Signed Digit Addition and Related Operations with Threshold Logic

Sorin Cotofana, *Senior Member*, *IEEE*, and Stamatis Vassiliadis, *Fellow*, *IEEE*

Abstract—Assuming signed digit number representations, we investigate the implementation of some addition related operations assuming linear threshold networks. We measure the depth and size of the networks in terms of linear threshold gates. We show first that a depth-2 network with $O(n)$ size, weight, and fan-in complexities can perform signed digit symmetric functions. Consequently, assuming radix-2 signed digit representation, we show that the two operand addition can be performed by a threshold network of depth-2 having $O(n)$ size complexity and $O(1)$ weight and fan-in complexities. Furthermore, we show that, assuming radix-$(2n-1)$ signed digit representations, the multioperand addition can be computed by a depth-2 network with $O(n^3)$ size with the weight and fan-in complexities being polynomially bounded. Finally, we show that multiplication can be performed by a linear threshold network of depth-3 with the size of $O(n^3)$ requiring $O(n^3)$ weights and $O(n^2 \log n)$ fan-in.

Index Terms—Computer arithmetic, signed-digit number representation, signed-digit arithmetic, carry-free addition, redundant adders, redundant multipliers, threshold logic, neural networks.

1 Introduction

High performance addition and addition related operations, such as multiplication, play an important role in the computer-based computational paradigm. A major impediment to improving the speed of arithmetic execution units incorporating addition and addition related operations is the presence of carry and borrow chains. One solution for the elimination of carry chains is the use of redundant representation of operands, proposed by Avizienis in [1]. The Signed Digit (SD) number representation method allows, under certain assumptions, the so-called "totally parallel addition" [1], which limits the propagation of the carries at the expense of some overhead in data storage space and in processing time for the conversion of the results and potentially of the operands.

The redundant representation operates as follows: For any radix $r \geq 2$, a sign-digit integer number $X = (x_{n-1}, \ldots, x_1, x_0)_{SD_r}$, represented with n digits, has the algebraic value $X = \sum_{i=0}^{n-1} x_i \times r^i$. Each digit x_i of the X number can assume its value in the digit set $\Sigma_r = \{-\alpha, -\alpha+1, \ldots, -1, 0, 1, \ldots, \alpha-1, \alpha\}$. The cardinality of the set Σ_r is $2\alpha+1$ and the maximum digit magnitude α must satisfies the relations stated in (1).[1]

$$\left\lceil \frac{r-1}{2} \right\rceil \leq \alpha \leq r-1. \tag{1}$$

1. Note that, for a given radix r, it might be that α is not unique, therefore, there can be more than one possible digit set.

• *The authors are with the Electrical Engineering Department, Delft University of Technology, PO Box 5031, 2600 GA Delft, The Netherlands. E-mail: {Sorin, Stamatis}@Plato.ET.TUDelft.nl.*

Manuscript received 18 Nov. 1998; accepted 5 Apr. 1999.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 104899.

In order to have minimum redundancy and, as a consequence, minimum storage overhead, one can assume that $\alpha = \lceil \frac{r}{2} \rceil$, but, in order to break the carry chain, i.e., to have "totally parallel addition," the value of α should satisfy the relations stated in (2).

$$\left\lceil \frac{r+1}{2} \right\rceil \leq \alpha \leq r-1. \tag{2}$$

Based on sign-digit representation, a number of high-speed architectures[2] have been reported, see, for example [2], [3], [4], [5], [6]. Thus far, all the investigations in SD arithmetic architectures assumed logic implementation with technologies that directly implement Boolean gates. Currently, other possibilities exist in VLSI for the implementation of Boolean functions using threshold devices in CMOS technology [7], [8], [9], [10]. In assuming Threshold Logic (TL), the basic processing element can be a Linear Threshold Gate[3] (LTG) computing the Boolean function $F(X)$ such that:

$$F(X) = sgn(\mathcal{F}(X)) = \begin{cases} 1 & \text{if } \mathcal{F}(X) \geq 0 \\ 0 & \text{if } \mathcal{F}(X) < 0 \end{cases}$$
$$\mathcal{F}(X) = \sum_{i=1}^{n} \omega_i x_i - \psi, \tag{3}$$

where the set of input variables and weights are defined by $X = (x_1, x_2, \ldots, x_{n-1}, x_n)$ and by $\Omega = (\omega_1, \omega_2, \ldots, \omega_{n-1}, \omega_n)$, respectively. Such an LTG contains a threshold value, ψ, a summation device, Σ, computing $\mathcal{F}(X)$, and a threshold element, T, computing $F(X) = sgn(\mathcal{F}(X))$.

Given that TL may be promising, it is of interest to investigate new schemes applicable to such a new technology. To this end, assuming binary nonredundant

2. Serial, on-line, and parallel.

3. Such a threshold gate corresponds to the Boolean output neuron introduced in the McCulloch-Pitts neural model [11], [12] with no learning features.

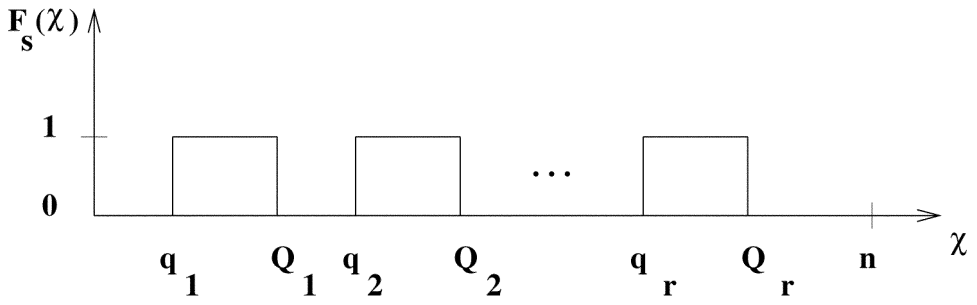

Fig. 1. Interval-based representation of F_s.

representations, a number of recent proposals regarding addition and multiplications, see, for example, [13], [14], [15], [16], [17], [18], [19], [20], have been developed that assume threshold, rather than Boolean, logic.

Thus far, there are no studies assuming redundant representations and TL. In this paper, we assume SD number representation and we investigate linear threshold networks for $2-1$ addition, multioperand addition, and multiplication. We assume that the operands are n-SD numbers and we are mainly concerned with establishing the limits of the circuit designs using threshold-based networks. We measure the depth and the size of the networks we propose in terms of LTGs.

The main contributions of our proposal can be summarized as:

- Any SD symmetric function can be implemented by a depth-2 feed-forward Linear Threshold Network (LTN) with $O(n)$ size, weight and fan-in values.
- Assuming radix-2 redundant operand representation, the addition of two n-SD numbers can be computed by a depth-2 LTN with $O(n)$ size and $O(1)$ weight and fan-in values.
- Assuming radix-$(2n-1)$ redundant operand representation, the multioperand addition of n n-SD numbers can be computed by an explicit depth-2 LTN with the size in the order of $O(n^3)$, with the maximum weight value in the order of $O(n^3)$, and the maximum fan-in value in the order of $O(n^2)$.
- Assuming radix-$(2n-1)$ operand representation, the multiplication of two n-SD numbers can be computed by an explicit depth-3 LTN with the size in the order of $O(n^3)$. The maximum weight value is in the order of $O(n^3)$ and the maximum fan-in value is in the order of $O(n^2 \log n)$.

We also note here that, while our results are primarily theoretical, there exist technology proposals, see, for example, [10], which may implement at least some of the proposed schemes, e.g., two operand addition.

The presentation is organized as follows: In Section 2, we discuss background information on Boolean symmetric functions and their implementation with TL and introduce some preliminary results; in Section 3, we present TL schemes for the $2-1$ addition of radix-2 SD numbers; in Section 4, we study the multiplication of radix-2 SD numbers and we present schemes for the multioperand addition and the multiplication of radix-$(2n-1)$ SD numbers; we conclude the presentation with some final remarks.

2 Background and Preliminaries

In order to make this presentation self-consistent, we introduce in this section the definition of Boolean symmetric functions and some TL-based implementation techniques that we will use in our investigation.

Definition 1. *A Boolean function of n variables F_s is symmetric if and only if for any permutation σ of $<1,2,\ldots,n>$, $F_s(x_1,x_2,\ldots,x_n) = F_s(x_{\sigma(1)},x_{\sigma(2)},\ldots,x_{\sigma(n)})$.*

For any n input variable symmetric Boolean function F_s, the sum $\chi = \sum_{i=1}^{n} x_i$ ranges from 0 (all input variables are 0) to n (all input variables are 1). Inside this definition domain $[0,n]$, there are r intervals $[q_j,Q_j], j=1,2,\ldots,r$, for which if $\chi \in [q_j,Q_j]$, then F_s is equal to 1 and, outside these intervals, the function is 0. This is graphically depicted in Fig. 1 and formally described by (4).

$$F_s(x_1,x_2,\ldots,x_n) = \begin{cases} 1 & \text{if } \chi = \sum_{i=1}^{n} x_i \in [q_j,Q_j], \quad j=1,2,\ldots,r \\ & 0 \le q_1 \le Q_1 < q_2 \le Q_2 < \ldots < q_r \le Q_r \le n \\ 0 & \text{elsewhere.} \end{cases} \tag{4}$$

The number of intervals depends on the function definition and we proved elsewhere [21] that, for any Boolean symmetric function, the maximum number of intervals r is upper bounded by $\lceil \frac{n+1}{2} \rceil$.

Definition 2. *A Boolean function of n variables F_{gs} is generalized symmetric[4] if it entirely depends on $\chi = \sum_{i=1}^{n} w_i x_i$, the weighted sum of its input variables, with w_i, $i=1,2,\ldots,n$, given integer constants.*[5]

In essence, a generalized symmetric Boolean function F_{gs} is either a symmetric Boolean function or a nonsymmetric Boolean function that can be transformed into a symmetric Boolean function by trivial transformations, e.g., assignment of different weight values to the inputs or input

4. This definition and, also, Definition 1 are not specific to functions with Boolean input variables. The symmetry is an intrinsic property of the function and do not depend on the input variable type. Therefore, they also appy to functions of other types of input variables, e.g., integer, real.

5. The weights w_i can be also real numbers, but we have assumed integer values here because of practical considerations related to the LTG fabrication technology [7], [10].

Decimal Sum	Binary Sum			
	s3	s2	s1	s0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	
6	0	1	1	
7	0	1	1	
8	1	0	0	
9	1	0	0	
10	1	0	1	
11	1	0	1	
12	1	1	0	

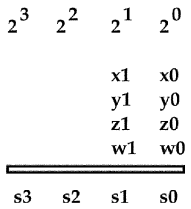

Fig. 2. Four 2-bit multioperand addition.

replication. F_{gs} can be described as a function of $\chi = \sum_{i=1}^{n} w_i x_i$ and the definition domain extends from $[0, n]$ to $[0, \chi_{max}]$, where $\chi_{max} = \sum_{i=1}^{n} w_i$. All the results that stand true for symmetric Boolean functions can be also applied to generalized symmetric Boolean functions.

