Carry Disregard Approximate Multipliers

Nima Amirafshar*, Ahmad Sadigh Baroughi*, Hadi Shahriar Shahhoseini*§, and Nima TaheriNejad†

***Abstract*—Several challenges in improving the performance of computing systems have given rise to emerging computing paradigms. One of these paradigms is approximate computing. Many applications require different levels of accuracy and are error-tolerance to a certain degree. Approximate computations can reduce the calculation complexities significantly and thus improve the performance. Here, we propose a methodology for designing approximate N-bit array multipliers based on carry disregarding. We evaluate and analyze the proposed multipliers both experimentally and theoretically. The proposed 8-bit multipliers, compared to the exact multiplier, reduce the critical path delay, power consumption, and area by 29%, 29%, and 30%, on average. Compared to the existing approximate array architectures in the literature, they have improved 14.3%, 22.8%, and 26.4%, respectively. Compared to the exact 16-bit multiplier, the proposed 16-bit multipliers have reduced the delay, power consumption, and area by 35%, 24%, and 23% on average. In an image processing application, we have also demonstrated the applicability of a wide range of proposed multipliers, which have Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) over 30 dB and 94%, respectively.**

***Index Terms*—Approximate computing, carry disregard multiplier, power-efficient, image processing.**

I. Introduction

One main goal of computer architecture design is to achieve high performance. We notice a substantial growth in data volume with different characteristics and, as a result, many processing operations in various applications, so that the number and variety of applications have also increased significantly. Therefore, the architecture of today's computer systems is limited for processing such a significant amount of data, leading to inefficiency. Hence, the architecture of processing systems should be centered on data characteristics. Many specific applications do not mandate precise calculations. Numerous applications are inherently error-tolerant, such as Machine Learning, Scientific Computing, Data Analytics, and Signal Processing [1]–[6]. Also, human perceptual limitations make it possible to use approximations in many further applications, such as Image Processing and Multimedia [7]–[9]. Therefore, approximate computing is efficient and endeavors to achieve high efficiency in speed, area, and power or energy consumption by compromising accuracy.

Approximate computing methods can be divided into software, architecture, and circuit levels [10]. We can mention Loop Perforation, Code Perforation, and Inexact Program Versions at the software level [11]. Methods such as Instruction Set Architecture (ISA) Extension [12], Approximate Accelerator, and Approximate Storage are at the architectural level [13]. The circuit level also includes methods such as Voltage and Frequency Scaling. However, one of the most common is Inexact Hardware, which designs inaccurate numerical and logical units [14], [15]. Multiplication is one of the most common numerical operations. Conventional exact multipliers have significant critical path delays and power consumption, which, due to their repeated use in various applications, lead to limited efficiency and increased energy consumption. Hence, approximate multipliers have become popular and improved a wide range of different applications.

This paper presents approximate array multipliers based on carry disregarding. They significantly reduce power consumption, critical path delay, and area compared to exact and existing approximate multipliers. Applications are error tolerant up to a certain level. Hence, the proposed approximate multipliers have different error levels, and some have the highest accuracy compared to previous approximate multipliers. Also, they have created a better balance between accuracy and hardware performance criteria. The key contributions of this paper can be summarized as follows:

1) A methodology for designing efficient approximate N-bit array multipliers based on carry disregarding,
2) Theoretical analysis of the proposed methodology, providing equations for accuracy criteria for optimal design,
3) Design approximate 8-bit and 16-bit array multipliers with balanced hardware efficiency and accuracy criteria,
4) Achieving the most efficient designs (mostly are Pareto frontier) in terms of power, area, power-delay product, and accuracy compared to recent state-of-the-art designs,
5) Evaluating proposed designs in image processing.

The rest of this paper is as follows: Section II reviews some related works. In Section III, we investigate the background of exact multipliers. Sections IV and V propose our 8-bit approximate multipliers and their extension, respectively, and Section VI theoretically analyzes the accuracy of the proposed multipliers. We describe our experiments and results in Section VII and make a comprehensive multidimensional comparison with recent state-of-the-art approximate multipliers in Section VIII. Also, Section IX implies the pertinence of proposed designs in image processing application, and the paper is concluded in Section X.

II. Related Work

In general, multipliers consist of three main computation stages: Partial Product (PP) generation, PP accumulation (reduction), and a final addition ascertaining the multiplication result. There are three main architectures to accumulate PPs and reduce them: the Carry-Save Adder (CSA) array, the Wallace tree, and the Dadda tree [10]. Tree-based PP accumulation

* Authors are with Iran University of Science and Technology, Tehran, Iran. E-mail: {nima_amirafshar, sadighbaroughi_a}@elec.iust.ac.ir, shahhoseini@iust.ac.ir
† Author is with Heidelberg University, Heidelberg, Germany and TU Wien, Vienna, Austria. E-mail: nima.taherinejad@ziti.uni-heidelberg.de
§ Corresponding Author: shahhoseini@iust.ac.ir

has less delay than array structures. In contrast, they have more power consumption and area than array structures due to the more significant number of computing units and higher complexity [10], [16]. The array-based multiplier has a straightforward, uniform, and modular architecture. Correspondingly, such architecture mostly has less power consumption and area; also, the development and management of this architecture are more comprehensible and more optimal than Wallace- and Dadda-based architectures [10], [16].

Designers employ the approximation in three vital computing stages of multipliers. One of the ways is operand truncation, which takes advantage of the fact that not all operand bits are equally substantial. Hence, a substantially smaller core multiplier results from merely choosing a portion of the operand bits [17], [18]. On the other hand, we can apply approximation during the PP accumulation stage. The fundamental component for accumulating PPs in approximate Wallace and Dadda multipliers is the approximate compressor, which has undergone much research; for example, [19]–[21].

Regarding array multipliers, using approximate adders such as approximate Full-Adder (FA), Half-Adder (HA), and compressor is one of the conventional methods for applying approximation in reducing PPs. Hence, [22] proposed an imprecise 4:2 compressor, which by using it in an array structure, designed an approximate 8-bit multiplier. Another method is to design a small-scale approximate multiplier and use it recursively to design larger ones. [23] first proposed an approximate 2 2 multiplier as a building block, then by using it, designed larger multipliers recursively. The main idea of this paper is to change the logical function of the 2-bit multiplier such that the approximate multiplier converts the only available 4-bit output (i.e., 3 3 = 9) to show the approximate form with 3-bit (i.e., 7). As a result, it has a simpler circuit and lacks adders and XOR gates. [24] proposed approximate 2-bit multipliers based on the concept of equating similar output bits and also designed larger multipliers recursively. In exact 2-bit multiplication, Most-Significant Bit (MSB) and Least-Significant Bit (LSB) are opposite in only three cases. Hence, [24] removed the logic circuit related to LSB, connected MSB to LSB, and finally managed to reduce the maximum magnitude of the error compared to [23].

Another conventional method is PP truncation. [25] proposed Broken-Array Multiplier (BAM) truncates several PPs using Vertical Break Level (VBL) and Horizontal Break Level (HBL) parameters. Additionally, [15] proposed the probabilistic analysis-based PP array and used the proposed Propagation and Generation (PG) function to replace numerous PPs with different values. Afterward, some of them were truncated, which shortened the delay. Also, it uses approximate FA and HA for accumulating PPs and designs large multipliers recursively. Carry propagation is the primary limiting factor for array multipliers. Hence, [14] presented an approximate array multiplier in which specific columns of Partial Product Unit (0) disregard all generated carry, and our proposed designs share some architectural similarity with [14].

We note that approximate compute units beyond Complementary Metal-Oxide Semiconductor (CMOS) are gaining currency as well. For instance, in [26], the authors have proposed highly-scalable Majority Gates (MGs) based on spin-CMOS technology and used them to design approximate compressors. In [27] and [9], the authors designed various approximate adders for In-Memory Computation (IMC) using memristive stateful logics. However, they are outside the scope of this work, since they require technologies beyond CMOS.

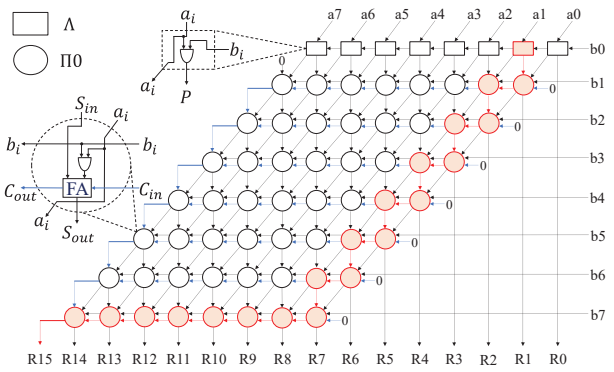


Fig. 1: Conventional exact 8-bit multiplier architecture and logic circuit of partial product unit (0)

III. Background

Figure 1 shows a conventional 8-bit array multiplier whose main component is the 0; each 0 consists of an AND Gate ( ) for single-bit multiplication and a full adder. This multiplier has a significant critical path delay due to the high dependence between 0s. An n-bit array multiplier has n^2 elements. Its critical path has $3n$ 3 elements in which the number of 0s and s are $3n$ 4 and 1, respectively. For simplicity in calculating the critical path delay, we assume that the delay of the , OR, and XOR gates are 1, 1, and 2 cycles, respectively. Hence the critical path delay will be $12n$ 15. As the multiplier scale increases (i.e., larger n), the total number of cells increases significantly. Thus, the dependence between 0s continues over a larger area, leading to a significant increase in critical path delay. For example, in 8-, 16-, 32-, and 64-bit multipliers, the total number of cells is 64, 256, 1024, and 4096, respectively, and the critical path delays are 81, 177, 369, and 753. Figure 1 shows the critical path of an 8-bit multiplier in red.

