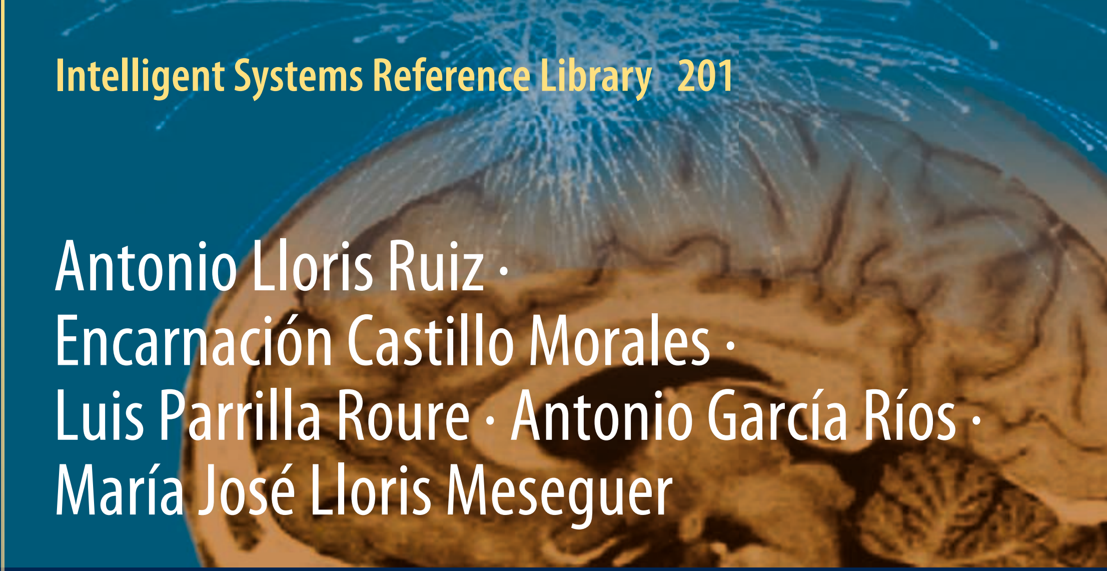
Intelligent Systems Reference Library 201



Antonio Lloris Ruiz · Encarnación Castillo Morales · Luis Parrilla Roure · Antonio García Ríos · María José Lloris Meseguer

Arithmetic and Algebraic Circuits

Springer

Intelligent Systems Reference Library

Volume 201

Series Editors

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland

Lakhmi C. Jain, KES International, Shoreham-by-Sea, UK

Antonio Lloris Ruiz ·
Encarnación Castillo Morales · Luis Parrilla Roure ·
Antonio García Ríos · María José Lloris Meseguer

Arithmetic and Algebraic Circuits

Springer

Antonio Lloris Ruiz
Departamento de Electrónica y
Tecnología de Computadores
Campus Universitario Fuentenueva
Universidad de Granada
Granada, Spain

Encarnación Castillo Morales
Departamento de Electrónica y
Tecnología de Computadores
Campus Universitario Fuentenueva
Universidad de Granada
Granada, Spain

Luis Parrilla Roure
Departamento de Electrónica y
Tecnología de Computadores
Campus Universitario Fuentenueva
Universidad de Granada
Granada, Spain

Antonio García Ríos
Departamento de Electrónica y
Tecnología de Computadores
Campus Universitario Fuentenueva
Universidad de Granada
Granada, Spain

María José Lloris Meseguer
Oficina Española de Patentes y Marcas
O.A. (OEPM), Madrid, Spain

ISSN 1868-4394 ISSN 1868-4408 (electronic)
Intelligent Systems Reference Library
ISBN 978-3-030-67265-2 ISBN 978-3-030-67266-9 (eBook)
https://doi.org/10.1007/978-3-030-67266-9

To our children and grandchildren

Julio

Lucía, Adriana and Pablo

José Luis and Sofía

Marina

Ana, Carmen and Jaime

who are the future

Prologue

Arithmetic Circuits are those digital circuits with inputs interpreted as numbers and whose outputs provide the results of some arithmetic operation over the inputs (addition, subtraction, multiplication, or division). These initial objectives (the elemental arithmetic operations) have been expanded so any mathematical function (trigonometrics, exponentials, logarithmics, etc.) is included as the purpose of the arithmetic circuits.

As a first definition, *Algebraic Circuits* are those digital circuits whose behaviour can be associated with any algebraic structure. Specifically, a polynomial is associated to each circuit, so that the evolution of the circuit will correspond to the algebraic properties of the polynomial. **LFSR**s (**L**inear **F**eedback **S**hift **R**egisters) and **CA**s (**C**ellular **A**utomata), included in this first denomination of algebraic circuits, are grouped under the name of *basic algebraic circuits.*

As a second definition, *Algebraic Circuits* are those digital circuits implementing the different operations within some algebraic structure. Specifically, in this book, this definition references to finite or Galois fields. The implementation of this *algebraic circuits* requires **LFSR**s and some basic arithmetic circuits.

