Design Efficient Approximate Multiplication Circuits Through Partial Product Perforation

Georgios Zervakis, Kostas Tsoumanis, Sotirios Xydis, Dimitrios Soudris and Kiamal Pekmestzi

Abstract—**Approximate computing has received significant attention as a promising strategy to decrease power consumption of inherently error tolerant applications. In this paper, we focus on hardware level approximation by introducing the Partial Product Perforation technique for designing approximate multiplication circuits. We prove in a mathematically rigorous manner that in partial product perforation the imposed errors are bounded and predictable, depending only on the input distribution. Through extensive experimental evaluation, we apply the partial product perforation method on different multiplier architectures and expose the optimal architecture–perforation configuration pairs for different error constraints. We show that, compared with the respective exact design, the partial product perforation delivers reductions of up to 50% in power consumption, 45% in area and 35% in critical delay. Also, the product perforation method is compared with state-of-the-art approximation techniques, i.e. truncation, Voltage Over-Scaling and logic approximation, showing that it outperforms them in terms of power dissipation and error.**

Index Terms—**Approximate computing, approximate multiplier, approximate arithmetic circuits, error analysis, low power**

I. Introduction

In modern embedded electronic devices, power consumption is a first class design concern. Considering that a large number of application domains are inherently tolerant to imprecise calculations, e.g. Digital Signal Processing (DSP), data analytics and data mining [1] approximate computing appears as a promising solution to reduce their power dissipation. Such applications a) process large redundant data sets or noisy input data derived from the real world, b) do not have a "golden" result, c) perform statistical/probabilistic computations and/or d) demand human interaction, thus, their exactness is relaxed due to limited human perception [2], [3]. Approximate computing can be applied at both software and hardware level.

Hardware level approximation mainly targets arithmetic units, such as adders and multipliers widely used in portable devices to implement multimedia algorithms, e.g., image and video processing. The most commonly used techniques for the generation of approximate arithmetic circuits are truncation [4], [5], Voltage Over-Scaling (VOS) [2], [6] and simplification of logic complexity (i.e., alteration of the truth table) [7]–[9]. Extensive research has been conducted on approximate adders [6], [7], [10], [11] providing significant gains in terms of area and power while exposing small error. However, research activities on approximate multipliers are limited. Efficient approximate multipliers introduced in [8], [9], [12], [13] target the approximation of the partial product accumulation but do not examine approximations on the partial product generation.

Approximate hardware circuits, contrary to software approximations, offer transistors reduction, lower dynamic and leakage power, lower circuit delay and opportunity for downsizing. Motivated by the limited research on approximate multipliers, compared to the extensive research on approximate adders, and explicitly the lack of approximate techniques targeting the partial product generation, we introduce the Partial Product Perforation method for creating approximate multipliers. Inspired from [14], we omit the generation of some partial products, thus, reducing the number of partial products that have to be accumulated, we decrease the area, power and depth of the accumulation tree. The major contributions of this work are summarized as follows:

- We adopt and apply, for the first time, the software based perforation technique [14], on the design of hardware circuits, obtaining optimized design solutions regarding the power–area–error trade-offs.
- We analyze in a mathematically rigorous manner the arithmetic accuracy of partial product perforation and prove that it delivers a bounded and predictable output error. Our error analysis is not bound to a specific multiplier architecture and can be applied with error guarantees to every multiplication circuit regardless of its architecture. Such a rigorous analysis enables precise error estimation over input data distributions.
- We explore and characterize the efficiency of the product perforation method on several multiplier schemes exposing its power–area impact on different architectures. This is the first time that such an exploratory analysis over different approximate multiplier architectures is offered to the designer, enabling also, the selection of the optimum architecture–perforation configuration for given error constraints.
- We show that partial product perforation outperforms related state-of-the-art works in terms of power consumption and error, as well as output quality, when applied to image processing and data analytics algorithms.

More specifically, we apply the partial product perforation on 16 different multiplier architectures, using industrial strength tools, i.e. Synopsys Design Compiler and PrimeTime. Through extensive experimental evaluation, we present the

Manuscript received September 15, 2015; revised January 4, 2016; accepted February 9, 2016. This work has been partially supported by the E.C. program AEGLE under H2020 Grant Agreement No: 644906.

The authors are with the Department of Electrical and Computer Engineering, National Technical University of Athens. E-mail: {zervakis, kostastsoumanis, sxydis, dsoudris, pekmes}@microlab.ntua.gr

optimal approximate multiplier configurations for various error constraints. We show that, compared to the accurate multiplier, product perforation offers reductions of up to 50% in power consumption, 45% in area and 35% in critical delay for 0.1% normalized mean error distance [15]. Moreover, it is compared with state-of-the-art approximate computing works that use either VOS [6], logic approximation [9], or truncation [4], outperforming them significantly in terms of power dissipation and error. Finally, we examine the scalability of our technique by applying it on different bit-width multipliers and show that the delivered savings increase with the width increase.

The rest of the paper is organized as follows: In Section II, we discuss related literature with emphasis on circuit level approximation. Section III introduces the partial product perforation technique providing the corresponding error analysis error and error correction methods. In Section IV, we examine product perforation on different multiplier architectures, exposing the optimal architecture–perforation configuration pairs under differing error constraints. Finally, Section V evaluates the product perforation method by comparing it with related state-of-the-art works and Section VI concludes the paper.

II. Related Work

In this section, related research in the field of hardware approximate computing is discussed. Both general-purpose approximation techniques [4], [6], [16] applied to any arithmetic circuit, as well as circuit-specific approximation either to adder [7], [10], [11] or multiplier designs [8], [9], [13], [17], [18], have been presented.

Regarding to the general approximation techniques, VOS [2], [6] and truncation [4], [5], [12] have been proposed. VOS is applied in any circuit by lowering the supply voltage below its nominal value. Decreasing the supply voltage reduces the circuit's power consumption, but produces errors caused by the number of paths that fail to meet the delay constraints [2]. In [12], the authors proposed an automated generation of large precision floating point multipliers in FPGAs, using sophisticated truncation over underutilized DSPs. In [5], a truncated multiplier with a constant correction term is proposed, significantly decreasing the error imposed by typical truncation. [4] proposed a truncated multiplier with variable correction that outperforms [5] in terms of error. Probabilistic pruning and logic minimization techniques have been presented in [16], using a greedy approach to generate approximate circuits. These techniques systematically eliminate circuit's components and simplify logic complexity according to the circuit's activity profile and output significance. Both techniques heavily depend on the application's characteristics, and in addition the induced approximation error are not rigorously bounded.

Extensive research has been conducted targeting the implementation of approximate adders [7], [10], [11]. In [11], the authors developed a probability proof, estimating that the longest carry chain in an *n*-bit adder is log*n*, and produced a fast inexact adder limiting the carry propagation. In [10], approximation is performed by decomposing the addition circuit in an accurate and an approximate inaccurate part. In [7], the authors build imprecise full adder cells, requiring fewer transistors, by approximating their logic function and then use them to build imprecise adders. Although the authors propose the use of such adders targeting to build approximate multipliers, it is not clear how they can be used in different tree architectures and how their error scales in the case of multi-operand addition. Targeting the creation of approximate multipliers, [8] proposed a simplified imprecise 2x2 multiplier cell used as the basic block for constructing larger multiplier architectures. [9] presented two approximate 4:2 compressors by modifying the respective accurate truth table, which were then used to build two approximate multipliers outperforming [8]. The approximate compressors of [9] are used in Dadda tree with 4:2 reduction. However, different multiplier architectures were not explored. Based on an approximate adder that limits the carry propagation, [13] presented a fast and low-power multiplier scheme with higher error than [9]. However, in all the aforementioned approaches, the imposed error cannot be predicted as it depends on carry propagation and the circuits' implementation and requires simulations over all possible inputs in order to be calculated.

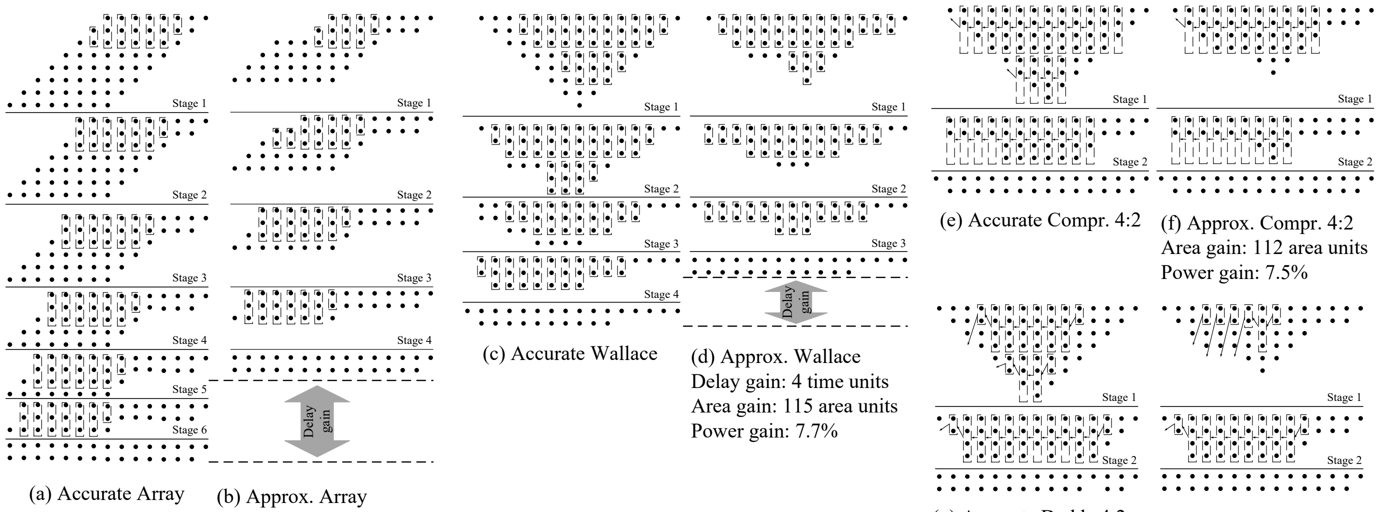
Recently, [17], [18] proposed the use of $m \times m$ multipliers to perform an $n \times n$ multiplication (with $m < n$). In [17] the authors statically split the multiplicand in three *m*-bit segments and perform the multiplication utilizing the segment containing the most significant 1 (leading one). However, as stated in [18], *m* needs to be at least $n/2$ to attain acceptable accuracy, thus limiting the energy savings and the scalability of this approach. In [18] the authors extended the idea of leading-one segments to enable dynamic range multiplication and added a correction term. Although [18] delivers higher accuracy designs than [17] using smaller values for *m*, their approach requires the allocation of extra complex circuitry, i.e. two leading one detectors, two complex multiplexers for segment selection, one log(*n*)-bit comparator, a log(*n*)-bit adder, and one 2*n*-bit barrel shifter. These extra components are expected to highly increase the circuits complexity introducing non trivial delay, area, and energy overheads that may considerably decrease the approximation benefits [17]. This is expected to be more evident in designs targeting too small error values, in which the need of larger *m* values is required.

In this paper, we target the design of power–error efficient multiplication circuits. We differentiate from previous works by exploring approximation on the generation of the partial products. The proposed method can be easily applied in any multiplier architecture without the need of a special design, in contrast to related works. In addition, the error imposed by perforation depends only on the configuration parameters and, in contrast to existing work, can be analytically calculated without the need of exhaustive simulations. The latter is critical as, given the application's inputs, a precise estimation of the output quality can be extracted. Finally, the knowledge of the induced error permits the selection of the configuration that maximizes the power savings for a specific error bound.

III. Analyzing Partial Product Perforation

A. Method Analysis

In this section, the partial product perforation method for the design of approximate hardware multipliers is described.



(a) Accurate Array

(b) Approx. Array
Delay gain: 8 time units
Area gain: 112 area units
Power gain: 7.8%

(c) Accurate Wallace

(d) Approx. Wallace
Delay gain: 4 time units
Area gain: 115 area units
Power gain: 7.7%

(e) Accurate Compr. 4:2

(f) Approx. Compr. 4:2
Area gain: 112 area units
Power gain: 7.5%

(g) Accurate Dadda 4:2

(h) Approx. Dadda 4:2
Area gain: 127 area units
Power gain: 7.8%

Fig. 1. The partial product reduction process for 8×8 multiplication with a) Accurate Array, b) Approx. Array c) Accurate Wallace, d) Approx. Wallace, e) Accurate Compressor 4:2, f) Approx. Compressor 4:2, g) Accurate Dadda 4:2 h) Approx. Dadda 4:2. Approximation is performed by perforating the 3^{rd} and 4^{th} partial products. The boxes with 4 dots are 4:2 compressors, those with 3 are full adders and those with 2 are full or half adders.

Consider two n-bit numbers A and B. The result of their multiplication $A \times B$ is obtained after summing all the partial products Ab_i, where b_i is the i^{th} bit of B. Thus,

$$A \times B = \sum_{i=0}^{n-1} Ab_i 2^i,\ b_i \in \{0,1\}. \tag{1}$$

The partial product perforation technique omits the generation of k successive partial products starting from the j^{th} one. A perforated partial product is not inserted in the accumulation tree and hence, n full adders can be eliminated. Applying product perforation with j and k configuration values on the multiplication $A \times B$ produces the approximate result

$$A \times B|_{j,k} = \sum_{\substack{i=0,\\ i \notin [j,j+k)}}^{n-1} Ab_i 2^i,\ b_i \in \{0,1\}. \tag{2}$$

Note that $j \in [0, n-1]$ and $k \in [1, \min(n-j, n-1)]$.

Similarly, when Modified Booth Encoding (MBE) [19] is used for generating the partial products, the result of the approximate multiplication is given by:

$$A \times B|_{j,k} = \sum_{\substack{i=0\\ i \notin [j,j+k)}}^{n/2-1} A\mathbf{b}_i^{MB} 4^i,\ \mathbf{b}_i^{MB} \in \{0, \pm 1, \pm 2\}. \tag{3}$$

Fig. 1 depicts an example of applying the partial product perforation method on different 8-bit multipliers with j=2 and k=2 configuration values. For each architecture, the dot diagrams [19] of the accurate and the respective perforated tree are presented. The "dots" represent the bits of the partial products that have to be accumulated, while the "stages" the delay of the reduction process followed by each tree. The dashed boxes with four dots are 4:2 compressors, those with three are full adders and those with two are either full- or half-adders. Through the proposed approximation technique, the power, area and delay of the multiplication circuit are decreased, making though the computation imprecise. The higher the order of a perforated partial product, the greater the error imposed at the final result. Also, since the addition is an associative and commutative operation, when more than one partial products are perforated, the total error results from the addition of the errors produced from the perforation of each partial product separately.

We use **the notation D[*j*,*k*,*c*]** to label the different approximate multiplier architectural configurations. The parameter "D" refers to the tree architecture, j is the order of the first perforated partial product and k the number of the perforated partial products. If no j and k are specified, the respective notation refers to the exact design. Finally, c corresponds to the partial product generation technique and takes the values "s" for Simple Partial Products (SPP) or "m" for MBE. For example, Fig. 1a depicts the array[s] configuration, while Fig. 1b the array[2,2,s].

Partial product perforation should not be confused with the truncation technique. Truncation eliminates the circuit that produces specific least significant bits (LSB) of the accumulation tree, while perforation skips the generation of partial products and thus, decreases the number of operands to be accumulated. For example, in an 8-bit array multiplier, perforating a partial product removes 8 full adders from the accumulation tree and reduces its delay. In order to attain similar circuit reduction using truncation, the 6 LSB have to be truncated. However, truncating the 6 LSB does not offer any delay reduction. Moreover, in this example, truncation delivers in all cases incorrect results, whereas the outputs of perforation are 50% correct. Finally, perforating one partial product (out of eight) results in a 12.5% loss of information while truncating the 6 LSB (out of 16) results in a 37.5% information loss. In Section V, the perforation and truncation techniques are quantitatively compared in greater detail regarding error and power metrics, in order to further expose their differences.

B. Error Analysis

A critical issue for the approximate computing is the error imposed during computations and how it affects the final result. In this section, an error evaluation analysis of the partial product perforation technique is presented. We evaluate error utilizing the error metrics proposed in [15], i.e. Error Distance (*ED*), Mean Error Distance (*MED*) and Normalized *MED* (*NMED*), as effective metrics for quantifying the accuracy of approximate arithmetic circuits. *ED* is defined as the absolute distance of the fully accurate product *P* and the approximate one P', $ED = |P - P'|$. The *MED* is the average of *EDs* for all inputs and $NMED = MED/P_{max}$, where $P_{max} = (2^n - 1)^2$ in the case of an *n*-bit multiplier [13]. The Relative Error Distance (*RED*) is defined as $RED = ED/P$ and the Mean *RED* (*MRED*) is similarly obtained [13].

1) Error Evaluation: When applying the product perforation on a *n*-bit multiplier using SPP generation, the *ED* of multiplying two numbers *A*, *B* is calculated as follows:

$$\begin{aligned} ED(A,B) = \left|P - P'\right| &= A\sum_{i=0}^{n-1} b_i 2^i - A \sum_{\substack{i=0,\\ i\notin[j,j+k)}}^{n-1} b_i 2^i \\ &= A\sum_{i=j}^{j+k-1} 2^i b_i = A2^j x_B, \end{aligned} \tag{4}$$

where $x_B \in [0, 2^k)$ and

$$x_B = \sum_{i=0}^{k-1} 2^i b_{j+i} = \lfloor B/2^j \rfloor \bmod 2^k. \tag{5}$$

If p_A and p_B are the probability density functions of *A* and *B*, respectively, then the *MED* is calculated from:

$$MED = \sum_{\forall A,B} p_A(A) p_B(B) ED(A,B). \tag{6}$$

Without loss of generality, the rest of our analysis considers a uniform distribution over the overall n-bit numbers, i.e., $(A,B) \in [0,2^n)^2$. Hence, $p_A(A) = 1/2^n \ \forall A$ and $p_B(B) = 1/2^n \ \forall B$. Therefore, *MED* is given from:

$$MED = \sum_{\forall A,B} \frac{ED(A,B)}{2^n 2^n} = \frac{1}{2^{2n}} \sum_{\forall A}\sum_{\forall B} ED(A,B). \tag{7}$$

Assuming that ED_A is the sum of *EDs* $\forall B$ for a given *A*, then:

$$\begin{aligned} ED_A = \sum_{\forall B} ED(A,B) &= 2^{n-k} \sum_{\forall x_B} x_B 2^j A \\ &= \frac{2^n 2^j (2^k - 1) A}{2} \end{aligned} \tag{8}$$

and the sum of all *EDs* is:

$$\begin{aligned} \sum_{\forall A} ED_A = \sum_{\forall A} \frac{2^n 2^j (2^k-1)A}{2} &= \frac{2^n 2^j (2^k - 1)}{2} \Big(\sum_{A=0}^{2^n - 1} A \Big) \\ &= \frac{2^j 2^{2n} (2^k - 1)(2^n - 1)}{4}. \end{aligned} \tag{9}$$

Using (9), (7) equals:

$$MED = \frac{2^j 2^{2n}(2^k - 1)(2^n - 1)}{2^{2n} 4} = \frac{2^j (2^k - 1)(2^n - 1)}{4}. \tag{10}$$

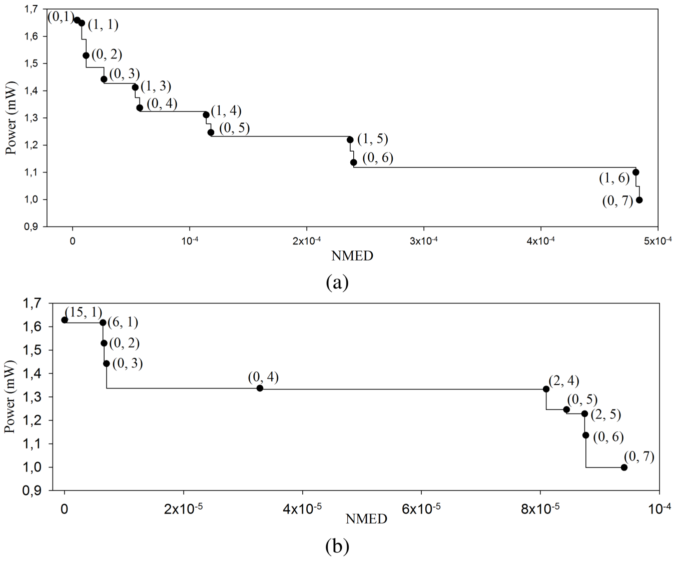


Fig. 2. The Pareto power–*NMED* graph of a 16-bit Dadda 4:2 multiplier with a) uniform input distribution in $[0, 2^{16})$ and b) inputs obtained from audio benchmarks. All the configurations that feature $NMED < 5 \times 10^{-5}$ are presented. Next to each point is denoted the respective (*j, k*) configuration.

Thus,

$$NMED = \frac{MED}{(2^n - 1)^2} = \frac{2^j(2^k - 1)}{4(2^n - 1)}. \tag{11}$$

Similarly,

$$RED(A,B) = \frac{ED(A,B)}{A \times B} = \frac{x_B 2^j}{B} \tag{12}$$

and

$$MRED = \frac{2^n}{2^{2n}} \sum_{\forall B} \frac{x_B 2^j}{B} = \sum_{\forall B} \frac{x_B 2^j}{2^n B}. \tag{13}$$

Previous analysis provide rigorous expressions of error metrics, enabling fast error analysis of differing product perforation configurations. As shown later in Section IV, these analytical error expressions are used in an exploration loop for deriving optimized approximate design solutions. The analytical equations (11) and (13) consider uniform distribution, thus in case of differing distributions[1] they should be adjusted according to the new probability density functions (PDF), since the power–error efficiency of approximate designs highly depends on the multiplier's operands distribution. In most applications, e.g. multimedia, the inputs are highly correlated [16]. As an intuitive example, Fig. 2a depicts the power–*NMED* Pareto graph for a 16-bit Dadda 4:2 multiplier when *A, B* follow the uniform distribution over the overall range of *n*-bit numbers, while Fig. 2b presents the same graph with inputs derived from the GSM 06.10 audio benchmark [20]. As shown, increasing *k* values result to lower power consumption but increased error values, while the selection of the *j* value mostly depends on the input distribution. Intuitively, for a uniform distribution over all possible *n*-bit numbers (Fig. 2a), where all the bits have equal probability of being one or zero, *j* should be kept small to minimize the error. This is also confirmed from Fig. 2a where the 58% of the Pareto configurations feature

[1]In case of different input distributions, starting from equation (6) we apply the same steps given the respective PDFs of the input operands.

$j=0$ and the 42% $j=1$. However, as presented in Fig. 2b, when the inputs are correlated without following a uniform distribution, we observe that the Pareto front is formed by configurations featuring many different *j* values, i.e., 0, 2, 6, and 15. Previous example shows that there is not a "golden" value for the *j* and *k* but their selection highly depends on the error constraints and the inputs PDF.

2) Error Correction Methods: In this section, we introduce two methods to decrease the error induced from the application of partial product perforation. They are implemented as extra components complementing the multiplication circuit, thus their area, power and delay overheads as well as the error reduction they offer, do not depend on the architecture of the multiplier. Although multiplication is commutative, i.e. $A \times B = B \times A$, this does not apply in perforated multipliers. From (4), when multiplying $A \times B$, the imposed error is proportional to the multiplicand *A* and the term x_B and thus, decreasing one of these operands decreases the error delivered to the output. As a result, comparing *A,B* or x_A, x_B before the multiplication and swapping accordingly *A,B* can reduce the error.

- **Method 1**: Comparing x_A, x_B

In this method x_A, x_B are compared before the multiplication and, if $x_B > x_A$, A and B are swapped. Therefore, the imposed error is $ED(A,B) = A2^j x_B$, when $x_A \geq x_B$, and $ED(A,B) = B2^j x_A$, when $x_B > x_A$. Hence, *MED* equals:

$$\begin{aligned} MED &= \sum_{\forall A,B} p_A(A)p_B(B)ED(A,B) \\ &= 2^j\Big(\sum_{\substack{\forall A,B:\\ x_A \geq x_B}} p_A(A)p_B(B)x_B A \\ &+ \sum_{\substack{\forall A,B:\\ x_A < x_B}} p_A(A)p_B(B)x_A B\Big). \end{aligned} \tag{14}$$

If *A*, *B* follow the uniform distribution in $[0,2^n)$ (14) equals:

$$\begin{aligned} MED &= 2^j\Big(\sum_{\substack{\forall A,B:\\ x_A \geq x_B}} \frac{x_B A}{2^n 2^n} + \sum_{\substack{\forall A,B:\\ x_A < x_B}} \frac{x_A B}{2^n 2^n}\Big) \\ &= \frac{2^j}{2^{2n}}\Big(\sum_{\substack{\forall A,B:\\ x_A = x_B}} x_B A + 2\sum_{\substack{\forall A,B:\\ x_A < x_B}} x_A B\Big). \end{aligned} \tag{15}$$

Every number A can be written in the form:
$A = M_A 2^{j+k} + x_A 2^j + L_A$, where $M_A \in [0, 2^{n-(j+k)})$, $x_A \in [0,2^k)$ and $L_A \in [0,2^j)$. M_A and L_A are computed similarly to x_A.
The sum ($S1(y)$) of all numbers A that have $x_A = y$, where y is a constant and $y \in [0,2^k)$, is given by:

$$\begin{aligned} S1(y) &= \sum_{\substack{\forall A:\\ x_A = y}} A = \sum_{\substack{\forall A:\\ x_A = y}} \big(M_A 2^{j+k} + x_A 2^j + L_A\big) \\ &= \sum_{\forall M_A}\sum_{x_A = y}\sum_{\forall L_A} \big(M_A 2^{j+k} + x_A 2^j + L_A\big) \\ &= 2^j \frac{(2^{n-(j+k)}-1)2^{n-(j+k)}}{2} 2^{j+k} + \\ &+ 2^{n-(j+k)} 2^j y 2^j + \\ &+ 2^{n-(j+k)} \frac{(2^j-1)2^j}{2}. \end{aligned} \tag{16}$$

Supposing that B is fixed and $x_B = z$, we get that:

$$2\sum_{\substack{\forall A:\\ x_A < z}} x_A B = 2^{n-k} 2B \sum_{x_A < z} x_A = 2^{n-k} z(z-1)B \tag{17}$$

and

$$\sum_{\substack{\forall A:\\ x_A = z}} zA = z\sum_{\substack{\forall A:\\ x_A = z}} A = zS1(z). \tag{18}$$

By evaluating (17) for all *B*, we obtain:

$$\begin{aligned} 2\sum_{\substack{\forall A,B:\\ x_A < x_B}} x_A B &= \sum_{\forall B} 2^{n-k} z(z-1)B \\ &= 2^{n-k}\sum_{z=0}^{2^j-1} z(z-1)S1(z). \end{aligned} \tag{19}$$

By evaluating (18) for all *B*, we obtain:

$$\sum_{\substack{\forall A,B:\\ x_A = x_B}} x_B A = \sum_{\forall B} x_B S1(x_B) = 2^{n-k}\sum_{z=0}^{2^j-1} zS1(z). \tag{20}$$

Using (19) and (20), (15) is equal to:

$$MED = \frac{2^j 2^{n-k}}{2^{2n}}\big(\sum_{z=0}^{2^j-1} z^2 S1(z)\big) \tag{21}$$

$$\text{and } NMED = \frac{2^j 2^{n-k}}{2^{2n}(2^n-1)^2}\big(\sum_{z=0}^{2^j-1} z^2 S1(z)\big). \tag{22}$$

The sum of all *REDs* is given by:

$$\begin{aligned} \sum_{\forall A,B} RED(A,B) &= 2^j\Big(\sum_{\substack{\forall A,B:\\ x_A \geq x_B}} \frac{x_B}{B} + \sum_{\substack{\forall A,B:\\ x_A < x_B}} \frac{x_A}{A}\Big) \\ &= 2^j\Big(\sum_{\substack{\forall A,B:\\ x_A = x_B}} \frac{x_B}{B} + 2\sum_{\substack{\forall A,B:\\ x_A > x_B}} \frac{x_B}{B}\Big). \end{aligned} \tag{23}$$

Denoting $CI = 2^k - 1$ and using that

$$\sum_{\substack{\forall A,B:\\ x_A > x_B}} \frac{x_B}{B} = \sum_{\forall B}\sum_{\substack{\forall A:\\ x_A > x_B}} \frac{x_B}{B} = \sum_{\forall B}\Big(\frac{x_B}{B}2^{n-k}(CI - x_B)\Big) \tag{24}$$

and

$$\sum_{\substack{\forall A,B:\\ x_A = x_B}} \frac{x_B}{B} = \sum_{\forall B}\sum_{\substack{\forall A:\\ x_A = x_B}} \frac{x_B}{B} = \sum_{\forall B}\big(\frac{x_B}{B}2^{n-k}\big), \tag{25}$$

(23) is equal to:

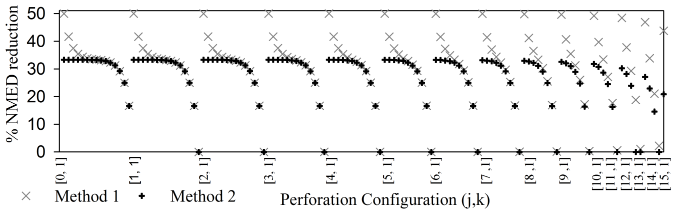
$$\begin{aligned} \sum_{\forall A,B} RED(A,B) &= 2^j 2^{n-k}\sum_{\forall B}\Big(\frac{x_B}{B}\Big(1 + 2(CI - x_B)\Big)\Big) \\ &= 2^j 2^{n-k}\sum_{B=1}^{2^n-1}\Big(\frac{x_B}{B}\Big(1 + 2(CI - x_B)\Big)\Big) \end{aligned} \tag{26}$$

and *MRED* is calculated as a relation of *j* and *k* from:

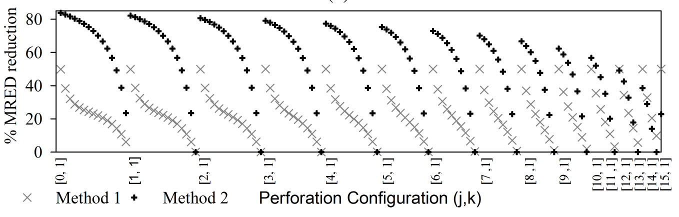
$$MRED = \frac{2^j}{2^{n+k}}\sum_{B=1}^{2^n-1}\Big(\frac{x_B}{B}\Big(1 + 2(CI - x_B)\Big)\Big). \tag{27}$$

- **Method 2**: Comparing *A,B*

In this method A, B are compared before the multiplication and, if $A > B$, A and B are swapped. As a result the induced



(a)



(b)

Fig. 3. The percentage reduction of a) *NMED* and b) *MRED* achieved by the correction Methods 1 and 2 with respect to the *NMED* and *MRED* values obtained by product perforation without correction. The x-axis contains all the [*j, k*] configurations.

error $ED(A,B) = A2^j x_B$, when $A \leq B$ and $ED(A,B) = B2^j x_A$, when $A > B$. Similarly to Method 1:

$$\begin{aligned} MED &= \frac{2^j}{2^{2n}}\Big(\sum_{\substack{\forall A,B:\\ A\leq B}} x_B A + \sum_{\substack{\forall A,B:\\ A>B}} x_A B\Big) \\ &= \frac{2^j}{2^{2n}}\Big(\sum_{\substack{\forall A,B:\\ A=B}} x_A A + 2\sum_{\forall A}\sum_{\substack{\forall B:\\ B<A}} x_A B\Big) = \frac{2^j}{2^{2n}}\sum_{A=1}^{2^n-1} x_A A^2, \end{aligned} \tag{28}$$

$$NMED = \frac{2^j \sum_{A=1}^{2^n-1} x_A A^2}{2^{2n}(2^n-1)^2}, \tag{29}$$

$$\text{and } MRED = \frac{2^j}{2^{2n}}\sum_{B=1}^{2^n-1}\Big(\frac{x_B}{B} + 2x_B\Big). \tag{30}$$

Fig. 3 depicts the error improvement achieved by Methods 1 and 2, for a 16-bit (*n*=16) multiplier and all the product perforation configurations (*j,k*). Fig. 3a presents the *NMED* reduction attained by the correction methods with respect to the *NMED* of product perforation without an error correction method. Fig. 3b illustrates the respective graph for the *MRED* metric. The proposed corrective methods offer both *NMED* and *MRED* reduction. Method 1 offers higher *NMED* reduction, while Method 2 achieves higher *MRED* reduction. On average, Method 1 offers 30% *NMED* and 24% *MRED* reduction, while Method 2 offers 26% and 50% reduction, respectively. As a result, the selection of a corrective method depends on the application in which the perforated multiplier will be used. If the magnitude of the error is more important than its absolute distance from the accurate result, then Method 2 should be preferred; if not, then Method 1 should be selected. However, the implementation of Method 1 requires a *k*-bit comparator, while Method 2 requires a *n*-bit one and thus, Method 1 induces smaller area and power overheads. As a result, since both methods offer significant *NMED* and *MRED* reductions and Method 1 induces less power overhead, it should be preferred in the case the application is unknown.

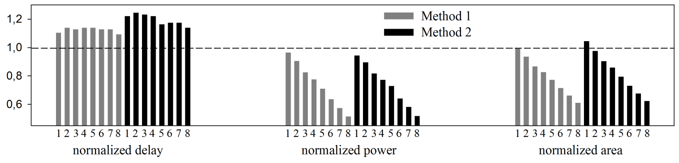


Fig. 4. The normalized delay, power and area metrics achieved by applying product perforation with correction and with j=1 and k=1..8 on a Dadda 4:2 multiplier, with respect to those of the accurate design.

Methods 1 and 2 decrease the error metrics, but their implementation requires an additional comparator. Fig. 4 presents the impact of correction Method 1 or 2 on the delay, power, and area on the Dadda 4:2 multiplier, in respect to the accurate design. Since the complexity of the comparator is mainly affected by the perforation variable *k*, Fig. 4 depicts perforation configurations that feature *j*=1 and *k*= 1 to 8 (similar results are obtained for other *j* and for MBE designs). As expected, using Method 1 with perforation induces 13% overhead on critical delay, but also retains 26% and 20%, on average, power and area saving. The respective values for Method 2 are 20%, 26%, and 17%.

The *NMED* and *MRED* analytical relations show that *the error imposed by the product perforation method is bounded and predictable*. Therefore, when the application's input dataset is determined, it can be used to calculate the optimal combination of *j* and *k* that produce an error less than a desired upper bound.

IV. Exploring the Efficiency of Partial Product Perforation

In this section, the partial product perforation method is applied to various multiplier architectures in order to explore how their power consumption, area, delay, and accuracy behave considering the perforation configuration variables *j* and *k*. This analysis targets to expose the optimal architecture–configuration pair for determined error values regarding both power dissipation and area complexity. This is critical, since different configurations may not have the same impact on a multiplier architecture, e.g. an architecture may be the power optimal one when accurate calculations are performed, but suboptimal when partial product perforation is applied.

Both SPP and MBE techniques are considered in our analysis. Regarding the accumulation tree, the most common architectures are used: 1) Array, 2) Balanced delay, 3) Compressor 4:2, 4) Counter 7:3, 5) Dadda, 6) Dadda with 4:2 compressors, 7) Redundant binary and 8) Wallace [19], [21], [22]. The Array is the simplest way to accumulate the partial products. It consists of successive Carry-Save Adders (CSA) and has the least complexity but the highest delay. The Wallace tree reduces to the least possible the number of partial products in each layer and is theoretically the fastest multi-operand adder. However, it has very complex interconnections that do not permit practical implementations. The Balanced delay tree provides a more regular routing and minimizes the number of wiring trucks. The Compressor 4:2 tree has also a regular structure and sums the partial products as a binary tree does, using 4:2 compressors instead of CSAs. Unlike the Wallace tree, Dadda makes the fewest reductions

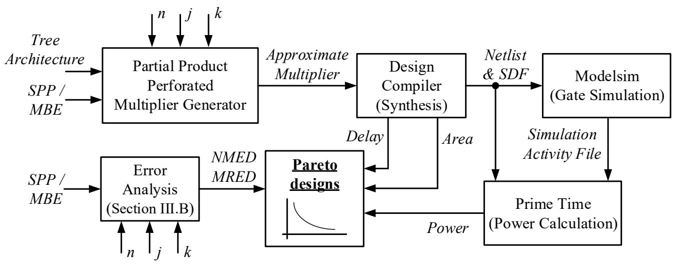

Tree Architecture
SPP / MBE
n j k
Partial Product Perforated Multiplier Generator
Approximate Multiplier
Design Compiler (Synthesis)
Netlist & SDF
Modelsim (Gate Simulation)
Delay
Area
Simulation Activity File
SPP / MBE
Error Analysis (Section III.B)
NMED MRED
Pareto designs
Prime Time (Power Calculation)
Power

Fig. 5. The flow used to evaluate the Partial Product Perforation method on different multiplier architectures.

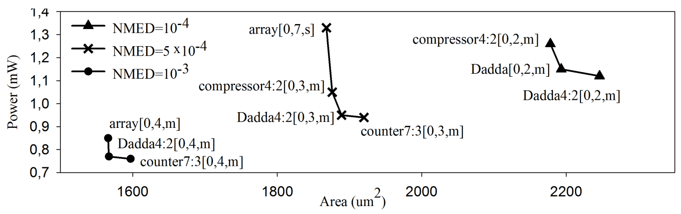


Fig. 6. The Power–Area Pareto curves for different *NMED* values[4].

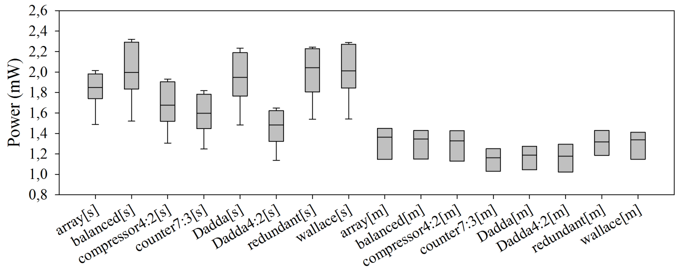


Fig. 7. Box plots of power consumption for $NMED < 5 \times 10^{-4}$.

needed in each layer and can achieve similar overall delay, but requires less gates. The Dadda tree is based on 3:2 counters (full adders) but also 2:2 counters (half adders) to reduce the hardware complexity. The Dadda 4:2 and Counter 7:3 trees use the same reduction strategy with the Dadda tree using though 4:2 and 7:3 compressors, respectively. In the Redundant binary tree, the partial products are in a redundant representation and the addition is performed by redundant binary adders [23] in the form of a binary tree. A Carry Look-Ahead adder is used as the final adder in all multipliers. Fig. 1 depicts some typical reduction schemes of the aforementioned tree architectures and the respective perforated trees with configuration *j*=*k*=2. Using the unit gate model[2] [24], the area of the Array is decreased by 112au and its delay by 8tu. The respective values for the Wallace tree are 115au and 4tu. The delay of the Dadda 4:2 and Compressor 4:2 is not decreased but their area decrease is 127au and 112au, respectively.

Exploration and analysis: The flow used for our evaluation is summarized in Fig. 5. For our analysis, 16-bit unsigned[3] multiplier architectures are considered. They are implemented in structural Verilog and synthesized using Synopsys Design Compiler and the TSMC 65nm standard cell library. We simulate the designs using Modelsim and calculate their power consumption with Synopsys PrimeTime triggering the average mode of calculation. All the possible combinations of *j* and *k* are explored and 1376 architectural configurations are examined in total. The metrics measured for each design are the *NMED*, *MRED*, minimum delay and, at the relaxed period of 2ns, its power consumption and area complexity. In [25], a detailed power, area and delay characterization and analysis of the examined perforated multiplier architectures has been performed showing that the aforementioned metrics are scaling gracefully, i.e. average slope -0.16%, -242% and -0.03% respectively, for increased values of *k*.

Since power, area, and delay metrics scale differently for each multiplier architecture when different error values are considered, we illustrate in Fig. 6 the power–area Pareto curves for different *NMED* values in order to distinguish the optimal designs. We consider the *NMED* values of $10^{-4}, 5\times 10^{-4}$ and 10^{-3} which enclose a large set of different partial product perforation configurations while keeping the error small. The optimal accurate design is the Dadda[m]. Moreover, the Dadda4:2[m][5] architecture appears in all curves but with different product perforation configuration (i.e., different *j* and *k* values), depending on the *NMED* bound. In respect to the accurate design, perforation achieves up to 50% power, 45% area and 35% delay reductions for only 0.1% error (i.e., $NMED < 10^{-3}$).

Aiming to elucidate the impact of partial product perforation on each multiplier architecture, we examine their power variation (i.e., the range of power values) for a bounded error. Fig. 7 presents the box plot diagram for all the architectures with regard to power, considering all the product perforation configurations that result to $NMED < 5{\times}10^{-4}$. The MBE-based architectures exhibit smaller variation and lower median than the respective SPP-based ones. The lowest median and variation values are observed for the counter7:3[m] architecture. Thus, its power consumption for various perforation configurations is concentrated in a smaller range, making its power behavior more predictable. The same conclusion is confirmed in Fig. 6 where the counter7:3[m] for *NMED* values $5{\times}10^{-4}$ and 10^{-3} is the Pareto optimal point with the lowest power.

V. Experimental Evaluation

A. Comparative Study on Circuit Level

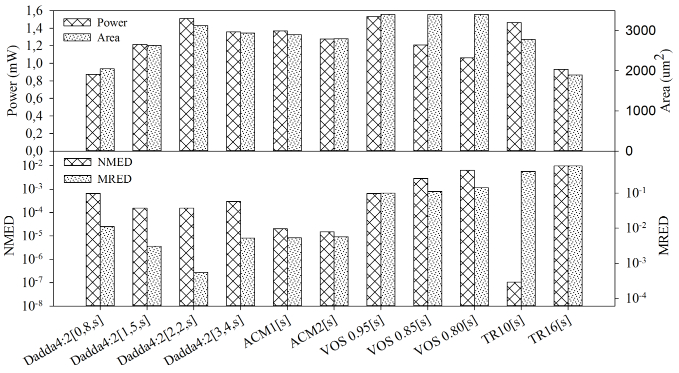
In this section, we extensively evaluate the efficiency of partial product perforation in terms of power, area, and error, and we compare it with state-of-the-art approximation techniques, which apply either truncation [4], logic approximation [9] or the VOS technique [6]. Using the two inexact 4:2 compressors of [9] at the 16 LSB columns, two approximate 16-bit multipliers ACM1 and ACM2 are implemented in structural Verilog

[2]Area/delay of a full adder is 7 area units (au)/ 4 time units (tu), of a half adder 3au/2tu and of a 4:2 compressor 14au/6tu

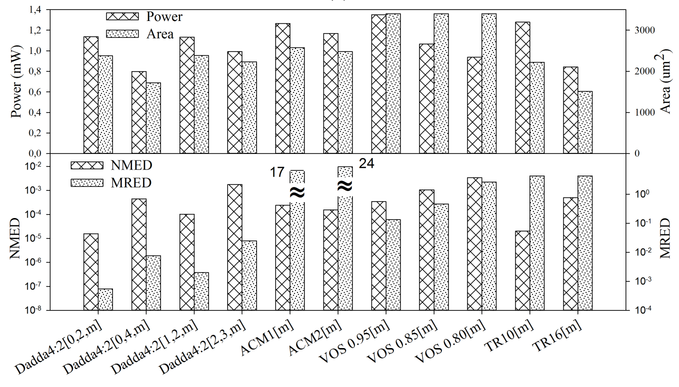
[3]Applying product perforation to signed multiplication is performed similar to the unsigned one, except that we do not perforate the last partial product. Therefore, no extra circuit is needed and similar results are expected.

[4]The respective *MRED* values of the designs can be derived in a straightforward manner from the error equations presented in Section III-B utilizing the annotated *j* and *k* parameter values.

[5]In the remainder, we consider as driver circuit the Dadda 4:2.



(a)

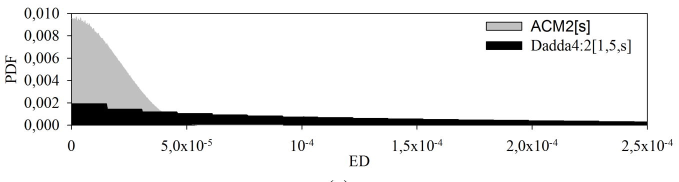


(b)

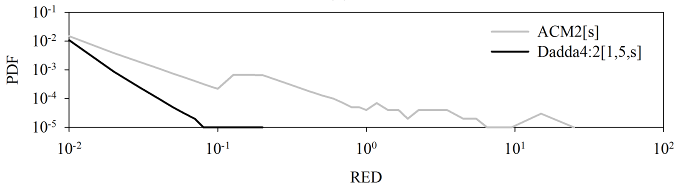
Fig. 8. Comparison of partial product perforation with ACM1, ACM2 [9], TR10, TR16, and VOS for (a) SPP and (b) MBE architectures.

and synthesized at 2ns using Synopsys Design Complier and PrimeTime. Error metrics calculation is performed through exhaustive Matlab simulation. In order to compare the partial product perforation with the VOS technique, we use the Synopsys Composite Current Source model (CCS) [26]. CCS models are proven to deliver signoff-level accuracy to within 2% of HSPICE simulation, are designed to be scalable for voltage, temperature and process, and offer better accuracy than the Non-Linear Delay and Power Models [26]. For the exact multiplier architectures of Section IV, we scale the supply voltage from 1V (nominal) to 0.80V and measure their power consumption and error metrics using 10^5 randomly generated inputs. Regarding truncation, two truncated multipliers with variable correction [4] that use the Dadda 4:2 tree to accumulate the partial products are implemented. In the first one (TR10) the 10 LSBs are truncated while in the second (TR16) the 16 ones. For the perforated multipliers, the error correction Method 1 (Section III-B2) is used.

Fig. 8 presents comparative results on the power, area, *NMED*, and *MRED* metrics after applying: i) the four different partial product perforation configurations, ii) the approximate compressors according to the technique presented in [9] (ACM1 and ACM2), iii) the VOS technique and iv) the truncation (TR10 and TR16) on a 16-bit Dadda 4:2 multiplier using SPP (Fig. 8a) and MBE (Fig. 8b). The examined perforated designs exhibit different order of perforation (*j* variable) and they are on (designs Dadda4:2[0,8,s] and Dadda4:2[1,5,s]) or close to (designs Dadda4:2[2,2,s] and Dadda4:2[3,4,s]) the power-*NMED* Pareto optimal curve of the Dadda 4:2



(a)



(b)

Fig. 9. a)The Probability Density Function of the *ED* for the ACM2 [9] and the partial product perforation Dadda 4:2 multiplier with *j* = 1 and *k* = 5. *ED* is in the Q0.32 number format (fixed point representation of 32-bit integers in the range [0:1)). b) The respective Probability Density Function of the *RED*.

architecture. Similar selection has been performed for the MBE-based designs.

The proposed Partial Product Perforation for the SPP-based designs, included in Fig. 8a, delivers power savings of up to 49% and area reduction of up to 40% compared to the respective accurate design, while the *NMED* value is 6.5×10^{-4} at most and the *MRED* one goes up to 1.1×10^{-2}. The respective values for MBE-based configurations (Fig. 8b) are 47% power savings, 38% area reduction, *NMED* 1.8×10^{-3}, and *MRED* 2.5×10^{-2}. **The approximate compressors multipliers** ACM1, ACM2 [9] with SPP (Fig. 8a) have 15%, 20% power and 15%, 18% area savings, respectively, over the accurate Dadda 4:2 multiplier. Their *NMED* values are 2×10^{-5} and 1.5×10^{-5}, while their *MRED* ones are 5.3×10^{-3} and 5.6×10^{-3}, respectively. For the MBE (Fig. 8b), ACM1, ACM2 have 16%, 23% power savings and 8%, 11% area reduction, respectively, over the accurate Dadda 4:2 multiplier. Their *NMED* values are 2.4×10^{-4} and 1.6×10^{-4} while their *MRED* ones are 17 and 24 respectively. Regarding the MBE-based designs, [9] is less efficient since less partial products compared to the SPP technique are accumulated in the tree and and an error occurring in one column has a greater impact on the output. **VOS** does not deliver any area reduction, offering though significant power savings compared to the accurate design. When decreasing the supply voltage of the SPP-based design to 0.80V (Fig. 8a), the power consumption is 1.06mW (i.e., 37.9% less than the accurate one). Similarly, the power consumption of the MBE-based design (Fig. 8b) is 0.94mW (i.e., 37.7% less than the precise design). However, even for small power savings (10% at 0.95V), the *NMED* and *MRED* values of VOS are too large, more than 0.65 and 10 respectively, as VOS errors are mainly impacting MSBs, resulting to large *ED*. **The truncated multipliers** TR10 and TR16 [4], when SPP is used, offer 14%, 46% power savings and 18%, 44% area reduction for 1.1×10^{-7}, 1.2×10^{-1} *NMED* and 0.4, 0.8 *MRED*, respectively. The respective values for the MBE-based designs are 15%, 44% power savings, 20%, 46% area reduction, 2×10^{-5}, 5.0×10^{-4} *NMED* and 4.2, 4.3 *MRED*.

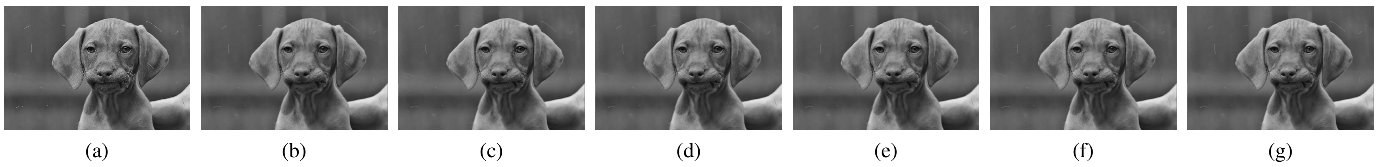


Fig. 10. The a) 16-bit input image and the result of the geometric mean filter using the b) accurate multiplier Dadda4:2[s], c) Dadda4:2[1,5,s] w/o correction, d) Dadda4:2[1,5,s] w/ correction Method 1, e) Dadda4:2[3,4,s] w/o correction, f) Dadda4:2[3,4,s] w/ correction Method 1 and g) ACM2.

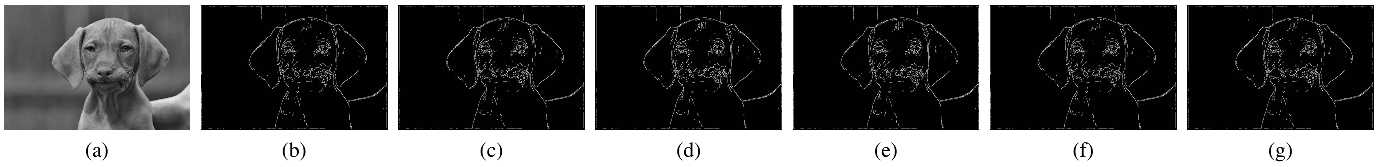


Fig. 11. The a) 16-bit input image and the result of the Canny edge detection using the b) accurate multiplier Dadda4:2[s], c) Dadda4:2[1,5,s] w/o correction, d) Dadda4:2[1,5,s] w/ correction Method 1, e) Dadda4:2[3,4,s] w/o correction, f) Dadda4:2[3,4,s] w/ correction Method 1 and g) ACM2.

On average, the partial product perforation configurations, illustrated in Fig. 8, exhibit lower *MRED* values than ACM2, but higher *NMED*. The large *NMED* value of partial product perforation implies that it may produce large *ED*. However, the small value of *MRED* shows that such large *ED* is insignificant compared to the accurate result. The aforementioned points can be further explained based on the error analysis of Section III-B. As shown, the *ED* is proportional to the inputs and, thus, it can be as large as the input numbers. However, $RED = x_B 2^j / B$ and since few partial products are removed, the nominator is much smaller than *B*, resulting to small relative error values. On the other hand, [9] produces smaller *ED*, but its errors are of greater significance compared to the exact results. This behavior is also captured by Fig. 9 where the Probability Density Function (PDF) of the *ED* and *RED* for ACM2 and Dadda4:2[1,5,s] is presented. ACM2 exhibits lower *NMED* but higher *MRED* compared with Dadda4:2[1,5,s]. Fig. 9a depicts the PDF of the *ED* for the aforementioned multipliers. ACM2 has significantly greater error probability, but its probable error values are concentrated in a smaller range. In contrast, the Dadda4:2[1,5,s] errors are spread to a wider range and have almost equal, but very low, probability to appear. Fig. 9b depicts the same graph for the *RED* metric. As presented in Fig. 9b, ACM2[s] produces larger *RED* values than Dadda4:2[1,5,s] and with greater probability.

To summarize, the partial product perforation technique shows significant gains compared to the accurate design and state-of-the-art approximate techniques. On average, compared to VOS, partial product perforation configurations attain 3% lower power consumption and 96% lower *MRED*, when SPP is used, and 9% and 99%, respectively, when MBE is used. Compared to [9] for SPP schemes, their power consumption and their *MRED* are 6% and 9% lower, respectively. For MBE schemes, the respective values are 17% lower power and 3 orders of magnitude lower *MRED*. Compared with the SPP truncation [4], the perforated multipliers of Fig. 8 deliver on average 3% higher power for 99% lower *MRED*, while for MBE the respective values are 4% lower power and 2 orders of magnitude lower *MRED*. Finally, Table I offers a more straightforward comparison among the examined approximation schemes, by ranking them according to their savings and error metrics. The examined designs have been grouped in four sub-groups each one with designs exposing similar power and/or error characteristics. In each sub-group, the perforated multipliers deliver the lowest power and *MRED* values and, in most cases, the lowest *NMED* and area as well.

TABLE I
RANKING OF THE SAVINGS AND ERRORS OF THE APPROXIMATE MULTIPLIERS

Design		Power Gain		Area Gain		NMED		MRED	
SPP	MBE	SPP	MBE	SPP	MBE	SPP	MBE	SPP	MBE
[0,8,s]	[0,4,m]	1	1	2	2	8	7	6	3
TR16		2	2	1	1	11	8	11	9
VOS 0.80		3	3	11	11	10	11	9	7
VOS 0.85		4	5	11	11	9	9	8	6
[1,5,s]	[1,2,m]	5	6	3	6	5	3	2	2
ACM2		6	8	5	7	2	4	5	11
[3,4,s]	[0,2,m]	7	7	7	5	6	1	3	1
ACM1		8	9	6	8	3	5	4	10
TR10		9	10	4	3	1	2	10	8
[2,2,s]	[2,3,m]	10	4	8	4	4	10	1	4
VOS 0.95		11	11	11	11	7	6	7	5

B. Comparative Study on Real Life Applications

In this section, we evaluate the efficiency of the proposed technique on real-life use cases from the image processing and data analytics domains. For our analysis, we consider the Canny edge detection [27] and Geometric Mean filters from the image processing domain and the K-means clustering [28] from the data analytics domain, respectively. All the examined algorithms are implemented in C++, while for the image processing ones, OpenCV library is used.

Geometric mean filter removes noise from images, offering better results than the arithmetic mean filter for Gaussian type noise. The geometric mean filter with parameter *r* filters an image by replacing each pixel's value by the geometric mean of the values of all the neighboring pixels that are inside a $(2r + 1) \times (2r + 1)$ block centred on that pixel. For our evaluation, the *r* parameter is set to 3. We approximate the Geometric mean by replacing the multiplication between the pixels with an approximate 16×16 multiplier. We used as input the 16 bits (16 bits/pixel) grayscale image depicted in Fig. 10a. To evaluate the accuracy of the output images of the Geometric mean we use the Peak Signal Noise Ratio (*PSNR*).

Canny edge detection [27] filter, is considered to be an optimal edge detector. Specifically, i) it masks the image by applying a Gaussian filter to remove the noise, ii) it calculates the gradient of the image to find the edge strength, iii) it applies a non-maximum suppression to keep only the local maxima, iv) it determines the potential edges by thresholding, v) and, finally, it tracks edges by hysteresis, i.e, suppresses all the edges that are weak and not connected to strong edges. The size of the Gaussian kernel is 7×7 with 1.1 standard deviation value and uses 16-bit fixed point arithmetic. We approximate Canny edge by replacing the multiplication in the Gaussian filter with an approximate 16×16 multiplier. We used as input the 16 bits grayscale image depicted in Fig. 11a. The percentage of the edges detected using the approximate multiplier over those detected using the accurate one is used as our quality metric.

K-means is a popular algorithm for clustering data points from a multi-dimensional space into k clusters. It uses a two phase iterative method and aims to partition the data points into sets, so as to minimize the within-cluster sum of distance functions of each point in the cluster to the center. We use the Euclidean distance as a distance function. We approximate the K-means algorithm by replacing the multiplications in the calculation of the Euclidean distance with an approximate 16×16 multiplier. We use a random generated input dataset of 100,000 4-dimensional points with 16 bits per dimension. The input dataset is clustered in 100 clusters. To evaluate the accuracy of the K-means algorithm we use the average relative L2-Norm, i.e., $\left\langle \frac{|\mathbf{x_{acc}} - \mathbf{x_{approx}}|_2}{|\mathbf{x_{acc}}|_2} \right\rangle$.

Similar to [9], [10], the approximate multiplier is considered as part of a general processing system that implements the aforementioned algorithms. The rest of hardware components (except the multiplier) are considered to deliver accurate results and thus, any applications inaccuracy and energy savings result from the usage of the approximate multiplier. The energy values of each multiplication operation are delivered by post-synthesis simulations of the approximate multipliers on the input data traces extracted by the applications execution. Note that in the Canny edge detection and Geometric mean algorithms the number of the multiplications depends only on the image size and thus, it is the same for the accurate as well as the approximate version of the algorithm. On the other hand, the iterations performed by the K-means algorithm are not constant and as a result, the number of multiplications in the accurate may differ from the ones in the approximate version.

Fig. 10 depicts both the input image and the output image of the geometric mean filter when using the accurate multiplier Dadda4:2[s], the perforated multipliers Dadda4:2[1,5,s] and Dadda4:2[3,4,s] with and without any correction method, and the approximate multiplier ACM2. Fig. 11 shows the same images for the Canny edge detection. Table II summarizes the values of the energy savings and quality metrics of each application when using the aforementioned multipliers.

The use of the Dadda4:2[1,5,s] multiplier results in 85.95 dB PSNR for the geometric mean and 91.04% edges detected for the Canny edge detection. The application of the corrective Method 1 with the Dadda4:2[1,5,s] results in a small decrease of the energy savings (7.41%), but delivers better outputs as the PSNR increases by 2.9% and the edges detected by 7.6%. The Dadda4:2[3,4,s] multiplier detects the 84.79% of the edges and its PSNR is 89.93 dB. The use of correction Method 1 with the Dadda4:2[3,4,s] decreases the energy reduction by 10%, detects 16.6% more edges, and increases its PSNR by 3.1%. When ACM2[s] [9] is used, the output image has 86 dB PSNR and 97.85% edges detected. When we compare Dadda4:2[1,5,s] with ACM2, we observe that the former offers 25.6% higher energy reduction, detects 7% less edges, and has the same PSNR as the latter. When we compare ACM2[s] with Dadda4:2[3,4,s] using Method 1, we find that the latter delivers 18.6% lower energy savings, detects 1.8% more edges, and has 7.8% higher PSNR. Finally, when we compare Dadda4:2[1,5,s] using Method 1 with ACM2[s], the former achieves 16.3% higher energy reduction, detects 0.5% more edges, and has 2.8% higher PSNR. Regarding to the K-means algorithm, using a correction Method with product perforation does not deliver any quality improvement. This is explained by the fact that in the Euclidean distance the multiplier is used as a squarer and as a result swapping the multiplicands does not decrease the multiplication's error. Moreover, we observe that using ACM2[s] in the K-means algorithm does not offer any energy reduction. The implementation of the K-means algorithm with ACM2[s] fails to converge and exits after reaching a maximum number of allowed iterations. As a result, although ACM2[s] has lower power consumption compared with the accurate multiplier, the increased number of multiplications results in an energy increase of the K-means algorithm.

C. Impact of Bit-width Scaling

In this section, we examine the scalability of the proposed technique in terms of increased multiplier's bit-width. More specifically, we study the impact of scaled bit-widths, i.e. 16- up to 128-bits, on the proposed perforation technique focusing on the delivered accuracy (*NMED, MRED*) and power and area gains. We consider the Dadda 4:2 as our driver architecture solution and $NMED \leq 10^{-4}$ as our quality constraint. Fig. 12a depicts for each of the examined bit-widths the power and area reduction delivered by the perforated Dadda 4:2 solutions in respect to their accurate designs. In a complementary manner and for the same scaled bit-widths, Fig. 12b presents the *NMED* and *MRED* values when targeting 50% power reduction. Specifically, for $NMED \leq 10^{-4}$, the power and area gains for 16-bit width is 21% and 31%, respectively. The respective gains in the case of 128-bit width design scales up to 74% and 91% regarding to power and area, respectively. Similarly, Fig. 12b shows that for the same relative power gain, i.e. 50%, the 16-bit solution delivers an *NMED* and *MRED* value of 1.95×10^{-3} and 2.61×10^{-2}, respectively. For the 128-bit solution, *NMED* and *MRED* reduce to 1.73×10^{-18} and 2.05×10^{-16}, respectively. Thus, partial product perforation

TABLE II
EVALUATION OF PARTIAL PRODUCT PERFORATION IN IMAGE PROCESSING AND DATA ANALYTICS ALGORITHMS

Multiplier	**Canny Edge**		**Geometric Mean**		**K-Means**	
	PSNR (dB)	**Energy Gain (mJ)**	**Edges Detected (%)**	**Energy Gain (mJ)**	**avg. Relative L2-Norm (%)**	**Energy Gain (mJ)**
Accurate Dadda4:2[s][a]	Inf.	0	100.00	0	0	0
Dadda4:2[1,5,s]	85.95	1.18×10^{-3}	91.04	1.94×10^{-2}	5.08	18.94
Dadda4:2[1,5,s] - Meth. 1	88.45	1.09×10^{-3}	98.33	1.79×10^{-2}	5.08	18.04
Dadda4:2[3,4,s]	89.93	8.51×10^{-4}	84.79	1.40×10^{-2}	7.18	9.13
Dadda4:2[3,4,s] - Meth. 1	92.75	7.63×10^{-4}	99.58	1.25×10^{-2}	7.18	8.04
ACM2[s]	86.00	9.38×10^{-4}	97.85	1.54×10^{-2}	8.97	-6.06

[a]The energy required for the accurate multiplication process in the Canny Edge, the Geometric mean, and the K-means algorithm is 3.73×10^{-3}mJ, 6.13×10^{-2}mJ, and 45.14mJ, respectively.

offers better results as the multiplier's bit-width increases, i.e., higher power and area reduction for the same error constraints or lower error values for the same power savings.

This good scaling behavior for increased multiplier's bit-widths can be also theoretically confirmed utilizing the error analysis of Section III-B. Let us assume two multipliers M_1, M_2 with different bit-widths $n_1,\ n_2$ with $n_1 < n_2$ having the same *j* value for the partial product perforation. For both multipliers to achieve the same *NMED* the following relation should hold, according to Eq. (11):

$$\frac{2^j(2^{k_1}-1)}{4(2^{n_1}-1)} = \frac{2^j(2^{k_2}-1)}{4(2^{n_2}-1)} \implies \frac{(2^{n_2}-1)}{(2^{n_1}-1)} = \frac{(2^{k_2}-1)}{(2^{k_1}-1)}. \quad (31)$$

Given that $n_1 < n_2 \implies k_1 < k_2$. High k values imply the perforation of more partial products. Thus, for two approximate multipliers with the same *NMED* but different bit-widths, the higher the multiplier's bit-width the higher the the number of partial products that should be perforated, and thus the higher the power gains achieved in respect to their accurate counterparts.

VI. CONCLUSION

In this paper, we proposed the partial product perforation technique for producing approximate hardware multipliers. The proposed technique omits a number of partial products enabling high area and power savings, while retaining high accuracy. Through a rigorous error analysis, we analytically characterised the induced error metrics proving that the error is bounded and predictable and we proposed two error correction methods that trade a small increase in power for high error reduction. We explored product perforation on a large set of multiplier architectures, evaluating its impact on different architectures and error bounds. In comparison to state-of-the-art approximation techniques, we showed that the proposed approach achieves significant gains in power, area, and quality metrics of image processing and data analytics algorithms. Finally, we showed that our technique is scalable, offering better results as the multiplier's bit-width increases.

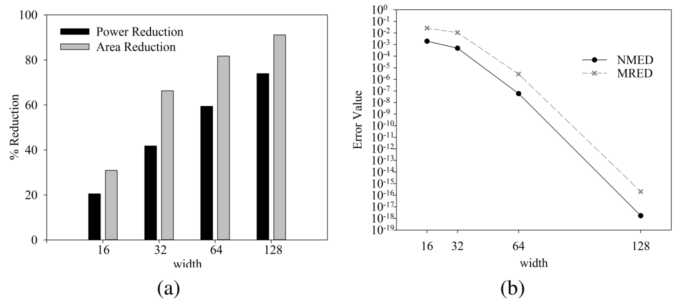


Fig. 12. Impact of multiplier's bit-width scaling on partial product perforation. a) Power and area gains for $NMED \leq 10^{-4}$, and b) *NMED* and *MRED* values when targeting 50% power savings.

REFERENCES

[1] V. K. Chippa *et al.*, "Analysis and characterization of inherent application resilience for approximate computing," in *Design Automation Conference*, May 2013, pp. 1–9.

[2] R. Venkatesan *et al.*, "Macaco: Modeling and analysis of circuits for approximate computing," in *Int. Conf. on Computer-Aided Design*, Nov. 2011, pp. 667–673.

[3] S. T. Chakradhar and A. Raghunathan, "Best-effort computing: Re-thinking parallel software and hardware," in *Design Automation Conference*, Jun. 2010, pp. 865–870.

[4] E. King and E. Swartzlander, "Data dependent truncated scheme for parallel multiplication," in *Proc. of the Thirty First Asilomar Conference on Signals, Circuits and Systems*, 1998, pp. 1178–1182.

[5] M. Schulte and E. Swartzlander, "Truncated multiplication with correction constant," in *VLSI Signal Processing VI*, Oct 1993, pp. 388–396.

[6] Y. Liu *et al.*, "Computation error analysis in digital signal processing systems with overscaled supply voltage," *IEEE Trans. VLSI Syst.*, vol. 18, no. 4, pp. 517–526, Apr. 2010.

[7] V. Gupta *et al.*, "Impact: Imprecise adders for low-power approximate computing," in *Int. Symp. on Low Power Electronics and Design*, Aug. 2011, pp. 409–414.

[8] P. Kulkarni *et al.*, "Trading accuracy for power with an underdesigned multiplier architecture," in *24th Int. Conf. on VLSI Design*, Jan. 2011, pp. 346–351.

[9] A. Momeni *et al.*, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984–994, Apr. 2015.

[10] N. Zhu *et al.*, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Trans. VLSI Syst.*, vol. 18, no. 8, pp. 1225–1229, Aug. 2010.

[11] A. K. Verma *et al.*, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Design, Automation and Test in Europe*, Mar. 2008, pp. 1250–1255.

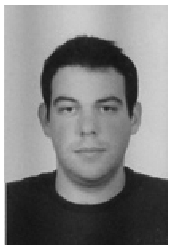
[12] Banescu *et al.*, "Multipliers for floating-point double precision and beyond on fpgas," *SIGARCH Comput. Archit. News*, vol. 38, no. 4, pp. 73–79, Jan. 2011.

[13] C. Liu *et al.*, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Design, Automation and Test in Europe*, Mar. 2014.

[14] S. Sidiroglou *et al.*, "Managing performance vs. accuracy trade-offs with loop perforation," in *Foundations of software engineering (ESEC/FSE)*, Sep. 2011, pp. 124–134.

[15] J. Liang *et al.*, “New metrics for the reliability of approximate and probabilistic adders,” *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1760–1771, Jun. 2012.
[16] Lingamneni *et al.*, “Synthesizing parsimonious inexact circuits through probabilistic design techniques,” *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 93:1–93:26, May 2013.
[17] S. Narayanamoorthy *et al.*, “Energy-efficient approximate multiplication for digital signal processing and classification applications,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 23, no. 6, pp. 1180–1184, June 2015.
[18] S. Hashemi *et al.*, “DRUM: A dynamic range unbiased multiplier for approximate applications,” in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 2015*, November 2015, pp. 418–425.
[19] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. NY: Oxford University Press, 2000.
[20] C. Lee *et al.*, “Mediabench: a tool for evaluating and synthesizing multimedia and communications systems,” in *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, Dec 1997, pp. 330–335.
[21] D. Zuras and W. McAllister, “Balanced delay trees and combinatorial division in vlsi,” *Solid-State Circuits, IEEE Journal of*, vol. 21, no. 5, pp. 814–819, Oct 1986.
[22] L. Dadda, “Some schemes for parallel multipliers,” *Alta Frequenza*, vol. 34, no. 5, pp. 349–356, Mar 1965.
[23] B. Jose and D. Radhakrishnan, “Delay optimized redundant binary adders,” in *Electronics, Circuits and Systems, 2006. ICECS ’06. 13th IEEE International Conference on*, Dec 2006, pp. 514–517.
[24] N. Weste and D. M. Harris, *Datapath Subsystems, CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 2010.
[25] G. Zervakis *et al.*, “Approximate multiplier architectures through partial product perforation: Power-area tradeoffs analysis,” in *Proc. of the 25th Great Lakes Symposium on VLSI*, 2015, pp. 229–232.
[26] G. Mekhtarian, *Composite Current Source (CCS) Modeling Technology Backgrounder*. Synopsys, Inc., Nov. 2005.
[27] J. Canny, “A computational approach to edge detection,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, no. 6, pp. 679–698, Nov 1986.
[28] C. Ranger *et al.*, “Evaluating mapreduce for multi-core and multiprocessor systems,” in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, Feb 2007, pp. 13–24.

Georgios Zervakis received his Diploma at the Department of Electrical and Computer Engineering from the National Technical University of Athens, Greece in 2012. Since 2012 he is a Ph.D. student at the National Technical University of Athens in the field of digital and microprocessor system design. His research interests include approximate computing, VLSI arithmetic circuits, low power design and cryptography.



Kostas Tsoumanis received his Diploma at the Department of Electrical and Computer Engineering from the National Technical University of Athens in 2010. He is currently working on his Ph.D. thesis at the National Technical University in the Department of Electrical and Computer Engineering. His research interests include hardware-efficient implementation of arithmetic operations and low-power design of Digital Signal Processing algorithms. He is a co-author in research papers published in international conferences. He is an IEEE student member.

Sotirios Xydis received the Diploma and the Ph.D. degree in electrical and computer engineering from the National Technical University of Athens, Athens, Greece, in 2005 and 2011, respectively. He was for two years as Post-Doctoral Research Fellow position with Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy. Currently, he is a Research Associate at National Technical University of Athens. His research interests include design space exploration for system level and datapath synthesis, design and optimization of arithmetic VLSI circuits and power management multi-/many-core and reconfigurable architectures. He has published over 60 technical and research papers in scientific books, international journals and conferences. Dr. Xydis is the recipient of the two best paper awards from the NASA/ESA/ IEEE International Conference on Adaptive Hardware and Systems 2007 (AHS 2007) and from the 4th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures (PARMA 2013).

Dimitrios Soudris Dimitrios Soudris received his Diploma in Electrical Engineering from the University of Patras, Greece, in 1987. He received the Ph.D. Degree in Electrical Engineering, from the University of Patras in 1992. He was working as a Professor in Dept. of Electrical and Computer Engineering, Democritus University of Thrace for thirteen years since 1995. He is currently working as Associate Professor in School of Electrical and Computer Engineering, Dept. Computer Science of National Technical University of Athens, Greece. His research interests include embedded systems design, reconfigurable architectures, reliability and low power VLSI design. He has published more than 340 papers in international journals and conferences. Also, he is coauthor/coeditor in seven books of Kluwer and Springer. He is leader and principal investigator in numerous research projects funded from the Greek Government and Industry, European Commission (ESPRIT II-III-IV and 5th & 7th IST), ENIAC-JU and European Space Agency. He has served as General Chair and Program Chair for PATMOS 99 and 2000, respectively, General Chair of IFIP-VLSI-SOC 2008 and General Co-Chair of PARMA Workshop 2013. Also, he received an award from INTEL and IBM for the EU project LPGD 25256, awards in ASP-DAC 05 and VLSI 05 for EU AMDREL project IST-2001-34379.

Kiamal Pekmestzi received his Diploma in Electrical Engineering from the National Technical University of Athens, Athens, Greece, in 1975, and the Ph.D. degree in Electrical Engineering from the University of Patras, Patras, Greece, in 1981. From 1975 to 1981 he was a Research Fellow with the Electronics Department, Nuclear Research Center ”Demokritos”. From 1983 to 1985, he was a Professor with the Higher School of Electronics, Athens, Greece. Since 1985, he has been with the National Technical University of Athens, where he is currently a Professor with the Department of Electrical and Computer Engineering. His research interests include efficient implementation of arithmetic operations, design of embedded and microprocessor-based systems, architectures for reconfigurable computing, VLSI implementation of cryptography, and digital signal processing algorithms.