To clarify the generalized symmetric Boolean function concept, let us consider the 4 2-bit multioperand addition producing a 4-bit result. The truth table and the schematic diagram for such a function are depicted in Fig. 2. First, it can be observed that, in order to produce the sum at bit position 0, we need to consider only the bits in the first column (LSB position). It can be easily verified that the Boolean function computing the sum's LSB, $s_0(x_0, y_0, z_0, w_0)$ is symmetric because it can be clearly determined by the integer value of $\chi = x_0 + y_0 + z_0 + w_0$, i.e., if $\chi = 0$, then $s_0 = 0$, if $\chi = 1$, then $s_0 = 1$, if $\chi = 2$, then $s_0 = 0$, if $\chi = 3$, then $s_0 = 1$, and if $\chi = 4$, then $s_0 = 0$. This property, however, does not hold for the other sum bits. For example, the Boolean function $s_1(x_0, y_0, z_0, w_0, x_1, y_1, z_1, w_1)$ is not a symmetric Boolean function as its value depends on the positioning of the inputs and cannot always be correctly determined from the $x_0 + y_0 + z_0 + w_0 + x_1 + y_1 + z_1 + w_1$ value.

The s_1 function is, however, a generalized symmetric Boolean function as it can be made to be a symmetric Boolean function if a weight of 2 is associated with the input bits in the column 1. Consequently, the s_1 sum bit can be computed by a symmetric Boolean function $s_1(\chi)$, where $\chi = x_0 + y_0 + z_0 + w_0 + 2(x_1 + y_1 + z_1 + w_1)$, which interval-based representation is graphically depicted in Fig. 3.

Given that symmetric (generalized or not) functions constitute a frequently used class of Boolean functions and because they are expensive to implement in hardware, in terms of area and delay, their implementation with feed-forward LTNs has been the subject of numerous theoretical and practical scientific investigations, see, for example, [22], [23], [24], [25], [16], [21].

The most network-size efficient approach known so far for the depth-2 implementation of symmetric Boolean function with TL is the telescopic sum method, introduced by Minick in [23]. The method can be used for the

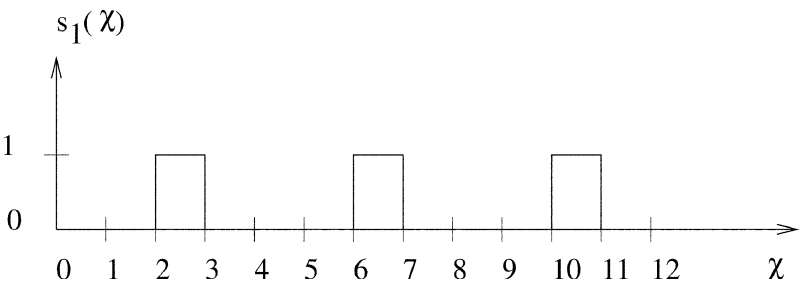

Fig. 3. Interval-based representation of s_1.

implementation of any Boolean symmetric function and produces depth-2 feed-forward LTNs with the size in the order of $O(n)$, measured in terms of LTGs, and with linear weight and fan-in values. We shortly describe this method by introducing the following lemma.

Lemma 1 [23]. *Any Boolean symmetric function $F_s(x_1, x_2, \ldots, x_n)$, described as in (4), can be implemented by a two-layer feed-forward LTN with a size complexity measured in terms of LTGs in the order of $O(n)$ as follows:*

$$F_s = sgn\left[\sum_{i=1}^{n} x_i - \left[t_0 + \sum_{j=1}^{r} t_j u_j\right]\right], \tag{5}$$

where

$$t_0 = q_1 t_j = q_{j+1} - q_j \qquad (j = 1, 2, \ldots, r-1),$$
$$u_j = sgn\left[\sum_{i=1}^{n} x_i - (Q_j + 1)\right]$$
$$t_r = n + 1 - q_r \; if \; Q_r \neq n \; and \; t_r = 0 \; if \; Q_r = n.$$

A formal proof of Lemma 1 and implementation examples can be found in [26].

Given that we assume SD operands (that is, we consider functions with no Boolean input variables), we need to map them into general Boolean functions. In order to achieve this mapping, we first have to choose a representation for the SDs. One possible representation is the 2's complement [27].[6]

Given a fixed radix r, an SD number is represented as $(s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$. In this presentation, we will consider that any digit s_i can assume a value in the symmetric[7] digit set $\{-\alpha, -\alpha+1, \ldots, 1, 0, 1, \ldots, \alpha-1, \alpha\}$, with the maximum digit magnitude α satisfying (1) or (2). The cardinality of the digit set is $2\alpha + 1$ and, consequently, any SD s_i can be binary represented by a k-tuple $(x_{k-1}, \ldots, x_1, x_0)$ with $k = \lceil \log(2\alpha + 1) \rceil$ and $x_l \in \{0, 1\}$, for $l = 0, 1, \ldots, k-1$.

For the particular case of the 2's complement codification of the SDs, the dimension of the k-tuple can also be computed as $k = 1 + \lceil \log(\alpha + 1) \rceil$. For each s_i, $i = 0, 1, \ldots, n-1$, the values of x_l, $l = 0, 1, \ldots, k-1$, are to be computed such as $s_i = -2^{k-1} x_{k-1} + \sum_{l=0}^{k-2} 2^l x_l$. Assuming 2's complement representation codification of the SDs, we will prove (in the following lemma) that any generalized symmetric SD function can be implemented by a depth-2 LTN with polynomially bounded size.

Lemma 2. *Let $\mathcal{F}(s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$ be an arbitrary generalized symmetric function of n SD variables, with $s_i \in \{-\alpha, -\alpha+1, \ldots, -1, 0, 1, \ldots, \alpha-1, \alpha\}$ and α satisfying (1) or (2) for a fixed radix r. $\mathcal{F}$ can be implemented by an LTN with the cost in the order of $O(n)$.*

Proof. Given that $\mathcal{F}$ is generalized symmetric, it can be expressed as in (6), where w_i, $i = 0, 1, \ldots, n-1$, are arbitrary integer constant weights.

$$\mathcal{F}(s_{n-1}, s_{n-2}, \ldots, s_1, s_0) = \mathcal{F}\left(\sum_{i=0}^{n-1} w_i s_i\right). \tag{6}$$

Under 2's complement representation of the SDs s_i, (6) is equivalent to:

$$\mathcal{F}(s_{n-1}, \ldots, s_1, s_0) = \mathcal{F}\left(\sum_{i=0}^{n-1} w_i \left(-2^{k-1} x_{k-1} + \sum_{l=0}^{k-2} 2^l x_l\right)\right)$$
$$= \mathcal{F}\left(\sum_{i=0}^{n-1} w_i \left(-2^{\lceil \log(\alpha+1) \rceil} x_{\lceil \log(\alpha+1) \rceil} + \sum_{l=0}^{\lceil \log(\alpha+1) \rceil - 1} 2^l x_l\right)\right). \tag{7}$$

As a consequence of (7), $\mathcal{F}$ is expressed as a generalized Boolean symmetric function of $n(1 + \lceil \log(\alpha + 1) \rceil)$ variables, then it can be computed with the scheme in Lemma 1. The size of the LTN implementing $\mathcal{F}$ depends, on the number of intervals on the definition domain. Given that, in our case, the maximum absolute value any digit can assume is $\alpha \leq r - 1$, the argument of $\mathcal{F}$ as described in (7), in the worst case scenario, can take any value inside the definition domain $[-\sum_{i=0}^{n-1} w_i r, \sum_{i=0}^{n-1} w_i r]$. Consequently, the maximum number of intervals is upper bounded by

$$\left\lceil \frac{2r \sum_{i=0}^{n-1} w_i + 1}{2} \right\rceil.$$

Because we assumed that the weights w_i and the radix r are arbitrary integer constants, the LTN cost is in the order of $O(n)$. Obviously the weight and fan-in values are in the order of $O(n)$. □

3 Signed Digit $2-1$ Addition

In this section, we investigate $2-1$ addition schemes using a "totally parallel" [1] addition approach. We use a fixed radix of 2 and the corresponding digit set $\{\overline{1}, 0, 1\}$, where $\overline{1}$ denotes -1. We consider two n-SD integers $X = (x_{n-1}, \ldots, x_1, x_0)_{SD_2}$ and $Y = (y_{n-1}, \ldots, y_1, y_0)_{SD_2}$ and propose two schemes to compute the sum $Z = X + Y$, represented as $Z = (z_{n-1}, \ldots, z_1, z_0)_{SD_2}$.

Traditionally, in the context of Boolean logic, the $2-1$ addition of radix-2 SD represented operands has been achieved with two-step approaches [2], [27], [3]: First, an intermediate carry c_i and an intermediate sum s_i satisfying the equation $x_i + y_i = 2c_i + s_i$ are computed for each digit position i. Second, the sum digit z_i, $i = 0, 1, \ldots, n-1$, is computed as $s_i + c_{i-1}$.

In our approach, we will use the "totally parallel" addition described in Table 1 [3]. We also assume that any digit x in the set $\{\overline{1}, 0, 1\}$ is represented in the 2's complement notation by two bits, as is shown in Table 2. Note that, in this codification, the combination $x^+ = 0$ and $x^- = 1$ is not allowed and cannot appear during the computations.

6. There are also other possibilities, but the 2's complement notation seems to be the natural choice. Later on we will suggest that, in some particular cases, other codification schemes are more convenient as they lead to the reduction of the network depth.

7. The symmetry of the digit set is not a restriction. We make this assumption for simplicity of notations. Digit sets which are not symmetric can also be considered without changing the results we report in the next sections.

