Design of efficient modulo $2^n + 1$ multipliers

H.T. Vergos and C. Efstathiou

Abstract: A new modulo $2^n + 1$ multiplier architecture is proposed for operands in the weighted representation. A new set of partial products is derived and it is shown that all required correction factors can be merged into a single constant one. It is also proposed that part of the correction factor is treated as a partial product, whereas the rest is handled by the final parallel adder. The proposed multipliers utilise a total of $(n + 1)$ partial products, each n bits wide and are built using an inverted end-around-carry, carry-save adder tree and a final adder. Area and delay qualitative and quantitative comparisons indicate that the proposed multipliers compare favourably with the earlier solutions.

1 Introduction

Residue arithmetic has been used widely in several applications of computer systems. Such applications include the design of generic or specialised digital signal processors (DSPs) that adopt a residue number system (RNS) [1–6], number theoretic transforms that are widely used in convolution/correlation computations [7–10], the implementation of cryptographic algorithms [11] and fault-tolerant digital system design. The adoption of an RNS has been proved very useful in enhancing the performance of modulation components used in communications [12] and in digital filter design [6], in which it can help reduce the power consumption when the number of the taps used in the filter increases [13].

Modulo $2^n + 1$ arithmetic, in particular, has attracted special attention, mainly as a part of the well-known three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, which has been extensively used in general- and special-purpose RNS implementations. In an RNS that uses the three-moduli set $\{2^n - 1, 2^n, 2^n + 1\}$, the modulo $2^n + 1$ becomes a bottleneck, since it has to deal with $(n + 1)$-bit operands, whereas the rest two channels operate on n-bit ones. To overcome this problem, Leibowitz [7] introduced the diminished-1 representation. Under this representation, each number is decreased by 1. Arithmetic operations are in this way performed on n-bit operands, whereas the result is derived in an alternative manner when one or both operands are zero.

The prime moduli of the form $2^n + 1$ apart from being useful for ordinary RNSs, are vital in the Fermat number transform (FNT) and useful in cryptography. The Fermat number $2^{16} + 1$, in particular, being the only Fermat number of practical interest, was chosen for the modulo multiplier in the works of Benaissa *et al.* [9] and Sunder *et al.* [10] and for the square-and-multiply modulo exponentiator in the implementation of the international data encryption algorithm [11].

Due to these applications of modulo $2^n + 1$ arithmetic, numerous algorithms and architectures have been proposed for modulo $2^n + 1$ components including adders [14–16], multi-operand adders and residue generators [17], squarers [18] and multipliers [14, 19–26].

Although a modulo $2^n + 1$ product can be computed by look-up tables implemented in ROM, the exponential growth of the memory required makes such solutions unsuitable for medium or large values of n. Several solutions have therefore emerged that rely on arithmetic combinational blocks. Table 1 summarises their architectural characteristics.

In the work of Hiasat [19], it was proposed that multiplication is carried out with arithmetic blocks. An $(n + 1)$-bit binary multiplier was used, along with a residue generation circuit. A major advance in the design of modulo $2^n + 1$ multipliers was achieved by Wrzyszcz and Milford [20]. The authors observed that the multiplication array required can be reduced to $n \times n$, since several groups of partial-product bits cannot be simultaneously at 1. The architecture of Wrzyszcz and Milford [20], however, suffers from the use of three n-bit parallel adders connected in series and a final row of multiplexors. In the work of Wang *et al.* [21], diminished-1 multipliers with n-bit input operands were considered. Apart from the multiplication array, a zero partial-product counting circuit is required. Handling of zero operands and results was not considered. The first attempt to apply the radix-4 Booth recoding to modulo $2^n + 1$ multiplication appeared in the work of Ma [22]. The partial products used are $(n + 1)$ bits wide; furthermore, one carry save adder (CSA) stage is required for result correction, along with a diminished-1 final parallel adder with a carry input. Such an adder is implemented by a further CSA stage and a diminished-1 parallel adder. The multipliers presented in the work of Zimmerman [14] also use Booth encoding and n bits for their operands' representation but depart from the diminished-1 discipline, since, all operands are in weighted form, except the 2^n operand which is represented by the all 0s vector. This is done for achieving an efficient design block for use in the international data encryption algorithm (IDEA) block cipher. Although in the work of Zimmerman [14] a scheme is described for using the multipliers for weighted/diminished-1 operands, this scheme implies a dedicated circuit that handles the 2^n value in the case of weighted operands or the incorporation of a second parallel modulo adder in the case of diminished-1 operands. Multipliers in which one operand uses weighted representation, whereas the other uses the

doi:10.1049/iet-cdt:20060026

Paper first received 13th February and in revised form 20th June 2006

H.T. Vergos is with the Computer Engineering and Informatics Department, University of Patras, 26500 Patras, Greece

C. Efstathiou is with the Informatics Department, TEI of Athens, 12210 Egaleo, Athens, Greece

E-mail: vergos@ceid.upatras.gr

Table 1: Architectural characteristics of the current and previous proposals

	Hiasat [19]	Wrzyszcz and Milford [20]	Wang *et al.* [21]	Ma [22]	Zimmerman [14]
Operand representation	weighted	weighted	diminished	diminished	proprietary
Number of partial products	$n+1$	n	n	$\lceil n/2 \rceil + 2$	$\lceil n/2 \rceil$
Width of partial partial products	$n+1$	n	n	$n+1$	n
Multiplier encoding	not discussed	none	none	Booth	Booth
Number of parallel adders	1	3	1[a]	1[a]	1
Other circuits required	Residue generator	Multiplexer row	Counter of zero partial products	CSA stage	Extra circuits for diminished-1 adaptation
Handling of zero operands/results	yes	yes	no	no	proprietary

	Curiger [23]	Efsthiou *et al.* [24]	Sausa and Chaves [25]	Shaves and Sousa [26]	This proposal
Operand representation	proprietary	diminished	diminished/ weighted	weighted	weighted
Number of partial products	$\lceil n/2 \rceil$	n	$\lceil n/2 \rceil + 2$	$n+1$ or $n+2$	n
Width of partial partial products	n	n	n	n	n
Multiplier encoding	Booth	none	Booth	none	none
Number of parallel adders	1	1	1[a]	1	1
Other circuits required	None	None	CSA stage, combinational circuit for correction computation	none	none
Handling of zero operands/results	proprietary	no	yes	yes	yes

[a]with specific design

diminished-1 were investigated in the work of Curiger [23]. These multipliers are however specific to the cryptographic application targeted.

