Published in IET Computers & Digital Techniques
Received on 12th September 2013
Revised on 19th June 2014
Accepted on 29th July 2014
doi: 10.1049/iet-cdt.2013.0133

ISSN 1751-8601

Truncated ternary multipliers

Behrooz Parhami

Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA
E-mail: parhami@ece.ucsb.edu

Abstract: Balanced ternary number representation and arithmetic, based on the symmetric radix-3 digit set $\{-1, 0, +1\}$, has been studied at various times in the history of computing. Among established advantages of balanced ternary arithmetic are representational symmetry, favourable error characteristics and rounding by truncation. In this study, we show an additional advantage: that of lower-error truncated multiplication with the same relative cost reduction as in truncated binary multipliers.

1 Introduction

Binary, and more generally radix-2^h, arithmetic is predominant in digital systems, to the extent that we seldom question its superiority or optimality. Decimal arithmetic, which until recently was mostly implemented by means of software, has emerged as a candidate for hardware realisation, with a variety of proposed representations, algorithms and design methods [1]. Radices other than 2^h and 10 have been mostly ignored.

Early in the history of electronic computers, the choice of number representation radix was given much attention, with the binary system prevailing at the end [2, 3]. Historically, ternary (radix-3) representation came quite close to being chosen over binary as the preferred method, eventually losing by a narrow margin. It was argued that under some fairly realistic assumptions about circuit cost and latency, the radix 3 is closer to the theoretically optimal radix e than any other integer radix. However, practical engineering considerations favoured radix-2 over radix-3 [4].

Despite the negative outcome above, the Setun computer, working with balanced ternary arithmetic, was built in 1958 at Moscow State University and found to be quite usable and competitive [5]. The TERNAC computer [6], implemented at State University of New York, Buffalo in 1973, emulated ternary arithmetic operations by representing each ternary number as two binary numbers, one positive and the other negative. Other ternary arithmetic systems and projects, in simple or extended form, have appeared [7–11]. Most such proposals envisage multivalued signals to encode the digit values in balanced or standard ternary. Unfortunately, multivalued signalling and logic, extensively studied within a research community with conferences and a specialised journal, and practically available for decades, has proven uncompetitive in most instances.

The history of a balanced ternary number system and arithmetic goes further back than the efforts just cited. In 1820, John Leslie [12] presented methods for computing in any radix, with an arbitrary digit set. Two decades later, Augustin Cauchy discussed signed-digit numbers in various bases and Leon Lalanne followed by expounding on the virtues of balanced ternary numbers. In 1840, Thomas Fowler (1777–1843), a contemporary of Charles Babbage, chose balanced ternary to build his calculating machine in England [13].

A balanced ternary fraction $x = (x_{-1}\ x_{-2} \dots\ x_{-k})_{\text{three}}$, with each x_i being from the digit set $\{-1, 0, +1\}$, denotes the value $\Sigma_{-k \le i \le -1}\ x_i 3^i$. Values of such k-digit fractions go from $-(1 - 3^{-k})/2$ to $(1 - 3^{-k})/2$, a fully symmetric range approximated by $(-0.5, 0.5)$. Negation, or change of sign, is a digit-wise operation. Addition of balanced ternary digits produces a sum and a carry, both of which are of the same kind. The product of two such digits is similarly a balanced ternary digit. Thus, ternary arithmetic produces block diagrams that are identical to the corresponding binary operations: only circuit-level details within digit-operators change. Balanced ternary arithmetic has favourable error characteristics compared with binary. Chopping produces a maximum error of <1/2 ulp (where 'ulp', or 'unit in least position', is r^{-k} when we keep k fractional digits in radix-r representation), leading to simplified rounding in floating-point arithmetic.

The foregoing history and list of advantages provide motivation for investigating other potential benefits of balanced ternary arithmetic. In this paper, we expose one such advantage in performing truncated multiplication.

