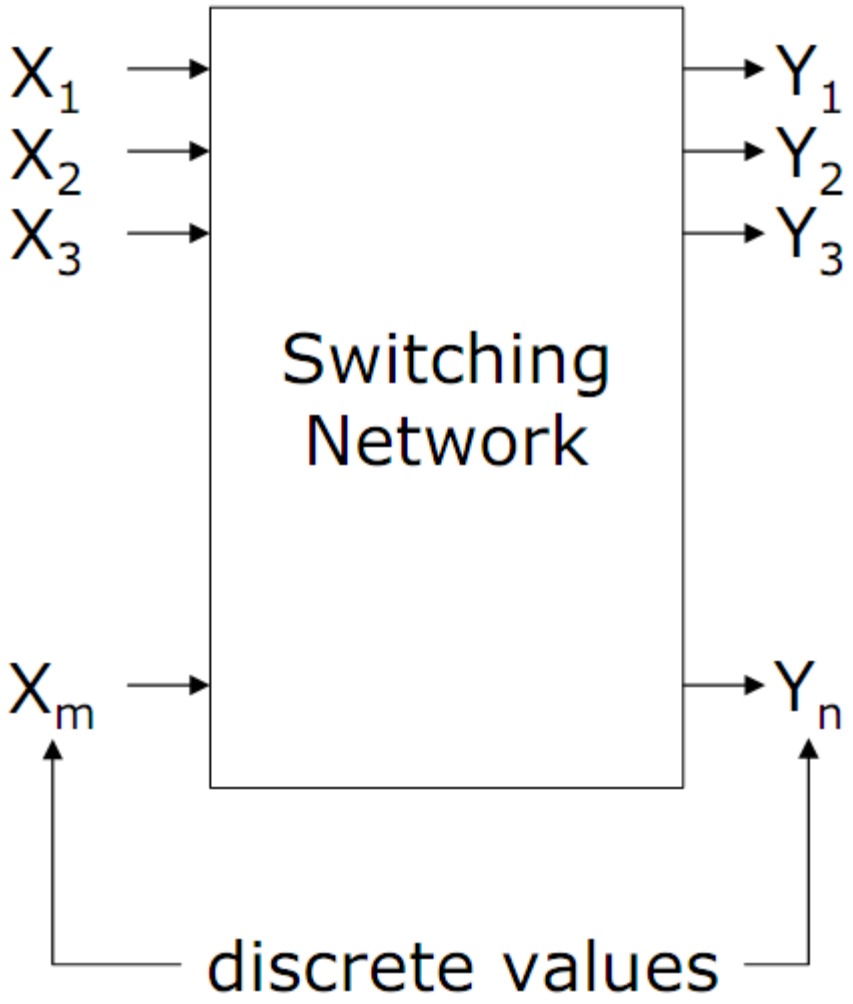
ELECTRICAL
Engineering KMITNB

# Digital Circuits and Logic Design

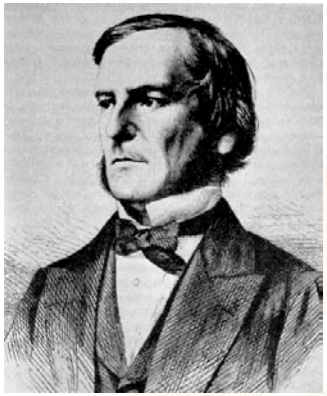
## Lecture2 :Logic Circuits

# Logic Circuits

- Logic circuits perform operations on digital signals
  - Implemented as electronic circuits where signal values are restricted to a few discrete values
- In binary logic circuits there are only two values, 0 and 1
- The general form of a logic circuit is a switching network

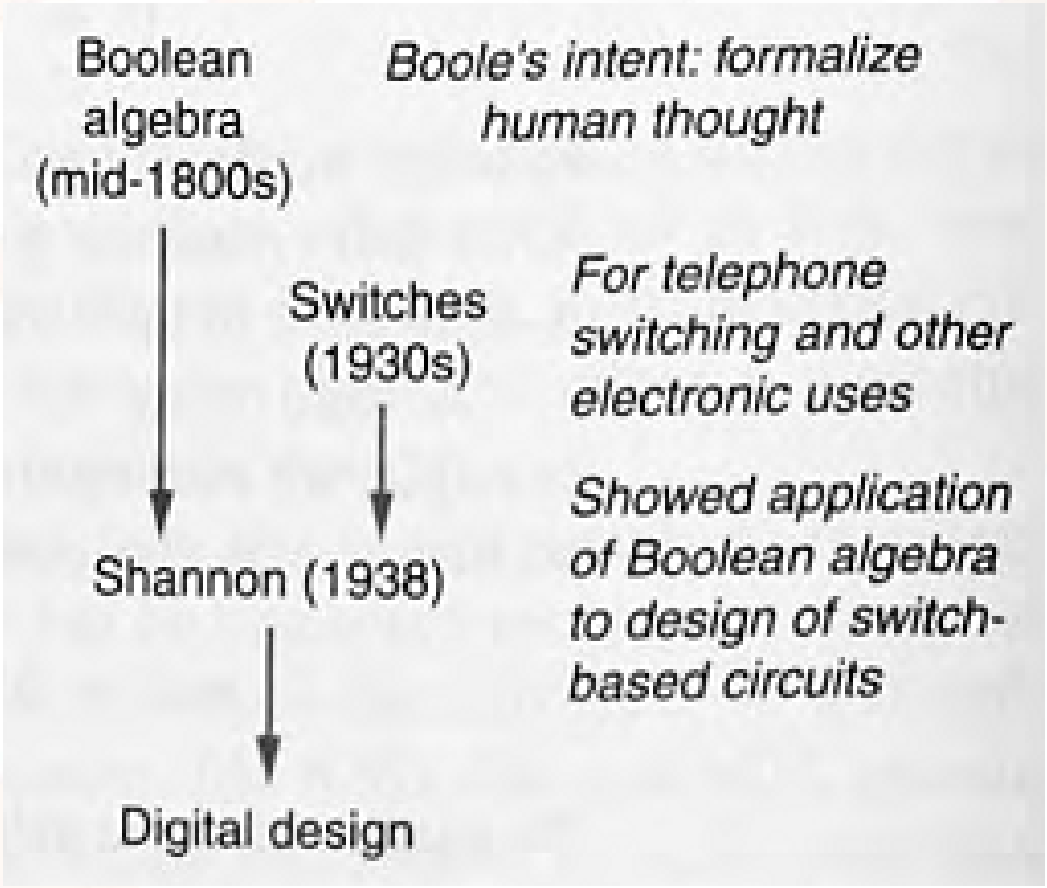


# Boole and Shannon



**George Boole (1815-1864)**

**Claude Elwood Shannon (1916-2001)**



# Boole and Shannon

Boolean logic was developed in the mid-1800s by the mathematician **George Boole**, not to build digital circuits but rather as a scheme for using algebraic methods to formalize human logic and thought

In 1938, an MIT graduate student named **Claude Shannon** wrote a paper (based on his Masters thesis) describing how **Boolean algebra** could be applied to switch-based circuits, by showing that "on" switches could be treated as a 1 (or true). and "off ' switches as a 0 (or false), by connecting those switches in a certain way

**We can build digital circuits by doing math.**

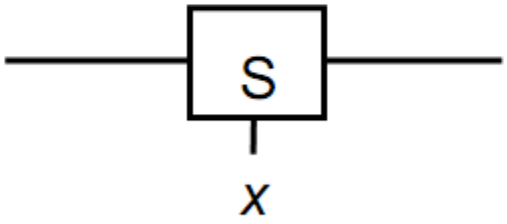
# Boolean Algebra

- Direct application to switching networks
  - Work with 2-state devices => 2-valued Boolean algebra (switching algebra)
  - Use a Boolean variable (X, Y, etc.) to represent an input or output of a switching network
  - Variable may take on only two values (0, 1)
  - X=0, X=1
  - These symbols are not binary numbers, they simply represent the 2 states of a Boolean variable
  - They are not voltage levels, although they commonly refer to the low or high voltage input/output of some circuit element

# Variables and functions

- The simplest binary element is a switch that has two states
- If the switch is controlled by x, we say the switch is open if x = 0 and closed if x = 1

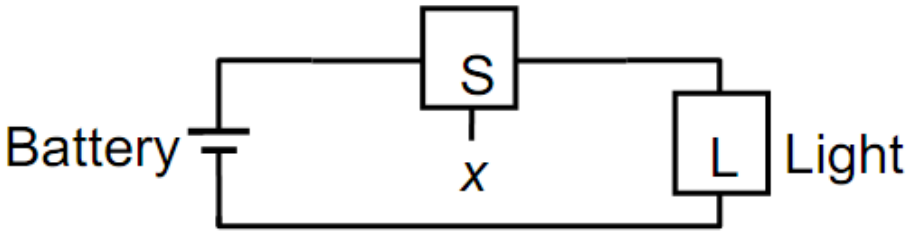

(a) Two states of a switch



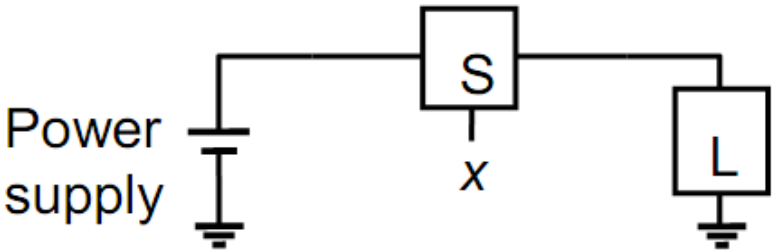

(b) Symbol for a switch

# Variables and functions

- Assume the switch controls a lightbulb as shown
  - The output is defined as the state of the light L
- If the light is on => L=1
- If the light is off => L=0
- The state of L, as function of x is
  - L(x)=x
- L(x) is a logic function
- x is an input variable



