LeAp: Leading-one Detection-based Softcore Approximate Multipliers with Tunable Accuracy

Zahra Ebrahimi, Salim Ullah, Akash Kumar
Chair For Processor Design, Center for Advancing Electronics Dresden (CfAED),
Technische Universität Dresden, Germany
Corresponding Author's Email: zahra.ebrahimi_mamaghani@tu-dresden.de

***Abstract*—Approximate multipliers are ubiquitously used in diverse applications by exploiting circuit simplification, mainly specialized for *Application-Specific Integrated Circuit* (ASIC) platforms. However, the intrinsic architectural specifications of *Field-Programmable Gate Arrays* (FPGAs) prohibited comparable resource gains when directly applying these techniques. LeAp is an area-, throughput-, and energy-efficient approximate multiplier for FPGAs which efficiently utilizes 6-input *Look-up Tables* (6-LUTs) and fast carry chains in its novel approximate log calculator to implement Mitchell's algorithm. Moreover, three novel error-refinement schemes with negligible area overhead and independent from multiplier-size, have boosted accuracy to⩾99%. Experimental results obtained from Vivado, *Artificial Neural Network* (ANN) and image processing applications indicate superiority of proposed multiplier over accurate and state-of-the-art approximate counterparts. In particular, LeAp outperforms the 32x32 accurate multiplier by achieving 69.7%, 14.7%, 42.1%, and 37.1% improvement in area, throughput, power, and energy, respectively. The library of RTL and behavioral implementations will be open-sourced at *https://cfaed.tu-dresden.de/pd-downloads*.**

***Index Terms*—Field-Programmable Gate Arrays, Approximate Multiplier, Mitchell's Multiplication Algorithm, Energy-Efficiency, Area-Optimization.**

I. Introduction

With the cease of Dennard scaling era, the strive to prevent breakdown of Moore's law has led to re-emergence of approximate computing which trades accuracy to meet the available budget. This technique has become pronounced considering the ever-increasing cognitive applications in computer vision, big data, and probabilistic-machine learning. The paramount design constraints in these applications are energy dissipation and latency as they are constantly fed with bulk of data entailing fast outcome computation in a limited energy budget. Multipliers are one of the most energy-hungry units extensively employed in the main kernels of aforementioned programs which carves out a prominent niche for their approximation. Of particular instances are, Deep Neural Networks which enjoy a renaissance in today's computational world (11×10^9 Multiply-Accumulate operation/image in Resnet-152 dominates 99% of computational energy [1]). Hence, reducing the area and energy of computational units while increasing throughput (which is of the same importance as memory optimization [2]) enables supporting of larger networks, especially in fixed-budget IoT/wearable gadgets.

Field-Programmable Gate Arrays (FPGAs), rewarded by high degree of parallelism to accelerate these applications, have been augmented with hard-wired DSP blocks to excel fixed- and floating-point multiplication. Nevertheless, hosting off-the-shelf DSP blocks cannot always guarantee to fulfill design requirements in a variety of application domains. As shown in [3], the fixed locations of DSP blocks within FPGAs increase routing complexity and result in increasing the critical path delays of some circuits. Furthermore, in concurrently executing programs' environments or multiplication-intensive applications, the limited ratio of DSP blocks versus LUTs ($<$0.001) forces designers to utilize LUT-based softcore multipliers. Finally, the degraded performance and heavy utilization of DSP blocks by a single application (Viterbi decoder, Reed-Solomon and JPEG encoders) discussed in [4] are from motivational examples that testify on this inefficiency of hard multipliers in many applications. Therefore, in spite of availability of DSP blocks-based hard multipliers, soft- *Intellectual Property* (IP) versions are provided by major FPGA vendors such as Xilinx and Intel [5].

Most of the works on approximate multiplier have focused on either simplifying *Partial Product* (PP) generation/accumulation/summation or using smaller multipliers by LSB truncation which are mainly deployed in *Application Specific Integrated Circuit* (ASIC) fabrics. However, two main challenges still exist: a) These techniques are not generic since approximation principles as defined for ASIC neglect the differences in underlying reconfigurable infrastructure and yield insignificant improvements when directly synthesized and ported to FPGAs [7]. b) In contrast to ASIC, limited studies have evaluated these techniques on FPGAs [4], [9], [19]–[21]. Two drawbacks are linked to these works: designs with hierarchical implementation approach are not efficiently scalable, as integrating smaller imprecise instances may lead to further deterioration of the output due to more and bigger errors being generated by cascading. This obstructs their usage in majority of applications that utilize wide-input multipliers. Second, these multipliers have mainly considered PP-based designs and not considered other options that can profit underlying architectural characteristics of FPGAs and render higher savings. Another point should be noted is that modern error-resilient systems accept different accuracy bounds. Hence, it is highly desirable to enable adjustable precision for the same circuit with minimal overhead that do not violate user constraints while still producing viable results. This highlights the need for exploring novel avenues to provide a roadmap enabling multipliers with tunable accuracy-resource trade-offs specifically in FPGAs.

