An improved maximally redundant signed digit adder

Ghassem Jaberipur*, Saeid Gorgin

Article info
Article history:
Received 17 July 2008
Received in revised form 23 June 2009
Accepted 9 December 2009
Available online 21 January 2010

Keywords:
Computer arithmetic
Maximally redundant signed-digit number system
Signed digit adder
Weighted bit-set encoding
Inverted encoding of negabits
Logical techniques and design

Abstract
Signed digit (SD) number systems support digit-parallel carry-free addition, where the sum digits absorb the possible signed carries in \((-1, 0, 1)\). Radix-$2^h$ maximally redundant SD (MRSD) number systems are particularly attractive. The reason is that, with the minimal \((h + 1)\) bits per SD, maximum range is achieved. There are speculative MRSD adders that trade increased area and power for higher speed, via simultaneous computation of three sum digits, while anticipating one of the possible signed carries. However, the nonspeculative approach that uses carry-save two's complement encoding for intermediate sum digits, has proved to be more efficient.

In this paper, we examine three previous nonspeculative MRSD adders and offer an improved design with significant savings in latency, area consumption and power dissipation. The enhanced performance is mainly due to the elimination of sign extension of the signed carries. The latter leads to a sum matrix of positively and negatively weighted bits that normally complicate the use of standard adder cells. However, with inverted encoding of negatively weighted bits, we manage to efficiently use such cells. The claimed performance measures are supported by 0.13 \(\mu\)m CMOS technology synthesis.

1. Introduction

Addition is considered as the fundamental operation in digital arithmetic, such that there is normally one or more add operations embedded in the implementation of other arithmetic operations. For example, subtraction is essentially a special case of addition due to the common practice that the radix complement of the subtrahend is added to the minuend. Furthermore, the bulk of multiplication is normally a process of multi-operand addition. Finally, division algorithms are based either on iterative subtractions or converging multiplications [23]. Therefore, any improvement in the addition speed potentially reduces the latency of other digital arithmetic operations.

Carry propagation, throughout the adder cells of conventional ripple-carry adders, slows down the addition process which may be accelerated via using a fast addition technique [28]. However, in the case of conventionally represented numbers (e.g., positional binary or decimal), addition speed is logarithmically bounded by the number of digits [26]. Nevertheless, unconventional number systems provide the possibility of addition with sub-logarithmic and even constant latency. For instance, the wide-word addition operands may be transformed to several short-word residues with respect to the moduli of a residue number system [25]. Such small residues are added in parallel channels with shorter carry chains; hence sub-logarithmic latency. Extremely fast constant-time addition, however, is possible via redundant number systems that support carry-free addition [22].
The conventional carry-free addition algorithm [22] computes the position sum digits by parallel addition of equally weighted digits of the operands. A position sum digit \( p_i \) in position \( i \) is decomposed to an interim sum digit \( w_i \) in the same position, and a transfer digit \( t_{i-1} \) to be carried to position \( i + 1 \). In most practical cases the transfer digit is actually a signed carry in \([-1,0,1]\). The final sum digits are formed via parallel computations \( s_i = w_i + t_i \) for all digit positions. A transfer digit generated in position \( i \), will be absorbed in position \( i + 1 \) and thus will not further propagate to position \( i + 2 \); hence carry-free addition.

The number of digits in the digit set of a redundant number system is more than the radix. For more practical \( 2^h \) radices (e.g., \( h = 4 \) or radix 16), representation of each redundant digit requires more than \( h \) bits. For example, with the minimal \( h + 1 \) bits for each redundant radix-\( 2^h \) signed digit (SD), the digit set can vary from \([-2^{h-1},2^{h-1}]\) with \( 2^h + 1 \) digit values to \([-2^h + 1,2^h - 1]\) with \( 2^{h+1} - 1 \) digit values.

It is well known that conversion of a nonredundant number to a redundant one is possible in constant time that is not dependent on the number of digits, while the reverse conversion requires carry propagation. Therefore, it may appear that the gained speed via a single constant-time addition is lost. However, should a whole computation or a composite operation (e.g., multiplication or division) include many interdependent additions, the redundant sums can be directly used as operands of other operations. The compounded speed gain, due to several constant-time additions, compensates for the latency of final conversion process. Recalling the required extra redundancy bits (at least one for each radix-\( 2^h \) digit), such benefit is merely realized if it is possible to store the intermediate results in widened registers, thus avoiding conversion to a nonredundant representation.

There are various computer arithmetic applications that could use a redundant format to represent intermediate results. For example, typical function evaluation hardware algorithms (e.g., in digital signal processing [21]) are composed of several add/sub operations that are possibly intermixed with more complex multiply/divide operations. Also, CORDIC computation [4], super long word computations in cryptography [27], and even the floating-point units [6,16] can use redundant number systems to enhance speed. Division is a good example of composite operations that is often implemented via a recurrence of subtractions.