In the work of Efstathiou *et al.* [24], diminished-1 multipliers have been proposed that use an $n \times n$ partial-product array along with a CSA tree. These multipliers were analytically and experimentally shown to outperform those of Wang *et al.* [21] and Ma [22] in terms of delay and power. However, treatment of zero operands or results was not discussed. Modulo $2^n + 1$ multipliers for both diminished-1 and weighted operand representation with treatment of zero operands have been presented in the work of Sousa and Chaves [25]. For both cases, radix-4 Booth recoding is employed to reduce the number of partial products into approximately half. Both architectures, however, require a CSA stage for the addition of a correction factor that is derived by a small combinational circuit, as well as a final modulo parallel adder similar to that of Wang *et al.* [21], that is with a carry input. Furthermore, in the case of diminished-1 operands, another correction is also introduced. The analytical and experimental results indicated in the work of Sousa and Chaves [25] show that the multipliers proposed outperform the earlier solutions of Ma [22] and Zimmerman [14] if the latter is adopted to diminished-1 or ordinary representation. Comparative results against the multipliers proposed by Efstathiou *et al.* [24] were not however given. Taking into account the analytical models for the delays and areas presented in the works of Efstathiou *et al.* [24] and Sousa and Chaves [25], we conclude that the architecture of Efstathiou *et al.* [24] provides smaller multipliers. It also leads to faster designs when $n < 16$ and to designs with the same delay as those described in the work of Sousa and Chaves [25] for $n \geq 16$. This should not be surprising, since it is well known [14] that although Booth recoding leads to a shallower adder tree (about two full adder (FA) stages are saved on the critical path when the number of partial products is cut in half), this saving may be compensated or

overwhelmed by the delay of the recoding logic. Non-Booth encoded modulo $2^n + 1$ multipliers have recently been investigated in the work of Chaves and Sousa [26]. Two architectures have been proposed; one with $n + 3$ partial products and one with $n + 2$ partial products.

In this manuscript, we propose a novel architecture, for weighted representation of the input operands, which utilises the observations made in the work of Wrzyszcz and Milford [20]. The main improvements of the proposed architecture lie in:

- using different partial products than those used in the work of Wrzyszcz and Milford [20]. The new partial products can be computed faster.
- using only one total correction factor. This correction factor is derived analytically and is shown to be a constant. Therefore no extra circuit is required to compute it and
- splitting this correction factor in two parts. One part is introduced as a partial product, whereas the addition of the second part is assigned to the final-stage adder. This enables us to use a fast parallel-prefix inverted end-around-carry (EAC) parallel adder [15] (equivalently, a diminished-1 modulo $2^n + 1$ adder) as the final-stage adder.

The proposed architecture does not use Booth recoding and utilises a total of $(n + 1)$ n-bit wide partial products. The resulting multipliers obviously outperform the solutions described in the work of Chaves and Sousa [26] in both area and delay, since they use one or two partial products less. We therefore compare the proposed multipliers with those of Efstathiou *et al.* [24], which according to the earlier discussion are currently the most efficient ones. Both analytical and experimental comparison results are presented. Our results indicate that the proposed multipliers offer the same or higher operation speed as those of Efstathiou *et al.* [24], while in parallel being more compact. Considering however, that the proposed multipliers accept operands in weighted form whereas those presented in the work of Efstathiou *et al.* [24] accept operands in diminished-1 representation, it is clear that the proposed multipliers can be used more efficiently, since they do not require time- and hardware-consuming input/output translators, nor any further circuit for handling zero operands and results.

2 Proposed architecture

The proposed multiplier architecture is based on merging the correction factors that result from the formation and the reduction of the new partial-products into a single correction factor. This is described in detail in the following subsections.

2.1 Partial-product formation

Let $A = a_n a_{n-1} \cdots a_1 a_0$ and $B = b_n b_{n-1} \cdots b_1 b_0$ denote two $(n + 1)$-bit numbers in the range $[0, 2^n + 1)$. Obviously, in the weighted representation, if a_n is 1 then all the remaining bits in the representation of A are 0, and the same is true for the remaining bits of B if b_n is 1. This observation enabled Wrzyszcz and Milford [20] to reduce the multiplication array from an $(n + 1) \times (n + 1)$-bit size down to $n \times n$. We follow a similar procedure to derive our new partial products.

Let $|X|_Y$ denote the modulo Y residue of X. For the multiplication of A with B, we then have

$$\mathrm{R} = |A \times B|_{2^n+1} = \left| \sum_{i=0}^{n} a_i 2^i \sum_{j=0}^{n} b_j 2^j \right|_{2^n+1}$$

$$= \left| \sum_{i=0}^{n} \left(\sum_{j=0}^{n} p_{i,j} 2^{i+j} \right) \right|_{2^n+1} \quad (1)$$

where the partial-product bit $p_{i,j}$ is the logical AND of a_i with b_j, that is $p_{i,j} = a_i \wedge b_j$. Relation (1) indicates that the partial-product matrix shown in Fig. 1 is required for modulo $2^n + 1$ multiplication. The partial-product matrix can be divided into four groups A, B, C and D shown in Fig. 1. Note that only terms from one group can be different from 0 at the same time. Therefore instead of arithmetically adding, terms from different groups can be logically ORed.

We first perform the logical OR (hereafter denoted by $\vee$) of corresponding terms of groups B, D and A in the columns with weight 2^n up to 2^{2n-2} and the logical OR of the two terms of groups B and D with weight 2^{2n-1}. Since $|2^{2n-1}|_{2^n+1} = 2^{n-1} + 1$, the latter term, $q_{n-1} = p_{n,n-1} \vee p_{n-1,n}$ can be substituted by two terms q_{n-1} in the columns with weight 2^{n-1} and 1, respectively, and ORed with any term of group A there. Moreover, since $|2^{2n}|_{2^n+1} = 1$, the term $p_{n,n}$ can be repositioned in the rightmost column and ORed with $p_{0,0}$. After these modifications, our partial-product matrix has the form shown in Fig. 2, in which q_i denotes $p_{n,i} \vee p_{i,n}$.