In general, the 0s depend on the adjacent 0s due to carry and summation inputs. However, the carry has a much more significant effect on increasing the critical path delay. For example, if we disregard all the carries, we will see that these disregarding put all the columns of 0s together independently, and they can operate in parallel. Accordingly, the main idea of the proposed methods in this paper is to disregard carries. Also, we can convert large-scale multiplications into smaller ones by using the distributive property. According to Equation (1), we can convert an 8-bit multiplication to two 8 4 multiplications.

$$A \quad B = A(BH \quad 2^4 + BL) = (A \quad BH)2^4 + (A \quad BL); \quad (1)$$

where A and B are 8 bits, and BH and BL are the most-significant bits and least-significant bits of B, respectively. As shown in Figure 2, we can design an 8-bit multiplier

using two smaller 8 4 multipliers and a Carry Look-ahead Adder (CLA). In this case, the two 8 4 multipliers operate independently and in parallel. Eventually, the CLA determines the final result of 8-bit multiplier. There are two advantages to using two smaller multipliers together. First, they have a much less critical path delay due to their smaller scale than an 8-bit multiplier, which equals 49 cycles. On the other hand, the parallel operation of the two 8 4 multipliers, which causes their delay overlap, is the second advantage. Finally, a CLA significantly reduces the delay due to carries propagation in calculating the sum of the results of two 8 4 multipliers. Therefore, using this structure can improve the critical path delay, which is the basis of the proposed designs this paper. Nevertheless, there is still dependence between 0s in every 8 4 multiplier. So, this is the leading cause of the delay in the overall performance of the 8-bit multiplier. Therefore, the paper's primary purpose is to use approximate computing and apply appropriate error levels in the calculations to reduce the dependence between 0s to achieve a better balance between accuracy and critical path delay, power consumption, and area.

IV. Proposed Approximate 8-bit Multipliers

This paper proposes approximate 8-bit multipliers that are inspired by the multiplier of Figure 2 and based on two 8 4 multipliers. In this architecture, 8 4 multipliers operate in parallel; therefore, reducing the critical path delay in 8 4 multipliers significantly reduces the 8-bit multiplier delay. The carry propagation in 0s of 8 4 multipliers is the leading reason for the strong dependence between 0s and blocks the possibility of parallel operation of them; hence, causing a significant delay and an increase in power consumption. Therefore, if we disregard carries in 0 columns, each column can operate in parallel independently of the other column, reducing delay. Also, by disregarding carry, we can use simple partial product units with less hardware complexity. Figure 3 shows our proposed partial product units. In general, we can divide the design process into two steps. First, we must design approximate 8 4 multipliers based on disregarding the carry. For each 8 4 multiplier, there are many combinations of 0s columns to disregard the carry. Each of these creates a new approximate 8 4 multiplier. In the second step, we have to choose two multipliers from the approximate 8 4 multipliers of the previous step, for which many combinations are possible too. In the end, we will have a large number of approximate 8-bit multipliers, each having a different critical path delay, power consumption, area, and levels of accuracy. This paper aims to use approximate computing and reduce the level of accuracy more optimally.

1) Proposed approximate 8 4 multipliers: Figure 4 shows some of the proposed carry-disregard-based approximate 8 4 multipliers. The total number of proposed 8 4 multipliers is 9, which are cd_2 to cd_a. In all of them, our starting point is to disregard the carries of the second column because the first column has no dependence on the carry and contains only one . Therefore in all proposed approximate 8 4 multipliers, we have disregarded the carries from the second column to a specific column. Hence, in their name, there is a number in hexadecimal form, which shows from the second column to which column we disregard the carries. For example, the cd_6 approximate multiplier (i.e., Figure 4c) disregards the carries from the Column 2 to the Column 6. As a result, in this multiplier, all columns 1 to 7 operate independently and in parallel. As stated earlier, we can use simpler and more efficient units instead of 0s since we disregard carries in partial product units. Therefore, we propose three partial product units and use them in our approximate multipliers. Figure 3 shows the logic circuits of Carry Disregard Partial Product Unit (1), Half-adder-Based Partial Product Unit (2), and Full-adder-Based Partial Product Unit (3) as follows: The 1 uses one for single-bit multiplication and one XOR gate for determining the sum of S_{in} with output. 1 has no inputs and outputs for the carry (i.e., C_{in} and C_{out}, respectively) and disregards them. The 2 has a for single-bit multiplication and a half-adder for calculating the sum of S_{in} with output. The 2 has a C_{out} output, but the only difference with a conventional 0 is that it disregards the C_{in} input. The 3 has two s for single-bit multiplications and a full-adder for determining the sum of S_{in} with the output of two s. This unit has the C_{out} output but disregards the C_{in} input. The 3 is a combination of two 0s, so its use reduces the two carry outputs (i.e., the C_{out} outputs in the two 0s) to one. Therefore, for example, the 2 in Column 5 of cd_3 (i.e., Figure 4b), can operate independently and in parallel with its previous columns.

The approximate multiplier cd_2 (i.e., Figure 4a) disregards only a carry of Column 2. Therefore, the Column 2 has a 1, and columns 1 to 3 operate independently and in parallel. As a result, its critical path delay is 41 cycles, less than the delay of the exact 8 4 multiplier (i.e., 49 cycles). In all approximate 8 4 multipliers in Figure 4, we show the critical path in red. Approximate multipliers cd_3 (i.e., Figure 4b), cd_4, cd_5, and cd_6 (i.e., Figure 4c), follow a similar procedure in their architecture. The cd_3 disregards all the carries up to Column 3, so all the elements up to this column are 1 type. In Column 4, there is an 3 and an 2. Column 5 also contains an 2, so other elements of this multiplier are 0s. The approximate multipliers cd_4 and cd_6 to disregard the carries up to columns 4 and 6, respectively. The critical path delay of the approximate multipliers cd_3 to cd_6 is 37, 33, 29, and 25 cycles, respectively. We can see that if we disregard the carries in more columns, the critical path delay decreases.

The approximate multiplier cd_7 (i.e., Figure 4d) up to Column 7 disregards all the carries, and as a result, all columns 1 to 8 operate independently and in parallel. Up to Column 7, its cells are 1s, and in Column 8, it has one 3 and one 2. However, since in cd_7, the first partial product unit in Column 9 does not generate any carry, therefore, we use a 1. The critical path delay of the cd_7 is 21 cycles. The cd_8 (i.e., Figure 4e) up to Column 8 disregards all the carries; hence, all columns 1 to 9 operate independently and in parallel. Therefore, columns 2 to 8 have 1s, and in Column 9, because the first partial product unit does not generate any carry, we use a 1. The cd_8 also has two 3s and one 2 that we connect their C_{out} output to the S_{in} input of the adjacent cell. Since we disregard the carries in more columns, cd_8 has a smaller critical path delay, which is 13 cycles. The cd_9 (i.e., Figure 4f)

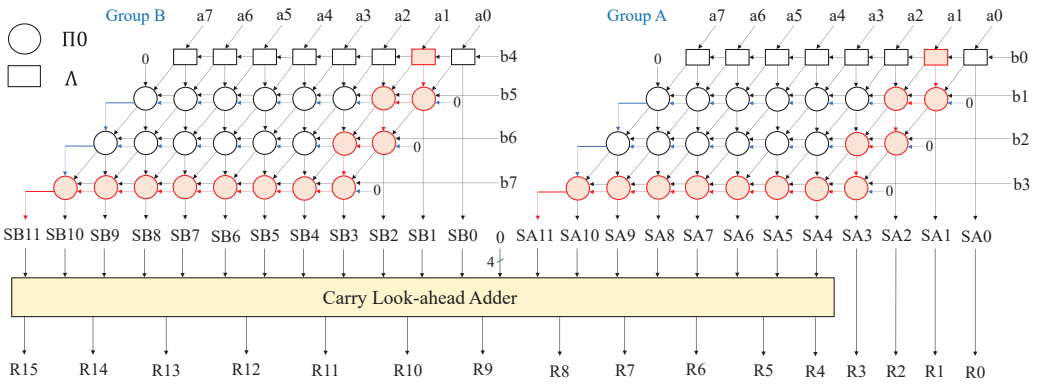
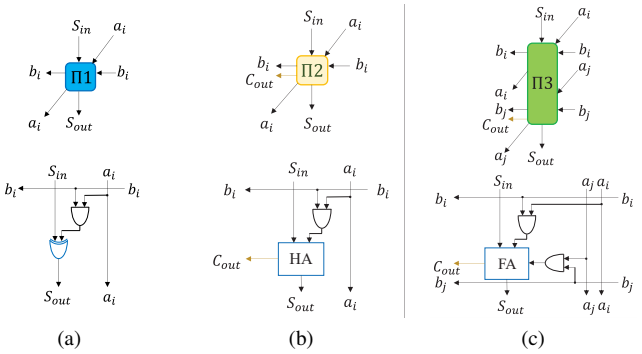


Fig. 2: The architecture of an exact 8-bit multiplier using two exact 8 4 multipliers.



