EFFICIENT ARCHITECTURES FOR MODULO 2^n-1 SQUARERS

A. Spyrou[1]*, D. Bakalis*[2]*, and H. T. Vergos*[1]

[1] Computer Engineering and Informatics Department
University of Patras
Patras, Greece

[2] Electronics Laboratory, Physics Department
University of Patras
Patras, Greece

ABSTRACT

Two novel architectures for designing modulo 2^n-1 squarers are given. The first one does not perform any encoding on the input operand, while the second one uses Booth-encoding. Pre-layout estimates indicate that both architectures result in area and/or delay efficient modulo 2^n-1 squarers. The non-encoded modulo squarers are more suitable for small values of n while the Booth-encoded modulo squarers are more suitable for medium and large values of n.

Index Terms— Squaring operation, modulo 2^n-1 arithmetic circuits, residue number system.

1. INTRODUCTION

Squaring is an operation met very often in digital signal processing applications, such as vector quantization [1], image compression and equalization [2] [3] and mean square error estimation. Several design approaches [4-7] have been proposed in the past to increase the performance of binary squarers.

Residue arithmetic has been also used in digital computing systems for many years. A Residue Number System (RNS) has been adopted in the design of several digital systems [8] [9], such as digital signal processors [10], [11], FIR filters [12-14], Discrete Cosine Transform processors [15] and communication components [16-18].

In an RNS, a number X is represented as a set of residues (X_1, X_2, ..., X_M), where X_i = X mod m_i (hereafter denoted by $X_i = |X|_{m_i}$). The m_is, with $1 \le i \le M$, are pair-wise relative prime integers and compose the base of the RNS. Every operation, suppose $\circ$, is executed in parallel on the corresponding residues of the two operands over the corresponding modulo arithmetic and produces a new set of residues. That is, $Z = X \circ Y = (Z_1, Z_2, \ldots, Z_M)$ is computed as:

$$(Z_1, Z_2, \ldots, Z_M) = (X_1, X_2, \ldots, X_M) \circ (Y_1, Y_2, \ldots, Y_M)$$
$$= (|X_1 \circ Y_1|_{m_1}, |X_2 \circ Y_2|_{m_2}, \ldots, |X_M \circ Y_M|_{m_M})$$

Since each Z_i is computed in a distinct arithmetic unit (commonly used channel) and its computation depends only on X_i, Y_i and m_i, all channel computations can be performed in parallel without the need of carry propagation among the channels, leading to significant speedup over the corresponding binary operations. Three-moduli bases of the form {2^n-1, 2^n, 2^n+1} have received significant attention, mainly due to the existence of very efficient combinational converters from/to the binary system. Therefore, the design of efficient modulo arithmetic components for the above-mentioned moduli is vital in RNS-based applications.

Several approaches can be used for designing efficient modulo squarers. For small values of moduli, either direct realizations of the minimized logic functions [19] or look-up tables can be utilized. However, these approaches are inefficient for medium or large values of moduli. Another solution is to use modulo multipliers whose inputs are driven by the same operand. The resulting modulo squarers however are unnecessarily complex and slow. Another approach is to derive the partial products required for the modulo squaring operation and utilize an array or tree adder-based approach to sum them. During the last years, several architectures based on the last approach have been presented for the most commonly used moduli, that is, 2^n+1 [19-21] and 2^n-1 [19] [22]. In the 2^n-1 moduli case, both architectures of [19] and [22] result in the same modulo squarers. However, none of them uses the Booth-encoding technique, which is commonly used in binary multipliers and squarers in order to reduce the number of partial products.

In this paper, we present two novel architectures for designing efficient non-encoded and Booth-encoded modulo 2^n-1 squarers. The proposed non-encoded modulo 2^n-1 squarers rely on the work of [6] for binary squarers and offer a reduced partial products matrix and reduced implementation area compared to the modulo squarers of [19] [22]. The proposed Booth-encoded modulo 2^n-1 squarers follow the Booth-folding technique proposed for binary squarers in [7] and can perform modulo squaring significantly faster compared to the modulo squarers that can be derived by the Booth-encoded modulo 2^n-1 multipliers of [23].

