DREAMx: A Data-driven Error Estimation Methodology for Adders Composed of Cascaded Approximate Units

Muhammad Abdullah Hanif*, *Member, IEEE,* Ayoub Arous*, and Muhammad Shafique, *Senior Member, IEEE*

***Abstract*—Due to the significance and broad utilization of adders in computing systems, the design of low-power approximate adders has received a significant amount of attention from the system design community. However, the selection and deployment of appropriate approximate modules require a thorough design space exploration, which is (in general) an extremely time-consuming process. Towards reducing the exploration time, different error estimation techniques have been proposed in the literature for evaluating the quality metrics of approximate adders. However, most of them are based on certain assumptions that limit the usability of such techniques for real-world settings. In this work, we highlight the impact of those assumptions on the quality of error estimates provided by the state-of-the-art techniques and how they limit the use of such techniques for real-world settings. Moreover, we highlight the significance of considering input data characteristics to improve the quality of error estimation. Based on our analysis, we propose a systematic data-driven error estimation methodology, DREAMx, for adders composed of cascaded approximate units, which covers a predominant set of low-power approximate adders. DREAMx in principle factors in the dependence between input bits based on the given input distribution to compute the Probability Mass Function (PMF) of error value at the output of an approximate adder. It achieves improved results compared to the state-of-the-art techniques while offering a substantial decrease in the overall execution(/exploration) time compared to exhaustive simulations. Our results further show that there exists a delicate trade-off between the achievable quality of error estimates and the overall execution time.**

***Index Terms*—Approximate Computing, Approximate Adders, Error Prediction, Error Estimation, Error PMF.**

I. Introduction and Related Work

Approximate Computing (AC) has emerged as a promising computing paradigm for pushing the boundaries of energy and performance efficiency of computing systems to new horizons. Functional approximation of arithmetic modules at the hardware level has surfaced as a highly effective technique for achieving desirable quality-efficiency trade-offs [1] [2]. Adders, being the fundamental operators in computing systems, have received a significant amount of attention from the AC community, where a number of low-power and low-latency approximate adders have been proposed for different application domains and systems [3]. However, to enable

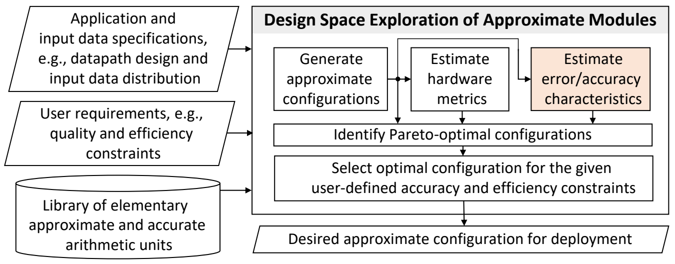

Application and input data specifications, e.g., datapath design and input data distribution
User requirements, e.g., quality and efficiency constraints
Library of elementary approximate and accurate arithmetic units
Design Space Exploration of Approximate Modules
Generate approximate configurations
Estimate hardware metrics
Estimate error/accuracy characteristics
Identify Pareto-optimal configurations
Select optimal configuration for the given user-defined accuracy and efficiency constraints
Desired approximate configuration for deployment

Fig. 1. A generic design space exploration flow, highlighting the dependence of the process on fast-yet-accurate error estimation for identifying suitable approximate configurations for a given application/scenario.

time- and resource-efficient selection of approximate modules for deployment in real-world systems, automated tools are required that can explore the vast design space of approximate modules and output suitable configurations based on the given user requirements/constraints (see Fig. 1). The efficiency, as well as quality, of these tools mainly depends on the error estimation block (highlighted with orange color in Fig. 1). *Thus, efficient-yet-accurate error estimation is crucial for effective design space exploration of approximate modules.*

State-of-the-art and their limitations: Exhaustive/Monte-Carlo simulations is a possible approach for computing accurate error metrics of approximate configurations. However, it is highly time consuming and, thus, cannot be used for extensive design space exploration of approximate modules [6]. To overcome this issue, numerous studies have been carried out towards developing efficient methods for estimating error metrics of approximate adders [4]–[13]. Mazahir et al. [6] proposed a general technique for computing the Probability Mass Function (PMF) of error value of Low-Latency Approximate Adders (LLAAs). The technique first identifies the conditions on input that lead to error and then transforms them to simpler, independent events to compute the PMF. However, to simplify the analysis they consider the input bits/groups of bits to be independent of one another. Along similar lines, Yu et al. [7] proposed a technique based on Binary Decision Diagrams (BDDs) to compute the Error Rate (ER) of Variable Latency Speculative Adder (VLSA) [14] and Accuracy-Configurable Adder (ACA) [15] adders. Other techniques for error estimation of LLAAs include [8] and [9]. As a key focus of approximate computing is towards trading quality for power/energy efficiency and the above-mentioned techniques cannot be used for Low-Power Approximate Adders (LPAAs) such as adders

M. A. Hanif, A. Arous, and M. Shafique are with the eBRAIN Lab, New York University Abu Dhabi (NYUAD), PO Box 129188, Saadiyat Island, Abu Dhabi, United Arab Emirates (emails: {mh6117, aa10539, muhammad.shafique}@nyu.edu).

Manuscript received xx, 2024; revised xx, 2024.

