Diminished-1 Modulo $2^n + 1$ Squarer Design

H. T. Vergos
Computer Engineering & Informatics Dept.,
University of Patras, 26 500, Greece &
Computer Technology Institute,
3 Kolokotroni Str., 26 221 Patras, Greece.
E-mail : vergos@ceid.upatras.gr

C. Efstathiou
Informatics Department,
TEI of Athens,
12 210, Athens, Greece.
E-mail : cefsta@teiath.gr

Abstract

Squarers modulo M are useful design blocks for digital signal processors that internally use a residue number system and for implementing the exponentiators required in cryptographic algorithms. In these applications, some of the most commonly used moduli are those of the form $2^n + 1$. To avoid using $(n+1)$-bit circuits, the diminished-1 number system can be effectively used in modulo $2^n + 1$ arithmetic applications. In this paper, for the first time in the open literature, we formally derive modulo $2^n + 1$ squarers that adopt the diminished-1 number system. The resulting implementations are built using only full- or half-adders and a final diminished-1 adder and can therefore be pipelined straightforwardly.

1. Introduction

A non-positional residue number system (RNS) is defined by a set of L moduli, suppose $\{d_1, d_2, \ldots, d_L\}$, that are pair-wise relative prime. 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 set $\{a_1, a_2, \ldots, a_L\}$ of residues, where $a_i = |A|_{d_i}$, if $A \geq 0$ and $a_i = |D + A|_{d_i}$, if $A < 0$, with $D = d_1 \times d_2 \times \ldots \times d_L$. A RNS operation $\diamond$, is defined as $(z_1, z_2, \ldots, z_L) = (a_1, a_2, \ldots, a_L) \diamond (b_1, b_2, \ldots, b_L)$, where $z_i = |a_i \diamond b_i|_{d_i}$. Since the computation of z_i only depends on a_i, b_i and d_i, each z_i is computed in parallel in a separate arithmetic unit, often called *channel*. Note that each channel deals with small residues instead of wide numbers and since all channels operate in parallel significant speedup over the binary system may be achieved.

The adoption of a RNS is therefore a good choice for applications whose arithmetic operations are limited to addition, subtraction, multiplication and squaring [18, 17]. Several digital signal processors (DSPs) targeting applications such as filtering [10, 16, 14] or modulation for communication components [9, 15] have already been built adopting a RNS and many more are expected, provided that modulo d_i arithmetic components are available and efficient.

Efficient adders [8, 3, 2, 5], multipliers [7, 23] and squarers [13] have been presented for moduli of the $2^n - 1$ and $2^n + 1$ forms. Therefore, RNSs based on the set $\{2^n, 2^n - 1, 2^n + 1\}$ have received significant attention and are the most commonly used. For this set of moduli however, a new problem, namely the problem that the $2^n + 1$ channel has to deal with operands one bit wider than the other two, arises. To overcome this problem, and given that in the case of a zero operand the result can be derived straightforwardly, Leibowitz [11] introduced the diminished-1 representation.

In the diminished-1 representation each number is represented decreased by 1 modulo $2^n + 1$ and all arithmetic operations are inhibited for a zero operand (easily identified by the most significant bit being at 1 in its diminished-1 form). This representation has the great advantage that the numbers are represented by n bits. Its only disadvantage is that converters from / to the diminished-1 to / from weighted are required. However, since a RNS is used when a series of additions and multiplications take place, the conversions required are only a very small portion of the total computation time.

Although very efficient adders [24, 6, 20] and multipliers [22, 12, 4] have appeared when the diminished-1 number system is used for the modulo $2^n + 1$ channel, an efficient squarer circuit has not yet been presented. Although a modulo $2^n + 1$ multiplier can be also used for squaring, as we will show in this paper, a dedicated squarer may be implemented more efficiently.

