Variations on Multi-Operand Addition for Faster Logarithmic-Time Tree Multipliers

Behrooz Parhami

Dept. of Electrical & Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA
E-mail: parhami@ece.ucsb.edu

ABSTRACT

A carry-free addition process for radix-2 numbers with the digit set [0, 3] is introduced and applied to the synthesis of tree multipliers using multi-operand adders built as binary trees. Such binary trees are more regular than standard carry-save adder (Wallace or Dadda) trees and thus offer advantages in terms of VLSI realization. We show that certain designs derived from binary stored-double-carry numbers with digit set [0, 3] compare favorably with previously proposed multipliers using binary-tree multi-operand addition schemes based on reducing pairs of carry-save numbers with (4, 2) counters or combining pairs of borrow-save numbers using binary signed-digit adders. The advantages are particularly pronounced for word lengths that are at or close to halfway between consecutive powers of 2 (e.g., around 12, 24, or 48 bits).

I. Introduction

Over the years, many parallel multiplication schemes have been proposed based directly or indirectly on the concept of "stored" or "saved" carries/borrows. With the standard carry-save or Wallace-tree approach, each carry-save adder in the first level converts three conventional binary numbers into a BSC (binary stored-carry) number which can be viewed as using the digit set [0, 2] in radix 2. In subsequent stages, each carry-save adder combines 1.5 BSC (or equivalently 1 BSC and 1 conventional binary) numbers into a new BSC result, with the final BSC output converted to binary by a fast carry-propagate adder.

It is possible to add two (rather than 1.5) BSC numbers to obtain a BSC sum by using BSC adders. This forms the basis for Vuillemin's proposal to convert the irregular structure of 3-to-2 reduction trees to more regular binary trees [Vuil83]. For VLSI realization, the more regular layout of a binary tree compared to irregular carry-save adder trees is quite important because the area saved translates into lower cost and higher speed. However, if such BSC adders are implemented by using 2 carry-save adder levels, the advantage gained from the more regular layout may be partially, or even totally, lost to the increase in the number of logic levels on the critical path. These observations also apply to the use of BSD (binary signed-digit or stored-borrow) representation [Taka85], [Taka87], [Hara87].

We present a unified view of the parallel multiplier designs of Vuillemin and Takagi, Harata, et al cited above and show that their binary-tree reduction feature is provided by other redundant representations as well [Parh89]. Additionally, we demonstrate that one such alternate representation, that uses the digit set [0, 3] in radix 2, results in faster parallel multipliers for certain word lengths and that others provide potential advantages with certain design parameters and/or technological constraints.

II. GSD Arithmetic

Generalized signed-digit (GSD) number systems [Parh90] use the digit set $[-\alpha, \beta]$ in radix r with $\alpha \geq 0$, $\beta \geq 0$, and $\rho = \alpha+\beta+1-r$, where $\rho \geq 1$ is the *redundancy index* of the number system. Any GSD number system with $r > 2$ and $\rho \geq 3$ (or with $\rho = 2$, provided that $\alpha\neq 1$ and $\beta\neq 1$) supports carry-free addition. In all other cases (i.e., for $r=2$, $\rho=1$, or $\rho=2$ with $\alpha=1$ or $\beta=1$) a limited-carry addition algorithm is applicable which yields the ith sum digit s_i as a function of the six digits $a_i, b_i, a_{i-1}, b_{i-1}, a_{i-2}, b_{i-2}$ of the operands a and b [Parh90] Examples of GSD systems in limited-carry category are the stored-carry (SC), stored-borrow (SB), and stored-carry-or-borrow (SCB) representations corresponding to redundant radix-r digit sets $[0, r]$, $[-1, r-1]$, and $[-1, r]$. Extending these results to borrow-free and limited-borrow subtraction [Parh93], yields Algorithms 1 and 2, both of which begin with computing a *position result* $p_i = a_i \pm b_i$ and then proceeding as follows:

Algorithm 1 *(propagation-free add/subtract):* Break p_i into *transfer digit* t_{i+1} and *interim result* $w_i = p_i - (\pm r t_{i+1})$. The *final result* digit is $z_i = w_i \pm t_i$ (with no new transfer).

