Exact and Approximate Squarers for Error-Tolerant Applications

Exact and Approximate Squarers for Error-Tolerant Applications

Ke Chen, *Member, IEEE,* Chenyu Xu, Haroon Waris, Weiqiang Liu, *Senior Member, IEEE,* Paolo Montuschi, *Fellow, IEEE,* and Fabrizio Lombardi, *Life Fellow, IEEE*

Abstract—Approximate computing is considered an innovative paradigm with wide applications to high performance and low power systems. These applications have relaxed requirements for accuracy, so they can tolerate errors in results and achieve high performance. In approximate computing, multipliers have been widely studied, but squarers (as similar schemes) have not received much attention. In this paper, an accurate squarer is designed based on a Radix-8 Booth-folding square algorithm to reduce the number of partial products and the depth of the partial product array. Several approximate squarers (R8AS1, R8AS2 and R8AS3) are proposed based on the exact squarer to reduce power and delay. Two approximate partial product generators are also designed to simplify the Radix-8 Booth square encoder in R8AS1 and R8AS2. In addition, approximate compressors with compensation are used in the partial product compression stage to reduce additional area and power consumption in R8AS3. Synthesis results for power, area, and delay at 28nm CMOS technology are presented. Compared with designs in the technical literature with the same accuracy, the proposed 16-bit designs reduce the PDP by 37%; in general, the PDP is decreased by up to 51%. Finally, the proposed approximate squarers are implemented in a square-law detector as a communication application and achieve an SNR close to 30dB. Also, the three proposed approximate squarers are applied to the k-means clustering algorithm for machine learning to accomplish high performance in classification.

Index Terms—Radix-8 Booth-folding square algorithm, low power, approximate squarers, approximate compressors, square-law detector, k-means clustering.

1 Introduction

SQUARE operation is widely used in digital signal processing, such as error correction, image compression, vector quantization, equalization, and co-processors [1] [2]. Energy efficiency in hardware has become a significant challenge in today's DSP systems and machine learning. In addition, with an increase in workload for modern DSP applications, low power and high performance are strict requirement. However, it is difficult to further reduce power consumption and delay when results are required to be fully accurate for squarers. Furthermore, most applications are error-tolerant, so relatively high performance can be maintained even though errors are introduced [3]. Therefore, it is acceptable to use approximate computing to improve performance of squarers and reduce power with some accuracy loss. The square operation is commonly performed by a conventional multiplier, using the same multiplicand and multiplier, however, a multiplier accounts for a significant redundancy because it is designed for two independent operands, so leading to significant power and area consumption [4]. As shown in [5] [6], the use of specific folding squares instead of multipliers can reduce power by more than 50% and increase performance; therefore, it is also appropriate to design a specialized approximate squarer.

- *K. Chen, C. Xu, H. Waris and W. Liu are with the College of Electronic Information and Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 210016, China. E-mail: {chen.ke, xcy, haroonwaris, liuweiqiang}@nuaa.edu.cn.*
- *Paolo Montuschi is with the DAUIN, Politecnico di Torino, 10129 Turin, Italy. E-mail: paolo.montuschi@polito.it.*
- *F. Lombardi is with Department of Electrical and Computer Engineering, Northeastern University, Boston, United States, 02115. E-mail: lombardi@ece.neu.edu.*

Several schemes have been proposed for exact squarers design. In [7], a novel squaring algorithm is proposed, it precomputes the partial product coordination multiplication to improve the square operation. In [8], two new bit-serial squarers for long numbers of LSB first form are proposed; the computational efficiency is 50% higher than for the traditional squarer. A novel left-to-right leading digit first dual recoding is proposed in [9]; it generates an array with non-negative partial squares. The size of its partial product array is half of the conventional Radix-4 and Radix-8 multiplier designs.

The hardware of the squarer is similar to the multiplier, which including partial product generation, partial product compressor, and the final accumulation. Therefore, the conventional Radix-4 Booth folding method can be applied to implementing squarer [10]. In addition, for squarer's approximation, techniques used for a approximate multiplier such as approximate Booth encoding [11] can also be applied to a squarer. Based on the approximation techniques, the approximate squarer can be classified as truncated approximate squarer [12] - [16] and non-truncated approximate squarers [17]. For truncated approximate squarers, the computing accuracy is significantly reduced. On the other hands, research on non-truncated approximate squarers is inadequate, especially for approximate squarers based on Booth coding. [17] proposed Radix-4 Booth-folding Approximate squarers with input signal rearrangement method. However, the Radix-4 algorithm can only reduce the partial products by almost 50%.To the best of the authors' knowledge, a Radix-8 Booth-folding algorithm for squarer has not yet been proposed. Although the partial product generator of a Radix-8 Booth-folding encoding is more complex than

Radix-4, a Radix-8 Booth-folding encoding can reduce more partial products, and therefore the depth of the partial product array; fewer compressors are then needed for the partial product accumulation trees, as already shown by the hybrid low Radix Booth multipliers of [18].

In this paper, a Radix-8 Booth-folding square algorithm and a squarer based on Radix-8 Booth-folding encoding are proposed. The hardware for odd multiples such as $(\times 3)$ in the Radix-8 Booth-folding encoding are complex, so these terms are approximated to simplify the encoder circuit. In addition, when considering the trade-off between power consumption and accuracy, approximate 4-2 compressor, full and half adders are used in the partial product accumulate unit to further reduce power consumption. The main contributions of this work are summarized as follows:

1) The Radix-8 Booth-folding algorithm for squarer and Radix-8 Booth-folding encoding are proposed to reduce the number and height of the partial products, so reducing the number of adders used for partial product matrix compression.
2) Two approximate encoders are designed to simplify the circuit complexity of the Radix-8 Booth encoding. The proposed Radix-8 Booth folding encoding, two approximate encoders and approximate compressors with compensation are used in three approximate squarers (R8AS1, R8AS2 and R8AS3) to assess the trade-off between power and accuracy.
3) The proposed designs achieve 20% to 52% energy saving compared with the latest approximate squarer; they are used to test for the square-law detector in the process of amplitude modulation and demodulation. In addition, all approximate squarers are utilized to calculate the Euclidean distance in the k-means clustering algorithm for data classification.

