HEALM: Hardware-Efficient Approximate Logarithmic Multiplier with Reduced Error

Shuyuan Yu*, Maliha Tasnim*, Sheldon X.-D. Tan*
* Department of Electrical and Computer Engineering, University of California, Riverside, CA 92521
syu070@ucr.edu, mtasn004@ucr.edu, stan@ece.ucr.edu

***Abstract*—In this work, we propose a new approximate logarithm multipliers (ALM) based on a novel error compensation scheme. The proposed hardware-efficient ALM, named *HEALM*, first determines the truncation width for mantissa summation in ALM. Then the error compensation or reduction is performed via a lookup table, which stores reduction factors for different regions of input operands. This is in contrast to an existing approach, in which error reduction is performed independently of the width truncation of mantissa summation. As a result, the new design will lead to more accurate result with both reduced area and power. Furthermore, different from existing approaches which will either introduce resource overheads when doing error improvement or lose accuracy when saving area and power, *HEALM* can improve accuracy and resource consumption at the same time. Our study shows that 8-bit HEALM can achieve up to 2.92%, 9.30%, 16.08%, 17.61% improvement in mean error, peak error, area, power consumption respectively over REALM, which is the state of art work with the same number of bits truncated. We also propose a single error coefficient mode named HEALM-TA-S, which improves the ALM design with a truncation adder (TA) for mantissa summation. Furthermore, we evaluate the proposed HEALM design in a discrete cosine transformation (DCT) application. The result shows that with different values of k, HEALM-TA can improve the image quality upon the ALM baseline by 7.8∼17.2dB in average and HEALM-SOA can improve 2.9∼15.8dB in average, respectively. Besides, HEALM-TA and HEALM-SOA outperform all the state of art works with $k = 2, 3, 4$ on the image quality. And the single coefficient mode, HEALM-TA-S, can improve the image quality upon the baseline up to 4.1dB in average with extremely low resource consumption.**

I. Introduction

Approximate computing enables efficient trade-off among accuracy, area, latency and power for more efficient error tolerant applications implementation such as machine learning and multimedia workloads [1]. Those workloads are heavily dominated by the multiplication operations and hence design of hardware-efficient multiplier has been intensively investigated recently. The primary goal of the approximate multiplier design is to reduce the power and area for the least accuracy loss.

A number of approximate multiplier designs have been proposed recently [2]–[11]. Those approximate multipliers employ some ad-hoc truncation or reduction methods or mathematically formulated approximation schemes. Most of the existing methods, however, lack the systematic configurability for accuracy vs. area/power/latency trade-off. On the other hand, a class of approximate multipliers that are mathematically formulated include logarithmic multipliers, which convert multiplication into only shift and addition operations. Due to the inherent approximate nature of logarithmic operation and the easy accuracy manipulation of the resulting addition, the area, latency and power can be traded off at the cost of accuracy. The logarithmic multiplier was originally proposed by Michelle [12]. Since then, many approximate logarithmic multipliers (ALM) have been proposed to improve Michelle's work [9], [10], [13], [14]. Most of those methods focused on how to reduce and compensate the errors introduced in the piece-wise approximation of the log function, which tends to cause negative errors.

Recently Ansari *et al.* [14] developed an approximate scheme to make the error distribution more balanced (double sided errors) for the ALM method. Saadat *et al.* [10] further introduced a general error compensation technique, called *REALM*, using an analytically generated error reduction factor lookup table for different regions of input operands. The benefit of this method is that it can generate more balanced errors by designing and providing configurable design trade-off between area and precision. However, this method uses one lookup table for all truncation configuration in the approximate addition, which may lead to large errors especially for low precision cases as we will show in this work.

Based on the observation, in this work, we propose a new hardware efficient approximate logarithmic multiplier, named *HEALM*, with a novel error reduction scheme for low precision (8-bit to 16-bit) multiplication. The key contributions of this work are listed as follows:

1. HEALM first determines the truncation width for mantissa summation in ALM based on the resource requirement or design constraints. Then the error compensation or reduction is performed via a lookup table, which stores error compensation coefficients for different regions of input operands. This is different from the existing approach like REALM [10], in which error reduction is performed independently of the width truncation of mantissa summation.
2. The error behaviors of HEALM show that HEALM can achieve lower mean error (1.12%) and peak error (4.71%) than the ALM baseline. Unlike existing approaches which will either introduce resource overheads while doing error improvement or sacrifice accuracy when saving area and power, HEALM can improve the error metrics and resource consumption at the same time. 8-bit HEALM-TA with $k = 3$ achieves 2.17% / 9.75% in mean / peak error, and provides 9.38% in area reduction. 8-bit HEALM-SOA with $k = 3$ achieves 1.78% / 7.65% in mean / peak error, and provides 1.42% in area reduction. Compared with REALM, which is the state of art work, HEALM can achieve up to 2.92%, 9.3%, 16.08%, 17.61% improvement in mean error, peak error, area, power consumption respectively with the same number of bits truncated. Furthermore, 16-bit HEALM-TA improves all of the four design metrics (mean error, peak error, area, power) against the ALM baseline by 2.12%, 5.28%, 17.21%, 21.00%; and HEALM-SOA can improve the design metrics by 2.38%, 5.96%, 13.59%, 17.15%.
3. We propose a single error coefficient mode named HEALM-TA-S, which can do error improvement on ALM design with a truncation adder (TA) doing the mantissa summation. 8-bit HEALM-TA-S achieves 4.26% in mean error, 17.12% in peak error, saving 34.8% and 41.9% in area and power consumption and 16-bit HEALM-TA-S achieves 4.87%, 12.02% in mean and peak error, saving 30.59% and 40.09% in area and power consumption when compared to the ALM baseline.
4. We also evaluate the *HEALM* in a *discrete cosine transformation* (DCT) application. The result shows that with different values of k, HEALM-TA can improve the image quality upon the ALM baseline by 7.8∼17.2dB in average and HEALM-SOA can improve 2.9∼15.8dB in average, respectively. Besides, HEALM-TA and HEALM-SOA outperform all the state of art works with $k = 2, 3, 4$ on the image quality. In addition the single coefficient mode, HEALM-TA-S, can improve the image quality upon the baseline up to 4.1dB in average with extremely low resource consumption.

