DRUM: A Dynamic Range Unbiased Multiplier for Approximate Applications

Soheil Hashemi
School of Engineering
Brown University
Providence, Rhode Island 02912
Email: soheil_hashemi@brown.edu

R. Iris Bahar
School of Engineering
Brown University
Providence, Rhode Island 02912
Email: iris_bahar@brown.edu

Sherief Reda
School of Engineering
Brown University
Providence, Rhode Island 02912
Email: sherief_reda@brown.edu

***Abstract*—Many applications for signal processing, computer vision and machine learning show an inherent tolerance to some computational error. This error resilience can be exploited to trade off accuracy for savings in power consumption and design area. Since multiplication is an essential arithmetic operation for these applications, in this paper we focus specifically on this operation and propose a novel approximate multiplier with a dynamic range selection scheme. We design the multiplier to have an unbiased error distribution, which leads to lower computational errors in real applications because errors cancel each other out, rather than accumulate, as the multiplier is used repeatedly for a computation. Our approximate multiplier design is also scalable, enabling designers to parameterize it depending on their accuracy and power targets. Furthermore, our multiplier benefits from a reduction in propagation delay, which enables its use on the critical path. We theoretically analyze the error of our design as a function of its parameters and evaluate its performance for a number of applications in image processing, and machine classification. We demonstrate that our design can achieve power savings of 54% – 80%, while introducing bounded errors with a Gaussian distribution with near-zero average and standard deviations of 0.45% – 3.61%. We also report power savings of up to 58% when using the proposed design in applications. We show that our design significantly outperforms other approximate multipliers recently proposed in the literature.**

I. Introduction

In the recent years, power efficiency has emerged as one of the most critical goals of hardware design. Enormous efforts have already been devoted to improve the power consumption in various levels from algorithmic and software level down to circuit and transistor level. Emerging applications in digital signal processing, computer vision, and machine learning have created new power consumption challenges due to their high computational demands. At the same time, these applications also afford new opportunities for novel low-power implementations. In particular, the common feature of these application domains is an inherent tolerance for limited and insignificant inaccuracies. This error tolerance arises for multiple reasons, including imperfect perception in human sense, noisy input signal, redundancy in the input data or the absence of a universal best answer [1]. This inherent error tolerance can be exploited using approximate computing, which effectively allows designers to trade off computational accuracy for savings in power consumption and computational complexity.

One approach to realize approximate circuits is to use voltage over-scaling (VOS), where the supply voltage is reduced below the safe operating threshold in favor of saving power. This approach creates timing errors on the critical paths as the circuit slows down with lower voltages. Since the computations of the most significant bits are usually on the critical paths, VOS is likely to lead to large errors and potentially the circuit can enter a metastable state. A second approach is to design approximate circuits where the approximation error is controlled by construction. These approximations could be introduced into the major building blocks (e.g. arithmetic circuits) that are used in the circuit, or they could be introduced into the Boolean or high-level descriptions of arbitrary circuits [2]. Approximate arithmetic focuses on designing basic arithmetic units such as approximate adders and multipliers that can substitute or augment exact arithmetic units in programmable processors (e.g., CPUs and GPUs) or in custom circuits (e.g., ASICs and accelerators). In general, it is expected that approximate arithmetic will deliver higher power savings in custom circuits compared to CPUs, where data transfer and control instructions form the bulk of the computations.

In this paper we focus on approximate multiplier design. Multipliers are particularly attractive to approximate due to their high cost in terms of power and silicon area, and their prominent utilization in the aforementioned emerging application domains. Our contributions are as follows.

- We present a novel Dynamic Range Unbiased Multiplier (DRUM) for approximate applications. The dynamic nature of our design enables the multiplier to "zoom in" on the most significant digits of the numbers, all while making the multiplier highly scalable and configurable. Since a typical computation involves a large number of multiplications, the unbiased nature of our multiplier leads to near-zero average error. As a result, our unbiased design promotes errors to cancel each other rather than accumulate in the final result of the computation.
- The configurable aspect of our design enables smooth trade-off between accuracy loss and power consumption based on the designer's priorities. Furthermore, our approximate design utilizes a core accurate multiplier that is smaller in size than the full multiplier. Thus, designers can still use their preferred multiplier design for the core component with no need for sacrificing typical design practices.

978-1-4673-8388-2/15/$31.00 ©2015 IEEE

- We present an analytical formulation for the introduced maximum and average error as a function of the parameters that determined the multiplier configuration. We evaluate the multiplier's power consumption, area and timing improvements using a 65 nm library. We analyze the inaccuracy and power savings as a function of the size of input numbers and the configuration of the multiplier.
- We evaluate the performance of the DRUM architecture on three different applications: image filtering, JPEG compression, and a perceptron classifier. We implement our applications in Verilog and report the area and power savings achieved when utilizing the proposed multiplier. We demonstrate that the proposed design has negligible impact on application quality, while delivering large power savings reaching up to 58%.
- To demonstrate superiority over existing approaches, we compare the performance of our approximate multiplier against two established approximate multiplier designs in the literature.

The rest of the paper is organized as follows. First, we review some of the previous work in approximate computing and approximate arithmetic in Section II. In Section III, we describe the proposed approximate multiplier and elucidate its error analysis and scalability. In Subsection IV we analyze the accuracy, area and power of our proposed DRUM design, and then in subsection IV-B we evaluate the accuracy and power of three applications that leverage our DRUM multiplier. Finally, we summarize our conclusions in Section V.

