Low-Power Approximate Multipliers Using Encoded Partial Products and Approximate Compressors

Mohammad Saeed Ansari, Honglan Jiang, Bruce F. Cockburn, Jie Han

Abstract—**Approximate computing has been considered to improve the accuracy-performance trade-off in error-tolerant applications. For many of these applications, multiplication is a key arithmetic operation. Given that approximate compressors are a key element in the design of power-efficient approximate multipliers, we first propose an initial approximate 4:2 compressor that introduces a rather large error to the output. However, the number of faulty rows in the compressor's truth table is significantly reduced by encoding its inputs using generate and propagate signals. Based on this improved compressor, two 4×4 multipliers are designed with different accuracies and then are used as building blocks for scaling up to 16×16 and 32×32 multipliers. According to the mean relative error distance (MRED), the most accurate of the proposed 16×16 unsigned designs has a 44% smaller power-delay product (PDP) compared to other designs with comparable accuracy. The radix-4 signed Booth multiplier constructed using the proposed compressor achieves a 52% reduction in the PDP-MRED product compared to other approximate Booth multipliers with comparable accuracy. The proposed multipliers outperform other approximate designs in image sharpening and joint photographic experts group (JPEG) applications by achieving higher quality outputs with lower power consumptions. For the first time, we show the applicability and practicality of approximate multipliers in multiple-input multiple-output (MIMO) antenna communication systems with error control coding.**

Index Terms—**approximate computing, multiplier, MIMO, image sharpening, JPEG.**

I. Introduction

THE continuing shrinkage in the minimum feature size has made integrated circuit behavior increasingly vulnerable to process, voltage and temperature (PVT) variations as well as soft errors [1], [2]. Thus the challenge of ensuring strictly accurate computing is increasing [3]. On the other hand, there exist many applications, such as multimedia and machine learning, which do not necessarily need fully accurate results. Such applications are tolerant of small inaccuracies [4]-[6] and so approximate computing can be applicable due to its potentially significant reduction in design costs while still producing sufficiently accurate results [7], [8].

Multiplication is a key arithmetic operation that is optimized in digital processors. Many approximate multiplier designs have been proposed in the literature [9]-[18]. In [9], the authors proposed two approximate error accumulation techniques to perform partial product accumulation, resulting in approximate multipliers AM1 and AM2. The truncation of the least significant bits (LSBs) of the partial products is considered in [5], resulting in designs referred to as TAM1 and TAM2. Approximate compressors AC1 and AC2 are proposed in [11] to reduce the delay and power consumption. While exact compressors are used for the 8 most significant bits (MSBs), the two proposed designs are employed for the 8 LSBs in a 16×16 multiplier, referred to as ACM3 and ACM4, respectively.

Approximation in the partial product tree is addressed in the broken-array multiplier (BAM) [12], approximate Wallace tree multiplier (AWTM) [13], and the error-tolerant multiplier (ETM) [14]. The so-called under-designed multiplier (UDM) uses 2×2 approximate multiplier blocks to construct larger multiplier blocks [15]. The approximate multiplier proposed in [16] approximates the binary logarithm of the multiplicand and multiplier and then adds them and generates the final approximate product using exponentiation.

In this article, we suggest an initial approximation for a 4:2 compressor in which several rows in the compressor's truth table are faulty. However, the inputs to the compressors, i.e., the partial products of the multiplication, can be encoded using generate and propagate signals so that the error rate of the compressor is reduced significantly. Using the proposed compressor, we design two 4×4 multipliers in which approximation is employed in the partial product reduction tree, which is the most expensive part of the design of a multiplier [18]. The two proposed designs are then used to construct 16×16 and 32×32 multipliers that are synthesized by the Synopsys Design Compiler for ST's 28-nm CMOS process.

The remainder of this article is organized as follows: Section II provides the required background. Section III presents the proposed multiplier designs and discusses the hardware implementation in detail. Section IV reports the hardware and error performance metrics of the proposed and other approximate multipliers. Image sharpening, joint photographic experts group (JPEG) compression, and multiple-input multiple-output (MIMO) interference nulling applications are considered in Section V to provide a practical evaluation of the

This work was financially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) (Project No. RES0018685).

M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, T6G 1H9, Canada (e-mail: {ansari2, honglan, cockburn, jhan8}@ ualberta.ca).

proposed designs. Finally, Section VI concludes the article.

II. Background

A. Partial product accumulation in 4×4 multipliers

Consider two 4-bit unsigned operands $\alpha = \sum_{i=0}^{3} \alpha_i 2^i$ and $\beta = \sum_{j=0}^{3} \beta_j 2^j$. The partial product array $\boldsymbol{pp}$ is a 4×4-bit array of the partial product bits $pp_{i,j} = \alpha_i . \beta_j$, where $i, j \in \{0, 1, 2, 3\}$. Table 1 gives all the partial products for a 4-bit multiplication and their corresponding product bits.

The product is denoted by $\gamma = \sum_{k=0}^{7} \gamma_k 2^k$. The bits of γ are produced in stages going from the LSB to the MSB. According to Table 1, $\gamma_0 = pp_{0,0}$ and there is no further operation in Stage 0. In Stage 1, to generate γ_1, we can simply use a half adder that produces a sum bit γ_1 and a carry bit (c_1) for the next stage. Since the half adder circuit is already a simple design, there is no need to approximate it.

Table 1. Original partial product of the multiplication.

Stage 7	**Stage 6**	**Stage 5**	**Stage 4**	**Stage 3**	**Stage 2**	**Stage 1**	**Stage 0**
	$pp_{3,3}$	$pp_{3,2}$	$pp_{3,1}$	$pp_{3,0}$	$pp_{2,0}$	$pp_{1,0}$	$pp_{0,0}$
		$pp_{2,3}$	$pp_{2,2}$	$pp_{2,1}$	$pp_{1,1}$	$pp_{0,1}$	
			$pp_{1,3}$	$pp_{1,2}$	$pp_{0,2}$		
				$pp_{0,3}$			
$\boldsymbol{\gamma_7}$	γ_6	γ_5	γ_4	γ_3	γ_2	γ_1	γ_0

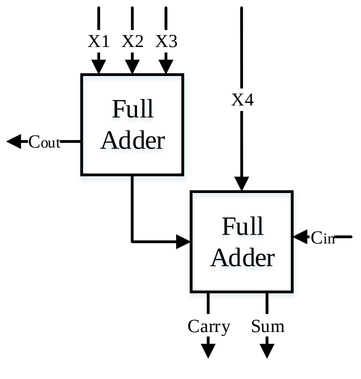
In Stage 2, there are three pp terms and the carry from the previous stage (c_1) that must be added together. Thus, a 4:2 compressor is required to generate γ_2 and a carry for the next stage.

B. Exact $4:2$ compressor

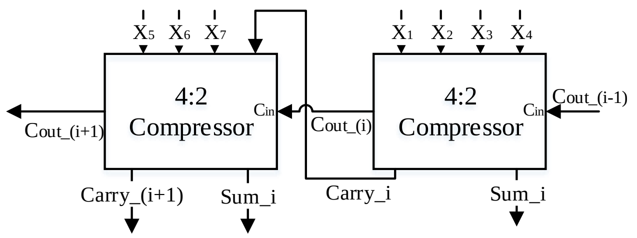
The function of the exact 4:2 compressor is implemented by using two appropriately connected full adders (see Fig. 1(a)) as given by

$$\begin{aligned} Sum &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus C_{in}, \\ C_{out} &= (x_1 \oplus x_2).x_3 + (\overline{x_1 \oplus x_2}).x_1, \\ Carry &= (x_1 \oplus x_2 \oplus x_3 \oplus x_4).C_{in} + (\overline{x_1 \oplus x_2 \oplus x_3 \oplus x_4}).x_4. \end{aligned} \tag{1}$$

The Sum output has the same weight as the four input signals while the C_{out} is used as the carry in for the next higher-order compressor and the output $Carry$ is weighted like a pp bit in a one-bit-higher position. Note that C_{out} and $Carry$ have the same weight. The two stages of an exact 4:2 compressor chain are shown in Fig. 1(b) [17].



(a) Compressor design.



(b) Compressor chain.

Fig. 1. Exact compressor.

III. Proposed Multiplier Design

A. Modified approximate 4:2 compressor

The function of an exact 4:2 compressor can be approximated to reduce the hardware cost. It has been shown that C_{out} does not have a significant impact on the compressor's accuracy [11], so C_{out} is ignored in our design. Moreover, our SPICE simulations confirm that an XOR gate consumes more power and is slower than the AND and OR gates, as shown in Table 2.

Table 2. Normalized relative comparison of AND, OR and XOR gates.

Gate	**Delay**	**Power consumption**
AND	0.58	0.42
OR	0.79	0.40
XOR	1.00	1.00

Ignoring C_{out} and not using XOR gates as well as our goal to use as few gates as possible led to the approximate compressor truth table given in Table 3. As shown in Table 3, there are five/seven incorrect values for the approximate $Carry/Sum$ outputs which can contribute to error in the function output.

Table 3. Truth table of the proposed approximate compressor.

