DIGITAL TECHNICS

Dr. Bálint Pődör

Óbuda University,
Microelectronics and Technology Institute

1. LECTURE:
COMBINATONAL CIRCUITS BASIC CONCEPTS

1st (Autumn) term 2018/2019

1. LECTURE: COMBINATONAL CIRCUITS BASIC CONCEPTS

1. General introduction to the course
2. Combinational circuits: basic concepts
3. Boolean algebra and logic functions: a review

AIMS AND SCOPE OF THE COURSE

This course will give an overview of the basic concepts and applications of digital technics, from Boolean algebra to microprocessors.

The lectures will cover more advanced materials and subjects than those contained the introductory three semester course of the B.Sc. programme. It will focus more on the general concepts of the subject and less on the practical details.

In this respect it is supposed that the students have already a good foundation and a certain level of hands-on experience in digital technics and electronics.

3

TOPICS IN FOCUS

Basic concepts of digital technics

Programmable Logic Devices (PLDs) and Field Programmable Gate Arrays (FGPAs)

Digital (combinational) design and synthesis

Synchronous sequential circuits analysis and synthesis

Arithmetic circuits, adders and multipliers

MOS, CMOS and VLSI digital circuits.

D/A and A/D converters.

4

COMBINATIONAL CIRCUITS:

AN INTRODUCTION

WITH EXAMPLES

5

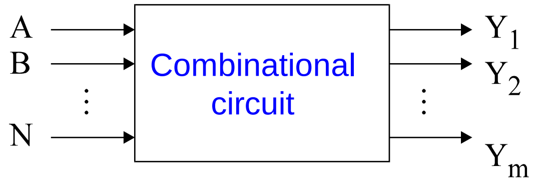
DIGITAL NETWORKS: CLASSIFICATION

Digital/logic circuits/networks can be classified into two groups:

1. Combinational logic networks
Results of an operation depend *only* on the present inputs to the operation
Uses: perform arithmetic, control data movement, compare values for decision making

2. Sequential logic networks
Results depend on both the inputs to the operation *and* the result of the previous operation
Uses: counter, controllers, etc.

COMBINATIONAL CIRCUITS: GENERALIZED MODEL AND PROPERTIES



$Y_i = F_i\ (A, B, ..., N) \quad i = 1, 2, ... M$

Black-box model of combinational circuits.

- The combinational circuit maps an input (signal) combination to an output (signal) combination.
- A combinational circuit is a circuit with no ”memory”.
- The same input combination always implies the very same output combination (except transients).
- The reverse is not true. For a given output combination different input combinations can belong.

7

COMBINATIONAL CIRCUITS: GENERALIZED MODEL AND PROPERTIES

- The independent variables of a Boole function sometimes denoted by the capital letters of an English ABC.
- The Latin word for letter is Literal, for a logic expression of „n” variables frequently called an expression of „n literals”.
- In the followings we apply that name too.

8

COMBINATIONAL CIRCUITS: GENERALIZED MODEL AND PROPERTIES

- The aforementioned „sequences" of logic variables are not sequences of time (in case of Combinational Circuits)
- At the input they can be considered as a binary combination of given values of n literals
- Similarly at the output they can be considered as a binary combination of values of m literals. (m generally does not equal to n.)
  (Sometimes vertexes of n or m binary components respectively.)
- E.g. $\langle 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \rangle$

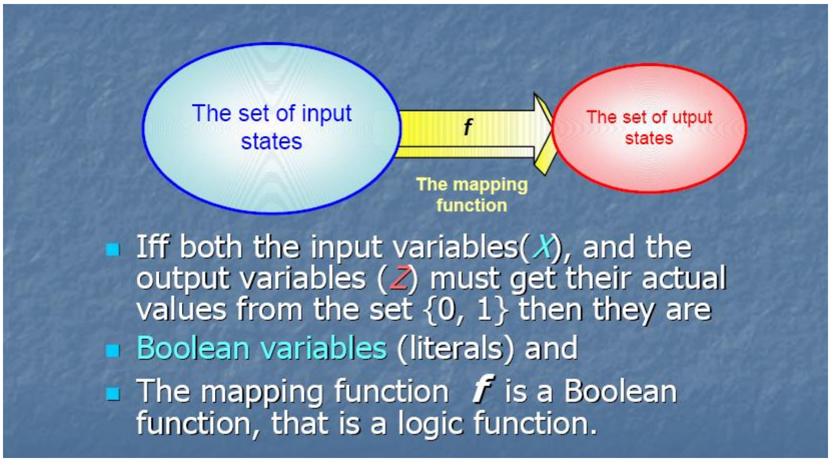
9

COMBINATIONAL CIRCUITS: GENERALIZED MODEL AND PROPERTIES

- The Combinational Circuit is mapping an input (signal) combination to an output (signal) combination. This is why it's called Combinational Circuit. (CC)
- The very same input combination always implies the very same output combination (when the circuit is in its stationary state).
  E.G. In case a ROM the device answers the very same output when the given particular input address is applied.

10

COMBINATIONAL CIRCUITS: THE MAPPING FUNCTION



11

STATIC MODEL OF COMBINATIONAL CIRCUITS

A static model represents the state sequence of a circuit, i.e. its event history and never describes its transients.

