RNS multiplication/sum-of-squares units

D. Adamidis and H.T. Vergos

Abstract: Digital signal processing and multimedia applications often profit from the use of a residue number system. Among the most commonly used moduli, in such systems, are those of $2^n-1$ and $2^n+1$ forms and among the most commonly used operations are multiplication and sum-of-squares. These operations are currently performed using distinct design units and/or consecutive machine cycles. Novel architectures for combined units that perform modulo $2^n - 1$/diminished-1 modulo $2^n + 1$ multiplication or sum-of-squares depending on the value of a control signal are proposed.

1 Introduction

Non-positional residue number systems (RNS) have been proposed as an attractive alternative to the binary system for applications, whose arithmetic operations are limited to addition, subtraction, multiplication and squaring [1]. An RNS is defined by a set of $F$ moduli, suppose $\{m_1, m_2,\ldots, m_F\}$, that are pair-wise relative prime numbers. Assuming that $|A|_M$ denotes the modulo $M$ of $A$, that is, the least non-negative remainder of the division of $A$ by $M$, an integer $A$ has a unique representation in the RNS, given by the residues set $\{a_1, a_2,\ldots, a_F\}$, where $a_i = |A|_{m_i}$, if $A \geq 0$ and $a_i = |M + A|_{m_i}$, if $A < 0$, with $M = m_1 \times m_2 \times \cdots \times m_F$ and $1 \leq i \leq F$. An RNS operation $\diamond$ on $A$ and $B$ is defined as $Z = A \diamond B \equiv (z_1, z_2,\ldots, z_F) = (a_1, a_2,\ldots, a_F) \diamond (b_1, b_2,\ldots, b_F)$, where $z_i = |a_i \diamond b_i|_{m_i}$. That is, the computation of $z_i$ depends only on $a_i$, $b_i$ and $m_i$. Therefore each $z_i$ can be computed in parallel in a separate arithmetic unit, often called channel. Since each channel deals with small residues instead of wide numbers and all channels operate in parallel without any carry propagation from one to another, significant speed-up over the binary system may be achieved.

Several digital signal processors (DSPs) have been reported that adopt an RNS [2, 3]. RNS adoption has been proved very useful in enhancing the performance of modulation components used in communications [4] and in digital filter design [5], in which it can help reduce the power consumption when the number of taps used in the filter increases [6]. Therefore there is an increasing interest in efficient modulo $m_i$ arithmetic components. Very efficient architectures foradders [7, 8], multipliers [9, 10] and squarers [11] have appeared in the open literature when $m_i$ is of the $2^n - 1$ and $2^n + 1$ forms. It is therefore no surprise that RNS based on the set $\{2^n, 2^n-1, 2^n+1\}$ has received significant attention and is the most commonly used.

In an RNS that uses a moduli set of the $\{2^n, 2^n-1, 2^n+1\}$ form, however, the modulo $2^n + 1$ channel deals with operands that are 1 bit wider than the operands of the other two channels. To overcome this problem, and given that in the case of a zero operand the result can be derived straightforwardly in all the above operations, Leibowitz [12] introduced the diminished-1 representation, in which each number is represented decreased by 1 modulo $2^n + 1$ and all arithmetic operations are inhibited for a zero operand. This representation has the great advantage that $n$ bits are used for representing all operands. Its only disadvantage is that converters from/to the diminished-1 to/from the weighted representation are required. Since the use of an RNS is attractive in applications where a series of operations take place before a new conversion is required, the time spent on conversions is a very small portion of the total computation time. Efficient adders [13, 14], multipliers [15] and squarers [16] are available in the open literature for the diminished-1 number system.

Another operation often found in many DSP and multimedia algorithms is the sum-of-squares ($A^2 + B^2$) computation. This is commonly used in Euclidean branch calculation [17], image compression [18], pattern recognition [19], motion estimation in computer vision systems, Viterbi convolution codes [20], channel equalisation [21], filtering [22] and statistics. For performing the sum-of-squares operation, a multiplier–adder or a squarer–adder pair of units can be used. This approach, however, requires a significant delay for implementing the sum-of-squares function as well as the use of intermediate storage memory. Significant savings may result by having a distinct sum-of-squares unit. The later solution though increases significantly the implementation area.

This paper proposes a solution that stands between these two extremes. For the first time in the open literature, we introduce novel units that can perform either modulo multiplication or modulo sum-of-squares depending on the value of a control signal, for moduli of both the $2^n - 1$ and $2^n + 1$ forms. We will hereafter denote these units as MMSSU–1 and MMSSU+1, respectively. In the case of the modulo $2^n + 1$ arithmetic, we consider that the operands follow the diminished-1 representation. As we will show, these units can be built with small overhead on the area and delay of a corresponding modulo multiplier. On the
other hand, they are capable of providing significant savings in the delay of a modulo sum-of-squares calculation, compared to the case that the calculation is performed by two rounds of modulo multiplication and one modulo addition.

2 Proposed MMSSU\textsubscript{\textminus1} architectures

Suppose that \( A \) and \( B \), with \( A = a_{n-1}a_{n-2}\cdots a_0 \) and \( B = b_{n-1}b_{n-2}\cdots b_0 \), are two \( n \)-bit numbers in modulo \( 2^n - 1 \) representation. Since \( A = \sum_{i=0}^{n-1} a_i2^i \) and \( B = \sum_{j=0}^{n-1} b_j2^j \), for their modulo \( 2^n - 1 \) multiplication, we obtain

\[
|A \times B|_{2^n-1} = \left| \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j 2^{i+j} \right|_{2^n-1} = \left| \sum_{i=0}^{n-1} PP_i \right|_{2^n-1}
\]

where \( PP_i \) denotes a partial product and

\[
PP_i = \left| \sum_{j=0}^{n-1} a_i b_j 2^{i+j} \right|_{2^n-1}
\]

According to Wang \textit{et al.} \cite{9}, for a number \( G = g_{n-1}g_{n-2}\cdots g_0 \) in \([0, 2^n - 1]\), it holds that

\[
|G2^n|_{2^n-1} = g_{n-1-j}g_{n-2-j} \cdots g_{|j|}g_0 \quad (1)
\]

Therefore we can express the above partial products, directly from the bits of \( A \) and \( B \) in \( n \) bits, if we move the partial product bits with \( i+j \geq n \) to the \( |n+i+j| \) bit position. For example, in the case that \( n = 8 \) the partial products shown in the left-hand side of Table 1 can be derived.