This paper is organized as follows: Section II reviews several recently proposed approximate multiplication designs. Section III presents the proposed *HEALM* design including the inexact adders and the error reduction techniques. Section IV shows the experimental results for the error metrics, area, power and comparison results with state of art methods. Finally, section V concludes the paper.

This work is supported in part by NSF grants under No. CCF-1816361, in part by NSF grant under No. CCF-2007135 and No. OISE-1854276.

II. Review of related work

Recently, various designs of approximate unsigned integer multipliers have been proposed. Earlier designs often involve ad-hoc based approximations, such as recursive multipliers [2] which consist of 2×2 multiplication blocks, simplification of Wallace tree [3], simplifying partial product generation/summation [4]–[6], others used a smaller multiplier by extracting m-bit fragment from the N-bit precision inputs. Such as [7], [8].

Among these methods, many recent approximate multipliers are developed based on the classic approximate logarithmic multiplier proposed by Mitchell, also called *ALM*, as it shows good overall performance and has flexibility for trade-offs among area, power and accuracy [12]. Specifically, for ALM design, the two inputs A and B are first represented by the following format: $2^{k_a} \cdot (1+x)$ and $2^{k_b} \cdot (1+y)$, respectively. Then the multiplication result can be approximated as (1).

$$C_{ALM} = \begin{cases} 2^{k_a+k_b} \cdot (1+x+y), & x+y<1, \\ 2^{k_a+k_b+1} \cdot (x+y), & x+y \geq 1 \end{cases} \tag{1}$$

Here C_{ALM} is the approximate multiplication result. The ALM design requires four steps to finish the multiplication process. First it utilizes leading-one detectors (LOD) to find the leading bit '1' as the integer part; second, barrel shifters are used to re-align the rest of the bits as the fraction part; then it sums the two fraction and integer parts up as k_a+k_b+x+y; and finally it shifts back with the same bits. Although ALM suffers from high absolute MRED (mean relative error distance) and peak relative error of 3.76% and 11.11%, respectively, it can perform a good trade-off among accuracy, area and power.

To further improve the accuracy of the ALM method, several derivative works have been proposed by means of different error compensation mechanisms. For instance, the MBM design tried to add a fixed single error-correction term to the final result [9]. This was further improved by the LeAp multiplier, which added different error coefficients to the fraction parts based on the value ranges of the results [15]. The REALM multiplier design further improved the compensation scheme by using a lookup table to store $M \times M$ coefficients / factors for $M \times M$ partitions of input ranges with some hardware resource overheads [10]. These works indeed improved the error metrics of the approximate logarithmic multiplication without incurring too much resource overheads.

One important observation is that the ALM design will become less effective in reducing area and power when the precision of inputs decreases. Ebrahimi *et al.* [15] recently showed that 32-bit ALM can have more area and power reduction than 16-bit ALM. However, low precision operation is important as emerging machine learning workloads can be performed (at least for inference) using low precision operations. For instance, 16-bit fixed point is demonstrated to be sufficient for training neural networks with no loss in classification accuracy [16]. 8-bit precision is sufficient for inference with minimal accuracy loss [17].

Some previous works [13], [14] tried to do further area reduction by replacing the exact adder with an inexact one. Since the exact adder unit is the bottleneck of the ALM critical path and occupies large area, this idea does help in area saving. But the inexact adder also introduces extra error, and the error can become quite significant especially in the 8-bit case (shown later in Sec. IV). The REALM design [10] did error compensation for ALM and achieved extremely low error bias with very low peak error even under the circumstance that truncates the lower part in the mantissa summation. However, the results are also obtained under 16-bit precision only. We'll show that REALM under 8-bit precision will not perform as well as the 16-bit case in Sec. IV.

In this work, we will focus on the 8-bit and 16-bit precision hardware efficient approximate logarithmic multiplier design and demonstrate the superior performance of the proposed new design against the ALM [12] baseline and other state of art works like LeAp [15], REALM [10], ALM-SOA [13], ILM-EA [14] and ILM-AA [14].

III. Proposed hardware-efficient approximate multiplier

In this section, we show the details of the proposed hardware-efficient ALM design by considering the bit or width truncation in the mantissa summation part and error compensation at the same time.

Consider an inexact adder with N-bit inputs A and B, then the sum S has $N+1$ bits. Let k be the number of bits in the lower part of the sum which are approximated. The binary representation of A is $A_{N-1}A_{N-2}...A_kA_{k-1}...A_0$ and B is in a similar form. The upper part $A_{N-1}A_{N-2}...A_k$ and $B_{N-1}B_{N-2}...B_k$, which are denoted as A_H and B_H, respectively, will perform the exact summation to obtain the higher part of S, $S_NS_{N-1}...S_k$, which is denoted as S_H. The lower part $A_{k-1}...A_0$ and $B_{k-1}...B_0$, which are denoted as A_L and B_L, will perform the approximate summation to obtain the lower part of S: $C_{k-1}S_{k-1}...S_0$. Note that C_{k-1} is the carry bit to the exact summation of upper parts.

