Minimally Biased Multipliers for Approximate Integer and Floating-Point Multiplication

Hassaan Saadat, Haseeb Bokhari, and Sri Parameswaran, *Senior Member, IEEE*

***Abstract*—Approximate multipliers enable the saving of area and power for implementation of many modern error-resilient compute-intensive applications. In this paper, we first propose a novel error-configurable minimally biased approximate integer multiplier (MBM) design. The proposed MBM design is devised by coupling a unique error-reduction mechanism with an approximate log based integer multiplier. Next, we propose an optimization (by removing leading-one detection and barrel shifting logic) of the MBM and a class of state-of-the-art approximate integer multipliers (DRUM and SSM), so that they can be efficiently used in approximate floating-point (FP) multipliers. Then, we propose a set of new approximate FP multipliers and we show that these FP multipliers lie on the Pareto front on the design spaces of area *versus* error and power *versus* error. We synthesize the designs using the TSMC 45-nm standard-cell library. Results show that the MBM integer design offers optimal points in the design space, offering up to 75% area reduction and 84% power reduction with <0.1% error bias, when compared with the accurate version. The proposed approximate FP multipliers offer better error-efficiency tradeoffs than traditional precision scaling. The FP design space can offer up to 57× power and 28× area improvement for less than 25% peak error, 7% mean error, and 4% error bias, when compared with the IEEE-754 single-precision FP multiplier. We also perform application-level evaluations of the proposed approximate integer and FP multipliers, showing that our proposed multipliers enable significant power and area reduction with minimal degradation in applications' output quality.**

***Index Terms*—Approximate multipliers, floating-point (FP), hardware approximate computing (AC), integer.**

I. Introduction

APPROXIMATE computing (AC) aims to reduce the gap between computational requirements and available processing resources [3], [4]. Researchers have shown that some highly compute-intensive applications (such as multimedia processing and machine learning) are error-resilient [5], [6]. These applications tolerate computational errors with minimal degradation in the output quality. Controlled errors are allowed in the implementation of these applications to achieve simpler and efficient systems with minimal degradation in the final output quality [3], [4]. These error-resilient applications mainly consist of a few dominant kernels [7], which rely heavily on multiplication. Multipliers, therefore, are natural candidates of approximation to gain computing efficiency in the overall system. For the approximation of a multiplier, its construction is modified to achieve a simpler and efficient hardware design at the tradeoff of allowing some inaccuracies [8].

Different applications allow different levels of approximations for producing acceptable quality results [4]. Therefore, a large and dense design space of approximate multipliers is desirable, which can offer a variety of error *versus* resource-efficiency configurations and tradeoffs [9]. Hardware multipliers can be of two types: one, integer multiplier; and the other, floating-point (FP) multiplier. In this paper, we propose: 1) a new error-configurable minimally biased approximate unsigned integer multiplier (MBM); and, 2) a number of approximate FP multipliers (using the MBM multiplier), which constitute a wide and dense design space.

The proposed MBM integer multiplier is designed to have minimal error bias. Minimal error bias prevents accumulation of errors when successive multiplications are performed in an application, and it is desirable in approximate unsigned integer multipliers [1]. We show that our proposed error-configurable MBM multiplier provides optimal design points in the power *versus* error and area *versus* error design spaces of state-of-the-art approximate integer multipliers.

FP multipliers have received relatively less attention for approximation than their integer counterparts, even though they are more resource-hungry. FP multipliers are desirable in computing systems due to the following advantages.

1) They offer *large dynamic range* for handling real numbers.
2) Large and small numbers can be represented with *bounded relative precision*, as opposed to fixed-point.
3) FP arithmetic makes it *easier to program* than fixed-point.
4) In retunable applications, such as adaptive filters and neural networks, FP arithmetic-based systems allow simple updating of coefficients without the need to reanalyze the dynamic ranges and reconfigure the design, as opposed to the fixed-point arithmetic-based systems, thus, favoring *design reusability*.
5) The flexibility and simplicity offered by FP arithmetic in the design-cycle results in *faster design turn-around time*.

Manuscript received April 3, 2018; revised June 8, 2018; accepted July 2, 2018. Date of current version October 18, 2018. This paper was presented in the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) 2018 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Hassaan Saadat.)*

H. Saadat and S. Parameswaran are with the School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia (e-mail: h.saadat@unsw.edu.au; sri.parameswaran@unsw.edu.au).

H. Bokhari was with the Software Engineering Group, Open Access Pty. Ltd., Sydney, NSW 2113, Australia.

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

Digital Object Identifier 10.1109/TCAD.2018.2857262

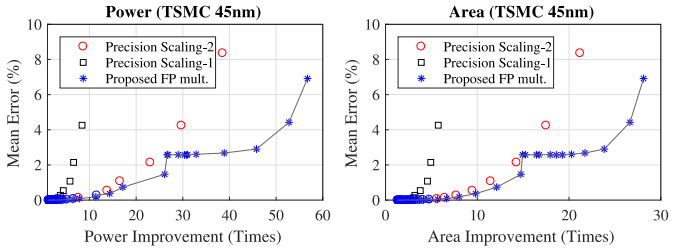

Fig. 1. Proposed approximate FP multipliers *versus* traditional precision scaling. The proposed multipliers offer better tradeoffs.

Hence, it is beneficial to design resource-efficient FP multipliers which may be relatively less precise but can still offer the advantages discussed above [10]. We use an optimized version of the proposed MBM (and two other state-of-the-art approximate integer multipliers) as the mantissa multiplier in the FP multiplier unit, to generate a large and dense design space of approximate FP multipliers. As shown in Fig. 1, our proposed approximate FP multipliers offer better tradeoffs than the traditional precision scaling methods in FP arithmetic. The FP multiplier design space can offer more than 50× power improvement and more than 25× area improvement with respect to IEEE-754 single-precision FP multiplier with a mean error of <5%. Finally, we also evaluate the application-level performance of the proposed integer MBM and approximate FP multipliers using applications from the domain of multimedia processing and machine learning.

A. Contributions of this Paper

The key contributions of this paper can be outlined as follows.

1) We present, for the first time, a new minimally-biased approximate unsigned integer multiplier: the MBM. The MBM design is error-configurable. The MBM multiplier is designed by coupling a novel error-reduction mechanism with the simplest linearly-approximated log based multiplier, known as Mitchell's multiplier (MA) [11]. The proposed error-reduction scheme is based on an analytical framework.
2) For the first time, we show that the new MBM (and a class of existing approximate integer multipliers, such as DRUM [1] and SSM [2]) can be optimized specifically for use in FP arithmetic, as the mantissa multiplier.
3) Next, using the optimized MBM and other approximate integer multipliers in a configurable manner, we design a new set of resource-efficient approximate FP multipliers. We show that our approximate FP multipliers provide better tradeoffs than the traditional precision scaling methods.

B. Paper Organization

Section II summarizes the existing works on approximate integer and FP multipliers. The proposed MBM multiplier design is discussed in Section III. Its error and design metrics, along with a comparison with state-of-the-art approximate multipliers, are presented in Section IV. Section V presents the method for designing the proposed approximate FP multipliers. The design metrics and error results for the proposed FP multipliers are discussed in Section VI. The proposed integer and FP multipliers are evaluated for application-level performance in Section VII. Finally, concluding remarks are presented in Section VIII.

II. Related Work

A. Approximation of Integer Multipliers

The approximate integer multiplier designs in the literature consider single-cycle parallel multipliers as base designs for approximation. UDM [8] and UDMB [9] consist of approximate 2×2 multiplier blocks which are used to build larger multipliers. AWTM [12] and AMPER [13] multipliers involve approximation in the partial product accumulation.

Leading-One Detection Based Approximate Multipliers: These multipliers involve detection of the position of the leading-one in the operands. In ETM [14], the higher bits of the input operands are sent to a multiplication part, and the lower bits are sent to a nonmultiplication part. When there is no 1 in the higher bits of both inputs, the lower bits are sent to the multiplication part. In SSM [2], an m-bit segment is extracted from each n-bit input operand. Depending upon the position of the leading-one in an operand, the extracted m-bit segment may start at one of the two fixed positions in the operand—n^{th} bit or $(n-m)^{\text{th}}$ bit from the right—such that the leading-one is included in the extracted segment. The remaining bits are truncated. The extracted segments are passed to a smaller $m\times m$-sized multiplier. Its output is then scaled, depending upon the total number of least-significant bits (LSBs) truncated from both input operands, to obtain the approximate product. The ESSM [2] version allows three starting positions when $m = (n/2)$. The DRUM design [1] is similar to the SSM design; a k-bit segment is extracted from each n-bit input operand and a smaller $k\times k$-sized multiplier is used. However, it differs from SSM in the sense that the extracted k-bit segment starts at the exact position of the leading-one in the input operand. Moreover, the LSB of each of the extracted k-bit segment is set to 1.

