GRAPH-BASED OPTIMIZATION FOR A CSD-ENHANCED RNS MULTIPLIER

G. Dimitrakopoulos[†] *and V. Paliouras*[‡]

[†]Computer Engineering and Informatics Dept., University of Patras, 26 500, Patras, Greece
[‡]Electrical and Computer Engineering Dept., University of Patras, 26 500, Patras, Greece
e-mails: dimitrak@ceid.upatras.gr, paliuras@ee.upatras.gr

ABSTRACT

A novel hardware algorithm, architecture and an optimization technique for residue multipliers are introduced in this paper. The proposed architecture exploits certain properties of the bit products to achieve low-complexity implementation via a set of introduced theorems that allow the definition of a graph based design methodology. In addition the proposed multiplier employs the *Canonic Signed Digit* (CSD) encoding to minimize the number of bit products required to be processed. Performance data reveal that the introduced architecture achieves area$\times$time complexity reduction of up to 55%, when compared to the most efficient previously reported design.

1. INTRODUCTION

The Residue Number System (RNS) can efficiently perform computations in digital signal processing (DSP) applications such as the computation of the Fast Fourier Transform and digital filtering [1], since it provides carry-free addition and multiplication and borrow-free subtraction. Several very large scale integration (VLSI) architectures have been proposed for RNS multiplication, implemented, either by memory table lookup techniques, or combinatorial logic [2]–[7].

In this paper properties of bit products that need to be processed in a residue multiplier are investigated. The introduction of a corresponding set of theorems allows the definition of a graph-based design methodology that permits optimization, which was previously only partially possible and by means of exhaustive simulation [7]. The combined effect of an introduced hardware multiplication algorithm and the proposed methodology leads to area$\times$time efficient multipliers shown to outperform previous designs and offer an efficient alternative to binary multiplication for DSP applications.

2. RNS BASICS AND THE PROPOSED ALGORITHM

In RNS each operand is encoded as a vector of residues computed with respect to a set of pairwise relatively prime integers $\{m_1,m_2,\ldots,m_M\}$. All integers A,B with $0 \le A,B < \prod_{i=1}^{M} m_i$ have a unique RNS representation $\{A_1,A_2,\ldots,A_M\}$ and $\{B_1,B_2,\ldots,B_M\}$ where $A_i = \langle A\rangle_{m_i}$, $B_i = \langle B\rangle_{m_i}$ for $i = 1,2,\ldots,M$ and $\langle x\rangle_{m_i}$ denotes the operation x modulo m_i. The operations of residue multiplication, addition, and subtraction are performed by $C_i = \langle A_i \circ B_i\rangle_{m_i}$ and $\circ$ denotes any of the aforementioned operations. RNS is a carry-free number system in the sense that C_i can be computed with no information from the residues (A_j,B_j), $i \neq j$, thus enabling parallel operation on the residues A_i, B_i.

Let A, B, and C be n-bit residues modulo m, with $n = \lfloor \log_2 m\rfloor + 1$. Since A and B can be written in binary form as $A = \sum_{i=0}^{n-1} a_i 2^i$, $B = \sum_{j=0}^{n-1} b_j 2^j$ with $a_i, b_j \in \{0,1\}$, then the residue multiplication operation $C = \langle A\times B\rangle_m$ can be written as

$$C = \Big\langle \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j 2^{i+j} \Big\rangle_m = \Big\langle \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j \langle 2^{i+j}\rangle_m \Big\rangle_m = \langle Y\rangle_m .$$

where

$$Y = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j \langle 2^{i+j}\rangle_m . \tag{1}$$

Hence, the product C can be derived by mapping Y to its residue modulo m. The value of Y, as defined in (1), can be expressed in binary form as $Y = \sum_{k=0}^{n'-1} y_k 2^k$, where n' is the word length of the maximum value of Y, $n' = \lfloor \log_2 Y_{\max}\rfloor + 1$, and y_k denotes the kth bit of Y. The bits y_k of Y to which the bit product $a_i b_j$ contributes, according to (1), depend on the encoding scheme selected for the value $\langle 2^{i+j}\rangle_m$. Particularly, the bit weights $\langle 2^{i+j}\rangle_m$ can be encoded as follows

$$\langle 2^{i+j}\rangle_m = \sum_{k=0}^{n_c-1} \widehat{z}_{k,i,j} 2^k , \tag{2}$$

where $\widehat{z}_{k,i,j} \in \{\bar{1},0,1\}$ and n_c denotes the required word length for the encoding of $\langle 2^{i+j}\rangle_m$, with $n_c \ge n$. The word length n_c can be greater than n in the case that a signed-digit encoding is assumed for $\langle 2^{i+j}\rangle_m$. By replacing (2) in (1), it is obtained that

$$Y = \sum_{k=0}^{n_c-1}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a_i b_j\, \widehat{z}_{k,i,j}\, 2^k . \tag{3}$$

