FEMTO: Fast Error Analysis in Multipliers through Topological Traversal

Deepashree Sengupta and Sachin S. Sapatnekar
Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis, MN 55455, USA.

***Abstract*—Approximate computing has emerged as a circuit design technique that can reduce system power without significantly sacrificing the output quality in error-resilient applications. However, there are few approaches for systematically and efficiently determining the error introduced by approximate hardware units. This paper focuses on the development of error analysis techniques for approximate multipliers, which are a key hardware component used in error-resilient applications, and presents a novel algorithm that efficiently determines the probability distribution of the error introduced by the approximation. The accuracy of the technique is demonstrated to be comparable to Monte Carlo simulations, while being significantly less computationally intensive.**

I. Introduction

Approximate computing [1] has emerged as a promising approach that can potentially achieve large efficiencies in the design of error-resilient systems, such as those that implement applications related to multimedia (image/video processing and streaming), data mining, and human auditory and visual perception. By deliberately introducing approximations, this approach reduces hardware costs, in terms of energy, power, and area, while ensuring that errors are limited to a level that can be tolerated by the end-user.

A vital ingredient of any methodology based on approximate design is a fast and accurate procedure that can quantify the distribution of error injected into a computation by an approximation scheme. The most common building blocks that are used to build hardware for error-resilient computations are adders and multipliers. While existing methods have made some progress in analyzing errors in adders [2]–[5], design of the approximate multipliers [6]–[9] still relies on error metrics from Monte Carlo simulations for performance evaluation since there are no known analytical methods that can scalably and accurately analyze the error in multipliers.

In this work, we propose a novel algorithm, **FEMTO**: **F**ast **E**rror Analysis in **M**ultipliers through **TO**pological Traversal, to efficiently quantify the errors in the output of an approximate multiplier by determining their probability of occurrence. The errors in approximate circuits which follow discrete asymmetric distributions [10], are propagated through networks using a topological traversal, and FEMTO uses the frequency domain to reduce computation.

At the gate level of approximate circuit design, the error of a logic function can be quantified by comparing the truth table of the approximate and exact implementations. However, this is not scalable beyond a small number of inputs because the size of the truth table grows exponentially with the number of inputs. Prior approaches that attempt to overcome the computational bottleneck of error estimation can be classified into two categories:

1) Methods that estimate the range of error: These methods capture the range of the error in approximate computation in terms of its minimum and maximum value, and are primarily based on interval and affine arithmetic [11], with modifications [3], [10], [12] suitable for asymmetric distributions of errors in approximate circuits. However, these approaches are computationally intensive, may lead to storage explosion [10], and often overestimate errors [13].

2) Methods that capture the statistics of the error distribution: These methods use the computationally intensive Monte Carlo simulations using millions of random input vectors to obtain various error metrics in an approximate computation such as the error rate, error significance, average error, and mean square error to quantify the error in approximate systems [2], [5], [6], [8], [14].

In several scenarios, it is essential to determine the entire probability distribution of error, e.g. for hypothesis testing in stochastic sensor circuits [15] and for accuracy evaluation [8], [9] of approximate circuits (which is currently performed by exhaustive/Monte Carlo simulations). FEMTO captures the entire probability distribution of error, and is significantly faster than Monte Carlo simulations. The advantage of obtaining an analytical expression for the error probability mass function (PMF) is that the error range, its statistics and percentiles can be easily deduced from the cumulative distribution function (CDF), and this method can be used in any framework that relies on Monte Carlo simulations to evaluate performance of approximate multipliers.

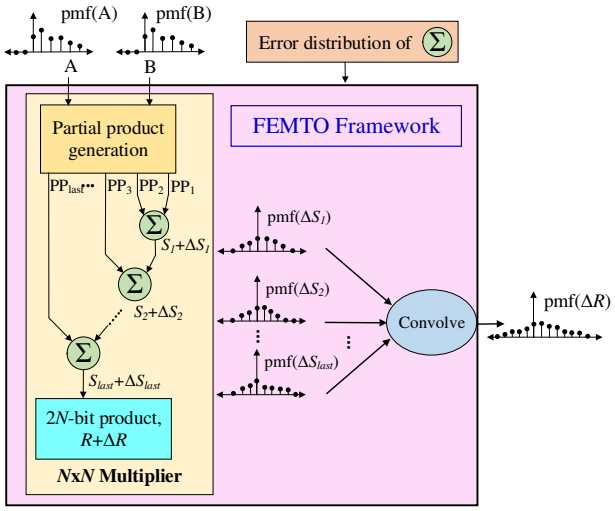


Fig. 1. Schematic of the FEMTO algorithm on an unsigned multiplier.

Our algorithm is schematically represented in Fig. 1 on an N-bit × N-bit unsigned multiplier. The statistics of the two N-bit operands A and B (i.e., the probability that they take on values 1 and 0) are provided as an input to our approach. The multiplication process generates partial products (PP_1, $\cdots$, PP_{last}) as shown in the figure, and the computation proceeds by successively adding each partial product to the partial sum computed so far. Each such addition is performed by an array, Σ, of approximate and/or exact full adders, and is characterized by an error PMF for the adder array. Our approach proceeds as follows:

1) We obtain the PMF of the error of each full adder (Section II).
2) We use these PMFs to compute the PMF of ΔS_i, the error introduced by the approximate adders at the i^{th} level, over the statistical distribution of inputs A and B.
3) The total error introduced by the multiplication is the sum of the

ΔS_i variables. We show that the PMF of the total error can be expressed as a convolution of a weighted set of error PMFs for individual full adders, and demonstrate how this convolution is performed efficiently in the frequency domain in an intelligent manner, avoiding an explosion in the number of terms in the frequency-domain representation of the PMF (Section III).

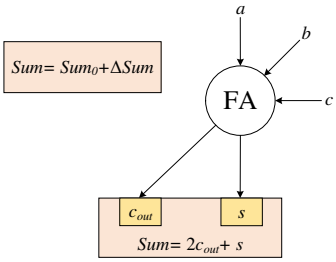
4) We enhance the speed of our algorithm by partitioning the N-bit operands into K-bit slices (Section IV).

We experimentally validate our results in Section V on a set of approximate multiplier schemes and conclude the paper in Section VI.

II. Characterizing the PMF of full adders

In principle, the PMF of any combinational structure can be characterized through its truth table and the statistics of the inputs. However, the size of the truth table increases exponentially with the size of the input space, and such a direct characterization is impractical for a multiplier. Hence, we work with a fundamental unit that can reasonably be characterized – in this case, a full adder (FA) – and develop the error PMF for the multiplier hierarchically. Specifically, the error PMF of a single adder is used to obtain the error PMF of each row of FAs that sums the partial products, and finally the error distribution of the entire approximate multiplier. This section explains how we use the input distribution and Boolean function of an FA to obtain its output error distribution.

Let us explain our approach with the example of an approximate FA, "appx1" from [16], shown in Fig. 2. The truth table of its output, Sum, as compared against the exact output, Sum_0, is also shown in the figure. The inputs, a, b and c, are modeled as random variables with a known distribution, and the error injected by the multiplier is denoted as ΔSum. Since the inputs are binary, we represent their probability of being 1 as p_a, p_b and p_c, respectively. Similarly, $p'_x = 1 - p_x$, $(x = a, b, c)$, are the probabilities of a, b and c, respectively, to be 0. The PMF of the resultant sum (Sum) combining both the output bits, s and c_{out}, and the PMF of the error (ΔSum) in the resultant sum are defined by $f_{Sum}(n)$ and $f_{\Delta Sum}(n)$, respectively.



a	b	c	Sum	Sum0	ΔSum
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	2	1	1
0	1	1	2	2	0
1	0	0	0	1	-1
1	0	1	2	2	0
1	1	0	2	2	0
1	1	1	3	3	0

Fig. 2. Full adder (FA) with the associated truth table ("appx1" from [16]).

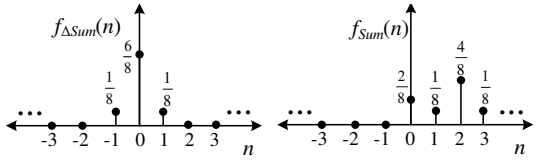


Fig. 3. Output signal and error distribution for the "appx1" adder from [16].

If the inputs are independent, and represented by an identical uniform distribution (p_a=p_b=p_c=0.5), then $f_{\Delta Sum}(n)$ and $f_{Sum}(n)$ can trivially be obtained from the truth table, and are depicted in the two plots in Fig. 3. For example, the PMF of ΔSum can be computed by observing that it takes the value 0 in six of eight entries in the truth table, and the values -1 and 1 in the remaining two entries. When the inputs are equiprobable, this leads to the PMF shown here. The PMFs in the figure can equivalently be represented as a weighted sum of discrete Kronecker delta functions as:

$$f_{\Delta Sum}(n) = \frac{6}{8}\delta(n) + \frac{1}{8}\delta(n-1) + \frac{1}{8}\delta(n+1) \quad (1)$$

$$f_{Sum}(n) = \frac{2}{8}\delta(n) + \frac{1}{8}\delta(n-1) + \frac{4}{8}\delta(n-2) + \frac{1}{8}\delta(n-3) \quad (2)$$

The coefficients of the delta functions in (1) and (2) are the PMF values of the associated random variable (ΔSum or Sum) and are the length of the stems in Fig. 3. When the inputs are not equiprobable, the coefficient of $\delta(n-k)$ in the PMF of a random variable is the probability of the variable to be k; hence, assuming the inputs to be independent, the coefficients can also be expressed as functions of p_a, p_b and p_c as shown in Table I and II.

TABLE I
PMF of the FA output.

n	$f_{Sum}(n)$
0	$p_0 = p'_a p'_b p'_c + p_a p'_b p'_c$
1	$p_1 = p'_a p'_b p_c$
2	$p_2 = p'_a p_b + p_a p'_b p_c + p_a p_b p'_c$
3	$p_3 = p_a p_b p_c$

TABLE II
PMF of the errors in FA output.

n	$f_{\Delta Sum}(n)$
-1	$e_{-1} = p_a p'_b p'_c$
0	$e_0 = 1 - p_a p'_b p'_c - p'_a p_b p'_c$
1	$e_1 = p'_a p_b p'_c$

For a general approximate FA, ΔSum can range from -3 to 3, as the two output bits may have error of any of -1, 0 or 1 (although for the "appx1" adder shown in Fig. 2, it only ranges from -1 to 1). Hence, the PMF of ΔSum and Sum can, in general, be expressed as a sum of Kronecker delta functions as:

$$f_{\Delta Sum}(n) = \textstyle\sum_{i=-3}^{3} e_i \delta(n-i) \quad (3)$$

$$f_{Sum}(n) = \textstyle\sum_{i=0}^{3} p_i \delta(n-i) \quad (4)$$

where the p_is and e_is are expressed as functions of p_a, p_b and p_c similar to Table I and II, respectively, and can be computed by substituting the values from the knowledge of the input distribution.

