Analog Integrated Circuits and Signal Processing (2025) 124:53
https://doi.org/10.1007/s10470-025-02459-0

Design space exploration of array-based approximate squaring unit for error-tolerant computing

Mahmoud Masadeh[1] · Alain Aoun[2] · Sofiène Tahar[2]

Received: 7 April 2025 / Revised: 30 May 2025 / Accepted: 24 June 2025
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2025

Abstract
The squaring circuit is an essential computational element of Digital Signal Processing (DSP) designs that directly affect their area, speed and power consumption. Various DSP applications have noisy and redundant input data. Thus, implementing an approximate squaring function will cause minor quality degradation with a significant reduction in hardware costs. In this paper, we perform a design space exploration (DSE) of an energy-efficient array-based approximate squaring function. The proposed designs are 8-bit unsigned and signed, with reduced area, power, and delay. Towards this goal, we introduced four energy-efficient approximate **I**ne**x**act **F**ull **A**dders (IxFAs) that are suitable for the squaring function. The proposed IxFAs and 14 existing approximate full adders (FAs) are used to perform a DSE of approximate squaring units with various configurations based on the type of the used approximate FAs and the level of approximation. The IxFA-based squaring designs have a reduced area, power, and delay compared to the exact array squarer. Moreover, compared to the state-of-the-art, the proposed designs have less area, energy, and power consumption while offering competitive quality. They were further tested for DSP applications and showed high-quality results.

Keywords Approximate computing · Approximate full adder · Design space exploration · Squaring unit · Array design · Energy efficiency

1 Introduction

The exceptionally large number of battery-powered Internet of Things (IoT) sensors perform data acquisition, which provides noisy input data for real-time digital signal processing (DSP) applications. Such battery-powered sensors should have a reduced footprint, small delay, and reduced power consumption, where a degraded accuracy of the final results is acceptable [1]. DSP applications efficiently manipulate noisy inputs. Thus, exact results could be sacrificed for fast calculations with reduced power consumption [2]. Error-tolerant computing refers to the ability of the system to continue functioning when one of its components has an inexact output. Thus, an error-tolerant system contains approximate components without significant accuracy loss of the result.

Arithmetic modules define the performance of computing systems. The basic operations used to execute DSP algorithms are addition/subtraction, multiplication, squaring, and shifting [3]. Thus, improving the design of these operations will enhance the final performance of DSP processors. While *squaring function* is a widely-used arithmetic operation in multimedia and DSP applications [4], its implementation can be resource-intensive, especially when high accuracy and speed are required. The hardware design of the squaring units is simpler than the design of general-purpose multipliers because they need half of the partial products. Thus, squaring units are generally faster and has higher throughput as they are used extensively in various applications [5], e.g., Viterbi algorithm, object tracking, vector quantization and equalization, adaptive filtering, gradient adaptation, pattern recognition, image compression,

✉ Mahmoud Masadeh
mahmoud.s@yu.edu.jo

Alain Aoun
a_alain@encs.concordia.ca

Sofiène Tahar
tahar@encs.concordia.ca

1 Computer Engineering Department, Yarmouk University, Irbid, Jordan

2 Electrical and Computer Engineering, Concordia University, Montréal, QC, Canada

Published online: 07 July 2025

cryptography, Euclidean branch calculation, and motion estimation [6].

General-purpose processors can perform DSP operations employing fundamental computing blocks. However, their area, latency and power usage might not satisfy the requirements of battery-powered real-time applications, e.g., IoT devices. Thus, dedicated circuits with reduced area, delay, and power consumption are desirable [7].

DSP units with a parallel architecture are appropriate for high-performance applications, i.e., notepads and laptops [8, 9]. However, they have a large design area and power consumption. On the other hand, DSP units with a recursive architecture are suitable for small and low-power IoT nodes, which often have limited memory storage and operand size. Recursive DSP units [4], which we target in this work, are embedded within various digital systems used in our daily lives. For example, 8-bit AVR (**A**lf and **V**egard's **R**ISC processor) microcontrollers, which are a family of microcontrollers developed since 1996 by Atmel, are widely used for various types of IoT devices [10]. Thus, energy-efficient designs can have a significant impact on the power consumption of the overall system.

Applications, such as image processing, audio signal processing, and machine learning and pattern recognition, can tolerate inaccuracies in their results. Thus, exact squaring units are unnecessary. Using approximate squaring functions, which have a simplified design with reduced area, less power consumption, and lower latency, will have significant benefits in terms of hardware efficiency, power consumption, and execution speed of DSP units. This paper explains how an array-based exact squaring function can be approximated systematically. For this aim, we propose four new energy-efficient approximate full adders (FAs) suitable for the squaring function. Then, use them with an already existing 14 FAs and perform a design space exploration for *array-based approximate squaring designs*. We achieved fast, compact, low-power, and energy-efficient designs which fit a set of applications. Introducing approximation into hardware design reduces its critical path, area, and power consumption. Thus, energy efficiency is improved for acceptable accuracy degradation. The proposed array-based approximate squaring design is intended for integer arithmetic. However, it applies to floating-point (FP) operations represented in the IEEE-754 standard. The FP number's sign and exponent are calculated accurately, while themantissa is evaluated approximately based on the proposed design. The *contributions* of this article are as follows:

1. Explain how an array-based exact squaring function can be approximated systematically.
2. Propose four energy-efficient approximate FAs (IxFAs) and evaluate their design characteristics, i.e., accuracy, area, power, and delay.
3. Utilize the four proposed IxFAs and 14 existing approximate FAs and perform a design space exploration (DSE) of approximate squaring units with various configurations based on the type of the FAs and the level of approximation.
4. Conduct an evaluation of the proposed squaring designs based on popular DSP applications.

The rest of this article is organized as follows. Section 2 explains the design of an exact array multiplier and its simplification into exact array squarer. Section 3 introduces related work on designing approximate squaring circuits. Section 4 proposes four new FAs that are suitable for the approximation of the squaring designs. Section 5 explains the proposed squaring designs and discusses their accuracy and resource usage. Section 6 compares the proposed approximate squaring units with state-of-the-art approximate squarers. The performance of the proposed squarers is analyzed with a real-time application in Section 7. Finally, Section 8 concludes the paper with some future directions.

2 Accurate array multiplier and squaring circuit

We consider DSP functions in microcontroller units (MCUs) as our target practical applications, where 8-bit designs are sufficient to represent the sensory collected data [5]. The proposed designs are easily expandable to 16 and 32 bits. Next, we describe an 8-bit exact array multiplier, and an 8-bit unsigned and signed exact squaring function where both designs are array-based multipliers with two identical inputs. The approximation procedure is generic and can be applied to any N × N array multiplier.

2.1 Accurate array multiplier

An n-bit array multiplier is composed of n^2 AND gates for partial products (PP) generation, and $n-1$ n-bit adders for partial products accumulation [7]. An array multiplier has a simple design and regular structure. Thus, it is suitable for the multiplication of small numbers, e.g., microcontrollers for IoT devices [11]. However, for the multiplication of large numbers, high-speed multipliers are required, or the array multipliers could be pipelined for fast operation.

Due to its importance and applicability in various applications, there are various designs of approximate array multipliers, where they replace the exact FAs with approximate FAs (e.g., [12–14]). They differ in the type of the used FAs,

							Multiplicand	A8	A7	A6	A5	A4	A3	A2	A1
							Multiplier	B8	B7	B6	B5	B4	B3	B2	B1
								B1A8	B1A7	B1A6	B1A5	B1A4	B1A3	B1A2	B1A1
							B2A8	B2A7	B2A6	B2A5	B2A4	B2A3	B2A2	B2A1	
						B3A8	B3A7	B3A6	B3A5	B3A4	B3A3	B3A2	B3A1		
					B4A8	B4A7	B4A6	B4A5	B4A4	B4A3	B4A2	B4A1			
				B5A8	B5A7	B5A6	B5A5	B5A4	B5A3	B5A2	B5A1				
			B6A8	B6A7	B6A6	B6A5	B6A4	B6A3	B6A2	B6A1					
		B7A8	B7A7	B7A6	B7A5	B7A4	B7A3	B7A2	B7A1						
	B8A8	B8A7	B8A6	B8A5	B8A4	B8A3	B8A2	B8A1							
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Fig. 1 Partial Products of an 8-bit Unsigned Array Multiplier

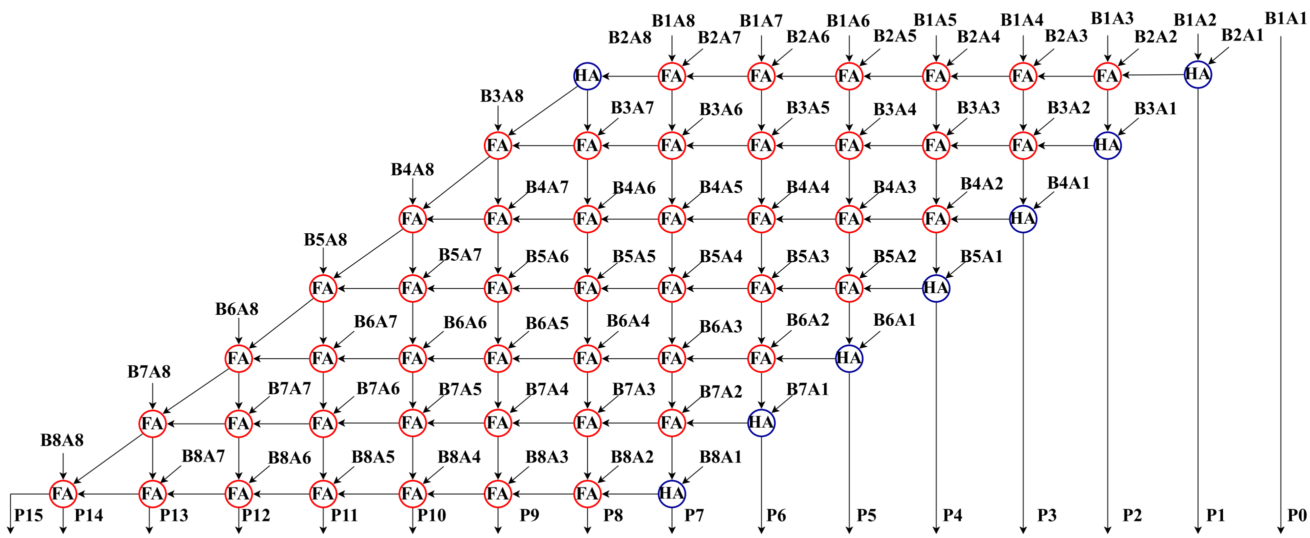


Fig. 2 Structure of Partial Product Accumulation and Summation for an 8-bit Unsigned Array Multiplier

the amount of approximation and approximation structure, i.e., replacing arrows or columns in the partial product array.

As shown in Fig. 1, an 8-bit array multiplier includes 64 partial products generated by 64 AND gates. A8 to A1 represent the "*multiplicand*", B8 to B1 represent the "*multiplier*", B1A1 denotes the first partial product, B8A8 represents the last partial product, and P15 to P0 represent the final "*product*" [15]. As shown in Fig. 2, partial product accumulation requires 7 carry-ripple adders each of size 8-bit. Thus, 56 full-adders (FAs) are required, where eight FAs are replaced by half-adders (HAs) since the third input is zero. The HAs are highlighted in blue color in Fig. 2.