Then, for obtaining an $n \times n$ partial-product array, every term, suppose s, that appears to the left of the column with weight 2^{n-1}, needs to be repositioned. Considering that

$$|s2^i|_{2^n+1} = \left|-s2^{|i|_n}\right|_{2^n+1} = \left|(2^n + 1 - s)2^{|i|_n}\right|_{2^n+1}$$

$$= \left|\bar{s}2^{|i|_n} + 2^n 2^{|i|_n}\right|_{2^n+1}$$

we can invert and reposition every s term of column i, with $n \le i \le 2n - 2$, to the $i - n$th column, taking into account a correction factor of $2^{|i|_n} 2^n$, for each such complementation and repositioning. For computing the correction factor, COR_1, required for moving all s terms, we can observe that there is only one such term in the second partial

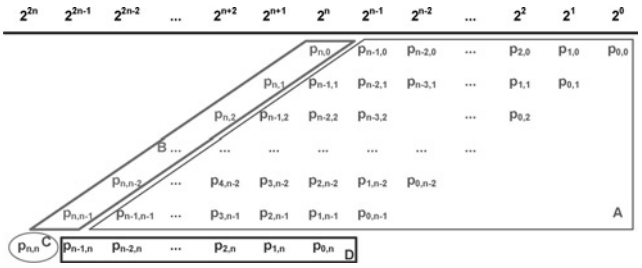

Fig. 1 *Initial partial-product matrix*

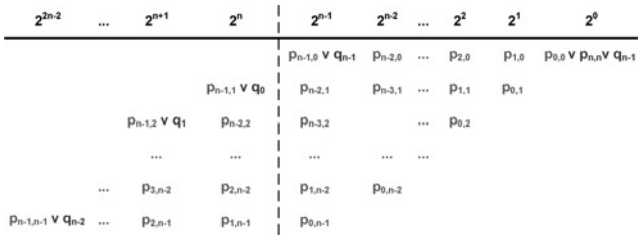

Fig. 2 *Modified partial-product matrix*

product in Fig. 2, imposing a correction of $2^0 2^n$, whereas the third has two such terms imposing a correction of $(2^0 + 2^1)2^n = (2^3 - 1)2^n$ and so on, up to the nth partial-product which will require a correction factor equal to $(2^0 + 1^1 + \cdots + 2^{n-2})2^n = (2^{n-1} - 1)2^n$. Summing all these required corrections, we get that COR_1 is given by

$$\begin{aligned}\mathrm{COR}_1 &= 2^n(2(1 + 2 + 2^2 + \cdots + 2^{n-2}) - (n-1)) \\ &= 2^n(2^n - n - 1)\end{aligned} \tag{2}$$

In this way, the reduced $(n \times n)$ matrix of partial products (PP_i) presented in Table 2 is derived, along with the correction factor indicated by (2), which needs to be taken into account. Although the above partial products were derived following the observations of Wrzyszcz and Milford [20], they are somewhat different. Even the more complex of them can be derived by AND–OR-invert gates, that can be very efficiently implemented in CMOS technology. On the contrary, the more complex partial products of Wrzyszcz and Milford [20] require OR-AND-XOR gates that are more area and time consuming.

2.2 Partial-product reduction

The n partial products in Table 2 and the correction factor in (2), that is $n + 1$ partial products in total, must be added modulo $2^n + 1$, until two final summands are produced. This can be performed by using either an $(n - 1)$-stage CSA array or a CSA tree (Wallace [27] or Dadda [28] trees). It is well known that in integer multipliers, a CSA tree results to irregular architectures. However, in our case, the resulting array is completely regular, that is well-suited for VLSI implementations. This is because the same number of bits appears in every column of the array and as we will show, the carry outputs at the most significant bit position of each stage will be used as carry inputs of the subsequent stage.

Suppose that the carry output of the nth column of stage i is denoted by c_i. This signal has a weight of 2^n. Since

$$|c_i 2^n|_{2^n+1} = |-c_i|_{2^n+1} = |2^n + \overline{c_i}|_{2^n+1}$$

the carries out of the most significant bit position can be complemented and added to the least significant bit position of the next stage, forming an inverted EAC CSA tree. A correction factor of 2^n must be taken into account for each such carry recirculation. During the addition of our $n + 1$ partial products, $n - 1$ carries of weight 2^n will be generated and therefore the correction, COR_2, that would be required for the inverted EACs is

$$\mathrm{COR}_2 = |2^n(n-1)|_{2^n+1} \tag{3}$$

The total correction required is consequently given by the addition of the factors derived in (2) and (3)

$$\begin{aligned}\mathrm{COR} &= |2^n(2^n - n - 1) + 2^n(n-1)|_{2^n+1} \\ &= |2^n(2^n - 2)|_{2^n+1} = 3\end{aligned}$$

Therefore (1) takes the form

$$\mathrm{R} = |A \times B|_{2^n+1} = \left|\sum_{i=0}^{n-1} \mathrm{PP}_i + 3\right|_{2^n+1} \tag{4}$$

where the n partial products have been presented in Table 2.

2.3 Final-stage addition

A straightforward implementation for the multipliers, which stems from (4), is to use COR as an extra partial product, along with a fast modulo $2^n + 1$ adder (for example [16]) which accepts the two summands produced by the reduction scheme (array/tree) and produces the product. In this manuscript, however, we propose an alternative, more effective solution. The solution that we propose is based on several observations.

The first observation is that the fastest inverted EAC parallel adder known [15] (equivalently, the fastest diminished-1 modulo $2^n + 1$ adder) provides three gate equivalents less delay than the fastest modulo $2^n + 1$ adder available [16], while in parallel leading to more compact designs. Therefore it would be beneficial if instead of a modulo $2^n + 1$ adder, an inverted EAC parallel adder could be used. It should be noted, however, that since such an adder can provide only n bits of the result, the remaining bit must be derived in an alternative manner.