We implement two representative approximate adders (or inexact adders), one is the truncation adder (TA), the other is the set one adder (SOA) [13] with error improvement to the ALM. These adders have very small complexity, which are suitable for implementing our *HEALM* designs, named as *HEALM-TA* and *HEALM-SOA* with error improvement. To carry out the error compensation, we first analyze the error profile of ALM when the exact adder used for the mantissa summation is replaced with an approximate adder. Then, we perform specified error compensation for HEALM with each approximate adder under different k values, where k represents the number of bits in the inexact mantissa summation part, to achieve the best trade-off among the error metrics and the hardware resources. The structure of the *HEALM* design is shown in Fig. 1.

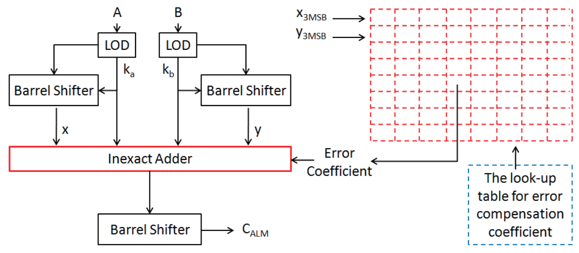
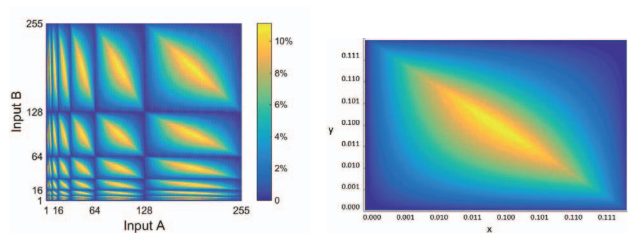


Fig. 1: HEALM design: mantissa summation in ALM design replaced with an error compensated approximate adder.



(a) (b)

Fig. 2: (a) The error behavior of ALM (8×8). (b) The error behavior of the fractional part of ALM in an power-of-two interval.

A. HEALM with truncation adder: HEALM-TA

1) The error behaviors of ALM-TA: Before we discuss our proposed HEALM design with truncation adder, or *HEALM-TA*, we need to show the error behavior of ALM with a simple TA, represented as *ALM-TA*. First, we give a brief introduction on the concept of TA. The TA simply truncates the lower part of the inputs A and B, which makes the inexact part of the mantissa summation S_L equal to zero. Thus the mantissa summation with TA will only calculate the upper part S_H. An N-bit adder is actually truncated to an $N-k$-bit adder. We use a 7-bit case with $k=4$ to describe the concept of TA, which is shown in Fig. 3(a).

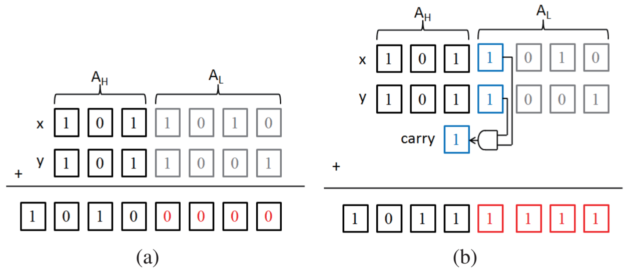


Fig. 3: (a) A 7-bit truncation adder (TA) example. (b) A 7-bit set one adder (SOA) example.

The error behavior of ALM-TA is similar as the error behavior of the ALM method. We have shown the ALM method in (1) in Sec. II, and now the exact multiplication written in log-based form can be expressed as (2).

$$\begin{aligned} C_{Exact} &= 2^{k_a+k_b} \cdot (1+x) \cdot (1+y) \\ &= 2^{k_a+k_b} \cdot (1+x+y+xy) \end{aligned} \quad (2)$$

$$\begin{aligned} Error_{ALM} &= C_{Exact} - C_{ALM} \\ &= \begin{cases} 2^{k_a+k_b} \cdot (xy), & x+y<1, \\ 2^{k_a+k_b+1} \cdot (1-x-y+xy), & x+y \geq 1 \end{cases} \end{aligned} \quad (3)$$

$$\begin{aligned} Error_{ALM-TA} &= C_{Exact} - C_{ALM-TA} \\ &= \begin{cases} 2^{k_a+k_b} \cdot (1+x_H x_L + y_H y_L \\ +x \cdot y - 1 - x_H - y_H), & x+y<1, \\ 2^{k_a+k_b+1} \cdot (1+x_H x_L + y_H y_L \\ +x \cdot y - 2 \cdot x_H - 2 \cdot y_H), & x+y \geq 1 \end{cases} \\ &= \begin{cases} 2^{k_a+k_b} \cdot (x_L + y_L + xy), & x+y<1, \\ 2^{k_a+k_b+1} \cdot (1+x_L+y_L \\ +x \cdot y - x_H - y_H), & x+y \geq 1 \end{cases} \end{aligned} \quad (4)$$

Based on (1) and (2), we can calculate the error of ALM as (3). The error behavior of ALM showing a proportional replication in each power-of-two interval is demonstrated in Fig. 2. Hence, we can perform error compensation to the fractional part before barrel shifting operation to save resource, which is also demonstrated in [15].

