Computer Basics (COMPB 119) notes

Bachelor in Computer Science and Engineering

First year - First semester

UNIVERSIDAD AUTONOMA DE MADRID

Contents

1 Boolean Algebra and Logic Design

1.1 Analog vs. digital systems

> **Definition 1.1.** An analog system is the one that works with analog signals. In this case, physical values are represented using analog signals, which are just infinite posible real values.

> **Definition 1.2.** A digital system is the one that works with digital signals. Physical values are represented using digital signals that can take two possible values, which both are exclusive. A digital signal takes discrete values, in general, only 2.

Computers can't interpret and operate with analog signals, then, we should convert them to digital signals using an ADC (Analog to digital converter). To achieve this we can use the **binary numeral system** (discussed in the next section) to represent any number just with using two posible values.

1.2 Positional numeral systems

> **Definition 1.3.** A positional numeral system is a system for representation of numbers by an ordered set of numeral symbols called digits whose value depends on its position in the actual number. For each position a unique symbol or a limited set of symbols is used.

> **Definition 1.4.** In a positional numeral system, the radix, or base, is the number of unique digits, including the digit zero, used to represent numbers.

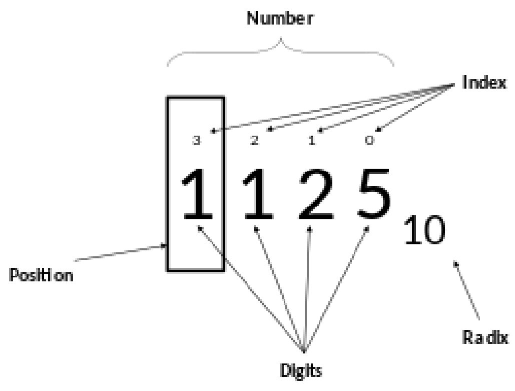


Figure 1: *Glossary of terms used in the positional numeral systems.*

In any standart positional numeral system, a number is conventionally written as $(x)_y$ with x as the string of digits and y as its base, although for the base 10 the subscript is usually assumed and omitted together with the pair of parenthesis.

Examples of commonly used positional numeral systems include the **decimal system** (base 10), the **binary system** (base 2) and the **hexadecimal system** (base 16).

Definition 1.5. In a positional numeral system with base b $(b > 1)$, a string of digits $d_1 \dots d_n$ denotes the number $d_1 b^{n-1} + d_2 b^{n-2} + \dots + d_n b^0$, where $0 \leq d_i \leq b$.

In contrast to decimal, or base 10, which as a ones' place, tens' place, hundreds' place, and so on, base b would have a ones' place, then a b^1s' place, a b^2s' place, etc.

Proposition 1.6. A number with n digits with base b can represent b^n different numbers in the range $[0, b^n - 1]$.

1.2.1 Decimal numeral system

Definition 1.7. The decimal numeral system is a positional numeral system with base 10, meaning that it uses 10 digits from 0 to 9.

In decimal number system, the succesive positions to the left of the decimal point represents units, tens, hundreds, thousands and so on.

Each position represents a specific power of the base (10). For example, the decimal number 5374 consists of the digit 4 in the units position, 7 in the tens position, 3 in the hundreds position, and 5 in the thousands position, and by definition 1.5 its value can be written as

$$5 \cdot 10^3 + 3 \cdot 10^2 + 7 \cdot 10^1 + 4 \cdot 10^0 = 5000 + 300 + 70 + 4 = 5374 \tag{1.1}$$

Proposition 1.8. A number with n decimal digits from 0 to 9 can represent 10^n different numbers in the range $[0, 10^n - 1]$.

For example, when $n = 3$, $10^3 = 1000$ different numbers can be represented within the range $[0, 999]$.

1.2.2 Binary numeral system

Definition 1.9. The binary numeral system is a positional numeral system with base 2, which uses only two symbols: typically 0 and 1.

Each binary digit is referred to as a **bit**. A binary number made up of 4 binary digits, or 4 bits, receives the name **nibble**, and a group of two nibbles, or eight bits, defines what is called a **byte**.