Redundant radix-\( r \) signed digit (SD) number systems with digits in \([-\alpha, \alpha]\), where \( \alpha \geq \frac{[2^h]}{2^h} \), were originally introduced by Avezienis [2]. They represent a class of balanced redundant number systems with a good number of applications [7]. The radix-\( 2^h \) maximally redundant SD (MRSD) number systems (\( \alpha = 2^h - 1 \)), in particular, lead to more efficient SD adders [19,6,9,13,14].

In this paper, we present a new MRSD addition scheme based on the redundant weighted bit-set (WBS) encoding [10] of intermediate results within the add operation. This is shown to improve the previous contributions in terms of speed, power dissipation, and layout area. The rest of the paper is organized as follows. A general background on redundant number systems emphasizing MRSD representations is offered in Section 2. The WBS encoding of SD number systems and inverted encoding of negabits [15], as a vehicle for fast and flexible redundant arithmetic, are revisited in Section 3. The main contribution of this paper appears in Section 4. The results of the simulations and 0.13 \( \mu \) CMOS technology synthesis of all the previous works on MRSD addition schemes and that of proposed one, is offered in Section 5. Finally, we draw our conclusions in Section 6.

Table 1 provides a summary of the terms and conventions that are used throughout the paper, where more clarification will be in order on actual use.

### 2. Background

The foremost use of redundant number systems in modern digital processors dates back to 1956 when carry-save adders (CSA) were introduced [5]. In CSA, the result and one of the operands are represented in redundant carry-save format, while

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
<th>Representation</th>
<th>Logical state</th>
<th>Arithmetic value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Symbolic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dot</td>
<td>Mixed</td>
<td></td>
</tr>
<tr>
<td>Posibit</td>
<td>Positively weighted bit</td>
<td>x</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>Negabit</td>
<td>Negatively weighted bit</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mixed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 1 Definitions of terms and conventions.

- **CE**: Conventional bit encoding
- **IE**: Inverted bit encoding
- **Polarity**: Positive for posibits and negative for negabit
- **2CL**: Two’s complement
- **Transfer**: The carry digit of digit parallel addition (called signed carry when in \([-1,0,1]\))
- **MRSD**: Maximally redundant signed digit

---
the other operand is nonredundant. However, if both operands and the result of an arithmetic operation are represented as redundant numbers the operation is called fully redundant [8]. The first fully redundant addition scheme was offered by Aveziens [2], where both operands and the result are represented in a redundant signed digit (SD) number system. The SD number system was, after three decades, generalized by Parhami as generalized signed digit (GSD) number systems [22]. GSD is a fixed radix contiguous number system with radix-\( r \) digit set \( \{C_0, a, b\} \), where \( a, b \geq 0 \). It is deemed redundant if the redundancy index \( \rho \), defined by Eq. (1), is positive and nonredundant if it is zero.

\[
\rho = \alpha + \beta + 1 - r \geq 0, \quad (1)
\]

\[
\rho = 2\alpha + 1 - r \geq 0. \quad (2)
\]

The balanced GSD, with \( \alpha = \beta \), leads to Eq. (2) and coincides with the SD number systems of Aveziens. For more practical values of \( r = 2^h \), Eq. (2) leads to \( \alpha \geq 2^{h-1} \). The minimum value of \( \alpha \) (i.e., \( 2^{h-1} \)), corresponds to the minimally redundant digit set \( [-2^{h-1}, 2^{h-1}] \), where \( \rho = 1 \) and at least \( h + 1 \) bits are required to represent a digit. However, with the same number of bits, \( \alpha \) could be as much as \( 2^h - 1 \) (i.e., nearly twice) corresponding to maximally redundant digit set \( [-2^h + 1, 2^h - 1] \), where \( \rho = 2^h - 1 \). The latter is attractive since it provides maximum range of numbers with minimum number of redundancy bits (i.e., one bit per SD). For example, Fahmy and Flynn [6] have used the radix-16 (\( h = 4 \)) maximally redundant signed digit (MRSD) number system to represent the significand of floating point numbers, where each 5-bit 2CL digit represents digit values in \( [15, 15] \). Fig. 1 depicts a 4-digit carry-free addition in the latter MRSD number system, where digits 10 to 15 are coded as A to F, respectively.

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. The sum digit in position \( i \) for the cases of \( \rho = 0 \) (i.e., nonredundant), \( \rho = 1 \), and \( \rho \geq 2 \) depends on:

- \( \rho = 0 \): In a conventional nonredundant number system the sum digit in each position \( i \) \( (\geq 0) \), in the worst case, is a function of \( 2i + 2 \) operand digits; namely the two operand digits per each of the positions \( i, i - 1, \ldots, 1, 0 \).
- \( \rho = 1 \): For the minimally redundant SD number systems the sum digit in position \( i \) is a function of operand digits in the same position, and also those in positions \( i - 1 \) and \( i - 2 \), where they exist. The constant time addition in this case is called carry-limited [22].
- \( \rho \geq 2 \): In this case of SD number systems, for \( r > 2 \), including the MRSD case of \( \rho = r - 1 \), the sum digit in each position \( i \) \( (\geq 1) \) depends on, no more than, four digits in positions \( i \) and \( i - 1 \) [12]. This constant time addition is called carry-free [22].

