SURVEY & TUTORIAL SERIES

Multiple-Valued Logic: A Tutorial and Appreciation

Kenneth C. Smith
University of Toronto

This tutorial is meant to intrigue and inform the reader whose interests and experience might not yet include multiple-valued logic. By raising and reviewing issues often of concern to those first encountering the topic, it sets the stage for the five articles on multiple-valued logic that follow. It assumes that the typical reader is broadly conversant with the modern field of (binary) digital computing and is at least generally aware of its mathematical foundations and constructs, as well as the nature of its important recent developments. Thus, this tutorial places the developments and potential of multiple-valued signals and logic in the relevant context of binary or two-valued signals.

This tutorial places the developments and potential of multiple-valued signals and logic in the relevant context of binary and two-valued signals.

Two-valued signals currently dominate the field of digital-computing machinery, but, historically, there is no lack of emphasis on radices (or bases) higher than two.[1] Base 10 has been important for all the obvious reasons, particularly in mechanical systems and in the users' view of electronic machines. Babbage's original design used 10-valued mechanisms, as did generations of later mechanical calculators. Modern electronic calculators present data in base 10 for user convenience, as did early machines such as the IBM 1620. With very few exceptions, such machines used inherently binary electronic circuitry and logic internally, with information encoded in binary-coded-decimal format, but often with computation done in (coded) decimal arithmetic.

Thus, the choice of radix may be optimized separately in either the conceptual or actual (that is, implementation) domain, and a "best" choice could be one where both optima coincide.

As we will explore shortly, issues of concern at the conceptual level include notation, operational description, and recognition of symmetry. At the actual level they include the use of physical space, noise margins in signal space, and conversion with binary.[2,3]

MVL's role in the binary world

Concerning the role of multiple-valued logic (MVL) and data representation in the dominantly two-valued or binary world, a multiplicity of views exists. First, it is unlikely that binary will capitulate or in any way wither and die. Besides its obvious entrenchment, there are very good reasons why two is a special value. For example, every designer of binary logic knows that a logic gate, say NAND or NOR, need have a fan-in of only two to be perfectly general, although not particularly flexible. He knows as well that, while large gate fan-out is very convenient, a fan-out of two is all that is necessary.

Thus, even from a binary perspective, we see that two is enough but more can be better. This leads us to the first view presented in this tutorial—namely, that multivalued logic has at least the potential for enriching the current two-valued reality.

Conversion with binary. Because of the necessary coexistence with binary logic, radix conversion is a topic of immediate concern. To the practicing binary logic designer, the required converters represent no great challenge conceptually. Such multiple-valued-to-binary and binary-to-

0018-9162/88/0400-0017$01.00 © 1988 IEEE

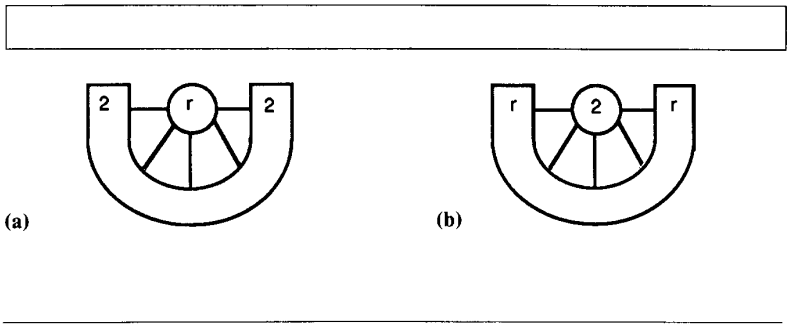


Figure 1. The digital universe: (a) binary view; (b) multiple-valued view.

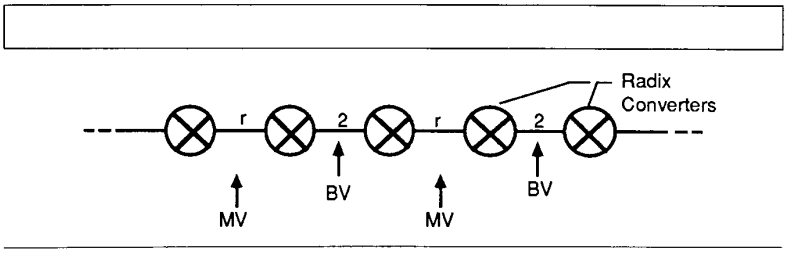


Figure 2. Another look at reality, described as a chain of two-valued and *r*-valued parts from two viewpoints: BV, binary-valued, and MV, multiple-valued.

multiple-valued converters are merely particular cases of the analog-to-digital and digital-to-analog converters with which the designer is already familiar. In particular and on the positive side, the required converters have very low precision by present-day analog standards and, accordingly, can be quite simple structurally. Fortunately, this simplicity is quite consistent with the obvious desire for speed, since in most cases (but not all!) too great a speed penalty might overwhelm the other advantages that multiple-valued data representation and logic can provide.

On the issue of conversion, a moment's reflection suggests that some particular choices of value in a multiple-valued system may be more attractive than others.

A special choice of radix—a power of two. Conversion with binary is most efficient with special choices of radix, ones in which no information is lost or left unused. These radices are exact powers of two, that is 2^n, for $n = 1, 2$, etc., of which obviously two itself, four, possibly eight, and potentially 16 show considerable promise. For example, as indicated later, in the area of memory-system applications, there have been commercial developments in four-valued ROM and proposals for even 16-valued RAM systems.

Two views of multiple-valued systems. As already noted, present and potential multiple-valued systems must be viable in the binary real world. Two such views are expressed symbolically in Figure 1. In each, the focus of attention is at the top and center—"on the ball," so to speak—and emanates from a related universe "U." As in many ideological issues, only the labels are different.

Figure 1's views can be combined and/or extended (and certainly detailed) as indicated in Figure 2. A modern view of digital reality, it shows a collection of interacting parts as labeled links coupled by converters (marked "x"). For simplicity, it reduces the obvious potential generality of the interconnection to only a single dimension. Also for simplicity, it indicates only radix 2 and radix *r* sections, although hybrids using a multiplicity of radices are clearly possible.

From Figure 2, one can appreciate the nature of two recurring issues in the multiple-valued literature (and related detailed views). They are represented pictorially by the arrow pairs BV and MV. BV, for binary view, presents the system as primarily binary, but possibly consisting of linkages that may be internally multivalued. The multivalued ROM alluded to earlier, and described later, exemplifies such a system, one in which the user's view is clearly binary, while the reality of its multivalued implementation may remain unrevealed or even concealed. Arithmetic processors constitute another important example, where multiple-valued data representation provides additional conceptual and implementational flexibility. Alternatively, MV, for multiple-valued view, presents the system as multiple-valued or as having multiple-valued segments, which in turn (internally) may have binary parts.

