Optimal-Depth Circuits for Prefix Computation and Addition

Chi-Hsiang Yeh
Dept. of Electrical & Computer Engineering
Queen's University
Kingston, ON K7L 3N6, Canada

E.A. Varvarigos, and B. Parhami
Dept. of Electrical & Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA

Abstract

Addition and prefix computation are among the most fundamental problems in arithmetic and algebraic computation. In this paper, we present efficient circuits for performing prefix computation and addition with small depth and size and flexible fan-in (i.e., the maximum fan-in can be selected as a small constant or a larger constant/nonconstant number). In particular, we show that any prefix operation of n inputs can be computed using a circuit of fan-in k, depth $\log_k n + o(\log_k n) + O(1)$, *gate complexity* $O(n)$, *and edge complexity* $O(n\log^{\overbrace{*\,*\,\cdots\,*}^{d-1}} n)$, *for any constant integer d. We show that the sum of two n-bit numbers can be found using an AND-OR circuit of fan-in k, depth* $\log_k n + o(\log_k n) + O(1)$, *and edge complexity* $O(n(\log^{\overbrace{*\,*\,\cdots\,*}^{d-1}} n)^2)$, *for any constant integer d. In particular, the depths of our circuits for prefix computation and addition are optimal within a factor of* $1 + o(1)$, *for any fan-in* $k = n^{o(1)}$.

1 Introduction

The delay required to perform addition is crucial to the performance of many computationally intensive applications. Prefix computation is also an important problem that appears often in arithmetic and algebraic computations [5]. There have been a very large number of papers investigating fast and/or cost-effective solutions to these problems [1, 2, 4, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17]. In this paper, we propose efficient circuits to perform prefix computation and addition. Our constructions provide efficient tradeoffs between fan-in, depth, and edge complexity, and have depths that are considerably smaller than those in [10, 11].

The *depth* of a circuit is defined as the length of the longest path from any input to any output node of the circuit, while the *fan-in* of a circuit is defined as the largest fan-in among all the gates contained in it. The *edge complexity* and the *gate complexity* of a circuit are defined as the number of edges and the number of gates in the circuit, respectively. In this paper, we show that any prefix (associative) operation of n inputs can be performed using a circuit of depth $\log_k n + o(\log_k n) + O(1)$, gate complexity $O(n)$, edge complexity $O(n\log^{\overbrace{*\,*\,\cdots\,*}^{d-1}} n)$ †, and fan-in k, for any constant integer d, assuming that a gate to perform the operation is available. The depth of this circuit is optimal within a factor of $1 + o(1)$ when $\log_k n$ is not a constant. The problem of computing all prefixes of n inputs, given any associative operator, has been considered [10], where a circuit of depth $2\log_k n + o(\log_k n) + O(1)$, edge complexity $O(n\log^* n)$, and fan-in k was proposed. Our result improves the depth of the circuit given in [10] by a factor of 2 asymptotically and further reduces the edge complexity when $d > 2$.

We also show that the sum of two n-bit numbers can be found using an AND-OR circuit of depth $\log_k n + o(\log_k n) + O(1)$, edge complexity $O(n(\log^{\overbrace{*\,*\,\cdots\,*}^{d-1}} n)^2)$, and fan-in k, for any constant integer d. The depth of our circuit for addition is optimal within a factor of $1 + o(1)$ when $\log_k n$ is not a constant. In [10], it was shown that addition can be performed using an AND-OR circuit of depth $4\log_k n + o(\log_k n) + O(1)$, edge complexity $O(n(\log^* n)^2)$, and fan-in k. Our result improves the depth of the circuit by a factor of 4 asymptotically and also has smaller edge complexity for $d > 2$. The resulting addition circuits can be used as building blocks for efficient multiplication circuits. In particular, they lead to a unit-weight threshold circuit [3, 10, 17] to compute the product of two n-bit integers, which has depth approximately equal to $3\log_k n + 1.44\log_2 d$, edge complexity $O(n^{2+1/d}\log(d+1))$, and fan-in k [17].

