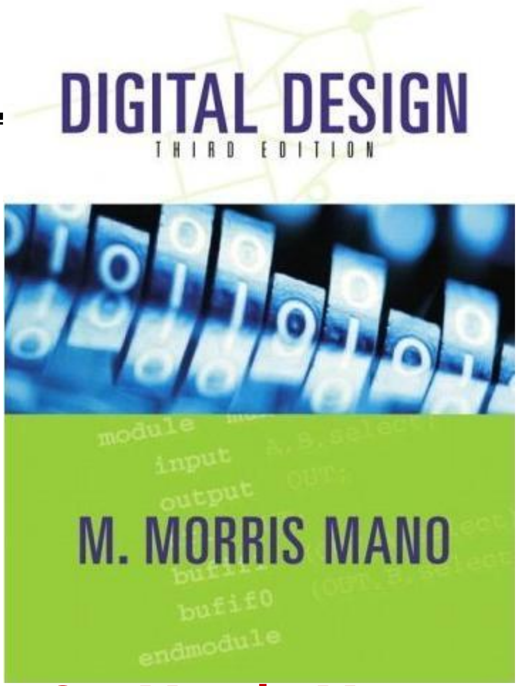


Digital Design 3e, Morris Mano

Chapter 2 – Boolean Algebra and Logic Gates

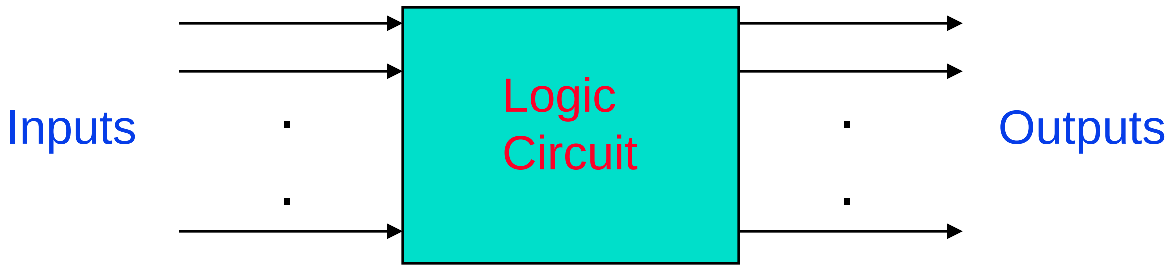
BOOLEAN ALGEBRA

Part 1 - 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**

Digital Systems

- **Analysis problem:**

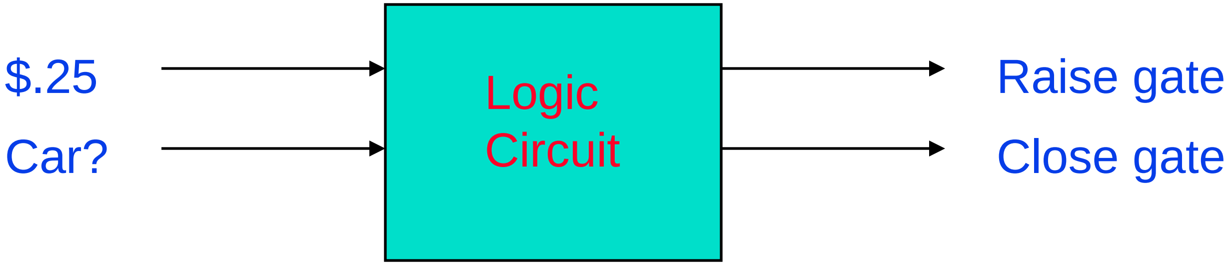


  - **Determine binary outputs for each combination of inputs**

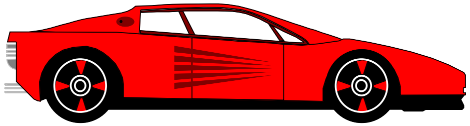
- **Design problem: given a task, develop a circuit that accomplishes the task**
  - **Many possible implementation**
  - **Try to develop “best” circuit based on some criterion (size, power, performance, etc.)**

Toll Booth Controller

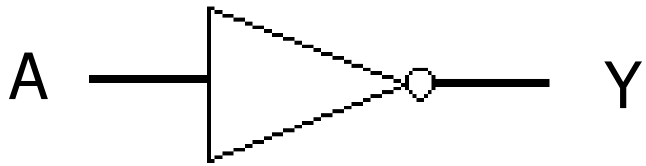
- Consider the design of a toll booth controller
- Inputs: quarter, car sensor
- Outputs: gate lift signal, gate close signal



- If driver pitches in quarter, raise gate.
- When car has cleared gate, close gate.



Describing Circuit Functionality: Inverter



Symbol

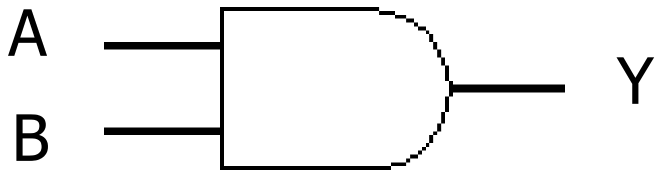
Truth Table

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

Input Output

- **Basic logic functions have symbols.**
- **The same functionality can be represented with truth tables.**
  - **Truth table completely specifies outputs for all input combinations.**
- **The above circuit is an inverter.**
  - **An input of 0 is inverted to a 1.**
  - **An input of 1 is inverted to a 0.**

The AND Gate

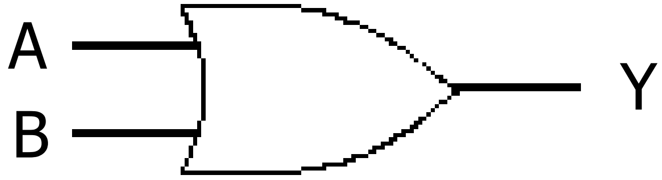


- This is an AND gate.
- So, if the two inputs signals are asserted (high) the output will also be asserted. Otherwise, the output will be deasserted (low).