Algorithm 2 *(limited-propagation add/subtract):* Use p_i to generate a *range estimate* e_{i+1} for the transfer digit t_{i+1}. Then, based on both p_i and e_i, compute a *transfer digit* t_{i+1} and an *interim result* $w_i = p_i - (\pm r t_{i+1})$. The *final result* digit is $z_i = w_i \pm t_i$ (again with no new transfer).

In both cases, t_{i+1} can be restricted to the minimal range

$$-\lceil \alpha/(r-1)\rceil = -\lambda \leq t_{i+1} \leq \mu = \lceil \beta/(r-1)\rceil \qquad (1)$$

and its selection based on comparing p_i to some known comparison constants. We need $\lambda+\mu$ comparison constants to divide the range of p_i into $\lambda+\mu+1$ intervals. The range estimate $e_{i+1} \in \{low, high\}$ is a binary indicator (selected by comparing p_i to a constant) which restricts the transfer digit t_{i+1} into one of two subintervals; the *low* subinterval $[-\lambda, \mu']$ or the *high* subinterval $[-\lambda', \mu]$, where:

$$-\lambda < -\lambda' \leq \mu' < \mu \qquad (2)$$

We now briefly review the implications of these results for practical hardware realizations [Parh94]. A propagation-free adder/subtractor based on Algorithm 1 can be implemented in 4 logic levels: 2 levels for generating the transfer digits t_{i+1} and interim results w_i and 2 more levels for computing the final result digits z_i. Algorithm 2 may require two additional logic levels on top of the above four levels to establish the initial range estimates e_{i+1}. In either case, the number of levels may be reducible through optimization and use of clever coding techniques. For example, the work of Takagi et al [Taka85] contains a limited-carry binary signed-digit adder in which the generation of e_i and z_i each requires a single level of OR/NOR gates, thus leading to only 4 logic levels for the entire adder.

1058-6393/97 $10.00 © 1997 IEEE

III. CSA vs Binary Trees

Carry-save adder trees used in some parallel multipliers tend to be quite irregular and thus difficult to lay out in VLSI. This led Vuillemin [Vuil83] to the proposal for pairwise combination of BSC numbers to gain structural regularity. Vuillemin essentially suggests a 2-level carry-save adder to reduce two BSC numbers into one. Thus, for his scheme, each BSC level has twice as many gate levels as a simple carry-save adder. The main advantage gained is that it is structured as a binary tree which can be designed and laid out recursively. The greatest advantage is gained when the number of operands n is a power of two.

Table I contrasts the depth of carry-save reduction trees with that of binary-tree reduction circuits for different number n of operands. If the multiple-operand reduction is used for multiplication, the parameter n represents the length of multiplier (or half of multiplier length if Booth's encoding is used). Table I suggests that Vuillemin's scheme never reduces the number of gate levels compared to a Wallace-tree carry-save circuit. At best, when n is a small power of two, the number of gate levels remains the same. For some other values of n that are of practical significance in floating-point or sign-magnitude integer formats (e.g., 24, 48, 53, 56, 63) an increase of up to 14% in the number of gate levels is observed. The advantage of his scheme, however, result from the regularity of interconnections which affects both the cost (area) and signal delays in VLSI implementation. Takagi et al [Taka85], [Taka87], [Hara87] also use the same principle, but replace the stored-carry intermediate values of Vuillemin with stored-borrow (aka binary signed-digit or BSD) numbers.

Table I
Number of Adder Levels in Wallace Trees vs Binary Trees for Reducing n Operands to 2 Binary or 1 BSC Number

Number n of Inputs	Wallace-Tree Full Adders	Binary-Tree BSC Adders	Reduction Factor
3	1	1	1.00
4	2	1	2.00
5 - 6	3	2	1.50
7 - 8	4	2	2.00
9	4	3	1.33
10 - 13	5	3	1.67
14 - 16	6	3	2.00
17 - 19	6	4	1.50
20 - 28	7	4	1.75
29 - 32	8	4	2.00
33 - 42	8	5	1.60
43 - 63	9	5	1.80
64	10	5	2.00
65 - 94	10	6	1.67
95 -128	11	6	1.83

We will see, in subsequent sections of the paper, that there are other ways of affecting the reduction of n input operands using a binary tree structure. Any such binary-tree reduction scheme in which each adder requires 4 logic levels is potentially competitive with the above-mentioned designs. If the number of levels per adder can be reduced to 3 or 2 without complicating the interconnections, then a significant gain may result.

We next present a new multi-operand addition scheme in Section IV and then discuss its application in designing high-speed tree multipliers in Section V.