† $\log_2^{\overbrace{*\,*\,\cdots\,*}^{d}} n = \min\{l \geq 0 | \underbrace{\log_2^{\overbrace{*\,*\,\cdots\,*}^{d-1}} \cdots \log_2^{\overbrace{*\,*\,\cdots\,*}^{d-1}}}_{l} n \leq 1\}$, $d \geq 1$. See Section 2 for the definitions of $\log^* n$ and $\log^{**} n$.

0-7803-6514-3/00/$10.00©2000 IEEE

2 Prefix Computation

We let $\circ$ be an arbitrary associative binary operator; that is, $(a \circ b) \circ c = a \circ (b \circ c)$. In the prefix computation problem we want to compute the partial "products" $z_i = x_1 \circ x_2 \circ x_3 \circ \cdots \circ x_i$ for all $1 \leq i \leq n$. In what follows, we will assume that the functional gate for computing the binary function $\circ$ is available, and we will obtain a depth-optimal circuit that has linear gate complexity and almost linear edge complexity.

Before we describe the results, it is useful to introduce some notation. For $n \geq 1$, we let

$$\log_2^* n = \min\{l \geq 0 | \underbrace{\log_2 \cdots \log_2}_{l} n \leq 1\}.$$

That is, $\log_2^* n$ is the minimum number of terms (levels) in the iterated exponential $2^{2^{\cdot^{\cdot^{2}}}}$ required to make it satisfy $2^{2^{\cdot^{\cdot^{2}}}} \geq n$, and is a function that grows very slowly with n. For example, the values that $\log_2^*$ takes at $n = 1, 2, 4, 16, 65536$, and 2^{65536} are $0, 1, 2, 3, 4$, and 5, respectively. We also define

$$\log_2^{**} n = \min\{l \geq 0 | \underbrace{\log_2^* \cdots \log_2^*}_{l} n \leq 1\},$$

and, more generally,

$$\log_2^{\overbrace{*\cdots*}^{d}} n = \min\{l \geq 0 | \underbrace{\log_2^{\overbrace{*\cdots*}^{d-1}} \cdots \log_2^{\overbrace{*\cdots*}^{d-1}}}_{l} n \leq 1\},$$

for any integer $d \geq 1$. Clearly, $\log_2^{**} n$ grows even slower with n than $\log_2^* n$, and for the values of n that we expect in applications, $\log_2^{**} n$ is a very small number.

Theorem 2.1 *The prefixes x_1, $x_1 \circ x_2$, $x_1 \circ x_2 \circ x_3$, ..., $x_1 \circ x_2 \circ x_3 \circ \cdots \circ x_n$ can be computed using a circuit of depth $\log_k n + o(\log_k n) + O(1)$, gate complexity $O(n)$, edge complexity $O(n \log^{\overbrace{*\cdots*}^{d-1}} n)$, and fan-in k, for any constant integer d.*

Proof: The proof will use induction on d. We first present the prefix circuit for the case $d = 1$. We can assume, without loss of generality, that n is a power of 2. We partition the inputs $x_1, x_2, x_3, \ldots, x_n$ into two groups of size $n/2$, then divide each such group into two groups of size $n/4$, and so on, until we obtain groups of size 1. Clearly, there are $2n-2$ groups overall and $\log_2 n$ different group sizes.