*These authors contributed equally to this work

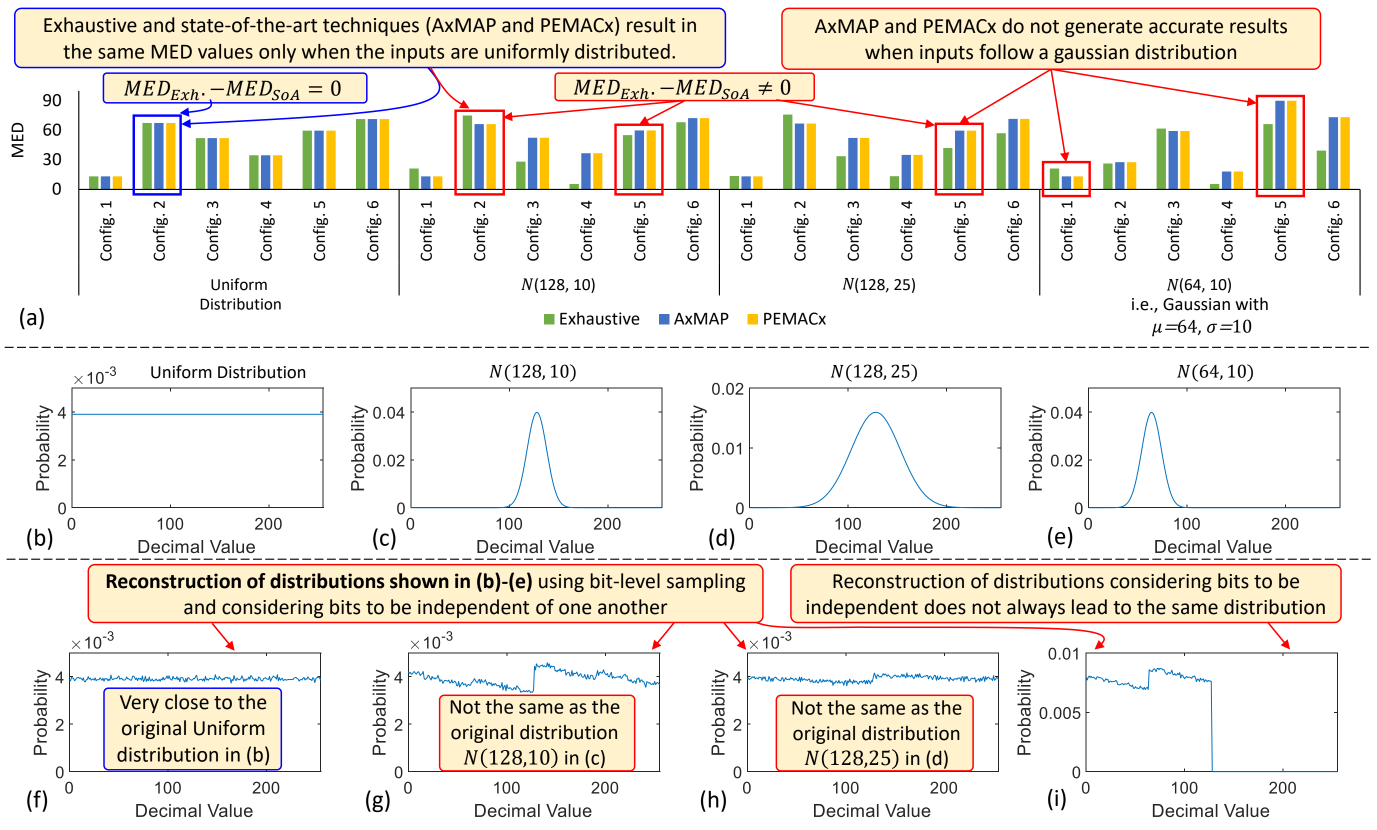


Fig. 2. (a) Error estimates of six different 8-bit LPAA configurations computed using exhaustive simulations, AxMAP [4], and PEMACx [5] considering four different input distributions. The first three configurations are randomly selected and are $Config.\ 1 = [1,\ 7,\ 2,\ 6,\ 7,\ 2,\ 0,\ 0]$, $Config.\ 2 = [6,\ 6,\ 2,\ 1,\ 3,\ 7,\ 1,\ 7]$, and $Config.\ 3 = [7,\ 3,\ 1,\ 4,\ 2,\ 5,\ 4,\ 2]$. The last three configurations are homogeneous configurations and are $Config.\ 4 = [1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]$, $Config.\ 5 = [2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2,\ 2]$, and $Config.\ 6 = [3,\ 3,\ 3,\ 3,\ 3,\ 3,\ 3,\ 3]$. Note that here each element in the configuration vector corresponds to the type of approximate full-adder used at that location. The left most element corresponds to the type of the least significant full-adder and the right most element corresponds to the type of the most significant full-adder in the adder. The types of full-adders (i.e., 0-7) are presented in Table I. The four different input distributions considered for these experiments are presented in (b)-(e), where (b) is uniform distribution, (c) is Gaussian with $\mu = 128$ and $\sigma = 10$, (d) is Gaussian with $\mu = 128$ and $\sigma = 25$, and (e) is Gaussian with $\mu = 64$ and $\sigma = 10$. (f)-(i) represent the reconstructed version of the distributions when the reconstruction is performed using bit-level sampling while assuming all the bits to be independent of one another, i.e., the assumption used in state-of-the-art error estimation techniques. The complete procedure for distribution reconstruction for the above-mentioned scenario is presented in Appendix A.

composed of approximate Full Adders (FAs) proposed in [16] [17], specialized techniques have been proposed for analyzing the error characteristics of LPAAs. Ayub et al. [12] proposed an error occurrence estimation technique for LPAAs. However, it is capable of computing only the probability of error (i.e., ER) at the output of an LPAA configuration and cannot be used to estimate the error magnitude, such as Mean Squared Error (MSE) and Mean Error Distance (MED), as highlighted in [5]. To address this limitation, PEMACx [5] proposed a technique to compute the PMF of error value of LPAAs composed of smaller approximate units. Although the technique can be used for arbitrary probability distribution of input bits, it assumes all the input bits to be independent of one another. Similarly, AxMAP [4] and [10] are also designed for computing the error metrics of LPAAs. However, these techniques also consider the input bits to be independent of one another or are valid only for uniformly distributed inputs, respectively. Other solutions such as [11] and [13] that have been proposed for evaluating the error characteristics of approximate circuits are also based on similar assumptions.

In summary, the state-of-the-art techniques for error estimation of LPAAs either assume the input bits to be independent of one another or focus only on computing the error probability, which does not provide any insight about the error magnitude. Our motivational case study in the following section illustrates that considering input bits to be independent of one another can have a significant impact on the accuracy of the generated estimates, depending on the given scenario, mainly input distribution. In this work, we leverage this motivational analysis for designing an improved error estimation technique that can offer better estimates (close to the results generated through exhaustive simulations) for a wide range of LPAAs and input distributions.

A. Motivational Case Study: Highlighting the Significance of Considering Important Input Data Characteristics for Error Analysis

In this section, we compare the error estimates computed using exhaustive simulations with the estimates computed using state-of-the-art error estimation techniques, AxMAP [4] and PEMACx [5], for different 8-bit LPAA configurations.

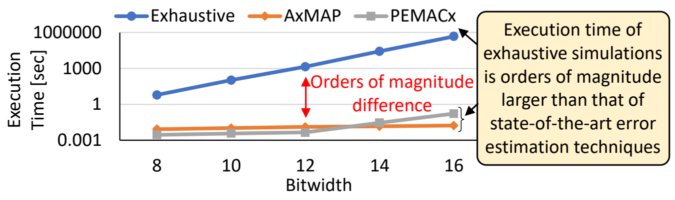


Fig. 3. Execution time comparison between exhaustive simulations and state-of-the-art error estimation techniques AxMAP and PEMACx

To highlight the impact of input data distribution on the accuracy of estimates generated using different techniques, we consider three different input distributions: (1) uniform distribution, (2) Gaussian distribution with $\mu = 128$ and $\sigma = 10$, and (3) Gaussian distribution with $\mu = 128$ and $\sigma = 25$. Fig. 2(a) shows that AxMAP [4] and PEMACx [5] both generate accurate results (same as exhaustive simulations) for uniformly distributed inputs case, but result in significant deviation from the ground truth when inputs follow a Gaussian distribution (i.e., non-uniform distribution). The key reason behind this deviation can be associated with the assumptions considered in both state-of-the-art works, i.e., all the input bits are independent of one another. To validate this, we computed the bit-level probabilities for all the considered distributions, and then for each case, reconstructed the distribution through sampling while considering all the input bits to be independent. As can be seen in Fig. 2, only the reconstruction of the uniform distribution (Fig. 2(f)) is close to its original distribution (Fig. 2(b)). However, in all other cases where the given distribution is not uniform, the reconstructed distribution is significantly different from the original distribution, which can be observed by comparing Fig. 2(g) with Fig. 2(c), Fig. 2(h) with Fig. 2(d), and Fig. 2(i) with Fig. 2(e). *In summary, the analysis clearly shows that considering inputs bits to be independent is not a valid assumption for all types of input distributions.*

B. Scientific Challenges Targeted in this Paper

As highlighted in Section I-A, ignoring dependence between input bits can result in significant deviation from the actual error estimates, and the state-of-the-art techniques that consider input bits to be independent regardless of the input distribution cannot provide accurate error estimation in all cases. Therefore, the key challenge targeted in this research is that considering the execution time of state-of-the-art techniques is orders of magnitude less than that of exhaustive simulations (see Fig. 3), *how can we incorporate the information of dependence between input bits in the error estimation methodologies while maintaining their execution time benefits. In other words, how to design a method that produces error estimates closer to exhaustive simulations while having execution time in the range of the execution time of the state-of-the-art error estimation techniques.*

C. Our Novel Contributions

To address the above research challenge, in this paper, we make the following key contributions.

1) We analyze the impact of the assumptions used in the state-of-the-art error estimation techniques on the quality of their results for LPAAs.
2) We highlight the significance of considering input data characteristics, specifically dependence between input bits, for improving the quality of error estimation techniques.
3) We propose a systematic data-driven error estimation methodology, DREAMx, for adders composed of cascaded approximate units, which covers a predominant set of low-power approximate adders. Our methodology factors in the dependence between input bits to get a more representative view of the input characteristics and generate improved results compared to state-of-the-art techniques while offering a significant reduction in execution time compared to exhaustive simulations.
4) We present extensive results to show the effectiveness of DREAMx over state-of-the-art techniques designed for LPAAs. We demonstrate the efficacy of the proposed technique for efficient and accurate design space exploration. Our results show that there exists a delicate trade-off between the achievable quality of error estimates and the overall execution time.

II. Background and Preliminaries

Fig. 4 shows a generic structure of an N-bit LPAA composed of N full adders. The adder takes in two N-bit operands A and B and a carry-in (C_{in}) as inputs to generate an N-bit sum (S) and a carry-out (C_{out}). Each full adder can be an accurate or an approximate full adder. The truth tables of some of the existing approximate full-adder designs are presented in Table I. Ax. Type 1 to Ax. Type 5 represent the five low-power approximate full-adders proposed in [16] while Ax. Type 6 and Ax. Type 7 correspond to the variants proposed in [17]. The generic structure shown in Fig. 4 is also valid for other low-power approximate adder designs such as Lower-Part-OR Adder (LOA) [18], Equally Segmented Adder (ESA) [3], and Quality-Area optimal low-latency Adder (QuAd) [19].