The second observation is that when two n-bit operands suppose S and C that follow the weighted representation are used as inputs to a diminished-1 modulo $2^n + 1$ adder, its output will be equal to $|\mathrm{S} + \mathrm{C} + 1|_{2^n+1}$. This observation leads us to the conclusion that for using an inverted EAC parallel adder, part of the correction factor should be assigned to the final adder, whereas the rest is treated as a partial product. We therefore rewrite (4) as

$$\begin{aligned}\mathrm{R} = |A \times B|_{2^n+1} &= \left|\sum_{i=0}^{n-1} \mathrm{PP}_i + 3\right|_{2^n+1} \\ &= \left|\left|\sum_{i=0}^{n-1} \mathrm{PP}_i + 2\right|_{2^n+1} + 1\right|_{2^n+1}\end{aligned} \tag{5}$$

Let $\mathrm{S} = s_{n-1}s_{n-2}\cdots s_0$ and $\mathrm{C} = c_{n-2}c_{n-1}\cdots c_0\overline{c_{n-1}}$ denote the sum and carry n-bit vectors that are produced by the multi-operand addition $|\sum_{i=0}^{n-1} \mathrm{PP}_i + 2|_{2^n+1}$, that is $\mathrm{S} + \mathrm{C} = |\sum_{i=0}^{n-1} \mathrm{PP}_i + 2|_{2^n+1}$. Substituting this in (5), results in

$$\mathrm{R} = |A \times B|_{2^n+1} = |\mathrm{S} + \mathrm{C} + 1|_{2^n+1} \tag{6}$$

We can now observe that the most significant bit of the multiplication, is 1 only when $\mathrm{R} = 2^n$. From (6) we get that $\mathrm{R} = 2^n <=> |\mathrm{S} + \mathrm{C} + 1|_{2^n+1} = 2^n$. Taking into account that S and C are n-bit vectors, we then get that $\mathrm{R} = 2^n <=> \mathrm{S} + \mathrm{C} + 1 = 2^n$ or equivalently that $\mathrm{S} + \mathrm{C} = 2^n - 1$. That is the most significant bit of the

Table 2: Reduced partial-product matrix

2^{n-1}	2^{n-2}	2^{n-1}	...	2^2	2^1	2^0
$\mathrm{PP}_0 = p_{n-1,0} \vee q_{n-1}$	$p_{n-2,0}$	$p_{n-3,0}$	...	$p_{2,0}$	$p_{1,0}$	$p_{0,0} \vee q_{n-1} \vee p_{n,n}$
$\mathrm{PP}_1 = p_{n-2,1}$	$p_{n-3,1}$	$p_{n-4,1}$	...	$p_{1,1}$	$p_{0,1}$	$\overline{p_{n-1,1} \vee q_0}$
$\mathrm{PP}_2 = p_{n-3,2}$	$p_{n-4,2}$	$p_{n-5,2}$	...	$p_{0,2}$	$\overline{p_{n-1,2} \vee q_1}$	$\overline{p_{n-2,2}}$
...	...	...	...	...	...	...
$\mathrm{PP}_{n-2} = p_{1,n-2}$	$p_{0,n-2}$	$\overline{p_{n-1,n-2} \vee q_{n-3}}$	...	$\overline{p_{4,n-2}}$	$\overline{p_{3,n-2}}$	$\overline{p_{2,n-2}}$
$\mathrm{PP}_{n-1} = p_{0,n-1}$	$\overline{p_{n-1,n-1} \vee q_{n-2}}$	$\overline{p_{n-2,n-1}}$	...	$\overline{p_{3,n-1}}$	$\overline{p_{2,n-1}}$	$\overline{p_{1,n-1}}$

multiplication is 1 only when S and C are complementary vectors. As explained later, this observation enables us to compute the most significant bit distinctly from the rest. In the following, we focus on the n least significant bits of R.

Let R_n denote the n-bit vector of the least significant bits of R. We then have that

$$\begin{aligned} R_n &= ||A \times B|_{2^n+1}|_{2^n} = ||S + C + 1|_{2^n+1}|_{2^n} \\ &= \begin{cases} |S + C + 1 - (2^n + 1)|_{2^n}, & \text{if } S + C + 1 \geq 2^n + 1 \\ |S + C + 1|_{2^n}, & \text{otherwise} \end{cases} \\ &= \begin{cases} |S + C - 2^n|_{2^n}, & \text{if } S + C \geq 2^n \\ |S + C + 1|_{2^n}, & \text{otherwise} \end{cases} \\ &= \begin{cases} |S + C|_{2^n}, & \text{if } S + C \geq 2^n \\ ||S + C|_{2^n} + 1|_{2^n}, & \text{otherwise} \end{cases} \end{aligned} \quad (7)$$

The latter relation reveals that the n least significant bits of the product can be handled by an n-bit adder that increases the binary sum of its inputs by one when the carry output is 0 and leaves it unchanged in the case of a carry output. This is exactly the function performed by an inverted EAC parallel adder. We therefore conclude that, if a total correction factor of 2 is used as an extra partial product, an inverted EAC parallel adder used as the final adder will accept S and C at its inputs and will provide R_n.

Very fast inverted EAC adders based on parallel prefix carry computation units have appeared in the works of Zimmerman [14] and Vergos *et al.* [15]. For integer adders, a parallel prefix carry computation unit is derived from the following. Let $A = a_{n-1}a_{n-2}\cdots a_1a_0$ and $B = b_{n-1}b_{n-2}\cdots b_1b_0$ denote the two n-bit addition operands and let the terms $g_i = a_i \wedge b_i$ and $p_i = a_i \oplus b_i$ denote the carry generate and propagate terms at bit position i, respectively. By defining $\circ$ as an operator that associates generate and propagate pairs and produces a new pair according to the equation

$$(g_x, p_x) \circ (g_y, p_y) = (g_x \vee p_x \wedge g_x, p_x \wedge p_y)$$

the computation of a carry c_i of the integer addition of A and B is equivalent to computing G_i under the prefix equation

$$(G_i, P_i) = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \cdots \circ (g_1, p_1) \circ (g_0, p_0)$$

Once the carries have been computed, the sum bits, s_i, are computed by $s_i = p_i \oplus c_{i-1}$.

For attaining an inverted EAC adder, simple solutions, such as the connection of the carry output of an integer adder back to the carry input via an inverter, are not well-suited, since they suffer from oscillations. In the work of Zimmerman [14], it was proposed that the output carry is driven back via an inverter to a late carry increment stage composed of nodes implementing a prefix operator. Therefore no oscillations occur and if the carry computation unit is designed according to the fast algorithms presented in the works of Kogge and Stone [29] and Ladner and Fisher [30], the derived inverted EAC adders feature an operating speed close to the corresponding integer adders.

The need for an extra prefix stage that handles the re-entering carry has been cancelled in the parallel-prefix inverted EAC adders proposed in the work of Vergos *et al.* [15]. This was achieved by performing carry re-circulation at each existing prefix level. As a result, parallel-prefix adder architectures with $\log_2 n$ have been derived, that is inverted EAC adders that can achieve the same operating speed as the corresponding integer adders. For the sake of completeness, we revisit some of the theories developed in the work of Vergos *et al.* [15] in the following.