IV. BSDC Numbers

A limited-propagation adder/subtractor requiring six logic levels is clearly not competitive with the 4-level realization resulting from two cascaded carry-save adders. If we can somehow break the sequential relationship between computing the range estimates and the actual transfer digits, then other 4-level realizations may become possible. To do this, we note that the range estimate e_{i+1} is a mechanism by which the operand digits in position i affect the sum digit in position $i + 2$ (through modifying the transfer digit t_{i+2} generated by Stage $i + 1$). Thus if some information is sent directly from position i to position $i + 2$, the need for propagation through position $i + 1$ may be eliminated.

For binary stored-double-carry (BSDC) numbers, with the digit set [0, 3] in radix 2, this can be easily accomplished. Each position sum p_i is in the range 0 to 6 and can be broken into an interim sum w_i, a carry c_{i+1}, and a double-carry d_{i+2} which will go directly to position $i + 2$ (Fig. 1). Thus, the addition process involves two steps:

$$a_i + b_i = p_i = 4d_{i+2} + 2c_{i+1} + w_i \tag{3a}$$
$$s_i = c_i + d_i + w_i \tag{3b}$$

In these arithmetic equations, w_i, c_i, and d_i are single-bit values while a_i, b_i, and s_i are stored-double-carry digits with values in [0, 3]. Computing p_i in binary as $(d_{i+2}c_{i+1}w_i)_2$ directly determines the values of d_{i+2}, c_{i+1}, and w_i.

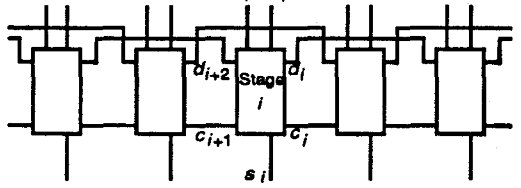

Fig. 1. BSDC adder with parallel transfers c_{i+1} and d_{i+2}.

The actual design of a BSDC adder will depend on the encoding used for a_i, b_i, and s_i. In Subsections A through D below, we examine 4 possible encodings and their associated designs. Subsequently, we will refer to these encodings as "Encoding A", "Encoding B", etc.

A. Two-Bit Binary Encoding

First, assume that we encode each BSDC digit in [0, 3] by its two-bit binary representation; i.e., by using the (2, 1) weighted code. Let the two-bit binary representations of a_i, b_i, and s_i be $(X_ix_i)_2$, $(Y_iy_i)_2$, and $(Z_iz_i)_2$, respectively. Then, the computation needed consists of a 2-bit addition to determine the sum of a_i and b_i and a full adder to compute the sum digit s_i. The following logic expressions specify our BSDC adder stage which requires 4 logic levels:

$$d_{i+2} = X_iY_i + X_ix_iy_i + x_iY_iy_i \tag{4a}$$
$$c_{i+1} = X_i \oplus Y_i \oplus x_iy_i = X_iY_i'x_i' + X_i'x_iY_i + X_iY_i'y_i' + X_i'Y_iy_i' + X_ix_iY_iy_i + X_i'x_iY_i'y_i \tag{4b}$$
$$w_i = x_i \oplus y_i = x_iy_i' + x_i'y_i \tag{4c}$$
$$Z_i = c_id_i + c_iw_i + d_iw_i \tag{4d}$$
$$z_i = c_i \oplus d_i \oplus w_i = c_id_iw_i + c_i'd_iw_i' + c_id_i'w_i' + c_i'd_i'w_i \tag{4e}$$

B. Three-Bit Unary Encoding

Another possible scheme is to encode the digits in [0, 3] as unary numbers; i.e., by using a (1, 1, 1) weighted code. Thus 0 and 3 will have unique representations (000 and 111), while 1 and 2 will each have 3 representations (001, 010, 001, and 011, 101, 110, respectively). The position sum $p_i = (d_{i+2}c_{i+1}w_i)_2$ is obtained by a 6-input parallel counter, and s_i has the 3-bit unary representation $(c_id_iw_i)_1$; i.e., it implies no further computation.