For the modulo \( 2^n - 1 \) reduction of the products into two final addends a full adder (FA) based tree architecture with end-around-carry (EAC) can be used. Consider that \( c_n \) is the carry output at the most significant bit position of some stage \( i \) in the reduction scheme. Since \( c_n \) has a weight of \( 2^n \) and \( |c_n2^n|_{2^n-1} = |c_n|_{2^n-1} \), \( c_n \) can be simply added at the least significant bit position of the next stage, forming an end-around-carry modulo adder tree architecture. A parallel modulo \( 2^n - 1 \) adder \cite{7} can finally be used for obtaining the modulo product.

For the squares of \( A \) and \( B \), we have that

\[
A^2 = A \times A = \sum_{i=0}^{n-1} a_i 2^{2i} + \left( \sum_{i=1}^{n-1} \sum_{j=0}^{n-1} a_i a_j 2^{i+j} \right)
\]

and

\[
B^2 = B \times B = \sum_{i=0}^{n-1} b_i 2^{2i} + \left( \sum_{i=1}^{n-1} \sum_{j=0}^{n-1} b_i b_j 2^{i+j} \right)
\]

Therefore their sum-of-squares is

\[
A^2 + B^2 = \sum_{i=0}^{n-1} (a_i + b_i)2^{2i} + \left( \sum_{i=1}^{n-1} \sum_{j=0}^{n-1} (a_i a_j + b_i b_j)2^{i+j+1} \right) \quad (2)
\]

The single bit additions indicated by the term \( \sum_{i=0}^{n-1} (a_i + b_i)2^{2i} \) of (2) can be substituted by the corresponding sum and carry terms, leading to (\( \oplus \) is used to denote the exclusive-OR logic function)

\[
A^2 + B^2 = \sum_{i=0}^{n-1} (a_i \oplus b_i)2^{2i} + \sum_{i=0}^{n-1} (a_i b_i)2^{2i+1} + \left( \sum_{i=1}^{n-1} \sum_{j=0}^{n-1} (a_i a_j + b_i b_j)2^{i+j+1} \right) \quad (3)
\]

Therefore the partial products for the modulo \( 2^n - 1 \) sum-of-squares operation can be derived by applying (1) to the terms indicated in (3). For example, in the \( n = 8 \) case, the 10 partial products indicated in the right-hand side of Table 1 are derived. In the general case, \( n + 2 \) \((n + 1)\) partial products are derived when \( n \) is even (odd).

Again an FA-based tree architecture with EAC can be used to reduce the partial products into two final addends. A modulo \( 2^n - 1 \) parallel adder can then provide the sum-of-squares.

From the above analysis, it is obvious that the modulo \( 2^n - 1 \) multipliers and sum-of-squares units have a very similar structure, composed of a partial product generation subcircuit, a modulo partial product reduction unit and a final parallel adder. In the following two sections, we propose two architectures for unifying these designs into a single module.

2.1 Simple MMSSU\textsubscript{\textminus1} architecture

Let \( m \) denote the operation select signal of the MMSSU, that is, when \( m = 0 \) the MMSSU performs multiplication on its input operands, whereas in the \( m = 1 \) case, the sum-of-squares of the input operands is generated. A
straightforward solution to obtain a MMSSU$_{-1}$ is to add two all 0s partial products to the original multiplication matrix, so that both multiplication and sum-of-squares matrices have the same dimensions, then use a multiplexer for every bit that differs in these and finally use the reduction tree architecture of the sum-of-squares function followed by a parallel adder to obtain the final result. This solution forms our simple MMSSU$_{-1}$ architecture.

The simple architecture requires in the worst case:

- \( n \) XOR gates for forming the partial product bits of the form \( a_i \oplus b_j \).
- \( 2 \binom{n}{2} \) AND gates for generating the partial product bits of the form \( a_ia_j \) or \( b_ib_j \).
- \( n^2 \) AND gates for the generation of the partial product bits of the form \( a_ib_j \).
- \( n^2 \) multiplexers for selecting the correct partial product bits according to the desired operation. Multiplexers are not required for the zero partial product bits of both matrices.
- \( n \) AND gates that perform the logic function \( m(a_i \oplus b_j) \) and provide the rest \( n \) partial product bits that are not driven to multiplexers.
- An FA-based partial product reduction tree that provides the final two summands. We consider that this tree follows a Dadda [23] architecture. Since we use as basis the sum-of-squares partial product reduction scheme, the Dadda tree is presented with \( n + 2 \) partial products. It is well known that the depth in FA stages of a Dadda tree is a function, suppose \( D(k) \), of its number of operands (see Table 3 of Vergos and Efstathiou [16] for all practical values). The number of FAs required is \( n^2 \).
- A parallel modulo \( 2^n - 1 \) adder.

Obviously, the critical path of the simple MMSSU$_{-1}$ architecture, starts at the input of a XOR gate and passes through an AND gate, the adder tree architecture and the parallel adder.