This book is an expansion of our previous book *Algebraic Circuits*, including now arithmetic circuits, as both, arithmetic and algebraic, have many in common. Besides the addition of new material, each chapter includes a collection of exercises for didactic purposes.

The reader mainly interested in algebraic circuits will find the corresponding materials in Chaps. 1, 2, 3, 9, 10, 11, and 12; those interested only in arithmetic circuits may obviate Chaps. 3, 9, 10, 11. and 12.

Each chapter has been written as autonomous as possible from the rest of the book, thus avoiding back-consultation; this has the drawback of some redundancy, particularly of Chaps. 1 and 2 with those devoted to arithmetic circuits.

Chapter 1 is devoted to number systems, and a complete revision of the different representations of integer numbers is made, including redundant systems. The main procedures for the implementation of the fundamental arithmetic operations (addition, subtraction, multiplication, division, and square root) are also presented.

The implementation of those arithmetic circuits used for the construction of algebraic circuits is the purpose of Chap. 2. Addition, subtraction, multiplication, division (with special attention to modular reduction), and square root are implemented. Also, comparators and shifters, which can be considered to actually perform arithmetic operations but usually not considered as such, are described.

Chapter 3 deals with residue number systems, which are systems for numerical representation with interesting applications under the appropriate circumstances. Also, the Galois fields GF(p) are introduced in this chapter, since modular operations for prime values of p have to be implemented in GF(p).

Chapter 4 is mainly dedicated to the floating-point representation of real numbers, used profusely in arithmetic circuits. Rounding schemes and the IEEE 754 standard are presented, as well as circuit design to implement the main floating-point arithmetic operations. Also, the logarithmic system for real number representation is described.

Chapters 5–8 expand the basic arithmetic circuits presented in Chap. 2. As mentioned above, and in order to make each chapter autonomous, some redundance is allowed in these chapters.

Addition and subtraction are explored in detail in Chap. 5, specially all the questions associated to carry propagation for the construction of fast adders. Multioperand adders are described, to be used in the design of multipliers.

Multiplication is approached in Chap. 6, studying both combinational and sequential multipliers, as well as some special multipliers.

Division, the most complex of the basic arithmetic operations, is the object of Chap. 7. Division algorithms and their hardware implementations are sufficiently covered.

The computation of the most commonly used mathematical functions (logarithms, exponentials, trigonometrics, etc.) is the purpose of Chap. 8. The **CORDIC** algorithm is used as a general introduction to the procedures described in this chapter.

The basic algebraic circuits are the objective of Chap. 9. Regarding **LFSR**s, classic circuits, those storing a single bit in each cell, are introduced first. Then, they are generalized defining circuits with cells storing more than one bit each. The **CA**s studied in this chapter are mainly one-dimensional and linear, although two-dimensional **CA**s are also defined.

Chapter 10 is devoted to the Galois fields GF(2^{n}), presenting circuits to implement sums, products, divisions, squares, square roots, exponentiations, and inversions using power representation and the standard, normal, and dual basis. Also, the operations in the composite Galois fields GF($(2^{\mathrm{n}})^{\mathrm{m}}$) are detailed.

Chapter 11 is parallel to Chap. 10, but refers to Galois fields GF(p^n) and GF($(p^n)^m$).

Chapter 12 presents two very simple cryptographic applications of Galois fields: the first is based on the use of discrete logarithms, and as a real example, the Galois field GF(2^{233}) is used. The second is devoted to elliptic curves, and as a real example, the Galois field GF(2^{192}—2^{64}—1) is used.

All related mathematical fundamentals concerning Galois fields are divided into three appendices structuring everything that is used in the corresponding chapters, without any demonstration of most of the theorems and algorithms. The objective of these appendices is to provide an immediate source and to unify the nomenclature. Readers interested in in-depth details may use the indicated references. In Appendix A, the postulates and theorems about Galois fields are provided. Appendix B is devoted to the algebra of polynomials, paying particular attention to the different forms of representation. Appendix C includes all matters relating to elliptic curves used in the application examples developed in Chap. 12. Appendix D elaborates on errors, an important question when dealing with arithmetic circuits. Finally, Appendix E describes some important algorithms for function implementation, while Chebyshev and Legendre sequences of orthogonal polynomials are also presented.

Written as a self-contained text, this book is mainly intended as a practical reference for designers of hardware applications, but also may be used as textbook for courses on arithmetic and/or algebraic circuits. The exercises at the end of each chapter facilitate the practice of the corresponding concepts.

Contents

Chapter 1
Number Systems

This chapter is devoted primarily to study the various representations used for integers, both binary and decimal, but also mentioned the fixed point representation of fractional numbers. Next, we analyze the basic operations (addition, subtraction, multiplication and division) with integers, without going into implementation details. Chapter finishes studying the representation of integers using signed digits. The following chapter discusses in more detail these elementary operations, presenting different implementations.

1.1 Introduction

The use of numbering systems for keeping records of cereals or livestock, and enabling the transactions or exchanges between different groups, are part of the first signs of civilization in the various cultures that emerged at the dawn of mankind, when humans went from nomadic to sedentary and agriculture and livestock raise.