After replacing the exact mantissa summation in ALM ("$x+y$") with approximate summation by using TA, the error will also accumulate. We use $Error_{ALM-TA}$ to represent it from time being. The error behavior of ALM-TA can be calculated as (4) and also has a proportional replication in each power-of-two interval. Fig. 4(a) demonstrates the error behavior of ALM-TA with $k=4$ in an interval. Notice that the error behavior though distributes nearly symmetric, which is similar as the error profile of ALM in a single interval. It further shows proportional replication in each $1/8$ interval. Also, for TA summation, no matter what the lower half of the inputs are (here represented as x_L and y_L, as the inputs of the mantissa summation is the fractional part x and y of input A and B, respectively), the approximate summation is only determined by the exact summation part ($x_H + y_H$). Thus, we partition the fractional xy-space into 64 blocks (8×8) with the red dash line, which are the most significant 3 bits (3 MSBs) of x and y, as shown in Fig. 4; and recalculate the average error in each block as shown in Fig. 4(b). Also, for ALM-TA with more than 3 bits in the exact summation part ($k<4$), we still use the 8×8 blocks partition to calculate the average error to save resource. The experimental results shown in Sec. IV will prove that the partition with 8×8 is sufficient to achieve acceptable accuracy improvement and good resource saving.

2) The proposed HEALM-TA error compensation: Based on the aforementioned observation on the error behavior of ALM-TA, we can perform specified error compensation and propose our HEALM idea. We first generate a lookup table, which is of the same size as 8×8 blocks partition, as an error compensation pattern. An example of the pattern is shown in Fig. 4(c). The error coefficient, Err_{coeff}, which is added to the approximate mantissa summation, is generated by searching the lookup table based on the 3 MSBs of x and y to perform the specified error compensation. Note that when the error compensation pattern is simple (usually the value of k is large), such as the example we show in Fig. 4(c), the lookup table can be simplified to several large squarish area. Like the case shown in Fig. 4(c), the blue area is equivalent to the sum of 3 rectangular regions, which can be described much simpler than an 8×8 lookup table, thus saving the resource consumption.

$$C_{HEALM-TA} = \begin{cases} 2^{k_a+k_b} \cdot (1+x_H+y_H \\ +Err_{coeff}), & x+y<1, \\ 2^{k_a+k_b+1} \cdot (x_H+y_H \\ +Err_{coeff}), & x+y \geq 1 \end{cases} \quad (5)$$

The HEALM-TA method can be expressed as (5), where $C_{HEALM-TA}$ is the product of HEALM-TA method. And the value of the error coefficients are determined by the average error of each block. We notice that if $x+y \geq 1$, the error coefficient Err_{coeff} will be added twice. Thus, the equivalent error in these blocks where $x+y \geq 1$ should be as half as its initial value. So we divided the Err_{coeff} for these blocks to half. And for those blocks where $x+y$ could be either smaller or larger than 1, we further perform error compensation arrangement to achieve the possible smallest peak relative error. Note that the mantissa summation of $x+y$ is replaced with the approximate summation now, so we need to do quatization of the error coefficients to ensure that the precision of these coefficients no larger than the precision of the exact summation part. In the case of $k=4$ as shown in Fig. 4(c), the exact summation only has 3 bits. So Err_{coeff} also need to be a 3-bit parameter. Actually in this case, the error coefficient will either be 1/8 or 2/8 as shown in Fig. 4(c). Our proposed HEALM-TA design performs well especially when the value of k is large. And 8-bit HEALM-TA with $k=3$ can improve the traditional ALM design in both error metrics and area, which is never achieved by the previous works with 8-bit precision. We'll prove this later in Sec. IV.

3) Single coefficient mode: HEALM-TA-S: Furthermore, we propose a single coefficient mode, named as *HEALM-TA-S* to perform error compensation on ALM-TA with almost no resource overheads. As an N-bit simple TA with k bits truncated, it consists of 1 HA (half adder) and $N-k-1$ FAs (full adder), which is shown in Fig. 6(a) (in the example of $k=4$, the exact summation part includes 2 FA and 1 HA). To perform the simplest error compensation, the error coefficient for the whole fractional space is set to be the same value, which is $2^{-(N-k)}$ (1/8 in this case); and the HA is replaced with an FA at the LSB (least significant bit) location to obtain the smallest resource overheads. The structure of the mantissa summation part of HEALM-TA-S design is shown in Fig. 6(b). Note that the input carry bit (C_{in}) for the FA at LSB is always set to '1' according to the error coefficient. We'll prove later in Sec. IV that HEALM-TA-S can perform a good trade-off among the error metrics and the hardware performance especially when k is large for HEALM-TA-S design.

B. HEALM with set one adder: HEALM-SOA

Besides HEALM-TA, we also propose another HEALM design with set one adder, or *SOA*, called *HEALM-SOA*. Simple ALM design with mantissa summation replaced with SOA (ALM-SOA) has already been proposed before [13]. Based on ALM-SOA, we further perform error compensation similar to Sec. III-A.

