Reviewing High-Radix Signed-Digit Adders

Peter Kornerup
University of Southern Denmark

***Abstract*—Traditionally higher radix values of the form $\beta = 2^r$ have been employed for recoding of multipliers, and for determining quotient- and root-digits in iterative division and square root algorithms, usually only for quite moderate values of r, like 2 or 3. For fast additions, in particular for the accumulation of many terms, generally redundant representations are employed, most often in binary carry-save or borrow-save, but in a number of publications it has been suggested to recode the addends into a higher radix. It is shown that there are no speed advantages in doing so if the radix is a power of 2, on the contrary there are significant savings in using standard 4-to-2 adders, even saving half of the operations in multi-operand addition.**

Keywords: Multi-operand addition, signed-digit, carry-save, high-radix

I. Introduction

In the implementation of multiplication, by recoding the binary multiplier into a higher radix $\beta = 2^r$, the number of multiplicand multiples to be accumulated is reduced by a factor r. However, the advantage of recoding the multiplier into a higher radix than 4 is quite limited, due to the problem of generating multiplicand multiples. Generally accumulation is performed in a tree structure using redundant representations, and thus in time proportional to the logarithm of the number of addends.

In division and square root algorithms with digit-wise quotient or root determination, the number of algorithm cycles is reduced by a factor of r, corresponding to the number of digits to be determined being reduced. Updating the remainder requires fast subtraction, generally performed in some redundant binary representation.

In these algorithms, and in many other, there is a need for repeated additions or subtractions. Traditionally the redundant binary carry-save or borrow-save representations are being employed, allowing constant-time operations. Such additions are often denoted "carry-free additions", although they are not really free of carries, but to emphasize that carries only travel into a neighbor position, see e.g., [Avi61]. Note that in such applications the encoding of output digits is supposed to be the same as that of the input digits, to allow for repeated additions.

The use of high-radix, signed-digit redundant digit sets for addition in radix values of the form 2^r for $r \geq 2$ has over the years been proposed and investigated in a number of publications, e.g., [Par93], [MI99], [JPG05], [JP07], [JG10], [GJ11], which we shall analyze here. As the addition takes place in digit parallel, the complete addition takes place in time independent of the number of digits of the operands. Employing for radix $\beta = 2^r$ the maximally redundant digit set $D = \{-2^r + 1, \cdots, 0, \cdots, 2^r - 1\}$, the authors encode the digits as 2's complement numbers in $r + 1$ bits, where the leading bit has negative weight -2^r. Each sum digit is rewritten by emitting a carry/transfer-digit[1] in $\{-1, 0, 1\}$, and an incoming carry is added to the modified digit. The authors claim to obtain high-speed addition ([JP07] even claims "Ultrahigh-Speed" in the title), yet they never compare the speed against adders based on carry-save or borrow-save binary digit encodings, as generally employed in multiplier implementations and many other applications.

We shall here analyze such radix 2^r, $r \geq 2$, addition algorithms, in particular the area and timing of the digit operations, and how these parameters depend on the chosen radix, the encoding of digits and carries. We compare such adders to equivalent adders based on combining r redundant radix-2 borrow-save (signed digit) adders over the digit set $\{-1, 0, 1\}$ and carry-save adders over the digit set $\{0, 1, 2\}$. These digit sets have been extensively used in multipliers as they allow more regular binary tree structures than tree structures based directly on 3-to-2 full-adders. These encodings also allows forming the sum of two standard binary numbers by simply pairing bits of the operands, to yield the encoding of the sum digits, thus in zero time and logic, approximately halving the area of a multi-operand addition.