This idea appears in practice in at least two distinguishable ways. It's one way of constructing some multiple-valued gates, as well as a way for understanding many others. Quite separately, it describes a general scheme for using multiple-valued linkages to reduce wiring complexity between otherwise binary subsystems, in either VLSI or larger scale environments.

Later in this tutorial, in sections on application and circuit implementation, I discuss these issues further.

Representation

Extending systems from binary to radix *r*—that is, from the usual two-valued to a multiple-valued representation—requires choices involving the relative separation and ordering of values, issues (both conceptual and actual) that are less visible in two-valued systems.

While other possibilities exist, one usually thinks of the multiple values as *monotonic*, being separated in an established sequence at (nominally) equal intervals. As a consequence, just as in binary, a conceptual notion of relative magnitude arises, one extreme magnitude being larger (or smaller) than the other. Notationally, one extremum can be called zero and the other one, with intervening values represented by fractions.

Thus, a three-valued system (also called a ternary system) exists for which the values are called (0, ½, 1). Just as in binary, the "largest" value is considered to be one (the unit value, unity, *u*, "all there is," etc.). However, just as with "negative logic" in radix 2, the actual value of the physical variable corresponding to one need not be the largest.

Other notations exist as well. For an odd radix such as 3, the balanced ternary (also called "signed binary"!) notation (-1, 0, +1) is often used to emphasize symmetry. More generally for radix *r*, the values can be labeled (0, 1, 2, . . . , (r − 1)), the corresponding three-valued notation being (0, 1, 2). The latter style of notation (and the name *n*-valued rather than *n*-ary) is emphasized in the present description.

Binary-related radices

As already noted, power-of-two radices ($r = 2^k$, k integer) have an obvious special relationship to the conventional binary system. In addition to the fact that conversion to and from binary is particularly convenient, the conversion is quite "efficient," since the information capacity of a single *r*-valued digit is exactly matched by that of *k* binary digits (where $r = 2^k$). This "direct-packing" property of binary-related radices is quite important in some of the more obvious applications of multivalued coding. Such is the case, for example, with the application of four-valued digit encoding (called quaternary and for which $k = 2$) in several commercial ROM designs and CCD designs described shortly. More recently, development of a 16-valued RAM design, for which $k = 4$, has been reported, although a product is not yet commercially available. Moreover, the success of a recently

Multiple-valued functions of one variable

The number and naming of one-variable functions is a non-issue in binary, but such is not the case in multiple-valued logic systems. The reason is a combinatorial one: an *r*-valued system has *r* possible outputs for each of *r* possible input states and, accordingly, r^r functions of a single *r*-valued variable. This corresponds to 2^2 or 4 possible functions in binary—namely, the identity, the complement, and the constants 0 and 1. Thus, in binary, only half the available functions—identity and complement—are really "active" or "interesting," and little more can be said.

But a multiple-valued radix-4 system, for example, has 256 such functions, only four of which are constant and 252 of which need names. Clearly, even in a four-valued system, the process of naming can get out of hand.

Nevertheless, for a variety of reasons related to our binary heritage, some special functions have been identified and labeled. The accompanying table views some that can be defined independently of the particular radix *r*. However, for illustration, examples in both radices 2 and 4 are provided.

Included also is a general-purpose functional notation. It consists simply of a string of *r* *r*-valued digits delimited by the symbols < >. In this string, each item is the output value produced by the corresponding input in the standardized reference input string (0,1,2, . . . , *r* − 1). For example, in the four-valued case illustrated, the identity function is labeled <0,1,2,3>, indicating the output is the same as the input for each value.

Selected functions of a single multiple-valued variable with four-valued and two-valued examples.

Ref. no.	Function name *f*	Preferred notation	Function definition $x_0 = f(x_i)$	Other notation	Four-valued example Function	Output	Two-valued example Function	Output	Binary equivalent
1	Identity	x	x		x	<0,1,2,3>	x	<0,1>	Identity
2	Complement	$\bar{x}$	$r-1-x$		$\bar{x}$	<3,2,1,0>	$\bar{x}$	<1,0>	Complement
3	Successor	$\vec{x}$	$(x+1)_{\text{mod}r}$	suc(x) x_{+1}	$\vec{x}$	<1,2,3,0>	$\vec{x}$	<1,0>	Complement
4	Predecessor	$\overleftarrow{x}$	$(x-1)_{\text{mod}r}$	x_{-1}	$\overleftarrow{x}$	<3,0,1,2>	$\overleftarrow{x}$	<1,0>	Complement
5	Cycle	x_*	$(x+k)_{\text{mod}r}$ $\{-r+1 \le k \le r-1\}$	$\vec{x}^k$	x_1	<1,2,3,0>	x_0	<0,1>	Identity
					x_{-1}	<1,2,3,0>	x_1	<1,0>	Complement
6	Literal	$\overset{a,b}{x}$	$r-1$ if $a \le x \le b$ 0 if $a > x > b$		$\overset{1,2}{x}$	<0,3,3,0>	$\overset{0,1}{x}$	<1,1>	Constant
					$\overset{0,3}{x}$	<3,3,3,3>	$\overset{0,0}{x}$	<1,0>	Complement
							$\overset{1,1}{x}$	<0,1>	Identity
7	Universal literal	x^S	$r-1$ if $x = s \in S$ 0 if $x \ne s \in S$		$x^{1,2,3}$	<0,3,3,3>	x^0	<1,0>	Complement
					x^1	<0,3,0,0>	x^1	<0,1>	Identity

Table 1. The number of functions of one variable in radix r.

r	r^r
2	4
3	27
4	256
5	3,125
6	46,656
7	823,543
8	16,777,216

developed quaternary multiplier can be at least partly ascribed to this direct-packing property.[4]

In view of these facts, many of the examples presented in this tutorial are four-valued.

Multiple-valued functions

Functions of one variable. One can appreciate both the related power and potential complexity of multiple-valued systems by considering functions of only one variable. In general, for a single-variable function in radix r, r possible outputs are available for each of the r possible input values. Accordingly, there are r^r such one-variable functions. Thus, for the binary case, where $r = 2$, there are $2^2 = 4$ possible functions of a single variable x_i, namely, for $x_i = 0$ or $x_i = 1$, the output x_o can be 0 or 1. That is, the output can be the same as the input ($x_o = x_i$, an identity function), reversed from the input ($x_o = \bar{x}_i$, the binary complement function) or constant (either 0 or 1). As the radix increases, so does the number of functions—quite dramatically, as shown in Table 1, where in radix 4, for example, the number of single-variable functions is seen to be 256.

The consequence of this increase is that the usual binary practice of simply naming the few available functions (that is, identity, complement, and constant 0 or 1) gets out of hand in higher-radix systems. Thus, while some named functions prevail, a more general approach is needed. One approach is simply to define the function as an output r-tuple, a simple list (or string) of outputs corresponding on a one-to-one basis to inputs in the usual (monotonic) ascending order.