III. Overview of the FEMTO Algorithm

Consider a 4-bit $\times$ 4-bit array multiplier with operands, A ($a_3a_2a_1a_0$) and B ($b_3b_2b_1b_0$), as shown in Fig. 4. The full adders in the array are each indexed as FA_{ij}, where i corresponds to the row number, starting from the top, and j to the position of the FA in the row, starting from the least significant bit, as shown in Fig. 4. The output of a single adder, FA_{ij}, is modeled as the random variable, $Sum_{ij} = Sum_{ij,0} + \Delta Sum_{ij}$, where $Sum_{ij,0}$ is the true sum (corresponding to an exact FA), and ΔSum_{ij} is the error due to the approximate addition, similar to the example of the FA in Fig. 2.

Before proceeding further, let us comment on the input data distribution, and our assumptions regarding the correlation between the various Sum_{ij} random variables. Although we assume the inputs, A and B, to be independent random variables, their distribution is a user-input (and can be any arbitrary distribution) from which the signal probability, p_{a_i} and p_{b_i}, of each input bit, a_i and b_i, respectively, can be inferred. In addition to the error PMF, FEMTO has the capability to produce the output PMF (signal probability of the output bits of the multiplier), which can be used as input signal probability in subsequent multipliers within a data flow graph. Thus FEMTO can propagate the probability distribution of the data from input to the output of the multiplier. Additionally, within the multiplier, we consider the correlation between the s and c_{out} bits of any adder by combining them into a two-bit output, Sum, and the corresponding error, ΔSum, both expressed in decimal, with their PMF characterized by similar methods as Tables I and II, respectively. This technique captures the most important correlation which is the interdependence of the two output bits of any adder within the

multiplier array. Although we ignore the correlations between the outputs of different adders by considering Sum_{ij}s to be independent random variables, this assumption does not affect the quality of our results since correlations due to reconvergent fanout tend to be diluted as the logic depth of the reconvergent fanout paths increases.

Finding the error PMF for the multiplier array involves three steps:

1) determining the input probabilities for all inputs of each individual FA$_{ij}$, and using the approach in Section II to compute the PMF of ΔSum_{ij},
2) finding the PMF of the error, ΔS_i, introduced by the i^{th} row of the multiplier array, and
3) finding the PMF of the entire multiplier, i.e., the PMF of the sum of the ΔS_i variables over all rows, i.

Step 1: The first step simply involves probability propagation within a Boolean network, and we use established techniques for this purpose [17]. Based on this, we obtain the PMF of ΔSum_{ij}, denoted by $f_{\Delta Sum_{ij}}(n)$, as a sum of delta functions similar to (3).

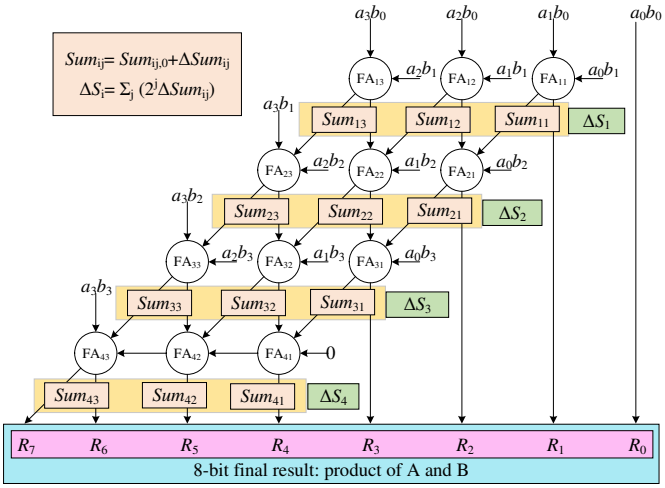


Fig. 4. Structure of a 4-bit × 4-bit array multiplier.

Step 2: Next, we determine the error in the partial product accumulation, ΔS_i, in the i^{th} row, which is the total error resulting from an array of $N-1$ approximate FAs, as depicted in Fig. 4 for $N=4$. For each row, $i \in \{1, \cdots, N\}$, a simple analysis yields:

$$\Delta S_i = \sum_{j=1}^{N-1} 2^{i+j-1} \Delta Sum_{ij} \tag{5}$$

Since we consider the ΔSum_{ij} random variables to be independent[1], we can utilize the fact that the PMF of sum of independent random variables equals the convolution of the PMF of those random variables. Hence the PMF of ΔS_i can be expressed as:

$$f_{\Delta S_i}(n) = \bigotimes_{j=1}^{N-1} f_{2^{i+j-1}\Delta Sum_{ij}}(n) \tag{6}$$

where $\bigotimes$ is the convolution operator, applied here to convolve $N-1$ operands, and $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$ is the PMF of the random variable, $2^{i+j-1}\Delta Sum_{ij}$.

If the absolute value of the largest output error of FA$_{ij}$ is M (e.g., $M=1$ for the "appx1" FA in Section II), then using (3),

$$f_{2^{i+j-1}\Delta Sum_{ij}}(n) = \sum_{k=-M}^{M} e_k \delta(n - 2^{i+j-1}k) \tag{7}$$

[1]The assumptions are also borne out by results.

Step 3: The error, ΔR, in the multiplier output is simply the sum of the errors, ΔS_i, over all N rows. Assuming the ΔS_i random variables to be independent, we obtain the error PMF of the multiplier result, $f_{\Delta R}(n)$, by convolving the $f_{\Delta S_i}(n)$ PMFs:

$$f_{\Delta R}(n) = \bigotimes_{i=1}^{N} f_{\Delta S_i}(n) = \bigotimes_{i=1}^{N} \bigotimes_{j=1}^{N-1} f_{2^{i+j-1}\Delta Sum_{ij}}(n) \tag{8}$$

We implement the following techniques to solve the above convolution problem to obtain the PMF of error in the final product:

A. Use the Z-Transform [18] to convert the convolution into a friendlier multiplication in the frequency domain, yielding a polynomial in z. This polynomial can have exponential number of terms, and special techniques are required to manage the cost of working in the transform domain.
B. Use the Inverse Fast Fourier Transform (IFFT) [18] to infer the PMF of ΔR from the polynomial obtained in the previous step.

Next, we explain each of these techniques in detail.

A. Representing the convolution using the Z-Transform

According to the principles of transform calculus, the Z-Transform of a convolution of multiple functions in the original domain is equivalent to the product of the Z-Transforms of those individual functions in the transform domain. Hence we can represent $F_{\Delta R}(z)$, the Z-Transform of $f_{\Delta R}(n)$ in (8), as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N} \prod_{j=1}^{N-1} F_{2^{i+j-1}\Delta Sum_{ij}}(z) \tag{9}$$

where $F_{2^{i+j-1}\Delta Sum_{ij}}(z)$, is the Z-Transform of the PMF, $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$. Applying the Z-Transform to both sides of (7),

$$F_{2^{i+j-1}\Delta Sum_{ij}}(z) = \sum_{k=-M}^{M} e_k z^{-2^{i+j-1}k} \tag{10}$$

Substituting (10) in (9), we can rewrite $F_{\Delta R}(z)$ as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N} \prod_{j=1}^{N-1} \left(\sum_{k=-M}^{M} e_k z^{-2^{i+j-1}k} \right) \tag{11}$$

$$= z^{-E} \sum_{i=0}^{2E} a_i z^i \tag{12}$$

$$= z^{-E} \phi(z) \tag{13}$$

where ΔR ranges from $-E$ to E, with $E = (2^{2N-1} - 2)M$ (derivation of E is omitted due to space limitations), and the a_is are the coefficients of the polynomial in z, denoted by $\phi(z)$ in (13). Performing the Inverse Z-Transform of (12),

$$f_{\Delta R}(n) = \sum_{i=0}^{2E} a_i \delta(n + i - E) \tag{14}$$

Hence a_i is the probability of the error, ΔR, to be $-(i-E)$. Thus finding the PMF of the error reduces to the problem of finding the coefficients, a_i, in $\phi(z)$ $\left(= \sum_{i=0}^{2E} a^i z^i\right)$, which is a polynomial of degree $2E$ with non-negative coefficients.

While this scheme presents a clear picture of our computation scheme, the cost of a direct implementation of this idea is prohibitive. The most expensive step is the determination of the coefficients, a_i, by multiplying the terms in (11). Therefore, we develop an efficient scheme for finding the coefficients, a_i.

B. Using the IFFT to infer $f_{\Delta R}(n)$ from $\phi(z)$ and $F_{\Delta R}(z)$

In this subsection, we present a method for efficiently computing the a_i coefficients in (14).

So far we have worked in the Z-Transform domain to formulate the error PMF equation, $F_{\Delta R}(z)$. Let us now consider discrete Fourier domain to determine the coefficients, a_i, in $\phi(z)$ from (13), by using their Fourier-Transformed values followed by performing Inverse Fast Fourier Transform (IFFT). This interchange of domains is possible since, by definition, Z-Transform is equivalent to Discrete Time Fourier Transform (DTFT) when the magnitude of $|z| = 1$ [18].

We begin by observing that the DTFT of the sequence, $\{a_0, a_1, \cdots, a_{2E}\}$, is given by the Fourier coefficients,

$$A_k = \sum_{i=0}^{2E} a_i \exp\left(-j\frac{2\pi ik}{2E+1}\right) = \sum_{i=0}^{2E} a_i z_k^i \tag{15}$$

where $z_k = \exp\left(-j\frac{2\pi k}{2E+1}\right)$. It is interesting to note that the values of z_k are the reciprocal of the $(2E+1)^{\text{th}}$ complex roots of unity.

Therefore, if we take $\phi(z)$ in (13) and substitute $z = z_k$, for the reciprocal of each of the $(2E+1)^{\text{th}}$ complex roots of unity, we obtain the Fourier coefficient, A_k. In other words,

$$A_k = \phi(z_k) = z_k^E \prod_{i=1}^{N} \prod_{j=1}^{N-1} \left(\sum_{k=-M}^{M} e_k z_k^{-2^{i+j-1}k} \right) \tag{16}$$

This provides us with the discrete Fourier coefficients of the sequence of a_is, which are then obtained by performing Inverse Discrete Time Fourier Transform (IDFT) of the A_k values as:

$$a_i = \frac{1}{2E+1} \sum_{k=0}^{2E} A_k exp\left(j\frac{2\pi ik}{2E+1}\right) \tag{17}$$

To compute the IDFT in (17) efficiently, we use the Inverse Fast Fourier Transform (IFFT) to obtain the values of the a_is. As explained in the previous subsection, obtaining the a_is directly provides the PMF of the error, ΔR, in the multiplier output.

IV. Enhancing efficiency of FEMTO

The error PMF obtained by the FEMTO algorithm provides probability of occurrence of errors with **unit-granularity**. In other words, we obtain $f_{\Delta R}(n)$ in (14), for each integer value of n within the error range, i.e., with unit spacing between successive values of n.

To enhance efficiency, we propose to process the multiplication by partitioning each of the N-bit operands, A and B, in an N-bit $\times$ N-bit multiplier, into K-bit slices. The product of A and B is obtained using the results of K^2 N/K-bit $\times$ N/K-bit multipliers as shown in Fig. 5. We call this the **partitioned-granularity** approach of obtaining the PMFs.

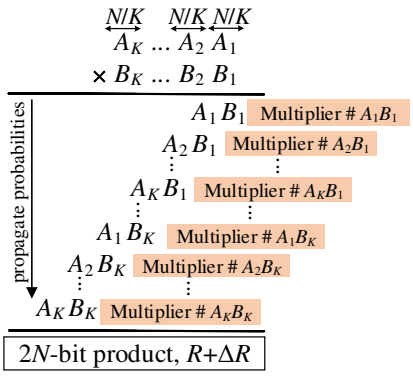


Fig. 5. N-bit $\times$ N-bit multiplier from N/K-bit $\times$ N/K-bit multipliers.

We then process the computation hierarchically. Instead of an FA, as in the previous section, we now use an N/K-bit $\times$ N/K-bit multiplier as the fundamental unit. However, while the PMF of an FA can be exactly characterized through the truth table, this is not the case for the N/K-bit $\times$ N/K-bit multiplier; instead, we use the approach in Section III to obtain this PMF. Characterizing the PMF of the N/K-bit $\times$ N/K-bit multiplier requires two types of input data: the error distribution of the FAs, which is provided in Section II, and the PMFs of the inputs to the multiplier, which is computed using the probability propagation algorithm in Step 1 of Section III (note that these probabilities are cheap to compute, and do not change whether we use this hierarchical scheme or the previous "flat" scheme).

Given the PMF of the N/K-bit $\times$ N/K-bit multiplier as the fundamental block, the scheme in Section III can now be used to find the PMF of the multiplier error.

Practical runtime enhancement of FEMTO: The error range, $-E_K$ to E_K ($E_K \sim 2^{2N/K}$), of the N/K-bit $\times$ N/K-bit multiplier, can be grouped or "binned" into P windows for further runtime enhancement of FEMTO, where P is chosen empirically. This idea is best explained with an example in Fig. 6. If the output error in the N/K-bit $\times$ N/K-bit multiplier is represented by the random variable, ΔR, with $E_K = 15$, and the PMF of ΔR is $f_{\Delta R}(n)$, then we can group ΔR into $P = 5$ windows to obtain the binned version of the PMF, $f_{\Delta R,bin}(n)$ with fewer data points.

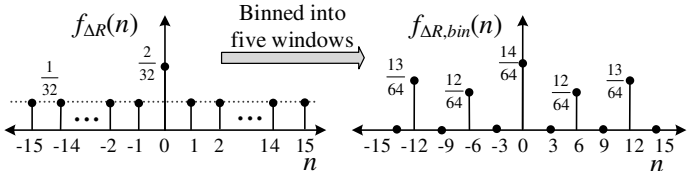


Fig. 6. An error PMF (not to scale) with unit-granularity (left) and its binned version into five windows (right).

There is a trade-off between the computational effort, and the accuracy of the PMF obtained by this approach. While the algorithm speeds up by $\left(\frac{2E_K}{P}\right)$ X, inaccuracy is introduced due to the representation of $\frac{2E_K}{P}$ error values by a single value. Therefore, P should be chosen depending on the value of E_K to maintain an acceptable runtime-accuracy trade-off.

V. Results

We implement FEMTO in MATLAB R2010b in a 2.53 GHz Intel Core i3 CPU with 4Gb RAM and 64-bit Windows 7 OS, and present the results relating to approximate unsigned multipliers. The approximate adders that constitute the multipliers are the various transistor level approximations of the mirror adders [16], the Boolean expressions of which are mentioned in Table III for the two outputs, s and c_{out}, and inputs, a, b and c.

TABLE III
Five versions of approximate adders from [16].

Approximate version	s	c_{out}
appx1	$\bar{a}bc + abc$	$abc + ab\bar{c} + a\bar{b}c + \bar{a}bc + \bar{a}b\bar{c}$
appx2	$\overline{c_{out}}$	$ab + \bar{a}bc + a\bar{b}c$
appx3	$\overline{c_{out}}$	$abc + ab\bar{c} + a\bar{b}c + \bar{a}bc + \bar{a}b\bar{c}$
appx4	$\bar{a}\bar{b}c + \bar{a}bc + abc$	a
appx5	b	a

Since in practice, no more than $\sim$50% of the resultant bits are usually approximated to maintain accuracy [8], [16], we use a similar strategy to approximate different number adders to implement the

multipliers. We obtain the PMF of the errors normalized to the dynamic range of the output of the approximate multiplier. The normalization factor, $\mathcal{R}$, is the total range (difference of maximum and minimum values) of the output when the multiplier is implemented using different combinations of approximate and accurate adders. The outputs are obtained by performing 2000 Monte Carlo logic simulations on the approximate multiplier. Such a normalization step is necessary since the same magnitude of error may have different levels of severity depending on the magnitude of the product.

First, we present the results for the combination of approximate and accurate adders to construct the multipliers, such that exactly 50% of the product bits from the second-least significant bit position are approximate. Referring to Fig. 4, this means that for the 4×4 multiplier, the resultant bits, $R_1, \cdots, R_4$, are approximate and the rest are exact. The least significant bit (LSB), R_0, is exact since it is not produced by any adder and is simply the output of an AND gate. In general, for an $N \times N$ multiplier, to approximate 50% of the LSBs in the product, $N-i$ LSB adders in the i^{th} row of the adder array in each partial product accumulation level should be approximate, with the rest being accurate, for $i = 1, \cdots, N$.

We implement FEMTO on 6×6, 8×8, and 16×16 approximate multipliers. While the error PMFs for 6×6 and 8×8 are obtained by the unit-granularity approach, the error PMF for 16×16 multiplier is obtained by the partitioned-granularity approach using $K = 2$, i.e., using the results of the 8×8 multiplier, and $P = 32$ windows to enhance the practical runtime, as explained in Section IV.

We compare the mean and standard deviation of the error PMFs obtained by FEMTO and Monte Carlo simulations using the absolute value of the normalized percentage error in mean and standard deviation, $\Delta\mu_{norm}$ and $\Delta\sigma_{norm}$, respectively, defined as:

$$\Delta\mu_{norm} = 100 \times \frac{|\mu_{FEMTO} - \mu_{MC}|}{\mathcal{R}} \tag{18}$$

$$\Delta\sigma_{norm} = 100 \times \frac{|\sigma_{FEMTO} - \sigma_{MC}|}{\mathcal{R}} \tag{19}$$

where μ_{FEMTO} (μ_{MC}) and σ_{FEMTO} (σ_{MC}) are the mean and standard deviation of the error PMF of each multiplier obtained by FEMTO (Monte Carlo simulation), respectively, and $\mathcal{R}$ is the normalizing factor defined earlier. We summarize the $\Delta\mu_{norm}$ and $\Delta\sigma_{norm}$ values, respectively, in Tables IV and V, corresponding to the error PMFs of the 6×6, 8×8, and 16×16 approximate multipliers (with 50% of the product bits approximated).

Clearly, the error is less than 1% as observed by the $\Delta\mu_{norm}$ and $\Delta\sigma_{norm}$ values for all the approximate multipliers (with different versions of the approximate adders) considered here. This indicates that the error statistics obtained by FEMTO are very similar to those obtained by the Monte Carlo simulations.

TABLE IV
NORMALIZED PERCENTAGE ERROR IN ESTIMATED MEAN ($\Delta\mu_{norm}$) OF PMF OBTAINED BY FEMTO COMPARED AGAINST MONTE CARLO SIMULATION.

Multiplier → **Adder ↓**	6×6	8×8	16×16
appx1	0.23	0.12	0.12
appx2	0.09	0.06	0.41
appx3	0.14	0.13	0.67
appx4	0.01	0.01	0.02
appx5	0.02	0.02	0.26

TABLE V
NORMALIZED PERCENTAGE ERROR IN ESTIMATED STANDARD DEVIATION ($\Delta\sigma_{norm}$) OF PMF OBTAINED BY FEMTO COMPARED AGAINST MONTE CARLO SIMULATION.

Multiplier → **Adder ↓**	6×6	8×8	16×16
appx1	0.10	0.06	0.26
appx2	0.00	0.01	0.26
appx3	0.01	0.04	0.03
appx4	0.38	0.11	0.03
appx5	0.44	0.18	0.31

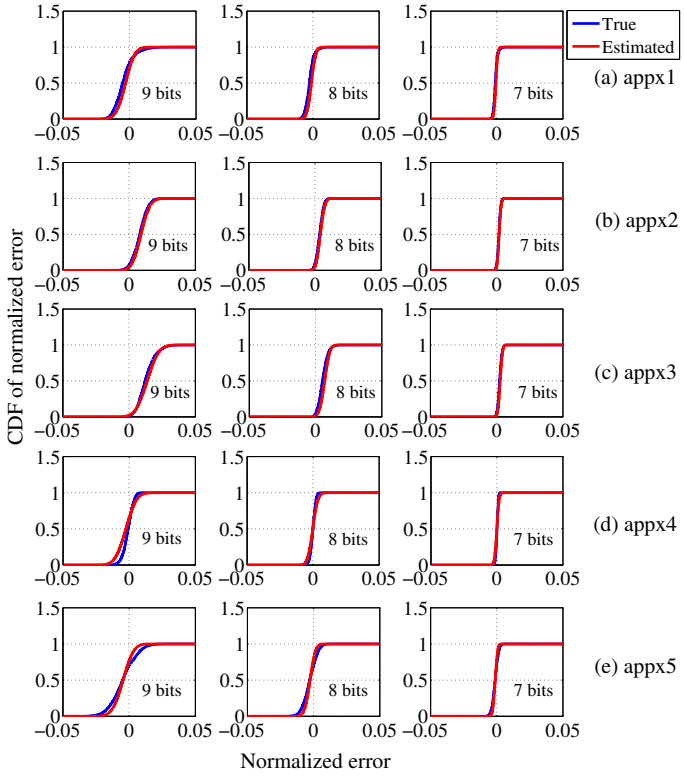


Fig. 7. CDF of normalized error for 8×8 multipliers with different percentages (45%, 50% and 55%) of LSBs in the product being approximate. The adders in (a)-(e) are the ones from Table III, with seven, eight and nine bits (45%, 50% and 55% of 16 bits) being approximated in the product.

Next, we consider cases when different percentages (45%, 50% and 55%) of LSBs are approximate in the product. Due to limited space, we only present the error CDFs of the 8×8 multipliers as an example. Each column of subplots in Fig. 7 corresponds to a specific number of approximate bits in the 16-bit product of the 8×8 multiplier. The numbers, 7, 8 and 9 bits, respectively, correspond to 45%, 50% and 55% of the 16-bits being approximated. Each row of subplots in Fig. 7 corresponds to the type of adders from Table III used to implement the approximate multiplier. The blue plot labeled “True" represents the CDF obtained by 2000 Monte Carlo simulations, and is assumed to be the reference or golden CDF. The red plot labeled “Estimated" is the CDF obtained by FEMTO.

As expected, the errors are reduced with fewer approximate bits in the product. Additionally, we observe that the PMFs obtained by FEMTO are very close to those obtained by Monte Carlo simulation for different levels of approximation in the multiplier (as seen across each row of subplots in Fig. 7). Hence, for the various approximate multipliers considered here, FEMTO can predict the error distribution with accuracy comparable to Monte Carlo simulations.

To compare with one of the existing approaches of obtaining error

PMF in approximate circuits, we also generate the error PMF of the multipliers using the modified interval arithmetic (MIA) based approach [10]. The authors of [10] had reported MIA to be very efficient in terms of runtime and storage complexity. Hence we compare our approach with the MIA-based one to obtain the error PMF. The multipliers are approximated such that 50% of the product LSBs are approximate, and the rest are accurate. We compare the accuracy of the PMFs obtained by both FEMTO and MIA (implemented based on [10]), by observing the Hellinger distance [19] from the corresponding PMFs obtained from Monte Carlo simulations. Hellinger distance is a well-known metric to compare probability distributions, and is defined as:

$$\text{Hellinger distance} = \frac{1}{\sqrt{2}}\sqrt{\sum_{\text{all } n}\left(\hat{f}(n) - f(n)\right)^2} \qquad (20)$$

where $\hat{f}(n)$ and $f(n)$ are the estimated and the true (Monte Carlo) PMFs, respectively. The factor, $\sqrt{2}$, ensures that this distance ranges from 0 to 1. We plot the Hellinger distances of the error PMFs obtained by FEMTO and MIA for the 6×6 and 8×8 multipliers in Fig. 8, from the PMFs obtained by Monte Carlo simulations. The smaller the distance, the closer is the estimate to the true PMF; thus clearly our approach yields more accurate PMFs compared to the MIA-based ones as seen by the lower values of the Hellinger distance for the approximate multipliers considered here.

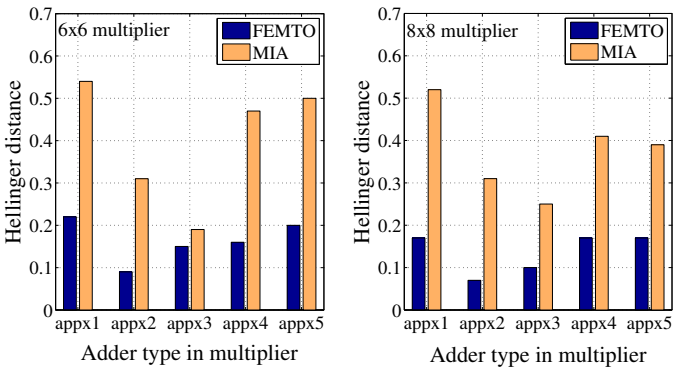


Fig. 8. Hellinger distance between Monte Carlo-generated PMF, and PMFs obtained by FEMTO and our implementation of MIA.

The runtime to obtain the PMFs are summarized in Table VI. The 16×16 multiplier did not finish computation within 3600 seconds which was set as the maximum observation time using the MIA-based approach, hence its runtime is labeled "timed out" in the table for all approximate versions of the adders.

TABLE VI
RUNTIME COMPARISON TO OBTAIN THE ERROR PMFS.

Multiplier →	6×6		8×8		16×16	
Adder ↓	FEMTO	MIA	FEMTO	MIA	FEMTO	MIA
appx1	0.8s	6.6s	14.5s	174.2s	363.8s	timed out
appx2	0.7s	9.3s	14.0s	265.1s	364.6s	timed out
appx3	0.8s	9.7s	13.9s	257.5s	365.3s	timed out
appx4	0.8s	9.6s	14.4s	263.3s	369.9s	timed out
appx5	0.6s	10.1s	8.9s	293.5s	362.5s	timed out

Clearly, not only is the accuracy of our method higher than MIA, but also the runtime shows excellent improvements. In conclusion, although MIA can quantify the error PMFs, due to the above reasons of inaccuracy and poor runtime, it has not been implemented yet in any multiplier design.

Our approach is thus superior to the existing approaches to quantify the error PMF, and can be incorporated into the multiplier design framework to speed up the performance evaluation process.

VI. CONCLUSION

We have proposed a fast algorithm to analytically obtain the error PMF in an unsigned multiplier. The algorithm has been implemented to obtain error PMFs in different types and sizes of approximate multipliers comprising of transistor-level approximate adders. We have validated the results of our algorithm against Monte Carlo simulations, in terms of the various statistics and the error CDFs. The ease with which the PMF can be obtained will help approximate circuit designers to use FEMTO in an optimization framework to evaluate the circuit performance, and design low power approximate systems within a specified error tolerance.

ACKNOWLEDGMENT

This work was supported by the NSF grant (CCF-1017778 and CCF-1162267).

REFERENCES

[1] J. Han and M. Orshansky, "Approximate computing: An Emerging Paradigm For Energy-Efficient Design," in *Proc. ETS*, pp. 1–6, 2013.
[2] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, "Modeling and Synthesis of Quality-Energy Optimal Approximate Adders," in *Proc. ICCAD*, pp. 728–735, 2012.
[3] N. Zhu, W. L. Goh, and K. S. Yeo, "An Enhanced Low-Power High-Speed Adder For Error-Tolerant Application," in *Proc. ISIC*, pp. 69–72, 2009.
[4] Y. Emre and C. Chakrabarti, "Low Energy Motion Estimation via Selective Approximations," in *Proc. ASAP*, pp. 176–183, 2011.
[5] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and Analysis of Circuits for Approximate Computing," in *Proc. ICCAD*, pp. 667–673, 2011.
[6] E. Swartzlander, "Truncated Multiplication with Approximate Rounding," in *Proc. 33rd Asilomar Conf. Signals, Systems, Computers*, vol. 2, pp. 1480–1483, 1999.
[7] K. J. Cho, K. C. Lee, J. G. Chung, and K. K. Parhi, "Design of Low-Error Fixed-Width Modified Booth Multiplier," *IEEE T VLSI Syst*, vol. 12, no. 5, pp. 522–531, 2004.
[8] B. Shao and P. Li, "Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design," *IEEE T CIRCUITS-I*, vol. 62, no. 4, pp. 1081–1090, 2015.
[9] C. Liu, J. Han, and F. Lombardi, "A Low-Power, High-Performance Approximate Multiplier with Configurable Partial Error Recovery," in *Proc. DATE*, pp. 1–4, 2014.
[10] J. Huang, J. Lach, and G. Robins, "Analytic Error Modeling for Imprecise Arithmetic Circuits," in *Proc. SELSE*, 2011.
[11] J. Stolfi and L. de Figueiredo, "An Introduction to Affine Arithmetic," 2003.
[12] J. Huang, J. Lach, and G. Robins, "A Methodology for Energy-Quality Tradeoff Using Imprecise Hardware," in *Proc. DAC*, pp. 504–509, 2012.
[13] W. T. J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical Analysis and Modeling for Error Composition in Approximate Computation Circuits," in *Proc. ICCD*, pp. 47–53, 2013.
[14] Y. C. Liao, H. C. Chang, and C. W. Liu, "Carry Estimation for Two's Complement Fixed-Width Multipliers," in *Proc. SIPS*, pp. 345–350, 2006.
[15] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, "Stochastic Computation," in *Proc. DAC*, pp. 859–864, 2010.
[16] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-Power Digital Signal Processing Using Approximate Adders," *IEEE T Comput. Aid. D.*, vol. 32, no. 1, pp. 124–137, 2013.
[17] S. Devadas, K. Keutzer, and J. White, "Estimation of Power Dissipation in CMOS Combinational Circuits Using Boolean Function Manipulation," *IEEE T Comput. Aid. D.*, vol. 11, no. 3, pp. 373–383, 1992.
[18] A. V. Oppenheim and A. S. Willsky, *Signals and Systems*. Prentice-Hall, 1997.
[19] M. S. Nikulin, "Hellinger distance." *Encyclopedia of Mathematics*. http://www.encyclopediaofmath.org/index.php/Hellinger_distance.