Design and Synthesis of High Speed Low Power Signed Digit Adders

Gh. Jaberipur$^{1,2}$  S. Gorgin$^{3,4}$

1- Associate Professor, Department of Electrical & Computer Engineering, Shahid Beheshti University, Tehran, Iran
2- School of Computer Science, Institute for Research in Fundamental Science (IPM), Tehran, Iran.
jaberipur@sbu.ac.ir

3-PhD Student, Department of Electrical & Computer Engineering, Shahid Beheshti University, Tehran, Iran
4- School of Computer Science, Institute for Research in Fundamental Science (IPM), Tehran, Iran.
gorgin@sbu.ac.ir

Abstract:

Signed digit (SD) number systems provide the possibility of constant-time addition, where inter-digit carry propagation is eliminated. Such carry-free addition is primarily a three-step process; adding the equally weighted SDs to form the primary sum digits, decomposing the latter to interim sum digits and transfer digits, which commonly belong to \{-1, 0, 1\}, and finally adding the transfers to the corresponding (i.e., with the same weight) interim sum digits. All the final sum digits are therefore obtained in parallel. The special case of radix-2$^h$ maximally redundant SD number systems is more attractive due to maximum symmetric range (i.e., [-2$^h$, 2$^h$]) with only one redundancy bit per SD, and the possibility of more efficient carry-free addition. The previous relevant works use three parallel adders that compute sum and sum±1, where some speed-up is gained at the cost of more area and power. In this paper, we propose an alternative nonspeculative addition scheme that uses carry-save encoding for representation of the primary sum and interim sum digits and computes the transfer digits via a fast combinational logic. The simulation and synthesis of the proposed adder, based on 0.13 µm CMOS technology, shows advantages in terms of speed, power and area.

Keywords: Computer arithmetic, Carry-free addition, Signed-digit number systems, Low power design, Maximal redundancy.
1. Introduction

Addition is the basic computer arithmetic operation. One study [1] shows that the share of add/subtract operations, among all the floating point arithmetic operations, is 55%. Multiplication, with its 37% share, is the next frequent operation that is often implemented via several add operations.

Traditional ripple-carry adders are very slow due to the long chain of carry-propagation logic. Consequently, the latency of an n-digit carry-ripple adder is linearly depending on n (i.e., O (n)) [2]. Carry-accelerating techniques, for example, in carry look-ahead adders [3] or carry select adders [4] improve the order of latency to O (log n) and O (sqrt(n)), respectively. However, Constant-time (i.e., O (1)) adders are not possible if the sum, as usual, is to be represented in a conventional nonredundant format [5]. Otherwise, carry propagation may be totally eliminated if the sum is allowed to be represented in a redundant format [6].

Signed digit (SD) redundant number systems, where each digit may have a positive or negative value and there is not a single sign bit for the whole number, have been used in several computer arithmetic circuits [7]. SD number systems fall within the generalized signed digit (GSD) number systems that are fixed radix and contiguous number systems with digit set [-α, β], where α, β ≥ 0 [6]. A GSD number system is deemed redundant (nonredundant) if the redundancy index ρ = α + β + 1 – r is positive (zero), where r is the radix. In the ordinary SD number systems that have balanced digit sets (i.e., α = β), we have ρ = 2α+1 – r > 0. In practice r is a power of two (e.g., 2^k), where the latter inequality leads to α ≥ 2^k-1. The case of α = 2^k-1 corresponds to the minimally redundant digit set [-2^k-1, 2^k-1], where ρ = 1 and at least h+1 bits are needed to represent a digit. But with the same number of bits α could grow nearly twice up to 2^k-1 corresponding to maximally redundant digit set [-2^k+1, 2^k-1] with ρ = 2^-1. The latter is particularly attractive due to maximum range of numbers with minimum number of redundancy bits. The higher radix SD number systems [8], where h ≥ 2, trade-off less storage and data paths for slower arithmetic [9]. For example, the radix-2 (i.e., h = 1) SD number system uses two bits to represent each digit in [-1, 1], thus doubling the storage and number of data paths with respect to conventional nonredundant binary systems. However, addition speed is the highest in comparison with the cases with higher radices (i.e., h ≥ 2).

