An Analytical Approach for Error PMF Characterization in Approximate Circuits

Deepashree Sengupta, Farhana Sharmin Snigdha, Jiang Hu, and Sachin S. Sapatnekar

***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 exists only a 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 circuits consisting of adders and multipliers, which are the key hardware components used in error-resilient applications. A novel algorithm has been presented, using the Fourier and the Mellin transforms, that efficiently determines the probability distribution of the error introduced by approximation in a circuit, abstracted as a directed acyclic graph (DAG). The algorithm is generalized for signed operations through two's complement representation, and its accuracy is demonstrated to be within 1% of Monte Carlo simulations, while being over an order of magnitude faster.**

***Index Terms*—Approximate Computing, Fourier Transform, Mellin Transform, Error Distribution.**

I. Introduction

While performance reliability of circuits is imperative in safety-critical and security applications, deliberate inaccuracy is permissible in certain error-tolerant applications, thus allowing relaxation of design effort and reduction of system power. Recognition, mining, and synthesis applications related to image, video, and audio processing [1] have significant levels of error-tolerance, since they pertain to the limited human perception of visual and auditory senses. Approximate computing [2], [3] is a new design paradigm that leverages this inherent error-tolerance, and can potentially achieve large efficiencies in the design by deliberately introducing errors in computation by either simplifying a circuit logic or architecture, or by modifying its operating conditions.

A vital ingredient of any methodology based on approximate design is a fast and accurate procedure that can quantify the error injected into a computation by an approximation scheme. We propose a novel technique to quantify this error through its probability mass function (PMF) in this paper, using the concepts of transform calculus.

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. Additionally, in several scenarios, it is essential to determine the entire probability distribution of error, e.g., for hypothesis testing in stochastic sensor circuits [4] and for accuracy evaluation [5], [6] of approximate circuits (which is currently performed by exhaustive/Monte Carlo simulations). In this paper, we present an algorithm that captures the entire probability distribution of the error in approximate circuits. A preliminary version of our work appeared in [7], where we quantified the error PMFs in unsigned approximate multipliers.Our approach first develops methods that compute the entire error PMF for individual blocks such as adders and multipliers. When these blocks are placed in a DAG, our approach propagates the error PMF through the DAG using a topological traversal to compute the error PMF at the DAG outputs. Some prior works have addressed the first aspect, but the second aspect has not received much attention.

A. Related work

Prior approaches that attempt to overcome the computational bottleneck of error estimation in approximate circuits (generally, individual circuit blocks) can be classified into two categories: (a) those that estimate the range of error, and (b) those that estimate the error statistics. The first category, which captures the range of the error in terms of its minimum and maximum value, is primarily based on interval and affine arithmetic [8]–[10]. The runtime for such interval-based approaches increases exponentially when more intervals are required for large ranges of signals, and they clamp the errors to maximum/minimum values in the range, thus often overestimating/underestimating the actual errors. While the theoretical complexity of our proposed approach is also exponential, we apply various techniques to improve the practical runtime, and achieve Monte-Carlo like accuracy, without overestimating the errors as evident from the results.

The second category, which estimates the error statistics, uses either computationally intensive Monte Carlo simulations using millions of random input vectors or other error composition techniques to obtain various quality metrics in an approximate computation, such as the error rate, error significance, average error, and mean square error [6], [11]–[14]. Our work, on the other hand, provides an analytical expression for the entire error PMF, which has more information and yields the simpler quality metrics listed above.

While [15] proposes a method to characterize error PMF in the basic block of adders only, [16] develops a technique for modeling operation-level error PMFs (in adders and multipliers) similar to our earlier approach [7], and propagates only certain metrics (such as the mean, variance, or the extremum values) instead of the actual PMF, as implemented in our work. In fact, the existing literature lacks a rigorous exploration of the error propagation through an approximate circuit comprising of not only adders, but also multipliers, and this is what we attempt to achieve in this paper. To the best of our knowledge, this is the first work that analytically computes the total error PMF in an approximate circuit considering both the errors generated at each approximate operation, and the errors propagated through them to the outputs of the circuit.

Our approach is applied to approximation schemes where the fundamental block, such as a full adder or a group of full adders can be abstracted by a Boolean function, e.g., as in [17]. For cases where approximation is introduced by modifying the logic, such a function can be obtained from the truth table, as explained in Section III. Although the equations outlined in the rest of the paper correspond to the specific case of logic approximations, they can be extended to more general cases if a Boolean function can be formulated at full adder level.

B. *Summary of contribution*

The input to our algorithm is the data flow graph of a circuit, represented as a directed acyclic graph (DAG) whose nodes correspond to fixed-point arithmetic units that can potentially be approximated, and whose edges indicate the connections between these units. Since the most common circuits that are used to build hardware for error-resilient computations are adders/subtractors [13], [14], [17], [18] and multipliers [5], [6], [12], we consider these operations within the approximate DAGs for our analysis.

We consider signed operations using the two's complement representation, where approximation is introduced in each node by simplifying the logic function of the constituent full adders (FAs) [17] of a fixed-width adder or multiplier array within the DAG under study. The Boolean function of the simplified FAs is also an input to our approach, along with the statistics of the operands to each node connected to primary inputs of the DAG. Our approach proceeds as follows:

1) For an FA, we obtain the PMF at the output, and the error in the output as impulse functions from its truth table that can be computed using its Boolean function (Section III).
2) Next, we use the above PMFs to compute the PMF of error generated at each approximate node of a circuit (Section IV). Specifically, for adders, the operation proceeds through an array of approximate and/or exact FAs, and the error generated in an approximate adder is the weighted sum of errors generated within the approximate FAs in the array. for multipliers, the operation proceeds by generating partial products, and successively adding each partial product to the partial sum computed so far. Each such addition is performed by an array of approximate and/or exact FAs, and the error generated in the approximate multiplier is the weighted sum of errors in these arrays. We show that the PMF of the error generated at each node can be expressed as a convolution of a weighted set of error PMFs for individual FAs, and demonstrate how we perform this convolution efficiently by using the Fourier transform [19] on the PMFs.
3) Since the error at the output of a node within a DAG consists of both the generated error (explained in the previous step) and the error that is propagated from its inputs, we next obtain the PMF of such propagated errors, using suitable assumptions regarding the statistics of the inputs, and the errors in the inputs. The error propagation through adders is implemented by adding the errors in its inputs, which translates to convolution of the input error PMFs to obtain the error PMF at the output. We use the Fourier transform to efficiently solve this convolution problem. The error propagation through multipliers proceeds by adding the products of error in one input with the true value of the other input. We propose a novel technique by using the Mellin transform [20], followed by the Fourier transform, to obtain the PMF of the propagated error.
4) Finally, we combine the PMF of the generated and the propagated errors at the output of each node, and through a topological traversal of the entire DAG, we compute the PMF of error at its primary outputs (Section V).

We present the results of error PMF computation for a few benchmark circuits for digital signal processing (DSP) applications, represented as DAGs [21], and approximated using the FAs from [17], in Section VI, and conclude in Section VII.

II. A DAG Model for an Approximate Circuit

An approximate circuit represented as a DAG can be illustrated in Fig. 1, where each node implements a fixed-width signed operation. Since a circuit may be used in a variety of user conditions (inputs vectors, voltage, temperature, etc.), to ensure its robustness, the inputs to an approximate circuit are typically considered as random variables [17], [22] whose distribution depends on the type of application the circuit is used for. As a result, the output is also a random variable, and so is the error in the output introduced due to approximations in the circuit. A schematic of both the input and output PMFs are also depicted in Fig. 1.

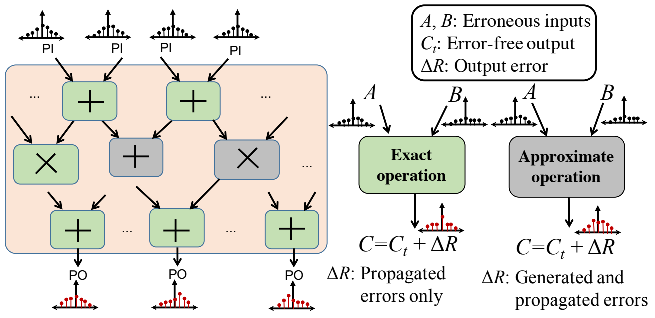


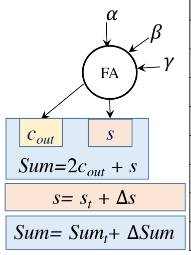
Fig. 1. Representation of a DAG with approximate operations, producing erroneous outputs that can be characterized by PMFs.

Let us consider each node of this DAG to represent an N-bit operation, having N-bit inputs and an N-bit output. Hence, the carry bit out of the most significant bit (MSB) FA of an adder node is ignored; similarly, the N MSBs within a multiplier node are ignored, by considering only the lower triangle of the multiplier array (further explanations through Figs. 4 and 5, respectively, in Section IV). Within such a node, transistor-level FA logic simplifications [17], [23] introduce approximation when the operation is implemented as an array of exact and approximate FAs. The structure of the array depends on whether the operation is an addition (or subtraction) or a multiplication. Since an approximate implementation of the hardware unit yields the benefit of using fewer resources [2] than its exact counterpart, typically some of the least significant bits (LSBs) can be allowed to be erroneous, as this introduces a limited level of approximation. If y such LSBs are approximate, for example, then, the higher the value of y, the greater are the power savings due to the imprecise hardware, although the error is also higher. The corresponding error, e, can potentially range from $-(2^y-1)$ to (2^y-1), and its exact value depends on the inputs. Typically inputs are assumed to be random variables following a known distribution (e.g., uniform or normal). Hence, e, being a function of these inputs, is also a random variable, as is the output of this node (irrespective of whether or not this node is approximate). To obtain the error PMF at the primary outputs of the DAG, we also need the PMFs of the actual outputs at each internal node since they are the inputs to the successive nodes within the DAG. Hence, our approach characterizes both the PMF of the output, and the error in the output of a DAG whose nodes are candidates for approximation.

III. Output Distribution of Full Adders

In principle, the output distribution 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 multi-bit adder or a multiplier. Hence, we work with a fundamental unit that can reasonably be characterized – in this case, an FA – and develop the error PMF for a multi-bit adder and multiplier hierarchically. Specifically, for an adder, the error PMF of a single FA is used to obtain the error PMF of the output, and for a multiplier, the error PMF of a single FA 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, and output error distribution.

Let us explain our approach with the example of an approximate FA, appx1, from [17], shown in Fig. 2. The single-bit inputs are α, β, and γ, and the outputs are c_{out} and s, which are combined as the two-bit output, Sum. The error injected by the adder is denoted by ΔSum which combines the error in both c_{out} and s. Since we will be analyzing signed operations, for the FA in the MSB position, we also observe the error, Δs, injected in the sign bit, which corresponds to the sum bit of this FA. Hence, the truth table in Fig. 2 lists the outputs, s, c_{out}, and the combined Sum, along with the exact outputs, s_t and Sum_t. The inputs are modeled as random variables with a known distribution. Since the inputs are binary, we represent their probability of being 1 as p_α, p_β, and p_γ, respectively, also known as the signal probability of those inputs. Similarly, $p'_x = 1 - p_x$, $x \in \{\alpha, \beta, \gamma\}$, are the probabilities of α, β, and γ, respectively, to be 0. The PMF of the adder output, Sum, combining both the output bits (s and c_{out}), and the PMF of the error, ΔSum, in the result, are defined by $f_{Sum}(n)$ and $f_{\Delta Sum}(n)$, respectively, where $n \in [0,3]$ for Sum since it denotes the two-bit output, $2c_{out} + s$, and $n \in [-3,3]$ for ΔSum, as the error within the two output bits may be any of -1, 0 or 1. Since we also need to compute the PMF of the error, Δs, in the sum bit, s, if the FA is at the MSB in an adder or a multiplier array, we denote its PMF by $f_{\Delta s}(n)$, where $n \in [-1,1]$ for Δs. However, it is to be noted that for the appx1 adder shown in Fig. 2, $n \in [-1,0]$ and $n \in [-1,1]$ for Δs and ΔSum, respectively.



α	β	γ	c_{out}	s	s_t	Δs	Sum	Sum_t	ΔSum
0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	1	1	0
0	1	0	1	0	1	-1	2	1	1
0	1	1	1	0	0	0	2	2	0
1	0	0	0	0	1	-1	0	1	-1
1	0	1	1	0	0	0	2	2	0
1	1	0	1	0	0	0	2	2	0
1	1	1	1	1	1	0	3	3	0

Fig. 2. Full adder (FA) with the associated truth table (appx1 from [17]).

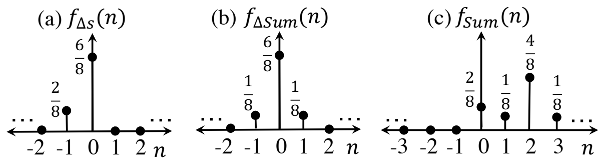


Fig. 3. Output signal and error distribution for the appx1 adder from [17].

If the inputs are independent, and represented by an identical uniform distribution (p_α=p_β=p_γ=0.5), then $f_{\Delta s}(n)$, $f_{\Delta Sum}(n)$, and $f_{Sum}(n)$ can be obtained from the truth table, and are depicted in Figs. 3(a)–(c). 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, leading to the PMF shown in Fig. 3(b). The three PMFs can equivalently be represented as a weighted sum of discrete Kronecker delta functions as:

$$f_{\Delta s}(n) = \frac{6}{8}\delta(n) + \frac{2}{8}\delta(n+1) \tag{1}$$

$$f_{\Delta Sum}(n) = \frac{6}{8}\delta(n) + \frac{1}{8}\delta(n-1) + \frac{1}{8}\delta(n+1) \tag{2}$$

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

where the coefficients of the delta functions are the length of the corresponding stems in Fig. 3.

However, when the inputs to the approximate FA are not uniformly distributed, each coefficient of $\delta(n-v)$ in Eqs. (1) to (3) represents the probability of the corresponding random variable to be v, where $v \in [-1,1]$, $v \in [-3,3]$, and $v \in [0,3]$, for Δs, ΔSum, and Sum, respectively. The coefficients of the delta functions can be expressed as functions of p_α, p_β, and p_γ as shown in Table I. For example, from the truth table of the appx1 FA as depicted in Fig. 2, we observe that error, Δs, in the sum bit is -1 when the input bitset, (α, β, γ) is either $(0,1,0)$, or $(1,0,0)$. Hence, the first entry of the first column, x_{-1}, in Table I, which is the probability of Δs to be -1 is formulated as $p'_\alpha p_\beta p'_\gamma + p_\alpha p'_\beta p'_\gamma$. Similarly, the other entries are populated in this table by observing the input bitsets when Δs, ΔSum, and Sum, take the value, v.

TABLE I
PMFs ASSOCIATED WITH THE APPX1 FA OUTPUT (Sum) AND OUTPUT ERRORS (Δs AND ΔSum).

n	$f_{\Delta s}(n) = x_n$	$f_{\Delta Sum}(n) = e_n$	$f_{Sum}(n) = p_n$
-1	$p'_\alpha p_\beta p'_\gamma + p_\alpha p'_\beta p'_\gamma$	$p_\alpha p'_\beta p'_\gamma$	$--$
0	$1 - p'_\alpha p_\beta p'_\gamma - p_\alpha p'_\beta p'_\gamma$	$1 - p_\alpha p'_\beta p'_\gamma - p'_\alpha p_\beta p'_\gamma$	$p'_\alpha p'_\beta p'_\gamma + p_\alpha p'_\beta p'_\gamma$
1	0	$p'_\alpha p_\beta p'_\gamma$	$p'_\alpha p'_\beta p_\gamma$
2	$--$	0	$p'_\alpha p_\beta + p_\alpha p'_\beta p_\gamma + p_\alpha p_\beta p'_\gamma$
3	$--$	0	$p_\alpha p_\beta p_\gamma$

Hence, the PMF of Δs, ΔSum, and Sum can, in general, be expressed as a sum of Kronecker delta functions as:

$$f_{\Delta s}(n) = \sum_{v=-1}^{1} x_v \delta(n-v) \tag{4}$$

$$f_{\Delta Sum}(n) = \sum_{v=-3}^{3} e_v \delta(n-v) \tag{5}$$

$$f_{Sum}(n) = \sum_{v=0}^{3} p_v \delta(n-v) \tag{6}$$

where x_v, e_v, and p_v are, respectively, the probabilities of Δs, ΔSum, and Sum to be v, and expressed as functions of p_α, p_β, and p_γ similar to the terms in Table I. The probabilities, p_α, p_β, and p_γ, can be computed from the knowledge of the input distribution, to obtain the PMFs in Eqs. (4) to (6).

IV. Distribution of Generated Error in DAG Nodes

Let us consider an approximate node in a DAG, with two N-bit operands, $A(a_{N-1}a_{N-2}\cdots a_0)$ and $B(b_{N-1}b_{N-2}\cdots b_0)$, producing an output, $C(c_{N-1}c_{N-2}\cdots c_0)$. We assume the inputs to be error-free, where A_t and B_t are the true values of inputs, and C comprises of only the error generated at this node, denoted by ΔR. The true output, C_t, would have been the output if this node was exact. In this section we explain the methodology to obtain the PMF of the errors generated at the individual nodes of a DAG, i.e., the PMF of ΔR.

There are quite a few applications (image compression and classification, audio filtering, matrix multiplication, etc.) that are amenable to approximation, which can be implemented using add/subtract/multiply operations. Hence, in our work, we consider these three operations as candidates for approximate nodes within a DAG. We consider two's complement signed operations, and for the adder (or subtractor) we assume the sum bit output of the MSB FA to be the sign bit as shown in Fig. 4. For multipliers, we assume the lower triangle of the Modified Baugh-Wooley two's-complement multiplication array structure [24] as shown in Fig. 5, so that N LSBs correspond to the actual result of the N-bit$\times$$N$-bit multiplier, that goes into the subsequent stages. The sum bit output of the FA in the penultimate row and the MSB column of the array is the sign bit as highlighted in Fig. 5.

Before proceeding further, let us comment on the input data distribution, and our assumptions regarding the correlation between the random variables associated with the outputs and errors of each FA in the adder or the multiplier array. Although we assume the inputs, A and B, to be independent random variables, their distribution is a user-specified 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, $i \in \{0, \cdots, N-1\}$. The assumption of input independence is necessary since the transforms we use to render our approach very easy to implement, are applicable for the addition and multiplication of only independent random variables. However, as illustrated in our experimental results, this assumption produces good enough error estimate for up to 50% approximate LSBs, which is the typical maximum approximation range for acceptable output quality [3], [17].

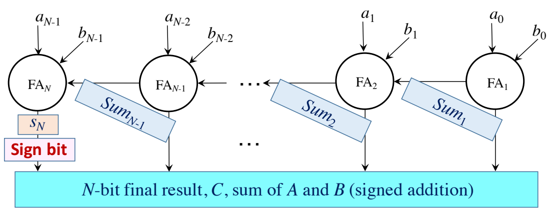


Fig. 4. A signed adder where some of the LSB FAs can be approximated to introduce approximation in the circuit.

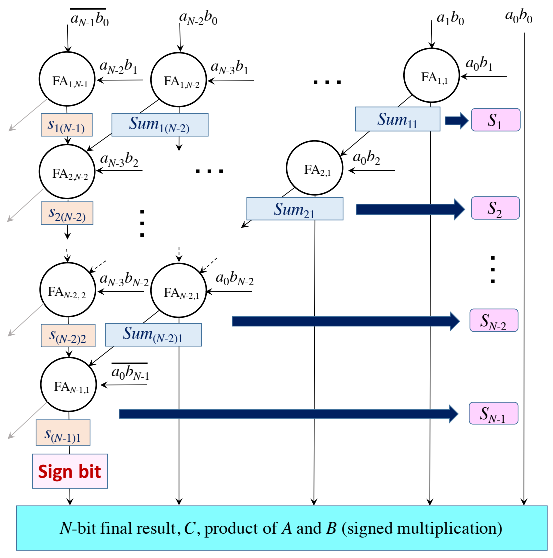


Fig. 5. A signed array multiplier where some of the LSB FAs can be approximated to introduce approximation in the circuit.

Additionally, we consider the correlation between the sum and carry out bits of any FA and the error introduced in them, by combining them into a two-bit output, Sum, and the corresponding error, ΔSum, both expressed in decimal, with their PMFs characterized by similar methods as Table I. This technique captures the interdependence of the two most highly correlated output bits of any FA. If a DAG has reconvergence that involves a larger logical depth comprising of such FAs, then there are more inputs that are involved in generating the corresponding output, and these other signals dilute the correlation. This concept of correlation dilution with logical depth has been used in power estimation too [25], where signal probabilities can be correlated due to reconvergent fanout, but if a reconvergent node has a larger transitive fanin, the role of correlations is significantly weakened. Hence, even if we assume independence of different FAs within the adder or multiplier array, it 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. However, we ignore the specific case of immediate reconvergent paths, for example, when two inputs of a node are additive or multiplicative inverse of each other. This is a limitation of our approach and the topic of ongoing research.

A. PMF of Generated Errors in Approximate Adders

For an adder, the FAs in the array are each indexed as FA$_i$, where i corresponds to the position of the FA in the adder, starting from the LSB side, so that i ranges from 1 to N for an N-bit signed adder as shown in Fig. 4. The two single bit outputs, s_i and $c_{out,i}$, from the i^{th} adder, FA$_i$, are combined to a two-bit output, Sum_i, and modeled as the random variable, $Sum_i = Sum_{i,t} + \Delta Sum_i$, where $Sum_{i,t}$ is the true sum (corresponding to the output of an exact FA), and ΔSum_i is the error due to the approximate addition, combining the error in both the output bits, similar to the example of the FA in Fig. 2. Although the structure shown in the figure corresponds to a Ripple Carry Adder, this concept can be extended to any adder architecture. Finding the error PMF for the adder involves two steps:

- **Step 1**: determining the signal probabilities for all inputs of each individual FA$_i$ in the adder, and using the approach in Section III to compute the PMF of ΔSum_i, $i \in [1, N-1]$, along with the PMF of Δs_N, and
- **Step 2**: finding the error PMF of the entire adder, i.e., the PMF of the weighted sum of the ΔSum_i variables along with that of Δs_N which is the error in the sum bit of the MSB (N^{th}) FA and is also the sign bit in the output.

Let us now explain each step in further detail:

Step 1: The first step involves probability propagation within a Boolean network, and we use established techniques for this purpose [26]. For example, if the two inputs, α and β, to a two-input AND gate have signal probabilities of p_α and p_β, respectively, then the probability of the output of the AND gate to be 1 is $p_\alpha p_\beta$. Similarly, we can compute the probability of each signal within the adder array in Fig. 4 to be 1 or 0, using the Boolean function of each FA. Based on this, we obtain the PMF of ΔSum_i, denoted by $f_{\Delta Sum_i}(n)$, as a sum of delta functions similar to Eq. (5). We also obtain the PMF of Δs_N, which is the error in the sum bit of the MSB FA, and denote it by $f_{\Delta s_N}(n)$ similar to Eq. (4).