2.2 Accurate unsigned array squaring circuit

As shown in Fig. 1, the partial products are symmetrical around the diagonal (from top-right to bottom-left). Based on that, appropriate folding techniques were introduced in [16] and developed in [17]. Moreover, the authors of [18] propose a new method that utilizes more regularity of merged PP to reduce more bits and the height of the PP tree. The authors of [19] also introduced a VLSI implementation of a 350MHz 0.35μm 8-bit merged squarer. They reduced the depth of the partial product array where they merged all PPs on the diagonal with those of the folded and shifted sub-arrays. However, the work in [16–19] used exact basic units.

Fig. 3 shows the simplified exact partial product tree of an *8-bit exact unsigned array squaring* circuit with 28 partial products generated by 28 AND gates which is greatly simplified compared to an 8-bit array multiplier. Moreover, Fig. 4 shows that 21 FAs and 7 HAs are required to generate the final "*product*", i.e., P15 to P0. As shown in Fig. 1, the height of the partial products array is N for an N-bit multiplier. However, the height of the squarer is reduced to $\lfloor N/2 \rfloor + 1$, which significantly decreases the hardware complexity and delay.

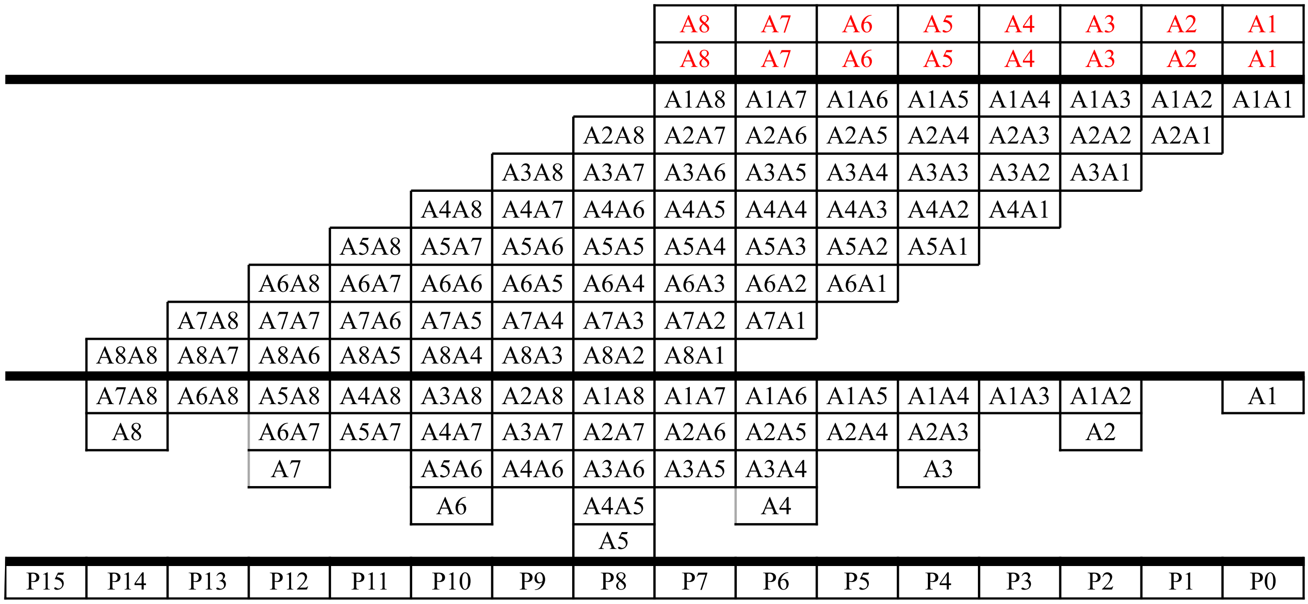


Fig. 3 Partial Products of an 8-bit Unsigned Array Squaring Circuit

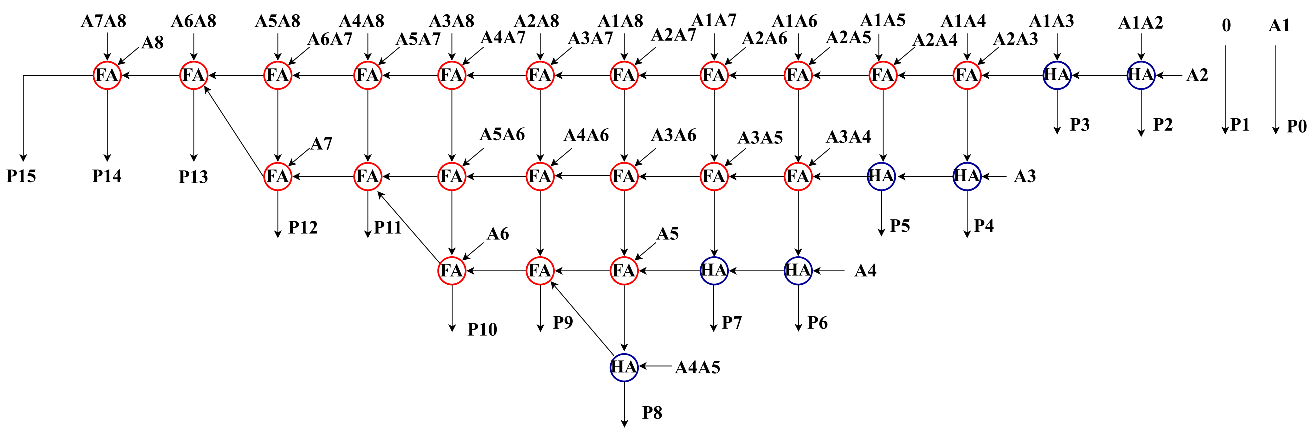


Fig. 4 Structure of Partial Product Accumulation and Summation for an 8-bit Unsigned Array Squaring Circuit

2.3 Accurate signed array squaring circuit

We modify the simplified structure of the unsigned squaring unit to support signed numbers. An 8-bit signed number encodes the magnitude using 7-bits only while the most significant bit (MSB), i.e., 8th bit, encodes the sign. Moreover, the square of a signed number is thesquare of its magnitude, i.e., $x^2 = |x|^2$. Hence, the tree of partial products can be further simplified. As shown in Fig. 5, the squaring of a 7-bit magnitude requires 21 AND gates for partial products generation. As shown in Fig. 6, partial product accumulation requires 15 FAs and 6 HAs for a 7-bit input, where the height of the tree is reduced by one level compared to the 8-bit unsigned design. Moreover, a hardware unit that extracts the magnitude of an 8-bit signed number must be added to the 7-bit squarer. For this purpose, we compute the 2's complement of the signed number to extract its magnitude. The MSB of the signed number indicates if the number is positive or negative. Thus, if the MSB is logic '1', i.e., the number is negative, a fast way of computing the 2's complement is to search for the first occurring logic '1' and then compute the inverse of all succeeding bits. Hence, the *i*-th bit of a negative number must be inverted if any of the preceding bits is logic '1' when the MSB is logic '1'. Subsequently, we propose the usage of Eq. (1) to compute the *magnitude* of a signed number (A) where we look for any occurring logic '1' in the preceding bits along with the MSB to identify if the bit A_i must be inverted. The *i*-th bit magnitude representation of an *n*-bit signed number is computed using XNOR, NAND and OR gates. Fig. 7 depicts the schematic of the computation in Eq. (1) where the hardware requires $n-2$, $n-1$ and $n-1$, OR, NAND and XNOR

						A8	A7	A6	A5	A4	A3	A2	A1
						A8	A7	A6	A5	A4	A3	A2	A1
							A1A7	A1A6	A1A5	A1A4	A1A3	A1A2	A1A1
						A2A7	A2A6	A2A5	A2A4	A2A3	A2A2	A2A1	
					A3A7	A3A6	A3A5	A3A4	A3A3	A3A2	A3A1		
				A4A7	A4A6	A4A5	A4A4	A4A3	A4A2	A4A1			
			A5A7	A5A6	A5A5	A5A4	A5A3	A5A2	A5A1				
		A6A7	A6A6	A6A5	A6A4	A6A3	A6A2	A6A1					
	A7A7	A7A6	A7A5	A7A4	A7A3	A7A2	A7A1						
	A7A6	A7A5	A7A4	A7A3	A7A2	A7A1	A6A1	A1A5	A1A4	A1A3	A1A2		A1
	A7		A6A5	A6A4	A6A3	A6A2	A5A2	A4A2	A2A3		A2		
			A6		A5A4	A5A3	A4A3		A3				
					A5		A4						
P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Fig. 5 Partial Products of a 7-bit Unsigned Array Squaring Circuit

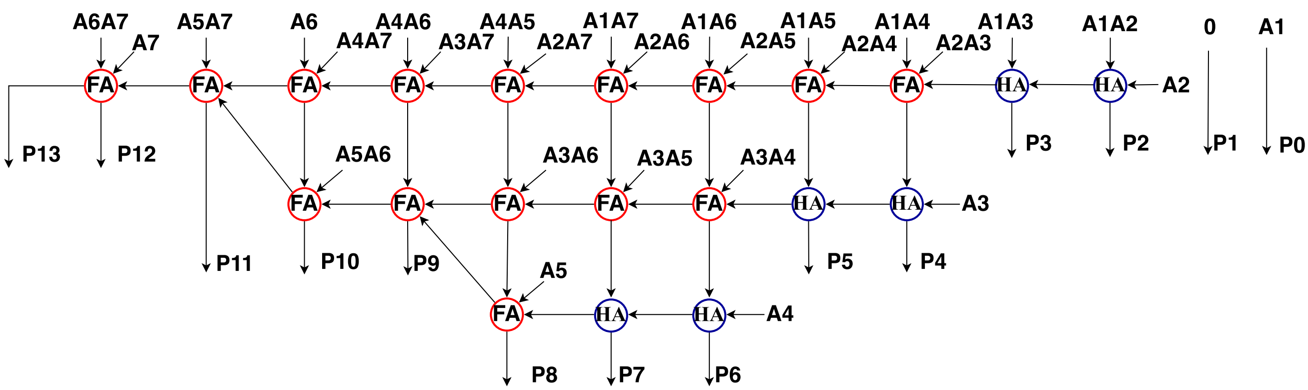
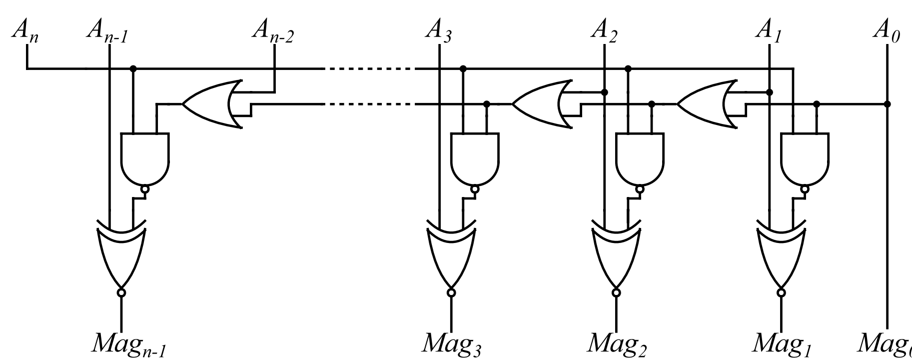


