Constant-Time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations

Dhananjay S. Phatak, *Member*, *IEEE*, Tom Goff, and Israel Koren, *Fellow*, *IEEE*

Abstract—It is well-known that constant-time addition, in which the execution delay is independent of operand lengths, is feasible only if the output is expressed in a redundant representation. There are many ways of introducing redundancy and the specifics of the redundant format employed can have a major impact on the performance of constant-time addition and digit set conversion. This paper presents a comprehensive analysis of constant-time addition and simultaneous format conversion. We consider full as well as partially redundant representations, where not all digit positions are redundant. The number of redundant digits and their positions can be arbitrary, yielding many possible redundant representations. Format conversion refers to changing the number and/or position of redundant digits in a representation. It is shown that such a format conversion is feasible during (i.e., simultaneous with) constant time addition, even if all three operands (the two inputs and single output) are represented in distinct redundant formats. We exploit "equal-weight grouping" (EWG), wherein bits having the same weight are grouped together to achieve the constant-time addition and possible simultaneous format conversion. The analysis and data show that EWG leads to efficient implementations. We compare VLSI implementations of various constant-time addition cells and demonstrate that the conventional 4:2 compressor is the most efficient way to execute constant time-addition. We show interesting connections to prior results and indicate possible directions for further extensions.

Index Terms—Redundant representations, constant-time addition, simultaneous format conversion, redundant adders, carry-save addition, signed-digit addition, 4:2 compressor.

1 Introduction

A positional radix-β number system represents an n-digit value V as a string of digits, $(d_{n-1}, d_{n-2}, \cdots d_0)$, where

$$\sum_{i=0}^{n-1} d_i \cdot \beta^i = V.$$

The value that each digit, d_i, can assume is determined by the digit set for that position, $\mathcal{D}_i$, such that $d_i \in \mathcal{D}_i$. In conventional representations, the digit set is the same for all positions and is defined by $\mathcal{D} = \{d | 0 \leq d \leq \beta - 1\}$. A number system is redundant if there is some value which does not have a unique representation. In other words, a given number system is redundant if there exists an n-digit value V which satisfies

$$V = \sum_{i=0}^{n-1} d_i \cdot \beta^i = \sum_{i=0}^{n-1} d'_i \cdot \beta^i, \quad d_i, d'_i \in \mathcal{D}_i,$$

and there is at least one position j where $d_j \neq d'_j$. This implies that, for some digit position k, the cardinality of the digit set $\mathcal{D}_k$ satisfies $|\mathcal{D}_k| > \beta$. We call such a position, a *redundant digit position*.

Addition can be thought to be an instance of the digit set conversion problem [2], [3], [4], [5]. In this context, we consider the addition of two operands X and Y yielding the result $Z = X + Y$. The digit set $\mathcal{D}_i^x + \mathcal{D}_i^y$ can be thought of as the input digit set for position i and $\mathcal{D}_i^z$ as the output digit set. Addition is then the operation of converting from one digit set to another. In most cases, the range of $\mathcal{D}_i^x + \mathcal{D}_i^y$ is larger than $\mathcal{D}_i^z$, making carry propagation necessary. This results in the carry relationship

$$\beta \cdot c_i + z_i = x_i + y_i + c_{i-1}, \tag{1}$$

where c_{i-1} is the carry-in to position i, c_i is the carry-out, and both are members of a carry set $\mathcal{C}$. An alternate treatment of addition based on digit set operations can be found in [1], which provides a framework for designing adders based on contiguous sets.

1.1 Constant-Time Addition

Constant-time addition is possible at a redundant digit position i if the value of c_i can be determined by considering only a fixed number of previous input digits, making it independent of c_{i-1}. The number of previous digits required constitutes a right context, or look-back [2], [3], [4], and is henceforth denoted by $\mathcal{L}$. The operation of constant-time addition at redundant digit positions can be explained conceptually as the two-step process described below [6].

- *D.S. Phatak and T. Goff are with the Computer Science and Electrical Engineering Department, University of Maryland, Baltimore County, Baltimore, MD 21250. E-mail: {phatak, tgoff1}@umbc.edu.*
- *I. Koren is with the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003. E-mail: koren@euler.ecs.umass.edu.*

Manuscript received 11 Apr. 2000; revised 27 Feb. 2001; accepted 23 Apr. 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111880.

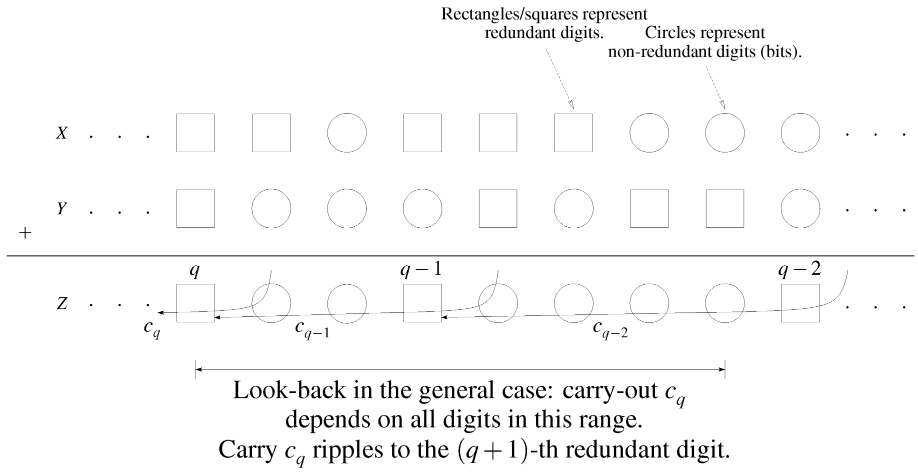

Fig. 1. Constant-time addition of partially redundant operands: Carries propagate *in parallel* between consecutive redundant digits of the *output*.

Step 1: Based on its fixed right context, every redundant digit position generates an intermediate sum, σ_i, and an intermediate carry-out, c_i, where

$$\beta \cdot c_i + \sigma_i = x_i + y_i = \theta_i. \tag{2}$$

In other words, (2) expresses the sum of the operand digits $\theta_i = x_i + y_i$ as the pair (c_i, σ_i).

Step 2: The final sum z_i is formed by $z_i = \sigma_i + c_{i-1}$, where $z_i \in \mathcal{D}_i^z$.

If there are nonredundant digit positions in the result Z, carries must ripple through them [7], [8] and they are determined by (1).

As described in [2], [3], [4], the carry-out of a digit position can depend on the input operands and the output digit set at that position, as well as the operands and output digit sets at all digit positions that fall within fixed-length right and left contexts. If a left context is actually used, the carry into some position can be dependent on the input operands at that position.

1.2 Radix-2 Redundant Representations

Since radix-2 representations are the most commonly used, this paper concentrates only on those representations based on underlying radix-2 digit sets. As specific examples, we consider redundant digit sets that are variants of the well-known signed-digit (SD) and carry-save (CS) representations. These digit sets are defined as

$$\mathcal{D}^{(SD)} = \{-1, 0, 1\} \qquad \mathcal{D}^{(CS2)} = \{0, 1, 2\} \tag{3}$$

$$\begin{aligned} \mathcal{D}^{(SD3^{(-)})} &= \{-2, -1, 0, 1\} \quad \mathcal{D}^{(SD3^{(+)})} = \{-1, 0, 1, 2\} \\ \mathcal{D}^{(CS3)} &= \{0, 1, 2, 3\}. \end{aligned} \tag{4}$$

We consider two types of number systems based on each of these digit sets, a fully redundant system and a partially-redundant system. A fully redundant number system is one in which all digit positions of a number are redundant and the characteristics of such systems are well-known [9], [10], [11], [12]. Implementations of adders for fully redundant representations have also been widely investigated; a sampling can be found in [8], [13], [14], [15], [16], [17], [18].

In a partially redundant number system, only some digit positions are redundant [7], [8]. In this paper, redundant digit positions are indicated by rectangles or squares (see Fig. 1) and digits in these positions can assume any of the values from one of the sets listed in (3)-(4) above. It is possible to use different redundant digit sets (from the above list) at different positions, but, for the sake of simplicity, we restrict ourselves to representations where all redundant digit positions have the same digit set. *Format conversion* therefore refers to changing the number of redundant digits and their positions in a representation, while retaining the same digit set at each redundant position.

Nonredundant digits are represented by circles and require only a single bit to encode the possible values which they can assume, namely $\{0, 1\}$. Some possible partially redundant formats are illustrated in Fig. 1. As the figure shows, redundant digits can be placed at arbitrary positions.

Note that a redundant binary digit needs at least two bits to represent it. In fact, all of the redundant digit sets listed above need exactly two bits to represent their digit values. In essence, we consider redundant representations where some digit positions are allocated two bits and ask the question: Given this basic redundancy, which number representations lead to the most efficient implementations and best exploit the available redundancy?

