Zero, Sign, and Overflow Detection Schemes for Generalized Signed-Digit Arithmetic

Behrooz Parhami

Dept. of Electrical & Computer Engineering
University of California
Santa Barbara, CA 93106, USA

ABSTRACT

A generalized signed-digit (GSD) number system uses the digit set $\{-\alpha, -\alpha+1, \ldots, \beta-1, \beta\}$ in radix-r positional notation, with $\alpha, \beta \geq 0$ and $\rho = \alpha+\beta+1-r > 0$. Most GSD number systems support carry-free addition and borrow-free subtraction and even those that do not, can be dealt with using limited-propagation algorithms which yield the ith sum or difference digit as a function of three consecutive digits in each of the operands. Thus, GSD number systems are suitable for realizing high-speed special-purpose arithmetic "engines" in VLSI. To do this, the arithmetic operations of addition and subtraction must be supported by zero, sign, and overflow detection procedures. Algorithms for the implementation of these support functions are presented in this paper.

Index Terms: Asymmetric signed-digit numbers, Computer arithmetic, Number representation, Overflow, Redundant number representation systems, Sign detection, Signed-digit arithmetic, Zero detection.

1. Introduction

For any radix $r \geq 3$, there are one or more *signed-digit* (SD) number representation systems [1]. These *ordinary* SD (OSD) number systems correspond to different values of α in the range $r/2 < \alpha < r$, from the minimally redundant system ($\alpha = \lfloor r/2 \rfloor + 1$) to the maximally redundant one ($\alpha = r-1$), where α determines the set $\{-\alpha, \ldots, -1, 0, 1 \ldots, \alpha\}$ of the $2\alpha + 1$ digit values used. The most important property of OSD number representation systems is the possibility of performing carry-free addition and (by changing all the digit signs in the subtrahend) borrow-free subtraction. The most serious drawbacks of OSD number systems are difficult sign detection (sign of a number is the sign of its most significant nonzero digit) and the overhead for conversion to/from conventional non-redundant representation. The storage overhead, which was once considered to be an important disadvantage of all redundant representations, is now less of an issue because of decreasing hardware costs.

The propagation-free arithmetic property of OSD numbers has caused renewed interest in redundant number representations in view of their suitability for systolic and VLSI implementation (see, e.g., [3]). However, OSD number systems are not the only representations with this property. Previously, the author has defined the class of *generalized signed-digit* (GSD) number systems [5,6] having the digit set $\{-\alpha, -\alpha+1, \ldots, \beta-1, \beta\}$ in radix r with $\alpha \geq 0$, $\beta \geq 0$, and $\rho = \alpha+\beta+1-r$, where $\rho \geq 1$ is the *redundancy index* of the number system. Figure 1 shows that GSD number systems cover all previously known useful redundant number representation systems and also lead to new number systems not examined before. An example of such new representations is the class of stored-carry-or-borrow (SCB) number systems which finds applications in the design of systolic counters [2,4].

It has been shown that any GSD number system with $r>2$ and $\rho \geq 3$ (or with $\rho=2$, provided that $\alpha \neq 1$ and $\beta \neq 1$) supports carry-free addition and that even in the exceptional cases (i.e., for $r=2$, $\rho=1$, or $\rho=2$ with $\alpha=1$ or $\beta=1$) a limited-carry addition algorithm is applicable which yields the ith sum digit s_i as a function of the digits x_i, y_i, x_{i-1}, y_{i-1}, x_{i-2}, and y_{i-2} of the operands x and y [6]. Examples of GSD systems in the latter category are the SC, SB, and SCB representation methods (see Figure 1), including of course their radix-2 special cases. These results can be extended to borrow-free and limited-borrow subtraction, yielding the following two algorithms.