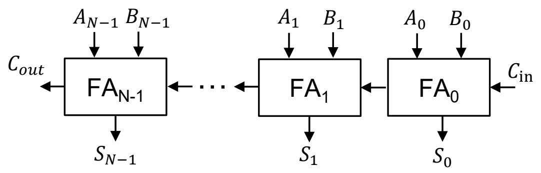


Fig. 4. A generic structure of a multi-bit adder composed of cascaded full-adder units. Each full adder can be an accurate or an approximate full adder.

III. Methodology

A. Concept

As highlighted in Fig. 2, the bits of real-world input data can be significantly dependent on one another. Ignoring this key characteristic can result in inequivalence between the actual and the considered probability distribution of input data, as highlighted in Fig. 2 for three different Gaussian distributions. In this section, we argue that considering dependence between

TABLE I
TRUTH TABLES OF STATE-OF-THE-ART APPROXIMATE FULL ADDER DESIGNS COVERED IN [16] AND [17].THE ERROR CASES ARE HIGHLIGHTED IN RED.

Inputs			Accu. FA		Ax. Type 1		Ax. Type 2		Ax. Type 3		Ax. Type 4		Ax. Type 5		Ax. Type 6		Ax. Type 7	
A	B	C_{in}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1	0	1	0	1	0	0	0	1	1	1	0
0	1	0	1	0	0	1	1	0	0	1	0	0	1	0	1	0	1	0
0	1	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1	1	1
1	0	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	1	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1
1	1	0	0	1	0	1	0	1	0	1	0	1	1	1	0	0	0	1
1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1

only the highly dependent bits may be sufficient to generate error estimates closer to the values generated using exhaustive simulations (i.e., the ground truth). To study this, we consider the same three Gaussian distributions illustrated in Fig. 2(c-e) and reconstruct the distribution through bit-level sampling while considering all the input bits to be independent, considering dependence between two most dependent bits, considering dependence between three most dependent bits, and considering dependence between four most dependent bits. To reconstruct a distribution considering input bits to be independent, we follow the following procedure:

- **Step 1:** Using the user-provided input distribution, compute the probability distribution of each bit. For example, if an operand A is composed of 8 bits, i.e., $A = a0, ..., a6, a7$, where A can take any integer value between 0 and 255, and the PMF of A is given as $p_A(a)$, the probability of each bit location a_i being 1 (i.e., $p_{a_i}(a_i = 1)$) can be computed by summing the probabilities of all cases that lead to $a_i = 1$ (for all i in 0,...,7).
- **Step 2:** Using sampling, the bit-level distributions (p_{a_i}s) computed in **Step 1**, and considering the input bits to be independent of one another, we build a sample set of 1 million samples. For computing each sample, we sample one bit from each bit distribution and then combine them using $A = \sum_{i=0}^{7} a_i * 2^i$, where a_i is the bit sampled from the distribution of the i^{th} bit location.
- **Step 3:** Build a distribution (or histogram) using the sample set from Step 2.

To reconstruct a distribution considering some of the input bits to be dependent on one another, in Step 2 of the above-mentioned procedure, we take into account the conditional probabilities of the dependent bits during sampling while keeping rest of the procedure the same.

Fig. 5 presents the reconstructed distributions. By comparing Fig. 5(c), Fig. 5(d) and Fig. 5(e) with Fig. 5(a), it can be observed that as more number of highly dependent bits are considered dependent on one another, the reconstructed distribution starts becoming more correlated with the original distribution. The same trend can be observed for the other two distributions presented in Fig. 5(f) and Fig. 5(k). *Therefore, considering the highly dependent bits as dependent during error analysis can lead to better error estimates for approximate modules.*

B. Methodology Overview

An overview of the proposed methodology is presented in Fig. 6. The methodology takes in probability distribution of input operands, bit width of inputs, elementary approximate adders (which can be single-bit as well as multi-bit modules), and user-defined constraints as inputs. Using the probability distribution of input operands, it computes conditional probability matrices ($CPMs$), which contain conditional probability information for all the bits, and for all possible cases (see ⓐ in Fig. 6). These matrices along with the probability of bits (see ⓑ in Fig. 6) are then used to identify dependence between bits using user-defined constraints (see ⓒ in Fig. 6). Based on the dependence information, a Link Vector (LV) is generated, which specifies the bit locations that should be considered together (see ⓓ in Fig. 6). The methodology then uses the unique set of number of bit locations that should be considered together to identify the different sizes of required modules. In the next step, the methodology generates all possible configurations for each module size and builds an extended library of elementary approximate modules (see ❷ in Fig. 6). The library and the link vector LV are then used to compute the PMF of error of all the possible (or required) configurations considering partial independence between modules (see ❸ in Fig. 6). The details of all steps are presented in the following subsections.

C. Dependence Identification

This section presents our method for identifying dependence between input bits and computing the Link/connectivity Vector (LV). The LV vector defines the bit locations that are dependent on each other. The steps are explained in detail below.

1) Conditional Probability Estimation: : Algo. 1 presents our method for computing Conditional Probability Matrices ($CPMs$). We mainly compute four CPMs, i.e., CPM{0,0}, CPM{0,1}, CPM{1,0}, and CPM{1,1}. CPM{0,0} is for storing $P(a_i = 0|a_j = 0) \forall i \in 2, 3, ..., N$, where $j \in i - CL, ..., i - 1$. CL represents *coverage length*, which is used to limit the number of bit locations that should be considered for dependency checking. For example, $CL = 2$ means that for each bit location, dependence should be checked with only the neighboring lower significance two bits. An illustration of CPM is presented in ⓐ in Fig. 6.

2) Dependence Identification based on User-defined Constraint: : For identifying dependence between two bits a_i and a_j, we compare the conditional probability $P(a_i|a_j)$ with the probability $P(a_i)$ using the following equation.

$$abs(P(a_i|a_j) - P(a_i)) \geq \alpha \qquad (1)$$

Here, α is a strictness parameter, as we know from the probability theory that $P(X|Y) = P(X)$ when X and Y are independent of each other. The range of α is between 0 and 1, where $\alpha = 0$ corresponds to the most strict scenario and $\alpha = 1$ corresponds to least strict scenario. In other words, α defines the level of softness in deciding whether two bit locations should be considered dependent or independent. We use all four conditional matrices for identifying dependence

