A Suggestion for a Fast Residue Multiplier for a Family of Moduli of the Form $(2^n - (2^p \pm 1))$

AHMAD A. HIASAT

Electronics Engineering Department, Princess Sumaya University, PO Box 1438, Amman 11941, Jordan
Email: aahiasat@psut.edu.jo

The family of moduli that has the form $(2^n-(2^p\pm 1))$ is considered in this paper. A suggestion for a fast residue multiplier for this family of moduli is introduced. The multiplication algorithm proposed in this paper generates $(2n+p-2)$ partial products; however, it compresses the magnitude of each partial product to be less than 2^n. Although it requires an additional integrated circuit area compared with the most recent published study, the new proposed modular multiplier has a logarithmic delay, which makes it faster than any other modular multiplier. Moreover, it is even faster than binary-based iterative array multipliers with a final CLA addition. The proposed modular multiplier is very suitable for medium and large dynamic ranges.

Received 8 August 2002; revised 10 March 2003

1. INTRODUCTION

Residue Number System (RNS) is an arithmetic system that has the merit of carry-free arithmetic operations. The use of RNS increases the speed of computations. Addition, subtraction and multiplication can be conducted on residue digits concurrently and independently [1, 2]. Moreover, RNS demonstrated an efficiency in realizing different types of applications using the above operations. Specifically, RNS has been realized in applications involving the design of adaptive coded multicarrier modulation systems [3], delta–sigma modulators [4], Discrete Fourier Transform, Number Theoretic Transform and many other applications [1, 2]. This implies that designing efficient modular components like adders and multipliers is a very essential issue in implementing different RNS-based applications and processors [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26].

Many researchers have addressed the modular multiplication. Ramnarayan [5] introduced a modular multiplier for moduli of the form $(2^n - 2^k - 1), k < n$. Soderstrand and Vernia [6] presented the square-law modular multiplier. The design consisted of ROMs only, which imposed limitations on the size of the moduli and the speed performance. Jullien [7] introduced a technique for multiplying numbers modulo a prime number. However, this approach has the same cons of the approach in [6]. Taylor [8] proposed different structures for modular multipliers for the moduli set $(2^n - 1, 2^n, 2^n + 1)$. Hiasat [9] suggested a residue multiplier modulo $(2^n \pm 1)$ using a binary-based multiplier followed by an adder. Dugdale [10] introduced a residue multiplier using decomposition of factors, where the moduli are assumed to be non-prime. The multiplication is performed with an index calculus technique applied to the prime factors of each modulus. Radhakrishnan and Yuan [11] introduced two approaches for the design of a modular multiplier. The first approach uses additive index, while the other one uses pseudo-primitive root as the generator for the elements of multiplicative group. Although their design has considerable savings over that proposed by Jullien [7], the size of the ROMs used make it unsuitable for large moduli. Alia and Martinelli [12] proposed a modular multiply structure. It requires binary adders and multipliers. The algorithm is based on an approximate non-integer binary multiplication, in which the error is then corrected. Their architecture has been shown to be optimal compared with ROM-based structures. Walter [13] introduced an implementation of a modular multiplier tailored for cryptographic applications, where multiplication is implemented via repeated addition. Parker and Benaissa [14] introduced a serial modular multiplier. Elleithy and Bayoumi [15] proposed a systolic architecture for modulo multiplication which consists, mainly, of modular adders. Di Claudio *et al*. [16] presented also a multiplier based on a new pseudo-RNS and utilized, basically, arithmetic components. Bajard *et al*. [17] presented an RNS Montgomery modular multiplier for very large operands. The algorithm is based on an adapted mixed-radix approach. Hiasat [18] also proposed a modular multiplier using tow cascaded binary multipliers and two binary adders. Paliouras *et al*. [19] proposed a combinatorial RNS multiplier suitable for small dynamic ranges. Savas and Koc [20] introduced an algorithm for computing the Montgomery modular multiplication inverse. Curiger *et al*. [21] proposed a $(2^n + 1)$ modular multiplier without using memory. Wrzyszcz and Milford [22] also proposed a modulo $(2^n + 1)$ multiplier which does not use a special binary representation of numbers. Ma [23]

presented modulo $(2^n + 1)$ multiplier using a carry-save adder and two other modular adders. Zimmermann [24] presented modulo $(2^n \pm 1)$ adders and multipliers using end-around-carry modular adders based on parallel-prefix concepts.

In the remaining part of this section, the family of moduli that has the form $(2^n - (2^p \pm 1))$ is explored, briefly. In Section 2, the proposed algorithm for the modular multiplier is presented. Its hardware implementation is introduced in Section 3. Section 4 compares the new multiplier with other existing ones.

The modulus m can take any value. However, in RNS, a moduli set consists of r moduli defined as: $\{m_1, m_2, m_3, \ldots, m_r\}$, where any pair within this moduli set needs to be prime or relatively prime. For a particular positive integer n, some of the prime integers or relatively prime integers, m, have the form:

$$m = 2^n - (2^p \pm 1) \tag{1}$$

The positive integer p can assume values in the range: $p \in [2, (n-2)]$. However, p cannot assume the value n because this would lead to a trivial case. A value of $p = n-1$ puts m into the form of $m = 2^n \pm 1$. This particular value of m has been covered extensively in the literature [21, 22, 23, 24, 25]. In fact, Rader [25] was the first to propose the use of carry-save adders to build modulo Mersenne or Fermat numbers to compute error-free convolutions of real integer sequences.

Table 1 shows typical values of integers resulting from the formula given in (1). Values of p and the corresponding values of m are listed for $n \in [8, 12]$. Nevertheless, n can assume any positive value outside the given interval. The prime integers are in boldface. Examining Table 1 for any particular n reveals that just a few of the shown integers are prime. Nevertheless, a considerable number of these integers are relatively prime. Thus, the form given in (1) can be thought of as a modulus generator that can form many moduli sets. Smaller values of n are suitable in residue-based digital signal processing applications [1, 2, 3, 4], for moduli sets of the form $(2^{2k-4}, 2^{2k-1}-1, 2^{2k-1}+1, 2^{2k-1}-2^k+1, 2^{2k-1}+2^k+1)$, where $n = 2k-1$ and $p = k$.

Residue-based modular multiplier (like the one proposed in this paper) is realized by parallel addition of small wordlengths [27, 28]. On the other hand, cryptographic applications use larger values of n ($n > 512$ bits). The heart of most cryptosystem-based modular multipliers in the RSA case is an adder that realizes modulo multiplication of such huge operands by repeated addition [17, 20]. When compared with residue-based modular multipliers, the cryptosystem-based modular multipliers require much more time and area, due to their huge wordlength and implementation architecture.

