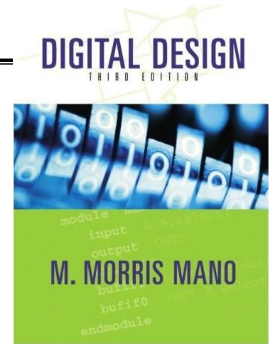
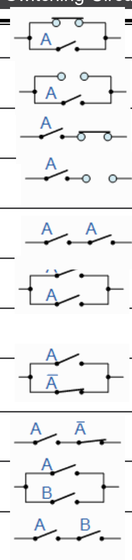


Digital Design 3e, Morris Mano

Chapter 2 – Boolean Algebra and Logic Gates

Dr. Eng. Kamalanath Samarakoon

Expression	Rule(s) Used
A(A + B) + (B + AA)(A + B)	Original Expression
AA + AB + (B + A)A + (B + A)B	Idempotent (AA to A), then Distributive, used twice.
AB + (B + A)A + (B + A)B	Complement, then Identity. (Strictly speaking, we also used the Commutative Law for each of these applications.)
AB + BA + AA + BB + AB	Distributive, two places.
AB + BA + A + AB	Idempotent (for the A's), then Complement and Identity to remove BB.
AB + AB + AT + AB	Commutative, Identity; setting up for the next step.
AB + A(B + T + B)	Distributive.
AB + A	Identity, twice (depending how you count it).
A + AB	Commutative.
(A + A)(A + B)	Distributive.
A + B	Complement, Identity.

Boolean Expression	Description	Equivalent Switching Circuit	Boolean Algebra Law or Rule
A + 1 = 1	A in parallel with closed = "CLOSED"		Annulment
A + 0 = A	A in parallel with open = "A"		Identity
A . 1 = A	A in series with closed = "A"		Identity
A . 0 = 0	A in series with open = "OPEN"		Annulment
A + A = A	A in parallel with A = "A"		Idempotent
A . A = A	A in series with A = "A"		Idempotent
NOT A = A	NOT NOT A (double negative) = "A"		Double Negation
A + A = 1	A in parallel with NOT A = "CLOSED"		Complement
A . A = 0	A in series with NOT A = "OPEN"		Complement
A+B = B+A	A in parallel with B = B in parallel with A		Commutative
A.B = B.A	A in series with B = B in series with A		Commutative
A+B = A.B	invert and replace OR with AND		de Morgan's Theorem
A.B = A+B	invert and replace AND with OR		de Morgan's Theorem

Overview

- Logic functions with 1's and 0's
  - Building digital circuitry
- Truth tables
- Logic symbols and waveforms
- Boolean algebra
- Properties of Boolean Algebra
  - Reducing functions
  - Transforming functions

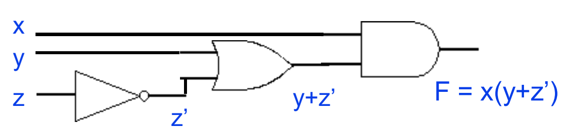
Part 2 - Overview

- Expressing Boolean functions
- Relationships between algebraic equations, symbols, and truth tables
- Simplification of Boolean expressions
- Minterms and Maxterms
- AND-OR representations
  - Product of sums
  - Sum of products

Boolean Functions

- Boolean algebra deals with binary variables and logic operations.
- Function results in binary 0 or 1

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

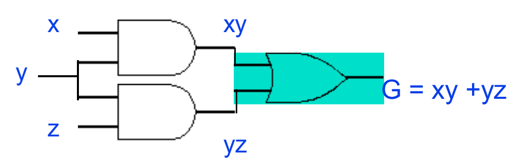


F = x(y+z')

Boolean Functions

- Boolean algebra deals with binary variables and logic operations.
- Function results in binary 0 or 1

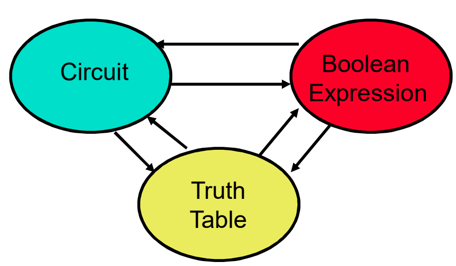
x	y	z	xy	yz	G
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	1	1



We will learn how to transition between equation, symbols, and truth table.