To tackle above-mentioned challenges, we present *Leading-one detection-based area- and energy-efficient Approximate multiplier* (LeAp) with tunable accuracy, specifically tailored for FPGAs. The motivation behind elaborating LeAp upon Mitchell's algorithm is the facilitated implementation of multiplication translated to addition in his algorithm which is the simplest linearly-approximated logarithmic multiplier. This translation enables savings of area, power, delay, and perfectly fits FPGAs, as they are already equipped with fast carry chains hardened to accelerate addition. Occupying less resource will provide the opportunity for reconfigurable accelerators to further execute multiplication-intensive workloads. The novel

978-1-7281-4123-7/20/$31.00

TABLE I: Summary of Approximate Multiplier in the Literature

Approach	Work	Description	Platform	Improvement	Accuracy up to
Partial product generation/ addition/ accumulation	[4]	4-, 8-, and 16-bit multipliers with approximate PPs using 4x2 instances	FPGA	{Area, energy, and latency} +	99.7%
	[6]	Up to 16-bit adders using approximate half and full adder		{Area, energy} ++, Latency +	92%
	[7]	4x4 and 8x8 multiplier with approximate partial products		{Area, power, and latency} +	Configurable
	[8], [9]	2x2 multiplier based on simplifying Karnaugh Map and fast adders	ASIC	Power ++, Latency +	96.6%
	[10], [11]	Library of larger multiplier and adders using 2x2 instances		{Area, power} +	Configurable
	[12]	MxM multiplier using $\frac{M}{2}$x$\frac{M}{2}$ with carry-prediction and truncation of PPs		{Area, power} ++, Latency +	98.5%
Resize MxM to NxN multiplier	[13]	Select the N bit starting from leading one	ASIC	{Area, power} +++	Configurable
	[14]	Pass MSB bits to multiplier if it consists of leading one, else LSBs		{Power, latency} +	99%
	[15]	Select M bit from one of 2 fixed position (including the leading one)		Energy ++	99%
Using Mitchell's algorithm [16]	[16]	Translate multiplication to addition	ASIC	{Area, power} +++	96.2%
	[17]	Error reduction through piece-wise approximation of inputs		Area-delay product ++	99.9%
	[18]	Adding one error-reduction term to [16]		{Area, power} ++	97.2%
	LeAp	Efficient approximate log calculator and error-reduction schemes	FPGA	{Area, power & energy}++, Throughput +	99.2%

contributions of this paper are outlined as follows:

- **Novel architecture of approximate log calculator and Mitchell's algorithm customized for FPGAs**. Leading-one detection and fractional part alignment (log calculation) are two inherent steps in Mitchell's algorithm. Prior works used large priority encoders and barrel shifters while LeAp implements both steps concurrently with the same, small number of FPGA primitives [6]. This saves ~70% area compared to accurate multiplier, plus smaller delay and energy compared to both accurate and Mitchell's multiplier.
- **Achieving precision variability by three novel error-reduction schemes with minimal overhead**. Our mechanisms that can tune error to a desirable bound (average relative error $<1\%$), are easily scalable and can be coupled with multiplier of any size. It is noteworthy, unlike existing state-of-the-art error-refinement approaches [17], [18] that a non-trivial circuitry is needed for error-refinement step, LeAp neither incurs additional adder to the original design, nor depends on the intermediate outcome during the Mitchell's algorithm. In contrast, the addition of error-coefficients in LeAp is devised based on directly configuring same LUTs and their associated fast carry chains for addition of fractional part (provided in Xilinx UNISIM library [22]) and omitting usage of predefined IPs. This enables Vivado to perform synthesis optimizations leading to diminished area and latency of final design [23].

We evaluate LeAp against established approximate multipliers in the literature with respect to accuracy and performance metrics. Experimental measurements indicate that LeAp surpasses cutting-edge approximate multipliers in terms of area, throughput, and energy while improving accuracy of Mitchell's method. The RTL and behavioral models of the proposed LeAp will be open-sourced and available online at *https://cfaed.tu-dresden.de/pd-downloads* to allow accelerate research on approximate multipliers.

II. Related Work

A. Partial product approximation

In [4], [6]–[9] inexact simplified 2x2, 4x4, 8x8 multipliers/adders are designed, while [10], [11] has provided libraries of approximate adders and multipliers with various resource-accuracy trade-off. The main shortcoming attributed to these works is weak-scalability when transported to larger input-width, i.e., simplification of karnaugh map or PP tree should start from scratch, otherwise *error drastically increases* as it becomes accumulated in a recursive design approach [8].

