Posibits, Negabits, and Their Mixed Use in Efficient Realization of Arithmetic Algorithms

G. Jaberipur[1] and B. Parhami[2],
[1]Dept. of Electrical and Computer Engr., Shahid Beheshti University, and
School of Comp. Science, Inst. for Research in Fundamental Science (IPM)
[2]Dept. of Electrical and Computer Engr., University of California
Santa Barbara, CA 93106-9560, USA
E-mail: jaberipur@sbu.ac.ir, parhami@ece.ucsb.edu

Abstract— **Positively weighted and negatively weighted bits (posibits, negabits) have been used in the interpretation of 2's-complement, negative-radix, and binary signed-digit number representation schemes as a way of facilitating the development of efficient arithmetic algorithms for various application domains. In this paper, we show that a more general view of posibits and negabits, along with their mixed use in any combination (using inverse encoding for negabits), unifies a number of diverse implementation schemes, while at the same time making the resultant designs more efficient by avoiding custom or modified hardware elements and restricting the implementation to the use of standard arithmetic cells. Such standard cells have been highly optimized and are continually improving due to their wide applicability. Other practical benefits of our formulation include facilitation of low-voltage and low-power design, again due to the widespread availability of standard cells in variants optimized for low-voltage operation or energy economy. Pedagogical benefits include more intuitive explanations for a number of widely used transformations, such as Booth's recoding and column compression.**

I. Introduction

One way to improve the speed and efficiency of arithmetic-intensive applications is through nonstandard number formats. Residue number system (RNS) representation constitutes an option that is particularly suitable for signal processing applications [1], [2], [3]. Redundant number representation offers a second, somewhat more versatile, option. For much of the 50-year history of redundant representations, beginning with the seminal work of Metze and Robertson, who proposed stored-carry numbers [4], and Avizienis, who extended it to signed-digit representations [5], digit sets have been encoded using conventional sign-and-magnitude or 2's-complement format. Other encodings, where used, have been haphazard, with little or no exploration of alternatives. Clearly, encodings used influence hardware designs, leading to a variety of custom implementations which have to be optimized for speed and power consumption from scratch, thus wasting a great deal of time and effort.

A key strength of redundant representations is in their carry-free addition property, making addition even faster than in RNS. While redundant representations lead to slower multiplication compared with RNS, they can be quite competitive overall, given the elimination of the final carry-propagate addition required in standard weighted representations. Furthermore, elimination of forward (binary to RNS) and reverse (RNS to binary) conversions, and the possibility of multiplierless implementations in some cases [6] can mitigate the speed loss. One drawback of redundant representations, as usually applied, is their need for nonstandard hardware building blocks that must be designed from scratch (e.g., [7]). In this paper, we show how speed can be improved via uniform treatment of positive and negative bit values (posibits, negabits), allowing signed-digit arithmetic to be performed using the same highly efficient and extensively optimized circuitry used for unsigned digits.

There are already quite a few arithmetic algorithms and implementations in which mixed posibits and negabits are used to gain pedagogical and practical benefits. For example, viewing the most-significant bit of a 2's-complement number as having a negative weight is a well-known method for simplifying direct multiplication of signed numbers [8]. Similarly, negatively weighted bit positions have been used to simplify the interpretation of, and algorithm design for, number systems with negative and imaginary radices [9]. Negabits have also been used in the $\langle n, p \rangle$ encoding of binary signed digits [10]. Beginning in the early 2000s, we noticed that many efficient and popular encodings for redundant representations share the property that they utilize posibits and negabits with power-of-2 weights [11]. Over the intervening years, we have applied such encodings, in their basic and generalized forms, to many design problems, thus garnering uniformity and efficiency in a number of applications [12], [13], [14]. A main goal of this paper is to share these ideas with designers in a way that makes their appreciation and application more likely.

In describing arithmetic algorithms and associated transformations, it is customary to denote an ordinary bit, or posibit, by a heavy dot (●), thus producing a visual representation of numbers and algorithm steps in "dot notation" (Fig. 1a). Using a small hollow circle (○) to denote a negabit allows us to visualize 2's-complement numbers, negabinary or radix-$\tilde{2}$ numbers, and other representations formed by a specific mix of posibits and negabits in an extended dot notation (Figs. 1b and 1c).

Redundant representations, with multiple dots in some positions, allow us to take advantage of their carry-free arithmetic property. Additionally, representational redundancy can lead to faithful representation of arbitrary digit sets such as [0, 9] (Fig. 1d) and [–6, 6] (Fig. 1e), which would otherwise have to be encoded using the wider ranges of [0, 15] and [–8, 7] (e.g., as in Figs. 1a and 1b), respectively.