Before proceeding with the main topic of truncated multiplication, we should note that the implementation of radix-3 arithmetic with its three-valued digits does not necessarily entail using ternary logic [14], in the same way that decimal or other higher-radix arithmetic systems do not necessitate the use of ten-valued or other non-standard signalling and logic devices. Even though ternary logic is a good match to the digit set used in balanced ternary arithmetic, implementations with binary logic are likely to be more practical and cost-effective in the near future. Research on ternary logic has subsided since its peak in the 1980s, which included the study of low-power metal-oxide semiconductor (MOS) variants [15], but there has been a steady stream of new results over the past three decades, both with mature IC technologies (e.g. [16]) and with less conventional and emerging nanotechnologies (e.g. [17]).

2 Truncated binary multiplication

Array and tree multipliers for k-bit fractional operands x and y typically begin by generating the k^2 bitwise product terms $x_i y_j$ and then proceed to compress these bits into two rows of bits (carry-save result) for final processing by a carry-propagate adder [18]. The final exact result is a $2k$-bit fraction. If we need a k-bit fraction as the final product, as is the case in many signal processing applications, then the $2k$-bit result can be rounded to a k-bit result to obtain the most precise possible k-bit value. Alternatively, if a slight error in the final k-bit product is tolerable, some of the discarded k bits at the lower end of the exact $2k$-bit result can be ignored (not produced) to save on the compression and final addition costs [19]. This strategy often reduces latency and power dissipation as well [20].

Referring to Fig. 1, which depicts an 8×8 fractional binary multiplication, we note that up to half of the bitwise product terms to the right of the grey vertical line can be dropped at the cost of a few ulps of error in the final result. If we simply do not produce those dots, that are collectively worth 8 ulp/2 + 7 ulp/4 + 6 ulp/8 + 5 ulp/16 + 4 ulp/32 + 3 ulp/64 + 2 ulp/128 + ulp/256 = 7.004 ulp, we can reduce the cost, latency and power requirements of the multiplier at the expense of losing <4 bits of precision in the worst case. The expected error would be about 7.004/4 ulp = 1.751 ulp, a precision loss of a tad under 2 bits.

More generally, for a $k \times k$ fractional multiplication, with the partial product terms truncated after position $-k$, the maximum error $\varepsilon_{\max}$, in units of ulp, is

$$\begin{aligned}\varepsilon_{\max} &= k/2 + (k-1)/2^2 + (k-2)/2^3 + \cdots + 1/2^k \\ &= k\left(1/2 + 1/2^2 + 1/2^3 + \cdots + 1/2^k\right) \\ &\quad - (1/4)\left[1 + 2/2 + 3/2^2 + \cdots + (k-1)/2^{k-2}\right]\end{aligned}$$

Instead of evaluating the expression above, it is easier for our needs later in this paper to evaluate the more general radix-r version, where the maximum magnitude of a dropped dot is m, and then setting $r = 2$ and $m = 1$.

$$\begin{aligned}\varepsilon_{\max} &= m\left[k/r + (k-1)/r^2 + (k-2)/r^3 + \cdots + 1/r^k\right] \\ &= mk\left[1/r + 1/r^2 + 1/r^3 + \cdots + 1/r^k\right] \\ &\quad - (m/r^2)\left[1 + 2/r + 3/r^2 + \cdots + (k-1)/r^{k-2}\right]\end{aligned}$$

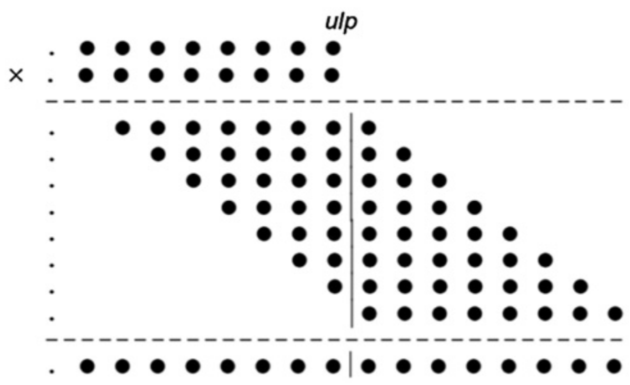

Fig. 1 *Truncated 8 × 8 binary multiplication*

Denoting $1/r$ by z, the sums within square brackets can be replaced by their closed-form equivalents

