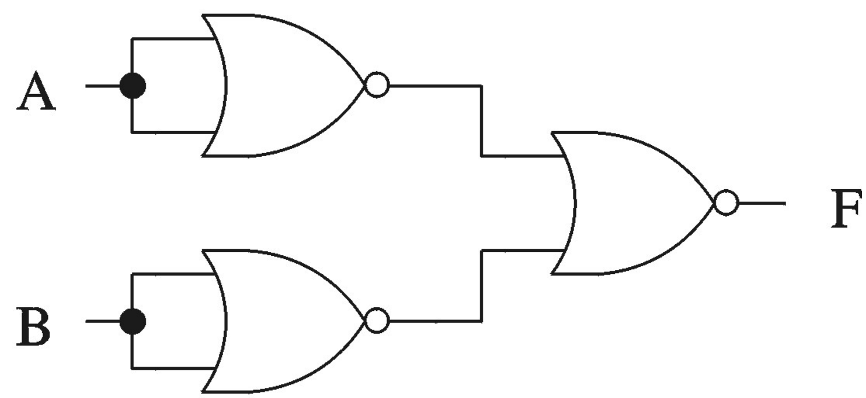
- Proving NOR gate is universal



OR gate



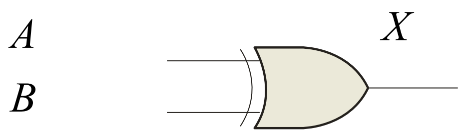
NOT gate

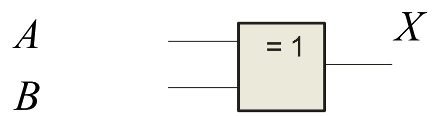


AND gate

XOR and XNOR Gates

The XOR Gate





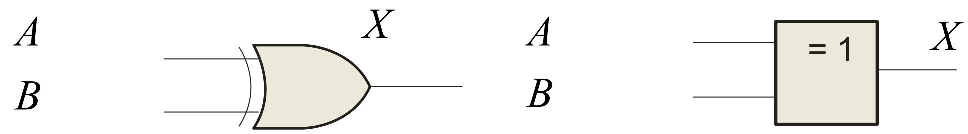
The **XOR gate** produces a HIGH output only when the inputs are at opposite logic levels. The truth table is

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

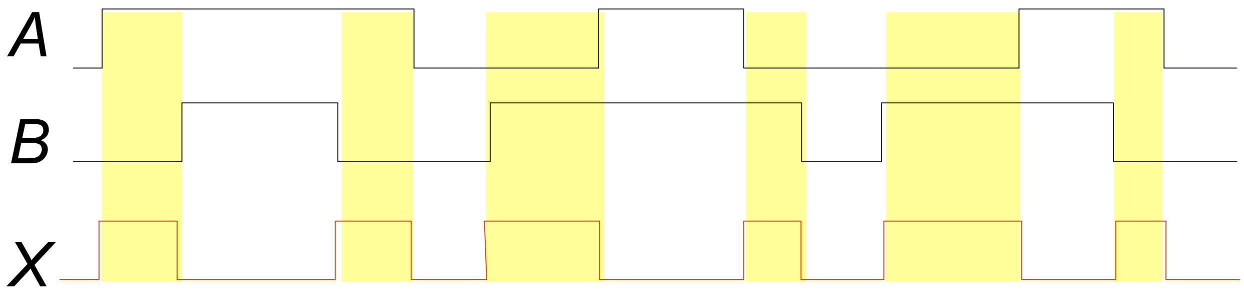
The **XOR** operation is written as $X = \overline{A}B + A\overline{B}$.
Alternatively, it can be written with a circled plus sign between the variables as $X = A \oplus B$.

XOR and XNOR Gates …

The XOR Gate



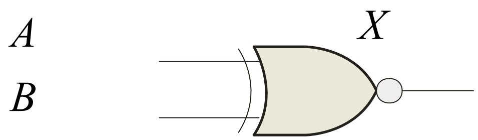
Example waveforms:

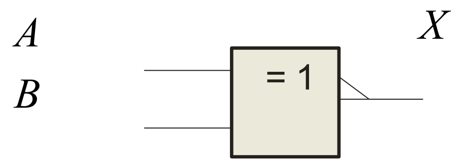


Notice that the XOR gate will produce a HIGH only when exactly one input is HIGH.