The time as variable never occurs in the functions of the states of the circuit.

For that it’s not necessary applying differential equations as in case of analogous circuits.

12

STATIC MODEL OF COMBINATIONAL CIRCUITS

All the operating models of logic automata are static models. (That's valid for systems modeling as well.)

For that there are no time dependent logic functions (that is for that static models).

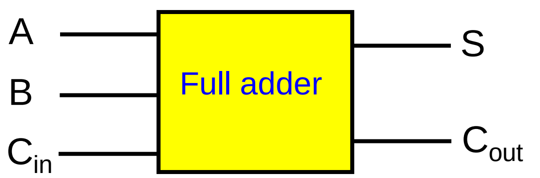
The conception of such static models are favorites of systems modeling as well.

13

EXAMPLE: 1-BIT FULL ADDER

- Its function is to add two bits and the carry from the previous position, and to generate the sum and the carry

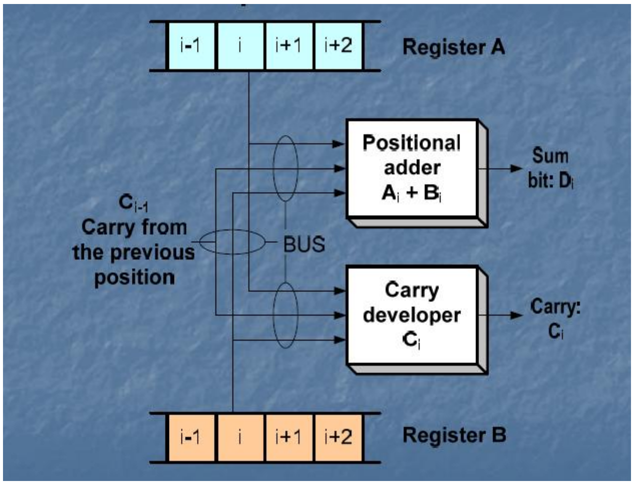
$S = S(A,B,C_{in})$ $\qquad$ $C_{out} = C(A,B,C_{in})$



The full adder as a combinational logic circuit will be used throughout in these lectures as vehicle to demonstrate and explain various concepts in digital logic

14

EXAMPLE: 1-BIT FULL ADDER



FULL ADDER

The task has been divided into two part-tasks:

Adder block of the bits of i-th positions

and

Carry (Ci) producing block for the i-th position.

Note
Both blocks are driven by the same bus
Both blocks have single output each
It's obvious that applying such Full Adders an optional number of parallel bits can be added.

FULL ADDER: BOOLEAN FUNCTIONS

Sum

$$S_i = \bar{A}_i\bar{B}_iC_{i-1} + \bar{A}_iB_i\bar{C}_{i-1} + A_i\bar{B}_i\bar{C}_{i-1} + A_iB_iC_{i-1}$$

Carry

$$C_i = \bar{A}_iB_iC_{i-1} + A_i\bar{B}_iC_{i-1} + A_iB_i\bar{C}_{i-1} + A_iB_iC_{i-1}$$

$$= A_iB_i + A_iC_{i-1} + B_iC_{i-1} = A_iB_i + (A_i + B_i)C_{i-1}$$

$$= A_iB_i + (A_i \oplus B_i)C_{i-1}$$

The sum can be expressed as a three-variable exclusive OR function ($S_i = A_i \oplus B_i \oplus C_i$).

The carry is the three-variable majority function and can also be expressed in various other algebraic forms.

17

FULL ADDER: GENERAL RELEVANCE

The full adder is the fundamental building block in many arithmetic circuits, such as adders and multipliers.

Since these circuits strongly affect the overall performance in current digital ICs, their speed optimization is crucial in high performance applications, and typical applications require a tradeoff between power consumption and speed.

In addition, as arithmetic circuits significantly contribute to the overall power budget, their power consumption reduction becomes the main objective to pursue in low-power ICs used in portable electronic equipment.

18

LOGIC OR BOOLEAN ALGEBRA

A SHORT OVERVIEW

OR BOOLEAN ALGEBRA IN A NUTSHELL

19

BOOLEAN ALGEBRA

Logic circuits are the basis for modern digital computer and other digital systems. To appreciate how digital systems operate one needs to understand digital logic and Boolean algebra.

Boolean logic forms the basis for computation in modern binary computer systems. One can represent any algorithm, or any electronic computer circuit, using a system of Boolean equations.

20

BOOLEAN ALGEBRA: ITS ROOTS

The Boolean algebra is a brand of mathematics that was first developed systematically, because of its applications to logic, by the English mathematician *George Boole*, around 1850.

A modern engineering application is to switching, digital and computer circuit design.

Contributions by *Augustus De Morgan* (contemporary of Boole) and by *Claude Shannon* (1930'ies and 1940'ies) are also important.

21

BOOLEAN ALGEBRA AND DIGITAL CIRCUITS

The connection between Boolean algebra and switching circuits has been established by Claude Shannon in the 1930's.

Boolean algebra is the main analytical tool for the analysis and synthesis of logic circuits and networks.

