IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 25, 2021, accepted January 15, 2022, date of publication January 20, 2022, date of current version February 1, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3145001

A Practical Energy/Power Reduction Approach for Parallel Decimal Multiplier

SAEID GORGIN[1,2], HELIA ZAREIE NEJAD[3], AND JEONG-A. LEE[1], (Senior Member, IEEE)

[1]Department of Computer Engineering, Chosun University, Gwangju 61452, South Korea
[2]Department of Electrical Engineering and Information Technology, Iranian Research Organization for Science and Technology, Tehran 33531-36846, Iran
[3]Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin 34185-1416, Iran

Corresponding authors: Saeid Gorgin (gorgin@chosun.ac.kr) and Jeong-A. Lee (jalee@chosun.ac.kr)

This work was supported in part by the Brain Pool Program funded by the Ministry of Science and ICT through the National Research Foundation of Korea under Grant NRF-2021H1D3A2A02040040, and in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2020R1I1A3063857.

ABSTRACT Decimal computation is highly demanded in many human-centric applications such as banking, accounting, tax calculation, and currency conversion. Hence the design and implementation of radix-10 arithmetic units attract the attention of many researchers. Among the basic decimal arithmetic operations, multiplication is not only a frequent operation but also has high complexity and considerable power consumption. Therefore, this paper concentrates on this issue and studies a general design methodology that reduces power/energy consumption via localizing switching activity without compromising target performance. This method decomposes a digit multiplier into smaller ones, like Karatsuba's algorithm, while the multiplicand and the multiplier can be partitioned into different sizes. We take advantage of various size partitions in two types of symmetric and asymmetric, which final designs provide specific characteristics. All designs were implemented using VHDL and synthesized in the Design-Compiler toolbox with TSMC 130 nm Technology file. The results are significant; in respect to the original design, by a random test vector, 25% power reduction is achieved with no effect on latency and a negligible area penalty. Moreover, the experimental results indicate a potential for considerable reduction of power dissipation based on statistical properties of possible input data.

INDEX TERMS Decimal computation, digital arithmetic, low power design, parallel multiplier.

I. INTRODUCTION

Despite the fast and effective implementation of binary arithmetic functions, using decimal computation has been revived. This revitalization has been done for three main reasons; (1) the advances in VLSI technology, (2) the appearance of a large amount of decimal data in human-centric applications such as financial, commercial, scientific, and internet-based applications so that the software implementations do not satisfy the high-performance requirements [1], and finally (3) the lack of exact binary representation for some decimal fractions (e.g., 0.2). The former made hardware realization of complex functions possible, while the two others pushed the designers to use hardware-implemented decimal arithmetic units for coping with the complexity of processing a massive amount of data with acceptable precision and time.

Due to the importance of decimal arithmetic, the new feature of decimal representation and related operations is added to the latest revision of IEEE 754 standard for Floating-point arithmetic [2], [3]. Moreover, concerted activity in both industry and academia is progressing on decimal arithmetic. Several processors have been announced, such as IBM eServer z900 [4], IBM POWER6 [5], and IBM z10 [6], which are equipped with a dedicated decimal arithmetic unit. On the other hand, a considerable number of research papers have been published on decimal arithmetic algorithms and hardware units such as decimal addition [two-operand (e.g., [7]) and multi-operand (e.g., [8])], decimal multiplication [sequential (e.g., [9]) and parallel (e.g., [10])], decimal division [subtractive (e.g., [11]) and multiplicative (e.g., [12])], and other arithmetic functions (e.g., [13]). Among various operations, decimal multiplication is known as one of the most complex operations, which is high frequency, time-consuming, and power-hungry. In addition, it is iteratively used to implement other useful operations such as division, square root, and function evaluation circuits like radix-10 exponentiation and logarithm. Hence, high-speed decimal multiplication takes particular attention, and several

The associate editor coordinating the review of this manuscript and approving it for publication was Zhigao Zheng.

publications have been published on this topic in less than ten years. The proposed methods of these articles are focused on latency and area as the main design parameters, while the power/energy consumption is neglected. Whereas, the cooling issue in high-performance processing systems and the limited power budget in embedded systems, as well as the effects of consumed power in the efficiency and reliability of digital circuits, make the power/energy consumption the most challenging parameter for nowadays hardware designers [14]. In [15], we provide a comparative study on the leakage and dynamic power consumption of released high-speed decimal multipliers and suggest some guidelines for EDA tools and hardware designers.

In this paper, we target to reduce power/energy consumption in high-speed decimal multipliers (i.e., parallel decimal multipliers). To achieve the best result, we use the highest level of design abstraction for applying power reduction techniques, since power optimizations at the higher levels of design abstraction have the maximum influence. Our suggested method is based on partitioning (like Karatsuba's algorithm) and reduces power/energy consumption by localizing *switching activity* without any negative impact on performance. In our proposed method, we implement a digit multiplier (i.e., IEEE 754-2008 standard) via smaller ones in two different types of symmetric and asymmetric. In symmetric designs, all smaller multiplier units have the same size, while different sizes of multipliers may be used in asymmetric ones.

The rest of this paper is organized as follows. In Section 2, as a background, we briefly describe the structure of parallel decimal multiplier and provide a theoretical foundation of power consumption, which paves the way for the following sections' discussions. The proposed method based on two types of symmetric and asymmetric partitioning is presented in Section 3. The essential criteria for selecting primary multiplier units (i.e., the small parallel decimal multipliers) and chosen decimal multiplication algorithm are discussed in Section 4. In Section 5, we explain the design flow of the synthesis tool to generate the reports of power consumption and experimental results of various implementations, where power comparison with original design is also provided. Finally, we conclude the paper in Section 6.

II. BACKGROUND

In this section, the theoretical background in two separate parts is concisely discussed. At first, the general structure of parallel decimal multipliers and the previous related works are explained. Then, in the second part, the requirement foundations about power/energy consumption are described. This information paves the way for a detailed discussion of the following sections.

A. PARALLEL DECIMAL MULTIPLIERS STRUCTURE

Decimal multipliers, like their binary counterparts, have three main steps, which are called partial product generation (PPG), partial product reduction (PPR), and the final addition (or redundant to not-redundant conversion). However, decimal multiplication is more complicated than binary multiplication in the entire aforementioned steps. The PPG in binary multiplication can be done by a simple AND-gate matrix. However, due to the wider range of decimal digits, in decimal multiplication, various techniques like lookup tables, decimal digit-multipliers, or pre-computed multiples must be used to provide various multiples of the multiplicand. Moreover, due to binary logic and BCD encoding of decimal numbers in the decimal implementations, for compensating the difference carry value in decimal and binary, all the decimal add operations in the PPR and final addition needs a correction step.