II. Previous Work

In this section, we describe some prior research that focuses on approximate multiplier design as they relate to our proposed approach in approximate computing. Gupta *et al.* proposed several approximate adder designs that, by removing some of the logic used in a traditional mirror adder, achieved improved power, area, and performance [3]. Mahdiani *et al.* propose a bio-inspired approach, where the addition results within the multiplier were approximated by using OR gates for the lower part of the inputs [4]. At the same time, new metrics and methodologies are proposed in [5], [6], [7] for evaluation and modeling of approximate adders.

Babic *et al.* [8] proposed a pipelined log-based approximation where the classical Mitchell multiplier [9] is utilized in an iterative approach to improve its accuracy. An error-tolerant multiplier design is proposed in [10] where the multiplication is divided into accurate (multiplication based) and inaccurate (non-multiplication based) parts. Liu *et al.* [11] proposed an approximate multiplier with configurable error recovery using fast approximate adders for the partial product addition stage. Kulkarni *et al.* proposed an under-designed 2×2 approximate multiplier block to generate partial products [12], where larger multipliers are built using the inaccurate block to calculate the partial products which are then added together to generate the final result. Narayanamoorthy *et al.* proposed a static truncation based approach where the higher, middle, or lower portions of the inputs are used to approximately calculate the result [13]. A limitation of these last two aforementioned approaches is that they do not scale to higher input widths easily and their benefits reduce notably as the input size grows. In Section IV of this paper, we evaluate our design thoroughly against these two approximate multipliers [12], [13].

As an alternative to changing the underlying hardware design, another possible approach is to use voltage overscaling (VOS) [14], [15], [16], [17]. While VOS provides a flexible approach to using the same multiplier in either exact or inaccurate modes, it usually affects the critical paths of the circuit. These critical paths typically compute the most significant bits of arithmetic operations, and as a result VOS often leads to significant errors. Depending on the timing error and the clock signal, VOS can lead the circuit into a metastable state.

While most of the focus has been on the design of approximate arithmetic units, some recent efforts have been devoted to developing approximate logic synthesis [2], [18], [19], where techniques are proposed for automated synthesis of circuits into approximate hardware design. These works can naturally leverage approximate arithmetic operators for their use within the synthesis algorithm, and thus our proposed approximate multiplier is useful for such approximate synthesis tools.

III. Approximate Multiplier

A. Multiplier Design

Our proposed approximate multiplier exploits the fact that not all bits of a number are equally important. Thus, we propose to limit the number of bits applied to the multiplier by carefully selecting a range of bits for each of the two operands of the multipliers. Therefore, in hardware cost, our approach reduces a large multiplier to a significantly smaller core multiplier and some steering logic that is required to detect and route the selected range of bits of each of the operands to the smaller core multiplier.

In our design, we propose a dynamic and fast bit selection scheme to reduce the size of the multiplier while introducing a bounded unbiased error to the multiplication result. Assuming each operand has n bits, our design uses two leading one detector (LOD) circuit blocks to dynamically locate the most significant '1' in each of the two operands as illustrated in Figure 1.a. For each operand, the location of the most significant '1' is then used to select the following $k-2$ consecutive number of bits based on the required accuracy. Here, k is a designer-defined value which specifies the bandwidth used in the core accurate multiplier. To reduce the error, we

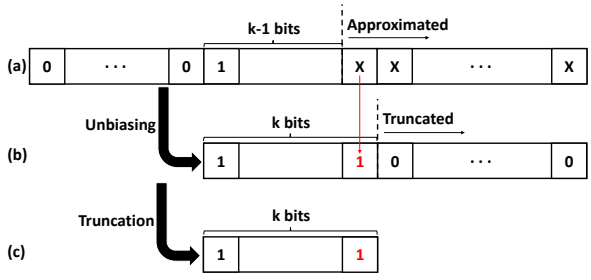

Fig. 1. A general example of the approximation process. (a) Original number. (b) Number after unbiasing. (c) Final approximated input.

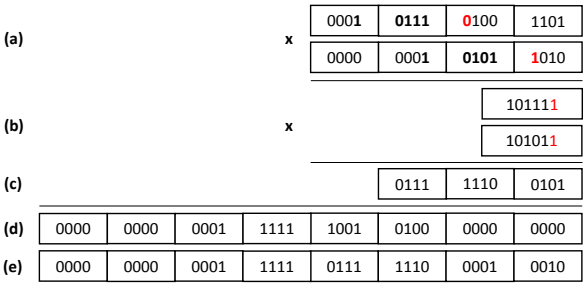

Fig. 2. A Numeric example of the proposed DRUM multiplier. ($n = 16$, $k = 6$) (a) The input numbers. (b) approximated inputs. (c) Result before shifting. (d) Approximate result. (e) Accurate result.

approximate the value of the remaining lower bits to equal its expected value. If the leading one is detected at index t, where $0 \leq t \leq n-1$, then the value of the remaining $t-k+2$ lower bits is between 0 and $2^{t-k+2}-1$. Assuming a uniform distribution for the values of the operands, then the expected value is almost equal to 2^{t-k+1}. Thus, we unbias the approximation of the last $t-k+2$ bits of the number with '1' placed at bit location $t-k+1$ (the highest bit) and zeros for the rest of the indexes from index $t-k$ down to index 0 as illustrated in Figure 1.b. Finally, these least $t-k+1$ zero bits are truncated leading to k-bits as illustrated in Figure 1.c. These k-bit operands are then forwarded to the inputs of a $k \times k$ accurate multiplier. The result of this multiplication will then be shifted, using a barrel shifter, according to the location of the two leading ones to generate the final approximate result.