The inverted EAC adder carry, suppose c_i^* is equal to G_i^*, where G_i^* is computed according to the parallel prefix equations

$$(G_i^*, P_i^*) = \begin{cases} \overline{(G_{n-1}, P_{n-1})}, \\ \quad \text{if } i = -1 \\ (G_i, P_i) \circ \overline{(G_{n-1,i+1}, P_{n-1,i+1})}, \\ \quad \text{if } 0 \leq i \leq (n-2) \end{cases} \quad (8)$$

and

- $\overline{(G, P)} = (\bar{G}, P)$
- $G_{a,b}$ and $P_{a,b}$, with $a > b$, are, respectively, the group generate and propagate signals for the group a, $a-1$, $a-2$, $\ldots$, $b+1$, b, computed by $(G_{a,b}, P_{a,b}) = (g_a, p_a) \circ (g_{a-1}, p_{a-1}) \circ \cdots \circ (g_b, p_b)$.

In the cases that the equations indicated by (8) require more than $\log_2 n$ prefix levels for their implementation, we can transform them into equivalent ones by introducing t_i, $t_i = a_i \vee b_i$, and taking into account that if $(G^x, P^x) = (g, p) \circ \overline{(G, P)}$ and $(G^y, P^y) = (\overline{(t, \bar{g})} \circ (G, P))$ then $G^x = G^y$ [15]. This enables us to equivalently compute a carry whose equation is given by a prefix equation of the form $(g, p) \circ \overline{(G, P)}$ as $(\overline{(t, \bar{g})} \circ (G, P))$. For area–time efficient designs, this transformation should be applied j times recursively to the equations of the form $(g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \cdots \circ (g_1, p_1) \circ \overline{(G_{n,i+1}, P_{n,i+1})}$ given by (8), until

$$n - 1 - i + j = \begin{cases} n, & \text{if } i > \dfrac{n}{2} - 1 \\ \dfrac{n}{2}, & \text{if } i \leq \dfrac{n}{2} - 1 \end{cases}$$

Since S and C are the inputs of the final adder and $P_{n-1} = p_{n-1} \wedge p_{n-2} \wedge \cdots \wedge p_0$, we conclude that the most significant bit of the result which should be 1 only in the case that S and C are complementary vectors is equal to the group propagate signal out of the n bits of the final inverted EAC adder.

2.4 An example of the proposed architecture

Table 3 lists the required partial products in a modulo 17 multiplier that follows the proposed architecture. The last partial-product represents the total correction factor when an inverted EAC adder is used as the final adder.

Fig. 3 presents a block diagram of the proposed modulo 17 multiplier. The simple gates required for the formation of the partial-product bits are not shown. The blocks used are half-adders (HA), FAs and simplified FA blocks (FA^+), that is FAs with one of their inputs set at 1 and the final adder. The output carries at the most significant bit of each stage are complemented and driven to the least significant bit of the subsequent stage. The two final derived summands are added in the final parallel adder.

It should be noted that the partial-product bits of equal weight are not driven randomly in the FAs of the corresponding column. For achieving the least delay, the partial-product bits derived earlier should be driven to the FAs at the top levels of the CSA tree, whereas late arriving signals to FAs of subsequent tree levels. For example the FAs of the rightmost column in Fig. 3, perform the addition of 0, $\overline{p_{1,3}}$, $\overline{p_{2,2}}$, $\overline{p_{3,1} \vee q_0}$ and $p_{4,4} \vee p_{0,0} \vee q_3$ along with the inverted carries that overflow at the leftmost column. The

Table 3: Partial-product matrix in modulo 17 multiplication

2^3	2^2	2^1	2^0
$PP_0 = p_{3,0} \vee p_{3,4} \vee p_{4,3}$	$p_{2,0}$	$p_{1,0}$	$p_{0,0} \vee p_{4,4} \vee p_{4,3} \vee p_{3,4}$
$PP_1 = p_{2,1}$	$p_{1,1}$	$p_{0,1}$	$p_{3,1} \vee p_{0,4} \vee p_{4,0}$
$PP_2 = p_{1,2}$	$p_{0,2}$	$\overline{p_{3,2} \vee p_{1,4} \vee p_{4,1}}$	$\overline{p_{2,2}}$
$PP_3 = p_{0,3}$	$\overline{p_{3,3} \vee p_{2,4} \vee p_{4,2}}$	$\overline{p_{2,3}}$	$\overline{p_{1,3}}$
$PP_4 = 0$	0	1	0

addition of 0 cannot be avoided, since doing so alters the number of inverted EACs in the CSA tree and invalidates all the previous analysis. However, the FAs accepting the bits from the constant partial product can be simplified to HA or FA^+. Since it is expected that the signals $\overline{p_{3,1}}$ and $\overline{p_{2,2}}$ would be the ones computed earlier, these along with the 0 operand should be examined as candidates for the first addition stage.

The final adder required in this case is shown in Fig. 4. The parallel-prefix carry computation unit of the adder computes the carries according to the following equations

$$c_{-1}^* = \overline{(g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1) \circ (g_0, p_0)}$$

$$c_0^* = \overline{(\overline{t_0}, \overline{g_0}) \circ (g_3, p_3) \circ (g_2, p_2) \circ (g_1, p_1)}$$

$$c_1^* = (g_1, p_1) \circ (g_0, p_0) \circ \overline{((g_3, p_3) \circ (g_2, p_2))}$$

$$c_2^* = (g_2, p_2) \circ (g_1, p_1) \circ \overline{((\overline{t_0}, \overline{g_0}) \circ (g_3, p_3))}$$

From Fig. 3, it is evident that the proposed multipliers have a regular structure that is well suited for VLSI implementation.

3 Area – delay analysis and comparisons

In this section, we analyse the area and time complexity of the proposed multipliers. We also compare them with the multipliers proposed in the work of Efstathiou *et al.* [24], which according to the discussion in the introductory section are the most efficient in both area and delay terms.

For our qualitative comparisons, we use the unit-gate model proposed in the work of Tyagi [31]. This model considers that all 2-input monotonic gates count as one gate-equivalent for both area and delay, whereas a 2-input XOR/XNOR gate counts as two equivalents.