Truth Table

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

The OR Gate



- **This is an OR gate.**
- **So, if either of the two input signals are asserted, or both of them are, the output will be asserted.**

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

Describing Circuit Functionality: Waveforms

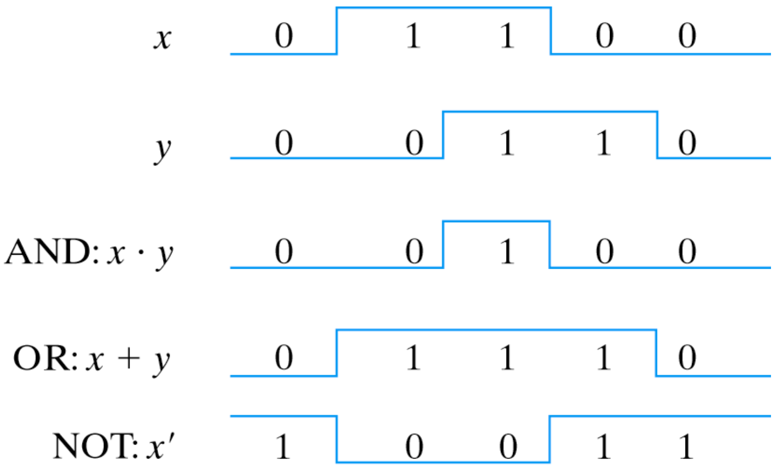


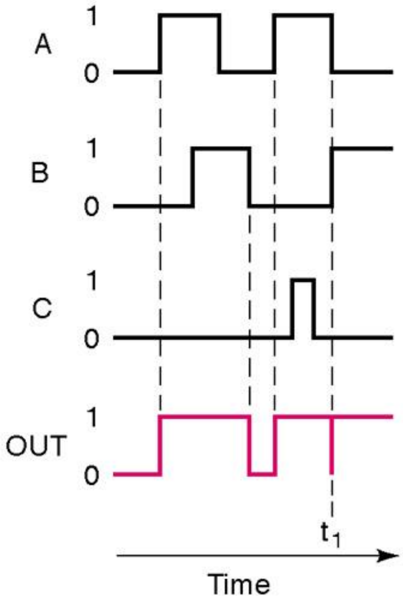
Fig. 1-5 Input-output signals for gates

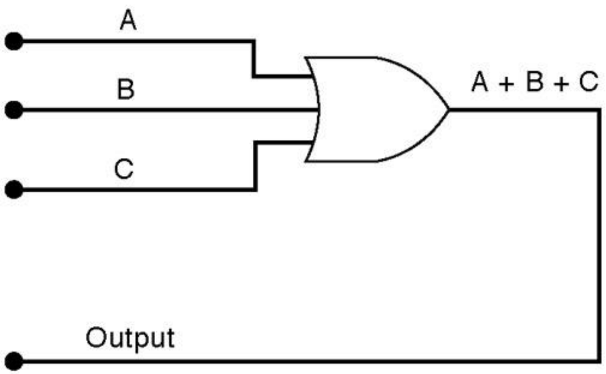
AND Gate

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

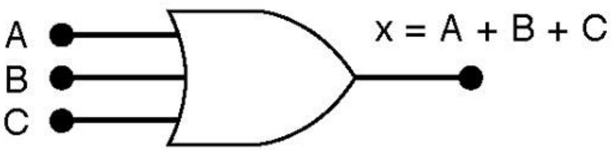
- **Waveforms provide another approach for representing functionality.**
- **Values are either high (logic 1) or low (logic 0).**
- **Can you create a truth table from the waveforms?**

Consider three-input gates





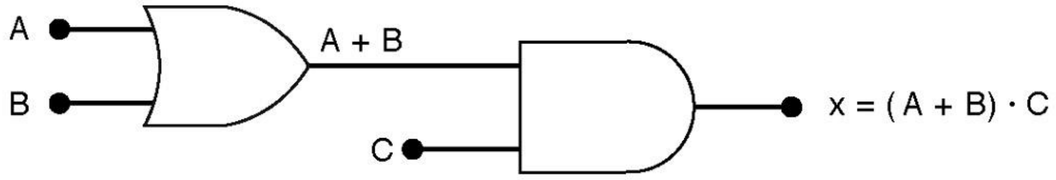
3 Input OR Gate



A	B	C	x = A + B + C
0	0	0	0
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	1

Ordering Boolean Functions

- How to interpret A•B+C?
  - Is it A•B ORed with C ?
  - Is it A ANDed with B+C ?
- Order of precedence for Boolean algebra: AND before OR.
- Note that parentheses are needed here :



Boolean Algebra

- A *Boolean algebra* is defined as a closed algebraic system containing a set K or two or more elements and the two operators, . and +.
- Useful for identifying and *minimizing* circuit functionality
- Identity elements
  - a + 0 = a
  - a . 1 = a
- 0 is the identity element for the + operation.
- 1 is the identity element for the . operation.

Commutativity and Associativity of the Operators

- **The** Commutative Property:

  For every a and b in K,

  - **a + b = b + a**
  - **a . b = b . a**

- **The** Associative Property:

  For every a, b, and c in K,

  - **a + (b + c) = (a + b) + c**
  - **a . (b . c) = (a . b) . c**

Distributivity of the Operators and Complements

- **The** Distributive Property:

  For every a, b, and c in K,

  - **a + (b . c) = (a + b) . (a + c)**
  - **a . (b + c) = (a . b) + (a . c)**

- **The Existence of the** Complement**:**

  For every a in K there exists a unique element called a’ (*complement of a*) such that,

  - **a + a’ = 1**
  - **a . a’ = 0**

- **To simplify notation, the . operator is frequently omitted. When two elements are written next to each other, the AND (.) operator is implied…**

