Published in IET Computers & Digital Techniques
Received on 21st April 2011
Revised on 13th January 2012
doi: 10.1049/iet-cdt.2011.0059

Special Issue: High-Performance Computing System Architectures: Design and Performance

ISSN 1751-8601

Efficient realisation of arithmetic algorithms with weighted collection of posibits and negabits

G. Jaberipur[1,2] *B. Parhami*[3]

[1]*Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran1983963113, Iran*
[2]*School of Computer Science, Institute for Research in Fundamental Science (IPM), Tehran, Iran*
[3]*Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA93106-9560, USA*
E-mail: jaberipur@sbu.ac.ir

Abstract: Most common uses of negatively weighted bits (negabits), normally assuming arithmetic value −1(0) for logical 1(0) state, are as the most significant bit of 2's-complement numbers and negative component in binary signed-digit (BSD) representation. More recently, weighted bit-set (WBS) encoding of generalised digit sets and practice of inverted encoding of negabits (IEN) have allowed for easy handling of any equally weighted mix of negabits and ordinary bits (posibits) via standard arithmetic cells (e.g., half/full adders, compressors, and counters), which are highly optimised for a host of simple and composite figures of merit involving delay, power, and area, and are continually improving due to their wide applicability. In this paper, we aim to promote WBS and IEN as new design concepts for designers of computer arithmetic circuits. We provide a few relevant examples from previously designed logical circuits and redesigns of established circuits such as 2's-complement multipliers and modified booth recoders. Furthermore, we present a modulo-$(2^n + 1)$ multiplier, where partial products are represented in WBS with IEN. We show that by using standard reduction cells, partial products can be reduced to two. The result is then converted, in constant time, to BSD representation and, via simple addition, to final sum.

1 Introduction

In the past couple of decades, the computing discipline and its associated products market have been increasingly dominated by media processing and a host of supporting communications and digital signal processing (DSP) applications. The steady rise in the performance of general purpose and DSP chips has driven not only the market for embedded computing devices, but also influenced the broader scene, from personal computing platforms to supercomputers. Greater technological capabilities and the ever-increasing demand for more functionality in computing and communication devices have driven a tightly wound cycle of advances that has increased not only the processing speed, but has done so more economically and at lower energy dissipation. The demand for even greater performance continues to drive the development of hardware for signal processing, to the extent that DSP and graphics chips are now at the leading edge of performance, cost-effectiveness and energy efficiency.

One way to improve the speed and efficiency of DSP and other arithmetic-intensive applications is through non-standard number formats. Residue number system (RNS) representations have led the way in this regard since the 1980s [1–3]. Redundant number representations [4, 5] have also been used widely, particularly in reducing the complexity of digital filter implementation via multiplierless designs [6]. Of course, the main strength of redundant representations is in their carry-free addition property, making addition even faster than in RNS. Although 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 implementation in some cases can mitigate the speed loss.

One drawback of redundant and other unconventional number representations, as currently used in various applications, is their need for non-standard hardware building blocks that must be designed from scratch. In this paper, we show how speed can be improved via the introduction and uniform treatment of positively and negatively weighted bits, allowing virtually any arithmetic algorithm on signed operands (including signed-digit, and other redundant or hybrid-redundant arithmetic operations) to be performed using the same highly efficient and extensively optimised circuitry used for unsigned values.

Considering the most-significant bit of a 2's-complement number as having a negative weight is a well-known method for simplifying direct signed multiplication [7]. Similarly, negatively weighted bit positions have been used to simplify the interpretation of, and algorithm development for, number systems with negative and imaginary radices [8]. Negatively weighted bits (negabits) have also been used in the $\langle n, p \rangle$ encoding of binary signed digits [9] and, more recently, in weighted bit-set (WBS) encoding of redundant number systems [10], signed-LSB representation of

modulo-$(2^n \pm 1)$ residues [11] and signed-digit adders [12, 13], where a combination of weighted encoding of a digit set and a power-of-2 radix allows efficient implementation of arithmetic with redundant operands.

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. 1*a*). Using a small hollow circle (○) to denote a negabit allows us to visualize 2's-complement numbers (Fig. 1*b*), negabinary or radix-(−2) numbers (Fig. 1*c*), and other representations formed by a specific mix of posibits and negabits in an extended dot notation. Redundant representations, with multiple dots in some positions, allow us to take advantage of their carry-free arithmetic property. In addition, representational redundancy can lead to faithful representation of arbitrary digit sets such as [0, 9] (Fig. 1*d*) and [−6, 6] (Fig. 1*e*), which would otherwise have to be encoded using the wider ranges of [0, 15] and [−8, 7] (e.g. as in Figs. 1*a* and *b*), respectively.

It is the mixed use of arbitrary combination of posibits and negabits in various bit positions (e.g. Fig. 1*e*) that forms the focus of this paper. In general, there may be several WBS encodings for a given digit set. For example, a collection of six negabits and six posibits, all weighted 1, also faithfully represent [−6, 6]. However, two-deep or canonical WBS encodings (i.e. those containing at most two bits in each binary position, as in Fig. 1) are preferred because of the possibility of more efficient implementation of arithmetic operations [10].

Any non-canonical WBS encoding can be converted to an equivalent canonical one using the range-preserving

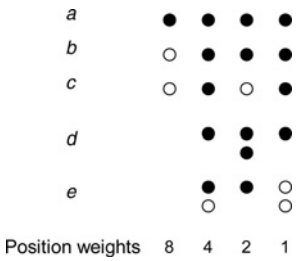

Fig. 1 *Number representations of varying ranges in extended dot notation*

a Unsigned binary number in [0, 15]
b Two's-complement number in [−8, 7]
c Negabinary number in [−10, 5]
d A faithful encoding of digit set [0, 9]
e A failthful encoding of digit set [−6, 6]