Representation Conversion

- Need to transition between boolean expression, truth table, and circuit (symbols).
- Converting between truth table and expression is easy.
- Converting between expression and circuit is easy.
- More difficult to convert to truth table.



Truth Table to Expression

- Converting a truth table to an expression
  - Each row with output of 1 becomes a product term
  - Sum product terms together.

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

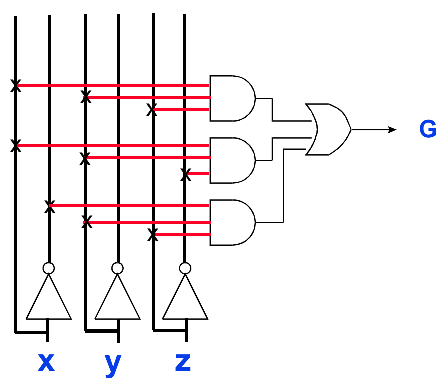
Any Boolean Expression can be represented in sum of products form!

xyz + xyz' + x'yz

Equivalent Representations of Circuits

- All three formats are equivalent
- Number of 1's in truth table output column equals AND terms for Sum-of-Products (SOP)

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



G = xyz + xyz' + x'yz

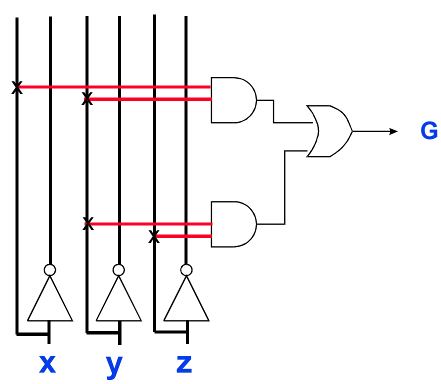
Reducing Boolean Expressions

- Is this the smallest possible implementation of this expression? **No!** G = xyz + xyz' + x'yz
- Use Boolean Algebra rules to reduce complexity while preserving functionality.
- Step 1: Use Theorem 1 (a + a = a)
  - So xyz + xyz' + x'yz = xyz + xyz + xyz' + x'yz
- Step 2: Use distributive rule a(b + c) = ab + ac
  - So xyz + xyz + xyz' + x'yz = xy(z + z') + yz(x + x')
- Step 3: Use Postulate 3 (a + a' = 1)
  - So xy(z + z') + yz(x + x') = xy.1 + yz.1
- Step 4: Use Postulate 2 (a . 1 = a)
  - So xy.1 + yz.1 = xy + yz = xyz + xyz' + x'yz

Reduced Hardware Implementation

- Reduced equation requires less hardware!
- Same function implemented!

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



G = xyz + xyz' + x'yz = xy + yz

Minterms and Maxterms

- Each variable in a Boolean expression is a literal
- Boolean variables can appear in normal (x) or complement form (x')
- Each AND combination of terms is a minterm
- Each OR combination of terms is a maxterm

For example:
Minterms

x	y	z	Minterm	
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
…				
1	0	0	$xy'z'$	m_4
…				
1	1	1	xyz	m_7

For example:
Maxterms

x	y	z	Maxterm	
0	0	0	$x+y+z$	M_0
0	0	1	$x+y+z'$	M_1
…				
1	0	0	$x'+y+z$	M_4
…				
1	1	1	$x'+y'+z'$	M_7

Representing Functions with Minterms

- Minterm number same as row position in truth table (starting from top from 0)
- Shorthand way to represent functions

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

$$G = xyz + xyz' + x'yz$$

$$\downarrow$$

$$G = m_7 + m_6 + m_3 = \Sigma(3, 6, 7)$$

Conversion Between Canonical Forms

- Easy to convert between minterm and maxterm representations
- For maxterm representation, select rows with **0's**

x	y	z	G
0	0	0	0 ←
0	0	1	0 ←
0	1	0	0 ←
0	1	1	1
1	0	0	0 ←
1	0	1	0 ←
1	1	0	1
1	1	1	1

$G = xyz + xyz' + x'yz$

↓

$G = m_7 + m_6 + m_3 = \Sigma(3, 6, 7)$

↓

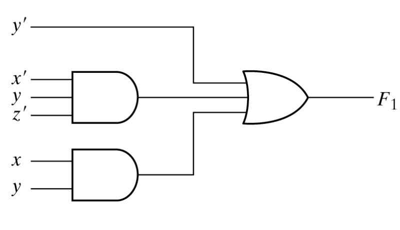
$G = M_0M_1M_2M_4M_5 = \Pi(0,1,2,4,5)$

↓

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

Representation of Circuits

- All logic expressions can be represented in 2-level format
- Circuits can be reduced to minimal 2-level representation
- Sum of products representation most common in industry.