Fig. 1. Illustration of a 4-digit radix-16 MRSD addition.

Fig. 2. An illustration of the three steps of SD addition (Algorithm 1).
One drawback of redundant number systems, however, is that in practical cases of \( r = 2^h \) more than \( h \) bits are needed for representation of each digit. For example, consider the \( k \)-digit radix-2 (i.e., \( h = 1 \)) SD number system that employs two bits to represent each digit in \([-1,1]\). The representation range of this number system, utilizing \( 2k \) bits, is \([-2^h + 1,2^h - 1]\). However, the same range is covered by a \( k + 1 \)-bit two's complement representation. This difference in the number of utilized bits, translates to roughly doubling the number of data paths in the radix-2 SD arithmetic circuitry. For higher radix SD number systems (i.e., with \( h \geq 2 \)), however, less overhead on storage and data paths is compromised for slower, yet constant time, arithmetic [9].

A carry-free addition scheme for radix-2\( ^h \) SD number systems, with digit set \([-\alpha,\alpha]\), is described below, where \( \alpha > 2^{h-1} \). The function of this algorithm is also illustrated by Fig. 2.

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

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

*Output:* A \( k \)-digit radix-2\( ^h \) SD number \( S = s_{k-1}, \ldots,s_0 \), where \(-\alpha \leq x_iy_i s_i \leq \alpha\), for \( 0 \leq i \leq k - 1 \).

For \( i = 0 \) to \( k - 1 \) perform the following digit-parallel steps:

I. Compute \( p_i = x_i + y_i \), leading to the \( k \)-digit position-sum \( P = p_{k-1}, \ldots,p_0 \), \( -2\alpha \leq p_i \leq 2\alpha \).

II. Decompose \( p_i \) to transfer \( t_i \) and interim sum \( w_i \), such that \(-\alpha + 1 \leq w_i \leq \alpha - 1\), \( p_i = w_i + 2^h t_{i+1} \), and \( t_{i+1} = -1, 0, \) and \( 1 \) for \( p_i \leq -\alpha, -\alpha < p_i < \alpha \), and \( p_i \geq \alpha \), respectively.

III. Compute \( s_i = w_i + t_i \), where no new transfer is to be generated by this final addition.

Steps I and III involve an \( h \)-bit addition. Step II requires an \( h \)-bit comparison whose complexity is in the same order as that of one \( h \)-bit addition. Therefore, a simple implementation of Algorithm 1 would lead to a fairly slow carry-free adder. However, the encoding of signed digits, and the value of \( \alpha \), significantly influence the complexity and can be manipulated to simplify and speed-up some of the steps above. For example, two’s complement representation of signed digits is believed to lead to the most efficient signed digit addition schemes [9]. That’s why it is the encoding of choice in all the six designs discussed in this paper. Likewise, the case of MRSD is attractive due to the potential for faster SD addition. The supporting intuition is partly due to the minimum number of invalid digit-values within those supported by the two’s complement encoding (i.e., only \(-2^h \) is invalid).

The previous implementations of MRSD adders can be categorized based on speculation of sum digits:

- **Speculative:** In this approach the steps of Algorithm 1 are parallelized via extensive hardware redundancy. For example, the speculative MRSD addition schemes of [6,9] precompute three sum-digits corresponding to three possible transfer values as abstractly depicted in Fig. 3. The actual sum digit in each position is selected by the transfer value coming from the next lower position. The main difference between these two schemes is on how the actual transfer value is computed. The adder in [6] directly implements Step II of Algorithm 1, where the transfer logic has \( h + 4 \) inputs. However, Step II in [9] compares the position sums with \( 2^{h-1} \) instead of \( \alpha = 2^{h-1} - 1 \). This technique has led to simpler transfer logic with only three inputs.