The product of the elements in each of the $2n-2$ groups can be computed using a circuit of depth $\lceil \log_k \frac{n}{2} \rceil$, gate complexity $O(n)$, edge complexity $O(n \log n)$, and fan-in k. Given the preceding circuit, the product $P_i = x_1 \circ x_2 \circ \cdots \circ x_i$ for any integer $i \in [1, n]$ can be obtained by computing the product of at most $\log_2 n$ groups of different sizes. As a result, the products $P_i = x_1 \circ x_2 \circ \cdots \circ x_i$, $i = 1, 2, 3, \ldots, n$, can be obtained using another circuit of depth $\lceil \log_k \log_2 n \rceil$, gate complexity $O(n)$, and edge complexity $O(n \log n)$. Thus, the partial products x_1, $x_1 \circ x_2$, $x_1 \circ x_2 \circ x_3$, ..., $x_1 \circ x_2 \circ \cdots \circ x_n$ can be simultaneously computed using a circuit of depth $\log_k n + o(\log_k n) + O(1)$, gate complexity $O(n)$, edge complexity $O(n \log n)$, and fan-in k.

For the inductive step, we assume that the prefix of n inputs can be computed using a circuit of depth $\log_k n + o(\log_k n) + O(1)$, gate complexity $O(n)$, edge complexity $O(n \log^{\overbrace{*\cdots*}^{d-1}} n)$, and fan-in k, for some integer $d \geq 1$. We will show that the prefix computation can be performed using a circuit of depth $\log_k n + o(\log_k n) + O(1)$, gate complexity $O(n)$, edge complexity $O(n \log^{\overbrace{*\cdots*}^{d}} n)$, and fan-in k. We partition the inputs $x_1, x_2, x_3, \ldots, x_n$ into consecutive groups of size $\log_2^{\overbrace{*\cdots*}^{d-1}} n$, and then divide each group into consecutive groups of size $\log_2^{\overbrace{*\cdots*}^{d-1}} \log_2^{\overbrace{*\cdots*}^{d-1}} n$, and so on, until we obtain groups of size 1. Clearly, there are $\log_2^{\overbrace{*\cdots*}^{d}} n$ different group sizes and fewer than $2n$ groups. To compute the product of the elements in each of the groups, we use a circuit of depth $\lceil \log_k \log_2^{\overbrace{*\cdots*}^{d-1}} n \rceil$, gate complexity $O(n)$, and edge complexity $O(n \log^{\overbrace{*\cdots*}^{d}} n)$. If a group has been divided into subgroups with partial products $p_1, p_2, p_3, \ldots, p_t$, then we compute the prefix $p_1, p_1 \circ p_2, p_1 \circ p_2 \circ p_3, \ldots, p_1 \circ p_2 \circ \cdots \circ p_t$. We also compute the prefixes at the top level by combining groups of the largest size. All these prefixes can be computed using a circuit of depth $\log_k n + o(\log_k n) + O(1)$, gate complexity $O(n)$, and edge complexity $O(n \log^{\overbrace{*\cdots*}^{d}} n)$ by the inductive hypothesis. It can be seen that a product $x_1 x_2 \cdots x_i$ for any integer $i \in [1, n]$ can be obtained by computing the product of at most $\log^{\overbrace{*\cdots*}^{d}} n$ products from these obtained prefixes. As a result, the partial products $P_i = x_1 \circ x_2 \circ \cdots \circ x_i$, $i = 1, 2, 3, \ldots, n$ can be computed using another circuit of depth $\lceil \log_k \log_2^{\overbrace{*\cdots*}^{d}} n \rceil$, gate complexity $O(n)$, and edge complexity $O(n \log^{\overbrace{*\cdots*}^{d}} n)$. This shows that all partial products can be computed using a circuit of depth $\log_k n + o(\log_k n) + O(1)$, gate complexity $O(n)$, edge complexity

$O(n\log^{\overbrace{*\cdots*}^{d}} n)$, and fan-in k, which completes the induction. □

In [10], it was shown that prefix computation can be performed using a circuit of depth $2\log_k n + o(\log_k n) + O(1)$, edge complexity $O(n\log^* n)$, and fan-in k. Theorem 2.1 improves the depth of the circuit by a factor of 2 asymptotically, and further reduces the edge complexity (when d is a constant larger than 2). Since a trivial lower bound on the depth required to compute a function with n inputs is $\log_k n$, where k is the maximum fan-in of a gate in the circuit, the depth of our prefix circuit is optimal within a factor of $1+o(1)$ from the lower bound, provided that $\log_k n$ is not a constant.