Modulo $2^n + 1$ squarers also find great applicability in some cryptographic algorithms. In these algorithms the encryption and decryption processes, involve modular exponentations of the form $D = \left|A^E\right|_{2^n+1}$, which may be im-

Proceedings of the EUROMICRO Systems on Digital System Design (DSD'04)
0-7695-2203-3/04 $ 20.00 IEEE

plemented using square and multiply algorithms. In several cases the operands used, that is A, D and $2^n + 1$ are very wide, in the order of thousands of bits. The adoption of a RNS can also help in this case to speedup the computations. As an example, the square and multiply algorithm was used in [25] for implementing the modulo $2^{16} + 1$ exponentiator circuit required in the International Data Encryption Algorithm. Instead of a squarer, multipliers modulo $2^{16} + 1$ have been used, along with a n-bit representation in which all operands are represented in weighted form except the operand 2^n which is represented as an all 0s operand.

In this paper, we formally derive novel squarers modulo M, where M is of the form $2^n + 1$, when the operands adopt the diminished-1 representation. The derived squarers can be implemented by using only a carry save array (CSA) composed of full (FA) and half (HA) adders and a final modulo $2^n + 1$ diminished-1 parallel adder [24, 6, 20]. The resulting implementations, can perform squaring much faster than a diminished-1 multiplier and can be very easily pipelined up to the FA stage.

The rest of the paper is organized as follows. The derivation of the new squarers is given in Section II. An example of the proposed implementation is presented in Section III. The area and delay requirements of the proposed squarers are analyzed in Section IV. They indicate that the proposed squarers offer significant delay savings over a multiplier circuit designed according to [4]. Our conclusions are drawn in the last section.

2. Novel Squarers

In this section, we introduce a new architecture for modulo $2^n + 1$ squarers of diminished-1 operands. We first explain the derivation of the partial products. We then consider the reduction of the partial products in two summands.

Let A be a $(n+1)$-bit number, $A \in [0, 2^n + 1)$ and let $A_{-1} = a_{n-1}a_{n-2}\ldots a_1a_0$ denote its diminished-1 representation. Assume that Q denotes the square of A modulo $2^n + 1$, that is, $Q = |A^2|_{2^n+1}$. We then have :

$$\begin{aligned} Q_{-1} + 1 &= \left|(A_{-1} + 1)^2\right|_{2^n+1} \\ &= \left|A_{-1}^2 + 2 \times A_{-1} + 1\right|_{2^n+1} \end{aligned}$$

or equivalently,

$$\begin{aligned} Q_{-1} &= \left|A_{-1}^2 + 2 \times A_{-1}\right|_{2^n+1} \\ &= \left|\, \left|A_{-1}^2\right|_{2^n+1} + |2 \times A_{-1}|_{2^n+1} \,\right|_{2^n+1}. \end{aligned} \quad (1)$$

The term $|A_{-1}^2|_{2^n+1}$ of (1) can be expressed as

$$\begin{aligned} \left|A_{-1}^2\right|_{2^n+1} &= \left|\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i a_j 2^{i+j}\right|_{2^n+1} \\ &= \left|\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i a_j \left|2^{i+j}\right|_{2^n+1}\right|_{2^n+1}. \end{aligned} \quad (2)$$

Taking into account that $i + j \leq 2n - 2$, (2) can be written as

$$\left|A_{-1}^2\right|_{2^n+1} = \left|\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i a_j (-1)^s 2^{|i+j|_n}\right|_{2^n+1},$$

where

$$s = \begin{cases} 0, & \text{if } i + j < n \\ 1, & \text{if } i + j \geq n. \end{cases} \quad (3)$$

Since for $z \in \{0, 1\}$ it holds that

$$\left|-z\right|_{2^n+1} = \left|2^n + 1 - z\right|_{2^n+1} = \left|2^n + \overline{z}\right|_{2^n+1}, \quad (4)$$

then (3) can be expressed as

$$\left|A_{-1}^2\right|_{2^n+1} = \left|\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_{i,j} 2^{|i+j|_n}\right|_{2^n+1},$$

where

$$x_{i,j} = \begin{cases} a_i a_j, & \text{if } i + j < n \\ \left|2^n + \overline{a_i a_j}\right|_{2^n+1}, & \text{if } i + j \geq n \end{cases} \quad (5)$$

and $\overline{t}$ denotes the complement of bit t.