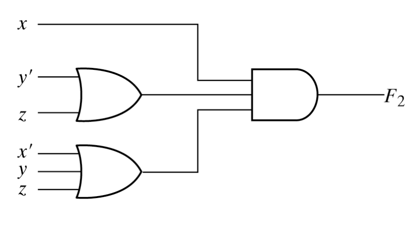


Fig. 2-3 Two-level implementation

Part 2 – Summary

- Truth table, circuit, and boolean expression formats are equivalent
- Easy to translate truth table to SOP and POS representation
- Boolean algebra rules can be used to reduce circuit size while maintaining function
- All logic functions can be made from AND, OR, and NOT
- Easiest way to understand: Do examples!
- Next time: More logic gates!

Part 3 - Overview

- More 2-input logic gates (NAND, NOR, XOR)
- Extensions to 3-input gates
- Converting between sum-of-products and NANDs
  - SOP to NANDs
  - NANDs to SOP
- Converting between sum-of-products and NORs
  - SOP to NORs
  - NORs to SOP
- Positive and negative logic
  - We use primarily positive logic in this course.

Logic functions of N variables

- Each truth table represents one possible function (e.g. AND, OR)
- If there are N inputs, there are 2^{2^N}
- For example, is N is 2 then there are 16 possible truth tables.
- So far, we have defined 2 of these functions
  - 14 more are possible.
- Why consider new functions?
  - Cheaper hardware, more flexibility.

x	y	G
0	0	0
0	1	0
1	0	0
1	1	1

The NAND Gate

- This is a NAND gate. It is a combination of an AND gate followed by an inverter. Its truth table shows this…
- NAND gates have several interesting properties…
  - NAND(a,a)=(aa)' = a' = NOT(a)
  - NAND'(a,b)=(ab)'' = ab = AND(a,b)
  - NAND(a',b')=(a'b')' = a+b = OR(a,b)

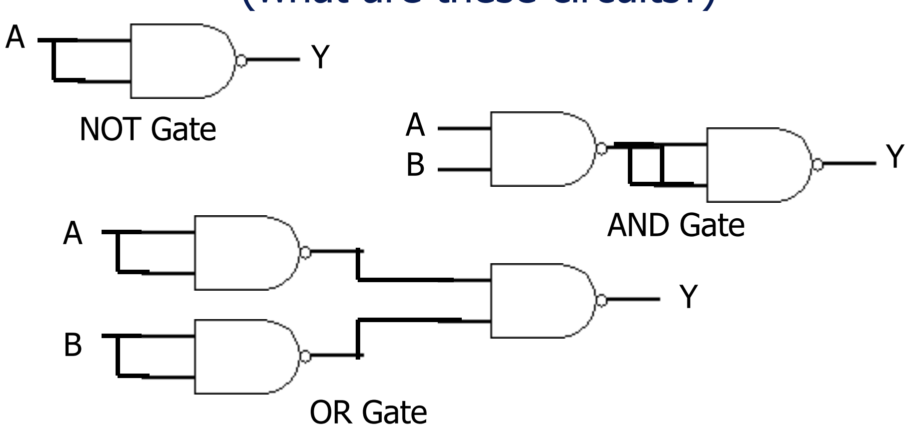
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

The NAND Gate

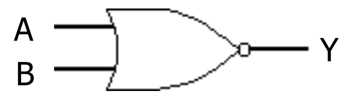
- **These three properties show that a NAND gate with both of its inputs driven by the same signal is equivalent to a NOT gate**
- **A NAND gate whose output is complemented is equivalent to an AND gate, and a NAND gate with complemented inputs acts as an OR gate.**
- **Therefore, we can use a NAND gate to implement all three of the *elementary operators* (AND,OR,NOT).**
- **Therefore,** ANY switching function can be constructed using only NAND gates**. Such a gate is said to be *primitive* or *functionally complete*.**

NAND Gates into Other Gates

(what are these circuits?)