Different architectures are often compared using the unit-gate model [24] approximations. Under this model, all 2-input monotonic gates count as one gate equivalent for both area and delay, whereas a 2-input XOR or XNOR gate or a \( 2 \rightarrow 1 \) multiplexer counts as 2 gate equivalents for both area and delay. Using this model, we can compute that the area \( \left(A_{\text{Simple},2^n-1}\right) \) and delay \( \left(T_{\text{Simple},2^n-1}\right) \) of the simple MMSSU$_{-1}$ architecture are given by (we assume the parallel-prefix implementation of [7] for the final adder)

\[
A_{\text{Simple},2^n-1} = n(11n + 3\log n + 6)
\]

\[
T_{\text{Simple},2^n-1} = 4D(n + 2) + 2\log n + 6
\]

### 2.2 Reduced area MMSSU$_{-1}$ architecture

The large number of multiplexers required in the simple MMSSU$_{-1}$ architecture is considered in this section. To this end, we define the following variables, for \( 0 \leq i \leq n - 1 \)

\[
c_i = a_i \land b_i
\]

\[
d_i = a_i \lor b_i
\]

\[
e_i = m(a_i \oplus b_i)
\]

where \( \land \) denotes the logic AND function. Using the newly introduced variables, the original multiplication and sum-of-squares matrices can be made very similar using of the following rules:

1. Each partial product bit of the form \( a_i b_j \) with \( j > i \) \((j < i)\) is substituted with its equivalent \( a_i c_j \) \((d_i b_j)\) in the multiplication partial product matrix. Bits of the form \( a_i b_j \) are not substituted.
2. In the sum-of-squares partial product matrix each bit of the form \( a_i a_j \) is substituted by \( a_i c_j \), each bit of the form \( b_i b_j \) becomes \( d_i b_j \) and finally each bit \( a_i \oplus b_j \) is substituted with \( e_i \).

In our example case of \( n = 8 \), Table 1 takes the form of Table 2, where the last two partial products are in the multiplication case equal to zero.

After these transformations, the resulting partial product tables for modulo multiplication and sum-of-squares are essentially the same, with the first column of the multiplication matrix transferred at the least significant position in the sum-of-squares one. We can therefore in both cases use the tree architecture of the sum-of-square case for our partial product reduction. In the case of multiplication, however \((m = 0)\), the most significant bits of the two final addends are computed at the rightmost FAs of the tree architecture. Considering then, the completely circular nature of the carry computation unit in every modulo \( 2^n - 1 \) adder [7], we conclude that this will provide a similarly rotated modulo \( 2^n - 1 \) sum when its operands are rotated. Therefore, the two addends produced by the FA tree are

<table>
<thead>
<tr>
<th>Multiplication partial products</th>
<th>Sum-of-squares partial products</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2^7)</td>
<td>(2^7)</td>
</tr>
<tr>
<td>(2^6)</td>
<td>(2^6)</td>
</tr>
<tr>
<td>(2^5)</td>
<td>(2^5)</td>
</tr>
<tr>
<td>(2^4)</td>
<td>(2^4)</td>
</tr>
<tr>
<td>(2^3)</td>
<td>(2^3)</td>
</tr>
<tr>
<td>(2^2)</td>
<td>(2^2)</td>
</tr>
<tr>
<td>(2^1)</td>
<td>(2^1)</td>
</tr>
<tr>
<td>(2^0)</td>
<td>(2^0)</td>
</tr>
</tbody>
</table>

Table 2: New form of partial products
directly driven to a final parallel modulo $2^n - 1$ adder and $n$ multiplexers controlled by $m$ perform the required right rotation in the case of a multiplication operation. Fig. 1 presents the proposed reduced area MMSSU$_{-1}$ when $n = 4$. The logic gates required for producing the partial product bits are not shown.

In the general case, the reduced area architecture is composed of:

- $n$ XOR gates for forming the terms of the form $a_i \oplus b_i$.
- $n$ AND gates for forming the $e_i$ terms.
- $2n2 → 1$ multiplexers for producing the $c_i$ and $d_i$ terms.
- $2^n2$ AND gates for generating the partial product bits of the form $a_i c_i$ or $d_i b_i$.
- $n$ AND gates for generating the partial product bits of the form $a_i b_i$.
- An FA based partial product reduction tree architecture, composed of $n^2$ FAs with a depth of $D(n + 2)$ FAs.
- A parallel modulo $2^n - 1$ adder.
- $n2 → 1$ multiplexers for performing a right rotation of the sum.

The critical path of the reduced area MMSSU$_{-1}$ architecture starts at an XOR gate followed by an AND gate that implements an $e_i$ term and passes through the tree of FAs, the parallel adder and the rotation multiplexers. Adapting the approximations of the unit-gate model, we can compute that the area ($A_{\text{Reduced},2^n-1}$) and delay ($T_{\text{Reduced},2^n-1}$) of the reduced area MMSSU$_{-1}$ architecture are given by (we assume the parallel-prefix implementation of the work of Kalambaukas et al. [7] for the final adder)

$$A_{\text{Reduced},2^n-1} = n(8n + 3 \log n + 13)$$
$$T_{\text{Reduced},2^n-1} = 4D(n + 2) + 2 \log n + 8$$

2.3 Qualitative comparisons

In this section, using the approximations of the unit-gate model we compare the proposed MMSSU$_{-1}$ against two baseline architectures capable of performing modulo multiplication and sum-of-squares, hereafter called system A and system B. In system A, only a multiplier and a modulo adder are available. Therefore, modulo multiplication is performed in one cycle, whereas the sum-of-squares operation is performed in three consecutive cycles; two multiplication cycles followed by a summation one. System B offers a multiplier, a squarer and a modulo adder. The sum-of-squares operation requires two consecutive squaring cycles followed by an addition. System B obviously computes the sum-of-squares faster than A since a dedicated squarer circuit is significantly faster than a multiplier used for performing squaring.