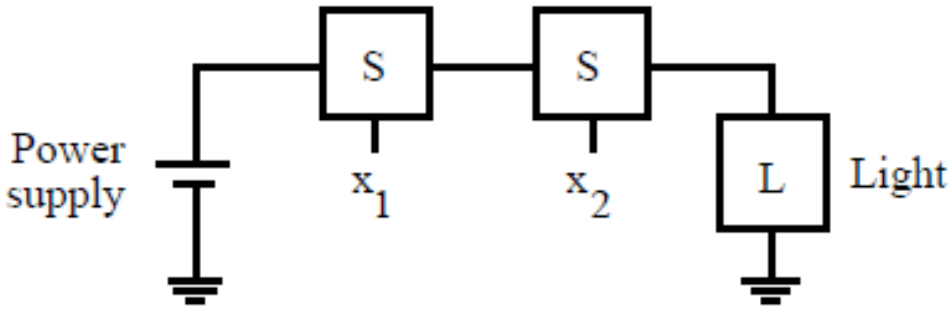

(a) Simple connection to a battery




(b) Using a ground connection as the return path

# Variables and functions (AND)

- Consider the possibility of two switches controlling the state of the light
- Using a series connection, the light will be on only if both switches are closed
  - L(*x1*, *x2*)= *x1*· *x2*
  - L=1 iff (if and only if) *x1 AND x2 are 1*




The logical AND function (series connection)

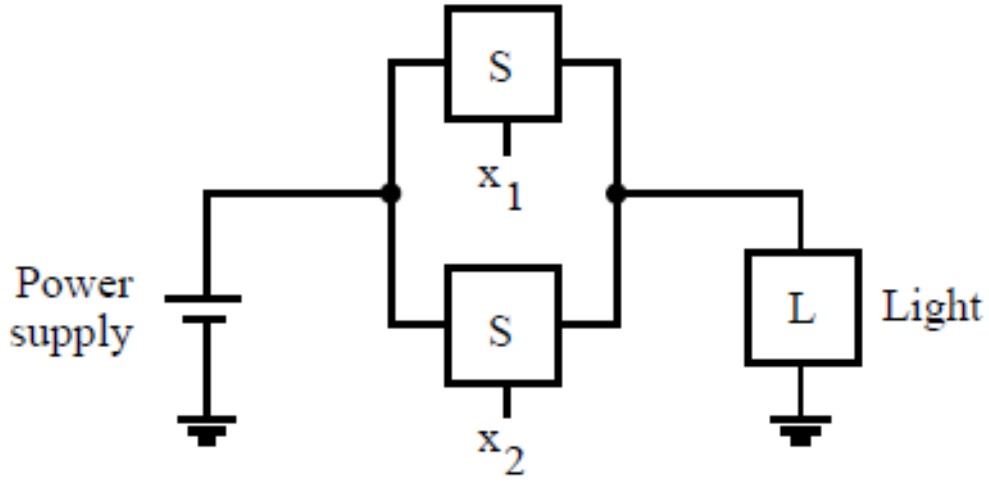
"·" AND operator

$x_1 \cdot x_2 = x_1 x_2$

The circuit implements a logical **AND** function

# Variables and functions (OR)

- Using a parallel connection, the light will be on only if either or both switches are closed
  - L(*x1, x2)= x1+ x2*
  - L=1 if *x1 OR x2 is 1 (or both)*



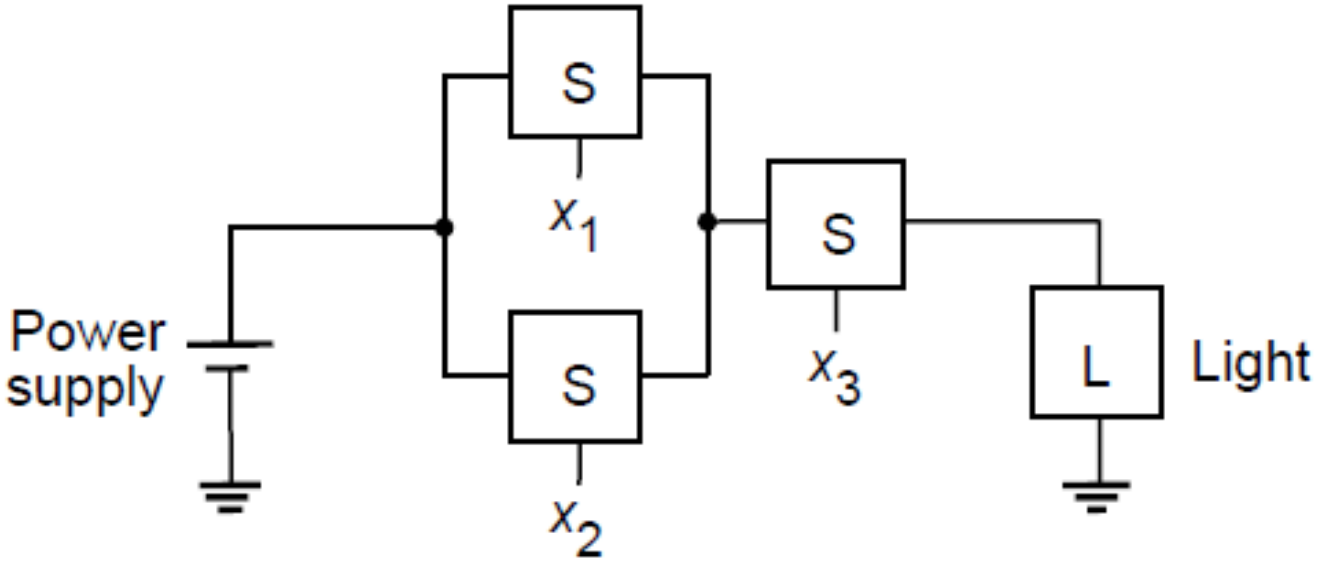

The logical OR function (parallel connection)

"+" OR operator

The circuit implements a logical **OR** function

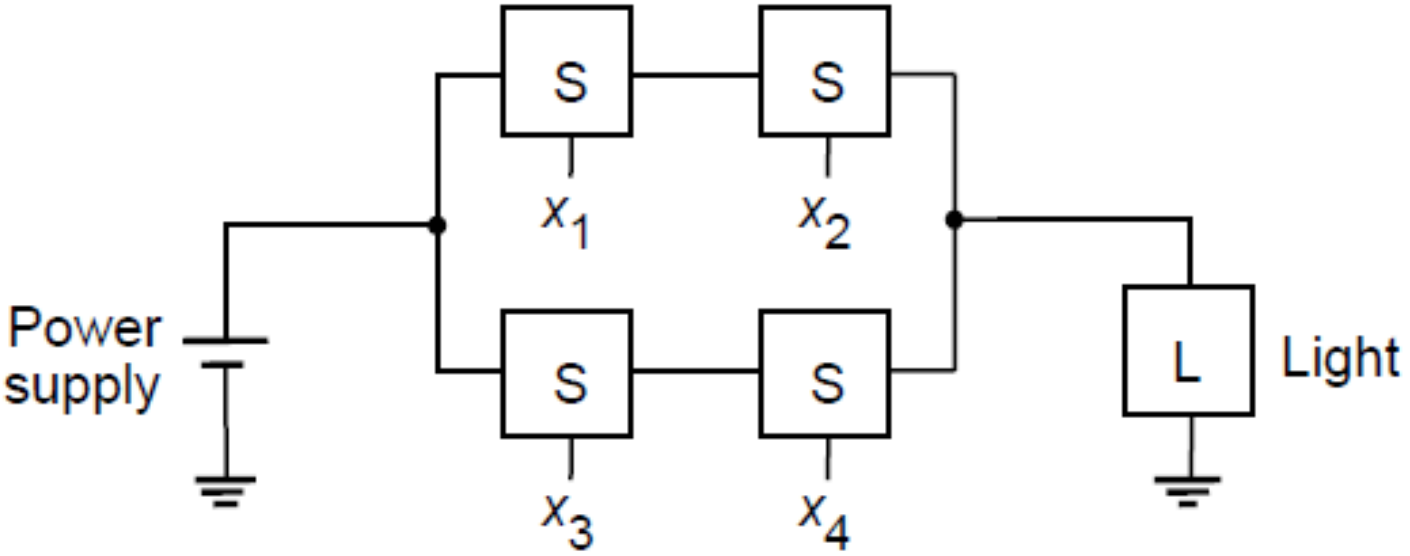
# Variables and functions

- Various series-parallel connections would realize various logic functions
  - $L(x1, x2, x3) = (x1 + x2) \cdot x3$



# Variables and functions

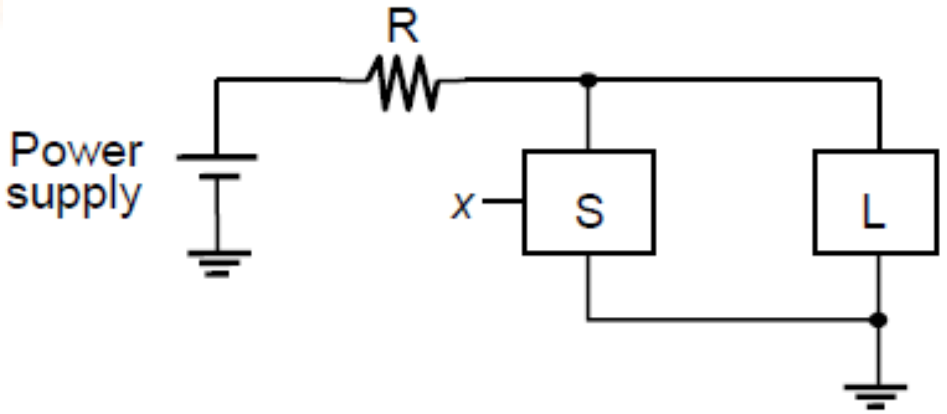
• What would the following logic function look like if implemented via switches?