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{x_3}$	$\boldsymbol{x_4}$	**Carry**	**Sum**
0	0	0	0	0/0 ✓	0/0 ✓
0	0	0	1	0/0 ✓	1/1 ✓
0	0	1	0	0/0 ✓	1/1 ✓
0	0	1	1	1/1 ✓	0/1 ×
0	1	0	0	0/0 ✓	1/1 ✓
0	1	0	1	1/0 ×	0/1 ×
0	1	1	0	1/0 ×	0/1 ×
0	1	1	1	1/1 ✓	1/1 ✓
1	0	0	0	0/0 ✓	1/1 ✓
1	0	0	1	1/0 ×	0/1 ×
1	0	1	0	1/0 ×	0/1 ×
1	0	1	1	1/1 ✓	1/1 ✓
1	1	0	0	1/1 ✓	0/1 ×
1	1	0	1	1/1 ✓	1/1 ✓
1	1	1	0	1/1 ✓	1/1 ✓
1	1	1	1	0/1 ×	0/1 ×

$$\boldsymbol{approximate\ sum} = (\boldsymbol{x_1} + \boldsymbol{x_2}) + (\boldsymbol{x_3} + \boldsymbol{x_4})$$
$$\boldsymbol{approximate\ carry} = (\boldsymbol{x_1}.\boldsymbol{x_2}) + (\boldsymbol{x_3}.\boldsymbol{x_4})$$

To manage this source of inaccuracy, we encode the inputs to the compressor using $propagate$ and $generate$ signals given by

$$\begin{aligned} P_{i,j} &= pp_{i,j} + pp_{j,i}, \\ G_{i,j} &= pp_{i,j}.pp_{j,i}. \end{aligned} \tag{2}$$

This encoding ensures that although the circuit may have a fairly large number of faulty outputs, it in fact rarely produces those outputs [18]. To see how this approach affects the compressor's accuracy, consider Stage 2 in which the following terms are added: $pp_{2,0}$, $pp_{1,1}$, $pp_{0,2}$ and c_1. Table 4, where NA stands for $Not\ Applicable$, shows how encoding the partial products using (2) helps to improve the design accuracy compared to the situation in Table 3.

Note that all possible input combinations for the 4×4 multiplier were considered ($2^4 \times 2^4$=256) to obtain the probability of each input combination in Table 4. Using the proposed technique, the number of faulty $Carry/Sum$ values reduces from 5/7 to 2/4. Note that the two approximated cases for the $carry$ signal occur only with a small probability of 0.078 (0.0624+0.0156), see Table 4.

It is also worth mentioning that the following combinations in Table 4 cannot occur, so they do not contribute to the output errors for the approximate compressor:

1. "0, 1" for $(pp_{1,1},\ c_1)$: since $c_1 = pp_{0,1}.pp_{1,0} = (\alpha_0.\beta_1).(\alpha_1.\beta_0)$, $c_1 =$ "1" means that α_0, β_1, α_1, and β_0 are "1". Consequently, $pp_{1,1} = \alpha_1.\beta_1 =$ "1". Hence, it is impossible to have the "0, 1" combination for $(pp_{1,1},\ c_1)$.
2. "0, 1, 1" for $(c_1, pp_{1,1}, G_{2,0})$: having $c_1 = pp_{0,1}.pp_{1,0} = (\alpha_0.\beta_1).(\alpha_1.\beta_0) =$ "0" and $pp_{1,1} = \alpha_1.\beta_1 =$ "1" means at least one of a_0 or b_0 is "0", which leads to $G_{2,0} = pp_{2,0}.pp_{0,2} = (\alpha_2.\beta_0).(\alpha_0.\beta_2) =$ "0". Thus, the "0, 1, 1" combination for $(c_1, pp_{1,1}, G_{2,0})$ is not applicable.
3. "0, 1" for $(P_{2,0}, G_{2,0})$: $G_{2,0} = pp_{2,0}.pp_{0,2} =$ "1" means that both $pp_{2,0}$ and $pp_{0,2}$ are "1". Therefore, $P_{2,0} = pp_{2,0}+pp_{0,2} =$ "1" and so we cannot have the "0, 1" combination for $(P_{2,0}, G_{2,0})$.

Table 4. Truth table for the Stage 2 compressor.

$P_{2,0}$	$G_{2,0}$	$pp_{1,1}$	c_1	Carry	Sum	Probability
0	0	0	0	✓	✓	0.4218
0	0	0	1	NA	NA	0.0000
0	0	1	0	✓	✓	0.1251
0	0	1	1	✓	×	0.0156
0	1	0	0	NA	NA	0.0000
0	1	0	1	NA	NA	0.0000
0	1	1	0	NA	NA	0.0000
0	1	1	1	NA	NA	0.0000
1	0	0	0	✓	✓	0.2814
1	0	0	1	NA	NA	0.0000
1	0	1	0	×	×	0.0624
1	0	1	1	✓	✓	0.0312
1	1	0	0	✓	×	0.0468
1	1	0	1	NA	NA	0.0000
1	1	1	0	NA	NA	0.0000
1	1	1	1	×	×	0.0156

According to (2) and the laws of Boolean algebra, when $P_{2,0}$ and $G_{2,0}$ are used as the x_1 and x_2 inputs to the compressor, the sum and $carry$ signals in the compressor for Stage 2 can be simplified as

$$\begin{gathered} sum = x_1 + x_3, \\ carry = x_2 + x_4. \end{gathered} \tag{3}$$

To compute γ_3 in Stage 3, the four $pp_{i,j}$ terms and the carry c_2 from Stage 2 should be added and therefore a 5:2 compressor is required. Since the proposed compressor is a 4:2 design, we can merge two of these five signals to reduce them to four, as specified by

$$x_1 = c_2, x_2 = G_{3,0} + G_{2,1}, x_3 = P_{2,1}, x_4 = P_{3,0}. \tag{4}$$

where x_1, x_2, x_3 and x_4 are the inputs to the compressor that generates γ_3 and a carry out (c_3) for the next stage. Table 5 shows how altering the partial products affects the compressor's truth table in Stage 3. As in Stage 2, the design can be simplified by using Boolean algebra. The resulting simplified compressor design for Stage 3 is then given by

$$\begin{gathered} sum = x_1 + x_3 + x_4, \\ carry = x_1.x_2 + x_3.x_4. \end{gathered} \tag{5}$$

The calculation of γ_4 in Stage 4 is exactly like the calculation of γ_2 in Stage 2. It uses a 4:2 compressor to add $pp_{3,1}$, $pp_{2,2}$, $pp_{1,3}$ and c_3 to generate γ_4 and a carry out for the next stage (c_4). Table 6 shows the effect of the change on the partial products. As shown in Table 6, the number of faulty cases has been reduced and those that remain are less likely to happen.

Note that the only output that differs in Table 6 and Table 4 is the carry signal. In fact, the carry signals in these two stages are generated from different terms. Using the same argument as in Table 4, when $G_{3,1} = 1$, $P_{3,1}$ must be "1". Hence, the entries that do not follow this are NA entries. Also, note that according to the laws of Boolean algebra, the $carry$ and sum signals in the compressor for Stage 4 can be simplified as:

$$\begin{gathered} sum = x_1 + x_2 + x_3, \\ carry = x_2 + x_3.x_4. \end{gathered} \tag{6}$$

Table 5. Truth table for the Stage 3 compressor.

c_2	$G_{3,0} + G_{2,1}$	$P_{2,1}$	$P_{3,0}$	Carry	Sum	Probability
0	0	0	0	✓	✓	0.3087
0	0	0	1	✓	✓	0.1953
0	0	1	0	✓	✓	0.1952
0	0	1	1	✓	×	0.1092
0	1	0	0	NA	NA	0.0000
0	1	0	1	×	×	0.0273
0	1	1	0	×	×	0.0315
0	1	1	1	✓	✓	0.0233
1	0	0	0	✓	✓	0.0079
1	0	0	1	×	×	0.0156
1	0	1	0	×	×	0.0158
1	0	1	1	✓	✓	0.0314
1	1	0	0	NA	NA	0.0000
1	1	0	1	✓	✓	0.0079
1	1	1	0	✓	✓	0.0038
1	1	1	1	×	×	0.0273

Table 6. Truth table for the Stage 4 compressor.

$P_{3,1}$	$G_{3,1}$	$pp_{2,2}$	c_3	Carry	Sum	Probability
0	0	0	0	✓	✓	0.4057
0	0	0	1	✓	✓	0.0156
0	0	1	0	✓	✓	0.1173
0	0	1	1	✓	×	0.0233
0	1	0	0	NA	NA	0.0000
0	1	0	1	NA	NA	0.0000
0	1	1	0	NA	NA	0.0000
0	1	1	1	NA	NA	0.0000
1	0	0	0	✓	✓	0.2106
1	0	0	1	×	×	0.0700
1	0	1	0	×	×	0.0390
1	0	1	1	✓	✓	0.0547
1	1	0	0	✓	×	0.0193
1	1	0	1	✓	✓	0.0272
1	1	1	0	✓	✓	0.0039
1	1	1	1	×	×	0.0116

B. Two approximate 4×4 *multipliers*

The two proposed 4×4 approximate multipliers are referred to as: (1) M1, that considers the carry from the previous stage (c_4) and uses an exact full-adder to add pp terms $pp_{3,2}$, $pp_{2,3}$, and c_4; and (2) M2, that ignores c_4 and uses an exact half adder to add $pp_{3,2}$ and $pp_{2,3}$. They differ in how the product bit γ_5 is produced. Hence, M1 and the larger multipliers that are constructed using it are more accurate than M2 and its scaled-up variants. However, since c_4 is generated from the four LSBs, it does not introduce a large error in an 8×8 multiplier. Note that ignoring c_4 breaks the longest path (that is, the carry propagation) and it is a common technique to reduce the circuit's latency [14], [19].