Eq. (3) reveals that depending on the value of $\widehat{z}_{k,i,j}$ being $\bar{1}$, 0, or 1, the bit product $a_i b_j$ should be subtracted, ignored, or added to the kth bit of Y.

Assume that the bit product $a_i b_j = 1$ and that it should be subtracted from the kth digital position, i.e., $\widehat{z}_{k,i,j} = \bar{1}$. In this case, the partial result $p = a_i b_j\, \widehat{z}_{k,i,j}\, 2^k = -2^k$ is a term of the multiple summations in (3) and it is written in n'-bit two's complement format, as follows

$$p = \overset{n'-1}{1}\;1 \ldots \overset{k+1}{1}\;\overset{k}{1}\;\overset{k-1}{0} \ldots \overset{0}{0} = 2^{n'} - 2^k . \tag{4}$$

Due to (4), in case that $\widehat{z}_{k,i,j} = \bar{1}$ and $a_i b_j = 1$, based on two's complement arithmetic, the addition of the term p to the sum (3) is equivalent to the modulo-$2^{n'}$ addition of the value $2^{n'} - 2^k$. Therefore the addition of $a_i b_j\, \bar{1}\, 2^k$ is equivalent to the two's complement addition of $p' = 2^{n'} - 2^k + \overline{a_i b_j} 2^k$. Thus, in order to evaluate (3), each term $a_i b_j\, \widehat{z}_{k,i,j}\, 2^k$ is encoded according to the values assumed by $\widehat{z}_{k,i,j} \in \{\bar{1},0,1\}$, as

$$a_i b_j\, \widehat{z}_{k,i,j}\, 2^k \rightarrow \begin{cases} \overline{a_i b_j}\, 2^k + q_k\,, & \text{if } \widehat{z}_{k,i,j} = \bar{1} \\ 0\,, & \text{if } \widehat{z}_{k,i,j} = 0\,, \\ a_i b_j 2^k\,, & \text{if } \widehat{z}_{k,i,j} = 1 \end{cases} \tag{5}$$

where $q_k = 2^{n'} - 2^k$. Due to (3) and (5), Y can be computed by

$Y = \left\langle Y' + Q \right\rangle_{2^{n'}}$, where Q is a cumulative additive constant, computed as $Q = \sum_{k,i,j} q_k$ and Y' is equal to

$$Y' = \sum_{k=0}^{n_c-1} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left(\sum_{\widehat{z}_{k,i,j}=1} a_i b_j + \sum_{\widehat{z}_{k,i,j}=\bar{1}} \overline{a_i b_j} \right) 2^k. \quad (6)$$

Initially Y' and Q are decomposed into the n-bit lower parts Y'_L and Q_L, and the $(n' - n)$-bit upper parts, Y'_H and Q_H, respectively, which give

$$Y = \left\langle (Y'_H + Q_H) 2^n + Y'_L + Q_L \right\rangle_{2^{n'}}. \quad (7)$$

It holds that $Y'_L + Q_L = c\,2^n + \langle Y'_L + Q_L \rangle_{2^n}$, where c is a 1-bit carry due to the n-bit addition of Y'_L and Q_L. As both Y'_L and Q_L are n-bit quantities, (7) is written as $Y = \langle c'\,2^{n'} + R + \langle Y'_L + Q_L \rangle_{2^n} \rangle_{2^{n'}}$, where $c' \in \{0, 1\}$ is the carry bit generated by the $(n' - n)$-bit addition of Y'_H, Q_H, and c. The term $R = \langle (Y'_H + Q_H + c) 2^n \rangle_{2^{n'}}$ can be expressed as $R = \langle (c'' 2^{n'-n} + R') 2^n \rangle_{2^{n'}}$, where c'' is a single-bit carry and R' is an $(n'-n)$-bit quantity, $R' = \langle Y'_H + Q_H + c \rangle_{2^{n'-n}}$. Equivalently, $R = \langle c'\,2^{n'} + R'\,2^n \rangle_{2^{n'}} = \langle R'\,2^n \rangle_{2^{n'}}$. Since $R' \leq 2^{n'-n} - 1$ then $R'\,2^n \leq 2^{n'} - 2^n$, hence it follows that $R \leq 2^{n'} - 2^n$. Therefore, since $0 \leq \langle Y'_L + Q_L \rangle_{2^n} \leq 2^n - 1$, it follows that $R + \langle Y'_L + Q_L \rangle_{2^n} \leq 2^{n'} - 1$, which implies that

