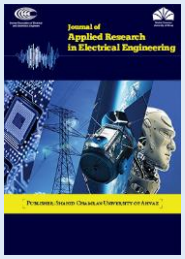
Shahid Chamran University of Ahvaz

Iranian Association of Electrical and Electronics Engineers

Journal of Applied Research in Electrical Engineering

Homepage: http://jaree.scu.ac.ir/



Research Article/ Review Article/ Applied Article

Design of Low-Power Approximate Logarithmic Multipliers with Improved Accuracy

Mojtaba Arab nezhad [1], Ali Mahani [1*]

[1] *Electrical engineering department, Shahid Bahonar University, Kerman, Iran*

[*] *Corresponding Author: amahani@uk.ac.ir*

Abstract: Approximate computing is considered a promising way to design high-performance and low-power arithmetic units recently. This paper proposes an energy-efficient logarithmic multiplier for error-tolerant applications. The proposed multiplier uses a novel technique to calculate the powers of two products to reduce critical path complexity. Also, a correction term is provided to improve the multiplier accuracy. Additionally, the use of approximate adders in our design is investigated, and optimal truncation length is obtained through simulations. We evaluated our work both in accuracy and hardware criteria. Experiments on a 16-bit proposed multiplier with approximate adder show that power-delay product (PDP) is significantly reduced by 34.05% compared to the best logarithmic multipliers available in the literature, while the mean relative error distance (MRED) is also decreased by 21.1%. The results of embedding our multiplier in the dequantization step of the JPEG standard show that the image quality is improved in comparison with other logarithmic multipliers; Also, a subtle drop in image quality compared to utilizing exact multipliers proves the viability of our design.

Keywords: Logarithmic Multiplier, Approximate Computing, Error-Tolerant

Article history
Received 19 December 2020; Revised 15 June 2021; Accepted 13 August 2021; Published online 13 August 2021
Note: Content is final as presented, with the exception of pagination.

How to cite this article
Ali Mahani; Mojtaba Arab nezhad. "Design of Low-Power Approximate Logarithmic Multipliers with Improved Accuracy", *J. Appl. Res. Electr. Eng.*, Vol. 2, No. 1, 2022, DOI: 10.22055/jaree.2021.36119.1018

1. INTRODUCTION

Significant computational demands of large-scale applications such as scientific computing, social media, and financial analysis have exceeded available resources [1]. Machine learning algorithms are becoming more accurate every day and, in many areas, have gone beyond human accuracy, but this accuracy comes at the expense of increased computations [2]. Due to recent advances in technology and the end of Dennard scaling, it has become difficult to improve the performance of computing systems at current power levels [3]. A wide range of applications that require huge computations can maintain their output well enough despite some computational error. Some of these applications are as follows [4]:

- Applications such as machine learning and adaptive filters that are inherently error-tolerant.
- In digital signal processing, because the inputs are often noisy, accuracy is limited.
- In image processing, due to limitations in human cognition, the existence of some errors in calculations is not detectable in the output.

Approximate computing introduces some errors in the calculations but simplifies the arithmetic operations. Therefore, approximate computing can be considered as a promising way to reduce power consumption. Approximate computing techniques can be applied to various levels, such as hardware, architecture, algorithm, and software [5].

Adders and multipliers are the arithmetic units that are the main subject of hardware-level approximations [6]. In the aforementioned applications, there are an abundant number of arithmetic processing that involve addition and multiplication. To design high-performance arithmetic processors, it is necessary to optimize the performance and power consumption of its main components, namely adders and multipliers. For this reason, much attention has been paid to approximate computing techniques at the circuit level to improve these units.

Multiplication is more elaborate than the addition operation and has always been a limiting factor to improve speed and area [7]. Hence enhancing this operation can result in considerable improvement in the whole design. Also, most applications mentioned above consist of some dominant kernels that intensively rely on multiplication. So, multipliers become primary candidates for approximation computing to improve whole system performance [8]. A conventional multiplier consists of partial product generation, accumulation, and final addition [9]. Various parts of a multiplier are capable of applying approximate techniques [5]. Different approximation approaches are proposed to design highly efficient multipliers [10]: Approximate recursive multipliers are built of 2×2 approximate multiplier blocks to form a complete multiplier [11]. In [12], a dynamic truncation method based on leading-one position has been introduced, which reduces the multiplier bit-width. Paper [13] proposed using an m×m multiplier to design approximate n×n multipliers where m<n. Approximate radix-4 booth multiplier [14] is another multiplication technique.

A different category from traditional multipliers are logarithmic multipliers that use binary logarithms to simplify multiplication operations. In the logarithm domain multiplication converts to addition. Multipliers that use logarithm transformation are inherently erroneous. Such error occurs for the following reasons: 1) a limited number of precision bits and 2) errors that happen at the time of transformation to the logarithmic system. Mitchell introduced the first logarithmic multiplier [15]. In conventional approximate multipliers, the accuracy is high, but the area and power consumption are also high. But in logarithmic multipliers reducing hardware overhead as well as reducing power, take precedence over multiplier accuracy. This property makes logarithmic multipliers suitable for large-scale applications that require high parallelism [16], [17].

