An Augmented Iterative Array for High-Speed Binary Division

MAURUS CAPPA AND V. CARL HAMACHER

Abstract—**An augmented iterative array for binary division (IAD), is described. It uses carry-save reduction and carry-look-ahead principles to achieve high speed. Logic cost and speed comparisons with two other design techniques are presented. An 8-bit prototype model that operates in under 500 ns has been built from commercially available high-speed MSI TTL integrated circuits to verify the feasibility of the IAD scheme.**

Index Terms—**Division network, iterative array, nonrestoring division.**

I. Introduction

LARGE general-purpose digital computers require high-speed arithmetic units. There are numerous special-purpose digital processors, such as fast Fourier transform machines, that also have this requirement. High-speed addition [1] and multiplication [2]–[4] have been reasonably thoroughly studied. The iterative logic cell has been stressed as an important design factor from a fabrication and diagnostic standpoint.

The purpose of this paper is to present the design of a new high-speed iteratively structured binary division array named IAD (iterative array divider). The logic cost and speed of this design is compared to a basic division array similar to those proposed in [5], [6], and to another recent high-speed scheme [7]. The construction of an 8-bit prototype divider using standard high-speed MSI TTL integrated circuits is briefly described in a final section.

II. Basic Division Array

An iterative array which implements the conventional nonrestoring division algorithm is presented here. It is essentially the same as arrays proposed by others [5], [6] and is included here as the starting point from which two design changes lead to the much faster IAD presented in Section III.

In binary division, successively right-shifted versions of the divisor are subtracted from or added to the dividend and resulting partial remainders. The sign of the partial remainder determines the quotient bit and further, in nonrestoring division, determines whether to add or subtract the shifted divisor in the next cycle. The basic step of the nonrestoring algorithm can be concisely stated with the help of the following notation:

$$\begin{aligned} \text{dividend:} \quad & A = A_0 . A_1 A_1 \ldots A_N \\ \text{divisor:} \quad & D = D_0 . D_1 D_2 \ldots D_N \\ \text{(partial) remainder:} \quad & R = R_0 . R_1 R_2 \ldots R_N \\ \text{quotient:} \quad & Q = Q_0 . Q_1 Q_2 \ldots Q_N. \end{aligned}$$

The binary point is assumed to lie between the 0 and 1 subscripted components of each vector. The operands are assumed to be positive, normalized fractions, so that $A_0 = D_0 = 0$ and $A_1 = D_1 = 1$. Since $\frac{1}{2} \leq A, D < 1$, the quotient is positive and lies in the range $\frac{1}{2} < Q < 2$. The (partial) remainder R is a signed fraction, and R_0 is the sign bit with R being represented in 2's complement form.

An $N+1$-bit quotient can then be generated from an N-bit divisor and dividend by the following (nonrestoring) binary division algorithm.

DO FOR $i = 0$ to N.

Step 1:[1] Generate new partial remainder: if $Q_{i-1} = 1$, $R \leftarrow R - D$; if $Q_{i-1} = 0$, $R \leftarrow R + D$.

Step 2: Quotient bit: set $Q_i = \overline{R_0}$.

Step 3:[2] Shift partial remainder: $R \leftarrow 2(R)$.

A two-dimensional array of iteratively structured logic can implement the nonrestoring algorithm directly. Basically, the array consists of rows of carry-propagate adders with one controlled input per bit. See Fig. 1(a) and (b) for an example with a 4-bit divisor and an 8-bit dividend. Each logic cell consists of a full adder and an EXCLUSIVE-OR gate. The EXCLUSIVE-OR gate controls the divisor input to the full adder. The control signal S determines whether an add or subtract is to be performed. Subtraction is performed in 2's complement form by forming the 1's complement of the divisor and forcing a carry into the low-order bit position. The operation of the array is mostly self-explanatory. The 4-bit divisor and double-length 8-bit dividend are introduced at the top and right edges of the array. Since $A_0 = D_0 = 0$ and $A_1 = D_1 = 1$, the upper left three cells could have been eliminated provided that all input–output connections are sent directly through the original locations of the cells. A 5-bit quotient is developed at the left of the array. Note that each quotient bit is passed on to the next row as the control signal S and that this

Manuscript received May 8, 1972; revised August 28, 1972. This work was supported in part by the scholarship program and Grant A-5192 of the National Research Council of Canada.