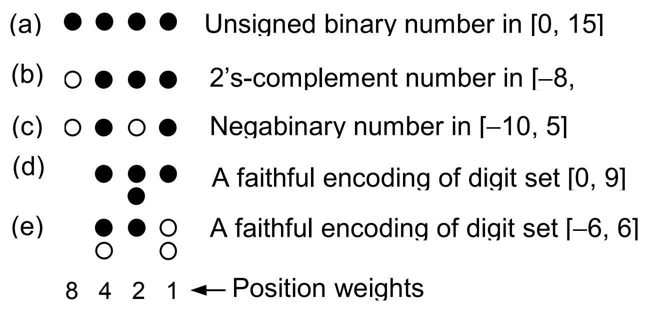

(a) ● ● ● ● Unsigned binary number in [0, 15]
(b) ○ ● ● ● 2's-complement number in [–8,
(c) ○ ● ○ ● Negabinary number in [–10, 5]
(d) A faithful encoding of digit set [0, 9]
(e) A faithful encoding of digit set [–6, 6]
8 4 2 1 Position weights

Fig. 1. Number representations of varying ranges in extended dot notation.

It is the mixed use of arbitrary combinations of posibits and negabits in various bit positions (e.g., Fig. 1e) that forms the focus of this paper. In general, there may be several weighted bit-set (WBS) encodings for a given digit set. For example, a collection of 6 negabits and 6 posibits, all weighted 1, also faithfully represent [–6, 6]. However, 2-deep or canonical WBS encodings (i.e., those containing at most two bits in each binary position, as in Fig. 1) are preferred due to the possibility of more efficient implementation of arithmetic operations [12]. Any noncanonical WBS encoding can be converted to an equivalent canonical one using the range-preserving transformations of Fig. 2 to redistribute the extra dots in columns with more than 2 dots.

Some results on WBS encodings, and associated arithmetic algorithms, follow immediately from the preceding discussion. For example, it is easy to see that any arbitrary digit set [$-\alpha$, β] can be faithfully represented using canonical (2-deep) WBS encoding; simply encode the digit set with α negabits and β posibits of weight 1 and then apply the transformations of Fig. 2 in multiple rounds to reduce the depth to 2. Figure 3 depicts such transformations for $\alpha = \beta = 6$. Also, adding two WBS-encoded numbers can be viewed as the operation of depth reduction from 4 to 2, where the depth of 4 results from aligning the corresponding positions of the 2-deep operands. Finally, subtraction can be converted to addition by changing posibits to negabits, and vice versa, in the subtrahend.

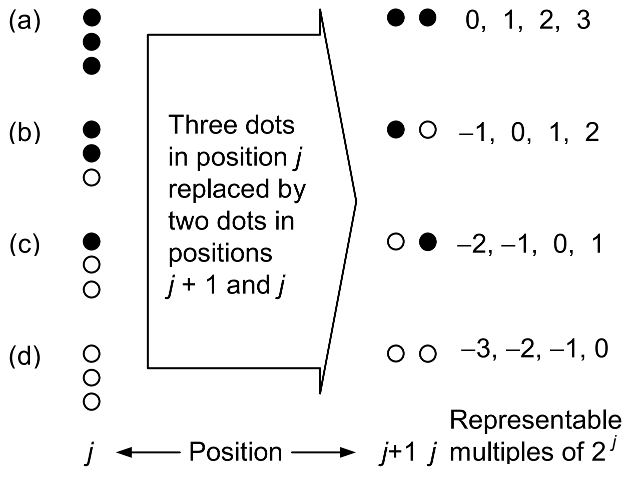

(a)
(b)
(c)
(d)
Three dots in position j replaced by two dots in positions j + 1 and j
0, 1, 2, 3
–1, 0, 1, 2
–2, –1, 0, 1
–3, –2, –1, 0
j ← Position → j+1 j
Representable multiples of 2^j

Fig. 2. Range-preserving transformations for WBS encodings.

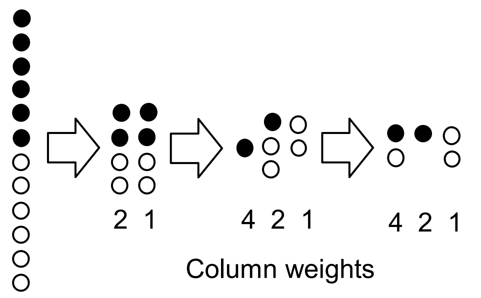


Fig. 3. Faithful WBS representations of digit set [–6, 6].