$L(x1, x2, x3, x4) = ?$

# Inversion

- Before, actions occur when a switch is closed. What about the possibility of an action occurring when a switch is opened?
  - $L(x) = \overline{x}$
  - Where L=1 if *x=0 and L=0 if x=1*
- L(*x*) *is the inverse (or complement) of x*




$\overline{X}, X', \text{NOT } X$

The circuit implements a logical **NOT** function

# Inversion of a function

- If a function is defined as
  - $f(x1, x2) = x1 + x2$
- Then the complement of *f is*
  - $\overline{f}(x1, x2) = \overline{x1 + x2} = (x1 + x2)'$
- Similarily, if
  - $f(x1, x2) = x1 \cdot x2$
- Then the complement of *f is*
  - $\overline{f}(x1, x2) = \overline{x1 \cdot x2} = (x1 \cdot x2)'$

# Truth tables

- Tabular listing that fully describes a logic function
  - Output value for all input combinations (valuations)

| $x_1$ | $x_2$ | $x_1 \cdot x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**AND**

| $x_1$ | $x_2$ | $x_1 + x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**OR**

| $x_1$ | $x_1'$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

**NOT**

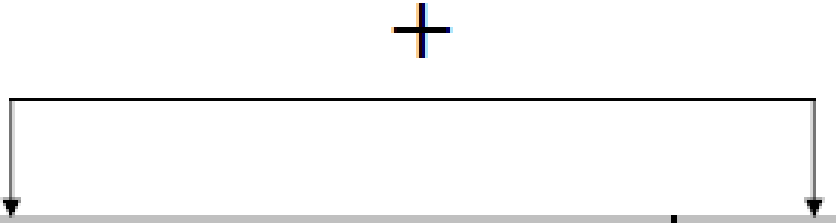
# Truth tables

- Truth table for AND and OR functions of three variables

| $x_1$ | $x_2$ | $x_3$ | $x_1 \cdot x_2 \cdot x_3$ | $x_1 \mid x_2 \mid x_3$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

# Truth tables of functions

- If L(x,y,z)=x+yz, then the truth

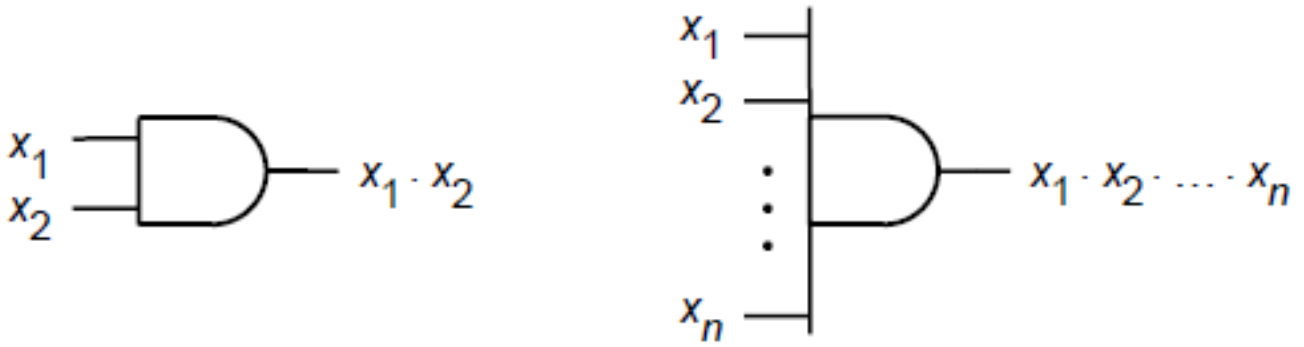


| x | y | z | yz | x+yz |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

# Logic gates and networks

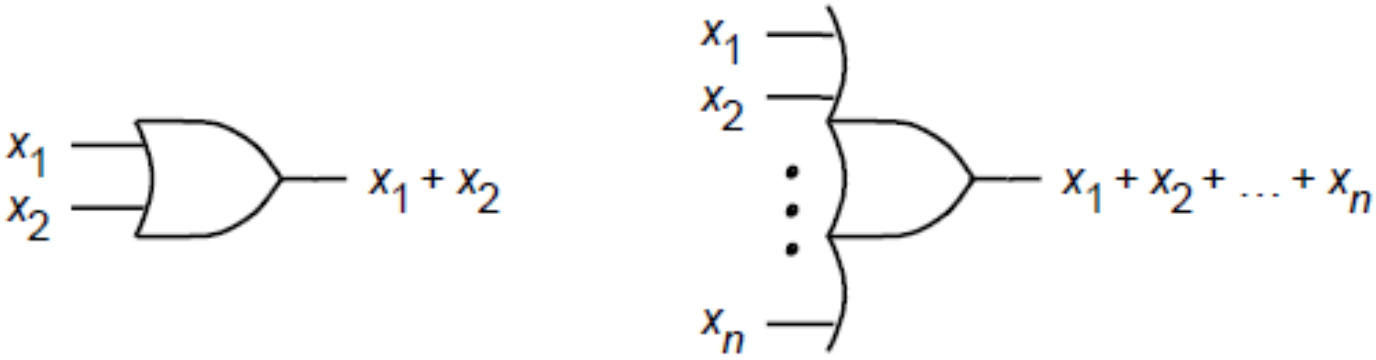
• Each basic logic operation (AND, OR, NOT) can be implemented resulting in a circuit element called a ***logic gate***

• A logic gate has one or more inputs and one output that is a function of its inputs

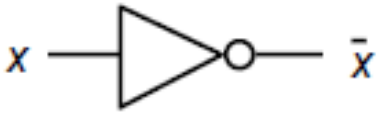



AND gates

# Logic gates and networks



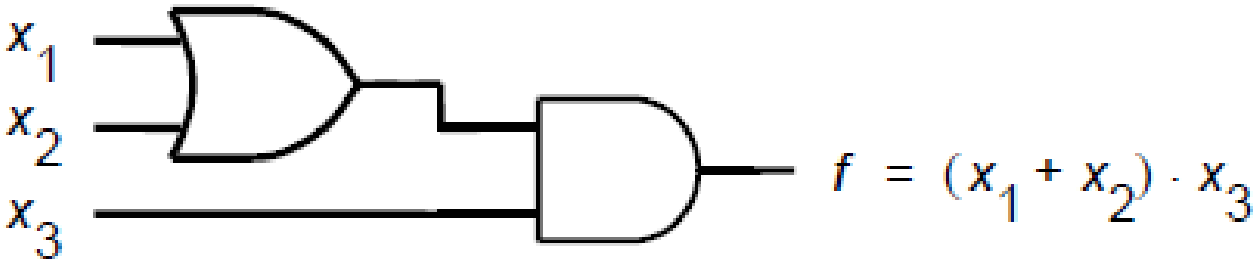

OR gates




NOT gate

# Logic gates and networks

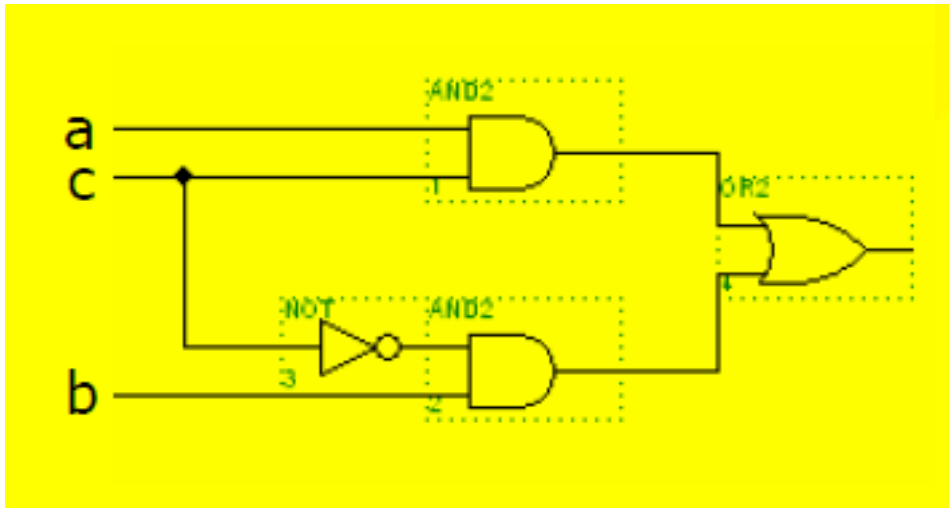
- A larger circuit is implemented by a network of gates
  - Called a logic network or logic circuit