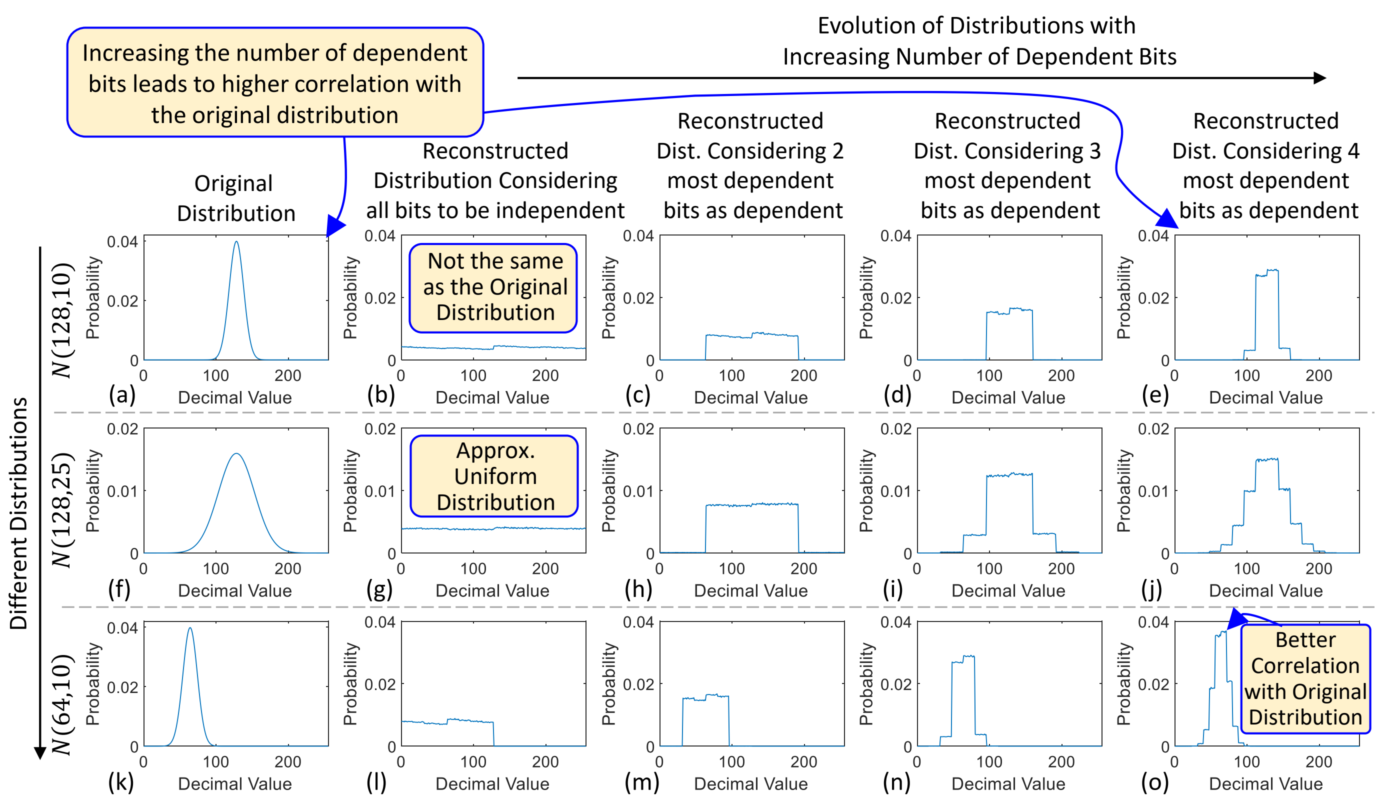
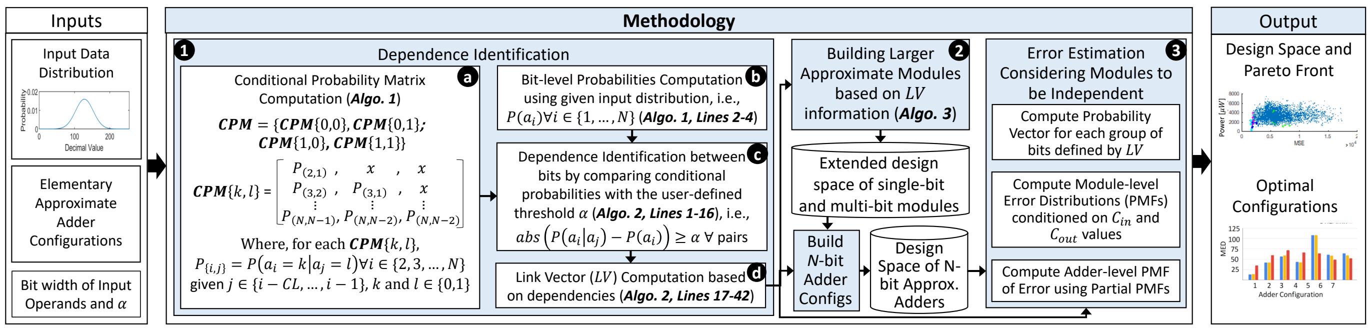


Fig. 5. Impact of increasing the number of dependent input bits during distribution reconstruction process on the correlation of the reconstructed distribution with the original input distribution. Each row presents results for a specific distribution. The first column presents the original distributions. The second column presents the reconstructed distributions when all the bits are considered independent during the reconstruction process. The third column presents the reconstructed distributions when only the two most dependent bits are considered dependent during the reconstruction process. The fourth and fifth columns show the reconstructed distributions considering three and four most dependent bits as dependent (respectively) during the reconstruction process.



Inputs
Input Data Distribution
Elementary Approximate Adder Configurations
Bit width of Input Operands and α
Methodology
1 Dependence Identification
Conditional Probability Matrix Computation (Algo. 1) a
$CPM = \{CPM\{0,0\}, CPM\{0,1\}; CPM\{1,0\}, CPM\{1,1\}\}$
$CPM\{k,l\} = \begin{bmatrix} P_{(2,1)}, & x, & x \\ P_{(3,2)}, & P_{(3,1)}, & x \\ \vdots & \vdots & \vdots \\ P_{(N,N-1)}, & P_{(N,N-2)}, & P_{(N,N-2)} \end{bmatrix}$
Where, for each $CPM\{k,l\}$,
$P_{\{i,j\}} = P(a_i = k | a_j = l) \forall i \in \{2,3,\ldots,N\}$ given $j \in \{i-CL,\ldots,i-1\}$, k and $l \in \{0,1\}$
Bit-level Probabilities Computation using given input distribution, i.e., $P(a_i) \forall i \in \{1,\ldots,N\}$ (Algo. 1, Lines 2-4) b
Dependence Identification between bits by comparing conditional probabilities with the user-defined threshold α (Algo. 2, Lines 1-16), i.e., $abs\left(P(a_i|a_j) - P(a_i)\right) \geq \alpha \; \forall$ pairs c
Link Vector (LV) Computation based on dependencies (Algo. 2, Lines 17-42) d
Building Larger Approximate Modules based on LV information (Algo. 3) 2
Extended design space of single-bit and multi-bit modules
Build N-bit Adder Configs
Design Space of N-bit Approx. Adders
Error Estimation Considering Modules to be Independent 3
Compute Probability Vector for each group of bits defined by LV
Compute Module-level Error Distributions (PMFs) conditioned on C_{in} and C_{out} values
Compute Adder-level PMF of Error using Partial PMFs
Output
Design Space and Pareto Front
Optimal Configurations

Fig. 6. Overview of our DREAMx methodology. In the first step, the methodology identifies the dependence between input bits by analyzing their conditional probabilities. Once the dependence is identified, the information is used to generate larger approximate blocks, and then, existing methodologies are adapted to offer efficient-yet-accurate error estimates.

between bit locations. Our complete procedure for dependence identification is presented in Algo. 2 Lines 1-16.

3) Connectivity Checking and Link Vector Generation: : Once dependence between different bit locations has been identified, the methodology finds the sets of locations that should be considered together during analysis. The locations within each set defines the locations that are dependent on one another. The methodology achieves this by comparing each pair of dependent locations identified by Section III-C2. The complete procedure for computing the LV is presented in Algo. 2 from Line 17 to Line 42.

D. Building Larger Approximate Modules

LV defines the bit locations that should be considered together, and thus, it also defines the sizes of the modules required. For constructing larger approximate modules by merging multiple smaller approximate modules, we propose Alog. 3. It takes in the truth tables of two approximate adder blocks (i.e., $T1$ and $T2$), to construct the truth table (T) of the merged module, considering $T1$ to be at the lower significance location. The format of the truth table matrices is defined according to the format shown in Table I. For example, the matrix $T1$ in the case of an approximate full adder would be

Algorithm 1 Pseudo-code for conditional probability estimation

```
Input:
$Dist$: Input distribution array defining probabilities for all input values
$BW$: Bitwidth of inputs
$CL$: Coverage length to define the set of bits for each bit location that should be considered for dependence checking
Initialize:
$Conditional\_Matrix$=[]; %Matrix for storing conditional probabilities of all the bits with others (defined based on $CL$)
$Bit\_Probabilities$=[]; %Array for storing probability of each bit being 1
 1: $Dec\_Values$ = [0:$2^{BW}-1$]
 2: for $i = \{1, 2, ..., BW\}$ do
 3:     $Bit\_Probabilities$(i) = sum($Dist$(mod($Dec\_Values$, $2^i$) $\geq$ $2^{i-1}$));
 4: end for
 5: for $i = \{BW, BW-1, ..., 2\}$ do
 6:     for $j = \{i-1, i-2, ..., max(1, i-CL)\}$ do
 7:         for $bit\_i\_val = \{i-1, i-2, ..., max(1, i-CL)\}$ do
 8:             for $bit\_j\_val = \{i-1, i-2, ..., max(1, i-CL)\}$ do
 9:                 inds_for_i = xor(mod($Dec\_Values$,$2^i$) $\geq$ $2^{i-1}$, $bit\_i\_val$)
10:                 inds_for_j = xor(mod($Dec\_Values$,$2^j$ $\geq$ $2^{j-1}$, $bit\_j\_val$)
11:                 $Conditional\_Matrix\{bit\_i+1, bit_j+1\}(i-1, i-j)$ = sum($Dist$(inds_for_i & inds_for_j))/sum($Dist$(inds_for_j))
12:             end for
13:         end for
14:     end for
15: end for
16: Return $Conditional\_Matrix$, $Bit\_Probabilities$
```