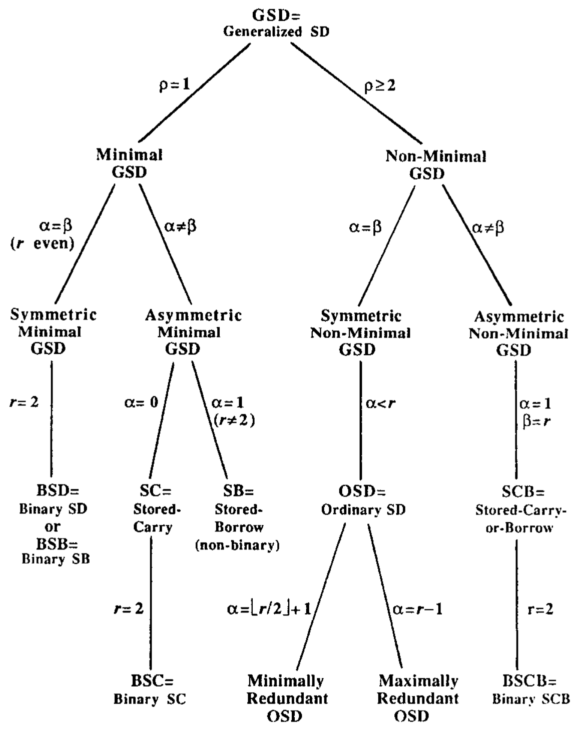

Figure 1. The Hierarchical Relationships of Redundant Number Representation Systems.

Algorithm 1 *(propagation-free addition and subtraction):* Let the two operands have x_i and y_i as the ith digits. For each position i, a *position result* $p_i = x_i \pm y_i$ is computed which is then broken into a *transfer digit* t_{i+1} and an *interim result* $w_i = p_i \mp rt_{i+1}$. The *final result* digit is $z_i = w_i \pm t_i$ whose computation should produce no new transfer. ♦

Algorithm 2 *(limited-propagation addition and subtraction):* Let the two operands have x_i and y_i as the ith digits. In stage 1, for each position i, a *position result* $p_i = x_i \pm y_i$ is computed and used to generate a *range estimate* e_{i+1} for the final transfer digit t_{i+1}. In stage 2, the position result p_i and the range estimate e_i are used to compute a *transfer digit* t_{i+1} and an *interim result* $w_i = p_i \mp rt_{i+1}$. The *final result* digit is $z_i = w_i \pm t_i$ whose computation should produce no new transfer. ♦

The transfer digit t_{i+1} of Algorithms 1 and 2 is in the range

$$-\lambda = -\lceil \alpha / (r-1) \rceil \le t_{i+1} \le \lceil \beta / (r-1) \rceil = \mu \qquad (1)$$

and is selected based on comparing p_i to known comparison constants. The range estimate $e_{i+1} \in$ {low,high} is a binary indicator (selected by comparing p_i to a constant) which restricts the transfer digit t_{i+1} into one of two closed subintervals; the *low* subinterval $[-\lambda, \mu']$ or the *high* subinterval $[-\lambda', \mu]$, where λ' and μ' are constants satisfying:

$$-\lambda < -\lambda' \le \mu' < \mu \qquad (2)$$

Obviously, addition and subtraction must be supported by the auxiliary functions of zero, sign, and overflow detection if high-speed special-purpose VLSI arithmetic "engines" are to be implemented based on the GSD representation. This paper deals with algorithms for the above-mentioned support functions.

2. Zero Detection for GSD Numbers

Whenever zero has the unique all-zeros representation, zero detection for GSD numbers is similar to that of conventional non-redundant positional systems. Thus, we first prove the following uniquness theorem.

Theorem 1: Zero has the unique k-digit all-zeros representation in a GSD number system if and only if at least one of the following four conditions hold: (1) $k = 1$, (2) $\alpha = 0$, (3) $\beta = 0$, (4) $\max(\alpha,\beta) < r$.

Proof: For $k = 1$, uniqueness is obvious. If $\alpha = 0$ ($\beta = 0$), then all the digit values are non-negative (non-positive) and thus zero must be represented by the all-zeros vector. Suppose that $k > 1$, $\alpha > 0$, $\beta > 0$, and $\max(\alpha,\beta) < r$. Then, the value of the number $z_{k-1}z_{k-2} \ldots z_1z_0$ can be written as:

$$\text{value}(z_{k-1}z_{k-2} \ldots z_1z_0) = z_0 + r \times \text{value}(z_{k-1}z_{k-2} \ldots z_1)$$

A necessary condition for the above value to be zero is $z_0 = 0 \bmod r$. Thus, since $\max(\alpha,\beta) < r$, we must have $z_0=0$. We can deduce by induction that $z_1 = \ldots = z_{k-2} = z_{k-1} = 0$. This concludes the sufficiency proof. To show the necessity of at least one of the four conditions, we prove that zero has multiple representations if $k > 1$, $\alpha > 0$, $\beta > 0$, and $\max(\alpha,\beta) \ge r$. If $\alpha \ge r$, then $0\ 0 \ldots 0\ 1\ {-r}$ is an alternate representation of zero. Similarly, if $\beta \ge r$, then $0\ 0 \ldots 0\ {-1}\ r$ has the same value as the all-zeros vector. ♦

Fortunately, even if the conditions of Theorm 1 do not hold, zero detection is not much more difficult. A logic signal ζ_i is propagated from right to left, indicating at each position i, whether the digits to the right represent a multiple of r (i.e., zero with an appropriate transfer digit). The procedure is formalized in the following algorithm.