Fig. 6 Structure of Partial Product Accumulation and Summation for a 7-bit Unsigned Array Squaring Circuit

Fig. 7 Schematic of the 2's Complement Computation



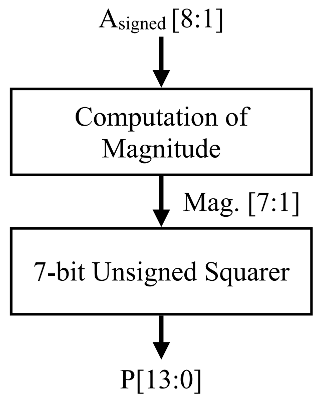


Fig. 8 Implementation of the 8-bit Signed Array Squaring Circuit

Table 1 Area, Power, and Delay for the Exact Array Multiplier and Exact Squaring Function

Design	Area (μm^2)	Power (μW)	Delay (ps)
Array Multiplier	363.89	14.14	1,060
Unsigned Squaring Unit	172.37	4.26	767
Signed Squaring Unit	152.53	5.79	698

gates, respectively. The implementation of the proposed 8-bit signed squarer is shown in Fig. 8.

$$Mag_i = \begin{cases} A_i; & \text{for } i = 1 \\ XNOR\{A_i, NAND[A_n, OR(A_0, ..., A_{i-1})]\}; & \text{for } 1 < i < n \end{cases} \quad (1)$$

2.4 Hardware metrics

We analyze various design metrics, i.e., area, power, and delay, to show the benefits of simplifying the design of squaring function, both signed and unsigned, compared to standard multiplication. We synthesized these designs using *Cadence Innovus* [20]. The synthesis is performed using a Cadence Generic Process Design Kit (GPDK) based on the 45nm CMOS technology node. Table 1 shows the characteristics of the 8-bit exact array multiplier and squaring designs. The unsigned squarer unit offers a reduction of 52.6%, 69.9%, and 26.7% in area, power, and delay respectively, compared to the array multiplier. Similarly, the proposed signed squarer has an average area, power, and delay reduction of 58.1%, 59.1%, and 34.2%, respectively, compared to the array multiplier. The characteristics of the designed approximate squares are given in Section 5.2.

3 Related work

In this section, we include a short literature review on some of the relevant works of the squaring circuit, which is a unary operation with a single input. Previous work on the approximation of the squaring function is based on various techniques including lookup table-based solutions [21, 22], linear interpolation of the squaring function [23], simplification of squaring algorithm [24], fixed width operation [12], and simplified combinational logic for logarithmic design [25] [26]. Another technique of developing approximate designs, while having full-bit width results, is utilizing approximate building blocks. Accordingly, different designs of approximate arithmetic units have been investigated, e.g., [4, 7, 27, 28]

Traditional ROM-based look-up tables (LUT) are restricted to applications with small bit-size operands. For applications with large bit-size operands, the increase in the look-up table size is preventive [21]. For instance, the authors of [22] proposed a hybridLUT-based architecture for the low-error and efficient fixed-width squarer circuits, employing both LUT-based and simple conventional logic circuits. Thus, more area is added which is inadequate for IoT applications. In [23], the authors presented a linear approximation for the squaring function. They used shift, concatenation and addition operations instead of using multipliers and LUTs. Moreover, some compensation techniques were used to reduce the maximum relative errors.

The authors of [24] presented two simple combinational logic designs for bit-parallel approximate squarers, which are based on simplified combinational logic. The obtained hardware elements grow linearly with the width of input bits. However, the proposed designs are mainly suitable for LUT-based field-programmable gate array (FPGA) implementations only. In a different approach, the authors of [12] proposed a model for array-based approximate arithmetic computing, i.e., multiplier and squarer design with fixed-width operands. Using the fixed-width operation, the output is N bits for an input of N bits, where the N least significant bits (LSB) of the result are truncated. Thus, an Error Compensation Unit (ECU) is identified as a key building block with extra design overhead.

The authors of [25] proposed an approximate squaring design utilizing simple algorithmic interpolation, where the proposed design is based on shift and subtract operations. Similarly, in [26], the authors proposed a low-error approximate logarithmic squaring function with low power. However, when compared with the Mitchell algorithm [29], which is the baseline method for most logarithmic multipliers and logarithmic squaring functions, both [25] and [26] have higher power and area consumption. Regarding their accuracy, the design proposed in [26] has the lowest average error compared to logarithmic designs. Moreover, when an error compensation module is added to [25] it gives the lowest mean relative error distance (MRED) with a noticeable increase in its area, power, and delay. Thus, there is no algorithmic design better than [29] in all metrics.

The authors of [30] proposed an approximate partial products generator for squarer (APPGS). Then, presented three designs of approximate **radix-4 Booth squarer** using their proposed APPGS, which are called *ABS1*, *ABS2*, and *ABS3*. *ABS1* is designed based on approximate partial products (PPs) which are generated at the lowest *r* significant columns of the partial product matrix. The remaining PPs are generated accurately. Then, all of these PPs are accumulated with exact adders and compressors. *ABS2* is similar to *ABS1* except that all PPs are accumulated using approximate arithmetic blocks, i.e., half adders, full adders, and compressors, with an input signal rearrangement technique to reduce the error. *ABS3* is generated by adding an error recovery module to *ABS2* to enhance its output quality. However, the precision of their designs should be further improved. This work has three configurations based on different building blocks and different amounts of approximation [30]. More recently, the work in [31] offered a similar approach for approximating the **Booth squarer**, where the authors proposed approximation at two levels, namely, PP generation and the compressor. Using the proposed approximate PP and approximate compressor, the authors proposed two approximate Booth folding squarer (*ABFS*), i.e., *ABFS1* and *ABFS2*. However, the work in [31] does not provide many insights on the techniques used to achieve the approximation and lacks extensive quality analysis.

Due to functional similarities between our work and the above-related research, in Section 6, we will compare our results with those presented in [30, 31] in terms of accuracy. Moreover, we provide a comparison in terms of area, power, and delay.

Most of the related work has drawbacks such as the need for large memory storage, the usage of an error correction unit with significant overhead, targeting FPGA only, or the inability to achieve significant area, power, and delay reduction when implemented on ASIC. Thus, in this paper, we propose energy-efficient approximate squarers along with four approximate FAs that are most suitable for the squaring designs. The approximate squaring units are approximated with various configurations based on: (*i*) the type of the used approximate full adders, where we use 14 well-known FAs and four new FAs; and (*ii*) the level of approximation, i.e., the number of exact FAs replaced with their approximate counterparts.

4 Approximate full adders

Arithmetic units are the most power-hungry parts of hardware designs [32]. Therefore, hardware approximation reduces the power consumption of these arithmetic units [33]. Squaring designs are basic building blocks used many in arithmetic units. Array-based squaring units consist of a set of full adders controlled by carry generation and propagation. Generally, the most suitable technique of arithmetic units is replacing the exact FAs with their approximate counterparts.

We propose four power-efficient **I**ne**x**ac **F**ull **A**dders, i.e., IxFA1, IxFA2, IxFA3, and IxFA4, that are suitable for the structure of the squaring design. These designs are obtained by introducing more approximations into already existing designs. Therefore, they have a relatively high error rate and error distance. However, their design characteristics, i.e., area, power, and delay, would be competitive. The schematic diagrams for these designs are shown in Fig. 9a – 9d, respectively. Table 2 shows the truth tables of the IxFAs where the erroneous values of Cout and Sum are highlighted in bold.

We compare the proposed IxFAs with 14 related designs, namely, AFA [34], AMA1, AMA2, AMA3 [35], AXA1, AXA2, AXA3 [36], IFA [37], InXa1, InXa2, InXa3 [38], NxFA [39], and TGA1, TGA2 [40]. In the sequel, we refer to these FAs as **E**xisting Ine**x**act **F**ull **A**dders (ExFAs). Table 3 shows the error rates (ER) and error distances (ED) of

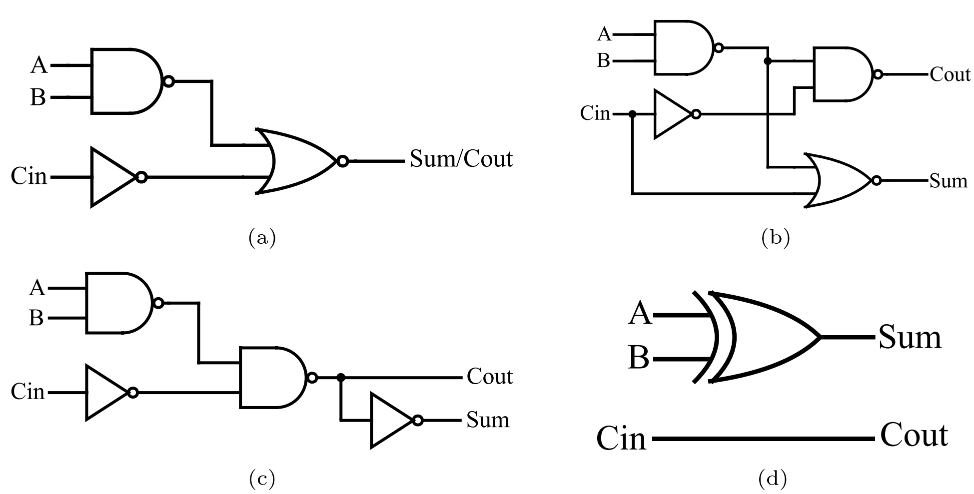


Fig. 9 Schematic of the Proposed **a**. IxFA1, **b.** IxFA2, **c**. IxFA3, and **d**. IxFA4

Table 2 Truth Table for IxFA1, IxFA2, IxFA3 and IxFA4

Inputs			IxFA1		IxFA2		IxFA3		IxFA4	
A	B	Cin	Cout	Sum	Cout	Sum	Cout	Sum	Cout	Sum
1	1	1	1	1	1	**0**	1	**0**	1	**0**
1	1	0	**0**	0	1	**1**	1	0	**0**	0
1	0	1	**0**	0	1	0	1	0	1	**1**
1	0	0	0	**0**	0	**0**	0	1	0	1
0	1	1	**0**	0	1	0	1	0	1	**1**
0	1	0	0	**0**	0	**0**	0	1	0	1
0	0	1	0	**0**	**1**	**0**	**1**	**0**	**1**	**0**
0	0	0	0	0	0	0	0	**1**	0	0

Table 3 Error Rates (ER) and Error Distances (ED) of various Approximate Full Adders (FAs)

Work	Design	ER	ED
[34]	**AFA**	0.25	0.25
[35]	**AMA1**	0.25	0.25
	AMA2	0.25	0.25
	AMA3	0.375	0.375
[36]	**AXA1**	0.5	0.5
	AXA2	0.5	0.5
	AXA3	0.25	0.25
[37]	**IFA**	0.25	0.25
[38]	**InXa1**	0.25	0.5
	InXa2	0.25	0.25
	InXa3	0.25	0.25
[39]	**NxFA**	0.375	0.375
[40]	**TGA1**	0.25	0.25
	TGA2	0.25	0.25
Proposed	**IxFA1**	0.75	1.125
	IxFA2	0.625	0.625
	IxFA3	0.375	0.375
	IxFA4	0.625	0.75

the above FAs. Among the proposed IxFAs, we notice that IxFA3 has the least ER and ED.

The hardware metrics, i.e., area, power, and delay, of the 18 FAs were measured. The synthesis is performed using the *Cadence Innovus* tool with 45nm GPDK process technology. Fig. 10 shows the area, power, delay, and power-area-delay-product (PADP) of the IxFAs compared to the ExFAs. As shown in Fig. 10a, the average area of the ExFAs is $4.935\mu m^2$ while the IxFAs have an average area of $2.651\mu m^2$. IxFA1, IxFA2, IxFA3 and IxFA4 have an **area** of $2.052\mu m^2$, $3.42\mu m^2$, $2.394\mu m^2$, and $2.736\mu m^2$, respectively. Thus, their area is competitive, i.e., less/better than the area of 13 ExFAs. The average **power** consumption is $79.82nW$ for all ExFAs and $26.98nW$ for the proposed designs. The proposed FAs, i.e., IxFA1, IxFA2, IxFA3, and IxFA4, have a power consumption of $12.64nW$, $40.66nW$, $20.95nW$, and $33.67nW$, respectively. Thus, their power consumption is less than 13 ExFAs. The average **delay** is $47.43fs$ for the 14 ExFAs and $24fs$ for the proposed ones. The delay of the IxFA1, IxFA2, IxFA3 and IxFA4 is $34fs$, $28fs$, $28fs$, and $6fs$, respectively. Thus, their delay is better than 12 ExFAs. The average **PADP** of the IxFAs is 1683.1 and 24338.2 for the 14 ExFAs.

The PADP of the proposed IxFAs is better than 13 ExFAs. We notice that the proposed IxFAs have a competitive area, power, delay, and PADP compared with 14 well-known FAs. On the other hand, as the quality and resource usage are more relevant at the application level, i.e., the squarer, in the sequel, we use all the 14 ExFAs along with the four proposed IxFAs. As such, we offer a comprehensive analysis of the behavior of the proposed approximate squarer.

5 Proposed squaring function

In this section, we illustrate various designs of an *8-bit approximate array squaring* (*AAS*) function. We propose to gradually replace the FAs of each column with approximate ones. For efficiency of reference, the vertical replacement of FAs, i.e., the level of approximation, is referred to as *V1* to *V11* as highlighted in Fig. 11.

We generated **154** *AAS* designs using the 14 well-known FAs, i.e., 14 × 11. Also, we generated **44** *AAS* designs

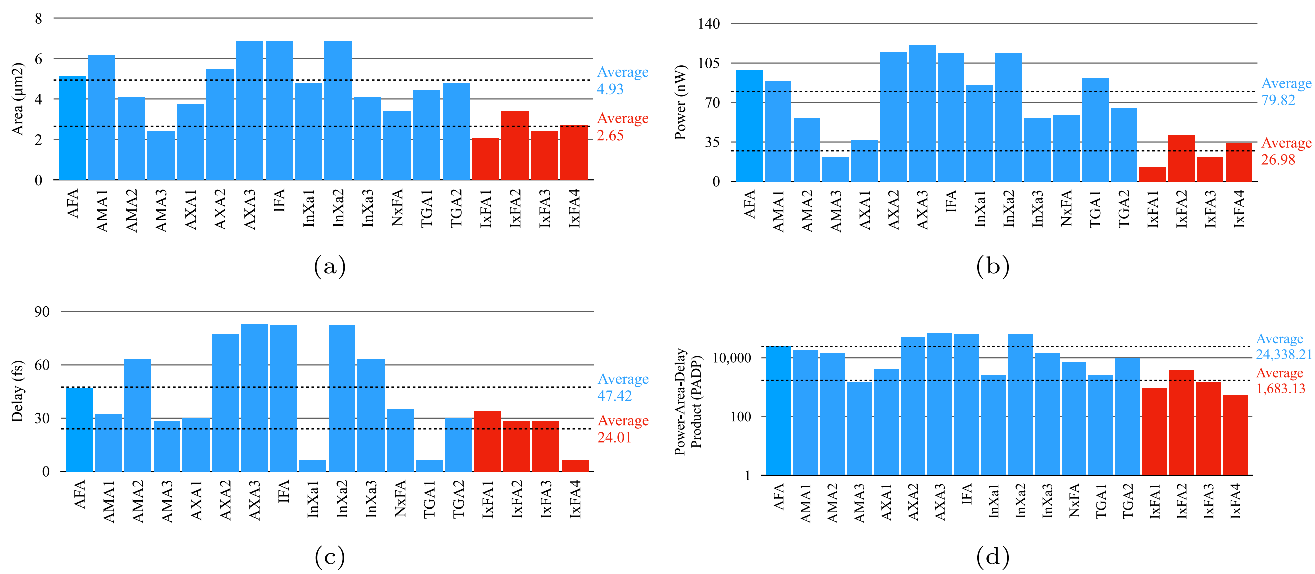


Fig. 10 a. Area, **b.** Power, **c.** Delay of the Critical Path, and **d.** PADP for the 20 Approximate FAs

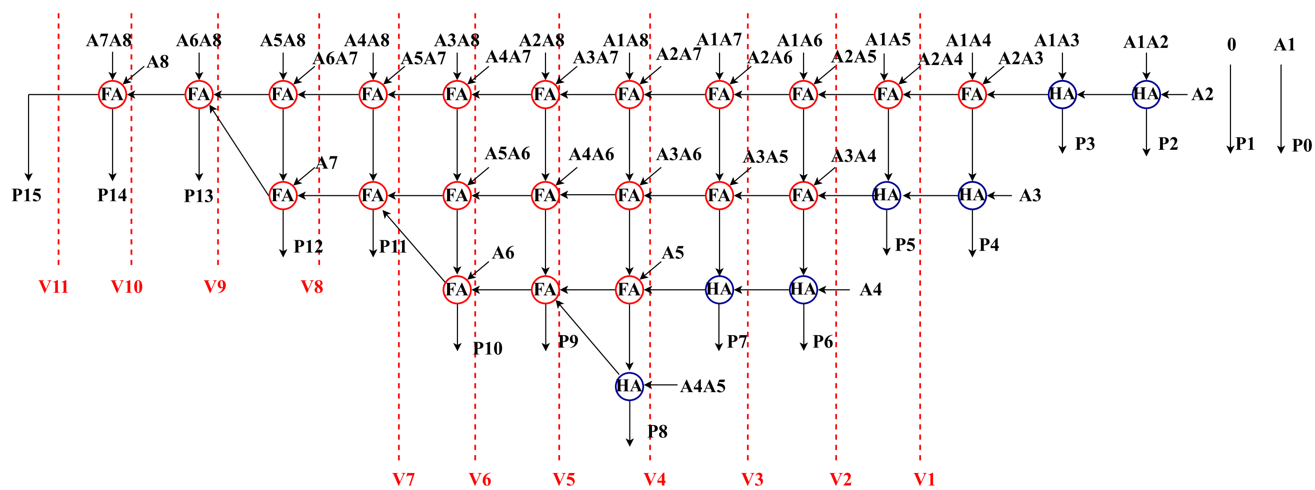


Fig. 11 Different configurations for "Columns of Approximate FAs"of *the AAS*

utilizing our proposed IxFAs, i.e., 4 × 11. Moreover, we approximated the *8-bit signed approximate array squaring* (*SAAS*) in all possible 9 levels of vertical replacement. Thus, we generated**126** designs using the FAs from the literature, i.e., 14 × 9, and **36** designs utilizing our IxFAs, i.e., 4 × 9. Next, we evaluate the various accuracy metrics and the resource usage, i.e., area, power and delay, of all proposed designs. Then, compare the metrics of the designs that are based on the IxFAs with the designs that are based on related FAs. The comparison of the proposed inexact full adders, i.e., IxFAs, with the approximate FAs proposed in the literature, is based on the average metrics, e.g., average area, for all levels of approximation when using a given type of FA. Furthermore, we refer to the *AAS* and *SAAS* designs based on the ExFA as ExFA-based *AAS* and ExFA-based *SAAS* designs, respectively.

5.1 Accuracy of the proposed designs

We implemented all designs in MATLAB [41] to assess their precision. The accuracy of the designs is for the entire 8-bit input domain. The quality analysis of the approximate designs is performed using various error metrics that have been proposed in the literature [1]. Namely, error rate (ER), mean error distance (MED), normalized mean error distance (NMED), mean relative error distance (MRED), and mean square error (MSE). Figs. 12 and 13 show the values of the error metrics of *AAS* and *SAAS* designs, respectively.

Fig. 12ashows the **ER** for *AAS*. The average ER for *AAS* designs when utilizing the ExFAs is 69.7%, while the ER for *AAS* designs utilizing the IxFAs is 76.9%. The same behavior is shown in Fig. 13awhich shows the **ER** for *SAAS*. The ER for ExFA-based *SAAS* designs is 64.4%, while the

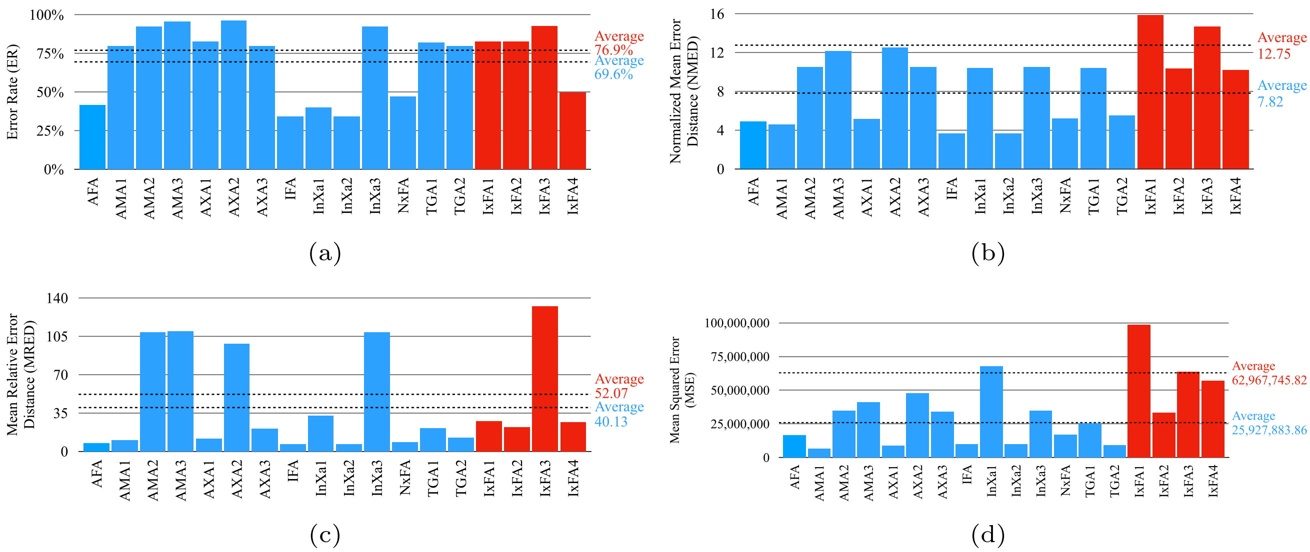


Fig. 12 Accuracy Metrics: **a**. ER, **b**. NMED, **c**. MRED, and **d**. MSE, of the Proposed 8-bit *AAS*. (Each bar is the average of 14 designs)

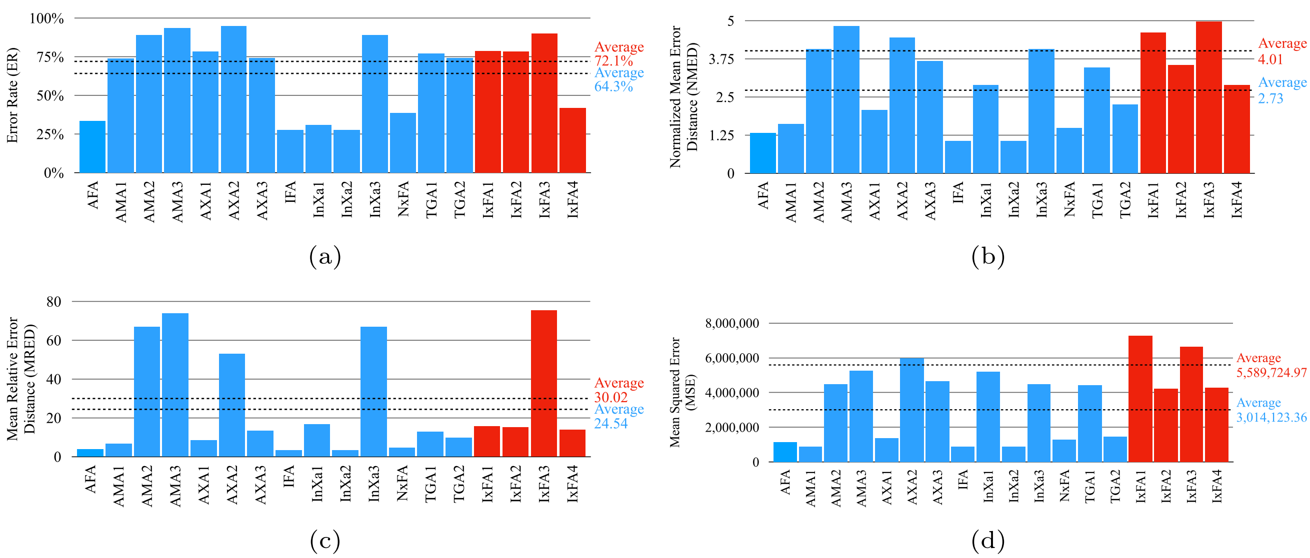


Fig. 13 Accuracy Metrics: **a**. ER, **b**. NMED, **c**. MRED, and **d**. MSE, of the Proposed 8-bit *SAAS*. (Each bar is the average of 9 designs)

ER for *SAAS* designs utilizing the IxFAs is 72.1%. Thus, the designs based on the IxFAs have on average a slightly higher ER than the designs utilizing the 14 ExFAs. However, the ER for IxFA4-based *AAS* and *SAAS* designs are better than 11 approximate ExFA-based *AAS* and 9 *SAAS* designs.

Fig. 12bshows the **NMED** for the *AAS*. The average NMED for ExFA-based *AAS* designs is 7.82, while the NMED for *AAS* designs utilizing the IxFAs is 12.75. The same behavior is shown in Fig. 13bwhich shows the **NMED** for the *SAAS*. The NMED for ExFA-based *SAAS* designs is 2.73, while the NMED for *SAAS* designs utilizing the IxFAs is 4.01. Thus, the designs based on the IxFAs have on average a higher NMED than the designs utilizing the 14 well-known FAs.

The **MRED** of signed and unsigned designs are shown in Figs. 12c and 13c, respectively. The MRED for ExFA-based *AAS* designs is 40.13, while the MRED for *AAS* designs utilizing the IxFAs is 52.07. The MRED for ExFA-based *SAAS* designs is 24.54, while it is 30.02 for *SAAS* designs utilizing the IxFAs.

Fig. 12dshows the **MSE** for *AAS*. The MSE for ExFA-based *AAS* designs ranges from $6.69 \times 10^{+06}$ to $6.77 \times 10^{+07}$ with an average of $2.59 \times 10^{+07}$. The average MSE for *AAS* designs utilizing the IxFAs is $6.29 \times 10^{+07}$. On the other hand, Fig. 13dshows the **MSE** for *SAAS*. The MSE for ExFA-based *SAAS* designs ranges from $8.54 \times 10^{+05}$ to

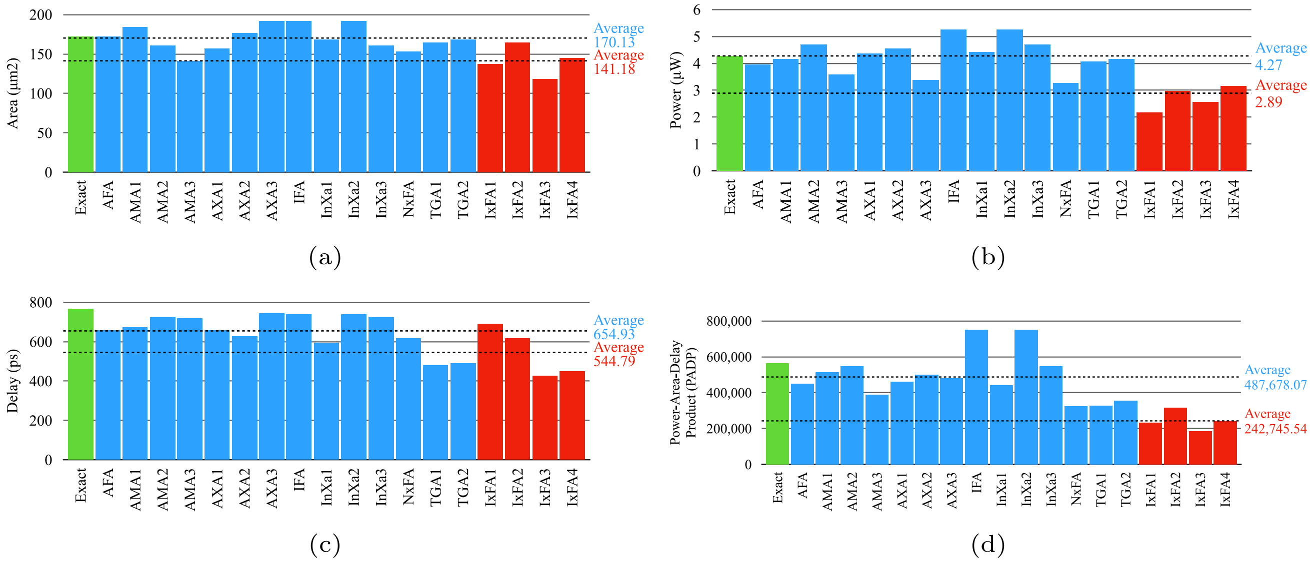


Fig. 14 a. Area, **b**. Power, **c**. Delay and **d**. Power-Area-Delay-Product, of the Proposed 8-bit *AAS*. (Each bar is the average of 11 designs)

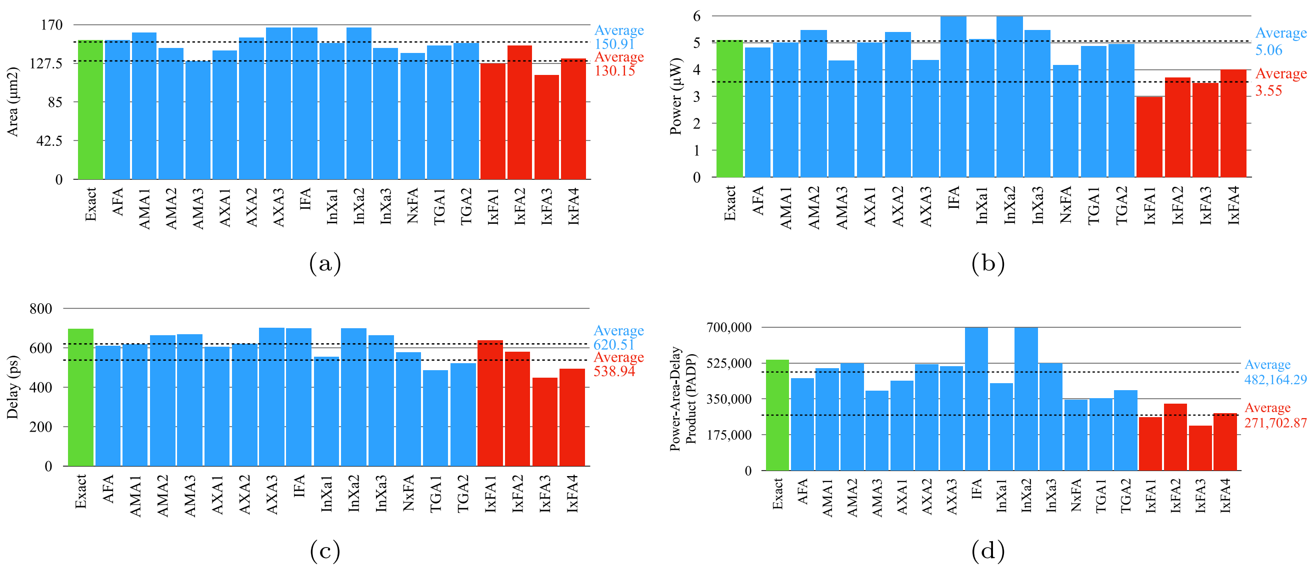


Fig. 15 a. Area, **b**. Power, **c**. Delay and **d**. Power-Area-Delay-Product, of the Proposed 8-bit *SAAS*. (Each bar is the average of 9 designs)

$5.95 \times 10^{+06}$ with an average of $3.01 \times 10^{+06}$. The average MSE for *SAAS* designs utilizing the IxFAs is $5.59 \times 10^{+06}$.

Even though the IxFA-based designs have on average lower quality than the ExFA-based designs, some designs offer a competitive quality. For instance, NMED of the *AAS* designs based on IxFA2 and IxFA4 have similar or better quality when compared to six ExFA-based *AAS* designs. Moreover, the *AAS* designs based on IxFA1, IxFA2 and IxFA4 offer a competitive MRED. These behaviors can be extended to the *SAAS* designs as we can notice from Figs. 12 and 13 that the behavior of the error in the *AAS* and *SAAS* is similar for the various approximate FAs. However, the absolute value for the metrics of *SAAS* is less compared to *AAS* since the output is 14-bit rather than 16-bit.

5.2 Hardware metrics

We implemented all of the approximate squaring designs in VHDL, where we used *Siemens QuestaSim*[42] for functional verification. On the other hand, the synthesis is performed using the *Cadence Innovus* tool [20] with 45nm GPDK. Figs. 14 and 15 show the area, power, delay and power-area-delay product (PADP) for *AAS* and *SAAS* designs, respectively, when using various approximate FAs.

5.2.1 Area of the proposed *AAS* and *SAAS* designs

Fig. 14ashows the area of the different *AAS* designs. The area for ExFA-based *AAS* designs ranges from 114.9μm^2 to

Table 4 Area (Min, Max, Avg) of various proposed Designs

	Area (μm^2)			
	AAS		SAAS	
	ExFA-based	IxFA-based	ExFA-based	IxFA-based
Min	114.9	74.6	111.5	85.8
Max	208.3	171.7	178.2	151.8
Avg	170.1	141.2	150.9	130.2

Table 5 Power (Min, Max, Avg) of various proposed Designs

	Power (μW)			
	AAS		SAAS	
	ExFA-based	IxFA-based	ExFA-based	IxFA-based
Min	2.3	0.9	3.2	1.8
Max	5.9	4.3	6.6	5.2
Avg	4.3	2.7	5.1	3.6

208.3μm^2 with an average of 170.1 μm^2 while the area of the exact squarer is 172.37 μm^2. The area of some approximate designs is greater than the exact design. Such behavior is expected as the synthesis is performed in medium effort in the various hardware metrics. Subsequently, the tool may generate a hardware solution that minimizes some aspects of the hardware metrics, e.g., power and delay, for an increase in other metrics. On the other hand, the average area of our proposed designs ranges from 74.6μm^2 to 171.7μm^2 with an average of 141.2 μm^2. Fig. 15ashows the area of different *SAAS* designs that are averaged for various vertical replacements (V1–V9) when using a given type of approximate FA. The average area for ExFA-based *SAAS* designs is 150.91μm^2 while the area of the exact signed squarer is 152.53 μm^2. The designs based on the proposed IxFAs have an average area of 130.2μm^2. Moreover, similar to the AAS design, some SAAS results in a higher area than the exact design.

Table 4 shows the maximum, minimum, and the average area of the 154 ExFA-based AAS, 44 IxFA-based AAS, 126 ExFA-based SAAS, and 36 IxFA-based SAAS. The average area of IxFA-based *AAS* and *SAAS* designs is better than the average area of all ExFA-based designs.

5.2.2 Power of the proposed *AAS* & *SAAS* designs

Fig. 14bshows the power consumption of different *AAS* designs. The power consumption of the ExFA-based *AAS* designs ranges from 2.3μW to 5.9μW, with an average of 4.3μW,while the power of the exact squarer is 4.26μW. Similar to the behavior noted in the area usage, the power consumption of some approximate designs is greater than the exact design due to their internal structure. The *AAS* designs utilizing the proposed IxFAs have a power consumption between 0.9μW and 4.3μW with an average of 2.7μW. As shown in Fig. 15b, the power of ExFA-based *SAAS* designs ranges from 3.2μW to 6.697μW with an average of 5.1μW.

Table 6 Delay (Min, Max, Avg) of various proposed Designs

	Delay (*ps*)			
	AAS		SAAS	
	ExFA-based	IxFA-based	ExFA-based	IxFA-based
Min	323	194	344	295
Max	804	768	727	702
Avg	655	545	621	539

However, the power of the exact signed squarer is 5.79μW. The designs based on the proposed IxFAs have an average power of 3.6μW with a minimum power of 1.8μW and a maximum power of 5.2μW.

Table 5 shows the maximum, minimum, and the average power of the 154 ExFA-based AAS, 44 IxFA-based AAS, 126 ExFA-based SAAS, and 36 IxFA-based SAAS. The average power consumption of IxFA-based *AAS* and *SAAS* designs is better than the average power of all ExFA-based designs.

5.2.3 Delay of the proposed *AAS* and *SAAS* designs

As shown in Fig. 14c, the delay of the ExFA-based *AAS* designs ranges from 323*ps* to 804*ps* with an average of 655*ps*. However, the delay of the exact unsigned squaring unit is 767*ps*. The delay for *AAS* designs utilizing the proposed IxFAs ranges from 194*ps* to 768*ps* with an average of 545*ps*. Fig. 15cshows the delay of the signed designs, which ranges from 344*ps* to 727*ps* with an average of 621*ps*. However, the delay of the exact signed squaring unit is 698*ps*. The delay for *SAAS* designs utilizing the proposed IxFAs ranges from 295*ps* to 702*ps* with an average of 539*ps*. Table 6 shows the maximum, minimum, and the average delay of the 154 ExFA-based AAS, 44 IxFA-based AAS, 126 ExFA-based SAAS and 36 IxFA-based SAAS.

5.2.4 PADP of the proposed *AAS* and *SAAS* designs

To summarize the performance of the various proposed designs, we use the Power-Area-Delay Product (PADP), where Figs. 14d and 15d show the PADP for different *AAS* and *SAAS* designs, respectively. The PADP values are measured in $\mu m^2 \cdot \mu W \cdot ps$. Utilizing the ExFAs, the PADP of the *AAS* designs ranges from 325 K to 748 K, with an average of 488K. The PADP of the designs based on the ExFAs proposed in [37, 38], i.e., IFA and InXa2, is higher than the exact design. This can be assigned to the internal structure of the approximate FAs where the synthesis tool is unable to optimize/simplify the structure of the approximate squarer. Utilizing the proposed IxFAs, the *AAS* designs have a PADP between 184 K and 315 K with an average of 242K. As shown in Fig. 15d, the PADP of ExFA-based *SAAS* designs ranges from 344 K to 695 K with an average 482K.

However, the designs based on the proposed IxFAs have an average PADP of 271 K with a minimum value of 220 K and a maximum value of 325K. **The average PADP of IxFA-based *AAS* and *SAAS* designs is better than the average PADP of all ExFA-based designs.**

6 Comparison with related work

The proposed inexact full adders, i.e., IxFAs, achieved minimal resource usage with competitive quality. In this section, we compare the proposed approximate squarers with the work presented in the literature. Since the ABS and ABFS designs proposed in [30], and [31], respectively, support signed numbers, we will limit the comparison in this section to the *SAAS* designs.

The various ABS designs have an **area** ranging from $390\mu m^2$ to $433\mu m^2$ with an average of $417.32\mu m^2$. However, the area of the *SAAS* designs ranges from $85.8\mu m^2$ to $178.2\mu m^2$ with an average of $145.8\mu m^2$. Thus, all of the proposed designs utilizing the 18 types of FAs have a smaller area than the ABS designs.

Regarding the **power**, the various ABS designs have a power consumption ranging from $95\mu W$ to $123\mu W$ with an average of $110\mu W$. The power of the *SAAS* designs varies from $1.81\mu W$ to $6.59\mu W$ with an average of $4.73\mu W$. Thus, all of the proposed designs utilizing the 18 types of FAs have a smaller power than the ABS designs.

The **delay** of the ABS designs ranges from 560*ps* to 840*ps* with an average of 710*ps*, while the *SAAS* designs have a delay between 295*ps* to 727*ps* with an average of 603*ps*. Similar to the area and power consumptions, all of the proposed designs utilizing the 18 types of FAs have smaller delays, and thus smaller PADPs than the ABS designs.

We approximate the *SAAS* designs based on the vertical replacement of FAs, i.e., 9 levels of approximation. We used 14 ExFAs along with our four proposed IxFAs for every level. To summarize the comparison with [30, 31] and identify the outperforming design(s), we perform a comparison based on the power-area-delay product (PADP) and the output quality (NMED) of the designs. Fig. 16 compares the designs proposed in [30, 31] and our *SAAS* designs. From the ExFA-based *SAAS* designs, for every level of approximation, i.e., V1–V9, we selected two designs: (*i*) the design with the best/lowest NMED which is named $\mathrm{ExFA_{min}}$; and (*ii*) the design with the worst/highest NMED which is named $\mathrm{ExFA_{max}}$. From Fig. 16, we can notice that the PADP of the $\mathrm{ExFA_{min}}$ fluctuates. This is due to the fact that for different levels of approximation, different designs achieved minimal NMED. Subsequently, each design has a different PADP. In addition, due to graphical limitation, Fig. 16 does not show one value of $\mathrm{ExFA_{min}}$ where one design achieved an NMED of 0, i.e., no error.

Moreover, for every level of approximation, i.e., V1–V9, we computed the average NMED and the average PADP for the ExFA-based designs, where we denote these values as $\mathrm{ExFA_{avg}}$ in Fig. 16. Furthermore, we show the NMED and PADP for the IxFA-based *SAAS* designs in Fig. 16 for the various levels of approximation, i.e., V1–V9. The IxFA-based *SAAS* designs are denoted in Fig. 16 based on the IxFA used in the design. For example, the *SAAS* design based on IxFA1 is shown as IxFA1 in Fig. 16.

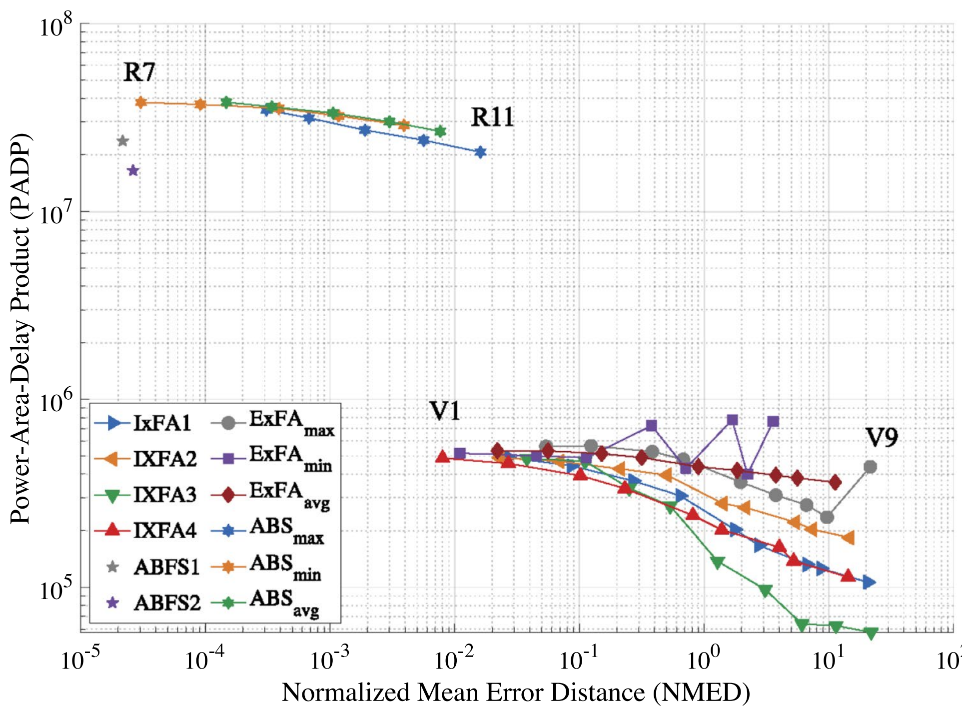


Fig. 16 PADP versus NMED for the *SAAS*, ABS [30] and ABFS [31] Designs

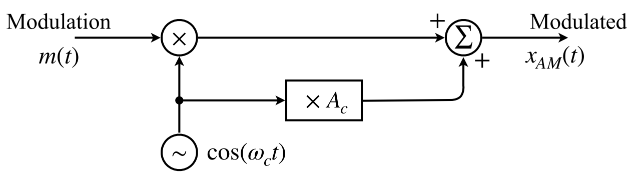


Fig. 17 Generating an AM Signal [5]

The designs proposed in [30], i.e., ABS, have various configurations with the degrees of approximation varying from R7 to R11, where for a larger R the error increases. Thus, similar to the ExFA-based *SAAS* designs, we select the ABS design with the minimum, maximum, and average NMED. These designs are denoted in Fig. 16 as ABS_{min}, ABS_{max}, and ABS_{avg}, respectively. The proposed *SAAS* designs have less PADP compared to the related work with an acceptable quality (NMED). In practice, given the large number of proposed designs, a designer can choose the most suitable design based on the output quality and the required hardware resources as they cover a wider range of PADP and NMED.

Compared to the designs proposed in [31], i.e., ABFS1 and ABFS2, we can notice a great reduction in PADP. Moreover, since the authors proposed only two designs, the range of NMED is quite small. The proposed *SAAS* designs utilizing IxFAs show superiority to the ExFA-based designs. The IxFA4-based designs have the least PADP for four levels of approximation (V1–V4) where a higher level of approximation resulted in a larger error with a reduced PADP. Similarly, the IxFA3-based designs deliver another set of low PADP for six levels of approximation (V4–V9). Thus, we determine these designs to be the Pareto optimal. On another note, we observe that the designs based on IxFA1 and IxFA2 achieve a similar quality to the designs based on ExFAs while offering a lower PADP. Subsequently, the *SAAS* designs based on our IxFAs have an efficient PADP and acceptable NMED.

7 Applications

We evaluate the performance of the proposed designs using two DSP applications, i.e., square law demodulation [5] and averaged root mean square of gray-scale images [25].

7.1 Square law demodulation

Modulation changes one or more properties of the *carrier signal*, i.e., *c*(*t*). For that, a *modulation signal, m*(*t*), is used, where its frequency is lower than the carrier signal. In amplitude modulation (AM) the strength of the carrier signal *c*(*t*) is changed by the *m*(*t*). In real-life applications, *m*(*t*) would be a complex signal composed of *sine* and *cosine* signals.

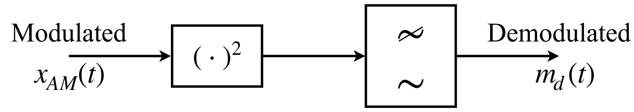


Fig. 18 AM Demodulation by Squaring and Low-pass Filtering [5]

Here, we use a sinusoidal signal, as given in Eq. (2). The sinusoidal carrier signal *c*(*t*) is given by Eq. (3), where A_c and f_c are the amplitude and frequency of the carrier signal, respectively. As shown in Fig. 17, the resultant modulated signal is called $x_{AM}(t)$ which is given by Eq. (4). The value of the *modulation index*, i.e., μ, equals $\frac{A_m}{A_c}$.

$$m(t) = A_m cos(\omega_m t) \tag{2}$$

$$c(t) = A_c cos(\omega_c t) = A_c cos(2\pi f_c t) \tag{3}$$

$$x_{AM}(t) = A_c\ cos(\omega_c t)\ [1 + \mu\ cos(\omega_m t)] \tag{4}$$

On the receiver side, we want to restore *m*(*t*) from the received $x_{AM}(t)$ by using a simple signal rectifier followed by a low-pass filter (LPF). A square law demodulator (SLD) is used for the demodulation of the AM signals. Thus, at the receiver, we square the received $x_{AM}(t)$ where the obtained expression is given by Eq. (5). As shown in Fig. 18, the SLD passes the AM signal through a squaring circuit. Then, the generated signal is transferred through an LPF to attenuate signals with high frequencies [5]. Finally, the SLD generates a copy of the original message.

$$x_{AM}^2(t) = A_c^2\ cos^2(\omega_c t)[1 + \mu\ cos(\omega_m t)]^2 \tag{5}$$

$$m_d(t) = A_m A_c cos(\omega_m t) + (\frac{\mu A_c}{2})^2 cos(2\omega_m t) \tag{6}$$

The squaring of the received $x_{AM}(t)$ consists of constant terms, several high-frequency components, and other terms. The LPF removes high-frequency components. The direct current (DC) block removes the constant terms, and the remaining signal is the demodulated signal as given in Eq. (6). To assess the efficiency of the proposed approximate squaring designs, we built a MATLAB model for the modulation/demodulation process. To convert floating-point numbers to 8-bit integers we used a scaling factor of 2^8. To obtain a negligible quantization noise, we used a sampling frequency of 20 *kHz*. To eliminate high-frequency terms we used a built-in lowpass filter with a normalized passband frequency of 0.1 π/second. As shown in Section 6, the usage of the IxFA resulted in superior squaring designs. Thus, in this section we use the IxFA-based *AAS* designs to evaluate the square of $x_{AM}(t)$.

The Euclidean distance $E_{A,B}$ between two signals, i.e., *A* and *B*, measures the straight-line distance between two points in the two signals and is given by Eq. (7) [43], where S is the number of points. To numerically differentiate the performance of the squaring functions, the Euclidean distance between the correctly demodulated signal $m(t)$ and those obtained employing the approximate squaring functions, i.e., $m_d(t)$, are computed based on Eq. (7).

$$E_{A,B} = \sqrt{\sum_{i=1}^{S}(A_i - B_i)^2} \tag{7}$$

The Euclidean distance of the output waveforms generated by SLD with the proposed squarers are evaluated for one full cycle of message signal with 1024 samples, i.e., $S = 1024$, and shown in Table 7. The IxFA-based *AAS* designs are evaluated for the various levels of approximation, i.e., V1-V11. On the other hand, Table 8 shows the Euclidean distance for the 3 designs of [30]. Among the proposed designs, we have 28 ones with Euclidean distances of less than 0.492. Therefore, they have the best quality. Moreover, We have five designs (highlighted in italic) with Euclidean distances of less than 0.887, which are competitive to the related work. The remaining 11 designs (highlighted in bold) have an Euclidean distance greater than the related work [30].

7.2 Averaged root mean square of gray-scale images

In image processing applications, an 8-bit gray-scale image (X) can be represented as a two-dimensional array of unsigned integers. Each pixel $x(i, j)$ has a value that ranges from 0 to 255, where $i = 1,..., r$ and $j = 1,..., c$ are the coordinates of the pixel in an $r \times c$ image. The measure of information provided by an image is called *image energy*, which is quantified by calculating the root mean square (RMS) of image pixels as given in Eq. (8). The total number of pixels is represented by $n = r \times c$. Thus, energy evaluation requires a large number of squaring operations.

Image energy represents the rate of change in the color, brightness or magnitude of the pixels over limited ranges, which is used in various image processing algorithms [25],

Table 7 The Euclidean Distance of the Proposed Designs

Configuration	IxFA1	IxFA2	IxFA3	IxFA4
V1	0.0584	0.0583	0.0605	0.0583
V2	0.0586	0.0594	0.0675	0.0587
V3	0.0584	0.0590	0.0803	0.0609
V4	0.0645	0.0633	0.1019	0.0615
V5	0.1244	0.1066	0.1714	0.0717
V6	0.2161	0.1471	0.3784	0.0952
V7	*0.5956*	0.4200	*0.6693*	0.1556
V8	*0.8293*	0.4720	**1.1299**	0.2416
V9	**2.1613**	**1.3274**	**1.8869**	*0.5000*
V10	**2.4537**	**2.3060**	**3.4693**	*0.6669*
V11	**6.3353**	**3.7689**	**6.1328**	**1.1080**

Table 8 The Euclidean Distance of the Related Designs

The Squaring Design	The Euclidean Distance
ABS1 [30]	0.746
ABS2 [30]	0.887
ABS3 [30]	0.492

e.g., image segmentation, classification, fusion, and image distance measure. For a set of images, the average RMS is calculated by Eq. (9), for m images.

$$RMS = \frac{1}{n}\sqrt{\sum_{i,j} x^2(i,j)} \tag{8}$$

$$ARMS = \frac{1}{m}\sum_{k=1}^{m} RMS_k \tag{9}$$

We calculated the RMS for a set of 50 8-bit gray-scale images, each of size 256 × 256 pixels. For that, we used a set of 44 IxFAs-based AAS designs. Fig. 19 summarizes the results of the image processing. Fig. 19ashows the values of the Average root mean square (ARMS) for the images evaluated using various squaring designs. The ARMS for an exact design is 0.416, and we consider a value of 0.416 ± 10% as acceptable, i.e., from 0.374 to 0.458. Thus, we have 7 designs with ARMS < 0.374 and 4 designs with ARMS > 0.458. These 11 designs are clearly shown

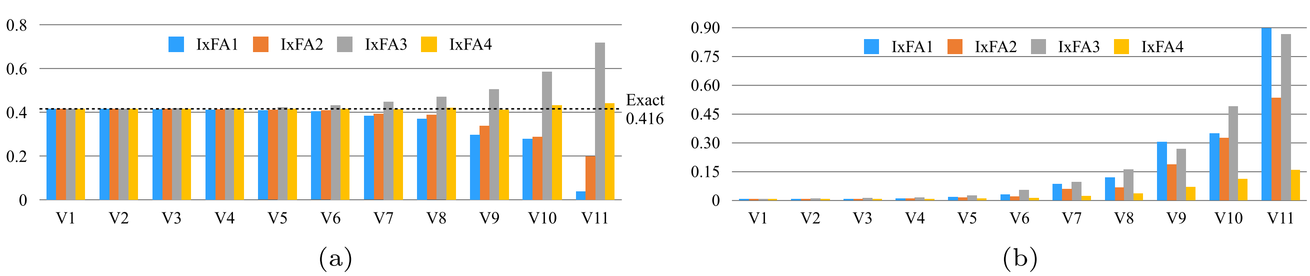


Fig. 19 a. Average Root Mean Square (ARMS), and **b** Mean Relative Error Distance (MRED) for 50 Images Evaluated Using Various Squaring Designs

in Fig. 19a. Thus, 33 designs have an acceptable ARMS, i.e., $0.458 \geq \text{ARMS} \geq 0.374$.

Averaging a set of RMS values for an approximate design could result in a value that is close to the exact one [25] even though some values may have a positive error and others have a negative error. Thus, for each squaring design, we calculate the mean relative error (MRE) based on 50 images, where the error represents the difference between the exact RMS and the approximate RMS as expressed in Eq. (10).

$$MRE = \frac{1}{m}\sum_{i=1}^{m} \frac{|RMS_{Exact_i} - RMS_{Approx_i}|}{RMS_{Exact_i}} \tag{10}$$

Fig. 19bshows the obtained mean relative error for the various designs. If we considered a maximum value of 0.1 as an acceptable MRE, we find that the 13 designs with MRE > 0.1 and 11 of them matched the ones with unacceptable ARMS. Thus, at the application level, we showed that many of the proposed designs have a very satisfying output quality.

8 Conclusion

Approximate computing reduces the execution time and energy consumption of error-resilient applications, e.g., DSP, by slackening the quality requirements. In this paper, we performed a design space exploration, i.e., investigated and proposed various designs of 8-bit array-based approximate squaring functions that can be implemented with numerous levels of accuracy by changing (*i*) the type of the used approximate full adders, and (*ii*) the number of theseapproximate adders, which are organized column-based in the partial product array. For this aim, we proposed four new energy-efficient approximate FAs, i.e., IxFA, suitable for the squaring function. The detailed analysis confirmed that the IxFA-based squaring units have optimized results for both hardware and accuracy parameters. The unsigned approximate squaring units, i.e., *AAS*, based on the proposed IxFAs, have an average reduction of 18.14%, 36.38%, and 28.94%, in area, power and delay, respectively. Similarly, the proposed signed approximate squaring units, i.e., *SAAS*, have a reduced area, power, and delay compared to the exact signed squarer, with an average of 14.67%, 38.69%, and 22.79%, respectively. Moreover, the *SAAS* designs have a less power-area-delay product (PADP) compared to the state-of-the-art approximate squarers while delivering a competitive quality.

Finally, the efficiency of the proposed designs is verified by employing them in real-world applications, namely, a square-law demodulator and an averaged root mean square of gray-scale images. The results showed better accuracy for 28 proposed designs for both applications than related work. In approximate computing, the final application quality depends on the applied individual inputs, where, for some values, the output accuracy could be degraded significantly. Therefore, as a future direction, we will investigate the design of a run-time adaptive squaring function based on the applied inputs. As another future direction, we propose to approximate the squarer designs in heterogeneous replacements, i.e., using more than one approximate FA in a given approximate squarer.

Acknowledgements A preliminary version of this work has been published in ICM 2024 [44].

Author contributions M.M. conceptualized the study, developed the proposed approximate squaring designs, and performed the design space exploration. A.A. contributed to the implementation, data analysis, and preparation of figures and tables. S.T. supervised the research, provided critical insights into DSP application relevance, and revised the manuscript for important intellectual content. All authors contributed to the writing, reviewed, and approved the final manuscript.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Masadeh, M., Hasan, O., & Tahar, S. (2021). Machine-learning-based self-tunable design of approximate computing. *IEEE Transactions on Very Large Scale Integration Systems, 29*(4), 800–813.
2. Tirupathireddy, A., Sarada, M., & Srinivasulu, A. (2021). Energy-efficient approximate adders for DSP applications. *Analog Integrated Circuits and Signal Processing, 107*, 649–657.
3. Sasmal, M., & Joseph, T. (2024). Approximate multiplier design With LFSR-Based stochastic sequence generators for edge AI. *IEEE Computer Architecture Letters, 23*(1), 91–94.
4. Masadeh, M., Hasan, O., & Tahar, S. (2019). Input-Conscious Approximate Multiply-Accumulate (MAC) unit for energy-efficiency. *IEEE Access, 7*, 147129–147142.
5. Leis, J. W. (2018). *Communication Systems Principles Using MATLAB*. John Wiley & Sons.
6. Krishna, L. H., Sk, A., Rao, J. B., Veeramachaneni, S., & Sk, N. M. (2024). Energy-efficient approximate multiplier design with lesser error rate using the probability-based Approximate 4: 2 compressor. *IEEE Embedded Systems Letters, 16*(2), 134–137.
7. Masadeh, M., Hasan, O., & Tahar, S. (2018). Comparative study of approximate multipliers. *Great Lakes Symposium on VLSI* (pp. 415–418). ACM.
8. Moses, J., Balasubramani, S., & Krishnamoorthy, U. (2025). Novel approximate Booth multipliers (ABm-eRx) based on efficient encoding and reduction for error-tolerant applications. *Analog Integrated Circuits and Signal Processing, 123*(2), 34.

9. Yang, Z., Han, J., & Lombardi, F. (2015). Approximate compressors for error-resilient multiplier design. In: *International Symposium on Defect and Fault Tolerance in VLSI and Nano Systems*, (pp. 183–186).
10. 8-bit PICÂ® and AVRÂ® Microcontrollers (2021). https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus, Last accessed on 2025-04-04.
11. Muralidharan, V., & Kumar, N. S. (2020). Design and implementation of low power and high speed multiplier using quaternary carry look-ahead adder. *Microprocessors and Microsystems, 75*, Article 103054.
12. Shao, B., & Li, P. (2015). Array-based approximate arithmetic computing: A general model and applications to multiplier and squarer design. *IEEE Transactions on Circuits and Systems I: Regular Papers, 62*(4), 1081–1090.
13. Balasubramanian, P., Nayar, R., & Maskell, D. L. (2021). Approximate array multipliers. *Electronics, 10*(5), 630.
14. Yamamoto, T., Taniguchi, I., Tomiyama, H., Yamashita, S., & Hara-Azumi, Y. (2016). A systematic methodology for design and analysis of approximate array multipliers. In: *Asia Pacific Conference on Circuits and Systems*, (pp. 352–354).
15. Masadeh, M., Elderhalli, Y., Hasan, O., & Tahar, S. (2021). A Quality-Assured Approximate Hardware Accelerators-Based on Machine Learning and Dynamic Partial Reconfiguration. *Journal of Emerging Technologies in Computing Systems, 17*(4), 1–19.
16. Chen, T. C. (1971). A binary multiplication scheme based on squaring. *IEEE Transactions on Computers C-, 20*(6), 678–680.
17. Liddicoat, A. A., & Flynn, M. J. (2000). Parallel square and cube computations. *Asilomar Conference on Signals, Systems and Computers, 2*, 1325–1329.
18. Bui, S. & Stine, J.E. (2014). Additional optimizations for parallel squarer units. In: *IEEE International Symposium on Circuits and Systems*, (pp. 361–364).
19. Kolagotla, R., Griesbach, W., & Srinivas, d.H. (1998). LSI Implementation of 350 MHz 0.35 [CDATA[\mu]]μm 8 bit merged squarer. *V. Electronics Letters, 34*(1), 47–48.
20. Cadence: Innovus Implementation System. Last accessed on 2025-04-04. https://www.cadence.com/ko_KR/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html.
21. Wey, C.-L., & Shieh, M.-D. (1998). Design of a high-speed square generator. *IEEE Transactions on Computers, 47*(9), 1021–1026.
22. Hoang, V.-P. & Pham, C.-K. (2012). Low-error and efficient fixed-width squarer for digital signal processing applications. In: *International Conference on Communications and Electronics*, (pp. 477–482). IEEE.
23. Park, I.-C., & Kim, T.-H. (2010). Multiplier-less and table-less linear approximation for square-related functions. *IEICE Transactions on Information and Systems E93.D, 11*, 2979–2988.
24. Langlois, J. M. P., & Al-Khalili, D. (2006). Carry-free approximate squaring functions with O(n) complexity and O(1) delay. *IEEE Transactions on Circuits and Systems II: Express Briefs, 53*(5), 374–378.
25. Avramović, A., Babić, Z., Raič, D., Strle, D., & Bulić, P. (2014). An approximate logarithmic squaring circuit with error compensation for DSP applications. *Microelectronics Journal, 45*(3), 263–271.
26. Ansari, M.S., Cockburn, B.F. & Han, J. (2020). Low-power approximate logarithmic squaring circuit design for DSP applications. *IEEE Transactions on Emerging Topics in Computing*.
27. Anguraj, K. T. S. S. P. (2022). Cmos implementation and performance analysis of known approximate 4: 2 compressors. *Journal of Electronic Testing, 38*(4), 353–370.
28. Anguraj, K. T. P. (2023). Design and realization of area-efficient approximate multiplier structures for image processing applications. *Microprocessors and Microsystems, 102*, Article 104925.
29. Mitchell, J. N. (1962). Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers EC-, 11*(4), 512–517.
30. Reddy, K. M., Vasantha, M., Kumar, Y. N., & Dwivedi, D. (2020). Design of approximate booth squarer for error-tolerant computing. *IEEE Transactions on Very Large Scale Integration Systems, 28*(5), 1230–1241.
31. Xu, C., Cui, Z., Chen, K., & Liu, W. (2022). Design and analysis of energy-efficient approximate booth-folding squarers with precision recovery. *Electronics Letters, 58*(9), 349–351.
32. Mazahir, S., Hasan, O., & Shafique, M. (2018). Adaptive approximate computing in arithmetic datapaths. *IEEE Design & Test, 35*(4), 65–74.
33. Venkatesan, R., Agarwal, A., Roy, K., & Raghunathan, A. (2011). Macaco: Modeling and analysis of circuits for approximate computing. In: *IEEE/ACM International Conference on Computer-Aided Design*, (pp. 667–673).
34. Venkatachalam, S., & Ko, S.-B. (2017). Design of Power and Area Efficient Approximate Multipliers. *IEEE Transactions on Very Large Scale Integration Systems, 25*(5), 1782–1786.
35. Gupta, V., Mohapatra, D., Raghunathan, A., & Roy, K. (2012). Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 32*(1), 124–137.
36. Yang, Z., Jain, A., Liang, J., Han, J. & Lombardi, F. (2013). Approximate XOR/XNOR-based adders for inexact computing. In: *IEEE International Conference on Nanotechnology*, (pp. 690–693).
37. Guo, Y., Sun, H. & Kimura, S. (2018). Design of power and area efficient lower-part-OR approximate multiplier. In: *IEEE Region 10 Conference*, (pp. 2110–2115). IEEE.
38. Almurib, H.A.F., Kumar, T.N., & Lombardi, F. (2016). Inexact designs for approximate low power addition by cell replacement. In: *Design, Automation and Test in Europe,* (pp. 660–665).
39. Waris, H., Wang, C., Liu, W., Han, J., & Lombardi, F. (2022). Hybrid partial product-based high-performance approximate recursive multipliers. *IEEE Transactions on Emerging Topics in Computing, 10*(1), 507–513.
40. Yang, Z., Han, J., & Lombardi, F. (2015). Transmission gate-based approximate adders for inexact computing. In: *IEEE/ACM International Symposium on Nanoscale Architectures*, (pp. 145–150).
41. Mathworks: What is MATLAB. Last accessed on 2025-04-04. https://www.mathworks.com/discovery/what-is-matlab.html.
42. Siemens Digital Industries Software: Questa Advanced Simulator. Last accessed on 2025-04-04. https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/
43. Anton, H., & Rorres, C. (2014). *Elementary Linear Algebr Applications Version*. John Wiley & Sons.
44. Masadeh, M., Aoun, A., Tahar, S. (2024). Energy-Efficient Approximate Squaring Unit. In: *International Conference on Microelectronics*, (pp. 1–5).

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.