In the most primitive number systems, each entity to account (sheep, amount of wheat, soldier, etc.) is represented by a more or less idealized image. For example, to verify that the shepherd who is entrusted with a flock is loss free, a pebble can be placed in a bowl for each head of cattle. Thus, when the flock returns it can be easily checked that at least there are so many head of cattle as pebbles in the bowl. Something similar was done in certain cultures when a war expedition was initiated. At the departure, each man deposited a stone (or other representative object) in a receptacle, and at the return of the expedition, each soldier picked up a stone from the receptacle. The stones not collected corresponded to casualties in battle. Note that the meaning of *Calculus* in Latin is precisely *pebble*, referring to the use of these objects in elementary arithmetic.

Either way, in all civilizations the need of using specific representations for groups of entities arose. For example, a herd of 147 sheep can be represented by 147 clay balls. Now if you want to reduce the number of balls, it is possible to use balls of

A. Lloris Ruiz et al., *Arithmetic and Algebraic Circuits*, Intelligent Systems Reference Library 201, https://doi.org/10.1007/978-3-030-67266-9_1

different sizes or shapes, representing the same herd with a large ball, four medium and seven small. Thus arises, by accounting needs, numbering bases, which certainly were not introduced for their better or worse suitability to the needs of calculation (i.e., they were more or less suitable for different arithmetic operations), but only by convenience in representation.

Due to anthropomorphic reasons is logical to assume (as indeed happened) that the first bases used were 5 and 10, because always have been counted with the fingers, and in one hand we have 5 fingers and between the two we have 10. The same explanation seems to be valid for the use of the base 20 in widely separated civilizations if in this case are used as reference both hands and feet. Even the use of the base 12, currently held in certain contexts, is explained by anthropologists because the phalanges of the four fingers opposing the thumb are twelve; jerking a thumb each of the twelve phalanges, with one hand it is easy to count up to 12. What it lacks is a clear explanation from the Sumerian use of the base 60, use that we continue for measuring time and angles. The base 60 seems to correspond to the joint use of bases 5 and 12; with one hand we can count dozens, and with the other we can count up to five dozen; with two hands we can easily count to sixty.

1.1.1 Additional Notation

Currently the base used in complete generality is 10, sometimes coexisting, in certain contexts, with the bases 60 and 12. To represent the different quantities are no longer used pebbles or clay pellets, but the figures from 0 to 9, whose origin is in India and came to the western world by the Arabs, which is why it is known among us as Arabic numerals. Sometimes (for example, to number the front pages of some books) Roman numerals are used, in which numeric values are represented using letters of the alphabet: basically the symbols I (one), V (five), X (ten), L (fifty), C (hundred), D (five hundred) and M (thousand), with some modifications in order to represent large numbers.

The Roman numeral system, in a similar way than other civilizations, is based on the additional principle for representing numbers. As known, when representing a given value using Roman numbers, it is decomposed in additions or differences of a prefixed values (one, five, ten, ...). In this sense, the Roman numeration is an **additional notation**. Although the values to be added are ordered from the highest to lowest, the value represented by each symbol not depends on the position occupied in the number being represented. As an example, in the number MMCMXLVIII, each M takes the value 1000, and C $= 100$, but because it is at the left of the third M, this value must be subtracted. The same occurs with the X with relation with the L, and the three I have the value 1 each one, resulting the final value of 2948.

As it is well known, Roman numerals result very cumbersome for arithmetic operations which explains the underdevelopment of the calculus in this culture.

1.1.2 Positional Notation

The usual decimal number system employs the important concept of positional or relative value: it is a **positional notation**, where each figure represents different values depending on the position occupied. As an example, in number 6362′65, the first 6 has the value 6000, the second 60, and the third 0’6, according to:

$$6362'65 = 6 \times 10^3 + 3 \times 10^2 + 6 \times 10^1 + 2 \times 10^0 + 6 \times 10^{-1} + 5 \times 10^{-2}$$

This form of representation makes arithmetic operations easier, improving the representation of large numbers.

While not always aware of it, in our daily life we use numbering systems handling simultaneously different bases. For example, the time taken by our computer for performing a complex calculation can be measured in days, hours, minutes and seconds. For the minutes and seconds, base 60 is used, while base 24 for hours. In any case, the addition and subtraction of time intervals are easily performed. Thus, adding 2 days, 15 h, 36 min, 18 s to 3 days, 12 h, 25 min and 52 s we have:

2	15		36	18
3	12		25	53
5	27		61	71

resulting after appropriate reductions, 6 days, 4 h, 2 min, 11 s.

Obviously, during a long time, only natural number were used, then the concepts of negative numbers, rational number, irrational numbers and imaginary number were appearing. In this chapter are considered essentially different representations of integers.

1.2 Positional Notation Using One Base

In this section, the representations of unsigned numbers with positional notation are considered; in Sect. 1.4, signed number representation will be contemplated.