  - **a + b . c = (a + b) . (a + c)**
  - **a + bc = (a + b)(a + c)**

Duality

- **The principle of *duality* is an important concept. This says that if an expression is valid in Boolean algebra, the dual of that expression is also valid.**
- **To form the dual of an expression, replace all + operators with . operators, all . operators with + operators, all ones with zeros, and all zeros with ones.**
- **Form the dual of the expression**

  $a + (bc) = (a + b)(a + c)$

- **Following the replacement rules…**

  $a(b + c) = ab + ac$

- **Take care not to alter the location of the parentheses if they are present.**

Involution

- **This theorem states:**

  $a'' = a$

- **Remember that $aa' = 0$ and $a+a'=1$.**
  - **Therefore, a' is the complement of a and a is also the complement of a'.**
  - **As the complement of a' is unique, it follows that $a''=a$.**
- **Taking the double inverse of a value will give the initial value.**

Absorption

- **This theorem states:**

  $a + ab = a$ $\qquad\qquad$ $a(a+b) = a$

- **To prove the first half of this theorem:**

$$
\begin{aligned}
\mathbf{a + ab} &\mathbf{= a \,.\, 1 + ab} \\
&\mathbf{= a\,(1 + b)} \\
&\mathbf{= a\,(b + 1)} \\
&\mathbf{= a\,(1)}
\end{aligned}
$$

$$\mathbf{a + ab = a}$$

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.

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

Part 1 - Summary

- Basic logic functions can be made from AND, OR, and NOT (invert) 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 can be combined
- Rules for associativity, commutativity, and distribution are similar to algebra
- DeMorgan's rules are important.
  - Will allow us to reduce circuit sizes.

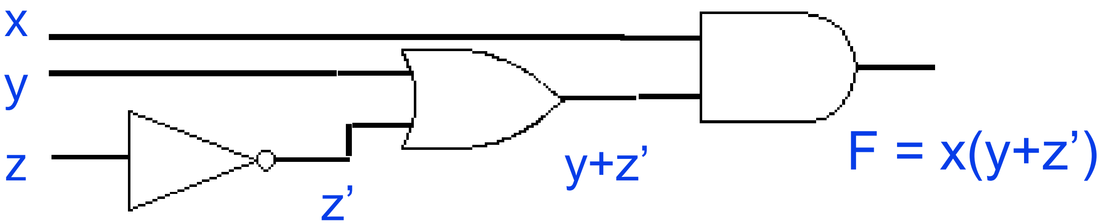
Part 2 - Overview

- 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

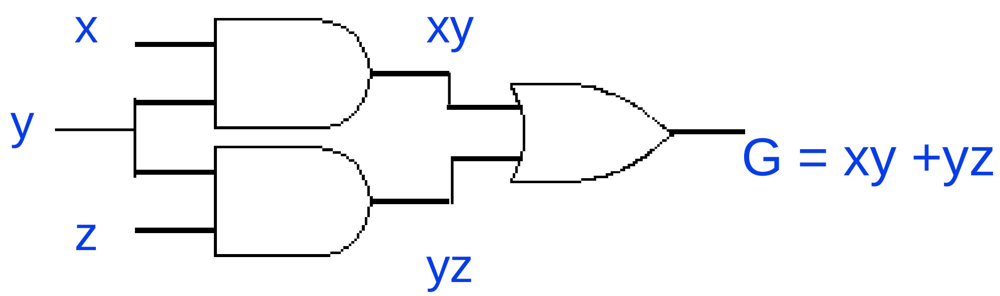


F = x(y+z')

Boolean Functions

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

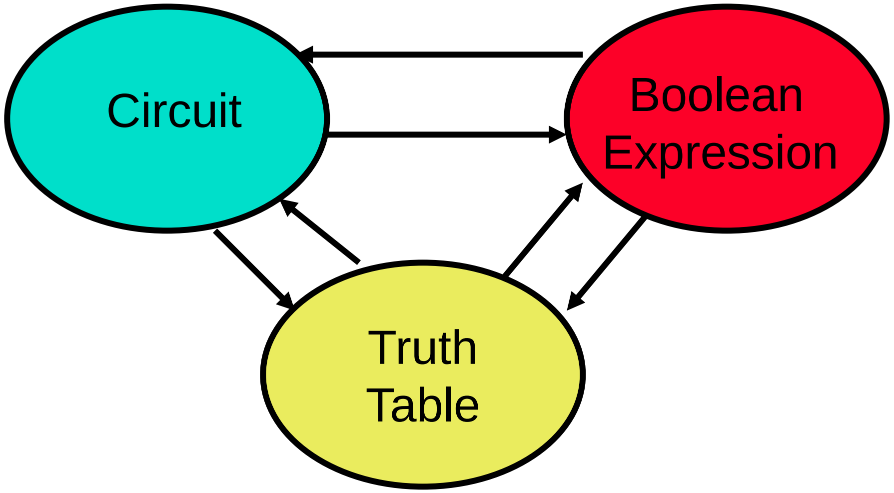
x	y	z	xy	yz	G
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	1	1