Algorithm 2 Pseudo-code for dependence & LV computation

```
Input:
$Conditional\_Matrix$, $Bit\_Probabilities$
$\alpha$: Threshold value for defining two bits dependent/independent
Initialize:
$LV$=[]; %Link vector that defines which bits should be considered together
 1: $Compare\_Mat$ = repmat($Bit\_Probabilities$(2:end)', 1, size($Conditional\_Matrix\{1,1\}$,2));
 2: for $i = \{1, ...,$ size($Conditional\_Matrix$,1)$\}$ do
 3:     for $j = \{1, ...,$ size($Conditional\_Matrix$,2)$\}$ do
 4:         $Dependence\_Matrix\{i,j\}$ = abs($Conditional\_Matrix\{i,j\}$ - (not(i-1) - $Compare\_Mat \times -1^{i-1}$)) $\geq \alpha$;
 5:         for $k = \{1, ...,$ size($Dependence\_Matrix\{i,j\}$,1)$\}$ do
 6:             for $l = \{k+1, ...,$ size($Dependence\_Matrix\{i,j\}$,2)$\}$ do
 7:                 $Dependence\_Matrix\{i,j\}$(k,l) = 0 %Invalid entries = 0
 8:             end for
 9:         end for
10:         if $i == 1 and j == 1$ then
11:             Dependence = $Dependence\_Matrix\{i,j\}$
12:         else
13:             Dependence = Dependence — $Dependence\_Matrix\{i,j\}$
14:         end if
15:     end for
16: end for
17: ranges_max = (sum(Dependence,2)!=0).*[2:size(Dependence,1)+1]'
18: ranges_min = min(Dependence.*repmat([-1:-1:(-1)*size(Dependence,2)],size(Dependence,1),1),[],2)+ranges_max
19: for i = 1:length(ranges_max) do
20:     merge_check = 0;
21:     for j = i+1:length(ranges_max) do
22:         if ranges_max(i) $\geq$ ranges_min(j) then
23:             ranges_min(j) = min([ranges_min(i),ranges_min(j)]);
24:             merge_check = 1;
25:         end if
26:     end for
27:     if merge_check == 1 then
28:         ranges_min(i) = 0;
29:         ranges_max(i) = 0;
30:     end if
31: end for
32: ranges_min = ranges_min(ranges_min != 0)
33: ranges_max = ranges_max(ranges_max != 0)
34: $LV$ = ones(1,ranges_min(1)-1)
35: for i = 1:length(ranges_min) do
36:     $LV$ = [$LV$, ranges_max(i)-ranges_min(i)+1];
37:     if i != length(ranges_min) then
38:         if ranges_min(i+1)>ranges_max(i)+1 then
39:             $LV$ = [$LV$, ones(1,ranges_min(i+1)-range_max(i)-1)]
40:         end if
41:     end if
42: end for
43: Return $LV$
```

composed of two vectors, i.e., $[Sum, C_{out}]$, considering each vector to be a column vector defined in the same sequence as shown in Table I. The format of the output is also the same; for example, when merging two approximate full adders, the output would be in the format $[Sum_0, Sum_1, C_{out}]$. The algorithm is used to construct an extended design space of single-bit and multi-bit approximate modules.

E. PMF of Error Computation

LV defines the sections of an LPAA that can be considered independent. Therefore, we use this information to leverage the concepts proposed in state-of-the-art error estimation techniques. We propose to consider the bit locations that are highly dependent on each other as a part of a single multi-bit block and use larger approximate modules to consider the locations together. This enables us to compute the probability of each input combination of the larger block and, thereby, take into account the dependence between bit locations by computing the probability vector for each group of bits defined by LV. For implementation, we compute module-level error distributions / PMFs of error conditioned on C_{in} and C_{out} values to overcome the bias introduced due to probabilities of C_{in} and C_{out} signals. Then, we compute the adder-level PMF of error by using the Partial PMFs, similar to the method proposed in PEMACx [5].

IV. Results

We implemented DREAMx using MATLAB 2023b and performed experiments on an Intel Core i5 system with 16 GB RAM. Our primary investigation involved a comparison between DREAMx and exhaustive simulations. To introduce dependencies among the input bits, we conducted our experiments using various Gaussian distributions. Furthermore, we conducted a comparative analysis between DREAMx and a state-of-the-art technique, PEMACx [5]. The comparison is based on the assessment of relative errors in a dependency-driven scenario using exhaustive simulations method as reference. The relative error for MSE is calculated using the following equation:

$$RelativeError = \frac{|MSE_{PEMACx} - MSE_{Exhaustive}|}{MSE_{Exhaustive}} * 100 \quad (2)$$

Algorithm 3 Pseudo-code for merging truth tables of appx. blocks

```
Input:
    $T1$: Truth table of the first approximate block
    $T2$: Truth table of the second approximate block
Initialize:
    [$T1\_rows$, $T1\_cols$] = size($T1$)
    [$T2\_rows$, $T2\_cols$] = size($T2$)
    $T$ = zeros($T1\_rows \times T2\_rows$ / 2, $T1\_cols + T2\_cols$ - 1)
1: T(:, 1:$T1\_cols$-1) = repmat($T1$(:, 1:end-1), $T2\_rows$ / 2, 1)
2: indexes = repmat($T1$(:,end), $T2\_rows$ / 2, 1) +
                    repelem([0: $T2\_rows$ / 2 - 1]', $T1\_rows$) $\times$ 2
3: T(:, $T1\_cols$:end) = $T2$(indexes+1, :)
4: Return $T$
```

A. Results on Synthetic Data

To demonstrate the applicability of our approach, we compared our results to those generated using open-source code of PEMACx [5] and exhaustive simulations for different LPAA configurations. The results are generated for both 8-bit and 12-bit LPAA configurations (see Fig. 7). For this comparison, we considered normal distribution with $\mu = 128$ and $\sigma = 25$, i.e., $N(128, 25)$, as the input distribution for the 8-bit configurations and normal distribution with $\mu = 512$ and $\sigma = 25$, i.e., $N(512, 25)$, for the 12-bit configurations. We evaluated the performance of both PEMACx and our own method within this context of high dependence between input bits. As can be seen in Fig. 7, our proposed method exhibit a significant accuracy advantage over PEMACx. Furthermore, DREAMx significantly reduces the execution time compared to exhaustive simulations (see Fig. 8b). For instance, the proposed DREAMx methodology requires only 0.008 sec to generate the output with 98.06% accuracy for the 8-bit adder configuration $\{1, 2, 3, 6, 5, 7, 4, 5\}$ on the normal input distribution $N(128, 25)$ while the exhaustive simulations take 3.75 sec. Thus, DREAMx leads to a speedup of 468x compared to exhaustive for 8-bit. Note that, for this comparison, we considered different number of merged bits for DREAMx. Towards this, Fig. 8b also highlights that as the number of merged bits increases, the execution time progressively approaches that of the exhaustive simulations. This is primarily due to the increased time required to process larger truth table and error vectors of merged units placed at merged bit locations. Therefore, as the size of the merged unit increases, the sizes of the vectors and the overall execution time also increase, which results in reduction in the overall speedup achieved.

B. Impact of hyperparameter: alpha, sigma and no. of merged bits on accuracy vs exec. time

We attempted to comprehend the effects of merged bits on execution time and relative error in order to better analyze our methodology. Figure 8 (b) shows clearly that the execution time increased with the number of merged bits while the relative error decrease (see figure 8 (a)).