In general, any unsigned value ***N***, can be represented as a weighted sum of powers of another value, *b*, called base or radix, as follows:

$$N_b = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + \cdots + a_{-m} b^{-m} + \ldots \tag{1.1}$$

being N_b the representation of ***N*** in base *b*. If the base *b* is preset, N_b is given by the figures $a_n a_{n-1} \ldots a_1 a_0 a_{-1} \ldots a_{-m} \ldots$, and consists of an integer part $(a_n a_{n-1} \ldots a_1 a_0)$ and a fractional part $(a_{-1} \ldots a_{-m} \ldots)$, which can contains infinite figures. Until now, each figure in base *b* can take values in 0, 1, …, $b - 1$ (*b* different values), in which case the expression of $\boldsymbol{N}$ in base b is unique; that is, if $a_i \in \{0, 1, \ldots, b-1\}$, there is a unique expansion of $\boldsymbol{N}$ as a sum of powers of b.

Later, representations using positional notation in which each figure can take integer values in ranges different from $\{0, 1, \ldots, b-1\}$ will be considered. However, always will be assumed that 0 is among the values that each figure can take, and that the integer values that each figure can take are consecutives.

Independently of b, the integer part of N_b represents the part of $\boldsymbol{N}$ being greater or equal than unity, while the fractional part corresponds to the part of $\boldsymbol{N}$ being less than unity. Therefore, if the same number $\boldsymbol{N}$ is represented in two different bases, b_1 and b_2, and N_1, N_2 are the respective representations, the integer part of N_1 will be equal to the integer part of N_2, and the fractional part of N_1 will be equal to the fractional part of N_2.

When using positional notation without limiting the number of digits, any number can be represented. In fact, given an integer number N to be represented using radix b, n digits are required, being n the value resulting from:

$$b^{n-1} < \boldsymbol{N} + 1 \leq b^n$$

Conversely, if the number of digits, n, is set, the maximum integer value, N, that can be represented is given by:

$$\boldsymbol{N} = b^n - 1$$

In real systems for data storing and/or processing, always will be a limit in the number of available digits, thus, there will be a limited range for values representation.

The radix b can take any value: it can be positive, negative, integer, fractional, rational, irrational or imaginary. Nevertheless, the more reasonable selection consists on using natural numerals (the other options, in general, are not advantageous), then being the minimal value $b = 2$ ($b = 1$ obviously has no sense). Radix 2 is widely used in the computers world, as known.

1.2.1 Most Efficient Radix

Thinking about the values of b, it results logical the raising of the question if, under certain assumptions, there is a recommended value for b, i.e. if there is a most efficient basis, regardless of anthropological reasons.

The objective in the different chapters of this book is the designing of circuits implementing arithmetic and algebraic operations. These circuits, when computing a given operation between two digits, will be more complex as the value of b raises. The same is true for the memory elements needed for storing each digit. In other words, the cost $\boldsymbol{c_d}$ per digit, for both processing and storing information, is a function of b, $\boldsymbol{c_d} = \boldsymbol{f}(b)$. Assuming that $\boldsymbol{f}(b)$ is lineal [6]:

$$c_d = k_d b$$

Thus, for a number with n digits, the total cost will be:

$$C = k_d bn$$

If a range of M values is required for data processing, using radix b, n digits will be required, and:

$$M = b^n - 1 \quad \Rightarrow \quad n = \frac{\ln M}{\ln b}$$

thus:

$$C = k_d b \frac{\ln M}{\ln b} = K \frac{b}{\ln b} \quad (K = k_d \ln M)$$

The best radix b will be the one minimizing $\boldsymbol{C}$:

$$\frac{dC}{db} = k \frac{\ln b - 1}{(\ln b)^2} = 0 \quad \Rightarrow \quad \ln b = 1 \quad \Rightarrow \quad b = e$$

Then, the most efficient radix, assuming linearity for $\boldsymbol{f(b)}$, will be $\boldsymbol{e} = 2'718\ldots$ If an integer radix is desired, the recommended value is $b = 3$, because is the nearest integer to $\boldsymbol{e}$ number. Radix $b = 2$ results slightly less efficient, but has the advantage that electronic implementations in this base are the most reliable (it is easier to distinguish between two states instead of three states) and is the base used in computers.

1.2.2 Base Conversion

When using positional notation with only one radix, often there is a need of making a conversion from one base to another. Given the representation $\boldsymbol{N}_1$, of a number in base b_1, the goal is to obtain the representation $\boldsymbol{N}_2$ corresponding to the same number, but in terms of base b_2. For completing this conversion, the base b_1 or b_2, where the arithmetic operations for the conversion will be computed, must be chosen. Usually, one of the bases is 10, and it will be the preferred option. Thus, without entailing loss of generality, in what follows it is assumed that one of the bases implied in the conversion process is 10. Consequently, the problem of base conversion will be decomposed in two: the conversion from base 10 to any other base b, and from base b to base 10. First, the conversion from base 10 to base b is considered.