3 AND-OR Circuits for Addition

The addition of two operands is the most frequently encountered operation in computer arithmetic units, and the speed at which it can be executed is crucial to the performance of almost all computing systems. In what follows, we use the techniques developed in Section 2 for prefix computation to obtain an addition circuit that has almost linear edge complexity and is depth-optimal within a factor of $1+o(1)$ when $\log_k n$ is not a constant.

Theorem 3.1 *The sum of two n-bit integers can be computed using an AND-OR circuit of depth* $\log_k n + o(\log_k n) + O(1)$, *edge complexity* $O(n(\log^{\overbrace{*\cdots*}^{d-1}} n)^2)$, *and fan-in k, for any constant integer d.*

Proof: Let $X = (x_{n-1},\ldots,x_0)_2$ and $Y = (y_{n-1},\ldots,y_0)_2$ be the two n-bit numbers to be added. We will first present the addition circuit for the base case $d = 2$, and then use induction on d to obtain the result for the general case.

The addition of the two numbers is done in 6 phases:

- Phase 1: In this phase, we compute the *carry-propagate* and *carry-generate* for each pair of the input bits x_i and y_i. This is in turn done in two parallel steps:
  - Phase 1a: For $i = 0,1,2,\ldots,n-1$, we compute the carry-propagate $p_i = x_i + y_i$ using one layer of OR gates.
  - Phase 1b: For $i = 0,1,2,\ldots,n-1$, we compute the carry-generate $g_i = x_i y_i$ using one layer of AND gates in parallel with Phase 1a.
- Phase 2: In this phase, we compute the *level-2 group-carry-propagate* and *group-carry-generate* for each of the *level-2 groups*, which are defined as follows. We partition the carry-propagates $p_0, p_1, p_2, \ldots, p_{n-1}$ and the carry-generates $g_0, g_1, g_2, \ldots, g_{n-1}$ into consecutive level-2 groups of size $\log_2 n$, and then partition the elements in each of the groups into consecutive level-2 groups of size $\log_2\log_2 n$, and repeat this procedure until we obtain groups of size 1. Clearly, there are $\log_2^* n$ different level-2 group sizes and fewer than $2n$ level-2 groups.
  - Phase 2a: We compute level-2 group-carry-propagate $P_{i+h-1:i}$ for each level-2 group with elements $p_{i+h-1}, p_{i+h-2}, \ldots, p_{i+1}, p_i$ according to
    $$P_{i+h-1:i} = \bigwedge_{l=i}^{i+h-1} p_l = p_{i+h-1} \wedge p_{i+h-2} \wedge \cdots \wedge p_i.$$
  - Phase 2b: We compute level-2 group-carry-generate $G_{i+h-1:i}$ for each level-2 group with elements $p_{i+h-1}, p_{i+h-2}, \ldots, p_{i+1}, p_i$, and $g_{i+h-1}, g_{i+h-2}, \ldots, g_{i+1}, g_i$ according to
    $$G_{i+h-1:i} = \bigvee_{j=i}^{i+h-1} \left(g_j \bigwedge_{l=j+1}^{i+h-1} p_l \right),$$
    using a prefix circuit (Theorem 2.1 with $d = 2$).
- Phase 3: We compute the *level-1 group-carry-propagate* and *group-carry-generate* for each of the *level-1 groups*, which are defined as follows. We partition all the level-2 groups of size $\log_2 n$ into two level-1 groups of size $\frac{n}{2\log_2 n}$ (that is, each of these level-1 groups contains $\frac{n}{2\log_2 n}$ level-2 groups of size $\log_2 n$), and then divide each of the level-1 group into two level-1 groups of size $\frac{n}{4\log_2 n}$, and continue this procedure until we obtain level-1 groups of size 1. If a level-2 group is divided into u level-2 subgroups during Phase 2, we also partition these subgroups into two level-1 groups of size $u/2$, then divide each of the level-1 groups into two level-1 groups of size $u/4$, and repeat this procedure until we obtain level-1 groups of size 1.
  - Phase 3a: We compute the level-1 group-carry-propagate of a level-1 group according to
    $$P_{i_{h+1}-1:i_1} = P_{i_{h+1}-1:i_h} \wedge P_{i_h-1:i_{h-1}} \wedge \cdots \wedge P_{i_2-1:i_1},$$
    where $P_{i_{h+1}-1:i_h}$, $P_{i_h-1:i_{h-1}}, \ldots, P_{i_3-1:i_2}, P_{i_2-1:i_1}$ are the level-2 group-carry-propagates of the level-2 groups contained in it.
  - Phase 3b: We compute the level-1 group-carry-generate for each of the level-1 groups using a prefix circuit (Theorem 2.1 with $d = 2$), whose inputs are level-2 group-carry-propagates and group-carry-generates of the level-2 groups contained in the level-1 group.