Section 2 describes in fairly general terms the process of performing in the fastest possible way multi-operand addition for any general radix, and in particular for radix 2. In Section 3 we describe the digit-wise addition process employing a maximally redundant digit set representation, for the case of radix 2^r, $r \geq 2$, as well as for the binary case of $r = 1$. Section 4 describes more details of the implementation of these binary redundant digit-adders, noting that there is essentially no difference between use of carry-save and borrow-save representations in the implementation of multi-operand addition. Next Section 5 provides details of some maximally redundant radix 2^r adders, showing diagrams exemplified for radix 16. It is concluded that there is no speed advantage in recoding into a higher radix, unless the digits themselves are redundantly represented. Section 6 briefly concludes the paper.

[1]We will in general use the notation "carry" for a transfer digit, independent of its sign

II. Multi-operand addition

When adding a multiple of operands, say k of n digits, the fastest possible way to do it is to add them in a binary tree, each addition performed using a redundant representation of the intermediate sums, to allow constant time addition at the nodes of the tree, with bounded carry-propagation between positions, i.e, without any "ripple-effect". We will here concentrate on adding a pair of digits from two operands, where it alternatively is possible to accumulate several digits of the same weight in some redundant representation, and only at the end converting the digit sums (as over-redundant digits) back into the wanted digit set, as suggested in [KS05] for decimal multi-operand addition.

Assuming that the initial operands are in some non-redundant representation, at most needing some constant time conversion or recoding of the non-redundant representation, the leaves of the tree must be able to add two such addends, with their sum in the chosen redundant representation to be employed internally in the tree. At the root of the tree, the final result must in general be converted back to some non-redundant representation, most likely the same as that of the original operands. The accumulation of k, n-digit operands can thus be performed in time O($\log\ k$), with final conversion in time O($\log\ n$).

With radix $\beta \geq 2$, when the operand representation is employing the standard non-redundant digit set $D = \{0, 1, \cdots, \beta-1\}$, then very often the symmetric, maximally redundant digit set $D' = \{-\beta+1, \cdots, -1, 0, 1, \cdots, \beta-1\}$ is used internally at the nodes, since then $d \in D \Rightarrow d \in D'$, and at most a simple modification of the digit encoding is necessary at the leaf nodes. This also allows for some simple handling of sign-magnitude or complement representations of operands.

Although this discussion applies to any value of $\beta \geq 2$, we will restrict the detailled analysis to situations where β is a power of 2.

For $\beta = 2$ the situation is particularly simple, as a pairing of the bits of two operands provides an encoding of either their sum or difference in a redundant representation at the leaf nodes, to be used at the internal nodes of the tree. Let $x = \sum_0^{n-1} x_i 2^i$ and $y = \sum_0^{n-1} y_i 2^i$ be two binary integers, then $x - y = \sum_0^{n-1} d_i 2^i$ with $d_i = x_i - y_i$ being a digit in the redundant digit set $\{-1, 0, 1\}$. The pair of bits (y_i, x_i) then provides an encoding of the digit d_i, where x_i has positive weight and y_i has negative weight. If the sum of the operands is wanted, y can easily be negated by inversion. We denote this the borrow-save encoding, and use the notation $d_i \sim (d_i^n, d_i^p)$ with the component of negative weight in the first position.

Alternatively with the same operands, $x+y = \sum_0^{n-1} s_i 2^i$ where $s_i = x_i + y_i \in \{0, 1, 2\}$, representing the digit sum, the digits can be encoded as the pairs $s_i \sim (x_i, y_i)$. This carry-save encoding can be employed in the rest of the tree. But since there is no principal difference between the use of the carry-save and the borrow-save representation, we will generally consider the latter.

The actual implementation of the internal radix 2, 4-to-2 addition nodes of the tree will be described in Section 4 below, but note that addition at the leaf nodes essentially comes for free, thus halving the number of operations to be performed, as well as saving space.