B. Resize multiplication by Leading-one detection/truncation

Studies in this category use two approaches: 1) MSBs of the inputs are forwarded to smaller accurate multiplier and then shifted to the output appropriately which *impose error cases equal to 100%*. 2) large priority encoders and barrel-shifters are used for combinational implementation of *Leading One Detector* (LOD) and extracting fractional-parts [24]. However, these ASIC-based approaches *weakly fit FPGAs due to using layers of multiplexers (poorly map on LUTs [25])*. Targeting FPGAs, authors in [26] proposed a 8-bit LOD which uses consecutive levels of LUTs to implement each bit of LOD-output since each bit is a function of all bits in the input. The negative resultant of this cascading is exacerbation of circuit latency. Moreover, using large barrel-shifters for re-aligning fractional part is inevitable which further increases overhead.

C. Approximate Multiplication Algorithm

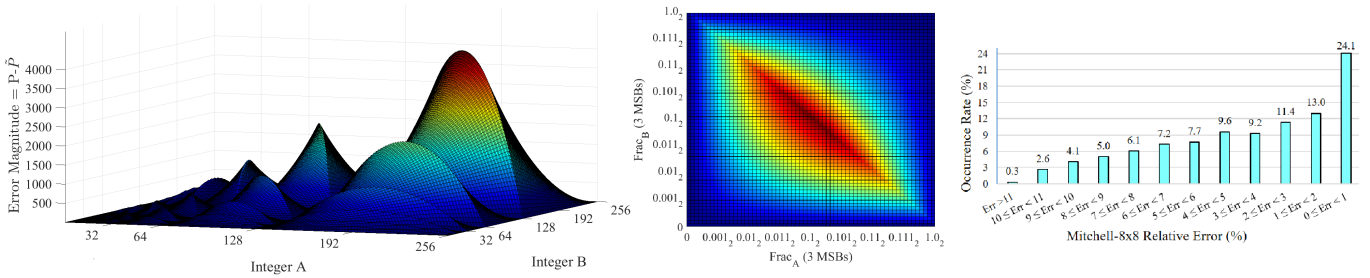
Designs in this branch exploit conversion of multiplication to addition in logarithmic-based representation, which considerably simplifies the circuit. Among the existing techniques, Mitchell's method has the best resource-efficiency in unsigned multipliers [17], albeit having a high error (11.11% peak and 3.85% mean relative error). It consists of four steps: finding the leading one as integer part, re-aligning rest of the bits as fractional part, addition of both parts, and finally shifting fractional part w.r.t integer part. In [27], large multipliers are recursively build upon proposed error-free of 2x2 Mitchell's multiplier. However, having high area and latency overheads (~67%) filters the proposed design out from pareto-optimal curves. Other works like [17] investigated piece-wise linear error-reduction schemes to calculate log and anti-log individually for each segment within power-of-two-interval. These methods are applied to each multiplier input separately, *neglecting magnitude of error after multiplication*. Tackling piece-wise approximation overhead, a newly proposed work (MBM) [18] has proposed a single error-correction term (for all input combinations) which is added once after computing summation of fractional part. Through showing *this single error-reduction term weakly fits all input combinations* (output overflow after adding error-reduction term), we propose three approaches that enable tunable/higher accuracy.

III. Proposed Approximate Multiplier

A. Mitchell's Multiplication thm

Consider the binary representation for N-bit unsigned input A which can be written as Eq. 1, where k reveals the position of the leading one. The rest of the bits (starting from position $k-1$ to 0) are considered as the fractional part which fall in the range $0 \leq x < 1$. In linear mathematics, $log_2(1+x)$ is approximated to x for this range, therefore the approximate log value of input A is shown in Eq. 2.

$$A = 2^k \sum_{i=0}^{k-1} 2^i b_i = 2^k(1+x) \Rightarrow 51 = 2^5(1+0.10011), 11 = 2^3(1+0.011) \quad (1)$$



(a) (b) (c)

Fig. 1: Mitchell's error: (a) side view of error magnitude (8x8 multiplier), (b): top view of error (same for any size of multiplier) (c): relative error distribution

$$Log_2(A) \cong k + x \quad \Rightarrow \quad 51 = 101.10011, 11 = 11.011 \tag{2}$$

With the same manner for input B, summation of two parts is obtained in Eq. 3. Finally the anti-log is applied and binary representation of approximate product is calculated in Eq. 4:

$$\widetilde{Log_2}(\tilde{p}) = k_1 + k_2 + x_1 + x_2 \Rightarrow \; K_s = 1000, \; X_s = 0.11111 \tag{3}$$

$$\tilde{P} = \begin{cases} 2^{k_1+k_2}(1 + x_1 + x_2), & x_1 + x_2 < 1 \\ 2^{k_1+k_2+1}(x_1 + x_2), & x_1 + x_2 \geq 1 \end{cases} \tag{4}$$