- Phase 4: We compute the *level-2 section-carry-propagate* and *section-carry-generate* for the sections starting from each of the level-2 groups and ending at the rightmost level-2 group (within the same level-1 group). This is done in two parallel steps:
  - Phase 4a: If a level-2 group is divided into u level-2 subgroups during Phase 2, then we determine whether the *section* i, $i = 1,2,3,...,u$, which contains the i^{th}, $(i+1)^{th}$, $(i+2)^{th}$, ..., u^{th} level-2 subgroups, can propagate a carry. This can be done by an AND gate when $k \geq \log_2 u$ (or a tree of AND gates otherwise), whose inputs are the $\log_2 u$ level-1 group-carry-propagates. These results are called level-2 section-carry-propagates.
  - Phase 4b: For $i = 1,2,3,..., \frac{n}{\log_2 n}$, we determine whether there is a carry generated and propagated to the leftmost bit of the i^{th} leftmost level-2 group of size $\log_2 n$. This can be done by using a prefix circuit (Theorem 2.1 with $d = 2$), whose inputs are $\log_2(\frac{n}{\log_2 n})$ level-1 group-carry-propagates and $\log_2(\frac{n}{\log_2 n})$ level-1 group-carry-generates. If a level-2 group is divided into u level-2 subgroups during Phase 2, we also determine whether there is a carry generated within the former level-2 group and propagated to the leftmost bit of the i^{th} level-2 subgroup for $i = 1,2,3,...,u$. The results are called level-2 section-carry-generates.
- Phase 5: We determine each of the carry bits c_i, $i = 0,1,2,...,n$, using a prefix circuit, whose inputs are $\log_2^* n$ level-2 section-carry-propagates and $\log_2^* n$ level-2 section-carry-generates.
- Phase 6: Each of the output bits z_i, $i = 0,1,2,...,n$, can be computed by

$$z_i = c_i \oplus x_i \oplus y_i = (c_i \wedge (x_i \odot y_i)) \vee (\bar{c}_i \wedge (x_i \oplus y_i)).$$

The edge complexity of the circuit is dominated by the computation of the group-carry-generates in Phases 2b and 3b, the section-carry-generates in Phase 4b, and the carry bits in Phase 5, which require a total of $O(n(\log^* n)^2)$ edges. The depth of the circuit is dominated by the computation of the group-carry-propagates and the group-carry-generate for the level-1 groups of size $\frac{n}{2\log_2 n}$ (Phase 3), which requires a circuit of depth $\log_k n \pm o(\log_k n) + O(1)$.