The rest of this paper is organized as follows: An exact squarer based on Radix-8 Booth-folding encoding is introduced in Section II. In Section III, two approximate partial product generators are proposed according to the characteristics of the Radix-8 Booth encoder. Combined with approximate compressors, three approximate squarers are designed. In Section IV, the hardware simulation and error results of the approximate squarers are comprehensively analyzed and compared. Approximate squarers are used in communication and machine learning applications to verify their performance in Section V. Section VI concludes the paper.

2 Radix-8 Booth Folding Encoding

Like a multiplier, a squarer also consists of folding encoding, partial product generator (PPG), partial product matrix compression, and final accumulation. The significant difference of the Booth algorithm is that only one input is encoded in the multiplier, while in a squarer, both inputs must to be encoded. However, using Booth encoding circuits for both inputs make the partial product generator extremely complicated. Therefore, one of the inputs needs to be expanded. The derivation process of the Radix-8 Booth-folding squarer operation is as follows.

TABLE 1: Truth Table of $C_{i,j}$ for Radix-8 Booth Encoders

$a_{3i+2}a_{3i+1}a_{3i}a_{3i-1}$	A_i	C_i	$C_{i,4}$	$C_{i,3}$	$C_{i,2}$	$C_{i,1}$	$C_{i,0}$
0000	0	0	0	0	0	0	0
0001	+1	1	0	0	0	0	1
0010	+1	1	0	0	0	0	1
0011	+2	4	0	0	1	0	0
0100	+2	4	0	0	1	0	0
0101	+3	9	0	1	0	0	1
0110	+3	9	0	1	0	0	1
0111	+4	16	1	0	0	0	0
1000	-4	16	1	0	0	0	0
1001	-3	9	0	1	0	0	1
1010	-3	9	0	1	0	0	1
1011	-2	4	0	0	1	0	0
1100	-2	4	0	0	1	0	0
1101	-1	1	0	0	0	0	1
1110	-1	1	0	0	0	0	1
1111	0	0	0	0	0	0	0

Consider an N-bit signed binary integer A, given by:

$$A = -a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a^i 2^i \tag{1}$$

Let N=12, after being encoded by the Radix-8 Booth algorithm, A is represented as:

$$A = A_3 2^9 + A_2 2^6 + A_1 2^3 + A_0 2^0 \tag{2}$$

where $A_i = -4a_{3i+2} + 2a_{3i+1} + a_{3i} + a_{3i-1}$ and $a_{-1} = 0$. The result of the Radix-8 encoded A^2 can be written as :

$$\begin{aligned} A^2 =& 2A_3A_0 2^9 + 2A_2A_0 2^6 + 2A_1A_0 2^3 + A_0A_0 2^0 \\ &+ 2A_3A_1 2^{12} + 2A_2A_1 2^9 + A_1A_1 2^6 \\ &+ 2A_3A_2 2^{15} + A_2A_2 2^{12} \\ &+ A_3A_3 2^{18} \end{aligned} \tag{3}$$

By introducing two variables P_i and C_i, we obtain:

$$\begin{aligned} A^2 =& (P_0 2^4 + C_0)2^0 + (P_1 2^4 + C_1)2^6 + (P_2 2^4 + C_2)2^{12} \\ &+ C_3 2^{18} \end{aligned} \tag{4}$$

And by expanding to N-bit, where:

$$C_i = A_i \times A_i \qquad for\ 0 \leqslant i \leqslant \lceil N/3 \rceil - 1 \tag{5}$$

$$\begin{aligned} P_i &= A_i \sum_{t=i+1}^{N/3-1} A_t 2^{3t-3(i+1)} \\ &= A_i \times B_i + A_i \times a_{3i+2} \quad for\ 0 \leqslant i \leqslant \lceil N/3 \rceil - 2 \end{aligned} \tag{6}$$

and

$$B_i = -a_{N-1}2^{N-3i-4} + a_{N-2}2^{N-3i-5} + \ldots + a_{3i+3}2^0 \tag{7}$$

As per the Radix-8 Booth encoders of Table 1, the value of the encoded C_i = {0,1,4,9,16}. Then $C_{i,1}$ is '0' because none of the values among $C_{i,1}$ has a weight of 2. The Boolean function of a partial product C_i is given as follows:

$$C_{i,0} = a_{3i-1} \oplus a_{3i} \tag{8}$$

$$C_{i,2} = (a_{3i-1} \odot a_{3i})(a_{3i} \oplus a_{3i+1}) \tag{9}$$

$$C_{i,3} = (a_{3i-1} \oplus a_{3i})(a_{3i+1} \oplus a_{3i+2}) \tag{10}$$

$$C_{i,4} = (a_{3i-1} \odot a_{3i})(a_{3i} \odot a_{3i+1})(a_{3i+1} \oplus a_{3i+2}) \tag{11}$$

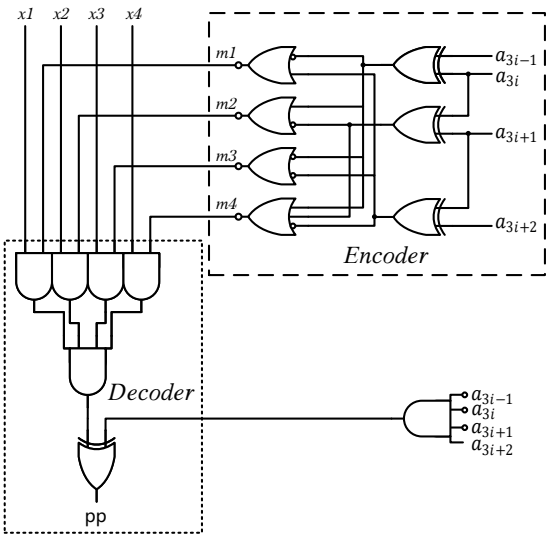

Fig. 1: The partial product generator of $P_{i,j}$.