Step 2: The generated error in the adder output, C (depicted in Fig. 4), is represented by ΔR, and is the weighted sum of the errors, ΔSum_i, over LSB $N-1$ FAs, and the error, Δs_N, in the sum bit of the MSB FA. Since we implement two's complement operation, Δs_N actually represents the error in the sign bit, and hence, has a negative weight associated with itself inside ΔR. A simple analysis yields:

$$\Delta R = \left(\sum_{i=1}^{N-1} 2^{i-1} \Delta Sum_i\right) - 2^{N-1}\Delta s_N \tag{7}$$

Since we consider the ΔSum_i, $i = 1, \cdots, N-1$, and Δs_N to be independent random variables, we utilize the fact that PMF of sum of independent random variables equals the convolution of the PMF of those variables. Hence, the PMF, $f_{\Delta R}(n)$, of the generated error, ΔR, at the adder node is obtained as:

$$f_{\Delta R}(n) = \left(\bigotimes_{i=1}^{N-1} f_{2^{i-1}\Delta Sum_i}(n)\right) \bigotimes f_{-2^{N-1}\Delta s_N}(n) \tag{8}$$

where $\bigotimes$ is the convolution operator. If the absolute value of the largest output error of FA$_i$ is M, using Eqs. (4) and (5), we obtain the constituent PMFs in Eq. (8) as:

$$f_{2^{i-1}\Delta Sum_i}(n) = \sum_{v=-M}^{M} e_v^{(i)} \delta(n - 2^{i-1}v) \tag{9}$$

$$f_{-2^{N-1}\Delta s_N}(n) = \sum_{v=-1}^{1} x_v^{(N)} \delta(n + 2^{N-1}v) \tag{10}$$

where $e_v^{(i)}$ and $x_v^{(N)}$ are the probabilities of ΔSum_i and Δs_N to be v, respectively, and the superscripts, (i) and (N), indicate the position in the error-producing FAs in the adder array. Since ΔSum_i (Δs_N) is shifted by 2^{i-1} (-2^{N-1}) in Eq. (9) (Eq. (10)), the locations of the impulses in the error distributions are scaled by this factor as well, as shown inside the Kronecker delta functions.

We implement the following techniques to solve the convolution problem in Eq. (8) to obtain the PMF of ΔR:

1) Use the Z-transform [19] to convert the convolution into a friendlier multiplication, yielding a polynomial in z. This polynomial can have an exponential number of terms, and special techniques are required to manage the cost of working in the transform domain.

2) Use the inverse fast Fourier transform (IFFT) [19] to infer the PMF of ΔR from the above polynomial.

Next, we explain each of these techniques in detail.

1) Representing the convolution using the Z-transform: Since the Z-transform of a convolution of functions in the original domain is equivalent to the product of the Z-transforms of those functions in the transform domain, we can represent the Z-transform of $f_{\Delta R}(n)$ in Eq. (8), by $F_{\Delta R}(z)$ as:

$$F_{\Delta R}(z) = \left(\prod_{i=1}^{N-1} F_{2^{i-1}\Delta Sum_i}(z)\right) F_{-2^{N-1}\Delta s_N}(z) \tag{11}$$

where $F_{2^{i-1}\Delta Sum_i}(z)$ and $F_{-2^{N-1}\Delta s_N}(z)$, are the Z-transform of the PMFs, $f_{2^{i-1}\Delta Sum_i}(n)$ and $f_{-2^{N-1}\Delta s_N}(n)$, from Eqs. (9) and (10), respectively. Applying the Z-transform to both sides of Eqs. (9) and (10), we obtain:

$$F_{2^{i-1}\Delta Sum_i}(z) = \sum_{v=-M}^{M} e_v^{(i)} z^{-2^{i-1}v} \tag{12}$$

$$F_{-2^{N-1}\Delta s_N}(z) = \sum_{v=-1}^{1} x_v^{(N)} z^{2^{N-1}v} \tag{13}$$

Substituting $F_{2^{i-1}\Delta Sum_i}(z)$ and $F_{-2^{N-1}\Delta s_N}(z)$ from Eqs. (12) and (13), respectively, in Eq. (11), we can rewrite $F_{\Delta R}(z)$ as:

$$\begin{aligned} F_{\Delta R}(z) &= \prod_{i=1}^{N-1}\left(\sum_{v=-M}^{M} e_v^{(i)} z^{-2^{i-1}v}\right)\left(\sum_{v=-1}^{1} x_v^{(N)} z^{2^{N-1}v}\right) && (14) \\ &= z^{-\mathcal{E}} \sum_{i=0}^{2\mathcal{E}} a_i z^i && (15) \\ &= z^{-\mathcal{E}} \Phi(z) && (16) \end{aligned}$$

where ΔR ranges from $-\mathcal{E}$ to $\mathcal{E}$, with $\mathcal{E} = 2^{N-1}(M+1) - M$, and the a_is are the coefficients of the polynomial in z, denoted by $\Phi(z)$ in Eq. (16). The derivation of $\mathcal{E}$ has been deferred to Appendix A for better readability. Performing the inverse Z-transform of Eq. (14), we obtain,

$$f_{\Delta R}(n) = \sum_{i=0}^{2\mathcal{E}} a_i \delta(n + i - \mathcal{E}) \tag{17}$$

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

While the above equations present 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, $i \in [0, 2\mathcal{E}]$, by multiplying the terms in Eq. (14). Therefore, we develop an efficient scheme for finding the coefficients, a_i as explained next.

2) Using the IFFT to infer $f_{\Delta R}(n)$ from $\Phi(z)$ and $F_{\Delta R}(z)$: So far we have worked in the Z-transform domain to formulate the error PMF, $F_{\Delta R}(z)$. Let us now consider discrete Fourier domain to determine the coefficients, a_i, in $\Phi(z)$ from Eq. (16), by using inverse fast Fourier transform (IFFT). The interchange of domains is possible since, by definition, the Z-transform is equivalent to the discrete time Fourier transform (DTFT) when the magnitude of $|z| = 1$ [19].

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

$$\mathcal{A}_k = \sum_{i=0}^{2\mathcal{E}} a_i \exp\left(-\mathbf{j}\frac{2\pi i k}{2\mathcal{E}+1}\right) = \sum_{i=0}^{2\mathcal{E}} a_i z_k^i \tag{18}$$

where $z_k = \exp\left(-\mathbf{j}\frac{2\pi k}{2\mathcal{E}+1}\right)$, $\mathbf{j} = \sqrt{-1}$, and $k \in [0, 2\mathcal{E}]$. It is interesting to note that the values of z_k are the reciprocal of the $(2\mathcal{E}+1)^{\text{th}}$ complex roots of unity. Therefore, if we evaluate $\Phi(z)$ in Eq. (16) by substituting $z = z_k$ for each $k \in [0, 2\mathcal{E}]$, we obtain the Fourier coefficient, $\mathcal{A}_k$. In other words,

$$\begin{aligned}\mathcal{A}_k &= \Phi(z_k) = z_k^{\mathcal{E}} F_{\Delta R}(z_k) \\ &= z_k^{\mathcal{E}} \prod_{i=1}^{N-1}\left(\sum_{v=-M}^{M} e_v^{(i)} z_k^{-2^{i-1}v}\right)\left(\sum_{v=-1}^{1} x_v^{(N)} z_k^{2^{N-1}v}\right)\end{aligned} \tag{19}$$

This provides us with the discrete Fourier coefficients of the sequence of a_is, which can then obtained by performing inverse discrete time Fourier transform (IDFT) of the $\mathcal{A}_k$s:

$$a_i = \frac{1}{2\mathcal{E}+1}\sum_{k=0}^{2\mathcal{E}} \mathcal{A}_k \exp\left(\mathbf{j}\frac{2\pi i k}{2\mathcal{E}+1}\right) \tag{20}$$

To compute the IDFT in Eq. (20) efficiently, we use the IFFT to obtain the values of the a_is. As observed in Eq. (17), obtaining the a_is directly provides the PMF of the generated error, ΔR, in the adder output.

We summarize the steps to obtain the PMF of an approximate adder (or subtractor) node in Algorithm 1.

Algorithm 1 Pseudo-code to obtain the error PMF of an approximate adder (or subtractor) node within a DAG.

```
Input: Adder architecture; truth tables of the FAs.
Input: Signal probabilities of each input bit of the adder.
Output: Error PMF, $f_{\Delta R}(n)$, of the adder output.
1: Compute the internal signal probabilities, $e_{v_1}^{(i)}$, for $i \in [1, N-1]$, $v_1 \in [-M, M]$, and $x_{v_2}^{(N)}$, $v_2 \in [-1, 1]$, from the bitwise input signal probabilities, similar to Table I
2: for each $i$ from 1 to $N-1$ do
3:     Compute $f_{2^{i-1}\Delta Sum_i}(n)$ from Eq. (9)
4: end for
5: Compute $f_{-2^{N-1}\Delta s_N}(n)$ from Eq. (10)
6: Formulate $f_{\Delta R}(n)$ as a convolution problem in Eq. (8)
7: To solve the convolution, perform Z-transform on $f_{\Delta R}(n)$ to formulate $F_{\Delta R}(z) = z^{-\mathcal{E}}\Phi(z)$ as Eq. (16), where $\mathcal{E}$ is derived in Appendix A
8: for each $k$ from 0 to $2\mathcal{E}$ do
9:     Compute $z_k = \exp\left(-\mathbf{j}\frac{2\pi k}{2\mathcal{E}+1}\right)$, $\mathbf{j} = \sqrt{-1}$
10:    Evaluate $\mathcal{A}_k = \Phi(z_k)$ as shown in Eq. (19)
11: end for
12: IFFT: Using the $\mathcal{A}_k$s, obtain $a_i$, $i \in [0, 2\mathcal{E}]$ from Eq. (20)
13: Obtain $f_{\Delta R}(n) = \sum_{i=0}^{2\mathcal{E}} a_i \delta(n + i - \mathcal{E})$ as Eq. (17)
```

B. PMF of Generated Errors in Approximate Multipliers

For a multiplier, the FAs are each indexed as FA$_{ij}$ in the array, where i corresponds to the row number, starting from the top ($i \in \{1, \cdots, N-1\}$), and j corresponds to the position of an FA in a particular row, starting from the LSB ($j \in \{1, \cdots, N-i\}$), as shown in Fig. 5. The output of FA$_{ij}$ is modeled as the random variable, $Sum_{ij} = Sum_{ij,t} + \Delta Sum_{ij}$, where $Sum_{ij,t}$ is the true sum (corresponding to the output of an exact FA), and ΔSum_{ij} is the error due to the approximate addition, similar to the example in Fig. 2.

Similar to the error PMF of an adder, finding the error PMF for the multiplier array involves three steps, while the extra step is due to the two-dimensional structure of the multiplier as opposed to the one-dimensional one for the adder:

- **Step 1**: determining the signal probabilities for all inputs of each individual FA$_{ij}$, and using the approach in Section III to compute the PMF of ΔSum_{ij} and $\Delta s_{i(N-i)}$, the latter being the error in the sign bit in the i^{th} row,
- **Step 2**: finding the PMF of the error, ΔS_i, introduced by the i^{th} row of the multiplier array, and
- **Step 3**: finding the PMF of the entire multiplier, i.e., the PMF of the sum of the ΔS_i variables over all rows, i.

Let us now explain each step in further detail:
Step 1: The first step is similar to that of the generated error PMF computation of adder as explained in the previous subsection, to obtain the PMF of ΔSum_{ij} and $\Delta s_{i(N-i)}$ denoted by $f_{\Delta Sum_{ij}}(n)$ and $f_{\Delta s_{i(N-i)}}(n)$, respectively.
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 $N-i$ FAs, as depicted in Fig. 5 for any general N, and $i \in \{1, \cdots, N-1\}$. For each row, $i \in \{1, \cdots, N-1\}$, a simple analysis yields:

$$\Delta S_i = \sum_{j=1}^{N-i-1} 2^{i+j-1}\Delta Sum_{ij} - 2^{N-1}\Delta s_{i(N-i)} \tag{21}$$

where the negative weight, -2^{N-1}, is associated with the sum bit, $s_{i(N-i)}$, in the i^{th} row, which actually corresponds to the the sign bit in the multiplier result.

Since we consider the ΔSum_{ij} and $\Delta s_{i(N-i)}$ random variables to be independent, 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:

$$f_{\Delta S_i}(n) = \bigotimes_{j=1}^{N-i-1} f_{2^{i+j-1}\Delta Sum_{ij}}(n) \bigotimes f_{-2^{N-1}\Delta s_{i(N-i)}}(n) \tag{22}$$

where $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$ and $f_{-2^{N-1}\Delta s_{i(N-i)}}(n)$ are the PMFs of the random variable, $2^{i+j-1}\Delta Sum_{ij}$ and $-2^{N-1}\Delta s_{i(N-i)}$, respectively.

If the absolute value of the largest output error of FA$_{ij}$ is M, then using Eqs. (4) and (5), we obtain the constituent PMFs in Eq. (22) similar to Eqs. (9) and (10) as:

$$f_{2^{i+j-1}\Delta Sum_{ij}}(n) = \sum_{v=-M}^{M} e_v^{(ij)} \delta(n - 2^{i+j-1}v) \tag{23}$$

$$f_{-2^{N-1}\Delta s_{i(N-i)}}(n) = \sum_{v=-1}^{1} x_v^{(i(N-i))} \delta(n + 2^{N-1}v) \tag{24}$$

where $e_v^{(ij)}$ and $x_v^{(i(N-i))}$ are the probabilities of ΔSum_{ij} and $\Delta s_{i(N-i)}$ to be v, respectively, and the superscripts, (ij) and $(i(N-i))$, indicate the position in the error-producing FAs in the multiplier array.

Step 3: The generated error, ΔR, in the multiplier output is simply the sum of the errors, ΔS_i, over all $N-1$ rows. Assuming the ΔS_i random variables to be independent, we obtain the error PMF, $f_{\Delta R}(n)$, by convolving the $f_{\Delta S_i}(n)$ PMFs from Eq. (22) as:

$$f_{\Delta R}(n) = \bigotimes_{i=1}^{N-1} \left(\bigotimes_{j=1}^{N-i-1} f_{2^{i+j-1}\Delta Sum_{ij}}(n) \right) \bigotimes f_{-2^{N-1}\Delta s_{i(N-i)}}(n) \tag{25}$$

The techniques to solve the convolution problem in Eq. (25) are similar to those for Eq. (8) for adders as explained in Sections IV-A1 and IV-A2. Hence, we first represent $F_{\Delta R}(z)$, the Z-transform of $f_{\Delta R}(n)$ in Eq. (25), as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N-1} \left(\prod_{j=1}^{N-i-1} F_{2^{i+j-1}\Delta Sum_{ij}}(z) \right) F_{-2^{N-1}\Delta s_{i(N-i)}}(z) \tag{26}$$

where $F_{2^{i+j-1}\Delta Sum_{ij}}(z)$ and $F_{-2^{N-1}\Delta s_{i(N-i)}}(z)$ are the Z-transforms of the PMFs, $f_{2^{i+j-1}\Delta Sum_{ij}}(n)$ and $f_{-2^{N-1}\Delta s_{i(N-i)}}(n)$, respectively, outlined in Eqs. (23) and (24). These two Z-transforms can be obtained similar to Eqs. (12) and (13), respectively, and can be substituted in Eq. (26), to rewrite Eq. (26) as:

$$F_{\Delta R}(z) = \prod_{i=1}^{N-1} \prod_{j=1}^{N-i-1} \left(\sum_{v=-M}^{M} e_v^{(ij)} z^{-2^{i+j-1}v} \right) \left(\sum_{v=-1}^{1} x_v^{(i(N-i))} z^{2^{N-1}v} \right) \tag{27}$$

$$= z^{-\mathcal{F}} \sum_{i=0}^{2\mathcal{F}} m_i z^i = z^{-\mathcal{F}} \Theta(z) \tag{28}$$

where ΔR ranges from $-\mathcal{F}$ to $\mathcal{F}$, with $\mathcal{F} = 2^{N-1}(MN - 3M + N - 1) + 2M$, and the m_is are the coefficients of the polynomial in z, denoted by $\Theta(z)$ in Eq. (28). The derivation of $\mathcal{F}$ has been deferred to Appendix B for better readability. Performing the inverse Z-transform of Eq. (27),

$$f_{\Delta R}(n) = \sum_{i=0}^{2\mathcal{F}} m_i \delta(n + i - \mathcal{F}) \tag{29}$$

Hence, m_i is the probability of the generated error, ΔR, in the multiplier output to be $-(i - \mathcal{F})$, and can be computed using the ideas presented in Section IV-A2.

C. *Complexity of PMF Computation of Generated Errors*

For N_a approximate bits in an N-bit adder, the computational complexity is $\mathcal{O}\left(N_a 2^{N_a}\right)$, and for an N-bit multiplier, it is $\mathcal{O}\left(N_a^2 2^{N_a}\right)$. Both the derivations are deferred to Appendix D for better readability.

V. Distribution of Output Error in a DAG

The previous section summarized the computation of the error PMFs for individual adder (or subtractor) and multiplier nodes of a DAG. In this section, we propose techniques to compute the error PMF for an entire DAG in general, comprising these nodes. In other words, in this section, we compose the block PMFs, as computed in Section IV, to determine the PMF for a DAG consisting of these blocks.

The error in the output, C, of any approximate node within a DAG comprises (a) the error generated at this node, and (b) the error propagated through the operands, A and B as illustrated in Fig. 6 for approximate adders and multipliers. We have already discussed the computation of the PMF of generated error in DAG nodes in Section IV. Here we present a methodology to obtain the PMF of the error, ΔC, at the output of a node combining the errors propagated through its inputs, A and B, and the errors, ΔR, generated at this node, as shown in Fig. 6.

To obtain the PMF of the total error, ΔC, propagated to the output of a node in a DAG, we need the PMF of the true value of the inputs, A_t and B_t, and the input errors, ΔA and ΔB. These are discrete random variables, and hence, their PMFs can be modeled as the sum of delta functions as:

$$f_{A_t}(n) = \sum_{-E}^{E} p_k^A \delta(n-k); \; f_{B_t}(n) = \sum_{-E}^{E} p_k^B \delta(n-k) \tag{30}$$

$$f_{\Delta A}(n) = \sum_{-F}^{F} p_k^{\Delta A} \delta(n-k); \; f_{\Delta B}(n) = \sum_{-F}^{F} p_k^{\Delta B} \delta(n-k) \tag{31}$$

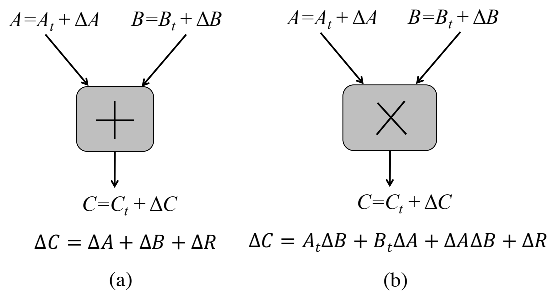


Fig. 6. (a) Approximate adder, and (b) multiplier with inputs, A and B, both of which can be erroneous, producing approximate output, C.

where the inputs (input errors) range from $-E$ to E ($-F$ to F), and p_k^X represents the probability of the variable, X, to be k, where $X \in \{A_t, B_t, \Delta A, \Delta B\}$. It is to be noted that we drop the subscript, t, from $p_k^{A_t}$ and $p_k^{B_t}$ in Eq. (30) and simply use p_k^A and p_k^B, instead, for simpler notation, and ΔC is shown in Figs. 6(a) and (b) for both adder and multiplier, whose PMF can be obtained using the PMFs listed in Eqs. (30) and (31), and the methods explained in the next two subsections.

A. Error Propagation through Adders

The total error, ΔC, at the output of an approximate adder is represented by $\Delta A+\Delta B+\Delta R$ as illustrated in Fig. 6(a). The generated error, ΔR, and both the input errors, ΔA and ΔB, can be assumed to be independent random variables. Hence, the PMF of ΔC can be formulated as:

$$f_{\Delta C}(n) = f_{\Delta A}(n) \bigotimes f_{\Delta B}(n) \bigotimes f_{\Delta R}(n) \tag{32}$$

where $f_{\Delta A}(n)$ and $f_{\Delta B}(n)$ are defined in Eq. (31), and $f_{\Delta R}(n)$ is obtained from Eq. (17) in Section IV-A. This convolution problem can be solved by using the concepts explained in Section IV-A, specifically, by first applying Z-transform on both sides of Eq. (32), followed by evaluating the resulting polynomial at the reciprocal of $(2\mathcal{E}_C + 1)^{\text{th}}$ roots of unity, where $\mathcal{E}_C$ is the maximum value of ΔC from Eq. (32), and then performing an IFFT on those evaluations.

Similarly, PMF of C_t can be obtained by convolving the PMFs of A_t and B_t defined in Eq. (30). This is the PMF of the output of the adder node if it had implemented an exact operation, and may be used to characterize error PMFs in successive stages within the DAG.

B. Error Propagation through Multipliers

While ΔC for an adder is given by a simple sum of three random variables, ΔC for a multiplier involves the sum of product of random variables ($A_t\Delta B$, $B_t\Delta A$, and $\Delta A\Delta B$). We will employ the Mellin transform [20] to first compute the PMF of the product of random variables followed by established techniques using the Fourier transform to compute the PMF of their sum. Hence, let us first provide a few key concepts of the Mellin transform that we will use to compute the PMF of the product of random variables.

Definition. *The Mellin transform is an integral transform that may be regarded as the multiplicative version of the Fourier transform [20]. It is applicable only for functions defined on the positive real axis. For two positive random variables, X and Y, with PMFs, $f_X(n)$ and $f_Y(n)$, respectively, the following statements hold true regarding their Mellin transforms:*

(i) The Mellin transform of $f_X(n)$ and $f_Y(n)$, are represented by $F_X(s)$ and $F_Y(s)$, respectively, as:

$$F_X(s) = \mathcal{M}[f_X(n); s] = \int_0^\infty n^{s-1} f_X(n) dn$$
$$F_Y(s) = \mathcal{M}[f_Y(n); s] = \int_0^\infty n^{s-1} f_Y(n) dn$$

where s is a complex variable that corresponds to the transformed domain.

(ii) Product Rule: If X and Y are independent random variables in addition to being positive, then

$$\begin{aligned} F_{XY}(s) &= \mathcal{M}[f_{XY}(n); s] = \mathcal{M}[f_X(n); s]\,\mathcal{M}[f_Y(n); s] \\ &= F_X(s)F_Y(s) \end{aligned} \tag{33}$$

(iii) If $f_X(n)$ is represented by a sum of delta functions, $\sum_{k=1}^{N} x_k\delta(n-k)$, its Mellin transform is obtained as:

$$\mathcal{M}\left[\sum_{k=1}^{N} x_k\delta(n-k);\ s\right] = \sum_{k=1}^{N} x_k k^{s-1} \tag{34}$$

Now, in the total error, ΔC, propagated from the inputs of an approximate multiplier to its output (illustrated in Fig. 6(b)), we can ignore $\Delta A\Delta B$, since it is negligible compared to the other terms, so that ΔC can be reformulated as:

$$\Delta C \approx A_t\Delta B + B_t\Delta A + \Delta R \tag{35}$$

If A_t, ΔB, B_t, and ΔA are all independent random variables, the PMF of ΔC can be formulated as:

$$f_{\Delta C}(n) = f_{A_t\Delta B}(n) \bigotimes f_{B_t\Delta A}(n) \bigotimes f_{\Delta R}(n) \tag{36}$$

where $f_{\Delta R}(n)$ is obtained from Eq. (29) in Section IV-B. To solve the convolution in Eq. (36), we need the PMFs of $A_t\Delta B$ and $B_t\Delta A$. This can achieved by using the *Product Rule* of the Mellin transform of PMFs, as explained above. However, the Mellin transform is only valid for positive random variables, while both A_t and ΔB (or B_t and ΔA) can be either positive or non-positive (negative or zero) since we implement signed operations. Hence, we use the positive and non-positive parts of X, $X \in \{A_t, B_t, \Delta A, \Delta B\}$, and apply the Mellin transform separately to handle the non-positive cases.

For example, to find the PMF of $C_1 = A_t\Delta B$, we compute four different PMFs for C_1, when (a) $A_t \geq 0$, $\Delta B \geq 0$, (b) $A_t{\geq}0$, $\Delta B{<}0$, (c) $A_t{<}0$, $\Delta B{\geq}0$, and (d) $A_t{<}0$, $\Delta B{<}0$, as sum of delta functions (impulse train), illustrated in Figs. 7(a)-(d). Clearly, the PMF of $C_1 = A_t\Delta B$ is the sum of all four impulse trains, and is depicted by $f_{C_1}(n)$ in Fig. 7(e). Similar logic is used to obtain the PMF of $B_t\Delta A$. Detailed derivation of these PMFs is deferred to Appendix C for better readability.

Since the PMF, $f_{\Delta R}(n)$, of the generated error, ΔR, from Eq. (29) can also be represented by an impulse train, we can use the concepts explained in Section IV-A, to solve the convolution problem in Eq. (36). Specifically, we can apply Z-transform on both sides of Eq. (36), followed by evaluating the resulting polynomial at the reciprocal of $(2\mathcal{F}_C + 1)^{\text{th}}$ roots of unity, where $\mathcal{F}_C$ is the maximum value of ΔC from Eq. (36), and then performing an IFFT on those evaluations.

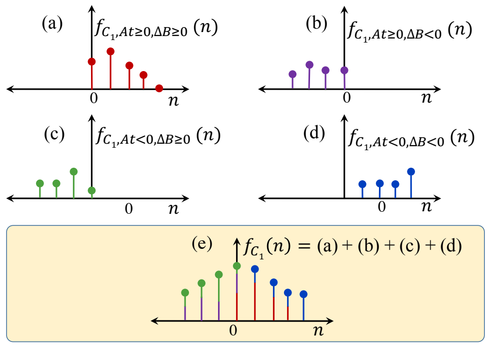

(a)
$f_{C_1,At\geq 0,\Delta B\geq 0}(n)$
(b)
$f_{C_1,At\geq 0,\Delta B<0}(n)$
(c)
$f_{C_1,At<0,\Delta B\geq 0}(n)$
(d)
$f_{C_1,At<0,\Delta B<0}(n)$
(e)
$f_{C_1}(n) = (a) + (b) + (c) + (d)$
0
n

Fig. 7. PMF of the product, C_1, as a sum of four PMFs depending on the sign of the two operands, A_t and ΔB.

C. Error PMF at Output Nodes of a DAG

We follow a multi-level approach to obtain the error PMF at the output node of an approximate DAG, where we first analyze adders and multipliers, as explained in the previous subsections, and then use the results to analyze the entire DAG. Hence, each node of an approximate DAG with V nodes can be traversed topologically while performing the unit operations of error generation and propagation through that node. The inputs connected to the primary input nodes of the DAG are error-free. The total error accumulates during the topological traversal, and appears as error at the primary output of the DAG. The computational complexity to obtain the error PMF when N_a bits are approximate in each of the N-bit wide nodes, is $\mathcal{O}\left(VN_a 2^{N+N_a}\right)$. The derivation is presented in Appendix D, and includes both the complexity of characterizing adders/multipliers, and that of traversing the DAG with V nodes.

VI. Experimental Setup and Results

We implemented the ideas presented here in C++, and the experiments were performed on a 64-bit Ubuntu server with a 3 GHz Intel® Core™2 Duo CPU E8400 processor. We use greyscale pixel value of the images from the LFW Face Database [27], to generate the inputs for performing Monte Carlo simulations against which we validate our proposed algorithm. This ensures that correlation exists in the input data. Mean subtraction is performed for each sample to visualize results for signed numbers, since pixel values range from 0 to 255. Finally, we use the FFTW library [28] to perform the fast Fourier transform related calculations in C++, for error PMF calculation of multiplier nodes. The approximate adders that constitute the adder and multiplier nodes within an approximate DAG are the various transistor level approximations of the mirror adders from [17].

A. PMF of Generated Error in Approximate Nodes

We consider 16-bit Ripple Carry Adders and Modified Baugh-Wooley Multipliers as the approximate nodes of a DAG, with the inputs from pixel values of facial images as explained earlier. The generated error distributions of 16-bit adders and 16-bit×16-bit multipliers are computed assuming four LSBs to be approximate by replacing the corresponding FAs by either of the five versions of the approximate FAs from [17]. We assume the bitwise input signal probabilities to be 0.5, since the distribution of the inputs till the most significant approximate bit is symmetric about zero. For internal nodes, although the inputs are not guaranteed to be symmetric, the propagated errors dominate the generated errors, and the signal probability assumption does not affect the results significantly.

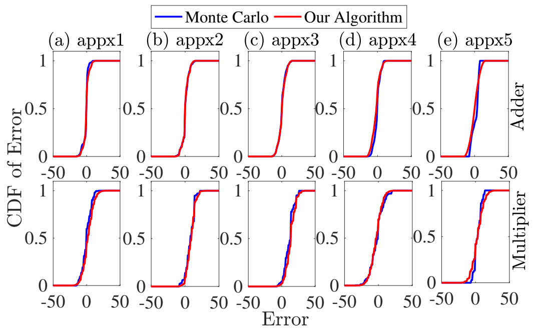


Fig. 8. Distribution of error in an 16-bit signed adder (top row) and 16-bit×16-bit signed multiplier (botom row) implemented with the appx1-appx5 versions of the FAs from [17].

We compare the cumulative distribution functions (CDFs) of the error PMFs obtained by our approach with those obtained by 6000 Monte Carlo simulations in Fig. 8 for adders (top row) and multipliers (bottom row). Each column of Fig. 8 indicated by (a)-(e), corresponds to the error CDF for an adder or a multiplier, when implemented using the approximate FAs from [17]. Clearly, the estimated distributions show excellent match with those obtained from Monte Carlo simulations.

To compare the statistics, we normalize the output errors to the dynamic range of the output of the approximate adder and multiplier since the same magnitude of error may have different levels of severity depending on the magnitude of the output. The normalization factor, $\mathcal{R}$, is the total range (difference of maximum and minimum values) of the output when implemented using different combinations of approximate and accurate FAs. We compare the standard deviation of the error PMFs obtained by our algorithm and Monte Carlo simulations, using the absolute value of the normalized percentage error in standard deviation, $\Delta\sigma_{norm}$, defined as:

$$\Delta\sigma_{norm} = 100 \times \frac{|\sigma_{est} - \sigma_{MC}|}{\mathcal{R}} \tag{37}$$

where σ_{est} and σ_{MC} are the standard deviations of the generated error PMF at adder/multiplier output, estimated by our algorithm, and through Monte Carlo simulations, respectively, and $\mathcal{R}$ is the normalizing factor defined earlier.

TABLE II
Normalized percentage errors in estimated standard deviations ($\Delta\sigma_{norm}$) for approximate adders and multipliers.

FA type → **DAG↓**	appx1	appx2	appx3	appx4	appx5
Adder	0.19%	0.11%	0.09%	0.58%	0.89%
Multiplier	0.07%	0.02%	0.00%	0.07%	0.21%

Table II lists the $\Delta\sigma_{norm}$ values of the 16-bit adders and multipliers in second and third columns, respectively, when approximation is introduced by replacing four LSB FAs by each approximate version of the FA from [17]. Clearly, the error statistics obtained by our algorithm show an excellent match with those obtained by Monte Carlo simulations, as indicated by the negligible errors in Table II.

B. Output Distribution in DAGs

We consider three different DAGs for image processing applications, namely, MPEG Decoder (MPEG), Matrix Multiplier (MM), and Horner-Bezier Filter (HB), from the ExpressDFG Benchmark Suite [21], comprising of adder and multiplier nodes, for demonstrating our algorithm. We observe the error PMFs in the deepest output node of each DAG, the details of which are outlined in Table III. A comprehensive description of these DAGs can be found in [21].

TABLE III
DESCRIPTION OF THE LONGEST PATHS OF DAGS USED IN THE RESULTS.

DAG $\rightarrow$ **Features**$\downarrow$	MPEG	MM	HB
Depth of output	5	5	7
Number of multipliers	3	4	5
Number of adders	3	3	5

The number of approximate bits or the type of approximate FAs within each node of a DAG is a user-specified input. Here we assume all the nodes except those connected to the primary inputs to be approximated by each type of FAs, having the same number of approximate LSBs (25%), similar to the approaches in [17], and each node is 16-bit wide. As mentioned earlier, the inputs for each DAG are the mean-subtracted pixel values of facial images. The error CDFs corresponding to the DAGs are illustrated in Fig. 9, where each column of subplots (a)-(e), corresponds to the type of FAs from [17] used to implement the approximate nodes within the respective DAG. Each row of the subplots corresponds to the different DAGs under study.

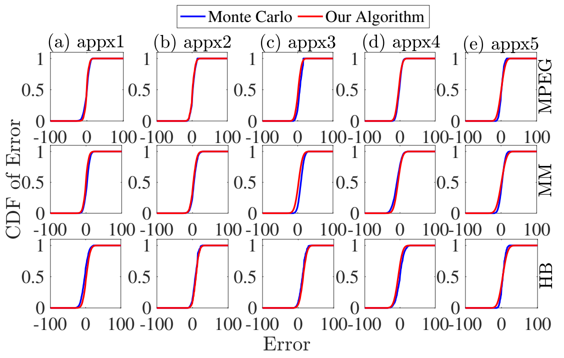


Fig. 9. Distribution of error in the deepest output node of the DAGs consisting of 16-bit adders and multipliers implemented with the appx1-appx5 versions of the FAs from [17].

By performing the Monte Carlo simulations multiple times with a very large sample size for one DAG, we obtain the mean and standard deviation of the errors to be 2 and 39, respectively. So to achieve 95% confidence such that the statistics from Monte Carlo PMF using n samples will not differ from their true values by more than 5%, we should select $n = \left(\frac{100\times 0.196\times 39}{5\times 2}\right)^2 = 5843$, based on [29]. Hence, we round it off to 6000, and use the same number of samples for all five DAGs. The error distribution obtained by 6000 Monte Carlo simulations is assumed to be the reference. Clearly from Fig. 9, the CDFs obtained by our algorithm show a very good match with those obtained through Monte Carlo simulations.

To quantify the Monte-Carlo like accuracy of our algorithm, we compute the Hellinger distance [30] between the estimated and the Monte Carlo PMFs, and it is defined as:

$$\text{Hellinger distance} = \frac{1}{\sqrt{2}}\sqrt{\sum_{\text{all } n}\left(\sqrt{\hat{f}(n)} - \sqrt{f(n)}\right)^2} \quad (38)$$

where $\hat{f}(n)$ and $f(n)$ are the estimated and the Monte Carlo PMFs, respectively. The factor, $\sqrt{2}$, ensures that this distance ranges from 0 to 1. The threshold value to judge the closeness of the two PMFs so that below this value, the two PMFs can be practically assumed to be the same is chosen to be 0.26 [31]. We plot these distances between our estimated error PMFs and the corresponding PMFs from Monte Carlo simulations in Fig. 11(a), and for most cases, they are less than the threshold.

Next, to highlight the heterogeneity in errors with more aggressive approximation, we plot the error CDFs in MPEG as an example, implemented by the appx1 FA, for different number of approximate bits, N_a, in Fig. 10, with the same inputs. Evidently from the figure, our method matches results from Monte Carlo simulations very well even with high degrees of approximation, and the Hellinger distance between the two is less than the threshold value of 0.26.

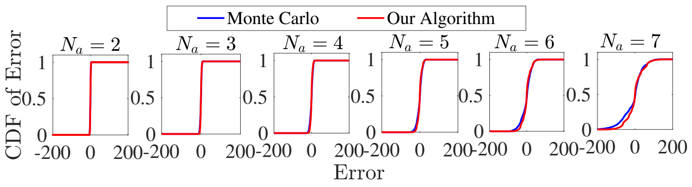


Fig. 10. Error CDFs for MPEG with appx1 version of the FA from [17], and N_a approximate bits, $N_a \in [2, 7]$ for 16-bit operands.

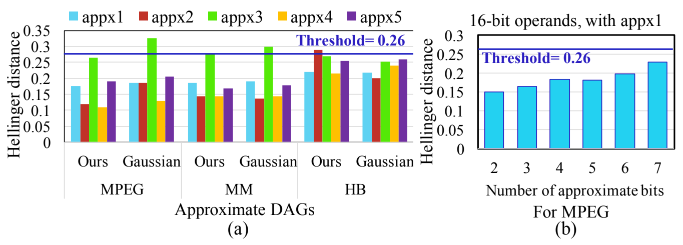


Fig. 11. (a) Hellinger distance between PMFs obtained by our approach and Gaussian approximation, as compared to the Monte Carlo PMFs. (b) Hellinger distance between PMFs by our approach and Monte Carlo for MPEG for different number of approximate bits implemented by appx1 FA.

Now, even for correlated inputs, if we decompose them into their binary equivalent, and plot the values of the lower order bits, we can observe them to be uniformly distributed without any correlation. Intuitively, this corresponds to the notion that the major information content is held by the higher order bits, where the effects of correlation tend to show up. As a result, the lower order bits can be assumed to practically independent.

So our assumption of input independence, provides accurate results when the lower order bits are approximated, and the estimated error can track the actual error very well. If more bits are approximated, then our assumptions can lead to pessimistic estimates of error, which is reflected through the increasing mismatch in the CDFs and the increasing Hellinger distances in Figs. 10 and 11(b), respectively. However, the match is good enough up until 50% of the bit length, which is the typical approximation range [3], [17] to limit the total error.

For comparison purposes in Fig. 11(a), we also construct Gaussian PMFs with the error mean and variances obtained by simply computing the statistics at the output of each node (instead of the entire PMF), and then plot the Hellinger distances between these PMFs and the Monte Carlo PMFs. This exercise gives an indication that although the error statistics may be the same for the artificially generated PMFs, the error PMF is not completely Gaussian for many cases, as indicated by the threshold violation in Fig. 11 for DAGs where our method generates a more accurate error PMF.

TABLE IV
NORMALIZED PERCENTAGE ERRORS IN ESTIMATED STANDARD DEVIATIONS ($\Delta\sigma_{norm}$ DEFINED IN EQ. (37)) AND PSNR VALUES.

DAG↓	**FA type→**	appx1	appx2	appx3	appx4	appx5
MPEG	$\Delta\sigma_{norm}$	0.01%	0.02%	0.03%	0.05%	0.12%
	PSNR$_{est}$	58.64	57.74	56.73	54.94	56.14
	PSNR$_{MC}$	55.92	56.61	54.23	56.28	57.37
MM	$\Delta\sigma_{norm}$	0.02%	0.01%	0.02%	0.04%	0.13%
	PSNR$_{est}$	59.33	57.35	56.66	55.15	56.04
	PSNR$_{MC}$	58.02	56.46	54.10	52.96	57.50
HB	$\Delta\sigma_{norm}$	0.00%	0.00%	0.00%	0.01%	0.01%
	PSNR$_{est}$	69.29	66.28	63.43	67.09	68.79
	PSNR$_{MC}$	68.24	66.91	64.48	67.34	69.82

Finally, we summarize $\Delta\sigma_{norm}$ in Table IV, corresponding to the error PMFs of the DAGs, for each FA from [17]. To show the impact of the approximation error at the application level, we also list the peak signal to noise ratios (PSNR) of the output (in dB) in the table, using both the estimated error PMF, and Monte Carlo simulations, denoted by PSNR$_{est}$ and PSNR$_{MC}$, respectively. Clearly, the $\Delta\sigma_{norm}$ indicates an excellent match between our approach and the simulations. Additionally, the PSNR$_{est}$ values are very close to the corresponding PSNR$_{MC}$ values, thus emphasizing the accuracy of our method at the application level.

VII. CONCLUSION

We have proposed a novel technique in this paper to analytically construct the PMF of errors generated due to approximating various adder and multiplier nodes in a DAG, using the concepts of transform calculus. We use the Fourier transform to propagate the errors through the adder nodes, while we implement a combination of the Fourier and the Mellin transform, to propagate the errors through multiplier nodes within a DAG. The novelty of employing Mellin transform lies in its equivalence to the Fourier transform to compute the PMF of the product of random variables. Our technique generates the error PMF with the accuracy of the traditional Monte Carlo simulations, as observed through the Hellinger distance metric as well as the difference in normalized standard deviations between our estimated and the Monte Carlo simulations.

APPENDIX A

Derivation of $\mathcal{E}$ in Eq. (15)**:** In the expansion of Eq. (14), $\mathcal{E}$ is the magnitude of the exponent of z corresponding to the term with the most negative exponent. Let this term be $T_{\mathcal{E}}$:

$$\begin{aligned} T_{\mathcal{E}} &= \left(\prod_{i=1}^{N-1} e_M^{(i)} z^{-2^{i-1}M}\right) x_{-1}^N z^{-2^{N-1}} \\ &= \left(x_{-1}^N \prod_{i=1}^{N-1} e_M^{(i)}\right) z^{-M(2^0+2^1+\cdots+2^{N-2})} z^{-2^{N-1}} \end{aligned} \tag{39}$$

Hence, $\mathcal{E}$ is obtained by adding the exponents of z in $T_{\mathcal{E}}$, as:

$$\mathcal{E} = M(2^{N-1}-1) + 2^{N-1} = 2^{N-1}(M+1) - M \tag{40}$$

APPENDIX B

Derivation of $\mathcal{F}$ in Eq. (28)**:** In the expansion of Eq. (27), $\mathcal{F}$ is the magnitude of the exponent of z corresponding to the term with the most negative exponent. Let this term be $T_{\mathcal{F}}$:

$$T_{\mathcal{F}} = \prod_{i=1}^{N-1}\left(\prod_{j=1}^{N-i-1} e_M^{(ij)} z^{-2^{i+j-1}M}\right)\left(x_{-1}^{(i(N-i))} z^{-2^{N-1}}\right) \tag{41}$$

$$\begin{aligned} \text{Hence, } \mathcal{F} &= M(2^1+\cdots+2^{N-2}) + M(2^2+\cdots+2^{N-2}) + \cdots \\ &\quad + M(2^{N-2}) + (N-1)2^{N-1} \end{aligned} \tag{42}$$

Simplifying, $\mathcal{F}$ is obtained as:

$$\mathcal{F} = 2^{N-1}(MN - 3M + N - 1) + 2M \tag{43}$$

APPENDIX C

Derivation of PMF of $A_t\Delta B$ and $B_t\Delta A$ in Eq. (36)**:** As evident from Eqs. (30) and (31), the inputs (input errors) range from $-E$ to E ($-F$ to F). However, since by definition, the Mellin transform is only applicable on positive random variables, the domains of A_t, B_t, ΔA, and ΔB must be split into positive and non-positive parts, and treated separately. Hence, we rewrite the PMFs in Eqs. (30) and (31) as:

$$\begin{aligned} f_{A_t}(n) &= \sum_{-E}^{-1} p_k^A \delta(n-k) + \sum_{1}^{E} p_k^A \delta(n-k) + p_0^A \delta(n) \\ &= f_{A_{neg}}(n) + f_{A_{pos}}(n) + f_{A_t}(0) \end{aligned} \tag{44}$$

$$\begin{aligned} f_{B_t}(n) &= \sum_{-E}^{-1} p_k^B \delta(n-k) + \sum_{1}^{E} p_k^B \delta(n-k) + p_0^B \delta(n) \\ &= f_{B_{neg}}(n) + f_{B_{pos}}(n) + f_{B_t}(0) \end{aligned} \tag{45}$$

$$\begin{aligned} f_{\Delta A}(n) &= \sum_{-F}^{-1} p_k^{\Delta A} \delta(n-k) + \sum_{1}^{F} p_k^{\Delta A} \delta(n-k) + p_0^{\Delta A} \delta(n) \\ &= f_{\Delta A_{neg}}(n) + f_{\Delta A_{pos}}(n) + f_{\Delta A}(0) \end{aligned} \tag{46}$$

$$\begin{aligned} f_{\Delta B}(n) &= \sum_{-F}^{-1} p_k^{\Delta B} \delta(n-k) + \sum_{1}^{F} p_k^{\Delta B} \delta(n-k) + p_0^{\Delta B} \delta(n) \\ &= f_{\Delta B_{neg}}(n) + f_{\Delta B_{pos}}(n) + f_{\Delta B}(0) \end{aligned} \tag{47}$$

where p_k^X represents the probability of the variable, X, to be k, $X \in \{A_t, B_t, \Delta A, \Delta B\}$. The new variables, X_{neg} and X_{pos}, correspond to the negative and positive parts of X, and will be used to obtain the PMFs of $A_t\Delta B$ (and $B_t\Delta A$).

To simplify our notations, let us define a new variable, $C_1 = A_t\Delta B$, whose PMF, $f_{C_1}(n)$, comprises the following four functions, each representing the probability of C_1 to be n based on the positivity or negativity of A_t and ΔB.

- $f_{C_1,A_t\geq 0,\Delta B\geq 0}(n)$ is the probability of C_1 to be n when both A_t and ΔB are positive.
- $f_{C_1,A_t\geq 0,\Delta B<0}(n)$ is the probability of C_1 to be n when A_t (ΔB) is positive (negative).
- $f_{C_1,A_t<0,\Delta B\geq 0}(n)$ is the probability of C_1 to be n when A_t (ΔB) is negative (positive).
- $f_{C_1,A_t<0,\Delta B<0}(n)$ is the probability of C_1 to be n when both A_t and ΔB are negative.

If each of the above functions can be represented by the impulse train as shown in Figs. 7(a)–(d), for example, then $f_{C_1}(n)$ is simply the summation of them, as depicted in Fig. 7(e), and hence, given by:

$$\begin{aligned} f_{C_1}(n) &= f_{C_1,A_t\geq 0,\Delta B\geq 0}(n) + f_{C_1,A_t\geq 0,\Delta B<0}(n) \\ &\quad + f_{C_1,A_t<0,\Delta B\geq 0}(n) + f_{C_1,A_t<0,\Delta B<0}(n) \end{aligned} \tag{48}$$

We had separated the negative and positive parts of A_t and ΔB in Eqs. (44) and (47), to define new random variables, X_{neg} and X_{pos} where $X = \{A_t, \Delta B\}$, and since the Mellin transform can only be applied when the random variables are positive, we inverse the domain of the PMFs when either of the multiplicands, A_t or ΔB, is negative. We also consider the cases differently when either of them is zero. Hence, we perform the following replacements to the terms in Eq. (48):

$$\begin{aligned} f_{C_1,A_t\geq 0,\Delta B\geq 0}(n) &= f_{C_1,A_t=A_{pos},\Delta B=\Delta B_{pos}}(n) \\ &\quad + f_{C_1,A_t=0 \text{ or } \Delta B=0}(n) \end{aligned} \tag{49}$$

$$\begin{aligned} f_{C_1,A_t\geq 0,\Delta B<0}(n) &= \mathcal{I}(f_{C_1,A_t=A_{pos},\Delta B=-\Delta B_{neg}}(n) \\ &\quad + f_{C_1,A_t=0,\Delta B=-\Delta B_{neg}}(n)) \end{aligned} \tag{50}$$

$$\begin{aligned} f_{C_1,A_t<0,\Delta B\geq 0}(n) &= \mathcal{I}(f_{C_1,A_t=-A_{neg},\Delta B=\Delta B_{pos}}(n) \\ &\quad + f_{C_1,A_t=-A_{neg},\Delta B=0}(n)) \end{aligned} \tag{51}$$

$$f_{C_1,A_t<0,\Delta B<0}(n) = f_{C_1,A_t=-A_{neg},\Delta B=-\Delta B_{neg}}(n) \tag{52}$$

where $\mathcal{I}(.)$ represents the operation of taking the mirror image of the enclosed function about the $n=0$ axis in Fig. 7. We can use the Mellin transform to rewrite Eqs. (49) to (52) as:

$$\begin{aligned} &f_{C_1,A_t\geq 0,\Delta B\geq 0}(n) \\ &= \mathcal{M}^{-1}\big(\mathcal{M}[f_{A_{pos}}(n);s]\mathcal{M}[f_{\Delta B_{pos}}(n);s]\big)+\psi_1 \end{aligned} \tag{53}$$

$$\begin{aligned} &f_{C_1,A_t\geq 0,\Delta B<0}(n) \\ &= \mathcal{I}\left(\mathcal{M}^{-1}\big(\mathcal{M}[f_{A_{pos}}(n);s]\mathcal{M}[f_{-\Delta B_{neg}}(n);s]\big)+\psi_2\right) \end{aligned} \tag{54}$$

$$\begin{aligned} &f_{C_1,A_t<0,\Delta B\geq 0}(n) \\ &= \mathcal{I}\left(\mathcal{M}^{-1}\big(\mathcal{M}[f_{-A_{neg}}(n);s]\mathcal{M}[f_{\Delta B_{pos}}(n);s]\big)+\psi_3\right) \end{aligned} \tag{55}$$

$$\begin{aligned} &f_{C_1,A_t<0,\Delta B<0}(n) \\ &= \mathcal{M}^{-1}\big(\mathcal{M}[f_{-A_{neg}}(n);s]\mathcal{M}[f_{-\Delta B_{neg}}(n);s]\big) \end{aligned} \tag{56}$$

where $\mathcal{M}^{-1}(.)$ denotes the inverse Mellin transform, and ψ_i, $i = 1\cdots 3$, covers the cases when A_t and ΔB are zeros in Eqs. (49) to (51). Detailed solutions of Eqs. (53) to (56) using Eqs. (44) and (47), are provided below:

$\underline{f_{C_1,A_t\geq 0,\Delta B\geq 0}(n)}$: The two terms in Eq. (53) are,

$$\begin{aligned} &\mathcal{M}^{-1}\big(\mathcal{M}[f_{A_{pos}}(n);s]\mathcal{M}[f_{\Delta B_{pos}}(n);s]\big) \\ &= \mathcal{M}^{-1}\left(\left(\sum_{k=1}^{E} p_k^A k^{s-1}\right)\left(\sum_{k=1}^{F} p_k^{\Delta B} k^{s-1}\right)\right) \\ &= \mathcal{M}^{-1}\left(\sum_{k=1}^{EF} c_k^{pp} k^{s-1}\right) = \sum_{k=1}^{EF} c_k^{pp}\delta(n-k) \end{aligned} \tag{57}$$

$$\text{and } \psi_1 = c_0^{pp}\delta(n) \tag{58}$$

where c_k^{pp} can be computed as:

$$c_k^{pp} = \begin{cases} p_0^A \sum_{k=0}^{F} p_k^{\Delta B} + p_0^{\Delta B}\sum_{k=0}^{E} p_k^A - p_0^A p_0^{\Delta B}, & k=0 \\ \sum_{i \text{ factor of } k} p_i^A p_{k/i}^{\Delta B}, & \text{otherwise} \end{cases}$$

Thus $f_{C_1,A_t\geq 0,\Delta B\geq 0}(n)$ from Eq. (49) is obtained as:

$$f_{C_1,A_t\geq 0,\Delta B\geq 0}(n) = \sum_{k=0}^{EF} c_k^{pp}\delta(n-k) \tag{59}$$

$\underline{f_{C_1,A_t\geq 0,\Delta B<0}(n)}$: The two terms inside $\mathcal{I}(.)$ in Eq. (54) are,

$$\begin{aligned} &\mathcal{M}^{-1}\left(\mathcal{M}[f_{A_{pos}}(n);s]\mathcal{M}[f_{-\Delta B_{neg}}(n);s]\right) \\ &= \mathcal{M}^{-1}\left(\left(\sum_{k=1}^{E} p_k^A k^{s-1}\right)\left(\sum_{k=1}^{F} p_{-k}^{\Delta B} k^{s-1}\right)\right) \\ &= \mathcal{M}^{-1}\left(\sum_{k=1}^{EF} c_k^{pn} k^{s-1}\right) = \sum_{k=1}^{EF} c_k^{pn}\delta(n-k) \end{aligned} \tag{60}$$

$$\text{and } \psi_2 = c_0^{pn}\delta(n) \tag{61}$$

where c_k^{pn} can be computed as:

$$c_k^{pn} = \begin{cases} p_0^A \sum_{k=1}^{F} p_{-k}^{\Delta B}, & k=0 \\ \sum_{i \text{ factor of } k} p_i^A p_{-k/i}^{\Delta B}, & \text{otherwise} \end{cases}$$

Thus $f_{C_1,A_t\geq 0,\Delta B<0}(n)$ from Eq. (50) is obtained as:

$$f_{C_1,A_t\geq 0,\Delta B<0}(n) = \mathcal{I}\left(\sum_{k=0}^{EF} c_k^{pn}\delta(n-k)\right) = \sum_{k=0}^{EF} c_k^{pn}\delta(n+k) \tag{62}$$

$\underline{f_{C_1,A_t<0,\Delta B\geq 0}(n)}$: The two terms inside $\mathcal{I}(.)$ in Eq. (55) are,

$$\begin{aligned} &\mathcal{M}^{-1}\left(\mathcal{M}[f_{-A_{neg}}(n);s]\mathcal{M}[f_{\Delta B_{pos}}(n);s]\right) \\ &= \mathcal{M}^{-1}\left(\left(\sum_{k=1}^{E} p_{-k}^A k^{s-1}\right)\left(\sum_{k=1}^{F} p_k^{\Delta B} k^{s-1}\right)\right) \\ &= \mathcal{M}^{-1}\left(\sum_{k=1}^{EF} c_k^{np} k^{s-1}\right) = \sum_{k=1}^{EF} c_k^{np}\delta(n-k) \end{aligned} \tag{63}$$

$$\text{and } \psi_3 = c_0^{np}\delta(n) \tag{64}$$

where c_k^{np} can be computed as:

$$c_k^{pn} = \begin{cases} p_0^{\Delta B} \sum_{k=1}^{E} p_{-k}^A, & k=0 \\ \sum_{i \text{ factor of } k} p_{-i}^A p_{k/i}^{\Delta B}, & \text{otherwise} \end{cases}$$

Thus $f_{C_1,A_t<0,\Delta B\geq 0}(n)$ from Eq. (51) is obtained as:

$$f_{C_1,A_t<0,\Delta B\geq 0}(n)=\mathcal{I}\left(\sum_{k=0}^{EF}c_k^{np}\delta(n-k)\right)=\sum_{k=0}^{EF}c_k^{np}\delta(n+k) \quad (65)$$

$\underline{f_{C_1,A_t<0,\Delta B<0}(n)}$: Eq. (56) are,

$$\begin{aligned}&\mathcal{M}^{-1}\left(\mathcal{M}[f_{-A_{neg}}(n);s]\mathcal{M}[f_{-\Delta B_{neg}}(n);s]\right)\\&=\mathcal{M}^{-1}\left(\left(\sum_{k=1}^{E}p_{-k}^{A}k^{s-1}\right)\left(\sum_{k=1}^{F}p_{-k}^{\Delta B}k^{s-1}\right)\right)\\&=\mathcal{M}^{-1}\left(\sum_{k=1}^{EF}c_k^{nn}k^{s-1}\right)=\sum_{k=1}^{EF}c_k^{nn}\delta(n-k) \quad (66)\end{aligned}$$

where c_k^{nn} can be computed as:

$$c_k^{nn}=\sum_{i \text{ factor of } k}p_{-i}^{A}p_{-k/i}^{\Delta B}$$

Since A_t and ΔB are strictly negative here, C_1 is non-zero, and $f_{C_1,A_t<0,\Delta B<0}(n)$ from Eq. (51) is obtained as:

$$f_{C_1,A_t<0,\Delta B<0}(n)=\sum_{k=1}^{EF}c_k^{nn}\delta(n-k) \quad (67)$$

The impulse trains in Eqs. (59), (62), (65), and (67) are schematically depicted in Figs. 7(a)–(d), and can be simply added to obtain $f_{C_1}(n)=f_{A_t\Delta B}(n)$ as shown in Fig. 7(e).

Finally, by exchanging A_t and ΔB with B_t and ΔA, respectively, in the above discussion, and using the definitions of X_{neg} and X_{pos}, $X\in\{B_t,\Delta A\}$ from Eqs. (45) and (46), respectively, we can obtain the PMF, $f_{B_t\Delta A}(n)$, of $B_t\Delta A$ too.

Appendix D

Derivation of computational complexity of error PMFs: The complexity of the algorithm arises from the PMF computation of both the generated and the propagated error within, and through the adders or/and multipliers of the DAG. We use the fact that the convolution of two vectors of length, n, takes $\mathcal{O}(n\log n)$ operations using FFT.

Generated errors in N-bit adders: If the error in a stand-alone FA ranges from $-M$ to M, then it ranges from -2^iM to 2^iM if the FA is located at the i^{th} bit in the FA array of the adder, $i\in[0,N_a-1]$, where N_a is the number of approximate bits in the adder. Hence, the convolution of error PMF at the output of the i^{th} and $(i+1)^{\text{th}}$ FAs requires $\mathcal{O}\left(\left(2^{(i+1)}M+1\right)\log\left(2^{(i+1)}M+1\right)\right)$ operations, starting with the second to LSB FA, i.e., $i\geq 1$. We formulate the number of computations to obtain the required PMF, by $T_A^{(g)}$:

$$\begin{aligned}T_A^{(g)}&\approx\mathcal{O}\left(\sum_{i=2}^{N_a}2^iM\log\left(2^iM\right)\right)\\&=\mathcal{O}\left(M\left(\sum_{i=2}^{N_a}2^i\right)\log M+M\sum_{i=2}^{N_a}i2^i+\log\left(2^{\sum_{i=2}^{N_a}i}M^{N_a-1}\right)\right)\\&=\mathcal{O}\left(N_a2^{N_a}\right) \quad (68)\end{aligned}$$

Generated errors in $N\times N$-bit multipliers: Due to the two-dimensional structure, convolutions are performed in each row of the multiplier, and the LSB is error-free. Similar to the adders, we can formulate the number of computations to obtain the generated error PMF in multipliers, by $T_M^{(g)}$:

$$\begin{aligned}T_M^{(g)}&=\mathcal{O}\left(\sum_{j=0}^{N_a-1}\sum_{i=2}^{N_a-1-j}\left(2^{(i+j+1)}M+1\right)\log\left(2^{(i+j+1)}M+1\right)\right)\\&\approx\mathcal{O}\left(\sum_{j=0}^{N_a-1}\sum_{i=3}^{N_a-j}\left(2^{(i+j)}M\right)\log\left(2^{(i+j)}M\right)\right)\\&=\mathcal{O}\left(\left(4M\log M\right)\sum_{k=1}^{N-2}k2^k+4M\sum_{k=1}^{N_a-2}k(k+2)2^k\right) \quad (69)\end{aligned}$$

To solve the Eq (69), we first derive the following key relations using the concepts of mathematical progressions:

$$\sum_{k=1}^{N_a-2}k2^k=(N_a-3)2^{N_a-1}+2;\ \sum_{k=1}^{N_a-2}k^22^k=(N_a^2-6N_a+3)2^{N_a-1}-6$$

Then, after simplification, we obtain $T_M^{(g)}\approx\mathcal{O}\left(N_a^22^{N_a}\right)$.

Propagated error PMF through the entire DAG: For V_A adder nodes in a DAG, the propagated error PMF can be obtained by convolving the generated error PMFs in each node. These errors range from $-\mathcal{E}$ to $\mathcal{E}$, where $\mathcal{E}=M(2^{N_a-1}-1)+2^{N_a-1}$, as derived in Appendix A. For $V_A=2$, this requires $\mathcal{O}(2\mathcal{E}\log 2\mathcal{E})$ operations, with the error ranging from $-2\mathcal{E}$ to $2\mathcal{E}$, and the corresponding PMF has $4\mathcal{E}$ stems. However, this range can be binned suitably so that the error PMF has $2\mathcal{E}$ stems, instead. Thus for any general V_A, the error PMF can be obtained in $T_A^{(p)}$ operations, which is simplified as:

$$T_A^{(p)}=\mathcal{O}\left((V_A-1)2\mathcal{E}\log 2\mathcal{E}\right)\approx\mathcal{O}\left(V_AN_a2^{N_a}\right) \quad (70)$$

Next, let us consider a multiplier with two erroneous inputs, $A_t+\Delta A$ and $B_t+\Delta B$ in the DAG, coming from preceding multiplier nodes, which are directly connected to the error-free primary inputs. Each input to this node can range from -2^{N-1} to 2^{N-1}, and the corresponding errors, from $-\mathcal{F}$ to $\mathcal{F}$, where $\mathcal{F}=2^{N_a-1}(MN_a-3M+N_a-1)+2M$, as derived in Appendix B. There are three steps for the propagated error PMF computation, and at the end of each, we bin the domain of the PMFs into $2\mathcal{F}$ stems:

1) PMFs of $A_t\Delta B$, $B_t\Delta A$: This takes $\mathcal{O}\left(4\mathcal{F}2^N\right)$ operations by the Mellin transform.
2) PMF of $(A_t\Delta B+B_t\Delta A)$: Using the binned PMFs, this takes $\mathcal{O}\left(2\mathcal{F}\log 2\mathcal{F}\right)$ operations.
3) PMF of $(A_t\Delta B+B_t\Delta A)$ plus the generated error: This takes additional $\mathcal{O}\left(2\mathcal{F}\log 2\mathcal{F}\right)$ operations.

Hence, for V_M multiplier nodes with $\mathcal{P}$ primary inputs, the propagated PMF can be obtained in $T_M^{(p)}$ operations, where

$$T_M^{(p)}=\mathcal{O}\left((V_M-\mathcal{P})4\mathcal{F}(2^N+\log 2\mathcal{F})\right)\approx\mathcal{O}\left(V_MN_a2^{N+N_a}\right) \quad (71)$$

For the entire DAG, the error PMF can be obtained in $T_A^{(g)}+T_A^{(p)}+T_M^{(g)}+T_M^{(p)}=\mathcal{O}\left((V_A+V_M)N_a2^{N+N_a}\right)$ operations.

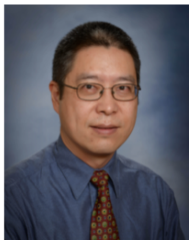
References

[1] Y. K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nguyen, and M. Smelyanskiy, "Convergence of Recognition, Mining, and Synthesis Workloads and Its Implications," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 790–807, 2008.

[2] J. Han and M. Orshansky, “Approximate Computing: An Emerging Paradigm For Energy-Efficient Design,” in *Proceedings of the IEEE European Test Symposium*, 2013, pp. 1–6.
[3] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, “Invited - Cross-layer Approximate Computing: From Logic to Architectures,” in *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, 2016, pp. 99:1–99:6.
[4] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, “Stochastic Computation,” in *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, 2010, pp. 859–864.
[5] C. Liu, J. Han, and F. Lombardi, “A Low-Power, High-Performance Approximate Multiplier with Configurable Partial Error Recovery,” in *Proceedings of the IEEE Design, Automation, and Test in Europe*, 2014, pp. 1–4.
[6] B. Shao and P. Li, “Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 4, pp. 1081–1090, 2015.
[7] D. Sengupta and S. S. Sapatnekar, “FEMTO: Fast Error Analysis in Multipliers Through Topological Traversal,” in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 294–299.
[8] J. Stolfi and L. de Figueiredo, “An Introduction to Affine Arithmetic,” *Trends in Applied and Computational Mathematics*, vol. 4, no. 3, pp. 297–312, 2003.
[9] J. Huang, J. Lach, and G. Robins, “Analytic Error Modeling for Imprecise Arithmetic Circuits,” in *Proceedings of the Workshop on Silicon Errors in Logic - System Effects*, 2011.
[10] J. Huang, J. Lach, and G. Robins, “A Methodology for Energy-Quality Tradeoff Using Imprecise Hardware,” in *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, 2012, pp. 504–509.
[11] 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 *Proceedings of the IEEE International Conference on Computer Design*, 2013, pp. 47–53.
[12] E. Swartzlander, “Truncated Multiplication with Approximate Rounding,” in *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, vol. 2, 1999, pp. 1480–1483.
[13] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, “Modeling and Synthesis of Quality-Energy Optimal Approximate Adders,” in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 728–735.
[14] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, “MACACO: Modeling and Analysis of Circuits for Approximate Computing,” in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 667–673.
[15] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, “Probabilistic Error Modeling for Approximate Adders,” *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 515–530, 2017.
[16] S. Lee, D. Lee, K. Han, E. Shriver, L. K. John, and A. Gerstlauer, “Statistical Quality Modeling of Approximate Hardware,” in *Proceedings of the IEEE International Symposium on Quality Electronic Design*, 2016, pp. 163–168.
[17] 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.
[18] Y. Emre and C. Chakrabarti, “Low Energy Motion Estimation via Selective Approximations,” in *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2011, pp. 176–183.
[19] A. V. Oppenheim and A. S. Willsky, *Signals and Systems*. New Jersey, NJ. Prentice-Hall, 1997.
[20] M. D. Springer, *The Algebra of Random Variables*. New York, NY. Wiley, 1979.
[21] “EXPRESS Benchmarks,” http://www.ece.ucsb.edu/EXPRESS/benchmark/, accessed March 20, 2017.
[22] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, “Joint Precision Optimization and High Level Synthesis for Approximate Computing,” in *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 104.1–104.6.
[23] H. R. Mahdiani, A. Ahmadi, S. M. 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.
[24] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, NY. Oxford University Press, 2000.
[25] J. C. Costa, J. C. Monteiro, and S. Devadas, “Switching Activity Estimation Using Limited Depth Reconvergent Path Analysis,” in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*, 1997, pp. 184–189.
[26] S. Devadas, K. Keutzer, and J. White, “Estimation of Power Dissipation in CMOS Combinational Circuits Using Boolean Function Manipulation,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 3, pp. 373–383, 1992.
[27] G. B. Huang, M. Mattar, H. Lee, and E. Learned-Miller, “Learning to Align from Scratch,” in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, 2012, pp. 764–772.
[28] M. Frigo and S. G. Johnson, “The Design and Implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.
[29] M. R. Driels and Y. S. Shin, “Determining the Number of Iterations for Monte Carlo Simulations of Weapon Effectiveness,” *Technical Report, Naval Postgraduate School, Monterey, CA*, 2004.
[30] S.-H. Cha, “Comprehensive Survey on Distance/Similarity Measures Between Probability Density Functions,” *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 1, no. 4, pp. 300–307, 2007.
[31] R. Beran, “Minimum Hellinger Distance Estimates for Parametric Models,” *The Annals of Statistics*, vol. 5, no. 3, pp. 445–463, 1977.

Deepashree Sengupta received the B. Tech. and M.Tech degrees from the Indian Institute of Technology, Kharagpur, and the PhD degree from the University of Minnesota. She has interned at Taiwan Semiconductor Manufacturing Company Limited and Intel Corporation. Her research interests include circuit reliability issues in nanometer scale regime, and low power design of CMOS circuits. She is a recipient of the Doctoral Dissertation Fellowship from the University of Minnesota, and is currently a Senior R&D Engineer at Synopsys Inc.

Farhana S. Snigdha received the B.Sc. degree from Bangladesh University of Engineering and Technology, Dhaka and is currently pursuing Ph.D. degree in Electrical Engineering at the University of Minnesota. She has interned at Cadence Design Systems. Her research interests include design and optimization of low power system and architecture, and is currently working on real-time approximate low-power image processing as well as various Neural Network architectures.



Jiang Hu (F'16) received the B.S. degree from Zhejiang University (China), the M.S. and the Ph.D. degrees from the University of Minnesota. He has worked with IBM Microelectronics from 2001 to 2002, and has been a faculty at the Texas A&M University, thereafter. His research interests include VLSI circuit optimization, adaptive design, hardware security, and power management for large scale computing systems. He has received two conference Best Paper awards, the IBM Invention Achievement Award, and the Humboldt Research Fellowship. He was an associate editor of IEEE TCAD from 2006 to 2011, and is currently an associate editor for the ACM TODAES.

Sachin S. Sapatnekar (S'86, M'93, F'03) received the B. Tech. degree from the Indian Institute of Technology, Bombay, the M.S. degree from Syracuse University, and the Ph.D. degree from the University of Illinois. He taught at Iowa State University from 1992 to 1997 and has been at the University of Minnesota since 1997, where he holds the Distinguished McKnight University Professorship and the Robert and Marjorie Henle Chair in the Department of Electrical and Computer Engineering. He has received seven conference Best Paper awards, a Best Poster Award, two ICCAD 10-year Retrospective Most Influential Paper Awards, the SRC Technical Excellence award and the SIA University Researcher Award. He is a Fellow of the ACM and the IEEE.