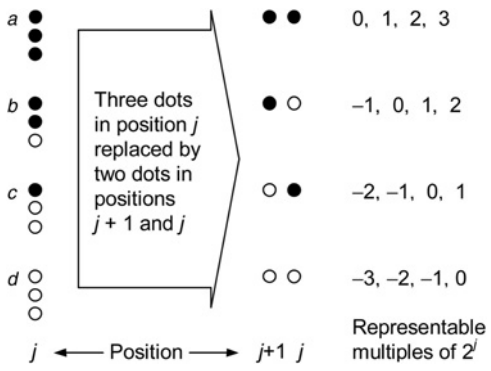

Fig. 2 *Range-preserving transformations for WBS encodings*

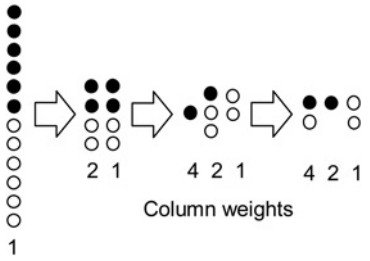

Fig. 3 *Four faithful WBS representations of the digit set [−6, 6]*

transformations of Fig. 2 to redistribute the extra dots in columns with more than two 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 $[-\alpha, \beta]$ can be faithfully represented using canonical (two-deep) WBS encoding; simply encode the 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. Fig. 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 four results from aligning the corresponding positions of the two-deep operands. Finally, subtraction can be converted to addition by changing posibits to negabits, and vice versa, in the subtrahend.

In this paper, we aim to promote the use of WBS and inverted encoding of negabits (IEN) in the design of digital arithmetic circuits. Therefore we reproduce relevant examples (Examples 1 and 3 in Section 2, and Examples 4–6 in Section 4) from some of the previously published works [12, 14], and our corresponding conference paper [15]. Furthermore, we present an efficient method for converting any canonical WBS encoding to any desired two-deep encoding (see Example 7 in Section 4). In particular, conversion from two-deep WBS to BSD is important because of ease of converting the latter to its equivalent 2's-complement number. As another new example, we present the design strategy for a modulo-$(2^n + 1)$ multiplier (see Example 8 in Section 4), where we show how WBS and IEN techniques help in reducing the design effort and increase design reliability.

2 Representations and algorithms

One of the important notions in the design of digital circuits for arithmetic-intensive and other applications is that of bit compression. For example, a half-adder (HA) can be

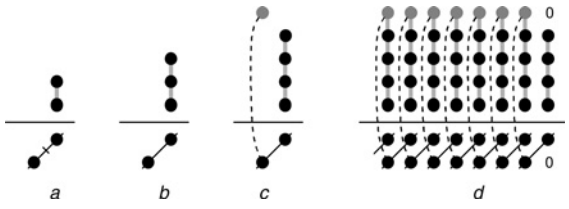

Fig. 4 *Basic bit compression and redistribution operations in dot notation*

a HA
b FA
c (4; 2)
d (4; 2) in multiple columns

doi: 10.1049/iet-cdt.2011.0059

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. 4*a*. Similarly, a full-adder (FA), also known as (3; 2)-counter, compresses three dots to two dots, corresponding to the sum and carry bits, as shown in Fig. 4*b*. When applied to multiple columns of three dots at once, this leads to reduction of three binary numbers to two binary numbers in a scheme known as carry-save addition. Finally, the (4; 2)-compressor of Fig. 4*c* is capable of compressing a column of four dots into two dots in adjacent columns, plus a carry bit that is sent to the next higher column. This is possible because four dots in a column when combined with an incoming carry (the fifth dot) represent a value in [0, 5] and thus can be represented by one dot of the same weight and two dots of double that weight (Fig. 4*d*).

In the rest of this section, we present three examples to illustrate the applications and advantages of our unifying framework.

(*Unsigned and 2's-complement multiplication*): Steps of conventional unsigned multiplication and 2's-complement multiplication based on Baugh–Wooley [7] scheme for 4-bit operands are depicted in Figs. 5*a* and *b*, respectively. The essence of the Baugh–Wooley conversion step 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 [5], using the identity $(0-1)_{\text{two}} = (-1 \quad 1)_{\text{two}}$. However, Fig. 5*b* is a demonstration of our new interpretation of the Baugh–Wooley technique, with the use of negabit symbols. The three horizontally aligned negabits, in the bottom row of partial products, collectively represent a negative number (e.g. $-xyz$) that can be replaced by a 2's-complement number (e.g. $-1\,\bar{x}\,\bar{y}\,\bar{z}+1$ composed of a '-1' in the column of the leftmost posibit, inversion of the three negabits regarded as posibits, and a '1' in the intermediate column. The same is true for the other three diagonally aligned negabits. Therefore we get at all-posibit partial products, with the two 1's (-1's) becoming a single 1 (-1) in the next more significant column. The partial products with negabits are only for illustration purpose and the all-posibit partial products are directly generated via an AND/NAND matrix, where the NAND gates produce the posibits that correspond to the aforementioned negabits.

(*Partial-product generation in mod-*(2^n+1) *multiplication*): In mod-(2^n+1) multiplication, each posibit c in position $n+i$ $(0 \le i \le n-1)$ of a partial product is ruled by (1), where c^- denotes a negabit whose arithmetic value equals $-c$ and $\bar{c} = 1-c$. Thus, a posibit c of weight 2^{n+i} may be removed and a negabit c^- of weight 2^i introduced instead

$$|2^{n+i}c|_{2^n+1} = |2^i(2^n+1-1)c|_{2^n+1} = 2^i(-c) = 2^i c^- = 2^i\bar{c} - 2^i \quad (1)$$

Based on this transformation, Fig. 6 illustrates a dot-notation representation of mod-(2^n+1) partial product generation for $n=4$. The product $\bullet_4 \times \bullet_4$, of the most-significant bits (MSBs), originally weights 2^{2n} and since $2^{2n} = (2^n+1)(2^n-1)+1$ it appears as a posibit in position 0. Note that each operand is represented with $n+1$ bits, with MSB being 1 only for 2^n. The previously proposed partial product reduction schemes (e.g. [16])

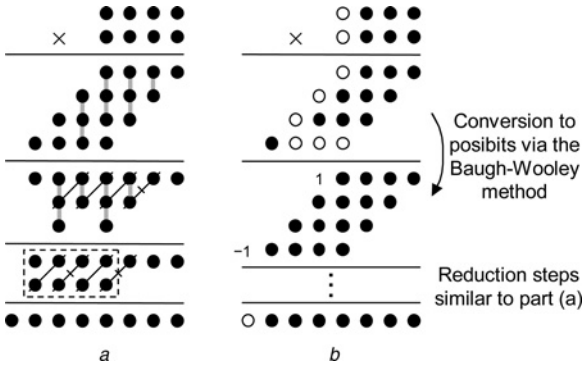

Fig. 5 *Integer multiplication viewed as bit compression and addition*

a Unsigned
b Two's complement

require that each negabit c^- be replaced with a posibit $\bar{c}$ and a constant negabit -1, a scheme that entails a non-trivial algorithm to compute the collective value of the constant negabits to form a new operand below the partial products. We will see in Section 3 that IEN allows the authors to manipulate the negabits directly, thus widening the design space, saving some design effort, possibly circuit resources and latency.