XOR and XNOR Gates …

The XNOR Gate



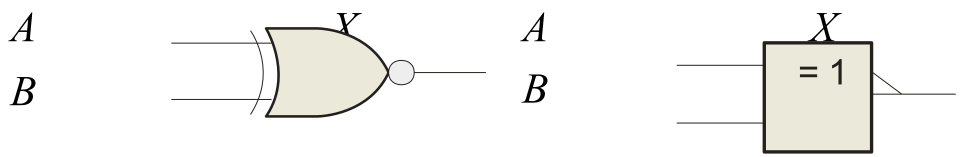


The **XNOR gate** produces a HIGH output only when the inputs are at the same logic level. The truth table is

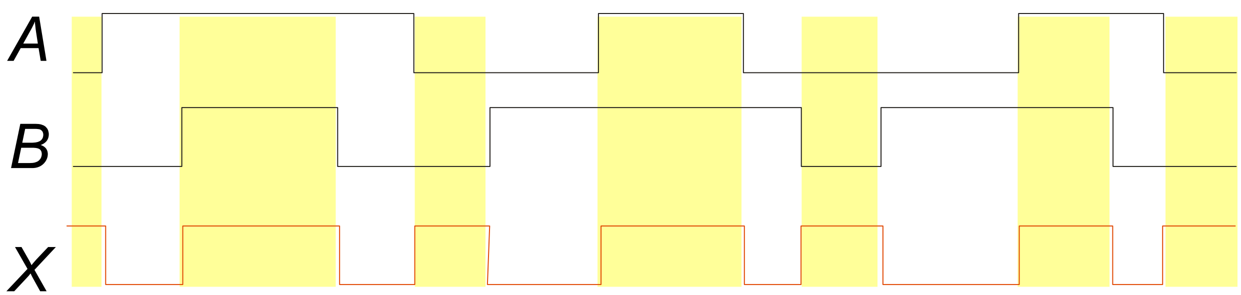
Inputs		Output
A	*B*	*X*
0	0	1
0	1	0
1	0	0
1	1	1

The **XNOR** operation can be shown as $X = AB + \bar{A}\bar{B}$.

The XNOR Gate



Example waveforms:

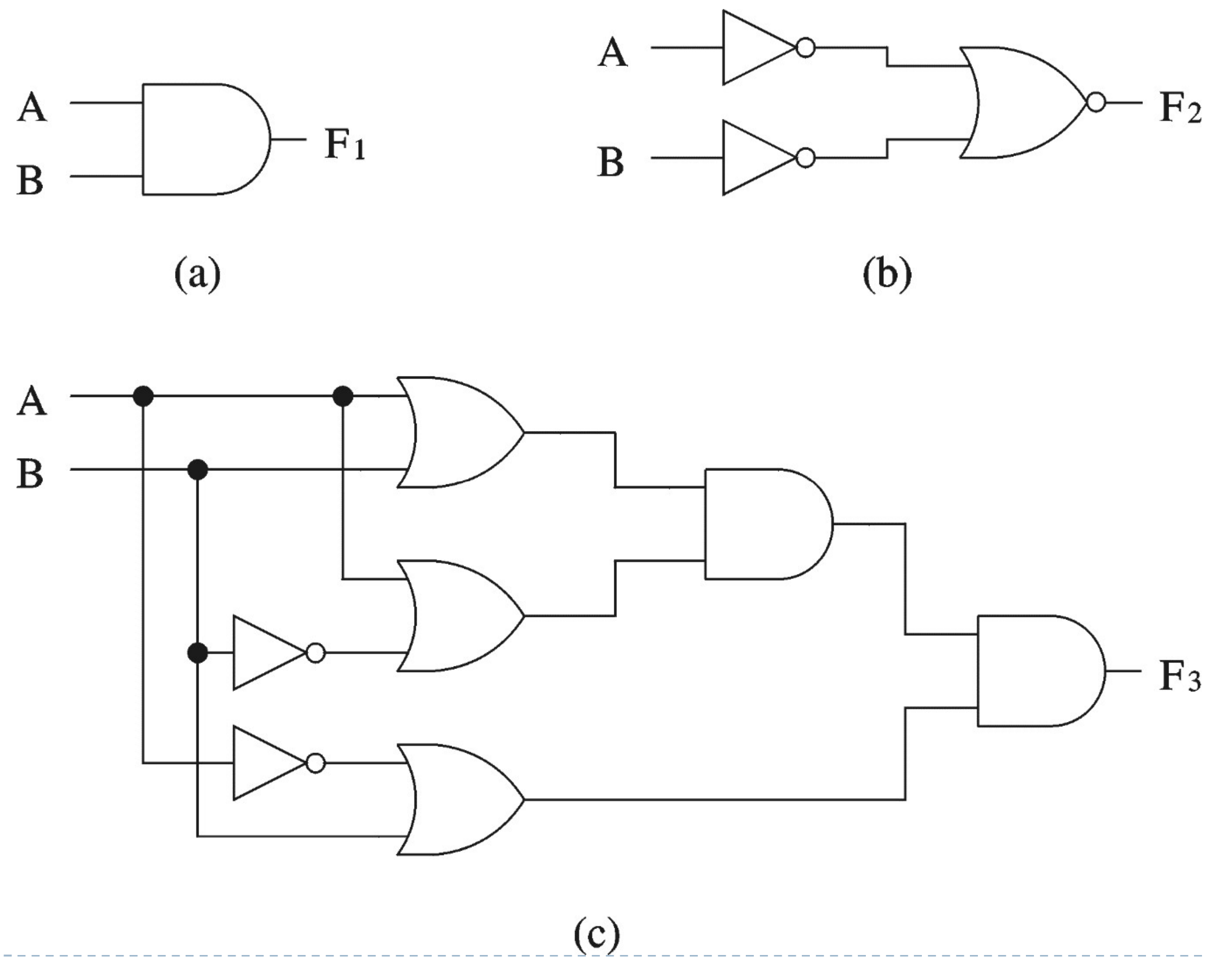


Notice that the XNOR gate will produce a HIGH when both inputs are the same. This makes it useful for comparison functions.

Logical Equivalence

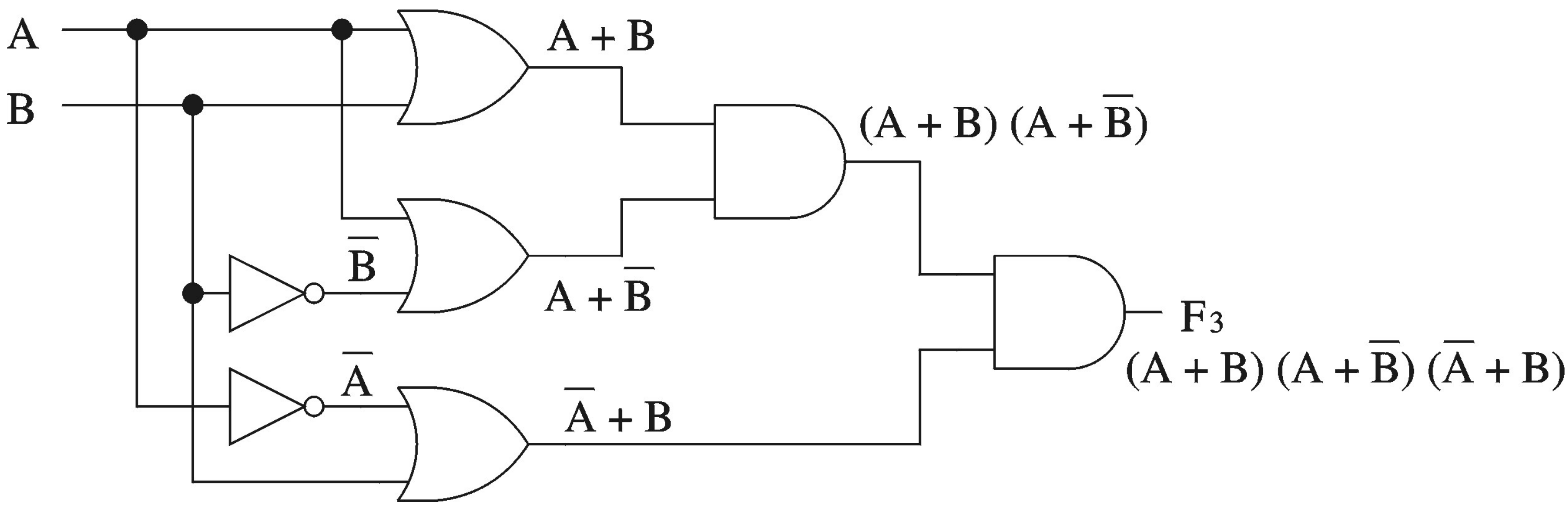
- When two circuits implement same logic function

Example: All three circuits implement F = A B function



Logical Equivalence ...