$$\begin{aligned}z + z^2 + z^3 + \cdots + z^k &= z(1-z^k)/(1-z) \\ 1 + 2z + 3z^2 + \cdots + (k-1)z^{k-2} & \\ = \mathrm{d}\left[z + z^2 + z^3 + \cdots + z^{k-1}\right]/\mathrm{d}z & \\ = \mathrm{d}\left[z(1-z^{k-1})/(1-z)\right]/\mathrm{d}z & \\ = \left[1 + (k-1)z^k - kz^{k-1}\right]/(1-z)^2 &\end{aligned}$$

Substituting the closed-form expressions above, with z replaced by $1/r$, we can simplify $\varepsilon_{\max}$ to

$$\varepsilon_{\max} = m\left[k/(r-1) - 1/(r-1)^2 + r^{-k}/(r-1)^2\right] \qquad (*)$$

For binary arithmetic, we have $r = 2$ and $m = 1$, leading to the following special case of (*), again in units of ulp

$$\varepsilon_{\max}(r{:}2,\, c{:}0) = k - 1 + 2^{-k}$$

The notation 'c:0' in the preceding equation means that no column to the right of the ulp column is formed or evaluated in the truncated multiplier.

The preceding result shows that the maximum error for truncated binary multiplication is $\sim k - 1$, when k is relatively large, and that for $k = 8$, the max-error expression evaluates to $8 - 1 + 2^{-8} = 7.004$ ulp (0.02736 in absolute terms), matching our earlier result.

Instead of dropping all dots to the right of the ulp column, we can keep a few columns and drop the rest, in an effort to reduce the amount of error at the expense of more hardware/energy and greater latency. For example, if we keep only the ulp/2 column and drop the rest, the previously calculated worst-case error is reduced by $k/2$ ulp, but then we incur an error of up to ulp/2 when we round off the additional product bit produced past the ulp column to derive the final k-bit result. Thus, the max error in this case is about half the previous value

$$\varepsilon_{\max}(r{:}2,\, c{:}1) = k - 1 + 2^{-k} - k/2 + 1/2 = (k-1)/2 + 2^{-k}$$

For the preceding example, the maximum error is 3.504 ulp for $k = 8$. This represents a loss of about 2 bits in precision. The average error in this case is 3.504/4 ulp = 0.851 ulp, a tad <1 bit loss in precision. The effect of the average or maximum error on computation results is application-dependent and has been studied by various researchers in different application contexts [21–23]. An alternative is to use fixed error compensation to transform an asymmetric error range $[0, \varepsilon]$ to a nearly symmetric range $[-\gamma, \varepsilon - \gamma]$, where the worst-case error magnitude is $\max(\gamma, \varepsilon - \gamma)$, or to use variable compensation to reduce the error even further. We will discuss these approaches after discussing ternary multiplication.

3 Truncated ternary multiplication

In discussing ternary arithmetic, we opt to represent the three digit values as $N(-1)$, $Z(0)$, and $P(+1)$ to distinguish digit values from possible encodings of these values, which may involve the use of posibits in {0, 1} and negabits in {−1, 0} [24]. Thus, whereas −1 is one of the two possible values of a single negabit, the ternary digit N may have a 2-bit encoding consisting of a negabit and a posibit. Using the negabit/posibit scheme, N will be encoded as 10, Z as 00 or 11 and P as 01.

Given that the product of two balanced ternary digits is a balanced ternary digit (Fig. 2), the multiplication dot diagram in Fig. 1 remains valid for balanced ternary multiplication. In fact, the balanced ternary number system is the only non-binary representation for which the preceding statement can be made.

