More Efficient Radix-2 Algorithms for Some Elementary Functions

P. W. BAKER

Abstract—**de Lugish [1] has defined efficient algorithms in radix 2 for certain elementary functions such as Y/X, $Y/X^{1/2}$, $Y + \ln X$, $Y \cdot \exp(X)$, etc. His technique requires a systematic 1-bit left shift of a partially converged result, together with two 4-bit comparisons to select a ternary digit for the next iteration. This selection of digits reduces the average number of full precision additions to about 1/3 of those required in conventional schemes [3]. This paper develops modified algorithms in radix 2 which are more efficient when the time for a full precision addition is comparable to the time for a shift and comparison. The modified procedure is developed for Y/X in detail where more than a 40 percent decrease in execution time is achieved for only a marginal increase in cost.**

Index Terms—**Digital arithmetic, elementary functions, iterative algorithms, radix 2, variable left shift.**

INTRODUCTION

DE LUGISH [1] has defined efficient algorithms in radix 2 for the functions $Y/X, Y/X^{1/2}$, $\ln X$, $\exp(X)$, the trigonometric functions, and the inverse trigonometric functions. These algorithms are based on the continued product normalization procedure which uses multipliers of the form $(1 + s_k 2^{-k})$ so that multiplication is reduced to a shift-add sequence. Chen [2] has also discussed a scheme for the automatic computation of the first four mentioned functions. He uses a bit counting technique which reduces the number of shift-add sequences, on the average, to about $n/2$. More recently, Ercegovac [4] has used a radix-16 approach to speed up the computation of reciprocals, logarithms, and exponentials. He estimates that the radix-16 algorithms take about 3/4 of the time of the corresponding de Lugish algorithms, but are less efficient when cost is taken into account.

This paper develops radix-2 algorithms which use an extension of de Lugish's technique and which are more efficient when the time for a full precision addition is comparable to the time for a shift and low precision comparison.

DE LUGISH'S METHOD

As a vehicle for discussion, we take the algorithm for Y/X where Y and X are n-bit numbers, which is typical of the whole class and which may be described as follows:

$$X_0 = X,\ Y_0 = Y, \qquad 0.5 \le X, Y < 1 \tag{1}$$

$$X_{k+1} = X_k(1 + s_k 2^{-k-1}), X_k \to 1, \qquad 0 \le k < n \tag{2}$$

$$Y_{k+1} = Y_k(1 + s_k 2^{-k-1}), Y_k \to Y/X \text{ as } k \to \infty. \tag{3}$$

Instead of using (2), de Lugish substitutes

$$R_k = (X_k - 1) \cdot 2^k \tag{4}$$

$$R_{k+1} = 2R_k + s_k + s_k R_k 2^{-k} \tag{5}$$

and selects s_k, $k \ge 1$, according to the following:

$$s_k = \begin{cases} 1, & \text{if } R_k < -3/8 \\ -1, & \text{if } R_k \ge 3/8, \qquad k \ge 1 \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

For $k = 0$,

$$s_0 = \begin{cases} 2, & \text{if } -1/2 \le R_0 < -1/4 \\ 0, & \text{if } -1/4 \le R_0 < 0. \end{cases}$$

This method of selecting s_k yields a probability of 2/3 for selecting $s_k = 0$ (see [1]) and leads to the following properties of R_k:

$$|R_k| < 1, \qquad \text{for } k \ge 0$$
$$-3/4 \le R_k < 3/4, \qquad \text{for } k \ge 2. \tag{7}$$

A hardware configuration to compute R_k for $k \ge 1$ according to (5) is shown in Fig. 1. The transformation from R_0 to R_1 may be regarded as a separate initialization process and will not be considered here. The evaluation of Y_k will take place in a similar unit controlled by the unit shown in Fig. 1. Each iteration requires a single left shift of R and two comparisons to determine the value of s_k. If $s_k = 0$, the loop 1 is taken, bypassing both the variable shifter and adder. The two comparisons may be performed conveniently as shown by feeding the four high-order bits of R into a small read-only memory (ROM). The output will be needed to control the clocking pulses for register R. If $s_k = 0$, a clocking pulse may occur after the settling time of the components in loop 1. If $s_k = \pm 1$, the clock pulse must wait for the extra delay through the components of loop 2.

When the time for an addition is much greater than that for a shift and compare, the above scheme will compute an n-bit result in about $n/3$ addition times. This will be a significant improvement over schemes which use $n/2$ [2] or n [3] additions. However, it is possible to construct

Manuscript received April 29, 1974; revised April 29, 1975. This work was supported by the Australian Research Grants Committee.

The author is with the Department of Computer Science, School of Electrical Engineering, University of New South Wales, Kensington, N. S. W., Australia.

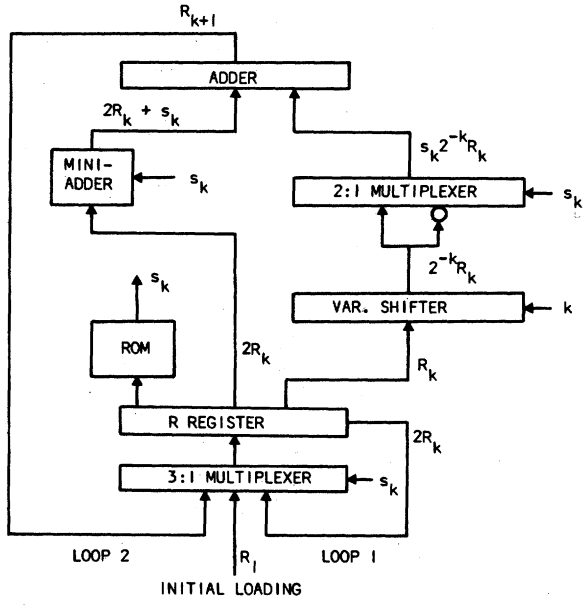


Fig. 1. R_k normalization configuration for de Lugish's method.

adders using carry lookahead techniques to speed up addition; if carry lookahead may occur over up to 8 bits and over up to 8 groups of bits, then an addition of up to 64 bits precision may take place in only 10 gate delays. Notwithstanding considerations of extra delays due to high fan-out, this means that the delay for an addition will be comparable to the delay of a small ROM.

In order to obtain a timing estimate for Fig. 1, we assume that the various components have the following delays in terms of gates, each gate (AND, OR, and NOT) having an assigned delay of one unit: master-slave register—4 gates; 2 (or 3) to 1 multiplexer (mplxr), address to output—4 gates; 2:1 mplxr, data in to output—2 gates; 1 to n-bit variable shifter—7 gates, mini-adder—3 gates. We also assume that there will be an extra gate delay for the address input to the 3:1 mplxr associated with the initial loading of the R register. The timing estimates for loops 1 and 2 of Fig. 1 are as follows.

Loop 1:

ROM	6
3:1 multiplexer	4
R register	4
extra	1
Total =	15 gates.

Loop 2:

ROM	6, variable shifter	7
(mini-adder	3), 2:1 mplxr	3
adder	10	
3:1 mplxr	2	
R register	4	
Total =	26 gates.	

Observe that in loop 2, the address input to the 3:1 mplxr settles before the data lines. Hence the 3:1 mplxr's contribution to the loop 2 delay is only 2 gates. The total delay in generating an n-bit result ($n \leq 56$) will be about $n/3 \cdot 26 + 2n/3 \cdot 15 \simeq 18.66\ n$ gates. Observe that more than half the time is taken up by loop 1. An extension to de Lugish's scheme is now proposed which eliminates loop 1 but does not excessively increase the delay through loop 2.

MODIFIED CONFIGURATION

Instead of having a single bit shift in loop 1, we inspect the m high-order bits of R_k (including the sign bit) to obtain

1) a left-shift value j which lies between 0 and $m - 1$, and

2) the value of $\sigma_k \in \{-1,1\}$

which are used in the modified recursion equations[1]

$$R_{k+j} := 2^j R_k + \sigma_k + \sigma_k R_k 2^{-k} \tag{8}$$

$$Y_{k+j} := Y_k(1 + \sigma_k 2^{-k-j}) \tag{9}$$

with the selection rule

$$\sigma_k = \begin{cases} 1, & \text{if } R_k < 0 \\ -1, & \text{if } R_k \geq 0. \end{cases}$$

The practical choices for m, as will be shown, are 6, 7, or 8.

This modified scheme is shown in Fig. 2 where a ROM is used to store j. This value of j is used to control a variable left-shift network (from 0 to $m - 1$ bits) through which R_k passes before it enters loop 2. For $m = 6$, the contents of the ROM, whose address input is the 6 high-order bits of R_k, are shown in Table I where x denotes either 0 or 1. Note that the sign of σ_k (0 for + and 1 for minus) is always equal to the one's complement of the leading bit (the sign bit) of R_k. Hence σ_k is available as soon as the R register has settled and need not be stored in the ROM. With $6 \leq m \leq 8$, the ROM word length will be 3 bits to accommodate j in binary code.

ROM Lookup Table

The action to be taken for all the 2^m possible leading bits of R_k will be discussed for the case $m = 6$ and with reference to Table I. The contents of Table I, for $j \geq 1$, may be justified from (6) as follows. Because of the symmetry of Table I, only the case $R_k \geq 0$ will be considered. For $j = 1$, the action taken is identical to that in (6) and needs no explanation.

For

$$R_k < 3 \cdot 2^{-(j+1)}, \qquad j \geq 2$$

it is easily seen that, in de Lugish's method,

$$s_k = s_{k+1} = \cdots = s_{k+j-2} = 0.$$

Moreover, if

$$3 \cdot 2^{-(j+2)} \leq R_k < 3 \cdot 2^{-(j+1)}$$

then

[1] Note that we have used the assignment operator (:=) to avoid the contradictions that would arise in (8) and (9) when $j = 0$ and if the equality sign (=) were used.

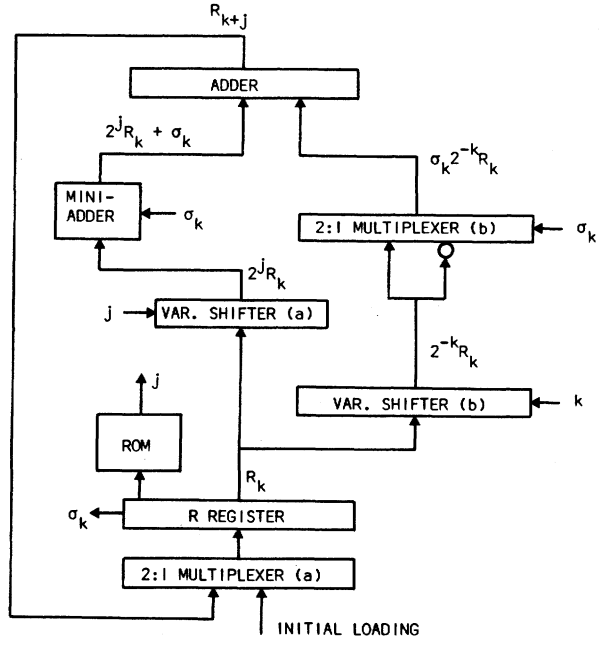


Fig. 2. R_k normalization configuration for modified method.

TABLE I

	High-order bits of R_k	Number of bits j to be passed over
A	0.00000	5
	0.00001	4
	0.00010	4
	0.00011	3
	0.0010x	3
	0.0011x	2
B	0.010xx	2
	0.011xx	1
C	0.10xxx	1
	0.11xxx	0
	1.00xxx	0
	1.01xxx	1
B	1.100xx	1
	1.101xx	2
A	1.1100x	2
	1.1101x	3
	1.11100	3
	1.11101	4
	1.11110	4
	1.11111	5

Note: x can be either 0 or 1.

$$s_{k+j-1} = -1.$$

So the unsuccessful trials on k to $k + j - 2$ (which yield s_k, etc., $= 0$) can be avoided if R_k is shifted over j bits with $s_{k+j-1} = \sigma_k = -1$.

For the second entry in Table I, $j = 4$ must be chosen since $R_k \geq 3 \cdot 2^{-6}$ is possible. Similar arguments apply to the first entry.

The choice of $j = 0$ for $R_k = 0.11xxx\cdots$ or $R_k = 1.00xxx\cdots$ may be justified as an attempt to produce implicit minimal recoding when $k \gg 1$. (By minimal recoding, we mean a recoding that will reduce the number of $s_k \neq 0$ and hence shift and add operations to a minimum.)

TABLE II

m	$1/\langle s \rangle$ Theoretical	Experimental
5	0.422	—
6	0.376	0.372
7	0.354	0.352
8	0.344	0.342
9	0.339	—
10	0.336	—

For example, if $R_k = 0.11111\cdots$, the resultant value is $R_{k+0} := 1.11111\cdots$ (2's complement). The next iteration will now produce a maximum shift of 5. If we chose $j = 1$ instead, the following calculation would result:

$$R_{k+1} := 1.11111\cdots$$

$$:= \frac{-1}{0.11111}\cdots$$

and the next and subsequent shift values would only be $j = 1$ instead of $j = 5$.

A derivation of the bound on R_k proceeds as follows: de Lugish [1] has shown that $R_1 \in [-\frac{1}{2},1)$. By applying (8) to the ranges of R_k corresponding to all the values of j in Table I, it can be seen that the extreme values of R_{1+j} arise when $j = 5$ and

1) $R_1 \in ([0,2^{-5})$ or

2) $R_1 \in [-2^{-5},-0)$.

Using (8), the range $[0,2^{-5})$ maps onto $[-1,-2^{-6})$ and the range $[-2^{-5},-0)$ maps onto $[-2^{-6},1)$. So $R_6 \in [-1,1)$. It can also be seen that $R_k, 2 \leq k \leq 5, \in (-c,c)$ where $c > 2^{-5}$. A proof by induction that $R_k \in [-1,1)$, $k > 6$ is now obvious. Hence, for the modified algorithm,

$$-1 \leq R_k < 1, \qquad \text{for all } k. \tag{10}$$

Average Number of Iterations

The average number of iterations N_{av} is related to the average shift number $\langle s \rangle$. If P_0 denotes the probability of choosing $s_k = 0$, then

$$\langle s \rangle = \frac{1}{1 - P_0}.$$

For an n-bit word, $n \to \infty$,

$$N_{av} = \frac{n}{\langle s \rangle}.$$

In the Appendix, an expression is derived for $\langle s \rangle$ as a function of m, $m \geq 5$, assuming $n = \infty$. Theoretical values of $1/\langle s \rangle$ for $m \geq 5$ are given in Table II. These values were verified by simulating 5000 trials on 60 bit operands uniformly distributed over $(\frac{1}{2},1)$. These corresponding experimental estimates of $1/\langle s \rangle$ are also reported in Table II.

Speed of Modified Scheme

In order to obtain a timing estimate for Fig. 2, we employ the same assumptions used for Fig. 1. Since $\sigma_k = 0$ is not allowed, loop 1 of Fig. 1 may be eliminated, as shown in Fig. 2. The delay for one iteration in Fig. 2 is made up of

ROM	6
variable shifter (a)	5
adder	10
2:1 mplxr (a)	2
R register	4
Total =	27 gates.

Note that the delay for the mini-adder is not included. The mini-adder causes no extra delay because the carry into the most significant adder block from the group carry generate block is 5 gates; and the mini-adder only has a delay of 3 gates. The variable shifter (a) is merely an $(m - 1)$ to 1 multiplexer whose delay from address to output we assume to be 5 gates.

If an 8-bit ROM lookup is used, an n-bit result, on the average, will be computed in about $9.29n$ gates. From Table II, it can be seen that little is gained by consulting more than 8 leading bits of R_k, but that a significant extra number of iterations will be needed for $m < 6$.

The efficiency of the modified scheme is derived from the fact that the normalization procedure allows updating of the iteration counter k after an iteration. However, the result evaluation, (9), requires k to be updated with j before the current iteration can be completed. In order not to lose speed, then the best procedure is to stagger the result calculation one step behind the normalization calculation. If this is done, the generation of Y/X will require an extra iteration, yielding a total delay of $9.29\ (n + 3)$ gates. For $m = 6$, the average delay is $10.15\ (n + 3)$ gates. Therefore, the modified scheme will generate Y/X in less than 60 percent of the time needed for de Lugish's scheme. The cost increase, which is about m gates due to the 0 to $m - 1$ variable shifter, is less than 10 percent for a parallel implementation. As well as making for faster execution, the configuration in Fig. 2 will require simpler control circuitry, since only one train of synchronous clock pulses will be required for clocking the R register.

The timing estimates for the above-mentioned schemes assume that the delay of the ROM is 6 gates, independent of m. The speed of commercially available ROM's tends to decrease as their bit count increases, and so the timing comparisons will tend to favor the modified scheme. Even so, a comparison of the two schemes, assuming an implementation in TTL logic, yields a speed improvement in the modified method of greater than 40 percent.

With a variable shift number j available at every iteration, the counter, which is normally incremented by one, will now require the capacity to be incremented by an amount up to 7. The most convenient realization of this counter is a 6-bit fast adder in a loop with a master–slave register. This register can be clocked by the same pulse which clocks the R register.

EXTENSION TO OTHER ELEMENTARY FUNCTIONS

The above modification to the generation of Y/X may be extended to other elementary functions. Some of these will be discussed briefly.

1) The generation of $Y + \ln X$ uses the same normalization equation (8) and only requires (9) to be replaced with

$$Y_{k+j} := Y_k - \ln (1 + \sigma_k 2^{-(k+j)}).$$

2) The generation of $Y \cdot X$ is straightforward, and the same ROM can be used to select the shift value j.

3) The generation of $Y \cdot \exp (X)$ also uses the same ROM to select j, but complications arise because the normalizing equation

$$R_{k+j} := 2^j R_k - 2^{k+j} \ln (1 + \sigma_k 2^{-(k+j)})$$

requires the counter to be incremented at the start of an iteration.

4) The multiplicative square root algorithm is similar to the Y/X algorithm and the same lookup table may be used. The speed increase will not be as great as the Y/X case, since the computational loop will contain two cascaded full precision adders (see [1]).

The additive algorithm for square root will require only one full adder, but will need up to 5 ROM tables to cover the initial condition range $X \in [\frac{1}{4},1)$ (see [1]).

DISCUSSION AND CONCLUSION

The timing comparison of the two schemes for Y/X are based on the fastest practical Boolean realizations of the various components. Significantly different results might be obtained if a hardware unit were to be built using commercially available logic.

These modified minimal recoding algorithms of de Lugish will have at least two advantages over algorithms that use Chen's bit counting technique [2]. First, Chen's bit counting will be less efficient than the table method presented herein. In Chen's scheme, the number of bits to be counted at the kth iteration is about k or more, which may be time consuming for large k. Second, although the average number of iterations in Chen's scheme is about $n/2$, the maximum possible value is n. In de Lugish's scheme, minimal recoding selection of s_k should yield a maximum of $(n + 1)/2$. This derives from the fact that with *canonical* recoding, each nonzero digit is separated by at least one zero. We conjecture that this maximum holds in the modified method for $m \geq 6$. This conjecture is supported by the following reasoning. For the maximum number of iterations to be achieved, the shifts per selection of σ_k must be a combination that results in an average shift of 2 bits for each selection. If any one selection yields, say, $j = 5$, with a next shift value of $j = 0$, then the

combined shift distance for the two selections is still greater than 4.

Hence these modified de Lugish algorithms should offer a speed improvement factor of at least 2 when worst case computation time is important.

APPENDIX

CALCULATION OF AVERAGE SHIFT VALUE $\langle s \rangle$

Given a table with 2^m entries, let $P(i)$ be the probability that the ith entry will be accessed and let $j(i)$ be the shift value corresponding to that ith entry. Then the shift average is

$$\langle s \rangle = \sum_{i=1}^{2^m} P(i) \cdot j(i). \tag{11}$$

In order to determine $\langle s \rangle$ then, expressions for $P(i)$ must be obtained. These expressions will be derived for the case $n = \infty$. As a prelude to the general development, we discuss the case for $m = 6$ with reference to Table I.

For the limiting case $n = \infty$, and as $k \to n$, the third term on the right in (8) may be ignored, yielding

$$R_{k+j} = 2^j R_k + \sigma_k, \qquad k \to \infty. \tag{12}$$

Considering all the table entries, excluding the first and last two, it can be seen that R_{k+j}, as computed by (12), will be in one of two ranges:

$$\begin{aligned} &\text{range } A\text{: } 0 \leq |R_{k+j}| < 2^{-2} \\ &\text{range } B\text{: } 2^{-2} \leq |R_{k+j}| < 2^{-1}. \end{aligned} \tag{13}$$

The result will be in range A when

$$R_k = \pm 0.\underbrace{00\cdots}_{\text{0 to 3 zeros}}11xx\cdots$$

or

$$R_k = \pm 0.00\cdots0100xx\cdots$$

and in range B when

$$R_k = \pm 0.00\cdots0101xx\cdots.$$

For the second and second last entry, the result will be in range A if $R_k = \pm 0.000011xx\cdots$ and in range B if $R_k = \pm 0.000010xx\cdots$. Considering now the first and last entries, it can be seen that if $R_k = \pm 0.000000xx\cdots$, then R_{k+j} will lie in

$$\text{range } C\text{: } 2^{-1} \leq |R_{k+j}| \leq 1. \tag{14}$$

Ranges A, B, and C are marked in Table I. If the less significant bit patterns of each entry are equally likely, then each A entry will have the same access probability P_A, each B entry a probability P_B, and each C entry a probability P_C. In order to determine expressions for these probabilities, we proceed as follows.

Consider the repeated use of (12) for $k \leq n$. Since $n = \infty$, the number of table accesses, which we denote by N_T, will also be ∞. If we let N_A, N_B, and N_C be the number of times entries in the ranges A, B, and C are accessed, then

$$N_T = N_A + N_B + N_C.$$

Now, for a general table of 2^m entries, the number of B entries = the number of A entries = 2^{m-2}, and the number of C entries = 2^{m-1}. For tables where $m \geq 5$, the following expressions may be obtained for N_B and N_C:

$$N_B = 2N_T \cdot \{2^{m-4} \cdot P_C + 2^{m-5} \cdot P_B + P_A \cdot (2^{m-5} + 2^{-2})\} \tag{15}$$

$$N_C = P_A \cdot N_T. \tag{16}$$

From the definition of probability, we get

$$P_B = N_B/(N_T \cdot 2^{m-2}) \tag{17}$$

and

$$P_C = N_C/(N_T \cdot 2^{m-1}) \tag{18}$$

$$= P_A \cdot 2^{-m+1}. \tag{19}$$

But, since the probabilities of all the table entries must add up to 1, we get

$$P_A + P_B + 2P_C = 2^{-m+2}. \tag{20}$$

So from (15)–(19) we get

$$P_B = \alpha(m) P_A \tag{21}$$

where

$$\alpha(m) = 2^{-m+2} + \tfrac{1}{3}. \tag{22}$$

From (19)–(21) we get

$$P_A = 2^{-m+2} \cdot (1 + \alpha(m) + 2^{-m+2})^{-1}. \tag{23}$$

A detailed examination of an m-bit table shows that the following expression for $\langle s \rangle$, which is an explicit version of (11), holds for $m \geq 5$:

$$\begin{aligned} \langle s \rangle = {} & 2 \cdot P_A \cdot \{(m-1) + 2(m-2) + \beta(m) + 2^{m-4}\} \\ & + 2 \cdot P_B \cdot \{3 \cdot 2^{m-4}\} + 2 \cdot P_C \cdot \{2^{m-3}\} \end{aligned} \tag{24}$$

where

$$\beta(5) = 0$$

and

$$\beta(m) = \sum_{i=1}^{m-5} 3 \cdot 2^{i-1} \cdot (m - 2 - i), \qquad m > 5. \tag{25}$$

This expression for $\langle s \rangle$, with P_A, P_B, and P_C evaluated using (22), (23), (21), and (19) was used to give the results in Table II for $5 \leq m \leq 10$. As $m \to \infty$, $\langle s \rangle \to 3$, as in de Lugish's method.

ACKNOWLEDGMENT

The author would like to thank his colleagues for their assistance during the preparation of this paper. The author is particularly indebted to the referees, whose detailed critiques were responsible for significant improvements in the manuscript.

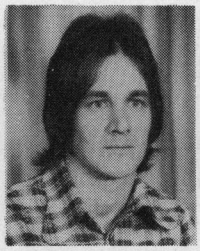
REFERENCES

[1] B. G. de Lugish, "A class of algorithms for automatic evaluation of certain elementary functions in a binary computer," Ph.D. dissertation, Dep. Comput. Sci., Univ. Illinois, Urbana, June 1970.

[2] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios and square roots," *IBM J. Res. Develop.*, vol. 16, pp. 380–388, July 1972.

[3] W. H. Specker, "A class of algorithms for ln x, exp x, sin x, cos x, $\tan^{-1} x$ and $\cot^{-1} x$," *IEEE Trans. Electron. Comput.* (Short Notes), vol. EC-14, pp. 85–86, Feb. 1965.

[4] M. D. Ercegovac, "Radix-16 evaluation of certain elementary functions," *IEEE Trans. Comput.*, vol. C-22, pp. 561–566, June 1973.



P. W. Baker was born in Melbourne, Vic., Australia, on July 24, 1946. He received the B.E. degree from the University of New South Wales, Kensington, N.S.W., Australia, in 1971, and has recently completed research work for the Ph.D. degree in computer science.

Since October 1974 he has held the position of Lecturer in the Department of Computer Science, University of New South Wales. His research interests include computer hardware, computer arithmetic, and differential equations.

Equational Realizations of Switching Functions

FRANK M. BROWN

***Abstract*—Equational logic is an approach to combinational synthesis based on the equation $f(x) = 1$ rather than on the function $f(x)$. The central problem of equational logic is to find a system of equations $g_i(x) = h_i(x)$ $(i = 1,2,\cdots,k)$, of the simplest possible form, that has the same solutions as $f(x) = 1$. Given such a k-equation system, $f(x)$ may be realized as the output of a k-wide digital comparator whose inputs are the $2k$ g's and h's constituting the system.**

This paper continues the investigation of equational logic begun in [5], where it was shown that the equivalence of the equation $f(x) = 1$ to a system of equations is intimately tied to the existence of a bilinear representation for $f(x)$. The concept of a bilinear representation is employed in the present paper to develop two approaches to equational synthesis, namely, separation of arguments and equation solving.

***Index Terms*—Boolean equations, Boolean matrices, combinational logic, digital comparators, functional decomposition.**

I. INTRODUCTION

A COMBINATIONAL circuit realizing a switching function $f(x)$ may be thought of as a solution verifier for the Boolean equation

$$f(x) = 1. \tag{1}$$

The circuit's output has the value 1, that is, if and only if the input vector x is a solution for (1).

Suppose (1) is equivalent to (i.e., has the same solutions as) a k-equation system of the form

$$\begin{aligned} g_1(x) &= h_1(x) \\ &\cdot \\ &\cdot \\ &\cdot \\ g_k(x) &= h_k(x). \end{aligned} \tag{2}$$

Then $f(x)$ may be realized by the structure shown in Fig. 1. The unit labeled "$\equiv$" is a k-wide digital comparator (conveniently implemented with k open-collector EXCLUSIVE-NOR gates); the unit labeled "function generator," which produces the g's and h's of (2), is a multiple-output combinational circuit.

We shall refer to the structure of Fig. 1 as an *equational realization* of $f(x)$. Suppose, for example, that it is desired to realize the function

$$f = \bar{B}\bar{C}\bar{E} + \bar{A}\bar{B}D + ABD + A\bar{C}\bar{E} + AD\bar{E} + BCDE + AC\bar{D}E + BC\bar{D}\bar{E} + \bar{A}B\bar{C}\bar{D}E,$$

which is expressed in simplified sum-of-products form. The two-equation system

$$\begin{aligned} AB + C\bar{D} &= B\bar{E} + AE \\ \bar{A}B\bar{C} &= \bar{C}\bar{D}E \end{aligned} \tag{3}$$

is equivalent to the equation $f = 1$, but clearly has simpler form. An equational realization of f, corresponding to the system (3), is shown in Fig. 2.

For simplicity, the cost of an equational realization will

Manuscript received September 5, 1974; revised March 15, 1975.

The author is with the Department of Electrical Engineering, University of Kentucky, Lexington, Ky. 40506.