A Novel Architecture and a Systematic Graph-Based Optimization Methodology for Modulo Multiplication

Giorgos Dimitrakopoulos and Vassilis Paliouras, *Member, IEEE*

***Abstract*—A novel hardware algorithm, a VLSI architecture, and an optimization methodology for residue multipliers are introduced in this paper. The proposed design approach identifies certain properties of the bit products that participate in the residue product computation and subsequently exploits them to reduce the complexity of the implementation. A set of introduced theorems is used to identify the particular properties. The introduced theorems are of significant practical importance because they allow the definition of a graph-based design methodology. In addition, a bit-product weight encoding scheme is investigated in a systematic way, and exploited in order to minimize the number of bit products processed in the proposed multiplier. Performance data reveal that the introduced architecture achieves area × time complexity reduction of up to 55%, when compared to the most efficient previously reported design.**

***Index Terms*—Computer arithmetic, design methodology, modulo multiplication, residue multiplier, residue number system (RNS).**

I. Introduction

RESIDUE or equivalently modulo multiplication plays an essential role in residue number system (RNS) applications [1]–[3], and in cryptographic systems [4]. Several very-large-scale-integration (VLSI) architectures have been proposed for the design of residue multipliers, implemented, by memory table lookup techniques, combinatorial logic, or a combination of both.

For small moduli, ROM-based architectures result in efficient implementations. However, the particular architectures become inefficient for larger moduli since the table-lookup size grows exponentially with the input word length. The cost of the memory storage for residue multiplication can be reduced by restricting the allowed moduli to be prime numbers thus allowing the application of the index transform [5] or the submodular index transform [6].

A variety of adder-based architectures have been proposed in the literature, offering low area × time complexity compared to the ROM-based structures. In particular, DiClaudio *et al.* introduced the pseudoRNS representation, which enables building reprogrammable modulus multipliers and simplifies the computation of DSP algorithms such as finite-impulse response (FIR) filtering, correlations, and discrete Fourier transform (DFT) [7]. Elleithy and Bayoumi presented an architecture for modular multiplication, which consists mainly of modular adders and is suitable for medium and large moduli [8]. Stouraitis *et al.* [9] introduced full-adder (FA) based architectures for RNS multiply–add operations, which adopt the carry–save array paradigm, while the same architecture has been refined by Soudris *et al.* in [10]. In [11], Paliouras and Stouraitis introduced the multifunction RNS architectures, which enable adder-based structures to perform several residue-arithmetic operations and conversions for various moduli simultaneously. Multifunction architectures are based on the concept of sharing hardware among units, which perform an operation with regard to different moduli, while nonoccuring input combinations are exploited to reduce the hardware complexity of the architecture. In particular, hardware simplification is achieved by reducing certain 1-bit adders to logic OR gates. The particular simplification has been applied by Paliouras *et al.* to the design of adder-based single-modulus residue multipliers [12]. The main drawback of the methodology applied in both [11] and [12] is that it relies on exhaustive simulations of all possible input combinations to identify input bit-product pairs or triplets that cannot be asserted simultaneously. Additionally, the corresponding architectures include a recursive modulo-reduction stage, formed by cascaded adders, which significantly affects the overall delay. Finally, Hiasat introduced a modular multiplication architecture highly efficient in terms of area and time performance [13].

Additionally, signed-digit based architectures have been proposed to achieve higher performance for adder-based arithmetic circuits [14]. The signed-digit notation allows several encodings of a given number among which the canonic-signed-digit (CSD) representation, which uses the digit set $\{\bar{1}, 0, 1\}$, where $\bar{1} = -1$, requires the minimal number of nonzero digits [15]. Recently, Hartley applied the particular property in the design of fixed-coefficient digital filters [16], while Wrzyszcz *et al.* introduced a technique to derive fixed-coefficient RNS filters [17]. The particular technique exploits the cyclic properties of residue arithmetic and its efficiency depends on the modulus of operation. In the same direction, Piestrak has proposed the design of multioperand modulo adders that exploit the periodicity of the $\langle 2^k \rangle_m$ bit-weight sequence [18].

This paper introduces an architecture that exploits both the signed-digit representation of the bit-product weight sequence $\langle 2^k \rangle_m$ and newly found properties of the bit products in order to reduce the hardware complexity of the residue multiplier. In particular, the bit-weight sequence $\langle 2^k \rangle_m$, which dictates the digital positions to which input bit products should be added, is encoded in an innovative way that minimizes the number of input

Manuscript received June 18, 2002; revised August 22, 2003. This paper was recommended by Associate Editor K. Parhi.

G. Dimitrakopoulos is with the Computer Engineering and Informatics Department, University of Patras, Patras GR 26500, Greece (e-mail: dimitrak@ceid.upatras.gr).

V. Paliouras is with the Electrical and Computer Engineering Department, University of Patras, Patras GR 26500, Greece (e-mail: paliuras@ee.upatras.gr).

Digital Object Identifier 10.1109/TCSI.2003.820243

bit products processed by the residue multiplier. The final encoding of the weights $\langle 2^k \rangle_m$ is selected by means of a proposed biased-signed-digit encoding (BSDE) technique. Furthermore, the introduced architecture does not require recursive modulo decomposition, achieving in this way a significant acceleration over previously reported designs [11], [12].

Additionally, this paper introduces a systematic optimization procedure, called the bit-products compatibility analysis (BPCA), which replaces the simulation-based design approach applied in [11] and [12]. BPCA exploits a set of introduced theorems that reveal several properties of the bit products processed in a modulo multiplier. In particular, the problem of identifying disjoint sets of bit products, that their sum is less or equal to one, for all legitimate input combinations, is formulated using an introduced graph, called the bit-product compatibility graph (BPCG), that enables exact solutions to be reached in efficient computational time. To the authors' knowledge the particular simplification was only partially possible until now, and only by means of exhaustive simulation. An innovative feature of the proposed methodology is also the exploitation of triplets of inverted bit products, the sum of which can only assume values in the set {2,3}, with the objective to reduce the complexity of certain 1-bit adder cells. The combined application of the proposed bit-weight encoding and the proposed BPCG-based optimization leads to residue multipliers of significantly reduced complexity when compared to existing designs. Area × time complexity savings of up to 55% are achieved, when compared to the most competitive residue multiplier [13].

The remainder of the paper is organized as follows. Section II describes the proposed residue multiplication algorithm, while Section III introduces particular properties of bit products that appear in residue multiplication. Section IV describes the proposed hardware organization. In Section V, an optimization methodology is introduced, which reduces the complexity of the multiplier. The hardware complexity of the proposed multiplier is analyzed in Section VI, and the obtained performance is compared against previously reported results in Section VII. Finally, conclusions are drawn in Section VIII.

II. Proposed Residue Multiplication Algorithm

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

$$C = \left\langle \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \, 2^{i+j} \right\rangle_m = \left\langle \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \langle 2^{i+j} \rangle_m \right\rangle_m \tag{1}$$

and $\langle x \rangle_m$ denotes the operation x modulo m. Assuming that Y denotes the nested summations in (1), i.e.,

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

the residue product C equals to $C = \langle Y \rangle_m$. Hence, due to (1) and (2), the product C can be derived by mapping Y to its residue modulo m. The value of Y, as defined in (2), can be expressed in binary form as

$$Y = \sum_{k=0}^{n'-1} y_k 2^k \tag{3}$$

where n' is the word length of the maximum value $Y_{\max}$ assumed by Y, $n' = \lfloor \log_2 Y_{\max} \rfloor + 1$, and $y_k \in \{0, 1\}$ denotes the kth bit of Y. The bits y_k of Y to which the bit product $a_i b_j$ contributes depend on the encoding scheme selected for the bit-product weight $\langle 2^{i+j} \rangle_m$. Particularly, the bit-product weights $\langle 2^{i+j} \rangle_m$ can be encoded as follows:

$$\langle 2^{i+j} \rangle_m = \sum_{k=0}^{n_c-1} \widehat{z}_{k,i,j} 2^k \tag{4}$$

where $\widehat{z}_{k,i,j} \in \{\overline{1}, 0, 1\}$ and n_c denotes the required word length for the encoding of $\langle 2^{i+j} \rangle_m$, with $n_c \geq n$. 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 (4) in (2), it is obtained that Y can be expressed as

$$Y = \sum_{k=0}^{n_c-1} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \, \widehat{z}_{k,i,j} \, 2^k. \tag{5}$$

Expression (5) reveals that depending on the value of $\widehat{z}_{k,i,j}$ being $\overline{1}$, 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 $a_i b_j$ is asserted, i.e., $a_i b_j = 1$, and that it should be subtracted from the kth digital position, i.e., $\widehat{z}_{k,i,j} = \overline{1}$. 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 (5) and it is written in n'-bit two's complement format as follows:

$$p = \overset{n'-1}{1} \; \overset{}{1} \ldots \overset{k+1}{1} \; \overset{k}{1} \; \overset{k-1}{0} \; \overset{}{\ldots} \; \overset{0}{0} = 2^{n'} - 2^k \,. \tag{6}$$

Due to (6), in case that $\widehat{z}_{k,i,j} = \overline{1}$ and $a_i b_j = 1$, based on the properties of n'-bit two's complement arithmetic, the contribution of the term p to the sum (5) is equivalent to the modulo-$2^{n'}$ addition of the value $2^{n'} - 2^k$. Therefore the addition of $a_i b_j \, \overline{1} \, 2^k$ is equivalent to the n'-bit two's complement addition of p' defined as

$$p' = 2^{n'} - 2^k + \overline{a_i b_j} 2^k \tag{7}$$

since the following two cases can be distinguished.

1) If $a_i b_j = 1$, p' of (7) is reduced to $2^{n'} - 2^k$, which represents $a_i b_j \, \overline{1} = -a_i b_j$.
2) If $a_i b_j = 0$, p' of (7) is reduced to $2^{n'} - 2^k + 2^k = 2^{n'}$, which represents zero, because the bit weight $2^{n'}$ is ignored in n'-bit two's complement addition.

Therefore, in order to evaluate (5) using two's complement arithmetic, the term $a_i b_j \, \widehat{z}_{k,i,j} \, 2^k$ is encoded according to the three possible values assumed by $\widehat{z}_{k,i,j} \in \{\bar{1}, 0, 1\}$, as

$$a_i b_j \, \widehat{z}_{k,i,j} \, 2^k \rightarrow \begin{cases} \overline{a_i b_j} \, 2^k + q_k, & \text{if } \widehat{z}_{k,i,j} = \bar{1} \\ 0, & \text{if } \widehat{z}_{k,i,j} = 0 \\ a_i b_j 2^k, & \text{if } \widehat{z}_{k,i,j} = 1 \end{cases} \tag{8}$$

where $q_k = 2^{n'} - 2^k$. Due to (5) and (8), Y can be computed by

$$Y = \Bigg\langle \underbrace{\sum_{k=0}^{n_c-1} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \, 2^k}_{k,i,j:\widehat{z}_{k,i,j}=1} + \underbrace{\sum_{k=0}^{n_c-1} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \overline{a_i b_j} \, 2^k}_{k,i,j:\widehat{z}_{k,i,j}=\bar{1}} + Q \Bigg\rangle_{2^{n'}} \tag{9}$$

where Q is a cumulative additive constant, computed as

$$Q = \sum_{\substack{k,i,j: \\ \widehat{z}_{k,i,j}=\bar{1}}} q_k = \sum_{\substack{k,i,j: \\ \widehat{z}_{k,i,j}=\bar{1}}} (2^{n'} - 2^k) \tag{10}$$

and $k, i, j : \widehat{z}_{k,i,j} = 1$ or $\bar{1}$ denotes the values of k, i, j such that $\widehat{z}_{k,i,j} = 1$ or $\bar{1}$, respectively. It should be noted that when the bit-product weights $\langle 2^{i+j} \rangle_m$ are encoded using the digit set {0,1}, i.e., $\widehat{z}_{k,i,j} \neq \bar{1}$, for each $0 \leq i, j \leq n-1$ and $0 \leq k \leq n_c - 1$, then, the cumulative constant Q is zero, implying that no correction needs to be performed. Equation (9) reveals that the number of bit products that contribute to each y_k is directly dependent on the number of the nonzero digits $\widehat{z}_{k,i,j}$ in the encoding of the bit weights $\langle 2^{i+j} \rangle_m$. Therefore, the number of bit products $a_i b_j$ and/or inverses $\overline{a_i b_j}$ that need to be added to compute Y, can be minimized by properly encoding the bit weights $\langle 2^{i+j} \rangle_m$. A procedure to select the encoding scheme for $\langle 2^{i+j} \rangle_m$ that minimizes the nonzero digits $\widehat{z}_{k,i,j}$, is introduced in Section V.