The basic goal of this paper is to design a modular arithmetic multiplier. Specifically, it is intended to evaluate:

$$T = |Z|_m = |XY|_m \tag{2}$$

where $X, Y, T \in [0, m)$, $X = \sum_{j=0}^{n-1} x_j 2^j$, x_j is the jth bit of X, $Y = \sum_{j=0}^{n-1} y_j 2^j$, y_j is the jth bit of Y, T is the least non-negative remainder when dividing the product XY by m, $n = \lceil \log_2 m \rceil$ and $\lceil . \rceil$ is the ceiling value of $(\cdot)$ (i.e. the next integer greater than or equal to $(\cdot)$).

TABLE 1. Integers in the form of $\left(2^n - (2^p \pm 1)\right)$.

p	$n=8$	$n=9$	$n=10$	$n=11$	$n=12$
2	**191**, **193**	**383**, 385	**1019**, **1021**	2043, 2045	**4091**, **4093**
3	223, 225	447, **449**	1015, 1017	**2039**, 2041	4087, 4089
4	**239**, **241**	**479**, 481	1007, **1009**	2031, 2033	**4079**, 4081
5	247, 249	495, 497	**991**, 993	2015, **2017**	4063, 4065
6	**251**, 253	**503**, 505	959, 961	1983, 1985	4031, 4033
7		507, **509**	895, 897	1919, 1921	**3967**, 3969
8			767, **769**	1791, 1793	3839, 3841
9				1535, 1537	**3583**, 3585
10					3071, 3073

2. THE NEW PROPOSED MODULAR MULTIPLIER ALGORITHM

The product $Z = XY$ can be expressed as:

$$Z = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} z_{ij} 2^{i+j} \tag{3}$$

where z_{ij} refers to the bit resulting from ANDing the bits x_i and y_j.

Figure 1 shows the layout of all bits in Z. Thus, Z can be viewed as a summation of n rows where the ith row is defined as: $\sum_{j=0}^{n-1} z_{ij} 2^{i+j}$. However, the magnitude range of each row is different. For instance, the very last row, $\sum_{j=0}^{n-1} z_{(n-1)j} 2^{n-1+j}$, has a magnitude range: $0 \leq \sum_{j=0}^{n-1} z_{(n-1)j} 2^{n-1+j} < 2^{2n-1}$. Evaluating T, defined in (2), implies subjecting Z to modulus m. In the following analysis, it is intended to compress the magnitude of each row in Z, using the modulus m to be less than 2^n. The expression for Z, given in (3), can be rewritten as:

$$Z = R_1 + L_1 \tag{4}$$

where R_1 refers to the right part of Z, shown in Figure 1. Equivalently,

$$R_1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} z_{ij} 2^{i+j}, \quad 0 \leq (i+j) < n \tag{5}$$

The ith row in R_1 refers to $\sum_{j=0}^{n-1} z_{ij} 2^{i+j}$, where $(i+j) < n$. Obviously, each ith row in R_1 is bounded by: $0 \leq \sum_{j=0}^{n-1} z_{ij} 2^{i+j} < 2^n$.

Similarly, L_1 refers to the left part of Z, shown in Figure 1. Equivalently,

$$L_1 = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j}, \quad (2n-1) > (i+j) \geq n \tag{6}$$

Since the objective is to compress the magnitude of each row in Z (and consequently in L_1) to be less than 2^n, (6) can

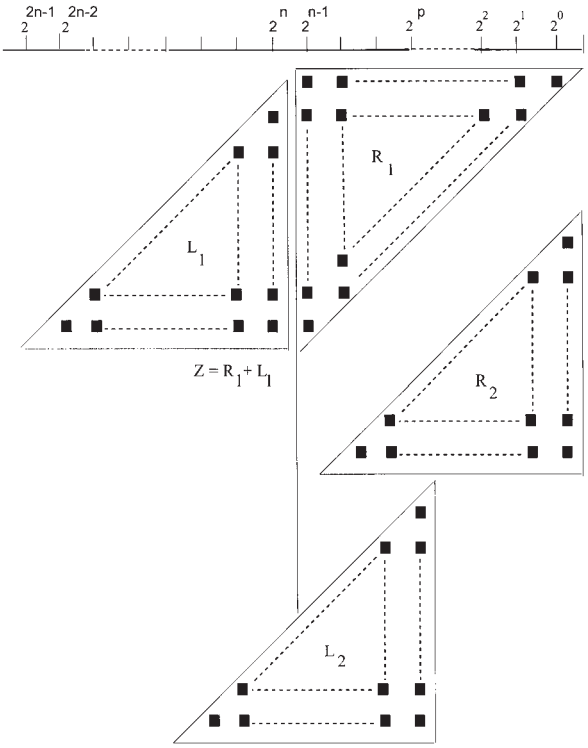

FIGURE 1. Splitting $Z = XY$ into R_1 and L_1, and replacing L_1 with R_2 and L_2.

be rewritten as:

$$L_1 = 2^n \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n} \right), \quad n \le (i+j) < (2n-1) \tag{7}$$

Taking modulus m of (1) and considering $(2^n - (2^p + 1))$ rather than $(2^n - (2^p - 1))$ at this stage, then $|2^n|_m$ can be written as:

$$\left|2^n\right|_m = (2^p + 1) \tag{8}$$

Thus, (8) can be substituted into (7), where L_1 is expressed as:

$$L_1 = (2^p + 1) \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n} \right), \quad n \le (i+j) < (2n-1) \tag{9}$$

The value of L_1 in (9) can be split into:

$$L_1 = \underbrace{\left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n} \right)}_{0 \le i+j-n < n} + \underbrace{\left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n+p} \right)}_{p \le i+j-n+p < n+p} \tag{10}$$

Consider the first summation on the right-hand side (RHS) of (10) to be R_2, namely,

$$R_2 = \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n} \right), \quad 0 \le (i+j-n) < n \tag{11}$$

Each ith row in R_2 is bounded by: $0 \le \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n} < 2^{n-1}$. However, this is not the case with the other summation on the RHS of (10), L_2, where,

$$L_2 = \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n+p} \right), \quad p \le i+j-n+p < n+p \tag{12}$$

where Figure 1 shows L_1, R_2 and L_2.

Equivalently, L_2 can be decomposed into two summations:

$$L_2 = \underbrace{\left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n+p} \right)}_{p \le (i+j-n+p) < n} + \underbrace{\left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n+p} \right)}_{n+p > (i+j-n+p) \ge n} \tag{13}$$

Referring to the first summation on the RHS of (13) as R_3, then:

$$R_3 = \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n+p} \right), \quad p \le (i+j-n+p) < n \tag{14}$$

Each ith row in R_3 is bounded by: $0 \le \sum_{j=1}^{n-1} z_{ij} 2^{i+j-n+p} < 2^n$.