(*Booth recoding for radix-4 multiplication*): Booth recoding can be more readily understood with the help of WBS. For example, the modified (i.e. radix-4) Booth recoding, is essentially converting a 2's-complement number to its equivalent number in the minimally redundant radix-4 number system (i.e. conversion from the digit set [0, 3] to [−2, 2]). Let $Y = y_{2n-1}^{-}y_{2n-2}\cdots y_{2i+1}y_{2i}y_{2i-1}\cdots y_1y_0$ where a '$-$' superscript identifies a negabit, represents a 2's complement number to be Booth recoded. This can be accomplished by simply shifting each posibit y_{2i+1} one binary position to the left and filling its place by a negabit y_{2i+1}^{-} with the same logical state (see Fig. 7). This process leads to a canonical WBS representation of the desired minimally redundant radix-4 number, where the ith radix-4

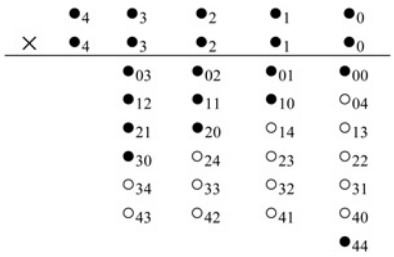

Fig. 6 *Partial product generation in modulo-17 multiplication*

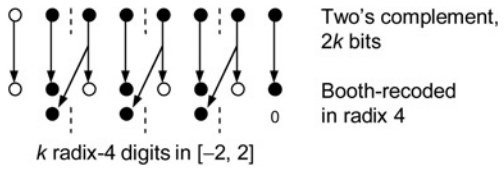

Fig. 7 *Justification of modified Booth's recoding via extended dot notation*

digit is represented by a doubled negabit y^-_{2i+1} and two posibits y_{2i} and y_{2i-1}. Note that this transformation is just an easier interpretation of the conventional modified Booth recoding, which is of pedagogical value. Otherwise, (2) for generating the Booth selection signals S_i (sign), T_i (twice or shifted) and O_i (One) remains the same

$$S_i = y^-_{2i+1}, \quad T_i = \overline{y^-_{2i+1}} y_{2i} y_{2i-1} \vee y^-_{2i+1} \overline{y_{2i} y_{2i-1}}, \quad O_i = y_{2i} \oplus y_{2i-1} \tag{2}$$

3 Inverted encoding of negabits

The conventional representation of negative numbers or negative component of redundant numbers is based on assigning negative weights to whole numbers or digits. For example, sign-magnitude representation uses a separate sign bit that can be interpreted as the sign of the whole number. In the n-bit 2's-complement representation of binary numbers, the most-significant bit is considered to have the negative weight -2^{n-1}. The $\langle n, p \rangle$ encoding of a binary signed digit in radix-2 position i uses a pair of $\pm 2^i$-weighted bits [9]. Finally, the digit set $[-9, 9]$ is represented as two 4-bit numbers with oppositely signed weights [17].

We have previously introduced the concept of negative bit (negabit for short) instead of negatively weighted bit [18], which simplifies some number representations, as was used in Figs. 1–3 (Section 1) and Examples 1–3 (Section 2). However, recalling Example 2, in reducing the partial products of Fig. 6, one cannot directly use the standard reduction cells such as HAs, FAs and other compressors and counters. A simple solution for this kind of problem was offered in [10] via the introduction of IEN, that is, assigning the logical state 0 (1) to arithmetic value −1 (0). This is the opposite of conventional encoding in 2's-complement numbers, where the most-significant bit equals 0 (1) for positive (negative) values. This reverse convention turns out to be quite rewarding. It has been shown elsewhere [10] that with IEN any equally weighted triple mix of posibits and negabits can be summed up correctly with a standard full adder. Half adders and other counters and compressors can similarly accept any mix of posibits and negabits, generating correct results as if they were functioning on posibits only (see Figs. 8 and 9, with further relevant explanations to follow).

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 optimised based on different criteria (e.g. latency, area and power) for a multitude of implementation technologies [19]. To process conventional negabits in the same way, inverters are typically inserted on some inputs/outputs of standard cells [20], adding some latency, compromising circuit regularity and introducing the need for area/power re-optimisation. 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 lead to non-trivial speed improvement.

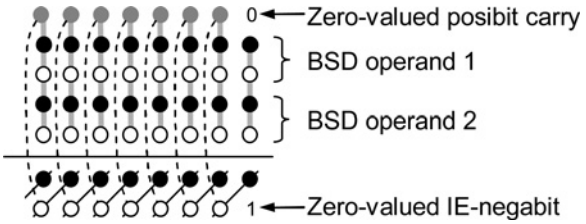

Fig. 9 *Using (4; 2)-compressors as redundant binary adders*

The key to improvements resulting from IEN is the property that their logical and arithmetic values vary in the same direction. Representing the value −1's logical 0 and the value 0 as logical 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. For example, Fig. 8 shows the schematic representation of a full (half) adder used to combine a set of 3 (2) bits, which includes from 0 to 3 (2) negabits [10]. 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 in all seven cases depicted in Fig. 8. Recall that a standard full (half) adder operates on posibits in a way that enforces the identity $x + y + c_{in} = 2c_{out} + s$ $(x + y = 2c_{out} + s)$, with numerical details shown in (3). For clarity in studying Fig. 8, the reader is reminded that a '−' superscript identifies a negabit.