Consider, by way of example, a four-valued variable for which the reference input string is (0, 1, 2, 3). The corresponding identity output function is straightforward and is written as $<0, 1, 2, 3>$. In this notation, $< >$ signifies an output string whose members are in one-to-one correspondence with those of the reference input string. Thus, for the identity function, 0 input produces 0 output, 1 input produces 1 output, and so on. Correspondingly, a particular complement, written $<3, 2, 1, 0>$ and for which, in general, $x_o = \bar{x}_i = r - 1 - x_i$, is often called diametrical negation or inversion. But what about all the others? Clearly there are a lot of possibilities (in fact, 254 remaining combinations), and simple names are quickly exhausted.

Historically, there are several specially named multiple-valued functions of a single variable. As well as the (diametrical) inverter already described, these include the "cycle" and the "literal" functions. The *cycle*, for which $x_o = \overrightarrow{x}_i = (x_i + 1)_{\text{mod}r}$ in general (where in modular arithmetic addition a result $\geq r$ is reduced by r) becomes $<1, 2, 3, 0>$ in the particular four-valued example. A variation, called a downward or negative cycle ($\overleftarrow{x}_i$), in which addition is replaced by subtraction, becomes $<3, 0, 1, 2>$ in the radix-4 example. Cycling by a value k, rather than simply by 1, is also possible.

The *literal*, for which $x_o = {}^a x_i^b = r - 1$ if $a \leq x_i \leq b$ and $x_o = 0$ otherwise, would become $<0, 3, 3, 0>$ in the four-valued case in which $a = 1$ and $b = 2$. When $a = b$, a *simple literal* ($x_i^a = {}^a x_i^a$) results, becoming $<0, 3, 0, 0>$ in the four-valued case in which $a = 1$. This particular idea has been extended to provide a *universal literal*, for which $x_o = x_i^{S_i} = r - 1$ if x_i takes a value a_i in the set S_i and $x_o = 0$ otherwise. Thus, in a four-valued case where S_i is (0, 2), $x_i^{0,2}$ becomes $<3,0,3,0>$. (The reader will come to appreciate the true notational power of the universal literal when a discussion of issues related to implementation is reached.)

In addition, there are other special functions identified historically for a variety of reasons, including convenience of application, analysis, and implementation. Generally, in hindsight, they can be described in terms of the universal literal notation. For a summary of some of the notable functions of a single variable, see the table in the sidebar titled "Multiple-Valued Functions of One Variable." Figure 3 shows transfer graphs or transfer characteristics of some of the more interesting and useful functions of one variable, namely, the identity, complement, single-threshold inverting, and cycling functions. Note that this figure shows the input switching thresholds midway between each of the input values.

Functions of more than one variable. In the discussion of functions of a single variable, we have already seen the conceptual and notational utility of two simple concepts, that of the string notation and input-output transfer graphs. These ideas—although available—are rarely employed in binary for two reasons: first, because their use with two values is somewhat degenerate and, second, because convenient higher level functional notations are available.

In contrast, for two or more variables, the tabular and mapping techniques common to binary are used directly for multiple-valued design. The only important difference is one of size, obviously related to the increased number of values to be tabulated. For example, the truth table for a two-variable four-valued two-output function (the arithmetic sum and carry) shown in Table 2 has $4^2 = 16$ rows. Obviously, however, four binary digits are needed for the same input information content embodied in two radix-4 digits, with the corresponding table also having $2^4 = 16$ rows. Figure 4 shows a two-variable four-valued map for the arithmetic sum function.

Thus, for the purpose of reducing size alone, if for no other reason, there is an important need for a multiple-valued functional notation.[3]

For generality in describing functions of two or more multiple-valued variables, consider x_i to be such a variable with a value from the set $V = \{0, 1, 2, 3, \ldots, r-1\}$ and $X = \{x_1, x_2, \ldots, x_n\}$, the set of n such variables. However, for expediency, examples will be posed in terms of two variables in a four-valued system, that is for $n = 2$ and $r = 4$.

While the number of functions of even two r-valued variables can be quite large (r^{r^2} in general) in comparison to the two-valued case (where $2^{2^2} = 16$), fortunately, the number of basic common constructs does not grow correspondingly. Note also that for the particular case of $r = 2$, each directly reduces to a familiar binary form.

Historically, the most important multiple-valued connectives have been the

maximum (MAX) and *minimum* (MIN) functions, whose outputs are respectively the largest and the smallest of the n values presented. Note that for two-valued variables, *maximum* corresponds to the binary OR while *minimum* corresponds to AND. This correspondence justifies the following notation, where for multiple-valued variables x_i, $1 \leq i \leq n$,

MAX $(x_1, x_2, \ldots, x_n)$ = $x_1 \vee x_2 \vee \ldots x_n$, is the largest x_i, and

MIN $(x_1, x_2, \ldots, x_n)$ = $x_1 \cdot x_2 \cdot \ldots x_n$, is the smallest x_i.

Corresponding as well to the binary tradition is the notion of *sums* and *products*, with the *sum of products* and *product of sums* as general functional constructs.

In addition, however, motivated by the natural association of multiple-valued variables with arithmetic, there are other, more arithmetic, connectives including arithmetic sum, difference, and product forms. Because of their relatively simple electronic implementation, only sum and difference forms will be considered in this limited tutorial. Note that the arithmetic-product forms, though not considered here, may assume greater importance with the potential application of multiple-valued techniques to optical signal processing. Thus, besides MAX and MIN, there are TSUM and MODSUM operations, which are truncated and modular summation,[5] respectively.

Accordingly, for a two-variable, four-valued environment,

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

$x_1 \vee x_2$ = MAX (x_1, x_2)

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

$x_1 \boxplus x_2$ = TSUM (x_1, x_2) = MIN $(x_1 + x_2, 3)$

$x_1 \oplus x_2$ = MODSUM (x_1, x_2) = $(x_1 + x_2)_{\text{mod}4}$

where the symbolism is

- $\cdot$ for MIN (AND)
- $\vee$ for MAX (OR)
- $+$ for arithmetic addition (plus)
- $\boxplus$ for truncated addition
- $\oplus$ for modulo-r addition (Exclusive-OR, or EXOR, in binary)

Generally speaking, which connective is most appropriate depends on the technology available and the application's need for minimization. Thus, in some technol-

