Boolean Algebra

Product of Maxterms

- To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms.
  - This may be done by using the distributive law,

    $x + yz = (x + y)(x + z)$.
  - *Then any missing variable x in each OR term is ORed with xx'.*
- The product symbol, , denotes the ANDing of maxterms;
- An alternative procedure for deriving the maxterms of a Boolean function
  - obtain the truth table of the function directly from the algebraic expression and then read the maxterms from the truth table.

Product of Maxterms – Example

- Express the Boolean function *F = xy + x'z as a product of maxterms.*
  - Convert the function into OR terms by using the distributive law
    - $F = xy + x'z = (xy + x')(xy + z)$

      $= (x + x')(y + x')(x + z)(y + z) = (x' + y)(x + z)(y + z)$
  - The function has three variables: x, y, and z
  - Each OR term is missing one variable
    - $x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$
    - $x + z = x + z + yy' = (x + y + z)(x + y' + z)$
    - $y + z = y + z + xx' = (x + y + z)(x' + y + z)$
  - Combining all the terms and removing those which appear more than once;
    - $F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$

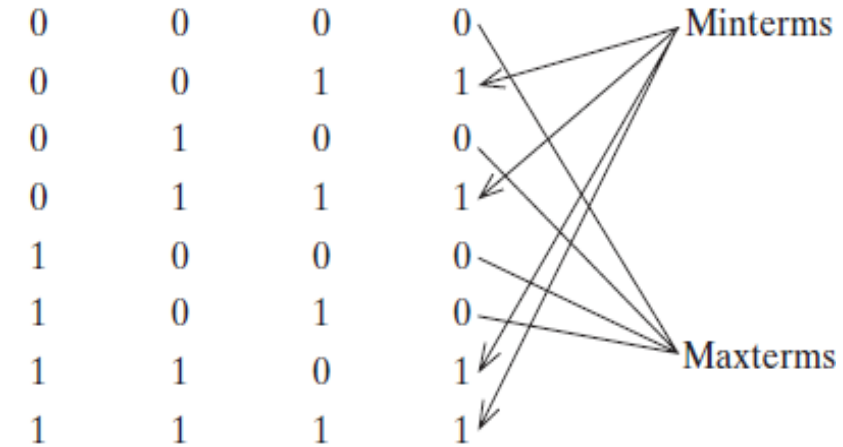
      $= M_0M_2M_4M_5$

    (ie) $F(x, y, z) = \prod(0, 2, 4, 5)$

Conversion between Canonical Forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.
- Boolean expression: *F = xy + x'z*
  - The function expressed as a sum of minterms as:
    - *F(x, y, z) =∑ (1, 3, 6, 7)*
  - The function expressed as a product of maxterms as:
    - *F(x, y, z) = ∏(0, 2, 4, 5)*

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Standard Forms

- Another way to express Boolean functions is in *standard form*
- *Problem with canonical form:*
  - minterm or maxterm must contain, by definition, *all the variables, either complemented or uncomplemented*
- The terms that form the function may contain one, two, or any number of literals.
- There are two types of standard forms:
  - sum of products
  - products of sums

Sum of Products(SOP) & Product of Sums(POS)

- SOP
  - The *sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each.*
  - *The sum denotes the ORing of these terms.*
  - *Example:*

    $$F1 = y' + xy + x'yz'$$

  - The logic diagram of a sum-of-products expression consists of a group of AND gates followed by a single OR gate.
- *POS*
  - A *product of sums is a Boolean expression containing OR terms, called sum terms.*
  - Each term may have any number of literals.
  - The *product denotes the ANDing of these* terms.
  - Example

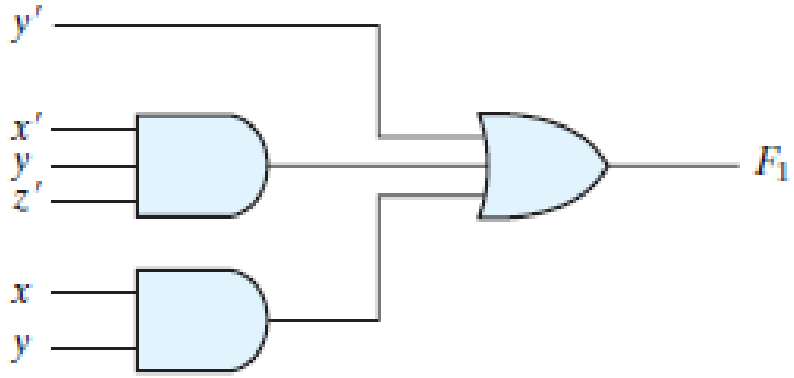
    $$F2= x(y' + z)(x' + y + z')$$

  - The logic diagram of a sum-of-products expression consists of a group of AND gates followed by a single OR gate.

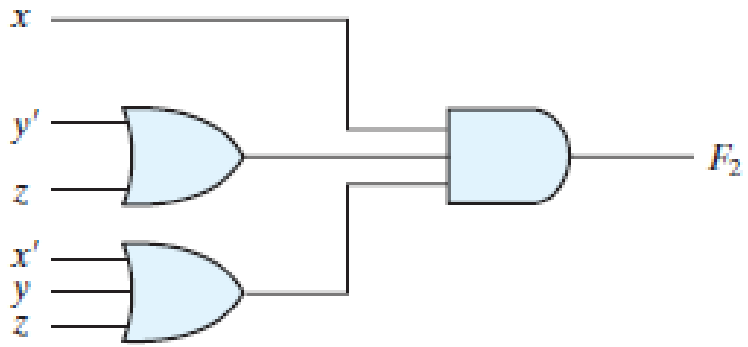
Two level Implementation

- In SOP
  - each product term requires an AND gate, except for a term with a single literal.
  - The logic sum is formed with an OR gate whose inputs are the outputs of the AND gates and the single literal.
- In POS
  - The gate structure of the product-of-sums expression consists of a group of OR gates for the sum terms (except for a single literal), followed by an AND gate.

- It is assumed that the input variables are directly available in their complements, so inverters are not included in the diagram.
- This circuit configuration is referred to as a *two-level implementation.*



Sum of Products



Product of Sums

Two level implementation

- In general, a two-level implementation is preferred because it produces the least amount of delay through the gates when the signal propagates from the inputs to the output.
- However, the number of inputs to a given gate might not be practical.

Nonstandard form

- *F3 = AB + C(D + E)*
  - is neither in sum-of-products nor in product-of-sums form.
  - It can be changed to a standard form by using the distributive law to remove the parentheses:

    F3= AB + C(D + E) = AB + CD + CE

Other Logic Functions

- When the binary operators AND and OR are placed between two variables, *x and y,* they form two Boolean functions, *x . y and x + y,*
- There are $\mathbf{2^{2^n}}$ *functions for n binary variables.*
  - *For two variables, n = 2,*
  - *The* number of possible Boolean functions is 16.
- The 16 functions listed can be subdivided into three categories:
  - Two functions that produce a constant 0 or 1.
  - Four functions with unary operations: complement and transfer.
  - Ten functions with binary operators that define eight different operations: AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication

Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Other Logic Operations

- Constants for binary functions can be equal to only 1 or 0.
- The complement function produces the complement of each of the binary variables.
- A function that is equal to an input variable has been given the name *transfer,*
  - *because the variable x or y is transferred* through the gate that forms the function without changing its value.
- Of the eight binary operators, two (inhibition and implication) are used by logicians,
  - but are seldom used in computer logic.
- The AND and OR operators have been mentioned in conjunction with Boolean algebra.
- The other four functions are used extensively in the design of digital systems.

Other Logic Operations

- The NOR function is the complement of the OR function,
  - and its name is an abbreviation of *not-OR.*
- *The NAND is the complement of AND*
  - *and is an* abbreviation of *not-AND.*
- *The exclusive-OR, abbreviated XOR, is similar to OR, but* excludes the combination of *both x and y being equal to 1;*
  - *it holds only when x and y* differ in value. (It is sometimes referred to as the binary difference operator.)
- Equivalence is a function that is 1 when the two binary variables are equal
  - (i.e., when both are 0 or both are 1).
- The exclusive-OR and equivalence functions are the complements of each other.
  - For this reason, the equivalence function is called exclusive-NOR, abbreviated XNOR.