A Novel Low Complexity Combinational RNS Multiplier Using Parallel Prefix Adder

Mohammad R. Reshadinezhad[1], **Farshad Kabiri Samani**[2]

[1] **Department of Computer engineering, University of Isfahan, Isfahan, Isfahan 8174673440, Iran**

[2] **Department of Electrical and Computer engineering, Lenjan Branch, Islamic Azad University, Isfahan, Iran**

Abstract

Modular multiplication plays an important role in encryption. One of the encryption methods which need fast modular multiplication is RSA where large numbers are needed to empower large modules. In such methods, in order to show numbers, RNS is usually used with multiplication as the core. Modulo 2^n+1 multipliers are the primitive computational logic components widely used in residue arithmetic, digital signal processing, fault-tolerant and cryptography. Here, two residue number system multipliers are introduced, both based on classifications of couples or triplets of input operands, which results in a low complexity RNS multiplier. The first modular multiplier is a combinational circuit which enables parallel prefix adder application in modulo 2^n+1. The second modulo 2^n+1 multiplier uses n+1 partial product, each with \$n\$ bit width, constructed by utilizing an inverted end-around-carry, carry save adder (CSA) tree and a parallel adder at the end. The performance and efficiently of the proposed multipliers are evaluated and compared with that of the earlier fastest modulo 2^n+1 multipliers. The proposed multipliers are considerably faster and more compact than that of the hardware implementations, which make them a viable option for efficient designs.

Keywords: *Modular multiplier, residue number systems (RNS), modulo 2^n+1 multiplier, parallel prefix adders, low complexity RNS multiplier.*

1. Introduction

A Residue number system is a non-weight numeric system [1] which has gained importance during the last decade, because some of the mathematical operations can be divided into categories of sub-operations based on RNS [2]. Addition, subtraction and multiplication are performed in parallel on the residues in distinct design units (often called channels), avoiding carry propagation among residues [3]. Therefore, arithmetic operations such as, addition, subtraction and multiplication can be carried out more efficiently in RNS than in conventional two's complement systems. That makes RNS a good candidate for implementing variety of applications [4] such as: digital Signal Processing (DSP) for filtering, convolutions, FFT computation [5] [6], fault-tolerant computer systems [1] [6] [7], communication [8] and cryptography [9].

A residue number system is a represented by k integer modules $m_{k-1}, \dots, m_1\ m_0$. An integer encrypts a number in a remainders set with respect to the prime numbers module. An integer variable $x \in [\alpha, \beta]$ is uniquely represented by $(x_{k-1}, \dots, x_1, x_0)$ where, $x_i = x \bmod m_i$ for $0 \le i \le k-1$, and $[\alpha,\beta]$ is the dynamic range of residue number, the cardinality of which is $M=\beta-\alpha+1$. The modules are chosen to be pair-wise prime to each other in order to maximize the cardinality, such that $M= m_{k-1}, \dots, m_1, m_0$. In RNS, mathematical operations are performed on some small integers concurrently. Here an obvious advantage is that the carry does not pass through modules. To compute a mathematical operation on x and y in residue number system, we use a notation $x \odot y$, where $\odot$ could be one of the operations like, addition, subtraction or multiplication represented by Eq. (1):

$$\begin{gathered} x \odot y = (x_{k-1}, \cdots, x_1, x_0) \odot (y_{k-1}, \cdots, y_1, y_0) \\ z = (z_{k-1}, \cdots, z_1, z_0) \text{ and, } z_i = |(x_i \odot y_i)|_{m_i} \end{gathered} \qquad (1)$$

One of the easiest ways to perform residue multiplication is to use a pure table look-up structure where, each modulo m_i multiplier can be implemented through ROM's. Designs adopting ROM approaches are based on look up table, which are excellent for smaller modules multipliers; however, they occupy vast area as the number of bits gets larger, therefore, the ROM size increases exponentially as the number of bits in each operand increases [10] [11] [12] [13].

Another approach in designing RNS multipliers is to use adder-based multiplier architectures, which have been proposed in [14] [15] [16] [17]. Hiasat introduced an efficient modular multiplier which is suitable for different module sizes, medium and large size module particularly [14]. DiClaudio *et al.* presented the pseudo-RNS representation [15]. Wrzyszcz *et al.* have introduced an architecture tailored to fixed coefficient [16]. Stouraitis *et al.* [17] presented full adder based single modulus architecture for RNS multiplier and accumulator. The multipliers presented by Zimmerman [24] allow the use of Wallac-tree addition and Booth recoding of partial products for speed-up. They also used parallel prefix adders to implement their fast and simple end-around carry adder for modulo $2^n \pm 1$. The diminished-1 modulo 2^n+1 multiplier proposed by Efstathiou *et al.* [25] uses an n×n partial-product array together with a CSA tree. They needed n+3 partial products and treatment of zero operands was not discussed. Vergos *et al.* [26] introduced a new modulo 2^n+1 multiplier architecture for operands in weighted representation. Their proposed multiplier utilizes a total of n+1 partial product. They used an inverted end-around-carry (ECA), carry-save adder tree and a final parallel adder. In 2007 another RNS multiplier by Vergos *et al.* was introduced. The analytical and experimental results presented in [27] show that the multipliers proposed, outperform the earlier solutions of [24] [25] [26].

In this article a background on RNS multipliers is reviewed in section 2. In section 3, the bases of RNS adders used in RNS multiplier architecture are discussed [3] [14] [15] [16] [17] [18]. The proposed multiplier architectures are presented in section 4. Subsequently, in section 5, the obtained operational result is compared to the available results, and finally the conclusions are drawn in section 6.

2. Background of RNS Multipliers

As mentioned in the introduction, a variety of RNS multipliers are designed, like table look-up multipliers, index transform multipliers, quarter square multipliers, and array multipliers. Each modulo m_i multiplier can be implemented by ROM's, which employs look-up table and is excellent for smaller modulo multiplier, or an array of full adders cells is used to implement the multiplier, which has a linear delay in respect to the number of bits in each operand. The concepts of RNS multiplication is to evaluate $\langle x_i \times y_i \rangle_{m_i}$ and eliminate the *i* subscript in the analysis and interpret each residue bit in its binary form, through expressions for X and Y as equations:

$$X = \sum x_i 2^i \quad and, \; Y = \sum y_i 2^i \tag{2}$$

The modular multiplication of numbers X and Y can be carried out through equation (3).

$$R = \langle x \times y \rangle_M = \left\langle \sum\sum x_i y_j \left\langle 2^{(i+j)} \right\rangle_M \right\rangle_M \tag{3}$$

where, X, Y, and R are n-bit residues modulo M, and $x_i, y_j \in \{0,1\}$ represents the i_{th} and j_{th} bit of X and Y, respectively. In this article a novel VLSI implementation for calculation of residue multiplication is introduced that allow the computation of the result by equation (4).

$$R_i = \langle X_i \times Y_i \rangle_{m_i} \tag{4}$$

Modular multiplications are categorized into two methods. In the First method, the multiplication of operands takes place completely and then a reduction takes place on the final result. This method is called Reduction after multiplication (RAM). In the second method, the reduction is applied during the multiplication steps, and is called Reduction During Multiplication (RDM). Both methods can be divided into three different classes based on the hardware implementation method used.