$$\Rightarrow \quad \tilde{P} = 111111000 = 504, \quad P_{accurate} = 561$$

B. LeAp: Proposed LUT-based Multiplier

1) Proposed approximate log calculator: In our proposed method, detection of leading one is augmented with simultaneously re-aligning fractional part. To this end, each input is divided into 4-bit segments and logical OR function is applied to the bits in each group to find segments having a bit with value '1'. Afterwards, we determine the position of leading one in the most significant group in at most 4 cycles. To this end, we exploit one of two slices in a Virtex-7 CLB consisting four 6-LUTs with associated fast carry chain and use the slice as a downward counter. For instance, consider 16-bit input '0000000110101010'. The leading one would be in position 8. We first detect that the leading one is located in the second segment from right hand side, and then load decimal value 11 into the registers connected to their dedicated carry chain. Afterwards, in each cycle, we count down the value of register and shift bits to the left by one bit until the leading one is found. The segmentation granularity enables a trade-off in latency-energy which could be deterministic parameter for designer in relatively-large multipliers. It should be noted that we analyzed 4 to 8-bit segmentation for inputs, for the sake of brevity we present result for 4-bit grouping which rendered the best energy-delay product.

2) Proposed Error-Reduction Schemes: Mitchell's error for 8x8 multiplier is plotted in Fig.1 based on Eq. 5 through which three points can be observed:

$$E_P = P - \tilde{P} = \begin{cases} 2^{k_1+k_2}(x_1 x_2), & x_1 + x_2 < 1 \\ 2^{k_1+k_2}(1 - x_1 - x_2 + x_1 x_2), & x_1 + x_2 \geq 1 \end{cases} \tag{5}$$

- Fig.1 (a) shows that magnitude of error is different in each power-of-two interval. Which is to say, a single correction term added to output will not fit for all multiplier sizes.
- Eq. 5 proves proportionally replication of error in each power-of-two. Hence, irrespective of k1/k2, a unique refinement scheme may fit all multiplier size and it can be added to fractional part before scaling to save more resources.
- Fig.1 (b) demonstrates that error is non-uniformly distributed in the whole interval, but it is nearly symmetrical: for most of input combinations, it tends to be near zero (at the beginning and end of each interval), and reach to its peak in the middle. Part (c) of this figure demonstrates that approximate products with highest relative error are significantly less than others (especially cases with less than 1% error), but their magnitude is multiple times of the average error throughout the whole interval, i.e., 0.0833 which is obtained from Eq. 6. Therefore, assigning merely a single reduction term (like [18]) to the whole interval is not efficient and results in output overflow.

$$Avg_{E_P} = \frac{1}{(1-0)(1-0)} \int_0^1 \int_0^1 E_p dx_2 dx_1 =$$
$$\left(2^{k_1+k_2+1} \int_0^1 \int_{1-x_1}^1 (x_1 x_2 - x_1 - x_2) dx_2 dx_1 + \right. \tag{6}$$
$$\left. 2^{k_1+k_2} \int_0^1 \int_0^{1-x-1} (x_1 x_2) dx_2 dx_1 \right) = 0.0833 \times 2^{k_1+k_2}$$

Insights obtained from these points and coping with overflow problem in MBM [18] incentivize using different error-reduction terms appropriately opted based on fractional part. Building upon these observations, we performed a design space exploration on many error-reduction schemes through which the squarish region between each power-of-two is effectively divided to multiple regions each of which having distinctive coefficient representing their approximate average error. In each scheme we attempt to optimize two factors: 1) Minimize $error\,distribution \times error\,magnitude$ in each region (can be estimated to integral of error-magnitude of the region). 2) Minimize resource overhead in our partitioning by only analyze 4 MSBs of fractional parts to select appropriate error-reduction term for each pair of input combination.

2-coefficient approach: This scheme targets an amended accuracy while enduring minimal overhead imposed by error-coefficient selection circuitry. To this end, we divided the main squarish region to two squares (Fig.2 (a)). For numbers bounded in the innermost zone with higher relative error we applied coefficient equal to 0.0938, obtained from measurement similar to Eq. 6 for this interval. Analogously, the outer-zone coefficient is obtained in the same manner. This partitioning is based on only checking at most four MSBs of fractional parts. Note that although the proposed partitioning is not optimum (as the highest errors are bounded in an rhombus shape), yet it is significantly rewarding in terms of accuracy with an endurable overhead. In contrast, implementation of checking rhombus borders in hardware is costly and will not further deliver considerable improvement in the overall average relative error. This is due to the fact that the number of high-magnitude errors reduces as we reach the middle of power-of-two (refer to Fig. 1 (c)). Therefore, we have elaborately opted the region margins that eventuates in best resource-error trade-off, obtained by trial and error from both analysis of behavioral in-depth error characterization in C++ and implementation reports from Vivado.