Considering that each multiplier follows the architecture proposed in the work of Wang et al. [9] and all adders the architecture of Kalambaukas et al. [7], we can obtain that their area requirements and delays are given by

$$A_{\text{Multiplier},2^n-1} = 8n^2 + 3n \log n - 10n$$
$$T_{\text{Multiplier},2^n-1} = 4D(n) + 2 \log n + 4$$
$$A_{\text{Adder},2^n-1} = 3n \log n + 4n$$
$$T_{\text{Adder},2^n-1} = 2 \log n + 3$$

According to Piestrak [11], a modulo $2^n - 1$ squarer has half the partial products of the corresponding modulo $2^n - 1$ multiplier. Therefore, its area and delay requirements can be modelled as

$$A_{\text{Squarer},2^n-1} = 7n^2 + 3n \log n + \frac{n^2 - 21n}{2}$$
$$T_{\text{Squarer},2^n-1} = 4D\left(\frac{n}{2}\right) + 2 \log n + 4$$

The above approximations enable us to model our baseline systems’ areas and delays when performing the sum-of-squares operation as

$$A_{A,2^n-1} = n(8n + 6 \log n - 6)$$
$$T_{A,2^n-1} = 8D(n) + 6 \log n + 11$$
$$A_{B,2^n-1} = n(8n + 7\left\lfloor\frac{n}{2}\right\rfloor + 9 \log n + \frac{n - 33}{2})$$
$$T_{B,2^n-1} = 8D\left(\frac{n}{2}\right) + 6 \log n + 11$$

The graphs in Fig. 2 present for several values of $n$ the area and delay estimations in the baseline systems assumed, as well as, in systems that adopt one of the proposed MMSSU$_{-1}$ architectures. The estimations show that the required implementation area of the proposed architectures lies within the requirements of the two baseline architectures. The proposed reduced area architecture requires only 6% or less implementation area more than the baseline architecture A, when $n ≥ 16$. The area of baseline architecture B is significantly larger than that of both proposed architectures. Its area requirements raise over 40% more than that of the proposed reduced area architecture when $n ≥ 12$. Both proposed architectures offer significantly reduced sum-of-squares execution delays compared to both baseline architectures, for $n ≥ 8$. The savings offered reach 54% in the case of the proposed simple architecture and the $n = 32$ case. Obviously, one may add a dedicated sum-of-squares unit in any of the baseline systems.
assumed, in order to compensate for this delay. This, however, would increase further the implementation area of such a system. Of course the price that has to be paid in every case of using one of the proposed MMSSU\_1 architectures is longer multiplication or simple squaring times compared to those offered by the baseline systems. For example, multiplication time is increased by 11% on the average when \( n \geq 16 \). Since the above results are based on a model that neglects the delay and area for routing as well as fan-in and fan-out, we present quantitative results on a model that neglects the delay and area for routing as well as fan-in and fan-out, we present quantitative results from full static CMOS implementations in Section 4.

### 3 Proposed MMSSU\_1 architectures

The multiplication of A and B, in diminished-1 modulo \( 2^a + 1 \) arithmetic, can be treated in an analogous way given in the preceding section. In the work of Efstathiou et al. [15], it has been shown that we can express each partial product of the diminished-1 modulo \( 2^a + 1 \) multiplication of A and B in \( n \) bits. The bits with weight \( 2^{n-i} \), \( 0 \leq i \leq n-1 \), are complemented and shifted to bit position \( \lceil n + i \rceil \). For each such complementation and shifting, a correction factor of \( 2^{n-i} \) must be added.

Let \( X_{-1} \) denote the diminished-1 representation of \( X \), that is, \( X_{-1} = X - 1 \) and \( X \neq 0 \). Then, for the modulo \( 2^a + 1 \) product \( P \) and the sum-of-squares \( SS \) of \( A \) and \( B \), with \( A_{-1} = a_n - a_{n-2} \cdots a_0 \) and \( B_{-1} = b_n - b_{n-2} \cdots b_0 \), in the diminished-1 representation we have, respectively, that

\[
|P_{-1}|_{2^a+1} = |P - 1|_{2^a+1} = |(A_{-1} + 1) \times (B_{-1} + 1) - 1|_{2^a+1} = |(A_{-1}B_{-1}) + A_{-1} + B_{-1}|_{2^a+1}
\]

\[
|SS_{-1}|_{2^a+1} = |SS - 1|_{2^a+1} = |(A_{-1} + 1)^2 + (B_{-1} + 1)^2 - 1|_{2^a+1} = |A_{-1}^2 + B_{-1}^2 + 2A_{-1} + 2B_{-1}|_{2^a+1}
\]

The above analysis indicates that in the diminished-1 modulo \( 2^a + 1 \) multiplication/sum-of-squares operations, more partial products are required compared to the corresponding modulo \( 2^a - 1 \) cases. For the multiplication operation, one is needed for \( A_{-1} \) and a second for \( B_{-1} \). For the sum-of-squares operation, one is required for \( 2A_{-1} \) and a second for \( 2B_{-1} \). A final extra partial product in both cases is also required for the total correction factor, which in the sum-of-squares case should also incorporate the constant term of (5).

In the multiplication case, when \( n = 8 \), by expanding (4) and applying the complementation and shifting method, we can obtain the first 10 partial products (PP\_0 up to PP\_9) listed in the left-hand side of Table 3. The rest two products will represent the required correction. In the following, we compute the total correction required in the general case. Let CP denote the total correction factor for the multiplication operation. CP can be computed by following the procedures presented in the work of Efstathiou et al. [15, 16] as CP = CP\_1 + CP\_2, where CP\_1 is the correction imposed by the partial product generation and CP\_2 is the correction imposed by the partial product reduction.

