Redundancy Management in Arithmetic Processing via Redundant Binary Representations

Dhananjay S. Phatak, Tom Goff
Electrical Engineering Department
State University of New York
Binghamton, NY 13902-6000
phatak@ee.binghamton.edu

Israel Koren
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003

Abstract

It is well known that constant-time addition, in which the execution delay is independent of operand length, is feasible only if the output is expressed in a redundant representation. 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 introduce the notion of "equal-weight grouping" (EWG) wherein, bits having the same weight are grouped together to achieve the constant-time addition and/or simultaneous format conversion operations. We also address some of the issues recently raised in [1] which establishes necessary and sufficient conditions for constant-time addition or format conversion and indicate possible extensions of the theory developed therein.

1 Introduction

If the value of the carry-out c_i from digit position i can be determined by considering only a fixed number of less significant and/or more significant digit positions, then it can be rendered independent of the carry-in, c_{i-1}, making it possible to achieve constant-time addition. This is possible only if the output digit-set at position i is sufficiently redundant. The less significant digit positions considered during constant-time addition constitute a right context or look-back. More significant digit positions form a left context.

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:

$$\mathcal{D}^{(SD)} = \{-1,0,1\} \quad (1)$$

$$\mathcal{D}^{(SD3^{(-)})} = \{-2,-1,0,1\} \quad (2)$$

$$\mathcal{D}^{(SD3^{(+)})} = \{-1,0,1,2\} \quad (3)$$

$$\mathcal{D}^{(CS2)} = \{0,1,2\} \quad (4)$$

$$\mathcal{D}^{(CS3)} = \{0,1,2,3\} \quad (5)$$

We consider two types of number systems based on each of these digit sets; a fully-redundant (FR) system and a partially-redundant (PR) system. A FR system is one in which all digit positions of a number are redundant and the characteristics of such a system are well known [3, 4]. In a PR system only some digit positions are redundant. While the fixed delay for constant-time addition is minimized when the output is expressed in a FR form, PR representations can be used to address other design constraints such as area, power, etc. [5, 6].

In general, it is possible to use different redundant digit sets at different positions but, for the sake of simplicity, we restrict ourselves to representations where all redundant positions use the same digit set. Some possible partially-redundant formats are illustrated in Figure 1. As the figure shows, redundant digits can be spaced at arbitrary positions.

Note that digit sets (1)-(5) all need exactly two bits to represent their values. In essence, we consider representations where some positions are allocated two bits and ask the question: 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 PR systems, we consider those where every k-th digit is redundant. This simplification is merely for the sake of illustration; the results apply when redundant digits are arbitrarily spaced.

1.1 Redundant Binary Encodings

A two-bit binary redundant digit, $\hat{x}_i$, can be thought of as having *higher* and *lower* significant bits, x_i^h and x_i^l re-

0-7803-5700-0/99/$10.00©1999 IEEE

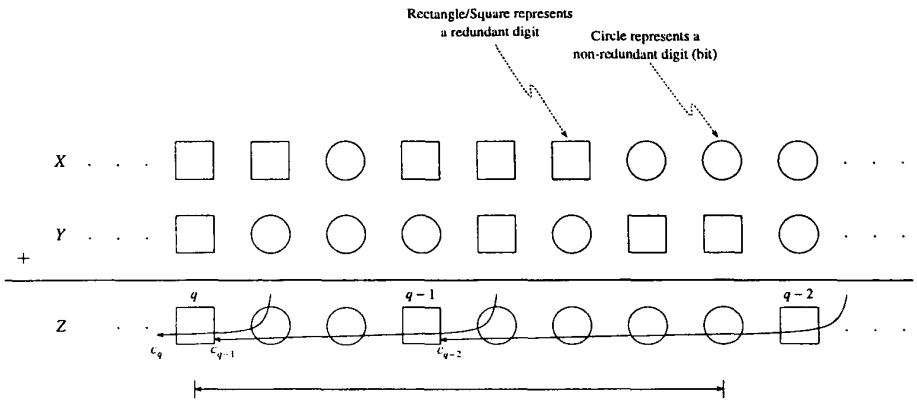

Figure 1. Constant-time addition of partially-redundant operands

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

($SD3^{(\pm)}$ refers to either $SD3^{(+)}$ or $SD3^{(-)}$)

Table 1. Redundant Radix-2 Number Systems

spectively (the hat notation is used to distinguish between redundant and non-redundant digits). Note that arbitrary bit combinations can be used to represent redundant digit values, but we concentrate on encodings that satisfy the relationship:

$$\text{value of digit } \hat{x}_i = \pm 2 \cdot x_i^h \pm x_i^l \tag{6}$$

For the SD and $SD3^{(-)}$ digit set encodings, x_i^h is given a weight of -2 and x_i^l a weight of 1. Thus $\hat{x}_i = -2 \cdot x_i^h + x_i^l$, which is simply a two's complement encoding. The digit set for $SD3^{(+)}$ can be realized by changing the sign of both x_i^h and x_i^l, that is $\hat{x}_i = 2 \cdot x_i^h - x_i^l$. Both encodings are give below.

$$\mathcal{D}^{(SD3^{(-)})}: \quad (0,0) \equiv 0,\ (0,1) \equiv +1,\ (1,1) \equiv -1,\ (1,0) \equiv -2 \tag{7}$$

$$\mathcal{D}^{(SD3^{(+)})}: \quad (0,0) \equiv 0,\ (0,1) \equiv -1,\ (1,1) \equiv +1,\ (1,0) \equiv +2 \tag{8}$$

For carry-save based redundant representations, x_i^h has a weight of $+2$ and x_i^l a weight of 1 making $\hat{x}_i = 2 \cdot x_i^h + x_i^l$, which yields the following encodings:

$$\mathcal{D}^{(CS2)}: \quad (0,0) \equiv 0,\ (0,1) \equiv 1,\ (1,0) \equiv 2,\ (1,1) \text{ not allowed} \tag{9}$$

$$\mathcal{D}^{(CS3)}: \quad (0,0) \equiv 0,\ (0,1) \equiv 1,\ (1,0) \equiv 2,\ (1,1) \equiv 3 \tag{10}$$

1.2 Equal-Weight Grouping

Note that the encoding chosen for both signed-digit and carry-save redundant digits ensures that x_i^l and x_{i-1}^h have the same weight (magnitude) 2^i, i.e., the digits $\hat{x}_i$ and $\hat{x}_{i-1}$ *overlap* each other. When two numbers are added, this overlap can be exploited to reduce the range of digit sums and to predict the range of an incoming carry [7]. Figure 2 shows two redundant digits, $\hat{x}_i$ and $\hat{x}_{i-1}$, of a number X. The arrows indicate the individual bits that make up each redundant digit. The bits x_i^l and x_{i-1}^h both have a weight of 2^i and are grouped and form an alternate interpretation of the bits. 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.

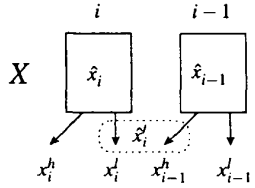

Figure 2. Equal-weight Grouping

2 Characteristics of Constant-Time Addition

To compare the number systems listed in Table 1 we consider two types of constant-time additions. We consider adding two operands of a given format and expressing the

result in that same format. For example, adding two $CS2$ numbers together to form a $CS2$ result ($CS2 + CS2 \rightarrow CS2$). We also consider the possibility of constant-time addition and simultaneous "format conversion". In this case two FR numbers of the same format are added and the result gets expressed in a PR format, for example ($SD + SD \rightarrow SD\_k$) (it can be shown that this leads to the longest context or look-back [7] for a given output format).

Figure 3 illustrates constant-time addition without format conversion. At position i, the EWG digits $\hat{x}_i^l$ and $\hat{y}_i^l$ are added together to form the group sum θ_i. Based on θ_i and the previous group sums $(\theta_{i-1}, \theta_{i-2}, \cdots)$ that make up the position i's right context, a carry-out, c_i, and intermediate sum, σ_i, are chosen which satisfy:

$$\theta_i = x_i^l + y_i^l + x_{i-1}^h + y_{i-1}^h = 2 \cdot c_i + \sigma_i \qquad (11)$$

The final sum, z_i, is then formed by adding σ_i and c_{i-1} which is the carry-in from the previous position:

$$z_i = \sigma_i + c_{i-1} \quad \text{this must not cause an overflow} \qquad (12)$$

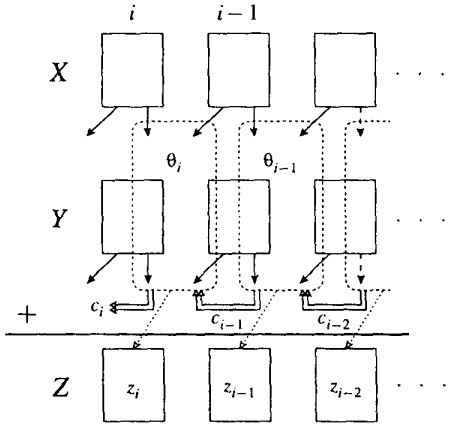

Figure 3. Constant-Time Addition without Format Conversion

The differences between the representations listed in Table 1 can be characterized by the carry-set needed to satisfy (11),(12) and look-back, $\mathcal{L}$, required for constant-time addition.

We now consider a specific example of constant-time addition without format conversion based on the $CS2$ digit set (other cases had to be omitted due to the length constraint. Further details can be found in [7]).

2.1 $CS2$ Addition without Format Conversion

It can be shown that when adding two $CS2$ numbers without format conversion, no look-back (right context) is required and a carry set of $C^{(CS2)} = \{0, 1\}$ is sufficient [7]. The rules which accomplish this addition are shown in Table 2. Note that there is no need to look back at any previous digits, in other words, the look-back is $\mathcal{L} = 0$. As discussed later in Section 4, this operation needs a left-context. Table 2 does not explicitly show a dependence on the left context because the chosen encoding makes it possible to infer appropriate information about the operand digits in adjacent higher significant position from the digits in the current position.

θ_i	$x_i^h + y_i^h$	c_i	σ_i
0	×	0	0
1	×	0	1
2	2	0	2
	otherwise	1	0
3	×	1	1
4	×	1	2

Table 2. Rules for $CS2$ Addition without Format-Conversion

3 Comparison

Table 3 gives a summary of the look-back distances, $\mathcal{L}$, and carry sets needed for the types of redundant binary addition considered. The table clearly shows that equal-weight grouping can lead to smaller carry sets and that the minimum look-back occurs only when EWG is employed.

Among the cases with zero look-back, implementations of those with smaller carry-sets should be more efficient in terms of area and critical path delay. Given this, the carry-save representations ($CS2$ and $CS3$) are more likely to result in better designs than signed digit representations.

Note that Table 3 compares the various representations at an abstract level, in terms of the size of the carry-set and the 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. To this end we designed, layed-out and simulated adder cells for the following cases:

(i) $SD + SD \rightarrow SD$ (The cell in [8] is the most efficient to the best of our knowledge, so we layed out this cell.)

(ii) $SD3^{(-)} + SD3^{(-)} \rightarrow SD3^{(-)}$ (Newly designed [7])

(iii) $CS2 + CS2 \rightarrow CS2$ (Newly designed [7])

(iv) $CS3 + CS3 \rightarrow CS3$ (The extremely efficient 4:2 compressor presented in [9] was used)

For the sake of brevity, the gate diagrams and details of these cells are omitted, those can be found in the references cited.

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

Table 3. Comparison of Binary Constant-Time Addition Techniques

Table 4 shows the SPICE 3f5 simulation results using the TSMC SCN025 0.25 micron technology process with a 2.5 volt supply (which is available from MOSIS). It should be noted that these results are highly layout dependent. These layouts were done to get some idea of the relative comparison of the various redundant adder cells. The critical path simulations indicate that the carry-save representations considered here lead to faster implementations than the signed-digit implementations.

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

Table 4. Critical path from SPICE simulations

In light of the above results, it can be seen that for a multiply operation, using the $CS3$ representation with the compressor presented in [9] is likely to yield the fastest implementations. Note that converting partial products from two's complement format to $CS3$ format is trivial; it requires no gates at all. This is illustrated by Figure 4 which shows that merely grouping the bits appropriately leads to a valid output in the $CS3$ representation.

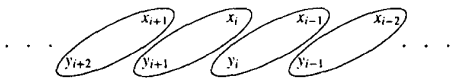

Figure 4. Adding 2's Complement Operands to Generate an Output in $CS3$ Format

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. In effect, multipliers based on these intermediate representations must endure an additional (albeit small) delay at the top level.

Furthermore, the 4:2 compressor that performs $CS3+CS3 \rightarrow CS3$ is smaller and probably faster than other cells. Hence multipliers based on $CS3$ should be faster than those based on other redundant representations.

4 Discussion

In this section we consider some theoretical issues. In particular, we look at some of the results presented in [1] and address an open problem stated therein.

Note that in Table 2, when the sum of operands at position i equals 2, two different carry values can be produced. Since only bits from the current EWG digit appear in this table, it appears that the rules do not imply any context. However, careful scrutiny reveals that there is an implicit left context. This can be understood with the aid of Figure 5.

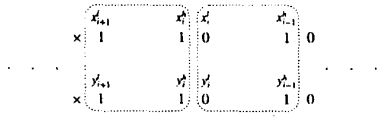

Figure 5. Specific case which the the rules from Table 2 are designed to handle.

This figure illustrates the case when the group sum at the $(i+1)$-th position equals 4. According to the rules from Table 2, this results in an intermediate sum of 2 at position

$i+1$. If position i then generated a carry-out, an overflow would occur at position $i+1$ since 3 is not a member of the $\mathcal{D}^{(CS2)}$ digit set.

This illustrates a simple example of how an overflow at the next significant digit can be avoided by having position i check if x_{i-1}^h and y_{i-1}^h both equal 1 and generate its carry-out accordingly. It can be shown that not generating a carry-out for a group sum of 2 when $x_{i-1}^h + y_{i-1}^h = 2$ is sufficient to avoid all possible overflows at all higher significant positions [7]. Thus, the carry generated by any position i is designed to handle all digit possibilities for the more significant position $(i+1)$, thereby implying a left context.

The more fundamental rules governing $CS2 + CS2 \rightarrow CS2$ constant-time addition which explicitly show the dependence on a left context are shown in Table 5. In fact, Table 2 can be thought of as *derived* from or a special case of Table 5. The reason the left context is not explicitly used in Table 2 is to simplify the implementation.

θ_i	θ_{i+1}	c_i	σ_i
0	×	0	0
1	×	0	1
2	{2,4}	0	2
	otherwise	1	0
3	×	1	1
4	×	1	2

Table 5: *CS2* Addition Rules from Table 2 re-written to explicitly show the dependence on the left context

In [1], it is stated that "so far no example of a complete digit set E has been found, where it was not sufficient to use only the right context for the carry ...". In essence, illustration of a case requiring left context was stated as an open problem. We believe that the *CS2* representation, along with the rules in Table 2 (which is derived from Table 5) constitute an example of a complete digit set which requires a left context, given the minimally sufficient carry-set $\{0,1\}$.

5 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. The comparison revealed that EWG, along with redundant representations based on the carry-save format lead to a smaller carry set, and very likely smaller/faster hardware implementations than those that employ redundant representations based on the signed-digit format. This, in turn, indicates that for word parallel implementations, redundant formats based on the carry-save representation are expected to outperform redundant formats based on signed-digit representations. Representations based on signed digits are likely to be more useful and efficient than those based on carry-save formats only for digit/bit serial applications, where the ability to incrementally approximate the desired result by sequentially outputting both positive and negative digits is indispensable. We also address some of the issues recently raised in [1]; and have proposed a solution to an open problem presented there.

References

[1] P. Kornerup, "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.

[2] P. Kornerup, "Digit-Set Conversions: Generalizations and Applications," *IEEE Transactions on Computers*, vol. C-43, pp. 622–629, May 1994.

[3] B. Parhami, "Generalized signed-digit number systems: a unifying framework for redundant number representations," *IEEE Transactions on Computers*, vol. C-39, pp. 89–98, Jan. 1990.

[4] C. Nagendra, R. M. Owens, and M. J. Irwin, "Unifying Carry-Sum and Signed-Digit Number Representations," Tech. Rep. CSE–96–036, Computer Science and Engineering Department, Pennsylvania State University, 1996.

[5] 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. on Computers*, vol. TC–43, pp. 880–891, Aug. 1994.

[6] J. J. J. Lue and D. S. Phatak, "Area × Delay ($A \cdot T$) Efficient Multiplier Based on an Intermediate Hybrid Signed–Digit (HSD–1) Representation," *Proc. of the 14th IEEE International Symposium on Computer Arithmetic*, pp. 216–224, April 1999.

[7] D. S. Phatak, T. Goff and I. Koren, "On Constant-time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations," *IEEE Transactions on Computers*, submitted.

[8] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi, "Design of high speed MOS multiplier and divider using redundant binary representation," *Proc. of the 8th Symposium on Computer Arithmetic*, pp. 80–86, 1987.

[9] N. Ohkubo and Suzuki, M., et. al., "A 4.4-ns CMOS 54 × 54-b Multiplier Using Pass-Transistor Multiplexor," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 251–256, Mar. 1995.