The main benefit of SD number systems is the possibility of constant time addition, where the latency is small and independent of the number of digits in the operands. A drawback, however, is that conversion of a redundant result to a conventional human readable representation (e.g., nonredundant binary or decimal) may require word wide carry propagation. Therefore, use of redundant representations is justifiable only when there are several arithmetic operations to take place before conversion. This situation occurs in the implementation of composite arithmetic operations, such as multiplication or division with several embedded additions or subtractions, respectively [2] or in floating point addition [10]. It can be found also in whole computations such as function evaluation in general purpose or special purpose hardware units (e.g., digital signal processors [11]). In such applications, the slow final conversion should be well compensated by the latency savings that are compounded over many constant-time redundant operations. Therefore, very fast redundant adders are on demand.

In the addition of any two weighted numbers, the sum digit in position i obviously depends on the two operand digits in the same position. Moreover, for the cases of ρ = 0 (i.e., nonredundant), ρ = 1, and ρ ≥ 2, it also depends on all, two, and one less significant digit pairs, respectively. This is further explained below.

- ρ = 0: In a conventional nonredundant number system the sum digit in each position i (≥ 0) is a function of 2(+1) operand digits; namely two operand digits per each of the positions i, i – 1, 1, 0.
- ρ = 1: For minimally redundant number systems the sum digit in position i (≥ 2) is a function of the digits in positions i, i – 1 and i – 2. The constant time addition in this case is called carry-limited [6].
- ρ ≥ 2: In this case, that includes the maximally redundant case of ρ = r – 1, the sum digit in each position i (≥ 1) depends on only four operand digits in positions i and i – 1 [12]. The constant time addition in this case is called carry-free [6].

Each of the three steps of conventional constant time SD addition algorithm (Algorithm 1) is roughly as slow as an h-bit adder. Therefore, it would be desirable to parallelize or fuse these steps for more speed. For example, the case of maximally redundant SD (MRS) number systems are attractive due to its potential for improvements leading to less latency; noteworthy are the speculative MRS number systems [8], where h+1-bit addition or carry look-ahead are allowed to be represented in a redundant format [6]. Consequently, the latency of an n-digit SD addition is linearly depending on n (i.e., O (n)) [2]. Carry-look-ahead adders reduce this latency to O (log n) and O (sqrt(n)), respectively. However, Constant-time (i.e., O (1)) adders are not possible if the sum, as usual, is to be represented in a conventional nonredundant format [5]. Otherwise, carry propagation may be totally eliminated if the sum is allowed to be represented in a redundant format [6].

In this paper we present an improved nonspeculative addition scheme for MRS number systems with power of two radix r = 2^h. The paper is organized as follows. The conventional three-step carry-free SD addition algorithm is explained in Section 2. Some previous works on improved MRS number systems, including a recent one [13], are briefly reviewed in Section 3. The contribution of this paper is presented in Section 4. The results of synthesis and simulation of this work versus previous contributions, all based on 0.13 μm CMOS technology, are reported in Section 5. Finally we draw our conclusions in Section 6.

2. SD Carry-Free Addition Algorithm

In the conventional nonredundant number systems, cardinality ξ of the digit set is equal to the radix r (e.g., the binary digit set [0, 1] or decimal [0, 9]). The
conventional radix-complement or diminished-radix-complement number systems notwithstanding, in order to allow negative numbers, one could think of digit sets with signed values (e.g., nonredundant radix-10 digit set \([-5, 4]\) and radix-16 digit set \([-8, 7]\)). To build up a balanced range number system, however, it is necessary to allow \(\xi > r\) for even radices (e.g., binary, decimal, or hexadecimal digit sets \([-1, 1], [-5, 5],\) or \([-8, 8]\), respectively). The positive redundancy index \(\rho = \xi - r > 0\), in such cases leads to redundant number systems, where some values may be redundantly represented by more than one digit combination.