- **Nonspeculative:** In this approach, the steps of Algorithm 1 are combined such that the sum digits are directly computed as \( s_i = x_i + y_i + t_i - 2^h t_{i+1} \), where a fast combinational logic computes the transfer values from the bits of the digits of the two operands. For example, the MRSD adders of [13,14] use the most significant bits of the digits of the two operands \( x_i \) and \( y_i \) for an estimation of the transfer value and use the rest of the bits to compute a correction flag. The respected architectures, for \( h = 4 \) (i.e., radix-16), are depicted in Figs. 4 and 5 respectively, where the main difference is that in Fig. 5 a simplified carry look-ahead adder replaces the FA chain of the second row. However, the nonspeculative design

![Fig. 3. A radix-2^h digit-slice of the speculative MRSD adder of [6,9].](image-url)
in [19] uses no correction flag and only the two most significant bits from each operand to compute the transfer value as $T_{i}^{1} = 2^{T_{i}^{0}},$ where $T_{i}^{0}$ is a negatively weighted bit (negabit) and $T_{i}^{0}$ is a normal bit (posibit). Then the adder starts the main sum computation. Fig. 6 depicts an instance of this MRSD adder for $h = 4$ (i.e., radix-16), where the control logic computes $t_{i+1}$ to be used for computation of $s_{i+1}$ and injects the negative term $-2^{T_{i}^{0}}$ into the computation of $s_{i}$. One drawback of the latter three nonspeculative designs, for large $h$, is the high fan-out due to the negabit component of the transfer bit (i.e., $T_{i}^{0}$).

A rough comparison among Figs. 3–6 show that the latency of all addition schemes is around the delay of $h + 1$ cascaded full adders, where the critical delay path in Figs. 4–6 is marked by a thick gray curve. However, the hardware complexity of the speculative ones is mainly due to three $(h + 1)$-bit adders versus the equivalent of roughly two $h$-bit adders for the nonspeculative ones.
3. Weighted bit-set encoding of redundant numbers

In the conventional weighted-position number systems there is exactly one radix-$r$ digit per each position, where the digit in position $i$ weighs $r^i$. However, some encodings for redundant number systems allow for more than one digit in the same position. For example, the binary carry-save number system often uses two equally weighted bits per binary positions to represent the digit set $[0,2]$. As another example, consider that a binary signed digit (BSD) may be represented by a pair of equally weighted negabit and posibit vis-à-vis the borrow-save or $(n,p)$ encoding [22]. The decimal signed digit of [20], as a high-radix example, uses equally weighted positive and negative BCD digits to represent $[-9,9]$. Finally, a radix-16 carry-save two's complement digit in $[-16,14]$ is composed of two equally weighted two's complement digits in $[-8,7]$. Recalling the conventions of Table 1, for the symbolic representation of posibits and negabits, Table 2 provides the symbolic notations, dot notations and numerical examples for the latter four examples.

The binary carry-save and borrow-save encodings have been unified and generalized as the weighted bit-set (WBS) encoding [10]. In a WBS-encoded number, a binary position may hold any number of posibits and negabits. For example, the second and third columns of Table 2 include the symbolic and dot notation representation of the WBS encodings of the represented examples, respectively. Additionally, the WBS encoding can also faithfully represent nonredundant digit sets such that all the possible bit assignments represent valid digits. For example, see the last row of Table 2 for the faithful 4-2-2-1 representation of decimal digits in $[0,9]$, where the encoding is redundant (e.g., 4 can be represented as 1000 or 0110) though the digit set is not.

To see some actual application of WBS representation consider that Step I of Algorithm 1 may be implemented by equal weight aligning of the bits of $x_i$ and $y_i$ that leads to carry-save two's complement representation of $p_i$. This cost-free operation actually leads to an instance of a 2-deep (i.e., at most two bits in any weighted position) WBS encoding. This is shown in Fig. 7 using a mixed dot/symbolic notation.

Given this WBS representation of $p_i$, Steps II and III may be considered as an instance of digit-set conversion [17], where a number with digits $p_i \in [-2x,2x]$ is converted to an arithmetically equivalent number with digits $s_i \in [-x,x]$. There may be several ways for implementation of such digit-set conversion. For example, the MRSD adder in [19] does not directly follow the Steps of Algorithm 1. Instead it undertakes the following:

- Extracting $t_{i+1}$ via a 4-input (i.e., the two MSBs of $x_i$ and $y_i$) combinational logic.
- Forming a WBS encoding of $w_i$ mainly by direct wiring of selected bits of $x_i$ and $y_i$.
- Fast addition of $w_i + t_i$.

The 2-deep WBS representation of $w_i$ may be augmented by a WBS representation of $t_i$ to obtain a 3-deep (Fig. 8a and c) or 4-deep (Fig. 8b) WBS encoding of the sum-digit $s_i$. In both cases the final sum-digit generation is a question of digit set conversion to transform a WBS representation of the sum-digit to an equivalent two's complement digit.

The MRSD adder of [19] uses $h$ bits to represent $t_i$ as a 1-deep two's complement number (Fig. 8c). However, given that $t_i \in [-1,1]$, a 2-bit two's complement representation, such as $T_i^1$, $T_i^0$ in Fig. 8a, would do and intuitively leads to less area and energy consumption. Nevertheless, [19] has opted for an $h$-bit representation. Otherwise, as may be seen in Fig. 8a and b, a mix of equally weighted posibits and negabits would be difficult to handle.