M. Cappa is with Collins Radio Company, Toronto, Ont., Canada.

V. C. Hamacher is with the Departments of Electrical Engineering and Computer Science, University of Toronto, Toronto, Ont., Canada.

[1] For $i = 0$, $R = A$ = dividend and formally $Q_{-1} = 1$.

[2] If a double-length dividend is provided, its right half is brought into R_N a bit at a time by the application of Step 3.

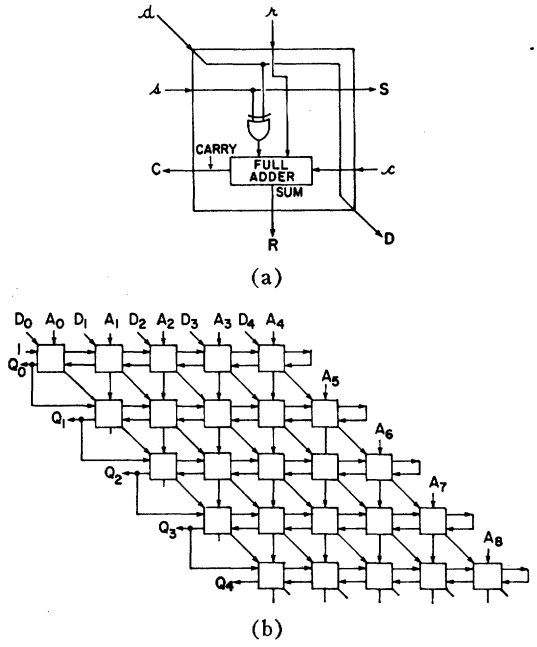

Fig. 1. Basic division array. (a) Typical cell. (b) Example: 8-bit dividend, 4-bit divisor.

signal is the low-order carry into the rightmost cell. The reason that the carry out signal from the left cell of each row is equal to the quotient bit for that position is a consequence of the use of 2's complement number representation and the nonrestoring algorithm.

This is a reasonable operational interpretation of the use of the array in a floating-point arithmetic unit where the input operands are normalized fractions. However, the array can also be interpreted as performing integer division in which a double-length dividend (in this case 8 bits) is introduced at the top and right diagonal of the array. Bit positions A_8, D_4, and Q_4 in Fig. 1(b) then have the binary weighting of unity, and the divisor should be larger than the left half of the dividend.

III. High-Speed Division Array (IAD)

The technique used in the IAD [8] is best understood as two rather major modifications of the implementation of the nonrestoring division algorithm described in Section II. The basic idea of the design is to eliminate the carry ripple time, which is proportional to N, along each row of the array. First, the partial remainder R is not actually developed in each row of the array, but is represented by two binary vectors S and C which, if added, would produce the correct partial remainder at that row level. Second, a single carry-lookahead subnetwork is used to determine from the S and C vectors what the carry into the sign bit would be, facilitating the determination of the sign of the resulting partial remainder, the quotient bit for that row level, and the control (add or subtract divisor) to the next row. The two vectors S and C are the result of a 3-to-2 carry-save reduction on the previous row's S and C vectors and the proper version of the shifted divisor (add or subtract). Subtraction of the divisor is again implemented using 2's complement addition as in the basic array. The carry-save reduction operation is achieved in only one cell delay, independent of the length of the operands. The speed of the lookahead network, divisor true or complement selection, and the single-cell delay for carry-save reduction are the basic factors determining the delay per bit of the quotient Q being developed.

The notation introduced in Section II can be used to describe the IAD technique. Vectors S and C are the sum and carry vectors, respectively, resulting from a 3-to-2 carry-save addition reduction. The IAD algorithm is then as follows.

DO FOR $i=0$ to N.

Step 1:[3] Generate the two-vector partial remainder: if $Q_{i-1}=1$, S', $C' \leftarrow S+C-D$; if $Q_{i-1}=0$, S', $C' \leftarrow S+C+D$.

Step 2: Lookahead: $CL=G_1+P_1G_2+P_1P_2G_3+\cdots+P_1\cdots P_{N-2}G_{N-1}$ where $G_j=S'_jC'_j$ (generate function) and $P_j=S_j'+C'_j$ (propagate function).

Step 3: Partial remainder sign: $R_0=S'_0\oplus C'_0\oplus CL$ and quotient bit: $Q_i=\overline{R_0}$.