Given a number $\boldsymbol{N}_{10}$ represented in base 10, the issue consists on finding its representation $\boldsymbol{N}_b$ in another base b. In the base conversion, the integer part and the fractional part are processed separately. Starting with the integer part, if N_{10} is an integer number [i.e., the integer part of (1.1)], represented by the digits $a_n a_{n-1} \ldots a_1 a_0$, in base b, applying the Horner scheme it results:

$$\begin{aligned} \boldsymbol{N}_{10} = N_b = a_n b^n + a_{n-1} b^{n-1} + \ldots + a_1 b^1 + a_0 b^0 = \\ = (\ldots(a_2 b + a_{n-1})b + a_{n-2})b + \ldots + a_1)b + a_0 \end{aligned} \tag{1.2}$$

As known, when dividing a dividend D by a divisor d, a quotient C and a remainder r, are generated:

$$D = C \cdot d + r \tag{1.3}$$

Comparing (1.2) and (1.3), it can be concluded that a_0 is the remainder resulting from dividing $\boldsymbol{N}_{10}$ por b (the division is performed in base 10). If the quotient of this division is named $\boldsymbol{C}_0$, then:

$$C_0 = (\ldots((a_n b + a_{n-1})b + a_{n-2})b + \ldots + a_2)b + a_1,$$

resulting a_1 as the remainder of dividing $\boldsymbol{C}_0$ by b, and so on for the quotients C_1, C_2, etc., until the last possible division, for which the remainder is a_{n-1}, and the quotient will be a_n.

In other words, for converting the representation of an integer N in base 10 to base b, the initial value N is divided by b, and all the successive quotients. Then, the digits of N represented in base b are, from the most significant to the less significant, the last quotient followed by all the remainders, from the last one to the fist one. As an example, the decimal number 947 can be expressed using radix 6 as follows:

$$\begin{aligned} 947 &= 157 \times 6 + 5 \\ 157 &= 26 \times 6 + 1 \\ 26 &= 4 \times 6 + 2 \end{aligned}$$

resulting:

$$947_{10} = 4215_6$$

and, converting 947_{10} to binary representation, it results on:

$$947_{10} = 1110110011_2$$

We will consider now the fractional part. Given a fractional number N_{10}, its polynomial expression in base b is the fractional part of (1.1):

$$\boldsymbol{N}_{10} = \boldsymbol{N}_b = a_{-1} b^{-1} + a_{-2} b^{-2} + \cdots$$

Multiplying by b (in base 10) the two members of the equality, it results:

$$\boldsymbol{N}_{10}b = a_{-1} + a_{-2}b^{-1} + \cdots$$

Thus, a_{-1} is the integer part resulting from the multiplication of $\boldsymbol{N}_{10}$ by b. Subtracting a_{-1} from both of the members and multiplying again by b can be obtained a_{-2}:

$$(\boldsymbol{N}_{10}b - a_{-1})b = a_{-2} + a_{-3}b^{-1} + \cdots$$

and so on. While the result of the multiplication by b is nonzero, new digits are added to the fractional part. This process may no terminate, resulting infinite digits in the fractional part. Some examples of base conversion are the following:

$$0.743_{10} = 0.512564\ldots_7$$
$$0.6_{10} = 0.100100100\ldots_2$$

For the inverse conversion, from base b to base 10, whether if converting the integer part and the fractional part, results enough to make the operations implied by the polynomial development (1.1) of $\boldsymbol{N}_b$. As an example,

$$\begin{aligned}4,601.73_8 = 4 \times 8^3 + 6 \times 8^2 + 1 \times 8^0 + 7 \times 8^{-1} + 3 \times 8^{-2} = 4 \times 512 \\ + 6 \times 64 + 1 + 7 \times 0.125 \\ + 3 \times 0.015625 = 2433.921875_{10}\end{aligned}$$

$$101101.0101_2 = 2^5 + 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-4} = 45.3125_{10}$$

A radix conversion especially simple occurs when one of the radices is a power of the other, i.e., the conversion is performed between radix B and b being:

$$B = b^m$$

In this situation, each digit in base B corresponds to m digits in base b. Thus, the conversion may be completed digit by digit in B or by m-digit blocks in b. Equally simple results the case when the two radices are power of the same number. As an example, when converting between bases 4 and 8, both of them power of 2, base 2 may be used as intermediate stage:

$$2301.201_4 = 10\ 11\ 00\ 01.10\ 00\ 01_2 = 010\ 110\ 001.100\ 001_2 = 261.41_8$$

Some bases powers of two are considered in Sect. 1.2.3.

When considering fractional numbers (less than unity), like $3/7_{10}$, for converting to another base, (2 in this example), dividing by 7 is required, resulting a periodic decimal number, and later multiplying iteratively by 2. Nevertheless it results more convenient the interleaving of both operations, thus multiplying by 2 before dividing by 7, as follows:

$$\left\lfloor \frac{3}{7} \right\rfloor = 0, \quad \left\lfloor \frac{6}{7} \right\rfloor = 0, \quad \left\lfloor \frac{12}{7} \right\rfloor = 1$$

where $\lfloor x \rfloor$ is the greatest integer less than or equal to x. Taking into account that 12/7 − 7/7 = 5/7, this is the remainder after the first 1. Continuing with the remainder of the last division, multiplied by 2, it results:

$$\left\lfloor \frac{10}{7} \right\rfloor = 1, \quad \frac{10}{7} - \frac{7}{7} = \frac{3}{7}$$

Since 3/7 has appeared before, a periodic sequence is obtained. Finally, we have:

$$3/7_{10} = 0.011011\ldots_2$$

It is clear that the binary representation of any decimal fraction will be periodic. In fact, if the denominator is N, there will be $N - 1$ different remainders, and a repetition will be outlined in a maximum of $N - 1$ iterations.