II. Representations and Algorithms

One of the important notions in the design of arithmetic circuits for signal processing and other applications is that of bit compression. For example, a half-adder (HA) can be viewed as a dot redistribution tool that takes two dots in the same column and produces one dot each in the same and the next higher position, as depicted in Fig. 4a. Similarly, a full-adder (FA), also known as (3; 2)-counter, produces the sum and carry output bits that represent the count of 1s among its three equally-weighted input bits. When applied to multiple columns of 3 dots at once, this leads to reduction of 3 binary numbers to two binary numbers in a scheme known as carry-save addition. This operation can also be viewed as compressing 3-bit columns to 2-bit ones (Fig. 4b); hence the alternate designation of full adders as (3; 2)-compressors. Finally, the (4; 2)-compressor of Fig. 4c is capable of compressing a column of 4 dots into 2 dots in adjacent columns, plus a carry bit that is sent to the next higher column. This is possible because 4 dots in a column when combined with an incoming carry (the fifth dot) can be represented by one dot of the same weight and 2 dots of double that weight (Fig. 4d).

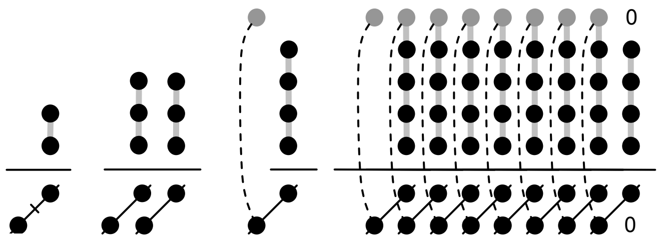


Fig. 4. Basic bit compression and redistribution operations in dot notation.

Example 1 [Multiplication of unsigned and 2's-complement binary integers]: Consider the problem of multiplying two k-bit unsigned binary numbers (Fig. 5a). This is done by forming the k^2 bitwise products x_iy_j (logical AND terms) and then using bit reduction and a final addition (the dashed box near the bottom of Fig. 5a) to form the unsigned product. Given that conventional bit compression methods work only on posibits, 2's-complement numbers are usually multiplied by first converting the bit-matrix composed of posibits and negabits into an equivalent matrix containing only posibits and then compressing the resulting bit matrix as before (Fig. 5b). We will see in Section 3 that inverted encoding of negabits allows us to manipulate them directly, thus saving some circuit resources and latency. $\square$

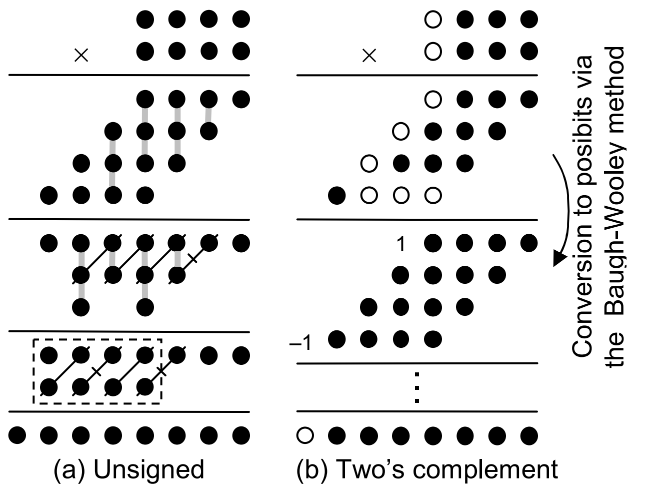


Fig. 5. Integer multiplication viewed as bit compression and addition.

The essence of the Baugh-Wooley [8] conversion step of Fig. 5b is the replacement of any negabit $-b$ by the posibit $1 - b$ (logical complement of b) and the constant negabit -1. Constant negabits are then gradually shifted to the left, and eventually discarded at the left end [15], using the identity $(0\ -1)_{\text{two}} = (-1\ 1)_{\text{two}}$. Thus, we see that all operations, including those on negabits, are converted to appropriate operations on posibits to allow the use of the standard, and highly developed, building blocks of Fig. 4.

Example 2 [Booth recoding for radix-4 multiplication]: Figure 6 demonstrates the pedagogical power of mixing posibits and negabits in a representation. This easily understood representation of modified Booth's recording (converting a 2's-complement number to minimally-redundant radix 4), is justified by the substitution of the i^{th} digit $(y_i)_{\text{two}}$ of the multiplier with the equivalent 2-bit number $(y_i\ y_i^-)_{\text{two}}$, for odd i, where y_i^- denotes a negabit with arithmetic worth of $\|y_i^-\| = -y_i$. The required Booth signals S_i, T_i, and O_i for "sign," "two" (i.e., doubled multiplicand), and "one" (i.e., the multiplicand) can be expressed in terms of the constituent bits of a minimally redundant radix-4 digit. The relevant expressions are found in the Eqn. set 1, where a "–" superscript identifies a negabit and each radix-4 digit in [–2, 2] is represented by a weight-2 negabit y_{2i+1}^- and two weight-1 posibits y_{2i} and y_{2i-1}. $\square$

$$S_i = y_{2i+1}^-, \quad O_i = y_{2i} \oplus y_{2i-1},$$
$$T_i = \overline{y_{2i+1}^-} y_{2i}\, y_{2i-1} \vee y_{2i+1}^- \overline{y_{2i}}\, \overline{y_{2i-1}} \qquad (1)$$

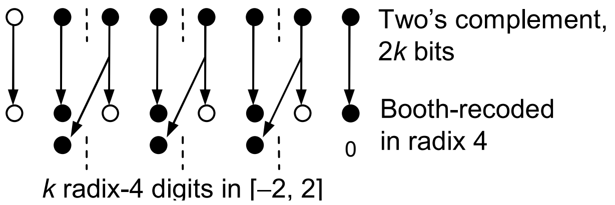


Fig. 6. Justification of modified Booth's recoding via extended dot notation.

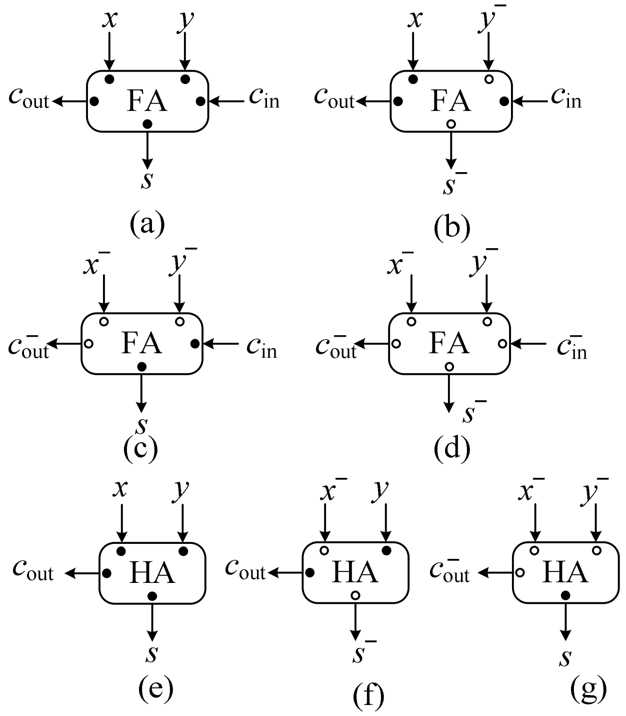
III. Inverted Encoding of Negabits

Conventionally (e.g., in 2's-complement numbers), a negabit is encoded by using logical 0 to denote the arithmetic value 0 and logical 1 to denote the arithmetic value −1. A negabit with inverted encoding (IE-negabit) uses the opposite convention, such that the arithmetic value $\|n\|$ of a negabit n is equal to $n - 1$. Given that each of these encodings can be converted to the other using a NOT gate, discussion of the alternate inverted encoding may appear trivial; thus, some explanation is in order. We will see shortly that IE-negabits can be processed, in conjunction with posibits, using standard half/full-adders and compressors with absolutely no circuit modification. This use of standard cells is very important, for it offers the advantage of being able to choose from a variety of readily available designs that are optimized based on different criteria (e.g., latency, area, and power) for a multitude of implementation technologies [16], [17]. To process conventional negabits in the same way, inverters are typically inserted on some inputs/outputs of standard cells [18], adding some latency, compromising circuit regularity, and introducing the need for area/power re-optimization. Note that even though the latency of an inverter is fairly small, removing one or more inversion layers in a carry-free adder that typically needs only 4-8 logic levels leads to nontrivial improvements.

The key to improvements resulting from IE-negabits is the property that their logical and arithmetic values vary in the same direction. Representing the value −1 (0) as logical 0 (1) is in effect a biased representation with a bias of 1. A posibit is unbiased (has a bias of 0), given that its logical and arithmetic values are identical. Note that as long as the sum of biases for the inputs matches those of the outputs, no adjustment will be needed when posibits and negabits are combined as if they were all posibits. Figure 7 shows schematic representations of a full-adder (half-adder) used to combine a set of 3 (2) bits, which includes from 0 to 3 (2) negabits [12]. Note that when a negabit is sent to the next higher position, its bias is effectively doubled. Thus, the sums of input and output biases are balanced (Eqn. set 2) in all seven cases depicted in Fig. 7. Recall that a standard full-adder (half-adder) operates on posibits in a way that enforces the identity $x + y + c_{\text{in}} = 2c_{\text{out}} + s$ ($x + y = 2c_{\text{out}} + s$).

(a)	$0 + 0 + 0 = 0 + 0$	(e)	$0 + 0 = 0 + 0$
(b)	$0 + 0 + 1 = 0 + 1$	(f)	$0 + 1 = 0 + 1$
(c)	$0 + 1 + 1 = 2 + 0$	(g)	$1 + 1 = 2 + 0$
(d)	$1 + 1 + 1 = 2 + 1$		(2)

For clarity in studying Fig. 7, the reader is reminded that a "–" superscript identifies a negabit.



x y c_{out} FA c_{in} s (a)
x y^- c_{out} FA c_{in} s^- (b)
x^- y^- c_{out}^- FA c_{in} s (c)
x^- y^- c_{out}^- FA c_{in}^- s^- (d)
x y c_{out} HA s (e)
x^- y c_{out} HA s^- (f)
x^- y^- c_{out}^- HA s (g)

Fig. 7. Full-adders and half-adder as universal combiners of posibits and negabits.

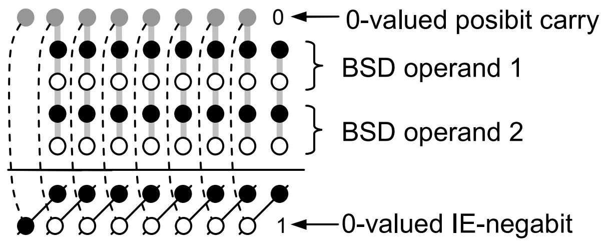


Fig. 8. (4; 2)-compressors as redundant binary adders.

Figure 8 depicts an instance of the universal (4; 2)-compressor of Fig. 4c acting as a redundant binary adder (RBA), or adder with binary signed-digit (BSD) inputs. The best RBA cell that we have encountered is based on a custom design [19] and has a latency of 3 XOR gates, the same as a conventional (4; 2) compressor, with gate counts also being comparable. It is worth noting that the design just mentioned uses a 2-bit encoding that is the same as the (n, p) encoding with IE-negabits. However, because of the ad hoc approach, the design effort is much greater and the resulting circuits cannot benefit from performance improvements on standard cells. Any available (4; 2)-compressor circuit, on the other hand, properly handles any 5-collection of posibits and IE-negabits in its inputs [20]. There are also other highly optimized compressors, exemplified by (5; 2) compressors [20], that may prove beneficial in reducing mixed posibit/negabit matrices where the depth is not a multiple of 4. This is yet another confirmation that the use of highly optimized standard cells is preferable whenever possible.

IV. Implementations and Applications

A highly efficient high radix maximally redundant signed digit adder, based on IE-negabits, has been recently offered [14]. We have previously used IE-negabits and WBS encoding for the implementation of efficient arithmetic circuits with redundant and hybrid-redundant representations [20], [22]. Implementation of BSD addition, by means of off-the-shelf (4; 2)-compressors, was mentioned in Section 3 (Fig. 8). In this section we provide three examples to better show how WBS encoding and IE-negabits facilitate the bit-level design and lead to efficient arithmetic circuitry.

A straightforward implementation of the digit-level addition algorithm for SD number systems [23] implies three h-bit carry-propagating operations in sequence. For $\rho \geq 2$, the sum digit in position i depends on the operand digits in the same position and those of position $i - 1$. For $\rho = 1$, however, it depends also on the operands digits in position $i - 2$. Example 3 provides a particular bit-level implementation, for $\rho = 1$, where there is only one h-bit carry-propagating operation and each bit within the sum digit in position i depends only on the bits of two operand digits; either on the bits of digits in positions i and $i - 1$ or those of positions $i - 1$ and $i - 2$. Example 4 deals with a useful cost-free value-preserving WBS transformation. A unified implementation of signed and unsigned multiplications is presented in Example 5.

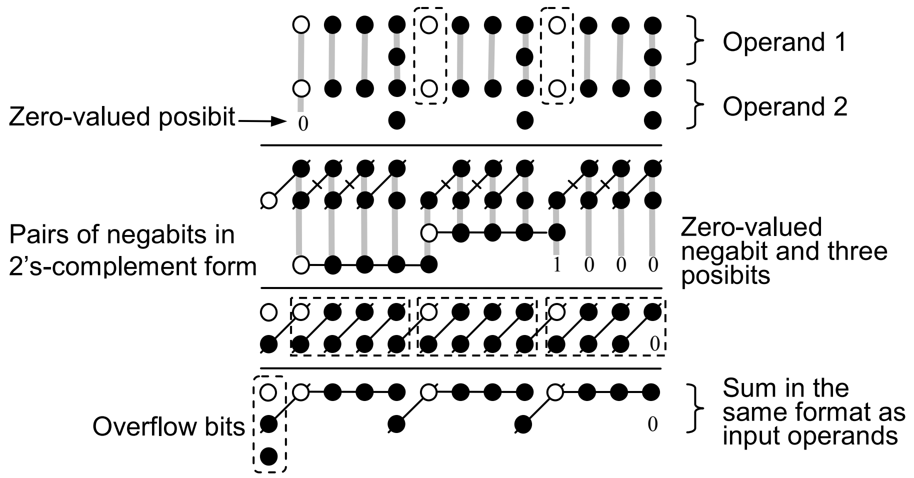

Operand 1
Operand 2
Zero-valued posibit
Pairs of negabits in 2's-complement form
Zero-valued negabit and three posibits
Overflow bits
Sum in the same format as input operands

Fig. 9. Dot-notation representation of symmetric hybrid-redundant addition.

Example 3 [Simplified minimally redundant radix-2^h SD addition]: Figure 9 depicts a conceptual representation of stored-posibit addition as a case of symmetric extended hybrid-redundant number system, where "extended" refers to our allowing negabits as well as posibits in nonredundant positions [22]. Recall that ordinary hybrid redundancy uses only posibits in such positions [7]. The particular number system shown is periodic, with a period of 4 positions, and thus corresponds to a radix-16 generalized signed-digit representation with the minimally redundant digit set [–8, 8]. The first stage of the addition process, depicted in Fig. 9, converts pairs of negabits in the input operands, with the exception of those in the leftmost position, to 5-bit two's-complement numbers (see the dashed boxes). The rest of the process consists of standard bit compression and a final set of 4-bit additions. Note that the stored posibit of the sum digit in position i does not depend on the operand digits in the same position and the bits of the two's complement main part do not depend on the operand digits in position $i – 2$. □

Example 4 [Value-preserving polarity inversion in faithfully represented balanced signed digits]: Consider the rightmost representation of the digit set [–6, 6] in Fig. 3. Such a faithfully represented signed digit is invertible by exchanging posibits and negabits, as shown in Fig. 10a, where it is easy to see that identical bit assignments to both representations yields equal arithmetic values. This provides the opportunity of regarding posibits (negabits) as if they were negabits (posibits), where such an interpretation would facilitate the design process. For example, Fig. 10b represents the essence of the transfer extraction scheme of a radix-16 maximally redundant signed-digit (MRSD) adder, with each redundant digit in [–15, 15] encoded as a 5-bit 2's-complement number. The carry-free addition process requires the extraction of a weight-16 transfer digit $t_{i+1} \in [–1, 1]$ from the operand digits in position i, whose sum ranges from –30 to 30, leaving a residual in [–14, 14]. Inverting the polarity of all bits in the top operand and the least-significant bit in the lower operand preserves the arithmetic value of the transformed bits, given that the transformation in position j increases (decreases) the arithmetic value by 2^j. This cost-free transformation (inversion occurs in the way the bits are viewed, rather than via an inversion circuit) provides a weight-16 negabit/posibit pair in the most-significant position that can serve as the desired t_{i+1}, except in a few input cases that are detectable via simple exception handling logic [14]. □

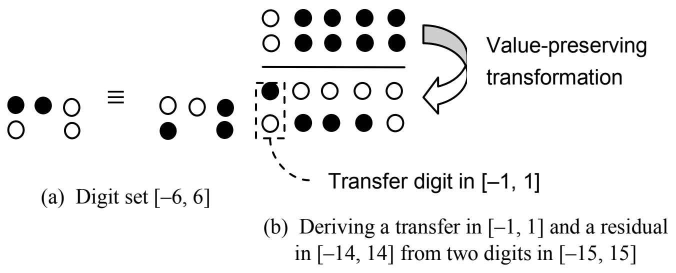

≡
Value-preserving transformation
Transfer digit in [–1, 1]
(a) Digit set [–6, 6]
(b) Deriving a transfer in [–1, 1] and a residual in [–14, 14] from two digits in [–15, 15]

Fig. 10. Cost-free value-preserving transformations.

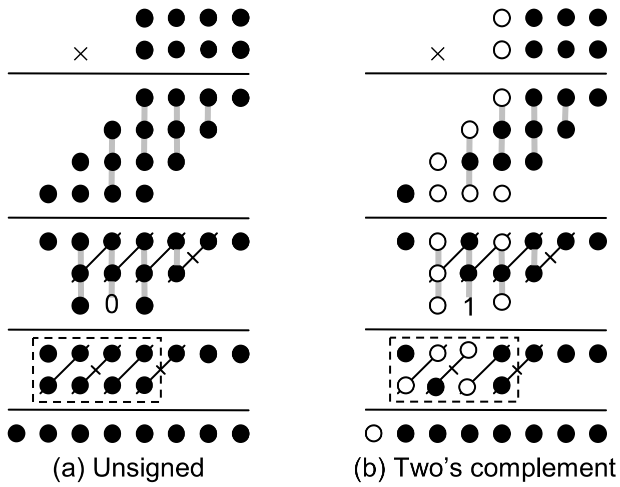


Fig. 11. Unsigned or signed multiplication with identical tree reduction circuits.

Example 5 [Two's complement multiplication]: Fig. 11 represents an implementation of 2's-complement multiplication using a tree reduction part that is identical to that of unsigned multiplication, assuming the use of IE-negabits for the partial products. Therefore, we use energy efficient NAND gates [24] wherever the inputs of a partial product generation cell are of opposite polarities. Among other advantages, this design offers the benefit of circuit sharing (i.e., the reduction tree and the final adder) between unsigned and signed multiplication, both of which are usually needed. Note that the constant posibit 0 (negabit 1) in Fig. 11a (11b) can be replaced by a sign input to decide on unsigned (0) or 2's-complement (1) multiplication. □

We conclude this discussion by demonstrating that IE-negabits can be used as part of nonredundant representations with equal ease. This is important because it indicates that conversion to conventional encoding is not necessary, even when we complete the redundant part of a computation and return to nonredundant format. Figure 12 shows the required full adder in the most significant position of a 2's-complement adder with conventional encoding (a) and IE-negabits (b), where *ov* is the overflow signal. The latter case is justified by Table I, where *s* is the posibit sum output of the FA and *S* is the true negabit sum in the most significant position (MSP) of the 2's complement result. Finally, negation as in conventional 2's-complement representation is done by inverting all the bits and adding 1, as shown below for the *k*-bit number $X = X_{k-1}x_{k-2}\ldots x_0$, where $\|e\|$ denotes the arithmetic value of logical expression *e*.

$$\begin{aligned}
\|\overline{X}+1\| &= \|\overline{X_{k-1}}\,\overline{x_{k-2}}\ldots\overline{x_0}+1\| \\
&= 2^k\left(\overline{X_{k-1}}-1\right)+2^{k-1}\overline{x_{k-2}}+\cdots+\overline{x_0}+1 \\
&= 2^k(1-X_{k-1}-1)+2^{k-1}(1-x_{k-2})+\cdots(1-x_0)+1 \\
&= -2^k(X_{k-1}-1)-2^{k-1}x_{k-2}+\cdots-x_0 \\
&= -2^k\|X_{k-1}\|-2^{k-1}\|x_{k-2}\|+\cdots-\|x_0\| \\
&= -\|X\|
\end{aligned}$$

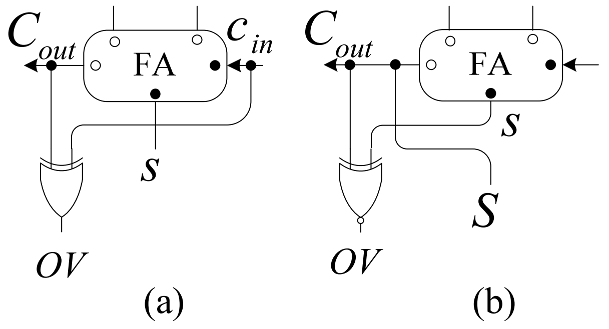


Fig. 12. Leftmost full adder in 2's complement adder: conventional (a), IE-negabits (b)

Table I. Justification for Fig. 12b

C_{out}	*s*	MSP sum	*ov*	*S*
0	0	−2	1	X
0	1	−1	0	0
1	0	0	0	1
1	1	1	1	X

V. Conclusion

We have shown, through a number of examples, that posibits and negabits can be advantageously intermixed. This approach offers both pedagogical and practical benefits. On the pedagogical front, viewing a number of different transformations, such as Booth recoding and column compression, in a unified way engenders a better understanding of why these methods work and how variants of such methods can be devised. Practical benefits include both a reduction in design effort and improvements in design parameters such as cost, speed, and/or power consumption. These practical benefits are direct results of our unified design strategy, based on the exclusive use of highly optimized standard building blocks or cells, for realizing arithmetic operations on representations composed of weighted posibits and negabits.

Even though we have used this method, and the associated inverted encoding of negabits, in our designs before, we thought that explicating the underpinnings of our design strategy, outlining its intuitive basis, and listing some of the key applications would be beneficial to designers of signal processing and other VLSI systems.

Our discussion has been qualitative, pointing to advantages in terms of easier exploration of design space, simpler conceptual design (thus, design time reduction and error avoidance), and more regular VLSI layout. A quantitative assessment of the benefits is only possible for specific applications, after full circuit-level implementation. We have done this in our previous publications cited in the references. It is worth noting that the use of standard arithmetic building blocks allows our designs to benefit from the continuous innovations that lead to faster, more compact, and lower power components such as half-adders, full adders, and bit compressors. For example, we note that full adders have improved over the years in terms of both the number of transistors and power requirements. Matching these improvements with ad-hoc custom designs, built from scratch, would be quite difficult and labor-intensive, if not impossible

Acknowledgment

Jaberipur's research is funded in part by Shahid Beheshti University and in part by IPM under Grant CS1388-2-03.

References

[1] Taylor, F. J., Digital Filter Design Handbook, *M. Dekker*, 1983.

[2] Soderstrand, M. A., W. K. Jenkins, G. A. Jullien, and F. J. Taylor (Editors), Residue Number System Arithmetic, *IEEE Press*, 1986.