The NOR Gate



- This is a NOR gate. It is a combination of an OR gate followed by an inverter. It's truth table shows this…
- NOR gates also have several interesting properties…
  - NOR(a,a)=(a+a)' = a' = NOT(a)
  - NOR'(a,b)=(a+b)'' = a+b = OR(a,b)
  - NOR(a',b')=(a'+b')' = ab = AND(a,b)

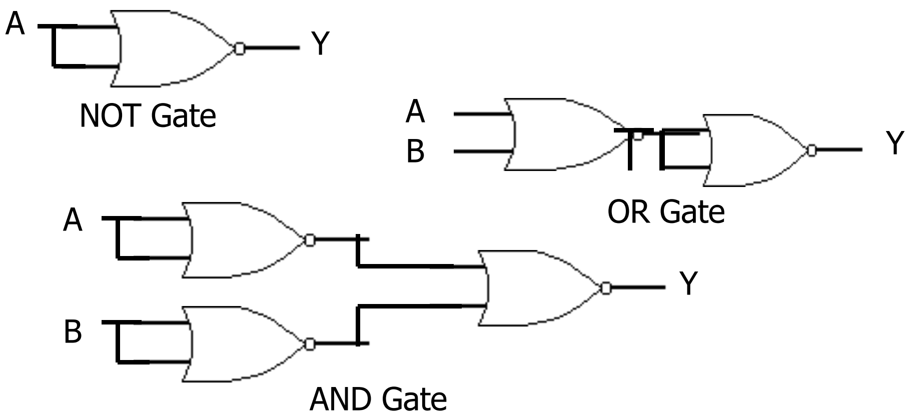
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Functionally Complete Gates

- Just like the NAND gate, the NOR gate is functionally complete…any logic function can be implemented using just NOR gates.
- Both NAND and NOR gates are very valuable as any design can be realized using either one.
- It is easier to build an IC chip using all NAND or NOR gates than to combine AND,OR, and NOT gates.
- NAND/NOR gates are typically faster at switching and cheaper to produce.

NOR Gates into Other Gates

(what are these circuits?)

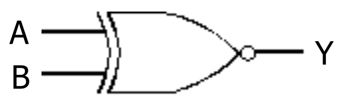


The XOR Gate (Exclusive-OR)

- **This is a XOR gate.**
- **XOR gates assert their output when exactly one of the inputs is asserted, hence the name.**
- **The switching algebra symbol for this operation is ⊕, i.e. 1 ⊕ 1 = 0 and 1 ⊕ 0 = 1.**

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

The XNOR Gate

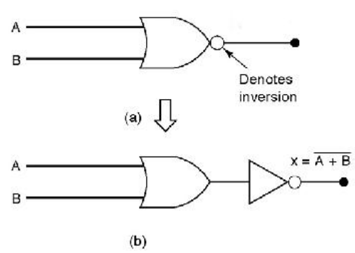


- This is a XNOR gate.
- This functions as an exclusive-NOR gate, or simply the complement of the XOR gate.
- The switching algebra symbol for this operation is ⊙, i.e. 1 ⊙ 1 = 1 and 1 ⊙ 0 = 0.

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

NOR Gate Equivalence

- NOR Symbol, Equivalent Circuit, Truth Table



		OR	NOR
A	B	$A + B$	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

(c)

DeMorgan's Theorem

- A key theorem in simplifying Boolean algebra expression is DeMorgan's Theorem. It states:

  (a + b)' = a'b' (ab)' = a' + b'

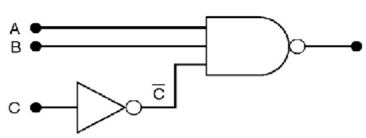
- **Complement the expression**

  a(b + z(x + a')) and simplify.

$$
\begin{aligned}
(a(b+z(x+a')))' &= a' + (b + z(x + a'))' \\
&= a' + b'(z(x + a'))' \\
&= a' + b'(z' + (x + a')') \\
&= a' + b'(z' + x'a'') \\
&= a' + b'(z' + x'a)
\end{aligned}
$$