1.2.3 Bases Power of Two

Computers are built using binary circuits, thus radix 2 is the natural option for operating in them. The **binary number system** (or binary numbers) is a positional system, where the different positions are power of 2:

$$\ldots .32\ 16\ 8\ 4\ 2\ 1.\ 0.5\ 0.25\ldots .$$

Each digit of a binary number can take the 0 and 1 values. Thus, it can be represented by a binary variable (as example, the state of a flip-flop). Also, it could be an option the use of radix −2, resulting the **negabinary number system**, advantageous in some situations, as will be outlined later. With this radix, the different positions take the value:

$$\ldots . -32\ 16\ -8\ 4\ -2\ 1. -0.5\ 0.25\ldots .$$

Arithmetic operations (addition, subtraction, multiplication and division) can be performed in binary using the corresponding tables, as shown in Table 1.1.

Table 1.1 (a) Addition table, (b) subtraction table, (c) multiplication table

A	Carry	*S*	Borrow	Multiplication
$0+0=0$	0	$0-0=0$	0	$0\times 0=0$
$0+1=1$	0	$0-1=1$	1	$0\times 1=0$
$1+0=1$	0	$1-0=1$	0	$1\times 0=0$
$1+1=0$	1	$1-1=0$	0	$1\times 1=1$
(a)		(b)		(c)

The following examples show the application of these tables when using positive operands, and considering positive results.

Addition example
The addition of decimal number 19 y 53 in binary is:

$$\begin{array}{rcccccccccl}
 & & & 1 & 0 & 1 & 1 & 1 & & \leftarrow \text{Carries} \\
19 & \rightarrow & & & 1 & 0 & 0 & 1 & 1 & \\
+53 & \rightarrow & + & 1 & 1 & 0 & 1 & 0 & 1 & \\
\hline
72 & \rightarrow & 1 & 0 & 0 & 1 & 0 & 0 & 0 &
\end{array}$$

Subtraction example
Subtracting 34 from 85 it results 51:

$$\begin{array}{rcccccccccl}
 & & 1 & 0 & 0 & 0 & 1 & 0 & & \leftarrow \text{Borrows} \\
85 & \rightarrow & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \\
-34 & \rightarrow & - & 1 & 0 & 0 & 0 & 1 & 0 & \\
\hline
51 & \rightarrow & 0 & 1 & 1 & 0 & 0 & 1 & 1 &
\end{array}$$

Multiplication example
Multiplying 25 by 13 it results 325:

$$\begin{array}{rcccccccccc}
25 & \rightarrow & & & & & 1 & 1 & 0 & 0 & 1 \\
\times 13 & \rightarrow & & & & \times & & 1 & 1 & 0 & 1 \\
\hline
 & & & & & & 1 & 1 & 0 & 0 & 1 \\
 & & & & & 0 & 0 & 0 & 0 & 0 & \\
 & & & & 1 & 1 & 0 & 0 & 1 & & \\
 & & & 1 & 1 & 0 & 0 & 1 & & & \\
\hline
325 & \rightarrow & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1
\end{array}$$

Division example
When dividing 437 by 38 it results a quotient of 11 and a remainder of 19:

```
437 →   1  1  0  1  1  0  1  0  1 | 1  0  0  1  1  0   ← 38
       −1  0  0  1  1  0            1  0  1  1          ← 11
       ------------------
           1  0  0  0  0  1  0
          −1  0  0  1  1  0
          ------------------
              1  1  1  0  0  1
             −1  0  0  1  1  0
             ------------------
19 →             1  0  0  1  1
```

Of these four arithmetic operations, the only essential is the sum, meaning that the other operations may be computed using algorithms based on sums. Another elemental operation implemented frequently in digital circuits is the comparison. Given two binary numbers of the same length, (in general, two characters A and B), this operation has to decide on the relative value of the binary representation of both: if $A > B$, or $A = B$, or $A < B$. In the next chapter the synthesis of these arithmetic units is studied.

When representing a magnitude in a computer, a limited number of bits is available, regardless of the chosen radix representation. The most extended and simple solution is the use of a fixed number n of bits for numeral representation. The finiteness in the number of bits involves limits on the numeric values that can be represented. When trying to represent a number that is outside these limits (i.e., a very large or very small) and the computer is not programmed to anticipate this contingency, an error known as **overflow** (very large) or **underflow** (very small) happens. Thus, when performing calculations on a computer it is important to consider the possible overflows or underflows, for any errors that may entail.

The binary expression of a numeral may get a large number of bits, resulting disadvantageous for handling. The octal and hexadecimal bases are used for obtaining more compact representations.

When using octal base, the symbols 0, 1, 2, 3, 4, 5, 6 and 7 are used. As $8 = 2^3$, one octal digit corresponds to three binary digits, and vice versa. Thus, converting from octal to binary (or binary to octal) results easy: only expanding (or compressing) the digits from the decimal point is required, adding zeros to the right if needed. As an example:

$$\begin{array}{cccccccccc}
46057.213_8 & = & 100 & 110 & 000 & 101 & 111' & 010 & 001 & 011_2 \\
 & & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow \\
 & & 4 & 6 & 0 & 5 & 7 & 2 & 1 & 3
\end{array}$$

$$\begin{array}{cccccccccc}
11 & 000 & 100 & 111 & 001. & 101 & 110 & 1_2 & = & 30471.564_8 \\
\leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & & \\
3 & 0 & 4 & 7 & 1 & 5 & 6 & 4 & &
\end{array}$$

In the last line, note that two zeros have been added implicitly to the right of the fractional part, resulting $100_2 = 4_8$.

When using radix 16 or hexadecimal (usually indicated by H sub index), the 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F symbols are used. Because $16 = 2^4$, one hexadecimal digit corresponds to four binary digits, and vice versa. Thus, for converting from hexadecimal to binary (or binary to hexadecimal) only expanding (or compressing, adding zeros to the left or the right, if needed) the digits, after the decimal point is required, as follows:

$$\begin{array}{ccccccccc}
3A57.2C3_H & = & 0011 & 1010 & 0101 & 0111' & 0010 & 1100 & 0011_2 \\
 & & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow \\
 & & 3 & A & 5 & 7 & 2 & C & 3
\end{array}$$

$$\begin{array}{ccccccccc}
1 & 0000 & 1000 & 1111. & 1001 & 1101 & 1_2 & = & 108H.9D8_H \\
\leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & \leftrightarrow & & \\
1 & 0 & 8 & H & 9 & D & 8 & &
\end{array}$$

Again, in this example, three zeros have been added to the fractional part of the binary number, resulting $1000_2 = 8_H$.

1.2.4 Modular Arithmetic

Sometimes, the operations with integer number are restricted to a limited range. In this situation the modular arithmetic may be interesting. Given a base or modulo M, $M > 0$, in the modular arithmetic, both the operands and the operations results are into the range $0 \le X < M$. Given an integer number N out of this range, its modular representation is derived by means of the modular reduction operation. The modular reduction consists on assigning to each integer N, the positive remainder resulting from dividing N by M. The result of the modular reduction is represented as N **mod** M (or also as $|N|_M$ or **mod** (N, M)). As an example, for 17, being reduced modulo 7 is 17 **mod** $7 = 3$, and for -3 modulo 5 is $-3\mathbf{mod}5 = 2$.

Given a modulo, M, and two integers, A and B, with modular representations r and s, respectively, we have:

$$\begin{aligned}
r = A \ \mathbf{mod}\ M \quad &\Rightarrow \quad \frac{A}{M} = C + \frac{r}{M} \\
s = B \ \mathbf{mod}\ M \quad &\Rightarrow \quad \frac{B}{M} = D + \frac{s}{M}
\end{aligned}$$

For the modular representation of the sum of A and B, $(A + B)$ **mod** M, we have:

$$\frac{A+B}{M} = (C + D) + \frac{r+s}{M}$$

Taking into account that $r + s$ may be greater than M, from the last equation it results:

$$(A + B)\,\mathbf{mod}\,M = (A\,\mathbf{mod}\,M + B\,\mathbf{mod}\,M)\,\mathbf{mod}\,M$$

For the modular representation of the product aA it results:

$$\frac{aA}{M} = aC + \frac{ar}{M}$$

Again, because ar may be greater than M, from this equality it results:

$$aA\,\mathbf{mod}\,M = \{a(A\,\mathbf{mod}\,M)\}\,\mathbf{mod}\,M$$

Applying all of these expressions to the development of an integer number N in positional notation as a sum of powers of the radix,

$$N = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b^1 + a_0 b^0$$

we have:

$$N\,\mathbf{mod}\,M = \big\{a_n\big(b^n\,\mathbf{mod}\,M\big) + a_{n-1}\big(b^{n-1}\,\mathbf{mod}\,M\big) + \cdots \\ + a_1\big(b^1\,\mathbf{mod}\,M\big) + a_0\big(b^0\,\mathbf{mod}\,M\big)\big\}\,\mathbf{mod}\,M$$

The modular reductions of the different powers of the radix can be pre-computed, being possible the simplification of the $N\mathbf{mod}M$ computing applying the following expression:

$$\left|a_n b^n + \cdots + a_0 b^0\right|_m = \left|a_n\left|b^n\right|_m + \cdots + a_0\left|b^0\right|_m\right|_m$$

Obviously, if M is a power of the base, $M = b^k$, the modular reduction of $N = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b^1 + a_0 b^0$ is trivial:

$$N\mathbf{Amod}\ M = a_{k-1} b^{k-1} + a_{k-2} b^{k-2} + \cdots + a_1 b^1 + a_0 b^0$$

Other procedure for performing the modular reduction (named **multiplicative modular reduction**) is based on state M as the difference: $M = b^k - a$, being $1 \leq a < b^{k-1}$. In this case, $N\mathbf{mod}M$ can be easily computed by means of successive multiplications by a, and modular reductions b^k, as it is shown in the following. Starting from:

$$N = c_0\, b^k + r_0$$

Multiplying c_0 (and the successive quotients) by a, and reducing modulo b^k, we obtain:

$$\begin{aligned} ac_0 &= c_1\, b^k + r_1 \\ ac_1 &= c_2\, b^k + r_2 \\ &\cdots \\ ac_{i-1} &= c_i\, b^k + r_i \end{aligned}$$

In each iteration, the result is multiplied by a and divided by b^k. Because $a < b^{k-1}$, after p iterations a zero quotient is obtained:

$$ac_{p-1} = 0\, b^k + r_p$$

Rearranging these equalities:

$$N = c_0\, b^k + r_0$$

$$\begin{aligned} 0 &= -ac_0 + c_1\, b^k + r_1 \\ 0 &= -ac_1 + c_2\, b^k + r_2 \\ 0 &= -ac_{i-1} + c_i\, b^k + r_j \\ &\cdots \\ 0 &= -ac_{p-1} + r_p \end{aligned}$$

Memberwise adding the equalities:

$$\begin{aligned} N = {} & c_0\big(b^k - a\big) + c_1\big(b^k - a\big) + \cdots + c_i\big(b^k - a\big) + \cdots + c_{p-1}\big(b^k - a\big) \\ & + r_0 + r_1 + \cdots + r_p \end{aligned}$$

And:

$$N\,\mathbf{mod}\,\big(b^k - a\big) = \big(r_0 + r_1 + \cdots + r_p\big)\,\mathbf{mod}\,\big(b^k - a\big)$$

Thus, the remainder modulo $b^k - a$ of N may be computed by multiplying iteratively by a, being N the first product. Each product is decomposed in two fragments, $A_i b^k + B_i$. Each $A_i \neq 0$ is the new multiplicand, and the calculus ends when $A_i = 0$. The remainder is $\Sigma B_i \mathbf{mod} (b^k - a)$.

In modular arithmetic, the addition, subtraction and multiplication (each of these three operations will be represented from now on with the symbol $\diamond$) of two integers is defined as follows. The division has sense only in some cases.

Given A, B and M ($0 \leq A < M$, $0 \leq B < M$), $A \diamond B \mathbf{mod} M$ is obtained by calculating $A \diamond B$ using normal arithmetic, and making a modular reduction to the result of $A \diamond B$.

Later, in Chap. 3, the modular arithmetic will be deeply studied when introducing the Residue Number System (**RNS**). Concretely, will be shown the properties derived when M is prime, being the possibility of defining division one of them.

1.2.5 Fractional Numbers: Fixed Point Representation

A fractional number consists of an integer part and a fractional part, both of them with a limited number of digits and separated by the named decimal mark (a point or a comma, or an apostrophe, depending on the country): 23.456, 0.0027 and 378.42196 are examples of fractional numbers. The real number can have a decimal development with infinite decimal digits (all of the irrational numbers and some rational). When a limited number of digits is used, like in digital systems, any real number will be represented as a fractional number.

For representing a fractional number N, besides the representation of the different digits, the representation (or indication) of the decimal mark position is required. In general, the number of the fractional digits can vary from a number N to another. Otherwise, if premised that the number of fractional digits is fixed for any number N, there is no need to indicate the decimal mark position. As an example, using 5 digits for the fractional part, the numbers 23.456, 0.0027 and 378.42196 will be represented as 2345600, 270 and 37842196. This idea of assigning always a fixed number of fractional digits, is the one used in the named **fixed point representation.**

If using positional notation with fixed point representation, each digit represents a different power of the radix and the decimal mark position can be indicated giving the power of the radix representing any digit. Specifically, the less significant digit is the one used for these task, defining the known as **unit in the last position** (*ulp*), i.e., which is the power of the radix corresponding to the less significant digit. If k digits are used for the fractional part, then $ulp = b^{-k}$. Obviously, *ulp* is also the difference between two consecutive representable values using k fractional digits. Thus, it can be used for measuring the **precision** that can be achieved with this representation. As a consequence, the difference between two consecutive values cannot be less than *ulp*.

When using a total of n digits (k for the fractional part and y $n - k$ for the integer part) for representing positive fractional number in positional notation in base b, the decimal value M of any represented number $d_{n-k-1} \ldots d_0 d_{-1} \ldots d_{-k}$ is:

$$M = \sum_{i=-k}^{n-k-1} d_i b^i$$

With k digits for the fractional part and $n - k$ for the integer part, the less positive representable value nonzero is b^{-k} (in this case, all of the digits are zero except the last, taking the 1 value), the maximum representable value is $b^{n-k} - b^{-k}$ (now, the n digits take the value $b - 1$); i.e.: