The Quasi-Serial Multiplier

EARL E. SWARTZLANDER, JR.

Abstract—A novel technique for digital multiplication is presented that represents a considerable departure from conventional (i.e., add and shift or fully parallel) multiplication algorithms. The quasi-serial multiplier generates the bits of the product sequentially from least significant to most significant. Each bit is computed by "counting" the number of ones in the corresponding column of the bit-product matrix and adding the previous carrys. This single operation yields both the product bit and the carrys for the next column. The quasi-serial multiplier requires $2n$ of these count and add operations to determine the product of two n-bit numbers.

Realization of this new multiplier requires only one unconventional operation—counting the number of ones in the columns of the bit-product matrix. Consequently attention is focused on the practical implementation of ones counters, circuits that indicate how many of their inputs are in the logic ONE state. A new approach to counter implementation is presented that uses quasi-digital (i.e., current summing) processing.

Index Terms—Current summing counters, digital multipliers, fast multipliers, full-adder counters, multiplier speed-simplicity comparison, parallel counters, quasi-serial multiplier.

Introduction

SCIENTIFIC and engineering applications of digital computers often involve considerable multiplication, e.g., eigenvalue computation, matrix inversion, fast Fourier transformation, etc. It is especially important for designers of special-purpose digital computers to be cognizant of multiplication algorithms that are potentially more efficient (e.g., in terms of speed and/or simplicity of implementation) than those that are presently being used.

The most obvious way to multiply two positive binary numbers is to sequentially inspect the bits of the multiplier from least significant to most significant. If a given bit is a 1, the multiplicand is added to the most significant half of a double-length accumulator; if it is a 0, the addition is not performed. The contents of the accumulator are shifted one bit to the right as each bit of the multiplier is inspected. When all bits of the multiplier have been considered, the accumulator contains the product. This is a basic form of the well-known add and shift multiplication scheme.

In 1946, Burks *et al.* [1] developed a variation of this scheme for multiplying numbers of either sign (with negative numbers expressed in two's complement form). Their algorithm requires correction of the product if either (or both) the multiplier or multiplicand is negative. About five years later Booth [2] presented a similar algorithm that does not require any correction steps.

Since then many clever approaches have been devised to speed the multiplication process; most either involve reducing the number of cycles required by the add and shift scheme by making available multiples of the multiplicand and shifting over groups of bits [3] or else involve parallel multipliers [4]-[7]. The matrix of bit products (i.e., a_ib_j; on Fig. 1) is generated in a single step with a large array of logic AND gates. This matrix is reduced to an equivalent two-row matrix with a network of full-adder circuits. The two-row matrix is summed with a fast (e.g., carry-lookahead) adder to generate the product. Although parallel multipliers are fast, they do require considerable hardware that precludes their use for many applications. In this paper attention is focused on the quasi-serial multiplication scheme that is potentially faster than add and shift multipliers, but much simpler than parallel multipliers.

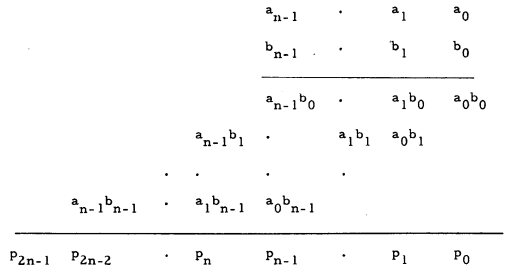


Fig. 1. The bit-product matrix.

In the following analysis, the sign and magnitude number system is used exclusively. An n-bit[1] number A is defined by

$$A = (-1)^{\sigma_a} \sum_{i=0}^{n-1} a_i 2^i \tag{1}$$

where σ_a the sign bit is 0 for $A \geqslant 0$ and is 1 for $A \leqslant 0$. The sign of the product of any two numbers A and B may be computed independently of the magnitude computation by

$$\sigma_p = \sigma_a \oplus \sigma_b \tag{2}$$

where $\oplus$ denotes modulo 2 addition (i.e., the EXCLUSIVE-OR logic function).

