Design of Approximate Booth Squarer for Error-Tolerant Computing

K. Manikantta Reddy, M. H. Vasantha, *Member, IEEE*, Y. B. Nithin Kumar, *Member, IEEE*, and Devesh Dwivedi

***Abstract*— To explore the benefits of approximate computing, this article proposes an approximate partial product generator for squarer (APPGS). Using APPGS, three designs of approximate radix-4 Booth squarers (ABS1, ABS2, and ABS3) are proposed. APPGS produces approximate partial products in r number of least significant columns of the partial product matrix. ABS2 and ABS3 utilize approximate adders and compressors with a novel input signal rearrangement method for the accumulation of approximate partial products. Moreover, the ABS3 features an error recovery module at k number of most significant columns of the approximate partial products. The proposed squarers with different values of r and k are simulated using 45-nm CMOS technology. The results indicate that the proposed squarers achieve optimized performance for both hardware and accuracy metrics. Compared to the exact Booth squarer, the 16-bit ABS1 with $r = 16$ achieves a reduction of 13.6%, 22.2%, and 13.7% in power, delay, and area, respectively, with a normalized mean error distance (NMED) of 4.6×10^{-6}. The ABS2 has power, delay, and area savings of 25.8%, 33.8%, and 19.8%, respectively, with an NMED of 7.2×10^{-6}. The ABS3 with $k = 6$ has 18.5% reduction in power, 29.4% reduction in delay, and 16.9% reduction in area with an NMED of 0.56×10^{-6}. The performance of the proposed squarers is evaluated with a telecommunication application, where the ABS3 with $k = 6$ produces an output signal with a signal-to-noise ratio of 32.45 dB.**

***Index Terms*— Approximate Booth squarer, approximate computing, approximate partial product generator for squarer (APPGS), input signal rearrangement (ISR), signal probability.**

I. Introduction

SQUARING operation is a fundamental requirement in many applications such as pattern recognition, image compression, vector quantization and equalization, adaptive filtering, and Euclidean branch calculation [1]. Even though the squaring operation can be performed using a multiplier, the existing simplification methods in [2]–[5] proved that a squarer circuit significantly increases the speed of squaring operation as compared to a multiplier. Therefore, to meet the operational necessities of real-time applications, it is desirable to have a dedicated fast squaring circuit.

Multiplication and squaring operations consist of three steps: generation of partial products, accumulation of partial products to reduce the partial product array into two rows, and addition of these two rows to obtain the final result. In the case of a multiplier, partial products can be obtained by multiplying each bit of the first operand with the second operand. When x_i bit of the first operand is multiplied with y_j bit of the second operand, then the partial product bit $PP_{i,j} = x_i y_j$ is generated in the $(i + j)$th column of partial product array. In the case of a squarer, as both the input operands are the same, partial product bits $PP_{i,j}$ and $PP_{j,i}$ are equal to $x_i x_j$. The folding technique proposed in [2] reduces the number of partial products of the squarer by exploiting this symmetry. Since the summation of the two partial products $x_i x_j$ and $x_j x_i$ is equal to $2x_i x_j$, these two partial products in $(i + j)$th column of the partial product array are written as a single partial product $x_i x_j$ in the $(i + j + 1)$th column. For an N-bit multiplier, the maximum height of partial product array is N, whereas it reduces to $\lfloor N/2 \rfloor + 1$ in a folded squarer. The reduction in the number of partial products reduces the hardware complexity in their accumulation. Booth encoding of input operands can be employed in partial product generation which further reduces the hardware complexity in the accumulation stage. The Booth folding encoding squarer [3] combines radix-4 Booth encoding of input operands and folding of partial products to further reduce the number of partial products to 50% as compared to simple folding technique. To reduce the propagation delay of a simple folded squarer, partial products are merged using logic simplifications in [4] and [5].

Approximate computing has emerged as an attractive paradigm to achieve low power and high speed designs with reduced complexity. In many applications related to human perception such as image and video processing, the numerical exactness of the output can be relaxed. Approximate squarers based on simplification of squaring algorithm [6]–[8] and fixed width operation [9]–[14] are proposed in the literature to reduce the power consumption, delay, and area at the cost of approximation error. Another approach in approximate computing while maintaining full bit width at the output is to use approximate arithmetic blocks for the accumulation of partial products. Using this approach, various approximate adders, compressors, and multipliers are extensively analyzed in the recent years [16]–[33].

Manuscript received October 23, 2019; revised January 11, 2020 and February 12, 2020; accepted February 13, 2020. Date of publication March 5, 2020; date of current version April 24, 2020. This work was supported by the Digital India Corporation, Ministry of Electronics and Information Technology, Government of India under Visvesvaraya Ph.D. Scheme. *(Corresponding author: K. Manikantta Reddy)*

The authors are with the Department of Electronics and Communications Engineering, National Institute of Technology Goa, Goa 403401, India (e-mail: reddy78922@gmail.com; vasanthmh@nitgoa.ac.in; nithin.shastri@nitgoa.ac.in; dev_dwivedi@yahoo.com).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2020.2976131

To overcome the tradeoff between the existing accurate and approximate squarers in terms of hardware performance and accuracy, this article proposes three designs of approximate Booth squarer. In the first design, an approximate partial product generator is proposed to increase the speed of generation of partial product bits. The hardware complexity of the squarer is further reduced in the second proposed design with the use of approximate adders and compressors for the accumulation of partial products. An input signal rearrangement (ISR) method is proposed to reorder the inputs of approximate blocks, which improves the accuracy of the squarer output. The third proposed design introduces an error recovery module to reduce the errors introduced due to the use of approximate Booth encoder. The proposed squarer designs along with various existing squarers are simulated. The results indicate that the proposed designs show the best optimum performance in terms of accuracy and hardware metrics. Finally, these designs are used in a square law demodulator (SLD) to demodulate an amplitude modulated (AM) signal. The proposed designs have a better signal-to-noise ratio (SNR) compared to other existing designs.

The rest of this article is organized as follows. A brief review of related prior works on approximate computing is provided in Section II. An approximate partial product generator is proposed in Section III. The proposed approximate Booth squarers are described in Section IV. Section V provides a detailed analysis of the accuracy and simulation results of proposed designs. Section VI compares the proposed designs with existing designs of squarer. The performance of the proposed squarers is analyzed with a real-time application in Section VII. Finally, conclusions are drawn in Section VIII.

II. Related Work

A brief literature review on some of the related works is presented in this section. At first, the available approximate squarers based on the approximation of the exact squaring equation and approximation of partial product array are discussed. Later, recently proposed approximation methods for the design of approximate multipliers are briefly reviewed. These methods include: 1) approximating multiplication equation; 2) approximating partial product equation; and 3) accumulating partial products using approximate adders or compressors.

Approximate squarers proposed in [6] and [7] extract a few basic terms in the exact squaring equation and generate Boolean equations based on precomputed sums (PCS). Later, systematic compensation methods are used to increase the accuracy of the output. The quadratic squaring algorithm proposed in [8] separates input operand into two parts (x and y) and the squaring operation is performed based on basic algebraic expression $(x + y)^2$. This approximate squarer neglects the quadratic term (y^2) that contains LSB bits of the input operand.

Approximate squarers based on fixed-width operation are proposed in [9]–[14]. For an N-bit input, the fixed-width squarers produce only N output bits by truncating N least significant bits (LSBs) of the output or by truncating partial products corresponding to N LSBs of the output. In order to reduce truncation error, dynamic error compensation methods are proposed in [9]–[12] by dividing the truncation part (TP) into two subparts called minor ($\mathrm{TP_{mi}}$) and major ($\mathrm{TP_{mj}}$). In the $\mathrm{TP_{mi}}$ part, a constant bias is added to the squarer result without generating partial products, where, as for the $\mathrm{TP_{mj}}$ part, partial products are generated. An approximate squarer based on the left-to-right leading digit high radix dual recoding is proposed in [13]. In this fixed-width squarer, input operands are encoded starting from MSB to LSB and the partial products at LSB columns are neglected after considering a few guard bits. The array-based approximate squarer proposed in [14] truncates the partial products corresponding to LSB columns of the partial product matrix and an optimized error compensation unit (ECU) is used to compensate for the truncation error. An approximate Squarer Accumulator with SelfHealing (SquASH) is proposed in [15] for calculating the inner product of a vector with itself. This approximate squarer accumulator (SAC) introduces two errors $+\delta$ and $-\delta$ into a pair of squarer circuits and some of these errors get canceled in the accumulation stage that leads to minimized errors at the SAC output.

To simplify the partial product accumulation, various approximate adders are proposed in [16]–[18] with logic complexity reduction and carry prediction techniques. A framework to generate and explore the architectural space of approximate multipliers using various elementary adder and multiplier blocks is presented in [19]. Approximate coarse-grained reconfigurable architectures proposed in [20] and [21] employs quality scalable approximate adders and multipliers. In recent literature, compressors are widely used to speedup the accumulation of partial products in high-speed multipliers. Few approximate designs of 4-2 compressors are proposed in [22] and [23], which are used in multiplier designs to reduce the power consumption and delay. In [24], an AND-OR approach is used to encode the partial products, then approximate adders and 4-2 compressors are used to accumulate these encoded partial products. An N-bit approximate multiplier is proposed in [25], which directly truncates $N/2$ LSB columns of a partial product array and uses approximate 4-2 compressor for the accumulation of partial products in the next $N/2$ LSB columns. Two approximate 4×4 multipliers (M1 and M2) are proposed in [26] using AND-OR encoding of partial products and various probability-based approximate 4-2 compressors for accumulation. Higher order multipliers are then designed by repeatedly using these approximate 4×4 multipliers blocks. Few approximate multipliers using approximate 2-1, 3-2, 4-2, 5-3, and 6-3 compressors are proposed in [27].