Approximate Log Based Integer Multipliers: The logarithm operation simplifies a multiplication to an addition. However, it requires computation of log and antilog. Several schemes for hardware implementation of log and antilog computation with varying accuracies and hardware complexities exist in the literature [15]. Among these, the linear approximation based techniques (Mitchell's method and its variants) are the most resource-efficient [15], and thus can be used to design resource-efficient approximate multipliers.

Mitchell proposed an efficient approximate multiplier (MA) [11] that uses the log multiplication property. Mitchell proposed to compute approximate binary log and antilog by linearly approximating the log-antilog curves between each power-of-two-interval. In MA, first the characteristic part of the approximate log value of each operand is obtained by determining the position of the leading-one. The rest of the bits in the operand are appended to the characteristic as the fractional part of the approximate log value. The approximate

log values of the two operands are then added. To this sum, the reverse of this log computation procedure is applied to obtain the approximate product, i.e., 1 is appended to the fractional part and scaled according to the characteristic part.

Existing Error-Reduction Schemes for Mitchell's Method: Mitchell's multiplication method suffers from a high and biased error (11.1% peak and 3.8% mean error). Mitchell himself proposed a scheme for reducing the multiplication error. However, it nearly doubles the required hardware. The other linear approximation based methods for log and antilog [15]–[19] are derived from Mitchell's method. They divide each power-of-two-interval into two or more regions and use piecewise approximations for each region. Implementation wise, they involve the use of multiplexers and addition of error-reduction terms, consequently increasing the hardware complexity.

These methods target error reduction in log computation and antilog computation individually, which consequently results in a reduction in multiplication error. Thus, these methods essentially need the addition of three error-reduction terms and three sets of associated multiplexers for performing multiplication: two for computing log of the two operands, and one for antilog of the sum. In contrast, our proposed MBM multiplier design consists of a single error-reduction term that results in minimal error bias as well as smaller mean and peak absolute relative error when compared with Mitchell's multiplier.

B. Approximation of Floating-Point Multipliers

In the IEEE-754 standard representation, an FP number consists of three parts: a sign bit (s), an exponent (e), and a mantissa (m). The IEEE-754 standard FP format requires the mantissa to be normalized, therefore, the most significant bit is always 1. This leading-one is not stored and called as the (*hidden bit*). The value represented by an FP number can be calculated as: value $= (-1)^s \times (1+m) \times 2^{e-\text{bias}}$. The algorithm for FP multiplication mainly consists of four components: 1) mantissa multiplication; 2) exponent addition; 3) rounding; and, 4) normalization. The hidden bit is appended to the mantissa before multiplication. Additionally, some logic is required for handling special quantities and exceptions.

Relatively little attention has been paid to the optimization of FP units in the context of approximate computing. The relevant research for the approximation of FP multipliers is presented in [10], [20], and [21]. Traditionally, precision scaling methods have been used to reduce the complexity. Precision scaling (double-input truncation) means that some bits from the mantissa are truncated.

Tong *et al.* [10] optimized power consumption of an FP multiplier. They implemented the mantissa multiplier as a multicycle digit-serial architecture. This is equivalent to single-input bit-truncation. They reported up to 66% reduction (3× improvement) in multiplication energy per operation. The scheme in [21] targets the mantissa multiplication by approximating it as: $(1+m_a)(1+m_b) = 1+m_a+m_b+m_am_b \approx 1+m_a+m_b$. A power reduction of 25× was reported with a nonconfigurable 25% peak error. The closest inspiration for our research is found in [20], which used Mitchell's multiplier for mantissa multiplication. A double-input bit-truncation method was then applied to achieve further gains in power efficiency. However, neither an optimization nor an error-reduction was applied to the Mitchell based mantissa multiplier. The power-accuracy design space in [20] is also sparse.

In this paper, we generate a wide and dense design space of FP multipliers. For this, we apply novel FP specific optimizations to our proposed MBM multiplier and (two other state-of-the-art approximate integer multipliers) and use them as mantissa multipliers in a configurable manner.

C. Approximate Computing Versus Other Computing Efficiency Techniques

The design techniques such as dynamic voltage and frequency scaling and near-threshold computing (NTC) target energy/power efficiency of digital circuits at the cost of speed (or area) efficiency [22], while aiming to be error-free [23]. Approximate computing, on the other hand, aims for computing efficiency by allowing controlled errors in the implementation of error-resilient applications [4]. Approximate computing can be considered orthogonal to NTC, and the philosophy of approximate computing has been used to address the problem of performance-loss in NTC [24].

III. Minimally Biased Multiplier

The proposed approximate integer multiplier (MBM) is built by coupling a novel resource-efficient error-reduction mechanism with the simplest linearly-approximated log based multiplier, known as Mitchell's multiplier. Therefore, we first present a mathematical formulation of Mitchell's method.

A. Mitchell's Multiplication Method

Consider an N-bit unsigned integer B with bits $b_{N-1}b_{N-2}\ldots b_1b_0$. Then B can be represented as

$$B = \sum_{i=0}^{N-1} 2^i b_i.$$

Assume the leading-one in the integer B is at position k, where $(N-1) \leq k \leq 0$. Then, without the loss of generality, we can write B as

$$B = 2^k\Big(1 + \sum_{i=0}^{k-1} 2^{i-k} b_i\Big). \tag{1}$$

This can be written as

$$B = 2^k(1+x) \tag{2}$$

by letting $x = \sum_{i=0}^{k-1} 2^{i-k} b_i$, where $0 \leq x < 1$ since $k \geq 0$.

The accurate binary log of B is

$$\log_2(B) = k + \log_2(1+x) \tag{3}$$

where k is essentially an integer, representing the characteristic part of the log value, and $\log_2(1+x)$ is the fractional part of the log value. Mathematically, in the linear approximation of

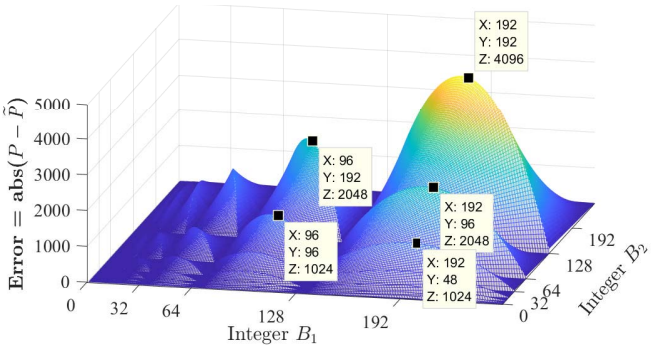

Fig. 2. Magnitude of error $abs(\widetilde{P}-P)$ in Mitchell's multiplication for $B_1 = \{0, 1, \ldots, 255\}$ and $B_2 = \{0, 1, \ldots, 255\}$.

log, the term $\log_2(1+x)$ is approximated as x, since $0 \leq x < 1$. Thus,

$$\widetilde{\log_2}(B) = k + x \tag{4}$$

where the function $\widetilde{\log_2}()$ represents approximate binary log.

For multiplication, we assume that there are two N-bit integers B_1 and B_2, with leading-ones at k_1 and k_2, respectively. They can be represented in the similar manner as

$$B_1 = 2^{k_1}(1 + x_1), \quad B_2 = 2^{k_2}(1 + x_2).$$

To perform multiplication using the log-property, the addition of the approximate log of the two integers, computed using (4), yields

$$\widetilde{\log_2}(\widetilde{P}) = k_1 + k_2 + x_1 + x_2 \tag{5}$$

where $\widetilde{P}$ represents the approximate product. For taking approximate antilog of this expression [adding 1 to the fractional part and scaling with respect to the characteristic, i.e., the inverse operation of the log-approximation in (4)], the term representing the fractional part should be in the range [0,1). However, in this case $0 \leq x_1 + x_2 < 2$ since $0 \leq x_1, x_2 < 1$. Therefore, we decompose it into two cases: $0 \leq x_1 + x_2 < 1$ (there is no carry out from the fractional part to the characteristic); and $1 \leq x_1 + x_2 < 2$ (there is a carry out to the characteristic part). We can write (5) as

$$\widetilde{\log_2}(\widetilde{P}) = \begin{cases} (k_1 + k_2) + (x_1 + x_2); & x_1 + x_2 < 1 \\ (k_1 + k_2 + 1) + (x_1 + x_2 - 1); & x_1 + x_2 \geq 1. \end{cases} \tag{6}$$

Now the inverse of the log approximation can be applied to obtain the Mitchell's approximate product $\widetilde{P}$

$$\widetilde{P} = \begin{cases} 2^{k_1+k_2}(1 + x_1 + x_2); & x_1 + x_2 < 1 \\ 2^{k_1+k_2+1}(x_1 + x_2); & x_1 + x_2 \geq 1. \end{cases} \tag{7}$$

B. Proposed Error-Reduction Scheme

The accurate product P, of the two integers B_1 and B_2 can be written as

$$P = 2^{k_1+k_2}(1 + x_1)(1 + x_2). \tag{8}$$

In Fig. 2, the magnitude of the error between the accurate product P and the approximate product $\widetilde{P}$ is shown for $B_1 = \{0, 1, \ldots, 255\}$ and $B_2 = \{0, 1, \ldots, 255\}$. We can observe that the magnitude of error, as well as the mean of error, is not the same within each power-of-two-interval of input operands (each pair of k_1 and k_2). They increase as k_1 and k_2 increase. Therefore, it suggests that, unlike the error-reduction schemes for log value [15], a fixed error-reduction term cannot be added to the final product to reduce the error because it is not bounded and it is dependent on k_1 and k_2.