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

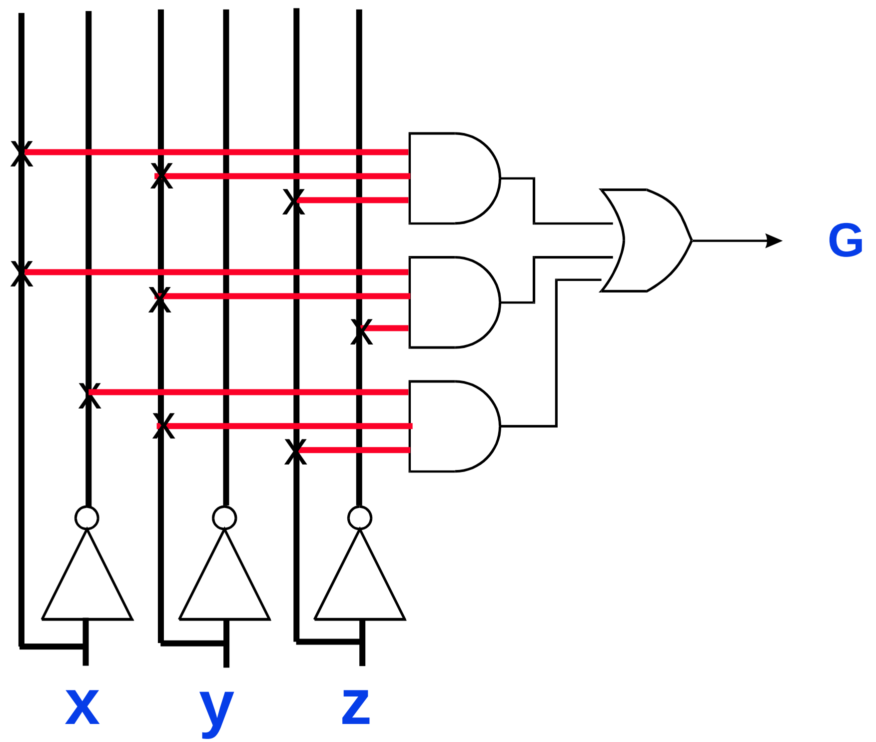
Any Boolean Expression can be represented in sum of products form!

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

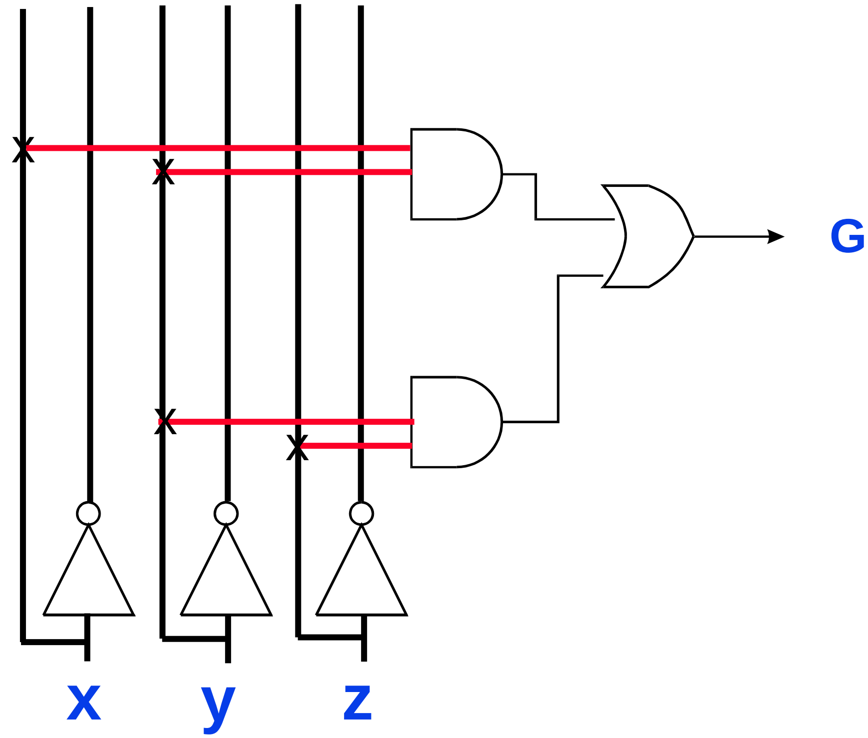
Reducing Boolean Expressions

- **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 Theorum 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 <u>minterm</u>
- Each OR combination of terms is a <u>maxterm</u>

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

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

$$\downarrow$$

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

Complementing Functions

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

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

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

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

Can we find a simpler representation?

Complementing Functions

- Step 1: assign temporary names
  - $b + c \rightarrow z$
  - $(a + z)' = G'$

$$G = a + b + c$$

$$G' = (a + b + c)'$$

- Step 2: Use DeMorgans' Law
  - $(a + z)' = a' . z'$
- Step 3: Resubstitute (b+c) for z
  - $a' . z' = a' . (b + c)'$
- Step 4: Use DeMorgans' Law
  - $a' . (b + c)' = a' . (b'. c')$

$$G = a + b + c$$

$$G' = a' . b' . c' = a'b'c'$$

- Step 5: Associative rule
  - $a' . (b'. c') = a' . b' . c'$

Complementation Example

- Find complement of $F = x'z + yz$
  - $F' = (x'z + yz)'$
- DeMorgan's
  - $F' = (x'z)' (yz)'$
- DeMorgan's
  - $F' = (x''+z')(y'+z')$
- Reduction -> eliminate double negation on x
  - $F' = (x+z')(y'+z')$

This format is called product of sums

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

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

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

$$G = M_0 M_1 M_2 M_4 M_5 = \Pi(0,1,2,4,5)$$

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

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.**

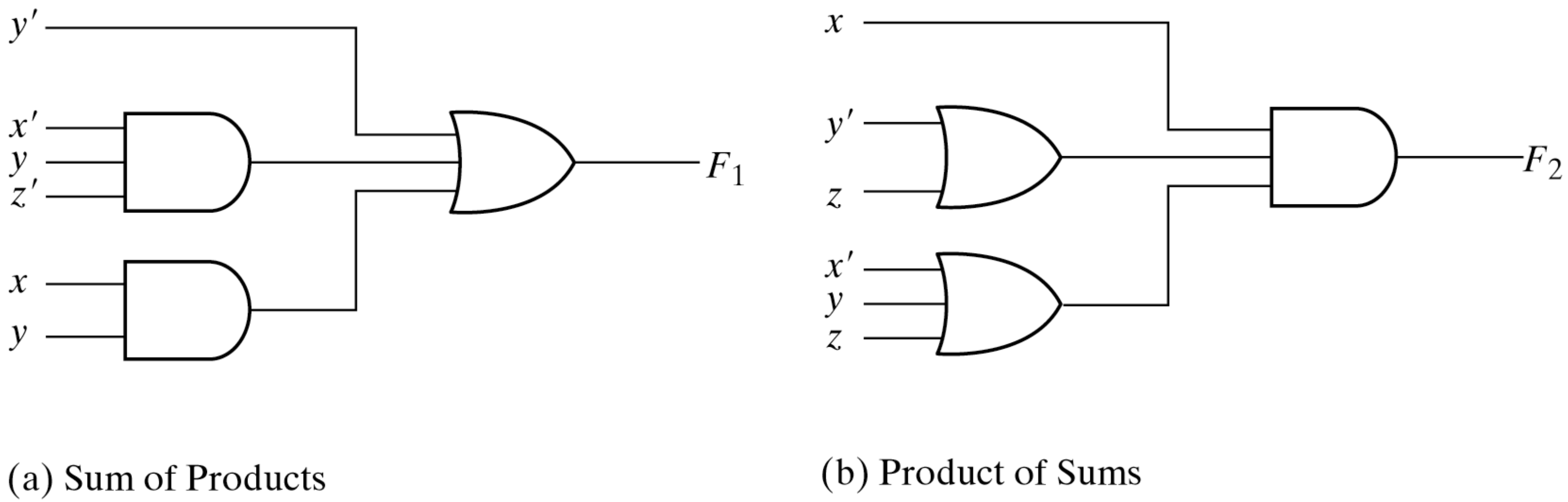


Fig. 2-3 Two-level implementation

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

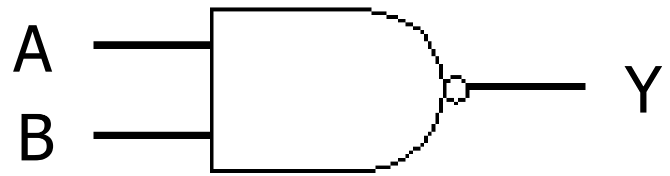
- More 2-input logic gates (NAND, NOR, XOR)
- Extensions to 3-input gates
- Converting between sum-of-products and NANDs
  - SOP to NANDs
  - NANDs to SOP
- Converting between sum-of-products and NORs
  - SOP to NORs
  - NORs to SOP
- Positive and negative logic
  - We use primarily positive logic in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

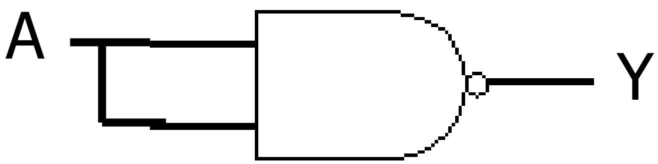
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

The NAND Gate

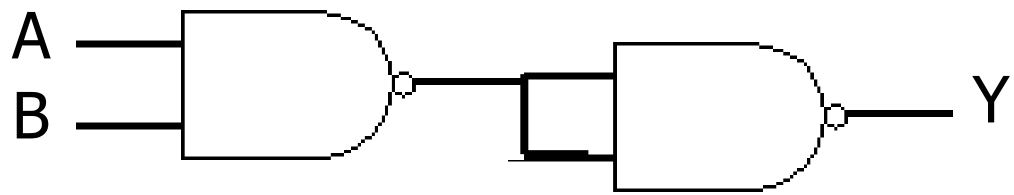
- **These three properties show that a NAND gate with both of its inputs driven by the same signal is equivalent to a NOT gate**
- **A NAND gate whose output is complemented is equivalent to an AND gate, and a NAND gate with complemented inputs acts as an OR gate.**
- **Therefore, we can use a NAND gate 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

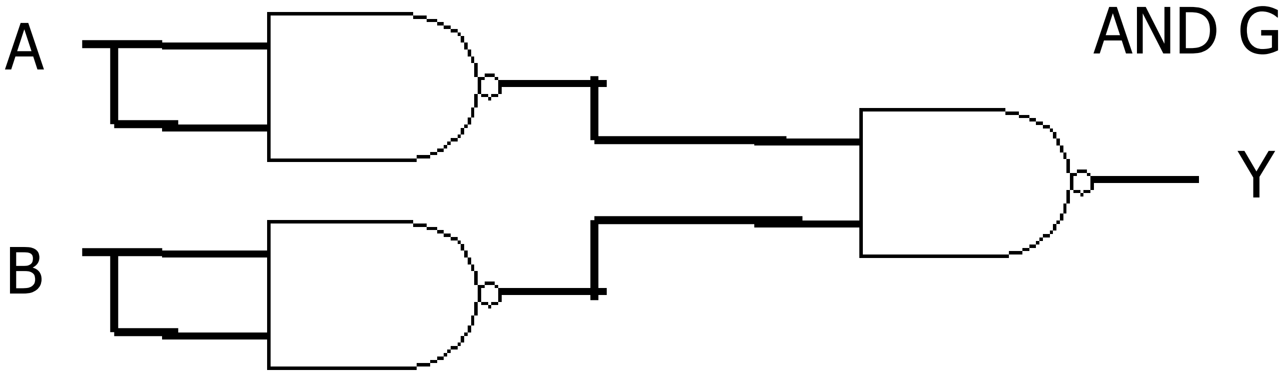
(what are these circuits?)



NOT Gate

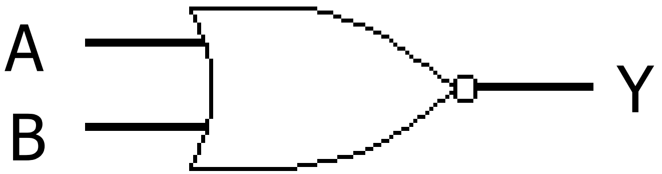


AND Gate



OR Gate

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)

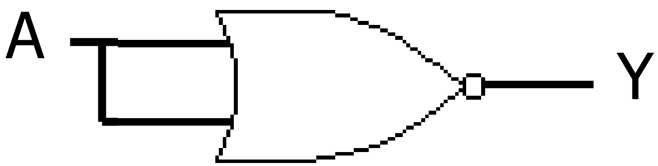
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Functionally Complete Gates