TABLE 1
Totally Parallel Addition at Digit Position i

x_i	y_i	x_{i-1}	y_{i-1}	c_i	s_i	z_i
0	0	x	x	0	0	c_{i-1}
1	1	x	x	1	0	
$\overline{1}$	$\overline{1}$	x	x	$\overline{1}$	0	
$\overline{1}$	1	x	x	0	0	
1	$\overline{1}$	x	x	0	0	
0 (1)	1 (0)	Both Nonnegative		1	$\overline{1}$	$c_{i-1}+\overline{1}$
0 (1)	1 (0)	Otherwise		0	1	$c_{i-1}+1$
0 ($\overline{1}$)	$\overline{1}$ (0)	Both Nonnegative		0	$\overline{1}$	$c_{i-1}+\overline{1}$
0 ($\overline{1}$)	$\overline{1}$ (0)	Otherwise		$\overline{1}$	1	$c_{i-1}+1$

It can be observed in Table 1 that the digits in position $i-1$ contribute into the computation of s_i and c_i only by their sign. Therefore, what we have to compute in order to implement the scheme presented in the table are the functions $s_i(x_i, y_i, x_{i-1}^-, y_{i-1}^-)$ and $c_i(x_i, y_i, x_{i-1}^-, y_{i-1}^-)$. These two functions, as is directly implied from the table, are not symmetric in their input variables. They can be made symmetric by computing the weighted sum of the inputs χ_s stated by (8) such that (9), (10) with proper determined weights w_i and w_{i-1} hold true for all the possible input combinations.

$$\chi_s = w_i(-2x_i^- + x_i^+ - 2y_i^- + y_i^+) + w_{i-1}(x_{i-1}^- + y_{i-1}^-) \quad (8)$$

$$s_i(\chi_s) = s_i(x_i, y_i, x_{i-1}^-, y_{i-1}^-) \quad (9)$$

$$c_i(\chi_s) = c_i(x_i, y_i, x_{i-1}^-, y_{i-1}^-). \quad (10)$$

We compute the weights w_i and w_{i-1} by taking into consideration the specific structure of the functions s_i and c_i. The choice for $w_{i-1}=1$ is straightforward. Given that, for the digits in position $i-1$, we take into account only the x^- bits, the minimum value of w_i should be equal[8] to 3. Consequently, the weighted sum χ_s in (8) can be computed as $-6(x_i^- + y_i^-) + 3(x_i^+ + y_i^+) + x_{i-1}^- + y_{i-1}^-$ and the description of the symmetric functions computing s_i and c_i is described in Table 3.

From the table, we derive the interval description (similar to the description of (4)) for the required Boolean functions:

$$s_i^+ = 1 \quad \text{if} \quad \chi_s \in \{[-3,-1],[3,5]\} \quad (11)$$

$$s_i^- = 1 \quad \text{if} \quad \chi_s \in \{[-3],[3]\} \quad (12)$$

$$c_i^+ = 1 \quad \text{if} \quad \chi_s \in \{[-6,-4],[-2,-1],[3],[6,8]\} \quad (13)$$

$$c_i^- = 1 \quad \text{if} \quad \chi_s \in \{[-6,-4],[-2,-1]\}. \quad (14)$$

Assume that $[\alpha]_i^{\stackrel{+}{=}}$ and $[\alpha]_i^{\stackrel{-}{=}}$ are computed as in (15), (16).

$$[\alpha]_i^{\stackrel{+}{=}} = \operatorname{sgn}\{-6(x_i^- + y_i^-) + 3(x_i^+ + y_i^+) + x_{i-1}^- + y_{i-1}^- - \alpha\} \quad (15)$$

$$[\alpha]_i^{\stackrel{-}{=}} = \operatorname{sgn}\{\alpha + 6(x_i^- + y_i^-) - 3(x_i^+ + y_i^+) - x_{i-1}^- - y_{i-1}^-\}. \quad (16)$$

We next introduce an implicit depth-1 implementation technique based on the fact that any symmetric Boolean function F_s, defined as in (4), can be expressed as:

$$F_s(x_1, x_2, \ldots, x_n) = q_1^{\stackrel{+}{=}} Q_1^{\stackrel{-}{=}} + q_2^{\stackrel{+}{=}} Q_2^{\stackrel{-}{=}} + \ldots + q_r^{\stackrel{+}{=}} Q_r^{\stackrel{-}{=}}, \quad (17)$$

where $q_j^{\stackrel{+}{=}} = 1$ if $\chi \geq q_j$, $Q_j^{\stackrel{-}{=}} = 1$ if $\chi \leq Q_j$, for $j = 1, 2, \ldots, r$, and $+$ and concatenation represent logical OR and AND, respectively.

Lemma 3. *Any Boolean symmetric function $F_s(x_1, x_2, \ldots, x_n)$, described in (17), can be implemented by an implicit depth-1 feed-forward LTN with the size in the order of $O(n)$ as follows:*

TABLE 2
Digit Codification of $x \in \{\overline{1}, 0, 1\}$

x	x^-	x^+	$-2^1x^- + 2^0x^+$
0	0	0	0
1	0	1	1
$\overline{1}$	1	1	-1

8. w_i has to be greater than the maximum value that can be assumed by $w_{i-1}(x_{i-1}^- + y_{i-1}^-)$ which, in this case, is 2.

$$F_s(x_1, x_2, \ldots, x_n) = q_1^{\overline{+}} + Q_1^{\overline{-}} + q_2^{\overline{+}} + Q_2^{\overline{-}} + \ldots + q_r^{\overline{+}} + Q_r^{\overline{-}} - r. \quad (18)$$

Proof. To verify (18), it will be shown that F_s is indeed 1 when the sum $\chi = \sum_{i=1}^{n} x_i$ lies inside an interval $[q_j, Q_j]$ for a specific j and that F_s is 0 when there is no j such that $\chi \in [q_j, Q_j]$ for all j, $1 \leq j \leq r$.

- Case 1: $\chi \in [q_j, Q_j]$ for a specific j, $1 \leq j \leq r$.
  In this case, $Q_l^{\overline{-}} = 1$ for $l = j, j+1, \ldots, r$, $Q_l^{\overline{-}} = 0$ for $l = 1, 2, \ldots, j-1$, $q_l^{\overline{+}} = 1$ for $l = 1, 2, \ldots, j$, and $q_l^{\overline{+}} = 0$ for $l = j+1, \ldots, r$. Therefore, $F_s = r - j + 1 + j - r$, i.e., is 1 as needed.
- Case 2: There is no j, $1 \leq j \leq r$, such that $\chi \in [q_j, Q_j]$.
  In this case, there are three possibilities: $\chi \in (Q_l, q_{l+1})$ for a given l, $1 \leq l \leq r$, $\chi \in [0, q_1)$, and $\chi \in (Q_r, n]$. We will prove that, in all of them, F_s is 0 as needed. In the first subcase, $Q_l^{\overline{-}} = 1$ for $l = j+1, j+2, \ldots, r$, $Q_l^{\overline{-}} = 0$ for $l = 1, 2, \ldots, j$, $q_l^{\overline{+}} = 1$ for $l = 1, 2, \ldots, j$, and $q_l^{\overline{+}} = 0$ for $l = j+1, \ldots, r$. Therefore,

  $$F_s = r - j - 1 + 1 + j - r,$$

  i.e., is 0. In the second subcase, $Q_l^{\overline{-}} = 1$ for $l = 1, 2, \ldots, r$ and $q_l^{\overline{+}} = 0$ for $l = 1, 2, \ldots, r$. Consequently, $F_s = r - r$, i.e., is 0. In the last subcase, $Q_l^{\overline{-}} = 0$ for $l = 1, 2, \ldots, r$ and $q_l^{\overline{+}} = 1$ for $l = 1, 2, \ldots, r$. Consequently, $F_s = r - r$, i.e., is 0.

Given that any $q_j^{\overline{+}}$ can be obtained with an LTG computing $sgn\{\chi - q_j\}$ and any $Q_j^{\overline{-}}$ with an LTG computing $sgn\{Q_j - \chi\}$, the entire network is built with $2r$ LTGs, i.e., the implementation cost is in the order of $O(n)$. All the input weights are 1 and the fan-in for all the gates is n. □

The method presented in Lemma 3 can also be applied for the implementation of generalized symmetric functions. Given that, in this case, the number of intervals is upper bounded by

$$\left\lceil \frac{\sum_{i=1}^{n} w_i + 1}{2} \right\rceil,$$

the implementation cost will be upper bounded by

$$2\left\lceil \frac{\sum_{i=1}^{n} w_i + 1}{2} \right\rceil,$$

i.e., is still in the order of $O(n)$.

Remark 1. *The scheme in Lemma 3 can be changed into an explicit one by connecting all the outputs of the gates computing $q_j^{\overline{+}}$ and $Q_j^{\overline{-}}$ to a gate with the threshold value of $r + 1$. The output of this extra gate will explicitly provide the value of F_s after the delay of 2 TGs.*

Remark 2. *If $q_1 = 0$, then $q_1^{\overline{+}}$ is always 1 and (18) becomes:*

$$F_s(x_1, x_2, \ldots, x_n) = Q_1^{\overline{-}} + q_2^{\overline{+}} + Q_2^{\overline{-}} + \ldots + q_r^{\overline{+}} + Q_r^{\overline{-}} - r + 1. \quad (19)$$

If $Q_r = n$, then $Q_r^{\overline{-}}$ is always 1 and (18) becomes:

$$F_s(x_1, x_2, \ldots, x_n) = q_1^{\overline{+}} + Q_1^{\overline{-}} + q_2^{\overline{+}} + Q_2^{\overline{-}} + \ldots + q_r^{\overline{+}} - r + 1. \quad (20)$$

If $q_1 = 0$ and $Q_r = n$, then $q_j^{\overline{+}}$ and $Q_j^{\overline{-}}$ are always 1 and (18) becomes:

$$F_s(x_1, x_2, \ldots, x_n) = Q_1^{\overline{-}} + q_2^{\overline{+}} + Q_2^{\overline{-}} + \ldots + q_r^{\overline{+}} - r + 2. \quad (21)$$

It should be noted that, if used in cascaded computation, the method described in Lemma 3 increases the fan-in of the next stage because the value of the function F_s is carried by $2r$ signals.

From Table 3 and using (15, (16), (17), the four Boolean symmetric functions describing the computations of the intermediate sum s_i and carry c_i can be expressed by the following:

$$s_i^{+} = [-3]_i^{\overline{+}}[-1]_i^{\overline{-}} + [3]_i^{\overline{+}}[5]_i^{\overline{-}} \quad (22)$$

$$s_i^{-} = [-3]_i^{\overline{+}}[-3]_i^{\overline{-}} + [3]_i^{\overline{+}}[3]_i^{\overline{-}} \quad (23)$$

$$c_i^{+} = [-6]_i^{\overline{+}}[-4]_i^{\overline{-}} + [-2]_i^{\overline{+}}[-1]_i^{\overline{-}} + [3]_i^{\overline{+}}[3]_i^{\overline{-}} + [6]_i^{\overline{+}}[8]_i^{\overline{-}} \quad (24)$$

$$c_i^{-} = [-6]_i^{\overline{+}}[-4]_i^{\overline{-}} + [-2]_i^{\overline{+}}[-1]_i^{\overline{-}}. \quad (25)$$

By applying Lemma 3, we derive from (22), (23), (24), (25) an implicit depth-1 implementation of the first step of the "totally parallel" addition scheme. Because $[-6]_i^{\overline{+}}$ and $[8]_i^{\overline{-}}$ are always 1 and Remark 2, we have that:

$$s_i^{+} = [-3]_i^{\overline{+}} + [-1]_i^{\overline{-}} + [3]_i^{\overline{+}} + [5]_i^{\overline{-}} - 2 \quad (26)$$

$$s_i^{-} = [-3]_i^{\overline{+}} + [-3]_i^{\overline{-}} + [3]_i^{\overline{+}} + [3]_i^{\overline{-}} - 2 \quad (27)$$

$$c_i^{+} = [-4]_i^{\overline{-}} + [-2]_i^{\overline{+}} + [-1]_i^{\overline{-}} + [3]_i^{\overline{+}} + [3]_i^{\overline{-}} + [6]_i^{\overline{+}} - 2 \quad (28)$$

$$c_i^{-} = [-4]_i^{\overline{-}} + [-2]_i^{\overline{+}} + [-1]_i^{\overline{-}} - 1. \quad (29)$$

In order to make the way this implicit scheme is working more intuitive, we depict in Fig. 4 the regions in which the threshold signals $[\alpha]_i^{\overline{+}}$ and $[\alpha]_i^{\overline{-}}$ are active for each of the four signals $s_i^{+}, s_i^{-}, c_i^{+}, c_i^{-}$.