$$\begin{aligned} &\text{(a) } 0+0+0=0+0 \quad \text{(b) } 0+0+1=0+1 \\ &\text{(c) } 0+1+1=2+0 \quad \text{(d) } 1+1+1=2+1 \\ &\text{(e) } 0+0=0+0 \quad \text{(f) } 0+1=0+1 \\ &\text{(g) } 1+1=2+0 \end{aligned} \tag{3}$$

Fig. 9 depicts an instance of the standard (4; 2)-compressor of Fig. 4*c* 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 [21] and has a latency of three XOR gates, the same as a conventional (4; 2) compressor, with gate counts also being comparable. It is worth noting that the design just

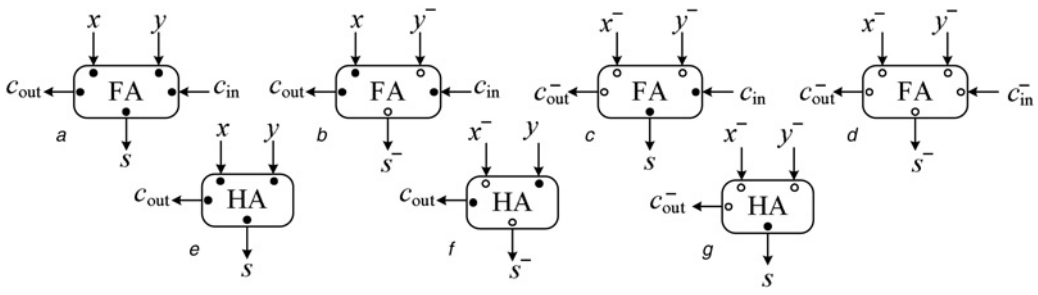

Fig. 8 *Full-adders and half-adders as universal combiners of posibits and negabits*

doi: 10.1049/iet-cdt.2011.0059

mentioned uses a 2-bit encoding that is the same as the $\langle n, p\rangle$ encoding with inversely encoded negabits (IE-negabits). However, because of the *ad hoc* approach, the design effort is much greater and the resulting circuits cannot benefit from regular performance improvements on standard cells. The universal (4; 2)-compressors of Fig. 9, on the other hand, can be replaced by any available (4; 2)-compressor circuit.

There are also other highly optimised compressors, exemplified by (5; 2) compressors [Chan04], that may prove beneficial in reducing arrays of mixed posibits and negabits, where the depth is not a multiple of 4 (see Example 8). This is yet another confirmation that the use of highly optimised standard cells is preferable to *ad hoc* designs, whenever possible.

4 Implementation and application

Use of IEN and WBS encoding has been shown to enhance creativity in designing arithmetic circuits that often leads to better performance as well. For example, hybrid-redundant adders [22] have been redesigned in a universal manner in [23], from which the design of a minimally redundant adder is reproduced in Example 4. A highly efficient maximally redundant signed-digit adder [12], and a less redundant version based on IEN [13], has been recently proposed, where the use of IEN leads to power and area savings via the elimination of the need for sign extension. Another interesting property of WBS encoding and IEN is the value-preserving transformation of a collection of bits that represent a symmetric range of negative and positive values. As it turns out, in such cases, interchanging the posibits with negabits and vice versa without changing the logical states preserves the actual value of the bit collection [12]. This property is further explained in Example 5 and used in Example 6 to lead to yet another interpretation of Baugh–Wooley 2's complement multiplication scheme.

Use of off-the-shelf HAs, FAs, counters and compressors, exemplified by (4; 2)-compressors, to reduce the depth of a WBS encoding (e.g. partial product reduction) often leads to a canonical two-deep WBS representation with an arbitrary mix of posibits and negabits. This mix can be added via any conventional carry-propagate adder (CPA). However, the one-deep result can hold posibits and negabits in arbitrarily weighted positions. This undesired phenomenon can be avoided via a small preprocessing step prior to CPA, as described in Example 7 for the first time in this paper. Finally, we take up Example 2 and show how the partial products of a modulo-$(2^n + 1)$ multiplier can be summed up with standard reduction cells via IEN (see Example 8).

(*Simplified minimally redundant radix-2h SD addition*): A straightforward implementation of the digit-level addition algorithm for radix-2^h signed digit (SD) number systems [9] implies three O(log h)-latency carry-propagating operations in sequence. A radix-2^h signed digit set is often denoted by the integer interval $[-\alpha, \alpha]$, with the redundancy index $\rho = 2\alpha + 1 - 2^h$. For $\rho \geq 2$, the sum digit in position i depends on the operand digits in the same position and those of position $i - 1$, vis-à-vis carry-free addition [9]. For $\rho = 1$ (e.g. with $\alpha = 2^{h-1}$), however, it depends also on the operands digits in position $i - 2$. Here, we provide 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 $i - 2$.

Fig. 10 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 non-redundant positions [14]. Recall that ordinary hybrid redundancy uses only posibits in such positions [22]. The particular number system shown is periodic, with a period of four positions, and thus corresponds to a radix-16 (i.e. $h = 4$) generalised signed-digit representation with the minimally redundant digit set $[-8, 8]$. The first stage of the addition process, depicted in Fig. 10, converts pairs of negabits in the input operands, with the exception of those in the leftmost position, to 5-bit 2'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 2's-complement main part do not depend on the operand digits in position $i - 2$. This conforms to the general result in [9] under limited-carry addition with $\rho = 1$. That is, as a whole, each sum-digit is a function of digits in three consecutive positions of the operands. However, the main part (the stored posibit) of position i depends only on the digits in positions i and $i - 1$ ($i - 1$ and $i - 2$). Therefore the digit dependency is the same as that existing for carry-free cases of $\rho \geq 2$.