If we simply do not produce the dots to the right of the grey vertical line, that are collectively worth no more than 8 ulp/3 + 7 ulp/9 + 6 ulp/27 + 5 ulp/81 + 4 ulp/243 + 3 ulp/729 + 2 ulp/2187 + ulp/6561 = 3.750 ulp in magnitude, we can reduce the cost, latency and power requirements of the multiplier at the expense of losing the equivalent of about 3 bits of precision in the worst case. Thus, the error because of truncation is in [−3.750, 3.750] ulp, having the expected or mean value 0.

More generally, for a $k' \times k'$ fractional multiplication truncated after the k'th position, the maximum error magnitude, in units of ulp, is obtained from (*) as

$$\varepsilon_{\max}(r{:}3,\, c{:}0) = k'/2 - 1/4 + 3^{-k'}/4$$

We have used k' instead of k to denote the number of radix-3 digits to account for the fact that, with comparable dynamic ranges, radix-2 and radix-3 representations possess different widths; more on this point later.

The result above shows that the maximum error is $\sim k'/2 - 1/4$ ulp when k' is relatively large and that for $k' = 8$, the max-error expression evaluates to $8/2 - 1/4 + 3^{-8} = 3.750$, matching our earlier result.

Instead of dropping all the dots appearing to the right of the ulp column, we can keep a few columns and drop the rest to reduce the amount of error at the cost of more hardware and greater latency. For example, if we keep only the ulp/3 column and drop the rest, the previously calculated worst-case error is reduced by $k/3$ ulp, but then we incur an error of up to ulp/2 when we round off the additional product bit produced past the ulp column. Thus, the max error in this case is about one-third of the previous value

$$\varepsilon_{\max}(r{:}3,\, c{:}1) = k'/6 + 1/4 + 2^{-k'}$$

This error is 1.587 ulp for $k' = 8$. This represents a loss that is less than the equivalent of 2 bits in precision. The average error in this case is still 0, given that both the values dropped and the rounding error are symmetric.

4 Error compensation

In the case of truncated binary multiplication, the error is uniformly negative, because the dropped bits make the result smaller. Referring to the truncated multiplication of Fig. 1 with worst-case error of $\varepsilon_{\max}$ $(r{:}2,\, c{:}0) = 7.004$ ulp, adding 4 ulp to the result, via inserting a constant ‘1’ in the column of weight 4 ulp, modifies the error to fall in the range [−3.004, 4.000], which has a 75.1% smaller worst-case magnitude. This is an example of constant error compensation [25].

In the previous example, the error compensation consisted of inserting a single bit in the partial-product matrix, thus having a negligible impact on cost and speed. The error compensation constant 3.5 ulp, or 11.1 ulp in binary, would make the final error more symmetric but it requires the insertion of 3 bits, including a bit in the position just past the least-significant bit, a position not being formed in this particular example. The point is that reduction of error comes with non-trivial costs in terms of circuit complexity and latency, which must be carefully weighed against the gain in accuracy.

A number of variable error compensation methods have also been considered in which the compensation varies according to some of the operand bits, making the compensation adaptive and the final error smaller [26–32]. Variable compensation often entails analysing average or mean-square error, rather than maximum or worst-case error. The latter approach to reducing the expected error is acceptable in some applications but may create problems in others. It is possible to base variable compensating term(s) on the worst-case error, but there is no simple procedure or algorithm, other than an exhaustive analysis, to determine the compensation term (s). It is worth noting that the area cost, latency and energy penalty for variable compensation methods increases with a reduction of the target error and tends to be greater if fully symmetric errors are desired.

In balanced ternary truncated multipliers, constant error compensation does not make sense because the error is already symmetric. Variable compensation, however, can be helpful if the compensation value is of the opposite sign of the prevailing error. Unlike the binary case, however, such a compensation scheme cannot be based on a small, fixed number of digits in the two operands. Intuitively, this is because positive digits when multiplied by negative digits produce negative digits. Thus, positive high-order digits in one operand provide no clue as to the signs of the product digits to be dropped.

Take the MSDs x_{-1} and y_{-1} of the two operands, for example. In truncated binary multiplication, the values of x_{-1} and y_{-1} are correlated with the density of 1s in the truncated portion, enabling the designer to insert x_{-1} and/or y_{-1} in appropriate columns to (partially) compensate for the dropped value. In truncated ternary multiplication, it is quite possible for the dropped value to be negative despite both x_{-1} and y_{-1} being Ps. So, neither the values of the MSDs nor the signs of the operands can be used to predict the sign of the error.

For the 4 × 4 truncated ternary multiplication of Fig. 3, in which $x_{-1} = y_{-1} = \mathrm{P}$, the total worth of the dropped bits is negative: $(-4)3^{-1} + (-1)3^{-2} + (+1)3^{-4}$ ulp = −1.432 ulp. This is only slightly less than the maximum possible negative error $-(k/2 - 1/4 + 3^{-k}/4) = -1.753$ ulp. We thus conclude that the signs of MSDs in the two operands, or the signs of the operands themselves, cannot be used to predict the direction of the error resulting from truncation. As a result, variable compensation schemes for ternary multiplication are likely to be more involved than in the binary case.

	N	Z	P
N	P	Z	N
Z	Z	Z	Z
P	N	Z	P

Fig. 2 *Truth table for balanced ternary multiplication*

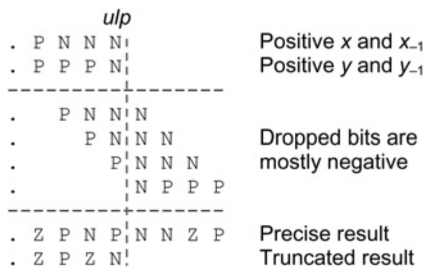

Fig. 3 *Example truncated 4 × 4 ternary multiplication*

Table 1 Maximum error values for radices 2–10

r	k''	$\varepsilon_{\max}$	$10^6\delta_{\max}$
2	16.000	15.000	228.9
3	10.095	4.798	73.2
4	8.000	10.222	156.0
5	6.891	6.641	101.3
6	6.190	10.782	164.4
7	5.699	8.299	126.7
8	5.333	11.863	181.1
9	5.047	9.844	150.3
10	4.816	13.069	199.6

5 Generalisation and discussion

Our study of truncated ternary multiplication has revealed yet another advantage of balanced ternary number representation and arithmetic: that of smaller and symmetric error in truncated multipliers. The error reduction, which is comparable with the amount provided by compensation schemes in truncated binary multipliers, is achieved at no added cost, latency or power.

One may think that the advantage arises from the higher radix and is not specific to balanced ternary representation. To show that such a supposition is false, we consider the (nearly) balanced radix-r digit set $[1 - r/2, r/2]$ if r is even and $[(1 - r)/2, (r - 1)/2]$ if r is odd. The magnitude of the product of two digits will reach the maximum value $r^2/4$ and $(r - 1)^2/4$ in the two cases, respectively. Substituting in (*), we obtain the following radix-r error in units of ulp

$$\varepsilon_{\max}(r,c{:}0)=[r^2/(r-1)^2][k''(r-1)-1+r^{-k''}]/4,\quad \text{if } r \text{ is even}$$
$$\varepsilon_{\max}(r,c{:}0)=[k''(r-1)-1+r^{-k''}]/4,\quad \text{if } r \text{ is odd}$$

Again, the use of k'' in lieu of k in radix-2 and k' in radix-3 is to remind us of the fact that the number width depends on r.

We see that the error in units of ulp increases nearly linearly with the radix-r. However, given that k'' and thus ulp $= r^{-k''}$ changes with the radix, it is more appropriate to compare the absolute errors $\delta_{\max}(r, c{:}0)$, given by

$$\delta_{\max}(r, c{:}0) = \varepsilon_{\max}(r, c{:}0)r^{-k''}$$

A final observation to allow us to numerically compare absolute errors is that the number k'' of digits in radix-r is related to the number k of digits in radix-2 by

$$k'' = k(\ln 2/\ln r) = 0.693k/\ln r$$

Putting the last four equations together, we tabulate the worst-case magnitude of the absolute error for radices 2 through 10 for $k = 16$ in Table 1. We note that the maximum error grows as the radix is increased, with odd radices always being better than the adjacent even radices. The error advantage of odd radices arises from a smaller maximum product m, as evident from the two equations for $\varepsilon_{\max}$ for even and odd radices. An earlier version of this paper also included comparisons with k'' (usually not an integer) rounded to the nearest integer. However, given that such comparisons between representations of different precisions is unfair as well as misleading (because of the more or less random rounding directions and errors for various values of k), these comparisons were dropped in the final version at the suggestion of the referees. It is hoped that a more detailed comparative study can be conducted in future to further understanding of the relative merits of various radices and associated digit sets in designing truncated multipliers.

The analysis presented so far and reflected in the results of Table 1 is admittedly quite primitive, because it takes only the maximum error into account. However, unless cost proves to be a highly non-linear function of the dynamic range, our conclusions should be safe.

We next present an approximate analysis that shows the cost and delay of a multiplier with operands in a desired dynamic range to be increasing functions of the radix-r. This fact, along with the error results of Table 1, suggests that there is no point in going beyond $r = 3$, given increases in error, cost and delay. Binary multipliers used in practice have circuit complexity on the order of k^2, or $O(k^2)$ in complexity theory notation. Asymptotically faster multiplier designs are known [18], but the constants hidden by the asymptotic notation make them impractical for run-of-the-mill multipliers. Their advantages begin to show up for operands that are hundreds of bits wide, where truncation schemes do not make sense. In going from k-bit radix-2 to k''-bit radix-r operands, the cost of forming the partial products remains virtually the same, as the reduced value of k''^2 relative to k^2 is wiped out by the increase of the complexity of digit multiplication circuitry from 1 (a single AND gate in radix-2) to $O(\log^2 r) = O(k''^2/k^2)$, where $\log r$ is the number of bits in the binary representation of a radix-r digit. The rest of the process, that is, the compression of partial-product bits, increases in complexity, except in radix-3 with a symmetric digit set, and in delay, because the product of two radix-r digits, being two digits wide, doubles the height of the matrix to be compressed [18].

Based on the foregoing discussion, the fact that balanced ternary arithmetic offers advantages over both $r = 2$ and $r > 3$ is evident from Table 1. Looking at the $\delta_{\max}$ values in the last column, which are better indicators of merit than $\varepsilon_{\max}$ (stated in units of ulp), we see a factor of $228.9/73.2 = 3.13$ improvement in error over radix-2 and a factor of $101.3/73.2 = 1.38$ improvement over the next best high-radix truncated multipliers. Even though one would not choose ternary arithmetic based solely on advantages in truncated multiplication, this additional benefit is significant and should be considered in future research on high-radix arithmetic.

6 Conclusions

A number of advantages of radix-3 arithmetic have been discovered and explored over the years, including smaller and symmetric computation errors and ease of rounding.

We have pointed out an additional advantage of balanced ternary arithmetic over other radices, including the prevalent binary representation, in the design of truncated multipliers, making it even more attractive for low-cost and/or low-energy application domains.

One complication with ternary arithmetic arises from the fact that the sign of a balanced ternary number depends on all of its digits in the worst case (leading 0s must be ignored and the sign of the first non-zero digit taken as the number's sign). However, circuits and design methods developed in connection with binary signed-digit numbers [33] can be adapted for this purpose.

We have not offered any hardware implementation or exact cost/latency modelling in this paper. One reason is that implementations can be quite varied, with various organisational choices influenced by technology (binary against multivalued logic) as well as application requirements and characteristics. We hope to be able to report on hardware implementation and modelling in the near future, beginning with gate- and transistor-level combinatorial models of the kind used by Acharyya *et al.* [34]. Over time, combined cost/latency/accuracy/energy models will be developed to allow design choices based on cost-effectiveness and other composite criteria.

Finally, our analysis uses the simplifying, pessimistic assumption that all dropped terms can have the worst-case value simultaneously. This simplifying assumption can create needlessly large error bounds. More detailed analyses and computer simulations can be used to derive sharper bounds for specific applications and/or implementation technologies [32].

7 Acknowledgment

Comments by anonymous referees in two rounds of review led to improvements in content and presentation, including the inclusion of Table 1 and several new references. The author is grateful to the referees for their contributions.

8 References

1 Wang, L.K., Erle, M.A., Tsen, C., Schwarz, E., Schulte, M.J.: 'A survey of hardware designs for decimal arithmetic', *IBM J. Res. Dev.*, 2010, **54**, (2), pp. 8:1–8:15

2 von Neumann, J.: 'First draft of a report on the EDVAC', *IEEE Ann. Hist. Comput.*, 1993, **15**, (4), pp. 27–75. [Reproduction of the original 1945 report]

3 Burks, A.W., Goldstine, H.H., von Neumann, J.: 'Preliminary discussion of the logical design of an electronic computing instrument' (Institute for Advanced Study Report, 1947, 2nd edn.)

4 Hayes, B.: 'Third base', *Am. Sci.*, 2001, **89**, (6), pp. 490–494

5 Klimenko, S.V.: 'Computer science in Russia: a personal view', *IEEE Ann. Hist. Comput.*, 1999, **29**, (3), pp. 16–30

6 Frieder, G.: 'Ternary computers – part 1: motivation & part 2: emulation'. Proc. Fifth Workshop Microprogramming, 1972, pp. 83–89

7 Halpern, I., Yoeli, M.: 'Ternary arithmetic unit', *Proc. IEE*, 1968, **115**, (10), pp. 1385–1388

8 Eichmann, G., Li, Y., Alfano, R.R.: 'Optical binary coded ternary arithmetic and logic', *Appl. Opt.*, 1986, **25**, (18), pp. 3113–3121

9 Stakhov, A.: 'Brousentsov's ternary principle, Bergman's number system and ternary mirror-symmetrical arithmetic', *Comput. J.*, 2002, **45**, pp. 221–236

10 Gundersen, H., Berg, Y.: 'A novel balanced ternary adder using recharged semi-floating gate devices'. Proc. IEEE Int. Symp. Multivalued Logic, 2006, pp. 18–21

11 Gundersen, H., Berg, Y.: 'A balanced ternary multiplication circuit using recharged semi-floating gate devices'. Proc. 24th Norchip Conf., 2006, pp. 205–208

12 Leslie, J.: 'The philosophy of arithmetic: exhibiting a progressive view of the theory and practice of calculation' (1820, 2nd edn.) William and Charles Tait

13 Glusker, M., Hogan, D.M., Vass, P.: 'The ternary calculating machine of Thomas Fowler', *Ann. Hist. Comput.*, 2005, **27**, (3), pp. 4–22

14 Parhami, B., McKeown, M.: 'Arithmetic with binary-encoded balanced ternary numbers'. Proc. 47th Asilomar Conf. Signals, Systems, and Computers, 2013, pp. 1130–1133

15 Balla, P.C., Antoniou, A.: 'Low power dissipation MOS ternary logic family', *IEEE J. Solid-State Circuits*, 1984, **19**, (5), pp. 739–749

16 Wu, W.W., Prosser, F.P.: 'CMOS ternary logic circuits', *IEE Proc. Circuits Devices Syst.*, 1990, **137**, (1), pp. 21–27

17 Lin, S., Kim, Y.-B., Lombardi, F.: 'CNTFET-based design of ternary logic gates and arithmetic circuits', *IEEE Trans. Nanotechnol.*, 2011, **10**, (2), pp. 217–225

18 Parhami, B.: 'Computer arithmetic: algorithms and hardware designs' (Oxford, 2010, 2nd edn.)

19 Swartzlander, E.E.: 'Truncated multiplication with approximate rounding'. Proc. 33rd Asilomar Conf. Signals, Systems, and Computers, 1999, pp. 1480–1483

20 Schulte, M.J., Stine, J.E., Jansen, J.G.: 'Reduced power dissipation through truncated multiplication'. Proc. IEEE Workshop Low-Power Design, 1999, pp. 61–69

21 Kidambi, S.S., El-Guibaly, F., Antoniou, A.: 'Area-efficient multipliers for digital signal processing applications', *IEEE Trans. Circuits Syst. II*, 1996, **43**, (2), pp. 90–95

22 Jou, J.M., Kuang, S.R., Chen, R.D.: 'Design of low-error fixed-width multipliers for DSP applications', *IEEE Trans. Circuits Syst. II*, 1999, **46**, (6), pp. 836–842

23 Van, L.-D., Wang, S.S., Feng, W.-S.: 'Design of the lower error fixed-width multiplier and its application', *IEEE Trans. Circuits Syst. II*, 2000, **47**, (10), pp. 1112–1118

24 Jaberipur, G., Parhami, B.: 'Efficient realisation of arithmetic algorithms with weighted collections of posibits and negabits', *IET Comput. Digit. Tech.*, 2012, **6**, (5), pp. 259–268

25 Schulte, M.J., Swartzlander, E.E.: 'Truncated multiplication with correction constant'. VLSI Signal Processing VI, 1993, pp. 388–396

26 Wires, K.E., Schulte, M.J., Stine, J.E.: 'Variable-correction truncated floating point multipliers'. Proc. 34th Asilomar Conf. Signals, Systems, and Computers, 2000, vol. 2, pp. 1344–1348

27 Strollo, A.G.M., Petra, N., De Caro, D.: 'Dual-tree error compensation for high performance fixed-width multipliers', *IEEE Trans. Circuits Syst. II*, 2005, **52**, (8), pp. 501–507

28 Van, L.-D., Yang, C.-C.: 'Generalized low-error area-efficient fixed-width multiplies', *IEEE Trans. Circuits Syst. I*, 2005, **52**, (8), pp. 1608–1619

29 Kuang, S.R., Wang, J.P.: 'Low-error configurable truncated multipliers for multiply-accumulate applications', *Electron. Lett.*, 2006, **42**, (16), pp. 904–905

30 Petra, N., De Caro, D., Garofalo, V., Napoli, E., Strollo, A.G.M.: 'Truncated binary multipliers with variable correction and minimum mean square error', *IEEE Trans. Circuits Syst. I*, 2010, **57**, (6), pp. 1312–1325

31 Petra, N., De Caro, D., Garofalo, V., Napoli, E., Strollo, A.G.M.: 'Design of fixed-width multipliers with linear compensation function', *IEEE Trans. Circuits Syst. I*, 2011, **58**, (5), pp. 947–960

32 De Caro, D., Petra, N., Strollo, A.G.M., Tessitore, F., Napoli, E.: 'Fixed-width multipliers and multipliers-accumulators with min-max approximation error', *IEEE Trans. Circuits Syst. I*, 2013, **60**, (9), pp. 2375–2388

33 Srikanthan, T., Lam, S.K., Suman, M.: 'Area-time efficient sign detection technique for binary signed-digit number system', *IEEE Trans. Comput.*, 2004, **53**, (1), pp. 69–72

34 Acharyya, A., Maharatna, K., Al-Hashimi, B.: 'Algorithm and architecture for N-D vector cross product computation', *IEEE Trans. Signal Process.*, 2011, **59**, (2), pp. 812–826