# Logic gates and networks

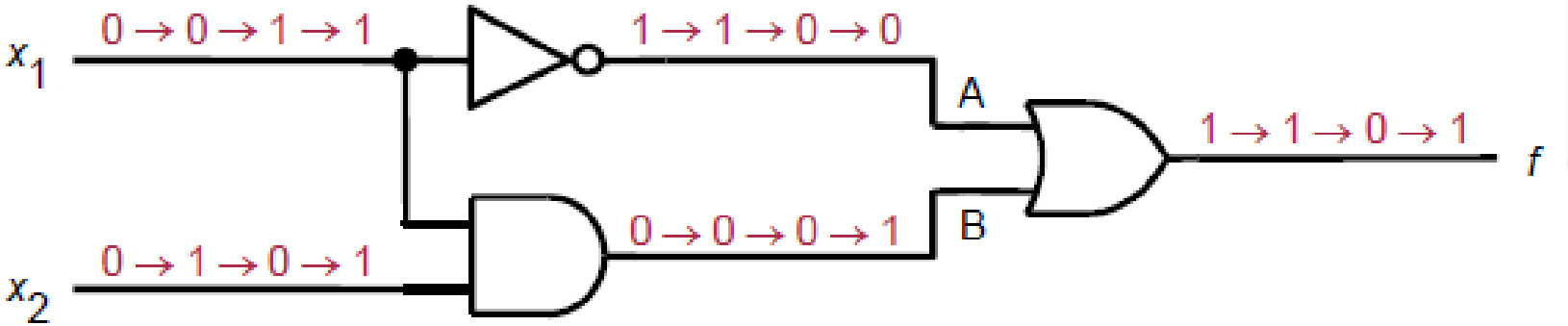
- Draw the truth table and the logic circuit for the following function
  - F(a,b,c) = ac+bc’

| a | b | c | ac | bc' | ac+bc' |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |



# Analysis of a logic network

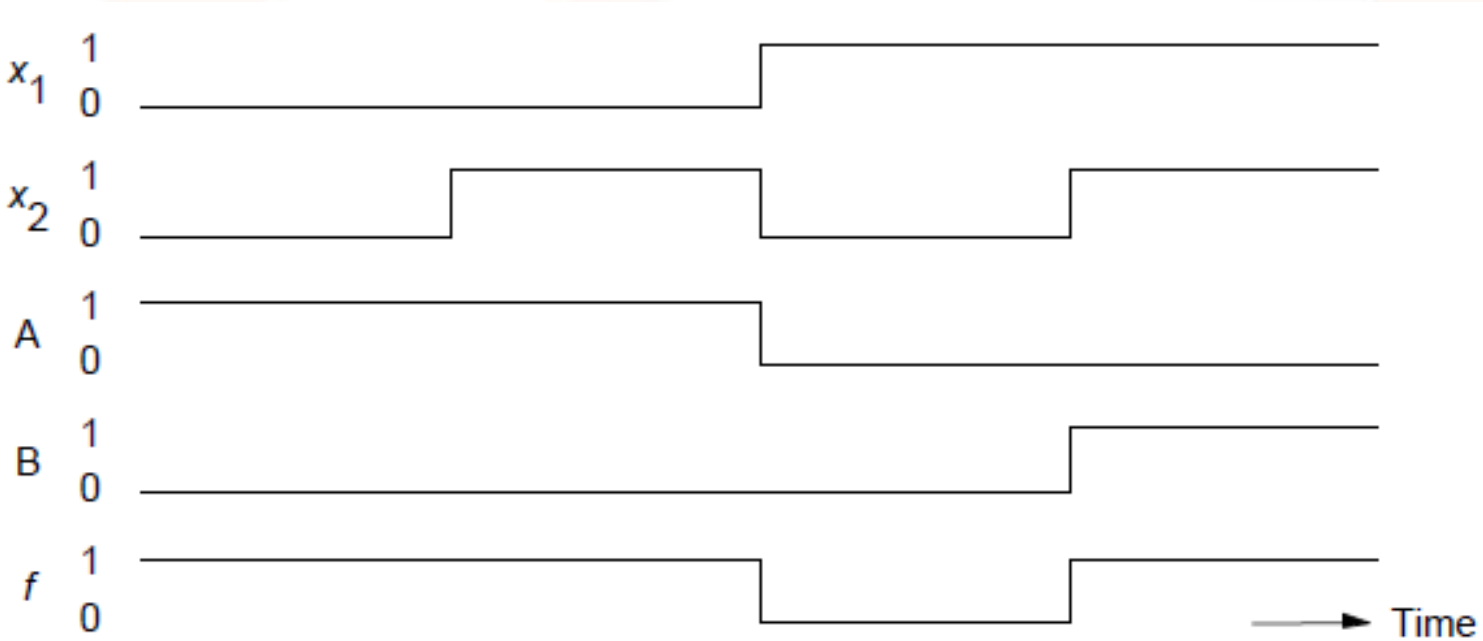
• To determine the functional behavior of a logic network, we can apply all possible input signals to it




Network that implements $f = \bar{x}_1 + x_1 \cdot x_2$

# Analysis of a logic network

• The function of a logic network can also be described by a timing diagram (gives dynamic behavior of the network)




Timing diagram

# Axioms of Boolean algebra

• Boolean algebra: based on a set of rules derived from a small number of basic assumptions (***axioms)***

- 1a $0 \cdot 0 = 0$
- 1b $1 + 1 = 1$
- 2a $1 \cdot 1 = 1$
- 2b $0 + 0 = 0$

- 3a $0 \cdot 1 = 1 \cdot 0 = 0$
- 3b $1 + 0 = 0 + 1 = 1$
- 4a If $x = 0$ then $x' = 1$
- 4b If $x = 1$ then $x' = 0$

# Axioms of Boolean algebra

• From the axioms are derived some rules for dealing with single variables

- 5a $x \cdot 0 = 0$
- 5b $x + 1 = 1$
- 6a $x \cdot 1 = x$
- 6b $x + 0 = x$
- 7a $x \cdot x = x$
- 7b $x + x = x$
- 8a $x \cdot x' = 0$
- 8b $x + x' = 1$
- 9 $x'' = x$

• Single-variable theorems can be proven by perfect induction

• Substitute the values *x=0 and x=1 into the* expressions and verify using the basic axioms

# Duality

- Axioms and single-variable theorems are expressed in pairs
  - Reflects the importance of ***duality***
- Given any logic expression, its dual is formed by replacing all + with · , and vice versa and replacing all 0s with 1s and vice versa
  - $f(a,b)=a+b$ dual of $f(a,b)=a \cdot b$
  - $f(x)=x+0$ dual of $f(x)=x \cdot 1$
- The dual of any true statement is also true

# Two & three variable properties

- 10a. $x \cdot y = y \cdot x$ *Commutative*
- 10b. $x + y = y + x$

- 11a. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ *Associative*
- 11b. $x + (y + z) = (x + y) + z$

- 12a. $x \cdot (y + z) = x \cdot y + x \cdot z$ *Distributive*
- 12b. $x + y \cdot z = (x + y) \cdot (x + z)$

- 13a. $x + x \cdot y = x$ *Absorption*
- 13b. $x \cdot (x + y) = x$

# Two & three variable properties

- 14a. $x \cdot y + x \cdot y' = x$ *Combining*
- 14b. $(x+y) \cdot (x+y') = x$

- 15a. $(x \cdot y)' = x' + y'$ *DeMorgan's*
- 15b. $(x+y)' = x' \cdot y'$ *Theorem*

- 16a. $x + x' \cdot y = x + y$
- 16b. $x \cdot (x' + y) = x \cdot y$

- 17a. $x \cdot y + y \cdot z + x' \cdot z = x \cdot y + x' \cdot z$ *Consensus*
- 17b. $(x+y) \cdot (y+z) \cdot (x'+z) = (x+y) \cdot (x'+z)$

# Proof by Perfect Induction

This means that you must construct a truth table, enumerate all possible input variable combinations and develop outputs in the table for each side of the equalities below and hence prove the equality.

- Use perfect induction to prove

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

# Proof by Perfect Induction

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

| $x$ | $y$ | $x'y$ | $x+x'y$ | $x+y$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |

equivalent

# Proof by Perfect Induction

- Use perfect induction to prove $(xy)'=x'+y'$

| $x$ | $y$ | $xy$ | $(xy)'$ | $x'$ | $y'$ | $x'+y'$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |

equivalent

# Proof (algebraic manipulation)

- Prove
  - $(X+A)(X'+A)(A+C)(A+D)X = AX$
  - $(X+A)(X'+A)(A+C)(A+D)X$
  - $(X+A)(X'+A)(A+CD)X$ (using *12b*)
  - $(X+A)(X'+A)(A+CD)X$
  - $(A)(A+CD)X$ (using *14b*)
  - $(A)(A+CD)X$
  - $AX$ (using *13b*)

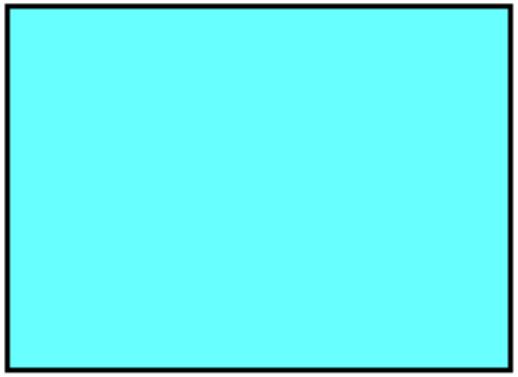
# Algebraic manipulation

- Algebraic manipulation can be used to simplify Boolean expressions
  - *Simpler expression => simpler logic circuit*
- Not practical to deal with complex expressions in this way
- However, the theorems & properties provide the basis for automating the synthesis of logic circuits in CAD tools
  - *To understand the CAD tools the designer should be aware of the fundamental concepts*

# Venn diagrams

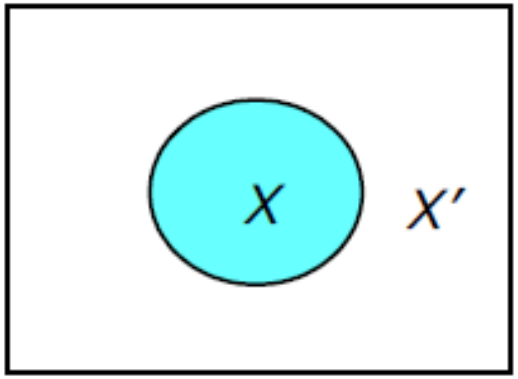
- Venn diagram: graphical illustration of various operations and relations in an algebra of sets
- A set s is a collection of elements that are members of s (for us this would be a collection of Boolean variables and/or constants)
- Elements of the set are represented by the area enclosed by a contour (usually a circle)

# Venn diagrams

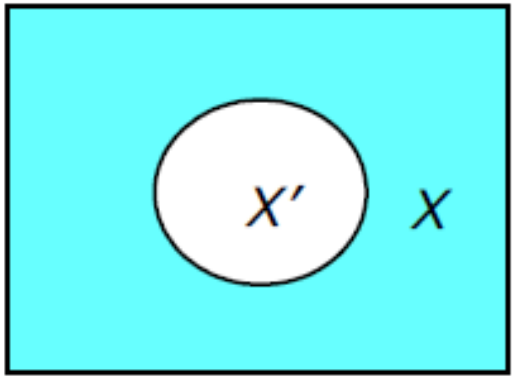


(a) Constant 1

(b) Constant 0

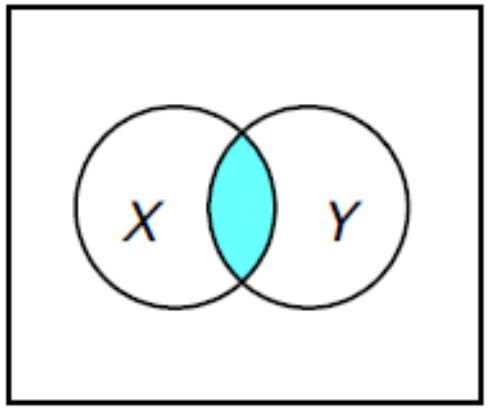



(c) Variable $X$

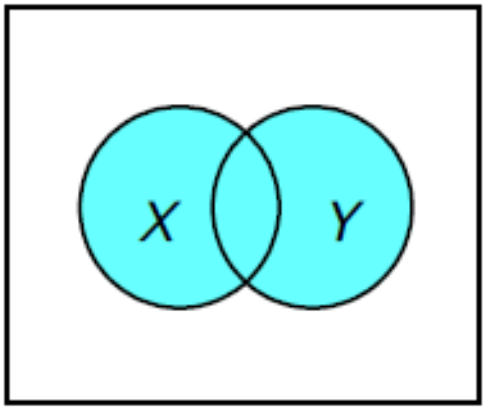



(d) $X'$

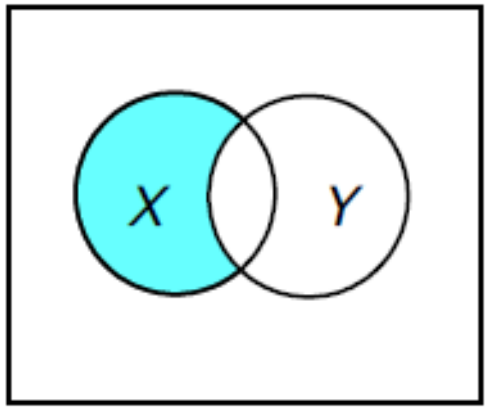
# Venn diagrams



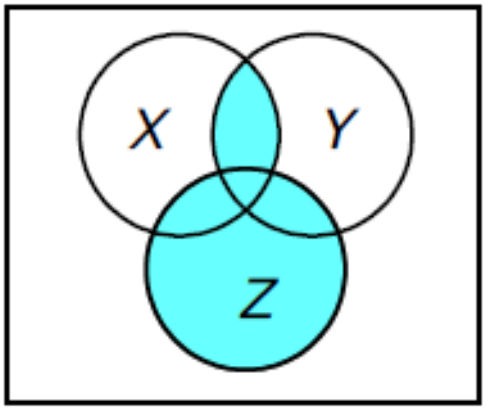

(e) $XY$




(f) $X+Y$

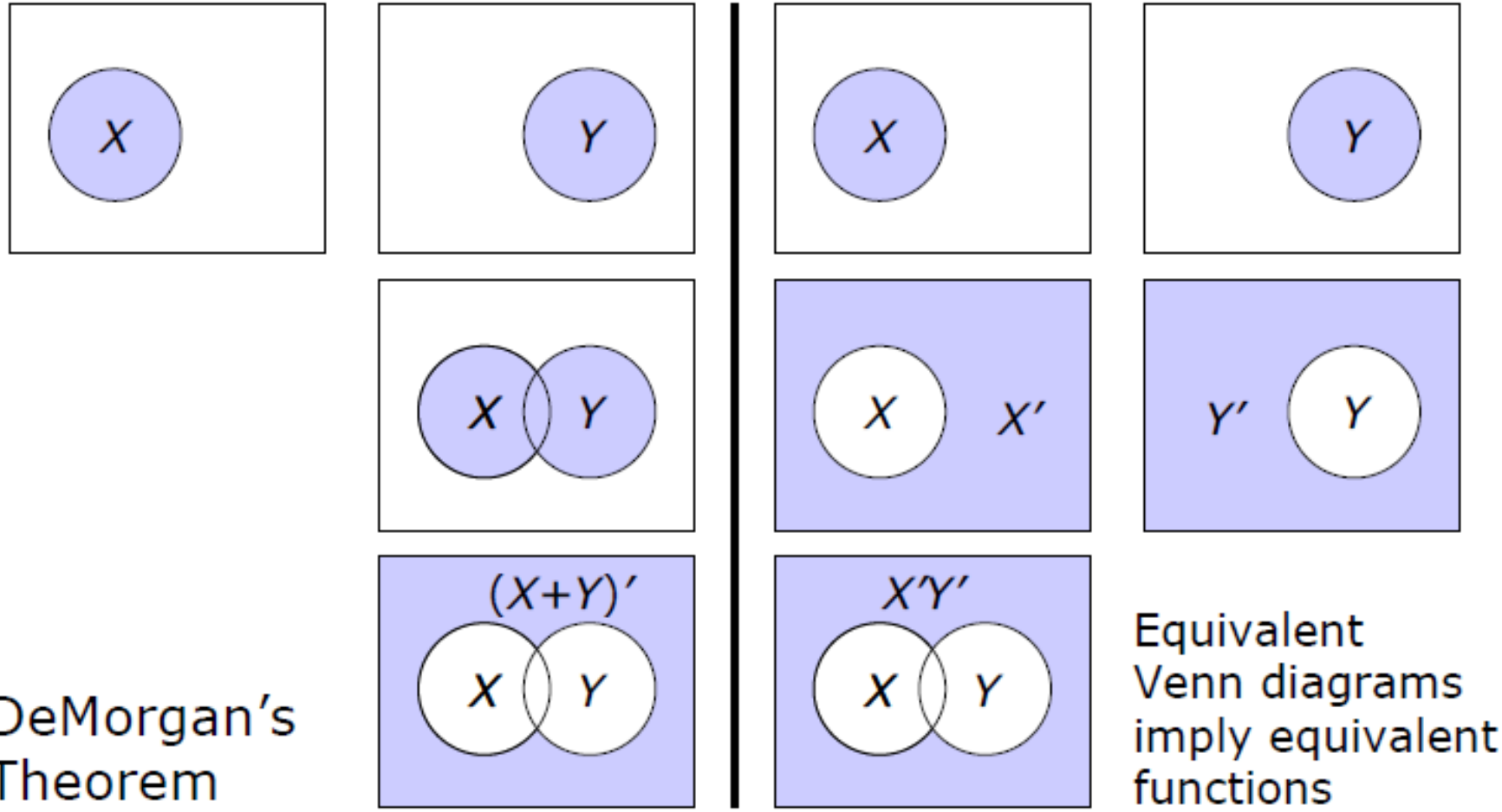



(g) $XY'$




(h) $XY+Z$

# Venn diagrams $(x+y)' = x'y'$




DeMorgan's Theorem

Equivalent Venn diagrams imply equivalent functions

# Notation and terminology

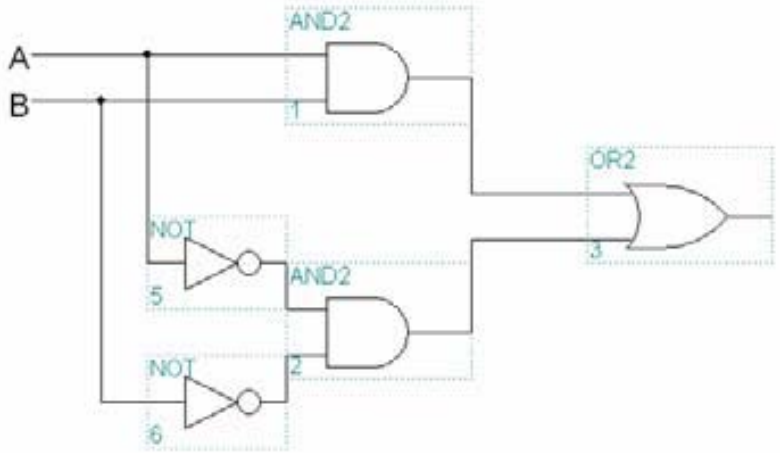
- Because of the similarity with arithmetic addition and multiplication operations, the *OR* and *AND* operations are often called the logical sum and product operations
- The expression
  - ABC+A'BD+ACE'
  - Is *a sum of three product* terms
- The expression
  - (A+B+C)(A'+B+D)(A+C+E')
  - Is *a product of three sum* terms