Boolean logic: Rules for handling Boolean constants and variables that can take on 2 values

–True/false; on/off; closed/open; yes/no; 1/0; high/low (voltage)
–Three fundamental operations: AND, OR and NOT

22

BOOLEAN ALGEBRA: RELEVANCE

In the 1930s, while studying switching circuits, Claude Shannon observed that one could also apply the rules of Boole's algebra in this setting, and he introduced switching algebra as a way to analyze and design circuits by algebraic means in terms of logic gate.

Shannon already had at his disposal the abstract mathematical apparatus, thus he cast his switching algebra as the two-element Boolean algebra.

In circuit engineering settings today, there is little need to consider other Boolean algebras, thus "switching algebra" and "Boolean algebra" are often used interchangeably.

BOOLEAN ALGEBRA: RELEVANCE

Efficient implementation of Boolean functions is a fundamental problem in the design of combinational circuits.

Modern electronic design automation tools for VLSI circuits often relay on an efficient representation of Boolean functions like (reduced ordered) binary decision diagrams (BDD) for logic synthesis and formal verification.

Connection Between
Boolean Calculus and Physical Circuits
Shannon 1938

Shannon
1916-2001

A Symbolic Analysis of Relay and Switching Circuits*

*Claude E. Shannon***

Shannon's advisor both MSc and PhD - a mathematician

* *Transactions American Institute of Electrical Engineers*, vol. 57, 1938. (Paper number 38-80, recommended by the AIEE committees on communication and basic sciences and presented at the AIEE summer convention, Washington, D.C., June 20-24, 1938. Manuscript submitted March 1, 1938; made available for preprinting May 27, 1938.)

** Claude E. Shannon is a research assistant in the department of electrical engineering at Massachusetts Institute of Technology, Cambridge. This paper is an abstract of a thesis presented at MIT for the degree of master of science. The author is indebted to Doctor F. L. Hitchcock, Doctor Vannevar Bush, and Doctor S. H. Caldwell, all of MIT, for helpful encouragement and criticism.

BOOLEAN OPERATORS

- AND
  - Result TRUE if and only if *both* input operands are true
  - C = A ● B

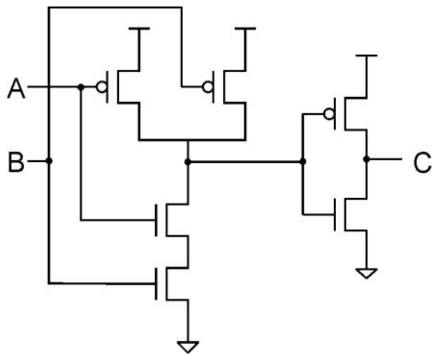
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

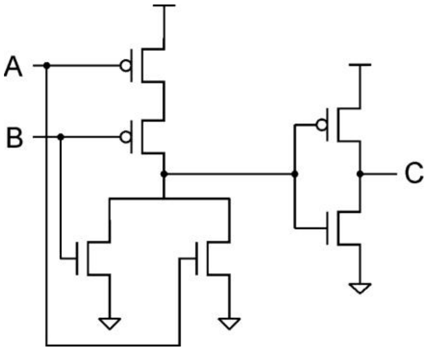
- OR
  - Result TRUE if *any* input operands are true
  - C = A + B

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

26

CMOS IMPLEMENTATION: AND, OR





AND gate

C = A B

OR gate

C = A + B

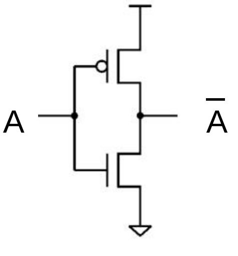
27

BOOLEAN OPERATORS

- NOT
  - Result TRUE if single input value is FALSE
  - C = $\overline{A}$

A	C
0	1
1	0

Implementation



28

BOOLEAN OPERATORS: EXCLUSIVE-OR

- EXCLUSIVE-OR
  - Result TRUE if either A or B is TRUE *but not both*
  - C = $A \oplus B$
  - Can be derived from OR, AND and NOT
    - $A \oplus B = (A + B) \bullet (\overline{A \bullet B})$

      A xor B equals A or B but not both A and B
    - $A \oplus B = (A \bullet \overline{B}) + (B \bullet \overline{A})$

      A xor B = either A and not B or B and not A

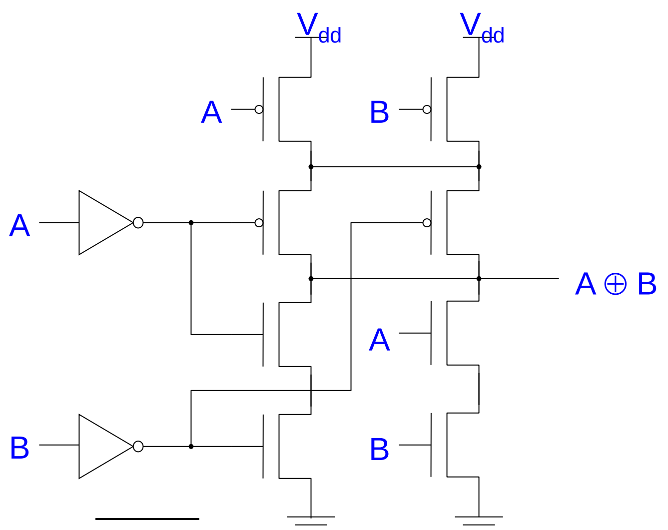
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

The XOR is not a fundamental Boolean operator, it can rather be considered as one of the (simplest) Boolean functions.

