INTRODUCTION TO DIGITAL SYSTEMS

JAMES PALMER, Ph.D. DAVID PERLMAN

SCHAUM'S OUTLINE OF

THEORY AND PROBLEMS OF INTRODUCTION to DIGITAL SYSTEMS

●

JAMES E. PALMER, Ph.D.
Professor of Electrical Engineering
Rochester Institute of Technology

DAVID E. PERLMAN
Associate Professor of Electrical Engineering
Rochester Institute of Technology

SCHAUM'S OUTLINE SERIES
McGRAW-HILL

New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

JAMES E. PALMER is Professor of Electrical Engineering at the Rochester Institute of Technology, in Rochester, New York (R.I.T.). He received his B.Sc. from the University of Western Ontario, his M.S.E.E. from the University of Pennsylvania, and his Ph.D from Case Institute of Technology. His research interests are the design of digital systems with an accent on product design and its concurrent engineering aspects. From 1968 to 1974 he was Director of Engineering at Gannon University. From 1974 to 1978 he was Head of the Electrical Engineering Department at RIT. Presently he teaches courses in the areas of digital system design and test and also in the areas of control system design.

DAVID E. PERLMAN is an Associate Professor of Electrical Engineering at the Rochester Institute of Technology, in Rochester, New York. He received B.E.E. and M.E.E. degrees from Cornell University and, following ten years as a design engineer and researcher at the Eastman Kodak Company, he left to become one of the founders and Vice President of Advanced Development of Detection Systems, Inc., in Fairport, New York, a position he held for thirteen years. He holds twelve patents. In 1982, Mr. Perlman joined the faculty at R.I.T., where he has been teaching graduate and undergraduate courses in electronics as well as undergraduate courses in circuits and digital systems.

Schaum's Outline of Theory and Problems of
INTRODUCTION TO DIGITAL SYSTEMS

10 11 12 13 14 15 CUS/CUS 0 9 8 7 6 5

ISBN 0-07-048439-2

Sponsoring Editor: John Aliano
Production Supervisor: Leroy Young
Editing Supervisor: Meg Tobin
Front Matter Editor: Maureen Walker

Library of Congress Cataloging-in-Publication Data

Palmer, James (James E.)
Schaum's outline of theory and problems of introduction to digital systems / James Palmer and David Perlman.
p. c.m.—(Schaum's outline series)
Includes index.
ISBN 0-07-048439-2
1. Digital electronics. 2. Logic circuits. I. Perlman, David (David E.) II. Title. III. Series.
TK7868.D5P35 1993
621.381—dc20 91-46678
CIP

McGraw-Hill
A Division of The ***McGraw-Hill*** *Companies*

Preface

The goal of this book is to introduce a unified design methodology into the introductory course in digital systems. It is based on the course "Introduction to Digital Systems" which is offered to freshmen and incoming transfer students in the electrical engineering curriculum at the Rochester Institute of Technology.

As is usual in books on this subject, the first chapter describes number systems in general and the binary system in particular as a prelude to introducing the two-valued logical variable and signals which represent it in all computer and digital circuits.

The next three chapters describe a coherent design procedure for systems using combinatorial (or combinational) logic. Three different means of specifying a combinatorial problem—the truth table, Boolean equations, and logic diagrams—are discussed in Chapter 2, while Chapter 3 deals with the manipulations of Boolean algebra and contains additional material on the construction and interpretation of Karnaugh maps. Here, the design problem is analyzed at a purely logic level, independent of hardware considerations, and the relation between K maps, Boolean equations, and logic diagrams is explored. Chapter 4 presents a structured approach to the hardware implementation of logic using mixed-logic methodology. The result is a totally unambiguous design tool which yields functional logic circuitry while preserving the identity of the original underlying Boolean relations.

Chapter 5 offers a description of commonly used MSI and LSI combinatorial logic elements with emphasis placed on devices (such as multiplexers and ROMs) which can be programmed for specific applications.