We extended our experiments using synthetic data, systematically altering alpha while keeping sigma constant at 25 as shown in Figure 10 (1), and likewise, adjusting sigma while holding alpha constant (see Figure 10 (2)). These variations

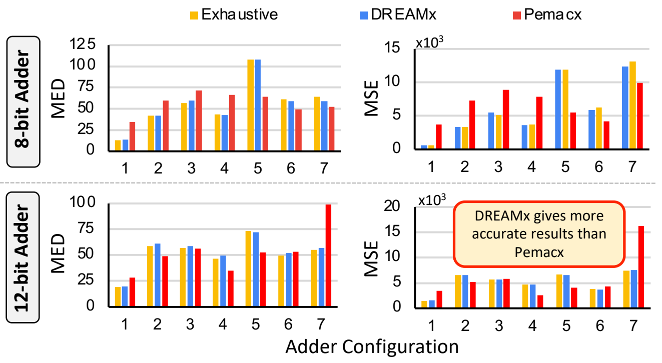


Fig. 7. Error characteristics of different 8-bit and 12-bit LPAAs. Left side and right side illustrate the MED and MSE, respectively. The top row is for 8-bit LPAA configurations and the bottom row is for 12-bit configurations with eight LS FAs approximated.

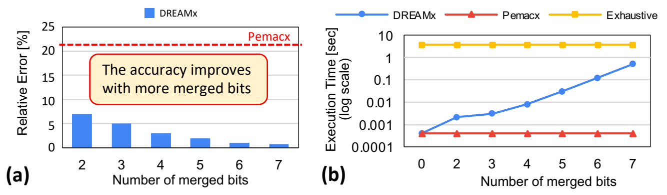


Fig. 8. Impact of number of dependent bits used in DREAMx on the accuracy and execution time for an 8 bits adder with a random configuration=[1 3 2 4 1 5 6 7] and with a normal input distribution $N(\mu = 128, \sigma = 25)$ and threashold $\alpha = 0.1$

were designed to provide deeper insights into how both sigma and threshold alpha influence the accuracy and execution time of DREAMx.

Notably, when alpha is increased, the number of merged bits decreases, which reduces DREAMx's accuracy while simultaneously reducing execution time. A similar trend is observed for sigma, where larger sigma values result in fewer merged bits, causing an increase in relative error but a decrease in execution time.

C. Design space exploration for 8-bit adder

We conducted a comprehensive Design Space Exploration (DSE) for an 8-bit LPAA considering Type 1-3 approximate full adder designs from Table I. We used the power values of the configurations and the power estimation model from PEMACx [5]. In this exploration, we used exhaustive analysis as a baseline to obtain actual error values for all configurations. Additionally, we employed DREAMx and PEMACx for evaluation and comparison, respectively. The actual Pareto frontiers are established through exhaustive simulation results, serving as the reference frontiers for evaluating the performance of DREAMx and PEMACx. In our methodology, we introduced multiple different scenario by varying numbers of merged (dependent) bits.

To illustrate the effectiveness of our methodology, DREAMx, with its dynamic bit merging feature, enabled us to explore a total of 6561 configurations, producing reasonable results in significantly less time compared to exhaustive analysis. For instance, as shown in Fig. 9(a), DREAMx yielded

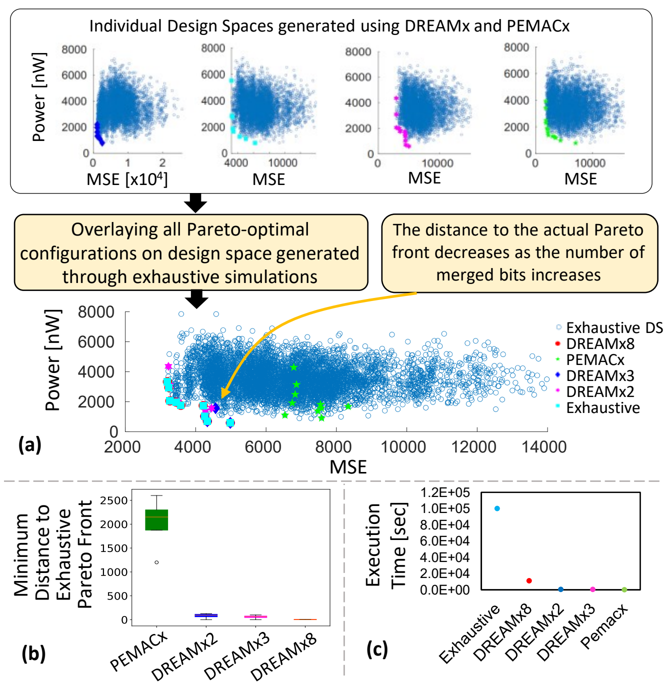


Fig. 9. Design space exploration results for different approaches: Exhaustive, PEMACx, DREAMx8 (case where 8 bits are merged), DREAMx3 (case where 3 bits are merged), DREAMx2 (case where 2 bits are merged) for synthetic normal distributions $N(128, 30)$.

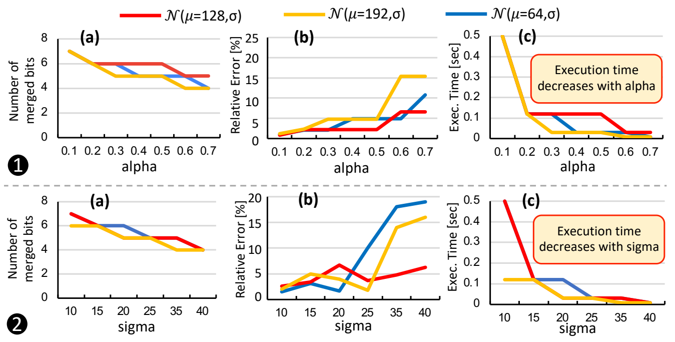


Fig. 10. Impact of varying alpha and sigma in DREAMx for different input distribution on execution time and relative error. Configurations [1 2 2 6 7 4 5 1] and [1 2 2 4 3 4 6 4] are used respectively for experiments (1) and (2).

a Pareto frontier that closely approximated the actual Pareto frontier in just 1224.24 seconds, while the exhaustive method required 99250.86 seconds, resulting in a 81x speedup.

Fig. 9(b) demonstrates the trade-off between accuracy and execution time as influenced by the number of merged bits. Notably, the minimal distance from the exhaustive Pareto frontier decreases as more bits are merged, enhancing accuracy but at the cost of increased execution time. Furthermore, the substantial disparity in the minimal distance from the exhaustive Pareto frontier for PEMACx underscores its inaccuracy in scenarios with high data interdependencies.

D. Effectiveness of DREAMx for Image Processing

To demonstrate the real-world applicability of our proposed methodology, we use image blending through addition as a case study, similar to [6]. An overview of the image blending pipeline considered for this case study is presented in Fig. 11.

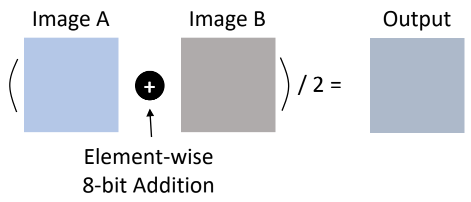


Fig. 11. An illustration of the image blending pipeline used to demonstrate the effectiveness of the proposed approach.

The images considered for this study are presented in Fig. 14, where each image is presented with its corresponding distribution.

We have implemented image blending application using several adder configurations. The configurations used are listed in the caption of Fig. 12. For each configuration, we generated MSE and MED values using simulations, PEMACx, and DREAMx. Fig. 12 presents different MED and MSE values for various approaches utilizing different adder configurations for both sample sets 1 and 2. We can observe from Fig. 12 that DREAMx produces MSE and MED values that are more closer to the exhaustive simulation results than PEMACx. Furthermore, Fig. 13 shows the results of image blending for the first five approximate adder configurations used in Fig. 12(a) on sample set 1, demonstrating the efficiency of DREAMx in estimating error values that are relatively close to exhaustive. Therefore, we conclude that DREAMx can be utilized for error estimation of LPAAs composed of cascaded approximate units for practical applications.