![Fig. 7. Carry-save two's complement representation of the position sum $p_i$.](image)
The common practice for digit-set conversion of a restricted WBS representation (i.e., negabits allowed only in the most significant positions) such as the one in Fig. 8c, uses standard counters and compressors (e.g., (3, 2) counters and (4; 2) compressors). This would reduce the original WBS number to an equivalent 2-deep WBS number, which should be followed by a digit-wide addition to lead to the final sum digit. However, the problem with an unrestricted WBS encoding (i.e., with mixed equally weighted negabits and posibits), such as those in Fig. 8a and b, is that the standard cells do not accept a mixture of opposite polarity bits. This problem has found a special-case preliminary solution in [24] within the context of two’s complement array multipliers. A general solution in [1] uses a family of full adders that may be distinguished by additional input/output inverters. Finally the (4; 2) compressors of Kornerup [18], for reduction of an equally weighted 4-deep mix of posibits and negabits, are basically made of two Aoki full adders. However, the latest solution offered in [11] uses inverted encoding of negabits that, exactly opposite to the conventional encoding, associates arithmetic value $-1$ and $0$ to logical states $0$ and $1$, respectively. This allows any binary counter or compressor to directly (i.e., with no inverting interface) accept any mixture of negabits and posibits. Fig. 9a–c illustrate the concept, with all possible I/O combinations, for full adders, half adders, and (4; 2) compressors, respectively.

4. The new MRSD adder

In this section we use the following additional conventions besides those in Table 1:

- Uppercase and lowercase versions of the same variable assume the same logical value.
- Inverted encoding of negabits is in effect.

Let the position sum-digit $p_i$ be represented by a carry-save two’s complement digit as was depicted in Fig. 7. The most significant negabits $X^h_i$ and $Y^h_i$ weigh $2^h$ as does the transfer $t_{i+1}$. Intuitively, one of the negabits (say $X^h_i$) may serve as the negabit component of an $(n, p)$ encoding of $t_{i+1}$ (see Fig. 10). However, we also need to extract a posibit from position $h$. One way, to do this, is to transform the carry-save two’s complement representation of $p_i$ to an appropriate equivalent WBS encoding based on the following theorem.

**Theorem 1 (Inverted polarity transformation).** The arithmetic value of a carry-save two’s complement number with inverted encoding of negabits is preserved by inverting the polarities of all the constituent bits of one of the two’s complement components and that of the least significant bit of the other.

**Proof.** Let $N$ and $p$ denote a negabit and a posibit that are to be regarded as posibit $n$ and negabit $P$. Then the following holds:

$$\|n\| = n = N - 1 + 1 = |N| + 1, \quad \|P\| = p - 1 - 1 = |p| - 1.$$  \hspace{1cm} (3)

As illustrated in Fig. 10 and supported by Eq. (3), converting $X^h_i$ to $x^h_i$ adds $2^h$ to the arithmetic value of $p_i$. The exact balancing effect is exercised by inverting the polarity of the two posibits in position zero, and a posibit in each of the positions 1 to $h - 1$. This, by Eq. (3), will deduct $2^h$ from the value of $p_i$ that levels the net effect of the overall transformation. □
The immediate and cost-free transformation of Fig. 10 leaves a negabit/posibit pair in position $h$. This pair may serve as the transfer to the next digit position (i.e., $t_{i+1}$), except for a few instances that the immediate interim sum $\bar{W}_i$ violates the desired range $[-2^h + 2, 2^h - 2]$. There are only five of these cases, out of $(2^h + 1)^2$ possible $(x_i, y_i)$ value-pairs. These five exceptions are listed as bold entries of Table 3, where $\bar{x}_i, \bar{y}_i, t_{i+1}$, and $\bar{W}_i$ are defined as:

$$
\begin{align*}
\bar{x}_i &= x_i^{-1}, ..., x_i^{0}, \\
\bar{y}_i &= y_i^{-1}, ..., y_i^{0}, \\
\bar{W}_i &= |\bar{x}_i| + |\bar{y}_i| = 2^h (|x_i^{h}| + |y_i^{h}|) - 1)
\end{align*}
$$

The rationale for each exception is explained below:

1. $x_i^h = 0, y_i^h = -1$: In this case the following holds, where the constant negabit `0 in position $h$ worth $-2^h$.

   $$-2^h + 1 \leq |x_i| = -|0x_i^{-1}, ..., x_i^{0}| \leq -1.$$ (5)

   The left inequality leads to $|x_i^{h-1}, ..., x_i^{0}| \geq 1$. Therefore,

   $$|\bar{x}_i| = |x_i^{h-1}, ..., x_i^{0}| = |x_i^{h-1}, ..., x_i^{0}| - 1 \geq 0$$

   (6)

   Likewise, $|1y_i^{h-1}, ..., y_i^{0}| \geq 0$ and $|\bar{y}_i| = |y_i^{h-1}, ..., y_i^{0}| = |y_i^{h-1}, ..., y_i^{0}| - 2^h + 1$ lead to $|\bar{y}_i| \geq 1 - 2^h$. This and the exception $\bar{W}_i = |\bar{x}_i| + |\bar{y}_i| = 1 - 2^h$ enforce $|x_i| \leq 0$. The latter and $|X_i| \geq 0$ from (6) can be simultaneously satisfied only for $|x_i| = 0 = |0, ..., 0|$, which leads to $|\bar{y}_i| = 1 - 2^h = |0, ..., 0|$. (2)

2. $x_i^h = -1, y_i^h = 0$: This case can be similarly elaborated to lead to $|\bar{x}_i| = -1 = |0, ..., 0|$, as the only valid value for $\bar{x}_i$, which further arrives at $|\bar{y}_i| = 2 = -2^h$. (3)

3. $x_i^h = -1, y_i^h = 1$: $0 < |1x_i^{h-1}, ..., x_i^{0}| \leq 2^h - 1 \Rightarrow |x_i^{h-1}, ..., x_i^{0}| \geq 0$. Likewise $|y_i^{h-1}, ..., y_i^{0}| \geq 0$. The following holds via polarity inversion by Theorem 1:

   $$|\bar{x}_i| = |1x_i^{h-1}, ..., x_i^{0}| \geq -1, |\bar{y}_i| = |1y_i^{h-1}, ..., y_i^{0}| \geq 2^h + 1.$$ (7)

There are two exceptions for $\bar{W}_i$ that lead to three other bold cases in Table 3:

- (3.1) $\bar{W}_i = -2^h$: $|\bar{x}_i| = \bar{W}_i - |\bar{y}_i| \leq -2^h \leq -2^h + 1 = -1$. This and $|\bar{X}_i| \geq -1$, from (7), enforce $|x_i| = 1$.

- (3.2) $\bar{W}_i = -2^h + 1$: $|\bar{x}_i| = \bar{W}_i - |\bar{y}_i| \leq -2^h + 1 \leq -2^h + 1 = 0$. This and $|\bar{X}_i| \geq 1$, from (7), enforce $|x_i| = -1$.

Table 3

<table>
<thead>
<tr>
<th>$X_i^h$</th>
<th>$Y_i^h$</th>
<th>$\bar{x}_i$ and $\bar{y}_i$</th>
<th>$t_{i+1}$</th>
<th>$\bar{W}_i$</th>
<th>$\bar{X}_i$ and $\bar{Y}_i$</th>
<th>$t_i$</th>
<th>$\bar{W}_i$</th>
<th>$\bar{X}_i$ and $\bar{Y}_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-1, 0$</td>
<td>$x_i, y_i \leq -1$</td>
<td>$2^{-(n+1)} - 1$</td>
<td>$-2$</td>
<td>$-1$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$0, 1$</td>
<td>$x_i \leq 0, y_i &gt; 0$</td>
<td>$2^{-2} - 1$</td>
<td>$0$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$1 - 2^{-(n+1)} - 1$</td>
<td>$0$</td>
<td>$1 - 2^{-(n+1)} - 1$</td>
</tr>
<tr>
<td>$-1, 0$</td>
<td>$x_i &gt; 0, y_i \leq -1$</td>
<td>$0$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$1$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
</tr>
<tr>
<td>$-1, -1$</td>
<td>$x_i, y_i \geq 0$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$1 - 2^{-(n+1)} - 1$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
<td>$2^{-(n+1)} - 2$</td>
</tr>
</tbody>
</table>

**Example 1 (Regarding a negabit as a posibit):** A negabit, in position $i$, with logical state 0 or 1 has the arithmetic worth of $-2^i$ or 0, respectively. The arithmetic worth of this negabit when regarded as a posibit is 0 or $2^i$, respectively. 

*Fig. 10. Inverted polarity transformation.*
Note that, in all the five exceptions in Table 3, we have $|t_{i+1}| = t_{i+1} - 1$ and consequently $|w_i| = 2^h + \hat{w}_i$. Note also that the case of $X^h = Y^h = 0$ is not an exception since it leads to $-2^h + 2 \leq \hat{w}_i \leq 2^h - 2$.