The needed 6-input parallel counter can be synthesized in several different ways. Each of the three outputs d_{i+2}, c_{i+1}, and w_i is a symmetric function of six logic variables. At one extreme is the two-level AND-OR realization which is perhaps too complex. Computation of d_{i+2} involves 4-out-of-6 majority decision on the inputs. The two-level circuit realizing this function needs 15 four-input AND gates followed by a 15-way OR circuit (the complement function requires the same number of gates, but with fewer input lines). The least significant sum bit is the odd parity over the 6 input bits and its generation by a two-level circuit implies 32 six-input AND gates plus a 32-way OR. At the other extreme is the use of half- and full-adder building blocks leading to 3 full adders, one half adder, and a 6-level logic circuit.

Neither of the above extremes is likely to be acceptable. We are thus led to a compromise solution with 4 logic levels. Here is one possible design. Two full adders (2 gate levels) convert the two groups of 3 inputs into 2-bit binary numbers $(X_ix_i)_2$ and $(Y_iy_i)_2$. A 2-level circuit then computes d_{i+2}, c_{i+1}, and w_i as specified for Encoding A. Thus whereas Encoding A requires a 2-bit adder plus a single-bit full adder per BSDC adder stage, Encoding B implies 2 full adders followed by a 2-bit adder. Thus we have both more gates and more wires here.

C. Unary Encoding with Unique Codes

Here the code vectors 000, 001, 011, and 111 are used to represent, 0, 1, 2, and 3, respectively. The interpretation of bit i in the three-bit code vector $v_3v_2v_1$ is that it is 1 if the digit being represented is at least i. The required logic, again assuming that the inputs a_i and b_i and the sum digit s_i are encoded as $\xi_iX_ix_i$, $\psi_iY_iy_i$, and $\zeta_iZ_iz_i$, respectively, is:

$$d_{i+2} = \xi_iy_i + \psi_ix_i + X_iY_i \tag{5a}$$
$$c_{i+1} = \xi_i\psi_i + \xi_iy_i' + Y_ix_i' + \xi_i'X_iY_i' + \psi_i'X_i'x_iy_i \tag{5b}$$
$$w_i = \xi_iy_i' + \psi_ix_i' + \xi_i\psi_iY_i + \xi_i'\psi_iX_i + X_i'x_iy_i' + Y_i'x_i'y_i + \xi_i'X_iY_i'y_i + \psi_i'X_i'Y_ix_i \tag{5c}$$
$$\zeta_i = c_id_iw_i \tag{5d}$$
$$Z_i = c_id_i + c_iw_i + d_iw_i \tag{5e}$$
$$z_i = c_i + d_i + w_i \tag{5f}$$

D. Three-Bit Weighted Encoding

Yet another variation is to use the (3, 2, 1) weighted code with code vectors 000, 001, 010, and 100 representing 0, 1, 2, and 3, respectively. Then the position sum is obtainable by a circuit that is much simpler and/or faster than a 6-input parallel counter. The required logic, assuming that the inputs a_i and b_i and the sum digit s_i are encoded as $\xi_iX_ix_i$, $\psi_iY_iy_i$, and $\zeta_iZ_iz_i$, respectively, is:

$$d_{i+2} = \xi_i\psi_i + \xi_iY_i + \xi_iy_i + \psi_iX_i + \psi_ix_i + X_iY_i \tag{6a}$$
$$c_{i+1} = x_iy_i + \xi_iY_i'y_i' + \psi_iX_i'x_i' + \xi_i'X_iY_i + \psi_i'X_iY_i' \tag{6b}$$
$$w_i = \xi_i\psi_i'y_i' + \xi_i'\psi_ix_i' + \xi_i'x_i'y_i + \psi_i'x_iy_i' \tag{6c}$$
$$\zeta_i = c_id_iw_i \tag{6d}$$
$$Z_i = c_id_iw_i' + c_i'd_iw_i + c_id_i'w_i \tag{6e}$$
$$z_i = c_i'd_iw_i' + c_id_i'w_i' + c_i'd_i'w_i \tag{6f}$$

Strong error checking capability is provided if one uses the 1-out-of-4 code 0001, 0010, 0100, 1000 to represent 0-3. This complicates the logic and also increases the number of wires. Thus, unlike the 1-out-of-3 or "three-rail" encoding of Takagi et al [Taka87], this "four-rail" encoding is not worthy of further consideration.