III. The digit-wise addition process

Adding two numbers digit-wise in the maximally redundant set $D = \{-2^r+1, \cdots, 0, \cdots, 2^r-1\}$ for radix $\beta = 2^r$, $r \geq 2$, generates first an intermediate sum represented in the digit set $D' = \{-2^{r+1}+2, \cdots, 2^{r+1}-2\}$, from which carries in the set $C = \{-1, 0, 1\}$ are extracted, leaving behind digits from a much reduced digit set $D" = \{-2^r+2, \cdots, 2^r-2\}$. Note that this digit set must contain at least 2^r different values. In a subsequent step incoming carries are added into the positions, such that all digits now are in the same digit set D as the operands. Addition thus consists in a number of parallel digit-wise additions followed by a digit set conversion, in three sequential steps, as shown in Figure 1.

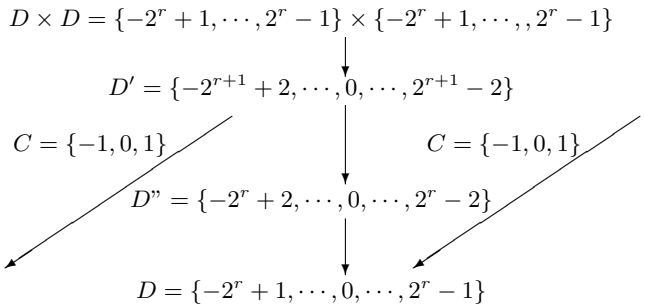
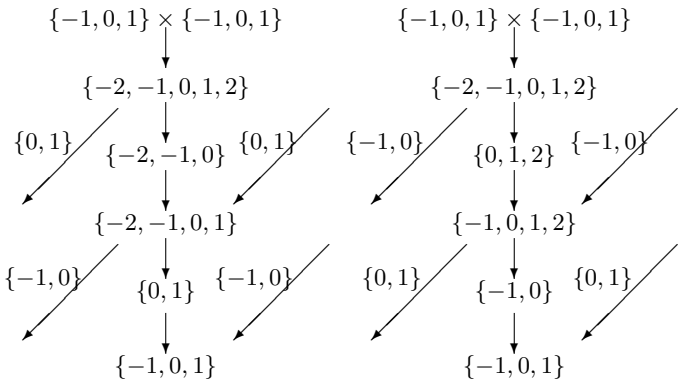


Figure 1. Maximally redundant radix 2^r addition for $r \geq 2$

The first step consists in the digit-wise addition, the next in extracting the outgoing carry and modifying the digit accordingly. The last step is adding an incoming carry to the modified digit. The essential thing to observe is that the outgoing carry is independent of the incoming.

Note that this process is possible for any value of $r \geq 2$, but not for $r = 1$, the binary borrow-save representation with $D = \{-1, 0, 1\}$, as we shall discuss below. Also observe that it involves three sequential digit-wise add or subtract operations. However, the carry-extraction takes place at the most-significant end of the digit, and is thus in general faster than the add operations. Unless the encoding of the digit values themselves is redundant, these operations will be slower the larger the digit set is.

For radix 2, with the digit set $\{-1, 0, 1\}$ the situation seems more complicated. Following Avizienis [Avi61], the digit set conversion here must take place in two phases for a total of five steps, which takes two forms depending on the sign of the incoming carry:



The process is most easily described using addition tables, employing what is denoted "limited carry" in [Par90] or "carry anticipation" in [KM10]. Each digit position determines whether the incoming carry is in $\{-1, 0\}$ or $\{0, 1\}$, and decides between two different ways of adding the two operands, as described in Table I, that is between the use of two different addition tables α^- and α^+, where the α-tables describe as a pair cd the emission of carry c and new digit d, respectively for an incoming non-positive or non-negative carry, and the γ-table describes the absorption of carries, where no further carries are generated.