The sixth Sum output of the full adder in design M1 and the half adder in design M2 are both denoted by γ_5 and the corresponding carry signal, c_5, goes to the next stage to be added to $pp_{3,3}$ using an exact half adder. The Sum and $Carry$ outputs of this final half adder produce γ_6 and γ_7, respectively, see Fig. 2. Fig. 2 summarizes the two mentioned designs by showing the employed blocks for reducing the partial products. These blocks include: (1) half adders, (2) full adders, and (3) 4:2 compressors. The structure of compressors for Stage 2, 3 and 4 are specified in Tables 4, 5, and 6, respectively.

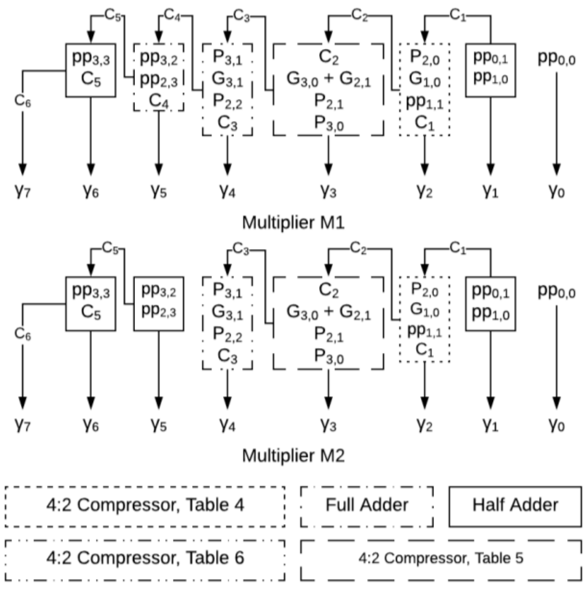


Fig. 2. Partial product reduction in multipliers M1 and M2.

C. Scaling up to larger multipliers

In order to construct larger, e.g. 16×16 and 32×32, approximate multipliers, the two proposed 4×4 multipliers are combined in an array structure. For instance, to construct an 8×8 multiplier using a 4× 4 design, the two 8-bit operands A and B are partitioned into two 4-bit nibbles, namely α_H and α_L for A and β_H and β_L for B. Note that α_H and β_H are the 4 MSBs and α_L and β_L indicate the 4 LSBs of A and B, respectively. Each two of these four nibbles (in total 4 possible combinations) are multiplied using 4×4 multipliers and the partial products are then shifted (based on the nibble's importance) and added together (using a Wallace tree architecture) to produce the final multiplication result. Building $2n$×$2n$ multipliers using n×n multipliers is specified in Fig. 3 and is described by:

$$\begin{aligned}\gamma = \alpha \times \beta = (2^n \times \alpha_H + \alpha_L) \times (2^n \times \beta_H + \beta_L) \\ = 2^{2n} \times (\alpha_H \times \beta_H) + 2^n \\ \times \left((\alpha_H \times \beta_L) + (\alpha_L \times \beta_H)\right) + (\alpha_L \times \beta_L) \\ = 2^{2n} \times P_1 + 2^n \times (P_2 + P_3) + P_4.\end{aligned} \tag{7}$$

Note that each partial product P_i where $i \in \{1, 2, 3, 4\}$ in (7) is generated using an n×n multiplier and multiplications by 2^{2n} and 2^n are simply done by $2n$-bit and n-bit left shifts, respectively. Given that P_4 is the least and P_1 is the most significant partial products, whereas P_2 and P_3 are equivalently significant, multipliers with different accuracies can be designed with different configurations. We propose six 8×8 approximate multipliers, three of which, i.e. M8-1, M8-3, and M8-5, use M1 and the other three use M2 as their main building block. Table 7 shows how each of these six 8×8 multipliers is constructed.

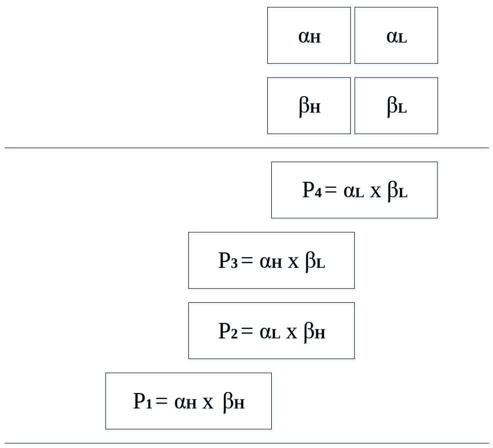


Fig. 3. Building *2n*×*2n* multipliers using *n*×*n* multipliers.

Table 7. Using M1 and M2 to construct 8×8, 16×16, and 32×32 designs.

Size	Design	P1	P2	P3	P4
8×8	M8-1	M1	M1	M1	M1
	M8-2	M2	M2	M2	M2
	M8-3	Exact	M1	M1	M1
	M8-4	Exact	M2	M2	M2
	M8-5	Exact	Exact	Exact	M1
	M8-6	Exact	Exact	Exact	M2
16×16	M16-1	M8-1	M8-1	M8-1	M8-1
	M16-2	M8-2	M8-2	M8-2	M8-2
	M16-3	M8-3	M8-3	M8-3	M8-3
	M16-4	M8-4	M8-4	M8-4	M8-4
	M16-5	M8-5	M8-5	M8-5	M8-5
	M16-6	M8-6	M8-6	M8-6	M8-6
32 × 32	M32-5	M16-5	M16-5	M16-5	M16-5
	M32-6	M16-6	M16-6	M16-6	M16-6

According to Table 7, M8-1 and M8-2 use 4×4 approximate multipliers M1 and M2, respectively, to generate all four partial products from P_1 to P_4. M8-3 and M8-4 are more accurate designs in which the most significant partial product, P_1, is generated using an exact 4×4 multiplier and M1 and M2 are respectively used to generate P_2, P_3, and P_4. M8-5 and M8-6 are the most accurate designs in which only the least significant partial product, P_4, uses approximate multipliers M1 and M2, respectively, and the other three partial products are generated using exact multipliers.

Note that 16×16 and 32×32 approximate multipliers can be constructed by considering (6). We scaled up the six 8×8 designs in Table 7 to form six 16×16 and 32×32 multipliers.

Using the six 8×8 multipliers in Table 7 to construct 16×16 ones, as specified in Fig. 3 and (6), we obtain 6^4 possible 16×16 multiplier designs. Since this is a large number, we only consider six designs using the simple scheme shown in Table 7. These designs are (1) the most accurate scaled-up variants using M1 and M2, referred to as M16-5 and M16-6, respectively; (2) the most hardware efficient scaled-up variants using M1 and M2, referred to as M16-1 and M16-2, respectively; and (3) two designs (one using M1, i.e.M16-3 and the other one using M2, i.e. M16-4) that have a good trade-off between accuracy and hardware. Only one type of 8×8 multipliers is used to construct the 16×16 designs. The most accurate variants of the 16×16 multipliers, i.e. M16-5 and M16-6, are selected to construct 32×32 multipliers M32-5 and M32-6, respectively, as described in Table 7.

The same design approach can be applied to any n×n multiplier where n is a power of 2. Since we have six 8×8 multipliers and four n×n multipliers are required to build a 2n×2n multiplier, the number of possible designs is given by:

$$(6^4)^{\log_2\left(\frac{n}{8}\right)} = (6^4)^{\log_2(n)-3} = 6^{\left(\frac{n^2}{64}\right)}. \quad (8)$$

According to (8), the number of possible designs exponentially increases with n^2. These designs have a wide range of accuracy-hardware trade-offs and could be utilized in different applications, based on application requirements.

D. Extension to signed Booth multipliers

The proposed approximate compressor can also be utilized in signed Booth multipliers. In a Booth multiplier, the partial products are generated using a Booth encoder, and the major difference between the unsigned and signed Booth multiplication is in the generation of the partial products. Therefore, the partial products in Booth multipliers can be accumulated using approximate compressors, but not the sign extension bits [18]. Table 8 shows the radix-4 encoding algorithm in which the multiplier X is divided into overlapping groups of three bits (X_{i-1}, X_i, and X_{i+1}), starting from the LSB. Z is the encoded value to be multiplied with the multiplicand Y.

Following [20], an 8×8 Booth multiplier was designed and implemented using the proposed approximate compressors for the 8 LSBs while the 8 MSBs use exact compressors, see Fig. 4. Note that the sign extension of the partial product array is usually simplified by using the Baugh-Wooley algorithm as shown in Fig. 4 [20]. The hot one (HO) in Fig. 4 indicates the negative encoded values, i.e. HO = 1 for the -2Y and -1Y entries in Table 8 and HO = 0 for the +2Y and +1Y entries. $\bar{\Delta}$ denotes a negated sign bit.

Table 8. Radix-4 Booth encoding.