V. New Tree Multipliers

Application of the BSDC number system defined in Section IV to implement a high-speed tree multiplier for unsigned numbers is straightforward if one specifies how the standard binary multiples of the multiplicand are combined to form the initial BSDC numbers and how the final BSDC result is converted to standard binary. In the following discussion, the circuit and time complexity associated with the final high-speed adder will be ignored since this is shared by all the schemes under consideration. Circuit complexity is measured in terms of gate count. This is a suitable approximation since all the circuits being considered have a binary-tree topology and all the gates (with very few exceptions) have fan-ins of 2 to 4. Time complexity is measured by the number of gate levels which is appropriate for the same reasons. However, in comparing the two-bit encoding A to three-bit encodings B, C, and D, the larger wire area of the latter must also be considered.

As a point of reference, Vuillemin's design [Vuil83] has the following cost and delay parameters

$$\kappa_V(n) = 18\,\gamma(\lceil n/2\rceil, n+1, 2) \tag{7a}$$
$$\delta_V(n) = 4\lceil\log_2\lceil n/2\rceil\rceil \tag{7b}$$

where $\gamma(1, m, \sigma) = 0$ and

$$\gamma(k, m, \sigma) = \lfloor k/2\rfloor m + \gamma(\lceil k/2\rceil, m+\sigma+1, 2\sigma) \tag{8}$$

The function γ provides the total number of BSC adder digit slices required in the design, with each digit slice needing two full adders or 18 gates. The recursive definition of γ reflects the fact that to add k BSC numbers each having m digits and shifted to the left by σ digits with respect to the previous one, we must first add $\lfloor k/2\rfloor$ pairs to get a set of $\lceil k/2\rceil$ $(m+\sigma+1)$-digit BSC numbers whose relative shift distance is 2σ (see Fig. 2).

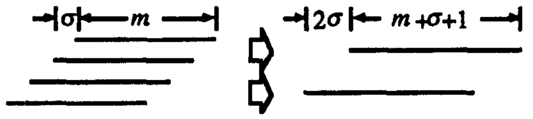

Fig. 2. Parameters for recursive binary reduction.

Actually, Eq. (8) is slightly pessimistic since it assumes the expansion of operand lengths by one digit with every addition. Referring to Fig. 2, we note that the addition of two m-digit operands, one of which is shifted by $\sigma\geq 2$ digits with respect to the other, creates an $(m+\sigma+1)$-digit result. The extra digit comes from the carry generated by the leftmost digit. If a double-carry were possible at the left end, the length expansion would have been by two digits. For $\sigma\geq 2$ double carry is impossible and the "single" carry will generate a new leftmost digit that is 0 or 1. Hence, in the next step, length expansion does not occur at all, as the digit 1 produces no carry. Accordingly, a more accurate version of (8), with the recurrence unfolded twice, is

$$\gamma(k, m, \sigma) = \lfloor k/2\rfloor m + \lfloor\lceil k/2\rceil/2\rfloor(m+\sigma+1) + \gamma(\lceil\lceil k/2\rceil/2\rceil, m+3\sigma+1, 4\sigma) \tag{9}$$

where $\gamma(1, m, \sigma) = 0$ and $\gamma(2, m, \sigma) = m$. The difference of the two versions is insignificant and the inaccuracy affects all designs virtually uniformly. Thus, we use the simpler Eq. (8) in our subsequent analyses.

The design of Takagi et al [Taka85], when used without Booth's encoding, has the following parameters:

$$\kappa_T(n) = 13\,\gamma(n, n, 1) \tag{10a}$$
$$\delta_T(n) = 4\lceil\log_2 n\rceil \tag{10b}$$

A. Two-Bit Binary Encoding

Clearly, three standard binary numbers can be combined by a set of full adders into a BSDC number with two-bit binary encoding of digits if the sum and carry bits are kept in the same digit position. This requires two logic levels. With this encoding, the final BSDC result can be converted to standard binary by a conventional carry-propagate adder if the more significant bit in each position is shifted to the next higher adder position. Thus given n operands, the total gate count and delay (excluding the contribution of the final high-speed adder that converts a BSDC number into a standard binary number) are given by:

$$\kappa_A(n) = 9(n-1)\lfloor (n+1)/3 \rfloor + 23\ \gamma(\lceil n/3 \rceil, n+2, 3) \qquad (11a)$$
$$\delta_A(n) = 2 + 4 \lceil \log_2 \lceil n/3 \rceil \rceil \qquad (11b)$$

The expression for κ_A is based on the fact that $\lfloor (n+1)/3 \rfloor$ $(n-1)$-bit carry-save adders are needed in the initial stage, followed by a (possibly incomplete) tree of BSDC adders to combine the $\lceil n/3 \rceil$ BSDC numbers each with $n+2$ digits.