Equation (9) can be expressed as

$$Y = \Big\langle Y^* + Q \Big\rangle_{2^{n'}} \tag{11}$$

where Y^* is defined by

$$Y^* = \underbrace{\sum_{k=0}^{n_c-1} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \, 2^k}_{k,i,j:\widehat{z}_{k,i,j}=1} + \underbrace{\sum_{k=0}^{n_c-1} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \overline{a_i b_j} \, 2^k}_{k,i,j:\widehat{z}_{k,i,j}=\bar{1}} . \tag{12}$$

Equation (11) is computed as follows. Initially the n'-bit quantities 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, as follows:

$$Y^* = Y_H^* 2^n + Y_L^* \tag{13}$$

$$Q = Q_H 2^n + Q_L \tag{14}$$

which, when replaced in (11), give

$$Y = \Big\langle (Y_H^* + Q_H) 2^n + Y_L^* + Q_L \Big\rangle_{2^{n'}} . \tag{15}$$

It holds that

$$Y_L^* + Q_L = c \, 2^n + \Big\langle Y_L^* + Q_L \Big\rangle_{2^n} \tag{16}$$

where $c \in \{0, 1\}$ is a 1-bit carry due to the n-bit addition of Y_L^* and Q_L. Due to (16) and since both Y_L^* and Q_L are n-bit quantities, (15) is written as

$$Y = \Big\langle (Y_H^* + Q_H + c) \, 2^n + \Big\langle Y_L^* + Q_L \Big\rangle_{2^n} \Big\rangle_{2^{n'}} \tag{17}$$

$$= \Big\langle c' \, 2^{n'} + \Big\langle (Y_H^* + Q_H + c) \, 2^n \Big\rangle_{2^{n'}} + \Big\langle Y_L^* + Q_L \Big\rangle_{2^n} \Big\rangle_{2^{n'}} \tag{18}$$

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 $D = \Big\langle (Y_H^* + Q_H + c) \, 2^n \Big\rangle_{2^{n'}}$ of (18) is written as

$$D = \Big\langle (c'' \, 2^{n'-n} + D') 2^n \Big\rangle_{2^{n'}} \tag{19}$$

where $c'' \in \{0, 1\}$ is a single-bit carry and D' is an $(n' - n)$-bit quantity, $D' = \Big\langle Y_H^* + Q_H + c \Big\rangle_{2^{n'-n}}$. Equivalently, $D = \Big\langle c'' \, 2^{n'} + D' \, 2^n \Big\rangle_{2^{n'}} = \Big\langle D' \, 2^n \Big\rangle_{2^{n'}}$. Since $D' \leq 2^{n'-n} - 1$ then $D' \, 2^n \leq 2^{n'} - 2^n$, hence it follows that:

$$D = \Big\langle D' 2^n \Big\rangle_{2^{n'}} = D' 2^n \leq 2^{n'} - 2^n . \tag{20}$$

Therefore, since $0 \leq \langle Y_L^* + Q_L \rangle_{2^n} \leq 2^n - 1$ and due to (20), it follows that:

$$\Big\langle (Y_H^* + Q_H + c) \, 2^n \Big\rangle_{2^{n'}} + \Big\langle Y_L^* + Q_L \Big\rangle_{2^n} \leq 2^{n'} - 1 \tag{21}$$

which implies that (18) can be written as

$$Y = \Big\langle (Y_H^* + Q_H + c) \, 2^n \Big\rangle_{2^{n'}} + \Big\langle Y_L^* + Q_L \Big\rangle_{2^n} . \tag{22}$$

Equation (22) reveals that Y can be evaluated as a simple addition, without the need for an external modulo operation, as in (11). Therefore, the sought residue product C, with $C = \langle Y \rangle_m$, can be expressed as

$$C = \Big\langle \Big\langle (Y_H^* + Q_H + c) \, 2^n \Big\rangle_{2^{n'}} + \Big\langle Y_L^* + Q_L \Big\rangle_{2^n} \Big\rangle_m$$

$$= \Big\langle R_1 + R_2 \Big\rangle_m \tag{23}$$

where

$$R_1 = \Big\langle \Big\langle (Y_H^* + Q_H + c) \, 2^n \Big\rangle_{2^{n'}} \Big\rangle_m \tag{24}$$

$$R_2 = \Big\langle Y_L^* + Q_L \Big\rangle_{2^n} . \tag{25}$$

The value of R_1 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 be designed to receive the $n' - n$ most significant bits of Y^* and the bit c and compute the n-bit value R_1 of (24). Based on the modulo arithmetic property $\langle m - a \rangle_m = \langle -a \rangle_m$, the value of R_1 is selected to be negative and encoded in $(n + 1)$-bit two's complement format. Therefore, since $-m \leq R_1 \leq 0$ and $0 \leq R_2 \leq 2^n - 1$, it is derived that $-m \leq R_1 + R_2 \leq 2^n - 1$, which implies that

$$-m < R_1 + R_2 < 2m . \tag{26}$$

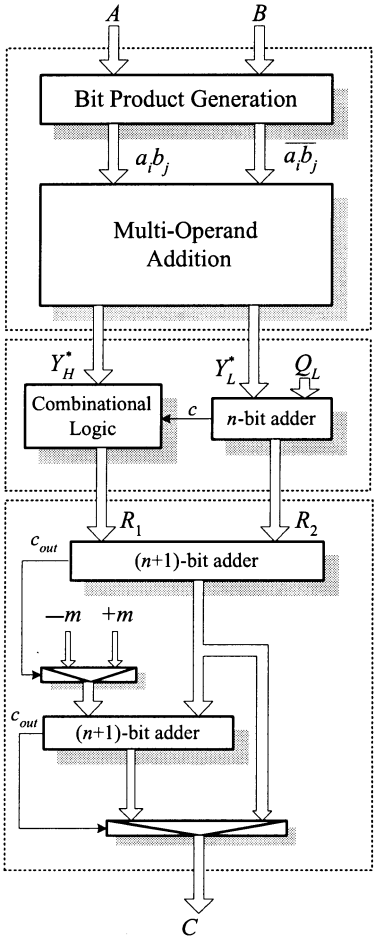


Fig. 1. Three-stage residue multiplier.

Inequalities (26) dictate that three cases can be distinguished in the computation of the residue product C, depending on the value of $R_1 + R_2$

$$C=\left\langle R_1+R_2\right\rangle_m=\begin{cases} R_1+R_2+m, & \text{if } -m \le R_1+R_2 < 0 \\ R_1+R_2, & \text{if } 0 \le R_1+R_2 < m \\ R_1+R_2-m, & \text{if } m \le R_1+R_2 < 2m \end{cases}. \tag{27}$$

Therefore, the computation of the residue product C initially requires the computation of Y^* using (12), followed by the computation of (24) and (25). The final result is then obtained using (27). An implementation of the proposed algorithm is shown in Fig. 1.

III. Properties of Bit Products in a Residue Multiplier

This section introduces a set of theorems that organize the bit products involved in a residue product computation into sets which can be processed by simple gates instead of adders. In addition, in this section, a new property of inverted bit products in a residue multiplier is identified and a corresponding theorem is introduced.

In the following, it is assumed that 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_ib_j or $\overline{a_ib_j}$ that contribute to an output bit y_k exhibit specific relationships, which can be exploited to lead to low-complexity residue multipliers. Definitions 1 and 2 introduce the concept of *compatible* and *incompatible* bit products and the subsequent Theorems 1 and 2 reveal the conditions that should be satisfied in order two bit products to be compatible.

Definition 1: The bit products a_ib_j and a_kb_l are called compatible bit products when $a_ib_j + a_kb_l \le 1$, for all legitimate input combinations of the residues A and $B$with $0 \le i, j, k, l \le n-1$ and $0 \le A, B < m$.

Definition 2: Any N bit products $a_{i_1}b_{j_1}, a_{i_2}b_{j_2}, \ldots, a_{i_N}b_{j_N}$ such that $\sum_{k=1}^{N} a_{i_k}b_{j_k} > 1$, for at least one valid input combination, $0 \le A, B < m$, are called incompatible bit products.

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, directly reducing the cost of addition and preventing the generation of a carry bit that would contribute to more significant output bits. In the following, theorems are introduced that can be used to directly identify compatible bit products without the need to perform an exhaustive simulation as done in [11] and[12].

Theorem 1: The bit products a_ib_j and a_kb_l are compatible if and only if $2^i + 2^k \ge m \bigvee 2^j + 2^l \ge m$ and $i \ne k \vee j \ne l$.

Theorem 2: If a_ib_j and a_kb_l are compatible bit products then the bit products a_jb_i and a_lb_k are also compatible.

In order to clarify the compatibility property assume, for example, the bit products a_3b_2 and a_2b_0, in case of a modulo-11 residue multiplier. Since, $2^3 + 2^2 = 12 > 11$, then according to Theorem 1 , the bit products a_3b_2 and a_2b_0 are compatible. Therefore, for all possible combinations of the input operands $A, B \in \{0, 1, \ldots, 10\}$ it holds that $a_3b_2 + a_2b_0 \le 1$. On the contrary, assume the bit products a_0b_1 and a_2b_2. In this case, it holds that $2^0 + 2^2 = 5 < 11$ and $2^1 + 2^2 = 6 < 11$. Therefore, the bit products a_0b_1 and a_2b_2 are incompatible. The sum of the incompatible bit products a_0b_1 and a_2b_2 can assume all possible values from the set {0, 1, 2} and, thus, they must be added using a 1-bit adder cell. In case that $A = 9 = 1001_2$ and $B = 1 = 0001_2$ it holds that $a_0b_1 + a_2b_2 = 0 + 0 < 1$, but this is not sufficient for the two bit products to be compatible, since in case that $A = 5 = 0101_2$ and $B = 6 = 0110_2$, it holds that $a_0b_1 + a_2b_2 = 1 + 1 = 2 > 1$, which violates the condition of Definition 1.

To exploit the property of compatibility between two bit products, an efficient method for the identification of compatible bit products is introduced. Particularly, for each a_ib_j, bit products of the form $a_{i+w}b_{j-t}$ and $a_{i+w}b_{t-j}$ that are compatible to a_ib_j, are sought by constraining the values assumed by i and j so that the index values $i+w$, $j-t$, and $t-j$ are legitimate. The variables w and t are used to span the complete index space of i and j for each a_ib_j. It is noted that due to Theorem 2 for each bit product of the form $a_{i+w}b_{j-t}$ or $a_{i+w}b_{t-j}$ identified to be compatible to a_ib_j, the bit products $a_{j-t}b_{i+w}$ or $a_{t-j}b_{i+w}$ are also compatible to a_jb_i. For each of the two forms of candidate bit products, the upper and lower bounds of i and j are defined by Propositions 1 and 2.

Proposition 1: The bit products a_ib_j *and* $a_{i+w}b_{j-t}$ are compatible when $\lceil \log_2(m/(2^w+1)) \rceil \le i \le n-1-w$ and $t \le j \le n-1$, with $1 \le w \le n-1$ and $0 \le t \le n-1$.

Proposition 2: The bit products $a_i b_j$ and $a_{i+w} b_{t-j}$ are compatible when $\lceil \log_2 (m/(2^w+1)) \rceil \leq i \leq n-1-w$ and $0 \leq j \leq t$, with $1 \leq w \leq n-1$ and $0 \leq t \leq n-1$.

The identification of all compatible bit-product pairs in case of a modulo-11 residue multiplier, is clarified via the following example.

Example 1: Since $m = 11$, then, the number of bits required to represent the input operands is $n = 4$ bits. First, the compatible bit-product pairs of the form $(a_i b_j, a_{i+w} b_{j-t})$ are investigated. Due to Proposition 1 , the values that the variables w and t can assume are $w \in \{1,2,3\}$ and $t \in \{0,1,2,3\}$. For each combination of w and t the legitimate values of the indexes i, j of the compatible bit-product pairs are obtained as follows.