To answer this question, we consider the number representations listed in Table 1. Among the partially redundant systems, we consider those where every kth digit is redundant (i.e., the digits at positions $k-1, 2k-1, \cdots$, are redundant). Consequently, the partially redundant systems which we consider are denoted SD_k, $SD3^{(\pm)}\_k$, $CS2\_k$, and $CS3\_k$. Note that these representations with uniform distance between redundant digits are equivalent to high-radix (2^k) redundant representations. However, the addition methods presented herein lead to better *implementations* than those that can be derived simply by assuming high-radix arithmetic. Furthermore, formats with nonuniform distances between redundant digits cannot be considered

TABLE 1
Redundant Radix-2 Number Systems

Number System	Description
SD	Digits at all positions $\in \mathcal{D}^{(SD)}$
SD_k	Every k-th digit $\in \mathcal{D}^{(SD)}$; all others $\in \{0,1\}$
$SD3^{\langle\pm\rangle}$	Digits at all positions $\in \mathcal{D}^{(SD3^{\langle\pm\rangle})}$
$SD3^{\langle\pm\rangle}\_k$	Every k-th digit $\in \mathcal{D}^{(SD3^{\langle\pm\rangle})}$; all others $\in \{0,1\}$
$CS2$	Digits at all positions $\in \mathcal{D}^{(CS2)}$
$CS2\_k$	Every k-th digit $\in \mathcal{D}^{(CS2)}$; all others $\in \{0,1\}$
$CS3$	Digits at all positions $\in \mathcal{D}^{(CS3)}$
$CS3\_k$	Every k-th digit $\in \mathcal{D}^{(CS3)}$; all others $\in \{0,1\}$

The notation $SD3^{\langle\pm\rangle}$ refers to either $SD3^{\langle+\rangle}$ or $SD3^{\langle-\rangle}$.

high-radix (2^k) representations. The methods developed in this paper are also applicable to formats where the placement of redundant digits is arbitrary.

While the theory developed in [1] can be applied to the digit sets $\mathcal{D}^{(SD)}$, $\mathcal{D}^{(SD3^{\langle-\rangle})}$, $\mathcal{D}^{(SD3^{\langle+\rangle})}$, and $\mathcal{D}^{(CS3)}$, it does not cover addition methods based on the $CS2$ number representation that are described in Section 7. Also, the approach taken in [1] does not cover addition at the nonredundant digit positions of SD_k and $CS2\_k$.

1.3 Redundant Binary Encodings

All of the number systems in Table 1 need two bits to represent each redundant digit. However, specific encodings should be chosen which lend themselves to efficient implementations. Consider the encoding of an operand X as $(x_{n-1}, x_{n-2}, \cdots, x_0)$, where x_i is the radix-2 (possibly redundant) digit in the ith position. For clarity, a hat notation will be used to distinguish a redundant digit from a nonredundant digit ($\hat{x}_i$ indicates that the ith digit of X is redundant and is encoded using two bits, x_j indicates that the jth digit is nonredundant and is encoded using a single bit). The bits representing a redundant digit $\hat{x}_i$ can be thought of as having *higher* and *lower* significant bits (x_i^h, x_i^l), respectively. Note that arbitrary bit combinations can be used to represent a redundant digit, $\hat{x}_i$, but we concentrate on weighted encodings that satisfy the relationship

$$\hat{x}_i = \pm 2 \cdot x_i^h \pm x_i^l. \tag{5}$$

Weighted encodings for all digit sets of cardinality 4 (i.e., $SD3^{\langle\pm\rangle}$ and $CS3$) must be of the form shown in (5). Here, the bit x_i^h can be interpreted as a transfer-digit [6]. It will be shown that such encodings lead to efficient implementations.

In signed-digit representations, we refer to the higher significant bit, x_i^h, as the *polarity* bit and the lower significant bit, x_i^l, as the *magnitude* bit. For the digit sets $\mathcal{D}^{(SD3^{\langle-\rangle})}$ and $\mathcal{D}^{(SD)}$, redundant digits are encoded as $\hat{x}_i = -2 \cdot x_i^h + x_i^l$, which corresponds to a two's complement encoding. The digit set $\mathcal{D}^{(SD3^{\langle+\rangle})}$ is realized by simply changing the sign of both x_i^h and x_i^l, that is, $\hat{x}_i = 2 \cdot x_i^h - x_i^l$. Since the digit set $\mathcal{D}^{(SD)}$ does not include -2, the bit pattern $(x_i^h, x_i^l) = (1, 0)$ is not valid in either the SD or the SD_k representations.

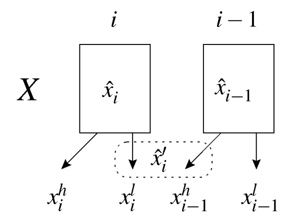

Fig. 2. Equal-weight grouping.

Similarly, following the literature, for carry-save-based redundant representations, we refer to the higher significant bit, x_i^h, as the *carry* bit and the lower significant bit, x_i^l, as the *sum* bit. Here, the digit sets $\mathcal{D}^{(CS3)}$ and $\mathcal{D}^{(CS2)}$ are encoded as $\hat{x}_i = 2 \cdot x_i^h + x_i^l$. The digit set $\mathcal{D}^{(CS2)}$ does not include 3, which makes the bit pattern $(x_i^h, x_i^l) = (1, 1)$ invalid for the $CS2$ and $CS2\_k$ representations.

Note that digits sets with cardinality 3 (i.e., $\mathcal{D}^{(SD)}$ and $\mathcal{D}^{(CS2)}$) can employ an encoding of the type $x_i^h = \pm x_i^h \pm x_i^l$, with both bits having the same relative weight. For the SD case, such an encoding is known as the borrow-save encoding. These encodings allow digits to be formed from any of the four possible 2-bit patterns (i.e., no invalid combinations). For the sake of consistency when comparing with digit sets of cardinality 4, we do not consider such encodings.

1.4 Equal-Weight Grouping

The chosen encoding of both signed-digit and carry-save redundant digits ensures that x_i^l and x_{i-1}^h have the same weight, i.e., the digits $\hat{x}_i$ and $\hat{x}_{i-1}$ *overlap* each other. This overlap can be exploited to reduce the range of digit sums that must be generated and to predict the range of an incoming carry when two numbers are added. Fig. 2 shows two redundant digits, $\hat{x}_i$ and $\hat{x}_{i-1}$, of a number X drawn as squares. The arrows are used to indicate the individual bits that make up each digit. Instead of having digits of the form $\hat{x}_i = 2 \cdot x_i^h + x_i^l$, the bits can create "Equal-Weight Grouped" (EWG) digits of the form $\hat{x}'_i = x_i^l + x_{i-1}^h$ without affecting the value of the original operand X. To illustrate the impact of equal-weight grouping, consider adding two $CS3$ (i.e., conventional carry-save format) numbers X and Y, where the digit set is $\mathcal{D}^{(CS3)} = \{0, 1, 2, 3\}$. Normally, the digit sum at position i, $\theta_i = 2 \cdot x_i^h + x_i^l + 2 \cdot y_i^h + y_i^l$, would be in the range $0 \le \theta_i \le 6$. This must be expressed as a final sum $0 \le z_i \le 3$ (assuming the output format is the same as the input, i.e., $CS3$) and a carry-out, c_i, which may be larger than 1. If EWG digits are added instead, the digit sum $\theta'_i = x_i^l + x_{i-1}^h + y_i^l + y_{i-1}^h$ is restricted to the range $0 \le \theta'_i \le 4$. This is still expressed with a final sum of $0 \le z_i \le 3$, but the carry-out, c_i, will be at most 1. As a result, the number of values needed for the carry-out is reduced.

Another benefit of working with bits originally from distinct digits arises when considering digit sets which exclude some bit patterns, as in $CS2$ or SD. In these cases, the higher-significant bits from the less-significant digits, x_{i-1}^h and y_{i-1}^h, provide some information about what range the less-significant digit sum, θ_{i-1}, is in and, therefore, the range of the incoming carry. Note, however, that, in these cases, the range of the digit sum is not affected by the equal-weight grouping.

1.5 Partially Redundant Representations

While the fixed delay for constant-time addition is minimized when the output is fully redundant, other possibilities exist that address different design constraints (such as area or power). For example, a fully redundant output requires twice as many bit-lines as a nonredundant output. To reduce the number of bit-lines, the number of redundant output digits can be reduced. For the signed-digit family, such a framework was illustrated in [7] and a similar framework exists for the carry-save family [8].

In general, two operands, X and Y, with redundant digits at arbitrary positions can be added to produce an output, Z, with redundant digits at positions completely unrelated to the redundant digit positions in either X or Y. It can be shown that such addition and simultaneous format conversion is possible in constant time, independent of the word-length [2], [3], [4], [7], [8]. Obviously, the right context $\mathcal{L}$ depends on the specific operand formats in question. It can be verified that the worst-case delay (i.e., longest context or look-back) occurs when all of the digits in both operands X and Y are redundant and only some digits of the output Z are redundant. As shown in Fig. 1, the critical path delay of such constant-time addition and simultaneous format conversion depends mainly on the distance between redundant digits in the output. It can be shown that, in all cases, the context that is sufficient to generate the correct intermediate sum and carry-out, c_q, of the qth redundant digit position includes all radix-2 digits up to (but not including) the $(q-2)$nd redundant digit, irrespective of whether or not the redundant digits are uniformly spaced. In other words, the context, $\mathcal{L}$, now spans up to two larger groups or "super-digits." It may be possible to look at only the upper few digits of the previous group, thereby shortening $\mathcal{L}$ and the critical path. However, the critical path is still much longer than that achieved by the EWG scheme.

Given this framework for constant-time addition with and without simultaneous format conversion, we now consider the specific cases of the redundant radix-2 number systems listed in Table 1 and identify the ones that lead to efficient implementations. In Sections 2 and 3, we discuss addition of SD numbers with conversion to an output format of SD_k and without such a conversion, respectively. In Sections 4 and 5, we consider $SD3^{(\pm)}$ addition with and without format conversion to $SD3^{(\pm)}\_k$, respectively. Addition of $CS2$ and $CS3$ numbers, with and without format conversion, is discussed in Sections 6, 7, 8, and 9. Section 10 compares the 10 previously presented number systems and, in Section 11, we show implementations of several adder cells and present the corresponding cell delays. Section 12 discusses some theoretical issues and final conclusions are presented in Section 13.

Once again, it should be noted that the uniform distance between redundant digits in the partially redundant formats considered in the following sections is only for the sake of illustration. The ensuing analysis and results are general and apply even when the distance between redundant digits in the output is nonuniform.

2 SD ADDITION WITH FORMAT CONVERSION

The operation under consideration expresses its output in a partially redundant form. The two input operands, X and Y, are in the conventional signed-digit format with a digit set of $\mathcal{D}^{(SD)} = \{-1, 0, 1\}$. The output, Z, is expressed in the SD_k format, where every kth digit $(k > 1)$ is a member of $\mathcal{D}^{(SD)}$ and the remaining digits are nonredundant bits. Note that, because of the $\mathcal{D}^{(SD)}$ digit set encoding, the bit pattern $(x_i^h, x_i^l) = (1, 0)$ never occurs. In other words, $x_i^h = 1 \Rightarrow x_i^l = 1$ and $x_i^l = 0 \Rightarrow x_i^h = 0$. For the sake of clarity, we first consider nonredundant positions. First, assume that the carry set, $\mathcal{C}^{(SD\_k)}$, is limited to $\{-1, 0, 1\}$; it will be shown below that a carry value of -2 is never needed.

Consider the rth position $(0 \leq r \leq k-2)$ which has a nonredundant output and a weight of 2^r. The four input bits which have the weight of 2^r are $(x_r^l, y_r^l, x_{r-1}^h, y_{r-1}^h)$. Let their sum be denoted by $\theta_r = x_r^l + y_r^l - x_{r-1}^h - y_{r-1}^h$, where $-2 \leq \theta_r \leq +2$. The final output bit, z_r, must satisfy the basic carry relationship stated in (1). Using the definition of θ_r, this becomes

$$2 \cdot c_r + z_r = \theta_r + c_{r-1}. \tag{6}$$

It would appear that, since c_{r-1} can take any of three values, $\{-1, 0, 1\}$, the sum $\theta_r + c_{r-1}$ is in the range $[-3, +3]$. If the value -3 did occur, it would have to be expressed as $(-4+1)$, which implies that a carry-out of $c_r = -2$ is needed. Fortunately, this situation never arises. In other words, if $\theta_r = -2$ then $c_{r-1} \geq 0$, i.e., the incoming carry cannot be negative. In fact, the following stronger result holds.

Theorem 1. *For EWG addition which uses the SD encoding, if* $x_{i-1}^h \cdot y_{i-1}^h = 1$, *then* $c_{i-1} \geq 0$. *In other words, if both polarity bits of an EWG digit are negative, the carry-in cannot be negative.*

Proof. If $x_{i-1}^h \cdot y_{i-1}^h = 1$, then x_{i-1}^l and y_{i-1}^l (which participate in generating θ_{i-1}) must both equal 1 since the bit combination $(x_{i-1}^h, x_{i-1}^l) = (1, 0)$ can never occur. Consequently, if $x_{i-1}^h \cdot y_{i-1}^h = 1$, the digit sum $\theta_{i-1} = x_{i-1}^l + y_{i-1}^l - x_{i-2}^h - y_{i-2}^h$ is restricted to $\theta_{i-1} \in \{0, 1, 2\}$. If $\theta_{i-1} = 1$ or 2, then, even if $c_{i-2} = -1$, $\theta_{i-1} + c_{i-2} \geq 0$, yielding $c_{i-1} \geq 0$, as required.

The only remaining case is when $\theta_{i-1} = 0$. Since $x_{i-1}^l = y_{i-1}^l = 1$, the digit sum $\theta_{i-1} = 0$ can only occur when $x_{i-2}^h = y_{i-2}^h = 1$, implying $\theta_{i-2} \geq 0$. This means that an incoming carry of -1 to position i can only occur if there is an incoming carry of -1 to some previous position $i-m$, after a string of m previous digit sums equal to 0 (meaning $\theta_{i-j} = 0$ for $1 \leq j \leq m$, $m \geq 1$). This is impossible since the string of previous digit sums, θ_{i-1} through θ_{i-m} equal to 0 must terminate somewhere, possibly at the least significant digit of the number, with $\theta_{i-m-1} \in \{0, 1, 2\}$ (due to the SD encoding). This implies the carry-in to position $i-m$ is $c_{i-m-1} \geq 0$ or, more specifically, $c_{i-m-1} \in \{0, 1\}$. Since $\theta_{i-m} = 0$, the incoming carry, c_{i-m-1}, will be absorbed at position $i-m$ and there is no carry-in to position i, or $c_{i-1} = 0$. □

TABLE 2
Rules for Constant-Time Addition ($SD + SD \rightarrow SD\_k$) at a Position with a Redundant Output

θ_i	θ_{i-1}	Possible c_{i-1}	Carry-out c_i	Intermediate Sum σ_i
-2	$\times$	$\times$	-1	0
-1	$\theta_{i-1} \leq 0$	$\{-1,0\}$	-1	1
	otherwise	$\{0,1\}$	0	-1
0	$\times$	$\times$	0	0
1	$\theta_{i-1} \leq 0$	$\{-1,0\}$	0	1
	otherwise	$\{0,1\}$	1	-1
2	$\times$	$\times$	1	0

The $\times$ symbol indicates "don't cares," i.e., the value of θ_{i-1} and c_{i-1} are inconsequential in these cases.

This result in effect shows that $\theta_i + c_{i-1} \geq -2$ or, in other words, it will never be -3, thereby obviating the need for the carry value -2. Once this is established, the rules of operation at the unsigned (nonredundant) digit position are straightforward and are summarized by

$$\begin{aligned} &2 \cdot c_i + z_i = \theta_i + c_{i-1} \\ &\quad \text{where } c_i, c_{i-1} \in \{-1, 0, 1\} \text{ and } z_i \in \{0, 1\}. \end{aligned} \tag{7}$$

Next, we consider a position which has a redundant output digit. This position can generate the carry-out by looking only at the bits of the previous position. Note that Theorem 1 applies regardless of whether the output is in redundant format. Also, since the output digit is redundant, it can assume a value of -1, which allows multiple ways of expressing an output of ± 1. Table 2 summarizes the rules for constant-time addition and simultaneous format conversion at a redundant output position. Note that the intermediate sum σ_i is determined so that, for any possible incoming carry c_{i-1}, there will never be a new outgoing carry generated when calculating the final sum $z_i = \sigma_i + c_{i-1}$. This is a result of Theorem 1 and the rules shown in Table 2 are, in fact, identical to the case where the operation under consideration is $SD + SD \rightarrow SD$, without format conversion.

We would like to point out that, without the equal-weight grouping which results in "exporting" the polarity bits from the previous digit, any format conversion during addition becomes significantly more complex. It can be verified that, without EWG, the carry set needed becomes $\{-2, -1, 0, 1\}$, which is more complex than the EWG scheme. Worse yet, the look-back required to determine the carry-out at every redundant position is *much* longer since a carry of value -2 greatly complicates the rules (because -2 is not an allowed output digit). It can be shown that, in this case, a look-back of length $2k - 1$ radix-2 digit positions is sufficient to generate the correct intermediate sum and carry at each redundant output position.

3 SD Addition without Format Conversion

The operation under consideration expresses its output in a fully redundant form. The two input operands, X and Y, as well as the output, Z, are in the conventional signed-digit format, where the digit set is $\mathcal{D}^{(SD)} = \{-1, 0, 1\}$. Considering EWG digits, the four bits that contribute to the digit-wise sum of the operands, θ_i, are x_i^l, y_i^l, each with a weight of $+1$, and x_{i-1}^h, y_{i-1}^h, each with a weight of -1. As a result, θ_i is in the range $[-2, +2]$. It can be verified that the carry set $\mathcal{C}^{(SD)} = \{-1, 0, 1\}$ is sufficient in this case. The rules for this constant-time addition without format conversion are summarized in Table 3.

Table 3 shows the only allowable (c_i, σ_i) combinations for expressing the digit sum $\theta_i \in \{-2, 0, 2\}$. There are multiple ways of expressing digit sum $\theta_i \in \{-1, 1\}$ and the rules in Table 3 are justified by the following observations: For θ_i to equal -1, at least one of the polarity bits must equal -1. In this case, the carry-in satisfies $c_{i-1} \in \{0, 1\}$ as proven in Lemma 1 below. The EWG digit sum θ_i can equal 1 in the following two ways:

1. When only one magnitude bit equals 1 and all other bits are 0, or $x_i^l \oplus y_i^l = 1$ and $x_{i-1}^h = y_{i-1}^h = 0$. In this case, θ_{i-1} can assume any value in the range $-2 \leq \theta_{i-1} \leq 2$ and, as a result, $c_{i-1} \in \{-1, 0, 1\}$. Consequently, if $x_{i-1}^h = y_{i-1}^h = 0$, we must look back at x_{i-1}^l and y_{i-1}^l to determine the correct setting of σ_i in order to avoid further carry generation when determining the final sum.
2. When both magnitude bits equal 1 and only one polarity bit equals 1, or $x_i^l = y_i^l = 1$ and $x_{i-1}^h \oplus y_{i-1}^h = 1$. Lemma 1 applies in this case, ensuring that the incoming carry is restricted to $c_{i-1} \in \{0, 1\}$. No consideration of θ_{i-1} is needed and

TABLE 3
Rules for Constant-Time Addition $SD + SD \rightarrow SD$

θ_i		θ_{i-1}	Possible c_{i-1}	Carry-out c_i	Intermediate Sum σ_i
-2		$\times$	$\times$	-1	0
-1		$\times$	$\{0,1\}$	0	-1
0		$\times$	$\times$	0	0
1	$x_{i-1}^h = y_{i-1}^h = 0$	$x_{i-1}^l \vee y_{i-1}^l = 1$	$\{0,1\}$	1	-1
		otherwise	$\{-1,0\}$	0	1
	otherwise	$\times$	$\{0,1\}$	1	-1
2		$\times$	$\times$	1	0

The symbol $\vee$ denotes the "OR" function.

$\theta_i = 1$ can be expressed as the intermediate sum $\sigma_i = -1$ and a carry-out of $c_i = 1$.

Lemma 1. *For EWG addition without format conversion which uses the SD encoding, if $x_{i-1}^h \vee y_{i-1}^h = 1$, then $c_{i-1} \geq 0$. In other words, if either polarity bit of an EWG digit is negative, the carry-in cannot be negative.*

Proof. Theorem 1 applies when $x_{i-1}^h = y_{i-1}^h = 1$, so the remaining case is when exactly one of x_{i-1}^h or y_{i-1}^h equals 1. Without loss of generality, assume that $x_{i-1}^h = 0$ and $y_{i-1}^h = 1$. This implies that $y_{i-1}^l = 1$ and, consequently, $\theta_{i-1} \geq -1$. This corresponds to one of the following conditions:

1. If $\theta_{i-1} \geq 0$, then $\theta_{i-1} + c_{i-2} \geq -1$, which means $c_{i-1} \geq 0$. This is because the redundant output digit $\hat{z}_{i-1}$ can assume the value -1, meaning $\theta_{i-1} + c_{i-2} = -1$ does not lead to an outgoing carry, or $c_{i-1} = 0$.
2. Since $y_{i-1}^l = 1$, the only way $\theta_{i-1} = -1$ can occur is if $x_{i-1}^l = 0$, $x_{i-2}^h = 1$, and $y_{i-2}^h = 1$. This in turn implies that $c_{i-2} \geq 0$ (by Theorem 1). This allows $\theta_{i-1} = -1$ to be left as the intermediate sum $\sigma_{i-1} = -1$ with no carry-out, or $c_{i-1} = 0$. □

Note that the only time a look-back is needed is when $\theta_i = +1$ and $x_{i-1}^h = y_{i-1}^h = 0$. Because of the equal-weight grouping, no look-back is necessary when $\theta_i = -1$. Although the context, $\mathcal{L}$, equals one digit position, as in conventional SD addition without equal-weight-grouping, Table 3 can be thought of as simpler than the corresponding table(s) in other SD addition schemes proposed so far. For instance, there are more don't cares in Table 3 than in the corresponding table(s) from [16] and its derivatives. This may lead to a simplification of switching expressions and, hence, the implementation. The fundamental difference is that, for schemes which do not use equal-weight grouping, it is necessary to look back at the previous digit position when $\theta_i = -1$ as well as when $\theta_i = +1$.

4 $SD3^{\langle\pm\rangle}$ Addition with Format Conversion

Two closely related types of redundant digit representations are considered in this section, $SD3^{\langle-\rangle}$ and $SD3^{\langle+\rangle}$. Again, this operation expresses its output in a partially redundant form. For $SD3^{\langle-\rangle}$, each redundant digit is in the digit set $\mathcal{D}^{(SD3^{\langle-\rangle})} = \{-2,-1,0,1\}$ and, for $SD3^{\langle+\rangle}$, the digit set $\mathcal{D}^{(SD3^{\langle+\rangle})} = \{-1,0,1,2\}$ is used.

In $SD3^{\langle-\rangle}$, it can be verified that the carry set $\mathcal{C}^{(SD^{\langle-\rangle}k)} = \{-2,-1,0,1\}$ is sufficient. The rules at a nonredundant output position are then simple and summarized by

$$2 \cdot c_i + z_i = \theta_i + c_{i-1} \quad \text{where } c_i, c_{i-1} \in \{-2,-1,0,1\} \text{ and } z_i \in \{0,1\}. \tag{8}$$

Next, consider a position with redundant output which can assume any value in the range $[-2,+1]$. The rules to generate the intermediate sum and carry-out are summarized in Table 4. From the third and fifth columns of the table, it is seen that $\sigma_i + c_{i-1}$ is always in the range $[-2,+1]$.

TABLE 4
Rules for Constant-Time Addition ($SD3^{\langle-\rangle} + SD3^{\langle-\rangle} \rightarrow SD3^{(SD^{\langle-\rangle}k)}$) at a Redundant Output Position

θ_i	θ_{i-1}	Possible c_{i-1}	Carry-out c_i	Intermediate Sum σ_i
−2	×	×	−1	0
−1	−2	{−2,−1}	−1	1
	−1	{−2,−1,0}	−1	1
	0	{−1,0}	*Either representation ok*	
	1	{−1,0,1}	0	−1
	2	{0,1}	0	−1
0	×	×	0	0
1	−2	{−2,−1}	0	1
	−1	{−2,−1,0}	0	1
	0	{−1,0}	*Either representation ok*	
	1	{−1,0,1}	1	−1
	2	{0,1}	1	−1
2	×	×	1	0

This shows that the second constant-time addition step will never generate a carry when determining the final sum. Note that, in this case, the carry set, $\mathcal{C}^{(SD3^{\langle-\rangle}k)}$, and the destination digit set, $\mathcal{D}^{(SD3^{\langle-\rangle})}$, are identical. Therefore, leaving behind an intermediate sum of $\sigma_i = 0$ is always safe.

As mentioned earlier, if the source and destination digit set is $\mathcal{D}^{(SD3^{\langle+\rangle})} = \{-1,0,1,2\}$ instead of $\mathcal{D}^{(SD3^{\langle-\rangle})}$, the polarity bits should be assigned a positive weight and the magnitude bits a negative weight. Once again, all four bits (two magnitude and two polarity bits) of the same weight can be grouped together so the digit sum, θ_i, is in the range $[-2,+2]$. It can be verified that the carry set $\mathcal{C}^{(SD3^{\langle\pm\rangle}k)} = \{-2,-1,0,1\}$ is sufficient. The rules for addition at nonredundant positions are again summarized by (8). The rules for a redundant position are similar to those in Table 4 and are omitted for the sake of brevity (please refer to the technical report [19] for details).

For both digit sets, if the equal-weight grouping method is not employed, the operation ($SD3^{\langle\pm\rangle} + SD3^{\langle\pm\rangle} \rightarrow SD3^{\langle\pm\rangle}\_k$) requires a larger carry set and longer context than the corresponding case when equal-weight grouping is employed.

5 $SD3^{\langle\pm\rangle}$ Addition without Format Conversion

Again, both $SD3^{\langle-\rangle}$ and $SD3^{\langle+\rangle}$ will be considered for constant-time addition, but without any format conversion. First, consider the digit set $\mathcal{D}^{(SD3^{\langle-\rangle})} = \{-2,-1,0,1\}$. Like the $SD3^{\langle\pm\rangle}\_k$ case, the digit sum, θ_i, is in the range $[-2,+2]$. It can be shown that the carry set $\mathcal{C}^{(SD3^{\langle-\rangle})} = \{-1,0,1\}$ is sufficient in this case. The rules for constant-time addition are summarized by

$$2 \cdot c_i + \sigma_i = \theta_i \quad \text{where } c_i \in \{-1,0,1\} \text{ and } \sigma_i \in \{-1,0\}. \tag{9}$$

TABLE 5
$CS2$ Addition Rules: (a) Redundant Position Rules, (b) Simplified Rules for Non-Format-Conversion Addition

θ_i	θ_{i-1}	c_i	σ_i
0	×	0	0
1	{0,1}	0	1
	{2,3,4}	1	−1
2	×	1	0
3	{0,1}	1	1
	{2,3}	2	−1
	4	–	–
4	×	2	0

(a)

θ_i	$x^h_{i-1} + y^h_{i-1}$	c_i	σ_i
0	×	0	0
1	×	0	1
2	2	0	2
	otherwise	1	0
3	×	1	1
4	×	1	2

(b)

Note that now -1 and 0 are "safe-digits" to leave behind as an intermediate sum since

$$\{-1,0\} + \mathcal{C}^{(SD3^{\langle -\rangle})} = \{-1,0\} + \{-1,0,1\} = \{-2,-1,0,1\},$$

which is the digit set $\mathcal{D}^{(SD3^{\langle -\rangle})}$. Furthermore, there is no explicit look-back, meaning no dependence on θ_{i-1}. The intermediate sum, σ_i, and carry-out, c_i, at each position i depend *only* on the four operand bits in the current group. In contrast, for all conventional signed-digit adder-cell implementations, σ_i and c_i depend on six operand bits, for instance, those in [16] and their derivatives. Thus, it can be expected that a cell which implements $SD3^{\langle -\rangle} + SD3^{\langle -\rangle} \rightarrow SD3^{\langle -\rangle}$ is less complex than a cell that performs $SD + SD \rightarrow SD$.

Using the digit set $\mathcal{D}^{(SD3^{\langle +\rangle})} = \{-1,0,1,2\}$, where the polarity of the bits in each redundant digit are reversed, the rules for constant time addition are summarized by

$$2 \cdot c_i + \sigma_i = \theta_i \quad \text{where } c_i \in \{-1,0,1\} \text{ and } \sigma_i \in \{0,1\}. \quad (10)$$

Here, 0 and $+1$ are safe digits to leave behind as an intermediate sum. It is clear from (9) and (10) that the gate-level implementation of a cell that performs $SD3^{\langle +\rangle} + SD3^{\langle +\rangle} \rightarrow SD3^{\langle +\rangle}$ can be identical to that of a cell performing $SD3^{\langle -\rangle} + SD3^{\langle -\rangle} \rightarrow SD3^{\langle -\rangle}$.

6 $CS2$ ADDITION WITH FORMAT CONVERSION

This section considers $CS2$ constant-time addition with format conversion. The digit set at each redundant position is $\mathcal{D}^{(CS2)} = \{0,1,2\}$ and, as mentioned earlier, the encoding prevents the bit combination $(x^h_i, x^l_i) = (1,1)$ from occurring. The following lemma is essential in determining the carry set required for this case.

Lemma 2. *For EWG addition which uses the $CS2$ encoding, if $x^h_{i-1} \cdot y^h_{i-1} = 1$, then $c_{i-1} \le 1$. In other words, if both lower bits of an EWG digit equal 1, the carry-in cannot be 2. (Note that, for the SD representations, the corresponding result is stated in Theorem 1.)*

Proof. Let the sum $x^h_{i-1} + y^h_{i-1} = \theta^h_i$. Because of the $CS2$ encoding, $\theta^h_i = 2$ implies $x^l_{i-1} = y^l_{i-1} = 0$ and, as a result, $\theta_{i-1} \in \{0,1,2\}$. Since $\theta_{i-1} \in \{0,1\}$ never produces a carry-out of 2, the only concern is when $\theta_{i-1} = 2$, which, in turn, implies $\theta_{i-2} \in \{0,1,2\}$. Consequently, position $i-1$ could produce a carry-out of 2 only if $\theta_{i-2} = 2$. This is now a recursive argument since position $i-2$ can produce a carry-out of 2 only if $\theta_{i-3} = 2$ and so on. Since all the numbers being considered are assumed to be of some fixed length, the question becomes: Can a string of intermediate sums $\theta^h_i\, \theta_{i-1}\, \theta_{i-2} \cdots \theta_{i-m} = 2\,2\,2 \cdots \theta_{i-m}$ terminate with $\theta_{i-m} > 2$? This is not possible due to the encoding selected. Therefore, $\theta_{i-m} \in \{0,1,2\}$, which implies $c_{i-m} \in \{0,1\}$, which in turn implies $c_{i-1} \in \{0,1\}$. □

Lemma 2 implies that if the digit sum at position i, $\theta_i = 4$, then a carry-in of 2 can never occur. This means that the carry set $\{0,1,2\}$ is sufficient. The rules for nonredundant output digit positions are then summarized by

$$2 \cdot c_i + z_i = \theta_i + c_{i-1} \quad \text{where } c_i, c_{i-1} \in \{0,1,2\} \text{ and } z_i \in \{0,1\}. \quad (11)$$

Next, consider a redundant output digit position which can determine the range of an incoming carry by examining the previous digit sum, θ_{i-1}. These rules are shown in Table 5a. The only apparent abnormality is that an intermediate sum of $\sigma_i = -1$ is allowed, which is not a valid final sum. However, this only occurs when a positive carry-in ($c_{i-1} > 0$) is guaranteed, according to (11). This is simply a matter of notation in order to make the table consistent with the relationship $2 \cdot c_i + \sigma_i = \theta_i$.

7 $CS2$ ADDITION WITHOUT FORMAT CONVERSION

Although the rules from Table 5a for $CS2\_k$ addition at a redundant position apply when there is no format conversion, they are based on the assumption that the incoming carry comes from a nonredundant position. If the previous position is also redundant, it has a larger capacity, which could limit the range of its carry-outs. This is possible if the following carry-relationship is satisfied:

$$x^l_i + y^l_i + x^h_{i-1} + y^h_{i-1} + c_{i-1} \le z_{i_{\max}} + 2 \cdot c_i \quad \text{where } z_{i_{\max}} = 2. \quad (12)$$

Without restricting x^l_i and y^l_i, the carry-out can be limited to $c_i \in \{0,1\}$ if

$$x^h_{i-1} + y^h_{i-1} + c_{i-1} \le 2. \quad (13)$$

TABLE 6
$CS3$ Addition Rules: (a) Redundant Position Rules, (b) Simplified Rules for Non-Format-Coversion Addition

θ_i	θ_{i-1}	c_i	σ_i
0	$\times$	0	0
1	0,1	0	1
	2	*Either representation ok*	
	3,4	1	-1
2	$\times$	1	0
3	0,1	1	1
	2	*Either representation ok*	
	3,4	2	-1
4	$\times$	2	0

(a)

θ_i	c_i	σ_i
0	0	0
1	0	1
2	1	0
	or	
	0	2
3	1	1
4	1	2

(b)

Lemma 3. *For EWG addition without format conversion which uses the $CS2$ encoding, if $x^h_{i-1} \cdot y^h_{i-1} = 1$, then $c_{i-1} = 0$. In other words, if both lower bits of an EWG digit equal 1, there will be no carry-in to that position.*

Proof. Since $x^h_{i-1} = y^h_{i-1} = 1$, the encoding restricts the previous digit sum to $\theta_{i-1} \in \{0,1,2\}$. In this case, since 2 is a valid final sum, a carry-out from the previous position, c_{i-1}, would only occur if $\theta_{i-1} = 2$ and there were a carry-in to the previous position of $c_{i-2} = 1$. Having $\theta_{i-1} = 2$ then restricts θ_{i-2} to $\theta_{i-2} \in \{0,1,2\}$ and the scenario discussed in Lemma 2 now exists with a string of digit sums equal to 2. The encoding dictates that this string of digit sums equal to 2 must eventually terminate at some position $i - m$ with $\theta_{i-m} \in \{0,1,2\}$. If $\theta_{i-m} = 2$, then position $i - m$ must be the least significant digit of the number (or the string would continue). In this case, since there is no carry-in to position $i - m$ and 2 is a valid final sum, there is no need for a carry-out of position $i - m$. Similarly, if $\theta_{i-m} \in \{0,1\}$, there is no need for a carry-out of position $i - m$. Therefore, positions $i - m + 1$ through $i - 1$ can express their digit sum of 2 as an intermediate sum of 2 and no carry-out. This ensures that $c_{i-1} = 0$. □

Lemma 3 shows that (13) is always satisfied, meaning the carry set $\mathcal{C}^{(CS2)} = \{0,1\}$ is sufficient. Given this, the rules for $CS2$ addition without any format conversion can be simplified, as shown in Table 5b. Note that there is no need to look back at any previous digits, in other words, the look-back is $\mathcal{L} = 0$.

8 $CS3$ Addition with Format Conversion

Here, the redundant digits can take any value from the digit set $\mathcal{D}^{(CS3)} = \{0,1,2,3\}$. It can be verified that the carry set needed for $CS3$ addition with format conversion is $\mathcal{C}^{(CS3\_k)} = \{0,1,2,3\}$. Given this carry set, the rules for $CS3$ addition with format conversion for a redundant position are shown in Table 6a.

As before, an intermediate sum of $\sigma_i = -1$ is left behind only when a carry-in of $c_{i-1} > 0$ is guaranteed. This renders the final sum $\hat{z}_i = \sigma_i + c_{i-1} \in \{0,1,2\}$. The carry-out and intermediate sum for a nonredundant position simply follow:

$$2 \cdot c_i + z_i = \theta_i + c_{i-1} \quad \text{where } c_i, c_{i-1} \in \{0,1,2,3\} \text{ and } z_i \in \{0,1\}. \tag{14}$$

9 $CS3$ Addition without Format Conversion

Here, every output digit position is redundant and can assume any value in $\mathcal{D}^{(CS3)} = \{0,1,2,3\}$. Since 3 is an allowable digit, the carry-relationship

$$\theta_{i_{\max}} + c_{i-1_{\max}} \le z_{i_{\max}} + 2 \cdot c_{i_{\max}}, \tag{15}$$

simplifies to $c_{i_{\max}} \ge 1$ (assuming $c_{i-1_{\max}} = c_{i_{\max}}$). This makes the carry set $\mathcal{C}^{(CS3)} = \{0,1\}$ sufficient for $CS3$ addition without format conversion. The rules for determining σ_i and c_i are given in Table 6b. Again, they are stated only in terms of θ_i, without any dependency on the previous group sum, which makes the look-back $\mathcal{L} = 0$.

10 Comparison

Table 7 gives a summary of the look-back distances, $\mathcal{L}$, and carry sets needed for the 10 types of redundant binary addition considered. The table clearly shows that equal-weight grouping can lead to smaller carry sets and a smaller

TABLE 7
Comparison of Radix-2 Constant-Time Addition Techniques

Operation	Carry Set		Look-Back $\mathcal{L}$ (Number of radix-2 digits)	
	Equal-Weight Grouping (EWG)	No EWG	EWG	No EWG
$SD + SD \to SD$	$\{-1,0,1\}$	$\{-1,0,1\}$	1	1
$SD + SD \to SD\_k$	$\{-1,0,1\}$	$\{-2,-1,0,1\}$	1	$2k-1$
$SD3^{\langle -\rangle} + SD3^{\langle -\rangle} \to SD3^{\langle -\rangle}$	$\{-1,0,1\}$	$\{-2,-1,0,1\}$	0	1
$SD3^{\langle +\rangle} + SD3^{\langle +\rangle} \to SD3^{\langle +\rangle}$	$\{-1,0,1\}$	$\{-1,0,1,2\}$	0	1
$SD3^{\langle -\rangle} + SD3^{\langle -\rangle} \to SD3^{\langle -\rangle}\_k$	$\{-2,-1,0,1\}$	$\{-4,-3,-2,-1,0,1\}$	1	$2k-1$
$SD3^{\langle +\rangle} + SD3^{\langle +\rangle} \to SD3^{\langle +\rangle}\_k$	$\{-2,-1,0,1\}$	$\{-2,-1,0,1,2,3\}$	1	$2k-1$
$CS2 + CS2 \to CS2$	$\{0,1\}$	$\{0,1,2\}$	0	1
$CS2 + CS2 \to CS2\_k$	$\{0,1,2\}$	$\{0,1,2,3\}$	1	$2k-1$
$CS3 + CS3 \to CS3$	$\{0,1\}$	$\{0,1,2,3\}$	0	1
$CS3 + CS3 \to CS3\_k$	$\{0,1,2,3\}$	$\{0,1,2,3,4,5\}$	1	$2k-1$

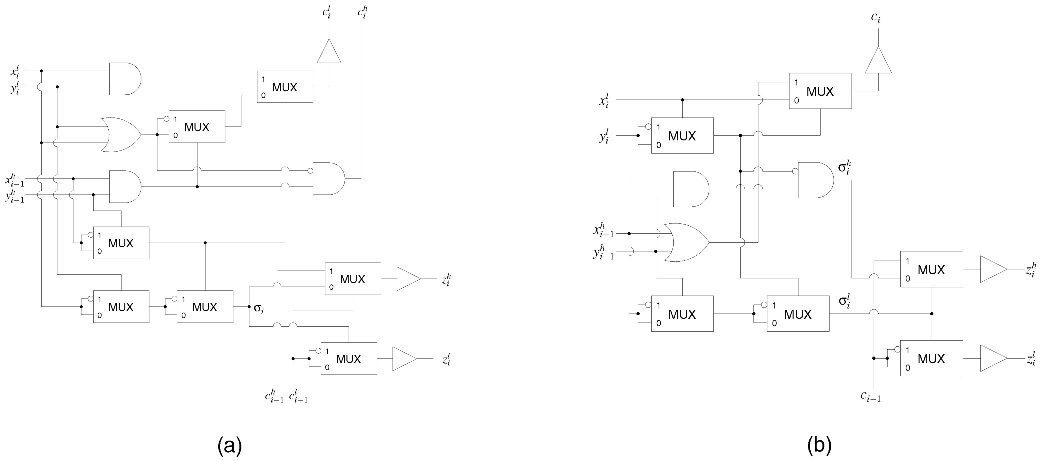

Fig. 3. Redundant adder cells. (a) Cell to perform $SD3^{\langle -\rangle} + SD3^{\langle -\rangle} \rightarrow SD3^{\langle -\rangle}$. (b) Cell to perform $CS2 + CS2 \rightarrow CS2$.

look-back. The longest carry propagation path increases with both the right context and the distance between redundant digits. Consequently, the smallest critical path delay of an implementation can be expected under the following conditions:

1. The look-back, $\mathcal{L}$, is minimized.
2. The distance to the closest higher-order redundant digit is minimized, which happens when all output digits are redundant.

Table 7 shows that the minimum look-back occurs only when the proposed equal-weight grouping is employed. Among the cases with zero look-back, those with the smallest carry set should be selected since a smaller carry set usually implies less complex logic, which should translate into smaller area and critical path delay. Applying these criteria, it is seen that the $CS2$ and $CS3$ representations (i.e., the carry-save representations) are more likely to result in better designs than the signed digit representations. When format conversions are considered, the minimally redundant ($CS2$ and SD) representations outperform their overly redundant counterparts ($CS3, SD3^{(\pm)}$) in terms of the carry set needed.

Format conversions can be highly effective for $\mathrm{Area} \times \mathrm{Delay}$ ($A \times T$) efficient multiplier designs. For instance, in [20], it was shown that multipliers based on SD_k with $k = 2$ have a lower $A \times T$ product than those based on the full SD representation of [16]. It turns out that adding partial products (which are in two's-complement format) to directly generate outputs in this SD_k format is costly in terms of area and delay. A better approach is to add the partial products and generate outputs in SD format at the top level of the partial product accumulation tree. At the next level of the tree, the $SD + SD \rightarrow SD\_k$ format conversion can be carried out during the addition.

Format conversions are also useful if there is a need to *gradually* introduce or remove redundancy in number representations. Note that, by controlling the number and placement of the redundant digits, one can cover the entire spectrum of representations from two's complement (no redundant digits) to fully redundant (such as SD or CS, where all digits are redundant).

Table 7 compares the various representations at an abstract level, in terms of carry set size and look-back $\mathcal{L}$. While this comparison can provide a good high-level assessment, actual VLSI implementations are necessary to gauge the relative merits and disadvantages of the various redundant representations. In the next section, we present simulation results from the VLSI layouts of several adder cells.

11 Implementation

In order to verify some of the comparison results included in Table 7, we designed, laid out, and simulated adder cells for the following cases:

1. $SD + SD \rightarrow SD$: The cell in [16] is the most efficient to the best of our knowledge, so we laid out this cell.
2. $SD3^{\langle -\rangle} + SD3^{\langle -\rangle} \rightarrow SD3^{\langle -\rangle}$: Shown in Fig. 3a. The four input operand bits of equal weight are $(x_i^l, y_i^l, x_{i-1}^h, y_{i-1}^h)$. The carry, $c_i \in \{-1, 0, 1\}$, is encoded using bits (c_i^h, c_i^l) with a two's complement encoding, that is, $c_i = -2 \cdot c_i^h + c_i^l$. The intermediate sum σ_i is encoded as a single bit since $\sigma_i \in \{-1, 0\}$. The output is encoded by (z_i^h, z_i^l) and can assume any of the four values $\{-2, -1, 0, 1\}$. Note that this cell can also implement $SD3^{\langle +\rangle} + SD3^{\langle +\rangle} \rightarrow SD3^{\langle +\rangle}$ by interchanging the positive and negative inputs $(x_i^l \leftrightarrow x_{i-1}^h, y_i^l \leftrightarrow y_{i-1}^h)$.
3. $CS2 + CS2 \rightarrow CS2$: Shown in Fig. 3b. The four input operand bits of equal weight are $(x_i^l, y_i^l, x_{i-1}^h, y_{i-1}^h)$. The carry $c_i \in \{0, 1\}$ needs only a single bit line. The intermediate sum, $\sigma_i \in \{0, 1, 2\}$, is encoded as $\sigma_i = 2 \cdot \sigma_i^h + \sigma_i^l$. The output is encoded by (z_i^h, z_i^l) and can assume any of the three values $\{0, 1, 2\}$.
4. $CS3 + CS3 \rightarrow CS3$: This is nothing but a 4:2 compressor employed in conventional multipliers. The 4:2 compressor presented in [21] is extremely efficient and, hence, we laid out this compressor.

For the sake of brevity, the gate diagrams and details of cells 1 and 4 are omitted, those can be found in the references cited. Cells 2 and 3 were newly designed and

TABLE 8
Critical Path Delay through One Cell from SPICE Simulations

Adder Cell	Critical Path Delay (ns)
SD	0.78750
$SD3^{\langle - \rangle}$	0.96025
$CS2$	0.66100
$CS3$	0.46580

their gate diagrams are shown in Fig. 3a and Fig. 3b, respectively. In both the figures, it is seen that the carry-out is generated based only on the bits of the *current* group, i.e., there is no look-back.

Layouts of all four cells were simulated in the TSMC SCN025 0.25 micron technology process (available from MOSIS) with a 2.5 volt supply. The designs were first verified at the logic level. Berkeley SPICE 3f5 was used to estimate the critical path delay of each cell, which included appropriate fan-in and fan-out loading for all components. The results are summarized in Table 8. The relative order of the simulated critical path delay agrees with the results from the cost estimate procedure described in [1] (excluding $CS2$, which [1] does not cover).

It should be noted that the SPICE simulation results are highly layout dependent. These layouts were done to get some idea of the relative comparison of the various redundant adder cells. The $CS2$ and $SD3^{\langle - \rangle}$ cells in particular could be made more compact, which might have a significant impact on the overall delay. In any case, the critical path simulations clearly demonstrate that the carry-save representations considered here lead to faster implementations.

12 Discussion

The practical implication of the results presented above is that the $CS3$ representation along with the 4:2 compressor is the most efficient way to execute constant-time addition. In light of this, for a multiply operation, it can be seen that using the $CS3$ representation with the compressor presented in [21] is likely to yield the fastest implementations (faster and smaller than those based on the SD or $CS2$ representations using cells 1 and 3 mentioned in Section 11). This can be inferred for the following reasons:

1. Converting partial products from two's complement format to $CS3$ format is trivial; it requires no logic gates at all. Merely grouping the bits of the input operands appropriately leads to a valid $CS3$ representation of the output. For example, for input operands, X and Y, grouping bit x_i with bit y_{i-1} creates a valid $CS3$ digit.

   In contrast, if the $CS2$ or conventional SD representation is employed, two's complement partial products must be added to generate outputs in their respective formats. In each of these cases, a small delay worth about one full adder is required to achieve this conversion [7], [16], [20]. In effect, multipliers based on $CS2$ or SD intermediate representations must endure an additional (albeit small) delay at the top level.
2. The 4:2 compressor that performs $CS3 + CS3 \rightarrow CS3$ is smaller and faster than other cells.

These two factors, 1 and 2, together imply that multipliers based on $CS3$ can be expected to outperform multipliers based on other redundant representations.

There is a more fundamental reason for the superiority of the 4:2 compressor. Note that, for both the SD and $CS2$ adder cells, the digit sums are from an input digit set of cardinality 5, that is, $|\{\mathcal{D}_i^x + \mathcal{D}_i^y\}| = 5$. This corresponds to digit sums in the range $[-2, 2]$ and $[0, 4]$ for SD and $CS2$, respectively. This is true regardless of whether or not EWG is employed for these representations. The cardinality of the output digit set in both cases is 3 since valid output digits are in the range $[-1, 1]$ and $[0, 2]$ for SD and $CS2$, respectively. Thus, redundant addition based on these representations converts an operand from an input digit set having cardinality 5 to a result from an output digit set of cardinality 3.

Note that, in $CS3$ addition, after EWG, the digit sums are from an input digit set of cardinality 5 (digit sums in the range $[0, 4]$). The output is also in $CS3$ and, therefore, the cardinality of the output digit set is 4.

It is intuitively clear that converting an input digit set of cardinality 5 into an output digit set of cardinality 3 is a harder task than converting it into an output digit set with cardinality 4. Therefore, cells such as 1 and 3 from Section 11 are fundamentally more complex, hence, bigger and slower.

In fact, Akoi et al. [22] recently proposed a clever method to employ a 4:2 compressor-like cell to execute constant-time SD addition by using the borrow-save encoding ($\hat{x}_i = x_i^h - x_i^l$). In effect, their method employs a 4:2 compressor to perform $SD + SD \rightarrow SD3^{\langle - \rangle}$, that is, a digit set of cardinality 5 gets converted to digit set of cardinality 4. Since the $SD3^{\langle - \rangle}$ output is a weighted encoding, EWG on the *output* is used to retrieve the original borrow-save encoding without any extra logic.

In closing, we show the relationship of this work to the results presented in [1]. The examples of constant-time addition without format conversion that we have described can be rewritten using the notation from [1], as shown below.

$$\begin{aligned}
SD: &\quad 2\langle 2^1 \rangle + \langle 2^1 \rangle \Leftarrow \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^1 \rangle + \langle 1^1 \rangle + \langle 2^1 \rangle \\
SD3^{\langle - \rangle}: &\quad 2\langle 2^1 \rangle + \langle 3^2 \rangle \Leftarrow \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^1 \rangle + \langle 1^1 \rangle + \langle 2^1 \rangle \\
SD3^{\langle + \rangle}: &\quad 2\langle 2^1 \rangle + \langle 3^1 \rangle \Leftarrow \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^1 \rangle + \langle 1^1 \rangle + \langle 2^1 \rangle \\
CS2: &\quad 2\langle 1^0 \rangle + \langle 2^0 \rangle \Leftarrow \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^0 \rangle \\
CS3: &\quad 2\langle 1^0 \rangle + \langle 3^0 \rangle \Leftarrow \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^0 \rangle + \langle 1^0 \rangle.
\end{aligned}$$

The notation shows that the sum of digit sets to the right of the decomposition operator ($\Leftarrow$) are expressed using the digit sets to the left of the operator. A digit set $\langle \delta^\omega \rangle$ is characterized by its diminished cardinality, δ, and negative offset from zero, ω. This represents digits in the range $[-\omega, -\omega + \delta]$ and must include 0. Further details regarding the notation and decompositions can be found in [1].

The analysis in [1] requires that the total diminished cardinality to the left of the decomposition operator, δ_{out}, be greater than or equal to the total diminished cardinality of the right side, δ_{in}. The condition that $\delta_{out} \geq \delta_{in}$ is satisfied in

TABLE 9
Cost Estimates from [1]

Digit Set	CG's Needed	PA's Needed	Total
SD	2	3	5
$SD3^{\langle\pm\rangle}$	3	3	6
$CS3$	2	2	4

all cases above except $CS2$, where $\delta_{out} = 4$ and $\delta_{in} = 5$. Therefore, $CS2$ addition presented in this paper lies outside the framework developed in [1].

The hardware cost estimation approach from [1] can be applied to all cases except $CS2$, with the results shown in Table 9, where Carry Generator is abbreviated CG and Partial Adder PA. The number of Carry Generators listed below is the total number of "redundant" and "nonredundant" carry generators from [1]. As mentioned earlier, the ranking of the totals listed here agrees with our measurements shown in Table 8.

When constant-time addition with simultaneous format conversion is considered, the methodology from [1] cannot be applied to nonredundant positions of the SD_k and $CS2\_k$ formats since $\delta_{out} \not\geq \delta_{in}$. Overall, we have presented some "nonobvious" cases of addition that the theory from [1] does not permit.

13 Conclusion

This paper presents a comprehensive analysis of constant-time addition and simultaneous format conversion, where the source and destination digit sets are based on binary redundant numbers. We investigated encodings that enable "equal-weight grouping" (EWG), wherein bits having the same weight are grouped together during the constant-time addition operation. The analysis and data show that EWG leads to smaller carry sets and context or look-back. These in turn lead to efficient implementations for constant-time addition and simultaneous format conversion of redundant numbers based on the carry-save (CS) and signed-digit (SD) representations. We compared VLSI implementations of various cells to perform constant-time addition and demonstrated that the conventional 4:2 compressor is the most efficient way to execute constant time-addition. Practical implications of this work are immediate and were illustrated via a comparison of multiplier implementations. We explored the fundamental issues underlying constant-time addition and indicated the reasons which render the 4:2 compressor the most efficient way to implement constant-time addition. We also presented some interesting connections to the results from [1].

Possible future work includes finding redundancy metrics which capture the complexity of hardware implementations based on the redundant format under consideration without the need to go through VLSI implementations. Another issue is to extend the necessary and sufficient conditions for constant-time addition derived in [2] to the case where the digit sets at all digit positions are not identical. Such a framework allows for arbitrary spacing of redundant digit positions throughout a representation, as well as the ability to vary the types of redundant digits used. It is conceivable that examples of situations where both left and right contexts are required could arise in such cases. Since digit sets could be radically different from one digit position to the next, it is possible that each position would also need to examine its left context in order to select the appropriate or acceptable carry-out value.

Acknowledgments

The authors would like to thank Professor Naofumi Takagi for his insightful remarks which led them to this investigation. They also wish to thank Professors Milos Ercegovac, Neil Burgess, and Peter Kornerup for their suggestions which improved the quality of this paper. They appreciate the constructive comments from the anonymous reviewers. This work was supported in part by US National Science Foundation grants ECS-9875705 and CDA-80082. A preliminary version of this manuscript was presented as an invited paper at ASILOMAR '99 [8].

References

[1] T.M. Carter and J.E. Robertson, "The Set Theory of Arithmetic Decomposition," *IEEE Trans. Computers,* vol. 39, no. 8, pp. 993-1005, Aug. 1990.

[2] P. Kornerup, "Necessary and Sufficient Conditions for Parallel and Constant Time Conversion and Addition," *Proc. 14th IEEE Symp. Computer Arithmetic,* pp. 152-156, Apr. 1999.

[3] P. Kornerup, "Digit-Set Conversions: Generalizations and Applications," *IEEE Trans. Computers,* vol. 43, no. 5, pp. 622-629, May 1994.

[4] A.M. Nielsen and P. Kornerup, "Redundant Radix Representation of Rings," *IEEE Trans. Computers,* vol. 48, no. 11, pp. 1153-1165, Nov. 1999.

[5] M.D. Ercegovac and T. Lang, "On Recoding in Arithmetic Algorithms," *J. VLSI Signal Processing,* vol. 14, pp. 283-294, 1996.

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

[7] D.S. Phatak and I. Koren, "Hybrid Signed-Digit Number Systems: A Unified Framework for Redundant Number Representations with Bounded Carry Propagation Chains," *IEEE Trans. Computers,* special issue on computer arithmetic, vol. 43, no. 8, pp. 880-891, Aug. 1994. (Unabridged version available at http://www.cs.umbc.edu/~phatak/publications/hsdtrc.pdf).

[8] T. Goff, D.S. Phatak, and I. Koren, "Redundancy Management in Arithmetic Processing via Redundant Binary Representations," *Proc. ASILOMAR '99 (Ann. Conf. Signals Systems and Computers),* pp. 1475-1479, Oct. 1999.

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

[10] C. Nagendra, R.M. Owens, and M.J. Irwin, "Unifying Carry-Sum and Signed-Digit Number Representations," Technical Report CSE-96-036, Computer Science and Eng. Dept., Pennsylvania State Univ., 1996.

[11] I. Koren, *Computer Arithmetic Algorithms.* Amherst, Mass.: Brookside Court Publishers, 1998.

[12] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs.* Oxford Univ. Press, 2000.

[13] R.T. Borovec, "The Logical Design of a Class of Limited Carry-Borrow Propagation Adders," Technical Report 275, Computer Science Dept., Univ. of Illinois, Champaign-Urbana, 1968.

[14] C.Y. Chow and J.E. Robertson, "Logical Design of a Redundant Binary Adder," *Proc. Fourth IEEE Symp. Computer Arithmetic,* pp. 109-115, 1978.

[15] 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.

[16] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi, "Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation," *Proc. Eighth Symp. Computer Arithmetic,* pp. 80-86, 1987.

[17] M.D. Ercegovac and T. Lang, "Effective Coding for Fast Redundant Adders using the Radix-2 Digit Set {0, 1, 2, 3}," *Proc. 31st Asilomar Conf. Signals Systems and Computers,* pp. 1163-1167, 1997.

[18] T. Goff, D.S. Phatak, and I. Koren, "Efficient Arithmetic Implementations Based on Carry-Save Representations," *Proc. SPIE 45th Ann. Meeting, Int'l Symp. Optical Science and Technology,* Aug. 2000.

[19] D.S. Phatak, T. Goff, and I. Koren, "On Constant Time Addition and Simultaneous Format Conversion," Technical Report TR-01-CSEE-2, Computer Science and Electrical Eng. Dept., Univ. of Maryland Baltimore County (UMBC), Feb. 2001.

[20] J.J.J. Lue and D.S. Phatak, "Area $\times$ Delay ($A \cdot T$) Efficient Multiplier Based on an Intermediate Hybrid Signed-Digit (HSD-1) Representation," *Proc. 14th IEEE Int'l Symp. Computer Arithmetic,* pp. 216-224, Apr. 1999.

[21] N. Ohkubo et al., "A 4.4-ns CMOS 54 $\times$ 54-b Multiplier Using Pass-Transistor Multiplexor," *IEEE J. Solid-State Circuits,* vol. 30, pp. 251-256, Mar. 1995.

[22] T. Akoi, Y. Sawada, and T. Higuchi, "Signed Weight Arithmetic and Its Application to Field Programmable Digital Filter Architecture," *IEICE Trans. Electronics,* vol. E82-C, pp. 1687-1698, Sept. 1997.

Dhananjay Phatak received the BTech degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1985, the MS degree in microwave engineering in 1990, and the PhD degree in computer systems engineering in 1993, both from the Electrical and Computer Engineering Department, University of Massachusetts, Amherst. From 1994 until 2000, he was an assistant professor of electrical engineering at the State University of New York, Binghamton. Since Fall 2000, he has been an associate professor in the Computer Science and Electrical Engineering Department at the University of Maryland Baltimore County (UMBC). His current research interests are in computer arithmetic algorithms and their VLSI implementations, mobile and high-performance computing and networks, digital and analog VLSI design and CAD, and neural networks. In the past, he has worked on microwave and optical integrated circuits. Dr. Phatak has published articles in IEEE transactions in several diverse areas (microwave theory and techniques, neural networks, computers), as well as in other premier journals and conferences in his areas of research. He has been active on technical program committees of conferences in his areas of research. He has obtained research support from the US National Science Foundation (NSF), Lockheed Martin, and AetherSystems Inc. He was a recipient of the NSF's CAREER award in 1999. He has filed for patents based on his work in computer arithmetic algorithms and implementations and on novel codes in CDMA (wireless cellular) systems. He teaches classes in his areas of research and on other topics in computer engineering. He is a member of the IEEE.

Tom Goff received the BS degree in electrical engineering and the MS degree in computer science from the State University of New York at Binghamton in 1997 and 2000, respectively. He is currently pursuing the PhD degree in computer science at the University of Maryland, Baltimore County. His research interests include computer arithmetic algorithms and their implementations, wireless and mobile networking, and distributed computing.

Israel Koren (S'72-M'76-SM'87-F'91) received the BSc, MSc, and DSc degrees from the Technion-Israel Institute of Technology, Haifa, in 1967, 1970, and 1975, respectively, all in electrical engineering. He is currently a professor of electrical and computer engineering at the University of Massachusetts, Amherst. Previously, he was with the Technion-Israel Institute of Technology. He also held visiting positions with the University of California at Berkeley, University of Southern California, Los Angeles, and University of California, Santa Barbara. He has been a consultant to several companies including IBM, Intel, Analog Devices, AMD, Digital Equipment Corp., National Semiconductor, and Tolerant Systems. Dr. Koren's current research interests include techniques for yield and reliability enhancement, fault-tolerant architectures, real-time systems, and computer arithmetic. He has published extensively in several IEEE transactions and has more than 150 publications in refereed journals and conferences. He currently serves on the editorial board of the *IEEE Transactions on VLSI Systems*. He was a co-guest editor for the *IEEE Transactions on Computers* special issue on high yield VLSI systems, April 1989, and the special issue on computer arithmetic, July 2000, and served on the editorial board of these transactions between 1992 and 1997. He has also served as general chair, program chair, and program committee member for numerous conferences. He edited and coauthored the book *Defect and Fault-Tolerance in VLSI Systems,* volume 1 (Plenum, 1989). He is the author of the textbook *Computer Arithmetic Algorithms* (1999). He is a fellow of the IEEE.