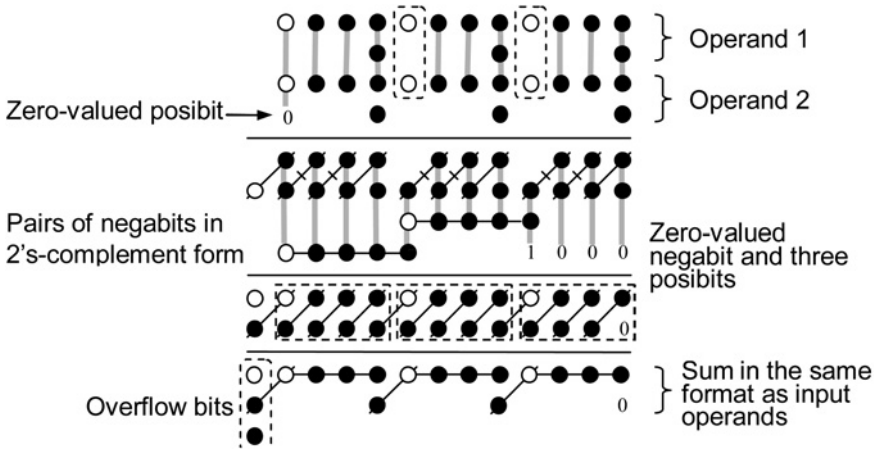

Fig. 10 *Dot-notation representation of symmetric hybrid-redundant addition*

(*Value-preserving polarity inversion in faithfully represented balanced signed digits*): Consider an $\langle n, p\rangle$ encoded binary signed digit and also the rightmost representation of the digit set [−6, 6] in Fig. 3. Such faithfully represented signed digit sets are invertible by exchanging posibits and negabits, as shown in Fig. 11*a*. We know that identical bit assignments to both representations of Fig. 11*a* yield equal arithmetic values [12]. 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. 11*b* represents the essence of the transfer extraction scheme of a radix-16 maximally redundant signed-digit (MRSD) adder, with each redundant signed digit in [−15, 15] and encoded as a 5-bit two's-complement number. The carry-free addition process requires the extraction of a weight-16 transfer digit $t_{i+1} \in [-1, 1]$ from 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 arithmetical value of the transformed bits, given that the transformation in position j increases (decreases) the arithmetical value by 2^j. This cost-free transformation (inversion occurs in the way the bits are viewed, rather than via an inversion gate) 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 [12].

A similar value-preserving polarity inversion can be used for yet another simple justification of the Baugh–Wooley method. This is illustrated in Fig. 12, in which the partial products of Part I are generated with NAND gates, where exactly one of the two bits is a negabit. Part II is exactly the same as Part I, except for a negabit/posibit position swap for better illustration of the dashed-border bit collection whose polarities are inverted in Part III. The posibit constant 0 and the IE-negabit constant 1 (with arithmetic worth of 0) are inserted (see Part IV) to allow for the second transformation, again illustrated by dashed-border bit collections. Note that after polarity inversion, the constant 0 (1) is represented by a negabit −1 (posibit 1) in V, where the bit collection is exactly the same as the bottom part of Fig. 5*b*. The reader is reminded that, as in Fig. 5*b*, the bits of Part V are directly generated via a NAND/AND matrix from the bits of the 2's-complement operands.

(*Identical partial product reduction tree for unsigned and 2's-complement multiplication*): Fig. 13 represents an implementation of 4-bit 2's-complement multiplication using energy efficient NAND gates [24], wherever the inputs of a partial product generation cell are of opposite polarities, leading to a tree reduction part assuming the use of IE-negabits for the partial products. The final adder uses a half adder in position 3 and full adders in the next three positions [see cell (c) in Fig. 8 for an explanation on how the full adders work]. Therefore the required circuitry is exactly the same as the one needed for Fig. 5*b*. However, the main advantage here is again pedagogical, given the simple concept behind the scheme of Fig. 13*b* in comparison to that of Baugh–Wooley (Fig. 5*b*).

(*Conversion from 2-deep WBS to 2's complement*): The operation of the final adder in Fig. 13*b* is a special case of

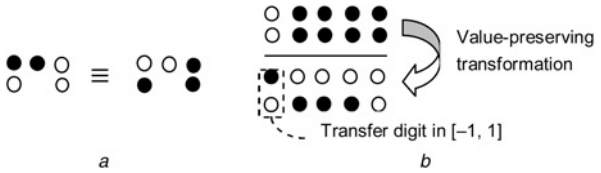

Fig. 11 *Cost-free value-preserving transformations*

a Digit set [−6, 6]
b Deriving a transfer in [−1, 1 and a residual in [−14, 14] from two digits in [−15, 15]

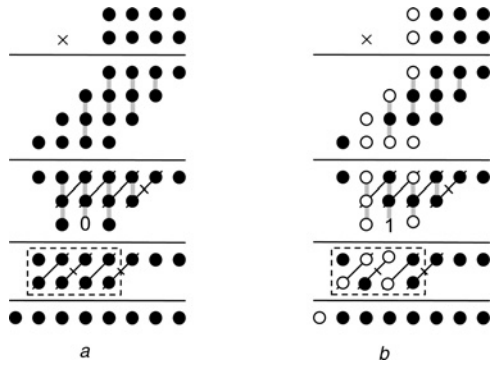

Fig. 13 *Unsigned or signed multiplication with identical tree reduction circuits*

a Unsigned
b Two's complement

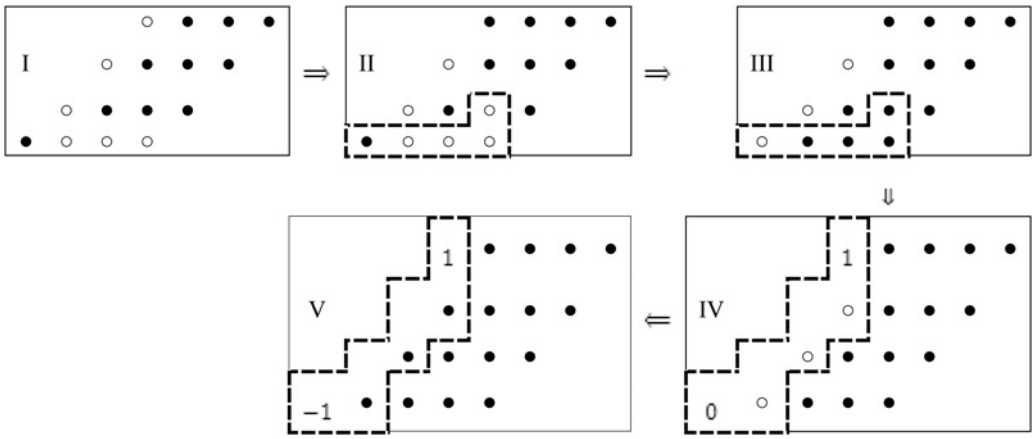

Fig. 12 *Alternative justification of the Baugh–Wooley multiplication method*

doi: 10.1049/iet-cdt.2011.0059

the general problem of converting a two-deep WBS number to its 2's complement equivalent. It is easy to see that the simple adder solution used for Fig. 13*b* works for converting an $\langle n, p\rangle$-encoded BSD number, with IE-negabits, to 2's complement, provided a forced carry-in is used in the form of an IE-negabit 1^- with arithmetic value 0. For example, the situation for a conversion from 4-bit BSD to 5-bit 2's-complement representation is depicted in Fig. 14, where each of the four full adders receiving two negabits and one posibit produces a posibit sum and a negabit carry, as in cell (c) in Fig. 8. However, this simple method for the special case of BSD numbers does not always work for the aforementioned general case.

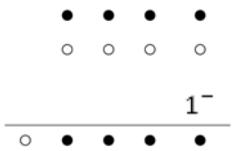

Fig. 14 *BSD-to-2CL conversion via binary addition with* $c_{in} = 1$

One solution is to first convert the source two-deep WBS number to its BSD equivalent, and then follow this by simple addition. The conversion to BSD, as summarised in Table 1, is possible by means of an inverter (I), a half-adder (HA), or an excess-1 half-adder (HA^{+1}) in each binary position $i \geq 0$. The cell choice depends on the bit combination in position i and the incoming bit from the possible HA or HA^{+1} in position $i-1$, during the conversion process. Recall that a variable or parenthesised expression with a '$-$' superscript denotes a negabit. The bit that comes into position 0 can be assumed to be either a posibit or a negabit.

Deciding on the actual cell for each binary position at design time is a sequential process that is further explained in Example 8. However, as is evident from the left target column in Table 1 (i.e. position $i+1$), the incoming bit to position $i+1$ does not depend on the one for position i. Therefore the actual run time delay is at most equal to that of a single XOR gate.

(*Modulo-*(2^n+1) *multiplication*): The partial product generation (PPG) scheme of Fig. 6 allows energy-efficient

Table 1 Conversion from 2-deep WBS to BSD

Case	Dot	Symbol	Target (in 2 columns)				Cell type
			Dot	Symbol	Dot	Symbol	
1	●s ●	x y	●	x	○ ●	$(\bar{x})^-$ y	I
2	●s ○	x y^-			● ○	x y^-	-
3	○s ●	x^- y			○ ●	x^- y	-
4	○s ○	x^- y^-	○	x^-	● ○	$\bar{x}$ y^-	I
5	●s ●s ●	x x' y	●	$x \vee x'$	○ ●	$(x \odot x')^-$ y	HA^{+1}
6	●s ●s ○	x x' y^-	●	xx'	● ○	$x \oplus x'$ y^-	HA
7	○s ○s ●	x^- x'^- y	○	$(x^- \vee x'^-)^-$	○ ●	$(x^- \odot x'^-)^-$ y	HA^{+1}
8	○s ○s ○	x^- x'^- y^-	○	$(x^- x'^-)^-$	● ○	$x^- \oplus x'^-$ y^-	HA
9	●s ○s ●	x x'^- y	●	x x'^-	○ ●	$(x \oplus x'^-)^-$ y	HA
10	●s ○s ○	x x'^- y^-	○	$(x \vee x'^-)^-$	● ○	$x \odot x'^-$ y^-	HA^{+1}
Position	i		$i+1$		i		i

$\oplus$ is XOR; $\odot$ represents XNOR; I stands for inverter s-indexed bits are source bits and others in the same column are incoming bits

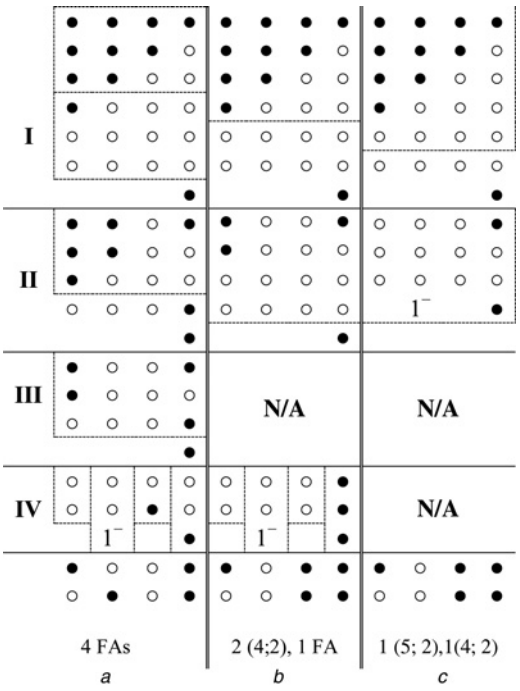

Fig. 15 *Three modulo-17 PPR*

a 4 FAs
b 2 (4; 2), 1 FA
c 1 (5; 2), 1 (4; 2)

NAND gates to replace the original AND gates for generating end-around partial product IE-negabits. Fig. 15*a* depicts a carry-save adder (CSA) scheme for partial product reduction (PPR), where each dashed box in Stages I–III represents an n-bit CSA ($n = 4$) and stage IV simply converts the result to a representation that can lead to the desired product via any n-bit adder. This stage uses FA, HA, HA^{+1}, HA, from right to left, respectively. The performance is enhanced in Figs. 15*b* and *c* by also allowing the use of (4; 2) and (5; 2) compressors. Note that the cells used in each of the three designs belong to the set of standard arithmetic building blocks, often available in multiple optimised forms for mapping onto FPGAs, custom VLSI implementation and other digital design styles [25].

