Engr354: Digital Logic Circuits

Chapter 2: Introduction to Logic Circuits

Dr. Curtis Nelson

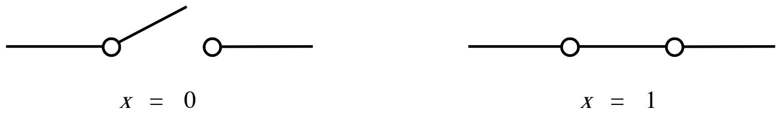
Chapter 2 Objectives

- Define and illustrate basic logic functions and circuits;
- Present Boolean algebra for dealing with logic functions;
- Illustrate logic gates and synthesis of simple circuits;
- Review CAD tools and the VHDL hardware description language.

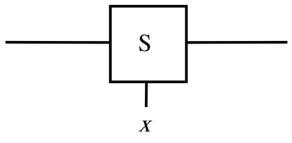
Binary Logic Circuits

- Logic circuits perform operations on digital signals;
- These circuits are implemented using electronic components;
- Binary logic circuits can be found in one of two states
  - 0 or 1;
  - off or on;
  - down or up;
  - not asserted or asserted;
  - etc.

Switch Representation



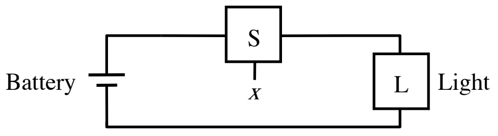
(a) Two states of a switch



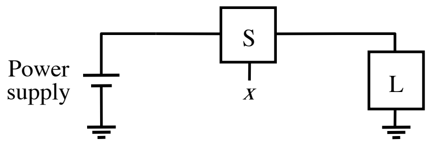
(b) Symbol for a switch

Switch Example

$L(x) = x$

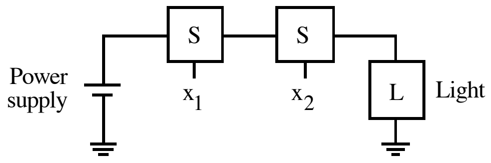


(a) Simple connection to a battery



(b) Using a ground connection as the return path

Two Basic Functions

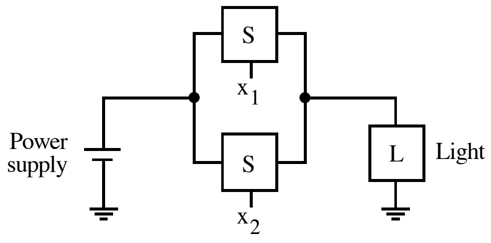


$L(x_1, x_2) = x_1 \cdot x_2$

$L = 1$ if $x_1 = 1$ **and** $x_2 = 1$

$L = 0$ otherwise

(a) The logical AND function (series connection)



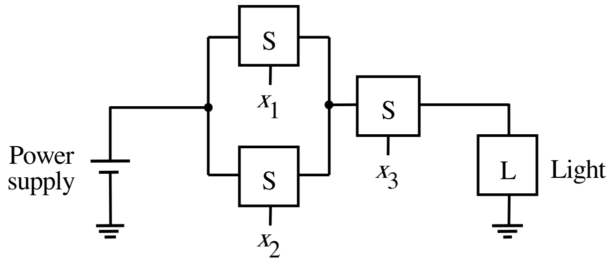
$L(x_1, x_2) = x_1 + x_2$

$L = 1$ if $x_1 = 1$ **or** $x_2 = 1$

$L = 0$ otherwise

(b) The logical OR function (parallel connection)

A Series-Parallel Example



$L(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3$

Truth Tables

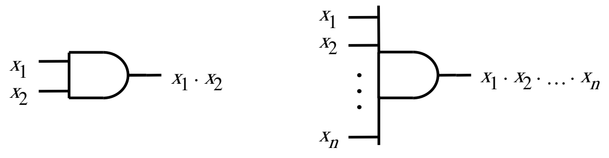
- All combinations of inputs on the left;
- Outputs on the right;
- 2-input **AND** and **OR** functions shown below.

x_1	x_2	$x_1 \cdot x_2$	$x_1 + x_2$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1
		AND	OR

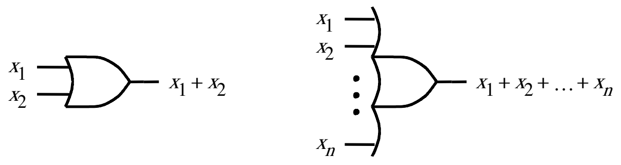
3-Input And and Or Functions

x_1	x_2	x_3	$x_1 \cdot x_2 \cdot x_3$	$x_1 + x_2 + x_3$
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