The second step of the "totally parallel" addition is the computation of $z_i = s_i + c_{i-1}$. Following the reasoning used for the computation of $s_i^{+}, s_i^{-}, c_i^{+}, c_i^{-}$:

$$z_i^{+} = \widehat{[-1]_i^{\overline{-}}} + \widehat{[1]_i^{\overline{+}}} - 1 \quad (30)$$

$$z_i^{-} = \widehat{[-1]_i^{\overline{-}}}, \quad (31)$$

where

$$\chi_z = -2(s_i^{-} + c_{i-1}^{-}) + s_i^{+} + c_{i-1}^{+} \quad (32)$$

$$\widehat{[-1]_i^{\overline{-}}} = \mathrm{sgn}\{-1 - \chi_z\} \quad (33)$$

$$\widehat{[1]_i^{\overline{+}}} = \mathrm{sgn}\{\chi_z - 1\}. \quad (34)$$

TABLE 3
c_i and s_i as Symmetric Functions of x_i, y_i, x_{i-1}^-, and y_{i-1}^-

χ_s	x_i	y_i	x_{i-1}^-	y_{i-1}^-	c_i	s_i
0,1,2	0	0	x	x	0	0
6,7,8	1	1	x	x	1	0
-6,-5,-4	$\overline{1}$	$\overline{1}$	x	x	$\overline{1}$	0
0,1,2	$\overline{1}$ (1)	1 ($\overline{1}$)	x	x	0	0
3	0 (1)	1 (0)	0	0	1	$\overline{1}$
4,5	0 (1)	1 (0)	Otherwise		0	1
-3	0 ($\overline{1}$)	$\overline{1}$ (0)	0	0	0	$\overline{1}$
-2,-1	0 ($\overline{1}$)	$\overline{1}$ (0)	Otherwise		$\overline{1}$	1

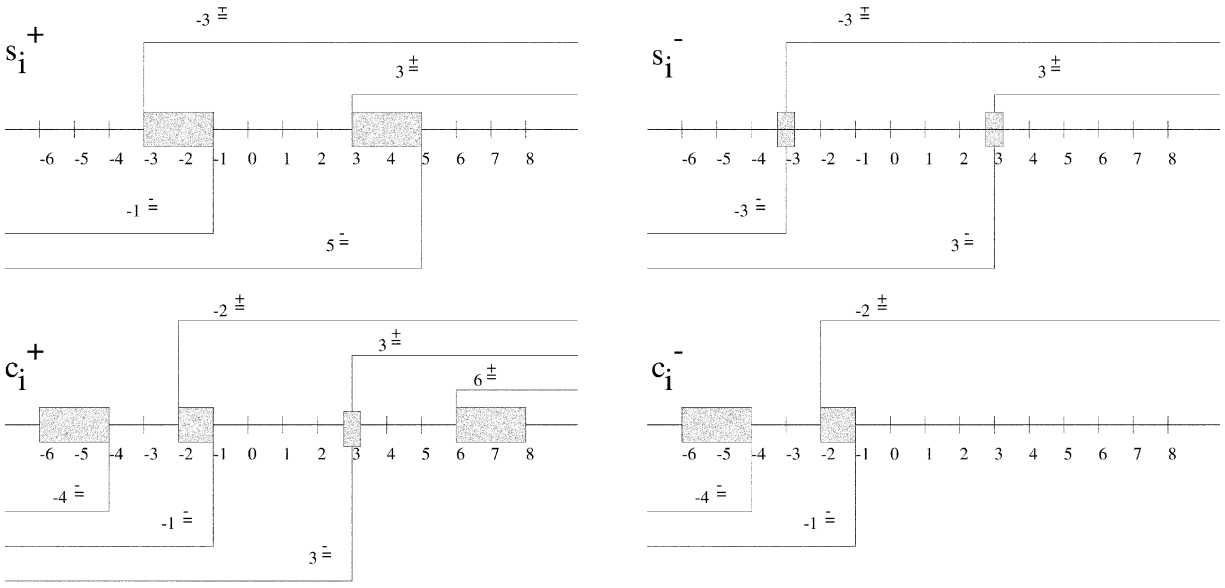

Fig. 4. Description of threshold signals for $s_i^+, s_i^-, c_i^+, c_i^-$.

Theorem 1. *Assuming radix-2 SD operand representation and the SD codification in Table 2, the addition of two n-SD numbers can be computed by an implicit depth-2 LTN with* $11n + 2$ *LTGs, a maximum weight value of* 6, *and a maximum fan-in of* 12.

Proof. The quantities $\widehat{[-1]_i^=}$ and $\widehat{[1]_i^=}$ in (33), (34) can be computed by doing the proper substitutions, using (26), (27), (28), (29), as:

$$\widehat{[-1]_i^=} = \operatorname{sgn}\{-1-\chi_z\} = \operatorname{sgn}\{2(s_i^- + c_{i-1}^-) - s_i^+ - c_{i-1}^+ - 1\}$$
$$= \operatorname{sgn}\Big\{[-3]_i^+ + 2[-3]_i^= + [3]_i^+ + 2[3]_i^= - [-1]_i^= - [5]_i^=$$
$$+[-4]_{i-1}^= + [-2]_{i-1}^+ + [-1]_{i-1}^= - [3]_{i-1}^+ - [3]_{i-1}^= - [6]_{i-1}^+ - 3\Big\} \tag{35}$$

$$\widehat{[1]_i^=} = \operatorname{sgn}\{\chi_z - 1\} = \operatorname{sgn}\{-2(s_i^- + c_{i-1}^-) + s_i^+ + c_{i-1}^+ - 1\}$$
$$= \operatorname{sgn}\Big\{-[-3]_i^+ - 2[-3]_i^= - [3]_i^+ - 2[3]_i^= + [-1]_i^= + [5]_i^=$$
$$-[-4]_{i-1}^= - [-2]_{i-1}^+ - [-1]_{i-1}^= + [3]_{i-1}^+ + [3]_{i-1}^= + [6]_{i-1}^+ + 1\Big\}. \tag{36}$$

Consequently, (30), (31) provide an implicit depth-2 implementation scheme for the computation of the sum digit z_i. On the first level of the network, we compute, for each digit position i, $i = 0, 1, \ldots, n-1$, the values $[-4]_i^=$, $[-3]_i^=$, $[-3]_i^+$, $[-2]_i^+$, $[-1]_i^=$, $[3]_i^=$, $[3]_i^+$, $[5]_i^=$, and $[6]_i^+$, i.e., we use nine TGs per digit. On the second level, we need two TGs for each digit position i, $i = 0, 1, \ldots, n-1$, in order to compute $\widehat{[-1]_i^=}, \widehat{[1]_i^=}$ as stated by (35), (36). Therefore, the network producing all the sum digits can be

TABLE 4
New Digit Codification for $x \in \{\overline{1}, 0, 1\}$

x	x^-	x^+	$-2^1x^- + 2^0x^+$
0	0	0	0
1	0	1	1
$\overline{1}$	1	0	-2

constructed with $11n$ TGs. For the digit position $n-1$, we have to produce the carry-out. This can be explicitly generated in depth-2 at the expanse of two TGs computing:

$$c_{n-1}^{+} = \operatorname{sgn}\Big\{[-4]_{\overline{1}}^{-}{}_{n-1} + [-2]_{\overline{1}}^{+}{}_{n-1} + [-1]_{\overline{1}}^{-}{}_{n-1} + [3]_{\overline{1}}^{+}{}_{n-1} + [3]_{\overline{1}}^{-}{}_{n-1} + [6]_{\overline{1}}^{+}{}_{n-1} - 2\Big\} \tag{37}$$

$$c_{n-1}^{-} = \operatorname{sgn}\Big\{[-4]_{\overline{1}}^{-}{}_{n-1} + [-2]_{\overline{1}}^{+}{}_{n-1} + [-1]_{\overline{1}}^{-}{}_{n-1} - 1\Big\}. \tag{38}$$

Therefore, the cost of the entire addition network is $11n+2$, i.e., of $O(n)$ complexity. Obviously, the weight values and fan-in values do not depend on n. The maximum fan-in is 12 and the maximum weight value is 6, i.e., having $O(1)$ complexity. □

Note that, for this scheme, the value of z_i^+ is carried by two signals and one threshold value and z_i^- is actually depth-2 explicitly computed. If used in cascaded computation, this method will increase with 1 the fan-in of the next stage and will contribute with 1 to the threshold value of some of the gates in the next stage.

If we compare the scheme introduced in Theorem 1 with the depth-2 scheme presented in [28], which has a network size of $25n+5$, a maximum fan-in of 26, and a maximum weight value of 123, one can observe that we achieved a substantial reduction in network size, weight, and fan-in values for the same network depth. However, the new depth-2 scheme is implicit and this fact increases the fan-in of the stage requiring as inputs the digits z_i. In the remainder of this section, we show that it is possible to explicitly compute the sum while maintaining the network depth and complexity.

The method described by (30), (31) is implicit because of the way we compute the final sum bit z_i^+. All the other signals, i.e., z_i^-, c_{n-1}^+, and c_{n-1}^- are explicitly computed with two levels of TGs. Consequently, (30) has to be modified to appear as $z_i^- = [\widehat{\gamma}]_{\overline{1}}^{-}$ or $z_i^- = [\widehat{\gamma}]_{\overline{1}}^{+}$ without inducing fundamental changes to (31), (37), (38). To this end, we assume that, in order to represent a SD x in the set $\{\overline{1}, 0, 1\}$, we use the codification described in Table 4 instead of the 2's complement codification in Table 2. Note that, with this new codification, the combination $x^+ = 1$ and $x^- = 1$ is not allowed and cannot appear during the computations.