- $(w = 1, t = 0)$: Due to Proposition 1, the values of i and j are bounded by $2 \leq i \leq 2$ and $0 \leq j \leq 3$. Thus, $i \in \{2\}$ and $j \in \{0,1,2,3\}$, and the resulting compatible bit product pairs are

$$\begin{aligned}(a_2 b_0, a_{2+1} b_{0-0}) &= (a_2 b_0, a_3 b_0)\\ (a_2 b_1, a_{2+1} b_{1-0}) &= (a_2 b_1, a_3 b_1)\\ (a_2 b_2, a_{2+1} b_{2-0}) &= (a_2 b_2, a_3 b_2)\\ (a_2 b_3, a_{2+1} b_{3-0}) &= (a_2 b_3, a_3 b_3).\end{aligned}$$

- $(w = 1, t = 1)$: The values of i and j are bounded by $2 \leq i \leq 2$ and $1 \leq j \leq 3$. Thus $i \in \{2\}$ and $j \in \{1,2,3\}$, and the resulting compatible bit product pairs are

$$\begin{aligned}(a_2 b_1, a_{2+1} b_{1-1}) &= (a_2 b_1, a_3 b_0)\\ (a_2 b_2, a_{2+1} b_{2-1}) &= (a_2 b_2, a_3 b_1)\\ (a_2 b_3, a_{2+1} b_{3-1}) &= (a_2 b_3, a_3 b_2).\end{aligned}$$

- $(w = 1, t = 2)$: The values of i and j are bounded by $2 \leq i \leq 2$ and $2 \leq j \leq 3$. Thus, $i \in \{2\}$ and $j \in \{2,3\}$, and the resulting compatible bit product pairs are

$$\begin{aligned}(a_2 b_2, a_{2+1} b_{2-2}) &= (a_2 b_2, a_3 b_0)\\ (a_2 b_3, a_{2+1} b_{3-2}) &= (a_2 b_3, a_3 b_1).\end{aligned}$$

- $(w = 1, t = 3)$: The values of i and j are bounded by $2 \leq i \leq 2$ and $3 \leq j \leq 3$. Thus $i \in \{2\}$ and $j \in \{3\}$, and the resulting compatible bit product pair is

$$(a_2 b_3, a_{2+1} b_{3-3}) = (a_2 b_3, a_3 b_0).$$

In case that $w = 2$, it follows that $2 \leq i \leq 1$, which is false; hence, it does not allow the existence of a legitimate index i for a compatible bit-product pair irrespectively of the values of t. The same holds for $w = 3$, since according to Proposition 1, it is derived that $1 \leq i \leq 0$. Therefore, the following set of compatible bit-product pairs is identified via Proposition 1

$$\begin{aligned}\mathcal{K}_{\text{Prop-1}} = \{&(a_2 b_0, a_3 b_0), (a_2 b_1, a_3 b_1), (a_2 b_2, a_3 b_2), (a_2 b_3, a_3 b_3)\\ &(a_2 b_1, a_3 b_0), (a_2 b_2, a_3 b_1), (a_2 b_3, a_3 b_2), (a_2 b_2, a_3 b_0)\\ &(a_2 b_3, a_3 b_1), (a_2 b_3, a_3 b_0)\}.\end{aligned}$$

In a similar manner, the following compatible bit-product pairs of the form $(a_i b_j, a_{i+w} b_{t-j})$ are identified by means of Proposition 2

$$\begin{aligned}\mathcal{K}_{\text{Prop-2}} = \{&(a_2 b_0, a_3 b_0), (a_2 b_0, a_3 b_1), (a_2 b_1, a_3 b_0), (a_2 b_0, a_3 b_2)\\ &(a_2 b_1, a_3 b_1), (a_2 b_2, a_3 b_0), (a_2 b_0, a_3 b_3), (a_2 b_1, a_3 b_2)\\ &(a_2 b_2, a_3 b_1), (a_2 b_3, a_3 b_0)\}.\end{aligned}$$

The unique compatible bit-product pairs of both sets $\mathcal{K}_1$ and $\mathcal{K}_2$ are as follows:

$$\begin{aligned}\mathcal{K} = \{&(a_2 b_0, a_3 b_0), (a_2 b_1, a_3 b_1), (a_2 b_2, a_3 b_2), (a_2 b_3, a_3 b_3)\\ &(a_2 b_1, a_3 b_0), (a_2 b_2, a_3 b_1), (a_2 b_3, a_3 b_2), (a_2 b_2, a_3 b_0)\\ &(a_2 b_3, a_3 b_1), (a_2 b_3, a_3 b_0), (a_2 b_0, a_3 b_2), (a_2 b_0, a_3 b_3)\\ &(a_2 b_1, a_3 b_2)\}.\end{aligned}$$

Due to Theorem 2, the following set of bit-product pairs are also compatible:

$$\begin{aligned}\mathcal{K}^* = \{&(a_0 b_2, a_0 b_3), (a_1 b_2, a_1 b_3), (a_2 b_2, a_2 b_3), (a_3 b_2, a_3 b_3)\\ &(a_1 b_2, a_0 b_3), (a_2 b_2, a_1 b_3), (a_3 b_2, a_2 b_3), (a_2 b_2, a_0 b_3)\\ &(a_3 b_2, a_1 b_3), (a_3 b_2, a_0 b_3), (a_0 b_2, a_2 b_3), (a_0 b_2, a_3 b_3)\\ &(a_1 b_2, a_2 b_3)\}.\end{aligned}$$

Thus, the complete set $\mathcal{K}_{11}$ of compatible bit products, in case that $m = 11$, is

$$\begin{aligned}\mathcal{K}_{11} = \{&(a_2 b_0, a_3 b_0), (a_2 b_1, a_3 b_1), (a_2 b_2, a_3 b_2), (a_2 b_3, a_3 b_3)\\ &(a_2 b_1, a_3 b_0), (a_2 b_2, a_3 b_1), (a_2 b_3, a_3 b_2), (a_2 b_2, a_3 b_0)\\ &(a_2 b_3, a_3 b_1), (a_2 b_3, a_3 b_0), (a_2 b_0, a_3 b_2), (a_2 b_0, a_3 b_3)\\ &(a_2 b_1, a_3 b_2), (a_0 b_2, a_0 b_3), (a_1 b_2, a_1 b_3), (a_2 b_2, a_2 b_3)\\ &(a_3 b_2, a_3 b_3), (a_1 b_2, a_0 b_3), (a_2 b_2, a_1 b_3), (a_2 b_2, a_0 b_3)\\ &(a_3 b_2, a_1 b_3), (a_3 b_2, a_0 b_3), (a_0 b_2, a_2 b_3), (a_0 b_2, a_3 b_3)\\ &(a_1 b_2, a_2 b_3)\}. \qquad (28)\end{aligned}$$

□

The identification of compatible bit products is depended on the value of the modulo m, according to Theorems 1 and 2. The existence of compatible bit products is found to be limited to certain values of m. In particular, the values of n-bit moduli that allow the existence of compatible bit products are identified by Proposition 3.

Proposition 3: At least one compatible bit-product pair exists, for every legitimate combination of input residues if and only if n-bit modulo m is in the range $2^{n-1} \leq m \leq 2^{n-1} + 2^{n-2}$.

Consider for example, all possible 4-bit moduli $\{8, 9, 10, \ldots, 15\}$. When m belongs to the set {8, 9, 10, 11, 12}, the identification of compatible bit products is possible. However, according to Proposition 3, in case that $m \in \{13, 14, 15\}$ no compatible bit products exist.

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.

Definition 3: An N-tuple of bit products $a_{i_1} b_{j_1}, a_{i_2} b_{j_2}, \ldots, a_{i_N} b_{j_N}$ is called a compatible N-tuple, when $\sum_{k=1}^{N} a_{i_k} b_{j_k} \leq 1$,

for all legitimate input combinations of the residues A and B with $0 \leq i_k, j_k \leq n-1$.

A compatible N-tuple of bit products 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 directly identifies compatible N-tuples, by exploiting the existence of particular compatible bit-product pairs, found by means of Propositions 1 and 2.

Theorem 3: Let $x_1, x_2, \ldots, x_N$ such that $x_i \in \{0,1\}$ for each $1 \leq i \leq N$. Then, $x_1 + x_2 + \cdots + x_N \leq 1$ 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 $x_i + x_j \leq 1$.

Example 2: Assume that we need to check if the bit products a_2b_2, a_3b_0, a_2b_3 constitute a compatible triplet, i.e., $a_2b_2 + a_3b_0 + a_2b_3 \leq 1$, in case of a modulo-11 residue multiplier. Instead of trying to identify separate conditions or to rely on exhaustive simulations, the existence of compatible bit-product pairs will be exploited. From the set $\mathcal{K}_{11}$ of (28) in Example 1, it can be verified that the three possible bit-product pairs (a_2b_2, a_3b_0), (a_2b_2, a_2b_3), (a_3b_0, a_2b_3) are compatible, that is

$$\begin{aligned} a_2b_2 + a_3b_0 &\leq 1 \\ a_2b_2 + a_2b_3 &\leq 1 \\ a_3b_0 + a_2b_3 &\leq 1\,. \end{aligned}$$

Therefore, according to Theorem 3 it follows that $a_2b_2 + a_3b_0 + a_2b_3 \leq 1$. □

Definition 4: A N-tuple of inverted bit products $\overline{a_{i_1}b_{j_1}}, \overline{a_{i_2}b_{j_2}}, \ldots, \overline{a_{i_N}b_{j_N}}$ is called a constraint N-tuple when $N - 1 \leq \sum_{k=1}^{N} \overline{a_{i_k}b_{j_k}} \leq N$, for all legitimate input combinations of the residues A and B with $0 \leq i_k, j_k \leq n-1$.

The proposed residue multiplication algorithm requires the computation of Y^* using (12), which includes the summation of inverted bit products $\overline{a_ib_j}$. Therefore, properties of inverted bit products are investigated, with the objective to minimize the cost of their addition. Definition 4 describes a property of inverted bit-product N-tuples, which resembles the compatibility property of Definition 3, in the sense that the sum of particular inverted bit products can only assume a limited set of values. Furthermore, the introduced Theorem 4 transforms the problem of identifying an inverted bit-product N-tuple, the sum of which assumes a limited set of values, i.e., a constraint N-tuple, to the problem of identifying a corresponding compatible bit-product N-tuple.

Theorem 4: Let $x_1, x_2, \ldots, x_N$ such that $x_i \in \{0,1\}$ for each $1 \leq i \leq N$. If $x_1 + x_2 + \cdots + x_N \leq 1$ then $N - 1 \leq \bar{x}_1 + \bar{x}_2 + \cdots + \bar{x}_N \leq N$.

Example 3 describes the identification of constraint N-tuples by means of Theorems 3 and 4.

Example 3: Consider the following three inverted bits products $\overline{a_2b_2}$, $\overline{a_1b_3}$, and $\overline{a_3b_3}$, in case of a modulo-11 residue multiplier. In order to examine whether the three inverted bit products constitute a constraint triplet the corresponding noninverted bit products a_2b_2, a_1b_3, and a_3b_3 are assumed. Since all possible bit-product pairs (a_2b_2, a_1b_3), (a_2b_2, a_3b_3), (a_1b_3, a_3b_3) are identified to be compatible, according to (28) of Example 1, then, due to Theorem 3, it follows that $a_2b_2 + a_1b_3 + a_3b_3 \leq 1$. Following Theorem 4, it is derived that $2 \leq \overline{a_2b_2} + \overline{a_1b_3} + \overline{a_3b_3} \leq 3$, and hence, $\overline{a_2b_2}$, $\overline{a_1b_3}$, and $\overline{a_3b_3}$ form a constraint triplet. □

An efficient methodology for the identification of compatible and constraint N-tuples is presented in Section V.

IV. Organization of Proposed Residue Multiplier

This section describes a hardware architecture that implements the algorithm presented in Section II. The proposed residue multiplier is organized in three stages as shown in Fig. 1. The first stage computes Y^* defined by (12), using cascaded columns of 1-bit adders, organized in a way such 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 R_1 and R_2, using (24) and (25), while the third stage performs the final residue mapping via the conditional correction described by (27).

In particular, to design the kth output column of the first stage, $0 \leq k \leq n_c - 1$, 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

$$\mathcal{S}_k = \{a_ib_j \mid \hat{z}_{k,i,j} = 1\} \tag{29}$$

$$\mathcal{N}_k = \{\overline{a_ib_j} \mid \hat{z}_{k,i,j} = \bar{1}\} \tag{30}$$

where $i, j \in \{0, 1, \ldots, n-1\}$. The cardinality of $\mathcal{S}_k$ and $\mathcal{N}_k$, denoted as $|\mathcal{S}_k|$ and $|\mathcal{N}_k|$ respectively, directly depends on the number of nonzero elements, i.e., $\{\bar{1}, 1\}$, that appear in the encoding of the bit-product weights $\langle 2^{i+j} \rangle_m$ as defined in (4). The total number of bit products that participate in the kth output column equals to $|\mathcal{S}_k| + |\mathcal{N}_k|$. In Section V, an optimization methodology is introduced, according to which the encoding of $\langle 2^{i+j} \rangle_m$ is selected so that the number of bit products to be added on the kth output column is minimized. Additionally, exploiting the data-dependent properties of the bit products of the sets $\mathcal{S}_k$ and $\mathcal{N}_k$, as described in Section III, allows the use of simple gates to implement the multibit addition on each output column. The organization of the multioperand addition of the bits per output column is independent of the proposed architecture and can be derived using existing methodologies, including carry–save adder arrays [19], Wallace [20], and Dadda trees [21]. In the following, the discussion is limited to the carry–save array organization that imposes the carries produced by a certain output column to be processed by the subsequent more significant column. In particular, on the kth output 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 output column according to the following design rules.

- The N-bit products $a_{i_1}b_{j_1}, a_{i_2}b_{j_2}, \ldots, a_{i_N}b_{j_N}$ identified to be compatible are added with an N-input OR gate.
- Every triplet of inverted bit products that their sum assumes the values two or three, i.e., $2 \leq \overline{a_{i_1}b_{j_1}} + \overline{a_{i_2}b_{j_2}} + \overline{a_{i_3}b_{j_3}} \leq 3$ are added using a three-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 output 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

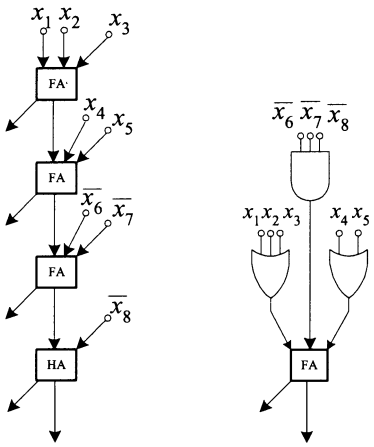


Fig. 2. Hardware reduction of the kth output column. Following the proposed optimization, the cumulative constant Q has to be augmented with 2^{k+1}.

using the property of Theorem 4, the proposed architecture uses the maximal number of three-input AND gates to add them in order to limit the corresponding carry propagation to the next more significant column only.

- The output of the OR gates and the AND gates along with the remaining bit products and the carry bits are added using 1-bit 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.

An example column organization is shown in Fig. 2, where for the contributing bits x_i holds that $x_1+x_2+x_3 \leq 1$, $x_4+x_5 \leq 1$, and $2 \leq \overline{x}_6+\overline{x}_7+\overline{x}_8 \leq 3$. The exploitation of the data-dependent properties of the bits x_i has eliminated two FA and one HA cells needed to add the eight bits in a conventional organization.

In the second stage of computation a n-bit ripple-carry adder adds the n least significant bits of the cumulative constant Q to the n least significant bits of Y^*. Since the bits of Q are predetermined due to (10), the addition is performed using simplified FA cells and HA cells [22, p. 83]. The carry bit produced from the $(n-1)$st bit position is processed along with the $n'-n$ most significant bits of Y^* by a simple combinational-logic network, which computes the value of R_1 as defined by (24). The third stage of the proposed multiplication architecture computes the final residue product C. It consists of two $(n+1)$-bit ripple-carry adders and two n-bit multiplexers that perform the final corrective step described by (27).

V. Proposed Optimization Methodology

This section introduces a novel graph-based optimization methodology, which aims to the overall reduction of the hardware complexity of the residue multiplier. Since the complexity of the second and the third stage of the proposed residue multiplier's organization is constant, the reduction of the complexity of the first stage is sought, which is directly dependent to the cardinality of the sets $\mathcal{S}_k$ and $\mathcal{N}_k$, $0 \leq k \leq n_c-1$. The proposed optimization methodology is twofold. At first, the biased signed-digit encoding (BSDE) is applied on the weights $\langle 2^{i+j} \rangle_m$ in order to minimize the number of bit products of both forms a_ib_j and $\overline{a_ib_j}$ that are distributed on the kth output column. Subsequently large compatible tuples are efficiently identified via an introduced graph-theoretic approach, called BPCA.

A. Biased Signed Digit Encoding

Assume that δ is such that $\langle 2^{i+j} \rangle_m = m-\delta$, where $0 < \delta \leq m$, for each bit product a_ib_j. Then it follows that:

$$\langle m-\delta \rangle_m = \langle -\delta \rangle_m = \langle 2^{i+j} \rangle_m \tag{31}$$

which reveals that in the summation $\sum_i \sum_j a_ib_j \langle 2^{i+j} \rangle_m$ either negative terms $a_ib_j\,(-\delta)$ or positive terms $a_ib_j\,(m-\delta)$ can be added without affecting the sought residue product value. Therefore, separate encodings can be assumed for each case, respectively. In particular, for each bit product a_ib_j, according to (31) and (4), four cases are distinguished.

- The weight $\langle 2^{i+j} \rangle_m$ is written as $\langle 2^{i+j} \rangle_m = m-\delta$ and it is encoded using positive binary digits, i.e., $\widehat{z}_{k,i,j} \in \{0,1\}$. In this case, the number of bits required for the encoding is $n_c = n$.
- The weight $\langle 2^{i+j} \rangle_m$ is written as $\langle 2^{i+j} \rangle_m = -\delta$ and it is encoded using negative binary digits only, i.e., $\widehat{z}_{k,i,j} \in \{0,\overline{1}\}$. Equivalently, the number of bits required for the encoding is $n_c = n$.
- The weight $\langle 2^{i+j} \rangle_m$ is written as $\langle 2^{i+j} \rangle_m = m-\delta$ and it is encoded using the CSD representation. In this case, the terms $\widehat{z}_{k,i,j}$ can assume every value of the digit set $\{\overline{1},0,1\}$ and the encoding requires up to $n+1$ bits to represent the otherwise n-bit value $\langle 2^{i+j} \rangle_m$.
- The weight $\langle 2^{i+j} \rangle_m$ is written as $\langle 2^{i+j} \rangle_m = -\delta$ and it is encoded using the CSD representation. At first the value $-\delta$ is expressed in n-bit two's complement format, which is subsequently transformed to its equivalent CSD representation.

The CSD representation of the weights $\langle 2^{i+j} \rangle_m$ is of particular interest since it requires the minimal number of nonzero 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, as the one given by Parhi [15, p. 507].

Let $\beta^+_{i,j}$ denote the number of $\widehat{z}_{k,i,j}$, $0 \leq k \leq n_c-1$, such that $\widehat{z}_{k,i,j} = 1$ in the encoding of $\langle 2^{i+j} \rangle_m$ and $\beta^-_{i,j}$ be the number of $\widehat{z}_{k,i,j}$, $0 \leq k \leq n_c-1$, $\widehat{z}_{k,i,j} = \overline{1}$, respectively. Therefore, for each weight, the encoding that minimizes the sum $\beta^+_{i,j}+\beta^-_{i,j}$ is sought, $\beta^+_{i,j}+\beta^-_{i,j} = \mathcal{B}_{\min}$, because in this case the number of bit products to be processed by the proposed residue multiplier is minimized. In case that several encodings achieve an identical minimal sum $\mathcal{B}_{\min}$, the encoding with maximum number of positive nonzero digits, i.e., $\beta^+_{i,j} > \beta^-_{i,j}$, is preferred, as it facilitates the use of OR gates. OR-gate based optimization is preferable because it is more effective as it requires the existence of at least a compatible bit-product pair, while the AND-gate optimization requires the existence of an inverted bit-product triplet, with the property $2 \leq \overline{a_ib_j}+\overline{a_kb_l}+\overline{a_pb_q} \leq 3$.

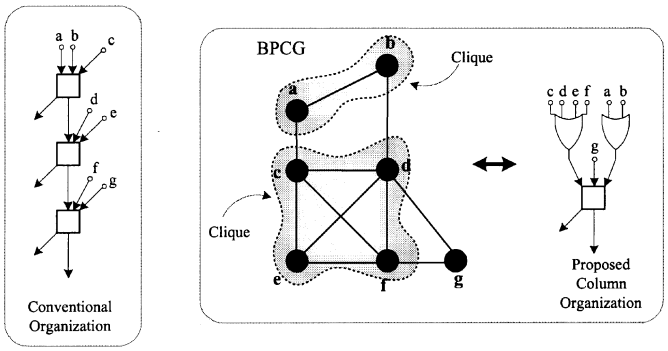


Fig. 3. Example BPCG and the resulting column organization due to the clique partitioning.

B. Bit-Products Compatibility Analysis

The proposed BPCA methodology organizes the bit products of the kth column of Y^* into disjoint sets, the sum of the elements of which assumes a limited set of values. Hence, the corresponding multibit addition can be performed by simple gates instead of 1-bit adder cells. In particular, the compatible bit-product pairs, identified by means of Propositions 1 and 2 that participate on the kth output column, are organized into triplets, quadruples, or N-tuples in general, in a way, such that the number of terms added to the kth bit of Y^* is minimized.

The identification of the maximal N-tuples that exist in each $\mathcal{S}_k$ is achieved by a graph-theoretic formulation. Specifically, for the kth output column an undirected graph $G_k(V_k, E_k)$ is constructed, called the BPCG. The set of bit products that participate in $\mathcal{S}_k$ 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 a graph partitioning problem. Specifically, the BPCG is partitioned in a way such that each partition forms a complete subgraph (*clique*) and the number of disjoint partitions is minimal [23]. The identification of compatible bit-product N-tuples is equivalent to the clique partitioning problem on the BPCG. Assume that a complete subgraph $G_s(V_s, E_s)$ with $V_s \subseteq V_k$ and $E_s \subseteq E_k$ is identified on the BPCG G_k. Then, 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 that their equivalent vertices appear in G_s. Hence, for all possible bit-product pairs $(a_{i_k} b_{j_k},\ a_{i_l} b_{j_l})$ with $1 \leq k,\ l \leq |V_s|$ and $k \neq l$, that have their representative vertices u_k and u_l in the clique G_s, it holds that $a_{i_k} b_{j_k} + a_{i_l} b_{j_l} \leq 1$. So, from Theorem 3 it follows that $\sum_{r=1}^{|V_s|} a_{i_r} b_{j_r} \leq 1$. Thus, the $|V_s|$ bit-product terms $a_{i_r} b_{j_r} \in \mathcal{S}_k$, form a compatible $|V_s|$-tuple. Finally, since the graph partitioning procedure seeks the minimum number of cliques in G_k then the number of the formed N-tuples is minimal too. Fig. 3 shows the relationship between the compatible bit products identified via clique partitioning on the BPCG and the resulting output column organization.

In general, clique partitioning of a graph G_k is a NP-complete problem. However, based on the assumptions presented by De Michelli [23, pp. 64–67], an exact solution is reached in polynomial time by finding the minimum number of colors required to color the complement graph of G_k, denoted as $\overline{G}_k$. In this case, the vertices of $\overline{G}_k$ that are assigned the same color, form a complete subgraph in G_k, and thus correspond to bit products that form an N-tuple. Therefore, the maximum number of independent terms of $\mathcal{S}_k$ that are finally added in the kth output column, is equal to the total number of cliques in the BPCG G_k, or, equivalently, is equal to the total number of distinct colors needed to color the vertices of $\overline{G}_k$.

To identify constraint N-tuples of inverted bit products $\{\overline{a_{i_1} b_{j_1}}, \overline{a_{i_2} b_{j_2}}, \ldots, \overline{a_{i_N} b_{j_N}}\} \in \mathcal{N}_k$ the sum of which assumes a limited set of values, the following procedure is proposed. The BPCG G'_k that corresponds to the bit products $a_{i_r} b_{j_r}$, such that $\overline{a_{i_r} b_{j_r}} \in \mathcal{N}_k$ and $1 \leq r \leq N$, is initially constructed. The solution of clique partitioning on G'_k, derives N-tuples with the property $\sum_{r=0}^{N-1} a_{i_r} b_{j_r} \leq 1$, with $\overline{a_{i_r} b_{j_r}} \in \mathcal{N}_k$. Therefore, due to Theorem 4, it follows that $N - 1 \leq \sum_{r=0}^{N-1} \overline{a_{i_r} b_{j_r}} \leq N$, which implies that any three inverted bit products of the constraint N-tuples can be processed by an AND gate.

TABLE I
(a) POWERS OF TWO AND THE CSD ENCODING OF THE POSITIVE VALUE OF THE WEIGHTS $\langle 2^{i+j} \rangle_{11}$ AND THE SIGNED POWERS OF TWO AND THE CSD ENCODING OF THE NEGATIVE VALUE OF THE WEIGHTS $\langle 2^{i+j} \rangle_{11}$. (b) FINAL SELECTED ENCODING FOR EACH BIT-PRODUCT WEIGHT $\langle 2^{i+j} \rangle_{11}$, $0 \leq i + j \leq 6$

$i+j$	$\langle 2^{i+j} \rangle_{11}$	Positive Encodings		$\langle 2^{i+j} \rangle_{11}$	Negative Encodings	
	$11 - \delta$	Powers-of-Two	CSD	$-\delta$	Signed Powers-of-Two	CSD
0	1	0001	0001	−10	$\bar{1}0\bar{1}0$	$\bar{1}0\bar{1}0$
1	2	0010	0010	−9	$\bar{1}00\bar{1}$	$\bar{1}00\bar{1}$
2	4	0100	0100	−7	$0\bar{1}\bar{1}\bar{1}$	$\bar{1}001$
3	8	1000	1000	−3	$00\bar{1}\bar{1}$	$0\bar{1}01$
4	5	0101	0101	−6	$0\bar{1}\bar{1}0$	$\bar{1}010$
5	10	1010	1010	−1	$000\bar{1}$	$000\bar{1}$
6	9	1001	1001	−2	$00\bar{1}0$	$00\bar{1}0$

(a)

$i+j$	$\langle 2^{i+j} \rangle_{11}$	Selected Encoding	Value
0	$\langle 2^0 \rangle_{11}$	0001	1
1	$\langle 2^1 \rangle_{11}$	0010	2
2	$\langle 2^2 \rangle_{11}$	0100	4
3	$\langle 2^3 \rangle_{11}$	1000	8
4	$\langle 2^4 \rangle_{11}$	0101	5
5	$\langle 2^5 \rangle_{11}$	$000\bar{1}$	−1
6	$\langle 2^6 \rangle_{11}$	$00\bar{1}0$	−2

(b)

C. The Proposed Modulo-11 Residue Multiplier

The proposed multiplier architecture along with the application of the introduced optimization techniques are clarified via the design of a modulo-11 residue multiplier. Since $m = 11$, then $n = 4$ bits are required to represent the input operands. The sum $i + j$ of the indexes i, j of a bit product $a_i b_j$ can assume values that range from 0 to $2(n - 1) = 6$.

Application of BSDE: Table I(a) presents the candidate encodings of $\langle 2^{i+j} \rangle_{11} = 11 - \delta$ and $\langle 2^{i+j} \rangle_{11} = -\delta$ for all possible values of the sum $i+j$ of the indexes i and j of a bit product $a_i b_j$.

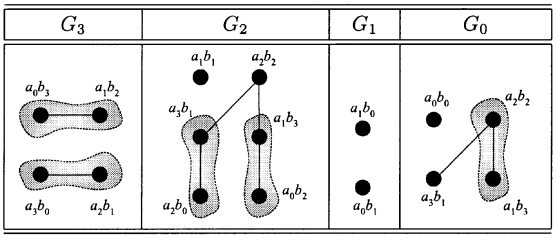
TABLE II
DISTRIBUTION OF THE BIT PRODUCTS TO THE SETS $\mathcal{S}_k$ AND $\mathcal{N}_k$

2^3		2^2		2^1		2^0	
$\mathcal{S}_3$	$\mathcal{N}_3$	$\mathcal{S}_2$	$\mathcal{N}_2$	$\mathcal{S}_1$	$\mathcal{N}_1$	$\mathcal{S}_0$	$\mathcal{N}_0$
a_0b_3	$\emptyset$	a_0b_2	$\emptyset$	a_0b_1	$\overline{a_3b_3}$	a_0b_0	$\overline{a_3b_2}$
a_1b_2		a_1b_1		a_1b_0		a_1b_3	$\overline{a_2b_3}$
a_2b_1		a_2b_0				a_2b_2	
a_3b_0		a_1b_3				a_3b_1	
		a_2b_2					
		a_3b_1					

TABLE III
(a) COMPATIBLE BIT-PRODUCT PAIRS IDENTIFIED ON THE kTH OUTPUT COLUMN AND (b) CORRESPONDING BIT-PRODUCT COMPATIBILITY GRAPHS G_k OF THE PROPOSED MODULO-11 RESIDUE MULTIPLIER

2^3		2^2		2^1		2^0	
$\mathcal{S}_3$	$\mathcal{N}_3$	$\mathcal{S}_2$	$\mathcal{N}_2$	$\mathcal{S}_1$	$\mathcal{N}_1$	$\mathcal{S}_0$	$\mathcal{N}_0$
(a_0b_3, a_1b_2)	$\emptyset$	(a_3b_1, a_2b_0)	$\emptyset$	$\emptyset$	$\emptyset$	(a_2b_2, a_1b_3)	(a_3b_2, a_2b_3)
(a_3b_0, a_2b_1)		(a_1b_3, a_0b_2)				(a_2b_2, a_3b_1)	
		(a_1b_3, a_2b_2)					
		(a_3b_1, a_2b_2)					

(a)



(b)

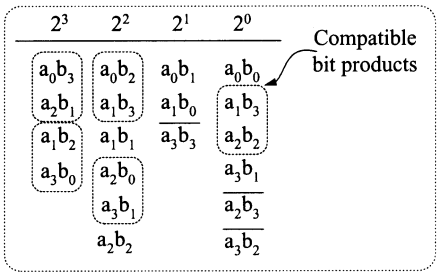
Table I(b) presents the final selected encoding scheme for each $\langle 2^{i+j} \rangle_{11}$. In each case the selected encoding is the one that contains the minimum number of nonzero digits. In case that two or more encodings contain the same number of nonzero elements, the one with the largest number of positive digits is selected.

Distribution of Bit Products: In the following, the sets $\mathcal{S}_k$ and $\mathcal{N}_k$ are derived for each output column k, with $0 \leq k \leq 3$. The bit products that participate on each $\mathcal{S}_k$ and $\mathcal{N}_k$ are shown in Table II.

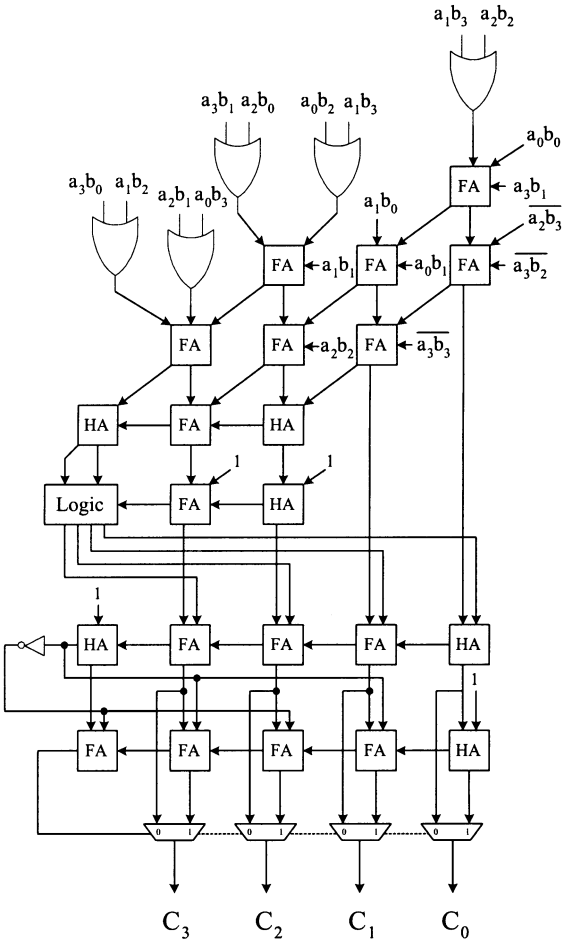
Application of BPCA: As obtained in (28) of Example 1, the set $\mathcal{K}_{11}$ of all compatible bit-product pairs that appear in a modulo-11 residue multiplier, is

$$\begin{aligned}\mathcal{K}_{11} = \{&(a_2b_0, a_3b_0), (a_2b_1, a_3b_1), (a_2b_2, a_3b_2), (a_2b_3, a_3b_3)\\ &(a_2b_1, a_3b_0), (a_2b_2, a_3b_1), (a_2b_3, a_3b_2), (a_2b_2, a_3b_0)\\ &(a_2b_3, a_3b_1), (a_2b_3, a_3b_0), (a_2b_0, a_3b_2), (a_2b_0, a_3b_3)\\ &(a_2b_1, a_3b_2), (a_0b_2, a_0b_3), (a_1b_2, a_1b_3), (a_2b_2, a_2b_3)\\ &(a_3b_2, a_3b_3), (a_1b_2, a_0b_3), (a_2b_2, a_1b_3), (a_2b_2, a_0b_3)\\ &(a_3b_2, a_1b_3), (a_3b_2, a_0b_3), (a_0b_2, a_2b_3), (a_0b_2, a_3b_3)\\ &(a_1b_2, a_2b_3)\}.\end{aligned}$$

Therefore, based on the set $\mathcal{K}_{11}$, the compatible bit-product pairs identified among the bit products that participate on the kth output column, are tabulated in Table III(a). For the bit products



(a)



(b)

Fig. 4. (a) Final organization of the output columns of the proposed modulo-11 residue multiplier. (b) Resulting design.

$\{\overline{a_3b_2}, \overline{a_2b_3}\} \in \mathcal{N}_0$ and $\{\overline{a_3b_3}\} \in \mathcal{N}_1$ their noninverted equivalents are assumed, see Example 3.

The compatible bit-product pairs of Table III(a) allow the construction of the corresponding BPCGs G_k, $0 \leq k \leq 3$. It is noted that for the sets $\mathcal{N}_2$ and $\mathcal{N}_3$ and the sets $\mathcal{N}_0$ and $\mathcal{N}_1$ no graph is constructed, since $\mathcal{N}_2 = \mathcal{N}_3 = \emptyset$, while the number of the bit products $\{\overline{a_2b_3}, \overline{a_3b_2}\} \in \mathcal{N}_0$ and $\{\overline{a_3b_3}\} \in \mathcal{N}_1$ are less than three on each column and thus the AND-based optimization cannot be applied. The corresponding BPCGs G_k, are shown in Table III(b) and the cliques identified on each G_k are depicted as shaded regions. The final organization of the output columns of the proposed modulo-11 residue multiplier, along with the derived sets of compatible bit products is depicted in Fig. 4(a).

TABLE IV
VALUES OF R_1 ACCORDING TO (24) FOR THE PROPOSED MODULO-11 RESIDUE MULTIPLIER

	$R_1 = \left\langle \left\langle (Y_H^* + Q_H + c)\, 2^n \right\rangle_{2^{n'}} \right\rangle_m$	
$Y_5^* Y_4^* c$	Value	Binary
0 0 0	−7	**1** 1 0 0 1
0 0 1	−11	**1** 0 1 0 1
0 1 0	−11	**1** 0 1 0 1
0 1 1	−6	**1** 1 0 0 1
1 0 0	−6	**1** 1 0 0 1
1 0 1	−1	**1** 1 1 1 0
1 1 0	−1	**1** 1 1 1 0
1 1 1	−7	**1** 1 0 0 1

Final Design Organization: Following the design rules presented in Section IV for the kth output column, $0 \leq k \leq 3$, it follows that the bit products that belong to a clique in G_k are added by an OR gate, while the carry bits along with the remaining bit products are added using FA and HA cells, as shown in Fig. 4(b). At this stage, the computation of Y^* is completed and the resulting output word length is $n' = 6$ bits.

Using (10) the cumulative constant constant Q can be computed. The value of Q stems from the distribution of the inverted bit products $\{\overline{a_2 b_3}, \overline{a_3 b_2}\}$ and $\overline{a_3 b_3}$ to the columns of weight 2^0 and 2^1, respectively. Therefore

$$Q = 2 \cdot (2^6 - 2^0) + (2^6 - 2^1) = 188. \tag{32}$$

According to (11), only the n' less significant bits of constant Q are required. Therefore the final value of Q equals to 60, since

$$\langle Q \rangle_{2^6} = \langle 188 \rangle_{64} = 60 \rightarrow 111\,100_2. \tag{33}$$

From (33), it is derived that $Q_L = \langle Q \rangle_{2^n} = \langle 60 \rangle_{16} = 12 \rightarrow 1100_2$, and $Q_H = 3$. The contents of the small lookup table (LUT) that implements (24) are shown in Table IV. It is noted that the LUT returns the negative values of (24) and it is implemented with a simple combinational logic circuit.

The addition of Q to Y^* and the final residue mapping stage according to (27) are shown in Fig. 4(b). Further logic-level optimization of the architecture in Fig. 4(b) is possible. In particular, the FAs with a constant input '1' can be reduced to modified HAs, while the implementation of a HA with a constant input is trivial [22, p. 83]. Such optimizations are not shown, since they are out of the scope of the example.

VI. HARDWARE COMPLEXITY

The first stage of the proposed residue multiplier computes the value of Y^*, according to (12). Each bit product $a_i b_j$ is distributed to the kth output column, either in its normal or complemented form, depending on the selected encoding of the weight $\langle 2^{i+j} \rangle_m$. Then, the sets $\mathcal{S}_k$ and $\mathcal{N}_k$ are constructed for the kth column and the bit products $a_i b_j \in \mathcal{S}_k$ are processed in order to identify disjoint sets of compatible bit products via the clique partitioning on the BPCG G_k, as described in Section V.

A. Area Complexity

The bit products that belong to the ith clique in G_k, with size $N_{i,k}$ greater than one, are added using a $N_{i,k}$-input OR gate. Let λ_k denote the number of disjoint cliques identified in the BPCG G_k, with size $N_{i,k} > 1$. Similarly, assume that λ_k^* constraint $N_{i,k}^*$-tuples of inverted bit products are found in the set $\mathcal{N}_k$. The particular $N_{i,k}^*$-tuples, $i = 0, 1, \ldots, \lambda_k^* - 1$ are processed by AND gates. In order to prevent generation of carries with weight larger than 2^{k+1}, the inverted bit products of the ith constraint $N_{i,k}^*$-tuple are partitioned into disjoint inverted bit-product triplets. Therefore, each of the λ_k^*-tuples requires $\left\lfloor N_{i,k}^*/3 \right\rfloor$ three-input AND gates. It is noted that each of the λ_k^* constraint $N_{i,k}$-tuples augments the cumulative constant Q by a value of $\left\lfloor N_{i,k}^*/3 \right\rfloor 2^{k+1}$.