The number of partial products for the scheme explained above in Example 8 is $n + 3$. This can be reduced to n at the expense of more complex three-gate-delay PPG and custom wiring of the inputs to CSA tree, so as to keep the extra PPG delay off the critical path [16]. Unfortunately, to apply a similar scheme in that work for reduction trees with more efficient (4; 2) and (5; 2) compressors, as in Figs. 15*b* and *c*, would require redesigning the compressors' internal wiring, thus jeopardising all the test, optimisation and synthesis efforts implicit in their designs. Furthermore, custom wiring, as used in the case of $n = 4$ [16], may not always be beneficial. For example, the left side of Fig. 16 shows the number of CSA (FA) levels, and the corresponding delay in terms of XOR gates, for $p \in [4, 13]$ partial products (i.e. $n \in [4, 10]$). Only for $n = 4$, two levels are saved compared with $n + 3 = 7$ partial products. In cases where $n \in [5, 9]$, only one level is saved compared with $n + 3 \in [8, 12]$. There are five CSA levels for both $n = 10$ and $n + 3 = 13$ partial products, implying no saving. The right side of Fig. 16 shows the number of reduction stages and the reduction cell types used. For example, for $n + 3 = 7$, the critical delay path consists of two (4; 2) compressors based on Fig. 15*b*. Note that the figures presented in Fig. 16consider neither Stage IV of Fig. 15 nor the half adder stage (because of constant partial product) of [16].

The arguments regarding the dubious advantage of custom wiring notwithstanding, the method of [16] can be applied here to achieve an n-deep WBS partial product matrix for modulo $2^n + 1$ multiplication as in Fig. 17, with due explanations to follow.

The two negabit entries in the same column of Fig. 6, with one of the indices being 4 (e.g. $\circ_{14}$ and $\circ_{41}$) and any other negabit in the same column (e.g. $\circ_{32}$), can be replaced by a new negabit which represents the true sum of the original negabits (e.g. $\circ_1$ in Fig. 17). This compression of three negabits into one is possible because when the most-significant bit of a modulo-$(2^n + 1)$ operand is 1, its other n bits are 0's. Thus, $\bullet_{i4} + \bullet_{4i} + \bullet_{3(i+1)} = \bullet_{i4} \vee \bullet_{4i} \vee \bullet_{3(i+1)}$, implying that $\circ_i$ can be obtained by NORing $\bullet_{i4}$,$\bullet_{4i}$ and $\bullet_{3(i+1)}$, for $0 \le i \le n - 1$. For the same reason, it holds that $\bullet_{00} + \bullet_{44} = \bullet_{00} \vee \bullet_{44}$, and thus the sum is represented by a single posibit $\bullet_0$ in Fig. 17. To avoid the depth of 5 in the most-significant column of Stage II in Fig. 17, we use the trick offered in [16], replacing $\circ_3$ with $\bullet_3$ in the same column and another one in the rightmost column based on (4). The reason is that the computation of $\circ_3$, as described by (4), can be reformulated as in (5). However, as before, we have $\bullet_{34} + \bullet_{43} + \bullet_{30} = \bullet_{34} \vee \bullet_{43} \vee \bullet_{30}$. The result is shown as the single posibit $\bullet_3$ of Stage III in Fig. 17. Similarly, we have $\bullet_0' = \bullet_0 + \bullet_{34} + \bullet_{43} = \bullet_{00} \vee \bullet_{44} \vee$

	p	Reduction cells			
		FA	# XOR	(5; 2), (4; 2), FA	# XOR
n	4	2	4	0, 1, 0	3
	5	3	6	1, 0, 0	4
	6	3	6	0, 2, 0	6
	7	4	8	0, 2, 0	6
	8	4	8	0, 2, 0	6
	9	4	8	1, 1, 0	7
$n+3$	10	5	10	1, 1, 0	7
	11	5	10	2, 0, 0	8
	12	5	10	0, 3, 0	9
	13	5	10	0, 3, 0	9

Fig. 16 *Partial product reduction with different reduction cells*

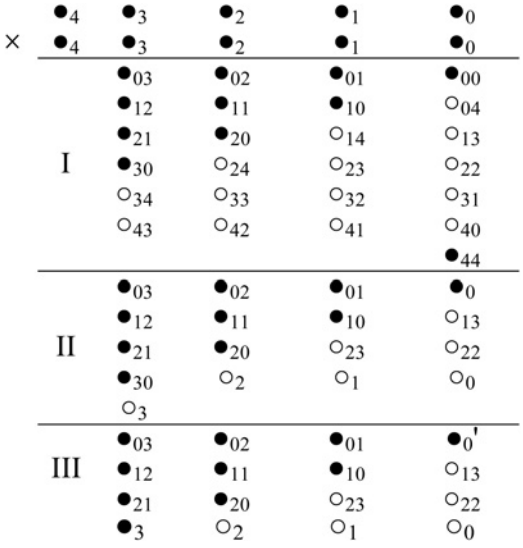

Fig. 17 *Efficient modulo-17 PPG*

doi: 10.1049/iet-cdt.2011.0059

$\bullet_{34} \vee \bullet_{43}$

$$|2^7(\bullet_{34} + \bullet_{43})|_{17} = |8 \times (17 - 1)(\bullet_{34} + \bullet_{43})|_{17} = -8(\bullet_{34} + \bullet_{43}) = 8\circ_3 \quad (4)$$

$$|2^7(\bullet_{34} + \bullet_{43})|_{17} = |(17 \times 7 + 9)(\bullet_{34} + \bullet_{43})|_{17} = (8 + 1)(\bullet_{34} + \bullet_{43}) \quad (5)$$

With the four-deep partial product matrix of Stage III in Fig. 17, we need only two CSA levels to obtain a two-deep matrix. However, a half-CSA stage is required to convert the latter to a form suitable for input to any binary adder, including the inverted end around carry adder of [16]. A similar half-CSA stage is used in [16] because of an extra constant partial product (see Example 2). Note that Stages I–II are used merely for illustration; the actual PPG circuitry would directly generate the bit matrix of Stage III.

Advantages of the modular multiplier design of Example 8 in comparison with previous designs (e.g. [16]) include easier exploration of the design space, simpler conceptual design and use of a variety of standard optimised cells.

5 Conclusions

By using several examples from our previous works and introducing two new case studies, we have demonstrated the advantages of intermixing posibits and negabits as elements of weighted bit sets for use in representing signed digit sets or for direct representation of integers in a desired range. More specifically, the advantages fall into the two categories of pedagogical (better understanding) and practical (more efficient hardware realisation). 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 improvement in design parameters such as cost, speed and/or power consumption. These practical benefits are direct consequences of our unified design strategy, based on the exclusive use of highly optimised standard building blocks or cells, for realising arithmetic operations on representations composed of weighted posibits and negabits.

The design flow for implementing an arithmetic operation on redundant operands based on the digit set $[-\alpha, \beta]$ may consist of the following steps:

1. Start with α negabits and β posibits, all equally weighted (i.e. in the same column) and use the transformation of Fig. 2 in multiple rounds, until a two-deep WBS encoding of the digit set is obtained.
2. Design the required arithmetic circuits as if all the bits were posibits, but consider the polarity of the bits in the intermediate results as the output bits of the relevant cells (e.g. those in Fig. 8).
3. Use Table 1 to convert the final WBS encoding to the desired output format.

Even though we have used this method, and the associated IEN, 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. We have augmented the aforementioned known design examples with two new applications specifically developed for this presentation: WBS-to-BSD-to-2CL format conversion, and modulo-$(2^n + 1)$ multiplication. The examples offered here are by no means exhaustive. Exploration of new application domains constitutes part of our plans for future work.

Our discussion in this paper 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 use of highly optimised standard cells that are available in the literature and in various design libraries. 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. A quantitative assessment of the benefits would be possible only for specific applications, after full circuit-level implementation. We have not done this in the current paper, but instead refer the reader to our previous publications, cited in the references, which do offer quantitative evaluations and comparisons.

6 References

1 Taylor, F.J.: 'Digital filter design handbook' (M. Dekker, 1983)
2 Soderstrand, M.A., Jenkins, W.K., Jullien, G.A., Taylor, F.J. eds.: 'Residue number system arithmetic' (IEEE Press, 1986)
3 Omondi, A., Premkumar, B.: 'Residue number systems: theory and implementation' (Imperial College Press, 2007)
4 Avizienis, A.: 'Signed-digit number representation for fast parallel arithmetic', *IRE Trans. Electron. Comput.*, 1961, **10**, pp. 389–400
5 Parhami, B.: 'Computer arithmetic: algorithms and hardware designs' (Oxford University Press, 2010, 2nd edn.)
6 Meyer-Baese, U.: 'Digital signal processing with field programmable gate arrays' (Springer, 2007, 3rd edn.)
7 Baugh, C.R., Wooley, B.A.: 'A two's complement parallel array multiplication algorithm', *IEEE Trans. Comput.*, 1973, **22**, (6), pp. 1045–1047
8 Koren, I., Maliniak, Y.: 'On classes of positive, negative, and imaginary radix number systems', *IEEE Trans. Comput.*, 1981, **30**, (5), pp. 312–317
9 Parhami, B.: 'Generalized signed-digit number systems: a unifying framework for redundant number representations', *IEEE Trans. Comput.*, 1990, **39**, (1), pp. 89–98
10 Jaberipur, G., Parhami, B., Ghodsi, M.: 'Weighted two-valued digit-set encodings: unifying efficient hardware representation schemes for redundant number systems', *IEEE Trans. Circuits Syst. I*, 2005, **52**, (7), pp. 1348–1357
11 Jaberipur, G., Parhami, B.: 'Unified approach to the design of modulo-$(2^n \pm 1)$ adders based on signed-LSB representation of residues'. Proc. 19th IEEE Int. Symp. Computer Arithmetic, 2009, pp. 57–64
12 Jaberipur, G., Gorgin, S.: 'An improved maximally redundant signed digit adder', *Comput. Electr. Eng.*, 2010, **36**, (3), pp. 491–502
13 Gorgin, S., Jaberipur, G.: 'A family of signed digit adders'. Proc. 20th IEEE Symp. on Computer Arithmetic, Tubingen, Germany, 25–27 July 2011, pp. 112–120
14 Jaberipur, G., Parhami, B.: 'Constant-time addition with hybrid-redundant numbers: theory and implementations', *Integr. VLSI J.*, 2008, **41**, (1), pp. 49–64
15 Jaberipur, G., Parhami, B.: 'Posibits, negabits, and their mixed use in efficient realization of arithmetic algorithms'. Proc. 15th CSI Int. Symp. on Computer Architecture and Digital Systems, 2010, pp. 3–9
16 Vergos, H.T., Efstathiou, C.: 'Design of efficient modulo $2^n + 1$ multipliers', *IET Comput. Digit. Tech.*, 2007, **1**, (1), pp. 49–57
17 Nikmehr, H., Philips, B.: 'Fast decimal floating-point division', *IEEE Trans. Comput.*, 2006, **14**, (9), pp. 951–961
18 Jaberipur, G., Parhami, B., Ghodsi, M.: 'Weighted bit-set encodings for redundant digit sets: theory and applications'. Proc. 36th Asilomar Conf. on Signals Systems and Computers, November 2002, pp. 1629–1633
19 Chang, C.-H., Gu, J., Zhang, M.: 'A review of 0.18-μm full adder performances for tree structured arithmetic circuits', *IEEE Trans. VLSI Syst.*, 2005, **13**, (6), pp. 686–695

20 Kornerup, P.: 'Reviewing 4-to-2 adders for multi-operand addition', *J.VLSI Signal Process.*, 2005, **40**, (1), pp. 143–152
21 Kim, Y., Song, B.S., Grosspietsch, J., Gillig, S.F.: 'A carry-free 54b × 54b multiplier using equivalent bit conversion algorithm', *IEEE J. Solid-State Circuits*, 2001, **36**, (10), pp. 312–317
22 Phatak, D.S., Koren, I.: 'Constant-time addition and simultaneous format conversion based on redundant binary representations', *IEEE Trans. Comput.*, 2001, **50**, (11), pp. 1267–1278
23 Jaberipur, G., Parhami, B., Ghodsi, M.: 'An efficient universal addition scheme for all hybrid-redundant representations with weighted bit-set encoding', *J. VLSI Signal Process.*, 2006, **42**, (2), pp. 149–158
24 Abid, Z., El-Razouk, H., El-Dib, D.A.: 'Low power multipliers based on new hybrid full adders', *Microelectron. J.*, 2008, **39**, pp. 1509–1515
25 Aguirre-Hernandez, M., Linares-Aranda, M.: 'CMOS full-adders for energy-efficient arithmetic applications', *IEEE Trans. VLSI Syst.*, 2011, **19**, (4), pp. 718–721

doi: 10.1049/iet-cdt.2011.0059