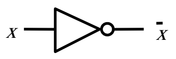
Basic Gates



(a) AND gates

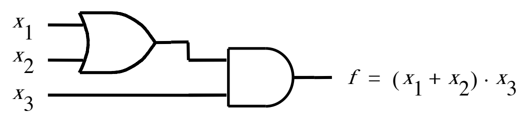


(b) OR gates

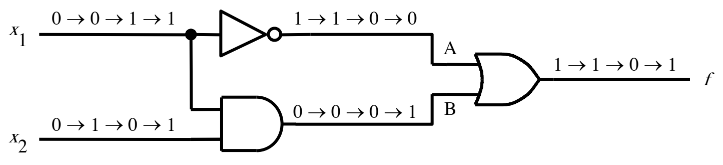


(c) NOT gate

Example Using Basic Gates



Sequencing of Inputs

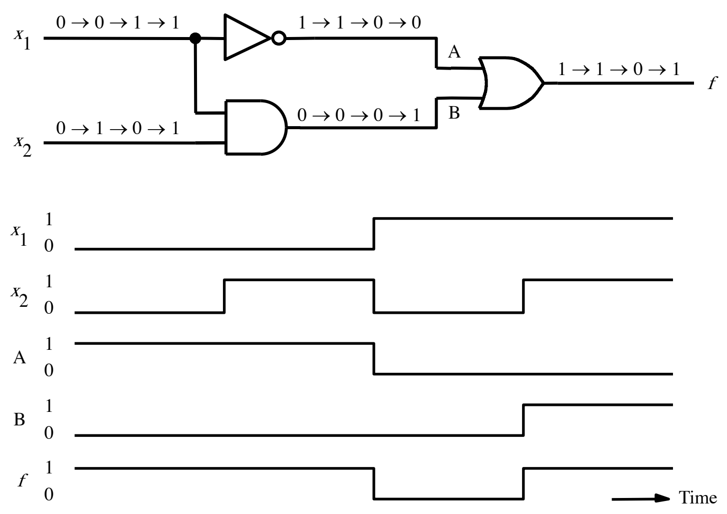


Network that implements $f = \bar{x}_1 + x_1 \cdot x_2$

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

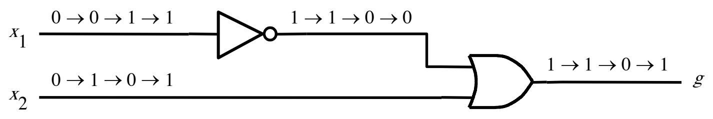
Truth table for f

Timing Diagram


x_1
$0 \rightarrow 0 \rightarrow 1 \rightarrow 1$
$1 \rightarrow 1 \rightarrow 0 \rightarrow 0$
A
$1 \rightarrow 1 \rightarrow 0 \rightarrow 1$
f
$0 \rightarrow 0 \rightarrow 0 \rightarrow 1$
B
$0 \rightarrow 1 \rightarrow 0 \rightarrow 1$
x_2
x_1
x_2
A
B
f
1
0
Time

Timing diagram

Example


x_1
$0 \rightarrow 0 \rightarrow 1 \rightarrow 1$
$1 \rightarrow 1 \rightarrow 0 \rightarrow 0$
$1 \rightarrow 1 \rightarrow 0 \rightarrow 1$
g
$0 \rightarrow 1 \rightarrow 0 \rightarrow 1$
x_2

Network that implements $g = \bar{x}_1 + x_2$

- Draw a timing diagram below:

Boolean Algebra

- In 1849, George Boole published a scheme for describing logical thought and reasoning;
- In 1930s, Claude Shannon applied Boolean algebra to describe circuits built with switches;
- Boolean algebra provides the mathematical foundation for digital design.