Algorithm 3 *(zero detection for GSD numbers):* Let the number under test have the k-digit GSD representation $z_{k-1}z_{k-2} \ldots z_1z_0$. Set $z_k = 0$, $\zeta_0 =$ **true**, $t_0 = 0$ and compute sequentially for $i = 0, 1, \ldots, k$:

```
    $u_i := z_i + t_i$ ;
    $t_{i+1}$ := if $u_i = 0 \bmod r$ then $u_i/r$ else anything ;
    $\zeta_{i+1}$ := if $u_i = 0 \bmod r$ then $\zeta_i$ else false .
```

The zero test result is ζ_{k+1}. ♦

Note that in the special cases covered by Theorem 1, we have $t_i = 0$, $u_i = z_i$, and ζ_{i+1}:= **if** z_i=0 **then** ζ_i **else false**. Thus, the test result is the logical AND of *position test* signals which indicate whether $z_i = 0$. For large values of k, this must be implemented as a tree of AND gates due to fan-in limitations. In the general case, transfer-skip and transfer-lookahead techniques can be used in much the same way as conventional carry-skip and carry-lookahead to speed up the hardware realization of Algorithm 3.

Example 1: That the radix-10 SCB number 0 0 –1 9 9 9 9 10 (see Figure 1) represents zero is established by Algorithm 3 as follows:

i :	9	8	7	6	5	4	3	2	1	0
z_i :		0	0	0	–1	9	9	9	9	10
u_i :		0	0	0	0	10	10	10	10	10
t_i :	0	0	0	0	1	1	1	1	1	0
ζ_i :	**true**	**true**	**true**	**true**	**true**	**true**	**true**	**true**	**true**	**true**

We have ζ_9 = **true**. Thus, the zero test result is positive. ♦

Let the range of t_{i+1} in Algorithm 3 be $-\lambda_z \le t_{i+1} \le \mu_z$. To find minimal values for λ_z and μ_z, we first note that:

$$-(\alpha + \lambda_z) \le u_i \le \beta + \mu_z$$

If $u_i \ne 0 \bmod r$, then the value of t_{i+1} is immaterial. So, we assume that $u_i = rt_{i+1}$. Thus:

$$-(\alpha + \lambda_z) / r \le t_{i+1} \le (\beta + \mu_z) / r$$

This yields:

$$\lambda_z^{min} = \lfloor (\alpha + \lambda_z^{min}) / r \rfloor \quad \Rightarrow \quad \lambda_z^{min} = \lfloor \alpha / (r-1) \rfloor \qquad (3)$$

$$\mu_z^{min} = \lfloor (\beta + \mu_z^{min}) / r \rfloor \quad \Rightarrow \quad \mu_z^{min} = \lfloor \beta / (r-1) \rfloor \qquad (4)$$

Note that the values of λ_z^{min} and μ_z^{min} given by (3) and (4) are less than or equal to the corresponding λ^{min} and μ^{min} values given by (1) for addition and subtraction.

3. Sign Detection for GSD Numbers

Unlike OSD numbers, the sign of the most significant nonzero digit of a GSD number does not necessarily indicate the sign of the number. For example, the BSCB number (see Figure 1) 0 0 –1 2 1 is positive. Obviously, the sign detection problem arises only if $\alpha > 0$ and $\beta > 0$.

Theorem 2: The sign of a GSD number is given by the sign of its most significant nonzero digit if and only if $\max(\alpha,\beta) < r$.