$$Y = \left\langle \left(Y'_H + Q_H + c \right) 2^n \right\rangle_{2^{n'}} + \left\langle Y'_L + Q_L \right\rangle_{2^n}. \quad (8)$$

Eq. (8) reveals that Y can be evaluated as a simple addition, without the need for an external modulo operation, as in (7). The residue product C can be expressed as $C = \langle D + Y'' \rangle_m$ where $D = \left\langle \langle (Y'_H + Q_H + c) 2^n \rangle_{2^{n'}} \right\rangle_m$ and $Y'' = \langle Y'_L + Q_L \rangle_{2^n}$. The value of D is only dependent on the $n' - n$ most significant bits of Y' and the carry bit c, since Q is a constant. A simple combinational circuit can receive the $n' - n$ most significant bits of Y' and the bit c and compute the n-bit value D. Based on the modulo arithmetic property $\langle m - a \rangle_m = \langle -a \rangle_m$, the value of D is selected to be negative and encoded in $(n + 1)$-bit two's complement format. Therefore, since $-m \leq D \leq 0$ and $0 \leq Y'' \leq 2^n - 1$ it is derived that $-m \leq Y'' + D \leq 2^n - 1$, which implies that

$$-m < Y'' + D < 2m. \quad (9)$$

Inequalities (9) dictate that three cases can be distinguished in the computation of the sought residue product C:

$$C = \begin{cases} Y'' + D + m, & \text{if } -m \leq Y'' + D < 0 \\ Y'' + D, & \text{if } 0 \leq Y'' + D < m \\ Y'' + D - m, & \text{if } m \leq Y'' + D < 2m \end{cases}. \quad (10)$$

Therefore the computation of C by (10), requires the computation of Y' using (6), followed by the computation of D and Y''. An implementation of the proposed algorithm is shown in Fig. 1.

3. PROPERTIES OF BIT PRODUCTS

The legitimate values of the input residues A and B are bounded by the value of the modulo m. Therefore the combinations of the n-bit input operands that comprise at least one of the values $m, m + 1, \ldots, 2^n - 1$ do not occur. In the following, it is shown that the bit products $a_i b_j$ or $\overline{a_i b_j}$ that contribute to bit y_k of Y exhibit specific relationships, which can be exploited to lead to low-complexity designs. Definition 1 introduces the concept of *compatible* bit products and the subsequent theorems reveal the conditions that should be satisfied in order two bit products to be compatible without the need of exhaustive simulation as in [7].

Definition 1. *The bit products $a_i b_j$ and $a_k b_l$ are called compatible bit products when $a_i b_j + a_k b_l \leq 1$, for all legitimate input combinations of the residues A and B with $0 \leq i,j,k,l \leq n-1$ and $0 \leq A,B < m$.*

Compatible bit products are of practical interest since they can be added by means of a 2-input OR gate instead of a 1-bit adder, thus reducing the cost of addition and preventing the generation of a carry bit that would contribute to the more significant output bits.

Theorem 1. *The bit products $a_i b_j$ and $a_k b_l$ are compatible if and only if $2^i + 2^k \geq m \bigvee 2^j + 2^l \geq m$ and $i \neq k \vee j \neq l$.*

Proof: Initially assume that $a_i b_j$ and $a_k b_l$, are compatible, i.e., $a_i b_j + a_k b_l \leq 1$, for all legitimate values of the input residues A and B. Therefore there are no combinations of A and B for which $a_i b_j + a_k b_l > 1 \Leftrightarrow a_i b_j = 1 \bigwedge a_k b_l = 1$. It follows that

$$a_i = \bigwedge a_k = 1 \Leftrightarrow A = \sum_{r=0}^{n-1} a_r 2^r \geq a_i 2^i + a_k 2^k = 2^i + 2^k \quad (11)$$

$$b_j = 1 \bigwedge b_l = 1 \Leftrightarrow B = \sum_{r=0}^{n-1} b_r 2^r \geq b_j 2^j + b_l 2^l = 2^k + 2^l. \quad (12)$$