The preceding addition circuit can be extended to the general case $d \geq 2$ by using a technique similar to that used in the prefix computation circuit of Theorem 2.1. In particular, for the case $d = 3$, we partition the carry-propagates $p_0, p_1, p_2, \cdots, p_{n-1}$ and the carry-generates $g_0, g_1, g_2, \cdots, g_{n-1}$ into consecutive *level-3 groups* of size $\log_2^* n$, and then divide each group into consecutive groups of size $\log_2^* \log_2^* n$, and continue this procedure until we obtain groups of size 1. We then compute *level-3 group-carry-propagates* and *group-carry-generates*. In turn, we partition all the level-3 groups of size $\log_2^* n$ into level-2 groups of size $\log_2(\frac{n}{\log_2^* n})$ and then divide each of the level-2 group into level-2 groups of size $\log_2 \log_2(\frac{n}{\log_2^* n})$, and continue this procedure until we obtain groups of size 1.

If a level-3 group is divided into u level-3 subgroups, we also partition these subgroups into level-2 groups of size $\log_2 u$, divide each of them into level-2 groups of size $\log_2 \log_2 u$, and continue this procedure until we obtain groups of size 1. We then compute the level-2 group-carry-propagates and group-carry-generates. Similar to the base case $d = 2$, we also compute level-1 group-carry-propagates, level-1 group-carry-generates, level-2 section-carry-propagates, and level-2 section-carry-generates. We then compute *level-3 section-carry-propagates* and *section-carry-generates*, whose definitions are similar to their level-2 counterparts. In particular, the level-3 section-carry-generates are computed using a prefix circuit (Theorem 2.1 with $d = 3$). Finally, we compute the carry bits using a prefix circuit, whose inputs are $\log_2^{**} n$ level-3 section-carry-propagates and $\log_2^{**} n$ level-3 section-carry-generates, to obtain the output bits.

The previous constructions can be easily generalized to the case $d \geq 4$. We simply insert $d-2$ phases before Phase 2 of the base case $d = 2$ for the computation of the group-carry-propagates and group-carry-generates of levels i, $i = d, d-1, d-2, \ldots, 3$. We also insert $d-2$ phases after Phase 4 of the base case for the computation of the section-carry-propagates and section-carry-generates of levels i, $i = 3,4,5,\ldots,d$. We use prefix circuits (Theorem 2.1 with parameter d) for the computation of group-carry-propagates and section-carry-generates. The edge complexity of the entire circuit is dominated by these prefix subcircuits, which collectively have edge complexity $O(n(\log^{\overbrace{*\cdots*}^{d-1}} n)^2)$. The depth of the circuit is still dominated by the computation of the group-carry-propagates and the group-carry-generate for level-1 groups that have the largest size (Phase 3 of the base case), which requires a circuit of depth $\log_k n \pm o(\log_k n)$.

The simple case $d = 1$ can be easily obtained by removing Phases 2 and 4 of the addition circuit for $d = 2$, and modifying Phases 3 and 5 of that circuit. The details are omitted. □

In [10], it was shown that addition can be performed using an AND-OR circuit of depth $4\log_k n + o(\log_k n) + O(1)$, edge complexity $O(n(\log^* n)^2)$, and fan-in k. Our result improves the depth of the circuit by a factor of 4 asymptoti-

cally and further reduces the edge complexity. Moreover, the depth of our addition circuit is optimal within a factor of $1 + o(1)$ from the trivial lower bound $\log_k 2n$ when $\log_k n$ is not a constant.

4 Conclusion

In this paper, we have proposed several circuits to perform prefix computation and addition. Our constructions provide effective tradeoffs among edge complexity, circuit depth, and maximum fan-in through the flexibility provided in the choice of the parameters k (fan-in) and d (levels of hierarchy). Our circuits appear to be considerably more depth-efficient than the best previous circuits, assuming similar edge complexity and fan-in (or, alternatively, considerably more cost-effective for similar circuit depth). Moreover, the depths of all the circuits presented in this paper are optimal within a small constant factor with any fan-in restriction.

