COMBINATIONAL LOGIC DESIGN

Part 3

Digital Design and Computer Architecture, 2nd Edition

David Money Harris and Sarah L. Harris

ELSEVIER

Boolean Algebra 1/2

- A set of elements B
  - There exist <u>at least</u> **two** distinct elements $x, y \in B$ s. t. $x \neq y$
- Binary operators: + and ·

  <u>closure</u> w.r.t. both + and ·

  $x, y \in B, (x+y) \in B, (x \cdot y) \in B$

  + (additive) identity ?

  $0: \; x + 0 = 0 + x = x$

  · (multiplicative) identity ?

  $1: \; x \cdot 1 = 1 \cdot x = x$

  commutative w.r.t. both + and ·

  $x + y = y + x \qquad x \cdot y = y \cdot x$

  associative w.r.t. both + and ·

  $x+(x+y) = (x+x)+y \qquad x \cdot (x \cdot y) = (x \cdot x) \cdot y$
- Distributive law:

  is · distributive over + ?

  yes : $x \cdot (y+z) = (x \cdot y)+(x \cdot z)$

  is + distributive over · ?

  yes : $x+(y \cdot z) = (x+y) \cdot (x+z)$

  We do not have the second one in ordinary algebra

Boolean Algebra 2/2

- Complement
  - $\forall x \in B$, there exist an element $x' \in B \ni$
    - $x + x' = 1$ ($\cdot$ identity) and
    - $x \cdot x' = 0$ (+ identity)
  - Not available in ordinary algebra
- No inverses in Boolean Algebra!
- Differences between ordinary and Boolean algebra
  - Ordinary algebra deals with real numbers (infinite)
  - Boolean algebra deals with elements of set B (finite)
  - Complement
  - Distributive law
  - **Do not substitute laws from one to another where they are not applicable**

Two-Valued Boolean Algebra (Switching Algebra)

- B = {0, 1}
- Check the axioms
  - Two distinct elements, $0 \neq 1$
  - Closure, associative, commutative, identity elements
  - Complement

    $x + x' = 1$ and $x \cdot x' = 0$
  - Distributive law

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

$y \cdot z$	$x + (y \cdot z)$

$x + y$	$x + z$	$(x + y) \cdot (x + z)$

Two-Valued Boolean Algebra

- Two-valued Boolean algebra is actually equivalent to the binary logic that we defined heuristically before

  Operations:

  - · → AND
    - we often drop · and write xy instead of x · y
  - + → OR
  - Complement → NOT

- Binary logic is the application of Boolean algebra to gate-type circuits
  - Two-valued Boolean algebra is developed in a formal mathematical manner
  - This formalism is necessary to develop theorems and properties of Boolean algebra

Duality Principle

- An important principle
  - every algebraic expression deducible from the axioms of Boolean algebra remains valid if the operators and identity elements are interchanged together
- Example:
  - $x + x = x$
  - $x + x = (x+x) \cdot 1$ (identity element)
    $= (x+x) \cdot (x+x')$ (complement)
    $= x+(x \cdot x')$ (+ over ·)
    $= x$ (complement)
  - duality principle
    $x + x = x \;\Rightarrow\; x \cdot x = x$

Duality Principle & Theorems

- Theorem a:
  - $x + 1 = 1$ hmmm?
  - $$\begin{aligned} x + 1 &= (x + 1) \cdot 1 \\ &= (x + 1) \cdot (x + x') \\ &= x + (1 \cdot x') \\ &= x + x' \\ &= 1 \end{aligned}$$
- Theorem b: (using duality)
  - $x \cdot 0 = 0$

Absorption Theorem

$\mathbf{x + xy = x}$ hmmmmm?

$= x \cdot 1 + x \cdot y$

$= x \cdot (1+y)$

$= x \cdot 1$

$= x$

Involution Theorem

$(x')' = x$

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

Involution Theorem

$(x')' = x$

- $x + x' = 1$ and $x \cdot x' = 0$
- Complement of x' is x
- Complement is unique

DeMorgan's Theorem

$\mathbf{(x + y)' = x' \cdot y'}$

Using duality >> $\mathbf{(x \cdot y)' = x' + y'}$

Truth Tables for DeMorgan's Theorem

$(x + y)' = x' \cdot y'$

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

x'	y'	$x' \cdot y'$	$x' + y'$
1	1		
1	0		
0	1		
0	0		

Operator Precedence
(Boolean Operator Priority Order)

1. Parentheses
2. NOT
3. AND
4. OR

Pi**NA** c**O**lada?

Boolean Functions

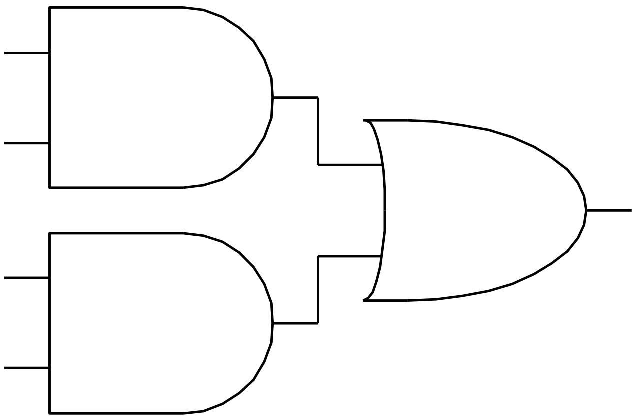
- Consists of
  - binary variables (in either normal or complement form)
  - the constants: 0 and 1
  - logic operation symbols: “+” and “·”
- Example:
  - $F_1(x, y, z) = x + y' z$
  - $F_2(x, y, z) = x' y' z + x' y z + xy'$

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

ELSEVIER

Rules of Combinational Composition

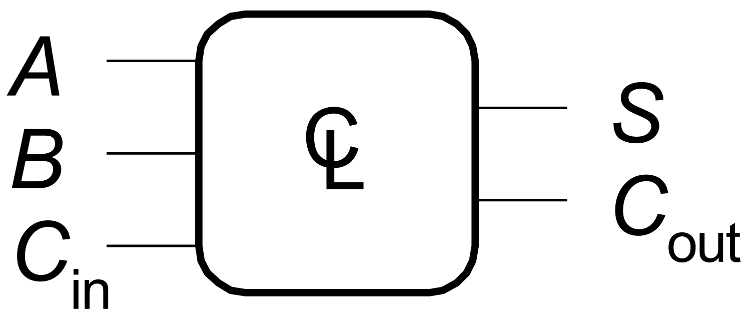
- Every element is combinational
- Every node is either an input or connects to *exactly one* output
- The circuit contains no cyclic paths
- **Example:**



Boolean Equations

- Functional specification of outputs in terms of inputs
- **Example:** $S = F(A, B, C_{in})$

  $C_{out} = F(A, B, C_{in})$



$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Some Definitions

- **Complement**: variable with a bar or ' over it

  $\bar{A}, \bar{B}, \bar{C}$

- **Literal**: variable or its complement

  $A, \bar{A}, B, \bar{B}, C, \bar{C}$

- **Implicant**: product of literals

  $AB\bar{C}, \bar{A}C, BC$

- **Minterm**: product that includes all input variables

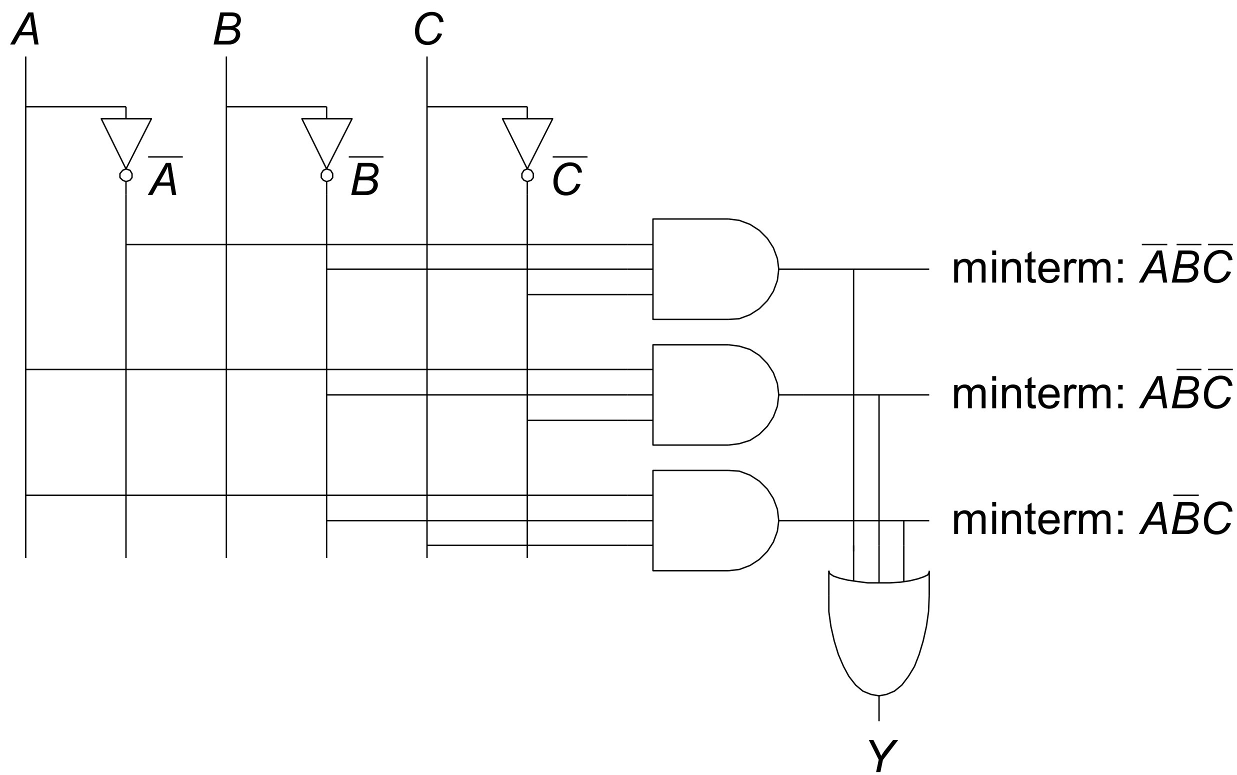
  $AB\bar{C}, A\bar{B}\bar{C}, ABC$

- **Maxterm**: sum that includes all input variables

  $(A+\bar{B}+C), (\bar{A}+B+\bar{C}), (\bar{A}+\bar{B}+C)$

From Logic to Gates

- Two-level logic: ANDs followed by ORs
- Example: $Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$



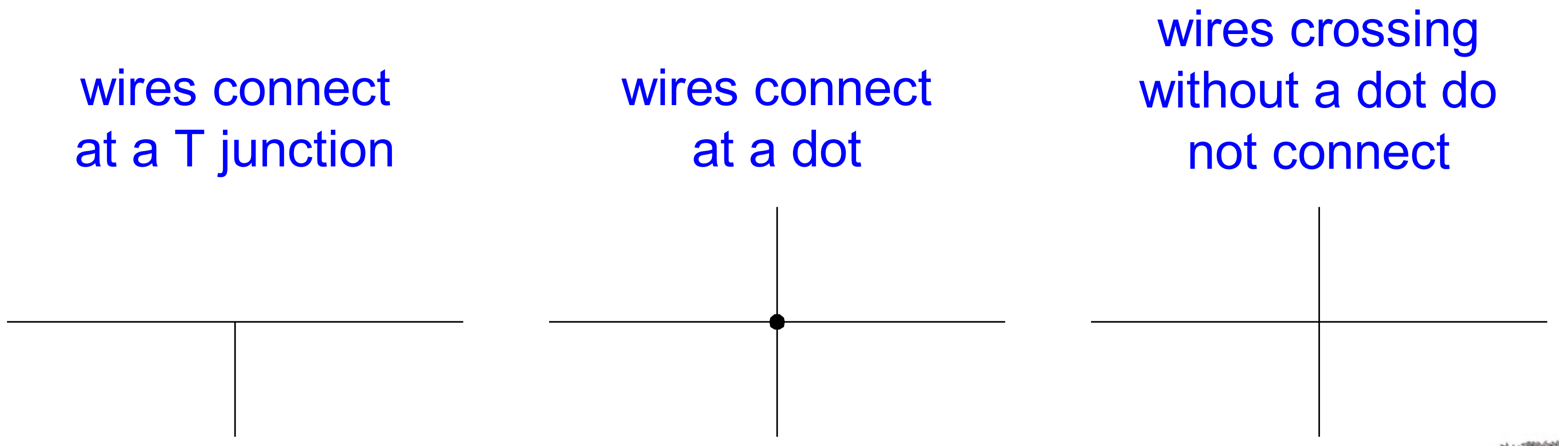
Circuit Schematics Rules

- Inputs on the left (or top)
- Outputs on right (or bottom)
- Gates flow from left to right
- Straight wires are best

COMBINATIONAL LOGIC DESIGN

Circuit Schematic Rules (cont.)

- Wires always connect at a T junction
- A dot where wires cross indicates a connection between the wires
- Wires crossing *without* a dot make no connection

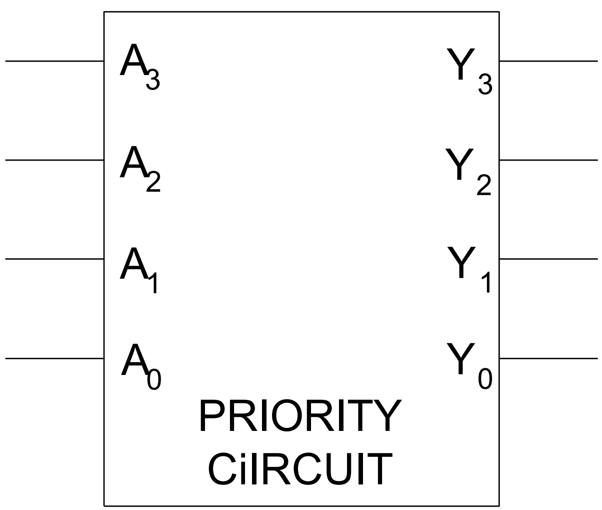


ELSEVIER

Multiple-Output Circuits

- **Example: Priority Circuit**

  Output asserted corresponding to most significant TRUE input

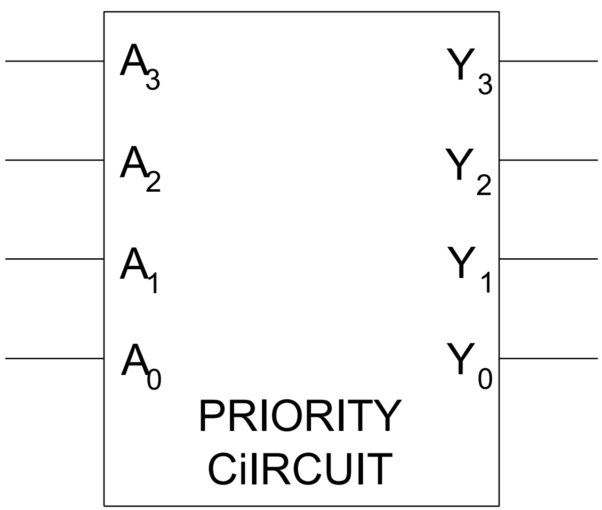


A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Multiple-Output Circuits

- **Example: Priority Circuit**

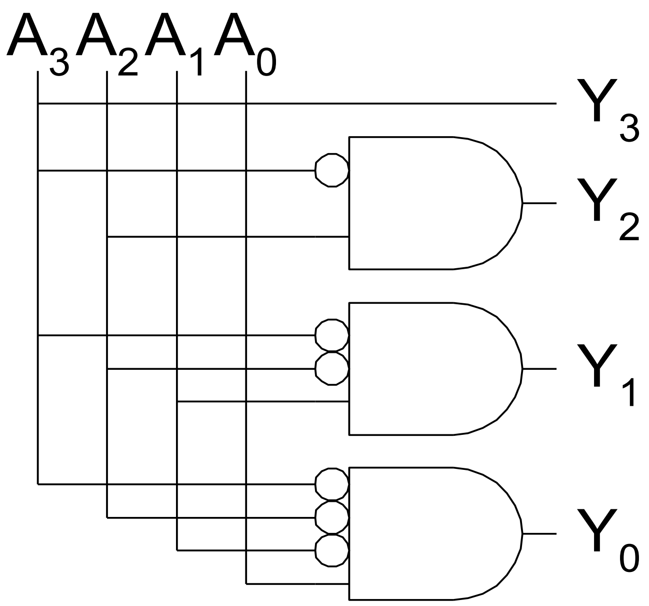
  Output asserted corresponding to most significant TRUE input



A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

Priority Circuit Hardware

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0



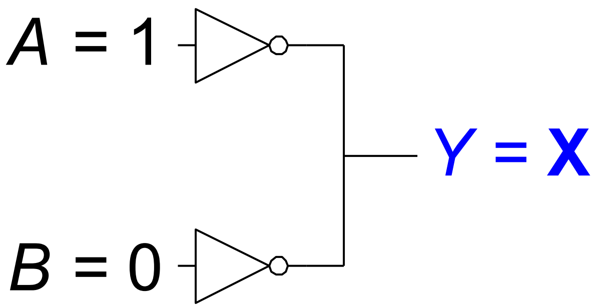
Don't Cares

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

Contention: X

- Contention: circuit tries to drive output to 1 **and** 0
  - Actual value somewhere in between
  - Could be 0, 1, or in forbidden zone
  - Might change with voltage, temperature, time, noise
  - Often causes excessive power dissipation

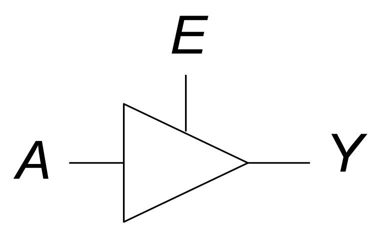


- **Warnings:**
  - Contention usually indicates a **bug**.
  - **X is used for both "don't care" and contention** -> look at the context to tell them apart

Floating: Z

- Floating, high impedance, open, high Z
- Floating output might be 0, 1, or somewhere in between
  - A voltmeter won't indicate whether a node is floating

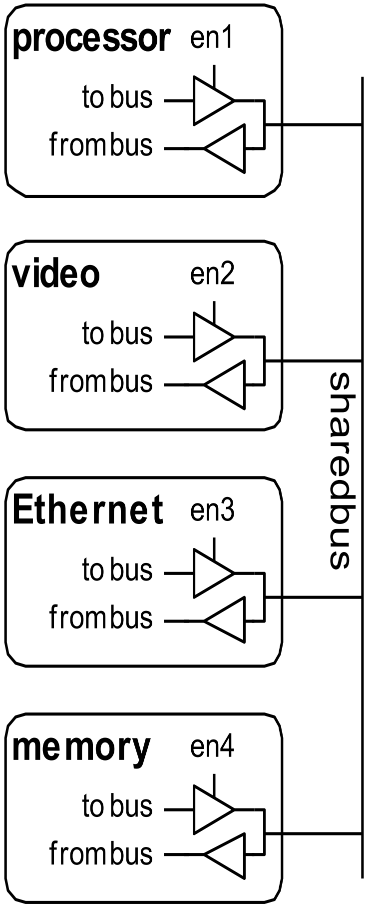
Tristate Buffer



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Tristate Busses

- Floating nodes are used in tristate busses
  - Many different drivers
  - Exactly one is active at once

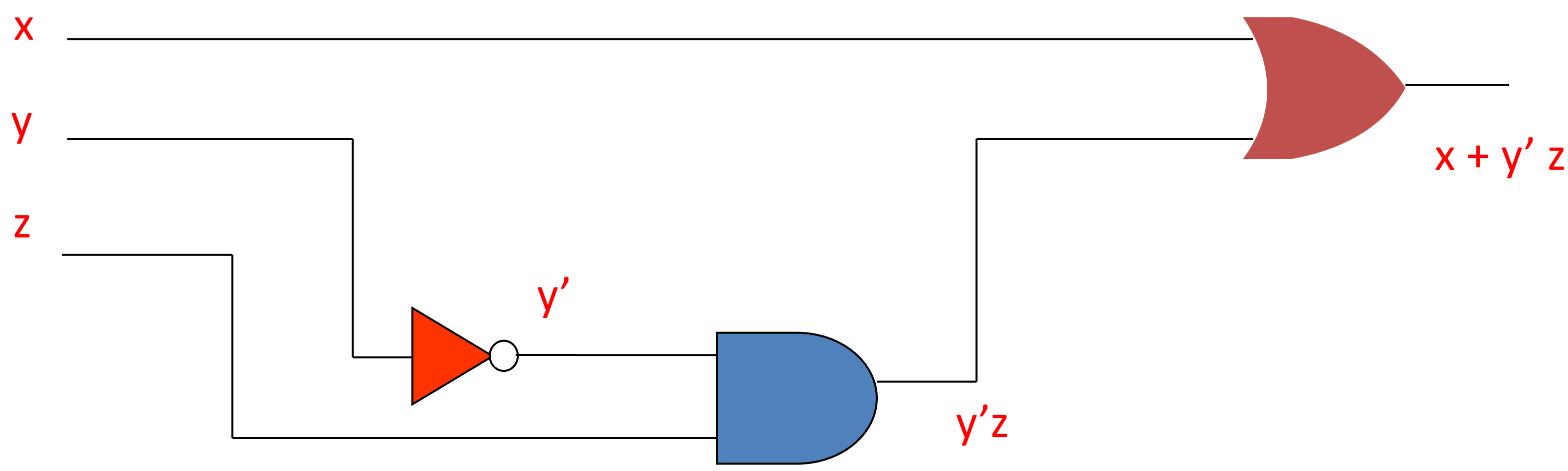


Logic Circuit Diagram of F_1

$F_1(x, y, z) = x + y' z$

Logic Circuit Diagram of F_1

$F_1(x, y, z) = x + y' z$



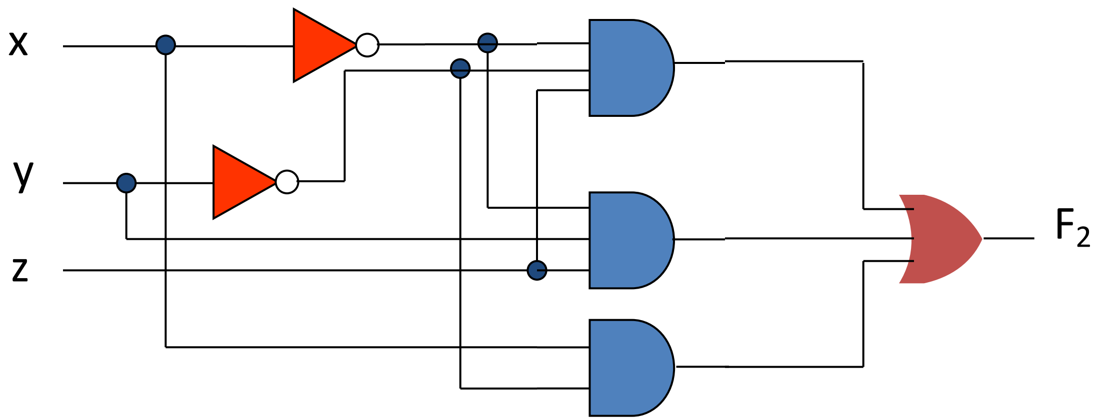
Gate Implementation of $F_1 = x + y' z$

Logic Circuit Diagram of F_2

$F_2 = x' \, y' \, z + x' \, y \, z + xy'$

Logic Circuit Diagram of F_2

$F_2 = x'\,y'\,z + x'\,y\,z + xy'$



Algebraic manipulation

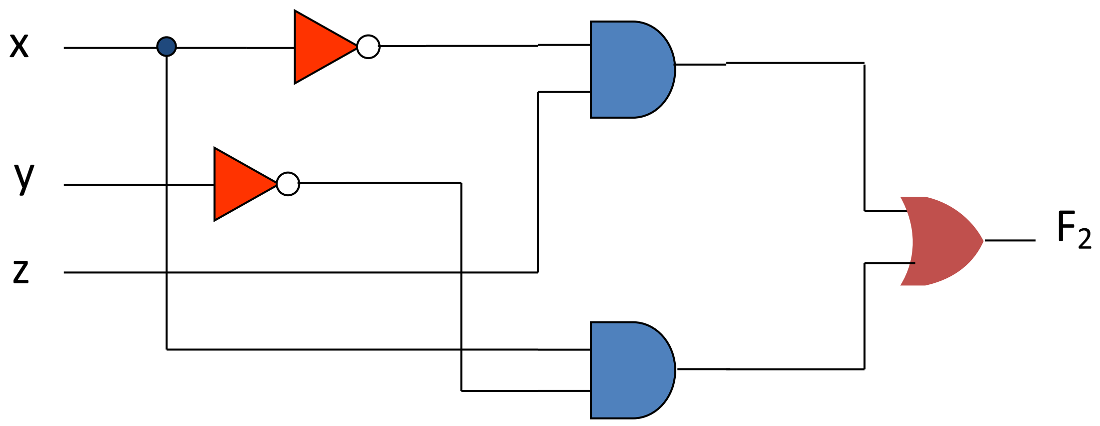
$$\begin{aligned} F_2 &= x'\,y'\,z + x'\,y\,z + xy' \\ &= x'z(y'+y) + xy' \\ &= x'z + xy' \end{aligned}$$

Alternative Implementations of F_2

$F_2 = x' z + xy'$

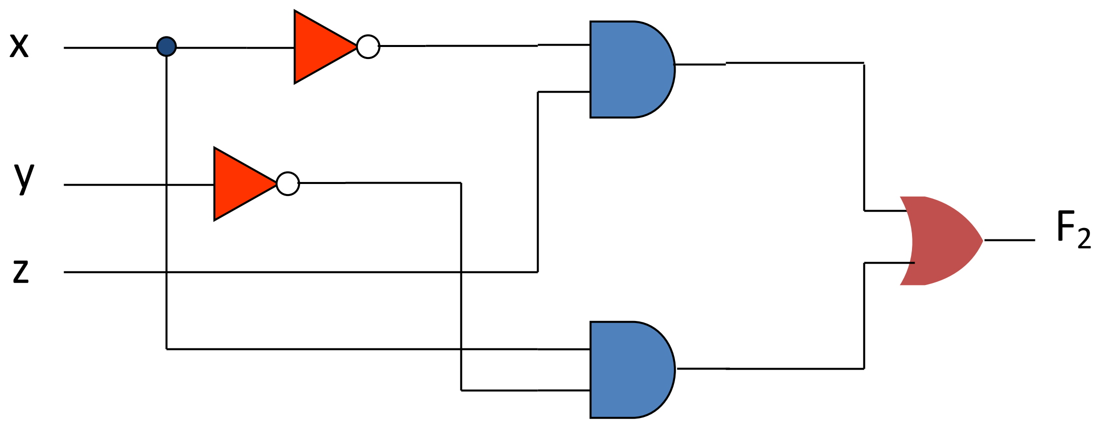
Alternative Implementations of F_2

$F_2 = x' z + xy'$

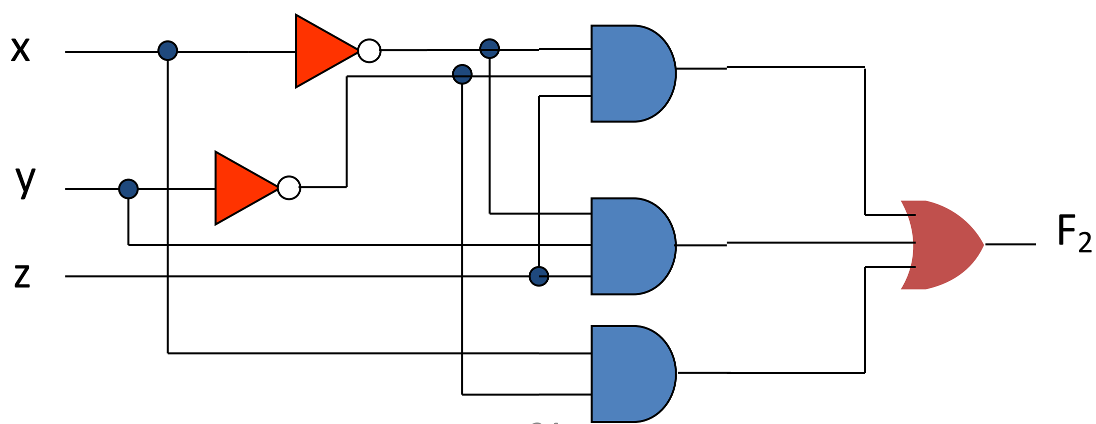


Alternative Implementations of F_2

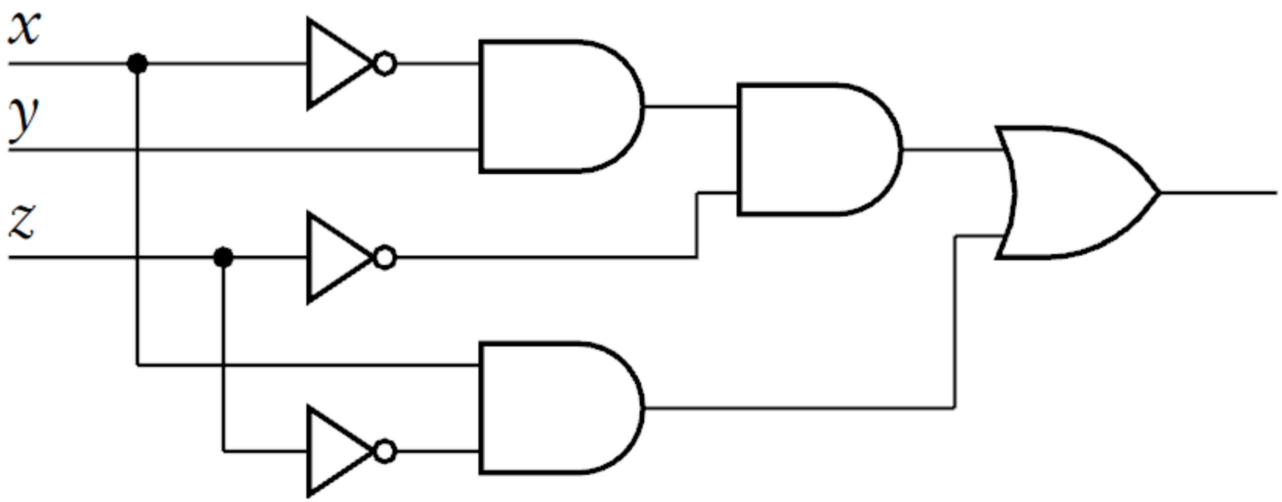
$F_2 = x' z + xy'$



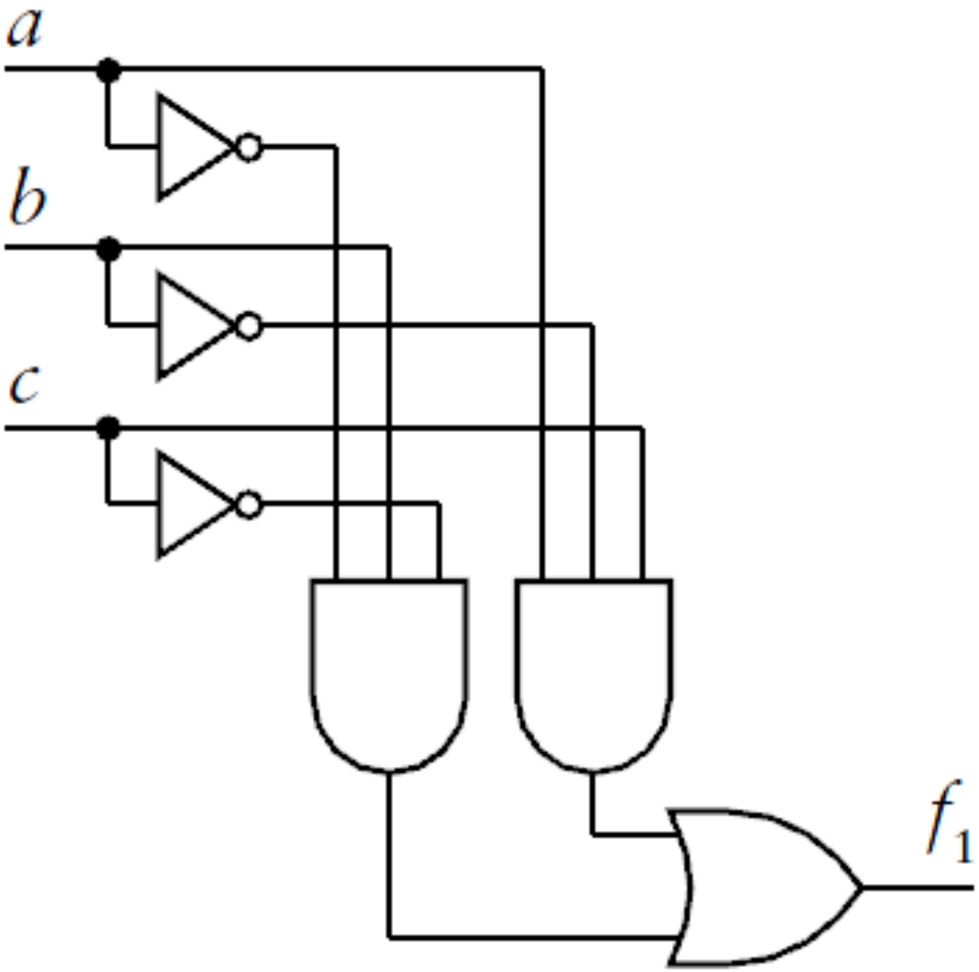
$F_2 = x' y' z + x' y z + xy'$



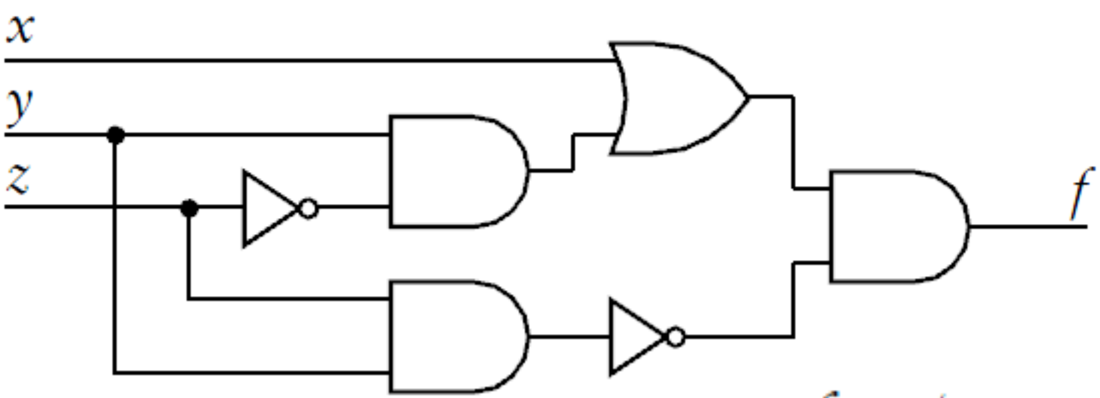
Example



Example

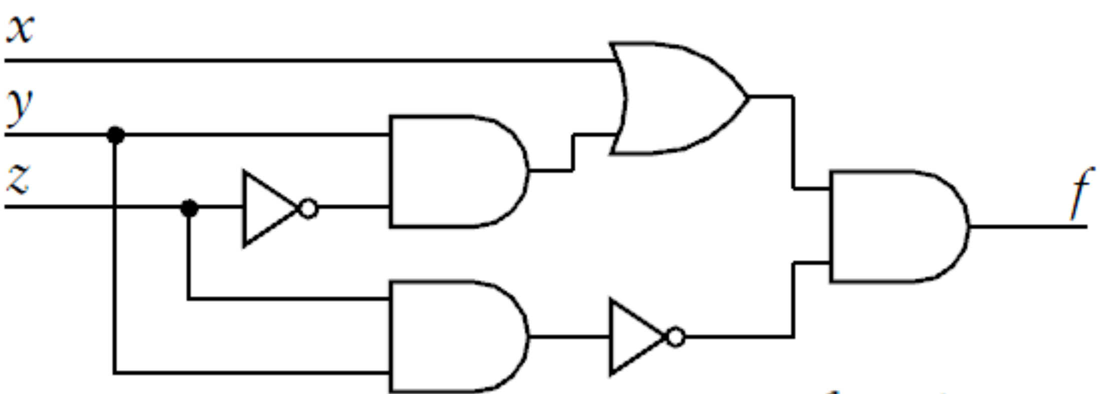


Example



$$f = (x + y \cdot \bar{z}) \cdot \overline{(y \cdot z)}$$

Example



$$
\begin{aligned}
f &= (x + y \cdot \overline{z}) \cdot \overline{(y \cdot z)} \\
&= (x + y \cdot \overline{z}) \cdot (\overline{y} + \overline{z}) \\
&= x \cdot (\overline{y} + \overline{z}) + (y \cdot \overline{z}) \cdot (\overline{y} + \overline{z}) \\
&= x \cdot \overline{y} + x \cdot \overline{z} + y \cdot \overline{z} \cdot \overline{y} + y \cdot \overline{z} \cdot \overline{z} \\
&= x \cdot \overline{y} + x \cdot \overline{z} + 0 \cdot \overline{z} + y \cdot \overline{z} \cdot \overline{z} \\
&= x \cdot \overline{y} + x \cdot \overline{z} + 0 + y \cdot \overline{z} \cdot \overline{z} \\
&= x \cdot \overline{y} + x \cdot \overline{z} + 0 + y \cdot \overline{z} \\
&= x \cdot \overline{y} + x \cdot \overline{z} + y \cdot \overline{z}
\end{aligned}
$$

OTHER LOGIC OPERATORS - 1

- AND, OR, NOT are logic operators
  - Boolean functions with two variables
  - These are three of the 16 possible two-variable Boolean functions

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

x	y	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

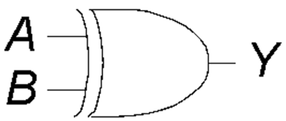
ELSEVIER

OTHER LOGIC OPERATORS - 2

- Some of the Boolean functions with two variables
  - Constant functions: $F_0 = 0$ and $F_{15} = 1$
  - AND function: $F_1 = xy$
  - OR function: $F_7 = x + y$
  - XOR function:
    - $F_6 = x'y + xy' = x \oplus y$ **: x or y, but not both**
  - XNOR (Equivalence) function:
    - $F_9 = xy + x'y' = (x \oplus y)'$ **: x equals y**
  - NOR function:
    - $F_8 = (x + y)' = (x \downarrow y)$ (Not-OR)
  - NAND function:
    - $F_{14} = (xy)' = (x \uparrow y)$ (Not-AND)

More Two-Input Logic Gates

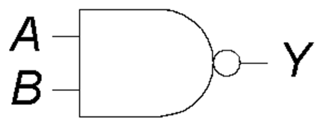
XOR



$$Y = A \oplus B$$

A	B	Y
0	0	
0	1	
1	0	
1	1	

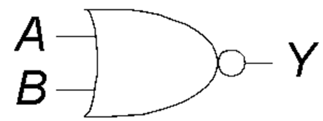
NAND



$$Y = \overline{AB}$$

A	B	Y
0	0	
0	1	
1	0	
1	1	

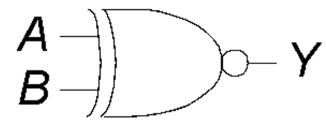
NOR



$$Y = \overline{A + B}$$

A	B	Y
0	0	
0	1	
1	0	
1	1	

XNOR

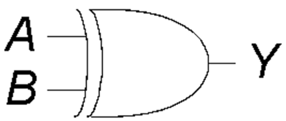


$$Y = \overline{A \oplus B}$$

A	B	Y
0	0	
0	1	
1	0	
1	1	

More Two-Input Logic Gates

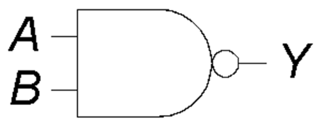
XOR



$$Y = A \oplus B$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

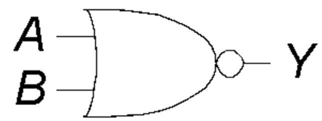
NAND



$$Y = \overline{AB}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

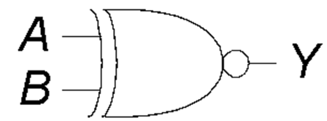
NOR



$$Y = \overline{A + B}$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR



$$Y = \overline{A \oplus B}$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Simplifying Boolean Equations

Example 1:

- $Y = AB + \overline{A}B$

Simplifying Boolean Equations

Example 1:

- $Y = AB + \overline{A}B$

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

$$= B(1)$$

$$= B$$

Simplifying Boolean Equations

Example 2:

- $Y = A(AB + ABC)$

Simplifying Boolean Equations

Example 2:

- $Y = A(AB + ABC)$

$$= A(AB(1 + C))$$

$$= A(AB(1))$$

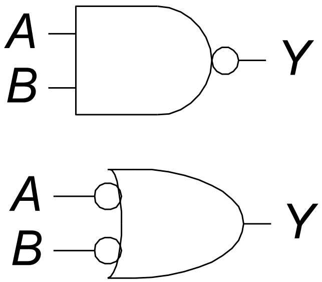
$$= A(AB)$$

$$= (AA)B$$

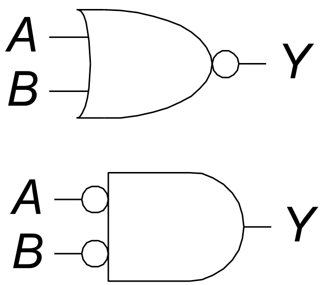
$$= AB$$

DeMorgan's Theorem

- $Y = \overline{AB} = \overline{A} + \overline{B}$

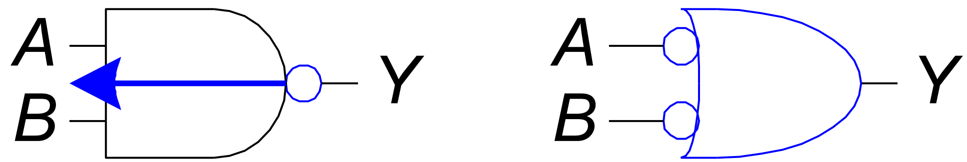


- $Y = \overline{A + B} = \overline{A} \cdot \overline{B}$



Bubble Pushing

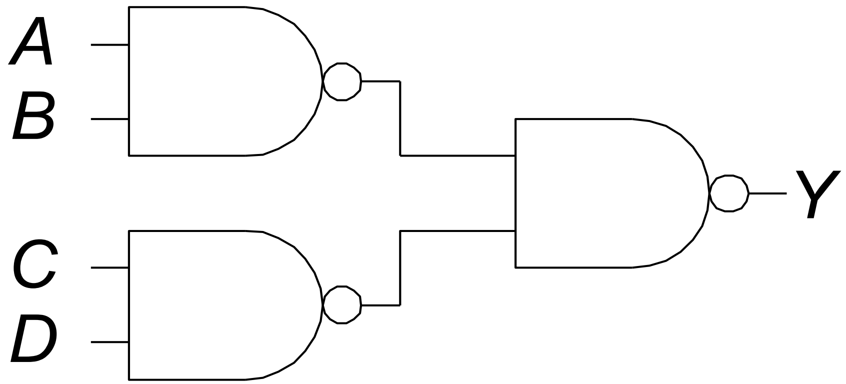
- **Backward:**
  - Body changes
  - Adds bubbles to inputs



- **Forward:**
  - Body changes
  - Adds bubble to output

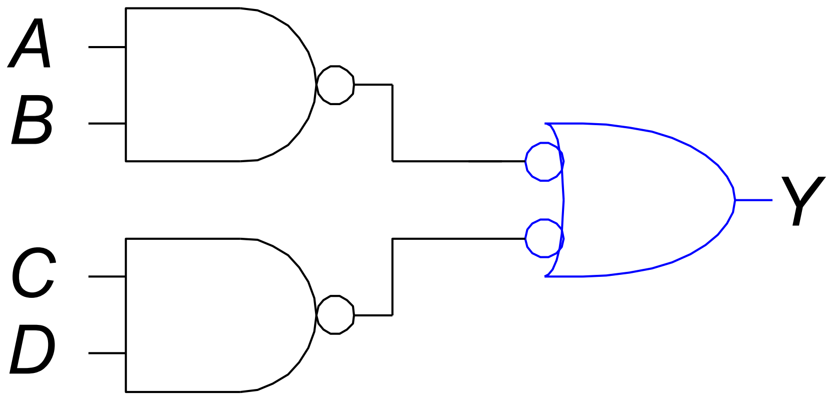
Bubble Pushing

- What is the Boolean expression for this circuit?



Bubble Pushing

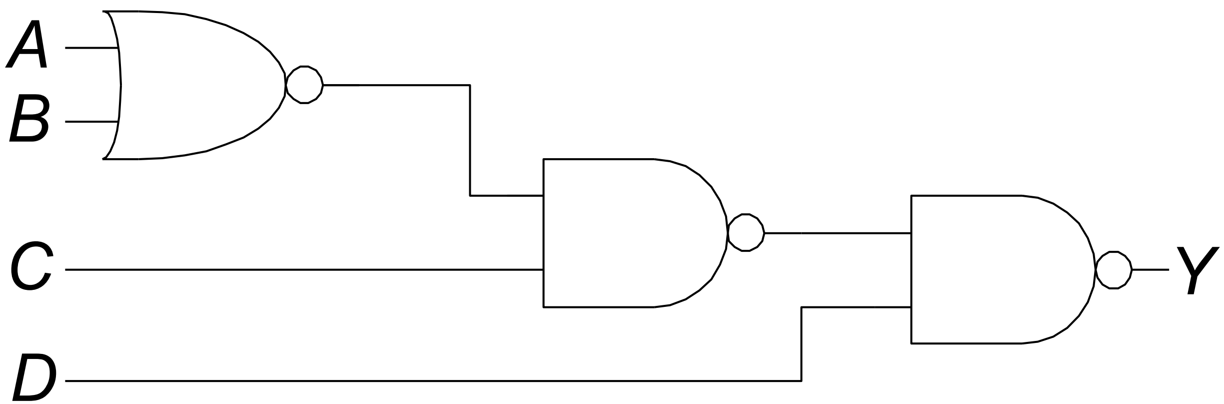
- What is the Boolean expression for this circuit?



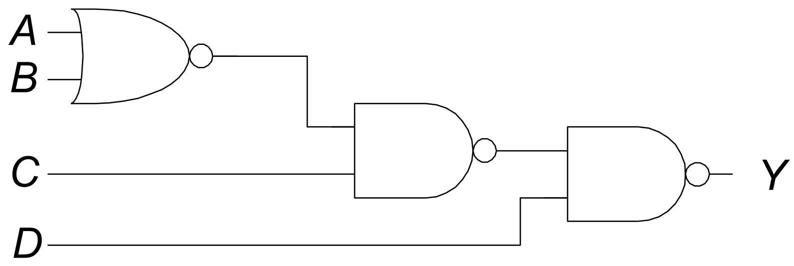
$$Y = AB + CD$$

Bubble Pushing Rules

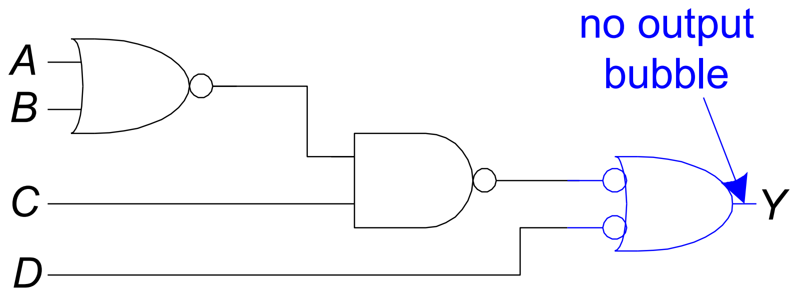
- Begin at output, then work toward inputs
- Push bubbles on final output back
- Draw gates in a form so bubbles cancel



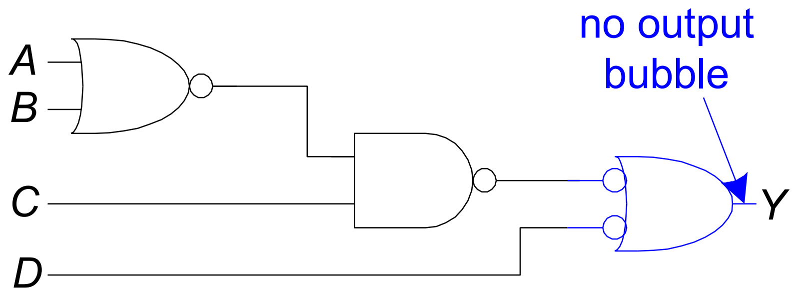
Bubble Pushing Example

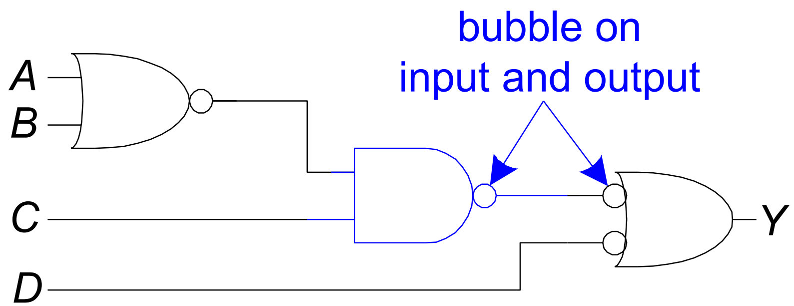


Bubble Pushing Example

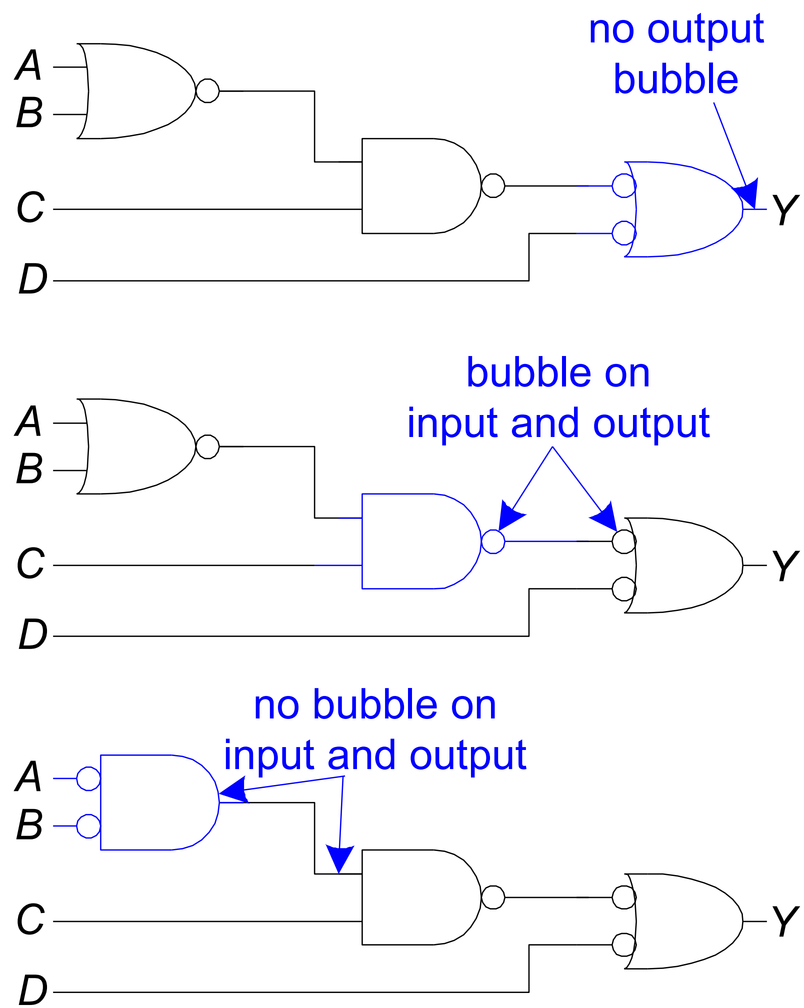


Bubble Pushing Example





Bubble Pushing Example



$$Y = \overline{A}\,\overline{B}C + \overline{D}$$

Complement of a Function

- F’ is complement of F
  - We can obtain F’ simply by interchanging of 0s and 1s in the truth table

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

F =

F’ =

Generalizing DeMorgan's Theorem

- We can also utilize DeMorgan's Theorem
  - $(x + y)' = x'\,y'$
  - $(A + B + C)'$

    $= (A+B)'C'$

    $= A'B'C'$
- **We can generalize DeMorgan's Theorem**
  - $(x_1 + x_2 + \ldots + x_N)' = x_1' \cdot x_2' \cdot \ldots \cdot x_N'$
  - $(x_1 \cdot x_2 \cdot \ldots \cdot x_N)' = x_1' + x_2' + \ldots + x_N'$

Example: Complement of a Function

- Example:
  - $F_1 = x'yz' + x'y'z$
  - $F_1' = (x'yz' + x'y'z)'$
    $= (x'yz')'(x'y'z)'$
    $= (x + y' + z)(x + y + z')$

  - $F_2 = x(y'z' + yz)$
  - $F_2' = (x(y'z' + yz))'$
    $= x'+(y'z' + yz)'$
    $= x' + (y + z)(y' + z')$

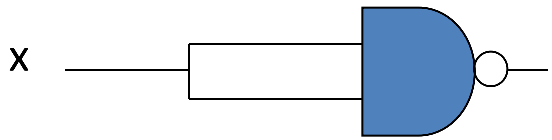
- Easy Way to Complement: use DeMorgan's

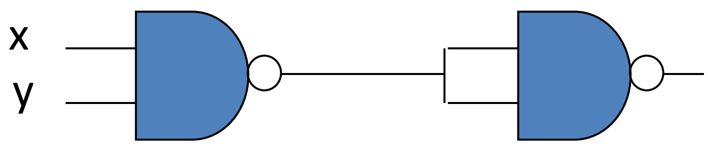
Universal Gates

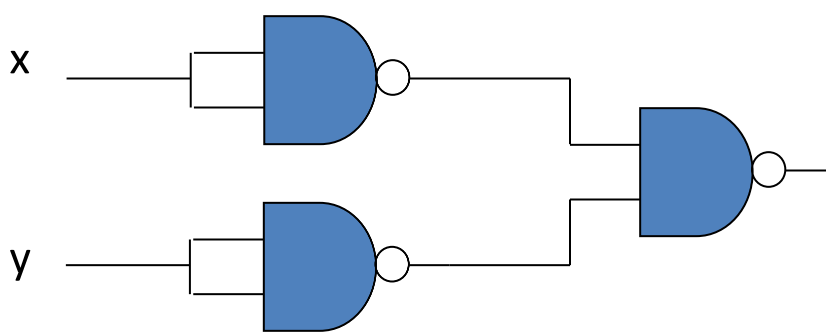
- **NAND and NOR gates are <u>universal</u>**
- We know <u>any</u> Boolean function can be written in terms of three logic operations:
  - AND, OR, NOT
- In return, **NAND** gate can implement these three logic gates by itself
  - **So can NOR gate**

x	y	(xy)’	x’	y ’	(x’ y’)’
0	0	1	1	1	
0	1	1	1	0	
1	0	1	0	1	
1	1	0	0	0	

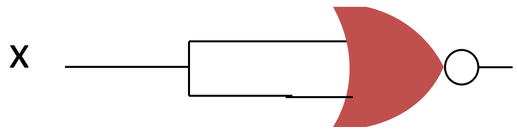
NAND Gate

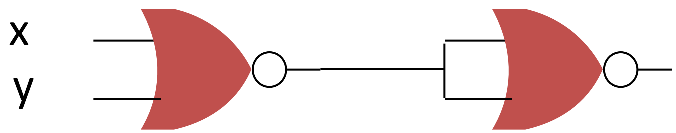


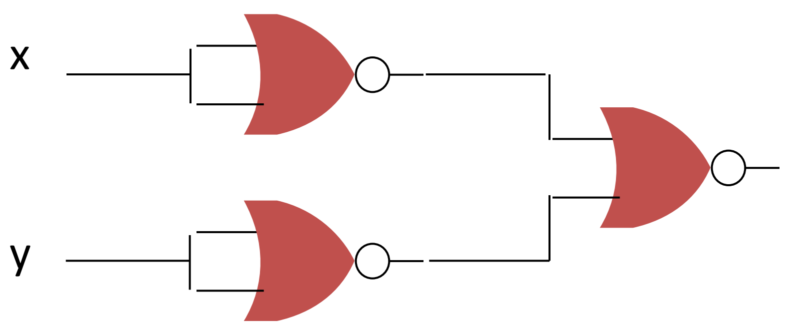




NOR Gate



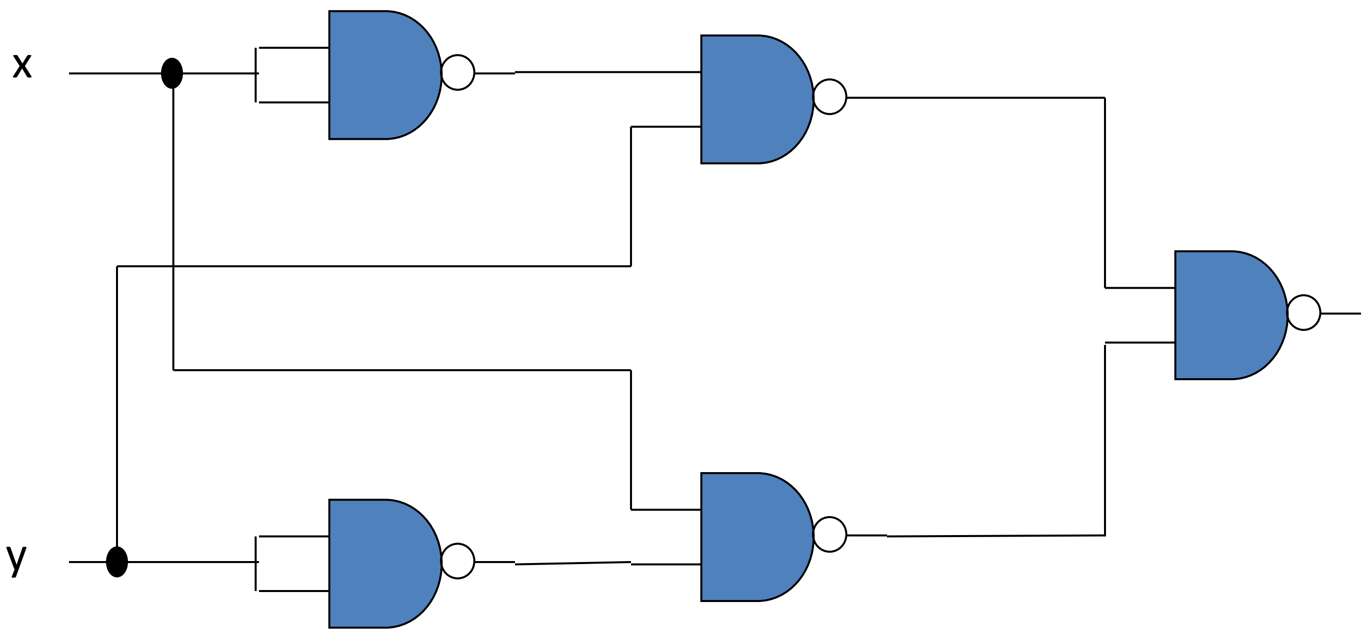




Designs with NAND gates Example 1/2

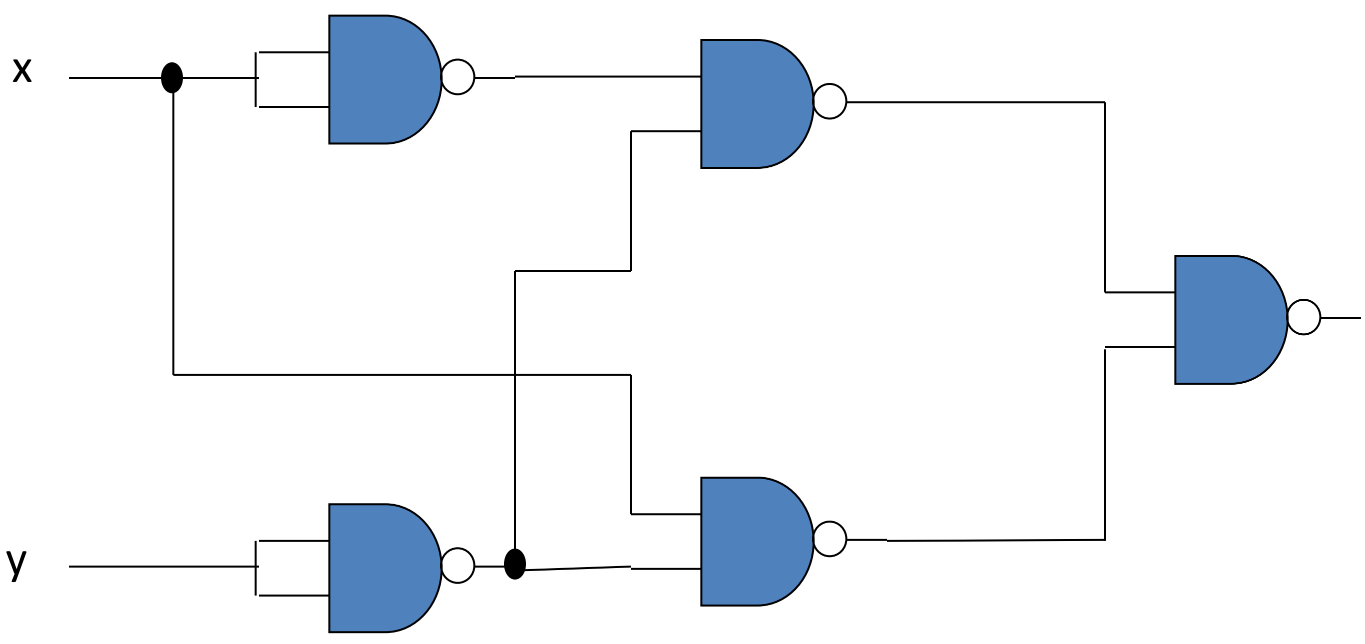
- A function:

  $F_1 = x'\,y + xy'$



Example 2/2

$F_2 = x' \, y' + xy'$



Multiple Input Gates

- AND and OR operations:
  - They are both commutative and associative
  - No problem with extending the number of inputs
- NAND and NOR operations:
  - They are commutative but <u>not associative</u>
  - Extending the number of inputs is not obvious
- <u>Example</u>: NAND gates
  - $((xy)'z)' \neq (x(yz)')'$

  - $((xy)'z)' = xy + z'$
  - $(x(yz)')' = x' + yz$

ELSEVIER

Nonassociativity of NOR operation

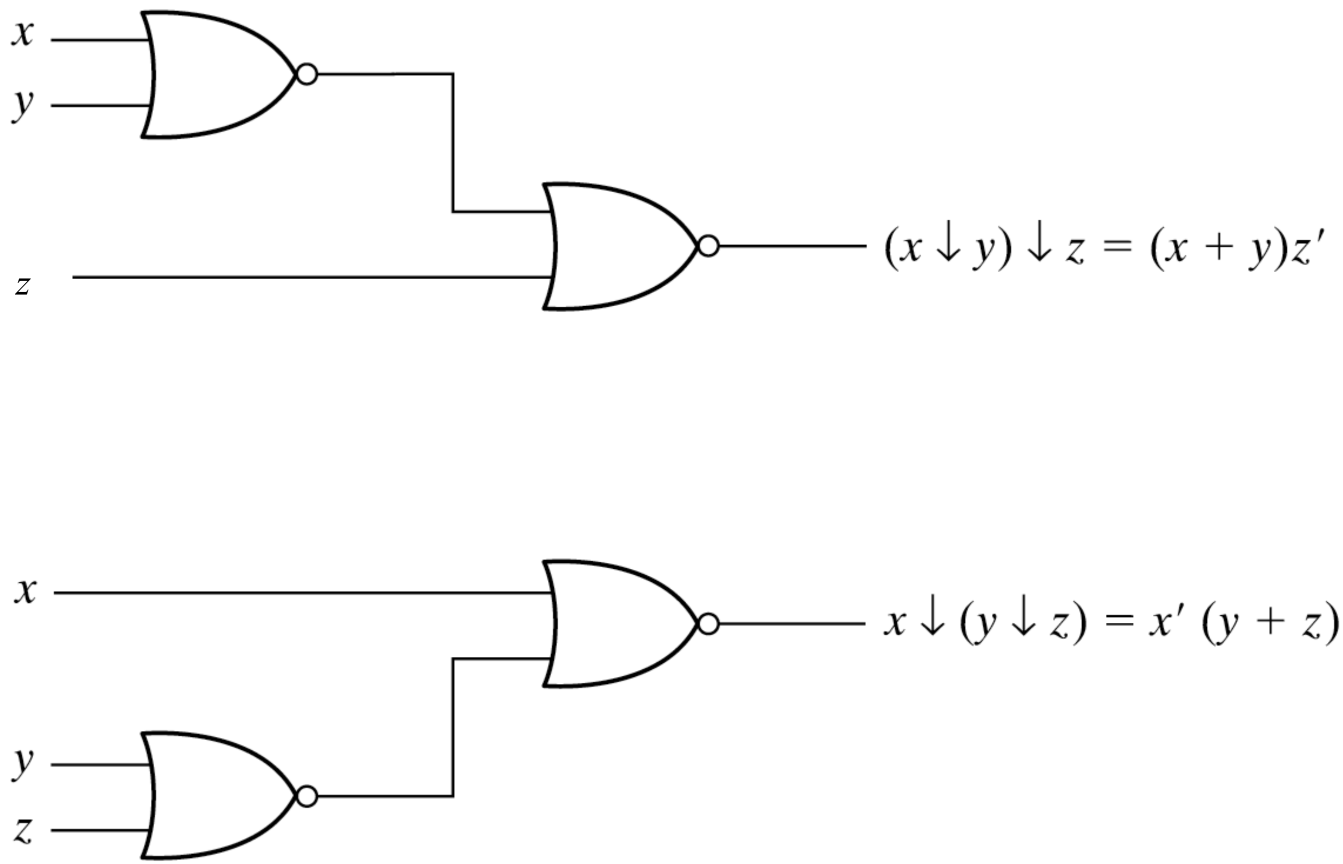


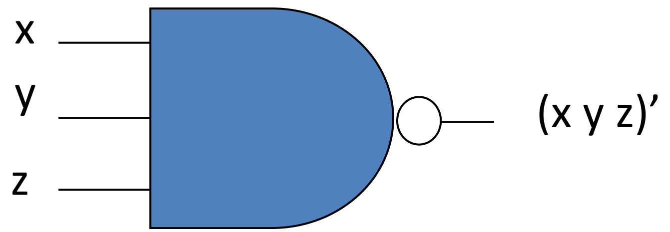
Fig. 2-6 Demonstrating the nonassociativity of the NOR operator; $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

ELSEVIER

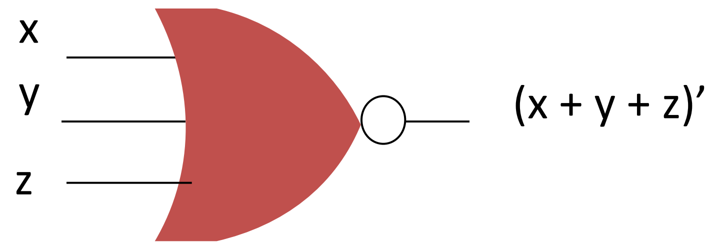
Multiple Input Universal Gates

- To overcome this difficulty, we define multiple-input NAND and NOR gates in slightly different manner

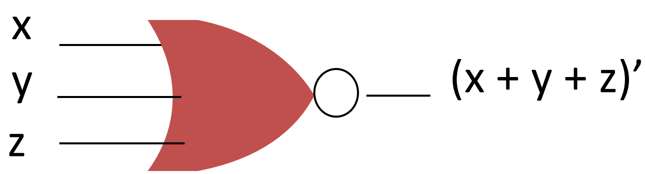
Three input NAND gate: (x y z)’



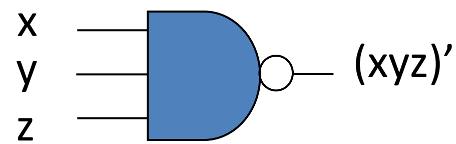
Three input NOR gate:(x + y + z)’



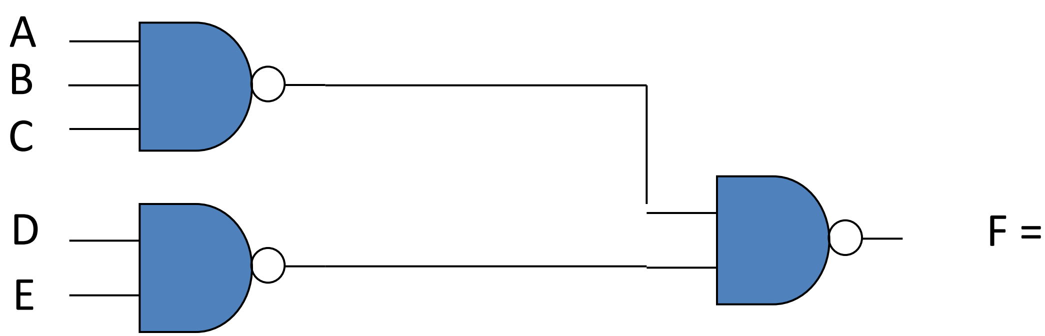
Multiple Input Universal Gates



3-input NOR gate



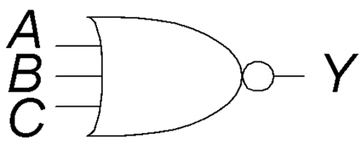
3-input NAND gate



Cascaded NAND gates

Multiple-Input Logic Gates

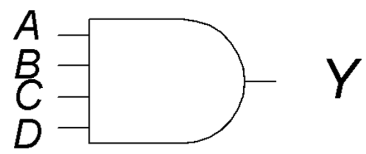
NOR3



$$Y = \overline{A+B+C}$$

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

AND4



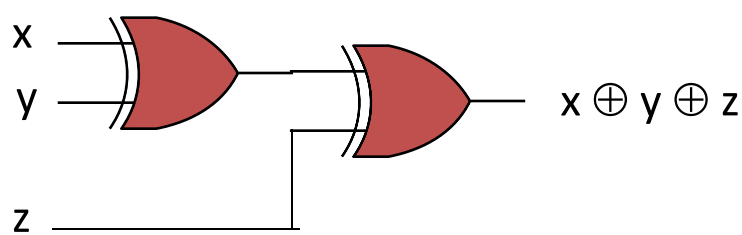
$$Y = ABCD$$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

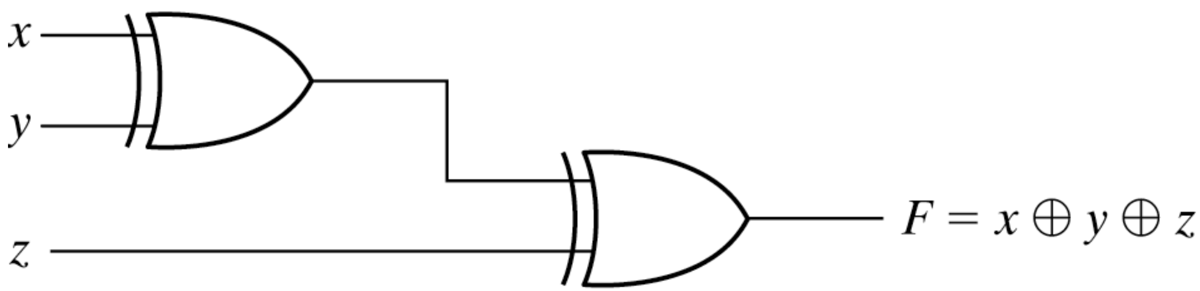
- Multi-input XOR: Odd parity

XOR and XNOR Gates

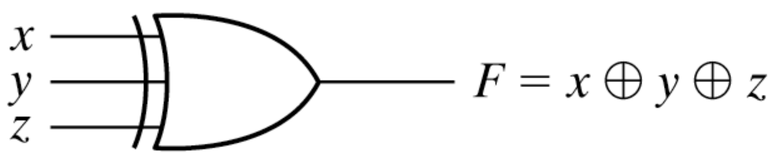
- XOR and XNOR operations are both commutative and associative.
- No problem manufacturing multiple input XOR and XNOR gates
- However, they are more **costly from hardware point of view.**
- Therefore, we usually have 2-input XOR and XNOR gates



3-input XOR Gates



(a) Using 2-input gates



(b) 3-input gate

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

(c) Truth table

Fig. 2-8 3-input exclusive-OR gate

Sum-of-Products (SOP) Form

- All equations can be written in SOP form
- Each row has a **minterm**
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- Form function by ORing minterms for which the output is TRUE
- >> Thus, a sum (OR) of products (AND terms)

A	B	Y	minterm	minterm name
0	0	0	$\overline{A}\ \overline{B}$	m_0
0	1	1	$\overline{A}\ B$	m_1
1	0	0	$A\ \overline{B}$	m_2
1	1	1	$A\ B$	m_3

$$Y = \mathbf{F}(A, B) =$$

Sum-of-Products (SOP) Form

- All equations can be written in SOP form
- Each row has a **minterm**
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- Form function by ORing minterms for which the output is TRUE
- >> Thus, a sum (OR) of products (AND terms)

A	B	Y	minterm	minterm name
0	0	0	$\overline{A}\ \overline{B}$	m_0
0	1	1	$\overline{A}\ B$	m_1
1	0	0	$A\ \overline{B}$	m_2
1	1	1	$A\ B$	m_3

$Y = \mathrm{F}(A, B) =$

Sum-of-Products (SOP) Form

- All equations can be written in SOP form
- Each row has a **minterm**
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- Form function by ORing minterms for which the output is TRUE
- >> Thus, a sum (OR) of products (AND terms)

A	**B**	**Y**	**minterm**	**minterm name**
0	0	0	$\overline{A}\ \overline{B}$	m_0
0	1	1	$\overline{A}\ B$	m_1
1	0	0	$A\ \overline{B}$	m_2
1	1	1	$A\ B$	m_3

$$Y = F(A, B) = \overline{A}B + AB = \Sigma(1, 3)$$

Product-of-Sums (POS) Form

- All Boolean equations can be written in POS form
- Each row has a **maxterm**
- A maxterm is a sum (OR) of literals
- Each maxterm is FALSE for that row (and only that row)
- Form function by ANDing the maxterms for which the output is FALSE
- Thus, a product (AND) of sums (OR terms)

A	*B*	*Y*	maxterm	maxterm name
0	0	0	$A + B$	M_0
0	1	1	$A + \overline{B}$	M_1
1	0	0	$\overline{A} + B$	M_2
1	1	1	$\overline{A} + \overline{B}$	M_3

$$Y = \mathbf{F}(A, B) = (A + B)(\overline{A} + B) = \Pi(0, 2)$$

Boolean Equations Example

- You are going to the cafeteria for lunch
  - You won't eat lunch ($\overline{E}$)
  - If it's not open ($\overline{O}$) or
  - If they only serve corndogs (C)
- Write a truth table for determining if you will eat lunch (E).

O	*C*	*E*
0	0	
0	1	
1	0	
1	1	

Boolean Equations Example

- You are going to the cafeteria for lunch
  - You won't eat lunch ($\overline{E}$)
  - If it's not open ($\overline{O}$) or
  - If they only serve corndogs (C)
- Write a truth table for determining if you will eat lunch (E).

O	C	E
0	0	0
0	1	0
1	0	1
1	1	0

SOP & POS Form

- SOP – sum-of-products

O	*C*	*E*	minterm
0	0		$\overline{O}\ \overline{C}$
0	1		$\overline{O}\ C$
1	0		$O\ \overline{C}$
1	1		$O\ C$

- POS – product-of-sums

O	*C*	*E*	maxterm
0	0		$O + C$
0	1		$O + \overline{C}$
1	0		$\overline{O} + C$
1	1		$\overline{O} + \overline{C}$

SOP & POS Form

- SOP – sum-of-products

O	C	E	minterm
0	0	0	$\overline{O}\,\overline{C}$
0	1	0	$\overline{O}\,C$
1	0	1	$O\,\overline{C}$
1	1	0	$O\,C$

$$Y = O\overline{C}$$
$$= \Sigma(2)$$

- POS – product-of-sums

O	C	E	maxterm
0	0	0	$O + C$
0	1	0	$O + \overline{C}$
1	0	1	$\overline{O} + C$
1	1	0	$\overline{O} + \overline{C}$

$$Y = (O + C)(O + \overline{C})(\overline{O} + \overline{C})$$
$$= \Pi(0, 1, 3)$$

Min- & Maxterms with n = 3

			Minterms		Maxterms	
x	y	z	term	designation	term	designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Formal Expression with Minterms

xyz	m_i	M_i	F
000	$m_0=x'y'z'$	$M_0=x+y+z$	F(0,0,0)
001	$m_1=x'y'z$	$M_1=x+y+z'$	F(0,0,1)
010	$m_2=x'yz'$	$M_2=x+y'+z$	F(0,1,0)
011	$m_3=x'yz$	$M_3=x+y'+z'$	F(0,1,1)
100	$m_4=xy'z'$	$M_4=x'+y+z$	F(1,0,0)
101	$m_5=xy'z$	$M_5=x'+y+z'$	F(1,0,1)
110	$m_6=xyz'$	$M_6=x'+y'+z$	F(1,1,0)
111	$m_7=xyz$	$M_7=x'+y'+z'$	F(1,1,1)

$$F(x, y, z) = F(0,0,0)m_0 + F(0,0,1)m_1 + F(0,1,0)m_2 + F(0,1,1)m_3 + F(1,0,0)m_4 + F(1,0,1)m_5 + F(1,1,0)m_6 + F(1,1,1)m_7$$

Formal Expression with Maxterms

xyz	m_i	M_i	F
000	$m_0=x'y'z'$	$M_0=x+y+z$	F(0,0,0)
001	$m_1=x'y'z$	$M_1=x+y+z'$	F(0,0,1)
010	$m_2=x'yz'$	$M_2=x+y'+z$	F(0,1,0)
011	$m_3=x'yz$	$M_3=x+y'+z'$	F(0,1,1)
100	$m_4=xy'z'$	$M_4=x'+y+z$	F(1,0,0)
101	$m_5=xy'z$	$M_5=x'+y+z'$	F(1,0,1)
110	$m_6=xyz'$	$M_6=x'+y'+z$	F(1,1,0)
111	$m_7=xyz$	$M_7=x'+y'+z'$	F(1,1,1)

$$F(x, y, z) = (F(0,0,0)+M_0)\,(F(0,0,1)+M_1)\,(F(0,1,0)+M_2)\,(F(0,1,0)+M_3)\,(F(1,0,0)+M_4)\,(F(1,0,1)+M_5)\,(F(1,1,0)+M_6)\,(F(1,1,1)+M_7)$$

Example

xyz	m_i	M_i	F
000	$m_0=x'y'z'$	$M_0=x+y+z$	0
001	$m_1=x'y'z$	$M_1=x+y+z'$	1
010	$m_2=x'yz'$	$M_2=x+y'+z$	1
011	$m_3=x'yz$	$M_3=x+y'+z'$	0
100	$m_4=xy'z'$	$M_4=x'+y+z$	0
101	$m_5=xy'z$	$M_5=x'+y+z'$	0
110	$m_6=xyz'$	$M_6=x'+y'+z$	1
111	$m_7=xyz$	$M_7=x'+y'+z'$	0

F(x, y, z) = ? in minters
F(x, y, z) = ? in maxterms

Example

xyz	m_i	M_i	F
000	$m_0=x'y'z'$	$M_0=x+y+z$	0
001	$m_1=x'y'z$	$M_1=x+y+z'$	1
010	$m_2=x'yz'$	$M_2=x+y'+z$	1
011	$m_3=x'yz$	$M_3=x+y'+z'$	0
100	$m_4=xy'z'$	$M_4=x'+y+z$	0
101	$m_5=xy'z$	$M_5=x'+y+z'$	0
110	$m_6=xyz'$	$M_6=x'+y'+z$	1
111	$m_7=xyz$	$M_7=x'+y'+z'$	0

$F(x, y, z) = x'y'z + x'yz' + xyz'$

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

Boolean Functions in Canonical Form

x	y	z	F_1	F_2
0	0	0	0	1
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- $F_1(x, y, z) =$
- $F_2(x, y, z) =$

Important Properties

- Any Boolean function can be expressed
  - as **a sum of minterms**
  - as **a product of maxterms**
- Example:
  - $F'(x,y,z) = \Sigma\ (0, 2, 3, 5, 6)$

    $=$
  - How do we find the complement of F'?
  - $F(x,y,z) =$

Important Properties

- Any Boolean function can be expressed
  - as **a sum of minterms**
  - as **a product of maxterms**
- Example:
  - $F'(x,y,z) = \Sigma\,(0, 2, 3, 5, 6)$

    $= x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - How do we find the complement of F'?
  - $F(x,y,z) =$

Important Properties

- Any Boolean function can be expressed
  - as **a sum of minterms**
  - as **a product of maxterms**
- Example:
  - $F'(x,y,z) = \Sigma\ (0, 2, 3, 5, 6)$

    $= x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - How do we find the complement of F'?
  - $F(x,y,z)=(x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' +z)$

    $=$

    $=$

Canonical Form

- If a Boolean function is expressed as a *sum of minterms* or *product of maxterms,* the function is said to be in **canonical form**.
- Example: $F = x + y'z \rightarrow$ canonical form?
  - No
  - But we can put it in canonical form.
  - $F = x + y'z = \Sigma\,(7, 6, 5, 4, 1)$
- Alternative way:
  – Obtain the truth table first and then the canonical term.

Example: Product of Maxterms

- $F = xy + x'z$
  - Use the distributive law of + over ·
  - F
$$
\begin{aligned}
F &= xy + x'z \\
&= xy(z+z') + x'z(y+y') \\
&= xyz + xyz' + x'yz + x'y'z \\
&= \Sigma\,(7,6,3,1)
\end{aligned}
$$

Example: Product of Maxterms

- F = xy + x'z
  - Use the distributive law of + over ·
  - F $= xy + x'z$

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

    $= xyz + xyz' + x'yz + x'y'z$

    $= \Sigma\,(7,6,3,1)$

    $= \Pi\,(4, 5, 0, 2)$

Conversion Between Canonical Forms

- Fact:
  - The complement of a function (given in sum of minterms) can be expressed as a sum of minterms missing from the original function
- Example:
  - $F(x, y, z) = \Sigma(1, 4, 5, 6, 7)$
  - $F'(x, y, z) =$
  - Now take the complement of F' and make use of DeMorgan's theorem
  - $(F')' = \qquad =$
  - $F = M_0 \cdot M_2 \cdot M_3 = \Pi(0, 2, 3)$

General Rule for Conversion

- <u>Important relation</u>:
  - $m_j' = M_j$
  - $M_j' = m_j$
- <u>The rule</u>:
  – Interchange symbols Π and Σ, and
  – list those terms missing from the original form
- <u>Example</u>: F = xy + x'z

$F = \Sigma(1, 3, 6, 7) \rightarrow F = \Pi(?, ?, ?, ?)$

Standard Forms

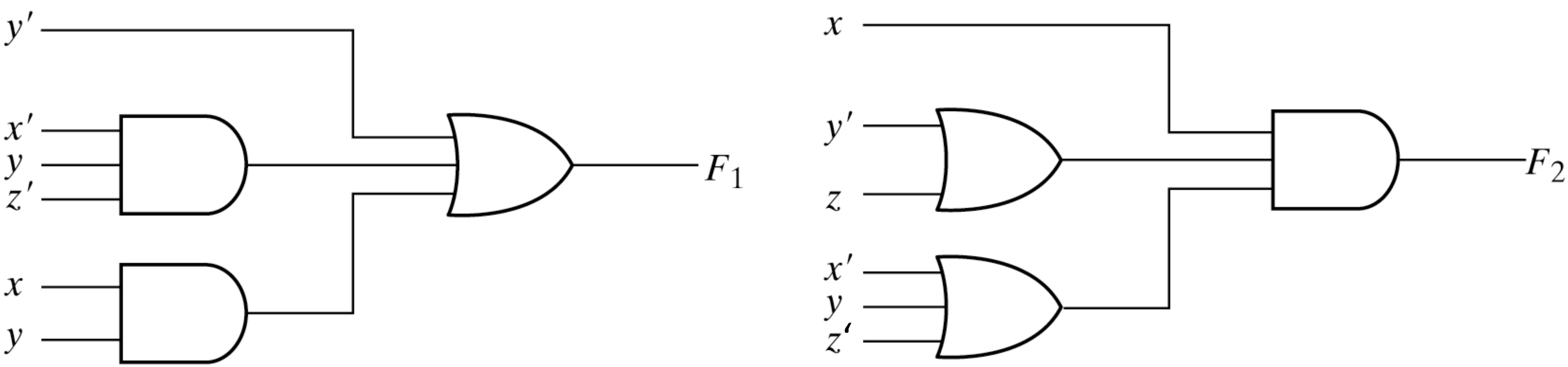
- Fact:
  - Canonical forms are very seldom the ones with the least number of literals
- Alternative representation:
  - Standard form
    - a <u>term</u> may contain any number of literals
  - Two types
    1. the sum of products
    2. the product of sums
  - Examples:
    - $F_1 = y' + xy + x'yz'$
    - $F_2 = x(y' + z)(x' + y + z')$

Example: Standard Forms

- $F_1 = y' + xy + x'yz'$
- $F_2 = x(y' + z)(x' + y + z')$

Example: Standard Forms

- $F_1 = y' + xy + x'yz'$
- $F_2 = x(y' + z)(x' + y + z')$



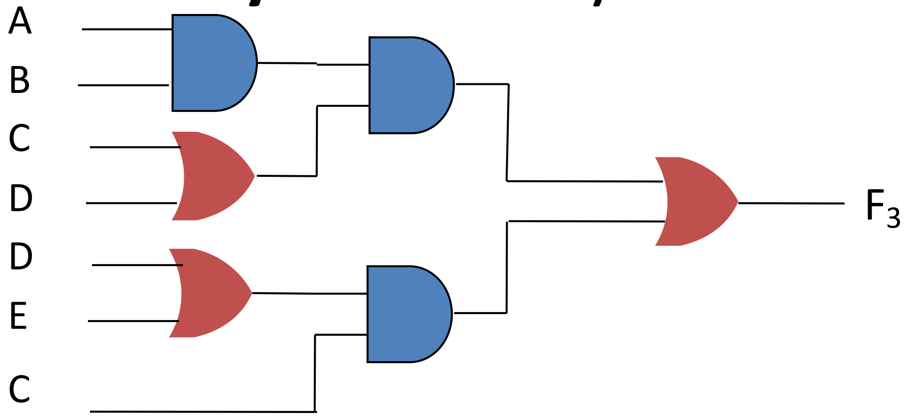
(a) Sum of Products

(b) Product of Sums

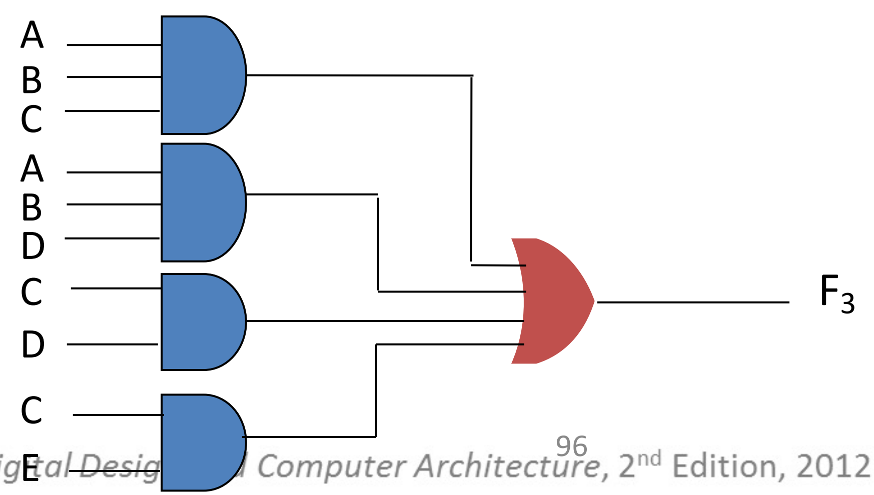
Fig. 2-3 Two-level implementation

Nonstandard Forms

- Example:
  - $F_3 = AB(C+D) + C(D + E)$
  - This **hybrid form** yields three-level implementation



– The standard form: $F_3 = ABC + ABD + CD + CE$



Complexity of Digital Circuits

- Directly related to the complexity of the algebraic expression we use to build the circuit.
- Truth table
  - may lead to different implementations
  - Question: which one to use?

Gate-Level Minimization

- Finding an optimal gate-level implementation of Boolean functions.
  - So far: ad hoc.
    - Difficult to perform manually

- Need a more systematic (algorithmic) way
  - Can use computer-based logic synthesis tools
    - Exp: espresso logic minimization software
  - Karnough Map (K-map) can be used for manual design of digital circuits.

The Map Method

- The truth table representation of a function is unique.
- But, not the algebraic expression
  - Several versions of an algebraic expression exist.
  - Difficult to minimize algebraic functions manually.
- The map method is a simple procedure to minimize Boolean functions.
  - Pictorial form of a truth table.
  - Called *Karnough Map* or *K-Map*.
    - Boolean expressions can be minimized by combining terms
    - This way, K-maps minimize equations graphically

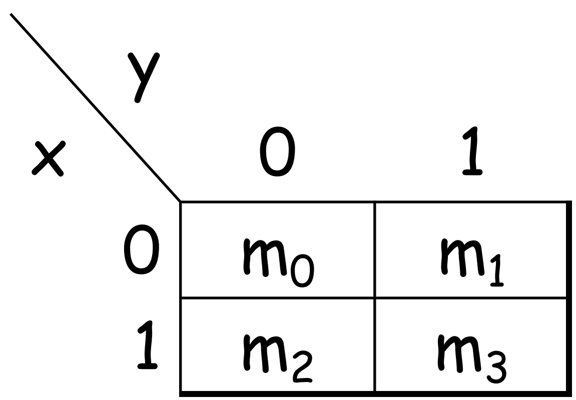
Two-Variable K-Map

- Two variables: x and y

  → 4 minterms:

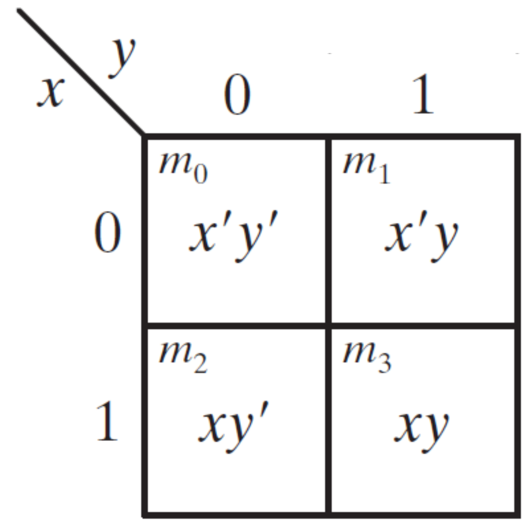
  - $m_0 = x'y'$ → 00
  - $m_1 = x'y$ → 01
  - $m_2 = xy'$ → 10
  - $m_3 = xy$ → 11

- Four squares (cells) for four minterms



(a)

- Figure (b) shows the relationship between the squares and the variables *x* and *y*.



(b)

Example: Two-Variable K-Map

x \ y	0	1
0	1	1
1	1	0

- $F = m_0 + m_1 + m_2 = x'y' + x'y + xy'$
- F = …
- F = …
- F = …

Remember the Shortcuts

- $x + x = x \quad \Leftrightarrow \quad x \cdot x = x$
- $x + 1 = 1 \Leftrightarrow x \cdot 0 = 0$
- $x + xy = x \Leftrightarrow x \cdot (x+y) = x$ [Absorption]
- $(x + y)' = x' \cdot y' \Leftrightarrow (x.y)'=x'+y'$ [DeMorgan]

Example: Two-Variable K-Map

x \ y	0	1
0	1	1
1	1	0

- $F = m_0 + m_1 + m_2 = x'y' + x'y + xy'$
- F = …
- F = …
- F = …
- $F = x' + y'$

- We can do the same optimization by combining <u>adjacent cells.</u>

Example: Two-Variable K-Map

Strategy:

1. Cover as many adjacent 1 cells as possible by the orders of two in circles in each direction
   - e.g., 1x1, 1x2, 2x1, 2x2, 1x4, 4x1, 2x4, 4x2, 4x4, ...
   - Every 1 must be circled at least once.
   - Each circle must be as large as possible
   - Adjacency is established either vertically or horizontally (not diagonally: cells touching each other by corners only are not adjacent)
2. Express each circle as a product of literals by only including the literals that are the same (all true or all complement) in the circle
3. Form the final expression by OR(+)ing the circle expressions

x \ y	0	1
0	1	1
1	1	0

K-Map Definitions

- **Complement:** variable with a bar over it

  $\bar{A}, \bar{B}, \bar{C}$
- **Literal:** variable or its complement

  $\bar{A}, A, \bar{B}, B, C, \bar{C}$
- **Implicant:** product of literals

  $A\bar{B}C, \bar{A}C, BC$
- **Prime implicant:** implicant corresponding to the largest circle in a K-map

Example: Two-Variable K-Map

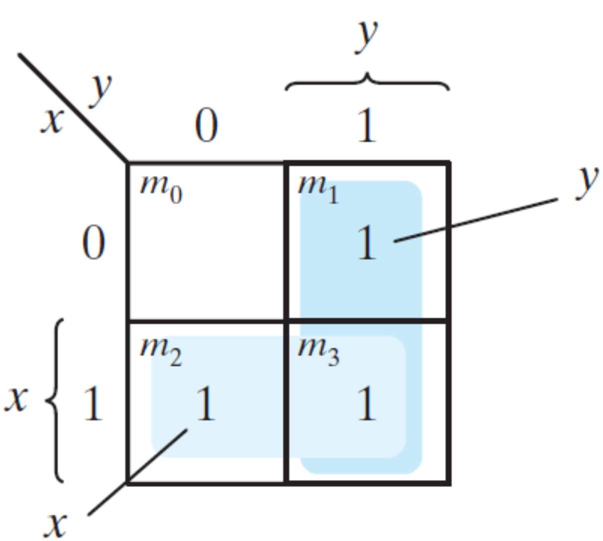
Strategy:

1. **Cover as many adjacent 1 cells as possible by the orders of two in circles in each direction**
   - **e.g., 1x1, 1x2, 2x1, 2x2, 1x4, 4x1, 2x4, 4x2, 4x4, ...**
   - **Every 1 must be circled at least once.**
   - **Each circle must be as large as possible**
   - **Adjacency is established either vertically or horizontally (not diagonally: cells touching each other by corners only are not adjacent)**
2. **Express each circle as a product of literals by only including the literals that are the same (all true or all complement) in the circle**
3. **Form the final expression by OR(+)ing the circle expressions**

x \ y	0	1
0	1	1
1	1	0

Two-Variable K-Map (Cont.)

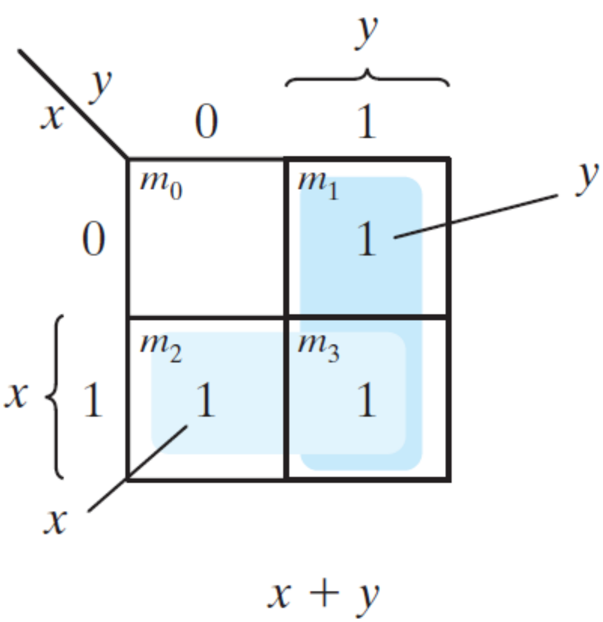
- May only be useful to represent 16 Boolean functions.



Two-Variable K-Map (Cont.)

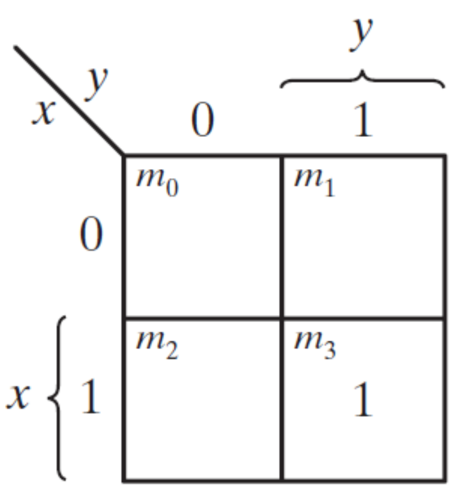
- May only be useful to represent 16 Boolean functions.
  - Exp: If *m1=m2=m3=1 then*

    m1+m2+m3=x'y+xy'+xy=x+y (OR function)



Two-Variable K-Map (Cont.)

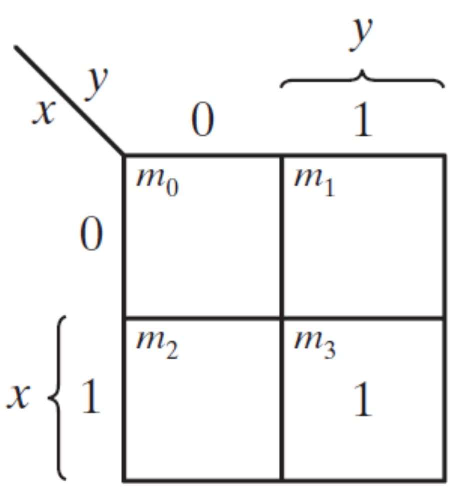
- May only be useful to represent 16 Boolean functions.



Two-Variable K-Map (Cont.)

- May only be useful to represent 16 Boolean functions.
  - Exp: If *only m3=1 then*

    m1=xy (AND function)



xy

Three-Variable Map

- There are 8 minterms for 3 variables.
- So, there are 8 squares.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

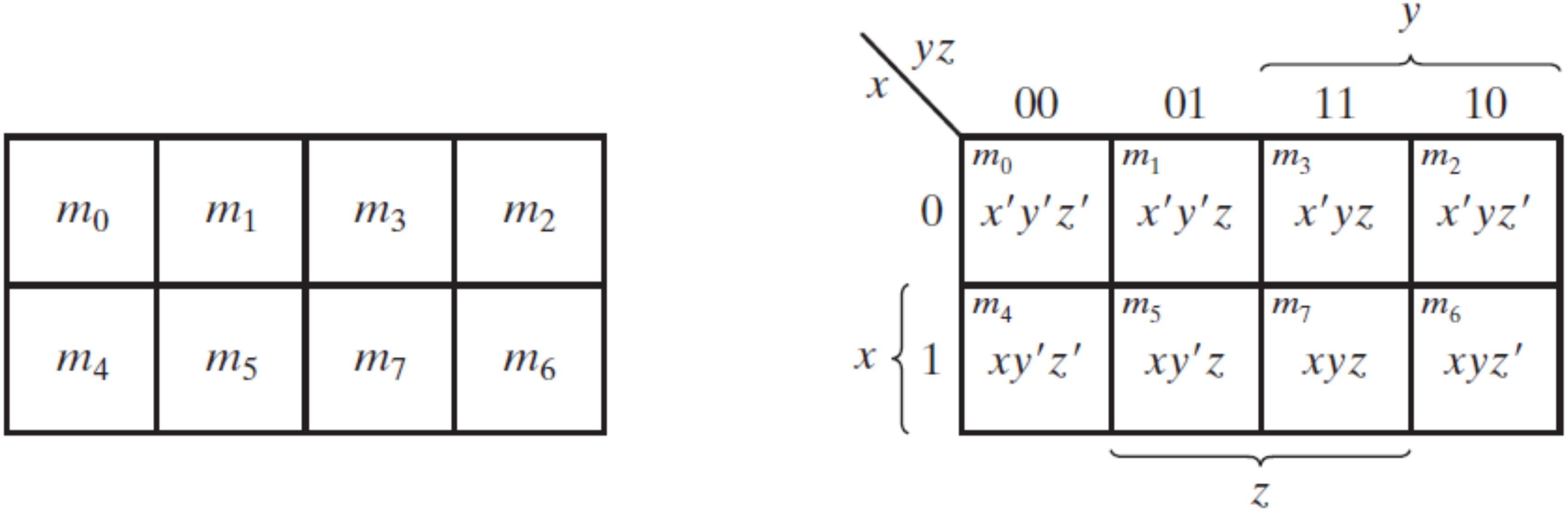
$x \backslash yz$	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x 1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

y (columns 11, 10); z (columns 01, 11)

(b)

Three-Variable Map

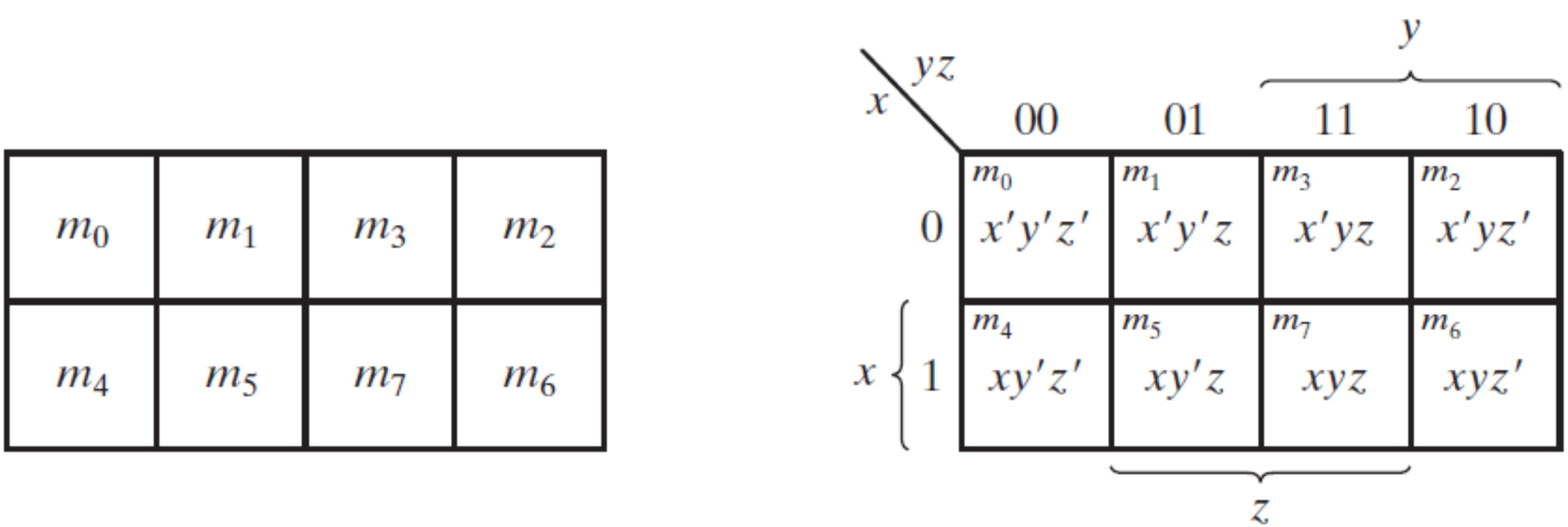
- **Minterms are arranged not in a binary sequence, but in a sequence similar to the Gray code.**
- <u>Adjacent squares</u>: they **differ by only one variable**, which is complement (primed) in one square and true (not primed) in the other
  - $m_2 \leftrightarrow m_6$, $m_3 \leftrightarrow m_7$
  - $m_2 \leftrightarrow m_0$, $m_6 \leftrightarrow m_4$



(a) (b)

Three-Variable Map (Cont.)

- For instance, the square for square *m5* corresponds to row 1 and column 01: *101*
  - Another way to look: *m5=xy’z*
- When variables are 0, they are primed (x’), Otherwise not primed (x)



(a) (b)

Three-Variable Map (Cont.)

- Two adjacent squares differ by one variable (one primed, the other is not).
  - So, they can be minimized
  - Ex: *m5+m7=xy'z+xyz=xz(y'+y)=xz*
- **try to cover as many adjacent squares as possible by the orders of two.**

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

$x \backslash yz$	00	01	11 (y)	10 (y)
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x 1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'
		z	z	

(b)

Example

A	B	C	Y
0	0	0	**1**
0	0	1	**1**
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Y

C \ AB	00	01	11	10
0	1	0	0	0
1	1	0	0	0

Example

$$Y = \overline{A}\,\overline{B}$$

A	B	C	Y
0	0	0	**1**
0	0	1	**1**
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Y — C \ AB	00	01	11	10
0	1	0	0	0
1	1	0	0	0

Example

Y: C \ AB	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

Truth Table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

K-Map

Y: C \ AB	00	01	11	10
0	0	1	1	0
1	0	1	0	0

Example

C \ AB (Y)	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

Truth Table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

K-Map

C \ AB (Y)	00	01	11	10
0	0	1	1	0
1	0	1	0	0

$$Y = \bar{A}B + B\bar{C}$$

Adjacent squares

- **Squares on opposite edges are adjacent.** These adjacent squares don’t touch each other.

  >> Circles may wrap around the edges

  - *m0 is adjacent to m1, m2 and m4*
  - *m4 is adjacent to m0, m5 and m6*
  - *m0+m2 = x’y’z’+x’yz’ = x’z’(y’+y) = x’z’*
  - *m4+m6 = xy’z’+xyz’ = xz’(y’+y) = xz’*

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

					y	
	$x \backslash yz$	00	01	11	10
	0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x	1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'
				z		

(b)

In 3-Variable Karnaugh Maps

- 1 single square represents a minterm with 3 literals
- Circle of 2 adjacent squares represent a term with 2 literals
- Circle of 4 adjacent squares represent a term with 1 literal
- Circle of 8 adjacent squares produce F=1

Example: Three-Variable K-Map

- $F_1(x, y, z) = \sum (2, 3, 4, 5)$

x \ yz	00	01	11	10
0				
1				

- $F_1(x, y, z) =$

- $F_2(x, y, z) = \sum (3, 4, 6, 7)$

x \ yz	00	01	11	10
0				
1				

- $F_2(x, y, z) =$

Example: Three-Variable K-Map

- $F_1(x, y, z) = \sum (2, 3, 4, 5)$

x \ yz	00	01	11	10
0	0	0	1	1
1	1	1	0	0

- $F_1(x, y, z) =$

- $F_2(x, y, z) = \sum (3, 4, 6, 7)$

x \ yz	00	01	11	10
0	0	0	1	0
1	1	0	1	1

- $F_2(x, y, z) =$

Example: Three-Variable K-Map

- $F_1(x, y, z) = \sum (2, 3, 4, 5)$

x \ yz	00	01	11	10
0	0	0	1	1
1	1	1	0	0

- $F_1(x, y, z) = xy' + x'y$

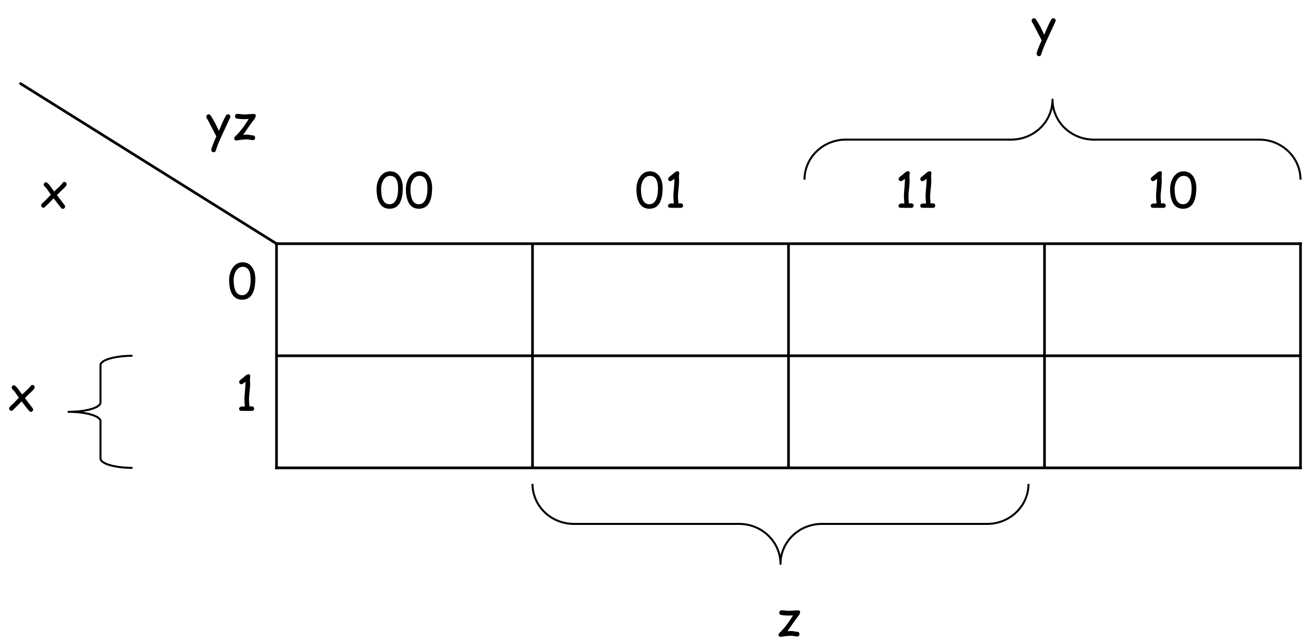
- $F_2(x, y, z) = \sum (3, 4, 6, 7)$

x \ yz	00	01	11	10
0	0	0	1	0
1	1	0	1	1

- $F_2(x, y, z) = xz' + yz$

Example

- $F_1(x, y, z) = \sum (0, 2, 4, 5, 6)$



$F_1(x, y, z) =$

Example

- $F_1(x, y, z) = \sum (0, 2, 4, 5, 6)$

y

x \ yz	00	01	11	10
0	1	0	0	1
x: 1	1	1	0	1

z

$F_1(x, y, z) =$

Example

- $F_1(x, y, z) = \sum (0, 2, 4, 5, 6)$

y

x \ yz	00	01	11	10
0	1	0	0	1
x { 1	1	1	0	1

z

$F_1(x, y, z) =$

Finding Sum of Minterms

- If a function is not expressed in sum of minterms form, it is possible to get it using K-map
  - Example: $F(x, y, z) = x'z + x'y + xy'z + yz$

x \ yz	00	01	11	10
0				
1				

Finding Sum of Minterms

- If a function is not expressed in sum of minterms form, it is possible to get it using K-map
  - <u>Example</u>: $F(x, y, z) = x'z + x'y + xy'z + yz$

x \ yz	00	01	11	10
0				
1				

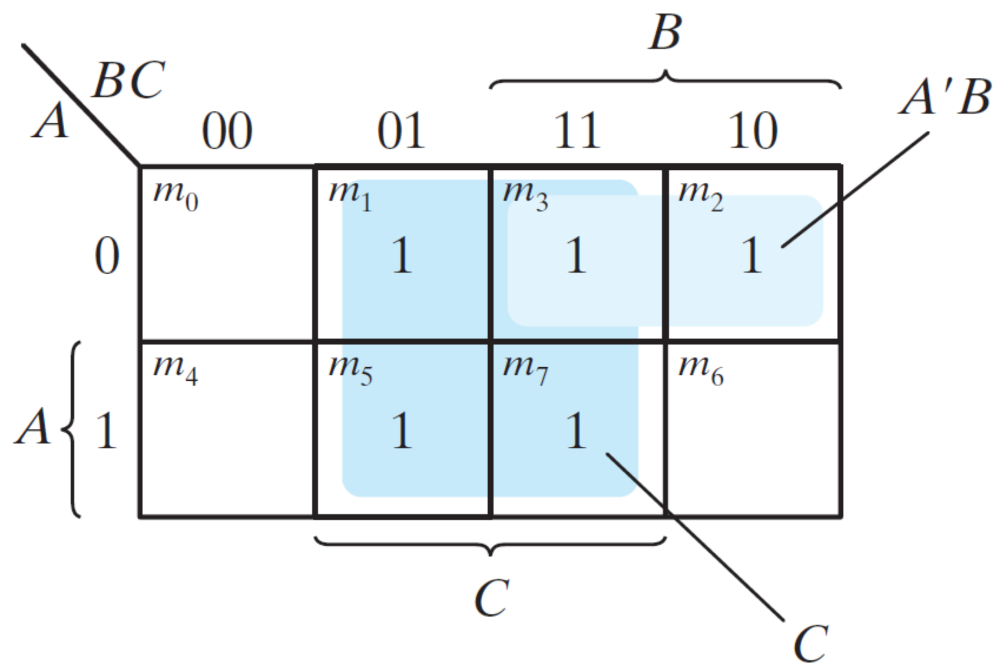
$F(x, y, z) = x'y'z + x'yz + x'yz' + xy'z + xyz$

Example

- $F = A'C + A'B + AB'C + BC$
  - Express F as a sum of minterms.
    - $F(A,B,C) =$

Example

- *F=A'C+A'B+AB'C+BC*
  - Express *F* as a sum of minterms.
    - *F(A,B,C)=Σ(1,2,3,5,7)*
  - Find the minimal sum-of-products expression
    - *F=C+A'B*



Four-Variable K-Map

- Four variables: x, y, z, t
  - >> 8 literals
  - >> 16 minterms

z

xy \ zt	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

y

x

t

Four-Variable Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

(a)

$wx \backslash yz$	00	01	11	10
00	m_0 $w'x'y'z'$	m_1 $w'x'y'z$	m_3 $w'x'yz$	m_2 $w'x'yz'$
01	m_4 $w'xy'z'$	m_5 $w'xy'z$	m_7 $w'xyz$	m_6 $w'xyz'$
11	m_{12} $wxy'z'$	m_{13} $wxy'z$	m_{15} $wxyz$	m_{14} $wxyz'$
10	m_8 $wx'y'z'$	m_9 $wx'y'z$	m_{11} $wx'yz$	m_{10} $wx'yz'$

y x w z

(b)

Example: Four-Variable K-Map

$F(x,y,z,t) = \Sigma\ (0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

xy \ zt	00	01	11	10
00				
01				
11				
10				

$F(x,y,z,t) =$

Example: Four-Variable K-Map

$F(x,y,z,t) = \Sigma\,(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

xy \ zt	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	0	0

$F(x,y,z,t) =$

Example: Four-Variable K-Map

$F(x,y,z,t) = \Sigma\,(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

xy \ zt	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	0	0

$F(x,y,z,t) =$

Example: Four-Variable K-Map

- $F(x,y,z,t) = x'y'z' + y'zt' + x'yzt' + xy'z'$

xy \ zt	00	01	11	10
00				
01				
11				
10				

- $F(x,y,z,t) =$

Example: Four-Variable K-Map

- $F(x,y,z,t) = x'y'z' + y'zt' + x'yzt' + xy'z'$

xy \ zt	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

- $F(x,y,z,t) =$

Example: Four-Variable K-Map

- $F(x,y,z,t) = x'y'z' + y'zt' + x'yzt' + xy'z'$

xy \ zt	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

- $F(x,y,z,t) =$

Example

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Y

CD \ AB	00	01	11	10
00				
01				
11				
10				

Example

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Y

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

Example

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Y

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

$$Y = \overline{A}C + \overline{A}BD + A\overline{B}\,\overline{C} + \overline{B}\,\overline{D}$$

Prime Implicants

- **Prime Implicant**: is a product term obtained by combining maximum possible number of adjacent cells in the map
  - A single 1 on the map represents a prime implicant if it is not adjacent to any other 1s.
  - Two adjacent 1s form a prime implicant, provided that they are not within a group of four adjacent 1s.
  - So on..

- Circle = Prime Implicant

Essential Prime Implicants

- If a minterm can be covered by only one prime implicant, that prime implicant is said to be an **Essential Prime Implicant**

- In other words:

If a prime implicant covers a minterm that is not covered by any other, then it is an *essential prime implicant*

Example: Prime Implicants

- F(x,y,z,t) = Σ (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)

xy \ zt	00	01	11	10
00				
01				
11				
10				

Example: Prime Implicants

- $F(x,y,z,t) = \Sigma\ (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

xy \ zt	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

Example: Prime Implicants

- $F(x,y,z,t) = \Sigma\,(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

xy \ zt	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

$F(x,y,z,t) = y't' + yt + xy' + zt$

- Which ones are the **essential prime implicants?**

Example: Prime Implicants

- $F(x,y,z,t) = \Sigma\,(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

xy \ zt	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

- Why are these the ***essential prime implicants?***

Example: Prime Implicants

- $F(x,y,z,t) = \Sigma\,(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$

xy \ zt	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

xy \ zt	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

- $y't'$ -> essential since m_0 is covered only in it
- yt -> essential since m_5 is covered only in it
- They together cover $m_0, m_2, m_8, m_{10}, m_5, m_7, m_{13}, m_{15}$

Example: Prime Implicants

xy \ zt	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

xy \ zt	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

- m_3, m_9, m_{11} are not yet covered.
- How do we cover them?
- There is actually <u>more than one way.</u>

Example: Prime Implicants

xy \ zt	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	1	0
10	1	1	1	1

- Both y'z and zt cover m_3 and m_{11}.
- m_9 can be covered in two different prime implicants: xt or xy'

- m_3, m_{11} → zt or y'z
- m_9 → xy' or xt

Example: Prime Implicants

- $F(x, y, z, t) = yt + y't' + zt + xt$
- or $F(x, y, z, t) = yt + y't' + zt + xy'$
- or $F(x, y, z, t) = yt + y't' + y'z + xt$
- or $F(x, y, z, t) = yt + y't' + y'z + xy'$
- What to do?
  1. Find out all the essential prime implicants
  2. Then find the other prime implicants that cover the minterms that are not covered by the essential prime implicants. More than one way to choose those.
  3. Simplified expression is the logical sum of the essential implicants plus the other implicants

Five-Variable Map

- Downside:
  - Karnaugh maps with more than four variables are not simple to use anymore.
  - 5 variables → 32 squares, 6 variables → 64 squares
  - Somewhat more practical way for F(x, y, z, t, w) :

yz \ tw	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

x = 0

yz \ tw	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

x = 1

Five-Variable Maps

- Adjacency:
  - Each square in the $x = 0$ map is adjacent to the corresponding square in the $x = 1$ map.
  - For example, $m_4 \leftrightarrow m_{20}$ and $m_{15} \leftrightarrow m_{31}$

yz \ tw	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

$x = 0$

yz \ tw	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

$x = 1$

Five-Variable Map

- Not simple, it is not usually used.
- 5 variables >> 32 squares.

$A = 0$

$BC \backslash DE$	00	01	11	10
00	m_0 0	m_1 1	m_3 3	m_2 2
01	m_4 4	m_5 5	m_7 7	m_6 6
11	m_{12} 12	m_{13} 13	m_{15} 15	m_{14} 14
10	m_8 8	m_9 9	m_{11} 11	m_{10} 10

D (columns 11, 10); E (columns 01, 11); B (rows 11, 10); C (rows 01, 11)

$A = 1$

$BC \backslash DE$	00	01	11	10
00	m_{16} 16	m_{17} 17	m_{19} 19	m_{18} 18
01	m_{20} 20	m_{21} 21	m_{23} 23	m_{22} 22
11	m_{28} 28	m_{29} 29	m_{31} 31	m_{30} 30
10	m_{24} 24	m_{25} 25	m_{27} 27	m_{26} 26

C (columns 11, 10); E (columns 01, 11); B (rows 11, 10); C (rows 01, 11)

Example: Five-Variable Map

$$F(x, y, z, t, w) = \Sigma\,(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$$

yz \ tw	00	01	11	10
00				
01				
11				
10				

$x = 0$

yz \ tw	00	01	11	10
00				
01				
11				
10				

$x = 1$

- $F(x,y,z,t,w) =$

Example: Five-Variable Map

$F(x, y, z, t, w) = \Sigma\,(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$

yz \ tw	00	01	11	10
00	1			1
01	1			1
11		1		
10		1		

x = 0

yz \ tw	00	01	11	10
00				
01		1	1	
11		1	1	
10		1		

x = 1

- F(x,y,z,t,w) =

Example: Five-Variable Map

$F(x, y, z, t, w) = \Sigma\,(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$

yz \ tw	00	01	11	10
00	1			1
01	1			1
11		1		
10		1		

x = 0

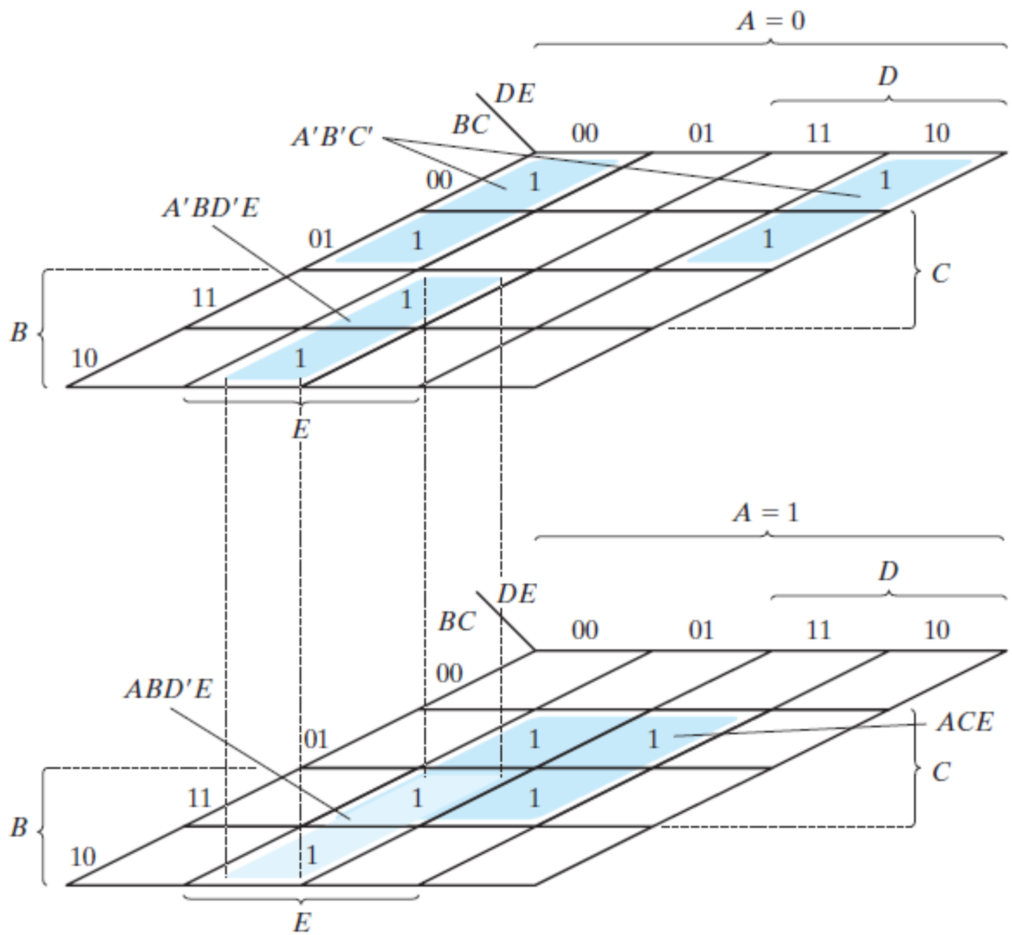
yz \ tw	00	01	11	10
00				
01		1	1	
11		1	1	
10		1		

x = 1

- $F(x,y,z,t,w) =$

Example

- An alternative look would be:



Many-Variable Maps

- 6-variables: Use **four** 4-variable maps to obtain 64 squares required for six variable optimization
- Alternatively: Can use computer optimization
  - Quine-McCluskey method
  - Espresso method

Product of Sums Simplification

- So far we have seen simplified expressions from Karnaugh maps in sum of products form.

- But simplified product of sums can also be derived from Karnaugh maps.
- Method:
  - A square with 1 represents a "minterm"
  - Similarly an empty square (a square with 0) represents a "maxterm".
  - Treat the 0s in the same manner as we treat 1s

  >> Combine them as we did for sum-of-products
- This way we obtain *F'* (complement of the function)
- Take (the complement of *F'*) : *((F')')* to obtain *F*

Example: Product of Sums

- $F(x, y, z, t) = \Sigma\ (0, 1, 2, 5, 8, 9, 10)$
  - Simplify this function in
    a. sum of products
    b. product of sums

xy \ zt	00	01	11	10
00	1	1		1
01		1		
11				
10	1	1		1

$F(x, y, z, t) =$

Example: Product of Sums

- F(x, y, z, t) = Σ (0, 1, 2, 5, 8, 9, 10)
  - Simplify this function in
    a. sum of products
    b. product of sums

xy \ zt	00	01	11	10
00	1	1		1
01		1		
11				
10	1	1		1

F(x, y, z, t) =

Example: Product of Sums

- $F(x, y, z, t) = \Sigma\,(0, 1, 2, 5, 8, 9, 10)$
  - Simplify this function in
    a. sum of products
    b. product of sums

xy \ zt	00	01	11	10
00	1	1		1
01		1		
11				
10	1	1		1

$$F(x,y,z,t) = y't' + y'z' + x'z't$$

Example: Product of Sums

1. Find F'(x,y,z,t) in Sum of Products form by grouping 0's
2. Apply DeMorgan's theorem to F' (use dual theorem) to find F in Product of Sums form

xy \ zt	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

Example: Product of Sums

1. Find F’(x,y,z,t) in Sum of Products form by grouping 0’s
2. Apply DeMorgan’s theorem to F’ (use dual theorem) to find F in Product of Sums form

xy \ zt	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

Example: Product of Sums

1. Find F’(x,y,z,t) in Sum of Products form by grouping 0’s
2. Apply DeMorgan’s theorem to F’ (use dual theorem) to find F in Product of Sums form

xy \ zt	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

$$F' = yt' + zt + xy$$

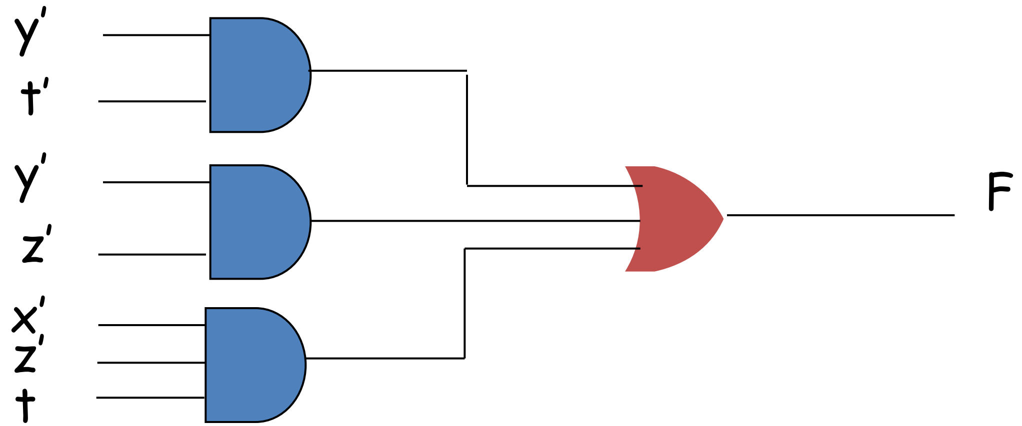
Example: Product of Sums

1. Find F'(x,y,z,t) in Sum of Products form by grouping 0's
2. Apply DeMorgan's theorem to F' (use dual theorem) to find F in Product of Sums form

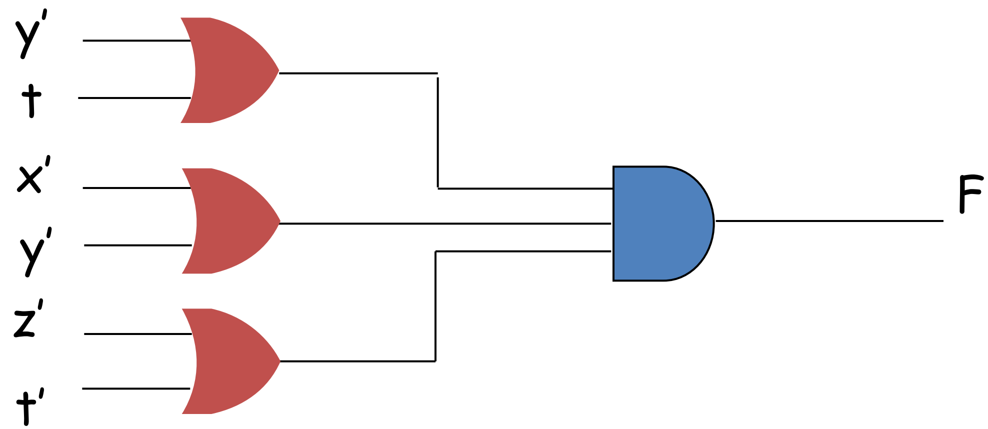
xy \ zt	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

$$F(x,y,z,t) = (y'+t)(z'+t')(x'+y')$$

Example: Product of Sums



F(x,y,z,t) = y't' + y'z' + x'z't: sum of products implementation



F = (y' + t)(x' + y')(z' + t'): product of sums implementation

Product of Maxterms

- If the function is originally expressed in the product of maxterms canonical form, the procedure is also valid
- Example:

$F(x, y, z) = \Pi\ (0, 2, 5, 7)$

x \ yz	00	01	11	10
0				
1				

F(x, y, z) =

Product of Maxterms

- If the function is originally expressed in the product of maxterms canonical form, the procedure is also valid
- Example:

$F(x, y, z) = \Pi\,(0, 2, 5, 7)$

x \ yz	00	01	11	10
0	0			0
1		0	0	

$F'(x, y, z) = xz + x'z'$

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

Product of Maxterms

- If the function is originally expressed in the product of maxterms canonical form, the procedure is also valid
- Example:

$F(x, y, z) = \Pi\ (0, 2, 5, 7)$

x \ yz	00	01	11	10
0	0	1	1	0
1	1	0	0	1

$F'(x, y, z) = xz + x'z'$

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

$F(x, y, z) = x'z + xz'$

Product of Sums

- To enter a function F expressed in product of sums in the map
  1. take its complement: F’
  2. Find the squares corresponding to the terms in F’
  3. Fill these square with 0’s and others with 1’s.
- Example:
  – $F(x, y, z, t) = (x' + y' + z')(y + t)$
  – $F'(x, y, z, t) =$

xy \ zt	00	01	11	10
00				
01				
11				
10				

Product of Sums

- To enter a function F expressed in product of sums in the map
  1. take its complement, F’
  2. Find the squares corresponding to the terms in F’
  3. Fill these square with 0’s and others with 1’s.
- Example:
  - $F(x, y, z, t) = (x' + y' + z')(y + t)$
  - $F'(x, y, z, t) =$

xy \ zt	00	01	11	10
00	0			0
01				
11			0	0
10	0			0

Don’t Care Conditions

- Some functions are undefined for certain input combinations
  - Such function are referred as <u>incompletely specified functions</u>
  - **therefore, the corresponding output values do not have to be defined**
  - This may significantly reduce the circuit complexity
  - Example: Four bit binary codes has six unused combinations.

Unspecified Minterms

- For unspecified minterms, we do not care what value the function produces.
- Unspecified minterms of a function are called don’t care conditions.
- We use “X” symbol to represent them in a Karnaugh map.
- Useful for further simplification
- **The Xs in the map can be taken as 0 or 1 to realize the best simplification.**

>> a “don't care” (X) is circled (i.e., taken as 1) only if it helps to minimize the equation

Example: Don’t Care Conditions

- $F(x, y, z, t) = \Sigma(1, 3, 7, 11, 15)$: function
- $d(x, y, z, t) = \Sigma(0, 2, 5)$: don’t care conditions

xy \ zt	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

$F =$

$F_1 =$ or

$F_2 =$

Example: Don’t Care Conditions

- $F(x, y, z, t) = \Sigma(1, 3, 7, 11, 15)$: function
- $d(x, y, z, t) = \Sigma(0, 2, 5)$: don’t care conditions

xy \ zt	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

$F =$

$F_1 =$ or

$F_2 =$

Example: Don't Care Conditions

- $F_1 = zt + x'y' = \Sigma(0, 1, 2, 3, 7, 11, 15)$
- $F_2 = zt + x't = \Sigma(1, 3, 5, 7, 11, 15)$
- The two functions are algebraically unequal
  - But as far as the function (F with d) is concerned: both alternatives are acceptable
- Look at the simplified product of sums expression for the same function F.

xy \ zt	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

$F' =$

$F =$

Example: Don't Care Conditions

- $F_1 = zt + x'y' = \Sigma(0, 1, 2, 3, 7, 11, 15)$
- $F_2 = zt + x't = \Sigma(1, 3, 5, 7, 11, 15)$
- The two functions are algebraically unequal
  - But as far as the function F with d is concerned: both functions are acceptable
- Look at the simplified product of sums expression for the same function F.

xy \ zt	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

$F' =$

$F =$

K-Maps with Don't Cares

A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Y

CD \ *AB*	00	01	11	10
00				
01				
11				
10				

K-Maps with Don’t Cares

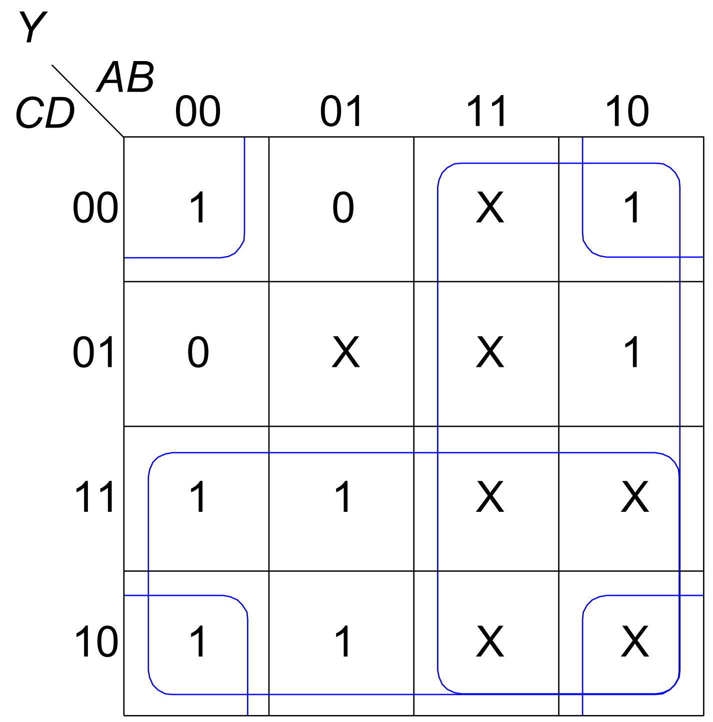
A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Y

CD \ *AB*	00	01	11	10
00	1	0	X	1
01	0	X	X	1
11	1	1	X	X
10	1	1	X	X

K-Maps with Don't Cares

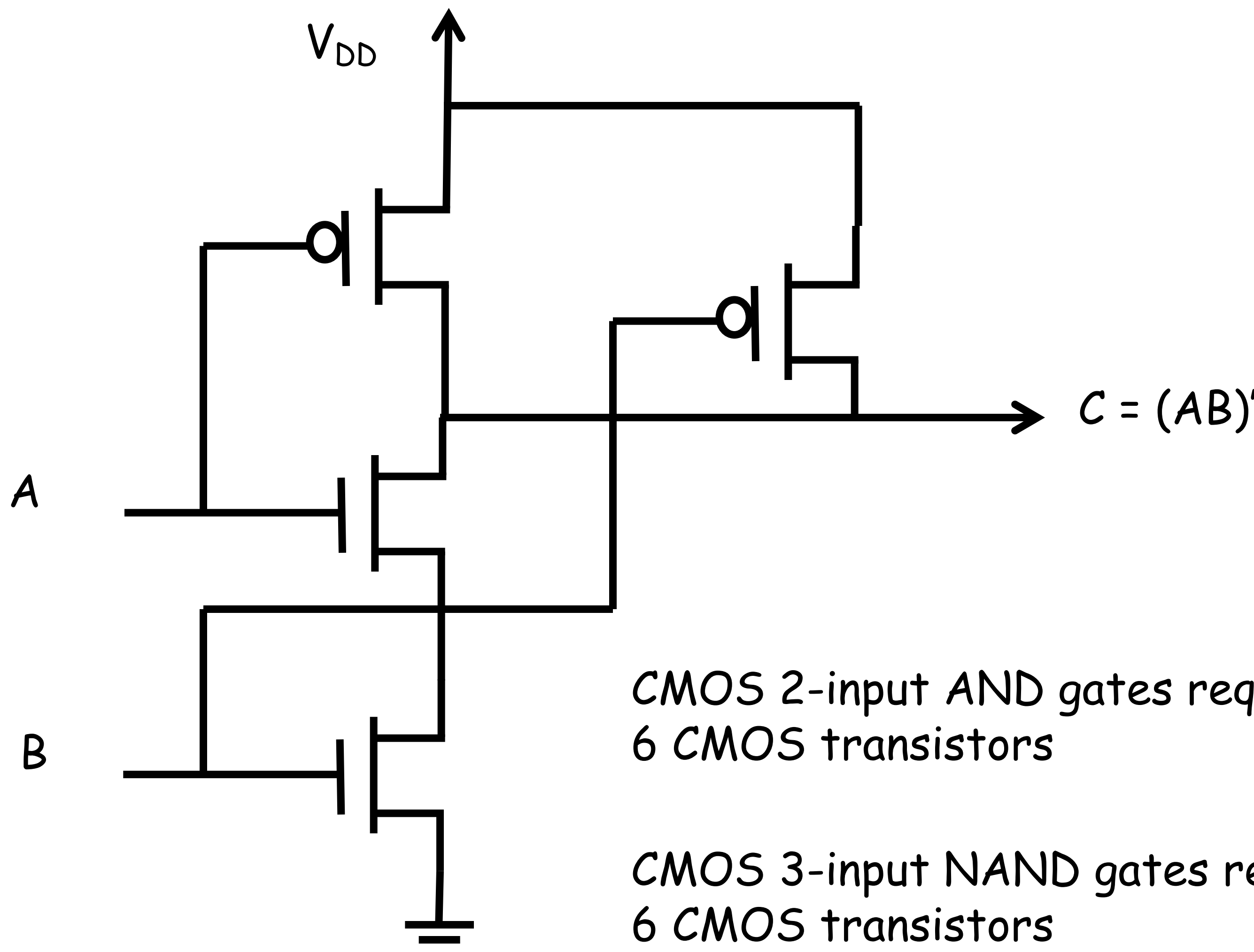
A	*B*	*C*	*D*	*Y*
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



$$Y = A + \overline{B}\overline{D} + C$$

NAND and NOR Gates

- NAND and NOR gates are easier to fabricate

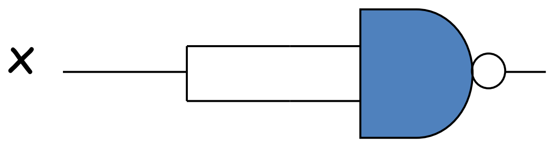


CMOS 2-input AND gates requires
6 CMOS transistors

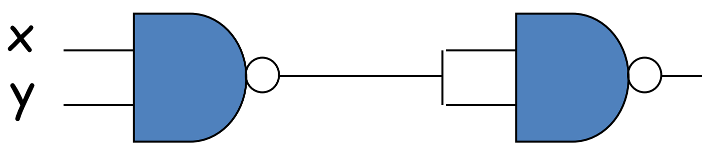
CMOS 3-input NAND gates requires
6 CMOS transistors

Design with NAND or NOR Gates

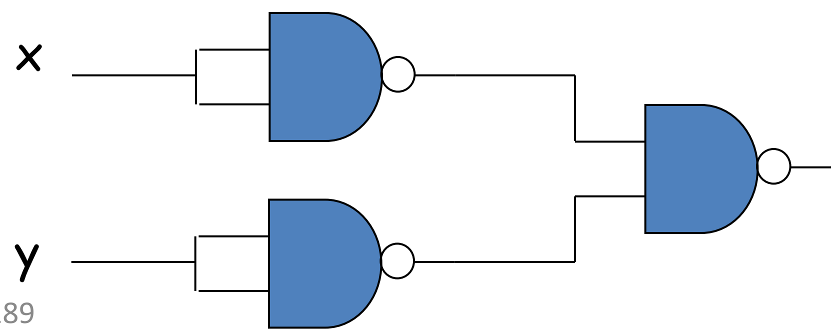
- It is beneficial to derive conversion rules <u>from</u> Boolean functions given in terms of AND, OR, an NOT gates <u>into</u> equivalent NAND or NOR implementations



$(x\,x)' = x'$ ➔ NOT

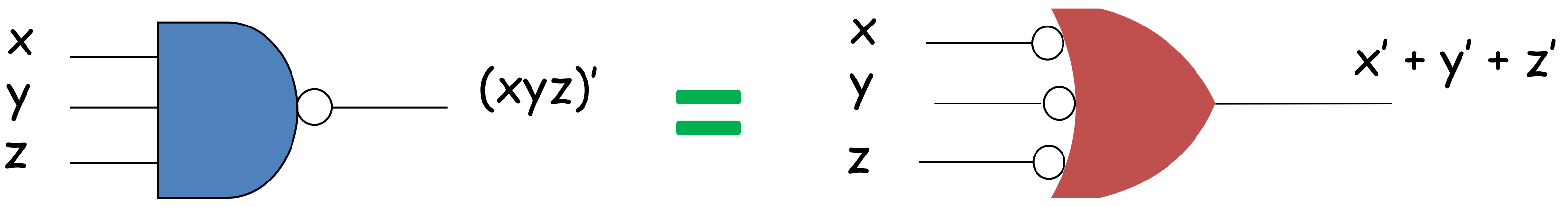


$[\,(x\,y)'\,]' = x\,y$ ➔ AND



$(x'\,y'\,)' = x + y$ ➔ OR

New Notation

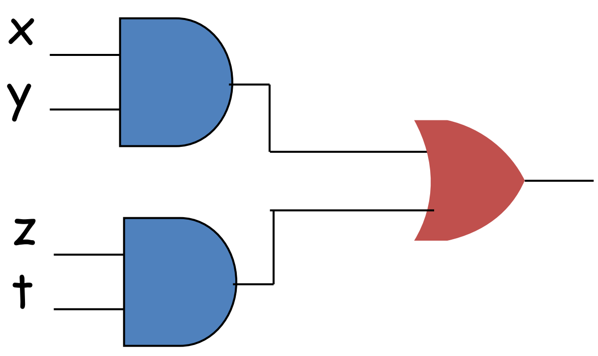


AND-invert

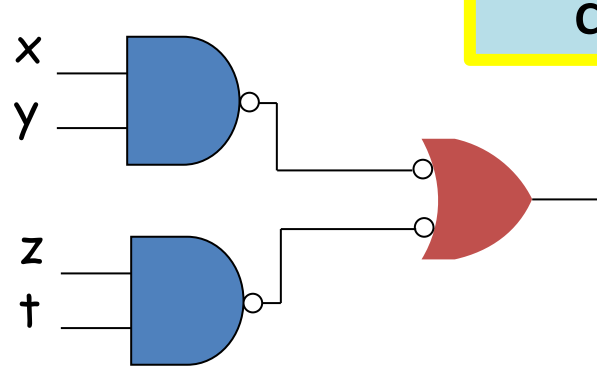
Invert-OR

- Implementing a Boolean function with NAND gates is easy if it is in <u>sum of products form</u>.
- Example: F(x, y, z, t) = xy + zt

ALWAYS USE THIS INTERMEDIATE FORM WITH BUBBLES TO AVOID CONFUSION

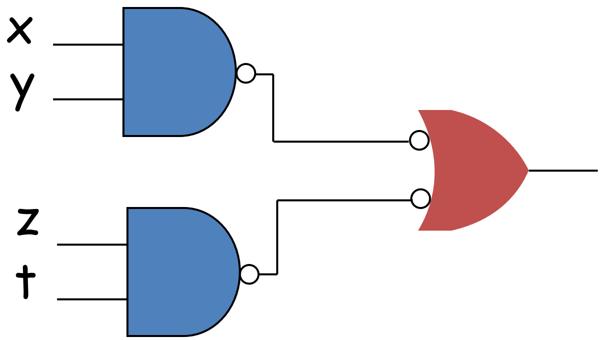


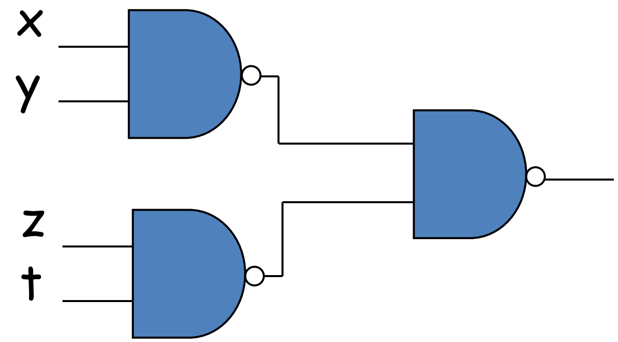
F(x, y, z, t) = xy + zt



F(x, y, z, t) = ((xy)')' + ((zt)')'

The Conversion Method





$((xy)')' + ((zt)')' \quad = xy + zt \quad = \quad [\,(xy)'\,(zt)'\,]'$

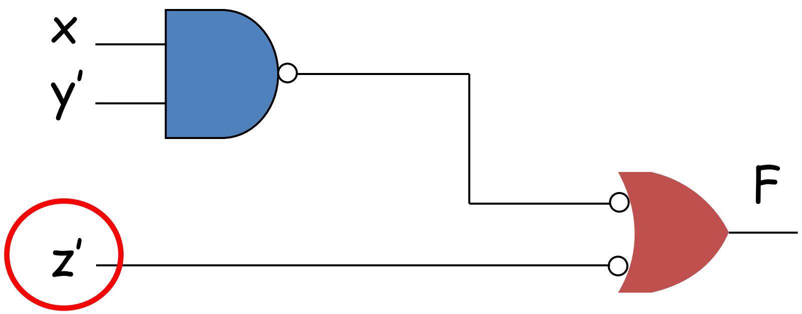
- Example: $F(x, y, z) = \sum(1, 3, 4, 5, 7)$

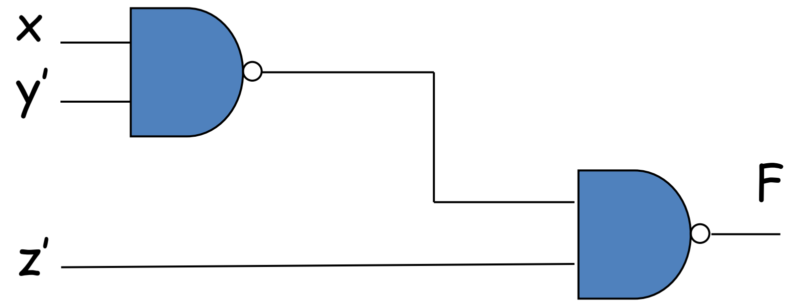
x \ yz	00	01	11	10
0		1	1	
1	1	1	1	

$F = z + xy'$

$F = (z')' + ((xy')')'$

Example: Design with NAND Gates





$F = (z')' + ((xy')')'$

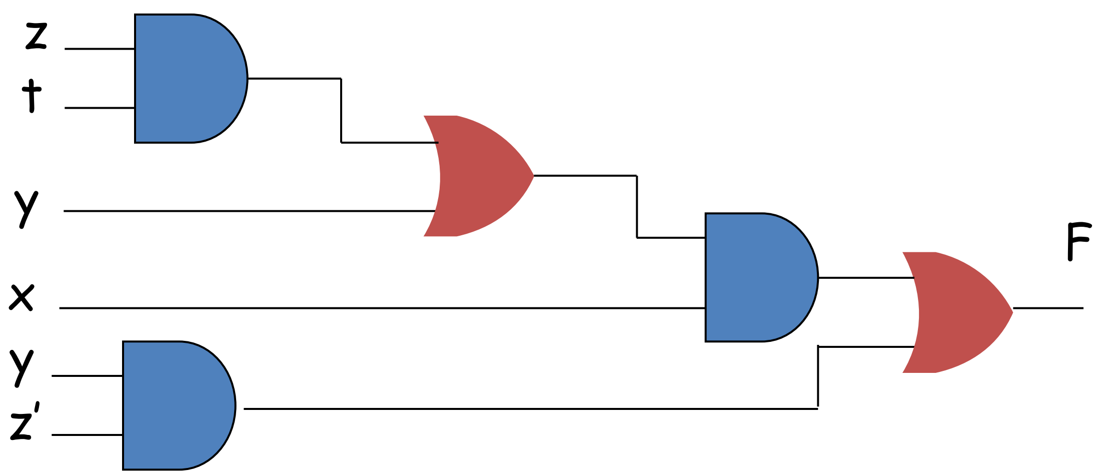
$F = z + xy'$

- Summary
  1. Simplify the function
  2. Draw a NAND gate for each product term
  3. Draw a NAND gate for the OR gate in the 2^{nd} level,
  4. **A product term with single literal needs an inverter in the first level.** <u>Assume single, complemented literals are available.</u>

Multi-Level NAND Gate Designs

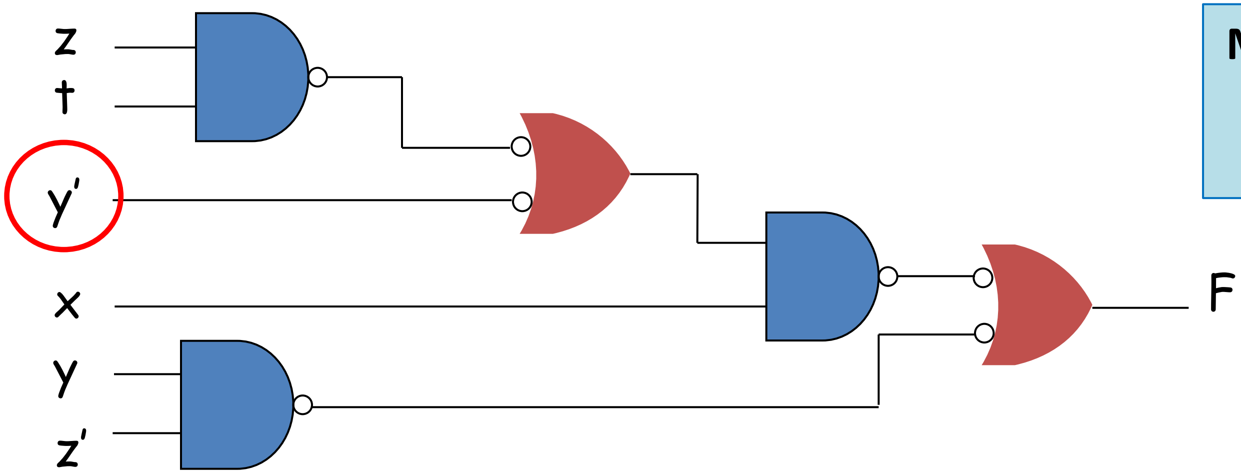
- The standard form results in two-level implementations
- Non-standard forms may raise a difficulty
- Example: F = x(zt + y) + yz’

4-level implementation

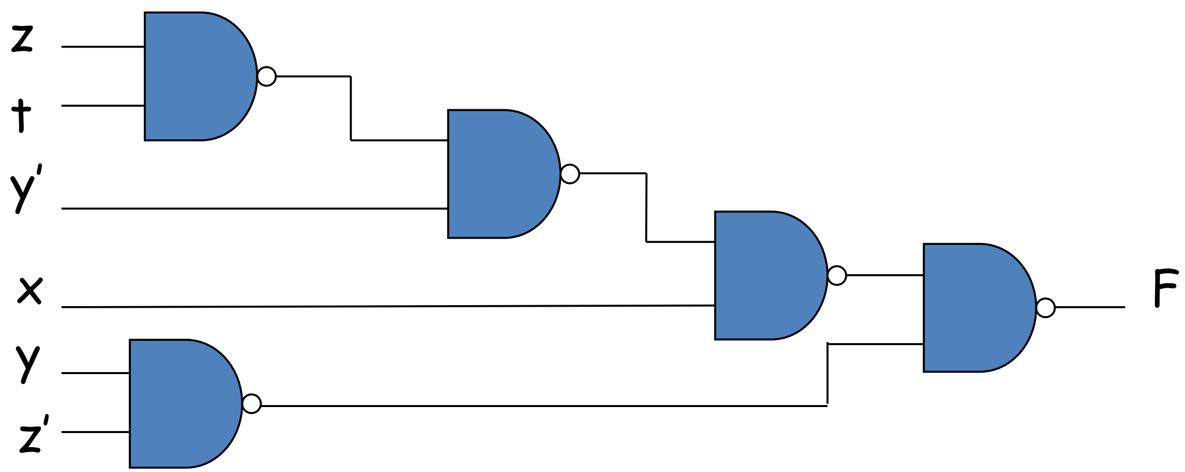


Example: Multilevel NAND…

$F = x(zt + y) + yz'$



MUST COMPENSATE FOR EVERY NON-COUPLED BUBBLE



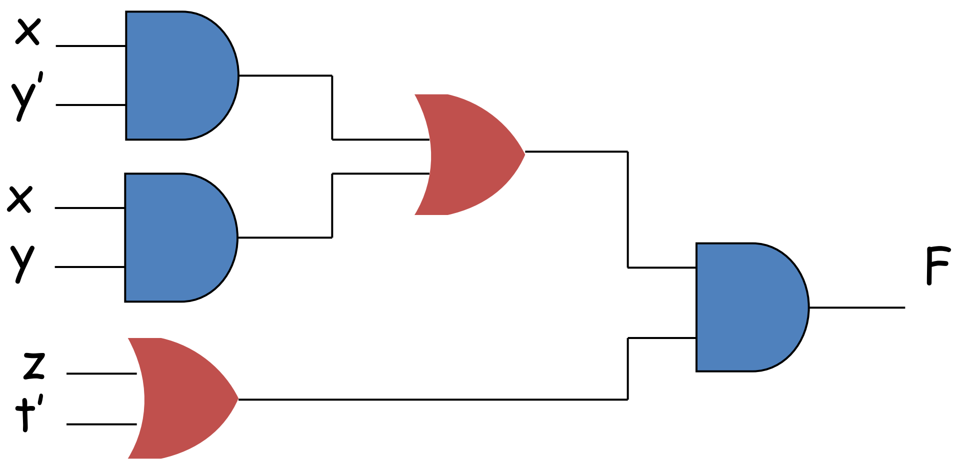
Design with Multi-Level NAND Gates

<u>Rules</u>

1. Convert all AND gates to NAND gates
2. Convert all OR gates to NAND gates
3. **Insert an inverter (one-input NAND gate) at the output if the final operation is AND**
4. Check the bubbles in the diagram. For every bubble along a path from input to output there must be another bubble. **If not so, complement the input literal**

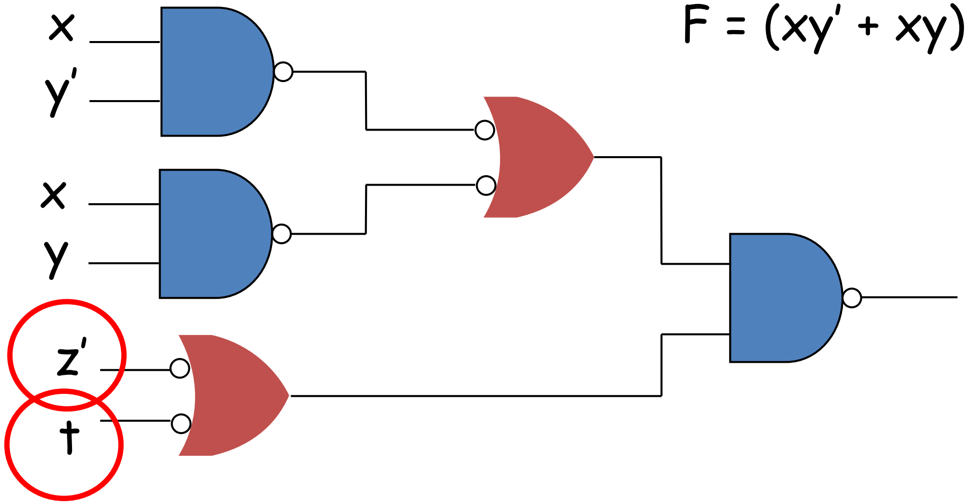
Another (Harder) Example

- Example: F = (xy’ + xy)(z + t’)
  - (three-level implementation)



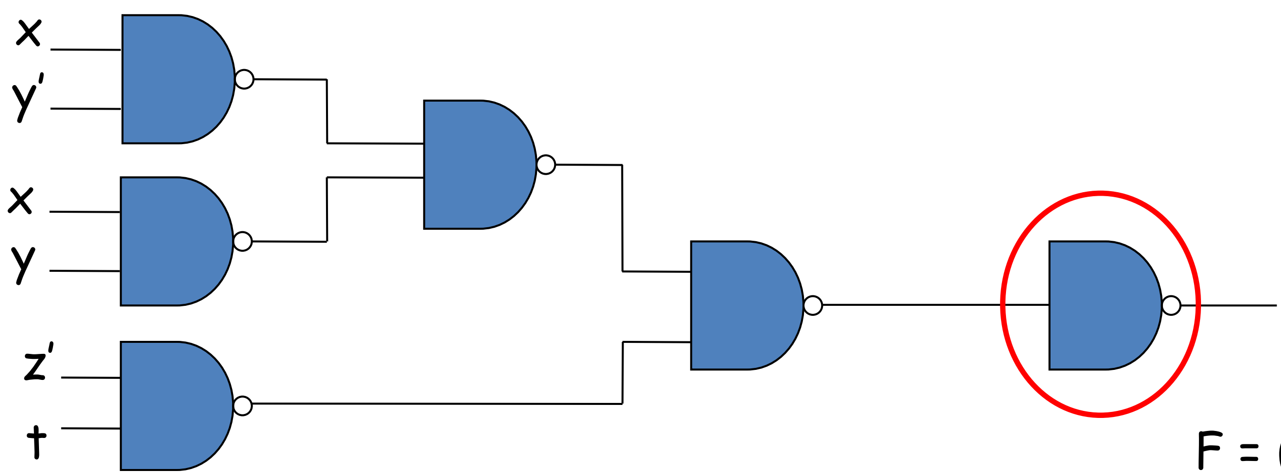
Example: Multi-Level NAND Gates

$F = (xy' + xy)(z + t')$



$G = [\,(xy' + xy)(z' + t)\,]'$

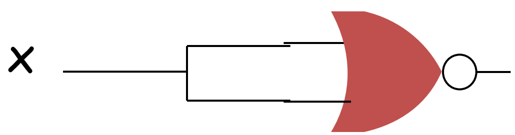
$F = (xy' + xy)(z + t')$



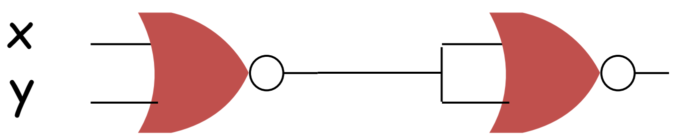
$F = (xy' + xy)(z + t')$

Design with NOR Gates

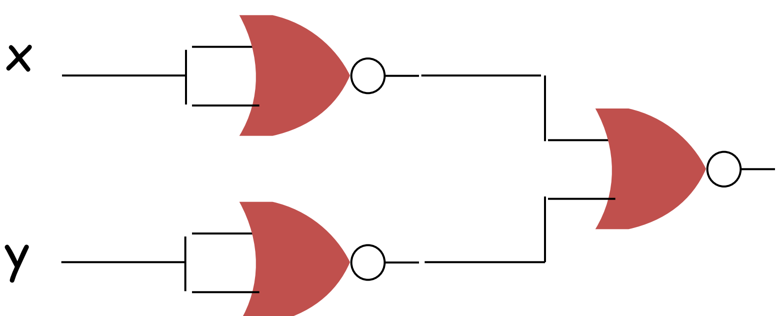
- NOR is the dual operation of NAND.
  - All rules and procedure we used in the design with NAND gates apply here in a similar way.
  - Function is implemented easily if it is in <u>product of sums form</u>.



$(x + x)' = x' \rightarrow$ NOT

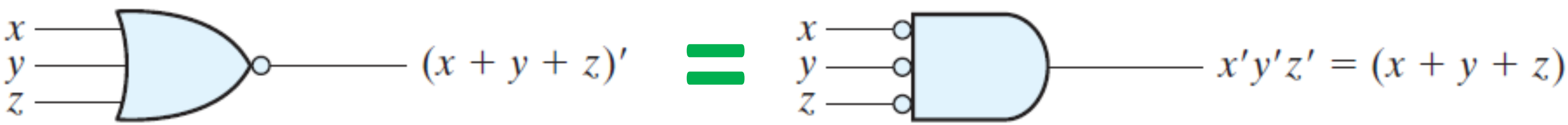


$[\,(x+ y)'\,]' = x + y \rightarrow$ OR



$(x' + y'\,)' = x \cdot y \rightarrow$ AND

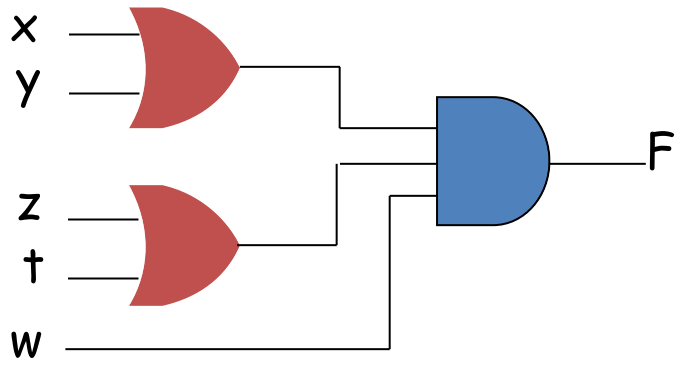
Logic operations with NOR gates

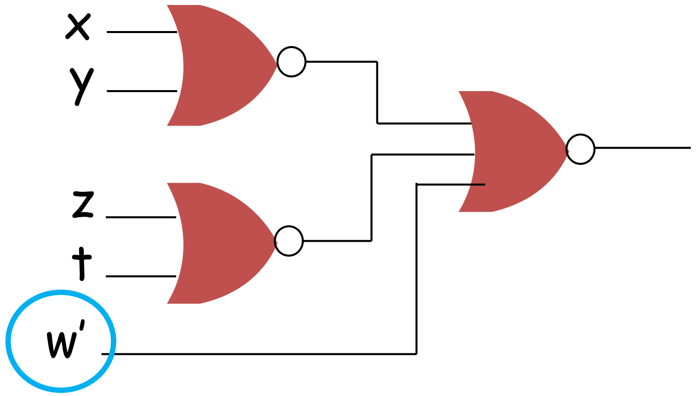


(a) OR-invert (b) Invert-AND

Example: Design with NOR Gates

- F = (x+y) (z+t) w

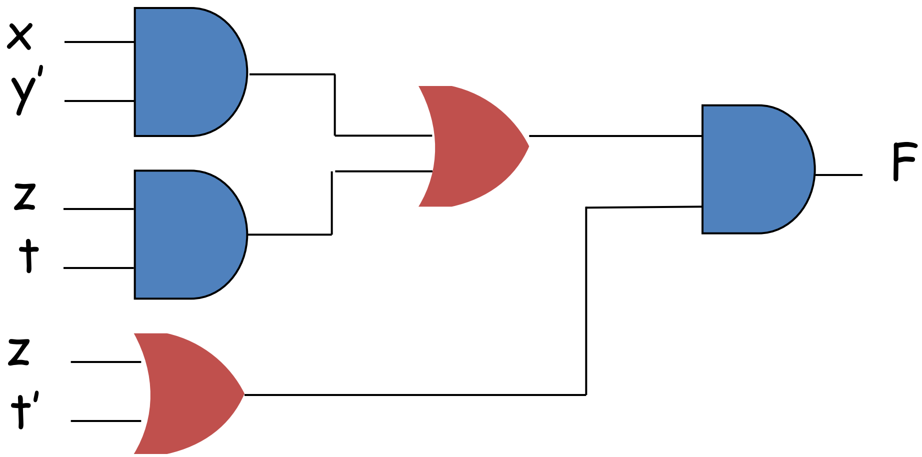


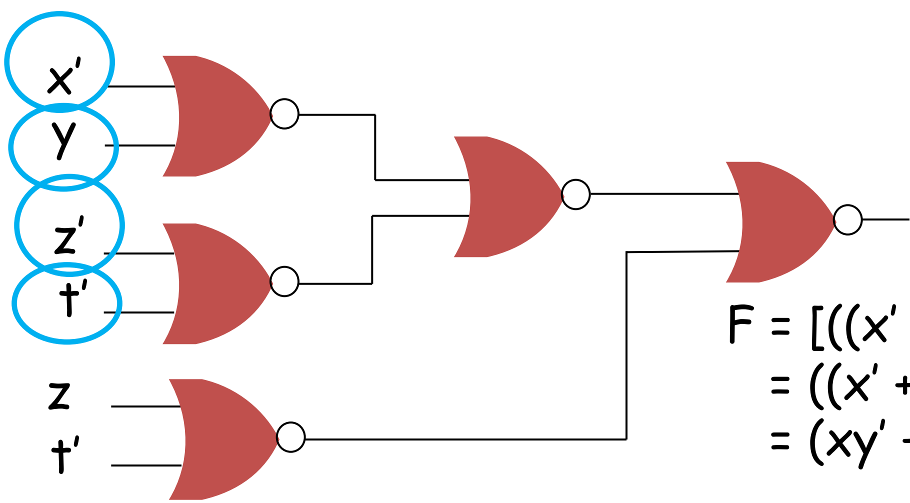


F = (x + y) (z + t) w

Example: Design with NOR Gates

- $F = (xy' + zt)(z + t')$

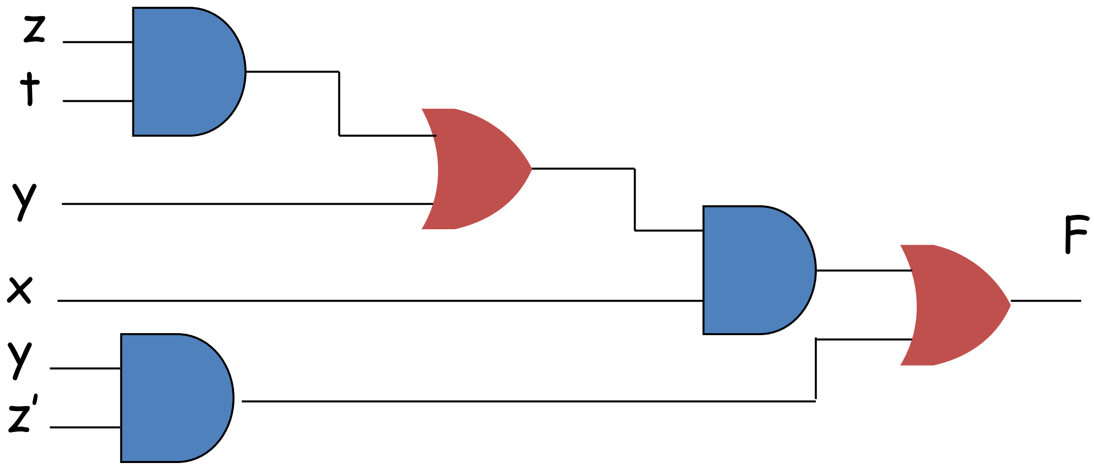


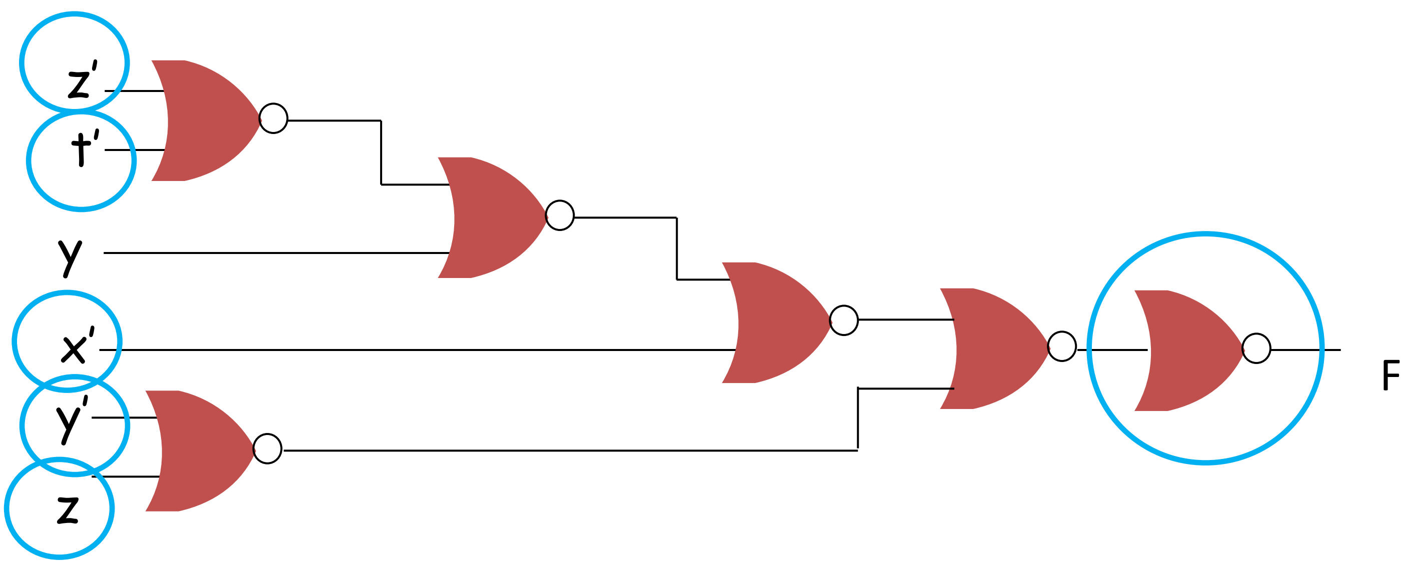


$$
\begin{aligned}
F &= [((x' + y)' + (z' + t')')' + (z + t')']' \\
&= ((x' + y)' + (z' + t')')(z + t') \\
&= (xy' + zt)(z + t')
\end{aligned}
$$

Harder Example

- Example: $F = x(zt + y) + yz'$





Exclusive-OR Function

- The symbol: $\oplus$
  - $x \oplus y = xy' + x'y$
  - $(x \oplus y)' = xy + x'y'$

- Properties
  - $x \oplus 0 = x$
  - $x \oplus 1 = x'$
  - $x \oplus x = 0$
  - $x \oplus x' = 1$
  - $x \oplus y' = x' \oplus y = (x \oplus y)'$: XNOR

- Commutative & Associative
  - $x \oplus y = y \oplus x$
  - $(x \oplus y) \oplus z = x \oplus (y \oplus z)$

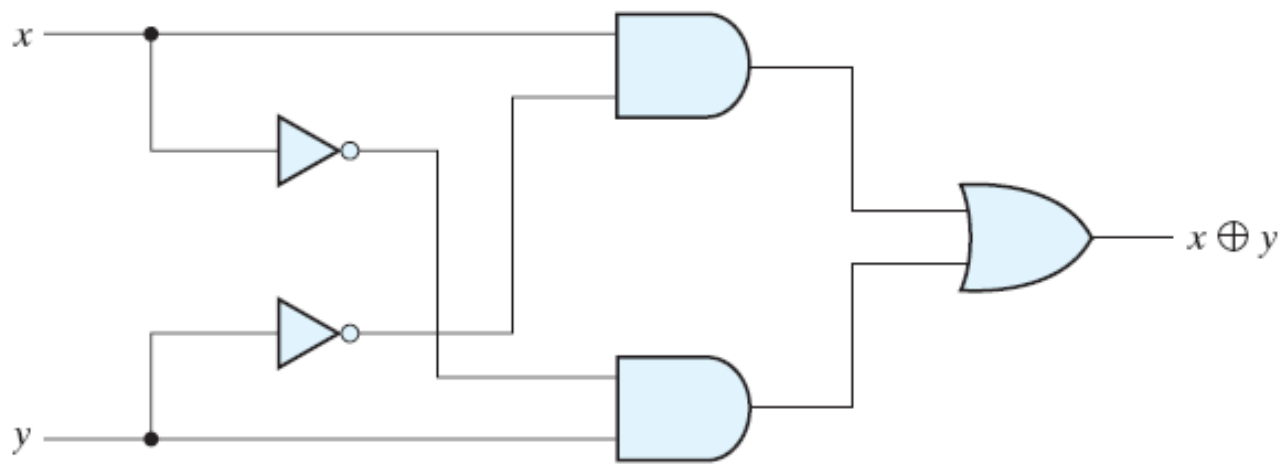
Exclusive-OR Function

- XOR gate is **not** universal
  - Only a limited number of Boolean functions can be expressed in terms of XOR gates
- XOR operation has very important applications in arithmetic and error-detection circuits.
- Odd Function

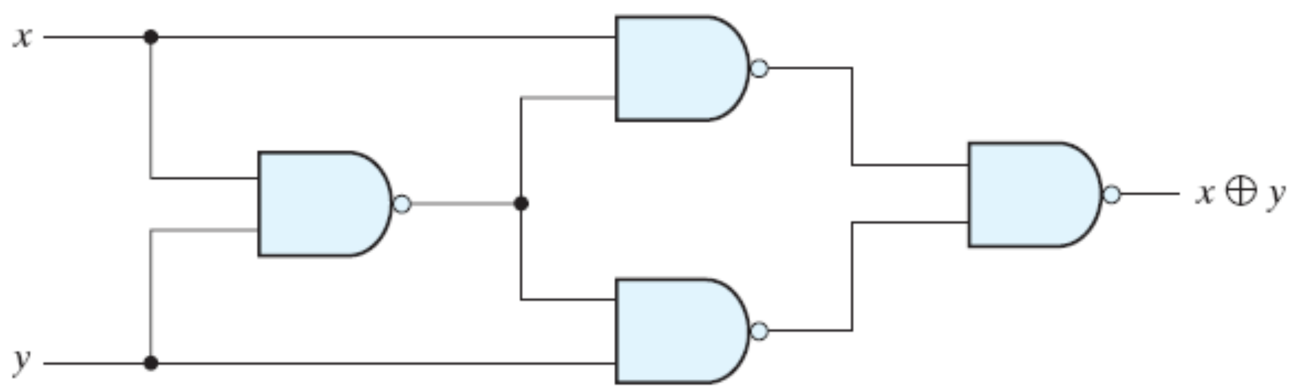
$$
\begin{aligned}
(x \oplus y) \oplus z &= (xy' + x'y) \oplus z \\
&= (xy' + x'y)\, z' + (xy' + x'y)'\, z \\
&= xy'z' + x'yz' + (xy + x'y')\, z \\
&= xy'z' + x'yz' + xyz + x'y'z \\
&= \Sigma\,(4, 2, 7, 1)
\end{aligned}
$$

x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

XOR Implementations



(a) With AND–OR–NOT gates



(b) With NAND gates

Odd Function

- If an odd number of variables are equal to 1, then the function is equal to 1.
- Therefore, multivariable XOR operation is referred as the Odd function.

x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Odd function

x \ yz	00	01	11	10
0	1	0	1	0
1	0	1	0	1

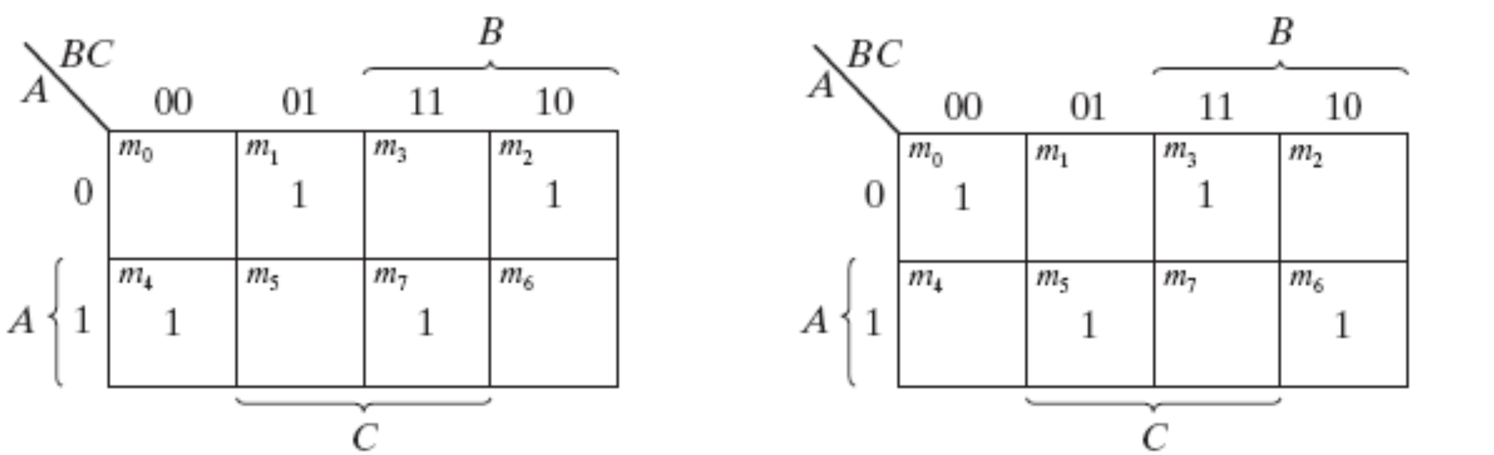
Even function

Odd Function

- n variable XOR function is odd function defined as:

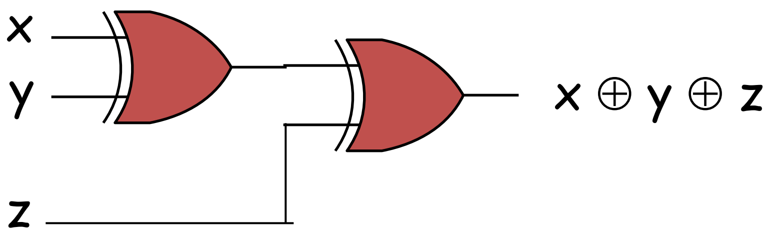
The logical sum of the $2^n/2$ minterms whose binary numerical values have an odd number of 1's

- If n=3, 4 minterms have odd number of 1's



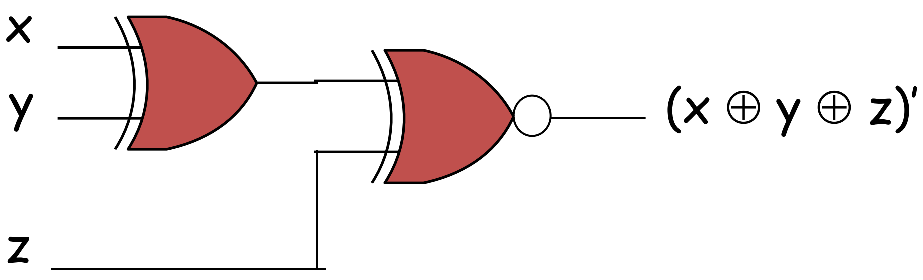
(a) Odd function $F = A \oplus B \oplus C$ (b) Even function $F = (A \oplus B \oplus C)'$

Odd & Even Functions



Odd function

- $(x \oplus y \oplus z)' = ((x \oplus y) \oplus z)'$



Parity Generation

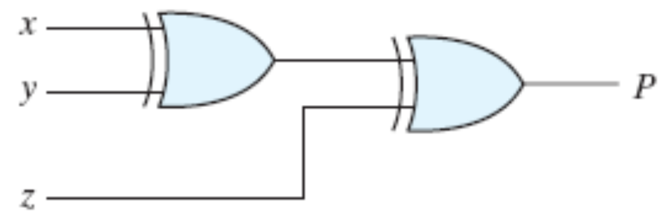
- A parity bit is an extra bit included with a binary message
  - e.g., odd parity: If number of 1's in data is even, parity bit is set to 1; else 0. So that the number of 1's including parity is always odd
- The circuit that generates the parity bit in the transmitter is called *parity generator*.
- Parity bit can be generated using XOR function.

7 bits of data	(count of 1-bits)	8 bits including parity	
		even	odd
0000000	0	0000000**0**	0000000**1**
1010001	3	1010001**1**	1010001**0**
1101001	4	1101001**0**	1101001**1**
1111111	7	1111111**1**	1111111**0**

3-Bit Parity Generator

Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	**y**	**z**	**P**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



(a) 3-bit even parity generator

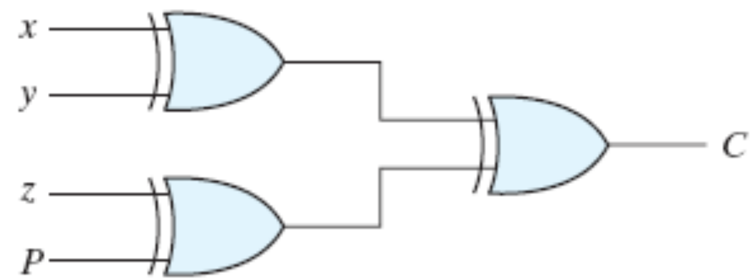
Parity Checker

- Bits are transmitted to the destination with parity.
- The circuit that checks the parity in the receiver is called a *parity checker*.
- Parity checker can be implmeneted with XOR gates.

4-Bit Parity Checker

Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	**y**	**z**	**P**	**C**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



(b) 4-bit even parity checker

Adder Circuit for Integers

- Addition of two 2-bit numbers
  - $Z = X + Y$
  - $X = (x_1\ x_0)$ and $Y = (y_1\ y_0)$
  - $Z = (z_2\ z_1\ z_0)$
- Bitwise addition
  1. $z_0 = x_0 \oplus y_0$ (sum)
     $c_1 = x_0\ y_0$ (carry)
  2. $z_1 = x_1 \oplus y_1 \oplus c_1$
     $c_2 = x_1\ y_1 + x_1\ c_1 + y_1\ c_1$
  3. $z_2 = c_2$

Adder Circuit

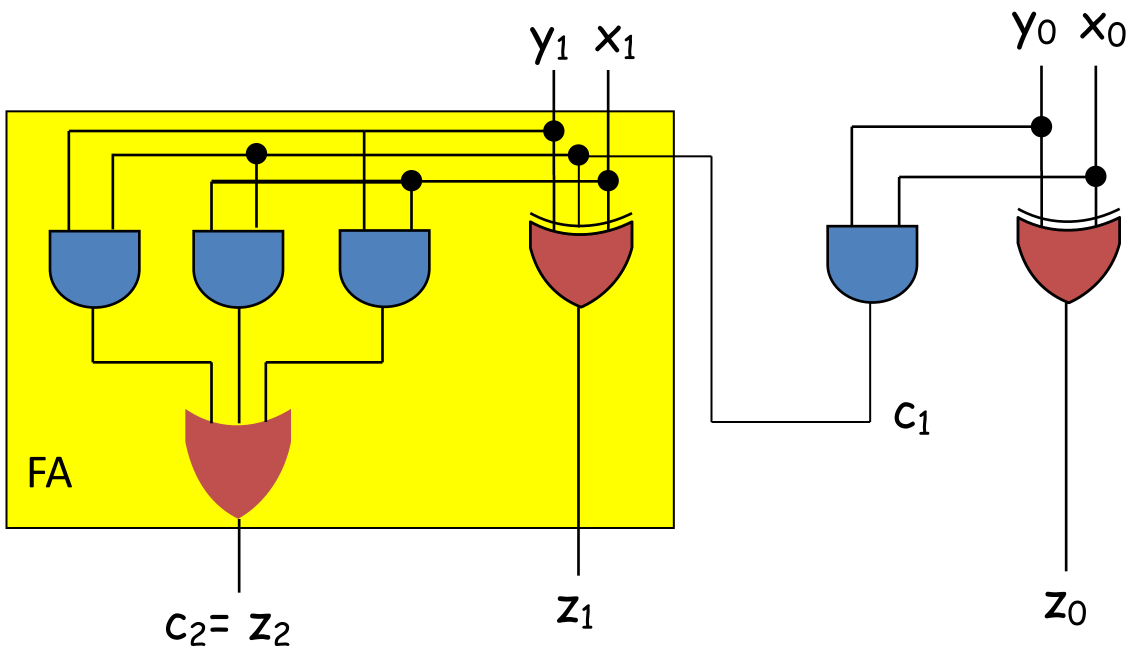
$z_1 = x_1 \oplus y_1 \oplus c_1$

$c_2 = x_1 y_1 + x_1 c_1 + y_1 c_1$

$z_0 = x_0 \oplus y_0$

$c_1 = x_0 y_0$

$z_2 = c_2$



Comparator Circuit

- F(X>Y)

  $X = (x_1\ x_0)$ and $Y = (y_1\ y_0)$

$x_1 x_0$ \ $y_1 y_0$	00	01	11	10
00				
01				
11				
10				

Comparator Circuit

- F(X>Y)

  $X = (x_1\ x_0)$ and $Y = (y_1\ y_0)$

$x_1 x_0$ \ $y_1 y_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$F(x_1, x_0, y_1, y_0) = x_1y_1' + x_1x_0y_0' + x_0y_0'y_1'$