The number of bits added at the kth column is

$$b_k = |\mathcal{S}_k| + |\mathcal{N}_k| - \sum_{i=0}^{\lambda_k - 1} N_{i,k} + \lambda_k - 3 \sum_{i=0}^{\lambda_k^* - 1} \left\lfloor \frac{N_{i,k}^*}{3} \right\rfloor + \lambda_k^*. \tag{34}$$

Therefore, the numbers of FA and HA cells required to form the kth column, are given by

$$\mathcal{W}_k^{\mathrm{FA}} = \left\lfloor \frac{b_k + r_k - 1}{2} \right\rfloor \tag{35}$$

$$\mathcal{W}_k^{\mathrm{HA}} = \left\langle b_k + r_k - 1 \right\rangle_2 \tag{36}$$

where r_k denotes the number of carries produced at the $(k-1)$st column. Since each FA and HA cell produces a carry bit, the number of carries, r_{k+1}, to the next more significant digital position, i.e., the $(k+1)$st column, is computed as

$$r_{k+1} = \mathcal{W}_k^{\mathrm{FA}} + \mathcal{W}_k^{\mathrm{HA}} \tag{37}$$

with $r_0 = 0$. Additionally, the number of OR gates, with $N_{i,k}$ inputs each, placed at the kth column is equal to λ_k, where $0 \leq k \leq n_c - 1$, while the number of three-input AND gates is $\left\lfloor N_{i,k}^*/3 \right\rfloor$ per ith constraint $N_{i,k}^*$-tuple. Hence, the total area complexity of the first stage is obtained through

$$C_{\mathrm{area,first}} = n^2 a_{\mathrm{AND}} + \sum_{k=0}^{n'-1} \left(\mathcal{W}_k^{\mathrm{FA}} a_{\mathrm{FA}} + \mathcal{W}_k^{\mathrm{HA}} a_{\mathrm{HA}} \right) + \sum_{k=0}^{n_c - 1} \left(\sum_{i=0}^{\lambda_k - 1} N_{i,k} a_{\mathrm{OR}_{N_{i,k}}} + \sum_{i=0}^{\lambda_k^* - 1} \left\lfloor \frac{N_{i,k}^*}{3} \right\rfloor a_{\mathrm{AND}_3} \right) \tag{38}$$

where n' is the word length of the largest number obtained at the output of Y^* and a_{FA}, a_{HA}, a_{AND}, a_{AND_3}, and $a_{\mathrm{OR}_{N_{i,k}}}$ denote the area complexity of a FA, a HA, a 2-input AND gate, a three-input AND gate, and an $N_{i,k}$-input OR gate, respectively. To quantify the area complexity of the architectures, it is assumed according to [24] that $a_{\mathrm{FA}} = 7$, $a_{\mathrm{HA}} = 3$, $a_{\mathrm{AND}} = 1.5$, $a_{\mathrm{AND}_3} = 2$, and, $a_{\mathrm{OR}_{N_{i,k}}} = 0.5(N_{i,k} + 1)$ gate equivalents.

The second and the third stage of the proposed residue multiplier maps Y^* to the final residue product. The n-bit adder, which adds the n-bit least significant part of Y^* with the lower constant n bits of Q has a complexity lower than $n-1$ HA cells and an inverter, as one of the operands is a constant [22, p. 83].

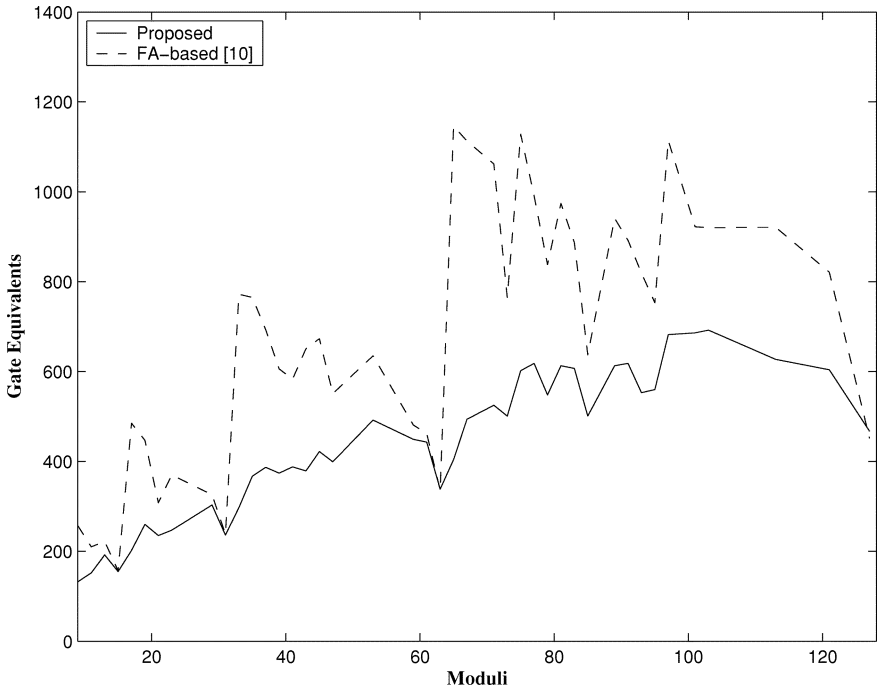


Fig. 5. Area performance of the proposed and the FA-based [10] residue multipliers.

The remaining of the second stage comprises a very low-complexity LUT. Therefore, the area complexity of the second stage is roughly equal to

$$C_{\text{area, second}} = (n-1)a_{\text{HA}}. \tag{39}$$

The third stage consists of a n-bit two-operand adder with a complexity of $n-1$ FAs and one HA, a $(n+1)$-bit adder with a complexity of n FAs and one HA, and an n-bit two-to-one multiplexer, as shown in Fig. 1. Hence, area complexity of the third stage of the proposed multiplier is

$$C_{\text{area, third}} = (2n-1)a_{\text{FA}} + 2a_{\text{HA}} + na_{\text{MUX}} \tag{40}$$

where a_{MUX} denotes the area complexity of an one-bit two-to-one multiplexer and it is equal to 1.5 gate equivalents. The overall area complexity equals to the sum of $C_{\text{area, first}}$, $C_{\text{area, second}}$, and $C_{\text{area, third}}$, that is

$$C_{\text{area}} = C_{\text{area, first}} + C_{\text{area, second}} + C_{\text{area, third}}. \tag{41}$$

B. Time Complexity

The delay through the first stage of the proposed multiplier can be computed by taking into consideration the carry–save organization and the height of each column. The maximum delay T_k of the kth column is

$$T_k = \mathcal{W}_k^{\text{FA}} + \mathcal{W}_k^{\text{HA}} + t_g \tag{42}$$

where t_g denotes the delay of an OR or an AND gate. Due to the carry–save organization of the first stage of the proposed multiplier, the corresponding delay is

$$\begin{aligned} T_{\text{first}} &= \max_{k=0,1,\ldots,n'-1} \{T_k + (n'-1-(k-1))\} \\ &= \max_{k=0,1,\ldots,n'-1} \{T_k + n' - k\} \end{aligned} \tag{43}$$

to reflect that the maximum-length critical path ends at the lowest 1-bit adder of the $(n'-1)$st column, while it commences from a column of height T_k and comprises a carry propagation from the kth to the $(n'-1)$st digital position. The maximum delay through the second stage of the proposed residue multiplier is computed as follows. Due to the carry–save organization of the first stage, the critical path commences from the $(n'-1)$st bit of Y^* to the derivation of the output:

$$\begin{aligned} T_{\text{second}} &= t_{\text{LUT}} + (n-1)t_{\text{FA}} + t_{\text{HA}} + nt_{\text{FA}} + t_{\text{HA}} + t_{\text{MUX}} && (44) \\ &= (2n-1)t_{\text{FA}} + 2t_{\text{HA}} + t_{\text{LUT}} + t_{\text{MUX}} && (45) \end{aligned}$$

where t_{HA}, t_{FA}, t_{LUT}, and t_{MUX} denote the delay of a HA, a FA, a LUT, and a 1-bit multiplexer respectively. It is assumed that $t_{\text{HA}} = 1$, $t_{\text{FA}} = 2$, $t_{\text{LUT}} = 2$, and $t_{\text{MUX}} = 2$ gate delays.

VII. Performance Evaluation

In this section, the performance of the proposed residue multiplier is compared to that of several previously reported architectures in terms of area, time, and area $\times$ time complexities. First, the performance of the proposed multiplier is compared to the performance of a FA-based residue multiplier designed according to the architecture presented by Soudris *et al.* in [10]. Figs. 5–7 present the area, time and area $\times$ 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 since both the number of the bit products that contribute to each output column is minimized due to the BSDE and the organization of each output column is simplified by means of the BPCA methodology. In general, the area $\times$ time savings achieved span 22% to 85%. Additionally, since the proposed architecture eliminates the need of the recursive modulo reduction stages used in [10] the overall delay is significantly reduced. Since the design of the proposed residue

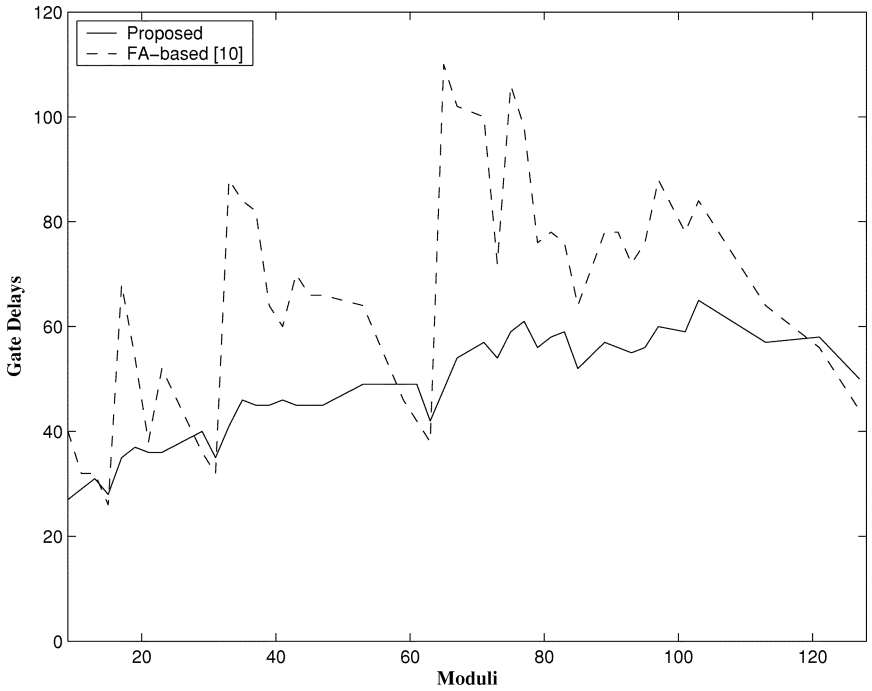


Fig. 6. Time performance of the proposed and the FA-based [10] residue multipliers.

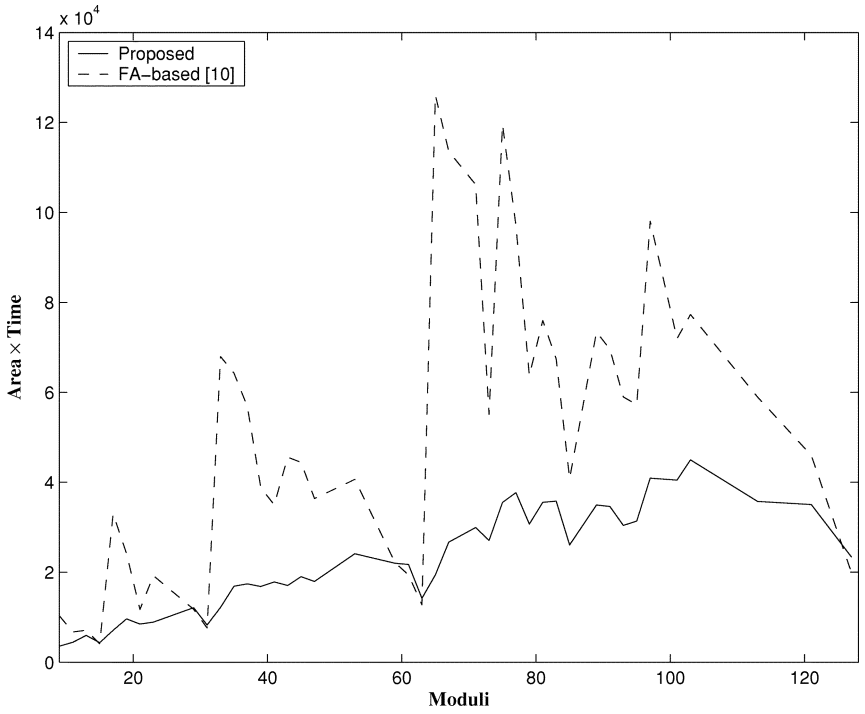


Fig. 7. Area × time performance of the proposed and the FA-based [10] residue multipliers.

multipliers and the architecture presented in [10] is modulo dependent, then in case that $m \in \{2^n - 1, 2^n\}$ both designs exhibit almost equal performance. It is noted that the area × time savings is computed as $(AT - AT_{\text{prop}})/AT \times 100\%$, where AT denotes the area × time complexity of the previously presented architectures, while AT_{prop} denotes the corresponding complexity of the proposed residue multipliers.