(a) (b) (c)

Fig. 3: Circuits and symbols of (a) 1, (b) 2, and (c) 3

disregards all carries up to Column 9, and columns 1 to 10 operate in parallel and independently. The cd_9 has two 2s, and we connect the C_{out} output of the first one to the S_{in} input of the second one. Also, its other partial product units are of the 1 type. The cd_a up to Column 10 disregards all carries, and the partial product unit in Column 11 does not generate carry. Therefore, all its elements are of 1 type, hence, it has the simplest circuit among other proposed approximate 8 4 multipliers, which leads to lower power consumption and smaller area. The critical path delay of the cd_9 and cd_a equals 7 cycles, the most negligible delay compared to all the proposed approximate 8 4 multipliers.

2) Approximate 8-bit Carry Disregard Multiplier (CDM8): After designing the approximate 8 4 multipliers, we now have to choose two multipliers for groups A and B. Therefore, we must choose so that the final 8-bit multiplier has the least possible delay and, at the same time, is optimal in terms of power consumption and area. In this architecture, we can divide the factors causing the delay in the 8-bit multiplier into three parts: (i) The delay of the 8 4 multiplier of Group A, (ii) the delay of the 8 4 multiplier of Group B, and (iii) the delay of CLA. Accordingly, we can minimize the final delay when the three delays are as small as possible, and the units operate in parallel. The two 8 4 multipliers of groups A and B are completely independent and operate in parallel. Nevertheless, the CLA adder depends on the results of both. As a result, we can say that CLA is a limiting factor. So if we can provide the CLA input pair of bits as quickly as possible, the CLA can start operating correctly. For example, according to Figure 2, by producing the pairs of bits $SA4$ and $SB0$

TABLE I: Proposed approximate CDM8s

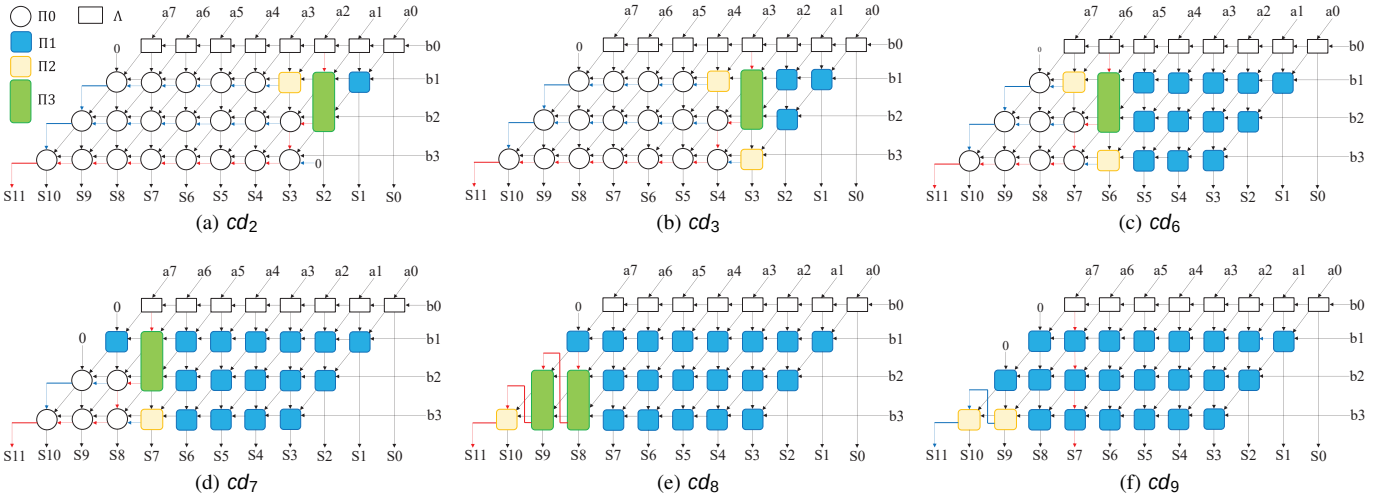
Design	CDM8_44	CDM8_50	CDM8_62	CDM8_73	CDM8_74	CDM8_84	CDM8_95	CDM8_a6	CDM8_a7	CDM8_a8	CDM8_a9	CDM8_aa
Group A	cd_4	cd_5	cd_6	cd_7	cd_7	cd_8	cd_9	cd_a	cd_a	cd_a	cd_a	cd_a
Group B	cd_4	*Exact*	cd_2	cd_3	cd_4	cd_4	cd_5	cd_6	cd_7	cd_8	cd_9	cd_a

with a minor delay compared to each other and continuing this process to the more significant bits, respectively, the CLA can operate parallel to groups A and B. Therefore, to achieve this parallelism, we must choose 8 4 multipliers of groups A and B in a specific way.

Table I shows the approximate 8-bit CDM8_xy multipliers with different ranges of accuracy, critical path delay, power consumption, and area. The hexadecimal numbers x and y determine the type of 8 4 multipliers of groups A and B (i.e., cd_x and cd_y), respectively. For example, the CDM8_84 multiplier in groups A and B uses the approximate 8 4 multipliers cd_8 and cd_4, disregarding the carry up to Column 8 and Column 4, respectively. This approximate 8-bit multiplier determines the output of Column 8 of Group A (i.e., $SA7$) after 13 cycles and the output of Column 4 of Group B (i.e., $SB3$) after 13 cycles. While the exact multiplier of Figure 2 obtains the outputs of Column 8 of Group A and Column 4 of Group B after 37 and 21 cycles, respectively, which differ by 16 cycles. Hence, the proposed multiplier CDM8_84 not only achieves the outputs of Column 8 of Group A and Column 4 of Group B faster but also the difference between them is 0 cycle. Therefore, the CLA can calculate the sum of the two faster. On the other hand, in CDM8s, we used simpler 1, 2, and 3, which also reduced power consumption and area.

V. Extension of Proposed Approximate Multiplier

Figure 5 shows the architecture of the proposed approximate 16-bit multipliers. Each multiplier has four 8-bit approximate multipliers and four approximate 8-bit adders. The four 8-bit multipliers are independent and operate in parallel, which reduces the critical path delay of the approximate 16-bit multiplier. Also, each approximate adder is a smaller-scale adder consisting of one Approximate 8-bit Carry Look-ahead Adders (CLAx). Using such adders reduces many hardware complexities. CLAxs only have no carry output (i.e., C_{out}); as



(a) cd_2 (b) cd_3 (c) cd_6

(d) cd_7 (e) cd_8 (f) cd_9

Fig. 4: Proposed approximate 8 4 carry disregard multipliers

a result, the approximate adders related to the least-significant bits (i.e., CLAx1 and CLAx2) are independent of the approximate adders in the most-significant bits (i.e., CLAx3 and CLAx4). Consequently, their parallel operation improves the delay of the approximate 16-bit multiplier. In general, each proposed approximate N-bit multiplier (i.e., N 16) has four N=2-bit smaller scale multipliers (i.e., Mul_1 to Mul_4) and four N=2-bit smaller scale approximate adders. Each N=2-bit approximate adder is CLA-based, consisting of $\frac{N}{16}$ 1 exact 8-bit CLAs and one 8-bit CLAx in its most-significant bits. The N=2-bit approximate adder disregards the final carry; therefore, the approximate adders related to the least-significant bits and the approximate adders in the most-significant bits are independent and operate in parallel.

The design process of approximate N-bit multipliers (N 16) is similar, and we can divide it into two steps. In the first step, we must design the desired approximate N=2-bit multipliers. Hence, the starting point is designing of approximate 16-bit multiplier and using it recursively for the larger multiplier. For the proposed approximate 16-bit multiplier, we use the CDM8s that we introduced earlier. The CDM8s have much lower critical path delay than conventional 8-bit multipliers and operate in parallel. Hence, these factors cause a significant reduction in the delay of the 16-bit multiplier. In the second step, we have to choose four multipliers among the CDM8s. There are many choices for this, each of which creates a new approximate multiplier. Generally, the final delay of the 16-bit multiplier depends on the delay of four 8-bit multipliers and the delay of approximate adders (i.e., CLAxs). Hence, we see a significant reduction in the delay of the approximate 16-bit multiplier when these units can operate in parallel. Nevertheless, the CLAxs can not operate in parallel with 8-bit multipliers because CLAxs depend on the results of 8-bit multipliers. However, each CLAx has eight pairs of bits as inputs. Hence if we can provide the pairs of input bits as quickly as possible, the CLAx can start the summation process correctly. Also, another essential factor is the time difference of providing two bits in each pair. In other words, as regards each pair, if we can provide two bits with a negligible

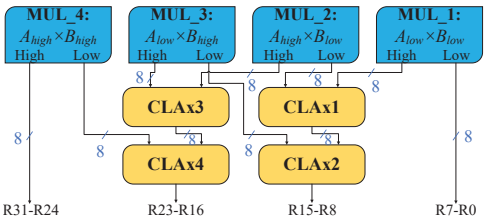


Fig. 5: Proposed approximate 16-bit multiplier architecture

time difference, then CLAx can start its calculations sooner. The proposed 8-bit multipliers similarly had such a situation regarding their CLAs, which we discussed comprehensively in Section IV and explained by mentioning an example.