**Example 1 (decimal redundancy):** Consider the decimal digit set \([-7, 7]\), where \(\xi = 15 > r = 10\). In this redundant decimal number system, 2008 may be also represented as 201\((-2)\), where 2008 = 2×1000 + 1×10 + \((-2)\)×1, or 1979 represented also as 20\((-2)\)\((-1)\).

The signed digit (SD) number systems, introduced by Avezienis [14], represent a special case of redundant number systems, where the radix-r digit set \([-\alpha, \alpha]\) is balanced. The constraint \(\alpha \geq r/2\) is required for continuity of the represented numbers. This leads to \(\xi = 2\alpha + 1 > r\). The most useful property of redundant number systems is the possibility of carry-free addition, where the carry propagation chain is limited to a few number of digits [6].

This chain is only one digit long for SD numbers in case of \(r \geq 3\) and \(\alpha \geq (r+1)/2\) [14], where the carry generated in any position \(i\) will not propagate beyond position \(i + 1\). A general carry-free addition scheme for radix-\(2^h\) SD number systems with digit set \([-\alpha, \alpha]\), is described by Algorithm 1, where \(\alpha > 2^{-h}\). Fig. 1 depicts a block-diagram representation of the Algorithm.

**Algorithm 1** (Carry-free SD addition):

**Input:** Two \(n\)-digit radix-\(2^h\) SD numbers \(X = x_{n-1}\ldots x_0\) and \(Y = y_{n-1}\ldots y_0\), where \(-\alpha \leq x_i, y_i \leq \alpha\).

**Output:** An \((n+1)\)-digit radix-\(2^h\) SD number \(S = s_n\ldots s_0\)

1. Compute the \(n\)-digit radix-\(2^h\) SD number \(P = p_{n-1}\ldots p_0 = X + Y\), by digit-parallel computation of \(p_i = x_i + y_i\) for \(0 \leq i \leq n - 1\), where \(-2\alpha \leq p_i \leq 2\alpha\).
2. Decompose \(p_i\) to transfer \(i_{n+1}\) and interim sum \(w_i\), for \(0 \leq i \leq n - 1\), such that \(-\alpha + 1 \leq w_i \leq \alpha - 1\), \(p_i = w_i + 2^h \times i_{n+1}\), and \(i_{n+1} = -1, 0\), for \(p_i \leq -\alpha, -\alpha < p_i < \alpha\), and \(p_i \geq \alpha\), respectively.
3. Form \(s_i = w_i + i_{n+1}\), for \(0 \leq i \leq n - 1\), and set \(s_n = i_{n+1}\). No new transfer will be generated in this step.

![Fig. 1. The three steps of SD addition (Algorithm 1)](image)

Each of the Steps I and III, of Algorithm 1, involves an \(h\)-bit addition and Step II requires an \(h\)-bit comparison whose complexity is, in general, in the same order as that of an \(h\)-bit addition. It would be desirable to reduce the overall latency roughly to that of two or one \(h\)-bit addition. The representation or encoding of signed digits is greatly influential on the latency of SD addition. Two’s complement encoding of signed digits is believed to lead to the most efficient signed digit addition schemes [8], and is the encoding of choice in all the four designs studied in this paper.

**Example 2** (Decimal Carry-free addition): Consider the decimal SD digit set \([-7, 7]\), where \(\xi = 15 > r = 10\). Fig. 2 illustrates the application of Algorithm 1 on two 4-digit decimal SD numbers.

<table>
<thead>
<tr>
<th>(i)</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_i)</td>
<td>2</td>
<td>3</td>
<td>-5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>(y_i)</td>
<td>5</td>
<td>6</td>
<td>-6</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Step I**

| \(p_i\) | 7 | 9 | -11 | 6 |

**Step II**

| \(w_i\) | -3 | -1 | -1 | 6 |

**Step III**

| \(s_i\) | 1 | -2 | -2 | -1 | 6 |

Fig. 2. A decimal SD addition

### 3. Efficient SD Addition Schemes

We address three previous efficient MRSD addition schemes numbered as a (Fig. 3), b (Fig. 4), and c (Fig. 6). Steps I and III of Algorithm 1 are fused in a and b, in order to simultaneously compute \(P_i = p_1, p_0\), and \(p_{i+1}\). Each scheme, via Step II, derives \(i_{n+1}\), and the three corresponding speculated sum values, in its own way. Note that, in Figs. 3 (5) there are two (one) \(h+1\)-bit operations in the critical delay path. The speculative approach, with three parallel adders as in a and b, is probably the most straightforward approach. However, the nonspeculative scheme c, only computes \(p_i\) and simultaneously extracts \(i_{n+1}\) directly from \(x_i\) and \(y_i\).

a. Fahmy and Flynn [10] have used 2’s complement encoding of the MRSD radix-16 number system
to represent redundant digit floating-point numbers with digit set \([-15, 15]\). The main idea, in their SD adder, is to speculatively compute \(p_i - 1, p_i, p_i + 1\) in parallel for all hexadecimal positions, decompose the primary sum digits to \(16t_{i\alpha}\) and the speculative interim sum values \(w_i - 1, w_i, w_i + 1\), respectively. The transfer digit \(t_i\) selects one of the latter three values as the correct \(s_i\).

b. The SD addition scheme of [8] is based on an alternative treatment of Step II of Algorithm 1, where \(p_i\) is compared to \(2^{h-1}\) instead of \(\alpha\). The corresponding speculative adder architecture is shown by Fig. 4. The rationale for comparison with \(2^{h-1}\) can be understood from Fig. 5, which is primarily showing the valid transfer values based on Algorithm 1 for \(a \leq 2^{h-1}\). This figure also shows two overlapping regions where either of the two corresponding values, whether on the dashed parts or solid parts, can be assumed by the transfer digit. However, the method in [8] only uses the dashed lines for transfer assignment, which is actually a translation of comparison of \(p_i\) with \(2^{h-1}\). This comparison can be done in constant time independent of the value of \(h\).

c. The interim sum \(w_i\) and the transfer \(t_{i\alpha}\) may be expressed directly as functions of \(x_i\) and \(y_i\). However, these functions are hard and inefficient to implement even for moderate values of \(h\) (e.g., eight-input functions for \(h = 4\)). It has been shown in [13] that \(t_{i\alpha}\) can generally be defined as a function of just the most significant bits of \(x_i\) and \(y_i\), except for few cases that may be detected by a moderately simple combinational logic. This architecture is depicted in Fig. 6, where the operation of the lower adder starts as soon as the transfer \(t_i\) is available at a time that is considerably in advance of completion of operation of the upper adder.

Fig. 5. The overlapping regions of valid values for \(t_{i\alpha}\)

In the next section, we follow scheme c, but with some simplifications that lead to further improvements in latency, power dissipation and layout area.

4. Improved SD Addition Scheme

Algorithm 1 requires, as the first step, the actual addition \(x_i + y_i\). However, one may consider \(x_i\) and \(y_i\) as the two components of a carry-save two’s complement encoding of \(p_i\), which is a special case of weighted bit-set (WBS) encoding [15]. It is illustrated, via symbolic/dot notation, in Fig. 7, where posibits (i.e., normal bits) and negabits (i.e., negatively weighted bits) are represented by lowercase letters inside black dots and upper case letters inside white dots, respectively. With this encoding Step I is bypassed.

Fig. 7. Carry-save two's complement representation of the position sum \(p_i\)
The two negabits $X^h$ and $Y^h$ weigh $2^h$, and as such may directly contribute to the value of the transfer $t_{i+1}$, whose weight is also $2^h$. In fact, if we somehow manage to have one positibit and one negabit in position $h$, the bit-pair may collectively represent a valid $t_{i+1}$ in $[−1, 1]$. To arrange this, observe that arithmetic value of the bit collection $\{X^h, X^{-1}, Y^h, Y^{-1}\}$, with respect to position $h−1$, falls within $[−2, 2]$. The same range of values may be represented by an equivalent collection of a positibit in position $h$ and two negabits in position $h−1$, as shown in Fig. 8. Table I shows the details of this transformation, where the target positibit and negabits are represented by primed variables and it is easily seen that $x'_i = X^h_i, \tilde{x}'_i = X^{-1}_i$ and, $Y^{-1}_i = Y^{-1}_i$.

![Fig. 8. Equivalent representation of position sum $p_i$ via the transformation of Table I](image)

### Table 1. Justification of transformation from Fig. 7 to Fig. 8

<table>
<thead>
<tr>
<th>$X^h_i$</th>
<th>$Y^{-1}_i$</th>
<th>$y^{h^{-1}}_i$</th>
<th>Value</th>
<th>$x'^{h^{-1}}_i$</th>
<th>$Y^{h^{-1}}_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 1 0</td>
<td>1 1 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
<td>0 1 1 0</td>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
<td>0 1 1 1</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 0 0</td>
<td>0 1 1 0</td>
<td>0 0 1 0</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>0 1 0 1</td>
<td>0 1 1 1</td>
<td>0 0 1 1</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
<td>0 1 1 0</td>
<td>0 0 1 1</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>0 1 1 1</td>
<td>0 1 1 1</td>
<td>0 0 1 1</td>
<td>0 0 0 1</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

To extract the transfer value, let $\hat{x}_i = X^h_i x^{-2}_i \cdots x^{-h}_i$ and $\hat{y}_i = Y^{-1}_i y^{-2}_i \cdots y^{-h}_i$. Then an immediate conclusion of the arrangement (i.e., bit partitioning) of Fig. 8 would be $\hat{t}_{i+1} = \hat{x}_i - \hat{y}_i$ and $\hat{w}_i = \hat{x}_i + \hat{y}_i$. Unfortunately, however, it turns out that $\hat{t}_{i+1}$ and $\hat{w}_i$ are not always correct. The exceptions, which can be recognized by a flag $\varphi = x^{-1}_i \hat{y}_i \cdots \hat{y}^{h-1}_i \hat{y}_i \hat{x}^{-1}_i$, and the correct $t_{i+1}$ and $w_i$ are listed in Table II. The transfer $\hat{t}_{i+1}$ is corrected by simply subtracting 1 in all the cases that $\varphi = 1$ or $t_{i+1} = \hat{t}_{i+1} - \varphi$. This leads to the following equations.

$$\hat{x}_i = X^h_i \hat{y}_i, \quad Y^{-1}_i = Y^{-1}_i \hat{y}_i \hat{x}^{-1}_i \tag{1}$$

Similarly, the interim sum $\hat{w}_i$ gets corrected by adding $2^h$. This may be effectively done by adding 1 to both $X^{-1}_i$ and $Y^{-1}_i$ in case of $\varphi = 1$. Note that this is always possible, since the value of the latter two negabits in all the correction cases is $−1$. This leads to Eqns. 2 and 3, respectively, where $X^{-1}_i$ and $Y^{-1}_i$ are the sign bits in the carry-save two’s complement representation of $w_i$.

$$\hat{x}_i = \begin{cases} \left(-1 + 1\right) = 0 & \text{if } \varphi = 1 \\ X^{-1}_i & \text{else} \end{cases}$$

$$\hat{y}_i = Y^{-1}_i \varphi_i \tag{3}$$

The transformation from Fig. 7 to Fig. 8 and the latter corrections (i.e., Eqns. 1, 2 and 3) are collectively shown in the first three parts of Fig. 9. The overall delay up to this point is the delay of flag $\varphi$, and a two-input gate in Eqns. 1 to 3. However, since the first $(h−1)$ bits of $x_i$ and $y_i$ remain intact, one may start adding them at time 0 (i.e., when computation of $\varphi$ begins) to compute the first $(h−1)$ bits of $w_i$. The carry out of position $h−2$, a positibit, and the two negabits in position $h−1$ feed the full adder in that position. For proper functioning of this full adder its two negabit inputs and the negabit carry-out should be inverted [16]. The result is shown in the first row of part 4 of Fig. 9. Recalling Eqn. 1, the two most significant bits of part 3 are extracted to form $t_{i+1}$. Moreover, to prepare for the last step, the transfer from position $i$ (i.e., $t_i$) is converted to 2’s complement number $T_i t_i − 1 \cdots t_0$ using the logic of Fig. 10.

![Fig. 9. Digit slice of SD addition in position i](image)

### Table 2. The exceptions for easy extraction of transfer and the corrections

<table>
<thead>
<tr>
<th>$X^h_i$</th>
<th>$Y^h_i$</th>
<th>Range of $x_i$ and $y_i$</th>
<th>$\hat{t}_{i+1}$</th>
<th>$\hat{w}_i$</th>
<th>Exceptions for $(x_i, y_i)$ pair</th>
<th>Correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>$x_i \geq 0, y_i \geq 0$</td>
<td>1</td>
<td>$-2^h + p_i$</td>
<td>$(0, 0), (0, 1), and (1, 0)$</td>
<td>0 $p_i$</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1 0</td>
<td>$x_i \geq 0, y_i &lt; 0$</td>
<td>0</td>
<td>$p_i$</td>
<td>$(0, -2^h + 1)$</td>
<td>$-1 2^h + p_i$</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0</td>
<td>$x_i &lt; 0, y_i \geq 0$</td>
<td>0</td>
<td>$p_i$</td>
<td>$(-2^h + 1, 0)$</td>
<td>$-1 2^h + p_i$</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
<td>$x_i &lt; 0, y_i &lt; 0$</td>
<td>$-1$</td>
<td>$2^h + p_i$</td>
<td>None</td>
<td>None None</td>
</tr>
</tbody>
</table>
Fig. 10. Conversion of the transfer digit to an equivalent 
$h+1$-bit two’s complement number

The last part of Fig. 9 is an illustration of Step III of 
Algorithm 1, which may be implemented using an $h+1$- 
bit two’s complement adder. However, given that no 
new carry would be generated in this step, Eqn. 4 rules 
the most significant bit of the result, where $c_i^t$ is the 
carry into position $h$.

$$S^h = c_i^t \land (W^h \lor T^h) \lor W^h \land T^h$$  \hspace{1cm} (4)

Fig. 11 depicts a digit slice of the overall SD adder 
based on Fig. 9, where the bold line is the critical delay 
path. However, the two full adder chains may be 
replaced by carry look-ahead (CLA) logic, as shown in 
Fig. 12, in order to reduce latency. The required CLA 
to replace the lower full adder chain is a simplified one, 
for the bits of one of the operands are all the same (i.e., 
$T^h = t^{h-1} = \cdots = t^1 = t_i$ as shown in Fig. 10). This leads 
to Eqn. 5, where $c_i^t$ is the carry-into position 1 of the 
$i^{th}$ digit. For large $h$ (e.g., $h = 4k$, $k \geq 2$), a CLA tree 
with simplified group-generate and group-propagate 
paths may be used. Eqn. 6 provides simplified equations for sum bits of digit slice $i$ of the SD adder 
for $h = 4$, where $a = \overline{x_i^{h-1}} \land \overline{Y_i^{h-1}}$ and $b = x_i^{h-1} \land Y_i^{h-1}$. A 
regular implementation of these equations is depicted by 
Fig. 13, where the lower half adder in position zero 
of Fig. 11 and the logic of Fig. 10 are fused for further efficiency. However, in the actual synthesis, gates with 
higher fan-in may be used.

$$c_i^t = t_i \sum_{j=1}^h w_i^j + (t_i + \prod_{j=1}^h w_i^j)c_i^t$$  \hspace{1cm} (5)

$$s_i^0 = w_i^0 \oplus (a \land b)$$

$$s_i^1 = w_i^1 \oplus (a \land \overline{w_i^1} \land b \land w_i^0)$$

$$s_i^2 = w_i^2 \oplus (a \land \overline{w_i^2} \land \overline{w_i^1} \land b \land w_i^0)$$

$$s_i^3 = w_i^3 \oplus (a \land \overline{w_i^3} \land \overline{w_i^2} \land \overline{w_i^1} \land b \land w_i^0)$$

$$s_i^4 = a \land \overline{w_i^4} \land \overline{w_i^3} \land \overline{w_i^2} \land \overline{w_i^1} \land b \land w_i^0 \land W_i^a$$  \hspace{1cm} (6)

5. Synthesis and Simulation Results

SD adders operate in a digit-parallel manner. 
Therefore, for comparison sake, synthesis and 
simulation of one digit-slice of the SD adder leads to 
reasonable performance measures for the whole adder. 
The SD adder of Fig. 11, as modified based on Fig. 12, 
has been checked for correctness by exhaustive test via 
VHDL code of one digit-slice. We consider, for the 
sake of comparison, three previous relevant designs as 
reference, namely [10], [8], [13]. Besides the delay 
imbalance discussed earlier, considerable area/power 
imbalance with respect to [10] and [8] is 
expected due to less active hardware redundancy. 
Some moderate improvement is also expected with 
reference to [13] due to less complexity of transfer 
logic and simplified CLA logic.

To confirm the latter expectations, all the four designs 
were synthesized by Synopsis Design Compiler based 
on TSMC 0.13 µm CMOS technology. In this 
endeavour, with the goal of maximum possible speed, 
we looked for the maximum time constraint that could 
not be met by any of the four synthesized designs. This 
was found to be 0.4 ns. The results are compared in 
Table III, where the 34% less PD (i.e., product of 
delay and power) of the proposed design with respect 
to the best previous one is quite noticeable.
### 6. Conclusions

We examined three previous efficient implementations of maximally redundant signed digit adders. Then, we proposed a new MRSD addition scheme based on carry-save two’s complement encoding of primary sum-digits that are readily available by simply aligning the equally weighted digits of the operands. The first step of conventional SD addition algorithm is thus bypassed. The primary sum digits are partitioned to a transfer part and an interim sum. Whereas this simple partitioning leads to invalid results in few exceptional cases of the operands, a flag is computed to indicate the exceptions and to enforce corrections. Finally, the last step of conventional SD addition (i.e., adding the interim sum digits with the transfer coming from the next less significant digit position) is performed by a simplified carry look-ahead logic.

The new MRSD adder is checked for correctness via exhaustive tests based on VHDL code describing the adder. Synthesis and simulation of the proposed adder shows better performance in terms of delay, power dissipation and layout area in comparison with three previous contributions. Fig. 14, based on the results tabulated in Table III, depicts these advantages.

Research is on going for further performance improvement in SD adders, and use of them in more sophisticated hardware units such as multiplication, division, and floating-point arithmetic circuits. The through benefit of the proposed adder can be better evaluated in realistic benchmark applications of such complex hardware units.

#### Acknowledgment
This research was supported by Shahid Beheshti University under grant # 600/2050.

#### References