In the following, the performance of the proposed residue multiplier is compared to the adder-based architectures presented in [7], [8], and [13]. The area complexity of the pseudoRNS multiplier [7] equals the complexity of $2.5\ n$-bit multipliers and $2\ n$-bit adders and the overall delay is $(7n-3)t_{\text{FA}}$. The complexity of the residue multiplier presented by Hiasat [13] is equal to $1.25\ n$-bit multipliers and $3\ n$-bit adders and the corresponding delay is equal to $(5n-1)t_{\text{FA}}$. It is noted that the area complexity of an n-bit multiplier is assumed equal to the complexity of $n(n-1)$ FAs and n^2 AND gates. The structure presented by Elleithy and Bayoumi [8] consists of n

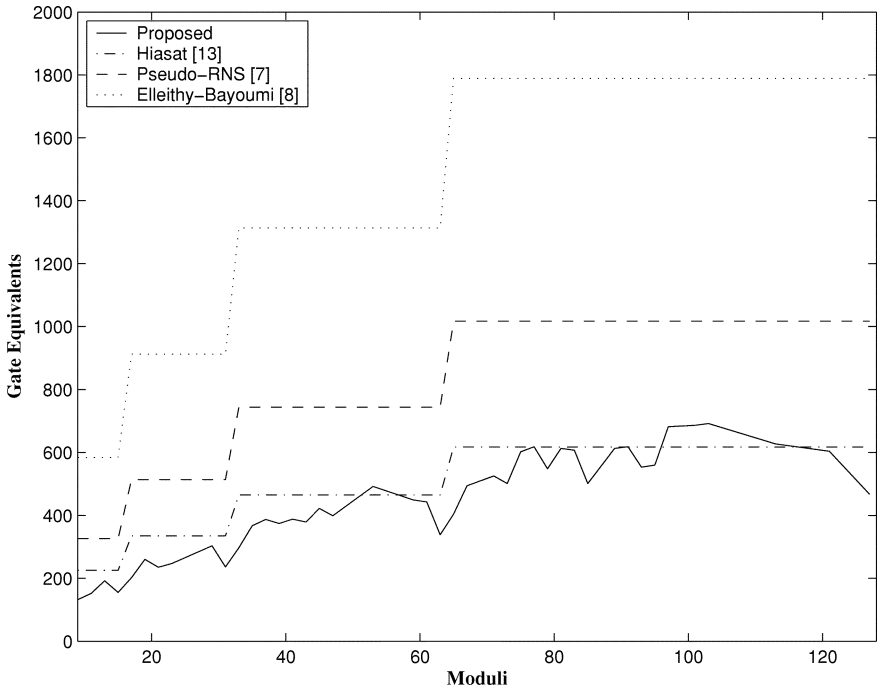


Fig. 8. Area performance of the proposed residue multiplier compared to the area complexity of [7], [8], and [13].

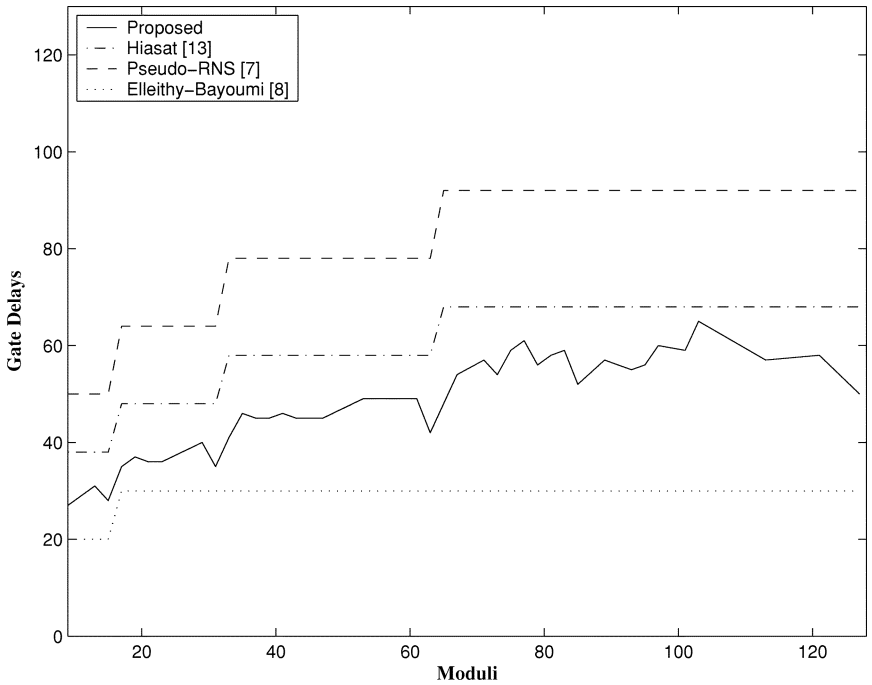


Fig. 9. Time performance of the proposed residue multiplier compared to the time complexity of [7], [8], and [13].

modular adders and n^2 AND gates, with a modular adder being equal to $5n$ FAs, while the overall delay is $(\,5\,\lceil \log_2 n \rceil\,)\,t_{\rm FA}$.

Fig. 8 reveals that the proposed residue multiplier is more efficient than the one proposed by Hiasat [13] when the corresponding n-bit modulo m is less than $2^{n-1}+2^{n-2}$ as defined by Proposition 3, while it is slightly worse when $m > 2^{n-1}+2^{n-2}$ since no BPCA-based optimization can be performed. The area reduction compared to the residue multipliers presented in [7] and [8] is 33% to 60% and 63% to 77%, respectively. The proposed residue multiplier is faster compared to the residue multipliers [7] and [13] as depicted in Fig. 9. It is noted that the delay of the architecture by Elleithy and Bayoumi stems from the tree organization of the modular adders, a technique, which can also be applied to all the other adder-based architectures for delay reduction.

In general, the proposed architecture along with the introduced optimization methodology leads to low-complexity residue multipliers compared to the most efficient architectures presented in literature so far to the authors' knowledge. The area $\times$ time savings achieved when compared to the most

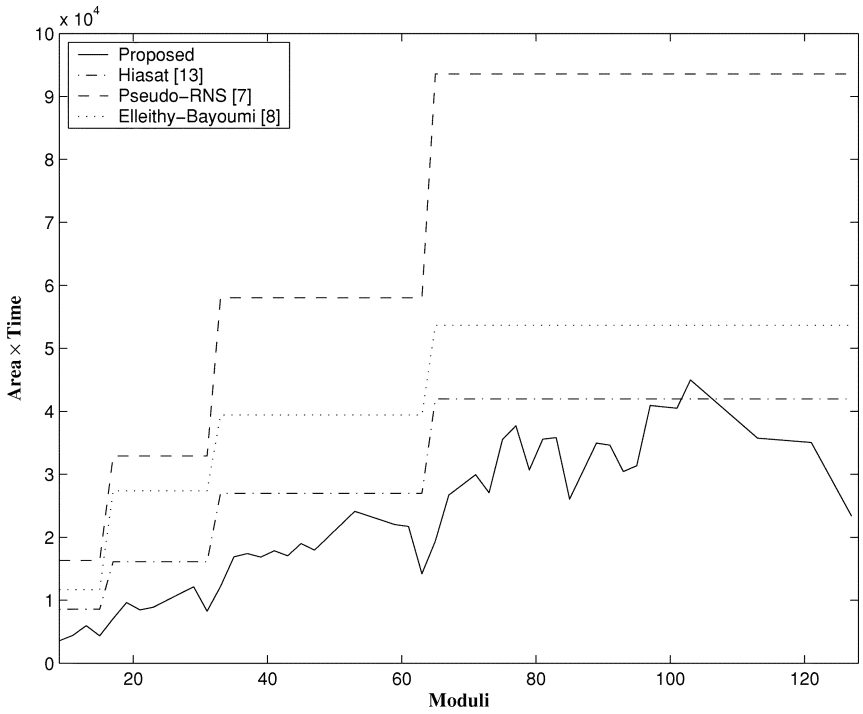


Fig. 10. Area × time performance of the proposed residue multiplier compared to the area × time complexity of [7], [8], and [13].

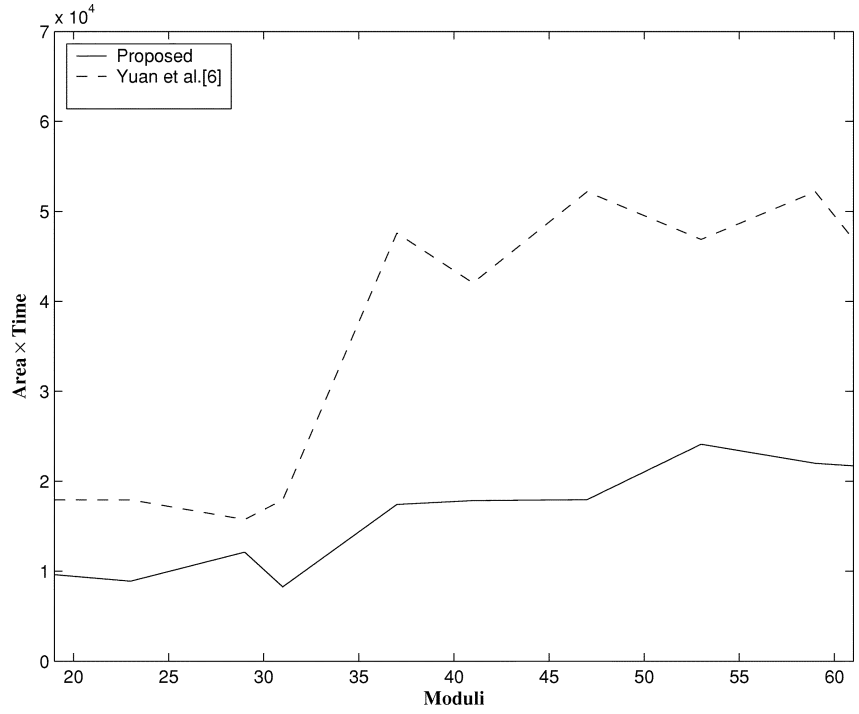


Fig. 11. Area × time complexity comparison of the proposed multiplier to [6].

competitive residue multiplier [13] range from 18% to 55%, as shown in Fig. 10

The proposed multiplier is finally compared to the submodular index transform-based multiplier of Radhakrishnan and Yuan [6] as shown in Fig. 11. In all cases, it is found that the proposed multiplier is more efficient in terms of area × time complexity achieving complexity savings of more than 55%. It is noted that the proposed multiplier is not limited to prime moduli as required by the index-transform multipliers.

VIII. Conclusion

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. Furthermore, the recursive modulo decomposition procedure used in previously proposed multipliers is replaced by a constant-complexity scheme. The complexity of the first stage of the proposed multiplier is minimized via an optimization methodology,

based on an assortment of introduced theorems and propositions, which allow the exploitation of data-dependent characteristics of the bit products, in a computationally efficient manner. The introduced methodology relies on the solution of the clique partitioning problem on the introduced BPCG graphs. In addition, a new property of inverted bit products is described. Complexity comparisons reveal that the derived residue multiplier is very efficient in the area $\times$ time sense, compared to previously reported designs.

APPENDIX

A. Proof of Theorem 1

Initially, assume that a_ib_j and a_kb_l, where a_ib_j, $a_kb_l \in \{0,1\}$, are compatible, i.e., $a_ib_j + a_kb_l \leq 1$, for all legitimate values of the input residues A and B. Therefore, there are no combinations of A and B for which

$$a_ib_j + a_kb_l > 1 \Leftrightarrow a_ib_j = 1 \bigwedge a_kb_l = 1 \Leftrightarrow a_i = 1 \bigwedge b_j = 1 \bigwedge a_k = 1 \bigwedge b_l = 1. \quad (46)$$

Due to (46), it follows that

$$a_i = 1 \bigwedge a_k = 1 \Leftrightarrow A = \sum_{r=0}^{n-1} a_r 2^r \geq a_i 2^i + a_k 2^k = 2^i + 2^k \quad (47)$$

$$b_j = 1 \bigwedge b_l = 1 \Leftrightarrow B = \sum_{r=0}^{n-1} b_r 2^r \geq b_j 2^j + b_l 2^l = 2^k + 2^l. \quad (48)$$

Equivalences (47) and (48) show that when $a_ib_j + a_kb_l \leq 1$ for all legitimate inputs, there are no combinations of A and B such that

$$A \geq 2^i + 2^k \bigwedge B \geq 2^j + 2^l \quad (49)$$

for every residue modulo-m value of A and B. Since (49) is false, it follows that at least one of the conditions in (49) is false, or both. Hence

$$2^i + 2^k \geq m \bigvee 2^j + 2^l \geq m \quad (50)$$

because $0 \leq A < m$ and $0 \leq B < m$. Since the bit products a_ib_j and a_kb_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

$$2^i + 2^k \geq m \bigvee 2^j + 2^l \geq m \quad (51)$$

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 $B = \sum_{j=0}^{n-1} b_j 2^j < m$. So, for each $i, k \in \{0, 1, \ldots, n-1\}$ with $i \neq k$

$$a_i 2^i + a_k 2^k < m \quad \bigwedge \quad b_j 2^j + b_l 2^l < m. \quad (52)$$

In the case that $a_i = a_k = b_j = b_l = 1$ there is a contradiction between (51) and (52). So, it obvious that (51) and (52) 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_ib_j or a_kb_l is also equal to zero and thus their sum is $a_ib_j + a_kb_l \leq 1$, since it is either zero or one.