The rest of the paper is organized as follows: The proposed modulo 2^n-1 squaring architectures are derived in

2^{14}	2^{13}	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_7	a_0a_6	a_0a_5	a_0a_4	a_0a_3	a_0a_2	a_0a_1	a_0a_0
						a_1a_7	a_1a_6	a_1a_5	a_1a_4	a_1a_3	a_1a_2	a_1a_1	a_1a_0	
					a_2a_7	a_2a_6	a_2a_5	a_2a_4	a_2a_3	a_2a_2	a_2a_1	a_2a_0		
				a_3a_7	a_3a_6	a_3a_5	a_3a_4	a_3a_3	a_3a_2	a_3a_1	a_3a_0			
			a_4a_7	a_4a_6	a_4a_5	a_4a_4	a_4a_3	a_4a_2	a_4a_1	a_4a_0				
		a_5a_7	a_5a_6	a_5a_5	a_5a_4	a_5a_3	a_5a_2	a_5a_1	a_5a_0					
	a_6a_7	a_6a_6	a_6a_5	a_6a_4	a_6a_3	a_6a_2	a_6a_1	a_6a_0						
a_7a_7	a_7a_6	a_7a_5	a_7a_4	a_7a_3	a_7a_2	a_7a_1	a_7a_0							

Fig. 1. Initial partial products matrix for the A^2 term in the modulo 2^8-1 case.

2^{14}	2^{13}	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_7	a_0a_6	a_0a_5	a_0a_4	a_0a_3	a_0a_2	a_0a_1		a_0
					a_1a_7	a_1a_6	a_1a_5	a_1a_4	a_1a_3	a_1a_2		a_1		
				a_2a_7	a_2a_6	a_2a_5	a_2a_4	a_2a_3		a_2				
			a_3a_7	a_3a_6	a_3a_5	a_3a_4		a_3						
		a_4a_7	a_4a_6	a_4a_5		a_4								
	a_5a_7	a_5a_6		a_5										
a_6a_7		a_6												
a_7														

Fig. 2. Folded partial products matrix for the A^2 term in the modulo 2^8-1 case.

Section 2. Section 3 presents qualitative and quantitative comparison results. Finally, conclusions are drawn in the last section.

2. MODULO 2^n-1 SQUARERS

In this section we present two architectures for designing modulo 2^n-1 squarers. The first subsection deals with non-encoded squarers whereas the second subsection deals with Booth-encoded squarers.

2.1. Non-encoded modulo 2^n-1 squarers

Let $A = a_{n-1} \cdots a_0$ be the n-bit input operand of the modulo 2^n-1 squarer, with $0 \le A < 2^n - 1$. Then, the output is equal to:

$$\left|A^2\right|_{2^n-1} = \left|A \times A\right|_{2^n-1} = \left|\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i a_j 2^{i+j}\right|_{2^n-1} \qquad (1)$$

For the clarity of presentation, the n=8 case is considered in the following. The derived methodology however can be straightforwardly applied to any other value of n. The partial products matrix required for the squaring term of relation (1), when n=8, is given in Figure 1. Since $a_i a_i = a_i$ and $a_i a_j + a_j a_i = 2a_i a_j$, the matrix of Figure 2 can be used equivalently.

An n-bits wide matrix can then be derived for the modulo squaring operation by repositioning each partial product bit, suppose x, that has a weight larger than 2^{n-1} taking into account that:

$$\left|x2^{i+n}\right|_{2^n-1} = \left|x2^{i}\right|_{2^n-1} \qquad (2)$$

Figure 3 presents the 8-bits wide matrix that is derived as the result of repositioning every x bit that has a weight equal to 2^{i+8}, i=0,...,6, to the column with weight 2^i. The matrix consists of 36 partial product bits and the maximum height among all columns is equal to 6. This is the partial products matrix also used in the architectures of [19] and [22].