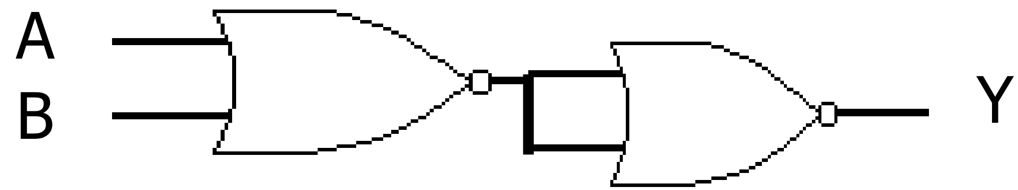
- Just like the NAND gate, the NOR gate is functionally complete…any logic function can be implemented using just NOR gates.
- Both NAND and NOR gates are very valuable as any design can be realized using either one.
- It is easier to build an IC chip using all NAND or NOR gates than to combine AND,OR,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

NOR Gates into Other Gates

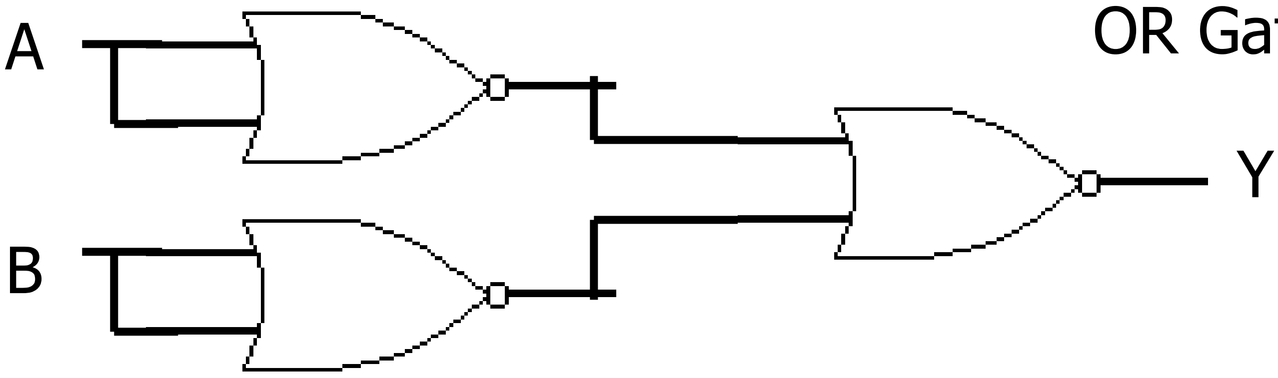
(what are these circuits?)



NOT Gate

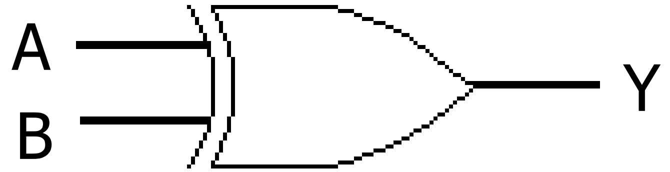


OR Gate



AND Gate

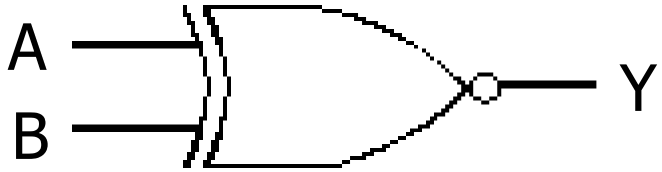
The XOR Gate (Exclusive-OR)



- **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. $1 \oplus 1 = 0$ and $1 \oplus 0 = 1$.**

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

The XNOR Gate

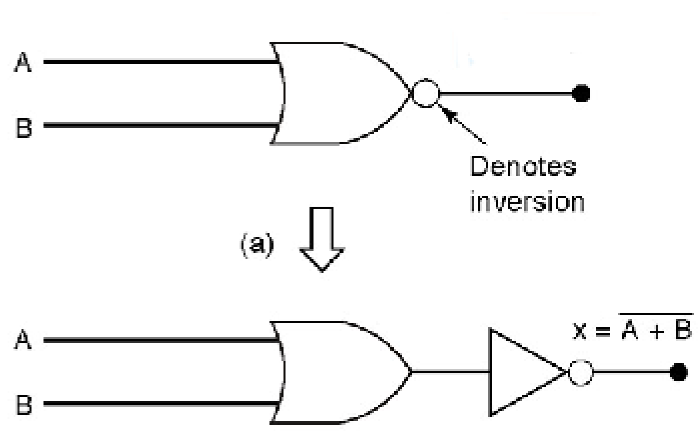


- **This is a XNOR gate.**
- **This functions as an exclusive-NOR gate, or simply the complement of the XOR gate.**
- **The switching algebra symbol for this operation is ⊙, i.e. 1 ⊙ 1 = 1 and 1 ⊙ 0 = 0.**

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

NOR Gate Equivalence

- **NOR Symbol, Equivalent Circuit, Truth Table**



(b)