Equation (5) indicates that one way to form the partial products, is to complement each bit $a_i a_j$ with $i + j \geq n$, and place it at bit position $|i + j|_n$, provided that a correction equal to $\left|2^n 2^{|i+j|_n}\right|_{2^n+1}$ is taken into account. Therefore, (5) can be reformulated as

$$\left|A_{-1}^2\right|_{2^n+1} = \left|\sum_{i=0}^{n-1}(PP_i + C_i)\right|_{2^n+1} \quad (6)$$

where PP_i denotes the i-th partial product

$$PP_i = \sum_{j=0}^{n-1-i} a_i a_j \, 2^{|i+j|_n} + \sum_{j=n-i}^{n-1} \overline{a_i a_j} \, 2^{|i+j|_n}, \quad (7)$$

and C_i is the corresponding correction factor. Note that PP_0 does not contain any complemented bits and thus $C_0 = 0$. On the other hand, for $i \neq 0$, the value of C_i depends on the number of the complemented bits $\overline{a_i a_j}$, and is given by

$$C_i = \sum_{j=n-i}^{n-1} \left|2^n \, 2^{|i+j|_n}\right|_{2^n+1} = 2^n \, (2^i - 1). \quad (8)$$

	2^{n-1}	2^{n-2}	...	2^1	2^0		
$PP_0 =$	a_0a_{n-1}	a_0a_{n-2}	...	a_0a_1	a_0,	$C_0 =$	0
$PP_1 =$	a_1a_{n-2}	a_1a_{n-3}	...	a_1a_0	$\overline{a_1a_{n-1}}$,	$C_1 =$	$2^n(2^1-1)$
$PP_2 =$	a_2a_{n-3}	a_2a_{n-4}	...	$\overline{a_2a_{n-1}}$	$\overline{a_2a_{n-2}}$,	$C_2 =$	$2^n(2^2-1)$
...							
$PP_{n-2} =$	$a_{n-2}a_1$	$a_{n-2}a_0$	...	$\overline{a_{n-2}a_3}$	$\overline{a_{n-2}a_2}$,	$C_{n-2} =$	$2^n(2^{n-2}-1)$
$PP_{n-1} =$	$a_{n-1}a_0$	$\overline{a_{n-1}}$	...	$\overline{a_{n-1}a_2}$	$\overline{a_{n-1}a_1}$,	$C_{n-1} =$	$2^n(2^{n-1}-1)$.

Table 1. Partial products and correction factors

According to the above, and taking into account that $a_ia_i = a_i$ the partial products and correction factors presented in Table 1 are derived for the term $|A_{-1}^2|_{2^n+1}$ of (1). The total correction, C_P, required for the formation of the above n partial products is equal to

$$C_P = \sum_{i=0}^{n-1} C_i = C_0 + \sum_{i=1}^{n-1} 2^n(2^i - 1) = 2^n(2^n - 1 - n). \tag{9}$$

We can now notice by observing the columns of the partial products, that in the same column some terms appear twice, once as a_ia_j and once as a_ja_j, or once as $\overline{a_ia_j}$ and once as $\overline{a_ja_i}$. Since $a_ia_j = a_ja_i$ ($\overline{a_ia_j} = \overline{a_ja_i}$) and $a_ia_j + a_ia_j = 2 \times a_ia_j$ ($\overline{a_ia_j} + \overline{a_ja_i} = 2 \times \overline{a_ia_j}$) each such pair of product bits that appears in the column with bits of weight 2^{i+j} can be replaced by one product bit a_ia_j ($\overline{a_ia_j}$) in the column with bits of weight 2^{i+j+1}, that is, in the next to the left column. The pairs of the leftmost column, can also, as explained earlier, according to (5) be complemented and placed at the rightmost column, if a correction factor equal to 2^n is taken into account for each such complementation and placement.