Based on this, we propose fourteen approximate 16-bit unsigned multipliers called 16-bit Carry Disregard Multiplier (CDM16), which have different accuracy, delay, power consumption, and area levels. Table II shows the type of 8-bit multipliers in each proposed CDM16. We have named each proposed 16-bit multiplier CDM16_$wxyz$, in which the hexadecimal digits w, x, y, and z show that we have disregarded the carry in multipliers mul_1 to mul_4, up to which bit of their output. Therefore, these digits determine the type of CDM8s that we have considered for each of them. For example, in the CDM16_b330, we used CDM8_a8, CDM8_40, CDM8_40, and an exact multiplier for the multipliers mul_1 to mul_4, respectively. Choosing 8-bit multipliers in this way reduces the delay of 16-bit multipliers. Also, using CDM8s and CLAxs reduce hardware complexity and consequently improves power consumption and area.

VI. Accuracy Analysis

A. Accuracy Criteria

Accuracy criteria indicate the error level and, consequently, the quality of the outputs. Several criteria have been proposed to assess the accuracy of approximate arithmetic circuits [19], [20], [28], [29]. Let us assume that P_i is the output of the exact multiplier that calculates the multiplication of N-bit operands A_i and B_i accurately (i.e., $P_i = A_i B_i$). Also,

TABLE II: Proposed approximate CDM16s

Design	Mul_1: $A_{low}B_{low}$	Mul_2: $A_{high}B_{low}$	Mul_3: $A_{low}B_{high}$	Mul_4: $A_{high}B_{high}$
CDM16_0000	*Exact*	*Exact*	*Exact*	*Exact*
CDM16_3000	*CDM*8_40	*Exact*	*Exact*	*Exact*
CDM16_7000	*CDM*8_84	*Exact*	*Exact*	*Exact*
CDM16_8000	*CDM*8_95	*Exact*	*Exact*	*Exact*
CDM16_8330	*CDM*8_95	*CDM*8_40	*CDM*8_40	*Exact*
CDM16_b330	*CDM*8_a8	*CDM*8_40	*CDM*8_40	*Exact*
CDM16_f770	*CDM*8_aa	*CDM*8_84	*CDM*8_84	*Exact*
CDM16_f880	*CDM*8_aa	*CDM*8_95	*CDM*8_95	*Exact*
CDM16_f883	*CDM*8_aa	*CDM*8_95	*CDM*8_95	*CDM*8_40
CDM16_fbb3	*CDM*8_aa	*CDM*8_a8	*CDM*8_a8	*CDM*8_40
CDM16_fff7	*CDM*8_aa	*CDM*8_aa	*CDM*8_aa	*CDM*8_84
CDM16_fff8	*CDM*8_aa	*CDM*8_aa	*CDM*8_aa	*CDM*8_95
CDM16_fffb	*CDM*8_aa	*CDM*8_aa	*CDM*8_aa	*CDM*8_a8
CDM16_ffff	*CDM*8_aa	*CDM*8_aa	*CDM*8_aa	*CDM*8_aa

assume that $\overset{X}{i}$ is the output of the approximate multiplier, which an error can accompany. Table III shows all the accuracy criteria we used in this paper. Error (E_i) and Error Distance (ED_i) are primary criteria that other accuracy criteria calculated by them. All the proposed approximate multipliers are carry-disregard-based. Therefore their output is always smaller than the output of the exact multiplier for all possible input combinations. Consequently, in our approximate multipliers, E_i and ED_i are equal.

B. CDM8 Accuracy Analysis

Our purpose in this section is to provide an equation for ED_i. This equation can determine the ED_i for all possible input combinations for any design of CDM8. Also, by obtaining ED_i, we can easily calculate other accuracy criteria. The CDM8_xy multipliers have two 8 4 multipliers in groups A and B (i.e., cd_x and cd_y), then an exact CLA calculates the summation of the 8 4 multipliers output. Therefore, according to the Equation (2), ED_i for each CDM8_xy (i.e., ED^8_{xy}) is equal to the sum of the product of the ED_i of 8 4 multipliers (i.e., ED_{cd_x} and ED_{cd_y}) in own weight. First, we determine the ED_{cd_x} of the Group A 8 4 multiplier. The cd_x contains 11 columns, which ignores all the carries from Column 2 to Column x. Therefore, in the first step, for each combination of inputs, we determine the number of carries that Column 2 to Column x generate independently (i.e., without propagating the carries from one column to the adjacent column). In the second step, for each column, we multiply the number of generated carries (i.e., $C_k(A_i;B_i)$) by its weight and calculate the sum of their results. Hence, Equation (3) calculates ED_{cd_x}.

$$ED^8_{xy} = ED_{cd_x} + 2^4 \quad ED_{cd_y} \tag{2}$$

$$ED_{cd_x}(A_i;B_i) = \overset{X}{_{k=0}} 2^k \quad C_k(A_i;B_i) \tag{3}$$

In Equation (3), $k = 0$ indicates that the 8 4 multiplier does not disregard any carries and thus will work accurately; hence for $k = 0$, the $C_0(A_i;B_i)$ is 0. We want to determine the number of carries generated by each column for each input combination (i.e., $C_k(A_i;B_i)$). Pair of logical ones in each column generate a carry independently. Therefore, the number of generated carries in each column equals the integer part of the division of the number of logical ones by two. That is,

$$C_k(A_i;B_i) = \frac{L_{1\_k}(A_i;B_i)}{2} \;; \tag{4}$$

where $L_{1\_k}(A_i;B_i)$ determines the number of logical ones in column k per A_i and B_i inputs. If we determine the summation of the logical ones and logical zeros in each column, then we get the number of logical ones in that column. Based on this, Equation (5) determines the number of carries each column generates for each input combination.

$$C_k(A_i;B_i) = \begin{cases} 0 & k = 0;1 \\ \left\lfloor \frac{1}{2} \sum_{n=0}^{k-1} a_{k-n-1} b_n \right\rfloor & 2 \le k \le 4 \\ \left\lfloor \frac{1}{2} \sum_{n=0}^{3} a_{k-n-1} b_n \right\rfloor & 5 \le k \le 8 \\ \left\lfloor \frac{1}{2} \sum_{n=k-8}^{3} a_{k-n-1} b_n \right\rfloor & 9 \le k \le 11 \end{cases} \tag{5}$$

Where k is the column number of the cd_x, a_0 a_7 and b_0 b_3 are A_i and B_i input bits, respectively. Also, calculating the ED_{cd_y} is entirely similar to the ED_{cd_x} so that we can determine the number of generating carries in each column of the cd_y (i.e., $C^*_k(A_i;B_i)$, according to Equation (6).

$$C_k(A_i;B_i) = \begin{cases} 0 & k = 0;1 \\ \left\lfloor \frac{1}{2} \sum_{n=0}^{k-1} a_{k-n-1} b_{n+4} \right\rfloor & 2 \le k \le 4 \\ \left\lfloor \frac{1}{2} \sum_{n=0}^{3} a_{k-n-1} b_{n+4} \right\rfloor & 5 \le k \le 8 \\ \left\lfloor \frac{1}{2} \sum_{n=k-8}^{3} a_{k-n-1} b_{n+4} \right\rfloor & 9 \le k \le 11 \end{cases} \tag{6}$$

Where k is the column number of the cd_y multiplier, a_0 a_7 and b_4 b_7 are A_i and B_i input bits, respectively. As a result, we can calculate the ED_{cd_y} according to Equation (7) for the inputs A_i and B_i. Consequently, we can calculate the ED^8_{xy} according to Equation (8).

$$ED_{cd_y}(A_i;B_i) = \overset{X}{_{k=0}} 2^k \quad C_k(A_i;B_i) \tag{7}$$

$$ED^8_{xy}(A_i;B_i) = \overset{X}{_{k=0}} 2^k \quad C_k(A_i;B_i) + 2^4 \overset{X}{_{k=0}} 2^k \quad C_k(A_i;B_i) \tag{8}$$

C. CDM(N bit) Accuracy Analysis

This section aims to provide an equation for ED_i for all possible input combinations for any design of CDM(N bit) (for N 16). We first determine the ED_i equation for any design of CDM16 and then generalize it for CDM(N bit) multipliers. In general, the ED_i of each CDM16 is due to the ED_i of the four CDM8s and the ED_i of the four CLAxs. Therefore, the ED_i of each CDM16 is equal to the sum of the

TABLE III: Accuracy criteria

Accuracy Criteria	Equation	Description
Error (E_i)	$E_i = {}^{P}_{i} - {}^{X}_{i}$	Refers to the difference between the exact multiplier's output and the approximate multiplier's output [28].
Error Distance (ED_i)	$ED_i = \|E_i\| = \|\ {}^{P}_{i} - {}^{X}_{i}\|$	Represents the absolute value of E_i [19], [20], [28], [29].
Mean Error Distance (MED)	$\frac{1}{2^{2N}} \sum_{i=1}^{2^{2N}} ED_i$	Refers to the average of ED_i for all possible input combinations [19], [29].
Relative Error Distance (RED_i)	$ED_i = {}^{P}_{i} \quad \forall \ {}^{P}_{i} \neq 0$	Refers to the ratio of ED_i to the output of the corresponding exact multiplier that is not equal to zero [20], [28], [29].
Mean Relative Error Distance (MRED)	$\frac{1}{2^{2N}} \sum_{i=1}^{2^{2N}} RED_i$	Refers to the average of RED_i for all possible input combinations [19], [20], [28], [29].
Normalized Mean Error Distance (NMED)	$MED/(2^N - 1)^2$	Refers to the average of ED_i divided by the most considerable exact multiplier output, i.e., $(2^N - 1)^2$ [19], [20], [28], [29].
Probability of Correctness (PC)	$\#\ _C/2^{2N}$	Refers to the ratio of the number of correct outputs of the approximate multiplier to the total number of possible outputs. $\#\ _C$ is the number of correct outputs of the approximate multiplier [20], [28].
Number of Effective Bits (NoEB)	$2N - \log_2(1 + \sqrt{MSE})$	MSE is the mean squared error obtained as the average of ED_i^2 for all possible input combinations, i.e., $MSE = \frac{1}{2^{2N}} \sum_{i=1}^{2^{2N}} ED_i^2$ [20], [28].