α^-	-1	0	1
-1	$\bar{1}0$	$\bar{1}1$	00
0	$\bar{1}1$	00	01
1	00	01	10

α^+	-1	0	1
-1	$\bar{1}0$	$0\bar{1}$	00
0	$0\bar{1}$	00	$1\bar{1}$
1	00	$1\bar{1}$	10

γ	-1	0	1
-1		$0\bar{1}$	00
0	$0\bar{1}$	00	01
1	00	01	

Table I
ADDITION TABLES FOR REDUNDANT BORROW-SAVE ADDITION

Thus addition of operands over the digit set $\{-1, 0, 1\}$ can now be implemented in radix 2 for $r = 1$ in three steps as above for $r \geq 2$, it just requires some information about the incoming carry, which turns out to be easily obtainable. This kind of addition has been extensively employed in multiplication when accumulating the partial products in a binary tree since [TYY85], [HNN+87], but the equivalent addition in carry-save over the digit set $\{0, 1, 2\}$ has been used since [Wei81].

IV. RADIX 2, 4-TO-2 ADDITION

Starting with the redundant radix 2 addition, several possible (e.g., two-bit [PGK01], and even three-bit [EL97]) encodings of the digits are possible, some of which were investigated in [Kor05]. All of these were shown to be feasible for addition in what has been denoted 4-to-2 adders, realizable by simple modifications of a carry-save, 4-to-2 adder for the addition of two operands over the digit set $\{0, 1, 2\}$, as shown in Fig. 2, using the two-bit encoding with the digit value being the sum of the encoding bits. This type of adder was originally proposed by Weinburger in [Wei81]. There are several possible implementations of it, including some very efficient ones based on pass-transistor based selectors [OSY+95].

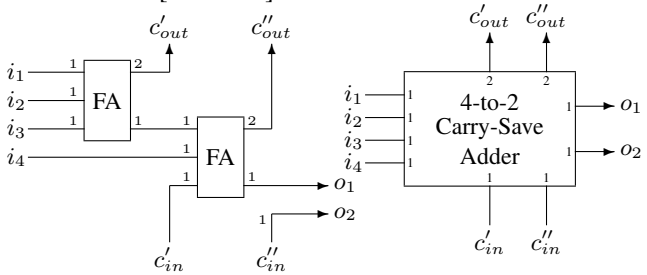


Figure 2. A 4-to-2 adder from full-adders, and its box-representation

Employing the binary borrow-save encoding as a bit-pair (x^n, x^p), where the left-most bit x^n has negative weight, representing the digit value $x = x^p - x^n \in \{-1, 0, 1\}$. Addition of (x^n, x^p) and (y^n, y^p) can be realized by inverting some of the connections as shown in Fig. 3.

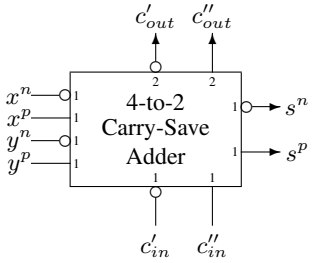


Figure 3. 4-to-2, borrow-save adder

Alternatively, the digit encoding could be 2's complement, where a pair (x^h, x^l) encodes the digit value $d = -2x^h + x^l \in \{-1, 0, 1\}$ (the pair $(1, 0)$ excluded), and similarly for the carry. Based on the defining equation we find

$$\begin{aligned} c^l_{out} &= (x^l + y^l + c^l_{in}) \text{ div } 2 \\ s^l &= (x^l + y^l + c^l_{in}) \bmod 2 \\ c^h_{out} &= (x^h + y^h + c^h_{in}) \text{ div } 2 \\ s^h &= (x^h + y^h + c^h_{in}) \bmod 2 \end{aligned}$$

and thus Figure 4:

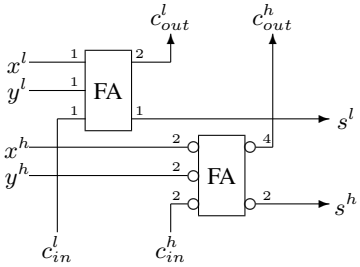


Figure 4. 4-to-2, 2's complement adder

As shown in [Kor05], it may be noted that in a multi-digit adder composed of a linear array of such adders, the inverters are not needed on the internal carry signals. Similarly, when such adders are connected in an array for multi-operand addition, the inverters are not needed anywhere internally. This also shows that an implementation using borrow-save representation is identical to one using carry-save, except for some inverters at the boundary of the array.