Proposition 1.10. A number with n binary digits, 0 and 1, can represent 2^n different numbers in the range $[0, 2^n - 1]$.

For instance, when $n = 3$, $2^3 = 8$ different numbers can be represented within the range $[0, 7]$.

Because of its straightforward implementation in digital electronic circuitry using logic gates, the binary system is used by almost all modern computers and computer-based devices.

Example 1.1. Convert 10101_2 from binary to decimal.
By definition 1.5 we get

$$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 1^0 = 16 + 4 + 1 = 21 \tag{1.2}$$

Example 1.2. Convert 47_{10} from decimal to binary.

$$\begin{aligned} 47 \quad \% \quad 2 = 1 \quad &\rightarrow \quad 23 \quad \% \quad 2 = 1 \quad \rightarrow && (1.3)\\ &\rightarrow \quad 11 \quad \% \quad 2 = 1 \quad \rightarrow && (1.4)\\ &\rightarrow \quad 5 \quad \% \quad 2 = 1 \quad \rightarrow && (1.5)\\ &\rightarrow \quad 2 \quad \% \quad 2 = 0 \quad \rightarrow \quad 2/2 = 1 && (1.6) \end{aligned}$$

Putting the last quotient and the remainders in inverse order we get that 47_{10} is 101111_2 in binary.

1.2.3 Hexadecimal numeral system

Definition 1.11. The hexadecimal system is a positional numeral system with base 16, commonly used to abbreviate binary numbers.

The different equivalences in the hexadecimal, decimal and binary systems are shown in the following table.

Hexadecimal digit	Decimal equivalent	Binary equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Table 1: *Hexadecimal, decimal and binary equivalences.*

Remark. Note that every hexadecimal digit is a nibble (it's made up of 4 bits).

Example 1.3. Convert $4AF_{16}$ (or $0x4AF$) from hexadecimal to binary.
By looking at table 1 we know that

$$4 = 0100 \qquad A = 1010 \qquad F = 1111$$

Then, the hexadecimal number $4AF$ is 010010101111 in binary.

Example 1.4. Convert $0x4AF$ from hexadecimal to decimal.
By definition 1.5 between number systems can be do the following operation:

$$\begin{aligned} 4AF &= 4 \cdot 16^2 + 10 \cdot 16^1 + 15 \cdot 16^0 = 4 \cdot 2^8 + 160 + 15 \\ &= 4 \cdot 256 + 160 + 15 = 1199 \end{aligned}$$

1.3 Axioms, theorems and properties of Boolean algebra

Boolean algebra is a mathematical tool used for analysis and synthesis of binary digital systems. It's named after George Bool, a british mathematician from the 19^{th} century.

Definition 1.12. A boolean variable is a digital signal which only has 1 out of 2 possible values in an instant, which are mutually exclusive. Those values are usually represented as 0 and 1, but another representation could be OFF and ON, or HIGH and LOW.

Definition 1.13. Let A be a Boolean variable. We call A the **true form** of the variable and $\overline{A}$, its inverse, the **complementary form**. $\overline{A}$ is the complement of A.

Note. The "true form" of a variable does not mean that A is true, but merely that A does not have a line over it.

Name	**Property**	**Dual**
Indentity	$B \cdot 1 = B$	$B + 0 = B$
Annulment	$B \cdot 0 = 0$	$B + 1 = 1$
Idempotent	$B \cdot B = B$	$B + B = B$
Double negation	$\overline{\overline{B}} = B$	$\overline{\overline{B}}$
Complement	$B \cdot \overline{B} = 0$	$B + \overline{B} = 1$
Commutative	$B \cdot C = C \cdot B$	$B + C = C + B$
Associative	$(B \cdot C) \cdot D = B \cdot (C \cdot D)$	$(B + C) + D = B + (C + D)$
Distributive	$(B \cdot C) + (B \cdot D) = B \cdot (C + D)$	$(B + C) \cdot (B + D) = B + (C \cdot D)$