2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
	a_3		a_2		a_1		a_0
a_0a_6	a_0a_5	a_0a_4	a_0a_3	a_0a_2	a_0a_1		a_0a_7
a_1a_5	a_1a_4	a_1a_3	a_1a_2			a_1a_7	a_1a_6
a_2a_4	a_2a_3				a_2a_7	a_2a_6	a_2a_5
				a_3a_7	a_3a_6	a_3a_5	a_3a_4
			a_4a_7	a_4a_6	a_4a_5		a_4
		a_5a_7	a_5a_6		a_5		
	a_6a_7		a_6				
	a_7						

Fig. 3. Initial partial products matrix for the modulo 2^8-1 squarer.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
a_2a_3	$\bar{a}_2a_3$	a_1a_2	$\bar{a}_1a_2$	a_0a_1	$\bar{a}_0a_1$	a_0a_7	$a_0\bar{a}_7$
a_0a_6	a_0a_5	a_0a_4	a_0a_3	a_0a_2			
a_1a_5	a_1a_4	a_1a_3				a_1a_7	a_1a_6
a_2a_4					a_2a_7	a_2a_6	a_2a_5
				a_3a_7	a_3a_6	a_3a_5	
			a_4a_7	a_4a_6		a_3a_4	$\bar{a}_3a_4$
		a_5a_7		a_4a_5	$\bar{a}_4a_5$		
		a_5a_6	$\bar{a}_5a_6$				
a_6a_7	$\bar{a}_6a_7$						

Fig. 4. Modified partial products matrix for the modulo 2^8-1 squarer.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$s_{4,3,2}$	$c_{3,2,1}$	$s_{3,2,1}$	$c_{2,1,0}$	$s_{2,1,0}$	$c_{1,0,7}$	$s_{1,0,7}$	$c_{4,3,2}$
$s_{0,7,6}$	$c_{7,6,5}$	$s_{7,6,5}$	$c_{6,5,4}$	$s_{6,5,4}$	$c_{5,4,3}$	$s_{5,4,3}$	$c_{0,7,6}$
a_1a_5	a_0a_5	a_0a_4	a_0a_3	a_3a_7	a_2a_7	a_2a_6	a_1a_6
	a_1a_4		a_4a_7		a_3a_6		a_2a_5

Fig. 5. Final partial products matrix for the modulo 2^8-1 squarer.

Furthermore, by applying [5] that:

$$(a_ia_j + a_i)2^k = (a_ia_j)2^{k+1} + (a_i\bar{a}_j)2^k \qquad (3)$$

n=8 times in the partial product matrix of Figure 3 we derive the partial products matrix of Figure 4, which also has 36 partial product bits, but a reduced maximum height equal to 5. Considering [6] that:

$$(a_i\bar{a}_j)2^{l+1} + (a_ia_k + a_ja_k)2^l = c_{i,j,k}2^{l+1} + s_{i,j,k}2^l \qquad (4)$$

where $c_{i,j,k} = a_i(\bar{a}_j \vee a_k)$ and $s_{i,j,k} = (a_i \oplus a_j)a_k$ (the $\vee$ and $\oplus$ symbols denote logical OR and XOR operations, respectively), we can further reduce the total number of partial product bits. Relation (4) indicates that the addition of three partial product bits in adjacent columns can be replaced by two pre-calculated partial product bits. n=8 such triplets exist in Figure 4. Each one is uniquely shaded in Figure 4. The grouping of bits a_2a_3 and a_2a_4 (a_6a_7 and a_0a_6) that have weights equal to 2^7 with the bit $\bar{a}_3a_4$ ($a_0\bar{a}_7$) that has a weight equal to 2^0 can be justified due to the fact that $|x2^n|_{2^n-1} = |x2^0|_{2^n-1}$. Hence, bit $\bar{a}_3a_4$ ($a_0\bar{a}_7$) can be treated as having weight equal to 2^8 and the $c_{4,3,2}$ ($c_{0,7,6}$) bit resulting from relation (4), which also has a weight equal to 2^8 can be driven back to the column with weight 2^0 due to the same reason.