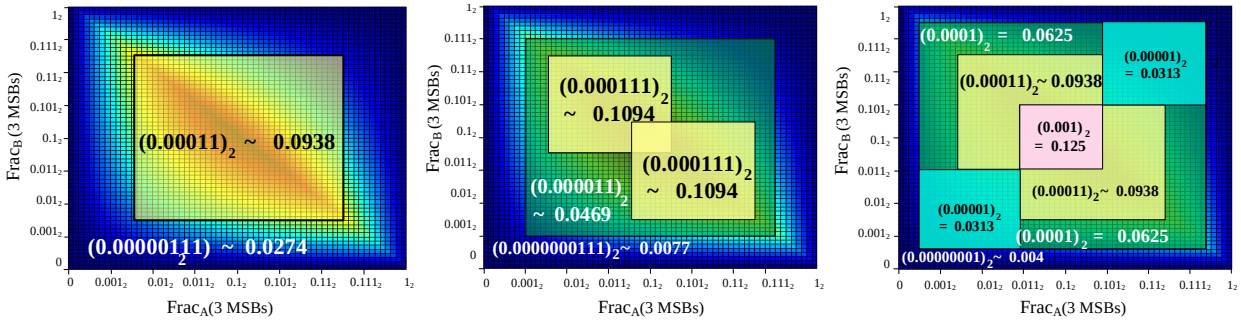


Fig. 2: Proposed error reduction schemes for all multiplier size, (a) 2-coefficient approach (b) 3-coefficient approach (c) 5-coefficient approach

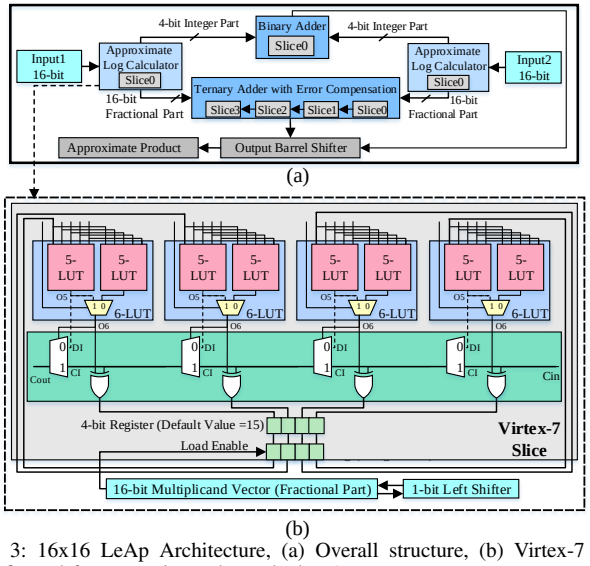


Fig. 3: 16x16 LeAp Architecture, (a) Overall structure, (b) Virtex-7 slice (configured for approximate log calculator)

3- and 5-coefficient approaches: Referring to Fig. 1 (b) the high-error zone is not squarish but still symmetrical. Therefore, to gain higher precision, we analyzed variants of partitioning with their associated resource usage, among those, the hardware implementation of partitioning illustrated in Fig. 2 (b) and (c) resulted in the smallest overhead and pruned other candidates (for sake of brevity, we have skipped 4-coefficient). Based on the application constraints, designer can further increase the number of coefficients. In both approaches still four MSBs of the fractional parts are analyzed but with a bit deeper conditional checking of borders. This is while in another work study [17], 4 up to 14 bits are checked which its penalty would thwart the gain. It is noteworthy to mention another accuracy drawback (beside output overflow) attributed to mechanism proposed in [18]: There are three cases that peak error is still equal to its counterpart in original Mitchell, 11.11% and seven others higher than 8%. These cases occur when summation of integer parts is small and therefore, some bits of error-reduction term will not be shifted to the final output. Authors neither measured these errors, nor presented a handling circuit, and simply suggested to put a comparator and avoid adding error-reduction term in these cases. In our schemes, such errors are alleviated by assigning appropriate coefficient in each scenario.

C. Hardware Implementation of LeAp

Fig. 3 (a) shows proposed multiplier design. As discussed in section III detection of leading one and re-aligning fractional parts are performed concurrently which utilizes one slice including four 6-LUTs with their associated carry logic, a 1-bit shifter and a register for saving the intermediate input (which is shifted to the left in each clock), as shown in Fig. 3 (b). Counting down the 4-bit register (needed in 16-bit LOD), is equal to adding it with binary "1111" and simply neglecting the overflow. In doing so, we connect the carry chain output (a 4-bit register with default value of 15) to the first inputs (I_0) of four LUTs in the slice and assign '1' to I_1 (Fig. 3(b)). LUTs are configured to implement truth-tables of carry-generate (O_5) and carry-propagate (O_6) functions, so the associated carry chain implements a 4-bit downward counter. Therefore, as soon as position of leading one is found, fractional part (which has been already shifted to left in each clock) is also re-aligned and located in the intermediate register. Afterwards, adding integer (4-bit) and fractional parts (15-bit) are implemented with CLA adders, fulfilled by one and four FPGA slices, respectively.

Another benefit of directly using LUT primitives from Xilinx UNISIM library [22] is that they can be configured in such a way that implement ternary adder which perfectly suits our proposed error-refinement method. *In this manner, we combined the process of adding error-reduction term with fractional parts within the same resources in a single step*. Regardless of adder size, only one more bit at MSB is needed in ternary addition compared to binary version, since $frac1_i$+$frac2_i$+error $term_i$+Cin (Cout of from previous bit) may result in 3 bits which necessitates one more LUT at the end of the chain. Therefore, in our proposed schemes no extra circuitry is needed for adding the error-reduction term. In addition, the delay of the LUT primitive is fixed and adding error-reduction term at the same time that fractional parts are added keeps the overall delay of the circuit nearly untouched (unlike [18] where addition is dependent on the carry out from summation of fractional part, which lengthens design bottleneck path). The only supplementary circuitry is for deciding which coefficient should be applied.

IV. Results and Discussion

A. Experimental Setup

Here, we evaluate the efficiency of LeAp compared with baseline and cutting-edge multipliers including area-optimized accurate multiplier (LogiCORE v.12) [5] provided by Xilinx Vivado 17.4, precision reduced multiplier that uses smaller accurate multiplier for MSB multiplication (Trunc) with similar area to LeAp, Mitchell (MA) [16], state-of-the-arts MBM [18], and [4] (16x16 LUT-based multiplier based on approximate 4x4 instances with accurate (CA) or approximate (CC) adder) Note that the rest of similar logarithmic multipliers have relatively higher resource consumption than MBM, therefore

TABLE II: Summary of design metrics in 16x16 multipliers

Approach	Area (6-LUT)	Throughput (/μS)	Power (mW)	Energy (μJ)	Avg Relative Error (%)	Peak Error (%)
Accurate [5]	280	203	31.5	328	-	-
CA [4]	245	201	30.4	341	0.3	19.04
CC [4]	240	416	32.7	306	14.88	96.7
Trunc12x12	139	221	24.1	242	0.2	100
MA [16] comb.	249	212	27.1	276	3.85	11.11
MA [16] seq.	121	138	15.4	264	3.85	11.11
MBM [18]	128	94	17.5	331	2.63	8.81
LeAp-2	132	238	21.6	224	1.56	6.67
LeAp-3	135	224	23	232	1.23	5.72
LeAp-5	139	217	24.7	246	0.98	4.76

TABLE III: Summary of design metrics in 32x32 multipliers

Approach	Area (LUT)	Throughput (/μS)	Power (mW)	Energy (/μJ)
Accurate [5]	1103	163	65.8	672
MA [16]	246	53	23.5	490
MBM [18]	258	42	28.2	635
LeAp-2	334	187	38.1	423
LeAp-3	337	174	40.8	437
LeAp-5	342	166	43.3	451

we omit comparing them again in this paper. To ensure a fair design, we implemented all multipliers in HDL, synthesized and implemented in Vivado 17.4 for Virtex-7 FPGA. Area, throughput, and power are directly reported from Vivado Simulations and Power Analyzer for the 100 Million inputs uniformly distributed in a random order over whole input interval. In addition, energy dissipation is measured based on the total execution time and its power consumed for all the inputs feeded to the multiplier. Note that metrics are reported separately since weighted product of resource-error is more of designer's preference and not by itself an appropriate figure of merit, i.e., it lacks distinctiveness and results in 0 for accurate multiplier [10]. In addition, to ensure scalability of LeAp, all Mitchell's based designs are implemented in 32-bit as well. The behavioral structure of multipliers are also developed in MATLAB, C++, and Python to calculate the error magnitude, average absolute relative error, and peak absolute relative error (referred to as relative and peak errors, respectively) for all possible multiplier inputs. Multipliers are also deployed in both Artificial Neural Network (ANN) and Gaussian Image Smoothing and Multiply-based Image Blending applications to test the effectiveness of LeAp in real world applications.

B. Evaluation and Characterization of Proposed Multipliers

Design metrics and error analysis are summarized in Table II and III, where x in "LeAp-x", denotes number of error-reduction coefficient. The following conclusions are notable:

- **LeAp vs. PP-based multiplication**: Comparing Mitchell-based multipliers with truncated and hierarchical-based counterparts, designed upon incorporating smaller inexact instances [4] justifies three points: 1) thanks to translation of multiplication to addition, area of Mitchell-based multipliers grows by factor of ~2.5 while compared to ~4 for others. This further highlights efficiency of LeAp in larger-width. 2) Although average error of Trunc12x12 is better than 16x16 LeAp, referring to previous point, in larger multipliers more LSBs need to be truncated, i.e., Trunc18x18 have same area to 32x32 LeAp. This will further deteriorate its accuracy and increase cases with large error near or equal to 100% (hundreds of millions), while in LeAp most of errors are $<1\%$. 3) approximation applied on hierarchical multipliers is rewarding in accuracy-resource trade-off only when it is done from scratch for each multiplier size, otherwise stockpiled error in larger designs significantly scarifies output accuracy to gain resource efficiency.
- **LeAp corroborates its superiority by improving resource consumption**: Augmenting Mitchell's algorithm with our proposed log calculator architecture, efficiently customized for FPGAs with novel error-reduction schemes, delivers improvement in all design metrics compared to the accurate multiplier while tolerating less than 1.6% average relative error (especially LeAp-2 saves area and energy by 52.8% and 28.7% in 16x16 design). This is while both CA [4] and MBM [18] dissipate even **more energy** (with lower throughput) than accurate multiplier. LeAp resource gains become even more pronounced in 32x32 multiplier (up to 69.7%, 14.7%, 42.1%, and 37.1% improvement in area, throughput, power, and energy, respectively). In addition, as it can be observed, increasing the number of coefficients can boost precision with a negligible cost, benefiting from the fact that coefficient selection only depends on few MSBs of fractional parts and one more error-coefficient would only increase the conditional statements by one. We capitalized most of extra LUTs in our design to implement the proposed fast log calculator which improves total execution time and also contribute to reduction of energy dissipation. As a result, LeAp has improved energy over all multipliers, including original Mitchell. Note that although error-reduction scheme proposed in MBM has small area and power compared to LeAp, its prolonged critical path and lack of fast approximate log calculator resulted in execution time and energy overhead.
- **The error-refinement approach outperforms the existing ones**: Compared to Mitchell's multiplier, both error metrics are enhanced by 74.5% for relative error and 57.1% in peak error for 16x16 multiplier. Note that exhaustive 32x32 test is massively time-consuming, however, unlike hierarchical-designs, Mitchell's avg-error does not significantly changes in larger multipliers. In particular, the proposed error-reduction scheme is independent from input size and outperforms MBM design in accuracy with respect to both error metrics (reduced to less than its half for relative error).

C. High-Level ANN & Image Processing Applications

To further evaluate the efficacy of LeAp in state-of-the-art multiplier-exhaustive applications, LeAp-5 and MBM [18] are deployed in the behavioral implementations of Image Blending application with '*Miscellaneous*' test images in USC-SIPI Database [28]. The average *Peak signal-to-noise ratio* (PSNR) value produced by LeAp-5 (32.19dB) is better than MBM (31.23dB). In another application (Gaussian Image Smoothing) LeAp also has surpassed MBM in PSNR (27.4dB over 24.9dB). Examples of visual quality for both applications are illustrated in Fig. 4 and Fig. 5.

We have also utilized approximate multipliers during the inference phase of an ANN [29] for classification of MNIST *Handwritten Digits* [30] and *Fashion* [31] datasets. Two different network configurations, i.e., two and three hidden layers have been tested. The total number of input nodes, hidden layers, and output layers are 28x28, 100, and 10 respectively. For both datasets, networks are trained with 50,000 images using double-precision floating point numbers with a batch size of 10. Subsequently, during the inference phase, the networks are evaluated using double-precision floating point

(a) Original image (b) Accurate (c) LeAp-5 (d) MBM [18]

Fig. 4: PSNR of Image blending application computed with respect to the accurate multiplier-based filter: LeAp-5 PSNR=33.5, MBM [18] PSNR=32.6

(a) Noise-induced (b) Accurate (c) LeAp-5 (d) MBM [18]

Fig. 5: PSNR of Gaussian noise removal filter computed to the original noise-free image: (a) 20.5 (b) Accurate=28.7 (c) LeAp-5=28.4 (d) MBM [18]=27.9

and 8-bit fixed-point numbers. Table IV shows the classification accuracy, area and energy gain by using approximate multipliers in 10,000 test images. As shown by the results, the error-resilience of ANN makes the requirement for an accurate multiplier trivial for classification accuracy. In fact, for the MNIST *fashion* dataset, many of the quantization errors have been masked by approximate multipliers. For MNIST *handwritten digits* dataset, accurate and approximate multipliers-based networks produce almost similar classification accuracy. Interestingly, LeAp achieves precision of accurate multiplier while outperforming in terms of area and energy by 22% and 16% (also surpasses MBM in terms of energy). Note that, referring to result in Table II, MBM will even have worse energy than accurate counterpart, provided that user exploits 16-bit multipliers in ANN.

TABLE IV: ANN metrics exploiting accurate and approximate multipliers

MNIST Dataset	No. of Hidden layers	Nodes in each Hidden layer	Inference Accuracy %			
			Double Precision	8-bit Fixed Precision		
			Accurate	Accurate	LeAp-5	MBM [18]
Digits	2	100	97.09	96.67	96.67	96.62
Digits	3	100	96.56	96.23	96.22	96.17
Fashion	2	100	85.18	84.07	84.07	84.08
Fashion	3	100	85.29	84.26	84.27	84.39
Area (normalized to 8-bit accurate)			-	1	0.82	0.78
Energy (normalized to 8-bit accurate)			-	1	0.68	0.84

V. Future Works and Conclusion

We proposed an approximate multiplier aimed at area, throughput, and energy-efficiency of soft multipliers in FPGAs while maintaining higher accuracy compared to the cutting-edge counterparts. As future tracks, we intend to assess the applicability of proposed multiplier in other domains, e.g. being utilized as mantissa multiplier for floating point numbers. Moreover, we target to enable online error-configurability using LSB truncation which can be supported in partial reconfiguration. The provision of altering accuracy in a higher level of abstraction allows the application developer to adjust the arbitrary output accuracy specifically for incoming inputs and meet the user budget. This is especially desirable for high-input applications such as neural networks since, first, it enables accuracy configurability of multipliers at the granularity of intra- or inter-layer. Second, energy of memory-level operations also will be improved as the size of data is reduced. Last but not least, we only considered accurate addition, nevertheless, approximate adders in the literature are orthogonal to the contributions of our proposed architecture and can be employed to add LSBs of fractional parts in Mitchell's algorithm in tandem with accurate ones for MSBs without imposing high level of inaccuracy.

References

[1] S. Jain *et al.*, "Compensated-DNN: energy efficient low-precision deep neural networks by compensating quantization errors," in *DAC, 2018*.

[2] P. Judd *et al.*, "Reduced-precision strategies for bounded memory in deep neural nets," *arXiv preprint*, 2015.

[3] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *TCAD, 2007*.

[4] S. Ullah *et al.*, "Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators," in *DAC, 2018*.

[5] Xilinx, "LogiCORE ip multiplier v12.0." [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf

[6] B. S. Prabakaran *et al.*, "DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems," in *DATE, 2018*.

[7] S. Ullah *et al.*, "SMApproxlib: library of FPGA-based approximate multipliers," in *DAC, 2018*.

[8] P. Kulkarni *et al.*, "Trading accuracy for power with an underdesigned multiplier architecture," in *VLSID, 2011*.

[9] C. Liu *et al.*, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *DATE, 2014*.

[10] S. Rehman *et al.*, "Architectural-space exploration of approximate multipliers," in *ICCAD, 2016*.

[11] V. Mrazek *et al.*, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *DATE, 2017*.

[12] K. Bhardwaj *et al.*, "Power-and area-efficient approximate wallace tree multiplier for error-resilient systems," in *ISQED, 2014*.

[13] S. Hashemi *et al.*, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *ICCD, 2015*.

[14] K. Y. Kyaw *et al.*, "Low-power high-speed multiplier for error-tolerant application," in *EDSSC, 2010*.

[15] S. Narayanamoorthy *et al.*, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *TVLSI, 2015*.

[16] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRETEC, 1962*.

[17] J. Y. L. Low and C. C. Jong, "Unified mitchell-based approximation for efficient logarithmic conversion circuit," *TC, 2015*.

[18] H. Saadat *et al.*, "Minimally biased multipliers for approximate integer and floating-point multiplication," *TCAD, 2018*.

[19] Y. Zhang *et al.*, "Hierarchical synthesis of approximate multiplier design for field-programmable gate arrays (FPGA)-csrmesh system," *IJCA, 2018*.

[20] M. Osta *et al.*, "Approximate multipliers based on inexact adders for energy efficient data processing," in *NGCAS, 2017*.

[21] M. Shafique *et al.*, "Cross-layer approximate computing: From logic to architectures," in *DAC, 2016*.

[22] Xilinx, "Xilinx 7 series FPGA programmable guide for HDL designs," 2013. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_hdl.pdf

[23] A. Ehliar, "Optimizing xilinx designs through primitive instantiation," in *FPGAworld, 2010*.

[24] U. Lotrič and P. Bulić, "Applicability of approximate multipliers in hardware neural networks," *Neurocomputing, 2012*.

[25] Y. O. M. Moctar *et al.*, "Reducing the cost of floating-point mantissa alignment and normalization in FPGAs," in *FPGA, 2012*.

[26] Z. Li *et al.*, "FPGA design and implementation of an improved 32-bit binary logarithm converter," in *NCWMC, 2008*.

[27] S. S. Bhairannawar *et al.*, "FPGA based recursive error free mitchell log multiplier for image filters," in *ICCIC, 2012*.

[28] "Sipi image database (2019)." [Online]. Available: http://sipi.usc.edu/database/database.php?volume=misc

[29] "MNIST-cnn." [Online]. Available: https://github.com/integeruser/MNIST-cnn

[30] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *ISPREG, 2012*.

[31] H. Xiao *et al.*, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR, 2017*.