The number of pairs of equal product bits that appear in the leftmost column is $\lfloor \frac{n}{2} \rfloor$, where $\lfloor x \rfloor$ denotes the greater integer which is less or equal to x. The total correction required by the simplification of the same terms is therefore equal to :

$$C_S = 2^n \lfloor \frac{n}{2} \rfloor. \tag{10}$$

The resulting partial products matrix have different forms, depending on whether n is odd or even. In the case that n is odd, each column of the newly formed matrix has exactly $\frac{n+1}{2}$ bits. When n is even on the other hand, the columns with bits of weight 2^{i+j}, with $i + j \in \{0, 2, 4, \ldots, n-2\}$ have $2 + \frac{n}{2}$ bits, whereas the rest columns have $\frac{n}{2} - 1$ bits.

Since we have derived the required partial products for the term $\left|A_{-1}^2\right|_{2^n+1}$, of (1), we now turn our focus to the term $|2 \times A_{-1}|_{2^n+1}$. This term leads to a new partial product equal to A_{-1} shifted one position to the left. According to (5) the a_{n-1} bit that overflows at the left end, can be complemented and placed at bit position 0, provided that a correction factor, C_N, equal to 2^n is taken into account. Therefore the $(n+1)$-th partial product of the proposed squarers is given by :

	2^{n-1}	2^{n-2}	...	2^2	2^1	2^0	
$PP_n =$	a_{n-2}	a_{n-1}	...	a_1	a_0	$\overline{a_n}$,	$C_N = 2^n$

The proposed squarers utilize one last partial product, that is, the diminished-1 modulo $2^n + 1$ representation of the sum of all correction factors required. These correction factors are C_P, C_S, C_N, but we must also take into account any correction factor introduced during the reduction of the partial products into two summands. In the following we explain how the latter correction factor can be computed.

We consider that the reduction of the partial products into two summands is performed by using a full adder (FA) based tree architecture. Tree architectures have been introduced by Wallace [21]. Dadda reduced their area requirements in [1]. Consider that c_n is the carry output at the most significant bit position of some stage i in the reduction scheme. c_n has a weight of 2^n. Since

$$\left|c_n 2^n\right|_{2^n+1} = \left| -c_n\right|_{2^n+1} = \left|2^n + \overline{c_n}\right|_{2^n+1}, \tag{11}$$

c_n 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. For computing the total correction factor required, we consider the following two cases :

- n is odd.

  Then, during the reduction of the $\frac{n+1}{2} + 2$ partial products, $\frac{n+1}{2}$ carries are produced. The correction required for the reduction scheme is then equal to $C_{R,odd} = 2^n \frac{n+1}{2}$.

  Let C denote the modulo $2^n + 1$ sum of all correction factors, that is, the modulo $2^n + 1$ sum of C_P, C_S, C_N and $C_{R,odd}$. Note that in parallel, C, is our last partial product. We then have :

$$\left|C\right|_{2^n+1} = \left|C_P + C_S + C_N + C_{R,odd}\right|_{2^n+1} = \\ = \left|2^n(2^n - n - 1) + 2^n\frac{n-1}{2} + 2^n + 2^n\frac{n+1}{2}\right|_{2^n+1} = 1. \tag{12}$$

2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
						a_0a_6	a_0a_5	a_0a_4	a_0a_3	a_0a_2	a_0a_1	a_0
					a_1a_6	a_1a_5	a_1a_4	a_1a_3	a_1a_2	a_1	a_1a_0	
				a_2a_6	a_2a_5	a_2a_4	a_2a_3	a_2	a_2a_1	a_2a_0		
			a_3a_6	a_3a_5	a_3a_4	a_3	a_3a_2	a_3a_1	a_3a_0			
		a_4a_6	a_4a_5	a_4	a_4a_3	a_4a_2	a_4a_1	a_4a_0				
	a_5a_6	a_5	a_5a_4	a_5a_3	a_5a_2	a_5a_1	a_5a_0					
a_6	a_6a_5	a_6a_4	a_6a_3	a_6a_2	a_6a_1	a_6a_0						

Table 2. Initial partial product matrix

- n is even.

  As analyzed above, in this case all columns of the partial products matrix do not have the same number of partial product bits. We apply a FA at each column with bits of weight 2^{i+j}, with $i+j \in \{0, 2, 4, .., n-2\}$, thereby reducing the number of partial product bits from $\frac{n}{2}+4$ to $\frac{n}{2}+2$. In parallel, the carries that are produced from these FAs, increase the number of partial product bits at each of the rest columns from $\frac{n}{2}+1$ to $\frac{n}{2}+2$. As a result, we get a completely rectangular array, in which every column has $\frac{n}{2}+2$ partial product bits.