V. Analysis of radix $\mathbf{2^r}$, maximally redundant additions

Fig. 1 illustrated the process of adding digits in a symmetric and redundant signed-digit set $D = \{-\alpha, \cdots, 0, \cdots, \alpha\}$ for radix 2^r, $r \geq 2$, where $\alpha = 2^r - 1$ was chosen so that the digit set is maximally redundant. Actually, it is sufficient that $\alpha > 2^{r-1}$ for the incoming carry to be absorbed. The carries in $\{-1, 0, 1\}$ are determined by inspection of the intermediate sum digits, obtained by adding the two operand digits, and the carry is emitted in some suitable encoding, leaving behind a modified digit of reduced range which can absorb the incoming carry.

In a number of publications [MI99], [JPG05], [JP07], [JG10], [GJ11], maximally redundant redundant radix 2^r additions have been investigated, employing various digit and carry encodings.

Since the most recent paper [GJ11] is based on the preceding ones and improves on them, we shall here concentrate on its designs. And as in that paper we shall illustrate them exemplified for radix 16, i.e., for $r = 4$. The digits are encoded in 2's complement, using respectively the maximally redundant digit set (denoted MRSD), employing two different encodings for the carry, the borrow-save (c^n, c^p) and the 2's complement (c^h, c^l). The authors have evaluated the performance (in CMOS based on VHDL descriptions) of 7 designs in all combinations of digit and carry encodings.

We shall here look at the MRSD designs, choosing only the two fastest designs, respectively the D (using borrow-save encoding of the carry) and the F design (using 2's complement), as illustrated in Fig. 5.

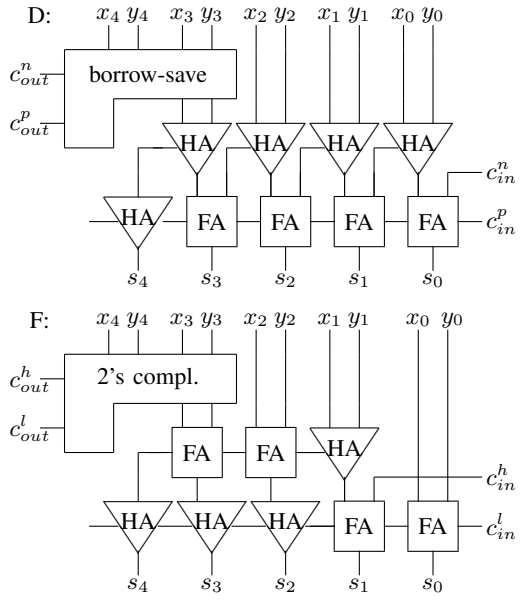


Figure 5. The radix 16 digit-adder designs D and F of [GJ11]

The HA-triangles are half-adders and the FA-boxes are full-adders, possibly with inverters on negative signals, but not shown. The L-shaped boxes contain the logic for determining the carry bits to emit, and two modified leading operand bits of negative weight.

Observe that the internal adders are connected in a ripple-carry structure, due to the digit representation being the non-redundant 2's complement, hence the timing of the digit adder is $O(r)$, i.e., linear in r. But the digit addition may also be performed using a carry look-ahead structure, thus in time $O(\log r)$, at some increase in area. To obtain a faster addition and carry absorption, it is necessary to use a redundant digit representation.

One such possibility for adding radix 16 digits, is combining four 4-to-2 borrow-save adders over the digit-set $\{-1, 0, 1\}$, which together provides an equivalent radix 16 digit adder over the same maximally redundant redundant digit set as above. Since each digit now is encoded in 8 bits, compared to 5, a radix 16 digit adder now has 16 input and 8 output lines, compared to respectively 10 and 5, thus the interconnect structure of the adders is more complex and requires more area. But note that only half as many such adders may be needed in a multi-operand addition array, since the sum of two standard non-redundant binary operands is simply obtained by pairing the bits of the operands, directly forming their sum in carry-save encoding (or by inverting one operand in borrow-save), to be used as further input.