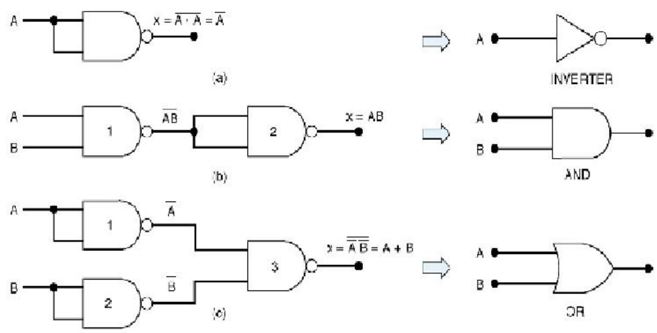
Example

- Determine the output expression for the below circuit and simplify it using DeMorgan's Theorem

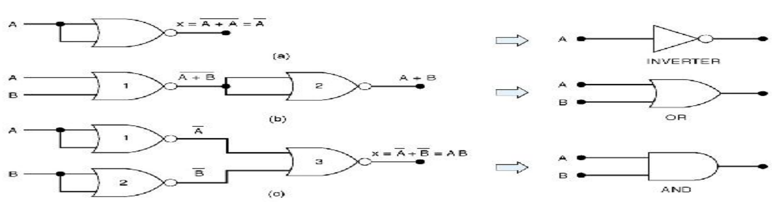


$z = \overline{A \cdot B \cdot \overline{C}} = \overline{A} + \overline{B} + \overline{\overline{C}} = \overline{A} + \overline{B} + C$

Universality of NAND and NOR gates



Universality of NOR gate



- **Equivalent representations of the AND, OR, and NOT gates**

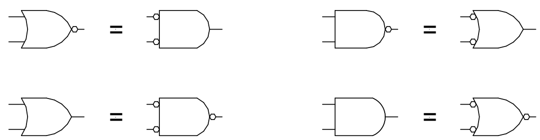
Combinational logic circuits

- The output is depend on the present inputs
- Design steps
  - Express the problem in English
  - Identify the inputs and outputs
  - Draw the truth table to satisfy the conditions of the problem
  - Build the circuit using basic logic gates
  - Convert to NAND only or NOR only if required

Problem 1

- Design a combinational circuit to give output 1 when the main electricity supply is not available in the night.
- Inputs
  - Main Supply Available Y=1, N=0
  - Night Y=1, N=0
- Output
  - 1 when the above requirements are satisfied

NAND-NAND & NOR-NOR Networks



push bubbles **or** ***introduce in pairs*** **or** ***remove pairs.***

Conversion Between Forms

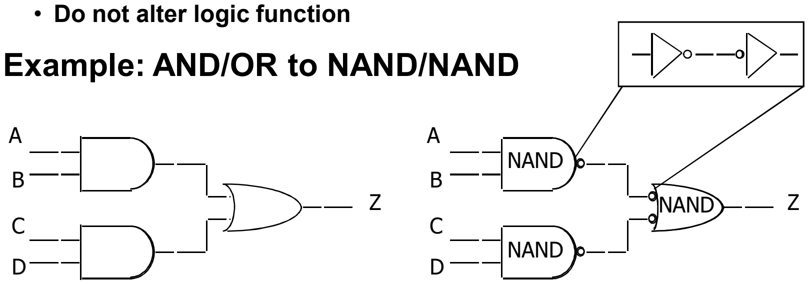
° **Convert from networks of ANDs and ORs to networks of NANDs and NORs**

- **Introduce appropriate inversions ("bubbles")**

° **Each introduced "bubble" must be matched by a corresponding "bubble"**

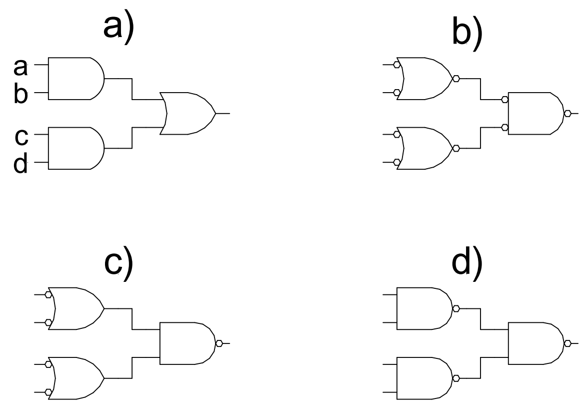
- **Conservation of inversions**
- **Do not alter logic function**