  During their reduction into two summands $\frac{n}{2}$ carries are produced. Therefore, the correction required for the reduction scheme is equal to $C_{R,even} = 2^n \frac{n}{2}$. For the total correction, in this case we also get :

$$\left|C\right|_{2^n+1} = \left|C_P + C_S + C_N + C_{R,even}\right|_{2^n+1} = \\ = \left|2^n(2^n - n - 1) + 2^n\frac{n}{2} + 2^n + 2^n\frac{n}{2}\right|_{2^n+1} = 1. \quad (13)$$

Since C is treated in the proposed architecture as an extra partial product, we have to use in our reduction scheme its diminished-1 representation, i.e., C_{-1}, which is equal to the all 0s n-bit vector. Note that although $C_{-1} = 0$, it cannot be ignored during the reduction of the partial products, since in this case less than the computed carries of weight 2^n will be produced.

An implementation of the proposed architecture is therefore composed of AND or NAND gates that form a bit of each partial product, a Dadda tree that reduces the partial products into two summands, and a modulo $2^n + 1$ adder for diminished-1 operands [20] that accepts these two summands and produces the required product.

3. An example of the proposed squarers

In this section we present an example of the derived diminished-1 modulo $2^n + 1$ squarers. Consider the design of a modulo $2^7 + 1$ squarer. Let $A_{-1} = a_6a_5a_4a_3a_2a_1a_0$ be the input operand. We start off by the partial product matrix for the A^2_{-1} shown in Table 2.

We then complement each bit a_ia_j with $i + j \geq 7$, and place it at bit position $|i + j|_7$. This results into the following partial product matrix :

	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$PP_0 =$	a_0a_6	a_0a_5	a_0a_4	a_0a_3	a_0a_2	a_0a_1	a_0
$PP_1 =$	a_1a_5	a_1a_4	a_1a_3	a_1a_2	a_1	a_1a_0	$\overline{a_1a_6}$
$PP_2 =$	a_2a_4	a_2a_3	a_2	a_2a_1	a_2a_0	$\overline{a_2a_6}$	$\overline{a_2a_5}$
$PP_3 =$	a_3	a_3a_2	a_3a_1	a_3a_0	$\overline{a_3a_6}$	$\overline{a_3a_5}$	$\overline{a_3a_4}$
$PP_4 =$	a_4a_2	a_4a_1	a_4a_0	$\overline{a_4a_6}$	$\overline{a_4a_5}$	$\overline{a_4}$	$\overline{a_4a_3}$
$PP_5 =$	a_5a_1	a_5a_0	$\overline{a_5a_6}$	$\overline{a_5}$	$\overline{a_5a_4}$	$\overline{a_5a_3}$	$\overline{a_5a_2}$
$PP_6 =$	a_6a_0	$\overline{a_6}$	$\overline{a_6a_5}$	$\overline{a_6a_4}$	$\overline{a_6a_3}$	$\overline{a_6a_2}$	$\overline{a_6a_1}$

Identifying, pairs of equal terms in every column and substituting them with one term to the next to the left column, enables us to reduce the set of partial products into the following (note that again the pairs of the leftmost column are replaced by their complement term in the rightmost column) :

	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$PP_0 =$	a_5a_0	$\overline{a_6a_5}$	$\overline{a_6a_4}$	$\overline{a_6a_3}$	$\overline{a_6a_2}$	$\overline{a_6a_1}$	$\overline{a_6a_0}$
$PP_1 =$	a_4a_1	a_4a_0	a_3a_0	$\overline{a_5a_4}$	$\overline{a_5a_3}$	$\overline{a_5a_2}$	$\overline{a_5a_1}$
$PP_2 =$	a_3a_2	a_3a_1	a_2a_1	a_2a_0	a_1a_0	$\overline{a_4a_3}$	$\overline{a_4a_2}$
$PP_3 =$	a_3	$\overline{a_6}$	a_2	$\overline{a_5}$	a_1	$\overline{a_4}$	a_0