Derivation of the Algorithm

The magnitude of the product of any two n-bit numbers A and B is given by

$$|A \cdot B| = \sum_{i=0}^{n-1} a_i 2^i \sum_{j=0}^{n-1} b_j 2^j \tag{3}$$

or

Manuscript received March 17, 1971; revised October 16, 1972.
The author is with Hughes Aircraft Company, Culver City, Calif. 90230.

[1]The sign bit is not included when the bits comprising a number are counted.

$$|A \cdot B| = \sum_{j=0}^{n-1} \left(\sum_{i=0}^{n-1} a_i 2^i b_j \right) 2^j. \tag{4}$$

Equation (4) is the basis for the previously described add and shift algorithm as may be verified by inspection of Fig. 1. After the jth bit of the multiplier has been inspected the accumulator contains the sum of the first j rows of partial products.

Alternatively the product may be viewed in terms of the sums of the columns of the bit-product matrix.

$$|A \cdot B| = \sum_{i=0}^{2n-2} \left(\sum_{k=0}^{i} a_k b_{i-k} \right) 2^i$$

$$a_i, b_i = 0, \quad \text{for } i < 0 \text{ or } i > n-1. \tag{5}$$

Define

$$s_i = \sum_{k=0}^{i} a_k b_{i-k} \tag{6}$$

where s_i is the sum of the entries in the ith column of the partial-product matrix that is shown in Fig. 1. Thus

$$|A \cdot B| = \sum_{i=0}^{2n-2} s_i 2^i. \tag{7}$$

Since for columns 1 through $2n-3$, s_i may exceed 1, a simple recursion is used to form P, a $2n$-bit binary number that is equal in value to the product. Let

$$\left.\begin{aligned} p_0 &= s_0 \qquad c_0 = 0 \\ p_i &= (s_i + c_{i-1}) \bmod 2 \\ c_i &= (s_i + c_{i-1} - p_i)/2 \end{aligned}\right\} \text{for } i = 1, 2, \cdots, 2n-1. \tag{8}$$

This recursion sums the s_i term defined by (6), and thus reduces the sum for each column of the bit-product matrix to a product bit p_i and a carry term c_i. The product may be written in terms of the product magnitude bits and the sign bit from (8) and (2)

$$A \cdot B = (-1)^{\sigma_P} \sum_{i=0}^{2n-1} p_i 2^i. \tag{9}$$

Implementation

The algorithm defined by (2), (6), and (8) may be directly implemented with a $2n-1$ stage shift register, an n-bit holding register, n-logic AND gates, an n-input counter,[2] a small [i.e., Entier[3] $(\log_2(2n-1))$ bit] add and shift register, and an EXCLUSIVE-OR logic gate. Fig. 2 is a block diagram of the multiplier. To initiate the multiplication, the multiplicand is entered into the top n bits of the large shift register and the multiplier is stored in the n-bit storage register. During the first cycle, if the least significant bit of the multiplicand and the least significant multiplier bit are both logic ONES, one of the AND gates generates a ONE, which is counted by the counter, added to the contents of the add and shift register (hereafter called the carry register), and the least significant bit of the count is shifted out as the first (i.e., least significant) bit of the product. The multiplicand is shifted down by one bit causing the second column of the bit-product array to be generated by the AND gates. The number of ONES is determined by the counter, added to the carry register contents, and the least significant bit is shifted into the product register. When a total of $2n$ of these cycles have been performed the complete product is in the product register.