After the above transformations, the partial products matrix of Figure 5 is attained for $|A^2|_{2^8-1}$. The number of required partial product bits has been reduced from 36 to 28 while in parallel the maximum height of the columns has been reduced from 6 to 4, at the cost of requiring the $s_{i,j,k}$ and $c_{i,j,k}$ bits of relation (4).

According to the previous analysis, the corresponding circuit of a modulo 2^8-1 squarer utilizes: (a) 8 blocks for deriving the $s_{i,j,k}$ and $c_{i,j,k}$ bits of relation (4), (b) some end-around-carry carry save adders, composed of full adders and half adders, in order to reduce the height of each column of Figure 5 to 2, that is, in order to derive two final 8-bit summands, and (c) one parallel modulo 2^8-1 adder (which is equivalent to a 1's complement binary adder) to add the two final summands and derive the result.

2.2. Booth-encoded modulo 2^n-1 squarers

A common method used in multipliers and squarers for reducing the number of partial products is to use the encoding of the modified Booth algorithm. Let $A = a_{n-1}\cdots a_0$ be the n-bit input operand of the modulo 2^n-1 squarer as before. We will concentrate our analysis on even values of n, which is the most common case.

Taking into account that $A = |A|_{2^n-1}$, we have [23] that:

$$A = \left|\sum_{i=0}^{n/2-1}(-2a_{2i+1} + a_{2i} + a_{2i-1})2^{2i}\right|_{2^n-1} = \left|\sum_{i=0}^{n/2-1} A_i 2^{2i}\right|_{2^n-1} \qquad (5)$$

where $A_i = -2a_{2i+1} + a_{2i} + a_{2i-1} \in \{-2,-1,0,+1,+2\}$ and $a_{-1} = a_{n-1}$. Relation (5) indicates that we can partition operand A into 3-bit groups in order to encode A in modulo 2^n-1 arithmetic.

Efficient binary squarers that use a Booth-encoded operand have been recently proposed in [7]. Even greater area and delay efficiency was achieved by a folding technique. Both these techniques are explored in the following for deriving an efficient Booth-folded modulo 2^n-1 squarer architecture. For the clarity of presentation, we again concentrate on the n=8 case, but the generalization is straightforward. When n=8, from (5) we have that:

$$A = \left|\sum_{i=0}^{3} A_i 2^{2i}\right|_{2^8-1} = \left|A_3 2^6 + A_2 2^4 + A_1 2^2 + A_0 2^0\right|_{2^8-1}$$

and

$$|A^2|_{2^8-1} = |A \times A|_{2^8-1} = \left|\left|\sum_{i=0}^{3} A_i 2^{2i}\right|_{2^8-1} \times \left|\sum_{i=0}^{3} A_i 2^{2i}\right|_{2^8-1}\right|_{2^8-1}$$
$$= \left|\sum_{i=0}^{3} A_i 2^{2i} \times \sum_{i=0}^{3} A_i 2^{2i}\right|_{2^8-1} \qquad (6)$$

Thus, the square of A modulo 2^8-1 can be computed using the Booth-encoded parts of A instead of its bits. The resulting partial products matrix is shown in Figure 6 and since $A_iA_j + A_jA_i = 2A_iA_j$, it is equivalent to that of Figure 7. Let $C_i = A_i \times A_i$, $i = 0,\ldots,3$ and $P_i = \sum_{k=i+1}^{3} A_kA_i 2^{2(k-i-1)}$, $i = 0,\ldots 2$ as defined in [7]. Obviously, C_i terms are unsigned numbers. Since they assume only one of the three values {0, 1, 4}, they can be

2^{12}	2^{10}	2^{8}	2^{6}	2^{4}	2^{2}	2^{0}
			A_3A_0	A_2A_0	A_1A_0	A_0A_0
		A_3A_1	A_2A_1	A_1A_1	A_0A_1	
	A_3A_2	A_2A_2	A_1A_2	A_0A_2		
A_3A_3	A_2A_3	A_1A_3	A_0A_3			

Fig. 6. Initial partial products matrix for the Booth-encoded modulo 2^8-1 squarer.