Logic circuit families usually offer XOR „gates” too.

29

CMOS IMPLEMENTATION: XOR



Note $A \oplus B = \overline{A \odot B} = \overline{A}\,\overline{B} + A B$ (!)

NOTATIONS:
ENGINEERING VERSUS MATHEMATICS

In electrical engineering and digital electronics the dot (•), the plus sign (+), and the overline (bar) are used for the logic operations of AND, OR and NOT respectively.

In mathematics, more specifically in mathematical logic (Boolean) algebra the respective notations are

$\wedge$ (conjunction),
$\vee$ (disjunction), and
$\neg$ (negation).

31

AND, OR, NOT, NAND, NOR

All logic functions and circuits can be described in terms of the three fundamental elements.

While the NOT, AND, OR functions have been designed as individual circuits in many circuit families, by far the most common functions realized as individual circuits are the NAND and NOR circuits. A NAND can be described as equivalent to an AND element driving a NOT element. Similarly, a NOR is equivalent to an OR element driving a NOT element.

The reason for this strong bias favouring inverting outputs is that the transistor, and the vacuum tube which preceded it, are by nature inverters or NOT-type devices when used as signal amplifiers. Electric and electronic switches (gates) do not readily perform the OR and AND logic operations, but most commercially available gates do perform the combined operations AND-NOT (NAND) and OR-NOT (NOR).

32

BOOLEAN POSTULATES

1. Boolean algebra is defined on a set of two-valued elements

2. Each element of the set has its complementary also belonging to the set

3. Logic operations: conjunction (logic AND) and disjunction (logic OR)

4. Logic operations are commutative, associative, and distributive

5. Special elements of the set are the unity (its value is always 1) and the zero (its value is always 0)

MANIPULATE EXPRESSIONS BOOLEAN MINIMIZATION

Need a way to manipulate expressions.
Rules of ‘adding’, ‘multiplying’ plus associative, distributive laws etc. The rules are very similar to basic algebra.

One can transform one Boolean expression into an equivalent expression by applying the postulates the theorems of Boolean algebra. This is important if one wants to convert a given expression to a *canonical form (a standardized form) or if one wants to minimize the* number of literals (asserted or negated variables) or terms in an expression. Minimizing terms and expressions can be important because electrical circuits often consist of individual components that implement each term or literal for a given expression. Minimizing the expression allows the designer to use fewer electrical components and, therefore, can reduce the cost of the system.

BOOLEAN THEOREMS

commutative law

$$A \bullet B = B \bullet A$$
$$A + B = B + A$$

associative law

$$A \bullet (B \bullet C) = (A \bullet B) \bullet C = A \bullet B \bullet C$$
$$A + (B + C) = (A + B) + C = A + B + C$$

distributive law

$$A \bullet (B + C) = A \bullet B + A \bullet C$$
$$A + (B \bullet C) = (A + B) \bullet (A + C)$$

The theorems above appear in pairs. Each pair form a *dual*.

35

DE MORGAN'S THEOREM

De Morgan's theorem occupies an important place in the logic (Boolean) algebra

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

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

In logic, De Morgan's laws (or theorem) are rules in formal logic relating pairs of dual logic operators in a systematic manner expressed in terms of negation. De Morgan's theorem may be applied to the negation of a disjunction or to the negation of a conjunction in all part of a formula.

36

DE MORGAN’S THEOREM

Negation of a disjunction

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

Since two things are false, it’s also false that either of them is true.

Negation of conjunction

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

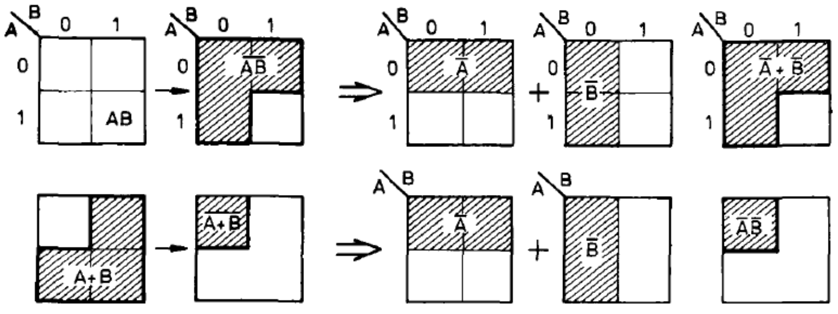
Since it is false that two things together are true, at last one of them should be false.

37

DE MORGAN’S THEOREM ON THE K-MAP

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

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



Representation of De Morgan’s theorem on the Karnaugh map or Veitch diagram.

38

DE MORGAN'S THEOREM

De Morgan's formulation of his theorem influenced the algebraization of logic undertaken by Boole, which cemented De Morgan's claim to the find, although a similar observation was made by Aristotle and was known to Greek and Medieval logicians, e.g. to William Ockham (1325), the great medieval scholastic philosopher.

In electrical engineering context the negation operator can be written as an overline (bar) above the terms to be negated.

In the originate the mnemonic

"break the line, change the operation"

39

BOOLEAN THEOREMS: DUALITY

The theorems above appear in pairs. Each pair form a *dual*. An important principle in the Boolean algebra system is that of *duality*. Any valid expression you can create using the postulates and theorems of Boolean algebra remains valid if you interchange the operators and constants appearing in the expression. Specifically, if one exchanges the • (AND) and + (OR) operators and swaps the 0 and 1 values in an expression, one will wind up with an expression that obeys all the rules of Boolean algebra. *This does not mean the dual expression computes the same values*, it only means that both expressions are legal in the Boolean algebra system. Therefore, this is an easy way to generate a second theorem for any fact one proves in the Boolean algebra system.

40

GENERALIZATIONS OF DE MORGAN'S THEOREMS

The De Morgan's theorem is an important tool in the analysis and synthesis of digital and logic circuits. Its generalization to several variables is stated below

$$\overline{A\,B\,C\,\ldots} = \overline{A} + \overline{B} + \overline{C} + \ldots$$

$$\overline{A + B + C + \ldots} = \overline{A}\,\overline{B}\,\overline{C}\,\ldots$$

41

SHANNON'S GENERALIZATION OF DE MORGAN'S THEOREMS

The De Morgan-Shannon's theorem refers to the logic or Boolean functions constructed using logic multiplications and additions

$$\overline{f(A, B, C, \ldots, +, \bullet)} = f(\overline{A}, \overline{B}, \overline{C}, \ldots, \bullet, +)$$

The negation of the function can be performed by negating each variable and replacing all logic summations (ORs) with logic multiplications (ANDs) and replacing all logic multiplications (ANDs) with logic summations (ORs).

42

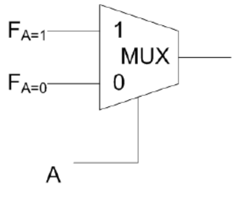
SHANNON'S EXPANSION THEOREMS

$F(X_1, X_2,\ldots X_n) = X_1\, F(1,X_2,\ldots X_n) + \overline{X_1}\, F(0,X_2,\ldots X_n)$

$F(X_1, X_2,\ldots X_n) = [X_1 + F(0,X_2,\ldots X_n)]\, [\overline{X_1} + F(1,X_2,\ldots X_n)]$

Application: decomposition of logic functions to smaller blocks, or for implementation using MUX based logic cells.
Example:

$$F(A,B,C,..) = A.F_{A=1} + \overline{A}.F_{A=0}$$



43

BOOLEAN FUNCTIONS

The one- and two-variable operations of Boolean algebra can be considered as functions of one and two variables, respectively.
In the case of generalized functions the number of variables is extended only.
n-variable Boolean or logic function

$$Z = f(X_1, X_2, \ldots\ldots\ldots X_n)$$

The particular truth value of Z is defined by the f function.

44

WHAT IS A BOOLEAN FUNCTION?

- The two-variable-operations of a Boolean algebra can be considered as functions of two variables.
- In case of generalised functions the number of variables is extended only.
- A $Z = f(X_1, X_2, \ldots X_i, \ldots, X_n)$ Boolean function of n-variables the values of
  - $X_i \in \{0,1\}$ and the function f defines the particular truth value of
  - $Z \in \{0,1\}$.

45

How many different logic function of n variables do exist?

The operations of two arguments are as many as $2^{2^2} = 16$

This is the number of arguments

In case of n variables 2^{2^n}

n=3 → 256

n=4 → 67 840

n=5 → 602 265 600

If the number of variables is increasing, the number of ddifferent logic functions is growing very fast.

46

CLASSIFICATION OF BOOLEAN FUNCTIONS OF TWO VARIABLES

Name of the function	f(A,B)
Logical constants	0, 1
Functions of one variable	A, $\overline{A}$, B, $\overline{B}$
AND, OR, NAND, NOR	$A \bullet B$, $A+B$, $\overline{A \bullet B}$, $\overline{A+B}$
XOR ($A \oplus B$), XNOR ($A \odot B$)	$\overline{A}\,B + A\,\overline{B}$, $\overline{A}\,\overline{B} + A\,B$
INHIBITION	$A \supset B$, $B \supset A$
IMPLICATION	$A \rightarrow B$, $B \rightarrow A$

47

BOOLEAN FUNCTIONS OF TWO VARIABLE

A B	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

0 AND A B OR NOR $\overline{B}$ $\overline{A}$ NAND 1

One variable

One variable

Boolean constants

48

THE 16 POSSIBLE BOLEAN FUNCTINS OF TWO VARIABLES

Function #	Description
0	Zero or Clear. Always returns zero regardless of A and B input values.
1	Logical NOR (NOT (A OR B)) = (A+B)'
2	Inhibition = BA' (B, not A). Also equivalent to B>A or A < B.
3	NOT A. Ignores B and returns A'.
4	Inhibition = AB' (A, not B). Also equivalent to A>B or B<A.
5	NOT B. Returns B' and ignores A
6	Exclusive-or (XOR) = A ⊕ B. Also equivalent to A≠B.
7	Logical NAND (NOT (A AND B)) = (A• B)'
8	Logical AND = A• B. Returns A AND B.
9	Equivalence = (A = B). Also known as exclusive-NOR (not exclusive-or).
10	Copy B. Returns the value of B and ignores A's value.
11	Implication, B implies A = A + B'. (if B then A). Also equivalent to B >= A.
12	Copy A. Returns the value of A and ignores B's value.
13	Implication, A implies B = B + A' (if A then B). Also equivalent to A >= B.
14	Logical OR = A+B. Returns A OR B.
15	One or Set. Always returns one regardless of A and B input values.