In an SOA, different from TA, all the bits in S_L part are set to logic '1' to produce a balanced error in the ALM-SOA method. For the S_H, which is the exact summation part, $S_H = A_H + B_H + C_{in}$, where the carry bit C_{in} is obtained by doing an *AND* operation of the MSB in A_H and B_H (A[k-1] and B[k-1], respectively), as expressed in (6), suppose the SOA is an N-bit summation with k bits in the approximate summation part S_L.

$$\begin{aligned} S_H = S[N:k] &= A[N-1:k] + B[N-1:k] + C_{in} \\ C_{in} &= A[k-1]B[k-1] \\ S_L[i] &= 1, i \in [0, k-1] \end{aligned} \quad (6)$$

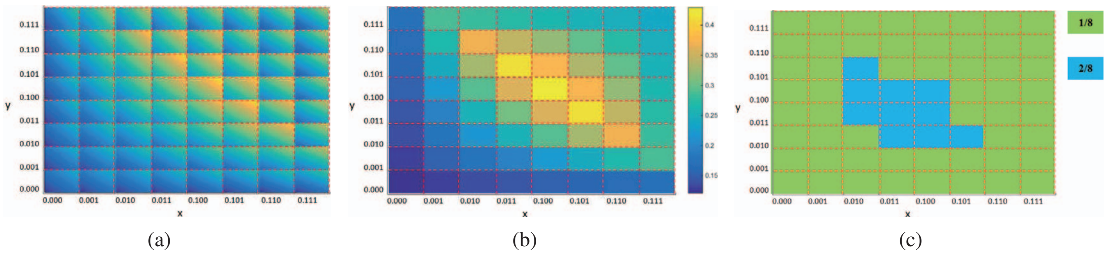


Fig. 4: (a) The error behavior of ALM-TA with $k = 4$. (b) The error behavior of ALM-TA with $k = 4$ doing average operation in each block. (c) The error compensation pattern for HEALM-TA with $k = 4$.

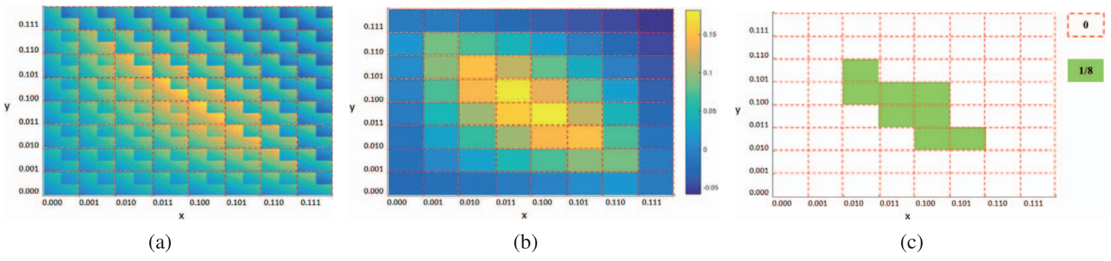


Fig. 5: (a) The error behavior of ALM-SOA with k=4. (b) The error behavior of ALM-SOA doing average operation in each block. (c) The error compensation pattern for HEALM-SOA with k=4.

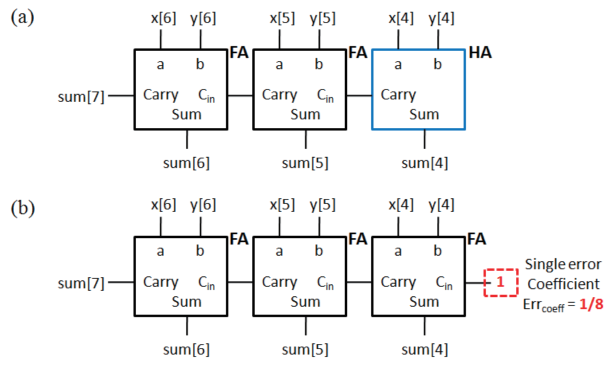


Fig. 6: The mantissa summation unit of HEALM-TA-S design with $k = 4$: (a) The exact summation part with no error compensation. (b) The exact summation part improved with a single error coefficient, replacing the HA with an FA.

Then, based on (1) and (6), we can calculate the error of ALM-SOA in an power-of-2 interval as (7), where $C_{in} = x[k-1]y[k-1]$, and $S_{SOA} = \Sigma 2^i/2^N$, $i \in \{0, 1, ..., k-1\}$. Similar as ALM-TA, we show the error behavior of ALM-SOA ($k = 4$) with an example in Fig. 5(a).

$$\begin{aligned} Error_{ALM-SOA} &= C_{Exact} - C_{ALM-SOA} \\ &= \begin{cases} 2^{k_a+k_b} \cdot (1 + x_H x_L + y_H y_L + x \cdot y \\ -1 - x_H - y_H - C_{in} - S_{SOA}), & x + y < 1, \\ 2^{k_a+k_b+1} \cdot (1 + x_H x_L + y_H y_L + x \cdot y \\ -2 \cdot (x_H + y_H + C_{in} + S_{SOA})), & x + y \geq 1 \end{cases} \\ &= \begin{cases} 2^{k_a+k_b} \cdot (x_L + y_L + x \cdot y - C_{in} - S_{SOA}), & x + y < 1, \\ 2^{k_a+k_b} \cdot (1 + x_L + y_L + x \cdot y \\ -x_H - y_H - C_{in} - S_{SOA}), & x + y \geq 1 \end{cases} \end{aligned} \tag{7}$$

The HEALM-SOA idea is similar as HEALM-TA. The error compensation pattern of HEALM-SOA is shown in Fig. 5(c). We partition the fractional space into 8×8 blocks and calculate the average error following the same way as HEALM-TA, which is shown in Fig. 5(b). Then based on the error distribution of ALM-SOA, we generate a specified error compensation pattern in a lookup table form. The error coefficient which is added to the mantissa summation part is determined by the 3 MSBs of x and y as HEALM-TA. Similar to HEALM-TA, HEALM-SOA also selects the error compensation patterns to achieve the smallest possible peak relative error and can provide improvement upon the traditional ALM design in terms of both the error metrics and resource consumption. We'll show this later in Sec. IV.

Note that unlike HEALM-TA, the LSB summation of the exact summation part in HEALM-SOA should consider the carry bit C_{in} from the S_L (approximate summation part). The LSB summation also requires a FA instead of HA in the exact summation part of HEALM-SOA. We cannot directly add a bit '1' as error compensation to the LSB location. So HEALM-SOA will not have a single error coefficient mode like "HEALM-SOA-S".

IV. Experimental results and discussions

In this section, we evaluate the performance of the proposed *hardware-efficient approximate logarithmic multiplier with reduced error*, named HEALM under 8-bit precision. We also compare HEALM against the ALM (approximate logarithmic multiplier) baseline [12] and other state of art works with the same precision. Furthermore, we demonstrate 16-bit HEALM design results compared with the baseline and state of art works as a complementary.

A. Experimental setup

To evaluate the performance of the proposed HEALM design, we first compare the error metrics and the hardware performance of HEALM with its original version: a classical ALM proposed by Mitchell, which is selected as the baseline. We also compare HEALM with other state of art improved ALMs. These improved ALMs include: LeAp [15], REALM [10], ALM-SOA [13], ILM-EA [14], ILM-AA [14]. For REALM design, we compare REALM8 which did the same partition in the fractional space (in an power-of-2 interval) as HEALM does for fair comparison.

All the above mentioned 8-bit multipliers are implemented in Verilog HDL and synthesized with Synopsys Design Compiler using EDK 32nm standard cell library [18] as single-cycle designs, and at the same timing constraints of 2.5ns (400 MHz working frequency) for area and power consumption comparison. For 16-bit multipliers, we implemented with the same library but at the timing constraints of 5ns (200MHz).

For the error metrics evaluation, we developed behavioral simulation models for all the multipliers listed in Table I in MATLAB and measured the accuracy using 1 million random inputs uniformly distributed over the set $\{0, 1, ..., (2^8 - 1)\}$. The errors are reported with respect to the exact results. The error metrics used to report the error behavior include: mean error (mean of absolute relative error, also referred as MRED in some previous works [14]); and peak error (maximum value of the absolute relative error). All the error metrics are in percentages.

TABLE I
ERROR METRICS AND HARDWARE PERFORMANCE COMPARISON FOR THE 8-BIT MULTIPLIERS.

Logarithmic Multiplier Design	k	Mean Error /%	Peak Error /%	Area /μm^2	Power /μW
HEALM-TA (proposed)	1	1.12	4.86	1216.84	114.50
	2	1.40	8.25	938.05	92.17
	3	2.17	9.75	743.63	74.14
	4	3.66	13.77	595.46	51.41
HEALM-TA-S (proposed)	1	3.21	11.11	763.70	71.19
	2	3.17	12.02	716.69	65.68
	3	3.39	13.79	614.01	56.62
	4	4.26	17.12	534.72	42.32
HEALM-SOA (proposed)	1	1.13	4.71	1175.41	113.96
	2	1.38	5.90	966.76	90.32
	3	1.78	7.65	808.94	75.05
	4	3.12	12.17	664.33	59.55
ALM with/without Approximate Adder					
ALM [12]	0	3.76	11.11	820.63	72.83
ALM-TA	1	4.02	12.03	763.45	69.41
	2	4.79	13.83	702.71	65.07
	3	6.58	17.25	612.74	56.20
	4	10.29	23.53	533.19	41.73
ALM-SOA [13]	1	3.50	11.11	776.92	72.11
	2	3.23	11.46	736.26	68.54
	3	3.07	12.36	702.96	64.46
	4	3.47	14.13	596.98	50.63
Other Improved Logarithmic Multipliers from the Literature					
LeAp [15]	0	1.38	4.71	1040.21	106.43
REALM [10]	0	0.90	3.96	1235.14	124.75
	1	1.06	4.76	1128.40	108.48
	2	1.81	8.27	993.96	96.46
	3	3.25	12.80	908.82	82.56
	4	6.58	23.07	709.57	62.40
ILM-EA [14]	0	2.84	11.11	1221.92	107.89
ILM-AA [14]	1	2.91	12.25	1186.85	104.72
	2	3.09	14.24	1046.06	95.03
	3	3.64	17.35	985.57	87.74
	4	5.47	23.47	887.47	77.34

B. Performance evaluation

The error metrics and the hardware performance for the implemented multipliers are shown in Table I. Since the proposed HEALM designs utilize the inexact adder in the mantissa summation, we use a parameter k in Table I to represent the number of bits in the inexact summation part. For example, in the demonstrated cases shown in Fig. 3(a) and Fig. 3(b), k equals to 4. For ALM, LeAp, and ILM-EA designs, as these multipliers do not have an inexact summation unit, k equals to 0. For the REALM design, the value of the error configuration parameter mentioned in the work [10] is equivalent to the number of bits in the inexact summation part, which is represented by k in our work. To avoid ambiguity, we use the same notation as HEALM design does for easy comparison.

Table I demonstrates that with the same value of k, the proposed HEALM-TA design improves the error metrics upon the ALM-TA design, reducing up to 6.63%, 9.76%, in mean and peak error, respectively; HEALM-SOA improves the error metrics upon the ALM-SOA design, reducing up to 2.37%, 6.40%, in mean and peak error, respectively. When compared with the ALM baseline, HEALM-TA and HEALM-SOA can improve the mean / peak error by 2.64% / 6.25%, and 2.63% / 6.40%, respectively. When compared with REALM, which is the state of art work, the HEALM designs can improve mean error, peak error, area and power consumption by up to 2.92%, 9.3%, 16.08%, 17.61% respectively with the same value of k. The smallest mean error that HEALM-TA and HEALM-SOA can achieve are 1.12% and 1.13%, respectively. And the smallest peak error that HEALM-TA and HEALM-SOA can obtain are 4.86% and 4.71%, respectively. The 16-bit multiplication results are summarized in Table II. Due to limited space, we simply show the results of $k = 9$ and compare with several state of art works. 16-bit HEALM-TA can improve all of the four design metrics (mean error, peak error, area, power) against the ALM baseline by 2.12%, 5.28%, 17.21%, 21.00%; and 16-bit HEALM-SOA can improve the design metrics by 2.38%, 5.96%, 13.59%, 17.15%.

TABLE II
ERROR METRICS AND HARDWARE PERFORMANCE COMPARISON FOR THE 16-BIT MULTIPLIERS.

Approach	k	Mean Error /%	Peak Error /%	Area /μm^2	Power /μW
ALM [12]	0	3.76	11.11	1825.52	110.91
REALM [10]	0	0.75	3.70	2383.36	164.50
	9	1.06	5.27	1572.90	94.07
LeAp [15]	0	0.98	4.76	1990.71	128.20
ALM-TA	9	4.88	12.93	1263.86	66.26
HEALM-TA-S	**9**	**4.87**	**12.02**	**1267.16**	**66.45**
HEALM-TA	**9**	**1.64**	**5.83**	**1511.39**	**87.62**
ALM-SOA [13]	9	3.07	12.03	1383.56	74.10
HEALM-SOA	**9**	**1.38**	**5.15**	**1577.47**	**91.89**

Considering the trade-off among error metrics improvement and resource consumption, the previous works can improve either error metrics or resource consumption (area, power) aspect, but can hardly improve both of these aspects especially when the precision is small (like 8-bit precision) as shown in Table I. To better illustrate this, we show the relationship between the mean error / peak error and area / power for all the listed multipliers in Fig. 7. The rectangular area with the red dash border line in all four sub figures represents that a design outperforms the classical ALM design both in error metrics and resource consumption aspects. Notice in Fig. 7(a), only the proposed HEALM-TA and HEALM-SOA with $k = 3$ improve both the peak error and area aspects, decreasing the peak error with 1.36% and 3.46%, respectively. In Fig. 7(c), though some previous works like ALM-SOA outperforms ALM in both mean error and area aspects, only a little improvement in mean error (at most 0.69%) was obtained. In contrast, the proposed HEALM-TA and HEALM-SOA with $k = 3$ reduce the mean error by 1.59% and 1.98% when compared to ALM, respectively; and provides 9.38% and 1.42% in area reduction at the same time. Besides, HEALM-TA and HEALM-SOA design with $k = 4$ can reduce the mean error by 0.10% and 0.64% when compared to ALM and reduce power by 29.41% and 18.23% at the same time.

The results of HEALM-TA-S (single error coefficient mode) design in Table I shows that HEALM-TA-S can do better trade-offs between accuracy and resource consumption especially when the value of k is large. In case of $k = 4$, which is the largest value of k, HEALM-TA-S decreases the mean / peak error by up to 6.03% / 6.11%, respectively when compared to ALM-TA. Note that HEALM-TA-S achieves this improvement with almost no resource overheads. It also saves 34.84% / 41.89% of area / power with 8-bit inputs, respectively; and 30.59% / 40.09% with 16-bit inputs when compared to the ALM baseline.

C. An image processing application evaluation

Now, we show how the proposed *HEALM* designs compare to state of art methods in an multimedia application. Discrete cosine transformation (DCT) is a commonly used lossy image compression method. The quality of the compressed images is usually evaluated using metrics such as PSNR (peak signal noise ratio) and higher PSNR value represents better image quality. We implement the proposed HEALM design with 8-bit precision in the DCT-iDCT (inverse DCT) workloads, and compare with other logarithmic multipliers on five example images. To be fair, the mantissa summation parts of all the compared logarithmic multipliers are inexact unit, except for the ALM, which is chosen as the baseline. We show the results of image compression in Table III. The result shows that with different values of k, HEALM-TA can improve the image quality upon the ALM baseline by from 7.8$\sim$17.2dB in average and HEALM-SOA can improve 2.9$\sim$15.8dB in average, respectively. Besides, HEALM-TA and HEALM-SOA design outperform all the other state of art works when $k = 2, 3, 4$ by at least 6.3dB, 6.3dB, 8.8dB, respectively. Note that the single coefficient mode design HEALM-TA-S performs the best when $k = 4$, making improvement upon the ALM baseline by 4.1dB in average with extremely low resource consumption as mentioned before. This is due to the error behavior of ALM, whose outputs are always smaller than the exact product. And HEALM-TA-S with $k = 4$ will have a more balanced error than the cases of $k = 1, 2, 3$.

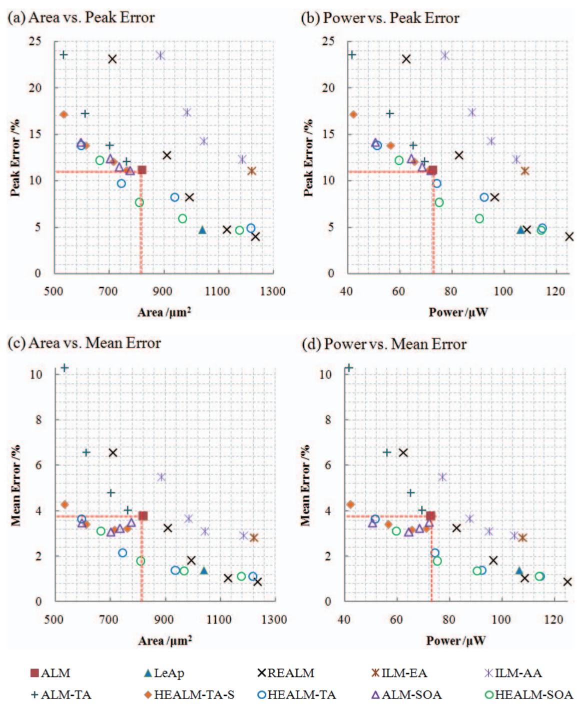


Fig. 7: Comparison of HEALM with baseline and state of art works.

V. Conclusion

In this work, we have proposed a novel hardware-efficient approximate logarithmic multiplier, called *HEALM*. The proposed design, first determined the truncation width for mantissa summation in ALM. Then the error reduction is performed via a lookup table for multiple partitioned input ranges. Numerical results showed that *HEALM* and its enhanced designs could lead to more accurate results with reduced area and power at the same time than the existing ALM baseline design. It also outperformed the state of art design, REALM, with up to 2.92%, 9.30%, 16.08%, 17.61% improvement in mean error, peak error, area, power consumption for 8-bit precision. For discrete cosine transformation (DCT) application, with different values of k, HEALM-TA could improve the image quality upon the ALM baseline by 7.8∼17.2dB in average and HEALM-SOA could improve 2.9∼15.8dB in average, respectively. Besides, HEALM-TA and HEALM-SOA outperformed all the state of art works with $k = 2, 3, 4$ on the image quality.

References

[1] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2015.

[2] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*, pp. 346–351, IEEE, 2011.

[3] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power-and area-efficient approximate wallace tree multiplier for error-resilient systems," in *Fifteenth International Symposium on Quality Electronic Design*, pp. 263–269, IEEE, 2014.

[4] B. S. Prabakaran, S. Rehman, M. A. Hanif, S. Ullah, G. Mazaheri, A. Kumar, and M. Shafique, "Demas: An efficient design methodology for building approximate adders for fpga-based systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 917–920, IEEE, 2018.

[5] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar, "Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators," in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[6] S. Ullah, S. S. Murthy, and A. Kumar, "Smapproxlib: library of fpga-based approximate multipliers," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[7] 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)*, pp. 418–425, IEEE, 2015.

TABLE III
PSNR (dB) for images after DCT-iDCT using logarithmic multipliers.

Approach	Lena	Boat	Barbara	House	Pepper
ALM (baseline)	19.1	18.7	19.3	18.4	18.7
k = 1					
ILM-AA	27.9	26.6	28.1	24.8	27.4
ALM-TA	18.9	18.6	19.2	18.2	18.5
ALM-SOA	19.2	18.8	19.5	18.6	18.9
REALM	37.2	36.5	36.9	38.4	36.4
HEALM-TA	36.1	35.7	36.2	36.5	35.6
HEALM-TA-S	20.4	19.9	20.6	19.9	20.1
HEALM-SOA	34.3	33.9	34.5	36.2	34.4
k = 2					
ILM-AA	27.6	26.2	27.7	24.3	27.1
ALM-TA	17.4	17.1	17.6	16.5	17.0
ALM-SOA	19.8	19.4	20.1	19.4	19.5
REALM	27.1	26.8	27.1	27.9	27.2
HEALM-TA	33.2	32.7	33.4	35.1	33.3
HEALM-TA-S	19.9	19.5	20.1	19.5	19.6
HEALM-SOA	31.7	31.4	32.3	33.1	31.7
k = 3					
ILM-AA	24.5	23.1	24.5	22.4	23.9
ALM-TA	15.0	14.8	15.2	14.3	14.7
ALM-SOA	19.9	19.5	20.1	19.5	19.6
REALM	21.2	21.2	21.4	19.6	20.8
HEALM-TA	26.5	25.6	26.2	28.1	26.4
HEALM-TA-S	18.8	18.4	18.9	18.7	18.3
HEALM-SOA	29.8	29.5	30.0	31.1	29.7
k = 4					
ILM-AA	20.2	18.9	20.1	18.5	19.5
ALM-TA	14.1	13.7	14.1	13.4	13.7
ALM-SOA	18.9	18.9	19.6	16.7	18.9
REALM	19.8	20.0	19.8	17.8	19.5
HEALM-TA	29.1	28.7	29.1	25.9	28.5
HEALM-TA-S	23.1	22.0	22.6	24.3	22.3
HEALM-SOA	22.2	22.1	22.6	19.4	22.4

[8] 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, 2014.

[9] 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.

[10] H. Saadat, H. Javaid, A. Ignjatovic, and S. Parameswaran, "Realm: reduced-error approximate log-based integer multiplier," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1366–1371, IEEE, 2020.

[11] S. Yu, Y. Liu, and S. X.-D. Tan, "COSAIM: Counter-based stochastic-behaving approximate integer multiplier for deep neural networks," in *Proc. Design Automation Conf. (DAC)*, pp. 1–6, Dec. 2021.

[12] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[13] 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.

[14] 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)*, pp. 928–931, IEEE, 2019.

[15] Z. Ebrahimi, S. Ullah, and A. Kumar, "Leap: Leading-one detection-based softcore approximate multipliers with tunable accuracy," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 605–610, IEEE, 2020.

[16] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, pp. 1737–1746, 2015.

[17] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[18] R. Goldman, K. Bartleson, T. Wood, K. Kranen, V. Melikyan, and E. Babayan, "32/28nm educational design kit: Capabilities, deployment and future," in *2013 IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, pp. 284–288, IEEE, 2013.