# Precedence of operations

• In the absence of parentheses, operations in a logical expression are performed in the order

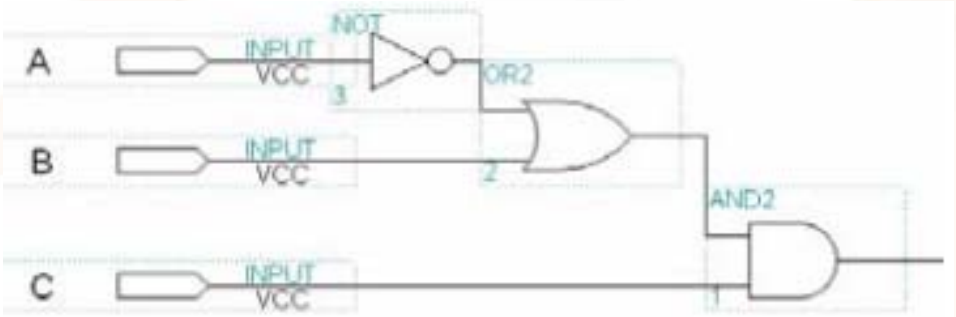
– *NOT, AND, OR*

• Thus in the expression AB+A'B', the variables in the second term are complemented before being ANDed together. That term is then ORed with the ANDed combination of A and B (the AB term)

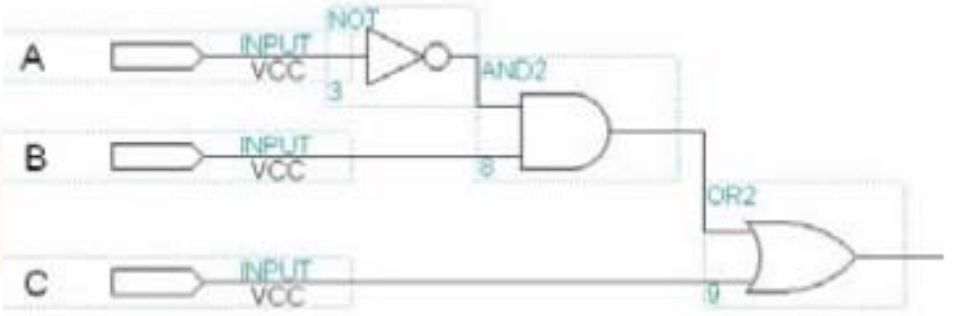


# Precedence of operations

- Draw the circuit diagrams for the following
  - f(a,b,c)=(a'+b)c



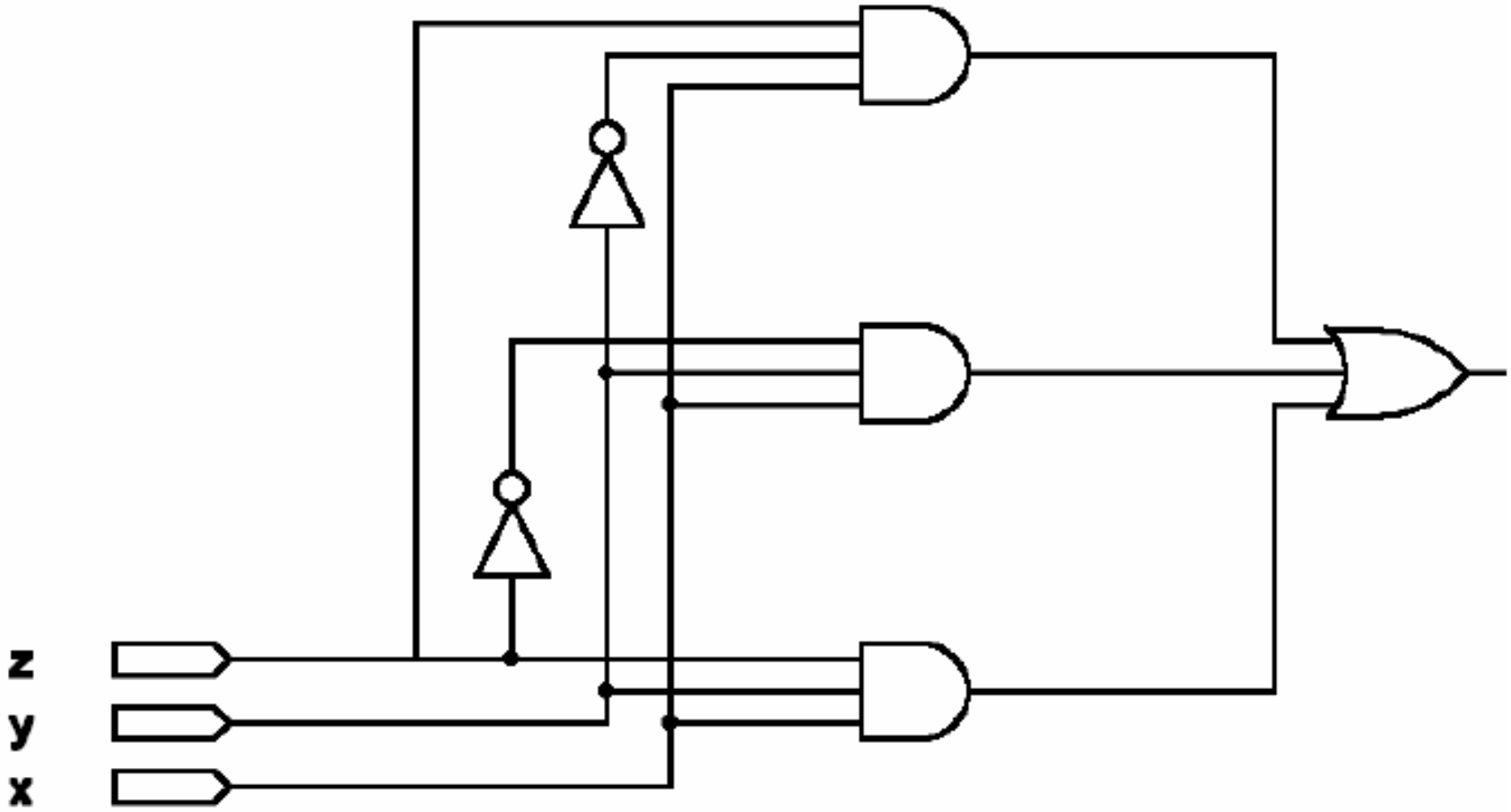

  - f(a,b,c)=a'b+c



# Example logic circuit design

- Assume we want to design a logic circuit with three inputs *x, y, and z*
- The circuit output should be 1 only when *x=1* and either *y or z (or both) is 1*
  - Three possible combinations
- x=1, y=0, z=1 => xy'z
- x=1, y=1, z=0 => xyz'
- x=1, y=1, z=1 => xyz
- The function could be written as
  - f(x,y,z)=xy'z+xyz'+xyz
- sum of products form

# Example logic circuit design



# Example logic circuit design

- Implements f correctly, $f(x,y,z)=xy'z+xyz'+xyz$ but is not the simplest such network

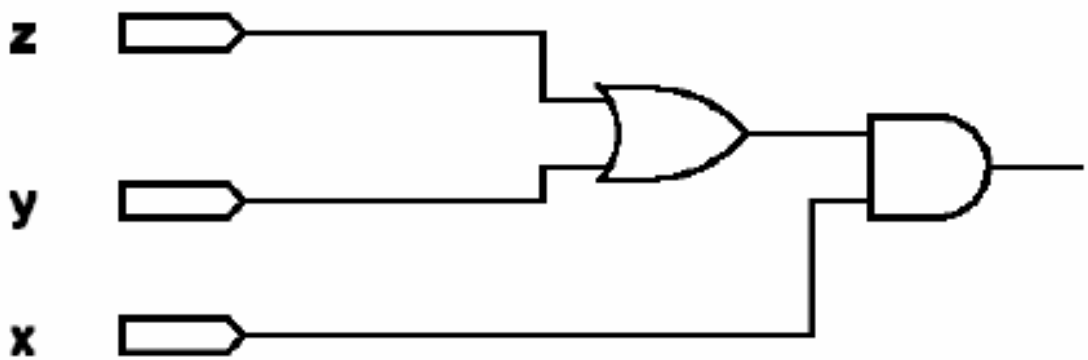
| $x$ | $y$ | $z$ | $f$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

$xy'z+xyz'+xyz$

$xy'z+xy$ from 14a

$x(y'z+y)$ from 12a

$x(y+z)$ from 16a



# Example logic circuit design

- Obviously, the cost (in terms of gates and connections) of this network is much less than the initial network
- The process of generating a circuit from a stated desired functional behavior is called ***synthesis***
- Generation of AND-OR style networks from a truth table is one of many types of synthesis techniques that we will cover