In this paper, we introduce a new logarithmic multiplier to optimize power consumption and reduce hardware area and latency, while improving multiplier accuracy in terms of error amount as well as error distribution. The main contributions of this article are summarized as follows:

- A new multiplication algorithm is presented that uses less hardware resources than previous designs and is therefore more power efficient.
- We have introduced and used a correction term that improves the multiplier error characteristic.
- A new method has been proposed to calculate the product of the power of number two, which reduces the critical path delay significantly.
- The use of approximate adders in the proposed design has been investigated and the truncation length parameter for compromise between circuit complexity and multiplier approximation error has been introduced so that the proposed design can be adjusted for different applications.

The rest of the paper is organized as follows: in section 2, we have introduced logarithmic multipliers, notably Mitchell's algorithm. The main problems of these multipliers are described, and the main approaches to alleviate them are reviewed. The first part of section 3 is devoted to proposing the multiplication algorithm. The remainder of this section deals with the hardware architecture of the multiplier.

Error analysis and simulation results are presented in section 4. Section 5 implements our multiplier in JPEG image compression and decompression standard and evaluates the output image's quality. Finally, section 6 concludes the paper.

2. REVIEW AND RELATED WORKS

Due to the complexity of the multiplication operation, approximate multipliers are designed for trade-off between accuracy and design efficiency. various approximation methods have been proposed to simplify multiplier circuit. Exploring available references shows that approximation techniques in multipliers are mainly grouped in logarithmic and non-logarithmic categories. Non-logarithmic multipliers usually use approximation techniques to simplify different parts of a typical multiplier such as partial product generation [14], [22] and partial product accumulation [23], [24]. These multipliers have relatively low approximation errors and are more complex hardware instead.

Logarithmic multipliers, as their name implies, convert complex multiplication operation into simpler addition operation in the logarithm domain, which results in more compact hardware than non-logarithmic multipliers. unlike a conventional multiplier, a logarithmic multiplier needs logarithm conversion, addition, and antilogarithm stages. Because of inherent error in logarithm transformation, they are approximate multipliers. There are different ways to convert binary numbers into the logarithmic numbers system: 1) iterative methods, which are very time-consuming and need several cycles to converge to an acceptable result. 2) look-up table-based methods that are accurate but need complex and increased hardware. 3) using a piece-wise linear approximation of the function $\log x$. The third method is high-speed, and implementing this method needs relatively fewer resources.

The First logarithmic multiplier, which uses a piece-wise linear approximation, was proposed by Mitchell [15]. Here, the algorithm is briefly expressed. Assume that we want to multiply two fixed-point numbers A and B; they can be represented in the form $2^{k_{A,B}}(1 + x_{A,B})$ and x_A, x_B are between [0,1). 2^{k_A} and 2^{k_B} are the largest powers of two smaller than or equal to A and B, respectively. It means k_A and k_B represent the position of the most significant one in A and B. Taking the logarithms of A and B, we have $\log_2 A, B = k_{A,B} + \log_2(1 + x_{A,B})$. Mitchell's method to compute this term is to use the approximation $\log_2(1 + x) \approx x$. Thus, the multiplication is simply calculated with only shift and add operations.

The problem with Mitchell's algorithm is that this method has a relatively large error and always underestimates the logarithms, so the product is, in any case, smaller than or equal to exact results. The Mitchell's multiplier accuracy improvement methods can be categorized into four main groups [25], as shown in **Fig. 1**.

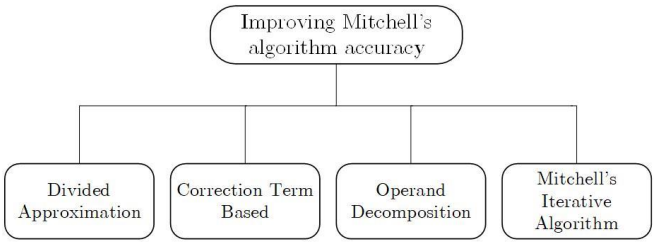


Fig. 1: Different classes of improving Mitchell's accuracy

Mitchell's method is based on a piece-wise linear approximation in which the lines are in intervals between powers of two. Each line has two intersections with the exact logarithm curve; intersections are at powers of two. In divided approximation methods, a range of Mitchell's algorithms is divided into some more fine-grained intervals, and in each, a more precise equation is derived to approximate the curve better.

In [26], the authors proposed implementing a logarithm converter based on Mitchell's method. For accuracy improvement, regions between powers of two are split into some smaller intervals, and within them, a more precise equation is placed instead of Mitchell's original equation for approximating the logarithms. Worst case relative error in original Mitchell's conversion is 5.36%, but in [26], this error is reduced to 0.93, 0.43, and 0.15 percent for 2-region, 3-region, and 6-region correcting algorithms, respectively. For calculating these equations, an error-correcting circuit must be placed in hardware, which means increased hardware.