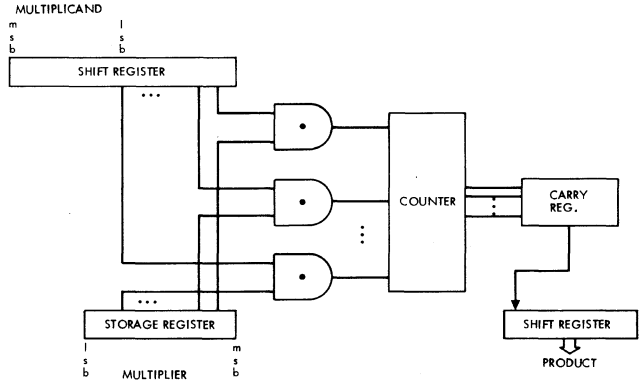


Fig. 2. Block diagram of the quasi-serial multiplier.

Multiplicand / Multiplier	Counter	Carry Register After Add	Carry Register After Shift	Product Bit
19 / 1 0 0 1 1 0 0 0 0 / 0 1 1 0 1 / 22	0	0	0	0
0 1 0 0 1 1 0 0 0 / 0 1 1 0 1	1	1	0	1
0 0 1 0 0 1 1 0 0 / 0 1 1 0 1	10	10	1	0
0 0 0 1 0 0 1 1 0 / 0 1 1 0 1	1	10	1	0
0 0 0 0 1 0 0 1 1 / 0 1 1 0 1	1	10	1	0
0 0 0 0 0 1 0 0 1 / 0 1 1 0 1	10	11	1	1
0 0 0 0 0 0 1 0 0 / 0 1 1 0 1	1	10	1	0
0 0 0 0 0 0 0 1 0 / 0 1 1 0 1	0	1	0	1
0 0 0 0 0 0 0 0 1 / 0 1 1 0 1	1	1	0	1
0 0 0 0 0 0 0 0 0 / 0 1 1 0 1	0	0	0	0

Product is: $0\,1\,1\,0\,1\,0\,0\,0\,1\,0_2$

$19 \times 22 = 418_{10}$

Fig. 3. Steps in the computation of 19 X 22 with the quasi-serial multiplier.

The execution of the jth cycle yields s_j [defined by (6)] as the output of the ones counter, and c_i [from (8)] as the contents of the carry register. An example that demonstrates this procedure is shown in Fig. 3. The problem 19 X 22 is solved with this quasi-serial algorithm.

Under certain conditions, the quasi-serial multiplier may be simplified somewhat. If the multiplier number will be available for the duration of the multiplication, its n-stage storage

[2] In this paper, a counter is a circuit that indicates how many of the inputs are in the logic ONE state.

[3] Entier (X) is the largest integer n such that $n \leq X$.

register may be deleted. Similarly, if the multiplicand is available sequentially at a compatible clock rate it may be loaded into an n-stage shift register instead of the $2n - 1$ stage register that is shown in Fig. 2. If the product is to be processed serially, the output shift register can be eliminated.

The delay for each cycle is the sum of the gate delay, the counter delay, the addition delay of the carry register, and the shift delay. If the carry register is implemented with a 1 + Entier $(\log_2(2n - 1))$ stage carry-lookahead adder, the delay is

$$T_{mult} = 2n(T_{gate} + T_{count} + T_{cla} + T_{shift}) \quad (10)$$

where T_{count} is the delay of an n-input counter and T_{shift} is the delay of a shift register. If a full-adder counter (see Foster and Stockton [8] for details of counter implementation with full adders) is used, its delay is approximately

$$T_{count} \approx T_{add}(\text{Entier } (\log_2(n)) + \text{Entier } (\log_3(n - 1))) \quad (11)$$

where T_{add} is the delay of a full adder.

Assuming that the delay of a carry-lookahead adder is nearly equal to the delay of an adder module (i.e., a full or half adder)

$$T_{mult} \approx 2n(T_{gate} + T_{shift} + T_{add}(1 + \text{Entier } (\log_2(n)) + \text{Entier } (\log_3(n - 1)))). \quad (12)$$

From (12) it is evident that the counter delay dominates the multiplier delay. A potentially much faster counter is described in the Appendix.