Booth encoding of input operands can be employed in partial product generation to simplify the partial product accumulation, but the complexity of encoder increases with high radix encoding. Therefore, to reduce the complexity of Booth multiplication, approximate radix-4 and radix-8 Booth multipliers are proposed in [28]–[30] by approximating partial products generation. Fast multipliers [31] can be designed by representing partial product rows with redundant binary (RB) format. Approximate radix-4 RB multiplier proposed in [32] generates approximate RB partial products and uses approximate RB 4-2 compressors for the accumulation. An approximate multiplier with minimal error bias that prevents the accumulation of errors during successive approximate multiplications is proposed in [33]. This multiplier is designed based on a linearly approximated logarithmic multiplier with a unique error-reduction method. An approximate Multiplier Accumulator with Internal SelfHealing (MACISH) is proposed in [34]. The multiplier in this MAC introduced either $+\delta$ or $-\delta$ error into the 2×2 size multiplier subblocks and some of these errors are canceled with each other to achieve near-to-zero average error.

The ongoing demand for machine learning applications and their inherent error-tolerant properties have led to enormous research on approximate hardware accelerators for deep neural networks (DNNs). Since adder and multiplier are the basic building blocks of these accelerators, various acceleration methods have been proposed in the recent literature [35]–[38] to efficiently utilize the available approximate adders and multipliers. In modern communication systems, graphic processing units, high-performance co-processors, and DSPs, it is desirable to have a high speed squarer circuit. Therefore, this article proposes three approximate squarer circuits with an approximation of partial product generation and accumulation. A novel Approximate Partial Product Generator for Squarer (APPGS) that reduces hardware complexity of partial product generation is proposed in Section III.

III. Proposed Approximate Partial Product Generator for Squarer

A brief overview of the operation of conventional Radix-4 Booth folding encoding squarer [3] is provided at the beginning of this section. Later, APPGS is proposed by approximating the logic equation that generates partial products of this conventional squarer.

Consider an N-bit binary signed number X, which is represented using two's complement notation as

$$X = -x_{N-1}2^{N-1} + x_{N-2}2^{N-2} + \cdots + x_1 2^1 + x_0 2^0. \quad (1)$$

Using radix-4 Booth encoding, X can be represented as

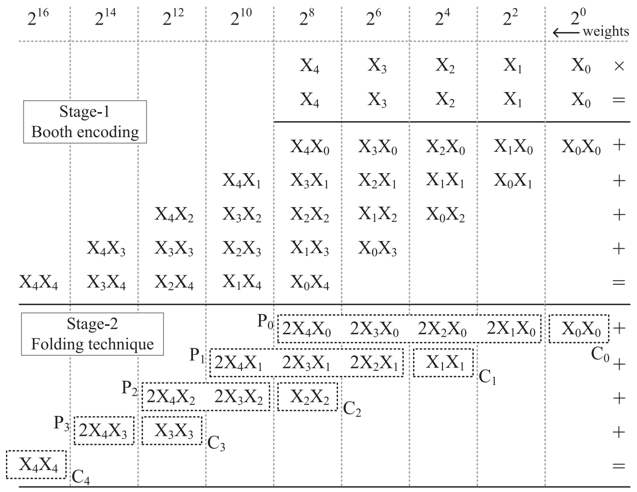
$$X = X_{\frac{N}{2}-1}2^{N-2} + X_{\frac{N}{2}-2}2^{N-4} + \cdots + X_1 2^2 + X_0 2^0 \quad (2)$$

where $X_i = -2x_{2i+1} + x_{2i} + x_{2i-1}$ (assuming $x_{-1} = 0$) are the radix-4 Booth encoded digits of X. For example, when $N = 10$, the square of X can be computed using Booth folding encoding technique as shown in Fig. 1. Stage-1 shows the partial products obtained using Booth encoding and Stage-2 shows the simplified partial product array using the folding technique. The square of X can be represented as

$$\begin{aligned}
X^2 &= (2X_4 2^8 + 2X_3 2^6 + 2X_2 2^4 + 2X_1 2^2)X_0 2^0 + X_0 X_0 2^0 \\
&\quad + (2X_4 2^6 + 2X_3 2^4 + 2X_2 2^2)X_1 2^4 + X_1 X_1 2^4 \\
&\quad + (2X_4 2^4 + 2X_3 2^2)X_2 2^8 + X_2 X_2 2^8 \\
&\quad + (2X_4 2^4)X_3 2^{12} + X_3 X_3 2^{12} + X_4 X_4 2^{16} \quad (3)\\
&= (P_0 2^3 + C_0)2^0 + (P_1 2^3 + C_1)2^4 + (P_2 2^3 + C_2)2^8 \\
&\quad + (P_3 2^3 + C_3)2^{12} + C_4 2^{16}. \quad (4)
\end{aligned}$$

In general for an N-bit squarer

$$X^2 = C_{\frac{N}{2}-1}2^{4(\frac{N}{2}-1)} + \sum_{i=0}^{\frac{N}{2}-2}(P_i 2^3 + C_i)2^{4i} \quad (5)$$



2^{16} 2^{14} 2^{12} 2^{10} 2^8 2^6 2^4 2^2 2^0
weights
Stage-1
Booth encoding
X_4 X_3 X_2 X_1 X_0 ×
X_4 X_3 X_2 X_1 X_0 =
X_4X_0 X_3X_0 X_2X_0 X_1X_0 X_0X_0 +
X_4X_1 X_3X_1 X_2X_1 X_1X_1 X_0X_1 +
X_4X_2 X_3X_2 X_2X_2 X_1X_2 X_0X_2 +
X_4X_3 X_3X_3 X_2X_3 X_1X_3 X_0X_3 +
X_4X_4 X_3X_4 X_2X_4 X_1X_4 X_0X_4 =
Stage-2
Folding technique
P_0 $2X_4X_0$ $2X_3X_0$ $2X_2X_0$ $2X_1X_0$ X_0X_0 + C_0
P_1 $2X_4X_1$ $2X_3X_1$ $2X_2X_1$ X_1X_1 C_1 +
P_2 $2X_4X_2$ $2X_3X_2$ X_2X_2 C_2 +
P_3 $2X_4X_3$ X_3X_3 C_3 +
X_4X_4 C_4 =

Fig. 1. Partial products matrix for 10-bit squarer generated using radix-4 Booth encoding and folding techniques [3].

where

$$C_i = X_i \times X_i \quad \text{for } 0 \le i \le N/2 - 1 \quad (6)$$

$$P_i = W_i \times X_i + x_{2i+1} \times X_i \quad \text{for } 0 \le i \le (N/2 - 2) \quad (7)$$

$$W_i = -x_{N-1}2^{N-2i-3} + x_{N-2}2^{N-2i-4} + \cdots + x_{2i+2}2^0. \quad (8)$$

In the binary form of C_i, the middle bit $C_{i,1}$ is always equal to logic-0 as shown in Table I. The logic expressions to generate the remaining bits are given by the equations

$$C_{i,0} = x_{2i} \oplus x_{2i-1} \quad (9)$$

$$C_{i,2} = x_{2i+1}\overline{x_{2i}}\,\overline{x_{2i-1}} + \overline{x_{2i+1}}x_{2i}x_{2i-1}. \quad (10)$$

Equation (7) can be further simplified as follows.
1. When X_i is positive, x_{2i+1} is zero, then $P_i = W_i \times X_i$
2. When X_i is negative, x_{2i+1} is one, then
$P_i = -W_i \times |X_i| - |X_i| = (\overline{W_i} + 1) \times |X_i| - |X_i| = \overline{W_i} \times |X_i|$

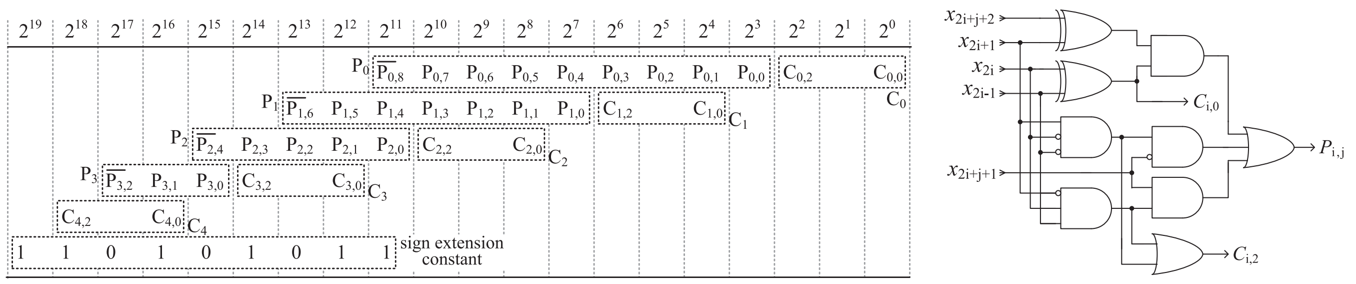
Thus, (7) can be written as

$$P_i = \begin{cases} W_i \times |X_i|, & \text{if } x_{2i+1} = 0 \\ \overline{W_i} \times |X_i|, & \text{if } x_{2i+1} = 1 \end{cases} \quad (11)$$

where $\overline{W_i}$ is obtained by complementing each individual bit of W_i. From (11), $P_i = 2W_i$ for $X_i = +2$, then each bit of P_i can be obtained by left shifting W_i by 1-bit position. Similarly, each individual bit $P_{i,j}$ of a partial product row P_i for different values of X_i can be obtained as shown in Table II. Partial product bits $P_{i,j}$ and $C_{i,j}$ of the 10-bit conventional Booth folding encoding squarer [3] are shown in Fig. 2(a). The sign extension prevention technique [39] is used for the accumulation of partial products by complementing the most significant bit (MSB) of each P_i term and then adding a constant as shown in Fig. 2(a).