B. Three-Bit Unary Encoding

Given the (1,1,1) unary encoding of each BSDC digit, three binary numbers directly define a BSDC number. As for the final conversion, a row of full adders is needed to convert the single BSDC number into a BSC number before the final carry-propagate addition. Here we have:

$$\kappa_B(n) = 32\ \gamma(\lceil n/3 \rceil, n+2, 3) + 18n \qquad (12a)$$
$$\delta_B(n) = 4 \lceil \log_2 \lceil n/3 \rceil \rceil + 2 \qquad (12b)$$

The expression for κ_B is based on the fact that $2n$ full adders are needed in the final stage yielding the product in BSC form. These full adders are preceded by a binary tree of BSDC adders to combine the $\lceil n/3 \rceil$ BSDC numbers each with $n+2$ digits. We note that the delays $\delta_A(n)$ and $\delta_B(n)$ are equal whereas the gate counts $\kappa_A(n)$ and $\kappa_B(n)$ cannot be compared in a straightforward manner. More on this later.

C. Unary Encoding with Unique Codes

The BSDC adder in this case is simpler than that required with Encoding B (25 gates versus 32 gates per digit slice). Thus, what is lost in the initial conversion of binary numbers to the required encoding may be regained in the rest of the design. Initially, three n-bit binary numbers can be converted to a BSDC number with this encoding using 6 gates per bit position, as specified by the expressions for ζ_i, Z_i, and z_i in (5). Then, we have:

$$\kappa_C(n) = 6n \lfloor (n+1)/3 \rfloor + 25\ \gamma(\lceil n/3 \rceil, n+2, 3) + 18n \qquad (13a)$$
$$\delta_C(n) = 2 + 4 \lceil \log_2 \lceil n/3 \rceil \rceil + 2 \qquad (13b)$$

The expression for κ_C in (13) is justified as was done for κ_B in (12), except that the first term $6n \lfloor (n+1)/3 \rfloor$ is due to the initial converters as explained above. We note that the delay $\delta_C(n)$ is greater than $\delta_A(n) = \delta_B(n)$ whereas the gate count $\kappa_C(n)$ cannot be compared to $\kappa_A(n)$ and $\kappa_B(n)$ in general terms (see Section VI).

D. Three-Bit Weighted Encoding

For the (3, 2, 1) encoding, a set of n parallel circuits, each implementing the expressions for ζ_i, Z_i, and z_i in (6), can convert 3 standard binary numbers to BSDC representation. At the bottom of the binary tree, a row of logical OR gates (2 per digit position) can be used to compute the two-bit representation $(b + c, a + c)$ of a three-bit (3, 2, 1)-encoded BSDC digit (c, b, a). Thus we have:

$$\kappa_D(n) = 9n \lfloor (n+1)/3 \rfloor + 27\ \gamma(\lceil n/3 \rceil, n+2, 3) + 4n \qquad (14a)$$
$$\delta_D(n) = 2 + 4 \lceil \log_2 \lceil n/3 \rceil \rceil + 1 \qquad (14b)$$

The expression for κ_D in (14) is justified as was done for κ_C in (13), but the initial converters are slightly more complex. Note that $\delta_D(n)$ is 1 unit less than $\delta_C(n)$. The relationship between the gate counts $\kappa_D(n)$ and $\kappa_C(n)$ is less obvious. Since $\kappa_D(n)$ has larger coefficients for the first two terms that are quadratic in n and a smaller coefficient for its linear term, the benefit of Encoding D, if any, will materialize for short word lengths.

VI. Comparisons and Discussion

Table II shows the delays of the various implementations for different word lengths in gate levels. Clearly, the delay of T is always greater than that of V and the delays of C and D are always greater than $\delta_A(n) = \delta_B(n)$. Thus is high-speed operation is desired, the choice is always between V and A/B. As can be seen from Table II, the optimal design with respect to speed depends on the value of n. When n is greater than a power of 2 but no more than 1.5 times that power, the A or B design is faster by 2 gate levels. Otherwise, V is faster by 2 gate levels. Fig. 3 supplies an explanation of this result comparing V to A for n = 24. Each box in Fig. 3 represents 2 CSA (4 logic) levels. The first circuit level in the bottom diagram of Fig. 3 consists of simple CSAs with 2 logic levels.