The remainder of the book is primarily concerned with synchronous sequential logic. The construction and use of timing diagrams is developed in Chapter 6 where computer-aided design tools such as schematic capture and simulation software are introduced. The logical function of basic memory elements (flip-flops) is discussed in Chapter 7 and some important MSI and LSI combinations of flip-flops are covered in Chapter 8, which deals with registers, counters, and data storage devices. In Chapter 9, the basic operation of programmable devices containing both combinatorial logic and flip-flops is discussed. Chapter 10 illustrates both traditional design procedures and the use of algorithmic state machine charts as design tools for synchronous sequential logic and for simple state machines. Chapter 11 takes a nontraditional view of logic elements as control device components and provides an introduction to programmable gate arrays and their operation.

A word about symbols is appropriate here since, unfortunately, no single notation has achieved universal acceptance. While the bubble has been used for many years, in positive logic notation, to indicate logical inversion, it is also currently used as an alternative to the half-arrow to denote a low-TRUE signal in mixed logic systems. Since reserving the half-arrow to exclusively designate low-TRUE is less ambiguous, the authors have emphasized this notation for use in developing the unified design process presented in chapters 2–4. Bearing in mind, however, that most currently available schematic capture and simulation software packages produce bubbles and not half-arrows, we cannot arbitrarily banish the low-TRUE bubble. Its application is discussed in Section 4.3 and, it will be found scattered throughout the book (as in Figs. 4-76, 4-79 and 5-38) in order to present students with examples of the symbology that they are likely to come across in "the real world".

The situation is no less confused when it comes to denoting connections (or lack thereof) in programmable logic devices. In chapters 9 through 11, "x's", solid circles or solid rectangles are used to indicate connections while hollow circles or no symbol at all indicate the lack of a connection. Since, in all likelihood, the reader will come across all or several of the above conventions, it was decided to present a generous sprinkling of each in the examples and problems, taking care to avoid any ambiguity in the meaning of a symbol.

This book is designed to function as either a text for an introductory course in the design of digital systems or for use as a supplement to other textbooks. Typical of the Schaum's Outline Series, it contains numerous worked examples as well as supplementary problems with answers. It is important to note that in design there are often several valid solutions to a given problem. In these cases, the authors have used their best judgment in selecting the solution given and, when appropriate, have presented alternative approaches to a representative group of problems as new techniques are developed in successive chapters.

It will be noted that many of the logic and timing diagrams in this book have been computer generated and several generic observations concerning schematic capture and simulation appear in the text and in App. C. This is a natural consequence of the fact that it has become increasingly difficult to treat the design of digital systems without reference to Electronic Design Automation (EDA) software. The authors have chosen to use LogicWorks™, a somewhat scaled-down version of DesignWorks™ digital logic design software from Capilano Computing Systems Ltd., because it is an extremely user-friendly simulation package which is attractively priced for educators and students. Interested readers should write Capilano at 406-960 Quayside Drive, New Westminster, B.C., Canada V3M 6G2 or call (800) 444-9064 or (604) 522-6200 for further information and/or a demonstration disc.

The authors would like to express their appreciation for the helpful comments of Dr. Charles Schuler who reviewed the manuscript and kept our spirits up with constructive encouragement. We would also like to express our appreciation to several "generations" of undergraduate students at RIT who gave us invaluable feedback on the effectiveness of our pedagogy and the accuracy of problem solutions. In the not so distant past, the creation of hundreds of diagrams integrated with the text required the acknowledgment of gargantuan (and incredibly patient) effort by one or more harassed secretaries and/or artists. We need make no such mention here since the job was done "in house" with the aid of an Apple Macintosh II* computer running two screens—one for drawing and the other for text, which made the project reasonably manageable and usually enjoyable. We do, however, want to especially acknowledge the patience and support of our wives, Mary Palmer and Marjorie Lu Perlman, both of whom put up with a lot of lost cohabitation during the many late nights and weekends that went into this project.

JAMES E. PALMER
DAVID E. PERLMAN

* Apple and Macintosh are registered trademarks of Apple Computer, Inc.

Contents

Chapter 1

Numbers and the Binary System

1.1 INTRODUCTION

In modern digital systems it is necessary to electronically store and process large quantities of data in the presence of electrical noise and interfering signals. The data is usually in a binary (two valued) form since this allows the use of reliable and easily replicated storage and computational devices comprising large numbers of logically connected electronic switches fabricated within integrated circuits. Such devices, containing thousands (and in many cases, millions) of transistors, are inherently resistant to faults because the voltage or current levels representing the two binary states are far enough apart to prevent errors caused by spurious interference. Various data encoding and error checking schemes, such as Gray coding and parity checks, are often used to reduce the already low probability of undetected errors. Since the binary number system is universally employed in digital processing, it is useful to understand its relationship to other number systems, as well as the properties of number systems in general and the methods of conversion from one to another.

1.2 NUMBER SYSTEMS

In everyday use, numbers are represented in the decimal (base 10) system which has 10 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). This system is *weighted* in that it makes use of a *positional notation* wherein the value assigned to a particular digit is determined by its position in the sequence of digits which represents a given number. Consider the base 10 number 853828. The digit 8 occurs three times in the sequence, but each occurrence has a different weight because the digit occupies a different position corresponding to a power of the base. This arrangement is shown below.

10^5	10^4	10^3	10^2	10^1	10^0	Column weights
8	5	3	8	2	8	Digits

$$853828 = 8 \times 100{,}000 + 5 \times 10{,}000 + 3 \times 1000 + 8 \times 100 + 2 \times 10 + 8 \times 1$$

The left-most 8 is weighted by 10^5, the next 8 by 10^2, and the last by 10^0. This positional notation is easily extended to decimal *fractions*, in which case, negative powers of the base 10 are used:

$$0.725 = 7 \times 10^{-1} + 2 \times 10^{-2} + 5 \times 10^{-3}$$

The Binary System

It is possible to express a number in any base. In the binary case, the base is 2 and only two symbols are needed (0 and 1). Each digit is called a "bit" and, again, positional notation is used. To find the decimal equivalent of any binary number, merely write the decimal equivalent of each of the powers of 2, multiply by the appropriate binary digit, and add the results.

EXAMPLE 1.1 Express the binary number 1100111.1101 as a decimal (base 10) number.

Since the integer part has seven digits (bits), the most significant has a weight of 2^6 or 64. Its decimal equivalent may be easily computed as

$$\begin{aligned}1100111 &= 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0\\ &= 1 \times 64 + 1 \times 32 + 0 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1\\ &= 103_{10}\end{aligned}$$

For the decimal part,

$$\begin{aligned} .1101 &= 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\ &= 1 \times 0.5 + 1 \times 0.25 + 0 \times 0.125 + 1 \times 0.0625 \\ &= 0.8125_{10} \end{aligned}$$

Since binary numbers require only two symbols, they are ideally suited for representation by electronic devices since only two easily distinguishable states, such as ON and OFF (conducting and nonconducting), are required.

The advantages of binary are best illustrated by considering the effect of noise or interference on the performance of a data-processing system. In the binary case, when data is to be transmitted or retrieved from storage, it is necessary for the receiver to determine which of two levels a given signal is nearer. A threshold for decision can be set up midway between these two levels so that any additive noise which is less than the difference between the signal level and the threshold is ignored. With decimal storage on the other hand, a system with the same overall voltage range assigned to its signals has a much smaller noise immunity because the given range must be divided into 10 separate levels (see Fig. 1-1).

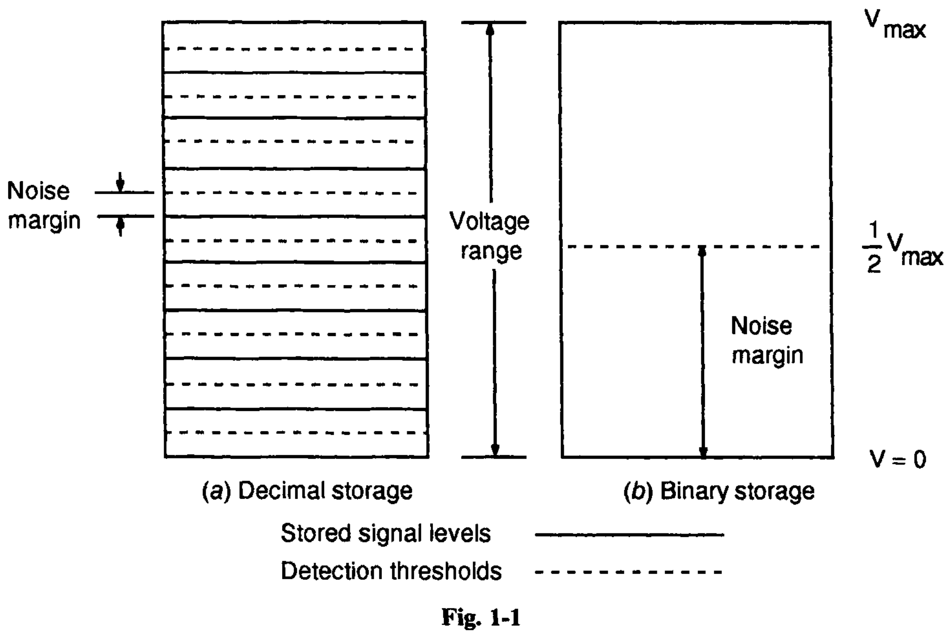


Fig. 1-1

In data systems, we speak of a figure of merit called *noise margin* which is defined as the maximum noise voltage (or current) which can be tolerated without causing an undesirable output change.

EXAMPLE 1.2 Compare the basic noise margins of binary and decimal data systems having ideal hardware components.

For the binary case, a 1 is stored as V_{max} [typically 5 volts (V)] and a 0 as approximately 0 V. The threshold would be set at $V_{max}/2$, and any noise less than this value is ignored. In the decimal system, there would be 10 equally spaced storage levels between 0 and V_{max} (0, $V_{max}/9$, 2 $V_{max}/9$, etc.) and there would be thresholds set up halfway between adjacent storage levels ($V_{max}/18$, 3 $V_{max}/18$, etc.). Any noise which is more than $V_{max}/18$ would result in an erroneous data reading. For the binary case with $V_{max} = 5$ V, the noise margin would be $5/2 = 2.5$ V. The decimal system with the same V_{max}, on the other hand, would have a noise margin of only $(5/9)/2 = 0.28$ V which is obviously less desirable than the binary case.

Octal and Hexadecimal Systems

While the binary system provides great practical advantages for the storage and processing of data in digital systems because it makes use of only two symbols, a given number expressed in binary consists of a much longer sequence of digits than the corresponding decimal number. If data is to be entered manually, only a two-key keyboard would be needed, but these keys have to be struck many times. This data-entry problem is often solved by treating binary numbers in groups.

Octal numbers make use of 3-bit groups in accordance with the following table:

Binary	Octal Digit
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Each octal symbol represents the numerical equivalent of a binary 3-bit group, and the eight symbols constitute a base 8 number system. In this case, an eight-key keyboard is necessary for data entry, but it need be struck only one-third as often as a binary keyboard.

EXAMPLE 1.3 Express the octal number 247 as a decimal and a binary number.

The octal number is positional, with the lowest-order (right-most) digit being weighted by $8^0 = 1$ and the highest-order digit by $8^2 = 64$. Thus $247 = 2 \times 64 + 4 \times 8 + 7 \times 1 = 167_{10}$.

Reference to the preceding table indicates that conversion to binary is easily achieved by grouping:

$$\begin{tikzcd}
 & 247 \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
010 & 100 & 111
\end{tikzcd}$$

This conversion is easily checked by determining the decimal equivalent of the resulting binary number, 10100111. Note that leading zeros may be dropped. The most significant bit is in the eighth place to the left and is therefore weighted by $2^7 = 128$. Thus, $10100111 = 128 + 32 + 4 + 2 + 1 = 167$.

A general method of converting between numbers of different bases is discussed in Sec. 1.3.

Hexadecimal notation extends the grouping idea to 4 bits and constitutes a base 16 number system. The table of corresponding bit groups and hexadecimal symbols is shown below.

Binary	Hex	Binary	Hex	Binary	Hex	Binary	Hex
0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

The hexadecimal symbols 0 to 9 are the decimal equivalents of the first ten 4-bit binary groups. To represent the last six groups, we need new symbols since there are no single decimal digits which represent numbers larger than 9. The first six letters of the alphabet are used for this purpose as shown. In the hexadecimal system, 16 keys are needed for a keyboard, but the striking rate is only one-fourth of that required with a binary keyboard.

EXAMPLE 1.4 Hexadecimal-binary conversion. (*a*) Convert 1101011100110 into an equivalent hex number. (*b*) Convert 4B2F into binary.

(*a*)	1101011100110 = (0001)(1010)(1110)(0110)	Group by 4s
	= 1 A E 6	Convert individually
	= $1AE6	The dollar sign is commonly used to indicate a hex number
(*b*)	4B2F = (0100)(1011)(0010)(1111)	Convert individually
	= 0100101100101111	Ungroup
	= 100101100101111	Drop leading zero

1.3 CONVERSION BETWEEN BASES

The following is a general method that may be used to convert numbers between any pair of bases:

1. Integers and fractions are converted separately.
2. The integer portion is converted using *repeated division by the new base* and using the sequence of remainders generated to create the new number. *Arithmetic is done in terms of the old base.*
3. The fractional part is converted by repeated *multiplication by the new base,* using the generated integers to represent the converted fraction. Again, *the arithmetic is done in the old base.*

EXAMPLE 1.5 Convert the decimal number 278.632 into an equivalent binary number.

Step 1. The integer is 278. The fraction is 0.632.

Step 2. Integer conversion.

Division	Generated remainder	
2)278		
2)139	0	
2)69	1	
2)34	1	
2)17	0	Read *up* to form: 100010110
2)8	1	
2)4	0	
2)2	0	
2)1	0	
0	1	Most significant bit (MSB)

Note that once a remainder has been formed, it plays no further role in the arithmetic. The integer process will always terminate.

Step 3. Fractional conversion.

Multiplication	Generated integer	
0.632 × 2 = 1.264	1	MSB
0.264 × 2 = 0.528	0	
0.528 × 2 = 1.056	1	
0.056 × 2 = 0.112	0	Read *down* to form: .101000011
0.112 × 2 = 0.224	0	
0.224 × 2 = 0.448	0	
0.448 × 2 = 0.896	0	
0.896 × 2 = 1.792	1	
0.792 × 2 = 1.584	1	

Note that once an integer has been formed it plays no further role. This process *may not terminate*; it is usually carried on only until accuracy requirements have been satisfied.

EXAMPLE 1.6 Convert the decimal number 123.456 to an equivalent octal (base 8) number.

Integer conversion:

Division	Generated remainder	
$8\,\overline{)123}$		
$8\,\overline{)15}$	3	
$8\,\overline{)1}$	7	Read *up* to form 173
0	1	

Fractional conversion:

Multiplication	Generated integer	
$0.456 \times 8 = 3.648$	3	
$0.648 \times 8 = 5.184$	5	Read *down* to form 0.3513
$0.184 \times 8 = 1.472$	1	
$0.472 \times 8 = 3.776$	3	

The process has been arbitrarily terminated.

$$123.456_{10} = 173.3513_8 \qquad \text{(approximately)}$$

Check:

$$173_8 = 1 \times 64 + 7 \times 8 + 3 \times 1 = 123_{10}$$

$$0.3513_8 = 3 \times 0.1250 + 5 \times 0.0156 + 1 \times 0.0020 + 3 \times 0.0002 = 0.4556_{10}$$

1.4 BASIC BINARY ARITHMETIC

All the number systems discussed previously are positionally weighted, making it possible to do arithmetic one digit at a time with the use of *carries*. Complete addition and multiplication tables can be developed by repetitive application of the rules for a single digit.

Binary Addition

The binary addition table is quite simple and is shown below, where the two digits involved are denoted by X and Y. C_i is the *carry-in* from a preceding lower-order addition.

This is the classic 1 + 1 = 2

Carry, C_i	0	0	0	0	1	1	1	1
X digit	0	0	1	1	0	0	1	1
Y digit	0	1	0	1	0	1	0	1
	0	1	1	10	1	10	10	11

Note the presence of a *carry-out* which is generated in all single-bit additions where the result exceeds 1.

EXAMPLE 1.7 Addition of two long digit strings.

	011110001
X number	1010111001
Y number	0011010101
Sum	1110001110
Carry out	0011110001

When added, each pair of digits produces a sum and a carry-out when the sum exceeds 1. This carry becomes the carry-in for the next higher order digit as shown. When, for example, X = 1, Y = 1, and the carry-in is also 1, the sum is 3 (binary 11). The left bit, having a decimal value of 2, is carried to the next higher order column, leaving a 1 in the sum position directly below.

Binary Subtraction

Subtraction could be discussed in a similar fashion, making use of a *borrow* and producing a *difference.* In practice, however, subtraction is accomplished by the same hardware which is used for addition through the use of *complementary arithmetic.* In the binary case, negative numbers are represented as the *2s complement* of the corresponding positive binary number (see Examples 1.8 and 1.9 below). Subtracting a given number X from another binary number Y is accomplished by taking the 2s complement of X to convert it to $-X$ and *adding* this to Y. In this method, the left-most digit is interpreted as a *sign bit* (0 for positive, 1 for negative) which is treated as any other bit except that a carry-out from the addition of sign bits is neglected.

The 2s complement of a binary number is obtained by exchanging the 1s and 0s of the original number and adding 1 to the result.

EXAMPLE 1.8 Subtract 185_{10} from 230_{10} by converting to binary and using 2s complement arithmetic.

How many binary digits will be required for the computation is determined by the largest number (including the answer). In this case, the number 230 is largest and requires 8 bits plus one additional for the sign bit. Thus, the binary equivalent of 185 is written 010111001. *Note that leading zeros have no effect on the value.*

We convert this to a negative number by taking its 2s complement:

Step 1: Invert the 1s and 0s.

$$010111001 \rightarrow 101000110$$

Step 2. Add 1.

$$\begin{array}{r} 101000110 \\ +\underline{\qquad\quad 1} \\ 101000111 \end{array} = -185_{10}$$

Next, again using nine places, convert 230 to binary and add this to the result of step 2 above:

$$\begin{array}{rr} +230 = & 011100110 \\ -185 = & \underline{101000111} \\ & 1000101101 \end{array}$$

Neglecting the sign bit carry (extra bit on the left) yields 000101101 whose left-most bit is 0, indicating that the result is positive.

Check: 000101101, converted to decimal, is +45.

EXAMPLE 1.9 Subtract 230_{10} from 185_{10} by converting to binary and using 2s complement arithmetic.

The binary equivalent of 230 is 011100110, and its 2s complement is obtained by inverting the 1s and 0s and adding 1:

$$-230 = 100011010$$

Next, we add this to the binary equivalent of 185:

$$\begin{array}{rr} -230 = & 100011010 \\ +185 = & \underline{010111001} \\ & 111010011 \end{array}$$

The left-most bit is a 1 indicating that the result is negative. To obtain the desired magnitude, we take the 2s complement of our result since $-(-X) = X$.

$$\begin{array}{r} 000101100 \\ +\underline{\qquad\quad 1} \\ 000101101 \end{array}$$

The decimal equivalent is 45 which we have already determined to be negative.

1.5 CODES

Binary-Coded Decimal

Binary-coded decimal (BCD) numbers are essentially decimal numbers encoded in a convenient two-valued (binary) form. Each decimal digit is represented, in order, by its 4-bit binary equivalent; 4 bits being the minimum number required to represent the decimal integers 0 to 9. Since there are 16 possible combinations of 4 bits, six of these are unused in the BCD system.

EXAMPLE 1.10 Compare the binary and BCD representations of the decimal number 278.

In Example 1.5, it was shown that the binary equivalent of 278 is 100010110. The conversion was achieved by treating the given decimal number as a whole. In BCD conversion, *each decimal digit is encoded separately:*

$$278_{10} = (\underbrace{0010}_{2})(\underbrace{0111}_{7})(\underbrace{1000}_{8}) = 001001111000 \quad \text{(BCD)}$$

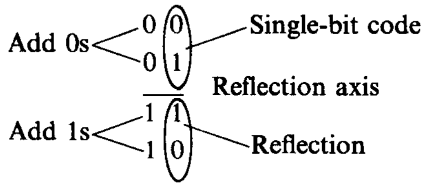
Gray Code

Another two-valued code which has engineering significance is the Gray code, sometimes referred to as reflected binary code. It is not a positionally weighted code and, for this reason, is not suitable for arithmetic operations.

Single-bit Gray code is identical to a single-bit binary code:

0
1

Two-bit Gray code is obtained by "reflecting" the single-bit Gray code in an imaginary mirror as shown below. For the second digit, 0s are added above the reflection axis and 1s below it.



Three-digit Gray code is formed by using the two-digit code as a basis for reflection and again adding 0s above and 1s below.

000
001
011
010
——— Reflection axis
110
111
101
100

The process can be repeated for any number of digits.

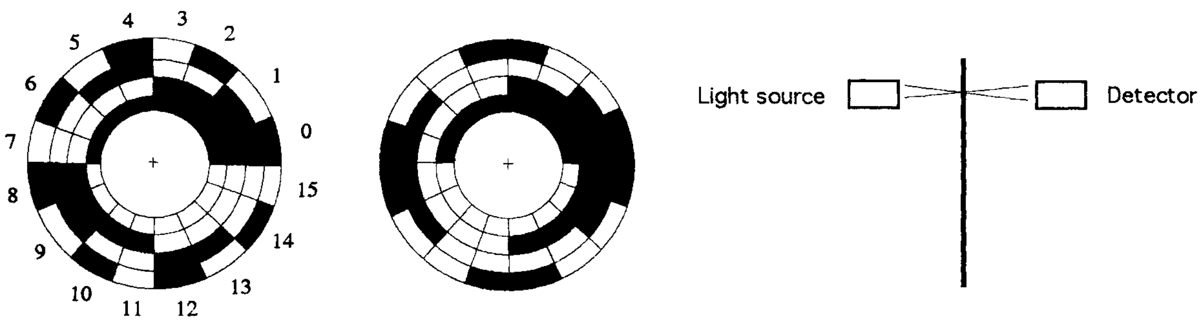
The reflection process used in Gray code generation ensures that this code will have a *unit distance* property, meaning that successive code groups will differ in only 1 bit. For reference, a 4-bit Gray code is shown in Table 1.1 along with its decimal and binary equivalents. Observe the unit distance property of the Gray code as numbers progress through the sequence. Contrast this with the binary code where, for example, on passing from 7 to 8, all bits change.

Table 1.1

Decimal	Binary	Gray	Decimal	Binary	Gray
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

One of the main applications of the Gray code is in measurement, a typical example of which is described in Example 1.11.

EXAMPLE 1.11 Position Encoding Wheel. In a robotic system, the motion of the "arm" is often directed by a microprocessor which generates a signal commanding certain rotations of joints. This command is often applied to circuitry which controls the direction and speed of an electric motor. It is necessary for the computer to know the actual position of a joint and to compare it with the command in order to be sure that the desired motion has been carried out. Position measurement is often accomplished by connecting a small code wheel to the motor shaft. This wheel consists of concentric circular tracks which contain patterns of transparent and opaque sectors as shown in Fig. 1-2. Each track is individually associated with a light source and a light detector. When a transparent sector of the track is between the source and the detector, light is transmitted and an electric signal is produced by the detector. No output occurs when an opaque sector passes between the source and detector.



