# E&CE 223
# Digital Circuits & Systems

## Lecture Transparencies
## (Boolean Algebra & Logic Gates)

M. Sachdev





# Section 2: Boolean Algebra & Logic Gates

- **Major topics**
  - Boolean algebra
  - Boolean algebra theorems
  - Two valued Boolean algebra
  - Minterms
  - Sum-of-products
  - Maxterms
  - Product-of-sums
  - Karnaugh maps
  - Don't care conditions
  - Types of gates
  - NAND & NOR gates
  - AND-OR-INVERT
  - Prime implicants
  - Quine-McCluskey method

# Switching & Boolean Algebra

- **A branch of algebra used for describing and designing systems of two valued state variables**
  - Used by Shannon (1938) to design relay circuits
  - Basic concepts were applied to logic by Boole (1854) hence is known as Boolean algebra
  - Switching Algebra is two valued Boolean algebra
- **Boolean algebra**
  - A set B of elements {a,b,c .....} together with two binary operators (.) and (+) , form a Boolean algebra iff the following **postulates** hold (Huntington 1904):

**1.** There is closure wrt both (.) and (+); i.e. for all $a, b \in B$, we obtain a unique $c \in B$

e.g., c = a + b and c = a.b





**2.** There exists in B identity elements {0,1} wrt (+) and (.) such that

a + 0 = a and a.1 = a

**3.** (+) and (.) are commutative, i.e.,

a + b = b + a; a.b = b.a

**4.** (+) and (.) are distributive , i.e.,

a.(b+c) = (a.b) + (a.c); a+(b.c) = (a+b).(a+c)

**5.** For each element $a \in B$ , there exists an element $a' \in B$ such that

a + a’ = 1 and a.a’ = 0

Note: a’ is called complement of a [a’ is also written as $\bar{a}$ ]

**6.** There exists at least two elements $a, b \in B$ such that $a \neq b$

- In general, the number of elements may be $2^n$. Switching algebra and logic use the n = 1 case

# Basic Theorems of Boolean Algebra

- **Duality principle**
  - Every algebraic identity deducible from the postulates of Boolean algebra remains valid if binary operators (.) and (+), and the identity elements 0 and 1 are interchanged throughout
  - Proof

  Follows from the symmetric definition of Boolean algebra with respect to the two binary operators and respective identity elements, e.g.,

  a + b = b + a;   a.b = b.a;

  a + a' = 1   a.a' = 0





- **Uniqueness theorems [not in the book]**
  - The identity elements are unique
  - Proof

  Let * be either of the binary operators, and assume it has two identity elements $e_1$ and $e_2$

  Then for any $a \in B$, from Postulate 2 we have:

  $a*e_1 = a$

  $a*e_2 = a$

  Now, let $a = e_2$ in the first equation, and $a = e_1$ in the second equation, yielding

  $e_2*e_1 = e_2$

  $e_1*e_2 = e_1$

  From postulate 3 we have

  $e_2*e_1 = e_1*e_2$

  and hence $e_2 = e_1$

  QED

- **Theorem 1(a): x + x = x**

| | |
|---|---|
| x + x = (x + x).1 | by postulate 2 |
| = (x + x)(x + x’) | by postuate 5 |
| = x + xx’ | by postulate 4 |
| = x + 0 | by postulate 5 |
| = x | |

- **Theorem 1(b): x .x = x**

| | |
|---|---|
| x.x = x.x + 0 | by postulate 2 |
| = x.x + x.x’ | by postuate 5 |
| = x(x + x’) | by postulate 4 |
| = x.1 | by postulate 5 |
| = x | |

  - Note: the theorem 1(b) is dual of theorem 1(a)





- **Theorem 2(a): x + 1 = 1**

| | |
|---|---|
| x + 1 = 1.(x + 1) | by postulate 2 |
| = (x + x’).(x + 1) | by postuate 5 |
| = x + x’.1 | by postulate 4 |
| = x + x’ | by postulate 5 |
| = 1 | |

- **Theorem 2(b): x .0 = 0**
  - by duality
- **Theorem 3: (x’)’ = x**

1. Complement of x’ is (x’)’
2. From postulate 5 which defines the complement, we have

x’ + x = 1    and x’.x = 0

therefore x is the complement of x’