A generalized logic equation to obtain the partial products $P_{i,j}$ is derived by writing five variable Karnaugh map (K-map) as shown in Fig. 3(a). The simplified logic equation for $P_{i,j}$ is given by the following equation:

$$\begin{aligned}
P_{i,j} &= \overline{x_{2i+j+1}}x_{2i+1}\overline{x_{2i}}\,\overline{x_{2i-1}} + x_{2i+j+1}\overline{x_{2i+1}}x_{2i}x_{2i-1} \\
&\quad + (x_{2i+j+2} \oplus x_{2i+1})(x_{2i} \oplus x_{2i-1}) \\
&\quad \text{for } 0 \le i \le (N/2 - 2),\ 0 \le j \le (N - 2i - 2). \quad (12)
\end{aligned}$$



2^{19} 2^{18} 2^{17} 2^{16} 2^{15} 2^{14} 2^{13} 2^{12} 2^{11} 2^{10} 2^{9} 2^{8} 2^{7} 2^{6} 2^{5} 2^{4} 2^{3} 2^{2} 2^{1} 2^{0}
P0 $\overline{P_{0,8}}$ $P_{0,7}$ $P_{0,6}$ $P_{0,5}$ $P_{0,4}$ $P_{0,3}$ $P_{0,2}$ $P_{0,1}$ $P_{0,0}$ $C_{0,2}$ $C_{0,0}$ C0
P1 $\overline{P_{1,6}}$ $P_{1,5}$ $P_{1,4}$ $P_{1,3}$ $P_{1,2}$ $P_{1,1}$ $P_{1,0}$ $C_{1,2}$ $C_{1,0}$ C1
P2 $\overline{P_{2,4}}$ $P_{2,3}$ $P_{2,2}$ $P_{2,1}$ $P_{2,0}$ $C_{2,2}$ $C_{2,0}$ C2
P3 $\overline{P_{3,2}}$ $P_{3,1}$ $P_{3,0}$ $C_{3,2}$ $C_{3,0}$ C3
$C_{4,2}$ $C_{4,0}$ C4
1 1 0 1 0 1 0 1 1 sign extension constant
(a)
x_{2i+j+2}
x_{2i+1}
x_{2i}
x_{2i-1}
x_{2i+j+1}
$C_{i,0}$
$P_{i,j}$
$C_{i,2}$
(b)

Fig. 2. Partial product matrix of 10-bit conventional Booth folding encoding squarer and partial product generator (a) Partial product matrix [3]. (b) Partial product bits generator.

TABLE I

TRUTH TABLE TO GENERATE $C_{i,j}$ BITS

x_{2i+1}	x_{2i}	x_{2i-1}	X_i	C_i	$C_{i,2}$	$C_{i,1}$	$C_{i,0}$
0	0	0	0	0	0	0	0
0	0	1	+1	1	0	0	1
0	1	0	+1	1	0	0	1
0	1	1	+2	4	1	0	0
1	0	0	-2	4	1	0	0
1	0	1	-1	1	0	0	1
1	1	0	-1	1	0	0	1
1	1	1	0	0	0	0	0

TABLE II

TRUTH TABLE TO GENERATE $P_{i,j}$ BITS

x_{2i+1}	x_{2i}	x_{2i-1}	X_i	P_i	$P_{i,j}$	
0	0	0	0	0	0	0
0	0	1	+1	W_i	$W_{i,j}$	x_{2i+j+2}
0	1	0	+1	W_i	$W_{i,j}$	x_{2i+j+2}
0	1	1	+2	$2W_i$	$W_{i,j-1}$	x_{2i+j+1}
1	0	0	-2	$2\overline{W_i}$	$\overline{W_{i,j-1}}$	$\overline{x_{2i+j+1}}$
1	0	1	-1	$\overline{W_i}$	$\overline{W_{i,j}}$	$\overline{x_{2i+j+2}}$
1	1	0	-1	$\overline{W_i}$	$\overline{W_{i,j}}$	$\overline{x_{2i+j+2}}$
1	1	1	0	0	0	0

The generalized partial products generator circuit is shown in Fig. 2(b). Equation (12) can be further simplified for the following cases:

$$\text{If } i = 0 \quad x_{2i-1} = 0 \tag{13}$$
$$\text{If } j = 0 \quad x_{2i+j+1} = x_{2i+1} \tag{14}$$
$$\text{If } j = N-2i-2 \quad x_{2i+j+1} = x_{2i+j+2}. \tag{15}$$

When $j = N - 2i - 2$, that is at the MSB of each row of partial products matrix, the $P_{i,j}$ is inverted. Therefore, from Fig. 3(b), the logic equation for $\overline{P_{i,j}}$ can be directly written as

$$\overline{P_{i,j}} = x_{2i+j+1}\overline{x_{2i}}\;\overline{x_{2i-1}} + \overline{x_{2i+j+1}}x_{2i}x_{2i-1} + x_{2i+j+1} \odot x_{2i+1}. \tag{16}$$

From Fig. 3, $P_{i,j} = 1$ for 12 out of 32 entries and $\overline{P_{i,j}} = 1$ for 10 out of 16 entries in their K-maps. The hardware complexity of partial products generator is reduced by approximating some of the entries in the K-maps that require significant part of the overall hardware. Therefore, the APPGS is proposed by separating (12) into two parts. One part of the equation is used as an approximated partial product bit $P'_{i,j}$ which reduces the complexity and increase the speed of partial products generation. The other part is used to recover the errors introduced due to approximated partial products in the form of error recovery bits $E_{i,j}$. Similarly, $\overline{P_{i,j}}$ is also separated into $\overline{P'_{i,j}}$ and $E_{i,j}$. The operation of the proposed APPGS is represented by the following equations:

When $0 < j < N - 2i - 2$

$$P'_{i,j} = (x_{2i+j+2} \oplus x_{2i+1})(x_{2i} \oplus x_{2i-1}) \tag{17}$$
$$E_{i,j} = \overline{x_{2i+j+1}}x_{2i+1}\overline{x_{2i}}\;\overline{x_{2i-1}} + x_{2i+j+1}\overline{x_{2i+1}}x_{2i}x_{2i-1} \tag{18}$$

When $j = N - 2i - 2$

$$\overline{P'_{i,j}} = x_{2i+j+1} \odot x_{2i+1} \tag{19}$$
$$E_{i,j} = x_{2i+j+1}\overline{x_{2i+1}}\;\overline{x_{2i}}\;\overline{x_{2i-1}} + \overline{x_{2i+j+1}}x_{2i+1}x_{2i}x_{2i-1}. \tag{20}$$

The $P'_{i,j}$ and $\overline{P'_{i,j}}$ are obtained by introducing four and two errors in the K-maps of $P_{i,j}$ and $\overline{P_{i,j}}$, respectively, as highlighted in Fig. 4. Therefore, in both the cases the probability of error in approximated partial product is $P_e = 1/8$.

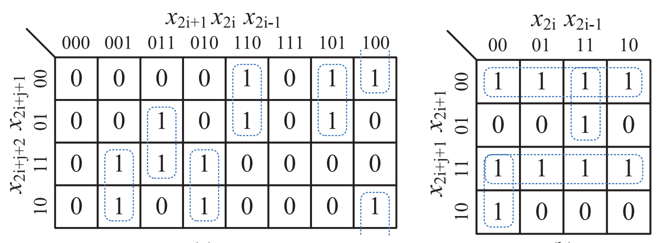


Fig. 3. K-maps to generate logic equations for accurate partial products. (a) $P_{i,j}$. (b) $\overline{P_{i,j}}$ (MSB).

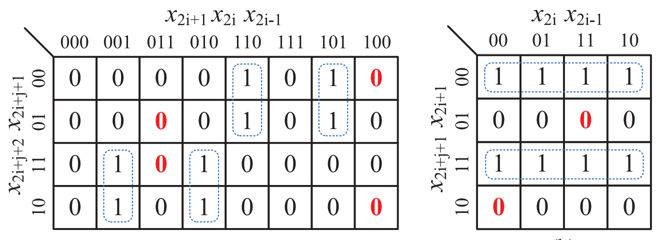


Fig. 4. K-maps to generate logic equations for approximate partial products. (a) $P'_{i,j}$. (b) $\overline{P'_{i,j}}$ (MSB).

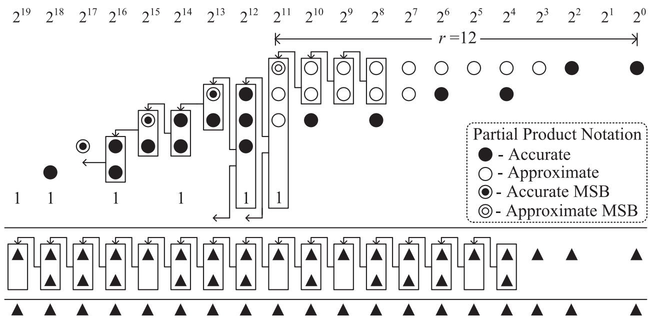


Fig. 5. Accumulation of partial products in ABS1 for $N = 10$ and $r = 12$.

Moreover, in all these approximations, logic-1 output of accurate partial product is modified to logic-0. Hence, accumulation of these approximated partial products produces approximate output (S) which is always smaller than accurate output (X^2). The error introduced due to the use of APPGS can be fully or partially recovered by using error recovery bits. For example, the error recovery bits can be accumulated to generate error recovery sum (E). Furthermore, S and E can be added to obtain the accurate squarer output as given by the following equation:

$$X^2 = S + E. \tag{21}$$

However, the generation of E in parallel to S and addition of S and E to get the accurate squarer output will increase the power consumption and delay as compared to an approximate squarer without any error recovery. Therefore, an error compensation method is proposed in Section IV-C to partially recover the error due to approximate partial products.

IV. Proposed Approximate Booth Squarers

Three approximate squarer architectures with different strategies for accumulating approximate partial products are proposed in this section to reduce the power consumption and delay while maintaining minimal errors at the output.

A. Approximate Booth Squarer-1 (ABS1)

The proposed ABS1 is designed by generating approximate partial products using proposed APPGS at only "r" number of LSB columns of the partial product matrix. The remaining partial products in MSB columns are generated using accurate equations (12)–(16). Accurate half adder, full adder, and 4-2 compressor [40] are used for the accumulation of all the partial products. Error compensation circuit is not included in the ABS1 design to get the maximum advantage in terms of hardware metrics such as power, delay, and area. The partial product accumulation of proposed ABS1 for $N = 10$ and $r = 12$ is shown in Fig. 5 with dot notation, where the arrow lines indicate the carry propagation path.