product of each CDM8s error distance and the CLAxs error distance in their corresponding weight. That is,

$$ED^{16} = ED^{8}_{X1Y1} + 2^8(ED^{8}_{X2Y2}) + 2^8(ED^{8}_{X3Y3}) + 2^{16}(ED^{8}_{X4Y4}) + 2^8(ED_{CLAx\_L}) + 2^{16}(ED_{CLAx\_M}); \quad (9)$$

where ED^{8}_{XiYi} is the error distance of each CDM8, ED_{CLAx_L} is the error distance of the least-significant part approximate adders of the CDM16 (i.e., CLAx1 and CLAx2), and ED_{CLAx_M} is the error distance of the most-significant part approximate adders of the CDM16 (i.e., CLAx3 and CLAx4). Therefore, we have to first calculate the ED^{8}_{XiYi} for each approximate 8-bit multiplier, which is possible using Equation (8). Next, we have to obtain the ED_i of the approximate adders (i.e., ED_{CLAx_L} and ED_{CLAx_M}). Hence, we must determine the least-significant and most-significant parts of each MUL_1 to MUL_4 output. The output of these four multipliers can be calculated by Equations (10) to (13). According to Equation (14), we can separate their most-significant part and least-significant part (i.e., $R_{m_{i\_H}}$ and $R_{m_{i\_L}}$), where r_{i_0} to r_{i_15} are the bits of the MUL_i multiplier output. As a result, according to Equation (15), by dividing the output of each multiplier by 2^8, the integer part of the division result will be the most-significant part of the desired multiplier output. By obtaining the most-significant part, we can easily calculate least-significant part using Equation (16).

$$MUL\_1: \ R_{m_1} = A_L B_L \quad ED^{8}_{X1Y1}(A_L; B_L) \quad (10)$$

$$MUL\_2: \ R_{m_2} = A_H B_L \quad ED^{8}_{X2Y2}(A_H; B_L) \quad (11)$$

$$MUL\_3: \ R_{m_3} = A_L B_H \quad ED^{8}_{X3Y3}(A_L; B_H) \quad (12)$$

$$MUL\_4: \ R_{m_4} = A_H B_H \quad ED^{8}_{X4Y4}(A_H; B_H) \quad (13)$$

$$R_{m_i} = \sum_{n=0}^{15} 2^n \quad r_{i\_n} = 2^8 \quad R_{m_{i\_H}} + R_{m_{i\_L}} = 2^8 \sum_{n=8}^{15} 2^{n-8} \quad r_{i\_n} + \sum_{n=0}^{7} 2^n \quad r_{i\_n} \quad (14)$$

$$R_{m_{i\_H}} = \left\lfloor \frac{R_{m_i}}{2^8} \right\rfloor \quad (15)$$

$$R_{m_{i\_L}} = R_{m_i} \quad 2^8 \quad R_{m_{i\_H}} \quad (16)$$

The CLAxs work like a exact 8-bit CLA, except that they disregard the last generated carry, which weights 2^8. Hence, to calculate the ED_{CLAx_L}, we need to obtain the number of carries that the CLAx1 and CLAx2 disregard. According to Equation (17), by dividing the sum of the $R_{m_{1\_H}}$, $R_{m_{2\_L}}$, and $R_{m_{3\_L}}$ by the weight of the last generated carry (i.e., 2^8), the integer part of the division result is the number of carries which the CLAx1 and CLAx2 disregard. In the same way, according to Equation (18), we can get the number of carries that the CLAx3 and CLAx4 disregard.

$$N_{CLAx\_L}(A_i; B_i) = \left\lfloor \frac{R_{m_{1\_H}} + R_{m_{2\_L}} + R_{m_{3\_L}}}{2^8} \right\rfloor \quad (17)$$

$$N_{CLAx\_M}(A_i; B_i) = \left\lfloor \frac{R_{m_{2\_H}} + R_{m_{3\_H}} + R_{m_{4\_L}}}{2^8} \right\rfloor \quad (18)$$

Hence, by Equation (19), we can calculate the ED_{CLAx_L}. It is equal to the product of the number of carries that the CLAx1 and CLAx2 disregard (i.e., N_{CLAx_L}) by the weight of those carries (i.e., 2^8). Similarly, Equation (20) calculates the ED_{CLAx_M}. Finally, using Equation (21), we can calculate the error distance of each CDM16. The ED_i of each CDM(N bit) (for N 16) is equal to the sum of the product of each MUL_1 to MUL_4 error distance and the approximate adders error distance in their corresponding weight. The weight corresponding to MUL_1 to MUL_4 are 1, $2^{N=2}$, $2^{N=2}$, and 2^N, respectively, and the weight corresponding to approximate adders in the least-significant and most-significant parts are $2^{N=2}$ and 2^N, respectively. According to Equations (22) and (23), we can determine the most-significant and least-significant parts of each MUL_1 to MUL_4 output. Similar to the 16-bit multipliers, we can get the error distance of each CDM(N bit). Hence, Equation (24) can easily calculate ED^N for N 16.

$$ED_{CLAx\_L}(A_i; B_i) = 2^8 \quad N_{CLAx\_L}(A_i; B_i) \quad (19)$$

$$ED_{CLAx\_M}(A_i; B_i) = 2^8 \quad N_{CLAx\_M}(A_i; B_i) \quad (20)$$

$$ED^{16} = ED^{8}_{X1Y1} + 2^{8}(ED^{8}_{X2Y2} + ED^{8}_{X3Y3}) + 2^{16}(ED^{8}_{X4Y4}) + 2^{16}\left(\frac{\left\lfloor \frac{R_{m_1}}{2^8} \right\rfloor + R_{m_2} \quad 2^8 \left\lfloor \frac{R_{m_2}}{2^8} \right\rfloor + R_{m_3} \quad 2^8 \left\lfloor \frac{R_{m_3}}{2^8} \right\rfloor}{2^8}\right) + 2^{24}\left(\frac{\left\lfloor \frac{R_{m_2}}{2^8} \right\rfloor + \left\lfloor \frac{R_{m_3}}{2^8} \right\rfloor + R_{m_4} \quad 2^8 \left\lfloor \frac{R_{m_4}}{2^8} \right\rfloor}{2^8}\right) \quad (21)$$

$$R_{m_i\_H} = \left\lfloor \frac{R_{m_i}}{2^{N=2}} \right\rfloor \quad (22)$$

$$R_{m_i\_L} = R_{m_i} \quad 2^{N=2} \quad R_{m_i\_H} \quad (23)$$

$$ED^{N} = ED_{MUL1} + 2^{\frac{N}{2}}(ED_{MUL2} + ED_{MUL3}) + 2^{N}(ED_{MUL4}) + 2^{N}\left(\frac{\left\lfloor \frac{R_{m_1}}{2^{\frac{N}{2}}} \right\rfloor + R_{m_2} \quad 2^{\frac{N}{2}} \left\lfloor \frac{R_{m_2}}{2^{\frac{N}{2}}} \right\rfloor + R_{m_3} \quad 2^{\frac{N}{2}} \left\lfloor \frac{R_{m_3}}{2^{\frac{N}{2}}} \right\rfloor}{2^{\frac{N}{2}}}\right) + 2^{\frac{3N}{2}}\left(\frac{\left\lfloor \frac{R_{m_2}}{2^{\frac{N}{2}}} \right\rfloor + \left\lfloor \frac{R_{m_3}}{2^{\frac{N}{2}}} \right\rfloor + R_{m_4} \quad 2^{\frac{N}{2}} \left\lfloor \frac{R_{m_4}}{2^{\frac{N}{2}}} \right\rfloor}{2^{\frac{N}{2}}}\right) \quad (24)$$

VII. Experiments and Results

A. Hardware Efficiency Criteria

Critical path delay, power consumption, and area are the main criteria for hardware evaluation and analysis. On the other hand, combining these main criteria and considering them together is very important. Therefore, we consider Power Delay Product (PDP), Power Area Delay Product (PADP), Power Delay Error Product (PDEP), and Power Area Delay Error Product (PADEP) criteria for a more comprehensive evaluation and analysis of different designs. In PDEP and PADEP criteria, we use MRED for error.

B. Experimental setups

We used Verilog HDL to describe the proposed approximate multipliers and the ISE Design Suite-Xilinx to verify them. The Genus Synthesis Solution was then used to synthesize the proposed designs with 45-nm NanGate technology. Afterward, we analyzed the three primary hardware efficiency criteria: critical path delay, power consumption, and area. For evaluating the accuracy, we determined all the mentioned accuracy criteria in Section VI-A using Python for the proposed 8-bit and 16-bit multipliers for all possible input combinations (i.e., 2^{16} and 2^{32}, respectively).