49

BOOLEAN FUNCTIOS AND OPERATIONS OF TWO VARIABLES: A SUMMARY

From the 16 possible two-variable Boolean functions

6 can be considered as trivial
(2 of them are constants, 4 of them are in fact one-variable functions)

From the 10 non-trivial functions
2 (AND and OR) and their complementary (AND-NOT and OR-NOT)
as well as the EXCLUSIVE-OR (antivalency)
and the EXCLUSIVE-NOR (equivalency)
are of significance for the practice.

50

IC IMPLEMENTATION: E.G. 74HC/HCT181

Total 16 arithmetic operations (add, subtract, plus, shift, plus 12 others)
- Total 16 logic operations (XOR, AND, NAND, NOR, OR, plus 11 others)
- Capable of active-high and active-low operation

FUNCTION TABLES

MODE SELECT INPUTS				ACTIVE HIGH INPUTS AND OUTPUTS	
S_3	S_2	S_1	S_0	LOGIC (M=H)	ARITHMETIC[2] (M=L; C_n=H)
L	L	L	L	$\overline{A}$	A
L	L	L	H	$\overline{A+B}$	A + B
L	L	H	L	$\overline{A}B$	$A+\overline{B}$
L	L	H	H	logical 0	minus 1
L	H	L	L	$\overline{AB}$	A plus $A\overline{B}$
L	H	L	H	$\overline{B}$	(A + B) plus $A\overline{B}$
L	H	H	L	$A \oplus B$	A minus B minus 1
L	H	H	H	$A\overline{B}$	$A\overline{B}$ minus 1
H	L	L	L	$\overline{A}+B$	A plus AB
H	L	L	H	$\overline{A \oplus B}$	A plus B
H	L	H	L	B	($A+\overline{B}$) plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	logical 1	A plus A[1]
H	H	L	H	$A+\overline{B}$	(A + B) plus A
H	H	H	L	A + B	($A+\overline{B}$) plus A
H	H	H	H	A	A minus 1

Notes to the function tables

1. Each bit is shifted to the next more significant position.
2. Arithmetic operations expressed in 2s complement notation.

MODE SELECT INPUTS				ACTIVE LOW INPUTS AND OUTPUTS	
S_3	S_2	S_1	S_0	LOGIC (M=H)	ARITHMETIC[2] (M=L; C_n=L)
L	L	L	L	$\overline{A}$	A minus 1
L	L	L	H	$\overline{AB}$	AB minus 1
L	L	H	L	$\overline{A}+B$	$A\overline{B}$ minus 1
L	L	H	H	logical 1	minus 1
L	H	L	L	$\overline{A+B}$	A plus ($A+\overline{B}$)
L	H	L	H	$\overline{B}$	AB plus ($A+\overline{B}$)
L	H	H	L	$\overline{A \oplus B}$	A minus B minus 1
L	H	H	H	$A+\overline{B}$	$A+\overline{B}$
H	L	L	L	$\overline{A}B$	A plus (A + B)
H	L	L	H	$A \oplus B$	A plus B
H	L	H	L	B	$A\overline{B}$ plus (A + B)
H	L	H	H	A + B	A + B
H	H	L	L	logical 0	A plus A[1]
H	H	L	H	$A\overline{B}$	AB plus A
H	H	H	L	AB	$A\overline{B}$ plus A
H	H	H	H	A	A

Notes to the function tables

1. Each bit is shifted to the next more significant position.
2. Arithmetic operations expressed in 2s complement notation.

4-bit arithmetic logic unit 74HC/HCT181

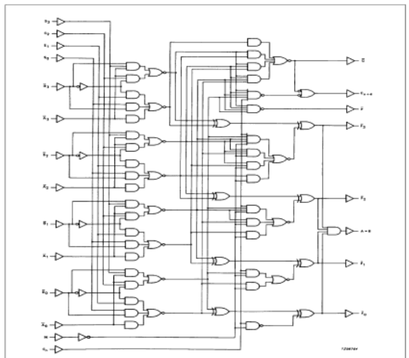


Fig 6 Logic diagram

WAYS TO DEFINE LOGIC FUNCTIONS

How to define a logic function?

There are several possible ways to do it.

Truth table or its special forms, like

- characteristic number
- canonical (algebraic) forms

(These are unique definitions – with some conditions)

Developed in 1854 by George Boole

Further developed by Claude Shannon (Bell Labs)

Outputs are computed for all possible input combinations

Graphic representation with Karnaugh map

(This one is unique too)

So called algebraic form, which is not unique.

Some appropriate function-definition language, like VHDL

DEFINITION OF A LOGIC FUNCTIONS BY ITS 1 VALUES

List the indices of those variable combinations to which a 1 value of the function belongs to.

This kind of definition also needs the agreement on the weights of the variables.

This definition is known as (extended) SOP (sum-of-products) or disjunctive canonical form.

This definition is also unique similarly to the characteristic number.

53

The concept of minterm and its algebraic form

- The minterm is a special „n” variable logic function, whose characteristic number contains a single 1 only.
- The Hamming weight of a minterm’s characteristic number is always 1
- It is an „n” „literal” logic product, in which
  - all of the „n” variables occur
  - either in ponated or in negated form.

THE DISJUNTIVE CANONICAL FORM

- The minterms of all n variable functions are n literal (elementary) products.
- Any f (logic) function of n variables can be defined by a sum of n variable minterms (SOP)
- This is unambiguous only iff the weighig of the independent variables is given.
- Since logic summarising is disjunction, for this form is known as Disjunctive Canonical Form of logic function *f*. (Unique.)
  Also known as SOP.

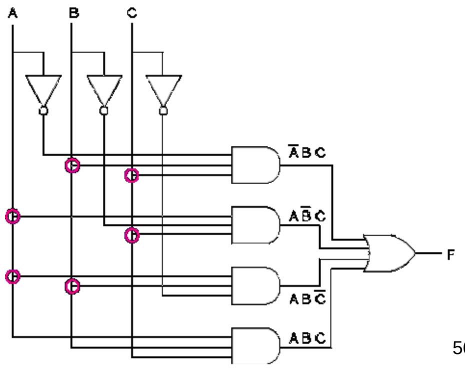
55

EXAMPLE: SOP FORM OF THE MAJORITY GATE

- The (extended) SOP form for the 3-input majority gate is:
- M = m3 + m5 +m6 +m7 = $\Sigma^3(3, 5, 6, 7)$
- Each of the 2^n terms are called minterms, running from 0 to $2^n - 1$
- Note the relationship between minterm number and Boolean value.

Two-level AND-OR implementation:

CMOS (SSI) implementation: 4062 logic dual majority gate



56

THE CONJUNCTIVE CANONICAL FORM (EXTENDED) PRODUCT-OF-SUMS

There exist an other canonical form, the conjunctive canonical form, or the (extended) product-of-sums (POS) form.
This is also unique.

The two canonical forms are equivalent and can be transformed into each other using the De Morgan theorem.

57

CONVERSION OF CANONICAL FORMS

Demo: SOP to POS conversion

$$F(ABC) = \Sigma^3(2,3,4,6)$$

1st step: obtain the minterm indices of the negated function (0-s in the truth table)

$$\overline{F}(ABC) = \Sigma^3(0,1,5,7)$$

2nd step: to obtain the maxterm indices complement the minterm indices of the negated function

$$F(ABC) = \Pi^3(0,2,6,7)$$

58

DEFINITION OF A LOGIC FUNCTION BY ITS MINTERMS

Example: 1-bit full adder

i	A_i (4)	B_i (2)	C_{i-1} (1)	S_i	C_i
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

$$S_i = (1,2,4,7)$$

$$C_i = (3,5,6,7)$$

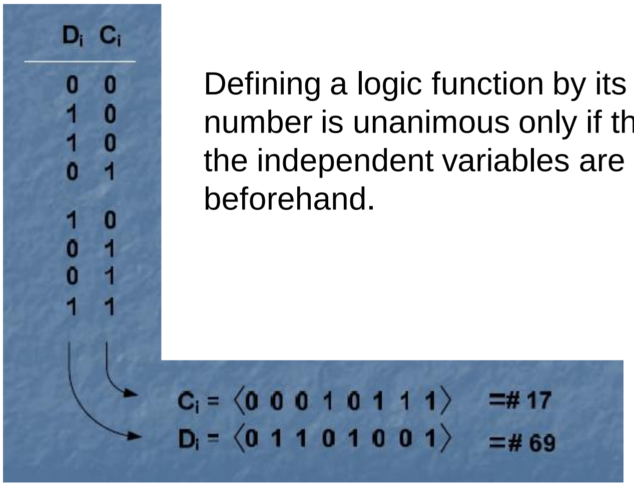
$$S_i = m_1^3 + m_2^3 + m_4^3 + m_7^3 =$$

$$= \Sigma^3(1,2,4,7)$$

E. g. $m_2^3 = \bar{A}\, B\, \bar{C}$, etc.

59

DEFINITION OF THE FUNCTION BY CHARACTERISTIC NUMBER



Defining a logic function by its characteristic number is unanimous only if the weights of the independent variables are fixed beforehand.

60

THE DISJUNCTIVE CANONICAL FORM (EXPANDED) SUM-OF-PRODUCTS

The minterms of all n-variable functions are n-literal products

Any logic function of n-variables can be defines by a sum of n-variable minterms (sum of products, SOP)

This is unambiguous only if the weighing of the independent variables is given.

Since logic summing is disjunction, this form is also known as disjunctive canonical form of a logic function.