The area requirements of the proposed multipliers consist of the partial-product-bit formation gates, the CSA tree and the final adder. Considering that $(n+1)^2$ AND (or NAND) gates are required for the formation of the $p_{i,j}$ (or $\overline{p_{i,j}}$) terms, $2(n-1)$ OR (or NOR) gates are required for the $\overline{p_{i,j} \vee q_i}$ terms, and four more OR gates are required for the two more complex product bits in PP_0, we can compute that the area required for the formation of partial-product bits is

$$A_{\text{partial}} = (n+1)^2 + 2(n-1) + 4 = n^2 + 4n + 3 \qquad (9)$$

gate equivalents. The CSA tree that performs the reduction of the partial products in two final summands is composed of $(n-1)$ rows of n FAs each. However, since one of these rows accepts the constant correction factor, its n FAs can be simplified to $n-1$ HAs and one FA^+. Considering that an FA, an HA and an FA^+ have an area of 7, 3 and 3 gate equivalents, respectively, we get that the area required for the partial-product reduction is

$$A_{\text{CSA}} = 7n(n-2) + 3(n-1) + 3 = 7n^2 - 11n \qquad (10)$$

gate equivalents. The area of the last stage adder was computed in the work of Vergos *et al.* [15] as

$$A_{\text{adder}} = \frac{9}{2} n \lceil \log n \rceil + \frac{1}{2} n + 6 \qquad (11)$$

Summing the area requirements of (9)–(11), we can get that the proposed multipliers have an area of

$$A_{\text{proposed}} = 8n^2 + \frac{9}{2} n \lceil \log n \rceil - \frac{13}{2} n + 9$$

equivalent gates, whereas the area of the multipliers proposed in the work of Efstathiou *et al.* [24] was

$$A[24] = 8n^2 + \frac{9}{2} n \lceil \log n \rceil + \frac{1}{2} n + 4$$

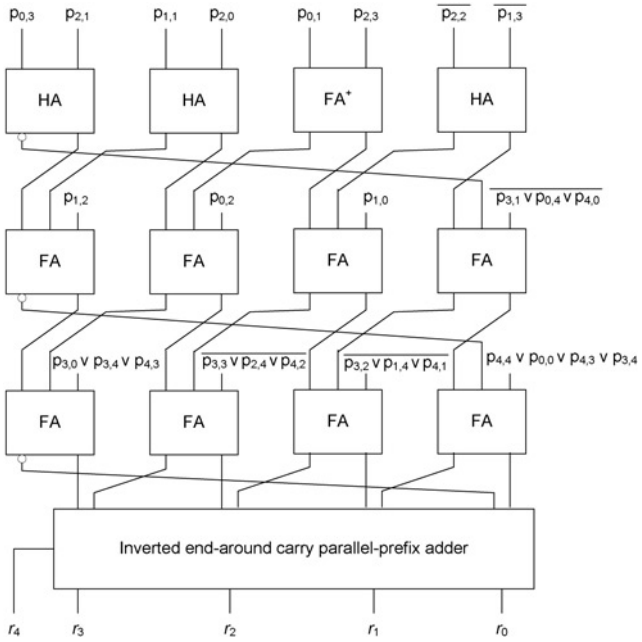

Fig. 3 *Proposed architecture of a modulo 17 multiplier*

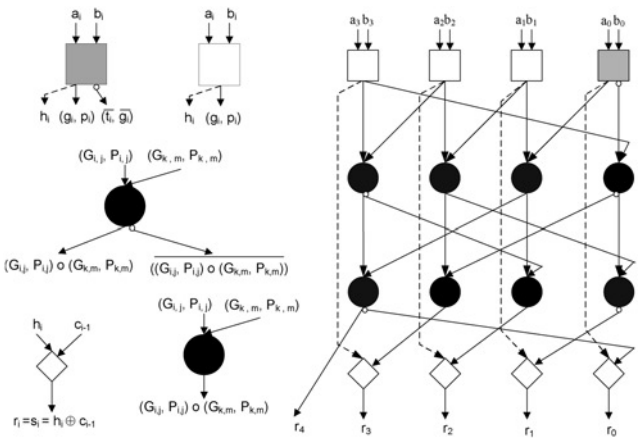

Fig. 4 *Final adder of the proposed modulo 17 multiplier*

equivalent gates. We therefore conclude that the proposed architecture leads to more compact designs. Table 4 lists for various operand sizes the savings in area offered by the proposed multipliers. Over the examined range, an average saving of 5.5% in the required area is observed.

The delay of the proposed multipliers also consists of three parts, namely, the delay of the partial-product formation, the delay of the CSA tree and the delay of the parallel inverted EAC adder. Since, the $p_{i,j}$ and $(\overline{p_{i,j}})$ terms are computed in 1 time unit and all the remaining partial-product terms in 3 time units, the upper bound of the delay for forming the partial-product matrix is 3 time units. However, as explained previously, in several cases we can parallelise some of this delay with that of the first stage of the CSA tree by driving the late-arriving partial-product bits to the FAs of subsequent stages. Hiding the delay of the late computed partial-product bits is not possible, however, if for an optimum adder tree depth, all partial-product bits are required in the first level of the tree. This is the case, when $n+1$ is a number of the Dadda sequence (6, 9, 13, 19, 28, 42, 63, ...). Therefore the delay for the partial-product formation is modelled as

$$T_{\text{partial}} = \begin{cases} 3, & \text{if } n+1 \text{ is a number of the Dadda sequence} \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

time units. The delay of a CSA tree designed according to Dadda [28] can be modelled as

$$T_{\text{CSA}} = 4D(n+1) \tag{13}$$

where $D(k)$ denotes the depth in FAs of a k-operand CSA tree, available in Table 5.1 in the work of Koren [32]. In the special case of $n = 4$, since the first stage of the tree can be constructed using only HAs and FA$^+$, we have that $T_{\text{CSA}} = 10$. The delay of the parallel-prefix inverted EAC adder is [15]

$$T_{\text{adder}} = 2\lceil \log\, n \rceil + 3 \tag{14}$$

Summing the delays of (12)–(14), we conclude that the delay of the proposed multipliers can be modelled as

$$T_{\text{proposed}} = \begin{cases} 18, & \text{if } n = 4 \\ 4D(n+1) + 2\lceil \log n \rceil + 6, & \text{if } n+1 \text{ is a number of the Dadda sequence} \\ 4D(n+1) + 2\lceil \log n \rceil + 4, & \text{otherwise.} \end{cases}$$

time units. The multipliers proposed in the work of Efstathiou *et al.* [24] exhibit a delay of

$$T[24] = \begin{cases} 4D(n+3) + 2\lceil \log\, n \rceil + 2, \\ \quad \text{if } n+1 \text{ or } n+2 \text{ is a number of the} \\ \quad \text{Dadda sequence} \\ 4D(n+3) + 2\lceil \log\, n \rceil + 4, \\ \quad \text{otherwise} \end{cases}$$

Table 5: Delay comparisons

n	4	8	12	16	20	24	28	32
T_{proposed}	18	28	34	36	42	42	46	46
T[24]	22	28	34	36	42	42	46	46

Table 5 lists for various operand sizes the delays offered by the proposed architecture and the architecture of Efstathiou *et al.* [24]. Form Table 5, it is obvious that in most cases, the two architectures offer similar execution delay.