Similarly, consider the last summation in the RHS of (13) to be L_3. If there are no terms in L_3 for which $(i+j-n+p) \ge n$ then analysis and simplification stop right here.

However, if this is not the case, L_3 can be rewritten in the form:

$$L_3 = 2^n \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-2n+p} \right), \quad n+p > (i+j-n+p) \ge n \tag{15}$$

Substituting (8) into (15) leads to:

$$L_3 = \underbrace{\left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-2n+p} \right)}_{0 \le (i+j-2n+p) < p} + \underbrace{\left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-2n+2p} \right)}_{2p > (i+j-2n+2p) \ge p} \tag{16}$$

Referring to the first summation on the RHS of (16) as R_4, then:

$$R_4 = \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-2n+p} \right), \quad 0 \le (i+j-2n+p) < p \tag{17}$$

where each row in R_4 is less than 2^n.

Similarly, referring to the other summation on the RHS of the same equation as L_4, then:

$$L_4 = \left(\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} z_{ij} 2^{i+j-2n+2p} \right), \quad 2p > (i+j-2n+2p) \ge p \tag{18}$$

If there are no terms in L_4 for which $(i+j-2(n-p)) \geq n$, analysis stops. If this is not the case, L_4 in (18) can be decomposed into two summations, namely, R_5 and L_5:

$$L_4 = \underbrace{\left(\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} z_{ij}2^{i+j-2n+2p}\right)}_{p\leq(i+j-2n+2p)<n} + \underbrace{\left(\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} z_{ij}2^{i+j-2n+2p}\right)}_{2p>(i+j-2n+2p)\geq n} \tag{19}$$

where

$$R_5 = \underbrace{\left(\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} z_{ij}2^{i+j-2n+2p}\right)}_{p\leq(i+j-2n+2p)<n} \tag{20}$$

and

$$L_5 = \underbrace{\left(\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} z_{ij}2^{i+j-2n+2p}\right)}_{2p>(i+j-2n+2p))\geq n} \tag{21}$$

Figure 2 shows how each L_i is replaced with R_{i+1} and L_{i+1}.

The analysis continues in that sense until each row in any summation is less than 2^n. Generally speaking, R_k, where $k \geq 2$, can be expressed as:

$$R_k = \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} z_{ij}2^E \tag{22}$$

where

$$E = \begin{cases} 0 \leq \overbrace{i+j-n-\left\lfloor\frac{k-1}{2}\right\rfloor(n-p)}^{E} < p, & k \text{ even} \\ \leq \underbrace{i+j-\left\lfloor\frac{k-1}{2}\right\rfloor(n-p)}_{E} < n, & k \text{ odd} \end{cases}$$

and $\lfloor . \rfloor$ is the floor value of $(\cdot)$, (i.e. the largest integer less than or equal to $(\cdot)$). Moreover, it has to be observed in all cases of R_k, $k \geq 2$, that $i + j \geq n$.

Similarly, L_k, where $k \geq 1$, can be expressed as:

$$L_k = \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} z_{ij}2^E \tag{23}$$

where

$$E = \begin{cases} n \leq \overbrace{i+j-\left\lfloor\frac{k}{2}\right\rfloor(n-p)}^{E}, & k \text{ odd} \\ p \leq \underbrace{i+j-\left\lfloor\frac{k}{2}\right\rfloor(n-p)}_{E}, & k \text{ even} \end{cases}$$

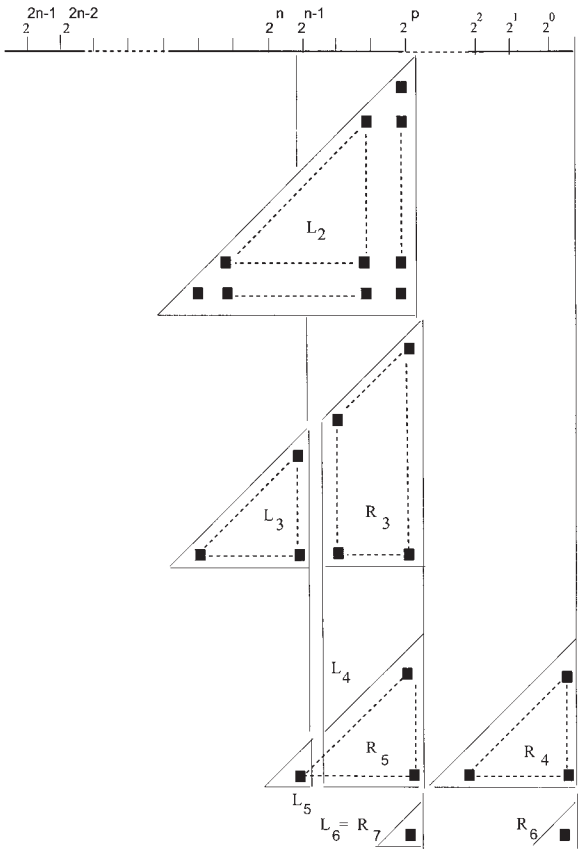

FIGURE 2. Splitting L_2 for the case $q = 3$.

Similarly, it has to be observed in all cases of L_k, that $i + j \geq n$.

In the approach followed above, each time $|2^n|_m$ is replaced by $(2^p + 1)$, L_k is reduced in size by $(n - p)$ bits. Consequently, R_{k+1} and R_{k+2} summations are produced. Starting with L_1 with a bit width of $(n - 1)$, the total number of summations resulting is $2q$, where $q = \lceil (n-1)/(n-p) \rceil$. These summations are: $R_2, R_3, R_4, \ldots, R_{2q}, R_{2q+1}$. However, in evaluating T, the summation R_1 has to be taken into consideration. Thus,

$$T = \left| \sum_{k=1}^{2q+1} R_k \right|_m \tag{24}$$

The following is a simplified algorithm for computing T:

Input: X and $Y \in [0, m)$, $m = 2^n - 2^p - 1$
Output: T, where $T = |XY|_m$
Procedure:
begin
Initialize $T = 0$;
Compute $Z = XY$, $q = \lceil \frac{n-1}{n-p} \rceil$;
For each i, $1 \leq i \leq (2q + 1)$
Compute R_i
Compute $T = (T + R_i) mod\ m$;
end

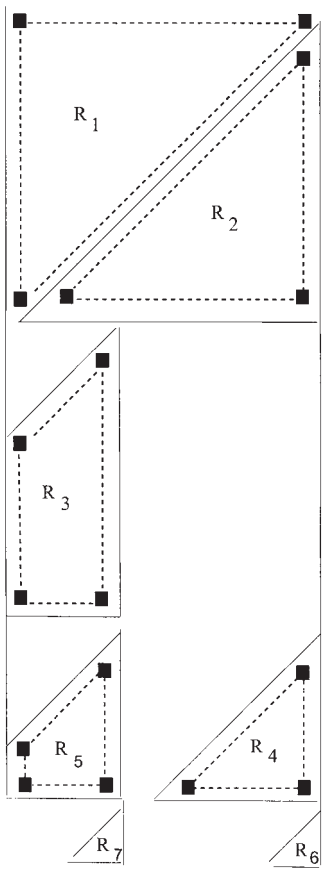

FIGURE 3. Layout of all R_i, $i = 1, \ldots, (2q+1)$, for the case $q = 3$.