Step 4:[4] Shift partial remainder: S, $C \leftarrow 2(S', C')$.

The general Step 1, formulated above, can be illustrated by the following operation:

$$\begin{array}{l} S_0.S_1S_2\ldots S_{N-1}A_{N+i} \\ C_0.C_1C_2\ldots 0 \quad Q_{i-1} \\ \pm D_0.D_1D_2\ldots D_{N-1}D_N \\ \hline S_0'.S_1'S_2'\ldots S_{N-1}'S_N' \\ \quad / \quad / \quad / \quad \ldots \quad / \quad / \\ C_0' \quad \underbrace{C_1'C_2'\ldots C_{N-1}'} \quad 0 \\ \qquad CL \text{ determines lookahead bit} \end{array}$$

$$\overline{Q_i} = \overline{\overline{R_0}}. \tag{1}$$

This nonrestoring algorithm which uses lookahead and carry-save addition reduction to speed up the division process can be implemented best in an augmented array using three types of logic cells and a minor amount of additional logic. The logic cells are illustrated in Fig. 2(a) and the array schematic for the same size operands as in Fig. 1 is shown in Fig. 2(b). The *CLA* cell incorporates logic to determine the carry into the sign position of the partial remainder. This logic is dependent on the word length N, and for long word lengths ($N \geq 10$), two levels of lookahead must be used in order to minimize fan-in. A maximum fan-in of 8 has been assumed for the analysis of performance in Section IV.

Since the form of the lookahead network *CLA* varies as a function of wordlength N, it is not strictly technically correct to call IAD an iterative array in the ac-

[3] For $i=0$: $S=A=$dividend, $Q_{-1}=1=C_N$, and $C=0\cdots 0\ 1$.

[4] After the shift, the low-order bits of a double-length dividend can be introduced into the vacated S_N position, one bit at a time, and C_N can be set to Q_i to accomplish the 2's complementing function on D.

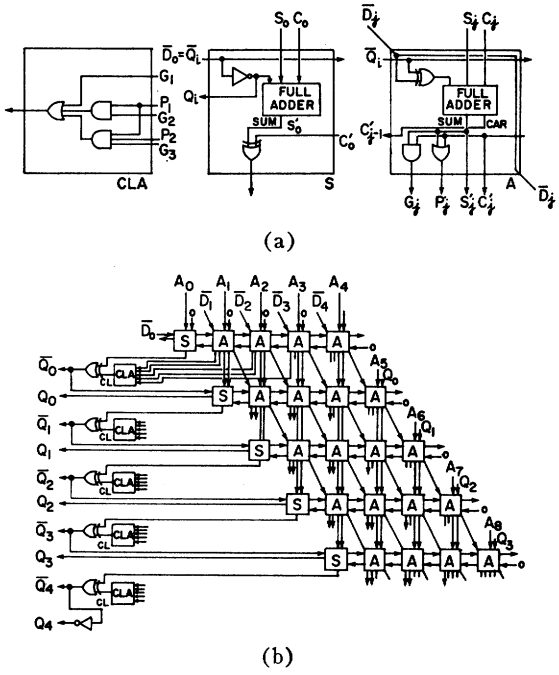

Fig. 2. IAD system. (a) The typical cells. (b) Example: 8-bit dividend, 4-bit divisor.

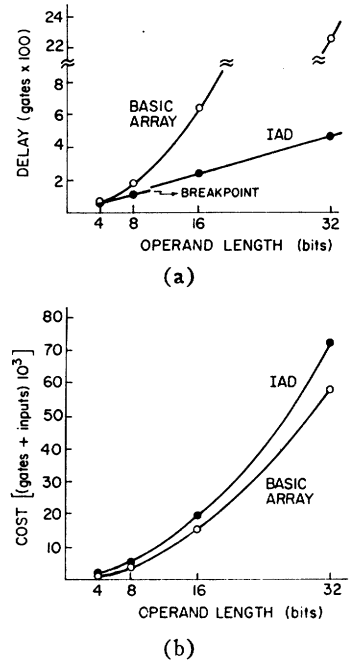

Fig. 3. Comparison of IAD and basic array. (a) Speed of IAD versus basic array. (b) Cost of IAD versus basic array.

cepted sense of that term which usually means a single fixed-cell design repeated in a regular pattern to accommodate any operand length. That is why we have added the modifier "augmented" at the beginning. The left-edge sign cells S and the EXCLUSIVE-OR gates are another departure from strict uniformity.