Table II
The Delays of Various Binary-Reduction Designs in Gate Levels as a Function of *n*.

n	V	T	A,B	C	D
7 - 8	8	12	10	12	11
9 - 12	12	16	10	12	11
13 - 16	12	16	14	16	15
17 - 24	16	20	14	16	15
25 - 32	16	20	18	20	19
33 - 48	20	24	18	20	19
49 - 64	20	24	22	24	23
65 - 96	24	28	22	24	23
97 -128	24	28	26	28	27
129 -192	28	32	26	28	27

Table III shows that Design V always has the lowest cost. Thus in cases where it also has the smallest number of gate levels according to Table II, it is the best design. Design A reaches its lowest relative complexity where it also has a speed advantage over V. Thus, Design A is promising for word lengths that are equal or close to 12, 24, 36, 48, and 72 bits. Designs B, C, and D are always more complex than A in view of their 3-wire encodings. However, C and D offer some error detection capability and may be worth considering for fault-tolerant designs. Design B is always inferior to A and, unlike C or D, has no redeeming feature.

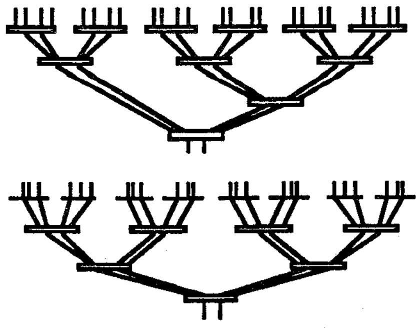

Fig. 3. Binary-tree realizations of 24-operand addition.

Our comparisons are based on the requirement for adding n operands of length n with relative shifts of 1 bit. If small $g \times g$ building-block multipliers (rather than AND gates) are used to generate the partial products matrix, then $(n/g)^2$ operands of length g^2 and varying relative shifts will have to be added. The effect of the new group size parameter g on the relative efficiencies of the various schemes discussed earlier has not been investigated and constitutes a possible area for further research. Some results along these lines have been presented by Wey and Chang [Wey90], although their approach is quite different.

Table III
The Costs of Various Binary-Reduction Designs in Number of Gates for Selected Values of n.

n	V	T	A	B	C	D
8	540	845	741	912	888	896
12	1368	1989	1454	1688	1654	1722
16	2394	3484	3090	3648	3393	3619
20	3960	5577	4670	5192	4975	5417
24	5562	7904	6279	6864	6609	7251
28	7542	10712	9386	10520	9841	10831
32	9720	13871	11970	12960	12363	13745
36	12744	17849	14475	15528	14865	16587
40	15498	21840	18455	20048	19180	21508
44	18630	26312	22043	23384	22402	25178
48	21960	31135	25444	26848	25772	29028
52	25830	36556	31470	33864	32277	36415
56	29736	42211	36062	38096	36367	41093
60	34020	48347	40359	42456	40605	45951
64	38502	54834	46683	49536	47400	53752

A final point concerns the practical motivation for limited-carry and parallel-carries addition processes from the point of view of circuit complexity which affects both the practicality of certain designs and their speed-cost tradeoffs. In any redundant number system that does not support carry-free addition, the sum digit s_i can be found directly as a function of 3 digits from each operand (i.e. a_i, a_{i-1}, a_{i-2}, b_i, b_{i-1}, and b_{i-2}). In a radix-2 redundant system, each digit requires at least 2 bits for its representation. This means that the circuit to generate s_i has to compute at least 2 logic functions of 12 or more variables. For higher radices, the situation is even worse. Thus the limited-carry process is a way to systematically reduce the circuit complexity by condensing the information from a_{i-2} and b_{i-2} into the "range estimate" e_{i-1} and then condensing the information from e_{i-1}, a_{i-1}, and b_{i-1} into the transfer digit t_i. Use of parallel carries, as discussed in this paper, breaks the above sequential chain by allowing the $(i-2)$th stage to directly send a bit of information to the ith stage.