2^{12}	2^{10}	2^{8}	2^{6}	2^{4}	2^{2}	2^{0}
			$2A_3A_0$	$2A_2A_0$	$2A_1A_0$	A_0A_0
		$2A_3A_1$	$2A_2A_1$	A_1A_1		
	$2A_3A_2$	A_2A_2				
A_3A_3						

Fig. 7. Folded partial products matrix for the Booth-encoded modulo 2^8-1 squarer.

2^{14}	2^{13}	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}
					$P_{0,6}$	$P_{0,5}$	$P_{0,4}$	$P_{0,3}$	$P_{0,2}$	$P_{0,1}$	$P_{0,0}$	$C_{0,2}$		$C_{0,0}$
			$P_{1,4}$	$P_{1,3}$	$P_{1,2}$	$P_{1,1}$	$P_{1,0}$	$C_{1,2}$		$C_{1,0}$				
	$P_{2,2}$	$P_{2,1}$	$P_{2,0}$	$C_{2,2}$		$C_{2,0}$								
$C_{3,2}$		$C_{3,0}$												

Fig. 8. Booth-folded partial products matrix for the modulo 2^8-1 squarer.

represented by 3 bits, with the middle bit always equal to 0. On the other hand, the P_i terms represent signed two's complement numbers. They can be easily computed using a simple one's complement circuit as described in [7]. In the case of n=8, P_0, P_1, and P_2 have 7, 5, and 3 bits, respectively. Denoting the j-th bit of C_i and P_i as $C_{i,j}$ and $P_{i,j}$, respectively, Figure 8 presents the bits of the P_i and C_i terms that have to be summed for computing the multiplication term of relation (6).

Starting from the matrix of Figure 8, one has to reposition every bit with weight larger than 2^7 to a column with weight less than or equal to 2^7 for deriving a 8-bits wide matrix for the modulo squaring operation. Since C_2 and C_3 are unsigned numbers, the $C_{2,2}$ and $C_{2,0}$ ($C_{3,2}$ and $C_{3,0}$) partial product bits can be easily moved from columns with weights 2^{10} and 2^8 (2^{14} and 2^{12}) to columns with weights 2^2 and 2^0 (2^6 and 2^4), respectively, due to (2). However, the $P_{i,j}$ partial product bits have to be treated differently since the P_i terms represent signed two's complement numbers. Let $X = x_{k-1} \cdots x_0$ denote a signed two's complement number with k bits (k>n). Obviously, x_{k-1} represents the sign bit of X. Then, it holds that:

$$\left|X\right|_{2^n-1} = \left|-x_{k-1}2^{k-1} + x_{k-2}2^{k-2} + \cdots + x_0 2^0\right|_{2^n-1}$$
$$= \left|-x_{k-1}2^{k-1} + \cdots + (x_{n+1} + x_1)2^1 + (x_n + x_0)2^0\right|_{2^n-1}$$

and thus, for computing $\left|X\right|_{2^n-1}$, we have to: (a) move all bits of X with weights larger than 2^{n-1} (except the sign bit) to the corresponding columns with weights less than or equal to 2^{n-1}, according to relation (2), and (b) add an n-bits correction term equal to $1\ldots1\overline{x}_{k-1}1\ldots1$, where $\overline{x}_{k-1}$ has a weight equal to $2^{|k-1|_n}$. This correction term can be justified as follows: If $X \geq 0$, then x_{k-1}=0 and $\left|-x_{k-1}2^{k-1}\right|_{2^n-1} = 0 = \left|2^n - 1\right|_{2^n-1}$. If $X < 0$, then x_{k-1}=1 and $\left|-x_{k-1}2^{k-1}\right|_{2^n-1} = \left|-2^{|k-1|_n}\right|_{2^n-1} = \left|2^n - 1 - 2^{|k-1|_n}\right|_{2^n-1}$.