Eqs. (11) and (12) show that when $a_i b_j + a_k b_l \leq 1$ for all legitimate inputs, there are no combinations of A and B such that

$$A \geq 2^i + 2^k \bigwedge B \geq 2^j + 2^l, \quad (13)$$

for every residue modulo-m value of A and B. Since (13) is false, it follows that at least one of the conditions in (13) is false, or both. Hence, $2^i + 2^k \geq m \bigvee 2^j + 2^l \geq m$, because $0 \leq A, B < m$. Since the bit products $a_i b_j$ and $a_k b_l$ do not represent the same term, then the index i cannot be equal to k when j is equal to l and vice versa. Therefore, it follows that $i \neq k \vee j \neq l$. Next suppose that

$$2^i + 2^k \geq m \bigvee 2^j + 2^l \geq m, \quad (14)$$

with $i \neq k \vee j \neq l$. Since A, B are residues modulo-m then $A = \sum_{i=0}^{n-1} a_i 2^i < m$ and $B = \sum_{j=0}^{n-1} b_j 2^j < m$. Therefore,

$$a_i 2^i + a_k 2^k < m \quad \bigwedge \quad b_j 2^j + b_l 2^l < m. \quad (15)$$

In the case that $a_i = a_k = b_j = b_l = 1$ there is a contradiction between (14) and (15). So, it obvious that (14) and (15) are both satisfied when there is at least one of the (a_i, a_k) and (b_i, b_k) respectively, that is zero. In this case, at least one of $a_i b_j$ or $a_k b_l$ is also equal to zero and thus their sum is $a_i b_j + a_k b_l \leq 1$, since it is either zero or one. ☐

Theorem 2. *If $a_i b_j$ and $a_k b_l$ are compatible bit products then the bit products $a_j b_i$ and $a_l b_k$ are also compatible.*

Proof: For all legitimate input combinations of input residues A and B assume that $a_i b_j + a_k b_l \leq 1 \Leftrightarrow b_j a_i + b_l a_k \leq 1$, due to the commutativity of the multiplication. Since A and B span the identical set of legitimate values $\{0, 1, \ldots, m-1\}$, by interchanging the names of the variables A and B along with the corresponding bits a and b it follows that $a_j b_i + a_l b_k \leq 1$ for all legitimate input values. Hence, the bit products $a_j b_i$ and $a_l b_k$ are compatible. ☐

The definition of compatible bit-product pairs can be generalized to compatible N-tuples of bit products, the sum of which is always less or equal to one. A compatible N-tuple can be processed by an N-input OR gate, thus significantly reducing the cost of N-bit addition and eliminating the generation of the corresponding carries. In the following, a theorem is introduced that identifies compatible N-tuples exploiting the existence of particular compatible bit-product pairs.

Theorem 3. *Let $x_1, x_2, \ldots, x_N$, with $x_i \in \{0, 1\}$ and $1 \leq i \leq N$. Then $x_1 + x_2 + \ldots + x_N \leq 1$ if and only if for all possible pairs x_i, x_j with $1 \leq i, j \leq N$ and $i \neq j$ holds that $x_i + x_j \leq 1$.*

The proposed residue multiplication algorithm requires the computation of Y' using (6), which includes summation of inverted bit products $\overline{a_i b_j}$. Therefore properties of inverted bit products are investigated, with the objective to minimize the cost

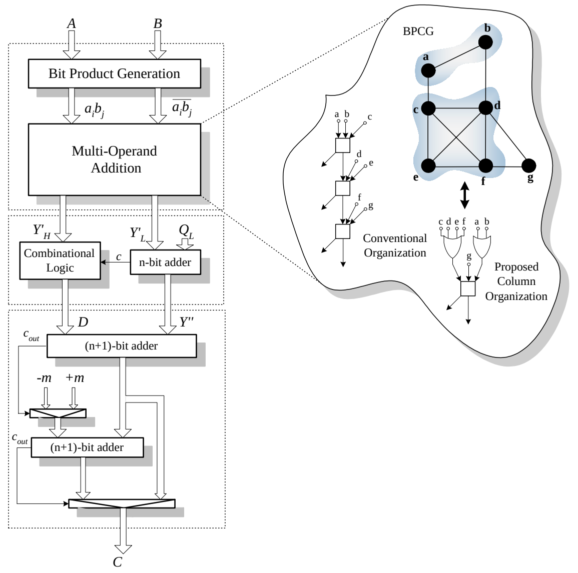

Figure 1: The three-stage residue multiplier. The complexity of the first stage is minimized by the proposed BPCA.