It should be noted at this point that simply adding lookahead alone to the basic array of Section II does not achieve the same results as the IAD. Much more lookahead circuitry would be needed to generate all partial remainder bits quickly in each row to achieve the same speed as in the IAD. It is the combination of carry-save reduction and sign-bit lookahead that achieves the high speed at reasonable cost.

The S cells are in the sign position of the partial remainder and compute the binary sum of D_0, S_0, C_0 from operation (1), in the FULL ADDER. This sum is then added (in the EXCLUSIVE-OR gate) to the carry output C_0' from the 3-to-2 reduction in the adjacent A cell. Finally, the result is added to the carry-lookahead bit CL in the EXCLUSIVE-OR gate at the output of the CLA cell, the answer being R_0, which is $\overline{Q_i}$ for the ith row. The quotient bit Q_i is actually derived from the inverter output in the S cell of the next row.

The A cells which constitute the main body of the array perform the 3-to-2 carry-save reduction called for in Step 1 of the IAD algorithm and illustrated in (1). The functions P (propagate) and G (generate) are also computed in the A cell.

IV. IAD Performance

It is important to note that the IAD presented in Section III results in division times that essentially increase only linearly with word length. The trend is actually proportional to $N \log N$ where the log N term results from the lookahead function, with the maximum gate fan-in determining the base of the logarithm. The division time in the basic design of Section II increases proportional to N^2 due to the carry propagation along each row.

The two designs will be compared with respect to speed and cost for word lengths from 4 bits to 32 bits. From 10 bits up, two levels of lookahead are needed in the CLA cells of the IAD to maintain the worst case fan-in at 8 inputs. Speed is evaluated in terms of the number of gate delays (each gate having an assigned delay of one unit) and cost in terms of gates plus inputs. All synthesis is evaluated in terms of AND, OR, and NOT gate implementations of the various cell functions. Even though this may be only a rough gauge of actual delay when integrated circuits are used, it is valid as a comparative tool when considering alternate designs. Worst case delays for a full $N+1$-bit quotient are as follows. Basic array: $T=2N^2+7N+5$, where the delay per bit is 3 for the EXCLUSIVE-OR gate plus $2(N+1)$ for carry ripple; and IAD: $T=12(N+1)$ for $N\leq 9$, or $14(N+1)$ for $10\leq N\leq 32$ where the delay per bit is 7 for the A cell plus 2 (or 4) for the CLA cell plus 3 for the EXCLUSIVE-OR gate. It should be noted that a functionally equivalent A cell having lower delay but more gates could be designed. Similarly, the CLA cell and the EXCLUSIVE-OR gate could be combined to reduce overall delay, again at increased cost.

Approximate speed and cost curves are plotted in Fig. 3(a) and (b). Note that at $N=16$ bits, the IAD is about 2.6 times faster than the basic array and about 22 percent more expensive; while at $N=32$ bits, the figures are five times faster and 25 percent more expen-

sive. The number of gates and inputs involved for either array at the larger values of N is clearly only feasible if the arrays are implemented using reasonably large-scale integration of sublogic blocks.

It is also instructive to compare the IAD design procedure with another recent high-speed divider design technique reported by Stefanelli [7]. Working with his best design, procedure four, it is easy to see that N quotient bits are computed in a delay of approximately

$$\underbrace{\lceil N(\tau_m + 2\tau_{pc} + \tau_s) \rceil}_{\text{divider}} + \underbrace{\lceil N\tau_b \rceil}_{\text{converter}}$$

where τ_m, τ_{pc}, and τ_s are the delays through the multiplier gate, parallel counter, and subtractor of [7, fig. 10] and τ_b is the delay through each box of the converter of [7, fig. 11]. These circuits are actually ternary, although the external inputs and outputs (divisor, dividend, and quotient) are binary. For comparison to the IAD, it is reasonable to assume at least three units of delay for each of τ_{pc}, τ_s, and τ_b; this being the delay through a standard two-level binary synthesis of each function assuming binary-coded ternary representation, and assuming input complements are not available. One unit of delay is assigned to τ_m since it corresponds to an AND-gate function. Therefore, total delay through the Stefanelli divider (procedure four) is approximately $13N$ compared to $14N$ for IAD. A cost comparison between these two dividers is somewhat more difficult because no detailed synthesis has been specified in the Stefanelli paper. In any event, a reasonably straightforward line of reasoning leads to the conclusion that a comparison of the array costs can be approximated by a comparison of the cost of the A cell used in the IAD with the cost of a four-input two-output ternary parallel counter (Stefanelli) since these two modules are the basic components in the main body of the respective arrays. The A cell of the IAD should be substantially cheaper than any implementation of the Stefanelli counter.