Notation

- INVERSION: $\overline{x} = x' = !x = \text{NOT } x$

  $\overline{f(x_1,x_2)} = \overline{x_1 + x_2} = (x_1 + x_2)' = !(x_1 + x_2)$

  $= \text{NOT}(x_1 + x_2)$
- AND: $x_1 \cdot x_2 = x_1 \wedge x_2 = x_1\, x_2$
- OR: $x_1 + x_2 = x_1 \vee x_2$

Precedence of Operations

- In the absence of parentheses, operations are performed in this order: NOT, AND, OR

$$x_1 x_2 + x_1' x_2' = (x_1 x_2) + ((x_1') (x_2'))$$

Principle of Duality

- On the following pages, axioms and theorems are listed in pairs to show the principle of duality;
- Given a logic expression, its *dual* is found by exchanging $+$ and $\bullet$ operators and 0's and 1's;
- The dual of any true statement is **always** true.

Axioms of Boolean Algebra

1.	$0 \cdot 0 = 0$	$1 + 1 = 1$
2.	$1 \cdot 1 = 1$	$0 + 0 = 0$
3.	$0 \cdot 1 = 1 \cdot 0 = 0$	$1 + 0 = 0 + 1 = 1$
4.	if $x = 0$ then $\overline{x} = 1$	if $x = 1$ then $\overline{x} = 0$

Single-Variable Theorems

5.	$x \cdot 0 = 0$	$x + 1 = 1$
6.	$x \cdot 1 = x$	$x + 0 = x$
7.	$x \cdot x = x$	$x + x = x$
8.	$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$
9.	$\overline{\overline{x}} = x$	

2- and 3-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. $x + y \cdot z = (x + y) \cdot (x + z)$	

2- and 3-Variable Properties

13a. $x + x \cdot y = x$	Absorption
13b. $x \cdot (x + y) = x$	
14a. $x \cdot y + x \cdot \overline{y} = x$	Combining
14b. $(x + y) \cdot (x + \overline{y}) = x$	
15a. $\overline{x \cdot y} = \overline{x} + \overline{y}$	DeMorgan's Thm
15b. $\overline{x + y} = \overline{x} \cdot \overline{y}$	
16. $x + \overline{x} \cdot y = x + y$	$x \cdot (\overline{x} + y) = x \cdot y$

Truth Table Proof of DeMorgan's Theorem

15a. $\overline{x \cdot y} = \overline{x} + \overline{y}$ DeMorgan's Theorem

x	y	$x \cdot y$	$\overline{x \cdot y}$	$\overline{x}$	$\overline{y}$	$\overline{x} + \overline{y}$
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

LHS RHS

Boolean Algebra Examples

1) (a + b)(a' + b') =

2) a'b'c' + a'b'c + ab'c' + ab'c =

3) a'b'c' + a'b'c + a'bc' + abc' + ab'c' + ab'c =

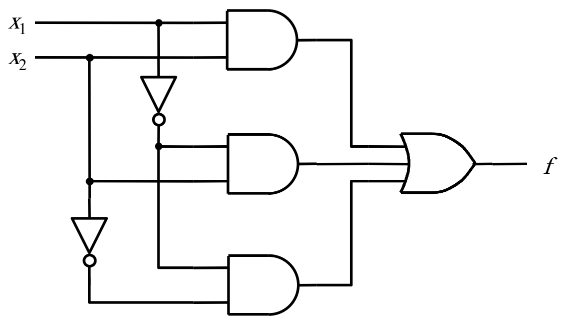
1) ab' + ba'
2) b'
3) b' + c'

Synthesis

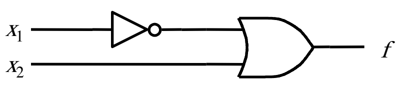
- Synthesis is the process of creating a circuit from a specification, i.e. a truth table, schematic, VHDL code, etc.

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

Two Implementations of the Function *f*



(a) Sum-of-products form



(b) Minimal-cost realization

Minterms and Maxterms

- Minterms
  - For a function of *n* variables, a product term in which each of the *n* variables appears once is called a minterm
    - Variables may appear in complemented or uncomplemented form;
    - A given minterm is formed by including x_i if $x_i = 1$ and by including not(x_i) if $x_i = 0$.
- Maxterms (Principle of duality applies)
  - For a function of *n* variables, a sum term in which each of the *n* variables appears once is called a maxterm
    - Variables may appear in complemented or uncomplemented form.
    - A given maxterm is formed by including x_i if $x_i = 0$ and by including not(x_i) if $x_i = 1$.