The corresponding SOP is termed as expanded sum-of-products form

61

EXAMPLE: THE TWO CANONICAL FORMS OF XOR

The (trivial) extended SOP form (disjunctive canonical form) of the XOR function (F(A,B) = A ⊕ B is

$$\mathbf{F = \overline{A}\,B + A\,\overline{B} = \Sigma^2\,(1,2)}$$

The extended POS form (conjunctive canonical form) of the same function is

$$\mathbf{F = (\overline{A} + \overline{B})\,(A + B) = \Pi^2\,(0,3)}$$

62

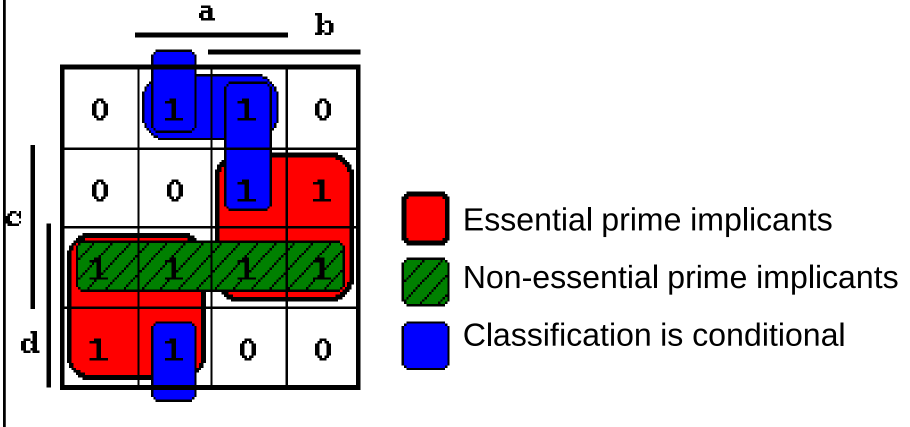
CANONICAL FORMS, MINTERMS, AND PRIME IMPLICANTS

The ”standard sum-of-products form” is also known as the ”minterm canonical form” or ”canonic sum function”. It is a ”sum” (OR) of minterms.

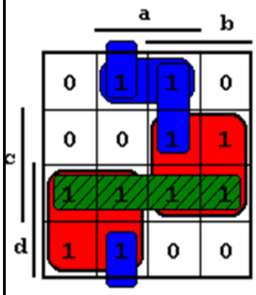
A minterm is also known as the ”standard product” or ”canonic product term”. This is a term where each variable is used once and once only.

A ”prime implicant” is an implicant of a function which does not imply any other implicant of the function.

PRIME IMPLICANTS: CLASSIFICATION



PRIME IMPLICANTS TABLE



	1	3	6	7	8	9	12	13	14	15
6, 7,14,15			x	x					x	x
8,9,12,13					x	x	x	x		
12,13,14,15							x	x	x	x
1,3	x	x								
1,9	x					x				
3,7		x		x						

65

REVISION QUESTIONS

1. Describe and discuss the fundamental properties of combinational logic circuits.

2. State and interpret DeMorgan’s theorem.

3. State and interpret Shannon’s extension of DeMorgan’s theorem.

4. Interpret and explain the following concepts:
(standard/extended) sum-of-product form, also known as (minterm/disjunctive) canonical form
(standard/extended) product-of-sum form, also known as (maxterm/conjunctive) canonical form.
Prime imnplicants: essential and non-essential

REVISION QUESTIOS

4. What is the function of a (1-bit) full adder? Write down the truth table showing all the possible input and output conditions.

Show that the sum (**S**) function of the one-bit full adder can be expressed in terms of the three inputs (**A**, **B**, and $\mathbf{C_{in}}$) and of the carry out ($\mathbf{C_{out}}$) as

$$\mathbf{S = (A + B + C_{in})\overline{C_{out}} + A\,B\,C_{in}}$$

What is the function of a (1-bit) half adder? How is it different from a full adder?

PROBLEMS AND EXERCISES

1. Find the canonic algebraic forms of the logic function

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

2. List the minterm and maxterm indices of the logic function below

$$\mathbf{F(A\,B\,C) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,C}$$

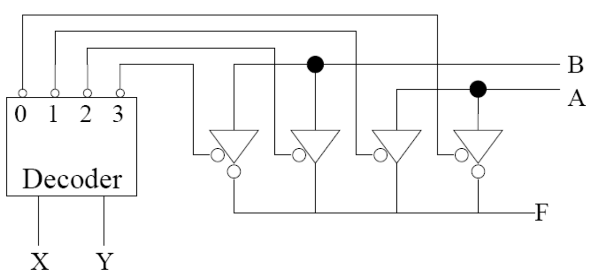
PROBLEMS AND EXERCISES

3. Draw the diagram of the implementation of the 1-bit full adder using

3.1. Homogenous NOR circuit.
3.2. 4-to-1 multiplexers and additional simple gates as necessary.
3.3. Transistor level diagram in CMOS architecture (28 transistors).

PROBLEMS AND EXERCISES

4. Determine the function F(A,B,X,Y) in algebraic form, realized by the following network.



THE END

71