Under this assumption, the quantity χ_s can be expressed as in (39) and it can take values in the definition interval $[-12, 8]$.

$$\begin{aligned}\chi_s &= 3(x_i + y_i) + x_{i-1}^- + y_{i-1}^- \\ &= 3(-2^1x_i^- + 2^0x_i^+ - 2^1y_i^- + 2^0y_i^+) + x_{i-1}^- + y_{i-1}^- \\ &= -6(x_i^- + y_i^-) + 3(x_i^+ + y_i^+) + x_{i-1}^- + y_{i-1}^-.\end{aligned} \tag{39}$$

Thus, the first step of the "totally parallel" addition scheme is described in Table 5. From the table, it can be deduced that the Boolean symmetric functions describing the computations of the intermediate sum s_i and carry c_i are as follows:

$$s_i^+ = [-5]_{\overline{1}}^{+}[-4]_{\overline{1}}^{-} + [4]_{\overline{1}}^{+}[5]_{\overline{1}}^{-} \tag{40}$$

$$s_i^- = [-6]_{\overline{1}}^{+}[-6]_{\overline{1}}^{-} + [3]_{\overline{1}}^{+}[3]_{\overline{1}}^{-} \tag{41}$$

$$c_i^+ = [3]_{\overline{1}}^{+}[3]_{\overline{1}}^{-} + [6]_{\overline{1}}^{+}[8]_{\overline{1}}^{-} \tag{42}$$

TABLE 5
c_i and s_i as Functions of χ_s

χ_s	x_i	y_i	x_{i-1}^-	y_{i-1}^-	c_i	s_i
0,1,2	0	0	x	x	0	0
6,7,8	1	1	x	x	1	0
-10,-11,-12	$\overline{1}$	$\overline{1}$	x	x	$\overline{1}$	0
-3,-2,-1	$\overline{1}$ (1)	1 ($\overline{1}$)	x	x	0	0
3	0 (1)	1 (0)	0	0	1	$\overline{1}$
4,5	0 (1)	1 (0)	Otherwise		0	1
-6	0 ($\overline{1}$)	$\overline{1}$ (0)	0	0	0	$\overline{1}$
-5,-4	0 ($\overline{1}$)	$\overline{1}$ (0)	Otherwise		$\overline{1}$	1

TABLE 6
z_i as Functions of χ_z

χ_z	s_i	c_{i-1}	z_i
0	0	0	0
1	0 (1)	1 (0)	1
−2	$0(\overline{1})$	$\overline{1}$ (0)	$\overline{1}$
−1	$1(\overline{1})$	$\overline{1}$ (1)	0

$$c_i^- = [-12]_{\bar{i}}^{+}[-10]_{\bar{i}}^{-} + [-5]_{\bar{i}}^{+}[-4]_{\bar{i}}^{-}. \tag{43}$$

As was proven in Lemma 3, from these equations we can derive an implicit depth-1 implementation of the first step of the "totally parallel" addition scheme. Because $[-12]_{\bar{i}}^{+}$ and $[8]_{\bar{i}}^{-}$ are always 1, the results of Remark 2 can also be included in the derivation. Thus,

$$s_i^+ = [-5]_{\bar{i}}^{+} + [-4]_{\bar{i}}^{-} + [4]_{\bar{i}}^{+} + [5]_{\bar{i}}^{-} - 2 \tag{44}$$

$$s_i^- = [-6]_{\bar{i}}^{+} + [-6]_{\bar{i}}^{-} + [3]_{\bar{i}}^{+} + [3]_{\bar{i}}^{-} - 2 \tag{45}$$

$$c_i^+ = [3]_{\bar{i}}^{+} + [3]_{\bar{i}}^{-} + [6]_{\bar{i}}^{+} - 1 \tag{46}$$

$$c_i^- = [-10]_{\bar{i}}^{-} + [-5]_{\bar{i}}^{+} + [-4]_{\bar{i}}^{-} - 1. \tag{47}$$

The second step of the "totally parallel" addition is the computation of $z_i = s_i + c_{i-1}$. In this case, $\chi_z = -2(s_i^- + c_{i-1}^-) + s_i^+ + c_{i-1}^+$ and the second step can be described by Table 6. Following the same reasoning applied previously for the computation of $s_i^+, s_i^-, c_i^+, c_i^-$, this step can be implemented by:

$$z_i^+ = \widehat{[1]_{\bar{i}}^{+}} \tag{48}$$

$$z_i^- = \widehat{[-2]_{\bar{i}}^{-}}. \tag{49}$$

Theorem 2. *Assuming radix-2 SD operand representation and the SD codification in Table 4, the addition of two n-SD numbers can be computed by an explicit depth-2 LTN with $12n + 2$ LTGs, a maximum weight value of 10 and a maximum fan-in of 14.*

Proof. By proper substitutions, using (44), (45), (46), (47), (48), (49) provide an explicit depth-2 implementation scheme of the $2 - 1$ addition as follows:

$$\begin{aligned} z_i^- &= \operatorname{sgn}\{-2 - \chi_z\} = \operatorname{sgn}\{2(s_i^- + c_{i-1}^-) - s_i^+ - c_{i-1}^+ - 2\} \\ &= \operatorname{sgn}\Big\{2[-6]_{\bar{i}}^{+} + 2[-6]_{\bar{i}}^{-} + 2[3]_{\bar{i}}^{+} + 2[3]_{\bar{i}}^{-} \\ &\quad -[-5]_{\bar{i}}^{+} - [-4]_{\bar{i}}^{-} - [4]_{\bar{i}}^{+} - [5]_{\bar{i}}^{-} \\ &\quad +2[-10]_{\bar{i}-1}^{-} + 2[-5]_{\bar{i}-1}^{+} + 2[-4]_{\bar{i}-1}^{-} - [3]_{\bar{i}-1}^{+} \\ &\quad -[3]_{\bar{i}-1}^{-} - [6]_{\bar{i}-1}^{+} - 5\Big\} \end{aligned} \tag{50}$$

$$\begin{aligned} z_i^+ &= \operatorname{sgn}\{\chi_z - 1\} = \operatorname{sgn}\{-2(s_i^- + c_{i-1}^-) + s_i^+ + c_{i-1}^+ - 1\} \\ &= \operatorname{sgn}\Big\{-2[-6]_{\bar{i}}^{+} - 2[-6]_{\bar{i}}^{-} - 2[3]_{\bar{i}}^{+} - 2[3]_{\bar{i}}^{-} + [-5]_{\bar{i}}^{+} \\ &\quad +[-4]_{\bar{i}}^{-} + [4]_{\bar{i}}^{+} + [5]_{\bar{i}}^{-} \\ &\quad -2[-10]_{\bar{i}-1}^{-} - 2[-5]_{\bar{i}-1}^{+} - 2[-4]_{\bar{i}-1}^{-} + [3]_{\bar{i}-1}^{+} \\ &\quad +[3]_{\bar{i}-1}^{-} + [6]_{\bar{i}-1}^{+} + 2\Big\}. \end{aligned} \tag{51}$$

On the first level, we compute, for each digit position i, $i = 0, 1, \ldots, n-1$, the values $[-10]_{\bar{i}}^{-}$, $[-6]_{\bar{i}}^{-}$, $[-6]_{\bar{i}}^{+}$, $[-5]_{\bar{i}}^{+}$, $[-4]_{\bar{i}}^{-}$, $[3]_{\bar{i}}^{-}$, $[3]_{\bar{i}}^{+}$, $[4]_{\bar{i}}^{+}$, $[5]_{\bar{i}}^{-}$, and $[6]_{\bar{i}}^{+}$, i.e., we use 10 TGs per digit. On the second level, we need two TGs for each digit position i, $i = 0, 1, \ldots, n-1$, in order to compute $\widehat{[-2]_{\bar{i}}^{-}}, \widehat{[1]_{\bar{i}}^{+}}$ as stated by (50), (51). For the digit position $n-1$, we have to produce the carry-out. This can be also explicitly generated in depth-2 at the expanse of two TGs computing:

$$c_{n-1}^+ = \operatorname{sgn}\Big\{[3]_{\bar{n-1}}^{+} + [3]_{\bar{n-1}}^{-} + [6]_{\bar{n-1}}^{+} - 1\Big\} \tag{52}$$

$$c_{n-1}^- = \operatorname{sgn}\Big\{[-10]_{\bar{n-1}}^{-} + [-5]_{\bar{n-1}}^{+} + [-4]_{\bar{n-1}}^{-}\Big\}. \tag{53}$$

Therefore, the cost of the entire addition network is $12n + 2$. The maximum fan-in is 14 and the maximum weight value is 10. □

One can observe that all the quantities involved in Theorem 2 are in the same order of magnitude as in Theorem 1. Even though the scheme in Theorem 1 requires slightly larger maximum fan-in (14 instead of 12) and weight values (10 instead of 6), it has the advantage of explicitly computing the sum digits after the delay of 2 TGs.

4 Signed Digit Multioperand Addition and Multiplication

Threshold networks for multioperand addition and multiplication of n-bit binary operands have been reported [14], [15], [26], [29]. Generally speaking, multioperand addition and multiplication can be achieved in two steps, namely: First, reduce a multioperand addition (in multiplication, such addition is required for the reduction of the partial product matrix) into two rows; second, add the two rows to produce the final result. In addition to these two steps, the multiplication also requires a third step, the production of the partial product matrix. In this section, we investigate these processes. For such a scheme and nonredundant representations, the following has been suggested:

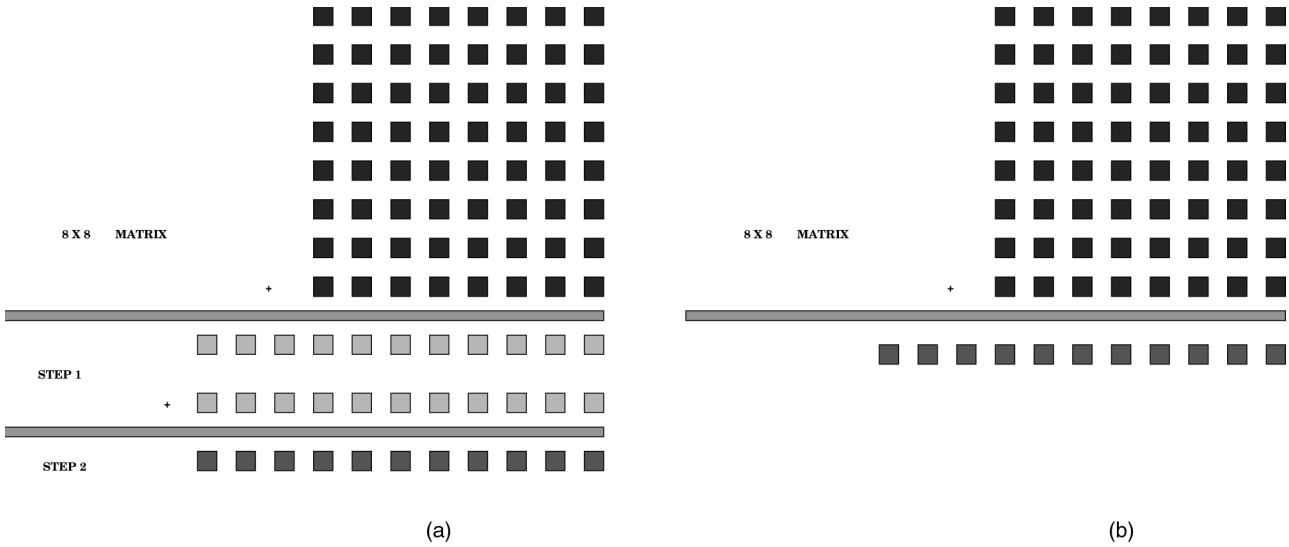

Fig. 5. Addition of eight 8-bit numbers. (a) Two-step reduction. (b) One-step reduction.