Three-Variable Minterm and Maxterm Table

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \overline{x}_1\overline{x}_2 x_3$	$M_1 = x_1 + x_2 + \overline{x}_3$
2	0	1	0	$m_2 = \overline{x}_1 x_2\overline{x}_3$	$M_2 = x_1 + \overline{x}_2 + x_3$
3	0	1	1	$m_3 = \overline{x}_1 x_2 x_3$	$M_3 = x_1 + \overline{x}_2 + \overline{x}_3$
4	1	0	0	$m_4 = x_1\overline{x}_2\overline{x}_3$	$M_4 = \overline{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\overline{x}_2 x_3$	$M_5 = \overline{x}_1 + x_2 + \overline{x}_3$
6	1	1	0	$m_6 = x_1 x_2\overline{x}_3$	$M_6 = \overline{x}_1 + \overline{x}_2 + x_3$
7	1	1	1	$m_7 = x_1 x_2 x_3$	$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$

Sum-of-Products (SOP) Form

- A function *f* can be represented by an expression that is a sum of minterms.
- For example, the function below could be represented as

  $f(x_1,x_2,x_3) = !x_1!x_2x_3 + x_1!x_2!x_3 + x_1!x_2+x_3 + x_1x_2!x_3$
- Using short hand notation

  $f(x_1,x_2,x_3) = m_1+m_4+m_5+m_6$ or

  $f(x_1,x_2,x_3) = \Sigma$ m(1,4,5,6)

Row number	x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Product-of-Sums (POS) Form

- The same function *f* can be represented by an expression that is a product of sums of maxterms (this follows from DeMorgan's laws).
- For example, the function below could also be represented as

  $f(x_1,x_2,x_3) = (x_1+x_2+x_3)(x_1+!x_2+x_3)(x_1+!x_2+!x_3)(!x_1+!x_2+!x_3)$
- Using short hand notation,

  $f(x_1,x_2,x_3) = M_0M_2M_3M_7$ or

  $f(x_1,x_2,x_3) = \Pi$ M(0,2,3,7)

Row number	x_1	x_2	x_3	$f(x_1,x_2,x_3)$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Canonical

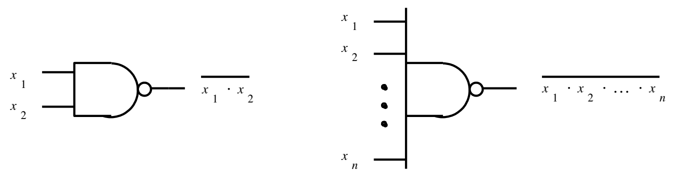
- A logic expression is said to be canonical if each term contains all variables, either complemented or uncomplemented.

2-Variable Example

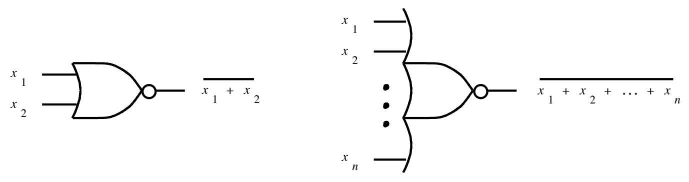
- Synthesize the following function using four different, but equivalent, expressions
  - Minterm form;
  - Maxterm form;
  - Canonical form;
  - Minimum form.

x_1	x_2	$f(x_1.x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

NAND and NOR Gates

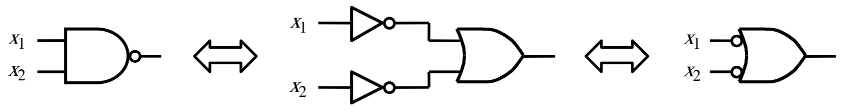

x_1
x_2
$\overline{x_1 \cdot x_2}$
x_1
x_2
x_n
$\overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n}$