C. Results

Table IV, shows the results of our evaluations regarding the hardware efficiency and accuracy criteria for the proposed unsigned 8-bit multipliers. Compared to the exact unsigned 8-bit multiplier (denoted as "Exact8" in Table IV), CDM8s improve the essential hardware efficiency criteria, i.e., critical path delay, power consumption, and area, by 29%, 29%, and 30%, respectively. Among all proposed CDM8s, CDM8_51 has the highest delay, power consumption, and area (improved by 15.7%, 8.8%, and 10.4%, respectively, compared to the Exact8). However, it has the lowest MRED, which is 0.0039. The lowest power consumption and area belong to CDM8_aa (improved by 51.5% and 48.1%, respectively, compared to the Exact8). Nevertheless, has the highest MRED is 0.2145 among all the CDM8s. Meanwhile, the CDM8_95 with MRED of 0.0518 has the lowest critical path delay, which has improved it by 36% compared to the exact multiplier and significantly reduced power and area by 31.9% and 33.3%, respectively.

The PDP indicates the energy efficiency, and the PADP evaluates the energy efficiency and area together. CDM8s have improved them by 48.1% and 64.8%, respectively. CDM8_aa has the best PDP and PADP, mainly because of its lower power consumption and area than all CDM8s. Also, the PDEP evaluates the energy efficiency and MRED together. The PADEP, the product of PDP, Area, and MRED, evaluates all these criteria together. CDM8_51 has the lowest PDEP and PADEP, mainly due to its very low MRED compared to all CDM8s. Also, all the CDM8s have a completely accurate result in most different input combinations, and their errors are significantly less in cases with inaccurate results.

Table V, shows the results of our evaluations for the proposed unsigned 16-bit multipliers. Compared to the exact unsigned 16-bit multiplier (denoted as "Exact16" in Table V), CDM16s improve the critical path delay, power consumption, and area by 35%, 24%, and 23%, respectively. Among all CDM16s, CDM16_0000, CDM16_3000, CDM16_7000, and CDM16_8000 have the highest delay, power consumption, area, PDP, and PADP, respectively, but have the least MRED, which is 0.009806. However, compared to the Exact16, they improved mentioned hardware criteria by 24%, 8%, 6%, 30%, and 34%, on average. The CDM16_ffff and CDM16_fffb have the lowest power consumption, area, PDP, and PADP, respectively, but have the highest MRED among all CDM16s. Compared to the Exact16, they improved the hardware criteria along with the delay by 42%, 43%, 65%, 60%, and 40%, on average. CDM16_f880 and CDM16_fbb3 have the lowest delay among all CDM16s. These two multipliers have reduced the delay by 43% compared to Exact16 and have improved the power, area, PDP, and PADP by 30%, 30%, 60%, and 72%, on average. Also, among all CDM16s, CDM16_f880 and CDM16_fbb3 have the best PDEP and PADEP, respectively.

VIII. Comparison and Discussion

Table IV shows the hardware efficiency and accuracy criteria of some existing approximate unsigned 8-bit array multipliers in the literature that we intend to compare with the CDM8 multipliers. CDM8s have reduced the critical path delay, power consumption, and area by 14.3%, 22.8%, and 26.4% on average compared to the other approximate array multipliers in Table IV; CDM8s have also improved PDP, PADP, PDEP, and PADEP by 37.1%, 52.7%, 37.5%, and 64.1% respectively. Regarding the accuracy criteria, CDM8s have improved MRED by 17.6%. Therefore, as the results show, the CDM8s have improved both the accuracy and the hardware efficiency.

Regarding the power consumption and PDP, [30] obtained the best result, followed by the three proposed multipliers CDM8_aa, CDM8_a9, and CDM8_a8, respectively. [30] Compared to them have improved power consumption, PDP, and

TABLE IV: Hardware efficiency and accuracy criteria of the proposed 8-bit multipliers and comparable literature.

Proposed Method	Hardware efficiency criteria							Accuracy criteria				
	Area(m^2)	Power(W)	Delay(nS)	PDP	PADP	PDEP	PADEP	MED	NMED	MRED	NoEB	PC (%)
Exact8 *	300.6	85.61	0.76	65.064	19558	0	0	0	0	0	16	100
CDM8_44	257.7	76.777	0.65	49.905	12860	0.653	168.344	97.75	0.00150	0.0133	8.49	52.9
CDM8_51	269.2	78.071	0.641	50.043	13472	0.195	52.405	14.25	0.00022	0.0039	11.29	59.57
CDM8_62	253.2	74.851	0.584	43.713	11068	0.349	88.434	35.25	0.00054	0.0081	10.10	51.1
CDM8_73	237.5	70.826	0.563	39.875	9470	0.644	152.946	89.25	0.00137	0.0164	8.87	42.52
CDM8_74	227.7	66.27	0.529	35.057	7982	0.834	189.982	157.25	0.00242	0.0241	8.08	37.09
CDM8_84	216.3	62.704	0.522	32.731	7080	1.030	222.873	225.25	0.00346	0.0319	7.63	33.58
CDM8_95	200.6	58.337	**0.486**	28.352	5687	1.448	290.511	441.25	0.00678	0.0518	6.66	29.05
CDM8_a6	193.1	55.601	0.502	27.912	5390	2.108	407.034	777.25	0.01195	0.0766	5.81	26.40
CDM8_a7	183.8	52.056	0.502	26.132	4803	2.793	513.305	1321.25	0.02032	0.1084	5.00	24.63
CDM8_a8	171	48.724	0.502	24.459	4182	3.731	638.008	2409.25	0.03705	0.1548	4.09	23.06
CDM8_a9	166.2	47.57	0.505	24.023	3992	4.581	761.388	3689.25	0.05673	0.1936	3.36	22.43
CDM8_aa	**156.1**	41.548	0.548	22.768	**3554**	4.809	750.668	4713.25	0.07248	0.2145	2.85	22.26
Other approximate 8-bit array multipliers in the literature												
[14]	278.7	82.5	0.65	53.6	14955	**0.096**	**26.77**	**5.75**	**0.00008**	**0.0018**	**12.42**	**67.58**
[15]	301.6	143.5	0.77	110.5	33340	3.13	943.52	397.95	0.00612	0.0283	NR†	30.27
[30]	217.3	**40**	0.53	**21.2**	4607	1.71	371.76	578.72	0.0089	0.0807	NR†	NR†
[25]	349.3	50.6	0.58	29.3	10251	7.42	2590.49	1664.64	0.0256	0.2527	NR†	NR†

* Exact unsigned 8-bit multiplier (using two exact 8×4 multipliers).

† Not Reported.

TABLE V: Hardware efficiency and accuracy criteria of the proposed 16-bit multipliers.

Method	Hardware efficiency criteria							Accuracy criteria				
	Area(m^2)	Power(W)	Delay(nS)	PDP	PADP	PDEP	PADEP	MED	NMED	MRED	NoEB	PC (%)
Exact16 *	1348.4	501.4	1.35	677	912718	0	0	0	0	0	32	100
CDM16_0000	1318.6	479.9	1.03	494	651714	4.84	6390	**8222699**	**0.001914**	**0.009806**	**7.98**	**19.61**
CDM16_3000	1298.9	473.3	1.02	486	631316	4.76	6190	8222699	0.001914	0.009806	7.98	13.85
CDM16_7000	1237.7	453.1	1.02	464	575408	4.55	5642	8222699	0.001914	0.009806	7.98	8.19
CDM16_8000	1225.2	448.2	1.02	459	563364	4.50	5524	8222699	0.001914	0.009806	7.98	7.55
CDM16_8330	1183.9	436.9	0.94	411	486725	4.03	4777	8222766	0.001914	0.009815	7.98	5.12
CDM16_b330	1155.5	427.3	0.94	402	464574	3.94	4560	8222766	0.001914	0.009816	7.98	4.49
CDM16_f770	1022.5	375.9	0.79	300	306769	2.99	3059	8223980	0.001914	0.009974	7.98	2.04
CDM16_f880	990.6	365.1	**0.77**	284	281707	**2.86**	2842	8225459	0.001915	0.010089	7.98	1.81
CDM16_f883	970.6	359.3	0.78	280	272696	3.08	2993	8609497	0.002	0.010976	7.95	1.63
CDM16_fbb3	905.7	335.1	**0.77**	259	234914	2.97	**2692**	8610775	0.002	0.011462	7.95	1.33
CDM16_fff7	825.1	306.0	0.81	249	205814	6.69	5525	23076674	0.005373	0.026846	6.66	1.07
CDM16_fff8	809.2	301.8	0.81	245	199047	8.70	7047	36831042	0.008575	0.035407	5.92	1.04
CDM16_fffb	773.3	**289.0**	0.81	**235**	182165	18.44	14262	156630850	0.036469	0.078296	3.59	1.00
CDM16_ffff	**767.9**	**289.0**	0.81	**235**	**180910**	25.39	19501	307625794	0.071626	0.107794	2.35	0.99

* Exact 16-bit multiplier (using four exact 8-bit multipliers).

MRED by 12.8%, 10.7%, and 55.9%, respectively. However, those three proposed multipliers are the best in terms of area and PADP. They are also among the best in terms of delay. Hence, compared to [30], they have improved by 24.3%, 15.1%, and 2.3%, respectively. Regarding critical path delay, CDM8_95 is the best, which is 8.3% better compared to [30]. Also, CDM8_95 has a higher accuracy than [30], which improved it by 35.8%. On the other hand, most multipliers [14], [15], and [25] have the highest delay, power consumption, area, PDP, and PADP; Hence, compared to [14], [15], and [25], the three mentioned proposed multipliers are 22.7%, 50.2%, 46.9%, 63.2%, and 80% better, respectively.

