College of Computer and Information Sciences
Department of Computer Science

CSC 220: Computer Organization

Unit 3
Logic Functions

Logic Functions

- Number of functions
  - With N logical variables, we can define

    2^N combination of inputs
  - A function relates outputs to inputs
  - Some of them are useful
    - AND, NAND, NOR, XOR, …
  - Some are not useful:
    - Output is always 1
    - Output is always 0

Logic Functions

- Logical functions can be expressed in several ways:
  - Truth table
  - Logical expressions
  - Graphical form
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function

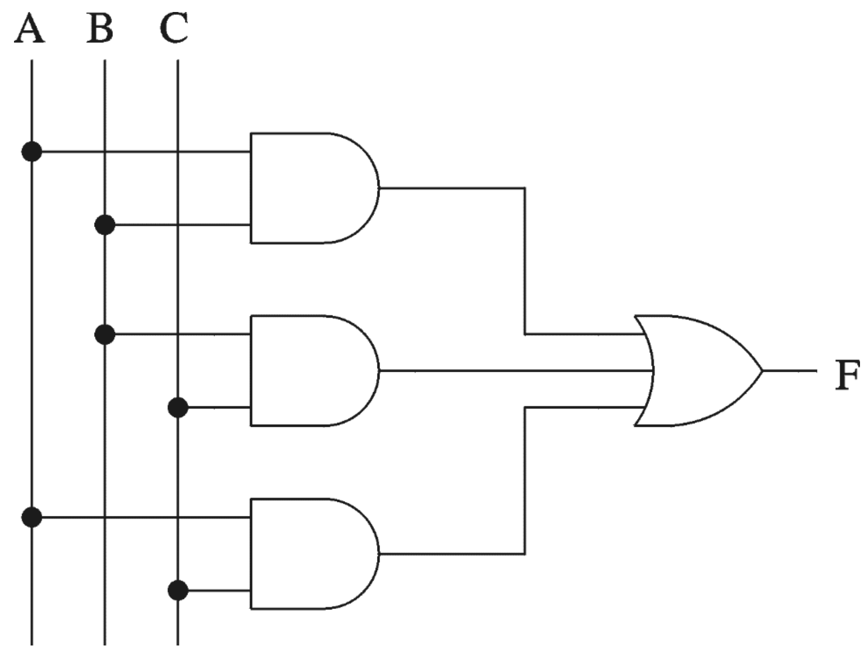
Logic Functions (cont.)

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

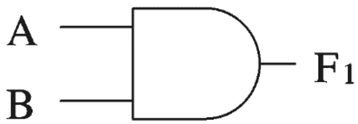
- Logical expression form

$$F = A\,B + B\,C + A\,C$$

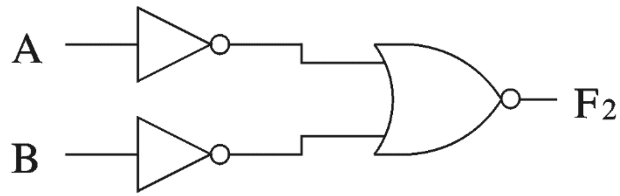


Logical Equivalence

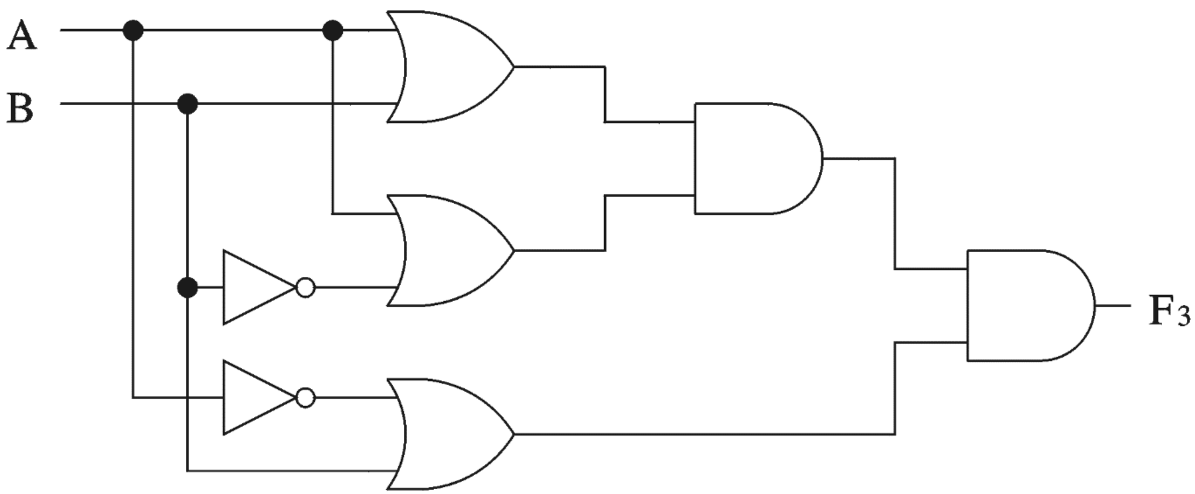
- All three circuits implement F = A B function



(a)



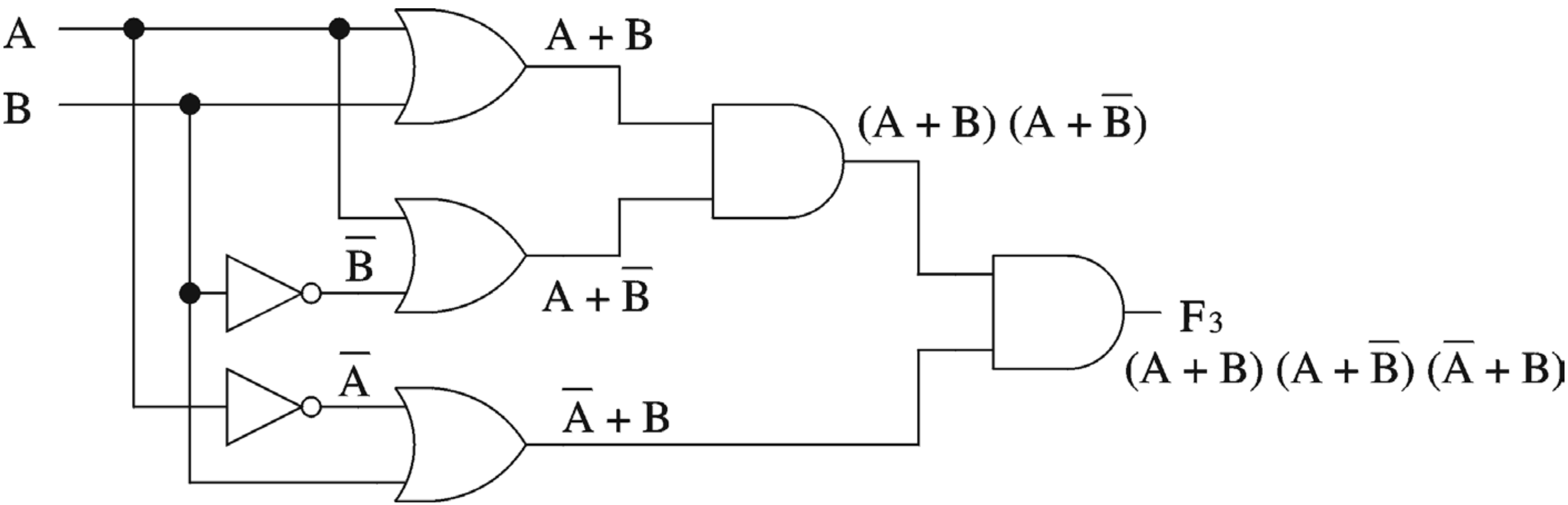
(b)



(c)

Logical Equivalence

- Derivation of logical expression from a circuit
  - Trace from the input to output
    - Write down intermediate logical expressions along the path



Logical Equivalence (cont.)

- Proving logical equivalence: Truth table method

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

- **Digital logic function :**
  - **SOP and POS forms**

Standard Forms for 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.

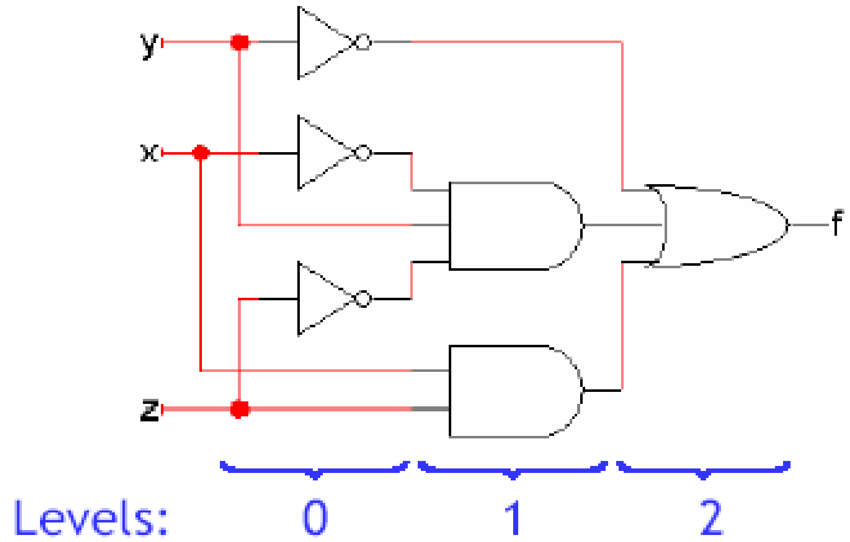
Sum-of-Products (SOP)

1 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.



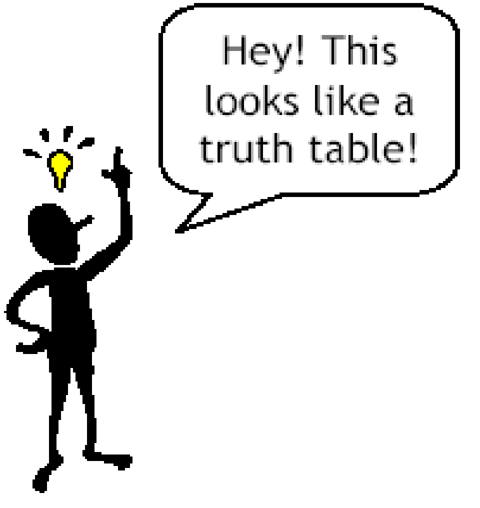
Minterms

- 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.

Minterm	True when	Shorthand
x'y'z'	xyz = 000	m_0
x'y'z	xyz = 001	m_1
x'y z'	xyz = 010	m_2
x'y z	xyz = 011	m_3
x y'z'	xyz = 100	m_4
x y'z	xyz = 101	m_5
x y z'	xyz = 110	m_6
x y z	xyz = 111	m_7



Minterms

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-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
  - Only the minterms for which $f_i = 1$ appear in the expression for function F.
  - $F = \Sigma (m_i) = \Sigma m(i)$

Denotes the logical sum operation

shorthand notation

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

- 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.

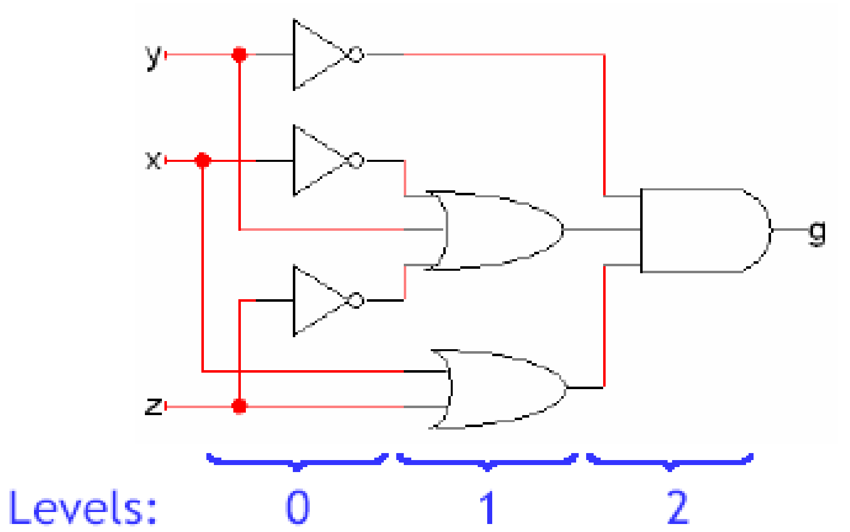
Product-of-Sums (POS)

2 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:

$$x + y + z \quad x + y + z' \quad x + y' + z \quad x + y' + z'$$
$$x' + y + z \quad x' + y + z' \quad x' + y' + z \quad x' + y' + z'$$

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

Maxterm	False when	Shorthand
$x + y + z$	xyz = 000	M_0
$x + y + z'$	xyz = 001	M_1
$x + y' + z$	xyz = 010	M_2
$x + y' + z'$	xyz = 011	M_3
$x' + y + z$	xyz = 100	M_4
$x' + y + z'$	xyz = 101	M_5
$x' + y' + z$	xyz = 110	M_6
$x' + y' + z'$	xyz = 111	M_7

Maxterms

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-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.
  - $F = \prod (M_i \,.\, f'_i)$
    - where M_i is a Maxterm
    - and f'_i is the complement of the corresponding functional output
  - Only the Maxterms for which $f_i = 0$ appear in the expression for function F.
  - $F = \prod (M_i) = \prod M(i)$

Denotes the logical product operation

shorthand notation

Product-of-Sums

- 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.

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.