Class One: these methods use some popular modules like: 2^n, 2^n+1, or 2^n-1 where n is either small or big. To implement these kinds of methods, logical circuits are often used instead of ROMs. As a result, these methods are limited to specific modules [3] [11] [14] [19].

Class Two: these kinds of Implementations are capable of working with any modules but there backbone structure is based on ROM and usually the logical circuits are not applied in such methods. The memory size increases rapidly with an increase in n. Consequently, this method is not a good choice for large modules [10] [12] [13].

Class three: these types of architectures are employed for average and large modules, which are usually hybrid and use binary adders and multipliers together with some ROMs in small size and logical elements [14].

In this article, we focus on class one and chose modulo 2^n+1 to do the multiplication and reduction. The multiplier inputs are categorized as that of the Vassilis *et al.* paper [11]. To get the final result we use adder introduced in [18] in order to optimize time and area.

2.1 Vassilis's Multiplier

The organization of this part is based on the residue computing units of the previous reports [11] [17] [20] [21]. Let us consider a modulo-5 multiplier with the bit length of 3 using equation (3). According to the sources above, the first stage of the residue multiplier represents the nested addition of equation (3), whereas the second and the third part, is the reduction which takes place on the outer module of the right side of equation (3). Specifically, the second stage alternatively converts its input to a number with a residue the same as modulo, *m*, that of 5, and word length of *n=3*. Finally, in the last stage, maps its inputs to find residue via a conditional addition; therefore, the third stage is implemented through equation (5):

$$Y = \begin{cases} Y' & if \quad Y' < m \\ Y' - m & if \quad Y' \geq m \end{cases} \qquad (5)$$

The first main condition in each step can be presented with a set of $Z_{k,i}$ from input bits which participate in the i^{th} output bit of a stage or of the cascaded part of the stage, called recursions [12]. Here, k=0 shows that the set $Z_{k=0,i}$ is related to the first stage of equation (3), while $0< k \leq r$ points to the k^{th} recursion of the second stage as is evident by (3). The outputs bits which are the result of multiplication of x_i and y_j can be identified by writing $\langle 2^{i+j} \rangle_5$ in binary format as it is shown by (3). For instance, for i=1 and j=2, we have:

$$\langle 2^{i+j} \rangle_5 = \langle 2^3 \rangle_5 = (011)_2 = 2^1 + 2^0 \qquad (6)$$

As indicated in [11], $\langle x \rangle_2$ implies that x is a binary number, and therefore, because of equation (6), the bit multiplication of x_1y_2 points to the output bits of weighted positions 2^1 and 2^0, as shown in equation (7).

$$x_1 y_2 \langle 2^{i+j} \rangle_5 = x_1 y_2 2^1 + x_1 y_2 2^0 \qquad (7)$$

Consequently, it can be stated that x_1y_2 are elements of, $Z_{0,0}$, and $Z_{0,1}$. This technique is used to compute all $Z_{k,i}$ for each of recursions in the second stage. Another way to describe each recursion in multiplication is the use of flag bit chain ($q_{i,j,k}$), where, $q_{i,j,k}$ є{0,1}, and each $q_{i,j,k}$ express whether the j^{th} input bit participates in the result of i^{th} bit or not during the k^{th} recursion [21]. Then, $Z_{k,,i}$ based description is identical to a flag bit sequence, since

$$Z_{k,i} = \{ y_{k,j} \mid q_{i,j,k} = 1 \} \qquad (8)$$

where, $y_{k,j}$ is the input bit of k^{th} recursion of weight $\langle 2^j \rangle_m$ replaces $y_{k,j}$ in equation (8). By defining the $Z_{k,i}$ or the flag bit sequence, an architecture including one bit full adders (FAs) and half adders (HAs) can be implemented using the methodology used by Vassilis *et al.* [11], knowing sets of $Z_{k,i}$ or the flag bit chain.

The multiplier architecture introduced by Vassilis *et al.*, was organized in three stages: in the first and second stages they used carry-saved array, and each stage had columns of FAs, HAs, and OR gates. They introduced a hardware reduction procedure in a way that it reduces the number of 1-bit adders in a column by OR gates. First, they contemplated the recursion of the second stage such that equation (9) is an input to the k^{th} recursion and n_k represents the word length of the maximum value of Y_k, which is calculated through simulation. There exist the couples (y_{k,j^1},y_{k,j^2})

$$Y_k = \sum_{j=0}^{n_k - 1} y_{k,j} 2^j \qquad (9)$$

or triplets ($y_{k,j1}$, $y_{k,j2}$,$y_{k,j3}$) from input bits of $y_{k,j}$ є{0,1}, such that $0 < j_1, j_2, j_3 < n_k$, is assigned to the i^{th} column, i.e., $q_{j1,i,k} = q_{j2,i,k} = 1$ or $q_{j1,i,k} = q_{j2,i,k} = q_{j3,i,k} = 1$, the summation of which does not generate a carry for any Y_k є I_k={0,1,…,$Y_{max,k-1}$}. Said otherwise, for the triplets it holds

$$y_{k,j_1} + y_{k,j_2} + y_{k,j_3} \leq 1 \; \forall \; Y_k \in I_k \qquad (10)$$

whereas, for the couples it is:

$$y_{k,j_1} + y_{k,j_2} \leq 1 \; \forall \; Y_k \in I_k \qquad (11)$$

where, I_k is the set of valid input values.

In order to minimize the number of bits added to a column, the largest number of separate couples or triplets, produced by bits composed in each of the $Z_{k,i}$ set, that satisfy the terms of equations (10) and (11), respectively. The number of bits to be added in a column is minimized when the number of bits included in a specific couples or triplets is maximized. Thus, the input bits are grouped into appropriate sets of couples and triplets by dictating equations (10) and (11). The design procedure introduced by [11] uses equation (10), (11), and the couples (*x,y*) or triplets (*x,y,z*) of *x,y,z* є $Z_{k,i}$ and categorize them into sets of $C_i^{(t)}$ and $C_i^{(d)}$. Also, bits of $Z_{k,i}$ are categorized into set of disjoint triplets or couples, for each column *i*, called $C_i^{'}$, and a set such that collects the remainder of the bits that are not member of any of the triples or couples, called C_i^f. From the number of possible $C_i^{'}$ and C_i^f sets that can shape the sets of $C_{i,min}^{'}$ and $C_{i,min}^f$, and minimize the summation result S_i^c of input bits added in the i^{th} column given as

$$S_i^c = \left\| C_i^{'} \right\| + \left\| C_i^f \right\| \qquad (12)$$

are pinpointed and ventured in the derivation of the architecture of residue multiplier [11].

In order to clarify the design procedure the authors gave an example of a modulo-5 residue multiplier, Fig. (1). The multiplication operands were chosen to be A and B, and the digit positions to which the input bit product a_ib_j contribute, are shown in Table (1). Using the third column of this table, the corresponding $Z_{k=0,i}$ sets are configured and depicted in Table (2). In the first step of the proposed multiplier, by knowing the content of $Z_{0,i}$, the $C_i^{(t)}$ and $C_i^{(d)}$ using corresponding equations in [11] are determined.