Proof: Suppose that the most significant nonzero digit of the GSD number $y_{k-1}y_{k-2} \ldots y_1y_0$ is y_j $(0 \le j \le k-1)$ and that it is positive. Then, the minimum value that the number can have is:

$$\text{value}(1 \underbrace{-\alpha \ \ldots -\alpha \ -\alpha}_{j \text{ digits}}) = r^j - \alpha(r^{j-1} + \ldots + r + 1) = [\alpha + r^j(r-1-\alpha)]/(r-1)$$

Clearly, the above value is positive for $\alpha \le r-1$. Similarly, if the most significant nonzero digit is negative and $\beta \le r-1$, the number is bound to be negative. This concludes the sufficiency proof. To prove the necessity, we show that with $\max(\alpha,\beta) \ge r$, we have positive (negative) numbers with leading negative (positive) digits. If $\alpha \ge r$, we have value($1\ {-r}\ {-1}$) = –1. Similarly, for $\beta \ge r$, value($-1\ r\ 1$) = 1. ♦

Thus, in special cases which satisfy the condition of Theorem 2, sign detection for GSD numbers is no more difficult than that for OSD numbers. As was the case for zero detection, a simple extension of the right-to-left scan will allow us to detect the sign of a GSD number in all cases.

Algorithm 4 *(sign detection for GSD numbers):* Let the number under test have the k-digit GSD representation $z_{k-1}z_{k-2} \dots z_1z_0$. Set $z_k = 0$, $\sigma_0 = \textbf{pos}$, $t_0 = 0$ and compute sequentially for $i = 0, 1, \dots, k$:

```
$u_i := z_i + t_i$ ;
$t_{i+1} :=$ if $u_i \geq 0$ then $\lfloor u_i/r \rfloor$ else $\lceil u_i/r \rceil$ ;
$v_i := u_i - rt_{i+1}$ ;
$\sigma_{i+1} :=$ if $v_i = 0$ then $\sigma_i$ else if $v_i > 0$ then pos else neg.
```

The sign test result is σ_{k+1}. ♦

Note that in the special cases covered by Theorem 2, we have $t_i = 0$, $v_i = u_i = z_i$, and σ_{i+1} is determined by σ_i and the sign of z_i. Again, transfer-skip and transfer-lookahead techniques can be used in much the same way as conventional carry-skip and carry-lookahead to speed up the hardware realization of Algorithm 4.

Example 2: That the radix-10 SCB number (see Figure 1) 0 0 −1 9 9 10 9 10 represents a positive number is established by Algorithm 4 as follows:

i:	9	8	7	6	5	4	3	2	1	0
z_i:		0	0	0	−1	9	9	10	9	10
u_i:		0	0	0	0	10	10	11	10	10
t_i:	0	0	0	0	1	1	1	1	1	0
v_i:		0	0	0	0	0	0	1	0	0
σ_i:	pos	pos	pos	pos	pos	pos	pos	pos	pos	pos

We have $\sigma_9 = \textbf{pos}$. Thus, the number is positive. ♦

It is interesting to note that Algorithms 3 and 4 can be combined into a single algorithm whose hardware realization is only slightly more complex than either algorithm alone. Thus, the overall hardware requirement is reduced by using the following version of Algorithm 4 with three-valued σ_i signals ($\sigma_i \in \{-1,0,1\}$) for zero and sign detection.

Algorithm 5 *(combined zero and sign detection for GSD numbers):* Let the number under test have the k-digit GSD representation $z_{k-1}z_{k-2} \dots z_1z_0$. Set $z_k = 0$, $\sigma_0 = 0$, $t_0 = 0$ and compute sequentially for $i = 0, 1, \dots, k$:

```
$u_i := z_i + t_i$ ;
$t_{i+1} :=$ if $u_i \geq 0$ then $\lfloor u_i/r \rfloor$ else $\lceil u_i/r \rceil$ ;
$v_i := u_i - rt_{i+1}$ ;
$\sigma_{i+1} :=$ if $v_i = 0$ then $\sigma_i$ else signum($v_i$) .
```

The three-valued test result is σ_{k+1}, whose interpretation is the same as that of the signum function used above (i.e., −1:negative, 0:zero, +1:positive).♦

The range of t_{i+1} in Algorithms 4 and 5 is the same as that obtained for Algorithm 3 at the end of Section 2.

4. Detection of Overflow in GSD Arithmetic

In dealing with fixed-length k-digit numbers, an *apparent overflow* occurs when the outgoing transfer digit t_k is nonzero. We call this an "apparent overflow" since even with $t_k \neq 0$, the result of the arithmetic operation may be representable as a k-digit GSD number. The apparent overflow simply signifies that a particular representation of the result has more than k digits. When the result has no k-digit GSD representation, we say that a *real overflow* has occurred. Obviously, real overflow is much harder to detect than apparent overflow.

Fortunately, however, it is not necessary for the hardware to detect real overflow. In a special-purpose system (e.g., a systolic arithmetic engine), an intermediate result with apparent overflow can be tagged as invalid for the rest of the computation. While it is true that with this approach, more results are tagged than what is absolutely necessary, overflows are so infrequent that no serious problem is encountered. A partial remedy is to maintain a guard digit on the left which acts as the "invalid" tag when not zero. Frequently, it is possible to use idle cycles to reset a nonzero guard digit to 0 by changing the most significant digit of the number (e.g., changing 1 −6 to 0 4 in decimal), thus enabling the computation to continue. At any rate, the net effect of using apparent overflows instead of real overflows is that the range of numbers that we can deal with is occasionally restricted.

On the other hand, if overflow is dealt with by using an exception-handling software routine, there is no problem at all since the value of t_k and the *apparent result* $x_{k-1}x_{k-2} \dots x_1x_0$ (which, as in the case of conventional overflow, differs from the correct result by a multiple of r^k) provide sufficient information for the overflow handling routine to deal with the situation. An attempt to obtain a valid k-digit representation of the result may constitute part of this software routine.

The detection of real overflow and the correction procedure in the case of a non-real overflow are discussed next. In the rest of this section, we will assume t_k to be an additive transfer digit, so that $t_kx_{k-1}x_{k-2} \dots x_1x_0$ is a correct (k+1)-digit representation of the result. Obviously, for a borrow-type transfer digit, the sign of t_k must be changed if the subsequent results are to be applicable.

Algorithm 6 *(detection of real overflow):* Given that an outgoing transfer digit $t_k \neq 0$ has been produced with the apparent result $x_{k-1}x_{k-2} \dots x_1x_0$, set $t'_0 = 0$ and compute sequentially for $i = 0, 1, \dots, k-1$:

```
$u_i := x_i + t'_i$ ;
$t'_{i+1} :=$ if $t_k > 0$ then $-\lfloor(\beta - u_i)/r\rfloor$ else $\lfloor(u_i + \alpha)/r\rfloor$ ;
$x'_i := u_i - rt'_{i+1}$ .
```

Then compute $u_k := t_k + t'_k$. There is real overflow iff $t_k \times u_k > 0$. ♦

Note that the computation of x'_i in Algorithm 6 is redundant. The reason for its inclusion is that it simplifies the description of Algorithm 7, as we will see shortly. An intuitive explanation of Algorithm 6 is that it obtains, in a right-to-left sequential scan, the largest (smallest) possible digits in the range $-\alpha \leq x'_i \leq \beta$ that can be used for representing the result producing $t_k > 0$ ($t_k < 0$). If even with these digits, the outgoing transfer digit u_k is nonzero and of the same sign as t_k, then real overflow has occurred.

Example 3: Consider a GSD system with $r = 10$, $\alpha = 5$, and $\beta = 10$. The minuend 6 5 5 5 and the subtrahend −5 5 3 4 yield the apparent difference 1 0 2 1 with the outgoing additive transfer digit $t_4 = 1$. Algorithm 6 works as follows:

i:	4	3	2	1	0
x_i:		1	0	2	1
u_i:		0	0	2	1
t'_i:	−1	−1	0	0	0
x'_i:		10	10	2	1

Since $u_4 = t_4 + t'_4 = 0$, the overflow is not real. ♦

In executing Algorithm 6, the sign of all transfer digits is opposite that of t_k. The magnitude of the transfer digit t'_j satisfies:

$$|t'_j| \leq \lfloor(\alpha + \beta)/r\rfloor = 1 + \lfloor(\rho - 1)/r\rfloor \tag{5}$$

Thus, for $\rho \leq r$, a binary transfer digit is sufficient. It is also interesting to note that for non-redundant representations $\alpha + \beta + 1 = r$. Thus, all transfer digits are zero and an apparent overflow is always a real overflow.