[3] Omondi, A. and B. Premkumar, Residue Number Systems: Theory and Implementation, *Imperial College Press*, 200

[4] Metze, G. and J. E. Robertson, "Elimination of Carry Propagation in Digital Computers," *Information Processing '59 (Proc. of UNESCO Conf., June 1959)*, 1960, pp. 389-396.

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

[6] Meyer-Baese, U., Digital Signal Processing with Field Programmable Gate Arrays, *Springer, 3rd ed.*, 2007.

[7] Phatak, D. S. and I. Koren, "Constant-Time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations," *IEEE Trans. Computers*, Vol. 50, No. 11, pp. 1267-1278, November 2001.

[8] Baugh, C. R. and Wooley, B. A., "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Trans. Computers*, Vol. 22, No. 6, pp. 1045-1047, December 1973.

[9] Koren, I., and Y. Maliniak, "On Classes of Positive, Negative, and Imaginary Radix Number Systems," *IEEE Trans. Computers*, Vol. 30, No. 5, pp. 312-317, May 1981.

[10] Parhami, B., "Carry-Free Addition of Recoded Binary Signed-Digit Numbers," *IEEE Trans. Computers*, Vol. 37, No. 11, pp. 1470-1476, November 1988.

[11] Jaberipur, G., B. Parhami, and M. Ghodsi, "Weighted Bit-Set Encodings for Redundant Digit Sets: Theory and Applications," *Proc. 36th Asilomar Conf. Signals, Systems, and Computers*, November 2002, pp. 1629-1633.

[12] Jaberipur, G., B. Parhami, and M. Ghodsi, "Weighted Two-Valued Digit-Set Encodings: Unifying Efficient Hardware Representation Schemes for Redundant Number Systems," *IEEE Trans. Circuits and Systems I*, Vol. 52, No. 7, pp. 1348-1357, July 2005.

[13] Jaberipur, G. and B. Parhami, "Unified Approach to the Design of Modulo-(2n ± 1) Adders Based on Signed-LSB Representation of Residues, *Proc. 19th IEEE Int'l Symp. Computer Arithmetic*, 2009, pp. 57-64.

[14] Jaberipur, G. and S. Gorgin, "An Improved Maximally Redundant Signed Digit Adder," *Computers & Electrical Engineering, in press*, doi:10.1016/j.compeleceng.2009.12.002

[15] Parhami, B., Computer Arithmetic: Algorithms and Hardware Designs, *Oxford University Press*, 2nd ed., 2010.

[16] Chang, C.-H., J. Gu, and M. Zhang, "A Review of 0.18-μm Full Adder Performances for Tree Structured Arithmetic Circuits," *IEEE Trans. VLSI Systems*, Vol. 13, No. 6, pp. 686-695, June 2005.

[17] Aguirre-Hernandez, M. and M. Linares-Aranda, "CMOS Full-Adders for Energy-Efficient Arithmetic Applications," *IEEE Trans. VLSI Systems*, doi, 10.1109/TVLSI.2009.2038166.

[18] Kornerup, P., "Reviewing 4-to-2 Adders for Multi-Operand Addition," *J. VLSI Signal Processing*, Vol. 40, No. 1, pp. 143-152, May 2005. 7.

[19] Kim, Y. et al., "A Carry-Free 54b × 54b Multiplier Using Equivalent Bit Conversion Algorithm," *IEEE J. Solid-State Circuits*, Vol. 36, No. 10, pp. 312-317, October 2001.

[20] Jaberipur, G., B. Parhami, and M. Ghodsi, "An Efficient Universal Addition Scheme for All Hybrid-Redundant Representations with Weighted Bit-Set Encoding," *J. VLSI Signal Processing*, Vol. 42, No. 2, pp. 149-158, February 2006.

[21] Chang, C.-H., J. Gu, and M. Zhang, "Ultra Low-Voltage Low-Power CMOS 4-2 and 5-2 Compressors for Fast Arithmetic Circuits," *IEEE Trans. Circuits and Systems I*, Vol. 51, No. 10, October 2004.

[22] Jaberipur, G., B. Parhami, and M. Ghodsi, "Constant-Time Addition with Hybrid-Redundant Numbers: Theory and Implementations," *Integration, the VLSI J.*, Vol. 41, No. 1, pp. 49-64, January 2008.

[23] Parhami, B., "Generalized Signed-Digit Number Systems: A Unifying Framework for Redundant Number Representations," *IEEE Trans. Computers*, Vol. 39, No. 1, pp. 89-98, January 1990.

[24] Abid, Z., H. El-Razouk, and D. A. El-Dib, "Low Power Multipliers Based on New Hybrid Full Adders," *Microelectronics J.*, Vol. 39, pp. 1509-1515, 2008.