# Logic synthesis

- If a function f is described in a truth table, then an expression that generates f can by obtained (synthesized) by
  - Considering all rows in the table where *f=1*, *or*
  - By considering all rows in the table where *f=0*
- This will be an application of the principal of duality

# Minterms

- For a function of *n variables f(a,b,c,…n)*

– A minterm of *f is a product of n literals (variables)*in which each variable appears once in either true or complemented form, but not both

- f(a,b,c) -- minterm examples: abc, a’bc, abc’
- f(a,b,c) -- invalid examples: ab, c’, a’c

– An *n variable function has 2n valid minterms*

ELECTRICAL
Engineering KMITNB

# Minterms

| *Row number* | *x* | *y* | *z* | *Minterm* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $m_0=x'y'z'$ |
| 1 | 0 | 0 | 1 | $m_1=x'y'z$ |
| 2 | 0 | 1 | 0 | $m_2=x'yz'$ |
| 3 | 0 | 1 | 1 | $m_3=x'yz$ |
| 4 | 1 | 0 | 0 | $m_4=xy'z'$ |
| 5 | 1 | 0 | 1 | $m_5=xy'z$ |
| 6 | 1 | 1 | 0 | $m_6=xyz'$ |
| 7 | 1 | 1 | 1 | $m_7=xyz$ |

- Each row of a truth row corresponds to a single minterm
- When a function is written as a sum of minterms, the form is called a standard (or canonical) ***sum-of-products***

# Minterm notation

- An equation may be written in terms of m-notation

$f(a,b,c)=m_0+m_1+m_2+m_4$

$f(a,b,c)=a'b'c'+a'b'c+a'bc'+ab'c'$

$\underbrace{000}_{0} \quad \underbrace{001}_{1} \quad \underbrace{010}_{2} \quad \underbrace{100}_{4}$

$f(a,b,c)=\Sigma m(0,1,2,4)$

| *a* | *b* | *c* | *f* |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 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 | 0 |

# Minterm notation examples

- What is the minterm notation for the following function?
  - f(a,b,c)=abc+a'bc+abc'+a'b'c

- What is the function (in terms of variables) if the minterm notation is the following?
  - f(a,b,c)= Σm(1,5,6,7)

# Logic synthesis

- Duality suggests that:
  - If it is possible to synthesize a function *f by* considering the truth table rows where *f=1*, *then* it should also be possible to synthesize *f by* considering the rows for which *f=0*.
- This approach uses the complement of minterms, which are called ***maxterms***

# Maxterms

| *Row number* | *x* | *y* | *z* | *Maxterm* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $M_0=x+y+z$ |
| 1 | 0 | 0 | 1 | $M_1=x+y+z'$ |
| 2 | 0 | 1 | 0 | $M_2=x+y'+z$ |
| 3 | 0 | 1 | 1 | $M_3=x+y'+z'$ |
| 4 | 1 | 0 | 0 | $M_4=x'+y+z$ |
| 5 | 1 | 0 | 1 | $M_5=x'+y+z'$ |
| 6 | 1 | 1 | 0 | $M_6=x'+y'+z$ |
| 7 | 1 | 1 | 1 | $M_7=x'+y'+z'$ |

- Each row of a truth row corresponds to a single maxterm
- When a function is written as a product of maxterms, the form is called a standard (or canonical) ***product-of-sums***

# Maxterm notation

- An equation may be written in terms of M-notation

$f(a,b,c)=M_3 \cdot M_5 \cdot M_6 \cdot M_7$

$f(a,b,c)=(a+b'+c')(a'+b+c')(a'+b'+c)(a'+b'+c')$

$$\underbrace{0\,1\,1}_{3} \quad \underbrace{1\,0\,1}_{5} \quad \underbrace{1\,1\,0}_{6} \quad \underbrace{1\,1\,1}_{7}$$

$f(a,b,c)=\Pi M(3,5,6,7)$

| *a* | *b* | *c* | *f* |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 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 | 0 |

# Maxterm notation examples

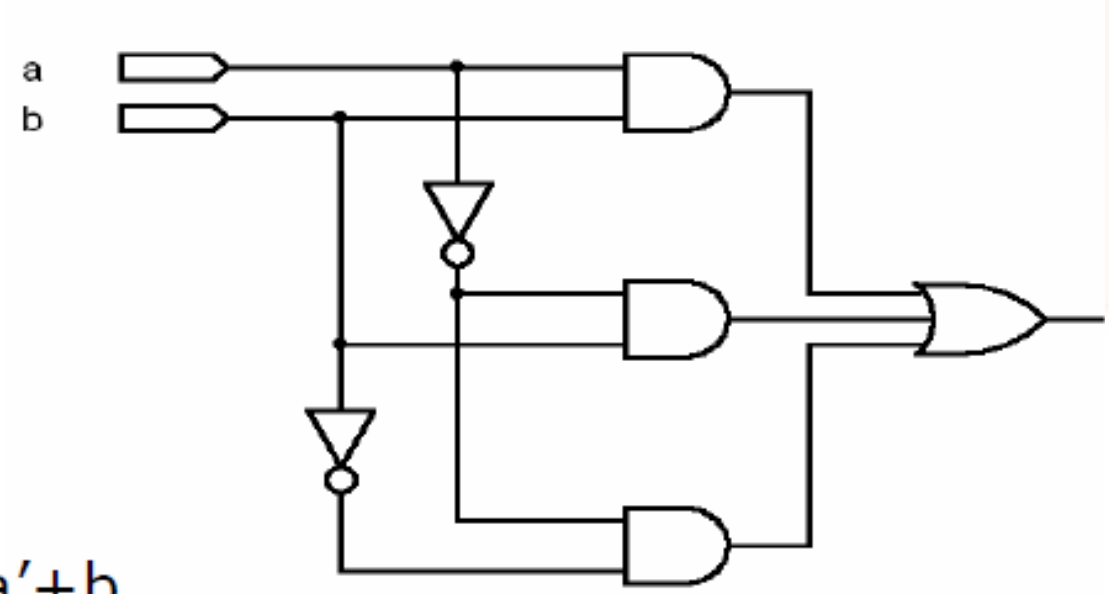
- What is the maxterm notation for the following function?
  – f(a,b,c)=(a+b+c)(a’+b+c)(a+b+c’)(a’+b’+c)

- What is the function (in terms of variables) if the maxterm notation is the following?
  – f(a,b,c)= ΠM(1,5,6,7)

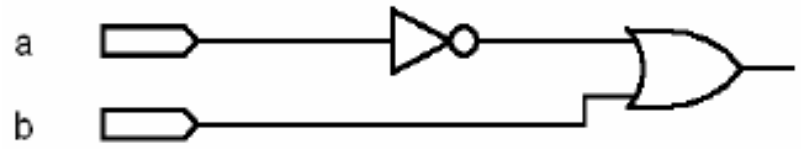
# Sum-of-products and minimality

- A function expressed in standard sum-of-products (or product-of-sums) form may not be minimal

| *a* | *b* | *f* |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




$f(a,b)=a'b'+a'b+ab=a'+b$



# Design examples

- Logic circuits provide a solution to a problem
- Some may be complex and difficult to design
- Regardless of the complexity, the same basic design issues must be addressed

1. *Specify the desired behavior of the circuit*
2. *Synthesize and implement the circuit*
3. *Test and verify the circuit*

# Three-way light control

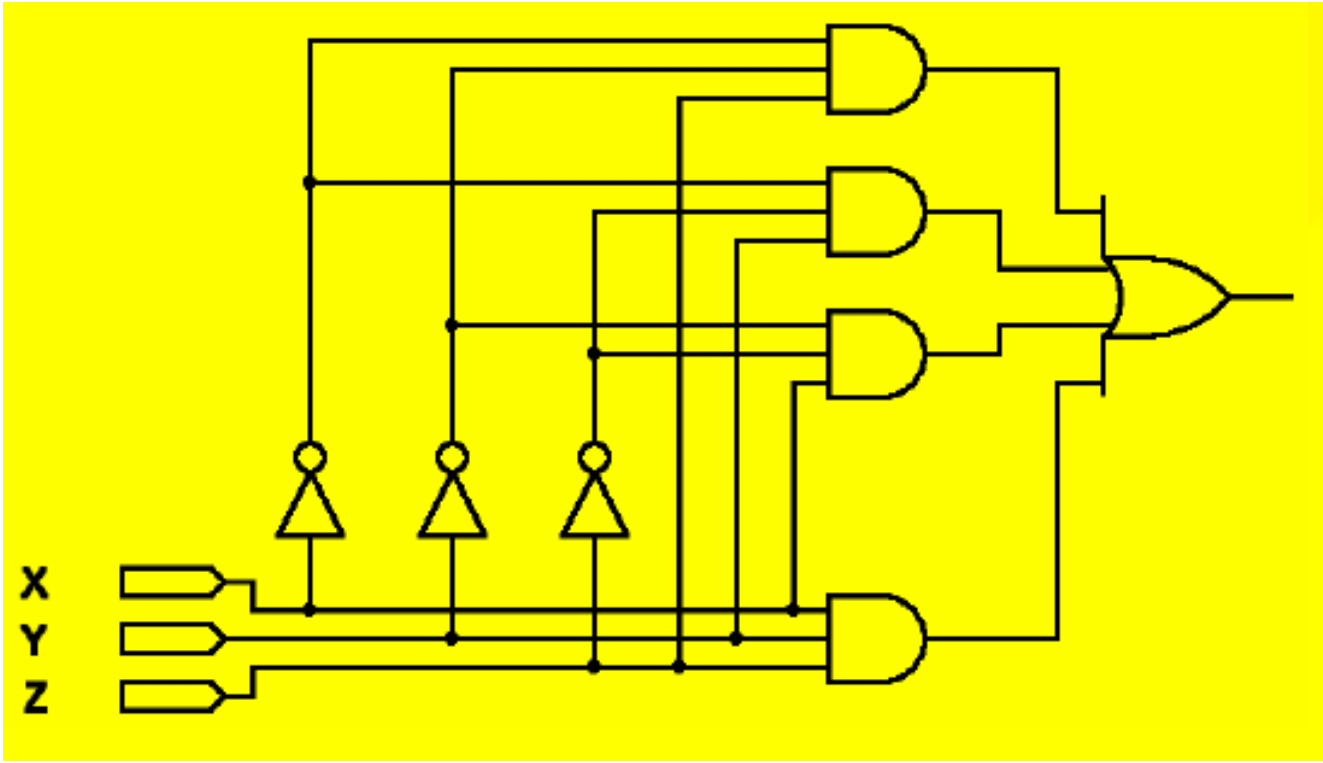
- Assume a room has three doors and a switch by each door controls a single light in the room.
  - Let x, y, and z denote the state of the switches
  - Assume the light is off if all switches are open
  - Closing any switch turns the light on. Closing another switch will have to turn the light off.
  - Light is on if any one switch is closed and off if two (or no) switches are closed.
  - Light is on if all three switches are closed