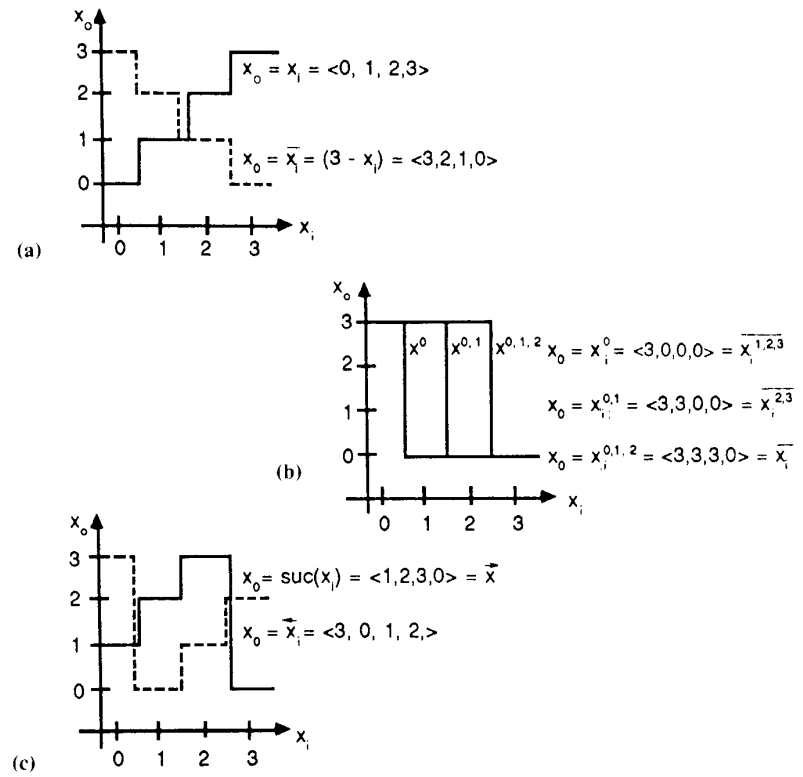


Figure 3. Transfer characteristics of sample functions of a single four-valued variable x: (a) identity x and complement ($\bar{x}$); (b) single-threshold inversion; (c) successor (suc(x)) and cycle ($\vec{x}$, $\overleftarrow{x}$).

Table 2. A truth table for the sum (S) and carry (C) outputs of a half adder for two four-valued inputs (x_1, x_2)

x_1	x_2	S	C
0	0	0	0
0	1	1	0
0	2	2	0
0	3	3	0
1	0	1	0
1	1	2	0
1	2	3	0
1	3	0	1
2	0	2	0
2	1	3	0
2	2	0	1
2	3	1	1
3	0	3	0
3	1	0	1
3	2	1	1
3	3	2	1

x_1

x_2 \ S	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Figure 4. A map of the sum (S) output function of a four-valued half adder having two inputs (x_1, x_2).

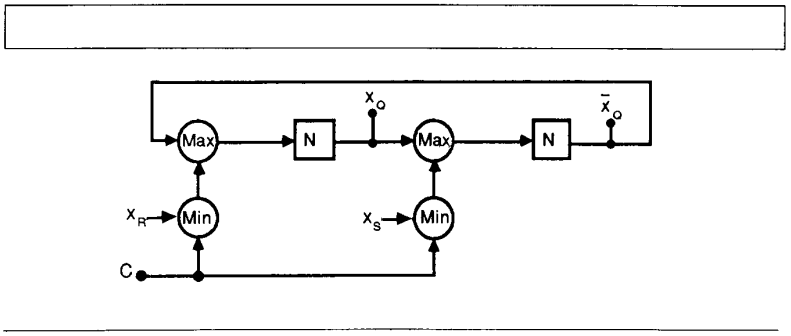


Figure 5. A clocked multiple-valued storage element using complement (N), MAX, and MIN gates.

ogies (notably those using current or charge), summation operations are very convenient. However, the corresponding complexities of don't-care conditions make minimization relatively difficult. Conversely, in these same technologies, MAX and MIN are somewhat complex, but don't-care conditions are straightforward and highly visible, making minimization relatively easy.

Storage techniques in MVL

Techniques for storage of digital information in multiple-valued logic systems relate directly to those commonly encountered in binary systems. The major difference is that the information represented by a signal, and stored within a single multistable element, can be much greater than in the binary case. However, in both the binary and multiple-valued worlds, gating (that is, the decision of whether to renew or retain the information stored) remains an essentially binary function.

In both binary and multiple-valued systems, two types of storage mechanism are normally encountered. The most economical scheme in binary, namely, dynamic storage using a capacitor, becomes particularly attractive with multiple-valued signals since a single capacitor with a simple transmission gate has the potential for storing much more information.[6] Practical problems, generally related to noise margins in readout and refresh operations, abound in such systems. Nevertheless, as far as the storage medium and simple gating alone are concerned, the storage efficiency of multiple values on a single capacitor is certainly attractive. This is particularly the case in hybrid implementations using a combination of dynamic and static storage for economy and speed—for example, in master-slave register-element construction.

Structures for static storage in multiple-valued systems generally follow the pattern set in the binary case, namely, that of an active positive-feedback loop. For example, just as in binary, a feedback loop consisting of two inverters or complement circuits (that is, gates for which $x_o = \bar{x}_i$) will exhibit stable states, in particular r of them in a radix-r system. Gating is also similar in multiple-valued and binary schemes, with MAX and MIN replacing OR and AND. A clocked SET-RESET flipflop following this general approach is shown in Figure 5.

In Figure 5, N represents a complement circuit. In a four-valued environment, its function $\bar{x}$ is described by $<3,2,1,0>$ (that is, the output corresponding to the input string (0,1,2,3)). All signals x are multiple-valued; signal C is a two-valued clock that alternates between the values of zero and three. Signals x_R and x_S represent multiple-valued Reset and Set inputs, respectively, which are usually (but not necessarily) complementary. Signals x_Q and $\bar{x}_Q$ are the (usually) complementary multiple-valued outputs of the four flop. When the clock C is low (at zero), the lower input of each of the MAX gates is held low, and the storage state prevails. When the clock is high (at three), the four-flop state is changed to a value established by x_R and x_S. For $x_R = \bar{x}_S$, $x_Q = \bar{x}_Q = x_S$ once C has fallen to zero. Note that the static storage element shown in Figure 5, is "signal-level-restoring," that is, has correctly quantized output-signal values, by virtue of the design of the complement circuits.

Implementation

Basic issues. While there is much to be said about implementation, and much of that is explored in other articles in this special issue of *Computer*, it is likely of interest here, for the general reader, to explore some of the underlying issues.

First, let us recognize once again that the fundamental abundance of binary devices, originally mechanical and now electronic, underlies the current dominance of binary technology. The early mechanical age of computation saw a recurring appearance of multiple-valued techniques, for example, detented cogs having 10 stable positions. As fortune would have it, however, multistate electronic devices are far rarer.[3] Whether this is good or bad, time alone will tell.