3. HARDWARE IMPLEMENTATION

Figure 3 shows the layout of different R_i, $i = 1, 2, \ldots, (2q+1)$, for the case $q = 3$. In each summation R_k, $k = 3, 5, 7, \ldots, (2q+1)$, there are v_k rows, where v_k is given by:

$$v_k = \left((n-1) - \left(\left\lfloor \frac{k}{2} \right\rfloor - 1\right)(n-p)\right) \qquad (25)$$

Each row in R_k has $(n-p)$ bits, where the weight of the LSB is 2^p while that of the MSB is 2^{n-1}. The magnitude range of any row is $[2^p, 2^n)$, unless the bits of a row are all zeros. Similarly, for each $R_k, k = 2, 4, 8, \ldots, 2q$, there are v_k rows, where v_k is given in (25). Thus, the ith row ($i = 1, 2, \ldots, v_k$) in R_k has i bits. The weight of the LSB is 2^0 while that of the MSB is 2^{i-1}. The magnitude range of any row is less than 2^i. This leads to the fact that the pair (R_k, R_{k+1}), where k is an even integer larger than or equal to 4, do not overlap in their magnitudes. Thus, they can be combined to form v_k vectors, each of n bits. For instance the pair (R_4, R_5) forms $((n-1)-(n-p))$ vectors, while the pair (R_6, R_7) forms $((n-1)-2(n-p))$ vectors. Generally speaking, the pair (R_k, R_{k+1}), where k is even, forms $((n-1)-(\lfloor k/2 \rfloor - 1)(n-p))$ vectors. The pair (R_1, R_2) can be combined to form n vectors. The summation R_3 will stand by itself, containing $(n-1)$ vectors each with $(n-p)$ bits. Thus, the number of vectors associated with each pair is:

$$\begin{array}{ll} R_1 + R_2 & \rightarrow\ n \\ R_3 & \rightarrow\ (n-1) \\ R_4 + R_5 & \rightarrow\ (n-1)-(n-p) \\ \vdots & \vdots \\ R_{2q} + R_{2q+1} & \rightarrow\ (n-1)-(q-1)(n-p) \end{array}$$

Denoting v to be the total number of vectors resulting from all pairs, then

$$v = n + q(n-1) - \frac{q(q-1)}{2}(n-p) \qquad (26)$$

If p is selected such that $p \leq n/2$, then $q = 2$ and v reduces to:

$$v = 2n + p - 2 \qquad (27)$$

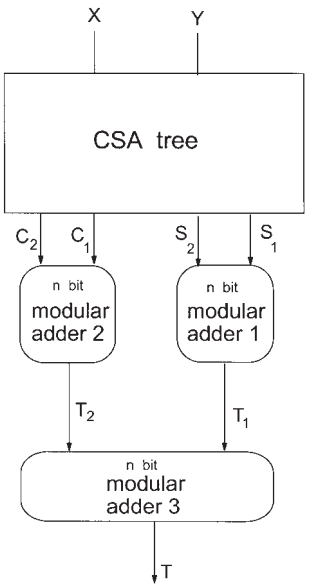

FIGURE 4. The new proposed modular multiplier.

3.1. Design details

Based on the analysis given in this section, the modular multiplication problem has been converted into a multioperand addition of v vectors, each of n bits or less. The structure proposed to realize the modular multiplier algorithm of this paper is shown in Figure 4. It can be described as follows:

(i) A Carry-Save Adder (CSA) tree adds all the v vectors and expresses its output as a Sum, S, and a Carry, C. The corresponding binary representations are: $S = \sum_{i=0}^{n+L-1} s_i 2^i$ and $C = \sum_{i=0}^{n+L-1} c_i 2^i$, where: $L = \lceil \log_2 v \rceil$, s_i is the ith bit of S and c_i is the ith bit of C. S can be expressed as: $S = S_2 2^n + S_1$, where $S_1 = \sum_{i=0}^{n-1} s_i 2^i$ and $S_2 = \sum_{i=n}^{n+L-1} s_i 2^{i-n}$. Similarly, C can be expressed as: $C = C_2 2^n + C_1$, where $C_1 = \sum_{i=0}^{n-1} c_i 2^i$ and $C_2 = \sum_{i=n}^{n+L-1} c_i 2^{i-n}$. Therefore, $T = |S + C|_m$.

Equivalently, $T = |S_1 + C_1 + 2^n(S_2 + C_2)|_m$. Substituting $|2^n|_m = (2^p + 1)$, then $T = |S_1 + (2^p + 1)S_2 + C_1 + (2^p + 1)C_2)|_m$. Clearly, if p is selected such that $p \leq n/2$, then $L \leq \left\lceil \log_2\left(\frac{5}{2}n - 2\right)\right\rceil$. By simple mathematical inspection, this leads to $L \leq n/2$, when $n \geq 10$. Accordingly, the binary representation of $(2^p + 1)S_2$ is given by,

$$(2^p+1)S_2 \xrightarrow{\text{binary}} \overbrace{0\cdots0}^{(n-2p)\text{ bits}}\ \overbrace{0\cdots0 s_{n+L-1}s_{n+L-2}\cdots s_{n+1}s_n}^{p\text{ bits}} \times \overbrace{0\cdots0 s_{n+L-1}s_{n+L-2}\cdots s_{n+1}s_n}^{p\text{ bits}} \quad (28)$$

Similarly, the binary representation of $(2^p + 1)C_2$, is given by,

$$(2^p+1)C_2 \xrightarrow{\text{binary}} \overbrace{0\cdots0}^{(n-2p)\text{ bits}}\ \overbrace{0\cdots0 c_{n+L-1}c_{n+L-2}\cdots c_{n+1}c_n}^{p\text{ bits}} \times \overbrace{0\cdots0 c_{n+L-1}c_{n+L-2}\cdots c_{n+1}c_n}^{p\text{ bits}} \quad (29)$$