It should also be noted that the size of the Stefanelli counter is actually a slowly varying function of N, which is difficult to determine, and the above size (four inputs, two outputs) is for the case of $N=12$. For larger N, a slightly larger counter would be required.

A comment should be made about fan-out problems. They certainly exist in all three designs discussed in the preceding, for the larger values of N, but their elimination would affect the cost and speed of each design to about the same degree, so that the parameters computed above remain a valid indication of the relative merits of the various design procedures.

V. Construction of an 8-Bit Divider

In order to verify the practicality of the IAD design procedure, a prototype divider was built from commercially available integrated circuits. In the prototype, a 16-bit dividend is divided by an 8-bit divisor yielding an 8-bit quotient. The total network is mounted on eight standard cards, holding a total of 136 IC's, each card generating one quotient bit. The IC technology used was SSI and MSI high-speed TTL with a basic gate delay of about 6 ns. The worst case delay per quotient bit was computed to be 65 ns implying that all eight quotient bits could be generated in about 512 ns. Experimentally, worst case delays of under 500 ns were measured. Since we have demonstrated that speed is almost linear with operand length, it is feasible to construct combinational division circuits from currently available high-speed TTL IC's that operate within about a memory cycle (1 μs) in a 12- or 16-bit computer, assuming that a word-length accuracy quotient is generated. Whether or not the cost can be justified clearly depends on the application.

References

[1] I. Flores, *The Logic of Computer Arithmetic*. Englewood Cliffs, N. J.: Prentice-Hall, 1963.

[2] F. C. Hennie, *Finite-State Models for Logical Machines*. New York: Wiley, 1968, ch. 10.

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

[4] S. D. Pezaris, "A 40-ns 17-bit by 17-bit array multiplier," *IEEE Trans. Comput.* (Short Notes), vol. C-20, pp. 442–447, Apr. 1971.

[5] J. C. Majithia, "Nonrestoring binary division using a cellular array," *Electron. Lett.*, vol. 6, pp. 303–304, 1970.

[6] H. H. Guild, "Some cellular logic arrays for non-restoring binary division," *Radio Electron. Eng.*, vol. 39, pp. 345–348, 1970.

[7] R. Stefanelli, "A suggestion for a high-speed parallel binary divider," *IEEE Trans. Comput.*, vol. C-21, pp. 42–55, Jan. 1972.

[8] M. Cappa, "Cellular iterative arrays for multiplication and division," M.S. thesis, Dep. Elec. Eng., Univ. Toronto, Ont., Canada, Oct. 1971.

❖

Maurus Cappa was born in Barile, Italy, on December 22, 1947. He received the B.A.Sc. and the M.A.Sc. degrees in electrical engineering from the University of Toronto, Toronto, Ont., Canada, in 1970 and 1971, respectively. His graduate work in the Computer Group of Electrical Engineering at the University of Toronto was sponsored by a National Research Council Scholarship.

He is currently with Collins Radio Company, Toronto, Ont., Canada, where he is involved in the design of frequency synthesizers for use in military transceivers.

❖

V. Carl Hamacher (S'66–M'68) was born in London, Ont., Canada, on September 28, 1939. He received the B.A.Sc. degree in engineering physics from the University of Waterloo, Waterloo, Ont., Canada, in 1963, the M.Sc. degree in electrical engineering from Queen's University, Kingston, Ont., Canada, in 1965, and the Ph.D. degree in electrical engineering from Syracuse University, Syracuse, N. Y., in 1968.

He was appointed Assistant Professor in both Electrical Engineering and Computer Science at the University of Toronto, Toronto, Ont., Canada, in 1968, and is currently an Associate Professor in those departments. His research interests include automata studies, iterative array structures, and computer organization.

Dr. Hamacher is a member of the Association for Computing Machinery and Sigma Xi.