(a) NAND gates

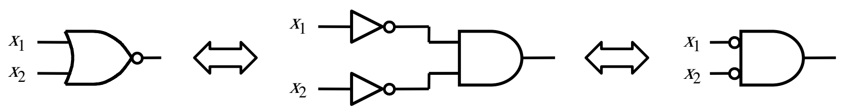


(b) NOR gates

DeMorgan’s Theorems in Terms of Logic Gates



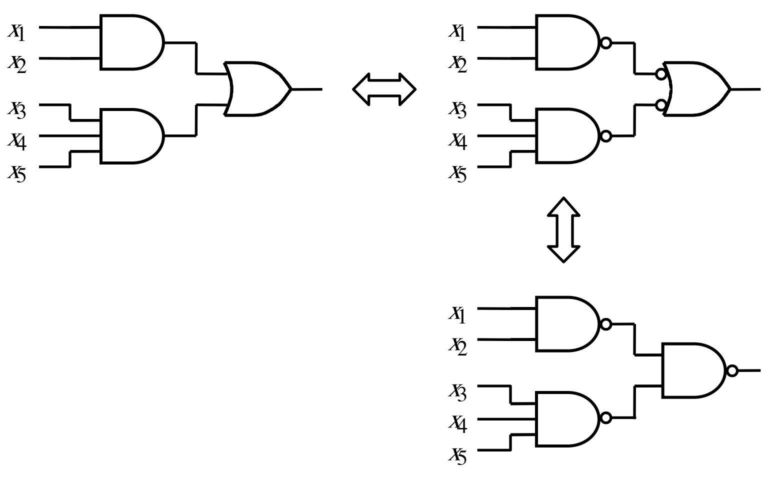
(a) $\overline{x_1 x_2} = \overline{x}_1 + \overline{x}_2$



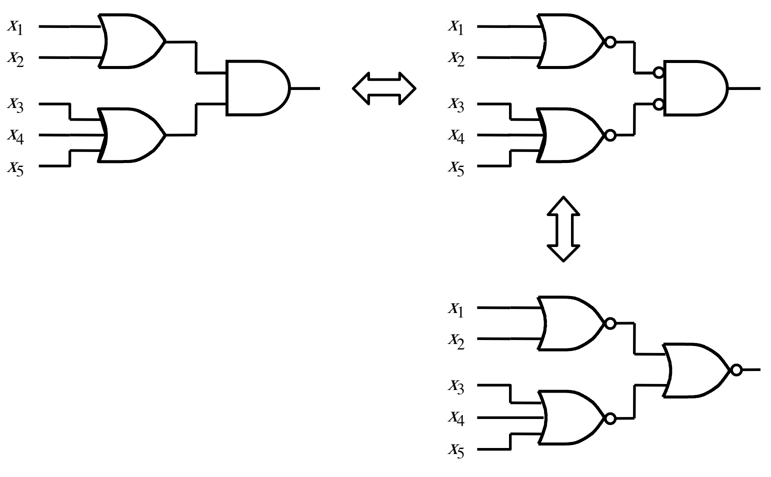
(b) $\overline{x_1 + x_2} = \overline{x}_1 \overline{x}_2$

- Function vs. Gate

SOP Implementation Using NAND Gates



POS Implementation Using NOR Gates

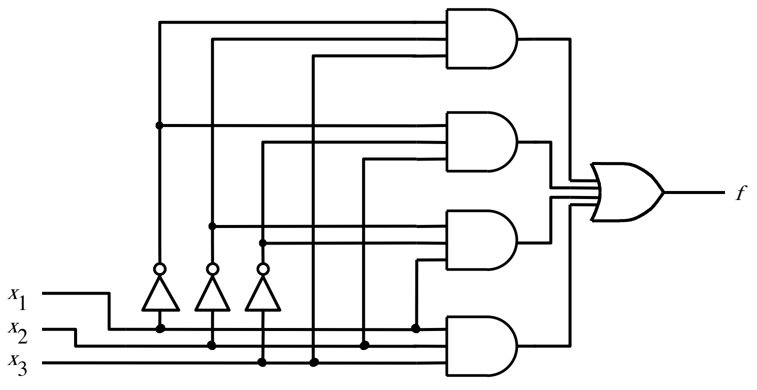


Example

- Synthesize the following function using four different, but equivalent, expressions
  - Minterm form;
  - Maxterm form;
  - Canonical form;
  - Minimum form.

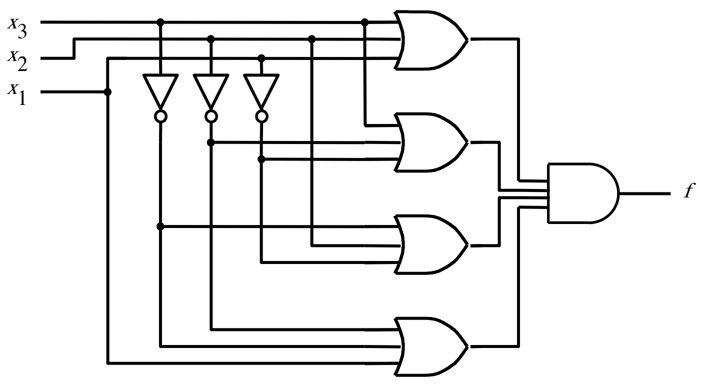
x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Canonical SOP Implementation



(a) Sum-of-products realization

Canonical POS Implementation



(b) Product-of-sums realization

Minimal Implementation

Textbook Problem 2.28 Brown 3rd Ed.