VII. Conclusion

We have presented a unified view of the regular binary-tree reduction feature of certain VLSI fast multiplier designs. We have argued that existing designs are stored-carry and stored-borrow instances of a general paradigm and have shown that similar advantages can be gained with other redundant representations. We demonstrated that one of these alternate representations results in faster parallel multipliers for certain word lengths and that others provide potential advantages with certain design parameters and/or technological constraints. Intuitively, the improvements result from better utilization of the 2 binary signals needed to encode a 3-valued digit in [0, 2] or [–1, 1]. This latter signed digit set is inefficient in 2 ways. First, it utilizes only 3 of 4 possible signal combinations. Second, even the three values are not fully utilized in that negative weights appear only in specific patterns in the partial products bit-matrix obtained from 2's-complement operands, particularly if Booth's encoding is not used.

Our method provides additional design points for fast multipliers based on binary trees. Clearly, the availability of a spectrum of alternatives is beneficial in that even the most ingenious designers cannot foresee all future design needs or potentials/limitations of emerging technologies. One technological development that could make our designs very attractive is the availability of high-speed 4-valued logic circuits [Etie88], [Kawa94], [Parh96], [Smit88]. Our BSDC carry-free addition scheme is also applicable to the design of large (accumulative) parallel counters [Parh95], since the number of levels and the irregularity of connections in parallel counters are similar to those of CSA trees used in parallel multipliers.

References

[Etie88] Etiemble, D. and M. Israel, "Comparison of Binary and Multivalued ICs According to VLSI Criteria", *Computer*, Vol. 21, No. 4, pp. 28-42, Apr. 1988.

[Hara87] Harata, Y. et al, "A High-Speed Multiplier Using a Redundant Binary Adder Tree", *IEEE J. Solid- State Circuits*, Vol. 22, No. 1, pp. 28-34, Feb. 1987.

[Kawa94] Kawahito, S. et al, "High-Speed Area-Efficient Multiplier Design Using Multiple-Valued Current-Mode Circuits", *IEEE Trans. Computers*, Vol. 43, No. 1, pp. 34-42, Jan. 1994.

[Parh89] Parhami, B., "A New Method for Designing Highly Parallel Binary Multipliers", *Proc. 3rd Int'l Parallel Processing Symp.*, Mar. 1989, pp. 176-185.

[Parh90] Parhami, B., "Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations", *IEEE Trans. Computers*, Vol. 39, No. 1, pp. 89-98, Jan. 1990.

[Parh93] Parhami, B., "On the Implementation of Arithmetic Support Functions for Generalized Signed-Digit Number Systems", *IEEE Trans. Computers*, Vol. 42, No. 3, pp. 379-384, Mar. 1993.

[Parh94] Parhami, B., "Implementation Alternatives for Generalized Signed-Digit Addition", *Proc. Asilomar Conf. Signals, Systems, and Computers*, Oct./Nov. 1994, pp. 157-161.

[Parh95] Parhami, B. and C.-H. Yeh, "Accumulative Parallel Counters", *Proc. Asilomar Conf. Signals, Systems, and Computers*, Oct./Nov. 1995, pp. 966-970.

[Parh96] Parhami, B., "Comments on 'High-Speed Area-Efficient Multiplier Design Using Multiple-Valued Current Mode Circuits' ", *IEEE Trans. Computers*, Vol. 45, No. 5, pp. 637-638, May 1996.

[Smit88] Smith, K.C., "Multiple-Valued Logic: A Tutorial and Appreciation", *Computer*, Vol. 21, No. 4, pp. 17-27, Apr. 1988.

[Swar90] Swartzlander, E.E. Jr., *Computer Arithmetic*, Vol. I, IEEE Computer Society Press, 1990 (Reprints of key papers on tree multipliers, parallel counters, etc.).

[Taka85] Takagi, N., H. Yasuura, and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree", *IEEE Trans. Computers*, Vol. 34, No. 9, pp. 1310-1317, Sep. 1985.

[Taka87] Takagi, N. and S. Yajima, "On-Line Error-Detectable High-Speed VLSI Multiplier Using Redundant Binary Representation and Three-Rail Logic", *IEEE Trans. Computers*, Vol. 36, pp.789-796, Nov.1987.

[Vuil83] Vuillemin, J., "A Very Fast Multiplication Algorithm for VLSI Implementation", *Integration: the VLSI J.*, Vol. 1, pp. 39-52, 1983.

[Wey90] Wey, C.-L. and T.-Y. Chang, "Design and Analysis of VLSI-Based Parallel Multipliers", *IEE Proc. Pt. E*, Vol. 137, No. 4, pp. 328-336, July 1990.