B. Proof of Theorem 2

For all legitimate input combinations of input residues A and B, assume that it holds that

$$a_ib_j + a_kb_l \leq 1 \Leftrightarrow b_ja_i + b_la_k \leq 1 \quad (53)$$

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, from (53) it follows that $a_jb_i + a_lb_k \leq 1$, for all legitimate input values. Hence, the bit products a_jb_i and a_lb_k are compatible.

C. Proof of Proposition 1

Due to Theorem 1 the sum of bit products $a_ib_j + a_{i+w}b_{j-t}$ is less or equal to one when

$$2^i + 2^{i+w} \geq m \bigvee 2^j + 2^{j-t} \geq m. \quad (54)$$

The first case $2^i + 2^{i+w} \geq m$ implies that $2^i(1 + 2^w) \geq m$ and since $2^w + 1 > 0$ for each $0 \leq i \leq n-1$ it follows that

$$2^i \geq \frac{m}{2^w + 1} \quad (55)$$

and thus

$$i \geq \left\lceil \log_2 \left(\frac{m}{2^w + 1} \right) \right\rceil \quad (56)$$

since i is integer. In addition, in order the indexes $i+w$ and $j-t$ to assume legitimate values, the following constraints should be satisfied:

$$0 \leq i + w \leq n-1 \quad \bigwedge \quad 0 \leq j - t \leq n-1 \quad (57)$$

or, equivalently

$$-w \leq i \leq n-1-w \quad \bigwedge \quad t \leq j \leq n-1+t. \quad (58)$$

Due to (56) and (58) and the fact that $j \leq n-1$, it follows that:

$$\left\lceil \log_2 \left(\frac{m}{2^w + 1} \right) \right\rceil \leq i \leq n-1-w \quad (59)$$

and

$$t \leq j \leq n-1. \quad (60)$$

Due to Theorem 1, it follows that the indexes of the examined bit products should satisfy the constraint $i \neq i + w \vee j \neq t - j$. By setting $w \neq 0$, i.e., $1 \leq w \leq n-1$, the produced compatible bit-product pairs remain valid in all cases.

The case $2^j + 2^{j-t} \geq m$ is similar to $2^i + 2^{i+w} \geq m$ when the bit products a_jb_i and $a_{j-t}b_{i+w}$ are considered. Therefore, since Theorem 2 guarantees that the bit products $a_{j-t}b_{i+w}$ are compatible to a_jb_i when $a_{i+w}b_{j-t} + a_ib_j \leq 1$, then the case $2^j + 2^{j-t} \geq m$ follows.

D. Proof of Proposition 2

Due to Theorem 1, if $a_ib_j + a_{i+w}b_{t-j} \leq 1$ then $2^i + 2^{i+w} \geq m \bigvee 2^j + 2^{t-j} \geq m$. The constraints for i can be proved in the same way as in Proposition 1. Additionally, in order for the index $t-j$ to be legitimate, the following inequality must hold:

$$0 \leq t - j \leq n-1 \quad (61)$$

which implies that $t \geq j \geq -n + 1 + t$. Since $j \geq 0$ and $t - n + 1 \leq 0$, it follows that $0 \leq j \leq t$. For the case of $2^j +$

$2^{t-j} \geq m$, similar assumptions as in the proof of Proposition 1 can be made and thus needs no further analysis.

E. Proof of Proposition 3

By merging the constraints of Propositions 1 and 2 in order $a_i b_j + a_{i+w} b_{j-t} \leq 1$ and $a_i b_j + a_{i+w} b_{t-j} \leq 1$ to hold, the indexes i, j have to be constrained as follows:

$$\left\lceil \log_2 \left(\frac{m}{2^w + 1} \right) \right\rceil \leq i \leq n - 1 - w \quad \text{and} \quad 0 \leq j \leq n - 1. \tag{62}$$

Since the index j spans the whole index space from 0 to $n-1$, no further analysis is needed. Therefore, from (62), it can be assumed that there exists at least one compatible bit-product pair when

$$\left\lceil \log_2 \left(\frac{m}{2^w + 1} \right) \right\rceil \leq n - 1 - w \tag{63}$$

in order the constraint for index i to be valid. Due to the fact that $x \leq \lceil x \rceil < x + 1$, from (63) it follows that:

$$\log_2 \left(\frac{m}{2^w + 1} \right) \leq n - 1 - w. \tag{64}$$

Equation (64) implies that

$$\frac{m}{2^w + 1} \leq 2^{n-1-w} \tag{65}$$

which can be expressed as

$$m \leq 2^{n-1} + 2^{n-1-w} \tag{66}$$

where $1 \leq w \leq n-1$. Since for every n-bit modulo holds that $2^{n-1} \leq m < 2^n$, then the values of m that allow the existence of compatible bit products for all legitimate input combinations, are

$$2^{n-1} \leq m \leq 2^{n-1} + 2^{n-2}. \tag{67}$$

Assume that $m > 2^{n-1} + 2^{n-2}$. Since $2^w + 1 > 0$, $1 \leq w \leq n-1$, it is obtained that

$$\frac{m}{2^w + 1} > \frac{2^{n-1} + 2^{n-2}}{2^w + 1} > 1 \tag{68}$$

for $n > 2$ and $w > 1$. Equation (68) gives

$$\log_2 \left(\frac{m}{2^w + 1} \right) > \log_2 \left(\frac{2^{n-1} + 2^{n-2}}{2^w + 1} \right) \Rightarrow$$
$$\Rightarrow \left\lceil \log_2 \left(\frac{m}{2^w + 1} \right) \right\rceil > \log_2 \left(\frac{2^{n-1} + 2^{n-2}}{2^w + 1} \right) \tag{69}$$

since $\lceil x \rceil \geq x$. It holds that

$$\log_2 \left(\frac{2^{n-1} + 2^{n-2}}{2^w + 1} \right) > n - 1 - w \tag{70}$$

because (70) is equivalent to

$$\frac{2^{n-1} + 2^{n-2}}{2^w + 1} > 2^{n-1-w}$$
$$\Leftrightarrow 2^{n-1} + 2^{n-2} > (2^w + 1) 2^{n-1-w} \tag{71}$$
$$\Leftrightarrow 2^{n-1} + 2^{n-2} > 2^{n-1} + 2^{n-1-w} \tag{72}$$
$$\Leftrightarrow 2^{n-2} > 2^{n-1-w} \tag{73}$$
$$\Leftrightarrow n - 2 > n - 1 - w \tag{74}$$
$$\Leftrightarrow w > 1 \tag{75}$$

which is true. Hence, (69) and (70) reveal that $\lceil \log_2 (m/(2^w + 1)) \rceil > n - 1 - w$, when $m > 2^{n-1} + 2^{n-2}$ and $w > 1$, which means that no compatible bit products exist.

In the case where $w = 1$, it can be written that

$$\left\lceil \log_2 \left(\frac{m}{2^w + 1} \right) \right\rceil$$
$$= \left\lceil \log_2 \left(\frac{m}{3} \right) \right\rceil \geq \log_2 \left(\frac{m}{3} \right) > \log_2 \left(\frac{2^{n-1} + 2^{n-2}}{3} \right)$$
$$= \log_2 \left(2^{n-2} \frac{2+1}{3} \right) = n - 2 \tag{76}$$

hence $\lceil \log_2 (m/(2^w + 1)) \rceil > n-2 = n-1-w$, and therefore no compatible bit products exist for $w \geq 1$.

F. Proof of Theorem 3

First, suppose that $x_1 + x_2 + \cdots + x_N \leq 1$, which implies that there is at most one of all x_i such that $x_i = 1$. Based on this conclusion two cases can be distinguished.

- There is no x_i, such that $x_i = 1$. Hence, for each i, j with $i \neq j$, $x_i + x_j = 0$.
- There is exactly one x_i such that $x_i = 1$. This implies that $x_i + x_j = 1 + 0 = 1$ for each $i \neq j$ and $x_k + x_j = 0$ for each $i \neq k \neq j \neq i$.

Therefore, from both cases it follows that if $x_1 + x_2 + \cdots + x_N \leq 1$ then $x_i + x_j \leq 1$ for each i, j with $1 \leq i, j \leq N$ and $i \neq j$.

In the following assume that:

$$x_i + x_j \leq 1 \tag{77}$$

for each i, j with $i \neq j$. Therefore, in order the sum of each pair x_i, x_j to be less or equal to one, the following cases are distinguished.

- For every i, with $1 \leq i \leq N$, $x_i = 0$. In this case $\sum_{i=1}^{N} x_i = 0$.
- There exists at least one i such that $x_i = 1$. So, in order (77) to be satisfied, the term $x_i = 1$ must be unique and thus every x_j with $j \neq i$ must be equal to zero. In this case

$$\sum_{i=1}^{N} x_i = x_i + \sum_{j \neq i} x_j = 1 + 0 = 1. \tag{78}$$

Thus, in every case, the sum of the N integers x_i is $\sum_{i=1}^{N} x_i \leq 1$.

G. Proof of Theorem 4

Due to the fact that $\bar{x} = 1 - x$ the sum of the inverted bits $\bar{x}_i$ can be expressed as

$$\sum_{i=1}^{N} \bar{x}_i = N - \sum_{i=1}^{N} x_i. \tag{79}$$

Since $0 \leq x_1 + x_2 + \cdots + x_N \leq 1$, it follows that $0 \geq -(x_1 + x_2 + \cdots + x_N) \geq -1$. Adding N to the inequalities, it follows that:

$$N \geq N - (x_1 + x_2 + \cdots + x_N) \geq N - 1. \tag{80}$$

Therefore, by substituting (79) to (80), it is derived that $N \geq \bar{x}_1 + \bar{x}_2 + \cdots + \bar{x}_N \geq N - 1$.

REFERENCES

[1] N. Szabó and R. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. New York: McGraw-Hill, 1967.

[2] F. J. Taylor, "Residue arithmetic: A tutorial with examples," *IEEE Comput.*, vol. 17, pp. 50–62, May 1984.

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

[4] X. Lai and J. L. Massey, "A proposal for a new block encryption standard," in *Proc. Advances in Cryptology EUROCRYPT 90*, Berlin, Germany, 1990, pp. 389–404.

[5] G. A. Jullien, "Implementation of multiplication, modulo a prime number, with applications to number theoretic transforms," *IEEE Trans. Comput.*, vol. C-29, pp. 899–905, Oct. 1980.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

[12] V. Paliouras, K. Karagianni, and T. Stouraitis, "A low-complexity combinatorial RNS multiplier," *IEEE Trans. Circuits Syst. II*, vol. 48, pp. 675–683, July 2001.

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

[14] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 389–400, 1961.

[15] K. Parhi, *VLSI Digital Signal Processing Systems*. New York: Wiley, 1999.

[16] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 677–688, Oct. 1996.

[17] A. Wrzyszcz, D. Milford, and E. L. Dagless, "A new approach to fixed-coefficient inner product computation over finite rings," *IEEE Trans. Comput.*, vol. 45, pp. 1345–1355, Dec. 1996.

[18] S. J. Piestrak, "Design of residue generators and multioperand modular adders using carry–save adders," *IEEE Trans. Comput.*, vol. 43, pp. 68–77, Jan. 1994.

[19] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[20] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, Feb. 1964.

[21] L. Dadda, "Some schemes for parallel multipliers," *Alta Freq.*, vol. 34, pp. 349–356, 1965.

[22] B. Parhami, *Computer Arithmetic—Algorithms and Hardware Designs*. New York: Oxford Univ. Press, 2000.

[23] G. De Michelli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.

[24] *ES2 ECPD07 Library Databook*, ES2, Eur. Silicon Structures, Rousset, France, 1992.

Giorgos Dimitrakopoulos was born in Komotini, Greece, in 1978. He received the Diploma in computer engineering and informatics in 2001 from University of Patras, Patras, Greece, where he is currently working toward the M. Sc. degree.

Since 2001, he has been a Research Assistant at the Technology and Computer Architecture Laboratory of the Computer Engineering and Informatics Department, University of Patras. His research interests include computer arithmetic, VLSI signal processing architectures, and testing of digital systems.

Vassilis Paliouras (S'90–M'95) received the Diploma and the Ph.D. degrees in electrical engineering from the Electrical and Computer Engineering (ECE) Department, University of Patras, Patras, Greece, in 1992 and 1999, respectively.

He is an Assistant Professor at the ECE Department, where he teaches digital signal processing and VLSI system design-related courses. His research interests include computer arithmetic algorithms and circuits, and VLSI signal processing for video and communications. He has published more than 40 conference and journal articles.

Dr. Paliouras received the MEDCHIP VLSI Design Award in 1997, and the 2000 IEEE Circuits and Systems Society Guillemin–Cauer Best Paper Award. He serves as the Cochair of the 2004 International Workshop on Power and Timing Modeling, Optimization, and Simulation (PATMOS), Santorini, Greece.