Algorithm 7 *(correction of result with non-real overflow):* After executing Algorithm 6 and determining the values of u_k and x'_i for $i=0,1,\ldots,k-1$, set $t''_k = u_k$ and compute sequentially for $i = k-1, \ldots, 1, 0$:

```
    $v_i := x'_i + rt''_{i+1}$ ;
    $t''_i :=$ if $v_i > \beta$ then $v_i - \beta$ else if $v_i < -\alpha$ then $v_i + \alpha$ else $0$ ;
    $x''_i := v_i - t''_i$ ;
```

The corrected result is $x''_{k-1}x''_{k-2} \ldots x''_1x''_0$. Note that if $t''_i=0$, then $t''_j=0$ and $x''_j = x'_j$ for all $j < i$. Thus, execution of Algorithm 7 can end as soon as t''_i becomes zero. ♦

To find the range of transfer digits in Algorithm 7, we proceed as follows. Let $t_k > 0$. Then, we must have $u_k \le 0$ for the overflow to be non-real. Thus, all transfer digits t''_j in Algorithm 7 are non-positive. Similarly if $t_k<0$, all transfer digits t''_j will be non-negative. Since the sign of t''_j is determined by the sign of t_k, we need only represent its magnitude. In the following discussion, we take $t_k < 0$ and $u_k, t''_j \ge 0$. Let the range of t''_j be $[0,\theta]$. We have:

$$x''_i = v_i - t''_i = x'_i + rt''_{i+1} - t''_i \tag{6}$$

Combining (6) with the restriction $x''_i \le \beta$, we get:

$$t''_{i+1} \le (\beta - x'_i + t''_i) / r \tag{7}$$

We have $x'_i \ge -\alpha$ and $t''_i \le \theta$. Therefore, the right hand side of (7) is no greater than $(\alpha + \beta + \theta)/r$. So, we must have $\theta \le (\alpha + \beta + \theta)/r$, and this is equivalent to:

$$\theta = \lfloor (\alpha + \beta) / r \rfloor = 1 + \lfloor (\rho - 1) / r \rfloor \tag{8}$$

So the range of t''_j is the same as the range of t'_j as specified by (5). As an example, $r = 10$, $\alpha = 5$, and $\beta = 10$ yield $\theta = 1$.

5. Conclusion

We have considered zero, sign, and overflow detection schemes for generalized signed-digit arithmetic. Using these results along with previously obtained high-speed addition and subtraction algorithms, it is possible to design special-purpose arithmetic "engines" for processing of large volumes of numerical data, provided that the nature of computations is such that the conversion and reconversion overhead is amortized over a long sequence of arithmetic operations. Such conditions prevail in many signal processing algorithms and high-precision scientific applications.

It is natural to ask at this point whether the generalization from OSD to GSD number representation is worthwhile. In other words, is it ever advantageous to use an asymmetric rather that a symmetric digit set for a redundant number system? The answer to this question is positive, as evidenced from applications of the SC and SCB number systems. One might try to justify this generalization by asking the equivalent question: Are GSD number systems with asymmetric digit sets any more difficult to deal with than OSD number systems? Clearly the general addition and subtraction algorithms are no more complex than their counterparts for OSD representation. The complexity does increase if ρ exceeds $r - 1$, but this is not related to the symmetry or asymmetry of the digit set used.

Sign detection and overflow handling are difficult for all redundant number representations, independent of the digit set used. Theorem 1 indicates that zero detection can be equally simple for asymmetric digit sets, provided that certain conditions are satisfied. The most serious objection to the use of an asymmetric digit set is the increased complexity of negation (sign change), which is required if addition and subtraction hardware are to be shared. It has been shown that the increased complexity is small if a GSD number system satisfies the condition $\alpha = \beta \bmod (r - 1)$. Even if the above condition is not satisfied, negation is done totally in parallel with negligible speed penalty.

Acknowledgement

The research reported in this paper was carried out while the author was a Visiting Professor with the School of Computer Science, Carleton University, Ottawa, Canada.

References

[1] Avizienis, A., "Signed-Digit Number Representation for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers*, Vol. EC-10, pp. 389-400, 1961.

[2] Guibas, L.J. and F.M. Liang, "Systolic Stacks, Queues, and Counters," *Proc. of the Conf. on Advanced Research in VLSI*, MIT, 1982, pp. 155-164.

[3] Irwin, M.J. and R.M. Owens, "Digit-Pipelined Arithmetic as Illustrated by the Paste-Up System," *Computer*, Vol. 20, No. 4, pp. 61-73, Apr. 1987.

[4] Parhami, B., "Systolic Up/Down Counters with Zero and Sign Detection," *Proc. of the Symp. on Computer Arithmetic*, Como, Italy, May 1987, pp. 174-178.

[5] Parhami, B., "A General Theory of Carry-Free and Limited-Carry Computer Arithmetic," *Proc. of the Canadian Conf. on VLSI*, Winnipeg, Canada, Oct. 1987, pp. 167-172.

[6] Parhami, B., "Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations," to appear in *IEEE Transactions on Computers*.