We should however remember that the proposed multipliers offer handling of zero operands and results which is not offered by Efstathiou *et al.* [24]. Fig. 5*a* presents the extra circuits required by the proposal in the work of Efstathiou *et al.* [24] for handling zero operands and results. We consider that $X^* = x_{n-1}x_{n-2}\cdots x_0$ represents the diminished-1 representation of X and that x_z is a zero indication bit. A zero value is indicated by $x_z = 1$ and $X^* = 00\ldots0$. It is obvious that both the area and delay requirements of the multipliers proposed in the work of Efstathiou *et al.* [24] increase further when zero operands and results are considered. Moreover, since the proposed multipliers operate on operands with weighted representation they do not require any input/output translators in contrast to the architecture of Efstathiou *et al.* [24]. Fig. 5*b* and 5*c* present, respectively, the circuits required for translating from/to the weighted representation to/from the diminished-1. Although these circuits are only used at the start or the end of a computation cycle, their area and delay complexity should also be accounted for, when deciding on using a weighted or a diminished-1 representation.

Since the unit-gate model does not take into account the interconnect complexity, as well as, fan-in and fan-out requirements of the compared designs, we used a cell-based design approach to verify the qualitative comparisons. Each multiplier was described in HDL and mapped in a 0.18 μm CMOS standard cell library by a synthesis tool. After the initial mapping, we instructed the tool to perform iterative steps of delay optimisation on every design, until the fastest possible design was reached. Recursive area recovery steps were then applied. The derived netlists and the associated constraints were then passed to a standard cell place and route tool. All design constraints such as output load, max fan-out and floorplan initialisation were kept constant for each comparison. After annotating our netlists with the back-end information, static timing analysis was performed. The attained results are listed in Table 6. The reported area results include both cell and interconnect area and are given in μm^2. Delay results are reported in nanoseconds. On the average of the examined cases, the proposed architecture leads to 9.7% more compact and in parallel 6.7% faster multipliers.

4 Conclusions

A novel architecture for modulo $2^n + 1$ multiplier design is proposed in this paper. The proposed architecture improves

Table 4: Area comparisons

n	4	8	12	16	20	24	28	32
A_{proposed}	147	577	1299	2241	3529	5001	6729	8713
A [24]	170	628	1378	2348	3664	5164	6920	8932
Savings (%)	13.5	8.1	5.7	4.6	3.7	3.2	2.8	2.5

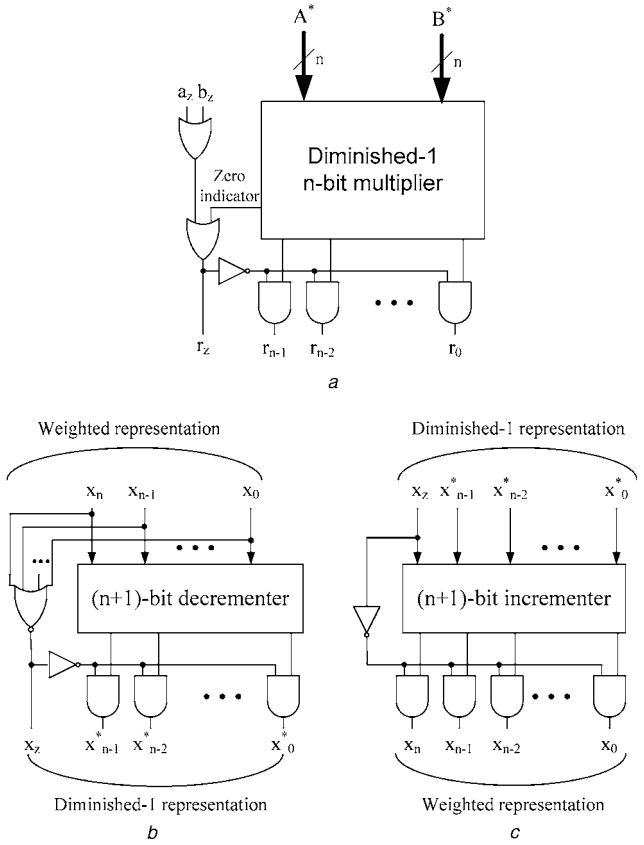

Fig. 5 *Overhead circuits required in the diminished-1 multipliers of Efstathiou et al. [24]*

a Circuit required for handling zero operands and results
b Input translator
c Output translator

greatly the one proposed in the work of Wrzyszcz and Milford [20] by combining all corrections into a single one, thereby decreasing the required parallel additions from 3 to 1. Even higher speed is attained by treating part of the required correction as a partial product whereas the rest is handled by the last stage adder. Our comparisons with the most efficient in delay and area terms solution indicate that the proposed multipliers offer the same or even better speed while being more compact. Considering moreover, that the proposed multipliers accept operands in weighted form whereas the solution presented in the work of Efstathiou *et al.* [24] operands in diminished-1 representation, it is clear that the proposed multipliers can be used more efficiently, since they do not require time-and hardware-consuming input and output translators nor any overhead circuit for handling of zero operands and results.

Table 6: Area and delay results from cell-based implementations

	Multipliers of Efstathiou *et al.* [24]		Proposed multipliers	
n	Area	Delay	Area	Delay
4	3 724	1.34	3 206	1.09
8	9 685	1.98	8 819	1.96
16	36 541	2.65	34 252	2.61
32	118 552	3.63	111 547	3.58

5 Acknowledgment

The research presented in this work was conducted within the framework of the Educational and Initial Vocational Training Program 'Archimedes' which is co-funded by the E.U. (75%) and by the Greek government (25%).