Since S_2 and C_2 are an indication of multiples of 2^n resulting from accumulating a total of v vectors, then the maximum value of S_2 or C_2 is less than v (each vector has a value less than 2^n). Thus,

$$\begin{aligned} (2^p+1)S_2 &< (2^p+1)v \\ (2^p+1)S_2 &< (2^{n/2}+1)\left(\tfrac{5}{2}n-2\right) \\ &< \tfrac{5}{2}n(2^{n/2}+1) - 2(2^p+1) \\ \text{but}\quad \tfrac{5}{2}n(2^{n/2}+1) &< 2^n, \quad \text{for } n \geq 10 \\ \text{then}\quad (2^p+1)S_2 &< 2^n - 2(2^p+1) \\ &< m - (2^p+1) \end{aligned}$$

Since $(2^p+1)S_2 < m-(2^p+1)$, then following a similar approach, $(2^p + 1)C_2 < m - (2^p + 1)$. Accordingly, T can be rewritten as:

$$T = |T_1 + T_2|_m \quad (30)$$

where

$$T_1 = \left|S_1 + (2^p+1)S_2\right|_m \quad (31)$$

$$T_2 = \left|C_1 + (2^p+1)C_2\right|_m \quad (32)$$

Since $0 \leq S_1 < 2^n$, then $0 \leq S_1 + (2^p+1)S_2 < 2m$. Similarly, since $0 \leq C_1 < 2^n$, then $0 \leq C_1 + (2^p + 1)C_2 < 2m$.

An n-bit modular adder (modular adder 1 in Figure 4) accepts S_1 and S_2 and evaluates T_1. Another modular adder (modular adder 2 in Figure 4) accepts C_1 and C_2 and evaluates T_2.

(ii) An n-bit modular adder (modular adder 3 in Figure 4), evaluates $|T_1 + T_2|_m$. The output of which is T.

Therefore, a CSA tree along with three modular adders are needed to realize the proposed multiplier. The design of the modular adder proposed in [26] is very suitable for this application.

3.2. Dealing with modulus $(2^n - (2^p - 1))$

The analysis introduced in Sections 2 and 3 so far has dealt with the modulus $(2^n - (2^p + 1))$. The same results in these sections are applicable for the case $(2^n - (2^p - 1))$. The only difference is that each L_i is now multiplied with $(2^p - 1)$ rather than $(2^p + 1)$. The value of each R_i, for which i is even, is multiplied by -1. This implies that (24) can be rewritten for the case of $(2^n - (2^p - 1))$ as,

$$T = \left|R_1 - R_2 + R_3 - R_4 \ \cdots - R_{2q} + R_{2q+1}\right|_m \quad (33)$$

Each R_k, for which i is even, can be expressed in two's complement notation as: $-R_k = \overline{R}_k + v_k$, where $\overline{R}_k$ is the ones complement of R_k. Thus, (33) can be rewritten as:

$$T = \left|R_1 + \overline{R}_2 + R_3 + \overline{R}_4 + \cdots + \overline{R}_{2q} + R_{2q+1} + |Q + G|_m\right|_m \quad (34)$$

where G is a precomputed operand whose numerical value depends on the ones complement of each R_i, where i is an even integer (the value is to be determined later on in this section); however, $Q = \sum_{k=2}^{2q} v_k$ (k is even) is the total number of ones added. For a value of $p \leq n/2$, $Q = n + p - 2$.

Therefore, all it takes to realize a modulo $(2^n - (2^p - 1))$ multiplier is to invert each even R_k, before applying it to the CSA tree and having an additional vector of value $|Q + G|_m$. The value of this additional vector is explained in the next subsection.

3.3. Study case

Evaluating T is a function of both values n and p. To have modulus-specific analysis, we choose $m = 991$. Thus, $m = 2^{10} - (2^5 + 1)$. Obviously, $n = 10$, while $p = 5$. Furthermore, considering the case in which $X = 662 = (1010010110)_2$ and $Y = 650 = (1010001010)_2$, then

$$Z = \begin{bmatrix} & & & & & & & & & z_{09} & z_{08} & z_{07} & z_{06} & z_{05} & z_{04} & z_{03} & z_{02} & z_{01} & z_{00} \\ & & & & & & & z_{19} & z_{18} & z_{17} & z_{16} & z_{15} & z_{14} & z_{13} & z_{12} & z_{11} & z_{10} & \\ & & & & & & z_{29} & z_{28} & z_{27} & z_{26} & z_{25} & z_{24} & z_{23} & z_{22} & z_{21} & z_{20} & & \\ & & & & & z_{39} & z_{38} & z_{37} & z_{36} & z_{35} & z_{34} & z_{33} & z_{32} & z_{31} & z_{30} & & & \\ & & & & z_{49} & z_{48} & z_{47} & z_{46} & z_{45} & z_{44} & z_{43} & z_{42} & z_{41} & z_{40} & & & & \\ & & & z_{59} & z_{58} & z_{57} & z_{56} & z_{55} & z_{54} & z_{53} & z_{52} & z_{51} & z_{50} & & & & & \\ & & z_{69} & z_{68} & z_{67} & z_{66} & z_{65} & z_{64} & z_{63} & z_{62} & z_{61} & z_{60} & & & & & & \\ & z_{79} & z_{78} & z_{77} & z_{76} & z_{75} & z_{74} & z_{73} & z_{72} & z_{71} & z_{70} & & & & & & & \\ z_{89} & z_{88} & z_{87} & z_{86} & z_{85} & z_{84} & z_{83} & z_{82} & z_{81} & z_{80} & & & & & & & & \\ z_{99} & z_{98} & z_{97} & z_{96} & z_{95} & z_{94} & z_{93} & z_{92} & z_{91} & z_{90} & & & & & & & & \end{bmatrix}$$

$$= \begin{bmatrix}
 & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & \\
 & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\
 & & & & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & & & \\
 & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & & \\
 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & & & & & & & \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & & & & \\
 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & & & & & & & &
\end{bmatrix}$$