Apart from the four above partial products, the complete matrix of the squarer must also include $|2 \times A_{-1}|_{2^n+1}$ and the total correction C, that is, the following two partial products :

	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$PP_4 =$	a_5	a_4	a_3	a_2	a_1	a_0	$\overline{a_6}$
$PP_5 =$	0	0	0	0	0	0	0

These partial products can be reduced into two summands, by the tree architecture indicated in Fig. 1, which is composed only by FA and HA blocks. Each such block produces a carry at its left and a sum at its right output. The carries produced at the most significant bit position (leftmost carries) are complemented and added to the bits of the

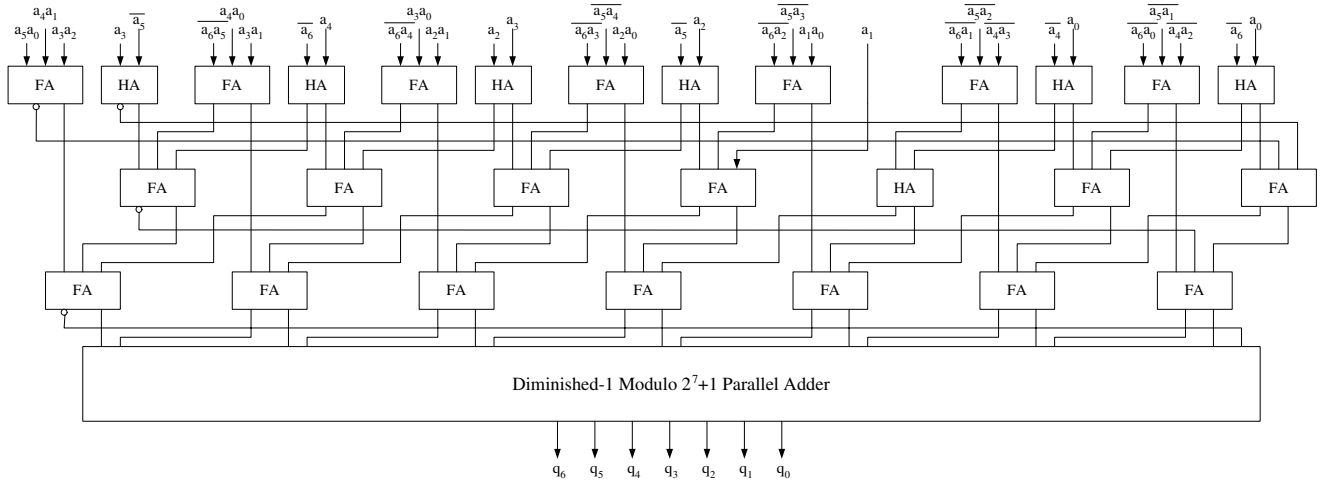

Figure 1. Proposed diminished-1 modulo $2^7 + 1$ squarer

least significant bit position. Note that in Fig. 1, 4 such carries are produced, in order to comply with the analysis preceding (12).

The two final summands are driven to a final parallel diminished-1 adder. It is obvious that the proposed squarer architecture is very regular and can be straightforwardly pipelined up to the FA stage, since the final adder if designed according to [20], can be pipelined up to the complex gate level.

4. Comparisons

In this section we examine the area and delay complexities of the proposed squarers. We further compare their delay against that of the multipliers proposed in [4]. The reasoning behind this comparison is that a squarer circuit would be finally included in a RNS implementation only if performing the squaring function by a multiplier would significantly slow down the execution rate.

For our comparisons, we adopt the approximations of the unit-gate model [19], that is, we consider that all 2-input monotonic gates count as one gate equivalent for both area and delay, while a 2-input XOR or XNOR gate counts as 2 gate equivalents for both area and delay.

In the proposed squarers, the required partial product bits can be derived in parallel by the use of $\frac{n(n-1)}{2}$ AND or NAND gates and $\lfloor \frac{n}{2} \rfloor$ inverters. We consider that these partial products are then reduced to two summands by the use of a Dadda tree. The depth in FA stages of a Dadda tree is a function, suppose $D(k)$, of its number of operands and is listed in Table 3 for all practical values.