However, we describe a key observation here. We can observe that although the error in each power-of-two-interval (each pair of k_1 and k_2) is different, it actually differs by a scaling factor of $2^{k_1+k_2}$. To elaborate this, we determine the error in the product mathematically, which can be written as

$$\begin{aligned} E_P &= \widetilde{P} - P \\ &= \begin{cases} -2^{k_1+k_2}(x_1 x_2); & x_1 + x_2 < 1 \\ -2^{k_1+k_2}(1 + x_1 x_2 - x_1 - x_2); & x_1 + x_2 \geq 1. \end{cases} \end{aligned} \tag{9}$$

We can observe that the error value is always negative for every value of x_1 and x_2. Thus, in order to reduce the overall error and push the average error close to zero, we can subtract the average error of each interval as the error-reduction term from the approximate product.[1] To determine the error-reduction term, we take the average error for given k_1 and k_2 over $x_1 \in [0, 1)$ and $x_2 \in [0, 1)$

$$\begin{aligned} \text{Average}_{\text{Error}} &= \frac{1}{(1-0)(1-0)} \int_0^1 \int_0^1 E_P dx_2 dx_1 \\ &= -2^{k_1+k_2} \left(\int_0^1 \int_0^{1-x_1} (x_1 x_2) dx_2 dx_1 \right. \\ &\qquad \left. + \int_0^1 \int_{1-x_1}^1 (1 + x_1 x_2 - x_1 - x_2) dx_2 dx_1 \right) \\ &= -(0.08333) \times 2^{k_1+k_2}. \end{aligned}$$

Thus, as expected the mean of error in each power-of-two-interval is a constant -0.08333 scaled by $2^{k_1+k_2}$.

From (7), we know that to calculate the approximate product, the sum of the fraction x_1 and x_2 is scaled by a factor $2^{k_1+k_2}$ or $2^{k_1+k_2+1}$ (depending upon if there is carry or not). Therefore, instead of adding the term $(0.08333) \times 2^{k_1+k_2}$ to the final product, we propose to add an error-reduction term before the fractional part is scaled. This saves the requirement of extra scaling hardware in the multiplier design. Mathematically, we propose to add the error-reduction term c as

$$\widehat{\widetilde{P}} = \begin{cases} 2^{k_1+k_2}(1 + x_1 + x_2 + c); & x_1 + x_2 < 1 \\ 2^{k_1+k_2+1}(x_1 + x_2 + c/2); & x_1 + x_2 \geq 1. \end{cases} \tag{10}$$

Thus, irrespective of the value of k_1 and k_2, a single and fixed error-reduction term can be added to reduce the error in multiplication. Equation (10) apparently looks complex, however, if we compare it with (7), we can see that we only need to add the term c to the sum of approximate log of the two operands in combination with a right shift operation for $c/2$. The resulting simple hardware is presented next.

[1]The expressions in (9) involve a product $x_1 x_2$. Hence computing the error exactly is not feasible.

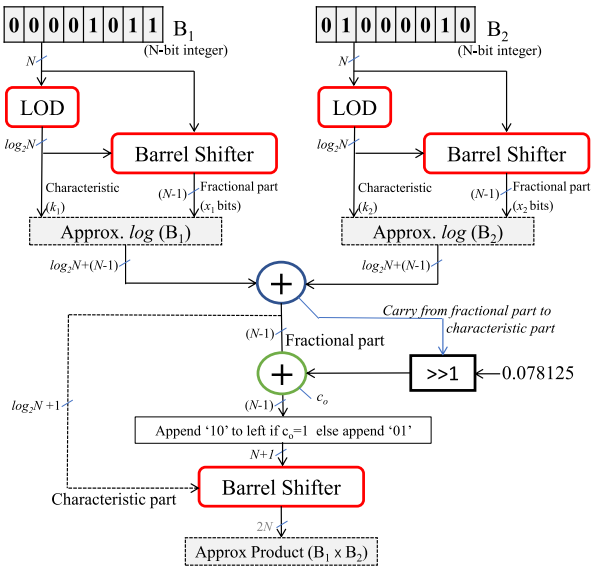

Fig. 3. Proposed MBM multiplier design. A constant error-reduction term is added before the final scaling at the output.

C. MBM Hardware Implementation

The binary representation of the error-reduction term $(0.08333)_{10}$ determined in the last section is $(0.00010101010101\ldots)_2$. As evident, to represent this term exactly, we need a fairly large number of bits, and consequently, a large adder with significant overhead would be required. We propose to approximate this error-reduction term as $(0.0001010)_2 = (0.078125)_{10}$ to reduce the hardware complexity further. Thus, we need only a 7-bit adder, and a 4-bit 2×1 mux (to perform $c/2$) for applying this error-reduction term.

The design for the proposed MBM multiplier realizing (10) is shown in Fig. 3. First, approximate log of the two N-bit integer operands B_1 and B_2 is computed using leading-one detectors (LODs) and barrel shifters. The output bits of the LOD and the barrel shifter for an integer B, represent the characteristic (k) and the fractional part (x) of the approximate log value of the integer, respectively. The approximate log values of the two integers are then added. If $(x_1 + x_2) \geq 1$, the error-reduction term is right-shifted by 1 (else kept as it is) and added to the fractional part of the sum of the two approximate log values. Next, "10" or "01" is appended to the left of this *error-reduced* sum depending upon the carry out from the second adder. This is then scaled by $2^{k_1+k_2}$ or $2^{k_1+k_2+1}$ using another barrel shifter and the characteristic part of the output of the first adder, thus, obtaining the approximate product of B_1 and B_2.

1) Handling Corner Cases: The introduction of the error-reduction hardware produces two corner cases in the design.

Overflow of output beyond 2N bits for N-bit multiplication: Since $0 \leq (x_1 + x_2) < 2$, the addition of the error-reduction term will push this expression to more than 2.0 when (x_1+x_2) is close to 2.0. This is a problem when $k_1 = k_2 = N$, as the multiplier output can overflow to $(2N + 1)$ bits. To solve this problem, we observe that this case happens only when both x_1 and x_2 are close to 1.0, and we know from (9) that the error in Mitchell's multiplication is small when x_1 and x_2 are close to 1.0. Therefore, we can simply ignore the error-reduction in such cases and avoid the overflow. We put a comparator in our multiplier to detect the corner case and then simply pick up the fractional term without error-reduction $(x_1 + x_2)$ using a 2×1 multiplexer. To avoid congestion, this is not shown in Fig. 3.

Loss of bits from error-reduction term when k_1 and k_2 are small: In MBM, the fractional sum $(x_1 + x_2 + 2^{-4} + 2^{-6})$ is scaled by a factor of $2^{k_1+k_2}$. In the case when $(k_1 + k_2)$ is small (<6), some of the bits from the term $(2^{-4} + 2^{-6})$ will still be in the fractional part after scaling, which needs to be chopped off. This is because the final output of the N-bit integer multiplication must be a $2N$-bit integer, without any fractional part. Thus, when $(k_1 + k_2) < 6$, one or more lower-bits of the error-reduction term are lost which may result in high peak error for some of the inputs. We put a condition in our design to detect the cases and handle it appropriately.

D. Error-Configurability in MBM

The proposed MBM design can be made error-configurable. In the multiplier, the fractional part of the approximate log value of an N-bit integer consists of $N-1$ bits (Fig. 3). Since it consists of the bits to the right of the leading-one (at position k) in the integer, $(N-1-k)$ least significant bits in it are always zero. We propose to truncate t least significant bits from the fractional part of the approximate log value of each operand to reduce the adder size in the multiplier. In the event that k is relatively large, such that the chosen $t > N-1-k$, some of the least significant data bits will also be lost while truncating the t lower bits, along with the $(N-1-k)$ zero bits. However, a large k means that the input number is large $(\geq 2^k)$, and thus, intuitively, the loss of a few least significant data bits will not incur a significant error. Further, we set the $(t+1)^{\text{th}}$ bit from the right in the approximate log to 1 in order to reduce the error incurred by truncation (when $t > 0$).

This truncation reduces the size of the main adder, the output barrel shifter as well as the input barrel shifters, and consequently the overall resource consumption of the MBM-t multiplier. Since our error-reduction term requires 7 bits in the fraction, therefore, we restrict t such that $t \leq (N-1) - 7$, so that the error-reduction mechanism remains unaffected.

IV. Results and Evaluation of MBM Integer Multipliers

A. Comparison Details

To demonstrate the usefulness of the proposed MBM multiplier, we compare the design metrics and error characteristics of the proposed MBM multiplier with an accurate multiplier and various state-of-the-art approximate multipliers. These multipliers include: DRUM [1], SSM [2], ESSM [2], UDM [8], UDMB [9], Mitchell (MA) [11], AWTM [12], AMPER [13], and ETM [14].

In addition, we also show the effectiveness of the proposed error-reduction mechanism over existing error-reduction mechanisms for MA. As mentioned in Section II, the linear approximation based variants of Mitchell's method involve

the addition of error-reduction terms. Among those, the error-reduction terms proposed by Abed and Siferd [18], [19] for log and antilog error reductions are the simplest to implement in hardware [15]. Thus, we build an approximate integer multiplier using Abed and Siferd's two-region log and Abed and Siferd's two-region antilog approximation. We refer to it as *Axm2* multiplier and compare it with our proposed MBM multiplier.

B. Experimental Details

We implemented 8-bit and 16-bit versions of all the above mentioned approximate unsigned integer multipliers in Verilog HDL as single-cycle designs. The accurate multipliers and the submultipliers in DRUM, SSM, ESSM, and ETM are implemented as Wallace multipliers. We applied the proposed error-configurability in MBM only for $N = 16$. We do not apply it for the 8-bit version of the multiplier, because we restrict t such that $t \leq (N-1)-7$.

All designs were synthesized using Cadence Encounter RTL Compiler for the TSMC 45-nm standard-cell library at the frequency of 1 GHz. Please note that although sequential elements are placed at the input and output ports of the multipliers for timing, we report the area-footprint and power consumption of only the computation logic (combinational logic) of the designs excluding the sequential elements.

Behavioral simulation models of the designs were also developed in MATLAB and C for error characterization. For 8-bit designs, exhaustive simulations were performed, whereas for the 16-bit designs, Monte Carlo simulations were performed using 2^{24} uniformly distributed random inputs over the interval $\{0, \ldots, (2^{16}-1)\}$. The errors are reported with respect to the accurate result. Three types of error metrics are used to report the error (as in [1]): peak absolute relative error (referred as peak error in this paper), mean of absolute relative error (referred as mean error in this paper), and mean of relative error (referred as error bias). We defined the relative error to be zero if both approximate output and accurate product are zero. The ETM and AWTM multiplier produce nonzero output even when either of the inputs is zero, which results in an *infinite* relative error. Therefore, error characterization for these multipliers was performed excluding the zero input values in the simulations.

C. Simulation and Synthesis Results

The error metric and design metric results for the 8-bit and 16-bit multipliers are presented in Table I. The area and power reduction percentages are calculated with respect to the area and power values of the corresponding accurate version of the N-bit multiplier.

Error improvement with respect to MA: In Table I, we can observe that the magnitude of error bias in the proposed MBM multiplier is close to 0% (0.05% for 8-bit and $< 0.1\%$ for 16-bit implementation). The error bias is $75\times$ lower than that in Mitchell's multiplier for 8-bit implementation, and $37\times$ lower for 16-bit implementation. The proposed MBM multipliers also have less error than the MA in terms of the other two error metrics (mean and peak error), albeit at the cost of some area and power overhead. The error-reduction mechanism overhead in the MBM multiplier (compared to the MA) is less significant in 16-bit implementation than it is in 8-bit implementation. This is because our proposed error-reduction is independent of the multiplier size (N), therefore, the overhead is less significant compared to the overall size of the multiplier when N is large.

TABLE I
DESIGN AND ERROR METRICS FOR THE PROPOSED AND THE MITCHELL-BASED INTEGER APPROXIMATE MULTIPLIERS

Multiplier	Error Metrics			Design Metrics			
	Error Bias (%)	Mean Error (%)	Peak Error (%)	Actual Value Area (μm^2)	Actual Value Power (μW)	% Reduction Area %	% Reduction Power %
Multiplier Size, N = 8-bit							
Accurate	-	-	-	**445.0**	**35.3**	-	-
MBM	0.05	2.69	7.81	288.4	21.6	35.2	38.7
Axm2 [18][19]	−0.69	1.32	14.29	428.1	40.6	3.8	−15.2
MA [11]	−3.76	3.76	11.11	221.6	16.0	50.2	54.6
Multiplier Size, N = 16-bit							
Accurate	-	-	-	**1900.9**	**182.2**	-	-
MBM	−0.09	2.58	7.81	686.3	47.9	63.9	73.7
MBM-1	−0.09	2.58	7.82	647.0	42.9	66.0	76.5
MBM-2	−0.09	2.58	7.83	635.3	40.3	66.6	77.9
MBM-3	−0.09	2.58	7.86	617.4	41.0	67.5	77.5
MBM-4	−0.09	2.58	7.90	597.2	38.2	68.6	79.0
MBM-5	−0.09	2.58	8.00	589.9	35.6	69.0	80.4
MBM-6	−0.09	2.59	8.19	556.7	33.7	70.7	81.5
MBM-7	−0.08	2.59	8.58	535.1	31.5	71.9	82.7
MBM-8	−0.08	2.60	9.36	476.1	28.6	75.0	84.3
Axm2 [18][19]	−0.71	1.36	14.29	815.7	69.0	57.1	62.1
MA [11]	−3.85	3.85	11.11	580.2	40.4	69.5	77.8

Proposed error-reduction scheme is better than the existing ones: We can also observe that the *Axm2* multiplier has nearly twice the peak error and more than $13\times$ the error bias when compared with MBM, for 8-bit implementation. Moreover, its area reduction with respect to the accurate multiplier is a mere 3.8% and the power consumption is even higher than the accurate multiplier. The area and power reduction percentages for the 16-bit *Axm2* multiplier are also smaller than those for the 16-bit MBM-t multipliers. Thus, our proposed approximate multiplier with the novel error-reduction mechanism is much more effective than the existing error-reduction mechanisms for MA.

Error-configurability in MBM design: Table I also includes the results of applying the proposed error-configurability mechanism to the 16-bit MBM multiplier. These multipliers are labeled as MBM-t. The suffix t denotes the number of bits truncated. It can be observed that this truncation has minimal effect on the error bias and mean error of the MBM multiplier. Only the peak error increases slightly with t. The area reduction (with respect to the accurate multiplier) improves from 63.9% to 75% and power reduction improves from 73.7% to 84.3%.

The proposed MBM offers Pareto optimal points when compared with existing approximate multipliers: In order to observe the optimality of the tradeoffs provided by the proposed MBM-t multipliers in comparison to existing approximate multipliers, we plot the error and the design metrics in the form of design spaces. The design spaces are: 1) Area *versus* error bias; 2) Power *versus* error bias; 3) Area *versus* mean error; 4) Power *versus* mean error; 5) Area *versus* peak

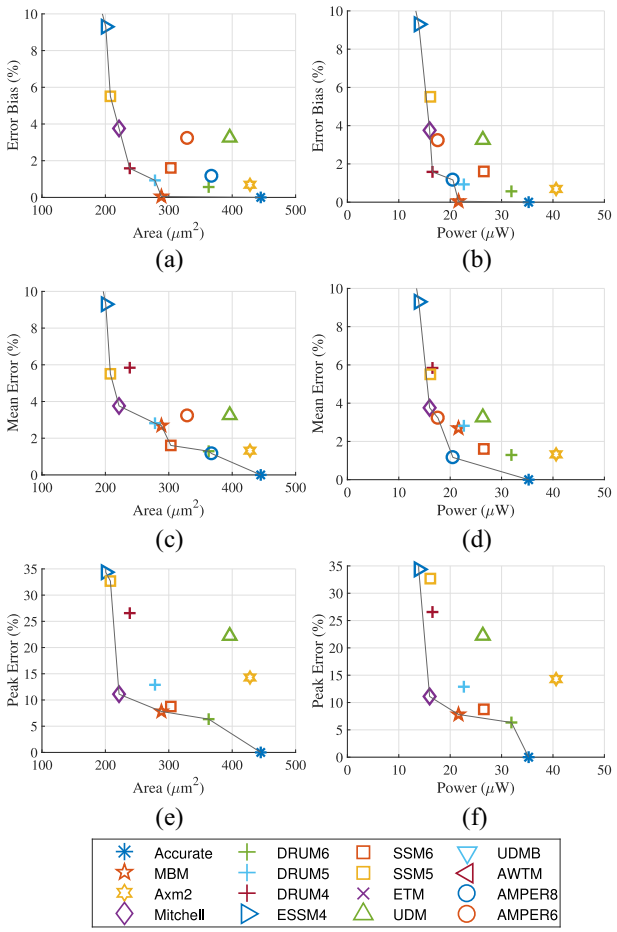

Fig. 4. 8-bit integer multiplier design spaces. The gray line outlines the Pareto front. The proposed MBM is part of the Pareto front in all except (d). (a) Area versus error bias. (b) Power versus error bias. (c) Area versus mean error. (d) Power versus mean error. (e) Area versus peak error. (f) Power versus peak error.

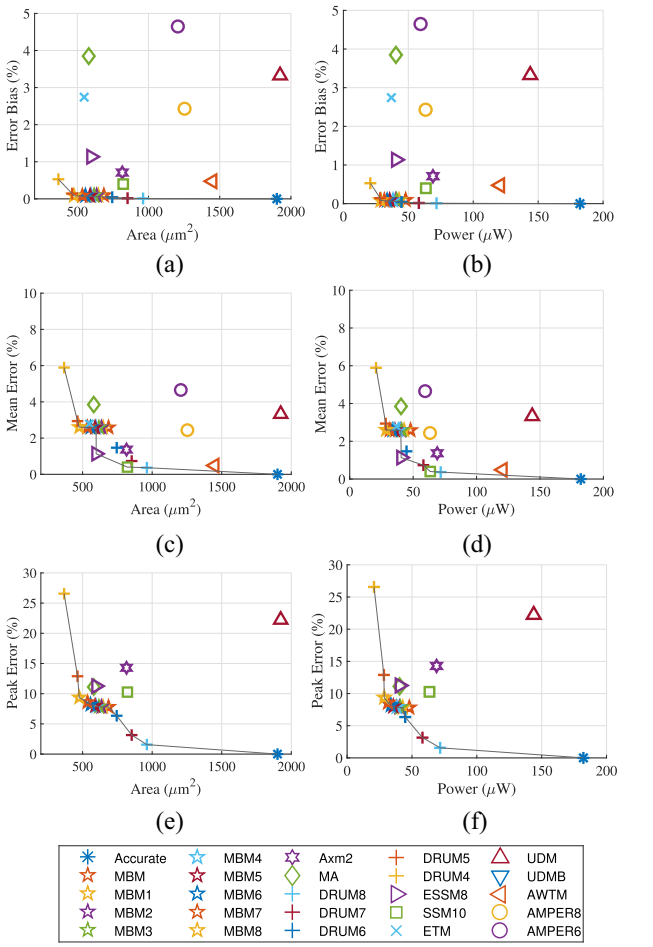

Fig. 5. 16-bit integer multiplier design spaces. The gray line outlines the Pareto front. The proposed MBM-*t* contributes to the Pareto front. (a) Area versus error bias. (b) Power versus error bias. (c) Area versus mean error. (d) Power versus mean error. (e) Area versus peak error. (f) Power versus peak error.

error; and, 6) Power *versus* peak error. These are presented in Fig. 4(a)–(f) for 8-bit designs and Fig. 5(a)–(f) for 16-bit designs. The error metrics are plotted on the *y*-axis and the area/power values are plotted on the *x*-axis. The optimal points in the design space are connected by a gray line. Some of the design points (multipliers) are not visible in the plotted design space. This is because they result in very high error value with a nonoptimal tradeoff.

In the 8-bit integer multiplier design spaces presented in Fig. 4, we can observe that, except for power *versus* mean error design space, the proposed MBM multiplier is a Pareto optimal point in all design spaces. The error bias of the MBM multiplier is the lowest among all evaluated approximate multipliers.

In the 16-bit integer design spaces (Fig. 5), all or some of the MBM-*t* multipliers contribute to the set of optimal points. Other multiplier designs that are included in the Pareto-front are: DRUM for error bias; SSM, ESSM, and DRUM for mean error; and DRUM for peak error. Thus, we can conclude that the proposed MBM multiplier along with error-configurability scheme can provide optimal error-efficiency tradeoffs in the design space consisting of well-known and state-of-the-art approximate integer multipliers.

Using the insights from these results, we propose a scheme for generating approximate FP multipliers, which is presented in the next section.

V. Proposed Approximate Floating-Point Multipliers

A. Determining Resource-Hungry Modules in FP Multiplier

To determine the percentage contribution of each module to the total resource consumption of an FP multiplier, we implemented and synthesized a single-cycle design of an IEEE-754 single-precision FP multiplier in Verilog HDL using the Cadence RTL Compiler.

We found that the mantissa multiplication module contributes to 91.1% of the total area and 92.7% of the total power consumed by an FP multiplier. Hence, for designing resource-efficient FP multipliers, we aim to approximate this module. Mantissa multiplication in IEEE-754 FP standard is essentially an unsigned integer multiplication. Therefore, we can utilize our proposed MBM multiplier and other existing approximate unsigned integer multipliers for mantissa multiplication in the FP multipliers to decrease the overall resource consumption of the FP multipliers.

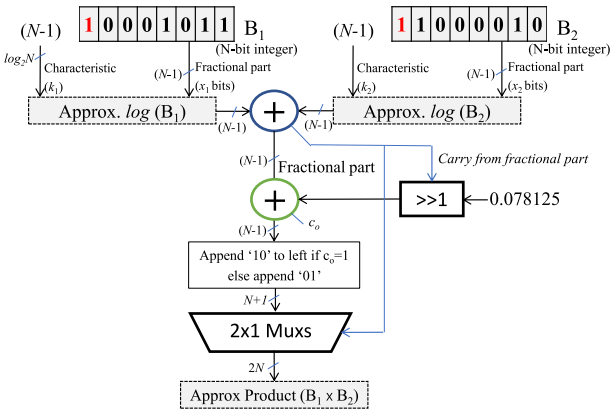

Fig. 6. Optimized MBM multiplier for FP arithmetic. The resource-hungry LODs and barrel shifters are eliminated.

The multipliers that we select to apply in the FP unit are MBM, SSM, DRUM, and MA. The reasons for selecting these multipliers are as follows.

1) These multipliers contribute to optimal (or near optimal) points in the approximate integer multiplier design spaces (Figs. 4 and 5).
2) These multipliers involve detection of leading-one and consequently render themselves for an FP specific hardware optimization (without any further loss of accuracy), as described in the next subsection.

B. Optimizing the Selected Approximate Integer Multipliers

As discussed above, the selected approximate integer multipliers (MBM, DRUM, SSM, and MA) involve detection of the position of the leading-one in each input operand. This demands the use of resource-hungry LODs and barrel shifters, as can be seen in Fig. 3. In the case of mantissa multiplication in FP arithmetic, the leading-one is always located at the MSB position due to normalization of the mantissa. Thus, when using the selected approximate integer multipliers as the mantissa multipliers, the position of the leading-one is always known. Therefore, we can eliminate the LODs and barrel shifters from these multipliers, to achieve a significant logic reduction in the mantissa multipliers without any further loss of accuracy.

As an example, the optimized MBM design is shown in Fig. 6. The resource-hungry LODs and barrel shifters at the inputs are eliminated (please compare with Fig. 3). The characteristics part for both input integers is always $(N-1)$ because the leading-one position is fixed. This also means that the characteristic part is not required to be added. This results in reduction in the size of the first adder. Moreover, in the antilog portion of this log-based multiplier, only shifting by 1 is required, if a carry is generated by the first adder. So, a simple 2×1 multiplexer replaces the large barrel shifter at the output. We achieved similar reduction of logic for DRUM, SSM, and MA multipliers. However, the diagrams are not presented due to space limitation. We refer to these multipliers as the *optimized* approximate integer multipliers, i.e., optimized MBM, optimized DRUM, optimized SSM, and optimized MA.

C. Base Models for the Approximate Floating-Point Multiplier Design Space

Five types of FP multiplier designs are created: differing with respect to the type of integer multiplier used as the mantissa multiplier. These designs are derived from the IEEE-754 single-precision format. Each of them consists of a 24-bit mantissa (including the hidden bit) with one sign bit, and the exponent is 8-bit. These designs are as follows.

1) An FP multiplier with the optimized MBM-t for mantissa multiplication (AFMB).
2) An FP multiplier with the optimized DRUM-k [1] (AFDR).
3) An FP multiplier with the optimized SSM-m [2] (AFSS).
4) An FP multiplier with the optimized MA [11] (AFMA).
5) An FP multiplier with an accurate multiplier for mantissa multiplication (AFAC).

In all these designs, rounding unit is omitted. Subnormal numbers are not supported. The exponent addition is accurate, as the sensitivity of exponent toward error makes it a less viable option for approximation [10]. The exception handling logic is also unmodified. These FP designs are referred to as the *base designs* or *base models*. We apply error configuration schemes on these base models, to generate a variety of approximate FP multipliers which provide various error-area and error-power configurations. This is explained in the following section.

D. Generating the Design Space

1) AFMB-t (With the Optimized MBM-t): As discussed in Section III-D, the MBM integer multiplier design is configurable with a parameter t, that corresponds to the number of bits truncated. We vary this parameter t in the optimized MBM based mantissa multiplier to generate a variety of FP multipliers. Please note that, in this mode, we do not impose the restriction $t \leq (N-1)-7$ (contrary to the case of MBM as a stand-alone integer multiplier) in order to obtain more design points with higher resource savings. Specifically, when $t \geq (N-1)-7$, we also truncate $(t-(N-1-7))$ least significant bits from the error-reduction term.

2) AFDR-t (With the Optimized DRUM-k): The integer DRUM multiplier is error-configurable with a parameter k [1]. This parameter k corresponds to the number of bits that are extracted from the two operands, which then require a smaller submultiplier at the cost of error in the product. When using the optimized DRUM as the mantissa multiplier, we vary this parameter k, consequently generating FP multipliers with different error-efficiency configurations. We define another parameter $t = (24-k)$, which is equivalent to the number of bits dropped from the two input operands.

3) AFSS-t (With the Optimized SSM-m): The integer SSM multiplier is error-configurable with a parameter m [2]. The parameter m corresponds to the number of bits that are extracted from the two operands. We vary this parameter m in the mantissa multiplier, consequently generating FP multipliers with different error-efficiency configurations. We define another parameter $t = (24-m)$, which is essentially equivalent to the number of bits dropped from

the two operands. Please note that the use of the error-configurable *optimized* SSM-*m* multiplier in FP multiplier is equivalent to the traditional precision scaling (double-input truncation) in an FP multiplier. In double-input truncation (DIT), t least significant bits for each input mantissa are truncated before multiplication. Therefore, AFSS-t multipliers will also serve as a comparison to the traditional precision scaling method.

4) AFMA-t (With Bit-Truncation on the Optimized MA): The MA based FP multiplier (AFMA) uses the optimized MA multiplier for mantissa multiplication. We apply the DIT method on AFMA to get more design points. Specifically, truncating t LSBs from the mantissas of the two operands reduces the size of AFMA further, at the cost of more error.

5) AFAC-t (With Single-Input Truncation): The single-input truncation (SIT) method is equivalent to the precision scaling used by Tong *et al.* [10] and Zhang *et al.* [20]. In SIT, t least significant bits from one of the input mantissas are truncated before multiplication. Thus, the size of the mantissa multiplier is effectively reduced to a $24\times(24-t)$ sized multiplier. We apply SIT on AFAC to generate various error-efficiency configurations. The inclusion of these multipliers in our design space serves as a comparison of our proposed novel approximate FP multipliers with existing works.

VI. Results: Approximate FP Multipliers

A. Implementation and Experiment Details

The base model multipliers, as well as their error-configurable versions, are single-cycle designs. The sub-multipliers were implemented as Wallace multipliers. The multiplier designs were coded in Verilog HDL and synthesized using Cadence Encounter RTL Compiler for the TSMC 45-nm standard-cell library with a clock rate of 1 GHz. For comparison purposes, we also designed and synthesized an IEEE-754 single-precision FP multiplier (FPM-R), with round-to-nearest rounding and without subnormal number support. Although sequential elements are placed at the inputs and outputs of the multipliers for timing, we report only the combinational area and power (that corresponds to the computational logic) of our multipliers in this paper. The error and resource comparisons are reported with respect to the reference FPM-R design.

Behavioral simulation models of the designs were also developed in MATLAB for error characterization and analysis. The relative error in the approximate FP multiplication is defined as

$$\text{rel\_err}\% = \frac{\text{Output}_{\text{Sing.Prec.}} - \text{Output}_{\text{approx}}}{\text{Output}_{\text{Sing.Prec.}}} \times 100. \tag{11}$$

The error characterization and analysis were performed through random input tests. First, the approximate base models were tested with one million uniformly distributed random inputs in the range [1.0, 2.0) to calculate the peak error, mean error, and error bias. This range was selected because we approximate only the mantissa module and not the exponent module. Therefore, the relative error values are unaffected by scaling (exponent value). A similar test for error analysis was performed in [20].

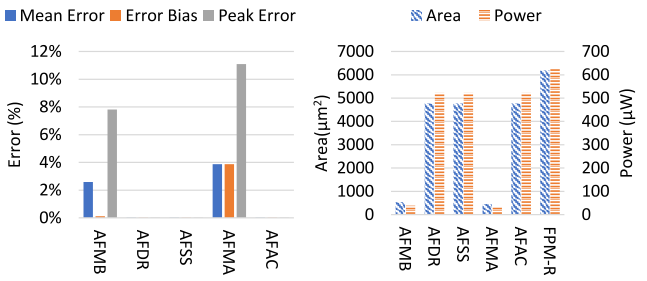

Fig. 7. Error and design metric results for *base models*. MBM based AFMB and MA based AFMA show >15× power and >10× area improvement.

The FP multipliers from the design space generated by the application of error-configuration schemes were tested with fifty thousand random inputs (uniformly distributed in [1.0, 2.0) range). The peak error, mean error, and error bias for each multiplier were calculated and reported.

B. Simulation and Synthesis Results

1) Results for the Base Models: Fig. 7 presents the design metric and error metric results for the implemented *base models* along with the results for the reference FPM-R design. As expected, the optimized MA based AFMA is the most resource-efficient multiplier but with the highest error. It shows a power improvement of 19.2× and an area improvement of 13.6× when compared to the reference FPM-R. The optimized MBM based AFMB also shows a significant power improvement (16.1×) and area improvement (11.7×). The error metric values for AFMB are lower than that of AFMA. The error bias for AFMB is very small. Please note that the base models are implemented with $t = 0$, therefore the AFDR, AFSS, and AFAC consists of 24-bit mantissa multipliers. They show a slight improvement in the design metric values due to the elimination of the rounding unit with a negligible error. Although these three base models generate similar results, the advantage of creating these base models will be clearly visible when the error-configuration schemes are applied (t is varied), as each of these base models will provide different design points. This is presented next.

2) Results for the Generated FP Multiplier Design Space: The application of the error-configuration schemes on the *base models*, as discussed in Section V-D, produces various error-area and error-power tradeoffs. We first present the variation of the three error metrics, power, and area with the parameter t (the number of bits truncated) in Fig. 8(a)–(e) for each base model. The error metric values in Fig. 8(a)–(c) do not change significantly for $t \leq 15$. The optimized MA based AFMA-t has the highest magnitude for each error metric. The optimized MBM based AFMB-t exhibits lower peak and mean error than AFMA-t. AFMB-t also produces a very small error bias for $t \leq 15$. The AFDR-t, AFSS-t, and AFAC-t have very small error values in the range $t \leq 15$. For the range $t > 15$, the error values for all base models increase. Fig. 8(d) and (e) presents the power and area variation with t where both metrics show similar trends. The area and power values of AFMA-t are less than the area and power of others for each value of t. The area and power values for AFMB-t are slightly higher than

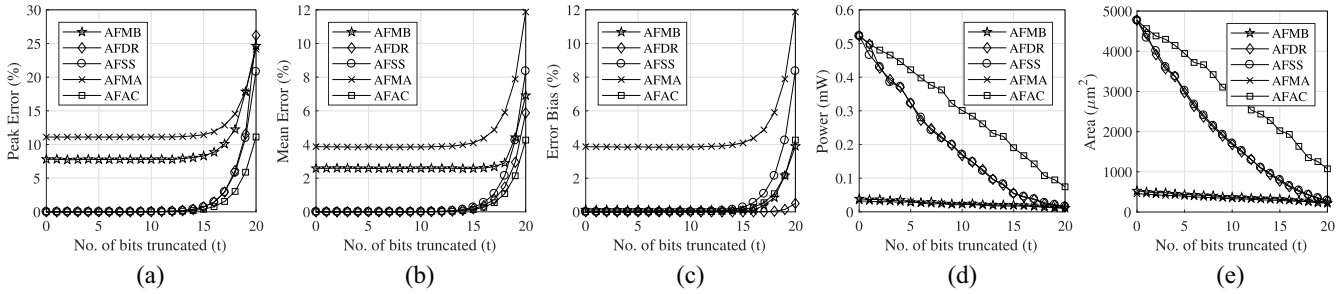

Fig. 8. Variation of (a) peak error, (b) mean error, (c) error bias, (d) power, and (e) area with the number of bits truncated t.

for AFMA-t but much less than those for AFDR-t, AFSS-t, and AFAC-t. The two design metrics have a high value for AFDR-t, AFSS-t, and AFAC-t when t is small, but decrease faster with t than for AFMB-t and AFMA-t. This is due to AFDR-t, AFSS-t, and AFAC-t consisting of sub-multipliers as opposed to sub-adders in the log-based AFMB-t and AFMA-t.

In order to analyze the error-power and error-area tradeoffs offered by these multipliers, we present six types of design spaces: 1) Power *versus* mean error; 2) Area *versus* mean error; 3) Power *versus* error bias; 4) Area *versus* error bias; 5) Power *versus* peak error; 6) Area *versus* peak error. These are shown in Fig. 9(a)–(f), respectively. These design spaces are plotted with the error metric on the y-axis and area/power improvement (with respect to the FPM-R) on the x-axis. The maximum area improvement offered by the proposed FP multipliers is 28× and maximum power improvement is nearly 57×. A few important observations can be made in Fig. 9(a)–(f). The optimized MBM based AFMB-t generally provides better design points than the optimized MA based AFMA-t multiplier. The optimized DRUM based AFDR-t generally offers better design points than traditional precision scaling (optimized SSM based AFSS-t). The AFSS-t is better than AFAC-t (single-input truncation) for all cases.

The optimal design points (constituting the Pareto front) are highlighted in red and connected with a gray line in each design space. For mean error design spaces, shown in Fig. 9(a) and (b), we can observe that our optimized MBM based AFMB-t and the optimized DRUM based AFDR-t contribute to the Pareto front. Specifically, for power *versus* mean error design space, shown in Fig. 9(a), AFDR-t provides optimal design points for achieving power improvements of 1–26×, while AFMB-t provides best design tradeoffs for achieving power improvements of 26–57×. In area *versus* mean error design space, shown in Fig. 9(b), AFDR-t provides optimal points for area improvements of 1–15×, whereas for area improvements of 15–28×, AFMB-t offers best tradeoffs.

For error bias design spaces, shown in Fig. 9(c) and (d), again the Pareto front consists of design points from AFMB-t and AFDR-t only. In the range of 1–26× power improvement and 1–15× area improvement, AFDR-t produces optimal points. In the range of 26–40.5× power improvement and 15–22.5× area improvement, both AFMB-t and AFDR-t contribute to the Pareto front. Beyond that, for area improvements

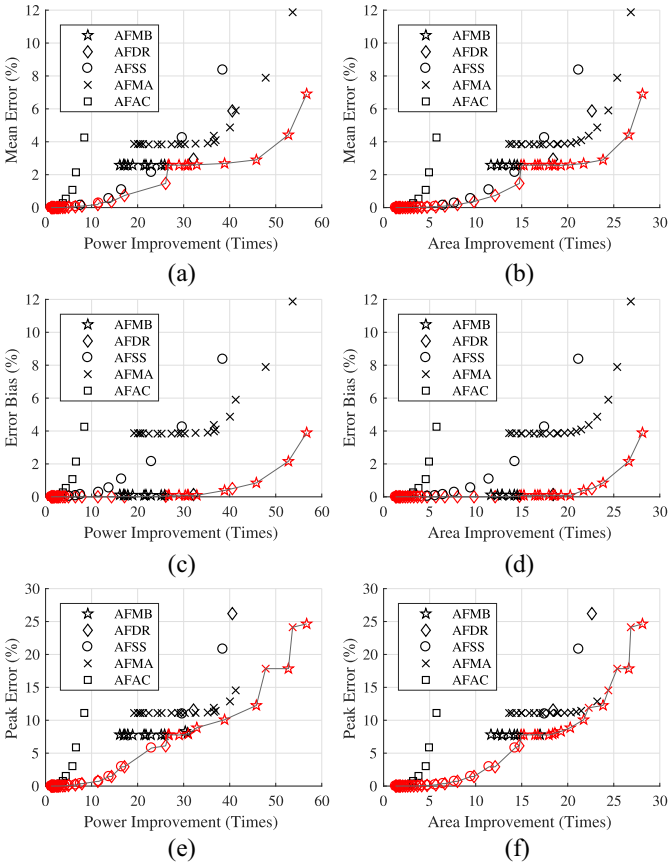

Fig. 9. Proposed FP multiplier design spaces. The gray line outlines the Pareto front with the Pareto points highlighted in red. The MBM based AFMB-t and DRUM based AFDR-t contribute to the Pareto front. (a) Power improvement *versus* mean error. (b) Area improvement *versus* mean error. (c) Power improvement *versus* error bias. (d) Area improvement *versus* error bias. (e) Power improvement *versus* peak error. (f) Area improvement *versus* peak error.

of 22.5–28× and power improvements of 40.5–57×, only AFMB-t contributes to the Pareto front.

The design spaces of peak error, shown in Fig. 9(e) and (f), are interesting as the Pareto front consists of design points from all base models except AFAC-t. The design points from AFDR-t and AFSS-t contribute to the lower range of optimal resource savings while AFMB-t and AFMA-t contribute to the higher range of optimal resource savings. Specifically, AFDR-t and AFSS-t offer area optimal points in the range of 1–15× area improvement and 1–26× power improvement.

(a) (b) (c)

Fig. 10. Image filtering results using accurate and the proposed MBM-8 multiplier. There is perceptually no difference in the two filtered images. (PSNR = 43 dB). (a) Original image. (b) Filtered—accurate. (c) Filtered—MBM8.

The AFMB-*t* and AFMA-*t* provide optimal points in the range of 15–28× area improvement and 26–57× power improvement.

Overall, we observe that design spaces are wide and dense. Moreover, our proposed integer MBM based FP multiplier contributes to optimal points in each of the design space offering up to 28× area improvement and 57× power improvement when compared with the IEEE single-precision FP multiplier. Please note that although some of these approximate FP multipliers involve log-based subcomponents, the overall FP multiplier unit can be considered as a black-box by the application as the computation of log and antilog is handled within the unit. Moreover, although we demonstrated the proposed approximation for FP multiplication using IEEE single-precision format, it can also be applied to double-precision and half-precision multipliers.

VII. Application Evaluations

A. Image Filtering (Using Integer Multipliers)

In the domain of image processing, Gaussian filtering is popular due to its desirable frequency domain characteristics. We use Gaussian filtering to demonstrate the application-level evaluation of our proposed approximate integer multipliers.

Experiment and Results: We use a 7×7 Gaussian kernel with $\sigma = 1.5$ to filter a 256×256-pixel grayscale standard *Mandrill* image. The image and the filter coefficients are represented in 16-bit fixed-point representation. We first perform filtering using accurate 16-bit multiplication in software, and then replace each accurate multiplication with the software model of the proposed MBM-8 multiplier. We use peak signal-to-noise ratio (PSNR) as the application-level quality metric to compare the quality of the image filtered using MBM-8, with respect to the image filtered using accurate multipliers.

The results are presented in Fig. 10. Fig. 10(a) shows the input image, and Fig. 10(b) and (c) presents the filtered images using 16-bit accurate and MBM-8 multipliers, respectively. We can observe that the difference in quality is imperceptible with the human eye. The PSNR value for Fig. 10(c) with respect to Fig. 10(b) is 43 dB.

B. JPEG Encoding and Decoding (Using Integer and FP Multipliers)

JPEG encoding and decoding consist of computing DCT and iDCT of an input image which involve computations with real numbers. In this application example, we perform the image coding and decoding using floating-point and fixed-point arithmetic. We then evaluate the quality of the compressed images with reference to the uncompressed image using PSNR metric. Please note that for fixed-point, additional implementation effort and time is required.

TABLE II
JPEG Compression Results Using Various Multipliers. The Values for Area/Power Improvement Per Multiplication Are With Respect to IEEE Single-Precision Multiplier. PSNR Values Are in dB

Multiplier Used		Power Improvement	Area Improvement	PSNR Value
Floating-Point	IEEE Single-Precision	-	-	29.12
	AFDR (t=10)	3.74×	3.68×	29.12
	AFDR (t=15)	11.36×	8.02×	29.04
	AFMB (t=05)	21.81×	14.20×	27.24
	AFMB (t=10)	26.66×	16.75×	27.24
	AFMB (t=15)	30.35×	19.27×	27.73
16-bit Fixed Point	Accurate	3.43×	3.25×	29.11
	MBM	13.04×	9.00×	27.06

We perform image compression of a 256×256-pixel standard *cameraman* image in grayscale. The results are presented in Table II. The table also presents area and power improvement per multiplication with respect to the IEEE single-precision FP multiplier (the FPM-R). The table shows that using accurate 16-bit fixed-point arithmetic allows us 3.43× power improvement with almost the same PSNR value of the compressed image as when the IEEE single-precision FP multiplier is used. However, we can observe that using our proposed floating-point AFDR-15 multiplier, we can achieve similar image quality with greater improvements of area (8.02×) and power (11.36×). Further power improvements of 21.81×, 26.66×, and 30.35× are also possible using the proposed AFMB-*t* with slight degradation in image quality. Therefore, despite the additional effort required in fixed-point arithmetic, the proposed approximate FP multipliers can offer higher area and power savings than the fixed-point arithmetic. The MBM based AFMB-*t* offers higher savings than the integer MBM because we used the optimized MBM in AFMB-*t*.

C. Convolutional Neural Network (Using FP Multipliers)

In the domain of machine learning, AlexNet is a popular CNN model for image classification proposed by Krizhevsky *et al.* [25]. The model is designed to classify 1000 object categories from ImageNet dataset [26]. AlexNet consists of nearly 0.7 billion FP multiplications. This makes it highly compute intensive.

Experiment and Results: We implemented a software realization of AlexNet CNN. The CNN is pretrained on ILSVRC-2012 dataset [26]. We performed classification over a dataset of 500 images from the ILSVRC-2012 validation set using IEEE single-precision FP operations. The measured classification error rates are Top-1[2] = 42.8% and Top-5[3] = 19.6%. We then replaced each FP multiplication in the implemented CNN

[2]The fraction of the test images for which the top prediction is incorrect.
[3]The fraction of the test images for which all top 5 predictions are incorrect.

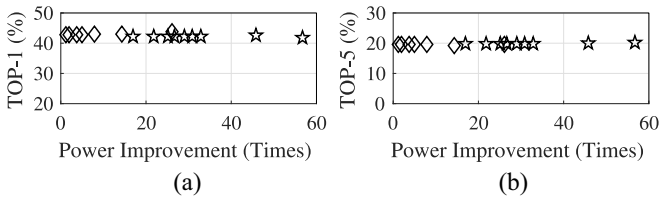

Fig. 11. AlexNet classification error rate for the selected approximate FP multipliers. Legend for this figure is the same as in Fig. 9. (a) Top-1 error. (b) Top-5 error.

with the software-based functional models of a subset of the proposed approximate FP multipliers (AFMB-t and AFDR-t) and performed classification over the same set of images. The selected approximate FP multipliers for this task are: MBM based AFMB-t, with $t = 2, 5, 8, 10, 12, 14, 16, 18, 20$; and DRUM based AFDR-$t$, with $t = 0, 5, 10, 12, 14, 16, 18$.

The Top-1 and Top-5 error rate results using each of the selected approximate multipliers are shown in Fig. 11, where the classification error rates are plotted against the power improvement per FP multiplier. We observe that these classification errors are very close to the classification error obtained by using single-precision FP implementation (42.8% and 19.6%). The Top-1 error using approximate FP multipliers fluctuates between 41.8% and 43.8% and the Top-5 error fluctuates between 19.2% and 20.2%. For the approximate multiplier (AFMB-t with $t = 20$) that offers 57× power improvement and 28× area improvement, the error results are 41.8% and 20.2% for Top-1 and Top-5, respectively. Thus, our proposed approximate multipliers can reduce the power and area contributed by the FP multipliers to the whole system up to 57× and 28×, respectively, with negligible or no degradation in the classification accuracy of AlexNet.

VIII. Conclusion

We presented an error-configurable minimally-biased approximate integer multiplier (MBM). We then proposed a variety of approximate FP multipliers using the proposed MBM and three other existing approximate integer multipliers. For designing these approximate FP multipliers, we first proposed a novel optimization technique for the approximate integers multipliers (MBM, DRUM, SSM, and MA) and used them in our FP multipliers as the mantissa multiplier. Our synthesis and simulation results show that the proposed MBM integer design provides Pareto optimal design points in the integer multiplier design space of power and area with respect to three different error metrics. We also show that the proposed approximate FP multipliers, built using optimized MBM and DRUM, can offer better tradeoffs than the traditional precision scaling methods while offering much higher area savings (>25×) and power savings (>50×) compared to IEEE single-precision multiplier. The application-level evaluations of the proposed approximate integer and FP multipliers demonstrate minimal degradation in the application-level accuracy.

References

[1] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *Proc. ICCAD*, Austin, TX, USA, Nov. 2015, pp. 418–425.

[2] 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 Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 6, pp. 1180–1184, Jun. 2015.

[3] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surveys*, vol. 48, no. 4, p. 62, 2016.

[4] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. DAC*, San Francisco, CA, USA, Jun. 2015, pp. 1–6.

[5] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *Proc. ASPLOS*, London, U.K., Mar. 2012, pp. 301–312.

[6] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. DAC*, Austin, TX, USA, May 2013, pp. 1–9.

[7] A. Raha and V. Raghunathan, "qLUT: Input-aware quantized table lookup for energy-efficient approximate accelerators," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, p. 130, 2017.

[8] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. Int. Conf. VLSI Design*, Jan. 2011, pp. 346–351.

[9] S. Rehman *et al.*, "Architectural-space exploration of approximate multipliers," in *Proc. ICCAD*, Austin, TX, USA, Nov. 2016, pp. 1–8.

[10] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar, "Reducing power by optimizing the necessary precision/range of floating-point arithmetic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 3, pp. 273–286, Jun. 2000.

[11] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vol. EC-11, no. 4, pp. 512–517, Aug. 1962.

[12] K. Bhardwaj, P. S. Mane, and J. Henkel, "Power-and area-efficient approximate Wallace tree multiplier for error-resilient systems," in *Proc. Int. Symp. Qual. Elect. Design*, Santa Clara, CA, USA, Mar. 2014, pp. 263–269.

[13] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proc. DATE*, Dresden, Germany, Mar. 2014, pp. 1–4.

[14] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Proc. EDSSC*, Hong Kong, Dec. 2010, pp. 1–4.

[15] J. Y. L. Low and C. C. Jong, "Unified Mitchell-based approximation for efficient logarithmic conversion circuit," *IEEE Trans. Comput.*, vol. 64, no. 6, pp. 1783–1797, Jun. 2015.

[16] M. Combet, H. V. Zonneveld, and L. Verbeek, "Computation of the base two logarithm of binary numbers," *IEEE Trans. Electron. Comput.*, vol. EC-14, no. 6, pp. 863–867, Dec. 1965.

[17] E. L. Hall, D. D. Lynch, and S. J. Dwyer, "Generation of products and quotients using approximate binary logarithms for digital filtering applications," *IEEE Trans. Comput.*, vol. C-19, no. 2, pp. 97–105, Feb. 1970.

[18] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power logarithmic converter," *IEEE Trans. Comput.*, vol. 52, no. 11, pp. 1421–1433, Nov. 2003.

[19] K. H. Abed and R. E. Siferd, "VLSI implementation of a low-power antilogarithmic converter," *IEEE Trans. Comput.*, vol. 52, no. 9, pp. 1221–1228, Sep. 2003.

[20] H. Zhang, W. Zhang, and J. Lach, "A low-power accuracy-configurable floating point multiplier," in *Proc. ICCD*, Seoul, South Korea, Oct. 2014, pp. 48–54.

[21] H. Zhang, M. Putic, and J. Lach, "Low power GPGPU computation with imprecise hardware," in *Proc. DAC*, San Francisco, CA, USA, Jun. 2014, pp. 1–6.

[22] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits," *Proc. IEEE*, vol. 98, no. 2, pp. 253–266, Feb. 2010.

[23] R. Pawlowski *et al.*, "A 530mV 10-lane SIMD processor with variation resiliency in 45nm SOI," in *Proc. Int. Solid-State Circuits Conf.*, San Francisco, CA, USA, Feb. 2012, pp. 492–494.

[24] L. B. Soares, S. Bampi, A. L. R. Rosa, and E. A. C. Costa, "Near-threshold computing for very wide frequency scaling: Approximate adders to rescue performance," in *Proc. NEWCAS*, Grenoble, France, Jun. 2015, pp. 1–4.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2012, pp. 1097–1105.
[26] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

Hassaan Saadat received the B.E. and M.S. degrees in electrical engineering from the National University of Sciences and Technology (NUST), Islamabad, Pakistan, in 2011 and 2015, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, University of New South Wales, Sydney, NSW, Australia.

He was a Research Assistant with NUST, where he researched on a project sponsored by the Electronics and Telecommunications Research Institute, Daejeon, South Korea. His current research interests include approximate computing and embedded systems for multimedia and machine learning applications.

Haseeb Bokhari received the B.E. degree in electronics engineering from the National University of Sciences and Technology, Islamabad, Pakistan, in 2010, and the Ph.D. degree from the University of New South Wales (UNSW), Sydney, NSW, Australia, in 2016.

He then held a Research Associate Position with UNSW, while researching on a Canon Information Systems Research Australia sponsored project. He was a Software Engineer with Open Access Pty. Ltd., Sydney. He is currently a Senior Firmware Engineer with Seagate Technology, Singapore. His current research interests include embedded systems, low power devices, and hardware–software co-design.

Sri Parameswaran (SM'04) received the B.E. degree in electrical and computer systems engineering from Monash University, Melbourne, VIC, Australia, in 1986, and the Ph.D. degree from the University of Queensland, Brisbane, QLD, Australia, in 1991.

He is a Professor with the School of Computer Science and Engineering, University of New South Wales, Sydney, NSW, Australia, where he also serves as the Program Director of Computer Engineering. His current research interests include system level synthesis, low-power systems, high-level systems, hardware security, and embedded systems for genomic computing.

Dr. Parameswaran is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and the *EURASIP Journal on Embedded Systems*. He has served on the Program Committees of several international conferences, such as the Design Automation Conference, the Design Automation and Test in Europe Conference, the International Conference on Computer-Aided Design, the International Conference on Hardware/Software Codesign and System Synthesis (as the Technical Program Committee Chair), and the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems.