High Radix Signed Digit Number Systems: Representation Paradigms

Ghassem Jaberipur* jaberigh@mehr.sharif.edu

Mohammad Ghodsi† ghodsi@sharif.edu

January 27, 2002

Abstract

Redundant signed digit number systems are popular in computationally intensive environments particularly because of their carry-free property which allows for digit-parallel addition. The time required for addition is particularly important because other arithmetic operations heavily depend on it. Signed digit number systems with high radices are of particular interest because of less memory requirement to represent a given number. But, the time required to perform digit-parallel addition is, by a relatively large coefficient, logarithmically proportional to the radix. Reduction of this coefficient is the prime goal of our study in this paper, where we emphasize on least cost implementations. We present a novel modification to the conventional carry-free addition algorithm for signed digit numbers, and investigate the impact of different representations of signed digits on reducing the time required to perform digit parallel addition. Three representation paradigms are considered, namely, signed-magnitude, two's complement, and one's complement. Following the common practice, and in order to achieve better results, we also focus on power-of-two radices. With the new algorithm, the time required to derive the transfer digit reduces to a small constant value which does not depend on the radix.

1 Introduction

Addition is widely recognized as a basis of other arithmetic operations. Adequate redundancy in a number system provides for digit-parallel addition, i.e., digit-wise addition of two numbers with no inter-digit carry propagation. A necessary and sufficient condition for digit-parallel addition has been studied in [7]. The Signed Digit (SD) number system was first introduced by Avizienis [1] where he proved the carry-free property for radix r ($r \geq 3$) SD numbers with a digit set $[-\alpha, \alpha]$. In a number system with the carry-free property, a carry generated in any digit position is absorbed in the next position. In any hardware realization of carry-free addition based on binary adders, a generated carry, in fact, propagates up to the most significant bit of the next digit, i.e., the carry is absorbed by that digit, so we can say that there is no inter-digit carry propagation. Adequate redundancy for the carry-free property is assured by the following constraint on digit values [3]:

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

For example, in the Binary Signed Digit (BSD) number system ($r = 2$) [10], there is not enough redundancy in the digit set $\{-1, 0, 1\}$, to provide for carry-free property. But BSD has the carry limited property [9]. In a number system with carry limited property, a carry generated in any digit position propagates through a limited number of consecutive digit positions. The BSD number system, nevertheless, has been

*Ph.D. student in Sharif University of Technology, and faculty member in Shahid Beheshti University

†Associate professor, Computer Engineering Department, Sharif University of Technology

extensively used for implementing all basic arithmetic operations [2, 8, 12]. The reason is that addition of two BSD numbers is possible with carry propagation limited to two binary digits, hence the possibility of very fast digit-parallel addition. But each binary signed digit is represented by two bits (twice the 1 bit needed to represent an unsigned binary digit), thus in BSD, the extra memory requirement is maximum (100%) as compared to SD systems with higher radices. The Hybrid Signed Digit (HSD) number system provides a framework for trade-off between speed and area (memory requirement) [11]. An HSD number is basically a binary number, except that some positions may as well hold a "-1" value (a BSD position). A carry generated in any position (BSD or binary) may propagate up to the next more significant BSD position. In the regular HSD number systems, the number of binary positions between consecutive BSD positions is constant. The major drawback of the HSD number system is the severe asymmetry that exists between the range of positive and negative values. For this reason, the HSD representation is not considered as one of the paradigms in our study. High Radix Signed Digit (HRSD) number systems have the benefit of lower memory requirement, while providing full symmetry between representable positive and negative values. But, the time required to add two high radix signed digits is by a relatively large coefficient proportional (or logarithmically proportional when a carry accelerating technique [10] is used) to the number of bits in the representation of one digit, where the latter is logarithmically proportional to the radix. We call this coefficient the *high radix coefficient*, and explore the possibilities for reducing it. The relative largeness of the high radix coefficient is due to the complexity of the carry-free addition algorithm [6], which takes several steps to perform the addition. BSD, HRSD, and the regular HSD are all special cases of the Generalized Signed Digit (GSD) number systems that is introduced in [9].

In this paper, our goal is to find the least-cost (i.e., minimal hardware) representation for signed digits with the least possible value for the high radix coefficient. To accurately define what we mean here by a minimal hardware implementation, we define a k-dependent cell as a hardware piece whose delay depends on k (linearly or logarithmically), where each signed digit is assumed to be represented by $(k+1)$ bits. Relevant examples relate to addition, or addition-like operations such as comparison, or zero detection, where all can be implemented by a $(k+1)$-bit (or k- bit in the case of sign-magnitude representation) adder cell. A minimal hardware implementation is one that uses the minimum number of k-dependent cells, where the same cell may be reused as needed. On the other extreme a maximal hardware implementation is one that uses any number of k-dependent cells in parallel, and reuses a k-dependent cell only when it does not increase the total delay. We will show that the value of the high radix coefficient is actually equal to the number of k-dependent cells in the critical path of the implementation. Any implementation may have some condition control circuitry with constant delay (that does not depend on k). We study three different representations for signed digits and introduce a novel modification to the Conventional Carry-Free Addition Algorithm (CCFAA) for HRSD numbers [9]. The organization of the paper is as follows: In Section 2, we note that the CCFAA has four steps, where each step includes a form of addition of two digits (i.e., addition, comparison or zero detection, increment, or decrement). The time required to perform each addition is dependent on the internal hardware representation of the signed digits. To have a basis for cost comparison of the cases studied in this paper, we try to parallel the steps of the CCFAA to the extent possible in Section 2.2. In Section 3, we introduce our modification to the CCFAA and prove its validity. Our novel *Compare with Half Radix Algorithm* (CHRA), introduces some simplifications in the implementation of the carry-free addition algorithm, which leads to the reduction of the high radix coefficient specially in a minimal hardware approach. In Section 4, we examine the sign-magnitude representation of signed-digits, where we show that the value of the high radix coefficient, on a minimal hardware approach is as high as 5. In Section 5, and 6 we show that with two's complement, and one's complement representations of a signed digit, the high radix coefficient can be substantially reduced, without increasing the hardware cost.

2 Conventional Carry-Free Addition Algorithm (CCFAA)

The HRSD number systems provide for carry-free addition. Table 1 depicts the different stages in addition of two HRSD numbers, where r is the radix and α denotes the maximum absolute value for a digit from the digit set $[-\alpha, \alpha]$. The addition algorithm, has four steps (as listed below), where each step may contribute to the value of the high radix coefficient.

	a_{n-1}	$\cdots$	a_1	a_0+	$A = \sum_{i=0}^{n-1} a_i \times r^i$	
	b_{n-1}	$\cdots$	b_1	b_0	$B = \sum_{i=0}^{n-1} b_i \times r^i$	
$P:$	p_{n-1}	$\cdots$	p_1	p_0	$p_i = a_i + b_i$	(1)
$T:$	t_{n-1}	$\cdots$	t_1	$0+$	$\lvert t_{i+1}\rvert = (\lvert p_i\rvert \geq \alpha)$, $\mathrm{sign}(t_{i+1}) = \mathrm{sign}(p_i)$	(2)
$W:$	w_{n-1}	$\cdots$	w_1	w_0	$w_i = p_i - t_{i+1} \times r$	(3)
$S:$	s_{n-1}	$\cdots$	s_1	s_0	$s_i = w_i + t_i$	(4)

Table 1: The CCFAA.

1. Parallel addition of digits in the same position of two n-digit HRSD numbers A and B, which results in the position sum vector P.
2. Comparison of the magnitude of position sum with α in order to derive the transfer vector T, where each transfer belongs to $\{-1, 0, 1\}$, t_0 is assumed to be zero, a nonzero t_n denotes an overflow, and the expression $|t_{i+1}| = (|p_i| \geq \alpha)$ means that if $(|p_i| \geq \alpha)$ then the absolute value of t_{i+1} is 1, otherwise it is 0.
3. Derivation of the interim sum vector W, by possibly adding r or $-r$ to the position sums.
4. Parallel addition of the interim sum vector W and the transfer T which generates the sum vector S. The transfer selection mechanism in step 2, guarantees that no new transfer is generated in this step.

Figure 1 depicts the derivation of t_{i+1} and w_i where t_{i+1} is the transfer to the $(i+1)$th position, w_i is the $(i+1)$th element of the vector W, the solid slopes serve as a graphical representation of Equation (3)

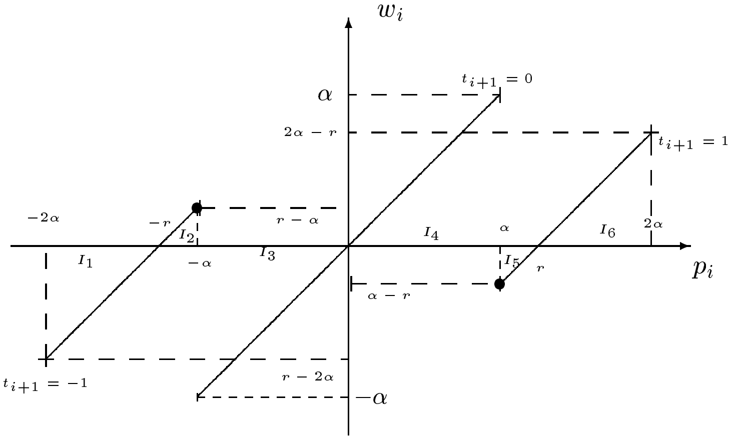

Figure 1: Derivation of t_{i+1} and w_i.

in Table 1, and the interval tags I_1, to I_6 will be referred to later.

2.1 The choice of α and preservation of the digit set $[-\alpha, \alpha]$

For a given radix r, the choice of $\alpha \in [\lceil \frac{r+1}{2} \rceil, r-1]$, provides for several signed digit number systems from the minimally redundant system with the carry-free property ($\alpha = \lceil \frac{r+1}{2} \rceil$), to the maximally redundant ($\alpha = r - 1$) system. The following lemma shows that for the practical case of $r = 2^k (k > 1)$, and also two other impractical cases, the choice of α has no impact on the memory requirement (i.e., the number of bits needed) for representing a signed digit.

Lemma 1 *For $2^k - 2 \le r \le 2^k$, the memory requirement for the digit set $[-\alpha, \alpha]$, does not depend on α.*

Proof. The number of digits in $[-\alpha, \alpha]$, is $2\alpha + 1$. Using the constraint on α (i.e., $\lceil \frac{r+1}{2} \rceil \le \alpha \le r - 1$), we can find the range of $2\alpha + 1$, as $2\lceil \frac{r+1}{2} \rceil + 1 \le 2\alpha + 1 \le 2r - 1$. Combining the latter with the inequalities for r, leads to:

$$2^k \le 2\alpha + 1 \le 2^{k+1} - 1.$$

From the inequalities for $2\alpha + 1$, it is obvious that regardless of the value of α the number of bits needed to represent a signed digit is exactly $k + 1$. □

Next, we study the sources of preservation of the digit set $[-\alpha, \alpha]$ under the carry-free addition algorithm (i.e., the possibility that the range of s_i, is exactly equal to $[-\alpha, \alpha]$):

Lemma 2 *Preserving the digit set $[-\alpha, \alpha]$ under carry-free addition is exclusively due to position sums p_i that satisfy $-\alpha < p_i < \alpha$, except for maximally redundant case ($\alpha = r - 1$), where $|p_i| = 2\alpha$, also leads to $|s_i| = \alpha$.*

Proof. For $-\alpha < p_i < \alpha$, we have $t_{i+1} = 0$, and thus $w_i = p_i$, and $-\alpha + 1 \le w_i \le \alpha - 1$. Therefore, the range of $s_i = w_i + t_i$ (where $t_i \in \{-1, 0, 1\}$), is $[-\alpha, \alpha]$. For $\alpha \le |p_i| \le 2\alpha$, by symmetry, we consider only $\alpha \le p_i \le 2\alpha$, where $t_{i+1} = 1$. We assume $\alpha = r - j$ for $1 \le j \le r - \lceil \frac{r+1}{2} \rceil$, and show that the only value for j, leading to the preservation of the digit set is 1. Substitution of p_i by $w_i + r$, and α by $r - j$ in $\alpha \le p_i \le 2\alpha$ leads to $-j \le w_i \le r - 2j$. Now to let s_i reach α, we should let $max(w_i) = \alpha - 1$, or $r - 2j = r - j - 1$, i.e., $j = 1$. □

2.2 Reduction of the high radix coefficient

Derivation of t_{i+1}, in equation (2) of the CCFAA, involves a comparison operation which generally have the same time complexity as that of an unsigned addition operation. Therefore, four digit-parallel addition-like operations are recognized in Table 1. The time required for each addition is dependent on k ($k = \lceil \log r \rceil$, where the number of bits in one digit is either k, or $k + 1$ depending on the value of α), so is the total addition time of two signed digits. Therefore, we can define the total addition time as a function of k such as $h \times \Delta(k) + c$, where h stands for the high radix coefficient, and c is a constant, which does not depend on k. $\Delta(k)$ may be a linear function of k where each digit-addition is implemented by a carry ripple technique, or may be logarithmic on k, where a carry accelerating technique such as carry look-ahead is used [10].

To reduce the high radix coefficient, an obvious approach is to parallel the steps of the CCFAA to the extent possible, which considerably increases the hardware cost of the implementation. The first, and second steps of the CCFAA (Table 1), cannot be paralleled, for obvious reason. But the rest of the computation can be done at the same time with step 2. The trick is to compute three groups of sum values depending on different values of t_i in parallel. In each group three values are computed in parallel depending on the three possible values of t_{i+1}. The groups for $t_i \in \{-1, 0, 1\}$ are:

$$(p_i - 1, p_i - 1 + r, p_i - 1 - r), (p_i, p_i + r, p_i - r), (p_i + 1, p_i + 1 + r, p_i + 1 - r).$$

In each group, depending on the value of t_{i+1} three different values of the interim sum, are added to t_i. The position sum p_i is computed in step 1, and the other 8 values may be computed in parallel with step 2, by 8 extra adders. Next, one of the groups is selected by the value of t_i, and then the final sum is selected by the value of t_{i+1}. The selection process is done in constant time. Therefore in such a maximal hardware implementation, only steps 1, and 2 contribute to the value of the high radix coefficient. We will show in sections 4, 5, and 6, that contribution of steps 1, and 2 depend on the representation of the signed digits, specially, in sign-magnitude representation when implemented with minimal hardware, the contribution of step 1 is going to be more than 1. But by using considerable extra hardware, it is possible to limit the latter to 1.

To achieve the same effect of reducing the high radix coefficient, but with keeping the hardware cost as low as possible, we follow an algorithm optimization approach. In the next section we introduce our novel algorithm through which the derivation of the transfer in step 2 of the CCFAA, can be done in constant time without using extra hardware. When we consider the impact of different representations of signed digits on the value of the high radix coefficient, we will show that the contribution of derivation of the interim sum in step 3, may also be reduced to zero, again without using any extra hardware. In the following sections we make the following assumptions for convenience and/or efficiency:

- $k = \lceil \log r \rceil$, and $r > 2$, where we assume that each signed digit is represented by $(k+1)$ bits (zero padding or sign extension may be applied if necessary).
- $|p_i| = 2^k u_i + v_i x_i$, where u_i is the most significant bit of $|p_i|$, and $v_i x_i$ is the unsigned binary number composed of v_i, the second most significant bit of $|p_i|$, and x_i representing the $(k-1)$ least significant bits of $|p_i|$ such that $0 \le x_i < 2^{k-1}$.

3 The Compare with Half Radix Algorithm (CHRA)

In the following theorem, we suggest that in step (2) of the CCFAA, $|p_i|$ may be compared with $\lceil r/2 \rceil$, instead of α. Then, for $r = 2^k$, the vector T may be derived with minimal delay after P is computed, such that the high radix coefficient is reduced by 1.

Theorem 1 *In the carry-free addition algorithm, the transfer may be derived by comparing $|p_i|$, with $\lceil r/2 \rceil$ (instead of α), as:*

$$t_{i+1} = \begin{cases} 1 & \textit{when } \lceil r/2 \rceil \le p_i \le 2\alpha \\ -1 & \textit{when } -2\alpha \le p_i \le -\lceil r/2 \rceil \\ 0 & \textit{when } -\lceil r/2 \rceil < p_i < \lceil r/2 \rceil \end{cases} . \tag{1}$$

Proof. It is sufficient to show that $|w_i| < \alpha$ for each of the above three intervals for p_i. Replacing p_i by $w_i + rt_{i+1}$ leads to:

$$-r + \lceil r/2 \rceil \le w_i \le 2\alpha - r, \text{ for } \ t_{i+1} = 1,$$

$$-2\alpha + r \le w_i \le -\lceil r/2 \rceil + r, \text{ for } \ t_{i+1} = -1, \text{ and}$$

$$-\lceil r/2 \rceil < w_i < \lceil r/2 \rceil, \text{ for } \ t_{i+1} = 0.$$

Enforcing $\lceil \frac{r+1}{2} \rceil \le \alpha \le r-1$ in the above inequalities, leads to $|w_i| \le \alpha - 1$, in all the three cases. □

Note that for $|p_i| = \lceil r/2 \rceil$, and for even values of r, $t_{i+1} = 0$ is also valid. We will show later that this imprecision is indeed useful in the two's complement paradigm of representation of signed digits. The CHRA is particularly efficient in practice, where $r = 2^k$.

Corollary 1 *For $r = 2^k$, the transfer is derived, with minimal delay, by comparing p_i with 2^{k-1}, i.e., $|t_{i+1}| = u_i \vee v_i$, and $sign(t_{i+1}) = sign\ (p_i)$.* □

With the CHRA, contrary to Lemma 2, position sum values p_i, satisfying $-\alpha < p_i < \alpha$, do not contribute in preserving the digit set $[-\alpha, \alpha]$, except for the minimally redundant case $\alpha = \lceil \frac{r+1}{2} \rceil$ with odd values of r, which is unfortunately not the case in Corollary 1. But, in the maximally redundant case ($\alpha = r - 1$) the preservation of the digit set $[-\alpha, \alpha]$, always holds by Lemma 2, and the choice of $\alpha = r - 1$, where $2^k - 2 \leq r \leq 2^k$, does not introduce any inefficiency, as compared to less redundant cases (Lemma 1). The latter results are summarized in the following corollary.

Corollary 2 *The compare with half radix algorithm preserves the digit set $[-\alpha, \alpha]$ in the maximally redundant signed digit number systems ($\alpha = r - 1$). Furthermore, for $2^k - 2 \leq r \leq 2^k$, and in particular for the practical case of $r = 2^k$, the choice of $\alpha = r - 1$ does not increase the memory requirement.* □

4 Sign-magnitude representation of HRSD numbers

Addition of two sign-magnitude digits, as described below, involves four steps by itself. All the four steps, in a maximal hardware approach may be paralleled such that the time required for a sign-magnitude addition is in the same order as the single step two's complement addition. In what follows, we consider the impact of the sign-magnitude representation of signed digits on different steps of the CCFAA, together with a time complexity analysis of a sign-magnitude addition.

4.1 Derivation of the position sum

This step of the CCFAA, involves one sign-magnitude addition operation, whose contribution to the value of the high radix coefficient, by the following analysis is 2(1), where the parenthesized figure relates to the maximal hardware approach. This is reflected in the first column, and first row of Table 5.

4.1.1 Sign-magnitude addition

Addition of two sign-magnitude digits involves the following four steps where we assume that each digit is represented by a sign (1 bit), and a k-bit magnitude:

1. Possible complementation of the second operand: If the signs of the two operands are different, the magnitude of the second operand should be complemented before addition. Complementation involves an increment operation which may be deferred to be fused later in step 2 below, as an "always high" carry-in signal. As such, this step does not exclusively contribute in the total time needed for addition of two sign- magnitude digits, except for a sign-bit comparison, and a conditional bit-wise inversion. That is, the contribution does not depend on k.

2. Addition of the magnitudes of the two operands. The contribution of this step to the total addition time depends on k.

3. Possible magnitude comparison of the two operands: If the two operands have different signs, then the sign of the result is the same as that of the operand with larger magnitude. In a minimal hardware approach, we may take advantage of the fact that magnitude comparison is necessary only when the signs are not alike, where the actual operation in Step 2 above is subtraction of magnitudes. For a non-zero result, the operand with larger magnitude can be determined from the subtraction result. For a zero result, the derived sign as such may be positive or negative, but unique zero representation requires a positive sign for zero magnitudes. We therefore need to determine if the subtraction result was zero or not. The time required for zero detection of a k-bit operand depends on k. The latter could be done in parallel with Step 2 [13], but staying with our minimal hardware approach, we can reuse the adder cell of Step 2 for zero detection. The trick is to add $2^k - 1$, to the subtraction result, and check for the carry-out signal. A low signal indicates that the subtraction result was zero. We can conclude now that in a minimal hardware approach, the exclusive elapsed time of this step depends on k.

4. Possible complementation of the result: If the sign of complemented operand in step 1 was originally positive, the result of the addition in Step 2 should be complemented. The contribution of this step in the total addition time normally depends on k. But the post two's complement operation has been reported to be avoidable in [13], without employing any extra k-dependent cell. The trick is to bit-wise complement the result when is necessary, and instead of increment operation as part of complementation, add to it the carry out of the magnitude addition. The latter addition as a sort of end-around-carry addition does not actually introduce another k-dependent operation besides the magnitude addition. Therefore taking advantage of the latter clever technique, the time required for this step is not k-dependent, even in a minimal hardware approach.

Summing up the partial contributions of the steps above in the total sign-magnitude addition time, we conclude that in a minimal hardware approach, two k-dependent addition operations (due to those of Steps 2, and 3 above), contribute in the total addition time, while the k- dependent delay in a maximal hardware approach equals to that of only 1 addition.

4.2 Derivation of the transfer and interim sum

Recalling Equation (2) of Table 1, we note that derivation of the transfer involves a magnitude comparison operation. The comparison operation has the same time complexity as that of a simple unsigned addition, and thus its contribution in the value of the high radix coefficient as reflected in the first column, and second row of Table 5 is 1.

To analyze the time complexity of the derivation of the interim sum by Equation (3) of Table 1, we recognize six cases depending on the six intervals of the values of p_i, denoted by I_1 to I_6, in Figure 1. In each case, as is shown in Table **??**, we can derive w_i, by replacing $2^k u_i + v_i x_i$ for $|p_i|$, and 2^k for r, in $w_i = p_i - rt_{i+1}$, followed by substitution of the related values (with regards to the respected intervals) for u_i and t_{i+1}. The choice of $r = 2^k$, follows the common practice, and simplifies the derivation.

Derivation of the interim sum								
Interval for p_i	p_i	sign(p_i)	u_i	$\lvert t_{i+1}\rvert$	t_{i+1}	w_i	sign(w_i)	$\lvert w_i\rvert$
$I_1 = [-2\alpha, -2^k]$	$-2^k u_i - v_i x_i$	1	1	1	-1	$-v_i x_i$	1	$v_i x_i$
$I_2 = [-2^k + 1, -\alpha]$	$-2^k u_i - v_i x_i$	1	0	1	-1	$-v_i x_i + 2^k$	0	$\overline{v_i x_i} + 1$
$I_3 = [-\alpha + 1, -1]$	$-2^k u_i - v_i x_i$	1	0	0	0	$-v_i x_i$	1	$v_i x_i$
$I_4 = [0, \alpha - 1]$	$2^k u_i + v_i x_i$	0	0	0	0	$v_i x_i$	0	$v_i x_i$
$I_5 = [\alpha, 2^k - 1]$	$2^k u_i + v_i x_i$	0	0	1	1	$v_i x_i - 2^k$	1	$\overline{v_i x_i} + 1$
$I_6 = [2^k, 2\alpha]$	$2^k u_i + v_i x_i$	0	1	1	1	$v_i x_i$	0	$v_i x_i$

Table 2: Summary of the derivation of w_i in the addition of two sign-magnitude signed digits.

In Table 2, we note that w_i is negative only when the number of "1"s in the three columns for sign(p_i), u_i, and $|t_{i+1}|$ is odd, i.e.,

$$\text{sign}(w_i) = \text{sign}(p_i) \oplus u_i \oplus |t_{i+1}| .$$

To find an easy implementation for $|w_i|$, we note in Table 2 that $|w_i| = v_i x_i$, except when $\overline{u_i}$ and $|t_{i+1}|$ are both "1" in which case $|w_i| = \overline{v_i x_i} + 1$, where $\overline{v_i x_i}$ is the bit-wise complement of $v_i x_i$. This observation can be summarized in the equation

$$|w_i| = multiplex(v_i x_i, \overline{u_i}\, |t_{i+1}| , \overline{v_i x_i} + 1),$$

where $multiplex(x, c, y)$ resolves to x when the bit-variable c is "0", and to y otherwise. The increment operation involved in the derivation of $|w_i|$ may be fused in step (4) of the CCFAA. Therefore, this step may be considered as not contributing in the value of the high radix coefficient, even in a minimal hardware approach. Finally step (4) of the CCFAA as a sign-magnitude addition contributes another "2" (1 in the maximal hardware approach) to the value of the high radix coefficient, making h, as reflected in Table 5, equal to 5(2). Applying the CHRA, reduces h, to 4(1).

5 Two's complement representation of high radix signed digits

In this section, we consider representing each signed digit, as a two's complement number. The range $[-2^k, 2^k-1]$, of a $(k+1)$-bit two's complement digit, covers the digit set $[-\alpha, \alpha]$, for $\lceil \frac{r+1}{2} \rceil \leq \alpha \leq r-1$, and $r = 2^k$.

5.1 Derivation of the two's complement position sum

To derive the position sum, we sign-extend (one bit to the left) the two $(k+1)$-bit signed digits represented in two's complement format, and then perform two's complement addition. The result will be a $(k+2)$-bit position sum. The contribution of this operation in the value of the high radix coefficient , as is reflected in the third, and fourth column, and first row of Table 5 is 1.

5.2 Derivation of the transfer, and the two's complement interim sum

The outcome of applying the CHRA on two's complement signed digits (with $r = 2^k$) is shown in Figure 2, and also in Table 3.

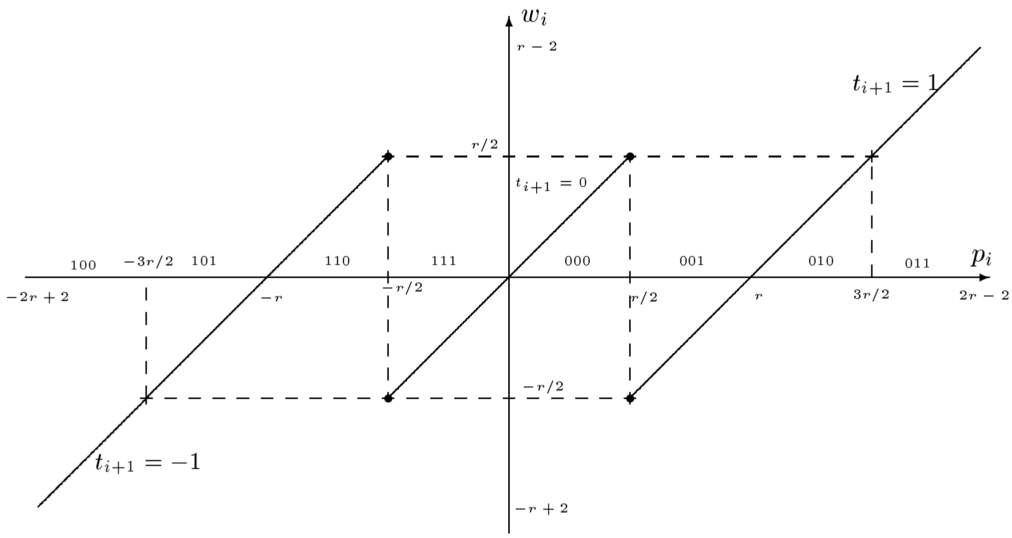

Figure 2: Derivation of t_{i+1} and w_i for two's complement representation.

The figure is drawn for the maximally redundant case $\alpha = r-1$, in which the 3 bit numbers on the intervals for p_i (i.e., sign(p_i), u_i, and v_i), stand for the three most significant bits of p_i. In the table, the columns 2-4 and 7-8 represent the three most significant bits of p_i, and the two most significant bits of w_i respectively, x_i stands for the $(k-1)$ least significant bits of p_i , and the two's complement representation of t_{i+1} is shown in the rightmost two columns, where the superscripts denote the bit positions. Note that, by Theorem 1, the choice of $t_{i+1} = 0$ in the last row of Table 3 includes the point with coordinates $(-r/2, -r/2)$ of Figure 2. As shown below, the latter choice is vital for simplification of the derivation of t_{i+1}. From Table 3, it can be easily verified that the transfer t_{i+1}, can be computed by a simple 3-input/2-output logic, as in the following logical equations:

$$t^1_{i+1} = sign(p_i)\overline{u_i v_i}, \quad t^0_{i+1} = (\overline{sign(p_i)} + \overline{u_i} + \overline{v_i})(sign(p_i) + u_i + v_i).$$

p_i	$sign(p_i)$	u_i	v_i	t_{i+1}	w_i	w_i^k	w_i^{k-1}	t_{i+1}^1	t_{i+1}^0
x_i	0	0	0	0	x_i	0	0	0	0
$2^{k-1}+x_i$	0	0	1	1	$-2^k+2^{k-1}+x_i$	1	1	0	1
2^k+x_i	0	1	0	1	x_i	0	0	0	1
$2^k+2^{k-1}+x_i$	0	1	1	1	$2^{k-1}+x_i$	0	1	0	1
$-2^{k+1}+x_i$	1	0	0	-1	-2^k+x_i	1	0	1	1
$-2^{k+1}+2^{k-1}+x_i$	1	0	1	-1	$-2^k+2^{k-1}+x_i$	1	1	1	1
$-2^{k+1}+2^k+x_i$	1	1	0	-1	x_i	0	0	1	1
$-2^{k+1}+2^k+2^{k-1}+x_i$	1	1	1	0	$-2^k+2^{k-1}+x_i$	1	1	0	0

Table 3: Derivation of w_i, and t_{i+1} in the addition of two's complement signed digits.

The $(k-1)$ least significant bits of w_i is equal to x_i (i.e., the $(k-1)$ least significant bits of p_i), and also $w_i^{k-1} = p_i^{k-1}$, as can be easily seen in Table 3. What remains is w_i^k, which is computable by a simple 3-input logic, implementing the following equation:

$$w_i^k = sign(p_i)\overline{u_i} + sign(p_i)v_i + \overline{u_i}v_i.$$

¿From the above equations, we can see that derivation of the transfer and the interim sum do not contribute to the value of the high radix coefficient, as reflected in the fourth column, and second and third row of Table 5. Finally, s_i can be derived by a simple two's complement increment/decrement logic, whose share in the value of the high radix coefficient is 1. The high radix coefficient for two's complement paradigm with the CCFAA, and the CHRA is thus $h = 3(2)$, and $h = 2(1)$ respectively, where the figures in parenthesis refer to the maximal hardware approach.

6 One's complement representation of signed digits

A signed digit can be represented in one's complement format, pretty much the same as that shown in the previous section for two's complement signed digits. Following the same analysis as in the previous section, derivation of the position sum contributes a "1" to the value of the high radix coefficient. Then, Table 4 resembling the derivation of w_i, and t_{i+1}, has been built up similar to Table 3, where there are two main differences between the two tables. First, one's complement encoding is used for t_{i+1} in the last two columns, and thus derivation of t_{i+1}^0 is simpler as $t_{i+1}^0 = \overline{sign(p_i)}(u_i + v_i)$. Second, the derivation of w_i, as seen in the second row, and the row before last of Table 4, requires an increment/decrement operation. But since t_i is available before it is possible to do the increment/decrement operation on w_i, the increment/decrement may be fused in the computation of $s_i = w_i + t_i$. Therefore, the high radix coefficient in this case is also $h = 3(2)$, and $h = 2(1)$ respectively. The value of the high radix coefficient in one's complement , and two's complement paradigms is the same, but the two's complement representation of signed digits is naturally preferable. The reason is the popularity of the two's complement representation in general, availability of optimized standard adder cells for two's complement binary representation, and the ease of converting widely used two's complement numbers to their signed digit equivalent and vice versa.

7 Conclusions

High radix signed digit number systems exhibit the carry-free property while economizing the memory requirement as compared to lower radix signed digit number systems. In this paper, we introduce the high radix coefficient as a measure for comparing the time required to perform carry-free addition of HRSD numbers with different representations, where we emphasize on least-cost implementations, which

p_i	$sign(p_i)$	u_i	v_i	t_{i+1}	w_i	w_i^k	w_i^{k-1}	t_{i+1}^1	t_{i+1}^0
x_i	0	0	0	0	x_i	0	0	0	0
$2^{k-1}+x_i$	0	0	1	1	$-2^k+1+2^{k-1}+x_i-1$	1	1	0	1
2^k+x_i	0	1	0	1	x_i	0	0	0	1
$2^k+2^{k-1}+x_i$	0	1	1	1	$2^{k-1}+x_i$	0	1	0	1
$-2^{k+1}+1+x_i$	1	0	0	-1	-2^k+1+x_i	1	0	1	0
$-2^{k+1}+1+2^{k-1}+x_i$	1	0	1	-1	$-2^k+1+2^{k-1}+x_i$	1	1	1	0
$-2^{k+1}+1+2^k+x_i$	1	1	0	-1	$1+x_i$	0	0	1	0
$-2^{k+1}+1+2^k+2^{k-1}+x_i$	1	1	1	0	$-2^k+1+2^{k-1}+x_i$	1	1	0	0

Table 4: Derivation of w_i, and t_{i+1} in the addition of one's complement signed digits.

	Sign-Magnitude		Two's Complement		One's Complement	
	CCFAA	CHRA	CCFAA	CHRA	CCFAA	CHRA
Position sum P	2(1)	2(1)	1(1)	1(1)	1(1)	1(1)
Transfer T	1(1)	0(0)	1(1)	0(0)	1(1)	0(0)
Interim sum W	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
Final sum S	2(0)	2(0)	1(0)	1(0)	1(0)	1(0)
High radix coefficient h	5(2)	4(1)	3(2)	2(1)	3(2)	2(1)

Table 5: Contribution of each step of the carry-free addition (Table 1) in the value of the high radix coefficient h, where the parenthesized figures relate to the maximal hardware approach.

is characterized by limiting the number of k-dependent cells to 1. A k-dependent ($k = \lceil \log r \rceil$, and r is the radix of the number system) cell is a $(k+1)$-bit (or k-bit) adder, comparator, or zero detector. We present a modification to the conventional carry-free addition algorithm for HRSD numbers, in order to reduce the high radix coefficient. One of the steps in carry-free addition involves comparing the magnitude of the position sum with the maximum absolute value (α) of the digit set. We present a theorem to prove that the comparison of the magnitude of the position sum with the half-radix $\lceil r/2 \rceil$, instead of α, will produce a valid transfer digit. We show that our modified algorithm, when applied for power-of-two radices ($r = 2^k, k > 1$), simplifies the comparison operation to a constant time derivation of a simple logical equation. We apply the modified algorithm to sign-magnitude, two's complement, and one's complement representations of signed digits, and designate the proposed method the Compare with Half-Radix Algorithm (CHRA). We show that use of the CHRA, with two's complement, or one's complement representation of signed digits in a minimal hardware (least-cost) approach has the same effect on reducing the high radix coefficient, as does the maximal hardware (most costly) implementation of the CCFAA, or CHRA with sign-magnitude representation. We present a comparison table (Table 5) of the application of the CHRA with that of the CCFAA on the three signed digit's representation paradigms studied in this paper, for both minimal hardware, and maximal hardware approaches. The table shows that the two's complement, and one's complement representations with the CHRA and the minimal hardware approach lead to a 60% lower value for the high radix coefficient (reducing from 5 to 2) over the sign-magnitude paradigm with the conventional carry-free addition algorithm. This is achieved for power-of-two radices ($r = 2^k, k > 1$), and the maximally redundant ($\alpha = r-1$) signed digit numbers (with the same memory requirement as any less redundant case), while the digit set $[-\alpha, \alpha]$ is fully preserved. The two's complement paradigm is preferred over one's complement because of the popularity of the two's complement representation in general. Some other even more efficient representation paradigms of signed digits are under investigation [4, 5].

References

[1] Avizienis, A., *Signed-Digit Number Representations for Fast Parallel Arithmetic, IRE Transactions on Electronic Computers*, EC-10:389–400, September 1961.

[2] Ercegovac, M.D., and Lang, T., *Division and Square Root: Digit-Recurrence Algorithms and Implementations,* Kluwer 1994.

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

[4] Jaberipur, G., *High Radix Carry Free Arithmetic: A Proposal for Ph.D. Dissertation, Sharif University of Technology, Department of Computer Engineering,* March 2000.

[5] Jaberipur, G., Parhami, B., and Ghodsi, M., *A Class of Stored-Transfer Representation for Redundant Number Systems,* to appear in *Proc. of the 35th Asilomar Conf. on Signals, Systems, and Computers*, Nov. 4-7, 2001, Pacific Grove, CA.

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

[7] Kornerup, K., *Necessary and Sufficient Conditions for Parallel and Constant Time Conversion and Addition,* in *Proc. 14th IEEE Symposium on Computer Arithmetic*, pp. 152–156, IEEE Computer Society, April 1999.

[8] Kuninobu S., Nishiyama T., Edamatsu H., Tanaguchi T., and Takagi N., *Design of high speed MOS multiplier and divider using redundant binary representation, Proc. Eighth Symp. Computer Arithmetic,* pp. 80-86, Como, Italy, 1987.

[9] Parhami, B., *Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations,* in *IEEE Transactions on Computers*, pp. 89–98. January 1990.

[10] Parhami, B., *Computer Arithmetic, Algorithms and Hardware Designs.* Oxford University Press, 1999.

[11] Phatak, D.S., and Koren, I., *Hybrid Signed Digit Number Systems: A Unified Framework for Redundant Number Representation with Bounded Carry Propagation Chains,* in *IEEE Transactions on Computers*, August 1994, Volume 43, number 8, pp. 880–891.

[12] Srinivas, H.R., and Parhi, K.K., *Computer Arithmetic Architectures with Redundant Number System,* in *Proc. of the 28th Asilomar Conf. on Signals, Systems, and Computers*, pp. 182-186, Oct. 31 - Nov. 2, 1994, Pacific Grove, CA.

[13] Vassiliadis, S., Lemon, D.S., and Putrino, M., *S/370 Sign-Magnitude Floating-Point Adder,* in *IEEE Journal of Solid-State Circuits*, August 1989, Volume 24, number 4, pp. 1062–1070.