Both $t_{i+1}$ and $\hat{w}_i$ are subject to correction due to the exceptions, enumerated above, corresponding to invalid values for $\hat{w}_i$ (i.e., $|\hat{w}_i| \geq 2^h - 1$). We define a flag $\phi_i$ to indicate whether the $(x_i,y_i)$ pair is an exception. The logical equation for this flag can be derived, as in Eqn. 8a, via a straightforward truth table. In all other cases, where $\phi_i = 0$, we have $|t_{i+1}| = t_{i+1}$ and $|w_i| = \hat{w}_i$. This leads to Eqs. (8b) and (8c).

$$
\begin{align*}
\phi_i &= x_ih - 1 \lor \cdots \lor x_i^h \land y_ih - 1 \lor \cdots \lor y_i^h \land x_i^0 \land y_i^0; \\
|t_{i+1}| &= t_{i+1} - \phi_i; \\
|w_i| &= 2^h \phi_i + \hat{w}_i.
\end{align*}
$$

(8a) (8b) (8c)

Recall the arrangements of Fig. 8 regarding the different encodings for $t_{i+1}$. It is easy to build a truth table for the negabit/posibit components of $t_{i+1}$ as a function of $X^h_i, Y^h_i$ and $\phi_i$. This leads to Eqs. (9a) and (9b) corresponding to Fig. 8a and b, respectively.

$$
\begin{align*}
T^1_{i+1} &= \overline{\phi_i} \land (x_i^h \lor Y^h_i) \lor (x_i^h \land Y^h_i). \\
T^0_{i+1} &= x_i^h \land Y^h_i, \\
\hat{t}_{i+1} &= \overline{\phi_i} \land (x_i^h \lor Y^h_i).
\end{align*}
$$

(9a) (9b)

To correct $\hat{w}_i$ to $w_i$, as ruled by Eq. (8c), the key observation is that adding $2^h$ to $\hat{w}_i$, in case of $\phi_i = 1$, is equivalent to incrementing the arithmetic values of $x_i^{h-1}$ and $Y_i^{h-1}$. Therefore, the bits in positions 0 to $h - 2$ of binary representation of $\hat{w}_i$ remain intact, and only $x_i^{h-1}$ and $Y_i^{h-1}$ turn to $x_i^{h-1}$ and $Y_i^{h-1}$ as in Eq. (10). This can be verified by careful examination of the bold entries of Table 3.

$$
X_i^{h-1} = x_i^{h-1} \lor \phi_i, \\
Y_i^{h-1} = Y_i^{h-1} \lor \phi_i.
$$

(10)

Since $T^0_{i+1}$, the negabits component of the transfer in (9b), does not depend on $\phi_i$, the arrangement of Fig. 8b is likely to lead to a faster adder.

The new MRSD addition scheme is illustrated in Fig. 11 with mixed symbolic and dot notation. The top level illustrates the carry-save two's complement representation of $p_i$ defined in Algorithm 1. The next level includes the transfer $t_i$ (i.e., the 2-deep rightmost box) coming from the next lower position, and $t_{i+1}$ (i.e., the 2-deep leftmost box) that is similarly sent to the next higher position. The $h$-long 2-deep box contains the bits of $w_i$. There are also $h - 1$ negabits added in positions 1 to $h - 1$, with logical state 1 and arithmetic value 0, where inverted encoding of negabits is in effect. Each 3-bit column in positions 0 to $h - 1$ constitutes the inputs to a full adder that results in a $(u, v)$ pair in the next level of Fig. 11. A standard enforced-carry-in adder produces the final two's complement sum digits as shown in the last level. However, note that an AND gate, instead of a full adder is all we need in position $h$. This can be easily verified via a truth table for $S_i^h$ in terms of $U_i^h$ and $C_i^h$ (i.e., the negabit carry-in), where the latter two negabits cannot be simultaneously $-1$ (i.e., logical $'0'$).

![Fig. 11. The new MRSD addition scheme.](image-url)
The full adders of the second level may be reduced to pseudo half adders ruled by Eq. (11). These are similar to \(N\)-cells of [3] except that, due to inverted encoding of negabits, no input/output inverters are needed here.

\[
v_j^i = x_j^i \oplus y_j^i, \quad U_{j+1}^i = x_j^i \lor y_j^i \quad \text{for } 1 \geq j \geq h - 1.
\] (11)

Two numerical examples corresponding to Fig. 11, for \(\varphi_i = 0\) and \(\varphi_i = 1\), are given in Fig. 12a and b. Furthermore, a high level implementation of Fig. 11, using standard full adders and pseudo half adders (\(HA^*\)), is depicted in Fig. 13. The leftmost lower cell is reduced to an AND gate, since no carry is allowed in the final sum-digit computation (see Algorithm 1). The full adder chain along positions 1 to \(h - 1\) can be replaced by any carry-accelerating adder (e.g., carry look-ahead [23]) for more speed.

5. Correctness and comparison

Recalling the discussion, in Section 2, on the previously reported MRSD adders we compare the performance of the proposed adder (Fig. 13) with the pioneering nonspeculative adder of [19] (Fig. 6) and our two recently published works [13,14] (Figs. 4 and 5). We will also show that these varieties of nonspeculative adders outperform the speculative designs of [6,9].