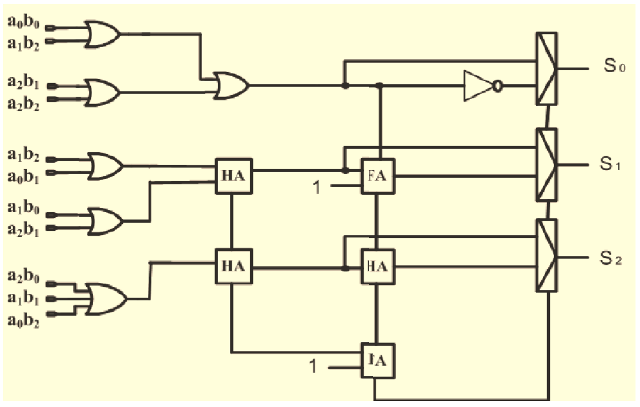


Fig. 1. Modulo-5 residue multiplier [11]

The second step in this modular design is to construct the sets of $C_i^{'}$ and C_i^{f} shown in [11] for corresponding column. The final step is to select $C_{2,min}^{'}$ and $C_{2,min}^{f}$ among different sets of $C_2^{'}$ and C_2^{f} . For example, in column $i$=2, the number of bits to be processed is determined by $C_2^{'}=(a_0b_2,a_1b_1,a_2b_0)$, and $C_{2,min}^{f}=0$, where sets $S_2^{c}=1$.

Table 1: Sets of $Z_{k=0,i}$, for $i$=0,1,2,... for a modulo-5 residue multiplier [11]

$i$	j	$\left\langle 2^{i+j} \right\rangle_5$	$(\left\langle 2^{i+j} \right\rangle_5)_2$
0	0	1	001
0	1	2	010
0	2	4	100
1	0	2	010
1	1	4	100
1	2	3	011
2	0	4	100
2	1	3	011
2	2	1	001

Table 2: Input assignment to digital positions for a modulo-5 residue multiplier [11].

$Z_{0,2}$	$Z_{0,1}$	$Z_{0,0}$
a_0b_2	a_0b_1	a_0b_0
a_1b_1	a_1b_0	a_1b_2
a_2b_0	a_1b_2	a_2b_1
	a_2b_1	a_2b_2

The organization of each column is defined by creating sets of $C_{i,min}^{'}$ and $C_{i,min}^{f}$. The bits added at the $i$th column are the carry bits from column ($i$-1)th column according to the carry-save paradigm. Input bits of set $C_{i,min}^{'}$, and bits created by two or three input OR gates. According to definition in [11], an OR gate is sufficient to add the bits of the couples or triplets of $C_{i,min}^{'}$; hence, the FAs or HAs can be replaced by OR gates. This is feasible since two or more bits cannot be defined concurrently and because, carry generation is not required and is eliminated. For more details refer to [11].

3. Parallel Prefix Adder

The modulo 2^n+1 addition is computed by $A=X+Y+2^n-1$; where, X and Y are $n+1$ bit operands in the range of $[0, 2^n]$, such that, $A= a_n\, a_{n-1} \dots a_1\, a_0$, $X= x_n\, x_{n-1} \dots x_1\, x_0$, and $Y=y_ny_{n-1} \dots y_1\, y_0$, according to [18] [22] and R is defined as $(r_n\, r_{n-1} \dots r_1\, r_0)$ and is computed by $|X+Y|_{2^n+1}$; where, $A=2^n+1$. The computation is made by equation (13):

$$R=|a_na_{n-1}\cdots a_1a_0+(2^n+1)\overline{a_{n+1}}|_{2^n+1} \qquad (13)$$

and to calculate R equation (14) is used.

$$A=X+Y+2^n-1=S+C \qquad (14)$$

In equation (14) S is the summation of bits in $i$th column and C is the abbreviation for carry out of column $i$-1. This addition is illustrated in Fig. (2).

	x_n	x_{n-1}	…	…	x_1	x_0
	y_n	y_{n-1}	…	…	y_1	y_0
		1	…	…	1	1
	s_n	s_{n-1}	…	…	s_1	s_0
c_n	c_{n-1}	…	…	c_1	c_0	
a_{n-1}	a_n	a_{n-1}	…	a_2	a_1	a_0

Fig. 2 Calculation of $A=X+Y+2^n-1$.

As it is seen in this Fig. (2), the summation of bits in each column is computed by $s_i=x_i\oplus y_i$, except the nth column which is given by equation $s_n=x_n\oplus y_n$. Carry generation due to the addition of each column is transferred to next column and is equal to $c_i=x_i+y_i$ and also carry transfer from nth column is computed by replacing index $i$ by n in the last equation. To calculate the final result R, the following terms have to be calculated.

$$r_0=a_0\oplus\overline{a_{n+1}}=s_0\oplus\overline{c_n+G_{n,1}}$$

where $G_{n,1}$ is the carry out of nth column while calculating A. For $1\le i\le n-1$, r_i and r_n are equated by:

$$r_i = a_i \oplus G^*_{i-1,1} = s_i \oplus c_{i-1} \oplus G^*_{i-1,1}$$

$$r_n = s_n + c_{n-1} + \overline{a_{n+1}} + G^*_{n-1,1}$$

where, $G^*_{i-1,1}$ is the carry into position i, in the addition $a_{n-1}a_{n-2}\cdots a_1a_0 + c_n + G_{n,1}$. The *n*-bit addition in Fig. (2) where *S* is being added to *C* can be computed using any carry-accelerate adder scheme like: carry look ahead adders [23], parallel prefix adders, end-round-carry, *etc*. [3] [18] [24] [25] [26] [27]. Technically, we can postpone the calculation of C_{in} (carry that enter into last significant bit position) to the final (last) stage of positional carry triggering. So, it is prevented from one new precursor calculation of carry. C_i is the carry going to position *i* when $C_0 = C_{in} = 0$, and *P* and *G* are variables that represent, generate and propagate expressions based on position of generation and propagation signal. The final carry into i^{th} position is used in calculating $s_i = p_i + c^*_i$. The two variables P_i and G_i are depend on all the g_j and p_{j-1} signals where ($0 \leq j \leq i$). These recursive computations can be evaluated through precursor carry look-ahead operations cells or parallels prefix adders [23] as:

$$G_i = g_{i-1} + P_{i-1}G_{i-2}, \quad g_i = \overline{x_i \oplus y_i}(x_{i-1} + y_{i-1})$$

$$P_i = P_{i-1}p_{i-1}, \quad \underline{P_i} = \overline{x_i \oplus y_i} + (x_{i-1} + y_{i-1})$$

The end-around-carry is $a_{n+1} = G_n + x_n y_n$, where both G_n and $x_n y_n$ cannot be equal to one. Computation of c^*_i and s_i can be obtained directly by $s_i = p_i \oplus c^*_i$ and $a_i = p_i \oplus c_i$. Hence, it is not necessary to calculate $s_n = p_n \oplus c^*_i$ and $s_n = a_{n+1} + a_n$ [18]. Efestathiou's adder is one of the fastest modulo addition introduced so far. This adder is shown in Fig. (3) which can be used to perform the modular multiplication of our concern. Refer to [18] for more detail derivations of the parallel prefix adder. In next section the novel modulo 2^n+1 residue multiplier is introduced.

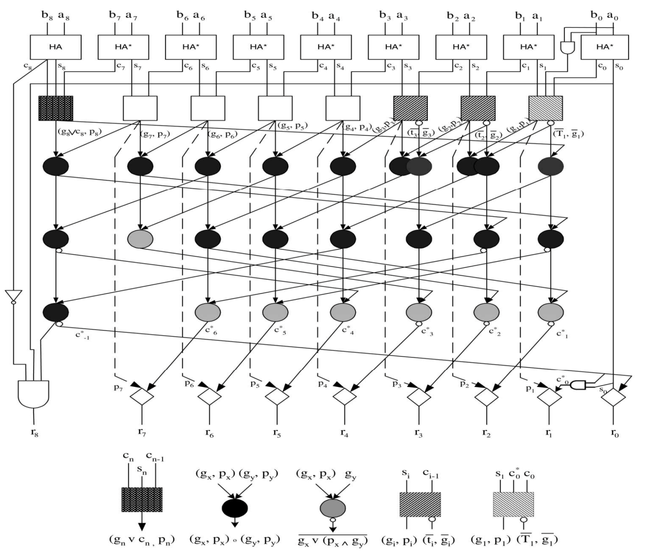


Fig. 3 Parallel prefix adder [18].

Multiplications in higher modules, based on reduction of partial products in a given module. In this proposed architecture the reduction of partial products are

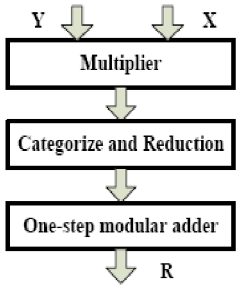


Fig. 4 General block diagram of proposed RNS multiplier.

according to definitions in [11]. Hence, the XOR gates in [26] and OR gates in [27], conventionally employed to generate sum digit are replaced by wired or gates which have no delays. The gates shown by a dot inside them are considered as wired OR gate. Wired OR gate is a gate where only and only one of its inputs has a value of one and the rest are zeros. In the second part we use carry save adders and finally an end-around-carry parallel prefix adder is used. This approach is referred to as design method II.

4. Proposed Modulo 2^n+1 Multiplication

A general block diagram for modulo *M* multiplier is illustrated in Fig. (4). The first block is a partial product generator. The second block refers to pre-processing of partial products according to [11]. The third block represents an *n*-bit binary long adder, using one of the adders like ripple-carry adder, carry accelerate adder, parallel prefix adder, carry skip adder, end-around-carry adder, and etc. [3] [18] [24] [27].
Two approaches can be adopted for the proposed multiplier architecture. The first approach is applied in lower modules like 5, 7 and 9, where by using truth table, a simplified design is obtained. This approach is called design method I and is presented in details in the following section. The second approach is for preforming

4.1 Design Method I

Design method I is suitable for lower modules like modulo 5, 7 and 9. The following new modulo 2^n+1 multiplier is a twofold structure formed through a low complexity combinational RNS multiplier [11], reduction combinational circuit and fast parallel prefix 2^n+1 adders [10] [17] [18]. The main concern in here is to figure out a

way to use parallel prefix structure in order to add up the outputs of the first stage of residue binary number multiplier of Fig. (1). Using modulo-5 for illustrating modulo multiplier by [11] is used in this study. Considering that the operands *A* and *B* are the inputs to the first block of the modular multiplier, the partial products are classified into couple and triplets. The classification of couples and triplets are managed in a way that the HAs and FAs in the *i*th column are replaced by wired OR gates instead of conventional OR gates which were used in [11]. Consequently, the first stage of this newly proposed multiplier is same as [11] and it is illustrated in Fig. (5). In order to obtain desired result, the outputs of the first stage must be in the range of modulo-5 in order to use in the Efstathiou's parallel prefix adder. Unfortunately, the results are not in modulo range. In order to use parallel prefix adder instead of the adder used in [11], the outputs of Fig. (5) have to be converted into modulo range. Therefore, for all the combinations of inputs, the input-output truth table shown in Table (3) is tabulated.
According to Table (3), for only three different input sets the outputs are not in the residue range and exceed modulo-5. The corresponding combinations of inputs are shown in bold faces in Table (3).

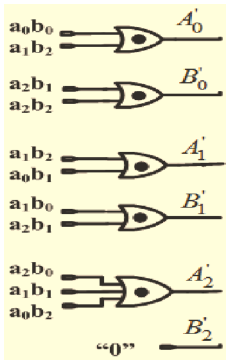


Fig. 5 First stage of RNS multiplier in modulo-5.

The identification of these states contribute to the proposed combinational circuit which makes the use of parallel prefix adder lieu of adders used in [11] possible. The combinational designed circuit is the second stage of RNS multiplier used in proposed design (see Fig. (6)). This combinational circuit is designed to keep all the sets of input values in modulo range.
According to the truth table shown in Table (3), the outputs of all the input sets for $A_2A_1A_0$ and $B_2B_1B_0$ are within the modulo range. After adding the second stage to the first that, the outputs will never go beyond the residue range is assured. Therefore, the Efstathiou's parallel prefix adder is used to perform addition and produce the final result of the RNS multiplier (see Fig. (7)).

As an example, it is assumed that $a_2a_1a_0$ and $b_2b_1b_0$ in Fig. (7) are 011 and 100, respectively. The outputs $A'_2A'_1A'_0$ and $B'_2B'_1B'_0$ of the first stage for these sets of inputs are equal to 111 and 000, accordingly.

Table 3: Truth table for possible combination of inputs in modulo-5

$a_2a_1a_0$	$b_2b_1b_0$	$A'_2A'_1A'_0$	$B'_2B'_1B'_0$	$A_2A_1A_0$	$B_2B_1B_0$
0 0 0	x x x	0 0 0	0 0 0	0 0 0	0 0 0
x x x	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 0 1	0 0 1	0 0 1	0 0 0	0 0 1	0 0 0
0 0 1	0 1 0	0 1 0	0 0 0	0 1 0	0 0 0
0 0 1	0 1 1	0 1 1	0 0 0	0 1 1	0 0 0
0 0 1	1 0 0	1 0 0	0 0 0	1 0 0	0 0 0
0 1 0	0 0 1	0 0 0	0 1 0	0 0 0	0 1 0
0 1 0	0 1 0	1 0 0	0 0 0	1 0 0	0 0 0
0 1 0	0 1 1	1 0 0	0 1 0	1 0 0	0 1 0
0 1 0	1 0 0	0 1 1	0 0 0	0 1 1	0 0 0
0 1 1	0 0 1	0 0 1	0 1 0	0 0 1	0 1 0
0 1 1	**0 1 0**	**1 1 0**	**0 0 0**	**0 1 0**	**1 0 0**
0 1 1	**0 1 1**	**1 1 1**	**0 1 0**	**0 1 1**	**0 0 1**
0 1 1	**1 0 0**	**1 1 1**	**0 0 0**	**0 1 1**	**1 0 0**
1 0 0	0 0 1	1 0 0	0 0 0	1 0 1	0 0 0
1 0 0	0 1 0	0 0 0	0 1 1	0 0 0	0 1 1
0 0 0	0 1 1	1 0 0	0 1 1	1 0 1	0 1 1
1 0 0	1 0 0	0 0 0	0 0 1	0 0 0	0 0 1

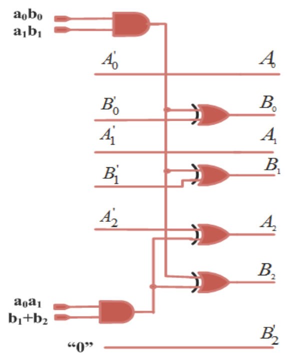


Fig. 6 Proposed reduction circuit in modulo-5.

It is noticeable that the outputs exceeded the residue range, modulo-5. The outputs of the first stage are the inputs to the second stage. That is the proposed combinational circuit introduced. Using these sets of inputs, the outputs $A_2A_1A_0$ and $B_2B_1B_0$ are evaluated as 011 and 100, respectively. Now, the set of outputs calculated are in modulo-5 range. Hence, these outputs are the inputs to the Efstathiou's parallel prefix adder in order to calculate the multiplication result.

4.1 Design Method II

The design method I has a good performance regarding delay and area of the hardware for lower modules like 5 to 9. Hence, another design is introduced which is comparable to state-of-the-art structures like that of the [24] [25] [26] [27]. Here, the proposed architecture consists of three parts. First part corresponds to generation of partial products and reduction of the partial products to the give module range. To reduce the partial products within the range of module set, in some parts the idea introduced by Vassilis *et al.* [11] and for some other parts the Efetathiou and Vergos's algorithm is applied. Next the attempt is made to reduce the partial products to rows of sum and carry operands using carry save adders. Finally to obtain the multiplication result an addition is performed to add the two operands from previous stage.

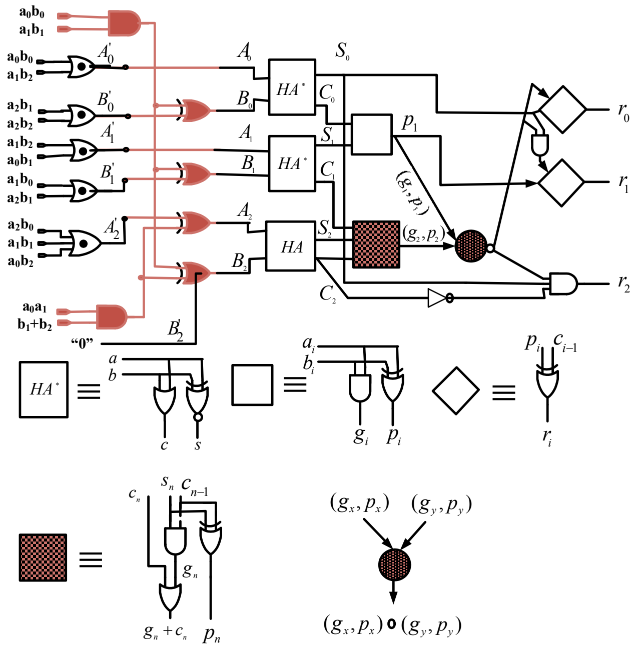
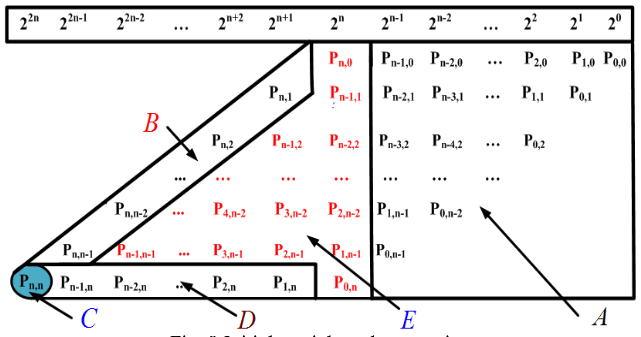


Fig. 7 Proposed modulo-5 RNS multiplier.

The $n \times n$ is partial product matrix is derived from the initial partial product matrix in Fig. (8), based on several observations. First observation is, the initial partial product matrix can be divided into five groups of A, B, C, D and E as it is shown in Fig. (8).


2^{2n} 2^{2n-1} 2^{2n-2} … 2^{n+2} 2^{n+1} 2^n 2^{n-1} 2^{n-2} … 2^2 2^1 2^0
$P_{n,0}$ $P_{n-1,0}$ $P_{n-2,0}$ … $P_{2,0}$ $P_{1,0}$ $P_{0,0}$
$P_{n,1}$ $P_{n-1,1}$ $P_{n-2,1}$ $P_{n-3,1}$ … $P_{1,1}$ $P_{0,1}$
$P_{n,2}$ $P_{n-1,2}$ $P_{n-2,2}$ $P_{n-3,2}$ $P_{n-4,2}$ … $P_{0,2}$
$P_{n,n-2}$ $P_{4,n-2}$ $P_{3,n-2}$ $P_{2,n-2}$ $P_{1,n-1}$ $P_{0,n-2}$
$P_{n,n-1}$ $P_{n-1,n-1}$ $P_{3,n-1}$ $P_{2,n-1}$ $P_{1,n-1}$ $P_{0,n-1}$
$P_{n,n}$ $P_{n-1,n}$ $P_{n-2,n}$ $P_{2,n}$ $P_{1,n}$ $P_{0,n}$
B
C
D
E
A
Fig. 8 Initial partial product matrix.

Next, For each term of $p_{i,j}$ belonging to groups B and D, $|P_{i,j}|_{2^n+1}$ is calculated and according to its weight it is scattered among the columns with the weights of 2^0 to 2^{n-1} in group A. Since $|2^{2n}|_{2^n+1}=1$, the term $p_{n,n}$ is moved to column with weight of 2^0. Finally, each partial product of the group E is inverted and repositioned at the $(i+j-n)$ column. This repositioning takes place based on equation 15. This equation shows that repositioning of

$$\begin{aligned}\left|a_i b_j 2^{i+j}\right|_{2^n+1} &= \left|-a_i b_j 2_n^{|i+j|}\right|_{2^n+1} \\ &= \left|(2^n+1-a_i b_j)2_n^{|i+j|}\right|_{2^n+1} \\ &= \left|\overline{a_i b_j}\, 2^{|i+j|_n} + 2^n 2^{|i+j|_n}\right|_{2^n+1}\end{aligned} \quad (15)$$

each bit needs a correction factor of $2^n 2^{|i+j|_n}$. In the first partial product vector, there is only one such bit and in the second partial product vector two bits must be transferred and so on. Hence, the correction factor for repositioning the partial product matrix is given by equation 16.

$$\begin{aligned}COR_1 &= 2^n(2(1+2+2^2+\dots+2^{n-2})-(n-1)) \\ &= 2^n(2^n-n-1)\end{aligned} \quad (16)$$

The partial products of the group A in fig. (8) along with equation 16, results in n+1 operands. By using carry save adders (CSA), reduction of the partial products into final summands Sum array and Carry array becomes possible. Assuming that the carry out of \$i\$th stage of CSA is c_i with weight of 2^n, this carry array can be reduced to:

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

Hence, the output carry array at the most significant bit position of each stage is used as input carry array to the next stage. Since in an n-1 stage (CSA) array, there exists n-1 carry out bits therefore there is a need for a second correction factor. This correction factor needed for the inverted (ECAs) and is presented by equation [17].

$$COR_2 = \left|2^n(n-1)\right|_{2^n+1} \quad (17)$$

The final correction factor is computed by adding equations 16 and 17 in order to find the total correction given by equation 18.

$$\begin{aligned}COR &= COR_1 + COR_2 \\ &= \left|2^n(n-1)+2^n(2^n-n-1)\right|_{2^n+1} \\ &= \left|2^n(2^n-2)\right|_{2^n+1} = 3\end{aligned} \quad (18)$$

After all the partial products are transferred from position 2^n to 2^{2n} into group A of Fig. (8), the reduction procedure introduced by [11] becomes possible. According to definition in [11], to minimize the number of bits added in each column of group A, an OR gate is sufficient to add the bits of the couples or triplets of $C'_{i,min}$. After performing partial product reduction, n-bit Sum vector (S) and n-bit Carry vector (C) are ready to be added in a final stage addition module. According to Zimmerman [24] where

$$\left|S+C+1\right|_{2^n+1} = \left|S+C+\overline{C_{out}}\right|_{2^n}$$

Table 5: *n* by *n* Partial-product matrix in modulo-17 multiplication

2^3	2^2	2^1	2^0
$pp_0=a_3b_0+a_2b_4+a_4b_1$	$a_2b_0+a_4b_1+a_1b_4$	$a_1b_0+a_4b_1$	$a_0b_0+a_4b_4+a_3b_4+a_4b_2$
$pp_1=a_2b_1+a_1b_4+a_4b_1$	$a_1b_1+a_4b_4+a_1b_4+a_4b_3$	$a_0b_1+a_1b_4$	$(a_2b_4+a_4b_1+a_2b_2)'$
$pp_2=a_1b_2+ab_2$	$a_0b_2+a_4b_2$	$(a_2b_4+a_4b_1+a_3b_2)'$	$(a_1b_4+a_4b_3+a_3b_1)'$
$pp_3=a_0b_3+a_4b_3+a_3b_4$	$(a_2b_4+a_4b_1+a_3b_3)'$	$(a_1b_4+a_4b_3+a_2b_3)'$	$(a_1b_3+a_4b_0+a_0b_4)'$
$pp_4=0$	0	1	0

the constant '1' in the above equation can be obtained from equation 18 which is the constant '3'. Hence the new final partial product is the constant '2' which will be added to the rest of the partial products.

As an example, let's consider modulo-17 multiplier for the proposed architecture here: First, the partial products of the columns 2^4 to 2^8 are transferred into appropriate positions according to what is explained above. Table (4) shows the partial products transfer into matrix A.

Table 4: Transfer of partial products to appropriate positions

2^3	2^2	2^1	2^0
a_3b_0	a_2b_0	a_1b_0	a_0b_0
a_2b_1	a_1b_1	a_0b_1	a_4b_4
a_1b_2	a_0b_2	a_4b_1	a_4b_3
a_0b_3	a_4b_2	a_1b_4	a_3b_4
a_4b_3	a_2b_4	$(a_3b_2)'$	a_4b_2
a_3b_4	a_4b_1	$(a_2b_3)'$	a_2b_4
a_4b_2	a_1b_4		a_4b_1
a_2b_4	$(a_3b_3)'$		a_1b_4
a_4b_1			$(a_3b_1)'$
a_1b_4			$(a_2b_2)'$
			$(a_1b_3+a_4b_0+a_0b_4)'$

The next step is the reduction of partial products shown in Table (4) into four rows. At this point, the objective is to identify the largest possible number of disjoint couples or triplets, formed by bits contained in each of the $Z_{k,i}$ sets, which satisfy conditions of the equations 10 or 11, respectively. By maximizing the number of bits in the particular couples or triples, the number of bits to be added in a column is minimized [11]. For example in position 2^0 the partial products of a_0b_0, a_4b_4, a_3b_4 and a_4b_2 satisfy the conditions of equations 10 or 11; therefore, they are wired OR together and since only and only one of the participated partial products could have value of 1, hence, the wired OR gate is used instead of conventional OR gate in order to perform logical OR of the inputs. As it was mentioned previously, a wired OR is an OR gate with a dot inside of it. Another example: in position 2^4 the partial products of a_4b_0, a_0b_4, and a_1b_3 satisfy the conditions in equation 10 or 11: therefore,

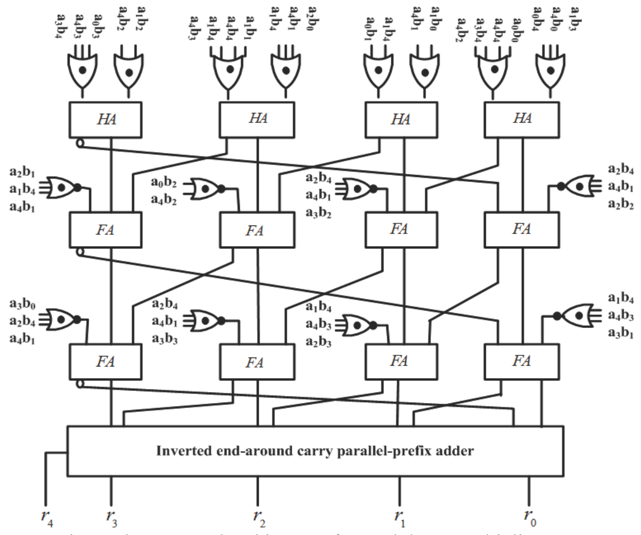


Fig. 9 The proposed architecture for modulo-17 multiplier.

they are wired OR together and then inverted and transferred to position 2^0. And yet in another example where the partial products a_2b_4 and a_4b_1 in position 2^0 satisfy equation (11), they can be added by the partial product a_2b_2 in the same column. After doing the required manipulations, $\overline{a_2b_4 + a_4b_1 + a_2b_2}$ is obtained as sum and $a_2b_4 + a_4b_1$ as carry which goes to next position is calculated. This approach is repeated for all the columns until the $n \times n$ partial product matrix is derived (see Table (5)). The block diagram of Table (5) is presented in Fig. (9). As it is shown in the figure partial products are implemented either by wired OR gates or with wired NOR gates. It is evident that the proposed multiplier out performs the previous multiplier designs and it is well suited for VLSI implementation.

5. Area-Delay analysis and comparisons

In the following section the area and time complexity of the proposed multipliers are analyzed. The two design methods are introduced and design method I is explained in order to be used for lower modulo sets like 5, 7 and 9. Design method II is a more general multiplier and can be employed for higher modulo sets.

Table 6: Comparison of the proposed multiplier against [17] and [11]

	Stouraitis's RNS multiplier [17]		Vassilis's RNS multiplier [11]		Proposed RNS multiplier	
Modulo	Time gate delay (*ns*)	Area Transistors	Time gate delay (*ns*)	Area Transistors	Time gate delay (*ns*)	Area Transistors
5	32	0.8750	16.5	0.6702	10	0.4300
9	74	0.8450	39.4	0.6430	22	0.3900
17	112	0.8790	56.5	0.6512	32.45	0.4200
33	132.43	0.7344	65	0.5458	38.32	0.3038
129	168.08	0.6834	90.3	0.5581	51.71	0.2903
257	212.2	0.8870	112	0.6705	61.33	0.4240

In both methods I and II, complexity reduction [11] is applied by input classification into pair and triple groups, so that the maximum sum of the elements of every group does not overshoot. In both design methods introduced here, it was shown that, one can use OR gate to replace a more complex half adders or full adders. This substitution is possible because, two or more bits cannot be simultaneously asserted, therefore, due to the definition, carry generation is omitted [11]. In this article we have done exactly the same and replaced half adders and full adders by OR gates. In design method I a combinational circuit added to keep the elements of each column in the residual range which allowed the parallel prefix adder used to compute the RNS multiplier result. The performance of the proposed RNS multiplier with respect to delay and area is compared to [17] and [11] as presented in Table (6). The comparison of different modules are computed and the results indicate that the proposed architecture is more optimized compared to that of the [17], [11] and [27].

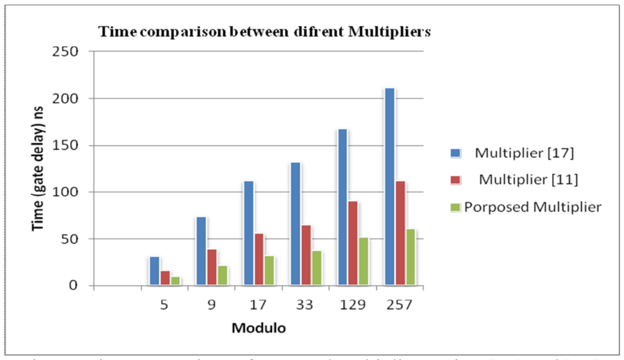


Fig. 10 Time comparison of proposed multiplier against [17] and [11].

In order to quantify the area complexity of the architecture, it is assumed that FA, HA, two input OR gate and three input OR gate each have complexity of 28, 12, 6 and 8 transistors, respectively [11]. The time and area comparison between that of the proposed structure and [17] and [11] are shown in Fig. (10) and Fig. (11), each in the order of modulo sets given. The preformed comparisons indicate that the proposed design method I has time and area optimization against [17] and [11]. For Modulo sets like 5, 7 and 9 proposed in design I is also superior to Vergos *et al.* [27]. They used unit gate model for qualitative comparisons: considered all 2-input monotonic gates count as one, 2-input XOR/XNOR gate count as two equivalents for both area and delay [27]. Using the same qualitative comparison in lower modulo-5, the delay and area calculated from [27] is 17 and 35 logic gates count respectively, whereas the delay and area for the newly proposed design method I is 13 and 35 logic counts. Area is estimated in number of required transistors, while time is expressed in gate delays.

As emphasized, design method II is general proposed multiplier for any modulo sets. Employing the same strategy adopted by [27] for computing delay and area, the area calculation is apportioned into three modules. The first module is to calculate the area needed for forming partial products. To produce the partial products of Fig. (8), there is a need for n^2+1 AND (NAND) gates and since the wired OR gates are used in this proposed in design therefore the total area required to produce partial products is given by equation (19).

$$A_{partial} = n^2 + 1 \qquad (19)$$

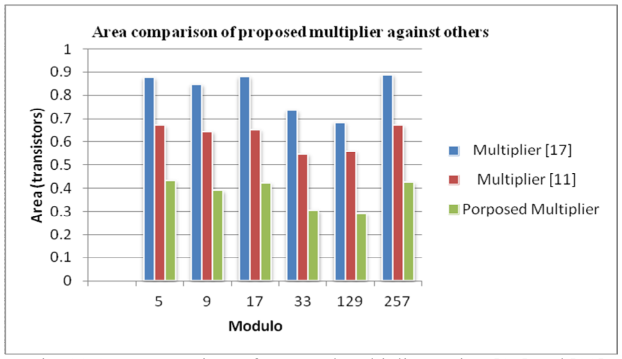


Fig. 11 Area comparison of proposed multiplier against [17] and [11].

The second module consists of CSA tree, and considering that each HA and FA has area of 3 and 7 respectively.

Looking at figure (8) and Table (5), in the first row of design method II there is 3n number of HAs; therefore, the area is 3n. The remaining rows have $n(n^2)$ FAs hence, the area is 7n(n-2). The total equivalent gate required for partial products is given by equation (20).

$$A_{CSA} = 3n + 7n(n-2) \qquad (20)$$

The last module is the end-around carry which was computed in [27] and is equal to equation (21).

$$A_{adder} = \frac{9}{2}n \times log_2 n + \frac{1}{2}n + 6 \qquad (21)$$

Adding equations (19), (20) and (21), would yield the total area equivalent gates occupied by the proposed multiplier and can be computed by equation (22).

$$A_{proposed} = 8n^2 - \frac{21}{2}n + \frac{9}{2}n\lceil log_2 n \rceil + 7 \qquad (22)$$

whereas, the area of the proposed multiplier by [27] is given by equation (23).

$$A_{[27]} = 8n^2 - \frac{21}{2}n + \frac{9}{2}n\lceil log_2 n \rceil + 11 \qquad (23)$$

As it is evident the proposed design has a better performance regarding area occupation of the hardware compared to [27]. Table (7) shows the saving in area offered by proposed multipliers for different operand sizes.

Table 7: Area comparisons

n	4	8	12	16	20	24	28	32
A Proposed	129	543	1227	2176	3386	4858	6591	8583
A [27]	147	577	1229	2241	3529	5001	6729	8713
Percent Saving	12.2	5.9	5.6	2.9	4.1	2.9	2.1	1.5

Computation of delay is based on three stages, namely, the contraction delay of partial products, the delay reduction of partial product via CAS adders and finally the delay of inverted EAC adder. The delay of partial product formation in this proposed design is one time unit, and in a worst case situation another time unit is the delay of wired or gates. Therefore, the delay here has the same derivation as [27], the only difference is that the delay for partial products here is 2 time unit when n+1 is a number of Dadda sequence (6, 9, 13, 19, 28, 42, 63,…). and 1 time unit otherwise. The delay of CSA parallel adders here is the same as [27]. At last, the time delay of inverted EAC adder is given by $T_{adder} = \lceil log_2 n \rceil + 3$ which is same as that of the [27]. Adding up the time delay of three stages of the proposed multiplier would give the total delay as:

$$Y = \begin{cases} 18 & \text{if } n=4 \\ 4D(n+1)+2\lceil \log n \rceil+5 & \text{if } n+1 \text{ is a number of Dadda sequence.} \\ 4D(n+1)+2\lceil \log n \rceil+4 & \text{otherwise} \end{cases}$$

The only difference between this time delay and that of the [27] is when n+1 is a number of Dadda sequence and that is because in this design wired OR gates are used. Table (8) shows the delay comparison between the proposed design and that of [27].