		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:**

  $(a + b)' = a'b'$ $\qquad$ $(ab)' = a' + b'$

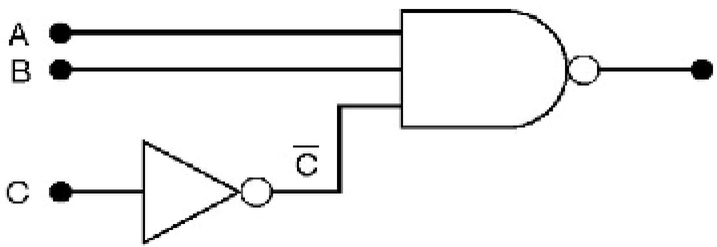
- **Complement the expression**

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

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

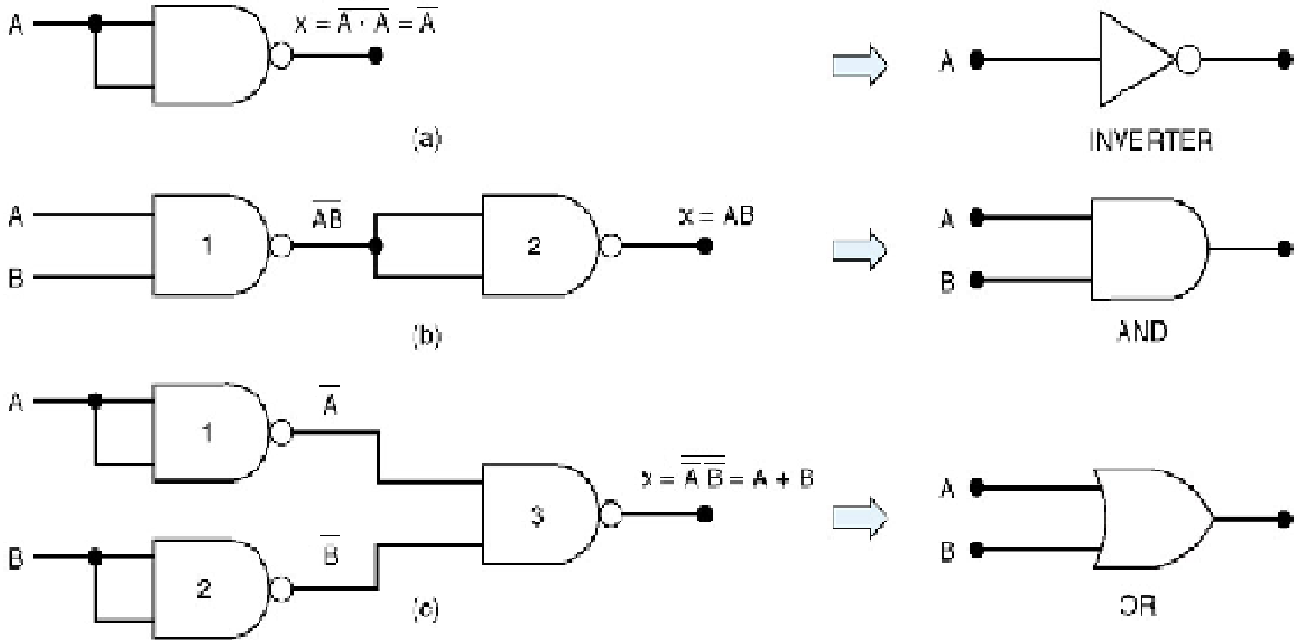
Example

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

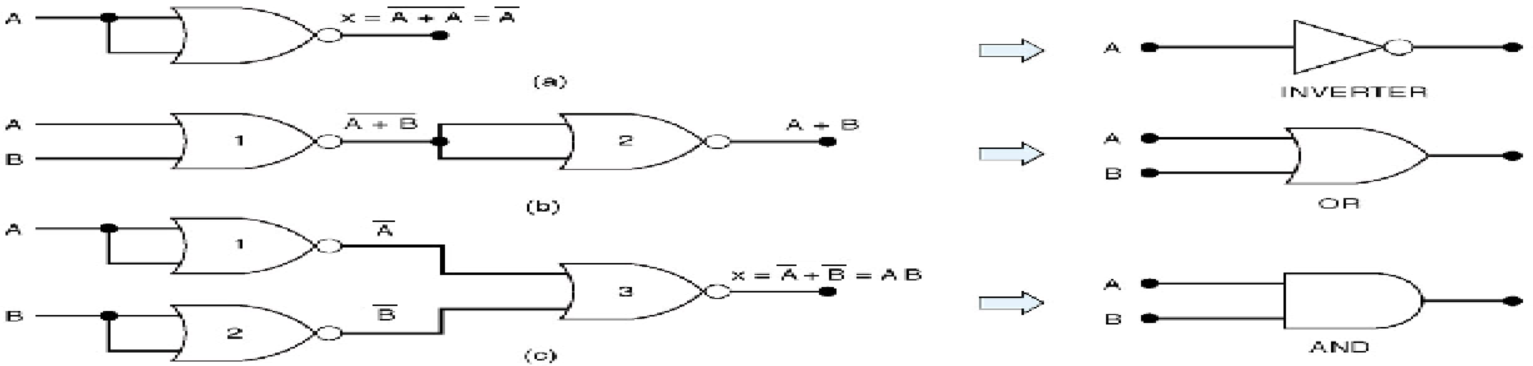


$$z = \overline{A \cdot B \cdot \overline{C}} = \overline{A} + \overline{B} + \overline{\overline{C}} = \overline{A} + \overline{B} + C$$

Universality of NAND and NOR gates



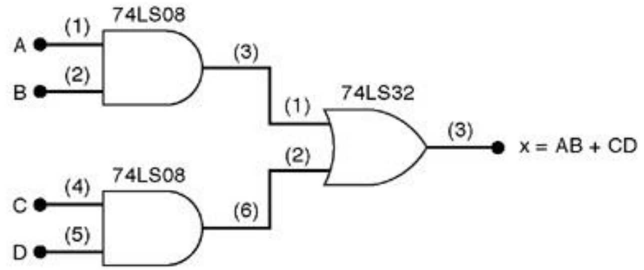
Universality of NOR gate



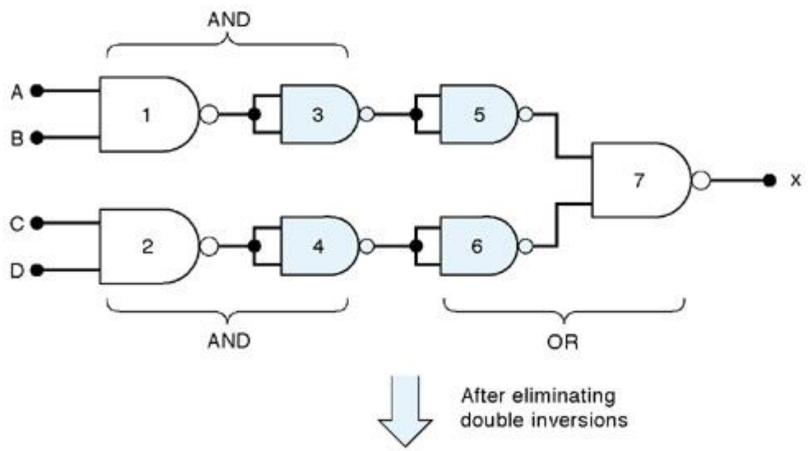
- **Equivalent representations of the AND, OR, and NOT gates**

Example

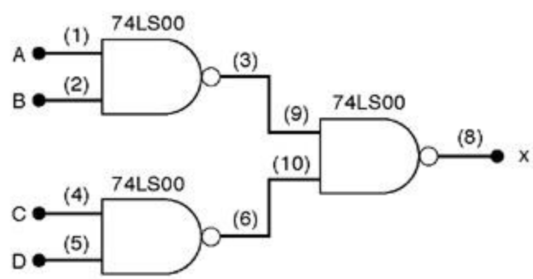
(a)


74LS08
A
(1)
B
(2)
(3)
74LS32
(1)
(2)
(3)
x = AB + CD
74LS08
C
(4)
D
(5)
(6)

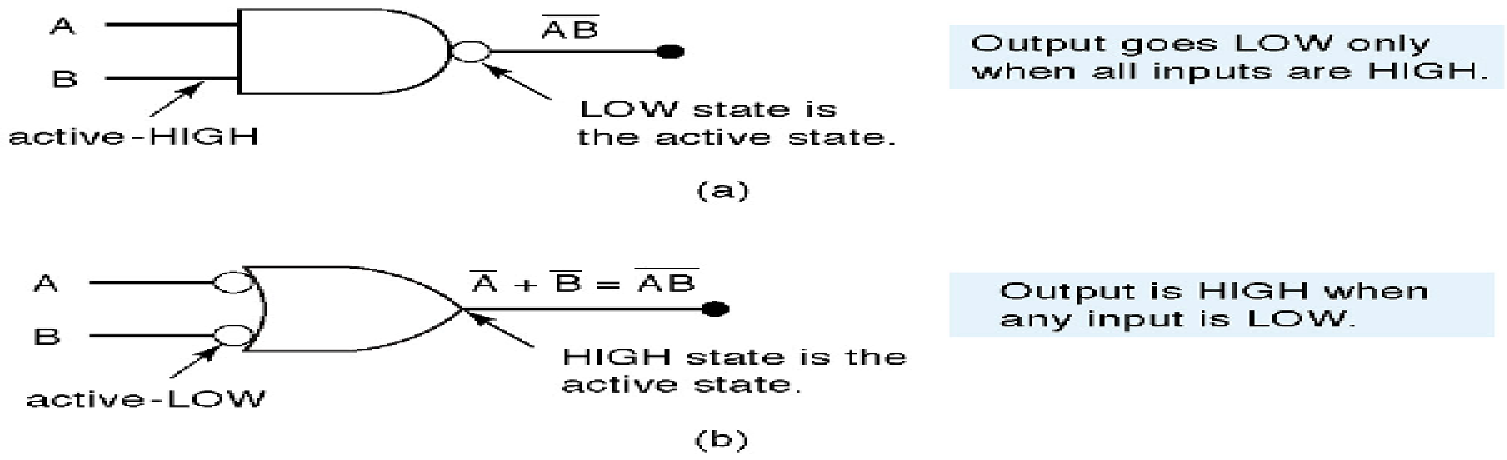
(b)


AND
A
B
1
3
5
7
x
C
D
2
4
6
AND
OR
After eliminating
double inversions

(c)

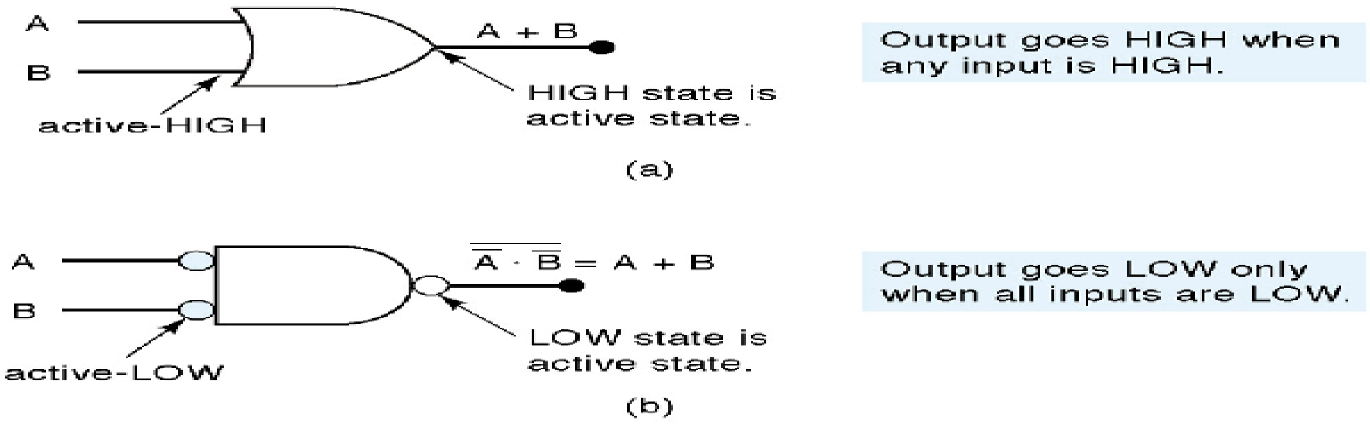

74LS00
A
(1)
B
(2)
(3)
(9)
74LS00
(8)
x
(10)
74LS00
C
(4)
D
(5)
(6)

Interpretation of the two NAND gate symbols



- **Determine the output expression for circuit via DeMorgan's Theorem**

Interpretation of the two OR gate symbols



- **Determine the output expression for circuit via DeMorgan’s Theorem**

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