(*a*) Binary coded (*b*) Gray coded Code wheel (side view)

Fig. 1-2

The wheel in Fig. 1-2*a* is binary coded with the outermost track corresponding to the least significant digit. With a radial array of separate source-detector pairs aligned with each track, we see that as the wheel rotates, each sector passing the detector array will produce unique combinations of outputs which may be interpreted as binary numbers. For example, if the source-detector array is located along a vertical line at the top of the wheel and an illuminated detector is considered to produce a binary 1, then if the wheel rotates counterclockwise by somewhat more than one sector, detector outputs will indicate the binary number 0010. In the case of the wheel in Fig. 1-2*b*, sectors are identified by Gray-coded numbers.

Consider the binary-coded case where the wheel is positioned so that the detector array is located along the dividing line between sectors 7 and 8. Note that the sectors on each side of the boundary are different for all tracks. The light sources and detectors are not individually aligned with perfect tolerance, nor are the source emissions of zero width; the light spreads. If a light source is on the line, it may or may not cause a detector output. Thus, the binary number produced may be anywhere from all 0s to all 1s depending upon alignment and light spread. Gross errors can occur.

On the other hand, if we use a Gray-coded wheel, because of the unit distance property, there is only one track per sector where light transmission on each side of a sector boundary is different, so only one Gray-coded bit can be erroneous. The Gray-coding scheme can only produce numbers corresponding to adjacent sectors, and, consequently, no large errors are possible.

While Gray code is quite suitable for measurement, it is not useful for arithmetic because, as previously mentioned, it has no positional weighting.

Conversion Between Binary and Gray Codes

This problem is handled efficiently by noting that the Gray code can be considered to be a "differentiated" version of the equivalent binary. Conversion proceeds according to the following rules.

A. Conversion from binary to Gray

1. The left-most digits are the same in both systems.
2. Read the binary number from left to right. A change (0 to 1 or 1 to 0) generates a 1 in the Gray-coded number; otherwise a 0 is generated.

EXAMPLE 1.12 Binary-Gray conversion. Convert 01101001101 binary to Gray.

```
Binary:   0 1 1 0 1 0 0 1 1 0 1
          ---------------------
          | | | | | | | | | | |
          D C S C C C S C S C C      S = same; C = change; D = duplicate
          | | | | | | | | | | |
          ---------------------
Gray:     0 1 0 1 1 1 0 1 0 1 1

               01101001101  (binary) = 01011101011  (Gray)
```

B. Conversion from Gray to binary

1. The left-most digits are the same in both systems.
2. Read the Gray number from left to right. A 1 means that the next binary digit must change; a 0 means the next binary digit is identical to the digit on its left.

EXAMPLE 1.13 Gray-binary conversion. Convert 1000110101010 Gray to an equivalent binary number.

```
Gray:     1 0 0 0 1 1 0 1 0 1 0 1 0
          -------------------------
          | | | | | | | | | | | | |
          D S S S C C S C S C S C S
          | | | | | | | | | | | | |
          -------------------------
Binary:   1 1 1 1 0 1 1 0 0 1 1 0 0
   1000110101010 (Gray) = 1111011001100 (binary)
```

ASCII Code

Not all data stored and processed by computer is numerical. Because of the practical advantages of the binary system, other sorts of data are stored in two-valued (binary-like) form as well. The most commonly encountered *alphanumeric* code is the *American Standard Code for Information Interchange* (*ASCII*) of which some representative keyboard characters are presented in Table 1.2. This list is not intended to be exhaustive; merely illustrative. Note that decimal digits are listed as BCD-encoded digits preceded by 011 (9 = 011 1001). Other three-digit prefixes are used for nonnumeric data.