Figure 2 illustrates a numerical example of the multiplication result in the proposed multiplier. Note that in this particular example, the unbiasing results in a significant reduction in the error and leads to a relative error of 0.27% compared to the biased multiplier with the same number of bits where the relative error is 1.86%. Here the biased multiplier is assumed to truncate the lower bits without changing the value of the highest bit to '1'. In other words, the lower bits are approximated by 0.

The proposed hardware design for our approximate multiplier is shown in Figure 3. The hardware design is composed of two parts: a steering logic and an arithmetic logic. The steering logic is responsible for dynamically selecting the right range of the input operands and feeding them to the arithmetic logic, and afterwards positioning the calculated result from the multiplier in the right index. Therefore the steering logic is compromised of LOD blocks, encoders, multiplexers, and a barrel shifter, while the arithmetic logic is conveniently a small multiplier. Here we use a Wallace-Tree multiplier as the core multiplier due to its higher utilization in applications. One key benefit of our architecture is that it does not restrict designers from using their own preferred designs as the smaller core multiplier. The significantly smaller arithmetic logic justifies the addition of steering logic, which overall leads to significant power and area savings compared to an accurate multiplier. In the special case where the leading '1' is within the least significant k bits, then our steering logic forwards the least k bits directly to the multiplier. This case is not illustrated in Figure 3 to avoid overcrowding the schematic, but it is implemented and evaluated as part of our design.

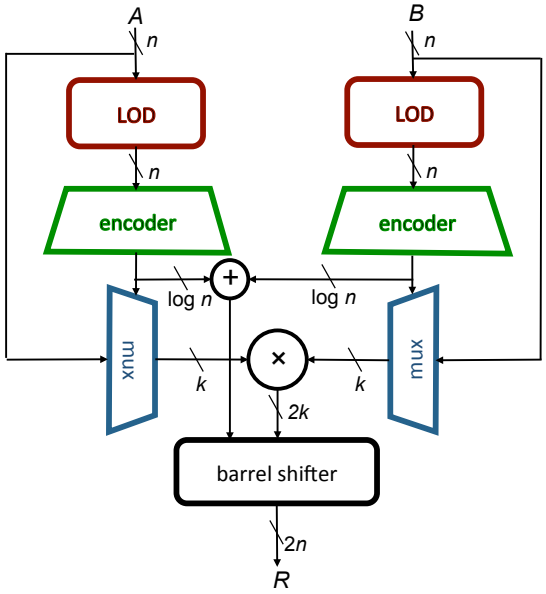

Fig. 3. The simplified schematics for the proposed approximate multiplier.

Handling of signed numbers: Due to higher utilization of unsigned multiplication in applications most amenable to approximation, i.e. image processing and machine vision, proposed arithmetic units in approximate computing have been largely focused on unsigned multiplication. Our proposed approximate design can support signed operation with a straightforward extension. For this purpose, preprocessing logic can be added in order to convert the signed operands to unsigned inputs using twos complement before forwarding them to an unsigned approximate block. The sign signal for the result is then calculated separately and the output will be negated if necessary. This scheme adds small area and power overheads which we quantify in Section IV. It should be deployed if necessitated by the application.

B. Error Analysis

In this subsection, we analytically calculate the expected error, and maximum error values as functions of the number of bits, n, of each operand, and number of bits, k, fed to the accurate multiplier. Given the lack of space to provide the full proof, we focus on providing and explaining the analytical results. Our analytical proofs were validated numerically against simulation results of the multiplier.

Given an operand, we provide in Figure 4 the meaning of the various symbols used in the analysis: t is the index of the leading one of the operand, U denotes the value of the consecutive $k-2$ bits that are subsequent to the leading one, and L is value of the remaining bits excluding the bit at position $t-k+1$. Before providing the multiplication error, we first consider the individual operands and formulate the truncation error. One can prove that for a random operand, A, approximated as discussed is previous subsection, the introduced error relative to the number is given by:

$$\text{Err}(A) = \begin{cases} X(U,L,t) = \frac{L}{2^t + (2U+1).2^{t-k+1}+L} & \text{if } A[t-k+1] = 1 \\ Y(U,L,t) = \frac{L - 2^{t-k+1}}{2^t + 2U.2^{t-k+1}+L} & \text{if } A[t-k+1] = 0 \end{cases}$$

In order to formalize the maximum error, we have to consider both the maximum positive error (MPE) and maximum negative error (MNE). Based on the equation, the MPE happens when L, and therefore t, is maximized and U is minimized. Thus, the MPE is equal to

$$\text{MPE} = \frac{2^{n-k}-1}{2^{n-1}+2^{n-k}-1}, \tag{1}$$

while the MNE happens for minimum values of L, U and t, and therefore it is equal to

$$\text{MNE} = 2^{-k+1}. \tag{2}$$

Using Equation (1) and Equation (2), the maximum truncation error (MTE) is given by Equation (3):

$$\text{MTE} = \max\{\text{MPE}, \text{MNE}\}. \tag{3}$$

Equation (4) gives the expected truncation error as a function of the input size, n, and and the number of bits used in the accurate multiplier k. We assume a uniform distribution on all of the operand numbers.