Hence, a_{3i+2} is represented as the sign bit of A_i. Thus (6) can be rewritten as:

$$P_i = \begin{cases} B_i \times A_i & if A_i \geq 0 \\ \overline{B_i} \times |A_i| & if A_i \leq 0 \end{cases} \quad (12)$$

In (12), when A_i = 2, the circuit of $2B_i$ can be implemented by shifting one bit to the left and two bits for the A_i = 4 scenario. At the same time, $3B_i$ is calculated by adding B_i and $2B_i$, which requires additional circuits when A_i = 3. Therefore, the circuit overhead can be reduced by considering this scenario. The approximate method for $3B_i$ is discussed in the next section. As per the Radix-8 Booth encoders of Table I, A_i is given by:

$$|A_i| = 4m_{4,i} + 3m_{3,i} + 2m_{2,i} + m_{1,i} \quad (13)$$

where, the $m_{j,i}$ are the control signals generated by the encoder for calculating the value of A_i when $A_i = \pm j$. The one-hot code is composed of $\{m_{4,i}, m_{3,i}, m_{2,i}, m_{1,i}\}$,which means only one of the four signals can be 1 at the certain time. For example, if $m_{4,i}$ is 1, the other three signals are all 0. The value of $|A_i|$ will be 4. Controlled by $m_{j,i}$, the value of $|A_i|$ can be one of {1, 2, 3, 4} to produce the value of the second column in Table 1. By replacing (7) and (13) into (12), merging different items with the same weight, and defining the weight as the variable j, the following expression is found.

$$P_{i,j} = [a_{3i+j+1}m_{4,i} + (a_{3i+j+2} + a_{3i+j+1})m_{3,i} + a_{3i+j+2} \\ m_{2,i} + a_{3i+j+3}m_{1,i}] \oplus (a_{3i+2}\overline{a_{3i+1}a_{3i}a_{3i-1}}) \quad (14)$$

The ranges of i and j are respectively [0, N/3-2] and [0, N-3i-2]. Fig. 1 shows in schematic form the partial product generator circuit of $P_{i,j}$, that consists of the Radix-8 Booth encoder and decoder.

As per previous expressions, the partial product array of each term can be obtained. Fig. 2 graphically shows the $P_{i,j}$ and $C_{i,j}$ partial products arrangement of a Radix-8 Booth squarer with a 12-bit input. The most significant bit of Pi,j in each row has a negative weight and must be subtracted by taking the two's complement. The constant term is the sign extension bit, that uses the sign extension prevention technique proposed in [19] to add constants and reduces the number of constant ones by pre-calculating their sum and sending the carry into the next column.

TABLE 2: Truth Table of Approximate Radix-8 Booth Encoders for Squarer

$a_{3i+2}a_{3i+1}a_{3i}a_{3i-1}$	A_i	AR8E1	ED	AR8E2	ED
0000	0	0	0	0	0
0001	+1	+1	0	+1	0
0010	+1	+1	0	+1	0
0011	+2	+2	0	+2	0
0100	+2	+2	0	+2	0
0101	+3	+2	-1	+2	-1
0110	+3	+4	1	+2	-1
0111	+4	+4	0	+2	-2
1000	-4	-4	0	-2	2
1001	-3	-4	-1	-2	1
1010	-3	-2	1	-2	1
1011	-2	-2	0	-2	0
1100	-2	-2	0	-2	0
1101	-1	-1	0	-1	0
1110	-1	-1	0	-1	0
1111	0	0	0	0	0

3 Approximate Radix-8 Booth Squarer

In this section, two approximate Radix-8 encoders (AR8E1 and AR8E2) are proposed for analysis on the Radix-8 Booth-folding encoding squarer to reduce the area and power of the partial product generators. Furthermore, approximate compressors are introduced into the partial product compressor unit to reduce the overall hardware for the squarer; so three approximate squarer schemes are proposed. Different approximate partial product generators are used in R8AS1 and R8AS2 to compare the effects of the approximate encoding on the accuracy and power; R8AS3 uses approximate compressors in the compression tree based on the approximate partial product generator that saves most of the power and area. In the design of the approximate squarer, the more bits are involved in the approximate modules, the more power reduction is achieved while introducing larger errors.

3.1 R8AS1 Booth Squarer

Compared with the array-based folding squarer, the Booth encoding squarer's partial product generator circuit is more complicated. Therefore, an approximation to the partial product generator can result in improvements in the Booth encoding speed and reduce power consumption and area with acceptable errors.

In the Radix-8 encoding of the squarer, m_3 in the partial product generator is more complicated which is implemented by adding m_1 and m_2. In order to avoid the generation of m_3, the approximate Radix-8 encoder (AR8E1) are proposed. In AR8E1, the m_3 terms are approximated to m_2 and m_4 respectively to produce negative and positive errors. In detail, AR8E1 encode the '0101' and '0110' case to +2 and +4, and '1001' and '1010' cases to -4 and -2, respectively. In this way of design, the maximum error distance will be +1 or -1, and the average error distance is zero. The truth table is shown in Table 2.