Mahalingam et al. [25] used the operand decomposition technique to reduce the error in Mitchell's multiplier. Operand decomposition was first introduced in [27] to reduce the array's switching activity and the tree multipliers. In [15], Mitchell showed that his algorithm is more accurate when there is no carryover in the mantissa part during the summation step. Operand decomposition reduces the number of "1" bits in the decomposed operands; this means less chance to produce a carryover from the mantissa part into the integer part when summing up the logarithms. It has been shown that operand decomposition reduces Mitchell's multiplication error by 44.7% on average, and to achieve accuracy further, this work can be used with other error reduction methods. The main drawback of this work is its hardware overhead: The multipliers are doubled; moreover, there is a need for a decomposition circuit and an adder to compute the final product.

Correction term-based approaches add a term to the results obtained from the original Mitchell's algorithm to reduce the error. McLaren in [28] showed that multiplication error is only related to the mantissa part (fractional parts x_A and x_B), thus repeating for every characteristic. For error correction, different correction values would be added to the final result based on various combinations of x_A and x_B; but it is impractical. McLaren split the range of $x \in [0,1)$ into eight regions of 0.125 and made a table of correction values for each combination of these ranges. With this modification mean of the errors reduces from 3.614 to 0.0363. the paper states that their method has increased area and power consumption about 30% over the original Mitchell's algorithm.

The first iterative logarithmic multiplier was presented in [15]. The magnitude of error in Mitchell's algorithm is $2^{k_1+k_2}(x_1x_2)$ when there is no carry and $2^{k_1+k_2}(x_1'x_2')$ when we have carryover from mantissa respectively (x_1' and x_2' are the two's complements of x_1 and x_2). Considering x_1x_2 or $x_1'x_2'$ as a new product, if this term is computed with another Mitchell multiplier and this correction value is added with the approximate product computed before, the error reduces significantly at the cost of extra hardware.

Several articles have attempted to optimize Mitchell's multiplier hardware. In [5], three different approximate adders are exploited in the adder stage of a logarithmic multiplier. They tried Various truncation lengths for adders and reported the effects on hardware efficiency and error criteria. The logarithmic multiplier in [17] was improved in different aspects: they used efficient fully parallel leading-one detectors, exploited efficient shift amount calculation, and finally introduced parameter w (the truncation width) and designed a customizable logarithmic multiplier for compromising between hardware costs and accuracy. A modified exact adder is proposed in [16], as in the final addition of the multiplier, some states do not occur; they can use a simplified adder.

The one-sided error distribution of Mitchell's method is another problem that must be considered. In [28], correction terms changed the distribution. About 68% of errors fall in the range -1.21 and 1.29, while errors in the original algorithm are between 0.507 and 6.721. in [5], using an inexact set-one adder causes a somewhat double-sided error distribution. Authors in [29] have proposed a novel logarithm conversion algorithm that differs from Mitchell's. in this algorithm, instead of finding the most significant power of two smaller than the operands, they find the nearest power of two to the operand. This modification leads to a reduced error and a double-sided error distribution, which avoids error accumulation in many applications like matrix multiplication.

However, finding the nearest ones needs more complex hardware and leads to dealing with negative numbers and subtractors. In the next section, we will present another way to improve accuracy, which at the same time reduces hardware costs. In section 4, we discuss selecting the approximation, which keeps the distribution of errors double-sided.

3. PROPOSED METHOD

In this section, our proposed approximate multiplier is introduced. At first, the multiplication algorithm is described, and then multiplier hardware is investigated.

3.1. Multiplication algorithm

Consider operands A and B that have to be multiplied. We can represent operands as follows:

$$\begin{cases} A = h_1 + q_1 = 2^{k_1} + q_1 \ \ where \ \ 0 \le q_1 < 2^{k_1} \\ B = h_2 + q_2 = 2^{k_2} + q_2 \ \ where \ 0 \le q_2 < 2^{k_2} \end{cases} \quad (1)$$

equation 1 shows the operands are decomposed into the largest power of two smaller or equal to them plus an extra term. So, the multiplication becomes:

$$P_{exact} = A \times B = 2^{k_1+k_2} + 2^{k_1}q_2 + 2^{k_2}q_1 + q_1q_2 \quad (2)$$

$$P_{approx} = 2^{k_1+k_2} + 2^{k_1}q_2 + 2^{k_2}q_1 + q_1q_{2_{approx}} \quad (3)$$

As seen in equation 2, the first term is 2 to the power of $k_1 + k_2$ Which can be simply computed with a shift operation. In [29], term $2^{k_1+k_2}$ was calculated by giving the summation of k_1 and k_2 to a decoder. However, here we directly shift 2^{k_1} to the left by the amount of k_2. Two other terms, $2^{k_1}q_2$ and $2^{k_2}q_1$, are products of an arbitrary number and a power of two. To produce these terms, we shift q_1 and q_2 to the left, respectively, by k_2 and k_1. In order to compute the last term, q_1q_2, which itself is a product term, we have used approximation. The approximation is as follows: q1 and q2 are approximated to the largest power of two smaller or equal to them, as shown in equation 4:

$$\begin{cases} q_1 = 2^{m_1}(1 + r_1) = 2^{m_1}x_1 & 0 \le r_1 < 1 \\ q_2 = 2^{m_2}(1 + r_2) = 2^{m_2}\,x_2 & 0 \le r_2 < 1 \end{cases} \quad (4)$$