of their addition. Theorem 4 introduces a property of inverted bit-product N-tuples that their sum assumes a limited set of values while their identification is transformed to the problem of identifying a corresponding compatible bit-product N-tuple.

Theorem 4. *Let* $x_1,x_2,\ldots,x_N$, *with* $x_i \in \{0,1\}$ *and* $1 \le i \le N$. *If* $x_1 + x_2 + \ldots + x_N \le 1$ *then* $N-1 \le \bar{x}_1 + \bar{x}_2 + \ldots + \bar{x}_N \le N$.

4. ORGANIZATION OF THE PROPOSED MULTIPLIER

The proposed residue multiplier is organized in three stages as shown in Fig. 1. The first stage computes Y' defined by (6), using cascaded columns of 1-bit adders, organized in a way that the kth column returns the kth output bit of Y'. The second stage processes the bits of Y' and the cumulative constant Q to calculate D and Y'', while the third stage performs the final residue mapping via the conditional correction described by (10). In particular, to derive the kth bit of Y', two sets $\mathcal{S}_k$ and $\mathcal{N}_k$ are constructed that consist of the bit products of the form a_ib_j and $\overline{a_ib_j}$ respectively, that contribute to the kth bit of Y'. Formally stated, $\mathcal{S}_k = \{a_ib_j \mid \widehat{z}_{k,i,j} = 1\}$ and $\mathcal{N}_k = \{\overline{a_ib_j} \mid \widehat{z}_{k,i,j} = \bar{1}\}$, where $i,j \in \{0,1,\ldots,n-1\}$. Exploiting the data-dependent properties of the bit products of the sets $\mathcal{S}_k$ and $\mathcal{N}_k$, as described in Section 3, allows the use of simple gates to implement the multi-bit addition on each output column. The discussion is limited to the carry-save array organization according to which on the kth column, the bit products of the sets $\mathcal{S}_k$ and $\mathcal{N}_k$ are added along with the carries generated from the $(k-1)$st column according to the following design rules:

I. Every N bit products of the form $a_{i_k}b_{j_k}$ identified to be compatible are added with an N-input OR gate.

II. Every triplet of inverted bit products that their sum assumes the values two or three, i.e., $2 \le \overline{a_{i_1}b_{j_1}} + \overline{a_{i_2}b_{j_2}} + \overline{a_{i_3}b_{j_3}} \le 3$ are added using a 3-input AND gate. Since the addition of the bit products $\overline{a_{i_1}b_{j_1}}$, $\overline{a_{i_2}b_{j_2}}$, and $\overline{a_{i_3}b_{j_3}}$ on the kth column always generates a carry bit, the weight 2^{k+1} is added to the cumulative constant Q. In case that more than three inverted bit products are identified using Theorem 4, the proposed architecture uses the maximal number of 3-input AND gates to add them in order to limit the corresponding carry propagation to the next more significant column only.

III. The output of the OR gates and the AND gates along with the remaining bit products and the carry signals are added using 1-bit full-adder (FA) and half-adder (HA) cells. The carries of the $(k-1)$st column are added as close as possible to the output of the kth column so that the delay of the carry-save array is minimized.

5. THE PROPOSED OPTIMIZATION METHODOLOGY

This section introduces a novel graph-based optimization methodology, the application of which reduces the hardware complexity of the first stage of the proposed architecture. The proposed optimization methodology is twofold. At first the weights $\langle 2^{i+j}\rangle_m$ are encoded in CSD format in order to minimize the number of bit products of both forms a_ib_j and $\overline{a_ib_j}$, distributed on the kth column. Subsequently large compatible tuples are efficiently identified via an introduced graph-theoretic approach, called *Bit Product Compatibility Analysis* (BPCA). The CSD representation of the weights $\langle 2^{i+j}\rangle_m$ is of particular interest since it requires the minimal number of non-zero digits to represent a value, thus enabling the distribution of the bit products a_ib_j to the minimal number of output columns. CSD encoding of a given number is unique and a simple algorithm can be employed to convert conventional binary representation to CSD representation [8, p. 507].