The delay of the multiplier with the quasi-digital (i.e., current summing) counter described in the Appendix is given by substituting T_{count} from (23) into (10)

$$T_{mult} = 2n(2T_{gate} + T_{settling} + T_{comp} + T_{cla} + T_{shift}). \quad (13)$$

Assuming

$$T_{cla} \approx T_{comp} \approx 3T_{gate}$$

and

$$T_{settling} \approx T_{shift} \approx T_{gate}$$

$$T_{mult} \approx 20nT_{gate}. \quad (14)$$

Comparison with Other Multipliers

Table I summarizes the complexity of the two forms of the quasi-serial multiplier, two forms of the basic add and shift multiplier, and the fully parallel multiplier. For comparison purposes, an analog comparator is assumed to be equivalent to a logic gate in complexity.

Evidently the add and shift multiplier with a ripple-carry adder is the least complex of the multipliers characterized in Table I. Since it requires an n-stage carry-lookahead adder, the other add and shift multiplier is somewhat more complex than the quasi-serial multipliers. The fully parallel multiplier is certainly the most complex of all.

The delays of the two quasi-serial schemes have been derived in the previous section, i.e., (12) and (14). The delays of the ripple-carry adder and the carry-lookahead adder forms of the add and shift multiplier are given by T_a and T_c, respectively (see e.g., Flores [9])

TABLE I
Multiplier Complexity

Method	Gates	Full Adders	Register Stages	Cla Adder Stages
Quasi-Serial with adder array counter	n	n	$5n + \text{Log}_2(2n)$	$\text{Log}_2(2n)$
Quasi-Serial with current summing counter	3n		$5n + \text{Log}_2(2n)$	$\text{Log}_2(2n)$
Add and Shift with ripple-carry adder	n	n	4n	
Add and shift with carry-lookahead adder	n		4n	n
Fully parallel	n^2	(n-1)(n-2)	4n	2n-1

TABLE II
Multiplier Delays

	Type of Multiplier				
	Quasi-Serial		Add and Shift		
Word Length	Adder Ctr.	Current Summing Ctr.	Ripple-Carry Accumulator	CLA Acc.	Parallel
5	140	100	80	20	10
10	400	200	310	40	16
15	600	300	690	60	19
20	920	400	1220	80	22
25	1150	500	1900	100	25
30	1560	600	2730	120	25
35	2030	700	3710	140	28
40	2320	800	4840	160	28
45	2610	900	6120	180	28
50	2900	1000	7550	200	31
55	3190	1100	9130	220	31
60	3480	1200	10860	240	31
65	4160	1300	12740	260	34
70	4480	1400	14770	280	34
75	4800	1500	16950	300	34
80	5120	1600	19280	320	34

$$T_a = n(T_{shift} + nT_{add}) \quad (15)$$

$$T_c = n(T_{shift} + T_{add}). \quad (16)$$

The delay of the parallel multipliers is given by the approximate relation [10]

$$T_{mult} \approx T_{gate} + T_{add}\,\text{Entier}\,(2\log_2(n) - 2) + T_{cla}. \quad (17)$$

The number of gate delays of the various multipliers are tabulated for word lengths between 5 and 80 in Table II. It is assumed that $T_{cla} \approx T_{add} \approx 3T_{gate}$ and that $T_{shift} \approx T_{gate}$ for the purpose of the table.

From Table II it appears that either of the quasi-serial multipliers is faster than the ripple-carry accumulator form of the add and shift multiplier for word lengths in excess of 15. Although neither of the quasi-serial multipliers is as fast as the add and shift multiplier implemented with a carry-lookahead accumulator, the complexity of carry-lookahead adders becomes excessive for large word lengths.

Conclusions

This paper describes the quasi-serial multiplication algorithm and compares it to both the well-known add and shift algorithm and to Dadda's fully parallel multiplication algorithm.