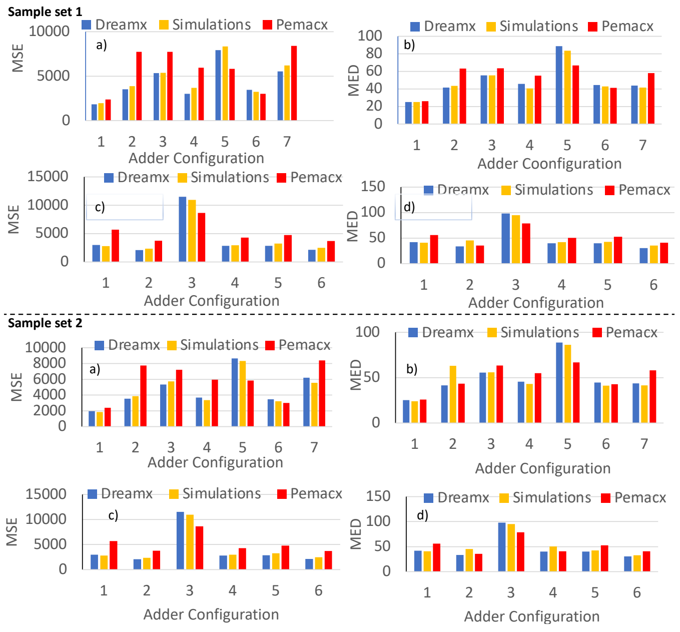
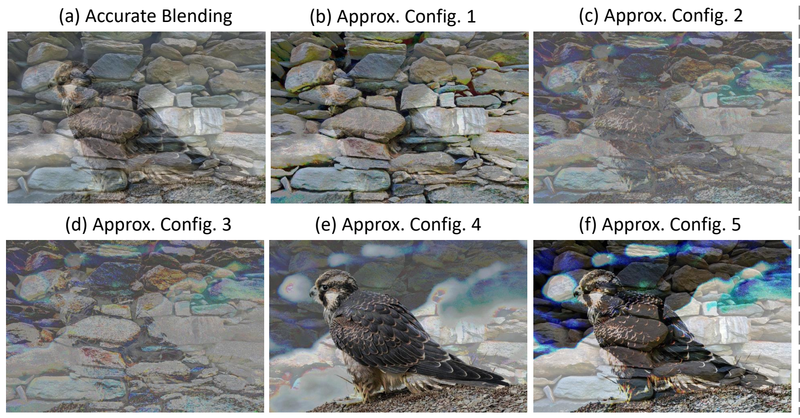


Fig. 12. Comparison of MSE and MED values computed using different approaches for image blending application using Sample set 1 and Sample set 2 presented in Fig. ??. In (a) and (b), for both the sample sets, adder configuration i corresponds to the 8-bit LPAA configuration [i i i i i i i i]. In (c) and (d), for both the sample sets, we considered randomly generated configurations which are: Config. 1 = [1 2 1 5 4 2 6 7], Config. 2 = [1 3 1 4 4 2 4 2], Config. 3 = [3 2 3 4 4 2 4 5], Config. 4 = [2 2 1 1 6 2 43], Config. 5 = [3 3 5 6 6 3 4 3] and Config. 6 = [5 4 5 4 7 1 1 3]



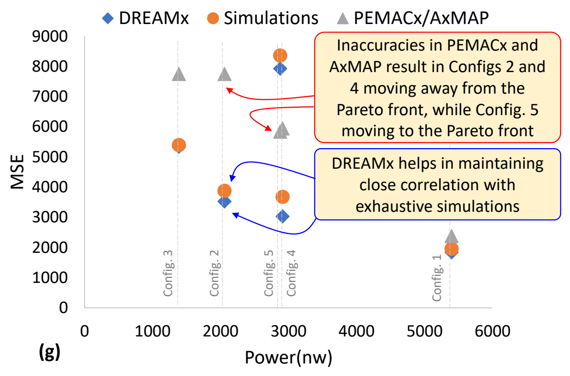


Fig. 13. (a)-(f) Image blending results for various approximate adder configurations on sample set 1. (g) Comparison of error estimates generated using DREAMx and PEMACx with those from exhaustive simulations. The figure clearly demonstrates that DREAMx produces error estimates with a strong correlation to the results of exhaustive simulations. Here, Config. i corresponds to corresponds to 8-bit LPAA with all full-adders of type i, i.e., Config. i = $[i, i, i, i, i, i, i, i]$. The power numbers for the full adders are taken from [5].

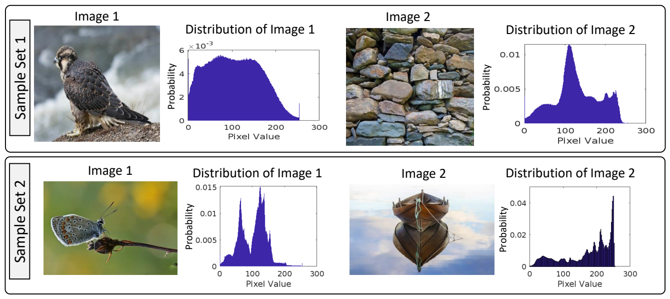


Fig. 14. Images used for the image blending experiments are shown, with the distribution of each image presented directly beside it.

V. Conclusion

In this work, we analyzed the impact of some of the commonly used assumptions in the state-of-the-art error estimation techniques on the quality of their results for LPAAs. Based on our analysis, we proposed a systematic data-driven error estimation methodology, DREAMx, for adders composed of cascaded approximate units, which covers a predominant set of low-power adders. DREAMx takes into account the dependence between input bits to compute the Probability Mass Function (PMF) of error value at the output of an LPAA. The results showed that considering dependence between input bits can significantly improve the quality of results while maintaining the fast execution time benefits of state of the art.

Acknowledgment

This research is supported by ASPIRE, the technology program management pillar of Abu Dhabi's Advanced Technology Research Council (ATRC), via the ASPIRE AARE Award on "Towards Extreme Energy Efficiency through Cross-Layer Approximate Computing".

Appendix A: Procedure to Reconstruct a Distribution

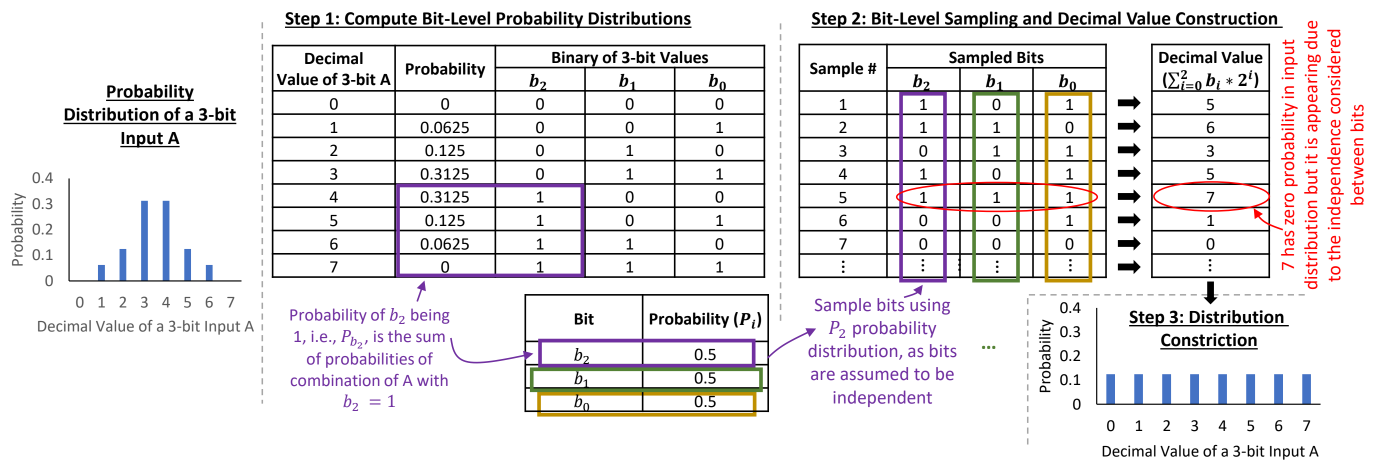
To reconstruct a distribution from bit-level probabilities through bit-level sampling while considering input bits to be independent, we follow the following steps:

- **Step 1:** Using the user-provided input distribution, we first compute the probability distribution of each bit. For example, if an operand A is composed of 8 bits, i.e., $A = b_7, b_6, ..., b_0$, where A can take any integer value between 0 and 255, and the PMF of A is given as $P_A(a)$, the probability of each bit location b_i being 1 (i.e., P_{b_i}) can be computed by summing the probabilities of all combinations that lead to $b_i = 1$. This step is performed for all the bit locations separately (0, 1, ..., 7) to generate $P_{b_0}, P_{b_1}, ..., P_{b_7}$.
- **Step 2**: Using sampling, the bit-level distributions $(P_{b_0}, P_{b_1}, ..., P_{b_7})$ computed in Step 1, and considering the input bits to be independent of one another, we build a sample set of 1 million samples. For each sample, we draw one bit from each bit distribution and combine them using A = $\sum_{i=0}^{7} b_i \times 2^i$, where b_i is the bit sampled from the distribution of the i^{th} bit location.
- **Step 3:** Construct distribution (normalized histogram) using the sample set from Step-2.