The proposed BPCA methodology organizes the bits that participate on the kth column into N-tuples so that the number of terms added to produce the kth bit of Y' is minimized. The identification of the maximal N-tuples that exist in each output column is achieved by a graph theoretic formulation. Specifically, for each kth column an undirected graph $G_k(V_k,E_k)$ is constructed, called the *Bit-Product Compatibility Graph* (BPCG). The set of bit products of $\mathcal{S}_k$ constitute the vertex set of G_k, while an edge (u,v) belongs in E_k when the bit products that correspond to the vertices u and v are compatible. After constructing the BPCG for each output column, the identification of N-tuples is equivalent to the clique partitioning problem, see Fig. 1. In particular, a complete subgraph $G_s(V_s,E_s)$ with $V_s \subseteq V_k$ and $E_s \subseteq E_k$, has the property that for each vertex $u_i \in V_s$ there exist edges (u_i,u_j) connecting u_i to the remainder of the vertices $u_j \in V_s$, with $u_i \neq u_j$. Based on the way graph G_k is constructed, it follows that each bit product represented by the vertex $u_i \in V_s$ is compatible to all the other bit products with equivalent vertices in G_s. Hence for all possible bit-product pairs $(a_{i_k}b_{j_k},\ a_{i_l}b_{j_l})$ with $1 \le k,l \le |V_s|$ and $k \neq l$, that have their representative vertices u_k and u_l in G_s, it holds that $a_{i_k}b_{j_k} + a_{i_l}b_{j_l} \le 1$. So, from Theorem 3 it follows that $\sum_{r=1}^{|V_s|} a_{i_r}b_{j_r} \le 1$. Thus the $|V_s|$ bit-product terms $a_{i_r}b_{j_r} \in \mathcal{S}_k$, form a compatible $|V_s|$-tuple. Equivalently, to identify the N-tuples of inverted bit products $\overline{a_{i_r}b_{j_r}} \in \mathcal{N}_k$ the sum of the elements of which assume a limited set of values, the following procedure is proposed. The BPCG G'_k that corresponds to the bit products $a_{i_r}b_{j_r}$, $\overline{a_{i_r}b_{j_r}} \in \mathcal{N}_k$, is initially constructed. The solution of clique partitioning on G'_k [9, pp. 64–67], derives N-tuples such that $\sum_{r=0}^{N-1} a_{i_r}b_{j_r} \le 1$, $\overline{a_{i_r}b_{j_r}} \in \mathcal{N}_k$. Therefore, due to Theorem 4 it follows that $N-1 \le \sum_{r=0}^{N-1} \overline{a_{i_r}b_{j_r}} \le N$; hence any three bit products of these N-tuples can be processed by an AND gate.

6. PERFORMANCE EVALUATION

In this section the performance of the proposed residue multiplier is compared to that of previously reported architectures in terms of area×time complexity. At first the performance of the proposed

n_{bin}	m_i	Area			Time			Area×Time
		RNS	Binary	Savings(%)	RNS	Binary	Savings(%)	Savings(%)
10	$\{11, 12, 13\}$	474	780	39.23%	31	36	13.88%	47.67%
12	$\{11, 17, 24\}$	557	1140	51.14%	36	44	18.18%	60.02%
14	$\{24, 23, 31\}$	707	1568	54.91%	38	52	26.92%	67.05%
16	$\{47, 48, 31\}$	922	2064	55.32%	45	60	25.00%	66.49%
24	$\{129, 257, 511\}$	1979	4728	58.14%	63	92	31.50%	71.33%

Table I: Area, time, and area×time performance of a binary multiplication, compared to proposed residue multiplication.