Table 2: *Basic properties of boolean algebra.*

Definition 1.14. (De Morgan's laws). De Morgan's laws let us the operations in the expression we are working with.

$$\left(\overline{B_0 \cdot B_1 \cdot \ldots \cdot B_{n-1}}\right) = \left(\overline{B_0} + \overline{B_1} + \ldots + \overline{B_{n-1}}\right) \tag{1.7}$$

$$\left(\overline{B_0 + B_1 + \ldots + B_{n-1}}\right) = \left(\overline{B_0} \cdot \overline{B_1} \cdot \ldots \cdot \overline{B_{n-1}}\right) \tag{1.8}$$

1.4 Basic Boolean algebraic operations and logic gates

Definition 1.15. Let A and B be two Boolean variables

Definition 1.16. A logic function is a circuit accepting at least one input logical value and outputs a logical value.

In order to describe the working principle of a logic function a **truth table** is used. In this table is specified all the posible outputs for all possible input logic values. Also it's a graphical representation of all cases that can happen in an algebraic relation and its respective results.

A	**B**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: *Example of a truth table. In this case is a XOR logic gate.*

Definition 1.17. Logic gates are the implementation of most basic logic functions.

Amplifier / BUFFER gate

Definition 1.18. The amplifier, or most commonly named, the BUFFER, is the simplest logic gate. It expects one input (A) and gives one output value (Z).

It follows the logic equation $Z = A$, meaning the input received by the buffer is not changed at all, and its corresponding truth table is the following one.

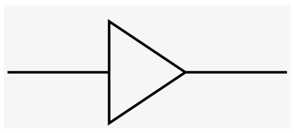


Figure 2: *Graphical representation of a BUFFER.*

A	**Z**
1	1
0	0

Table 4: *Truth table for a BUFFER.*

NOT gate / Inverter

Definition 1.19. The NOT gate, also called an **inverter**, gets one input value (A) and gives the opposite output value (Z), following the logic equation $Z = \overline{A}$.

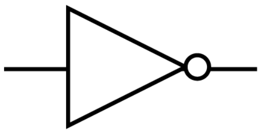


Figure 3: *Graphical representation of a NOT gate.*

A	**Z**
1	0
0	1

Table 5: *Truth table for a NOT gate.*

AND gate / Boolean multiplication

Definition 1.20. (AND gate / Logical AND). The AND gate gets at least two input values (A, B) and gives the one output value (Z). The output value Z will be true, $Z = 1$, if and only if both inputs A and B are simultaneously true.

The logic equation corresponding to the AND gate is $Z = A \cdot B$. Looking at this equation we could see the AND gate as a multiplication operation.

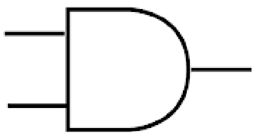


Figure 4: *Graphical representation of an AND gate.*

A	**B**	**Z**
0	0	0
0	1	0
1	0	0
1	1	1

Table 6: *Truth table for an AND gate.*

OR gate / Boolean addition

Definition 1.21. The OR gate gets a minimum of two input values (A, B) and gives the one output value (Z). The output value Z will be true, $Z = 1$, if at least one of the inputs A and B is true.

Then, the logic equation corresponding to the OR gate is $Z = A + B$. Looking at this equation we could see the OR gate as an addition operation.

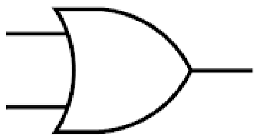


Figure 5: *Graphical representation of an OR gate.*

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

Table 7: *Truth table for an OR gate.*

NAND gate / Inverted AND gate

Definition 1.22. A NAND gate is a type of logic gate that gets at least two input values (A, B) and gives one output value (Z), which is 1 if at least one of the inputs is 0.

This logic gate can be though as a combination of an AND and a NOT gates, i.e. an inverted AND gate, and its corresponding logic equation is $Z = \overline{A \cdot B}$.

Note. $Z = \overline{A \cdot B}$ is not the same as $Z = \overline{A} \cdot \overline{B}$.

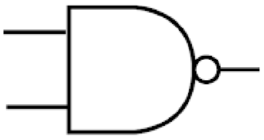


Figure 6: *Graphical representation of a NAND gate.*

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

Table 8: *Truth table for a NAND gate.*

NOR gate / Inverted OR gate

Definition 1.23. A NOR gate is a type of logic gate that gets at least two input values (A, B) and gives one output value (Z), which is 1 if both inputs are 0, and is 0 if at least one of the inputs is 1.

The corresponding logic equation for a NOR gate is $Z = \overline{A + B}$.

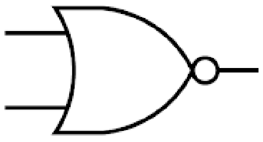


Figure 7: *Graphical representation of a NOR gate.*

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

Table 9: *Truth table for a NOR gate.*

XOR gate / OR-exclusive gate

Definition 1.24. A XOR gate, or OR-exclusive gate, is a type of logic gate that gets at least two input values (A, B) and gives one output value (Z), which is 1 if and only if one input is 1.

The XOR gate can be seen as a comparator of bits (if both bits are equal the output is 0); and its corresponding logic equation is $Z = A \oplus B$.

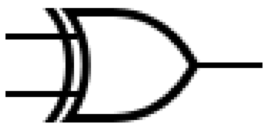


Figure 8: *Graphical representation of a XOR gate.*

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

Table 10: *Truth table for a XOR gate.*

XNOR gate / Inverted OR-exclusive gate

Definition 1.25. A XNOR gate is a type of logic gate that gets two input values (A, B) and gives one output (Z), which is 0 if and only if one input is 0.

The corresponding logic equation for a XNOR gate is $Z = \overline{A \oplus B}$.

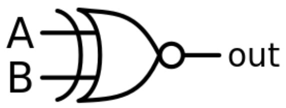


Figure 9: *Graphical representation of a XNOR gate.*

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	1

Table 11: *Truth table for a XNOR gate.*

1.5 Logic circuits and Boolean functions

Definition 1.26. (Logic circuit). A logic circuit is the combination of different logic gates by taking the output from one gate and using it as the input to another gate. It is composed of inputs, outputs, its functional specification and its temporal specification (delay)

Logic circuits enables computers to do more complex operations than they could accomplish with just a single gate.

Definition 1.27. (Combinational circuit). A combinational circuit is the one whose outputs depend only on the current values of the inputs; i.e. it combines the current input values to compute the output. For example, a logic gate is a combinational circuit.

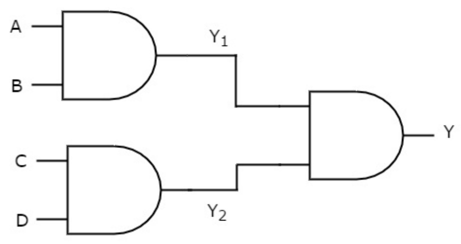


Figure 10: *Example of a logic circuit that takes four inputs (A, B, C, D) and gives one output (Y). In this case, the functional specification for this logic circuit is* $Y_1 = (A, B)$ $\quad Y_2 = (C, D)$ $\quad Y(Y_1, Y_2)$

Definition 1.28. (Sequential circuit). A sequential circuit is the one whose outputs depend on both current and previous values of the inputs; i.e. it depends on the input sequence. A combinational circuit is memoryless, but a sequential one has memory.

The functional specification of a logic circuit is usually expressed as a truth table or a Boolean function.

Definition 1.29. (Boolean function). A Boolean function is an expression that relates Boolean variables, in other words, it takes at least one variable (or literal) and gives and output value that can be either TRUE or FALSE.

Definition 1.30. (Implicant). The AND of one or more literals is called a **product** or an **implicant**.

Definition 1.31. (Minterm). A minterm is a product involving all of the inputs to the function.

For instance, $A\overline{B}C$ is a minterm for a function of the three variables A, B and C, but $\overline{A}B$ is not, because it doesn't involve C.

Definition 1.32. (Sum). Similarly, the OR of one or more literals is called a **sum**.

Definition 1.33. (Maxterm). A maxterm is a sum involving all of the inputs to the function.

For example, $A + \overline{B} + C$ is a maxterm for a function of the three variables A, B and C.

Note. Minterms and maxterms are dual concepts because of their complementary-symmetry relationship as expressed by De Morgan's laws.

1.5.1 Canonical disjunctive normal form (SOP form)

A truth table of n inputs contains 2^n rows, one for each possible value of the inputs. Each row in a truth table is associated with a minterm that is TRUE for that row, and only for that row. We can write a Boolean function for any truth table by summing each of the minterms for which the output is TRUE.

In table 12, for instance, the minterm for the first row is $\overline{A}\,\overline{B}$ because that expression is

TRUE when $A = 0$ and $B = 0$.

A	**B**	**Y**	**Minterm**	**Index**
0	0	0	$\overline{A}\,\overline{B}$	0
0	1	1	$\overline{A}B$	1
1	0	0	$A\overline{B}$	2
1	1	1	AB	3

Table 12: *Truth table with multiple TRUE minterms.*

For the truth table 12, taking the sum of each of the minterms (or rows) for which the output Y is TRUE gives $Y = \overline{A}B + AB$. This is called the **canonical disjunctive normal form**, or **sum-of-products (SOP) canonical form** of a function because it is the sum (OR) of products (ANDs forming minterms), and can be expressed as

$$Y = F(A,B) = \sum m_i,\ i \in \{1,3\} = \overline{A}B + AB \tag{1.9}$$

where m_i are the minterms to map.

Note. *Canonical form* is just a fancy word for standard form.

Note. The term SOP (Sum of Products) is widely used for the canonical form that its a disjunction (OR) of minterms.

The SOP form provides a Boolean function for any truth table with any number of variables. Unfortunately, SOP form doesn't necessarily generate the simplest function. We can use Boolean algebra properties and laws to simplify the expression or use Karnaugh maps to get the simplest one directly from the truth table.

1.5.2 Canonical conjunctive normal form (POS form)

An alternative way of expressing Boolean functions is the **canonical conjunctive normal form**, or **product-of-sums (POS) canonical form**. Each row of a truth table corresponds to a maxterm that is FALSE for that row. For example, the maxterm for the first row of a two-input truth table is $(A + B)$ because $(A + B)$ is FALSE when $A = 0$ and $B = 0$, and its expressed as

$$Y = F(A,B) = \prod M_i,\ i \in \{0\} = A + B \tag{1.10}$$

where M_i are the maxterms to map.

SOP produces the shortest functions when the output is TRUE on only a few rows of a truth table; POS is simpler when the output is FALSE on only a few rows of a truth table.

1.6 Karnaugh diagrams

Definition 1.34. (Karnaugh maps). Karnaugh maps (K-maps) are a graphical method for simplifying Boolean functions consisting on a sequence of cells where each one represents the binary value of an input. These cells are arranged so that the simplification of an expression consists on grouping adecuately the cells.

Proposition 1.35. For an expression of n variables 2^n cells are needed in the corresponding Karnaugh map.

Karnaugh maps make easier the planning of logic designs with simpler gate structure, leading to a more economic design; and can be used for expressions of up to 6 variables. For higher number of variables other methods such as the Quine-McClusky or CAD methods are used.

Definition 1.36. (Adjacent cells). Cells in a K-map whose values differ only in one variable. Adjacent cells have a Hamming distance of 1.

Cells on a K-map are disposed so that only one variable changes in between adjacent cells. Physically, each cell is adjacent to every inmediate next cell (4 sides).

Note. Cells cannot be adjacent diagonally.

Definition 1.37. (Cyclical adjacency). Upper cells are adjacent to its inmmediate lower cells and left cells are adjacent to its inmediate right.

Given the truth table for a function $Y = F(A, B, C)$ of variables A, B and C, the corresponding K-map is a matrix of 8 cells in which binary values of A and B are located on the left and C ones on the top.