Expanding the 4-to-2 adders in terms of full adders with their interconnections, as an equivalent to the above digit adders in Fig. 5, using the borrow-save encoding we find Fig. 6, again not showing inversions on negative signals. For general r the delay of the circuit in Fig. 6 is independent of the radix 2^r, and identical to that of a single 4-to-2 adder.

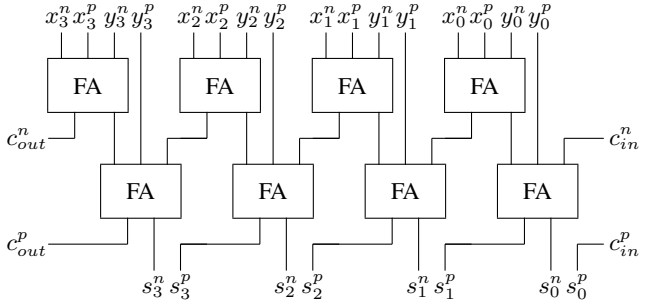


Figure 6. A radix 16 digit-adder design in 4-to-2 borrow-save encoding

Obviously, when generalized as a function of radix 2^r the delays of the circuits in Fig. 5 will be linear in r, whereas the delays in Fig. 6 are constant. Specialized for $r = 1$, the circuit in Fig. 5F must be functionally equivalent to the circuit in Fig. 6 for $r = 1$, such that the latter circuit corresponds to Fig. 4 with 2's complement encoding of operands and carry. Thus optimal implementations of the two circuits must have identical delays.

For $r = 2$ the circuit in Fig. 5F must be slower than for $r = 1$, as signals have to go through additional logic.

It is impossible that even if employing a Manchester carry chain or some carry-lookahead structure, that the logic can be faster for $r = 2$ than for for $r = 1$, and thus faster than the circuit in Fig. 6.

For $r \geq 2$, obviously the circuit of Fig. 6 will be faster than both of those shown in Fig. 5, due to the non-redundant representation of the digits in these.

When using the Fig. 6 design there is no need to group r bits of the operands, standard binary addends can immediately be used, whereas in the designs of Fig. 5 it is necessary to split the operands and convert the digits into radix 2^r, 2's complement.

When performing multiple additions using the borrow-save digit encoding it is necessary to use two leading guard digits beyond those needed to represent the final non-redundant sum [KM06]. The most significant guard digit can be eliminated in a constant time operation, if non-zero by just inverting the digit in the next position. But eliminating the least significant guard digit is at best a log-time operation, equivalent to converting the sum into a non-redundant representation. However, when a high radix like $2^r, r \geq 2$ or 10 is employed, a single guard digit is sufficient.

VI. Conclusions

It has been shown that the proposed recoding of addends into a higher radix of the form 2^r for $r \geq 2$, as in [Par93], [MI99], [JPG05], [JP07], [JG10], [GJ11], with digits encoded in non-redundant 2's complement representation, can not provide faster addition than using standard radix 2, carry-save or borrow-save adders applied directly to non-redundant binary addends. On the contrary, for multi-operand addition the delay is significantly larger when using such recoding into a higher radix, despite the claim "Ultrahigh-Speed" in the title of [JP07]. Furthermore, when employing the carry- or borrow-save encoding half of the additions come for free, since just pairing two non-redundant binary numbers forms their sum in carry-save or (with one of them inverted) in borrow-save. It is noted that (except for some inverters at the boundary) the logic of the adder array is identical, whether the carry-save or the borrow-save representation is used.

The situation is different for decimal addition. Encoding decimal digits in the set $\{-7, \cdots, 0, \cdots, 7\}$ in the compact 2's complement representation has been used efficiently in [GJ09], Finally, note that this analysis does not apply to FPGA implementations, in particular due to the availability in this technology of small fast carry-ripple adders.