$$E[\text{Err}(A)] = \frac{1}{2^n}\sum_{t=k}^{n-1}\sum_{L=0}^{2^{t-k+1}-1}\sum_{U=0}^{2^{k-2}-1}(X(U,L,t)+Y(U,L,t)) \tag{4}$$

If the input operands, A and B, are independent, the multiplication error can now be characterized in terms of truncation errors. Therefore, for maximum multiplication error (MME):

$$\text{MME} = \begin{cases} 2\text{MPE} - \text{MPE}^2 & \text{if MPE}>\text{MNE} \\ 2\text{MNE} + \text{MNE}^2 & \text{if MPE}<\text{MNE} \end{cases}$$

As an example, for the maximum error for the proposed multiplier with parameters $n = 16$ and $k = 6$, using Equations (1) and (2), we get MPE $= 3.027\%$ and MNE $= 3.125\%$. This will result in MTE $= 3.125\%$ and therefore MME $= 6.34\%$. The expected multiplication error is given by

$$\begin{aligned} E[\text{Err}(AB)] = {} & E[\text{Err}(A)] + E[\text{Err}(B)] \\ & + E[\text{Err}(A)].E[\text{Err}(B)]. \end{aligned} \tag{5}$$

Figure 5 plots the maximum error, the expected error, and the expected absolute error as a function of k as derived by the equations. In Subsection IV-A we confirm the validity of our analyses against the experimental results.

C. Multiplier Scalability

While most of the available approximate multipliers lose their benefits as the input size grows, the dynamic nature of our approach makes the proposed DRUM design easily scalable. In the proposed design, while the steering logic grows with the input size, the arithmetic logic does not need to grow to higher bandwidths to maintain computational accuracy, which reduces the increase in the area and power costs, making the achieved benefits more substantial.

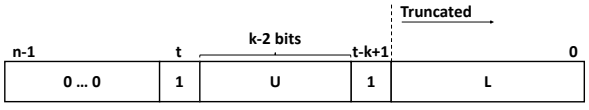

Fig. 4. The general case as used in the analytical discussion.

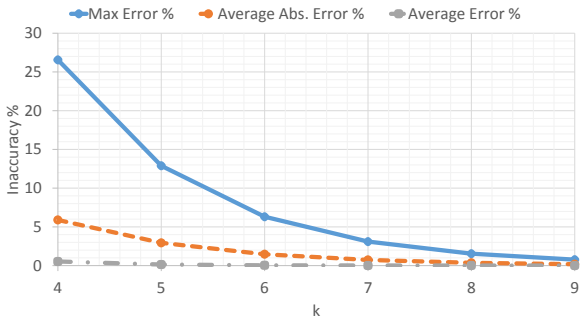

Fig. 5. Maximum Error, Average Absolute Error, and Average Error as given by the equations for different values of k.

TABLE I. AREA GROWTH AS A FUNCTION OF n AND k FOR THE PROPOSED DESIGN AND TWO ALTERNATIVES FOR COMPARISON.

Multiplier	Steering Logic	Arithmetic Logic	m & k
DRUM(k,n)	$O(n \log n)$	$O(k^2)$	$k \sim Const.$
SSM (m,n) [13]	$O(m)$	$O(m^2)$	$m \geq n/2$
Kulkarni (n) [12]	-	$O(n^2)$	-

Table I summarizes the relationship between the steering and the arithmetic logic area and the size of the input using $O(\cdot)$ notation for the proposed design as well as two previous approximate multipliers [12], [13]. Note that we are only considering the dominant factor in the table and consider the other factors to only affect the coefficients. We later evaluate the power consumption and design area exactly in Section IV. While the steering logic in our design might appear to be growing faster compared to the design proposed in [13], it is critical to note that for the SSM design to perform in acceptable range of accuracy, the value of m needs to be bigger than $n/2$. Therefore the value of m grows linearly as input size increases. Our proposed design, however, can achieve arbitrarily accurate output results with small and bounded changes of k. For instance, we will show in section IV that for multipliers with input sizes of 16, 24, and 32, the value of k does not need to increase in order to maintain the accuracy. Therefore, the overall growth in area of our proposed design will be increasingly smaller compared to SSM [13]. In comparison, the area of the design proposed in [12] grows quadratically as input size increases, making it less scalable.

IV. EXPERIMENTAL RESULTS

In this section, we empirically evaluate the proposed DRUM design in terms of computational accuracy as well as power and area performance. For comparison, we also implement and evaluate two recent and well-recognized approximate multiplier designs in the literature [12], [13]. All designs are coded in Verilog and synthesized using Synopsys DC compiler with an industrial 65-nm standard cell library at the typical process corner. We use high effort compiler options to optimize for area. We also use MentorGraphics ModelSim to evaluate the numerical output of the designs. Here, we first evaluate in Subsection IV-A the accuracy, power consumption, and design area characteristics of a standalone multiplier and continue in Subsection IV-B to evaluate the multiplier when used in applications.

A. *Stand-Alone Multiplier Results*

Our DRUM multiplier can produce arbitrarily accurate results by trading accuracy for power and area savings depending on the number of bits, k, chosen for its range. Thus, k is a design-time parameter that is determined depending on the acceptable overall design requirements. Note that the number of range bits k can vary from 1 to n, which makes the range of achievable accuracy using our method exceptionally wide, enabling the designer to limit the acceptable error rate as needed. In this subsection, the multiplier's computational accuracy results are evaluated against a set of two one-billion randomly generated input vectors and errors are reported relative to the results of an accurate multiplier.

We first consider $n = 16$ for a 16-bit multiplier and examine the impact of changing the range as controlled by k. Table II summarizes the results by providing the maximum error, average absolute error, average error and standard deviation of the error distribution calculated with respect to the accurate multiplier. The results show the expected trend that the error decreases as a function of k; furthermore, the reduction is exponential in k, where the errors are halved for every additional 1 bit in k as expected from our analysis in Section III-B. The plot in Figure 6 gives the achieved savings in design area and total power compared to the accurate multiplier, where the total power is the sum of dynamic and leakage power. For instance, we can save up to 71% of the multiplier power for the DRUM6 design (i.e. DRUM with $k = 6$), which has a near-zero average error distribution with a standard deviation error of 1.8%. Figure 7 shows the fitted Gaussian error distribution of the proposed multiplier for different values of k. Our unbiased error distribution will compensate for some of the error when utilized within real applications as will be demonstrated in Subsection IV-B. We verified that the experimental results confirm the proposed analytical derivations in Subsection III-B.

We next consider the impact of changing the multiplier size. Table III summarizes the accuracy results for the case of $n = 16$, 24 and 32. For all designs, we fix $k = 6$. As expected,

TABLE II. ACCURACY RESULTS FOR DIFFERENT k ($n = 16$).

	range					
	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
max error %	56.25	26.56	12.86	6.31	3.1	1.54
avg. abs error %	11.90	5.89	2.94	1.47	0.73	0.37
error bias %	2.08	0.53	-0.14	-0.04	0.01	0.01
std dev %	14.75	7.26	3.61	1.80	0.90	0.45

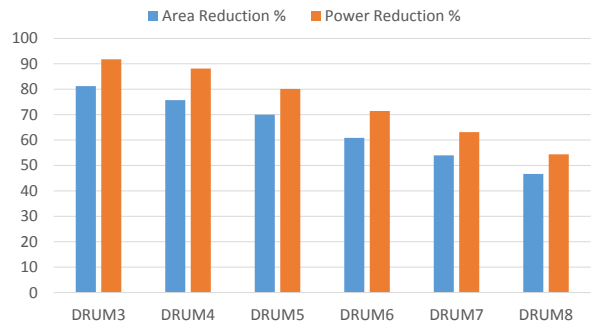

Fig. 6. Total power and area savings as a function of k. ($n = 16$)

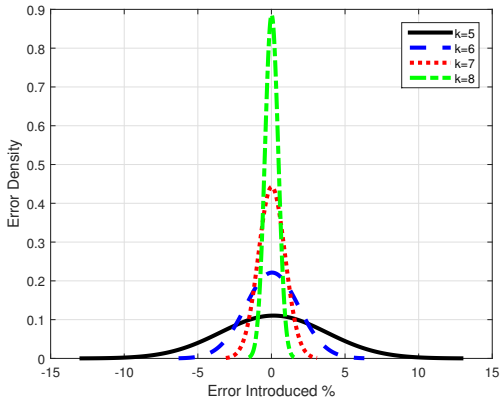

Fig. 7. The fitted Gaussian error distributions for the proposed multiplier for different k ($n = 16$).

TABLE III. ACCURACY FOR DIFFERENT INPUT SIZE ($k = 6$).

	multiplier size		
	$n = 16$	$n = 24$	$n = 32$
Max Error %	6.31	6.31	6.31
Average ABS Error %	1.466	1.467	1.467
Error Bias %	-0.043	-0.033	-0.033
std %	1.803	1.803	1.803

the results show that the error characteristics of DRUM do not change as a function of the multiplier size, because our design utilizes a dynamic range that always zooms on the first k bits starting at the position of the leading one. This feature enables our design to have an enormous scalability advantage over other approximate multipliers with static ranges.

Figure 8 gives the area and total power savings of the DRUM design for the three different input widths for both unsigned and signed implementations. We observe that the benefits of the proposed dynamic design grow almost linearly with input size, while keeping the accuracy within acceptable range. However, accuracy can be improved by adjusting the value of k to higher values. We also observe that the signed multiplier has reduced power savings over the unsigned version (e.g., 59% vs. 71% for n=16).

We compare the error characteristics for our DRUM6 design against two other approximate multipliers proposed in [12] and [13] and summarize the results in Table IV and provide the entire statistical distribution of errors in Figure 9. Our error distribution closely follows a Gaussian distribution with tight bounds for maximum error. In addition, the proposed

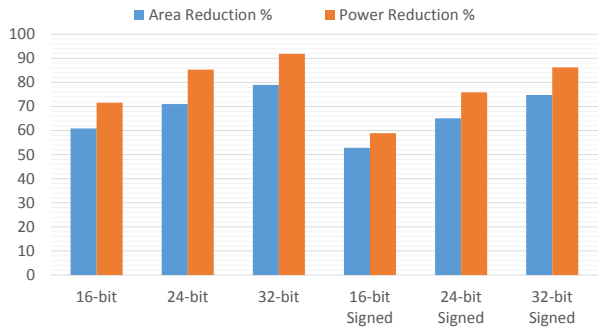

Fig. 8. Area and power savings as a function of input size for signed and unsigned multiplier (k=6).

TABLE IV. ERROR COMPARISON OF THE PROPOSED MULTIPLIER TO OTHER PUBLISHED WORK.

multiplier design	max error %	average abs. error %	average error %	std %
DRUM6	6.31	1.47	-0.043	1.8
ESSM8 [13]	11.07	1.14	1.14	0.96
Kulkarni [12]	22.22	3.32	3.32	7.93

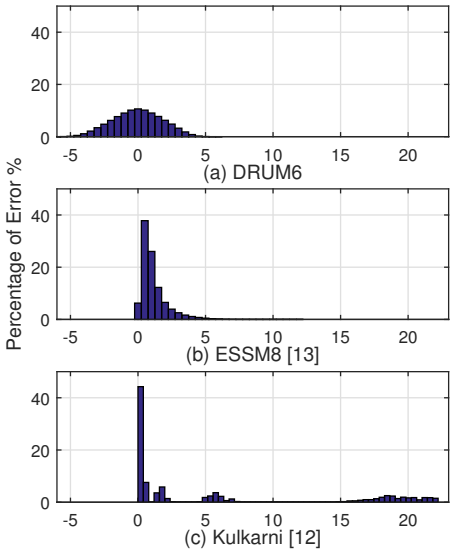

Fig. 9. The error distribution comparison for different multipliers.

multiplier has a relative absolute average error of 1.47% and an average error of -0.04% demonstrating the characteristics of an unbiased multiplier. The biased errors of the proposed approximate multipliers in the literature will lead to significant deterioration in performance in applications compared to our unbiased design. Furthermore, DRUM6 approximates each of the two inputs with its leading 6 bits and therefore guarantees a maximum error of 3.125% for each of the inputs and therefore a bounded maximum error of 6.31% in the result.

The power consumption, design area, and critical path delay of the DRUM6 multiplier are compared to other multipliers proposed by [12] and [13] in Table V. We observe that our power improvement is practically equal to ESSM8 [13] and much higher than Kulkarni [12], while delivering better area savings and timing. Furthermore, our power and area savings are realized with much higher accuracy as shown earlier in Table IV. The proposed DRUM multiplier also benefits from a shorter critical path. While almost all of the steering logic is on the critical path, as the design uses a quadratically smaller, hence faster, multiplier, the final critical path of the designs demonstrate an average of $1.7\times$ speedup over a traditional Wallace tree multiplier. In addition, as discussed in section III-A, approximating the inputs with their expected value will bias the error distribution very close to zero, which will lead to superior results when evaluated in the context of applications.

TABLE V. DESIGN COMPARISON OF THE PROPOSED MULTIPLIER TO OTHER PUBLISHED WORK.

Multiplier Design	Area (um^2)	Power (mW)	Area Savings	Power Savings	Critical Path (ns)
Accurate	2165	1.04	-	-	3.61
DRUM6	649.4	0.296	70%	71.45%	1.91
ESSM8 [13]	782.2	0.285	63.87%	72.52%	2.09
Kulkarni [12]	1955.2	0.823	9.69%	20.74%	3.53

In real application, as we will demonstrate in section IV-B, several of these approximate multiplication results are added in order to generate the final result, and therefore results from an unbiased approximate multiplier can potentially cancel some of the introduced error out, which will result in significant improvements in the application accuracy at no extra cost.

B. Application Results

In this section, we evaluate our DRUM design when used with three different applications from the domains of computer vision, image processing and data classification. The applications are image filtering, JPEG compression and a perceptron classifier. These applications are chosen due to their inherent tolerance to computational inaccuracies. Within the contexts of these applications, DRUM can be deployed in two ways. It could be either part of a general-purpose processor, where the software application invokes DRUM for executing multiplications, or DRUM could be used within custom circuits that execute the applications as accelerators in an ASIC or FPGA. Here, we choose the latter for evaluation of our proposed multiplier. We use Verilog to implement the applications in hardware and use DC compiler for synthesis. For each application, we use the lowest clock frequency for which the timing requirements are met and we make sure that the design frequency is kept constant for accurate and approximate designs. All of the accuracy results are based on fixed point simulations of the multipliers within their applications in MATLAB. The results are then compared against the approximate multipliers proposed in [12], [13].

1) Image Filtering: The first evaluated application is a Gaussian-based image smoothing. We use the standard 7×7 Gaussian kernel that is convolved with the image. The kernel uses 16-bit fixed-point arithmetic. The input image is a 200×200 greyscale image with 16-bit pixels.

Figure 10.a shows the original image, the resulting image from using the accurate multiplier in Figure 10.b, and the results from using DRUM for different values of $k = 3, 4, 5, 6$, and 7 in Figures 10.c to 10.f respectively. These figures highlight the significance of availability of a wide range in accuracy enabling the designer to choose the design based on their acceptable accuracy threshold. Note that, in this application, we calculate the SNRs with respect to a reference image filtered using the accurate multiplication.

Table VI compares the accuracy result of the DRUM6 design against the approximate multiplier design ESSM8 [13], and the under-designed multiplier by Kulkarni [12]. DRUM6 is selected here, for a fair comparison as it is within the same power budget as ESSM8 [13]. As it can be seen in the table, the near-zero bias of the error distribution in the proposed design leads to superior accuracy results compared to other truncate-based approximate designs at no extra cost leading to improvements higher than $5\times$ in linear scale.

2) JPEG Compression: As a more complex application, we test our multiplier in the JPEG compression algorithm. We replace the accurate multiplication in the algorithm with our proposed DRUM6 approximate multiplier and evaluate it as a function of the number of coefficients used in the

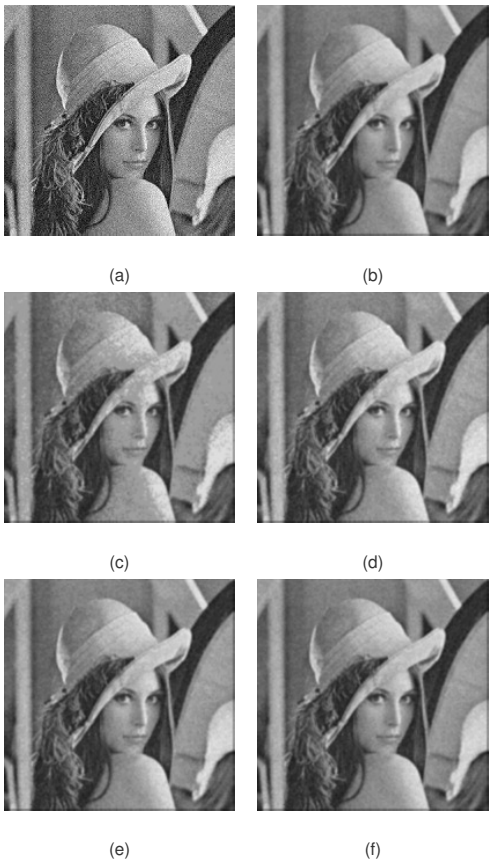

Fig. 10. Gaussian filtering results for different values of k. (a) Input image; (b) Filtered with accurate multiplier; (c) $k = 3$, $SNR = 51.13$ dB; (d) $k = 4$, $SNR = 53.38$ dB; (e) $k = 5$, $SNR = 67.26$ dB; (f) $k = 6$, $SNR = 91.04$ dB.

TABLE VI. GAUSSIAN FILTERING SNR COMPARISON FOR DIFFERENT MULTIPLIERS.

Approximate Multiplier	SNR (dB)
DRUM6	91.04
ESSM8 [13]	78.08
Kulkarni [12]	54.91

compression algorithm. Figure 11 shows the trend of Signal to Noise Ratio for the proposed multiplier as well as an accurate multiplier and [12], [13] for comparison. It is worth noting that as the number of coefficients increases, the number of used multiplications increases as well.

Table VII showcases the resulting SNR values when using 20 coefficients in the compression. As shown in the table, the degradation of the image quality in case of DRUM6 is very small, making the difference in the image barely noticeable in the output. Figure 12 compares visually the result of the proposed design to an accurate multiplier on a test image.

The JPEG compression results confirm the claim that in real applications the average error (error bias) is far more relevant than the average absolute error and therefore our proposed multiplier, having a near-zero average error, will outperform biased multipliers as demonstrated.

3) Perceptron Classifier: We evaluate our multiplier against a perceptron classifier. Perceptron classifiers are widely used in machine learning applications. In our work, we evaluate the classifier against 1000 two-dimensional points that belong

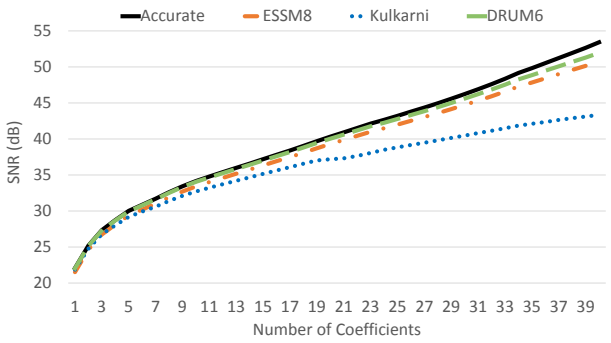

Fig. 11. Comparison of the jpeg compression results for the proposed multiplier vs multipliers as suggested in [12], [13].

TABLE VII. JPEG COMPRESSION ACCURACY FOR DIFFERENT MULTIPLIER DESIGNS.

Multiplier Design	Inaccurate SNR (dB)	Accurate SNR (dB)	SNR Reduction
DRUM6	40.04	40.32	0.69%
ESSM8 [13]	39.80		1.28%
Kulkarni [12]	37.19		7.76%

Fig. 12. JPEG compression algorithm. (a) Compressed using accurate multiplier, $SNR = 40.32dB$; (b) Compressed using DRUM6, $SNR = 40.04dB$.

to two classes {-1,1}. The inaccurate multipliers are used to replace the multiplier multiplying the perceptron weights and input points. We define the error rate as the percentage of mismatch between assigned class and the ground truth. The error rates (ER) are calculated against the true classes as assigned to the points. Figure 13 demonstrates the comparison of the classification problem with accurate multiplier and the proposed DRUM6 design.

When compared to the accurate multiplier, the proposed

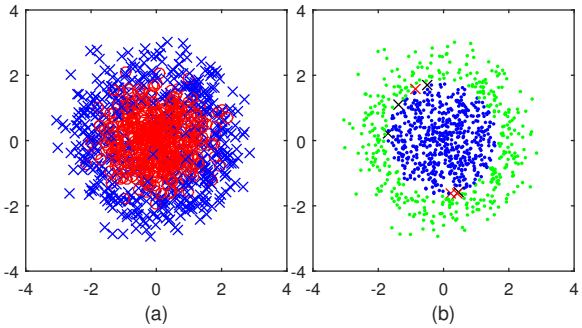

Fig. 13. Classification Application. (a) The input data set of classes -1,1. (red=1); (b) The outputs of accurate and approximate multipliers. (dots:matching classification, crosses:mismatch. red:Additional detection, black:False alarm.)

TABLE VIII. PERCEPTRON CLASSIFIER ERROR RATE RESULTS.

Multiplier	ER %
Accurate	15.0
DRUM6	15.1
ESSM8 [13]	15.2
Kulkarni [12]	51.7

TABLE IX. DESIGN AREA AND POWER SAVINGS FOR APPLICATION IMPLEMENTATIONS.

Application	Accurate Design		Approximate Design		Savings	
	area (um^2)	comb. power (mW)	area (um^2)	comb. power (mW)	area (%)	power (%)
Image Filtering	253982	15.55	186964	6.48	26.4	58.3
JPEG Compression	1862116	14.11	1357863	10.97	27.1	22.3
Perceptron Classifier	25022	2.24	19786	1.00	20.9	55.3

DRUM multiplier classifies four points, from the 1000 points in training set, in class 1 by mistake, and it detects three more points compared to the "accurate" design. It is important to note that even the design that uses the accurate multipliers cannot classify all points correctly. The comparison of the results with [12], [13] are presented in Table VIII.

Application power results: We also evaluate the power and area savings achieved when using the proposed multiplier within our three applications implemented in hardware as a custom circuit. Table IX summarizes the design area and power characteristics of the applications with and without utilization of the DRUM design. Here we only report the power consumption of the total combinational logic; the memory units are excluded from the analysis. For all of the three approximate hardware implementations we use DRUM with $k = 6$ and use 16×16, 32×32, and 16×32 input widths for image filtering, JPEG compression and perceptron classifier respectively. The power and area results show that the benefits of using the approximate multiplier depend on the proportional utilization of the multipliers against other components. In the cases of image filtering and perceptron classifier, since the multipliers comprise a substantial part of the hardware implementation, the achieved savings are over 50%, where as for JPEG compression, power savings are about 22%.

V. CONCLUSION

In this paper we proposed a Dynamic Range Unbiased Multiplier (DRUM) for approximate applications. Our design is highly scalable, enabling designers to configure it to trade power and accuracy as desired. Furthermore, its unbiased nature leads to reduction of errors when used for computations within applications. We analyzed DRUM's error and power consumption characteristics both analytically and empirically. With a bounded Gaussian error distribution with near-zero average and standard deviations of 0.45% – 3.61% and power savings of 54.38% – 80.13% for a subset of possible configurations, the DRUM design demonstrates great flexibility for trading accuracy for power. We also evaluated the design against three different applications in a comprehensive study. We demonstrated that the quality of application is not notably affected and the degradation is negligible. We also compared the DRUM design to two other recently proposed approximate multipliers and demonstrated superior performance. For example, on the image filtering application, with an application SNR of 91 dB, our design achieves power savings of 58%.

ACKNOWLEDGMENT

This research is supported by NSF grant 1420864.

REFERENCES

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

[2] K. Nepal, Y. Li, R. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, 2014, pp. 1–6.

[3] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

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

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

[6] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 728–735.

[7] J. Huang, J. Lach, and G. Robins, "A methodology for energy-quality tradeoff using imprecise hardware," in *ACM Proceedings of the 49th Annual Design Automation Conference (DAC)*, 2012, pp. 504–509.

[8] Z. Babić, A. Avramović, and P. Bulić, "An iterative logarithmic multiplier," *Microprocessors & Microsystems*, vol. 35, no. 1, pp. 23–33, 2011.

[9] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512–517, 1962.

[10] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC)*, 2010, pp. 1–4.

[11] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, 2014, pp. 95:1–95:4.

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

[13] S. Narayanamoorthy, H. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 23, no. 6, pp. 1180–1184, 2015.

[14] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem, "Probabilistic arithmetic and energy efficient embedded signal processing," in *ACM Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2006, pp. 158–168.

[15] M. S. Lau, K.-V. Ling, and Y.-C. Chu, "Energy-aware probabilistic multiplier: Design and analysis," in *ACM Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2009, pp. 281–290.

[16] K. Palem, "Energy aware computing through probabilistic switching: a study of limits," *IEEE Transactions on Computers*, vol. 54, no. 9, pp. 1123–1137, 2005.

[17] V. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *47th ACM/IEEE Design Automation Conference (DAC)*, 2010, pp. 555–560.

[18] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: Systematic logic synthesis of approximate circuits," in *49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2012, pp. 796–801.

[19] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2011, pp. 667–673.