Regarding the MRED, PC, PDEP, and PADEP, the [14] has obtained the best results, followed by the four proposed multipliers CDM8_51, CDM8_62, CDM8_73, and CDM8_44. [14] compared to them, improved MRED, PC, PDEP, and PADEP by 82%, 23.8%, 79.1%, and 76.8%, respectively, which shows an insignificant error of [14]. Nevertheless, those four proposed multipliers have more acceptable results in power consumption, delay, area, PDP, and PADP, which have improved by 9%, 8%, 9%, 15%, and 22%, respectively, compared to [14].

In the following, we intend to compare the CDM8s with other approximate multipliers with different architectures. We selected about 80 approximate multipliers of recent years [15], [17]–[19], [21]–[25], [29]–[45]. All these existing approximate multipliers are 8-bit, unsigned, and synthesized under 45 nm NanGate technology by reference papers. The architecture of the selected multipliers is different, so that we can divide them into four types: compressor-based multipliers [19], [21], [22], [31]–[33], [36], [37], [40]–[44], array multipliers [14], [15], [22]–[25], [30], logarithmic multipliers [29], [38], [39] and operand truncation-based multipliers [17], [18], [29], [34], [35], [39].

Figure 6 shows the power consumption, area, delay, and PDP of approximate multipliers in terms of MRED. Also, for each plot, we delineated the Pareto front to show the designs with the highest efficiency. Hence, the four proposed mul-

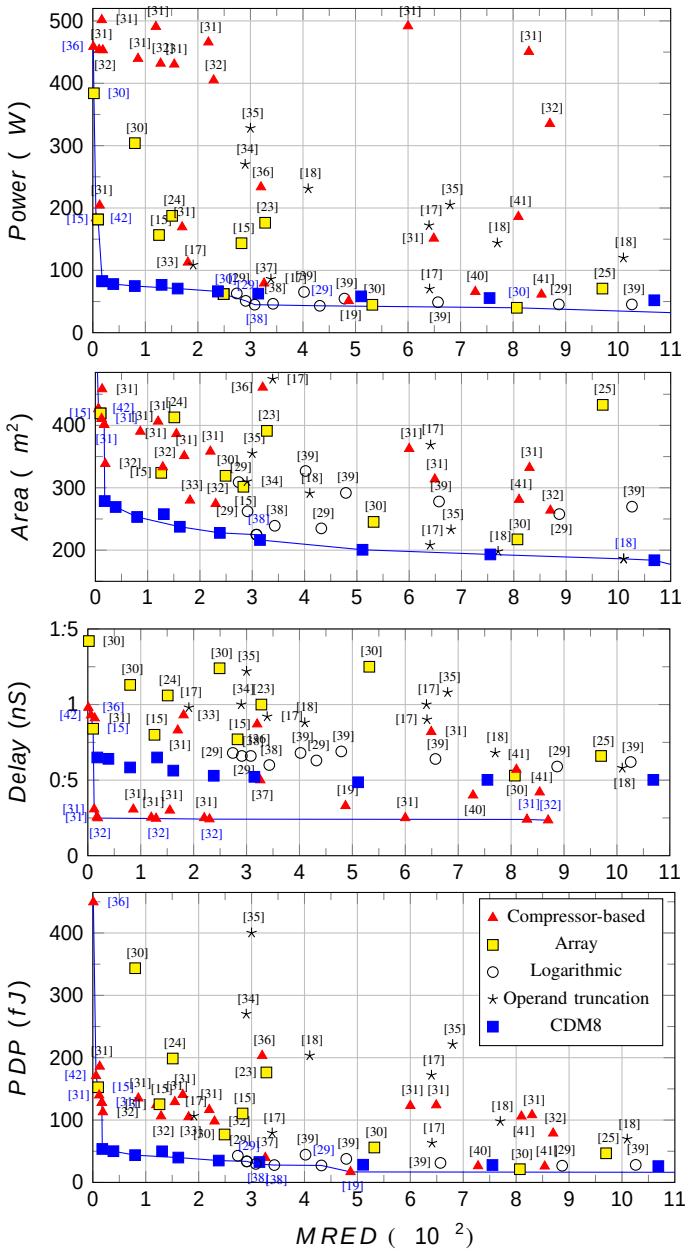


Fig. 6: Comparison of power consumption, area, delay, and PDP versus MRED for the approximate 8-bit multipliers.

tipliers, CDM8_51, CDM8_62, CDM8_73, and CDM8_74, are placed on the Pareto front of the power consumption. Regarding the area, the CDM8_51, CDM8_62, CDM8_73, CDM8_74, CDM8_84, CDM8_95, CDM8_a6, and CDM8_a7 are placed on the Pareto front. Regarding the delay, we see that compressor-based multipliers have the lowest delay and are placed on the Pareto front of critical path delay. Nevertheless, it should be noted that the CDM8s have the lowest delay after the compressor-based multipliers. Compared to them, CDM8s have far less power, area, and PDP, and they are among the best designs in terms of this criteria. Also, in terms of PDP, the CDM8_51, CDM8_62, CDM8_73, and CDM8_74, are the most optimal designs and are placed on the Pareto front of PDP. Therefore, we can conclude that the CDM8s, whether compared to array multipliers or other conventional architectures, are either the best or part of the best in many evaluation criteria; hence, CDM8s have a better balance.

TABLE VI: Gaussian smoothing 3 3 kernel [46]

Original			Modified		
0.095	0.118	0.095	97	121	97
0.118	0.148	0.118	121	151	121
0.095	0.118	0.095	97	121	97

TABLE VII: Performance of 8 8 approximate multipliers in Gaussian smoothing

	Design	SSIM (%)	PSNR (dB)
[14]	CDM8_40	99.99	63.04
Proposed	CDM8_44	99.89	52.24
	CDM8_51	99.98	60.11
	CDM8_62	99.95	56.66
	CDM8_73	99.89	52.95
	CDM8_74	99.86	51.38
	CDM8_84	99.73	48.02
	CDM8_95	99.07	41.52
	CDM8_a6	98.40	37.27
	CDM8_a7	94.46	30.78
	CDM8_a8	81.60	22.38
	CDM8_a9	67.15	15.77
	CDM8_aa	67.15	15.77
[28]	N8-L1	97.85	41.70
	N8-L2	97.66	39.50
	N8-5	97.98	43.00
	N8-6	97.98	43.00
[47]*	Ax8_1	97.96	43.00
	Ax8_2	97.85	39.20
	Ax8_3	97.25	35.60
[48]*	AxRM1	97.97	43.00
	AxRM2	97.90	41.50
	AxRM3	97.85	41.20
[49]	SSM_m4	94.39	26.80
	SSM_m4_u3	96.41	38.90
[50]*	DT2	97.67	42.31
	DT4	97.67	42.31
	DT8	97.37	35.61

* Reported by [28].

IX. Case Study: Image Processing

One of the most frequently considered error-resistant applications is image processing, and several papers test their suggested circuits in this setting. This paper assesses the use of image blurring in image processing. This application assists in clarifying the scope of applicability for the proposed designs. Low pass filtering in image processing generates image smoothing, which eliminates the abrupt spatial changes in the image. The low-pass filter alters a sliding kernel, which investigates each pixel individually concerning neighboring pixels. Each pixel must be processed via several multiplications, the number of which depends on the kernel size. The weighted average of the adjacent pixels serves as the affected pixel's actual value. Additionally, since the human eye cannot see insignificant fragments, image blurring is an application that can tolerate errors.

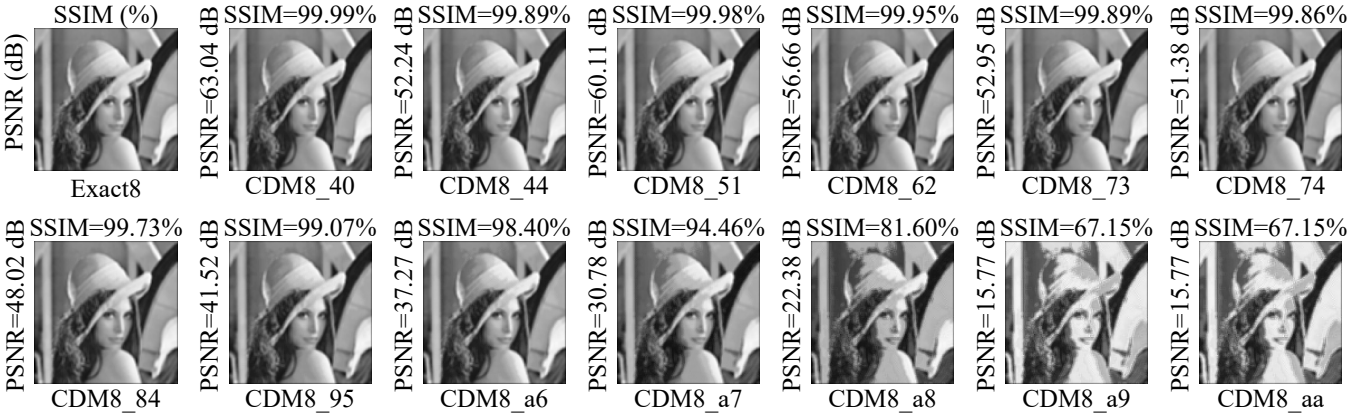


Fig. 7: Gaussian smoothing of images obtained with proposed 8-bit multipliers.