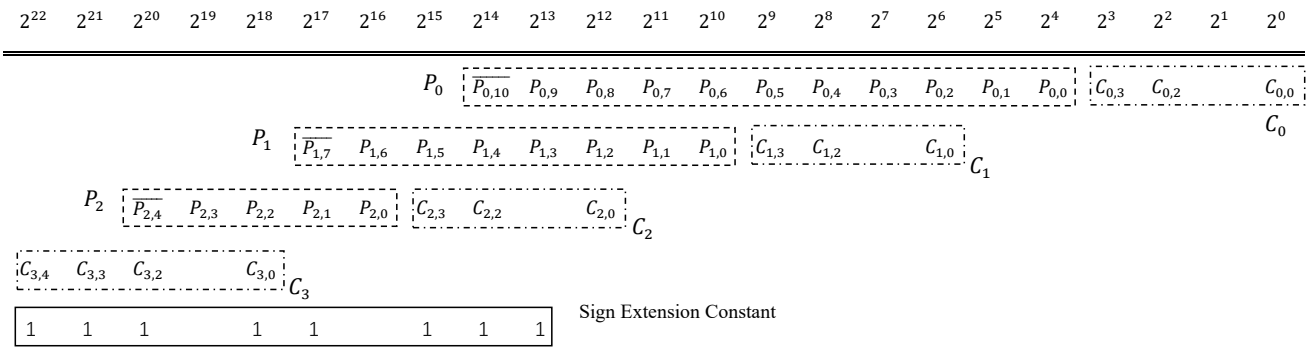

Fig. 2: Partial product matrix of 12-bit squarer using the proposed Radix-8 Booth folding encoding

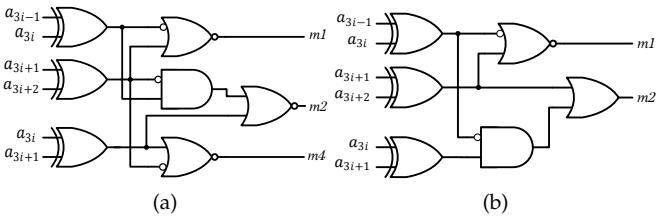

Fig. 3: Encoders of partial product generator $P_{i,j}$ used in an approximate Radix-8 squarer (a) AR8E1. (b) AR8E2.

This approximation introduces both positive and negative errors into the partial product generation circuit. When these two errors occur simultaneously in the circuit, they can compensate each other in the partial product compression process. Therefore, there is no need to generate the m_3 encoder circuit and the corresponding decoder, saving an addition in the partial product generator. Fig. 3(a) shows the encoder circuit of the partial product $P_{i,j}$.

As shown in Fig. 3(a), AR8E1 has no significant area advantage compared to an accurate encoder circuit; however; the serial adder that generates the $x3$ signal in the decoder is not required because the m_3 signal is discarded. Also, the dynamic power consumption of AR8E1 is reduced and area and power of the partial product generator circuit can be saved, with significant improvement in the critical path delay. The complete partial product generation circuit of the Radix-8 Booth folding squarer includes $P_{i,j}$ and $C_{i,j}$. Since $C_{i,j}$ can be generated only by the encoder, and no complex decoding circuit is required. Therefore $C_{i,j}$ is simple to generate and does not require approximation.

3.2 R8AS2 Booth Squarer

To further reduce the complexity of the partial product generator, both m_3 and m_4 are considered. They are approximately equal to m_2 in AR8E2, so reducing the complexity of the approximate partial product generator as equivalent to the Radix-4 Booth folding encoding.

As shown in Fig. 3(b), AR8E2 has one less gate than AR8E1, so achieving a rather small area improvement. However, the discarded output m_4 ensures that AR8E2 further reduces two multiplexed XOR gates, thereby reducing the dynamic power consumption of the approximate encoder.

TABLE 3: Truth Table of the Approximate Compressors

$x_4x_3x_2x_1$	AC_42		AC_32		AC_21	
	S_1S_2	ED	S_1S_2	ED	S	ED
0000	00	0	00	0	0	0
0001	00	-1	00	-1	0	-1
0010	00	-1	00	-1	0	-1
0011	10	-1	10	-1	1	-1
0100	00	-1	00	-1	X	X
0101	01	-1	01	-1	X	X
0110	01	-1	01	-1	X	X
0111	11	-1	11	-1	X	X
1000	00	-1	X	X	X	X
1001	01	-1	X	X	X	X
1010	01	-1	X	X	X	X
1011	11	-1	X	X	X	X
1100	10	-1	X	X	X	X
1101	11	-1	X	X	X	X
1110	11	-1	X	X	X	X
1111	11	-2	X	X	X	X

The complexity of the approximate decoder circuit is also reduced. The approximate input of the encoder of the partial product generator is shown in Table 2, and the error distance (calculated by considering the exhaustive input combinations) is also reported; the value of AR8E2 is always smaller than the exact product, hence the partial product generator causes a negative error. Therefore, if an appropriate approximation method is used to produce a positive error in the partial product compression stage, the errors of these two parts compensate for each other to improve the accuracy.

3.3 R8AS3 Booth Squarer

The designs of R8AS1 and R8AS2 employ two types of approximate partial product generation circuits for the squarer, leading to different power consumption and accuracy. The remaining circuits of the squarer, namely the partial product compression and the final summation, are all accurate. In this section, an approximation method for the partial product compression process is also used. The approximate partial product generator proposed for R8AS2 is used in R8AS3 by reducing the power of the partial product generator. In addition, the approximate 4-2 compressor, the approximate

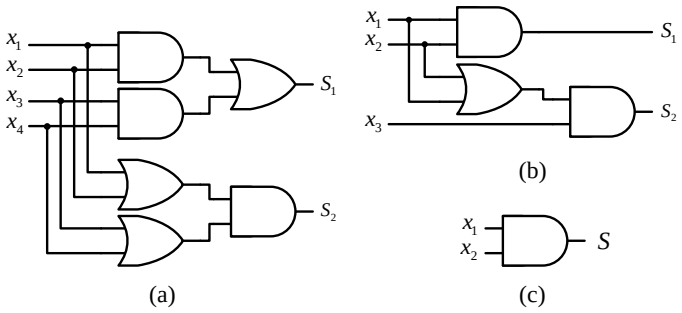

Fig. 4: Approximate compressors with constant compensation. (a) AC_42, (b) AC_32, (c) AC_21.

half and full adder with constant compensation are utilized to replace the accurate adders and decrease the power and area of the squarer.

The truth table of the approximate compressors (AC_42, AC_32 and AC_21) are shown in Table 3. The approximate 4-2 and 3-2 compressors have two sum output signals (S_1 and S_2) in place of the carry output signal. The approximate 2-1 compressor has only one output signal S. As shown in Table 3, these approximate compressors produce errors of -1 in all input states, except when the input is all zeros. When the input is '1111' in AC_42, the error distance is –2 because there are only two output signals with weights at the current bit. Since these approximate compressors have not carry output signals and introduce errors in most input situations, the circuit structure of their output signals becomes simpler. Fig. 4 shows the circuit diagram of the approximate compressor. Although the error probability of the approximate compressor is very high, the error can be recovered by a simple approach which will be demonstrated using the example of 32-bit R8AS3 squarer.

The accumulation diagram of 32-bit R8AS3 partial products is shown in Fig. 5. Approximate partial product generators and approximate compressors are used for lower 32-bit, where the $C_{i,j}$ term is precisely generated.For the column with two partial products, the AC_21 compressor is employed. Meanwhile, the AC_32s are used for processing the compression with three partial products case, and AC_42s are used for more than three partial products case scenarios. Table 3 demonstrates the truth table and the error distance of three types of approximate compressors. It can be observed that the expected error distance for each approximate compressor is almost -1, which means each for n_{th} column, the approximate compressor is expected to introduce a -1×2^n error. Besides, the six LSBs of the first row of partial product is set to 0 in order to maintain a negative error distance. In all, the error introduced by approximate compressors and truncations can be viewed as $\bar{1}\bar{1}...\bar{1}$ (32 bit $\bar{1}$) approximately. Therefore, the $100...0$ (32 bit 0) can be used to compensate for the error significantly, which is adding a constant at the 32_{nd} column of the partial product array. After adding a constant one in the higher bit, the approximate compressors can be thought as producing errors only when the input are all zeros and all ones in the approximate 4-2 compressor. So the error probabilities of approximate compressors (AC_42, AC_32 and AC_21) are 12.5%, 12.5% and 25% respectively.

4 Comparison and Analysis

The proposed approximate squarers are evaluated by considering the required hardware and the error analysis. This work evaluates the performance of 12-bit and 16-bit approximate squarers. In a 12-bit squarer, the approximate Radix-8 encoders and approximate compressors are implemented in the lowest 12 bits. In R8AS1 and R8AS2, the approximate encoders AR8E1 and AR8E2 are used in the least significant 12 bits, and also the approximate bit-width is 16 in 16-bit approximate squarers. The approximate half-adder and full-adder are implemented in 4-7 and 8-11 weight bits respectively; the constant compensation is added in bit 12. No approximate 4-2 compressor is used due to the low height of the partial product array. For a 16-bit approximate squarer, the approximate half-adder and full-adder are implemented in 4-7 and 8-11 weight bits respectively while the approximate 4-2 compressor is used in 12-15 weight bits; the constant compensation is added to bit 16. All approximate squarers use a recursive carry-lookahead adder consisting of 4-bit carry-lookahead adder in the final addition unit.

The squarers used for comparison include truncated squarers [12] and fixed-width squarers based on Radix-4 Booth folding encoding [15]. Also, the approximate Radix-4 Booth squarers (ABSx) of [17] are compared. For a fair comparison, in an n-bit (12 or 16) squarer, the truncation factor is set to n for the truncated schemes [12]. In the fixed width [15] squarers, the lower n bits of the partial product array are considered LP, including the $(n-1)$-bit LP_{minor} and the 1-bit LP_{major}. The approximate factor is set to n with a compensation bit number of k=4 for ABSx [17].

The proposed designs are implemented in Verilog HDL. The Synopsys Design Compiler tool under the NanGate 28-nm standard cell library is used to synthesize the squarers and evaluate power, area and delay with temperature of 25 degrees Celsius and 1V supply voltage. The normalized mean error distance (NMED) are assessed using the C programming language based on uniformly distributed input data. The PDP-NMED product (P-N) is used to evaluate the trade-off between hardware and accuracy of approximate squarers.

All comparison metrics of the proposed and existing approximate squarers are shown in Table 4. For 16-bit squarers, R8AS1 has the highest accuracy among the proposed approximate squarers due to the approximate partial data generator and its compensating features; R8AS2 provides a better trade-off between power consumption and accuracy. Compared with R8AS2, R8AS3 further reduces PDP by 16% due to the use of approximate compressors. The proposed approxiamte squarers save 49.6% to 31.2% PDP and 31.1% to 16.4% area compared with accurate Radix-8 Booth squarer. In addition, the three proposed approximate squarers achieve an 16% to 42% reduction in power consumption and up to 51% reduction in PDP compared with ABSx [17] and have the same or higher accuracy. Fig. 6 plots the PDP and NMED of all approximate designs. The proposed designs provide a comparable accuracy, and reductions in power. TRUN-SQ [12] provides power and delay advantages because they are truncated squarers; however, the truncation operation also causes larger errors. In addition, the dynamic error-compensation circuit and the

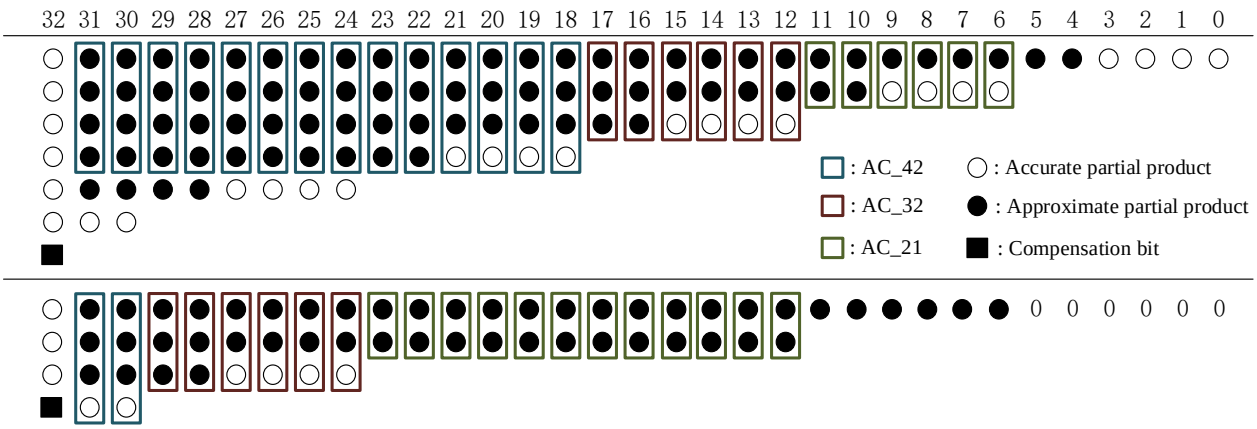

Fig. 5: Partial products accumulation diagram of 32-bit approximate squarer R8AS3

TABLE 4: Hardware Comparison and Error Analysis of Approximate Squarers

Designs		Area (μm^2)	Delay (ns)	Power (μW)	PDP (pJ)	NMED (10^{-4})	P-N (10^{-2})
12-bit	Accurate	337.1	0.97	207.9	201.7	—	—
	R8AS1	**276.6**	**0.85**	**165.4**	**140.6**	**2.024**	**2.85**
	R8AS2	**251.3**	**0.85**	**151.9**	**129.1**	**2.271**	**2.93**
	R8AS3	**234.7**	**0.81**	**139.7**	**113.2**	**3.916**	**4.43**
	ABS1 [17]	312.8	0.82	174.1	142.8	3.712	5.30
	ABS2 [17]	304.0	0.73	150.3	109.7	5.136	5.63
	ABS3 [17]	333.9	0.75	186.9	140.2	3.843	5.39
	TRUN-SQ [12]	224.6	0.61	118.7	72.4	64.431	46.65
	FW-PBFT [15]	361.2	0.87	180.9	157.4	81.269	127.92
16-bit	Accurate	440.7	1.20	267.8	321.4	—	—
	R8AS1	**368.8**	**1.05**	**210.5**	**221.0**	**0.104**	**23.1**
	R8AS2	**336.5**	**1.05**	**183.6**	**192.8**	**0.113**	**21.7**
	R8AS3	**303.7**	**1.02**	**158.7**	**161.9**	**0.235**	**38.0**
	ABS1 [17]	458.8	1.02	267.8	273.2	0.224	61.1
	ABS2 [17]	445.9	0.89	250.5	222.9	0.323	71.9
	ABS3 [17]	491.0	0.93	274.9	255.7	0.231	59.1
	TRUN-SQ [12]	374.4	0.75	197.9	148.4	4.576	679.1
	FW-PBFT [15]	519.8	1.18	258.4	304.9	6.742	2055.6

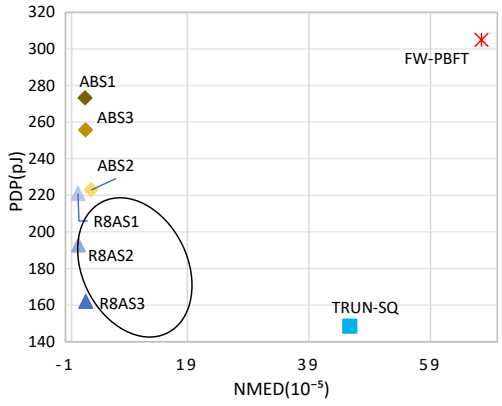

Fig. 6: PDP and NMED plot of the proposed approximate Radix-8 squarers and squarers found in the technical literature.

folding method rather than Radix-4 Booth folding algorithm are used in FW-PBFT [15], this results in the highest area and delay penalties. Among all approximation designs, the proposed three squarers show the largest advantages in PDP and NMED when combined. For 12-bit squarers, the proposed approximate squarers provide lower errors at the same PDP level compared with ABSx [17], because the circuit of the Radix-8 Booth folding algorithm occupies more area at a low bit-width. The lower bit-width decreases the advantage of this algorithm by reducing the number of partial products. Nevertheless, the proposed approximate squarers achieve an advantage in accuracy due to the lower error probability of the approximate encoder and the approximate compressors with compensation.

The proposed squares are further evaluated for hardware resource consumption and error under 32-bit. As is shown in Table. 5, R8AS1 and R8AS2 only use approximate encoders, so their performance is similar. Furthermore, R8AS3 uses the approximate compressors that save the most

TABLE 5: Hardware and Error Metrics of 32-bit Squarers

Designs	Area (μm^2)	Delay (ns)	Power (mW)	NMED (10^{-7})
Accurate	1813.0	4.77	0.650	—
R8AS1	1670.2	4.61	0.575	2.231
R8AS2	1605.8	4.58	0.558	2.857
R8AS3	1458.4	4.51	0.512	4.441

area, so R8AS3 has the lowest power consumption and area, but the accuracy is also reduced. Compared with exact squarer, they can save lots of hardware resources and have lower critical path delay.

5 Applications

5.1 Square-Law Detector

In communication application, detection techniques are required to extract low-frequency signals. For small signals in amplitude modulation (AM), the so-called square-law detector is used to demodulate the signal. In the demodulation process, the modulated amplitude signal is squared to obtain a signal of twice the frequency. The high-frequency component is removed by a low-pass filter, and the original signal is obtained by coherent demodulation. For a square-law detector of the demodulation process, the modulated signal needs to be squared as:

$$x^2(t) = [A^2(m(t)+1)^2 + A^2(m(t)+1)^2 \cdot \cos(2\pi f_c t)]/2 \quad (15)$$

where f_c and A stands for the frequency and amplitude of carrier signal, while $m(t)$ is the message signal. The high-frequency component of the squared signal can be filtered after passing through a low-pass filter. Then, the output is connected to the square root module and the DC component is subtracted to demodulate the message signal.

Simulink simulation is used to model the square-law detector. Fig. 7 shows the demodulated waveform after applying the proposed squarers to the square-law detector. To assess the performance of the proposed approximate squarers for demodulating signals, carrier signals and message signals at various frequencies and amplitudes are used to generate the modulated signals. In Fig. 6(a), the carrier signal with frequency f_c = 1kHz and amplitude A = 1V and the message signal with frequency f_m = 50Hz and amplitude M = 0.5V are used. The carrier signal with f_c = 1.5kHz and A = 1V and the message signal with f_m = 200Hz and M = 0.25V are also used in the simulation of Fig. 6(b). In addition, the sample frequency of all experiments is set to 20kHz, and the passband edge frequency of the low-pass filter satisfies the Nyquist sampling theory. In these cases R8AS1 has the best performance due to its lowest NMED. Although the three approximate squarers slightly shift the waveform up, all signals are demodulated smoothly. The SNR is used to evaluate the quality of the demodulated waveform. Fig. 8 illustrates the SNR of the demodulated waveforms of each approximate squarer. The proposed squarers have an SNR close to 30dB, and this result is consistent with the error analysis of Table 4.

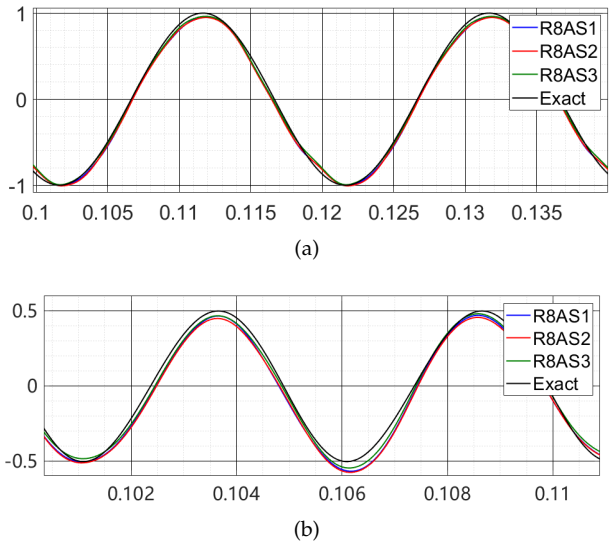

Fig. 7: AM demodulated signals using the proposed approximate squarers and an exact squarer: (a) the carrier signal with f_c = 1kHz, A = 1V and message signal with f_m = 50Hz, M = 0.5V, and (b) the carrier signal with f_c = 1.5kHz, A = 1V and message signal with f_m = 200Hz, M = 0.25V.

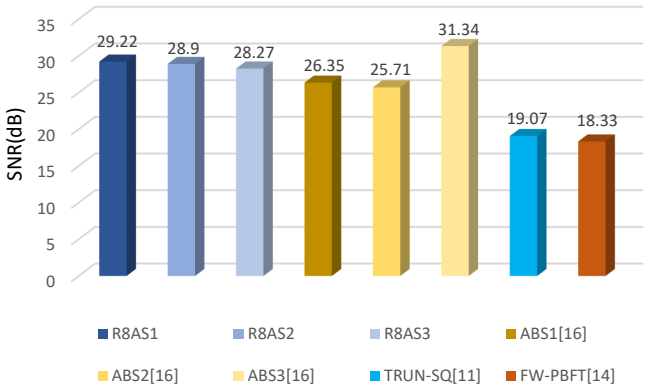

Fig. 8: SNR of demodulated waveforms for squarers.

5.2 K-means Clustering Results

The K-means clustering is widely utilized in the clustering paradigm as based on minimizing a formal objective function [20]. As a typical application of the unsupervised clustering algorithm, the k-means clustering algorithm is used to automatically classify samples into a category. In the clustering algorithm, the classification result is obtained according to the coordinates for the center point of the cluster and the Euclidean distance from the center point to the data point to be classified in each cluster. In this section, the proposed three approximate squarers are used to calculate the Euclidean distance and compared with the classification results using an exact squarer. The data classification results are evaluated by using the F_1-measure parameter for testing the performance of the classifier. The value of the F_1-measure is given by:

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} \quad (16)$$

where P is the precision rate and R is the recall rate.

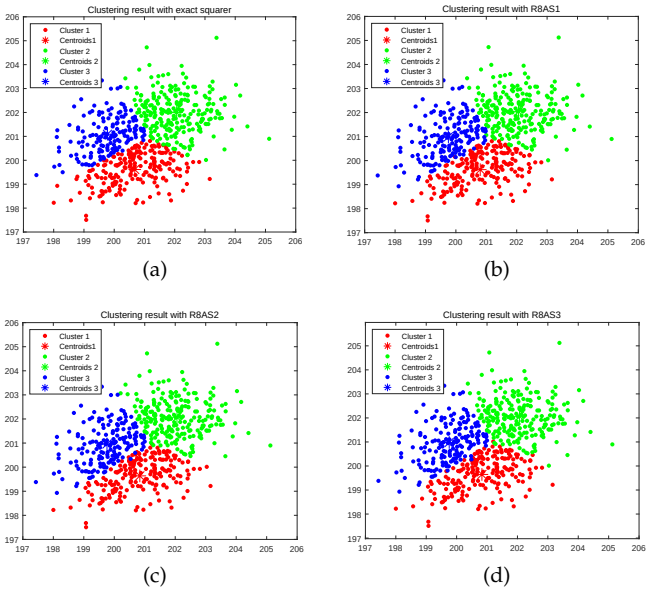

Fig. 9: Clustering results with different squarers: (a) exact squarer with F_1=1, (b) R8AS1 with F_1=0.9366, (c) R8AS2 with F_1=0.9236 and (d) R8AS3 with F_1=0.9106.

Fig. 9 shows the classification results obtained by using the exact squarer and the three proposed approximate squarers for the Euclidean distance. As experimental results, 600 randomly generated data points and three clustering centers with different colors are utilized. The use of the proposed approximate squarers in calculating the minimum Euclidean distance has no significant impact on the classification results. The F_1-measure values of the three proposed approximate squarers are also shown in Fig. 9. When the value of F_1-measure is equal to 1, then the classification result is fully accurate. R8AS1 has the highest value, again consistent with the error analysis of the approximate squarers in Section 4.

6 Conclusion

In this paper, a Radix-8 Booth folding method has been proposed to reduce the number of partial products and further simplify the design of the compression tree. Then two approximate partial product generators (AR8E1 and AR8E2) have been designed to reduce the complexity of the encoder and decoder. Also, the approximate compressors with constant compensation have been designed to reduce the compressor tree area and power, while compensating for the negative error generated by the approximate compressors. Finally, three different combinations of approximate squarers (R8AS1, R8AS2, and R8AS3) have been proposed to assess the trade-off between hardware and accuracy of the different design schemes. The proposed designs provide up to 51% and 68% reduction in PDP and NMED compared with the latest approximate squarers. Moreover, the effectiveness of the approximate squarers have been verified by applying them to the square-law detector and the k-means clustering algorithm.

Acknowledgments

This work is supported by grants from the National Natural Science Foundation of China (62101252, 62022041 and 61871216).

References

[1] F. Yu and A. Willson, "Multirate digital squarer architectures," in *Proc. 8th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.01EX483)*, vol. 1, pp. 177-180, 2001.

[2] R. Kolagotla, W. Griesbach, d. H. Srinivas, "VLSI implementation of 350MHz 0.35μm 8-bit merged squarer". *Electronics Letters*, vol. 34, no. 1, pp. 47-48, 1998.

[3] H Jiang, F Santiago, H Mo, et al, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp, 2108-2135, 2020.

[4] L. Dadda, "Squarers for binary numbers in serial form," in *Proc. 7th IEEE Symposium on Computer Arithmetic (ARITH)*, pp. 173-179, 1985.

[5] J. Pihl and E. J. Aas, "A multiplier and squarer generator for high performance DSP applications," in *Proc. 39th Midwest Symposium on Circuits and Systems*, IEEE, vol. 1, pp. 109-112, 1996.

[6] A. Deshpande and J. Draper, "Squaring units and a comparison with multipliers," in *Proc. 53rd IEEE International Midwest Symposium on Circuits and Systems*. IEEE, pp. 1266–1269, 2010.

[7] B. Phillips, "Optimised squaring of long integers using precomputed partial products," in *Proc. 15th IEEE Symposium on Computer Arithmetic (ARITH)*, pp. 73-79, 2001.

[8] E. Chaniotakis, P. Kalivas and K. Z. Pekmestzi, "Long number bit-serial squarers," in *Proc. 17th IEEE Symposium on Computer Arithmetic (ARITH)*, pp. 29-36, 2005.

[9] D. W. Matula, "Higher Radix Squaring Operations Employing Left-to-Right Dual Recoding," in *Proc. 19th IEEE Symposium on Computer Arithmetic (ARITH)*, pp. 39-47, 2009.

[10] A. G. Strollo and D. De Caro, "Booth folding encoding for high performance squarer circuits," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, no. 5, pp. 250–254, 2003.

[11] 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 Transactions on Computers*, vol. 66, no. 8, pp. 1435–1441, 2017.

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

[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 International Conference on Application-specific Systems, Architectures and Processors*. IEEE, pp. 91–97, 2009.

[14] A. Avramović, Z. Babić, and D. Raič, D. Strle and P. Bulić, "An approximate logarithmic squaring circuit with error compensation for DSP applications," *Microelectronics Journal*, vol. 45, no. 3, pp. 263–271, 2014.

[15] Y.-H. Chen, "Area-efficient fixed-width squarer with dynamic error-compensation circuit," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 9, pp. 851–855, 2015.

[16] K.-J. Cho and J.-G. Chung, "Low error fixed-width two's complement squarer design using Booth-folding technique," *IET Computers & Digital Techniques*, vol. 1, no. 4, pp. 414–422, 2007.

[17] K. M. Reddy, M. V asantha, Y. N. Kumar, and D. Dwivedi, "Design of approximate Booth squarer for error-tolerant computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 5, pp. 1230–1241, 2020.

[18] H. Waris, C. Wang, and W. Liu, "Hybrid low Radix encoding-based approximate Booth multipliers," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3367–3371, 2020.

[19] J. Fadavi-Ardekani, "M*n Booth encoded multiplier generator using optimized wallace trees," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 2, pp. 120–125, 1993.

[20] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881-892, July 2002.