$R_1 + R_2$

$$= \begin{bmatrix}
z_{09} & z_{08} & z_{07} & z_{06} & z_{05} & z_{04} & z_{03} & z_{02} & z_{01} & z_{00} \\
z_{18} & z_{17} & z_{16} & z_{15} & z_{14} & z_{13} & z_{12} & z_{11} & z_{10} & z_{19} \\
z_{27} & z_{26} & z_{25} & z_{24} & z_{23} & z_{22} & z_{21} & z_{20} & z_{29} & z_{28} \\
z_{36} & z_{35} & z_{34} & z_{33} & z_{32} & z_{31} & z_{30} & z_{39} & z_{38} & z_{37} \\
z_{45} & z_{44} & z_{43} & z_{42} & z_{41} & z_{40} & z_{49} & z_{48} & z_{47} & z_{46} \\
z_{54} & z_{53} & z_{52} & z_{51} & z_{50} & z_{59} & z_{58} & z_{57} & z_{56} & z_{55} \\
z_{63} & z_{62} & z_{61} & z_{60} & z_{69} & z_{68} & z_{67} & z_{66} & z_{65} & z_{64} \\
z_{72} & z_{71} & z_{70} & z_{79} & z_{78} & z_{77} & z_{76} & z_{75} & z_{74} & z_{73} \\
z_{81} & z_{80} & z_{89} & z_{88} & z_{87} & z_{86} & z_{85} & z_{84} & z_{83} & z_{82} \\
z_{90} & z_{99} & z_{98} & z_{97} & z_{96} & z_{95} & z_{94} & z_{93} & z_{92} & z_{91}
\end{bmatrix}$$

$$= \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1
\end{bmatrix}$$

$$R_3 = \begin{bmatrix}
- & - & - & - & z_{19} & - & - & - & - & - \\
- & - & - & z_{29} & z_{28} & - & - & - & - & - \\
- & - & z_{39} & z_{38} & z_{37} & - & - & - & - & \\
- & z_{49} & z_{48} & z_{47} & z_{46} & - & - & - & - & - \\
z_{59} & z_{58} & z_{57} & z_{56} & z_{55} & - & - & - & - & - \\
z_{68} & z_{67} & z_{66} & z_{65} & z_{64} & - & - & - & - & \\
z_{77} & z_{76} & z_{75} & z_{74} & z_{73} & - & - & - & - & - \\
z_{86} & z_{85} & z_{84} & z_{83} & z_{82} & - & - & - & - & - \\
z_{95} & z_{94} & z_{93} & z_{92} & z_{91} & - & - & - & - & -
\end{bmatrix}$$

$$= \begin{bmatrix}
- & - & - & - & 1 & - & - & - & - & - \\
- & - & - & 0 & 0 & - & - & - & - & - \\
- & - & 1 & 0 & 1 & - & - & - & - & - \\
- & 0 & 0 & 0 & 0 & - & - & - & - & - \\
0 & 0 & 0 & 0 & 0 & - & - & - & - & - \\
0 & 0 & 0 & 0 & 0 & - & - & - & - & - \\
1 & 0 & 0 & 1 & 0 & - & - & - & - & - \\
0 & 0 & 0 & 0 & 0 & - & - & - & - & - \\
0 & 1 & 0 & 1 & 1 & - & - & - & - & -
\end{bmatrix}$$

$R_4 + R_5$

$$= \begin{bmatrix}
- & - & - & - & z_{69} & - & - & - & - & z_{69} \\
- & - & - & z_{79} & z_{78} & - & - & - & z_{79} & z_{78} \\
- & - & z_{89} & z_{88} & z_{87} & - & - & z_{89} & z_{88} & z_{87} \\
- & z_{99} & z_{98} & z_{97} & z_{96} & - & z_{99} & z_{98} & z_{97} & z_{96}
\end{bmatrix}$$

$$= \begin{bmatrix}
- & - & - & - & 0 & - & - & - & - & 0 \\
- & - & - & 1 & 0 & - & - & - & 1 & 0 \\
- & - & 0 & 0 & 0 & - & - & 0 & 0 & 0 \\
- & 1 & 0 & 1 & 0 & - & 1 & 0 & 1 & 0
\end{bmatrix}$$

The outputs of the CSA tree that adds all the 23 vectors are:

$$S = \overbrace{00000}^{S_2}\overbrace{111101011}^{S_1}$$

$$C = \overbrace{00010}^{C_2}\overbrace{0010000000}^{C_1}$$

$$(2^5 + 1)S_2 = 0000000000$$

$$(2^5 + 1)C_2 = 0001000010$$

Accordingly,

$$T_1 = 0000001100$$

$$T_2 = 0011000010$$

$$T = 0011001110 \xrightarrow{\text{decimal}} 206$$

Consider now the case $m = 2^{10} - (2^5 - 1) = 993$. Obviously, $n = 10$, while $p = 5$, then

$$\overline{R}_2 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \overline{z}_{19} \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \overline{z}_{29} & \overline{z}_{28} \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & \overline{z}_{39} & \overline{z}_{38} & \overline{z}_{37} \\
1 & 1 & 1 & 1 & 1 & 1 & \overline{z}_{49} & \overline{z}_{48} & \overline{z}_{47} & \overline{z}_{46} \\
1 & 1 & 1 & 1 & 1 & \overline{z}_{59} & \overline{z}_{58} & \overline{z}_{57} & \overline{z}_{56} & \overline{z}_{55} \\
1 & 1 & 1 & 1 & \overline{z}_{69} & \overline{z}_{68} & \overline{z}_{67} & \overline{z}_{66} & \overline{z}_{65} & \overline{z}_{64} \\
1 & 1 & 1 & \overline{z}_{79} & \overline{z}_{78} & \overline{z}_{77} & \overline{z}_{76} & \overline{z}_{75} & \overline{z}_{74} & \overline{z}_{73} \\
1 & 1 & \overline{z}_{89} & \overline{z}_{88} & \overline{z}_{87} & \overline{z}_{86} & \overline{z}_{85} & \overline{z}_{84} & \overline{z}_{83} & \overline{z}_{82} \\
1 & \overline{z}_{99} & \overline{z}_{98} & \overline{z}_{97} & \overline{z}_{96} & \overline{z}_{95} & \overline{z}_{94} & \overline{z}_{93} & \overline{z}_{92} & \overline{z}_{91}
\end{bmatrix}$$

$$\overline{R}_4 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \overline{z}_{69} \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \overline{z}_{79} & \overline{z}_{78} \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & \overline{z}_{89} & \overline{z}_{88} & \overline{z}_{87} \\
1 & 1 & 1 & 1 & 1 & 1 & \overline{z}_{99} & \overline{z}_{98} & \overline{z}_{97} & \overline{z}_{96}
\end{bmatrix}$$

In $\overline{R}_2$ and $\overline{R}_4$, there are a total of 13 rows, thus, $Q = 13$. The string of ones in $\overline{R}_2$ and $\overline{R}_4$ can be removed. This allows the formation the two pairs: $(R_1, \overline{R}_2)$ and $(\overline{R}_4, R_5)$. The strings of ones removed form the binary representations of $\overline{R}_2$ and $\overline{R}_4$ can be precomputed modulo m. Assuming that this precomputed value modulo m is G, then the value $|Q + G|_m$ is applied as an additional input vector to the CSA tree. The computation for this case of $m = (2^n - (2^p - 1))$, is carried afterwards in the same way as the previous case of, $m = (2^n - (2^p + 1))$.