Since the complement is unique, (x’)’ = x

- **Theorem 4 (associative)**

  $x + (y + z) = (x + y) + z$ and $x.(y.z) = (x.y).z$

- **Theorem 5 (De Morgan's)**

  $(x + y)' = x'.y'$ and $(x.y)' = x' + y'$

- **The theorems involving 2 or 3 variables may be proven alebraically from postulates and theorems already proven**

- **Theorem 6(a): $x + x.y = x$ (absorption)**

| | |
|---|---|
| $x + x.y= x.1 + x.y$ | by postulate 2 |
| $= x.(1 + y)$ | by postuate 4 |
| $= x.(y + 1)$ | by postulate 3 |
| $= x.1$ | by postulate 5 |
| $= x$ | |

- **Theorem 6(b): $x.(x + y) = x$ by duality**





- **Theorem 4**

  ❍ Proof $x + (y + z) = (x + y) + z$

| | | | |
|---|---|---|---|
| Step 1: | $x + x.(y + z)$ | $= x$ | (T6) |
| | | $= x.(x + z)$(Th. 6) | |
| | | $= (x + x.y).(x + z)$ | (T6) |
| | | $= x + (x.y).z$ | (P4) |
| Step 2: | $x'+ x.(y.z)$ | $= (x' + x).(x' + y.z)$ | (P4) |
| | | $= 1.(x' + y.z)$ | (P5) |
| | | $= x' + y.z$ | (P2) |
| | | $= (x' + y).(x' + z)$ | (P4) |
| | | $= [1.(x' + y)].(x' + z)$ | (P2) |
| | | $= [(x' + x).(x' + y)].(x' + z)$ | (P5) |
| | | $= [x' +x.y].(x' + z)$ | (P4) |
| | | $= x' + (x.y).z$ | |

Step 3: Take (.) operation of the left hand side terms above

$[x + x.(y + z)].[x' + x.(y.z)] = [x + (x.y).z].[x' + (x.y).z]$

Then

| | | |
|---|---|---|
| $xx' + x.(y.z)$ | $= xx' + (x.y).z$ | (P4) |
| $0 + x.(y.z)$ | $= 0 + (x.y).z$ | (P5) |
| $x.(y.z)$ | $= (x.y).z$ QED | (P2) |

- **Theorem 5**

$(x + y)' = x'.y'$ and $(x.y)' = x' + y'$

| | | |
|---|---|---|
| 1. $(xy)(x'+y')$ | $= (xy)x' + (xy)y'$ | (P4) |
| | $= (xx')y + x(yy')$ | (Th4 & P3) |
| | $= 0.y + x.0$ | (P5) |
| | $= 0 + 0$ | (Th2) |
| | $= 0$ | (P2) |
| 2. $(x+y)+(x'+y')$ | $= [x+(x'+y')].[y+(x'+y')]$ | (P4) |
| | $= [(x+x') + y'].[(y+y') + x']$ | (Th4 & P3) |





| | | |
|---|---|---|
| | $= [1 + y'].[1 + x']$ | (P5) |
| | $= 1.1$ | (Th2) |
| | $= 1$ | (P2) |

- **therefore, by uniqueness of complement and P5**

$(x.y)' = x' + y'$

other relation follows from duality

# Two Valued Boolean Algebra

- **B = {0,1}**
- **Definition of (.) and (+) operations and of complements**

| a | b | a.b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| a | b | a+b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| a | a’ |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 0 | 1 |
| 1 | 0 |

| . | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

a.b

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

a+b





# Boolean Algebra: Applications

- **Logic**

| | |
|---|---|
| 1 | True |
| 0 | False |
| . | AND $A \wedge B$ |
| + | OR $A \vee B$ |
| ‘ | NOT |

- **Algebra of sets (classes)**

| | |
|---|---|
| 1 | $I$ (universal set) |
| 0 | $\varnothing$ (null set) |
| . | Intersection $A \wedge B$ |
| + | Union $A \vee B$ |
| ‘ | Complement (universal set less current set) |

