Fast modulo $2^n + 1$ multi-operand adders and residue generators

H.T. Vergos $^{a,*}$, D. Bakalis $^b$, C. Efstathiou $^c$

$^a$ Computer Engineering and Informatics Department, University of Patras, 26500 Patras, Greece
$^b$ Department of Physics, University of Patras, 26500 Patras, Greece
$^c$ Informatics Department, ATEI of Athens, 12210 Athens, Greece

**Abstract**

In this manuscript novel architectures for modulo $2^n + 1$ multi-operand addition and residue generation are introduced. The proposed arithmetic components consist of a translation stage, an inverted end-around-carry carry-save-adder tree and an enhanced diminished-1 modulo $2^n + 1$ adder. Qualitative and quantitative results indicate that the proposed architectures result in significantly faster and in several cases smaller circuits than the previously proposed.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Residue arithmetic has been used in digital computing systems for many years. In particular, modulo $2^n + 1$ arithmetic appears to play an important role in a variety of applications since it is used in pseudorandom number generation, cryptography [1–3] and convolution computations without round-off errors [4]. Moreover, it is most commonly met as a part of a residue number system (RNS) [5,6] which is an arithmetic system well suited to applications in which the operations are limited to addition, subtraction and multiplication. The RNS has been adopted in the design of digital signal processors [6–9], FIR filters [10,11] and communication components [12,13], offering enhanced operation speed and increased low-power characteristics.

In an RNS application which uses as its base the most frequently used three moduli set $\{2^n; 2^n/2; 2^n + 1\}$ the execution delay is dictated by the modulo $2^n + 1$ channel because this has to handle $(n + 1)$-bit wide operands. The diminished-1 representation [14] was introduced to alleviate this problem by having each operand represented decreased by one compared to its weighted representation and by deriving the results in an alternative manner when one or both operands or the results are zero. Hence the operands that are used in the computation units are $n$-bit wide. The need for handling zero operands and results separately, as well as, the need for converters from/to the weighted to/from the diminished-1 representation, make the use of the diminished-1 representation efficient only when a large number of calculations take place before a new conversion is required. In all other cases, modulo $2^n + 1$ components for operands in weighted representation are more suitable.

Several architectures have been presented for modulo $2^n + 1$ arithmetic components assuming the weighted operands representation. Among them, there are architectures for designing multi-operand modulo adders (MOMAs) [15–19] and residue generators (RGs) [18,19]. In this paper we present novel architectures for designing these arithmetic components that result in faster, and in several cases smaller, circuits than the previously proposed.

The rest of the paper is organized as follows. The next section presents an overview of multi-operand modulo adders and residue generators. The proposed architecture for MOMAs is analytically derived in Section 3. Its use in the design of efficient RGs is investigated in Section 4. Qualitative and quantitative comparison results against the architectures proposed in [18], that are currently considered the most efficient ones, are given in Section 5. Finally, the last section concludes the paper.

2. Overview of MOMAs and RGs

Hardware support for multi-operand modulo addition is highly appreciated in several multiply-and-add intensive computations performed over an RNS base such as digital filtering, convolution estimation and FFT transforms. Let $[X]_M$ denote the residue of $X$ taken modulo $M$. A MOMA is a circuit that accepts $k$ operands,
suppose $X_1, X_2, \ldots, X_k$, with $0 \leq X_1, X_2, \ldots, X_k < M$ and computes the residue of their sum taken modulo $M$, that is, it computes $S = (X_1 + X_2 + \cdots + X_k) \mod M$, with $0 \leq S < M$. In the following such a multi-operand modulo adder will be denoted as MOMA($k$, $M$).

Hwang [20, p. 99] suggested for the first time in the open literature a MOMA for $M = 2^n - 1$ implemented by a carry-save adder (CSA) tree with end-around carry (EAC). This MOMA has the same delay complexity as an integer multi-operand adder. The proposed architecture required several paralleladders connected in series. The problem of designing MOMAs for generalized moduli was considered in [16–19]. The architecture proposed for weighted MOMAs in [18] has been shown to be more efficient than those of [16,17]. However, it still requires two parallel adders connected in series (the second adder actually performs a constant subtraction) to provide an unbiased result and the CSA tree needs to be carefully designed for every distinct number of operands because of its irregularity. In [19] the need for two parallel adders connected in series was canceled at the cost of doubling each CSA stage of the tree and requiring four carry lookahead (CLA) units at the final stage that operate in parallel. Unfortunately, [19] does not effectively exploit the properties of arithmetic modulo $2^n + 1$. It therefore results in weighted MOMAs clearly inferior than those of [18].