B. Approximate Booth Squarer-2 (ABS2)

The ABS2 is proposed to further improve the hardware metrics as compared to ABS1. Similar to ABS1, "r" number of LSB columns of the partial product matrix are generated using proposed APPGS. Moreover, instead of using accurate arithmetic blocks, popular approximate arithmetic blocks

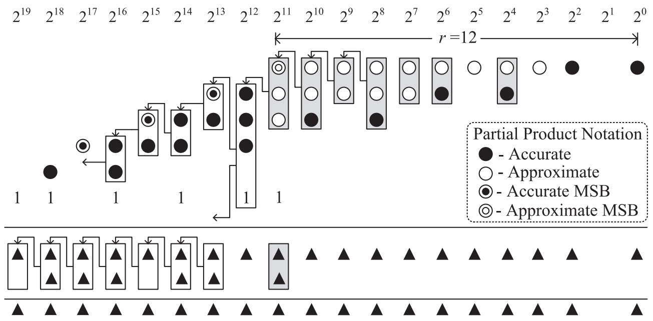


Fig. 6. Accumulation of partial products in ABS2 using approximate blocks for $N = 10$ and $r = 12$.

available in the literature [24], [25] are used to accumulate the partial products. The operation of these approximate blocks is given by the following equations:

Approximate Half Adder [24]:

$$\text{Sum} = a_1 + a_2 \quad \text{and} \quad \text{Carry} = 0 \tag{22}$$

Approximate Full Adder [24]:

$$\text{Sum} = (a_1 + a_2) \oplus a_3 \text{ and}$$
$$\text{Carry} = (a_1 + a_2)a_3 \tag{23}$$

Approximate 4-2 Compressor [25]:

$$\text{Sum} = (a_1 + a_2) \oplus (a_3 \oplus a_4) \text{ and}$$
$$\text{Carry} = (a_1 + a_2)(a_3 + a_4) + a_3a_4. \tag{24}$$

Here, a_1, a_2, a_3, and a_4 represent input bits to the approximate blocks. These approximate blocks are chosen because the absolute error difference between the erroneous outputs of these approximate blocks and the corresponding exact outputs is always 1, which is the minimum possible value. Moreover, in these approximate blocks, error at the output occurs only when both the inputs a_1 and a_2 becomes logic-1. Therefore, a method named as *Input Signal Rearrangement* is proposed in the following subsection to reduce the error at the output of the squarer that uses the considered approximate blocks. The accumulation of partial products of proposed ABS2 using these approximate blocks for $N = 10$ and $r = 12$ is shown in Fig. 6. The shaded rectangles represent the approximate adders and compressors.

1) Input Signal Rearrangement: As discussed, the considered approximate adders and compressors produce erroneous output only when both the inputs a_1 and a_2 becomes logic-1. Therefore, the input signals to these approximate blocks with least probability to become logic-1 can be connected to the input pins a_1 and a_2 to reduce the errors. The probability of output of an arithmetic block to become either logic-1 or logic-0 can be calculated using the concept of *signal probability* [41]. For example, assume the probabilities of inputs of the considered approximate half adder to become logic-1 as $P(a_1 = 1) = 1/3$ and $P(a_2 = 1) = 2/3$. Then, the probability of sum output of the approximate half adder can be calculated by writing the canonical form of sum as given below:

$$\text{Sum} = a_1\overline{a_2} + \overline{a_1}a_2 + a_1a_2 \tag{25}$$

TABLE III

PROBABILITIES OF PARTIAL PRODUCTS AND ERROR RECOVERY BITS TO BECOME LOGIC-1

Partial product	Probability
$P_{i,j}\ (j=0)$	1/4
$P_{i,j}\ (0<j<N-2i-2)$	3/8
$\overline{P_{i,j}}\ (j=N-2i-2)$	5/8
$C_{i,0}$	1/2
$C_{i,2}$	1/4
$P'_{i,j}\ (j<N-2i-2)$	1/4
$\overline{P'_{i,j}}\ (j=N-2i-2)$	1/2
$E_{i,0}$	0
$E_{i,j}\ (j\neq 0)$	1/8

$$P(\text{Sum}) = P(a_1)[1-P(a_2)]+[1-P(a_1)]P(a_2) + P(a_1)P(a_2) \quad (26)$$

$$P(\text{Sum}=1) = \frac{1}{3}\left[1-\frac{2}{3}\right]+\left[1-\frac{1}{3}\right]\frac{2}{3}+\frac{1}{3}\frac{2}{3}=\frac{7}{9} \quad (27)$$

$$P(\text{Sum}=0) = 1-P(\text{Sum}=1) = 1-\frac{7}{9}=\frac{2}{9}. \quad (28)$$

Without loss of generality, assume uniform distribution for the input bits $x_{N-1}, x_{N-2}, \ldots, x_1, x_0$ of the squarer. Then these input bits will have an equal probability of 1/2 to become either logic-1 or logic-0. As shown in Fig. 6, the inputs to an arithmetic block are either partial products or outputs of their preceding arithmetic blocks. Therefore, similar to (25)–(27), the probabilities of different partial products and error recovery bits to become logic-1 are calculated by writing their canonical form of the logical equation and these values are shown in Table III.

To reduce the error at the output of the squarer, the inputs to each approximate block are rearranged using the proposed ISR method which is shown in Fig. 7. In the ISR method, all the inputs are sorted according to the ascending order of their probabilities to become logic-1. The sorted inputs are connected to input ports $a_1, a_2, a_3, \ldots$ of the approximate block, respectively. Later, output probabilities of that approximate block are calculated by writing canonical form logic equation for each output signal. As the output of each block again becomes an input to the succeeding block, the output probabilities are calculated block by block starting from LSB and proceeding toward MSB. This process is repeated until the inputs to all the approximate blocks are rearranged according to Step 2 of the ISR method. All the probability calculations are performed before writing the HDL description and the rearranged input signal order for every approximate block is noted. Finally, in HDL, the modules of approximate blocks are instantiated in the top module of the design, where the port mapping is made according to the rearranged input signal order.

The input signal order obtained using probability analysis can be realized with the wiring logic alone (without any extra hardware) during the early stages of the design process. Therefore, this approach introduces only a one-time cost of calculation of these probabilities before writing the HDL description of a squarer for the given values of N and r. Hence, the proposed ISR method significantly reduces the overall error at the squarer output without any overhead on the hardware.

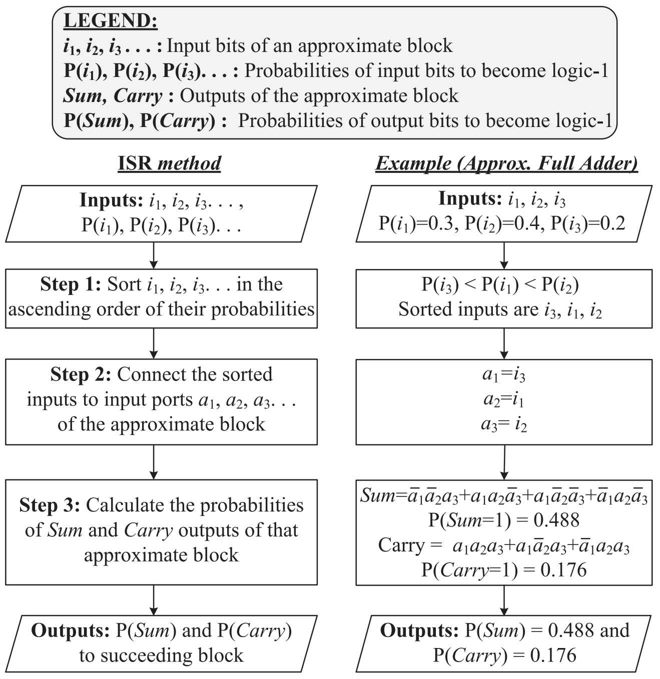

LEGEND:
$i_1, i_2, i_3\ldots$: Input bits of an approximate block
$P(i_1), P(i_2), P(i_3)\ldots$: Probabilities of input bits to become logic-1
Sum, Carry : Outputs of the approximate block
P(*Sum*), P(*Carry*) : Probabilities of output bits to become logic-1
ISR *method*
Example (Approx. Full Adder)
Inputs: $i_1, i_2, i_3\ldots$, $P(i_1), P(i_2), P(i_3)\ldots$
Step 1: Sort $i_1, i_2, i_3\ldots$ in the ascending order of their probabilities
Step 2: Connect the sorted inputs to input ports $a_1, a_2, a_3\ldots$ of the approximate block
Step 3: Calculate the probabilities of *Sum* and *Carry* outputs of that approximate block
Outputs: P(*Sum*) and P(*Carry*) to succeeding block
Inputs: i_1, i_2, i_3 $P(i_1)=0.3$, $P(i_2)=0.4$, $P(i_3)=0.2$
$P(i_3) < P(i_1) < P(i_2)$ Sorted inputs are i_3, i_1, i_2
$a_1=i_3$
$a_2=i_1$
$a_3=i_2$
$Sum=\bar{a}_1\bar{a}_2a_3+a_1a_2\bar{a}_3+a_1\bar{a}_2\bar{a}_3+\bar{a}_1a_2\bar{a}_3$
P(*Sum*=1) = 0.488
Carry = $a_1a_2a_3+a_1\bar{a}_2a_3+\bar{a}_1a_2a_3$
P(*Carry*=1) = 0.176
Outputs: P(*Sum*) = 0.488 and P(*Carry*) = 0.176

Fig. 7. Flow graph of ISR method with an example.

C. Approximate Booth Squarer-3 (ABS3)

The ABS3 is proposed by adding an error recovery module to the ABS2 design. For this purpose, a few error recovery bits are generated using (18) and (20), and they are accumulated to get a partial error recovery sum. From Table III, the probability of an error bit to become logic-1 is 1/8, which is very small. Therefore, an approximate error accumulation method is proposed to generate the partial error recovery sum. The proposed error accumulation method for an approximate squarer with $N = 32$ and $r = 25$ is shown in Fig. 8(a). Five MSB columns ($k = 5$) of error bits are used for the accumulation and remaining all columns of error bits are directly truncated. It can be noted that, as $P_{i,0} = P'_{i,0}$, the error bit $E_{i,0} = 0$. In Stage-1 of the accumulation of error bits, only 2-input and 3-input OR gates are used because the error bits have less probability to become logic-1. In Stage-2, accurate adders are used to get the final approximate sum of error bits. This sum vector is added to the ABS2 result at the corresponding bit positions. The overall architecture of ABS3 is shown in Fig. 8(b) for $N = 10$, $r = 12$, and $k = 4$. The shaded rectangles represent the approximate adders and compressors. A detailed analysis of simulation results is provided in Section V.

V. SIMULATION RESULTS

To analyze approximate designs in terms of accuracy, several error metrics such as error rate (ER), normalized mean error distance (NMED), and mean relative error distance (MRED) have been proposed in the literature [23], [42]. The worst case performance of the proposed squarers can be estimated using the maximum error (ME), which is defined

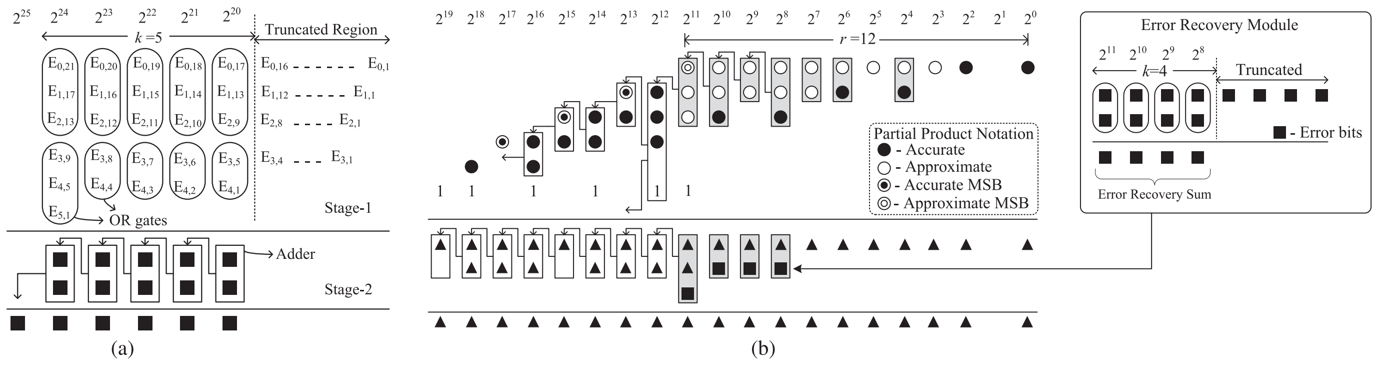


Fig. 8. Approximate error accumulation and architecture of ABS3 with error recovery module (a) Approximate accumulation of error bits. (b) Architecture of ABS3 for $N = 10$, $r = 12$, and $k = 4$.

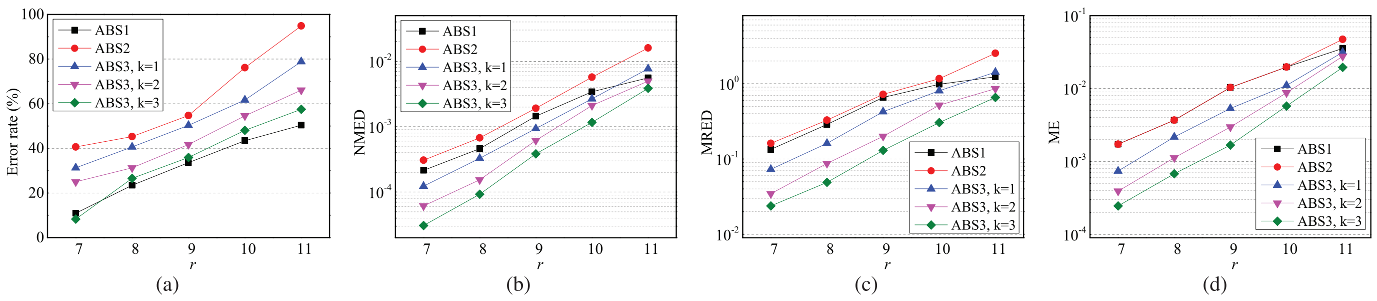


Fig. 9. Error metrics of proposed 8-bit squarer for different values of r and k (a) ER. (b) NMED. (c) MRED. (d) ME.

as a maximum error at the output of a squarer normalized by the maximum output of the accurate squarer. For an N-bit input, which is a signed number represented in 2's complement form, the maximum output of the squarer is $(2^{N-1})^2$. With the use of these four error metrics, an exhaustive error analysis is performed on the proposed squarers of sizes $N = 8$ and $N = 16$ using MATLAB.

A. Accuracy of the Proposed 8-bit Squarers

The accuracy of the proposed squarers is evaluated by varying r and k parameters. The considered error metrics for $r = 7, 8, 9, 10$, and 11; and $k = 1, 2$, and 3 are shown in Fig. 9. The ER of the proposed squarers increases with an increasing value of r. As shown in Fig. 9(a), the proposed ABS1 shows less ER compared to ABS2 and ABS3. The rise in ER in the case of ABS2 is due to the use of approximate adders and compressors for the accumulation of partial products. The ER of ABS3 gradually reduces with increasing values of k. On average, for the five values of r that are considered, the ER of ABS3 with $k = 3$ is reduced by 43.4% as compared to that of ABS2.

The NMED, MRED, and ME are increasing exponentially with increasing r. Therefore, these graphs are plotted on a logarithm scale as shown in Fig. 9(b)–(d). Even though ABS1 uses accurate adders and compressors for the accumulation of partial products, the rise in the error at the output of ABS2 with the use of approximate adders and compressors is very small as compared to ABS1 due to the use of the proposed ISR method. In the case of proposed ABS3, the values of NMED, MRED, and ME are small as compared to ABS1 and ABS2. Therefore, from Fig. 9, it can be inferred that the use of the error recovery module in ABS3 has significantly reduced the absolute error at the output of the squarer as compared to ABS1 and ABS2. The error rate of ABS3 is gradually decreasing with increasing values of k as compared to ABS2. On average, for the five values of r that are considered, the NMED, MRED, and ME of ABS3 with $k = 3$ are reduced by 77.6%, 76.3%, and 66.3%, respectively, as compared to ABS2.

B. Accuracy of the Proposed 16-bit Squarers

The accuracy of the proposed squarers is analyzed for $N = 16$ by varying the parameters r and k. The considered error metrics for $r = 12, 14, 16, 18$, and 20; and $k = 2, 4$, and 6 are shown in Fig. 10. Similar to the 8-bit input case, the error at the output of the proposed squarers increases with an increase in the value of r and the error of ABS3 decreases with an increase in the value of k. ABS2 produces larger errors among the three proposed squarers. On average, for the considered five values of r, the ER of ABS3 with $k = 6$ is reduced by 20.7% as compared to that of ABS2. The NMED, MRED, and ME of ABS3 with $k = 6$ are reduced by 90.1%, 95.5%, and 86.8%, respectively, as compared to those of ABS2.

To analyze the electrical performance of the proposed designs, hardware metrics such as power, delay, power delay product (PDP), and area are considered. All the proposed squarer designs are modeled using Verilog HDL and synthesized in Cadence Genus RTL compiler tool using 45-nm standard CMOS technology library. Subsequently, the Cadence Innovus tool is used for placement and routing. All the designs are synthesized with proper timing constraints. The functionality of designs is verified with one million randomly generated inputs. The compiler simulations are performed at

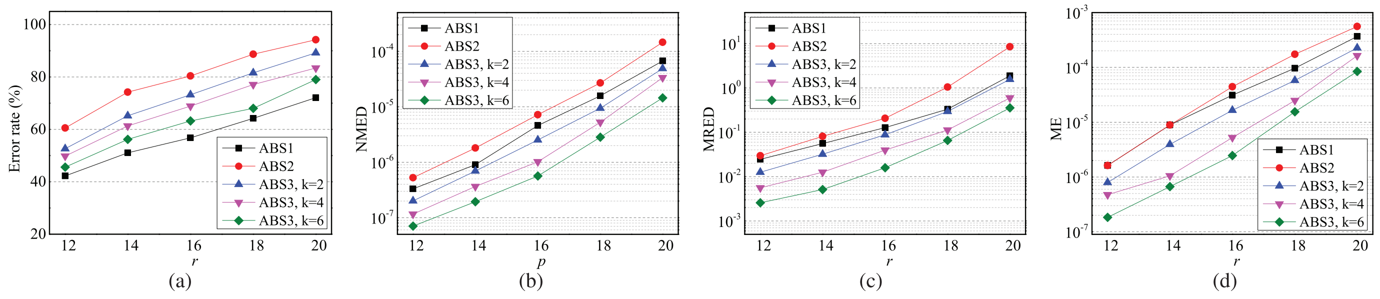


Fig. 10. Error metrics of proposed 16-bit squarer for different values of r and k. (a) ER. (b) NMED. (c) MRED. (d) ME.

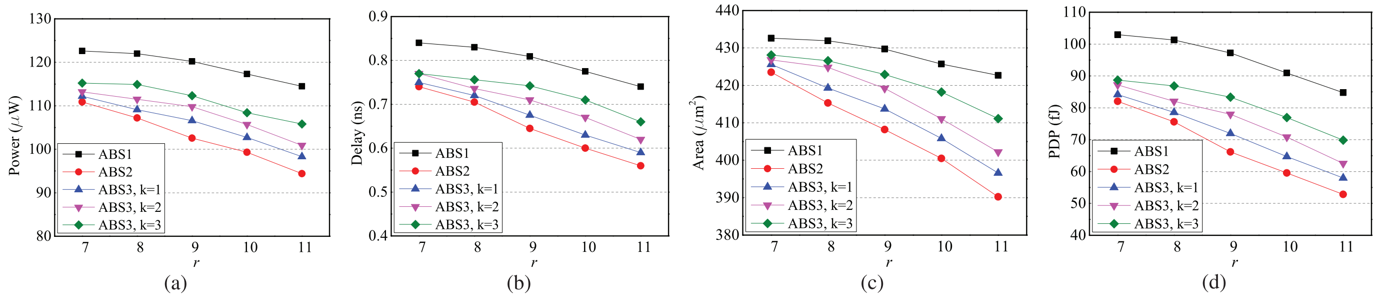


Fig. 11. Hardware metrics of proposed 8-bit squarer for different values of r and k. (a) Power. (b) Delay. (c) Area. (d) PDP.

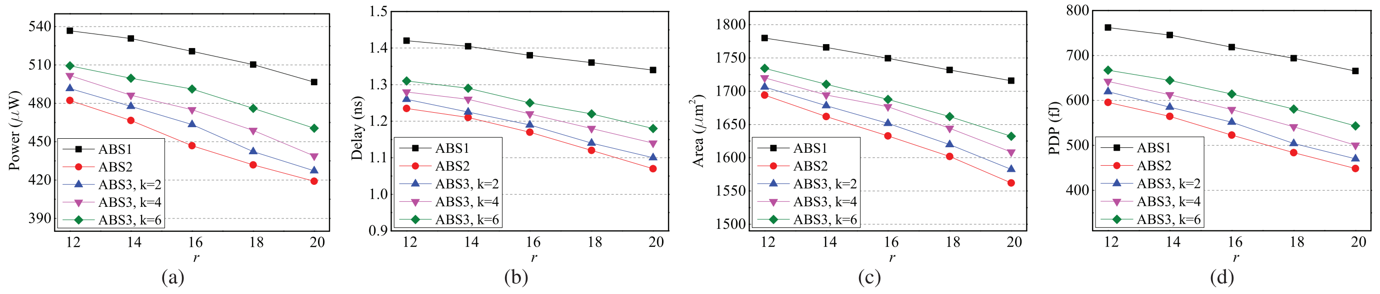


Fig. 12. Hardware metrics of proposed 16-bit squarer for different values of r and k. (a) Power. (b) Delay. (c) Area. (d) PDP.

room temperature with a supply voltage of 1 V and a clock frequency of 200 MHz. A detailed analysis of the simulation results for 8-bit and 16-bit squarers is provided in Sections V-C and V-D.

C. Simulation Results of Proposed 8-bit Squarer

The proposed squarers are analyzed in terms of hardware metrics by varying the parameters r and k. The simulation results for $r = 7, 8, 9,$ 10, and 11; and $k = 1, 2,$ and 3 are shown in Fig. 11. The power, delay, area, and PDP of the proposed squarers are decreasing with increasing values of r. ABS1 has the highest values for all the metrics as compared to ABS2 and ABS3. On average, ABS2 shows 13.8%, 18.6%, 4.9%, and 29.5% reduction in power, delay, area, and PDP, respectively, as compared to ABS1. The hardware metrics of ABS3 increases with an increase in the value of k, due to the increase in complexity of error recovery module. In the case of ABS3 with $k = 3$, on average, the power, delay, area, and PDP increases by 8.2%, 11.9%, 3.4%, and 20.7%, respectively, as compared to ABS2.

D. Simulation Results of Proposed 16-bit Squarer

The simulation results for $r = 12, 14, 16, 18,$ and 20; and $k = 2, 4,$ and 6 are shown in Fig. 12. Similar to 8-bit input case, all the hardware metrics decrease with increase in the value of r. ABS1 has the highest values for all the metrics as compared to ABS2 and ABS3. On an average, ABS2 shows 13.4%, 15.9%, 6.8%, and 27.1% reduction in power, delay, area, and PDP, respectively, as compared to ABS1. The ABS3 with $k = 6$ shows 8.5%, 7.7%, 3.4%, and 16.8% rise in power, delay, area, and PDP, respectively, as compared to ABS2.

As the accuracy metrics are increasing and hardware metrics are decreasing with increase in value of r, the PDP versus NMED graph is plotted to choose the optimized design among the proposed squarers. The PDP versus NMED graph for $N = 8$ is shown in Fig. 13. It can be observed that ABS1 takes high values for both the metrics. The designs with optimized values for both the metrics for $N = 8$ are highlighted in Fig. 13, which includes ABS3 with $r = 9$ and $k = 1$; ABS2 with $r = 8$; ABS3 with $r = 9$ and $k = 2$; and ABS3 with $r = 8$ and $k = 1$. Similarly, Fig. 14 shows the

TABLE IV
COMPARISON OF PROPOSED SQUARERS WITH EXISTING DESIGNS FOR $N = 16$

Type of Squarer	Design	Error Metrics				Hardware Metrics				Application	
		ER (%)	NMED ($\times 10^{-6}$)	MRED	ME ($\times 10^{-5}$)	Power (μW)	Delay (ns)	Area (μm^2)	PDP (fJ)	Euclidean distance	SNR (dB)
Proposed Approximate Booth Squarers	ABS1	56.86	4.5951	0.1273	3.1232	520.6	1.382	1649.4	719.46	0.746	26.35
	ABS2	80.91	7.1925	0.2065	4.4316	446.8	1.175	1532.7	524.99	0.887	25.71
	ABS3, k=2	72.44	2.5254	0.0876	1.6547	463.4	1.194	1551.6	553.30	0.492	29.90
	ABS3, k=4	69.32	1.0226	0.0396	0.5249	475.0	1.224	1576.6	581.40	0.346	31.34
	ABS3, k=6	62.70	0.5644	0.0158	0.2456	491.2	1.253	1587.8	615.47	0.274	32.45
	ABS1 (full recovery)	0	0	0	0	684.0	2.033	2215.4	1391.0	0	-
Accurate Squarers	Booth Folding [3]	0	0	0	0	602.5	1.776	1910.5	1070.1	0	-
	Folding & Merging [4]	0	0	0	0	764.2	2.515	2620.4	1922.0	0	-
	Folding & Merging [5]	0	0	0	0	753.4	2.486	2584.7	1872.9	0	-
Fixed Width Squarers (FWS) with error compensation	Booth FWS [9]	98.67	78.763	19.592	6.6142	407.3	1.087	1408.2	442.69	1.911	19.07
	Booth FWS [11]	99.24	82.467	22.612	4.1218	384.7	0.956	1346.0	367.82	1.919	19.07
	Booth FWS [13]	99.42	91.422	27.814	5.6760	396.6	1.004	1372.6	398.19	2.006	18.89
	Dadda FWS [12]	99.91	117.88	33.410	4.6671	502.4	1.601	1868.8	804.58	2.116	18.33
	Dadda FWS [14]	97.69	102.28	24.278	4.2631	486.6	1.565	1922.6	761.59	2.103	18.46
Pre-Computed Sums (PCS) & Quadratic approx. Squarers	PCS [6]	96.21	52.174	12.2426	13.294	582.6	1.621	1986.0	944.39	1.662	20.35
	PCS Approx-1 [7]	97.80	64.291	13.8642	22.321	546.4	1.362	1754.4	744.20	1.875	19.91
	PCS Approx-2 [7]	95.42	38.214	11.3146	15.171	560.7	1.534	1842.6	860.11	1.493	21.56
	Quadratic Approx. [8]	96.07	32.416	7.9425	8.216	611.8	1.676	1686.8	1025.4	1.418	21.70

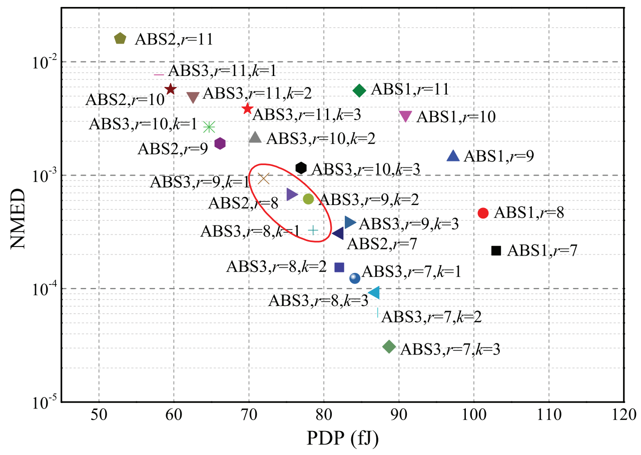


Fig. 13. PDP versus NMED of proposed approximate 8-bit squarers.

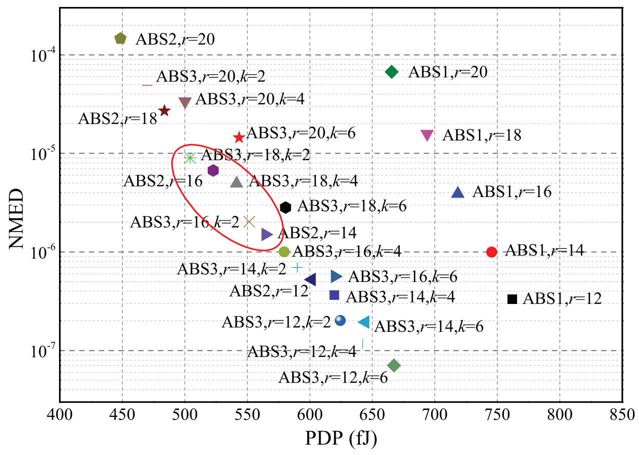


Fig. 14. PDP versus NMED of proposed approximate 16-bit squarers.

PDP versus NMED graph for $N = 16$. Likewise, the proposed ABS1 takes high values for both the metrics. The designs with optimized values for both the metrics for $N = 16$ are highlighted in Fig. 14, which includes ABS3 with $r = 18$ and $k = 2$; ABS2 with $r = 16$; ABS3 with $r = 18$ and $k = 4$; ABS3 with $r = 16$ and $k = 2$; and ABS2 with $r = 14$. The performance of the proposed squarers is compared with some of the existing designs of squarers in Section VI.

VI. COMPARISON WITH EXISTING DESIGNS

The accuracy and electrical performance of the proposed 16-bit squarers are compared with existing designs of accurate squarer with Booth Folding [3], accurate squarer with folding and merging [4], [5] and fixed width squarers (FWS) with Booth Folding architecture [9], [11], [13] and Dadda architecture [12], [14]. Along with these squarers, a few existing approximate squarer designs based on PCS with error compensation [6], [7] and quadratic approximation [8] are also considered for comparison. As the proposed squarers show optimized performance for $r = 16$, only designs with $r = 16$ are considered in this analysis. For a better comparison, in the case of Booth FWS [9], [11], 16 LSB columns of partial products are considered as TP, out of which two columns ($w = 2$) are dedicated for TP_{mj}. Similarly, in the case of Dadda FWS [12], 16 LSB columns are considered as LSP, out of which one column ($h = 1$) is dedicated for $\text{LSP}_{\text{major}}$ and one column for input correction (IC) part. In the design of Dadda FWS [14], for ECU, the 16th column in the partial product matrix is dedicated for TP_H and 15th column is used to generate signatures for error compensation. The Boolean equation-based squarers [6]–[8] are implemented using basic logic gates.

The proposed squarers along with considered squarers from the literature are simulated and the results are shown in Table IV. As compared to conventional accurate Booth Folding squarer [3], the proposed ABS1 achieves a reduction

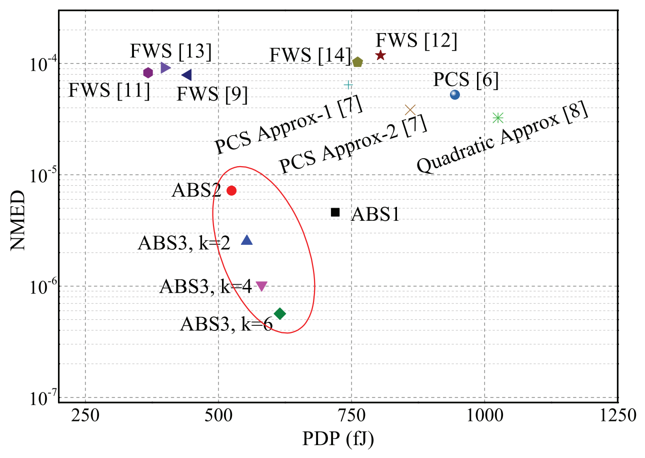


Fig. 15. PDP versus NMED of proposed approximate 16-bit squarers and various existing approximate designs.

of 13.6%, 22.2%, and 13.7% in power, delay, and area, respectively. ABS2 has power, delay, and area savings of 25.8%, 33.8%, and 19.8%, respectively. ABS3 with $k = 6$ has 18.5% reduction in power, 29.4% reduction in delay, and 16.9% reduction in area. The proposed designs show better performance in terms of accuracy as compared to all other existing designs. ABS1 is also simulated to fully recover the error by accumulating the error bits $E_{i,j}$ using accurate adders and compressors. The results show that ABS1 with full recovery mode has increased hardware complexity as compared to accurate squarer [3]. The fixed width squarers are consuming less power, delay, and area as compared to other designs, because they directly truncate the LSB columns of partial product matrix, but the error at the output of these squarers is significantly high compared to proposed designs. It is observed that the hardware complexity of Boolean equation-based designs [6]–[8] increases with an increase in input operand size.

The PDP versus NMED graph is shown in Fig. 15 for the better comparison of proposed designs with existing approximate designs in terms of electrical performance and accuracy. The Booth FWS squarers have smaller PDP but larger NMED. As highlighted, the proposed designs provide optimized performance for both the metrics. Out of the three proposed designs, the ABS3 has the least NMED and PDP.

VII. Application: Square Law Demodulator

In this section, the efficacy of the proposed squarers is analyzed by utilizing them in SLD [43], which is used in the field of telecommunication for the demodulation of AM signals. The SLD passes an AM signal through a squaring circuit and passes the resulting signal through a low-pass filter (LPF) to suppress the higher order frequency components. The output of SLD is a replica of the original message signal.

Consider a sinusoidal carrier signal $c(t)$ given by the following equation:

$$c(t) = A_c\cos(2\pi f_c t) \tag{29}$$

where A_c and f_c are the amplitude and frequency of the carrier signal. Let the message signal (baseband signal) be denoted by $m(t)$. Then the AM signal can be described as a function of time given by the following equation:

$$s(t) = A_c[1 + \mu m(t)]\cos(2\pi f_c t) \tag{30}$$

where μ is called modulation index, whose value should be selected such that $\mu m(t)$ lies in the interval $(-1,1)$. The square of $s(t)$ can be written as

$$\begin{aligned} s^2(t) &= A_c^2[1 + \mu m(t)]^2\cos^2(2\pi f_c t)\\ &= A_c^2[1 + \mu m(t)]^2\left[\frac{1 + \cos 4\pi f_c t}{2}\right]\\ &= \frac{A_c^2}{2}[1 + \mu m(t)]^2 + \frac{A_c^2}{2}[1 + \mu m(t)]^2\cos 4\pi f_c t. \end{aligned} \tag{31}$$

This signal passes through an LPF to suppress the terms whose spectral components are located around $2f_c$. The output of the LPF is given to square root block and the dc terms are subtracted to obtain the demodulated message signal.

To evaluate the performance of the proposed approximate squarers and other existing squarers, a Simulink model of an AM modulator and demodulator is designed as shown in Fig. 16. All the squarer designs are implemented as MATLAB functions and used to find the square of $s(t)$. All the designs are implemented with the same specifications, which are used for comparison in Section VI. A scaling factor of 2^{14} is used for converting floating point numbers to 16-bit integers. A carrier frequency, $f_c = 1$ kHz and a peak amplitude, $A_c = 1$ V are used in this analysis. A cosine signal with 50-Hz frequency is considered as a message signal. For effective analysis of the error due to approximate squarer, a sampling frequency of 20 kHz is used, so that the quantization noise is negligible. A 5th order Butterworth low pass filter from DSP System Toolbox is used to drop the higher frequency terms. Fig. 17 shows the output waveforms generated by SLD when the proposed approximate squarers are used for the squaring operation. It can be observed that ABS3 with $k = 2$ produces better results compared to ABS1 and ABS2. As k increases, quality of output waveform increases, but PDP and area of the squarer also increases. To measure the difference between reconstructed message signals using accurate and approximate designs, the Euclidean distance is calculated using the equation

$$||v-u|| = \sqrt{(v_1-u_1)^2+(v_2-u_2)^2+\cdots+(v_n-u_n)^2} \tag{32}$$

where $v_1, v_2, \ldots, v_n$ and $u_1, u_2, \ldots, u_n$ are samples of the output waveform generated by SLD using accurate and approximate designs. Similarly, the SNR [44] is also calculated using the equation

$$\text{SNR} = 10\log_{10}\left(\frac{P_{\text{signal}}}{P_{\text{noise}}}\right)\text{dB} = 10\log_{10}\left(\frac{E\left[A_{\text{signal}}^2\right]}{\sigma_{\text{noise}}^2}\right)\text{dB} \tag{33}$$

where P and A are power and amplitude of the output waveforms, respectively. $E[*]$ represents expected value or mean value and σ^2 represents variance. The noise samples are obtained by calculating the absolute difference between amplitudes of accurate and approximate output samples. The Euclidean distance and SNR of output waveforms generated by SLD with proposed squarers and all the considered FWS and approximate squarers are calculated for one full cycle of

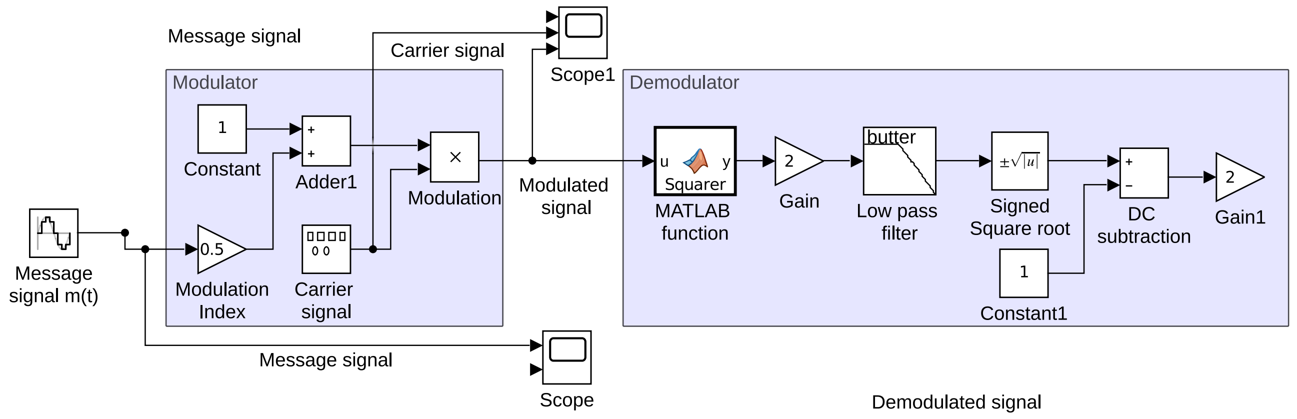


Fig. 16. Simulink setup used for the amplitude modulation and demodulation.

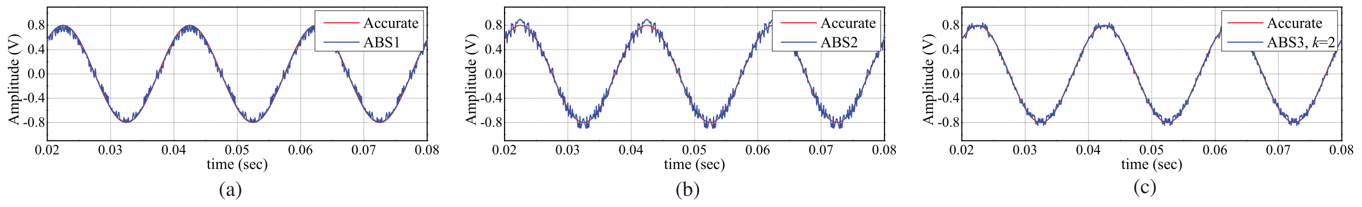


Fig. 17. Demodulated AM signals using proposed approximate designs.

TABLE V
RESOURCE UTILIZATION AND DELAY OF THE DEMODULATOR

Squarer design	LUTs	Slices	FFs	DSPs	Max. delay (ns)
Accurate [3]	428	161	248	3	21.66
ABS1	396	135	248	3	19.02
ABS2	382	127	248	3	18.30
ABS3, $k = 2$	385	129	248	3	18.49
ABS3, $k = 4$	388	130	248	3	18.66
ABS3, $k = 6$	390	130	248	3	18.78

message signal and shown in Table IV. It can be observed that the proposed approximate squarers perform better than the other designs by producing outputs with smaller Euclidean distance and higher SNR values.

A. Hardware Implementation

The SLD designs with proposed approximate squarers have been synthesized using Vivado design suite v2019.1 tool. Modulated signal samples generated from MATLAB HDL verifier are used in the test bench for functional verification. Finally, the demodulator designs are tested on Xilinx Spartan SP701 FPGA evaluation board with part number XC7S100. The resource utilization and critical path delays of the demodulator with accurate and proposed approximate squarers are summarized in Table V. The reported values include the hardware corresponding to the squarer block and other blocks that remains the same in all the demodulator designs.

The SLD application is considered in this article to validate the usage of proposed squarers in real-time applications. In the applications such as calculation of Euclidean norm among pixels for graphic processors, vector median filtering of color images [45] and rectangular to polar conversions in DSPs, the proposed squarers show a significant advantage in terms of hardware since squaring is the main operation in these types of applications.

VIII. Conclusion

This article proposed an approximate partial product generator to produce partial product bits in the squarers. Three approximate squarer designs are proposed to reduce the hardware complexity of squarer. The first proposed design uses accurate adders and compressors for the accumulation of approximate partial products and the second proposed design uses approximate adders and compressors for accumulation. The third design uses an error recovery module to improve the accuracy of the squarer output. The proposed designs along with existing designs of squarers are simulated using 45-nm CMOS technology. The proposed designs are analyzed by varying the number of columns (r) of partial products which are accumulated using approximate adders and compressors. The analysis proved that the proposed designs have optimized results for both hardware and accuracy parameters. Finally, the efficacy of proposed squarers is verified by employing them in SLD application.

References

[1] K. E. Wires, M. J. Schulte, L. P. Marquette, and P. I. Balzola, "Combined unsigned and two's complement squarers," in *Proc. 33rd Asilomar Conf. Signals, Syst., Comput.*, 1999, pp. 1215–1219.

[2] R. K. Kolagotla, W. R. Griesbach, and H. R. Srinivas, "VLSI implementation of 350 MHz 0.35 μm 8 bit merged squarer," *Electron. Lett.*, vol. 34, no. 1, pp. 47–48, Jan. 1998.

[3] A. G. M. Strollo and D. De Caro, "Booth folding encoding for high performance squarer circuits," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process*, vol. 50, no. 5, pp. 250–254, May 2003.

[4] S. Bui and J. E. Stine, "Additional optimizations for parallel squarer units," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Melbourne, VIC, Australia, Jun. 2014, pp. 361–364.

[5] A. Banerjee and D. K. Das, "A new squarer design with reduced area and delay," *IET Comput. Digit. Techn.*, vol. 10, no. 5, pp. 205–214, Sep. 2016.

[6] M.-H. Sheu and S.-H. Lin, "Fast compensative design approach for the approximate squaring function," *IEEE J. Solid-State Circuits*, vol. 37, no. 1, pp. 95–97, Jan. 2002.

[7] J. M. P. Langlois and D. Al-Khalili, "Carry-free approximate squaring functions with O(n) complexity and O(1) delay," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 5, pp. 374–378, May 2006.

[8] Z. Xun, J. Weiwei, and J. Dongming, "Circuit design of an improved approximate squaring function," in *Proc. 6th Int. Conf. ASIC*, Shanghai, China, 2005, pp. 1082–1084.

[9] K.-J. Cho and J.-G. Chung, "Low error fixed-width two's complement squarer design using booth-folding technique," *IET Comput. Digit. Techn.*, vol. 1, no. 4, p. 414, 2007.

[10] Y.-H. Chen, "Low-cost fixed-width squarer by using probability-compensated circuit," *Electron. Lett.*, vol. 50, no. 11, pp. 795–797, May 2014.

[11] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo, "Truncated squarer with minimum mean-square error," *Microelectron. J.*, vol. 45, no. 6, pp. 799–804, Jun. 2014.

[12] Y.-H. Chen, "Area-efficient fixed-width squarer with dynamic error-compensation circuit," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 9, pp. 851–855, Sep. 2015.

[13] S. R. Datla, M. A. Thornton, and D. W. Matula, "A low power high performance Radix-4 approximate squaring circuit," in *Proc. 20th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors*, Boston, MA, USA, Jul. 2009, pp. 91–97.

[14] B. Shao and P. Li, "Array-based approximate arithmetic computing: A general model and applications to multiplier and squarer design," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 4, pp. 1081–1090, Apr. 2015.

[15] G. A. Gillani, M. A. Hanif, M. Krone, S. H. Gerez, M. Shafique, and A. B. J. Kokkeler, "SquASH: Approximate square-accumulate with self-healing," *IEEE Access*, vol. 6, pp. 49112–49128, 2018.

[16] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013.

[17] W. Xu, S. S. Sapatnekar, and J. Hu, "A simple yet efficient accuracy-configurable adder design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 6, pp. 1112–1125, Jun. 2018.

[18] F. Ebrahimi-Azandaryani, O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Block-based carry speculative approximate adder for energy-efficient applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 1, pp. 137–141, Jan. 2020, doi: 10.1109/TCSII.2019.2901060.

[19] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proc. 35th Int. Conf. Comput.-Aided Des. (ICCAD)*, Austin, TX, USA, 2016, pp. 1–8.

[20] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, and M. Shafique, "Toward approximate computing for coarse-grained reconfigurable architectures," *IEEE Micro*, vol. 38, no. 6, pp. 63–72, Nov. 2018.

[21] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, and M. Shafique, "X-CGRA: An energy-efficient approximate coarse-grained reconfigurable architecture," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, to be published, doi: 10.1109/TCAD.2019.2937738.

[22] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 984–994, Apr. 2015.

[23] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Dual-quality 4:2 compressors for utilizing in dynamic accuracy configurable multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 4, pp. 1352–1361, Apr. 2017.

[24] S. Venkatachalam and S.-B. Ko, "Design of power and area efficient approximate multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 5, pp. 1782–1786, May 2017.

[25] M. Ha and S. Lee, "Multipliers with approximate 4–2 compressors and error recovery modules," *IEEE Embedded Syst. Lett.*, vol. 10, no. 1, pp. 6–9, Mar. 2018.

[26] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, "Low-power approximate multipliers using encoded partial products and approximate compressors," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 404–416, Sep. 2018.

[27] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra, "Approximate multipliers based on new approximate compressors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4169–4182, Dec. 2018.

[28] H. Jiang, J. Han, F. Qiao, and F. Lombardi, "Approximate radix-8 booth multipliers for low-power and high-performance operation," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2638–2644, Aug. 2016.

[29] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 booth multipliers for error-tolerant computing," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1435–1441, Aug. 2017.

[30] S. Venkatachalam, E. Adams, H. J. Lee, and S.-B. Ko, "Design and analysis of area and power efficient approximate booth multipliers," *IEEE Trans. Comput.*, vol. 68, no. 11, pp. 1697–1703, Nov. 2019.

[31] X. Cui, W. Liu, X. Chen, E. E. Swartzlander, and F. Lombardi, "A modified partial product generator for redundant binary multipliers," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1165–1171, Apr. 2016.

[32] W. Liu *et al.*, "Design and analysis of approximate redundant binary multipliers," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 804–819, Jun. 2019.

[33] H. Saadat, H. Bokhari, and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2623–2635, Nov. 2018.

[34] G. A. Gillani, M. A. Hanif, B. Verstoep, S. H. Gerez, M. Shafique, and A. B. J. Kokkeler, "MACISH: Designing approximate MAC accelerators with internal-self-healing," *IEEE Access*, vol. 7, pp. 77142–77160, May 2019.

[35] M. A. Hanif, F. Khalid, and M. Shafique, "CANN: Curable approximations for high-performance deep neural network accelerators," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, 2019, pp. 1–6.

[36] V. Mrazek, M. A. Hanif, Z. Vasicek, L. Sekanina, and M. Shafique, "autoAx: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, Jun. 2019, pp. 1–6.

[37] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Westminster, CO, USA, Nov. 2019, pp. 1–8.

[38] A. Marchisio, V. Mrazek, M. A. Hanif, and M. Shafique, "ReD-CaNe: A systematic methodology for resilience analysis and design of capsule networks under approximations," in *Proc. 23rd Design, Autom. Test Eur. (DATE)*, Grenoble, France, 2020. [Online]. Available: https://arxiv.org/abs/1912.00700

[39] J. Fadavi-Ardekani, "M*N booth encoded multiplier generator using optimized Wallace trees," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 1, no. 2, pp. 120–125, Jun. 1993.

[40] C.-H. Chang, J. Gu, and M. Zhang, "Ultra low-voltage low-power CMOS 4–2 and 5–2 compressors for fast arithmetic circuits," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 10, pp. 1985–1997, Oct. 2004.

[41] K. Roy and S. C. Prasad, *Low-Power CMOS VLSI Circuit Design*. Hoboken, NJ, USA: Wiley, 2009.

[42] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1760–1771, Sep. 2013.

[43] H. Taub, D. L. Schilling, and G. Saha, *Priciples of Communication Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, 2008.

[44] D. H. Johnson, "Signal-to-noise ratio," *Scholarpedia*, vol. 1, no. 12, p. 2088, 2006, doi: 10.4249/scholarpedia.2088.

[45] C. Seol and K. Cheun, "A low complexity Euclidean norm approximation," *IEEE Trans. Signal Process.*, vol. 56, no. 4, pp. 1721–1726, Apr. 2008.