Table 8: Time Comparisons

n	4	8	12	16	20	24	28	32
T Proposed	18	27	33	36	42	42	46	46
T [27]	18	28	34	36	42	42	46	46

6. Conclusion

Two novel low complexity combinational 2^n+1 RNS multipliers Using Parallel Prefix Adders are proposed in this article. The proposed designs use a suitable grouping of inputs into couples or triplets in a way that, the maximum sum of element does not exceed the unity. In order to be able to do the modular addition in parallel, a new proposed implementation is proposed. Comparisons against the state-of-the-arts modular multiplier architectures show that: the proposed multiplier I is more compact and offers a higher speed for lower modules and design method II is more compact and has a comparable delay. The proposed RNS multiplier can be applied more efficiently in practical approaches specially, in DSP systems where residue number systems are the main course.

References

[1] A. Omondi, and B. Premkumar, Residue Number Systems Theory and implementation, College Press page 523-541, Chap. 01 2007.

[2] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Tayor, “Residue Number System Arithmetic: Modern Applications in Digital Signal Processing”, IEEE Press, New York 1986.

[3] S.Timarchi, and K. Navi: “A Novel Modolo 2^n+1 Adder Scheme”, computer Society of Iran Computer Conference 2007(CSICC'07), Shahid Beheshti University, Tehran, Iran 2007.

[4] N. Szabo, and R. Tanak, Residue Arithmetic and its Application to Computer Technology, New York, LCCCN: 66-15186, McGraw-Hill Book Company 1967.

[5] R. Conway and J. Nelson: “Improved RNS FIR Filter Architectures”, IEEE Trans. On Circuit and Systems-II: Express Briefs, Vol. 51, No. 1, Jan. 2004.

[6] K. c. Posch, R. Posch: "Modulo Reduction in Residue Number Systems," IEEE Transaction On Parallel And Distributed Systems, Vol. 6 No.5, May 1995.

[7] L. Yang and L. Hanzo, "Redundant Residue Number System Based ERROR Correction Codes", IEEE 54th on Vehicular Technology Conference, Vol. 3, pp. 1472-1776, Oct. 2001.

[8] J. Ramirez, et al., "Fast RNS FPL-based Communications Receiver Design and implementation", Proc. 12th Int. Conf. Field Programmable logic, pp. 472-481, 2002.

[9] R. Rivest, A. Shamir, and l. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", Comm. ACM, Vol. 21, no.2, pp. 120-126, Feb. 1978.

[10]G. A. Jullien, "Implementation of Multiplication, Modulo a Prime Number, with Application to Theoretic Transforms", IEEE Transaction on Computers, Vol.29,No.10, pp.899-905, Oct. 1980.

[11]Vassilis Paliouras, Konstantina Karagianni and Thanos Stouraitis, "A Low-Complexity Combinatorial RNS Multiplier", IEEE Tran. Circuit and system-II: Analog and Digital Signal Processing, VOL.48, NO. 7, pp. 675-683 JUL. 2001.

[12]M. Soderstand and Cvernia, "A High Speed Low-Cost Modulo P Multiplier with RNS Arithmatic Application" ,Proc. IEEE,Vol 68, pp.529-532,Apr. 1980.

[13]D. Radhakrishnan, and Y. Yuan, "A Fast RNS Galois Field Multiplier", IEEE Transactions on Circuits and Systems, pp. 2909-2912, 1990.

[14]A. A. Hiasat, "New Efficient Structure for a Modular Multiplier for RNS", IEEE Transaction on Computers, Vol. 49, pp.170-174, Feb. 2000.

[15]E. D. DiClaudio, F. Piazza, and G. Orlandi, "Fast Combinatorial RNS Processors for DSP Applications", IEEE Trans. Comput. Vol. 44, pp. 624-633, May 1995.

[16]A. Wrzyszcz, D. Milford, and E. Dagless, "A New Approach to Fixed-Coefficient Inner Product Computation over Finite Rings", IEEE Trans. Computers, vol. 45, no. 12, pp. 1345-1355, Dec. 1996.

[17]T. Stouraitis, S. W. Kim, and A. Skavantzos, "Full Adder-based Arithmetic Units for Finite Integer Rings", IEEE Trans. Circuits Syst. II, vol.40, pp. 740-744, Nov. 1993.

[18]C. Efstathiou, HT. Vergos, D. Nikolos, "Fast Parallel-Prefix Modulo 2^n+1 Adders" IEEE TRANSACTIONS ON COMPUTERS, VOL. 53, NO. 9, Sep. 2004.

[19]J. l. beuchat, "A Family of Modulo 2^n+1 Multiplier", sep. 2004.

[20]D. J. Soudris, V. Paliouras, T. Stouraitis, and C. E. Goutis, "A VLSI Design Methodology for RNS Full Adder-based Inner Product Architectures", IEEE Trans. Circuits Syst. II, vol. 44, pp. 315-318, Apr. 1997.

[21]V. Paliouras and T. Stouraitis, "Multifunction Architectures for RNS Processors", IEEE Trans. Circuits Syst. II, vol. 46, pp. 1041-1054, Aug. 1999.

[22]S. Timarchi, K. Navi, and M. Hosseinzade, "New Design of RNS Subtractor for modulo 2n+1", 2nd IEEE International Conference on Information and Communication Technologies: From Theory to Application, pp. 24-28 Apr. 2006.

[19] P. M. Kogge, and H. S. Stone, "A Parallel Algorithms for the Efficient Solution of a General Class of Recurrence Equations" IEEE Trans. Computers, Vol. 22, No. 8, pp. 783-791, Aug. 1973.

[24]R. Zimmerman, "Efficient VLSI Implementation of Modulo $2^n \pm 1$ Addition and Multiplication", In Proc. of the 14th IEEE Symposium on Computer Arithmetic, pp. 158–167, April 1999.

[25] C. Efstathiou, H. T. Vergos, G. Dimitrakopoulos, and D. Nikolos, "Efficient diminished-1 modulo $2^n + 1$ ultipliers", IEEE Trans. Comput., Vol. 54, No. 4, pp. 491–496, 2005.

[26] H. T. Vergos, C. Efstathiou, "Novel Modulo 2^n+1 Multiplier", 9th EUROMICRO Conference on Digital System Design, pp. 168-175, 2006.

[27] H. T. Vergos, C. Efstathiou, "Design of efficient modulo 2^n+1 multipliers", IET Comput. Digit. Tech.,Vol. 1, No. 1, pp. 49-57, 2007.

Mohammad R. Reshadinezhad He was born in Isfahan, Iran, in 1959.He received his B.S. and M.S. degree from the Electrical Engineering Department of University of Wisconsin, Milwaukee, USA in 1982 and 1985,respectively. He has been in position of lecturer as faculty of computer engineering in University of Isfahan since 1991. He also received the Ph.D Deggree in computer architecture from Shahid Beheshti University, Tehran, Iran, in 2012. He is currently Assistant Professor in Faculty of computer Engineering of Isfahan University. His research interests are digital arithmetic, Nanotechnology concerning CNFET, VLSI implementation, logic circuits designs and Cryptography.

Farshad Kabiri Samani He received his B.S. and M.S. degree in computer engineering (hardware) from University of Najaf Abad, Iran and University of Arak, Iran in 2007 and 2010 respectively. He is currently working as a lecturer and researcher in Faculty of electrical and computer engineering department of Lenjan University, Lenjan, Iran. His research interests mainly focus on computer arithmetic algorithms and circuits, microprocessor architecture, and VLSI hardware designing.