6 References

1 Taylor, F.: 'A single modulus ALU for signal processing', *IEEE Trans. Acoust. Speech Signal Process.*, 1985, **33**, pp. 1302–1315
2 Bayoumi, M., Jullien, G.A., and Miller, W.C.: 'A look-up table VLSI design methodology for RNS structures used in DSP applications', *IEEE Trans. Circuits Syst. II*, 1987, **CAS-34**, pp. 604–616
3 DiClaudio, E., Piazza, F., and Orlandi, G.: 'Fast combinatorial RNS processors for DSP applications', *IEEE Trans. Comput.*, 1995, **44**, pp. 624–633
4 Keller, T., Liew, T.H., and Lajos, H.: 'Adaptive redundant residue number system coded multicarrier modulation', *IEEE J. Selected Areas Commun.*, 2000, **C-18**, (11), pp. 2292–2301
5 Ramirez, J., Garcia, A., Lopez-Buendo, S., and Lloris, A.: 'RNS-enabled digital signal processor design', *Electron. Lett.*, 2002, **38**, (6), pp. 266–268
6 Ramirez, J. and Meyer-Baese, U.: 'High performance, reduced complexity programmable RNS–FPL merged FIR filters', *Electron. Lett.*, 2002, **38**, (4), pp. 199–200
7 Leibowitz, LM.: 'A simplified binary arithmetic for the Fermat number transform', *IEEE Trans. Acoust. Speech Signal Process.*, 1976, **24**, pp. 356–359
8 Truong, T.K., Chang, J.J., Hsu, I.S., Pei, D.Y., and Reed, I.S: 'Techniques for computing the discrete Fourier transform using the quadratic residue Fermat number systems', *IEEE Trans. Comput.*, 1986, **C-35**, pp. 1008–1012
9 Benaissa, M., Bouridane, A., Dlay, S.S., and Holt, A.G.J.: 'Diminished-1 multiplier for a fast convolver and correlator using the Fermat number transform', *IEE Proc. G*, 1988, **135**, pp. 187–193
10 Sunder, S., El-Guibaly, F., and Antoniou, A.: 'Area-efficient diminished-1 multiplier for Fermat number-theoretic transform', *IEE Proc. G*, 1993, **140**, pp. 211–215
11 Zimmermann, R., Curiger, A., Bonnenberg, H., Kaeslin, H., Felber, N., and Fichtner, W.: 'A 177 Mb/s VLSI implementation of the international data encryption algorithm', *IEEE J. Solid-State Circuits*, 1994, **29**, (3), pp. 303–307
12 Ramirez, J., Garcia, A., Meyer-Baese, U., and Lloris, A.: 'Fast RNS FPL-based communications receiver design and implementation'. Proc. 12th Int. Conf. Field Programmable Logic, Lecture Notes in Computer Science, 2002, (Springer-Verlag), vol. 2438, pp. 472–481
13 Cardarilli, G.C., Nannarelli, A., and Re, M.: 'Reducing power dissipation in FIR filters using the residue number system'. Proc. 43rd IEEE Midwest Symposium on Circuits and Systems, 2000, pp. 320–323
14 Zimmerman, R.: 'Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication'. Proc. 14th IEEE Symposium on Computer Arithmetic, 1999, pp. 158–167
15 Vergos, H.T., Efstathiou, C., and Nikolos, D.: 'Diminished-one modulo $2^n + 1$ adder design', *IEEE Trans. Comput.*, 2002, **51**, pp. 1389–1399
16 Efstathiou, C., Vergos, H.T., and Nikolos, D.: 'Fast parallel-prefix modulo $2^n + 1$ adders', *IEEE Trans. Comput.*, 2004, **53**, pp. 1211–1216
17 Piestrak, S.J.: 'Design of residue generators and multioperand modular adders using carry-save adders', *IEEE Trans. Comput.*, 1994, **43**, pp. 68–77
18 Vergos, H.T., and Efstathiou, C.: 'Diminished-1 modulo $2^n + 1$ squarer design', *IEE Proc. – Comput. Digit. Tech.*, 2005, **152**, pp. 561–566
19 Hiasat, A.A.: 'A memoryless mod$(2^n \pm 1)$ residue multiplier', *Electron. Lett.*, 1992, **28**, (3), pp. 314–315
20 Wrzyszcz, A., and Milford, D.: 'A new modulo $2^a + 1$ multiplier'. Proc. Int. Conf. Computer Design (ICCD'93), 1995, pp. 614–617
21 Wang, Z., Jullien, G.A., and Miller, W.C.: 'An efficient tree architecture for modulo $2^n + 1$ multiplication', *J. VLSI Signal Process.*, 1996, **14**, pp. 241–248
22 Ma, Y.: 'A simplified architecture for modulo $(2^n + 1)$ multiplication', *IEEE Trans. Comput.*, 1998, **47**, (3), pp. 333– 337
23 Curiger, A.: 'VLSI architectures for computations in finite rings and fields'. Swiss Federal Institute of Technology, 1993
24 Efstathiou, C., Vergos, H.T., Dimitrakopoulos, G., and Nikolos, D.: 'Efficient diminished-1 modulo $2^n + 1$ multipliers', *IEEE Trans. Comput.*, 2005, **54**, pp. 491–496

25 Sousa, L., and Chaves, R.: 'A universal architecture for designing efficient modulo $2^n + 1$ multipliers', *IEEE Trans. Circuits Syst. I.*, 2005, **52**, pp. 1166–1178
26 Chaves, R., and Sousa, L.: 'Faster modulo $2^n + 1$ multipliers without booth recoding'. Proc. XX Conf. Design of Circuits and Integrated Systems (DCIS'05), 2005
27 Wallace, C.S.: 'A suggestion for a fast multiplier', *IEEE Trans. Electron. Comput.*, 1964, **EC-13**, pp. 14–17
28 Dadda, L.: 'On parallel digital multipliers', *Alta Frequenza*, 1976, **45**, pp. 574–580
29 Kogge, P.M., and Stone, H.S.: 'A parallel algorithm for the efficient solution of a general class of recurrence equations', *IEEE Trans. Comput.*, 1973, **C-22**, pp. 786–792
30 Ladner, R.E., and Fisher, M.J.: 'Parallel prefix computation', *J. ACM*, 1980, **27**, (4), pp. 831–838
31 Tyagi, A.: 'A reduced-area scheme for carry-select adders', *IEEE Trans. Comput.*, 1993, **42**, (10), pp. 1163–1170
32 Koren, I.: 'Computer arithmetic algorithms' (A.K. Peters, Natick, 2002, 2nd Edn.)