Even though asymptotic complexity results, particularly those based on circuit elements with nonconstant fan-in, may appear to have little bearing on the design of practical circuits with currently available technology, there are ample reasons for continued research in this area. A primary reason is to develop a better understanding of computational problems, methods for their solution, and tradeoffs between various cost/performance parameters. Such an understanding paves the way for the application of methods used to solve one class of problems in other, seemingly different, contexts. A second important reason is to establish fundamental limits on the time and hardware resources needed for performing certain computations. Bounds of this nature are useful, not only because they guide our quest for more cost-effective solutions, but also due to their influence in motivating research on alternative computational paradigms to circumvent the assumptions or constraints that led to the bounds. Additionally, problem sizes of practical interest continually expand with rising computational power and growing user expectations, thus narrowing the gap between practical and asymptotic analyses. Twenty years ago, research contemplating million-processor systems or billion-transistor chips seemed far removed from reality. Now, they are merely ambitious development projects.

References

[1] N. Alon and J. Bruck, "Explicit constructions of depth-2 majority circuits for comparison and addition," *SIAM J. Disc. Math.*, pp. 1-8, 1994.

[2] P. Beame, E. Brisson, and R. Ladner, "The complexity of computing symmetric functions using threshold circuits," *Theoretical Comput. Sci.*, pp. 253-265, 1992.

[3] J. Bruck, "Computing with networks of threshold elements," Ph.D. dissertation, Dept. Electrical Engineering, Stanford Univ., 1989.

[4] I. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[5] R.E. Ladner, and M.J. Fischer, "Parallel prefix computation," *J. ACM*, Vol. 27, No. 4, pp. 831-838, Oct. 1980.

[6] I. Parberry, *Circuit Complexity and Neural Networks*, Cambridge, Mass., MIT Press, 1994.

[7] B. Parhami, "Carry-free addition of recoded binary signed-digit numbers," *IEEE Trans. Computers*, Vol. 37, no. 11, Nov. 1988, pp. 1470-1467.

[8] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, 2000.

[9] N. Pippenger, "The complexity of computations by networks," *IBM J. Res. Develop.*, Vol. 31, No. 2, pp. 235-243, Mar. 1987.

[10] K.Y. Siu, V.P. Roychowdhury, and T. Kailath, *Discrete Neural Computation, A Theoretical Foundation*, Prentice Hall, New Jersey, 1995.

[11] K.Y. Siu, V.P. Roychowdhury, and T. Kailath, "Toward massively parallel design of multipliers," *J. Parallel Distributed Computing*, No. 24, pp. 86-93, Jan. 1995.

[12] E.E. Swartzlander, Jr., *Computer Arithmetic*, Vol. I/II, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[13] S. Vassilladis, S. Contofana, and K. Bertels, "2-1 addition and related arithmetic operations with threshold logic," *IEEE Trans. Computers*, Vol. 45, no. 9, pp. 1062-1067, Sep. 1996.

[14] C.-H. Yeh and B. Parhami, "Efficient pipelined multi-operand adders with high throughput and low latency: designs and applications," *Proc. Asilomar Conf. Signals, Systems, and Computers*, pp. 894-898, 1996.

[15] C.-H. Yeh and E. A. Varvarigos, "New efficient majority circuits for the computation of some basic arithmetic functions," *J. Comput. Information*, pp. 114-136, 1996.

[16] C.-H. Yeh and E. A. Varvarigos, "Depth-efficient threshold circuits for multiplication and symmetric function computation," *Proc. Int'l Computing and Combinatorics Conf., LNCS*, pp. 231-240, 1996.

[17] C.-H. Yeh, E. A. Varvarigos, B. Parhami, and H. Lee, "Optimal-depth threshold circuits for multiplication and related problems," *Proc. 33th Asilomar Conf. Signals, Systems, and Computers*, Vol. 2, Oct. 1999, pp. 1331-1335.