Since the complemented terms that appear at the partial products PP\_i, with \( 1 \leq i \leq n-1 \) require a correction of \( 2^a(2^i - 1) \), we can easily obtain that

\[
CP = \sum_{i=1}^{n-1} 2^a(2^i - 1) = 2^a(2^{n} - n - 1)
\]

The \( n + 4 \) partial products are reduced into two final summands by a multi-operand modulo \( 2^a + 1 \) FA-based tree architecture with complemented EAC [15]. Let \( c_n \) denote the carry output of the most significant bit position at some stage \( i \) of the reduction scheme. \( c_n \) has a weight of \( 2^a \). Since

\[
|c_n 2^a|_{2^a+1} = |c_n 2^a|_{2^a+1} = |2^a + c_n 2^a|_{2^a+1}
\]
can be complemented and added at the least significant bit position of the next stage provided that a correction of $2^n$ is taken into account. During the reduction of $n + 4$ partial products, $n + 2$ carries with weight $2^n$ are produced, leading to $\text{CP}_2 = 2^n(n + 2)$. Therefore,

$$|\text{CP}|_{2^{n-1}} = |\text{CP}_1 + \text{CP}_2|_{2^{n-1}} = 0$$

Since CP is treated in the proposed architectures as an extra partial product, we have to use in our reduction scheme its diminished-1 representation, $\text{CP}_{-1} = 2^n$. Since this value cannot be represented in $n$ bits, two partial products are inserted; the first is the all 1s partial product, whereas the second is the 00...01 partial product.

From (5), making the same substitutions as in (3), we can obtain that

$$|\text{SS}_{-1}|_{2^{n-1}} = |A_{-1}^2 + B_{-1}^2 + 2A_{-1} + 2B_{-1} + 1|_{2^{n+1}}$$

$$= \left(\sum_{i=0}^{n-1} (a_i \oplus b_i)2^{2i} + \sum_{i=0}^{n-1} (a_i b_i)2^{2i+1} + \text{NPP} + 1\right)_{2^{n+1}}$$

(6)

where

$$\text{NPP} = \left(\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} (a_i a_j + b_i b_j)2^{i+j+1}\right) + 2A_{-1} + 2B_{-1}$$

The 10 partial products required for the NPP when $n = 8$ are listed in the right-hand side of Table 3 (PP_0 up to PP_{10}). Again, each partial product is expressed in $n$ bits by complementations and shifts as explained earlier.

The $\sum_{i=0}^{n-1} (a_i \oplus b_i)2^{2i}$ term of (6) can be expressed by the last four partial products in the right-hand side of Table 3. (As will be explained later in the 1 in PP_{21} also incorporates the required total correction factor.) Since the non-zero bits of $\text{PP}_{n+2} (a_i b_i$ terms) and $\text{PP}_{n+4} (a_i a_j$ terms) can substitute the zero bits of the PP_i partial products, with $0 \leq i \leq (n - 1)$, we can then obtain a very regular partial product array, with $n + 4$ rows.

In the following, we compute the total correction factor required for the sum-of-squares operation, suppose CSS, and we show that this is equal to 3 independently of $n$. The total correction factor is given by $\text{CSS} = \text{CSS}_1 + \text{CSS}_2 + \text{CSS}_3$, where:

- $\text{CSS}_1$ is the correction factor imposed by the partial products formation. This is equal to

$$\text{CSS}_1 = \sum_{i=1}^{n} 2^n(2^i - 1) + 2 \times 2^n + 2^n(2^n - 1) = 2^n(3 \times 2^n - n - 1)$$

- $\text{CSS}_2$ is the correction factor required for the reduction of the partial products. Since $n + 2$ carries of weight $2^n$ are produced during the reduction of the $n + 4$ partial products into two final addends, we have that

$$\text{CSS}_2 = 2^n(n + 2)$$

- $\text{CSS}_3$ is equal to 1 and represents the last term of (6).

According to the above correction factor, we obtain that

$$|\text{CSS}|_{2^{n-1}} = |\text{CSS}_1 + \text{CSS}_2 + \text{CSS}_3|_{2^{n-1}} = 2^n(3 \times 2^n - n - 1) + 2^n(n + 2) + 1|_{2^{n+1}} = 3$$

Since we are using the diminished-1 representation of CSS in the partial products matrix, we have to use CSS_{-1} = 2, which explains the 1 in the second from the right bit of PP_{11} in Table 3.

The above analysis indicates that a diminished-1 modulo $2^n+1$ multiplier has a very similar structure to a sum-of-squares unit. Both are composed of a partial product generation subcircuit, a modulo partial product reduction unit and a final parallel adder. This will enable us in the following, to propose two architectures for the unification of these designs into a single unit.

3.1 Simple MMSSU_{n+1} architecture

Our simple MMSSU_{n+1} architecture unifies the multiplication and sum-of-squares operations by utilising a multiplexer for every partial product bit that differs in the two operation matrices. Suppose that the desired operation is selected by a signal $s$. When $s = 0$ ($s = 1$) multiplication (sum-of-squares) is performed. Therefore, the area
requirements of our simple MMSSU_{+1} architecture are in the worst case:
- n XOR (XNOR) gates for forming the partial product bits of the form \(a_i \oplus b_i\) (\(\bar{a}_i \oplus \bar{b}_i\)).
- \(2 \left( \frac{n}{2} \right)\) AND (NAND) gates for generating the partial product bits of the form \(a_i a_j \oplus \bar{a}_i \bar{a}_j\) or \(b_i b_j \oplus \bar{b}_i \bar{b}_j\).
- \(n^2\) AND (NAND) gates for the generation of the \(a_i b_j\) (\(\bar{a}_i \bar{b}_j\)) bits.
- \(n^2 + 3n\) multiplexers that select the correct partial product bits according to the desired operation. (Note that only \(n\) multiplexers are required for the last two partial products in the two operations.)
- An FA-based partial product reduction tree architecture. This reduces the \(n + 4\) partial products into two by a Dadda tree. The architecture uses \(n(n + 2)\) FAs.
- A parallel modulo \(2^{n+1}\) adder. We consider for it the parallel-prefix implementation presented in the work of Vergos et al. [14].

Obviously, the critical path of the simple MMSSU_{+1} architecture starts at the input of a XOR or XNOR gate, passes through a multiplexer, the adder tree and the parallel adder.

Using the unit-gate model approximations, we can compute that the area \(A_{\text{Simple,2}^{n+1}}\) and delay \(T_{\text{Simple,2}^{n+1}}\) of the simple MMSSU_{+1} architecture are given by

\[
A_{\text{Simple,2}^{n+1}} = 11n^2 + \frac{9}{2}n \log n + \frac{43}{2} n + 6
\]

\[
T_{\text{Simple,2}^{n+1}} = 4D(n + 4) + 2 \log n + 7
\]

### 3.2 Reduced Area MMSSU_{+1} Architecture

The reduced area architecture that we propose in this section aims at the reduction of the multiplexers required. We introduce the following variables based on the operation control signal \(s\):

- \(c_i = a_i s \lor b_i \bar{s}\) \(0 \leq i \leq (n-1)\)
- \(d_i = a_i \bar{s} \lor b_i s\) \(0 \leq i \leq (n - 1)\)
- \(e_i = s(a_i \oplus b_i)\) \(0 \leq i \leq \frac{n}{2}\)
- \(e_i = s(a_i \oplus b_i)\) \(\frac{n}{2} \leq i \leq (n - 1)\)

Utilising the above variables and following the substitution rules given in the modulo \(2^n - 1\) case, we can obtain very similar matrices for the multiplication and sum-of-squares operations. In our example case of \(n = 8\), the partial product matrices are transformed into the ones indicated in Table 4. The differences between these matrices are confined to:
- the leftmost column of the multiplication matrix is transferred at the least significant bit position in the sum-of-squares operation and
- some of the bits of this column are complemented.

We can therefore design a reduced area MMSSU_{+1} unit, starting off from a pure sum-of-squares unit and taking into account that in the case of a multiplication operation it computes the two final addends left rotated by one bit position and also requires some of the input bits to be complemented. This means that multiplexers need to be inserted at the following points:
- At the least significant bit position, for choosing between normal and complemented inputs.
- Before the final parallel adder, for performing a single bit right rotation of the operands, when \(s = 0\). Unfortunately, the carry computation unit of a diminished-1 modulo \(2^n + 1\) adder does not have a completely circular nature and therefore rotated operands do not yield rotated results. In contrast to the modulo \(2^n - 1\) case, we therefore have to insert the multiplexers before the final adder and not after it.
- At the carry inputs of the FAs of the most and the least significant bit positions, to select the appropriate carry between the one coming from the next to the left column and a complemented feedback carry from the previous FA stage. Two multiplexers are required per row of full adders of the Dadda tree.

Fig. 3 presents the proposed reduced area MMSSU_{+1} when \(n = 4\). The logic gates required for producing the partial product bits are not shown.

The area requirements of the reduced area architecture can be summarised into the following:

- \(n\) XOR (XNOR) gates for forming the partial product bits of the form \(a_i \oplus b_i(\bar{a}_i \oplus \bar{b}_i)\).

#### Table 4: New form of partial products

<table>
<thead>
<tr>
<th>Multiplication partial products</th>
<th>Sum-of-squares partial products</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{PP}_0) (a_0)</td>
<td>(\text{PP}_0) (a_0)</td>
</tr>
<tr>
<td>(\text{PP}_1) (a_1)</td>
<td>(\text{PP}_1) (a_1)</td>
</tr>
<tr>
<td>(\text{PP}_2) (a_2)</td>
<td>(\text{PP}_2) (a_2)</td>
</tr>
<tr>
<td>(\text{PP}_3) (a_3)</td>
<td>(\text{PP}_3) (a_3)</td>
</tr>
<tr>
<td>(\text{PP}_4) (a_4)</td>
<td>(\text{PP}_4) (a_4)</td>
</tr>
<tr>
<td>(\text{PP}_5) (a_5)</td>
<td>(\text{PP}_5) (a_5)</td>
</tr>
<tr>
<td>(\text{PP}_6) (a_6)</td>
<td>(\text{PP}_6) (a_6)</td>
</tr>
<tr>
<td>(\text{PP}_7) (a_7)</td>
<td>(\text{PP}_7) (a_7)</td>
</tr>
<tr>
<td>(\text{PP}_8) (a_8)</td>
<td>(\text{PP}_8) (a_8)</td>
</tr>
<tr>
<td>(\text{PP}_9) (a_9)</td>
<td>(\text{PP}_9) (a_9)</td>
</tr>
<tr>
<td>(\text{PP}<em>{10}) (a</em>{10})</td>
<td>(\text{PP}<em>{10}) (a</em>{10})</td>
</tr>
<tr>
<td>(\text{PP}<em>{11}) (a</em>{11})</td>
<td>(\text{PP}<em>{11}) (a</em>{11})</td>
</tr>
</tbody>
</table>
The critical path of the reduced area MMSSU\(_{+1}\) architecture is formed at the rightmost column of the array. It starts at a multiplexer that implements a \(c_i\) or \(d_i\) term, followed by an AND (NAND) gate and a multiplexer that decides whether the output of the gate or its complement is the required partial product bit. It then passes through the FA tree, the rotation multiplexers and the parallel adder. Since for \(n \geq 4\), \(D(n + 4) \leq n\), we conclude that the critical path through the FA tree includes at most two multiplexers. Adapting the approximations of the unit-gate model, we can compute that the area, \(A_{\text{Reduced}, 2n+1}\), and delay, \(T_{\text{Reduced}, 2n+1}\), of the reduced area MMSSU\(_{+1}\) architecture are modelled by

\[
A_{\text{Reduced}, 2n+1} = 8n^2 + \frac{9}{2}n \log n + \frac{63}{2}n + 18
\]

\[
T_{\text{Reduced}, 2n+1} = 4D(n + 4) + 2 \log n + 14
\]

### 3.3 Qualitative comparisons

Using the approximations of the unit-gate model, we again compare the proposed MMSSU\(_{+1}\) against the two baseline architectures A and B, where in A only a multiplier and a modulo adder are available, whereas B offers a multiplier, a squarer and a modulo adder. We consider that the multiplier follows the architecture proposed in the work of Efstathiou et al. [15], every parallel adder the architecture presented in the work of Vergos et al. [14] and the squarer the architecture proposed in the work of Vergos and Efstathiou [16]. The area and delays of these units can be modelled by

\[
A_{\text{Multiplier}, 2n+1} = 8n^2 + \frac{9}{2}n \log n + \frac{7}{2}n + 6
\]

\[
T_{\text{Multiplier}, 2n+1} = 4D(n + 2) + 2 \log n + 6
\]

\[
A_{\text{Adder}, 2n+1} = \frac{9}{2}n \log n + \frac{n}{2} + 6
\]

\[
T_{\text{Adder}, 2n+1} = 2 \log n + 3
\]

\[
A_{\text{Squarer}, 2n+1} = \frac{n^2}{2} + 7n \left[ \frac{n + 1}{2} \right] + \frac{9}{2}n \log n + \frac{n}{2} + 6
\]

\[
T_{\text{Squarer}, 2n+1} = 4D \left( \frac{n + 1}{2} \right) + 2 \log n + 4
\]

Using the above approximations, the area requirements and the delays of our baseline systems A and B when performing the sum-of-squares operation are modelled by

\[
A_{A, 2n+1} = 8n^2 + 9n \log n + 4n + 12
\]

\[
T_{A, 2n+1} = 8D(n + 2) + 6 \log n + 15
\]

\[
A_{B, 2n+1} = \frac{17}{2}n^2 + 7n \left[ \frac{n + 1}{2} \right] + \frac{27}{2}n \log n + \frac{9}{2}n + 18
\]

\[
T_{B, 2n+1} = 8D \left( \frac{n + 1}{2} \right) + 6 \log n + 11
\]

The charts given in Fig. 4 present for several values of \(n\) comparative estimations of the area and delay of the baseline systems assumed and the proposed architectures. The required implementation area of the proposed architectures again lies within the requirements of the two baseline architectures. The proposed reduced area architecture requires less than 5.9% implementation area more than the baseline architecture A, when \(n \geq 16\). The area of baseline architecture B on the other hand is significantly larger; on the average of the examined cases it requires more than 15 and 37% extra implementation area than the simple and the reduced area architectures, respectively.

Considering the execution delay, both proposed architectures offer significant savings during the sum-of-squares execution, for sufficiently large values of \(n\). The savings are greater in the case of the simple architecture. On the average of the examined cases, the savings offered are 53.4% and 31.8% over the baseline architectures A and B, respectively. The multiplexers added on the critical path of the proposed reduced area designs make them a little
### Qualitative Comparisons

<table>
<thead>
<tr>
<th>n</th>
<th>$A_{A,2^n+1}$</th>
<th>$A_{B,2^n+1}$</th>
<th>$A_{Simple,2^n+1}$</th>
<th>$A_{Reduced,2^n+1}$</th>
<th>$T_{A,2^n+1}$</th>
<th>$T_{B,2^n+1}$</th>
<th>$T_{Simple,2^n+1}$</th>
<th>$T_{Reduced,2^n+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>228</td>
<td>336</td>
<td>304</td>
<td>308</td>
<td>51</td>
<td>23</td>
<td>27</td>
<td>34</td>
</tr>
<tr>
<td>8</td>
<td>772</td>
<td>1146</td>
<td>990</td>
<td>890</td>
<td>73</td>
<td>45</td>
<td>33</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td>1599</td>
<td>2380</td>
<td>2041</td>
<td>1741</td>
<td>84</td>
<td>56</td>
<td>38</td>
<td>45</td>
</tr>
<tr>
<td>16</td>
<td>2700</td>
<td>4026</td>
<td>3454</td>
<td>2858</td>
<td>87</td>
<td>67</td>
<td>43</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>4069</td>
<td>6074</td>
<td>5224</td>
<td>4236</td>
<td>96</td>
<td>76</td>
<td>43</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>5706</td>
<td>8523</td>
<td>7353</td>
<td>5877</td>
<td>98</td>
<td>78</td>
<td>44</td>
<td>51</td>
</tr>
<tr>
<td>28</td>
<td>7607</td>
<td>11369</td>
<td>9837</td>
<td>7777</td>
<td>107</td>
<td>87</td>
<td>48</td>
<td>55</td>
</tr>
<tr>
<td>32</td>
<td>9772</td>
<td>14610</td>
<td>12678</td>
<td>9938</td>
<td>109</td>
<td>89</td>
<td>49</td>
<td>56</td>
</tr>
</tbody>
</table>