An example of the complete process for a 3-bit input is presented in Figure 15.

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.
[2] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Cross-layer approximate computing: From logic to architectures," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 99.
[3] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015, pp. 343–348.
[4] M. Rezaalipour, M. Rezaalipour, M. Dehyadegari, and M. N. Bojnordi, "AxMAP: Making approximate adders aware of input patterns," *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 868–882, 2020.
[5] M. A. Hanif, R. Hafiz, O. Hasan, and M. Shafique, "PEMACx: A probabilistic error analysis methodology for adders with cascaded approximate units," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
[6] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "Probabilistic error modeling for approximate adders," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 515–530, 2016.


Probability Distribution of a 3-bit Input A
Probability
0.4
0.3
0.2
0.1
0
0 1 2 3 4 5 6 7
Decimal Value of a 3-bit Input A
Step 1: Compute Bit-Level Probability Distributions
Probability of b_2 being 1, i.e., P_{b_2}, is the sum of probabilities of combination of A with $b_2 = 1$
Step 2: Bit-Level Sampling and Decimal Value Construction
Sample bits using P_2 probability distribution, as bits are assumed to be independent
...
7 has zero probability in input distribution but it is appearing due to the independence considered between bits
Step 3: Distribution Constriction
Probability
0.4
0.3
0.2
0.1
0
0 1 2 3 4 5 6 7
Decimal Value of a 3-bit Input A

Decimal Value of 3-bit A	Probability	Binary of 3-bit Values		
		b_2	b_1	b_0
0	0	0	0	0
1	0.0625	0	0	1
2	0.125	0	1	0
3	0.3125	0	1	1
4	0.3125	1	0	0
5	0.125	1	0	1
6	0.0625	1	1	0
7	0	1	1	1

Bit	Probability (P_i)
b_2	0.5
b_1	0.5
b_0	0.5

Sample #	Sampled Bits			Decimal Value ($\sum_{i=0}^{2} b_i * 2^i$)
	b_2	b_1	b_0	
1	1	0	1	5
2	1	1	0	6
3	0	1	1	3
4	1	0	1	5
5	1	1	1	7
6	0	0	1	1
7	0	0	0	0
⋮	⋮	⋮	⋮	⋮

Fig. 15. An illustrative example of how to reconstruct a distribution from bit-level probabilities through bit-level sampling while considering input bits to be independent for a 3-bit input.

[7] C. Yu and M. Ciesielski, "Analyzing imprecise adders using BDDs–a case study," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 152–157.

[8] Y. Wu, Y. Li, X. Ge, Y. Gao, and W. Qian, "An efficient method for calculating the error statistics of block-based approximate adders," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 21–38, 2018.

[9] L. Li and H. Zhou, "On error modeling and analysis of approximate adders," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2014, pp. 511–518.

[10] E. Farahmand, A. Mahani, M. A. Hanif, and M. Shafique, "Design and analysis of high performance heterogeneous block-based approximate adders," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 6, pp. 1–32, 2023.

[11] D. Sengupta, F. S. Snigdha, J. Hu, and S. S. Sapatnekar, "An analytical approach for error pmf characterization in approximate circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 1, pp. 70–83, 2018.

[12] M. K. Ayub, O. Hasan, and M. Shafique, "Statistical error analysis for low power approximate adders," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[13] Y.-Q. Dou and C.-H. Wang, "An optimization technique for pmf estimation in approximate circuits," *Journal of Computer Science and Technology*, vol. 38, no. 2, pp. 289–297, 2023.

[14] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 1250–1255.

[15] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proceedings of the 49th annual design automation conference*, 2012, pp. 820–825.

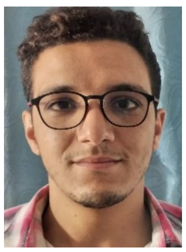
[16] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 32, no. 1, pp. 124–137, 2012.

[17] H. A. Almurib, T. N. Kumar, and F. Lombardi, "Inexact designs for approximate low power addition by cell replacement," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 660–665.

[18] 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.

[19] M. A. Hanif, R. Hafiz, O. Hasan, and M. Shafique, "Quad: Design and analysis of quality-area optimal low-latency approximate adders," in *Proceedings of the 54th Annual Design Automation Conference*, June 2017, pp. 1–6.

Muhammad Abdullah Hanif (Member, IEEE) received the B.Sc. degree in electronic engineering from the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology (GIKI), Pakistan, the M.Sc. degree in electrical engineering with a specialization in digital systems and signal processing from the School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad, Pakistan, and the Ph.D. degree in computer engineering from the Vienna University of Technology (TU Wien), Austria. He is currently a Research Group Leader at eBRAIN Lab, New York University Abu Dhabi (NYUAD), United Arab Emirates. His research interests include brain-inspired computing, machine learning, approximate computing, computer architecture, energy-efficient design, robust computing, system-on-chip design, and emerging technologies.



Ayoub Arous is currently a Research Engineer at eBRAIN Lab, New York University Abu Dhabi (NYUAD), United Arab Emirates. His research interests include approximate computing, error analysis, mathematical modeling, and machine learning.

Muhammad Shafique (Senior Member, IEEE) received the Ph.D. degree in computer science from Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, in 2011.

He established and led a highly recognized research group at KIT for several years as well as conducted impactful collaborative R&D activities across the globe. In 2016, he joined as a Full Professor in computer architecture and robust, energy-efficient technologies at the Faculty of Informatics, Institute of Computer Engineering, Technische Universität Wien (TU Wien), Vienna, Austria. Since 2020, he has been with New York University (NYU) Abu Dhabi, Abu Dhabi, UAE, where he is currently a Full Professor and the Director of eBrain Lab. He is a Global Network Professor with Tandon School of Engineering, NYU, New York, NY, USA. He is also a Co-PI/Investigator with multiple NYUAD Centers, including Center of Artificial Intelligence and Robotics (CAIR), Center of Cyber Security (CCS), Center for InTeractIng urban nEtworkS (CITIES), and Center for Quantum and Topological Systems (CQTS). His research interests include AI & machine learning hardware and system-level design, brain-inspired computing, quantum machine learning, cognitive autonomous systems, wearable healthcare, energy-efficient systems, robust computing, hardware security, emerging technologies, FPGAs, MPSoCs, and embedded systems. His research has a special focus on cross-layer analysis, modeling, design, and optimization of computing and memory systems. The researched technologies and tools are deployed in application use cases from Internet-of-Things (IoT), smart cyber–physical systems (CPSs), and ICT for development (ICT4D) domains. He has given several keynotes, invited talks, and tutorials, as well as organized many special sessions at premier venues. He has served as the PC Chair, the General Chair, the Track Chair, and a PC member for several prestigious IEEE/ACM conferences. He holds one U.S. patent and has (co-)authored 6 books, 10+ book chapters, 350+ papers in premier journals and conferences, and 100+ archive articles.

Dr. Shafique received the 2015 ACM/SIGDA Outstanding New Faculty Award, the AI 2000 Chip Technology Most Influential Scholar Award in 2020, 2022, and 2023, the ASPIRE AARE Research Excellence Award in 2021, six gold medals, and several best paper awards and nominations at prestigious conferences. He is a senior member of the IEEE Signal Processing Society (SPS) and a member of the ACM, SIGARCH, SIGDA, SIGBED, and HIPEAC.