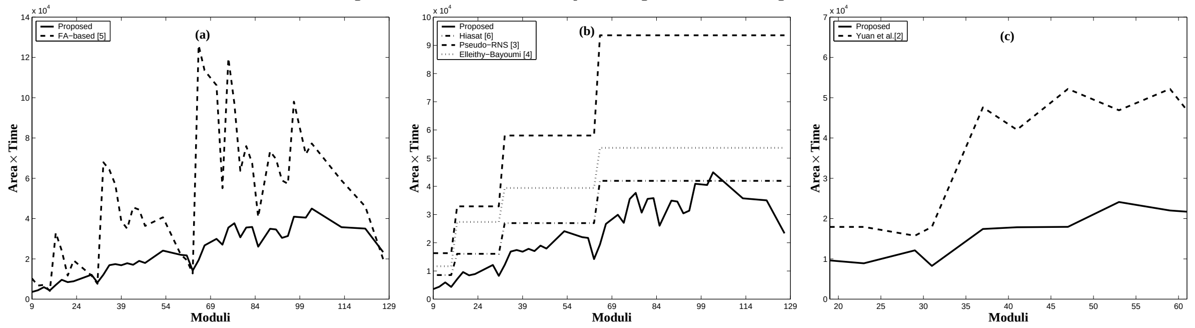

Figure 2: The area×time performance of the proposed residue multiplier compared to the architectures presented in [2]– [6].

multiplier is compared to that of a FA-based residue multiplier designed according to the architecture presented in [5]. Fig. 2a present the area×time performance of the proposed and the FA-based residue multipliers. In almost all cases the proposed architecture is more efficient than the FA-based residue multiplier and the area×time savings achieved span 22% to 85%.

In the following the proposed residue multiplier is compared to the architectures presented in [3], [4], and [6]. Fig. 2b reveals that the proposed methodology leads to low-complexity residue multipliers since both the number of the bit products that contribute to each output column is minimized due to CSD and the organization of each output column is simplified by means of the BPCA approach. The area×time savings achieved when compared to the most competitive residue multiplier [6] range from 18% to 55%. The proposed multiplier is finally compared to the submodular index transform-based multiplier [2] as shown in Fig. 2c. In all cases, the proposed multiplier is more efficient in terms of area×time complexity achieving complexity savings of more than 55%, while its application is not limited to prime moduli as required by the index-transform multipliers.

The proposed BPCA utilizes introduced bit-product properties to reduce the complexity of RNS multiplication. BPCA combined with the proposed hardware algorithm, presented in Section 2, leads to residue multipliers that can be used as a building block of multi-modulus RNS multipliers. The derived multi-modulus multipliers compare favorably to conventional binary multipliers of equivalent word length and architecture, as shown in Table I. Table I shows that substantial savings in both area and time complexity are achieved by exploiting the proposed residue multipliers and particular choices of moduli. Therefore the use of RNS and the proposed multiplier, leads to acceleration of the multiplication, as well as to hardware reduction of up to 58% for the particular word-lengths n_{bin}. It is stressed that the above analysis does not include the cost of the conversion from/to a binary representation. The target application domain, i.e., DSP algorithms, require one conversion per data sample, and substantial amount of processing, thus resulting in performance improvement, which can compensate the conversion overhead.

7. CONCLUSIONS

A novel residue multiplier is proposed in this paper. Initially, a bit-level algorithm is introduced, which features the design option of signed-digit encoding of the bit-product weights. The complexity minimization is achieved by using a proposed optimization methodology, based on an assortment of introduced theorems, which allow the exploitation of data-dependent characteristics of the bit products, in a computationally efficient manner. Complexity comparisons reveal that the derived residue multiplier is very efficient in the area×time sense, compared to previously reported designs.

8. REFERENCES

[1] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, 1986.

[2] D. Radhakrishnan and Y. Yuan, "Novel approaches to the design of VLSI RNS multipliers," *IEEE Trans. Circuits Syst. II*, vol. 39, pp. 52–57, Jan. 1992.

[3] E. D. DiClaudio, F. Piazza, and G. Orlandi, "Fast combinatorial RNS processors for DSP applications," *IEEE Trans. Comp.*, vol. 44, pp. 624–633, May 1995.

[4] K. M. Elleithy and M. A. Bayoumi, "A systolic architecture for modulo multiplication," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 725–729, Nov. 1995.

[5] 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.

[6] A. A. Hiasat, "New efficient structure for modular multiplier for RNS," *IEEE Trans. Comp.*, vol. 49, pp. 170–174, Feb. 2000.

[7] V. Paliouras and T. Stouraitis, "Multifunction architectures for RNS processors," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 1041–1054, Aug. 1999.

[8] K. Parhi, *VLSI Digital Signal Processing Systems*. Wiley, 1999.

[9] G. DeMichelli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.