The aim of this study is to investigate the impact of approximating more partial product units (0s) on the performance of Gaussian smoothing. The performance is measured by the Peak Signal-to-Noise Ratio (PSNR) and the Structural Similarity Index Measure (SSIM). This study focuses on comparing the performance of the proposed 8-bit approximate multipliers. We take the output image of the exact 8-bit multiplier, in which Group A and Group B have exact partial product units (0), as the baseline in our comparison. Therefore, PSNR and SSIM values of the output images of proposed 8-bit approximate multipliers are calculated regarding the output image of the exact 8-bit multiplier.

A. Experimental Setup

In this paper, a two-dimensional rotationally symmetric 3 3 Gaussian low-pass filter with a standard deviation of 1.5 operates as the kernel considered for image smoothing, similar to [21]. The kernel's floating-point values are rounded after being multiplied by 2^{10}. In this fashion, the kernel values are appropriate for the 8-bit input multipliers.

As in [46], the initial and revised kernels are presented in Table VI. A Gaussian smoothing filter involving the proposed multipliers has been performed to blur a test image. The resultant images are presented in Figure 7. The same processing has also been conducted utilizing an exact multiplier to compare proposed designs adequately.

The precise equivalent of proposed multipliers as well as exact multiplier were written in Python. These codes have the same accuracy criteria as Table IV. These codes are embedded into the multiplication process of Gaussian smoothing where the 8-bit 128 128 input image incorporates convolution with the modified kernel presented in Table VI.

B. Metrics

Quantitative measurement of each proposed multiplier's competence in image smoothing is represented by the SSIM and PSNR. When evaluating the quality of images in image processing, it is important to consider both the PSNR and the SSIM.

While PSNR is a commonly used metric for image quality assessment, it has some limitations. Specifically, PSNR is sensitive to small changes in pixel values and may not accurately reflect the perceived quality of an image. This is because PSNR only considers the mean squared error between the original and reconstructed images, without taking into account the structural similarity between the images.

On the other hand, SSIM takes into account the structural information of the image, by comparing local patterns of pixel intensities rather than just the overall pixel values. As a result, SSIM is a more perceptually relevant measure of image quality. It has been shown in several studies that SSIM correlates more closely with human perception of image quality than PSNR [51].

Therefore, by considering both PSNR and SSIM in image processing, we can obtain a more comprehensive evaluation of image quality. This can help ensure that the resulting images are not only technically accurate but also visually appealing to the human eye.

C. Results

Figure 7 shows the output images of Gaussian smoothing utilizing proposed 8-bit multipliers. Except for CDM8_a8, CDM8_a9, and CDM8_aa, the other multipliers have acceptable PSNR and SSIM over 30 dB and 94%, respectively. Consequently, CDM8_40 [14] to CDM8_a7 show their capability in image processing applications.

Figure 7 indicate that increasing the number of approximated partial product units leads to a degradation in the PSNR and SSIM. CDM8_44, which approximates four columns of the least significant bits of both Group A and Group B, shows more approximation compared to CDM8_51, resulting in a slightly worse performance in the case study. CDM8_73, which has more carry-disregarded columns in Group A but approximates three columns in Group B, shows the closest performance to CDM8_44. The study infers that Group B approximation primarily contributes to the accuracy and performance degradation.

The study shows that as the number of approximated columns in both Group A and Group B increases, the performance degradation is more significant. However, the degra-

dation is subtle from CDM8_44 to CDM8_95. The study concludes that with more hardware efficiency (see Table IV), it is possible to maintain acceptable performance in Gaussian smoothing, achieving over 41 dB and 99% of PSNR and SSIM, respectively. CDM8_a6 and CDM8_a7 maintain an acceptable performance level in the case study, with slightly noticeable performance drops. Since all columns of Group A for both proposed approximate multipliers are approximated, their performance in the study is slightly dropped, yet with PSNR and SSIM of over 30 dB and 94%, respectively.

In summary, the study demonstrates that increasing the number of approximated partial product units affects the performance of proposed 8-bit approximate multipliers in Gaussian smoothing, particularly in Group B approximation. However, with more hardware efficiency, it is possible to maintain acceptable performance in Gaussian smoothing even with a higher number of approximated partial product units e.g. using CDM8_73 instead of CDM8_44 which experiences more hardware efficiency with similar performance in our case study. In comparison with other 8-bit approximate multipliers in the literature with same case study, the proposed 8-bit approximate multipliers CDM8_44 to CDM8_84 maintain more PSNR and SSIM as shown in Table VII.

X. Conclusion

This paper proposes a methodology for designing approximate array multipliers based on carry disregard. Carries can be ignored in various ways, and each method leads to different results regarding the criteria of accuracy and hardware. As shown by our application case-study, a smaller number of carry disregard, does not necessarily lead to a more accurate and better performing multiplier. In other words, by judiciously selection the location of carry disregard, it is possible to disregard a larger number of carries (and thus gain better speed and smaller area) while gaining a better performance in terms of overall accuracy and suitability for the application. Thus, the essential point is to choose a suitable way to disregard the carries, which depends on the architecture of multipliers. Our study also shows that the absolute value of approximation metrics for the multiplier, does not necessarily predicts its performance in the end application entirely accurate, even though it is a good general indicator and a ball-park estimator. this method simplifies computing units and reduces hardware complexity. Therefore, it causes a significant improvement in hardware efficiency criteria. Compared to the exact multiplier, the proposed 8-bit approximate multipliers have improved critical path delay, power consumption, and area by 29%, 29%, and 30% on average. Also, compared to the existing approximate array architectures in the literature, they have reduced the delay, power consumption, and area by 14.3%, 22.8%, and 26.4% on average. The proposed 16-bit approximate multipliers have improved critical path delay, power consumption, and area by 35%, 24%, and 23% compared to the exact multiplier. The proposed multipliers generally have different accuracy levels, creating an acceptable and better balance between hardware efficiency and accuracy criteria. The proposed designs are based on conventional CMOS technology, while nowadays, we witness the emergence of new technologies such as spin-CMOS and memristive in-memory computing. It is expected that the difference in the basic technology will lead to different results. Within the same technology, CMOS in our case, the use of smaller-scale technologies brings about better results. We conducted image processing utilizing proposed 8-bit multipliers and demonstrated their PSNR and SSIM. Most of them are applicable in image blurring and have PSNR and SSIM over 30 dB and 94%, respectively.

For future work, we plan to use multi-operand approximate adders like compressors in the process of PP reduction. Hence, the compatibility of our proposed methodology with these types of adders gives us the idea of combining them, and we predict that we will achieve better results. In addition, as for the final summation step in the process of multiplication, there are various types of adders, such as CSA and Parallel Prefix Adder (PPA), which we intend to investigate their effect. On the other hand, different applications have different error tolerances, so dynamically adjusting the accuracy level of the proposed multipliers in various applications is one of our future works.

References

[1] W. Liu *et al.* A retrospective and prospective view of approximate computing [point of view]. *Proceedings of the IEEE*, 108(3), 2020.

[2] P. Schober *et al.* Sound source localization using stochastic computing. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery.

[3] M. L. Pasini and M. P. Laiu. Anderson acceleration with approximate calculations: applications to scientific computing, 2022.

[4] N. TaheriNejad and S. Shakibhamedan. Energy-aware adaptive approximate computing for deep learning applications. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 328–328, 2022.

[5] P. Schober *et al.* Stochastic computing design and implementation of a sound source localization system. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(1):295–311, 2023.

[6] H. Anzt *et al. Approximate Computing for Scientific Applications*, pp. 415–465. Springer International Publishing, 2022.

[7] S. E. Fatemieh *et al.* Approximate in-memory computing using memristive imply logic and its application to image processing. In *IEEE ISCAS*, pp. 1–5, 2022.

[8] C. Ossimitz and N. TaheriNejad. A fast line segment detector using approximate computing. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2021.

[9] S. E. Fatemieh *et al.* Fast and compact serial imply-based approximate full adders applied in image processing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(1):175–188, 2023.

[10] H. Jiang *et al.* Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE*, 108(12), 2020.

[11] Q. Xu *et al.* Approximate computing: A survey. *IEEE Design & Test*, 33(1):8–22, 2016.

[12] A. S. Baroughi *et al.* Axe: An approximate-exact multi-processor system-on-chip platform. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pp. 60–66, 2022.

[13] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4), mar 2016.

[14] N. Amirafshar *et al.* An approximate carry disregard multiplier with improved mean relative error distance and probability of correctness. In *Euromicro Conference on Digital Systems Design (DSD)*, pp. 1–7, 2022.

[15] H. Waris *et al.* Hybrid partial product-based high-performance approximate recursive multipliers. *IEEE Trans. Emerg. Topics Comput.*, 10(1):507–513, 2022.

[16] H. Jiang *et al.* A review, classification, and comparative evaluation of approximate arithmetic circuits. *ACM JTEC*, 13(4), 2017.

[17] S. Hashemi *et al.* Drum: A dynamic range unbiased multiplier for approximate applications. In *IEEE/ACM ICCAD*, pp. 418–425, 2015.

[18] S. Vahdat *et al.* Tosam: An energy-efficient truncation- and rounding-based scalable approximate multiplier. *IEEE TVLSI*, 27(5), 2019.