Furthermore, decimal multipliers like binary ones can be designed in a variety of ways, namely sequential [9], [16], [17], parallel [10], [18]–[20], and array [21], which offer area and speed trade-offs. In the former, the partial products are iteratively generated one after another based on multiplier digits. Then, each generated partial product is aligned and added to previous accumulated partial products. Via this technique, the product may be delivered in the minimum area; however, it is time-consuming and cannot satisfy the quest for high-performance processing. On the contrary, the parallel multipliers present low latency; however, not surprisingly, the area and power/energy consumption of these units are rather significant. Thanks to the astonishing advances in VLSI technology, the huge area challenge is alleviated. Nevertheless, power/energy consumption remains the most challenging issue for hardware designers [14].

In a parallel decimal multiplier, all partial products are generated simultaneously in the PPG step. As mentioned above, various methods exist for generating each partial product; using pre-computed multiples is dominant. In the naïve implementation, all the possible multiples of multiplicand X(i.e., $\{0X, 1X, \ldots, 9X)\}$ are needed. However, in practice, just a limited subset of multiples of the multiplicand, called primary multiples, are generated (e.g., $\{1X, 2X, 4X, 5X\}$ or $\{\pm 1X, \pm 2X, 5X, 10X\}$). The primary multiples can be generated in constant time. The other multiples are computed by using the primary multiples (e.g., $9X = 4X + 5X$ or $9X = 10X - 1X$). For constructing the PPG matrix (i.e., aligned all the partial products), the multiplier digits are recoded and these recoded values are used for selecting proper multiples. This process is shown in the upper part of Fig. 1.

The second step of parallel multiplication, partial product reduction (PPR), can be considered as a decimal multi-operand addition. In PPR, via a reduction tree, the final product in a redundant representation is computed. Finally, in the last step, a redundant to non-redundant conversion is done. Since the output of the PPR step is usually presented in two equally weighted numbers and the final product is achieved with a carry propagation addition (or conversion), traditionally, the last step is called final addition. The general architecture of a parallel decimal multiplier is shown in Fig. 1.

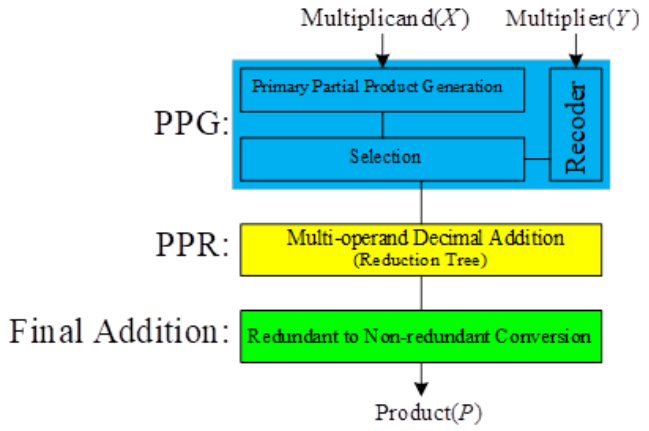

FIGURE 1. The general architecture of a parallel decimal multiplier.

The first parallel decimal multiplier was implemented in [18]. After that, several designs were been proposed in a period of fewer than ten years. These parallel decimal multipliers are partially or entirely distinguishable in the three main steps (i.e., PPG, PPR, and final addition), except that inputs and output are represented in binary coded decimal (BCD) encoding. A variety of redundant decimal digit sets and encodings are used for the representation of intermediate computation results, such as [0, 10] carry-save [18], [22], [−7, 7] signed digit [10], [23], overloaded decimal [0, 15] [20], double 4,2,2,1 [24], [−8, 8] SD [25], and overloaded excess-3 [0, 15] [26], [27].

B. ENERGY/POWER CONSUMPTION

The progress of increasing the number of transistors and clock frequencies to quench the high demand for more functionality in smaller and more effective portable devices makes power consumption one of the most critical issues in digital systems design [14]. In digital CMOS circuits, the power consumption (P_{total}) comes from two components, namely static (P_{static}) and dynamic ($P_{dynamic}$). The sub-threshold leakage through OFF transistors (I_{sub}), gate leakage through gate dielectric (I_{gate}), and junction leakage from source/drain diffusions (I_{junct}) are the main parts of static power that consume energy without any switching activity in the circuit [28]. On the other hand, dynamic power sources are charging/discharging load capacitances besides the short-circuit current of partially ON both pMOS and nMOS stacks.

Therefore, based on the above explanation, the total power consumption in CMOS circuits can be determined by Equation (1).

$$P_{total} = \underbrace{(I_{sub} + I_{gate} + I_{junct})V_{DD}}_{P_{static}} + \underbrace{\alpha C_L V_{DD}^2 f_{clk}}_{P_{dynamic}} \quad (1)$$

Besides the declared currents (I_{sub}, I_{gate}, and I_{junct}), V_{DD} is the supply voltage, α is the probability of switching the signal, C_L is the switching capacitance, and f_{clk} is the operational frequency.

In designing a low-power circuit, reducing any of the parameters mentioned above can be desirable; however, they strongly depend on the synthesis technology, except the probability of switching (α). Therefore, designers focus on α which has a direct impact on $P_{dynamic}$. There are several low-power designs (e.g. [29]–[31]) which take advantage of switching activity as a key parameter for saving power. The most commonly used switching activity reduction technique is clock gating which can be applied at RTL and gate-level of design abstraction. However, this technique is suitable for sequential design which reduces toggle rates on registers and cannot be applied for a fully combinational circuit. Moreover, power optimizations at the algorithm level (i.e., the arithmetic algorithm in this paper), the highest level of design abstraction, provide more choices for the designer to modify the design without compromising target performance or area.

Therefore, in this paper, we concentrate on the algorithm level of design abstraction to reduce switching activity. Based on our observations, the primary source of power consumption in parallel decimal multipliers is dynamic power, while the static power is negligible. However, static power consumption has grown exponentially due to scaling down to nanometer technology, and considering static power reduction techniques will be mandatory in the near future. It is worth mentioning that the proposed approach also paves the way for applying static power reduction techniques such as power gating.

Since reducing the operational frequency (f_{clk}) results in lower power consumption, just power consumption cannot be a convincing metric. In this paper, we use power-delay product or energy consumption for comparing different designs.

III. THE PROPOSED METHOD

The Karatsuba algorithm is a well-known technique for reducing the delay of large numbers multiplication which is based on divide and conquer. According to this algorithm, several high-speed multipliers have been proposed [32], [33]. However, this paper focuses on power reduction of decimal multiplication via partitioning, like the Karatsuba technique. It uses smaller multipliers (i.e., multiplier cells) and provides appropriate granularity for localizing switching activity. Nevertheless, some issues like the size of utilized multiplier cells and the basic multiplication algorithm should be studied carefully. The former is described in the following section, after a short overview of the decimal Karatsuba algorithm, and the latter is explained in the next section.

A. DECIMAL MULTIPLICATION VIA PARTITIONING

As mentioned above, the Karatsuba algorithm, via divide and conquer, accelerates the multiplication of large numbers. It was originally implemented for binary multiplication [34]. Based on this algorithm in decimal multiplication, for computing $P = X \times Y$ where $X = \sum_{i=0}^{k-1} x_i 10^i$ and $Y = \sum_{i=0}^{k-1} y_i 10^i$, the operands can be recursively divided into two

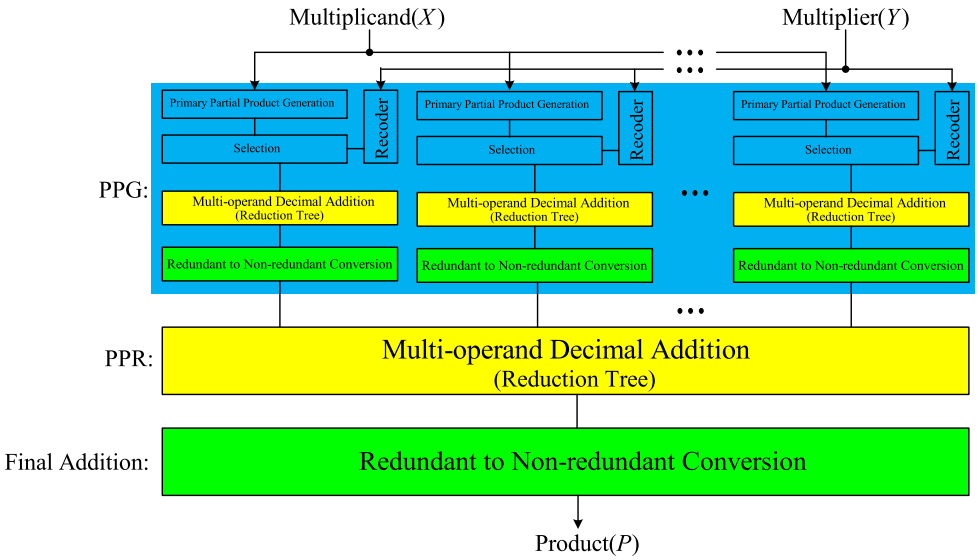

FIGURE 2. The abstract architecture of a parallel decimal multiplier via partitioning.

part X_H, X_L and Y_H, Y_L, respectively. In this regard, X_H and Y_H show the most significant parts, while X_L and Y_L demonstrate the least significant portions. Thus, by assuming $k = 2n$ and equal partitioning, the operands can be rewritten in the form of Equation (2).

$$\begin{aligned} X &= 10^n \sum_{i=0}^{n-1} x_{i+n} 10^i + \sum_{i=0}^{n-1} x_i 10^i = X_H 10^n + X_L \\ Y &= 10^n \sum_{i=0}^{n-1} y_{i+n} 10^i + \sum_{i=0}^{n-1} y_i 10^i = Y_H 10^n + Y_L \end{aligned} \quad (2)$$

After partitioning, based on the Karatsuba multiplication algorithm, the $P = X \times Y$ product is constructed by Equation (3).

$$\begin{aligned} P &= \left(X_H 10^n + X_L\right)\left(Y_H 10^n + Y_L\right) \\ &= 10^{2n} X_H Y_H + 10^n \left(X_H Y_L + X_L Y_H\right) + Y_L Y_L \end{aligned} \quad (3)$$

Implementation of Equation (3) needs four $n \times n$ digit decimal multiplications and two $2n$-digit and $3n$-digit additions. This algorithm can be recursively continued till it achieves to 1×1 digit multiplication. Therefore, the abstract architecture of decimal multiplication based on partitioning can be illustrated in Fig. 2.

In the above explanation, equal partitioning is assumed, which causes uniformity of the smaller multiplier cells (i.e., symmetric). However, in practice, different sizes of multiplier cells are possible (i.e., asymmetric). Since we want to consider all promising configurations and find the best one, based on statistical properties of possible inputs, we study the implementation of a 16 × 16 digit multiplier (i.e., IEEE 754-2008 standard) in two different types of symmetric and asymmetric in the following sub-sections.

B. SYMMETRIC APPROACH

For symmetric designs, we consider three different partitioning sizes of operands, namely, two, four, and eight. In the first design, multiplier and multiplicand are partitioned into two equal parts, so four 8 × 8 -digit multipliers are used to create a 16 × 16 -digit multiplier according to Equation (3). The output of multiplier cells must be aligned before the final reduction and conversion steps. This alignment is shown in Fig. 3, where the eight bits of the least significant part, without any extra computation, form the least significant part of the final result. However, the 8th to 23rd positions need a multi-operand addition. Also, an increment operation is required for the eight bits of the most significant part (increment with carry out of multi-operand addition).

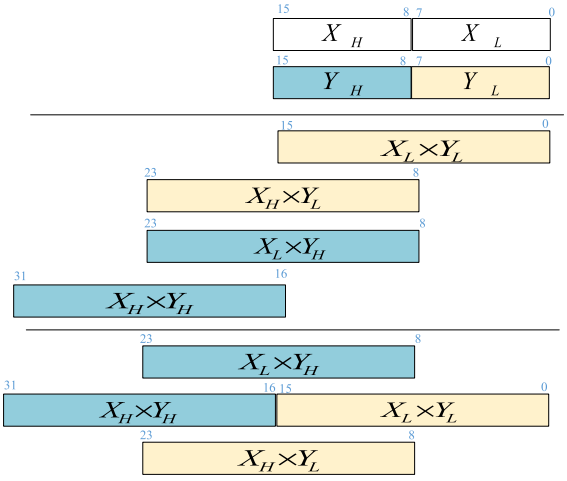

FIGURE 3. The arrangement of the 16 × 16-digit multiplier by four 8 × 8 multipliers (mult.16-8).

It should be mentioned that the algorithm of multiplier cells is essential since it has a significant impact on the detail design and low-level architecture of the main multiplier. Thus, the discussion about the multiplier cells and the related influential characteristics of the algorithm are presented in Section 4.

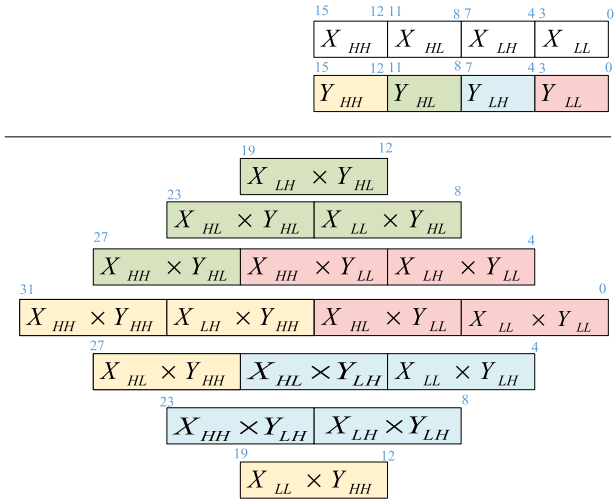

FIGURE 4. The arrangement of the 16 × 16-digit multiplier by sixteen 4 × 4-digit multipliers (mult16-4).

The idea of divide and conquer can recursively be used to implement each multiplier cell with smaller ones. For example, a $4n \times 4n$-digit multiplier needs four $2n \times 2n$-digit multipliers, as mentioned above. By more partitioning, each $2n \times 2n$-digit multipliers can be implemented via four $n \times n$-digit multipliers. According to these explanations, we can implement a 16×16-digit multiplier by using sixteen 4×4-digit multipliers, as shown in Fig. 4.

The extreme point of partitioning provides a parallel multiplier which its partial products generate via digit-by-digit multipliers. Since all the state-of-the-art multipliers use pre-computed multiplies (based on background information), we define the 2×2-digit multiplier as the smallest multiplier cell. According to this definition, we can partition operands into eight equal parts and construct a 16×16-digit multiplier by using sixty-four 2×2-digit multipliers, as shown in Fig. 5.

By reviewing the aforementioned architectures, two important issues should be considered. The first is the depth of the multi-operand adder, and the second is the output format of multiplier cells. The depth of the multi-operand adder in Fig. 3, 4, and 5 are 3, 7, and 15, respectively. Obviously, using smaller multiplier cells provides more granularity. However, it increases the area consumption. So, a detailed analysis with power dissipation monitoring is required to compromise the size of multiplier cells and area consumption.

To address the second issue, based on Fig. 2, the output of multiplier cells is in BCD format, which is provided after a redundant to non-redundant conversion. It causes an unnecessary time-consuming carry propagation that can be omitted since the output of multiplier cells feed to a multi-operand adder. Of course, this adder has to accept redundant inputs. This issue is related to the algorithm of multiplier cells, which is presented in Section 4.

C. ASYMMETRIC APPROACH

As mentioned before, different sizes of multiplier cells are possible. Although the bigger multiplier cell has more delay, it may increase the total delay of the main multiplier due to imbalance delay paths. Moreover, an asymmetric method can show considerable superiority for an input pattern with special statistical properties (The experimental results are provided in Section 5). Since possible partitioning for asymmetric design is many, this category's advantage is shown by studying just a straightforward partitioning to illustrate a specific pattern of power consumption in our proposed method.

In Fig. 6, the arrangement of the 16×16-digit multiplier is shown, which is partitioned asymmetrically. This multiplier is composed of various multiplier cells (i.e., four 4×4-digit multipliers, two 4×8-digit multipliers, two 8×4-digit multipliers, and one 8×8-digit multiplier).

IV. MULTIPLIER CELLS ALGORITHM AND MULTI-OPERAND ADDER STRUCTURE

This section discusses the multiplier cells algorithm and multi-operand adder structure, which processes the result of multiplier cells. To take advantage of state-of-the-art decimal multiplier designs, in Table 1, we summarize some characteristics of several previous relevant works, based on Multiplier Recoding (MR), pre-computed multiples values (PMV), Multiple Encoding (ME), Partial Product Digit Set (PPDS), Partial Product Digit Encoding (PPDE), Reduced Partial product Digit Set (RPDS), and Reduced Partial product Digit Encoding (RPDE). These characteristics and associated ranges are well explained in the corresponding references.

Among the mentioned characteristics, RPDS and RPDE are most important since they reflect the output format of multiplier cells. The reduced partial product of all designs needs five or more bits for representation, except [10], [20], and [25] which are representable just by four bits (Note that more bits mean more area and power consumption). The signed outputs of [10] and [25] cause difficulty in the multi-operand adder. Moreover, the structure and algorithm of multi-operand adders should be totally different from the utilized structure and algorithm of reduction parts of multiplier cells. However, in [20], not only the outputs of multiplier cells are unsigned, but also the same structure is used in all the reduction levels. In this design, X, $2X$, $4X$, and $5X$ (PMV) are generated parallelly in a carry-free manner. Then, for partial product reduction, a novel 2 to 1 reduction module (i.e., ODDS adder) is used, which provided output digit set (RPDS) and encoding (RPDE) are $[0, 15]$ and overloaded decimal, respectively. Therefore, a redundant to non-redundant conversion unit converts [0, 15] to [0, 9] with BCD encoding at the end of reduction.

In this paper, the algorithm of multiplier cells and structure of multi-operand adder is the same as [20]. In other words, the smaller multipliers (i.e., multiplier cells) that are 4×4, 4×8, 8×4, and 8×8 and 2×2-digits imitate the proposed 16×16 digit multiplier of [20], just the final conversion, as explained at the end of Section 3.2, is removed. It is worthy to note that this technique is general and can be applied

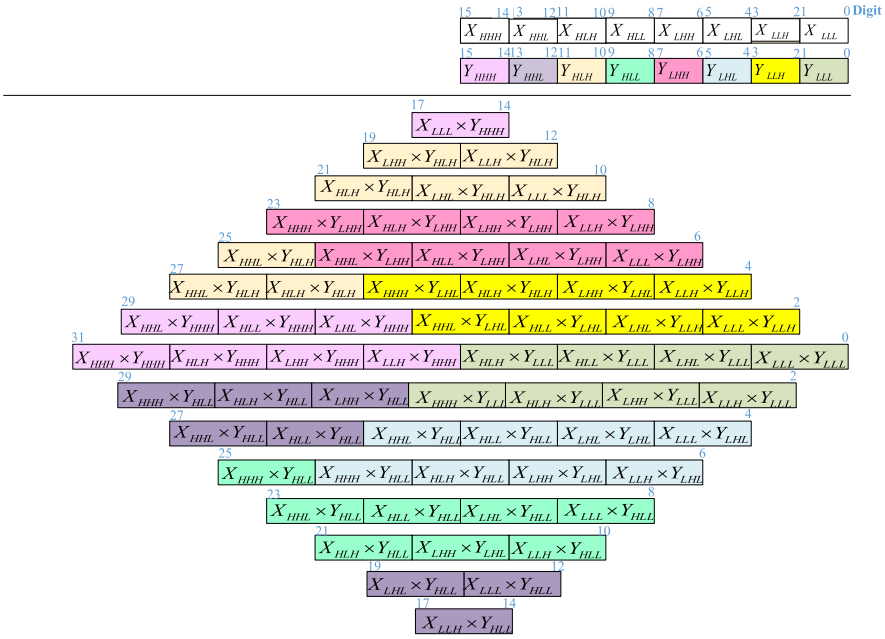

FIGURE 5. The arrangement of the 16 × 16-digit multiplier by sixty-four 2 × 2-digit multipliers.

TABLE 1. Summary of parallel decimal multipliers' characteristics.

Reference	MR	PMV	ME	PPDS	PPDE	RPDS	RPDE
[18]	None	$\{0, \pm1, \pm2,5,10\} \times X$	BCD	$[0, 18]$	Double BCD	$[0, 10]$	Carry-save
[20]		$\{0,1,2,4,5\} \times X$				$[0, 15]$	Overloaded decimal
[22]		$\{0,1,2,5,8,9\} \times X$				$[0, 10]$	Carry-save
[24] Radix-5		$\{0, \pm1, \pm2,5,10\} \times X$	4,2,2,1		Double 4,2,2,1	$[0, 18]$	Double 4,2,2,1
[24] Radix-10	$[-5, 5]$	$\{0,1,2,3,4,5\} \times X$		$[0, 9]$	4,2,2,1		
[25]			$[-8,8]$	$[-8,8]$	$-8,4,2,1,1$	$[-8,7]$	$-8,4,2,1$
[26]			Excess-3 BCD	$[0, 15]$	8,4,2,1	[0,24]	Double (Excess-6 BCD & BCD)
[27]							
[10]			$[-6,6]$ Sign-magnitude	$[-7,7]$	$-8,4,2,1$	$[-7,7]$	$-8,4,2,1$

to every decimal multiplier. However, as mentioned above, we select [20] algorithm due to its flexibility for providing the different sizes of multipliers. Moreover, the prosperity of this algorithm in terms of power consumption is studied in previous work [15].

V. EXPERIMENTAL RESULTS

In this section, we discuss how to implement, test, verify, and synthesize different designs. After that, we compare the power/energy consumption of various designs and the impact of the proposed technique on power/energy and area.

A. DESIGN AND IMPLEMENTATION

We modeled all designs with VHDL. We used a top-down methodology to manage the complexity of the hardware. After describing the hardware, they are verified by using a large number of test vectors, especially corner cases. For test and simulation, ModelSim toolbox is used.

B. LOGICAL SYNTHESIS AND POWER EXTRACTION FLOW

In order to achieve synthesis results with maximum accuracy, we used Synopsys Design Compiler suite and TSMC 130nm technology for synthesizing designs. At first, we synthesized designs for generating net-lists and ".saif" files, which the critical path delay constraints 10 ns. Then, in the second step, we used a ".do" file in ModelSim, which annotates created net-list in the previous phase and saves the annotated file in ".back-saif". Finally, we used the back-saif file for producing

TABLE 2. Power/energy consumption of proposed designs with coarse grain assumption.

#Scenario	Zero Digits	Power / Multiplier	Switching Power (mW)	Internal Power (mW)	Leakage Power (nW)	Total Power (mW)	Ratio
1	ALL Active	[20]	35.436	23.252	2.25e+04	58.688	100
		Mult16-8	29.066	21.053	2.53e+04	50.119	85.4
		Mult16-4	**25.939**	**18.094**	**2.91e+04**	**44.033**	**75.03**
		Mult16-2	32.267	22.828	3.47e+04	55.095	93.88
		Mult16-8-4	26.832	19.183	2.69e+04	46.015	78.4
2	$X = 0$	[20]	2.473	8.04e-02	2.25e+04	2.553	100
		Mult16-8	**1.158**	**6.80e-02**	**2.53e+04**	**1.226**	**48.02**
		Mult16-4	1.192	0.102	2.91e+04	1.294	50.69
		Mult16-2	1.189	0.131	3.47e+04	1.320	51.7
		Mult16-8-4	1.172	8.67e-02	2.69e+04	1.259	49.31
3	$X_H = 0$	[20]	21.296	12.157	2.25e+04	33.453	100
		Mult16-8	14.936	10.389	2.53e+04	25.325	75.70
		Mult16-4	**13.580**	**8.968**	**2.91e+04**	**22.549**	**67.41**
		Mult16-2	19.575	13.386	3.47e+04	32.961	98.53
		Mult16-8-4	14.074	9.639	2.69e+04	23.713	70.88
4	$X_L = 0$	[20]	21.908	12.412	2.25e+04	34.320	100
		Mult16-8	14.926	10.412	2.53e+04	25.338	73.83
		Mult16-4	**13.480**	**8.918**	**2.91e+04**	**22.398**	**65.26**
		Mult16-2	14.589	9.879	3.47e+04	24.467	71.29
		Mult16-8-4	13.792	9.449	2.69e+04	23.241	67.72
5	$X_H = 0$ $Y_H = 0$	[20]	14.040	6.209	2.25e+04	20.250	100
		Mult16-8	7.866	5.193	2.53e+04	13.058	64.48
		Mult16-4	7.144	4.485	2.91e+04	11.629	57.43
		Mult16-2	9.035	5.962	3.47e+04	14.998	74.06
		Mult16-8-4	**6.835**	**4.231**	**2.69e+04**	**11.067**	**54.65**
6	$X_H = 0$ $Y_L = 0$	[20]	14.169	6.470	2.25e+04	20.639	100
		Mult16-8	8.174	5.418	2.53e+04	13.592	65.86
		Mult16-4	**7.619**	**4.784**	**2.91e+04**	**12.402**	**60.09**
		Mult16-2	12.164	8.146	3.47e+04	20.309	98.4
		Mult16-8-4	8.397	5.655	2.69e+04	14.052	68.08
7	$X_L = 0$ $Y_H = 0$	[20]	14.843	6.751	2.25e+04	21.594	100
		Mult16-8	8.170	5.427	2.53e+04	13.598	62.97
		Mult16-4	**7.144**	**4.485**	**2.91e+04**	**11.629**	**53.85**
		Mult16-2	7.931	5,127	3.47e+04	13.058	60.47
		Mult16-8-4	7.221	4.558	2.69e+04	11.779	54.55
8	$X_L = 0$ $Y_L = 0$	[20]	13.928	6.142	2.25e+04	20.070	100
		Mult16-8	7.874	5.200	2.53e+04	13.074	65.14
		Mult16-4	**7.229**	**4.531**	**2.91e+04**	**11.760**	**58.59**
		Mult16-2	7.968	5.204	3.47e+04	13.173	65.64
		Mult16-8-4	7.877	5.246	2.69e+04	13.123	65.38

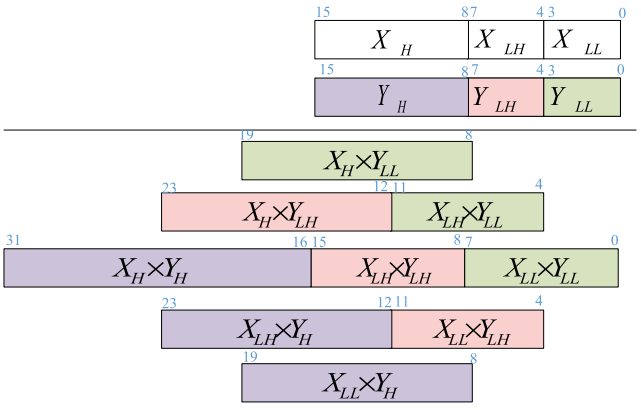

FIGURE 6. The arrangement of the 16 × 16-digit multiplier with asymmetric partitioning (mult16-8-4).

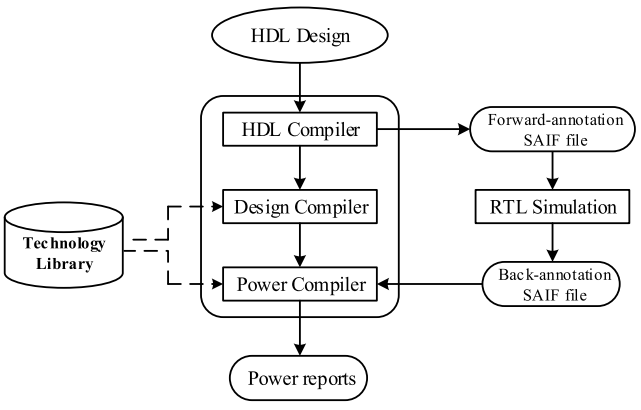

FIGURE 7. Method of calculating consumed power/energy and data flow (adopted from [35]).

detailed power consumption reports. Fig. 7 shows the data flow and different phases of synthesis by Design compiler. It should be mentioned that all designs have met the 10 ns time constraint.

C. SYNTHESIS RESULTS

Tables 2 and 3 contain synthesis results of the original multiplier [20] and proposed architectures based on partitioning. In these tables, the detailed power consumption

TABLE 3. Power/energy consumption of proposed designs with fine grain assumption.

#Scenario	Zero Digit	Power / Multiplier	Switching Power (mW)	Internal Power (mW)	Leakage Power (nW)	Total Power (mW)	Ratio
9	$X_{LL}=0$ $Y_H=0$	[20]	18.463	9.539	2.25e+04	28.003	100
		Mult16-8	12.108	8.230	2.53e+04	20.338	72.63
		Mult16-4	10.789	6.992	2.91e+04	17.781	63.5
		Mult16-2	12.399	8.320	3.47e+04	20.720	73.99
		Mult16-8-4	**10.278**	**6.648**	**2.69e+04**	**16.927**	**60.45**
10	$X_{HH}=0$ $Y_{LH}=0$	[20]	23.093	13.465	2.25e+04	36.558	100
		Mult16-8	17.074	11.867	2.53e+04	28.940	79.16
		Mult16-4	14.940	10.009	2.91e+04	24.950	68.25
		Mult16-2	20.492	14.165	3.47e+04	34.656	94.8
		Mult16-8-4	**13.425**	**9.232**	**2.69e+04**	**22.657**	**61.97**
11	$X_{LH}=0$ $Y_{HL}=0$	[20]	21.665	12.536	2.25e+04	34.201	100
		Mult16-8	**13.335**	**8.789**	**2.53e+04**	**22.124**	**64.69**
		Mult16-4	17.568	11.893	2.91e+04	29.461	86.14
		Mult16-2	17.416	11.749	3.47e+04	29.165	85.28
		Mult16-8-4	14.315	9.801	2.69e+04	24.116	70.51
12	$X_{LH}=0$ $Y_{LH}=0$	[20]	23.327	13.669	2.25e+04	36.996	100
		Mult16-8	16.994	11.821	2.53e+04	28.815	77.89
		Mult16-4	17.568	11.893	2.91e+04	29.461	79.63
		Mult16-2	18.461	12.648	3.47e+04	31.109	84.09
		Mult16-8-4	**16.020**	**11.135**	**2.69e+04**	**27.155**	**73.4**
13	$X_{LL}=0$ $X_{HL}=0$ $X_{HH}=0$	[20]	14.737	6.774	2.25e+04	21.511	100
		Mult16-8	8.447	5.528	2.53e+04	13.974	64.96
		Mult16-4	**7.284**	**4.441**	**2.91e+04**	**11.725**	**54.51**
		Mult16-2	10.706	6.990	3.47e+04	17.69	82.24
		Mult16-8-4	7.464	4.728	2.69e+04	12.192	56.68
14	$X_{LL}=0$ $X_{HL}=0$ $X_{HH}=0$ $Y_{HL}=0$	[20]	12.776	5.138	2.25e+04	17.959	100
		Mult16-8	6.286	4.039	2.53e+04	10.324	57.49
		Mult16-4	**5.498**	**3.288**	**2.91e+04**	**8.785**	**48.92**
		Mult16-2	7.567	4.861	3.47e+04	12.428	69.20
		Mult16-8-4	5.604	3.416	2.69e+04	9.020	50.23
15	$X_{LL}=0$ $X_{HL}=0$ $Y_{HH}=0$ $Y_{HL}=0$	[20]	14.022	6.295	2.25e+04	20.317	100
		Mult16-8	8.351	5.481	2.53e+04	13.832	68.08
		Mult16-4	**7.060**	**4.355**	**2.91e+04**	**11.415**	**56.18**
		Mult16-2	8.509	5.496	3.47e+04	14.005	65.7
		Mult16-8-4	7.630	4.955	2.69e+04	12.585	61.94
16	$X_{LL}=0$ $X_{HL}=0$ $X_{HH}=0$ $Y_{HH}=0$ $Y_{HL}=0$	[20]	10.742	3.605	2.25e+04	14.346	100
		Mult16-8	8.119	5.225	2.53e+04	13.345	93.02
		Mult16-4	**3.777**	**2.158**	**2.91e+04**	**5.935**	**41.37**
		Mult16-2	5.060	3.160	3.47e+04	8.220	57.3
		Mult16-8-4	4.157	2.485	2.69e+04	6.642	46.3

(i.e., Switching power, internal power, leakage power, total power consumption, and the ratio with respect to the original design) under various scenarios are summarized. In each scenario, five different architectures of a 16 × 16 digit multiplier are examined. The first one is original work which presented in [20], the second, third, and fourth ones are made via four 8 × 8 (i.e., Mult.16-8), sixteen 4 × 4 (i.e., Mult.16-4), and sixty-four 2 × 2 multipliers (i.e., Mult.16-2), respectively. The last one presents the results of asymmetric architecture, which contains one 8 × 8, four 4 × 4, two 4 × 8, and two 8 × 4-digit multipliers. The block diagrams of these designs were shown in Fig. 3 to 6.

As two extreme cases, in the first scenario, all the bits of multiplicand and multiplier are active (they are changed during the simulation). In contrast, in the second, all the bits of multiplicand are constant and equal to zero. The other possible scenarios are done by partitioning the multiplicand and multiplier into equal halves, which are illustrated in rows 3 to 8. The finer granularity partitioning is presented in Table 3, that the multiplicand and multiplier are partitioned into four equal parts, and the #scenario is labeled from 9 to 16. The total power consumptions of mentioned sixteen scenarios are plotted in Fig. 8, which provides a better view for comparing these architectures.

Based on these results, the proposed designs consume less power/energy than the original multiplier in all scenarios. However, the Mult16-2 design cannot compete with the others due to using tiny multiplier cells, which causes a large multi-operand addition and, as a result, higher power/energy consumption. The Mult16-8 in scenarios #2 and #11 shows the best results since these scenarios are matched with the architecture of Mult16-8 (See Fig.3). The asymmetric architecture (Mult16-8-4) provides best results in scenarios #5, #9, #10, and #12. In other circumstances, the Mult16-4 is the best. Moreover, this architecture can provide about 25% power/energy reduction in scenario #1, which can be considered as a general case when all signals are active.

In Fig. 9, the area dissipation of each design is shown. As expected, the implementation based on partitioning techniques enforces the area overhead. Although, this issue

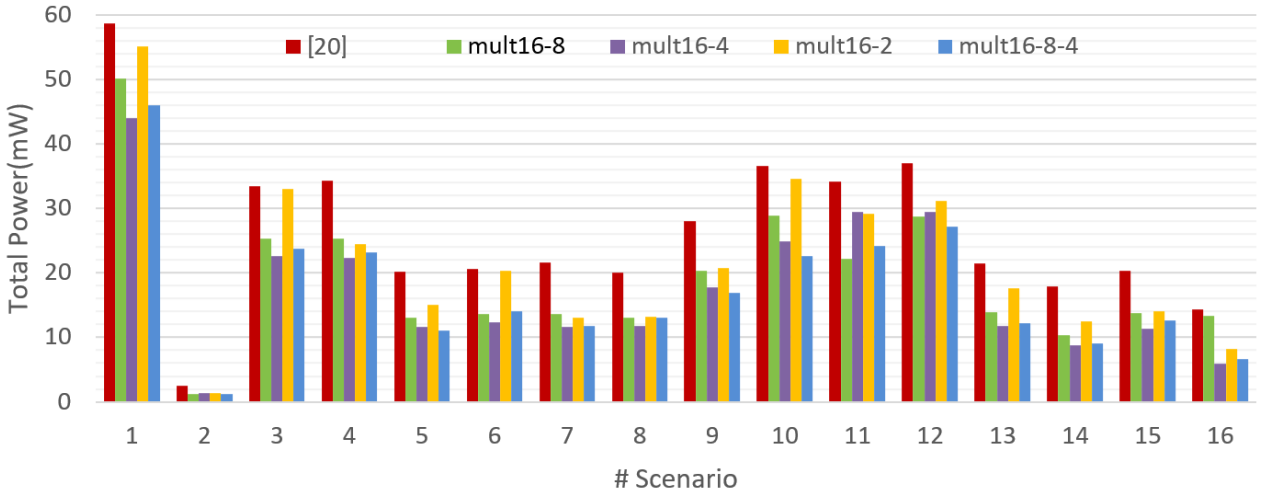

FIGURE 8. The total power/energy consumption of different designs in various scenarios.

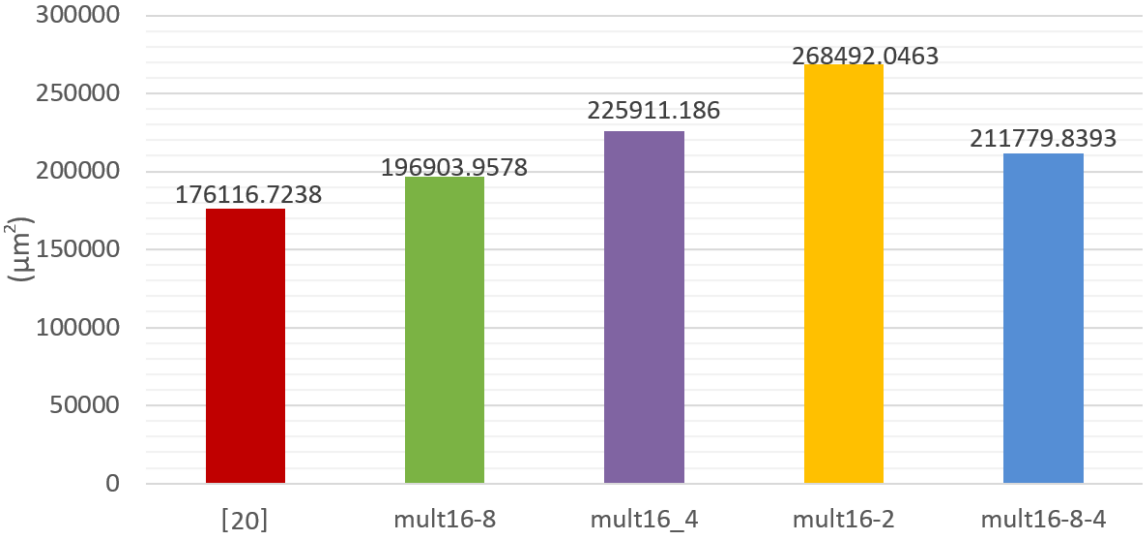

FIGURE 9. Comparison of area dissipation of different designs.

is previously highlighted by the leakage power amount in Tables 2 and 3. The circuit complexity and area dissipation are usually increased by applying power reduction techniques. However, the main target of these techniques, as explained in Section 2, is decreasing the switching activity of the entire circuit. Presented experimental results show, with acceptable area overhead, lower power/energy consumption is achievable.

VI. CONCLUSION

Due to the importance of decimal computation, in this paper, we study a design methodology for reducing the power consumption of decimal multipliers. The suggested method is based on partitioning, which constructs a large multiplier with the smaller ones. The proposed architectures and the original multiplier (which are used as multiplier cells) were implemented via structural VHDL and synthesized with Synopsys Design-compiler under 130 nm TSMC technology library in typical conditions. The experimental results demonstrate 25% power/energy reduction when all the input signals are active. Moreover, it shows that the architecture should be selected based on statistical properties of possible input data, while surprisingly, the asymmetric architecture can provide the best results on specific scenarios.

REFERENCES

[1] M. F. Cowlishaw, "Decimal floating-point: Algorism for computers," in *Proc. 16th IEEE Symp. Comput. Arithmetic*, Jun. 2003, pp. 104–111.
[2] *IEEE Standards Committee*, Standard 754-2008, 2008, doi: 10.1109/IEEESTD.2008.4610935.
[3] *Draft IEEE Standard for Floating-Point Arithmetic*, Standard P754/D2.50, Apr. 2019.
[4] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlough, "The IBM z900 decimal arithmetic unit," in *Proc. 25th Asilomar Conf. Signals, Syst. Comput.*, 2001, pp. 1335–1339.
[5] L. Eisen, J. Ward, H. Tast, and N. Mading, "IBM POWER6 accelerators: VMX and DFU," *IBM J. Res. Develop.*, vol. 51, no. 6, pp. 633–684, 2007.

[6] C. F. Webb, "IBM Z10: The next-generation mainframe microprocessor," *IEEE Micro*, vol. 28, no. 2, pp. 19–29, Mar. 2008.
[7] A. Vazquez and E. Antelo, "Conditional speculative decimal addition," in *Proc. 7th Conf. Real Numbers Comput.*, Jul. 2006, pp. 47–57.
[8] R. D. Kenney and M. J. Schulte, "High-speed multi operand decimal adders," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 953–963, 2005.
[9] M. A. Erle and M. J. Schulte, "Decimal multiplication via carry-save addition," in *Proc. 14th IEEE Int. Conf. Appl.-Specific Syst., Archit., Processors*, 2003, pp. 348–358.
[10] S. Gorgin and G. Jaberipur, "Sign-magnitude encoding for efficient VLSI realization of decimal multiplication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 1, pp. 75–86, Jan. 2017, doi: 10.1109/TVLSI.2016.2579667.
[11] A. Kaivani, A. Hosseiny, and G. Jaberipur, "Improving the speed of decimal division," *IET Comput. Digit. Techn.*, vol. 5, no. 5, pp. 393–404, 2011.
[12] A. Hosseiny and G. Jaberipur, "Decimal Goldschmidt: A hardware algorithm for radix-10 division," *Comput. Electr. Eng.*, vol. 53, pp. 40–55, Jul. 2016.
[13] V. Vázquez, J. Villalba-Moreno, E. Antelo, and E. L. Zapata, "Redundant floating-point decimal CORDIC algorithm," *IEEE Trans. Comput.*, vol. 61, no. 11, pp. 1551–1562, Nov. 2012.
[14] J. Rabaey, *Low Power Design Essentials*, 2009th ed. Springer, 2009.
[15] A. Malekpour, A. Ejlali, and S. Gorgin, "A comparative study of energy/power consumption in parallel decimal multipliers," *Microelectron. J.*, vol. 45, no. 6, pp. 775–780, Jun. 2014.
[16] R. D. Kenney, M. J. Schulte, and M. A. Erle, "A high-frequency decimal multiplier," in *Proc. 22nd IEEE Int. Conf. Compute. VLSI Comput. Processors*, Oct. 2004, pp. 26–29.
[17] M. A. Erle, E. M. Schwarz, and M. J. Schulte, "Decimal multiplication with efficient partial product generation," in *Proc. 17th IEEE Symp. Comput. Arithmetic*, Oct. 2005, pp. 21–28.
[18] T. Lang and A. Nannarelli, "A radix-10 combinational multiplier," in *Proc. Fortieth Asilomar Conf. Signals, Syst. Comput.*, 2006, pp. 313–317.
[19] A. Vazquez, E. Antelo, and P. Montuschi, "A new family of high. Performance parallel decimal multipliers," in *Proc. 18th IEEE Symp. Comput. Arithmetic*, Jun. 2007, pp. 195–204.
[20] S. Gorgin and G. Jaberipur, "A fully redundant decimal adder and its application in parallel decimal multipliers," *Microelectron. J.*, vol. 40, no. 10, pp. 1471–1481, Oct. 2009.
[21] S. Gorgin, G. Jaberipur, and B. Parhami, "Design and evaluation of decimal array multipliers," in *Proc. 43th Asilomar Signals, Syst., Comput.*, 2009, pp. 1782–1786.
[22] G. Jaberipur and A. Kaivani, "Improving the speed of parallel decimal multiplication," *IEEE Trans. Comput.*, vol. 58, no. 11, pp. 1539–1552, Nov. 2009.
[23] S. Gorgin and G. Jaberipur, "Fully redundant decimal arithmetic," in *Proc. 19th IEEE Symp. Comput. Arithmetic*, Jun. 2009, pp. 145–152.
[24] A. Vazquez, E. Antelo, and P. Montuschi, "Improved design of high-performance parallel decimal multipliers," *IEEE Trans. Comput.*, vol. 59, no. 5, pp. 679–693, May 2010.
[25] L. Han and S.-B. Ko, "High-speed parallel decimal multiplication with redundant internal encodings," *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 956–968, May 2013.
[26] A. Vazquez, E. Antelo, and J. D. Bruguera, "Fast radix-10 multiplication using redundant BCD codes," *IEEE Trans. Comput.*, vol. 63, no. 8, pp. 1902–1914, Aug. 2014.
[27] X. Cui, W. Dong, W. Liu, E. E. Swartzlander, and F. Lombardi, "High performance parallel decimal multipliers using hybrid BCD codes," *IEEE Trans. Comput.*, vol. 66, no. 12, pp. 1994–2004, Dec. 2017.
[28] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. London, U.K.: Pearsons, 2011.
[29] V. G. Moshnyaga and K. Tamaru, "A comparative study of switching activity reduction techniques for design of low-power multipliers," in *Proc. Int. Symp. Circuits Syst.*, 1995, pp. 1560–1563.
[30] N.-Y. Shen and O. T.-C. Chen, "Low-power multipliers by minimizing switching activities of partial products," in *Proc. IEEE Int. Symp. Circuits Systems. Proc.*, Oct. 2002, pp. 93–96.
[31] O. T. C. Chen, S. Wang, and Y.-W. Wu, "Minimization of switching activities of partial products for designing low-power multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 3, pp. 418–433, Jun. 2003.
[32] M. Vestias and H. Neto, "Parallel decimal multipliers and squarers using karatsuba-Ofman's algorithm," in *Proc. 15th Euromicro Conf. Digit. Syst. Design*, Sep. 2012, pp. 782–788.
[33] E. Kyritsis and K. Pekmestzi, "Hardware efficient fast FIR filter based on Karatsuba algorithm," in *Proc. 5th Int. Conf. Modern Circuits Syst. Technol. (MOCAST)*, May 2016, pp. 1–4.
[34] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *Doklady Akademii Nauk*, vol. 145, no. 2, pp. 293–294, 1962.
[35] *Synopsys Power Compiler User Guide*, document D-2010.03-SP2, 2010.

SAEID GORGIN received the B.S. degree in computer engineering from Islamic Azad University South Tehran Branch, Tehran, Iran, in 2001, the M.S. degree in computer engineering from Islamic Azad University Tehran Science and Research Branch, Tehran, in 2004, and the Ph.D. degree in computer system architecture from Shahid Beheshti University, Tehran, in 2010. He is currently an Associate Professor of computer engineering with the Department of Electrical Engineering and Information Technology, Iranian Research Organization for Science and Technology, Tehran. He is also a Visiting Scientist at the Computer Systems Laboratory, Department of Computer Engineering, Chosun University, South Korea. His current research interests include computing systems, computer arithmetic, and VLSI design.

HELIA ZAREIE NEJAD received the B.Sc. degree in computer engineering and the M.Sc. degree in computer systems architecture engineering from the Islamic Azad University of Qazvin, Iran, in 2011 and 2015, respectively. She is currently a Computer Programmer and a Backend Developer at "NSUN" Company, Tehran, Iran. She has a passion for broadening her knowledge of computer science and cutting-edge technology. Her current research interests include computer arithmetic, parallel processing, and low-power systems.

JEONG-A. LEE (Senior Member, IEEE) received the B.S. degree (Hons.) in computer engineering from Seoul National University, Seoul, South Korea, in 1982, the M.S. degree in computer science from Indiana University Bloomington, IN, USA, in 1985, and the Ph.D. degree in computer science from the University of California at Los Angeles, CA, USA, in 1990. From 1990 to 1995, she was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX, USA. Since 1995, she has been with Chosun University, Gwangju, South Korea. From 2008 to 2009, she served as a Program Director of the ECE Division, National Research Foundation of Korea. Her current interests include high-performance computer architectures, memory architecture, approximate computing, and genetic circuit modeling. She is a member of the National Academy of Engineering, South Korea.