![Fig. 12. MRSD addition of radix-16 numbers.](image)

![Fig. 13. A radix-2^h digit-slice of the new MRSD adder.](image)
The correctness of all the six adders is checked by running exhaustive tests (for \( h = 4 \)) on the corresponding VHDL codes, where additional speed is achieved via replacing the full adder chains of Figs. 3–6 and 13 by carry look-ahead blocks. Furthermore, we have used these codes to synthesize the six MRSD adders by Synopsys Design Compiler using a target library based on TSMC 0.13 \( \mu m \) process. To accomplish realistic driving and loading logic, one and four inverters were used, respectively. For dynamic and leakage power measurement, Synopsys Power Compiler has been used.

The synthesis was undertaken for a range of time constraints from 0.5 to 1.0 ns. Table 4 summarizes the synthesis results for the time constraint 0.5 ns that is the longest not met by any of the six designs. However, the experienced latencies that are naturally more than 0.5 ns, are shown along the corresponding other measures. Note that all the figures, in Table 4, relate to one radix-16 digit slice. Similar results for time constraints 0.6 ns, 0.7 ns, 0.8 ns, 0.9 ns and 1.0 ns are compiled in the curves of Fig. 14. Note that each curve stops where the corresponding design has not met the relevant time constraint.

Based on the ratios in Table 4, the latency, dynamic power, static power and area of the new proposed MRSD adder is 10%, 30%, 16% and 24% less than those of the corresponding best previous works (i.e., [14] for latency and power, and [19] for area). The speed gain is partly due to use of an AND gate, in Fig. 13, in place of the last full adder in the chain of the second row of the other circuits (Figs. 4–6). Also the latency of the transfer logic is less than those in [13,14]. The 24% less area is mainly due to using half adders in the first row of the proposed adder instead of full adders in [19]. This has been possible via elimination of sign extension in applying the negative component of the transfer digit in the proposed design.

**Table 4**

<table>
<thead>
<tr>
<th>Design name</th>
<th>Delay (ns)</th>
<th>Ratio</th>
<th>Power consumption</th>
<th>Area (( \mu m^2 ))</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dynamic (mW)</td>
<td>Static (pW)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ratio</td>
<td>Ratio</td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td>0.862</td>
<td>1.48</td>
<td>1.61</td>
<td>2.10</td>
<td>1.89</td>
</tr>
<tr>
<td>[9]</td>
<td>0.769</td>
<td>1.32</td>
<td>2.10</td>
<td>2.73</td>
<td>1.97</td>
</tr>
<tr>
<td>[19]</td>
<td>0.671</td>
<td>1.15</td>
<td>1.01</td>
<td>1.34</td>
<td>1.30</td>
</tr>
<tr>
<td>[13]</td>
<td>0.679</td>
<td>1.16</td>
<td>1.21</td>
<td>1.59</td>
<td>1.46</td>
</tr>
<tr>
<td>[14]</td>
<td>0.641</td>
<td>1.10</td>
<td>1.21</td>
<td>1.30</td>
<td>1.16</td>
</tr>
<tr>
<td>Proposed</td>
<td>0.584</td>
<td>1.00</td>
<td>0.77</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

![Fig. 14](imageLink)
6. Conclusion

We have examined three non-speculative [19,13,14] and two speculative [6,9] maximally redundant signed digit adders. The speculative designs aim at trading increased area and power dissipation for higher speed. However, the three inspected nonspeculative designs show improved performance in latency, area and power. The pioneering nonspeculative design of [19] uses the two most significant bits of the two \( (h+1) \)-bit operands for fast computation of the transfer digit in \([-1,1]\).

This digit, represented as an \( (h+1) \)-bit two’s complement number, takes part in the main sum computation.

The \( (h-1) \) extra sign-extension identical bits, and the corresponding fan-out, lead to superfluous area and energy consumption. However, the other two recent designs and the proposed MRSD adder use only one bit of each operand to provisionally encode the transfer digit. The transfer digit determined as such is shown to be correct except for few exceptional cases that are distinguished and corrected via a computed flag that is a function of all the bits of the two operands. In the proposed design, coexistence of posibits and negabits in the same weighted positions of the WBS representation of the sum digit leads to potential complexities. However, they are shown to be disentangled through inverted encoding of negabits that allows for equal conduct of posibits and negabits when using standard adder cells. It leads to taking advantage of an array of pseudo half adders (Fig. 13) instead of full adders (Figs. 4–6). This brings forward the potential for less area and energy consumption. It was also shown that the speed gain, in the proposed design, is partly due to the faster computation of the transfer digit and replacing the last full adder by an AND gate (Fig. 13). The claimed advantages (i.e., 10% less latency, 24% less area, and 30% less dynamic power for \( h = 4 \)) were supported by 0.13 μm CMOS technology synthesis.

Further research is on going for additional performance improvements in MRSD adders and possible integration of these adders in complex arithmetic units such as floating-point add/subtract circuits.

Acknowledgement

The authors wish to thank the anonymous reviewers for their constructive comments. This research was supported in part by IPM under Grant CS1388-2-03, and in part by Shahid Beheshti University.

References