VII. Acknowledgments

The author wants to thank Ghassem Jaberipur for help in understanding some details concerning the digit addition and carry-generation in the designs of [GJ11].

References

[Avi61] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, EC-10:389–400, September 1961.

[EL97] M.D. Ercegovac and T. Lang. Effective Coding for Fast Redundant Adders using the Radix-2 Digit Set $\{0, 1, 2, 3\}$. In *Proc. 31st Asilomar Conf. Signals Systems and Computers*, pages 1163–1167, 1997.

[GJ09] S. Gorgin and G. Jaberipur. Fully Redundant Decimal Arithmetic. In *Proc. 19th IEEE Symposium on Computer Arithmetic*, pages 145–152. IEEE, June 2009.

[GJ11] S. Gorgin and G. Jaberipur. A Family of High Radix Signed Digit Adders. In *Proc. 20th IEEE Symposium on Computer Arithmetic*, pages 112–121. IEEE, July 2011.

[HNN+87] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi. A High-Speed Multiplier Using a Redundant Binary Adder Tree. *IEEE Journal of Solid State Circuits*, pages 28–34, 1987.

[JG10] G. Jaberipur and S. Gorgin. An Improved Maximally Redundant Signed Digit Adder. *Computers and Electrical Engineering*, 36(3):491–502, May 2010.

[JP07] G. Jaberipur and B. Parhami. Stored-Transfer Representations with Weighted Digit-Set Encodings for Ultrahigh-Speed Arithmetic. *IET Circuits Devices Systems*, 1(1):102–110, February 2007.

[JPG05] G. Jaberipur, B. Parhami, and M. Ghodsi. Weighted Two-Valued Digit-Set Encodings: Unifying Efficient Hardware Representation Schemes for Redundant Number Systems. *IEEE Transactions on Circuits and Systems, Regular Papers*, 52(7):1348–1357, July 2005.

[KM06] P. Kornerup and J-M. Muller. Leading Guard Digits in Finite Precision Redundant Representations. *IEEE Transactions on Computers*, 55(5):541–548, May 2006.

[KM10] P. Kornerup and D.W. Matula. *Finite Precision Number Systems and Arithmetic*, volume 133 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Sept. 2010.

[Kor05] P. Kornerup. Reviewing 4-to-2 Adders for Multi-Operand Addition. *Journal of VLSI Signal Processing*, 40(1):143–152, May 2005. Previously presented at ASAP2002.

[KS05] R.D. Kenney and M.J. Schulte. High-Speed Multioperand Decimal Adders. *IEEE Transactions on Computers*, 54(8):953–963, August 2005.

[MI99] M.C. Mekhallalati and M.K. Ibrahim. New high radix maximally-redundant signed digit adder. In *Proc. of the 1999 IEEE International Symposium on Circuits and Systems (ISCAS'99)*, volume 1, pages 459–462, July 1999.

[OSY+95] N. Ohkubo, M. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakagome. A 4.4 ns CMOS 54 × 54-b Multiplier Using Pass Transistor Multiplexer. *IEEE Journal of Solid State Circuits*, 30(3):251–257, 1995.

[Par90] B. Parhami. Generalized signed-digit number systems: A unifying framework for redundant number representations. *IEEE Transactions on Computers*, C-39(1):89–98, January 1990.

[Par93] B. Parhami. On the Implementation of Arithmetic Support Functions for Generalized Signed Digit Number Systems. *IEEE Transactions on Computers*, C-42(3):379–384, March 1993.

[PGK01] D.S. Phatak, T. Goff, and I. Koren. Constant-Time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations. *IEEE Transactions on Computers*, 50(11):1267–1278, 2001.

[TYY85] N. Takagi, H. Yasuura, and S. Yajima. High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree. *IEEE Transactions on Computers*, C-34(9):789–796, September 1985.

[Wei81] A. Weinberger. 4-2 Carry-Save Adder Module. *IBM Technical Disclosure Bulletin*, 23, January 1981.