- The reduction of the multioperand addition (or the reduction of multiplication partial product matrix) into two rows can be achieved by depth-2 networks with the cost of the network, in terms of LTGs, in the order of $O(n^2)$ and a maximum fan-in in the order of $O(n \log n)$, see, for example, [15], [29].
- The entire multiplication can be implemented by a depth-4 network [14].

It was also suggested in [30], based on a result in [31], that multioperand addition can be computed in depth 2 and multiplication in depth 3, but no explicit construction for the networks and no complexity bounds are provided. A constructive approach can be derived if the result in [32] suggesting that a single threshold gate computing $F(x) = \text{sgn}\{\omega_0 + \omega_1 x_1 + \cdots + \omega_n x_n\}$ with arbitrary weights can be simulated by an explicit polynomial-size depth-2 network is used. Such a LOGSPACE-uniform construction as stated in [32] produces a network with $O(\log^{12} W(n))$ wires and the weights of those wires in order of $O(\log^8 W(n))$, for a total size of $O(n^{20} \log^{20} n)$. The total size for such a construction was further reduced to $O(n^{12} \log^{12} n)$ in [33]. LOGSPACE-uniform constructions for depth 2 multioperand addition and depth 3 multiplication has been suggested in [32], but the discussion about depth-2 multioperand addition or depth-3 multiplication schemes is marginal and no complexity bounds are explicitly given. In an attempt to assess the complexity of such a scheme for multioperand addition which operates on an n^2-input function instead of an n-input function, we can use the least expensive scheme in [32] and estimate that such a depth-2 multioperand addition or depth-3 multiplication network may require a total size of $O(n^{24} \log^{24} n)$.

In this section, we investigate the potential benefit that can be expected by using SD represented operands in TL multiplication schemes. First, we prove that multioperand addition can be achieved by a depth-2 network with $O(n^3)$ size, $O(n^3)$ weights, and $O(n^2)$ fan-in complexities. It must be noted that the proposed network performs an n operand to one result reduction in depth-2, not an n operand to two reduction in depth-2 as previously proposed schemes [15], [29] do. Subsequently, we show that the multiplication (that is, the generation of the partial products and the matrix reduction into one row representing the product) can be achieved with a depth-3 network with $O(n^3)$ size, $O(n^3)$ weights, and $O(n^2 \log n)$ fan-in complexities.

4.1 Depth-2 Multioperand Addition

It is well-known that, in order to perform n-bit multioperand addition, first, the n rows (representing the n numbers) are reduced to two, then the two rows are added to produce the final result. This two-step process is depicted, for the particular case of eight 8-bit numbers, in Fig. 5a. As indicated in the introduction of the section, the first step of multioperand addition not using redundant digit representations requires a depth-2 network and additional depth is required to perform the second step. In the following, we will prove that, if we assume SD operands in an appropriate representation radix the multioperand addition of n n-SD numbers and, consequently, the reduction of the partial product matrix of the multiplication operation, into one row, can be achieved in one computation step, as in Fig. 5b, requiring a depth-2 network. This is achieved by determining a radix which allows an n-digit "totally parallel" addition. Avizienis investigated this issue in [1], but from the dual point of view, by assuming a given radix-r SD representation and determine the maximum number of digits that can be added in "totally parallel" mode within that radix-r SD representation. In our investigation, the number of digits n is given and a minimum value for the radix-r must be found to compute n SD addition into a "totally parallel" mode. We answer to this question in the following lemma.

Lemma 4. *The simultaneous addition of n SDs can be done in a "totally parallel" mode by assuming a representation radix greater or equal with $2n - 1$.*

Proof. The simultaneous addition of n SDs can be done in a way similar to the addition of two digits. That is, in order to add the n digits $x_i^1, x_i^2, \ldots, x_i^n$ in a "totally parallel"

mode, we first have to produce an intermediate sum digit u_i and a transport digit t_i that satisfy (54) and, also, we have to satisfy the constraint indicating that the subsequent addition in (55) that gives the value of the sum digit z_i in the position i, can be performed without generating a carry-out. That is:

$$x_i^1 + x_i^2 + \cdots + x_i^n = u_i + rt_i \tag{54}$$

$$z_i = u_i + t_{i-1}. \tag{55}$$

We have to find the value of the radix r for which the computation in (54), (55) can be achieved and, also, the maximum absolute values that we can allow for the intermediate sum digit u_i and the transport digit t_i. In order to have consistency, we have to assume that $|x_i^j|_{max} = |z_i|_{max} = |x|_{max}$ and $|t_i|_{max} = |t_{i-1}|_{max} = |t|_{max}$. Therefore, if mapped in absolute maximal values, (54), (55) become:

$$n|x|_{max} \leq |u|_{max} + r|t|_{max} \tag{56}$$

$$|x|_{max} \geq |u|_{max} + |t|_{max}. \tag{57}$$

From (56), (57), we can derive the following inequalities:

$$\frac{n-1}{r-1}|x|_{max} \leq |t|_{max} \leq |x|_{max} - |u|_{max}. \tag{58}$$

In order to obtain the greatest range for $|t|_{max}$, we have to assume the maximum redundancy digit set, i.e., $|x|_{max} = r-1$ and, for the intermediate sum, an absolute maximum value of $|u|_{max} = \left[\frac{r}{2}\right]$. This, together with (58) and depending if we assume an odd radix r_o or an even one r_e, leads to $r_e \geq 2n$ or $r_o \geq 2n-1$. Therefore, in order to perform simultaneous addition of n SDs in a "totally parallel" mode, we have to use a representation radix greater or equal with $2n-1$. □

Assuming a representation radix of $2n-1$, we introduce the depth-2 multioperand addition scheme for n n-SD numbers.

Theorem 3. *Assuming radix-$(2n-1)$ SD representation, the multioperand addition of n n-SD numbers (that is, the reduction via addition of an n-digit n row matrix to one row) can be computed by an explicit depth-2 LTN with the size of $O(n^3)$. The maximum weight value is the order of $O(n^3)$ and the maximum fan-in value is in the order of $O(n^2)$.*

Proof. Assume that the n SD numbers we have to add are $x_i = (x_i^1, x_i^2, \ldots, x_i^n)$, with $i = 1, 2, \ldots, n$ and all the digits x_i^j, $i, j = 1, 2, \ldots, n$ can take value within the symmetric digit set

$$\mathcal{D} = \{\overline{2n-2}, \overline{2n-3}, \ldots, \overline{1}, 0, 1, \ldots, 2n-3, 2n-2\}.$$

Given that the radix-$(2n-1)$ allows for "totally parallel" addition of n SDs, we can compute the sum of the n numbers as follows: For each position i, produce an intermediate sum digit u_i and a transport digit t_i that satisfy $u_i + (2n-1)t_i = x_i^1 + x_i^2 + \cdots + x_i^n$; the sum digit z_i in the position i is computed as $z_i = u_i + t_{i-1}$ without generating a carry-out. If we assume that the greatest absolute values for the input digits, transport digits, and intermediate sum digits are $|x|_{max} = 2n-2$, $|t|_{max} = n-1$, and $|u|_{max} = n-1$, respectively, the sum digit z_i will depend only on the values of the digits in the columns i and $i-1$ of the multioperand addition matrix and can be computed with the two-step approach. With this scheme, the network implementing the multioperand addition contains one subcircuit performing this computation for each digit position i, $i = 1, 2, \ldots, n$. Obviously, the cost of the entire network is n times the cost of the circuit performing the "totally parallel" addition of n digits. The delay of the multioperand addition, the maximum weight, and fan-in values are imposed by their similar values in the circuit performing the "totally parallel" addition of n digits.

The direct implementation of this two-step computation procedure with the scheme in Lemma 1 is not convenient because it will lead to a depth-4 LTN. However, given that any generalized symmetric Boolean function can be implemented with a depth-2 network, we can reduce the depth of the network to 2 if we are able to compute the value of z_i with a symmetric function of $2n$ input variables, i.e., all the digits in the columns i and $i-1$ of the multioperand addition matrix. This can be done by observing the direct link that exists between the value of z_i and the value assumed by the weighted sum χ of all the $2n$ digits

$$x_i^1, x_i^2, \ldots, x_i^n, x_{i-1}^1, x_{i-1}^2, \ldots, x_{i-1}^n$$

in the columns i and $i-1$, computed as in (59).

$$\chi = (2n-1)\sum_{j=1}^{n} x_i^j + \sum_{j=1}^{n} x_{i-1}^j. \tag{59}$$

This link exists as a consequence of the fact that, under the maximum value assumptions we made for the input digits, transport digits, and intermediate product digits, the radix-$(2n-1)$ representation of the sum χ is (t_i, z_i, t_{i-1}), where the values of t_i, z_i, and t_{i-1} follow from (54), (55). The maximum absolute value that can be assumed by χ can be derived from (59) under the assumption that all the x_i^j, x_{i-1}^j digits are $2n-2$. This will lead to $|\chi|_{max} = 4n^2(n-1)$ and to a variation domain for χ equal to $[-4n^2(n-1), 4n^2(n-1)]$.

Because the digits involved into the computation in (59) belong to the set $\mathcal{D}$, we need $[\log(2n-1)] + 1$ bits for their 2's complement codification. Under this codification, each digit x_i^j is represented by a $([\log(2n-1)]+1)$-tuple $(x_i^{j,[\log(2n-1)]}, x_i^{j,[\log(2n-1)]-1}, \ldots, x_i^{j,1}, x_i^{j,0})$. Each of these bits will take part in the computation of χ with a weight that corresponds to its position inside the digit and following the 2's complement codification convention. With this assumption, (59) becomes:

$$\chi = (2n-1)\sum_{j=1}^{n}\left(-2^{[\log(2n-1)]}\ x_i^{j,[\log(2n-1)]} + \sum_{k=0}^{[\log(2n-1)]-1} 2^k x_i^{j,k}\right) + \sum_{j=1}^{n}\left(-2^{[\log(2n-1)]}\ x_{i-1}^{j,[\log(2n-1)]} + \sum_{k=0}^{[\log(2n-1)]-1} 2^k x_{i-1}^{j,k}\right), \tag{60}$$

assuming all of these product digit z_i can be expressed by a function $\mathcal{F}(\chi)$. Obviously, because of the weighted manner, we did the computation of the sum χ, the function $\mathcal{F}$ is symmetric in all of the input variables[9] and, consequently, it can be implemented using the method described in Lemma 1 with a depth-2 LTN.