- Proving logical equivalence:
  - Derivation of logical expression from a circuit
    - Trace from the input to output
    - Write down intermediate logical expressions along the path
  - Build the truth table relating inputs to the output for each circuit



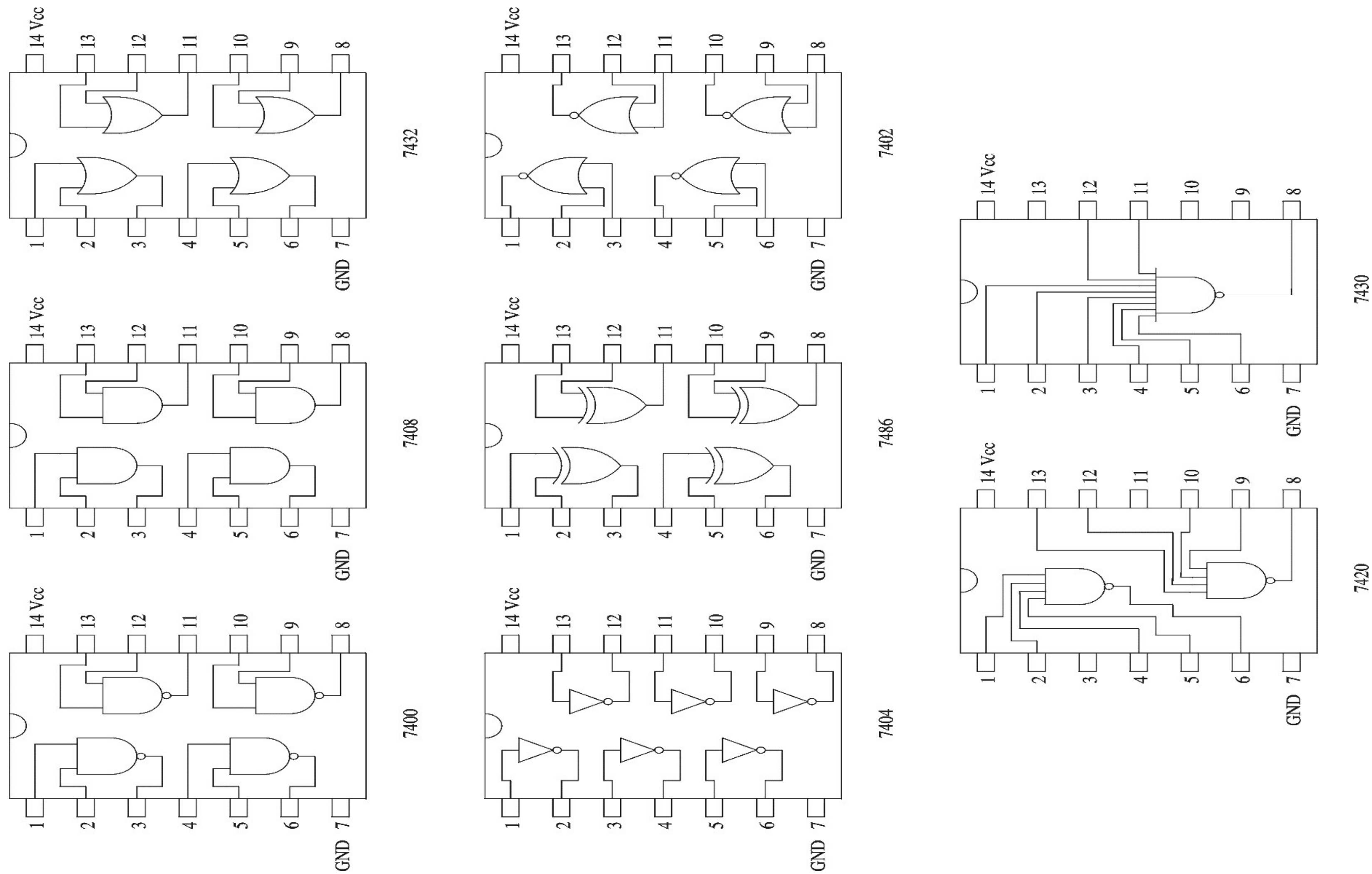
Logical Equivalence …

- Build the truth table for each circuit
- If each function give the same output, they are logically equivalent

A	B	F1 = A B	F3 = (A + B) ($\overline{A}$ + B) (A + $\overline{B}$)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

- Exercise:
  - Show that X⊕Y is logically equivalent to X'Y+XY'

Logic Chips



Logic Chips …

- Integration levels
  - SSI (small scale integration)
    - Introduced in late 1960s
    - 1-10 gates (previous examples)
  - MSI (medium scale integration)
    - Introduced in late 1960s
    - 10-100 gates
  - LSI (large scale integration)
    - Introduced in early 1970s
    - 100-10,000 gates
  - VLSI (very large scale integration)
    - Introduced in late 1970s
    - More than 10,000 gates