Two forms of the quasi-serial algorithm are described that may be considered as defining new points on the speed-simplicity spectrum of digital multipliers. It appears that the quasi-serial multipliers are roughly comparable to add and shift multipliers in both speed and simplicity.

Although pipelining has not been discussed it should be clear that any of the multipliers described herein can be operated in a pipelining mode to achieve greater throughput (at some sacrifice in initial delay).

The major component of the quasi-serial multiplier is a parallel ones counter that generates the count of how many of its inputs are in the logic ONE state. As counters are improved in speed, the speed of the quasi-serial multiplier will increase. Toward this goal a current summing counter is described in the Appendix that appears to have great potential for high-speed operation.

Appendix
Current Summing Counters

The counters described in this Appendix use analog current summing to generate a voltage that is proportional to the count. The voltage is digitized with an analog-to-digital converter to yield the "count" of how many inputs are active.[4]

The design of a current summing counter is illustrated by Fig. 4 which is the circuit of a seven-input counter. Each input that is either grounded (logic ZERO) or at the same voltage as the logic supply (logic ONE) passes into a network of N resistors. The voltage at the common node of the resistor network, V_n is approximated by

$$V_n \approx V_{\text{ref}} \cdot k/N \tag{18}$$

where V_{ref} is the voltage of the logic supply, k is the number of inputs in the logic ONE state, and N is the total number of inputs to the network. This relation is only approximate since the resistors and input voltage levels may deviate slightly from their nominal values. This voltage is applied to the inverting inputs of N analog comparators. The voltage applied to the positive input of the ith (from the bottom of Fig. 4) comparator is

$$V_{pi} = V_{\text{ref}}(2i - 1)/2N. \tag{19}$$

The outputs of the ith comparator are two lines c_i and $\bar{c}_i$ that are a function of the input voltages. If

$$\begin{aligned} V_{pi} > V_n \quad & c_i = 1 \text{ and } \bar{c}_i = 0 \\ V_{pi} = V_n \quad & \text{undefined} \\ V_{pi} < V_n \quad & c_i = 0 \text{ and } \bar{c}_i = 1. \end{aligned} \tag{20}$$

From (18) and (19) it is apparent that

$$V_{pi} < V_n, \quad \text{for } i \leqslant k \tag{21a}$$

and

$$V_{pi} > V_n, \quad \text{for } i > k. \tag{21b}$$

[4]Current summing counters are similar in function to quantizers described in [11].

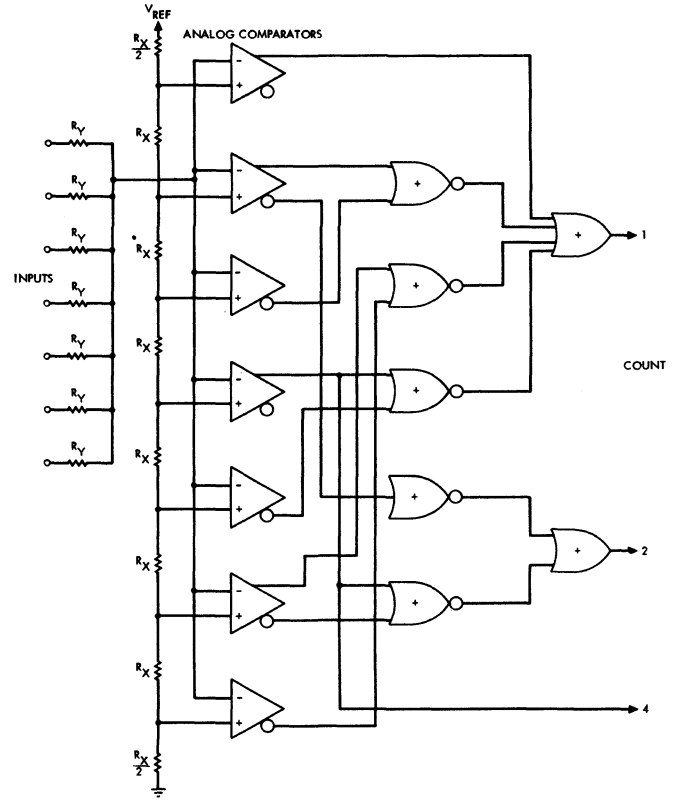


Fig. 4. Seven-input current summing counter.