Because z_i can assume any digit value in the set $\mathcal{D}$, we again need $[\log(2n-1)]+1$ bits for its codification. Therefore, in order to compute $\mathcal{F}(\chi)$, we have to compute $[\log(2n-1)]+1$ symmetric Boolean functions $\mathcal{F}_i(\chi)$, $i = 0, 1, \ldots, [\log(2n-1)]$. For the implementation of each symmetric Boolean function $\mathcal{F}_i(\chi)$, we need r_i LTGs in the first layer of the network, r_i being the number of intervals in the definition domain where $\mathcal{F}_i$ assume the value of 1 and one LTG in the second layer. Consequently, the computation of the function $\mathcal{F}(\chi)$ can be done with:

$$G = \sum_{i=0}^{[\log(2n-1)]} r_i + [\log(2n-1)] + 1 \tag{61}$$

LTGs. The definition domain for $\mathcal{F}(\chi)$ is given by $[-4n^2(n-1), 4n^2(n-1)]$ and, within it, $\mathcal{F}(\chi)$ can change its value at most $I = \frac{2\times 4n^2(n-1)+1}{2n-1}$ times. As a consequence, for each Boolean function $\mathcal{F}_i(\chi)$, the number of intervals r_i cannot be greater than I. Given that the changes of the values of $\mathcal{F}_i(\chi)$ can appear only in certain fixed positions common for all of them, we can use the gate sharing concept we introduced in [29]. In this way, the gates associated with the upper limit of the intervals can be shared between the networks implementing the Boolean functions $\mathcal{F}_i(\chi)$. This fact leads to an upper bound of $\left\lceil \frac{8n^2(n-1)+1}{2n-1} \right\rceil$ for the maximum number of TGs in the first level of the network. The second level of the network has to contain one gate for each $\mathcal{F}_i(\chi)$, i.e., bit position in the 2's complement representation of z_i, then it can be built with $[\log(2n-1)]+1$ gates.

Therefore, the network computing the sum digit z_i as $\mathcal{F}(\chi)$ can be built with at most

$$\left\lceil \frac{8n^2(n-1)+1}{2n-1} \right\rceil + [\log(2n-1)] + 1$$

LTGs. Because we need one such network for each digit position i and the multioperand addition matrix has n columns,[10] the cost of the entire multioperand addition is upper bounded by $n(\left\lceil \frac{8n^2(n-1)+1}{2n-1} \right\rceil + [\log(2n-1)] + 1)$.

Asymptotically speaking, this leads to an implementation of the multioperand addition of n n-SD numbers with a depth-2 network having the number of LTGs in the order of $O(n^3)$.

The maximum weight value is upper bounded by the dimension of the definition domain, i.e., $8n^2(n-1)+1$, and, consequently, it is in the order of $O(n^3)$. The maximum fan-in value is imposed by the gates in the second level of the network which take as inputs all the bits participating into the computation, i.e., $2n([\log(2n-1)]+1)$, and some outputs of the gates on the first level. The total number of gates in the first level of the network is upper bounded by $\left\lceil \frac{8n^2(n-1)+1}{2n-1} \right\rceil$ and, consequently, the maximum fan-in value is in the order of $O(n^2)$. □

We conclude our investigation on TL networks for the multiplication of SD operands by introducing a depth-3 LTN for multiplication which uses the multioperand addition scheme we presented in Theorem 3.

4.2 Depth-3 Multiplication

Multiplication is achieved with the generation and reduction of a partial product matrix. In the previous section, we showed that the multioperand addition (and, by extension, the reduction of the multiplication partial product matrix) can be performed in depth-2 using threshold networks and SD representations. In this section, we investigate the entire multiplication operation, including the generation of the partial product matrix.

In the case of nonredundant operand representation, the generation of the partial product matrix can be performed at the expanse of n^2 TGs in depth-1 because we need one AND gate to produce each partial product $z_{i,j} = x_i \times y_j$, $i, j = 0, 1, \ldots, n-1$. This may not be true for sign digit operands where each partial product $z_{i,j}$ is an SD which has to be computed as the product of two SDs x_i and y_j. In essence, even though, using TL and SD representation, the partial product reduction can be achieved by a depth-2, it is not said that multiplication can be achieved by a depth-3 network.

To achieve a depth-3 multiplication, we use Theorem 3 for the reduction of the partial product matrix and use implicit computations in the network connecting the partial product production and the first stage of partial product reduction. Given that, in order to use the scheme in Theorem 3, all the partial products $z_{i,j}$, $i, j = 0, 1, \ldots, n-1$, have to assume values inside the digit set $\mathcal{D} = \{\overline{2n-2}, \overline{2n-3}, \ldots, \overline{1}, 0, 1, \ldots, 2n-3, 2n-2\}$, we have to restrict the maximum absolute values for the SDs

9. The number of input Boolean variables is given by the product of the number of digits involved into the computation of z_i and the number of bits we need in order to represent a digit in $\mathcal{D}$, i.e., $2n([\log(2n-1)]+1)$.

10. If the multioperand addition matrix is the partial product matrix corresponding to the multiplication of two n-SD numbers, the number of columns is $2n$ and the cost changes as a consequence. However, this does not change the asymptotic cost.

x_i and y_j to $\lceil\sqrt{2n-2}\,\rceil$. In the following lemma, we assume that the operand digits are represented with the 2's complement codification discussed in Section 2 and prove that the entire partial product matrix can be produced by a depth-2 LTN with polynomially bounded size, weight, and fan-in values.

Lemma 5. *Assuming two n-SD operands $X = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ and $Y = (y_{n-1}, y_{n-2}, \ldots, y_1, y_0)$ with $|x_i| \le \lceil\sqrt{2n-2}\,\rceil$ and $|y_j| \le \lceil\sqrt{2n-2}\,\rceil$, the partial product matrix $\|z_{i,j}\|_{i,j=0,1,\ldots,n-1}$, $z_{i,j} = x_i \times y_j$ can be produced by a depth-2 LTN with the size measured in terms of LTGs in the order of $O(n^3)$. The maximum weight value is in the order of $O(n)$ and the maximum fan-in value is in the order of $O(n)$.*

Proof. We assume that all the SDs are represented in the 2's complement notation by $x_i = (x_i^{d-1}, x_i^{d-2}, \ldots, x_i^1, x_i^0)$ and $y_j = (y_j^{d-1}, y_j^{d-2}, \ldots, y_j^1, y_j^0)$. The value of d is imposed by the maximum absolute value of $\lceil\sqrt{2n-2}\,\rceil$ we have assumed for the operand digits and is equal to $\lceil\log\lceil\sqrt{2n-2}\,\rceil\rceil + 1$. With these assumptions, the partial product $z_{i,j}$ can be expressed as in the following equation:

$$z_{i,j} = x_i \times y_j = \left(-2^{d-1}x_i^{d-1} + \sum_{k=0}^{d-2} 2^k x_i^k\right)\left(-2^{d-1}y_j^{d-1} + \sum_{l=0}^{d-2} 2^l y_j^l\right). \tag{62}$$

On the other hand, $z_{i,j}$ is a SD in the set

$$\mathcal{D} = \{\overline{2n-2}, \overline{2n-3}, \ldots, \overline{1}, 0, 1, \ldots, 2n-3, 2n-2\}$$

and can be represented by the $([\log(2n-1)]+1)$-tuple $(z_{i,j}^{[\log(2n-1)]}, z_{i,j}^{[\log(2n-1)]-1}, \ldots, z_{i,j}^1, z_{i,j}^0)$. Consequently, each bit $z_{i,j}^r$, $r = 0, 1, \ldots, [\log(2n-1)]$ can be expressed by a symmetric Boolean function $F_r(\chi_m)$ with the weighted sum χ_m computed as in (63).

$$\chi_m = 2^{2d-2}x_i^{d-1}y_j^{d-1} + \sum_{k=0}^{d-2}\sum_{l=0}^{d-2} 2^{k+l}x_i^k y_j^l - \sum_{k=0}^{d-2} 2^{d+k-1}y_j^{d-1}x_i^k - \sum_{l=0}^{d-2} 2^{d+l-1}x_i^{d-1}y_j^l. \tag{63}$$

This function can be implemented with a depth-2 network, as shown in Lemma 1. By its construction, χ_m can assume values in the definition domain $[-(2n-2), 2n-2]$. Consequently, the definition domain for all the $F_r(\chi_m)$ describing the partial product $z_{i,j}$ is given by $[-(2n-2), 2n-2]$. Within this definition domain, any $F_r(\chi_m)$ can change its value at most $\frac{4(n-1)+1}{2}$ times. Using the same way of reasoning as in Theorem 3, an upper bound of $\left\lceil\frac{4(n-1)+1}{2}\right\rceil$ can be obtained for the maximum number of TGs in the first level of the network. The second level of the network has to contain one gate for each $F_r(\chi_m)$, i.e., bit position in the 2's complement representation of the partial product $z_{i,j}$, then it can be built with $[\log(2n-1)]+1$ gates.

Therefore, the network computing the partial product $z_{i,j}$ can be built with at most $\left\lceil\frac{4(n-1)+1}{2}\right\rceil + [\log(2n-1)] + 1$ LTGs. Because one such network for each digit pair (i,j), $i,j = 0, 1, \ldots, n-1$, is required, the cost of the network producing the entire partial product matrix is upper bounded by $n^2(\left\lceil\frac{4(n-1)+1}{2}\right\rceil + [\log(2n-1)] + 1)$.

This leads to an implementation cost of the depth-2 network producing the partial product matrix in the order of $O(n^3)$. The maximum weight value is upper bounded by the dimension of the definition domain for the $F_r(\chi_m)$ functions, i.e., $4(n-1)+1$, and, consequently, it is in the order of $O(n)$. The maximum fan-in value is imposed by the gates in the second level of the network which take as inputs all the bits participating into the computation, i.e., $2[\log\lceil\sqrt{2n-2}\rceil] + 2$, and some outputs of the gates on the first level. Because we proved that the total number of gates in the first level of the network is upper bounded by $\left\lceil\frac{4(n-1)+1}{2}\right\rceil$, the maximum fan-in value is also in the order of $O(n)$. □

By connecting the results for the multioperand addition and the generation of the partial product matrix for SD operands, we obtain a depth-4 scheme for the multiplication of SD numbers as stated in the following corollary:

Corollary 1. *Assuming radix-$(2n-1)$ SD representation the multiplication of two n-SD numbers can be computed by an explicit depth-4 LTN with the size measured in terms of LTGs in the order of $O(n^3)$. The maximum weight value is the order of $O(n^3)$ and the maximum fan-in value is in the order of $O(n^2)$.*

Proof. Trivial from Lemma 5 and Theorem 3. □

The delay of the multiplication network can still be reduced by producing the partial product matrix using an implicit computation scheme presented in Lemma 3.

Theorem 4. *Assuming radix-$(2n-1)$ SD representation the multiplication of two n-SD numbers can be computed by an explicit depth-3 LTN with the size in the order of $O(n^3)$. The maximum weight value is the order of $O(n^3)$ and the maximum fan-in value is in the order of $O(n^2 \log n)$.*

Proof. Trivial. First, use the implicit implementation (Lemma 3) in order to produce the partial products $z_{i,j}$ with the delay of one TG. This derivation will not change the asymptotic costs we derived in Lemma 5. Second, use the depth-2 multioperand addition in Theorem 3 to produce the product. The implicit computation of the partial products will only increase the fan-in of the gates in the first level of the network performing the multioperand addition from $2n([\log(2n-1)]+1)$ to at most $2n(4n-3)([\log(2n-1)]+1)$. This will change the asymptotic bound for the fan-in from $O(n^2)$ to $O(n^2 \log n)$. The asymptotic size of the network and the

maximum weight value will remain unchanged. Consequently, this depth-3 scheme has a network size in the order of $O(n^3)$ and the maximum weight value is the order of $O(n^3)$. □

5 Conclusions

We investigated LTNs for symmetric Boolean functions $2-1$ addition, multioperand addition, and multiplication. We assumed SD number representation and we were mainly concerned with establishing the limits of the circuit designs using threshold based networks. We have shown that, assuming radix-2 representation, the addition of two n-SD numbers can be computed by an explicit depth-2 LTN with $O(n)$ size and $O(1)$ weight and fan-in values. If a higher radix of $(2n-1)$ is assumed, we proved that the multioperand addition of n n-SD numbers can be computed by an explicit depth-2 LTN with the size in the order of $O(n^3)$, with the maximum weight value in the order of $O(n^3)$ and the maximum fan-in value in the order of $O(n^2)$. Finally, we have shown that the multiplication of two n-SD numbers can be computed by an explicit depth-3 LTN with the size in the order of $O(n^3)$. The maximum weight value is in the order of $O(n^3)$ and the maximum fan-in value is in the order of $O(n^2 \log n)$.

References

[1] A. Avizienis, "Signed-Digit Number Representations for Fast Parallel Arithmetic," *IRE Trans. Electronic Computers,* vol. 10, pp. 389-400, Sept 1961.

[2] C. Chow and J. Robertson, "Logic Design of a Redundant Binary Adder," *Proc. Fourth Symp. Computer Arithmetic,* pp. 109-115, Oct. 1978.

[3] N. Takagi, H. Yasuura, and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *IEEE Trans. Computers,* vol. 34, no. 9, pp. 789-796, Sept. 1985.

[4] M.D. Ercegovac and T. Lang, "Fast Radix-2 Division with Quotient-Digit Prediction," *J. VLSI Signal Processing,* vol. 1, pp. 169-180, Nov. 1989.

[5] M.D. Ercegovac and T. Lang, "Simple Radix-4 Division with Operands Scaling," *IEEE Trans. Computers,* vol. 39, no. 9, pp. 1,204-1,208, Sept. 1990.

[6] L. Ciminiera and P. Montuschi, "High Radix Square Rooting," *IEEE Trans. Computers,* vol. 39, no. 10, pp. 1,220-1,231, Oct. 1990.

[7] T. Shibata and T. Ohmi, "A Functional MOS Transistor Featuring Gate-Level Weighted Sum and Threshold Operations," *IEEE Trans. Electron Devices,* vol. 39, pp. 1,444-1,455, June 1992.

[8] T. Shibata and T. Ohmi, "Neuron MOS Binary-Logic Integrated Circuits—Part I: Design Fundamentals for Soft-Hardware Circuit Implementation," *IEEE Trans. Electron Devices,* vol. 40, pp. 570-575, Mar. 1993.

[9] T. Shibata and T. Ohmi, "Neuron MOS Binary-Logic Integrated Circuits—Part II: Simplifying Techniques of Circuit Configuration and their Practical Applications," *IEEE Trans. Electron Devices,* vol. 40, pp. 974-979, May 1993.

[10] H. Ozdemir, A. Kepkep, B. Pamir, Y. Leblebici, and U. Cilingiroglu, "A Capacitive Threshold-Logic Gate," *IEEE J. Solid-State Circuits,* vol. 31, pp. 1,141-1,150, Aug. 1996.

[11] W.S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin Math. Biophysics 5,* pp. 115-133, 1943 (reprinted in *Neurocomputing Foundations of Research,* J.A. Anderson and E. Rosenfeld, eds. MIT Press, 1988).

[12] W. Pitts and W.S. McCulloch, "How We Know Universals: The Perception of Auditory and Visual Forms," *Bulletin Math. Biophysics 9,* pp. 127-147, 1947 (reprinted in *Neurocomputing Foundations of Research,* J.A. Anderson and E. Rosenfeld, eds. MIT Press, 1988).

[13] K.Y. Siu and J. Bruck, "Neural Computation of Arithmetic Functions," *Proc. IEEE,* vol. 78, pp. 1,669-1,675, Oct. 1990.

[14] T. Hofmeister, W. Hohberg, and S. Kohling, "Some Notes on Threshold Circuits and Multiplication in Depth 4," *Information Processing Letters,* vol. 39, pp. 219-225, 1991.

[15] R. Lauwereins and J. Bruck, "Efficient Implementation of a Neural Multiplier," *Proc. Second Int'l Conf. Microelectronics for Neural Networks,* pp. 217-230, Oct. 1991.

[16] K. Siu, V. Roychowdhury, and T. Kailath, "Depth-Size Tradeoffs for Neural Computation," *IEEE Trans. Computers,* vol. 40, no. 12, Dec. 1991.

[17] S. Vassiliadis, S. Cotofana, and K. Bertels, "$2-1$ Addition and Related Arithmetic Operations with Threshold Logic," *IEEE Trans. Computers,* vol. 45, no. 9, pp. 1,062-1,068, Sept. 1996.

[18] S. Cotofana and S. Vassiliadis, "δ-bit Serial Addition with Linear Threshold Gates," *J. VLSI Signal Processing,* vol. 3, pp. 249-264, Dec. 1996.

[19] Y. Leblebici, H. Ozdemir, A. Kepkep, and U. Cilingiroglu, "A Compact High-Speed $(31,5)$ Parallel Counter Circuit Based on Capacitive Threshold-Logic Gates," *IEEE J. Solid-State Circuits,* vol. 31, pp. 1,177-1,183 Aug. 1996.

[20] W. Weber, S. Prange, R. Thewes, E. Wohlrab, and A. Luck, "On the Application of the Neuron MOS Transistor Principle for Modern VLSI Design," *IEEE Trans. Electron Devices,* vol. 43, pp. 1,700-1,708, Oct. 1996.

[21] S. Cotofana and S. Vassiliadis, "Periodic Symmetric Functions with Feed-Forward Neural Networks," *Proc. NEURAP '95/96 Neural Networks and Their Applications,* pp. 215-221, Mar. 1996.

[22] S. Muroga, "The Principle of Majority Decision Elements and the Complexity of Their Circuits," *Proc. Int'l Conf. Information Processing,* pp. 400-407, June 1959.

[23] R. Minnick, "Linear Input Logic," *IEEE Trans. Electronic Computers,* vol. 10, pp. 6-16, Mar. 1961.

[24] W. Kautz, "The Realization of Symmetric Switching Functions with Linear-Input Logical Elements," *IRE Trans. Electronic Computers,* vol. 10, pp. 371-378, Sept. 1961.

[25] R. Paturi and M. Saks, "On Threshold Circuits for Parity," *Proc. IEEE Symp. Foundations of Computer Science,* pp. 397-404, Oct. 1990.

[26] S. Vassiliadis, J. Hoekstra, and S. Cotofana, "Block Save Addition with Telescopic Sums," *Proc. 21st Euromicro Conf.,* pp. 701-707, Sept. 1995.

[27] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design.* New York: John Wiley & Sons, 1979.

[28] S. Cotofana and S. Vassiliadis, "2|1 Redundant Binary Addition with Threshold Logic," *Proc. IEEE 30th Asilomar Conf. Signals, Systems, and Computers,* pp. 889-893, Nov. 1996.

[29] S. Vassiliadis, S. Cotofana, and J. Hoekstra, "Block Save Addition with Threshold Logic," *IEEE 29th Asilomar Conf. Signals, Systems, and Computers,* pp. 575-579, Oct. 1995.

[30] K.-Y. Siu and V.P. Roychowdhury, "On Optimal Depth Threshold Circuits for Multiplication and Related Problems," *SIAM J. Discrete Math.,* vol. 7, pp. 284-292, May 1994.

[31] M. Goldmann, J. Håstad, and A. Razborov, "Majority Gates vs. General Weighted Threshold Gates," *Proc. Seventh Ann. Conf. Structure in Complexity Theory (SCTC '92),* pp. 2-13, June 1992.

[32] M. Goldmann and M. Karpinski, "Simulating Threshold Circuits by Majority Circuits," *SIAM J. Computing,* vol. 27, pp. 230-246, Feb. 1998.

[33] T. Hofmeister, "A Note on the Simulation of Exponential Threshold Weights," *Proc. COCOON: Ann. Int'l Conf. Computing and Combinatorics,* pp. 136-141, 1996.

Sorin Cotofana received the MS degree in computer science from the Polytechnica University of Bucharest, Romania, and the PhD degree in electrical engineering from Delft University of Technology, The Netherlands. He worked for a decade with the Research & Development Institute for Electronic Components (ICCE) in Bucharest. His work experience in ICCE was related to structured design of digital systems, design rule checking of IC's layout, logic and mixed-mode simulation of electronic circuits, testability analysis, and image processing. He is currently an assistant professor in the Electrical Engineering Department at Delft University of Technology, The Netherlands. His research interests include computer arithmetic, parallel architectures, embedded systems, neural networks, fuzzy logic, computational geometry, and computer-aided design.

Stamatis Vassiliadis is a professor in the Electrical Engineering Department at Delft University of Technology, The Netherlands. He has also served on the faculties of Cornell University, Ithaca, New York, and the State University of New York, Binghamton. He worked for a decade with IBM in the Advanced Workstations and Systems Laboratory in Austin, Texas, the Mid-Hudson Valley laboratory in Poughkeepsie, New York, and the Glendale laboratory in Endicott, New York. At IBM, he was involved in a number of projects regarding computer design, organizations, and architectures, and in the leadership of advanced research projects. A number of his design and implementation proposals have been implemented in commercially available systems and processors, including the IBM 9370 model 60 computer system, the IBM POWER II, the IBM AS/400 Models 400, 500, and 510, Server Models 40S and 50S, the IBM AS/400 Advanced 36, and the IBM S/390 G4 and G5 computer systems. For his work, he received numerous awards, including 23 levels of Publication Achievement Awards, 15 levels of Invention Achievement Awards, and an Outstanding Innovation Award for Engineering/Scientific Hardware Design in 1989. Six of his 65 patents have been rated with the highest patent ranking in IBM and, in 1990, he was awarded the highest number of patents in IBM.

Dr. Vassiliadis is a member of the IEEE Computer Society and an IEEE fellow. His research interests include computer architecture, embedded systems, hardware design and functional testing of computer systems, parallel processors, computer arithmetic, neural networks, fuzzy logic and systems, and software engineering.