Each of the n columns of the tree, is composed of *at most* $\lfloor \frac{n+1}{2} \rfloor$ FAs. (At most means that in several cases, further simplifications are possible. For example, in the first row of the squarer in Fig. 1, a_1, should be driven in both inputs of a HA at the column with bits of weight 2^2. This HA has been removed from Fig. 1, by substituting its sum output with 0 and its carry output with a_1. The first substitution also leads to the simplification of a FA of the second row into a HA. These simplifications depend on the actual value of n and therefore are not modelled in the sequel). The area and delay of a FA is 7 equivalent gates and 4 time units respectively.

k	$D(k)$ in FA stages
4	2
$5 \le k \le 6$	3
$7 \le k \le 9$	4
$10 \le k \le 13$	5
$14 \le k \le 19$	6
$20 \le k \le 28$	7
$29 \le k \le 42$	8
$43 \le k \le 63$	9
$63 \le k \le 94$	10

Table 3. FA stages in a k operand Dadda Tree

The area and delay of a n-bit parallel diminished-1 modulo $2^n + 1$ adder that follows the architecture proposed in [20] is $\frac{9}{2} n \log_2 n + \frac{1}{2} n + 6$ equivalent gates and $2 \log_2 n + 3$ time units. Therefore the area and delay requirements of the proposed modulo $2^n + 1$ squarers are :

$$\frac{n(n-1)}{2} + \frac{n}{2} + 7n \left\lfloor \frac{n+1}{2} \right\rfloor + \frac{9}{2} n \log_2 n + \frac{1}{2} n + 6 \text{ equivalent gates and}$$

$$1 + 4D \left\lfloor \frac{n+1}{2} \right\rfloor + 2 \log_2 n + 3 \text{ time units}$$

n	Multipliers of [4]	Proposed Squarers	Savings(%)
4	24	8	66.6
8	28	18	35.7
12	34	24	29.4
16	36	28	22.2
20	42	34	19.0
24	42	34	19.0
28	46	38	17.4
32	46	38	17.4

Table 4. Delay in equivalent gates

respectively.

The modulo $2^n + 1$ multipliers presented in [4], were compared against those presented in [22] and [12] and were found more efficient in both area and delay terms. Their delay using the unit-gate model is

$$\begin{cases} 4D(n+3) + 2\log_2 n + 2, & \text{if } n = 4, 5, 7, 8, 11, 12, \ldots \\ 4D(n+3) + 2\log_2 n + 4, & \text{otherwise.} \end{cases} \quad (14)$$

In Table 4 the delay of the multipliers presented in [4] and the proposed squarers are presented for several values of n, along with the savings offered by the proposed squarers. The proposed squarers are capable of offering savings in the delay of the squaring function up to 66.6% compared to when a multiplier circuit is used for it. The delay savings are well above 20% in the most interesting cases, from a practical point of view, that is, when $n \leq 16$. On the average of the examined cases, a dedicated squarer designed according to the proposed architecture offers 28.4% shorter delay.

5. Conclusions

Efficient modulo $2^n + 1$ squarers are useful design components in RNS and cryptography applications. In this paper we have derived a new architecture for designing diminished-1 modulo $2^n + 1$ squarers.

The proposed squarers offer significant savings in propagation delay over the case that a modulo $2^n + 1$ multiplier is used for performing the squaring function. The proposed architecture results in implementations with very regular structure well suited to VLSI implementations and straightforward to pipeline.

References

[1] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, 1965.

[2] G. Dimitrakopoulos, H. T. Vergos, D. Nikolos, and C. Efstathiou. A family of parallel-prefix modulo $2^n - 1$ adders. In *Proc. of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pages 326–336, 2003.

[3] G. Dimitrakopoulos, H. T. Vergos, D. Nikolos, and C. Efstathiou. A systematic methodology for designing area-time efficient parallel-prefix modulo $2^n - 1$ adders. In *Proc. of the IEEE International Symposium on Circuits and Systems*, pages 225–228, 2003.