With the approximation mentioned above, we approximate q_1 and q_2 as equation 4 with $k \in \{1,2,4\}$.

$$q_1q_2 = k \times 2^{m_1+m_2} \quad (5)$$

Computing this term becomes similar to the calculation of $2^{k_1+k_2}$Which was discussed earlier, and then calculating coefficient k. So, we can obtain this term by shifting 2^{m_1} to the left by m_2 .thus k is a power of two; the result can be obtained only by shifting $2^{m_1+m_2}$ to the left. The only approximation used in this work is the computation of the q_1q_2 term. The reason why we used such approximation is discussed in the next section. The complete workflow of the proposed multiplier is described in **Fig. 2**.

Algorithm 1 Proposed Approximate Multiplier

Inputs: A, B: n-bits
Output: P: $2n$-bits

// Find the leading one in A and B:
$h_1 \leftarrow \mathrm{LOD}(A)$
$h_2 \leftarrow \mathrm{LOD}(B)$
// Find q_1 and q_2:
$q_1 \leftarrow A[n-2:0]$ XOR h_1
$q_2 \leftarrow B[n-2:0]$ XOR h_2
// Find the position of the leading one:
$k_1 \leftarrow \mathrm{PE}(h_1)$
$k_2 \leftarrow \mathrm{PE}(h_2)$
// Calculate $2^{k_1+k_2}$
$2^{k_1+k_2} \leftarrow 2^{k_1} << k_2$
$T_1 = 2^{k_1}q_2 \leftarrow q_2 << k_1$
$T_2 = 2^{k_2}q_1 \leftarrow q_1 << k_2$
$m_1 \leftarrow \mathrm{PE}\ (q_1)$
$2^{m_2} \leftarrow \mathrm{LOD}\ (q_2)$
$2^{m_1+m_2} \leftarrow 2^{m_2} << m_1$
$2^{m_1+m_2+1} = 2^{m_1+m_2}$ & $'0'$
$T_3 = 2^{k_1+k_2} + 2^{m_1+m_2+1} \leftarrow 2^{k_1+k_2}$ OR $2^{m_1+m_2+1}$
$P \leftarrow T_1 + T_2 + T_3$

Fig. 2:Proposed Multiplication Algorithm

3.2. Correction term selection

To select the best option for approximating q_1q_2, three different k were candidates, i.e., 1, 2, and 4. This brings us three approximations $2^{m_1+m_2}$, $2^{m_1+m_2+1}$, and $2^{m_1+m_2+2}$. The absolute error for each option is calculated in equation (8).

$$error = \left|P_{exact} - P_{approx}\right| \quad (6)$$

Concerning equations (2) and (3), the equation (6) becomes:

$$error = \left|q_1q_2 - q_1q_{2_{approx}}\right| \quad (7)$$

$$\Rightarrow \begin{cases} error_1 = |2^{m_1+m_2}x_1x_2 - 2^{m_1+m_2}| & (k=1) \\ error_2 = |2^{m_1+m_2}x_1x_2 - 2^{m_1+m_2+1}| & (k=2) \\ error_3 = |2^{m_1+m_2}x_1x_2 - 2^{m_1+m_2+2}| & (k=4) \end{cases} \quad (8)$$

To select the best option, we decided to pick the k, which in most cases gives us the least error. To do so, two conditions were examined, and solving these inequalities leads to the following circumscriptions:

$$error_1 < error_2 \ \Rightarrow x_1x_2 < 1.5\,, \ \ 0 \le x_1,\ x_2 < 1 \quad (9)$$

$$error_2 < error_3 \ \Rightarrow x_1x_2 < 3\,, \quad 0 \le x_1,\ x_2 < 1 \quad (10)$$

x_1x_2 product is plotted in **Fig. 3**, and red and blue lines show the borders where x_1x_2 is 1.5 and 3 respectively. This plot clearly shows that in most cases (about 68%), $k = 2$ i.e.,

$2^{m_1+m_2+1}$ approximation for q_1q_2 has the minimum error, so we selected it for our design.

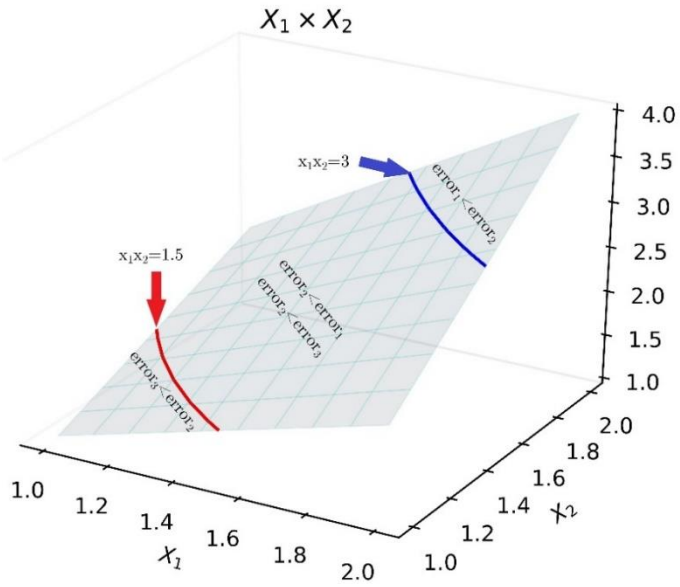


Fig. 3: x1 and x2 product plot separated by 1.5 and 3 lines

3.3. Hardware architecture

The hardware implementation of our proposed multiplier is described. The multiplier block diagram is shown in **Fig. 4**. LOD units are leading-one-detectors, which their structure is taken from [30]. LOD finds the most significant 1 in its input and keeps it in output while making other bits zero. Priority encoder (PE) determines the position of the most valuable 1 in number. It also has a zero flag, which becomes high in the case of zero input. Shifter blocks are combinational barrel shifters, and their architecture is the same as shifters proposed in [31].

Operands A and B are given to LOD1 and LOD2 as inputs. The PE1 and PE2 take the LOD1 and LOD2 outputs, which are in one hot representation format, and calculate k_1 and k_2. q_1 and q_2 are computed by XORing h_1 and h_2 with A and B. We have used a novel approach to calculate term $2^{k_1+k_2}$. In [29], the authors have used an adder and a decoder after PEs to find this term's value. However, in this paper, the adder and the decoder are eliminated. Instead, we have placed a shifter after LOD1, and the shift amount comes from PE2, and PE1 is no longer on its path. With these modifications, it seems the level of logic and hardware area must decrease to some extent. Shifter2 and shifter3 are responsible for calculating terms $2^{k_1}q_2$ and $2^{k_2}q_1$, respectively. To approximate q_1q_2, first, q_1 is given to PE3 to obtain m_1. Note that there is no need for LOD because the PE itself finds the position of most significant '1'. On the other hand, q_2 transfers through LOD3, and 2^{m_2} is computed. With the use of shifter4, we calculate $2^{m_1+m_2}$. In this paper, to reduce the mean error, we used approximation $2^{m_1+m_2+1}$. We reach this term easily by concatenating a '0' on the right side of $2^{m_1+m_2}$.

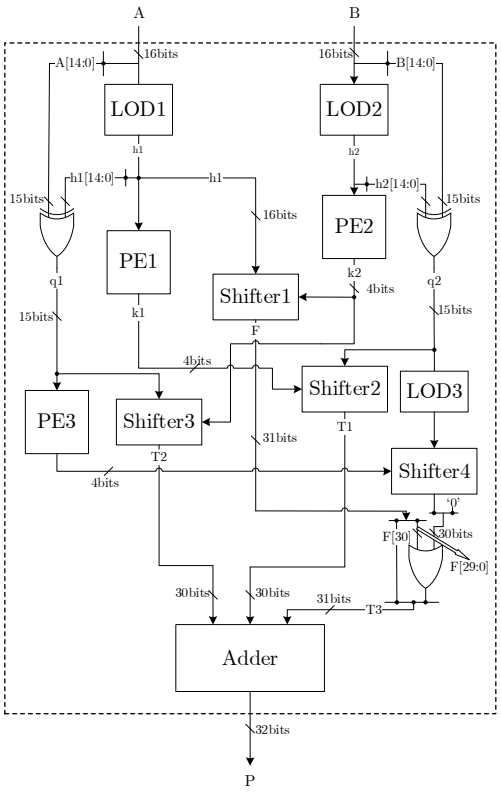


Fig. 4: Proposed Approximate Multiplier

The four terms calculated before must be added to produce the final result. To reduce the complexity of the adder stage, we consider two terms $2^{k_1+k_2}$ and $2^{m_1+m_2+1}$. Thus $2^{k_1+k_2}$ is a power of 2 and is always greater than $2^{m_1+m_2+1}$; we can OR them to find the addition. Lastly, the three terms are summed up in an adder, and the final product is obtained.

3.4. exploiting approximate adders

A large part of the logarithmic multiplier is devoted to adders. This urged us to investigate the use of approximate adders in our design. The utilization of approximate adders in logarithmic multipliers has been studied in [5]. Some types of approximate adders were exploited, and the results showed that set-one adders outperform other types. In [29], a modified version of set-one adders was introduced and employed in their logarithmic multiplier. Therefore, we bring our attention to set-one adders and explore the performance of our proposed multiplier with them.

An n-bit set-one adder with m truncated bits (SOA-m) is composed of an m-bit approximate part for the least significant bits (lower part of the augend and addend) and an exact part for (n−m) most significant bits. N-bit SOA-m is depicted in **Fig. 5**. Following expressions describe lower m bit of a set-one adder:

$$sum[m-1:0] = 1 \tag{11}$$

$$c_{in} = a[m-1] \text{ AND } b[m-1] \tag{12}$$

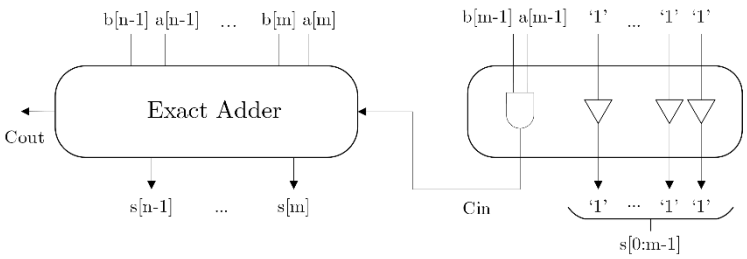


Fig. 5: SOA with m truncation bits

The adder used in our design sums up three terms to build the product. It is composed of a set of 3 to 2 compressors (full adder units) followed by a ripple carry adder. As the bit width of the result is twice the inputs in multiplication, the adder is costly in terms of power and area, and the carry chain causes a relatively high delay. Set-one adder can alleviate hardware overhead because there are no logical circuits for calculating the 'm' right-hand bits of the result. So, there is a reduction of 2m full adders in our design (m full adders in compressor stage plus m full adders in the ripple carry adder). The delay also significantly improves since SOA shrinks lengthy carry chain by m bits.

The effect of approximation bit numbers in the final adder on the accuracy of the proposed multiplier is investigated to pick the best value of m. MRED criterion is chosen for this purpose. As presented in **Fig. 6**, for a 16-bit multiplier, selecting m to values up to 16 nearly has no impact on our multiplier accuracy. Therefore, we chose 16 for maximum hardware saving. This means our design is more robust than previous works in [5] and [29], in which the MRED started to immediately increase when m was larger than 11 and 15, respectively.

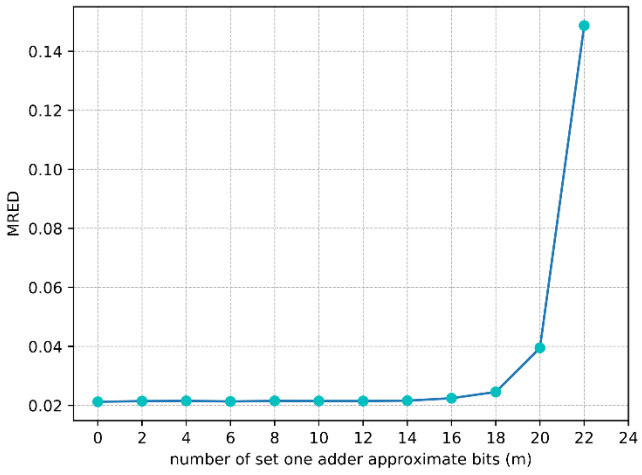


Fig. 6: effect of the number of approximate bits in adder stage on the accuracy of the 16-bit multiplier

4. EXPERIMENTAL RESULTS

we evaluate our work and compare it with some similar works available in the literature in this section. Prior to experimental results, error metrics for approximate designs are introduced. These criteria are measured for our proposed algorithm. Hardware simulations are done, and hardware metrics such as area, power, and delay are assessed. For evaluation, two 16×16 multipliers with both exact and approximate adders have been considered.

4.1. Accuracy evaluation

To assess the accuracy and error characteristics of the multiplier, the multiplication algorithm is implemented in behavioral level. Because exhaustive simulations are time-consuming, 10^7 pairs of random inputs were given to the model, and results were obtained. Error metrics, including error rate (ER), mean related error distance (MRED), and normalized error distance (NMED), are calculated. For comparison, multiplier designs available in papers [29], [15], [5] are considered, and the results are listed in **Table 1**. As expected, simulation results verify that our multiplier outperforms in terms of accuracy metrics. Because of an additional correction term, the MRED and NMED measures of the proposed algorithm are lower than LM [15] and ALM-SOA [5]. Although nearest-one detectors in [29] are removed, a good selection of correction terms can compensate for the effects, and even results show a reduction in mean relative error to about 25.6% than the best available work [29]. It is evident from **Table 1** that using set-one adders in our multiplier does not affect the accuracy severely, and even with a high number of truncated bits (m), accuracy metrics stay about their values with an exact adder.

Table 1: Error metrics

Multiplier	MRED	ER (%)	NMED
LM [15]	0.0384	99.77	0.0092
ILM-EA [29]	0.0289	99.95	0.0069
ALM-SOA-11 [5]	0.0330	98.97	0.0080
Proposed (Exact Adder)	0.0215	99.95	0.0064
Proposed (SOA-16 Adder)	0.0228	99.99	0.0064

4.2. Hardware evaluation

We coded a 16-bit multiplier for hardware assessment based on the proposed algorithm. All hardware simulations were done in Synopsys Design Compiler with default settings, using TSMC 180nm technology. **Table 2** presents obtained results from simulations. Compared to the proposed multiplier in [29], we removed costly subtractors and nearest-one detectors and replaced them with an array of XORs and leading-one detectors, respectively, expecting a power and area reduction in our design. Results confirmed that our modifications caused a meaningful improvement in both area and power consumption. The critical path in [29] is related to the path where the term $2^{k_1+k_2}$ is calculated. This path goes through an adder for computing $k_1 + k_2$ and a priority

encoder for calculating $2^{k_1+k_2}$. To reduce the delay, we proposed a novel way for $2^{k_1+k_2}$ computation, in which the adder and decoder are replaced with a shifter. The results show a 24% reduction in critical path delay with respect to ILM-EA.

As discussed in section 3.4, our algorithm has more persistence to approximation in its final adder so that we can exploit this characteristic for further hardware improvements. Simulation results show that by setting truncation bits (m) to 16, we can achieve significant hardware savings and reduce the large carry chain of the final adder to half its length.

Table 2: Hardware metrics

Multiplier	Power (mW)	Delay (nS)	Area (μm^2)	PDP (pJ)
LM [15]	6.00	34.94	146338	209.64
ILM-EA [29]	8.85	34.49	158629	305.23
ALM-SOA-11 [5]	4.29	23.50	124871	100.815
Proposed (Exact Adder)	5.31	29.29	139595	155.53
Proposed (SOA-16 Adder)	4.96	20.68	122214	102.57

To decide which multiplier design is preferable overall, i.e., both accuracy and hardware metrics, we compared PDP×MRED of the multipliers. The results are presented in **Table 3**.

Table 3: PDP×MRED of approximate multipliers

Multiplier	PDP×MRED
LM [15]	8.05
ILM-EA [29]	8.82
ALM-SOA-11 [5]	3.32
Proposed (Exact Adder)	3.34
Proposed (SOA-16 Adder)	2.33

Multiplier designs with lower MRED and PDP and, as a result, with lower PDP×MRED are more favorable. As seen from **Table 3**, our proposed multiplier with approximate adder has the least PDP-MRED product, and therefore it's the most hardware-efficient design over others while considering accuracy.

5. JPEG APPLICATION

We employed our work in the real-world application JPEG, an image compression standard [32], To show our proposed approximate multiplier's applicability. Image compression in jpeg is as follows: the image is first partitioned in 8×8 blocks. Then Discrete Cosine Transform (DCT) is calculated for each block, and after that, the quantization step is done. This step is attained by dividing the matrix of DCT coefficients by the quantization matrix in an element-wise fashion and rounding the results. Then an entropy coding is applied to the resulted matrix to reduce the image size. Image decompression starts by decoding the data and then dequantizing the blocks. Dequantizing is done by multiplying the matrix of quantization into each block. This step is where we have exploited our multiplier. After dequantization, the inverse of DCT (IDCT) is computed, and the image is formed.

For evaluating the applicability, we coded the lossy JPEG standard in MATLAB. We then implemented some approximate multipliers, including our design, in the dequantization part of the JPEG standard. Two measures Peak Signal to Noise Ratio (PSNR) and structural similarity (SSIM), are used to compare and inspect the applicability of multipliers. Both PSNR and SSIM are widely used in image processing; they assess the quality of a compressed image. The higher the PSNR, the better the quality of the compressed or reconstructed image. Simulation results in **Table 4** show that using the approximate multipliers does not significantly affect the decompressed image's quality. As expected, our multiplier has the least quality reduction in output image due to its lower MRED.

Table 4: PSNR and SSIM values for decompressed images

Multiplier	PSNR	SSIM
Exact	35.1281	0.9095
LM [15]	30.2662	0.9001
ILM-EA [29]	31.9424	0.8953
ALM-SOA-11 [5]	30.2744	0.8759
Proposed (Exact Adder)	32.9800	0.9037

6. CONCLUSION

In this paper, a new algorithm for logarithmic multiplication is proposed and analysed. The use of approximate adders (SOA) in the final stage of multiplication is also investigated. 16-bit multipliers were implemented using this algorithm, and the simulation results on showed that by using the appropriate correction term, the multiplier accuracy is significantly improved compared to previous similar works, so that MRED has decreased by about 25.6% compared to ILM-EA. analysing change of MRED with respect to truncation width of SOA showed that our design is more robust to adder truncation than previous designs, so that at ALM-SOA-11 and ILM-EA the best truncation width is 11 bits, but MRED in our design does not change much up to 16 bits. Which can be exploited for more hardware savings. Hardware synthesis also show an improvement of 2.12% and 12% in area, and latency respectively and a 13.5% increase in power consumption compared to best results available in the literature. The PDP×MRED criterion also shows that our multiplier shows the best performance among the existing designs by considering both error characteristics and hardware measures. Finally, we implemented our multiplier in JPEG standard, and results showed that our design is

applicable in such error-tolerant applications without notable quality degradation.

REFERENCES

[1] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Computing Surveys (CSUR),* vol. 48, no. May 2016, pp. 1-33, 2016.

[2] V. Sze, Y.-H. Chen, T.-J. Yang and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE,* vol. 105, no. 12, pp. 2295-2329, 2017.

[3] P. Yin, C. Wang, H. Waris, W. Liu, Y. Han and F. Lombardi, "Design and Analysis of Energy-Efficient Dynamic Range Approximate Logarithmic Multipliers for Machine Learning," *IEEE Transactions on Sustainable Computing,* 2020.

[4] R. Pilipović, P. Bulić and U. Lotrič, "A Two-Stage Operand Trimming Approximate Logarithmic Multiplier," *IEEE Transactions on Circuits and Systems I: Regular Papers,* 2021.

[5] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 65, no. 9, pp. 2856-2868, 2018.

[6] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra and G. Di Meo, "Comparison and Extension of Approximate 4-2 Compressors for Low-Power Approximate Multipliers," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 67, no. 9, pp. 3021-3034, 2020.

[7] D. Nandan, J. Kanungo and A. Mahajan, "An efficient VLSI architecture design for logarithmic multiplication by using the improved operand decomposition," *Integration,* vol. 58, pp. 134-141, 2018.

[8] H. Saadat, H. Bokhari and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 37, no. 11, pp. 2623-2635, 2018.

[9] H. Jiang, L. Liu, F. Lombardi and J. Han, "Approximate arithmetic circuits: Design and evaluation," in *Approximate Circuits*, Springer, 2019, pp. 67-98.

[10] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi and J. Han, "A comparative evaluation of approximate multipliers," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2016.

[11] P. Kulkarni, P. Gupta and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," in *2011 24th Internatioal Conference on VLSI Design*, 2011.

[12] S. Hashemi and S. Reda, "Approximate multipliers and dividers using dynamic bit selection," in *Approximate Circuits*, Springer, 2019, pp. 25-44.

[13] 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 Integration (VLSI) Systems,* vol. 23, no. 6, pp. 1180-1184, 2015.

[14] 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.

[15] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers,* vol. 4, pp. 512-517, 1962.

[16] M. S. Ansari, B. F. Cockburn and J. Han, "An Improved Logarithmic Multiplier for Energy-Efficient Neural Computing," *IEEE Transactions on Computers,* vol. 70, no. 4, pp. 614-625, 2021.

[17] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida and N. Bagherzadeh, "Efficient Mitchell's Approximate Log Multipliers for Convolutional Neural Networks," *IEEE Transactions on Computers,* vol. 68, no. 5, pp. 660-675, 2019.

[18] S. Mazahir, M. K. Ayub, O. Hasan and M. Shafique, "Probabilistic error analysis of approximate adders and multipliers," in *Approximate Circuits,* Springer, 2019, pp. 99-120.

[19] J. Ma, K. Man, T. Krilavicius, S. Guan and T. Jeong, "Implementation of high performance multipliers based on approximate compressor design," in *Proc. Int. Conf. Electrical and Control Technol*, 2011.

[20] 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, 2014.

[21] S. Hashemi, R. I. Bahar and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015.

[22] V. Leon, G. Zervakis, D. Soudris and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 26, no. 3, pp. 421-430, 2017.

[23] M. Ha and S. Lee, "Multipliers with approximate 4-2 compressors and error recovery modules," *IEEE Embedded Systems Letters,* vol. 10, no. 1, pp. 6-9, 2017.

[24] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro and N. Petra, "Approximate multipliers based on new approximate compressors," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 65, no. 12, pp. 4169-4182, 2018.

[25] V. Mahalingam and N. Ranganathan, "An efficient and accurate logarithmic multiplier based on operand

decomposition," in *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)*, 2006.

[26] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," *IEEE Transactions on Computers,* vol. 52, no. 11, pp. 1421-1433, 2003.

[27] M. Ito, D. Chinnery and K. Keutzer, "Low power multiplication algorithm for switching activity reduction through operand decomposition," in *Proceedings 21st International Conference on Computer Design*, 2003.

[28] D. J. McLaren, "Improved Mitchell-based logarithmic multiplier for low-power DSP applications," in *IEEE International [Systems-on-Chip] SOC Conference, 2003. Proceedings.*, 2003.

[29] M. S. Ansari, B. F. Cockburn and J. Han, "A Hardware-Efficient Logarithmic Multiplier with Improved Accuracy," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019.

[30] D. Nandan, J. Kanungo and A. Mahajan, "An efficient architecture of leading one detector," *International Journal of Pure and Applied Mathematics,* vol. 118, no. 14, pp. 267-272, 2018.

[31] J. Chen, C.-H. Chang, Y. Wang, J. Zhao and S. Rahardja, "New hardware and power efficient sporadic logarithmic shifters for DSP applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 37, no. 4, pp. 896-900, 2017.

[32] G. K. Wallace, "The JPEG still picture compression standard," *IEEE transactions on consumer electronics,* vol. 38, no. 1, pp. xviii-xxxiv, 1992.

[33] 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 (JETC),* vol. 13, no. 4, pp. 1-34, 2017.

[34] T. Stouraitis and V. Paliouras, "Considering the alternatives in low-power design," *IEEE Circuits and Devices Magazine,* vol. 17, no. 4, pp. 22-29, 2001.

[35] C. Basetas, I. Kouretas and V. Paliouras, "Low-power digital filtering based on the logarithmic number system," in *International Workshop on Power and Timing Modeling, Optimization and Simulation*, Springer, 2007, pp. 546-555.