- Design the simplest circuit that has three inputs, a, b, and c, which produces an output value of 1 whenever two or more of the input variables have the value of 1; otherwise, the output has to be 0.
- Implement the circuit three different ways
  - With minimum *and/or* logic;
  - With *nand* logic gates only;
  - With *nor* logic gates only.

Design Entry

- Truth tables
  - Practical only for small circuits.
- Schematic capture
  - Interconnect symbols in some *library;*
  - Facilitates *hierarchical design;*
  - Good for medium-to-large circuits;
  - Difficult to use for very large circuits.

Design Entry - Continued

- Hardware description languages (HDL's)
  - Similar to a programming language;
  - VHDL and Verilog HDL are IEEE standards;
  - Provide design *portability;*
  - Allow for sharing and design reuse;
  - Support hierarchical design;
  - Can be combined with schematics.

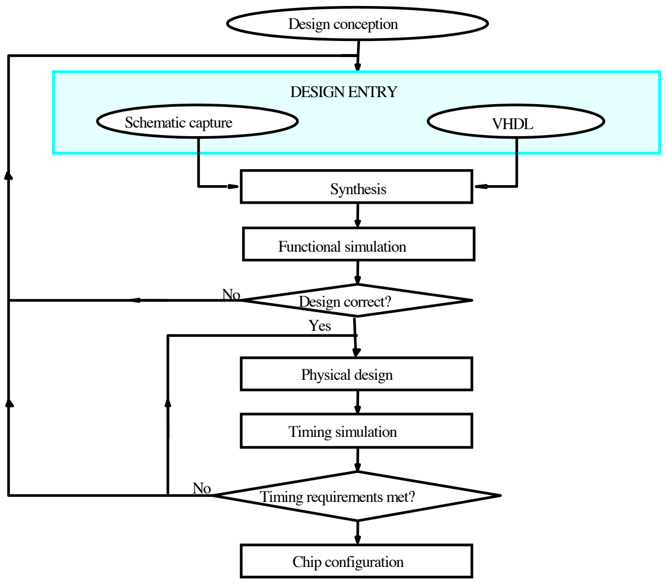
Logic Synthesis

- *Logic synthesis,* or *logic optimization,* is the process of translating a truth table, schematic, or VHDL code into a network of logic gates;
- What makes a circuit good depends on the application – do you want to optimize for speed, area, power?

Simulation

- A *functional simulator* is used to determine if a designed circuit operates correctly from a **logic** perspective.
- Circuit **verification**
  - User provides input values to the circuit;
  - Simulator determines the circuit response;
  - User checks responses against desired outputs.
- A *timing simulator* is used to check correctness by incorporating the ***electrical*** characteristics of a logic design in addition to the *logical* performance. This simulation requires
  - Technology mapping;
  - Layout synthesis.

A Typical CAD System



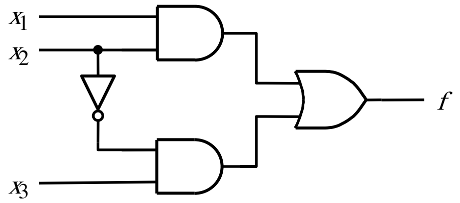
VHDL

- VHDL - Very high speed integrated circuit hardware description language;
- Original IEEE standard adopted in 1987;
- Revised standard in 1993;
- Originally used for documentation and simulation;
- Now, it is also used for synthesis;
- Very complex language, but only a subset is needed to design a wide range of circuits.

Representing Digital Signals in VHDL

- Each logic signal in a circuit is a data object in VHDL code;
- Data objects in VHDL are assigned *types;*
- A simple type is BIT which is used for objects that can take only 2 values: 0 or 1.

A Simple Logic Function Example



```
ENTITY example1 IS
      PORT ( x1, x2, x3   : IN      BIT ;
                  f                : OUT  BIT ) ;
END example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
      f <= (x1 AND x2) OR (NOT x2 AND x3) ;
END LogicFunc ;
```

Chapter 2 Summary

- Defined and illustrated basic logic functions and circuits;
- Presented Boolean algebra for dealing with logic functions;
- Illustrated logic gates and synthesis of simple circuits;
- Reviewed CAD tools and the VHDL hardware description language.