Applications that rely on modulo arithmetic need a circuit that accepts as input a $k$-bit binary operand $X$ and produces at the output the residue of this operand taken modulo $M$, that is, it computes $|X|_M$. This arithmetic component is called a residue generator and will be denoted as RG($k$, $M$) in the following. Efficient RGS for the generic modulo case as well as for specific moduli cases, such as $2^n + 1$, have been proposed in [18]. The RGS of [18], which are considered the most efficient ones for the modulo $2^n + 1$ case, follow the same architecture as the MOMAs proposed in [18]. That is, they are composed of a CSA tree and a parallel adder that since it may output the desired residue in biased format, it needs to be followed by a constant subtractor and a multiplexer to select between the result of the adder or the subtractor. Hence, two parallel adders connected in series have to be used in those circuits too.

In this manuscript we introduce a new architecture for designing weighted MOMAs. The proposed architecture relies on $n$-bit vectors computations. To this end, we introduce a translator circuit that enables to express the modulo $2^n + 1$ sum of two $(n+1)$-bit operands as a congruent modulo $2^n + 1$ sum of $n$-bit operands. Each translator circuit accepts two $(n+1)$-bit weighted operands, $A_i$ and $B_i$, with $0 \leq A_i, B_i \leq 2^n$ and computes two $n$-bit vectors $U_i$ and $Y_i$, such that $|A_i + B_i|_{2^n+1} = |Y_i + U_i|_{2^n+1}$. The formal derivation of the translator circuit is given in Section 3.1. It is shown that the translator circuit is a simplified CSA stage; therefore, it has small area requirements and a constant execution delay. For a $k$ operand MOMA $\lfloor k/2 \rfloor$ translator circuits are used in parallel. An inverted EAC CSA adder tree is then used for reducing the outputs of these translators along with a vector that represents a total correction term in two final addends. The total correction factor accounts for both the $+1$ term introduced by each translator and the correction due to the adder tree itself. The formal introduction of the inverted EAC CSA adder tree along with the analytical derivation of the total correction term are given in Section 3.2. The final module used in the proposed architecture is an enhanced diminished-1 modulo $2^n + 1$ parallel adder. This module accepts the outputs of the adder tree and computes the $(n+1)$-bit result of the multi-operand addition. The modifications required over a normal diminished-1 adder are presented in Section 3.3 and are shown to have very small implementation area while they do not contribute to the critical path of the diminished-1 adder.

Table 1 summarizes the modules used in the proposed MOMA architecture. For each one it presents its inputs and outputs and briefly describes its functionality. The proposed architecture requires just one parallel adder and simple CSA stages; therefore, it outperforms all previous proposals on designing weighted MOMAs in terms of delay. In several cases it also provides more compact multi-operand adders.

The proposed MOMA architecture can also be used in the design of a modulo $2^n + 1$ RG circuit. The resulting residue generators are also faster than those of [18]. Finally, in both the MOMA and RG cases, the proposed circuits are built around a completely regular inverted EAC CSA tree for every modulus value and number of operands or number of bits, respectively, resulting in more efficient CMOS VLSI realizations.

---

**Table 1**

<table>
<thead>
<tr>
<th>Translator module</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>$</td>
<td>A_i + B_i</td>
</tr>
<tr>
<td>Notes</td>
<td>1. A MOMA($k, 2^n + 1$) requires $\lfloor k/2 \rfloor$ transmitters that are used in parallel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. The outputs of the translators are driven to an inverted EAC CSA tree</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inverted EAC CSA tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
</tr>
<tr>
<td>Functionality</td>
</tr>
<tr>
<td>Notes</td>
</tr>
<tr>
<td>2. The outputs $F$ and $J$ of the tree are driven to the enhanced diminished-1 modulo $2^n + 1$ adder</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enhanced diminished-1 modulo $2^n + 1$ adder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
</tr>
<tr>
<td>Outputs</td>
</tr>
<tr>
<td>Functionality</td>
</tr>
<tr>
<td>Notes</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
3. Novel multi-operand modulo $2^n + 1$ adders

In this section we propose a novel architecture for a weighted MOMA that consists of a translation stage, an inverted EAC CSA tree and an enhanced diminished-1 modulo $2^n + 1$ adder. Each of these components is described in detail in the following.

3.1. The translator circuit

Suppose that $A = a_n a_{n-1} a_{n-2} \ldots a_1 a_0$ and $B = b_n b_{n-1} b_{n-2} \ldots b_1 b_0$ denote two $(n + 1)$-bit operands in the weighted representation, with $0 \leq A, B \leq 2^n$. Let $A_{n-1}$ and $B_{n-1}$ denote the n-bit vectors composed by the least significant bits of $A$ and $B$, respectively. Let $s_n$ and $c_n$ denote the sum and carry bits of the $(a_n + b_n)$ addition which have weights $2^n$ and $2^{n+1}$, respectively. For the modulo $2^n + 1$ addition of $A, B$ it then holds that

\[ |A + B|_{2^n+1} = |2^n c_n + 2^s s_n + A_{n-1} + B_{n-1}|_{2^n+1} \]

(1)

Let $x$ denote the complement of bit $x$. Since it holds that $(-x) = \overline{x} - 1$, from (1) we can further derive that

\[ |A + B|_{2^n+1} = |A_{n-1} + B_{n-1} + 2^s s_n + 3|_{2^n+1} \]

(2)

Let $D = 2^n - 4 + 2^s s_n$. $D$ is represented by the n-bit vector $111 \ldots 1c_n s_n$. Then, (2) is equivalent to

\[ |A + B|_{2^n+1} = |A_{n-1} + B_{n-1} + D + 1|_{2^n+1} \]

(3)

Relation (3) reveals that the modulo $2^n + 1$ sum of two weighted operands can be computed by the modulo $2^n + 1$ addition of n-bit operands and constants.

Suppose that $U = u_n u_{n-1} u_{n-2} \ldots u_0$ and $Y = y_n y_{n-1} y_{n-2} \ldots y_0 \overline{0}$ represent the n-bit sum vector and the $(n + 1)$-bit carry vector, respectively, of the carry-save addition of the three n-bit vectors $A_{n-1}, B_{n-1}$ and $D$ indicated in (3). We then have that

\[ |A_{n-1} + B_{n-1} + D + 1|_{2^n+1} = |Y^* + U|_{2^n+1} \]

(4)

Since the most significant bit of $Y^*$ has a weight equal to $2^n$ and for $x \in \{0, 1\}$ it holds that

\[ |x2^n|_{2^n+1} = |x - 1|_{2^n+1} \]

(5)

we get

\[ |Y^*|_{2^n+1} = |Y - 1|_{2^n+1} \]

(6)

where $Y = y_n y_{n-1} y_{n-2} \ldots y_0 \overline{0}$. Substituting this in (4) we get

\[ |A_{n-1} + B_{n-1} + D + 1|_{2^n+1} = |Y + U|_{2^n+1} \]

(7)

and (3) finally becomes

\[ |A + B|_{2^n+1} = |Y + U + 1|_{2^n+1} \]

(8)

The last equation reveals that the sum of the $(n + 1)$-bit weighted operands $A$ and $B$ modulo $2^n + 1$ is congruent to the modulo $2^n + 1$ sum of the n-bit vectors $Y$ and $U$ plus 1. Therefore, the inverted EAC CSA block presented in Fig. 1 which accepts the n-bit vectors $A_{n-1}$, $B_{n-1}$ and $D$ (or equivalently, since the bits of $D$ depend only on $a_n$ and $b_n$, the $(n + 1)$-bit vectors $A$ and $B$) and produces $Y$ and $U$ can be used as a translator circuit from the weighted to an n-bit pair representation provided that a correction equal to 1 is also taken into account.

The $(n - 2)$ leftmost full adders (FAs) of Fig. 1 can be simplified, since their input bits coming from operand $D$ are always equal to 1. Simplifications are also possible at the two rightmost FAs along with the accompanying NAND and XNOR gates by considering that, in a modulo $2^n + 1$ operand, $a_n$ ($b_n$) and $a_0$ ($b_1$ or $b_0$) cannot be simultaneously at 1.

Consider now the case of a MOMA($k, 2^n + 1$) circuit, that is, a circuit that accepts $k(n + 1)$-bit weighted operands, $X_1, X_2, \ldots, X_k$, with $0 \leq X_1, X_2, \ldots, X_k \leq 2^n$ and produces at the output the $(n + 1)$-bit residue of their sum taken modulo $2^n + 1$. Generalizing the above, we may group the $k$ operands in $[k/2]$ pairs and use $[k/2]$ translator circuits in parallel (if $k$ is odd then $X_k$ is paired with 0) to derive $[k/2]$ pairs of n-bit vectors $Y_1, Y_2, \ldots, Y_{[k/2]}$ and $U_1, U_2, \ldots, U_{[k/2]}$ such that

\[ |X_1 + X_2 + \cdots + X_k|_{2^n+1} = \sum_{i=1}^{[k/2]} |Y_i|_{2^n} + \sum_{i=1}^{[k/2]} U_i + \left\lfloor \frac{k}{2} \right\rfloor \]

(9)

3.2. The inverted EAC CSA tree

After the translation stage only n-bit vectors need to be added. Multi-operand modulo $2^n + 1$ addition of n-bit vectors can be performed by utilizing an inverted EAC CSA tree for reducing the outputs of the translators in two final n-bit vectors. These are then driven to a final diminished-1 parallel adder that is enhanced in order to provide a $(n + 1)$-bit result.

In the proposed architecture we need to add in modulo $2^n + 1$ arithmetic $(2[k/2] + 1)$ n-bit vectors in total, that is, the vectors $Y_1, Y_2, U_1$ with $0 < i \leq [k/2]$ and a vector, COR, which will represent the total correction required. The inverted EAC CSA tree that is used for this purpose is composed of several stages where in each stage one or more inverted EAC CSA blocks are present. Each inverted EAC CSA block consists of n FAs and an inverter, as shown in Fig. 1. It accepts at the input 3 n-bit vectors and produces at the output 2 n-bit vectors. In other words, it serves as a $(3,2)$ compressor for n-bit vectors in modulo $2^n + 1$ arithmetic. The inversion is justified as follows: the most significant bit of the carry vector output of a CSA has a weight equal to $2^n$. According to (5), it can be complemented and added to the least significant bit position of the carry vector output provided that a correction equal to −1 is also taken into account.

Let us now consider the n-bit correction term. COR should not only include the correction introduced by the translation stage ($([k/2])$ term of (9)), but also the correction introduced by the inverted EAC CSA tree. Considering that each inverted EAC CSA block reduces the number of addends by one, $(2[k/2] - 1)$ inverted EAC CSA blocks are required for attaining two final addends from the $(2[k/2] + 1)$ inputs of the inverted EAC CSA tree, each one providing an inverted feedback carry. Therefore, the accumulated correction required is $-(2[k/2] - 1)$. 

![Fig. 1. Translator circuit.](image-url)
Let $F$ and $J$ denote the two final addends produced by the inverted EAC CSA tree. From the above analysis we get
\[
\sum_{i=1}^{[k/2]} Y_i + \sum_{i=1}^{[k/2]} U_i + \text{COR} = F + J - \left(2 \frac{k}{2} - 1\right) 2^{n+1}
\]
or equivalently that
\[
\sum_{i=1}^{[k/2]} Y_i + \sum_{i=1}^{[k/2]} U_i + \frac{k}{2} 2^{n+1} = F + J - \left(2 \frac{k}{2} - 1\right) - \text{COR} + \frac{k}{2} 2^{n+1}
\]
which according to (9), results into
\[
\left|X_1 + X_2 + \cdots + X_k\right| 2^{n+1} = \left|\left(F + J + 1\right) - \frac{k}{2}\right| - \text{COR} 2^{n+1} \tag{10}
\]

3.3. The final stage adder

Let $A$ and $B$ denote two $(n+1)$-bit operands such that $0 < A, B < 2^n$. In the diminished-1 representation, $A^*$ and $B^*$ are used to represent $A$ and $B$, with $A^* = A - 1$ and $B^* = B - 1$ respectively. Their diminished-1 sum $S^*$ is
\[
S^* = |S - 1| 2^{n+1} = |A + B - 1| 2^{n+1} = (A^* + B^* + 1) 2^{n+1}
\]
and can be computed by a diminished-1 modulo $2^n + 1$ parallel adder [21].

If we use as COR the constant $\left|\frac{k}{2}\right| 2^{n+1}$, (10) takes the form
\[
\left|X_1 + X_2 + \cdots + X_k\right| 2^{n+1} = \left|(F + J + 1) - \frac{k}{2}\right| - \text{COR} 2^{n+1}
\]
The last relation reveals that we can derive the $n$ least significant bits of the weighted MOMA$(k, 2^n + 1)$ result by adding $F$ and $J$ in a diminished-1 parallel adder.

The most significant bit of the weighted MOMA$(k, 2^n + 1)$ however needs further treatment. This should be 1 only when $\left|X_1 + X_2 + \cdots + X_k\right| 2^{n+1} = |F + J + 1| 2^{n+1} = 2^n$. Since $0 \leq F, J \leq 2^n - 1$, the latter equation holds only when $F + J = 2^n - 1$, or equivalently, when $F$ and $J$ are bit-wise complementary. This condition can be easily detected as $P_{n-1} \cdot G_{n-1}$, where $P_{n-1}$ and $G_{n-1}$ are the group propagate and group generate signals at the most significant bit position of the diminished-1 adder that accepts $F$ and $J$. Relation $P_{n-1} \cdot G_{n-1}$ indicates that $F$ and $J$ are complementary since $P_{n-1}$ is one if there is at least one 1 in every pair of corresponding input bits, while $G_{n-1}$ will be in parallel be at one if no pair has both bits at 1. Since both terms are available in every fast adder, the extra hardware required for the most significant bit of the weighted addition is very small and no delay is added on the critical path of the adder.

It should be finally noted that if $\left|\frac{k}{2}\right| 2^{n+1} = \left|\frac{k}{2}\right| 2^{n+1} = 2^n$ then adding COR in the inverted EAC CSA tree is not required since in modulo $2^n + 1$ arithmetic an inverted EAC CSA addition of $A, B$ with $-1$ results into $A + B + (-1) + 1 = A + B$.

3.4. An example of the proposed weighted MOMA

In this section we design as an example a weighted MOMA(6,17) using the architecture derived in the previous section. The proposed circuit is presented in Fig. 2.

Let the input operands be denoted as $X_1$ up to $X_6$. Three translator circuits that operate in parallel are required in the translation stage. Each translator accepts a pair of operands and produces a pair of $U_i, Y_i$ vectors. The $U_i$ and $Y_i$ vectors along with the correction vector COR $= -3_{17} = 14_{10} = 1100_2$ are then used in the inverted EAC CSA tree. We assume that this is designed as a Dadda tree. It has a depth of four stages with stages 1, 2, 3 and 4 accepting 7, 6, 4 and 3 operands and resulting in 6, 4, 3 and 2 operands, respectively. Each stage has one inverted EAC CSA block composed of four FAs and an inverter excluding the second stage that is composed of two inverted EAC CSA blocks. The two vectors $F$ and $J$ resulting from the CSA tree are then used as inputs to an enhanced diminished-1 parallel adder that produces the 5-bit result denoted as $s_5s_4s_3s_2s_1s_0$.

Let us further consider the following input vectors for these operands: $X_1 = X_6 = 4_{10}, X_2 = 12_{10}, X_3 = X_5 = 16_{10}$ and $X_4 = 9_{10}$. The translator circuits produce the vectors $U_1 = 0111_2, Y_1 = 1000_2, U_2 = 1010_2, Y_2 = 1001_2, U_3 = 0111_2$ and $Y_3 = 0000_2$. The operation of the CSA tree is indicated in Fig. 3. $S$ and $C$, respectively, represent the sum and carry vectors produced by each CSA block of the tree. The most significant bit of every carry vector is complemented and used at the least significant bit position in the next addition. These bits are circled in Fig. 3 and, as predicted by the $(2 \frac{k}{2} - 1)$ term, their number is equal to 5.

Since the two final vectors $F$ and $J$ are not complementary, the most significant bit of the result is 0. The diminished-1 adder is an adder that increments the integer sum of its input vectors when the carry output of their integer addition is 0 and leaves it unchanged otherwise. In our case, since $F = 0101_2$ and $J = 0100_2$, the diminished-1 adder will provide 10102 as the $n$ least significant bits of the result. Hence, it is computed that $\left|X_1 + X_2 + \cdots + X_6\right| 17 = 010102 = 1010_2$.

4. Novel modulo $2^n + 1$ residue generators

4.1. Using a MOMA for RG

In this section we consider the design of an RG$(z, 2^n + 1)$ circuit, that is, the design of a circuit that produces $|X| 2^{n+1}$.
the residue of the z-bit operand X, taken modulo $2^n + 1$. A residue generator circuit can be designed taking advantage of the periodic properties of the powers of 2 taken modulo $2^n + 1$ [18]. If we partition the z bits of the operand in n-bit parts, suppose $p_0, p_1, \ldots, p_{|z|/n}$, starting from the least significant bits, we have

$$|X|_{z+1} = |p_{|z|/n}| \times 2^{(|z|/n)} + \cdots + p_3 \times 2^3 + p_2 \times 2^2 + p_1 \times 2 + p_0 \times 2^0_{z+1}$$  

(13)

Since for $i \geq 0$ it holds that $|2^n_{z+1}| = (-1)^i$, from (13) we get

$$|X|_{z+1} = \begin{cases} |p_{|z|/n}| - \cdots - p_3 + p_2 - p_1 + p_0 & \text{if } |\bar{z}| \text{ even} \\ |p_{|z|/n}| + \cdots + p_3 - p_2 + p_1 - p_0 & \text{if } |\bar{z}| \text{ odd} \end{cases}$$  

(14)

Furthermore, for an n-bit vector p it holds that $|p|_{z+1} = |\bar{p}|_{2z+1}$, where $\bar{p}$ denotes the bit-by-bit complement of p. Hence,

$$|X|_{z+1} = \begin{cases} |p_{|z|/n}| - \cdots - (p_3 + 2) + (p_2 + 2) + p_0 & \text{if } |\bar{z}| \text{ even} \\ |p_{|z|/n}| + \cdots + (p_3 + 2) + p_2 + (p_1 + 2) + p_0 & \text{if } |\bar{z}| \text{ odd} \end{cases}$$  

(15)

According to the above equation we conclude that an RG(z, $2^n + 1$) circuit can be designed using the following steps [18]:

1. Partition the z bits in parts of n bits each starting from the least significant bits. Each part is a valid residue taken modulo $2^n + 1$. The last part may contain less than n bits.
2. Invert the bits of the odd numbered parts and insert a correction equal to 2 for each such part.
3. Use a MOMA [18] to add the even and the inverted odd parts along with a total correction term.

Taking advantage of the new MOMA architecture proposed in Section 3 we can use in the last step a proposed MOMA([z/n], $2^n + 1$) also taking into account the following:

1. The required correction due to the inversion of the bits of the odd numbered parts should be merged with COR derived in the previous section; therefore, only one correction term should be used in the inverted EAC CSA tree.
2. Since the proposed MOMA architecture accepts at the translation stage (n + 1)-bit operands whereas the parts consist only of n bits, the most significant bits of both operands in all translators are zero. Hence operand D of (3) has a constant value of $2^n - 1$ in every translator. Therefore, the required translator in the residue generation case is built solely on components with an area and time complexity equal to that of a half adder.

4.2. An example of the proposed residue generator

Consider the example of an RG(16,17) circuit. Since $z = 16$ and $n = 4$, the 16-bit input operand X is partitioned in $k = 4$ parts of four bits each ($p_0 = x_3x_2x_1x_0$, $p_1 = x_7x_6x_5x_4$, $p_2 = x_{11}x_{10}x_9x_8$ and $p_3 = x_{15}x_{14}x_{13}x_{12}$). Then, the bits of parts $p_1$ and $p_3$ are inverted and a MOMA(4,17) designed according to the architecture introduced in Section 3 is used to add these four vectors along with a total correction term which is equal to $2 = -(k/2) + 2 \times 2$. The $(2 \times 2)$ term is the correction due to the inversion of $p_1$ and $p_3$. The resulting circuit is given in Fig. 4. Simplified translator circuits are used at the input point that accept 4-bit input operands instead of 5-bit ones that a MOMA(4,17) uses. This is due to the fact that the most significant bits of the corresponding 5-bit input vectors are equal to 0.

Let us now consider the computation of the residue of 596010 taken modulo 17 by the proposed RG(16,17) circuit. Since
596010 ≈ 0001011101001002 we have $p_0 = 1000_2$, $p_1 = 0100_2$, $p_2 = 0111_2$ and $p_3 = 0001_2$. The odd numbered parts are inverted and become $p'_0 = 0100_2$, $p'_1 = 0111_2$ and $p'_3 = 0001_2$. A proposed MOMA(4,17) circuit with operands $p_0, p_1, p_2$ and $p_3$ is then used. The translator in the right side of Fig. 4 accepts $p_0$ and $p_1$ and provides the outputs $Y_1 = 0110_2$ and $U_1 = 1100_2$. The other translator accepts $p_2$ and $p_3$ and provides $Y_2 = 1110_2$ and $U_2 = 0110_2$. The inverted EAC CSA tree then accepts the $Y_1, Y_2, U_1, U_2$ vectors and the total correction term 00102. The outputs of the inverted EAC CSA tree are $Y = 0101_2$ and $J = 0100_2$ and are driven to the diminished-1 adder. The latter produces 10102 and since $F$ and $J$ are not complementary, the most significant bit of the result is 0. We therefore attain that $[5960]_{17} = 01010_2 = 1010$.

To attain quantitative comparison results for the two architectures we developed an HDL generator capable of providing structural descriptions of the subcomponents required in the architectures under comparison. The following assumptions were made

(i) All binary adders follow the Kogge–Stone [22] parallel-prefix carry computation architecture.
(ii) The bias subtractor used in [18] also follows a Kogge–Stone architecture but is simplified since one of its operands is a constant.
(iii) All diminished adders used follow the parallel-prefix carry computation architecture of [21].
(iv) The CSA trees used in the proposed designs follow a Dadda adder tree architecture.

Due to the irregularity of the inverted EAC CSA trees in the architectures proposed in [18] these had to be described by hand in every examined case. A structural interconnection of these subcomponents provided the target MOMAs and RGs' descriptions.

After simulating the resulting descriptions, the designs were mapped in an 180 nm CMOS standard cell library assuming typical process parameters. A bottom-up approach was followed during mapping. Each basic cell (for example the half and full adder cells) was iteratively optimized until no further delay savings were possible. The cells then underwent successive area recovery steps. Finally, “don’t touch” primitives were applied to them. Then, the same optimization procedure was applied to every subcomponent and successively to the entire circuit. In this way every design was mapped in the target technology as an interconnection of already optimized blocks and the architecture in each description was preserved as much as possible. All constraints, such as maximum fan-out, output capacitance and available input drive strength, were kept constant for all architectures. The derived netlists for

### Table 2
Comparison results from static CMOS implementations.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$n$</th>
<th>[18] Proposed Savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Delay (ns) Area ($\mu$m²) Delay (ns) Area ($\mu$m²) Delay Area</td>
</tr>
<tr>
<td>MOMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2.38 6757 2.08 5667 12.6 16.1</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3.04 12140 2.41 10725 20.7 11.7</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>3.41 16925 2.64 15754 22.6 6.9</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>3.71 21357 3.10 20251 16.4 5.2</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2.74 13486 2.28 11815 16.8 12.4</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>3.37 23341 2.61 21613 22.6 7.4</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>3.75 32057 2.86 31439 23.7 1.9</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>4.07 40404 3.30 40245 18.9 0.4</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>3.12 28073 2.49 24739 20.2 11.9</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>3.80 46141 2.82 44360 25.8 3.9</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>4.14 63131 3.05 63858 26.3 1.2</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>4.46 79653 3.50 81296 21.5 21.1</td>
</tr>
<tr>
<td>RG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1.64 3484 1.29 3273 21.3 6.1</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>2.27 5517 2.00 5448 11.9 13.3</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>2.64 9875 2.31 10453 12.5 5.9</td>
</tr>
<tr>
<td>64</td>
<td>4</td>
<td>3.28 18458 3.10 19340 5.5 4.8</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>2.02 8078 1.49 7285 26.2 10.1</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>2.64 12389 2.20 11628 16.7 4.6</td>
</tr>
<tr>
<td>64</td>
<td>8</td>
<td>3.03 20678 2.54 21755 16.2 5.2</td>
</tr>
<tr>
<td>128</td>
<td>8</td>
<td>3.70 37151 3.30 39688 10.8 6.8</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>2.45 17482 1.70 16047 30.6 8.2</td>
</tr>
<tr>
<td>64</td>
<td>16</td>
<td>3.08 26109 2.40 24959 22.1 4.4</td>
</tr>
<tr>
<td>128</td>
<td>16</td>
<td>3.47 42860 2.75 45372 20.7 5.9</td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>4.12 76275 3.50 81373 15.0 6.7</td>
</tr>
</tbody>
</table>

5. Evaluation and comparisons

In this section we compare the proposed MOMA and RG architectures against those proposed in [18]. Let us first consider the case of a MOMA($2^k + 1$). The architecture of [18] at first partitions the $k-n$ input bits into $n$ subsets, where each subset corresponds to a distinct weight equal to a power of 2. One subset consists of $2k$ bits whereas all other subsets consist of $k$ bits. One further bit is also assigned to each subset due to the n-bit correction term that is required. Then, an inverted EAC CSA tree is used, that reduces the $k-n$ input bits to two n-bit summands and a carry input bit. These are added by a second parallel adder (actually a subtractor) has to be used in order to remove the bias. Finally, a (n+1)-bit 2-to-1 multiplexer has to be used in order to select between the two adder outputs.

On the other hand, the proposed architecture at first utilizes $[k/2]$ translator circuits that compress the $k-n$ input bits to $k-n$ bits. These bits, along with the proper n-bit correction term are driven to an inverted EAC CSA tree that produces two n-bit final summands. Finally, the enhanced diminished-1 modulo $2^k + 1$ adder is used to add these two summands and produce the output.

Taking into account that the delay of a diminished-1 adder is approximately equal to that of a binary adder [21] and that the delay of a translator circuit is slightly larger than that of a 2-to-1 multiplexer, we conclude that the proposed MOMA ($2^k + 1$) circuits are faster than those of [18] since they do not require the second parallel addition (bias subtraction) and since the inverted EAC CSA tree of [18] has approximately twice the height compared to that required by the proposed architecture. Considering the area requirements, the removal of the bias subtractor and the multiplexer and the use of a shallower inverted EAC CSA tree in the proposed architecture is compensated by the insertion of the translator circuits. The number of translator circuits required depends on the number of operands, whereas the area of each translator circuit depends on their width. Hence, we expect that the proposed MOMA circuits will also be more compact than those of [18] for small and medium values of $n$ and $k$.

Similar conclusions can be drawn for the case of residue generators since their design is based on multi-operand modulo adders. There are however two main differences:

(i) simplified translators are used in the proposed RGs making them slightly faster, and
(ii) both the RGs of [18] and the proposed RGs use the same inverted EAC CSA trees. Hence, the proposed RGs will be smaller than those of [18] only for small values of $n$ and $k$.  

ARTICLE IN PRESS
each architecture and the layout constraints for achieving the predicted area and delay were then passed to a standard cell place and route tool. The floorplan initialization information was kept constant for each compared MOMA and RG. After a constraint driven place and route procedure the netlists were back annotated with the extracted information and static timing analysis was performed. The derived results are given in Table 2.

The attained results indicate that the proposed architectures outperform those of [18] considering the operation speed. On the average of the examined cases the proposed MOMAs and RGs are 20.7% and 17.5% faster than the currently fastest proposals. This is attributed to the removal of the second parallel adder (bias subtractor) and the shallower adder tree.

Considering the area complexity, the proposed MOMAs are also more compact than the proposal of [18] when both \( n \) and \( k \) are not large. This is due to the second adder (bias subtractor) and the higher adder tree required by the proposal of [18]. For example, the proposed MOMA(4, \( 2n + 1 \)) and MOMA(4, \( 2n + 1 \)) circuits are 16% and 12% more compact than those of [18], respectively. However, a large value of \( k \) implies a large number of translators whereas a large value of \( n \) results in wider translators. On the other hand, in the RGs cases, as explained before, the proposed architecture is apart from delay, area efficient as well, only when both \( n \) and \( k \) are small.

Finally, considering the area \( \times \) delay \(^2 \) as a metric of efficiency for comparing different architectures and focusing only on the modulo \( 2^{n+1} + 1 \) case, which is the Fermat number with the most practical interest, the proposed MOMAs and RGs are by 43.3% and 38.6% more efficient than the proposals of [18], respectively.

6. Conclusions

Multi-operand adders and residue generators are essential building blocks in almost all applications that use an RNS. Since an RNS is usually adopted for the speedup that it can offer over the straight binary representation and considering that most often a channel of the \( 2^n + 1 \) form is the execution bottleneck of a \( (2^1, 2^2, 2^3, 2^4) \) RNS, fast architectures for weighted modulo \( 2^{n+1} + 1 \) MOMAs and RGs are highly appreciated.

In this manuscript we have proposed novel architectures for designing these components. Both the proposed MOMAs and RGs are based on the use of translator circuits that enable to perform weighted operand addition using congruent \( n \)-bit additions. The derived architectures lead to delay efficient designs. Quantitative results indicated that, on the average of all examined cases, savings of approximately 19.1% in the delay are achieved over the currently fastest designs. Finally, the proposed architectures have a completely regular inverted EAC CSA tree for every number of operands saving the designer from having to devise a minimum depth tree by hand.

References