Thus

$$c_i = 0 \text{ and } \bar{c}_i = 1, \quad \text{for } 0 < i \leqslant k \tag{22a}$$

and

$$c_i = 1 \text{ and } \bar{c}_i = 0, \quad \text{for } k < i \leqslant n. \tag{22b}$$

The n pairs of comparator outputs are decoded with a simple logic network that generates the binary count of the number of active inputs. Since it is attractive to use emitter coupled logic (ECL) to realize the comparator [12], the encoding network is implemented with logical NOR gates and the implied OR logic function. The use of ECL is especially attractive from speed considerations. It is possible to effect the comparison and encoding in less than 5 ns with Motorola MECL III circuit elements.

Denoting the delay of the quasi-digital counter by T_{count}

$$T_{\text{count}} = T_{\text{settling}} + T_{\text{comp}} + T_{\text{gate}} \tag{23}$$

where T_{settling} is the delay of the resistor network (and the delay of logic buffers that may be required to achieve accurate logic levels) and T_{comp} is the delay of the comparator. Since T_{comp} and T_{gate} are invariant with the size of the counter, the only variation in counter delay as the size is increased is due to variations in the settling time of the input network. The settling time is due to stray capacitance that causes the voltage that is applied to the comparators to settle exponentially to the correct value. The effective time constant of the resistor network is τ

$$\tau = R_y C_{\text{stray}}/N \tag{24}$$

where R_y is the value of each of the N-input resistors and C_{stray} is the stray capacitance at the resistor node. The maximum error in the voltage V_n as a function of time is

$$\epsilon_n = V_{ref} \exp(-t/\tau). \quad (25)$$

There is yet another error in the voltage V_n, which is due to errors in the resistor network. If all resistors are within a certain tolerance say ϵ_y the total error in the node voltage does not exceed ϵ_y/R_y. The voltages at each of the nodes of the reference-voltage divider are subject to a similar error of ϵ_x/R_x.

The comparators will all have correct input voltage levels when

$$V_{ref}/2N > \epsilon_n + |V_{offset}| + V_{ref}(|\epsilon_x/R_x| + |\epsilon_y/R_y|). \quad (26)$$

Assuming that all resistors have tolerance ΔR such that

$$\Delta R = |\epsilon_x/R_x| = |\epsilon_y/R_y|. \quad (27)$$

Solving (26) for ϵ_n and using (24) and (25)

$$T_{settling} = (-R_y C_{stray} \log(1/2N - 2\Delta R - |V_{offset}|/V_{ref}))/N. \quad (28)$$

Obviously the counter will operate if and only if

$$1/2N > 2\Delta R + |V_{offset}|/V_{ref}.$$

As N is increased, the accuracy of the resistors must be increased until $V_{ref} \approx 2N|V_{offset}|$ at which point, the circuit becomes inoperative. For $N < 100$ the offset voltage may be considered negligible and

$$\Delta R < 1/4N. \quad (29)$$

As the resistor accuracy is improved, the settling delay improves in accord with (28).

The implementation of an N-input current summing counter requires: $2N$ resistors that satisfy (29); N comparators; N two-input logic gates; and N-input buffers.[5]

Current summing counters do not appear to be feasible for values of N that are much in excess of 100 since such counters require high-accuracy resistors (and buffers), and may involve considerable settling time. It is certainly possible to construct larger counters by summing the outputs of smaller counters with full adders. The basic idea is similar to constructing digital counters from full-adder modules (i.e., three-input counters) as described by Foster and Stockton [8].

Acknowledgment

The author wishes to thank C. H. Downey of Ball Brothers Research Corporation for furnishing the problem that led to the development of the quasi-serial multiplier. Discussions with Prof. I. S. Reed of the University of Southern California and comments from the reviewers greatly improved this paper.

[5] Buffers are only required if N is large and the input voltages are not stable and uniform.

References

[1] A. W. Burks, H. H. Goldstine, and J. von Neumann, *Preliminary Discussion of the Logical Design of an Electronic Computing Machine*, part 1, vol. 1. Princeton, N.J.: Institute for Advanced Study, 1946. (Reprinted in J. von Neumann, *Collected Works*, vol. 5. New York: Macmillan, 1963, pp. 34–79.)

[2] A. D. Booth, "A signed binary multiplication technique," *Quart. J. Mech. Appl. Math.*, vol. 4, pt. 2, pp. 236–240, 1951.

[3] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pp. 67–91, Jan. 1961.

[4] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, Feb. 1964.

[5] L. Dadda, "Some schemes for parallel multipliers," *Alta Freq.*, vol. 34, pp. 346–356, May 1965.

[6] D. Ferrari and R. Stefanelli, "Some new schemes for parallel multipliers," *Alta Freq.*, vol. 38, pp. 843–852, Nov. 1969.

[7] A. Habibi and P. A. Wintz, "Fast multipliers," *IEEE Trans. Comput.* (Short Notes), vol. C-19, pp. 153–157, Feb. 1970.

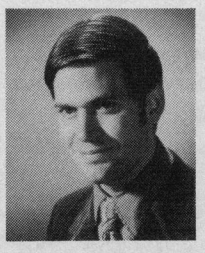
[8] C. C. Foster and F. D. Stockton, "Counting responders in an associative memory," *IEEE Trans. Comput.* (Short Notes), vol. C-20, pp. 1580–1583, Dec. 1971.

[9] I. Flores, *The Logic of Computer Arithmetic.* Englewood Cliffs, N.J.: Prentice-Hall, 1963, pp. 73–99.

[10] E. E. Swartzlander, Jr., "The inner product computer," Ph.D. dissertation, Dep. Elec. Eng., Univ. Southern California, Los Angeles, pp. 48–49, 1972.

[11] R. H. S. Riordan and R. R. A. Morton, "The use of analog techniques in binary arithmetic units," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 29–35, Feb. 1965.

[12] L. S. Garrett, "Integrated-circuit digital logic families, III," *IEEE Spectrum*, vol. 7, pp. 30–42, Dec. 1970.



Earl E. Swartzlander, Jr. (S'64–M'72) was born in San Antonio, Tex., on February 1, 1945. He received the B.S.E.E. degree from Purdue University, West Lafayette, Ind., in 1967, the M.S.E.E. degree from the University of Colorado, Boulder, in 1969, and the Ph.D. degree in electrical engineering (computer option) from the University of Southern California, Los Angeles, Calif., in 1972.

From 1967 to mid 1969 he was a Development Engineer in the Space Scientific Experiment Department of Ball Brothers Research Corporation, Boulder, Colo. He was responsible for the electronic systems design of the Harvard College Observatory experiment that will be flown as part of Skylab. In the spring of 1969 he was a Teaching Assistant at the University of Colorado where he taught classes in digital computer programming. From 1969 to 1972 he was a Howard Hughes Doctoral Fellow at the University of Southern California. From 1969 to the present he has also been employed by the Hughes Aircraft Company, Culver City, Calif. His current interests include computer arithmetic, the development of special-purpose computers, pattern recognition, and digital image processing.

Dr. Swartzlander is a member of Eta Kappa Nu, Sigma Tau, Omicron Delta Kappa, the Association for Computing Machinery, and the Optical Society of America. He is also the President of the Temperature Measurements Society.