X_{i+1}	X_i	X_{i-1}	Z	Operation on Y
0	0	0	0	0Y
0	0	1	+1	+1Y
0	1	0	+1	+1Y
0	1	1	+2	+2Y
1	0	0	-2	-2Y
1	0	1	-1	-1Y
1	1	0	-1	-1Y
1	1	1	0	0Y

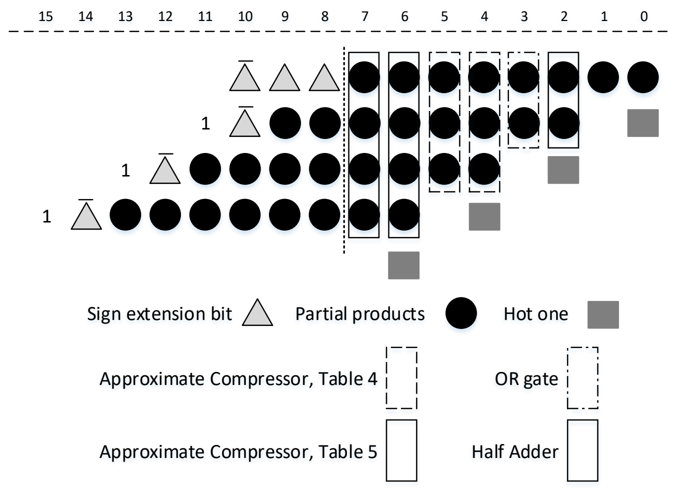


Fig. 4. Using approximate compressors in Booth multipliers.

Approximate encoders for constructing approximate radix-4 Booth multipliers are proposed in [19]. To evaluate the proposed compressor, however, we use exact encoders and apply approximate compressors to the exact partial products.

IV. Performance Evaluation

A. Accuracy analysis

An important metric for an approximate design is the output accuracy with respect to the exact result. We used the mean relative error distance (MRED) [21] as the metric to quantify the accuracy of the approximate designs. In order to compute the MRED, we first define the relative error distance (RED) by

$$ED = |M' - M|,$$
$$RED = \frac{ED}{M}. \quad (9)$$

In (9), M' denotes the generated approximate result while M denotes the exact result. The average value of all the REDs over the entire input space gives the MRED for each design.

Table 9 shows the MRED, the error rate (ER), and the normalized mean error distance (NMED, the mean error distance normalized by the maximum output of the accurate design) for several 16×16 unsigned multipliers recently reported in the literature. Note that the ER is the percentage of the multiplications for which the approximate design produces a different result than the exact one. Better designs will tend to have a low ER in addition to a small MRED.

Since an exhaustive simulation of all possible input combinations is very time-consuming, we simulated the accuracy of the approximate multipliers using Matlab with 10 million uniformly distributed input combinations [8], [10]. Altogether, MRED, ER, and NMED were simulated for 8×8 multipliers over their entire input space (65536 cases) and the results are also provided in Table 9.

The results in Table 9 show that the most accurate of the proposed 16×16 designs, M16-5 and M16-6, are more accurate than their competitors except AM2-16, which has the same MRED as M16-5. However, with respect to the ER and NMED, M16-5 is clearly more accurate than AM2-16. It is worth mentioning that the parameter k for AM2 and TAM2 is the number of MSBs used for error reduction [9], [5] and for ETM k indicates the number of LSBs in the non-multiplication part

[14]. k is also the mode number for AWTM and ACM [13], [11] and the vertical broken length in BAM [12]. Note that according to Table 9, the same trend as in 16×16 multipliers can be seen in 8×8 multipliers. In fact, the more significant MRED metric have similar values in both 16×16 and 8×8 multipliers.

Table 9. Accuracy comparison for 16×16 approximate multipliers.

Multiplier Size	Multiplier Type	MRED	ER (%)	NMED
16×16	**M16-1**	0.0644	96.71	5.7×10^{-2}
	M16-2	0.0839	96.67	7.2×10^{-2}
	M16-3	0.0168	94.74	1.2×10^{-3}
	M16-4	0.0224	94.65	1.9×10^{-3}
	M16-5	**0.0013**	72.49	$\mathbf{5.1\times10^{-6}}$
	M16-6	0.0017	**72.33**	5.7×10^{-6}
	UDM [15]	0.0333	80.99	1.4×10^{-2}
	AM2-16 [9]	**0.0013**	97.96	5.3×10^{-6}
	ETM-7 [14]	0.0156	99.99	2.2×10^{-3}
	ACM4 [11]	0.0026	99.97	6.4×10^{-6}
	MUL2 [18]	0.0020	84.67	7.1×10^{-6}
	DSM-8 [22]	0.0026	99.88	1.0×10^{-5}
	BAM-16 [12]	0.0021	99.97	3.5×10^{-5}
	TAM2-16 [5]	0.0020	99.98	3.1×10^{-5}
	AWTM-4 [13]	0.0033	99.94	8.3×10^{-6}
8×8	**M16-1**	0.0649	73.17	1.9×10^{-2}
	M16-2	0.0846	73.17	2.8×10^{-2}
	M16-3	0.0170	66.36	2.1×10^{-3}
	M16-4	0.0227	66.43	3.2×10^{-3}
	M16-5	**0.0013**	**36.22**	$\mathbf{6.8\times10^{-5}}$
	M16-6	0.0018	**36.22**	9.6×10^{-5}
	UDM [15]	0.0328	47.09	1.4×10^{-2}
	AM2-16 [9]	0.0014	95.23	5.3×10^{-4}
	ETM-3 [14]	0.0846	93.10	1.3×10^{-2}
	ACM4 [11]	0.0028	99.03	1.2×10^{-4}
	MUL2 [18]	0.0022	79.23	3.1×10^{-4}
	DSM-8 [22]	0.0031	99.47	5.6×10^{-3}
	BAM-16 [12]	0.0176	99.23	1.8×10^{-2}
	TAM2-16 [5]	0.0024	99.11	7.2×10^{-4}
	AWTM-4 [13]	0.1532	99.92	5.4×10^{-3}

We also measured the MRED, ER, and NMED for the radix-4 Booth multiplier. This proposed design is referred to as the compressor-based approximate Booth multiplier (CABM) and is compared to two state-of-the-art approximate Radix-4 Booth multipliers and the results are given in Table 10.

Table 10. Accuracy comparison of 8×8 Radix-4 Booth multipliers.

Multiplier Type	MRED	ER (%)	NED
AWBM1 [20]	0.051	98.26	0.30
AWBM2 [20]	0.029	91.49	**0.18**
CABM	**0.014**	**84.72**	**0.18**

Table 10 shows that the proposed design has the same NED as the AWBM2 while it has a smaller error rate. NED refers to the normalized error distance, which is the average error distance normalized by the maximum possible error. Moreover, CABM is the most accurate design with respect to the $MRED$.

B. Hardware analysis

All the designs were implemented in VHDL and then synthesized by using the Synopsys Design Compiler (DC) for ST's CMOS 28-nm process. The supply voltage and the temperature in all simulations were set to $1V$ and $25°C$, respectively. All designs were synthesized with the high effort on boundary optimization with a cell library that includes AND-OR-Inverter (AOI) logic gates. Note that the same timing constraints were used when synthesizing all multiplier designs. No attempt was made to find the optimal PDP since that search would impose different delays on the multipliers. Moreover, we used the default input-drive strength, output load, and switching activities for power analysis. It is also worth mentioning that we used Wallace-16 as the baseline exact multiplier for the comparison.

Table 11 shows the synthesized results for the circuit area, critical path delay, power consumption and the power-delay product for several designs. Note that the logic to produce generate and propagate signals is included in the implementation of the proposed designs.

As shown in Table 11, for the 16×16 unsigned designs, the fastest and the smallest design is ETM-7, which is 3.08% faster and 40.74% smaller and than our fastest and smallest design, M16-2; however, ETM-7 consumes 15.22% more power than M16-2. With respect to power consumption, the proposed designs M16-2 and M16-1 are the most power-efficient multipliers. Even our most accurate designs, M16-5 and M16-6, are among the most power-efficient and energy-efficient ones with relatively small PDP values. The only design that consumes less energy than M16-5 and M16-6, is ETM-7; however, ETM-7 is almost 10x less accurate than M16-5 and M16-6. Table 11 also shows that the proposed M16-2 has the lowest PDP value among all the designs.

Table 11. Hardware comparison of unsigned multipliers.

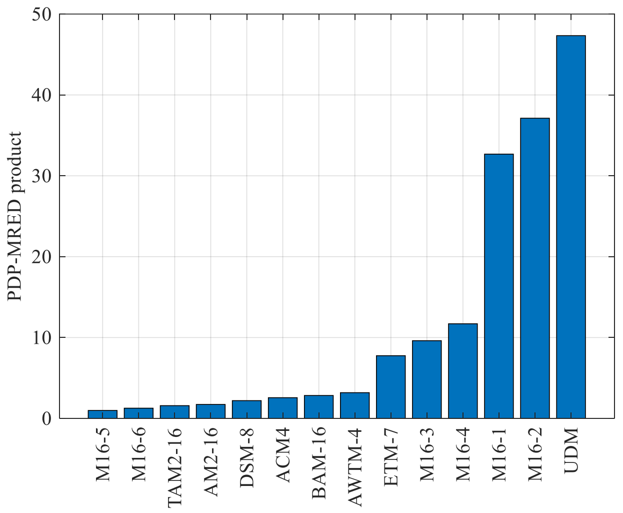
Multiplier size	Multiplier type	Delay (nS)	Power (μW)	Area (μm^2)	PDP (fJ)
16×16	**M16-1**	1.65	302.4	627.5	498.9
	M16-2	1.62	**268.4**	588.6	**434.8**
	M16-3	1.66	338.8	702.3	562.4
	M16-4	1.64	315.2	673.5	516.9
	M16-5	1.82	408.7	852.8	743.8
	M16-6	1.82	402.2	843.5	732.0
	UDM [15]	2.01	707.2	829.2	1421.47
	AM2-16 [9]	1.73	767.1	1045.1	1327.08
	ACM4 [11]	2.00	492.7	723.5	985.40
	MUL2 [18]	2.11	508.7	1011.5	1073.35
	ETM-7 [14]	**1.57**	316.6	**348.8**	497.06
	DSM-8 [22]	2.11	400.6	560.2	845.26
	BAM-16 [12]	2.34	442.6	441.0	1348.87
	TAM2-16 [5]	1.71	458.8	648.9	788.66
	AWTM-4 [13]	1.74	554.2	714.5	964.30
	Wallace-16	2.18	837.3	1034	1825.31
32×32	**M32-5**	3.35	841.1	1723.4	2817.68
	M32-6	3.35	**839.7**	1718.9	**2812.99**
	AM2-32 [9]	3.19	1601.4	2088.4	5108.46
	ACM4 [11]	3.54	1013.8	1501.6	3588.85
	MUL2 [18]	3.61	1203.7	2004.9	4345.35
	AWTM4 [13]	**3.16**	1344.9	**1431.1**	4249.88

The hardware measurements for 32×32 multipliers are also reported in Table 11. Note that the results for 16×16 designs showed that AM2, ACM4, MUL2 and AWTM4 are the four designs that have a comparable accuracy with M16-5 and M16-

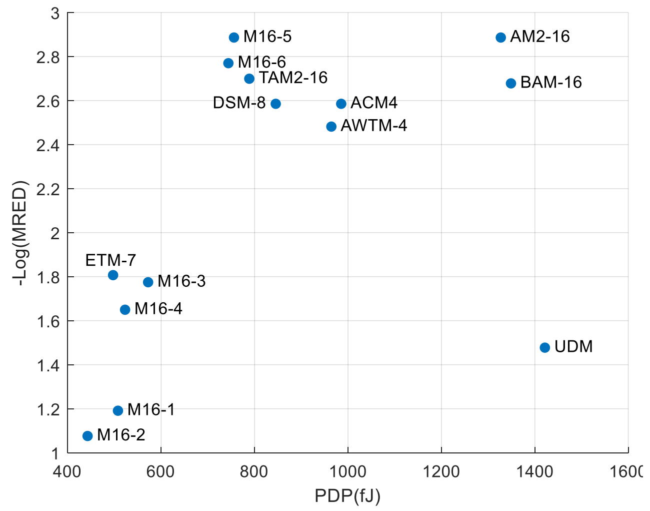
6 (in terms of MRED and NMED). Hence, we only report the power consumption, circuit area, and critical path delay for these six multipliers.

The same trend as in 16×16 designs can be seen in 32×32 multipliers. Clearly, M32-5 and M32-6 are the most hardware-efficient designs with at least 21.51% (for M32-5) and 21.61% (for M32-6) smaller PDP compared to ACM4, which has the smallest PDP among the other designs.

A similar comparison was done for the radix-4 Booth multipliers. As the results in Table 12 show, AWBM2 is slightly more efficient than the proposed CABM in terms of delay, power, and area; according to Table 10, however, it is more than 2x less accurate than the CABM. We further considered both MRED and PDP to evaluate different designs as in [8]. Fig. 5(a) compares the products of MRED and PDP values and Fig. 5(b) shows the -log$_{10}$(MRED) vs. PDP for the considered unsigned 16×16 multipliers.
Since the MRED values are so close, they are plotted on a logarithmic scale for better clarity. Note that designs at the top-left corner are the best designs, which have small PDPs with high accuracies. As the results in Fig. 5(a) show, M16-5 and M16-6 have the smallest PDP-MRED products.



(a) PDP-MRED product.



(b) MRED vs. PDP.
Fig. 5. MRED and PDP of the approximate multipliers.

The MRED-PDP products are also obtained for Radix-4 Booth multipliers and the results are given in Table 13. It is shown that the proposed CABM has the lowest MRED-PDP product.

Table 12. Hardware comparison of Radix-4 Booth multipliers.

Multiplier	Delay (nS)	Power (μW)	Area (μm^2)	PDP (fJ)
AWBM1 [20]	1.80	99.375	393.12	178.875
AWBM2 [20]	1.66	68.750	285.62	114.125
CABM	1.63	69.678	284.32	113.575
Exact Booth	2.01	125.42	436.87	252.094

Table 13. MRED-PDP product for three Radix-4 Booth multipliers.

Multiplier	PDP (fJ)	MRED	PDP × MRED
AWBM1 [20]	178.875	0.051	9.122
AWBM2 [20]	**114.125**	0.029	3.309
CABM	127.699	**0.014**	**1.788**

V. Applications

To evaluate the effectiveness of the proposed designs, we consider image sharpening and JPEG applications. In addition, an interference nulling calculation for the receiver in a MIMO wireless communication system is also considered as a new benchmark to evaluate approximate multipliers.

Note that 8×8 and 16×16 multipliers have been widely used in image processing applications in related articles [8]. Also, we scaled up the floating-point numbers in both JPEG and MIMO applications (numbers between 0 and 1) to 16-bit to have a good precision.

A. Image Sharpening

Image sharpening algorithms are widely used in image processing applications to enhance the sharpness of an image without producing halo artifacts [23], [24]. One image sharpening algorithm that uses approximate arithmetic is proposed in [25] and given by the following special filter:

$$S(x,y)=\frac{1}{4368}\sum_{i=-2}^{2}\sum_{j=-2}^{2}G(i+3,j+3)\,I(x-i,y-j),$$
$$G=\begin{pmatrix}16 & 64 & 112 & 64 & 16\\ 64 & 256 & 416 & 256 & 64\\ 112 & 416 & 656 & 416 & 112\\ 64 & 256 & 416 & 256 & 64\\ 16 & 64 & 112 & 64 & 16\end{pmatrix}. \quad (10)$$

In (10), I(x, y) denotes a pixel in the original image, S is the resulting processed image (using exact multipliers) and G defines the 5×5 impulse response of the spatial filter that operates on 5×5 blocks of pixels in the image [23].

The peak signal-to-noise ratio (PSNR) is an objective quality measure that is based on the mean squared error (MSE). (11) and (12) show how the MSE and PSNR are computed, respectively. Note that $\hat{S}$ denotes the processed image using approximate multipliers.

$$MSE=\frac{1}{mn}\times\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[\hat{S}(i,j)-S(i,j)]^2. \quad (11)$$
$$PSNR=10\times log_{10}(\frac{MAX_I^2}{MSE}). \quad (12)$$

Fig. 6 shows the original Lena test image and the seven sharpened images that used an exact multiplier and the six proposed approximate designs.

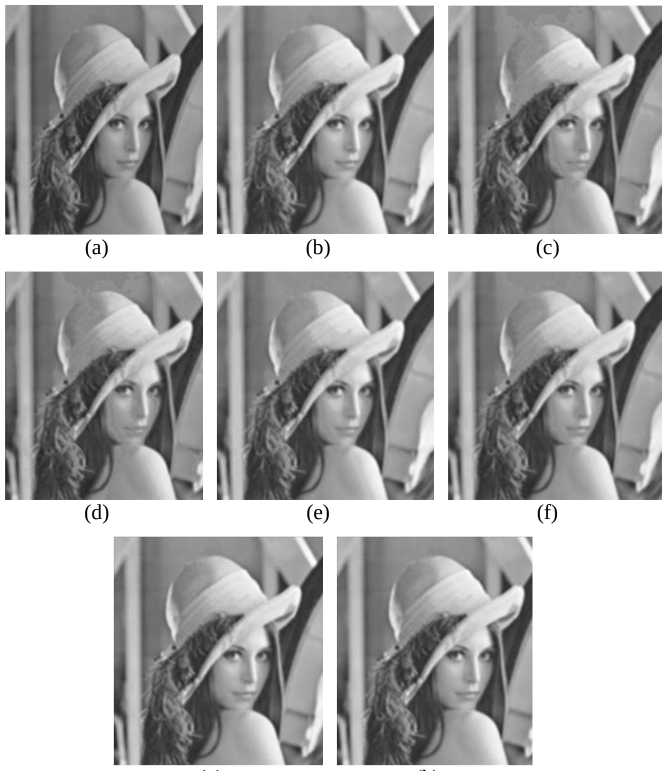


Fig. 6. Image sharpening. (a) Original image, and those sharpened using (b) Exact design, (c) M16-1, (d) M16-2, (e) M16-3, (f) M16-4, (g), M16-5, (h) M16-6.

Structural similarity SSIM [29] is another metric that measures the similarity between two images and is widely used as an image quality metric. We used the predefined Matlab function ***ssim*** to measure the SSIM between the sharpened image using the exact multiplier and those using an approximate multiplier. The PSNR and SSIM values for several approximate multipliers are depicted in Fig. 7, which shows that M16-5 is more accurate than the other designs. Note that with respect to the SSIM, our proposed designs, M16-5 and M16-6, are the best designs with the highest SSIM values. In general, the rankings of the weaker designs are slightly different in some cases but the trend is roughly the same as the PSNR values.

Comparing the designs that provide a PSNR of more than 30 dB (often a PSNR of 30 dB can be considered as good enough) reveals that M16-4 and M16-3 have a better trade-off with almost 72% and 70% smaller PDPs compared to the exact design (Table 11), respectively. Even the most accurate design, M16-5, achieves 59.25% saving on PDP, whereas the second-best design in terms of PSNR, AM2-16, has only 43.95% smaller PDP compared to the exact design (Table 11).

B. *JPEG Compression*

The JPEG compression standard is widely-used for saving storage space or transmission bandwidth for digital images [26]. This compression causes image quality degradation depending on the compression quality factor (QF). QF is a scaling factor ranging from 1 (high recovered image quality) to 100 (high compression ratio).

A basic strategy in JPEG image compression is to reduce the data correlation by transforming it from the time domain into the frequency domain. The human visual system is less sensitive to higher frequencies, therefore images can be compressed by suppressing their high-frequency components. The spatial-to-frequency domain transformation is done by applying the discrete cosine transform (DCT) [27].

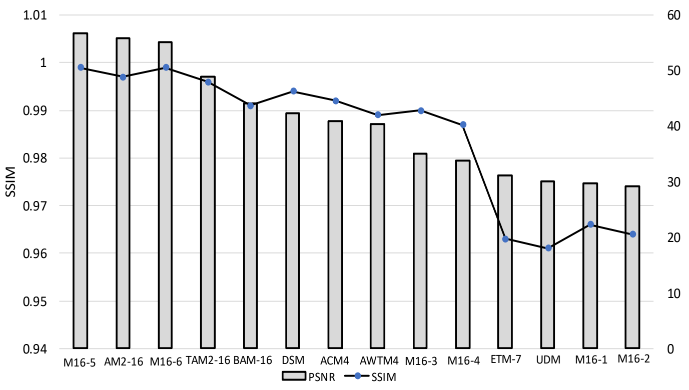


Fig. 7. PSNR and SSIM values for the image sharpening application.

In standard JPEG compression, the input image is divided into 8×8 pixel blocks. Then the DCT of each 8×8 pixel block is computed and unimportant DCT elements (that encode high frequencies) are discarded by multiplying the DCT coefficient matrix with a quantization matrix. The resulting matrix is then dequantized and its inverse DCT is computed. Finally, all the blocks are reassembled to form an image of the same size as the original one [28].

The matrix multiplication in the JPEG algorithm makes it a good application for evaluating approximate multipliers. Matlab code was written to compress the standard "camera man" image using the JPEG algorithm with the standard quantization matrix (Q) as given in

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}. \quad (13)$$

The original image and seven decompressed images, one using an exact multiplier and six using the proposed approximate multipliers, are shown in Fig. 8. The QF is set to 70 in all seven images. The quality of the decompressed images obtained by using exact and approximate multipliers are compared by using the PSNR and SSIM measures.

Table 14 reports the PSNR and SSIM values for several approximate multipliers for four increasing QFs. As the results in Table 14 show, M16-5 has the highest PSNR and SSIM values for the four considered QFs, followed by M16-6.
Note that the reference image for computing the SSIM and PSNR values in image sharpening application is the reconstructed image using exact multipliers. However, it would be more reasonable to use the original image, i.e. the image before compression, as the reference image in the JPEG compression application.

The results in Table 14 show that the exact multipliers in a JPEG compressor can be replaced by approximate multipliers for power and area saving purposes at the cost of negligible image quality degradation.

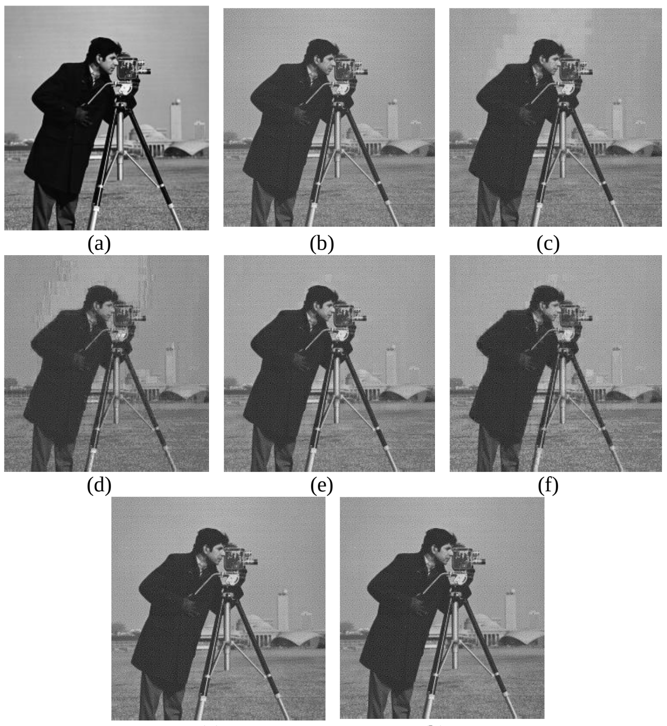


Fig. 8. JPEG compression for QF=70. (a) Original image, (b) Exact reconstruction, and those reconstructed using (c) M16-1, (d) M16-2, (e) M16-3, (f) M16-4, (g), M16-5, (h) M16-6.

Table 14. Decompressed image quality comparison.

Metric	Multiplier	QF=60	QF=70	QF=80	QF=90
PSNR	**Exact**	27.81	27.34	27.01	26.91
	M16-1	23.54	17.21	22.22	18.26
	M16-2	22.93	13.27	19.11	13.96
	M16-3	25.21	25.72	23.17	18.65
	M16-4	24.51	23.80	19.63	14.11
	M16-5	**26.43**	**26.65**	**25.72**	**25.54**
	M16-6	26.41	26.63	25.62	25.28
	AM2-16 [9]	26.17	26.07	25.51	24.48
	ACM4 [11]	26.02	25.95	25.13	24.21
	MUL2 [18]	26.21	26.44	25.64	25.37
	AWTM4 [13]	25.88	25.67	24.82	24.03
SSIM	**Exact**	0.98	0.98	0.97	0.97
	M16-1	0.83	0.79	0.79	0.79
	M16-2	0.83	0.79	0.79	0.73
	M16-3	0.95	0.93	0.90	0.87
	M16-4	0.95	0.93	0.90	0.83
	M16-5	**0.97**	**0.97**	**0.96**	**0.95**
	M16-6	**0.97**	0.96	**0.96**	**0.95**
	AM2-16 [9]	0.96	0.93	0.92	0.90
	ACM4 [11]	0.95	0.92	0.92	0.89
	MUL2 [18]	0.96	0.96	0.95	**0.95**
	AWTM4 [13]	0.93	0.92	0.91	0.88

C. Multiple-Input Multiple-Output (MIMO) Systems

Today, MIMO technology is being employed in wireless communications instead of the conventional single-input single-output (SISO) technology due to its higher data bandwidth and power efficiency over multipath fading channels [30], [31].

In digital communication, a transmitted '1/0' could be changed to a '0/1' due to various factors, such as noise and fading. The ratio of erroneous bits to the total number of transmitted bits over a channel is called the bit error rate (BER). Channel coding is a technique where functionally dependent bits are inserted so that most of the errors occur in data transmission over noisy communication channels can be detected and corrected.

Given the error tolerance provided by error correcting codes, computation errors in an approximate design are mixed with the errors caused by noise so a system can recover some of the approximation errors using error detection and correction coding. We use four coding schemes to evaluate the performance produced by the proposed approximate multipliers. The evaluation is done by using BER vs. SNR (signal-to-noise ratio) curves in the standard way that is used to illustrate the error correcting performance of codes. Note that the BER is a function of the noise power, i.e. the higher the SNR the better (i.e. the lower) the BER.

1. Methodology and experimental setup

MIMO methods take advantage of multiple transceiver and receiver antennas to produce higher total data throughput, however they require sophisticated, and costly, signal processing in the receiver. We modeled an 8×8 MIMO system in which all multiplications in the receiver block use the proposed approximate multipliers. In addition, three different codes were considered: Hamming (7, 4), extended Golay (24, 12) [32] and two variants of low density parity check (LDPC) codes: LDPC (1024, 512), and LDPC (2048, 1024) [33]. The input bit stream is coded using one of the above coding techniques and then it is transmitted over a noisy channel from 8 antennas, as shown in Fig. 9.

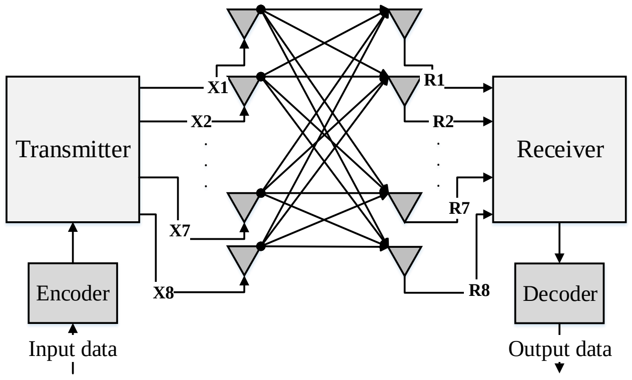


Fig. 9. Block diagram of an 8×8 MIMO system.

The system model for Fig. 9 is given by

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{N}. \tag{14}$$

in which $\boldsymbol{x}$ is the coded user bit stream, $\boldsymbol{H}$ is the channel matrix that models the interference in the channel, $\boldsymbol{N}$ models the additive white Gaussian channel noise, and $\boldsymbol{y}$ is the received corrupted bit stream. In an 8×8 MIMO system, $\boldsymbol{y}$, $\boldsymbol{x}$ and $\boldsymbol{N}$ are 8×1 complex matrices while $\boldsymbol{H}$ is a complex 8×8 matrix.

In the receiver block, the minimum mean squared error (MMSE) interference nulling matrix $\boldsymbol{w}$ is multiplied by the incoming signal vector $\boldsymbol{y}$. The MMSE approach aims to find the matrix w that minimizes the criterion, $E\{[\boldsymbol{wy}-\boldsymbol{x}][\boldsymbol{wy}-\boldsymbol{x}]^{\boldsymbol{H}}\}$. $\boldsymbol{w}$ is specified in

$$\boldsymbol{w} = [\boldsymbol{H}^{*}\boldsymbol{H} + \boldsymbol{N_0 I}]^{-1}\boldsymbol{H}^{*}. \quad (15)$$

where $\boldsymbol{N_0}$ is 2× the variance of the noise at the receiver antennas and $(.)^{*}$ is the conjugate transpose operator [34]. The final results at each receiver can be obtained by left-multiplying (using the approximate multipliers) the incoming signal vector y by the obtained w.

For the Hamming (7, 4) code, we generated 1420 4-bit samples and encoded them into 7-bit codewords. The extended (24, 12) Golay code works similarly to the Hamming (7, 4) code. The only difference is that 416 12-bit samples are generated, and the receiver decodes them using the maximum likelihood decoding technique [35]. Given a received codeword $\boldsymbol{x}$, the maximum likelihood decoding approach selects the codeword $\boldsymbol{y}$ from the codeword set that maximizes the conditional probability of $P(x|y\ sent)$. For both LDPC codes, the min-sum algorithm is used for decoding [36] with a maximum of 64 iterations. Note that the termination criterion is set to 1000 errors. At each SNR level, the min-sum algorithm is performed for either 64 iterations or 1000 errors, whichever condition is satisfied first. Clearly, increasing the number of iterations or the maximum number of errors in a block reduces the BER; however, doing so significantly increases the runtime. In this article, we aim to determine the general trend and reveal the practicality of the proposed approximate designs in MIMO receiver applications.

2. *Results and discussion*

The BER vs. SNR characteristic was computed for seven different cases: one for the exact multiplier and six for the six variants of the proposed design. The results are shown in Figs. 10 to 13 for the Hamming (7, 4), extended Golay (24, 12), LDPC (1024, 512), and LDPC (2048, 1024) codes, respectively.

Since the six proposed 16×16 approximate multipliers cover a wide range of accuracy, such that M16-2 and M16-5 are the least and the most accurate designs (Table 9), we only consider these six designs in this sub-section. First we aim to show the practicality of approximate multipliers in MIMO receiver applications in general. Second, we hypothesize that the performance of the other designs in the described MIMO system (Fig. 9) would be similar to one of the six proposed multipliers with the closest accuracy.

Figs. 10 to 13 show that for the lowest SNRs, and consequently, relatively high BERs, the exact and approximate designs are equally affected by noise. This implies that the computation errors caused by the use of approximate multipliers are insignificant compared to the errors caused by noise. Although the least accurate approximate multiplier designs should be quite acceptable at low SNR operation, there are few applications that will operate in that regime. When operating at higher, more typical SNR levels, Figs. 10 to 13 show that the most accurate variants of the proposed design, namely M16-5 and M16-6, can match the BER vs. SNR performance of a design that uses exact multipliers down to lower BERs.

As the SNR increases, the computation errors caused by the use of approximate multipliers will eventually dominate the random errors and produce a leveling off of the BER curve, a so-called error floor [37]. This is the operating region where the error correcting code cannot compensate for the multiplier's inaccuracies. This error floor can be easily seen, especially in Figs. 10 and 11 where the weakest codes, i.e. the (7, 4) Hamming code and the extended (24, 12) Golay code, are employed. Note that depending on the accuracy of the approximate design, the error floor is encountered at different SNR levels. The more accurate multipliers, such as M16-5 and M16-6, produce BERs that follow those of the exact design for higher SNRs compared to the BERs of the less accurate multipliers.

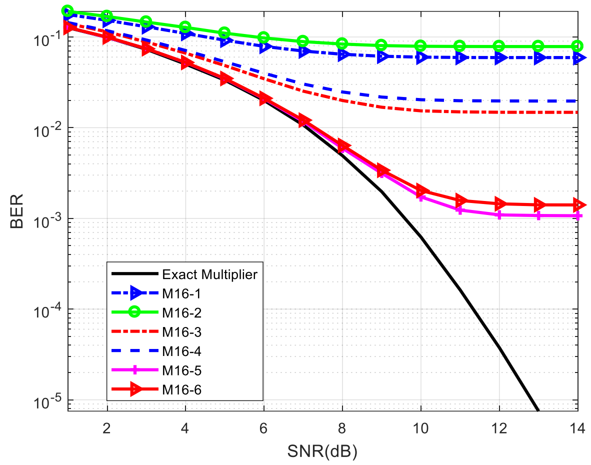


Fig. 10. BER vs. SNR for the (7, 4) Hamming code.

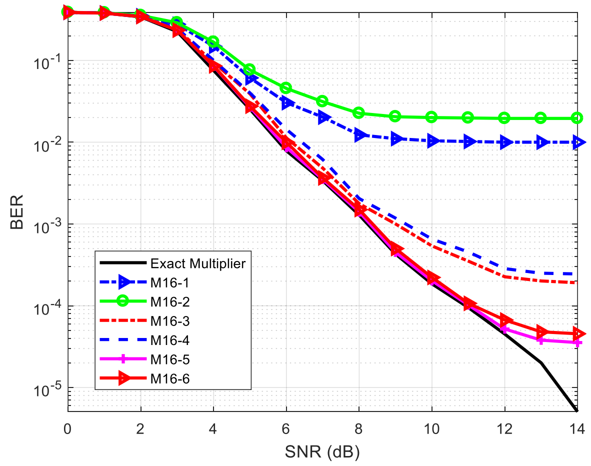


Fig. 11. BER vs. SNR for the extended (24, 12) Golay code.

According to the results in Figs. 10 and 11, the (7, 4) Hamming and the extended Golay codes are relatively weak, so they cannot compensate for the multipliers' inaccuracies and the BER cannot get lower than 10^{-3} for the (7, 4) Hamming code and almost 8×10^{-5} for the extended Golay code, which are entirely unacceptable for modern applications. However, Figs. 12 and 13 show that the most accurate designs, M16-5 and M16-6, produce BER performance that matches that of the exact multiplier down to much lower BERs when stronger LDPC codes are used.

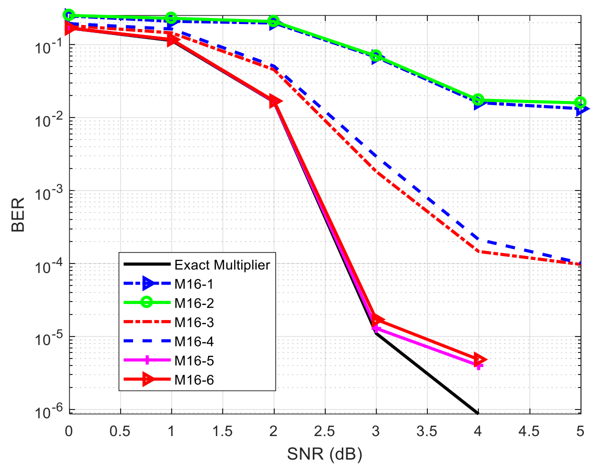


Fig. 12. BER vs. SNR for the (1024, 512) LDPC code.

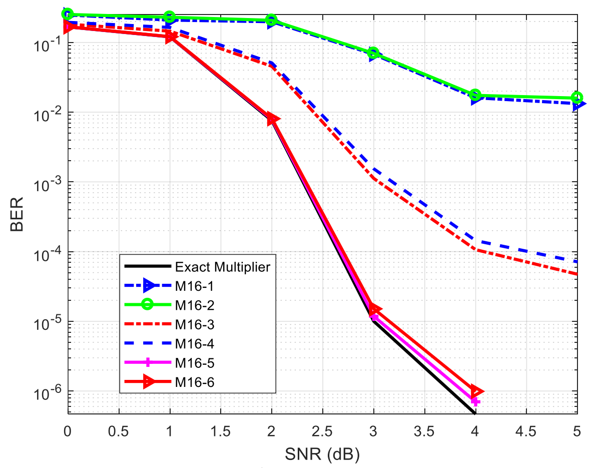


Fig. 13. BER vs. SNR for the (2048, 1024) LDPC code.

Generally, when stronger codes, such as the (2048, 1024) LDPC code, are employed, the approximate multipliers can match the BER performance of the exact design for higher SNRs (and hence lower BERs) since a stronger code can correct both random channel errors and approximation errors. In Fig. 13 the results of the strongest code that we considered, a (2048, 1024) LDPC code, show that the error floor for M16-4 happens at a BER lower than 10^{-4}. This means that for more accurate designs, such as M16-5 and M16-6, the error floor occurs at much lower BERs, probably somewhere closer to 10^{-7}.

Figs. 10 to 13 showed that approximate multipliers, especially M16-5 and M16-6, can safely replace exact multipliers in the MIMO system to reduce the power, delay, and area (M16-5 and M16-6 have 59.25% and 59.89% smaller PDP compared to the exact Wallace-tree multiplier, respectively) at a low cost in performance degradation. The advantages of the approximate multiplier implementation could be even more significant in larger MIMO systems, such as massive MIMO systems with 128 antennas, and also if many parallel multipliers are required to meet the required data throughput.

We performed some simulations and realized that using approximate multipliers increases the number of required iterations to get to a desired BER at a given SNR; the results are given in Table 15.

According to Table 15 the number of required iterations increases at higher SNR levels, where the errors caused by approximate multipliers dominate channel noise. Table 15 also shows that less accurate multipliers require more iterations to get to the desired BER at a given SNR level.

Table 15. Required increase in the number of iterations to get to a desired BER at a given SNR level.

Approximate Multiplier	(BER, SNR)	
	(10^{-5}, 3 dB)	(5×10^{-6}, 4 dB)
M16-5	3.5 %	9.3 %
M16-3	6.1 %	15.6 %

Analyzing the results at a reasonable operating point, e.g. an SNR of 4 dB for the (2048, 1024) LDPC code using the M16-5 approximate multiplier, shows a 9.3% increase in the number of required iterations. More iterations means more execution time and, consequently, more energy consumption. In fact, the energy consumption increases by 9.3%. However, as previously mentioned, M16-5 consumes 59% less energy than the exact multiplier and saves 20% on the area. Hence, it would still be practical to use approximate multipliers in this application.

Note that because LDPC simulation is intrinsically a probabilistic process because of the white Gaussian channel noise, we repeated the simulations 20 times and so the reported results in Table 15 are the average values.

VI. Conclusions

This article introduces an approximate 4:2 compressor that is employed to construct two 4×4 multipliers with different accuracies. The 4×4 designs are then scaled up to 16×16 and 32×32 multipliers that provide a wide range of accuracy-performance trade-offs. All six proposed multipliers are low-power designs. The least accurate of the proposed designs, M16-2, has the smallest PDP among other approximate designs (Table 11) while the most accurate of the proposed designs, M16-5, has 44% smaller PDP compared to AM2-16 that has a similar accuracy in MRED. Moreover, M16-5 is more accurate than the other approximate designs in the literature (Table 9). The proposed compressor is also employed in a radix-4 Booth multiplier, resulting in a low-power signed multiplier (CABM) with a small MRED. The simulation results reveal the advantages of the CABM over other designs in terms of MRED and PDP-MRED product.

The proposed multipliers are evaluated in image sharpening and JPEG applications. It is shown that M16-5 produces more accurate output than other approximate multipliers by achieving a higher quality (in terms of PSNR) while consuming less power. In addition, for the first time, approximate multipliers are evaluated in the interference nulling calculation of the MIMO baseband receiver. We measured how computation errors can be corrected along with errors caused by channel noise so that the transmitted data can be recovered without additional hardware cost using error detection and correction codes that are already present in the communication systems. It is shown that approximate multipliers can replace exact ones with low performance degradation. In the presence of strong channel codes, such as the LDPC (2048, 1024), the proposed most accurate design produces results close to the exact design with almost no performance loss for BERs of up to 10^{-6}.

Acknowledgment

The authors are grateful to Dr. Seok-Bum Ko and Suganthi Venkatachalam from the University of Saskatchewan for their comments and suggestions that led to improvements to the article. We would like to acknowledge the reviewers of this paper for their constructive comments, which we believe have strengthened the manuscript.

REFERENCES

[1] A. J. Sanchez-Clemente, L. Entrena, R. Hrbacek, and L. Sekanina, “Error mitigation using approximate logic circuits: a comparison of probabilistic and evolutionary approaches,” *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1871-1883, 2016.

[2] J. Schlachter, V. Camus, K. V. Palem, and C. Enz, “Design and applications of approximate circuits by gate-level pruning,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 5, pp.1694-1702, 2017.

[3] B. Moons and M. Verhelst, “Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 4, pp. 475–486, 2014.

[4] J. Han and M. Orshansky, “Approximate computing: an emerging paradigm for energy-efficient design,” *IEEE European Test Symposium*, pp. 1-6, 2013.

[5] C. Liu, “Design and analysis of approximate adders and multipliers,” *Master's Thesis*, University of Alberta, Canada, 2014.

[6] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, “AxNN: energy-efficient neuromorphic systems using approximate computing,” *International Symposium on Low power electronics and design*, pp. 27-32, 2014.

[7] K. Roy, A. Raghunathan, “Approximate computing: an energy-efficient computing technique for error resilient applications,” *IEEE Computer Society Annual Symposium on VLSI*, pp. 473-475, 2015.

[8] H. Jiang, C. Liu, L. Liu, F. Lombardi and J. Han, “A review, classification and comparative evaluation of approximate arithmetic circuits,” *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, no. 4, Article no. 60, 2017.

[9] C. Liu, J. Han, and F. Lombardi, “A low-power, high-performance approximate multiplier with configurable partial error recovery,” *Design, Automation & Test in Europe, 2014*, no. 1, pp. 1–4, 2014.

[10] C. H. Lin and I. C. Lin, “High accuracy approximate multiplier with error correction,” *IEEE International Conference on Computer Design*, pp. 33–38, 2013.

[11] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, “Design and analysis of approximate compressors for multiplication,” *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984–994, 2015.

[12] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, “Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, 2010.

[13] K. Bhardwaj, P. S. Mane, and J. Henkel, “Power- and area-efficient approximate Wallace tree multiplier for error-resilient systems,” *International Symposium on Quality Electronic Design*, pp. 263–269, 2014.

[14] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, “Low-power high-speed multiplier for error-tolerant application,” *IEEE International Conference on Electron Devices and Solid-State Circuits*, pp. 1-4, 2010.

[15] P. Kulkarni, P. Gupta, and M. Ercegovac, “Trading accuracy for power with an underdesigned multiplier architecture,” *IEEE International Conference on VLSI Design*, pp. 346-351, 2011.

[16] J. N. Mitchell, “Computer multiplication and division using binary logarithms,” *IRE Transactions on Electronic Computers*, vol. 4, pp. 512–517, 1962.

[17] N. Maheshwari, Z. Yang, J. Han, and F. Lombardi, “A design approach for compressor based approximate multipliers,” *IEEE International Conference on VLSI Design*, pp. 209-214, 2015.

[18] S. Venkatachalam, S. B. Ko, “Design of power and area efficient approximate multipliers,” *IEEE Transactions on VLSI*, vol. 25, no. 5, pp. 1–5, 2017.

[19] 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.

[20] L. Qian, C. Wang, W. Liu, F. Lombardi, and J. Han, “Design and evaluation of an approximate Wallace-Booth multiplier,” *IEEE International Symposium on Circuits and Systems*, pp. 1974-1977, 2016.

[21] J. Liang, J. Han, and F. Lombardi, “New metrics for the reliability of approximate and probabilistic adders,” *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760–1771, 2013.

[22] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, “Energy-efficient approximate multiplication for digital signal processing and classification applications,” *IEEE Transactions on Very Large Scale Integrgration Systems*, vol. 23, no. 6, pp. 1180–1184, 2014.

[23] A. C. Bovik, “Handbook of image and video processing,” USA, NY, New York, *Academic Press,* 2005.

[24] C. C. Pham and J. W. Jeon, “Efficient image sharpening and denoising using adaptive guided image filtering,” *IET Image Processing*, vol. 9, no. 1, pp. 71–79, 2015.

[25] M. S. K. Lau, K. V. Ling, and Y. C. Chu, “Energy-aware probabilistic multiplier: design and analysis,” *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems,* pp. 281-290, 2009.

[26] J. Yang, G. Zhu, Y. Q. Shi, “Analyzing the effect of JPEG compression on local variance of image intensity,” *IEEE Transactions on Image Processing*, vol. 25, no. 6, pp. 2647-2656, 2016.

[27] M. Shah, “Future of JPEG XT: privacy and security,” *PhD Thesis*, University of Texas Arlington, Texas, USA, 2016.

[28] N. Rathore, “JPEG image compression,” *International Journal of Engineering Research and Applications*, vol. *4*, no. 3, pp. 435-440, 2014.

[29] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity”, *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, 2004.

[30] Z. Babar, S. X. Ng, and L. Hanzo, “EXIT-chart-aided near-capacity quantum turbo code design,” *IEEE Transactions on Vehicular Technology*, vol. 64, no. 3, pp. 866–875, 2015.

[31] J. M. Chung, J. Kim, and D. Han, “Multihop hybrid virtual MIMO scheme for wireless sensor networks,” *IEEE Transactions on Vehicular Technology*, vol. 61, no. 9, pp. 4069–4078, 2012.

[32] P. Adde and R. Le Bidan, “A low-complexity soft-decision decoding architecture for the binary extended Golay code,” *IEEE International Conference on Electronics, Circuits and Syst. ICECS 2012*, pp. 705–708, 2012.

[33] H. Zhong, W. Xu, N. Xie, and T. Zhang, “Area-efficient min-sum decoder design for high-rate quasi-cyclic low-density parity-check codes in magnetic recording,” *IEEE Transactions on*

Magnetics, vol. 43, no. 12, pp. 4117–4122, 2007.

[34] D. Tse, “Fundamentals of Wireless Communications,” *Cambridge University Press*, 2005.

[35] M. Shirvanimoghaddam and S. Johnson, “Raptor codes in the low SNR regime,” *IEEE Transactions on Communications*, vol. PP, no. 99, pp. 1–12, 2016.

[36] J. Zhao, F. Zarkeshvari, and A. H. Banihashemi, “On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes,” *IEEE Transactions on Communications*, vol. 53, no. 4, pp. 549–554, 2005.

[37] W. Ryan and S. Lin, “*Channel codes: classical and modern*,” *Cambridge University Press*, 2009.