# Three-way light control

| 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 |

$$f(x,y,z)=m_1+m_2+m_4+m_7$$
$$f(x,y,z)=x'y'z+x'yz'+xy'z'+xyz$$

This is the simplest sum-of-products form.



# Multiplexer circuit

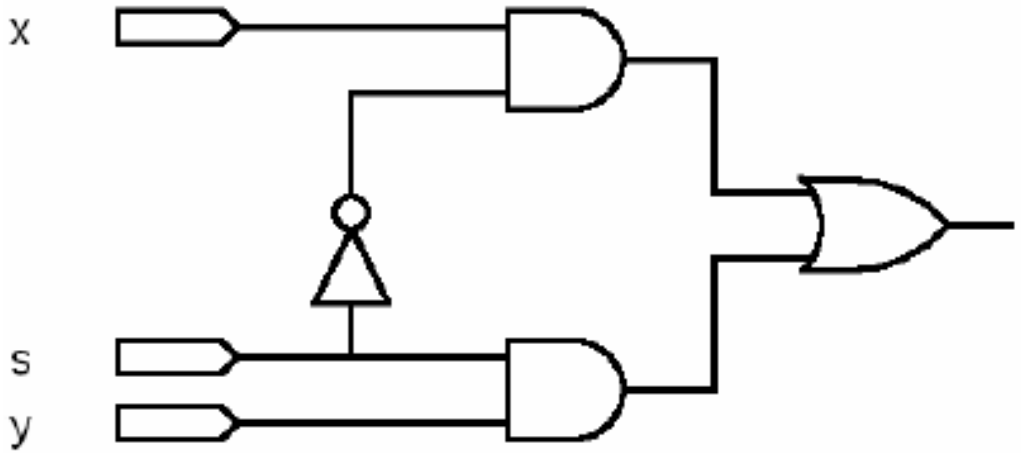
- In computer systems it is often necessary to choose data from exactly one of a number of sources
  - Design a circuit that has an output (f) that is exactly the same as one of two data inputs (x,y) based on the value of a control input (s)
    - If s=0 then f=x
    - If s=1 then f=y
  - The function f is really a function of three variables (s,x,y)
  - Describe the function in a three variable truth table

# Multiplexer circuit

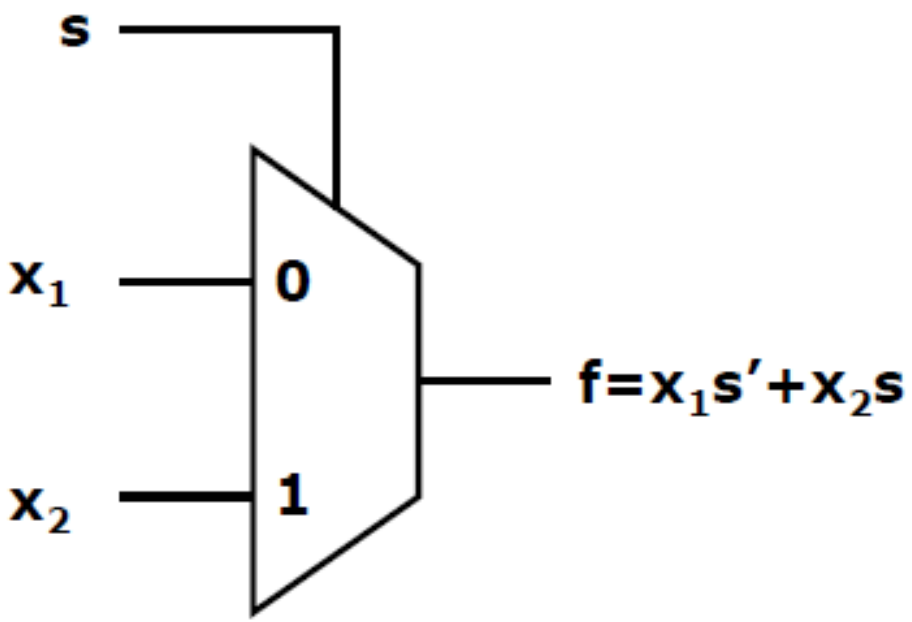
| s | x | y | f |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

convenient to put control signal on left

$$f(s,x,y)=m_2+m_3+m_5+m_7$$
$$f(s,x,y)=s'xy'+s'xy+sx'y+sxy$$
$$f(s,x,y)=s'x(y'+y)+sy(x'+x)$$
$$f(s,x,y)=s'x+sy$$



# Multiplexer circuit




Graphical symbol

| $s$ | $f(s,x_1,x_2)$ |
|---|---|
| 0 | $x_1$ |
| 1 | $x_2$ |

Compact truth table

ELECTRICAL
Engineering KMITNB

# Car safety alarm

- Design a car safety alarm considering four inputs
  - Door closed (D)
  - Key in (K)
  - Seat pressure (S)
  - Seat belt closed (B)
- The alarm (A) should sound if
  - The key is in and the door is not closed, or
  - The door is closed and the key is in and the driver is in the seat and the seat belt is not closed

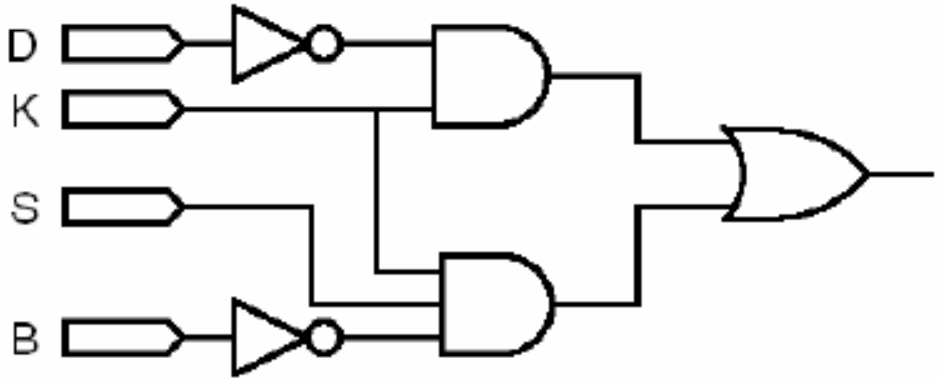
# Car Safety Alarm

$A(D,K,S,B)=\Sigma m(4,5,6,7,14)$

$A(D,K,S,B)=D'KS'B'+D'KS'B+D'KSB'+D'KSB+DKSB'$

$=D'KS'+D'KS+KSB'$

$=D'K+KSB'$




| D | K | S | B | A |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

ELECTRICAL
Engineering KMITNB

# H/W-Adder circuit

- Design a circuit that adds two input bits together (x,y) and produces two output bits (s and c)
  - S: sum bit
    - x=0, y=0 => s=0
    - x=0, y=1 => s=1
    - x=1, y=0 => s=1
    - x=1, y=1 => s=0
  - C: carry bit
    - x=0, y=0 => c=0
    - x=0, y=1 => c=0
    - x=1, y=0 => c=0
    - x=1, y=1 => c=1

# H/W-Majority circuit

- Design a circuit with three inputs (x,y,z) whose output (f) is 1 only if a majority of the inputs are 1
  - Construct a truth table
  - Write a standard sum-of-products expression for f
  - Draw a circuit diagram for the sum-of-products expression
  - Minimize the function using algebraic manipulation
    - During your minimization you can use any Boolean theorem, but leave the result in sum-of-products form (generate a minimum sum-of-products expression)
  - Draw the minimized circuit

# H/W- VST_Bot

Design Digital Circuit for Very Simple Tracking Robot (VST_Bot).The VST_Bot must be move in Track-line .

Digital Input Signals
each sensors S1,S2,S3
= 1 if in Track-line
= 0 if out Track-line

Digital Output Signals
ML =1 motor left on
0 motor left off
MR=1 motor right on
0 motor right off