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

$\longrightarrow$

$$\begin{array}{cc|c|c|}
 & & \multicolumn{2}{c}{C} \\
 & & 0 & 1 \\
\hline
 & 00 & 1 & 1 \\
\hline
 & 01 & 0 & 0 \\
\hline
AB & 11 & 1 & 0 \\
\hline
 & 10 & 1 & 0 \\
\hline
\end{array}$$

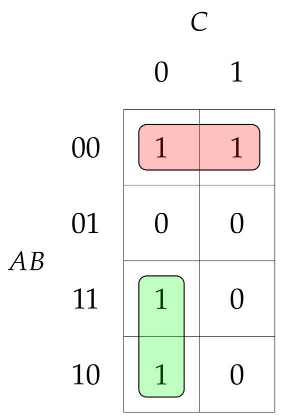
For instance, when $A = 0$, $B = 0$ and $C = 0$ the function has value 1, which corresponds in this case to the minterm $\overline{A} \cdot \overline{B} \cdot \overline{C}$.

1.6.1 SOP minimization through K-map

For every term of the SOP expression a 1 is added in the K-map cell correspondent to the product value. Then, taking the previous truth table,

A	B	C	Y	Minterm	Index
0	0	0	1	$\overline{A}\cdot\overline{B}\cdot\overline{C}$	0
0	0	1	1	$\overline{A}\cdot\overline{B}\cdot C$	1
0	1	0	0	$\overline{A}\cdot B\cdot\overline{C}$	2
0	1	1	0	$\overline{A}\cdot B\cdot C$	3
1	0	0	1	$A\cdot\overline{B}\cdot\overline{C}$	4
1	0	1	0	$A\cdot\overline{B}\cdot C$	5
1	1	0	1	$A\cdot B\cdot\overline{C}$	6
1	1	1	0	$A\cdot B\cdot C$	7

$\longrightarrow$



the SOP expression for the function described by the truth table is

$$Y=\sum m_i,\ i\in\{0,1,5,7\}=\overline{A}\cdot\overline{B}\cdot\overline{C}+\overline{A}\cdot\overline{B}\cdot C+A\cdot\overline{B}\cdot C+A\cdot B\cdot C \tag{1.11}$$

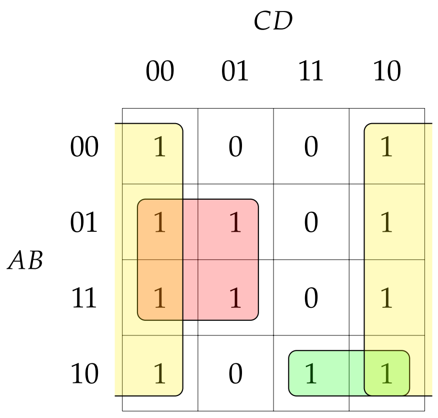
As one may see, this expression taken directly from the truth table is not simplified at all. In order to simplify the SOP expression after obtaining the K-map 3 steps should be followed:

1. **One's grouping.** A group has to contain 1, 2, 4, 8 or 16 cells, i.e. a power of 2 cells. For example, a K-map of 3 variables would have a maximum group of 8 cells.

   Each cell of a group has to be adjacent to one or more cells of the same group, but not all cells of the group must be adjacent to each other.

   We should maximize the number of 1s in each group.

   Each 1 of the map has to be included in, at least, one group. 1s already in a group can be included into other groups given that the overlaping groups contains non common 1s.



2. **Determination of each term of the SOP.** Each group of cells containing 1s gives one product term composed of all variables appearing in the group in only one form (complemented or non complemented).

   Variables complemented and non complemented inside the same group are deleted.

3. **SOP terms obtained.** Once obtained all minimum terms form the K-map they are added up to obtaine the minimal product expression.

> *Note.* In case we have a non-standard SOP with a term that lacks one or more variables to its expression it must be completed by counting all the possible terms.

1.6.2 POS minimization through K-map

The minimization process of a POS is basically the same as that for SOP although now the 0s are to be grouped to create the minimum number of sum terms. Rules applied to group 0s are the same as those to group 1s.

2 Combinational circuits