2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
	$C_{1,2}$		$C_{1,0}$		$C_{0,2}$		$C_{0,0}$
	$C_{3,2}$		$C_{3,0}$		$C_{2,2}$		$C_{2,0}$
$P_{0,4}$	$P_{0,3}$	$P_{0,2}$	$P_{0,1}$	$P_{0,0}$			$P_{0,5}$
$P_{1,0}$					$P_{1,3}$	$P_{1,2}$	$P_{1,1}$
			$P_{2,1}$	$P_{2,0}$			
1	1	$\overline{P}_{2,2}$	1	$\overline{P}_{1,4}$	1	$\overline{P}_{0,6}$	1

Fig. 9. Final Booth-folded partial products matrix for the modulo 2^8-1 squarer.

Based on the previous analysis, in order to reposition all P_i terms of Figure 8, we have to: (a) move all bits of the P_i terms with weights larger than 2^7 (except their sign bits) to the corresponding columns with weights less than or equal to 2^7, and (b) add an n-bits correction term to the partial products matrix for every P_i term. However, it can be easily proven that the three required correction terms can be merged to a single n-bits correction term equal to $11\overline{P}_{2,2}1\overline{P}_{1,4}1\overline{P}_{0,6}1$.

These transformations provide us with the partial products matrix of Figure 9 for computing $\left|A^2\right|_{2^8-1}$. We observe that there are 28 partial product bits, 5 of them having a constant value equal to 1. We also observe that the maximum height among all columns is equal to 5 and resides in columns with weights 2^0 and 2^4. However, both these columns also have a constant partial product bit, hence simplifications can be made at end-around-carry carry save adders that will add the partial products.

3. EVALUATION AND COMPARISONS

In this section we evaluate the architectures that were proposed in the previous section and compare them against previously proposed architectures. We compare the proposed architectures against the architecture of [19] [22] for designing non-encoded modulo 2^n-1 squarers and the

TABLE I
NUMBER OF PARTIAL PRODUCT BITS

n	*[19][22]*	*Proposed Non-encoded*	*[23]*	*Proposed Booth*
4	10	8	8	10
8	36	28	32	28
16	136	120	128	88
32	528	496	512	304
64	2080	2016	2048	1120
128	8256	8128	8192	4288

TABLE II
MAXIMUM HEIGHT

n	*[19][22]*	*Proposed Non-encoded*	*[23]*	*Proposed Booth*
4	4	2	2	4
8	6	4	4	5
16	10	8	8	7
32	18	16	16	11
64	34	32	32	19
128	66	64	64	35

architecture of [23] for Booth-encoded modulo 2^n-1 multipliers, for which it is assumed that both inputs are driven by the same operand. It should be noted that no other Booth-encoded modulo 2^n-1 squarer architecture has been reported in the open literature.

All four different architectures derive a matrix of partial product bits that are then reduced to two *n*-bit final summands, using end-around-carry carry save adders that are composed of full adders and/or half adders, and utilize a parallel modulo 2^n-1 adder that accepts the two *n*-bit summands and produces the correct result.

Table I presents the number of partial product bits that are derived by each architecture for 6 different values of *n* (*n*=4, 8, 16, 32, 64, and 128). The number of partial product bits provides a rough estimate of the area required by the end-around-carry carry save adders. Table I reveals that the proposed architecture for non-encoded modulo 2^n-1 squarers leads to less partial product bits compared to the architecture of [19] [22]. The proposed architecture for Booth-folded modulo 2^n-1 squarers also leads to less partial product bits compared to the architecture of [23] for all examined *n* values but the smallest. We therefore expect that the area requirements of the proposed circuits will be less than those of the previously proposed.

In Table II we present the maximum height of the partial products matrix derived in each architecture. This maximum height provides a rough estimate of the delay of each circuit. We can see that the maximum heights of the proposed non-encoded modulo squarers are smaller by 2 than those of [19] [22]. However, the proposed circuits have to wait for the calculation of the $c_{i,j,k}$ and $s_{i,j,k}$ partial product bits of relation (4). It is therefore expected that both the proposed circuits and those of [19] [22] will have comparable delays. We can also see that the proposed Booth-folded modulo squarers have smaller maximum heights than those of [23], for medium and large values of *n*. This is justified by the fact that [23] requires *n*/2 partial products whereas the proposed Booth-folded modulo squarers only *n*/4+3 partial products. It is therefore expected that, for medium and large values of *n*, the proposed Booth-folded modulo squarers will be faster than those of [23].

For obtaining realistic area and delay estimates, we described in HDL the modulo 2^n-1 squarers that result from every architecture for 4 different values of *n* (*n*=4, 8, 16, and 32). The same final modulo 2^n-1 adder [24] was used in all descriptions of the same size. Each description was thoroughly verified and then synthesized and mapped to a 90 nm CMOS technology, using a commercial synthesis tool and assuming typical parameters. The netlists were then optimized for area and delay using a standard optimization script. Finally, the area and delay estimates listed in Table III were attained.

As expected, the proposed non-encoded modulo 2^n-1 squarers offer comparable delay with that of the circuits of [19] [22] within less implementation area. In the modulo 2^4-1 squarer case, significant reduction in delay is also observed. This is due to the fact that in this case the proposed squarer does not require any partial product bit reduction whereas the squarer of [19] [22] has to use full adders in order to derive the inputs of the final modulo 2^4-1 adder. The estimates further indicate that the proposed Booth-folded modulo 2^n-1 squarers require siginificantly less area than those based on [23] while also being equally fast for small *n* and significantly faster for larger values of *n*. Finally, we can observe that, for small values of *n*, the proposed non-encoded architecture for modulo 2^n-1 squarers leads to smaller and faster circuits, while the Booth-encoded one is the best choice for modulo squarers with medium or large values of *n*.

TABLE III
AREA AND DELAY ESTIMATES OF MODULO 2^N-1 SQUARERS

	Non-encoded Squarers				*Booth-encoded Squarers*			
	[19] [22]		*Proposed*		[23]		*Proposed*	
n	*Area* (μm^2)	*Delay* (*ns*)	*Area* (μm^2)	*Delay* (*ns*)	*Area* (μm^2)	*Delay* (*ns*)	*Area* (μm^2)	*Delay* (*ns*)
4	821	0.35	767	0.27	1993	0.39	1338	0.40
8	4346	0.54	4104	0.53	5598	0.68	4552	0.57
16	19614	0.83	18453	0.84	22487	1.01	16600	0.81
32	78281	1.14	76303	1.14	88358	1.33	61138	1.06

4. CONCLUSIONS

An RNS that uses one or more moduli of the 2^n-1 form is often adopted in digital signal processing applications. Such applications often require a lot of squaring operations that profit form the existence of a dedicated squarer circuit.

To this end, two architectures were investigated in this paper for the design of modulo 2^n-1 squarers. The proposed non-encoded modulo 2^n-1 squarers offers similar execution rates with those of [19] [22] with reduced implementation area and are more suitable for small values of n. The proposed Booth-folded modulo 2^n-1 squarers, on the other hand, are more efficient, in terms of area and delay, than the modulo squarers that are based on [23] and are more suitable for medium and large values of n.

REFERENCES

[1] R. Jain, A. Madisetti, and R. L. Baker, "An integrated circuit design for pruned tree-search vector quantization encoding with an off-chip controller," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 147-158, 1992.

[2] Y. Fengqi, and A. N. Wilson, "Multirate digital squarer architectures," in *Proc. IEEE Int. Conference on Electronics, Circuits and Systems*, pp. 177-180, 2001.

[3] J. G. Proakis, *Digital Communication*, McGraw-Hill, 1995.

[4] T. C. Chen, "A binary multiplication scheme based on squaring," *IEEE Transactions on Computers*, vol. 20, no. 4, pp. 678-680, 1971.

[5] R. K. Karagotla, W. R. Griesbach, and H. T. Srinivas, "VLSI implementation of 350MHz 0.35μm 8 bit merged squarer," *Electronics Letters*, vol. 34, no 1, pp. 47-48, 1998.

[6] K.-J. Cho, and J.-G. Chung, "Parallel squarer design using pre-calculated sums of partial products," *Electronics Letters*, vol. 43, no 25, pp. 1414-1416, 2007.

[7] A. Strollo, and D. Caro, "Booth folding encoding for high performance squarer circuits," *IEEE Transactions on Circuits and Systems II*, vol. 50, no. 5, pp. 250-254, 2003.

[8] P. V. Ananda Mohan, *Residue Number Systems: Algorithms and Architectures*, Kluwer Academic Publishers, 2002.

[9] A. Omondi, and B. Premkumar, *Residue number systems: theory and implementation*, Imperial College Press, 2007.

[10] R. Chaves, and L. Sousa, "RDSP: A RISC DSP based on residue number system", in *Proc. Euromicro Symposium on Digital System Design*, pp. 128–135, 2003.

[11] J. Ramirez, A. Garcia, S. Lopez-Buedo, and A. Lloris, "RNS-enabled digital signal processor design," *Electronics Letters*, vol. 38, no 6, pp. 266–268, 2002.

[12] G. Cardarilli, A. Nannarelli, and M. Re, "Reducing power dissipation in FIR filters using the residue number system," in *Proc. IEEE Midwest Symposium on Circuits and Systems*, pp. 320-323, 2000.

[13] J. Ramirez, and U. Meyer-Baese, "High performance, reduced complexity programmable RNS-FPL merged FIR filters," *Electronics Letters*, vol. 38, no. 4, pp. 199-200, 2002.

[14] Y. Liu, and E. Lai, "Moduli set selection and cost estimation for RNS-based FIR filter and filter bank design", *Design Automation for Embedded Systems*, vol. 9, no. 2, pp. 123-139, 2004.

[15] P. G. Fernandez, and A. Lloris, "RNS-based implementation of 8x8 point 2D-DCT over field-programmable devices," *Electronics Letters*, vol. 39, no 1, pp. 21-23, 2003.

[16] U. Meyer-Baese, A. Garcia, and F. Taylor, "Implementation of a communications channelizer using FPGAs and RNS arithmetic," *Journal of VLSI Signal Processing*, vol. 28, no. 1-2, pp. 115-128, 2001.

[17] J. Ramirez, A. Garcia, U. Meyer-Baese, and A. Lloris, "Fast RNS FPL-based communications receiver design and implementation," in *Proc. Int. Conference on Field Programmable Logic*, pp. 472-481, 2002.

[18] M. Panella, and G. Martinelli, "An RNS Architecture for Quasi-Chaotic Oscillators," *Journal of VLSI Signal Processing*, vol. 33, no. 1-2, pp. 199-220, 2003.

[19] S. Piestrak, "Design of squarers modulo A with low-level pipelining," *IEEE Transactions on Circuits and Systems II*, vol. 49, no 1, pp. 31-41, 2002.

[20] H. T. Vergos, and C. Efstathiou, "Efficient modulo 2^k+1 squarers," in *Proc. Design of Circuits and Integrated Systems*, 2006.

[21] H. T. Vergos, and C. Efstathiou, "Diminished-1 modulo 2^n+1 squarer design," *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no 5, pp. 561-566, 2005.

[22] B. Cao, T. Srikanthan, C.-H. Chang, "A new design method to modulo 2^n-1 squaring," in. *Proc. Int. Symposium on Circuits and Systems*, pp. 664-667, 2005.

[23] C. Efstathiou, H. T. Vergos, and D. Nikolos, "Modified Booth Modulo 2^n-1 Multipliers," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 370-374, 2004.

[24] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. T. Vergos, and J. Kalamatianos, "High-Speed Parallel Prefix Modulo 2^n-1 Adders," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 673-680, 2000.