- **Switching algebra**

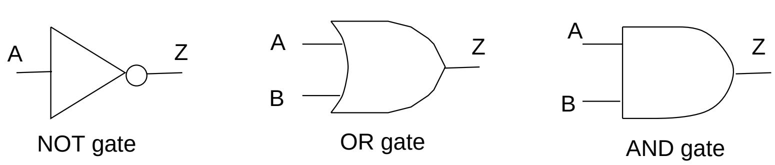
| | |
|---|---|
| 1 | High voltage (true) |
| 0 | Low voltage (false) |
| . | AND |
| | Output is high voltage iff **all inputs** are high voltage |
| + | OR |
| | Output is high voltage if **any** input is high voltage |
| ‘ | NOT (Inverter) |
| | Output is low voltage if input is high voltage |
| | Output is high voltage if input is low voltage |





# Digital Logic Gates (positive logic)

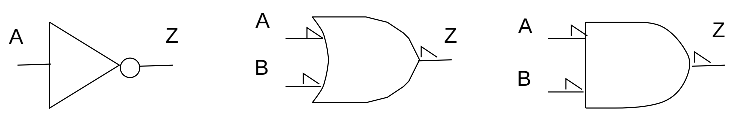
| | |
|---|---|
| 1 | High voltage (true) |
| 0 | Low voltage (false) |
| . | AND gate |
| | Output is high voltage iff **all inputs** are high voltage |
| + | OR gate |
| | Output is high voltage if **any** input is high voltage |
| ‘ | NOT (Inverter) gate |
| | Output is low voltage if input is high voltage |
| | Output is high voltage if input is low voltage |




NOT gate OR gate AND gate

# Digital Logic Gates (negative logic)

| | |
|---|---|
| 1 | Low voltage (true) |
| 0 | High voltage (false) |
| . | AND gate |
| | Output is low voltage iff **all inputs** are low voltage |
| + | OR gate |
| | Output is low voltage if **any** input is low voltage |
| ‘ | NOT (Inverter) gate |
| | Output is low voltage if input is high voltage |
| | Output is high voltage if input is low voltage |




NOT gate OR gate AND gate



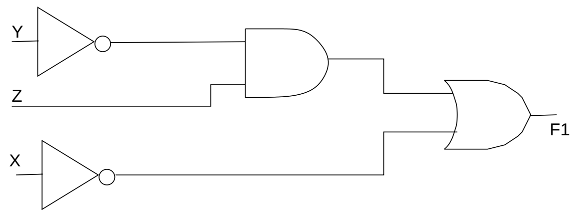


# Representation of Boolean Function

- **Boolean functions are represented as**
  - **Algebraic expressions:** $F_1 = f(x,y,z)$ $F_2 = x' + y'z$
  - **Truth table**

| **x** | **y** | **z** | **$F_1$** |
|---|---|---|---|
| 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 | 0 |

# Boolean Function: Implementation




- **Complement of a function**

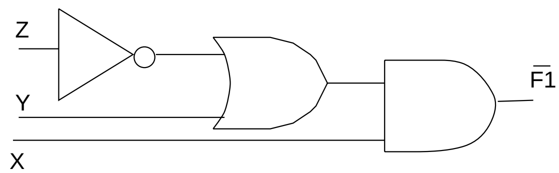
  F'( )= 1 if F( ) = 0 and F'( ) = 0 if F( ) = 1

  ❍ If F is expressed algebraically, F' is obtained by (repeated) applicayions of De Morgan's theorem, e.g.,

  F1 = x' + y'z

  F1' = (x' + y'z)' = x.(y'z)'

  1








# Extension of De Morgan's Theorem

(abcde ........)' = a' + b' +c' +d' + e' + ........

❍ and

(a + b + c + d + e + .....)' = a'b'c'd'e'.......

- **If the expression has parenthesis, apply De Morgan's theorem to the terms in the parenthesis, e.g.,**

  [ab(c + d)e]' = a' + b' + (c + d)' + e'

  = a' + b' + c'd' + e'

- **In general**

  [$f$(x,y,z,.,+)]' = $f$(x',y',z',+,.)

# Canonical Forms

- **Minterms**
  - A minterm is an AND term in which every literal (variable or its complement) in a function occurs once;
  - For n variables, there are $2^n$ minterms
  - Each minterms has a value of 1 for exactly one combination of values of the n variables, e.g., n = 3





| x y z | Corresponding minterm | designation |
|---|---|---|
| 0 0 0 | x'y'z' | $m_0$ |
| 0 0 1 | x'y'z | $m_1$ |
| 0 1 0 | x'yz' | $m_2$ |
| 0 1 1 | x'yz | $m_3$ |
| 1 0 0 | xy'z' | $m_4$ |
| 1 0 1 | xy'z | $m_5$ |
| 1 1 0 | xyz' | $m_6$ |
| 1 1 1 | xyz | $m_7$ |

- **One method of writing a Boolean function is in the canonical minterm form ( canonical sum of products form), e.g.**

F = x'y'z + xy'z + xyz'
$= m_1 + m_5 + m_6$
$= \sum(1, 5, 6)$

- **The canonical sum of products of F can be written directly from the truth table**

| x y z | F1 | Correspond-ing minterm |
|---|---|---|
| 0 0 0 | 1 | $m_0$ |
| 0 0 1 | 1 | $m_1$ |
| 0 1 0 | 1 | $m_2$ |
| 0 1 1 | 1 | $m_3$ |
| 1 0 0 | 0 | |
| 1 0 1 | 1 | $m_5$ |
| 1 1 0 | 0 | |
| 1 1 1 | 0 | |

F1 = $\sum(0, 1, 2, 3, 5)$

= $m_0 + m_1 + m_2 + m_3 + m_5$

= x'y'z' + x'y'z + x'yz' + x'yz + xy'z





- **The canonical sum of products of F' can also be written directly from the truth table**

F' = $\sum$ (all minterms not in F)

| x y z | F1 | Correspond-ing minterm |
|---|---|---|
| 0 0 0 | 1 | $m_0$ |
| 0 0 1 | 1 | $m_1$ |
| 0 1 0 | 1 | $m_2$ |
| 0 1 1 | 1 | $m_3$ |
| 1 0 0 | 0 | |
| 1 0 1 | 1 | $m_5$ |
| 1 1 0 | 0 | |
| 1 1 1 | 0 | |

F'1 = $\sum(4, 6, 7)$ (from previous example) = $m_4 + m_6 + m_7$

= xy'z' + xyz' + xyz

# Maxterms

- **A maxterm is an OR term in which every literal (variable or its complement) in a function occurs once.**
  - Each maxterm has a value of 0 for one combination of values of the n variables

| x y z | Corresponding minterm | designation |
|---|---|---|
| 0 0 0 | x + y + z | $M_0$ |
| 0 0 1 | x + y + z’ | $M_1$ |
| 0 1 0 | x + y’ + z | $M_2$ |
| 0 1 1 | x + y’ + z’ | $M_3$ |
| 1 0 0 | x’ + y + z | $M_4$ |
| 1 0 1 | x’ + y + z’ | $M_5$ |
| 1 1 0 | x’ + y’ + z | $M_6$ |
| 1 1 1 | x’ + y’ + z’ | $M_7$ |





# Maxterms and Minterms

- **Different arrangements for minterms and maxterms**

  $m_0$ = x’y’z’ = (x + y + z)’ = $M'_0$

  In general, $m_i = M'_i$

- **An alternate method of writing a Boolean function is the canonical maxterm form (canonical product of sums form)**

  F2 = (x + y + z)(x + y’ + z)(x + y’ + z’)(x’ + y + z)(x’ + y’ + z’)

  = $M_0M_2M_3M_4M_7$ = $\Pi(0, 2, 3, 4, 7)$

- **The canonical product of sums can be written directly from the truth table**

| x y z | F2 | Correspond-ing maxterm |
|---|---|---|
| 0 0 0 | 0 | $M_0$ |
| 0 0 1 | 1 | |
| 0 1 0 | 0 | $M_2$ |
| 0 1 1 | 0 | $M_3$ |
| 1 0 0 | 0 | $M_4$ |
| 1 0 1 | 1 | |
| 1 1 0 | 1 | |
| 1 1 1 | 0 | $M_7$ |

- F2 = (x + y + z)(x + y’ + z)(x + y’ + z’)(x’ + y + z)(x’ + y’ + z’)
- similarly, F’2 = $\Pi$ (all maxterms not in F2)
-





# Standard Forms

- **In canonical forms, each minterm (or maxterm) must contain all the variables, either true or complemented**
  - These forms can be simplified
  - In **standard forms**, terms may contain one, two, three, ... variables
  - Two types of standard forms (i) sum of products; (ii) product of sums
- **Examples**

  F1 = xy + y’z (sum of products, standard form)

  F2 = (x+y’)(y+z) (product of sums, standard form)

# Canonical vs Standard forms

- Standard forms are converted into canonical forms by use of identity elements, complement, and distributive postulates

F1 = xy + y'z (standard form)

= xy.1 + 1.y'z

= xy(z + z') + (x + x')y'z

= xyz + xyz' + xy'z + x'y'z (canonical form)

- Non standard forms are converted to standard forms in the same fashion

F3 = (xy + z)(xz + y'z) (non-standard form)

= xy(xz + y'z) + z(xz +y'z)

= xyz + xyy'z + xz + y'z

= xyz + xz + y'z

= xz + y'z (standard form)





# Simplification of Boolean Functions

- **Algebraic** - frequently tricky
- **Karnaugh map** - functions of 2,3,4,(5,6) variables
- **Tabular (Quinne-McClusky) method** - more than 4 variables (computer programs)

■ **Algebra**

F1 = x + x'y

= (x +x')(x + y)

= x + y

F2 = xy + yz + zx'

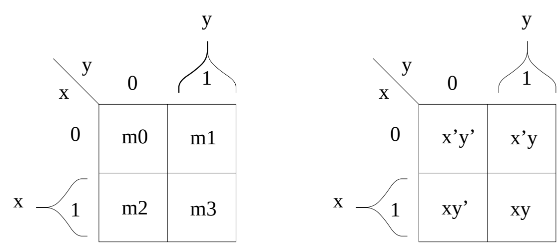
= xy + (x + x')yz + zx'

= xy(1 + z) + x'yz + zx'

= xy + x'z

# Karnaugh Maps (K Maps)

- **A Karnugh map is a graphical representation of a truth table**
  - The map contains one cell for each possible minterm
  - adjacent cells differ in only one literal, i.e., x or x’
- **Two variables, F =*f*(x,y)**



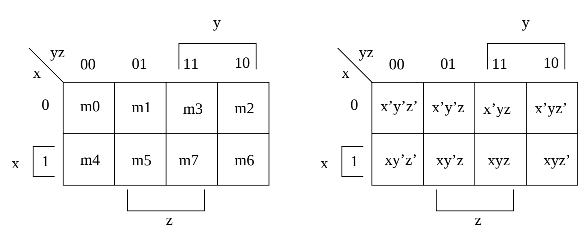

  - Function is plotted by placing 1 in cells corresponding to mint-erms of function
  - Example, F = x’y

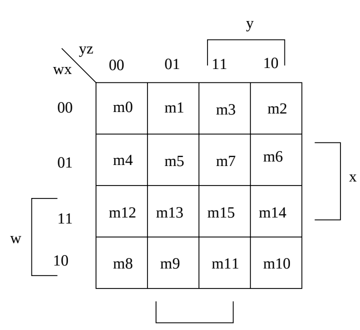




# K Maps with 3 and 4 Variables

- **3 variables, F = *f*(x,y,z); 4 variables, F = *f*(w,x,y,z)**



# Examples

F = w = x(x+x')(y+y')(z+z')

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | | |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |

F = x'y'

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | | |
| 11 | | | | |
| 10 | | | | |

F = wx'z

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | | |
| 11 | | | | |
| 10 | | | | |





# K Map Boolean Funct. Simplification

- **To write simplified function, find maximum size groups (minimum literals) that *cover all 1s* in map**
  - 8 cells --> single literal
  - 4 cells --> two literals
  - 2 cells --> three literals
  - 1 cell --> four literals
- **Guidelines for logic synthesis**
  - **Fewer groups:** fewer AND gates and fewer input to the OR gate
  - **Fewer literals (larger group):** fewer inputs to AND gate
- **Synthesis (design) objectives**
  - Smallest number of logic gates
  - Number of inputs to logic gate

# Example

- **Consider the following K map**
  - Nothing must be a single cell
  - Four groups of two cells each
  - nothing left uncovered
- **The group of 4 (xz) term is not needed**

F = w'xy' + wxy + w'yz + wy'z

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | |
| 01 | 1 | 1 | 1 | |
| 11 | | 1 | 1 | 1 |
| 10 | | 1 | | |





# Product of Sum Expression

- **Recall: Let F be the function**

  F' = $\sum$ (all minterms not in F)

  F = $\Pi$ (all minterms not in F)' (de morgan's theorem)
  - Therefore, one can obtain F' by grouping all 0s on K map, and then taking the complement to obtain product-of-sum form
  - Hence,

  F = (w' + x')(y' + z')(x' + z)

  in sum-of-product form
- **Should check both, sum of products, and product of sums**
  - One is often simpler than the other

F = x'z' + x'y' + w'y'z'

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | 0 | 1 |
| 01 | 0 | 1 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 1 | 1 | 0 | 1 |

F' = wx + yz + xz'

# Plotting Product of Sum

- Given, F = (w + x)(x + y’ + z)( y + z)

  F’ = w’x’ + x’yz’ +y’z’

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | | |
| 11 | | | | |
| 10 | | | | |





# Don’t Care (Incompletely Specified) Conditions

- **Some times, not all values of a function are defined**
  - Some input conditions will never occur
  - We don’t care what the output is for that input condition
- **In these cases, we can choose the output to be either 0 or 1, whichever simplifies the circuit**
  - **Example:** a circuit is to have an output of 1 if a binary coded decimal (BCD) digit is a multiple of 3
  -

| digit | w x y z | F |
|---|---|---|
| 0 | 0 0 0 0 | 0 |
| 1 | 0 0 0 1 | 0 |

| | | | |
|---|---|---|---|
| 2 | 0 0 1 0 | 0 | |
| 3 | 0 0 1 1 | 1 | |
| 4 | 0 1 0 0 | 0 | |
| 5 | 0 1 0 1 | 0 | |
| 6 | 0 1 1 0 | 1 | |
| 7 | 0 1 1 1 | 0 | |
| 8 | 1 0 0 0 | 0 | |
| 9 | 1 0 0 1 | 1 | |
| | 1 0 1 0 | - | don't care condition |
| | 1 0 1 1 | - | ,, |
| | 1 1 0 0 | - | ,, |
| | 1 1 0 1 | - | ,, |
| | 1 1 1 0 | - | ,, |
| | 1 1 1 1 | - | ,, |

- F = $\sum$(3,6,9) + d(10,11,12,13,14,15)





# Don't Care: Plotting

- **Don't cares are plotted as X in the K map**
  - **Sum of products:** treat X as 1 if it allows a larger group
  - **Product of sums:** Treat X as 0 of it allows a larger group
  - F1 = wz + xyz' + x'yz (sum of products, (a))
  - F2' = xz + w'y' + x'z' (recall F' = $\sum$ (all minterms not in F)
  - F2 = (x' + z')(w + y)(x + z)

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | |
| 01 | | | | 1 |
| 11 | X | X | X | X |
| 10 | | 1 | X | X |

(a)

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | | 0 |
| 01 | 0 | 0 | 0 | |
| 11 | X | X | X | X |
| 10 | 0 | | X | X |

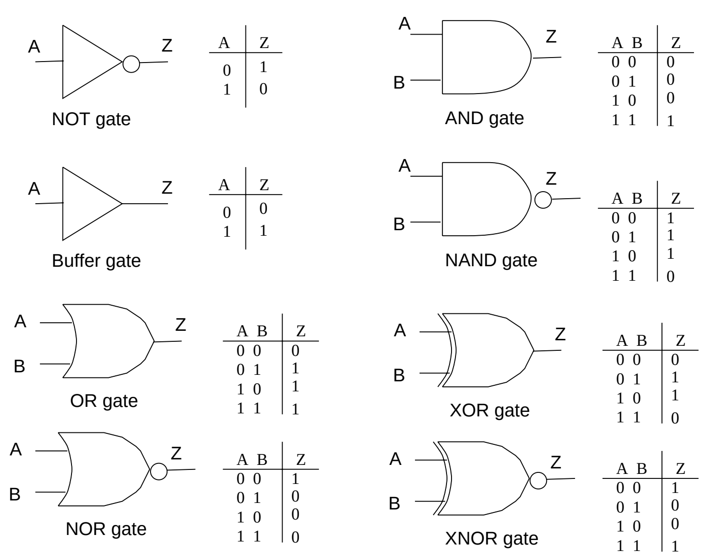
(b)

- **Observation:** In general, F1 is not equal to F2 due to different values chosen for don’t care cells





# More Logic Gates




NOT gate

| A | Z |
|---|---|
| 0 | 1 |
| 1 | 0 |

Buffer gate

| A | Z |
|---|---|
| 0 | 0 |
| 1 | 1 |

OR gate

| A B | Z |
|---|---|
| 0 0 | 0 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 1 |

NOR gate

| A B | Z |
|---|---|
| 0 0 | 1 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 0 |

AND gate

| A B | Z |
|---|---|
| 0 0 | 0 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |

NAND gate

| A B | Z |
|---|---|
| 0 0 | 1 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 0 |

XOR gate

| A B | Z |
|---|---|
| 0 0 | 0 |
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 0 |

XNOR gate

| A B | Z |
|---|---|
| 0 0 | 1 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |

# NAND and NOR Implementation

- ❍ A set of logic gates are **functionally complete** if any boolean function can be implemented by just these gates
- ❍ AND, OR, NOT
- ❍ AND, NOT

  $(x'y')' = x+y$ ==> OR gate
- ❍ OR, NOT
- ❍ NAND
- ❍ NOR

■ **NAND and NOR gates are easier to implement (smaller area, less power consumption, faster) than AND and OR gates**





# Logic Implementation with NAND/NOR

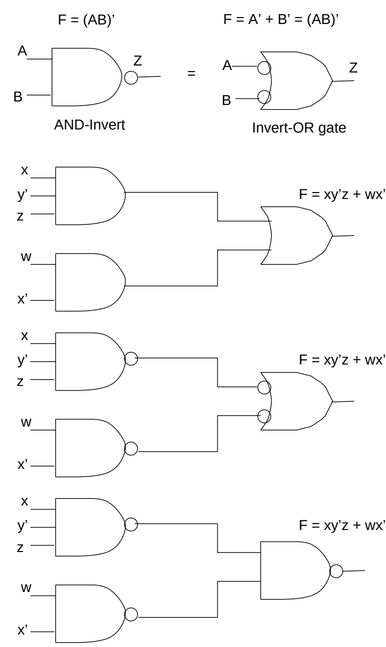
■ **Given F = xy'z + wx'**

- ❍ all implementations represent the same function
- ❍ Function can be implemented with NAND gates only

■ **Procedure from K map**

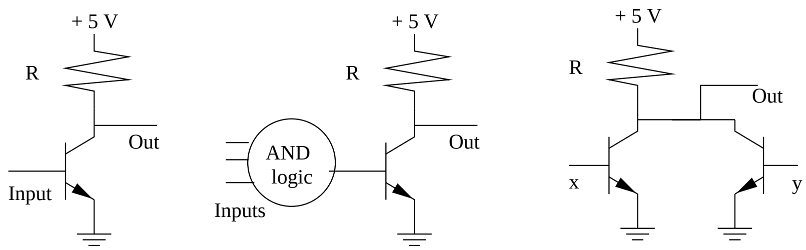
- ❍ present the simplified function in sum of product form (AND-OR)
- ❍ use De morgan's theorem to represent the function in NAND-NAND form

■ **Similar steps for NOR implementation starting from product of sums form**



# Other Two-Level Implementations

- **Wired Logic, Transistor-Transistor Logic (TTL)**
  - **Wired logic:** if outputs of two logic gates are shorted together
  - TTL style implementation allows wired connection




NOT gate    NAND gate    wired AND gate

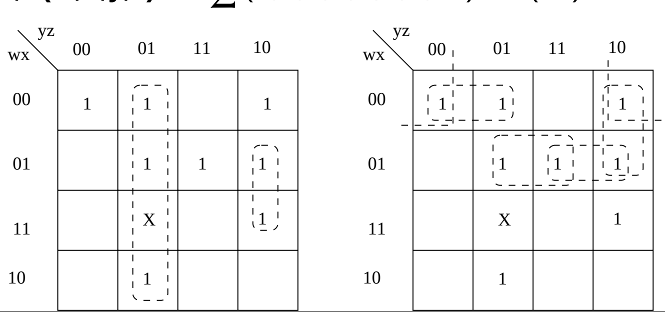
  - Other two level implementations are AND-OR-INVERT and OR-AND-INVERT





# Simplest Two-Level Expression

- **Some definitions**
  - **Implicant:** a grouping of one or more K map cells
  - **Prime implicant:** an implicant that is not a subset of another implicant
  - **Essential prime implicant:** a prime implicant that covers at least one minterm not covered by another prime implicant
- **Example, f(w,x,y,z) =** $\sum(0,1,2,5,6,7,9,14) + d(13)$




essential prime implicants    prime implicants

- ❍ **Essential prime implicants:** y’z, xyz’
- ❍ **Prime implicants:** w’x’y’, w’x’z’, w’xz, w’xy, w’yz’

minterms covered

| | 0 | 1 | 2 | 5 | 6 | 7 | 9 | 14 | |
|---|---|---|---|---|---|---|---|---|---|
| y’z (1,5,9,13) | | ✓ | | ✓ | | | ✓ | | * |
| xyz’ (6,14) | | | | | ✓ | | | ✓ | * |
| w’x’y’ (0,1) | ✓ | ✓ | | | | | | | A |
| w’x’z’ (0,2) | ✓ | | ✓ | | | | | | B |
| w’xz (5,7) | | | | ✓ | | ✓ | | | C |
| w’xy (6,7) | | | | | ✓ | ✓ | | | D |
| w’yz’ (2,6) | | | ✓ | | ✓ | | | | E |

■





- ■ **All minterms must be covered**
  - ❍ Essential prime imlicants must be included (*)
  - ❍ Different combinations of prime implicants are:

    B + C; or B + D; or A + C + E; or A + D + E
- ■ **B + C or B + D are the simplest, hence the simplest function implementation is**
  - ❍ F = y’z + xyz’ + w’x’z’ + w’xz or y’z + xyz’ + w’x’z’ + w’xy

# Tabulation (Quine-McCluskey) Method

- **The map method of simplification is convenient if number of variables does not exceed beyond 4 or 5**
  - Tabulation method is preferred for a function with large number of variables
  - for F = f(w,x,y,z) consider two adjacent minterms

    let a = m4 + m5 = w'xy'z' + w'xy'z = w'xy'

    or = 0100 + 0101 = 010-
  - similarly, let b = m12 + m13 = wxy'z' + wxy'z = wxy'

    or = 1100 + 1101 = 110-
  - similarly, c = m4 + m5 + m12 + m13 = a + b

    = w'xy' + wxy' = xy'

    = 010- + 110- = -10-





  - Adjacent minterms differ by a single bit in their binary representation
  - Tabulation method consists of grouping minterms and systematically checking for single bit differences
- **Example, f(w,x,y,z) = $\sum$(0,3,4,6,7,8,10,11,15) + d(5,9)**
  - Group minterms according to number of 1's in binary representation
  - Each element of each section is compared with each element of the section below it; all reductions are recorded in next column
  - Mark terms that combine
  - All unmarked terms are prime implicants

```
     w   x   y   z
0    0   0   0   0
--   -----------------
4    0   1   0   0
8    1   0   0   0
--   --------------------
3    0   0   1   1
5    0   1   0   1
6    0   1   1   0
9    1   0   0   1
10   1   0   1   0
--   -------------------
7    0   1   1   1
11   1   0   1   1
--   -----------------
15   1   1   1   1
```





```
 0     0,4    (4)    4,5,6,7       (1,2)
----   0,8    (8)    8,9,10,11     (1,2)
4      ------------  ----------------------
8      4,5    (1)    3,7,11,15     (4,8)
----   4,6    (2)
3      8,9    (1)
5      8,10   (2)
6      -----------
9      3,7    (4)
10     3,11   (8)
----   5,7    (2)
7      6,7    (1)
11     9,11   (2)
---    10,11  (1)
15     ----------
       7,15   (8)
       11,15  (4)
```

| Prime implicants | minterms covered 0 | 3 | 4 | 6 | 7 | 8 | 10 | 11 | 15 |
|---|---|---|---|---|---|---|---|---|---|
| 0,4 | ✓ | | ✓ | | | | | | |
| 0,8 | ✓ | | | | | ✓ | | | |
| 4,5,6,7 | | | ✓ | ✓ | ✓ | | | | |
| 8,9,10,11 | | | | | | ✓ | ✓ | ✓ | |
| 3,7,11,15 | | ✓ | | | ✓ | | | ✓ | ✓ |

- F(w,x,y,z) = 0,4 + 4,5,6,7 + 8,9,10,11 + 3,7,11,15
  0-00 + 01-- + 10-- + --11
  w'y'z' + w'x + wx' + yz
- or F(w,x,y,z) = 0,8 + 4,5,6,7 + 8,9,10,11 + 3,7,11,15
  -000 + 01-- + 10-- + --11
  x'y'z' + w'x + wx' + yz