**Fig. 4** Qualitative comparisons

### Implementation Results (Area is in $\mu m^2$ and Delay in $ns$)

<table>
<thead>
<tr>
<th>n</th>
<th>Multipliers of [9]</th>
<th>Simple MMSSU$_{-1}$</th>
<th>Reduced MMSSU$_{-1}$</th>
<th>Multipliers of [15]</th>
<th>Simple MMSSU$_{+1}$</th>
<th>Reduced MMSSU$_{+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8619</td>
<td>1.98</td>
<td>12377</td>
<td>2.43</td>
<td>12101</td>
<td>2.56</td>
</tr>
<tr>
<td>8</td>
<td>27409</td>
<td>2.84</td>
<td>40521</td>
<td>3.28</td>
<td>38917</td>
<td>3.41</td>
</tr>
<tr>
<td>16</td>
<td>106889</td>
<td>3.91</td>
<td>130213</td>
<td>4.20</td>
<td>118325</td>
<td>4.35</td>
</tr>
<tr>
<td>32</td>
<td>439835</td>
<td>4.88</td>
<td>453808</td>
<td>5.19</td>
<td>440661</td>
<td>5.37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>Multipliers of [9]</th>
<th>Simple MMSSU$_{-1}$</th>
<th>Reduced MMSSU$_{-1}$</th>
<th>Multipliers of [15]</th>
<th>Simple MMSSU$_{+1}$</th>
<th>Reduced MMSSU$_{+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10703</td>
<td>2.27</td>
<td>17524</td>
<td>3.33</td>
<td>17685</td>
<td>3.78</td>
</tr>
<tr>
<td>8</td>
<td>38653</td>
<td>2.94</td>
<td>45553</td>
<td>3.46</td>
<td>44048</td>
<td>4.00</td>
</tr>
<tr>
<td>16</td>
<td>133933</td>
<td>3.73</td>
<td>141419</td>
<td>4.21</td>
<td>136886</td>
<td>4.83</td>
</tr>
<tr>
<td>32</td>
<td>442089</td>
<td>4.54</td>
<td>459397</td>
<td>5.23</td>
<td>442952</td>
<td>5.73</td>
</tr>
</tbody>
</table>

**Fig. 5** Implementation results (area is in $\mu m^2$ and delay in $ns$)
slower than those that follow the simple architecture. The savings offered are in this case 45.1 and 18.9% over the baseline architectures A and B, respectively.

4 Quantitative comparisons

For area and delay estimations from real implementations, we developed a software tool capable of providing structural HDL descriptions of the proposed MMSSUs and the array multipliers proposed in the work of Wang et al. [9] and Efstathiou et al. [15]. All partial reduction schemes make use of a Dadda tree FA-based architecture and all final adders are constructed according to the fast parallel-prefix solutions proposed in the work of Kalamboukas et al. [17] and Vergos et al. [14]. Operand sizes of 4, 8, 16 and 32 bits were considered.

After extensive simulation, the designs were synthesised in a 0.25 μm static CMOS technology with the restriction of a maximum fan-out of 8 and optimised following a standard optimization script from Bhatnagar [25]. The synthesised netlists along with the design constraints to achieve them were then used for placing and routing the designs. All design constraints, such as output load, floorplan initialisation information and pin placement were held constant for each architecture. Final timing analysis was performed after all RC parasitic information was extracted from the layout and back-annotated to the gate-level netlist. Typical voltage and temperature operating conditions are assumed along with average gate delays.

The charts given in Fig. 5 present the attained results. Considering the implementation area required, our results indicate that the proposed MMSSU+1 become more attractive as n increases; in the example case of n = 32, the simple and reduced area MMSSU+1, respectively, require 3.18 and 0.1% more area than the modulo multiplier. The delay overhead imposed by the proposed MMSSU+1 is about 0.35 and 0.5 ns for the simple and the reduced area architectures, respectively. We should keep in mind, that this is the worst delay required, that is, the delay irrespectively of the operation performed. On the contrary, a system that uses only a multiplier and a modulo adder, will require two multiplication and one addition cycles for the sum-of-squares operation. Taking into account that in the same implementation technology a modulo 2^{16} + 1 adder has an implementation area of 10 586 μm^2 and a delay of 1.77 ns [13], we can obtain that the multiplier–adder solution requires, in the n = 16 case, an implementation area of 117 475 μm^2 and offers an execution delay of 9.59 ns for the sum-of-squares operation. This is more than double the delay of both proposed MMSSU+1 architectures. Therefore, in an application that uses the sum-of-squares operation often, the proposed MMSSU+1 units are very attractive. Of course, we must always keep in mind that, in an RNS, the maximum clock frequency is determined by the delay of the slowest channel used.

In the modulo 2^n+1 case, the reduced area architecture actually shows area savings over the simple one for n = 8, 16 or 32. In the smallest examined case, the multiplexers required for handling the carries and the operands before the last-stage adder overcome the savings on the multiplexers for the partial product bits selection. Again, as n increases, the use of the proposed MMSSUs becomes more attractive, since their area requirements are closer to those of a multiplier. In the n = 16 and 32 example cases, the reduced area MMSSU architecture leads to implementations with only 2.2 and 0.2% more implementation area than the corresponding multipliers. The simple and reduced area implementations are on the average of the examined cases 0.7 and 1.2 ns slower than the corresponding multiplier. Taking into account that in the same implementation technology a modulo 2^{16} + 1 adder has an implementation area of 12 489 μm^2 and a delay of 1.68 ns [13], we can again obtain a multiplier–adder solution requires in the n = 16 case an implementation area of 146 422 μm^2 and offers an execution delay of 9.14 ns for the sum-of-squares operation. This makes the proposed MMSSU+1 far more attractive in an application that uses frequently the sum-of-squares operation.

5 Conclusions

The use of an RNS has been greatly appreciated in several DSP and multimedia applications that may make heavy use of both multiplication and sum-of-squares operations. The implementation area for having distinct units for these operations is very large, and solutions based on employing a multiplier–adder pair or a squarer–adder pair for performing the sum-of-squares operation require several machine cycles.

To this end, we have proposed two architectures for units that perform either the |X × Y|_R or the |X^2 + Y^2|_R operation depending on the value of a control signal. We considered that R can either have the 2^n – 1 or the 2^n + 1 form and that the diminished-1 representation is used in the latter case.

6 References