° **Example: AND/OR to NAND/NAND**

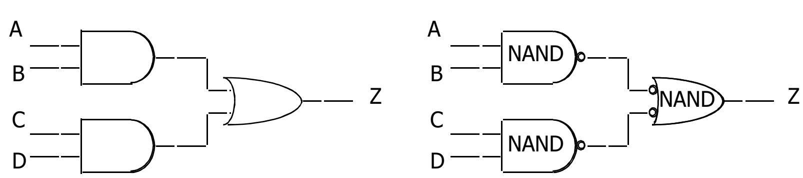


NAND-NAND Networks

° **Mapping from AND/OR to NAND/NAND**

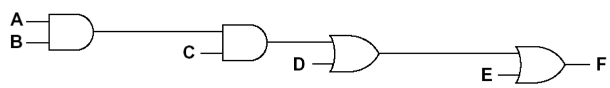


Conversion Between Forms (cont'd)

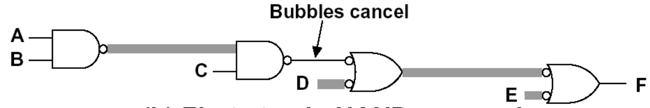


Conversion to NAND Gates

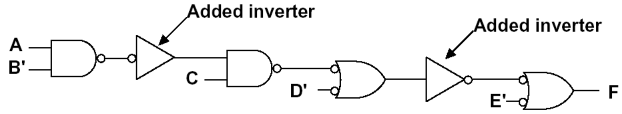
° **Find network of OR and AND gates**



(a) AND_OR network

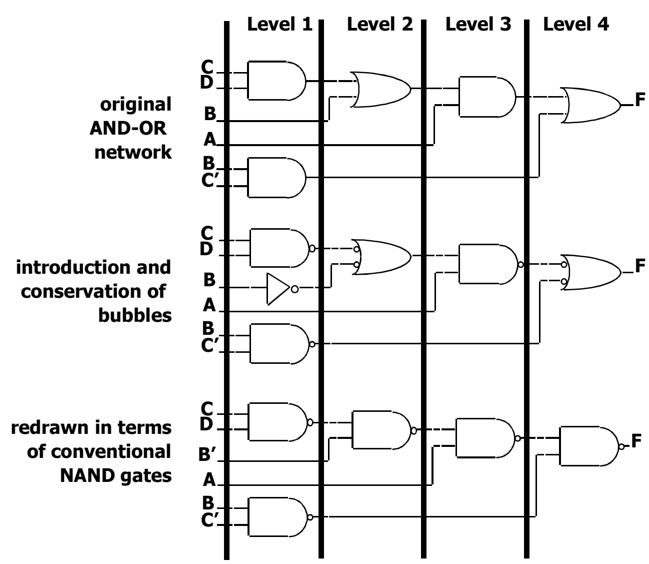


(b) First step in NAND conversion



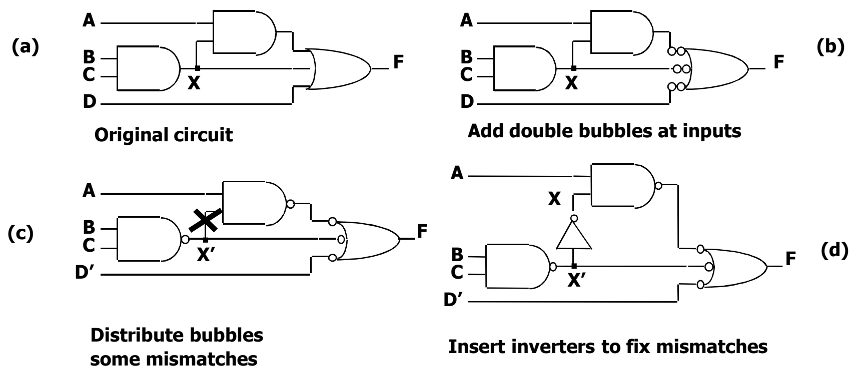
(c) Completed conversion

Conversion of Multi-level Logic to NAND Gates



Conversion Between Forms

- Example



Part 3 - Summary

- Basic logic functions can be made from NAND, and NOR functions
- The behavior of digital circuits can be represented with waveforms, truth tables, or symbols
- Primitive gates can be combined to form larger circuits
- Boolean algebra defines how binary variables with NAND, NOR can be combined
- DeMorgan’s rules are important.
  - Allow conversion to NAND/NOR representations