4. PERFORMANCE AND COMPARATIVE ANALYSIS

In order to evaluate the performance of the new proposed modular multiplier, its hardware requirements and time delay have to be evaluated. Thus, the hardware and delay of the CSA tree and the three modular adders are to be determined.

In the following analysis, it is assumed that $p \leq n/2$. This would simplify the analysis and give an upper bound for other cases of $p < n/2$. Under this assumption, $q = 2$. Accordingly, there are five Right summations, namely, R_1, R_2, R_3, R_4 and R_5. Three pairs are formed: (R_1, R_2), R_3 and (R_4, R_5). The vectors of each pair are added in a separate CSA sub-tree, thus, there are three parallel CSA sub-trees. The number of levels on a CSA tree as defined by the Avizienis recursive formula and summarized in Table 2 is given by: $\mathrm{d}(v) = 3\lfloor \mathrm{d}(v-1)/2 \rfloor + \mathrm{d}(v-1) \mod 2$, for $v = 2, 3, \ldots$, and initially $\mathrm{d}(1) = 3$ [29]. The pair $(R_1 + R_2)$ has a total of n n-bit vectors, thus, a CSA sub-tree adds these vectors and expresses the output as a sum and a carry. A total of $(n - 2)$ CSAs are needed. Equivalently, this sub-tree has a delay of $\mathrm{d}(n)$ FAs. The summation R_3 consisting of $(n - 1)$ vectors, where each is $(n - p)$ bits wide. Since it is assumed that $p \leq n/2$, then a maximum of $(n - 3)(n/2)$ FAs are needed for this sub-tree. Similarly, the pair (R_4, R_5) has a total of $((n/2) - 1)$ vectors each consists of n bits or less. Since R_4 and R_5 are identical, it is enough to compute one of them. The number of bits in each vector in R_4 or R_5 is less than $n/2$. A total of $((n/2) - 3)$ CSAs, or equivalently, $((n/2) - 3)(n/2)$ FAs are needed. The Sums and Carries from the three sub-trees are then added using four CSAs, to obtain the S and C of the CSA tree. This adds the delay of three FAs and the hardware of $4n$ FAs. Therefore, the total number of FAs needed by the CSA tree is $\frac{7}{4}n^2 - n$. The delay of the CSA tree is $(\mathrm{d}(n) + 3)\,\tau_{\mathrm{FA}}$, where τ_{FA} is the delay of a single-bit FA.

The modular adders used in Figure 4 can be realized using the structure proposed in [26]. It is composed of a binary CLA adder, n Half-adders (HAs), and n (2×1) MUXs. Assuming the area and delay of a (2×1) MUX are equivalent to an HA, then the n HAs and n MUXs are equivalent, roughly, to the area and delay of n FAs. The area and delay of an n-bit CLA adder are given by [30]: $n\lceil \log_2 n \rceil$ FA and $\left(\lceil \log_2 n \rceil + 1\right)\tau_{\mathrm{FA}}$ respectively. Thus, the area and the delay of the modular adder proposed in [26] are: $n\left(\lceil \log_2 n \rceil + 1\right)$ FAs and $\left(\lceil \log_2 n \rceil + 2\right)\tau_{\mathrm{FA}}$ respectively. Adding the delay in the two cascaded n-bit modular adders in Figure 4 would make the delay of the proposed modular multiplier equal to $(2\lceil \log_2 n \rceil + \mathrm{d}(n) + 7)\tau_{\mathrm{FA}}$. Similarly, adding the area of the three n-bit modular adders makes the total area of the proposed modular multiplier, in terms of the number of full adders, as: $\frac{7}{4}n^2 + 2n + 3n\lceil \log_2 n \rceil$.

TABLE 2. Avizienis recursive formula.

No. of levels, v	1	2	3	4	5	6	7	8	9	10
No. of vectors, $\mathrm{d}(v)$	3	4	6	9	13	19	28	42	63	94

TABLE 3. Area and delay comparison.

Multiplier	Area, # of FAs	Delay, τ_{FA}
Iterative array [31]	$(n-1)^2 + n\lceil \log_2 n \rceil$	$n + \lceil \log_2 n \rceil + 1$
Di Claudio *et al.* [16]	$\frac{5}{2}(n-1)^2 + n + \frac{7}{2}n\lceil \log_2 n \rceil$	$\frac{5}{2}n + \frac{7}{2}\lceil \log_2 n \rceil + 2$
Hiasat [18]	$\frac{5}{4}(n-1)^2 + 2n + \frac{9}{4}n\lceil \log_2 n \rceil$	$\frac{5}{4}n + \frac{9}{4}\lceil \log_2 n \rceil + 2$
Proposed	$\frac{7}{4}n^2 + 2n + 3n\lceil \log_2 n \rceil$	$2\lceil \log_2 n \rceil + d(n) + 7$

In this section, the new design will be compared with modular multipliers available in literature, that have relatively better performance. The performance of the design in [23] is less efficient than that in [24]. However, the performance of the design in [24] is similar to that of [31]. On the other hand, the structure of [19] is more suitable for small moduli. Moreover, the authors of [19] stated explicitly that their design is efficient in terms of area, but it is slower than that in [16]. Therefore, the new design will be compared with [16, 18, 31].

The modular-based multiplier proposed by Di Claudio *et al.* [16] requires two 2.5 $(n \times n)$ binary multipliers and one modular adder. However, the modular-based multiplier proposed by Hiasat [18] requires a 1.25 $(n \times n)$ binary multiplier, an n-bit CSA binary adder, and an n-bit modular adder. In the following analysis, it is assumed that all the three modular multipliers [16, 18], and the new proposed one use the same modular adder introduced recently in [26].

A binary-based iterative array multiplier [29, 32, 31], realized as carry-save arrays with final CLA addition, requires $(n - 1)^2$ FAs to produce the sum and carry of a CSA array, and an n-bit CLA. Table 3 shows the area and time delay for four multipliers. One of these is the binary-based iterative array multiplier [29, 32, 31], which was assumed to be a building block in realizing the modular multipliers in [16, 18]. The other three are modular-based ones, namely, the one proposed by Di Claudio *et al.* [16], the one proposed by Hiasat [18] and the one proposed in this paper. In all cases, the n^2 AND gates needed to create the partial product terms have been neglected. Table 4 shows the areas and delays of the four multipliers normalized to the iterative binary multiplier, for values of $n = 12, 16, 20$ and 24 bits. Although it requires a larger area compared with a binary-based iterative multiplier, the new modular

TABLE 4. Normalized areas and delays.

	$n = 12$		$n = 16$		$n = 20$		$n = 24$	
Multiplier	Area	Delay	Area	Delay	Area	Delay	Area	Delay
Iterative array [31]	1	1	1	1	1	1	1	1
Di Claudio *et al*. [16]	2.86	2.71	2.78	2.67	2.76	2.67	2.72	2.65
Hiasat [18]	1.68	1.53	1.58	1.48	1.55	1.47	1.51	1.44
Proposed	2.48	1.17	2.33	1.00	2.26	0.92	2.18	0.80

multiplier is faster for any $n > 16$. Comparison results in Table 4 show also that the new multiplier requires a larger area than that in [18], nevertheless, it is much faster.

5. CONCLUSIONS

In this paper, a new modulo $(2^n - (2^p \pm 1))$ multiplication algorithm is introduced. The algorithm compresses the magnitude of the partial products to be less than 2^n. The proposed hardware structure utilizes a CSA tree to add $(2n + p - 2)$ partial products. It also utilizes three n-bit modular adders to express the sum in a residue form. Due to the large number of partial products, the area of the multiplier is larger than another residue multiplier reported in literature. Nevertheless, the new residue multiplier has a logarithmic speed which makes it faster than any other similar multiplier. It is also faster than binary-based iterative array multipliers with a final CLA addition. The proposed design is most suitable for values of $p \leq n/2$.

REFERENCES

[1] Szabo, N. and Tanaka, R. (1967) *Residue Arithmetic and Its Applications to Computer Technology*. McGraw Hill, New York.

[2] Soderstrand, M., Jenkins, W., Jullien, G. and Taylor, F. (1986) *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, New York.

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

[4] Chren, W. (1999) Delta–sigma modulator with large OSR using one-hot residue number system. *IEEE Trans. Circ. Sys. II*, **46**, 1002–1008.

[5] Ramnarayan, A. (1980) Practical realization of mod p, p prime multiplier. *Electronic Lett.*, **16**, 466–467.

[6] Soderstrand, M. and Vernia, C. (1980) A high-speed low-cost modulo P_i multiplier with RNS arithmetic application. *Proc. IEEE*, **68**, 529–532.

[7] Jullien, G. (1980) Implementation of multiplication, modulo a prime number, with applications to theoretic transforms. *IEEE Trans Comp.*, **C-29**, 899–905.

[8] Taylor, F. (1982) A VLSI residue arithmetic multiplier. *IEEE Trans. Comp.*, **C-31**, 540–546.

[9] Hiasat, A. (1991) A memoryless mod $(2^n \pm 1)$ residue multiplier. *Electronic Lett.*, **28**, 414–415.

[10] Dugdale, M. (1994) Residue multipliers using factored decomposition. *IEEE Trans. Circ. Sys.-II Analog & Digital Signal Processing*, **41**, 623–627.

[11] Radhakrishnan, D. and Yuan, Y. (1992) Novel approaches to the design of VLSI RNS multipliers. *IEEE Trans. Circ. & Sys.-II: Analog & Digital Signal Processing*, **39**, 52–57.

[12] Alia, G. and Martinelli, E. (1991) A VLSI modulo m multiplier. *IEEE Trans. Comp.*, **C-40**, 873–878.

[13] Walter, C. (1993) Systolic modular multiplier. *IEEE Trans. Comp.*, **C-42**, 376–378.

[14] Parker, M. and Benaissa, M. (1994) VLSI structures for bit-serial modular multiplication using basis conversion. *IEE Proc.-Comput. Digit Tech.*, **141**, 245–251.

[15] Elleithy, K. and Bayoumi, M. (1995) A systolic architecture for modulo multiplication. *IEEE Trans. Circ. & Sys.-II Analog & Digital Signal Processing*, **42**, 725–729.

[16] Di Claudio, E., Piazza, F. and Orlandi, G. (1995) Fast combinatorial RNS processors for DSP applications. *IEEE Trans. Comp.*, **C-44**, 624–633.

[17] Bajard, J., Didier, L. and Kornerup, P. (1998) An RNS Montgomery modular multiplication algorithm. *IEEE Trans. Comp.*, **C-47**, 766–776.

[18] Hiasat, A. (2000) New efficient structure for a modular multiplier for RNS. *IEEE Trans Comp.*, **C-49**(2), 170–174.

[19] Paliouras, V., Karagianni, K. and Stouraitis, T. (2001) A low-complexity combinatorial RNS multiplier. *IEEE Trans. Circ. & Sys.-II*, **48**(7), 675–683.

[20] Savas, E. and Koc, C. (2000) The Montgomery modular inverse-revisited. *IEEE Trans. Comp.*, **49**(7), 763–766.

[21] Curiger, A., Bonnenberg, H. and Kaeslin, H. (1991) Regular VLSI architectures for multiplication modulo $(2^n + 1)$. *IEEE J. Solid-State Circ.*, **26**, 990–994.

[22] Wrzyszcz, A. and Milford, D. (1993) A new modulo $2^a + 1$ multiplier. In *Proc. ICCD'93*, Cambridge, MA, 21–23 October, pp. 614–617. IEEE Press, NY.

[23] Ma, Y. (1998) A simplified architecture for modulo $(2^n + 1)$ multiplication. *IEEE Trans. Comp.*, **C-47**, 333–337.

[24] Zimmermann, R. (1999) Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication. In *Proc. 14th IEEE Symposium on Computer Arithmetic*, Australia, 12–15 August. IEEE Press, NY.

[25] Rader, C. (1972) Discrete convolutions via Mersenne transforms. *IEEE Trans Comp.*, **21**, 1269–1273.

[26] Hiasat, A. (2002) High-speed and reduced-area modular adder structures for RNS. *IEEE Trans Comp.*, **51**, 84–89.

[27] Skavantzos, A. (1998) An efficient residue to weighted converter for a new residue number system. *Proc. 8th Great Lakes Symposium on VLSI*, Lafayette, LA, 15–17 February, pp. 185–191. IEEE Press, NY.

[28] Hiasat, A. (2003) Efficient residue to binary converter. *IEE Proc. CDT*, **150**, 11–16.

[29] Hwang, H. (1979) *Computer Arithmetic: Principles, Architecture and Design*. Wiley, New York.

[30] Lynch, T. and Swartzlander, E. (1992) A spanning tree carry lookahead adder. *IEEE Trans Comp.*, **41**, 931–939.

[31] Pekmestzi, K. (1999) Multiplexer-based array multipliers. *IEEE Trans Comp.*, **48**(1), 15–22.

[32] Waser, F. and Flynn, M. (1982) *Introduction to Arithmetic for Digital Systems Designers*. Holt, Rinehart and Winston, New York.