[4] C. Efstathiou, H. T. Vergos, G. Dimitrakopoulos, and D. Nikolos. Efficient modulo $2^n + 1$ tree multipliers for diminished-1 operands. In *Proc. of the* 10^{th} *IEEE Int. Conference on Electronics, Circuits and Systems*, pages 200–203, 2003.

[5] C. Efstathiou, H. T. Vergos, and D. Nikolos. Fast parallel-prefix modulo $2^n + 1$ adders. In *Proc. of the XVII Conference on Design of Circuits and Integrated Systems*, pages 65–70, 2002.

[6] C. Efstathiou, H. T. Vergos, and D. Nikolos. Modulo $2^n \pm 1$ adder design using select-prefix blocks. *IEEE Trans. Comput.*, 52:1399–1406, 2003.

[7] C. Efstathiou, H. T. Vergos, and D. Nikolos. Modified Booth modulo $2^n - 1$ multipliers. *IEEE Trans. Comput.*, 53:370–374, 2004.

[8] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. T. Vergos, and J. Kalamatianos. High-speed parallel-prefix modulo $2^n - 1$ adders. *IEEE Trans. Comput.*, 49(7):673–680, 2000.

[9] T. Keller, T. H. Liew, and L. Hanzo. Adaptive redundant residue number system coded multicarrier modulation. *IEEE J. Select. Areas Commun.*, C-18(11):2292–2301, 2000.

[10] T. Kwan and T. Martin. Adaptive detection and enhancement of multiple sinusoids using a cascade IIR filter. *IEEE Trans. Circuits Syst.*, 36:937–945, 1989.

[11] L. M. Leibowitz. A simplified binary arithmetic for the Fermat number transform. *IEEE Trans. Acoust., Speech, Signal Processing*, 24:356–359, 1976.

[12] Y. Ma. A simplified architecture for modulo $(2^n + 1)$ multiplication. *IEEE Trans. Comput.*, 47(3):333–337, 1998.

[13] S. Piestrak. Design of squarers modulo A with low-level pipelining. *IEEE Trans. Comput.*, 49:31–41, 2002.

[14] J. Ramirez, A. Garcia, S. Lopez-Buedo, and A. Lloris. RNS-enabled digital signal processor design. *Electronics Letters*, 38(6):266–268, 2002.

[15] J. Ramirez, A. Garcia, U. Meyers-Baese, and A. Lloris. Fast RNS FPL-based communications receiver design and implementation. In *Proc. of the* 12^{th} *International Conference on Field Programmable Logic, Lecture Notes in Computer Science Vol. 2438, Springer-Verlag*, pages 472–481, 2002.

[16] J. Ramirez and U. Meyer-Baese. High performance, reduced complexity programmable RNS–FPL merged FIR filters. *Electronics Letters*, 38(4):199–200, 2002.

[17] M. A. Soderstrand et al. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, 1986.

[18] N. Szabo and R. Tanaka. *Residue Arithmetic and its Aplications to Computer Technology*. McGraw-Hill, 1967.

[19] A. Tyagi. A Reduced-Area Scheme for Carry-Select Adders. *IEEE Trans. Comput.*, 42(10):1163–1170, 1993.

[20] H. T. Vergos, C. Efstathiou, and D. Nikolos. Diminished-one modulo $2^n + 1$ adder design. *IEEE Trans. Comput.*, 51:1389–1399, 2002.

[21] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Trans. Electron. Comput.*, EC-13:14–17, 1964.

[22] Z. Wang, G. A. Jullien, and W. C. Miller. An efficient tree architecture for modulo $2^n + 1$ multiplication. *Journal of VLSI Signal Processing*, 14:241–248, 1996.

[23] A. Wrzyszcz and D. Milford. A new modulo $2^a + 1$ multiplier. In *Proc. of the International Conference on Computer Design*, pages 614–617, 1993.

[24] R. Zimmermann. *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*. PhD thesis, Swiss Federal Institute of Technology, 1997.

[25] R. Zimmermann et al. A 177 Mb/s VLSI Implementation of the International Data Encryption Algorithm. *IEEE J. Solid-State Circuits*, 29(3):303–307, 1994.