But note two things in passing. First, while there is a lack of basic electronic multistate devices, relatively simple equivalent constructs of analog and binary devices are available. Second, the history of technology displays various recurring cycles; thus, it is not unreasonable to expect that some emerging technology may have more multivalued attributes. This is, for example, a possibility with electro-optical and optical technologies. Meanwhile, while waiting, many real developments can be implemented *now*. Expanding on this possibility is the direct motivation of this section and of other articles in this issue of *Computer*.

Signal representation. In electronic implementations of binary digital systems, it is evident that signals can be (and are) represented directly by voltage, current, or charge or by some combination. In fact, the interrelationship of these variables through the capacitive and resistive elements of a VLSI environment almost makes such distinctions irrelevant. Nevertheless, circumstances can bias one's deliberations in binary design (and in multiple-valued design for that matter). The existence of oscilloscopes, which measure voltage so easily, tends to bias thinking and descriptions (such as waveform presentations) in the voltage direction. Also, some technologies are easily described in one domain but less well in

A universal multiple-valued building block: The T-gate

Overview. As an aid to understanding the implementation aspects of multiple-valued logic functions, let's consider, at a general functional level, a particular "universal" element called the T-gate.[1,2]

The T-gate qualifies as a universal element in several different senses. First, it is logically complete; that is, all multiple-valued functions of one or more variables can be created with a T-gate. Second, its operation is intuitive; that is, one can relatively easily understand what it does, both as a single basic element and in collections. Third, it is easily implemented; that is, its construction in most, if not all, available technologies is quite straightforward. Fourth, it highlights two essential elements that must be embodied in any logic gate, namely, logic-value thresholding and logic-signal connection or switching.

But don't assume that a T-gate constitutes the ultimate in multiple-valued constructs. It has some shortcomings: While intuitive, the T-gate does not lead to a very effective scheme for minimization. While usable as both a model for all multiple-valued realizations and a means to their actual implementation, the T-gate's models and implementations are seldom minimal. Thus, for multiple-valued logic, the T-gate is a useful general-purpose tool; but like other general-purpose tools, it does everything to a degree but few things particularly well.

In terms familiar to a binary designer, the T-gate is simply a multiplexor. It is, however, specially adapted to an r-valued world. In particular, it has $r+1$ inputs, one of which is an r-valued control input whose value determines which of the other r (r-valued) inputs is selected for output.

The idea is shown conceptually in the figure at left below, where x_k is a four-valued controlling or selecting input that takes on values (0,1,2,3). For $x_k = 0$, input x_1 is selected; for $x_k = 1$, x_2 is selected, and so on. Thus, in general $x_0 = <x1, x2, x3, x4>$, using the string-sequence notation introduced in the main text.

The middle figure below, a schematic internal view of the T-gate, distinguishes the separate roles of the selected and selecting inputs. Here, the "x" symbols denote pass-transistor switches, or transmission gates, controlled by mutually exclusive binary signals under control of the selecting input x_k. The figure at right below is a more detailed schematic in which one-at-a-time operation of the r pass transistors is more precisely specified by use of four universal-literal gates driven by the selecting input x_k.

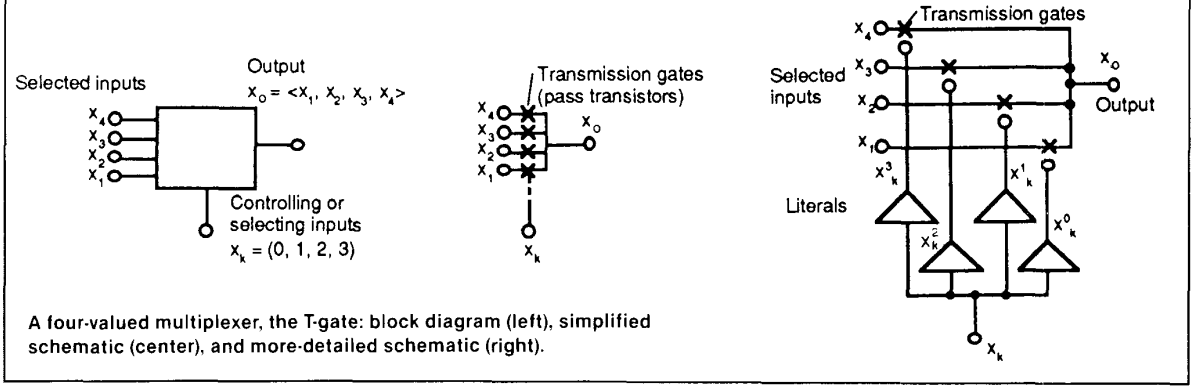
As usual with r-valued constructs, the two-valued T-gate is quite recognizable; in fact, it is normally described as a transmission gate. Incidentally, the identification of it as a general-purpose transmission gate, in which the controlled input and output are not restricted to two-valued signals and may in fact be analog, foreshadows its more-general multiple-valued application. This is exemplified in the sidebar on T-gate applications (see p. 26).

Related insights. Now that we've viewed some overall aspects of the T-gate, let's reflect on the insights it provides to general implementation problems.

As noted, the T-gate involves threshold detection, an inherently binary process. The thresholds, as the figures indicate, are relatively complex; however, being double, as the universal-literal notation in the right-hand figure shows, the switch-controlling outputs are still binary. In this sense, the T-gate follows the general scheme introduced as MV in the main text (Figure 2), where multiple-valued signals are first decoded (by thresholding) and then processed using binary techniques. On the other hand (a hint of a richer world), although T-gate switches are obviously under binary-signal control, signals in the controlled path can be more general. In particular, they can be multiple-valued.

This alone can be an interesting observation for those of binary persuasion, in view of the current importance of pass-transistor logic in binary VLSI. It is, perhaps, even more intriguing to reflect on two other associated facts. First, the passpath, often of considerable length in some VLSICs, if made multiple-valued, can pack more information.[3] Second, the pass-transistor-controlling signals, while still binary and therefore relatively numerous, are typically located in the immediate vicinity of the threshold detectors, and they can be quite short.

1. M. Kameyama, T. Hanyu, and T. Higuchi, "Design and Implementation of Quaternary nMOS Integrated Circuits for Pipelined Image Processing," *IEEE J. Solid-State Circuits*, Feb. 1987, pp. 20-27.
2. T. Higuchi and M. Kameyama, "Ternary Logic System Based on the T-Gate," *Proc. Fifth Int'l Symp. Multiple-Valued Logic*, Bloomington, Ind., May, 1985, pp. 290-304.
3. K. Horie et al., "Design of a Complementary Pass Gate Network for a Multiple-Valued Logic System," *Proc. IEEE 17th Int'l Symp. Multiple-Valued Logic*, Boston, May 1987, pp. 142-149.



A four-valued multiplexer, the T-gate: block diagram (left), simplified schematic (center), and more-detailed schematic (right).

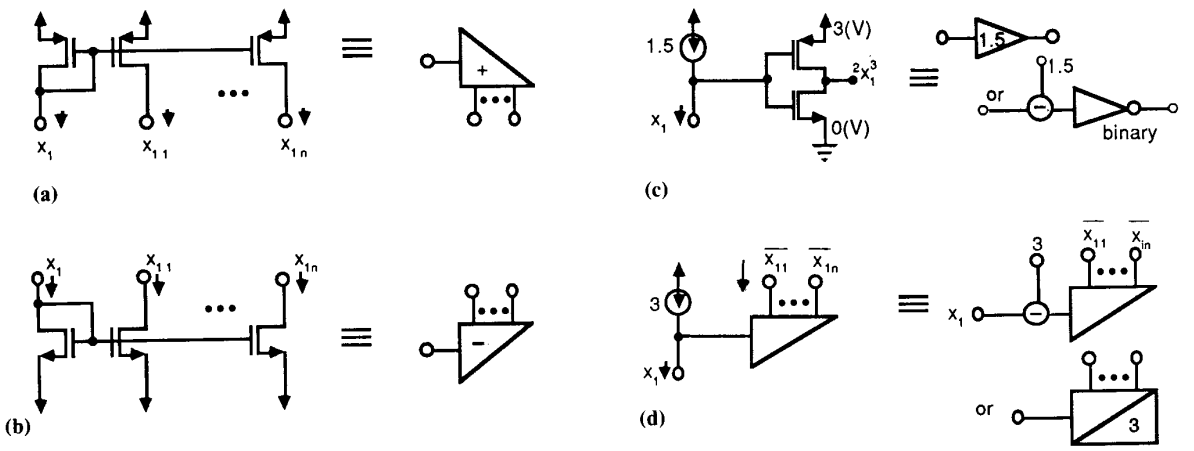


Figure 6. Component circuits of a simple current-mode four-valued circuit family: (a) positive replicator, (b) negative replicator, (c) literal ${}^2x^3$, (d) complement $\bar{x}$ with replicated outputs. In the circuits shown, the reference current values are conceptual. In practice, a small fixed current bias would be added to ensure that the mirrors are never cut off. This is a technique called "fat zero" in other, but related, contexts.

others. But in general, in the binary world, such distinctions do not matter; one uses the most convenient description, one which usually dwells on the assumed reality of zero and one, independent of implementation.

However, for multiple-valued systems in VLSI, it is no longer appropriate to ignore these distinctions, particularly between voltage and current representations. The reason is that each may be seen to impact directly (and dominantly) on one of two critical dimensions of the VLSI defining domain.

The fact is that a voltage representation impacts directly on the supply-voltage dimension of VLSI. In binary systems, design details affect noise margins in subtle ways, but it is the value of the power supply that really sets the tone. Witness the controversy currently surrounding the choice of a standard supply voltage for VLSI design. While five volts was once necessary for many reasons, it is now clearly far greater than necessary for many (perhaps most) applications and newer technologies, and it wastes considerable power and even reduces speed. In fact, other standards exist, certainly for portable applications such as watches, within VLSI application-specific and custom chips.

This issue is critical when considering a multiple-valued-voltage representation. It could even be argued that straightforward multiple-valued-voltage representation is feasible only because of the voltage surplus in most present-day binary systems.

On the other hand, true multiple-valued-current logic does not require increased voltage space (although current-voltage hybrids do). Rather, to maintain the particular device-current density that characterizes a particular VLSI process, device width must increase in proportion to the multiplicity of current values to be used.

The above discussion has highlighted a distinction, once ignored in binary, that is becoming important: multiple-valued signaling needs space. Whether this is in the voltage-supply dimension or in a linear chip dimension is an important design choice. The voltage choice, if taken, tends to be system-wide, while the current choice is more localized, affecting only the part of a VLSI system in which multiple-valued generating (and detecting) circuits exist. A positive aspect is that both are linear with radix.

This tutorial concentrates on multiple-valued signals in the current domain. Occasionally, however, hybrid systems will be referenced, ones in which a voltage representation is used quite locally, where perhaps smaller noise margins, and accordingly less voltage span, can be tolerated. Note that charge-based systems are also hybrid in a related sense, namely, that an inevitable variation of voltage exists across a fixed capacitor (or charge well). In a practical sense, however, such voltages can be small, since relevant connections are local. The common (but not essential) need to convert signals to the multiple-valued voltage domain is, of course, accompanied by the obvious problems associated with limited voltage space.

A current view. In any logic implementation, a major consequence of the voltage/current duality is that difficult voltage-mode operations become easy in current mode and (regrettably) vice versa. Thus, for a TTL NAND operating in voltage mode, fan-out is a straightforward wiring task, but fan-in requires some particular hardware for each input. In current mode, the reverse is true: fan-in is very simple by wire connection, but fan-out is much more complex, requiring circuitry of the type shown in Figures 6a and 6b. For this reason, current-mode binary circuitry was quite uncommon until creation of *binary* I^2L (integrated injection logic) recognized the convenient implementation and utility of multicollector transistors. The realization by Dao[7] and others of the relationship of these structures to current mirrors led to the development of *multiple-valued* I^2L in the 1970s. This was the first use of mirrors as signal-

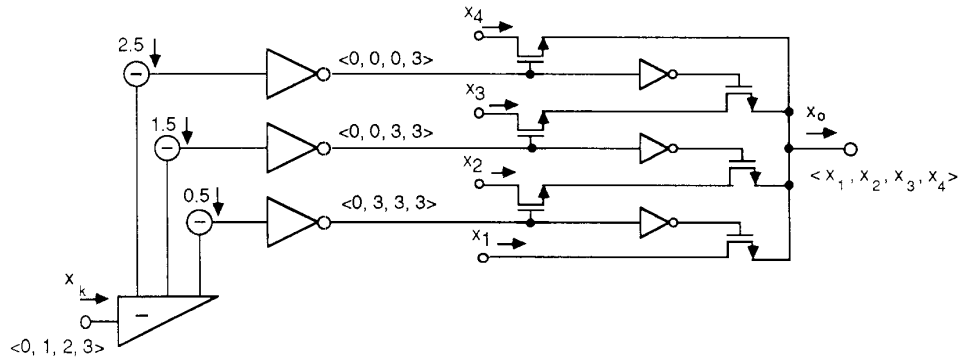


Figure 7. A T-gate implemented in a simple current-mode four-valued circuit family. The switches shown as nMOS are appropriate for this application. They are easily converted to CMOS by employing the existing (binary) CMOS inverters (shown as triangles) to drive the additional pMOS pass transistors.

current replicators in digital systems.

Component circuits of a simple current-mode multiple-valued circuit family are shown in Figures 6a-d and 7. Simplistically viewed, the scheme is radix-independent. However, for practical reasons related primarily to mirror imperfections, the radix is limited. Certainly, four-valued designs have been demonstrated and will be considered by way of example.

Figures 6a and 6b show replicators using simple mirrors to provide n copies of a multiple-valued current signal x_1. Figure 6c, the circuit of a particular literal, $x^{2,3}$, illustrates two general ideas: arithmetic subtraction by wiring and use of the (binary) sign of the result to operate a simple binary switch. In Figure 6d, arithmetic subtraction and mirroring are combined to produce n copies of the complement $\bar{x}_1$. These examples are simple functions of one variable; other multivariable constructs are available by wire connection at the input.

Figure 7's T-gate schematic illustrates the scheme's potential generality, its logical completeness and extension to multiple variables. The figure adds only a transmission gate (or pass-transistor switch) to the elements introduced in Figure 6. A simple nMOS version is shown for its simplicity and because most applications can tolerate its imperfections.

An applications overview

Memory. One obvious virtue of multiple-valued data representation is its potential for reducing the number of lines required for the parallel transmission of large amounts of data. The intense need for compaction in memory arrays has led to several commercial memory developments using multivalued data coding.[8]

Most prominent among these are several commercial ROM structures. Intel has used four-valued ROMs in two commercial products—the 8087 arithmetic coprocessor and a 432-03 peripheral-processor product. In each case, two-bit encoding reduced ROM array size enough to significantly increase array product viability. Motorola and General Instrument also report using four-valued ROM in somewhat more specialized applications—for example, driving a speech-synthesizer chip in an electronic toy.

These implementations use two distinct approaches. Intel's designs use a conventional IC process with four different channel widths for the ROM-cell MOS device. Even though the cell site must be larger to accommodate the largest transistor (conceptually three times as wide as in the binary case), the reduction by two in the number of cell sites and data lines produced an important reduction in array size. While the detection circuitry for four values of sense current is more complex, its inherent sharing in a memory environment saves considerable space overall.

On the other hand, in one of the Motorola designs and in the General Instrument designs, process complexity was increased to allow multithreshold MOS devices. Since the voltage dimension rather than a spatial dimension was used for coding, the resulting array was conceptually as small as a regular binary one. Furthermore, because of the relatively low speed of the application, General Instrument was able to use a very simple voltage-level scanning technique, which was slow but very economical of physical space.

Concerning other (more dynamic) applications of multiple-valued coding in memory, two CCD four-valued serial memory designs have been reported, one by IBM and one by Mitsubishi.[3] The general idea, multiple values of charge in a dynamic memory environment, has been extended in a pre-commercial study of a 16-valued RAM prototype reported at ISSCC by Hitachi.[6]

Communications. Specialists in relatively long-distance communications have long recognized the potential for information compaction and/or bandwidth reduction of nonbinary digital-signal

transmission. Thus, communications designers evolved, quite separately from the development of multiple-valued logic, a signaling form they call multilevel coding. The ugly realities of transmission media make received signals less than ideal and detection a messy process, but potential exists for application of multiple-valued techniques in the pre- and post-processing of signals.

Concerning commercial communications applications of a somewhat less-global scale, at least one continuing product, introduced by Motorola in 1980, comes to mind. The MC 145026/7/8 is a customizable family of CMOS chip pairs used for remote-control applications. Data is transmitted between the chips as a serial stream in which information is encoded in four-valued units consisting of two binary digits. However, implementing the required reliability in self-clocking uses only three of the four values. Each chip has nine pins for data and/or address, which

T-gate applications

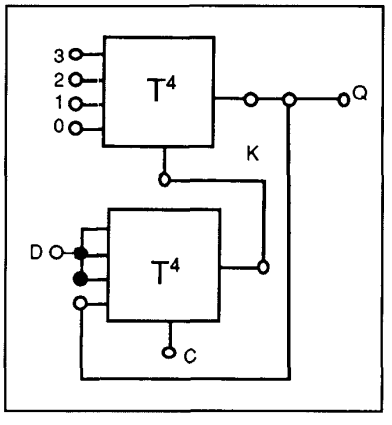
Single-variable functions. The T-gate, being a multiplexor under the control of the *r*-valued selecting input, can be used directly to implement any function of a single *r*-valued variable connected to its selecting input. For this purpose, the selected input corresponding to each selecting-input value is connected to a reference value appropriate to the output required by the desired single-variable function. Some four-valued examples are provided in the table below.

The idea can be extended to the formation of sequential machines. The figure at top right shows an arrangement that implements a simple four-valued D-type multiflop. Notice that a four-valued T-gate is used for gating, the feedback loop being closed when *C* is zero and the *D* input selected when *C* is one (or two or three). In this application, the upper T-gate functions as a four-valued quantizer for signal restoration.

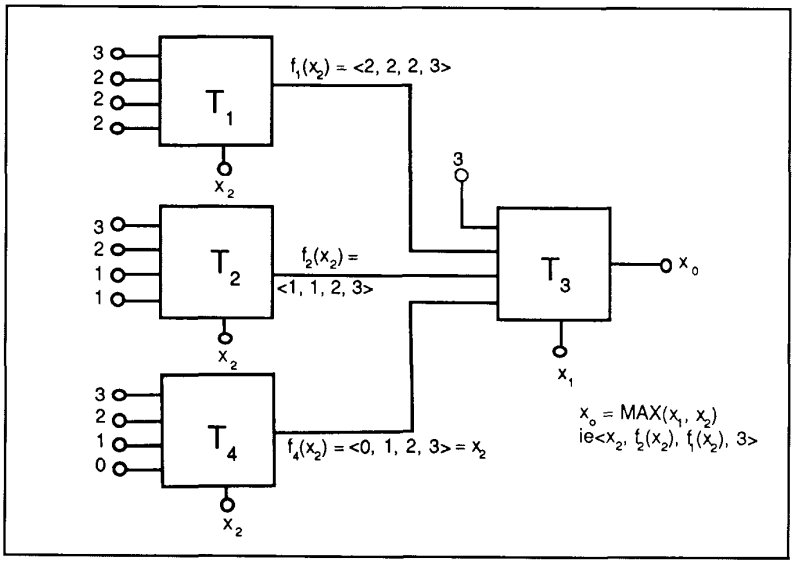
Multiple-variable functions. All functions of two or more variables can be constructed from T-gates. For example, the figure at bottom right presents a direct implementation of a four-valued MAX function of two variables using T-gates. Its output value is always the larger of two input values presented. The implementation is intended to correspond to a two-variable truth table in which x_1 is tabulated as the more global variable. Note that the fourth gate (T_4) is used only to provide a completely level-restored output, since only the controlling or selecting input of a T-gate is inherently level-restoring; for the other selected inputs, any input imperfection is propagated and (depending on implementation specifics) is possibly contaminated further.

Using the T-gate to implement all functions of one variable—selected examples.

x_1	x_2	x_3	x_4	$x_0 = f(x_k)$	Function	Symbol
0	1	2	3	<0,1,2,3>	Restoration	x_k
3	2	1	0	<3,2,1,0>	Complement	$\overline{x_k}$
3	0	3	0	<3,0,3,0>	Literal	$x_k^{0,2}$
2	2	2	2	<2,2,2,2>	Constant	2
0	0	3	3	<0,0,3,3>	Threshold	$x_k^{2,3}$



A simple D-type multiflop using T-gates, where T′ denotes a T-gate having an *r*-valued control input. Note that signals D, Q, and K are four-valued, while C is a two-valued clock in this four-valued example.



A T-gate implementation of the MAX function of two variables in a four-valued logic system.

can be used conventionally in a binary mode, but which also function in a ternary mode if desired. An open condition signals the third state (½) for input and output, the latter being the conventional tristate mode, well-known to binary designers. While the feature is quite simple and optional, the result is quite spectacular. Using multiple-valued coding can extend the chip pair's addressing range from $2^9 = 512$ to $3^9 = 19{,}683$, a dramatic improvement in a low-cost, pin-limited DIP environment.

Note that the tristate feature mentioned above, while ensconced unquestioningly in the binary world, is a multiple-valued idea in several senses. Moreover, in terms of the voltage versus current views of logic signaling described earlier, Motorola applies it in an "intimate" hybrid sense: the outer levels are voltage-driven by a gate output, while the inner (middle) level is current-driven by the gate output, with the corresponding voltage defined by the receiving input.

Testing. Three related areas of binary design needing more than a binary description are digital-signal dynamics, testing, and fault analysis. In response to this long-recognized need, a body of work has grown in which multiple-valued techniques play a part. Initially, the role was simply notational, for example, making explicit the transient (or otherwise undefined) state between zero and one. Gradually, more intervening conditions needed explicit consideration. Finally, algebraic aspects of multivalued logic were brought to bear in providing more formal tools for presentation and computerization of relevant tools.

Recently, for example, Hayes[9] developed a logic-circuit description that accurately represents the structure and behavior of MOS circuits using a $(3n+1)$-valued pseudo-Boolean algebra, where $n \geq 1$ is the number of fault "strengths" (ranging from open to short) to be modeled.

Another potential MVL application is in designing multiple-valued circuits that, while used primarily in the binary mode, exhibit and propagate failures in real time by transition to a multiple-valued mode. This idea has been exhibited by Hu[10] in actual circuits, although admittedly in a far from commercially viable form.

Arithmetic. The existence of a natural relationship between arithmetic and multiple-valued logic is a relatively old idea, exhibited in several quite different ways. I've already hinted at one way in the commentary on current-mode circuits, where current-mode summation of multiple-valued signals (as of analog signals) becomes quite trivial through wiring. In a quite different direction, but also an old idea, is the notion of multiple-valued recoding of binary signals to eliminate accumulating carry-propagation times in arithmetic processes. One such scheme is signed-quaternary, a seven-valued variant on four-valued signaling using the values $-3, -2, -1, 0, 1, 2$, and 3.

Another interesting attribute of multivalued coding in arithmetic is that truncated odd-multivalued encodings (3,5, etc.) produce unbiased results; that is, the less-significant part abandoned is equally often larger or smaller than one in the last-retained place. In binary (and other even bases), the rounding operation to produce an unbiased result is much more complex. Finally, because arrays of basic elements are common in arithmetic applications, the potential for increased use of each connecting link using multiple-valued signaling is particularly attractive.[3,4]

Broadly stated, the goal of this tutorial has been to stir the nonspecialist reader's interest in multiple-valued representation and logic. It has not been a survey, for that is either a more extensive or more intensive task. Rather, it has attempted to put into context, particularly for one proficient in binary systems, some of the major issues enveloping the topic. Clearly, more could be said; of this, much is presented in the specific context of the following articles. □

Acknowledgment

This work has been supported by the Natural Sciences and Engineering Research Council of Canada through Operating Grant A1753.

References

1. D.C. Rine, ed., *Computer Science and Multiple-Valued Logic, Theory and Applications*, 2nd edition, North-Holland, Amsterdam, 1984.
2. K.C. Smith, "The Prospects for Multivalued Logic: A Technology and Applications View," *IEEE Trans. Computers*, Sept. 1981, pp. 619-634.
3. S.L. Hurst, "Multiple-Valued Logic—Its Status and Its Future," *IEEE Trans. Computers*, Dec. 1984, pp. 1160-1178.
4. S. Kawahito et al., "A High-Speed Compact Multiplier Based on Multiple-Valued Bidirectional Current-Mode Circuits," *Proc. IEEE 17th Int'l. Symp. Multiple-Valued Logic*, Boston, May 1987, pp. 172-180.
5. S.P. Onneweer and H.G. Kerkhoff, "High-Radix Current-Mode CMOS Circuits Based on the Truncated Difference Operator", *Proc. IEEE 17th Int'l Symp. Multiple-Valued Logic*, Boston, May 1987, pp. 188-195.
6. M. Aoki et al., "A 16-Level/Cell Dynamic Memory," *IEEE ISSCC Digest*, Feb. 1985, pp. 246-247.
7. T.T. Dao, "Threshold I^2L and Its Applications to Binary Symmetric Functions and Multivalued Logic," *IEEE J. Solid-State Circuits*, Oct. 1977, pp. 463-472.
8. *D.A. Rich, "Survey of Multivalued Memories," IEEE Trans. Computers*, Feb. 1986, pp. 99-106.
9. J.P. Hayes, "Pseudo-Boolean Logic Circuits," *IEEE Trans. Computers*, July 1986, pp. 602-612.
10. M. Hu, "A Novel Built-In Test Technique for CMOS VLSI Circuits," *Proc. IEEE 16th Int'l Symp. Multiple-Valued Logic*, Blacksburg, Va., May 1986, pp. 80-84.

Kenneth C. Smith is a professor in the Departments of Electrical Engineering, of Computer Science, and of Library and Information Science at the University of Toronto. His research interests are in linear and digital electronics, computer architecture, instrumentation, and human factors. He is the author or coauthor of well over 100 journal and symposium papers and a contributor to several books, including *Microelectronics Circuits*, second edition, coauthored with Adel S. Sedra. He is also an advisor to corporations in the US and Canada.

An IEEE fellow, Smith received his BA, MA, and PhD degrees in engineering science, electrical engineering, and physics, respectively, from the University of Toronto.

Smith's mailing address is Dept. of Electrical Engineering, University of Toronto, Toronto, Ontario, M5S 1A4, Canada.